การปรับปรุงอัลกอริทึมการจัดสรรงานสำหรับฮาดูปคลัสเตอร์แบบต่างชนิดโดยใช้หลักการควบคุมการแออัด

นายอภิลักษณ์ ทรงวัฒนาสกุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2556

AN IMPOROVEMENT OF SCHEDULING ALGORITHM FOR HETEROGENEOUS HADOOP CLUSTER USING CONGESTION CONTROL CONCEPT

Mr. Apiluck Songwattanasakul

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2013

หัวข้อวิทยานิพนธ์	การปรับปรุงอัลกอริทึมการจัดสรรงานสำหรับฮาดูปคลัสเตอร์แบบต่างชนิดโดยใช้หลักการควบคุมการแออัด
โดย	นายอภิลักษณ์ ทรงวัฒนาสกุล
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร. ณัฐวุฒิ หนูไพโรจน์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(ศาสตราจารย์ ดร. บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร. วีระ เหมืองสิน)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ผู้ช่วยศาสตราจารย์ ดร. ณัฐวุฒิ หนูไพโรจน์)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร. เกริก ภิรมย์โสภา)

.....กรรมการภายนอกมหาวิทยาลัย

(ผู้ช่วยศาสตราจารย์ ดร. ภุชงค์ อุทโยภาศ)

อภิลักษณ์ ทรงวัฒนาสกุล : การปรับปรุงอัลกอริทึมการจัดสรรงานสำหรับฮาดูปคลัสเตอร์แบบต่างชนิดโดยใช้หลักการควบคุมการแออัด. (AN IMPOROVEMENT OF SCHEDULING ALGORITHM FOR HETEROGENEOUS HADOOP CLUSTER USING CONGESTION CONTROL CONCEPT) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร. ณัฐวุฒิ หนูไพโรจน์, 73 หน้า.

ฮาดูปเป็นโอเพนซอร์ซภายใต้การประมวลผลแบบเมพรีดิวที่ถูกใช้ในการประมวลผลข้อมูลขนาดใหญ่อย่างแพร่หลายโดยมีสมมติฐานที่อยู่บนหลักการพื้นฐานของคลัสเตอร์แบบเอกพันธุ์ แต่ปัจจุบันการขยายคลัสเตอร์โดยการใช้เทคโนโลยีคลาวด์ได้รับความนิยมทำให้คลัสเตอร์มีลักษณะแบบต่างชนิด ประกอบด้วยเครื่องที่หลากหลายทำให้การทำงานของฮาดูปบนคลัสเตอร์เหล่านี้ ไม่มีประสิทธิภาพที่ดีเท่าที่ควร งานวิจัยนี้จึงได้นำเสนอแนวทางการแก้ปัญหาด้วยอัลกอริทึมการแจกงานเมื่อคลัสเตอร์เป็นแบบต่างชนิด ซึ่งอัลกอริทึมที่นำเสนอ จะพิจารณาถึงประสิทธิภาพความเร็วของอุปกรณ์และการประมวลผล รวมไปถึงคุณสมบัติของงานที่เข้าสู่ระบบเพื่อการประมวลผล มาใช้เป็นพื้นฐานในการแจกงานโดยอ้างอิงหลักการการควบคุมการแออัดในระบบเครือข่าย เพื่อให้เกิดประสิทธิภาพสูงสุดในการแจกงาน

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2556

5571004021 : MAJOR COMPUTER SCIENCE
KEYWORDS: HADOOP / SCHEDULER / PARALLEL COMPUTING / BIG DATA

APILUCK SONGWATTANASAKUL: AN IMPOROVEMENT OF SCHEDULING ALGORITHM FOR HETEROGENEOUS HADOOP CLUSTER USING CONGESTION CONTROL CONCEPT. ADVISOR: ASST. PROF. NATAWUT NUPAIROJ, Ph.D., 73 pp.

Hadoop is the opensource software based on MapReduce Algorithm, which is widely used to support big data processing. Hadoop's effectiveness is based on its assumption of using homogeneous cluster. However, cluster expansion based on cloud technology is quite popular recently. Thus, these clusters become heterogeneous with various types of machines. This causes Hadoop to become ineffective when running on these clusters. To solve this problem, this research proposes a Hadoop's workload distribution algorithm on heterogeneous cluster. Our proposed algorithm considers the performance of I/O and processing capability, as well as, the nature of submitted jobs as basis for workload distribution based on the concept of network congestion control to maximize the efficiency of running Hadoop jobs on heterogeneous cluster.

Department: Computer Engineering Student's Signature .....
Field of Study: Computer Science Advisor's Signature .....
Academic Year: 2013

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความอนุเคราะห์อย่างดียิ่งจาก ผู่ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ได้สละเวลาให้ความรู้ คำปรึกษา ตรวจสอบและแก้ไขข้อผิดพลาดต่างๆ ตลอดจนการกำกับดูแลและคอยติดตามความก้าวหน้า ทำให้การวิจัยนี้สำเร็จไปได้ด้วยดี ผู้วิจัยขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ โอกาสนี้

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน, ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา และ ผู้ช่วยศาสตราจารย์ ดร.ภุชงค์ อุทโยภาศ กรรมการสอบวิทยานิพนธ์ ที่กรุณาสละเวลาให้คำแนะนำ ตรวจสอบ และแก้ไขวิทยานิพนธ์ฉบับนี้

ขอขอบพระคุณบิดา มารดา และญาติพี่น้องที่เป็นกำลังใจและสนับสนุนด้านทุนทรัพย์ในการศึกษา รวมไปถึงทุกท่านที่มีส่วนช่วยเหลือในการทำวิทยานิพนธ์ครั้งนี้ ซึ่งมิได้กล่าวนามในที่นี้

สุดท้ายนี้ หากมีสิ่งใดขาดตกบกพร่องหรือข้อผิดพลาดประการใด ผู้วิจัยขออภัยเป็นอย่างสูงในข้อบกพร่องและความผิดพลาดนั้น และหวังเป็นอย่างยิ่งว่าวิทยานิพนธ์ฉบับนี้จะเป็นประโยชน์สำหรับผู้ที่สนใจจะศึกษารายละเอียดต่อไป

สารบัญ

หน้า

หน้า

สารบัญตาราง

หน้า

สารบัญรูปภาพ

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันการประมวลผลบนอินเตอร์เน็ตในลักษณะการทำงานของ Search Engine, Social Network และ Cloud Computing Service นั้นมีปริมาณข้อมูลที่มีขนาดใหญ่มาก โดยมีขนาดตั้งแต่ 100 TB ไปถึง 100 PB ซึ่งจะต้องมีวิธีการในการจัดการและประมวลผลข้อมูลเหล่านี้ให้อยู่ในรูปแบบข้อมูลที่ต้องการ (Data Structure) ได้มีผู้พัฒนาโอเพนซอร์ซ ฮาดูปที่ถูกพัฒนาโดย Apache Community ได้รับต้นแบบการสร้างจากแนวคิดของกูเกิลโดยจุดประสงค์เพื่อจัดหาวิธีการประมวลผลในขนาดของข้อมูลที่มีขนาดใหญ่มาก (Big Data Processing) ภายใต้แนวคิดที่ว่าการประมวลผลไม่จำเป็นต้องใช้ส่วนเก็บข้อมูลที่มีราคาแพงหรือฮาร์ดแวร์ราคาสูง แต่จะเน้นให้ทุกเครื่องทำหน้าที่ทั้งการประมวลผลและเก็บข้อมูลด้วยรวมไปถึงการขยายขนาดของคลัสเตอร์ทำได้ง่าย อีกทั้งฮาดูปยังสามารถประมวลผลข้อมูลได้หลากหลายชนิดดังจะเห็นได้จากงานวิจัยมากมายนำฮาดูปไปใช้ในการประมวลผลทางด้านต่างๆ เช่น Information Storage and Retrieval [1] [2] , Web Search Engine [3] และ Image Processing [4] ริเริ่มโดยบริษัทกูเกิล ได้ใช้ GFS File System [5] และ Big Table Processing [6] หลังจากนั้นได้มีผู้พัฒนาต่อในเชิงการประมวลผลแบบ real-time [7]

ฮาดูปได้ทำการสร้างภายใต้สมมติฐานที่ว่า คลัสเตอร์ที่ใช้ในการประมวลผลนั้นเป็นแบบเอกพันธุ์ (Homogeneous Cluster) กล่าวคือ ทุกๆเครื่องที่ประมวลผลอยู่ในคลัสเตอร์เดียวกัน ต้องเป็นเครื่องที่มีประสิทธิภาพในการประมวลผลเท่ากันหรือใกล้เคียงกัน โดยการคำนวณความสามารถในการประมวลผลนั้นโดยจะนำไปคิดกับ คะแนนในการประมวลผล (Progress Score) ของทั้งคลัสเตอร์ฮาดูปแล้วนำมาเฉลี่ย ทำให้ได้ผลลัพธ์เป็นลักษณะของอัตราส่วนของ คลัสเตอร์ฮาดูปนั้นๆ โดย ส่วนแจกงานในคลัสเตอร์ฮาดูป (Hadoop Scheduler) จะใช้ลักษณะการแจกงานภายใต้เงื่อนไขที่ว่าเครื่องคอมพิวเตอร์เครื่องไหนนั้นประมวลผลได้ต่ำกว่าอัตราที่ทำการเฉลี่ยได้เครื่องนั้นแสดงว่ากำลังมีภาระในการทำงานหนัก จะไม่แจกจ่ายงานไปที่เครื่องคอมพิวเตอร์เครื่องนั้นจนกระทั่ง ค่าของการประมวลผลนั้นจะดีขึ้นหรือพ้นจากอัตราค่าที่กำหนดขึ้น (Threshold) แต่ทว่าในปัจจุบันการเจริญเติบโตทางธุรกิจถือว่าเจริญเติบโตขึ้นอย่างรวดเร็ว ไม่ว่าจะเป็นกูเกิลไดรฟ์ ซึ่งมีการเจริญเติบโตเพิ่มขึ้น 3 เท่า ในเวลา 1 ปีเท่านั้น ซึ่งหากเราคำนึงถึงในมุมมองของการขยายประสิทธิภาพของระบบ (Scalability) ระบบไม่สามารถจัดหาเครื่องที่มีการประมวลผลที่มี

ประสิทธิภาพใกล้เคียงกันได้ ดังนั้นระบบที่ใช้หลักการของคลัสเตอร์ประมวลผลส่วนใหญ่จะเป็นคลาสเตอร์แบบแยกกลุ่ม (Heterogeneous Cluster)

วิธีการแจกงานในคลัสเตอร์ฮาดูปนั้นที่กล่าวไปข้างต้นทำให้เกิดปัญหาในระบบที่เป็นคลัสเตอร์แบบต่างชนิด (Heterogeneous Cluster) เนื่องจากการที่ส่วนแจกงานในคลัสเตอร์ฮาดูปได้ทำการพิจารณาภาพโดยรวมโดยคำนวณจากอัตราส่วนของคะแนนประมวลผลโดยการคำนวณใช้หลักการคิดรวมทั้งคลัสเตอร์ หากเครื่องคอมพิวเตอร์มีความสามารถในการประมวลผลไม่เท่ากันแล้วจะทำให้เกิดปัญหาในการแจกจ่ายงาน งานวิจัยจึงได้นำเสนอแนวคิดต่างๆ ดังจะกล่าวถึงต่อไปนี้

- การแบ่งงานส่วนแจกงานสามารถทำในลักษณะของการแจกงานด้วยตนเอง (Self-Scheduling) รวมไปถึงคลัสเตอร์แบบต่างชนิด (Heterogeneous Cluster) ประสิทธิภาพในการประมวลผลย่อมไม่เท่ากันดังนั้นการจัดสรรข้อมูลไปให้เครื่องในคลัสเตอร์นำไปประมวลผลนั้นควรที่จะไม่เท่ากันด้วยเหมือนกัน
- ส่วนแจกงานในคลัสเตอร์ฮาดูปจะสามารถทราบได้อย่างไรว่าเครื่องที่ประมวลผลในคลัสเตอร์เครื่องใดมีประสิทธิภาพมากที่สุดหรือมีประสิทธิภาพน้อยที่สุดในลักษณะของงานที่กำลังประมวลผลอยู่

1.2 วัตถุประสงค์ของการวิจัย

วัตถุประสงค์ของงานวิจัยสามารถแบ่งได้เป็นหัวข้อต่างๆดังต่อไปนี้

1. เพื่อเพิ่มประสิทธิภาพการทำงานของส่วนแจกงานในคลัสเตอร์ฮาดูป (Hadoop Scheduler) ในคลัสเตอร์แบบต่างชนิด
2. เพื่อสร้างอัลกอริทึมการแจกงานในลักษณะของคลัสเตอร์ฮาดูป
3. งานวิจัยจะมีผลพัฒนาการแจกงานในลักษณะการประมวลผลแบบขนาน (Parallel Computing)

1.3 ขอบเขตของการวิจัย

งานวิจัยที่นำเสนอมีขอบเขตของการวิจัยดังต่อไปนี้

1. ไม่สนับสนุนลักษณะงานที่มีความสัมพันธ์ระหว่างกัน (Dependency Job)
2. ไม่สนับสนุนสถานการณ์ความล้มเหลวเนื่องจากฮาร์ดแวร์

3. การกำหนดค่าในคลัสเตอร์ เช่น ลักษณะของงาน หรือ ค่าการทำซ้ำของข้อมูล ฯลฯ ต้องสอดคล้องกับความเป็นจริง จึงได้ผลออกมาถูกต้องและมีประสิทธิภาพ
4. สถาปัตยกรรม ที่ใช้ในการประเมินเป็นเป็นคลัสเตอร์แบบต่างชนิด (Heterogeneous Cluster) และคลัสเตอร์แบบจำลอง (Virtual Machine)
5. สิ่งที่นำมาซึ่งเป็บการทำงานของฮาดูปเดิมนั้นถือว่าทำงานถูกต้องและมีประสิทธิภาพ

1.4 ขั้นตอนและวิธีดำเนินการวิจัย

งานวิจัยที่นำเสนอมีขั้นตอนและวิธีการดำเนินการวิจัยดังต่อไปนี้

1. ศึกษาและทำความเข้าใจการทำงานของฮาดูปและงานวิจัยที่เกี่ยวข้อง
2. ศึกษาวิธีการแก้โอเพนซอร์ซฮาดูปและอ่านการทำงานของอัลกอริทึมเดิม
3. ออกแบบวิธีการพัฒนาอัลกอริทึม
4. พัฒนาระบบต้นแบบ เพื่อสนับสนุนแนวคิดและขั้นตอนวิธีที่ได้นำเสนอ
5. กำหนดกรณีทดสอบและกรณีศึกษาเพื่อใช้ทดสอบระบบต้นแบบ
6. ทดสอบระบบต้นแบบ
7. วิเคราะห์ผลการทดลอง สรุปผล และข้อเสนอแนะ
8. จัดทำวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. เพิ่มประสิทธิภาพการทำงานให้กับส่วนแจกงานในคลัสเตอร์ฮาดูป
2. สร้างและสนับสนุนโดยใช้หลักการการเรียนรู้การแจกงานด้วยตัวเอง (Self-Scheduling) โดยนำหลักการของการควบคุมการแออัดของข้อมูลมาใช้ในการแก้ปัญหา

1.6 ลำดับการจัดเรียงเนื้อหาในวิทยานิพนธ์

วิทยานิพนธ์ได้ออกแบบเนื้อหาแบ่งเป็น 6 บทดังต่อไปนี้ บทที่ 1 เป็นบทนำซึ่งกล่าวถึงความเป็นมาและความสำคัญของปัญหา วัตถุประสงค์ของการวิจัย รวมถึงประโยชน์ที่คาดว่าจะได้รับ บทที่ 2 กล่าวถึง ทฤษฎีและงานวิจัยที่เกี่ยวข้อง บทที่ 3 กล่าวถึง วิธีการออกแบบและขั้นตอนการดำเนินงาน บทที่ 4 กล่าวถึง ขั้นตอนการพัฒนาระบบในเชิงปฏิบัติ บทที่ 5 กล่าวถึง การทดสอบและผลการทดลองและบทที่ 6 สรุปผลงานวิจัยและแนวทางการพัฒนาต่อ

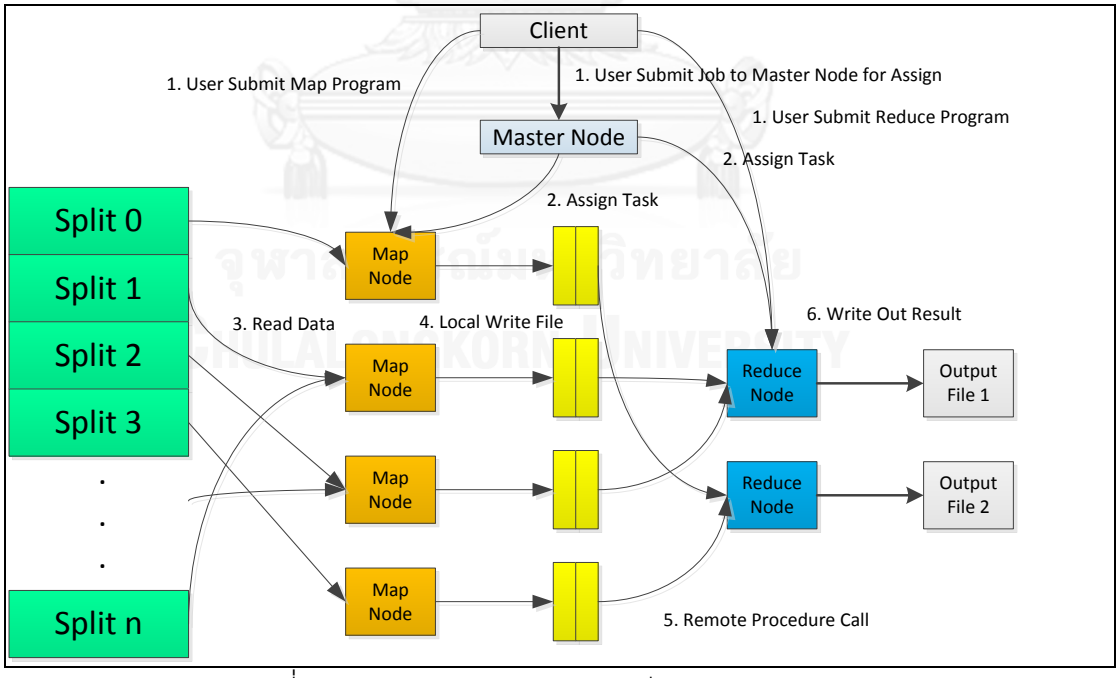
บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

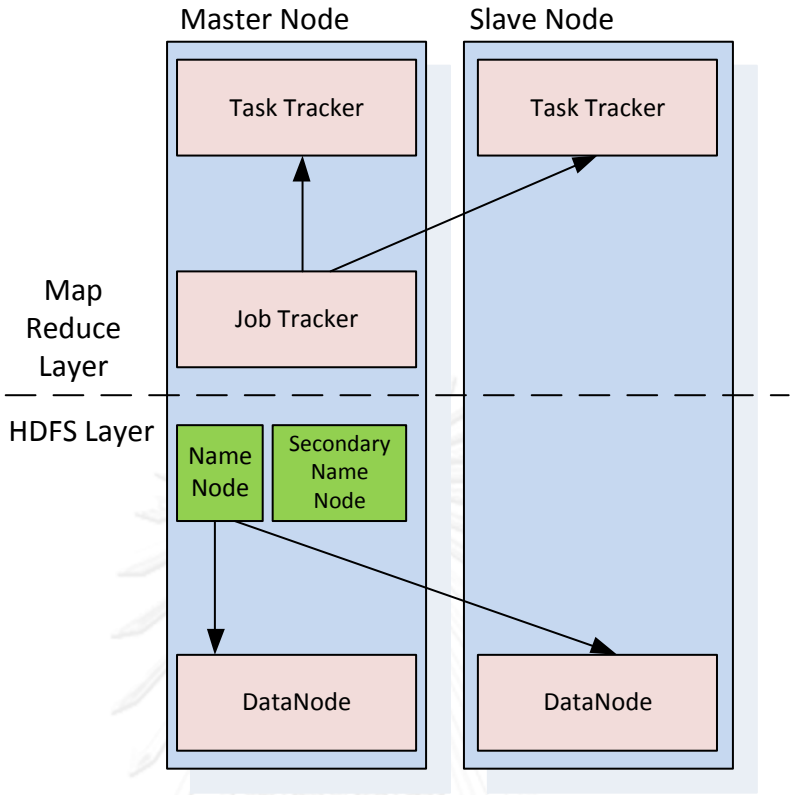
งานวิจัยนี้ ผู้วิจัยได้ค้นคว้า ศึกษาเอกสาร แหล่งความรู้ทางอินเตอร์เนต งานวิจัย รวมทั้งแนวคิดทฤษฎีต่างๆ ที่เกี่ยวข้อง ได้ดังนี้

2.1.1 ฮาดูปแมพรีดิว (Hadoop MapReduce)

แมพรีดิว (MapReduce) เป็นลักษณะการเขียนโปรแกรมชนิดหนึ่ง โดยสร้างขึ้นเพื่อประมวลผลข้อมูลที่มีขนาดใหญ่โดยใช้หลักการโดยการแบ่งปัญหาให้เป็นปัญหาย่อยๆ (Divide and Conquer Technic) แล้วทำการประมวลผลในรูปแบบเดียวกัน (Map Phase) แล้วทำการรวมผลลัพธ์ต่างๆให้อยู่ในลักษณะของผลลัพธ์ที่ต้องการ (Reduce Phase) โดยใช้หลักการการจับคู่ (Key/Value Pair) โดยส่วนใหญ่จะนำไปใช้ในงานประเภทต่างๆ เช่น การจับหาลักษณะของคำ (Distributed Grep), การนับจำนวนคำ (Word Count), การเรียงข้อมูล (Sorting) etc. โดยหลักการการประมวลผลหรือการแจกงานนั้นแสดงในรูปที่ 2.1 [8]



รูปที่ 2-1 แสดงการทำงานของแมพรีดิว (MapReduce)



รูปที่ 2-2 แสดงสถาปัตยกรรมของแมพรีดิว (MapReduce)

จากรูปที่ 2-2 ฮาดูปจะทำสร้างจาวาโปรเซส (Java Process) ในแต่ละเครื่องโดยเครื่องหลัก (Master Node) จะประกอบด้วย 3 โปรเซสดังต่อไปนี้

1. NameNode ทำหน้าที่เก็บข้อมูลที่อยู่ของไฟล์ HDFS ในคลัสเตอร์
2. Secondary NameNode เป็นโปรเซสสำรองเมื่อ NameNode เกิดการทำงานที่ไม่ปกติ
3. JobTracker มีหน้าที่ทำการแจกงานให้กับเครื่องที่อยู่ในคลัสเตอร์และคอยติดตามสถานะของเครื่องในคลัสเตอร์

เครื่องอื่นๆที่ไม่ใช่เครื่องหลักในคลัสเตอร์จะประกอบด้วย 2 โปรเซสดังต่อไปนี้

1. DataNode ทำหน้าที่หาที่อยู่ของข้อมูลให้แก่เครื่องที่ทำการประมวลผลว่าข้อมูลอยู่ที่เครื่องของตนเองหรือว่าจะต้องทำการเรียกข้อมูลจากระยะไกลเพื่อประมวลผล
2. TaskTracker ทำหน้าที่ส่งข้อมูลรายงานสถานะของเครื่องที่กำลังประมวลผล ในส่วนของ อัตราในการประมวลผล (Progress Rate) และ สถานะของทรัพยากร

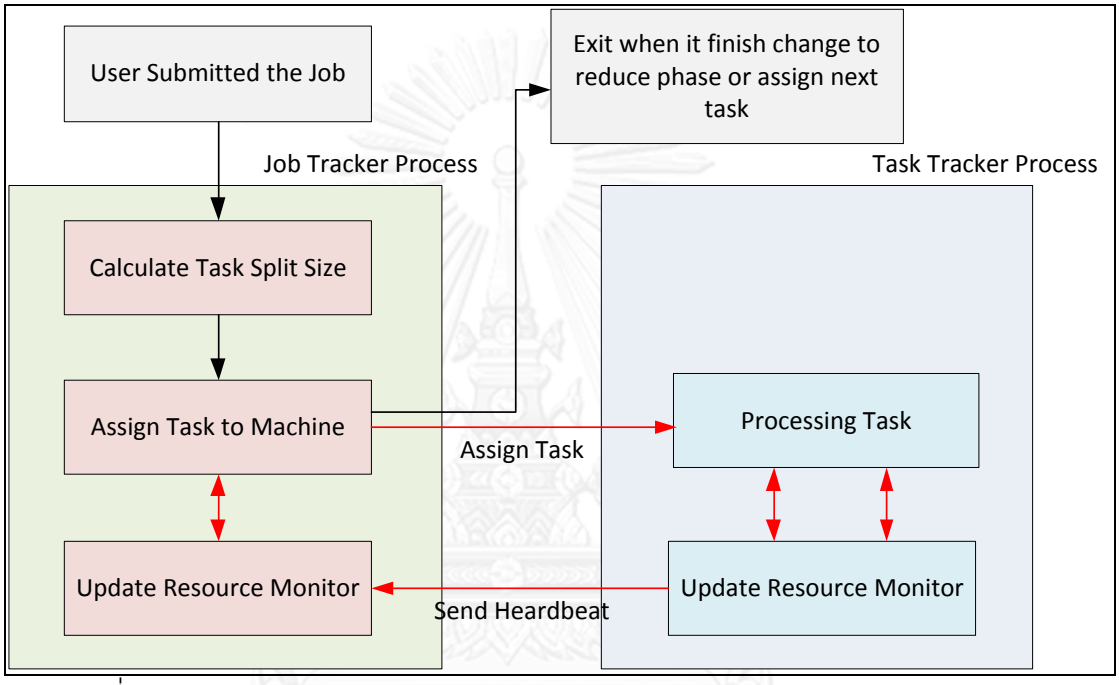
โดยการอธิบายขั้นตอนการทำงานของแมพรีดิวนั้นจะใช้ รูปที่ 2-1 และ รูปที่ 2-2 ในการอธิบายดังขั้นตอนดังต่อไปนี้

1. ผู้ใช้ร้องขอการประมวลโดยส่งคำสั่งการทำงานมาให้กับเครื่องหลัก (Master Node) จากรูปที่ 2-2 คำสั่งจะทำการร้องขอไปในส่วนของ JobTracker รวมไปถึงข้อมูลที่ใช้ โดยเครื่องหลักจะมีการเก็บคีย์ของข้อมูลเอาไว้เพื่อทำการอ้างอิงไปถึงข้อมูลจากรูปที่ 2-2 การเก็บคีย์ของข้อมูลจะเป็นหน้าที่ของ NameNode
2. เครื่องหลักทำการแจกงานไปยังเครื่องคอมพิวเตอร์อื่นในคลัสเตอร์ซึ่งจะใช้หลักการการแบ่งไฟล์เพื่อนำไปทำงานโดยขนาดของการแบ่งจะทำการแบ่งตามขนาดที่ได้ตั้งค่าไว้โดยผู้ดูแลระบบในขนาดที่เท่ากัน โดยหน้าที่ของการแจกงานนั้นจะเป็นหน้าที่ของ JobTracker ซึ่งอยู่ในเครื่องหลัก การแจกงานจะเป็นในลักษณะของการร้องขอจากเครื่องภายในคลัสเตอร์ที่มีสถานะว่าง (Ideal Mode) ในขณะนั้น โดย JobTracker จะทำการตรวจสอบสถานะของเครื่องในคลัสเตอร์ผ่านส่วนของ TaskTracker ที่อยู่ในเครื่องทุกเครื่องในคลัสเตอร์ เพื่อทราบถึงสถานะของเครื่องในขณะนั้น
3. เมื่อเครื่องในคลัสเตอร์ได้งานจากเครื่องหลัก เป็นหน้าที่ของ DataNode ในรูปที่ 2-2 จะมีการร้องขอที่อยู่ของข้อมูลที่จะทำการประมวลผลไปที่ NameNode เพื่อให้ทราบว่าข้อมูลนั้นอยู่ที่เครื่องของตัวเอง (Data Locality) [9] หรือว่าจะต้องทำการเรียกข้อมูลจากระยะไกล (Remote Copy) เมื่อได้ข้อมูลแล้วจะทำการประมวลผลในแมพเฟส (Map Phase)
4. เมื่อการประมวลผลงานในแมพเฟส (Map Phase) นั้นเสร็จสิ้น จะทำการเขียนข้อมูลกลับไปในไฟล์ชั่วคราว (Temporary File) เพื่อรอให้แมพเฟส (Map Phase) เสร็จสิ้นทั้งหมด โดยเมื่อเขียนไฟล์ชั่วคราวเสร็จ TaskTracker จะทำการร้องของานไปที่ JobTracker อีกครั้ง
5. หลังจากที่งานที่ประมวลได้ทำการประมวลผลในแมพเฟสเสร็จสิ้นทั้งหมดแล้วจะทำการจัดกลุ่มของคีย์ที่ได้เป็นผลลัพธ์จากแมพเฟส เพื่อนำไปใช้ต่อในรีดิวเฟส (Reduce Phase)
6. รีดิวเฟสทำการรวมข้อมูลโดยจัดกลุ่มคีย์ที่ซ้ำกันที่ได้จากผลลัพธ์ในแมพเฟส รวมกันเป็นคีย์เดียว ด้วยอัลกอริทึมการจัดเรียงข้อมูล (Sorting) โดยเมื่อเสร็จขั้นตอนนี้คีย์จะไม่ซ้ำกัน ทำการเขียนข้อมูลลง HDFS File เป็นการเสร็จสิ้นกระบวนการแมพรีดิว

7. JobTracker จะทำการแจ้งกลับไปยัง User ว่าทำงานเสร็จสิ้น

2.1.2 การแจกงานของฮาดูปในสภาวะปกติ (Hadoop Default Scheduler)

จากหัวข้อที่ 2.1.1 การแจกงานภายในคลัสเตอร์จะเป็นหน้าที่ของ JobTracker โดยการแจกงานนั้นจะเป็นไปตามขั้นตอนด้านล่างดังรูปที่ 2-3

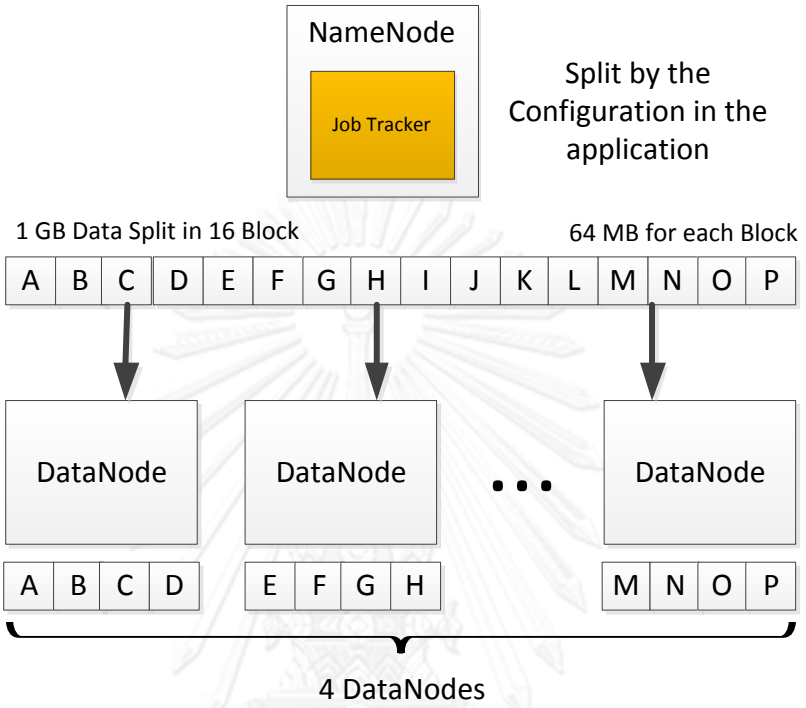


รูปที่ 2-3 แสดงการแจกงานและการติดต่อกันระหว่าง JobTracker และ TaskTracker

จากรูปที่ 2-3 การแจกงานระหว่างเครื่องหลักและเครื่องอื่นๆในคลัสเตอร์นั้นจะเป็นการติดต่อกันระหว่าง JobTracker และ TaskTracker เพื่อของานโดยขั้นตอนการทำงานมีดังต่อไปนี้

1. ยูสเซอร์ทำคำสั่งเริ่มทำงานในฮาดูปคลัสเตอร์ จะทำการส่งคำสั่งไปที่ JobTracker เพื่อทำการแจกงาน
2. ส่วนแจกงานในคลัสเตอร์ฮาดูปจะทำการประมวลผลขนาดของงานที่จะส่งไปประมวลผลที่เครื่องในคลัสเตอร์โดยขึ้นกับพารามิเตอร์ที่กำหนดโดยผู้ดูแลระบบที่ชื่อว่า mapred.max.split.size โดยขนาดของการแบ่งงานจะเป็นไปตามรูปที่ 2-4
3. เครื่องที่ถูกให้งานจะทำการประมวลผล โดยระหว่างการประมวลผลนั้นจะมีการส่ง Acknowledgement กลับไปที่ JobTracker โดยในทางปฏิบัติฮาดูปเรียกว่า

Heartbeat เพื่อรายงานสถานะของตัวเองเป็นระยะๆ โดยปกติจะใช้เวลาในการรายงานทุก 3 วินาที



รูปที่ 2-4 แสดงการแบ่งงานของส่วนแจกงานในคลัสเตอร์ฮาดูปในสภาวะปกติ

4. JobTracker จะรับข้อมูลที่รายงานจาก TaskTracker ทำการคำนวณหาค่าที่ระบุประสิทธิภาพของการประมวลผลเรียกว่า อัตราส่วนของคะแนนประมวลผล (Progress Score) โดยเราจะใช้ตัวย่อว่า PS โดยค่านั้นจะอยู่ระหว่าง 0 ถึง 1

2.1.3 **อัตราส่วนของคะแนนประมวลผล** (Progress Score)

การคำนวณอัตราส่วนของคะแนนประมวลผล (Progress Score :PS) ในฮาดูปสามารถแบ่งได้เป็น 2 ส่วน โดยการคำนวณในแมพเฟสและรีดิวเฟสนั้นจะมีความแตกต่างกันกล่าวคือ

1. แมพเฟสอัตราส่วนของคะแนนประมวลผล (Map Phase Progress Score) สามารถคำนวณได้จากค่าของการอ่านข้อมูลที่เข้าไปประมวลผล (Input Data Read)
2. รีดิวเฟสอัตราส่วนของคะแนนประมวลผล (Reduce Phase Progress Score) สามารถคำนวณได้จาก 3 ส่วนที่ถือว่าเป็นขั้นตอนย่อย (Sub-process) ได้แก่

a. Copy Phase จะเกิดขึ้นเมื่อแมพเฟสนั้นเสร็จสิ้นจะมีการก๊อปปี้ไปยังเครื่องที่จะทำการรีดิว
b. Sort Phase จะทำการเรียงข้อมูลด้วยคีย์ที่ได้จากผลลัพธ์ในแมพเฟส
c. Reduce Phase เมื่อทำการประมวลผลรีดิวโปรแกรม (Reduce Task)

โดยที่กล่าวมาข้างต้นสามารถเขียนเป็นสมการได้ดังสมการที่ 2-1 โดยกำหนดให้ความหมายของพารามิเตอร์ในสมการเป็นดังต่อไปนี้

M จำนวนของข้อมูลทั้งหมดที่ได้รับการประมวลผลเรียบร้อยแล้ว
N จำนวนของข้อมูลทั้งหมดที่ต้องประมวลผล
MT เวลาที่ใช้ในแมพเฟส (Map Phase)
K สถานะของการทำงานของแมพรีดิว โดยสถานะที่ 1 คือ Copy สถานะที่ 2 คือ Sort และ สถานะที่ 3 คือ Reduce
RT เวลาที่ใช้ในรีดิวเฟส (Reduce Phase)

$$PS = \begin{cases} \dfrac{M}{N} & For\ MT \\ \dfrac{1}{3} x(K + \dfrac{M}{N}) & For\ RT \end{cases} \quad (2\text{-}1)$$

เมื่อนำค่าทั้งคลัสเตอร์มาเฉลี่ยจะถูกเรียกว่า Average Progress Score (APS) เมื่อ T คือจำนวนงานทั้งหมดที่ต้องการประมวลผล โดยสามารถคำนวณค่าของ APS ได้จากสมการที่ 2-2

$$APS = \frac{\sum_{i=1}^{T} PS_i}{T} \quad (2\text{-}2)$$

2.1.4 การประมวลงานสำรอง (Speculative Task)

จากหัวข้อที่ 2.1.3 ได้กล่าวถึงอัตราส่วนของคะแนนประมวลผล โดยในทางปฏิบัติถ้าอัตราส่วนของคะแนนประมวลผลนั้นมีค่าน้อยกว่าค่าๆหนึ่งแล้ว ฮาดูปจะมีการประมวลงานสำรองหรือที่เราเรียกว่า Speculative Task หรือ Backup Task โดยการจะประมวลงานสำรองหรือไม่นั้นขึ้นอยู่กับสมการที่ 2-3

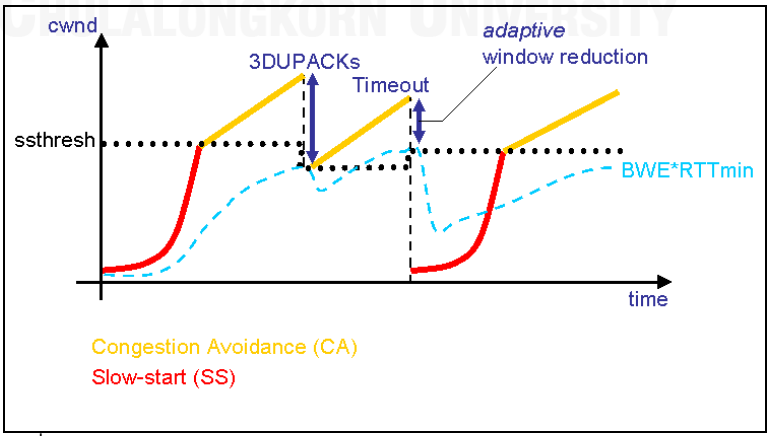
$$PS_{1..T} < APS - 0.2 \quad (2\text{-}3)$$

PS พารามิเตอร์นั้นสามารถทำงานได้ดีถ้าหากอยู่บนคลัสเตอร์แบบเอกพันธ์ แต่ถ้าหากเป็นคลัสเตอร์แบบต่างชนิดแล้ว จะทำให้เกิดปัญหาทางผู้เขียนจะขออธิบายนส่วนของงานวิจัยที่เกี่ยวข้องต่อไป

2.1.5 การควบคุมการแออัดของข้อมูล (Congestion Control)

โดยรูปที่ 2-5 แสดงการรูปแบบของกราฟ การควบคุมการแออัดของข้อมูลนั้นสามารถแบ่งออกได้เป็น 3 ส่วน ได้แก่

1. Slow Start Phase โดยในเฟสนี่คือเฟสเริ่มต้นของการส่งข้อมูลโดยเริ่มต้นจะเริ่มส่ง 1 Package Size แล้วทำการเพิ่มขึ้นเรื่อยๆในลักษณะของ Exponential จนกระทั่ง Response Time ถึงค่า Threshold ที่กำหนดไว้จะทำการเปลี่ยนเฟส เพราะ เป็นการเริ่มต้นเกิดการแออัดของข้อมูลเมื่อมี Response Time เพิ่มมากขึ้น
2. Congestion Avoidance Phase เมื่อค่า Response Time ถึงค่าที่กำหนดไว้แล้วจะทำการเพิ่ม Package Size ในลักษณะของ Adaptive one increase โดยเพิ่ม Package Size ทีละ 1 Package จนกระทั่งเกิด Timeout ขึ้นในระบบ
3. ขั้นตอนนี้จะเป็นการปรับค่าของ Package Size ที่ใช้ส่งลงมาสู่จุดเริ่มต้นจะเริ่มส่ง 1 Package Size อีกครั้งหลังจากนั้นจะทำ ขั้นตอนที่ 1 จนกว่า Threshold จะเท่ากับครึ่งหนึ่งของ Threshold เดิม ทำเช่นนี้เราจะได้ค่า Package Size ที่เหมาะสมในการส่ง



รูปที่ 2-5 แสดงหลักการการควบคุมการแออัด (Congestion Control)

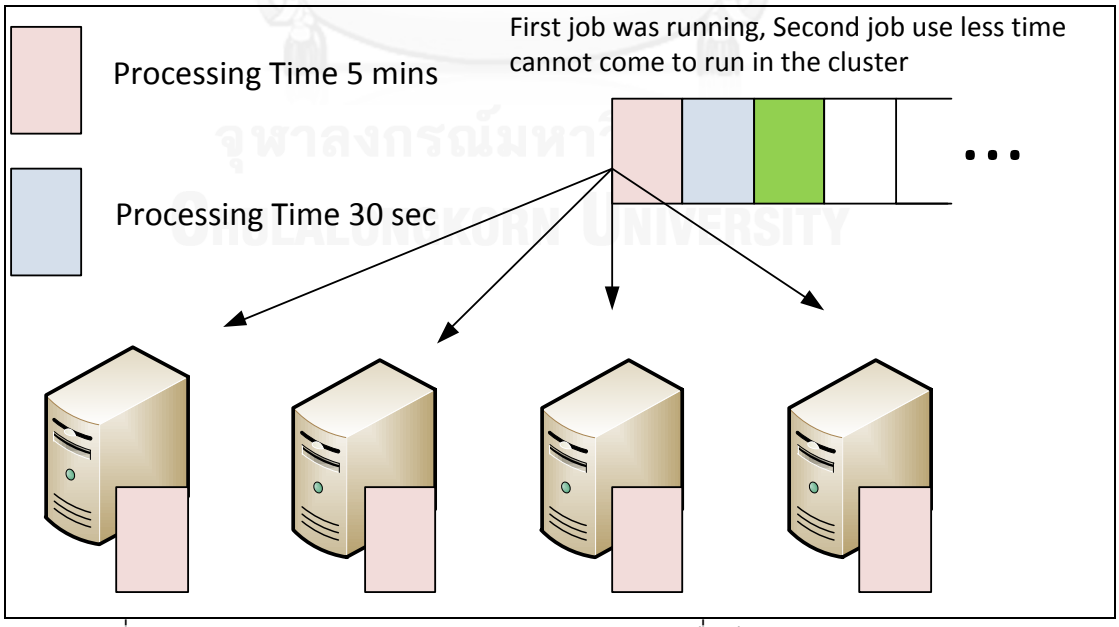
2.2 งานวิจัยที่เกี่ยวข้อง

ในส่วนของงานวิจัยที่เกี่ยวข้องได้มีผู้นำเสนอผลงานไว้เพื่อแก้ปัญหาที่เกิดขึ้นในการแจกงานในสภาวะปกติ (Hadoop Default Scheduler) โดยจะกล่าวถึงเป็นงานวิจัยเรียงตามลำดับตามความทันสมัยของงาน ดังจะกล่างต่อไปนี้

2.2.1 การแจกงานของฮาดูปในสภาวะความเท่าเทียมกัน (Hadoop Fair Scheduler)

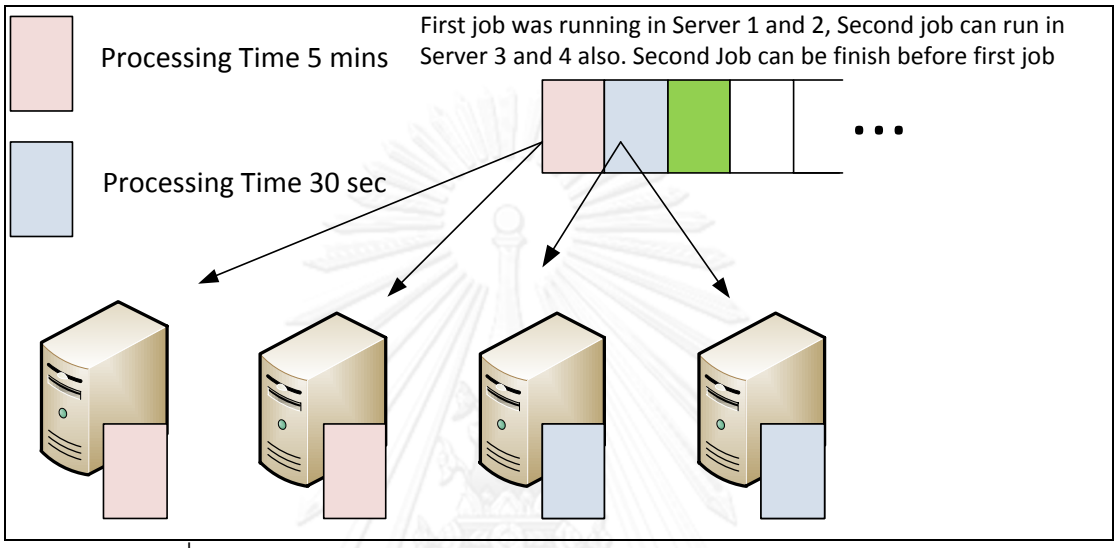
งานวิจัยนี้มีจุดประสงค์เพื่อแก้ปัญหาที่เกิดในการแจกงานในสภาวะปกติ (Hadoop Default Scheduler) โดยผู้นำเสนอได้เห็นถึงปัญหาดังต่อไปนี้ [10]

- เนื่องจากว่าการแจกงานของฮาดูปในแบบปกตินั้นจะเป็นลักษณะ FIFO คือ งานที่มาถึงก่อนจะได้ทำงานก่อนซึ่งจะทำให้เกิดปัญหาเมื่อมีงานที่ใช้เวลานานเข้ามาอยู่ในระบบ (Long Processing Job) หากมีงานในลักษณะที่เล็กกว่าและการประมวลผลเสร็จเร็วกว่าเข้ามาในระบบจะทำให้งานดังกล่าวนั้นไม่ได้ทำการประมวลผเนื่องจากต้องทำการรอคอยการเสร็จสิ้นของงานที่กำลังรันอยู่ ซึ่งทำให้กระทบต่อประสิทธิภาพของระบบโดยรวม ดังแสดงรูปที่ 2-6
- การประมวลผลในลักษณะของความเท่าเทียมกันจึงมีความจำเป็นเพื่อให้งานที่ใช้เวลาในการประมวลผลที่น้อยกว่าได้มีโอกาสประมวลผลในระบบ



รูปที่ 2-6 แสดงปัญหาของการแจกงานในสภาวะปกติเมื่อมีความหลากหลายของงาน

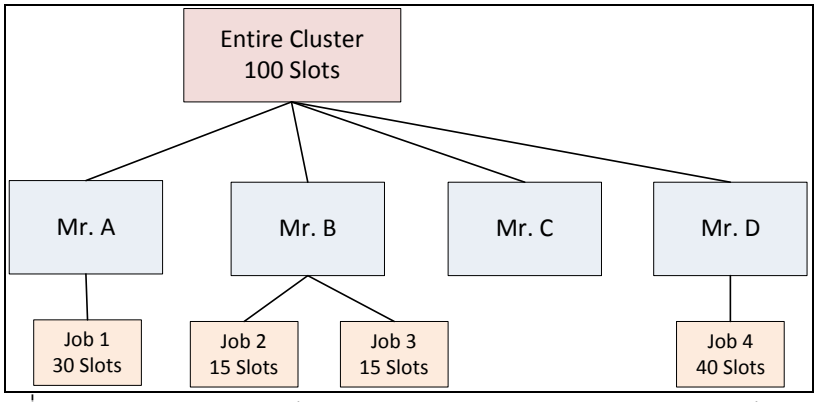
โดยผู้วิจัยได้นำเสนอการแก้ปัญหาโดยใช้ลักษณะของการจำกัดทรัพยากรในคลัสเตอร์ (Resource Pool Configuration) เพื่อให้งานที่มาที่หลังได้รับการประมวลผลไม่จำเป็นต้องมีการรอประมวลผลภายในคิว (Queue) โดยแสดงการทำงานในรูปที่ 2-7



รูปที่ 2-7 แสดงการแก้ปัญหาของการแจกงานในสภาวะความเท่าเทียมกัน

ผู้วิจัยได้นำไปปฏิบัติโดยงานแต่ละงานในแต่ลักษณะสามารถกำหนดลักษณะของทรัพยากรได้ เพื่อทำการจองไว้เมื่องานเข้ามาสู่ระบบ โดยการกำหนดจะใช้ตัวแปร 3 ตัวได้แก่ โดยจะมีลักษณะของการสร้างเป็นแผนภูมิต้นไม้ได้ดังรูป 2-8

1. จำนวนต่ำสุดของ Map Node ที่จะเกิดขึ้นได้
2. จำนวนต่ำสุดของ Reduce Node ที่จะเกิดขึ้นได้
3. จำนวนจำกัดในการประมวลผลในงานต่างๆ



รูปที่ 2-8 แสดงแผนภาพต้นไม้ของการแจกงานในสภาวะความเท่าเทียมกัน

2.2.2 LATE Scheduling Algorithm and SAMR Scheduling Algorithm

งานวิจัยนี้มีจุดประสงค์เพื่อแก้ปัญหาที่เกิดในการแจกงานในสภาวะปกติ (Hadoop Default Scheduler) ที่เกิดขึ้นเมื่อคลัสเตอร์เป็นลักษณะของคลัสเตอร์แบบต่างชนิด (Heterogeneous Cluster) [11] โดยผู้นำเสนอได้เห็นถึงปัญหาดังต่อไปนี้

- จากหัวข้อที่ 2.1.4 ว่าด้วยเรื่องของ การประมวลผลงานสำรอง (Speculative Task) ฮาดูปจะมีการทำงานสำรองเมื่อสมการที่ 2-3 เป็นจริง แต่ผู้วิจัยที่นำเสนอเกี่ยวกับ LATE Scheduling Algorithm เห็นว่าจะเกิดปัญหาเมื่ออยู่ในภาวะของคลัสเตอร์แบบต่างชนิด ดังตัวอย่างที่ได้กล่าวไว้ในงานวิจัย เมื่อเครื่อง A มีอัตราส่วนของคะแนนประมวลผลเป็น 0.7 เหลืออีก 100 s. จะทำงานเสร็จกับเครื่อง B ซึ่งคะแนนในการประมวลผลอยู่ที่ 0.5 เหลืออีก 50 s. สถานการณ์จะเกิดขึ้นก็ต่อเมื่องานนั้นเริ่มในเวลาที่แตกต่างกันซึ่งการประมวลผลงานสำรองควรที่จะเกิดกับเครื่อง A ซึ่งมีอัตราส่วนของคะแนนประมวลผลมากกว่าทำให้บางครั้งการประมวลผลงานสำรองผิดพลาดของการแจกงานในสภาวะปกติ
- ผู้วิจัยได้นำเสนอแนวทางการแก้ปัญหาโดยใช้หลักการของการหาเวลาที่จะเสร็จสิ้นมาใช้ในการประมวลผลงานสำรองให้เกิดประสิทธิภาพมากยิ่งขึ้น

ผู้วิจัยได้คำนวณค่าเวลาที่ใช้ในการประมวลผลในแต่ละงาน (Task: T) และขนาดของงานที่ประมวลผลไปแล้ว (TR) โดย LATE Scheduling Algorithm จะคำนวณหาค่าของอัตราการประมวลผลงาน (PR) ได้จาก สมการที่ 2-4

$$PR_{1...n} = \frac{PS_{1...n}}{TR_{1...n}} \tag{2-4}$$

เมื่อได้ค่าของอัตราการประมวลผลงาน (PR) เพื่อหาค่าเวลาที่เหลือในการประมวลผลงาน (Task :T) Time to End of parameter (TTE) ได้จากสมการ 2-5

$$TTE_{1...n} = \frac{1 - PS_{1...n}}{PR_{1...n}} \tag{2-5}$$

จากหัวข้อที่ 2.1.4 ฮาดูปการแจกงานในสภาวะปกติจะทำการประมวลผลงานสำรองเมื่อสมการที่ 2-3 เป็นจริงแต่ LATE Scheduling Algorithm จะทำการประมวลผลงานสำรองไปยังเครื่องที่มีค่า Time to End of parameter (TTE) มีค่ามากที่สุดในคลัสเตอร์ เมื่อ TaskTracker อยู่ในสถานะ Idle จะทำการประมวลผลงานสำรองก็ต่อเมื่อ

- จะไม่มีการส่งการประมวลผลงานสำรองไปยัง TaskTracker ที่มีค่าน้อยกว่า SlowNodeThershold
- จะไม่มีการส่งการประมวลผลงานสำรองให้กับงานที่มีค่า TTE มากกว่าค่าของ Speculative Cap

โดยแนวคิดของ LATE Scheduling Algorithm อยู่ภายใต้สมมติฐานที่ว่า เครื่องนั้นมีข้อมูลอยู่ที่เครื่องของตนเอง (Data Locality) ซึ่งประสิทธิภาพของเครือข่ายและประสิทธิภาพของ I/O ไม่ได้นำมาคิด โดยได้ทำการทดลองเปรียบเทียบกับการแจกงานในสภาวะปกติซึ่งมีประสิทธิภาพมากกว่าอยู่ 14%

ต่อมาได้มีผู้พัฒนาอีกกลุ่มหนึ่งได้นำเสนอแนวทางของ SAMR Scheduling Algorithm เป็นอัลกอริทึมต่อยอดจาก LATE Scheduling Algorithm โดยได้มีการนำเรื่องของการเก็บประวัติของข้อมูลในคลัสเตอร์ (History Cluster Information) เพื่อให้เกิดผลในการประมวลผลงานสำรองอย่างมีประสิทธิภาพมากยิ่งขึ้น โดยได้ทำการทดลองเปรียบเทียบกับการแจกงานในสภาวะปกติซึ่งมีประสิทธิภาพมากกว่าอยู่ 23% และ มีประสิทธิภาพมากกว่า 9% เมื่อเทียบกับ LATE Scheduling Algorithm

2.2.3 Hadoop Resource-aware Adaptive Scheduling

งานวิจัยของ Jorda Polo และคณะ เป็นงานวิจัยที่นำเสนออัลกอริทึมการแจกงานของฮาดูปเช่นเดียวกันแต่มีแนวคิดภายใต้พื้นฐานของเวลาที่คาดหวังว่างานนั้นจะสำเร็จ โดยใช้หลักการของ Placement Algorithm เข้ามาช่วยในการคิดโดยปัจจัยที่นำมาคำนวณนั้นได้แก่ เวลาในการประมวลผลสำเร็จในหนึ่งงาน (Job Completion Time), ประสิทธิภาพของเครือข่ายและประสิทธิภาพของ I/O โดยสนใจในลักษณะของงานที่ยูสเซอร์นั้นมีการร้องขอเข้ามาแล้วทำให้สำเร็จตามเวลาที่กำหนดไว้ อีกทั้งได้มีการ implement ลักษณะของ Priority Queue เพื่อเรียงลำดับความสำคัญของงาน งานวิจัยนี้ให้แนวคิดที่ว่าจะทำการคำนวณหาความสามารถใน

การเข้าถึงทรัพยากรเครือข่ายด้อย่างมีประสิทธิภาพได้อย่างไร โดยนำแนวคิดของงานวิจัยมาปรับปรุงให้เหมาะสม [12]

2.2.4 Hadoop Data Placement Scheduling

งานวิจัยของ Jiong Xie และคณะ ได้นำเสนอแนวทางการแบ่งงานของฮาดูปโดยใช้หลักการการคำนวณหาค่าของอัตราส่วนของการประมวล (Computing Ratio) ในทางเริ่มต้นซึ่งนำตัวอย่างของข้อมูลซึ่งเป็นตัวอย่างจำลองมาทำการ Submit เข้าไปในคลัสเตอร์เพื่อหาความสามารถในการประมวลผลในแต่ละเครื่องที่อยู่ในคลัสเตอร์ เพื่อนำมาคำนวณเป็น Computing Ratio ทำให้ทราบถึงความสามารถในการประมวลผลที่แท้จริง อีกทั้งได้นำเสนอวิธีการแจกงานในลักษณะของสถานะต่างๆ ในการประมวลผล โดยได้นำเสนอวิธีการแบ่งได้เป็น 3 สถานะ Initial Data placement Data Replacement และ Data Redistribution ซึ่งนำมาสู่แนวคิดของงานวิจัยในการแบ่งการประมวลผลและการคำนวณอัตราส่วนของการประมวล (Computing Ratio) [13]

2.2.5 Load Sharing Strategy and Congestion Control

งานวิจัยของ Natthakrit Sanguandikul นำเสนอการมอนิเตอร์สถานะของทรัพยากรในระบบซึ่งโดยใช้ทฤษฎีของการแชร์ข้อมูลในการประมวลผล (Load Sharing Strategy) โดยจะใช้หลักการของการ สร้าง workload ขนาดเล็กไปหาขนาดใหญ่โดยใช้จนกว่าจะถึงจุด perk ซึ่งจะทำให้ทราบถึง Performance สูงสุดของระบบนั้นว่าสามารถรองรับงานจริงๆได้มากน้อยเพียงใด ซึ่งนำมาประยุกต์ใช้กับประเภทของ Network ในชนิดต่างๆ เช่น WAN LAN MAN เป็นต้น โดยการใช้งาน Resource นั้นแปรผันตามลักษณะที่เกิดขึ้นจริง โดยงานวิจัยได้นำไปสู่แนวคิดการนำ Slow Start ไปใช้ในงานวิจัย [14]

บทที่ 3
วิธีการออกแบบและขั้นตอนการดำเนินงาน

งานวิจัยนี้ได้นำเสนอแนวทางการแก้ปัญหาดังที่กล่าวไปข้างต้นในส่วนของหัวข้อที่ 1.1 โดยบทนี้จะอธิบายถึงแนวทางการแก้ปัญหาในส่วนต่างที่กล่าวไปในข้างต้นดังต่อไปนี้

- การที่จะทำการแบ่งงานนั้นส่วนแจกงานสามารถทำในลักษณะของการแจกงานด้วยตนเอง (Self-Scheduling)
- เนื่องจากเป็นคลัสเตอร์แบบเยกกลุ่ม (Heterogeneous Cluster) ประสิทธิภาพในการประมวลผลย่อมไม่เท่ากันดังนั้นการจัดสรรข้อมูลไปให้เครื่องในคลัสเตอร์นำไปประมวลผลนั้นควรที่จะไม่เท่ากันด้วยเหมือนกัน

ในการแก้ปัญหาดังกล่าว ผู้วิจัยได้นำเสนอการแบ่งงานในลักษณะไม่คงที่โดยจะมีการคำนวณเพื่อทำการแบ่งงานที่เปลี่ยนแปลงไปตามความสามารถของระบบในระหว่างการประมวลผลซึ่งจะพิจารณาจากตัวแปรในส่วนของประสิทธิภาพของเครื่องข่ายและ I/O ภายนอก รวมไปถึงประสิทธิภาพในการประมวล เพื่อให้ส่วนแจกงาน (Scheduler) ทราบว่าควรที่จะแจกงานไปให้กับเครื่องในคลัสเตอร์เป็นขนาดเท่าไร รวมไปถึงการมีการใช้การแจกงานที่มองไปในอนาคต (Prediction Scheduler) เพื่อให้ได้ประสิทธิภาพสูงสุดในการแจกงานภายในคลัสเตอร์

ผู้วิจัยได้นำเสนออัลกอริทึม EFTF (Earliest Finish Time First) เพื่อแก้ปัญหาโดยใช้หลักการที่ได้กล่าวไปข้างต้น โดยผู้วิจัยจะขออธิบายส่วนของพารามิเตอร์ที่เกี่ยวข้องและคำศัพท์เฉพาะที่เกี่ยวข้อง เพื่อให้การทำความเข้าใจเป็นไปได้ง่ายยิ่งขึ้น หลังจากนั้นจะอธิบายการทำงานและการออกแบบร่วมไปถึงขั้นตอนการดำเนินงาน โดยรายละเอียดต่างๆเป็นดังต่อไปนี้

3.1 พารามิเตอร์ที่เกี่ยวข้องและคำศัพท์เฉพาะที่เกี่ยวข้อง

พารามิเตอร์ที่เกี่ยวข้องกับอัลกอริทึม EFTF (Earliest Finish Time First) แสดงในตารางที่ 3-1 และคำศัพท์เฉพาะต่างๆ แสดงในตารางที่ 3-2

ตารางที่ 3-1 แสดงพารามิเตอร์ที่เกี่ยวข้องกับอัลกอริทึม EFTF

พารามิเตอร์	ชื่อพารามิเตอร์	ความหมาย
NM	Number of machine	จำนวนเครื่องที่ประมวลผลภายในคลัสเตอร์ (หน่วย: เครื่อง)
N	Job Size	ขนาดของ Job ที่ Submit เข้ามาในระบบ (หน่วย:Bytes)
n	Number of Tasks	จำนวนของงานหลังจากแบ่งงานจากขนาดของ Job ที่ Submit เข้ามาในระบบ
np	Number of Tasks that are being processed	จำนวนของงานที่กำลังประมวลผล
$V_{1...n}$	Task Size	ขนาดของงานที่ประมวลผล ดังนั้นค่าของ V จะมีจำนวนเท่ากับ n จำนวน
$PerfR_{1...NM}$	Performance Score	คะแนนของการประมวลผลของเครื่องแต่ละเครื่องในคลาสเตอร์ โดยหากมีค่าเพิ่มขึ้นแสดงว่ามีประสิทธิภาพในการทำงานสูง ค่าของ Performance Score นั้นมีจำนวนเท่ากับ NM จำนวนกล่าวคือ มีจำนวนเท่ากับจำนวนเครื่องในคลัสเตอร์
$PR_{1...NM}$	Processing Rate	ความสามารถในการประมวลผลของเครื่องแต่ละเครื่องในคลาสเตอร์ โดยหากมีค่าเพิ่มขึ้นแสดงว่าความสามารถในการประมวลผลดี ค่าของ Processing Rate นั้นมีจำนวนเท่ากับ NM จำนวนกล่าวคือ มีจำนวนเท่ากับจำนวนเครื่องในคลัสเตอร์ (หน่วย: MB/s)
$IOR_{1...NM}$	I/O Utilization	ความสามารถในการอ่านและเขียนข้อมูลของเครื่องแต่ละเครื่องในคลาสเตอร์ โดยหากมีค่าเพิ่มขึ้นแสดงว่าการอ่านและเขียนข้อมูลในการประมวลผลดี ค่าของ I/O Uilization นั้นมีจำนวนเท่ากับ NM จำนวนกล่าวคือ มีจำนวนเท่ากับ

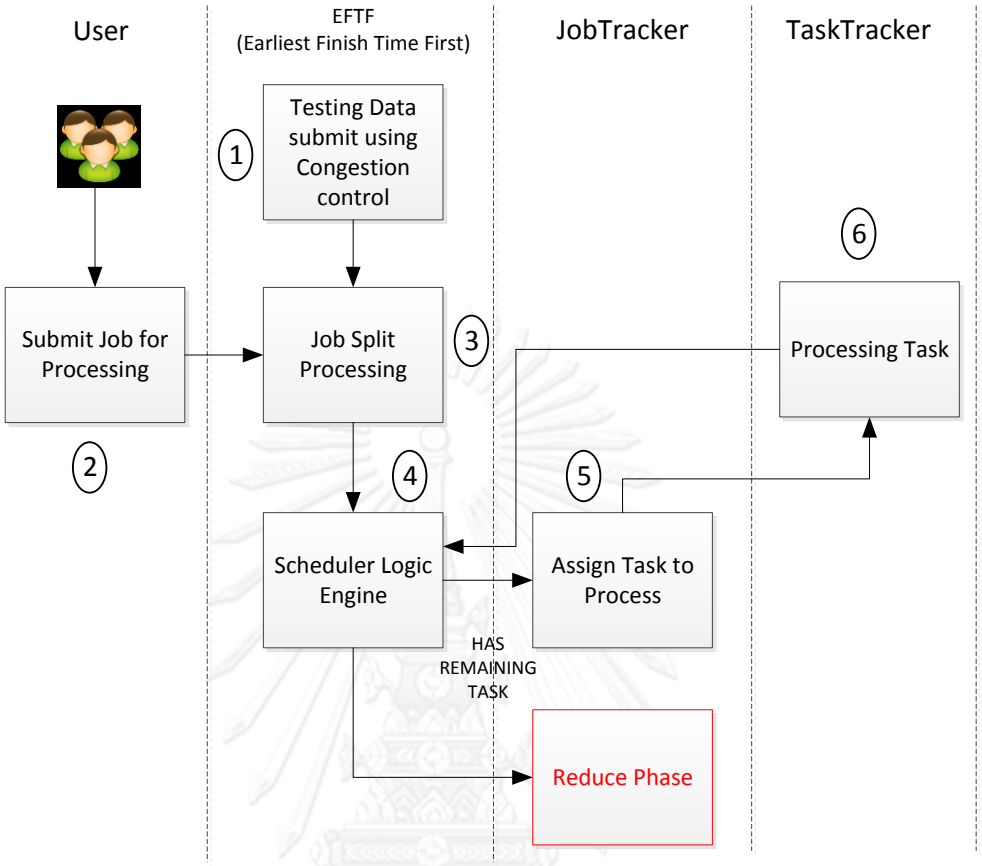
		จำนวนเครื่องในคลัสเตอร์ (หน่วย: MB/s)
$PRogR_{1...np}$	Progress Rate	เปอร์เซ็นต์ระหว่างงานที่ประมวลผลไปแล้วเทียบกับงานทั้งหมดค่าที่เป็นไปได้จะอยู่ระหว่าง 0 ถึง 1 ค่าของ Progress Rate จะมีเฉพาะงานที่กำลังประมวลผลอยู่กล่าวคือมีทั้งหมด np จำนวน
J	Sub-sprit Variable	จำนวนงานสูงสุดที่จะแจกให้ได้สำหรับ 1 เครื่อง (หน่วย: Job/เครื่อง)

ตารางที่ 3-2 แสดงคำศัพท์เฉพาะต่างๆที่เกี่ยวข้องกับอัลกอริทึม EFTF

คำศัพท์เฉพาะ	ความหมาย
เครื่องที่มีข้อมูลอยู่ (Data Locality) (m_L)	เครื่องภายในคลัสเตอร์ที่มีข้อมูลอยู่ภายในเครื่องพร้อมที่จะประมวลผล โดยเครื่องที่มีข้อมูลอยู่นั้นสามารถมีได้หลายเครื่อง $m_L = \{ m_{L_1}, m_{L_2}, m_{L_3}, ..., m_{L_n} \}$
เครื่องที่ร้องของาน (m_R)	เครื่องภายในคลาสเตอร์ประมวลผลเสร็จสิ้นหรือไม่มีงานที่จะต้องประมวลผลอีก มาร้องของานจากส่วนแจกงานภายในคลัสเตอร์
เครื่องที่คาดหมาย (m_E)	ระหว่างการแบ่งงานเพื่อแจกงานไปแต่ละเครื่อง อัลกอริทึม EFTF จะกำหนดเครื่องที่เหมาะสมที่จะได้งานนั้นในประมวลผล เพื่อเป็นตัวเลือกในการตัดสินใจของส่วนแจกงาน
Resource Information Estimator	ขั้นตอนการคำนวณค่าความสามารถในการประมวลผลและความสามารถในการอ่านเขียนข้อมูล โดยใช้หลักการการควบคุมการแออัด
Approximate Visual Test	การคำนวณหาค่า Confident Interval แล้วนำมาเปรียบเทียบในรูปแบบกราฟ Box Plots Diagram ผลที่ได้หากไม่มีการทับซ้อนจะทำให้เห็นความแตกต่างระหว่าง 2 สิ่งที่ทำการเปรียบเทียบ

3.2 ภาพรวมของการออกแบบของการแจกงานของอัลกอริทึม EFTF

โดยการอธิบายจะทำการอธิบายด้วยผังงานการทำงาน (Flow Chart) โดยจะอธิบายได้เป็นขั้นตอนในส่วนของผังงานการทำงานและจะอธิบายรายละเอียดเชิงลึกในหัวข้อถัดไปต่อจากผังงานการทำงาน โดยผังงานการทำงานแสดงในรูปที่ 3-1



รูปที่ 3-1 แสดงการทำงานภาพรวมของอัลกอริทึม EFTF

จากรูปที่ 3-1 สามารถอธิบายขั้นตอนการทำงานของงานวิจัยได้ดังต่อไปนี้

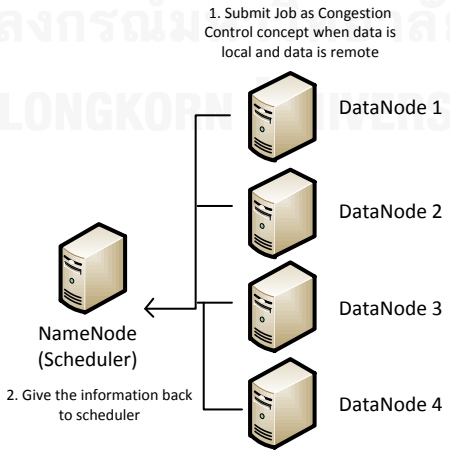
1. เมื่อคลัสเตอร์เริ่มต้นการทำงาน (Cluster Start-up) ระบบจะทำ Resource Information Estimation เพื่อประเมินค่าความสามารถในการประมวลผล (Processing Rate) และความสามารถในการอ่านเขียนข้อมูล (I/O Utilization) โดยนำข้อมูลทดสอบมาทำการทดลองประมวลผลโดยใช้หลักการการควบคุมการแออัดของข้อมูล ซึ่งค่าที่ได้จะนำไปใช้เป็นค่าตั้งต้นในการแจกงาน ผู้วิจัยจะกล่าวถึง Resource Information Estimation โดยละเอียดอีกครั้งในส่วนของหัวข้อที่ 3-3
2. เมื่อมียูสเซอร์เริ่มร้องขอการประมวลผล (Submit Job)
3. JobTracker จะแบ่งขนาดของข้อมูลที่จะประมวลผลโดยใช้การวิเคราะห์ลักษณะงาน (Workload Characteristic Analysis) ว่าลักษณะงานเป็นงานประเภทของงานที่ต้องใช้ความสามารถในการประมวลผล (Processing Bound Job) หรือ งานที่ต้องใช้

ความสามารถในการอ่านเขียนข้อมูล (I/O Bound Job) โดยจะใช้สมการการแบ่งงานที่แตกต่างกัน โดยหลักการซึ่งผู้วิจัยได้กล่าวไว้ในเกริ่นนำของบทที่ 3 ซึ่งเครื่องที่มีความสามารถในการทำงานมากก็จะได้งานมากด้วยเช่นเดียวกัน โดยผู้วิจัยจะกล่าวถึงโดยละเอียดอีกครั้งในส่วนของหัวข้อที่ 3-4

4. หลังจากที่แบ่งงานด้วยอัลกอริทึม EFTF แล้ว อัลกอริทึมยังปรับปรุงการแจกงานของส่วนแจกงานโดยมีการแก้ไขให้เหมาะสมกับการแบ่งงานที่แบ่งไว้ในข้อที่ 3 รวมไปถึงได้นำเอาหลักการของการรอคอยเพื่อจะได้การประมวลผลที่เร็วกว่า (Delay Task Assignment) โดยผู้วิจัยจะกล่าวถึงโดยละเอียดอีกครั้งในส่วนของหัวข้อที่ 3-4 ในขั้นตอนของการตัดสินใจแจกงานของส่วนแจกงาน
5. JobTracker แจกงานให้ TaskTracker เพื่อประมวลผลงานที่ได้รับมอบหมาย
6. TaskTracker ประมวลผลงาน รายงานสถานะของการประมวลผลให้กับส่วนแจกงานที่อยู่ภายใน JobTracker เพื่อให้มีการติดตามสถานะและสามารถปรับเปลี่ยนพารามิเตอร์ให้เหมาะสมกับสถานการณ์ในคลัสเตอร์

3.3 การสร้างข้อมูลเพื่อนำไปใช้ในการแจกงาน (Resource Information Estimation)

จากที่กล่าวไว้ในหัวข้อที่ 3.1 การสร้างข้อมูลเพื่อนำไปใช้ในการแจกงานนั้นจะได้จากหลักการการควบคุมการแออัดของข้อมูล (Congestion Control) ซึ่งการทำการควบคุมการแออัดของข้อมูลจะเป็นรูปแบบตามที่กล่าวไว้ในหัวข้อที่ 2.1.5 และ ในหัวข้อที่ 3.1 ในขั้นตอนที่ 4 ถึงขั้นตอนที่ 9 ในการได้มาซึ่งข้อมูลเหล่านี้นั้นจะมีขั้นตอนการทำงานดังรูปที่ 3-2



รูปที่ 3-2 แสดงการได้มาซึ่งข้อมูลประสิทธิภาพของการประมวลผลและI/O

1. จะทำการเตรียมข้อมูลไว้ที่เครื่องต่างๆในคลัสเตอร์เพื่อนำไปใช้ในการทำ Congestion Control เมื่อทำการเริ่มต้นคลัสเตอร์ (Cluster Startup) จะทำการประมวลผลหาค่าของประสิทธิภาพของเครื่องข่ายและ I/O ภายนอกรวมไปถึงประสิทธิภาพในการประมวล การทำการทดสอบนั้นจะทำทั้งเครื่องตัวเองและเครื่องที่อยู่ในระยะไกลในคลัสเตอร์เดียวกันด้วย
2. หลังจากกระบวนการนี้เสร็จสิ้นกระบวนการจะทำการเขียนไฟล์ด้วย SFTP ไปที่เครื่อง JobTracker เพื่อทำการปรับข้อมูลให้ทันสมัยก่อนที่เริ่มต้นประมวลผลงานแรก โดย JobTracker จะเก็บข้อมูลที่รวบรวมได้ไว้ใช้อ้าวอิงในการทำงานภายหลัง

JobTracker อ่านไฟล์แล้วสร้างเป็นข้อมูลในแบบ JSON Format ดังกล่าวไว้ในหัวข้อที่ 2.1.6 โดยจะประกอบด้วย Attribute ต่างๆดังต่อไปนี้ โดยจะแสดงในรูปที่ 3-3 และตัวอย่างที่ได้จากการประมวลผลจริงแสดงในรูปที่ 3-4

```
เครื่องที่ประมวลผล 1 :
        {
                Array Size : ขนาดของ Array
                Progress Rate: ความเร็วในการ Process
                I/O Rate : {
                        เครื่องที่อยู่ในคลัสเตอร์เดียวกัน 1 : [1234, 686 …, 546]
                        เครื่องที่อยู่ในคลัสเตอร์เดียวกัน 2 : [1234, 686 …, 546]
                        .
                        .
                        เครื่องที่อยู่ในคลัสเตอร์เดียวกัน n : [1234, 686 …, 546]
                }
        }
เครื่องที่ประมวลผล 2 :
        {

        }
```

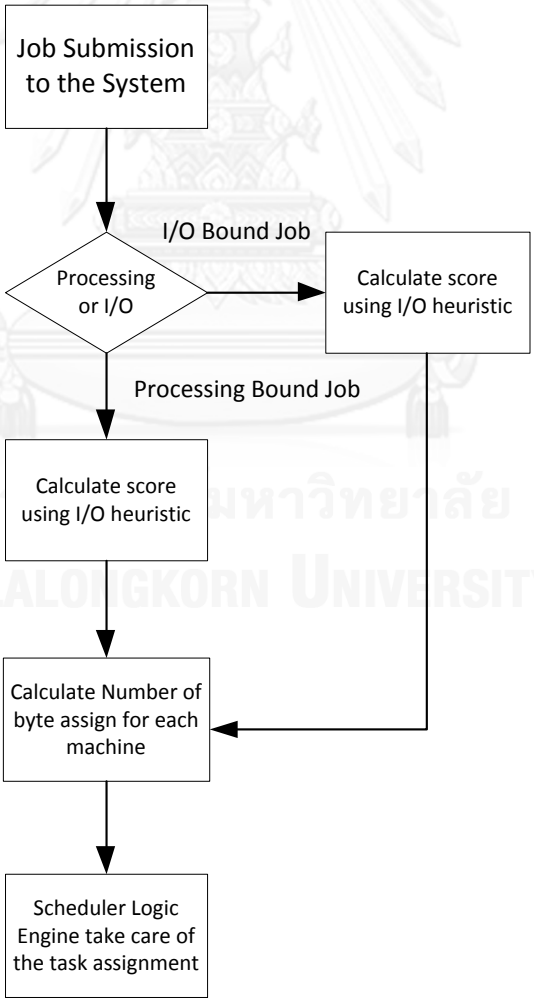
รูปที่ 3-3 แสดงรูปแบบฟอร์แมต JSON ที่ใช้ในการออกแบบการแจกงาน

```
"d1c-3":                                          // Machine Name
    {   "fixedArraySize":5                        // History Size
        "progressRate":[547,686,647,717,691],     // Progress Rate
        "ioRate":{                                // I/O Rate For each Machine
            "d1c-3":[38,38,41,36,38],
            "namenode":[3,6,4,5,3]
        }
    }
}
```

รูปที่ 3-4 ตัวอย่างฟอร์แมต JSON ที่ได้จากการแจกงานในคลัสเตอร์

3.4 กระบวนการการแบ่งงานในคลัสเตอร์ (Job Split Process)

เมื่อได้ข้อมูลของคลัสเตอร์ในส่วนของประมวลผลหาค่าของประสิทธิภาพของเครื่องข่ายและ I/O ภายนอกรวมไปถึงประสิทธิภาพในการประมวลแล้วนั้นเราจะสามารถนำมาใช้ในการแบ่งงานภายในคลัสเตอร์ด้วยวิธีการดังแสดงในรูปที่ 3-5



รูปที่ 3-5 ผังงานการทำงานของกระบวนการแบ่งงานในคลัสเตอร์

จากผังงานในรูปที่ 3-5 สามารถอธิบายการทำงานของกระบวนการแบ่งงานในคลัสเตอร์ได้ตามขั้นตอนดังต่อไปนี้

1. ขั้นตอนการวิเคราะห์ลักษณะของงาน ผู้ส่งงานจะต้องระบุใน properties file เพื่อแยกแยะระหว่างงานที่เป็น งานที่ใช้เขียนอ่านข้อมูลเป็นหลัก (I/O Bound) หรือว่างานที่ใช้การประมวลผลเป็นหลัก (Processing Bound)
2. ขั้นตอนการหาคะแนนในการแจกงาน ในการให้คะแนนสำหรับงานที่ต่างชนิดกัน ระบบจะพิจารณาจากสมการที่แตกต่างกัน โดยกำหนดให้ ความสามารถในการใช้เขียนอ่านข้อมูล (I/O Rate: $IOR_{1...NM}$), ความสามารถในการใช้หน่วยประมวลผล (Processing Rate: $PR_{1...NM}$) และ คะแนนในการประมวลผล (Performance Ratio: $PerfR_{1...NM}$) ดังนั้นถ้าหากงานเป็นลักษณะของงานที่ใช้เขียนอ่านข้อมูลเป็นหลัก (I/O Bound) จะใช้สมการที่ 3-1 มาทำการคำนวณ

$$PerfR_{1...NM} = 2IOR_{1...NM} + PR_{1...NM} \quad (3\text{-}1)$$

แต่ถ้าหากเป็นลักษณะงานที่ใช้การประมวลผลเป็นหลัก (Processing Bound) จะใช้สมการที่ 3.2 มาทำการคำนวณ

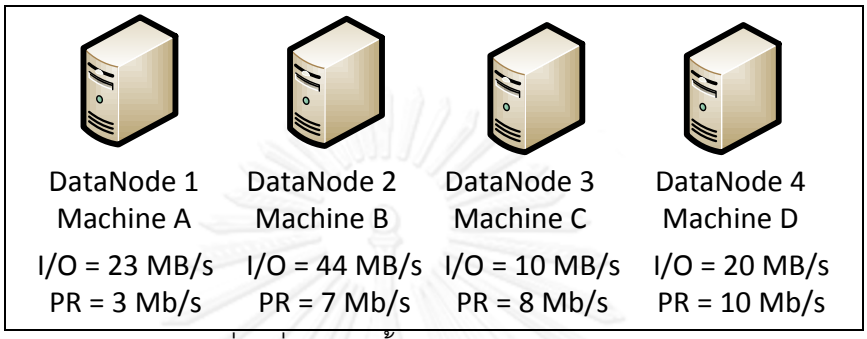
$$PerfR_{1...NM} = IOR_{1...NM} + 2PR_{1...NM} \quad (3\text{-}2)$$

3. ขั้นตอนการคำนวณขนาดที่จะแจกงานไปในแต่ละเครื่อง จากค่าของคะแนนในการประมวลผล (Performance Ratio: $PerfR_{1...NM}$) ในขั้นตอนที่ 2 และกำหนดให้ งานทั้งหมดนั้นมีขนาดเท่ากับ N Bytes, จำนวนงานที่ต้องการแบ่ง (n) และขนาดของงานที่จะประมวลผล ($V_{1...n}$) ผู้วิจัยนำเสนอการคำนวณหาขนาดที่จะแจกงานไปแต่ละเครื่องจากสมการที่ 3-3 และ สมการที่ 3-4

$$V_{1...n} = \frac{\frac{PerfR_i}{\sum_{i=1}^{MN} PerfR_i} \times N}{J} \quad (3\text{-}3)$$

โดยที่ $n = NM \times J$

เพื่อความเข้าใจที่ชัดเจนมากขึ้น ผู้วิจัยขอนำเสนอตัวอย่างโดยตัวแปรที่ใช้จะเป็นค่าที่ใกล้เคียงกับการทดลองจริง แต่จะมีการปัดเศษทศนิยมเพื่อให้เข้าใจง่ายและไม่เป็นการซับซ้อน ค่าที่แน่นอนตามจริงจะแสดงในบทที่ 5 ในส่วนของการทดลอง โดยรูปที่ 3-6 แสดงรูปแบบของเครื่องที่กำหนดขึ้น



รูปที่ 3-6 แสดงรูปแบบของเครื่องที่กำหนดขึ้นใช้เป็นตัวอย่างในการคำนวณขนาดของงานที่แจก

หากกำหนดให้ขนาดของงานที่เข้ามาในระบบเป็น 2GB และงานเป็นลักษณะของงานที่ใช้เขียนอ่านข้อมูลเป็นหลัก (I/O Bound) จะสามารถคำนวณได้ดังตารางที่ 3-3

ตารางที่ 3-3 แสดงการคำนวณขนาดของงานที่แจกไปในแต่ละเครื่อง

ชื่อเครื่อง	$PerfR_{1...n}$	$Num_{B_i} = \frac{PerfR_i}{\sum_{i=1}^{Num_m} PerfR_i} \times N$	Bytes Number
DataNode 1	2(23)+3 = 49	$\frac{49}{222} \times 2048 = 452.036$	452
DataNode 2	2(44)+7 = 96	$\frac{96}{222} \times 2048 = 885.621$	886
DataNode 3	2(10)+8 = 28	$\frac{28}{222} \times 2048 = 258.360$	258
DataNode 4	2(20)+10 = 50	$\frac{50}{222} \times 2048 = 461.261$	452
	SUM($PerfR_{1...n}$) = 222		

4. <u>ขั้นตอนของการตัดสินใจแจกงานของส่วนแจกงาน</u> ในฮาดูปปัจจุบันนั้นคำนึงถึงหลักการที่ว่าเครื่องที่มีข้อมูลจะได้งานไปก่อนแต่ส่วนแจกงานของอัลกอริทึม EFTF (Earliest Finish Time First) จะอาศัยหลักการที่ว่า การแจกงานไม่จำเป็นต้องแจกงานโดยทันที บางสถานการณ์ในคลัสเตอร์แบบต่างชนิดนั้นสามารถ ชะลอการแจกงานเพื่อให้เกิดประสิทธิภาพที่ดีกว่า ดังนั้นอัลกอริทึม EFTF จะคำนึงถึง Data

Locality และเครื่องที่ควรจะได้งานชิ้นนั้นไปทำงาน (Expected Host) เมื่อพิจารณาจากประสิทธิภาพ ดังนั้นในอัลกอริทึมจะพิจารณาการแจกงานโดยใช้ปัจจัยดังกล่าวเพิ่มขึ้น ผู้วิจัยได้จัดกลุ่มของสถานการณ์ในการแบ่งงานไว้ 4 แบบดังต่อไปนี้

กำหนดให้ เครื่องที่มีข้อมูลอยู่ $m_L = \{m_{L_1}, m_{L_2}, m_{L_3}, ..., m_{L_n}\}$

เครื่องที่ร้องของงาน m_R

เครื่องที่คาดหมาย m_E

เวลาที่ใช้ในการประมวลผลงานที่มีเหลืออยู่ที่เครื่อง $m_{L_i} = t_{rm_{L_i}}$

เวลาที่ใช้ในการประมวลผลงานใหม่ในเครื่อง $m_{L_i} = t_{nm_{L_i}}$

เวลาที่ใช้ในการประมวลผลงานของเครื่อง $m_R = t_{nm_R}$

เวลาที่ใช้ในการอ่านข้อมูลข้ามเครื่องจากเครื่อง m_L ไปยังเครื่อง m_R มีค่าเป็น $t_{T_{L\to R}}$

เวลาที่ใช้ในการประมวลผลงานที่มีเหลืออยู่ในเครื่อง $m_E = t_{rm_E}$

เวลาที่ใช้ในการประมวลผลงานใหม่ในเครื่อง $m_E = t_{nm_E}$

เวลาที่ใช้ในการอ่านข้อมูลข้ามเครื่องจากเครื่อง m_L ไปยังเครื่อง m_E มีค่าเป็น $t_{T_{L\to E}}$

- <u>เครื่องที่ร้องของงานนั้นมีข้อมูลอยู่และเป็นเครื่องที่คาดหมาย</u> ($m_R \in m_L$ และ $m_R = m_E$) เป็นกรณีที่ดีที่สุดที่สามารถเกิดขึ้นได้เนื่องจากในขั้นตอนของการแบ่งงานได้คำนึงถึงประสิทธิภาพของการทำงานทำให้ได้มาซึ่งเครื่องที่คาดหมาย รวมไปถึงเครื่องที่ร้องขอยังมีข้อมูลอยู่จึงไม่ต้องคัดลอกข้อมูลจากเครื่องอื่น เพราะฉะนั้นส่วนแจกงานจะให้งานแก่เครื่องที่ร้องของงานโดยทันที
- <u>เครื่องที่ร้องของงานนั้นไม่มีข้อมูลอยู่แต่เป็นเครื่องที่คาดหมาย</u> ($m_R \notin m_L$ แต่ $m_R = m_E$) กรณีนี้ส่วนแจกงานจะต้องพิจารณาว่าจะแจกงานไปที่เครื่องที่มาของงานหรือรอเครื่องที่มีข้อมูลอยู่ทำงานเสร็จแล้วจึงแจกงานไป โดยเปรียบเทียบกันระหว่างเครื่องที่ร้องของงานที่ไม่มีข้อมูลแต่เป็นเครื่องที่มีประสิทธิภาพการทำงานที่ดีหากคัดลอกข้อมูลจากเครื่องอื่นแล้ว จะทำงานได้เร็วกว่าเครื่องที่มีข้อมูลอยู่แล้วแต่ยังทำงานไม่เสร็จสิ้น ดังนั้นจะสามารถแบ่งได้เป็น 2 กรณี สามารถอธิบายได้ดังต่อไปนี้

- แจกให้กับเครื่องที่มาของงานก็ต่อเมื่อ สมการที่ 3-5 เป็นจริง โดยกำหนดให้เครื่อง m_L เป็นเครื่องที่มีข้อมูลอยู่ และ เครื่อง m_R เป็นเครื่องที่มาของงาน

$$t_{rm_{L_i}} + t_{nm_{L_i}} > t_{nm_r} + t_{T_{L\to R}} \tag{3-4}$$

$$\frac{V_{m_{L_i}}(1-PRogR_{m_{L_i}})}{PR_{m_{L_i}}} + \frac{V_{Assign}}{PR_{m_{L_i}}} > \frac{V_{Assign}}{PR_{m_R}} + \frac{V_{Assign}}{IOR_{m_R}} \tag{3-5}$$

- รอเครื่องที่มีข้อมูลอยู่ทำงานเสร็จแล้วจึงแจกงานไป ก็ต่อเมื่อสมการที่ 3-5 ไม่เป็นจริง

- เครื่องที่ร้องของงานนั้นมีข้อมูลอยู่แต่ไม่เป็นเครื่องที่คาดหมาย ($m_R \in m_L$ แต่ $m_R \neq m_E$) กรณีนี้ส่วนแจกงานจะต้องพิจารณาว่าจะแจกงานไปที่เครื่องที่มาของงานซึ่งมีข้อมูลอยู่ แต่ไม่ใช่เครื่องที่มีประสิทธิภาพที่ดีในการประมวลผงานนั้น ดังนั้นจะสามารถแบ่งได้เป็น 2 กรณีสามารถอธิบายได้ดังต่อไปนี้
  - แจกให้กับเครื่องที่มาของงานก็ต่อเมื่อ ต่อเมื่อ สมการที่ 3-7 เป็นจริง โดยกำหนดให้เครื่อง m_L เป็นเครื่องร้องขอและมีข้อมูลอยู่ และ เครื่อง m_E เป็นเครื่องที่คาดหมายเมื่อประมวลผลแล้วจะมีประสิทธิภาพดี

$$t_{rm_E} + t_{nm_E} + t_{T_{L\to E}} > t_{nm_{L_i}} \tag{3-6}$$

$$\frac{V_{m_E}(1-PRogR_{m_E})}{PR_{m_E}} + \frac{V_{Assign}}{PR_{m_E}} + \frac{V_{Assign}}{IOR_{m_E}} > \frac{V_{Assign}}{PR_{m_{L_i}}} \tag{3-7}$$

  - รอเครื่องที่มีข้อมูลอยู่ทำงานเสร็จแล้วจึงแจกงานไป ก็ต่อเมื่อสมการที่ 3-7 ไม่เป็นจริง

- เครื่องที่ร้องของงานนั้นไม่มีข้อมูลอยู่และไม่เป็นเครื่องที่คาดหมาย ($m_R \notin m_L$ และ $m_R \neq m_E$) จะใช้หลักการของการแจกงานแบบฮาดูปเดิมนั้นคือจะเลือกเครื่องใดที่มีประสิทธิภาพในการอ่านข้อมูลเครื่องนั้นจะได้งานไปก่อน

บทที่ 4
ขั้นตอนการพัฒนาระบบในเชิงปฏิบัติ

4.1 การสร้างข้อมูลเพื่อนำไปใช้ในการแจกงาน (Resource Monitoring Information)

โดยภายในส่วนของการสร้างข้อมูลเพื่อนำไปแจกงานนั้นผู้วิจัยขอแบ่งการทำงานออกเป็น 3 ส่วนย่อยดังต่อไปนี้

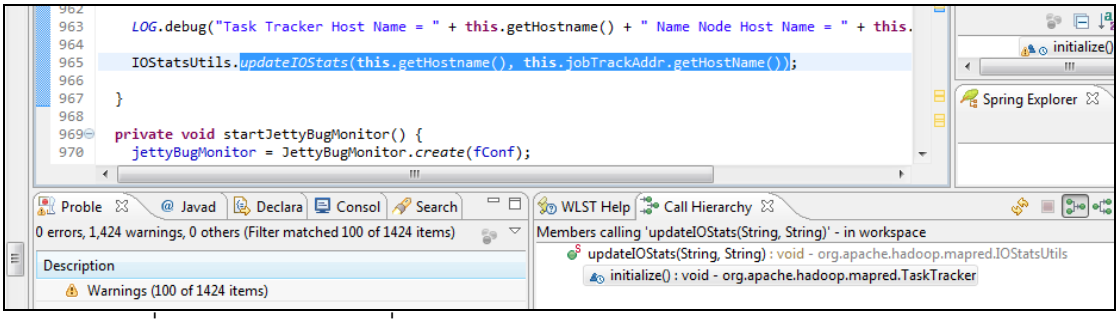
4.1.1 ข้อมูลเพื่อนำไปใช้ในการแจกงาน (Resource Monitoring Information)

ผู้วิจัยได้สร้างส่วนของการสร้างข้อมูลเพื่อการแจกงานในคลัสเตอร์ โดยได้สร้าง class ที่เรียกว่า IOStatUtils ขึ้นมาเพื่อทำการสร้างข้อมูลเมื่อคลัสเตอร์เริ่มทำการเริ่มต้น (Cluster Startup) โดยทางผู้วิจัยได้ตัดจาวาโปรแกรมบางส่วนของมาแสดงให้เห็นในรูปที่ 4.1

```
public static void updateIOStats(String hostname, String namenode) throws
IOException{
    FixedSizeArrayList fsa = new FixedSizeArrayList(50);
    File[] fileList = new File("./IORateFile").listFiles();
    Arrays.sort(fileList, SizeFileComparator.SIZE_COMPARATOR);
    for(File eachOfFile : fileList){
            IORateUtils iru = new IORateUtils(eachOfFile.getTotalSpace(),
eachOfFile.getPath());
            long start = System.currentTimeMillis();
            iru.compareWithBufferSize(256000);
            if((long) iru.getIORate() > 100){continue;}
            fsa.add((long) iru.getIORate());
            if(System.currentTimeMillis() - start > 20000){break;}
    }
    OutputFileUtils.writeRemoteFiles(String.valueOf(fsa.average()), hostname +
".monitor", namenode);
  }
```

รูปที่ 4-1 แสดงคลาส IOStatUtils เพื่อทำการสร้างข้อมูลเมื่อคลัสเตอร์เริ่มทำการเริ่มต้น

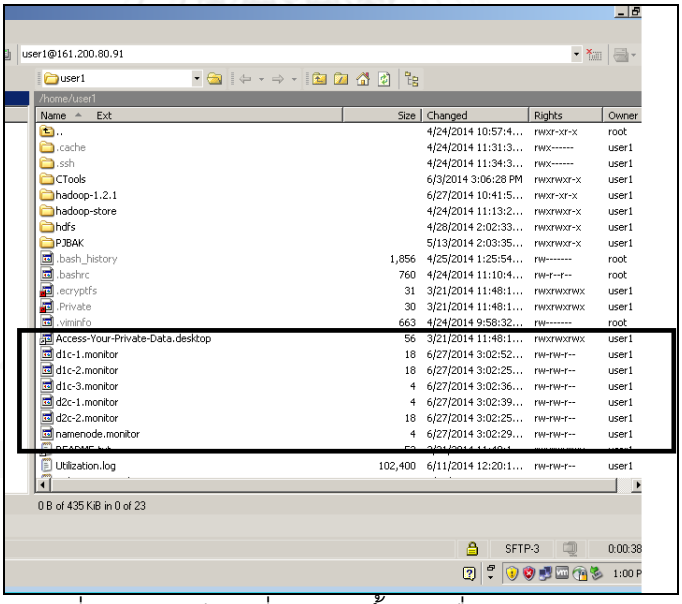
จะทำการเตรียมข้อมูลไว้ที่เครื่องต่างๆในคลัสเตอร์เพื่อนำไปใช้ในการทำ Congestion Control เมื่อทำการเริ่มต้นคลัสเตอร์ (Cluster Startup) จะทำการประมวลผลหาค่าของประสิทธิภาพของเครื่องข่ายและ I/O ภายนอกรวมไปถึงประสิทธิภาพในการประมวล การทำการทดสอบนั้นจะทำทั้งเครื่องตัวเองและเครื่องที่อยู่ในระยะไกลในคลัสเตอร์เดียวกันด้วย โดย class นั้นจะถูกเรียกในส่วนของการเริ่มต้น TaskTracker (Startup TaskTracker) ในไฟล์ TaskTracker.java หากผู้อ่านต้องการทราบถึงบรรทัดที่แน่นอนแสดงในรูปที่ 4-2 หลังจากขั้นตอนนี้จะทำการส่งข้อมูลไปรวมไว้ที่เครื่อง JobTracker โดยจะกล่าวถึงในหัวข้อที่ 3.2



รูปที่ 4-2 แสดงตำแหน่งที่ทำการเรียก Class IOStatUtils ใน TaskTracker.java

4.1.2 การรวมข้อมูลไปที่ JobTracker (TaskTracker Remote Transfer Information)

ผู้วิจัยได้สร้างส่วนของการ Remote Transfer Information จากเครื่องในคลัสเตอร์เพื่อทำการแจ้งข้อมูลให้ JobTracker ทราบ เพื่อนำข้อมูลดังกล่าวที่ได้จาก TaskTracker ไปใช้ในการแบ่งงานต่อไป โดยได้สร้าง Utility Class ขึ้นมาชื่อว่า OutputFileUtils โดยไลบรารี่ที่ใช้นั้นเป็นของ JSCH Community open source [15] โดย source code จะถูกเรียกในส่วนของขั้นตอนที่ 4.1.1 เสร็จสิ้นดังแสดงให้เห็นไปในรูปที่ 4-2 ไปแล้ว



รูปที่ 4-3 แสดงไฟล์ที่ถูกสร้างขึ้นในเครื่อง JobTracker

4.1.3 การรวมข้อมูลให้เป็นฟอร์แมต JSON

จากขั้นตอนที่ 4.1.2 เมื่อได้ไฟล์จากเครื่องต่างๆที่อยู่ในคลัสเตอร์แล้ว จากนั้นจะทำการรวมไฟล์ (Merge) เพื่อให้อยู่ในฟอร์แมตที่สามารถแปลงเป็น Object และ แปลงกลับเป็นตัวอักษร (Text) ได้อย่างสะดวก โดยในงานวิจัยนี้ผู้วิจัยได้ใช้ JSON ฟอร์แมตมาช่วยในส่วนนี้โดยได้สร้าง Utility Class ขึ้นมาชื่อว่า JSONUtils โดยไลบรารี่ที่ใช้นั้นเป็นของ Google Community open source [16] โดยมีลักษณะของ Source Code ที่ใช้ในการเปลี่ยนรูปแบบจาก JSON เป็น Object และ Object เป็น JSON ดังแสดงในรูปที่ 4-4

Object to JSON

```
OutputFileUtils.writeFileMonitor("./monitor.json", new
Gson().toJson(clusterDetails));
```

JSON to Object

```
public static HashMap<String, ClusterMonitorDetails> jsonToObject(
      String jsonText) {
    HashMap<String, ClusterMonitorDetails> hcm =
        new HashMap<String, ClusterMonitorDetails>();
    JsonParser jsonp = new JsonParser();
    JsonElement jsonRoot = jsonp.parse(jsonText);
    for (Entry<String, JsonElement> eachOfentry : jsonRoot.getAsJsonObject()
        .entrySet()) {
      JsonElement l1 = eachOfentry.getValue();
      int arrLen = l1.getAsJsonObject().get("fixedArraySize").getAsInt();
      ClusterMonitorDetails cmds = new ClusterMonitorDetails(arrLen);
      FixedSizeArrayList fsPr = cmds.getProgressRate();
      JsonArray ja1 =
          l1.getAsJsonObject().get("progressRate").getAsJsonArray();
      for (int i = 0; i < ja1.size(); i++) {
        fsPr.add(ja1.get(i).getAsLong());
      }
      JsonElement l2 = l1.getAsJsonObject().get("ioRate").getAsJsonObject();
      Map<String, FixedSizeArrayList> mapIO = cmds.getIoRate();
      for (Entry<String, JsonElement> eoel2 : l2.getAsJsonObject()
          .entrySet()) {
        JsonArray ja2 = eoel2.getValue().getAsJsonArray();
        FixedSizeArrayList fsPr2 = new FixedSizeArrayList(arrLen);
        for (int j = 0; j < ja2.size(); j++) {
          fsPr2.add(ja2.get(j).getAsLong());
        }
        mapIO.put(eoel2.getKey(), fsPr2);
      }
      hcm.put(eachOfentry.getKey(), cmds);
    }
    return hcm;
  }
```

รูปที่ 4-4 แสดงการการเปลี่ยนรูปแบบจาก JSON เป็น Object และ Object เป็น JSON

```
{
	"namenode":{
		"fixedArraySize":50,

	"progressRate":[671,409,691,639,0,0,0,0,404,568,1058,0,0,0,1273,1266,1315,0,0,0,1406,1388,1099,1109,1344,1348,0,0,0,
599,593,553,365,697,672,398,716,566,359,682,478,0,0,1381,1351,1203,1220,1508,1455,0],
		"ioRate":{

	"namenode":[4,4,4,2,3,3,2,3,2,3,3,0,0,0,0,2,5,10,10,3,3,2,2,3,4,4,2,2,3,3,17,4,4,3,2,4,4,2,4,3,3,5,3,3,4,4,3,3,4,4]
		}
	},
	"d1c-3":{
		"fixedArraySize":50,

	"progressRate":[703,650,825,717,652,717,928,686,807,721,739,635,890,607,789,914,860,736,672,742,866,715,782,841,67
4,727,912,843,1585,1563,0,0,0,710,670,674,625,687,596,800,659,611,730,787,0,1479,1474,0,0,0],
		"ioRate":{
			"d1c-
3":[38,38,41,36,38,38,36,35,37,37,28,30,70,62,52,54,43,44,43,44,38,48,44,52,52,52,53,57,43,54,49,49,45,47,27,59,59,63,61,61,59
,59,59,58,58,58,58,58,46,18],
			"namenode":[7,5,7,8,5,4,4,5,5,6,7,7,6,5,5,6,6,6,7,7,5,7,5,6,7,7,7,7,7,6,6,7,7,5,6,7,5,6,4,7,7,4,7,5,7,7,5,5,7,6]
		}
	},
}
```


รูปที่ 4-5 แสดงรูปแบบของไฟล์ฟอร์แมต JSON

เมื่อทำการรวมไฟล์ (Merge) เสร็จสิ้นจะเขียนลงไฟล์ในเครื่องของ JobTracker ชื่อว่า monitor.json เพื่อนำไปใช้ในการปรับข้อมูลให้ทันสมัยต่อไป โดยรูปแบบของไฟล์จะเป็นไปตามที่แสดงไว้ในบทที่ 3 ในหัวข้อที่ 3.2 โดยข้อมูลจริงจะแสดงในรูปที่ 4-5

4.2 การแบ่งงานในคลัสเตอร์ (Job Split Process)

ผู้วิจัยใช้วิธีการอธิบายตามรูปที่ 3-5 โดยจะอธิบายในลักษณะของการทำงานโดยการแบ่งงานนั้นจะสามารถแบ่งเป็นขั้นตอนได้ดังต่อไปนี้

4.2.1 การวิเคราะห์ลักษณะงาน (Job Characteristics)

ผู้วิจัยต้องการแยกแยะระหว่างงานที่เป็น งานที่ใช้เขียนอ่านข้อมูลเป็นหลัก (I/O Bound) หรือว่า งานที่ใช้การประมวลผลเป็นหลัก (Processing Bound) โดยในงานวิจัยจะให้ผู้ดูแลระบบเป็นผู้กำหนดลักษณะของงานว่าเป็นลักษณะใด โดยจะมี Properties ให้กำหนด โดยได้ทำการ Reuse Framework ของฮาดูปเอง โดยสามารถกำหนดใน XML ดังรูปที่ 4-6

```
<configuration>
        <property>
                <name>mapred.job.tracker</name>
                <value>namenode:54311</value>
                <description>The host and port that the MapReduce job tracker runs
 at.  If "local", then jobs are run in-process as a single map
 and reduce task.
        </description>
        </property>
        <property>
                <name>mapred.child.java.opts</name>
                <value>-Xmx512m</value>
        </property>
        <property>
                <name>org.apache.hadoop.examples.WordCount</name>
                <value>IO</value>
        </property>
        <property>
                <name>org.apache.hadoop.examples.Sort</name>
                <value>Processing</value>
        </property>
</configuration>
```

รูปที่ 4-6 แสดงรูปแบบของการตั้งค่าเพื่อกำหนดลักษณะของงาน

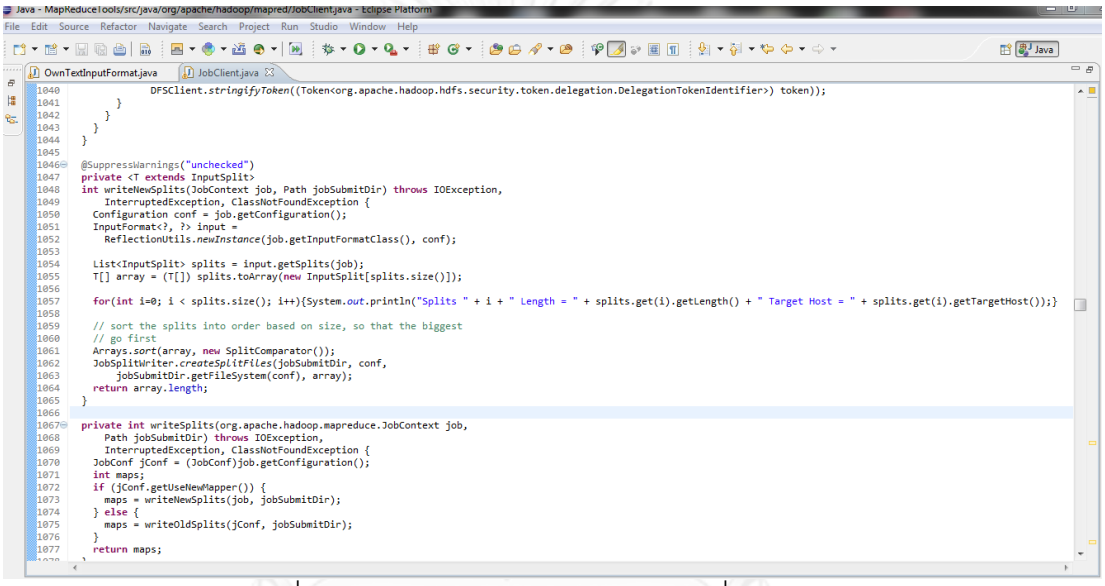
```
public List<InputSplit> getSplits(JobContext job) throws IOException {
    String monitorContent = OutputFileUtils.readFileMonitor("./monitor.json");
    Map<String, ClusterMonitorDetails> clusterDetails = JSONUtils.jsonToObject(monitorContent);
    updateOwnerIORate(clusterDetails);
    OutputFileUtils.writeFileMonitor("./monitor.json", new Gson().toJson(clusterDetails));
    if(clusterDetails.size() == 0){return super.getSplits(job);}
    else {
      List<InputSplit> splits = new ArrayList<InputSplit>();
      List<FileStatus> files = listStatus(job);
      for (FileStatus file: files) {
        Path path = file.getPath();
        FileSystem fs = path.getFileSystem(job.getConfiguration());
        long length = file.getLen();
        BlockLocation[] blkLocations = fs.getFileBlockLocations(file, 0, length);
        if ((length != 0) && isSplitable(job, path)) {
          SplitEntity[] splitSize = ownComputeSplitSize(length, 3, clusterDetails);
          long bytesRemaining = length;
          for(int i=0; i < splitSize.length; i++){
            int blkIndex = getBlockIndex(blkLocations, length-bytesRemaining);
            splits.add(new FileSplit(path, length-bytesRemaining, splitSize[i].getSplitSize(),
                blkLocations[blkIndex].getHosts(), splitSize[i].getHostname()));
            bytesRemaining -= splitSize[i].getSplitSize();
          }
          if (bytesRemaining > 0) {
            splits.add(new FileSplit(path, length-bytesRemaining, bytesRemaining,
                       blkLocations[blkLocations.length-1].getHosts(), this.bestMachine));
          }
        }
      }

      job.getConfiguration().setLong(NUM_INPUT_FILES, files.size());

      LOG.debug("Total # of splits: " + splits.size());
      return splits;
    }
  }
```

```
protected SplitEntity[] ownComputeSplitSize(long totalFileLength, int subSplitSize,
Map<String, ClusterMonitorDetails> clusterDetails){
    SplitEntity[] splitSizes = new SplitEntity[clusterDetails.size()*subSplitSize];
    int splitSizeIndex = 0;
    double sumWeightRatio = 0;
    long byteremainning = new Long(totalFileLength);
    double tempWeightRatio = 0;
    for(String hostName : clusterDetails.keySet()){
      ClusterMonitorDetails hostCM = clusterDetails.get(hostName);
      double weightRatio = hostCM.splitWeightRatio(hostName);
      sumWeightRatio += weightRatio;
      if(tempWeightRatio < weightRatio){this.bestMachine = hostName;tempWeightRatio =
weightRatio;}
    }
    Set<String> allHostName = clusterDetails.keySet();
    String[] allAHostName = allHostName.toArray(new String[allHostName.size()]);
    for(int i=0; i < allAHostName.length; i++){
      ClusterMonitorDetails hostCM = clusterDetails.get(allAHostName[i]);
      double assignSize =
hostCM.splitWeightRatio(allAHostName[i])*totalFileLength/sumWeightRatio;
      long exactlyAssign = 0;
      for(int j=0; j < subSplitSize; j++){
        splitSizes[splitSizeIndex] = new SplitEntity();
        splitSizes[splitSizeIndex].setSplitSize(Math.round(assignSize/3));
        splitSizes[splitSizeIndex].setHostname(allAHostName[i]);
        exactlyAssign += splitSizes[splitSizeIndex].getSplitSize();
        splitSizeIndex++;
      }
      byteremainning -= exactlyAssign;
    }
    return splitSizes;
  }
```

รูปที่ 4-7 แสดงลักษณะของ Source Code การคำนวณค่าคะแนนในการแจกงานและการแบ่งงาน

4.2.2 การคำนวณค่าคะแนนในการแจกงานและการแบ่งงาน

ผู้วิจัยได้ทำการสร้าง class ของตัวเองขึ้นมาโดย extend class ของฮาดูปเดิมโดยเปลี่ยนแปลงในส่วนของฟังก์ชั่น getSplits โดยทำการปฏิบัติตามขั้นตอนที่กล่าวไปในหัวข้อที่ 3.3 ในหัวข้อย่อยที่ 3 โดยจะลักษณะของโค๊ดเป็นไปตามรูปที่ 4-7 โดยฟังก์ชั่นนั้นจะถูกเรียกโดย class ของฮาดูปที่ชื่อว่า JobClient.java ฟังก์ชั่นชื่อว่า writeNewSplits หากเป็น MapRed และ writeOldSplits หากเป็น MapReduce แสดงในรูปแบบของ screenshot ในรูปที่ 4-8



รูปที่ 4-8 แสดง Call Hierarchy ของฟังก์ชั่น getSplits

4.3 การระบุเครื่องที่จะแจกงาน (Job Assignment on Machine)

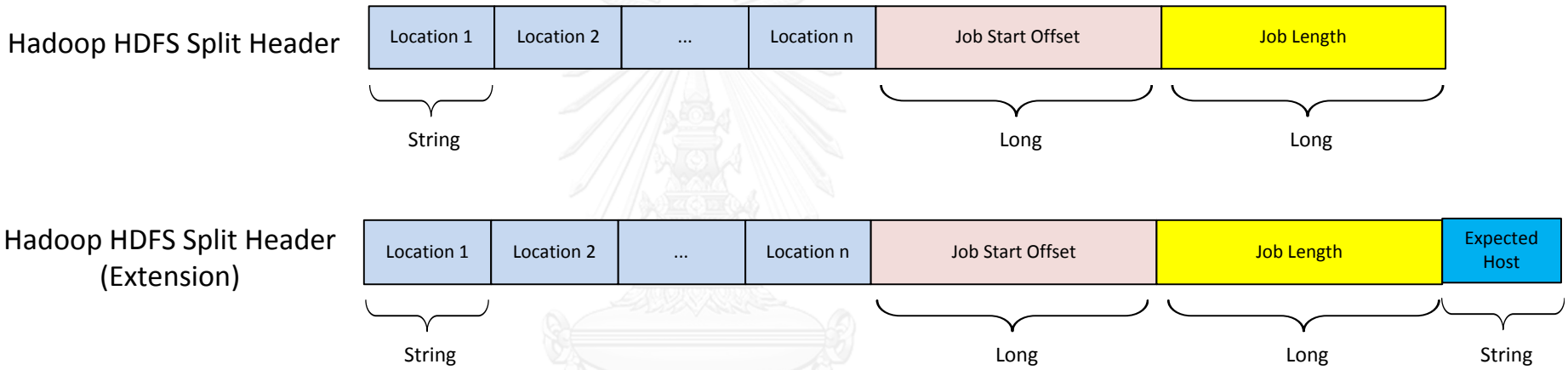
จากรูปที่ 3-1 ในขั้นตอนที่ 14 ได้กล่าวไว้ถึงการแจกงานเมื่อมีการร้องขอใหม่ได้มีการปรับปรุงจากการแจกงานในสภาวะปกตินั้นคือมีการเพิ่มในส่วนของการแจกงานที่แน่นอนโดยส่วนแจกงานจะรู้เมื่อทำการแบ่งงานเสร็จว่าจะต้องแจกงานนี้ไปที่เครื่องๆใดในคลัสเตอร์เพื่อให้เกิดประสิทธิภาพสูงสุดในการประมวลผล โดยในทางปฏิบัติผู้เขียนได้มีการปรับปรุงในส่วนของฮาดูปในส่วนของ HDFS Header เพื่อทำการเขียนข้อมูลในส่วนของเครื่องที่จะถูกแจกงานนี้เข้าไปใน HDFS Header โดยมีขั้นตอนการทำงานดังต่อไปนี้

1. เริ่มทำการเปลี่ยนแปลง Object ของ SplitMetaInfo ของฮาดูปเดิมโดยการเพิ่ม Attribute เข้าไปใน Object การเปลี่ยนแปลงนั้นแสดงในรูปที่ 4-9

```
public static class TaskSplitMetaInfo {
    private TaskSplitIndex splitIndex;
    private long inputDataLength;
    private String[] locations;
    private String targetHost;           // Add in the Attribute
    public TaskSplitMetaInfo(){
      this.splitIndex = new TaskSplitIndex();
      this.locations = new String[0];
    }
    public TaskSplitMetaInfo(TaskSplitIndex splitIndex, String[] locations,
        long inputDataLength, String targetHost) {
      this.splitIndex = splitIndex;
      this.locations = locations;
      this.inputDataLength = inputDataLength;
      this.targetHost = targetHost;
    }
}
```

รูปที่ 4-9 แสดงการเพิ่ม Attribute เข้าไปในส่วนของ SplitMetaInfo

2. หลังจากนั้นจะเพิ่มส่วนของ HDFS Header โดยการเปลี่ยนแปลงจะเพิ่มส่วนของเครื่องเป้าหมายที่จะทำการประมวลผลงานนั้นๆ โดยจะเป็นการขยายต่อท้ายจาก HDFS Header เดิม โดยฟอร์แมต HDFS Header เดิมและ ฟอร์แมต HDFS Header ใหม่ของงานวิจัยจะแสดงการเปรียบเทียบกันในรูปที่ 4-10
3. โดยการแก้ HDFS Header จะต้องทำการแก้ Constructor ในส่วนของฟังก์ชั่นที่เรียกใช้โดยประกอบด้วยที่ต่างๆดังต่อไปนี้
    a. Class JobSplitWriter แก้ในฟังก์ชั่น writeNewSplits และ writeOldSplits
    b. Class SplitMetaInfoReader แก้ในฟังก์ชั่น readSplitMetaInfo
4. หากไม่มีการปรับปรุงในข้อที่ 3 จะทำให้ไม่มีข้อมูลในการกำหนดเครื่องที่เหมาะสมกับงานนั้นๆ
5. โดยได้มีการเปลี่ยนแปลงในส่วนของ JobInProgress.java เพื่อให้มีการคำนึงถึงเครื่องที่จะแจกงานไปโดยฟังก์ชั่นที่แก้นั้นมีชื่อว่า findTaskFromList



รูปที่ 4-10 แสดงการเปรียบเทียบฟอร์แมต HDFS Header เดิมและ ฟอร์แมต HDFS Header ใหม่ของงานวิจัย

4.4 การปรับปรุงค่าของข้อมูลเพื่อนำไปใช้ในการแจกงาน

จากรูปที่ 3-4 จะมีกระบวนการการปรับปรุงค่าของข้อมูลเพื่อนำไปใช้ในการแจกงาน โดยในงานวิจัยได้ทำการปรับปรุงค่านี้ให้ทันสมัยโดยทุกครั้งที่มีการส่ง Heartbeat ระหว่าง JobTracker และ TaskTracker ให้ทำการปรับปรุงค่าลงไปในไฟล์ชื่อ monitor.json ที่กล่างไว้ในส่วนที่ 4.1.3 โดยการปรับปรุงค่ามีขั้นตอนการปรับปรุงในส่วนที่แสดงในรูปที่ 4-11

```
if(oldProgress == 0 && tip.getProgress() > 0){
      long timetoCopyData = (System.currentTimeMillis() - tip.getExecStartTime());
      taskidtoTimeCopyData.put(status.getTaskID(), timetoCopyData);
    }
    if(status.getFinishTime() > status.getStartTime()){
      long diffTime = (status.getFinishTime() - status.getStartTime());
      if(taskidtoTimeCopyData.containsKey(status.getTaskID())){
        updateClusterMonitoring(tip, status, taskTrackerHostName);
        OutputFileUtils.writeFileMonitor("./monitor.json",
gs.toJson(clusterMonitorMap));
      } else {}
    } else {}
```

```
private void updateClusterMonitoring(TaskInProgress tip, TaskStatus ts, String
taskTrackerHostName){
    ClusterMonitorDetails cmds;
    if(clusterMonitorMap.containsKey(taskTrackerHostName)){
      cmds = clusterMonitorMap.get(taskTrackerHostName);
    } else {
      cmds = new ClusterMonitorDetails(50);
      clusterMonitorMap.put(taskTrackerHostName, cmds);
    }
    long progressRate = (tip.getMapInputSize()/(ts.getFinishTime() -
ts.getStartTime()));
    if(progressRate < cmds.getProgressRate().average() +
1000){cmds.getProgressRate().add(progressRate);}
    if(taskIdtoHDFSLocation.containsKey(tip.getTIPId())){
      String targetHDFSHostname =
taskIdtoHDFSLocation.get(tip.getTIPId()).getHostname();
      if(cmds.getIoRate().containsKey(targetHDFSHostname)){
        FixedSizeArrayList fsa = cmds.getIoRate().get(targetHDFSHostname);
        long ioRate =
(tip.getMapInputSize()/taskidtoTimeCopyData.get(ts.getTaskID()))/1000;
        if(ioRate < fsa.average() + 100){fsa.add(ioRate);}
      } else {
        FixedSizeArrayList fsa = new
ClusterMonitorDetails(50).createFixedArraySize();
        fsa.add((tip.getMapInputSize()/taskidtoTimeCopyData.get(ts.getTaskID())));
        cmds.getIoRate().put(targetHDFSHostname, fsa);
      }
    }
  }
```

รูปที่ 4-11 แสดงการวิธีการปรับปรุงค่าของข้อมูลเพื่อนำไปใช้ในการแจกงาน

บทที่ 5
การทดสอบและผลการทดลอง

การทดสอบระบบของการปรับปรุงอัลกอริทึมการจัดสรรงานสำหรับฮาดูปคลัสเตอร์แบบต่างชนิดโดยใช้หลักการควบคุมการแออัด จุดประสงค์เพื่อ ทดสอบความถูกต้องของการประมวลผลเมื่อมีการเพิ่มเติมในส่วนของอัลกอริทึมที่ได้นำเสนองานวิจัยและเปรียบเทียบความสามารถโดยรวมของระบบว่าได้พัฒนาขึ้นจากการแจกงานในสภาวะปกติหรือไม่ อีกทั้งรวมไปถึงการตรวจทานข้อผิดพลาดและข้อจำกัดต่างๆของระบบ โดยส่วนที่จะทำการทดลองนั้นจะออกเป็น 3 ส่วน เพื่อตอบคำถามด้านล่างต่อไปนี้

- I/O Rate และ Processing Rate ที่ได้จากการใช้หลักการควบคุมการแออัด (Congestion Control) ได้ผลถูกต้องทั้งในขณะเริ่มต้นคลัสเตอร์ (Cluster Startup) และ เมื่อมีการปรับปรุงค่าระหว่าง JobTracker และ TaskTracker
- ความสามารถของระบบเมื่อทำการปรับปรุงการเพิ่มอัลกอริทึมที่มีในงานวิจัย เมื่อเทียบกับการแจกงานในภาวะปกตินั้นมีการพัฒนาเพิ่มขึ้นมากหรือน้อยเพียงใด

5.1 ลักษณะของเครื่องที่ใช้ในการทดลอง (Experiment Environment)

ทางผู้วิจัยได้จัดทำการทดลองขึ้นโดยเครื่องที่เรามีอยู่เป็นลักษณะของเซิร์ฟเวอร์ 1 เครื่อง ทำการแบ่งภายในโดยแยกออกจากกันอย่างอิสระทั้งหมด 6 เครื่อง โดยข้อมูลทั้งหมดในแต่ละเครื่องนั้นข้อมูลจะเป็นไปตามตารางที่ 5-1

ตารางที่ 5-1 แสดงข้อมูลของเครื่องเซิร์ฟเวอร์ที่ใช้ในการทดลองงานวิจัย

Machine Name	Core No.	Memory	OS	Java JDK
NameNode	1 Core	2 GB	Ubuntu 12.04	1.7.0_51
d2c-1	2 Core	4 GB	Ubuntu 12.04	1.7.0_51
d2c-2	2 Core	2 GB	Ubuntu 12.04	1.7.0_51
d1c-1	1 Core	6 GB	Ubuntu 12.04	1.7.0_51
d1c-2	1 Core	2 GB	Ubuntu 12.04	1.7.0_51
d1c-3	1 Core	4 GB	Ubuntu 12.04	1.7.0_51

งานวิจัยได้ใช้ฮาดูปเวอร์ชั่น 1.2.1 โดยนำมาจาก Apache Hadoop SVN ซึ่งเป็นเวอร์ชั่นล่างสุดที่สมบูรณ์ โดยเราได้ทำการปรับปรุงอัลกอริทึมในการแจกงานที่ได้กล่าวถึงไว้ในบทที่ 4 นั้นลงไปในเวอร์ชั่นนี้ และ ในการทดลองจะมีตัวชี้วัด (Benchmark) ที่ใช้ในการทดลองนั้นคือ การประมวลผลแบบนับคำ (Word Count) โดยลักษณะงานเป็นแบบงานที่ใช้เขียนอ่านข้อมูลเป็นหลัก (I/O Bound) และ การประมวลผลแบบการเรียง (Sorting) งานที่ใช้การประมวลผลเป็นหลัก (Processing Bound) ซึ่งตัวชี้วัดเหล่านี้ถือว่าเป็นตัวชี้วัดทั่วไปซึ่งมีผู้วิจัยก่อนหน้ามากมายได้นำมาใช้ในเป็น Benchmark ดังนั้นจึงเป็นตัวชี้วัดที่สามารถยอมรับได้ จึงนำมาใช้เป็นตัวที่ทำการทดลอง

5.2 การแบ่งงานที่เกิดขึ้นเมื่อมีข้อมูลเพื่อนำไปใช้ในการแจกงาน

การทดสอบในส่วนการแบ่งงานที่เกิดขึ้นเมื่อมีข้อมูลเพื่อนำไปใช้ในการแจกงาน นั้นเราได้ทำการประมวลผลงานในลักษณะการประมวลผลแบบนับคำ (Word Count) ขนาด 2 GB และการประมวลผลแบบการเรียง (Sorting) ขนาด 2 GB เช่นเดียวกัน โดยผลที่ได้นั้นเป็นไปตามตารางที่ 5-2 และ ตารางที่ 5-3

ตารางที่ 5-2 แสดงข้อมูลของขนาดที่แบ่งงานเมื่อทำการประมวลผลแบบนับคำ (Word Count)

Machine Name	Task Allocation Length(MB)	Total Task (MB)
namenode	49.896089, 49.896089, 49.896089	149.688267
d1c-3	94.746774, 94.746774, 94.746774	248.240322
d1c-2	87.721575, 87.721575, 87.721575	263.164725
d1c-1	88.120293, 88.120293, 88.120293	264.360879
d2c-2	169.097527, 169.097527, 169.097527	507.292581
d2c-1	177.084412, 177.084412, 177.084412	531.253236

ตารางที่ 5-3 แสดงข้อมูลของขนาดที่แบ่งงานเมื่อทำการประมวลผลแบบการเรียง (Sorting)

Machine Name	Task Allocation Length(MB)	Total Task (MB)
namenode	74.109782, 74.109782, 74.109782	222.3293
d1c-3	94.686840, 94.686840, 94.686840	284.0605
d1c-2	107.722467, 107.722467, 107.722467	323.1674
d1c-1	89.206924, 89.206924, 89.206924	267.6208

d2c-2	141.173249, 141.173249, 141.173249	423.5197
d2c-1	159.767378, 159.767378, 159.767378	479.3021

ตารางที่ 5-4 แสดงค่าจากไฟล์ monitor.json ของ Processing Rate ในรูปแบบของตารางโดยหาค่าเฉลี่ย

$PR_{1...NM}$	Processing Rate (MB/s)					
	NameNode	d1c-3	d1c-2	d1c-1	d2c-2	d2c-1
Average	0.61568	0.7	0.848	0.657	1.141	1.264

ตารางที่ 5-5 แสดงค่าจากไฟล์ monitor.json ของ I/O Utilization ในรูปแบบของตารางโดยหาค่าเฉลี่ย

$IOR_{1...NM}$	I/O Utilization (MB/s)					
	NameNode	d1c-3	d1c-2	d1c-1	d2c-2	d2c-1
Average	3.48	6.04	5.58	5.82	13.9	16.08

จะเห็นว่าเครื่อง d2c-2 และ d2c-1 จะได้งานไปมากที่สุดเนื่องจากมีทั้ง I/O และการประมวลผลที่ดีที่สุดในคลัสเตอร์ ทางส่วนแจกงาน (Scheduler) จึงให้งานไปที่เครื่องทั้งสองนี้มากกว่าเครื่องอื่น ส่วนเครื่องของ d1c-2 นั้นในส่วนของงานการประมวลผลแบบนับคำ (Word Count) ซึ่งงานเป็นลักษณะของงานแบบงานที่ใช้เขียนอ่านข้อมูลเป็นหลัก (I/O Bound) จะได้รับงานน้อยกว่า d1c-3 เหตุเนื่องจากค่าของความสามารถในการใช้ I/O ภายนอกได้คะแนนน้อยกว่า d1c-3 แต่ทว่าหากเป็นประเภทของงานมาเป็นงานที่ใช้การประมวลผลเป็นหลัก (Processing Bound) d1c-3 มีความสามารถในการประมวลผลที่เยอะกว่าทำให้ได้งานไปมากกว่า d1c-2 ในทางกลับกัน ส่วน NameNode นั้นถือว่าได้งานน้อยที่สุด เนื่องจากว่าเครื่อง NameNode ในคลัสเตอร์ต้องรับผิดชอบงาน JobTracker และ TaskTracker ในเวลาเดียวกันจึงทำให้ความสามารถในการประมวลผลนั้นเมื่อเปรียบเทียบกับเครื่องอื่นแล้วทำให้ประสิทธิภาพด้อยกว่า อีกทั้งเครื่อง NameNode เป็นเครื่องที่มีศักยภาพน้อยที่สุดในคลัสเตอร์ในแง่ของฮาร์ดแวร์

หากเราสรุปจากความสอดคล้องกันระหว่างการแบ่งงานของส่วนแจกงานซึ่งสามารถอ้างอิงได้จากตารางที่ 5-2 และ ตารางที่ 5-3 เทียบกับคะแนนที่มีอยู่ในไฟล์ monitor.json ซึ่งนำมาเปลี่ยนฟอร์แมตให้อยู่ในรูปแบบตารางที่ 5-4 และ ตารางที่ 5-5 และ ข้อมูลของเครื่องในคลัสเตอร์นั้นไปในทิศทางเดียวกันจึงสามารถสรุปได้ว่า ประสิทธิภาพของส่วนแจกงานนั้นทำได้อย่างที่

เป้าหมายไว้ของผู้วิจัยโดยแนวโน้มการแจกงานนั้นเป็นไปในทิศทางเดียวกันกับคุณสมบัติของเครื่องที่อยู่ในคลัสเตอร์ ผู้วิจัยได้แสดงข้อมูลทั้งหมดที่นำมาคิดค่าเฉลี่ยของตารางที่ 5-4 และ 5-5 ในภาคผนวก ก

5.3 การประเมินความสามารถของอัลกอริทึม (Algorithm Performance Evaluation)

ในการทดลองจะมีตัวชี้วัด (Benchmark) ที่ใช้ในการทดลองนั้นคือ การประมวลผลแบบนับคำ (Word Count) โดยลักษณะงานเป็นแบบงานที่ใช้เขียนอ่านข้อมูลเป็นหลัก (I/O Bound) และ การประมวลผลแบบการเรียง (Sorting) งานที่ใช้การประมวลผลเป็นหลัก (Processing Bound) โดยการสร้างงานนั้นจะใช้โปรแกรม Generator คำที่ผู้วิจัยได้เขียนขึ้นเอง ใช้อัลกอริทึมการสร้างคำแบบสุ่ม (Random) จากไฟล์ตัวอักษร (Text File) โดย Generate ในลักษณะ 4 ไฟล์ ไฟล์ละ 2 GB และ การสร้างคำแบบสุ่มเพื่อทำการ Sorting Key และ Value จะใช้โปรแกรม Generator เช่นเดียวกัน โดย Source Code และ โปรแกรมทั้งหมดจะอธิบายอย่างละเอียดอีกครั้งในภาคผนวก ดังนั้นการประมวลผลจะประมวลผลขนาดไฟล์ 2GB, 4GB, 6GB และ 8GB อย่างละ 5 ครั้งแล้วนำมาหาค่าเฉลี่ย โดยผลที่ได้ออกมานั้นอยู่ในตารางที่ 5-5 สำหรับการประมวลผลแบบนับคำ (Word Count) และ ตารางที่ 5-6 สำหรับการประมวลผลแบบการเรียง (Sorting)

ตารางที่ 5-6 แสดงเวลาที่ใช้ในการประมวลผลแบบนับคำ (Word Count)

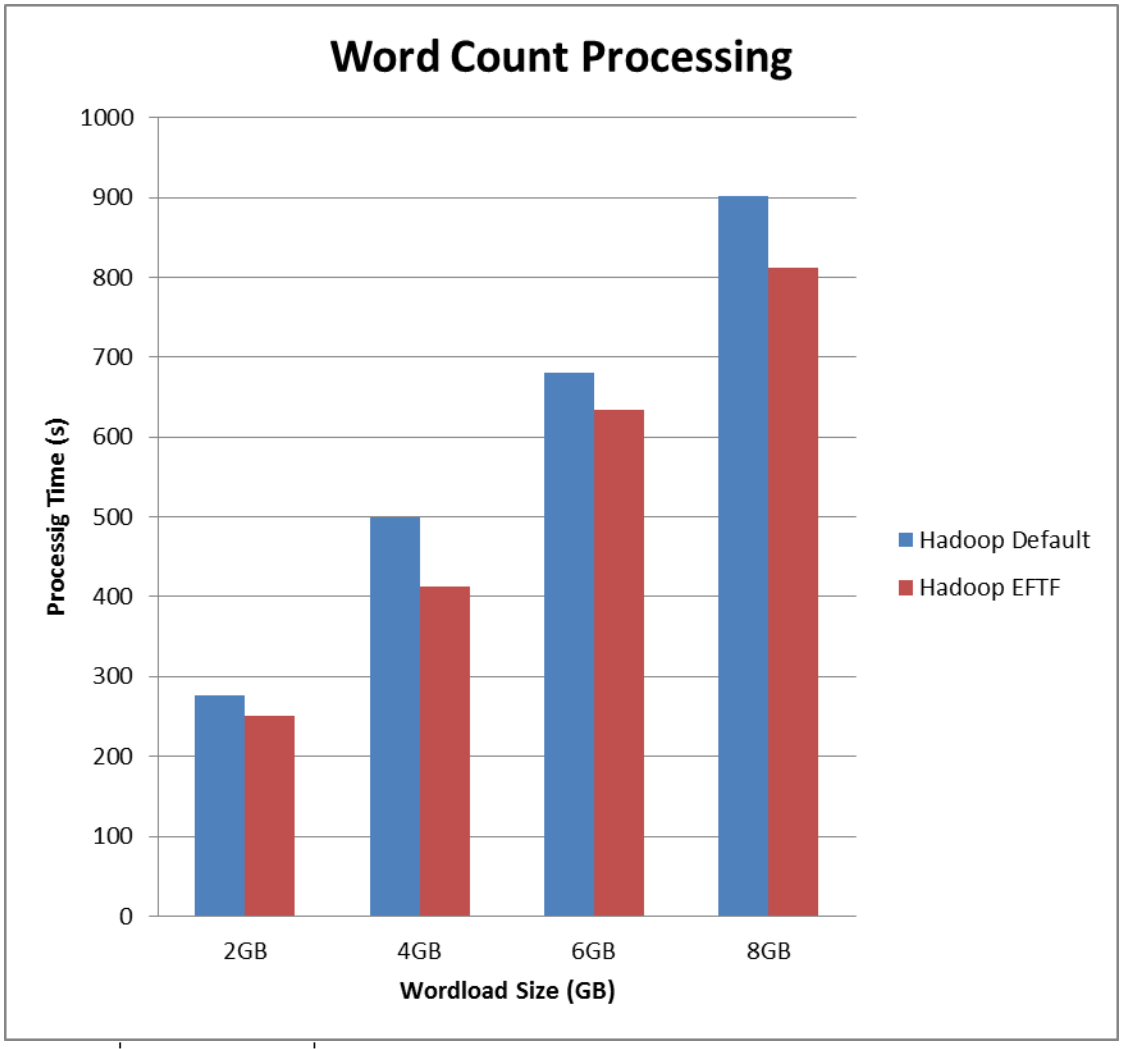
Data Size	Average Time		Delta Time	Percentage
	Hadoop Default (s)	Hadoop EFTF (s)		
2GB	277.2	250.8	26.4	9.523
4GB	498.8	412.2	86.6	17.36
6GB	679.8	633.6	46.2	6.796
8GB	901.8	811.4	90.4	10.02

ตารางที่ 5-7 แสดงเวลาที่ใช้ในการประมวลผลแบบเรียง (Sorting)

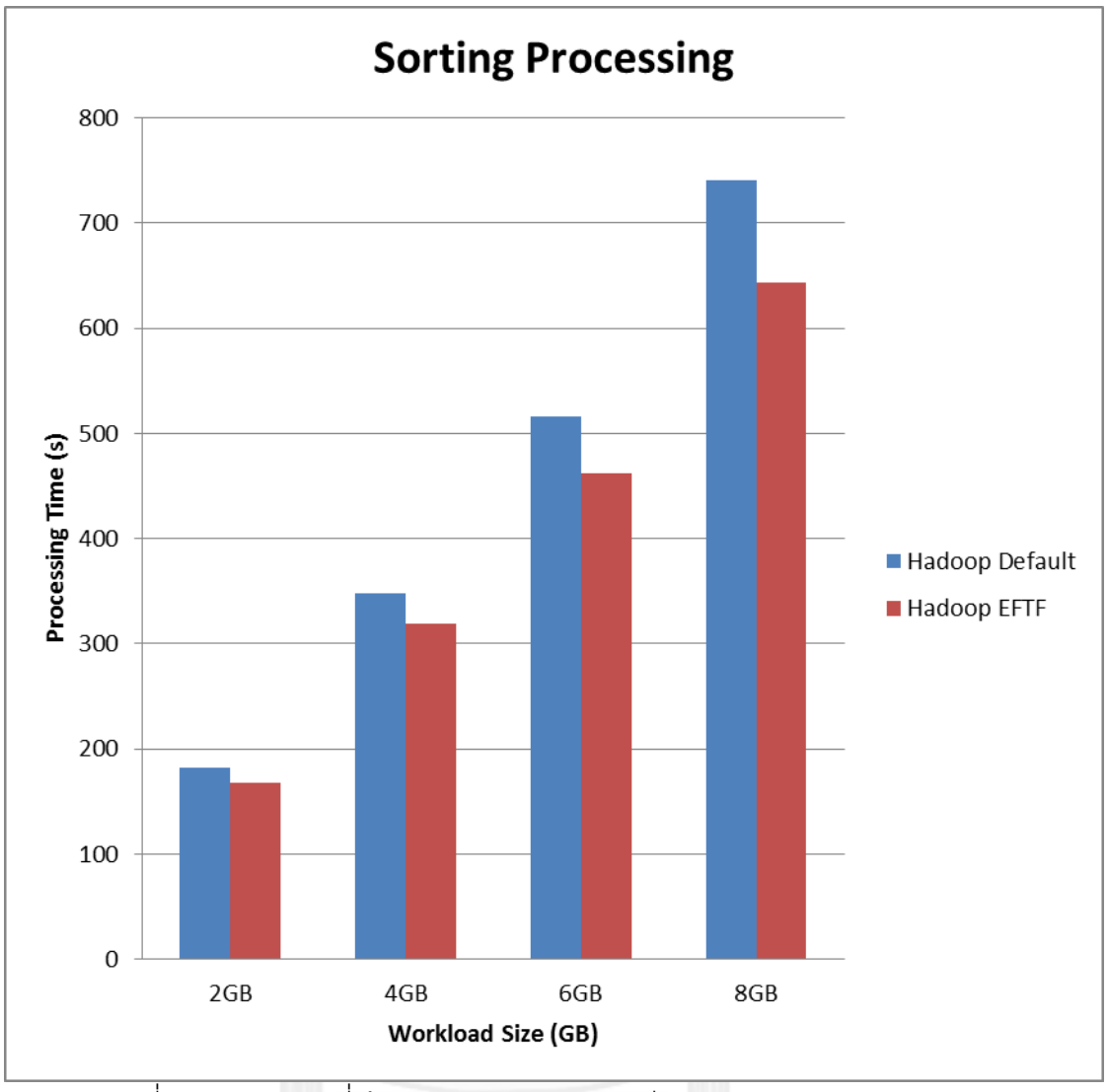
Data Size	Average Time		Delta Time	Percentage
	Hadoop Default (s)	Hadoop EFTF (s)		
2GB	181.6	167.6	14	7.709
4GB	347.8	319.2	28.6	8.223
6GB	515.8	462.2	53.6	10.39

8GB	741.2	644	97.2	13.11

จากตารางที่ 5-6 และ ตารางที่ 5-7 สามารถแสดงในรูปแบบของกราฟได้ในรูปที่ 5-1 และรูปที่ 5-2



รูปที่ 5-1 แสดงเวลาที่ใช้ในการประมวลผลแบบนับคำ (Word Count) ในรูปแบบกราฟ



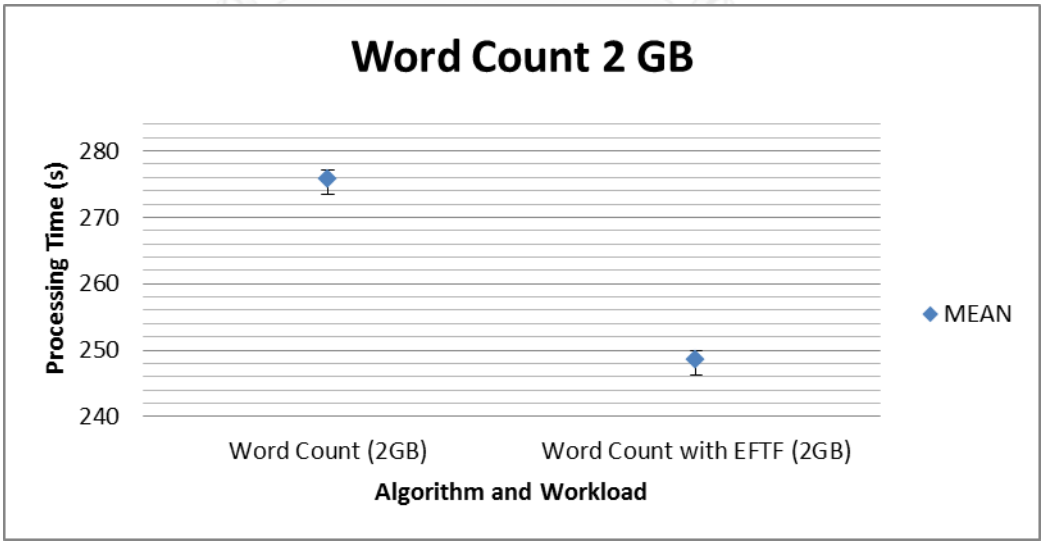
รูปที่ 5-2 แสดงเวลาที่ใช้ในการประมวลผลแบบเรียง (Sorting) ในรูปแบบกราฟ

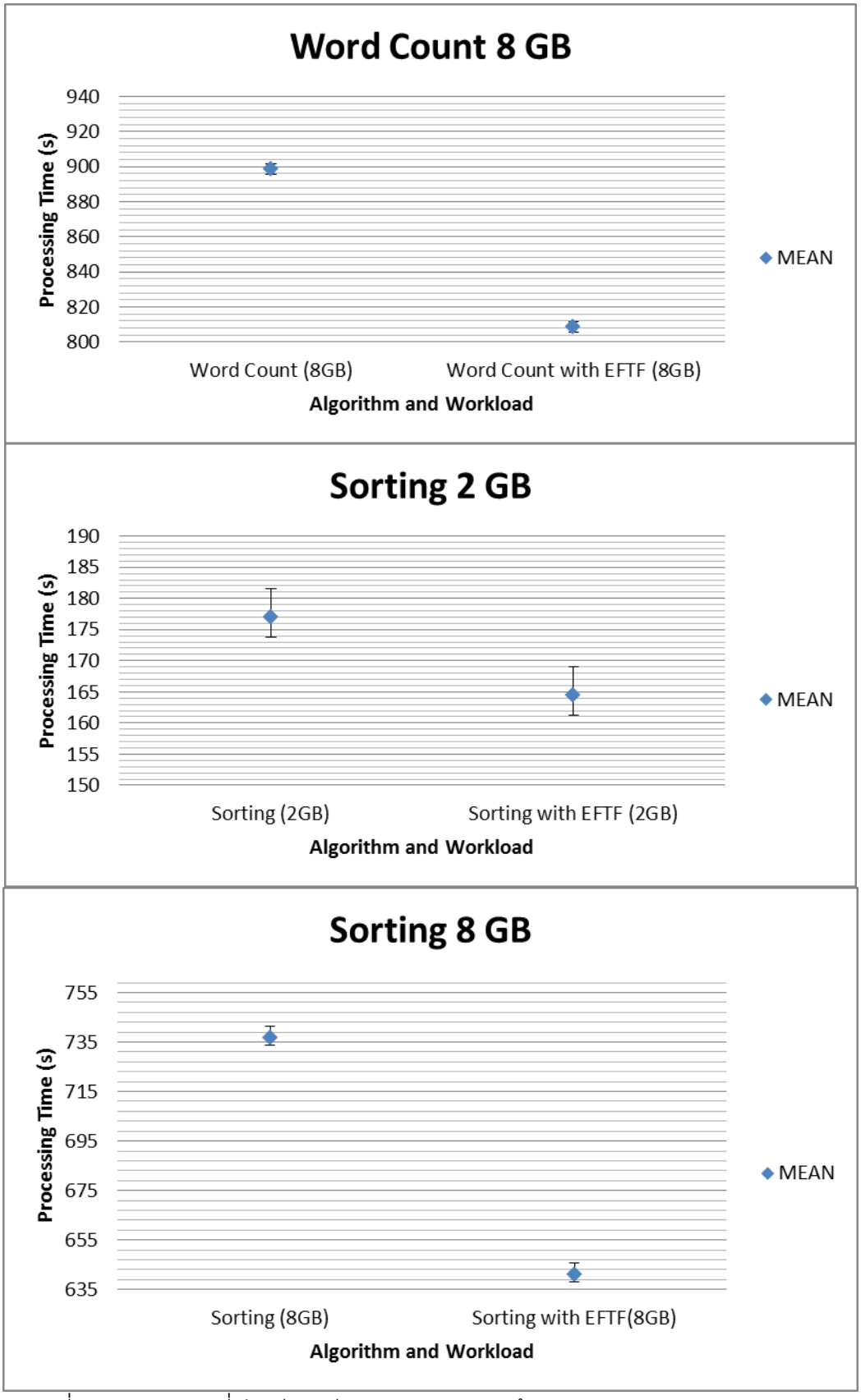
จากตารางที่ 5-6 และ ตารางที่ 5-7 สามารถสรุปได้ว่าด้วยอัลกอริทึม EFTF ที่ได้ทำการปรับปรุงเข้าไปในส่วนแจกงานในสภาวะปกตินั้น (Default Scheduler) มีผลให้เกิดการประมวลผลที่เร็วขึ้นประมาณ 10 – 11% ในการประมวลผลแบบนับคำ (Word Count) และ การประมวลผลแบบเรียง (Sorting) ซึ่งผู้วิจัยจะอธิบายอย่างละเอียดว่าเพราะเหตุใดผลถึงออกมาจากการประมวลผลถึงทำให้เร็วขึ้นโดยจะขอใช้ข้อมูลอย่างละเอียดเพื่อการอธิบายโดยใช้ การประมวลผลแบบนับคำขนาด 8 GB และ 2 GB และ การประมวลผลแบบเรียงขนาด 8 GB และ 2 GB มาทำการอธิบายผลการทดลอง

ตารางที่ 5-8 แสดงค่าของ Confident Interval ในส่วนของเวลาในการประมวลผลของ Benchmark ขนาด 2 GB และ 8GB

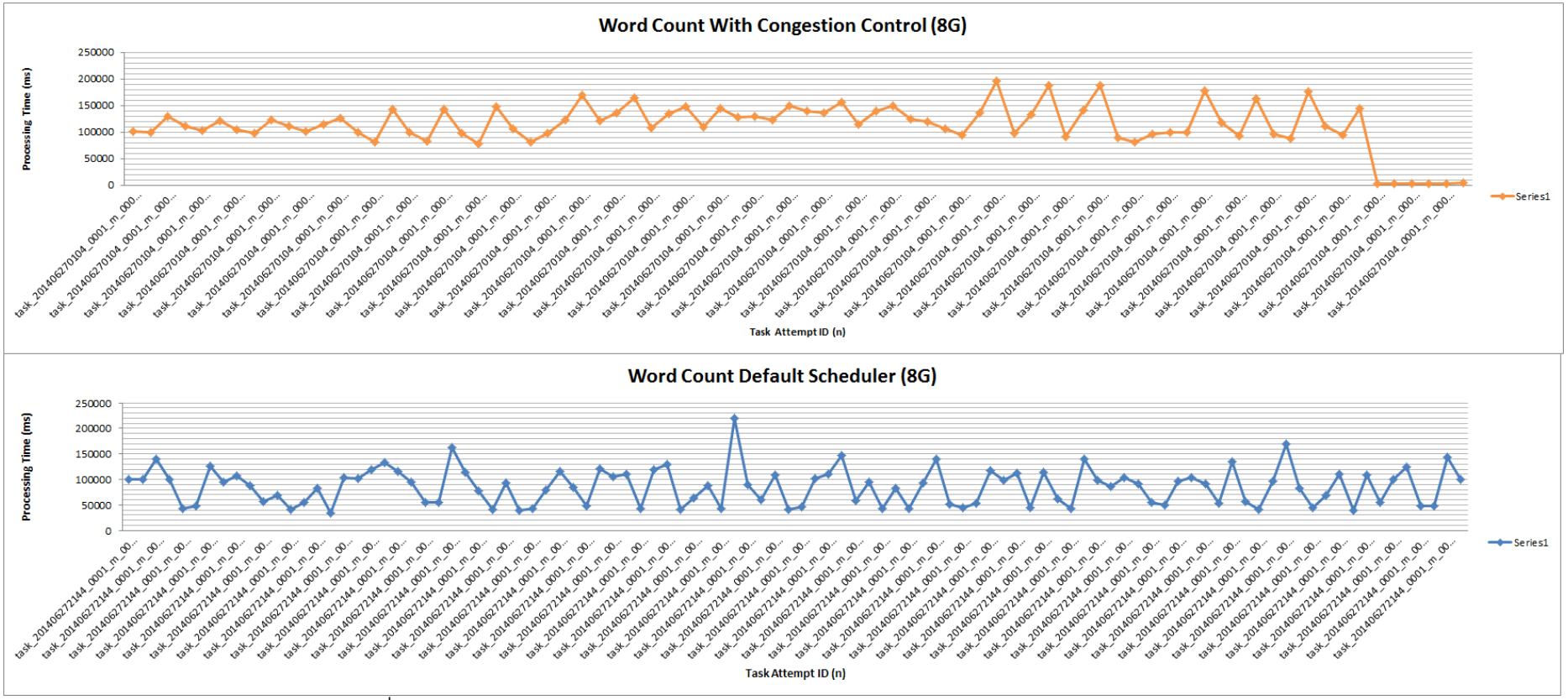
ชนิดของ Benchmark	Mean ($\bar{x}$) (s)	Standard Derivation (SD) (s)	Confident Interval at 90% (min, max) (s) $(\frac{\bar{x}-(z\times SD)}{\sqrt{n}}, \frac{\bar{x}+(z\times SD)}{\sqrt{n}})$
Word Count (2GB)	277.2	3.83	(275.78, 278.61)
Word Count with EFTF (2GB)	250.8	6.06	(248.55, 253.04)
Word Count (8GB)	901.8	8.01	(898.83, 904.76)
Word Count with EFTF (8GB)	811.4	7.89	(808.47, 814.32)
Sorting (2GB)	181.6	12.42	(176.99, 186.20)
Sorting with EFTF (2GB)	167.6	8.67	(164.38, 170.81)
Sorting (8GB)	741.2	12.27	(736.65, 745.74)
Sorting with EFTF(8GB)	644	8.15	(640.97, 647.02)

จากตารางที่ 5-8 เมื่อนำมาสร้างกราฟในรูปแบบ Approximate Visual Test จะได้ผลการทดลองดังรูปที่ 5-3

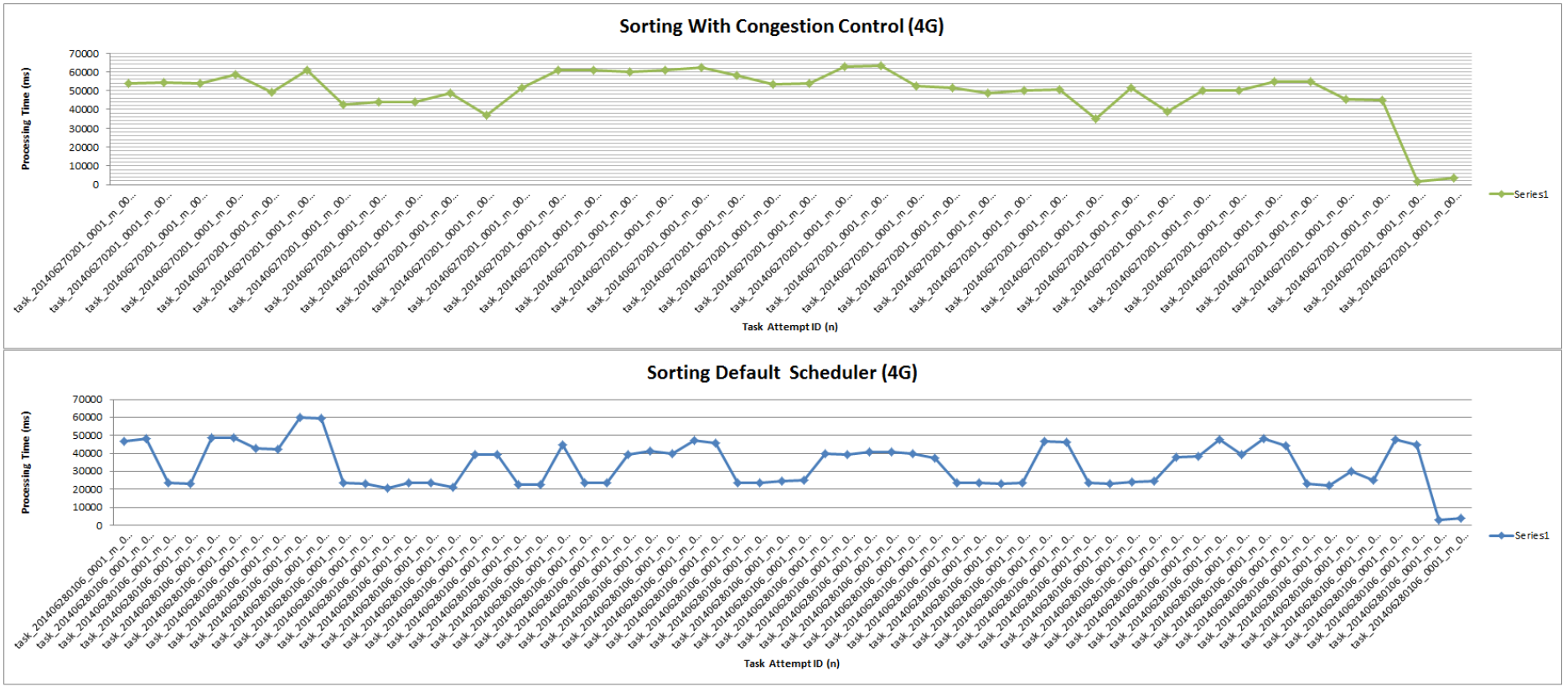




รูปที่ 5-3 แสดงเวลาที่ใช้เปรียบเทียบระหว่างอัลกอริทึม EFTF และ Default Algorithm



รูปที่ 5-4 แสดงกราฟเปรียบเทียบเวลาในการประมวลผลแบบนับคำ (Word Count)



รูปที่ 5-5 แสดงกราฟเปรียบเทียบเวลาในการประมวลผลแบบเรียง (Sorting)

จากตารางที่ 5.8 และรูปที่ 5.3 จะเห็นถึงข้อมูลในแต่ละส่วนว่าใช้เวลาในการประมวลผลนานเท่าไร เมื่อนำมาแสดงในลักษณะของการเปรียบเทียบ Approximate Visual Test โดยการนำ 2 อัลกอริทึม (Default และ EFTF) มาเปรียบเทียบ Confident Interval หากกราฟอยู่ส่วนที่ต่ำกว่าเมื่อทำการเปรียบเทียบ แสดงว่าอัลกอริทึมมีประสิทธิภาพมากกว่า จะเห็นได้ว่ารูปแบบกราฟไม่มีส่วนที่ทับซ้อนกัน (Overlapping) และรูปของ EFTF อยู่ต่ำกว่า Default ซึ่งแสดงให้เห็นว่าอัลกอริทึม EFTF มีประสิทธิภาพการประมวลผลดีขึ้นอย่างเห็นได้ชัด โดยสาเหตุที่ทำให้ส่วนแจกงานที่ใช้อัลกอริทึม EFTF นั้นประสิทธิภาพการประมวลผลดีกว่าส่วนแจกงานแบบปกติสามารถสรุปได้ดังต่อไปนี้

- การที่งานนั้นถูกแบ่งออกเป็นจำนวนมากทำให้การทำงานของ MapReduce นั้นช้าลงเนื่องจากการแจกงานไปหนึ่งครั้งเมื่อทำงานเสร็จสิ้นมีส่วนของ Cleanup Task ซึ่งส่วนนั้นถือว่าเสียเวลาอยู่ที่ประมาณ 3-5 วินาทีต่อ 1 งาน เนื่องจากผู้วิจัยได้คำนึงถึงขนาดในการแบ่งงานทำให้สามารถลดเวลาการ Cleanup งานลงส่งผลให้การทำงานเสร็จสิ้นนั้นทำได้เร็วขึ้น
- ถึงแม้ว่าจากค่าของ Confident Interval ในส่วนของฮาดูปในการแจกงานในสภาวะปกตินั้นจะมีค่า Confident Interval ที่น้อยกว่า แต่ว่าอัลกอริทึมที่นำเสนอในงานวิจัยนั้นสามารถทำงานเสร็จเร็วกว่า เพราะ จำนวนงานที่ทำในการแบ่งแต่ละส่วนนั้นน้อยกว่าเมื่อนำไปคูณกับจำนวนของงานแล้วจะใช้เวลาน้อยกว่า และหากวิเคราะห์จากข้อมูลของส่วนเบี่ยงเบนมาตรฐานจะเห็นว่าส่วนเบี่ยงเบนมาตรฐานมีค่าน้อยทำให้การทำงานเสร็จสิ้นค่อนข้างพร้อมกันเป็นในลักษณะของคลื่น (Wave)
- หากดูจากรูปที่ 5.4 และ รูปที่ 5.5 สืบเนื่องจากส่วนเบี่ยงเบนมาตรฐานที่กล่าวไปในข้างต้นส่งผลให้กราฟนั้นเกิดการเปลี่ยนแปลงแบบกะทันหันบ่อยซึ่งหากแบ่งงานเท่ากันนั้นสามารถทำงานได้ดีในคลัสเตอร์แบบเอกพันธุ์ แต่เกิดปัญหาในคลาสเจอร์แบบต่างชนิดเนื่องจากจะมีหากงานนั้นเข้ามาในลักษณะเป็นลำดับ (Sequential Job) จะเกิดสถานการณ์ที่ว่าเครื่องทั้งคลัสเตอร์เสร็จสิ้นการทำงานทั้งหมดแล้วแต่ว่ายังเหลืออยู่เครื่องหนึ่งที่ทำงานช้ากว่าเครื่องอื่นนั้นยังทำการประมวลผลอยู่ซึ่ง หากวัดประสิทธิภาพของ End to End Performance แล้วจะถือว่าทำได้ไม่ดี

บทที่ 6
สรุปผลงานวิจัยและแนวทางการพัฒนาต่อ

6.1 สรุปผลการวิจัยและแนวทางการพัฒนาต่อ

งานวิจัยได้นำเสนอวิธีการแจกงานของส่วนแจกงาน (Scheduler) ในคลัสเตอร์แบบต่างชนิดโดย เพื่อช่วยให้การแจกงานในการประมวลผลแบบกระจาย (Distributed Computing) นั้นเกิดประโยชน์อย่างมีประสิทธิภาพสูงสุด ซึ่งการประมวลผลแบบข้อมูลขนาดใหญ่ (Big Data Processing) ในลักษณะการประมวลผลแบบ MapReduce นั้นใช้กันอย่างแพร่หลายในองค์กรที่ต้องการการประมวลผลแบบรวดเร็วแต่ข้อมูลนั้นมีขนาดใหญ่ซึ่ง งานวิจัยนั้นได้นำเสนอการแบ่งงานที่เหมาะสมให้กับเครื่องในคลัสเตอร์ โดยใช้หลักการของการนำข้อมูลประสิทธิภาพของการอ่านเขียนข้อมูลและประสิทธิภาพในการประมวลผลนั้นมาใช้ให้เกิดประโยชน์ ด้วยการแจกเครื่องที่ไม่มี Data Locality อาจมีประสิทธิภาพในการประมวลผลมากกว่าเครื่องที่มี Data Locality เมื่ออยู่ในสภาวะของคลัสเตอร์แบบต่างชนิด

อีกทั้งผู้เขียนได้ทำการทดลองกับเครื่องจริงที่มีอยู่โดยไม่มีการจำลองการทดลอง (Simulation) ซึ่งการทำงานและการปรับปรุงต่างๆนั้นเป็นการลงมือปฏิบัติจริงนำโอเพนซอร์ซมาทำการแก้จริงและได้เสนอแนวทางการทำงานซึ่งบุคคลที่จะนำไปทำต่อสามารถนำไปใช้อ่านเป็นกระบวนการความรู้ถือเป็นเครื่องมือช่วยให้การทำงานนั้นรวดเร็วขึ้น โดยผลการทดลองที่ออกมาถือว่าประสบความสำเร็จในแง่มุมของเวลาในการประมวลผลนั้นถือว่าทำได้ดีขึ้น 10 – 11% เมื่อเทียบกับฮาดูปที่ทำอยู่ในปัจจุบัน ซึ่งผู้วิจัยมีความภูมิใจที่ได้สร้างสิ่งนี้และหวังว่าจะเป็นคุณประโยชน์แก่ผู้ที่ได้อ่านแล้วนำไปต่อยอดเพื่อปรับปรุงให้ดีขึ้นยิ่งๆขึ้นไปและแนวทางการพัฒนาต่อ การทำการประมวลผลงานสำรอง (Speculative Task) ณ ตอนนี้จะทำงานใหม่ทั้งงาน หากทำใหม่แค่บางส่วนจะเป็นการประหยัดส่วนของทรัพยากรเพิ่มมากขึ้นรวมไปถึงระหว่างกระบวนการ Map ไป Reduce Phase นั้นจะมีกระบวนการทำ Partitioning เพื่อแจกงานให้กับเครื่องทำ Reduce ณ ปัจจุบัน ฮาดูปทำงานในลักษณะเดียวคือแบ่ง Partitioning ตาม Data Locality แต่บางครั้งเครื่อง Reduce บางเครื่องไม่ได้งานเพราะว่าไม่มี Data อยู่ที่เครื่องของตัวเอง รวมไปถึงในส่วนของการเรียนรู้ลักษณะของงานที่เข้ามาประมวลผลโดยอัตโนมัติของแต่ละเครื่องที่ประมวลผลนั้นสามารถทำได้

รายการอ้างอิง

1. Lu, P., Wu, S., Shou, L. and Tan, K.-L, *An efficient and compact indexing scheme for large-scale data store.* International Conference on Data Engineering, 2013.
2. Yang, W.a.D., Y., *High-performance Distributed Indexing and Retrieval for Large Volume Traffic Log datasets on the cloud.* Proceedings of the 2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics, 2013.
3. L. Barroso, J.D., and U. Holzle, *Web search for a planet: The Google cluster architecture.* IEEE Micro, 2003. **23**.
4. Premchaiswadi, W., Tungkatsathan, A., Intarasema, S., Premchaiswadi, N. , *Improving performance of content-based image retrieval schemes using Hadoop MapReduce.* Proceedings of the 2013 International Conference on High Performance Computing and Simulation, 2013.
5. S. Ghemawat, H.G., and S.-T. Leung, *The Google File System.* Proceedings of the 9th ACM Symposium on Operating Systems Principles, 2003.
6. Chang, J.D., S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, *Bigtable: A distributed storage system for structured data.* Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, 2006.
7. Dhruba Borthakur, K.M., Karthik Ranganathan Samuel Rash, Joydeep Sen Sarma, Jonathan Gray, Nicolas Spiegelberg, Hairong Kuang, Dmytro Molkov, Aravind Menon, Rodrigo Schmidt and Amitanand Aiyer, *Apache Hadoop Goes Realtime at Facebook.* Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, 2011.
8. Ghemawat, J.D.a.S., *MapReduce: Simplified Data Processing on Large Clusters.* In Communications of the ACM, 2008. **51**(1).
9. Community, H., *An Introduction of HDFS Hadoop.*
10. Zaharia, M., *Job Scheduling with the Fair and Capacity Schedulers.* 2009.
11. M. Zaharia, A.K., A. D. Joseph, R. Katz, and I. Stoica, *Improving mapreduce performance in heterogeneous environments.* 8th Usenix Symposium on Operating Systems Design and Implementation, 2008. **8**(1).

12. Jorda Polo, C.C., David Carrera, Yolanda Becerra, Ian Whalley, Malgorzata Steinder, Jordi Torres and Eduard Ayguade, *Resource-aware Adaptive Scheduling for MapReduce Clusters.* Barcelona Supercomputing Center (BSC) and Technical University of Catalonia (UPC.

13. Jiong Xie, S.Y., Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, and Xiao Qin, *Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters.* 19th Int'l Heterogeneity in Computing Workshop, Atlanta April 2010.

14. Natthakrit Sanguandikul, N.N., *Agentless robust load sharing strategy for utilising heterogeneous resources over wide area network.* Maejo International Journal of Science and Technology, 2011.

15. Source, J.C.O., *JSON Community Open Source.*

16. source, G.C.o., *Google Community open source.*

ภาคผนวก

ภาคผนวก ก.
ขั้นตอนการทำงานอย่างละเอียดและข้อมูลทั้งหมดที่ใช้ในการทดลอง

ในหัวข้อนี้จะกล่างถึงการทำงานโดยละเอียดของอัลกอริทึม EFTF ที่ผู้วิจัยได้กล่าวไว้ในการออกแบบในหัวข้อที่ 3 รวมไปถึงข้อมูลทั้งหมดที่ใช้ในการทดลองเพื่อให้ผู้ที่สนใจสามารถนำไปอ้างอิงได้

1. ขั้นตอนการทำงานอย่างละเอียดของอัลกอริทึม

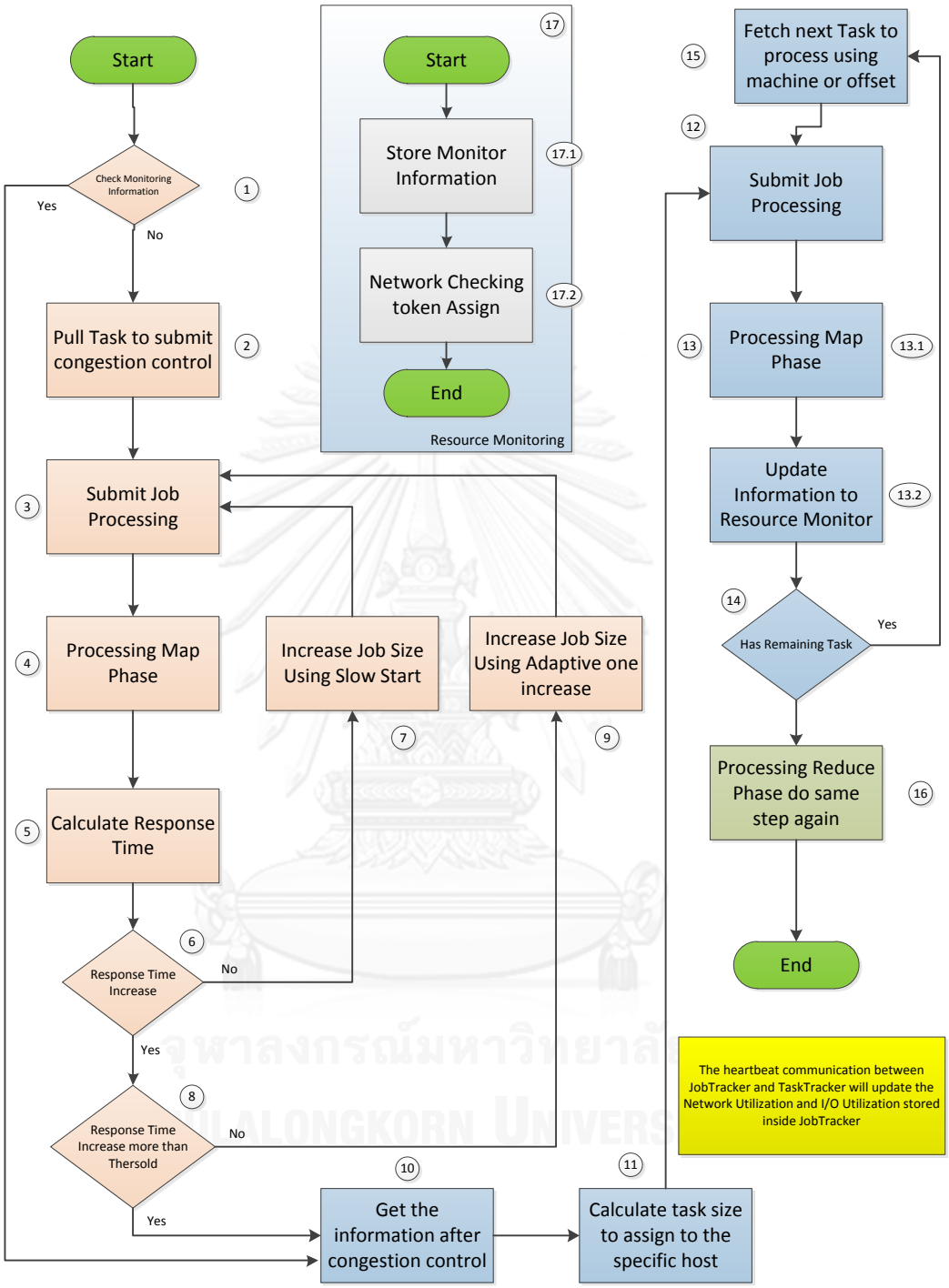
ผู้วิจัยได้นำเสนอหลักการทำงานแบบภาพรวมของอัลกอริทึม EFTF ไว้ในหัวข้อที่ 3.1 นั้นสามารถศึกษารายละเอียดการทำงานโดยละเอียดได้ดังต่อไปนี้

จากรูปที่ 6-1 สามารถอธิบายขั้นตอนการทำงานของงานวิจัยได้ดังต่อไปนี้

1. ทำการเช็คข้อมูลของคลัสเตอร์ว่ามีข้อมูลประสิทธิภาพของเครื่องข่ายและ I/O ภายนอกรวมไปถึงประสิทธิภาพในการประมวลและหรือไม่ ถ้าหากว่ามีให้ทำการข้ามไปในขั้นตอนที่ 10 ทันที ถ้าไม่มีให้เริ่มขั้นตอนที่ 2
2. นำข้อมูลมาทำการหาค่าของประสิทธิภาพของเครื่องข่ายและ I/O ภายนอกรวมไปถึงประสิทธิภาพในการประมวลใช้หลักการการควบคุมการแออัดของข้อมูล
3. ร้องขอการประมวลผล เพื่อเริ่มการทำ Map Program Processing
4. เริ่มการประมวลผล
5. ทำการคำนวณหาเวลาในการประมวลผล (Response Time) ของแต่ละเครื่อง ทำการตัดสินใจว่าจะเพิ่มขนาดของงานหรือว่าเพียงพอแล้วสำหรับเครื่องๆนั้น
6. ถ้าหากว่ายังไม่มีการเพิ่มของเวลาในการประมวลผลหรือเพิ่มเพียงเล็กน้อยนั้นจะ
7. ทำการเพิ่มขนาดแบบทวีคูณ (Exponential) ผู้วิจัยได้เรียกว่า Slow start Phase แล้วทำตามขั้นตอนที่ 3 ใหม่อีกครั้ง
8. เมื่อการเพิ่มของเวลาในการประมวลผลนั้นเกิดขึ้นอย่างเห็นได้ชัด
9. ทำการเริ่มเปลี่ยนลักษณะการเพิ่มขนาดของข้อมูลจากทวีคูณเป็นการเพิ่มที่ละหนึ่ง (Adaptive Repeat) หากเพียงพอแล้วเราก็จะสามารถรู้ได้ถึงงานขนาดเท่าไรที่เหมาะสมกับเครื่องๆนั้นในคลัสเตอร์ จากค่าของตัวแปรดังกล่าวจะนำไปใช้ในการ

คำนวณเพื่อหาขนาดของงานที่จะทำการแจกไปให้แก่เครื่องต่างๆเมื่อทำการร้องขอการประมวลผลจริง

10. นำข้อมูลประสิทธิภาพของเครื่องข่ายและ I/O ภายนอกรวมไปถึงประสิทธิภาพในการประมวลที่ได้จากการลองประมวลผลโดยใช้หลักการของการควบคุมการแออัดของข้อมูล มาสร้างเป็นลักษณะของข้อมูลที่เป็นระเบียบ (Information Format) โดยจะกล่าวถึงโดยละเอียดในหัวข้อที่ 3.2
11. นำข้อมูลที่ได้จากขั้นตอนที่ 10 มาทำการคำนวณหาขนาดของข้อมูลที่ทำการแจกจ่ายไปให้กับเครื่องต่างๆภายในคลัสเตอร์เมื่อเริ่มการประมวลผล โดยวิธีการแจกจ่ายงานนั้นจะกล่าวถึงอย่างละเอียดอีกครั้งในหัวข้อที่ 3.3
12. ร้องขอการประมวลผล เพื่อเริ่มการทำ Map Program Processing
13. เริ่มการประมวลผล Map Program Processing
    a. ระหว่างการประมวลผลจะมีการรายงานสถานะของ TaskTracker ไปยัง JobTracker ที่ในทางปฏิบัติถูกเรียกว่า การส่ง Heartbeat โดยจะมีการเพิ่มในส่วนของข้อมูลประสิทธิภาพของเครื่องข่ายและ I/O ภายนอกรวมไปถึงประสิทธิภาพในการประมวล เพื่อให้ได้มาซึ่งข้อมูลที่ทันสมัยตลอดเวลา (Update Monitoring Information)
14. JobTracker ทำการพิจารณาว่ายังคงมีงานที่ต้องการการประมวลผลอีกหรือไม่ ถ้ายังมีอยู่จะจ่ายงานต่อไปให้กับเครื่องที่กำลังร้องของงาน การแจกงานเมื่อมีการร้องขอใหม่ได้มีการปรับปรุงจากการแจกงานในสภาวะปกตินั้นคือมีการเพิ่มในส่วนของการแจกงานที่แน่นอนโดย ส่วนแจกงานจะรู้เมื่อทำการแบ่งงานเสร็จว่าจะต้องแจกงานนี้ไปที่เครื่องๆใดในคลัสเตอร์เพื่อให้เกิดประสิทธิภาพสูงสุดในการประมวลผล
15. เมื่อได้รับงานจะกลับไปเริ่มทำขั้นตอนที่ 12
16. การทำ Map Program Processing จะทำการประมวลผล Reduce Program Processing ต่อไป โดยใช้รูปแบบเดียวกับการประมวลผลใน Map Phase



รูปที่ 6-1 ผังงานขั้นตอนการทำงานปรับปรุงอัลกอริทึมของงานวิจัย

2. ข้อมูลต่างๆจากการทดลอง

ข้อมูลจากการทดลองที่นำเสนอในบทที่ 5 จะกล่าวถึงในส่วนข้อมูลขากการทดลองดังต่อไปนี้

ตารางที่ 7-1 แสดงค่าของไฟล์ monitor.json ของ Progress Rate ในรูปแบบของตารางในขณะทาการทดลอง (รูปแบบเต็ม)

	Progress Rate (MB/s)					
	NameNode	d1c-3	d1c-2	d1c-1	d2c-2	d2c-1
	0.671	0.703	0.806	0.874	1.019	1.488
	0.409	0.65	0.759	0.619	1.634	1.454
	0.691	0.825	0.758	0.644	1.727	1.608
	0.639	0.717	0.725	0.642	1.022	1.565
	0	0.652	0.605	0.869	1.63	0
	0	0.717	0.727	0	1.61	1.617
	0	0.928	0.597	0.929	1.07	1.511
	0	0.686	0.781	0.649	1.615	1.574
	0.404	0.807	0.745	0.669	1.295	1.491
	0.568	0.721	0.648	0.645	0.91	1.451
	1.058	0.739	0.771	0.941	1.607	1.321
	0	0.635	0.72	0.674	1.007	1.443
	0	0.89	0.713	0.626	1.63	1.31
	0	0.607	0	0.909	1.64	1.595
	1.273	0.789	1.64	1.193	1.626	1.507
	1.266	0.914	1.626	0	1.111	1.583
	1.315	0.86	0	0	1.703	1.385
	0	0.736	0	0.848	1.73	1.541
	0	0.672	0	0.679	1.025	1.382
	0	0.742	1.663	0.591	1.69	1.583

	1.406	0.866	1.607	0.677	1.113	1.489
	1.388	0.715	1.75	0.832	1.698	1.623
	1.099	0.782	1.753	0.627	1.763	1.52
	1.109	0.841	1.766	0.725	1.093	1.415
	1.344	0.674	1.766	0.831	1.668	1.296
	1.348	0.727	0	0.583	1.592	1.422
	0	0.912	0	0.643	1.707	1.332
	0	0.843	0	1.067	0	1.579
	0	1.585	0.955	0	0	1.431
	0.599	1.563	0.761	0	0	1.339
	0.593	0	0.839	1.675	0	1.573
	0.553	0	0.595	1.649	0	0
	0.365	0	0.744	0	1.333	0
	0.697	0.71	0.601	0	1.066	0
	0.672	0.67	0.912	0	1.794	0
	0.398	0.674	0.744	0.882	1.034	1.507
	0.716	0.625	0.66	0.6	1.489	1.483
	0.566	0.687	0.883	0.896	1.815	1.498
	0.359	0.596	0.732	0.45	1.046	1.283
	0.682	0.8	0.803	0.664	1.5	1.614
	0.478	0.659	1.693	0.951	1.847	1.37
	0	0.611	1.676	0.469	0.978	1.579
	0	0.73	1.662	0.621	1.358	1.593
	1.381	0.787	1.639	0.46	0	1.267
	1.351	0	1.806	0.941	0	1.478
	1.203	1.479	1.755	0.926	0	1.572
	1.22	1.474	0	1.348	0	1.404
	1.508	0	0	1.35	1.834	1.123

	1.455	0	0	0	1.007	0
	0	0	0	0	0	0
Average	0.61568	0.7	0.848	0.657	1.141	1.264

ตารางที่ 7-2 แสดงค่าของไฟล์ monitor.json ของ I/O Rate ในรูปแบบของตารางในขณะทำการทดลอง

	I/O Rate					
	NameNode	d1c-3	d1c-2	d1c-1	d2c-2	d2c-1
	4	7	5	5	16	20
	4	5	3	5	16	17
	4	7	6	5	16	19
	2	8	6	6	17	13
	3	5	5	7	16	17
	3	4	2	4	16	11
	2	4	4	7	16	19
	3	5	2	5	17	20
	2	5	6	6	15	19
	3	6	5	6	16	15
	3	7	5	5	16	19
	0	7	6	11	16	19
	0	6	5	5	16	19
	0	5	6	6	16	19
	0	5	6	4	16	19
	2	6	5	6	16	20
	5	6	6	5	9	18
	10	6	5	5	17	15
	10	7	5	7	17	19
	3	7	5	5	17	20
	3	5	6	5	19	19

	2	7	4	4	17	18
	2	5	6	13	17	18
	3	6	6	6	17	10
	4	7	6	6	16	19
	4	7	6	6	13	18
	2	7	7	6	17	18
	2	7	7	7	14	18
	3	7	5	7	15	17
	3	6	5	6	15	17
	17	6	6	6	16	17
	4	7	6	7	15	4
	4	7	7	6	15	16
	3	5	4	5	15	17
	2	6	7	5	14	18
	4	7	6	5	15	6
	4	5	6	6	16	17
	2	6	6	6	15	17
	4	4	7	5	0	4
	3	7	6	6	0	10
	3	7	4	5	0	17
	5	4	7	6	0	6
	3	7	6	5	11	17
	3	5	5	4	9	9
	4	7	7	5	14	16
	4	7	7	5	14	16
	3	5	6	6	13	17
	3	5	6	5	14	17
	4	7	7	6	7	17

	4	6	7	6	15	17
Average	3.48	6.04	5.58	5.82	13.9	16.08

ตารางที่ 7-3 แสดงเวลาที่ใช้ในการประมวลผลแบบนับคำ (Word Count) รูปแบบเต็ม

Algorithm	Benchmark	N	2GB	4GB	6GB	8GB
Hadoop Default	Word Count	1	282	506	674	912
		2	273	503	682	890
		3	277	494	686	902
		4	280	495	676	905
		5	274	496	681	900
	Average		277.2	498.8	679.8	901.8
Hadoop (With Algorithm)	Word Count	1	252	413	620	823
		2	256	412	641	810
		3	244	401	629	806
		4	245	421	638	815
		5	257	414	640	803
	Average		250.8	412.2	633.6	811.4
	Delta		26.4	86.6	46.2	90.4
	Delta (%)		9.523	17.36	6.796	10.02
	AVG Delta (%)		10.92649742			

ตารางที่ 7-4 แสดงเวลาที่ใช้ในการประมวลผลแบบการเรียง (Sorting) รูปแบบเต็ม

Algorithm	Benchmark	N	2GB	4GB	6GB	8GB
Hadoop Default	Sorting	1	194	336	524	740
		2	193	358	510	723
		3	174	355	517	756
		4	165	342	520	748
		5	182	348	508	739
	Average		181.6	347.8	515.8	741.2

Hadoop (With Algorithm)	Sorting	1	177	330	454	653
		2	155	304	470	632
		3	164	316	465	650
		4	174	325	463	643
		5	168	321	459	642
	Average		167.6	319.2	462.2	644
	Delta		14	28.6	53.6	97.2
	Delta (%)		7.709	8.223	10.39	13.11
	AVG Delta (%)		10.39298145			

ตารางที่ 7-5 แสดงเวลาที่ใช้ในการประมวลผลแบบนับคำ (Word Count) ในแต่ละส่วนของงานแบบปรับปรุงอัลกอริทึมโดยผู้วิจัย

Task Attempt ID	T (ms)	Task Attempt ID	T (ms)
task_201406270104_0001_m_000000	101952	task_201406270104_0001_m_000039	139187
task_201406270104_0001_m_000001	100409	task_201406270104_0001_m_000040	136123
task_201406270104_0001_m_000002	129970	task_201406270104_0001_m_000041	156997
task_201406270104_0001_m_000003	111607	task_201406270104_0001_m_000042	115105
task_201406270104_0001_m_000004	103271	task_201406270104_0001_m_000043	139758
task_201406270104_0001_m_000005	121792	task_201406270104_0001_m_000044	149155
task_201406270104_0001_m_000006	105570	task_201406270104_0001_m_000045	124059
task_201406270104_0001_m_000007	97653	task_201406270104_0001_m_000046	118975
task_201406270104_0001_m_000008	123538	task_201406270104_0001_m_000047	106646
task_201406270104_0001_m_000009	111660	task_201406270104_0001_m_000048	95203
task_201406270104_0001_m_000010	101944	task_201406270104_0001_m_000049	136135
task_201406270104_0001_m_000011	114716	task_201406270104_0001_m_000050	196313
task_201406270104_0001_m_000012	126406	task_201406270104_0001_m_000051	98746
task_201406270104_0001_m_000013	99368	task_201406270104_0001_m_000052	132867
task_201406270104_0001_m_000014	81049	task_201406270104_0001_m_000053	187303
task_201406270104_0001_m_000015	142247	task_201406270104_0001_m_000054	92040
task_201406270104_0001_m_000016	99534	task_201406270104_0001_m_000055	141241
task_201406270104_0001_m_000017	82715	task_201406270104_0001_m_000056	187033

task_201406270104_0001_m_000018	142412	task_201406270104_0001_m_000057	89729
task_201406270104_0001_m_000019	98727	task_201406270104_0001_m_000058	81781
task_201406270104_0001_m_000020	78670	task_201406270104_0001_m_000059	96525
task_201406270104_0001_m_000021	147516	task_201406270104_0001_m_000060	99700
task_201406270104_0001_m_000022	107272	task_201406270104_0001_m_000061	99095
task_201406270104_0001_m_000023	82203	task_201406270104_0001_m_000062	178600
task_201406270104_0001_m_000024	97655	task_201406270104_0001_m_000063	117937
task_201406270104_0001_m_000025	123540	task_201406270104_0001_m_000064	93495
task_201406270104_0001_m_000026	169735	task_201406270104_0001_m_000065	162354
task_201406270104_0001_m_000027	120617	task_201406270104_0001_m_000066	97045
task_201406270104_0001_m_000028	135767	task_201406270104_0001_m_000067	88822
task_201406270104_0001_m_000029	164765	task_201406270104_0001_m_000068	176511
task_201406270104_0001_m_000030	107720	task_201406270104_0001_m_000069	111221
task_201406270104_0001_m_000031	134598	task_201406270104_0001_m_000070	95481
task_201406270104_0001_m_000032	147824	task_201406270104_0001_m_000071	144096
task_201406270104_0001_m_000033	110429	task_201406270104_0001_m_000072	3664
task_201406270104_0001_m_000034	144668	task_201406270104_0001_m_000073	3353
task_201406270104_0001_m_000035	128674	task_201406270104_0001_m_000074	3048
task_201406270104_0001_m_000036	130384	task_201406270104_0001_m_000075	4010
task_201406270104_0001_m_000037	123206	task_201406270104_0001_m_000076	3210
task_201406270104_0001_m_000038	149451	task_201406270104_0001_m_000077	4597

ตารางที่ 7-6 แสดงเวลาที่ใช้ในการประมวลผลแบบนับคำ (Word Count) ในแต่ละส่วนของงานแบบ Default Scheduler (100 Task แรก)

Task Attempt ID	T (ms)	Task Attempt ID	T (ms)
task_201406272144_0001_m_000000	100313	task_201406272144_0001_m_000050	47114
task_201406272144_0001_m_000001	100620	task_201406272144_0001_m_000051	102379
task_201406272144_0001_m_000002	139680	task_201406272144_0001_m_000052	110326
task_201406272144_0001_m_000003	100591	task_201406272144_0001_m_000053	147384
task_201406272144_0001_m_000004	42574	task_201406272144_0001_m_000054	58857
task_201406272144_0001_m_000005	48789	task_201406272144_0001_m_000055	95412
task_201406272144_0001_m_000006	125361	task_201406272144_0001_m_000056	43618

task_201406272144_0001_m_000007	95761	task_201406272144_0001_m_000057	83184
task_201406272144_0001_m_000008	107937	task_201406272144_0001_m_000058	42109
task_201406272144_0001_m_000009	88212	task_201406272144_0001_m_000059	93691
task_201406272144_0001_m_000010	57086	task_201406272144_0001_m_000060	139970
task_201406272144_0001_m_000011	68678	task_201406272144_0001_m_000061	51646
task_201406272144_0001_m_000012	40784	task_201406272144_0001_m_000062	43980
task_201406272144_0001_m_000013	54228	task_201406272144_0001_m_000063	53269
task_201406272144_0001_m_000014	82368	task_201406272144_0001_m_000064	117326
task_201406272144_0001_m_000015	34636	task_201406272144_0001_m_000065	99027
task_201406272144_0001_m_000016	103697	task_201406272144_0001_m_000066	112999
task_201406272144_0001_m_000017	101274	task_201406272144_0001_m_000067	44444
task_201406272144_0001_m_000018	118433	task_201406272144_0001_m_000068	114735
task_201406272144_0001_m_000019	132600	task_201406272144_0001_m_000069	61664
task_201406272144_0001_m_000020	116363	task_201406272144_0001_m_000070	43334
task_201406272144_0001_m_000021	95630	task_201406272144_0001_m_000071	140667
task_201406272144_0001_m_000022	55266	task_201406272144_0001_m_000072	98056
task_201406272144_0001_m_000023	55675	task_201406272144_0001_m_000073	85472
task_201406272144_0001_m_000024	163199	task_201406272144_0001_m_000074	103862
task_201406272144_0001_m_000025	113616	task_201406272144_0001_m_000075	92225
task_201406272144_0001_m_000026	78168	task_201406272144_0001_m_000076	55281
task_201406272144_0001_m_000027	40579	task_201406272144_0001_m_000077	50059
task_201406272144_0001_m_000028	94003	task_201406272144_0001_m_000078	97394
task_201406272144_0001_m_000029	39700	task_201406272144_0001_m_000079	104431
task_201406272144_0001_m_000030	42175	task_201406272144_0001_m_000080	92365
task_201406272144_0001_m_000031	79605	task_201406272144_0001_m_000081	52793
task_201406272144_0001_m_000032	114976	task_201406272144_0001_m_000082	134011
task_201406272144_0001_m_000033	84942	task_201406272144_0001_m_000083	56132
task_201406272144_0001_m_000034	47576	task_201406272144_0001_m_000084	40478
task_201406272144_0001_m_000035	120848	task_201406272144_0001_m_000085	96920
task_201406272144_0001_m_000036	104538	task_201406272144_0001_m_000086	169302
task_201406272144_0001_m_000037	111287	task_201406272144_0001_m_000087	83604
task_201406272144_0001_m_000038	43609	task_201406272144_0001_m_000088	43893

task_201406272144_0001_m_000039	119186	task_201406272144_0001_m_000089	68559
task_201406272144_0001_m_000040	129060	task_201406272144_0001_m_000090	110284
task_201406272144_0001_m_000041	41175	task_201406272144_0001_m_000091	39086
task_201406272144_0001_m_000042	62902	task_201406272144_0001_m_000092	109704
task_201406272144_0001_m_000043	88793	task_201406272144_0001_m_000093	54330
task_201406272144_0001_m_000044	42108	task_201406272144_0001_m_000094	100370
task_201406272144_0001_m_000045	220485	task_201406272144_0001_m_000095	125311
task_201406272144_0001_m_000046	89514	task_201406272144_0001_m_000096	47693
task_201406272144_0001_m_000047	60422	task_201406272144_0001_m_000097	48346
task_201406272144_0001_m_000048	107987	task_201406272144_0001_m_000098	144044
task_201406272144_0001_m_000049	41928	task_201406272144_0001_m_000099	100150

ตารางที่ 7-7 แสดงเวลาที่ใช้ในการประมวลผลแบบเรียง (Sorting) ในแต่ละส่วนของงาน แบบปรับปรุงอัลกอริทึมโดยผู้วิจัย

Task Attempt ID	T (ms)	Task Attempt ID	T (ms)
task_201406270201_0001_m_000000	53711	task_201406270201_0001_m_000019	53813
task_201406270201_0001_m_000001	54457	task_201406270201_0001_m_000020	62574
task_201406270201_0001_m_000002	53745	task_201406270201_0001_m_000021	63184
task_201406270201_0001_m_000003	58603	task_201406270201_0001_m_000022	52515
task_201406270201_0001_m_000004	49365	task_201406270201_0001_m_000023	51293
task_201406270201_0001_m_000005	60920	task_201406270201_0001_m_000024	48590
task_201406270201_0001_m_000006	42653	task_201406270201_0001_m_000025	50198
task_201406270201_0001_m_000007	43763	task_201406270201_0001_m_000026	50589
task_201406270201_0001_m_000008	43780	task_201406270201_0001_m_000027	35216
task_201406270201_0001_m_000009	48605	task_201406270201_0001_m_000028	51444
task_201406270201_0001_m_000010	36724	task_201406270201_0001_m_000029	38585
task_201406270201_0001_m_000011	51431	task_201406270201_0001_m_000030	50069
task_201406270201_0001_m_000012	60655	task_201406270201_0001_m_000031	50069
task_201406270201_0001_m_000013	61082	task_201406270201_0001_m_000032	54940
task_201406270201_0001_m_000014	60153	task_201406270201_0001_m_000033	54824
task_201406270201_0001_m_000015	61080	task_201406270201_0001_m_000034	45261
task_201406270201_0001_m_000016	62478	task_201406270201_0001_m_000035	44892

task_201406270201_0001_m_000017	57951	task_201406270201_0001_m_000036	1823
task_201406270201_0001_m_000018	53146	task_201406270201_0001_m_000037	3372
task_201406270201_0001_m_000000	53711	task_201406270201_0001_m_000019	53813
task_201406270201_0001_m_000001	54457	task_201406270201_0001_m_000020	62574
task_201406270201_0001_m_000002	53745	task_201406270201_0001_m_000021	63184
task_201406270201_0001_m_000003	58603	task_201406270201_0001_m_000022	52515
task_201406270201_0001_m_000004	49365	task_201406270201_0001_m_000023	51293
task_201406270201_0001_m_000005	60920	task_201406270201_0001_m_000024	48590
task_201406270201_0001_m_000006	42653	task_201406270201_0001_m_000025	50198
task_201406270201_0001_m_000007	43763	task_201406270201_0001_m_000026	50589
task_201406270201_0001_m_000008	43780	task_201406270201_0001_m_000027	35216
task_201406270201_0001_m_000009	48605	task_201406270201_0001_m_000028	51444
task_201406270201_0001_m_000010	36724	task_201406270201_0001_m_000029	38585
task_201406270201_0001_m_000011	51431	task_201406270201_0001_m_000030	50069
task_201406270201_0001_m_000012	60655	task_201406270201_0001_m_000031	50069
task_201406270201_0001_m_000013	61082	task_201406270201_0001_m_000032	54940
task_201406270201_0001_m_000014	60153	task_201406270201_0001_m_000033	54824
task_201406270201_0001_m_000015	61080	task_201406270201_0001_m_000034	45261
task_201406270201_0001_m_000016	62478	task_201406270201_0001_m_000035	44892
task_201406270201_0001_m_000017	57951	task_201406270201_0001_m_000036	1823
task_201406270201_0001_m_000018	53146	task_201406270201_0001_m_000037	3372

ตารางที่ 7-8 แสดงเวลาที่ใช้ในการประมวลผลแบบเรียง (Sorting) ในแต่ละส่วนของงาน แบบ Default Scheduler

Task Attempt ID	T (ms)	Task Attempt ID	T (ms)
task_201406280106_0001_m_000000	46515	task_201406280106_0001_m_000031	24890
task_201406280106_0001_m_000001	48046	task_201406280106_0001_m_000032	39777
task_201406280106_0001_m_000002	23880	task_201406280106_0001_m_000033	39475
task_201406280106_0001_m_000003	22966	task_201406280106_0001_m_000034	40823
task_201406280106_0001_m_000004	48557	task_201406280106_0001_m_000035	40764
task_201406280106_0001_m_000005	48558	task_201406280106_0001_m_000036	40020
task_201406280106_0001_m_000006	42573	task_201406280106_0001_m_000037	37409

task_201406280106_0001_m_000007	42252	task_201406280106_0001_m_000038	23858
task_201406280106_0001_m_000008	59767	task_201406280106_0001_m_000039	23543
task_201406280106_0001_m_000009	59460	task_201406280106_0001_m_000040	23310
task_201406280106_0001_m_000010	23583	task_201406280106_0001_m_000041	23715
task_201406280106_0001_m_000011	23143	task_201406280106_0001_m_000042	46671
task_201406280106_0001_m_000012	20792	task_201406280106_0001_m_000043	46324
task_201406280106_0001_m_000013	23767	task_201406280106_0001_m_000044	23561
task_201406280106_0001_m_000014	23451	task_201406280106_0001_m_000045	22959
task_201406280106_0001_m_000015	21389	task_201406280106_0001_m_000046	24038
task_201406280106_0001_m_000016	39279	task_201406280106_0001_m_000047	24549
task_201406280106_0001_m_000017	39280	task_201406280106_0001_m_000048	37682
task_201406280106_0001_m_000018	22519	task_201406280106_0001_m_000049	38595
task_201406280106_0001_m_000019	22863	task_201406280106_0001_m_000050	47873
task_201406280106_0001_m_000020	44855	task_201406280106_0001_m_000051	39172
task_201406280106_0001_m_000021	23826	task_201406280106_0001_m_000052	48175
task_201406280106_0001_m_000022	23522	task_201406280106_0001_m_000053	44449
task_201406280106_0001_m_000023	39590	task_201406280106_0001_m_000054	22929
task_201406280106_0001_m_000024	41090	task_201406280106_0001_m_000055	22015
task_201406280106_0001_m_000025	40048	task_201406280106_0001_m_000056	30127
task_201406280106_0001_m_000026	47430	task_201406280106_0001_m_000057	25317
task_201406280106_0001_m_000027	45881	task_201406280106_0001_m_000058	47609
task_201406280106_0001_m_000028	23586	task_201406280106_0001_m_000059	44606
task_201406280106_0001_m_000029	23873	task_201406280106_0001_m_000060	2820
task_201406280106_0001_m_000030	24581	task_201406280106_0001_m_000061	3951

ตารางที่ 7-9 แสดงค่าส่วนเบี่ยงเบนมาตรฐาน (Standard Derivation) และค่าเฉลี่ย (Mean) การประมวลผลแบบนับคำ (Word Count) และ การประมวลผลแบบเรียง (Sorting) แบบปรับปรุงโดยงานวิจัยและแบบดั่งเดิม

	WC IMPL (8G)	WC ORI (8G)	SORT IMPL (4G)	SORT ORI (4G)
Standard Derivation (SD)	28118.52	36123.58	7373.17	11094.31
Mean ($\bar{x}$)	122034.9	86002.27	52009.94	34585.95

กำหนดให้ค่าของ $\frac{1-\alpha}{2}$ = 0.985 และ ค่า z = 0.8365

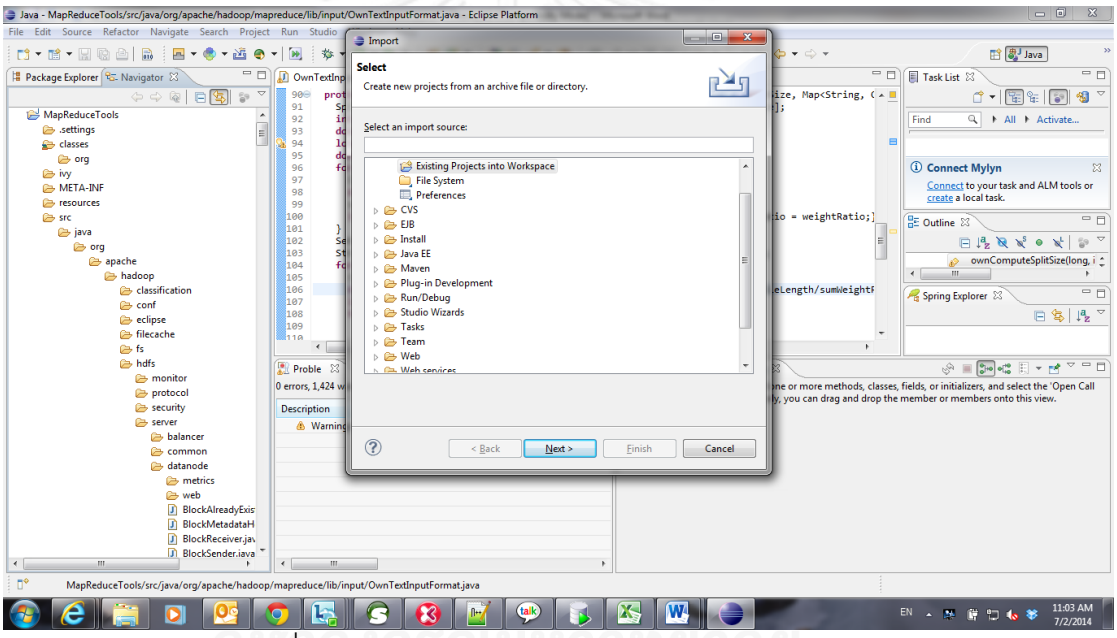
ตารางที่ 7-10 แสดงค่า Confident Interval ของการประมวลผลแบบนับคำ (Word Count) และการประมวลผลแบบเรียง (Sorting) แบบปรับปรุงโดยงานวิจัยและแบบดั่งเดิม

	n	Confident Interval at 97% (min, max) $(\frac{\bar{x}-(z\times SD)}{\sqrt{n}}, \frac{\bar{x}+(z\times SD)}{\sqrt{n}})$	Confident Interval at 97% (min, max) x n
WC IMPL (8G)	72	(121581.7, 122488.1)	(8753882.4, 8819143.2)
WC ORI (8G)	126	(85922.65, 86081.89)	(10826253, 10846318)
SORT IMPL (4G)	36	(50799.52, 53220.37)	(1828782.72, 1915920)
SORT ORI (4G)	60	(33387.86, 35784.04)	(2003220, 2147042)

ภาคผนวก ข.
ขั้นตอนการปรับปรุงฮาดูป

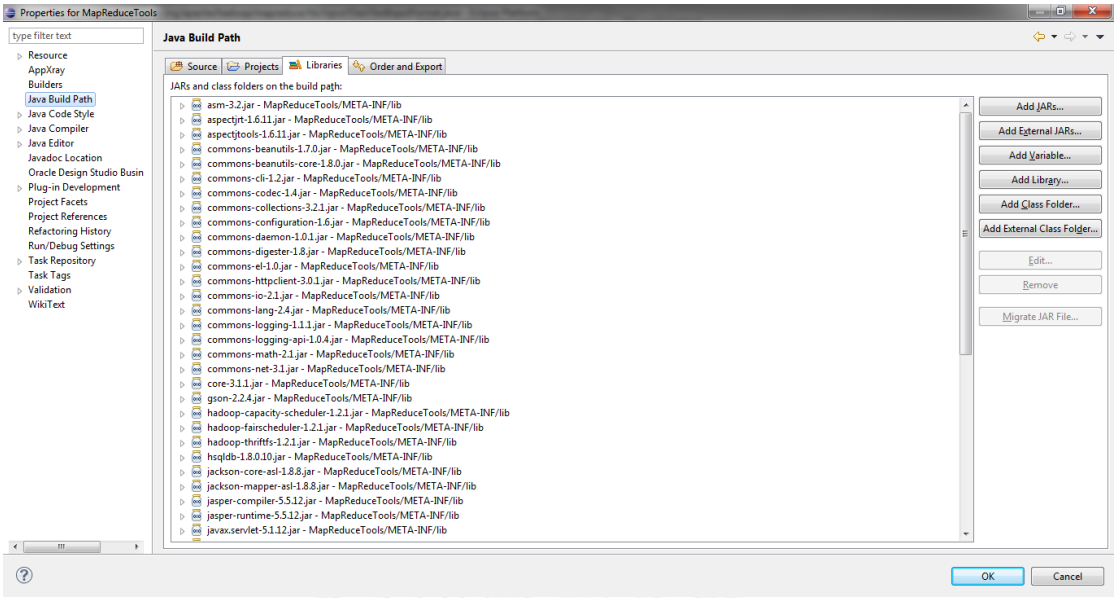
โดยการปรับปรุงฮาดูปนั้นมีขั้นตอนการปรับปรุงดังต่อไปนี้ โดยจะทำการอธิบายในลักษณะของการติดตั้ง, การนำเข้าโปรเจกและการปรับปรุง Source Code ไปยังเซิร์ฟเวอร์ โดยมีขั้นตอนการปฏิบัติดังต่อไปนี้

1. นำ Source Code ที่ Download จากส่วนของภาคผนวก ข มาทำการนำเข้าสู่ Eclipse IDE ดังรูปที่ 7-1



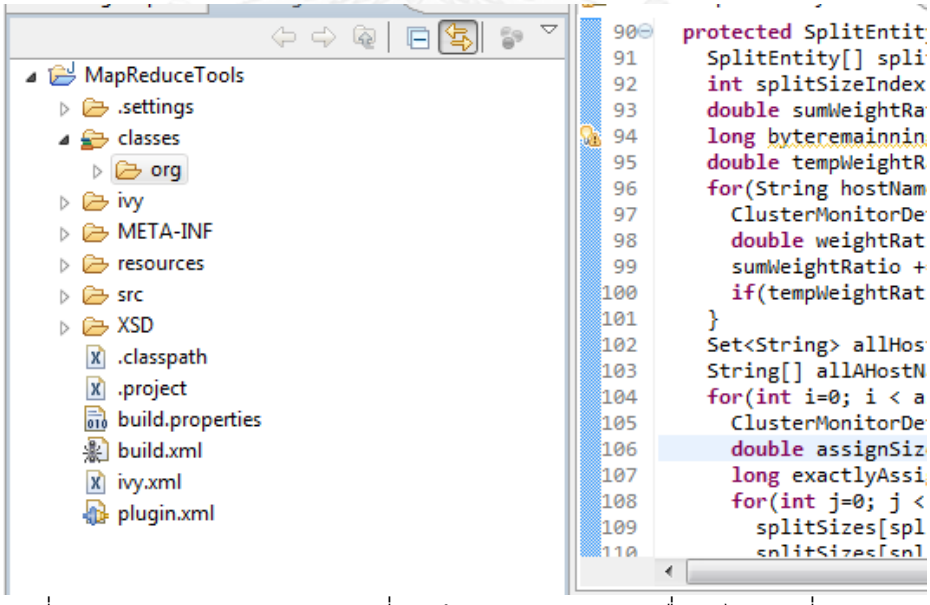
รูปที่ 7-1 แสดงการนำเข้าโปรเจกนำเข้าสู่ Eclipse IDE

2. หาก Source code Download จากฮาดูป Trunk ต้องทำการนำเข้าส่วนของไลบรารีจาก lib folder เพื่อให้สามารถ build project ได้ แต่หากนำจาก Source Code ของงานวิจัยมาทำการต่อยอดสามารถข้ามขั้นตอนนี้ได้ ดังแสดงในรูปที่ 7.2



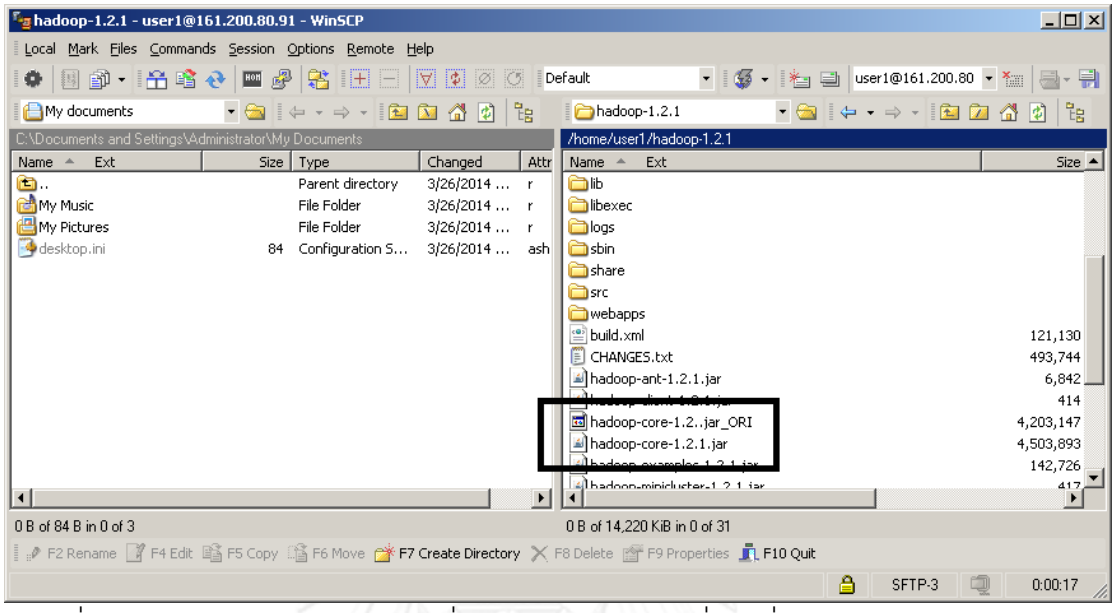
รูปที่ 7-2 แสดงการนำเข้าไลบรารีเพื่อทำการ Compile Project

3. ทำการแก้ Error ที่เกิดขึ้นให้หมดไป จากนั้นทำการ Build Project เพื่อนำไปวางที่เซิร์ฟเวอร์
4. เมื่อไม่มี Error แล้วและการ Build Project เสร็จสิ้นให้ทำการทำตาม Step ตามรูปที่ 7.3 โดยการ copy folder ที่อยู่ใต้ folder classes ทั้งหมด



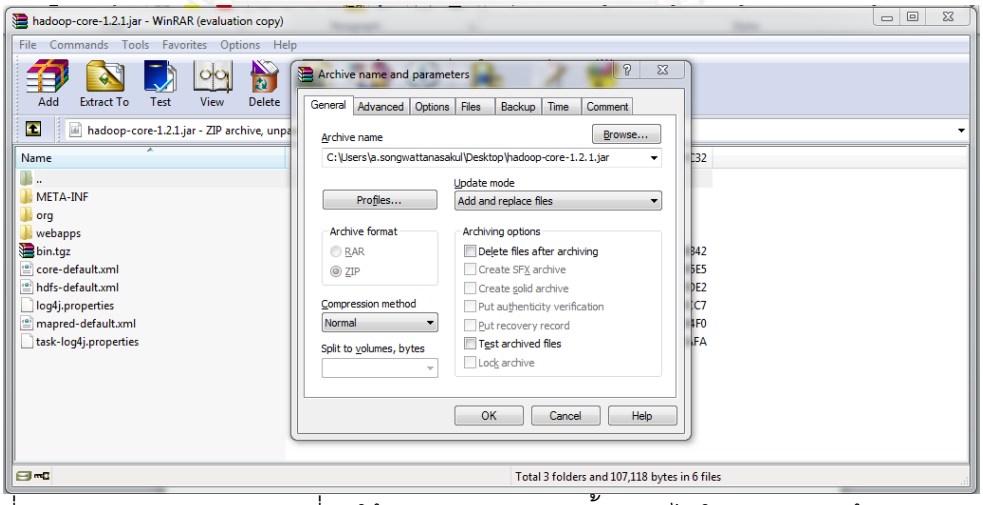
รูปที่ 7-3 แสดงการ copy folder ที่อยู่ใต้ folder classes เพื่อนำไปวางที่เซิร์ฟเวอร์

5. ทำการ copy file hadoop-core-*.jar จากเครื่องเซิร์ฟเวอร์ที่ทำหน้าที่เป็น NameNode นำมาไว้ที่ในเครื่อง



รูปที่ 7-4 แสดงการ copy file จากเครื่องเซิร์ฟเวอร์นำมาไว้ที่ในเครื่องด้วยโปรแกรม WINSCP

6. ทำการเปิด file hadoop-core-*.jar ด้วยโปรแกรม WINRAR แล้วทำการวาง (Paste) folder ที่อยู่ใต้ folder classes ทั้งหมด ไปใน JAR File ที่เปิดอยู่ดังรูปที่ 7-5



รูปที่ 7-5 แสดงการวางfolder ที่อยู่ใต้ folder classes ทั้งหมด ไปใน JAR File ด้วย WINRAR

7. ทำการ copy file hadoop-core-*.jar จากเครื่องตนเองกลับไปที่เครื่องเซิฟเวอร์ที่ทำหน้าที่เป็น NameNode

8. ทำการใช้คำสั่ง SCP Command เพื่อทำการ copy file hadoop-core-*.jar ไปที่เครื่อง DataNode ทั้งหมดที่อยู่ในคลัสเตอร์ โดยคำสั่ง Command นั้นแสดงในรูปที่ 7-6

รูปที่ 7-6 แสดงคำสั่ง SCP เพื่อทำการ copy hadoop-core-*.jar ในที่เครื่อง DataNode

9. ทำการ Restart ฮาดูปโดยใช้คำสั่ง bin/stop-all.sh และ start ด้วยคำสั่ง bin/start-all.sh แล้วทำการประมวลผลงานอีกครั้งจะเห็นการอัพเกรดของโปรแกรม

ภาคผนวก ค.
Source Code ต่าง ๆที่ใช้ในงานวิจัย

1. Source Code งานวิจัยสามารถ Download

   https://dl.dropboxusercontent.com/u/70232278/Research%20Utility/Hadoop%20Trunk.rar

2. Hadoop Tags

   https://svn.apache.org/repos/asf/hadoop/common/tags/release-1.2.1/

3. Word Count Generator

   https://dl.dropboxusercontent.com/u/70232278/Research%20Utility/TextFileGenerator.rar

ภาคผนวก ง.

ตารางแสดงค่า Z ในการคำนวณหาค่า Confident Interval

The *z* -table

Z	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0	0.5	0.504	0.508	0.512	0.516	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.591	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.648	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.67	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.695	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.719	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.758	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.791	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.834	0.8365	0.8389
1	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.877	0.879	0.881	0.883
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.898	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.937	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.975	0.9756	0.9761	0.9767
2	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.983	0.9834	0.9838	0.9842	0.9846	0.985	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.989
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.992	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.994	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.996	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.997	0.9971	0.9972	0.9973	0.9974

รูปที่ 7-7 แสดงตารางแสดงค่า Z ในการคำนวณหาค่า Confident Interval

ประวัติผู้เขียนวิทยานิพนธ์

นายอภิลักษณ์ ทรงวัฒนาสกุล เกิดเมื่อวันที่ 26 กันยายน พ.ศ. 2532 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต (วศ.บ.) สาขาคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปีการศึกษา 2553 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิทยาศาสตร์คอมพิวเตอร์ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2555