การพัฒนาเฟรมเวิร์กของระบบการคำนวณแบบกระจายผ่านทางเว็บเบราว์เซอร์

นายปริน เจียมอนันตพงศ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2555

Development of Browser-based Distributed Computing Framework

Mr. Parin Chiamananthapong

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2012

หัวข้อวิทยานิพนธ์	การพัฒนาเฟรมเวิร์กของระบบการคำนวณแบบกระจายผ่านทางเว็บเบราว์เซอร์
โดย	นายปริน เจียมอนันตพงศ์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร. เกริก ภิรมย์โสภา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

…….. คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

…… ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร. ณัฐวุฒิ หนูไพโรจน์)

…….. อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(ผู้ช่วยศาสตราจารย์ ดร. เกริก ภิรมย์โสภา)

…… กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร. วีระ เหมืองสิน)

…… กรรมการภายนอกมหาวิทยาลัย
(ดร.พงศ์ธวัช ชีพพิมลชัย)

ปริน เจียมอนันตพงศ์ : การพัฒนาเฟรมเวิร์กของระบบการคำนวณแบบกระจายผ่านทางเว็บเบราว์เซอร์. (Development of Browser-based Distributed Computing Framework) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร.เกริก ภิรมย์โสภา, 97 หน้า.

งานวิจัยนี้ได้ทำการศึกษาและพัฒนาระบบการประมวลผลแบบกระจายบนเว็บเบราว์เซอร์ขึ้นมา ทำให้ไคลเอนต์ไม่จำเป็นต้องติดตั้งโปรแกรมเพิ่มเติม และสามารถเข้าร่วมการคำนวณได้โดยง่ายเพียงแค่เปิดเว็บเบราว์สเซอร์ไปยังหน้าที่กำหนด นอกจากนี้ยังได้ประโยชน์ในเรื่องความปลอดภัยจากระบบกระบะทราย และความไม่เข้ากันต่างๆอีกด้วย โดยงานวิจัยนี้ได้มีการนำแนวคิดไปสร้างเป็นเฟรมเวิร์กต้นแบบชื่อ WebGrid.js และทำการประเมินใน 2 รูปแบบคือ การทดสอบผลประสิทธิภาพเบื้องต้นเทียบกับ BOINC และการทดลองนำเฟรมเวิร์กไปทดลองใช้งานจริง โดยผลที่ได้พบว่าเฟรมเวิร์กสามารถนำไปใช้งานได้จริงและให้ผลประสิทธิภาพในระดับที่ยอมรับได้

ภาควิชา .....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่อนิสิต .....
สาขาวิชา .....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก .....
ปีการศึกษา ..2555.....

5470268121 : MAJOR COMPUTER ENGINEERING
KEYWORDS: DISTRIBUTED COMPUTING / WEB TECHNOLOGIES

PARIN CHIAMANANTHAPONG : DEVELOPMENT OF BROWSER-BASED DISTRIBUTED COMPUTING FRAMEWORK. THESIS ADVISOR : ASST.PROF. KRERK PIROMSOPA, Ph.D., 97 pp.

This research, we study and design browser-based distributed computing framework. This platform, client doesn't need to install additional software and can participate easily by browsing a specific web page. In addition, it has several advantages such as secure sandbox environment and platform independence. We implement a prototype framework from this design called WebGrid.js and perform two evaluation; Performance benchmark comparison with BOINC and apply this framework for real environment. The results show that the framework is practically and have acceptable performance.

Department Computer Engineering Student's Signature .....
Field of Study Computer Engineering Advisor's Signature .....
Academic Year 2012

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความอนุเคราะห์อย่างยิ่งของผู้ช่วยศาสตราจารย์ ดร. เกริก ภิรมย์โสภา อาจารย์ที่ปรึกษา ซึ่งท่านได้ให้ความรู้ แนะนำแนวทางการวิจัย ตรวจสอบให้คำแนะนำ และสนับสนุนเป็นอย่างดี จนทำให้การวิจัยในครั้งนี้สำเร็จออกมาด้วยดี

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร. ณัฐวุฒิ หนูไพโรจน์ อาจารย์ ดร. พงศ์ธวัช ชีพพิมลชัย และผู้ช่วยศาสตราจารย์ ดร. วีระ เหมืองสิน กรรมการสอบวิทยานิพนธ์ ที่กรุณาเสียสละเวลา ให้คำแนะนำ ตรวจสอบ และแก้ไขวิทยานิพนธ์ฉบับนี้

ขอขอบพระคุณ คณะวิศวกรรมศาสตร์ และภาควิชาวิศวกรรมคอมพิวเตอร์ ที่ได้มอบทุนอุดหนุนการศึกษาในระดับบัณฑิตศึกษาจุฬาลงกรณ์มหาวิทยาลัย

ขอขอบพระคุณ เพื่อนๆนักศึกษาระดับบัณฑิตศึกษา ที่ได้สละเวลาและทรัพยากรเครื่องคอมพิวเตอร์เข้ามาช่วยทดสอบระบบที่ได้สร้างไว้

ท้ายที่สุด ผู้เสนอวิทยานิพนธ์ขอขอบคุณเพื่อน ๆ ทุก ๆ คน รวมทั้งครอบครัว ที่คอยติดตาม ให้กำลังใจและสนับสนุน รวมถึงท่านอื่น ๆ ที่มิได้กล่าวชื่อไว้ ณ ที่นี้ที่มีส่วนช่วยให้วิทยานิพนธ์สำเร็จได้ด้วยดี

สารบัญ

หน้า

หน้า

สารบัญตาราง

หน้า

สารบัญรูป

หน้า

หน้า

บทที่ 1
บทนำ

ในบทนำนี้จะแบ่งเป็นเจ็ดหัวข้อย่อย กล่าวถึงความเป็นมาและความสำคัญของปัญหา วัตถุประสงค์ ขอบเขตของการวิจัย ประโยชน์ที่คาดว่าจะได้รับ วิธีดำเนินการวิจัย ลำดับขั้นตอนในการเสนอผลการวิจัย และผลงานที่ตีพิมพ์จากวิทยานิพนธ์ ตามลำดับ ดังนี้

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันเทคโนโลยีทางด้านอินเทอร์เน็ตที่ได้มีการพัฒนาให้มีความเร็วในการติดต่อมากขึ้น ประกอบกับเครื่องคอมพิวเตอร์สำหรับใช้งานตามบ้านมีประสิทธิภาพมากขึ้นในราคาที่ถูกลง ทำให้เทคโนโลยีทางด้านการประมวลผลแบบกระจาย (distributed computing) ได้รับความนิยมมากขึ้น ไม่ว่าจะเป็นการประมวลผลแบบคลัสเตอร์ (cluster computing) และการประมวลผลแบบกริด (gird computing) ซึ่งเทคโนโลยีดังกล่าวแม้จะสามารถเพิ่มประสิทธิภาพของการทำงานของคอมพิวเตอร์ในระดับองค์กรได้ แต่ถ้าหากไม่มีงบประมาณที่เพียงพอก็จะไม่สามารถใช้งานได้

การประมวลผลแบบอุทิศ (volunteer computing) เป็นการประมวลผลแบบกระจายรูปแบบหนึ่งที่จะกระจายงานไปให้อาสาสมัคร โดยอาสาสมัครจะใช้ทรัพยากรในเครื่องคอมพิวเตอร์ของตนเองเพื่อช่วยในการทำงานต่างๆที่ได้รับแล้วส่งกลับไป ซึ่งวิธีดังกล่าวเป็นการเก็บรวบรวมทรัพยากรจากเครื่องของอาสาสมัครด้วยความเต็มใจและความยินยอมทั้งสองฝ่าย (หากฝ่ายอาสาสมัครไม่ได้ยินยอม จะเป็นการประมวลผลแบบปรสิต (parasitic computing) แทน) โดยหนึ่งในตัวอย่างของโครงการที่ประสบผลสำเร็จอย่างมากก็คือ SETI@home[1] ซึ่งใช้เฟรมเวิร์ก (framework) BOINC โครงการนี้สามารถเก็บรวบรวมทรัพยากรจากหลายประเทศทั่วโลกเป็นซูเปอร์คอมพิวเตอร์ที่มีความสามารถประมวลผลได้ 551.109 TeraFLOPS (ข้อมูล ณ วันที่ 26 มกราคม 2555) จะเห็นได้ว่าประสิทธิภาพที่ได้ใกล้เคียงกับซูเปอร์คอมพิวเตอร์ที่ติดใน 500 อันดับ [2] ของคอมพิวเตอร์ที่เร็วที่สุดในโลก อย่างไรก็ตามเฟรมเวิร์กนี่ยังมีข้อจำกัดอยู่คือ อาสาสมัครที่จะเข้าร่วมนั้นจำเป็นที่จะต้องติดตั้งโปรแกรมในการใช้งาน ซึ่งอาจจะมีปัญหาในความไม่เข้ากันของระบบเครื่องที่ใช้ หรือระบบปฏิบัติการที่อาสาสมัครใช้งานอยู่

โครงงานวิจัยนี้ได้ทำการศึกษาและพัฒนาระบบการประมวลผลแบบกระจาย โดยใช้เทคโนโลยีทางด้านเว็บแอปพลิเคชั่น (web application) ขึ้นมา ซึ่งเป็นรูปแบบโปรแกรมที่สามารถเข้าถึงได้โดยง่าย เพราะโปรแกรมเว็บเบราว์เซอร์ (web browser) ที่ใช้เข้าอินเทอร์เน็ตนั้นเป็น

โปรแกรมพื้นฐานที่มักติดมาพร้อมกับระบบปฏิบัติการอยู่แล้ว ทำให้อาสาสมัครไม่จำเป็นต้องติดตั้งโปรแกรมลงในเครื่อง และสามารถเข้าร่วมได้โดยง่ายเพียงแค่ใช้โปรแกรมเว็บเบราว์เซอร์เปิดไปตามหน้าเว็บที่ระบุเท่านั้น เนื่องจากการเข้าร่วมของอาสาสมัครที่ง่ายขึ้นทำให้ระบบสามารถขยายตัว (scale out) ได้โดยง่าย และอาสาสมัครจะมีความปลอดภัยมากกว่าเพราะโปรแกรมเว็บเบราว์เซอร์ไม่ได้เข้าถึงทรัพยากรเครื่องโดยตรง ทำให้เวลาโปรแกรมมีปัญหาก็จะหยุดทำงานในเฉพาะส่วนของเว็บเบราว์เซอร์เท่านั้น โดยที่ส่วนของระบบคอมพิวเตอร์ของอาสาสมัครจะไม่ได้รับความเสียหาย นอกจากนี้ เทคโนโลยีทางด้านของเว็บแอปพลิเคชั่นที่มีการพัฒนามากขึ้น เช่น HTML5 และ JavaScript 2.0 ทำให้เว็บแอปพลิเคชั่นมีความสามารถที่ใกล้เคียงกับโปรแกรมคอมพิวเตอร์ทั่วไปมากขึ้น จากเหตุผลที่ได้กล่าวมาข้างต้นจะพบว่าเว็บแอปพลิเคชั่นเป็นตัวเลือกที่มีความเหมาะสมสำหรับนำไปพัฒนาไปเป็นระบบการประมวลผลแบบกระจายได้

1.2 วัตถุประสงค์ของการวิจัย

การวิจัยมีวัตถุประสงค์ ดังนี้

1. เพื่อศึกษาถึงความเป็นไปได้ของระบบการประมวลผลแบบอุทิศโดยใช้เทคโนโลยีเว็บแอปพลิเคชั่น พร้อมทั้งวิเคราะห์เปรียบเทียบกับระบบการประมวลผลแบบอุทิศเดิมที่มีอยู่แล้วในด้านต่างๆ
2. เพื่อทำการออกแบบและพัฒนาระบบการประมวลผลแบบอุทิศโดยใช้เทคโนโลยีเว็บแอปพลิเคชั่น
3. เพื่อทำการทดลองวัดประสิทธิภาพเบื้องต้นโดยการนำระบบที่ได้ออกแบบไว้ไปทำให้เกิดผล(implement)
4. เพื่อเก็บข้อมูลการใช้งานโดยการนำระบบที่สร้างไว้ไปทดลองจริง

1.3 ขอบเขตของการวิจัย

ขอบเขตของการวิจัยถูกกำหนดไว้ ดังนี้

1. ไฟล์อินพุตของระบบเป็นแบบไฟล์ข้อความเท่านั้น ระบบจะไม่รองรับไฟล์อินพุตรูปแบบอื่นๆ เช่น ไฟล์ภาพ, เสียง, สื่อต่างๆ ฯลฯ
2. งานวิจัยนี้จะไม่เน้นพิจารณาถึงวิธีการจูงใจอาสาสมัครให้มาเข้าร่วมโครงการ
3. ระบบนี้จะตั้งอยู่บนสมมติฐานว่าอาสาสมัครและผู้พัฒนาไม่มีเจตนาที่จะทำลายระบบ

4. ในการทดลองวัดประสิทธิภาพด้านการขยายตัวนั้น (scaling) จะทำการทดลองด้วยจำนวนเครื่องสูงสุดเพียง 10 เครื่องเท่านั้น เนื่องจากการทดลองในด้านนี้จำเป็นต้องใช้ทรัพยากรจำนวนมาก จึงไม่ขอเน้นในงานวิจัยนี้
5. ในการทดลองนำระบบมาใช้จริงนั้น สามารถใช้ได้เฉพาะคนที่ใช้อินเทอร์เน็ตภายในจุฬาฯเท่านั้น เนื่องจากเครื่องเซิร์ฟเวอร์ (server) ที่ใช้นั้นใช้อินเทอร์เน็ตของจุฬาซึ่งมีข้อจำกัดคืออนุญาตให้ใช้เฉพาะพอร์ต 80 ในการติดต่อกับอินเทอร์เน็ตภายนอกเท่านั้น
6. ระบบนี้จะไม่รองรับเว็บเบราว์เซอร์ที่ไม่รองรับ HTML5 และ JavaScript 2.0
7. ไคลเอนต์ของระบบจะมีข้อจำกัดทางเว็บแอปพลิเคชั่นดังนี้ คือ ไม่สามารถเก็บข้อมูลที่มีขนาดเกิน 5 MB ได้ (เนื่องจากข้อจำกัดของ localStorage) และไม่สามารถใช้ทรัพยากรของเครื่อง เช่น GPU ได้

1.4 ประโยชน์ที่คาดว่าจะได้รับ

ประโยชน์ที่คาดว่าจะได้รับจากการวิจัย ได้แก่

1. ได้รับความรู้เกี่ยวกับเทคโนโลยีทางด้านเว็บแอปพลิเคชั่น
2. ได้เข้าใจถึงโครงสร้างสถาปัตยกรรมทางด้านการประมวลผลแบบอุทิศ
3. ได้ศึกษาถึงอัลกอริทึมต่างๆ เพื่อหางานที่เหมาะสมที่จะนำมาทำบนระบบประมวลผลแบบอุทิศ
4. ได้ทดลองสร้างระบบจริงทำให้รู้ถึงปัญหาที่เกิดขึ้นตอนใช้งานจริง ซึ่งสามารถนำความรู้เหล่านี้ไปใช้ประยุกต์และแก้ปัญหาในอนาคตได้

1.5 ขั้นตอนและวิธีดำเนินการวิจัย

วิธีดำเนินการวิจัย ถูกแบ่งเป็นห้าขั้นตอน ดังนี้

1. ศึกษาเทคโนโลยีและงานวิจัยที่เกี่ยวข้อง
2. ศึกษาโครงสร้างของระบบการประมวลผลแบบอุทิศ
3. รวบรวมเครื่องมือที่มีอยู่และทำการวัดผลประสิทธิภาพของเครื่องมือต่างๆ รวมไปถึงการวิเคราะห์ลักษณะต่างๆเทียบกับระบบเดิมที่มีอยู่แล้ว (BOINC) เพื่อหาจุดเด่นของแต่ละระบบ
4. สร้างระบบต้นแบบ (prototype) ขึ้นมาและทดลองวัดผลประสิทธิเบื้องต้น
5. นำระบบต้นแบบไปทดลองใช้จริง

6. สรุปผลการวิจัยและจัดทำวิทยานิพนธ์

1.6 ลำดับขั้นตอนในการเสนอผลการวิจัย

วิทยานิพนธ์นี้แบ่งเนื้อหาออกเป็น 7 บท ดังต่อไปนี้ บทที่ 1 เป็นบทนำซึ่งกล่าวถึง ความเป็นมาและความสำคัญของปัญหา รวมถึงวัตถุประสงค์ของการวิจัย บทที่ 2 กล่าวถึงทฤษฎีพื้นฐาน และงานวิจัยที่เกี่ยวข้องกับการวิจัยนี้ บทที่ 3 กล่าวถึงการวิเคราะห์ถึงการนำเว็บแอปพลิเคชั่นไปประยุกต์ เป็นระบบการคำนวณแบบกระจาย บทที่ 4 กล่าวถึงแนวทางการออกแบบระบบ บทที่ 5 กล่าวถึงแนวทางการทำให้เกิดผล (implementation) บทที่ 6 กล่าวถึงวิธีประเมินการวิจัย สรุปและอภิปรายผล และบทที่ 7 เป็นบทสรุปของงานวิจัย

1.7 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้รับการตอบรับให้ตีพิมพ์เป็นบทความทางวิชาการในหัวข้อเรื่อง “An Analysis of Using Web Application as Distributed Computing Platform” โดยนายปริน เจียมอนันตพงศ์ และผู้ช่วยศาสตราจารย์ ดร. เกริก ภิรมย์โสภา, ในงานประชุมวิชาการ “Advances in Computer Science (ACS 2013)” ณ เมือง ภูเก็ต ประเทศไทย วันที่ 10-12 เมษายน พ.ศ. 2556

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงแนวคิดและทฤษฎี รวมทั้งเอกสารและงานวิจัยที่เกี่ยวข้อง ดังนี้

2.1 แนวคิดและทฤษฎี

แนวคิดและทฤษฎีที่จะอธิบายในการวิจัยนี้ แบ่งเป็นสามส่วน ได้แก่

2.1.1 ระบบการประมวลผลแบบกระจาย (Distributed Computing)

ระบบการประมวลผลแบบกระจาย คือระบบคอมพิวเตอร์ที่เกิดจากการเชื่อมต่อเครื่องคอมพิวเตอร์หลายๆเครื่องเข้าด้วยกัน เพื่อช่วยในการประมวลผลหรือทำงานอย่างใดอย่างหนึ่ง ซึ่งแนวคิดนี้ได้มีการประยุกต์ใช้งานในหลายๆด้านอย่างแพร่หลาย โดยในที่นี้จะขอกล่าวถึงคุณสมบัติต่างๆของระบบทั้ง 4 ระบบ ดังนี้

2.1.1.1 การประมวลผลแบบคลัสเตอร์ (Cluster Computing)

การประมวลผลแบบคลัสเตอร์ คือระบบคอมพิวเตอร์ที่เกิดจากกลุ่มของคอมพิวเตอร์ที่อยู่ในพื้นที่เดียวกัน เช่น บนตู้ rack เดียวกัน หรือ ห้องเดียวกัน มาเชื่อมต่อกันเพื่อให้ได้ประสิทธิภาพที่สูงขึ้น โดยทั่วไปเครื่องคอมพิวเตอร์ที่อยู่ในระบบ มักจะเป็นเครื่องคอมพิวเตอร์ที่มีลักษณะเหมือนกัน (Homogeneous)

2.1.1.2 การประมวลผลแบบประมวลผลแบบกริด (Grid Computing)

การประมวลผลแบบกริด เป็นระบบที่เกิดจากการรวบรวมทรัพยากรคอมพิวเตอร์ในระบบขนาดใหญ่ ให้สามารถใช้งานได้เสมือนเครื่องคอมพิวเตอร์เครื่องเดียว เพื่อใช้ในการประมวลผลในงานๆหนึ่ง เช่น การประมวลผลการทดลองทางด้านวิทยาศาสตร์ กริดมักถูกใช้งานในด้านวิทยาศาสตร์และด้านการศึกษา ซึ่งต่างกับคลัสเตอร์ที่มักถูกใช้งานในทางด้านธุรกิจ เนื่องจากกริดเกิดจากการรวบรวมเครื่องคอมพิวเตอร์ต่างๆจากที่ต่างๆ ดังนั้นโดยส่วนใหญ่แล้วเครื่องคอมพิวเตอร์ที่อยู่ในระบบของกริด อาจจะมีลักษณะที่ไม่เหมือนกันได้ (Heterogeneous)

2.1.1.3 การประมวลผลแบบกลุ่มเมฆ (Cloud Computing)

การประมวลผลแบบกลุ่มเมฆ เป็นลักษณะของการทำงานกับระบบเครื่องคอมพิวเตอร์ที่ได้มีผู้ให้บริการผ่านทางระบบอินเทอร์เน็ต โดยผู้ใช้บริการสามารถใช้ทรัพยากรในระบบได้เท่าที่ได้ตกลงไว้กับผู้ให้บริการ ซึ่งการประมวลผลแบบกลุ่มเมฆมักใช้งานในทางด้านธุรกิจ

เพราะสามารถลดต้นทุนทางด้านระบบเครื่องคอมพิวเตอร์เซิร์ฟเวอร์ และผู้ใช้บริการสามารถขยายระบบให้มีขนาดใหญ่ขึ้นหรือเล็กลงได้ตามความต้องการ

2.1.1.4 **การประมวลผลแบบอุทิศ** (Volunteer Computing)

การประมวลผลแบบอุทิศ (หรืออีกชื่อหนึ่งคือ Desktop Grid) เป็นการประมวลผลแบบกระจายรูปแบบหนึ่งที่จะกระจายงานไปให้อาสาสมัคร โดยอาสาสมัครใช้ทรัพยากรในเครื่องคอมพิวเตอร์ของตนเองเพื่อช่วยในการทำงานต่างๆที่ได้รับแล้วส่งกลับไป ซึ่งวิธีดังกล่าวเป็นการเก็บรวบรวมทรัพยากรจากเครื่องของอาสาสมัครด้วยความเต็มใจและความยินยอมทั้งสองฝ่าย กล่าวคือฝ่ายอาสาสมัครเชื่อใจในเจ้าของโครงการว่าโครงการนั้นเชื่อถือได้ มีประโยชน์ต่อส่วนรวม ไม่นำเอาทรัพยากรของอาสาสมัครไปใช้ในด้านที่ผิด ในขณะที่ฝ่ายเจ้าของโครงการนั้นก็เชื่อใจในอาสาสมัครว่าไม่มีเจตนาที่จ้องจะทำลายโครงการนั้นๆ (เช่น การส่งคำตอบที่ผิดไป หรือการเจาะระบบต่างๆ) เป็นต้น

จุดหนึ่งของการประมวลผลแบบอุทิศที่แตกต่างจากระบบการคำนวณแบบอื่นๆ คือ ผู้ที่เข้าร่วมนั้นเป็นคอมพิวเตอร์ที่ใช้งานตามบ้าน ซึ่งมี trustworthy และความ reliable ต่ำกว่าแพลตฟอร์มอื่นๆที่ได้กล่าวมาข้างต้น เพราะอาจมีผู้เข้าร่วมบางคนที่ไม่หวังดี ส่งคำตอบผิดๆมาทำให้เจ้าของโครงการเสียหายได้ นอกจากนี้ ผู้เข้าร่วมสามารถออกจากโครงการได้ตลอดเวลา ดังนั้นจึงต้องมีการสำรองคำตอบหลายๆคำตอบในงานเดียวกัน แล้วตรวจสอบคำตอบโดยใช้การหาเสียงส่วนใหญ่ (Majority voting) เพื่อหาคำตอบที่ถูกต้อง

2.1.1.5 **การสรุปเปรียบเทียบคุณสมบัติต่าง ๆของระบบการคำนวณแบบกระจาย**

จากระบบต่างๆที่ได้กล่าวมา สามารถสรุปลักษณะและคุณสมบัติต่างๆของระบบดังกล่าวได้ด้วยอนุกรมวิธาน (Taxonomy)[3] และข้อมูลจากเว็บไซต์ BOINC ได้เป็นตารางดังนี้

ตารางที่ 1 ตารางเปรียบเทียบคุณสมบัติของระบบประมวลผลแบบกระจาย

Technology / Key Attributes	Cluster Computing	Grid Computing	Cloud Computing	Volunteer Computing
Resource Location	Close proximity	Close/Geographically dispersed	Geographically dispersed	Geographically dispersed
Means of Resource utilization	Harvest idle processor cycles	Harvest idle processor cycles	Run virtual operation systems on the same physical machine	Harvest idle processor cycles
Administrative Entity	Single	Multiple	Single/Multiple	Multiple
Heterogeneity and Multi-tenancy	No	Yes	Yes	Yes
Resource Coupling	Tightly coupled	Loosely coupled	Loosely coupled	Loosely coupled
Virtualization Support	No	Yes	Yes	No
SLA Driven	No	Yes	Yes	Yes
Elasticity	Limited	Limited	Unlimited	Limited
Service Oriented	No	Yes	Yes	Yes
Utility Pricing	No, used for internal purposes	Limited, often open for public use	Yes, billed based on usage	No
Performance	High throughput, Low Latency	High throughput, High Latency	High throughput, Low Latency	High throughput, High Latency
Driving Force and Assistance	Academic, Industry	Academic	Industry	Academic
Robustness	Limited, failed tasks are restarted	Limited, failed tasks are restarted	Strong, easy migration of VMs	Limited, failed tasks are restarted
Application Suitability	Scientific, commercial	Usually HPC and scientific	Commercial, legacy, content delivery	Scientific

2.1.2 เทคโนโลยีทางด้านเว็บแอปพลิเคชั่น

เว็บแอปพลิเคชั่น คือโปรแกรมที่สามารถเข้าถึงได้ด้วยโปรแกรมเว็บเบราว์เซอร์ผ่านทางเครือข่ายอินเทอร์เน็ต ผู้ใช้งานสามารถเข้าถึงโปรแกรมบนเว็บได้ทุกที่ตราบเท่าที่สามารถเชื่อมต่อกับอินเทอร์เน็ตได้ โดยที่ไม่จำเป็นต้องติดตั้งโปรแกรมลงบนเครื่องคอมพิวเตอร์ ทำให้สามารถเข้าถึงโปรแกรมได้โดยไม่ขึ้นกับระบบปฏิบัติการ หรือแพลตฟอร์มที่ใช้

ในปัจจุบัน เว็บแอปพลิเคชั่นได้รับความนิยมและมีจำนวนผู้ใช้งานมากขึ้น ทำให้มีความต้องการใช้งานในโปรแกรมที่มีความซับซ้อนมากขึ้น เช่น การดูวีดีโอบนเว็บ, การแก้ไขรูปต่างๆ, การแสดงผลแบบ 3 มิติ ฯลฯ ซึ่งเทคโนโลยีทางด้านเว็บในปัจจุบันนั้น ยังไม่สามารถทำได้หรือทำได้แต่ต้องใช้ปลั๊กอินช่วย เช่น flash, java applet ดังนั้นจึงได้มีมาตรฐานและเทคโนโลยีใหม่ขึ้นมาซึ่งได้แก่ HTML5, JavaScript 2.0 ทำให้เว็บแอปพลิเคชั่นมีความสามารถที่ใกล้เคียงกับโปรแกรมคอมพิวเตอร์ (Desktop Application) มากขึ้น

2.1.2.1 Asynchronous JavaScript and XML (AJAX) [4]

AJAX เป็นการประยุกต์ใช้เทคโนโลยีต่างๆทางด้านเว็บ เช่น JavaScript, DHTML, XML ฯลฯ มาช่วยในการส่งข้อมูลติดต่อระหว่าง client กับ server แบบ asynchronous ทำให้ผู้ใช้งานไม่จำเป็นต้องโหลดหน้าเว็บใหม่ทั้งหน้าเวลามีการเปลี่ยนแปลงข้อมูลบนหน้าเว็บ ซึ่งรูปแบบนี้ทำให้รูปแบบการใช้งานโปรแกรมบนเว็บมีความต่อเนื่องและหลากหลายมากขึ้น เช่น ระบบ google suggestion ที่ช่วยแนะนำคำค้นเว็บที่ใช้บ่อยขึ้นมา, โปรแกรม word processing หรือ แม้แต่ social network ที่มีการปรับปรุงนำเอาข้อมูลใหม่ๆป้อนบนหน้าเว็บตลอดเวลา เป็นต้น

หลักการทำงานเบื้องต้นของ AJAX คือ ฝั่ง client จะสร้าง XMLHttpRequest (XHR) ในการติดต่อรับ/ส่ง ข้อมูลกับ server อยู่เบื้องหลัง โดยรูปแบบข้อมูลที่ส่งอาจเป็น HTML, XML, JSON, ฯลฯ เมื่อ client ได้รับข้อมูลจาก server แล้ว จะใช้ Document Object Model (DOM) ในการแก้ไขข้อมูลต่างๆบนหน้าเว็บ

2.1.2.2 HTML5

HTML5 คือมาตรฐานใหม่ที่เพิ่มความสามารถให้กับเว็บ โดยในที่นี้จะสรุปเป็น 2 ส่วนคือ ส่วนแรกจะเป็นในส่วนของ markup tag ซึ่งเป็นส่วน tag ที่เพิ่มความสามารถและตรงกับลักษณะของข้อมูลมากขึ้น เช่น <audio> กับ <video> ซึ่งสามารถใส่สื่อวีดีโอหรือเสียงบนเว็บได้โดยตรง โดยที่ไม่ต้องใช้ flash ช่วยเหมือนเมื่อก่อน ทำให้รหัสต้นฉบับ (source code) เป็นระเบียบมากขึ้น และทำให้การจัดหมวดหมู่ข้อมูลง่ายขึ้นอีกด้วย

ในส่วนถัดมาจะเป็นในส่วนของคุณลักษณะใหม่เพิ่มเติม ซึ่งสามารถสรุปข้อมูลเบื้องต้นจาก[5] ได้เป็นตารางดังนี้

ตารางที่ 2 ตารางสรุปคุณสมบัติใหม่ของHTML5

คุณลักษณะ	คำอธิบาย
Canvas element	API สำหรับการแก้ไขรูปภาพ 2 มิติและภาพ bitmap เบื้องต้น
Cross-document messaging	API สำหรับการส่งข้อมูลข้ามเอกสารหรือหน้าเว็บที่ไม่ได้มาจากแหล่งเดียวกัน ซึ่งปกติเราไม่สามารถทำได้ (สามารถดูรายละเอียดเพิ่มเติมได้ในเรื่อง Cross-Origin XMLHttpRequest)
Geolocation	ระบบสำหรับดึงข้อมูลทางด้านภูมิศาสตร์ของเครื่องผู้ใช้ เช่น หาตำแหน่งของผู้ใช้จาก GPS
ContentEditable attribute	คุณสมบัติที่ทำให้ข้อมูลเอกสารนั้นสามารถแก้ไขได้ (เช่น ช่องสำหรับแก้ไขข้อมูลชื่อ)
Drag-and-Drop	เพิ่มคุณสมบัติ Drag-and-Drop
Web Storage	ระบบเก็บข้อมูลแบบ key-value บนเครื่อง client คล้ายกับ cookie แต่สามารถเก็บข้อมูลได้มากกว่าและมีประสิทธิภาพมากขึ้น
Local SQL Database	ระบบฐานข้อมูลบนเครื่อง client สามารถใช้ SQL ในการดำเนินการกับข้อมูลได้
Offline Web Application	ทำให้เว็บสามารถทำงานได้แม้ไม่มีการเชื่อมต่อกับอินเทอร์เน็ต

2.1.2.3 JavaScript

JavaScript คือ ภาษาโปรแกรมหลักของเว็บแอปพลิเคชั่นที่ทำงานบนฝั่งไคลเอนต์ จากข้อมูลของ[6]นั้น คุณสมบัติของ JavaScript สามารถสรุปได้ดังนี้

ความสามารถในการรองรับได้หลายระบบ (portability) JavaScript สามารถใช้งานได้ในหลายระบบ ทำให้ผู้พัฒนา (developer) สามารถที่จะพัฒนาโปรแกรมที่สามารถรองรับได้ในหลายระบบได้ทันที โดยที่ไม่ต้องทำการปรับค่าและแปลโปรแกรม(compile)ใหม่

ความรวดเร็วในการพัฒนา (rapid development) ภาษา JavaScript นั้นเป็นภาษาโปรแกรมประเภทพลวัต (dynamic) ที่สนับสนุนรูปแบบการเขียนโปรแกรมที่หลากหลาย (multi-paradigm) ได้แก่ การเขียนโปรแกรมเชิงวัตถุ (object-oriented programming), การเขียนโปรแกรมเชิงหน้าที่ (functional programming), ฯลฯ เนื่องจากการรองรับรูปแบบที่หลากหลายและการพัฒนาในด้านประสิทธิภาพ (performance) ของภาษา JavaScript ทำให้ภาษา JavaScript มีความเหมาะสมที่จะนำไปพัฒนาโปรแกรมต้นแบบ และถูกนำไปใช้หลากหลายในงานด้านอื่นๆที่ไม่ใช่เว็บแอปพลิเคชั่น ตัวอย่างเช่น Node.js [7] ใช้ภาษา JavaScript เป็นภาษาโปรแกรมในการพัฒนาเซิร์ฟเวอร์, ฐานข้อมูล CouchDB [8] ใช้ภาษา JavaScript เป็นภาษา

สำหรับการเรียกดูข้อมูล และ เครื่องมือสร้างเกม Unity3D [9] ใช้ภาษา JavaScript ในการเขียนโปรแกรมเกม

ความปลอดภัย (security) เนื่องจากเว็บแอปพลิเคชั่นสามารถเริ่มทำงานได้ทันทีที่เริ่มเปิดเว็บ ทำให้ประเด็นเรื่องความปลอดภัยเป้นเรื่องสำคัญ ภาษา JavaScript นั้นจะมีการใช้งานกระบะทราย (sandbox) เพื่อป้องกันความเสียหายที่อาจจะเกิดขึ้นกับระบบภายนอกได้

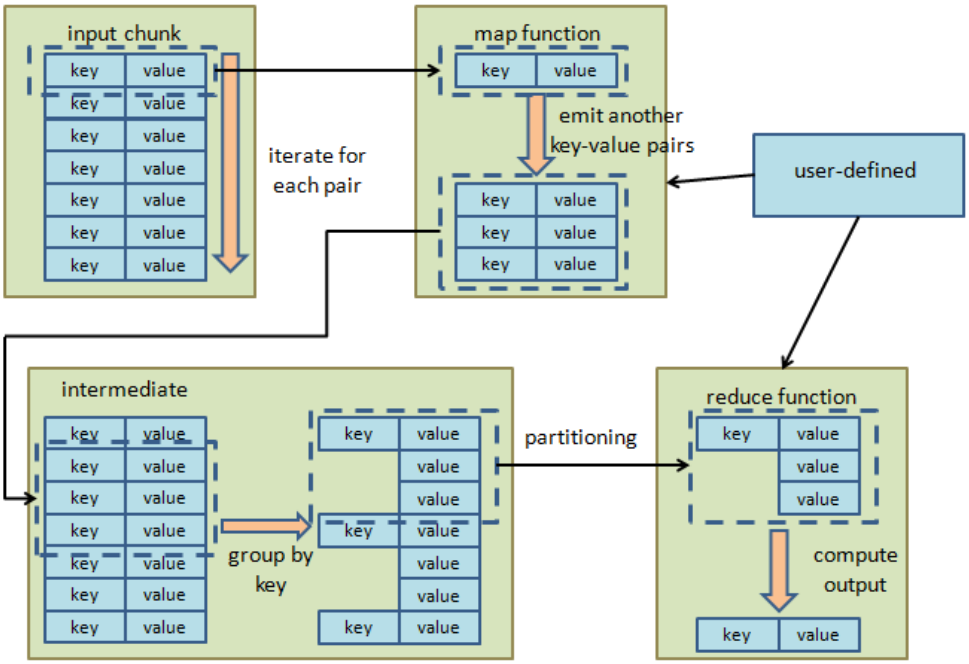
2.1.2.4 Web worker

เนื่องจากโครงสร้างของภาษา JavaScript เป็นการทำงานแบบเทรดเดียว (single thread) จึงทำให้เมื่อมีการประมวลผลบางอย่าง อาจจะทำให้การแสดงผลออกมาทางหน้าเว็บติดขัดได้

Web worker[10] เป็นส่วนหนึ่งของมาตรฐานใหม่ของ JavaScript ที่ถูกคิดค้นขึ้นมาเพื่อแก้ปัญหาดังกล่าว โดยหลักการของ Web worker จะเป็นการแยกการทำงานในส่วนของโปรเซส และส่วนแสดงผลออกจากกัน โดยในส่วนของโปรเซสนั้นจะถูกส่งให้เทรดเบื้องหลัง (background thread) จัดการ ทำให้การทำงานของส่วนประมวลผลไม่รบกวนการทำงานในส่วนของการแสดงผล โดย Web worker จะใช้การส่งข้อความ (message passing) ในการติดต่อและแลกเปลี่ยนข้อมูลกับส่วนอื่นๆ

2.1.3 MapReduce

MapReduce เป็นรูปแบบการเขียนโปรแกรมรูปแบบหนึ่ง ที่ใช้ในการประมวลผลกับข้อมูลที่มีขนาดใหญ่ โดยรูปแบบนี้จะช่วยลดความซับซ้อนของการเขียนโปรแกรมแบบขนาน เช่น การแบ่งเทรด, การถ่ายโอนข้อมูลไปยังเครื่องอื่นๆ, การแบ่งทรัพยากรต่างๆ ฯลฯ ให้เหลือเพียงแค่สองฟังก์ชันเท่านั้น คือ map และ reduce [11] ทำให้ผู้พัฒนาที่ไม่เคยมีประสบการณ์ในการเขียนโปรแกรมแบบขนานมาก่อน สามารถพัฒนาโปรแกรมได้โดยง่าย โดยรูปแบบแนวคิดการทำงานของ MapReduce สามารถสรุปเป็นภาพได้ดังนี้



รูปที่ 1 ขั้นตอนการทำงานของ MapReduce

1. ไฟล์อินพุตถูกแบ่งออกเป็นส่วนย่อยๆ (chunk) โดย chunk แต่ละตัวจะถูกส่งไปให้ mapper เพื่อจัดการต่อไป
2. mapper ทำการอ่านค่า key-value pair แต่ละคู่ แล้วทำการสร้างข้อมูล key-value pair ใหม่ตามที่ผู้ใช้งานได้กำหนดไว้
3. ข้อมูลที่ได้จาก mapper ถูกเรียกว่า intermediate และจะถูกส่งไปรวมกับข้อมูล intermediate อื่นๆ โดยข้อมูลที่มี key เหมือนกัน จะถูกรวมเข้าด้วยกัน
4. ข้อมูลที่ได้รวมกันแล้ว ถูกแบ่งพาร์ทิชัน (partitioning) ไปให้ reducer แต่ละตัว
5. reducer แต่ละตัวจะทำการประมวลผลเพื่อหาคำตอบสุดท้ายออกมา

2.2 โครงงานและงานวิจัยที่เกี่ยวข้อง

แบ่งออกเป็น 2 ประเภทคือ งานวิจัยที่เกี่ยวข้องกับสถาปัตยกรรมของระบบประมวลผลแบบอุทิศ และงานวิจัยที่เกี่ยวข้องกับการนำเทคโนโลยีเว็บแอปพลิเคชั่นมาประยุกต์ใช้ทางด้านระบบการประมวลผลแบบกระจาย

2.2.1 งานวิจัยที่เกี่ยวข้องกับสถาปัตยกรรมของระบบการประมวลผลแบบอุทิศ

งานวิจัยที่เกี่ยวข้องกับสถาปัตยกรรมของระบบการประมวลผลแบบอุทิศ ส่วนใหญ่จะเป็นงานวิจัยที่เกี่ยวข้องกับเฟรมเวิร์ก BOINC ซึ่งงานวิจัยที่น่าสนใจมีดังนี้

2.2.1.1 BOINC: A System for Public-resource Computing and Storage [12]

BOINC เป็นโปรแกรมสำหรับสร้างระบบการประมวลผลแบบอุทิศ ระบบนี้ใช้การประมวลผลจากเครื่องคอมพิวเตอร์ตั้งโต๊ะที่อยู่ตามบ้านทั่วไป โดยในงานวิจัยนี้จะกล่าวถึงความสามารถของเฟรมเวิร์ก และจุดที่แตกต่างของระบบนี้กับระบบการประมวลผลแบบกริดทั่วไป รวมไปถึงวิธีการแก้ไขปัญหาดังกล่าว ตัวอย่างเช่น ปัญหาที่ไคลเอนต์อาจจะส่งคำตอบที่ผิดพลาดมาได้ ซึ่งสามารถแก้ได้โดยการส่งงานเดียวกันไปให้หลายไคลเอนต์ แล้วตรวจสอบคำตอบเทียบกับคำตอบที่ได้จากไคลเอนต์อื่นๆ, ปัญหาการที่ไคลเอนต์สามารถออกจากการประมวลผลได้ตลอดเวลา ซึ่งปัญหานี้สามารถแก้ได้โดยการทำการคอยสร้างจุดตรวจสอบ (checkpoint) เป็นระยะในช่วงเวลาที่กำหนด และปัญหาการที่ไคลเอนต์ส่วนใหญ่อยู่หลังไฟร์วอลล์ หรือ NAT ทำให้เซิร์ฟเวอร์ไม่สามารถที่จะติดต่อกับไคลเอนต์โดยตรงได้ นอกจากนี้ ยังมีการใช้ระบบคะแนนในการจูงใจให้คนมาเข้าร่วมอีกด้วย

2.2.1.2 High-performance Task Distribution for Volunteer Computing [13]

งานวิจัยนี้จะเป็นการอธิบายถึงหลักการออกแบบสถาปัตยกรรมของระบบการประมวลผลแบบอุทิศ โดยงานวิจัยนี้เสนอว่าควรออกแบบโดยการแยกการทำงานออกเป็นโมดูล (module) ต่างๆ และติดต่อสื่อสารระหว่างโมดูลผ่านทางฐานข้อมูลส่วนกลาง โดยโมดูลดึงข้อมูลงานที่เกี่ยวข้องกับตนเองขึ้นมา เมื่อประมวลผลเสร็จแล้วก็จะแก้ตัวบ่งชี้ (flag) ของงานนั้น เพื่อให้โมดูลที่เกี่ยวข้องจัดการต่อไป ซึ่งสถาปัตยกรรมแบบนี้ แม้ว่าอาจจะมีข้อเสียตรงที่เพิ่มภาระให้กับฐานข้อมูลมากขึ้น แต่ก็มีข้อดีตรงที่ โมดูลแต่ละตัวเป็นอิสระต่อกัน ทำให้เมื่อเกิดการความล่าช้า (delay) หรือ เกิดความเสียหายขึ้นมา ก็ไม่ส่งผลต่อระบบโดยรวมและโมดูลตัวอื่นๆก็ยังสามารถทำงานได้ตามปกติ

2.2.2 งานวิจัยที่เกี่ยวข้องกับการนำเทคโนโลยีเว็บแอปพลิเคชั่นมาประยุกต์ใช้ทางด้านระบบการประมวลผลแบบกระจาย

งานวิจัยที่เกี่ยวข้องกับการนำเทคโนโลยีเว็บแอปพลิเคชั่นมาประยุกต์ใช้ทางด้านระบบการประมวลผลแบบกระจายนั้น สามารถแบ่งได้เป็น 3 ประเภทตามวิธีการดำเนินการ (implement) คือ งานวิจัยที่ใช้เทคโนโลยีปลั๊กอินในการดำเนินการ, งานวิจัยที่ใช้เทคโนโลยีทางด้านเว็บแอปพลิเคชั่นในการดำเนินการ และงานวิจัยที่ใช้เทคโนโลยีทางด้านเว็บแอปพลิเคชั่นสมัยใหม่ในการดำเนินการ

2.2.2.1 งานวิจัยที่ใช้เทคโนโลยีปลั๊กอินในการดำเนินการ

งานวิจัยกลุ่มนี้มักพบในช่วงปี 1989 – 2006 เนื่องจากข้อจำกัดทางด้านเว็บแอปพลิเคชั่น ตัวอย่างเช่น การส่งข้อมูลไปยังเซิร์ฟเวอร์โดยที่ผู้ใช้ไม่ต้องกดเอง (สมัยก่อนที่เว็บยังไม่สนับสนุน AJAX) หรือต้องการเพิ่มความสามารถอื่นๆที่เว็บแอปพลิเคชั่นไม่มี ทำให้ต้องนำเทคโนโลยีปลั๊กอินเข้ามาช่วยลดความจำกัดเหล่านั้น โดยงานวิจัยที่น่าสนใจมีดังนี้

2.2.2.1.1 Bayanihan: Building and Studying Web-based Volunteer Computing Systems Using Java [14]

งานวิจัยนี้ได้ถูกนำเสนอในปี 1989 ซึ่งเป็นงานวิจัยแรกที่มีการนำเทคโนโลยีทางด้านเว็บแอปพลิเคชั่นมาประยุกต์ใช้ในด้านการประมวลผลแบบกระจาย โดยในงานวิจัยนี้จะออกแบบระบบทั้งฝั่งเซิร์ฟเวอร์และฝั่งไคลเอนต์ สำหรับฝั่งไคลเอนต์ของระบบนี้สร้างโดยการนำโปรแกรม JAVA Applet ฝังลงบนหน้าเว็บในการใช้งาน

2.2.2.1.2 Rendering Animations with Distributed Applets [15]

งานวิจัยนี้ได้ถูกนำเสนอในปี 2009 โดยงานวิจัยนี้เป็นการนำเทคโนโลยีทางด้านเว็บแอปพลิเคชั่นมาช่วยในการประมวลผลทางด้านคอมพิวเตอร์กราฟิก โดยการให้แต่ละไคลเอนต์ที่ลง JAVA Applet และ Blender Plugin มาช่วยกันเรนเดอร์ภาพที่เซิร์ฟเวอร์แจกจ่ายให้ สุดท้ายภาพนิ่งที่เรนเดอร์เสร็จแล้วจะถูกนำไปรวมที่เซิร์ฟเวอร์เพื่อทำภาพเคลื่อนไหว (animation) ต่อไป ซึ่งงานวิจัยนี้ได้มีการต่อยอด[16] โดยการนำเครือข่ายสังคมออนไลน์ (social network) มาช่วยในการหาอาสาสมัครด้วย

2.2.2.2 งานวิจัยที่ใช้เทคโนโลยีเว็บแอปพลิเคชั่นมาใช้ในการดำเนินการ

งานวิจัยกลุ่มนี้มักพบตั้งแต่ปี 2007 ขึ้นไป เนื่องจากเทคโนโลยี AJAX ที่ทำให้เว็บแอปพลิเคชั่นสามารถทำงานได้อย่างต่อเนื่องมากขึ้น งานวิจัยกลุ่มนี้มักเน้นการนำงานประยุกต์ (application) ที่มีอยู่แล้วไปประยุกต์ใช้งานกับเว็บแอปพลิเคชั่น โดยหลักการของงานวิจัยกลุ่มนี้ใช้ภาษา JavaScript ในการพัฒนาการทำงานฝั่งไคลเอนต์และใช้ AJAX เป็นช่องทางในการติดต่อระหว่างไคลเอนต์กับเซิร์ฟเวอร์ โดยผลที่ได้สามารถสรุปได้ว่า เทคโนโลยีเว็บแอปพลิเคชั่นสามารถนำมาประยุกต์ใช้ทางด้านระบบการประมวลผลแบบกระจายได้ แต่อาจติดข้อจำกัดของเว็บแอปพลิเคชั่นบางประการ โดยงานวิจัยที่น่าสนใจมีดังนี้

2.2.2.2.1 Browser-based distributed evolutionary computation: performance and scaling behavior [17]

งานวิจัยนี้ได้ถูกนำเสนอในปี 2007 โดยงานวิจัยนี้จะเป็นการนำโปรแกรมการคำนวณวิวัฒนาการ (evolutionary computation) มาประยุกต์ใช้กับเว็บแอปพลิเคชั่น พร้อมทั้งวิเคราะห์ปัจจัยที่ส่งผลต่อประสิทธิภาพ ซึ่งผลสรุปที่ได้คือ ประสิทธิภาพที่ได้จากฝั่งไคลเอนต์ขึ้นกับโปรแกรมเว็บเบราว์เซอร์ที่ใช้เป็นหลัก และการแบ่งจำนวนงานเป็นกลุ่มข้อมูล (packet) ขนาดเล็กนั้นทำให้ประสิทธิภาพลดลงเพราะเสียความสิ้นเปลือง (overhead) ในการแบ่งกลุ่มข้อมูลและเสียเวลาในการขนส่งข้อมูลที่มีจำนวนครั้งการส่งมากขึ้น นอกจากนี้ยังมีงานวิจัยที่คล้ายกับงานนี้ได้แก่ [18,19]

2.2.2.2.2 Distributed Computing Through Web Browser [20]

งานวิจัยนี้ได้ถูกนำเสนอในปี 2007 งานวิจัยนี้ได้มีการใช้ฟังก์ชัน setTimeout ในการแก้ปัญหาการค้างของเว็บเบราว์เซอร์ เนื่องจาก JavaScript มีเทรดเดียวซึ่งใช้งานร่วมกันทั้งในส่วนการประมวลผลและการแสดงผลออกมาทางหน้าเว็บ ดังนั้นหากมีการใช้งานในส่วนประมวลผลมากเกินไป อาจส่งผลให้การแสดงผลค้างหรือตอบสนองช้าได้ นอกจากนี้งานวิจัยนี้ยังได้นำเสนอเมตริกในการวัด โดยการหาสัดส่วนเวลาระหว่างการคำนวณกับการส่งข้อมูล โดยงานที่เหมาะสมกับระบบที่เสนอมานี้คือ งานที่ใช้เวลาในการคำนวณเยอะและไม่ต้องส่งหรือถ่ายโอนข้อมูลมาก

2.2.2.2.3 RABC: A Conceptual Design of Pervasive Infrastructure for Browser Computing based on Ajax technologies [21]

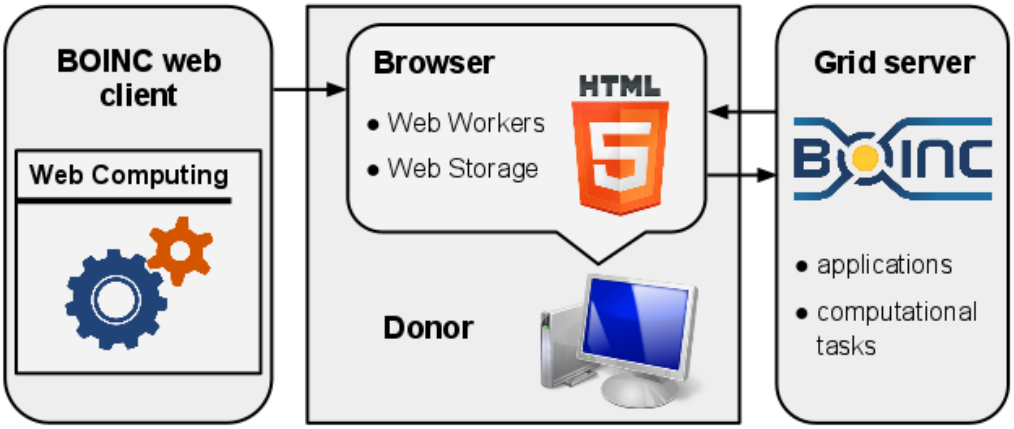
งานวิจัยที่ถูกพัฒนาขึ้นในปี 2007 โดยงานวิจัยนี้เป็นการเสนอระบบชื่อว่า RIKEN Ajax Browser Computation (RBAC) ขึ้นมา โดยระบบนี้ได้นำแนวคิดของระบบจัดการเนื้อหา (Content management system, CMS) เข้ามาใช้ในการจัดการรหัสโปรแกรมของผู้พัฒนาได้ ทำให้ระบบนี้สามารถมีผู้พัฒนาหลายคนได้ ผู้พัฒนาสามารถอัพโหลดรหัสโปรแกรมเพื่อให้ผู้ใช้งานคนอื่นช่วยทำงานได้ นอกจากนี้งานวิจัยนี้ยังมีการวัดผลประสิทธิภาพของภาษา JavaScript เทียบกับภาษาคอมพิวเตอร์อื่นๆ พบว่าประสิทธิภาพของ JavaScript ในช่วงนั้นด้อยกว่าภาษาคอมพิวเตอร์อื่นๆพอสมควร

2.2.2.3 งานวิจัยที่ใช้เทคโนโลยีเว็บแอปพลิเคชั่นสมัยใหม่มาใช้ในการดำเนินการ

งานวิจัยกลุ่มนี้มักพบตั้งแต่ปี 2011 ขึ้นไป เนื่องจากเป็นช่วงที่เทคโนโลยีทางด้านเว็บแอปพลิเคชั่นสมัยใหม่เข้ามา ได้แก่ HTML5, JavaScript 2.0 โดยงานวิจัยกลุ่มนี้มักเน้นการนำงานประยุกต์ที่มีอยู่แล้วไปประยุกต์ใช้งานกับเว็บแอปพลิเคชั่นสมัยใหม่เข้าช่วย โดยหลักการของงานวิจัยกลุ่มนี้คล้ายกับงานวิจัยกลุ่มที่แล้วแต่มีส่วนแตกต่างตรงที่งานวิจัยกลุ่มนี้จะมีการนำเทคโนโลยีใหม่เข้ามาช่วยลดข้อจำกัดของเว็บแอปพลิเคชั่นที่มีอยู่เดิม โดยงานวิจัยที่น่าสนใจมีดังนี้

2.2.2.3.1 GridBee [22]

โครงงานนี้ได้ถูกนำเสนอในช่วงปลายปี 2011 โดยงานนี้เป็นการทำโปรแกรมไคลเอนต์เวอร์ชั่นเว็บแอปพลิเคชั่นของ BOINC เป็นหลัก ซึ่งโครงสร้างของระบบจะเป็นดังรูป



รูปที่ 2 โครงสร้างของระบบ GridBee

โดยระบบนี้จะมีการนำ Web worker เข้ามาใช้ในการแก้ปัญหาเทรดเดียวของ JavaScript และมีการนำ localStorage มาเก็บข้อมูลจุดตรวจสอบของการคำนวณ

2.2.2.3.2 On-Demand Web Search Using Browser-Based Volunteer Computing [23]

งานวิจัยนี้ได้ถูกนำเสนอในปี 2012 โดยงานวิจัยนี้นำเสนอระบบการค้นหาเว็บที่มีการนำหลักการของการประมวลผลแบบอุทิศเข้ามาช่วย โดยระบบจะส่งไฟล์ข้อมูลหน้าเว็บไปให้ไคลเอนต์ช่วยประมวลผลในการทำดัชนี (index) บนโปรแกรมเว็บเบราว์เซอร์ งานวิจัยนี้ได้นำเสนอถึงสถาปัตยกรรมของระบบ แต่ไม่ได้มีการนำเสนอในด้านการวัดประสิทธิภาพของระบบ

2.2.2.3.3 Distributed Evolutionary Computing System Based on Web Browsers with JavaScript [24]

งานวิจัยนี้ได้ถูกนำเสนอในปี 2013 โดยงานวิจัยนี้นำเว็บแอปพลิเคชั่นมาประยุกต์ใช้ในงานด้านการคำนวณแบบวิวัฒนาการ โดยระบบที่ได้นำเสนอในงานวิจัยนี้ ได้ใช้ภาษา JavaScript พัฒนาทั้งฝั่งไคลเอนต์และฝั่งเซิร์ฟเวอร์ โดยฝั่งเซิร์ฟเวอร์นั้นได้ใช้ Node.js ในการพัฒนา งานวิจัยนี้มีการทดลองวัดประสิทธิภาพของระบบโดยการนำระบบมาใช้จริงในปัญหา flowshop scheduling

บทที่ 3
การวิเคราะห์ถึงการนำเว็บแอปพลิเคชั่นไปประยุกต์เป็นระบบการคำนวณแบบกระจาย

ในบทนี้ได้ถูกแบ่งออกเป็น 2 ส่วนคือ การวิเคราะห์ถึงความสามารถในการประมวลผลบนเว็บเบราว์เซอร์ และการเปรียบเทียบคุณสมบัติของการนำเว็บแอปพลิเคชั่นไปประยุกต์เป็นระบบการคำนวณแบบกระจาย (ซึ่งภายหลังขอเรียกว่า ระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์ (browser-based distributed computing)) เทียบกับระบบการคำนวณแบบกระจายที่มีอยู่แล้ว ในส่วนแรกจะเป็นการวิเคราะห์ถึงความสามารถของเว็บแอปพลิเคชั่นโดยการ วัดผลประสิทธิภาพของภาษา JavaScript ที่เป็นภาษาหลักที่ถูกใช้งานบนเว็บเบราว์เซอร์เทียบกับภาษาคอมพิวเตอร์อื่นๆ และวัดผลเทคโนโลยีรวมไปถึงเครื่องมืออื่นๆบนเว็บแอปพลิเคชั่นที่ถูกออกแบบมาเพื่อเพิ่มประสิทธิภาพบนเว็บแอปพลิเคชั่นเทียบกับประสิทธิภาพของภาษาที่ทำงานบนเครื่องเช่น C++ ในส่วนที่สองจะเป็นการเปรียบเทียบคุณสมบัติในด้านต่างๆของระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์เทียบกับเฟรมเวิร์ก BOINC

3.1 การวิเคราะห์ถึงความสามารถในการประมวลผลของเว็บเบราว์เซอร์

ในส่วนนี้เป็นการวิเคราะห์ถึงความสามารถในการประมวลผลบนเว็บเบราว์เซอร์ ซึ่งเนื้อหาจะถูกแบ่งออกเป็น 3 ส่วนได้แก่ ส่วนแรกจะเป็นการทดลองวัดผลประสิทธิภาพของภาษา JavaScript เทียบกับภาษาคอมพิวเตอร์อื่นๆ ส่วนที่สองจะเป็นการรวบรวมเครื่องมือต่างๆสำหรับเพิ่มประสิทธิภาพบนเว็บเบราว์เซอร์ และในส่วนสุดท้ายจะเป็นการทดลองเพื่อวัดผลประสิทธิภาพของเครื่องมือเหล่านั้น

3.1.1 การทดลองวัดผลประสิทธิภาพของภาษา JavaScript เทียบกับภาษาคอมพิวเตอร์อื่น ๆ

ภาษา JavaScript เป็นภาษาหลักบนฝั่งไคลเอนต์สำหรับเว็บแอปพลิเคชั่น โดยการใช้งานภาษา JavaScript ในอดีตนั้นมักเป็นการใช้งานเพื่อตรวจสอบค่าอินพุตของฟอร์มข้อมูล หรือการเล่นภาพเคลื่อนไหวเบื้องต้น ซึ่งจากผลการทดลองวัดประสิทธิภาพของ JavaScript ในงานวิจัย [21] นั้นให้ผลว่าภาษา JavaScript มีประสิทธิภาพที่ด้อยกว่าภาษาคอมพิวเตอร์อื่นๆ อย่างไรก็ตามปัจจุบันเทคโนโลยีของภาษา JavaScript นั้นก้าวหน้าขึ้น ตัวประมวลผลภาษา JavaScript (JavaScript engine) ใหม่ๆเช่น V8[25], SpiderMonkey[26] มีประสิทธิภาพที่ดีขึ้นมาก ส่งผลให้

ภาษา JavaScript ได้รับความนิยมมากขึ้น และเริ่มมีการย้ายการประมวลผลบางส่วนจากฝั่งเซิร์ฟเวอร์ไปยังฝั่งไคลเอนต์แทน ซึ่งสามารถสังเกตุได้จากการเติบโตของชุมชนของผู้ใช้ภาษา JavaScript จากเว็บไซท์ Github[27] ที่มีโปรแกรมที่สร้างจากภาษา JavaScript เป็นจำนวนมาก และมีคลังคำสั่ง (library) ที่มีการใช้งานภาษา JavaScript ในงานด้านอื่นๆมากขึ้น ตัวอย่างเช่น การแสดงผลข้อมูล (data virtualization), การจำลองโมเดลทางฟิสิกส์ (physic simulation), การประมวลผลทางด้านรูปภาพ (image processing), ฯลฯ

เนื่องจากสภาพแวดล้อมของภาษา JavaScript มีการเปลี่ยนไปจากสภาพแวดล้อมของภาษา JavaScript ที่มีการทดลองใน [21]ค่อนข้างมาก ในงานวิจัยนี้จึงมีการทดลองเพื่อวัดผลประสิทธิภาพของ JavaScript เทียบกับภาษาคอมพิวเตอร์อื่นๆขึ้นมาใหม่ เพื่อให้ผลการทดลองที่ได้มีความทันสมัยมากขึ้น

3.1.1.1 GigaFLOP count

การทดลองนี้จะเป็นการทดลองเพื่อวัดผลประสิทธิภาพของภาษา JavaScript เทียบกับภาษาคอมพิวเตอร์อื่นๆ นอกจากนี้ยังเป็นการเทียบประสิทธิภาพของตัวประมวลภาษา JavaScript ของแต่ะเว็บเบราว์เซอร์อีกด้วย โดยการทดลองจะเป็นการวัดเวลาที่ใช้ในการประมวลผลคำสั่งที่เกี่ยวข้องเลขจำนวนจริง (floating point) 1พันล้านคำสั่งของแต่ละภาษาคอมพิวเตอร์ ซึ่งสามารถอธิบายเป็นโค้ดของโปรแกรมได้ดังนี้

```
double giga_flop(int foo) {
  double x = 3.14159*foo;
  int i,j;
  for (i=0; i<500000000; i++) {
    x += 5.12313123;
    x *= 0.5398394834;
  }
  return x;
}
```

รายละเอียดของเครื่องคอมพิวเตอร์ที่ใช้ทดสอบมีดังนี้

- CPU: Intel Core i7-3770 Processor (3.4 GHz)
- หน่วยความจำ 16 GB
- ระบบปฏิบัติการ Windows 7 service pack 1 64 bit version

รายละเอียดของภาษาคอมพิวเตอร์ที่นำมาใช้ทดสอบมีดังนี้

- C++ (ใช้ Microsoft Visual Studio 2010 Express และค่าพารามิเตอร์ –O2 ในการแปลรหัสโปรแกรม)

- JAVA 1.7.0
- Python 3.3.0
- Google Chrome 23 (JavaScript)
- Mozilla Firefox 17 (JavaScript)
- Microsoft Internet Explorer 9.0 (JavaScript)

การทดลองจะวัดผลเวลาที่ใช้ทั้งหมด 5 รอบในแต่ละภาษาคอมพิวเตอร์ จากนั้นจึงตัดผลเวลาที่มากสุดและน้อยสุดออก และใช้ค่าเฉลี่ยของผลเวลาที่เหลือ โดยผลการทดลองที่ได้สามารถแสดงในตารางที่ 3

ตารางที่ 3 เวลาที่ใช้ในการประมวลผลคำสั่ง floating point 1 พันล้านคำสั่งของแต่ละภาษาคอมพิวเตอร์

programming language	execution time (ms)	GFLOPS	ratio (comparing to JAVA)
JAVA	1061	0.9429	1.000
C++	1064	0.9398	1.003
Python	58847	0.0170	55.49
JavaScript Chrome	1106	0.9042	1.043
JavaScript Firefox	1351	0.7404	1.273
JavaScript IE	4654	0.2149	4.388

จากผลการทดลอง เราสามารถสรุปได้ว่า ประสิทธิภาพของภาษา JavaScript ที่ทำงานบน Google Chrome และ Mozilla Firefox นั้นไม่แตกต่างจากภาษา C++ และ JAVA มาก โดย JavaScript ที่ทำงานบน Microsoft Internet Explorer นั้นจะใช้เวลามากกว่าโปรแกรมที่เขียนด้วย JAVA ประมาณ 4 เท่า และ Python จะใช้เวลามากกว่า 55 เท่า ตามลำดับ ซึ่งจากการทดลองนี้สามารถสรุปได้ว่า ภาษา JavaScript นั้นมีประสิทธิภาพพอที่จะสามารถนำมาทำระบบการคำนวณแบบกระจายได้

นอกจากนี้ การทดลองนี้มีจุดที่น่าสนใจอยู่ 2 จุดได้แก่ จุดแรกภาษา JAVA มีประสิทธิภาพสูงกว่า C++ ในการทดลองนี้ ซึ่งตรงจุดนี้สามารถอธิบายได้ว่า ภาษา JAVA นั้นมีการใช้งานในส่วนของ JIT (just-in-time compilation) [28] ซึ่งสามารถทำให้ตัวภาษาสามารถเพิ่มประสิทธิภาพในขณะที่โปรแกรมกำลังทำงานจากสภาพแวดล้อมของเครื่องคอมพิวเตอร์นั้นๆได้ (เช่น เครื่องคอมพิวเตอร์ที่มีหลายหน่วยประมวลผล) จุดที่สองคือประสิทธิภาพของภาษา Python

ที่ค่อนข้างต่ำเมื่อเทียบกับภาษาคอมพิวเตอร์อื่นๆ ซึ่งจุดนี้สามารถอธิบายได้ว่า ภาษา Python นั้นเป็นภาษาที่ไม่เน้นประสิทธิภาพแต่เน้นที่ประโยชน์การใช้งาน (functionality) และความเป็นประโยชน์ (productivity) แทน

3.1.2 เครื่องมือต่าง ๆสำหรับเพิ่มประสิทธิภาพบนเว็บเบราว์เซอร์

ในส่วนนี้จะเป็นการรวบรวมเครื่องมือต่างๆสำหรับเพิ่มประสิทธิภาพบนเว็บเบราว์เซอร์ โดยในงานวิจัยนี้จะแบ่งเครื่องมือดังกล่าวออกเป็น 2 กลุ่ม โดยกลุ่มแรกจะเป็นกลุ่มเครื่องมือที่มีอยู่บนเว็บเบราว์เซอร์อยู่แล้ว ส่วนอีกกลุ่มจะเป็นกลุ่มเครื่องมือที่มีเฉพาะบางเว็บเบราว์เซอร์เท่านั้น

3.1.2.1 กลุ่มเครื่องมือที่มีอยู่บนเว็บเบราว์เซอร์อยู่แล้ว

กลุ่มเครื่องมือเหล่านี้มักเป็นเครื่องมือหรือมาตรฐานบน JavaScript โดยเครื่องมือในกลุ่มนี้ ได้แก่

3.1.2.1.1 Typed array [29]

เป็นโครงสร้างข้อมูล (data structure) ที่ปรับปรุงให้คล้ายกับโครงสร้างข้อมูลอาร์เรย์ของภาษา C++ โดยปกติอาร์เรย์ของ JavaScript นั้นสามารถเก็บข้อมูลได้ทุกชนิด เช่น โครงสร้างวัตถุ (object), ตัวแปรพื้นฐานต่างๆ รวมไปถึง อาร์เรย์ของข้อมูลอื่นๆ ซึ่งโครงสร้างข้อมูลอาร์เรย์ประเภทนี้ได้เพิ่มความสิ้นเปลือง (overhead) ในการเก็บข้อมูลแต่ละตัวเพื่อแลกกับความยืดหยุ่นในการเก็บข้อมูลหลากหลายประเภท ดังนั้น Typed array จึงเป็นโครงสร้างอาร์เรย์สำหรับเก็บข้อมูลประเภทเดียวกันของภาษา JavaScript ที่ได้ตัดความสิ้นเปลืองเหล่านั้นลง และข้อมูลที่ถูกจัดเก็บด้วย Typed array นั้น สามารถเข้าถึงได้โดยตรงจากหน่วยความจำโดยที่ไม่ต้องไปค้นหาหรือแปลงรูปแบบข้อมูลก่อน

3.1.2.1.2 Web worker [10]

เป็นโครงสร้างเทรดสำหรับเว็บแอปพลิเคชั่น การใช้งาน web worker สามารถแยกการทำงานของโปรเซสออกจากส่วนแสดงผลได้ ดังนั้นจึงสามารถนำไปประยุกต์ใช้ในการเพิ่มประสิทธิภาพการประมวลผลได้โดยการแบ่งงานให้แต่ละเทรด worker ทำ โดย web worker จะสามารถติดต่อกับเทรดหลักได้โดยการส่งข้อความ (message passing) แต่จะไม่สามารถเข้าถึงข้อมูลของเทรดหลัก และไม่สามารถสามารถติดต่อกับเทรด worker อื่นๆได้

3.1.2.2 กลุ่มเครื่องมือที่มีอยู่เฉพาะบางเว็บเบราว์เซอร์

เครื่องมือในกลุ่มนี้สามารถเพิ่มประสิทธิภาพได้มากกว่าเครื่องมือในกลุ่มที่แล้ว แต่สามารถใช้ได้เฉพาะบางเว็บเบราว์เซอร์เท่านั้น โดยเครื่องมือในกลุ่มนี้ได้แก่

3.1.2.2.1 Google Native Client (NaCl) [30]

เป็นเทคโนโลยีกระบะทรายที่ทำให้สามารถนำรหัสโปรแกรมภาษา C++ มาทำงานบนเว็บเบราว์เซอร์ได้ NaCl พัฒนาโดย Google และสามารถใช้ได้บนเว็บเบราว์เซอร์ Google Chrome เท่านั้น Nacl นั้นได้ถูกออกแบบมาเพื่อให้เว็บเบราว์เซอร์มีประสิทธิภาพในระดับเดียวกับโปรแกรมบนเครื่องคอมพิวเตอร์ (desktop application) โดยที่ยังมีความปลอดภัยเหมือนเว็บแอปพลิเคชั่นอยู่จากระบบกระบะทรายสองชั้น (ชั้นหนึ่งจากตัว Nacl เอง ส่วนอีกชั้นเป็นของเว็บเบราว์เซอร์) ดังนั้น NaCl จึงเหมาะกับการใช้งานในเว็บแอปพลิเคชั่นที่ต้องการประสิทธิภาพสูง (high performance) โมดูลของ NaCl นั้นสามารถติดต่อกับ JavaScript บนหน้าเว็บหลักผ่านทางโปรโตคอล message passing เหมือนกับ web worker

NaCl มีความแตกต่างจาก Microsoft ActiveX, Microsoft Silverlight, Adobe Flash ตรงที่ NaCl นั้นจะเป็นเทคโนโลยีที่ให้รหัสโปรแกรมทำงานบนตัวเองที่เป็นกระบะทรายบนเว็บเบราว์เซอร์ ในขณะที่เทคโนโลยีตัวอื่นๆจะทำงานบนระบบปฏิบัติการบนเครื่องไคลเอนต์โดยตรง NaCl นั้นมีความคล้ายคลึงกับ JAVA Applet แต่แตกต่างกันตรงภาษาที่ใช้ โดย JAVA Applet นั้นจะใช้ภาษา JAVA ในขณะที่ NaCl จะใช้ภาษา C++

3.1.2.2.2 WebCL [31]

เป็น API ภาษา JavaScript เพื่อให้ใช้งาน OpenCL[32] บนเว็บเบราว์เซอร์ได้ โดย OpenCL นั้นจะเป็นเฟรมเวิร์คการคำนวณแบบขนาน (parallel computing) ที่สามารถใช้งานทรัพยากรอื่นๆ เช่น ซีพียูแบบมัลติคอร์ (multi-core CPU), หน่วยประมวลผลกราฟิก (GPU) บนเครื่องคอมพิวเตอร์เป้าหมายได้ การใช้งาน WebCL นั้นจะเขียนด้วยภาษา JavaScript ทั้งหมดยกเว้นส่วน kernel (คำสั่งที่จะไปทำงานบน GPU) นั้นจะต้องเขียนด้วยภาษา C++

เนื่องจาก WebCL ยังเป็นเฟรมเวิร์คที่อยู่ในช่วงต้นของการพัฒนา โดยในปัจจุบันจะมี WebCL อยู่ 2 ค่าย คือ ของ Nokia [33] ซึ่งสามารถใช้งานได้บน Mozilla Firefox (ต้องลงปลั๊กอินเพิ่ม) เท่านั้น และของ SamSung [34] ที่สามารถใช้งานได้เฉพาะเว็บเบราว์เซอร์ตระกูล webkit เท่านั้น

3.1.2.2.3 River trail [6]

เป็นเครื่องมือภาษา JavaScript สำหรับการคำนวณแบบขนานที่พัฒนาโดยบริษัท Intel เครื่องมือนี้สามารถใช้งานซีพียูแบบมัลติคอร์ได้โดยใช้โปรแกรมควบคุม (driver) OpenCL ของ Intel (OpenCL มีโปรแกรมควบคุมได้หลายเวอร์ชั่น โดยขึ้นกับบริษัทที่ผลิตอุปกรณ์ชิ้นนั้น) River trail นั้นได้นำเสนอภาษาการคำนวณแบบขนานในระดับที่สูงขึ้น (high-level abstraction) และใช้รูปแบบการคำนวณแบบกระจายเชิงข้อมูล (data parallelism) ทำให้สามารถใช้งานได้ง่าย และมีความปลอดภัยกว่าการที่ผู้พัฒนาต้องมาจัดการทรัพยากรต่างๆเอง

3.1.2.3 สรุปคุณสมบัติของเครื่องมือต่างๆ

คุณสมบัติของเครื่องมือต่างๆสำหรับเพิ่มประสิทธิภาพบนเว็บเบราว์เซอร์นั้นสามารถสรุปได้เป็นตารางที่ 4 ดังนี้

3.1.3 การทดลองเพื่อวัดผลประสิทธิภาพของเครื่องมือต่างๆสำหรับเพิ่มประสิทธิภาพบนเว็บเบราว์เซอร์

ในส่วนนี้จะเป็นการทดลองเพื่อวัดผลประสิทธิภาพของเครื่องมือต่างๆในหัวข้อ 3.1.2 โดยจะทำการทดลองเทียบกับภาษา C++ ซึ่งเป็นตัวแทนของโปรแกรมที่ทำงานบนเครื่องโดยตรง (native application)

3.1.3.1 Matrix multiplication

การทดลองนี้จะเป็นการทดลองเพื่อวัดประสิทธิภาพของเครื่องมือต่างๆ และเพื่อเปรียบเทียบกับโปรแกรมที่ทำงานบนเครื่องโดยตรง จึงได้ทำการใส่โปรแกรมที่เขียนด้วยภาษา C++ ขึ้นมา การทดลองนี้จะวัดผลโดยการวัดเวลาที่ใช้ในการคำนวณการคูณกันของเมทริกซ์ที่ขนาดต่างๆในแต่ละเครื่องมือ อัลกอริทึมสำหรับการคูณเมทริกซ์ในการทดลองนี้จะใช้อัลกอริทึมแบบพื้นฐานที่สุดที่มีความซับซ้อนทางเวลาเป็น $O(n^3)$ และค่าที่อยู่ในเมทริกซ์นั้นจะเป็นตัวแปรประเภท integer 32 บิต ที่มีค่าอยู่ระหว่าง -10 ถึง 10 ทั้งนี้เพื่อป้องกันการการล้น (overflow) ของผลลัพธ์จากการคูณของเมทริกซ์ รายละเอียดของเครื่องคอมพิวเตอร์ที่ใช้ทดสอบนั้นจะเหมือนกับการทดลองในหัวข้อ 3.1.1.1 และรายละเอียดของเครื่องมือต่างๆที่นำมาทดสอบมีดังนี้ (การทดลองนี้ไม่ได้นำ WebCL มาทดลองเนื่องจากโปรแกรมที่เขียนด้วย WebCL นั้นไม่สามารถทำงานได้บนสภาพแวดล้อมของเครื่องที่จะทดสอบ)

ตารางที่ 4 ตารางสรุปคุณสมบัติของเครื่องมือต่างๆ

features/tools	typed array	web worker	Google NaCl	WebCL	river trail
detail summary	an improve array for store typed variable.	threading model for web application	a sandbox technology that can execute native code within the web browser	a JavaScript wrapper of OpenCL for heterogeneous hardware parallel programming	a JavaScript parallel engine which provide high-level abstraction and data-parallelism for multi-core CPU
type and dependency	browser native	browser native	specific to Google Chrome web browser	- plug-in - OpenCL driver - WebCL plug-in - Mozilla Firefox(Nokia implementation) - Webkit-based browser (Samsung implementation)	- plug-in - Intel OpenCL driver - River Trail plug-in - Mozilla Firefox
safe execution	yes	yes	yes	no	yes
develop programming language	JavaScript	JavaScript	C/C++	C++ (kernel), JavaScript (WebCL API)	JavaScript
hardware utilization	no	no	no	CPU,GPU	CPU
thread model	no	yes	yes (pthread)	yes	yes
vendor/standard	Khronos	W3C, WHATWG	Google	Khronos	Intel

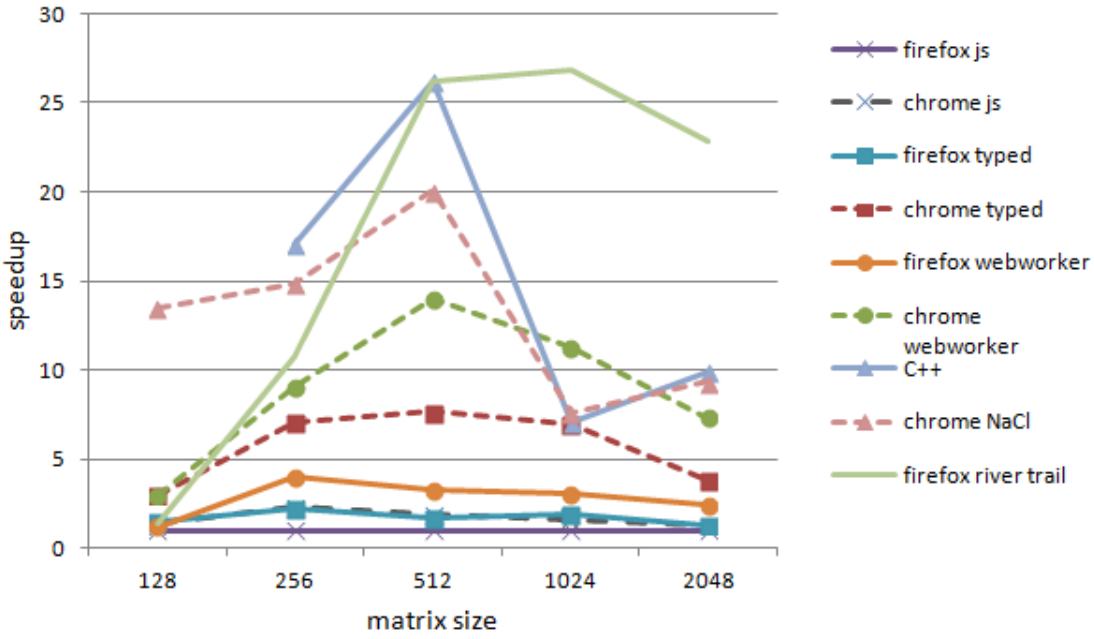
- โปรแกรมที่เขียนด้วยภาษา JavaScript ทำงานบน Google Chrome 23
- โปรแกรมที่เขียนด้วยภาษา JavaScript ทำงานบน Mozilla Firefox 17

- โปรแกรมที่เขียนด้วยภาษา JavaScript และมีการใช้งาน Typed array ทำงานบน Google Chrome 23
- โปรแกรมที่เขียนด้วยภาษา JavaScript และมีการใช้งาน Typed array ทำงานบน Mozilla Firefox 17
- web worker (จำนวน worker = 2) บน Google Chrome 23
- web worker (จำนวน worker = 2) บน Mozilla Firefox 17
- Google Native Client (ใช้ค่าพารามิเตอร์ –O2 ในการแปลรหัสโปรแกรม)
- River trail บน Mozilla Firefox 17
- C++ (ใช้ Microsoft Visual Studio 2010 Express และค่าพารามิเตอร์ –O2 ในการแปลรหัสโปรแกรม)

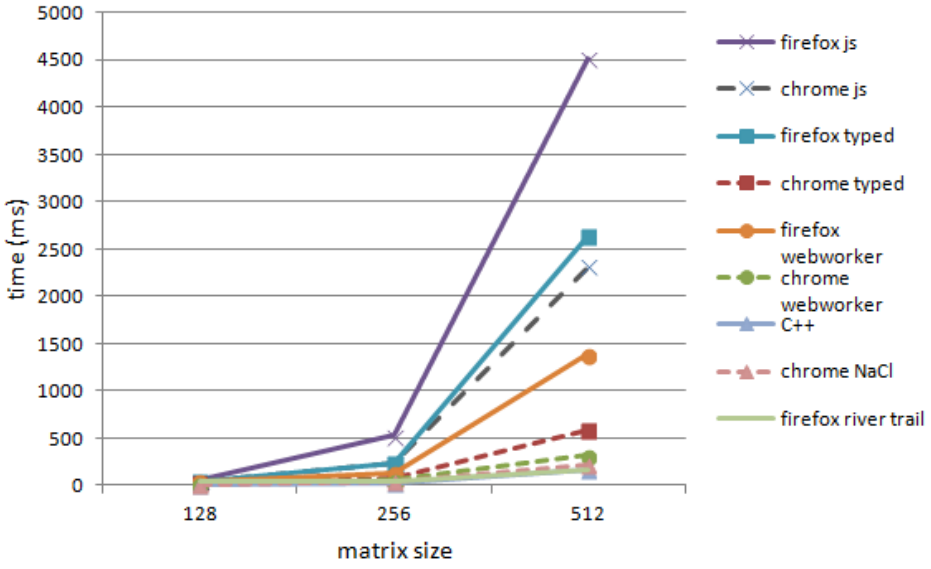
วิธีการวัดเวลาในการทดลองนี้จะเริ่มจับเวลาตั้งแต่การเรียกใช้งานฟังก์ชั่นการคูณเมทริกซ์และหยุดจับเวลาเมื่อระบบส่งคำตอบกลับมา ดังนั้นเวลาในที่นี้จึงนับรวมเวลาที่ใช้ในการถ่ายโอนข้อมูลเมทริกซ์จาก JavaScript ไปยังโมดูลต่างๆด้วยในกรณีของ web worker และ NaCl การทดลองนี้จะทำการทดลองอย่างละ 5 ครั้ง จากนั้นจะตัดเอาค่าเวลาที่มากสุดกับน้อยสุดออกไปและใช้ค่าเฉลี่ยจากค่าเวลาที่เหลือ ซึ่งสามารถสรุปผลการทดลองเป็นตารางที่ 5 และรูปที่ 3 ได้ดังนี้

ตารางที่ 5 เวลาที่ใช้ในการคูณเมทริกซ์ (ms) ที่ขนาดต่างๆของแต่ละเครื่องมือ

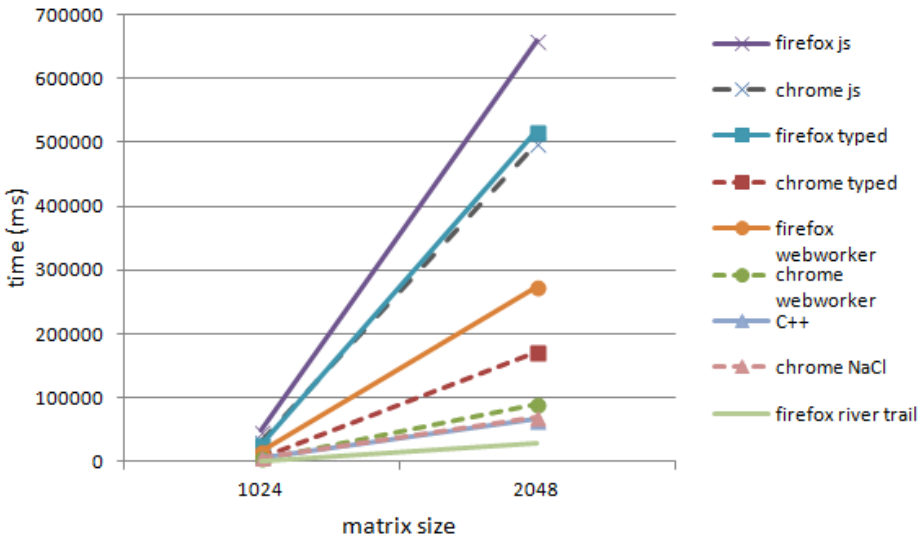
Tools / Matrices size	128 * 128	256 * 256	512 * 512	1024 * 1024	2048 *2048
Firefox JavaScript	54	529	4513	49242	659306
Chrome JavaScript	40	228	2311	31071	497782
Firefox typed array	37	235	2638	26465	518578
Chrome typed array	18	75	590	7069	172573
Firefox web worker	44	131	1392	16131	274786
Chrome web worker	18	58	322	4367	90466
Chrome NaCl	4	36	225	6493	70569
Firefox river trail	40	49	172	1836	28873
C++	0	31	172	7005	66691



รูปที่ 3 speedup ของประสิทธิภาพของการคูณเมทริกซ์ในเครื่องมือต่างๆ โดยมี Mozilla Firefox JavaScript เป็น baseline

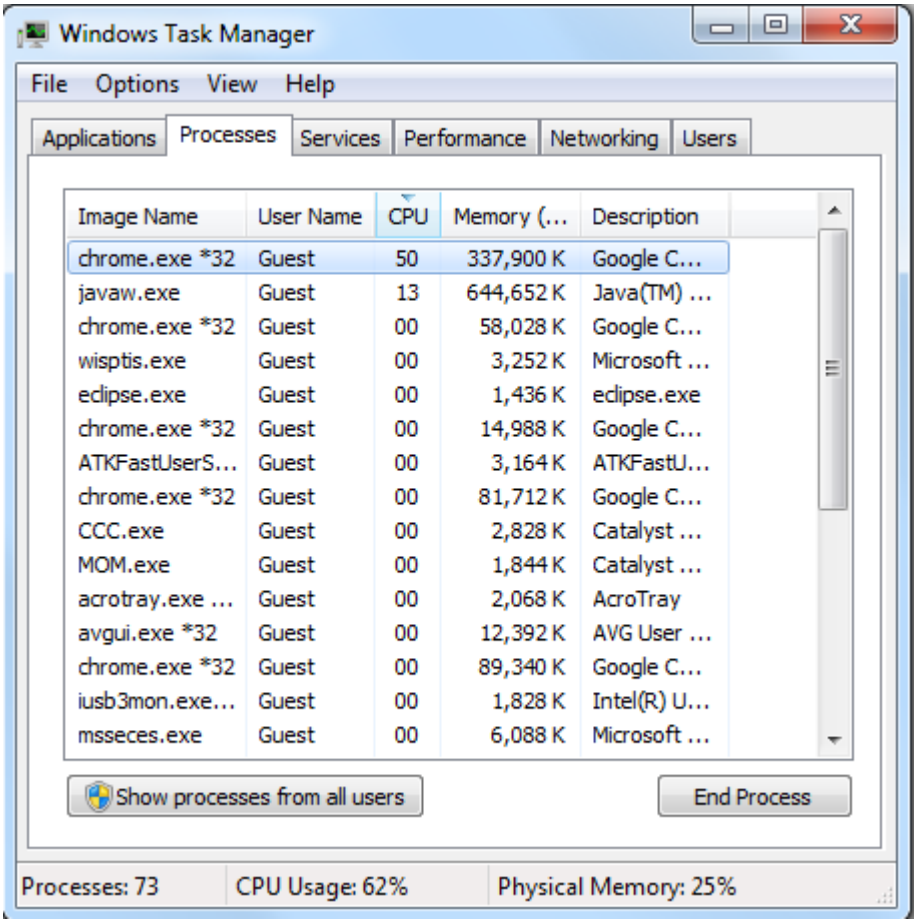


รูปที่ 4 เวลาที่ใช้ในการคูณเมทริกซ์ขนาด 128x128, 256x256 และ 512x512 ของแต่ละเครื่องมือ

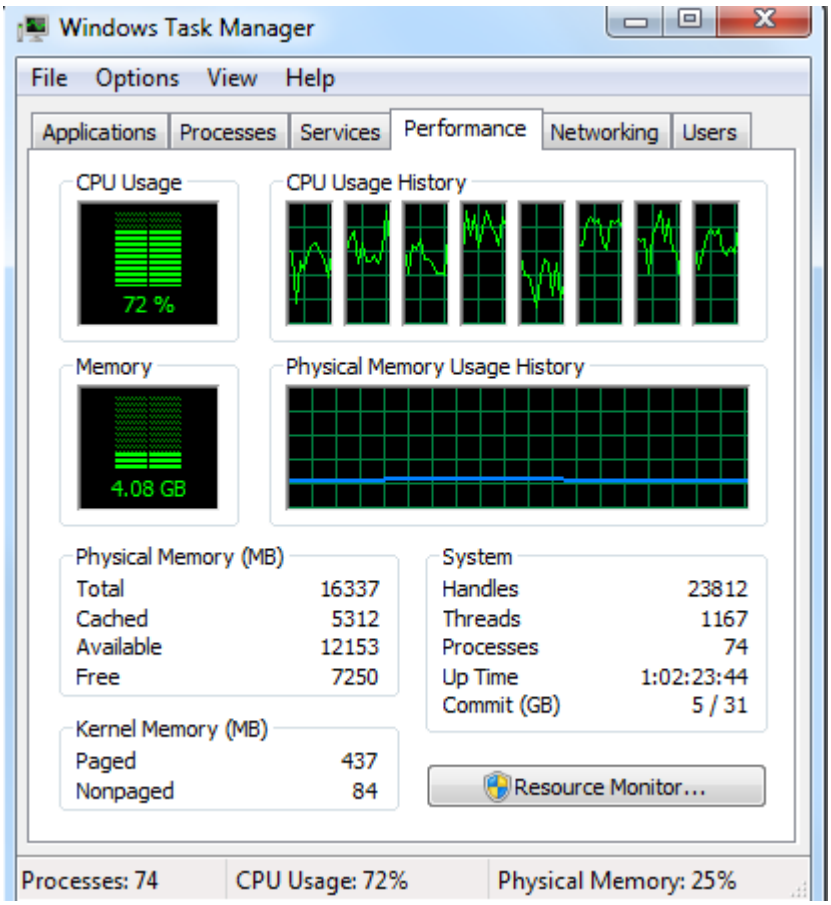


รูปที่ 5 เวลาที่ใช้ในการคูณเมทริกซ์ขนาด 1024x1024 และ 2048x2048 ของแต่ละเครื่องมือ

จากผลการทดลองนั้น เราจะสังเกตได้ว่าประสิทธิภาพของ River trail นั้นดีที่สุด ซึ่งสามารถอธิบายเหตุผลได้ว่าเป็นเพราะ river trail นั้นมีการใช้งานในส่วนของซีพียูหลายคอร์เข้าช่วย ทำให้มีประสิทธิภาพสูงกว่าเครื่องมืออื่นๆ สำหรับ C++ นั้นมีประสิทธิภาพที่ดีรองลงมา ยกเว้นกรณีของเมทริกซ์ขนาด 1024x1024 นั้น ในข้อสันนิษฐานเบื้องต้นนั้นคาดว่าเป็นเพราะขนาดของเมทริกซ์มีขนาดใหญ่กว่าขนาดของแคชบน CPU ทำให้ต้องเสียเวลาในการดึงข้อมูลจากหน่วยความจำ ส่งผลให้ประสิทธิภาพโดยรวมลดลง แต่กรณีที่ Google Chrome web worker ทำได้ดีกว่า C++ ที่เมทริกซ์ขนาด 1024x1024 นั้น คาดว่าเป็นเพราะ Google Chrome เป็นเว็บเบราว์เซอร์ชนิดหลายโปรเซส ซึ่งแต่ละแทปบนเว็บเบราว์เซอร์ของ Chrome นั้นจะเป็นโปรเซสย่อยๆที่มีโปรเซสหลักของเว็บเบราว์เซอร์ควบคุมอยู่อีกที ทำให้ web worker ของ Chrome นั้นสามารถใช้งานซีพียูหลายคอร์ได้ โดยการกระจายเทรด worker ให้แต่ละคอร์ของซีพียู



รูปที่ 6 การใช้งาน CPU ของ Google Chrome เมื่อใช้ web worker 4 worker บนเครื่องคอมพิวเตอร์ที่มีซีพียู 8 คอร์



รูปที่ 7 การใช้ทรัพยากรของ CPU ในแต่ละคอร์ของ Google Chrome ที่ใช้ web worker 4 worker บนเครื่องคอมพิวเตอร์ที่มีซีพียู 8 คอร์

หากพิจารณาถึงการใช้งานของซีพียูจะพบว่าเว็บเบราว์เซอร์ทั้งสองนั้นจะมีการใช้งานซีพียูแบบเต็มประสิทธิภาพ (ใช้งานซีพียูเต็มที่ในคอร์นั้นๆ) ภาพที่ 6 และ 7 แสดงให้เห็นว่าหากเครื่องของไคลเอนต์มีซีพียูหลายคอร์ การใช้ web worker จะทำให้เว็บเบราว์เซอร์ (ในการทดลองนี้ทดลองเฉพาะ Google Chrome และ Mozilla Firefox) จะสามารถใช้งานซีพียูหลายคอร์ได้โดยอัตโนมัติ ทั้งนี้ Chrome จะมีจุดเด่นข้อหนึ่งที่เหนือกว่า Firefox ตรงที่ Chrome เป็นเว็บเบราว์เซอร์ชนิดหลายโปรเซส ทำให้หน้าเว็บแท็บไม่ค้างในกรณีที่ใช้งาน JavaScript ประมวลผลมาก (ในกรณีที่ไม่ได้ใช้ web worker) ทำให้ไม่รบกวนการใช้งานในแท็บอื่นๆของเว็บเบราว์เซอร์

นอกจากนี้ การทดลองนี้ยังมีจุดที่น่าสนใจอยู่ 2 จุดได้แก่ จุดแรก river trail มีประสิทธิภาพที่ไม่ค่อยดีที่ขนาดเมทริกซ์เล็ก เพราะความสิ้นเปลืองของการใช้งานซีพียูหลายคอร์ที่ไม่คุ้มกับขนาดข้อมูลขนาดเล็ก และจุดที่สองคือ NaCl นั้นให้ประสิทธิภาพที่ใกล้เคียงกับภาษา

C++ แต่อาจจะมีประสิทธิที่แย่กว่าที่ข้อมูลขนาดเล็ก ซึ่งมาจากเหตุผลเรื่องความสิ้นเปลืองที่ได้อธิบายไว้ข้างต้น

3.2 การเปรียบเทียบคุณสมบัติในด้านต่าง ๆของระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์เทียบกับเฟรมเวิร์ก BOINC

ในส่วนนี้จะเป็นการเทียบคุณสมบัติในด้านต่างๆระหว่างระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์เทียบกับเฟรมเวิร์ก BOINC ซึ่งเป็นระบบที่มีอยู่แล้ว และเป็นระบบที่ได้รับความนิยมมากระบบหนึ่ง การเปรียบเทียบในส่วนนี้มีจุดประสงค์เพื่อศึกษาถึงข้อ แตกต่างและหา tradeoff ของทั้งสองระบบ ซึ่งอันที่จริงแล้ว BOINC นั้นมีแนวคิดที่แตกต่างจากระบบการคำนวณบนเว็บเบราว์เซอร์ โดย BOINC นั้นมีแนวคิดที่ว่าจะขอใช้ทรัพยากรเครื่องที่ไม่ได้ถูกใช้งาน (เช่น ช่วง screen saver) ทำให้ BOINC มีความสามารถที่จะตรวจสอบการใช้งาน CPU ได้ ในขณะที่เว็บเบราว์เซอร์ไม่สามารถทำเช่นนั้นได้ เนื่องจากสภาพแวดล้อมกระบะทรายของตัวเว็บเบราว์เซอร์เองที่ทำให้ไม่สามารถเข้าถึงทรัพยากรของเครื่องคอมพิวเตอร์ได้ ดังนั้นการใช้งานในส่วนของเว็บเบราว์เซอร์จึงต้องเกิดจากการที่ผู้ใช้เต็มใจที่จะยอมสละทรัพยากรส่วนหนึ่งในขณะที่ผู้ใช้งานยังใช้งานเครื่องคอมพิวเตอร์นั้นๆอยู่ อย่างไรก็ตาม BOINC ก็เป็นระบบที่รู้จักกันแพร่หลาย ประกอบกับยังไม่มีระบบการคำนวณแบบกระจายแบบเว็บเบราว์เซอร์ที่สมบูรณ์จนนำมาเปรียบเทียบได้ ดังนั้น BOINC จึงเป็นตัวเลือกที่เหมาะสมสุดในการนำมาเปรียบเทียบ นอกจากนี้ BOINC ยังเป็นโอเพนซอร์สและมีเอกสารที่ค่อนข้างสมบูรณ์ ทำให้การศึกษาหลักการทำงานทำได้ง่ายกว่าเฟรมเวิร์กตัวอื่นๆ ซึ่งการเปรียบเทียบลักษณะต่างๆของทั้งสองระบบนั้น สามารถสรุปได้เป็น ตารางที่ 6

ตารางที่ 6 การเปรียบเทียบลักษณะเด่นของระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์และเฟรมเวิร์ก BOINC

	BOINC	Browser-based
Performance	★★	★
Storage	★★	★
Resource management	★★	★
Barrier of entrance	★	★★
Portability & rapid development	★	★★
Security	★	★★
Cycle steal	★★	★★

จากตารางนี้เราสามารถสรุปได้ดังนี้

- Performance ในด้านของประสิทธิภาพนั้น BOINC ดีกว่า browser-based เพราะ BOINC ได้รับประสิทธิภาพจากภาษา C++ และสามารถเข้าใช้ทรัพยากรเครื่อง (ซีพียูหลายคอร์หรือหน่วยประมวลผล) ได้โดยตรง ในขณะที่ browser-based นั้นไม่สามารถทำได้เพราะติดสภาพแวดล้อมของกระบะทราย ถึงแม้ว่าจะมีบางเครื่องมือที่ทำให้สามารถเข้าถึงทรัพยากรเครื่องอื่นๆได้ แต่เครื่องมือเหล่านั้นก็สามารถใช้ได้เพียงบางเว็บเบราว์เซอร์และยังมีข้อจำกัดอื่นๆอยู่

- Storage ในส่วนนี้ BOINC ก็ยังดีกว่า browser-based เพราะ BOINC สามารถเข้าถึงหน่วยเก็บข้อมุลของไคลเอนต์ได้โดยตรง ในขณะที่เว็บเบราว์เซอร์มี localStorage[35] ซึ่งมีขนาดพื้นที่ให้เพียง 2-5 MB ถ้าผู้พัฒนาต้องการพื้นที่เก็บข้อมูลเพิ่ม ผู้พัฒนาสามารถใช้งานในส่วนของ FileSystem API[36] ได้ อย่างไรก็ตามเว็บเบราว์เซอร์ที่รองรับ FileSystem API ยังมีค่อนข้างจำกัดและยังไม่ค่อยแพร่หลาย

- Resource management การควบคุมการใช้งานทรัพยากรเครื่องก็เป็นอีกปัจจัยหนึ่งที่สำคัญเช่นกัน เพราะคุณสมบัตินี้สามารถจำกัดไม่ให้ใช้งานเครื่องไคลเอนต์มากเกินไป โดย BOINC สามารถเข้าถึงและควบคุมการใช้งาน CPU ได้ ในขณะที่เว็บเบราว์เซอร์นั้นไม่สามารถทำได้เพราะติดสภาวะกระบะทราย ทำให้ไม่สามารถเข้าถึงทรัพยากรของเครื่องได้โดยตรง

- Barrier of entrance ในส่วนนี้ browser-based ดีกว่า BOINC เพราะการเข้าร่วมของ BOINC นั้น ไคลเอนต์ต้องดาวน์โหลดและติดตั้งโปรแกรมลงเครื่องก่อน ในขณะที่เว็บเบราว์เซอร์นั้น ไคลเอนต์แค่เปิดเว็บเบราว์เซอร์ไปยัง URL ที่กำหนดไว้ก็สามารถเข้าร่วมได้ทันทีโดยที่ไม่ต้องลงโปรแกรมอะไรเพิ่มเติม

- Portability & rapid development ระบบการประมวลผลแบบอุทิศนั้นควรรองรับความหลากของแพลตฟอร์มและระบบปฏิบัติการให้ได้มากที่สุดเพื่อขยายความสามารถในการคำนวณโดยรวม ซึ่ง BOINC นั้น ผู้พัฒนาจำเป็นต้องสร้างโปรแกรมเฉพาะออกมาสำหรับแพลตฟอร์มนั้นๆ ในขณะที่ฝั่ง browser-based

ผู้พัฒนาสามารถพัฒนาโปรแกรมด้วยภาษา JavaScript เพียงครั้งเดียวแล้วสามารถนำไปทำงานบนแพลตฟอร์มต่างๆได้ทันที (write once run anywhere) และภาษา JavaScript นั้นถือว่าเป็นภาษาที่มีความรวดเร็วในการพัฒนา (rapid development) เนื่องจากเป็นภาษาโปรแกรมที่มีความยืดหยุ่นและสามารถรองรับรูปแบบการเขียนโปรแกรมที่หลากหลาย ดังนั้น browser-based ดีกว่า BOINC ในคุณสมบัตินี้

- Security ในส่วนนี้ browser-based ดีกว่า BOINC เพราะว่าเว็บเบราว์เซอร์นั้นมีการใช้งานสภาพแวดล้อมกระบะทราย จึงมีความปลอดภัยมากกว่าโปรแกรมของ BOINC นั้นทำงานบนระบบปฏิบัติการโดยตรง

- Cycle steal ทั้ง BOINC และ browser-based สามารถใช้งานซีพียูได้เต็มประสิทธิภาพทั้งคู่ (การตั้งค่าเบื้องต้นของ BOINC นั้นคือใช้งานซีพียูที่มีทั้งหมด 100% ในเวลาที่ไม่มีโปรเซสอื่นของผู้ใช้กำลังใช้งานอยู่) แต่ BOINC นั้นสามารถลดการทำงานของโปรเซสตนเองลงเพื่อไม่ให้รบกวนโปรเซสอื่นได้

จากตารางสามารถสรุปได้ว่า BOINC นั้นมีจุดเด่นที่ มีประสิทธิภาพสูงเพราะสามารถใช้ทรัพยากรของเครื่องไคลเอนต์ได้อย่างเต็มที่ ในขณะที่ browser-based นั้นมีจุดเด่นที่ ไคลเอนต์สามารถเข้าถึงได้ง่าย, มีความปลอดภัยและความง่ายในการพัฒนา ซึ่งจากจุดเด่นดังกล่าวสามารถสรุปได้ว่างานที่เหมาะสมกับ BOINC นั้นควรเป็นงานที่มีความต้องการประสิทธิภาพในการคำนวณสูงและผู้พัฒนาไม่มีปัญหาในการพัฒนาและทำให้เหมาะที่สุด (optimize) สำหรับระบบนั้นๆ ตัวอย่างเช่น งานคำนวณทางด้านวิทยาศาสตร์และคณิตศาสตร์ ในขณะที่งานที่เหมาะสมกับ browser-based นั้นจะเป็นงานที่ไม่เน้นเรื่องประสิทธิภาพในการคำนวณมากแต่ต้องการความรวดเร็วในการพัฒนาและความง่ายในการเข้าร่วม ตัวอย่างเช่น การจำลองมอนติคาร์โล, งานทางด้านการคำนวณเชิงวิวัฒนาการ, การจำลองอัลกอริทึมการซื้อขาย (algorithmic trading), bitcoin miner เป็นต้น สำหรับข้อเปรียบเทียบคุณสมบัติต่างๆแบบเชิงลึกสามารถดูได้ในตารางที่ 7

ตารางที่ 7 การเปรียบเทียบลักษณะต่างๆของระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์และเฟรมเวิร์ก BOINC แบบละเอียด

key attributes	BOINC	web application
barrier of entrance	Client manually install program, choose project and configuration resource.	Clients open the specific web page.
execution environment	native application compiled to specific platform	web browser
storage	local file system with user defined in size	- localStorage (2 – 5 MB) - file API (exclusive to Google Chrome, Safari)
developing language	C/C++ (default)	- JavaScript - C/C++ with Google NaCl
deployment	Client manual install BOINC Manager	instantly deployed with platform independence
protocol	- HTTP with client-server architecture - can extend to use p2p or other protocol	- HTTP with client-server architecture - can extend to use p2p with webRTC (early stage)
hardware utilization	fully supported - multi-core CPU managed by BOINC Manager - GPU: developer manual create program with GPU supported.	- support multi-core CPU and GPU via WebCL (plug-in) - multi-core CPU with River Trail (Intel OpenCL driver)
safe execution	no	yes with sandbox environment
thread	support	support with web workers
portable from existed program	BOINC wrapper, API for other programming language(ex. Python, JAVA)	- Emscripten (compile to LLVM then convert to JavaScript) - C/C++ (Google NaCl)
user base	computer user who want to contribute their computing power to science project and do not mind to install additional software	normal computer user who want to contribute their computing power to science project and do not want to install additional software
resource management	can control CPU cycle	cannot control CPU cycle directly. The only way to control manage the resource usage is limited the time to execute code with setTimeOut() function
use case	scientific and mathematical domain	adhoc projects with more relaxed constraints.

บทที่ 4
แนวทางการออกแบบระบบ

ในบทนี้กล่าวถึงแนวทางการออกแบบระบบของเฟรมเวิร์กการคำนวณแบบกระจายบนเว็บเบราว์เซอร์ โดยเนื้อหาในบทนี้ได้ถูกแบ่งออกเป็น 4 ส่วนคือ เป้าหมายของการออกแบบ, ภาพรวมของระบบ, รูปแบบข้อมูลของระบบ และสถาปัตยกรรมของระบบ ในส่วนแรกกล่าวถึงรายละเอียดของเป้าหมายและวัตถุประสงค์ของการออกแบบระบบ รวมไปถึงข้อสันนิษฐานเบื้องต้น (assumption) ส่วนที่สองกล่าวถึงรูปแบบข้อมูลที่ใช้ของระบบ รูปแบบของการเขียนโปรแกรมที่ใช้ในเฟรมเวิร์ก ส่วนที่สามกล่าวถึงภาพรวมของระบบ โดยอธิบายทั้งในฝั่งมุมมองของไคลเอนต์และเซิร์ฟเวอร์ เพื่อแสดงถึงแนวทางและภาพรวมของการออกแบบสถาปัตยกรรมของระบบ และส่วนสุดท้ายเป็นการอธิบายถึงการออกแบบสถาปัตยกรรมของระบบ

4.1 เป้าหมายของการออกแบบ

งานวิจัยนี้เป็นการนำเสนอระบบการประมวลผลแบบกระจาย โดยใช้เทคโนโลยีทางด้านเว็บแอปพลิเคชั่นเป็นหลัก ซึ่งมีคุณลักษณะของระบบ ดังนี้

- เป็นระบบเฟรมเวิร์กที่ช่วยให้ผู้พัฒนาสามารถสร้างโครงการสำหรับระบบการประมวลผลแบบกระจายบนเว็บเบราว์เซอร์ได้ง่ายขึ้น ทั้งนี้โครงการนั้นต้องไม่มีจุดประสงค์ในทางที่ผิด (เช่น การทำ Denial-of-service attack เป็นต้น)
- เป็นระบบเฟรมเวิร์กเว็บแอปพลิเคชั่น อาสาสมัครสามารถเข้าร่วมได้ทันที โดยใช้โปรแกรมเว็บเบราว์เซอร์เข้าไปเว็บตามที่ระบุไว้ และไม่จำเป็นต้องติดตั้งโปรแกรมอื่นๆเพิ่มเติม
- ลักษณะงานที่ทำควรเป็นงานที่สามารถแยกกันทำได้โดยอิสระ (embarrassingly parallel) กล่าวคือ ไม่มีการติดต่อระหว่าง worker node (ในที่นี้คืออาสาสมัคร) และแต่ละ worker node สามารถทำงานได้โดยเป็นอิสระต่อกัน

4.2 รูปแบบข้อมูลของระบบ (data model)

ในส่วนนี้จะเป็นการอธิบายถึง คำศัพท์และรูปแบบข้อมูลต่างๆที่ใช้ในเฟรมเวิร์กนี้ โดยในเฟรมเวิร์กจะมีการแบ่งรูปแบบของงานออกเป็น job, task, result และ workflow

- Job คือ ขั้นตอนการดำเนินงานขั้นตอนหนึ่ง ซึ่งในโครงการหนึ่งสามารถมี job ได้หลายตัว ซึ่ง job แต่ละตัวนั้นจะประกอบด้วยโปรแกรม JavaScript ที่

ทำงานบนฝั่งไคลเอนต์ รวมไปถึงรายละเอียดการตั้งค่าต่างๆของการดำเนินงาน เช่น ตำแหน่งที่เก็บไฟล์อินพุตของงานนี้ เป็นต้น

- Task คือ งานย่อยๆของ job นั้น โดยจำนวนของ task ใน job หนึ่งนั้นๆ จะสามารถกำหนดได้เองโดยผู้พัฒนา หรืออาจขึ้นกับจำนวนไฟล์อินพุตของ job นั้นๆก็ได้
- Result คือผลลัพธ์ที่ได้จากไคลเอนต์ในแต่ละ task โดยในแต่ละ task สามารถมีได้หลาย result เพื่อใช้ในการตรวจทานคำตอบกับ result ชุดอื่นๆ(ระบบใช้วิธีการหาเสียงข้างมาก (majority voting) ในการตรวจคำตอบ หากคำตอบนั้นมีผู้ตอบตรงกันหลาย result ก็มีโอกาสที่จะเป็นคำตอบที่ถูก โดยค่าที่แนะนำส่วนใหญ่คือ 3 กล่าวคือคำตอบที่ถูกต้องควรจะมีคนตอบตรงอย่างน้อย 3 คน ทั้งนี้ผู้พัฒนาสามารถปรับเปลี่ยนค่าเพื่อลดความสิ้นเปลืองลงได้) นอกจากนี้ result ในแต่ละชุดจะมีการสุ่มข้อความ MD5 ขึ้นมาเพื่อป้องกันไคลเอนต์ที่ไม่หวังดีปลอม result ขึ้นมา
- Workflow เป็นผังงานที่เชื่อมแต่ละ job เข้าด้วยกัน โดย workflow นั้นแสดงลำดับขั้นตอนการทำงานของ job

เพื่อให้เห็นภาพที่ชัดเจนขึ้น จึงขอยกตัวอย่างการสร้างโปรแกรมโดยใช้รูปแบบข้อมูลแบบนี้ขึ้นมา โดยโปรแกรมที่จะยกตัวอย่างคือ โปรแกรมการจำลองการสุ่มแบบมอนติคาร์โลเพื่อแก้ปริพันธ์จำกัดเขต (integral) ขึ้นมา โดยงานนี้จะเป็นการหาปริพันธ์จำกัดเขตจำนวนทั้งหมด 1000 ตัว เนื่องจากงานนี้สามารถแยกได้โดยอิสระและไม่ขึ้นกัน (การหาปริพันธ์จำกัดเขตแต่ละตัวไม่มีความเกี่ยวข้องกัน สามารถแยกหากันได้) ดังนั้นจึงสามารถแบ่งเป็นงานย่อยๆในไคลเอนต์ทำได้ ซึ่งในที่นี้ได้วางแผนให้ไคลเอนต์แต่ละคนหาปริพันธ์จำนวน 5 ตัว ไฟล์อินพุตของงานนี้จึงเป็น 1000/5 = 200 ไฟล์ (1 ไฟล์อินพุตต่องานย่อย 1 งาน)

- Job ในที่นี้มีอยู่ job เดียวคือการจำลองการสุ่มแบบมอนติคาร์โลเพื่อแก้ปริพันธ์จำกัดเขต โดยในส่วนนี้จะมีรหัสโปรแกรมสำหรับทำงานบนเว็บเบราว์เซอร์ของไคลเอนต์ และการตั้งค่าต่างๆ
- Task ในที่นี้มีอยู่ทั้งหมด 200 task (ตามจำนวนไฟล์อินพุต)
- Result มีจำนวนไม่แน่นอนขึ้นอยู่กับการตั้งค่าของ job ตัวอย่างเช่น หาก job มีการตั้งค่าให้ใน task มีการต้องเช็คคำตอบว่า result ที่ได้จากไคลเอนต์ควรมี

คำตอบที่เหมือนกันอย่างน้อย 3 result ระบบจะมีจำนวน result อย่างน้อย 200*3 = 600 result นั่นก็คือมีงานย่อยส่งให้ไคลเอนต์ทำอย่างน้อย 600 งาน

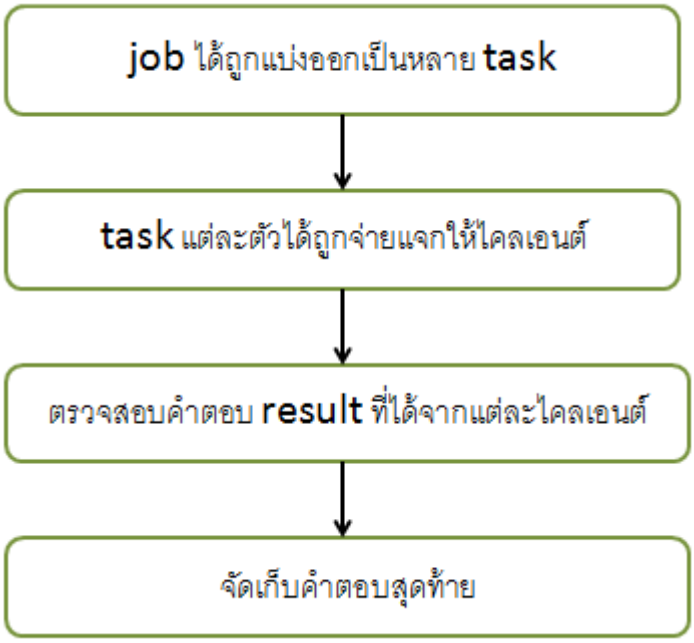
- Workflow เนื่องจากตัวอย่างนี้มีเพียงแค่ job เดียว จึงไม่จำเป็นต้องมี workflow ก็ได้

4.3 ภาพรวมของระบบ

เฟรมเวิร์กนี้ใช้ระบบสถาปัตยกรรมแบบไคลเอนต์-เซิร์ฟเวอร์ (client-server) โดยในส่วนนี้จะแบ่งภาพรวมออกเป็น 2 มุมมอง ได้แก่มุมมองในฝั่งของเซิร์ฟเวอร์ และมุมมองในฝั่งของไคลเอนต์

4.3.1 มุมมองในฝั่งของเซิร์ฟเวอร์

ภาพรวมของระบบในมุมมองของฝั่งเซิร์ฟเวอร์นั้น สามารถแสดงได้ดังรูปที่ 8 ดังนี้



รูปที่ 8 ภาพรวมของระบบฝั่งเซิร์ฟเวอร์

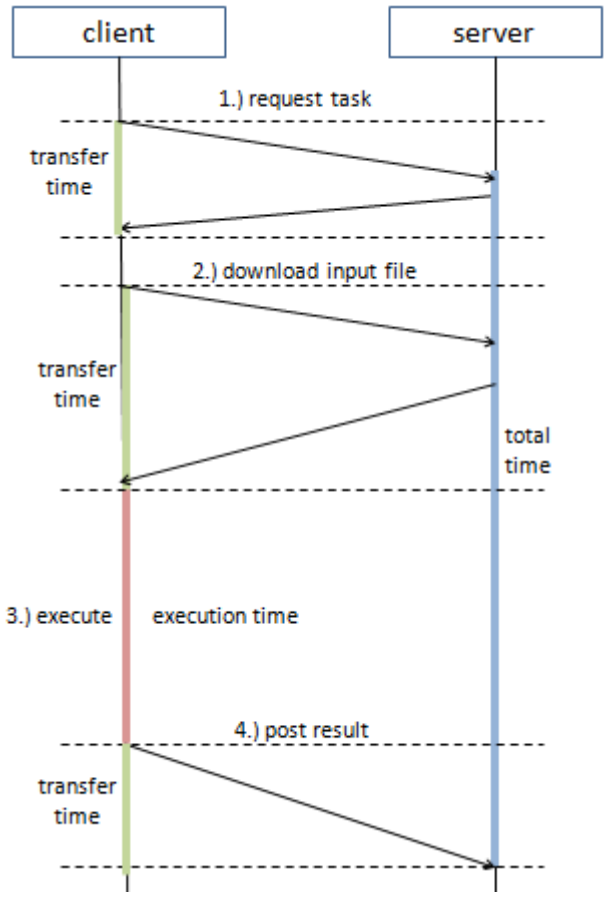
ขั้นตอนการทำงานของเซิร์ฟเวอร์สามารถสรุปได้ดังนี้

- job ถูกแบ่งออกเป็นหลาย task ย่อย และแต่ละ task ถูกแจกจ่ายให้ไคลเอนต์
- เซิร์ฟเวอร์ตรวจสอบคำตอบที่ได้จากไคลเอนต์

- เซิร์ฟเวอร์จัดเก็บคำตอบที่ถูกต้อง โดยวิธีการจัดเก็บขึ้นกับการตั้งค่าของ job นอกจากนี้ขั้นตอนนี้ยังคอยจัดการลบ result ที่ซ้ำซ้อนออกไปด้วย

4.3.2 มุมมองในฝั่งของไคลเอนต์

ภาพรวมของระบบในมุมมองของฝั่งไคลเอนต์นั้น สามารถแสดงได้ดังรูปที่ 9 ดังนี้



รูปที่ 9 ภาพรวมของระบบฝั่งไคลเอนต์

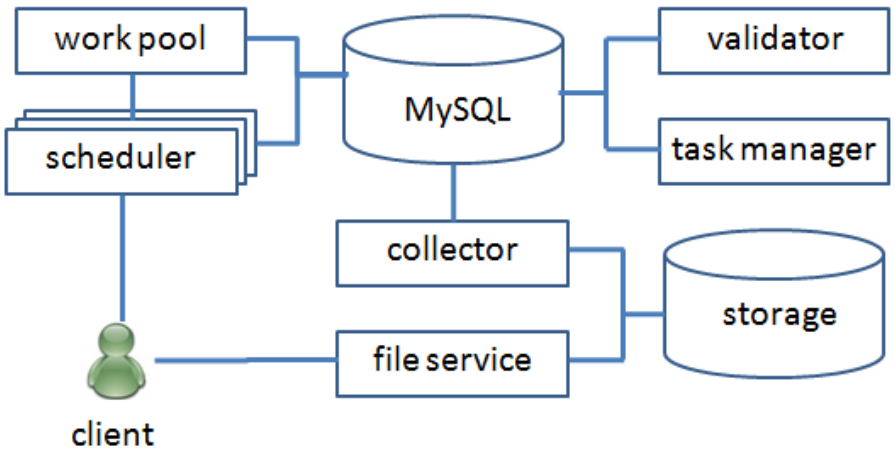
ขั้นตอนการทำงานของฝั่งไคลเอนต์ สามารถสรุปได้ดังนี้

- การทำงานเริ่มขึ้นเมื่อไคลเอนต์ได้เปิดเว็บเบราว์เซอร์ไปยังหน้าเว็บที่ได้กำหนดไว้ จากนั้น JavaScript บนหน้าเว็บนั้นทำการโหลดโปรแกรม JavaScript ต่างๆที่จำเป็นในการทำงานมา
- โปรแกรม JavaScript บนฝั่งไคลเอนต์นั้นทำการส่งคำร้อง (request) ของานไปยังเซิร์ฟเวอร์ เมื่อเซิร์ฟเวอร์ได้รับคำร้องแล้วจะส่ง task พร้อมกับ result กลับมาให้ไคลเอนต์ โดย task นั้นมีการระบุตำแหน่ง URL ของไฟล์อินพุตด้วย

- ไคลเอนต์ทำการดาวน์โหลดไฟล์อินพุตจากเซิร์ฟเวอร์ จากนั้นจึงเริ่มทำการประมวลผลตามโปรแกรม JavaScript ที่ได้ดาวน์โหลดมาจากเซิร์ฟเวอร์
- เมื่อไคลเอนต์ประมวลผลเสร็จแล้ว ไคลเอนต์จะส่ง result กลับไปยังเซิร์ฟเวอร์ และทำการส่งคำร้องของงานถัดไป ซึ่งกระบวนการดังกล่าวจะทำซ้ำไปเรื่อยๆ จนกว่าผู้ใช้งานจะปิดโปรแกรมเว็บเบราว์เซอร์ หรือไคลเอนต์ไม่สามารถติดต่อไปยังเซิร์ฟเวอร์ได้

4.4 **การออกแบบสถาปัตยกรรมของระบบ** (system architecture)

การออกแบบระบบนั้นจะใช้รูปแบบเดียวกับ BOINC [13] โดยการแยกการทำงานออกเป็นโมดูลต่างๆ แต่ละโมดูลติดต่อสื่อสารกันผ่านทางฐานข้อมูล MySQL ซึ่งรูปแบบนี้จะมีข้อดีตรงที่โมดูลแต่ละตัวจะเป็นอิสระต่อกัน ทำให้เมื่อเกิดเหตุขัดข้องขึ้น ความเสียหายไม่เกิดขึ้นกับทั้งระบบ และหากโมดูลบางโมดูลเกิดความล่าช้าขึ้น โมดูลอื่นๆก็จะไม่รับผลกระทบไปด้วย (หากโมดูลนั้นติดต่อกันเองโดยตรงโดยไม่ผ่านฐานข้อมูล เมื่อโมดูลเกิดความล่าช้าขึ้น อาจส่งผลให้การตอบรับและส่งข้อมูลกลับเกิดความล่าช้า ทำให้โมดุลที่มาติดต่อเกิดความล่าช้าไปด้วย) ซึ่งสถาปัตยกรรมของระบบจะประกอบไปด้วย 6 โมดูล โดยสามารถแสดงได้เป็น รูปที่ 10



รูปที่ 10 สถาปัตยกรรมของระบบ

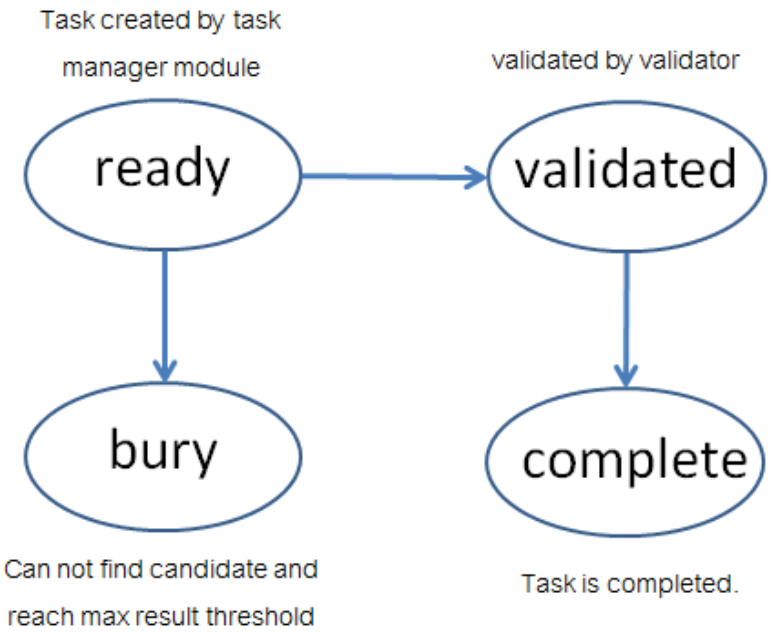
- **Scheduler** ทำหน้าที่คอยจัดการกับการคำรองของงานจากไคลเอนต์และคอยรับผลลัพธ์จากไคลเอนต์ สำหรับการตอบรับคำร้องของงานนั้น scheduler ส่งข้อมูลของ task, ตำแหน่งที่อยู่ของไฟล์อินพุต และรหัสโปรแกรมภาษา JavaScript

กลับไป ในส่วนของการรับผลลัพธ์จากไคลเอนต์นั้น โมดูลนี้อัปเดตข้อมูลที่เกี่ยวข้องบนฐานข้อมูล โมดูลนี้สามารถทำงานพร้อมกันได้หลายตัวเพื่อเพิ่ม throughput ได้

- **Work pool** ทำหน้าที่คล้าย shared memory สำหรับ scheduler โดยโมดูลนี้คอยทำการดึงข้อมูลมาจากฐานข้อมูล แล้วทำการจัดสรรแบ่งข้อมูลให้แต่ละ scheduler ซึ่งโมดูลนี้สามารถช่วยลดภาระในการติดต่อกับฐานข้อมูลได้ และช่วยให้สามารถแบ่งงานให้แต่ละ scheduler ได้อย่างเท่าเทียมกัน
- **File Service** เป็นส่วนที่เปิดการเชื่อมต่อแบบ HTTP เพื่อให้ไคลเอนต์สามารถเข้าไปดาวน์โหลดไฟล์อินพุตจาก Storage ได้
- **Validator** ทำหน้าที่คอยตรวจสอบผลลัพธ์ในแต่ละ task โดยใช้รูปแบบของการลงคะแนนเสียงส่วนใหญ่ (majority voting) ในการตัดสินหาคำตอบที่ถูกต้อง โดยขั้นตอนแรกระบบจะสร้าง result หลายๆตัวสำหรับ 1 task แล้วส่งไปให้ไคลเอนต์ใส่ผลลัพธ์ โดยคำตอบที่เป็นเสียงส่วนใหญ่จะถูกเลือกให้เป็นคำตอบที่ถูกต้อง และ task นั้นจะถูกเปลี่ยนสถานะเป็น validated ถ้าหากไม่มีคำตอบที่ซ้ำกันถึงค่าที่กำหนด ระบบจะทำการสร้าง result เพิ่มขึ้นมา หากมีจำนวน result เท่ากับค่า max result แล้วยังไม่มีตัวแทนของคำตอบที่ถูกต้องอีก validator จะปฏิเสธทุก result และเปลี่ยนสถานะของ task นั้นเป็น bury เพื่อให้ผู้ดูแลระบบได้ตัดสินใจต่อไป
- **Collector** คอยจัดการผลลัพธ์ที่ได้ผ่านการตรวจสอบคำตอบแล้ว โดยวิธีการจัดการกับไฟล์คำตอบนั้นจะขึ้นกับการตั้งค่าของ job นั้นๆ ซึ่งโดยพื้นฐานแล้วโมดูลนี้จะเลือกเก็บตัวแทนคำตอบและลบไฟล์คำตอบอื่นๆที่ซ้ำซ้อนออกไป
- **Task Manager** ทำหน้าที่คอยติดตามสถานะและดำเนินการกับ job นั้นๆ เช่น หาก job เสร็จสมบูรณ์แล้ว โมดูลนี้จะเริ่ม job ถัดไปตาม workflow และเริ่มสร้าง task ของ job นั้นๆตามจำนวนของอินพุตไฟล์ที่ได้ตั้งค่าไว้

โครงสร้างของระบบนี้ โมดูลแต่ละตัวจะติดต่อกันผ่านทางสถานะของ task และ result บนฐานข้อมูล โดยแต่ละโมดูลคอยเช็คหาสถานะของงานที่เกี่ยวข้องกับตัวเองบนฐานข้อมูลเป็นระยะๆ หากมีสถานะที่เกี่ยวข้องกับตนเองก็จะเริ่มทำงาน และเมื่อเสร็จสิ้นก็จะเปลี่ยนสถานะของงานเพื่อให้โมดูลอื่นจัดการต่อไป ตัวอย่างเช่น task ***t*** มี result บางส่วนที่ต้องตรวจสอบความถูกต้อง โมดูล validator ก็จะนับจำนวน result ของ task นั้นที่ไคลเอนต์ได้กรอกผลลัพธ์ลงไปแล้ว

และหาจำนวนผลลัพธ์ที่เป็นเสียงข้างมาก เมื่อหาได้แล้วก็จะเปลี่ยนสถานะของ *t* เป็น validate เมื่อโมดูล collector ตรวจพบ *t* ที่สถานะ validated แล้ว จึงทำการเขียนข้อมูลผลลัพธ์ลง storage และเปลี่ยนสถานะของ *t* เป็น complete เป็นต้น โดยความสัมพันธ์สถานะต่างๆของ task นั้นสามารถสรุปได้ดังรูปที่ 11



รูปที่ 11 ความสัมพันธ์ของสถานะต่างๆของ task

บทที่ 5
แนวทางการทำให้เกิดผล (implementation)

ในบทนี้เป็นการนำแนวคิดการออกแบบระบบจากบทที่แล้วมาทำให้เกิดผล โดยการนำแนวคิดดังกล่าวไปทำเป็นเฟรมเวิร์กที่มีชื่อว่า WebGrid.js ซึ่งในบทนี้แบ่งออกเป็นสองส่วน ได้แก่ ส่วนแรกจะเป็นการอธิบายถึงแนวทางการทำให้เกิดผลในแต่ละประเด็น และส่วนที่สองเป็นการอธิบายถึงเครื่องมือที่ใช้ในงานวิจัย

5.1 แนวทางการทำให้เกิดผลในด้านต่าง ๆ

งานวิจัยนี้ได้นำแนวคิดการออกแบบระบบจากบทที่แล้ว มาทำให้เกิดผลโดยการนำแนวคิดดังกล่าวมาทำเป็นเฟรมเวิร์กต้นแบบ (prototype) ที่มีชื่อว่า WebGrid.js โดยแบ่งงานได้ออกเป็น 5 ส่วนดังนี้

5.1.1 โครงสร้างของไคลเอนต์และการเก็บข้อมูลของไคลเอนต์

โปรแกรมสำหรับไคลเอนต์ในงานวิจัยนี้ถูกเขียนด้วยภาษา JavaScript ทั้งหมด โดยใช้ web worker เพื่อแก้ปัญหาเทรดเดียวของ JavaScript และใช้ localStorage ซึ่งเป็นมาตรฐานใหม่ของ HTML5 ในการเก็บข้อมูลบนฝั่งไคลเอนต์ โดย WebGrid.js จะมีรูปแบบ worker ของไคลเอนต์อยู่ 2 ประเภทคือ แบบปกติ (ผู้พัฒนาใส่รหัสโปรแกรมและขั้นตอนการทำงานต่างๆ รวมไปถึงการสร้างจุดตรวจสอบ (checkpoint) ด้วยตนเอง) และแบบ MapReduce (ดูรายละเอียดเพิ่มเติมได้ในหัวข้อ 5.1.3)

WebGrid.js จะมีการเก็บข้อมูลรายละเอียดของเครื่องไคลเอนต์ไว้ โดยข้อมูลเหล่านี้สามารถนำไปใช้ประโยชน์ในภายหลังได้ (เช่น การทำสถิติเพื่อการวางแผนแจกจ่ายงานให้กับไคลเอนต์) เนื่องจากเว็บเบราว์เซอร์นั้นใช้ระบบสภาพแวดล้อมกระบะทรายทำให้ระบบเฟรมเวิร์กนี้ไม่สามารถเก็บข้อมูลในระดับฮาร์ดแวร์ของเครื่องไคลเอนต์ได้(เช่น ซีพียูที่ใช้ เป็นต้น) ดังนั้นการเก็บข้อมูลของไคลเอนต์ในระบบนี้จึงเลือกใช้วิธีเก็บจากข้อความ user agent ซึ่งเป็นชุดข้อความบอกรายละเอียดเบื้องต้นของเครื่องไคลเอนต์ (ตัวอย่างเช่น Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.43 Safari/537.31 ซึ่งเป็น user agent ของ Google Chrome 26 บนระบบปฏิบัติการ Windows 7 64 bit) โดย WebGrid.js เลือกเก็บข้อมูล แพลตฟอร์มที่ใช้, ข้อมูลผู้ทำเว็บเบราว์เซอร์นั้น (vendor), ระบบปฏิบัติการที่ใช้, เลขที่อยู่ไอพี, timezone นอกจากนี้ ยังมีการวัดประสิทธิภาพเบื้องต้น(ใช้

วิธีการวัดประสิทธิภาพเดียวกับหัวข้อ 3.1.1.1) และเก็บเป็นค่า FLOPS (floating point operations per second) ของเครื่องไคลเอนต์นั้นเก็บไว้อีกด้วย

WebGrid.js สามารถรองรับการเก็บข้อมูลไคลเอนต์ทั้งหมด 5 เว็บเบราว์เซอร์ ได้แก่ Chromium, Google Chrome, Mozilla Firefox, Apple Safari และ Microsoft Internet Explorer เพื่อให้ผู้ใช้งานสามารถเข้าร่วมโครงการได้ทันที WebGrid.js นั้นไม่มีการสมัครบัญชีผู้ใช้ โดยระบบสามารถลงทะเบียนผู้ใช้งานและเก็บข้อมูลผู้ใช้งานลง localStorage ทันทีที่ผู้ใช้เข้าหน้าเว็บที่กำหนดไว้ เนื่องจากข้อมูลผู้ใช้งานนั้นอ้างอิงกับเว็บเบราว์เซอร์ที่ใช้ ดังนั้นหากผู้ใช้งานเปลี่ยนเว็บเบราว์เซอร์ ระบบจะถือว่าเป็นผู้ใช้งานคนใหม่ ในกรณีที่ผู้ใช้งานทำการล้างข้อมูลแคชของระบบข้อมูลของผู้ใช้งานที่เก็บบนเว็บเบราว์เซอร์นั้นจะหายไป และระบบจะทำการลงทะเบียนรหัสผู้ใช้งานใหม่เมื่อผู้ใช้เข้าระบบครั้งต่อไป ซึ่งหากมีการนำเฟรมเวิร์กนี้ไปใช้งานจริงก็อาจจะเพิ่มในส่วนของระบบการสมัครบัญชีผู้ใช้เพิ่มเติมได้ (WebGrid.js สามารถปรับให้ไม่เก็บข้อมูลไคลเอนต์ได้ โดยการปรับค่า host_enable เป็น false เพื่อยกเลิกระบบเก็บข้อมูลไคลเอนต์ ทั้งนี้การยกเลิกการเก็บข้อมูลไคลเอนต์จะทำให้ไม่สามารถใช้งานในส่วนของจุดตรวจสอบ (checkpoint) ได้)

5.1.2 ระบบจุดตรวจสอบ (checkpoint)

เนื่องจากพฤติกรรมของไคลเอนต์ในการประมวลผลแบบอุทิศนั้นจะสามารถออกได้ตลอดเวลา ทำให้เกิดความสูญเปล่าในกรณีที่งานนั้นต้องใช้เวลาในการประมวลผลนาน ระบบการประมวลผลแบบอุทิศจึงควรที่จะมีระบบจุดตรวจสอบไว้ เพื่อให้ไคลเอนต์สามารถกลับมาทำงานที่ค้างต่อได้ WebGrid.js จะมีวิธีการ implement ระบบจุดตรวจสอบดังนี้

เมื่อไคลเอนต์มีการเรียกของาน โมดูล scheduler ทำการจองงานสำหรับไคลเอนต์นั้นไว้ เพื่อไม่ให้โมดูล scheduler แจกงานนี้ให้ไคลเอนต์อื่น พร้อมทั้งใส่ค่าเวลา deadline เอาไว้ หากไคลเอนต์ที่จองงานนั้นไม่สามารถส่งผลลัพธ์กลับมาภายในเวลาที่กำหนด (ค่าเวลาปัจจุบันเลยค่าเวลา deadline ที่ได้กำหนดไว้) งานนั้นจะถูกปลดออกเพื่อให้โมดูล scheduler แจกจ่ายงานนี้ให้ไคลเอนต์อื่นต่อไป

เนื่องจากพื้นที่ของ localStorage ที่มีจำกัด (2-5 MB) ทำให้ระบบต้องเลือกเก็บเฉพาะข้อมูลที่จำเป็นเท่านั้น โดยระบบ WebGrid.js นั้นจะเลือกเก็บเฉพาะ เวลาที่ทำจุดตรวจสอบล่าสุด (last_check), เวลาที่ใช้ประมวลผลทั้งหมด (total_time), หมายเลขงาน (id), เวลาที่ใช้ในการดึงข้อมูลงาน (task_fetch), เวลาที่ใช้ในการดาวน์โหลดไฟล์อินพุต (input_fetch) และข้อมูลจุดตรวจสอบของการดำเนินงานนั้นๆ (ส่วนนี้ผู้พัฒนาสามารถเลือกเก็บข้อมูลได้เอง) ส่วนข้อมูลอื่นๆ เช่น ข้อมูลงานหรือไฟล์อินพุตนั้น ไคลเอนต์ต้องดาวน์โหลดข้อมูลเหล่านั้นมาจากเซิร์ฟเวอร์ใหม่อีก

รอบตามงานที่ได้จองเอาไว้ ผู้พัฒนาสามารถใช้งานระบบจุดตรวจสอบโดยใช้คำสั่ง load_checkpoint และ save_checkpoint ตามลำดับ

5.1.3 การประยุกต์ใช้ MapReduce ใน WebGrid.js

MapReduce เป็นรูปแบบโปรแกรมสำหรับใช้ในการประมวลผลข้อมูลขนาดใหญ่ โดยรูปแบบนี้ช่วยลดความซับซ้อนในการทำโปรแกรมแบบขนานให้เหลือเพียง 2 ฟังก์ชั่นหลักได้แก่ map และ reduce รูปแบบโปรแกรมนี้สามารถนำไปใช้งานได้หลากหลาย[37] แม้ว่างานนั้นอาจไม่ได้เป็นงานที่เน้นด้านการประมวลผลข้อมูล (data-intensive application) ก็ตาม ตัวอย่างเช่น ขั้นตอนวิธีเชิงพันธุกรรม (genetic algorithm)[38], โครงข่ายประสาทเทียม (neural network)[39], การแยกตัวประกอบของเลขจำนวนเต็ม (integer factorization)[40], ฯลฯ

WebGrid.js นั้นได้ถูกออกแบบให้รองรับการเขียนโปรแกรมแบบ MapReduce ได้ โดยรายละเอียดของแนวทางการทำให้เกิดผลมีดังนี้

5.1.3.1 รูปแบบ MapReduce ใน WebGrid.js

เนื่องจาก[37] พบว่ารูปแบบ MapReduce ดั้งเดิมนั้นได้ถูกกำหนดรูปแบบขั้นตอนการ map แล้วตามด้วย reduce ตายตัวเท่านั้น ซึ่งมีบางงานที่ไม่เข้ารูปแบบนี้เสมอไป เช่น งานเกี่ยวกับการหาข้อมูลที่อาจต้องการแค่ขั้นตอน map อย่างเดียว หรืองานบางอย่างที่ต้องการขั้นตอน reduce เพิ่มเพื่อหาผลรวมของคำตอบสุดท้ายอีกครั้งหนึ่ง ประกอบกับบางงานที่อาจมีความต้องการ MapReduce มากกว่า 1 ครั้ง การตั้งค่าให้ MapReduce ทำงานต่อเนื่องด้วยตนเองก็อาจเกิดความยุ่งยากได้ในบางครั้ง ซึ่งในปัจจุบันก็มีเฟรมเวิร์กที่เข้ามาช่วยทำให้งานส่วนนี้ง่ายขึ้น เช่น Cascading[41] ที่นำเสนอรูปแบบที่ดัดแปลงจาก MapReduce เดิมให้มีความยืดหยุ่นสำหรับงานที่เน้นการประมวลผลข้อมูลมากขึ้น (เช่น เพิ่มระบบ workflow เข้ามา) เป็นต้น

เพื่อเพิ่มความยืดหยุ่นในการเขียนโปรแกรมมากขึ้น รูปแบบ MapReduce ใน Webgrid.js จึงการมีการดัดแปลงจากรูปแบบดั้งเดิมเล็กน้อย โดยการเพิ่มการเก็บข้อมูลแบบ key-value และแบ่งรูปแบบการจัดเก็บคำตอบเป็น 2 แบบได้แก่ แบบ store และแบบ partition ซึ่งรูปแบบ store นั้นจะเป็นการเก็บคำตอบแบบปกติ (ข้อมูลแบบ key-value หรือรูปแบบที่ผู้พัฒนากำหนดขึ้นเอง) ส่วนรูปแบบ partition นั้น คำตอบได้ถูกแบ่งกลุ่ม (partition) ก่อนจัดเก็บลงเซิร์ฟเวอร์ (รายละเอียดเพิ่มเติมในหัวข้อ 5.1.3.2 และ 5.1.3.3)

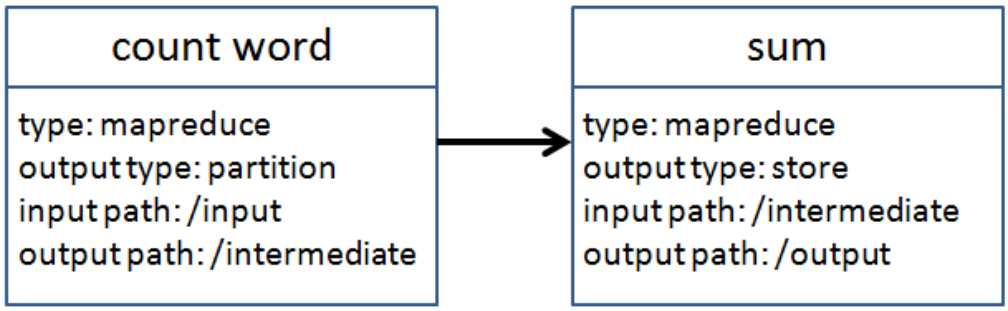
เพื่อให้ง่ายต่อการนำเสนอรูปแบบการเขียนโปรแกรม MapReduce บน WebGrid.js จึงขอยกตัวอย่างโปรแกรม wordcount (โปรแกรมนับคำว่าในข้อมูลนั้นมีคำนี้ทั้งหมดกี่คำ) ในรูปแบบ MapReduce ปกติเทียบกับรูปแบบ MapReduce บน WebGrid.js

รูปแบบโปรแกรม wordcount แบบ MapReduce ปกติ สามารถแสดงเป็นรหัสโปรแกรมได้ดังนี้

```
Map(String key, String value):
    // key : ชื่อของไฟล์ input
    // value : ข้อมูลในไฟล์ input นั้นๆ
    for each word w in value:
        emit( w, 1);
Reduce(String key,List values):
    //key : คำ
    //values : ค่าของ 1
        Int count = sum(values);
        emit(count);
```

ขั้นตอน map เป็นการอ่านค่าไฟล์อินพุตทีละบรรทัด, สกัดคำแล้วสร้างค่าข้อมูล key-value โดยการเอาคำเป็น key แล้วค่า value เป็น 1 สำหรับขั้นตอน reduce นั้นข้อมูลที่มีค่า key เหมือนกันโดยนำค่า value มารวมกัน (เป็น 1,1,1..) ดังนั้น reduce จึงทำการรวมค่าทั้งหมดมาตอบเป็นคำตอบสุดท้าย

สำหรับโปรแกรม wordcount บน WebGrid.js สามารถแสดงได้เป็นผังงานได้ดังรูปที่ 12



รูปที่ 12 โปรแกรม wordcount บน WebGrid.js

โปรแกรมนี้สามารถแสดงได้ด้วย workflow ที่ประกอบไปด้วย 2 job คือ count word และ sum โดยทั้งสองงานคล้ายกับ Mapper และ Reducer ใน MapReduce โดยงานแรกนั้นจะเป็นการนับคำและสร้างกลุ่มของค่า key-value ที่มีค่า key เป็นคำ และ value เป็นความถี่ที่เจอ ส่วน job sum นั้นจะเป็นการรวมค่าความถี่ของคำๆนั้น ค่า output type ของงานแรกต้องถูกตั้งเป็น partition เพื่อให้ทำการแบ่งส่วนของคำตอบในงานแรก และตั้งให้ input path ของงานที่สองนั้นตรงกับ output path ของงานแรก เพื่อให้งานที่สองเอาผลลัพธ์ที่ได้จากงานแรกมาใช้ต่อ

5.1.3.2 การดัดแปลงให้รองรับรูปแบบ MapReduce ในฝั่งไคลเอนต์ (mapreduce worker)

WebGrid.js นั้นมีรูปแบบโปรแกรมไคลเอนต์อยู่ 2 แบบ คือแบบปกติ (custom) ซึ่งผู้พัฒนาเขียนการดำเนินงานทั้งหมดด้วยตนเอง โดยใช้ API ที่ได้กำหนดไว้ กับแบบ MapReduce การเลือกใช้รูปแบบนั้นสามารถตั้งค่าได้ job ให้เป็นงานประเภท MapReduce ได้ที่ type ซึ่งการใช้รูปแบบ MapReduce นั้น ผู้พัฒนาจำเป็นต้องตั้งค่า job ในส่วนต่อไปนี้ (รายละเอียดของการตั้งค่า job ทั้งหมดสามารถดูได้ที่ ภาคผนวก ข)

- Input reader เป็นโปรแกรมภาษา JavaScript สำหรับการอ่านไฟล์อินพุตในแต่ละไฟล์ และแปลงค่าดังกล่าวเป็นข้อมูลแบบ key-value โดยค่าพื้นฐานคือ อ่านไฟล์ทีละบรรทัดและแปลงเป็น key-value ที่มีค่า value เป็นข้อความบรรทัดนั้น
- Algorithm เป็นโปรแกรมภาษา JavaScript สำหรับแปลงค่า key-value ใหม่จากค่า key-value ที่เป็นอินพุต ผู้พัฒนาสามารถสร้างค่า key-value ใหม่ได้โดยการใช้คำสั่ง emit
- Combiner เป็นโปรแกรมภาษา JavaScript สำหรับรวมค่าของ value ที่มี key เหมือนกัน เพื่อลดขนาดของผลลัพธ์ก่อนที่จะส่งกลับไปยังเซิร์ฟเวอร์
- Output type เป็นประเภทของผลลัพธ์ ซึ่งในเฟรมเวิร์กนี้ได้กำหนดไว้ 2 ประเภทคือ store และ partition โดย store (ซึ่งเป็นค่าพื้นฐาน) คือการเก็บผลลัพธ์แบบกลุ่ม key-value ส่วน partition นั้น ผลลัพธ์ที่ได้จะถูกแบ่งกลุ่มตามจำนวนค่าพารามิเตอร์ partition number ที่ได้ถูกตั้งค่าไว้
- Partition number เป็นจำนวนกลุ่มที่ต้องการแบ่ง การทำ partition บนฝั่งไคลเอนต์นั้นจะใช้สูตร hash(key) % partition number โดยฟังก์ชั่น

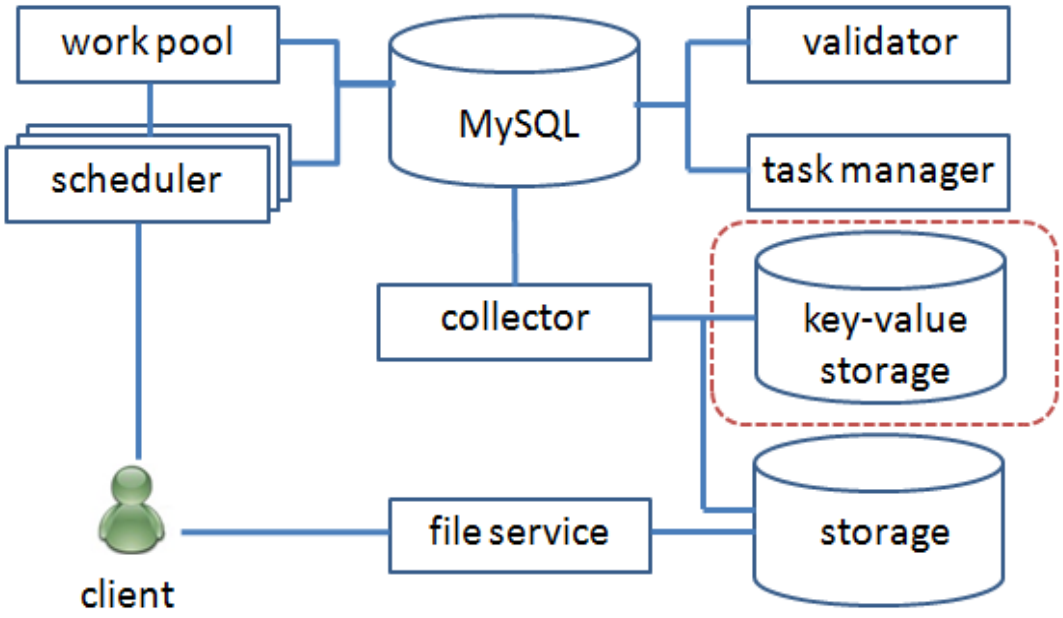
แฮชที่ใช้คือ MurmurHash3[42] ซึ่งเป็นฟังก์ชั่นแฮชที่ทำงานได้เร็วและมีการกระจายตัวที่ดี[43]

โดยโปรแกรม wordcount จากหัวข้อที่แล้วสามารถสร้างได้ดังนี้

Job: count word
Output type: partition
Input reader: default (อ่านไฟล์อินพุตทีละบรรทัด)
Algorithm:
function(key,value){
var words = value.split(‘ ‘);
for(var i=0;i<words.length;i++)
emit(words[i],1);
}
Combiner: รวมค่า value ที่มีค่า key เหมือนกัน
	Job: sum
	Output type: store
	input reader: default (อ่านไฟล์อินพุตทีละบรรทัด)
	Algorithm:
	function(key,value){
	var count = 0;
	for(var i=0;i<value.length;i++)
	count += value[i];
	emit(key,count);
	}
	Combiner: ไม่มี

5.1.3.3 การเก็บข้อมูลประเภท key-value ทางฝั่งเซิร์ฟเวอร์

เพื่อให้เฟรมเวิร์กรองรับการจัดเก็บข้อมูลประเภท key-value บนฝั่งเซิร์ฟเวอร์ จึงได้มีการเปลี่ยนสถาปัตยกรรมจากบทที่แล้ว เป็นดังรูปที่ 9

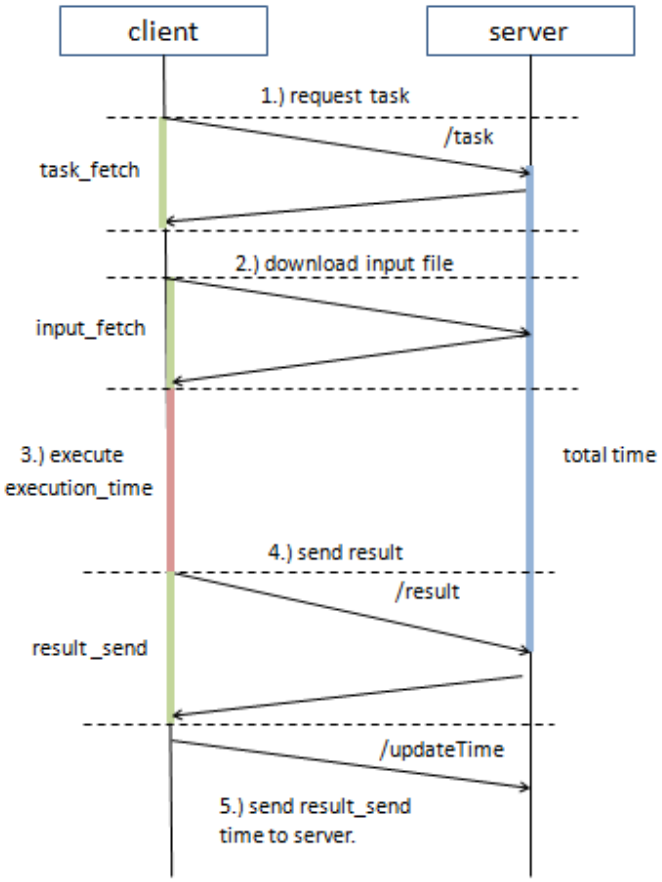


รูปที่ 13 สถาปัตยกรรมของระบบที่มีการเพิ่มให้รองรับรูปแบบ MapReduce

จากรูปที่ 13 พบว่ามีการเพิ่มในส่วนของฐานข้อมูลแบบ key-value (kvs) เข้ามา (ในส่วนที่มีวงกลมเส้นประสีแดง) โดยขั้นตอนการดำเนินงานที่เพิ่มเข้ามาคือ โมดูล collector ที่เมื่อพบข้อมูลจากงานที่มี output type เป็น partition นั้นก็ให้เก็บข้อมูลลง kvs แทนที่จะเขียนไฟล์แบบปกติ สาเหตุที่เลือกใช้ kvs นั้นเพราะ ในขั้นตอนการแบ่งส่วนและการจัดกลุ่มข้อมูลนั้นใช้หน่วยความจำมาก ประกอบกับการเก็บข้อมูล key-value ลงไฟล์นั้นมีความยุ่งยากต่อการแก้ไขและค้นหาข้อมูล ซึ่งการใช้ kvs เข้ามาช่วยจัดการนั้นจะมีประสิทธิภาพมากกว่า

5.1.4 การเก็บข้อมูลเวลาของไคลเอนต์

เพื่อประโยชน์ในการศึกษาและพัฒนาระบบ WebGrid.js จึงเพิ่มเรื่องการเก็บเวลาที่ไคลเอนต์ใช้ในการดำเนินงานต่างๆได้แก่ เวลาที่ใช้ทั้งหมดในการประมวลงาน (execution_time), เวลาที่ใช้ในการร้องของงาน (task_fetch), เวลาที่ใช้ในการดาวน์โหลดข้อมูลอินพุต (input_fetch) และ เวลาที่ใช้ในส่งผลลัพธ์กลับไปยังเซิร์ฟเวอร์ (result_send) โดยวิธีการเก็บข้อมูลเวลานั้นได้แสดงดังรูปที่ 10



รูปที่ 14 รายละเอียดข้อมูลเวลาที่ไคลเอนต์ใช้ในการดำเนินงาน

รูปที่ 14 นั้นมีความคล้ายคลึงกับขั้นตอนในรูปที่ 9 แต่มีการเพิ่มขั้นตอนสุดท้ายเข้ามาคือ ไคลเอนต์ส่งข้อมูล result_send ไปให้เซิร์ฟเวอร์ โดยการเก็บข้อมูลทางด้านการสื่อสารนั้นเป็นการวัดค่าที่ฝั่งไคลเอนต์ทั้งหมด เพราะเวลาบนเครื่องไคลเอนต์และเซิร์ฟเวอร์นั้นไม่เท่ากัน (จากการทดลองพบว่าเวลาคลาดเคลื่อนในช่วง 300 – 4000 ms) ทำให้การเก็บข้อมูลเวลานั้นควรยึดเพียงฝั่งเดียวเท่านั้น ทั้งนี้เวลา total time (ฝั่งขวา) นั้นสามารถคำนวณได้จากเวลา timestamp ที่เซิร์ฟเวอร์จ่ายงาน และเวลาที่เซิร์ฟเวอร์ได้รับผลลัพธ์จากไคลเอนต์นั้น

5.1.5 การตรวจสอบคำตอบและการเก็บผลลัพธ์แบบปรับแต่งได้ (custom validator & custom collector)

WebGrid.js นั้นมีการตรวจสอบคำตอบโดยการเทียบ MD5 checksum ซึ่งวิธีนี้อาจไม่เหมาะสมกับงานบางประเภทที่คำตอบที่ได้จากแต่ละไคลเอนต์ไม่เท่ากันได้ เพื่อเพิ่มความยืดหยุ่นของระบบ WebGrid.js ได้ออกแบบให้ผู้พัฒนาสามารถเขียนวิธีการตรวจสอบคำตอบและวิธีการจัดเก็บเองได้ ทั้งนี้ผู้พัฒนาต้องเขียนให้อยู่ในรูปแบบที่กำหนดเอาไว้

5.2 เครื่องมือที่ใช้ในงานวิจัย

เครื่องมือที่ใช้ในงานวิจัยนี้ ประกอบด้วย

5.2.1 แพลตฟอร์มที่ใช้ในการพัฒนาระบบในฝั่งเซิร์ฟเวอร์ : Node.js [7]

เป็นแพลตฟอร์มฝั่งเซิร์ฟเวอร์ตัวหนึ่งที่ใช้ภาษา JavaScript โดย Node.js ใช้ Event-driven และ non-blocking I/O model ทำให้ใช้ทรัพยากรของเครื่องน้อยกว่า, สามารถรองรับ connection ได้เป็นจำนวนมากและสามารถตอบสนอง request ได้ไว[44] ประกอบกับชุมชนผู้ใช้งาน Node.js ที่มีขนาดใหญ่ทำให้มีคลังคำสั่งหรือโมดูลให้เลือกใช้จำนวนมาก จากเหตุผลข้างต้นทำให้ Node.js เป็นแพลตฟอร์มที่เหมาะสมสำหรับโปรแกรมทางด้านเครือข่าย (network program) สำหรับโมดูลเสริมที่ใช้งานวิจัยมีดังนี้

- expressjs ช่วยในการทำให้การสร้าง restful web service
- node-mysql คลังคำสั่งที่ใช้ในการติดต่อกับฐานข้อมูล MySQL
- request ช่วยในการสร้าง HTTP request ในการติดต่อกับโมดูลอื่นๆของ WebGrid.js
- forever เป็นโมดูลที่ช่วยเฝ้าสังเกต (monitor) โปรเซสของ Node.js หากมีโปรเซสล่ม โมดูลนี้จะเริ่มการทำงานโปรเซสนั้นใหม่ให้ทันที

5.2.2 ฐานข้อมูลของระบบ : MySQL

ฐานข้อมูลสำหรับเก็บข้อมูลรายละเอียดของงานต่างๆ โดยในงานวิจัยนี้จะเลือกใช้ engine InnoDB ในการเก็บข้อมูล

5.2.3 ฐานข้อมูลสำหรับข้อมูลประเภท key-value : Kyoto Tycoon[45]

ฐานข้อมูลสำหรับการเก็บข้อมูล key-value ที่เลือกใช้ในงานวิจัยนี้ คือ Kyoto Tycoon โดย Kyoto Tycoon นั้นเป็นฐานข้อมูลที่พัฒนาขึ้นจาก Kyoto Cabinet โดยการเพิ่มชั้นของเครือข่ายเข้าไป (Kyoto Cabinet เป็นฐานข้อมูลแบบฝังตัว (embedded database) ซึ่งความสัมพันธ์ระหว่าง Kyoto Cabinet กับ Kyoto Tycoon นั้นจะเหมือนกับความสัมพันธ์ของ SQLite กับ MySQL) Kyoto Cabinet จัดเป็นฐานข้อมูลแบบ key-value ที่เร็วที่สุด[46] แต่งานวิจัยนี้ได้เลือกใช้ Kyoto Tycoon เพื่อรองรับการขยายตัวของระบบและขนาดข้อมูลในอนาคต เพราะจากงานวิจัย[47] พบว่าการใช้ฐานข้อมูลแบบฝังตัวในเครื่องๆเดียว แม้จะสามารถใส่ข้อมูลได้เร็วกว่าแต่อาจเกิดปัญหาคอขวดจากหน่วยเก็บข้อมูลได้

บทที่ 6
วิธีประเมินการวิจัย สรุปและอภิปรายผล

ในบทนี้กล่าวถึงวิธีการประเมินวิจัย สรุปและอภิปรายผลการทดลองที่ได้ โดยในงานวิจัยนี้มีวิธีการประเมินเฟรมเวิร์กเป็น 2 เรื่อง คือการทดลองวัดประสิทธิภาพเบื้องต้น และการทดลองนำเฟรมเวิร์กไปทดลองใช้งานจริง ในส่วนสุดท้ายของบทนี้จะเป็นการสรุปการอภิปรายผลในด้านต่างๆของระบบ

6.1 การทดลองวัดผลประสิทธิภาพเบื้องต้นของเฟรมเวิร์ก

ในส่วนนี้เป็นการทดลองวัดผลประสิทธิภาพของเฟรมเวิร์กที่ได้สร้างขึ้น (WebGrid.js) เทียบกับ BOINC ซึ่งเป็นระบบที่มีอยู่แล้ว เพื่อศึกษาถึงประสิทธิภาพของระบบที่สร้างขึ้นและปัจจัยต่างๆที่ส่งผลต่อประสิทธิภาพของระบบนั้นๆ

6.1.1 ขั้นตอนการทดลอง

การทดลองนี้จะทดลองสร้าง โปรแกรมการจำลองการสุ่มแบบมอนติคาร์โลเพื่อแก้ปริพันธ์จำกัดเขต (monte carlo integration) โดยมอนติคาร์โลเป็นอัลกอริทึมเชิงสุ่มชนิดหนึ่ง โดยอัลกอริทึมประเภทนี้มีจุดเด่นคือ ง่ายและทำงานได้เร็ว มีเวลาการทำงานที่แน่นอน แต่อาจจะให้คำตอบที่ผิดได้ ทั้งนี้ความแม่นยำของคำตอบขึ้นกับจำนวนครั้งที่สุ่มด้วย ยิ่งถ้ามีจำนวนครั้งที่สุ่มมาก โอกาสที่จะได้คำตอบที่ถูกต้องก็จะมีมากขึ้น การทดลองนี้ได้นำตัวอย่างอัลกอริทึมมาจากตัวอย่างโปรแกรมของ Enis Siniksaran[48] ซึ่งเป็นโปรแกรมตัวอย่างจากเว็บ wolfram โดยโปรแกรมนี้จะแสดงการหาค่าของอินทิกรัล (integral) โดยใช้อัลกอริทึมมอนติคาร์โล สาเหตุที่เลือกอัลกอริทึมมอนติคาร์โลนั้นเพราะ อัลกอริทึมนี้สามารถแยกทำงานกันได้และเพื่อให้งานที่ทำงานในแต่ละเครื่องนั้นมีความใกล้เคียงกันมากที่สุด

โปรแกรมที่ใช้ในการทดลองนี้เป็นการทดลองหาค่าอินทิกรัลของ f= e^x โดยทำการสุ่มตัวอย่างทั้งหมด 1,000,000ครั้ง (n = 1,000,000) ไฟล์อินพุทแต่ละไฟล์ประกอบด้วยข้อความ 5 บรรทัด แต่ละบรรทัดจะเป็นค่า a,b ซึ่งเป็นช่วงของอินทิกรัล และค่า a กับ b จะเป็นเลขจำนวนจริงมีค่าระหว่าง -10 ถึง 10 และมีทศนิยม 6 ตำแหน่ง

การทดลองนี้มีจุดประสงค์เพื่อวัดหาค่าปริมาณงาน (throughput) ของ WebGrid.js กับ BOINC ที่จำนวนเครื่อง 1,3,5 และ 10 เครื่องตามลำดับ โดยรายละเอียดของสภาพแวดล้อมที่ใช้ในการทดลองมีดังนี้

เครื่องเซิร์ฟเวอร์ที่ใช้ในการทดลองคือ Dell OptiPlex 755 โดยมีรายละเอียดดังนี้

- CPU: Intel Core 2 Duo E6750 Processor (2.67 GHz)
- หน่วยความจำ 4 GB
- ระบบปฏิบัติการ Ubuntu 12.04 LTS 32 bit version
- BOINC (คอมไพล์ที่ revision 25513)
- Node.js 0.8.1
- MySQL 5.5.24

เครื่องไคลเอนต์ที่ใช้ทดสอบมีรายละเอียดดังนี้

- CPU: Intel Core i5 3.20 GHz
- หน่วยความจำ 4.00 GB
- ระบบปฏิบัติการ Windows 7 service pack 1 32 bit version
- BOINC Manager 7.0.28(x86)
- Google Chrome 20.0

รายละเอียดของโปรแกรมอื่นๆ

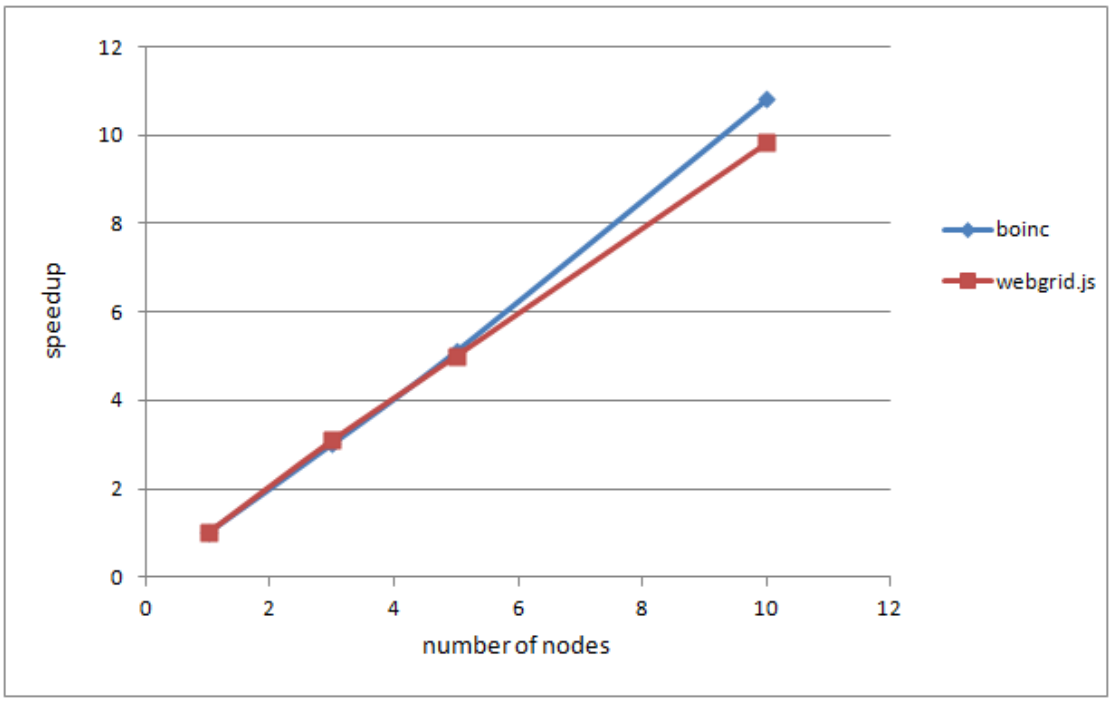
- โปรแกรมที่นำมาทดลองบน BOINC สำหรับการทดลองนี้ เป็นโปรแกรมที่มีการใช้หน่วยประมวลผลเพียงเทรดเดียว (single-threaded) และตัดในส่วนของ ระบบจุดตรวจสอบ (checkpoint) และการายงานผล (fraction done) ออกไป เนื่องจากงานที่ทดสอบในการทดลองนี้ ไม่ใช่งานที่ใช้เวลามาก (long-running job) ดังนั้นการใส่ฟังก์ชั่นดังกล่าว อาจทำให้เกิดความสิ้นเปลืองในการดำเนินการได้
- สำหรับ WebGrid.js ในการทดลองนี้ได้ปิดการทำงานในส่วนของ validator และ collector เนื่องจากโมดูลดังกล่าวไม่ได้มีประเด็นที่เกี่ยวข้องกับการทดลองนี้
- โปรแกรมไคลเอนต์ของ BOINC ในที่นี้ได้ใช้การตั้งค่าพื้นฐานทั้งหมด ยกเว้นในส่วนของการทำงานบนซีพียูหลายคอร์ ที่ในการทดลองนี้ได้ตั้งให้ทำงานบนซีพียูเพียงตัวเดียว
- ทำการวัดผลในแต่ละส่วน 3 ครั้งแล้วสรุปเป็นค่าเฉลี่ยออกมา

6.1.2 ผลการทดลองและการอภิปรายผล

ผลการทดลองเวลาที่ใช้ในการทำงาน สามารถสรุปได้เป็นตารางที่ 8 และรูปที่ 15 (speed up นั้นได้จากการเทียบกับปริมณงานที่ได้ในเฟรมเวิร์กนั้นที่จำนวน 1 เครื่อง)

ตารางที่ 8 ปริมาณงานที่ทำได้ใน 1 นาทีเปรียบเทียบระหว่าง BOINC กับ WebGrid.js ที่จำนวนเครื่อง 1,3,5 และ 10 เครื่องตามลำดับ

Nodes	BOINC			WebGrid.js		
	avg	avg/node	speedup	avg	avg/node	speedup
1	16.5	16.5	1.00	21.0	21.0	1.00
3	49.7	16.6	3.01	65.0	21.7	3.10
5	84.3	16.9	5.11	105	21.0	5.00
10	178	17.8	10.8	206	20.7	9.81



รูปที่ 15 ค่า speedup ระหว่าง BOINC กับ WebGrid.js

จากผลการทดลอง พบว่า WebGrid.js และ BOINC นั้นให้ผล speedup ค่อนข้างเป็นเส้นตรง โดย WebGrid.js นั้นจะให้ค่าปริมาณงานที่มากกว่า เนื่องจาก BOINC นั้นได้ถูกออกแบบสำหรับงานที่มีการใช้เวลาทำมาก เพื่อไม่ให้งานสูญเปล่าเมื่อไคลเอนต์ออกจากโปรแกรม จึงต้องมีการสำรองไฟล์ลงบนฮาร์ดดิสก์ ซึ่งกระบวนการนี้เสียเวลามากกว่า WebGrid.js ที่ข้อมูลถูกเก็บลง

บนหน่วยความจำ จุดที่น่าสังเกตอีกอย่างคือ BOINC นั้นให้ค่า speedup เป็น superlinear ซึ่งจากข้อสันนิษฐานเบื้องต้นประกอบกับผลการทดลองจากงานวิจัยนี้[49] คาดว่าสาเหตุมาจากโปรแกรม BOINC Manager นั้นมีการใช้งานในส่วนของ hyper-threading ของซีพียูเข้าช่วย ทำให้สามารถเกิดเหตุการณ์ super-linear ในบางกรณีได้

เนื่องจากจำนวนเครื่องที่ใช้ในการทดลองนี้ยังไม่มากพอที่จะเปลี่ยนแนวโน้มของ speedup ได้ประกอบกับการแนวทางอื่นๆในการทดลองวัดประสิทธิภาพด้านนี้ มีความซับซ้อนและใช้ทรัพยากรค่อนข้างมาก จึงขอหยุดการทดลองด้านประสิทธิภาพในเชิงปริมาณงานที่ทำ (throughput) แต่เพียงเท่านี้

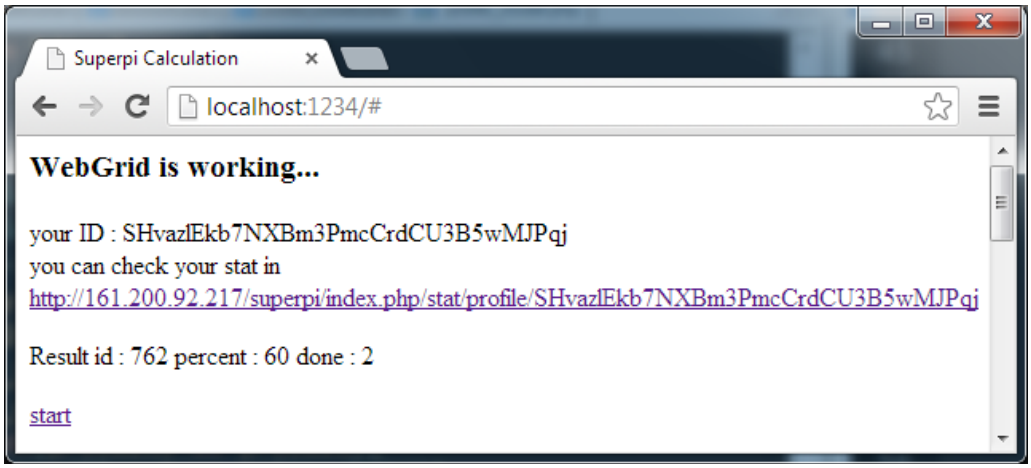
นอกจากนี้การทดลองนี้ยังมีจุดที่น่าสนใจอีกประเด็นคือ WebGrid.js นั้นให้ผล throughput ที่สูงกว่า BOINC ซึ่งหากลองดูงานวิจัยที่เกี่ยวข้องกับ BOINC ในช่วงปี 2010 – 2012 นั้นจะมีงานวิจัยหนึ่งที่พยายามเปลี่ยนแนวคิดของ BOINC ที่จากเดิมคือ high-throughput (เน้นจำนวนปริมาณที่ทำเสร็จ) เป็น real-time (เน้นให้หนึ่งงานทำเสร็จเร็วขึ้น) ตัวอย่างเช่น RT-BOINC[50] ซึ่งแนวคิดหลักคือการเปลี่ยนการเก็บข้อมูลที่แต่เดิมจะเก็บลง storage ต่างๆ เป็นการเก็บข้อมูลลงหน่วยความจำชั่วคราวแทน ซึ่งพบว่าแนวคิดนี้ส่วนหนึ่งสอดคล้องกับเว็บเบราว์เซอร์ที่เก็บข้อมูลลงหน่วยความจำเหมือนกัน ประกอบกับเว็บเบราว์เซอร์นั้นผู้เข้าร่วมสามารถเข้าร่วมได้ง่ายกว่า ทำให้แนวคิดการนำเว็บเบราว์เซอร์มาประยุกต์ใช้กับงานด้านนี้มีความน่าสนใจมากขึ้น ทั้งนี้อาจจะต้องมีการพิจารณาถึงต้นทุนทางด้านการติดต่อสื่อสาร (network latency) ด้วย

6.2 การทดลองนำเฟรมเวิร์กไปทดลองใช้งานจริง

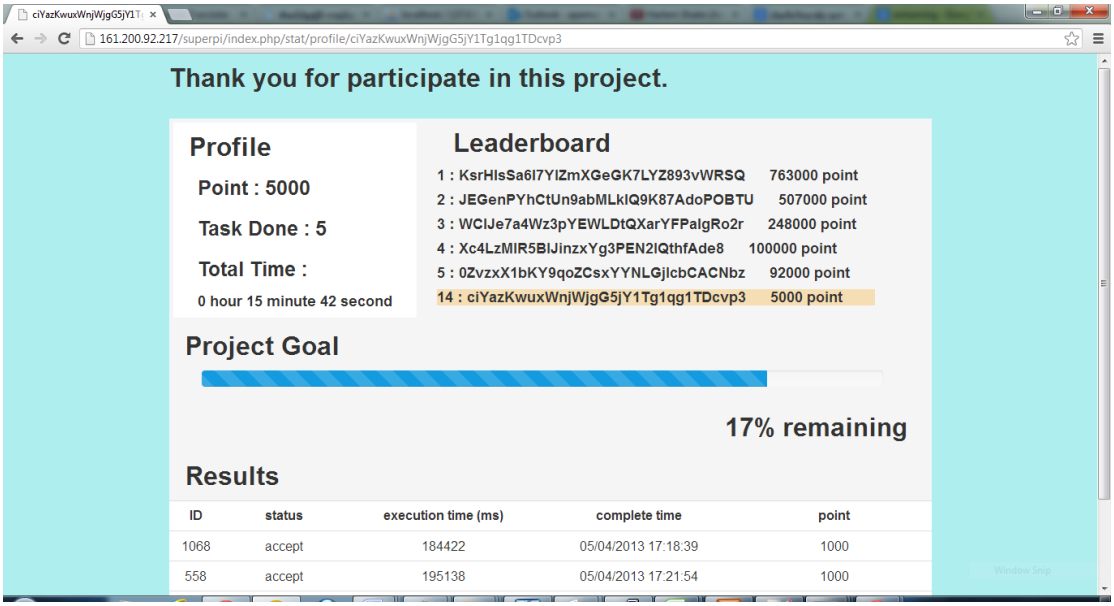
ในส่วนนี้เป็นนำเฟรมเวิร์กไปทดลองสร้างระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์ขึ้นมา พร้อมทั้งทดลองใช้กับอาสาสมัครจริงๆ โดยงานที่นำมาใช้ในการทดลองนี้มีทั้งหมด 2 งานคือ superPI และ wordcount (รายละเอียดของการสร้างโปรแกรมสำหรับทดลองบนเฟรมเวิร์กนั้นสามารถดูได้ที่ภาคผนวก ก)

รายละเอียดของเครื่องเซิร์ฟเวอร์ที่ใช้นั้นจะเหมือนกับเครื่องที่ใช้ในหัวข้อ 6.1 การทดลองนี้จะให้ประชาสัมพันธ์ให้อาสาสมัครเข้าเว็บที่กำหนดไว้ทางเครือข่ายสังคมออนไลน์ เนื่องจากเซิร์ฟเวอร์ใช้ระบบอินเทอร์เน็ตภายในจุฬาฯ ทำให้การติดต่อจากอินเทอร์เน็ตภายนอกไม่สามารถติดต่อพอร์ตอื่นๆของเซิร์ฟเวอร์นอกจากพอร์ต 80 ได้ และการใช้เครือข่ายส่วนตัวเสมือน (VPN) ก็ไม่สามารถทำได้เพราะระบบ VPN นั้นมีการแก้คำสั่ง JavaScript (ส่วนใหญ่เป็นการแก้ URL) ทำให้โปรแกรมทำงานผิดพลาดได้ จากข้อจำกัดดังกล่าวทำให้กลุ่มเป้าหมายเปลี่ยนจากกลุ่มคนทั่วไปเป็นกลุ่มนิสิตระดับบัณฑิตศึกษาแทน

เพื่อเพิ่มแรงจูงใจและทำให้โครงการดูน่าสนใจมากขึ้น จึงได้มีการเพิ่มในส่วนของการรายงานผลคะแนนขึ้นมา(ซึ่งระบบคะแนนนั้นเป็นระบบที่แยกออกจากเฟรมเวิร์กต่างหาก) โดยรูปแบบของเว็บ และหน้ารายงานผลคะแนน จะแสดงดังรูปที่ 16 และ 17 ตามลำดับ



รูปที่ 16 หน้าต่างของเว็บเบราว์เซอร์ Google Chrome เมื่อเข้าร่วมการคำนวณ



รูปที่ 17 หน้าต่างของเว็บรายงานผล

รูปที่ 17 เป็นภาพหน้าต่างแสดงหน้าตาของเว็บรายงานผล โดยแถบซ้ายจะเป็นการรายงานข้อมูลการเข้าร่วมของอาสาสมัคร แถบขวาเป็นส่วนของการแสดงผลอันดับผู้ที่บริจาคมากที่สุด 5 อันดับ พร้อมกับอันดับของอาสาสมัครปัจจุบัน ในส่วนถัดลงมาเป็นการแสดงผลความคืบหน้าของโครงการ และส่วนสุดท้าย(ด้านล่าง)เป็นการรายการงานทั้งหมดที่อาสาสมัครได้ทำ

6.2.1 superPI

6.2.1.1 **ลักษณะของงาน**

การทดลองนี้เป็นการทดลองหาค่า PI ด้วยวิธีการสุ่มแบบมอนติคาร์โล ทำการสุ่มทั้งหมด 1MM (1ล้านล้าน) รอบ เนื่องจากอัลกอริทึมนี้สามารถแยกกันทำงานได้ จึงแบ่งออกเป็นการทำงานย่อยๆเป็นงานละ 500M รอบ ดังนั้นการทดลองนี้จะมีจำนวนงานย่อย (task) ทั้งหมด 2000 งาน

เนื่องจากงานนี้เป็นงานที่เน้นด้านการประมวลผลแต่มีการส่งข้อมูลน้อย (compute-intensive application) ซึ่งจากการวิเคราะห์โดยการทดลองประมวลผลด้วยเครื่องที่มีสเปคเหมือนกับเซิร์ฟเวอร์แล้วพบว่า เวลาที่ใช้ในการประมวลผลต่อ 1 task อยู่ที่ 3 นาที ดังนั้นหากการใช้งานประมวลผลเพียงเครื่องเดียวจะใช้เวลาทั้งหมดประมาณ 4 วัน นอกจากนี้ จาก[20] ได้มีการวิเคราะห์อัตราส่วนระหว่างเวลาที่ใช้ในการคำนวณกับเวลาที่ใช้ในการเคลื่อนย้ายข้อมูลเพื่อช่วยในการตัดสินใจเบื้องต้นว่างถึงความเหมาะสมที่จะนำงานมาประยุกต์บนระบบการคำนวณบนเว็บเบราว์เซอร์ (ถ้าค่านี้มีค่าน้อยกว่า 1 มากๆ งานดังกล่าวอาจจะไม่เหมาะสมเพราะเสียเวลาในเรื่องของการเคลื่อนย้ายข้อมูลมากเกินไป) โดย WebGrid.js สามารถประยุกต์สูตรดังกล่าวได้ดังนี้ (สามารถดูรายละเอียดของค่าที่ใช้เพิ่มเติมได้ที่ รูปที่ 10)

C / R ratio = execution_time / (task_fetch+input_fetch+result_send)

- execution_time คือ เวลาที่ใช้ในการประมวลผล
- task_fetch คือ เวลาที่ใช้ในการร้องของงานจากเซิร์ฟเวอร์
- input_fetch คือ เวลาที่ใช้ในการดาวน์โหลดไฟล์อินพุต
- result_send คือ เวลาที่ใช้ในการส่งผลลัพธ์กลับ

จากการทดลองโดยเครื่องคอมพิวเตอร์ดังที่ได้กล่าวไว้ข้างต้น สามารถแทนค่าได้ดังนี้ C / L ratio = 174236 / (52+4+70) = 1382.83 พบว่างานนี้น่าจะมีความเหมาะสมในการนำมาใช้บนระบบนี้ อย่างไรก็ตามวิธีดังกล่าวเป็นเพียงวิธีการพิจารณาในเบื้องต้นเท่านั้น

6.2.1.2 **รายละเอียดและค่าที่ใช้ในการทดลอง**

รายละเอียดของการงาน superPI มีดังนี้

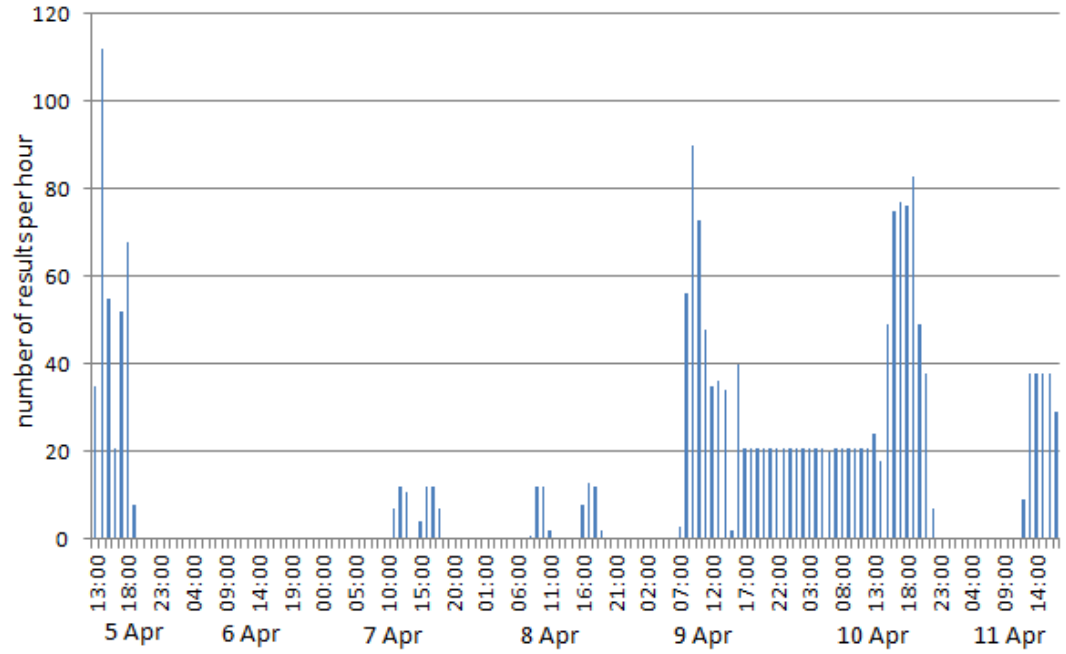
- มีจำนวน task ทั้งหมด 2,000 task
- เนื่องจากการทดลองนี้ตั้งบนสมมติฐานว่าไคลเอนต์จะไม่ทำลายระบบ ดังนั้นโมดูล validator จะให้ทุกผลลัพธ์ที่ไคลเอนต์ส่งมาถูกเสมอ และ

การทดลองนี้ตั้งค่า majority ที่ 1 ดังนั้น result ในงานนี้จะมีจำนวนเท่ากับ task หรือทั้งหมด 2,000 result

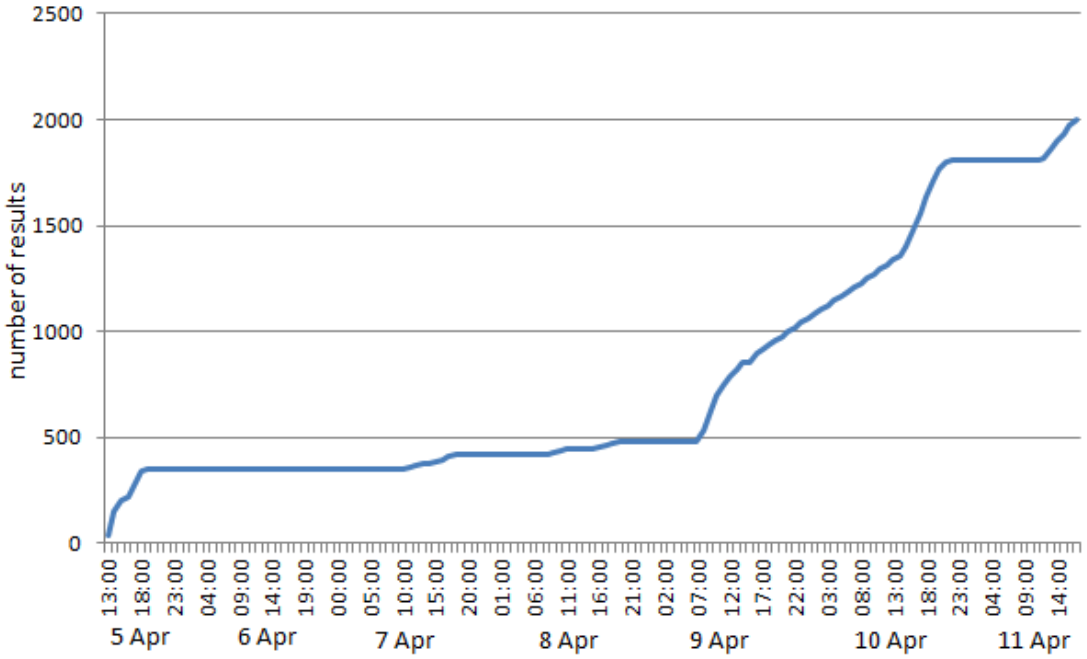
- ค่า output_type เป็น store
- มีไฟล์อินพุตเพียงไฟล์เดียวซึ่งเป็นไฟล์ที่กำหนดจำนวนรอบการสุ่มที่จะให้ไคลเอนต์ทำ ในที่นี้ตั้งค่าไว้ที่ 500M รอบ ส่วนไฟล์เอาต์พุตของงาน (เก็บค่าจำนวนครั้งที่สุ่มเจอทั้งหมด) มีจำนวนทั้งหมด 2,000 ไฟล์ (เท่ากับจำนวน task เนื่องจากการตั้งค่า output_type เป็น store)
- กำหนดค่า deadline_time ไว้ที่ 900,000 (15 นาที) ดังนั้นหากงานนั้นไคลเอนต์ที่จองอยู่ไม่ส่งกลับมายังเซิร์ฟเวอร์ภายใน 15 นาที งานนั้นจะถูกส่งให้ไคลเอนต์อื่นทำต่อไป
- ไคลเอนต์จะมีการทำจุดตรวจสอบ ทุกๆจำนวนรอบที่ 100M
- point ของอาสาสมัครแต่ละคนคิดจากจำนวนงานที่ผ่านการตรวจคำตอบแล้ว x 1000

6.2.1.3 **ผลการทดลองและการอภิปรายผล**

การทดลองนั้นเริ่มต้นในวันที่ 5 เมษายน 2556 เวลา 13:00 โดยประมาณ และสิ้นสุด(นับจากเวลาที่ไคลเอนต์ส่งผลลัพธ์งานชิ้นสุดท้ายมายังเซิร์ฟเวอร์)ในวันที่ 11 เมษายน 2556 เวลา 17:48 โดยคิดเป็นเวลาทั้งหมด 6 วัน 4 ชั่วโมง ข้อมูลแสดงจำนวนงานที่อาสาสมัครทำในรายชั่วโมง สามารถแสดงได้ดังรูปที่ 18 และรูปที่ 19



รูปที่ 18 จำนวนงานที่อาสาสมัครทำในแต่ละชั่วโมง



รูปที่ 19 จำนวน result ที่เสร็จในเวลาที่กำหนด

จากรูปที่ 18 และรูปที่ 19 พบว่าในช่วงแรกจะมีอาสาสมัครเข้าร่วมน้อย เนื่องจากกลุ่มอาสาสมัครที่ระบบรองรับได้นั้นต้องใช้อินเทอร์เน็ตภายในจุฬา ทำให้ระบบไม่มีคนเข้ามาในช่วงวันหยุดและช่วงเวลากลางคืน แต่เริ่มมีอาสาสมัครเข้ามาร่วมมากขึ้นในช่วงวันอังคารที่ 9

เมษายน (วันจันทร์เป็นวันหยุดชดเชย วันจักรี) ทั้งนี้บางช่วงเวลาเซิร์ฟเวอร์มีความขัดข้องทำให้ไม่จ่ายงานให้อาสาสมัครบางส่วนได้

ในส่วนของไคลเอนต์ที่เข้าร่วม(อาสาสมัคร 1 คนอาจจะมีได้หลายไคลเอนต์)นั้นมีอยู่ทั้งหมด 19 เครื่อง โดยจะแบ่งเว็บเบราว์เซอร์ที่ใช้เป็น Google Chrome 16 เครื่อง และ Mozilla Firefox 3 เครื่อง ใช้ระบบปฏิบัติการ Microsoft Windows 17 เครื่อง และ Linux 2 เครื่อง ตามลำดับ

สำหรับ ความสัมพันธ์ระหว่าง GFLOPS ของไคลเอนต์กับเวลาที่ใช้ประมวลผลโดยเฉลี่ยแสดงดังตารางที่ 9

ตารางที่ 9 ช่วง GFLOPS ของไคลเอนต์ และเวลาที่ใช้ประมวลผลโดยเฉลี่ย

GFLOPS	ค่าเฉลี่ยเวลาที่ใช้ในการประมวลผล (ms)	จำนวนไคลเอนต์
น้อยกว่า 0.5	471197	5
0.5 – 0.59	206025	2
0.6 – 0.69	180302	6
มากกว่า 0.7	96027	6

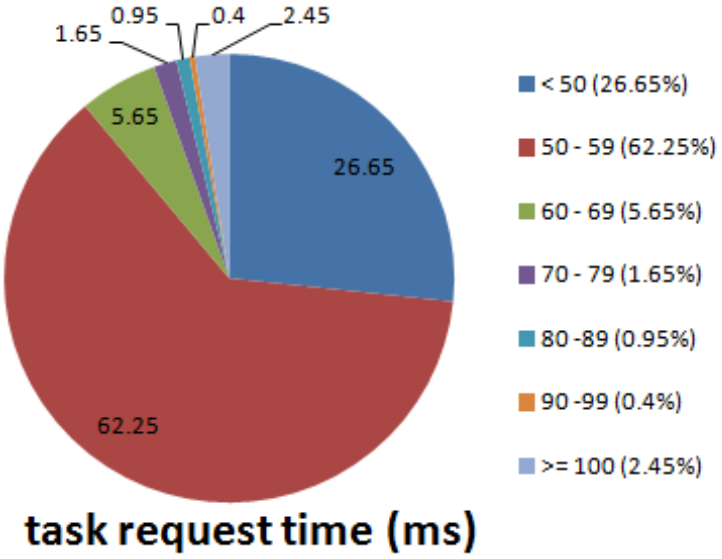
จากตารางพบว่าอาสาสมัครส่วนใหญ่จะใช้เว็บเบราว์เซอร์ที่สามารถประมวลผลได้มากกว่า 0.6 GFLOPS และใช้เวลาในการประมวลผลอยู่ในช่วงตั้งแต่ 90 – 470 วินาที โดยประมาณ

ตารางที่ 10 จำนวน result ที่ไคลเอนต์ประมวลผลเสร็จ

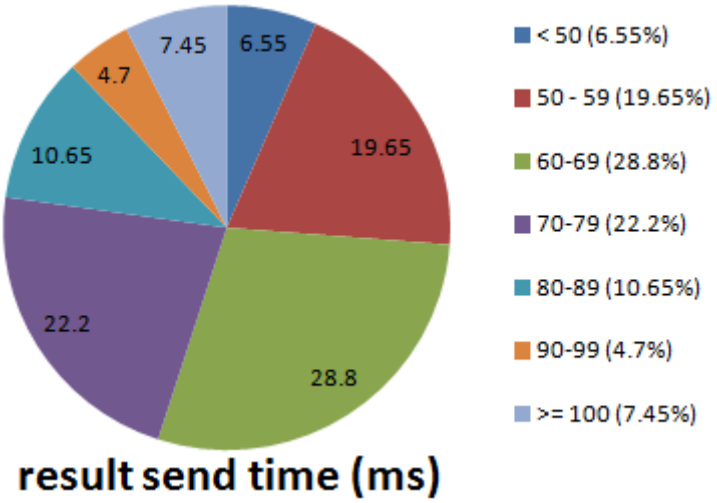
จำนวน result	จำนวนไคลเอนต์
น้อยกว่า 10	6
10 – 50	6
50 – 99	3
มากกว่า 100	4

จากตารางพบว่าไคลเอนต์ส่วนใหญ่จะประมวลผลน้อยกว่า 50 result และมีไคลเอนต์บางกลุ่มที่ช่วยประมวลผลมากกว่า 100 result

ในด้านของเวลาที่ใช้ในการติดต่อสื่อสารต่างๆ นั้นพบว่าเวลาที่ใช้ในการดาวน์โหลดไฟล์อินพุตนั้น 86.9% ของไคลเอนต์ใช้เวลาน้อยกว่า 10 ms ซึ่งสาเหตุมาจากการใช้งานเป็นการติดต่อสื่อสารภายในเครือข่ายอินเทอร์เน็ตจุฬา และไฟล์อินพุตของงานนี้มีเพียงไฟล์เดียวทำให้สามารถใช้ประโยชน์จากแคชได้ สำหรับเวลาในการร้องขอ task และเวลาในการส่งผลลัพธ์นั้นได้ถูกแสดงในรูปที่ 20 และรูปที่ 21



รูปที่ 20 เวลาที่ใช้ในการร้องของงานจากเซิร์ฟเวอร์



รูปที่ 21 เวลาที่ใช้ในการส่งผลลัพธ์กลับไปยังเซิร์ฟเวอร์

จากแผนภูมิวงกลมพบว่าเวลาที่ใช้ในการร้องของงานและเวลาที่ใช้ส่งผลลัพธ์ไปยังเซิร์ฟเวอร์น้อยกว่า 100ms เนื่องจากกลุ่มอาสาสมัครใช้อินเทอร์เน็ตภายในจุฬา และงานนี้เป็นงานที่เน้นการประมวลผลแต่ไม่เน้นเรื่องข้อมูล ทำให้ข้อมูลที่ส่งมีปริมาณน้อย โดยเวลาที่ใช้ในการร้องของงานนั้นประมาณ 50 ms เพราะเซิร์ฟเวอร์ต้องมีการจอง result นั้นสำหรับไคลเอนต์นั้นไว้ ในขณะเดียวกันเวลาที่ใช้ในการส่งผลลัพธ์ไปยังเซิร์ฟเวอร์จะใช้เวลามากกว่าเวลาร้องของงาน เพราะเซิร์ฟเวอร์ต้องเสียเวลาส่วนหนึ่งในการเขียนไฟล์คำตอบและปรับปรุงข้อมูลฐานข้อมูล

6.2.1.4 การวิเคราะห์ถึงความเหมาะสมของงาน

ปัจจัยที่ส่งผลต่อเวลาทั้งหมด (total execution time) ของงานบน WebGrid.js มีดังนี้

- ความเร็วสามารถในการประมวลของเซิร์ฟเวอร์
- ความสามารถในการประมวลผลของเครื่องไคลเอนต์แต่ละเครื่อง
- ขนาดและจำนวนงาน
- เวลาที่ใช้ในการติดต่อระหว่างไคลเอนต์กับเซิร์ฟเวอร์
- จำนวนเครื่องไคลเอนต์ในระบบ
- ปัจจัยอื่นๆ เช่น ช่วงเวลาที่ไคลเอนต์เข้าร่วม, เวลาทั้งหมดที่ไคลเอนต์บริจาคทรัพยากรให้, อายุขัยของไคลเอนต์ (ความต่อเนื่องในการเข้าร่วมของไคลเอนต์) ฯลฯ

โดยสามารถสรุป เวลาที่ใช้ทั้งหมดได้เป็นสมการเบื้องต้นดังนี้

$$t_{total} = t_{server} + t_{client}$$

เวลา t_{server} จะเป็นเวลาที่ใช้ในการประมวลผลฝั่งเซิร์ฟเวอร์ โดยจะแบ่งได้เป็น

$$t_{server} = t_{task\_create} + t_{result\_create} + t_{partition}$$

เวลา t_{task_create} คือเวลาที่เซิร์ฟเวอร์ใช้ในการสร้าง task เวลา t_{result_create} เป็นเวลาที่เซิร์ฟเวอร์ใช้ในการสร้าง result และเวลา $t_{partition}$ คือเวลาที่ใช้ในการทำการแบ่งกลุ่มข้อมูล (อ่านไฟล์และเก็บข้อมูลลงฐานข้อมูล key-value) ตามลำดับ เพื่อให้ง่ายต่อการคำนวณ ในงานวิจัยนี้จะถือว่าเวลาที่ใช้ในการทำ scheduler และเวลาที่ใช้ในการตรวจสอบข้อมูลมีผลน้อยมากจนสามารถละทิ้งได้

สำหรับเวลา t_{client} นั้นจะเป็นเวลาในส่วนของไคลเอนต์ เพื่อให้ง่ายต่อการคำนวณงานวิจัยนี้จึงละเว้นในประเด็นของช่วงเวลาที่ไคลเอนต์เข้าร่วม (ไคลเอนต์แต่ละคนเข้าร่วมไม่พร้อมกัน) และถือว่าไคลเอนต์ทุกเครื่องเข้าพร้อมกัน โดยสามารถสรุปเป็นสมการเบื้องต้นได้ดังนี้

$$t_{client} = \max\{t_i\}; i = 1,2,3, \dots, n$$

$$\mathrm{t_i} = \sum_{k=1}^{m} (\mathrm{t_{task_k}} + \mathrm{t_{input_k}} + \mathrm{t_{execution_k}} + \mathrm{t_{result_k}})$$

เนื่องจากไคลเอนต์แต่ละเครื่องสามารถทำงานแบบขนานได้ ดังนั้นเวลาในฝั่งไคลเอนต์นั้นจะเท่ากับ เวลาของไคลเอนต์เครื่องที่ใช้เวลาทำงานมากที่สุด ซึ่งเท่ากับผลรวมของเวลาที่ในการร้องขอ task จากเซิร์ฟเวอร์, เวลาที่ใช้ในการดาวน์โหลดไฟล์อินพุต, เวลาที่ใช้ในการประมวลผล และเวลาที่ใช้ในการส่ง result กลับไปยังเซิร์ฟเวอร์ ตามลำดับ

ในการทดลองนี้ หากใช้สมการดังกล่าว สามารถหาค่า t_{client} = 133,038,769 ms และ t_{server} นั้นจะใช้การประมาณค่า 300,000 ms (5 นาที และ $t_{partition}$ = 0 เนื่องจากงานนี้ไม่มีการใช้งานในส่วนของ partition) ได้เป็น 133,338,769 ms หรือประมาณ 37 ชั่วโมง ตามลำดับ ซึ่งจะพบว่าอาจไม่ตรงกับความเป็นจริงเพราะสมการดังกล่าวได้ละเว้นปัจจัยเรื่องช่วงเวลาในการเข้าร่วมของไคลเอนต์ออกไป (ในขณะเดียวกันหากใช้เครื่องเซิร์ฟเวอร์ประมวลผลงานนี้เพียงเครื่องเดียวจะใช้เวลาประมาณ 97 ชั่วโมง)

ในด้านของการประมาณค่าจำนวนชิ้นงานและจำนวนไคลเอนต์ที่เหมาะสมนั้นสามารถประมาณได้จากสมการเบื้องต้นดังนี้

$$t_{n,c} = t_{server} + (t_{avg} * n/c)$$

โดย $t_{n,c}$ นั้นจะเป็นค่าประมาณเวลาที่ใช้ทั้งหมดในการทำงาน n ชิ้นเมื่อมีไคลเอนต์จำนวน c เครื่องตามลำดับ t_{server} นั้นจะเหมือนกับสมการที่แล้ว (เป็นส่วนการทำงานแบบ

sequential) ส่วนพจน์หลังจะเป็นเวลาบนฝั่งไคลเอนต์ โดย t_{avg} จะเป็นเวลาเฉลี่ยที่ใช้ในการทำงาน 1 ชิ้น (รวมเวลาการเคลื่อนย้ายข้อมูลและติดต่อกับเซิร์ฟเวอร์) และ n เป็นจำนวนชิ้นงาน

$$t_{sequential} = t_{execution} * n$$

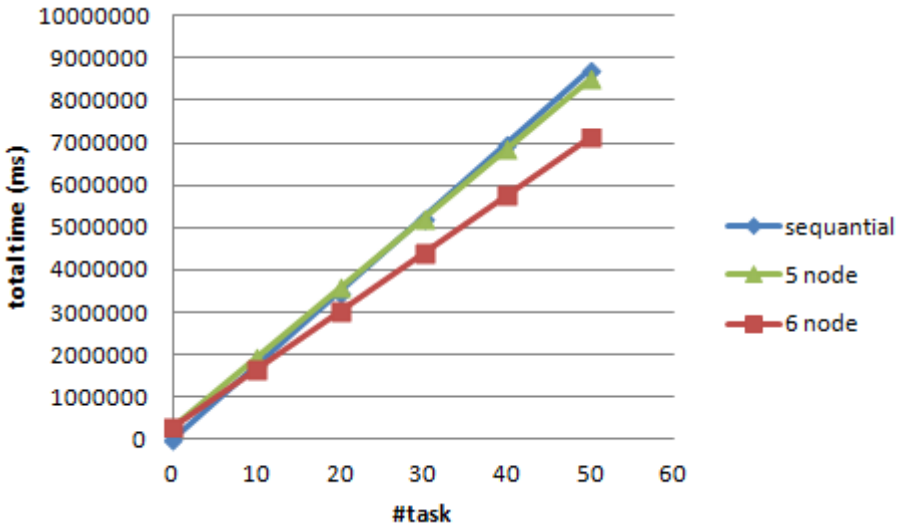
$t_{sequential}$ เป็นเวลาที่ใช้ในกรณีที่ประมวลผลเครื่องเดียว โดยสามารถคิดได้จากเวลาที่ใช้ในการประมวลผล คูณกับจำนวนชิ้นงาน การตัดสินใจหาจำนวนชิ้นงานที่เหมาะสมนั้นจะตัดสินใจจาก ค่าจำนวนไคลเอนต์และจำนวนชิ้นงานที่ทำให้

$$t_{n,c} < t_{sequential}$$

การทดลองนี้จะประมาณค่า t_{server} = 300,000 และใช้ค่า t_{avg} = 821,001 ซึ่งเป็นค่าเฉลี่ยของไคลเอนต์ที่ทำงานได้ช้าที่สุด สำหรับค่าเวลาประมวลผลของ sequential คือเวลาประมวลผลของเครื่องเซิร์ฟเวอร์ ซึ่งเท่ากับ 174,236 (หากใช้ค่าเฉลี่ยในการทำงาน 1 ชิ้นของไคลเอนต์ทั้งหมด จะพบว่ามีค่าเท่ากับ 160,029 ซึ่งพบว่าใช้เวลาน้อยกว่าเวลาที่ประมวลผลบนเครื่องเซิร์ฟเวอร์ ทั้งนี้เพราะไคลเอนต์ส่วนหนึ่งมีประสิทธิภาพเครื่องที่สูงกว่าเครื่องเซิร์ฟเวอร์ ทำให้เวลาประมวลผลเฉลี่ยลดลง) เวลาที่ใช้สามารถคำนวณได้เป็นตารางที่ 11 ดังนี้

ตารางที่ 11 เวลาที่ใช้ในการประมวลผลที่จำนวนไคลเอนต์และชิ้นงานต่างๆ ของงาน superPI

node / task	0	10	20	30	40	50
sequential	0	1742365	3484730	5227094	6969459	8711824
5 node	300000	1942002	3584005	5226007	6868009	8510012
6 node	300000	1668335	3036671	4405006	5773341	7141677



รูปที่ 22 เวลาที่ใช้ในการประมวลผลของจำนวนชิ้นงานต่างๆ

จากตารางที่ 11 และ รูปที่ 22 จะพบว่าจำนวนชิ้นงานที่เหมาะสมคือ 10 ชิ้นงาน และไคลเอนต์ตั้งแต่ 5 เครื่องขึ้นไป

ถึงแม้ว่างานนี้หากพิจารณาในด้านประสิทธิภาพจะพบว่ามีความเหมาะสมที่จะนำมาทำบน WebGrid.js แต่ในด้านของผลลัพธ์ที่ได้ของงาน (ความละเอียดของค่า PI จากวิธีการสุ่มแบบมอนติคาร์โล) จะพบว่าไม่มีความเหมาะสมเพราะ จาก[52] จะพบว่าค่า *PI* ที่หาได้จากวิธีนี้จะลู่เข้าหาค่าที่แท้จริงด้วยอัตราที่ต่ำมาก (การสุ่ม 1,000,000 รอบจะมีโอกาสสูงสุดที่ค่า PI จะแม่นยำถึงแค่หลักที่ 6 เท่านั้น)

6.2.2 wordcount

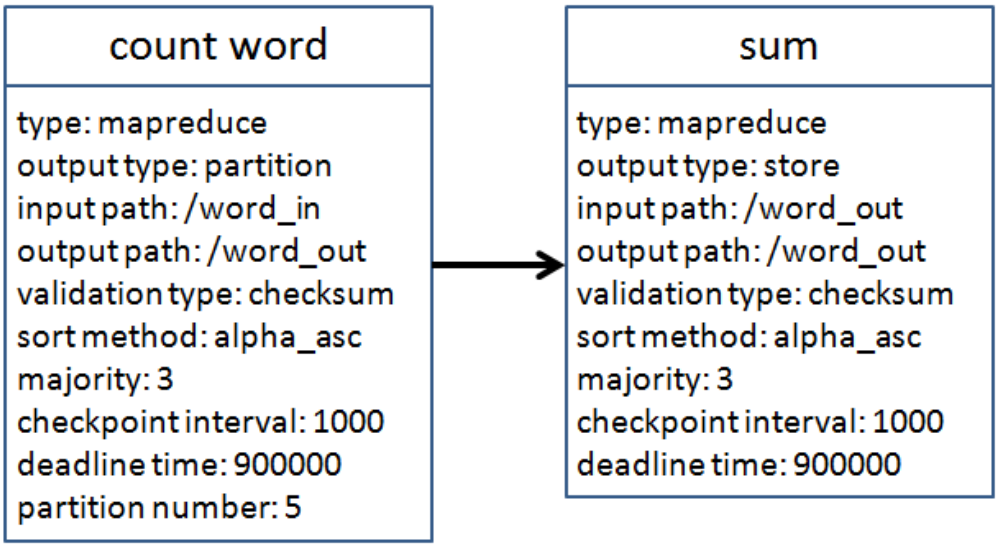
6.2.2.1 **ลักษณะของงาน**

การทดลองนี้เป็นการทดลองโปรแกรมนับคำ ซึ่งเป็นโปรแกรมพื้นฐานของรูปแบบ MapReduce โดยการทดลองนี้มีจุดประสงค์เพื่อทดสอบการทำงานรูปแบบ MapReduce ของเฟรมเวิร์ก และเพื่อศึกษาการใช้งานจริงในงานที่เน้นด้านข้อมูล (data-intensive application) ซึ่งตรงข้ามกับการทดลองที่แล้วที่เป็นงานเน้นด้านการประมวลผล การทดลองนี้เป็นการทำโปรแกรม wordcount กับไฟล์อินพุตขนาด 2.5 MB จำนวน 400 ไฟล์ รวมเป็นขนาดทั้งสิ้น 1 GB โดยไฟล์อินพุตนั้นสร้างโดยการสุ่มคำจากไฟล์พจนานุกรมที่มีจำนวนคำ 99150 คำ

เหมือนดังการทดลองที่แล้ว จากการวิเคราะห์เบื้องต้นด้วยอัตราส่วนเวลาที่ใช้ในการคำนวณกับเวลาที่ใช้ในการเคลื่อนย้ายข้อมูลได้ดังนี้ C/L ratio = 4483 / (242+58+2484) = 1.61 ซึ่งพบว่างานนี้น่าจะมีอาจจะยังไม่เหมาะสมในการนำมาใช้บนระบบนี้มากนัก เนื่องจากเวลาที่ใช้ในการเคลื่อนย้ายข้อมูลค่อนข้างสูงเมื่อเทียบกับเวลาที่ใช้ในการคำนวณ

6.2.2.2 **รายละเอียดและค่าที่ใช้ในการทดลอง**

รายละเอียดของโปรแกรมนั้น เป็นผังงานประกอบไปด้วยงาน 2 job ได้แก่ count word และ sum ซึ่งสามารถแสดงเป็นรูปได้ดังรูปที่ 23



รูปที่ 23 แสดงผังงานและรายละเอียดงานที่ใช้ในการทดลอง

งาน count word เป็นการนับคำจากไฟล์อินพุธ แล้วสร้างเป็นข้อมูล key-value เพื่อให้งาน sum รวมค่าที่ได้จากไฟล์อินพุตอื่นๆ โดยรายละเอียดของงาน count word เป็นดังนี้

- มีจำนวน task ทั้งหมด 400 task (ตามจำนวนอินพุตไฟล์)
- โมดูล validator ใช้วิธี MD5 checksum และมีการตั้งค่า majority ไว้ที่ 3 ทำให้งานนี้มี result อย่างน้อย 1200 result
- output_type เป็น partition เพื่อให้ทำการแบ่งกลุ่มผลลัพธ์ที่ได้ และมีค่า partition number เท่ากับ 5 (แบ่งข้อมูลเป็น 5 กลุ่มหรือเท่ากับ 5 reducer ใน MapReduce แบบปกติ)
- ไฟล์อินพุตมีทั้งหมด 400 ไฟล์ แต่ละไฟล์มีขนาด 2.5 MB ส่วนไฟล์ผลลัพธ์ชั่วคราว(ที่ไคลเอนต์ส่งมา ก่อนผ่านโมดูล collector เลือกเก็บคำตอบสุดท้าย)มีจำนวนเท่ากับ result ของระบบ(เบื้องต้นจะเท่ากับ 1200 ไฟล์ ในกรณีที่คำตอบของไคลเอนต์ถูกต้องทั้งหมดทำให้ระบบไม่ต้องสร้าง result เพิ่ม) ไฟล์เอาต์พุตของงานนี้มีทั้งหมด 5 ไฟล์ ซึ่งเท่ากับค่า partition number ที่ได้ตั้งไว้
- มีการใช้ combiner เพื่อลดขนาดข้อมูลก่อนส่งไปยังเซิร์ฟเวอร์ โดยหลักการ combiner ที่ใช้ในงานนี้คือ การรวมความถี่ของคำ (โดยปกติโปรแกรม wordcount หากไม่มี combiner ในขั้นตอน map จะส่งค่ารายการของ 1 ตามจำนวนความถี่ไป)

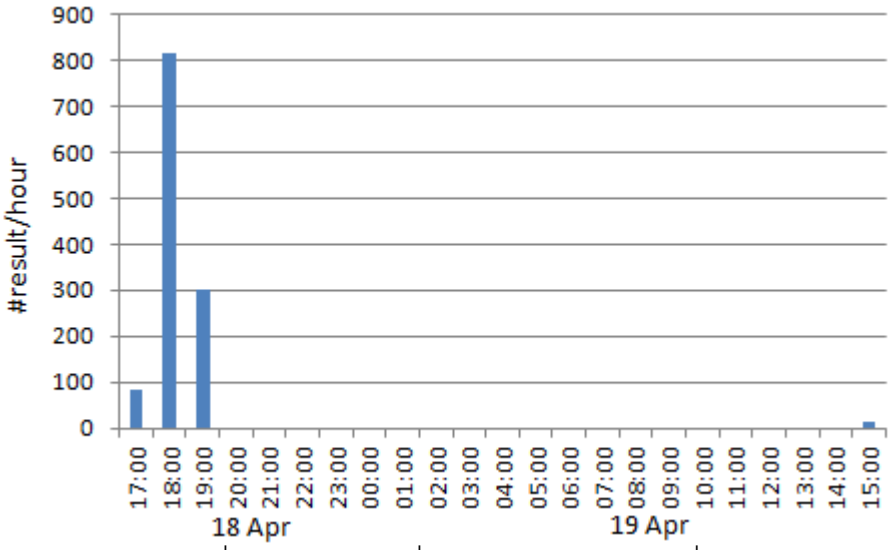
- ค่า checkpoint interval เท่ากับ 1000 นั้น หมายถึงให้ทำจุดตรวจสอบทุกๆค่า 1000 ค่า key-value ของอินพุต
- เนื่องจากงานนี้แต่ละ task อาจใช้เวลาในการคำนวณไม่เหมือนกัน ดังนั้นค่า point ในงานนี้คำนวณได้จาก ค่าที่น้อยสุดของ FLOPS * execution_time ของ task นั้น (งานนี้ 1 task มี 3 result) หารด้วย 10^{10}

งาน count word นั้น เมื่อ task นั้นสามารถรวบรวม result ได้ถึงจำนวน majority (ในที่นี้คือ 3)แล้ว โมดูล validator ก็จะเริ่มตรวจสอบ result เหล่านั้น หาก task นั้นมีจำนวน result ที่ถูกต้องตั้งแต่ค่า majority ขึ้นไป task นั้นก็จะถูกตั้งสถานะเป็น validated เพื่อให้โมดูล collector จัดเก็บผลลัพธ์นั้นลงฐานข้อมูล key-value และลบไฟล์ผลลัพธ์ชั่วคราวออก เมื่อ collector ได้เก็บรวบรวมผลลัพธ์ของทุก task แล้ว โมดูล task manager จะทำการดึงข้อมูลจากฐานข้อมูล key-value มาเขียนเป็นไฟล์ผลลัพธ์สุดท้าย และเริ่มสร้าง task และ result สำหรับงานต่อไป โดยงาน sum นั้นจะเป็นการรวมค่าที่อ่านได้จากงาน count word เพื่อหาคำตอบสุดท้าย รายละเอียดในจุดที่แตกต่างจากงาน count word มีดังนี้

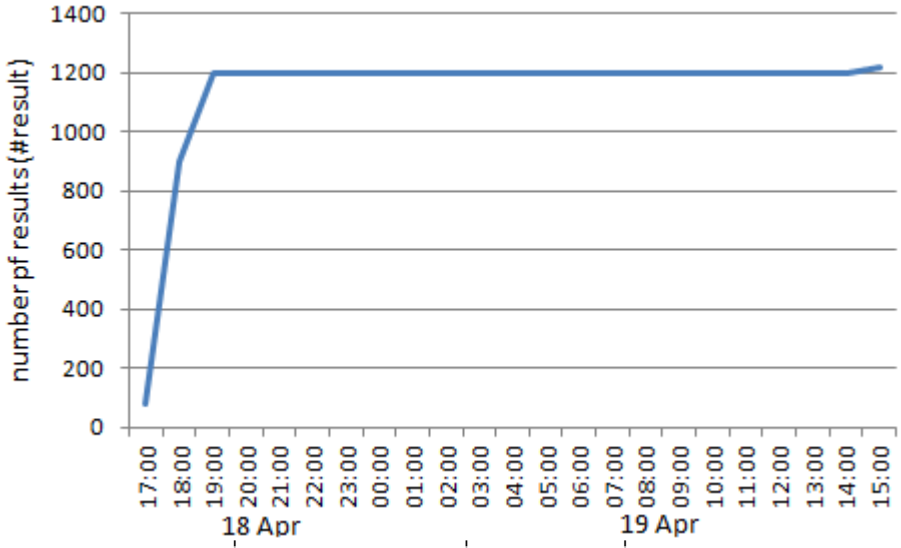
- มี task เพียงแค่ 5 task
- output type เป็น store เพราะไม่มีความจำเป็นที่จะต้องทำ partition คำตอบที่ได้
- ไฟล์อินพุตมี 5 ไฟล์ (ไฟล์คำตอบจากงานที่แล้ว) เนื่องจาก output type ของงานนี้เป็น store ไฟล์ผลลัพธ์สุดท้ายก็มีจำนวน 5 ไฟล์เช่นกัน

6.2.2.3 ผลการทดลองและการอภิปรายผล

การทดลองนั้นเริ่มต้นในวันที่ 18 เมษายน 2556 เวลา 16:00 โดยประมาณ และสิ้นสุด(นับจากเวลาที่ไคลเอนต์ส่งผลลัพธ์งานชิ้นสุดท้ายมายังเซิร์ฟเวอร์)ในวันที่ 19 เมษายน 2556 เวลา 15:17 โดยคิดเป็นเวลาทั้งหมด 23 ชั่วโมงโดยประมาณ ข้อมูลแสดงจำนวนงานที่อาสาสมัครทำในรายชั่วโมง สามารถแสดงได้ดังรูปที่ 24 และรูปที่ 25



รูปที่ 24 จำนวนงานที่อาสาสมัครทำในแต่ละชั่วโมง



รูปที่ 25 จำนวน result ที่เสร็จในเวลาที่กำหนด

จากรูปที่ 24 และรูปที่ 25 พบว่าในช่วงแรกจะมีอาสาสมัครเข้าร่วมจำนวนหนึ่งก่อนที่หยุดไปในช่วง 18:00 เป็นต้นไปเนื่องจากงาน count word นี้เป็นงานที่ไม่ใหญ่มาก ทำให้อาสาสมัครสามารถทำเสร็จได้อย่างรวดเร็ว ซึ่งอาสาสมัครสามารถทำ job count word เสร็จทั้งหมดก่อนเวลา 19:00 แต่อาสาสมัครได้ออกจากการคำนวณไปก่อนที่เซิร์ฟเวอร์จะจัดเก็บผลลัพธ์ดังกล่าวลงฐานข้อมูล key-value เสร็จทั้งหมดและสร้าง task สำหรับ job sum (ซึ่งเป็นงานที่เล็กมากๆ เพราะเป็นแค่การรวมค่าที่ได้ และมีจำนวน task ทั้งหมดเพียง 5 task) โดยได้มีอาสาสมัครเข้ามาประมวลผลงาน job sum ในช่วงเวลา 15:00

ในส่วนของไคลเอนต์ที่เข้าร่วมนั้นจะมีอยู่ทั้งหมด 7 เครื่อง โดยเป็นเว็บเบราว์เซอร์ Google Chrome และระบบปฏิบัติการ Windows ทั้งหมด

สำหรับ ความสัมพันธ์ระหว่าง GFLOPS ของไคลเอนต์กับเวลาที่ใช้ประมวลผลโดยเฉลี่ยแสดงดังตารางที่ 12 โดยมีข้อสังเกตว่าบางครั้งเครื่องไคลเอนต์ที่มีต่า GFLOPS สูงกว่าแต่กลับใช้เวลาประมวลผลสูงกว่า ตรงจุดนี้เป็นเพราะว่า task แต่ละ task นั้นไม่เหมือนกัน ทำให้ใช้เวลาในการประมวลผลไม่เท่ากัน (ซึ่งตรงข้ามกับการทดลองที่แล้วที่ task ทุก task เป็นงานเดียวกันหมด) ซึ่งบาง task ที่มีขนาดเล็ก(เช่น task ของ job sum) อาจจะดึงค่าเฉลี่ยของเวลาที่ใช้ในการประมวลผลลงได้

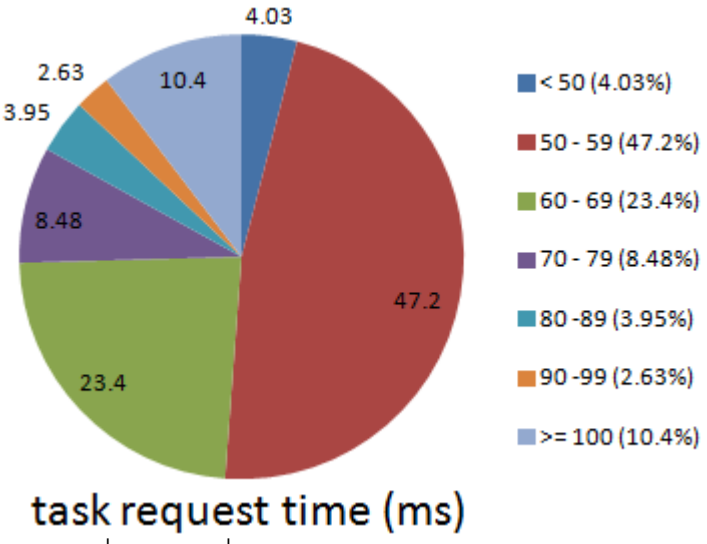
ตารางที่ 12 ช่วง GFLOPS ของไคลเอนต์ และเวลาที่ใช้ประมวลผลโดยเฉลี่ย

GFLOPS	ค่าเฉลี่ยเวลาที่ใช้ในการประมวลผล (ms)	จำนวนไคลเอนต์
0.5 – 0.59	7280	2
0.6 – 0.69	4775	2
0.7 – 0.79	5570	1
มากกว่า 0.8	2171	1

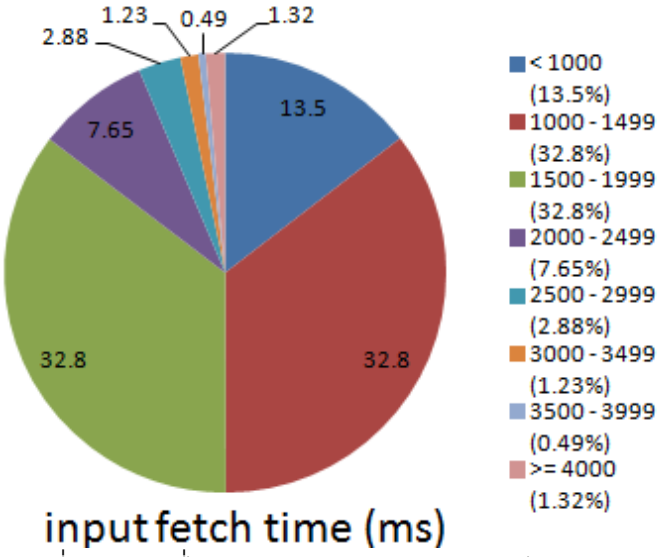
เนื่องจากการทดลองนี้มีงานขนาดเล็กที่สามารถทำเสร็จได้ในเวลาอันสั้น ทำให้มีเพียงไคลเอนต์ 2 เครื่องที่ได้ทำงานส่วนใหญ่ของระบบไป (มากกว่า 100 result ขึ้นไป) ส่วนไคลเอนต์ที่เหลือนั้นได้ทำงานเป็นช่วงสั้นๆ(น้อยกว่า 50 result)เท่านั้น

เนื่องจากการทดลองนี้เป็นการทดลองงานที่มีเน้นทางด้านข้อมูล ทำให้มีการเคลื่อนย้ายข้อมูลจำนวนมาก โดยเวลาที่ไคลเอนต์ใช้ในการร้องขอ task, ดาวน์โหลดไฟล์อินพุต และส่งผลลัพธ์กลับมายังเซิร์ฟเวอร์นั้น สามารถแสดงได้ดังรูปที่ 26, 27 และ 28 ตามลำดับ

จากรูปที่ 26 พบว่า เวลาที่ใช้ในการร้องของานนั้นไม่แตกต่างจากการทดลองที่แล้ว สาเหตุเพราะขนาดของข้อมูลที่ต้องใช้ในการส่งไม่แตกต่างมากนัก โดยเซิร์ฟเวอร์ต้องเสียเวลาส่วนหนึ่งในการแก้ไขฐานข้อมูลเพื่อจอง result ให้ไคลเอนต์นั้น ก่อนส่ง result นั้นให้ไคลเอนต์ไปทำต่อไป



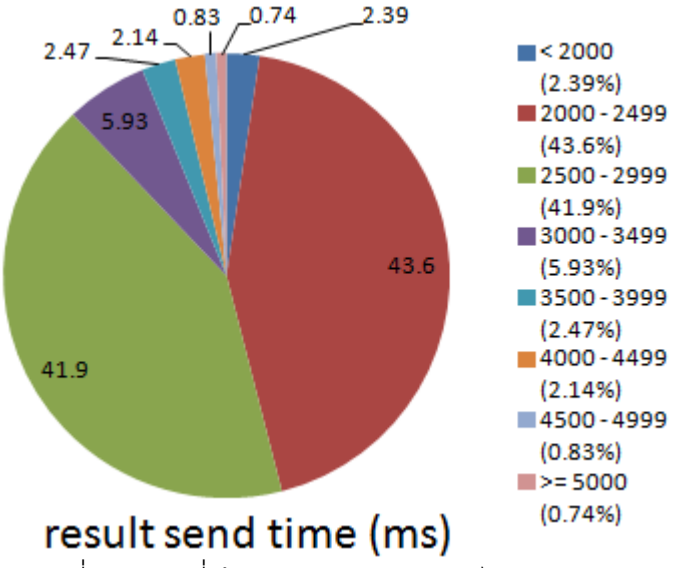
รูปที่ 26 เวลาที่ใช้ในการร้องของงานจากเซิร์ฟเวอร์



รูปที่ 27 เวลาที่ไคลเอนต์ใช้ในการดาวน์โหลดไฟล์อินพุต

จากรูปที่ 27 พบว่างานนี้จะใช้เวลาในการดาวน์โหลดไฟล์ค่อนข้างมาก (ประมาณ 1 วินาทีขึ้นไป) เพราะต้องเคลื่อนย้ายข้อมูลไฟล์อินพุตขนาด 2.5 MB และเนื่องจากแต่ละงานนั้นใช้ไฟล์อินพุตที่ต่างกัน ทำให้ไม่สามารถใช้ประโยชน์จากแคชได้

จากรูปที่ 28 พบว่าเวลาที่ใช้ในการส่งผลลัพธ์ของการทดลองนี้จะเพิ่มขึ้นจากการทดลองที่แล้วอย่างเห็นได้ชัด เพราะปริมาณข้อมูลที่ไคลเอนต์ต้องส่งกลับเซิร์ฟเวอร์นั้นมีมากขึ้น และเซิร์ฟเวอร์เสียเวลาส่วนหนึ่งในการเขียนไฟล์ผลลัพธ์ชั่วคราวรวมไปถึงการปรับปรุงฐานข้อมูลของระบบ โดยไคลเอนต์ส่วนใหญ่จะใช้เวลาประมาณ 2 วินาที



รูปที่ 28 เวลาที่ใช้ในการส่งผลลัพธ์กลับไปยังเซิร์ฟเวอร์

การทดลองนี้มีจุดที่น่าสังเกตอีกจุดคือ ฝั่งเซิร์ฟเวอร์ต้องมีการประมวลผลแบ่งข้อมูล partition ของผลลัพธ์และนำเข้าฐานข้อมูล key-value ที่ยังมีความล่าช้าอยู่ ซึ่งควรมีการปรับปรุงให้ดีขึ้น (optimize) ก่อนหากมีการนำเฟรมเวิร์กนี้ไปใช้จริงในด้านการทำงานแบบ MapReduce โดยอาจปรับปรุงโดยการลดจำนวนครั้งที่ติดต่อกับฐานข้อมูล key-value ลง ทั้งนี้แม้ฝั่งเซิร์ฟเวอร์อาจมีความล่าช้าในการทำงาน แต่ในส่วนของโมดูล scheduler ก็ยังสามารถแจกจ่ายงานและรับผลลัพธ์จากไคลเอนต์ได้ตามปกติ ความล่าช้าในส่วนของการเก็บผลลัพธ์ key-value นั้น ส่งผลเพียงแค่การปรับปรุงข้อมูลในส่วนของการแสดงผลความก้าวหน้าของงานช้าลงเท่านั้น เมื่อมองจากมุมมองของไคลเอนต์ ซึ่งเป็นข้อดีของการออกแบบสถาปัตยกรรมแบบแยกโมดูลที่ใช้ในเฟรมเวิร์กนี้

6.2.2.4 การวิเคราะห์ถึงความเหมาะสมของงาน

จากสมการเวลาที่ใช้ทั้งหมดจากการทดลองที่แล้ว การทดลองนี้จะใช้ค่าประมาณ $t_{task\_create} + t_{result\_create}$ = 300,000 เนื่องจากการทดลองนี้มีการใช้งานในส่วนของการ partition เพื่อความง่ายต่อการคำนวณ ค่า $t_{partition}$ จึงใช้การประมาณค่าประมาณ โดยกำหนดให้ใช้เวลา 5,000 ms (5 วินาที) ต่อการทำ partition ข้อมูลของ 1 task และกำหนดให้เวลาที่ใช้ในการประมวลผล job sum นั้นมีค่าน้อยมากจนสามารถละทิ้งได้ ทั้งนี้การทดลองนี้ใน 1 task มี 3 result ดังนั้นในการคำนวณเวลาสำหรับไคลเอนต์ WebGrid.js จึงต้องใช้จำนวนของ result แทน โดยค่า $t_{partition}$ = 5000 * 400 = 2,000,000 และ t_{client} = 6,494,978 สามารถคำนวณ t_{total} = 8,794,978 ms หรือประมาณ 147 นาที โดยหากใช้เครื่องเซิร์ฟเวอร์เพียงเครื่องเดียวจะประมาณค่าเวลาที่ใช้ได้เท่ากับ

1,793,333 ms หรือประมาณ 30 นาที ซึ่งจะพบว่าเครื่องเดียวจะใช้เวลาน้อยกว่า เพราะระบบ WebGrid.js จะต้องคิดในส่วนของเวลาที่ใช้ในการติดต่อและส่งข้อมูลระหว่างไคลเอนต์กับเซิร์ฟเวอร์, เวลาที่ใช้ในการทำการ partition ข้อมูล รวมไปถึงความสิ้นเปลืองจากการทำ result หลายชุดเพื่อใช้ในการเช็คคำตอบ (หากใช้ result เพียงชุดเดียว เวลาทั้งหลายจะลดลงเหลือประมาณ 58 นาที) ทั้งนี้เครื่องเซิร์ฟเวอร์ที่ใช้ตั้งอยู่บนสมมติฐานว่ามีหน่วยความจำไม่จำกัด จึงสามารถเก็บข้อมูลทั้งหมดลงหน่วยความจำได้ และเวลาที่ใช้ในการทำ partition ข้อมูล รวมไปถึงเวลาที่ใช้ในการเข้าถึงหน่วยความจำและไฟล์ข้อมูลมีค่าน้อยมากจนสามารถละทิ้งได้

ในด้านของการประมาณค่าจำนวนชิ้นงานที่เหมาะสมและจำนวนไคลเอนต์ เนื่องจากงานนี้มีการใช้ในส่วนของการทำ partition ข้อมูล ซึ่งสามารถคำนวณได้ดังนี้

$$t_{partition} = \begin{cases} t_p \; ; p < 0 \\ p \end{cases}$$

$$p = (t_p * n) - (t_{avg} * m/c)$$

โดยค่า t_p เป็นเวลาที่ใช้ในการทำ partition ข้อมูลของแต่ละ task, n เป็นจำนวน task ทั้งหมด, t_{avg} เป็นเวลาที่ไคลเอนต์ทำงานเสร็จในแต่ละงาน และ m เป็นจำนวน result ทั้งหมด เนื่องจากงานส่วน partition นั้นจะทำโดยโมดูล collector ซึ่งทำไปพร้อมกับช่วงเวลาที่ไคลเอนต์ประมวลผล

การทดลองจะใช้การประมาณค่า t_{task_create} + t_{result_create} = 300,000 ค่า t_{avg} = 5,067 ซึ่งเป็นค่าเฉลี่ยของไคลเอนต์ที่ทำงานได้ช้าที่สุด ค่า $t_{partition}$ จะใช้ค่าประมาณ 5,000 ต่อ 1 task สำหรับค่าเวลาประมวลผลของ sequential คือเวลาประมวลผลของเครื่องเซิร์ฟเวอร์ ซึ่งเท่ากับ 4,330 เพื่อให้มองเห็นแนวโน้มได้ชัดเจนขึ้น จึงให้ค่า m = n ซึ่งสามารถสรุปได้ดังตารางที่ 13

ตารางที่ 13 เวลาที่ใช้ในการประมวลผลที่จำนวนไคลเอนต์และจำนวนชิ้นงานต่างๆของงาน wordcount

node / task	0	100	200	300	400	500
sequential	0	433000	866000	1299000	1732000	2165000
5 node	300000	800000	1300000	1800000	2300000	2800000
10 node	300000	800000	1300000	1800000	2300000	2800000
20 node	300000	800000	1300000	1800000	2300000	2800000

จากตารางที่ 13 พบว่าการเพิ่มจำนวนไคลเอนต์ในงานนี้ไม่ได้ช่วยให้เวลาที่ใช้ในการประมวลผลทั้งหมดน้อยลง หากพิจารณาจากสมการจะพบว่า ค่าของ $t_{partition}$ สูงและเป็นคอขวดของระบบ (เซิร์ฟเวอร์ต้องใช้เวลาส่วนหนึ่งในการอ่านไฟล์และเขียนข้อมูลลงฐานข้อมูลแบบ key-value) การที่จะได้ประโยชน์จากระบบ WebGrid.js และจำนวนไคลเอนต์ที่เพิ่มขึ้น นั้นสามารถแก้ไขได้ดังนี้

- ลดค่า $t_{partition}$ ของระบบลง โดยการเพิ่มจำนวนฐานข้อมูล key-value หรืออาจจะลดจำนวนครั้งที่ติดต่อกับฐานข้อมูล key-value ลง
- เลือกงานที่เวลาที่ต้องใช้ในการประมวลผลบนฝั่งไคลเอนต์ให้มากพอที่จะชนะความสิ้นเปลืองของ $t_{partition}$ ได้ โดยอาจจะเพิ่มค่า C/L ratio ให้มากขึ้น

6.2.3 **อภิปรายผลการทดลอง และข้อสังเกตอื่นๆ**

จากการทดลองนำเฟรมเวิร์กไปทดลองใช้งานจริง สามารถอภิปรายผลการทดลอง และข้อสังเกตอื่นๆได้ดังนี้

- ในการทดลองที่ 6.2.1 ที่ต้องใช้ระยะเวลาในการดำเนินการหลายวัน พบว่าโปรเซสจะมีโอกาสที่จะล่ม ทำให้ไม่สามารถแจกจ่ายงานให้ไคลเอนต์ได้ในบางช่วงเวลา จากการทดลองโปรเซสที่ล่มจะมีอยู่ 2 กรณีคือ โปรเซสของ Node.js ล่ม กับโปรเซสฐานข้อมูล MySQL ล่ม ซึ่งหากเป็นกรณีแรกนั้นสามารถแก้ปัญหาได้โดยการใช้โมดูล forever เพื่อช่วยเฝ้าสังเกตและคอยเริ่มการทำงานให้โปรเซสนั้นใหม่ในกรณีที่โปรเซสนี้นล่มได้ แต่ถ้าหากเป็นกรณีหลัง ในการทดลองนี้ใช้วิธีแก้โดยการใช้คำสั่งเข้าไปเริ่มโปรเซสของฐานข้อมูลใหม่ ซึ่งเป็นวิธีการที่ไม่ค่อยมีประสิทธิภาพ ควรหาโปรแกรมสำหรับเฝ้าสังเกตโปรเซสต่างๆ เช่น monit[51] มาใช้งานหากมีการนำเฟรมเวิร์กนี้ไปใช้งานจริง
- ระบบการแสดงผลมีผลต่อการจูงใจให้คนเข้ามาร่วม จากการทดลองพบว่ามีอาสาสมัครที่พยายามทำคะแนนให้ติดใน leaderboard ก่อน แล้วถึงค่อยหยุดการทำงาน
- ระบบอาจเกิดข้อผิดพลาดได้ เนื่องจากเฟรมเวิร์กต้นแบบนี้ยังไม่ได้มีการ transaction ในการดำเนินงานเกี่ยวกับฐานข้อมูล

- ระบบยังมีการใช้งานหน่วยประมวลผลของเครื่องอาสาสมัครมากเกินไป จากข้อสันนิษฐานเบื้องต้นคาดว่า ระบบปฏิบัติการจะจัดสรรทรัพยากรมาให้โปรเซสเว็บเบราว์เซอร์มากที่สุดเพื่อให้การทำงานจบเร็วที่สุด แต่พอเว็บเบราว์เซอร์ทำงานเสร็จแล้ว ก็มีการร้องของานถัดไป ทำให้การทำงานของโปรเซสเว็บเบราว์เซอร์ไม่จบและไม่คืนทรัพยากรให้กับโปรเซสอื่นๆ ซึ่งการแก้ไขในเบื้องต้นคือ การใช้งานฟังก์ชั่น setTimeout() เพื่อให้หน่วยประมวลผลได้พัก และระบบปฏิบัติการจะได้นำทรัพยากรไปแบ่งให้โปรเซสอื่นๆ
- ในกรณีที่ใช้เว็บเบราว์เซอร์ Google Chrome หากเปิดเฟรมเวิร์กไว้มากกว่า 1 แท็บ (tab) จะเป็นการใช้งานซีพียูหลายคอร์โดยอัตโนมัติ เนื่องจาก Google Chrome เป็นเว็บเบราว์เซอร์ประเภทหลายโปรเซส ซึ่งแต่ละแท็บถือว่าเป็นโปรเซสใหม่อีกโปรเซสหนึ่ง ทำให้ระบบปฏิบัติการสามารถแบ่งแต่ละโปรเซสให้ทำงานบนแต่ละคอร์ของซีพียูได้
- จากการทดลองพบว่าระบบประมวลผลแบบอุทิศนั้นจะไม่สามารถประเมินเวลาที่ใช้ในการทำงานที่แน่นอนได้ เพราะขึ้นกับจำนวนอาสาสมัครที่เข้าร่วม ซึ่งจากการทดลองที่ 6.2.2 จะพบว่างานในส่วนของ sum ที่มีน้อยมาก แต่ต้องใช้เวลาถึง 17 ชั่วโมงในการทำเพราะไม่มีอาสาสมัครเข้ามาทำในช่วงเวลานั้น
- ในการทดลอง 6.2.2 พบว่ามีอาสาสมัครเข้ามาอีกจำนวนหนึ่งหลังจากงานทั้งหมดเสร็จสิ้นแล้ว โดยอาจจะต้องใช้เวลาถึง 2-3 วันกว่าอาสาสมัครส่วนใหญ่จะเริ่มเข้ามา
- งานที่เหมาะสมกับระบบควรเป็นงานที่มีการประมวลผลเยอะ แต่มีการขนส่งข้อมูลน้อย (ค่า C/L ratio สูง)

6.3 **สรุปผลการอภิปราย**

ในส่วนนี้เป็นการอภิปรายผลของระบบที่ได้พัฒนาขึ้นในด้านต่างๆ รวมไปถึงรายละเอียดเพิ่มเติมที่ผู้จัดทำงานวิจัยได้ไปหาข้อมูลเพิ่มเติม โดยสามารถสรุปได้ดังนี้

6.3.1 **ประสิทธิภาพ**

6.3.1.1 **การเลือกใช้เครื่องมือเพื่อเพิ่มประสิทธิภาพ**

จากการวิเคราะห์ถึงคุณลักษณะของเครื่องมือต่างๆ ในหัวข้อที่ 3.1.2 และการทดลองสร้างระบบขึ้นมา สามารถสรุปถึงการเลือกใช้งานเครื่องมือได้ดังนี้

- พยายามย้ายส่วนที่มีการประมวลผลของภาษา JavaScript ไปยังส่วนของ web worker เพื่อป้องกันหน้าเว็บค้างจากการใช้เทรดหลักในการประมวลผล นอกจากนี้ web worker ยังสามารถใช้ประโยชน์จากซีพียูหลายคอร์ได้ด้วย
- ใช้ typed array ในการจัดการ array ที่มีข้อมูลประเภทเดียวกัน (เช่น array ของค่า int) เพื่อลดความสิ้นเปลืองจาก array แบบปกติของ JavaScript
- หากต้องการประสิทธิภาพที่ใกล้เคียงกับภาษาที่ทำงานบนเครื่องคอมพิวเตอร์ (เช่น ภาษา C++) ก็สามารถพัฒนาส่วนของการประมวลผลด้วยใช้ native client ได้ เพราะ Google Chrome เป็นเว็บเบราว์เซอร์ที่มีผู้ใช้งานจำนวนมาก[53] ทั้งนี้การใช้งาน native client นั้นต้องพัฒนาด้วยใช้ภาษา C++ ซึ่งอาจทำให้การพัฒนาโปรแกรมยากกว่าการใช้งานภาษา JavaScript
- ถึงแม้ว่าการใช้งาน webCL และ river trail จะสามารถเพิ่มประสิทธิภาพได้มาก แต่ในการใช้งานจริงยังมีข้อจำกัดอีกมาก เพราะความหลากหลายของคอมพิวเตอร์ของไคลเอนต์ และไคลเอนต์จำเป็นต้องลง plug-in เพิ่ม

6.3.1.2 การปรับปรุงประสิทธิภาพของภาษา JavaScript

จากการทดลองในหัวข้อ 3.1 พบว่าแม้ภาษา JavaScript จะมีประสิทธิภาพที่ค่อนข้างสูงเมื่อเทียบกับภาษาโปรแกรมแบบพลวัตด้วยกัน แต่ภาษา JavaScript ก็ยังมีประสิทธิภาพที่ด้อยกว่าภาษาโปรแกรมแบบสถิต เนื่องจากความสิ้นเปลืองจากประเภทตัวแปรประเภทพลวัต ทั้งนี้ภาษา JavaScript สามารถใช้งาน optimizer เช่น Closure[54] เพื่อเพิ่มประสิทธิภาพและลดขนาดของโปรแกรมก่อนการนำไปใช้งานได้

6.3.2 การใช้ทรัพยากรซีพียู

จากการทดลองวัดผลของเครื่องมือต่างๆในหัวข้อที่ 3.1.2 และจากการทดลองในบทนี้ จะพบว่าทั้ง BOINC และ เว็บเบราว์เซอร์จะมีการใช้งานซีพียูเต็มประสิทธิภาพ(ใช้งานซีพียูเต็ม 100% ในคอร์นั้นๆ)ทั้งคู่ โดยการตั้งค่าพื้นฐานของ BOINC จะใช้งานซีพียูเต็ม 100% ทุกคอร์ที่มีอยู่ แต่ BOINC นั้นสามารถลดการใช้งานซีพียูของโปรเซสตนเองได้ เมื่อไคลเอนต์มีการเรียก

โปรเซสอื่นๆขึ้นมา ในขณะที่เว็บเบราว์เซอร์จะไม่สามารถควบคุมการใช้งานซีพียูได้ ดังนั้นในการใช้งานจริงจึงควรตั้ง web worker ไว้ที่ 1 worker ก่อน เพื่อให้ไคลเอนต์สามารถใช้งานอื่นๆได้ในคอร์ที่เหลือของซีพียู โดยเปิดตัวเลือกเพิ่มจำนวน web worker ให้ในกรณีไคลเอนต์ต้องการที่จะบริจาคทรัพยากรมากขึ้น นอกจากนี้ระบบสามารถลดการใช้งานซีพียูได้โดยการใช้คำสั่ง settimeout() ของภาษา JavaScript เพื่อหยุดการทำงานชั่วคราวและให้ระบบปฏิบัติการจัดสรรซีพียูให้กับโปรเซสอื่นๆ

6.3.3 จริยธรรม

เนื่องจากเทคโนโลยีที่ใช้บนไคลเอนต์นั้นเป็นเทคโนโลยีทางด้านเว็บแอปพลิเคชั่น ไคลเอนต์สามารถเริ่มงานได้ทันทีเมื่อเข้าสู่หน้าเว็บโดยที่ไคลเอนต์อาจจะไม่รู้ตัว ส่งผลให้แนวคิดนี้อาจถูกนำไปใช้ในทางที่ผิด เช่น ระบบการคำนวณแบบปรสิต (parasitic computing) ซึ่งเป็นระบบที่นำทรัพยากรของไคลเอนต์ไปใช้โดยที่ไคลเอนต์ไม่ยินยอม ทั้งนี้ผู้ที่ดูแลโครงการนั้นควรให้รายละเอียดเกี่ยวกับโครงการที่ทำ และโปรแกรมภาษา JavaScript บนไคลเอนต์จะไม่เริ่มงานจนกว่าไคลเอนต์จะเป็นคนกดเริ่มด้วยตนเอง

6.3.4 การแปลงโปรแกรมที่พัฒนาจากภาษาโปรแกรมแบบอื่นมาใช้งานบนเว็บเบราว์เซอร์

ในกรณีที่ผู้ดูแลโครงการมีโปรแกรมที่ถูกพัฒนาจากภาษาโปรแกรมอื่นอยู่แล้ว วิธีการที่จะนำโปรแกรมเหล่านั้นสามารถนำมาใช้บนเว็บเบราว์เซอร์มีอยู่ 2 แนวทางคือ การใช้ Emscripten[55] เพื่อแปลง LLVM (low level virtual machine) ไปเป็นภาษา JavaScript อีกแนวทางคือ การนำโปรแกรมเดิมไปพัฒนาต่อบน native client (เฉพาะภาษา C++ เท่านั้น)

6.3.5 ลักษณะของงานที่เหมาะสม

จากการทดลองจะพบว่าลักษณะงานที่เหมาะสมนั้น ควรเป็นงานที่มีลักษณะดังนี้

- สามารถแบ่งเป็นงานย่อยๆได้ โดยที่งานย่อยๆเหล่านั้นเป็นอิสระต่อกัน (embarrassingly parallel)
- ควรเป็นงานที่มีการใช้การประมวลผลเยอะ แต่การขนส่งเคลื่อนย้ายข้อมูลน้อย โดยจุดนี้สามารถประมาณได้โดยการใช้ค่า C/L ratio ซึ่งหากได้ค่านี้เยอะ ก็จะแสดงว่างานนี้เหมาะสมกับระบบมากขึ้น
- จุดตรวจสอบ (checkpoint) ของงานไม่ควรมีขนาดเกิน 5 MB เพราะเป็นขนาดพื้นที่สูงสุดของ localStorage ที่เว็บเบราว์เซอร์อนุญาตให้เก็บสำหรับเว็บนั้นๆ

- ในกรณีมีการใช้งานในส่วนของการ partition ข้อมูล ควรแบ่งงานให้มีส่วนของการประมวลผลของไคลเอนต์มากกว่าเวลาที่ใช้ในการทำ partition ข้อมูลบนเซิร์ฟเวอร์
- สำหรับเวลาที่ใช้ในการประมวลผลต่อ 1 ชิ้นงานย่อยที่เหมาะสมนั้นจะมีค่าไม่แน่นอน ขึ้นอยู่กับดุลพินิจของผู้ดูแลโครงการ โดยค่าเฉลี่ยของโครงการประมวลผลแบบอุทิศจะอยู่ที่ประมาณ 9 วัน[56] (เวลานี้คิดจากเวลาตั้งแต่เซิร์ฟเวอร์ส่งงานให้ไคลเอนต์ทำ จนถึงเวลาที่ไคลเอนต์ส่งผลลัพธ์ของงานกลับมายังเซิร์ฟเวอร์)

6.3.6 การเปรียบเทียบกับระบบการคำนวณบนเบราว์เซอร์อื่น ๆ

ในหัวข้อนี้เป็นการเปรียบเทียบคุณสมบัติระหว่าง WebGrid.js กับระบบการคำนวณบนเบราว์เซอร์ตัวอื่นๆ โดยเลือก GridBee เข้ามาเปรียบเทียบเพราะเป็นเฟรมเวิร์กเพียงตัวเดียวที่มีการเปิดเผยรหัสโปรแกรมและมีลักษณะที่ใกล้เคียงกับ WebGrid.js มากที่สุด ในขณะที่ระบบอื่นๆ ที่มีการนำเสนอผลงานการประชุมทางวิชาการนั้น มักเป็นระบบสำหรับงานใดงานหนึ่งโดยเฉพาะและไม่มีการเปิดให้ดาวน์โหลด ทำให้ไม่สามารถนำมาเปรียบเทียบคุณสมบัติได้อย่างเด่นชัดได้

จุดที่แตกต่างระหว่าง WebGrid.js กับ GridBee คือ GridBee เป็นเว็บเบราว์เซอร์ไคลเอนต์สำหรับเชื่อมต่อกับระบบการคำนวณแบบกระจาย ในขณะที่ WebGrid.js เป็นเฟรมเวิร์กที่มีทั้งฝั่งไคลเอนต์และเซิร์ฟเวอร์ โดยสามารถสรุปคุณสมบัติของทั้งสองเฟรมเวิร์กได้เป็นตารางที่ 14 ดังนี้

ตารางที่ 14 เปรียบเทียบคุณสมบัติของ GridBee และ WebGrid.js

	GridBee	WebGrid.js
BOINC support	✓	-
Native client	✓	-
Mapreduce	-	✓
Workflow	-	✓

ตารางที่ 14 เป็นการเปรียบเทียบคุณสมบัติระหว่าง GridBee และ WebGrid.js โดยสามารถสรุปได้ดังนี้

- BOINC support เนื่องจากเฟรมเวิร์ก GridBee เป็นเว็บไคลเอนต์ที่รองรับ BOINC ได้ ดังนั้นจึงเหมาะสมกับโครงการที่พัฒนาโปรแกรมโดยใช้ BOINC อยู่แล้ว เพราะสามารถใช้ระบบเดิมได้ทันที โดยที่ไม่ต้องย้ายส่วนของเซิร์ฟเวอร์ไปยังระบบใหม่ ทั้งนี้อาจต้องมีการพัฒนาหรือแปลงโปรแกรมอื่นๆที่มีอยู่แล้ว ให้เป็นโปรแกรมภาษา JavaScript เพื่อให้สามารถนำมาใช้บนเว็บเบราว์เซอร์ได้
- Native client เฟรมเวิร์ก GridBee สามารถใช้งานร่วมกับ native client ได้
- MapReduce เนื่องจาก WebGrid.js เป็นเฟรมเวิร์กที่มีส่วนการทำงานครอบคลุมในฝั่งของเซิร์ฟเวอร์ด้วย จึงสามารถรองรับการทำงานรูปแบบโปรแกรมแบบ MapReduce ได้
- Workflow เฟรมเวิร์ก WebGrid.js สามารถใช้งาน workflow เพื่อกำหนดลำดับการทำงานในแต่ละส่วนได้

จากคุณสมบัติดังกล่าว พบว่าหากเรามีโครงการที่พัฒนาด้วย BOINC อยู่แล้วหรือเป็นโครงการที่เน้นประสิทธิภาพเป็นประเด็นหลัก ก็ควรเลือกพัฒนาเฟรมเวิร์ก BOINC และเลือก GridBee เพื่อเพิ่มการรองรับการทำงานบนเว็บเบราว์เซอร์ ในขณะที่โครงการที่ต้องการในด้านอื่นๆ เช่น ความง่ายในการพัฒนา, ความง่ายในการจัดการงานต่างๆ นั้นควรเลือก WebGrid.js เพราะเฟรมเวิร์กนี้มีโครงสร้างที่ช่วยในการจัดการทางฝั่งเซิร์ฟเวอร์ด้วย เช่น การรองรับรูปแบบโปรแกรม MapReduce และ workflow ที่จัดการลำดับการทำงานต่างๆ

บทที่ 7
บทสรุป

ในบทนี้จะกล่าวถึงสิ่งที่ได้จากการวิจัย,แนวทางการวิจัยต่อ และบทสรุป ดังนี้

7.1 **สิ่งที่ได้จากการวิจัย** (Contribution)

สิ่งที่ได้จากการวิจัยนี้ ได้แก่

1. การรวบรวมและสำรวจเครื่องมือและเทคโนโลยีทางด้านเว็บแอปพลิเคชั่น เพื่อนำมาประยุกต์ใช้ในด้านการประมวลผลแบบกระจาย พร้อมทั้งมีการทดลองวัดประสิทธิภาพของเครื่องมือดังกล่าว
2. การสำรวจและวัดประสิทธิภาพของระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์เทียบกับ BOINC ซึ่งเป็นระบบที่มีอยู่แล้ว พร้อมทั้งวิเคราะห์ถึงข้อแตกต่าง,ลักษณะเด่น และงานที่เหมาะสมของทั้งสองระบบ
3. ออกแบบและพัฒนาระบบต้นแบบของการคำนวณแบบกระจายบนเว็บเบราว์เซอร์ขึ้นมา พร้อมทั้งทดลองวัดผลประสิทธิภาพเบื้องต้นเทียบกับ BOINC
4. พัฒนาโปรแกรมบนระบบต้นแบบและทำการทดลองกับอาสาสมัครจริง เพื่อศึกษาถึงลักษณะการใช้งานจริง

7.2 **แนวทางการวิจัยต่อ**

งานวิจัยนี้ได้นำเสนอเฟรมเวิร์กสำหรับสร้างระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์ขึ้นมา โดยเฟรมเวิร์กนี้ได้แสดงให้เห็นว่าระบบคำนวณแบบกระจายบนเว็บเบราว์เซอร์นั้นสามารถทำได้จริง อย่างไรก็ตามยังมีแนวทางการวิจัยต่อที่สามารถแบ่งออกได้เป็น 3 ประเด็น ดังนี้

1. ประยุกต์งานด้านอื่นๆให้สามารถทำบนระบบการคำนวณแบบกระจายบนเว็บเบราว์เซอร์ได้
2. ปรับปรุงจุดที่ยังบกพร่องของเฟรมเวิร์ก เช่น การทำให้เฟรมเวิร์กใช้ทรัพยากรของเครื่องอาสาสมัครน้อยลง, เพิ่มประสิทธิภาพด้วยการนำ NaCl มาใช้, ปรับปรุงระบบให้รองรับรูปแบบการเขียนโปรแกรมรวมไปถึงการใช้งานให้ง่ายขึ้น, วิเคราะห์ขั้นตอนการแจกจ่ายงานที่เหมาะสมกับเว็บเบราว์เซอร์นั้นๆ, วิเคราะห์ถึงการจูงใจให้คนมาเข้าร่วม และพัฒนาระบบให้มีความปลอดภัยมากขึ้น เป็นต้น

3. ทดลองพัฒนาโปรแกรมประเภท low-latency และนำระบบเพียร์ทูเพียร์มาประยุกต์ในการใช้งาน

7.3 บทสรุป

งานวิจัยนี้ได้ทำการศึกษาและพัฒนาระบบการประมวลผลแบบกระจาย โดยใช้เทคโนโลยีทางด้านเว็บแอปพลิเคชั่นขึ้นมา โดยโปรแกรมบนฝั่งไคลเอนต์ที่ใช้จะเป็นเว็บเบราว์สเซอร์ซึ่งเป็นโปรแกรมที่มีอยู่โดยทั่วไปตามเครื่องคอมพิวเตอร์ ทำให้ไคลเอนต์ไม่จำเป็นต้องติดตั้งโปรแกรมเพิ่มเติม และสามารถเข้าร่วมการคำนวณได้โดยง่ายเพียงแค่เปิดเว็บเบราว์สเซอร์ไปยังหน้าที่กำหนด นอกจากนี้ ยังช่วยลดปัญหาด้านความปลอดภัย และความไม่เข้ากันต่างๆอีกด้วย

รายการอ้างอิง

[1] University of California, Berkeley. SETI@home [Online]. Available from: http://setiathome.berkeley.edu/ [2012, July 30]

[2] TOP500.Org. Home | TOP500 Supercomputing Sites [Online]. Available from: http://www.top500.org/ [2012, July 30]

[3] Kaur, P.D., and I. Chana. Unfolding the Distributed Computing Paradigms. ACE 2010: Advances in Computer Engineering, pp. 339–342.

[4] Wikipedia. Ajax (programming) - Wikipedia, the Free Encyclopedia [Online]. Available from: https://en.wikipedia.org/wiki/Ajax_(programming) [2013, April 25]

[5] Anttonen, M., A. Salminen, T. Mikkonen, and A. Taivalsaari. Transforming the Web into a Real Application Platform: New Technologies, Emerging Trends and Missing Pieces. In Proceedings of the 2011 ACM Symposium on Applied Computing, pp.800–807

[6] Herhut, S., R. L. Hudson, T. Shpeisman, and J. Sreeram. Parallel Programming for the Web. In Proceedings of the 4th USENIX Conference on Hot Topics in Parallelism, 2012.

[7] Joyent, Inc. Node.js [Online]. Available from: http://nodejs.org/ [2012, July 25]

[8] The Apache Software Foundation. Apache CouchDB [Online]. Available from: http://couchdb.apache.org/ [2013, April 14]

[9] Unity Technologies. Unity - Game Engine [Online]. Available from: http://unity3d.com/ [2013, April 14]

[10] Mozilla Developer Network. Using Web Workers – MDN [Online]. Available from: https://developer.mozilla.org/en/Using_web_workers [2012, July 25]

[11] Dean, J., and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In Proceeding OSDI'04 Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation.

[12] Anderson, D.P. BOINC: a System for Public-resource Computing and Storage. In Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop.

[13] Anderson, D.P., E. Korpela, and R. Walton. High-performance Task Distribution for Volunteer Computing. In e-Science and Grid Computing, 2005. First International Conference.

[14] Sarmenta, L.F.G., and S. Hirano. Bayanihan: Building and Studying Web-based Volunteer Computing Systems Using Java. Future Generation Computer Systems 15, pp.675–686.

[15] McMahon, A., and V. Milenkovic. Rendering Animations with Distributed Applets. In Proceedings of the International Conference on Computer Graphics and Virtual Reality, 2009.

[16] Social Volunteer Computing. Journal on Systemics, Cybernetics and Informatics (JSCI) 9, pp.34–38.

[17] Merelo, J. J, A. M García, J. L.J Laredo, J. Lupión, and F. Tricas. Browser-based Distributed Evolutionary Computation: Performance and Scaling Behavior. In Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation, pp.2851–2858.

[18] Klein, J., and L. Spector. Unwitting Distributed Genetic Programming via Asynchronous Javascript and XML. In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp.1628–1635.

[19] Merelo, J. J, P.A Castillo, J. L.J Laredo, A. Mora Garcia, and A. Prieto. Asynchronous Distributed Genetic Algorithms with Javascript and JSON. In Proceedings of the Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence), pp. 2851–2858.

[20] Boldrin, F., C. Taddia, and G. Mazzini. Distributed Computing Through Web Browser. In Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th, pp.2020–2024.

[21] KONISHI, F., M. Ishii, S. Ohki, R. UMESTU, and A. KONAGAYA. RABC: A Conceptual Design of Pervasive Infrastructure for Browser Computing Based on Ajax Technologies. Cluster Computing and the Grid, 2007. CCGRID 2007.

[22] BME IK. GridBee Web Computing Framework | Distributed Computations in Browsers Made Possible [Online]. Available from: http://webcomputing.iit.bme.hu/ [2013, February 15]

[23] Krupa, T., P. Majewski, B. Kowalczyk, and W. Turek. On-Demand Web Search Using Browser-Based Volunteer Computing. In Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference, pp.184–190.

[24] Duda, Jerzy, and Wojciech Dłubacz. Distributed Evolutionary Computing System Based on Web Browsers with JavaScript. In Applied Parallel and Scientific Computing, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp.183–191

[25] Google Project. V8 - V8 JavaScript Engine - Google Project Hosting [Online]. Available from: http://code.google.com/p/v8/ [2013, January 13]

[26] Mozilla Developer Network. SpiderMonkey | MDN [Online]. Available from: https://developer.mozilla.org/en/docs/SpiderMonkey [2013, January 13]

[27] GitHub, Inc. GitHub • Build Software Better, Together [Online]. Available from: https://github.com/ [2013, April 15]

[28] IBM. IBM Research - Tokyo | Core Research Competency | Java JIT Compiler [Online]. Available from: http://www.research.ibm.com/trl/projects/jit/index_e.htm [2013, January 14]

[29] Mozilla Developer Network. JavaScript Typed Arrays - JavaScript | MDN [Online]. Avaliable from: https://developer.mozilla.org/en-US/docs/JavaScript/Typed_arrays [2013, January 11]

[30] Yee, B., D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native Client: A Sandbox for Portable, Untrusted X86 Native Code. In Security and Privacy, 2009 30th IEEE Symposium.

[31] Khronos Group. WebCL - Heterogeneous Parallel Computing in HTML5 Web Browsers [Online]. Available from: http://www.khronos.org/webcl/ [2013, January 13]

[32] Khronos Group. OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems [Online]. Available from: http://www.khronos.org/opencl/ [2013, January 13]

[33] NRC Tampere. WebCL [Online]. Available from: http://webcl.nokiaresearch.com/ [2013, January 13]

[34] Google Project. Webcl - WebCL for WebKit - Google Project Hosting [Online]. Available from: http://code.google.com/p/webcl/ [2013, January 13]

[35] Mark Pilgrim. Local Storage - Dive Into HTML5 [Online]. Available from http://diveintohtml5.info/storage.html [2013, January 11]

[36] HTML5 Rocks. Exploring the FileSystem APIs - HTML5 Rocks [Online]. Available from: http://www.html5rocks.com/en/tutorials/file/filesystem/ [2012, July 25]

[37] Chen, S., and S. W Schlosser. Map-reduce Meets Wider Varieties of Applications. Intel Research Pittsburgh, Tech. Rep. IRP-TR-08-05, 2008.

[38] Verma, Abhishek, Xavier Llorà, David E. Goldberg, and Roy H. Campbell. Scaling Genetic Algorithms Using MapReduce. In 2009 Ninth International Conference on Intelligent Systems Design and Applications.

[39] Liu, Z., H. Li, and G. Miao. MapReduce-based Backpropagation Neural Network over Large Scale Mobile Data. In Natural Computation (ICNC), 2010 Sixth International Conference, pp.1726–1730.

[40] Tordable, J. MapReduce for Integer Factorization [Online]. Avaliable from: http://www.javiertordable.com/files/MapreduceForIntegerFactorization .pdf. [2013, January 11]

[41] Cascading | Application Platform for Enterprise Big Data [Online]. Avaliable from: http://www.cascading.org [2012, July 25]

[42] Google Project. MurmurHash3 - Smhasher - MurmurHash3 Information and Brief Performance Results - Test Your Hash Functions. - Google Project Hosting [Online]. Available from: http://code.google.com/p/ smhasher/wiki/MurmurHash3 [2013, April 19]

[43] Stack Overflow. Which Hashing Algorithm Is Best for Uniqueness and Speed? – Programmers [Online]. Available from: http://programmers. stackexchange.com/questions/4 9 5 5 0 / which-hashing-algorithm-is-best-for-uniqueness-and-speed/145633#145633. [2012, July 25]

[44] Rice, Justin L., Vir V. Phoha, Pat Cappelaere, and Dan Mandl. Web Farm-inspired Computational Cluster in the Cloud." In CloudCom 2011

[45] Fal Labs. Kyoto Tycoon: a Handy Cache/storage Server [Online]. Available from: http://fallabs.com/kyototycoon/ [2012, July 25]

[46] Ilya Grigorik. Tokyo Cabinet: Beyond Key-Value Store - Igvita.com [Online]. Available from: http://www.igvita.com/2009/02/13/tokyo-cabinet-beyond-key-value-store/ [2013, April 19]

[47] Ogawa, H., H. Nakada, R. Takano, and T. Kudoh. Sss: An Implementation of Key-value Store Based Mapreduce Framework. In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference, pp.754–761.

[48] Wolfram. Wolfram Demonstrations Project: Mean Value Theorem for Integrals and Monte Carlo Integration [Online]. Available from: http://demonstrations.wolfram.com/MeanValueTheoremForIntegralsAndMonteCarloIntegration/ [2012, July 25]

[49] UTas ePrints. UTas ePrints - Distributed Computing and Communication in Peer-to-peer Networks [Online]. Available from: http://eprints.utas.edu.au/9733/ [2013, April 20]

[50] Yi, Sangho, Derrick Kondo, and David P. Anderson. Toward Real-time, Many-task Applications on Large Distributed Systems. In Proceedings of the 16th International Euro-Par Conference on Parallel Processing

[51] Tildeslash Ltd. Monit [Online]. Available from: http://mmonit.com/monit/ [2013, April 14].

[52] Computing Pi [Online]. Available from: ftp://ftp.kernel.org/pub/linux/kernel/people/paulmck/Answers/pi.html [2013, May 8]

[53] Wikipedia. Browser wars - Wikipedia, the free encyclopedia [Online]. Available from: http://en.wikipedia.org/wiki/Browser_wars [2013, Jan 11]

[54] Google Developers. Closure Tools — Google Developers [Online]. Available from: https://developers.google.com/closure/compiler/ [2013, May 9]

[55] A. Zakai. Emscripten: an LLVM-to-JavaScript compiler. in Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, 2011, pp. 301–312.

[56] University of California, Berkeley . The computational and storage potential of volunteer computing [Online]. Avaliable from: http://boinc.berkeley.edu/boinc_papers/internet/paper.pdf [2013, April 19]

ภาคผนวก

ภาคผนวก ก.
การติดตั้งเฟรมเวิร์ก WebGrid.js

ภาคผนวก ก.เป็นการแสดงขั้นตอนการติดตั้งเฟรมเวิร์ก WebGrid.js โดย WebGrid.js นั้นทำงานบนแพลตฟอร์ม Node.js และใช้ฐานข้อมูล MySQL เป็นฐานข้อมูลหลัก หากต้องการใช้รูปแบบโปรแกรมแบบ MapReduce บน WebGrid.js นั้น จำเป็นต้องติดตั้งฐานข้อมูล Kyoto Tycoon (ขณะนี้ฐานข้อมูลนี้รองรับเฉพาะระบบปฏิบัติการ Linux เท่านั้น) เพิ่มเติมด้วย

ขั้นตอนการติดตั้ง Node.js

Node.js นั้นรองรับได้หลายระบบปฏิบัติการ ซึ่งในภาคผนวกนี้ได้แสดงขั้นตอนการติดตั้งบนระบบปฏิบัติการ Windows และ Ubuntu โดยขั้นตอนการติดตั้ง Node.js มีดังนี้

1. ติดตั้งฐานข้อมูล MySQL
2. ดาวน์โหลดโปรแกรม Node.js จากเว็บไซท์ http://www.nodejs.org โดยสามารถแบ่งเป็นขั้นตอนสำหรับแต่ละระบบปฏิบัติการได้ดังนี้

   - Windows

   ดาวน์โหลดไฟล์ติดตั้งโปรแกรม Node.js และสามารถติดตั้งได้ทันที

   - Ubuntu

   ดาวน์โหลดรหัสโปรแกรม (source code) , แตกไฟล์ไปยังตำแหน่งที่ต้องการ และพิมพ์คำสั่งเหล่านี้ตามลำดับ

   ```
   ./configure
   make
   make install
   ```

3. สร้างโฟลเดอร์สำหรับโครงการ โดยโฟลเดอร์นี้จะเก็บไฟล์โปรแกรมของเฟรมเวิร์กและโมดูลต่างๆของ Node.js
4. ติดตั้งโมดูลต่างๆของ Node.js โดย Node.js มีโปรแกรมที่ชื่อว่า npm ไว้สำหรับจัดการโมดูลต่างๆสำหรับโครงการนั้นๆ วิธีการติดตั้งโมดูลสามารถทำได้โดยการเข้าไปที่โฟลเดอร์ที่ได้สร้างไว้ในข้อ 3.) และพิมพ์คำสั่งดังนี้

   ```
   npm install <ชื่อโมดูล>@<เวอร์ชั่นที่ต้องการ>
   ```

   โดยรายชื่อโมดูลที่จำเป็นมีดังนี้

   - express@2.5.11 เป็นโมดูลที่ช่วยให้การสร้าง restful เว็บเซอร์วิสสามารถทำได้ง่ายขึ้น

- mysql@0.9.6 โมดูลสำหรับติดต่อกับฐานข้อมูล MySQL
- request@2.9.203 โมดูลช่วยสำหรับการทำ HTTP request

5. ลงโมดูล forever เพื่อใช้ในการเฝ้าโปรเซสในฝั่งเซิร์ฟเวอร์ ในกรณีที่ต้องการ debug โปรแกรมฝั่งเซิร์ฟเวอร์ ให้ติดตั้งโมดูล node-inspector ไว้ด้วย โดยการพิมพ์คำสั่งต่อไปนี้

```
npm install -g forever
npm install -g node-inspector@0.1.10
```

<u>ขั้นตอนการติดตั้งเฟรมเวิร์ก WebGrid.js</u>

1. ดาวน์โหลดรหัสโปรแกรมของเฟรมเวิร์ก WebGrid.js ได้ที่ https://bitbucket.org/apemon/webgrid.js-release
2. แตกไฟล์ไปยังโฟลเดอร์ที่ได้สร้างไว้ ทำการแก้ไขไฟล์ mysql.js ให้เป็นค่าฐานข้อมูลที่ต้องการ
3. แก้ไขค่าไฟล์ config.js ให้เป็นรูปแบบที่ต้องการ โดยสามารถกำหนดค่าได้ดังนี้
   - scheduler_url รายการ url ของ scheduler
   - mount ตำแหน่งที่เก็บไฟล์งานต่างๆ โดยใน 1 ตำแหน่งจะมีค่าต่างๆดังนี้
     - logical_path พาธสำหรับใช้อ้างอิงในเฟรมเวิร์ก
     - physical_path ตำแหน่งของพาธ
     - read_permission สิทธิในการอ่านไฟล์
     - write_permission สิทธิในการเขียนไฟล์
     -url สำหรับใช้ในการเรียกเข้ามาจากภายนอก โดยปกติมักใช้ url เดียวกับโมดูล fileservice
   - workpool_url ตำแหน่ง url ของโมดูล workpool
   - host_enable เปิดการใช้งานการเก็บข้อมูลของ host
   - intermediate_delete ลบไฟล์ intermediate หลังจากหาผลลัพธ์สุดท้ายเสร็จแล้ว
   - ktserver_url ตำแหน่งที่อยู่ของฐานข้อมูล Kyoto Tycoon
   - ktserver_port ตำแหน่งพอร์ทของฐานข้อมูล Kyoto Tycoon
4. นำเข้าไฟล์ webgrid.sql และ platform.sql เข้าฐานข้อมูล MySQL ตามลำดับ
5. เข้าไปยังโฟลเดอร์ mockdata และพิมพ์คำสั่ง

```
node reset.js
```

เพื่อล้างข้อมูลเก่า และบันทึกค่าจากไฟล์ config.js ลงฐานข้อมูล

ขั้นตอนการติดตั้งฐานข้อมูล Kyoto Tycoon (ในกรณีที่ต้องการใช้งาน MapReduce)

การติดตั้งฐานข้อมูล Kyoto Tycoon จำเป็นต้องติดตั้งไลบรารี่ Kyoto Cabinet ก่อน โดยขั้นตอนการติดตั้งมีดังนี้

1. ดาวน์โหลดรหัสโปรแกรม Kyoto Cabinet จากเว็บไซท์ http://fallabs.com/kyotocabinet/
2. เข้าไปยังโฟลเดอร์รหัสโปรแกรมที่ได้ดาวน์โหลดไว้ และพิมพ์คำสั่งต่อไปนี้

```
./configure –enable-static
make
make install
```

3. ดาวน์โหลดรหัสโปรแกรม Kyoto Tycoon จากเว็บไซท์ http://fallabs.com/kyototycoon/ เข้าไปยังโฟลเดอร์ที่ได้ดาวน์โหลดไว้ และพิมพ์คำสั่งต่อไปนี้

```
./configure –enable-static
make
make install
```

4. ทดสอบการใช้งานโปรแกรมได้โดยการลองพิมพ์คำสั่งต่อไปนี้

```
ktserver <ชื่อฐานข้อมูล1>.kch <ชื่อฐานข้อมูล2>.kch
```

ตัวอย่างการใช้งานโปรแกรม WebGrid.js (โปรแกรม superPI)

ตัวอย่างการใช้งานโปรแกรม WebGrid.js โดยแสดงการสร้างงาน superPI ได้ดังนี้

1. สร้างไฟล์อินพุต (ไฟล์ที่มีตัวเลขจำนวนรอบที่จะให้ไคลเอนต์ทำการสุ่ม) พร้อมทั้งตั้งค่าพาธ (สามารถตั้งได้ที่ตาราง configuration ค่า mount ในฐานข้อมูล)
2. เข้าไปยังโฟลเดอร์หลักของ WebGrid.js เข้าไปที่โฟลเดอร์ mockdata แล้วเปิดไฟล์ superpi.js โดยสามารถใช้ไฟล์นี้เป็นตัวอย่างในการสร้างโครงการอื่นๆได้ โดยส่วนสำคัญในไฟล์ superpi.js จะเป็นดังนี้

```
var superpi = new job({
      name:'superpi',
      description:'superpi calculator’,
      input_directory:'/pi_in',
      output_directory:'/pi_out',
      sort_method:'none',
      output_type:'store',
      max_result:3,
      majority:1,
      type:'custom',
```

```
      validation_type:'custom',
      error_rate:1,
      checkpoint: true,
      deadline : true,
      deadline_duration: 1800000,
      custom: รหัสโปรแกรมของงาน
});
```

โดยโปรแกรมส่วนนี้จะเป็นการกำหนดค่าต่างๆของ job ในส่วนของรหัสโปรแกรมของงาน superPI สามารถแสดงได้ดังนี้

```
function execute(input){
      var iteration_num = input;
      var hit = 0;
      var i = 0;
      // load checkpoint first
      var cpobj = load_checkpoint();
      if(typeof(cpobj)!='undefined'){
            i = cpobj.index;
            hit = cpobj.hit;
      }
      for(;i<iteration_num;i++){
            if(i%100000000 == 0){
                  cpobj = {};
                  cpobj['hit'] = hit;
                  cpobj['index'] = i;
                  save_checkpoint(cpobj);

      report((i/iteration_num)*100);
            }
            x = Math.random();
            y = Math.random();
            d = Math.pow(x,2) + Math.pow(y,2);
            if(d <= 1) hit++;
      }
      output_send(hit);
}
```

ขั้นตอนของโปรแกรมดังกล่าวมีดังนี้

- เริ่มต้นดึงค่าจุดตรวจสอบที่มีด้วยฟังก์ชั่น load_checkpoint หากมีจุดตรวจสอบของงานก็นำค่าจุดตรวจสอบมาใช้
- วนรอบหาค่า pi ด้วยวิธีการมอนติคาร์โล หากถึงจำนวนรอบที่หารด้วย 100,000,000 ก็จะบันทึกจุดตรวจสอบโดยการใช้ฟังก์ชั่น save_checkpoint พร้อมทั้งรายงานความก้าวหน้าของงานโดยการใช้ฟังก์ชั่น report
- เมื่อวนรอบเสร็จสิ้นแล้ว โปรแกรมก็จะส่งผลลัพธ์กลับไปยังเซิร์ฟเวอร์ด้วยคำสั่ง output_send

ในส่วนสุดท้ายของไฟล์เป็นส่วนของคำสั่งในการสร้าง job และ task เนื่องจากงานนี้มีไฟล์อินพุตของระบบเพียงไฟล์เดียว จึงต้องใช้คำสั่งในการสร้าง task เอง (โดยปกติโมดูล task manager จะสร้าง 1 task ต่อ 1 ไฟล์อินพุต)

```
Record.init(mysql_info);
Record.Job.insert(superpi,function(err,results){
      if(err) return;
      Record.Task.createWork('superpi',
'in',2000,function(err,results){
            Record.Job.start('superpi');
      });
});
```

คำสั่ง Record.Job.insert จะเป็นการบันทึกงานลงฐานข้อมูล ส่วนคำสั่ง Record.Task.createWork จะเป็นคำสั่งสำหรับสร้าง task สำหรับไฟล์อินพุตชื่อ in จำนวน 2,000 task และ Record.Job.start เพื่อให้ job เริ่มทำงาน

3. แก้ไขค่าตามที่ต้องการ เสร็จแล้วจึงพิมพ์คำสั่งนี้ เพื่อให้สคริปท์ป้อนงานเข้าสู่ระบบ

```
node superpi.js
```

4. เปิดการทำงานของโมดูลอื่นๆ เนื่องจากงานนี้มีเพียงแค่ job เดียว จึงไม่จำเป็นมีโมดูล task manager คำสั่งที่ใช้ในการเปิดโมดูลต่างๆมีดังนี้

```
cd .. // back to main project directory
forever start –o log/workpool workpool.js
forever start –o log/scheduler scheduler.js
forever start –o log/validator validator.js
forever start –o log/collector collector.js
forever start fileservice.js
```

การใช้คำสั่ง forever เป็นการเรียกใช้โมดูล forever ให้ช่วยสังเกตการณ์ทำงานของโปรเซส หากโปรเซสนั้นล่ม โมดูล forever จะเริ่มการทำงานโปรเซสนั้นใหม่ทันที

5. เปิดเว็บเบราว์เซอร์แล้วพิมพ์ http://localhost:1234 (ไอพีหรือพอร์ทอื่นๆที่ได้ตั้งค่าไว้)

<u>ตัวอย่างการใช้งานโปรแกรม WebGrid.js (โปรแกรม wordcount)</u>

ตัวอย่างการใช้งานโปรแกรม WebGrid.js โดยแสดงการสร้างงาน wordcount ได้ดังนี้

1. เตรียมไฟล์อินพุตไปไว้ยังโฟลเดอร์ที่ต้องการ หากไม่มีไฟล์อินพุต สามารถใช้สคริปท์ wordcountgen.js ที่อยู่ในโฟลเดอร์ mockdata ในการสร้างไฟล์อินพุตขึ้นมาได้ โดยวิธีการใช้งานคือ ให้คัดลอกไฟล์ wordcountgen.js และไฟล์ word.txt ไปยังตำแหน่งที่ต้องการ แล้วจึงใช้คำสั่ง

```
node wordcountgen.js จำนวนบรรทัด จำนวนไฟล์ ชื่อไฟล์(prefix)
```

2. เข้าไปที่โฟลเดอร์ mockdata แล้วเปิดไฟล์ wordcount.js โดยเนื้อหาในไฟล์จะประกอบไปด้วย 2 job คือ count word และ sum ซึ่งสามารถแสดงได้ดังนี้

```
var wordcount_map = new job({
      name:'count word',
      description:'wordcount',
      input_directory:'/wordin',
      output_directory:'/wordout',
      sort_method:'alpha_asc',
      output_type:'partition',
      max_result:5,
      majority:3,
      type:'mapreduce',
      validation_type:'checksum',
      checkpoint: true,
      checkpoint_interval: 1000,
      deadline : true,
      partition_number: 5,
      input_reader: คำสั่งสำหรับการอ่านค่าอินพุต
      algorithm: คำสั่งในการประมวลผลแต่ละค่า key-value
      combiner: คำสั่งในการรวมค่าที่มี key เหมือนกัน
});
```

โดยหากไม่ใส่ค่า input_reader และ combiner ระบบจะใช้ค่าพื้นฐานคือ อ่านค่าไฟล์อินพุตทีละบรรทัดในส่วนของ input_reader และคำนวณผลรวมของค่า value ในส่วนของ combiner การทำงานของ algorithm สามารถแสดงได้ดังนี้

```
function algorithm(key,value){
      var words = value.split(' ');
      for(var i=0;i<words.length;i++){
            emit(words[i],1);
      }
}
```

ส่วนงาน sum จะมีรหัสโปรแกรมดังนี้

```
var wordcount_reduce = new job({
      name:'sum',
      description:'wordcount',
      input_directory:'/wordout',
      output_directory:'/wordout',
      sort_method:'alpha_asc',
      output_type:'store',
      max_result:5,
      majority:3,
      type:'mapreduce',
      validation_type:'checksum',
      checkpoint: true,
      checkpoint_interval: 1000,
      deadline : true,
      input_reader: คำสั่งสำหรับการอ่านค่าอินพุต
      algorithm: คำสั่งในการประมวลผลแต่ละค่า key-value
      combiner: คำสั่งในการรวมค่าที่มี key เหมือนกัน
```

```
});
```

รหัสของฟังก์ชั่นในงาน sum มีดังนี้

```
function algorithm(key,value){
      var sum = 0;
      for(var i=0;i<value.length;i++){
            sum += value[i];
      }
      emit(key,sum);
}
```

ในส่วนสุดท้ายของไฟล์เป็นส่วนของคำสั่งในการสร้าง job และสร้าง workflow ตามลำดับ โดยรูปแบบคำสั่งมีดังนี้

```
Record.init(mysql_info);
Record.Job.insert(wordcount_map,function(err,results){
      if(err) return;
      Record.Job.insert(wordcount_reduce,function(err2,results2){
            if(err2) return;
            Record.Workflow.connect(1,2,function(err3,results3){
                  if(err3) return;
                  return console.log(results3);
            });
      });
});
```

คำสั่ง Record.Job.insert จะเป็นการบันทึกงานลงฐานข้อมูล โดยจะใส่งาน count word ก่อน แล้วจึงค่อยใส่งาน sum ตามลำดับ สุดท้ายจึงสร้าง workflow ระหว่าง 2 งานตามลำดับ

3. ใส่งานเข้าไปในระบบ โดยการพิมพ์คำสั่งต่อไปนี้

```
node wordcount.js
```

4. เปิดการทำงานของโมดูลอื่นๆ โดยคำสั่งที่ใช้ในการเปิดโมดูลต่างๆมีดังนี้

```
cd .. // back to main project directory
forever start -o log/taskmgr taskmgr.js
forever start -o log/workpool workpool.js
forever start -o log/scheduler scheduler.js
forever start -o log/validator validator.js
forever start -o log/collector collector.js
forever start fileservice.js
```

5. เนื่องจากงานนี้มีการใช้งานฐานข้อมูล key-value ในการทำข้อมูล partition ด้วย ดังนั้นจึงต้องมีการเปิดการใช้งานฐานข้อมูล key-value โดยจำนวนของฐานข้อมูลจะเท่ากับค่า partition_number ที่เราได้ตั้งไว้

```
ktserver part_0.kch part_1.kch … part_n.kch
```

6. ไคลเอนต์เปิดเว็บเบราว์เซอร์ไปยังตำแหน่ง http://localhost:1234 เพื่อเริ่มการประมวลผล

การ debug โปรแกรม

- ฝั่งไคลเอนต์ สามารถทำได้โดยการใช้ระบบ debug ของแต่ละเว็บเบราว์เซอร์ ตัวอย่างเช่น Google Chrome สามารถ debug ได้โดยการ คลิกขวาเลือก Inspect element เลือกไปที่แท็บ source จากนั้นจึงทำการวางจุด breakpoint และ debug ตามที่ต้องการ
- ฝั่งเซิร์ฟเวอร์นั้น สามารถทำได้โดยการพิมพ์คำสั่ง

```
node-inspector
```

จากนั้นทำการเริ่มการทำงานโมดูลที่ต้องการ debug โดยพิมพ์คำสั่งดังนี้

```
node –debug-brk ชื่อไฟล์ที่ต้องการ
```

เปิดเว็บเบราว์เซอร์ และเข้าไปยังตำแหน่ง http://localhost:8080 เพื่อทำการ debug รหัสโปรแกรมบนฝั่งเซิร์ฟเวอร์

ภาคผนวก ข.
รายละเอียดของรูปแบบข้อมูลที่ใช้ใน WebGrid.js

ภาคผนวก ข.เป็นการแสดงรายละเอียดของรูปแบบข้อมูลที่ใช้ใน WebGrid.js โดยจะสามารถแบ่งได้เป็นดังนี้

job

id	รหัสของงาน
name	ชื่อของงาน
description	คำอธิบายเพิ่มเติมเกี่ยวกับตัวงาน
type	ประเภทของงาน โดยจะแบ่งออกเป็น 2 ประเภทคือ custom และ mapreduce
status	สถานะของงาน
create_time	ค่า timestamp ณ เวลาที่งานนี้ได้ถูกสร้างขึ้น
last_mod_time	ค่า timestamp ณ เวลาที่งานนี้ได้ถูกแก้ไขครั้งล่าสุด
complete_time	ค่า timestamp ณ เวลาที่งานนี้เสร็จสมบูรณ์
input_directory	ตำแหน่งที่อยู่ของไฟล์อินพุต โดยจะอ้างอิงเป็น logical path
output_directory	ตำแหน่งที่อยู่ของไฟล์เอาต์พุต โดยจะอ้างอิงเป็น logical path
sort_method	วิธีการจัดเรียงค่า key (เฉพาะงานแบบ mapreduce เท่านั้น)
output_type	ประเภทของเอาต์พุต โดยจะมีอยู่ 2 แบบคือ store และ partition
partion_number	จำนวน partition ของผลลัพธ์ (สำหรับ output_type แบบ partition)
validation_type	วิธีการตรวจสอบผลลัพธ์ที่ได้จากไคลเอนต์ โดยในเฟรมเวิร์กนี้จะมี checksum เทียบค่า MD5 checksum จากผลลัพธ์, avg หาค่าเฉลี่ยของผลลัพธ์ (ในกรณีที่เป็นตัวเลข) และ custom ซึ่งเป็นวิธีที่ผู้พัฒนาได้กำหนดขึ้นเอง ผ่านทางไฟล์ custom_validator.js
error_rate	ค่าความแตกต่างของผลลัพธ์ (สำหรับวิธีการตรวจสอบแบบ avg) โดยคิดเป็นค่า SD (ตัวอย่างเช่น ค่า error_rate = 0.5 จะหมายถึงผลลัพธ์ที่ถือว่าถูกต้องนั้นต้องไม่แตกต่างจากค่าส่วนใหญ่เกิน 0.5 SD)
max_result	จำนวน result สูงสุดของ task ในงานนี้

majority	จำนวนขั้นต่ำที่จะถือว่าผลลัพธ์ถูกต้อง
checkpoint	การใช้งานระบบ checkpoint อัตโนมัติ (สำหรับงานแบบ mapreduce)
checkpoint_interval	จำนวนรอบก่อนที่จะทำ checkpoint (ตัวอย่างเช่น 1,000 หมายถึงทำ checkpoint ทุกๆค่า key ที่ 1,000)
deadline	การใช้งานระบบ deadline
deadline_duration	ระยะเวลาของ deadline (มีหน่วยเป็น ms)
input_reader	ฟังก์ชั่น input_reader สำหรับงานแบบ mapreduce
algorithm	ฟังก์ชั่น algorithm สำหรับงานแบบ mapreduce
combiner	ฟังก์ชั่น combiner สำหรับงานแบบ mapreduce
custom	ฟังก์ชั่นการทำงานของ worker

task

id	รหัสของ task
status	สถานะของ task
create_time	ค่า timestamp ณ เวลาที่ task นี้ได้ถูกสร้างขึ้น
validate_time	ค่า timestamp ณ เวลาที่ task นี้ได้ถูกตรวจสอบข้อมูลผลลัพธ์
file_write_time	ค่า timestamp ณ เวลาที่ไฟล์เอาต์พุตของ task นี้ได้ถูกสร้างขึ้น
partition_time	ค่า timestamp ณ เวลาที่ผลลัพธ์ของ task นี้ได้ถูก partition แล้ว
complete_time	ค่า timestamp ณ เวลาที่ task นี้เสร็จสมบูรณ์
last_mod_time	ค่า timestamp ณ เวลาที่ task นี้ได้ถูกแก้ไขครั้งล่าสุด
job_id	ค่า id ของ job
candidate_result	ตัวแทน result ของ task นี้
input_path	ชื่อไฟล์อินพุตของ task นี้
output_checksum	ค่า MD5 checksum ของไฟล์เอาต์พุต
output_path	ตำแหน่งที่อยู่ของไฟล์เอาต์พุต

result

id	รหัสของ result
status	สถานะของ result
create_time	ค่า timestamp ณ เวลาที่ result นี้ได้ถูกสร้างขึ้น
dispatch_time	ค่า timestamp ณ เวลาที่ result นี้ได้ถูกแจกจ่ายให้ไคลเอนต์ (บันทึกที่ฝั่งเซิร์ฟเวอร์)
execution_time	เวลาที่ใช้ในการประมวลผลบนเครื่องไคลเอนต์
receive_time	ค่า timestamp ณ เวลาที่เซิร์ฟเวอร์ได้ result นี้จากไคลเอนต์ (บันทึกที่ฝั่งเซิร์ฟเวอร์)
client_send_time	ค่า timestamp ณ เวลาที่ไคลเอนต์ส่ง result กลับไปยังเซิร์ฟเวอร์ (บันทึกที่ฝั่งไคลเอนต์)
client_receive_time	ค่า timestamp ณ เวลาที่ไคลเอนต์ได้รับ result จากเซิร์ฟเวอร์ (บันทึกที่ฝั่งไคลเอนต์)
deadline_time	ค่า timestamp ณ เวลาที่เป็น deadline ของ result นี้
task_fetch	เวลาที่ใช้ในการร้องของานจากเซิร์ฟเวอร์
input_fetch	เวลาที่ใช้ในการดาวน์โหลดไฟล์อินพุต
result_send	เวลาที่ใช้ในการส่งผลลัพธ์กลับไปยังเซิร์ฟเวอร์
secret	ข้อความสุ่มความยาว 32 ตัวอักษร (ใช้กันไคลเอนต์ที่ไม่หวังดี สร้าง result ปลอมขึ้นมาเพื่อหลอกเซิร์ฟเวอร์)
user_agent	ข้อความ user agent ของไคลเอนต์
remote_address	ค่าหมายเลขไอพีของไคลเอนต์
host_id	ค่า id ของเครื่องไคลเอนต์
task_id	ค่า id ของ task
job_id	ค่า id ของ job
output_path	ตำแหน่งที่อยู่ของไฟล์เอาต์พุต
output_checksum	MD5 checksum ของไฟล์เอาต์พุต

workflow

id	รหัสของ workflow
source	job ต้นทาง
destination	job ปลายทาง

host

id	รหัสของ host
authen	รหัสผ่านของเครื่องไคลเอนต์
platform	แพลตฟอร์มของเครื่องไคลเอนต์ (ในที่นี้แบ่งตามเว็บเบราว์เซอร์)
version	เวอร์ชั่นของเว็บเบราว์เซอร์
timezone_offset	ค่า offset ของ timezone บนเครื่องไคลเอนต์
last_ip_addr	ค่าหมายเลขไอพีล่าสุดของเครื่องไคลเอนต์
create_time	ค่า timestamp ณ เวลาที่ host นี้ได้ถูกสร้างขึ้น
expire_time	ค่า timestamp ณ เวลาที่ host นี้หมดอายุ (ในกรณีที่ host หมดอายุแล้ว โมดูล scheduler จะดำเนินการปรับปรุงข้อมูลของ host ใหม่)
last_mod_time	ค่า timestamp ณ เวลาที่ host นี้ได้ถูกแก้ไขครั้งล่าสุด
last_active_time	ค่า timestamp ณ เวลาที่ host นี้ได้ติดต่อมายังเซิร์ฟเวอร์ครั้งล่าสุด
os_platform	ระบบปฏิบัติการที่ไคลเอนต์ใช้
flops	ค่า flops ของเครื่องไคลเอนต์
user_agent	ข้อความ user agent ของเครื่องไคลเอนต์

ประวัติผู้เขียนวิทยานิพนธ์

นายปริน เจียมอนันตพงศ์ เกิดเมื่อวันที่ 8 มีนาคม พ.ศ. 2532 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2553 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2554