ประสิทธิภาพการสืบค้นข้อมูลวิทยานิพนธ์ใน ThaiLIS ด้วยการประมวลผลแบบขนานบนคลัสเตอร์คอมพิวเตอร์โดยใช้เทคนิคแม็พรีดิวซ์

จักรกฤษณ์ แสงแก้ว [1]

บทคัดย่อ

งานวิจัยนี้มีวัตถุประสงค์เพื่อเปรียบเทียบประสิทธิภาพด้านความเร็วในการสืบค้นระหว่างการค้นผ่านระบบฐานข้อมูลเชิงสัมพันธ์ที่ใช้ในการให้บริการวิทยานิพนธ์ของ ThaiLIS และวิธีการประมวลผลแบบขนานบนคลัสเตอร์คอมพิวเตอร์ที่มีจำนวนโหนดแม่ข่ายที่แตกต่างกันโดยใช้เทคนิคแม็พรีดิวซ์ การวิจัยใช้ชุดข้อมูลวิทยานิพนธ์ที่สกัดจากฐานข้อมูล TDC (Thai Digital Collection) ในโครงการเครือข่ายห้องสมุดในประเทศไทยหรือ ThaiLIS (Thai Library Integrated System) จำนวนทั้งสิ้น 443,694 เรคคอร์ด ทำการประมวลผลข้อมูลเพื่อเปรียบเทียบความเร็วในการสืบค้น พบว่า ระบบฐานข้อมูลเชิงสัมพันธ์ใช้ความเร็วเฉลี่ย 1.86 วินาที ส่วนการประมวลผลแบบขนานบนคลัสเตอร์ที่ใช้แม่ข่าย 4 , 8, 12 และ 16 โหนดใช้เวลาในการประมวลผลน้อยกว่าอย่างชัดเจนด้วยความเร็ว 0.278, 0.135, 0.091 และ 0.065 วินาที ประสิทธิภาพด้านความเร็วการสืบค้นเพิ่มขึ้น 85.0% , 92.73%, 95.10% และ 96.50% ตามลำดับ ผลจากการวิจัยชี้ให้เห็นว่า ThaiLIS สามารถปรับปรุงบริการการสืบค้นข้อมูลวิทยานิพนธ์จากฐานข้อมูล TDC ให้มีประสิทธิภาพการค้นที่รวดเร็วขึ้นได้โดยเฉพาะอย่างยิ่งสำหรับกรณีที่จำนวนเรคคอร์ดที่มีอยู่ในฐานข้อมูลมีปริมาณเพิ่มมากขึ้นอย่างต่อเนื่องในอนาคต

คำสำคัญ: แม็พรีดิวซ์; การประมวลผลแบบขนาน; คลัสเตอร์คอมพิวเตอร์; วิทยานิพนธ์; TDC; ThaiLIS

[1] หน่วยวิจัยวิทยาการข้อมูลและนวัตกรรมดิจิทัล สาขาวิชาสารสนเทศศาสตร์ คณะวิทยาการสารสนเทศ มหาวิทยาลัยมหาสารคาม
อีเมล: chakkrit@msu.ac.th

The Effectiveness of Theses Data Retrieval from ThaiLIS Using Parallel Processing on Cluster Computer with Mapreduce Technique

Chakkrit Saengkaew [1]

Abstract

This research aimed to compare the speed effectiveness of thesis data retrieval between the relational database system for theses information services in ThaiLIS (Thai Library Integrated System) and the parallel processing system on cluster computer with different numbers of network's nodes by using Mapreduce technique. The research used the theses data set totally 443,694 records extracted from TDC (Thai Digital Collection) database in ThaiLIS. The data processing was done to compare the speed of data retrieval. It was found that the speed of relational database system consumed averagely 1.86 seconds, while the speed of parallel processing system on cluster computer with network's nodes at 4, 8, 12 and 16 consumed lesser time at 0.278, 0.135, 0.091 and 0.065 respectively. The research result indicated that ThaiLIS can improve the effectiveness of theses information retrieval services from TDC database to be faster, especially in case of the quantity of records in the database will be instantly increasing in the future.

Keywords: Mapreduce, Parallel Processing, Cluster Computer, Thesis, TDC, ThaiLIS

[1] Data Science and Digital Innovation Research Unit, Division of Information Science, Faculty of Informatics, Mahasarakham University, E-mail: chakkrit@msu.ac.th

บทนำ

ในยุคโลกาภิวัตน์นี้ ข้อมูลข่าวสารสามารถเชื่อมโยงถึงกันได้อย่างรวดเร็วในเสี้ยววินาทีอย่างที่ไม่เคยปรากฏมาก่อน เทคโนโลยีสารสนเทศและการสื่อสารยังคงต้องอาศัยหน่วยประมวลผลกลางเป็นพื้นฐานสำคัญสำหรับข้อมูลที่มีปริมาณมากและจำนวนผู้ใช้ที่สูงขึ้น การประมวลผลด้วยคอมพิวเตอร์เพียงลำพังเครื่องเดียวไม่สามารถจะรองรับปริมาณการคำนวณที่มากขึ้นได้อีกต่อไป จึงได้มีนักวิจัยที่สนใจนำเอาคอมพิวเตอร์หลายเครื่องมาประมวลผลช่วยกันเพื่อให้เกิดปริมาณงานที่ทำ (Throughput) ที่มากขึ้นด้วย งานวิจัยชิ้นนี้เป็นการนำเอาหลักการประมวลผลแบบขนานมาใช้งานและเปรียบเทียบให้เห็นถึงประสิทธิภาพการสืบค้นที่สูงขึ้น โดยใช้ข้อมูลวิทยานิพนธ์จาก TDC (Thai Digital Collection) ของโครงการเครือข่ายห้องสมุดในประเทศไทย หรือ ThaiLIS (Thai Library Integrated System) ซึ่งถือเป็นแหล่งทรัพยากรด้านงานวิจัยที่สำคัญของประเทศไทย ในปัจจุบันฐานข้อมูลวิทยานิพนธ์ไทยใน TDC ยังคงให้บริการเฉพาะสมาชิกเท่านั้น หากพิจารณาในด้านความเร็วการประมวลผลพบว่าในการสืบค้นแบบ "ส่วนใดส่วนหนึ่ง" ซึ่งเป็นการสืบค้นที่ต้องอาศัยการประมวลผลที่มีประสิทธิภาพสูงพบว่า TDC ใช้เวลาประมวลผลประมาณ 1.8 วินาทีต่อทรานแซคชั่น นอกจากนั้น ในอนาคตฐานข้อมูลวิทยานิพนธ์ไทยนี้อาจจะอนุญาตให้เข้าถึงได้โดยบุคคลทั่วไป ถึงเวลานั้นจะมีผู้ใช้เข้ามาสืบค้นข้อมูลจำนวนมากและส่งผลให้ใช้เวลาในการสืบค้นมากขึ้น จนกระทั่งไม่สามารถรองรับปริมาณการใช้งานได้ในที่สุด ในงานวิจัยครั้งนี้ผู้วิจัยนำเสนอวิธีการหนึ่งที่ช่วยลดเวลาในการประมวลผล โดยใช้แนวคิดของแม็พรีดิวซ์และการประมวลผลแบบขนานบนคลัสเตอร์คอมพิวเตอร์

วัตถุประสงค์การวิจัย

การวิจัยนี้มีวัตถุประสงค์เพื่อเปรียบเทียบประสิทธิภาพด้านความเร็วในการสืบค้นข้อมูลวิทยานิพนธ์จำนวนมากจากฐานข้อมูล TDC ระหว่างการสืบค้นผ่านระบบฐานข้อมูลเชิงสัมพันธ์แบบปกติ และการประมวลผลแบบขนานบนบนคลัสเตอร์คอมพิวเตอร์ที่มีจำนวนโหนดแม่ข่ายที่แตกต่างกัน โดยใช้เทคนิคแม็พรีดิวซ์

ขอบเขตการวิจัย

การวิจัยนี้มีขอบเขตในการศึกษาประสิทธิภาพของการสืบค้น โดยเน้นที่ความเร็วในการประมวลผลด้วยการสืบค้นแบบ "ส่วนใดส่วนหนึ่ง" จากคำสั่ง Like ภายในภาษา SQL ของระบบฐานข้อมูลเชิงสัมพันธ์และเปรียบเทียบกับความเร็วในการประมวลผลแบบขนานบนคลัสเตอร์คอมพิวเตอร์ที่มีจำนวนโหนดแม่ข่าย 4, 8, 12 และ 16 โหนดตามลำดับ โดยที่ชุดข้อมูลที่เก็บบนคลัสเตอร์ถูกจัดให้อยู่ในรูปแบบของบิ๊กเทเบิ้ล (Bigtable)

ทฤษฎีที่เกี่ยวข้อง

1. แม็พรีดิวซ์ (Mapreduce)

แม็พรีดิวซ์เผยแพร่เป็นครั้งแรกโดยบริษัทกูเกิ้ลเพื่อใช้สำหรับการประมวลผลแบบขนาน โดยกระจายงานไปยังคอมพิวเตอร์แม่ข่ายเพื่อให้ทำการประมวลผลพร้อมกัน (Map) และนำผลลัพธ์ที่ผ่านการประมวลผลจากแม่ข่ายมารวมกันด้วยการทำรีดิวซ์ (Reduce) โดยคำสั่ง map และ reduce (Dean and Ghemawat, 2008) เป็นแนวคิดของการโปรแกรมเชิงฟังก์ชั่น (Functional programming) และเป็นคำสั่งพื้นฐานของภาษาไพธอน (Python programming language) (Samimi, 2013) โดยหลักการทำงานเป็นการรวมกันของตัวแม็พ (Mapper)

และตัวรีดิวเซอร์ (Reducer) ที่ทำงานร่วมกันเพื่อประมวลผลข้อมูล โดยที่ตัวแม็พจะประมวลผลข้อมูลดิบและเก็บข้อมูลไว้ในรูปของคีย์และค่าข้อมูล (Key/Value) ซึ่งจะเรียกผลลัพธ์ที่อยู่ระหว่างการประมวลผลซึ่งยังไม่ได้ผลลัพธ์ (Intermediate results) ในขั้นตอนสุดท้ายตัวรีดิวเซอร์จะทำการรวมผลลัพธ์ที่อยู่ระหว่างการประมวลผลก่อนได้ผลลัพธ์เพื่อสร้างเป็นผลลัพธ์สุดท้าย

คำสั่ง map และ reduce ภายในภาษาไพธอนแสดงได้ดังนี้

>>> def adder(x,y): return x+y

>>> map(adder, [4,7,9], [2,1,3]) ซึ่งมีผลลัพธ์เท่ากับ [6, 8, 12]

คำสั่ง reduce เป็นการรวมผลลัพธ์ เช่น reduce(adder, [6, 8, 12]) ผลลัพธ์มีค่า : 26

คำสั่ง lambda เป็นการสร้างฟังก์ชั่นขณะโปรแกรมทำงาน โดยไม่ต้องกำหนดฟังก์ชั่นก่อนเหมือนการประกาศใช้ฟังก์ชั่นโดยทั่วไป มีความยืดหยุ่นสูง เรียกอีกชื่อหนึ่งว่า Anonymous function แสดงตัวอย่างการใช้ lambda ด้วยภาษาไพธอน เช่น adder = lambda x,y:x+y จากนั้นสามารถเรียกใช้ฟังก์ชั่น adder() ได้โดยตรงในขณะที่โปรแกรมกำลังทำงานแบบเรียลไทม์ เช่น map(lambda x,y:x+y, [4,7,9], [2,1,3]) ซึ่งมีค่าผลลัพธ์เป็น [6, 8, 12]

2. บิ๊กเทเบิ้ล (Bigtable)

บิ๊กเทเบิ้ลเป็นตารางการเก็บข้อมูลที่มีโครงสร้าง (Structureddata) ที่ได้รับการเผยแพร่โดยบริษัทกูเกิ้ล ออกแบบเพื่อให้สามารถขยายขนาดของข้อมูลออกไปได้ไม่จำกัด สามารถเก็บข้อมูลข้ามแม่ข่ายหลายพันเครื่องเข้าด้วยกัน (Chang*et al.,* 2008) กูเกิ้ลมีแอพลิเคชั่นมากกว่า 60 รายการที่ใช้โครงสร้างของบิ๊กเทเบิ้ล ได้แก่ ดัชนีเว็บ (Web indexing), Google Earth และ Google Finance เผยแพร่ใน ACM-TCM 2008 บิ๊กเทเบิ้ลประสบผลสำเร็จอย่างสูงเนื่องจากข้อมูลที่ใช้มีความยืดหยุ่น และนำไปใช้ในการแก้ปัญหาในงานด้านต่าง ๆ อย่างมีประสิทธิภาพ (รูปที่ 1)

Title	Creator	CreatorOrgname	CreatorEmail	Subject	DescriptionAbstract	Publisher	PublisherAddress	. . .	DateValid

รูปที่ 1 แสดงตัวอย่างของข้อมูลที่เก็บเนื้อหาข้อมูลของหน้าเว็บในบิ๊กเทเบิ้ล

3. Message Passing Interface (MPI)

MPI ย่อมาจาก Message Passing Interface ซึ่งเป็นชุดคำสั่ง API (Application Programming Interface) สำหรับภาษา C/C++ และ FORTRAN เพื่ออนุญาตให้คอมพิวเตอร์ในเครือข่ายสามารถประมวลผลแบบขนานได้โดยในตอนแรก MPI เน้นที่การกระจายหน่วยความจำ (Distributed memory) กับคอมพิวเตอร์ภายในคลัสเตอร์ ต่อมาได้มีการเพิ่มความสามารถในการกระจายและแบ่งหน่วยความจำร่วมกัน (Shared memory) ในปัจจุบันมีไลบรารี่สำหรับ MPI หลายตัว อาทิ OpenMPI, MPICH, HP-MPI, Intel MPI เป็นต้น (Graham et al., 2006)

4. การสร้างดีสโตร (Distro)

คลัสเตอร์คอมพิวเตอร์แต่ละตัวจะต้องถูกติดตั้งโปรแกรมตลอดจนการกำหนดค่าต่าง ๆ ภายในระบบปฏิบัติการและมีลักษณะคล้ายคลึงกันกับโหนดอื่น ๆ ดังนั้น การจัดการสร้างดีสโตร (Distro) เพื่อให้ได้ผลลัพธ์เป็นไฟล์ .ISO และสามารถนำไปบูต (Boot) กับแต่ละโหนดได้ด้วยระบบปฏิบัติการและซอฟต์แวร์จัดการต่าง ๆ แบบเดียวกัน เป็นวิธีหนึ่งที่มีความสะดวกและมีประสิทธิภาพ ซอฟต์แวร์ที่ใช้สร้างดีสโตรมีหลายตัว ได้แก่ RemasterSys และ SystemBack ในงานวิจัยนี้เลือกใช้โปรแกรม SystemBack ซึ่งมาพร้อมกับระบบปฏิบัติการ Linux Lite ใช้งานได้โดยไม่เสียค่าใช้จ่ายเกี่ยวกับลิขสิทธิ์และอยู่ในกลุ่มโอเพ่นซอร์ส (Rezkalla et al., 2015)

5. การประมวลผลแบบขนาน

การประมวลผลแบบขนาน ใช้หลักการของ Threading API ใน Symmetric multiprocessing (Martens, 2003) ส่วนใหญ่ใช้ process สร้างฟังก์ชั่น เช่น Unix fork system call สำหรับไลบรารี่ที่สนับสนุน Symmetric multiprocessing ได้แก่

(1) *Disp* เป็น python module สำหรับการประมวลผลพร้อมกัน โดยกำหนดตารางงานด้วยพารามิเตอร์ในรูปแบบ SIMD ทำให้เกิดประสิทธิภาพและความสามารถในการขยายระบบออกไปได้

(2) *Delegate* เป็น fork based process สร้างด้วย pickled data ส่งผ่านข้อมูลด้วย pipes

(3) *Forkmap* เป็น fork-based process creation ใช้ฟังก์ชั่น map (unix, mac, cygwin)

(4) *POSH* เป็น python object sharing อนุญาตให้ object ต่าง ๆ เก็บไว้ใน shared memory ทำงานใน posix/unix/linux เท่านั้น

(5) *Pp(Parallel python)* เป็น process based ทำงานด้วย job-oriented solution กับ cluster ทำงานในระบบปฏิบัติการ windows, linux, unix และ mac

(6) *Pprocess* (เป็นเวอร์ชั่นก่อนหน้าของ parallel/process) ทำงานเป็น fork-based process creation ด้วยวิธี asynchronous channel-based communication ใช้ข้อมูลแบบ pickled data

(7) *Processing* เป็น process-based ใช้ทั้ง fork บน unix หรือ subprocess module บนวินโดวส์ สร้างเป็น API เหมือน standard library threading API ที่มีการเตรียมอ็อบเจ็คเกี่ยวกับ queues และ semaphores

(8) *PySCP* เป็นsequential processes ทำงานในภาษาไพธอนที่อนุญาตให้สร้าง processes และ synchronized ได้

(9) *remoteD* เป็น fork-based process creation ทำงานกับตัวแปรชนิดดิกชันนารีในภาษาไพธอน (Fu et al., 2016; Crampton and Khambhammettu, 2008)

6. Cluster Computing

สถาปัตยกรรมคลัสเตอร์ (Cluster architecture) มีความสามารถในการขยายการประมวลผล (High scalability) และการใช้ทรัพยากรร่วมกัน (Shared resource) เกี่ยวข้องกับเทคโนโลยีการประมวลผลแบบกระจาย (Distributed computing technologies) ประกอบด้วยไลบรารี่ต่าง ๆ ได้แก่

(1) *Batchlib* เป็นdistributed computation system ทำงานเกี่ยวกับการเลือก processing service ปัจจุบันไม่มีการพัฒนาต่อแล้ว

(2) *Celery* เป็น distributed task queue ทำงานบน distributed message passing

(3) *Deap* เป็นการพัฒนาอัลกอริทึมที่ประกอบด้วยโมดูล parallelization ที่ชื่อว่า DTM ย่อมาจาก Distributed Task Manager ช่วยประมวลผลขนานบนคลัสเตอร์

(4) *Disco*เป็นการใช้ map-reduce โดยบริษัทโนเกีย ได้รับแรงบันดาลใจจาก Google mapreduce และ Apache hadoop

(5) *Distributed Python* เป็น simple python distributed computing framework ใช้ SSH และ multiprocessing และ subprocess modules ทำงานที่ top level สามารถสังเคราะห์รายการคำสั่งและนำไปประมวลผล (executed) ใน parallel และทำงานใน Python 2.6 และ 3.0

(6) *Exec_Proxy* เป็นระบบสำหรับการประมวลผล arbitary program และ transferring files โครงการนี้ยุติการพัฒนาแล้ว

(7) *ExecNet* เป็น asynchronous execution สำหรับ client-provided core fragments

(8) *Ipython* เป็น Ipythonshell สนับสนุน interactive parallel computing ข้าม multiple Ipython instances (Bernard, 2013) ผู้วิจัยเลือกใช้ Ipython สำหรับโครงการวิจัยการเปรียบเทียบประสิทธิภาพการประมวลผลแบบขนานบนคลัสเตอร์คอมพิวเตอร์ โดยใช้เทคนิคแม็พรีดิวซ์กับข้อมูลวิทยานิพนธ์ไทยในฐานข้อมูล TDC ของโครงการเครือข่ายห้องสมุดในประเทศไทย (ThaiLIS-Thai Library Integrated System) (Office of Information Technology Administration for Educational Development, 2017)

(9) *JuG* เป็น task based parallel framework

(10) *MPI4Py* เป็น MPI-based solution

(11) *NetWorkSpace* เดิมชื่อ Lindspaces สนับสนุนภาษาไพธอน

(12) *PaPY* เป็น parallel และ distributed work flow engine (RpyC) และการใช้ imap

(13) *Papyros* เป็น lightweight master-slave based parallel processing โดย clients จะส่งงานไปยังเครื่องมาสเตอร์ และใช้ multiple threads และ multiple processes ภายใน hosts ผ่าน PyRO

(14) *PP (Parallel Python)* เป็นโมดูลภาษาไพธอนที่เตรียมกลไกสำหรับ parallel execution สำหรับไพธอนโค้ดบน SMP (System with Multiple Processors) หรือ cores และคลัสเตอร์ผ่านเน็ตเวิร์ก

(15) *PyLinda* เป็น distributed computing โดยใช้ tuple spaces

(16) *PyMPI* ทำงานบน MPI-based solution

(17) *PyPar* เป็น numberic python และ MPI-based solution

(18) *PyPastSet* เป็น tuple-based structured ทำงานบน distributed shared memory system ในภาษาไพธอนโดยใช้ Pyro framework

(19) *PyPVM* เป็น PVM-based solution

(20) *PyNPVM* เป็น PVM-based solution สำหรับการคำนวณเชิงตัวเลขด้วย NumPy

(21) *PyRO* เป็น Python remote object ทำงานใน distributed object system โดยผ่านเน็ตเวิร์กและอ็อบเจ็คที่แบ่งไปยังคอมพิวเตอร์อื่น ๆ บนเน็ตเวิร์ก

(22) *Rthread* เป็น distributed computing ทำงานผ่าน SSH

(23) *Scientific Python* เป็นแพคเกจย่อยสำหรับงานประมวลผลขนาน ได้แก่ 1) Scientific distributed computing master slave สร้างโดย master-slave model ที่ตัวมาสเตอร์จะส่งความต้องการคำนวณและจะถูกประมวลผลโดย slave processes มีข้อดีคือทำงานได้หลาย ๆ โปรเซสลูก แต่ไม่เหมาะกับแอพลิเคชั่นประมวลผลขนานที่มีขนาดใหญ่และทำงานในลักษณะโมดูล 2) Scientific BSP เป็นอ็อบเจ็คสำหรับ Bulk synchronous parallel (BSP) model สำหรับการประมวลผลแบบขนาน มีข้อดีคือ ทำงานบน message passing และเป็น deadlocks ทำงานได้กับทุกแพลตฟอร์มที่มี MPI library หรือ BSPlib 3) Scientific MPI เป็นระบบโต้ตอบระหว่าง MPI ที่เน้นการรวมภาษาไพธอนและภาษาซีโดยใช้ MPI ผ่าน PyPAR และ PyMPI เหมาะกับการทำงานประมวลผลตัวเลขกับทุกแพลตฟอร์มที่มี MPI library

(24) *SCOOP (Scalable Concurrent Operations in Python)* เป็น distributed task module ที่ทำงานพร้อม ๆ กันในสภาพแวดล้อมที่ต่างกัน (Heterogenous grids)

(25) *Seppo* ทำงานบน PyRO mobile code มีกลไกแม็พฟังก์ชั่นแบบทำงานขนานพร้อมกัน

(26) *Star-P* เป็นแพลตฟอร์มสำหรับการประมวลผลขนานที่โต้ตอบกับผู้ใช้แบบอินเตอร์แอคทีฟผ่านเชลล์

(27) *Superpy* เป็นไลบรารี่ภาษาไพธอนสามารถประมวลผลข้ามคลัสเตอร์หรือโปรเซสในคอมพิวเตอร์เครื่องเดียว ข้อเด่นคือ 1) ส่งงานไปยังรีโมทหรือเครื่องเดียวกันด้วย XML RPC call 2) มีกราฟิกอินเตอร์เฟสเพื่อตรวจสอบและยุติการทำงานของ tasks 3) มีกราฟิกอินเตอร์เฟสสำหรับเรียก task ต่าง ๆ ได้ในทุกชั่วโมงหรือตามที่กำหนด 4) ทำงานภายในระบบปฏิบัติการไมโครซอฟต์วินโดวส์เป็น windows service 5) ข้อมูลเข้าและออกเป็นไพธอนอ็อบเจ็คที่ใช้แพคเกจ pickle 6) พัฒนาขึ้นจากภาษาไพธอนล้วน 7) สนับสนุนการทำ load balance เพื่อส่งงานไปยังแม่ข่ายที่เหมาะสมที่สุดในขณะนั้น

7. การประมวลผลในกลุ่มเมฆ (Cloud Computing)

การประมวลผลในกลุ่มเมฆคล้ายกับการประมวลผลคลัสเตอร์ ยกเว้นแหล่งทรัพยากรต่าง ๆ ซึ่งจัดการผ่านผู้ให้บริการ (Sobie et al., 2013) โดยไม่ต้องซื้อและติดตั้งฮาร์ดแวร์ และพัฒนาบนเวิร์คโหลดที่ทำงานพร้อม กันจำนวนมาก อีกทั้งมีราคาถูกและง่ายกว่า ได้แก่ (1) *Google App Engine* สนับสนุนภาษาไพธอน (2) *PiCloud* เป็น Cloudcomputing platform ที่รวมภาษาไพธอนและอนุญาตให้นักพัฒนาสามารถคำนวณ Amazon web service (AWS) (3) *StartCluster* เป็นเครื่องมือการคำนวณแบบคลัสเตอร์สำหรับ AWS cloud ออกแบบให้ใช้งานง่ายและจัดการคลัสเตอร์ต่าง ๆ ในลักษณะคอมพิวเตอร์เสมือนบน Amazon's EC2 cloud ช่วยให้สร้างคลัสเตอร์ได้ง่ายโดยเพิ่มและลบโหนดต่าง ๆ ที่กำลังทำงานอยู่ได้

8. การประมวลผลกริด (Grid Computing)

การประมวลผลกริดประกอบด้วยไลบรารี่ต่าง ๆ ที่สนับสนุนภาษาไพธอน ได้แก่ (1) *Ganga* เป็นการโต้ตอบระหว่างกริดที่พัฒนาโดย ATLAS และ LHCb จาก CERN (2) *PEG* เป็นส่วนขยายสำหรับประมวลผลกริดด้วยภาษาไพธอน (3) *PyGlobus* เป็นโปรเจ็ค python core สำหรับระบบกริด (Vestal et al., 2007)

9. การใช้งาน Ipython Notebook

ในการวิจัยนี้ ใช้การประมวลผลผลแบบขนานด้วย Ipython notebook ดังนี้

9.1 การสร้างโพรไฟล์สำหรับเครื่องมาสเตอร์ (Master profile) สำหรับเครื่อง master จะต้องทำการสร้างโพรไฟล์ เพื่อนำค่าโพรไฟล์ส่งไปให้คลัสเตอร์คอมพิวเตอร์ภายในระบบเน็ตเวิร์ก ในตัวอย่างนี้ ตั้งชื่อ

ว่า mycluster ด้วยคำสั่ง $ ipython profile create --parallel --profile=mycluster จากนั้นเรียก ipcontroller โดยระบุโพรไฟล์และไอพีของคอนโทรลเลอร์ ด้วยคำสั่ง $ ipcontroller --profile=mycluster --ip=192.168.1.111 เมื่อสร้างโพรไฟล์โปรแกรมจะสังเคราะห์ไฟล์ต่าง ๆ เพื่อใช้กำหนดค่าต่าง ๆ ในโฟลเดอร์ ~/.ipython/profile_mycluster/ ให้ทำการคัดลอกไฟล์ ipcontroller-engine.json ภายใต้โฟลเดอร์ดังกล่าวไปเก็บไว้ในแต่ละคลัสเตอร์คอมพิวเตอร์ทุก ๆ เครื่องที่ต้องการจะเชื่อมเข้าเป็นคลัสเตอร์คอมพิวเตอร์ของเน็ตเวิร์ก

9.2 การสร้างเครื่องงานหรือโหนด (workers) ในการสร้างเครื่องงานจะต้องทำการสร้างโพรไฟล์เหมือนกับในเครื่องมาสเตอร์ ด้วยคำสั่ง $ ipython profile create --parallel --profile=mycluster หลังจากสร้างโพรไฟล์เสร็จเรียบร้อย ทำการคัดลอกไฟล์ ipcontroller-engine.json จากเครื่องมาสเตอร์มาเก็บไว้ภายในคลัสเตอร์คอมพิวเตอร์ และเรียกคำสั่ง $ ipengine --profile =mycluster --file=ipcontroller-engine.json

9.3 การควบคุมคลัสเตอร์คอมพิวเตอร์จากเครื่องมาสเตอร์ เรียกใช้คำสั่ง ipython notebook --profile=mycluster หรือ ipythonqtconsole --profile=mycluster หากต้องการควบคุมผ่านหน้าเว็บหรือคอนโซลตามลำดับ

9.4 การเพิ่มเครื่องโหนดเข้าสู่ระบบคลัสเตอร์ จำเป็นต้องรีโมทเข้าไปยังคลัสเตอร์คอมพิวเตอร์ที่ต้องการนำมาเชื่อมเข้ากับคลัสเตอร์ โดยดำเนินการตามขั้นตอนที่ผ่านมาเพื่อเพิ่มคลัสเตอร์คอมพิวเตอร์ (worker) เมื่อดำเนินการเรียบร้อยแล้วจะพบว่ามีคลัสเตอร์คอมพิวเตอร์ที่เชื่อมต่อเข้ามาแสดงรายละเอียดด้วยคำสั่ง

```
$ fromIpython.parallel import Client
c = Client()
printc.ids
[0, 1, 2, 3]
```

ในตัวอย่างแสดงให้เห็นว่ามีการเชื่อมต่อโหนดจำนวน 4 โหนด

9.5 การส่งคำสั่งไปยังคลัสเตอร์คอมพิวเตอร์เพื่อประมวลผลสคริปต์ใช้คำสั่ง ดังนี้

```
fromIpython.parallel import Client
c = Client()
view = c[:]
vidw.activate()
view.run("my-thailis-mapfunction.py")
```

ตัวแปร view หมายถึงเครื่องคลัสเตอร์คอมพิวเตอร์ที่ใช้งานขณะนี้ กำหนดด้วย c[:] หมายถึง ทุกเครื่องที่เป็นสมาชิกภายในตัวแปรคลัสเตอร์ ส่วนตัวแปร view.activate() เป็นการเรียกให้ worker เตรียมพร้อมรับคำสั่งเพื่อนำไปประมวลผล ในขณะที่คำสั่ง view.run() เป็นการให้ worker แต่ละเครื่องนำสคริปต์ "my-thailis-mapfunction.py" ไปทำงานบนเครื่องของตนเอง

9.6 การตรวจสอบผลลัพธ์จากคลัสเตอร์คอมพิวเตอร์ (worker)

```
fromIpython.parallel import Client
c = Client()
```

```
view.c[:]
view.activate()
adder = lambda(adder a,b:a+b)
ans = c[0].apply(adder,3,2)
ifans.ready():
printans.result
```

ในตัวอย่างนี้สร้าง Anonymous function ชื่อ adder ด้วยคำสั่ง lambda โดยกำหนดพารามิเตอร์ 2 ตัว เมื่อกำหนดให้ตัวแปร ans เท่ากับ c[0].apply(adder,3,2) หาก worker 0 ทำงานเสร็จแล้วซึ่งตรวจสอบด้วยคำสั่ง ready() หากประมวลผลเสร็จแล้วจะคืนค่าเป็นผลลัพธ์จากการประมวลผลด้วยชื่อตัวแปร result ภายในอ็อบเจ็ค ans

9.7 การควบคุมให้ทุกโหนด (worker) ทำการอิมพอร์ตโมดูลด้วยคำสั่ง sync_import() โดยตัวอย่างต่อไปนี้แสดงการสั่งให้ทุกคลัสเตอร์คอมพิวเตอร์ทำการอิมพอร์ทโมดูล random และ time เขียนคำสั่งได้ ดังนี้

```
fromIpython.parallel import Client
c = Client()
view.c[:]
view.activate()
with c[:].sync_imports():
import random, time
```

9.8 การแบ่งข้อมูลไปยังแต่ละคลัสเตอร์คอมพิวเตอร์เท่า ๆ กัน โดยเครื่องสุดท้ายจะได้รับเฉพาะข้อมูลส่วนที่เหลือเท่านั้น ในตัวอย่างนี้สร้างตัวเลข 30 ชุดและกระจายไปยัง worker อย่างละเท่า ๆ กันด้วยคำสั่ง view.scatter('ans', range(30)) นอกจากนั้น ในการแสดงค่าข้อมูลที่อยู่บนคลัสเตอร์คอมพิวเตอร์แต่ละตัวจะใช้คำสั่ง view['ans'] โดยที่ตัวแปร ans เป็นผลลัพธ์ที่อยู่บนแต่ละคลัสเตอร์คอมพิวเตอร์

9.9 การรวมผลลัพธ์ด้วยคำสั่ง gather ดังนี้ view.gather('ans').result

9.10 การใช้ Magic %px คือ Parallel execution หรือการสั่งให้ view หรือ worker ทุกตัวทำคำสั่งที่ระบุลงไปด้วยคำสั่ง %px open("example.txt",'w').write("Hello World") โดยที่คำสั่ง %px เป็นการสั่งให้ทุก ๆ worker เขียนคำว่า "Hello World" ลงในไฟล์ชื่อ "example.txt" ที่อยู่บนโลคอลดิสค์ของแต่ละโหนด

การดำเนินการวิจัย

งานวิจัยนี้เป็นการวิจัยปฏิบัติการ โดยการเขียนโปรแกรมคอมพิวเตอร์เพื่อค้นหาวิธีการใหม่ในการเพิ่มประสิทธิภาพการสืบค้นข้อมูลจากฐานข้อมูลขนาดใหญ่ ด้วยการใช้เทคโนโลยีขั้นสูงเพื่อการประมวลผลแบบขนานบนคลัสเตอร์คอมพิวเตอร์ที่มีจำนวนโหนดแม่ข่ายที่แตกต่างกัน โดยใช้เทคนิคแม็พรีดิวซ์ มีขั้นตอนในการดำเนินการดังนี้

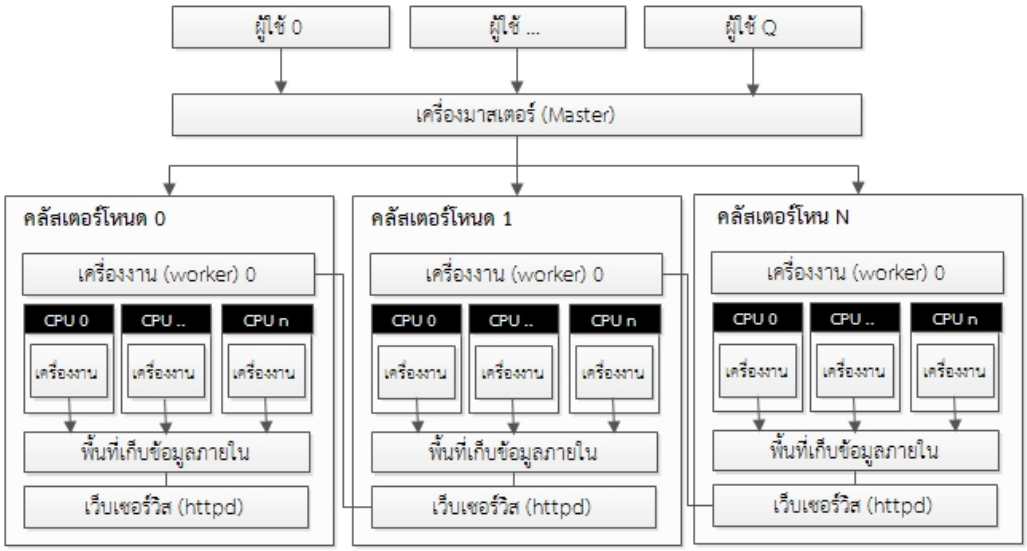
1. การพัฒนาคอมพิวเตอร์คลัสเตอร์ด้วยการเชื่อมต่อคอมพิวเตอร์คลัสเตอร์จำนวน 16 โหนดเข้ากับอุปกรณ์เน็ตเวิร์กด้วยความเร็ว 100 Mbps

2. การติดตั้งระบบปฏิบัติการลงบนคลัสเตอร์คอมพิวเตอร์ด้วยระบบปฏิบัติการ Linux Lite เวอร์ชั่น 2.8 และโปรแกรม IPython ลงบนคลัสเตอร์คอมพิวเตอร์ทั้งหมด โดยสร้างเป็นดีสโตร (Distro) เพื่อนำไปใช้เป็นระบบปฏิบัติการในรูปของไฟล์ .ISO ซึ่งสามารถบูตด้วย CD/DVD หรือ USB ได้

การแบ่งไฟล์ไปเก็บไว้ยังคลัสเตอร์คอมพิวเตอร์แต่ละเครื่อง โดยจะแบ่งไฟล์เป็นส่วน ๆ ตามจำนวนคลัสเตอร์คอมพิวเตอร์ เช่น หากใช้คลัสเตอร์คอมพิวเตอร์จำนวน 4 โหนดจะแบ่งฐานข้อมูลวิทยานิพนธ์เป็น 4 ส่วนในโครงสร้างของบิ๊กเทเบิ้ลและกระจายไฟล์ไปเก็บยังแต่ละโหนด ในทำนองเดียวกัน หากคลัสเตอร์ 16 โหนด จะทำการแบ่งข้อมูลออกเป็น 16 ส่วน แต่ละส่วนมีขนาดใกล้เคียงกันและกระจายไปยังแต่ละคลัสเตอร์คอมพิวเตอร์ (รูปที่ 2) ตัวอย่างรูปแบบของบิ๊กเทเบิ้ล ดังนี้ *“การเขียนโปรแกรมภาษาไพธอนด้วยตนเอง|จักรกฤษณ์ แสงแก้ว|chakkrit@msu.ac.th| อธิบายการเขียนโปรแกรมโดยใช้ภาษาไพธอน|”สำนักพิมพ์ สสท|2549|”* โดยที่แต่ละหลักจะคั่นด้วยเครื่องหมาย | และแต่ละแถวจะคั่นด้วยเครื่องหมายขึ้นต้นบรรทัดใหม่

รูปที่ 2 แสดงระบบคลัสเตอร์ที่ใช้ในงานวิจัยจำนวน 16 โหนด

3. การเขียนโปรแกรมภาษาไพธอนเพื่อรับค่าคำค้นและกระจายคำค้นไปยังแต่ละคลัสเตอร์คอมพิวเตอร์ด้วยแม็พฟังก์ชั่น เมื่อแต่ละโหนดส่งผลลัพธ์ของการสืบค้นกลับมาจะนำผลลัพธ์มารวมกันด้วยรีดิวซ์ฟังก์ชั่น โดยจะทำการจับเวลาที่ใช้ตั้งแต่เริ่มใช้ฟังก์ชั่นแม็พ และจนกระทั่งถึงการสิ้นสุดรีดิวซ์ฟังก์ชั่นของแต่ละรอบการสืบค้น ตัวอย่าง การส่งงานไปยังเครื่องงานหรือคลัสเตอร์คอมพิวเตอร์ (worker) ใช้คำสั่ง map โดยสร้างฟังก์ชั่น lambda ซึ่งเป็นฟังก์ชั่นที่สามารถสร้างในขณะที่โปรแกรมกำลังประมวลผลแบบเรียลไทม์ได้ด้วยคำสั่ง map(lambda x:'การเรียน' in x, [open('data/xab','r').read()]) ซึ่งตัวอย่างนี้หมายถึงการค้นหาคำว่า “การเรียน” ในไฟล์ที่อยู่บนแต่ละโหนด โดยจะคืนค่าเป็นจำนวนที่พบ หลังจากนั้นนำผลลัพธ์ของแต่ละโหนดมารวมกันด้วยฟังก์ชั่นรีดิวซ์ จะได้เป็นจำนวนที่พบทั้งหมดในทุกโหนด (รูปที่ 3)



รูปที่ 3 แสดงสถาปัตยกรรมแม็พรีดิวซ์ภายในคลัสเตอร์สำหรับการสืบค้นดาต้าเซ็ตวิทยานิพนธ์ TDC

4. การสืบค้นข้อมูลวิทยานิพนธ์จากฐานข้อมูล TDC

4.1 แนวคิดในการสืบค้นคือ การใช้แม็พรีดิวซ์ร่วมกับการประมวลผลแบบขนานซึ่งทำงานบนระบบคลัสเตอร์ โดยทำการจัดเก็บข้อมูลให้อยู่ในรูปแบบบิ๊กเทเบิ้ล ซึ่งเป็นโครงสร้างที่เก็บข้อมูลหลาย ๆ ฟิลด์ที่ประกอบกันเป็นเรคคอร์ดที่อยู่ในระบบฐานข้อมูลเชิงสัมพันธ์ ให้อยู่ในบรรทัดเดียวกันโดยคั่นด้วยเครื่องหมาย pipe (|) จากนั้นทำการแบ่งข้อมูลที่อยู่ในบิ๊กเทเบิ้ลเป็นส่วน ๆ และกระจายไปเก็บไว้ยังคอมพิวเตอร์โหนดที่อยู่ภายในคลัสเตอร์เขียนโปรแกรมด้วยแม็พรีดิวซ์ เพื่อส่งความต้องการไปยังคลัสเตอร์คอมพิวเตอร์แต่ละตัวซึ่งเรียกขั้นตอนนี้ว่าการ map เมื่อคลัสเตอร์คอมพิวเตอร์ได้รับคำสั่งจะทำการค้นข้อมูลเฉพาะในชิ้นส่วนที่ตนรับผิดชอบซึ่งมีขนาดเล็กไม่ใช่ฐานข้อมูลทั้งชุด จึงทำให้การค้นหาทำได้รวดเร็ว หลังจากนั้นคลัสเตอร์คอมพิวเตอร์จะทำการส่งผลลัพธ์ของการสืบค้นออกมาเป็นจำนวนที่พบ ในขั้นตอนนี้เรียกว่าการรีดิวซ์ (Reduce) สำหรับการเลือกใช้คำค้นเพื่อทดสอบความเร็วในการสืบค้น เลือกโดยการสุ่มเลือกจากคำค้นจากโปรแกรมตัดคำภาษาไทยโดยใช้หัวข้อวิทยานิพนธ์ทั้งหมดมาทำการตัดคำภาษาไทยจากนั้นทำให้เป็นเอกลักษณ์ไม่มีคำซ้ำกันและเลือกด้วยคำค้นด้วยวิธีการสุ่ม

4.2 วิธีการและขั้นตอนประกอบด้วย (1) การแปลงข้อมูลเชิงสัมพันธ์ให้อยู่ในโครงสร้างบิ๊กเทเบิ้ล (2) ทำการแบ่งข้อมูลออกเป็นส่วน ๆ แต่ละส่วนนำไปเก็บไว้บนคลัสเตอร์คอมพิวเตอร์ (3) กำหนดคำสั่งและให้คลัสเตอร์คอมพิวเตอร์ร่วมกันประมวลผลด้วย map(lambda x:'คำค้น' in x, [open('data/xab','r').read()]) ซึ่งตัวอย่างนี้ คือการค้นหาคำว่า "คำค้น" ในไฟล์ที่อยู่บนแต่ละโหนดเมื่อคลัสเตอร์คอมพิวเตอร์ประมวลผลเสร็จจะคืนค่าเป็นจำนวนที่พบ หลังจากนั้นนำผลลัพธ์ของแต่ละโหนดมารวมกันด้วยฟังก์ชั่นรีดิวซ์ จะได้เป็นจำนวนที่พบทั้งหมดในทุกโหนด

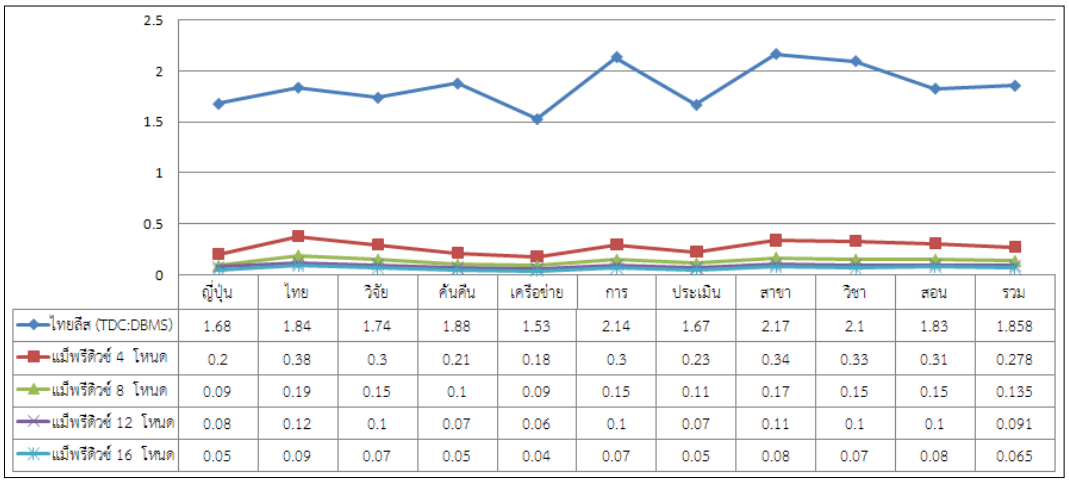
4.3 วิธีการบันทึกเวลาในการประมวลผลใช้การจับเวลาด้วยคำสั่ง %time ใน IPython notebook และเริ่มต้นโหลดหน้าเว็บที่ใส่คำค้นลงไปพร้อมกับยูอาร์แอลรีเควส (URL request) เมื่อสิ้นสุดการโหลดหน้าเว็บโปรแกรมจะรายงานเวลาที่ใช้ทั้งหมดในโหลดข้อมูล ซึ่งเป็นเวลาที่ใช้ประมวลผลจากนั้นทำการทดสอบซ้ำหลาย ๆ รอบ และนำผลลัพธ์เวลาที่ใช้ทำการคำนวณหาค่าเฉลี่ยของความเร็วในการสืบค้นของแต่ละคำ

สรุปผลการวิจัย

จากผลการเปรียบเทียบความเร็วในการสืบค้นข้อมูลโดยใช้ชุดข้อมูล (Data set) วิทยานิพนธ์จากฐานข้อมูล TDC จำนวนทั้งสิ้น 443,694 เรคคอร์ด เมื่อเปรียบเทียบกับคลัสเตอร์ที่มีจำนวนโหนดแม่ข่าย 4, 8, 12 และ 16 โหนดตามลำดับ พบว่าประสิทธิภาพด้านความเร็วในการประมวลผลสูงกว่าระบบฐานข้อมูลเชิงสัมพันธ์ที่ใช้คำสั่ง like ในภาษา SQL กล่าวคือ ระบบฐานข้อมูลเชิงสัมพันธ์ใช้เวลาประมวลผลเฉลี่ย 1.86 วินาที ส่วนการประมวลผลแบบขนานบนคลัสเตอร์ที่ใช้แม่ข่าย 4, 8, 12 และ 16 โหนด ใช้เวลาในการประมวลผลเฉลี่ยน้อยกว่า ด้วยความเร็ว 0.278, 0.135, 0.091 และ 0.065 วินาที คำนวณประสิทธิภาพด้านความเร็วการสืบค้นเท่ากับ 85.0% , 92.73%, 95.10% และ 96.50% ตามลำดับ (ตารางที่ 1 และรูปที่ 4)

ตารางที่ 1 แสดงเวลาการสืบค้นระหว่างฐานข้อมูลเชิงสัมพันธ์และแม็พรีดิวซ์คลัสเตอร์

คำค้น Keyword	ไทยลีส (TDC) DBMS	แม็พรีดิวซ์ จำนวน 4 โหนด	แม็พรีดิวซ์ จำนวน 8 โหนด	แม็พรีดิวซ์ จำนวน 16 โหนด	แม็พรีดิวซ์ จำนวน 16 โหนด
ญี่ปุ่น	1.68 s	0.20 s	0.09 s	0.08 s	0.05 s
ไทย	1.84 s	0.38 s	0.19 s	0.12 s	0.09 s
วิจัย	1.74 s	0.30 s	0.15 s	0.10 s	0.07 s
ค้นคืน	1.88 s	0.21 s	0.10 s	0.07 s	0.05 s
เครือข่าย	1.53 s	0.18 s	0.09 s	0.06 s	0.04 s
การ	2.14 s	0.30 s	0.15 s	0.10 s	0.07 s
ประเมิน	1.67 s	0.23 s	0.11 s	0.07 s	0.05 s
สาขา	2.17 s	0.34 s	0.17 s	0.11 s	0.08 s
วิชา	2.10 s	0.33 s	0.15 s	0.10 s	0.07 s
สอน	1.83 s	0.31 s	0.15 s	0.10 s	0.08 s



	ญี่ปุ่น	ไทย	วิจัย	ค้นคืน	เครือข่าย	การ	ประเมิน	สาขา	วิชา	สอน	รวม
ไทยลีส (TDC:DBMS)	1.68	1.84	1.74	1.88	1.53	2.14	1.67	2.17	2.1	1.83	1.858
แม็พรีดิวซ์ 4 โหนด	0.2	0.38	0.3	0.21	0.18	0.3	0.23	0.34	0.33	0.31	0.278
แม็พรีดิวซ์ 8 โหนด	0.09	0.19	0.15	0.1	0.09	0.15	0.11	0.17	0.15	0.15	0.135
แม็พรีดิวซ์ 12 โหนด	0.08	0.12	0.1	0.07	0.06	0.1	0.07	0.11	0.1	0.1	0.091
แม็พรีดิวซ์ 16 โหนด	0.05	0.09	0.07	0.05	0.04	0.07	0.05	0.08	0.07	0.08	0.065

รูปที่ 4 แสดงประสิทธิภาพด้านความเร็วในการสืบค้นในหน่วยวินาทีระหว่างเทคนิคแม็พรีดิวซ์และฐานข้อมูลเชิงสัมพันธ์ด้วยข้อมูลวิทยานิพนธ์ในฐานข้อมูล TDC ของระบบ ThaiLIS

ข้อเสนอแนะ

แนวคิดของการวิจัยชิ้นนี้เป็นการใช้เทคนิคแม็พรีดิวซ์เพื่อช่วยเพิ่มความเร็วการประมวลผล โดยการแบ่งข้อมูลให้มีขนาดเล็ก และกระจายข้อมูลไปยังคอมพิวเตอร์โหนดที่อยู่ภายในระบบคลัสเตอร์ ด้วยการแปลงข้อมูลที่อยู่ในลักษณะของตารางสัมพันธ์ (Relational database) ให้อยู่ในรูปของโครงสร้างบิ๊กเทเบิ้ล (Bigtable) ซึ่งข้อมูลแต่ละแถวจะถูกเก็บเอาไว้ภายในบรรทัดเดียวกันและคั่นด้วยเครื่องหมาย (|) จากนั้นพัฒนาโปรแกรมด้วยแนวคิดของ map-reduce เพื่อให้คอมพิวเตอร์โหนดทุกเครื่องทำงานพร้อม ๆ กัน ด้วยเทคนิคการกระจายงาน (map) และการรวมผลลัพธ์ของการประมวล (Reduce)

แนวทางในการนำงานวิจัยชิ้นนี้ไปใช้งาน ทำได้โดยการปรับโครงสร้างฐานข้อมูลเชิงสัมพันธ์ให้อยู่ในโครงสร้างบิ๊กเทิ้ล จากนั้นแบ่งข้อมูลเป็นชิ้นเล็ก ๆ และส่งไปเก็บไว้ในแต่ละคลัสเตอร์คอมพิวเตอร์และท้ายสุดคือพัฒนาโปรแกรมด้วยแนวคิดของแม็พรีดิวซ์ ด้วยคำสั่ง map และ reduce ข้อจำกัดคือ นักพัฒนาต้องมีความเข้าใจในการเขียนโปรแกรมเชิงฟังก์ชั่น (Functional programming) ซึ่งแม็พรีดิวซ์เป็นความรู้พื้นฐานในเรื่องดังกล่าว

แม้ว่าการวิจัยครั้งนี้จะเป็นการนำเครื่องคอมพิวเตอร์ส่วนบุคคลมาใช้ประมวลผลแบบขนานโดยทำงานภายในระบบคลัสเตอร์ แต่มีข้อจำกัดในเรื่องพื้นที่ห้องเก็บอุปกรณ์คอมพิวเตอร์อีกทั้งใช้พลังงานไฟฟ้าค่อนข้างสูงเพราะใช้คอมพิวเตอร์หลายเครื่องร่วมกันประมวลผล ในการวิจัยครั้งต่อไปควรศึกษาการนำคอมพิวเตอร์บอร์ดเดียว (Single board computer) หรือ Embedded Linux อาทิบอร์ด Raspberry Pi หรือ FriendlyARM Nano Pi ซึ่งเป็นคอมพิวเตอร์ขนาดเท่ามือถือสมาร์ทโฟนหลาย ๆ ชุดมาประมวลผลแบบขนานแทนคอมพิวเตอร์ส่วนบุคคลผ่านระบบคลัสเตอร์และเปรียบเทียบประสิทธิภาพของการสืบค้นต่อไป

กิตติกรรมประกาศ

ผู้วิจัยขอขอบคุณสาขาวิชาสารสนเทศศาสตร์ คณะวิทยาการสารสนเทศ มหาวิทยาลัยมหาสารคาม ที่สนับสนุนทุนวิจัยและคอมพิวเตอร์เพื่อใช้งานในการสร้างระบบคลัสเตอร์ ขอขอบคุณโครงการพัฒนาเครือข่ายระบบห้องสมุดในประเทศไทย (ThaiLIS) สำหรับชุดข้อมูลวิทยานิพนธ์ที่ใช้ในการวิจัย

รายการอ้างอิง

Bernard, J. (2013). Running scientific code using IPython and SciPy. *Linux J*, *228*, Article no. 3. Retrieved from http://dl.acm.org/citation.cfm?id=2492105&preflayout=tabs

Chang, F. *et al*. (2008). Bigtable: A distributed storage system for structured data. *ACM Trans Comp Syst*, *26*(2), Article 4, 26 pages.

Crampton, J. and Khambhammettu, H. (2008). On delegation and workflow execution models. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC'08),* pp. 2137-2144. ACM, New York, USA.

Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Commun ACM*, *51*(1), 107-103.

Fu, H., Pophale, S., Venkata, G.M. and Yu, W. (2016). DISP: Optimizations towards scalable MPI startup. In *Proceedings of the First Workshop on Optimization of Communication in HPC (COM-HPC'16),* pp. 53-62. IEEE Press, Piscataway, NJ, USA.

Graham, R. et al. (2006). Open MPI: A high-performance heterogeneous MPI. In *Proceedings of the 5th International Workshop on Algorithms Models and Tools for Parallel Computing on Heterogeneous Networks*. Barcelona, Spain.

Martens, J.D. (2003). A portable thread API for teaching operating systems. *J Comp SciColl*, *18*(6), 119-125.

Office of Information Technology Administration for Educational Development. (2017). ThaiLIS-Thai Library Integrated System. Retrieved from http://tdc.thailis.or.th/tdc

Rezkalla, M. M. N., Heussen, K., Marinelli, M., Hu, J. and Bindner, H. W. (2015). Identification of requirements for distribution management systems in the smart grid context. In *Proceedings of the 50th International Universities Power Engineering Conference (UPEC 2015)*, p. 6, IEEE. DOI: 10.1109/UPEC.2015.7339932

Samimi, H. (2013). Introduction to the Python programming language. *J Comp SciColl*, *29*(1), 8-9.

Sobie, R. *et al*. (2013). HTC scientific computing in a distributed cloud environment. In *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing (ScienceCloud'13)*, pp. 45-52. ACM, New York, USA.

Vestal, D., Thoma, T., Harris, R. and Schmitt, J. (2007). High-performance, low-cost grid computing through community cooperation. In *Proceedings of the 45th Annual Southeast Regional Conference (ACM-SE 45),* pp. 341-346. ACM, New York, USA.