วิธีการเชิงพันธุกรรมแบบขนานสำหรับคำถามที่เหมาะสมที่สุด แบบกระจายของการ JOIN จำนวนมาก

โดย

นายพิศาล สุขขี

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

ภาควิชาคอมพิวเตอร์

บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร

ปีการศึกษา 2552

วิธีการเชิงพันธุกรรมแบบขนานสำหรับคำถามที่เหมาะสมที่สุด แบบกระจายของการ JOIN จำนวนมาก

โดย
นายพิศาล สุขขี

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์
ภาควิชาคอมพิวเตอร์
บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร
ปีการศึกษา 2552

PARALLEL GENETIC ALGORITHMS FOR DISTRIBUTED QUERY OPTIMIZATION OF LARGE JOIN

By

Phisan Shukkhi

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

Department of Computing

Graduate School

SILPAKORN UNIVERSITY

2009

บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร อนุมัติให้วิทยานิพนธ์เรื่อง " วิธีการเชิงพันธุกรรมแบบขนานสำหรับคำถามที่เหมาะสมที่สุดแบบกระจายของการ JOIN จำนวนมาก " เสนอโดย นายพิศาล สุขขี เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

……..
(รองศาสตราจารย์ ดร.ศิริชัย ชินะตังกูร)
คณบดีบัณฑิตวิทยาลัย
วันที่.....เดือน..... พ.ศ.....

อาจารย์ที่ปรึกษาวิทยานิพนธ์
อาจารย์ ดร.สุนีย์ พงษ์พินิจภิญโญ

คณะกรรมการตรวจสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์.ดร.จันทนา ผ่องเพ็ญศรี)
...../...../.....

..... กรรมการ
(รองศาสตราจารย์.ดร.วรา วราวิทย์)
...../...../.....

..... กรรมการ
(อาจารย์ ดร.สุนีย์ พงษ์พินิจภิญโญ)
...../...../.....

48307307 : สาขาวิชาวิทยาการคอมพิวเตอร์
คำสำคัญ : ฐานข้อมูลแบบกระจาย/วิธีการเชิงพันธุกรรมแบบขนาน/คำถามที่เหมาะสมที่สุดแบบกระจาย/การคำนวณแบบขนาน/การประมวลผลแบบสอบถามข้อมูล
พิศาล สุขขี : วิธีการเชิงพันธุกรรมแบบขนานสำหรับคำถามที่เหมาะสมที่สุดแบบกระจายของการ JOIN จำนวนมาก. อาจารย์ที่ปรึกษาวิทยานิพนธ์ : อ.ดร.สุนีย์ พงษ์พินิจภิญโญ. 98 หน้า.

ปัญหาเรื่องการประมวลผลสอบถาม (Query Processing) เป็นปัญหาที่สำคัญปัญหาหนึ่งในฐานข้อมูลแบบกระจาย (Distributed Database) ปัญหานี้คือการเลือกคำตอบที่ดีที่สุดในเวลาที่ดีที่สุดโดยเฉพาะอย่างยิ่งเมื่อมีรีเลชันจำนวนมากในคำสั่งสอบถามถูกเรียกใช้ในฐานข้อมูลแบบกระจาย ปัญหานี้จะส่งผลกระทบโดยตรงต่อประสิทธิภาพของระบบการจัดการฐานข้อมูลแบบกระจาย (Distributed Database Management Systems, DDBMS) งานวิจัยนี้จึงนำเสนอวิธีพัฒนาประสิทธิภาพกระบวนการประมวลผลข้อมูลในระบบฐานข้อมูลแบบกระจายให้สามารถทำการคัดเลือกคิวรีเอ็กซิคิวทชันแพลนให้ได้อย่างรวดเร็วและได้คุณภาพที่ดียิ่งขึ้น โดยใช้เทคนิคการค้นหาแบบ Island-based Parallel Genetic Algorithm ซึ่งเป็นอีกเทคนิคหนึ่งสำหรับวิธีการเชิงพันธุกรรมแบบขนาน ซึ่งจากผลการวิจัยพบว่า เทคนิคการค้นหาโดยใช้วิธีเชิงพันธุกรรมแบบขนานสามารถพัฒนาการประมวลผลสอบถามแบบกระจายไม่เพียงสามารถลดเวลาในการค้นหาได้อย่างมีประสิทธิภาพแต่ยังสามารถให้คำตอบที่เป็นคำตอบที่ดีที่สุดได้อย่างรวดเร็ว

ภาควิชาคอมพิวเตอร์ บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร ปีการศึกษา 2552

ลายมือชื่อนักศึกษา.....

ลายมือชื่ออาจารย์ที่ปรึกษาวิทยานิพนธ์ .....

48307307 : MAJOR : COMPUTER SCIENCE
KEY WORDS : DISTRIBUTED DATABASE/PARARELL GENETIC ALGORITHM/DISTRIBUTED QUERY OPTIMIZATION/PARARELL COMPUTING/QUERY PROCESSING
PHISAN SHUKKHI : PARALLEL GENETIC ALGORITHMS FOR DISTRIBUTED QUERY OPTIMIZATION OF LARGE JOIN. THESIS ADVISOR : SUNEE PONGPINIGPINYO, Ph.D. 98 pp.

The query processing problem is an important problem in distributed databases. This problem is how to select the optimal solution in the optimal time especially when a large number of relations from a query has been accessed in the distributed databases. It will impact directly the overall performance of a distributed database system if the distributed query processing cannot optimize the query in a reasonable period of time due to their time complexity. So, in this paper, we present an efficient development method of distributed query processing that can choose the best query execution plan in efficient time by using parallel searching technique namely Island-based Parallel Genetic Algorithm. The experimental results show that the parallel searching technique can enhance the development method of distributed query processing not only reduce the search time efficiently but also give the fast convergence to the optimal solution.

Department of Computing Graduate School, Silpakorn University Academic Year 2009
Student's signature .....
Thesis Advisor's signature .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จได้ด้วยดี โดยได้รับความอนุเคราะห์จากอาจารย์หลายๆ ท่าน ผู้วิจัยขอขอบพระคุณอาจารย์ ดร.สุนีย์ พงษ์พินิจภิญโญ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้ให้คำแนะนำและให้คำปรึกษามาโดยตลอด พร้อมทั้งขอขอบพระคุณ รองศาสตราจารย์ ดร.จันทนา ผ่องเพ็ญศรี ที่กรุณาเป็นประธานกรรมการในการสอบและรองศาสตราจารย์ ดร.วรา วราวิทย์ ที่ได้กรุณาเป็นผู้ทรงคุณวุฒิ ที่ให้คำแนะนำตรวจสอบ คำชี้แนะและมุมมองที่ดีที่ทำให้การดำเนินงานวิจัยประสบผลสำเร็จและทำให้วิทยานิพนธ์มีความสมบูรณ์ยิ่งขึ้น

ผู้วิจัยขอขอบพระคุณคณาจารย์ ภาควิชาคอมพิวเตอร์ทุกท่านที่ได้ให้ความรู้และความช่วยเหลือ พร้อมทั้งให้กำลังใจแก่ผู้วิจัยตลอดระยะเวลาในการศึกษา ขอขอบคุณ คุณประวิม เหลืองสมานกุล ที่ให้การช่วยเหลือและอำนวยความสะดวกแก่ผู้วิจัยตลอดมา และขอขอบคุณพี่ๆ น้องๆ ทุกคน ที่เป็นกำลังใจ และช่วยเหลือ ทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลงด้วยดี

งานวิจัยนี้ได้รับทุนสนับสนุนส่วนหนึ่งจาก โครงการทุนวิจัยมหาบัณฑิต สำนักงานกองทุนสนับสนุนการวิจัย (สกว.) สาขาวิทยาศาสตร์และเทคโนโลยี ผู้วิจัยจึงขอขอบคุณ สำนักงานกองทุนสนับสนุนการวิจัย มา ณ โอกาสนี้

สุดท้ายนี้คุณค่าและคุณประโยชน์ของวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบเพื่อตอบแทนพระคุณของคุณพ่อแบน สุขขี ที่ให้การสนับสนุนและเป็นกำลังใจให้ตลอดมา และคุณพ่อพิสิษฐ์ ประมวลและคุณแม่มัลนิกา ประมวล ที่ให้การสนับสนุนและส่งเสริมการทำงานของผู้วิจัยในทุกๆ ด้าน

สารบัญ

บทที่ หน้า

สารบัญภาพ

ภาพที่ หน้า

บทที่ 1
บทนำ

ความเป็นมาและความสำคัญของปัญหา

การประมวลผลสอบถาม (Query Processing) เป็นหน้าที่หนึ่งของระบบการจัดการฐานข้อมูล (DBMS) โดยระบบการจัดการฐานข้อมูล จะทำการประมวลผลคำสั่งสอบถามข้อมูล (Query) ที่เกิดจากภาษาสอบถาม (Query Language) ได้แก่ ภาษา SQL (Structure Query Language) เพื่อหาคำตอบที่ดีที่สุดและถูกต้องตรงกับความต้องการของผู้ใช้ ได้อย่างเหมาะสม รวดเร็วและเสียค่าใช้จ่ายในการดำเนินการน้อยที่สุด

การประมวลผลสอบถาม เป็นกระบวนการในการเลือกแผน หรือวิธีการดำเนินการที่เหมาะสมในการสอบถามข้อมูลในฐานข้อมูล โดยระบบจัดการฐานข้อมูล จะมีตัวประมวลผลที่เรียกว่า ตัวประมวลผลสอบถาม ทำหน้าที่ในการเลือกวิธีการสำหรับดำเนินการที่เหมาะสมในการเข้าถึงข้อมูลเพื่อให้ได้คำตอบที่ดีที่สุดและถูกต้องตรงกับความต้องการของผู้ใช้งานมากที่สุด

โดยการดำเนินการหลักของการประมวลผลสอบถาม คือการแปลง High-level Query (Relational Calculus) หรือคำสั่งสอบถามข้อมูลระดับสูงที่ผู้ใช้งานทั่วไปสามารถเข้าใจได้ ให้เป็น Low-level Query (Relation Algebra) ที่ให้ผลลัพธ์สำหรับการสอบถามข้อมูลเหมือนกัน การแปลงจาก High-level Query ให้เป็น Low-level Query นั้นจะต้องเป็นไปอย่างถูกต้อง และมีประสิทธิภาพสำหรับในเรื่องของความถูกต้องนั้น การสอบถามในระดับ Low-level จะต้องให้ผลลัพธ์เช่นเดียวกับผลของการสอบถามข้อมูล ในระดับ High-level Query ต้นฉบับ นั่นหมายความว่า ทั้ง High-level Query และ Low-level Query จะต้องให้ผลลัพธ์สำหรับการสอบถามเดียวกัน

โดยที่ Low-level Query หรือ Relation Algebra จะแสดงถึงวิธีในการดำเนินการสำหรับการเข้าถึงเข้ามูล (Execution Strategy) เพื่อให้ได้ผลลัพธ์สำหรับการสอบถามข้อมูลนั้น ซึ่งวิธีในการดำเนินการเพื่อเข้าถึงข้อมูล คือการจัดลำดับ (Order) สำหรับการดำเนินการกับแต่ละรีเลชั่น (Relations) เพื่อให้ได้ผลลัพธ์สำหรับการสอบถามข้อมูล ซึ่งอาจกล่าวได้ว่าการจัดลำดับ

ในการดำเนินการระหว่างรีเลชั่นที่แตกต่างกัน ทำให้เกิดวิธีการดำเนินการเพื่อเข้าถึงข้อมูลที่มีความแตกต่างกันออกไป

แต่ปัญหาที่สำคัญอย่างยิ่งที่เกิดขึ้นคือ แต่ละ Relational Calculus สามารถแปลงให้เป็น Relation Algebra ได้อย่างหลากหลาย นั่นหมายถึงสำหรับหนึ่งคำสั่งสอบถาม จะมีวิธีการที่เป็นไปได้ในการดำเนินการเพื่อให้ได้คำตอบ (Execution Strategy) หลายวิธี ซึ่งแต่ละวิธีก็จะให้ประสิทธิภาพในการดำเนินการเพื่อให้ได้ผลลัพธ์ที่แตกต่างกัน หรือกล่าวได้ว่าสำหรับหนึ่งคำสั่งสอบถาม สามารถจัดลำดับในการดำเนินการระหว่างรีเลชั่นได้อย่างหลากหลาย ซึ่งการจัดลำดับในการดำเนินการระหว่างรีเลชั่นที่แตกต่างกัน จะให้ผลลัพธ์สำหรับการสอบถามข้อมูลเหมือนกัน แต่จะต้องสูญเสียค่าใช้จ่ายในการดำเนินการเพื่อให้ได้ผลลัพธ์ที่แตกต่างกันกันออกไป การค้นหาชุดของลำดับในการดำเนินการกับรีเลชั่น ที่ให้ค่าใช้จ่ายในการดำเนินการน้อยที่สุด จากชุดของลำดับที่เป็นไปได้อย่างหลากหลายภายในเวลาที่จำกัด จึงเป็นเรื่องที่ยากด้วยเช่นกัน

ในกรณีที่คำสั่งสอบถามมีการดำเนินการ JOIN ข้อมูลระหว่างรีเลชั่นจำนวนมากหรือตั้งแต่ 10 รีเลชั่นขึ้นไปเข้าด้วยกัน เพื่อให้ได้ผลลัพธ์ของคำสั่งสอบถามข้อมูลที่ผู้ใช้งานต้องการ ซึ่งในกรณีนี้จะทำให้เกิดจำนวนชุดของลำดับสำหรับการดำเนินการกับแต่ละรีเลชั่นที่เป็นไปได้อย่างหลากหลายโดยจะเพิ่มขึ้นเป็นจำนวนแฟคทอเรียล (Factorial) ของจำนวนรีเลชั่นที่เพิ่มขึ้น ดังนั้นการหาชุดลำดับของการดำเนินการระหว่างรีเลชั่นที่ให้ค่าใช้จ่ายน้อยที่สุด จากชุดลำดับที่เป็นไปได้ทั้งหมดภายในเวลาที่จำกัดจึงเป็นเรื่องที่ยากยิ่ง โดยปัญหาในการเลือกชุดลำดับสำหรับการดำเนินการระหว่างรีเลชั่นที่ดีที่สุดในกรณีที่คำสั่งสอบถามข้อมูลมีความซับซ้อนมากนี้เรียกว่า “**ปัญหา Optimization of Large Join Query**” ซึ่งเป็นปัญหาที่ไม่สามารถหาคำตอบที่ดีที่สุดได้ในเวลาที่จำกัด ดังนั้นจึงสามารถกล่าวได้ว่า ปัญหานี้เป็นปัญหาที่จัดอยู่ในประเภท NP-Complete Combinatorial Optimization (Swami and Gupta 1988)

สำหรับกระบวนการที่ DBMS ใช้สำหรับค้นหาวิธีการในการดำเนินการ (Execution Strategy) เพื่อให้ได้คำตอบของการสอบถามข้อมูล หรือการค้นหาชุดลำดับสำหรับการดำเนินการที่ให้ประสิทธิภาพในการเข้าถึงข้อมูลได้อย่างมีประสิทธิภาพมากที่สุดจากชุดลำดับที่เป็นไปได้ทั้งหมด นั้นคือกระบวนการ Query Optimization ซึ่งโดยในระดับแนวความคิด กระบวนการ Query Optimization จะต้องสามารถที่จะคัดเลือกชุดลำดับที่ดีที่สุดจากทั้งหมดที่เป็นไปได้จาก Relational

Calculus เดียวกัน นั่นหมายถึงผลลัพธ์ของกระบวนการ Query Optimization คือวิธีการที่ใช้สำหรับประมวลผลข้อมูลเพื่อให้ได้คำตอบสำหรับการสอบถามข้อมูล ที่มีประสิทธิภาพมากที่สุด ซึ่งจะเห็นได้ว่าผลลัพธ์ และประสิทธิภาพการทำงานของกระบวนการ Query Optimization จะส่งผลต่อประสิทธิภาพของระบบ DBMS เป็นอย่างยิ่ง

วิธีการที่กระบวนการ Query Optimization โดยทั่วไปใช้สำหรับค้นหาชุดลำดับสำหรับการดำเนินการกับแต่ละรีเลชั่นหรือ วิธีในการดำเนินการ (Execution Strategy) ที่ดีที่สุดโดยทั่วไปนั้นจะเป็นแบบ Exhaustive search ซึ่งจะเป็นวิธีการค้นหาแบบละเอียด โดยผลลัพธ์ที่ได้จะได้จากการเปรียบเทียบเพื่อหาวิธีดำเนินการที่ดีที่สุดจากทุกๆ วิธีที่สามารถเป็นไปได้ ซึ่งข้อดีสำหรับวิธีการนี้คือผลลัพธ์ที่ได้จะเป็นวิธีการในการดำเนินการที่ให้ค่า Minimum Cost สำหรับการดำเนินการเพื่อให้ได้คำตอบจริงๆ แต่ข้อเสียที่ร้ายแรงสำหรับวิธีการค้นหาวิธีการดำเนินการที่ดีที่สุดด้วย Exhaustive search นั้นจะเกิดขึ้นในกรณีที่แบบสอบถามข้อมูล (Query) มีการดึงข้อมูลจากรีเลชั่นจำนวนมาก ในกรณีนี้จะทำให้เกิดวิธีการดำเนินการที่เป็นไปได้จำนวนมากหรือจำนวนมหาศาล ซึ่งจะต้องสูญเสียเวลาอย่างมากในการเปรียบเทียบอย่างละเอียดเพื่อค้นหาวิธีการดำเนินการที่ดีที่สุดภายใน Search space ขนาดใหญ่ที่เกิดขึ้น ซึ่งเวลาที่ต้องสูญเสียไปในขั้นตอนของกระบวนการ Query Optimization นั้นก็จะส่งผลเสียโดยตรงต่อประสิทธิภาพการทำงานของ DBMS อย่างมากเช่นกัน

เพื่อที่จะเป็นการหลีกเลี่ยงค่าใช้จ่ายที่สูงมากซึ่งเกิดจากการค้นหาแบบ Exhaustive search ในกรณีที่คำสั่งสอบถามข้อมูลเป็นแบบ Large Join Query จึงได้มีการนำเอาเทคนิคการค้นหาแบบ Randomized เข้ามาทำการค้นหาแทน เช่น Iterative Improvement หรือ Simulated Annealing (SA) ซึ่งจะทำให้การค้นหาวิธีดำเนินการที่เป็นไปได้ภายใน Search space ได้ผลลัพธ์เป็นวิธีดำเนินการที่เหมาะสมที่สุดภายในเวลาที่รวดเร็วขึ้น

แต่อย่างไรก็ตาม ผลลัพธ์ที่ได้จากวิธีการค้นหาทั้ง Iterative Improvement หรือ Simulated Annealing นั้นก็ไม่ได้รับประกันว่า ผลลัพธ์ที่ได้นั้นคือ Query Execution Plan (QEP) ที่จะให้คำตอบที่มีค่าที่ดีที่สุดจาก QEPs ที่เป็นไปได้ทั้งหมด เพียงแต่เป็น QEP ที่มีค่าเหมาะสมที่สุดเท่าที่จะหาได้ในเวลาเหมาะสม

โดยสำหรับงานวิจัยชิ้นนี้ ผู้วิจัยได้มุ่งประเด็นการศึกษาไปที่การหาวิธีสำหรับแก้ปัญหาเรื่อง “Distributed Query Optimization of Large Join” ซึ่งจะเป็นการค้นหาวิธีที่จะทำให้สามารถที่จะ

ค้นหาวิธีการในการดำเนินการที่เหมาะสมที่สุด ภายในเวลาที่รวดเร็วในกรณีที่มีการดำเนินการ Join ข้อมูลระหว่างรีเลชั่นจำนวนมากคือตั้งแต่ 10 – 40 รีเลชั่น ภายใต้สภาวะแวดล้อมของระบบฐานข้อมูลแบบกระจาย (Distributed Database) ซึ่งมีปัจจัยต่างๆ ที่ซับซ้อนกว่าการแก้ปัญหาในเรื่อง Query Optimization ใน Centralized Database เช่น ขนาด Search space ที่มีขนาดใหญ่กว่า, การจัดลำดับสำหรับการดำเนินการกับแต่และรีเลชั่นที่มีความซับซ้อนกว่า และวิธีการสำหรับคำนวณเพื่อทำนายค่าใช้จ่าย (Cost) สำหรับการดำเนินการที่จะเกิดขึ้นที่มีความซับซ้อนกว่า, การคำนึงถึงขนาดของข้อมูลที่จะต้องมีการรับ-ส่งข้อมูลผ่านระบบ Network และมีปัจจัยสำหรับการคำนวณมากกว่าเป็นต้น

อีกเหตุผลที่สำคัญอย่างยิ่งคือ มีแนวโน้มว่าปริมาณข้อมูลในปัจจุบันจะมีขนาดเพิ่มมากขึ้นเรื่อยๆ และขนาดของระบบฐานข้อมูลแบบกระจาย (Distributed Database) จะมีการขยายตัวเพิ่มมากขึ้น ดังนั้นทำให้มีโอกาสเกิดการสอบถามข้อมูลในลักษณะแบบ Large Join Query เพิ่มมากขึ้นเรื่อยๆ หรือดังที่ Dong and Liang (2007) ได้นำเสนอไว้ว่า สำหรับ Relational Databases Application สมัยใหม่ ได้มีการใช้วิธีการ Join Query ที่ซับซ้อนมากขึ้น โดยที่มีระบบ Database ขนาดใหญ่เป็นส่วนหนึ่งของ Application ยกตัวอย่างเช่น ระบบ Decision support ระบบ Data warehouse คือศูนย์กลางข้อมูลของระบบ Decision support และใช้ Relational Model สำหรับการจัดเก็บ และจัดการบริหารข้อมูลต่างๆ ซึ่งจะต้องมีการรับ-ส่ง และแลกเปลี่ยนข้อมูลกันระหว่าง Database ต่างๆ ซึ่งตรงจุดนี้นั้นจะมีการใช้การดำเนินการแบบ Large Join Query เกิดขึ้น แต่ในขณะเดียวกันปัญหาเรื่อง Distributed Query Optimization of Large Join บนระบบฐานข้อมูลแบบกระจาย ก็ยังเป็นปัญหาและอุปสรรคสำคัญที่ทำให้กระบวนการ Query Optimization เป็นไปได้อย่างล่าช้า หรือไม่สามารถรอคอยคำตอบจากกระบวนการนี้ได้ซึ่งผลเสียจากระบวนการนี้เองจะส่งผลกระทบในด้านลบต่อระบบ Distributed Database ทั้งระบบเช่นกัน

ซึ่งในงานวิจัยนี้ผู้วิจัยได้เล็งเห็นว่า การพัฒนาของระบบคอมพิวเตอร์ ไม่ว่าจะเป็นด้าน Hardware หรือ Software ในปัจจุบันสามารถที่จะรองรับการคำนวณที่มีความซับซ้อนมากได้อย่างมีประสิทธิภาพ รวมทั้งเทคโนโลยีการคำนวณแบบขนาน (Parallel Computing) ซึ่งสามารถแก้ไขปัญหาการคำนวณที่มีความซับซ้อนต่างๆ ให้สามารถหาคำตอบได้อย่างรวดเร็วมากขึ้น ซึ่งปัญหาในเรื่อง “Distributed Query Optimization of Large Join” นี้ก็เป็นปัญหาในเรื่องการคำนวณที่มีความ

ซับซ้อนทำให้การหาคำตอบที่สุดที่สุดมีความล่าช้าจนถึงกับไม่สามารถรอคำตอบได้ ผู้วิจัยจึงได้มีความพยายามที่จะนำเอาการคำนวณแบบขนาน (Parallel Computing) เข้ามาแก้ปัญหานี้ด้วยเช่นกัน

โดยการแก้ปัญหาเรื่อง "Distributed Query Optimization of Large Join" นั้นผู้วิจัยได้มุ่งประเด็นไปที่การพัฒนาประสิทธิภาพการทำงานของ "Optimizer" ซึ่งเป็นองค์ประกอบหนึ่งภายในระบบจัดการฐานข้อมูลแบบกระจาย (DDBMS) ซึ่งจะทำหน้าที่ในการดำเนินกระบวนการ Query Optimization โดยกระบวนการ Query Optimization จะมีองค์ประกอบย่อยที่สำคัญ 3 ส่วนคือ Search space, Cost model และ Search strategy (Ozsu and Valduriez 1991)

ผู้วิจัยได้เลือกที่จะทำการพัฒนา Optimizer ให้สามารถทำงานได้อย่างมีประสิทธิภาพมากยิ่งขึ้นในกรณีที่เกิด Large Join Query ด้วยการพัฒนา Search strategy ให้สามารถทำการค้นหาวิธีดำเนินการที่เหมาะสมที่สุดในเวลาที่รวดเร็วที่สุด โดยผู้วิจัยได้เลือกวิธีการค้นหา คือ Island Base Parallel Genetic Algorithm ซึ่งเป็นวิธีการค้นหาแบบขนานมาใช้สำหรับการค้นหาวิธีดำเนินการที่เหมาะสมที่สุด ซึ่งวิธีการพัฒนาและทดลองนั้นผู้วิจัยได้ทำการจำลอง (Simulation) กระบวนการทำงานของกระบวนการ Query Optimization ขึ้นมาเพื่อความสะดวกต่อการควบคุมปัจจัยต่างๆ ที่มีผลต่อการทดลอง และเพื่อความสะดวกต่อการวัดผลของประสิทธิภาพที่ได้จากการพัฒนาระบบ และการดำเนินการต่างๆ ภายในการทดลองจะอยู่ภายใต้สมมติฐานของงานวิจัยที่ผู้วิจัยได้ตั้งและกำหนดขึ้น

วัตถุประสงค์การศึกษา

1. เพื่อสร้าง Distributed Query Optimizer ต้นแบบที่ใช้วิธีค้นหาแบบ Island Based Parallel Genetic Algorithm
2. เพื่อพัฒนากระบวนการ Query Optimization ที่สามารถรองรับการทำงานแบบ Parallel Computing
3. เพื่อบ่งชี้ประสิทธิภาพของ Island Based Parallel Genetic Algorithm ว่าสามารถแก้ปัญหาเรื่อง Distributed Query Optimization of Large Join ได้เท่าใด และอย่างไร
4. เพื่อศึกษาถึงปัจจัยที่มีผลต่อประสิทธิภาพการทำงานของ Island Based Parallel Genetic Algorithm ภายใต้แบบจำลองของ Distributed Query Optimizer ที่ได้สร้างขึ้น

สมมติฐานของการศึกษา

งานวิจัยนี้ได้มุ่งเน้นที่จะศึกษาถึงการนำ Island Base Parallel Genetic Algorithms มาใช้ในการแก้ปัญหา Distributed Query Optimization of Large Join บนฐานข้อมูลแบบกระจาย ดังนั้นการทดลองต่างๆ ที่เกิดขึ้นภายในงานวิจัย จึงเป็นไปเพื่อการวัดประสิทธิภาพของ Island Base Parallel Genetic Algorithms เป็นหลัก และกระบวนการที่งานวิจัยนี้ได้ทำการศึกษา เป็นกระบวนการที่เกิดขึ้นภายใน Distributed Query Optimizer ซึ่งเป็นองค์ประกอบหนึ่งของ Distributed Database Management System (DDBMS)

ซึ่งในการทำงานจริงของระบบ DDBMS ประสิทธิภาพในการทำงานของ Distributed Query Optimizer ไม่ได้ขึ้นอยู่กับประสิทธิภาพของ Search Algorithm เพียงอย่างเดียว ดังนั้นสำหรับขั้นตอนของการศึกษา ผู้วิจัยได้เลือกที่จะดำเนินการวิจัยโดยการจำลองการทำงานของ Distributed Query Optimizer เพื่อให้ผู้วิจัยสามารถเลือกและควบคุมปัจจัยแวดล้อมต่างๆ ที่มีผลต่อการทำงานของ Island Base Parallel Genetic Algorithm เพียงอย่างเดียว เพื่อให้ผลที่ได้จากการทดลอง เป็นสิ่งที่บ่งชี้ถึงประสิทธิภาพการทำงานของ Search Algorithm ที่ผู้วิจัยได้ทำการศึกษา

เพื่อเป็นการลดความซับซ้อนในการจำลองข้อมูลและรีเลชั่นต่างๆ ที่ถูกสร้างขึ้นใน Database และ Data Dictionary จำลอง สำหรับถูกใช้เป็นค่าตั้งต้นในการทดลองเพื่อวัดประสิทธิภาพของ Parallel Genetic Algorithms นั้นข้อมูลต่างๆ ที่ถูกจำลอง จะถูกสร้างขึ้นภายใต้ข้อกำหนดตามเงื่อนไขดังนี้

1. สำหรับการ JOIN ที่เกิดขึ้นแต่ละครั้งรีเลชั่นที่ถูกดำเนินการนั้น แต่ละรีเลชั่นจะถูกกำหนดให้อยู่กันคนละ Site และเป็นข้อมูลที่ไม่ซ้ำซ้อนกัน

2. สำหรับข้อมูลของแต่ละ Attribute นั้นจะมีลักษณะการกระจายตัวของข้อมูลแบบทั่วถึง (Uniform) และมีความเป็นอิสระ (Independent) จาก Attribute อื่นๆ

3. ผลของการ JOIN ของรีเลชั่นคู่ใดๆ จะก่อให้เกิดรีเลชั่นใหม่สำหรับเก็บผลลัพธ์ชั่วคราว (Intermediate Relation) ซึ่งขนาดของผลลัพธ์ชั่วคราวที่เกิดขึ้นนั้น สามารถคำนวณโดยการประมาณค่าโดยใช้ปัจจัย (Factor) ภายใน Data Dictionary

4. สำหรับแต่ละ Site นั้นจะต้องทำการ SELECT-PROJECT กับรีเลชั่นก่อนที่จะทำการส่งข้อมูลออกมา เพื่อเป็นการลดขนาดในการรับส่งข้อมูล

5. สำหรับการคำนวณค่า Cost เพื่อทำนายว่า วิธีการในการดำเนินการใดให้ประสิทธิภาพที่ดีที่สุดสำหรับการนำไป Execute นั้นจะมีปัจจัยหนึ่งที่สำคัญสำหรับการคำนวณ คือ อัตราขนส่งข้อมูลระหว่าง Site ต่างๆ ผ่านระบบเครือข่าย (Network Data Transfer Rate) ซึ่งในการทดลองจะทำการกำหนดให้เป็นปัจจัยที่มีการผันแปรอยู่ตลอดและจะมีการดึงข้อมูลส่วนนี้ได้จาก Data Dictionary

ขอบเขตของการศึกษา

1. ศึกษาวิธีการทำงานและทำการจำลอง (Simulation) ระบบ Distributed Query Optimizer เพื่อทำการทดลอง

2. ศึกษาการทำงานและการประมวลผลแบบขนาน (Parallel Computing)

3. ติดตั้งระบบคอมพิวเตอร์คลัสเตอร์ (Computer Clusters) และติดตั้งโปรแกรมต่างๆ ที่จำเป็นสำหรับการประมวลผลแบบขนาน (Parallel Computing) เพื่อใช้สำหรับดำเนินการวิจัย

4. ศึกษาเทคนิคการทำงานของ Island Base Parallel Genetic Algorithm

5. พัฒนาระบบจำลอง Distributed Query Optimizer ซึ่งใช้วิธีการค้นหา Island Base Parallel Genetic Algorithm

6. ดำเนินการวิจัยเพื่อทดสอบ และวัดประสิทธิภาพของการทำงานในเชิงเปรียบเทียบระหว่าง Island Base Parallel Genetic Algorithm กับเทคนิคการค้นหาแบบอื่นๆ เพื่อทดสอบว่า Island Base Parallel Genetic Algorithm เหมาะสมกับปัญหาและสามารถทำงานได้อย่างมีประสิทธิภาพภายใต้ข้อกำหนดและแบบจำลองของ Distributed Query Optimizer ที่ได้สร้างขึ้นหรือไม่

7. ศึกษาถึงปัจจัยต่างๆ ที่มีผลต่อการทำงานของ Island Base Parallel Genetic Algorithm ภายใต้ระบบ Distributed Query Optimizer ที่ได้จำลองขึ้น

8. รวบรวมผลการทดลองและสรุปผล

ขั้นตอนการศึกษา

ในงานวิจัยนี้สามารถแบ่งการศึกษาและการดำเนินการได้อย่างเป็นขั้นตอนดังนี้

1. รวบรวมความรู้จากเอกสารและแหล่งข้อมูลที่เกี่ยวข้องกับปัญหาที่ทำการศึกษา

2. ติดตั้งระบบ Clusters และติดตั้งโปรแกรมต่างๆ ที่จำเป็นต่อการคำนวณแบบขนาน

3. พัฒนาโปรแกรมเพื่อจำลองการทำงานของ Distributed Query Optimizer

4. ศึกษาขั้นตอนและการทำงานของ Island Base Parallel Genetic Algorithm เพื่อนำมาใช้ในกระบวนการ Distributed Query Optimization ในฐานข้อมูลแบบกระจาย (Distributed Database)

5. ทำการทดสอบประสิทธิภาพในการทำงานของ Island Base Parallel Genetic Algorithm พร้อมทั้งทำการศึกษาปัจจัยแวดล้อมในการทำงาน ที่มีผลต่อประสิทธิภาพของการทำงานของ Island Base Parallel Genetic Algorithm

6. เปรียบเทียบประสิทธิภาพของการทำงานของ Island Base Parallel Genetic Algorithm กับเทคนิคการค้นหาแบบต่างๆ

7. สรุปผลและข้อเสนอแนะ

เครื่องมือและอุปกรณ์

ซอฟต์แวร์

- ระบบปฏิบัติการ Linux Rock Clusters version 4.2
- โปรแกรม LAM/MPI version 7.1.3
- Java version 1.5.0_07
- mpiJava 1.2.5

ฮาร์ดแวร์

ระบบ Computer Clusters จำนวน 1 ชุดประกอบด้วย Front end node 1 เครื่อง และ Compute node อีก 3 เครื่อง

- CPU
- RAM 4.0 G.
- Intel(R) Xeon(TM) CPU 2.80GHz

ผลประโยชน์ที่คาดว่าจะได้รับ

ประโยชน์ที่คาดว่าจะได้รับจากงานวิจัย มีดังต่อไปนี้

1. สามารถพัฒนากระบวนการ Distributed Query Optimization ต้นแบบสำหรับใช้ในฐานข้อมูลแบบกระจาย (Distributed Database) ซึ่งใช้วิธีการค้นหาแบบ Island Base Parallel Genetic Algorithm ที่สามารถค้นหาวิธีการดำเนินการที่ดีที่สุดสำหรับคำสอบถาม ได้อย่างมีประสิทธิภาพและรวดเร็วในกรณีที่เป็น Large Join Query

2. สามารถบ่งชี้ได้ว่าการทำงานเพื่อแก้ปัญหาเรื่อง Distributed Query Optimization of Large Join โดยใช้ Island Base Parallel Genetic Algorithm สามารถทำงานได้อย่างมีประสิทธิภาพหรือไม่

3. สามารถบ่งชี้ถึงปัจจัยที่มีผลต่อประสิทธิภาพการทำงานของ Island Base Parallel Genetic Algorithm สำหรับการแก้ปัญหาเรื่อง Distributed Query Optimization of Large Join

บทที่ 2
วรรณกรรมที่เกี่ยวข้อง

การแก้ปัญหาเรื่อง Optimization of Large Join Query นับว่าเป็นปัญหาสำคัญปัญหาหนึ่งที่นักวิจัยหลายๆ คนให้ความสำคัญและพยายามที่จะนำเสนอวิธีการต่างๆ ที่มีแนวความคิดและวิธีการที่หลากหลายแตกต่างกันออกไป ดังนั้นผู้วิจัยจึงได้ทำการรวบรวม และคัดเลือกงานวิจัย ที่มีความสอดคล้องกับแนวคิดของงานวิจัยนี้ เพื่อนำมาศึกษาเป็นความรู้พื้นฐาน สำหรับดำเนินการวิจัยต่อไป ซึ่งมีรายละเอียดดังต่อไปนี้

กลุ่มงานวิจัยที่ศึกษาเกี่ยวกับการประยุกต์ใช้ Randomize Algorithms เพื่อใช้แก้ปัญหาเรื่อง Query Optimization

Swami and Gupta (1988) ได้นำเสนอว่าปัญหาเรื่อง Optimization of Large Join Queries นั้นก็คือ NP-Complete Combinatorial Problem จึงได้นำเสนอแนวคิดที่จะนำเทคนิคหลายๆ เทคนิคที่สามารถใช้แก้ปัญหาเรื่อง Combinatorial Problem มาประยุกต์ดัดแปลงเพื่อใช้สำหรับแก้ปัญหาเรื่อง Optimization of Large Join Queries โดยเทคนิคที่ Swami and Gupta (1988) ได้นำเสนอคือ Perturbation Walk, Quasi-Random Sampling, Iterative Improvement, Sequence heuristic และ Simulated Annealing และได้แสดงให้เห็นถึงวิธีการสำหรับดัดแปลงเทคนิคต่างๆ ที่กล่าวมาเพื่อประยุกต์ให้มีความเหมาะสมกับการแก้ปัญหานี้ นอกจากนี้ Swami and Gupta (1988) ได้นำเสนอการกำหนดค่าปัจจัยที่มีผลต่อการทำงาน (Tuning Parameters) ที่เหมาะสมสำหรับแต่ละเทคนิคเพื่อให้แต่ละเทคนิคสามารถทำงานได้อย่างเหมาะสมภายใต้เงื่อนไขของการทดลองเพื่อทำการเปรียบเทียบเพื่อทำการวัดค่าประสิทธิภาพของแต่ละเทคนิค Swami and Gupta (1988) ใช้เวลา (Time) เป็นปัจจัยสำคัญ สำหรับกำหนดการทำงานของเทคนิคต่างๆ เช่น การจำกัดเวลาการทำงานของแต่ละเทคนิค และในการเปรียบเทียบการทำงานของแต่ละเทคนิคนั้น Swami and Gupta (1988) ใช้ Analysis of Variance (ANOVA) ในการเปรียบเทียบ Algorithm ที่แตกต่างกันโดยผลของการเปรียบเทียบคือ

เทคนิค Iterative Improvement จะสามารถให้ประสิทธิภาพที่ดีที่สุดของการทำงานภายใต้เวลาที่จำกัด และ Simulated Annealing เป็นเทคนิคที่มีประสิทธิภาพรองลงมา

Swami (1989) ได้ทำงานวิจัยที่ขยายแนวคิดและวิธีการจากงานวิจัยเดิมคือ Swami and Gupta (1988) โดยงานวิจัยนี้ได้มุ่งศึกษาวิธีการทำงานแบบ Heuristic ที่สามารถใช้แก้ปัญหาในเรื่องนี้ได้ โดย Swami (1989) ได้ยกตัวอย่างวิธีการ Heuristic พื้นฐานที่ใช้ในการแก้ปัญหาเรื่องนี้อย่างเช่น Push Selections Down และ Perform Projection ซึ่งวิธีการ Heuristic ทั้งสองนี้สามารถช่วยลดขนาดของ Intermediate Result ซึ่งเป็นผลลัพธ์ชั่วคราวที่เกิดขึ้นจากการ JOIN ระหว่าง Relations ใดๆ ทำให้การทำงานเป็นไปได้อย่างรวดเร็วยิ่งขึ้น Swami (1989) ได้นำเสนอวิธีการ Heuristic 3 วิธีคือ KBZ Heuristic, Augmentation Heuristic และ Local Improvement และได้ทำการทดลองเปรียบเทียบประสิทธิภาพการทำงานของ Heuristic ทั้ง 3 วิธี จากการทดลอง Swami ได้ให้ความเห็นว่าวิธีการทั้งสามวิธี สามารถที่จะสร้างวิธีแก้ปัญหา (Solutions) ได้ไม่ครอบคลุม และถ้าหากเพิ่มเวลาในการทำงานขึ้น วิธีทั้งสามก็ไม่สามารถทำงานอย่างมีประสิทธิภาพเพิ่มขึ้น Swami จึงมีแนวคิดที่จะรวมการทำงานระหว่าง Heuristic กับเทคนิคที่ใช้แก้ปัญหา Combinatorial Optimization เพื่อเป็นการเพิ่มประสิทธิภาพของการทำงานให้สูงขึ้น โดยสามารถบรรยายถึงแนวคิดของการทดลองได้ดังนี้ คือ ทั้งวิธี Iterative Improvement (II) และ Simulate Annealing (SA) จะใช้วิธีการในการ Random state ขึ้นมาเพื่อใช้สำหรับเป็นจุดเริ่มต้นของการค้นหา ซึ่ง State ที่สุ่มขึ้นมาได้นั้นอาจไม่ใช่ค่าที่เหมาะสมทำให้การค้นหาไม่มีประสิทธิภาพ ด้วยเหตุนี้เองจึงเป็นหน้าที่ของ Heuristic ที่จะทำการค้นหาและเลือกจุดเริ่มต้นที่ดีที่เป็น Local minima จุดใดจุดหนึ่งใน Search space เพื่อใช้เป็น State เริ่มต้นให้กับ Iterative Improvement (II) และ Simulate Annealing (SA) และเมื่อจุดเริ่มต้นของทั้งสอง Algorithm นั้นเป็น Local minima แล้วการค้นหา Global minima ของทั้งสอง Algorithm ก็จะเป็นไปได้อย่างมีประสิทธิภาพและรวดเร็วขึ้น และผลจากการทดลองพบว่าวิธีการ AGI เป็นวิธีที่ให้ประสิทธิภาพดีที่สุด โดยวิธีนี้เป็นวิธีที่ใช้ Augmentation Heuristic เพื่อสร้างจุดเริ่มให้กับ Iterative Improvement เพื่อใช้ค้นหาคำตอบต่อไป

Ioannidis and Kang (1990) ได้นำเสนองานวิจัยที่ทำการเปรียบเทียบประสิทธิภาพของ Randomize Algorithm สองตัวคือ Iterative Improvement และ Simulated Annealing ในการนำมาใช้เพื่อแก้ปัญหาเรื่อง Optimization of Large Join Queries ซึ่ง Ioannidis และ Kang ได้นำเสนอวิธีการ

และผลของการวิจัย ที่มีความแตกต่างออกไปจากงานของ Swami and Gupta (1988) ซึ่งสามารถสรุปความแตกต่างของทั้งสองงานวิจัยได้ดังนี้คือ

ประการแรก ความแตกต่างของสมาชิกภายใน Search space กล่าวคือในงานวิจัยของ Swami and Gupta (1988) ภายใน Search space ประกอบไปด้วย Left-deep Tree เท่านั้นแต่ในงานของ Ioannidis and Kang (1990) จะประกอบไปด้วย Tree ประเภทต่างๆ ที่เป็นไปได้ทั้งหมด

ประการที่สอง ความแตกต่างของวิธีในการ JOIN โดยที่งานวิจัยของ Swami and Gupta (1988) จะทำการพิจารณาเฉพาะการ JOIN แบบ Hash-join แต่วิธีการของ Ioannidis and Kang (1990) จะใช้วิธีการ JOIN แบบ Nested-Loop และ Merge-Scan

ประการที่สาม ความแตกต่างในการคิดค่า Cost กล่าวคืองานของ Swami and Gupta (1988) ได้ทำการสร้าง Database จำลองขึ้นบน Main memory และใช้ CPU Time เป็นค่า Cost ที่เกิดขึ้น แต่สำหรับงานของ Ioannidis และ Kang ใช้ I/O Time เป็นค่า Cost ที่เกิดขึ้น

ประการที่สี่ Swami and Gupta (1988) ได้ใช้การประมาณค่าของ Local minimum สำหรับการทำงานของ Iterative Improvement แต่วิธีของ Ioannidis and Kang (1990) จะใช้วิธีที่แตกต่างกันออกไป

ประการที่ห้า ความแตกต่างของ Transformation Rules ที่ใช้สำหรับสร้าง Neighbors state วิธีการของ Swami and Gupta (1988) ได้ใช้ Transformation Rule ที่ชื่อว่า Cyclic Exchange แต่ในงานของ Ioannidis and Kang (1990) ใช้ Transformation Rule ที่อยู่บนพื้นฐานของ Algebraic Properties ของการ JOIN

Ioannidis and Kang (1990) ได้นำเสนอผลการทดลองที่แตกต่างออกไปจากงานของ Swami and Gupta (1988) โดยได้ทำการบรรยายถึงเหตุผลของความแตกต่างไว้ว่า การที่ทั้งสองไม่เห็นด้วยกับข้อสรุปของการทดลองของ Swami and Gupta (1988) ที่ได้ทำการสรุปผลของการทดลองไว้ว่า Iterative Improvement มีประสิทธิภาพการทำงานที่เหนือกว่า Simulated Annealing เพราะในเวลาทำการทดลองนั้นได้มีความแตกต่างกันของค่า Cost ของแต่ละ State มากจนเกินไป เพราะเนื่องมาจาก Transformation Rules ที่ Swami and Gupta (1988) ใช้นั้นก่อให้เกิดความแตกต่างของค่า Cost ของ Current state กับ Neighbors state มากจนเกินไป ทำให้รูปร่างของการเปลี่ยนแปลงของค่า Cost ปราศจากความเรียบ (Smooth) ด้วยเหตุผลนี้เองจึงเป็นสาเหตุสำคัญที่ทำให้วิธี Simulate

Annealing ไม่สามารถทำงานได้อย่างมีประสิทธิภาพ และด้วยเหตุผลนี้เองทำให้การทำงานของ Iterative Improvement สามารถทำงานได้ง่ายขึ้นอีกด้วย

จากงานวิจัยนี้ Ioannidis and Kang (1990) ได้ทำการสรุปผลที่ได้จากการทดลองไว้ดังนี้คือ Iterative Improvement จะทำงานได้รวดเร็วกว่า Simulate Annealing ในช่วงแรกเท่านั้น แต่ถ้ามีเวลาที่เพียงพอสำหรับการทำงานของ Simulate Annealing แล้ว Simulate Annealing จะมีประสิทธิภาพที่เหนือว่า Iterative Improvement

Pongpinigpinyo (1996) นำเสนอการแก้ปัญหาเรื่อง Optimization of Large Queries ในระบบ Distributed Database ซึ่งสำหรับ Distributed Database จะมีวิธีการสำหรับการคิดค่า Cost ที่มีความซับซ้อน ซึ่ง Pongpinigpinyo (1996) ได้นำวิธีการคิดหาค่า Cost (Cost Function) มาจาก System R* (Lohman 1985) ประเด็นหลักสำหรับงานวิจัยนี้คือการนำเสนอ Algorithm ที่ทำงานแบบ 2 Phase โดยเป็นการทำงานร่วมกันระหว่างวิธี Heuristic และ Simulated Annealing เพื่อใช้ในการแก้ปัญหานี้ โดยเรียกวิธีการ Search ที่พัฒนาขึ้นมานี้ว่า Two-State Simulated Annealing (TSSA) โดย Stage แรกเป็นการทำงานของวิธีการ Heuristic ที่ชื่อว่า The least join selectivity ซึ่งจะทำงานภายใต้ช่วงเวลาที่จำกัดเพื่อทำการค้นหา Local Minimum ที่ดีที่สุด เมื่อเสร็จสิ้นการทำงานของ The least join selectivity แล้วจะเข้าสู่การทำงานใน Stage ที่สองซึ่งเป็นการทำงานของ Simulated Annealing เพื่อทำการค้นหา Global Minimum ต่อไปซึ่งการทำงานของ Simulate Annealing จะใช้ State เริ่มต้นจากผลลัพธ์ที่ได้จาก Stage แรกซึ่งเป็นค่า Best Local Minimum แทนที่จะใช้ State แรกจากการสุ่มเลือกจาก Search space และในงานวิจัยนี้ สุนีย์ ได้นำเสนอการทดลอง ที่ทำการทดลองโดยเปรียบเทียบประสิทธิภาพของการทำงานของ Search Algorithm 3 แบบคือ Exhaustive Search, Simulated Annealing (SA), Two-State Simulated Annealing (TSSA) โดยการทดลองนั้นได้มีการปรับแต่งค่า Parameters ของ Simulated Annealing ใน Two-State Simulated Annealing คือปัจจัยเรื่องอุณหภูมิตั้งต้น (Initial Temperature) ให้มีค่าที่แตกต่างออกไปจาก Simulated Annealing แบบธรรมดา และจำนวนของ Relation ที่ใช้ทดลองมีค่าอยู่ระหว่าง 6 - 41 Relations ซึ่งผลที่ได้จากการทดลองนั้นสามารถทำการสรุปได้ว่าวิธีการ Two-State Simulated Annealing (TSSA) ให้ประสิทธิภาพการค้นหาที่รวดเร็วกว่า Simulate Annealing

Matysiak (1995) ได้นำเสนอการแก้ปัญหาเรื่อง Optimization of Large Join Queries โดยการนำวิธีการค้นหา Taboo Search (TS) มาใช้สำหรับแก้ปัญหา โดยจุดประสงค์หลักของงานวิจัยของ Matysiak (1995) คือ การทดลองเปรียบเทียบประสิทธิภาพของการทำงานของ Taboo Search (TS) กับ Simulated Annealing (SA) และ Iterative Improvement (II) แต่สำหรับการทดลองของ Matysiak (1995) นั้นได้มีการตั้งค่า Parameters ของ Randomize Algorithms ทั้งสองตัวคือ Simulated Annealing (SA) และ Iterative Improvement (II) ที่แตกต่างออกไปจากการทดลองของ Ioannidis and Kang (1990) และใช้การวิธีการเคลื่อนจาก State ปัจจุบันไปยัง State ต่อไปด้วยวิธีการแบบ Join Method Exchange ด้วยวิธีการเลือกแบบสุ่ม ซึ่งวิธีนี้เป็นการเปลี่ยนแปลงไปมาระหว่างวิธีการ Join สองแบบคือ แบบ Nested-Loop และวิธี Sort-Merge ซึ่งวิธีการคิดค่า Cost นั้นจะขึ้นอยู่กับรูปแบบของวิธีการ Join แต่ละแบบซึ่งจะมีวิธีการคิดที่แตกต่างกันออกไป

กลุ่มงานวิจัยที่ศึกษาเกี่ยวกับการประยุกต์ใช้ Genetic Algorithm เพื่อแก้ปัญหาเรื่อง Query Optimization

Bennett, Ferris and Ioannidis (1991) ได้นำเสนอวิธีการแก้ปัญหาเรื่อง Optimization of Large Join Query โดยใช้ Genetic Algorithm สำหรับค้นหา Best Query Execution Plan โดยในการทดสอบ Bennett, Ferris และ Ioannidis ได้ทำการแบ่ง Search space ออกเป็นสองลักษณะคือ Search space ที่ประกอบไปด้วย Outer Linear Join Processing Tree (Left-Deep Tree) เพียงอย่างเดียว และ Search space อีกประเภทที่ประกอบไปด้วยทั้ง Left-Deep Tree ผสมกับ Bushy Tree ซึ่ง Search space ประเภทนี้จะมีขนาดใหญ่กว่าประเภทแรกที่ประกอบไปด้วย Left-Deep Tree เพียงอย่างเดียว แต่ก็มีความเป็นไปได้ว่าคำตอบที่ดีที่สุดในการค้นหาอาจไม่ได้อยู่ใน Search space ที่ประกอบไปด้วย Left-Deep Tree เพียงอย่างเดียวดังนั้นการทดสอบเพื่อหาคำตอบที่ถูกต้องที่สุดภายใน Search space ที่ประกอบไปด้วย Tree ทั้งสองประเภทจึงมีความสำคัญอย่างมาก โดยในการทดสอบเพื่อวัดประสิทธิภาพการทำงานของ Algorithm Bennett, Ferris และ Ioannidis ได้นำผลที่ได้ไปเปรียบเทียบกับผลของการทำงานของ System R เพื่อทำการวัดประสิทธิภาพ แต่อย่างไรก็ตามการทำงานของ System R นั้นจะใช้ Search space ที่ประกอบไปด้วย Left-Deep Tree เพียงอย่างเดียวเท่านั้น

Dong and Liang (2007) นำเสนอที่มาของปัญหา สำหรับงานวิจัยของทั้งสองว่า Relational Databases Application สมัยใหม่ได้มีการใช้วิธีการ Join Query ที่ซับซ้อนมากขึ้น โดยที่มีระบบ Database ขนาดใหญ่เป็นส่วนหนึ่งของ Application ยกตัวอย่างเช่น ระบบ Decision support , ระบบ Data warehouse คือ ศูนย์กลางข้อมูลของระบบ Decision support และใช้ Relational Model สำหรับการจัดเก็บ และจัดการบริหารข้อมูลต่างๆ ซึ่งจะต้องมีการรับ-ส่ง และแลกเปลี่ยนข้อมูลกันระหว่าง Database ต่างๆ ซึ่งตรงจุดนี้นั้นจะมีการใช้การดำเนินการแบบ Large Join Query เกิดขึ้น

ดังนั้น Dong and Liang (2007) จึงได้นำเสนองานวิจัยที่มีแนวความคิดว่า การนำ Genetic Algorithm มาใช้ในการแก้ปัญหาเรื่อง Large Join Query Optimization สำหรับการทดลองที่ผ่านมาในงานวิจัยต่างๆ นั้นเป็นไปในรูปแบบของการนำ Genetic Algorithm มาวัดประสิทธิภาพการทำงานในการแก้ปัญหา โดยการเปรียบเทียบกับประสิทธิภาพของ Randomize Algorithm อื่นๆ และการทดลองจะเป็นไปในรูปแบบที่มีการกำหนดปัจจัยแวดล้อมที่มีผลต่อประสิทธิภาพของ Algorithm เอนเอียงไปสำหรับ Randomize Algorithm ใด Algorithm หนึ่งเท่านั้น หรืออาจจะมีการให้ความสำคัญกับปัจจัยใดปัจจัยหนึ่งเท่านั้น ซึ่งทางผู้วิจัยนั้นได้เชื่อว่า สำหรับปัญหาเรื่อง Large Join Query Optimization นั้นมีปัจจัยแวดล้อมต่างๆ หรือมีองค์ประกอบต่างๆ มากมายที่จะส่งผลต่อประสิทธิภาพการทำงานของ Genetic Algorithm และจากงานวิจัยที่ผ่านๆ มาผู้วิจัยเชื่อว่าวิธีการแบบ Hybrid Genetic Algorithm สามารถที่จะลดขนาดของ Search Space และเพิ่มประสิทธิภาพของ Genetic Algorithms ได้ โดย Dong and Liang (2007) ได้ทำการนำเสนอ Genetic Optimization Model ซึ่งรวบรวมปัจจัยต่างๆ ที่มีผลต่อประสิทธิภาพการทำงานของ Genetic Algorithm

Dong and Liang (2007) ยังได้นำเสนออีกว่า สาเหตุใดจึงคิดว่าการเพิ่มประสิทธิภาพสำหรับการ Query ด้วยวิธีการทั้งหมดนั้นยังมีปัจจัยอื่นๆ อีกมากมายเป็นตัวกำหนด คือ 1. Genetic Algorithms นั้นเป็น Randomize Algorithm ซึ่งประสิทธิภาพจะขึ้นอยู่กับธรรมชาติของแต่ละปัญหาและการกำหนดค่า Parameters ต่างๆ สำหรับ Genetic Algorithm 2. จากทฤษฎีเรื่อง No-Free-Lunch ไม่มี Algorithm ใดที่สามารถแก้ปัญหาใดๆ ได้ทุกๆ เรื่อง และไม่มีทางที่จะเหนือกว่าคู่แข่งอื่นๆ เสมอไปในทุกๆ ด้าน สำหรับการดำเนินการทดลองใน Dong and Liang (2007) ผู้วิจัยได้แสดงให้เห็นถึงปัจจัยที่เหมาะสม ซึ่งเป็นการแลกเปลี่ยนระหว่าง ประสิทธิภาพการทำงานของ Genetic Algorithm และความน่าเชื่อถือของคุณภาพของผลลัพธ์ที่ได้จาก Genetic Algorithm โดยทำการ

บรรยายความสัมพันธ์ระหว่างองค์ประกอบย่อยๆ ต่างๆ ที่มีผลต่อการทำงาน และได้ทำการทดลองเปรียบเทียบประสิทธิภาพของ Hybrid Genetic Algorithm ต่างๆ และได้กำหนดสมาชิกของ Search space ไว้เป็น Left-Deep Tree เพียงอย่างเดียวเท่านั้น

กลุ่มงานวิจัยที่ศึกษาเกี่ยวกับการประยุกต์ใช้ Genetic Programming เพื่อแก้ปัญหาเรื่อง Query Optimization

Stillger and Spiliopoulou (1996) ได้เสนอว่าการนำ Genetic Algorithm มาใช้สำหรับแก้ปัญหาเรื่อง Query Optimization นั้นพบกับปัญหาเรื่องความซับซ้อนและความกำกวมในการทำงาน โดยได้กล่าวถึงงานวิจัยของ Bennett et al. (1991) ว่าขั้นตอนการทำงานใน Bennett et al. (1991) นั้นจะต้องทำการแปลง Query Execution Plan (QEPs) ให้กลายเป็น Chromosomes ซึ่งอยู่ในรูปของสายอักขระสำหรับการทำงานของขั้นตอนการ Crossover, Selection และ Mutation และมี Fitness function ซึ่งทำงานโดยใช้พื้นฐานของ Cost Model แต่อย่างไรก็ตาม เมื่อทำการแปลง QEPs ไปอยู่ในรูปของ Chromosomes แล้วนั้นจะต้องทำการแปลงจากรูปของ Chromosomes ให้กลับมาอยู่ในรูปของ QEPs อีกทีหนึ่งเพื่อทำการคำนวณค่า Cost ในขั้นตอนของ Fitness function ซึ่งการแปลงรูปไปมาระหว่าง Chromosomes และ QEPs นั้นเป็นการสิ้นเปลืองเวลาอย่างมาก และที่สำคัญที่สุดคือการดำเนินการในขั้นตอนของการ Crossover นั้นอาจเป็นการทำลายโครงสร้างของ Tree ที่ใช้สำหรับแทนค่าของแต่ละ QEPs ให้ผิดเพี้ยนไปซึ่งส่วนนี้จึงเป็นสาเหตุที่ทำให้ต้องเสียเวลาในการซ่อมแซมโครงสร้างของ Tree ให้ถูกต้องก่อนทำการคำนวณค่า Cost โดย Fitness Function ซึ่งจากปัจจัยผลเสียเหล่านี้ทาง Stillger and Spiliopoulou (1996) จึงได้นำเสนอวิธีการคือ Genetic Programming เพื่อใช้สำหรับแก้ปัญหาต่างๆ ที่เกิดขึ้นกับ Genetic Algorithm โดยวิธีการของ Genetic Programming จะใช้รูปแบบของ Query Execution Plans (QEPs) ที่อยู่ในรูปโครงสร้างของ Tree เพียงอย่างเดียวโดยจะไม่มีการแปลงโครงสร้างของ Tree ให้การเป็นสายของอักขระสำหรับขั้นตอนการดำเนินของการ Crossover, Selection และ Mutation

Stillger, Spiliopoulou, and Freytag (1996) ได้นำเสนอวิธีในการดำเนินกระบวนการ Parallel Query Optimization โดยใช้ Genetic Programming ซึ่งเป็นการใช้ประโยชน์จาก Query Execution Plans ที่มีลักษณะแบบ Bushy Join Trees ซึ่งผู้วิจัยทั้งสามได้ให้ความเห็นว่า สามารถที่จะ

เพิ่มประสิทธิภาพการทำงานของระบบ DBMS ได้หากว่าระบบนั้นสามารถที่จะทำการ Query ข้อมูลได้ในรูปแบบการทำงานแบบขนาน หรืออาจเรียก Database ประเภทนี้ว่า Database Parallelism หากว่าผลลัพธ์ที่ได้จากกระบวนการ Query Optimization นั้นเป็นผลลัพธ์ที่สามารถรองรับหรือ สามารถนำไปประมวลผลได้ในรูปแบบการทำงานแบบขนาน ดั้งนั้นสิ่งสำคัญที่งานวิจัยนี้ได้มุ่งประเด็นสำหรับการศึกษาก็คือ การค้นหา Query Execution Plan ที่ดีที่สุดจาก Search space ที่มีแต่ Bushy Join Trees โดยใช้วิธีการค้นหาเป็นแบบ Genetic Programming

บทที่ 3
ทฤษฎีที่เกี่ยวข้อง

บทนี้จะกล่าวถึงทฤษฎีและอัลกอริทึมที่จำเป็นต่องานวิจัย โดยจะกล่าวตามลำดับได้ดังนี้

ฐานข้อมูลเชิงสัมพันธ์

ระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database) เป็นฐานข้อมูลที่ใช้โมเดลเชิงสัมพันธ์ (Relational Database Model) เนื่องด้วยแนวคิดของแบบจำลองนี้มีลักษณะที่คนใช้กันทั่ว กล่าวคือมีการจัดเก็บข้อมูลเป็นตาราง ทำให้เข้าใจและประยุกต์เพื่อใช้งานจริง ด้วยเหตุนี้ระบบฐานข้อมูลแบบนี้จึงได้รับความนิยมมาก ในแง่ของ Entity แบบจำลองนี้คือ แฟ้มข้อมูลในรูปแบบของตาราง และ Attribute ก็เปรียบเหมือนขอบเขตข้อมูล ส่วนความสัมพันธ์คือความสัมพันธ์ระหว่าง Entity

ฐานข้อมูลเชิงสัมพันธ์ คือ การเก็บข้อมูลในรูปแบบของตาราง (Table) ในแต่ละตาราง แบ่งออกเป็น แถวๆ และในแต่ละแถวจะแบ่งเป็นคอลัมน์ (column) ในทางทฤษฎีจะมีคำศัพท์เฉพาะแตกต่างออกไปเนื่องจากแบบจำลองแบบนี้เกิดจากทฤษฎีทางคณิตศาสตร์เรื่องเซต (Set) ดังนั้น จึงได้มีคำศัพท์เฉพาะดังนี้

- รีเลชั่น (Relation) คือ ตาราง (Table)
- ทูเปิล (Tuple) แถว (Row) หรือ เรคคอร์ด (Record) หรือ ระเบียน
- แอททริบิวท์ (Attribute) คอลัมน์ (Column) หรือ ฟิลด์ (Field)
- คาร์ดินัลลิตี้ (Cardinality) จำนวนแถว (Number of rows)
- ดีกรี (Degree) จำนวนแอททริบิวท์ (Number of attribute)
- คีย์หลัก (Primary key) ค่าเอกลักษณ์ (Unique identifier)
- โดเมน (Domain) ขอบข่ายค่าของข้อมูล (Pool of legal values)

ชนิดของ Relations

ในระบบจัดการฐานข้อมูลทั่วๆ ไป Relation อาจจำแนกออกได้เป็น 2 ประเภท ดังต่อไปนี้ คือ

Relation หลัก (Base Relation)

เป็น Relation ที่ถูกกำหนดขึ้นเพื่อเก็บข้อมูลและเพื่อนำข้อมูลไปใช้เมื่อมีการสร้าง Relation โดยใช้ Data Definition Language เช่น ใน SQL คำสั่ง CREATE TABLE เป็นการสร้าง Relation หลักหลังจากนั้นก็จะทำการเก็บข้อมูลเพื่อการเรียกใช้ข้อมูลในภายหลัง Relation หลักจะเป็นตารางที่จัดเก็บข้อมูลจริงไว้

วิว (View)

อาจเรียกอีกอย่างหนึ่งว่า Relation สมมติ (Virtual Relation) เป็น Relation ที่ถูกสร้างขึ้นตามความต้องการของการใช้ข้อมูลของแต่ละคน เนื่องจากผู้ใช้แต่ละคนต้องการใช้ข้อมูลในลักษณะที่แตกต่างกันจึงทำการกำหนดวิวของตัวเองขึ้นมาจาก Relation หลักเพื่อความสะดวกในการใช้ข้อมูลและช่วยให้การรักษาความปลอดภัยของฐานข้อมูลทำได้ง่ายขึ้น Relation ที่ถูกสมมติขึ้นมานี้จะไม่มีการเก็บข้อมูลจริงในระบบฐานข้อมูล

Distributed Relational Databases

ความแตกต่างระหว่าง Centralized Relational Database กับ Distributed Relational Database สามารถบรรยายได้ดังนี้คือ Distributed Database นั้น Relation ต่างๆ จะสามารถจัดเก็บไว้ในสถานที่ (Site) ที่แตกต่างกัน ซึ่งเชื่อมต่อกันโดยระบบ Network และสามารถที่จะเข้าถึง Relations ต่างๆ นั้นจากสถานที่ต่างๆ โดยผ่านระบบ Network เช่นกัน

Operations

สำหรับการดำเนินการ Data Manipulation Operations ที่ใช้ภายในงานวิจัยนี้คือ Projection, Selection, Join โดยสามารถอธิบายการดำเนินการต่างๆ ได้ดังนี้

Projection

ผลลัพธ์ของการดำเนินการ Projection กับ Relation R บน Set ของ Attributes A นั้นจะได้ค่าผลลัพธ์คือ ค่าที่อยู่ภายใน Attributes A (หรือ Column A) ภายใน Relation R

Selection

เป็นการคัดเลือกข้อมูลจากแต่ละ Tuples ใน Attribute A ของ Relation R โดยการคัดเลือกข้อมูลจะเป็นไปตามเงื่อนไขที่กำหนด คือ $\leq, \geq, \neq$ และ =

Join

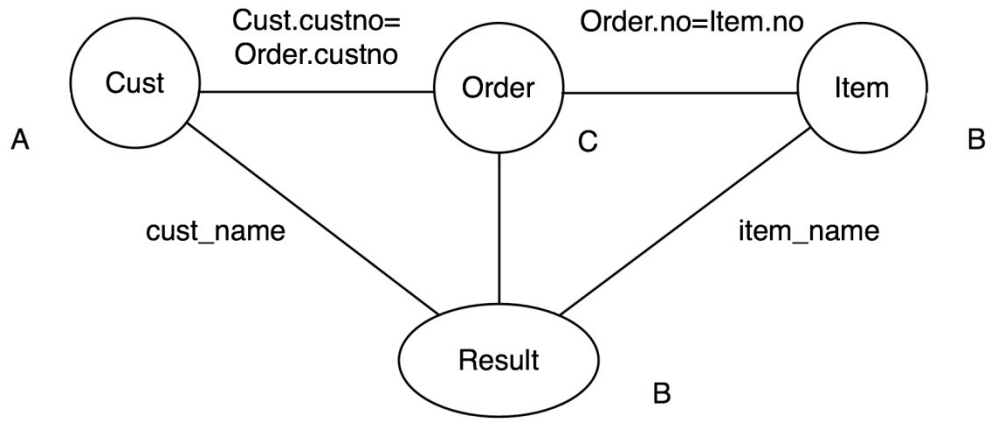
การ Join ระหว่าง Relations R1 และ R2 บน Attribute A สามารถเขียนแทนด้วย R1.A $\bowtie$ R2.A โดยผลลัพธ์ที่ได้จากการ Join คือค่าที่ได้จากการรวมระหว่างแถว (Row) ใน Relation R1 และ R2 ภายใต้เงื่อนไขที่กำหนด

Query Graph

เราสามารถแทนค่าของ Input Query ให้อยู่ในรูปของ Query Graph ได้ โดย Query Graph นั้นจะแสดงให้เห็นถึงความสัมพันธ์ระหว่างแต่ละ Relations ที่ถูกดำเนินการและเงื่อนไขของการดำเนินการระหว่าง Relations

Query Graph (Pongpinigpinyo 1996) นั้นสามารถแสดงให้เห็นถึงการดำเนินการต่างๆ ไม่ว่าจะเป็น Selection, Project และ Join ดังที่สามารถแสดงให้เห็นได้ในภาพที่ 1 โดยกำหนดให้ A, B, C นั้นคือ Site ที่แต่ละ Relations นั้นอยู่โดยเชื่อมต่อกันด้วยระบบ Computer Network

```
Query Q:    SELECT cust_name, item_name
            FROM  Cust, Item, Order
            WHERE Cust.custno=Order.custno and Order.no=Item.no
```



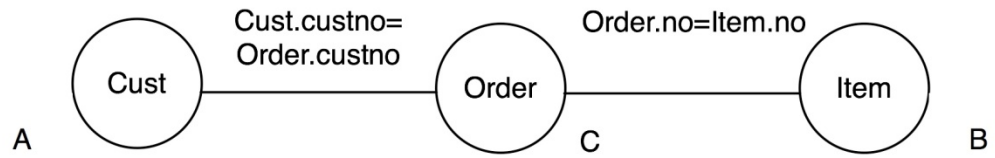
ภาพที่ 1 Query Graph ของ Query Q

จากภาพที่ 1 ภายใน Query Graph มีหนึ่ง node ถูกเรียกว่า “Result node” ถูกใช้เพื่อแสดงถึง Relations ที่เป็นผลลัพธ์ของการดำเนินการ และเรียก node อื่นๆ ว่า “Operand relation node” ซึ่งแสดงถึง Relations ที่ถูกดำเนินการ และ “Result node” จะถูกแทนโดยใช้วงรี และ “Operand relation node” แทนโดยใช้วงกลม และเส้นที่เชื่อมต่อกันระหว่างแต่ละ node ใช้แสดงเงื่อนไขสำหรับการดำเนินการระหว่างแต่ละ Relations โดยระหว่างแต่ละ node อาจมีมากกว่าหนึ่งเงื่อนไขสำหรับแต่ละการดำเนินการระหว่าง Relations แต่ละคู่

ความสำคัญของ Query Graph สำหรับงานวิจัยนี้คือ ถูกใช้เป็นค่า Input เริ่มต้นสำหรับดำเนินกระบวนการ Query Optimization

Join Graph

Join Graph (Pongpinigpinyo 1996) คือส่วนหนึ่งของ Query Graph แต่จะแสดงเพียงเฉพาะเงื่อนไขการ Join แบบเท่ากับ (Equijoin predicate) ระหว่างสอง Relations เพราะฉะนั้นแต่ละ Node ภายใน Join Graph จะถูกเชื่อมต่อกันด้วยเส้นที่แสดงเงื่อนไขของการ Join แบบเท่ากับ (Equijoin predicate) ซึ่งสามารถแสดง Join Graph ที่ได้จาก Query Graph ในภาพที่ 2 ดังนี้



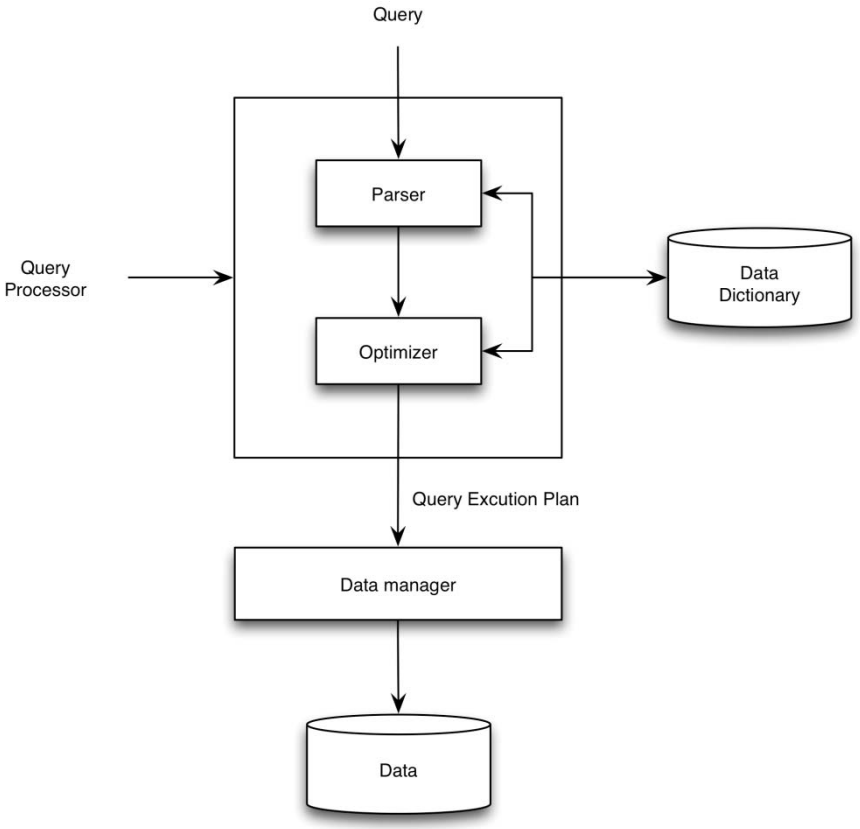
ภาพที่ 2 Join Graph of Query Q

การประมวลผลสอบถาม (Query Processing)

การประมวลผลสอบถาม (กระบวนการสอบถามข้อมูล 2004) เป็นหน้าที่หนึ่งของระบบการจัดการฐานข้อมูล โดยระบบการจัดการฐานข้อมูล (DBMS) จะทำการประมวลผลสอบถามที่เกิดจากภาษาสอบถาม (Query Language) ได้แก่ ภาษา SQL (Structure Query Language) และหาคำตอบที่ดีที่สุดและถูกต้องตรงกับความต้องการของผู้ใช้ ได้อย่างเหมาะสม รวดเร็วและเสียค่าใช้จ่ายในการดำเนินการน้อยที่สุด

การประมวลผลสอบถาม เป็นกระบวนการในการเลือกแผน หรือกลยุทธ์ที่เหมาะสมในการสอบถามข้อมูลในฐานข้อมูล โดยระบบจัดการฐานข้อมูล จะมีตัวประมวลผลที่เรียกว่า ตัวประมวลผลสอบถาม (Query Processor) ทำหน้าที่ในการเลือกแผนหรือกลยุทธ์ที่เหมาะสมในการเข้าถึงข้อมูลเพื่อให้ได้คำตอบที่ดีที่สุดและถูกต้องที่ตรงกับความต้องการของผู้ใช้งานมากที่สุด

โดยสามารถแสดงโครงสร้างและองค์ประกอบต่างๆ รวมทั้งลำดับของการทำงานของ Query Processor ได้ดังภาพที่ 3



ภาพที่ 3 โครงสร้างของ Query Processor

โดยสามารถอธิบายรายละเอียดของโครงสร้างสำคัญของ Query Processor ได้ดังนี้

1. ส่วนของตรวจสอบภาษา (Parser)

ทำหน้าที่ตรวจเช็คกฎและหลักเกณฑ์ (Syntax) ของคำสั่งสอบถามข้อมูลในรูป SQL โดยจะทำการตรวจสอบข้อมูลที่อยู่ในพจนานุกรมข้อมูลแล้วทำการแปลคำสั่งให้อยู่ในรูป Relational Algebra

2. ออฟติไมเซอร์ (Optimizer)

ทำหน้าที่ในการเลือกกลยุทธ์หรือวิธีการที่เหมาะสมเพื่อให้การเข้าถึงข้อมูลประหยัดเวลามากที่สุด โดยเมื่อคำสั่งในภาษาสอบถามผ่านตัวตรวจสอบภาษา (Parser) แล้วจะได้คำสั่งที่อยู่ในรูป Relational Algebra แล้ว ออฟติไมเซอร์จะหาทางเลือกที่เหมาะสมที่สุด ซึ่งสามารถ

ทำได้หลายวิธี วิธีการหนึ่งที่นิยมคือ การแปลง Relational Algebra ให้อยู่ในรูปแบบโครงสร้างของข้อมูลกราฟ เรียกว่า Operator Graph และทำการคำนวณค่าใช้จ่ายจากกราฟได้

กระบวนการ Query Optimization

Query Optimization หมายถึง กระบวนการที่ใช้สำหรับค้นหาวิธีในการ Execute คำสั่ง Input Query ที่รับเข้ามาให้เป็นไปอย่างมีประสิทธิภาพมากที่สุด โดยในการตัดสินว่าวิธีการใดเป็นวิธีที่ดีที่สุดนั้นจะอยู่บนพื้นฐานการทำงานของ Cost Function เป็นตัวกำหนด โดยเป้าหมายหลักในการทำงานของกระบวนการ Query Optimization คือการเลือกวิธีการ Execute ที่ต้องสูญเสียค่า Cost น้อยที่สุด

โดยกระบวนการ Query Optimization เป็นกระบวนการที่เกิดอยู่ในองค์ประกอบหลักของ ระบบ Database Management System (DBMS) ที่มีชื่อเรียกว่า Query Optimizer ซึ่งองค์ประกอบหลักที่สำคัญของ Query Optimizer ประกอบไปด้วย Search Space, Cost Model และ Search Strategy

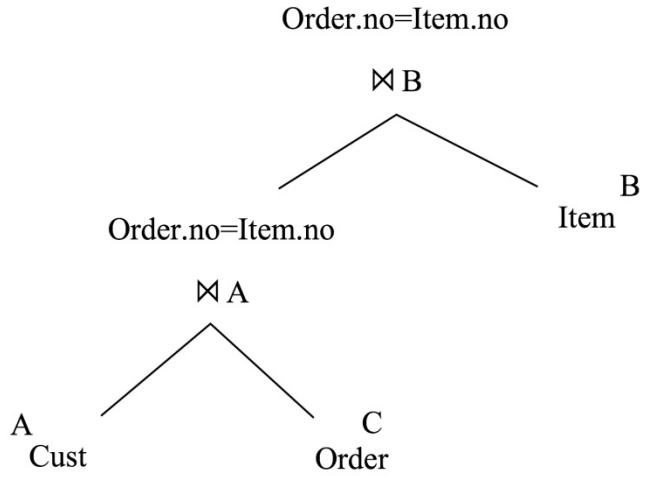
Search Space คือ กลุ่มของวิธีการที่เป็นไปได้ทั้งหมดสำหรับการ Execute คำสั่ง Input Query ที่รับเข้ามา โดยวิธีการแต่ละวิธีการที่เป็นไปได้ทั้งหมดนั้น สามารถที่จะใช้ Query Execution Plans (QEPs) ในการแทนค่าเพื่อให้เห็นถึงลำดับการทำงานของแต่ละวิธี หรืออาจกล่าวได้ว่า Search Space นั้นก็คือ Set ของ QEPs ที่เป็นไปได้ทั้งหมดนั่นเอง

Cost Model คือ วิธีในการคิดคำนวณค่า Cost ที่จะเกิดขึ้นจากแต่ละวิธีที่เป็นไปได้ ปรกติแล้ว Cost Model โดยทั่วไปสำหรับ Distributed Database นั้น การคิดค่า Cost ที่เกิดขึ้นจะสามารถทำได้โดยการหาผลรวมของ เวลาที่ใช้ไปในการประมวลผลและเข้าถึงข้อมูล (Local Processing Cost) กับเวลาที่ใช้ในการรับและส่งข้อมูลระหว่าง Site (Data Transfer Cost)

Search Strategy คือ วิธีที่ใช้สำหรับค้นหาว่าวิธีในการ Execute แบบใดที่ให้ค่า Cost ที่น้อยที่สุดจากวิธีการที่เป็นไปได้ทั้งหมด ซึ่งโดยปรกติแล้วภายในกระบวนการ Query Optimization นั้นจะใช้วิธีการค้นหาแบบละเอียด (Exhaustive) ซึ่งเป็นการเข้าไปค้นหาและเปรียบเทียบค่า Cost ที่เกิดขึ้นจากทุกๆ วิธีการที่เป็นไปได้ทั้งหมดทำให้ผลลัพธ์ที่ได้จากการค้นหามีความถูกต้องมากที่สุด

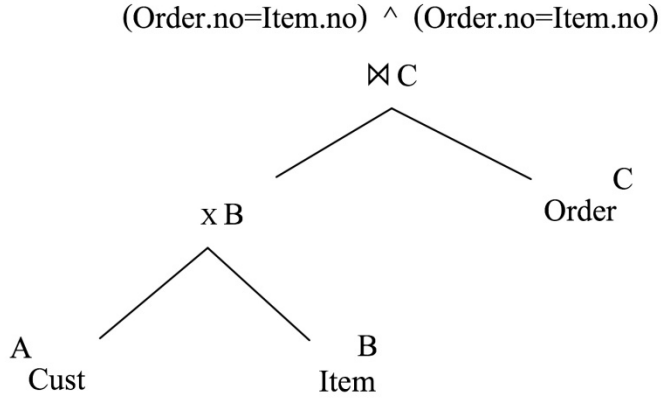
Join Processing Trees

Join Processing Trees (Pongpinigpinyo 1996) ถูกใช้เพื่อแสดงให้เห็นถึงลำดับในการดำเนินการกับแต่ละ Relations ของแต่ละ Query Execution Plans (QEPs) โดยกำหนดให้ Node ที่ไม่ใช่ Node ลูก (None-Leaf Node) แทน Join Operator และ Node ลูก (Leaf-Node) แต่ละ Node แทน Base Relations โดยเราสามารถคำนวณเพื่อหาผลลัพธ์ที่เกิดจากการดำเนินการในรูปของ Join Processing Trees ได้โดยการอ่านค่าแบบ Bottom up โดยสามารถยกตัวอย่างของ Join Processing Tree ที่เกิดขึ้นจาก Join Graph ในภาพที่ 3 ได้ดังภาพที่ 4



ภาพที่ 4 หนึ่งใน Query Execution Plan ที่เกิดขึ้น

แต่อย่างไรก็ตามในการสร้าง Join Processing Tree เพื่อแทนแต่ละ QEPs อาจมีบาง Join Processing Tree ที่ไม่สามารถใช้งานได้ ซึ่งจะเรียกกรณีนี้ว่า Invalid QEPs เนื่องจากการจัดลำดับของการ Join ระหว่าง Relations ที่ไม่เหมาะสม ซึ่งผลลัพธ์ที่ได้จะเรียกว่า “ผลคูณคาร์ทีเชียน” (Cartesian product) ซึ่งเกิดจากการดำเนินการ Join ระหว่าง Relation ที่ไม่ได้มีความสัมพันธ์ใดๆ ต่อกันยกตัวอย่างได้ดังภาพที่ 5



ภาพที่ 5 Invalid QEP (Cartesian Products)

จากภาพที่ 5 เมื่อดำเนินการ Join ระหว่าง Relation Cust และ Item แล้วจะได้ผลลัพธ์ที่ได้เป็น Cartesian product ซึ่งเป็นผลลัพธ์ที่มีขนาดใหญ่มากเกินความจำเป็น

ชนิดของ Join Processing Trees

ในแต่ละชุดของการดำเนินการบน Join Processing Tree นั้นแต่ละ Operation Node จะประกอบไปด้วย 2 Operand Node คือ Inner Node และ Outer Node โดยที่ Outer Node คือ Node ที่แทน Relation ที่มี Key หลัก (Primary Key) ที่ใช้สำหรับในการอ้างถึงความสัมพันธ์ใน Relation อื่นๆ และ Inner Node จะแทน Relations ที่ถูกอ้างถึงโดย Outer Node นั่นเอง

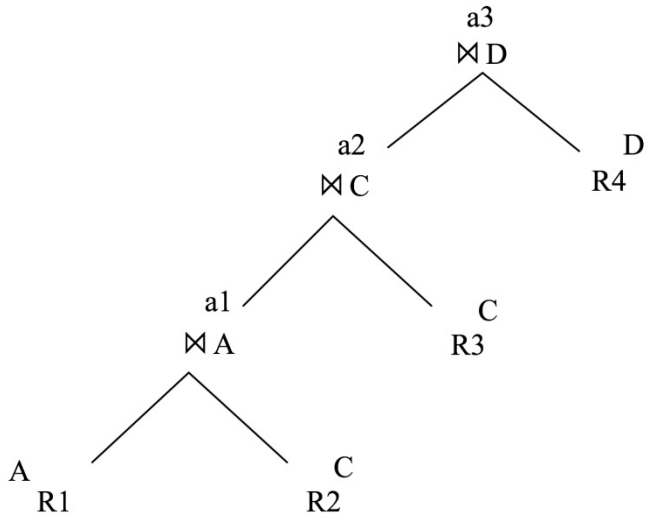
เราสามารถแบ่งแยกชนิดของ Join Processing Trees (Pongpinigpinyo 1996) ออกเป็นประเภทได้ดังต่อไปนี้

1. Linear Join Processing Tree สำหรับ Tree ชนิดนี้จะต้องมี Node ที่แทน Base Relation และทำหน้าที่เป็น Inner Node เสมอ

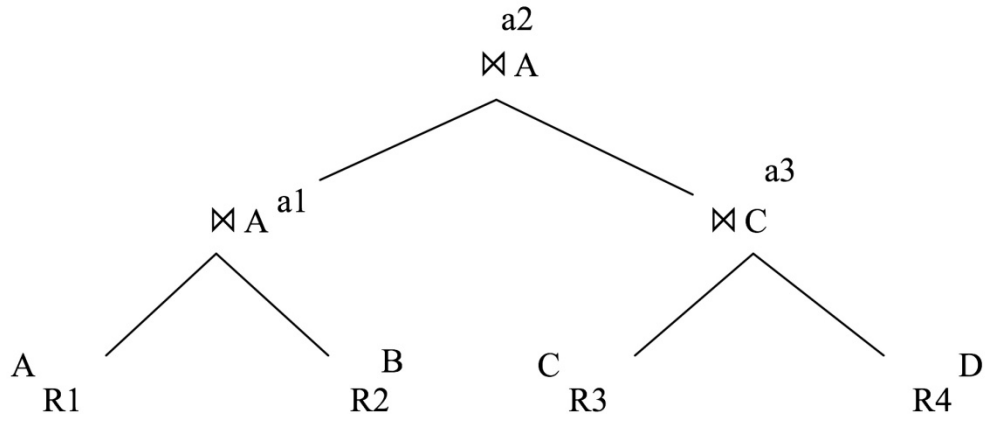
2. Bushy Join Processing Tree สำหรับ Tree ชนิดนี้ภายในจะมี Sub Trees ย่อยที่เป็นการดำเนินการระหว่าง Node ที่ไม่ได้เป็น Base Relations (None-Leaf Nodes) ทั้งสอง Nodes ซึ่งการคำนวณผลลัพธ์ที่จะเกิดขึ้นจากการดำเนินการระหว่างทั้งสอง Node นั้นจะต้องรอผลลัพธ์ที่ได้จากการดำเนินการก่อนหน้านั้น

สามารถทำการยกตัวอย่างรูปแบบของ Linear Join Processing Tree และ Bushy Join Processing Tree ได้ดังภาพที่ 6 และ 7 โดยกำหนดให้ R1, R2, R3, R4 คือ Relations ที่ถูกจัดเก็บไว้ที่ Site A, B, C, D ตามลำดับ

```
SELECT      *
FROM        R1, R2, R3, R4
WHERE       R1.a1 = R2.a2 and R2.a2=R3.a2 and R3.a3=R4.a3
```



ภาพที่ 6 ตัวอย่างของ Linear Join Processing Tree ที่เกิดขึ้น



ภาพที่ 7 ตัวอย่างของ Bushy Join Processing Tree ที่เกิดขึ้น

วิธีการคำนวณจำนวน Valid Query Execution Plans

ขนาดของ Search space ที่เกิดขึ้นจะขึ้นอยู่กับจำนวนของ Query Execution Plans ที่เป็นไปได้ทั้งหมด โดยที่จำนวนของ QEPs ที่เป็นไปได้ทั้งหมดนั้นจะประกอบไปด้วยทั้ง Valid QEPs และ Invalid QEPs โดยสำหรับ Invalid QEPs นั้นจะทำให้เกิดผลลัพธ์ของการดำเนินการ Join ที่เป็นผลคูณคาร์ทีเชียน (Cartesian Products) ซึ่งเป็นสิ่งที่ควรหลีกเลี่ยงอย่างยิ่งสำหรับกระบวนการ Query Optimization เมื่อพิจารณาในจุดนี้เราจึงจำเป็นอย่างยิ่งที่จะต้องหลีกเลี่ยงหรือตัด Invalid QEPs ออกไปจาก Search space ที่เราทำการพิจารณาซึ่งจะส่งผลทำให้ Search space ที่พิจารณามีขนาดเล็กลงไปด้วย ซึ่งจะทำให้การค้นหา QEP ที่ดีที่สุด เป็นไปได้อย่างรวดเร็วยิ่งขึ้น

จำนวนของ QEPs ที่เกิดขึ้นนั้นจะขึ้นอยู่กับวิธีการสำหรับการ Query ที่ใช้สำหรับดำเนินการ โดยเราสามารถทำการคำนวณเพื่อให้ทราบว่าจะมี QEPs เกิดขึ้นจำนวนเท่าใดที่จะเกิดขึ้นจากการ Query ในรูปแบบต่างๆ ซึ่งจะทำให้เราสามารถทราบถึงขนาดของ Search space ที่จะเกิดขึ้น โดยการ Query ที่ใช้สำหรับดำเนินการจะมี 2 รูปแบบคือ Linear Query และ Star Query

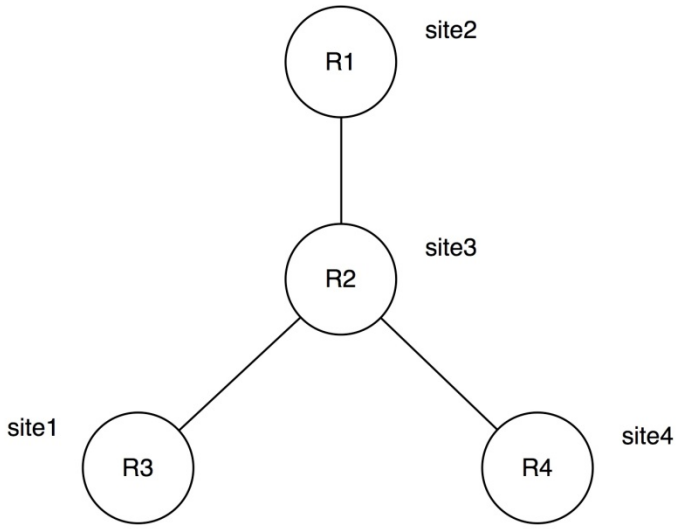
Linear Query ลักษณะของ Join Graph จะเป็นเส้นตรงโดยประกอบด้วย n Relations และ n-1 Edge ที่เชื่อมต่อ Relations ต่างๆ เข้าด้วยกัน โดยเราสามารถที่จะคำนวณจำนวน Valid QEPs ที่เกิดจากการ Join แบบ Linear Query ได้จากสูตร (Ozsu and Valduriez 1991)

$$m(2n - 2)! / n!(n - 1)! \qquad (1)$$

จากสูตรการคำนวณที่ (1) กำหนดให้ m คือจำนวน Site ที่มี และ n คือจำนวน Relations โดยเราสามารถแสดงให้เห็นถึงลักษณะของการ Join แบบ Linear Query ได้ดังภาพที่ 8

ภาพที่ 8 Linear Graph ที่ประกอบไปด้วย N Relations

Star Query ลักษณะของ Graph นั้นจะอยู่ในรูปของ Star โดยจะมี Relations ใด Relation หนึ่งที่ทำหน้าที่เป็น Hub และแตกสาขาอยู่ใน Graph ซึ่งสามารถแสดงให้เห็นได้ดังภาพที่ 9



ภาพที่ 9 Star Graph ที่ประกอบไปด้วย 4 Relations

ในกรณีที่มีการดำเนินการแบบ Star Query เราสามารถทำการคำนวณจำนวนของ Valid QEPs ได้จากสูตรการคำนวณที่ (2) ดังต่อไปนี้ (Ozsu and Valduriez 1991)

$$m\,(n-1)! \qquad (2)$$

หากเราพิจารณาเพียงแค่จำนวนของ Valid QEPs ที่เกิดขึ้นจากวิธีการ Query ที่แตกต่างกันทั้ง 2 แบบ จะเห็นได้ว่าจำนวน Valid QEPs ที่เกิดจากการดำเนินการแบบ Star Query นั้นมีจำนวนมากว่าการดำเนินการแบบ Linear Query ดังนั้นสำหรับงานวิจัยชิ้นนี้ทางผู้วิจัยจึงได้ทำการพิจารณาแค่การดำเนินการแบบ Linear Query เพราะขนาดของ Search space ที่เกิดขึ้นนั้นมีขนาดเล็กว่าการดำเนินการแบบ Star Query จึงทำให้สามารถค้นหาคำตอบได้รวดเร็วกว่า และสำหรับกระบวนการ Query Optimization โดยทั่วไปนั้นก็นิยมใช้การดำเนินการแบบ Linear Query สำหรับค้นหา QEPs ที่ดีที่สุดเช่นกัน

Database Statistics

ในกระบวนการ Query Optimization การที่จะสามารถคำนวณค่า Cost ที่จะเกิดขึ้นในแต่ละ QEPs นั้นเราจำเป็นที่ต้องใช้ข้อมูลต่างๆ ในหลายๆ ด้านเพื่อใช้สำหรับการคำนวณ ตัวอย่างเช่น ข้อมูลทางด้าน Network ข้อมูลทางการประมวลผลของแต่ละ Site และ ข้อมูลต่างๆ ของแต่ละ Relations เป็นต้น ซึ่งข้อมูลเหล่านี้จะถูกเก็บไว้เป็นข้อมูลสถิติอยู่ในระบบ โดยข้อมูลเหล่านี้นั้นจะมีการเปลี่ยนแปลงไปเรื่อยๆ ตามสภาพความเป็นจริงของระบบ โดยตัวระบบ DBMS จะเป็นผู้ที่เรียนรู้สภาพแวดล้อมของการทำงานเหล่านั้นและนำมาสร้างเป็นข้อมูลสถิติเก็บเอาไว้ภายในระบบ ดังนั้นในการคำนวณเราจึงสามารถที่จะนำข้อมูลเหล่านี้มาใช้สำหรับการคำนวณเพื่อหาผลลัพธ์ที่ต้องการได้เลย

องค์ประกอบหลักที่มีผลอย่างมากต่อประสิทธิภาพของวิธีการ Execute ต่างๆ ที่เป็นไปได้นั้น คือ ขนาดของผลลัพธ์ชั่วคราวที่เกิดขึ้นจากการดำเนินการ Join ระหว่างคู่ของ Relations ใดๆ หรือที่เรียกว่า Intermediate Relations ซึ่งจะเป็น Relations ที่เกิดขึ้นในระหว่างการทำงาน ในกรณีที่เป็นการ Join ระหว่างคู่ของ Relations ใดๆ ที่อยู่คนละ Site กันจะมีความจำเป็นที่จะต้องทำการส่งข้อมูลระหว่าง Sites เกิดขึ้น ถ้าหากขนาดของ Intermediate Relations มีขนาดใหญ่มากๆ ก็จะทำให้ต้องสูญเสียค่า Cost ในการขนส่งข้อมูลระหว่าง Site มากขึ้นเช่นกัน และในกรณีที่เกิด Intermediate Relations มีขนาดใหญ่แล้วสำหรับในการประมวลผลข้อมูลก็จะต้องสูญเสีย Cost ที่สูงขึ้นเช่นกัน ดังนั้น QEPs ที่ให้ประสิทธิภาพสำหรับการ Execution ที่ดีที่สุด จึงควรที่จะก่อให้เกิด Intermediate Relations ที่มีขนาดเล็กในระหว่างการดำเนินการจนเสร็จสิ้นกระบวนการ ดังนั้นจึงมีความจำเป็นอย่างยิ่งที่ต้องสามารถทำการคำนวณหาขนาดของ Intermediate Relations ที่จะเกิดขึ้นในระหว่างการดำเนินการ เพื่อใช้เป็นข้อมูลสำหรับการคำนวณหาค่า Cost ที่เกิดขึ้นจากแต่ละ QEPs โดยการคำนวณขนาดของ Intermediate Relations นั้น เราสามารถคำนวณได้โดยใช้ข้อมูลสถิติ (Database Statistics) มาเพื่อคำนวณได้

ดังนั้นจะเห็นได้ว่าสิ่งสำคัญอีกอย่างหนึ่งที่มีผลต่อความถูกต้องของผลลัพธ์ที่ได้จากการทำงานในกระบวนการ Query Optimization ก็คือความสามารถในการเรียนรู้และสร้างข้อมูลทางด้านสถิติของระบบ (Database Statistics) โดยตัว DBMS เพราะข้อมูลเหล่านี้จะเป็นตัวแปรสำคัญที่ใช้ในการคิดและคำนวณเพื่อหาคำตอบในกระบวนการ Query Optimization นั่นเอง

สำหรับในงานวิจัยชิ้นนี้ได้นำเอาข้อมูล Database Statistics เข้ามาใช้สำหรับการคำนวณ มีดังต่อไปนี้ (Ozsu and Valduriez 1991)

Join Selectivity คืออัตราส่วนของจำนวน Cardinality ของผลลัพธ์ที่เกิดขึ้นต่อจำนวน Cardinality ทั้งหมดภายใน Set ของ Cartesian product ที่เกิดขึ้น โดยสามารถแสดงได้จากสูตรการคำนวณดังสมการที่ (3) ดังนี้ (Ozsu and Valduriez 1991)

$$\text{Join Selectivity} = \text{Card}(S \bowtie_{a=b} T) / \text{Card}(S) \times \text{Card}(T) \quad (3)$$

Cardinality คือจำนวนของ Cardinality ของผลลัพธ์ที่เกิดจากการดำเนินการ Join แบบ Equijoin โดยกำหนดให้ Card($S \bowtie_{a=b} T$) แสดงจำนวนของ Cardinality ของผลลัพธ์ของการ Join แบบ Equijoin ระหว่าง Relation S ซึ่งมี Attribute a กับ Relation B ซึ่งมี Attribute b โดยสามารถแสดงสูตรการคำนวณเพื่อหาจำนวน Cardinality ของผลลัพธ์ที่เกิดขึ้นได้ดังสูตรการคำนวณดังสมการที่ (4) ดังต่อไปนี้ (Ozsu and Valduriez 1991)

$$\text{Card}(S \bowtie_{a=b} T) = \text{Card}(S) \times \text{Card}(T) \times \text{Join Selectivity} \quad (4)$$

สำหรับการหาจำนวน Cardinality ของ Intermediate Relations อาจทำได้อย่างง่ายๆ อีกวิธีหนึ่งก็คือ หากกำหนดให้ Attribute A เป็น Primary Key ของ Relation S และ Attribute B เป็น Foreign Key ของ Relations T ที่ใช้เชื่อมต่อกับ Relation S เราสามารถคำนวณหาจำนวนของ Cardinality ของ Intermediate Relation ที่เกิดขึ้นจากการ Join กันของ Relation S และ T ได้ดังสมการที่ (5) ดังนี้ (Ozsu and Valduriez 1991)

$$\text{Card}(S \bowtie_{a=b} T) = \text{Card}(T) \quad (5)$$

หากพิจารณาจากสมการที่ (5) จะเห็นได้ว่าทุกครั้งที่เราต้องการที่จะคำนวณจำนวนของ Cardinality ของ Intermediate Relation เราจะต้องทราบทุกครั้งว่า Relation ใดมี Primary Key และ

Relation ใดมี Foreign Key ดังนั้นก่อนทำการ Join ระหว่าง Relations ใดๆ จะต้องทำการตรวจสอบหาความสัมพันธ์ของ Relations ทุกครั้งเพื่อที่จะกำหนดว่า Relations ใดทำหน้าที่เป็น Outer Relations และ Relation ใดทำหน้าที่เป็น Inner Relations เพื่อเป็นการแก้ปัญหาดังกล่าวและเป็นการลดเวลาที่ใช้ในการตรวจสอบความสัมพันธ์ของ Relations จึงได้มีการนำข้อมูลทางสถิติ (Database Statistics) ที่มีชื่อว่า Join Selectivity Factor เข้ามาช่วยสำหรับการคำนวณหาจำนวน Cardinality ของ Intermediate Relations ดังที่แสดงไว้ในสมการที่ (4)

Distributed Cost Model

Cost Model ของ Query Optimizer นั้นจะประกอบไปด้วย Cost Function ที่ใช้สำหรับคำนวณค่า Cost ที่จะเกิดจากการดำเนินการ หรือการคำนวณเพื่อหาขนาดของ Intermediate Relations เป็นต้น

Cost Functions

สำหรับค่า Cost ของการทำงานแบบ Distributed Execution สามารถที่จะนำเสนอออกมาได้ในสองรูปแบบคือ เวลารวมทั้งหมดที่ต้องเสียไปสำหรับการทำงานของแต่ละส่วน (Total Time) หรือ เวลาที่ต้องเสียไปจาการทำงานจริงๆ ตั้งแต่เริ่มต้นจนจบกระบวนการ (Response Time) ซึ่งโดยทั่วไปแล้ว Response Time นั้นจะเริ่มวัดตั้งแต่เริ่มรับคำสั่ง Input เข้ามาจนกระทั่งเสร็จสิ้นการแสดงผลลัพธ์ที่ได้ และโดยปรกติแล้ว Response Time จะมีค่าน้อยกว่า Total Time เราบางกรณีในการทำงานจริงการทำงานอาจเป็นไปในรูปแบบของ Parallel เนื่องจาก Relations ที่อยู่กันคนละ Site สามารถที่จะทำการประมวลผลได้แบบ Parallel ได้เลยโดยที่ไม่จำเป็นจะต้องรอผลลัพธ์ระหว่างกัน ดังนั้นจึงอาจกล่าวได้ว่าหาก QEPs ใดมี Total Time น้อยก็จะมี Response Time ที่น้อยเช่นกัน ดังนั้นสำหรับงานวิจัยชิ้นนี้ได้เลือกใช้ Total Time สำหรับบ่งบอกค่า Cost ที่เกิดขึ้นจากแต่ละ QEPs เนื่องจากสามารถที่จะทำการคำนวณได้ง่ายกว่าการคำนวณ Response Time

สูตรการคำนวณโดยทั่วไปเพื่อใช้หาค่า Total Time สำหรับการทำงานแบบ Distribute Execution สามารถแสดงได้ดังนี้ (Ozsu and Valduriez 1991)

$$Total\_time = TCPU * \#insts + TI/O * \#I/Os + TMSG * \#msgs + TTR * \#bytes \quad (6)$$

จากสมการที่ (6) กำหนดให้

#insts	คือ จำนวนคำสั่งในโปรแกรมที่ต้องถูกประมวลผล
#I/Os	คือ จำนวนของข้อมูลที่ต้องถูกขนส่งออกไป หรือว่ารับเข้ามา
#bytes	คือ ขนาดของข้อมูลที่ถูกขนส่งออกไป หรือว่ารับเข้ามา
TCPU	คือ เวลาที่ใช้สำหรับการประมวลต่อ 1 คำสั่งของ CPU
TI/O	คือ เวลาใช้สำหรับการทำงานของอุปกรณ์ I/O
TTR	คือ เวลาที่ใช้สำหรับการขนส่งข้อมูล

Cartesian product

ในระบบ Relational Database นั้นถ้าหากมีการดำเนินการ Join ระหว่าง Relations ใดๆ ที่ไม่ได้มีความสัมพันธ์ระหว่างกันจะทำให้เกิดผลลัพธ์ที่เรียกว่า “ผลคูณคาร์ทีเชียน” (Cartesian product) ซึ่งเป็นรวมกันของข้อมูลทั้งหมดภายใน 2 Relations นั้นเข้าด้วยกัน ดังนั้นขนาดของผลลัพธ์ที่ได้จะมีขนาดเท่ากับจำนวนของ Cardinality ของทั้งสอง Relations นั้นคูณกัน ทำให้ผลลัพธ์ที่ได้มีขนาดใหญ่มาก ซึ่งจะทำให้ต้องสูญเสียเวลาในการประมวลผลข้อมูล และเวลาในการขนส่งข้อมูลอย่างมากตามมา

ดังนั้นในกระบวนการ Query Optimization จึงมีความจำเป็นอย่างยิ่งที่จะต้องหลีกเลี่ยงไม่ให้เกิดผลลัพธ์ที่เป็น Cartesian product และจะถือว่า QEPs ที่ก่อให้เกิดผลลัพธ์ที่เป็น Cartesian product นั้นเป็น Invalid QEPs

ปัญหาของการ Search ในกระบวนการ Query Optimization

สำหรับการค้นหาและตัดสินว่า QEPs ใดแสดงถึงวิธีการ Execute ที่ดีที่สุดนั้นสามารถทำได้โดยใช้ Cost Function ในการคิดค่า Total Time ที่จะเกิดขึ้นจากแต่ละ QEPs และเลือก QEP ที่ให้ค่า Total Time ที่มีค่าน้อยที่สุดเป็นคำตอบที่ดีที่สุด

โดยปรกติทั่วไปสำหรับกระบวนการ Query Optimization จะใช้วิธีการค้นหาแบบละเอียด (Exhaustive Search) สำหรับค้นหาว่า QEPs ใดที่ให้ค่า Total Time ที่น้อยที่สุด ซึ่งวิธีการค้นหาแบบนี้เป็นวิธีการที่มีประสิทธิภาพมาก โดยจะให้คุณภาพของผลลัพธ์ที่มีความถูกต้องถึง 100 เปอร์เซ็นต์ แต่ขนาดของ Search space หรือจำนวนของ QEPs จะมีจำนวนมากขึ้นไปเรื่อยๆ ตามจำนวนของ Relations ที่ถูกดำเนินการโดยการเติบโตของ Search space จะเป็นแบบ Factorial ดังนั้นการค้นหา QEP ที่ให้ค่า Total Time ที่น้อยที่สุดจึงเป็นไปได้อย่างล่าช้ามากยิ่งขึ้น ถึงแม้ว่าคุณภาพของผลลัพธ์จะมีความถูกต้องถึง 100 เปอร์เซ็นต์ก็ตาม แต่ก็ต้องสูญเสียเวลาเพิ่มขึ้นไปเรื่อยๆ ตามจำนวนของ Relations ที่เพิ่มขึ้นจนไม่สามารถที่จะรอคอยคำตอบได้

ดังนั้นในขั้นตอนการค้นหา QEPs ที่ดีที่สุดในกรณีที่มีจำนวนของ Relations ที่ถูกดำเนินการเป็นจำนวนมาก คือตั้งแต่ 10 Relations ขึ้นไปจัดว่าเป็นปัญหา NP-Complete เนื่องจากต้องใช้เวลาในการค้นหามากขึ้นไปเรื่อยๆ ตามจำนวนของ Relations ที่เพิ่มขึ้น การค้นหา QEP ที่ดีที่สุดในกรณีที่มีการ Join จำนวนมาก (Large Join Query) จึงไม่สามารถใช้วิธีการค้นหาแบบ Exhaustive Search เพื่อค้นหาได้ และเพื่อเป็นการแก้ปัญหาในเรื่องของวิธีในการค้นหา (Search Strategy) ที่เกิดขึ้นดังกล่าวได้มีการนำวิธีการค้นหาแบบ Randomize Algorithms เข้ามาแทนที่การค้นหาแบบ Exhaustive Search เพื่อให้สามารถทำการค้นหา QEPs ที่ดีที่สุดได้อย่างรวดเร็วยิ่งขึ้นและอยู่ภายใต้เวลาที่สามารถที่จะรอคอยคำตอบได้

อัลกอริทึมอบเหนียวจำลอง (Simulate Annealing Algorithm: SA)

กระบวนการอบเหนียวจำลอง (บุญเสริม กิจศิริกุล 1996) ถูกออกแบบมาให้สามารถหลุดออกจากสถานะที่ดีที่สุดเฉพาะที่ได้ โดยใช้แนวคิดของอุณหพลศาสตร์ของกระบวนการอบเหนียว ซึ่งเป็นการลดอุณหภูมิลงอย่างช้าๆ ระหว่างการหลอมเพื่อให้ได้โลหะที่อยู่ในสภาวะที่เหมาะสมที่สุด เป็นโลหะเหนียว ไม่เปราะ

แนวคิดการทำงานของ SA นั้นจะยอมให้ทิศทางของการค้นหาเป็นไปในทิศทางที่ไม่ดีในช่วงเริ่มต้นของกระบวนการค้นหาเพื่อเป็นการสำรวจทั่วๆ แบบหยาบๆ ก่อน แล้วจึงค่อยทำการค้นหาแบบละเอียดเมื่อเวลาผ่านไป ด้วยแนวคิดเช่นนี้ทำให้เราคาดหวังว่า ผลลัพธ์สุดท้ายที่ได้จะไม่ขึ้นอยู่กับสถานะแรกของการค้นหามากนัก เพื่อให้เข้าใจถึงอัลกอริทึมนี้จึงขออธิบายถึงการอบเหนียวของโลหะโดยย่อดังต่อไปนี้

การอบเหนียวโลหะเป็นการหลอมโลหะจนละลาย โดยการทำให้โลหะอยู่ในสถานะที่มีพลังงานสูง แล้วค่อยๆ ลดอุณหภูมิลงมาทีละน้อยจนโลหะเปลี่ยนสถานะกลับมาเป็นของแข็ง จุดมุ่งหมายคือทำให้โลหะกลับมาเป็นของแข็งในสถานะสุดท้ายที่มีพลังงานต่ำสุด โดยทั่วไปในธรรมชาติ สสารจะพยามเปลี่ยนตัวเองกลังมาจากสถานะที่มีพลังงานสูงไปสู่สถานะที่มีพลังงานต่ำ แต่ก็มีความน่าจะเป็นที่สสารเปลี่ยนจากพลังงานต่ำไปสู่พลังงานสูงอยู่บ้าง ความน่าจะเป็นนี้สามารถอธิบายได้จากสมการความน่าจะเป็นที่ (7) ดังนี้คือ

$$p = e^{-\Delta_{E/kT}} \qquad (7)$$

โดยที่

ΔE คือ ระดับพลังงานทีเปลี่ยนไป (เป็นค่าบวก)

T คือ อุณหภูมิ

K คือ ค่าคงที่ของโบลต์ซมันน์ (Boltzmann)

การเปลี่ยนแปลงจากพลังงานสูงไปต่ำนั้น มีความน่าจะเป็นที่จะเกิดในช่วงเริ่มต้นมากกว่าช่วงปลายของการอบเหนียวอัตราการลดอุณหภูมิของการอบเหนียวเรียกว่า “หมายกำหนดการอบเหนียว (Annealing Schedule)” ถ้าหมายกำหนดการอบเหนียวเร็ว กล่าวคือลดอุณหภูมิอย่างรวดเร็ว โลหะก็มีโอกาสเข้าสู่สถานะสุดท้ายที่มีพลังงานสูงอยู่ เราจึงต้องพิจารณาหมายกำหนดการอบเหนียวไม่ให้เร็วเกินไปแต่ถ้าใช้ไปก็ทำให้เสียเวลาโดยไม่จำเป็นเช่นกัน

แนวคิดของการอบเหนียวจำลองก็ได้มาจากการล้อเลียนการอบเหนียวของโลหะ โดยการเป็นกาค้นหาแบบลงเหวแบบหนึ่ง ซึ่งการค้นหาสามารถเป็นไปในทิศทางที่ไม่ดีได้โดยเฉพาะ

ในช่วงต้นของการค้นหา เพื่อสำรวจหาบริเวณที่น่าจะนำไปสู่คำตอบที่เป็นคำตอบที่ดีที่สุดได้ โดยสามารถแสดงสมการความน่าจะเป็นที่ใช้สำหรับกำหนดทิศทางในการค้นหาของ Simulated Annealing ได้ดังสมการที่ 8 ดังต่อไปนี้

$$p = e^{-\Delta_{E / T}} \tag{8}$$

โดยกำหนดให้

ΔE คือ ค่าฮิวริสติกที่เปลี่ยนไป (เป็นค่าบวก)

T คือ อุณหภูมิ

สมการนี้เมื่อนำไปใช้ร่วมกับวิธีการค้นหาแบบปีนเขา ก็จะช่วยให้กระบวนการค้นหาสามารถหลุดออกจากค่าที่ดีที่สุดเฉพาะที่ได้ตรงกับความต้องการของเรา ตัวอย่างเช่น เมื่อเราอยู่ในสถานะปัจจุบันที่มีค่าฮิวริสติกเท่ากับ A และเมื่อสร้างสถานะลูกที่มีค่า ฮิวริสติกเท่ากับ B ซึ่งมีค่าที่แย่ลง การค้นหาอาจไปยังสถานะลูกนี้ได้โดยการคำนวณค่าความน่าจะเป็น (p) ตามสมการที่ (8) ได้เป็น $e^{-|A-B| / T}$ ค่าที่ได้นี้จะมีค่าอยู่ระหว่าง 0 - 1 จากนั้นเราจะสุ่มตัวเลข (Random(0,1)) ขึ้นมาหนึ่งตัว ถ้าหากว่าค่า p มีค่ามากกว่าก็จะยอมรับสถานะลูก B เป็นสถานะต่อไปได้ สำหรับค่า Parameter T นั้นเป็น Parameter ของ Algorithm ซึ่งเราสามารถทำการปรับแต่งให้มีความเหมาะสมกับปัญหาแต่ละปัญหาที่เราสนใจได้ โดยเริ่มต้นนั้นเราสามารถกำหนดให้มีค่าที่มากๆ ได้และค่อยๆ ลดลงไปเรื่อยๆ เมื่อการทำงานผ่านไป

เมื่อพิจารณาจากสมการที่ (8) เราจะได้คุณสมบัติที่เราต้องการดังนี้คือ เมื่อมี T มาก (ในช่วงต้นของกระบวนการค้นหา) จะได้ p มีค่ามาก คือเรายอมให้ทิศทางในการค้นหาเป็นไปในทิศที่ไม่ดีได้ง่ายในช่วงต้นของการค้นหา แต่เมื่อ T น้อย (ในช่วงปลายของการค้นหา) จะได้ p ที่มีค่าน้อยคือการเริ่มลู่เข้าสู่คำตอบ จึงไม่ควรให้การค้นหากระโดดไปยังทิศทางที่แย่ลง เมื่อพิจารณา ΔE มีค่ามากจะได้ค่า p มีค่าน้อย กล่าวคือถ้าการก้าวกระโดดจากสถานะที่ดีไปยังสถานะที่ไม่ดีการกระโดดนั้นเป็นก้าวใหญ่ (เมื่อมีการเปลี่ยนแปลงค่าฮิวริสติกมาก) และจะเกิดขึ้นได้ยากกว่าการก้าวกระโดดที่เป็นก้าวเล็ก (เมื่อมีการเปลี่ยนแปลงค่าฮิวริสติกน้อย) ซึ่งเป็นคุณสมบัติที่น่าพอใจ

เนื่องจากเรามีความเชื่อว่า สถานะที่เป็นคำตอบจะต้องอยู่ในหลุมลึกๆ ถ้าจะหลุดออกจากหลุมลึกๆ จะต้องมีการก้าวกระโดดแบบก้าวใหญ่ๆ ซึ่งไม่ควรให้เกิดขึ้นได้ และสถานะที่ดีที่สุดเฉพาะที่มักเป็นหลุมตื้นๆ จึงมีโอกาสหลุดออกมาได้ง่ายหน่อย โดยสามารถแสดง Algorithm ของ Simulated Annealing ที่ใช้ในงานวิจัยนี้ได้ดังภาพที่ 10 (Ioannidis and Kang 1990)

```
1   Procedure SA() {
2     S = S0,
3     T = T0,
4     minS = S
5     while not (frozen) do {
6         while not (equilibrium) do {
7                 S' = random state in neighbors(S),
8                 ΔC = cost(S') – cost(S),
9                 if (ΔC <= 0) then S = S',
10                if (ΔC > 0) then S = S' with probability e^(-Δ_E / T) ,
11                if cost(S) < cost(minS) then minS = S,
12        }
13        T = reduce(T),
14    }
15    return(minS),
16  }
```

ภาพที่ 10 ลำดับและขั้นตอนการทำงานของ Simulated Annealing Algorithm

Iterative Improvements (II)

เป็น Algorithm ที่มีลักษณะการทำงานแบบลงเขาเพียงอย่างเดียว โดยจะเริ่มทำงานจากการสุ่ม (Random) State แรกขึ้นมาจาก State space หรือว่า Search space เพื่อใช้สำหรับเป็นจุดเริ่มต้นของการค้นหา และจากนั้นจะทำการพัฒนาคำตอบโดยการสุ่ม State ต่อๆ ไปขึ้นมาแล้วทำการเปรียบเทียบกับ State ตั้งต้นถ้าหาก State ใดที่ถูกสุ่มขึ้นมาให้ค่าฮิวริสติกที่ดีกว่า State ตั้งต้น ก็จะยอมรับให้ State นั้นเป็น State ต่อไปและใช้ State ที่ดีว่านั้นเป็น State ตั้งต้นสำหรับการค้นหา

ต่อๆ ไป โดยการทำงานจะเป็นการกระทำซ้ำๆ ต่อไปเรื่อยๆ จนกว่าจะเจอค่าที่ทำให้เข้าสู่เงื่อนไขในการหยุดการทำงาน (Stop Conditions)

โดยสามารถแสดงให้เห็นถึงลำดับและขั้นตอนในการทำงานของ II ที่ได้ถูกใช้ในงานวิจัยนี้ได้จากภาพที่ 11 (Ioannidis and Kang 1990) โดย Loop ภายในสุดที่อยู่ใน Algorithm นั้นเรียกว่า Local Optimization โดย Local Optimization จะเริ่มต้นโดยการสุ่มเลือก จากนั้นจะเป็นการทำงานโดยการวนรอบเพื่อสุ่มเลือก State ต่อๆ ไปขึ้นมาเพื่อเปรียบเทียบ และจะยอมรับ State ที่สุ่มขึ้นมาก็ต่อเมื่อมีค่าที่น้อยกว่า State ตั้งต้นเท่านั้น และจะกระทำต่อไปเรื่อยๆ เพื่อให้การค้นหาวิ่งลงไปสู่ค่าที่เป็นค่าที่ดีที่สุด หรือว่าเป็นค่าที่เรียกว่า Global Minimum

```
1   Procedure II() {
2     minS = S∞,
3     while not(Stoping_Condition) do {
4           S = random state,
5           while not (local_minimum(S)) do {
6                       S' = random state in neighbors(S),
7                       if cost(S') < cost(S) then S = S',
8           }
9           if cost(S) < cost(minS) then minS = S,
10    }
11     return(minS),
12  }
```

ภาพที่ 11 ลำดับและขั้นตอนการทำงานของ Iterative Improvement Algorithm

แต่ด้วยวิธีการทำงานของ II ที่เป็นรูปแบบง่ายๆ และมีพื้นฐานเป็นแบบ Greedy Algorithm ถึงแม้ว่าจะสามารถหาคำตอบออกมาได้อย่างรวดเร็ว แต่ก็ยังมีข้อเสียอีกอย่างที่สำคัญก็คือ คุณภาพของผลลัพธ์ที่ได้จาก II นั้นไม่สามารถที่จะลู่เข้าสู่คำตอบที่คำตอบที่ดีที่สุดในวงกว้างได้ (Global Minimum) แต่คำตอบที่ได้จะเป็นเพียงคำตอบที่ดีที่สุดเฉพาะที่ (Local Minimum)

Two Phase Optimization (2PO)

เป็น Algorithm ที่ถูกนำเสนอโดย Ioannidis and Kang (1990) ซึ่งเป็น Algorithm ที่มีการทำงานแบบผสมระหว่าง Iterative Improvement Algorithm กับ Simulate Annealing Algorithm โดยแยกการทำงานออกเป็น 2 Phase โดยกำหนดให้ Phase เป็นการทำงานของ Iterative Improvement Algorithm ซึ่งจะทำงานด้วยเวลาที่จำกัด จากนั้นจะเข้าสู่การทำงานของ Phase ที่สอง ซึ่งเป็นการทำงานของ Simulated Annealing Algorithm โดยการนำผลลัพธ์ที่ได้จาก Phase แรกมาให้เป็น State ตั้งต้นในการค้นหาของ Simulated Annealing Algorithm เพื่อใช้หาผลลัพธ์สุดท้ายต่อไป

จุดประสงค์ของการนำเสนอ 2PO ก็เพื่อต้องการจะพัฒนาการทำงานของ SA ให้สามารถที่จะลู่เข้าสู่คำตอบให้ได้อย่างรวดเร็วยิ่งขึ้น เนื่องจากข้อเสียที่สำคัญของ SA นั้นก็คือต้องใช้เวลามากในการค้นหาคำตอบ และต้องให้เวลาที่เหมาะสมเพื่อให้ SA สามารถลู่เข้าสู่คำตอบที่เป็นคำตอบที่ดีที่สุดในวงกว้าง (Global Minimum) ได้ จึงได้มีแนวคิดว่าถ้าในการค้นหาแต่ละครั้งของ SA เริ่มต้นด้วยจุดเริ่มต้นที่เป็นค่าน้อยที่สุดเฉพาะที่ (Local minimum) ซึ่งเป็นจุดที่มีค่าฮิวริสติกที่น้อยก็จะเป็นการกำหนดให้ค่าเวลาเริ่มต้นที่ใช้ในการค้นหา หรือค่า Parameter T น้อยตามไปด้วย และเป็นจุดที่อยู่ห่างจากค่าน้อยที่สุดในวงกว้าง (Global minima) ไม่มากนัก การค้นหาของ SA จึงน่าที่จะสามารถลู่เข้าสู่คำตอบที่เป็น Global minima ได้อย่างรวดเร็วมากยิ่งขึ้น

Genetic Algorithm

เป็นวิธีในการค้นหาแบบหนึ่งที่ใช้สำหรับการค้นหา โดยเป็นการพัฒนาและจำลองวิธีมาจากกระบวนการทางพันธุกรรมของสิ่งมีชีวิตจาก “ทฤษฎีวิวัฒนาการ” ของ Charles Darwin โดยนักวิทยาศาสตร์ สาขาวิทยาการคอมพิวเตอร์ ชื่อว่า John Holland ได้ทำการค้นคิดการลอกเลียนแบบขั้นตอนธรรมชาติของการพัฒนาสิ่งมีชีวิตขึ้นในปี 1970 โดยมีจุดมุ่งหมายในการพัฒนาเพื่อ

1. อธิบายการเปลี่ยนแปลงกระบวนการทางธรรมชาติของพันธุกรรม
2. เพื่อที่จะนำการกลไกลการเปลี่ยนแปลงเหล่านี้มาประยุกต์กับการเขียนโปรแกรม

จากการคิดค้นนี้ของ John Holland ทำให้สามารถค้นหาจุดที่เป็นจุดเหมาะสมที่สุด ไม่ว่าจะเป็นจุดต่ำสุด (Minimum Point) หรือจุดสูงสุด (Maximum Point) โดยหลักการในการค้นหาของ Genetic Algorithm นั้นก็คือ สิ่งมีชีวิตทั้งหมดจะมีทั้งลักษณะที่ดีและไม่ดี แต่สำหรับสิ่งมีชีวิตที่ดีนั้นจะได้รับการสนับสนุนให้ได้รับการถ่ายทอดลักษณะทางพันธุกรรม เพื่อให้ได้สิ่งมีชีวิตใหม่ที่มีลักษณะที่ดีขึ้น ในส่วนของสิ่งมีชีวิตที่มีลักษณะที่ไม่ดีนั้น จะไม่ได้รับการสนับสนุนหรือนำมาพิจารณา

Genetic Algorithm นั้นเป็นวิธีการหาคำตอบของปัญหาที่มีขนาดใหญ่และซับซ้อน เนื่องจากคุณสมบัติการเลียนแบบการถ่ายทอดคุณลักษณะทางพันธุกรรมตามธรรมชาติ และนำค่าที่เหมาะสมที่สุดจากประชากรรุ่นก่อนมาใช้พิจารณาสำหรับการสร้างประชากรรุ่นถัดมา และมีการใช้ตัวดำเนินการที่สำคัญคือ Select, Crossover, Mutation ซึ่งทำให้สามารถที่จะสร้างคำตอบได้อย่างหลากหลาย ดังนั้นการค้นหาโดยใช้ Genetic Algorithm จึงเป็นการค้นหาแบบกระจายในวงกว้าง ทำให้การค้นหาเป็นไปได้อย่างทั่วถึงภายใน Search space ที่มีอยู่

องค์ประกอบที่สำคัญของ Genetic Algorithm

มีองค์ประกอบที่สำคัญทั้งหมด 5 ส่วนด้วยกันดังนี้

1. Chromosome encoding คือ ขั้นตอนสำหรับแปลงทางเลือกสำหรับการแก้ปัญหาที่เป็นไปได้ให้อยู่ในรูปแบบของ Chromosome

2. Initial Population คือการสุ่มเลือกเพื่อสร้างประชากรต้นแบบขึ้นมาเพื่อใช้เป็นจุดเริ่มต้นของขั้นตอนการวิวัฒนาการ

3. Fitness Function คือ ฟังก์ชันสำหรับประเมินค่าความเหมาะสม เพื่อให้คะแนนสำหรับคำตอบต่างๆ ที่เป็นไปได้ของปัญหา

4. Genetic Operator คือ การดำเนินการต่างๆ ตามขั้นตอนของ Genetic Algorithm เพื่อให้การเกิดวิวัฒนาการไปสู่คำตอบที่ดีขึ้น ซึ่งได้แก่ Selection, Crossover และ Mutation

5. Parameter คือ ปัจจัยที่ส่งผลต่อการทำงานของ Genetic Algorithm เช่น ขนาดของประชากร, ความน่าจะเป็นของการ Crossover หรือ ความน่าจะเป็นของการ Mutation

1. Chromosome Encoding

ในการแปลงวิธีการสำหรับแก้ปัญหาที่เป็นไปได้ ให้อยู่ในรูปแบบของ Chromosome นั้นสามารถที่จะทำได้ในหลายรูปแบบซึ่งแล้วแต่ความเหมาะสมของปัญหา

1.1 Binary Encoding

เป็นรูปแบบการเข้ารหัสโครโมโซม แบบปรกติทั่วไปสำหรับ Genetic Algorithm โดยในแต่ลำตำแหน่งยีนบนโครโมโซม นั้นจะแทนด้วย 0 และ 1 เท่านั้น เช่น

Chromosome 1. 1 0 1 1 1 1 1 0 1 0 1 0 0

Chromosome 2. 0 0 1 1 0 1 0 0 1 0 1 1 1

1.2 Value Encoding

ทุกตำแหน่งของยีนส์จะมีค่าบางค่าที่สามารถเชื่อมโดยไปยังบางปัญหาได้ เช่น

Chromosome 1. a e i y k l m n o e i k y

Chromosome 2. bac right left bac bac bac left right

1.3 Permutation Encoding

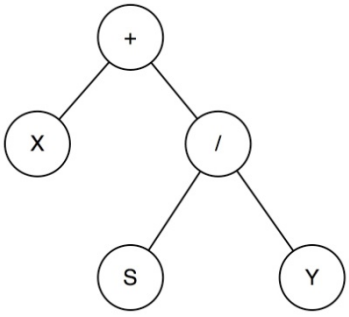
ทุกตำแหน่งของยีนส์บนโครโมโซม จะเป็นจำนวนนับที่เป็นลำดับต่างๆ การดำเนินการเพื่อแก้ปัญหา ปัญหาที่ใช้การเข้ารหัสโครโมโซมแบบนี้ เช่น Traveling Salesman Problem เป็นต้น

Chromosome 1. 1 2 4 3 5 8 7 6 10 9

Chromosome 2. 10 9 8 7 6 5 4 3 2 1

1.4 Tree Encoding

รูปแบบของยีนส์ทุกตำแหน่งบน Chromosome จะเป็น Node ของต้นไม้ ดังภาพที่ 12



ภาพที่ 12 การเข้ารหัส Chromosome แบบต้นไม้

2. Initial Population

ขั้นตอนนี้จะเป็นขั้นตอนแรกที่เกิดขึ้นก่อนที่จะเริ่มเข้ากระบวนการของ Genetic Algorithm โดยประชากรกลุ่มแรก หรือประชาการต้นกำเนิด จะเกิดจากการสุ่มเลือกขึ้นมาจาก กลุ่มของประชากรทั้งหมดที่มีอยู่ โดยในการสุ่มเลือกจะทำการสุ่มตามจำนวนของประชากรที่ได้กำหนดไว้เป็น Parameter ของ Algorithm

3. Fitness Function

โครโมโซมทุกตัวจะมีค่าความเหมาะสมของตัวเองเพื่อใช้สำหรับพิจารณาว่าโครโมโซมตัวนั้น เหมาะหรือไม่ที่จะนำมาใช้สืบทอดพันธุกรรมสำหรับสร้างโครโมโซมรุ่มใหม่ โดยวิธีการสำหรับคิดค่าความเหมาะสมนั้นจะใช้สมการที่สอดคล้องกับแต่ละปัญหา

4. Genetic Operator

Genetic Operator นั้นถือได้ว่าเป็นหัวใจหลักที่สำคัญของ Genetic Algorithm โดยการกระบวนการพื้นฐานที่สำคัญ 3 ขั้นตอนดังนี้

4.1 Selection

ในการคัดเลือกโครโมโซมขึ้นมาจากกลุ่มประชากรเพื่อใช้สำหรับสืบทอดพันธุกรรมให้กับลูกหลานนั้นมีความสำคัญมากเนื่องจากในการคัดเลือกควรที่จะเลือกโครโมโซมที่มีความเหมาะสมที่จะสืบทอดพันธุกรรมเท่านั้น จึงมีการพัฒนารูปแบบต่างๆ ที่ใช้สำหรับคัดเลือกโครโมโซม เช่น การคัดเลือกแบบ Roulette Wheel การคัดเลือกแบบ ranking และการคัดเลือกแบบ Elitist

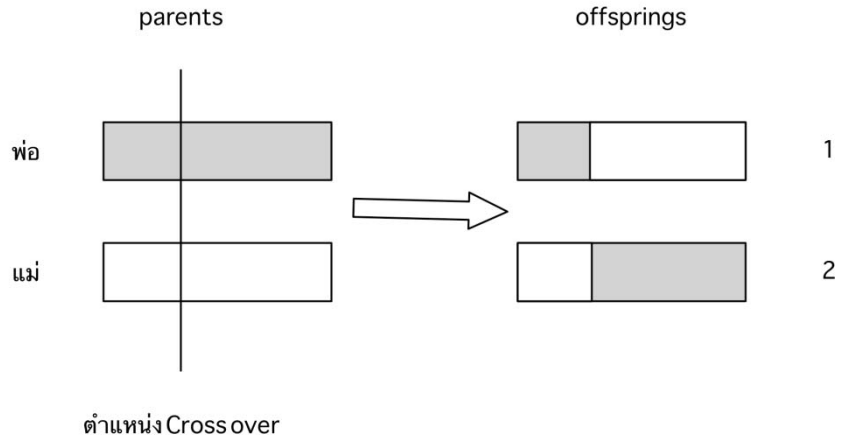
การคัดเลือกแบบ Roulette Wheel: เป็นการคัดเลือกแบบการให้น้ำหนักค่าความเหมาะสมกับแต่ละโครโมโซม โดยโครโมโซมที่มีค่าความเหมาะสมมากย่อมมีโอกาสถูกเลือกได้มาก และโครโมโซมที่มีค่าความเหมาะสมน้อยๆ ก็จะมีโอกาสถูกเลือกได้น้อย หรืออาจไม่ได้รับการคัดเลือกเลย

การคัดเลือกแบบ Ranking: คือจะทำการเลือกประชากรที่มีค่าความเหมาะสมที่ดีที่สุด โดยไม่สนใจประชากรตัวอื่นเลย

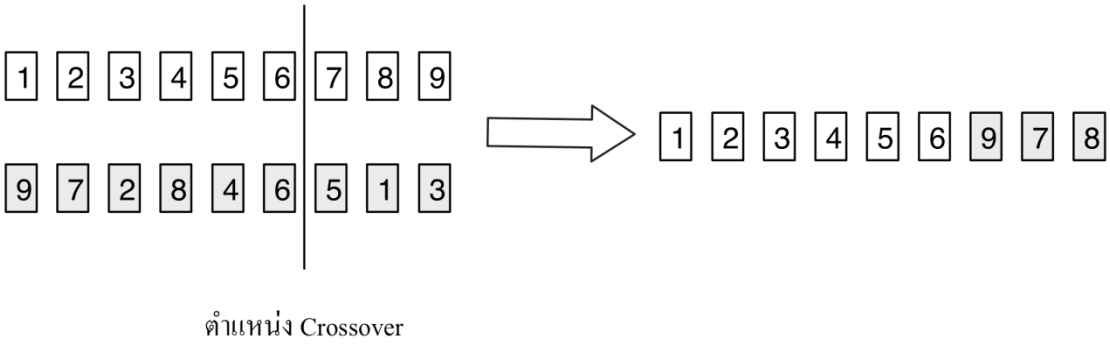
การคัดเลือกแบบ Elitist: โดยวิธีนี้จะทำการคัดลอกประชากรที่มีค่าความเหมาะสมที่ดีที่สุดเข้าไปในประชากรรุ่นต่อไปก่อน จากนั้นค่อยใช้วิธีการเลือกในแบบอื่นๆ เพื่อทำการคัดเลือกโครโมโซมตัวต่อๆ ไป

4.2 Crossover

เป็นกระบวนการที่สำคัญของ Genetic Algorithm ซึ่งเมื่อเกิดการ Crossover เกิดขึ้นในทางพันธุศาสตร์แล้ว จะทำให้เกิดการเปลี่ยนแปลงที่หลากหลายมากยิ่งขึ้น และสำหรับในการแก้ปัญหาก็เช่นกัน ก็จะทำให้เกิดความหลากหลายของคำตอบมากยิ่งขึ้นทำให้เรามีโอกาสมากยิ่งขึ้นในการเลือกคำตอบที่มีความเหมาะสมต่อสิ่งที่เราต้องการได้มากที่สุด ตัวอย่างสำหรับการ Crossover สามารถแสดงได้จากภาพที่ 13 และ 14 ดังต่อไปนี้



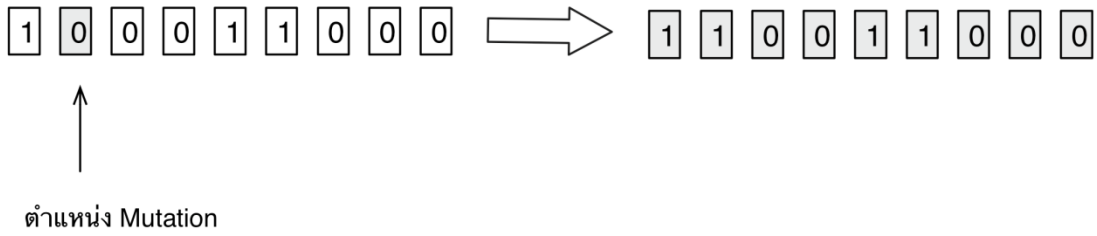
ภาพที่ 13 วิธีสำหรับการทำ Crossover



ภาพที่ 14 วิธีสำหรับการทำ Crossover กับข้อมูลในรูปแบบ Permutation Encoding

4.3 Mutation

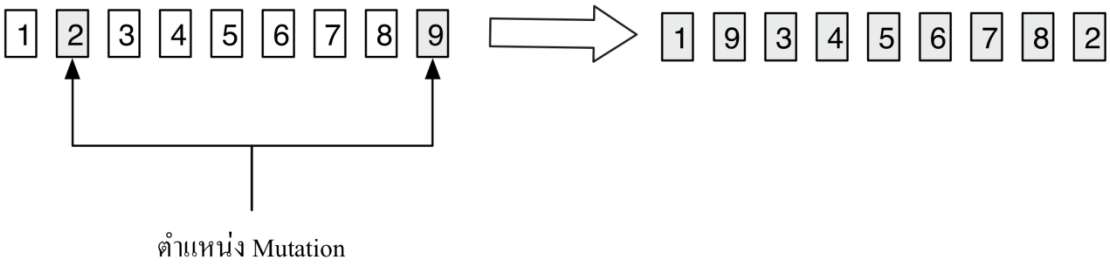
กระบวนการ Mutation นั้นเป็นจะเกิดขึ้นหลังจากกระบวนการ Crossover โดยเป็นกระบวนการที่ดำเนินการกับผลลัพธ์ที่ได้จากการ Crossover นั่นเอง โดยทั่วไปวิธีในการ Mutation นั้นจะเป็นการเปลี่ยนแปลงค่า ณ ตำแหน่งของยีนส์ที่ถูกกำหนดขึ้นโดยการสุ่ม ซึ่งสามารถแสดงให้เห็นถึงวิธีการ Mutation ในรูปแบบธรรมดาได้จากภาพที่ 15 ดังต่อไปนี้



ภาพที่ 15 วิธีสำหรับการทำ Mutation

สำหรับการทำ Mutation อาจเกิดได้มากว่าหนึ่งตำแหน่งของยีนส์บนโครโมโซม ขึ้นอยู่กับการสุ่มโดยมีความน่าจะเป็นในการเกิด Mutation เป็น Parameter ที่คอยกำหนดว่าจะเกิดการ Mutation ณ จุดนั้นหรือไม่ และเทคนิคของการทำ Mutation นั้นก็จะขึ้นอยู่กับรูปแบบของการ

ทำ Chromosome Encoding ยกตัวตัวอย่างเช่น การเข้ารหัสโครโมโซมแบบ Permutation Encoding เพื่อใช้สำหรับแก้ปัญหาเรื่อง Traveling Salesman Problem ดังแสดงได้ดังภาพที่ 16



ภาพที่ 16 การทำ Mutation กับ Chromosome ที่เข้ารหัสแบบ Permutation Encoding

4.4 Parameter

Parameter พื้นฐานที่มีความสำคัญต่อการทำงานของ Genetic Algorithm นั้นมี 3 ตัวคือ

4.4.1 Crossover Probability คือ ความน่าจะเป็นของการเกิด Crossover ซึ่งมีค่าอยู่ในช่วง 0 - 100 โดยทั่วไปค่าความเหมาะสมของความน่าจะเป็นในการเกิด Crossover จะอยู่ที่ 60% - 95% และในกรณีที่ไม่เกิดการ Crossover เกิดขึ้นจะเป็นการทำสำเนา (Copy) รูปแบบของพันธุกรรมจาก Parent ไปสู่ Offspring เลย ยกตัวอย่าง การทำ Crossover เช่น เรากำหนดให้ Crossover Probability มีค่าเป็น 85% ถ้าเราทำการสุ่มเลือกตัวเลขขึ้นมาเพื่อเปรียบเทียบกับค่า Crossover Probability ได้เท่ากับ 20 นั่นคืออยู่ในช่วงที่ <= 85 ในกรณีนี้จะยอมให้เกิดการ Crossover เกิดขึ้น

4.4.2 Mutation Probability คือ ความน่าจะเป็นของการเกิด Mutation จะมีค่าอยู่ในช่วง 0 - 100 ส่วนใหญ่ค่าความน่าจะเป็นของการเกิด Mutation จะถูกกำหนดไว้ให้อยู่ในช่วง 0% - 1% ต่อตำแหน่งของ Chromosome ในกรณีที่ไม่มีการ Mutation นั่นหมายความว่ามีเพียงการ Crossover เกิดขึ้นเพียงอย่างเดียว แต่ถ้าหากว่า เกิดการ Mutation 100% จะทำให้ทุกตำแหน่งใน Chromosome มีการเปลี่ยนแปลงทั้งหมด ซึ่งสำหรับใน Genetic Algorithm นั้นอาจเกิดกรณีนี้ขึ้นได้แต่ไม่บ่อยนัก ไม่เช่นนั้นการค้นหาจะเปลี่ยนจาก Genetic Algorithm การเป็น Random Search

4.4.3 Population Size หรือ จำนวนของประชากรในแต่ละรุ่น ถ้ามีจำนวนมากเกินไปจะทำให้ต้องเสียเวลาในการประมวลผลมากและทำงานได้ช้าลง

4.5 Stop Condition

เงื่อนไขในการหยุดการค้นหา สามารถกำหนดได้หลายวิธีดังนี้เช่น

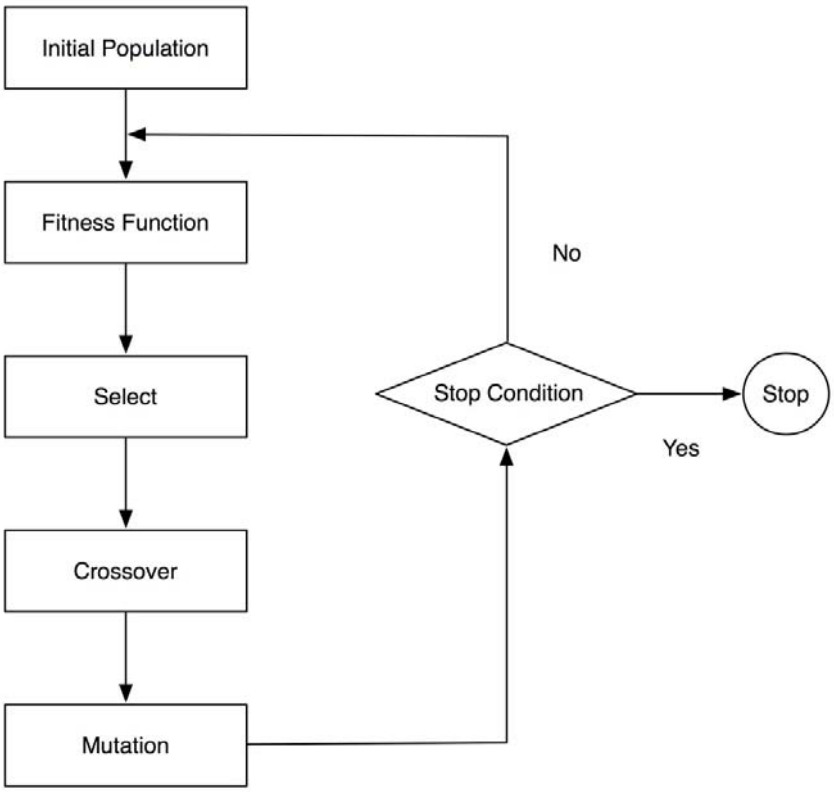
ครบรอบการทำงานที่ได้กำหนดไว้

พบเป้าหมายหรือคำตอบที่ต้องการ

พบคำตอบที่ได้เริ่มลู่เข้าสู่คำตอบ เช่น คำตอบที่ได้จากประชากรแต่ละรุ่น ไม่มีการเปลี่ยนแปลงหรือคงที่เป็นจำนวนที่ติดต่อกัน

4.6 ขั้นตอนการทำงานของ Genetic Algorithm

จากรายละเอียดของ Genetic Algorithm ที่ได้กล่าวมานั้นสามารถสรุปเป็นลำดับและขั้นตอนในการทำงานได้ดังภาพที่ 17



ภาพที่ 17 ขั้นตอนและลำดับการทำงานของ Genetic Algorithm

Island-Based Parallel Genetic Algorithm

เป็นอีกหนึ่งเทคนิคสำหรับ Parallel Genetic Algorithm ซึ่งได้รับแรงบันดาลใจจากความเป็นจริงในธรรมชาติ โดยพิจารณาว่าประชากรกลุ่มใหญ่ ประกอบไปด้วยกลุ่มประชากรย่อยๆ ชนิดเดียวกันหลายกลุ่ม แต่ถูกแบ่งแยกจากกันด้วยข้อจำกัดของขนาดพื้นที่หรือสภาพแวดล้อม และประชากรแต่ละกลุ่มนั้นมีทิศทางในการวิวัฒนาการเป็นของตัวเอง ไม่ได้ขึ้นอยู่กับประชากรกลุ่มอื่นๆ ซึ่งจุดนี้เองที่ทำให้เกิดความหลากหลายมากยิ่งขึ้นในการวิวัฒนาการ และในบางกรณีประชากรกลุ่มย่อยๆ อาจมีการแลกเปลี่ยนประชากรที่มีคุณภาพดีระหว่างกลุ่มกันได้ ทำให้แต่ละกลุ่มสามารถนำส่วนที่ดีไปใช้ในการวิวัฒนาการร่วมกัน

การทำงานของ Island-based PGA สำหรับงานวิจัยนี้ เริ่มต้นโดยแต่ละ Processor บนระบบ Computer Clusters สุ่มเลือกประชากรกลุ่มย่อยของตนเองขึ้นมาตามขนาดของประชากรที่กำหนด และดำเนินกระบวนการ Sequential GA กับกลุ่มประชากรของตนเอง เมื่อแต่ละกลุ่มได้ Generation ครบรอบจำนวนที่ได้กำหนดไว้ แต่ละ Processor จะทำการแลกเปลี่ยน (Migration) กลุ่มสมาชิกที่ดีสุด (Migrants) จำนวนหนึ่งกับ Processor ที่อยู่ข้างเคียง และดำเนินกระบวนการในรูปแบบนี้ต่อไปเรื่อยๆ เมื่อกระบวนการทั้งหมดสิ้นสุดลง จะทำการคัดเลือกผลลัพธ์ที่ดีที่สุดจากประชากรกลุ่มย่อยทั้งหมดเพื่อใช้เป็น Output ในกระบวนการ Query optimization

ขั้นตอนสำหรับ Island-based Parallel Genetic Algorithm

สามารถอธิบายถึงรายละเอียดของ Island-based PGA ที่ประยุกต์ใช้ในงานวิจัยนี้ได้ดังต่อไปนี้

Initialize sub-population: แต่ละ Processor บนระบบ Computer Cluster จะทำการสร้างกลุ่มประชากรของตัวเองขึ้นมาโดยใช้วิธีการสุ่มเลือก จนได้กลุ่มประชากรที่มีขนาด (Population Size) ตามที่กำหนดไว้

Fitness Function: แต่ละ Processor จะทำการหาค่าความเหมาะสมของสมาชิกแต่ละตัวภายในกลุ่มประชากรย่อยของตนเอง ซึ่งวิธีสำหรับการคิดค่าความเหมาะสมสำหรับประชากรนั้นจะอยู่บนพื้นฐานการคำนวณโดยใช้ Distributed Cost Model ที่กำหนดไว้

Selection: ใช้วิธี Selection แบบ Tournament ซึ่งจะดำเนินการโดยสุ่มเลือกสมาชิกในกลุ่มประชากรออกมาครั้งละ 10 QEPs แล้วเปรียบเทียบเอา QEPs ที่มีค่า Cost ที่น้อยที่สุดซึ่งได้จากการคำนวณใน Fitness Function เพื่อนำไปใช้สำหรับดำเนินการในขั้นตอน Genetic Operators ต่อไป

Genetic Operators: การเกิด Crossover และ Mutation เพื่อสร้างประชากรรุ่นถัดไปนั้นจะถูกควบคุมโดยค่าความน่าจะเป็นที่ใช้นั้นสามารถแสดงได้จากตารางที่ 1 และวิธีการสำหรับ Crossover และ Mutation เป็นแบบ One-Point

Migration: เป็นการแลกเปลี่ยนกลุ่มสมาชิกของประชากรในแต่ละ Processor กับ Processor ข้างเคียง โดยการแลกเปลี่ยนแต่ละครั้งได้กำหนดให้กลุ่มสมาชิกที่ถูกแลกเปลี่ยน (Migrants) จะถูกคัดเลือกจาก QEPs ที่มีค่าน้อยที่สุด ลดหลั่นลงมาจนครบจำนวนที่ได้กำหนดไว้ (Number of Migrants) เมื่อมีการรับส่งข้อมูลกันแล้ว แต่ละ Processor จะนำกลุ่มสมาชิกใหม่ที่ได้รับไปแทนที่กลุ่มสมาชิกเดิมที่อยู่ในเกณฑ์ที่อ่อนแอ หรือกลุ่มของ QEPs ที่มีค่า Cost มากที่สุดตามลำดับ และรูปแบบการรับส่งข้อมูลที่เกิดขึ้นในระบบ Computer Clusters เป็นแบบ Ring Topology

Neighbor State Functions

Randomize Algorithms นั้นจะทำการค้นหา Global Optimum State ที่อยู่ใน Search Space สำหรับวิธีการที่ใช้ในการค้นหาจะเป็นไปโดยการสุ่มเลือก State ขึ้นมาแล้วทำการเปรียบเทียบ และถ้าหาก State ที่สุ่มเลือกขึ้นมานั้นเป็น State ที่ดีกว่าเดิมจะทำการย้ายจาก State เดิมไปสู่ State ใหม่โดยการเคลื่อนย้ายจาก State เดิมไปสู่ State ใหม่นั้นเรียกว่าการ Move และสำหรับการค้นหาจะดำเนินไปโดยการ Move จาก State เดิมไปสู่ State ใหม่เรื่อยๆ ไปจนกว่าจะเจอ State ที่เป็น Global Optimum State (Pongpinigpinyo 1996)

สำหรับวิธีการเลือกสุ่ม State ขึ้นมาใหม่เพื่อทำการเปรียบเทียบกับ State ปัจจุบันนั้นจะทำการเลือกสุ่มจาก State ที่อยู่ข้างเคียงกับ State ปัจจุบัน โดยเรียก State ที่อยู่ข้างเคียงนี้ว่า Neighbor State สำหรับงานวิจัยชิ้นนี้การสุ่มเลือก Neighbor State ขึ้นมานั้นจะเป็นไปโดยการพิจารณาจากกฎที่เรียกว่า Transformation Rules

Transformation Rules

การสุ่มเลือก State ขึ้นมาเพื่อทำการเปรียบเทียบกับ State ปัจจุบัน หรือการเคลื่อนจาก State ปัจจุบันไปยัง State ใหม่จะเป็นไปอย่างมีกฎเกณฑ์ เพื่อให้การเปลี่ยนแปลงรูปร่างของ Cost Functions เป็นไปอย่างราบเรียบ ไม่มีการเปลี่ยนแปลงแบบก้าวกระโดดมากนัก เพื่อให้การค้นหาของ Randomize Algorithm อย่างเช่น Simulated Annealing Algorithm สามารถดำเนินไปอย่างมีประสิทธิภาพ

สำหรับ Transformations Rules มีดังต่อไปนี้ (Ozsu & Valduriez, 1991)

(1) Join Commutatively : $A \bowtie B \longrightarrow B \bowtie A$

(2) Join Associatively : $(A \bowtie B) \bowtie C \leftrightarrow A \bowtie (B \bowtie C)$

(3) Left Join Exchange : $(A \bowtie B) \bowtie C \longrightarrow (A \bowtie C) \bowtie B$

(4) Right Join Exchange : $A \bowtie (B \bowtie C) \longrightarrow B \bowtie (A \bowtie C)$

พิจารณาจากกฎข้อที่ (3) และ (4) ของ Transformations Rules จะก่อให้เกิดผลลัพธ์ที่เป็น Cartesian product สำหรับการ Query ที่เป็น Linear Query ดังนั้นสำหรับงานวิจัยชิ้นนี้จึงเลือกใช้เพียงแค่กฎข้อที่ (1) และ (2) เท่านั้น

Computer Clusters

Computer Cluster คือ กลุ่มของคอมพิวเตอร์ที่เชื่อมต่อกันและสามารถทำงานร่วมกัน ซึ่งเราสามารถมองได้ว่าเครื่องคอมพิวเตอร์ทั้งหมด ทำงานเสมือนเป็นเครื่องคอมพิวเตอร์เครื่องเดียวกัน ซึ่งเครื่องคอมพิวเตอร์ต่างๆ จะถูกเชื่อมต่อกันโดยระบบ Local Area Network (LAN) Computer Cluster ถูกพัฒนาขึ้นเพื่อใช้ในการเพิ่มประสิทธิภาพของการทำงานให้สูงขึ้นมากกว่าการทำงานในระดับ Single Computer

สามารถแยกระบบ Computer Cluster ตามลักษณะของการทำงานได้ดังนี้

High-availability (HA) clusters ถูกพัฒนาขึ้นมาเพื่อความคงทนของระบบ ซึ่งทำให้ระบบสามารถทำงานต่างๆ ได้ตลอดเวลาอย่างต่อเนื่อง โดยระบบจะประกอบไปด้วย Node ต่างๆ ที่มีลักษณะเหมือนกัน (redundancy nodes) ซึ่งสามารถที่จะทำงานแทนกันได้ หากมี node ใด node หนึ่งไม่สามารถทำงานได้ โดยระบบจะมีการตรวจสอบองค์ประกอบภายในระบบ หากพบว่ามี

องค์ประกอบใดไม่สามารถที่จะทำงานได้ ระบบจะนำงานที่ถูกกระทำค้างอยู่โดยองค์ประกอบที่ไม่สามารถทำงานได้นั้นไปให้องค์ประกอบอื่นทำต่อแทน

ตัวอย่างของงานที่จำเป็นจะต้องใช้ HA Cluster เช่น databases, file sharing บนระบบ network, business applications, customer services เช่น electronic commerce websites

Load-balancing clusters ถูกพัฒนาขึ้นมาเพื่อให้ระบบสามารถรองรับปริมาณงานที่มีจำนวนมากๆ ได้ ซึ่งทำงานโดย Front ends node เป็นตัวรับงานต่างๆ เข้ามาและมี commercial load balancers เป็นตัวกระจายงานออกไปให้ node ต่างๆ ที่อยู่ในระบบ Cluster ทำงานที่รับเข้ามา

High-performance computing (HPC) clusters

ทำงานโดยอาศัยการประมวลผลแบบขนาน (Parallel Computing) โดยทำการแบ่งงานที่รับเข้ามาออกเป็นส่วนย่อยๆ แล้วกระจายส่วนย่อยๆ เหล่านั้นออกไปประมวลผลตาม node ต่างๆ ซึ่งวิธีนี้จะทำให้สามารถทำงานที่ใช้เวลาในการประมวลผลนานๆ สามารถทำงานเสร็จได้อย่างรวดเร็วยิ่งขึ้น

Valid QEP

ภายใน Search space จะประกอบไปด้วย Query Execution Plan (QEP) ที่เป็นไปได้ทั้งหมดซึ่งเกิดจาการแปลง High-level Query ให้อยู่ในรูป Low-level Query ซึ่งในบาง Low-level Query นั้นอาจก่อให้เกิดผลลัพธ์ที่เป็นผลคูณคาร์ทีเชียน (Cartesian product) ซึ่งเป็นการทำให้เป็นการเสียเวลาในการสร้าง และค้นหาเพิ่มมากขึ้น ดังนั้นในการสร้าง QEP ขึ้นภายใน Search space จึงควรจะมีเฉพาะ Valid QEP เท่านั้น โดยสามารถให้คำนิยามคำว่า Valid QEP ที่ใช้สำหรับงานวิจัยนี้ได้ดังนี้ Stillger, Spiliopoulou และ Freytag (1996)

นิยาม : QEP จะ valid ต่อเมื่อ Join Tree ของ QEP นั้น valid นั่นคือ

1. ไม่มีผลคูณคาร์ทีเชียน (Cartesian product)

2. มีการ Join ด้วย Relation ที่ซ้ำกัน (Forced intersections) คือ แต่ละ Join Node ซึ่งแทนแต่ละ Relation จะปรากฏได้เพียงแต่ครั้งเดียวใน Join Tree เท่านั้น

บทที่ 4

วิธีดำเนินการวิจัย

งานวิจัยนี้มุ่งเน้นการพัฒนา และปรับปรุงการทำงานของกระบวนการ Distributed Query Optimization ในกรณีที่การสอบถามข้อมูลเกิดการ Join ของ Relations จำนวนมาก คือ ตั้งแต่ช่วงจำนวน 10 - 40 Relations ซึ่งปัญหาที่เกิดกับกระบวนการนี้ถือว่าเป็น Combinatorial Optimization Problem ในอีกรูปแบบหนึ่ง ผู้วิจัยได้เลือกที่จะทำการศึกษาเพื่อที่จะบ่งชี้ว่าวิธีสำหรับการค้นหา (Search Strategy) ภายในกระบวนการ Distributed Query Optimization โดยใช้ Island Based Parallel Genetic Algorithms เข้ามาช่วยให้กระบวนการ Distributed Query Optimization นั้นสามารถที่จะทำงานได้อย่างมีประสิทธิภาพและรวดเร็วมากยิ่งขึ้น โดยการนำเทคนิค Island Based Parallel Genetic เข้ามาทดลองและวัดประสิทธิภาพการทำงาน

ผู้วิจัยดำเนินการทดลองโดยการจำลองแบบ (Simulation) การทำงานของ Distributed Query Optimizer ซึ่งเป็นองค์ประกอบหนึ่งของระบบ DDBMS ซึ่งเป็นองค์ประกอบที่ใช้สำหรับดำเนินกระบวนการ Distributed Query Optimization และการจำลองแบบการทำงานของ Distributed Query Optimizer นั้นผู้วิจัยได้ทำการจำลองแบบและพัฒนาให้มีการรองรับการคำนวณแบบขนาน (Parallel Computing) ภายใต้มาตรฐาน Message Passing Interface 2 (MPI 2) และสำหรับการทดลองผู้วิจัยได้นำ เทคนิคสำหรับการค้นหาแบบต่างๆ มาทำการทดลองเชิงเปรียบเทียบเพื่อเป็นตัวชี้วัดประสิทธิภาพการทำงานของ Island Based Parallel Genetic Algorithm ที่มีต่อการแก้ปัญหาเรื่อง Distributed Query Optimization ภายใต้สภาวะแวดล้อมของการจำลองแบบที่ผู้วิจัยได้กำหนดขึ้น และการทดลองที่เกิดขึ้นได้ดำเนินการบนระบบ คอมพิวเตอร์คลัสเตอร์ ซึ่งประกอบไปด้วยเครื่อง Front-end 1 เครื่อง และเครื่อง Compute node 3 เครื่อง ทั้งหมดเป็นเครื่อง Intel Xeon 2.80GHz ขนาดของ Cache 2M และขนาด Memory 4Gb

ในการออกแบบการทดลองและการดำเนินงานภายในงานวิจัยชิ้นนี้ ผู้วิจัยสามารถแจกแจงรายละเอียดต่างๆ ในแต่ละขั้นตอนได้ดังต่อไปนี้

ศึกษาการคำนวณแบบขนาน

สำหรับการคำนวณแบบขนาน (Parallel Computing) ผู้วิจัยเลือกใช้มาตรฐาน Massage Passing Interface (MPI) เวอร์ชัน 2 โดยใช้โปรแกรม LAM/MPI version 7.1.3 (LAM/MPI Parallel Computing 2009) ในการจัดการการทำงานแบบ Parallel Computing บนระบบ Computer Clusters

ศึกษาและทำการจำลองแบบการทำงานของ Distributed Query Optimizer

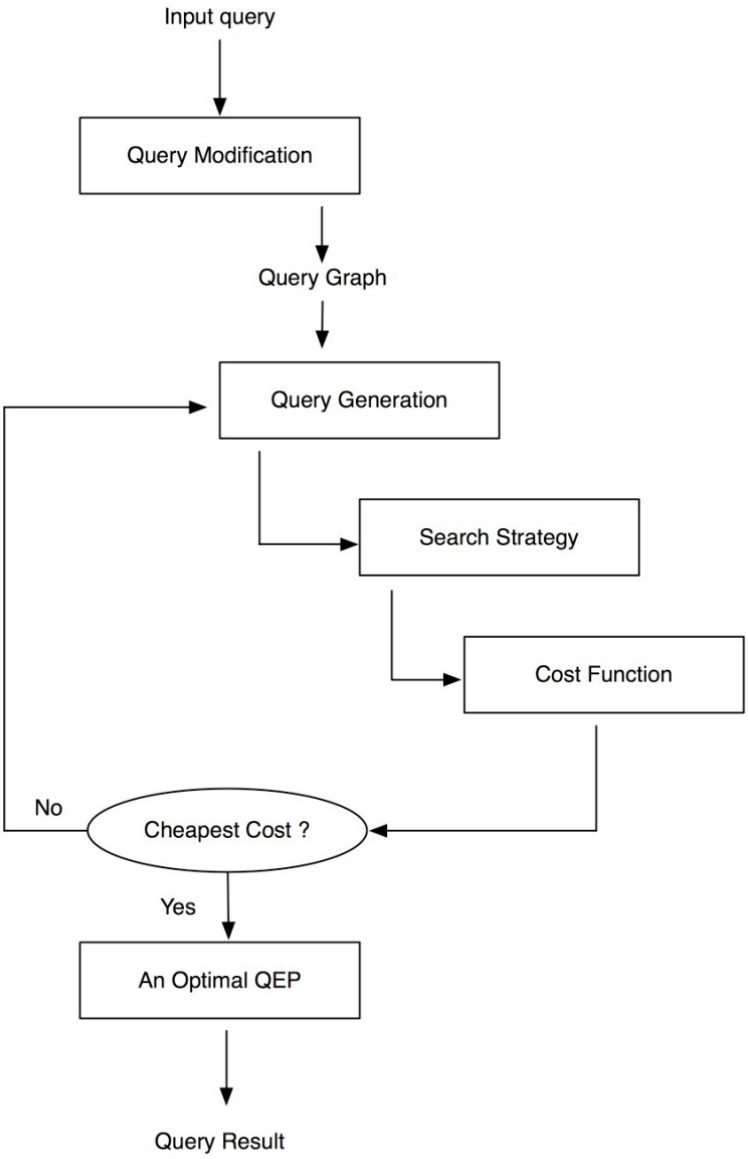
ผู้วิจัยได้ทำการเลือกที่จะจำลองแบบ (Simulated) การทำงานของ Distributed Query Optimizer เนื่องจากเพื่อความสะดวกในการควบคุมปัจจัยต่างๆ ที่มีผลต่อการทำงาน (Parameters) และสามารถที่จะทำการวัดประสิทธิภาพผลจากการทดลองได้อย่างถูกต้องและเที่ยงตรงโดยปราศจากปัจจัยรบกวนอื่นๆ

ในการจำลองแบบการทำงานของ Distributed Query Optimizer ผู้วิจัยได้ดำเนินการตามหลักการทำงานของกระบวนการ Distributed Query Optimization ซึ่งสามารถที่จะแสดงโครงสร้างการทำงานหลักๆ ได้ดังภาพที่ 18 (Pongpinigpinyo 1996)

จากภาพที่ 18 จะเห็นได้ว่าในส่วนการคำนวณนั้น ข้อมูลที่นำมาใช้สำหรับการคำนวณสามารถที่จะดึงมาจากส่วนประกอบที่เรียกว่า Data Dictionary ของระบบได้เลยโดยที่ไม่จำเป็นต้องเข้าไปยุ่งเกี่ยวกับข้อมูลที่อยู่ในแต่ละ Relations จริงๆ ซึ่งข้อมูลส่วนนี้เรียกว่า Database Statistics เป็นข้อมูลที่ระบบ DDBMS จะต้องทำการสร้างและนำมาจัดเก็บเอาไว้

ดังนั้นในการจำลองการแบบการทำงานของ Distributed Query Optimizer จึงมีความจำเป็นอย่างยิ่งที่ต้องทำการจำลององค์ประกอบที่เรียกว่า Data Dictionary โดยสำหรับงานวิจัยนี้องค์ประกอบส่วนนี้จะเป็นส่วนที่สร้างข้อมูลต่างๆ ที่ใช้สำหรับเป็นค่า Input ให้กับองค์ประกอบอื่นที่ใช้สำหรับการคำนวณต่อไป

รายละเอียดและองค์ประกอบหลักที่ใช้เพื่อทำการจำลองแบบ (Simulation) การทำงานของ Distributed Query Optimizer ให้สามารถทำงานได้สมบูรณ์ สามารถบรรยายได้ดังต่อไปนี้



ภาพที่ 18 โครงสร้างการทำงานโดยรวมของกระบวนการ Query Optimization

Query Generator

ในงานวิจัยชิ้นได้ทำการกำหนดขอบเขตและสมติฐานของงานวิจัยไว้คือ จะทำการพิจารณาเพียงเฉพาะการ Join ที่มีเงื่อนไขแบบเท่ากับ (Equality Join) โดยใช้ Model สำหรับการ Query แบบ Linear Query เท่านั้น เพื่อเป็นการกำหนดขนาดของ Search Space ที่เกิดขึ้นไม่ให้มีขนาดใหญ่และซับซ้อนเกินไป ในการสร้างค่าต่างๆ เพื่อให้เป็นค่าสำหรับการเริ่มต้นการทำงานทั้งหมดของกระบวนการ Distributed Query Optimization จะเริ่มต้นจาก Linear Query Join Graph

ดังนั้นการทำงานของระบบที่จำลองขึ้นนั้นจะทำการรับ Input Query Command แล้วทำการแปลงคำสั่งที่รับเข้ามาให้อยู่ในรูปของ Linear Join Graph แล้วส่งให้เป็นค่าเริ่มต้นสำหรับการทำงานของกระบวนการ Distributed Query Optimization ที่ได้จำลองขึ้นต่อไป

Database Configuration

การทำงานในส่วนนี้นั้นเป็นการจำลองแบบการทำงานในส่วนของ Data Dictionary ของระบบดังที่แสดงไว้ในภาพที่ 18 โดยมีความสำคัญคือการสร้างข้อมูลต่างๆ ที่มีความจำเป็นต่อการทำงาน เช่น Database Statistics โดยการสร้างค่าของข้อมูลในแต่ละครั้งจะเป็นการสุ่มค่าขึ้นมาตามขอบเขตที่ได้กำหนดให้ในข้อมูลแต่และตัว โดยข้อมูลที่สำคัญที่มีความจำเป็น มีดังต่อไปนี้

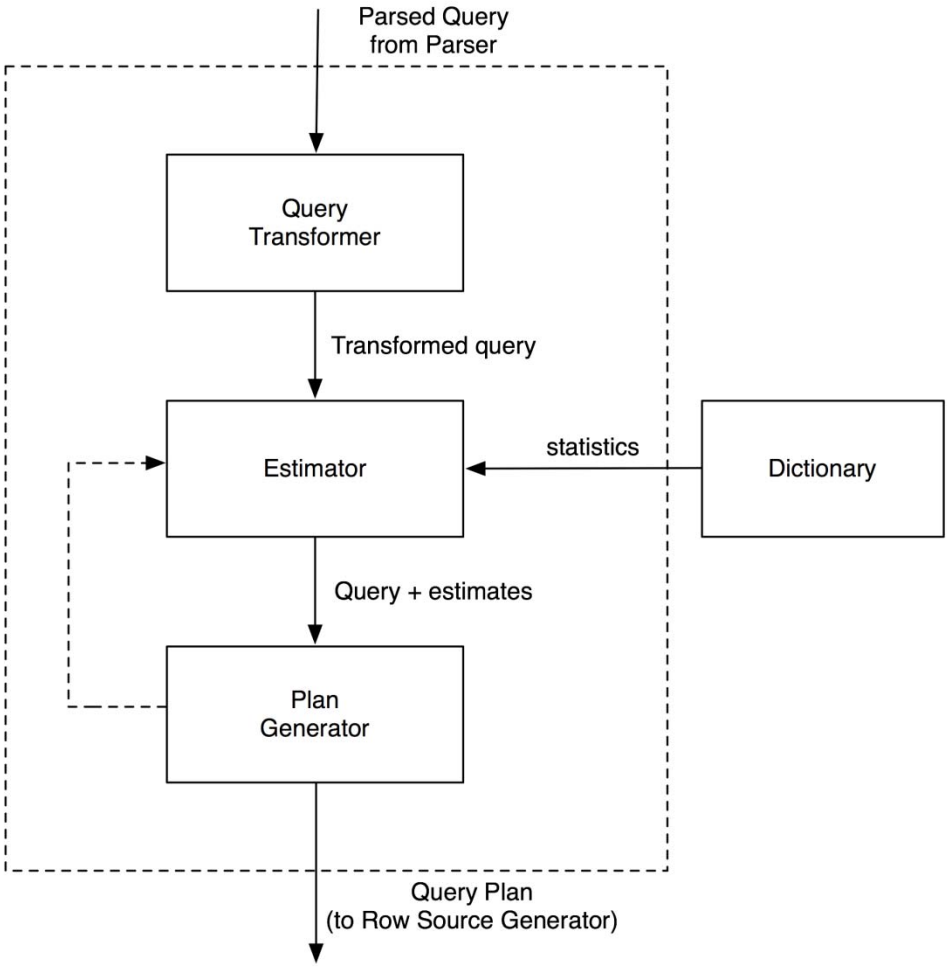
- **Relation Cardinalities** ค่าที่ทำการสุ่มขึ้นมาจะอยู่ในช่วง 1,000 - 100,000 ซึ่งจัดว่าเป็นจำนวน Cardinality ของ Relation ในฐานข้อมูลขนาดใหญ่ (Rho and March 1997 : 199-228) โดยขนาดของ Relation จะมีขนาดใหญ่หรือว่าเล็กนั้น จะดูได้จากจำนวน Cardinality ของแต่ละ Relation เป็นหลัก

- **Join Selectivity** ค่าที่ทำการสุ่มขึ้นมาจะอยู่ในช่วง 0.0 - 1.0 แต่ว่าไม่ได้รวมค่า 0.0 และ 1.0 ไปด้วยเพราะว่าถ้าค่าอยู่ที่ 0.0 นั่นหมายถึงไม่มีการ Join เกิดขึ้น และถ้าค่าเป็น 1.0 หมายความว่าในการ Join ผลลัพธ์ที่ได้จะเกิด Cartesian product (Pongpinigpinyo 1996)

- **Attribute Sizes** ขนาดของข้อมูลในแต่ละ Attribute จะมีขนาดอยู่ในช่วง 8 – 20 bytes (Rho and March 1997 : 199-228)

- **Data Transfer Rate** อยู่ในหน่วย Megabit per Second (Mbit/s) หรือ 10^6 โดยจะเป็นตัวเลขแบบต่อเนื่องในช่อง $1x10^6$ ถึง $4x10^6$

- ส่วนรายละเอียดการทำงานและการตัดสินใจต่างๆ ในกระบวนการ Distributed Query Optimization สามารถที่แสดงได้ดังภาพที่ 19 ต่อไปนี้ (Oracle 2009)



ภาพที่ 19 รายละเอียดการทำงานของกระบวนการ Query Optimizer

กำหนด Cost Model และวิธีการคิดค่า Cost

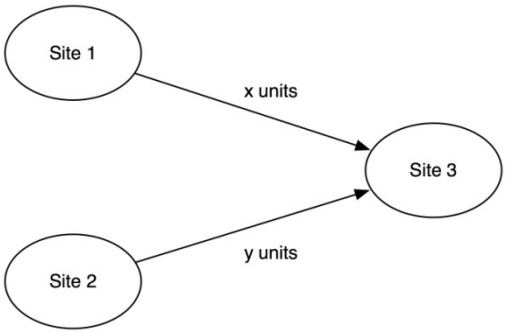
สำหรับแบบจำลองของ Distributed Query Optimizer ที่ผู้วิจัยได้สร้างขึ้นนั้น ผู้วิจัยได้ทำการเลือกใช้ Cost Model สำหรับ Distributed Database ที่ไม่ได้อ้างอิงอยู่กับวิธีการ Join ในรูปแบบใดรูปแบบหนึ่ง แต่จะเน้นในเรื่องของการคำนวณค่า Cost ที่เป็นเวลารวม (Total Time) ที่เกิดจากการรวมกันของเวลาที่จะต้องเสียไปในการประมวลในแต่ละ Site และเวลาที่จะต้องเสียไปจากการขนส่งข้อมูลระหว่าง Site ซึ่งสามารถแสดงสูตรสำหรับคิดคำนวณค่า Cost ได้ดังสมการที่ (9) (Ozsu and Valduriez 1991)

$$Total\_time = TCPU * \#insts + TI/O * \#I/Os + TMSG * \#msgs + TTR * \#bytes \quad (9)$$

โดยหากพิจารณาข้อมูลในปัจจุบัน ผู้วิจัยพบว่าประสิทธิภาพด้านที่เกี่ยวข้องกับการคำนวณระดับ Local Processing หรือความสามารถในด้านการประมวลผลของ CPU และความสามารถในการเข้าถึงข้อมูลผ่านระบบ I/O ของเครื่องคอมพิวเตอร์แต่ละเครื่องแทบจะไม่มีความแตกต่างกัน หรือมีความเหลื่อมล้ำใดๆ ดังนั้นสำหรับการคิดคำนวณค่าใช้จ่ายหรือเวลาที่ต้องใช้สำหรับการประมวลผลของแต่ละ QEP ภายใต้ Cost Function สำหรับงานวิจัยชิ้นนี้ จะกระทำอยู่ภายใต้ข้อสมมติในการทดลองว่า การประมวลผลระดับ Local Processing นั้นเท่ากันหมด จึงได้ทำการตัดการคำนวณในส่วนนี้ออกไป (Ozsu and Valduriez 1991) (Stillger, Spiliopoulou and Freytag 1996)

โดยสามารถที่จะทำการแสดงตัวอย่างของการคิดค่า Cost ที่เป็นเวลารวมทั้งหมดของการทำงานสำหรับแต่ละ QEPs ได้ดังต่อไปนี้ (Ozsu and Valduriez 1991)

กำหนดให้ A, B เป็น Relation ที่อยู่บน Site 1 และ Site 2 ตามลำดับ สำหรับในการดำเนินการ JOIN ที่เกิดขึ้นระหว่าง Relation A และ B จะกระทำโดยส่งข้อมูลของ Relation A จาก Site 1 และข้อมูลของ Relation B จาก Site 2 ออกไปเพื่อทำการประมวลผลและ JOIN ที่ Site 3 ดังภาพที่ 20



ภาพที่ 20 ตัวอย่างสำหรับการส่งข้อมูลที่เกิดจากการ Query

สมมุติให้ TMSG และ TTR เป็นตัวแปรที่สามารถแทนค่าได้ในหน่วยของเวลา เวลารวมที่เกิดขึ้นจากการขนส่งข้อมูลขนาด x หน่วยจาก Site 1 ไป Site 3 และ ข้อมูลขนาด y หน่วยจาก Site 2 ไป Site 3 สามารถคำนวณได้ดังสมาการที่ 10 Ozsu and Valduriez (1991)

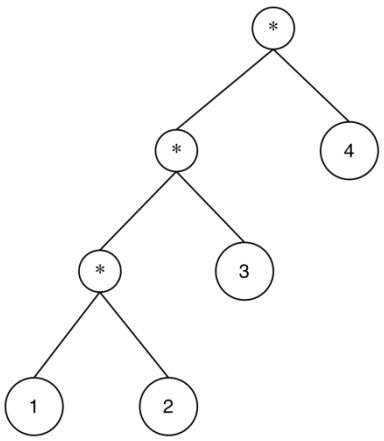
$$Total\_time = 2TMSG + TTR * (x + y) \quad (10)$$

โดยกำหนดให้ TMSG และ TTR คือค่าคงที่

กระบวนการทำงานของ Genetic Algorithm

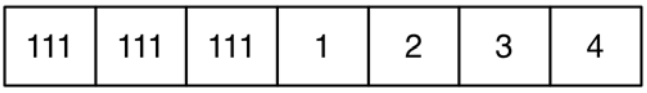
ผู้วิจัยสามารถอธิบายรายละเอียดสำหรับ Genetic Algorithm ขั้นตอนต่างๆ ที่ผู้วิจัยได้ใช้ในงานวิจัยนี้ได้ดังต่อไปนี้

Chromosome encoding เพื่อให้รูปแบบของการเข้ารหัส Chromosome มีความสอดคล้องกับตัวปัญหาให้มากที่สุด ผู้วิจัยจึงได้ทำการเลือกใช้การเข้ารหัส Chromosome แบบต้นไม้ (Tree encoding) ซึ่งทำให้สะดวกในการดำเนินการในขั้นตอนของการคิดค่า Cost ของแต่ละ Chromosome ในกระบวนการ Fitness Function ต่อไป โดยสามารถแสดงได้ดังภาพที่ 21



ภาพที่ 21 โครงสร้าง Chromosome แบบ Tree Encoding

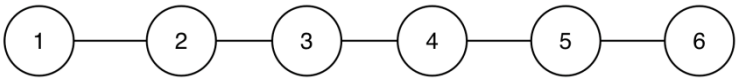
จากภาพที่ 21 ถึงแม้ว่าโครงสร้างที่แสดงจะแสดงในรูปของ Tree encoding แต่สำหรับการจัดเก็บในระดับโครงสร้างข้อมูล ผู้วิจัยได้เลือกที่จะจัดเก็บข้อมูลในรูปของ Array ซึ่งสามารถแสดงโครงสร้างการจัดเก็บข้อมูลในรูปแบบ Array ได้ดังภาพที่ 22 ซึ่งจะเป็นผลของการท่อง Tree แบบ Pre-order



ภาพที่ 22 โครงสร้างการจัดเก็บในรูปแบบของ Array

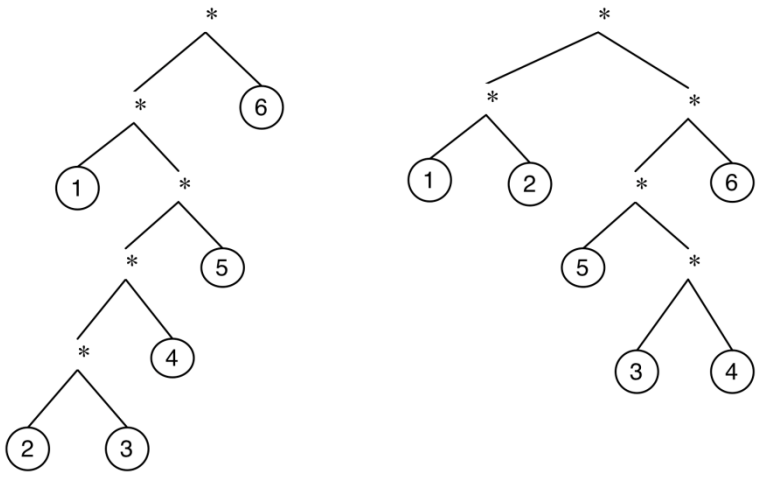
Crossover เพื่อให้สอดคล้องกับการเข้ารหัส Chromosome แบบต้นไม้ (Tree encoding) ดั้งนั้น เพื่อให้การ Crossover ซึ่งเป็นการดำเนินการระหว่าง Chromosome 2 ตัว เป็นไปอย่างสอดคล้องกับการเข้ารหัส Chromosome แบบต้นไม้ (Tree encoding) ผู้วิจัยจึงได้ทำการเลือกเทคนิคสำหรับการ Crossover แบบ Sub-tree crossover เข้ามาใช้สำหรับงานวิจัยชิ้นนี้ โดยสามารถแสดงขั้นตอนสำหรับการ Crossover ที่ผู้วิจัยได้ใช้ในงานวิจัยได้ดังนี้

1. เริ่มต้นการสร้าง Chromosome จาก linear join graph ดังภาพที่ 23 เพื่อเป็นส่วนสำหรับการกำกับลำดับการ join



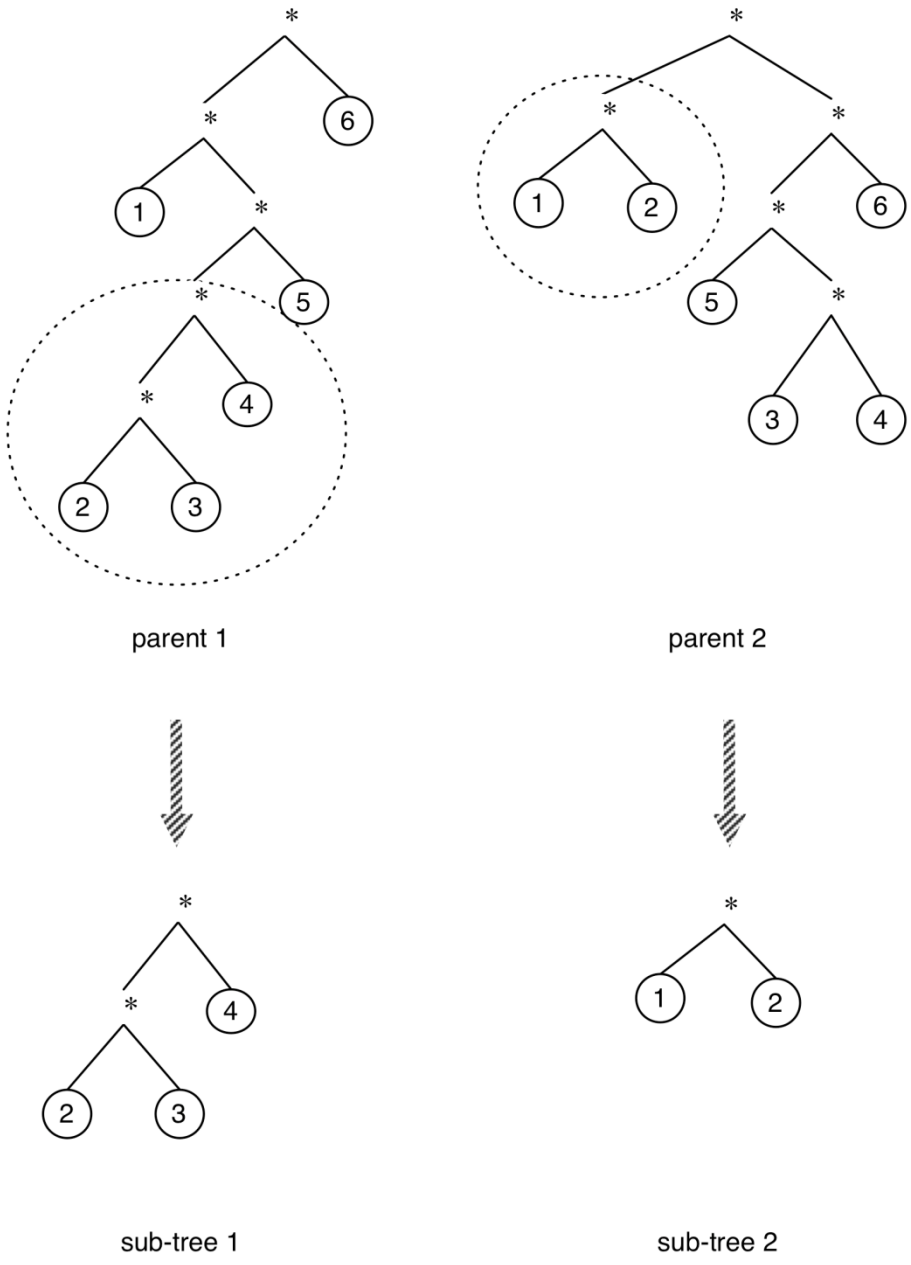
ภาพที่ 23 Linear join graph

2. เลือก Chromosome ต้นแบบขึ้นมา 2 ตัว ดังภาพที่ 24



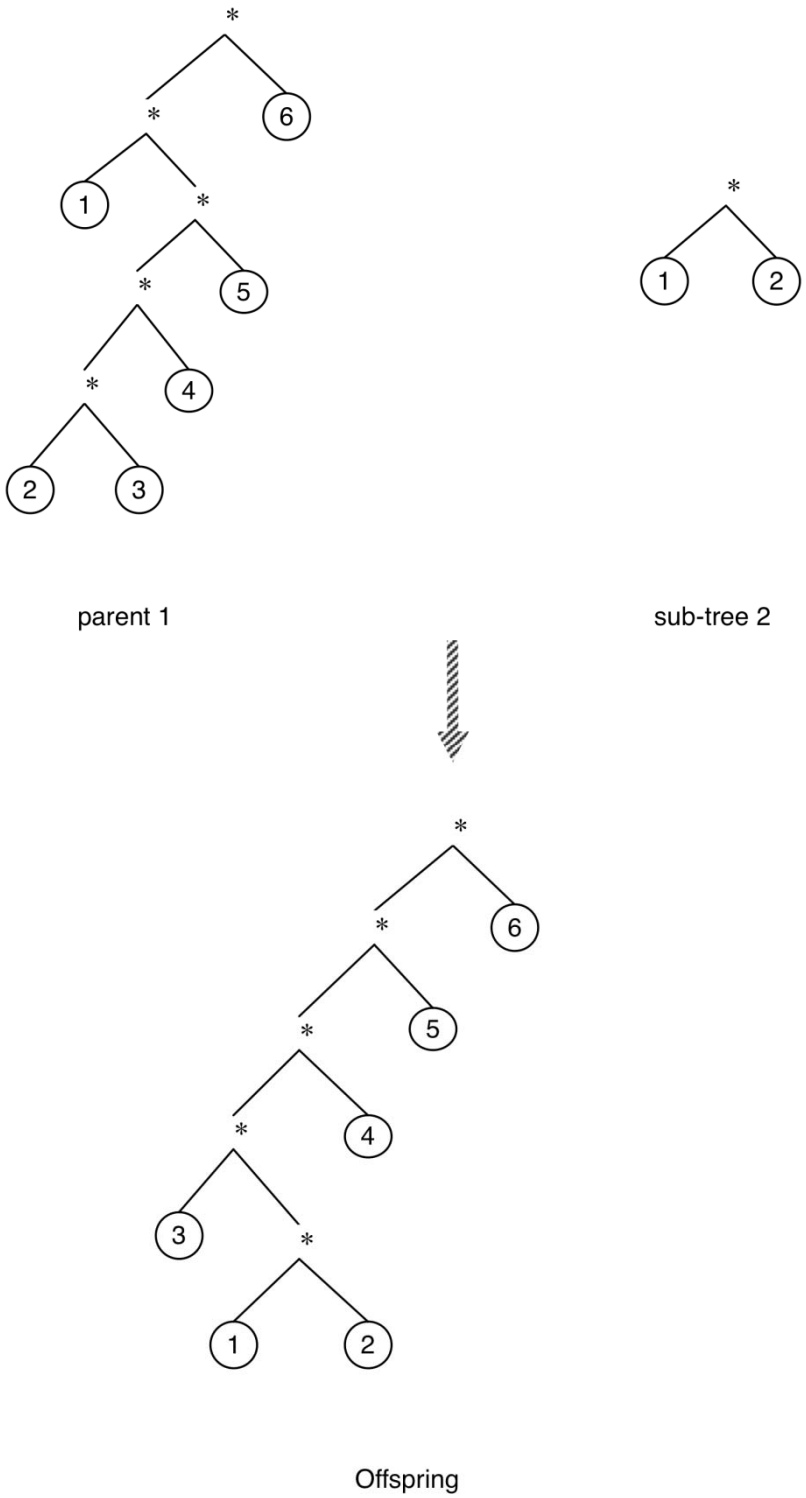
ภาพที่ 24 Parent 2 ตัว

3. สุ่มเลือก Sub-tree ภายใน Parent ที่เลือกมา ดังภาพที่ 25



ภาพที่ 25 เลือก Sub-tree จาก Parent ทั้งสอง

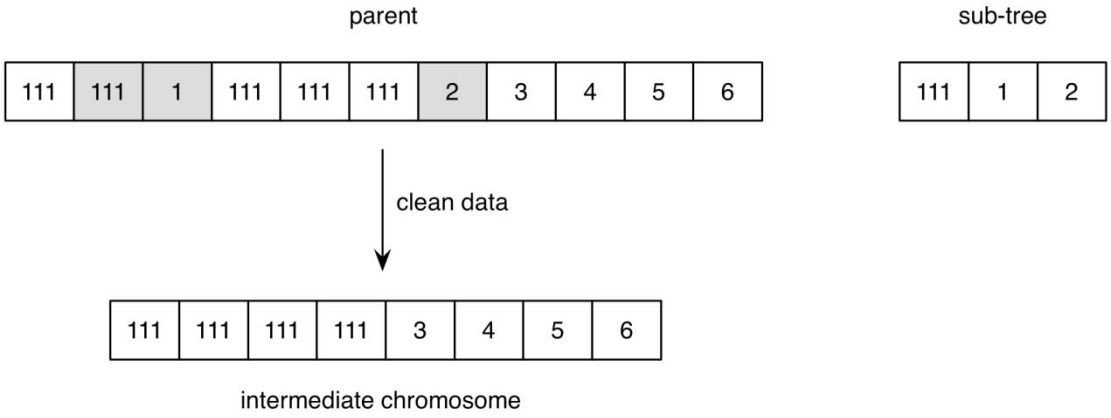
4. ทำการ Crossover โดยใช้ Sub-tree ดังภาพที่ 26



ภาพที่ 26 ผลลัพธ์จากการ Crossover

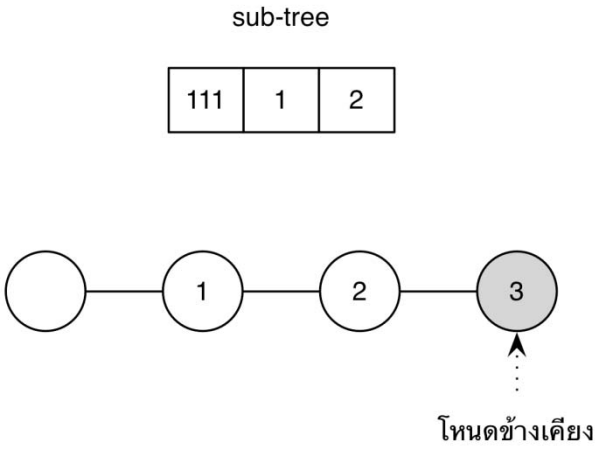
โดยในการดำเนินการในกระบวนการ Crossover จะดำเนินการในรูปแบบของ Array ซึ่งจะแสดงได้ดังขั้นตอนต่อไปนี้

ขั้นที่ 1 ทำการสร้าง Intermediate Chromosome ใหม่โดยการ Clean ข้อมูลภายใน Sub-tree ออกไปจาก Parent Chromosome ดังภาพที่ 27



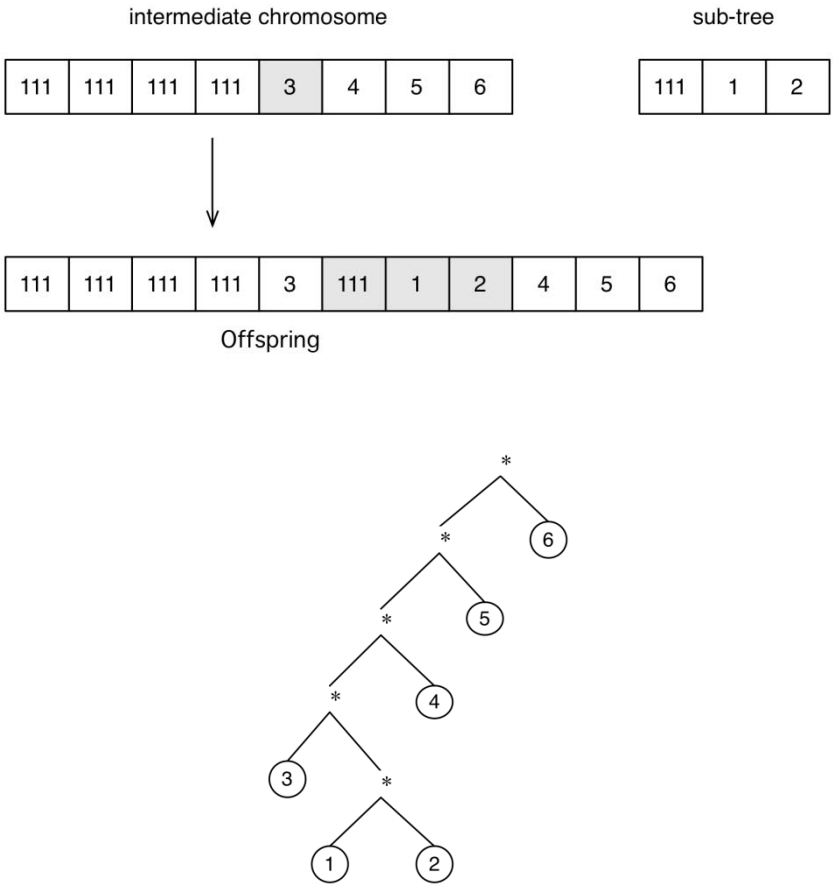
ภาพที่ 27 การสร้าง Intermediate Chromosome

ขั้นที่ 2 หาโหนดข้างเคียงของ Sub-tree ดังภาพที่ 28



ภาพที่ 28 เลือกโหนดข้างเคียงของ Sub-tree

ขั้นที่ 3 รวม Intermediate Chromosome และ Sub-tree เข้าด้วยกันโดยทำการแทรก Sub-tree เข้าไป ณ ตำแหน่งหลังของโหนดข้างเคียงที่ได้เลือกไว้ ดังภาพที่ 29



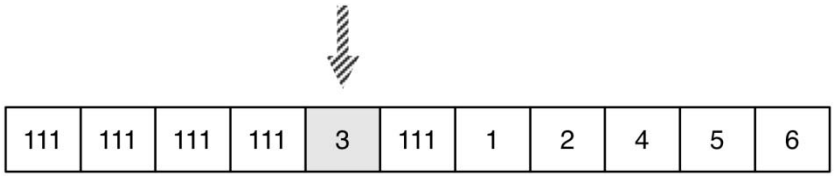
ภาพที่ 29 ผลลัพธ์ที่ได้จากการทำ Sub-tree Crossover

แต่สำหรับในบางกรณีที่เมื่อเสร็จสิ้นกระบวนการ Crossover แล้ว Offspring ที่ได้มีลักษณะเป็น Invalid Chromosome ดังนั้นวิธีการที่ผู้วิจัยใช้จึงไม่สามารถที่จะให้ Valid Chromosome ได้ในทุกกรณี

ดังนั้นจึงจำเป็นต้องมีการตรวจสอบความถูกต้องของ Chromosome หลังจากที่ทำการ Crossover แล้ว และจะมีการแก้ไขให้ถูกต้องตามกรณีที่เกิดขึ้นตามเงื่อนไขที่กำหนดไว้ในตัวโปรแกรมเพื่อให้ Chromosome นั้นแสดงถึง QEP ที่มีเป็น Valid QEP

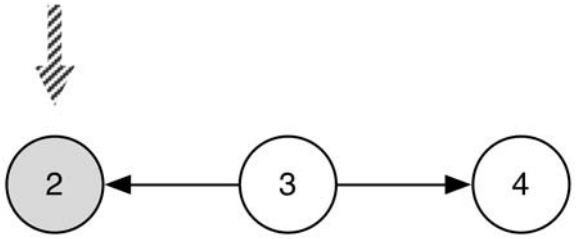
Mutation สำหรับในกระบวนการนี้ ผู้วิจัยได้ใช้วิธี Swap node เพื่อให้เกิดการ Mutation โดยในการเกิดการ Mutation แต่ละครั้งจะทำการ Swap node ภายใน Join Tree เพียง 1 ครั้ง โดยสามารถอธิบายขั้นตอนสำหรับการ Mutation ที่ผู้วิจัยได้ใช้ในงานวิจัยนี้ ได้ดังต่อไปนี้

ขั้นที่ 1 ทำการเลือก Terminal node ขึ้นมาหนึ่งโหนดด้วยการสุ่มเลือก ดังภาพที่ 30



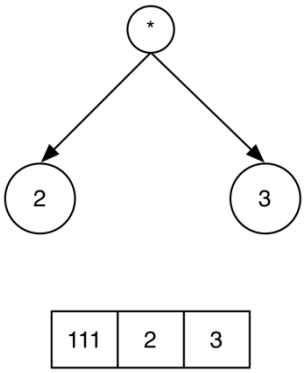
ภาพที่ 30 เลือกโหนดสำหรับการทำ Swap node mutation

ขั้นที่ 2 เลือกโหนดข้างเคียงของโหนดที่ถูกเลือกมา ดังภาพที่ 31

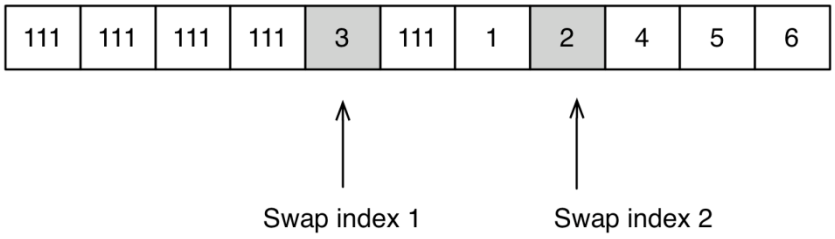


ภาพที่ 31 เลือกโหนดข้างเคียงโดยการสุ่ม

ขั้นที่ 3 ทำการสร้าง Sub-tree จาก โหนดที่เลือกมาและโหนดข้างเคียง ดังภาพที่ 32 และดังภาพที่ 33

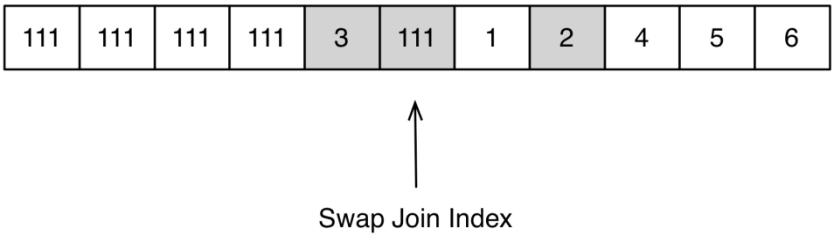


ภาพที่ 32 สร้าง Sub-tree จากโหนดที่เลือก



ภาพที่ 33 ทำการกำหนด Swap index

จากภาพที่ 31 ทำการกำหนด Swap index สำหรับโหนดที่เลือกมา โดยกำหนดให้โหนดที่เลือกครั้งแรกสุดเป็น Swap Index 1 และโหนดที่เลือกจากโหนดข้างเคียงของโหนดหลักให้เป็น Swap Index 2 และสำหรับขั้นตอนต่อมาจะทำการกำหนด Swap Join Index โดยจะทำการเลือกจากโหนด Join (111) ที่อยู่ใกล้กับ Swap Index 2 ซึ่งนับย้อนหลังกลับมาดังภาพที่ 34



ภาพที่ 34 ทำการกำหนด Swap Join index

จากนั้นทำการสร้าง Chromosome ใหม่โดยการ Copy ข้อมูลหรือ Element จาก Array เดิมโดยมีเงื่อนไขว่าจะต้องไม่เท่ากัน Swap Index 1, Swap Index 2 และ Swap Join Index และหาก Element นั้นอยู่ตรงกับ Swap Index 2 ให้ทำการแทนที่ด้วย Sub-tree ซึ่งจะได้ลัพธ์ของการ Mutation ตามภาพที่ 35

ภาพที่ 35 ผลลัพธ์จากการ Mutation

Selection สำหรับขั้นตอนกระบวนการเลือก Chromosome ต้นแบบเพื่อมาดำเนินการโดยใช้ Genetic Operator ผู้วิจัยใช้วิธีการคัดเลือกแบบ Tournament Selection ซึ่งเป็นการสุ่มเลือก Chromosome ต้นแบบมา 1 กลุ่มแล้วทำการเลือก Chromosome ตัวที่ดีที่สุดภายในกลุ่มที่สุ่มเลือกมาใช้สำหรับเป็น Chromosome ต้นแบบสำหรับดำเนินการต่อไป

สำหรับกระบวนการ Genetic Algorithm ผู้วิจัยได้ทำการเลือกใช้งานโปรแกรม Genetic Algorithm ที่มีลักษณะเป็น Open Source ซึ่งการพัฒนาโปรแกรมสามารถเป็นไปแบบต่อยอด โดยการพัฒนาต่อจากสิ่งที่มีผู้ได้ทำการพัฒนาเอาไว้แล้ว

โดยผู้วิจัยได้ทำการเลือก TinyGP (Poli, Langdon and McPhee 2008) ซึ่งผู้วิจัยได้นำมาทำการดัดแปลงให้สามารถทำงานได้ตามลักษณะที่ผู้วิจัยได้ทำการกำหนดเอาไว้ดังแสดงในข้างต้น

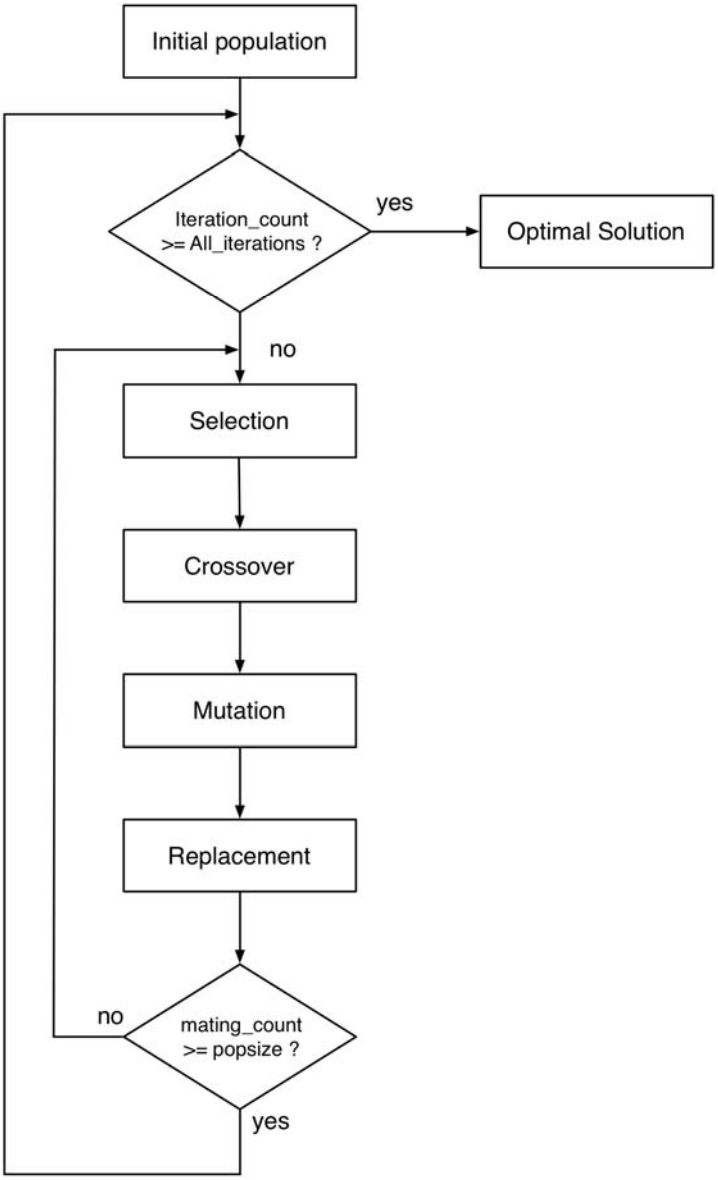
และสำหรับการกำหนดปัจจัย (Parameters) ต่างๆ ที่มีผลต่อการทำงานของ Genetic Algorithm ดังตารางที่ 1

ตารางที่ 1 การตั้งค่า Parameter สำหรับ Genetic Algorithm

Parameter	**Value**
Selection	Tournament size = 10
Crossover	Sub-tree crossover
Mutation	Swap-node mutation
Replacement	Steady-state GA
Population size	512
Crossover Probability	0.65

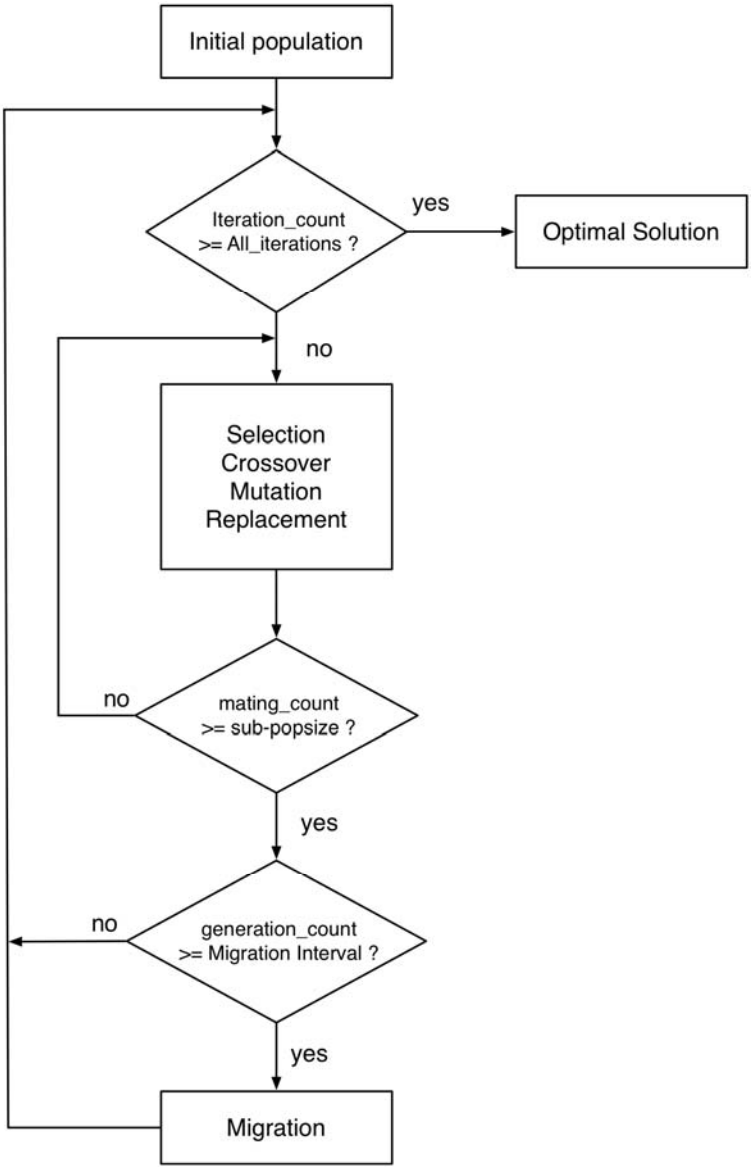
กระบวนการทำงานของ Island Based Parallel Genetic Algorithm

สำหรับกระบวนการทำงานของ Island Based Parallel Genetic Algorithm ที่ผู้วิจัยใช้ในงานวิจัยชิ้นนี้ อยู่บนพื้นฐานการทำงานหลักของ Steady-State Genetic Algorithm ซึ่งสามารถอธิบายการทำงานได้ดังภาพที่ 36 (Lu and Areibi 2004)



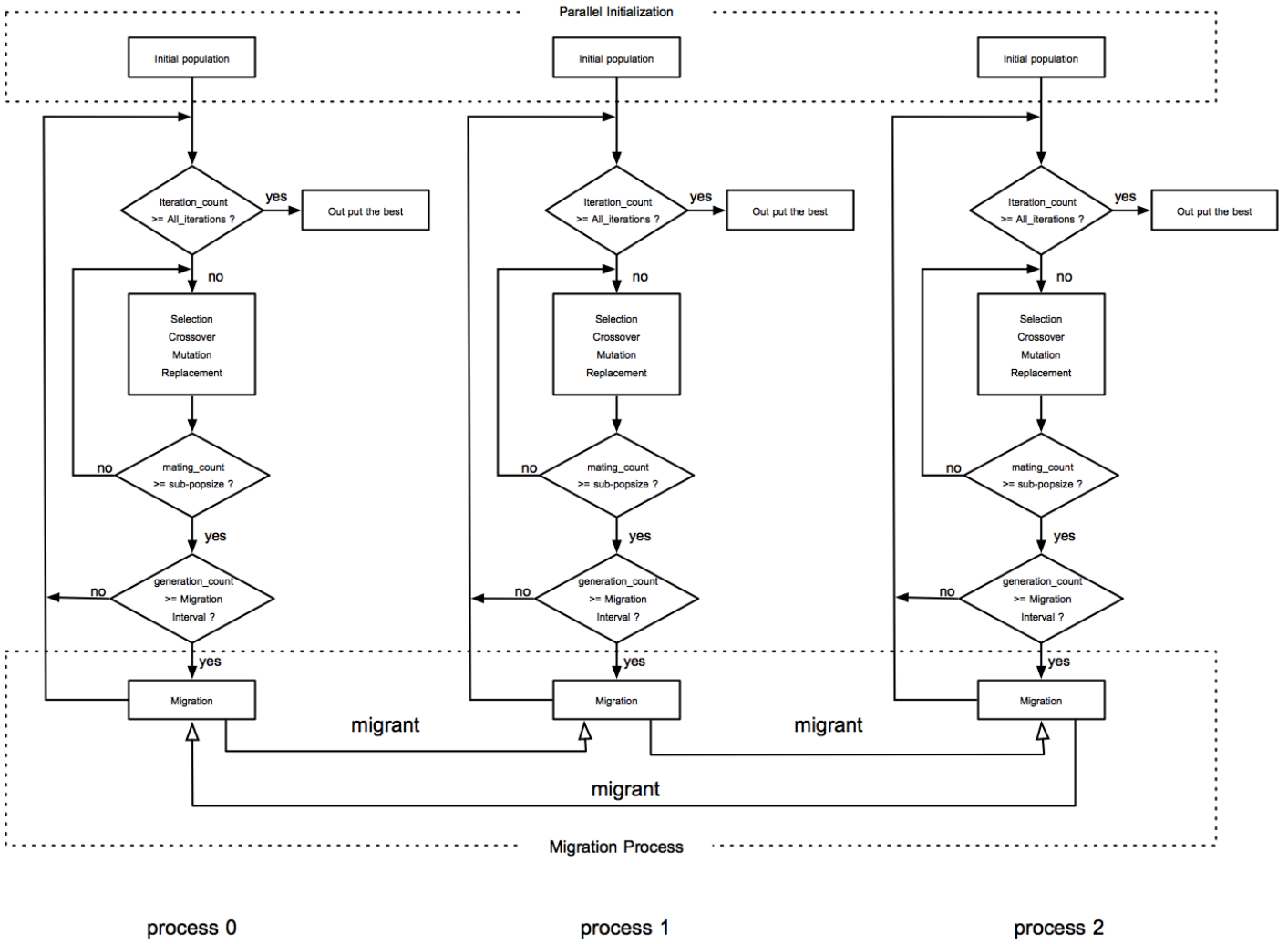
ภาพที่ 36 Steady-States Genetic Algorithm

จากภาพที่ 36 กระบวนการสำหรับ Island-based GA จะเป็นการเพิ่มส่วนการทำงานของการ Migration เข้าไปดังสามารถแสดงได้ดังภาพที่ 37 (Lu and Areibi 2004)



ภาพที่ 37 Island Based Genetic Algorithm

จากภาพที่ 37 แสดงให้เห็นถึงกระบวนการการทำงานของ Island base GA ที่มีการเพิ่มส่วนของการ Migration เข้าไปในกระบวนการ Steady state GA โดยกระบวนการ Migration นี้จะทำหน้าที่สำหรับแลกเปลี่ยน Individual ซึ่งให้ค่า Fitness ที่ดีที่สุด กับ Island base GA กระบวนการอื่น โดยสามารถแสดงได้ดังภาพที่ 38 (Lu and Areibi 2004) ซึ่งแสดงกระบวนการ Island base GA จำนวน 3 Process



ภาพที่ 38 Island-based Parallel Genetic Algorithm จำนวน 3 Process

โดยสามารถแสดงการตั้งค่า Parameters สำหรับการทำงานของ Island-based PGA ได้ดังตารางที่ 2

ตารางที่ 2 การตั้งค่า Parameter สำหรับ Island-based PGA

Parameter	**Value**
Selection	Tournament size = 10
Crossover	Sub-tree crossover
Mutation	Swap-node mutation
Replacement	Steady-state GA
Population size	512
Crossover Probability	0.65
Migration Rate	4
Migration Interval	20

Simulated Annealing

กระบวนการ Simulated Annealing (SA) ที่ผู้วิจัยใช้สำหรับดำเนินการในงานวิจัยชิ้นนี้ เพื่อใช้สำหรับเปรียบเทียบประสิทธิภาพการทำงานกับ Genetic Algorithm ทั้งในด้านความรวดเร็วในการค้นหาผลลัพธ์ และคุณภาพของผลลัพธ์ที่ได้จากการค้นหา

โดยผู้วิจัยได้ยึด กระบวนการ Simulated Annealing (SA) ที่ได้ออกแบบและตั้งค่า Parameter โดย Ioannidis and Kang (1990) ซึ่งสามารถแสดงได้ดังภาพที่ 39 (Ioannidis and Kang, 1990)

Procedure SA() {

$S = S_0$,

$T = T_0$,

minS = S,

while not (frozen) do {

while not (equilibrium) do {

```
                    S' = random state in neighbors(S),
                    ΔC = cost(S') – cost(S),
                    If(ΔC <= 0) then S = S',
                    If(ΔC > 0) then S = S', with probability e^(-ΔC/T),
                    If cost(S) < cost(minS) then minS = S,
                }
                T = reduce(T),
            }
            return(minS),
        }
```

ภาพที่ 39 Simulated Annealing Algorithm

โดยสามารถแสดงการตั้งค่า Parameters ที่ใช้สำหรับ SA ได้ดังตารางที่ 3 ได้ดังนี้ (Ioannidis and Kang, 1990)

ตารางที่ 3 การตั้งค่า Parameters สำหรับ SA

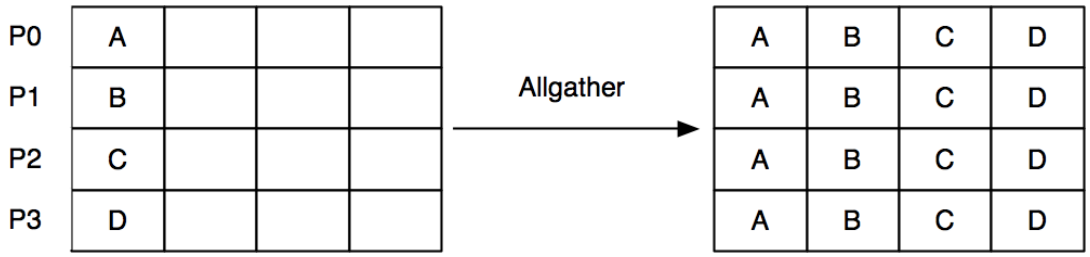
Parameters	**Value**
Initial State S_0	Random
Initial Temperature	2 * cost(S_0)
Frozen	T < 1 and minS unchanged for 4 stages
Equilibrium	16 * (number of join query)
Next state	Random neighbor
Temperature reduction	$T_{new} = 0.95 * T_{old}$

วิธีการ Initial Population โดยใช้วิธีการแบบขนาน

ผู้วิจัยได้ใช้ประโยชน์จากการคำนวณแบบขนานในการเพิ่มประสิทธิภาพการทำงานให้กับ Island Base GA ในแต่ละ Process ให้สามารถทำงานได้รวดเร็วขึ้น สำหรับขั้นตอน Initial Population

โดยผู้วิจัยได้ออกแบบการทำงานสำหรับขั้นตอน Initial Population ให้แต่ละ Process ทำการ Generate ประชากรต้นแบบของตัวเองในอัตราส่วนหนึ่ง เมื่อทำการสร้างกลุ่มประชากรต้นแบบสำหรับตัวเองเสร็จแล้ว แต่ละ Process จะทำการกระจายข้อมูลกลุ่มประชากรต้นแบบของตนเองไปให้กับ Process อื่นๆ

สำหรับในขั้นตอนนี้ผู้วิจัยได้ใช้ความสามารถในการรับส่งข้อมูลของ Message Passing Interface (MPI) เวอร์ชัน 2 โดยผู้วิจัยเลือกใช้การรับส่งข้อมูลแบบ All Gather ซึ่งจะเป็นการสื่อสารแบบ Collective Communication ซึ่งสามารถแสดงวิธีการสื่อสารแบบ All Gather ได้ดังภาพที่ 40 ดังนี้

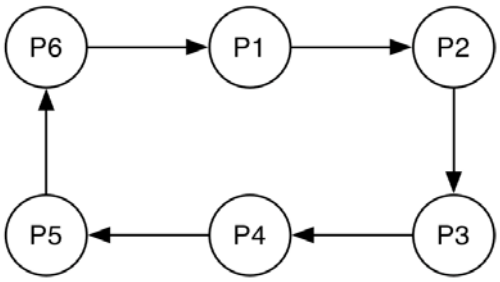


ภาพที่ 40 การสื่อสารด้วยวิธี All Gather

จากภาพที่ 40 แสดงวิธีการรับส่งข้อมูลด้วยวิธี All Gather ซึ่งจะเห็นว่า แต่ละ Process คือ P0, P1, P2, P3 จะทำการสร้างข้อมูลกลุ่มประชากรต้นแบบของตนแบบพร้อมๆ กัน เมื่อแต่ละ Process ทำการสร้างกลุ่มประชากรต้นแบบของตนเองเรียบร้อยแล้วจะทำการส่งข้อมูล ด้วยวิธี All Gather ซึ่งผลลัพธ์ที่ได้ จะทำให้ทุกๆ Process มีกลุ่มประชากรต้นแบบที่เหมือนกันในการเรียงลำดับที่เหมือนกัน

รูปแบบการสื่อสารข้อมูลในกระบวนการ Migration

ผู้วิจัยได้เลือกใช้รูปแบบในการสื่อสารข้อมูลสำหรับกระบวนการ Migration ในรูปแบบ Ring Topology ซึ่งจะเป็นการรับส่งข้อมูลเป็นแบบวงกลมซึ่งสามารถแสดงได้ดังภาพที่ 41 ดังนี้



ภาพที่ 41 รูปแบบการสื่อสารแบบ Ring Topology

โดยผู้วิจัยสามารถอธิบายวิธีการแลกเปลี่ยนข้อมูลในกระบวนการ Migration จากภาพที่ 41 ได้ดังนี้คือ GA Process ที่กำลังทำงานอยู่จะทำการแลกเปลี่ยนข้อมูลกับ GA Process ข้างเคียงที่อยู่ถัดไป เมื่อทำการส่งข้อมูลออกไปให้ Process ข้างเคียง ก็ทำการรอรับข้อมูลจากรอรับข้อมูลจาก Process อื่นที่อยู่ถัดมา ยกตัวอย่างเช่น Prcess P2 จะทำการส่งข้อมูล Migrant ออกไปให้กับ P3 เมื่อทำการส่งข้อมูลไปให้ P3 แล้วก็จะทำการรอรับข้อมูลที่ถูกส่งออกมาจาก P1 เป็นต้น เมื่อทำการรับข้อมูลจาก P1 เรียบร้อยแล้วก็จะดำเนินกระบวนการของตนเองต่อไป โดยจำนวนข้อมูลที่มีการแลกเปลี่ยนกันระหว่าง Process แต่ละครั้งหรือว่าจำนวน Migrant จะมีจำนวนเท่ากับ 4 Individual

ซึ่งข้อดีของกระบวนการ Migration ที่รูปแบบในการแลกเปลี่ยนข้อมูลแบบ Ring Topology นี้คือง่ายต่อการศึกษาและง่ายต่อการ Implement แต่ข้อเสียคือ การรับส่งข้อมูลนั้นจะต้องการหยุดรอกันในการรับส่งข้อมูลซึ่งทำให้การทำงานเป็นไปแบบไม่ต่อเนื่อง และหามี Process ใด Process หนึ่งไม่สามารถทำงานต่อไปได้ หรือถูก Interrupt ก็จะทำให้กระบวนการทั้งหมดหยุดชะงักลงไป

วิธีการคำนวณขนาดของ Intermediate Relation

การคำนวณขนาดของ Intermediate Relation ซึ่งเป็นผลลัพธ์ที่เกิดขึ้นจากการ Join ระหว่างคู่ของ Relations มีความสำคัญอย่างยิ่งต่อการทำนายค่า Cost ที่จะเกิดขึ้นจาก QEP แต่ละตัว

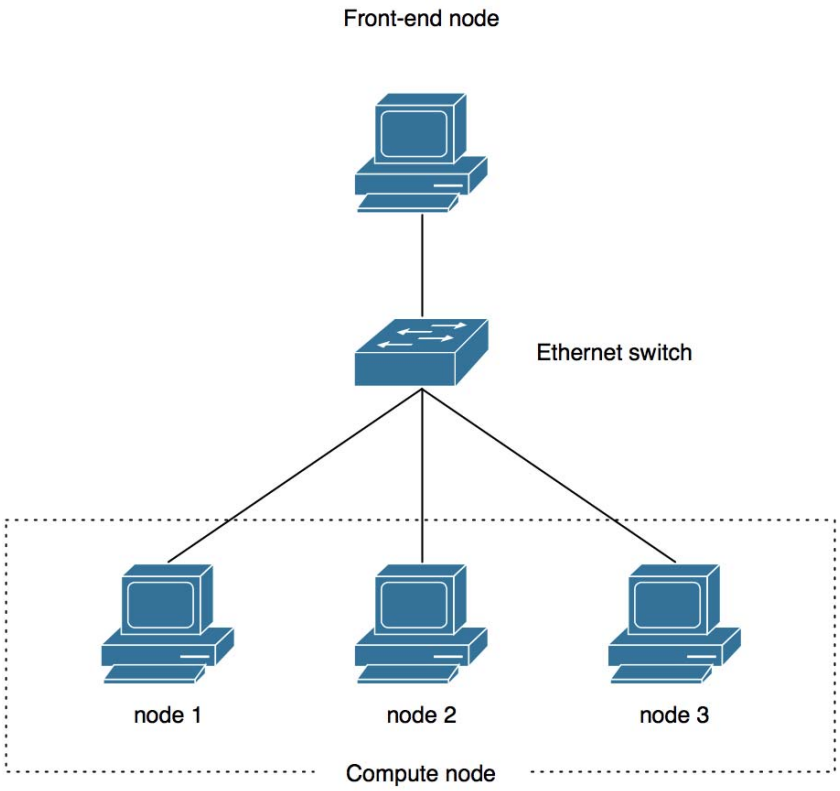
ซึ่งขนาดของ Intermediate Relation ที่เกิดขึ้นจะมีผลต่อเวลาที่จะต้องสูญเสียไปในการขนส่งข้อมูลผ่านระบบ Network ซึ่งหากขนาดของ Intermediate Relation มีขนาดใหญ่ก็จะต้องสูญเสียเวลาในการขนส่งข้อมูลมาก แต่หากขนาดของ Intermediate Relation มีขนาดเล็กก็จะทำให้การส่งข้อมูลเป็นไปได้อย่างรวดเร็ว โดยวิธีการคำนวณขนาดของ Intermediate Relation ที่ผู้วิจัยได้ใช้ในงานวิจัยนี้ผู้วัยได้อ้างอิงมาจากงานวิจัยเรื่อง Optimizing distributed join queries: A genetic algorithm approach โดย Rho and March (1997)

จำนวนหน่วยประมวลผล

การกำหนดจำนวนหน่วยประมวลเป็นอีกหนึ่งปัจจัยการทดลองที่สำคัญสำหรับ Island-based PGA ซึ่งจะมีการแบ่งการทำงานออกเป็นโปรเซสย่อยๆ ทำงานอยู่บนหน่วยประมวลผลต่างๆ ถึงแม้แต่ละโปรเซสจะทำงานอย่างเป็นอิสระกัน แต่จำเป็นต้องมีการแลกเปลี่ยนข้อมูลเพื่อทำให้มีการวิวัฒนาการที่ดียิ่งขึ้นโดยใช้ส่วนดีร่วมกัน การสื่อสารข้อมูลระหว่างกันจะต้องแลกเปลี่ยนข้อมูลผ่านสายสัญญาณเครือข่ายภายในของระบบคลัสเตอร์ ซึ่งหากมีการสื่อสารข้อมูลเป็นจำนวนมากพร้อมๆ กันอาจทำให้รับส่งข้อมูลระหว่างกันล่าช้าตามไปด้วย และอาจเป็นสาเหตุที่ทำให้กระบวนการต่างๆ ของ Island-based PGA ล่าช้าตามไปดังนั้นผู้วิจัยจึงได้เลือกที่จะทำการใช้จำนวนของหน่วยประมวลผลเพียง 4 หน่วยประมวลผล และสำหรับการทำงานของ Island-based PGA จะแบ่งการทำงานออกเป็น 4 โปรเซสย่อยต่อการทำงานหนึ่งครั้ง

กำหนดสถาปัตยกรรมของการทดลอง

ผู้วิจัยได้ทำการกำหนดสถาปัตยกรรมของการทดลอง โดยสามารถอธิบายได้ดังภาพที่ 42 ดังนี้



ภาพที่ 42 สถาปัตยกรรมในการทดลอง

จากภาพที่ 42 สามารถอธิบายได้ถึงสถาปัตยกรรมของระบบสำหรับการคำนวณแบบขนานได้ดังนี้คือ ในการดำเนินการประมวลผลการ Island-based PGA แต่ละครั้งจะมีการแบ่ง GA Process ออกเป็น Process ย่อยๆ ทั้งหมด 4 Process และจะกระจายออกไปดำเนินการอยู่บนแต่ละ Node โดยแต่ละ Node จะมี GA Process ดำเนินการอยู่เพียงแต่ 1 Process เท่านั้นสำหรับแต่ละกระบวนการ Island-based PGA

บทที่ 5

ผลการดำเนินงานวิจัย

จากที่ผู้วิจัยได้ทำการจำลองแบบการทำงานของ Distributed Query Optimizer ตามทฤษฎี และโครงสร้างที่ผู้วิจัยได้นำเสนอแล้ว ผู้วิจัยได้ทำการดำเนินการทดลอง และรวบรวมผลการทดลอง โดยสามารถจำแนกออกเป็นหมวดหมู่ดังต่อไปนี้

1. วัดประสิทธิภาพด้านความรวดเร็วในการให้ผลลัพธ์
2. วัดประสิทธิภาพด้านคุณภาพของผลลัพธ์
3. วัดประสิทธิภาพด้านการรองรับงานปริมาณมาก

การวัดประสิทธิภาพด้านความเร็วในการให้ผลลัพธ์

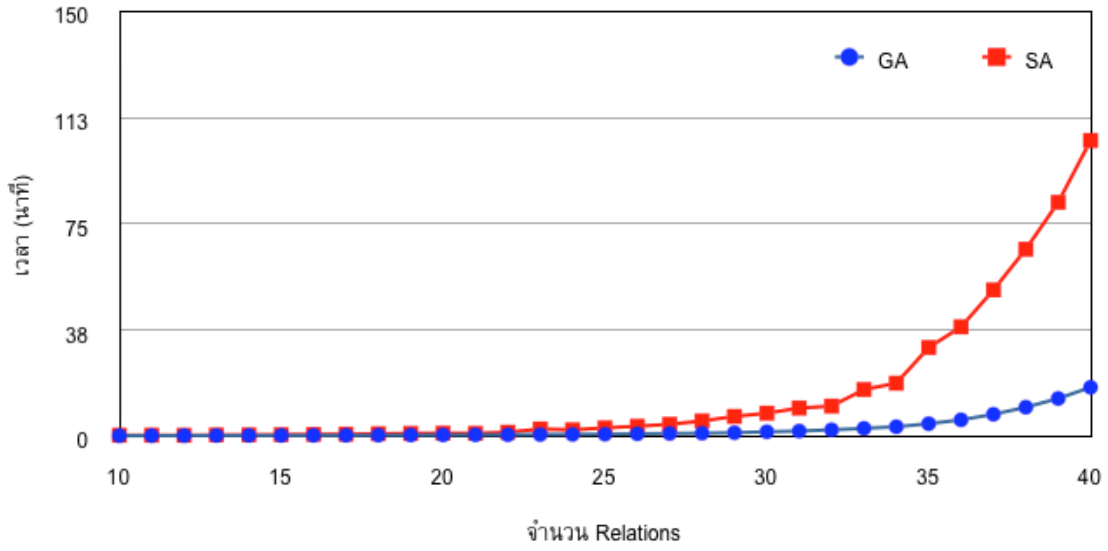
สำหรับวิธีการในการทดสอบความรวดเร็วในการทำงานของ Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบต่างๆ นั้นผู้วิจัยได้ทำการออกแบบการทดลองโดยการให้ Distributed Query Optimizer ต่างๆ ทำการค้นหาคำตอบของแบบสอบถามข้อมูล (Input Query) ที่มีการ JOIN ของ Relations ขนาดต่างๆ ตั้งแต่ 10 – 40 Relations และเพื่อให้ได้เวลาเฉลี่ยของการค้นหาคำตอบ ผู้วิจัยได้ทำการทดลองโดยกำหนดให้ Distributed Query Optimizer แต่ละแบบ ทำการค้นหาคำตอบ และบันทึกเวลาทั้งหมด 10 ครั้งในแต่ละ Input Query

โดยค่าเฉลี่ยของเวลาในการทำงานเพื่อค้นหาคำตอบที่ได้จากการทดสอบภายในงานวิจัยนี้ของ Query Optimizer ที่ใช้วิธีการค้นหาทั้ง 3 แบบที่แตกต่างกันนั้น สามารถแสดงได้ดังตารางที่ 4 ดังต่อไปนี้

ตารางที่ 4 เปรียบเทียบประสิทธิภาพด้านเวลาของการทำงานระหว่าง SA, GA และ Island-based PGA

	เวลาเฉลี่ยของการทำงาน (นาที)		
จำนวน Relation	SA	GA	Island-based PGA
10	0.273	0.133	0.144
11	0.313	0.149	0.160
12	0.265	0.164	0.176
13	0.382	0.182	0.203
14	0.402	0.208	0.216
15	0.502	0.227	0.243
16	0.607	0.252	0.269
17	0.663	0.277	0.294
18	0.823	0.312	0.325
19	0.928	0.35	0.370
20	1.018	0.383	0.392
21	0.948	0.429	0.422
22	1.367	0.469	0.464
23	2.493	0.537	0.485
24	2.214	0.583	0.518
25	2.891	0.658	0.589
26	3.463	0.793	0.648
27	4.171	0.912	0.710
28	5.32	1.008	0.744
29	6.944	1.202	0.823
30	8.077	1.517	0.935
31	9.898	1.786	1.113
32	10.58	2.203	1.217
33	16.427	2.714	1.411
34	18.675	3.303	1.677
35	31.261	4.371	1.954
36	38.56	5.734	2.321
37	51.58	7.682	2.935
38	65.978	10.185	3.798
39	82.517	13.263	4.617
40	104.351	17.223	5.881

ผู้วิจัยได้เริ่มทำการทดสอบเพื่อวัดประสิทธิภาพการทำงานด้านความรวดเร็วระหว่าง Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ Simulated Annealing (SA) และแบบที่ใช้วิธีการค้นหาแบบ Genetic Algorithm (GA) โดยสามารถนำเสนอผลของการเปรียบเทียบเวลาที่ใช้สำหรับการค้นหาคำตอบซึ่งเป็นการนำเอาข้อมูลจากตารางที่ 4 มาแสดงในลักษณะของ กราฟความสัมพันธ์ ได้ดังภาพที่ 43 ดังนี้



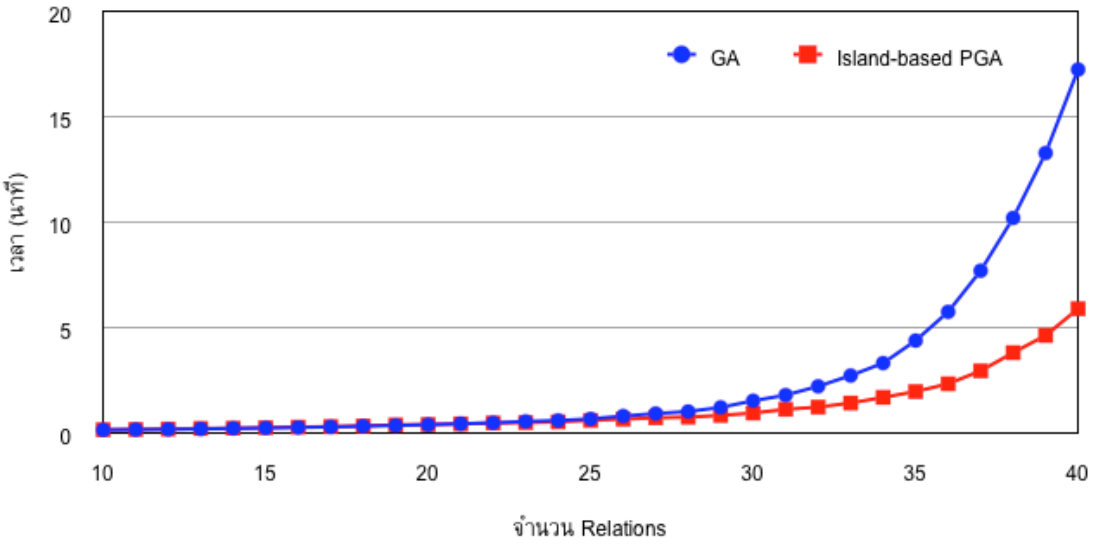
ภาพที่ 43 ผลการเปรียบเทียบประสิทธิภาพด้านเวลาเฉลี่ยที่ใช้ค้นหา Optimal QEP ระหว่าง Distributed Query Optimizer ที่ใช้ GA และ SA

จากภาพที่ 43 เป็นกราฟแสดงผลด้านความเร็วในการทำงาน จะเห็นได้ชัดเจนว่า Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ GA จะให้เวลาเฉลี่ยในการค้นหาผลลัพธ์ได้รวดเร็วกว่าแบบที่ใช้ SA

ผลลัพธ์ที่แสดงในภาพที่ 43 พบว่า GA สามารถทำให้ประสิทธิภาพการทำงานของ Query Optimizer ดีขึ้นจริง และเพื่อเป็นการพิสูจน์ข้อสมมติฐานของงานวิจัยชิ้นนี้ว่า การนำเอาความสามารถทางด้านการคำนวณแบบขนาน (Parallel Computing) เข้ามาประยุกต์ในการทำงานจะทำให้ประสิทธิภาพของการทำงานของ Query Optimizer ดีขึ้น ซึ่งในงานวิจัยนี้ได้นำเอา GA ซึ่งจาก

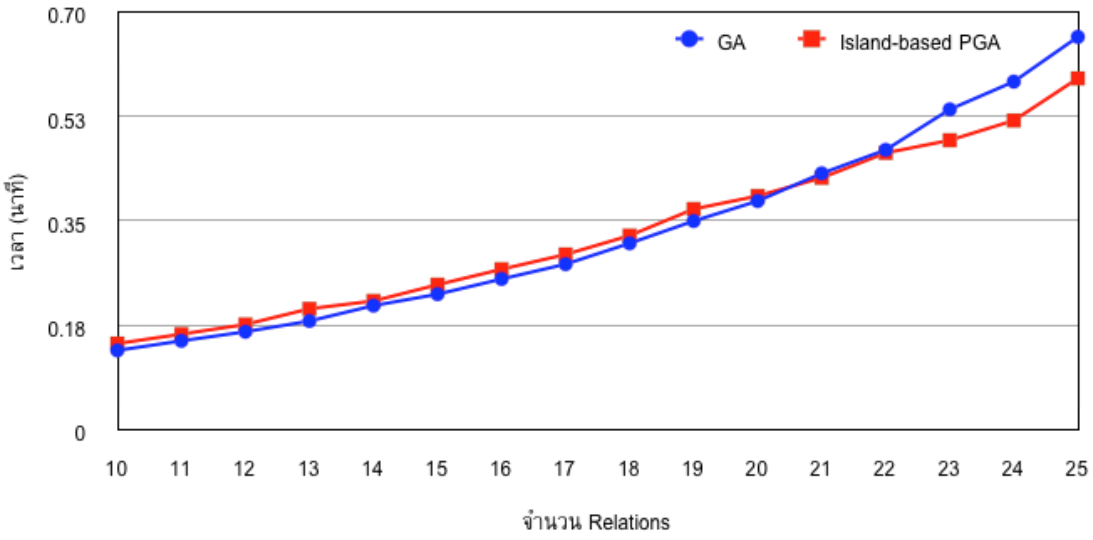
การทดลองที่ผ่านมา พบว่าเป็น Search Algorithm ที่มีความเหมาะสม และมีประสิทธิภาพในการแก้ปัญหานี้ เข้ามาดัดแปลงให้มีการคำนวณ และการทำงานแบบขนาน

ดังนั้นสำหรับการทดลองเพื่อเปรียบเทียบประสิทธิภาพในด้านความรวดเร็วในการค้นหาผลลัพธ์ ในขั้นต่อไปผู้วิจัยจึงเลือกที่จะเปรียบเทียบการทำงานระหว่าง Distributed Query Optimizer แบบที่ใช้วิธีการค้นหาแบบ GA และ Island-based Parallel Genetic Algorithm (Island-based PGA) โดยการนำเอาข้อมูลจากตารางที่ 4 มาทำการสร้างกราฟความสัมพันธ์เพื่อเปรียบเทียบเวลาเฉลี่ยที่ใช้ในการทำงาน โดยสามารถแสดงผลลัพธ์ของการทำงานได้ดังภาพที่ 44 ต่อไปนี้



ภาพที่ 44 ผลการเปรียบเทียบประสิทธิภาพด้านเวลาเฉลี่ยที่ใช้ค้นหา Optimal QEP ระหว่าง Distributed Query Optimizer ที่ใช้ GA และ Island-based PGA

จากภาพที่ 44 ผลการทดลองได้แสดงให้เห็นว่า โดยภาพรวม Island-based PGA จะให้เวลาเฉลี่ยในการค้นหาดีกว่า โดยจะให้ผลที่แตกต่างอย่างชัดเจนในช่วงที่มีการ JOIN ของ Relations ตั้งแต่ 22 Relation เป็นต้นไป แต่หากขยายขนาดของการแสดงผลลัพธ์ในกราฟความสัมพันธ์ในช่วง 10 - 25 ผลลัพธ์ที่ได้จะสามารถบ่งบอกถึงจุดที่คุ้มค่าต่อการนำเอาการคำนวณแบบ Parallel Computing เข้ามาใช้ในการแก้ปัญหาในเรื่อง Distributed Query Optimization ได้อย่างชัดเจนยิ่งขึ้น



ภาพที่ 45 ผลการเปรียบเทียบประสิทธิภาพด้านเวลาเฉลี่ยที่ใช้ค้นหา Optimal QEP ระหว่าง GA และ Island-based PGA

จากภาพที่ 45 แสดงให้เห็นว่าในช่วงที่มีการ JOIN ระหว่าง 10 – 21 Relations Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ GA สามารถให้ผลลัพธ์ที่รวดเร็วกว่าแบบที่ใช้ Island-based PGA ที่เป็นเช่นนี้เพราะในการทำงานแบบขนานบนสถาปัตยกรรมแบบ Share nothing นั้นจะต้องมีการสื่อสารหรือส่งข้อมูลหากันระหว่าง Process ดังนั้นค่าใช้จ่ายที่จะต้องมีเพิ่มเข้ามาก็คือ เวลาที่ใช้ในการรับส่งข้อมูลหากันระหว่างแต่ละ Process ผ่านการเชื่อมต่อร่วมกันโดยในช่วงข้อมูลที่มีการ Join ของ Relations ที่มีจำนวนไม่มากนักคือ 10 – 21 Relations เป็นช่วงที่การทำงานระหว่าง GA และ Island-based PGA ให้เวลาการทำงานที่เกือบจะเท่ากัน แต่ในส่วนของ Island-based PGA นั้นจะมีเวลาในส่วนของการรับส่งข้อมูลระหว่างแต่ละ Process เพิ่มเข้ามาทำในช่วง 10 – 21 Relations นั้น Island-based PGA มีเวลาในการทำงานที่มากกว่า GA ซึ่งในส่วนนี้ทำให้ผู้วิจัยสามารถสรุปผลจากการทดลองในเรื่องของความเร็วในการทำงานได้ว่า ในช่วง 10 – 21 Relation ข้อมูลต่างๆ เช่น ขนาดของ QEP ที่อยู่ในรูปของ Tree มีขนาดที่ไม่ใหญ่มาก การดำเนินการต่างๆ เช่น การสร้าง QEP ในกระบวนการ Initial Population และการดำเนินการในขั้นตอน Crossover และ Mutation จึงสามารถทำได้ง่ายและรวดเร็ว จนการทำงานแบบ Parallel Computing ไม่ได้มีผลต่อประสิทธิภาพในการทำงาน

แต่สำหรับในกรณีที่จำนวนของ Relations ที่มีจำนวนตั้งแต่ 21 - 40 Relations นั้นสิ่งที่ได้จากผลของการทดลองทำให้ผู้วิจัยวิเคราะห์ได้ว่า รูปร่างของ QEP ซึ่งอยู่ในรูปของ Tree มีความซับซ้อนมากขึ้น ซึ่งจะต้องเสียเวลาเพิ่มขึ้นมากสำหรับในแต่ละกระบวนการเช่น Initial Population, Crossover และ Mutation ซึ่งความซับซ้อนในรูปร่างของ QEP นี่เองที่กลายมาเป็นปัจจัยหลักที่มีผลต่อเวลาทำให้เวลาที่ต้องเสียไปในการดำเนินการเพิ่มมากขึ้นและ Distributed Query Optimizer แบบที่ใช้ Island-based PGA สามารถทำงานได้อย่างรวดเร็วกว่าเนื่องจากว่า ในขั้นตอนของการ Initial Population นั้นผู้วิจัยได้กำหนดให้แต่ละ Process ของกระบวนการทำการสร้าง QEP ขึ้นมาแค่บางส่วนเท่านั้น และทำการแลกเปลี่ยน QEP ของตนเองกับ Process ข้างเคียงซึ่งทำงานอยู่คนละหน่วยประมวลผล จนได้จำนวน QEP ครบจำนวนตามขนาดของ Population Size ที่กำหนดเอาไว้ ซึ่งด้วยวิธีนี้ทำให้แต่ละ Process สามารถสิ้นสุดกระบวนการ Initial Population ไปได้อย่างรวดเร็ว ซึ่งส่งผลให้เวลาที่ใช้ในการค้นหา Optimal QEP มีความแตกต่างกับแบบที่ใช้ GA อย่างชัดเจน

การวัดประสิทธิภาพด้านคุณภาพของผลลัพธ์

ผู้วิจัยได้ทำการทดลองวัดประสิทธิภาพด้านคุณภาพของผลลัพธ์ที่ได้จาก Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ SA, GA และ Island-based PGA โดยได้ออกแบบให้ Distributed Query Optimizer ทั้ง 3 แบบ ทำงานบน Search Space เดียวกัน หรืออาจกล่าวได้ว่ามีการทำงานโดยใช้ Database Profile เดียวกันเพื่อหาคำตอบของ Input Query เดียวกัน จากนั้นนำผลลัพธ์ที่ได้จาก Distributed Query Optimizer ทั้ง 3 แบบมาทำการเปรียบเทียบคุณภาพกันเพื่อสรุปผลว่า Distributed Query Optimizer แบบใดให้คุณภาพของผลลัพธ์ดีกว่ากัน

ข้อสมมติและปัจจัยแวดล้อมในการทดลอง

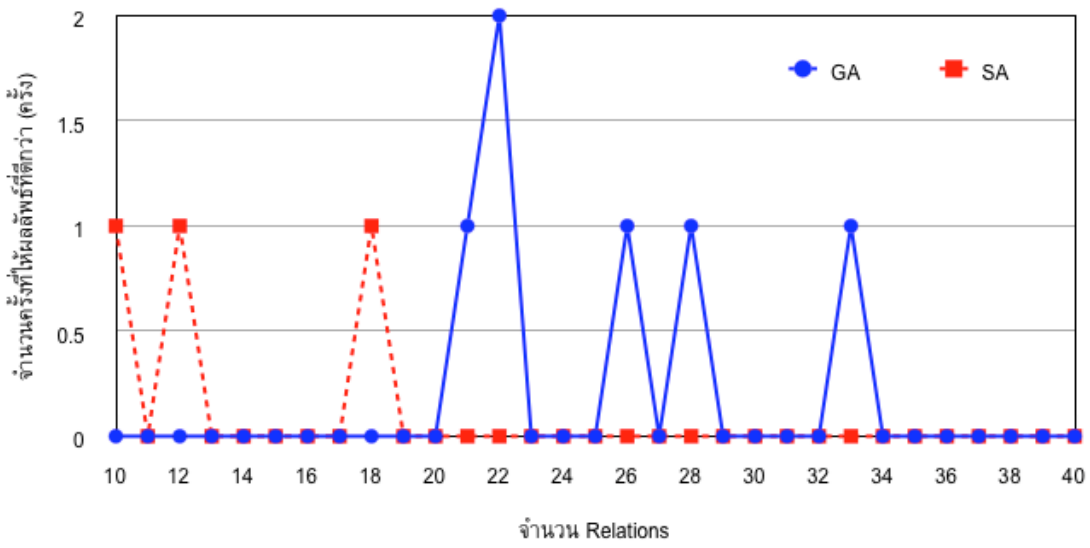
เนื่องจากผู้วิจัยได้ทำการทดลองภายใต้สภาวะแวดล้อมที่ได้ทำการจำลองแบบจากการทำงานของกระบวนการ Query Optimization จึงมีข้อจำกัดหลายประการที่เกิดขึ้นจากข้อกำหนดต่างๆ เช่น ในการ Generate QEP ขึ้นมาใน Search space นั้นจะเป็นการสร้างขึ้นจากข้อมูล Database Profile ซึ่งเป็นชุดตัวเลขที่ได้ทำการสุ่มค่าขึ้นมาจากช่วงที่กำหนด และในการสุ่มสร้าง Database Profile แต่ละครั้งค่าที่ได้จะเปลี่ยนแปลงไปตลอด ดังนั้นหากพิจารณาเซตของ QEPs ที่

เกิดขึ้น หรือ Search space ที่เกิดขึ้นนั้นจะไม่มีทางทราบค่าขอบเขตบน และขอบเขตล่างของค่า Cost ที่เป็นไปได้ของ Search space ที่เกิดขึ้น ดังนั้นผลลัพธ์ที่ได้จากการค้นหาจาก Distributed Query Optimizer ที่ใช้ Search Algorithm ที่ผู้วิจัยพิจารณาแต่และแบบ จึงเป็นการยากที่จะสามารถบ่งชี้ได้ว่าผลลัพธ์ที่ได้นั้น เป็นผลลัพธ์ที่ให้ค่า Cost ที่เป็นขอบเขตล่าง หรือเข้าใกล้กับขอบเขตล่างมากน้อยเพียงไร

ดังนั้นในการทดลองเพื่อให้ได้ข้อสรุปของการวัดประสิทธิภาพที่มีความแน่นอน ผู้วิจัยได้ออกแบบการทดลองให้ Distributed Query Optimizer ทั้ง 3 แบบ ทำการค้นหาคำตอบเมื่อใช้ Input Query เดียวกันโดยการมีสร้าง Database Profile ที่แตกต่างกัน 5 ชุดสำหรับแต่ละ Input Query จากนั้นจึงทำการนับจำนวนความถี่ของคำตอบที่เป็นคำตอบที่ดีที่สุดในกลุ่มของคำตอบที่ได้จากการทดลอง โดยสามารถแสดงผลลัพธ์ที่ได้ออกมาเป็นกราฟได้ตามหัวข้อการทดลองต่อไปนี้

เปรียบเทียบคุณภาพของผลลัพธ์ระหว่าง GA และ SA

สามารถแสดงผลลัพธ์ของการทดลองได้ดังภาพที่ 46 ดังนี้



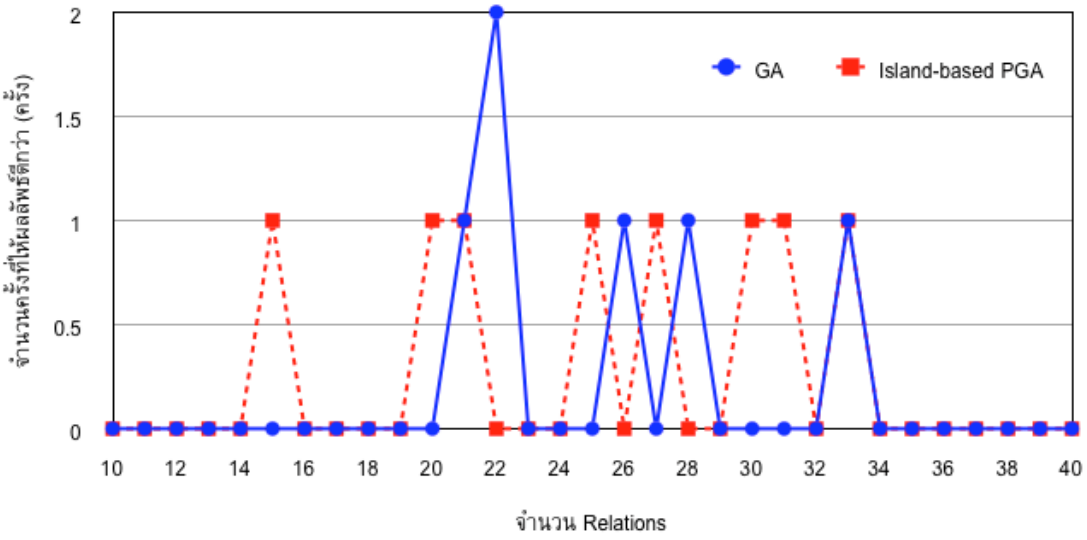
ภาพที่ 46 เปรียบเทียบคุณภาพของผลลัพธ์ระหว่าง GA และ SA

จากภาพที่ 46 เป็นกราฟแสดงความถี่ของการที่ได้คุณภาพของผลลัพธ์ที่ดีกว่า ซึ่งผลจากกราฟแสดงให้เห็นว่า จากการทดลองเปรียบเทียบคุณภาพของผลลัพธ์จาก Input Query ที่มีการ Join ระหว่าง Relations ตั้งแต่ 10 – 40 Relations นั้น Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ GA นั้นมีแนวโน้มที่จะให้คุณภาพผลลัพธ์ที่ดีกว่า Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ SA

จากผลการทดลองเปรียบเทียบประสิทธิภาพด้านคุณภาพของ Optimal QEP ที่ได้จากการค้นหา ดังแสดงได้ในรูปที่ 45 ทำให้ผู้วิจัยสามารถวิเคราะห์ได้ว่า Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ SA นั้นจะทำการพิจารณาเพื่อค้นหา Optimal QEP โดยเลือกจาก State ข้างเคียงของ State หรือ QEP ตัวปัจจุบันที่กำลังพิจารณาอยู่ ซึ่งเป็นการเลือกพิจารณาทีละจุดและเปรียบเทียบกับจุดปัจจุบัน ซึ่งโดยตัวธรรมชาติของปัญหาที่เรากำลังพิจารณาอยู่นั้นมีลักษณะที่มี Search Space ที่ใหญ่มาก และมี QEP ที่เป็นไปได้ภายใน Search Space อย่างหลากหลาย ดังนั้นการพิจารณาเพื่อค้นหา Optimal QEP จาก Search Space ขนาดใหญ่มากโดยการพิจารณาทีละจุดจึงเป็นเรื่องที่ยาก ซึ่งในจุดนี้จะแตกต่างจากการทำงายของ GA ซึ่งจะเป็นวิธีการค้นหาแบบพิจารณาจากจุดการค้นหาทีละหลายๆ จุดพร้อมๆ กัน ซึ่งมีโอกาสที่จะทำการค้นหาไปตลอดทั่วถึงภายใน Search Space ที่เกิดขึ้น เมื่อต้องทำการค้นหาซ้ำกันบ่อยๆ ตลอดเวลาของ Distributed Query Optimizer ที่ใช้วิธีการค้นหาทั้งสองแบบ และนำผลลัพธ์มาเปรียบเทียบกับ จึงทำให้ Distributed Query Optimizer แบบที่ใช้ GA มีโอกาสจะหา Optimal QEP ที่ให้คุณภาพที่ดีกว่าแบบที่ใช้ SA

เปรียบเทียบคุณภาพของผลลัพธ์ระหว่าง GA และ Island-based PGA

สามารถแสดงผลลัพธ์ของการทดลองได้ดังภาพที่ 47 ดังนี้



ภาพที่ 47 เปรียบเทียบคุณภาพของผลลัพธ์ระหว่าง GA และ Island-based PGA

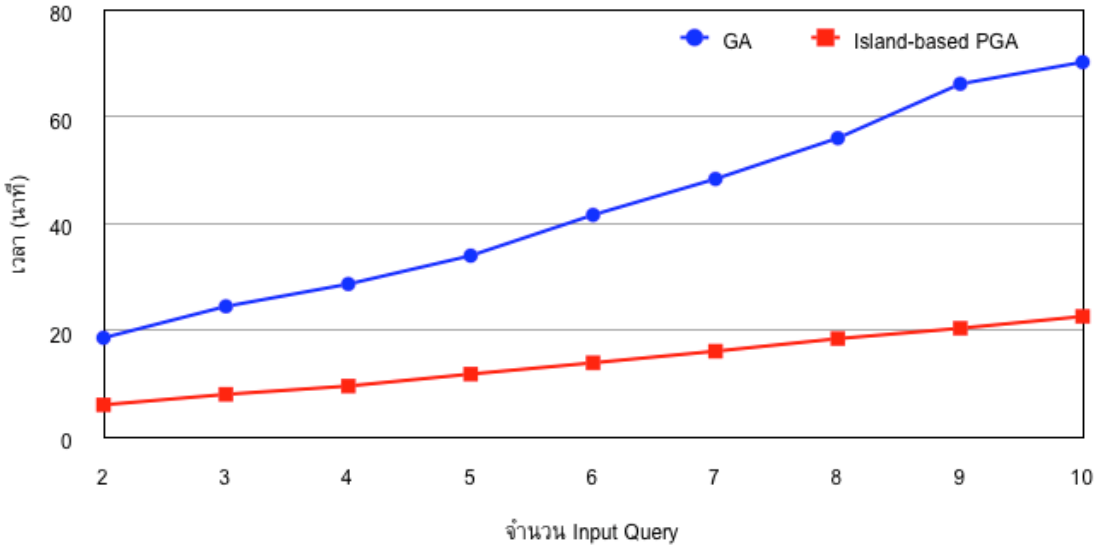
จากภาพที่ 47 เป็นกราฟแสดงความถี่ของการที่ได้คุณภาพของผลลัพธ์ที่ดีกว่า ซึ่งผลลัพธ์จากกราฟแสดงให้เห็นว่า Island-based PGA มีแนวโน้มที่จะให้คุณภาพของผลลัพธ์ที่ดีกว่า GA ซึ่งจากผลการทดลองที่ได้ทำให้ผู้วิจัยสามารถวิเคราะห์ได้ว่าสำหรับ Distributed Query Optimizer แบบที่ใช้ Island-based PGA ในการค้นหา Optimal QEP นั้นประกอบไปด้วย GA หลายๆ Process ที่ทำงานอย่างเป็นอิสระกัน ซึ่งด้วยเหตุนี้ทำให้ Island-based PGA มีทิศทางในการค้นหาที่หลากหลายกว่าแบบที่ใช้ GA ซึ่งทำให้ในกรณีที่ต้องมีการค้นหาที่บ่อยครั้งหรือตลอดเวลา Distributed Query Optimizer แบบที่ใช้ Island-based PGA จึงมีแนวโน้มที่จะสามารถให้คุณภาพของ Optimal QEP ที่ดีกว่าแบบที่ใช้ GA

การวัดประสิทธิภาพด้านการรองรับงานปริมาณมาก

ผู้วิจัยได้เลือกที่จะทดสอบประสิทธิภาพด้านการรองรับงานปริมาณมาก โดยทำการทดสอบเฉพาะในกรณีที่เป็นกรณีที่เลวร้ายที่สุดที่จะเป็นไปได้คือ ในกรณีที่มี Input Query ที่มีการ JOIN ข้อมูลทั้งหมด 40 Relations เข้าสู่กระบวนการ Distributed Query Optimization พร้อมกัน

โดยผู้วิจัยได้ทำการส่ง Input Query เข้าไปในกระบวนการ Distributed Query Optimization พร้อมๆ กันตั้งแต่ 2 – 10 Input Query เพิ่มขึ้นตามลำดับ จากนั้นทำการวัดเวลาที่ใช้ใน

การค้นหาคำตอบในกระบวนการ Distributed Query Optimization โดยได้ทำการเปรียบเทียบประสิทธิภาพการทำงานระหว่าง GA และ Island-based PGA ซึ่งสามารถแสดงผลได้ดังต่อไปนี้



ภาพที่ 48 ผลของการวัดประสิทธิภาพด้านการรองรับงานปริมาณมากของ GA และ Island-based PGA

จากภาพที่ 48 แสดงผลเปรียบเทียบประสิทธิภาพในด้านการรองรับงานปริมาณมากระหว่าง GA และ Island-based PGA ซึ่งได้ทำการทดสอบในกรณีที่เป็นกรณีที่เลวร้ายที่สุด (The Worst Case) ซึ่ง Input Query ที่เข้ามาสู่กระบวนการ Distributed Query Optimization นั้นจะเป็นการ JOIN ข้อมูลระหว่าง 40 Relations ทั้งหมด ซึ่งจะต้องใช้การคำนวณที่สูงมากที่สุดสำหรับงานวิจัยนี้

เมื่อวิเคราะห์จากภาพที่ 48 จะแสดงให้เห็นถึงแนวโน้มที่ชัดเจนว่าในกรณีที่มี Input Query เข้าสู่กระบวนการ Distributed Query Optimization พร้อมๆ กันเป็นจำนวนเพิ่มขึ้นเรื่อยๆ ประสิทธิภาพด้านความเร็วในการค้นหา Optimal QEP ของกระบวนการ Distributed Query Optimization ที่ผู้วิจัยได้พัฒนาขึ้นมีแนวโน้มที่จะลดลง แต่กระบวนการ Distributed Query Optimization แบบที่ใช้ Island-based PGA จะมีแนวโน้มว่าจะให้ประสิทธิภาพด้านการรองรับงานปริมาณมากได้ดีกว่าแบบที่ใช้ GA อย่างชัดเจน

จากผลการทดลองที่แสดงได้ในภาพที่ 48 ผู้วิจัยวิเคราะห์ว่า เมื่อมีการคำนวณเพื่อหาคำตอบหรือหา QEP ที่เหมาะสมที่สุดในกรณีที่มีการ Join ของ Relations จำนวนมาก QEP

ที่อยู่ในรูปของ Join Tree ก็จะมีขนาดใหญ่ขึ้นตามไปด้วย ซึ่งในจุดนี้เองเมื่อมีการคำนวณต่างๆ เกิดขึ้น เช่น การสร้าง QEPs ในขั้นตอนของการ Initial Population เมื่อ Join Tree มีขนาดใหญ่มากขึ้น Process ที่ทำงาน ณ ขณะนั้นจะต้องใช้ทรัพยากร และพลังสำหรับการคำนวณมากขึ้นเพื่อทำการสร้าง QEPs ให้ครบตามจำนวนที่กำหนด และตามเงื่อนไขให้เพื่อให้ได้ QEPs ที่มีความหลากหลาย และหากมี Process หลายๆ Process เข้ามาทำงานพร้อมๆ กันเป็นจำนวนมาก ก็จะทำให้มีการแย่งใช้งานทรัพยากรที่จำเป็นสำหรับการคำนวณ จึงทำให้เวลาที่ใช้สำหรับการคำนวณเพิ่มขึ้นตามจำนวนของ Process ที่เพิ่มขึ้นด้วยเช่นกัน

แต่เมื่อมีการนำเอาการคำนวณแบบ Parallel computing เข้ามาช่วยในการคำนวณ ทำให้ภาระในการสร้าง QEPs ของแต่ละ Process ลดลงเนื่องจากการที่ Query Optimizer ที่ใช้การค้นหาแบบ Island-based PGA จะมีการแลกเปลี่ยน QEPs ระหว่างกัน ในขั้นตอนของการ Initial Population จนครบจำนวนที่กำหนดของแต่ละ Process ทำให้ขั้นตอน Initial Population เสร็จสิ้นได้อย่างรวดเร็ว และส่งผลให้เป็นการประหยัดการใช้ทรัพยากรต่างๆ ที่จำเป็นต่อการคำนวณ ในกรณีที่มี Process สำหรับกระบวนการ Query Optimization เข้ามาทำงานจำนวนมากๆ พร้อมกัน และทำให้ประสิทธิภาพการทำงานของแต่ละ Process ลดลงไปไม่มากนัก

การศึกษาปัจจัยต่างๆ ที่มีความเหมาะสมต่อการแก้ปัญหาDistributed Query Optimization ด้วยวิธีการคำนวณแบบขนาน

สำหรับในหัวข้อการศึกษานี้ผู้วิจัยต้องการที่จะทำการศึกษาเพื่อที่จะต้องการบ่งชี้ว่ามีปัจจัย และ Parameter ตัวใดบ้างที่ส่งผลต่อการแก้ปัญหา Distributed Query Optimization โดยใช้การคำนวณแบบขนานด้วยวิธีการแบบ Island-based Parallel Genetic algorithm ซึ่งปัจจัยที่ผู้วิจัยให้ความสนใจมีดังต่อไปนี้ คือ

1. จำนวนหน่วยประมวลผล
2. จำนวน Process ย่อยที่ถูกแบ่งออกในการทำงานแต่ละครั้ง

เพื่อให้สามารถอธิบายว่าปัจจัยที่ผู้วิจัยให้ความสนใจศึกษานั้นมีผลต่อการ แก้ปัญหา Distributed Query Optimization โดยใช้การคำนวณแบบขนานด้วยวิธีการแบบ Island-based Parallel Genetic algorithm อย่างไรผู้วิจัยจึงได้ทำการทดลอง โดยผู้วิจัยสามารถอธิบายการทดลองได้ดังนี้

การกำหนดปัจจัยสำหรับการทดลอง

การทดลองภายในงานวิจัยนี้ได้ทำการทดลองบนระบบ Computer Clusters ซึ่งประกอบไปด้วยเครื่อง Front-end 1 เครื่อง และเครื่องที่เป็น Compute Node 4 เครื่อง โดยทั้งหมดเป็นเครื่อง Celeron 2.4 GHz. Cache size 128 kb Ram 1.00 G ใช้ระบบปฏิบัติการ Linux Rock Clusters 4.2 สำหรับสร้างระบบ Clusters และใช้ LAM/MPI version 7.1.3 สำหรับการทำงานแบบ Parallel Computing

สมติฐานของการทดลอง

เนื่องจากเครื่องคอมพิวเตอร์ที่ผู้วิจัยได้ใช้สำหรับการทดลองมีเพียงแค่ 1 CPU ดังนั้นในการแบ่ง Process ไปดำเนินการในแต่ละ Node สำหรับแต่ละกระบวนการ Island-based PGA นั้นจะแบ่งเพียงแค่ 1 Process ต่อ 1 Node เท่านั้น

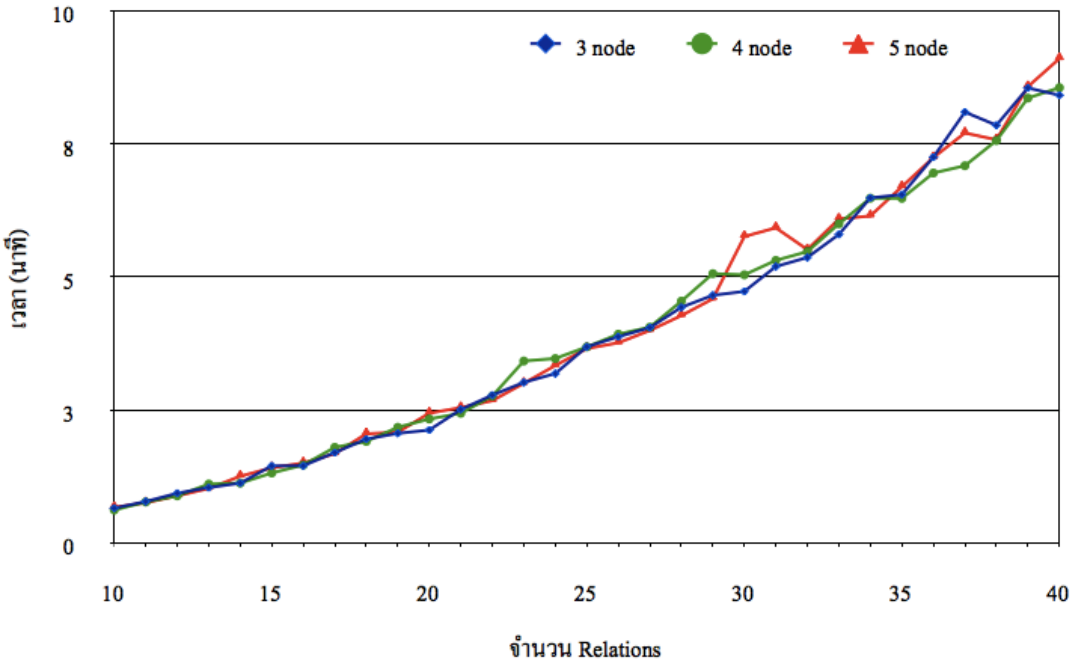
ผลการทดลอง

ผู้วิจัยได้ทำการทดลองดำเนินการกระบวนการ Island-based PGA โดยใช้ Parameters ที่มีผลต่อการทำงานของ Island-based PGA ดังตารางที่ 2 แต่สิ่งแต่แตกต่างออกไปคือผู้วิจัยจะค่อยๆ เพิ่มจำนวน Node ที่ใช้ดำเนินการมากขึ้นเรื่องโดยจะเพิ่มขึ้นจาก 3 Node ไปจนถึง 5 Node โดยผู้วิจัยสามารถแสดงผลของการทดลองได้ดังตารางที่ 5

ตารางที่ 5 เปรียบเทียบประสิทธิภาพด้านเวลาของการทำงานระหว่าง Island-based PGA ที่มีจำนวน Process ที่แตกต่างกัน

	เวลาเฉลี่ยของการทำงาน (นาที)		
จำนวน Relation	3 Node	4 Node	5 Node
10	0.6525	0.6164	0.6667
11	0.7769	0.7679	0.7537
12	0.9328	0.8837	0.8829
13	1.0387	1.1055	1.0216
14	1.1231	1.1233	1.2524
15	1.4466	1.3132	1.4094
16	1.4490	1.4626	1.5016
17	1.6958	1.7955	1.6756
18	1.9501	1.9066	2.0497
19	2.0577	2.1726	2.0774
20	2.1200	2.3279	2.4375
21	2.5068	2.4354	2.5414
22	2.7760	2.7582	2.6715
23	3.0157	3.4177	2.9976
24	3.1811	3.4644	3.3325
25	3.6841	3.6831	3.6510
26	3.8732	3.9216	3.7592
27	4.0441	4.0539	3.9909
28	4.4258	4.5419	4.2704
29	4.6529	5.0592	4.5855
30	4.7250	5.0372	5.7567
31	5.1932	5.3092	5.9200
32	5.3630	5.4730	5.5113
33	5.7958	5.9956	6.0850
34	6.4827	6.4778	6.1437
35	6.5446	6.4744	6.6926
36	7.2474	6.9507	7.2400
37	8.0950	7.0872	7.7026
38	7.8487	7.5600	7.5752
39	8.5487	8.3587	8.5729
40	8.4122	8.5544	9.1001

โดยสามารถนำเสนอผลที่ได้จากตารางที่ 5 ในรูปของกราฟความสัมพันธ์ดังภาพที่ 49 ดังนี้



ภาพที่ 49 เวลาที่ใช้ในการทำงานของ Island-based PGA ที่มีการแบ่งจำนวน Process ที่แตกต่างกัน

จากภาพที่ 49 ผู้วิจัยสามารถอภิปรายผลของการทดลองได้ดังต่อไปนี้ จากภาพที่ 49 จะเห็นได้ว่าผลลัพธ์ในด้านความเร็วที่ได้จากการทำงานของ Island-base PGA โดยการแบ่ง Process ออกเป็นจำนวน 3, 4, 5 Process ภาพใต้การทดลองที่ให้การทำงานเป็นแบบ 1 Node ต่อ 1 Process จะเห็นได้ว่าเวลาที่ออกมาค่อนข้างจะใกล้เคียงกันในภาพรวม

แต่สิ่งที่ผู้วิจัยได้ตั้งข้อสังเกตจากผลการทดลองที่ได้คือ จากกราฟจะเห็นได้ว่า รูปร่างของกราฟจะไม่เรียบซึ่งจะมีบางช่วงมีลักษณะกระโดด ซึ่งทางผู้วิจัยสามารถวิเคราะห์ได้ว่าเป็นผลอันเนื่องมากจากกระบวนการ Migration ที่ผู้วิจัยใช้ซึ่งมีรูปแบบการสื่อสารข้อมูลเป็นแบบ Ring Topology ซึ่งเป็นรูปแบบที่จะต้องมีการหยุดรอกันในการรับส่งข้อมูล และหากมี Process ใดหยุดชะงักจะทำให้ Process ทั้งหมดหยุดรอตามไปด้วย ซึ่งจะเห็นได้จากกราฟในบางช่วงที่มีลักษณะไม่เรียบ ซึ่งเป็นสาเหตุมาจากบาง Process อาจถูก Interrupt จากงานอื่นๆ ที่จำเป็นต้องใช้

ทรัพยากรร่วมกัน ซึ่งจุดนี้นับว่าเป็นจุดที่สามารถบ่งชี้ข้อเสียข้อวิธีการ Migration ที่ผู้วิจัยใช้อยู่ได้เป็นอย่างดี

ดังนั้นจากการทดลองผู้วิจัยสามารถทำการสรุปผลในส่วนนี้ได้ว่าในการทดลองตามปัจจัยของ Island-based PGA คือ ใช้ Migration Rate 4, การใช้ Initialization Population แบบ Parallel และจำนวน Sub-Population จำนวน 512 และการตั้งค่าตามตารางที่ 2 จะเห็นได้ว่าการเพิ่มหรือลดจำนวน Process และ Node ในช่วง 3 – 5 Node นั้นไม่ส่งผลอย่างชัดเจนต่อเวลาการทำงานโดยเฉลี่ยของ Island-based PGA ในการแก้ปัญหาเรื่อง Distributed Query Optimization

และการแบ่ง Process ออกเป็นจำนวนมากด้วยการเพิ่ม Node อาจไม่ส่งผลดีต่อการทำงานแบบ Parallel สำหรับงานวิจัยนี้ เนื่องจากในแต่ละกระบวนการของ Island-based PGA ที่ใช้วิธีการรับส่งข้อมูลแบบ Ring Topology หากมีการทำงานร่วมกันระหว่าง Process เป็นจำนวนมากก็ย่อมมีโอกาสเป็นไปได้ง่ายที่จะมี Process ใด Process หนึ่งถูก Interrupt และจะทำให้ Process อื่นๆ หยุดชะงักตามไปด้วยเป็นเรื่องง่ายขึ้น

บทที่ 6

สรุป วิเคราะห์ผล และข้อเสนอแนะ

สรุปและวิเคราะห์ผลการทดลอง

เพื่อให้การสรุป และวิเคราะห์ผลเป็นไปอย่างสอดคล้องกับวัตถุประสงค์การศึกษา ผู้วิจัยจึงทำการสรุป และวิเคราะห์ผลตามหัวข้อวัตถุประสงค์การศึกษาดังนี้

เพื่อสร้าง Distributed Query Optimizer ต้นแบบที่ใช้วิธีค้นหาแบบ Island Based Parallel Genetic Algorithm และเพื่อพัฒนากระบวนการ Query Optimization ที่สามารถรองรับการทำงานแบบ Parallel Computing

จากวัตถุประสงค์ในการวิจัยข้อนี้ผู้วิจัยสามารถสรุปได้ว่า ในการดำเนินงานวิจัยชิ้นนี้ผู้วิจัยสามารถที่จะทำการสร้าง Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ Island Based Parallel Genetic Algorithm และรองรับการทำงานแบบขนานโดยใช้มาตรฐาน MPI version 2 ขึ้นมาได้โดยการจำลองแบบ โดยสิ่งที่จำเป็นที่ต้องคำถึงสำหรับการพัฒนากระบวนการ Distributed Query Optimization สำหรับระบบ Distributed Database คือประสิทธิภาพในการประมวลผลคำสั่งสอบถามข้อมูล (Query Processing) แบบ Real Time ซึ่งจะต้องสามารถตอบสนองการทำงานของผู้ใช้งานได้อย่างรวดเร็วที่สุด และเวลาการทำงานเพื่อหาผลลัพธ์ของคำสั่งสอบถามข้อมูล จะต้องอยู่ในขอบเขตที่ผู้ใช้งานสามารถที่จะรอคอยผลลัพธ์ของการทำงานได้

กระบวนการ Distributed Query Optimization เป็นกระบวนการที่ใช้สำหรับค้นหาวิธีการประมวลผลคำสั่งสอบถามข้อมูล ที่มีความเหมาะสมที่สุด ซึ่งจะถูกนำไปใช้สำหรับประมวลผลคำสั่งสอบถามข้อมูลเพื่อให้ได้คำตอบตามที่ผู้ใช้งานต้องการ โดยกระบวนการ Distributed Query Optimization จะต้องมีการทำงานแบบ “คำนวณแบบครั้งต่อครั้ง” สำหรับแต่ละคำสั่งสอบถามข้อมูลที่เข้ามา โดยไม่สามารถที่จะเก็บคำตอบของกระบวนการ Distributed Query Optimization จาก คำสั่งสอบถามข้อมูล ก่อนหน้า มาใช้เป็นคำตอบสำหรับคำสั่งสอบถามข้อมูลต่อไปได้ เนื่องจาก

กระบวนการ Distributed Query Optimization นั้นมีปัจจัยแวดล้อมหลายอย่าง ซึ่งสามารถแปรผันได้ตลอดเวลา เช่น อัตราการขนส่งข้อมูลผ่านระบบ Network (Data Transfer Rate) จำนวน Cardinality ของแต่ละ Relations ที่อาจมีการเปลี่ยนแปลงได้ตลอดจากการใช้งานของผู้ใช้งานอื่นๆ

ดังนั้นสำหรับงานวิจัยนี้ ผู้วิจัยจึงได้มุ่งหวังที่จะทำการพัฒนากระบวนการ Distributed Query Optimizer ที่สามารถทำงานได้อย่างรวดเร็ว และตอบสนองต่อการทำงานของผู้ใช้งานได้อย่างดีที่สุด โดยการนำวิธีการค้นหาที่ใช้การคำนวณแบบขนาน คือ Island-based Parallel Genetic Algorithm มาเป็นวิธีที่ใช้สำหรับค้นหาวิธีการในการประมวลผลข้อมูลที่มีความเหมาะสมที่สุด ในกระบวนการ Distributed Query Optimizer โดยการวัดประสิทธิภาพที่ได้จากการพัฒนานั้น ผู้วิจัยได้ทำการทดลองเพื่อบ่งชี้ประสิทธิภาพในด้าน คือด้านความเร็วในการทำงาน และคุณภาพผลลัพธ์ที่ได้จากการทำงาน

เพื่อบ่งชี้ประสิทธิภาพของ Island Based Parallel Genetic Algorithm ว่าสามารถแก้ปัญหาเรื่อง Distributed Query Optimization of Large Join ได้เท่าใด และอย่างไร

ถึงแม้เป้าหมายหลักของงานวิจัยนี้ต้องการพิสูจน์ว่าการนำวิธีการค้นหาแบบขนานคือ Island-based PGA สามารถแก้ปัญหาเรื่อง Distributed Query Optimization ในกรณีที่เกิดการ Join จำนวนมากได้อย่างมีประสิทธิภาพหรือไม่ ซึ่งเมื่อพิจารณากระบวนการทำงานของ Island-based PGA จะเห็นว่าที่จริงแล้ววิธีการค้นหาแบบ Island-based PGA เป็นวิธีการค้นหาที่มีกระบวนการ GA แบบธรรมดาทำงานพร้อมๆ กันและมีการแลกเปลี่ยนข้อมูลที่มีคุณภาพระหว่างกัน ดังนั้นเพื่อเป็นการพัฒนา Query Optimizer ที่ใช้วิธีการค้นหาแบบ Island-based PGA ให้มีประสิทธิภาพ ผู้วิจัยจึงได้เริ่มพัฒนากระบวนการ GA แบบธรรมดาให้มีประสิทธิภาพก่อน

สำหรับการที่จะตัดสินได้ว่า Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ GA นั้นมีประสิทธิภาพหรือไม่ ผู้วิจัยได้เลือกที่จะวัดประสิทธิภาพโดยการเปรียบเทียบการทำงานกับ Distributed Query Optimizer แบบที่ใช้วิธีการค้นหาแบบ SA ซึ่งสำหรับวิธีการค้นหาแบบ SA นั้นจากงานวิจัยที่ผ่านมาที่เกี่ยวข้องกับการแก้ปัญหาเรื่อง Query Optimization นั้นได้มีการพิสูจน์และสรุปผลแล้วว่ามีประสิทธิภาพในการแก้ปัญหานี้ได้จริงสำหรับระบบ Centralize Database ซึ่งจากการทดลองเปรียบเทียบประสิทธิภาพในงานวิจัยนี้ระหว่าง Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ

GA และ SA ผู้วิจัยสามารถสรุปได้ว่า Distributed Query Optimizer แบบที่ใช้ GA ให้ประสิทธิภาพการทำงานที่ดีกว่าแบบที่ใช้ SA

เมื่อได้ข้อสรุปในเบื้องต้นว่า Distributed Query Optimizer แบบที่ใช้ GA มีประสิทธิภาพในการทำงานในกรณีที่เกิดการ Join ของ Relations จำนวนมากได้จริง ผู้วิจัยจึงได้ทำการพัฒนาต่อยอดจากวิธีการ GA แบบธรรมดาให้กลายเป็นวิธีการค้นหาแบบ Island-based PGA ซึ่งรองรับการทำงานแบบขนานและทำการเปรียบเทียบประสิทธิภาพของ Distributed Query Optimizer แบบที่ใช้ GA และแบบที่ใช้ Island-based PGA จากการทดลองโดยการเปรียบเทียบการทำงานระหว่าง Distributed Query Optimizer แบบที่ใช้วิธีการค้นหาแบบ GA และ Island-based PGA ผู้วิจัยสามารถสรุปได้ว่า Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ Island-based Parallel Genetic Algorithm นั้นสามารถทำงานรวดเร็วกว่าแบบที่ใช้ Genetic Algorithm ธรรมดาภายใต้เงื่อนในที่กำหนดขึ้นในงานวิจัยนี้คือ การใช้ Join Model แบบ Linear Join Model และวิธีการคิดค่า Cost หรือ Cost Function ที่ไม่ได้ขึ้นอยู่กับวิธีการ หรือ Join Algorithm แบบใดแบบหนึ่ง

และสำหรับข้อสรุปในเรื่องของคุณภาพของผลลัพธ์นั้น Distributed Query Optimizer ที่ใช้วิธีการค้นหาแบบ Island-based Parallel Genetic Algorithm ก็มีแนวโน้มที่จะให้คุณภาพของผลลัพธ์ที่ดีกว่าแบบที่ใช้ Genetic Algorithm และในด้านความคงทนต่อการทำงานสามารถสรุปได้ว่า Island-based Parallel Genetic Algorithm สามารถรองรับ Input Query พร้อมๆ กันจำนวนมากได้ดีกว่าแบบที่ใช้ GA

เพื่อศึกษาถึงปัจจัยที่มีผลต่อประสิทธิภาพการทำงานของ Island Based Parallel Genetic Algorithm ภายใต้แบบจำลองของ Distributed Query Optimizer ที่ได้สร้างขึ้น

ในระหว่างการทดลองผู้วิจัยสังเกตพบว่าสำหรับการทำงานของ GA ในขั้นตอน Initial Population นั้นเมื่อขนาดของ QEP ที่ต้องสร้างมีขนาดใหญ่ขึ้น หรือกล่าวได้ว่าเมื่อขนาดของการ Join มีจำนวนของ Relation มากขึ้น การสร้าง QEP ในขั้นตอน Initial Population ก็จะเป็นไปได้ล่าช้าขึ้นด้วย ตามจำนวนของจำนวน Relation ที่เพิ่มขึ้น

ดังนั้นเพื่อเป็นการแก้ไขปัญหาที่พบ และเพื่อเป็นการพัฒนาประสิทธิภาพให้ดียิ่งขึ้น ในขั้นตอนของการพัฒนา Distributed Query Optimizer แบบที่ใช้วิธีการค้นหาแบบ Island-based PGA ผู้วิจัยจึงได้นำการทำงานแบบ Parallel Computing เข้ามาประยุกต์ให้เกิดประโยชน์มากยิ่งขึ้น โดยใน

ขั้นตอนของการ Initial Population ซึ่งจะต้องมีการสร้าง QEP กลุ่มแรกตามจำนวนที่กำหนด สำหรับใช้เป็นกลุ่มประชากรต้นแบบ ผู้วิจัยได้ออกแบบให้มีการแลกเปลี่ยนกลุ่มประชากรต้นแบบระหว่าง Process ทำให้แต่ละ Process ทำการสร้าง QEP ขึ้นมาแค่บางส่วน แล้วทำการแลกเปลี่ยน QEP กับ Process อื่นๆ และ Process ในกระบวนการที่ดำเนินการอยู่ จะทำการรับ QEP จาก Process รอบข้างอื่นๆ เข้ามาเพื่อใช้สำหรับเป็นประชากรต้นแบบจนครบจำนวนที่กำหนด ซึ่งวิธีนี้จะทำให้แต่ละ Process ไม่ต้องเสียเวลาในการสร้างกลุ่มประชากรต้นแบบมากนัก และทำให้กระบวนการต่างๆ สามารถดำเนินไปได้อย่างรวดเร็วมากยิ่งขึ้น ซึ่งเป็นการเพิ่มประสิทธิการทำงานของ Distributed Query Optimizer แบบที่ใช้การค้นหาแบบ Island-based PGA ให้สามารถทำงานได้อย่างรวดเร็วยิ่งขึ้น

ข้อเสนอแนะ

จากงานวิจัยชิ้นนี้ผู้วิจัยได้นำเสนอการแก้ปัญหาเรื่อง Distributed Query Optimizer โดยใช้วิธีการคำนวณแบบขนานเข้ามาช่วย ซึ่งเทคนิคที่ผู้วิจัยใช้สำหรับการคำนวณแบบขนานคือมาตรฐาน MPI 2 ซึ่งเป็นมาตรฐานที่ใช้บนสถาปัตยกรรมแบบ Share Nothing ซึ่งจะต้องใช้เครื่อง Computer หลายเครื่องเข้ามาช่วยในการคำนวณแบบขนาน แต่ในปัจจุบัน และในอนาคตมีแนวโน้มว่าภายใน Computer แต่ละเครื่องจะมีจำนวนของ CPU มากขึ้นซึ่งจะสามารถทำการคำนวณแบบขนานได้ง่ายขึ้น ผสมกับเทคโนโลยีที่ทำให้เกิดมาตรฐานในการคำนวณแบบขนานบนสถาปัตยกรรมแบบ Share Memory เช่น OpenMP (OpenMP. (n.d.). Retrieved from OpenMP: http://openmp.org) เป็นต้น ดังนั้นผู้วิจัยจึงได้มีความคิดว่า เมื่องานวิจัยชิ้นนี้บ่งชี้ว่า วิธีการคำนวณแบบขนานบนสถาปัตยกรรมแบบ Share Nothing สามารถที่จะช่วยแก้ปัญหาในเรื่อง Distributed Query Optimization ได้เป็นอย่างดี ในอนาคตอาจมีการนำเทคโนโลยีการคำนวณแบบขนานบนสถาปัตยกรรมแบบ Share Memory มาทดลองแก้ปัญหาในเรื่องเดียวกันนี้ ซึ่งอาจจะสามารถลดค่าใช้จ่ายต่างๆ เกี่ยวกับเรื่อง Hard Ware ได้มากขึ้นและให้ประสิทธิภาพการทำงานที่ดียิ่งขึ้นกว่าเดิม

บรรณานุกรม

ภาษาไทย

กระบวนการสอบถามข้อมูล (Query Processing) [ออนไลน์]. เข้าถึงเมื่อ 10 พฤศจิกายน 2552. เข้าถึงได้จาก http://www.cs.hcu.ac.th/e-learning/distributed/lesson5/lesson_5_2.htm

บุญเสริม กิจศิริกุล. “ปัญญาประดิษฐ์.” เอกสารคำสอนวิชา 2110654 ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย, มีนาคม 2548. (อัดสำเนา)

ภาษาต่างประเทศ

Bennett, K., Michael C. Ferris, and Yannis E. Ioannidis. “A genetic algorithm for database query optimization.” In Proceedings of the Fourth International Conference on Genetic Algorithms, 400-407. Edited by Rick Belew and Lashon Booker. CA : Morgan Kaufman, 1991.

Dong, H., and Yiwen Liang. “Genetic Algorithms for Large Join Query Optimization.” In Proceedings of the 9th annual conference on Genetic and evolutionary computation, 1211-1218. Edited by ACM. London : ACM, 2007.

Guangfa Lu, and Shawki Areibi. “An Island-Based GA Implementation for VLSI Standard-Cell Placement.” In GECCO , 1138-1150. Edited by Springer-Verlag. Berlin Heidelberg : GECCO, 2004.

Ioannidis, Y. E., and Younkyung Cha Kang. “Randomized algorithms for optimizing large join queries.” In Proceedings of the 1990 ACM SIGMOD international conference on Management of data, 312-321. Edited by ACM Press. New York : ACM, 1990.

LAM/MPI Parallel Computing [Online]. Accessed 25 November 2009. Available from http://www.lam-mpi.org.

Matysiak, M., “Efficient Optimization of Large Join Queries Using Tabu Search.” Information Sciences 83, 1-2 (March 1995) : 77-88.

OpenMP [Online]. Accessed 10 November 2009. Available from http://openmp.org.

Oracle® Database Performance Tuning Guide [Online]. Accessed 15 November 2009. Available http://download east.oracle.com/docs/cd/B14117_01/server.101/b10752/optimops.htm

Özsu M. Tamer., and Patrick Valduriez. Principles of Distributed Database Systems. 2nd ed. New York : Prentice-Hall, 1991.

Pongpinigpinyo, S. "Distributed Query Qptimisation Using Two Stage Simulated Annealing." Ph.D. Dissertation, University of Tasmania, 1996.

Poli, R., William B. Langdon, and Nicholas F. McPhee. A Field Guild to Genetic Programming [Online]. Accessed 15 November 2009. Available http://cswww.essex.ac.uk/staff/ rpoli/TinyGP/

Rho, S., and Salvatore T. March. "Optimizing distributed join queries: A genetic algorithm approach." In Annals of Operations Research, 199-228. Edited by Springer. Netherlands : SpringerLink, 2004.

Stillger, M., and Myra Spiliopoulou. "Genetic Programing in Database Query Optimization." In Genetic Programing in Database Query Optimization, 388-393. Edited by German Research Council. Berlin Heidelberg : MIT Press, 1996.

Stillger, M., Myra Spiliopoulou, and Johann-christoph Freyta. "Parallel Query Optimization: Exploiting Bushy and Pipeline Parallelism with Genetic Programms." In Scientific Commons, 1-33. Edited by German Research Council. Berlin Heidelberg : MIT Press, 1996.

Swami, A. "Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques." In Proceedings of the 1989 ACM SIGMOD international conference on Management of data,367 – 376. Edited by ACM. CA : ACM, 1989.

Swami, A., and Anoop Gupta. "Opimization of Large Join Queries." ACM SIGMOD Record 17, 3 (June 1988) : 8 – 17.

ประวัติผู้วิจัย

ชื่อ – สกุล นายพิศาล สุขขี

ที่อยู่ 33 หมู่ที่ 10 ตำบลจอมบึง อำเภอจอมบึง จังหวัดราชบุรี

ประวัติการศึกษา

พ.ศ. 2546 สำเร็จการศึกษาระดับปริญญาวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยศิลปากร

พ.ศ. 2548 ศึกษาต่อระดับปริญญามหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยศิลปากร

ประวัติการทำงาน

พ.ศ. 2547-2548 นักวิชาการคอมพิวเตอร์ ศูนย์คอมพิวเตอร์ มหาวิทยาลัยศิลปากร วิทยาเขตพระราชวังสนามจันทร์ นครปฐม