การออกแบบระบบจัดเก็บข้อมูลและคิวรีข้อมูลรีซอร์สดิสคริปชันเฟรมเวิร์ค ขนาดใหญ่โดยใช้หน่วยประมวลผลกราฟิก

The Design of Triple Store and Query Processing on GPU for Large Scale Resource Description Framework Data

พิสิษฐ์ มรรคไพสิฐ[1,*] และ จันทนา จันทราพรชัย[1]

[1]ภาควิชาวิศวกรรมคอมพิวเตอร์, คณะวิศวกรรมศาสตร์, มหาวิทยาลัยเกษตรศาสตร์

ลาดยาว จตุจักร กรุงเทพมหานคร 10900

Pisit Makpaisit and Chantana Chantrapornchai

[1]Department of Computer Engineering, Faculty of Engineering,

Kasetsart University, Lay Yao, Chatuchak, Bangkok, 10900, Thailand

***Corresponding Author E-mail:** pisit.mak@ku.th

Received: Mar 20, 2023; **Revised:** Aug 08, 2023; **Accepted:** Aug 23, 2023

บทคัดย่อ

ข้อมูล Resource Description Framework (RDF) เป็นมาตรฐานของการแลกเปลี่ยนข้อมูลระหว่างเว็บ มีแนวโน้มที่จะขยายขนาดขึ้นอย่างรวดเร็วในอัตราเร่งที่สูงขึ้น เพื่อพัฒนาระบบการคิวรีข้อมูล RDF ที่สามารถค้นคืนข้อมูลได้อย่างรวดเร็วบนข้อมูลขนาดใหญ่ งานวิจัยนี้ได้นำเสนอระบบต้นแบบในการจัดเก็บและค้นคืนข้อมูล RDF ด้วยการใช้ประโยชน์จากหน่วยประมวลผลกราฟิก (GPU) โดยนำเสนอรูปแบบการจัดเก็บข้อมูลที่เหมาะสมกับข้อมูลชนิด RDF และประมวลผลบน GPU การออกแบบดรรชนี การสร้างระบบสำหรับคิวรีข้อมูลบน GPU รวมไปถึงเทคนิคการเพิ่มประสิทธิภาพ ได้แก่ การกรองข้อมูล และการกำหนดค่า ID ด้วยเทคนิคการหาความใกล้เคียงของข้อมูล โดยการแปลงจากข้อความเป็นเวกเตอร์และใช้เทคนิคการลดมิติข้อมูลเพื่อทำให้สามารถแปลงข้อมูลกลับมาเป็นตัวเลข ผลการทดลองแสดงให้เห็นว่าระบบที่ออกแบบและพัฒนาขึ้น ใช้พื้นที่เก็บข้อมูลเพียงประมาณ 1 ใน 6 ของข้อมูลดิบ และสามารถช่วยลดเวลาการคิวรีข้อมูลได้ โดยมีสปีดอัพจากเวลาที่ใช้คิวรีแบบดั้งเดิมสูงสุดที่ 29.57 เมื่อเทียบกับ RDF-3X และค่าสปีดอัพสูงสุด 45.23 เมื่อเทียบกับวิธีการจัดเก็บแบบกราฟ gStore

คำสำคัญ: อาร์ดีเอฟ, การประมวลคิวรี, สปาร์เคิล, หน่วยประมวลผลกราฟิก

Abstract

The Resource Description Framework (RDF) is commonly used as a standard for data interchange on the web. Due to the current big data era, its size is prone to increase drastically. In order to speed up the large RDF data query, we propose a novel RDF data representation along with the RDF query algorithm utilizing GPU processing. We also present the representation that is suitable for RDF data and GPU processing, the indexing approach, the querying process in the GPU and other techniques that increase the efficiency such as pre-upload filtering, ID assignment by using the term similarity of term vector, and dimensional reduction to transform back to term ID. The experiments show that the developed framework

utilizes the storage only 1/6 of the original one and can reduce the querying time. The speedup obtained can be up to 29.57 when compared with the RDF-3X system and 45.23 when compared to using gStore, a graph data store.

Keywords: RDF, Query Processing, SPARQL, GPU

1. บทนำ

Resource Description Framework หรือRDF เป็นมาตรฐานการแลกเปลี่ยนข้อมูลบนเว็บเชิงความหมาย (Semantic Web) ที่ถูกเสนอขึ้นโดยองค์กร W3C เพื่อให้เป็นมาตรฐานกลางในการจัดเก็บข้อมูลแบบมีโครงสร้าง และสามารถสืบค้นเชิงความหมายได้ผ่านทางมนุษย์หรือระบบอัตโนมัติ RDF ถูกใช้อยู่ในหลากหลายงาน เช่น ในการอธิบายอนุกรมวิธาน (Taxonomy) ของสัตว์ สารานุกรมของสภาพแวดล้อมโลก การแสดงข้อมูลสิทธิบัตรของสหรัฐอเมริกา หรือออนโทโลยี (Ontology) เป็นต้น การจัดเก็บข้อมูล RDF มีหลากหลายรูปแบบ รูปแบบที่เป็นที่นิยมในปัจจุบันคือรูปแบบที่เรียกว่า N-Triples ซึ่งจะเรียกข้อมูลแต่ละข้อมูลว่าทริปเปิล (Triple) และมีรูปแบบการแสดงคือ subject predicate object . ในหนึ่งไฟล์ข้อมูลจะประกอบด้วยทริปเปิลจำนวนหลายทริปเปิล โดยสามารถค้นหาข้อมูลที่ต้องการได้ด้วยภาษาสืบค้นข้อมูล RDF ที่เรียกว่า SPARQL ซึ่งย่อมาจาก Simple Protocol and RDF Query Language **รูปที่ 1** เป็นรูปตัวอย่างข้อมูล RDF ที่มีจำนวน 3 ทริปเปิล โดยอธิบายถึงข้อมูลของเอนทิตี (Entity) หนึ่งซึ่งเป็นบุคคล และมีชื่อจริงว่า Natee และเกิดเมื่อปี 1997 ในส่วน**รูปที่ 2** จะเป็นตัวอย่างคิวรี SPARQL ที่ค้นหารายการข้อมูลบุคคล โดยผลลัพธ์ที่ได้จะเป็นรายการของชื่อและโฮมเพจ

```
<http://example.com/simple#person1>
  <http://example.com/simple#first-name> "Natee" .
<http://example.com/simple#person1>
  <http://example.com/simple#birth-year> "1997" .
<http://example.com/simple#person1>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://example.com/simple#person> .
```

รูปที่ 1 ตัวอย่างข้อมูล RDF ในรูปแบบ N-Triples

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?homepage
WHERE {
   ?person foaf:name ?name .
   ?person foaf:homepage ?homepage .
}
```

รูปที่ 2 ตัวอย่างคิวรี SPARQL

มาตรฐานการสืบค้นข้อมูล SPARQL ในปัจจุบันได้มาถึงเวอร์ชัน 1.2 ซึ่งรวมไปถึงฟังก์ชันการทำงานขั้นสูงหลายฟังก์ชัน เช่น การอัปเดตข้อมูล ฟังก์ชันการรวม (Aggregate Function) การค้นหาแบบมีเงื่อนไข การจัดการข้อความ ฯลฯ เพื่อเป็นการทดสอบประสิทธิภาพการทำงานด้วย GPU เบื้องต้น งานวิจัยนี้จะจำกัดฟังก์ชันการทำงานเฉพาะคำสั่ง SELECT และ WHERE ในรูปแบบเบื้องต้นเท่านั้น ในคำสั่ง WHERE จะประกอบด้วย คำค้นหาย่อยเรียกว่าทริปเปิลแพตเทิร์น (Triple Pattern) ซึ่งมีรูปแบบคล้ายกับทริปเปิล แต่สามารถใช้ตัวแปร (Variable) ระบุได้ เพื่อเป็นการบอกให้ค้นหาข้อมูลที่ตรงกับรูปแบบดังกล่าว แต่ในส่วนของตัวแปรจะเป็นข้อมูลอะไรก็ได้ ตัวแปรจะขึ้นต้นด้วย ? เสมอ กลุ่มของทริปเปิลแพตเทิร์นทั้งหมดจะเรียกว่า Basic Graph Pattern (BGP) ซึ่งเราสามารถเปรียบเทียบการค้นหาข้อมูล RDF ได้เหมือนกับการหากราฟย่อย (Subgraph) ที่ตรงกับ BGP ที่กำหนดให้

ข้อมูล RDF มีขนาดใหญ่ขึ้นเรื่อย ๆ เป็นผลมาจากเทคโนโลยีการจัดเก็บข้อมูลที่มีศักยภาพสูงขึ้น การเติบโตของอินเทอร์เน็ต และความต้องการใช้งานข้อมูลเชิงความหมาย อันส่งผลต่อเนื่องมายังขนาดของไฟล์ RDF ที่ใหญ่ขึ้นอย่างมหาศาลและใช้เวลาในการคิวรีที่นานขึ้น มีงานวิจัยจำนวนมากที่พยายามเข้ามาแก้ปัญหาของคิวรีข้อมูล RDF ซึ่งมีทั้งรูปแบบของ การออกแบบวิธีจัดเก็บข้อมูลและการสืบค้นให้มีประสิทธิภาพมากขึ้น [1–3] การ

แบ่งข้อมูล RDF เป็นกลุ่มและทำการประมวลผลข้อมูลแบบขนาน [4],[5] การใช้คุณสมบัติหรือรูปร่างของข้อมูลช่วยในการค้นคืนข้อมูล [6],[7] การออกแบบและพัฒนาระบบโดยใช้ตัวเร่งฮาร์ดแวร์ (Hardware Accelerator) ในการลดเวลาคิวรี [8],[9] หรือสถาปัตยกรรมที่มีตัวประมวลผลหลายชนิด (Heterogenous Architecture) [10] และงานวิจัยที่ใช้เฟรมเวิร์คสำหรับระบบข้อมูลขนาดใหญ่ (Big Data) เป็นพื้นฐานของการสร้างระบบจัดเก็บและคิวรีข้อมูล [11],[12]

ในงานวิจัยที่ผ่านมาที่มีการใช้ GPU ในการเร่งประสิทธิภาพของการทำงานได้ผลลัพธ์ที่ดีในด้านของการลดเวลาการคิวรี แต่มักประสบปัญหาของเวลาในการทำงานที่เพิ่มขึ้นในส่วนของการส่งข้อมูลจากหน่วยความจำหลักไปยังหน่วยความจำบน GPU [8], [9], [11] ซึ่งหากสามารถลดเวลาการทำงานในส่วนนี้ได้ก็จะส่งผลให้การใช้งาน GPU เพื่อการค้นคืนข้อมูลทำได้รวดเร็วยิ่งขึ้น งานวิจัยนี้จึงมีเป้าหมายที่จะทดลองพัฒนาระบบจัดเก็บข้อมูล RDF ที่สามารถสืบค้นได้อย่างรวดเร็วผ่าน GPU ที่มีความสามารถในการประมวลผลข้อมูลได้จำนวนมากผ่านคอร์ประมวลผลจำนวนหลายพันคอร์ เพื่อออกแบบการเก็บข้อมูลและเทคนิคการค้นหาข้อมูล RDF บน GPU และการลดเวลาในการส่งข้อมูลระหว่างหน่วยความจำหลักและ GPU ที่จะช่วยให้สามารถลดเวลาการค้นคืนข้อมูลด้วย SPARQL

2. วัสดุ อุปกรณ์และวิธีการวิจัย

2.1 รูปแบบการจัดเก็บข้อมูล (Data Representation)

รูปแบบการจัดเก็บข้อมูลเป็นส่วนเริ่มต้นที่สำคัญที่สุดในการจัดเก็บข้อมูล RDF และการสืบค้นข้อมูล รูปแบบการจัดเก็บที่ดีนอกจากจะช่วยประหยัดพื้นที่ในการจัดเก็บแล้ว ยังสามารถช่วยให้สืบค้นข้อมูลได้เร็วขึ้น โดยเฉพาะอย่างยิ่งบน GPU ที่มีขนาดของหน่วยความจำจำกัด ขนาดของการจัดเก็บข้อมูล และข้อมูลที่จะต้องถูกย้ายไปมาระหว่างหน่วยความจำหลักและหน่วยความจำบน GPU จะยิ่งมีผลต่อประสิทธิภาพมากขึ้นไปอีก

งานวิจัยที่เกี่ยวข้องกับรูปแบบการจัดเก็บข้อมูลของ RDF มีหลากหลายรูปแบบ ซึ่งเหมาะสมกับสถาปัตยกรรมและการใช้งานที่ต่างกันออกไป เช่น การจัดเก็บในรูปแบบของกราฟ (ตัวอย่างเช่น ระบบ gStore [13],[14] และ AMBER [15]) การจัดเก็บในรูปแบบเมทริกซ์หรือเทนเซอร์ (ตัวอย่างเช่น Triple-ID Q [8] และ MAGiQ [9]) การจัดเก็บในรูปการแบ่งแนวตั้ง (Vertical Partitioning) (ตัวอย่างเช่น Hexastore[16] และ S2RDF[17]) หรือในรูปแบบการจัดเก็บแบบเป็นรายการของทริปเปิล

งานวิจัยนี้จะใช้การเก็บข้อมูลของทริปเปิลในรูปแบบของตัวเลข ID แทนการจัดเก็บแบบเป็นข้อความ (เรียกว่า Triple-ID [8]) โดยจะทำการสร้างฟังก์ชันสำหรับแปลงข้อความเป็น ID ที่ต้องการ ดังนั้นทริปเปิล 1 ทริปเปิลจะถูกเปลี่ยนเป็นตัวเลขหรือ ID 3 ตัว โครงสร้างการจัดเก็บข้อมูลทริปเปิลทั้งหมดจะเป็นรูปแบบการจัดเก็บแบบคอลัมน์ที่มีดรรชนี (Indexed Column-based) กล่าวคือจะไม่ได้เก็บข้อมูลทริปเปิลเดียวกันติดกันบนหน่วยความจำแบบอ้างอิงทีละทริปเปิล แต่จะเก็บเป็นอาร์เรย์ของแต่ละคอลัมน์ของข้อมูลทั้งหมด โดยจะมีการเรียงลำดับข้อมูลตามลำดับคอลัมน์ที่เก็บ และในส่วนของคอลัมน์แรกที่เป็น predicate จะเก็บอยู่ในรูปแบบของ CSR (Compressed Sparse Row) เนื่องจากข้อมูล predicate มีจำนวนที่แตกต่างกันไม่มาก ถึงแม้จะเป็นข้อมูล RDF ขนาดใหญ่ก็จะยังมีข้อมูล predicate ที่น้อย ซึ่งการใช้รูปแบบ CSR เข้ามาเก็บจะช่วยลดจำนวนข้อมูลซ้ำซ้อนที่ต้องเก็บลงไปได้มาก

ดรรชนีมีประโยชน์ในการช่วยให้เข้าถึงเข้ามูลที่ต้องการได้รวดเร็วมากขึ้น ตัวอย่างเช่น สำหรับทริปเปิลแพตเทิร์น *?v0 14 18* ถ้าข้อมูลเก็บอยู่ในรูปแบบดรรชนีของ Predicate Object Subject (POS) จะสามารถสแกนดรรชนีและได้ชุดข้อมูลที่เป็นคำตอบของทริปเปิลแพตเทิร์นนี้ทันที การมีดรรชนีหลายรูปแบบจะช่วยให้สามารถได้ข้อมูลที่ต้องการได้เร็วขึ้น ไม่ว่าจะเป็นทริปเปิลแพตเทิร์นแบบใด งานวิจัยนี้ออกแบบระบบที่สามารถเลือกการสร้างดรรชนีได้ 2 รูปแบบคือ Full Permutation Index และ Predicate Index ซึ่งสำหรับ Full permutation Index จะประกอบด้วยดรรชนีทุกรูปแบบได้แก่ POS, PSO, OPS,

OSP, SPO และ SOP ส่วน Predicate Index จะมีเฉพาะ POS และ PSO ซึ่งมีข้อดีในแง่ของการใช้พื้นที่การจัดเก็บที่น้อยลง แต่ยังสามารถนำมาใช้งานได้จริงกับข้อมูลหลายชุด เนื่องจากโดยปกติแล้วจะไม่ใช้ predicate เป็นตัวแปรในคิวรี SPARQL

รูปที่ 3 ในด้านซ้ายจะเป็นข้อมูล RDF ที่ถูกแปลงให้อยู่ในรูปของ ID แล้ว โดยแสดงให้เห็นในรูปแบบที่มีการเรียงข้อมูลจาก P, S และ O ตามลำดับ ด้านขวามือจะเป็นข้อมูลที่ถูกเก็บในระบบจริงแบบที่ใช้ดรรชนี PSO ข้อมูลของ P ที่เป็นดรรชนีจะเก็บในรูปแบบ CSR ที่บันทึกเฉพาะข้อมูลที่ไม่ซ้ำกัน และใช้ Offset เป็นตัวระบุว่าข้อมูลในชั้นถัดไปเริ่มต้นที่จุดใดของอาร์เรย์ ข้อมูลในชั้นถัดไปจะมีข้อมูลของ S และ O แยกกัน

P	S	O
3	7	20
3	9	12
3	12	18
4	3	2
4	10	8
5	5	1
5	6	4

ข้อมูลในรูปแบบ Triple-ID

P	Offset
3	0
4	3
5	5

S	O
7	20
9	12
12	18
3	2
10	8
5	1
6	4

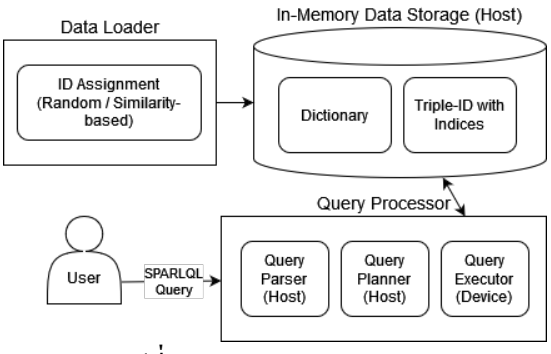
ข้อมูลในรูปแบบที่เก็บด้วย CSR ผ่านดรรชนี PSO

รูปที่ 3 ตัวอย่างการจัดเก็บข้อมูลด้วยดรรชนี PSO

กำหนดให้ข้อมูล RDF T มีจำนวนทริปเปิลเท่ากับ $|T|$ และแต่ละทริปเปิลประกอบขึ้นจาก ID ทั้งหมด 3 ตัว (ได้แก่ Subject Predicate และ Object) ให้ I เป็นเซ็ตของ Subject และ Object รวมกันและให้ P เป็นเซ็ตของ Predicate ดังนั้นข้อมูลที่ทำการเก็บจากดรรชนี POS และ PSO แต่ละชุดจะใช้ข้อมูลเท่ากับ $2|P| + 2|T|$ จำนวน ในการพัฒนาระบบจะเก็บแต่ละจำนวนด้วย Unsigned Integer ขนาด 4 ไบต์

2.2 สถาปัตยกรรมของระบบ

การทำงานของระบบจะแบ่งออกเป็น 3 ส่วน ได้แก่ ส่วนของการโหลดข้อมูล (Data Loader) ส่วนของการเก็บข้อมูล (Data Storage) และส่วนของการประมวลผลคิวรี (Query Processor) ดัง**รูปที่ 4**



รูปที่ 4 สถาปัตยกรรมของระบบ

การโหลดข้อมูลจะเป็นการอ่านไฟล์ N-Triple และนำเข้าระบบซึ่งจะมีขั้นตอนของการแปลงข้อความเป็น ID และการเรียงข้อมูลเพื่อสร้างดรรชนี ส่วนของการจัดเก็บข้อมูลจะทำการเก็บข้อมูลทั้งหมดบนหน่วยความจำหลักหรือรูปแบบ In-memory ซึ่งจะลดเวลาในการส่งข้อมูลในการประมวลจากฮาร์ดดิสก์ (แต่จะถูกจำกัดปริมาณข้อมูลสูงสุดที่เก็บได้) นอกจากนี้ยังจัดเก็บข้อมูลพจนานุกรมสำหรับแปลงข้อความเป็น ID และในการแปลงกลับ ส่วนของการประมวลผลคิวรีเป็นส่วนที่รับคิวรีจากผู้ใช้งานและส่งผลลัพธ์กลับไปยังผู้ใช้ ในส่วนนี้จะเริ่มต้นด้วยการแจงส่วน (Parsing) คิวรี และนำคิวรีย่อยทั้งหมดไปจัดเรียงลำดับการทำงาน ด้วย Query Execution Planner ซึ่งในที่นี้จะใช้เพียงแพลนเนอร์อย่างง่ายโดยการพยายามทำงานกับทริปเปิลแพทเทิร์นที่มีจำนวนข้อมูลน้อยการเป็นลำดับต้นๆ เมื่อได้แพลนการทำงานทั้งหมดแล้วระบบก็จะประมวลผลคิวรีด้วย GPU เพื่อให้ได้คำตอบกลับมา และแปลงคำตอบที่เป็น ID กลับเป็นข้อความก่อนส่งให้ผู้ใช้

2.3 การประมวลผลคิวรี

ในขั้นตอนการแจงส่วน จะใช้โอเพ่นซอร์ส Redland Rasqal เพื่อให้ได้คำค้นในรูปแบบที่มีโครงสร้าง และแบ่งข้อมูลในส่วนของตัวแปรที่ต้องการให้เป็นผลลัพธ์ (ใน SELECT) และทริปเปิลแพตเทิร์นทั้งหมดที่ต้องนำไปใช้ค้นหา หลังจากนั้นจึงใช้ฟังก์ชันที่สร้างขึ้นในช่วงของการสร้างข้อมูลแปลงทริปเปิลแพตเทิร์นทั้งหมดเป็น ID

งานวิจัยนี้ได้สร้างตัวดำเนินการ (Operator) ทั้งหมด 3 รูปแบบสำหรับการนำ BGP ไปคิวรีข้อมูล ตัวดำเนินการทั้งหมดจะถูกจัดเรียงโดยโปรแกรมเพื่อสร้างลำดับการทำงานที่ถูกต้องเพื่อให้ได้ผลลัพธ์ที่ต้องการ ขั้นตอนการทำงานของตัวดำเนินการทั้งหมดมีรายละเอียดดังนี้

2.3.1 การอัปโหลดข้อมูล

ขั้นตอนในการย้ายข้อมูลที่อยู่ในหน่วยความจำหลักขึ้นไปยังหน่วยความจำบน GPU ขั้นตอนนี้จะต้องเลือกดรรชนีที่ถูกต้องก่อน ใช้การสแกนข้อมูลผ่านดรรชนีเพื่อกำหนดช่วงของข้อมูลที่ต้องอัปโหลด ด้วยการสแกนค้นหาข้อมูลแบบทวิภาค (Binary Search) เช่น สำหรับทริปเปิลแพตเทิร์น *?v0 8 ?v1* สามารถใช้ดรรชนี POS หรือ PSO ในการอัปโหลด (ขึ้นกับการจัดลำดับตัวดำเนินงาน) ระบบจะทำการค้นหาหมายเลข *8* ที่คอลัมน์แรกด้วยการใช้การค้นหาข้อมูลแบบทวิภาค และหาช่วงของข้อมูลดังกล่าวเพื่อทำการอัปโหลด

2.3.2 การเชื่อมข้อมูล

ข้อมูลแบบคอลัมน์จะถูกนำมาเชื่อมข้อมูล สำหรับทริปเปิลแพตเทิร์นที่มีความเกี่ยวข้องกัน (มีตัวแปรเดียวกันอยู่ในทริปเปิลแพตเทิร์น) ซึ่งเป็นตัวดำเนินการแบบเดียวกับ Full Join ในระบบฐานข้อมูลเชิงสัมพันธ์ (Relational Database) สำหรับการเชื่อมข้อมูลที่จะใช้การประมวลผลบน GPU จะใช้อัลกอริทึม Sort-merge Join ในการเชื่อมข้อมูล เนื่องจากเป็นอัลกอริทึมสำหรับการเชื่อมข้อมูลที่มีประสิทธิภาพมากที่สุดบน GPU จากการทดลองในระบบที่ใช้ในการศึกษา

ผลลัพธ์ที่ได้จากการทำงานในส่วนของการอัปโหลดข้อมูลจะมีจำนวน 1 หรือ 2 คอลัมน์ซึ่งจะเท่ากับจำนวนตัวแปรในทริปเปิลแพตเทิร์นนั้น สมมติให้ทำการเชื่อมข้อมูล R_i และ R_j ที่มีข้อมูล c_1 และ c_2 คอลัมน์ตามลำดับ จำนวนคอลัมน์ของผลลัพธ์ที่ได้จะเท่ากับ $c_1 + c_2 –$ *จำนวนตัวแปรที่ซ้ำกันของ* R_i *และ* R_j

2.3.3 การสลับคอลัมน์

การเชื่อมข้อมูลด้วย Sort-merge Join มีข้อจำกัดในด้านของการที่ข้อมูลบนคอลัมน์หลักที่จะใช้ในการเชื่อมข้อมูลนั้นจะต้องมีการเรียงลำดับทั้งคู่จึงจะสามารถทำงานได้อย่างถูกต้อง แต่เนื่องจากคอลัมน์ของชุดข้อมูลที่เรียงกันอยู่ในแต่ละช่วงเวลาจะมีเพียงคอลัมน์เดียวเท่านั้น หากต้องการเชื่อมข้อมูล 2 ชุดโดยที่คอลัมน์ที่ใช้เชื่อมข้อมูลไม่ได้เรียงทั้งคู่ จะต้องทำการเรียงข้อมูลก่อน ซึ่งในที่นี้จะให้มีความหมายเหมือนกับสลับคอลัมน์ที่ต้องการเชื่อมข้อมูลมายังคอลัมน์แรก และทำการเรียงข้อมูล

การสลับคอลัมน์จะเกิดขึ้นกับข้อมูลที่เกิดการประมวลผลแล้วและอยู่บนหน่วยความจำ GPU ดังนั้นจึงสามารถดำเนินการเรียงแบบขนานผ่าน GPU ได้ทันที โดยความซับซ้อนของเวลาในการทำงานเป็น $O(Nlog(N)/p)$ เมื่อ N เป็นจำนวนของข้อมูล และ p เป็นจำนวนหน่วยประมวลผลที่สามารถทำงานพร้อมกันได้แบบขนาน ในขั้นตอนการเรียงนี้จะใช้ฟังก์ชัน sort ของไลบรารี Thrust

ตัวดำเนินการนี้ส่งผลต่อเวลาการทำงานในส่วนของเวลาที่เพิ่มขึ้น หากมีการสลับคอลัมน์หลายครั้ง และข้อมูลที่ต้องสลับมีขนาดใหญ่ อย่างไรก็ตามสามารถหลีกเลี่ยงการสลับคอลัมน์จำนวนมากได้ด้วยการจัดลำดับงานที่ดี โดยทั่วไปแล้วคิวรีข้อมูล RDF มักมีรูปแบบของคิวรีกราฟ (Query Graph) เป็นแบบ Star ซึ่งทำให้สามารถเชื่อมข้อมูลได้ทันทีโดยไม่ต้องทำการสลับคอลัมน์ก่อน แต่เมื่อใดที่เกิดเส้นทาง (Path) บนคิวรีกราฟที่มีความยาวมากกว่า 2 ก็จะเกิดการสลับคอลัมน์ขึ้นอย่างน้อย 1 ครั้ง

2.4 การกรองก่อนอัปโหลด (Pre-upload Filtering)

เพื่อเป็นการลดขนาดข้อมูลที่จะอัปโหลดไปยังหน่วยความจำ GPU เทคนิคที่ใช้ในขั้นตอนนี้คือการกรองข้อมูลที่เกินช่วงที่จะสามารถเชื่อมข้อมูลกันได้ออกไปก่อนที่จะทำการอัปโหลด ตัวอย่างเช่น หากข้อมูล R_i และ R_j ที่จะทำการเชื่อมข้อมูลด้วยตัวแปร *?v* โดยที่ค่าของข้อมูลในคอลัมน์ *?v* ของ *Ri* อยู่ในช่วง [*1150*, *90,000*] และค่าของของข้อมูลในคอลัมน์ *?v* ของ R_j อยู่ในช่วง [*4,400*, *111,555*] ข้อมูลที่เป็นไปได้ที่จะถูกเชื่อมข้อมูลจะอยู่ในช่วง [*4,400*, *90,000*] เท่านั้น ค่าที่เกินจากช่วงนี้จะไม่ถูกทำการเชื่อมข้อมูลอย่างแน่นอน จึงสามารถเลือกที่จะอัปโหลดเฉพาะส่วนดังกล่าวขึ้นไปยังหน่วยความจำ GPU และทำการเชื่อมข้อมูลเฉพาะในส่วนที่อัปโหลดขึ้นไป ซึ่งต่อไปจะเรียกเทคนิคนี้ว่าการกรองก่อนอัปโหลด

ในการทำงานของการกรองก่อนอัปโหลดจะทำการสร้างฟิลเตอร์ของแต่ละตัวแปรที่มีการเชื่อมข้อมูล ตัวแปรที่ถูกใช้แต่ไม่มีการเชื่อมข้อมูล จะไม่สร้างฟิลเตอร์ ฟิลเตอร์ของตัวแปรแต่ละตัวจะถูกกำหนดค่าเริ่มต้นให้อยู่ในช่วง $[0, \infty)$

จากนั้นทำการปรับค่าของช่วงดังกล่าวด้วยวิธีการเปรียบเทียบกับค่าของช่วงใหม่และกำหนดค่าด้วยวิธีการดังนี้

ให้ช่วงที่เป็นไปได้ของตัวแปร *?v* เป็น $B_v = [l_1, u_1]$ เมื่อระบบทำการตรวจสอบช่วงของข้อมูลของตัวแปร *?v* ในข้อมูลใหม่ที่พบและได้ช่วงเป็น $[l_2, u_2]$ จะทำการอัปเดตค่าของ B_v ใหม่เป็น $[max(l_1, l_2), min(u_1, u_2)]$

การอัปเดตฟิลเตอร์จะเกิดขึ้นในช่วงก่อนเริ่มการคิวรีข้อมูล ด้วยการสแกนผ่านดรรชนี ไปยังข้อมูลที่ตรงกับทริปเปิลแพตเทิร์นแต่ละตัวก่อน เพื่อให้ได้ค่าเริ่มต้นของฟิลเตอร์ของแต่ละตัวแปร จากนั้นทุก ๆ การเชื่อมข้อมูลที่เกิดขึ้นผ่านตัวแปร *?v* ใด ๆ จะนำช่วงของข้อมูลของผลลัพธ์ที่ได้มาอัปเดตตัวกรองของตัวแปร *?v* อีกครั้ง เพื่อลดการอัปโหลดข้อมูลในครั้งถัด ๆ ไปที่จะเกิดขึ้น

2.5 การแปลง ID

การแปลงข้อมูลจากข้อความเป็น ID สามารถทำได้ง่าย โดยการไล่ลำดับการปรากฏของข้อความนั้น เช่น เมื่อนำไฟล์ RDF มาแปลงเป็น ID ข้อความแรกที่เจอก็จะให้เป็นค่า 1 ข้อความที่เจอในครั้งถัด ๆ ไปก็จะเป็น 2, 3, 4, … ตามลำดับ หากว่าเจอข้อความนั้นซ้ำอีกครั้งก็จะทราบได้จากการเทียบกับข้อมูลที่เก็บไว้อยู่แล้วและไม่ต้องแทนค่าใหม่เนื่องจากมี ID ที่จับคู่กับข้อความนี้อยู่แล้ว

งานวิจัยนี้ค้นพบว่าการแปลง ID เป็นอีกหนึ่งส่วนสำคัญในการเพิ่มประสิทธิภาพให้กับการค้นคืนข้อมูลด้วย GPU เนื่องจากเทคนิคการกรองก่อนอัปโหลดอาศัยการตัดข้อมูลในตอนต้นและตอนปลายของข้อมูลแต่ละชุดออก ดังนั้นหากสามารถทำให้ข้อมูลที่เกี่ยวข้องกันมี ID ใกล้เคียงกัน ก็จะทำให้สามารถกรองข้อมูลออกไปก่อนได้มากขึ้น เทคนิคที่จะนำมาใช้ในการกำหนดค่า ID ของข้อมูลที่ใกล้กันคือเทคนิคของการแปลงข้อความแต่ละตัวเป็นเวกเตอร์เพื่อหาความใกล้เคียงของแต่ละค่า และใช้เทคนิคของการลดมิติข้อมูล (Dimension Reduction) เพื่อแปลงเวกเตอร์แต่ละตัวเป็นค่า ID ที่เป็นเลขตัวเดียว หรือข้อมูล 1 มิติ โดยการแปลงข้อความ *t* ให้อยู่ในรูปของเวกเตอร์ *w* ขนาด $|P|$ ทำได้ดังสมการที่ (1)

$$w_i = \begin{cases} 1, & (t, P_i, \_) \in T \; or \; (\_, P_i, t) \in T \\ 0, & Otherwise \end{cases} \quad (1)$$

เมื่อทำการแปลงข้อความทั้งหมดให้อยู่ในรูปของเวกเตอร์ขนาด $|P|$ แล้ว จะใช้เทคนิค PCA (Principal Component Analysis) หรือการวิเคราะห์องค์ประกอบหลักเพื่อแปลงข้อมูลให้อยู่ในรูปของเลขตัวเดียว จากนั้นจึงทำการกำหนด ID ให้กับข้อความทั้งหมดอีกครั้ง โดยค่าที่น้อยที่สุดที่ได้จาก PCA จะให้เป็นค่า 1 และค่าที่น้อยสุดเป็นค่าถัด ๆ ไปจะเป็น 2, 3, 4, … งานวิจัยนี้จะใช้ ID ที่ได้จากขั้นตอนนี้เพื่อไปเปรียบเทียบกับการทำงานโดยใช้การกำหนด ID แบบสุ่ม เพื่อวัดประสิทธิภาพของเทคนิคการแปลง ID ที่ได้คิดขึ้น

3. วิธีการวิจัย

สำหรับข้อมูลตัวอย่างที่จะนำมาใช้ในงานวิจัยนี้ จะใช้ข้อมูลที่สังเคราะห์ขึ้นจากชุดทดสอบ WatDiv [18] ที่มีข้อดีคือการกำหนดขนาดของข้อมูลที่ต้องการได้เอง และมีคิวรีในหลากหลายรูปแบบสำหรับใช้ทดสอบ WatDiv มีคำค้นทั้งหมด 4 ประเภทหรือ 4 คลาส โดยแต่ละประเภทจะมีรูปร่างของ BGP ดังนี้ ประเภท S เป็นรูปแบบ Star, ประเภท L เป็นรูปแบบ Linear, ประเภท F เป็นรูปแบบ Snowflake-Shaped และประเภท C เป็นรูปแบบ Complex

ในการสร้างชุดข้อมูลทดสอบด้วย WatDiv จะใช้ Scale-factor ที่ 2,000 หรือขนาด 2,000 เท่าจาก ประมาณ 100K ทริปเปิล ซึ่งจะได้ข้อมูลทริปเปิลทั้งหมด 219,838,295 ทริปเปิล โดยต่อจากนี้จะเรียกชุดข้อมูลนี้ว่า WatDiv 200M การทดลองเพื่อเปรียบเทียบกับระบบที่มีอยู่เดิมจะใช้ gStore และ RDF-3X ซึ่งเป็นงานวิจัยที่เป็นโอเพ่นซอร์สที่สามารถนำมาทดลองเปรียบเทียบได้ เพื่อแสดงให้เห็นว่าการใช้ GPU ในการคิวรีแทน CPU สามารถเพิ่มประสิทธิภาพได้อย่างไรบ้าง เครื่องเซิร์ฟเวอร์ที่ใช้ในการพัฒนาและทดสอบประสิทธิภาพมีรายละเอียดคือหน่วยประมวลผล Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz หน่วยความจำขนาด 256 GB และ GPU Nvidia Tesla V100 ขนาด 32GB

การทดลองจะแบ่งออกเป็น 3 ส่วน ได้แก่ การวัดขนาดพื้นที่จัดเก็บของแต่ละระบบ การวัดเวลาการประมวลผลของคิวรี และการเปรียบเทียบการแปลง ID ที่มีผลต่อประสิทธิภาพของการคิวรีข้อมูล

4. ผลการวิจัย

เพื่อให้ง่ายต่อการอ้างอิง ระบบที่พัฒนาขึ้นในงานวิจัยนี้จะใช้ชื่อในการเรียกว่า VEDAS และใช้ชื่อดังกล่าวในการแสดงผลการทดลอง

4.1 ขนาดการจัดเก็บข้อมูล

ทดสอบโดยการวัดขนาดพื้นที่การจัดเก็บข้อมูล WatDiv 200M บนระบบต่าง ๆ และแบบที่เป็นไฟล์ข้อมูลดิบในรูปแบบ N-Triple โดยมีหน่วยของการวัดเป็น GB ผลลัพธ์ที่ได้เป็นไปดัง**ตารางที่ 1**

ตารางที่ 1 ตัวอย่างแสดงรูปแบบของตาราง

การจัดเก็บ/ระบบ	ขนาดข้อมูล
N-Triple (ไฟล์ .nt)	30 GB
gStore	17 GB
RDF-3X	12 GB
VEDAS (Full Permutation Index)	12.5 GB
VEDAS (Predicate Index)	5.3 GB

4.2 เวลาที่ใช้ในการคิวรีข้อมูล

ในส่วนนี้จะเป็นการทดสอบเวลาที่ใช้ในการคิวรีข้อมูลของแต่ละคิวรีใน WatDiv โดยการวัดเวลาของการประมวลผลคิวรีของ RDF-3X, gStore และ VEDAS ทั้งแบบที่กำหนดค่า ID แบบสุ่ม (Random Assignment) และกำหนดค่า ID โดยใช้ความใกล้เคียงของข้อความ (Similarity) ซึ่งผลลัพธ์ของเวลาการทำงาน จำนวนข้อมูลที่เป็นผลลัพธ์แสดงไว้ดัง**ตารางที่ 2**

4.3 การเปรียบเทียบการแปลง ID

การทดลองนี้จะเป็นการวัดขนาดของข้อมูลในแต่ละตัวดำเนินการดัง**ตารางที่ 3** เพื่อเป็นการตรวจสอบประสิทธิภาพของเทคนิคการกำหนดค่า ID ที่คิดค้นขึ้น โดยจำนวนข้อมูลการอัปโหลดจะหมายถึงจำนวนแถว (Row) ของข้อมูลที่อัปโหลดจากหน่วยความจำหลักไปยังหน่วยความจำ GPU (หน่วยเป็นแถว) ซึ่งสามารถมีได้ตั้งแต่ 1-2 คอลัมน์ จำนวนข้อมูลของการเชื่อมข้อมูลจะเป็นจำนวนแถวของอินพุตทั้งสองตัวของการเชื่อมข้อมูล โดยไม่ได้ผ่านการคูณจำนวนคอลัมน์ และจำนวนข้อมูลของการสลับคอลัมน์จะเป็นจำนวนแถวของข้อมูลอินพุตเช่นกัน

5. อภิปรายผลและสรุป

งานวิจัยนี้ได้ออกแบบและพัฒนาระบบสำหรับจัดเก็บและคิวรีข้อมูล RDF โดยใช้ประโยชน์จากสถาปัตยกรรมของ GPU ที่สามารถประมวลผลแบบขนานได้อย่างรวดเร็วจากหน่วยประมวลผลจำนวนมหาศาลในฮาร์ดแวร์ ผลการทดลองพัฒนาและทำการวัดขนาดพื้นที่การจัดเก็บข้อมูลพบว่า รูปแบบการเก็บข้อมูลที่เสนอใช้พื้นที่เพียง 1 ใน 3 ของข้อมูลดิบในการเก็บข้อมูลทั้งหมด และสามารถลดขนาดการใช้พื้นที่เหลือเพียงประมาณ 18% ของข้อมูลดิบเมื่อทำการใช้เฉพาะดรรชนีที่เป็นแบบ Predicate Index และยังมีขนาดที่น้อยกว่าการเก็บข้อมูลบนระบบ RDF-3X และ gStore

การทดลองในส่วนของเวลาคิวรีข้อมูล ผลลัพธ์ที่ได้คือ VEDAS สามารถคิวรีข้อมูลโดยใช้เวลาน้อยกว่าระบบ RDF-3X และ gStore ในหลายคิวรี ซึ่งประสิทธิภาพที่สูงขึ้นมาจากพลังในการประมวลผลของ GPU ที่ช่วยลดเวลาการทำงานลงไปในขั้นตอนของการเชื่อมข้อมูล แต่จะยังมีบางคิวรีที่ VEDAS ประมวลผลได้ช้ากว่า เช่น คิวรี S1, S3, S4, L1 และ L3 ซึ่งจากการตรวจสอบเวลาการทำงานในแต่ละขั้นตอนก็พบว่าคิวรี S1, L1 และ L3 มีจำนวนข้อมูลที่ต้องอัปโหลดจำนวนมาก และมีการเชื่อมข้อมูลน้อย (เมื่อเทียบกับการอัปโหลด) ทำให้ใช้เวลาไปกับการอัปโหลดมาก ส่วนคิวรี S3 และ S4 เวลาการทำงานของ VEDAS ยังถือว่ามีความรวดเร็ว แต่ระบบ RDF-3X ทำงานได้รวดเร็วกว่า ซึ่งน่าจะมาจากคิวรีแพลน (Query Plan) สามารถคาดเดาผลลัพธ์ได้ว่าจะให้จำนวนแถวที่เป็น 0 และทำการออพติไมซ์ขั้นตอนการทำงาน

ตารางที่ 2 เวลาการคิวรีข้อมูลและสปีดอัพแต่ละคำค้นของแต่ละระบบบนข้อมูล WatDiv 200M

ประเภทคิวรี	คิวรี	เวลาการคิวรี (ms.)				สปีดอัพ		จำนวนผลลัพธ์
		RDF-3X	gStore	VEDAS (Random Assignment)	VEDAS (Similarity)	RDF-3X / VEDAS (Similarity)	gStore / VEDAS (Similarity)	
C	C1	64	699	17	**16**	4	43.69	0
	C2	1508	1301	55	**51**	29.57	25.51	39
	C3	97	3935	**87**	**87**	1.11	45.23	15941
F	F1	10	13	**9**	**9**	1.11	1.44	0
	F2	42	18	15	**13**	3.23	1.38	23
	F3	107	40	**21**	25	4.28	1.6	80
	F4	71	20	**18**	**18**	3.94	1.11	170
	F5	87	57	31	**22**	3.95	2.59	61
S	S1	21	**6**	26	17	1.24	0.35	3
	S2	41	160	**12**	**12**	3.42	13.33	5409
	S3	**4**	49	**7**	**7**	0.57	7	0
	S4	**5**	45	7	7	0.71	6.43	0
	S5	5	7	**4**	**4**	1.25	1.75	0
	S6	5	10	10	**3**	1.67	3.33	20
	S7	3	6	**2**	**2**	1.5	3	1
L	L1	50	**2**	6	6	8.33	0.33	1
	L2	22	6	**4**	**4**	5.5	1.5	525
	L3	**4**	**4**	6	6	0.67	0.67	6
	L4	11	22	**1**	**1**	11	22	655
	L5	18	64	5	**3**	6	21.33	764

ตารางที่ 3 จำนวนข้อมูลของแต่ละตัวดำเนินการในรูปแบบการกำหนดค่า ID ที่แตกต่างกัน

ประเภทคิวรี	ตัวดำเนินการ	จำนวนข้อมูล (แถว)	
		Random Assignment	Similarity
C	การอัปโหลด	114,367,374	89,442,022
	การเชื่อมข้อมูล	24,786,664	23,244,069
	การสลับคอลัมน์	552,421	552,385
F	การอัปโหลด	25,151,603	19,434,112
	การเชื่อมข้อมูล	25,543,095	19,434,112
	การสลับคอลัมน์	4,577	4,577
S	การอัปโหลด	14,500,616	8,946,191
	การเชื่อมข้อมูล	14,500,616	8,946,191
	การสลับคอลัมน์	0	0
L	การอัปโหลด	5,080,210	4,316,248
	การเชื่อมข้อมูล	5,115,114	4,351,152
	การสลับคอลัมน์	435,086	435,086

คิวรีที่มีทริปเปิลแพตเทิร์นจำนวนมากโดยส่วนใหญ่จะให้ประสิทธิภาพที่ดีบน VEDAS เนื่องจากหลังเกิดการอัปโหลดไปแล้วข้อมูลที่อยู่บนหน่วยความจำจะถูกนำมาใช้ซ้ำโดยไม่ต้องอัปโหลดใหม่ และแต่ละครั้งที่เกิดการเชื่อมข้อมูลขึ้นก็จะช่วยลดช่วงของข้อมูลที่เป็นไปได้ผ่านการกรองก่อนอัปโหลด ทำให้ข้อมูลสำหรับการอัปโหลดที่จะเกิดขึ้นหลังจากนั้นมีขนาดเล็กลง

การกำหนดค่า ID แบบใช้ความใกล้เคียงของข้อความให้ผลลัพธ์ที่ดีในระดับหนึ่ง จาก**ตารางที่ 2** คิวรีส่วนหนึ่งใช้เวลาที่น้อยลงหรือเท่าเดิมเมื่อใช้การการกำหนดค่า ID แบบใช้ความใกล้เคียงของข้อความ (ยกเว้นคิวรี F3, S1 และ S3) ถึงแม้จะยังมีคิวรีจำนวนมากที่เวลาไม่แตกต่างกับการกำหนด ID แบบสุ่ม และมีคิวรีที่ช้ากว่า (คิวรี F3) โดยเวลาที่ลดลงไปนี้เมื่อแจกแจงออกมาในรูปของจำนวนอินพุตของตัวดำเนินการดัง**ตารางที่ 3** ก็จะพบว่าเกิดจากจำนวนข้อมูลที่อัปโหลดลดลงไปจริง ซึ่งเป็นผลมาจากวิธีการกำหนด ID ที่นำเสนอ ข้อมูลจำนวนของการเชื่อมข้อมูลและการสลับคอลัมน์ที่ลดลงเป็นผลที่ตามมาจากการอัปโหลดข้อมูลที่น้อยลงเช่นกัน ดังนั้นการลดขนาดของข้อมูลที่มีการอัปโหลดจึงเป็นหัวใจสำคัญของการคิวรีข้อมูลบน GPU เพราะนอกจากจะลดเวลาในส่วนของการทำงานส่วนนี้ ยังส่งผลให้การทำงานส่วนอื่นลดลงไปด้วย

งานวิจัยนี้สามารถขยายผลไปยังฟังก์ชันการทำงานอื่น ๆ ของ SPARQL รวมไปถึงมาตรฐาน SPARQL ในเวอร์ชันใหม่ ซึ่งจะช่วยให้เกิดความสมบูรณ์แบบของระบบมากยิ่งขึ้น นอกจากนี้การพัฒนา Query Execution Planner ให้มีความเหมาะสมกับสถาปัตยกรรม และสามารถสร้างแพลนการทำงานที่ดีก็เป็นอีกหนึ่งในหัวข้อที่สำคัญในการพัฒนาต่อ รวมไปถึงการวิเคราะห์ผลกระทบของรูปแบบวิธีที่ใช้ในการกำหนดค่า ID ซึ่งอาจใช้วิธีการทางสถิติเข้ามาช่วยเพื่อให้เกิดความเข้าใจในแง่ของผลกระทบที่เกิดขึ้นจากการกำหนด ID รวมไปถึงวิธีในการกำหนด ID ที่ดีที่สุด (ซึ่งอาจขึ้นกับคิวรีโดยเฉลี่ยที่เกิดขึ้น)

6. กิตติกรรมประกาศ

งานวิจัยนี้ได้รับการสนับสนุนทุนวิจัยภายใต้โครงการปริญญาเอกกาญจนาภิเษก (คปก.) สำนักงานกองทุนสนับสนุนการวิจัย ในโครงการเลขที่ PHD/0171/2560

เอกสารอ้างอิง

[1] M. Atre, J. Srinivasan and J. A. Hendler, “BitMat: A main memory RDF triple store,”. Tetherless World Constellation, Rensselar Plytehcnic Institute, Troy NY, USA, Technical Rep., 2009.

[2] M. Galkin, K. M. Endris, M. Acosta, D. Collarana, M. E. Vidal and S. Auer, “SMJoin: A Multi-way Join

Operator for SPARQL Queries," in *Proc. 13th International Conference on Semantic Systems*, Amsterdam, Netherlands, Sep. 11–14, 2017, pp. 104–111.

[3] T. Neumann and G. Weikum, "The RDF-3X engine for scalable management of RDF data," *The VLDB Journal*, vol. 19, pp. 91–113, 2010, doi: 10.1007/s00778-009-0165-y.

[4] P. Peng, L. Zou, M. T. Özsu, L. Chen and D. Zhao, "Processing SPARQL queries over distributed RDF graphs," *The VLDB Journal*, vol. 25, pp. 243-268, 2016, doi: 10.1007/s00778-015-0415-0.

[5] S. Gurajada, S. Seufert, I. Miliaraki and M. Theobald, "TriAD: a distributed shared-nothing RDF engine based on asynchronous message passing," in *Proc. 2014 ACM SIGMOD international conference on Management of data*, Snowbird, UT, USA, Jun. 22–27, 2014, pp. 289–300.

[6] A. Bonifati, W. Martens and T. Timm, "SHARQL: Shape analysis of recursive SPARQL queries," in *Proc. 2020 ACM SIGMOD International Conference on Management of Data*, Portland, OR, USA, Jun. 14–19, 2020, pp. 2701–2704.

[7] K. Rabbani, M. Lissandrini and K. Hose, "Optimizing SPARQL queries using shape statistics," in *Proc. 24th International Conference on Extending Database Technology*, Nicosia, Cyprus, Mar. 23–26, 2021, pp. 505–510.

[8] C. Chantrapornchai and C. Choksuchat, "TripleID-Q: RDF query processing framework using GPU," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 2121–2135, 2018, doi: 10.1109/TPDS.2018.2814567.

[9] F. T. Jamour, I. Abdelaziz and P. Kalnis, "A demonstration of MAGiQ: matrix algebra approach for solving RDF graph queries," *Proceedings of the VLDB Endowment*, vol 11, no. 12, pp. 1978–1981, 2018, doi: 10.14778/3229863.3236239.

[10] Z. Yao, R. Chen, B. Zang and H. Chen, "Fast and concurrent RDF query processing using RDMA-assisted GPU graph exploration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 7, pp. 1619–1635, 2022, doi: 10.1109/TPDS.2021.3121568.

[11] S. Jiaming, X. Zhang, P. Peng, Z. Feng, and L. Zou. "Mapsq: A plugin-based mapreduce framework for sparql queries on gpu." in *Companion Proceedings of the The Web Conference*, Geneva, Switzerland, Apr. 23–27, 2018, pp. 81–82.

[12] T. Ren, G. Rao, X. Zhang, and Z. Feng, "SRSPG: A Plugin-based Spark Framework for Large-scale RDF Streams Processing on GPU," in *Proc. ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas)*, Auckland, New Zealand, Oct. 26–30, 2019, pp. 89-92.

[13] L. Zou, J. Mo, L. Chen, M. T. Özsu and D. Zhao, "gStore: answering SPARQL queries via subgraph matching," *Proceedings of the VLDB Endowment*, vol. 4, no. 8, pp. 482–493, 2011, doi: 10.14778/2002974.2002976.

[14] L. Zeng and L. Zou, " Redesign of the gStore sysem," *Frontiers of Computer science*, vol. 12, pp. 623–641, 2018, doi: 10.1007/s11704-018-7212-z.

[15] V. Ingalalli, D. Ienco, P. Poncelet and S. Villata, "Querying RDF Data Using A Multigraph-based Approach," in *Proc. 19th International Conference on Extending Database Technology*, Bordeaux, France, Mar. 15–18, 2016, pp. 245–256.

[16] C. Weiss, P. Karras, and A. Bernstein, "Hexastore: sextuple indexing for semantic web data management," *Proceedings of the VLDB Endowment*,

vol. 1, no. 1, pp. 1008–1019, 2008, doi: 10.14778/1453856.1453965.

[17] A. Schätzle, M. Przyjaciel-Zablocki, S. Skilevic and G. Lausen, "S2 RDF: RDF querying with SPARQL on spark," *Proceedings of the VLDB Endowment*, vol. 9, no. 10, pp. 804–815, 2016, doi: 10.14778/2977797.2977806.

[18] G. Aluç, O. Hartig, M. T. Özsu and K. Daudjee, "Diversified stress testing of RDF data management systems," in *13th International Semantic Web Conference*, Riva del Garda, Italy, October 19–23, 2014, pp. 197–212.