กลไกแคชเพื่อลดระยะเวลาในการค้นหาข้อมูลสำหรับฐานข้อมูลอนุกรมเวลา OpenTSDB

A Cache Mechanism for Reducing Query Time for OpenTSDB Time Series Database

ธาดา หวังธรรมมั่ง

Thada Wangthammang

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์
มหาวิทยาลัยสงขลานครินทร์

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Computer Engineering
Prince of Songkla University
2561

กลไกแคชเพื่อลดระยะเวลาในการค้นหาข้อมูลสำหรับฐานข้อมูลอนุกรมเวลา OpenTSDB

A Cache Mechanism for Reducing Query Time for OpenTSDB Time Series Database

ธาดา หวังธรรมมั่ง

Thada Wangthammang

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์
มหาวิทยาลัยสงขลานครินทร์

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Computer Engineering
Prince of Songkla University
2561

ชื่อวิทยานิพนธ์ กลไกแคชเพื่อลดระยะเวลาในการค้นหาข้อมูลสำหรับฐานข้อมูลอนุกรมเวลา OpenTSDB
ผู้เขียน นายธาดา หวังธรรมมั่ง
สาขาวิชา วิศวกรรมคอมพิวเตอร์

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

..... (ผู้ช่วยศาสตราจารย์ ดร. พิชญา ตัณฑัยย์)

คณะกรรมการสอบ

..... ประธานกรรมการ (ดร.สมชัย หลิมศิโรรัตน์)

..... กรรมการ (ผู้ช่วยศาสตราจารย์ ดร. พิชญา ตัณฑัยย์)

..... กรรมการ (ผู้ช่วยศาสตราจารย์ ดร.ปัญญยศ ไชยกาฬ)

..... กรรมการ (ผู้ช่วยศาสตราจารย์ ดร.เทพฤทธิ์ บัณฑิตวัฒนาวงศ์)

บัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ อนุมัติให้นับวิทยานิพนธ์ฉบับนี้ เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์

.....
(ศาสตราจารย์ ดร.ดำรงศักดิ์ ฟ้ารุ่งสาง)
คณบดีบัณฑิตวิทยาลัย

ขอรับรองว่า ผลงานวิจัยนี้มาจากการศึกษาวิจัยของนักศึกษาเอง และได้แสดงความขอบคุณบุคคลที่มีส่วนช่วยเหลือแล้ว

ลงชื่อ.....
(ผู้ช่วยศาสตราจารย์ ดร. พิชญา ตัณฑัยย์)
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

ลงชื่อ.....
(นายธาดา หวังธรรมมั่ง)
นักศึกษา

ข้าพเจ้าขอรับรองว่า ผลงานวิจัยนี้ไม่เคยเป็นส่วนหนึ่งในการอนุมัติปริญญาในระดับใดมาก่อน และไม่ได้ถูกใช้ในการยื่นขออนุมัติปริญญาในขณะนี้

ลงชื่อ.....
(นายธาดา หวังธรรมมั่ง)
นักศึกษา

ชื่อวิทยานิพนธ์ กลไกแคชเพื่อลดระยะเวลาในการค้นหาข้อมูลสำหรับฐานข้อมูลอนุกรมเวลา OpenTSDB
ผู้เขียน นายธาดา หวังธรรมมั่ง
สาขาวิชา วิศวกรรมคอมพิวเตอร์
ปีการศึกษา 2561

บทคัดย่อ

ปัจจุบันปริมาณข้อมูลของ Internet of things ได้เติบโตอย่างรวดเร็ว ซึ่งฐานข้อมูลอนุกรมเวลาก็เป็นส่วนสำคัญของระบบที่ช่วยจัดเก็บและดึงข้อมูลขนาดใหญ่เพื่อการวิเคราะห์ผลและการตรวจตราในระบบต่างๆ โดยพฤติกรรมของผู้ใช้งานจะมีการใช้งานการสอบถามซึ่งมีโอกาสที่ช่วงเวลาของอนุกรมเวลาทับซ้อนกับช่วงเวลาที่เคยค้นหาไว้แล้ว โดยเฉพาะอย่างยิ่งในกรณีของผู้ใช้ปรับตัวแปรของการสอบถามบนข้อมูลอนุกรมเวลาเดียวกันเพื่อที่จะแสดงผล การแคชข้อมูลที่เคยเรียกผ่านมาแล้วจะช่วยลดระยะเวลาในการตอบสนอง ในกรณีที่ช่วงเวลาการสอบถามปัจจุบันซ้อนทับกับช่วงเวลาเคยสอบถามไว้แล้ว วิทยานิพนธ์นี้เสนอกลไกแคชสำหรับฐานข้อมูลอนุกรมเวลา OpenTSDB ซึ่งเป็นฐานข้อมูลแบบเปิดเผยรหัสต้นฉบับ จากการศึกษากลไกแคชที่ได้ออกแบบพบว่าระยะเวลาในการตอบสนองสามารถลดลง 1.5 - 25 เท่าเมื่อเปรียบเทียบกับการสอบถามของฐานข้อมูล OpenTSDB แบบดั้งเดิมซึ่งความเร็วที่เพิ่มขึ้นนั้นขึ้นอยู่กับพฤติกรรมและตัวแปรของการสอบถามที่ผู้ใช้กำหนด

Thesis Title A Cache Mechanism for Reducing Query Time for OpenTSDB Time Series Database
Author Mr. Thada Wangthammang
Major Program Computer Engineering
Academic Year 2018

ABSTRACT

As the volume of Internet of Things (IoT) data grows rapidly, a time series database is a major part of the system that helps in storing and retrieving the massive data for analysis and monitoring. According to the user behaviors, the data queries often are similar or overlapped to the previous queries, especially when the user adjusts the query parameters on the same time series data for displaying on a visualization tool. Caching the previous data will help reduce the query response time, in case that the current query range is overlapping with the previous query ranges. This thesis proposes a cache mechanism for an open source time series database, OpenTSDB. Exploiting our cache mechanism, the query response time can be 1.5-25 times faster than the original OpenTSDB depending on the user query behaviors and related parameters.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความช่วยเหลืออย่างดียิ่งจากผู้ช่วยศาสตราจารย์ ดร. พิชญา ตัณฑัยย์ อาจารย์ที่ปรึกษาที่ได้ความอนุเคราะห์และสละเวลาอันมีค่ายิ่งในการแนะนำแนวทางการทำวิทยานิพนธ์ และให้ข้อคิดเห็นที่เหมาะสมและเป็นประโยชน์

ขอขอบพระคุณคณะกรรมการควบคุมการสอบวิทยานิพนธ์ ประกอบด้วย ดร.สมชัย หลิมศิโรรัตน์ ผู้ช่วยศาสตราจารย์ ดร.ปัญญยศ ไชยกาฬ และผู้ช่วยศาสตราจารย์ ดร.เทพฤทธิ์ บัณฑิตวัฒนาวงศ์ และคณาจารย์ทุกท่าน

ขอขอบคุณพระคุณบัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ ได้ให้ความกรุณามอบทุนการศึกษาผลการเรียนดีเด่นเพื่อเข้าศึกษาระดับปริญญาโท ประจำปีการศึกษา 2558 เพื่อสนับสนุนให้วิทยานิพนธ์นี้สำเร็จลุล่วงไปได้ด้วยดี

นอกจากนี้ต้องขอบพระคุณบิดา มารดา นายเมธา หวังธรรมมั่ง และคนในครอบครัวทุกท่าน ขอบคุณเพื่อนร่วมเรียนปริญญาโทและเอกทุกท่าน ซึ่งคอยช่วยเหลือและให้คำปรึกษาที่เป็นประโยชน์มาตลอด อีกทั้งขอขอบพระคุณ รองศาสตราจารย์ แพทย์หญิง ธัญญพัทธ์ เบญจวลีย์มาศ ดร.ทันตแพทย์หญิง ธัญนันท์ กำภู และนายอธิสิทธิ์ แซ่อั้ง ซึ่งเป็นผู้ที่ให้ทั้งแนวคิดและคำปรึกษาที่ดีเกี่ยวกับการบริหารจัดการเวลาและการดำเนินชีวิตมาโดยตลอด

สุดท้ายนี้ผู้วิจัยหวังเป็นอย่างยิ่งว่า วิทยานิพนธ์ฉบับนี้จะเป็นประโยชน์แก่ผู้ที่สนใจศึกษาค้นคว้า เพื่อสร้างสรรค์ผลงานวิจัยที่เป็นประโยชน์ต่อผู้อื่นอีกมากมาย

ธาดา หวังธรรมมั่ง

สารบัญ

สารบัญ (ต่อ)

รายการตาราง

ตาราง หน้า

รายการภาพประกอบ

รูป หน้า

รายการภาพประกอบ (ต่อ)

รูป หน้า

สัญลักษณ์คำย่อและตัวย่อ

CS	Chunk Size
et	End Time
FO	Fragment Order
IoT	Internet of Things
LRU	Least Recently Used
OpenTSDB	Open Time Series Database
op	Overlapping percentage
rs	Range Size
st	Start Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มา

Internet of things หรือ IoT ใช้งานอย่างกว้างขวางในหลากหลายด้าน ไม่ว่าจะเป็นด้านระบบแนะนำต่าง ๆ ระบบบ้านอัจฉริยะ ระบบประยุกต์ด้าน IoT ระบบตรวจตรา หรือการวิเคราะห์การเงิน [1] นักวิจัยและบริษัทต่าง ๆ ล้วนตื่นตัวกับเทคโนโลยีนี้เพื่อที่จะพัฒนาสินค้าและบริการ ยกตัวอย่างเช่น ระบบฟาร์มอัจฉริยะที่นำเซนเซอร์เข้ามาติดตั้งเพื่อตรวจสอบคุณภาพของน้ำแบบทันเวลาจริง (real-time system) ยิ่งไปกว่านั้นแล้ว ข้อมูลทางด้านนี้ยังมีขนาดใหญ่มาก อีกทั้งยังเติบโตอย่างรวดเร็วและต่อเนื่อง จึงทำให้จำเป็นต้องใช้พื้นที่ในการจัดเก็บจำนวนมาก การอ่านข้อมูลจากหน่วยจัดเก็บข้อมูลนั้น จะใช้เวลานานเท่าใดนั้นขึ้นอยู่กับชนิดของหน่วยจัดเก็บข้อมูล หนึ่งในชนิดข้อมูลที่เหมาะสมสำหรับการวิเคราะห์ข้อมูลนั้นคืออนุกรมเวลา ซึ่งจะจัดเก็บค่าของข้อมูลที่บันทึกไว้ร่วมกับเวลาที่บันทึก เช่น อุณหภูมิ สภาพอากาศรายชั่วโมง เป็นต้น เราสามารถนำความสัมพันธ์ของข้อมูลเหล่านี้มาใช้วิเคราะห์ในหลากหลายรูปแบบ เช่น การหาวัฏจักรหรือแนวโน้มของอนุกรมเวลา

อนุกรมเวลาคือชุดข้อมูลที่เรียงลำดับตามเวลา [2] โดยอนุกรมเวลาประกอบไปด้วยจุดข้อมูล (data point) ซึ่งในแต่ละจุดข้อมูลมีเวลาที่บันทึก และค่าที่วัดได้ ณ เวลานั้นๆ ข้อมูลอนุกรมเวลาส่วนใหญ่แล้วใช้ในการจัดเก็บข้อมูลจากสภาพแวดล้อมต่าง ๆ ขึ้นอยู่กับการใช้งาน โดยในหลายๆงานก็ได้นำข้อมูลอนุกรมเวลาไปใช้ เช่น การพยากรณ์สภาพอากาศ, ระบบคอมพิวเตอร์, การจราจร, กระบวนทางเคมี หรือแม้กระทั่งการพยากรณ์แนวโน้มของราคาหุ้น

อนุกรมเวลาส่วนใหญ่มักจะถูกเก็บเพื่อวัตถุประสงค์ของการวิเคราะห์ข้อมูลเป็นหลัก โดยการใช้มักจะนำไปใช้ในการพยากรณ์, การประเมินค่าฟังก์ชันการแลกเปลี่ยน (transfer functions estimation), การวิเคราะห์ผลกระทบของเหตุการณ์ที่ผิดปกติ (unusual intervention events effect) หรือระบบการควบคุมแบบเต็มหน่วย (discrete control system) [3] ตัวอย่างของข้อมูลอนุกรมเวลาในกรณีของการเก็บข้อมูลสภาพอากาศ เช่น ลำดับของอุณหภูมิที่ถูกบันทึกเป็นรายชั่วโมง, การบันทึกความชื้นทุกชั่วโมง หรือการบันทึกความเร็วทุกๆ หนึ่งนาที ซึ่งระยะห่างระหว่างเวลาที่บันทึก ของข้อมูลแต่ละจุดจะเป็นเท่าใดก็ได้

ฐานข้อมูลแบบอนุกรมเวลา [4] [5] คือฐานข้อมูลที่ถูกพัฒนาและปรับปรุงมาเพื่อเก็บข้อมูลในงานทางด้านอนุกรมเวลาโดยเฉพาะ ฐานข้อมูลแบบอนุกรมเวลาส่วนใหญ่แล้วจะถูกปรับแต่งให้เหมาะกับข้อมูลที่มีปริมาณมาก และสามารถที่จะจัดเก็บจุดข้อมูลของอนุกรมเวลาและคำอธิบายเพิ่มเติมของข้อมูลนั้นๆ เท่านั้น ดังนั้นฐานข้อมูลแบบอนุกรมเวลาสามารถจัดการข้อมูลแบบอนุกรมเวลาได้เหมาะสมกว่าฐานข้อมูลอเนกประสงค์ทั่วๆ ไป เช่น ฐานข้อมูลแบบมีความสัมพันธ์ (Relational database) หรือฐานข้อมูลแบบไม่มีความสัมพันธ์ (Non-relational database) อีกทั้งฐานข้อมูลแบบนี้ยังสามารถค้นหาข้อมูลแบบอนุกรมเวลา เลื่อนช่วงของเวลาที่ต้องการสอบถาม รวมหลายๆ อนุกรมเวลาเข้ามาเป็นหนึ่งอนุกรมเวลา หรือคำนวณเพื่อหาค่าของจุดที่หายไปของอนุกรมเวลา (interpolation) ปัจจุบันยังไม่มีมาตรฐานการออกแบบและการ

เชื่อมต่อที่ชัดเจนที่จะอธิบายว่าฐานข้อมูลแบบอนุกรมเวลานั้นควรเป็นแบบไหน และมีการเชื่อมต่ออย่างไร หลายๆ องค์กรได้มีความพยายามที่จะสร้างฐานข้อมูลแบบอนุกรมเวลาสำหรับใช้ในองค์กร ซึ่งแต่ละผู้พัฒนาได้ออกแบบความสามารถและ API ของตนแตกต่างกันไป

ฐานข้อมูลอนุกรมเวลาส่วนใหญ่แล้วมักจะมี 4 องค์ประกอบ คือ ส่วนของการเก็บข้อมูลอนุกรมเวลา ส่วนของการเก็บเกี่ยวข้อมูล ส่วนของการแสดงผล และส่วนของการประมวลผลข้อมูล ซึ่งบางฐานข้อมูลอนุกรมเวลามีองค์ประกอบไม่ครบ แต่ฐานข้อมูลเหล่านั้นมักอนุญาตให้เชื่อมต่อกับภายนอก ยกตัวอย่าง เช่น OpenTSDB มีส่วนของแสดงผลที่เหมาะสำหรับการใช้งานที่ไม่ซับซ้อน แต่ OpenTSDB ก็สามารถเชื่อมต่อกับโปรแกรมประยุกต์เพื่อการแสดงผลอนุกรมเวลา เช่น Grafana [6] ได้ ซึ่ง Grafana คือ หน้าต่างอเนกประสงค์สำหรับแสดงผลข้อมูลอนุกรมเวลา และมีส่วนขยายสำหรับการเชื่อมต่อกับ OpenTSDB อีกด้วย

ฐานข้อมูลแบบอนุกรมเวลาที่เปิดเผยรหัสต้นฉบับและได้รับความนิยม เช่น OpenTSDB KairosDB [7] InfluxDB [8] Prometheus [9] และ Elasticsearch [10] เป็นต้น โดยฐานข้อมูล KairosDB ถูกพัฒนาต่อมาจาก OpenTSDB รุ่นที่ 1.0 ซึ่งใช้ Cassandra [11] เป็นฐานข้อมูล และได้พัฒนาเพื่อการจัดเก็บข้อมูลประเภทข้อความและชนิดอื่นๆ [12] ส่วน OpenTSDB สามารถขยายตัวแบบใช้งานหลายเครื่องได้ง่ายว่า Prometheus และ InfluxDB [9] ทั้งนี้เพื่อรองรับการทำงานของผู้ใช้จำนวนมาก แต่ทว่าการติดตั้งและตั้งค่ายังคงสลับซับซ้อนเพราะว่า OpenTSDB ใช้ HBase เป็นฐานข้อมูล Elasticsearch ตั้งค่าได้ง่ายกว่า OpenTSDB และ InfluxDB แต่อย่างไรก็ตาม OpenTSDB มีสามารถขยายตัวได้ดีกว่า Elasticsearch และ InfluxDB แต่ Elasticsearch เหมาะสำหรับงานประเภทการตรวจการตามเวลาจริง (real time) มากกว่า ส่วน InfluxDB นั้นเป็นฐานข้อมูลที่ค่อนข้างใหม่ยังมีการปรับปรุงพัฒนาอยู่บ่อยครั้ง และไม่สามารถขยายตัวได้ดี [10] อีกทั้ง OpenTSDB ยังมีชุมชนของนักพัฒนาที่ดี

ความเร็วในการตอบสนองสำคัญอย่างยิ่งในสถานการณ์ฉุกเฉิน Weigel และคณะได้เสนอกรณีการศึกษาจากการใช้ฐานข้อมลอนุกรมเวลา Time Series Data Server (TSDS) [13] ดังนั้นความเร็วในการดึงข้อมูลออกจากฐานข้อมูลอนุกรมเวลาเพื่อวิเคราะห์นั้นสำคัญ โดยสามารถช่วยชีวิตผู้คนจำนวนมากได้ การเพิ่มประสิทธิภาพของฐานข้อมูลอนุกรมเวลานั้น ได้มีหลายงานวิจัยที่ได้ปรับปรุงฐานข้อมูลอนุกรมเวลาโดยใช้หลากหลายวิธีการ โดยขึ้นอยู่กับวัตถุประสงค์ของโปรแกรมประยุกต์ [14] [15] ซึ่งหนึ่งในวิธีการนั้นคือการย้ายจากการใช้ฮาร์ดดิสก์มาใช้หน่วยความจำหลักแทน และใช้ฮาร์ดดิสก์เป็นหน่วยความจำสำรองเพื่อเพิ่มประสิทธิภาพในการสอบถามโดยรวม เช่น Gorilla [8] โดยวิธีการนี้เหมาะสำหรับโปรแกรมประยุกต์แบบทันเวลาจริง เช่น ระบบการตรวจตรา อย่างไรก็ตามวิธีการนี้ถึงแม้จะเพิ่มความเร็วได้มาก แต่ก็ยังคงมีต้นทุนของหน่วยความหลัก อีกวิธีการคือการปรับปรุงประสิทธิภาพการสอบถามข้อมูลโดยการใช้เทคนิคการแคช โดยทั่วไปแล้ว ประโยชน์ของการใช้แคชคือเพิ่มความเร็วในการทำงาน และในอีกทางหนึ่งคือช่วยลดต้นทุนของการใช้หน่วยความหลักในการเก็บข้อมูล

ฐานข้อมูลอนุกรมเวลาส่วนใหญ่มักจะประยุกต์ใช้กับงานการตรวจตราและการวิเคราะห์ข้อมูล โดยความต้องการของผู้ใช้งานของระบบตรวจตราคือการพบเหตุการณ์ผิดปกติให้ทันเวลาจริง ซึ่งส่วนใหญ่แล้ว ระบบนั้นจะประยุกต์ใช้หน่วยความจำหลักก่อนการเข้าถึงหน่วยเก็บความจำถาวร ซึ่งโดยปกติแล้วจะมีความเร็วในการอ่านและเขียนช้ากว่า เช่น ฮาร์ดดิสก์ [15] ส่วนการวิเคราะห์ข้อมูลนั้นระบบยังคงต้องการการอ่านข้อมูลที่เร็วแต่ไม่ต้องถึงขั้นทันเวลาจริง ทั้งนี้ขึ้นอยู่กับความต้องการของระบบและงบประมาณ เทคนิค

การแคชสามารถเพิ่มความเร็วในการอ่านของระบบอนุกรมเวลาเพื่อวิเคราะห์ข้อมูลได้ ซึ่งเทคนิคการแคชนำไปประยุกต์เข้ากับระบบการเก็บข้อมูลอนุกรมเวลาสำหรับเซนเซอร์แบบลำดับชั้น ซึ่งมีการจัดเก็บกระจัดจายในหลายพื้นที่ [16] และมีหลายลำดับชั้นของแต่ละเขต โดย Deshpande และคณะได้ออกแบบกลไกแคชสำหรับจัดการข้อมูลที่มีความเหลื่อมกันในแต่ละเขตพื้นที่ ซึ่งเรียกว่า การพบข้อมูลในแคชแบบบางส่วน (Partially-matched) ส่วนระบบฐานข้อมูล Druid ถูกออกแบบสำหรับงานวิเคราะห์ข้อมูล [17] ในระบบมีกลไกการแคชเพื่อลดระยะเวลาในการสอบถามข้อมูล แต่อย่างไรก็ตามในบทความของทั้งสองระบบพวกเขาไม่ได้อธิบายรายละเอียดของการออกแบบและสถาปัตยกรรมของกลไกแคช แต่สนใจเฉพาะกลไกแคชสำหรับฐานข้อมูลแบบลำดับชั้นสำหรับการดึงข้อมูลให้เร็วขึ้นจากเขตพื้นที่ที่อยู่กระจัดกระจายกัน และเป็นการออกแบบเฉพาะเจาะจงกับระบบนั้นๆ

OpenTSDB [18] เป็นฐานข้อมูลอนุกรมเวลาชนิดหนึ่ง ที่อนุญาตให้ผู้ใช้สามารถจัดการและจัดเก็บข้อมูลอนุกรมเวลาได้ อีกทั้งยังมีความสามารถในการกระจายตัวข้อมูลและการขยายตัวของระบบ ฐานข้อมูลนี้สามารถแสดงผลข้อมูลขนาดใหญ่ของอนุกรมเวลาในรูปแบบกราฟได้อย่างรวดเร็ว แต่อย่างไรก็ตามฐานข้อมูลนี้ยังคงมีปัญหาในด้านการสอบถามข้อมูล เมื่อช่วงระยะเวลาของคำร้องขอปัจจุบันนั้นซ้อนทับกับช่วงระยะเวลาของคำร้องที่เคยสอบถามไว้แล้ว แต่ระบบยังคงดึงข้อมูลจากฐานข้อมูลใหม่ทั้งหมด ถ้าระบบสามารถค้นหาเฉพาะช่วงระยะเวลาที่ไม่เคยค้นหามาก่อนได้ ประสิทธิภาพการทำงานของฐานข้อมูล OpenTSDB ก็จะสูงขึ้น โดยสามารถลดระยะเวลาในการสอบถามข้อมูลได้ โดยเฉพาะอย่างยิ่งในกรณีการให้บริการของผู้ใช้จำนวนมาก ดังนั้นวิทยานิพนธ์นี้จึงเลือก OpenTSDB เป็นฐานข้อมูลสำหรับนำเสนอการออกแบบกลไกแคชสำหรับฐานข้อมูลอนุกรมเวลา

1.2 วัตถุประสงค์

ออกแบบและพัฒนากลไกแคชแบบซอฟต์แวร์สำหรับการสอบถามข้อมูลในฐานข้อมูลแบบอนุกรมเวลาเพื่อลดระยะเวลาโดยรวมในการสอบถามข้อมูลของ OpenTSDB

1.3 ขอบเขตของการวิจัย

- ใช้ฐานข้อมูล OpenTSDB เป็นกรณีศึกษาของกลไกที่จะนำเสนอ
- ใช้ระบบจัดการแคช Memcached สำหรับจัดการวัตถุแคช และใช้กระบวนการแทนที่แคช (Cache Replacement)
- พัฒนาระบบทดสอบบน Docker
- ข้อมูลที่นำมาทดสอบเป็นข้อมูลที่สร้างขึ้นแบบสุ่ม
- สถานการณ์ที่นำมาทดสอบระบบนั้นเป็นสถานการณ์จำลองเพื่อวัดประสิทธิภาพของกลไกแคช

- การวัดผลจะวัดจากระยะเวลาในการอ่านข้อมูลดิบจากฐานข้อมูล HBase เท่านั้น ไม่นับขั้นตอนอื่นๆ ที่อยู่ในกระบวนการสอบถามข้อมูลของ OpenTSDB
- การตั้งค่าฐานข้อมูล HBase ใช้การตั้งค่าแบบเริ่มต้นทั้งหมด
- กลไกแคชที่ได้พัฒนานั้นรองรับเพียงคำสั่งการสอบถามพื้นฐานที่มีการกำหนดระยะเวลาของการสอบถาม, metric และ tag หนึ่งคู่เท่านั้น กลไกแคชไม่ครอบคลุมความสามารถอื่นของฐานข้อมูล OpenTSDB

1.4 ประโยชน์ที่คาดว่าจะได้รับ

สามารถลดระยะเวลาในการแสดงผลของเครื่องมือแสดงผลกราฟได้ และโดยทั่วไปแล้วผู้ใช้สามารถสอบถามข้อมูลได้เร็วขึ้น

บทที่ 2

การตรวจเอกสาร

2.1 งานวิจัยที่เกี่ยวข้อง

หลายงานวิจัยได้พยายามปรับปรุงประสิทธิภาพของฐานข้อมูลในหลากหลายวิธี ขึ้นอยู่กับลักษณะงานที่นำไปประยุกต์ใช้ [14] [15] โดยสามารถแบ่งได้สองด้านคือ ด้านการปรับปรุงพัฒนาฐานข้อมูลอนุกรมเวลาเพื่อวัตถุประสงค์การใช้งานที่แตกต่างกัน และด้านการใช้เทคนิคแคชเฉพาะบางส่วน

2.1.1 ด้านการปรับปรุงพัฒนาฐานข้อมูลอนุกรมเวลา

โดยทั่วไปแล้ว ฐานข้อมูลอนุกรมเวลาสามารถแบ่งตามประเภทของการใช้งานได้ 2 ประเภท คือ ใช้เพื่อตรวจตราข้อมูลอนุกรมเวลาและเตือนผู้ใช้งาน และใช้เพื่อค้นหาข้อมูลอนุกรมเวลาที่ถูกเก็บในอดีต

ในการตรวจตราและการแจ้งเตือนข้อมูลอนุกรมเวลานั้นต้องการระบบที่สามารถทำงานได้ทันเวลาจริง เพื่อตรวจพบเหตุการณ์ที่ผิดปกติตามเงื่อนไขที่กำหนดไว้ โดยฐานข้อมูลชนิดนี้ใช้พื้นที่หน่วยความจำหลักเป็นจำนวนมากสำหรับการจัดการข้อมูลปริมาณมาก ๆ เช่น ฐานข้อมูลอนุกรมเวลาที่อยู่บนหน่วยความจำหลัก ซึ่งเรียกว่า Gorilla [8] และระบบจัดการข้อมูลแบบ stream สำหรับการตรวจตราที่ชื่อว่า Aurora [19]

การสอบถามข้อมูลอนุกรมเวลาที่ถูกเก็บในอดีตนั้นถูกออกแบบเพื่อเก็บข้อมูลขนาดใหญ่และเก็บข้อมูลเป็นระยะเวลานาน โดยส่วนใหญ่แล้วฐานข้อมูลอนุกรมเวลาประเภทนี้จะจัดเก็บจุดข้อมูลทั้งหมดบนฮาร์ดดิสก์ อย่างไรก็ตาม การจะเก็บข้อมูลอนุกรมเวลาทั้งหมดลงบนหน่วยความหลักนั้น จำเป็นต้องใช้พื้นที่ในการจัดเก็บมาก และส่งผลต่อต้นทุนในการทำเครื่องแม่ข่ายสำหรับการให้บริการ ดั้งนั้น ระบบการตรวจตราและการแจ้งเตือนนั้นโดยส่วนใหญ่แล้วจะไม่เก็บข้อมูลทั้งหมด แต่จะกำจัดข้อมูลเก่าออกเพื่อสำรองพื้นที่สำหรับข้อมูลที่จะเข้ามาใหม่ในระบบ

Gorilla เป็นฐานข้อมูลอนุกรมเวลาบนหน่วยความจำที่สามารถขยายตัวได้ ถูกนำไปใช้งานในระบบ Facebook [8] ซึ่งสามารถตรวจตราข้อมูลอนุกรมเวลาขนาดใหญ่ในระบบได้อย่างรวดเร็วตามความต้องการของ Facebook เพราะฉะนั้น Gorilla จึงได้ประยุกต์ฐานข้อมูลอนุกรมเวลาบนหน่วยความจำหลัก และมีระบบตรวจการสำหรับเตือนเหตุการณ์ผิดปกติภายในโครงสร้าง Facebook โดย Gorilla มีความสามารถในการแจ้งเตือนได้ใกล้เคียงเวลาจริงและค้นหาโดยใช้ปริมาณหน่วยความจำหลักมาก ซึ่งสามารถปรับปรุงพัฒนาให้ได้ผลลัพธ์เร็วขึ้นถึง 73 เท่าเมื่อเปรียบเทียบกับฐานข้อมูล HBase ที่เก็บข้อมูลลงบนฮาร์ดดิสก์ ซึ่ง Gorilla สนใจการตรวจตราระบบของกลุ่มคอมพิวเตอร์แบบทันเวลาจริงในสภาพแวดล้อมการทำงานเพื่อบริการผู้ใช้งาน โดยจะบันทึกข้อมูลย้อนหลังเพียง 26 ชั่วโมง ยิ่งไปกว่านั้น Gorilla ยังถูกออกแบบมาเพื่อฐานข้อมูลอนุกรมเวลาที่ทำงานอยู่บนหน่วยความจำหลัก ซึ่งไม่ใช่แคช ฐานข้อมูลนี้ไม่เหมาะกับระบบที่มีต้นทุนจำกัด โดย Gorilla ไม่ได้พิจารณาเรื่องปัญหาความจุของหน่วยความจำหลักต่ำ

2.1.2 ด้านการใช้เทคนิคแคชเฉพาะบางส่วน

เทคนิคนี้ส่วนใหญ่มักจะนำมาประยุกต์ใช้กับระบบฐานข้อมูลของเซนเซอร์ซึ่งมีข้อมูลจำนวนมาก และมีโอกาสที่ข้อมูลจะเป็นชิ้นเล็กๆ และส่วนใหญ่อยู่ในลักษณะของการเก็บเพื่อวิเคราะห์ข้อมูลย้อนหลังและของการเก็บเอาไว้นาน เมื่อผู้ใช้ต้องการสอบถามกลุ่มของข้อมูลชิ้นเล็กๆ นั้น เมื่อทุกๆ ชิ้นนั้นดึงออกมาจากฐานข้อมูลที่อยู่กระจัดกระจาย กระบวนการดังกล่าวอาจจะทำงานได้ช้ากว่าการเก็บข้อมูลลงบนฮาร์ดดิสก์ แนวทางการแก้ปัญหาคือเก็บข้อมูลชิ้นเล็กๆ ในแคชเพื่อให้อ่านค่าจากแคช จะทำให้ได้ข้อมูลเร็วขึ้น แต่ถ้าบางชิ้นไม่ได้อยู่ในแคช ระบบจะดึงส่วนที่เหลือของชิ้นส่วนของข้อมูลจากฐานข้อมูลในฮาร์ดดิสก์แล้วนำมารวมกับข้อมูลที่มีอยู่ในแคชอยู่แล้ว

Deshpande และคณะได้นำเสนอระบบ **Cache-and-Query** ซึ่งเป็นฐานข้อมูลสำหรับเก็บข้อมูลเซนเซอร์ที่อยู่ต่างที่กัน [16] หรือ IrisNet [20] ในระบบนี้สนใจเรื่องของการสอบถามข้อมูลเซนเซอร์จากฐานข้อมูลซึ่งจะจัดเก็บข้อมูลกระจายกันในหลายพื้นที่ ซึ่งอาจมีระยะห่างตั้งแต่ 10 - 100 ไมล์ โดยข้อมูลจัดเก็บในรูปแบบ XML ซึ่งหลายๆ เขตนั้นถูกเก็บข้อมูลแบบกระจายเป็นบริเวณกว้าง แต่ละเขตมีการจัดการโครงสร้างแบบลำดับชั้น แต่ละลำดับขึ้นอยู่กับภูมิภาคและสถานที่ ระบบนี้ไม่ได้เป็นฐานข้อมูลแบบมีศูนย์กลาง แต่จะจัดเก็บข้อมูลแยกกันแต่ละเขต ดังนั้น พวกเขาจึงเสนอแนวทางการแคชแบบ *partial match caching* ซึ่งจะเก็บข้อมูลลงแคชบางส่วนของแต่ละเขต และถ้าผู้ใช้ต้องการสอบถามข้อมูลจากหลายๆ เขต ไม่จำเป็นต้องเข้าถึงข้อมูลทุกเขตที่กระจายตัวกันอยู่พื้นที่ห่างไกล หากไม่ทำอย่างนั้นแล้ว ระบบจะใช้เวลานานมากในการเข้าถึงข้อมูลตรงๆ จากแต่ละเขตที่อยู่ห่างไกลกัน เทคนิคนี้จึงทำการแคชผลลัพธ์ของการสอบถามที่เคยดำเนินการมาแล้ว เพื่อลดระยะเวลาในการสอบถามข้อมูล แต่อย่างไรก็ตาม งานของพวกเขาอธิบายเพียงแค่หลักการย่อๆ ของการทำแคช และไม่ได้อ้างอิงช่วงเวลาที่เป็นจุดตัดของข้อมูลที่จะตัดแบ่งลงแคช เพียงแค่แยกข้อมูลแต่ละเขตออกจากกันว่าอยู่ในระบบแคชหรือไม่ และไม่ได้กล่าวถึงประสิทธิภาพของระบบแคช

Druid เป็นฐานข้อมูลอนุกรมเวลาซึ่งพัฒนาโดย Yang และคณะ [17] โดยระบบนี้จะเก็บเกี่ยวเหตุการณ์อนุกรมเวลาจากไฟล์ จากนั้นจะนำไปวิเคราะห์ข้อมูลเพื่อประมวลผล Online analytical processing (OLAP) เพื่อให้ทันเวลาจริง ฐานข้อมูล Druid ยังมี historical node สำหรับการประมวลผลการสอบถาม เพื่อที่จะลดระยะเวลาในการตอบรับของการสอบถามข้อมูล พวกเขาได้แบ่งช่วงของการสอบถามออกเป็นส่วนๆ โดยแบ่งจากเวลาเพื่อการทำแคช ส่วนงานชิ้นนี้ได้มีการแบ่งการสอบถามโดยอ้างอิงเวลาเช่นกันเพื่อการแคช แต่อย่างไรก็ตามพวกเขาไม่ได้กล่าวถึงรายละเอียดของการออกแบบกลไกแคชและการออกแบบยังเฉพาะเจาะจงสำหรับฐานข้อมูล Druid

วิทยานิพนธ์นี้นำเสนอกลไกสำหรับฐานข้อมูลอนุกรมเวลา OpenTSDB โดยการจัดเก็บช่วงข้อมูลที่มีระยะเวลาของข้อมูลอนุกรมเวลาที่เคยค้นหาไว้แล้วลงในหน่วยความจำหลักในลักษณะของแคช ดังนั้นเวลาในการประมวลผลจะลดลงในกรณีที่ช่วงระยะเวลาของคำร้องขอนั้นซ้อนทับกับคำร้องขอที่ผ่านมา เพื่อเพิ่มประสิทธิภาพให้กับการสอบถามข้อมูลอนุกรมเวลามากยิ่งขึ้น

2.2 ความรู้พื้นฐานและเครื่องมือที่เกี่ยวข้อง

หัวข้อนี้กล่าวถึงความรู้พื้นฐานที่เกี่ยวข้องกับการออกแบบกลไกแคชสำหรับฐานข้อมูล OpenTSDB และรวมถึงเครื่องมือที่ใช้ โดยแบ่งเป็น 5 หัวข้อได้แก่ อนุกรมเวลาและฐานข้อมูลอนุกรมเวลาที่ได้รับความนิยมมากในปัจจุบัน และการนำไปประยุกต์ใช้ในงานต่าง ๆ สถาปัตยกรรมและการทำงานเบื้องต้นของฐานข้อมูลอนุกรมเวลา OpenTSDB ซึ่งใช้ในวิทยานิพนธ์ฉบับนี้ ความรู้พื้นฐานในการออกแบบกลไกแคชโดยใช้หลักการของลำดับชั้นของหน่วยความจำ และในงานชิ้นนี้ใช้ Memcached ในการจัดการและจัดเก็บแคช สุดท้ายกล่าวถึงโครงสร้างของ Memcached เพื่อประกอบการออกแบบกลไกแคชสำหรับ OpenTSDB ที่ใช้ Memcached ร่วมด้วย

ตัวอย่างของการสอบถามข้อมูลจากฐานข้อมูลอนุกรมเวลา

ประเภทงานส่วนใหญ่ของการใช้ฐานข้อมูลแบบอนุกรมเวลาคือการสอบถามข้อมูลอนุกรมเวลา ฐานข้อมูลแบบอนุกรมเวลาส่วนใหญ่จะมีการออกแบบที่ทำให้สะดวกต่อการใช้ค้นหาตามช่วงของเวลา สมมติว่า เราต้องการสอบถามข้อมูลอุณหภูมิที่จัดเก็บไว้ในฐานข้อมูลอยู่ก่อนแล้วจากปีค.ศ. 2013 ถึง 2016 จากข้อมูลตัวอย่าง มีข้อมูลจำนวน 1 ล้านจุดต่อข้อมูลระยะเวลา 1 ปี คำสั่งในการสอบถามมีลักษณะได้หลากหลายรูปแบบ โดยรูปแบบข้างล่างนี้เป็นตัวอย่างอย่างง่ายในการสอบถามข้อมูล

```
QUERY temperature FROM 2013 TO 2016
```

ผลลัพธ์การสอบถามจะเป็นในรูปแบบของอาร์เรย์ของจุดข้อมูล โดยผลลัพธ์จะมี 3 ล้านจุดข้อมูล และผลลัพธ์ที่ได้ยังสามารถนำไปแสดงผลเป็นกราฟอนุกรมเวลา

2.2.1 OpenTSDB

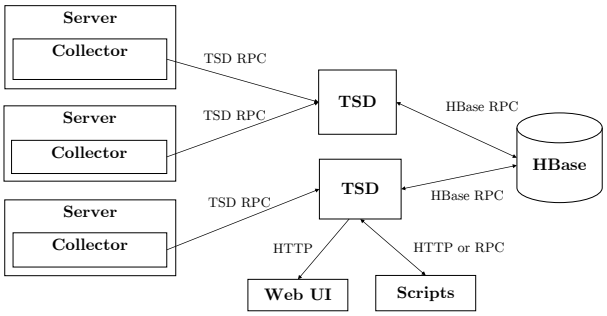
OpenTSDB [21] [22] [18] คือฐานข้อมูลที่เหมาะสำหรับการเก็บข้อมูลอนุกรมเวลาขนาดใหญ่บนฐานข้อมูล HBase ซึ่งสืบทอดความสามารถของ HBase ซึ่งอยู่ในระบบนิเวศของ Hadoop โดย HBase มีความสามารถในการจัดเก็บข้อมูลแบบกระจายตัวที่มีความสามารถในการขยายตัวได้ อีกทั้งยังมีความคงทนของข้อมูลสูงอีกด้วย OpenTSDB ถูกพัฒนาสำหรับจัดการงานทั่วๆไป ในการทำระบบตรวจการคอมพิวเตอร์ (Monitoring) โดยเก็บข้อมูลอนุกรมเวลา และรวบรวมหลาย ๆ อนุกรมเวลาจากระบบคอมพิวเตอร์ เช่น สถานะของระบบเครือข่าย ปริมาณงานของหน่วยประมวลผล ข้อมูลของระบบปฏิบัติการ เป็นต้น อีกทั้ง OpenTSDB ยังให้บริการ HTTP API สำหรับการบันทึกและอ่านค่าข้อมูลอนุกรมเวลา และข้อมูลอนุกรมเวลานั้นสามารถนำไปแสดงผลเป็นรูปแบบกราฟ โดย OpenTSDB มีส่วนเชื่อมต่อกับผู้ใช้พื้นฐานไว้สำหรับแสดงผลอนุกรมเวลา

หนึ่งในคุณลักษณะของ OpenTSDB คือการเก็บจุดข้อมูลทั้งหมด ซึ่งไม่เหมือนบางฐานข้อมูล ยกตัวอย่างเช่น RDDtool [23] มักจะถูกใช้ในงานประเภทการตรวจการระบบคอมพิวเตอร์ โดยฐานข้อมูลจะลดจำนวนจุดที่มีเวลาบันทึกนานมาแล้ว โดยการเอามารวมกัน เพื่อการลดพื้นที่การจัดเก็บ ดังนั้น OpenTSDB จึงเหมาะสำหรับจัดเก็บข้อมูลเซนเซอร์หรือข้อมูล IoT ซึ่งทุก ๆ จุดมีประโยชน์ต่อการนำไปวิเคราะห์ข้อมูล OpenTSDB ประกอบไปด้วยส่วนของการจัดเก็บข้อมูล และส่วนของการเก็บเกี่ยวข้อมูลใน

สภาพแวดล้อมหรือระบบที่ต้องการ โดยส่วนของการจัดเก็บข้อมูล ทำหน้าที่บันทึกและอ่านอนุกรมเวลา และส่วนของการเก็บเกี่ยวข้อมูล อีกทั้งฐานข้อมูล OpenTSDB ยังมี Tcollector [24] ซึ่งจะถูกนำไปติดตั้งในระบบหรือสภาพแวดล้อมที่ต้องการเก็บเกี่ยวข้อมูล เพื่อทำหน้าที่ส่งข้อมูลเข้าสู่ฐานข้อมูล

สถาปัตยกรรมของ OpenTSDB

OpenTSDB จะสร้างตัวทำงาน ซึ่งเรียกว่า *TSD (Time Series Daemon)* สำหรับจัดการงานของ OpenTSDB ซึ่งจะต้องมีอย่างน้อยหนึ่งตัวทำงานที่ทำงานอยู่และแต่ละตัวไม่ขึ้นต่อกัน รูปที่ 2.1 อธิบายถึงสถาปัตยกรรมของ OpenTSDB [25] ว่า TSD ติดต่อกับส่วนอื่นอย่างไร และเส้นทางเดินข้อมูลหลักๆ ถูกกำหนดโดยทิศทางของลูกศร TSD ทำหน้าที่เป็นตัวกลางในการเชื่อมต่อระหว่างการเขียนและการอ่านของ OpenTSDB กับ HBase ซึ่ง TSD สามารถทำงานเขียนข้อมูลไปยัง HBase ผ่านทางโปรโตคอล RPC (Remote Procedure Call) และ HTTP (Hypertext Transfer Protocol) เช่น ในการเก็บเกี่ยวข้อมูล (Collector) และสคริปต์ต่าง ๆ อีกทั้ง TSD สามารถอ่านข้อมูลจาก HBase โดยการสอบถามข้อมูลจาก HBase แต่ส่วนติดต่อผู้ใช้งาน TSD ใช้เฉพาะโปรโตคอล HTTP เท่านั้น



รูปที่ 2.1: สถาปัตยกรรมของ OpenTSDB

คำนิยามที่เกี่ยวข้องกับ OpenTSDB [26]

Metric คือชื่อของการวัดค่าอนุกรมเวลา ซึ่งเป็นข้อมูลประเภทข้อความ ใช้วัดหรือเก็บข้อมูลที่แปรเปลี่ยนไป เช่น อุณหภูมิ ความแรงของลม ความชื้น เป็นต้น

Timestamp คือเวลาที่บันทึก โดยจะอยู่ในรูปของเวลาแบบ Epoch [27] ซึ่งเป็นจำนวนวินาทีตั้งแต่ 00:00:00 (Coordinated Universal Time: UTC) วันที่ 1 มกราคม ค.ศ. 1970 โดยไม่ทดเวลาเป็นวินาที (Leap second)

Tag คือข้อมูลเพิ่มเติมของอนุกรมเวลาที่ถูกใช้ใน metric เดียวกัน ซึ่งจะเป็นคู่ของคีย์ (key) และค่าของคีย์ (value) แต่ละ metric จะต้องมีอย่างน้อย 1 tag โดยใน OpenTSDB ผู้ใช้สามารถใช้ประโยชน์จาก tag เพื่อการสอบถาม ซึ่งหมายความว่าผู้ใช้สามารถกรองอนุกรมเวลาที่ต้องการโดยการกำหนด tag

ตัวอย่างการใช้งาน tag เช่นสมมติว่าต้องการเก็บค่าของอุณหภูมิ ซึ่งแต่ละพื้นควรที่จะใช้ metric เดียวกัน นั่นคือ *temperature* และตัวอย่างของ metric และ tag แสดงได้ตามข้างล่างนี้

```
temperature country=Thailand province=Bangkok
temperature country=Thailand province=Phuket
```

จาก metric และ tag ข้างบน *temperature* เป็น metric
location=Bangkok เป็น tag
location เป็นคีย์
และ *Bangkok* เป็นค่าของคีย์

Data Point คือ จุดข้อมูลซึ่งประกอบไปด้วย *metric*, *tag*, *timestamp* และค่าที่บันทึก ยกตัวอย่างเช่น ถ้าวัดอุณหภูมิของสภาพแวดล้อมหนึ่งๆ ที่ภูเก็ต ประเทศไทย ณ เวลา 6 นาฬิกา (UTC) วันที่ 1 มกราคม ค.ศ. 2016 อุณหภูมิที่วัดได้มีค่า 30 องศาเซลเซียส และเวลาที่บันทึกเป็นรูปแบบ Epoch ซึ่งก็คือ 1451628000 ซึ่งเราสามารถเขียนอธิบายจุดข้อมูลได้ตามรูปแบบข้างล่างนี้

```
temperature country=Thailand province=Phuket 1451628000 30
```

Time Series คือ ลำดับของจุดข้อมูลซึ่งเรียงตาม *timestamp* หรือเวลาที่บันทึกข้อมูล

Downsampling คือฟังก์ชันการลดความละเอียดของอนุกรมเวลาเพื่อการวิเคราะห์ข้อมูล เช่น เพื่อดูภาพรวมของอนุกรมเวลา เป็นต้น หากต้องการรู้อุณหภูมิเฉลี่ยของแต่ละปีจากปี ค.ศ. 2013 ถึง 2016 ผู้ใช้สามารถเขียนคำสั่งการสอบถามอย่างง่ายได้ตามข้างล่างนี้

```
QUERY temperature FROM 1,February,2013 TO 1,February,2016
DOWNSAMPLING BY average 1 year
```

โดยผลลัพธ์จะได้ลำดับของ 3 จุดซึ่งเรียงตามเวลา โดยแต่ละจุดเป็นค่าเฉลี่ยของแต่ละปี

Interpolation คือ การสร้างค่าข้อมูลของจุดที่ไม่มีข้อมูลระหว่าง 2 จุดใด ๆ ในอนุกรมเวลา การทำ interpolation นั้นมีหลายวิธี แต่ OpenTSDB รุ่นที่ 2.3 รองรับเฉพาะการประมาณค่าแบบเชิงเส้น (linear interpolation) เท่านั้น [28]

2.2.2 ความรู้พื้นฐานของหน่วยความจำหลักและการแคช

ระบบคอมพิวเตอร์โดยส่วนใหญ่สามารถใช้งานหน่วยความจำได้หลายประเภท ตารางที่ 2.1 แสดงการเปรียบเทียบความเร็วและราคาต่อความจุ ของหน่วยเก็บข้อมูลแต่ละลำดับชั้น ได้แก่ ฮาร์ดดิสก์ โซลิดสเตตไดรฟ์ หน่วยความจำหลัก แคชและรีจิสเตอร์ [29] [30] โดยสรุปแล้ว หน่วยเก็บข้อมูลที่มีลำดับสูงซึ่งในที่นี้อยู่ใกล้ CPU มากกว่าจะทำงานเร็วกว่าหน่วยเก็บข้อมูลที่ลำดับต่ำกว่า ยกตัวอย่างเช่น ฮาร์ดดิสจะทำงานช้ากว่าหน่วยความจำหลัก เพราะฉะนั้นเทคนิคการใช้แคชบนระบบคอมพิวเตอร์ถูกนำมาประยุกต์ใช้อย่างแพร่หลายในการปรับปรุงความเร็วโดยรวมของการอ่านและเขียนข้อมูล เทคนิคการใช้แคชบนระบบคอมพิวเตอร์จึงถูกนำมาประยุกต์ใช้อย่างแพร่หลาย

ตารางที่ 2.1: การเปรียบเทียบความจุข้อมูลของแต่ละลำดับชั้น

ชนิด	ความจุ	ความเร็ว	ราคาต่อความจุ
ฮาร์ดดิส (Hard Disk)	★★★★★	★	★
โซลิดสเตตไดรฟ์ (Solid State Drive)	★★★★	★★	★★
หน่วยความจำหลัก (Main Memory)	★★★	★★★	★★★
แคช (Cache)	★★	★★★★	★★★★
รีจิสเตอร์(Register)	★	★★★★★	★★★★★

แคชคือหน่วยเก็บข้อมูลชั่วคราวซึ่งสามารถจัดเก็บข้อมูลเพื่อที่จะเพิ่มความเร็วในการอ่านและเขียนข้อมูล โดยทั่วไปแล้ว ความเร็วในการอ่านและเขียนของแคชจะเร็วกว่าหน่วยเก็บข้อมูลหลัก การใช้งานแคชสามารถประยุกต์ได้หลากหลาย เช่น ในการสอบถามข้อมูลจากฐานข้อมูล ข้อมูลที่ถูกประมวลผล ตัวอย่างรูปภาพ หรือ การแคชหน้าเว็บ เป็นต้น ข้อมูลที่จะถูกเก็บในแคชนั้นต้องการการเข้าถึงโดย CPU มาก่อนอย่างน้อยหนึ่งครั้ง โดยปกติแล้ว เมื่อข้อมูลถูกเข้าถึง กลไกพื้นฐานของแคชจะทำงาน ถ้ามีข้อมูลนั้นอยู่ในแคชอยู่แล้ว ข้อมูลนั้นจะส่งออกมาจากแคชแทนที่จะอ่านจากหน่วยเก็บข้อมูลปกติ หากยังไม่มีข้อมูลนั้นอยู่ในแคช ระบบก็จะเก็บข้อมูลนั้นลงแคช และระบบแคชสามารถที่จะคัดข้อมูลที่มีความสำคัญน้อยออกไปจากแคชได้ ในกรณีที่เนื้อที่จัดเก็บข้อมูลในแคชไม่เพียงพอหรือเป็นไปตามเงื่อนไขของกลไกของระบบการแทนที่แคช (Cache Replacement)

Memcached เป็นระบบการแคชข้อมูลแบบจับคู่คีย์และค่าบนหน่วยความจำหลัก (in-memory key-value cache system) [31] [14] ซึ่งใช้กลไกแบบ Least Recently Used (LRU) เพื่อกำจัดข้อมูลที่ถูกแคชโดยเรียงตามลำดับเวลาที่เคยถูกเข้าถึงนานที่สุด ยิ่งไปกว่านั้นระบบแคชนี้ยังสามารถสนับสนุนการเก็บข้อมูลบนหน่วยความจำหลัก แบบกระจายตัวโดยใช้หลายเครื่องแม่ข่ายเพื่อที่จะใช้งานพื้นที่การเก็บข้อมูลแคชร่วมกัน โดย Memcached จะมีไลบรารีทางฝั่งลูกข่ายสำหรับจัดการข้อมูลที่ต้องการจัดเก็บบนเครื่องแม่ข่ายเครื่องต่าง ๆ บนระบบหน่วยความจำแบบกระจายตัว ซึ่งไลบรารีทางฝั่งลูกข่ายนั้นมีหลายภาษา เช่น Python PHP Java C/C++ เป็นต้น และยังสามารถใช้ได้ทั้งการเชื่อมต่อแบบ UDP (User Datagram Protocol) และ TCP (Transmission Control Protocol) อีกด้วย

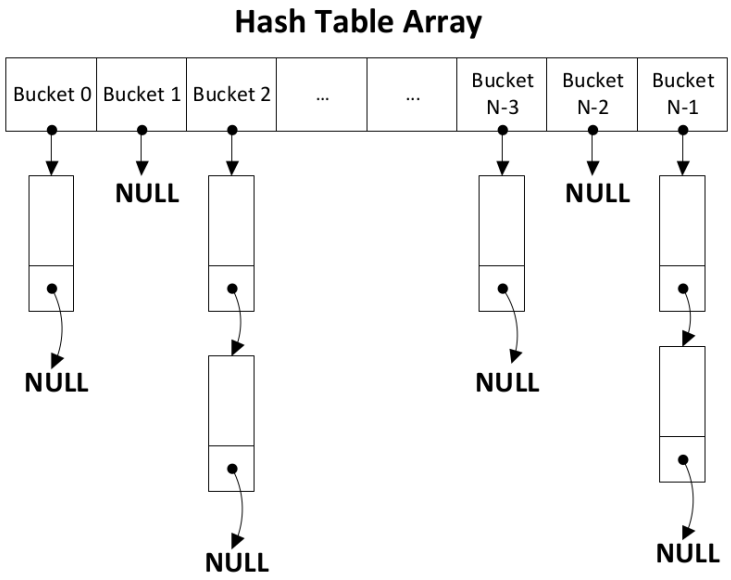
ระบบเก็บข้อมูลแบบจับคู่คีย์และค่าบนหน่วยความจำหลักอีกระบบที่ได้รับความนิยมคือ ฐานข้อมูล Redis [14] โดยฐานข้อมูล Redis สามารถนำไปใช้เป็นระบบแคช อีกทั้งยังสามารถจัดเก็บข้อมูลได้หลายประเภท เช่น strings, hashes, lists, sets และ sorted sets หรือแม้แต่การสอบถามแบบกำหนดขอบเขตการสอบถาม และสามารถกำจัดแคชแบบ LRU ถึงแม้ว่า Redis มีความสามารถหลากหลาย แต่ก็ติดตั้งและใช้งานยากกว่า รวมถึง การขยายตัวเพื่อรองรับผู้ใช้งานจำนวนมากยังไม่ดี

ใน วิทยานิพนธ์ นี้ ผู้วิจัยพิจารณา ว่า Memcached เหมาะสมกว่า Redis โดยที่ Memcached สามารถรองรับการเข้าถึงของผู้ใช้งานหลายคนได้พร้อมกัน และมีความสามารถในการขยายตัวได้ดี ทั้งยังมีขนาดเล็กทำให้จัดการในแง่ของการออกแบบเพื่อทดสอบแนวคิดของระบบแคชสำหรับฐานข้อมูลอนุกรมเวลาได้ง่าย อีกทั้ง Memcached มีผู้ใช้งานมากมาย เช่น ระบบโครงสร้างของ Facebook [32] และระบบของ Twitter [33]

2.2.3 โครงสร้างข้อมูลของ Memcached

โครงสร้างข้อมูลของ Memcached ถูกออกแบบเพื่อสนับสนุนการใช้ระบบการแคชแบบจับคู่คีย์และค่าข้อมูล และมี memory slab classes สำหรับการจัดเก็บข้อมูล ที่แตกต่างขนาดกัน อีกทั้งยังมีกลไกการกำจัดแคชออกจากระบบแบบ LRU Memcached ประกอบไปด้วย 4 โครงสร้างข้อมูลหลัก ดังต่อไปนี้

1. **Hash-table array** คือ แอเรย์แบบตาราง hash สำหรับการเข้าถึงข้อมูลในแคชแบบการจับคู่คีย์และค่าข้อมูล [31] ดังในรูปที่ 2.2

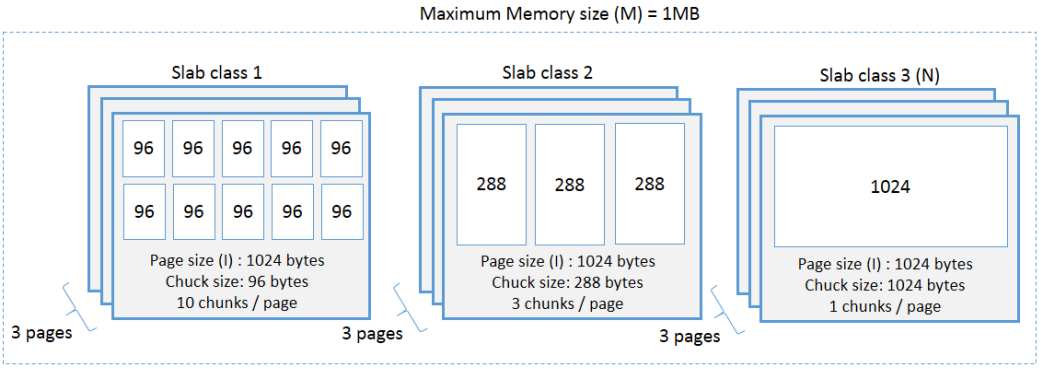


รูปที่ 2.2: โครงสร้างข้อมูลอาร์เรย์แบบตาราง hash

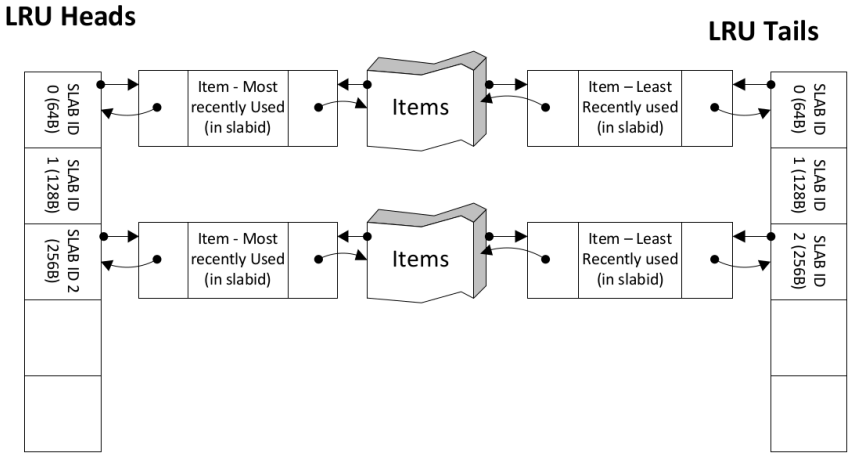
2. **Slab memory management** คือ ตัวจัดการ slab เพื่อการจองพื้นที่สำหรับจัดเก็บข้อมูลที่มีขนาดแตกต่างกัน รูปที่ 2.3 แสดงให้เห็นถึงตัวจัดการ slab ว่าทำการจองพื้นที่อย่างไร โดยมีค่าที่ต่างกันตามพารามิเตอร์ที่ตั้งค่า โดยมี Maximum memory size (M) หรือ ขนาดของหน่วยความจำหลักสูงสุดที่ใช้เป็นแคช ซึ่งมีค่า 1 เมกะไบต์ Chunk growth factor (F) มีค่าเท่ากับ 3 และ maximum page size (I) ซึ่งมีค่า 1024 ไบต์ จากพารามิเตอร์นี้ Memcached จะสร้างต้นแบบมา 3 slab classes เพื่อสำหรับเก็บข้อมูลที่แตกต่างขนาดกัน โดยภายในแต่ละ slab class สามารถเก็บข้อมูลได้หลาย page ซึ่งแต่ละ page สามารถเก็บข้อมูลได้เท่ากับ page size (I) ซึ่งมีขนาดสูงสุดคือ 1024 ไบต์

   เริ่มแรก 1 page และเมื่อเนื้อที่ของ page นั้นเต็ม page ใหม่ก็จะถูกสร้างออกมา ซึ่งใน page แรกสามารถบรรจุได้ 10 chunk ซึ่งขนาดของ chunk จะแตกต่างกันขึ้นอยู่กับลำดับของ slab class และ chunk growth factor (F)

3. **LRU management** คือ ตัวจัดการ LRU เพื่อที่จะกำจัดแคชที่หมดอายุออกด้วยกลไกแบบ LRU โดยโครงสร้างข้อมูลของ LRU ซึ่งแต่ละ slab class จะมี LRU list เป็นของตัวเอง [31] สามารถแสดงได้ดังรูปที่ 2.4



รูปที่ 2.3: ตัวอย่างการจองพื้นที่ของ slab ของ Memcached



รูปที่ 2.4: โครงสร้างข้อมูลของ LRU เพื่อจัดการแคชออกจากระบบ

4. **Cache item** คือ ข้อมูลที่ถูกแคช ประกอบไปด้วย คีย์และค่าข้อมูล โดยจะเป็นโครงสร้างข้อมูลลิงค์ลิสต์ของพอยน์เตอร์แบบทิศทางเดียว (single linked list pointers) สำหรับเก็บตารางแฮช และเป็นลิงค์ลิสต์ของพอยน์เตอร์แบบสองทิศทาง (double linked-list pointers) สำหรับตัวจัดการ LRU

การใช้งาน Memcached สามารถใช้งานผ่านคำสั่งขั้นพื้นฐานดังต่อไปนี้

- **GET** เพื่ออ่านค่าข้อมูลในแคชจากคีย์ที่กำหนด
- **STORE** เพื่อจัดเก็บข้อมูลในแคชจากคีย์ที่กำหนด
- **DELETE** เพื่อลบข้อมูลในแคชจากคีย์ที่กำหนด

บทที่ 3

วิธีการวิจัย

3.1 วิธีดำเนินการ

เครื่องมือแสดงผลอนุกรมเวลาส่วนใหญ่จะถูกเชื่อมต่อกับฐานข้อมูลอนุกรมเวลา ยกตัวอย่าง เช่น ผู้ใช้ต้องการเลื่อนหน้าต่างของการแสดงผลอนุกรมเวลาเพื่อจะดูข้อมูลบางช่วง การซูมเข้าหรือออก การเปลี่ยนค่า downsampling หรือปรับพารามิเตอร์อื่นๆ เป็นต้น การดึงข้อมูลจาก OpenTSDB มีส่วนประกอบ 2 ส่วนได้แก่ การรับข้อมูลดิบจากฐานข้อมูลและการประมวลผลหลังได้รับข้อมูล เพื่อใช้ในการแสดงผลบนเครื่องมือแสดงผล แต่ละครั้งที่ผู้ใช้เลื่อนหน้าต่างของช่วงอนุกรมเวลา ซึ่งหมายถึงคำร้องขอการสอบถามใหม่จะถูกสร้างขึ้นมา ถึงแม้ว่าคำร้องขอนั้นจะระบุเป็นอนุกรมเวลาเดียวกันกับที่เคยค้นหามาแล้วก็ตาม ดังนั้นจึงทำให้ OpenTSDB ต้องจัดการประมวลผลข้อมูลดิบในช่วงเดิม หรือที่เหมือนกันในบางช่วงเวลา แต่การประมวลผลหลังได้รับผลลัพธ์ส่งผลให้ค่าผลลัพธ์ที่จะคืนค่าให้กับผู้ใช้จะแตกต่างกันขึ้นอยู่พารามิเตอร์ของผู้ใช้ ถึงแม้ว่าจะเป็นอนุกรมเวลาเดียวกัน โดยทั่วไปแล้ว ในกรณีที่มีงบประมาณจำกัด ผู้ใช้มักจะใช้ฮาร์ดดิสก์เป็นตัวบันทึกข้อมูล ซึ่งมีความเร็วในการอ่านและเขียนช้ามากเมื่อเทียบกับหน่วยความจำหลัก เพราะฉะนั้น วิทยานิพนธ์ฉบับนี้จึงนำเสนอการออกแบบกลไกแคชของ OpenTSDB การออกแบบมีวัตถุประสงค์เพื่อลดระยะเวลาในการค้นหาของส่วนของอนุกรมเวลา

จากโครงสร้างการออกแบบฐานข้อมูล opentsdb การดึงข้อมูลจากฐานข้อมูล HBase เป็นเวลาส่วนใหญ่ที่ผู้วิจัยวิเคราะห์ว่าเป็นส่วนที่ทำให้ให้ฐานข้อมูลทำงานช้าที่สุด จากการทดสอบ opentsdb บน docker [34] และ opentsdb รุ่นที่ 2.2.0 โดยใช้ฐานข้อมูล HBase รุ่นที่ 1.1.5 บนโครงสร้างแบบโหนดเดี่ยวบน docker 1.11.2 โดยใช้การตั้งค่า docker [35] โดยใช้ Ubuntu desktop 16.04 CPU AMD A10-5800K 4 คอร์ หน่วยความจำหลัก 16 กิกะไบต์, ฮาร์ดดิสก์ WD WD20EZRX ที่มีชนิดการเชื่อมต่อแบบ SATA ที่ความเร็ว 6 กิกะบิต/วินาที ซึ่งมีความจุ 2 เทระไบต์

จากการวัดผลการทดลองหาค่าของ metric ในการสอบถามข้อมูลจาก OpenTSDB [36] ซึ่งเกิดจากการใช้โปรโตคอล HTTP โดยวัดว่าระยะเวลาในการตอบสนองของแต่ละโมดูลนั้นใช้เวลาเป็นเท่าใด จากการตั้งคำถามเพื่อทดสอบสมมติฐานว่าฐานข้อมูล OpenTSDB มีการกลไกแคชหรือไม่ โดย 3 metrics ที่เกี่ยวข้องกับการวัดมีหน่วยเป็นมิลลิวินาทีดังต่อไปนี้

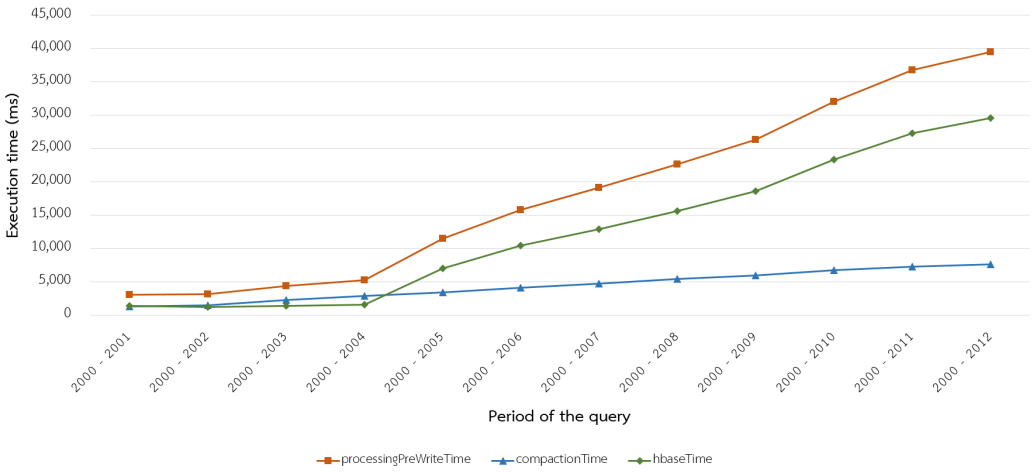
- *hbaseTime* เวลาที่ใช้ในการดึงข้อมูลจากฐานข้อมูล HBase
- *compactionTime* เวลาสะสมที่ใช้ในการประมวลผลแต่ละแถวของฐานข้อมูล HBase จากข้อมูลที่เก็บการแบบหลายคอลัมน์ทำให้รวมกันมาเป็นคอลัมม์เดียว
- *processingPreWriteTime* เวลาที่ใช้รวมทั้งหมดโดยไม่รวมเวลาที่ใช้ในการส่งข้อมูลผ่านเครือข่าย

การทดลอง ผลการทดลองเผยให้เห็นว่า OpenTSDB ไม่มีกลไกแคชเมื่อผู้ใช้ร้องขอการสอบถามที่ช่วงของการสอบถามซ้ำกัน ซึ่งการทดลองแสดงให้เห็นถึงสถานการณ์ที่พบกับเหตุกาณ์ที่มีช่วงเวลาของการสอบถามซ้ำซ้อนกับที่เคยผ่านมาแล้วในอดีต ในการทดลองนี้ ข้อมูลที่บรรจุในฐานข้อมูล OpenTSDB เป็นข้อมูลอนุกรมเวลาเดียวซึ่งประกอบไปด้วย 100 ล้านจุด ซึ่งบันทึกข้อมูลในช่วงเวลาประมาณ 12 ปี โดยใช้การทดสอบผ่านการสอบถาม 12 ครั้ง เพื่อที่จะหาค่าเฉลี่ยแต่ละปี โดยพารามิเตอร์ในการสอบถาม แสดงดังในตารางที่ 3.1.

ตารางที่ 3.1: พารามิเตอร์การสอบถาม

ลำดับการสอบถาม	เวลาตั้งต้น	เวลาสิ้นสุด	downsampling	จำนวนจุดที่นำมาวาดกราฟ
1	2000	2001	ค่าเฉลี่ย 1 ปี	1
2	2000	2002	ค่าเฉลี่ย 1 ปี	2
3	2000	2003	ค่าเฉลี่ย 1 ปี	3
		...		
12	2000	2012	ค่าเฉลี่ย 1 ปี	12

เวลาที่ใช้ในการสอบถามในตารางที่ 3.1 นำมาแสดงในรูปที่ 3.1 ในแกน Y หมายถึงเวลาที่ใช้ซึ่งมีหน่วยเป็นมิลลิวินาที และแกน X แสดงถึงช่วงของการสอบถาม



รูปที่ 3.1: การเปรียบเทียบระยะเวลาในการประมวลผลระหว่าง *hbaseTime*, *compactionTime* และ *processingPreWriteTime* เมื่อเพิ่มช่วงเวลาของการสอบถามให้กว้างขึ้นเรื่อยๆ

จากกราฟในรูปที่ 3.1 เวลา *processingPreWriteTime* คือเวลาที่ใกล้เคียงกับการทำ compaction หรือ *compactionTime* และการประมวลผลของ HBase หรือ *hbaseTime* รวมกัน ส่วน metric อื่นๆนั้นไม่มีนัยสำคัญเมื่อเปรียบเทียบกับ 3 metrics นี้ ดังนั้น *hbaseTime* ใช้เวลาส่วนใหญ่เมื่อเทียบกับเวลาที่ใช้ทั้งหมด (*processingPreWriteTime*) หรือประมาณ 74.19% ที่การสอบถามครั้งที่ 12 จากผลการทดลองพบว่า OpenTSDB ไม่มีกลไกแคช ซึ่งฐานข้อมูล OpenTSDB ยังคงอ่านข้อมูลทั้งหมดจากฐานข้อมูล HBase ทุกครั้ง ถึงแม้ว่าช่วงของการสอบถามมีการซ้ำซ้อนกับช่วงของการสอบถามที่ผ่านมาแล้วในอดีต ยกตัวอย่างเช่นการสอบถามครั้งที่ 11 คำสั่งการสอบถามต้องการร้องขอข้อมูลที่ช่วงเวลาปี 2000 ถึง 2011 และ

การสอบถามครั้งที่ 12 ที่ช่วงเวลาปี 2000 ถึงปี 2012 ผลการทดลองพบว่าในครั้งที่ 12 ยังมีการอ่านข้อมูลจากฐานข้อมูล HBase ซ้ำ ซึ่งรวมถึงช่วงเวลาในการสอบถามข้อมูลของครั้งที่ 11 ด้วย ดังนั้น วิธีการดึงข้อมูลแบบนี้จึงไม่มีประสิทธิภาพเท่าที่ควร ซึ่งเป็นเหตุให้วิทยานิพนธ์นี้เสนอไกแคชสำหรับฐานข้อมูล OpenTSDB

จากการที่ผู้วิจัยได้ทำการศึกษาเบื้องต้นเกี่ยวกับการทำงานของ OpenTSDB พบว่า OpenTSDB ไม่มีกลไกแคชในการสอบถามอนุกรมเวลาจากฐานข้อมูล เมื่อช่วงของการสอบถามปัจจุบันซ้ำซ้อนกับการสอบถามที่เคยค้นหาไว้แล้วในอดีต ดังนั้น ระยะเวลาการตอบสนองจึงไม่ได้ถูกทำให้เร็วขึ้น เพราะว่าทุกๆ ครั้งเมื่อคำร้องขอถูกสร้างขึ้นมาระบบจะหาข้อมูลจากฐานข้อมูลที่อยู่บนฮาร์ดดิสก์เสมอ ยิ่งไปกว่านั้นตำแหน่งการวางแคชในระบบ OpenTSDB มีผลอย่างมีนัยสำคัญสำหรับการลดระยะเวลาในการสอบถาม ในกรณีที่ ผู้ใช้ปรับพารามิเตอร์บนข้อมูลอนุกรมเวลาเดียวกัน เมื่อมีผู้ใช้งานหลายคนเข้าใช้งานเครื่องมือแสดงผล สมมติว่าข้อมูลดิบถูกแคชเอาไว้แล้ว ระยะเวลาในการสอบถามก็จะถูกลดลง ส่งผลต่อประสิทธิภาพการทำงานของทั้งระบบ

เพื่อที่จะจัดการข้อมูลดิบก่อนที่ระบบนำจะข้อมูลนี้ไปประมวลผลต่อ ดังนั้นตำแหน่งที่จะแคชจะอยู่บนฝั่งของเซิร์ฟเวอร์เพื่อที่จัดการข้อมูลดิบ แนวการแคชบนข้อมูลบนฝั่งเซิร์ฟเวอร์ สามารถทำได้สองแบบ คือการทำแคชภายในและแคชภายนอกของตัวโปรแกรม ประโยชน์ของการแคชภายในฐานข้อมูล คือจะไม่มีส่วนที่ทำให้เสียเวลาเพิ่มจากการส่งข้อมูลผ่านเครือข่าย ในขณะที่การแคชภายนอกนั้นจะต้องพิจารณาประเด็นนี้ในการออกแบบด้วย การออกแบบและพัฒนาการแคชภายในระบบฐานข้อมูลนั้นจะต้องคำนึงถึงประเด็นของการจัดการแคชและพื้นที่ รวมถึงข้อกำหนดในการนำข้อมูลในแคชออก แต่การทำแคชภายนอกนั้นสามารถประยุกต์ใช้ได้กับเครื่องมือที่มีอยู่ในปัจจุบัน ระบบจัดการแคชที่มีอยู่และได้รับความนิยมสูงมากในปัจจุบันคือ Memcached อีกทั้งยังเป็นระบบแบบกระจายตัว ผู้วิจัยจึงได้นำ Memcached มาเป็นส่วนหนึ่งของการออกแบบของกลไกแคชในวิทยานิพนธ์นี้เพื่อที่จะเสนอกลไกแคชสำหรับจัดการข้อมูลแบบอนุกรมเวลาโดยใช้ประโยชน์จาก Memcached ได้หลายทาง เช่น การเพิ่มความจุของแคชจากระบบกระจายตัว หรือ กลไกการเอาแคชที่ไม่ต้องการออกจากระบบ หรือแม้แต่การจองพื้นที่และการจัดการหน่วยความจำ โดยไม่ได้มุ่งประเด็นไปที่การจัดการแคชหรือเงื่อนไขการนำแคชที่ไม่ต้องการออกจากระบบ แต่สนใจเฉพาะแนวทางในการจัดการกับข้อมูลอนุกรมเวลาสำหรับการแคช

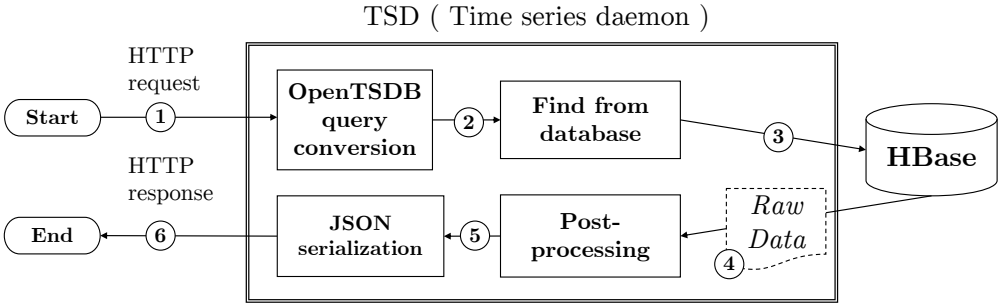
โดยทั่วไป แล้วข้อมูลที่ถูกแคชจะได้สถานะการตอบกลับจากระบบแคชอยู่ 2 สถานะ คือ พบในแคช (cache hit) และไม่พบในแคช (cache miss) อย่างไรก็ตาม ในกรณีของระบบฐานข้อมูลอนุกรมเวลาขนาดใหญ่นั้น การแคชข้อมูลทั้งหมดของทั้งอนุกรมเวลาอาจจะทำให้ใช้ปริมาณหน่วยความจำหลักมาก และต้องใช้เวลาเพิ่มขึ้นในการเก็บข้อมูลบางช่วงที่ไม่เกี่ยวข้องกับการสอบถาม โดยวิทยานิพนธ์ชิ้นนี้จะเก็บข้อมูลบางช่วงเวลาที่ผู้ใช้เคยค้นหาไว้แล้ว ซึ่งจะเรียกว่า การพบข้อมูลในแคชบางส่วน (partial cache hit)

วิทยานิพนธ์ฉบับนี้ปรับปรุงประสิทธิภาพของส่วนของการดึงข้อมูลหลักของ OpenTSDB จากสถาปัตยกรรมของ OpenTSDB ในรูปที่ 2.1 ทุกๆ คำร้องจะต้องดำเนินการโดย TSDs เท่านั้น และในแต่ละ TSD ทำหน้าที่ดำเนินการคำร้องจากผู้ใช้ ในหัวข้อนี้ได้แบ่งออกเป็นสองส่วน ส่วนของการดึงข้อมูลของ OpenTSDB แบบต้นฉบับแสดงรายละเอียดดังรูปที่ 3.2 และส่วนของการออกแบบกลไกแคชสำหรับ OpenTSDB เพื่อปรับปรุงให้การดึงข้อมูลของ OpenTSDB แบบต้นฉบับมีกลไกแคชสำหรับอนุกรมเวลาได้ดังแสดงในรูปที่ 3.3

3.1.1 ขั้นตอนวิธีการสอบถามข้อมูลของ OpenTSDB

สถาปัตยกรรม OpenTSDB นั้นจะสร้างและควบคุม TSDs ในการจัดการการเขียนและอ่านข้อมูล ซึ่งได้แสดงไว้แล้วในรูปที่ 2.1 ในกรณีของการอ่านข้อมูลจาก OpenTSDB เมื่อคำร้องจากผู้ใช้มาถึงที่ TSD ก็จะไปอ่านข้อมูลอนุกรมเวลาจากฐานข้อมูล HBase ในสถาปัตยกรรมการออกแบบ OpenTSDB ในส่วนของการอ่านข้อมูลมีหลายโมดูลที่เชื่อมโยงกันเพื่อจัดการคำร้อง โดยงานวิจัยนี้สนใจเฉพาะส่วนของการสอบถามข้อมูลของ OpenTSDB

งานวิจัยนี้เน้นเฉพาะส่วนของการจัดการข้อมูลดิบภายในกระบวนการสอบถามข้อมูลของ OpenTSDB เท่านั้น ซึ่งรูปที่ 3.2 แสดงให้เห็นว่ากระบวนการสอบถามข้อมูลจะอยู่ในภายใน TSD ซึ่งโดยปกติแล้ว TSD จะรับคำร้องเป็นลักษณะของคำร้อง HTTP แล้วจะแปลงให้อยู่ในรูปแบบของ OpenTSDB โดยโมดูล *OpenTSDB query conversion* และส่งคำสั่งการสอบถามไปทำงานที่โมดูล *Find from database* ซึ่งจะไปค้นหาข้อมูลจากฐานข้อมูล HBase และเมื่อข้อมูลทั้งหมดมาถึง ระบบจะนำข้อมูลอนุกรมเวลาดิบส่งไปยังโมดูล *Post-processing* เพื่อทำการประมวลผลข้อมูลตามคำร้องของผู้ใช้ เช่น การทำ downsampling หรือการทำ aggregation จากนั้นข้อมูลนั้นจะถูกแปลงไปรูปแบบ JSON ผ่าน *JSON serialization* เพื่อคืนค่าอนุกรมเวลาที่ผู้ใช้ต้องการ เพราะฉะนั้นโมดูลที่เกี่ยวข้องกับงานวิทยานิพนธ์นี้โดยตรงคือ *Find from database* เพื่ออ่านข้อมูลดิบจากฐานข้อมูล HBase

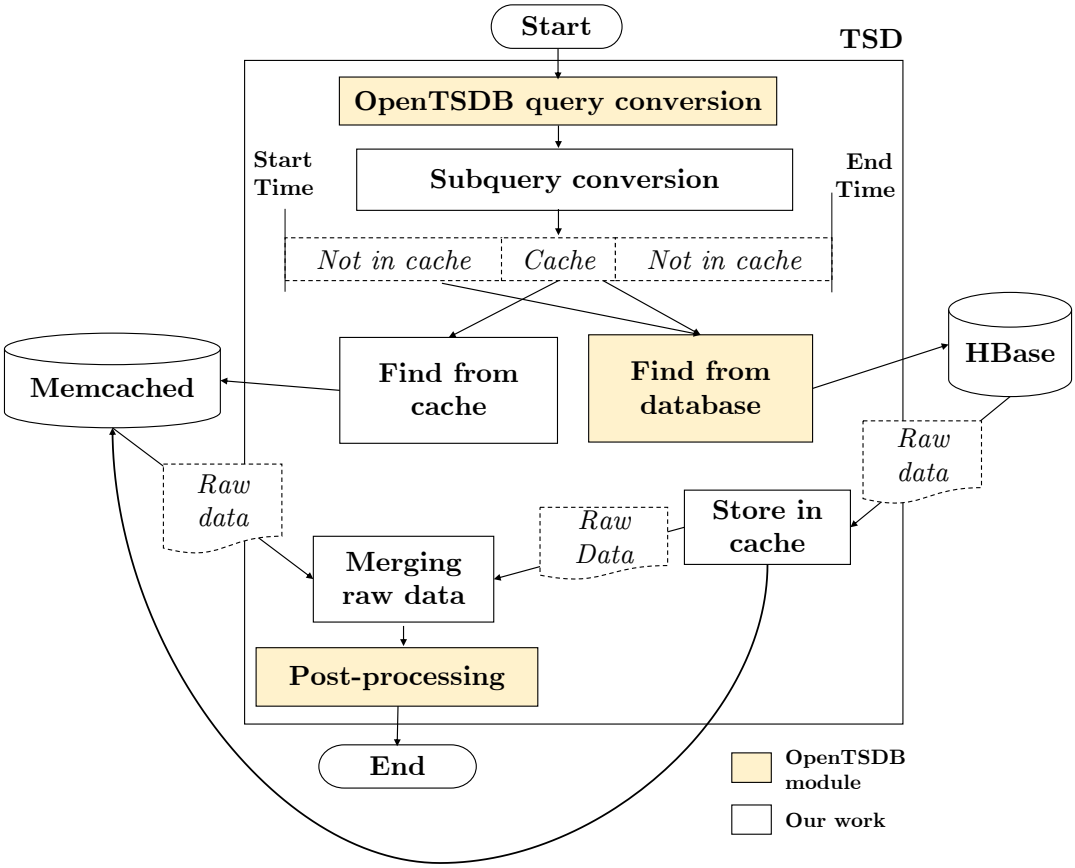


รูปที่ 3.2: ขั้นตอนวิธีการสอบถามข้อมูลของ OpenTSDB

3.1.2 การออกแบบกลไกแคชสำหรับ OpenTSDB

วิทยานิพนธ์ฉบับนี้เสนอกลไกการแคชสำหรับ OpenTSDB เพื่อลดระยะเวลาในการสอบถามข้อมูลอนุกรมเวลา ในกรณีที่มีข้อมูลบางช่วงของการสอบถามนั้นซ้ำกับช่วงของการสอบถามที่เคยเรียกใช้งานมาแล้ว โดยผู้วิจัยได้ออกแบบสถาปัตยกรรมเพื่อปรับปรุงการจัดการฐานข้อมูล OpenTSDB ดังแสดงในรูปที่ 3.3 โดยจะแทนที่การทำงานของโมดูล *Find from Database* ซึ่งเป็นส่วนหนึ่งของโมดูลที่สำคัญสำหรับสอบถามข้อมูลของ OpenTSDB จากรูปที่ 3.2 เพื่อลดระยะเวลาการสอบถามโดยมีการจัดเก็บข้อมูลลงบน Memcached และเรียกข้อมูลที่เคยเข้าถึงไว้แล้ว

หลังจากที่โมดูล *OpenTSDB query conversion* ซึ่งทำหน้าที่ในการแปลงคำร้องของผู้ใช้ให้อยู่ในรูปแบบของการสอบถาม OpenTSDB ดำเนินการเสร็จสิ้นแล้ว กลไกแคชจะรวมกลุ่มของข้อมูลอนุกรมเวลาแต่ละจุดข้อมูลกลายเป็นหนึ่ง fragment โดยใช้ *Cache Index* โดยกลไกแคชเริ่มต้นจากโมดูล *Subquery Conversion* ซึ่งจะมีหน้าที่ตัดสินใจว่าส่วนใดของช่วงทั้งหมดของอนุกรมเวลาที่ผู้ใช้ร้องขอนั้น มีช่วงไหนที่เคยเรียกแล้วหรือไม่ จากนั้นสร้างการสอบถามย่อย (*subquery*) ซึ่งแต่ละ subquery นั้นจะรับผิดชอบหน้าที่ที่แตกต่างกันสองอย่างคือดึงข้อมูลจากฐานข้อมูล (Find from database) และดึงข้อมูลจากแคช (Find from cache) โดยโมดูล *Find from database* จะรับข้อมูลดิบจาก HBase หากช่วงของการทำ subquery นั้นไม่ได้มีข้อมูลจัดเก็บอยู่ในแคช จากนั้นโมดูล *Store in cache* จะนำข้อมูลดิบไปจัดเก็บในแคช หากเป็นช่วงข้อมูลของ subquery ที่อยู่ในแคช โมดูล *Find from cache* จะทำหน้าที่ดึงข้อมูลที่อยู่ในแคชออกมา แต่หากผิดพลาดให้เรียกโมดูล *Find from database* มาทำงานแทน จากนั้นจะเป็นการรวมข้อมูลโดยโมดูล *Merging raw data* และส่งข้อมูลดิบที่รวมกันเสร็จแล้วไปให้โมดูล *Post-processing* ซึ่งอยู่ในขั้นตอนการสอบถามข้อมูลของ OpenTSDB ดังรูปที่ 3.2 เพื่อทำการประมวลผลตามคำร้องของผู้ใช้ต่อไป



รูปที่ 3.3: สถาปัตยกรรมของระบบแคชที่ออกแบบ

3.1.2.1 Cache Indexing

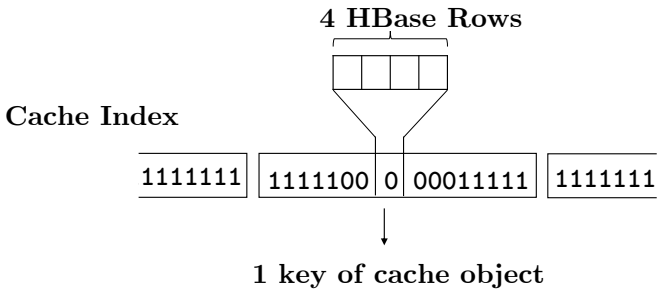
การใช้งานระบบแคชแบบภายนอกจะทำให้ระยะเวลาในการร้องขอข้อมูลจากแคชสูงขึ้น เนื่องจากมีการส่งข้อมูลผ่านเครือข่ายโดยเฉพาะในกรณีที่จำนวนข้อมูลที่จะเก็บในแคชมีขนาดใหญ่ และด้วยคุณลักษณะของข้อมูลที่เป็นอนุกรมเวลามักจะมีขนาดใหญ่มากจึงมักจะจัดเก็บในฐานข้อมูลอนุกรมเวลาขนาดใหญ่ ในโครงสร้างของข้อมูลของ OpenTSDB นั้น จากการคำนวณเบื้องต้นจากโครงสร้างฐานข้อมูลที่ออกแบบบน HBase สำหรับใช้ใน OpenTSDB หนึ่งจุดข้อมูลมีขนาดโดยประมาณ 22 ไบต์ ซึ่งมีขนาดเล็กมาก [37] สมมติถ้าเราต้องการจัดเก็บจุดข้อมูลหนึ่งจุดต่อหนึ่งวัตถุแคช ก็จะส่งผลให้ระยะเวลาในการรับส่งข้อมูลผ่านระบบเครือข่ายเพิ่มขึ้น OpenTSDB ไม่ได้เน้นที่จะจัดการข้อมูลครั้งละหนึ่งจุดข้อมูล แต่จุดข้อมูลนั้นจะถูกรวมกันเป็นหนึ่งแถวของตาราง HBase ซึ่งข้อมูลที่บันทึกลงไปในแต่ละแถวจะถูกแบ่งโดยใช้เวลาแต่ละจุดข้อมูลในการแบ่ง ซึ่งทุกๆ หนึ่งชั่วโมง จะรวมเป็นหนึ่งแถวของ HBase โดยขอบเขตของเวลาหนึ่งชั่วโมงเป็นค่าเริ่มต้นเรียกว่า HBaseRowPeriod

แต่อย่างไรก็ตาม เมื่อข้อมูลมีขนาดใหญ่ จำนวนแถวของ HBase ก็จะมีจำนวนมากตามไปด้วย ซึ่งก็จะส่งผลถึงระยะเวลาการรับส่งข้อมูลผ่านเครือข่ายที่มากขึ้นด้วย เพราะฉะนั้น การรวมกลุ่มของแถวของ HBase เป็นหนึ่ง fragment ซึ่งจำนวนแถวที่นำมารวมเรียกว่า chunk size (CS) พิจารณาในกรณีของ CS มีขนาดเล็ก จะส่งผลให้จำนวน fragment มีปริมาณมากด้วยเช่นกัน หรืออธิบายในอีกทางหนึ่ง จำนวน framgent คือจำนวนวัตถุที่จะถูกนำไปจัดเก็บลงบนแคช และจำนวนวัตถุที่จะนำไปลงแคชนั้นส่งผลต่อเวลาในการสอบถามข้อมูล โดยการอ้างถึง fragment ทำได้โดยผ่าน fragment order (FO) ซึ่งสามารถคำนวณได้ดังในสมการที่ 3.1 โดยที่มี T เป็นเวลาตั้งต้น และใช้หน่วยเวลาเป็นมิลลิวินาที

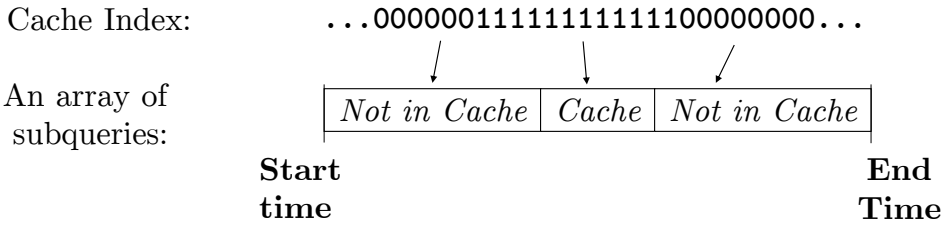
$$EndFO = \lfloor \frac{T}{RS \times HBaseRowPeriod} \rfloor \tag{3.1}$$

โดยปกติแล้วการใช้งานแคชทั่วไปจะมีการอ่านข้อมูลพลาดก่อนในครั้งแรกเนื่องจากเป็นครั้งแรกที่เรียกใช้ข้อมูลนั้นซึ่งยังไม่ได้จัดเก็บลงในแคช และเมื่อบันทึกข้อมูลลงในแคชครั้งต่อๆ ไปจะสามารถเรียกดูข้อมูลจากแคชได้ ดังนั้นเพื่อที่จะลดจำนวนของคำร้องผ่านเครือข่ายไปยังระบบแคช งานวิจัยชิ้นนี้จึงนำเสนอดัชนีแคช (cache index) สำหรับอนุกรมเวลาเพื่อลดอัตราในอ่านข้อมูลในแคชพลาด (Miss rate) อย่างไรก็ตามการใช้ดัชนีแคชส่งผลทำให้เกิดการใช้พื้นที่หน่วยความจำและต้องการเวลาในการประมวลผลเพิ่มขึ้น ซึ่งอาจจะส่งผลต่อประสิทธิภาพโดยรวม โดยดัชนีแคชนี้จะเป็นข้อมูลอาร์เรย์ของ long ขนาด 64 บิต ซึ่งใช้ตัวดำเนินการแบบบิตสำหรับจัดการดัชนีแคช ดังแสดงในรูปที่ 3.4

รูปที่ 3.4 ยังแสดงถึงตัวอย่างของการกำหนด chunk size เป็น 4 หรือใช้ข้อมูล HBase 4 แถว ต่อหนึ่ง fragment ซึ่งแต่ละ fragment จะถูกอ้างถึงด้วยหนึ่งบิตของดัชนีแคช โดยจะใช้คีย์ในการหาข้อมูลที่อยู่ใน Memcached โดยแต่ละบิต บิตหนึ่งหมายถึง fragment นั้นถูกจัดเก็บลงระบบแคชแล้ว และบิตศูนย์หมายถึง fragment นั้นยังไม่ได้จัดเก็บในแคช



รูปที่ 3.4: ดัชนีแคชที่ออกแบบสำหรับอ้างอิงข้อมูลในฐานข้อมูลอนุกรมเวลา OpenTSDB



รูปที่ 3.5: การสร้างอาร์เรย์ของ cache fragment จากดัชนีแคช

3.1.2.2 โมดูล Subquery Conversion

เพื่อที่จะจัดเก็บกลุ่มของข้อมูลแถวของ HBase ลงสู่ Memcached การแบ่งอนุกรมเวลาโดยเวลาของแต่ละจุดข้อมูลนั้นจำเป็นต่อการที่จะตัดสินใจว่าส่วนไหนของข้อมูลจะถูกจัดเก็บลงในแคช โดยโมดูลนี้จะมีอัลกอริทึมการตัดข้อมูลด้วยเวลา ดังแสดงในรูปที่ 3.3 แต่ละ fragment ประกอบไปด้วยส่วนของเวลาเริ่มต้น เวลาสิ้นสุด และสถานะของ fragment ที่ระบุว่า fragment นี้อยู่ในแคชหรือไม่

อัลกอริทึมการตัดอนุกรมเวลาด้วยเวลาที่นำเสนอในวิทยานิพนธ์นี้แบ่งออกเป็น 3 ขั้นตอน ดังนี้

1. การคำนวณหา จุดเริ่มต้นและสิ้นสุดของ fragment order ซึ่งจะจัดเก็บในแคช จะใช้สมการที่ 3.2 และ 3.1 ตามลำดับ

$$StartFO = \lceil \frac{T+1}{CS \times HBaseRowPeriod} \rceil \tag{3.2}$$

2. การสร้างอาร์เรย์ของ cache fragment โดยขั้นตอนนี้จะสร้าง fragment จากดัชนีแคชเพื่อที่จะหาบิตที่เหมือนกันซึ่งอยู่ติดกัน ดังแสดงในรูปที่ 3.5.

3. การแปลง fragment order ไปยังเวลาเริ่มต้นและสิ้นสุดของแต่ละ cache fragment ใช้สมการที่ 3.3 และ 3.4 ตามลำดับ

$$StartT = FO \times CS \times HBaseRowPeriod \tag{3.3}$$

$$EndT = (FO + 1) \times CS \times HBaseRowPeriod - 1 \tag{3.4}$$

3.1.2.3 โมดูล Find from Database

โมดูล Find from Database เป็นโมดูลต้นฉบับและใช้เป็นต้นแบบของการพัฒนา โดยทำหน้าที่ดึงข้อมูลดิบโดยใช้ช่วงของอนุกรมเวลาทั้งหมดจากคำร้องของผู้ใช้ การออกแบบนี้คงไว้ซึ่งการทำงานของโมดูลนี้แบบเดิม แต่เปลี่ยนวิธีการเรียกใช้งานเป็นช่วงเวลาที่กำหนดโดย subquery นั้นๆ เพื่อที่จะดึงเฉพาะช่วงเวลาบางส่วนของอนุกรมเวลาเท่านั้น

3.1.2.4 โมดูล Store in Cache

จากรูป 3.3 หลังจากที่โมดูล *Find from database* เสร็จสิ้น ขั้นตอนต่อไปคือกระบวนการของโมดูล *Store in cache* หรือการจัดเก็บข้อมูลลงแคช โดยโมดูลนี้จะทำหน้าที่ในการเก็บข้อมูลลงแคชแบบ asynchronous ตามการออกแบบของ OpenTSDB โดยคีย์ซึ่งเป็นตัวแทนของ fragment ที่จัดเก็บลงใน Memcached เป็นคีย์เดียวกับที่ใช้ระบุข้อมูลในแต่ละแถวของ HBase เพื่อไม่ให้ต้องสูญเสียเวลาในการแปลงข้อมูล และเนื่องจากโครงสร้างของ fragment ไม่ใช่อาร์เรย์ของไบต์ จึงจำเป็นต้องมีการแปลง (serialize) เป็นอาร์เรย์ของไบต์ ซึ่งจำเป็นสำหรับการจัดเก็บข้อมูลลง Memcached เพื่อที่จะพิสูจน์แนวคิด ผู้วิจัยได้สร้างอัลกอริทึมอย่างง่ายในการแปลง fragment

3.1.2.5 โมดูล Find from Cache

ตามที่ OpenTSDB ต้นฉบับจะมีโมดูล *Find from database* ซึ่งทำหน้าที่ดึงข้อมูลจาก HBase งานวิจัยนี้จึงประยุกต์การทำงานของโมดูล *Find from database* โดยการออกแบบ *Find from cache* ซึ่งมีช่องทางการรับข้อมูลเข้าและส่งข้อมูลออกจากโมดูลเหมือนกัน เพื่อให้ง่ายต่อการจัดการ โมดูล *Find from cache* จะทำหน้าที่ดึงข้อมูลที่เก็บไว้ใน Memcached ซึ่งจะต้องระบุคีย์ โดยที่ขอบเขตเวลาแต่ละ cache fragment จะถูกส่งไปยังโมดูล *Find from cache* ด้วยดังแสดงในรูปที่ 3.3 ดังนั้น คีย์ที่จะใช้หาข้อมูลใน Memcached นั้น จะใช้เวลาเริ่มต้นเป็นส่วนประกอบหลักของคีย์ ซึ่งจะถูกแปลงจาก fragment order ที่เริ่มต้นโดยใช้สมการที่ 3.3. หลังจากนั้น โมดูลนี้จะรับข้อมูลที่อยู่ในแคชจาก Memcached มาประมวลผลแบบ asynchronous ขั้นตอนสุดท้าย คือการแปลงข้อมูลจากอาเรย์ของไบต์ ให้อยู่ในรูปแบบของโครงสร้าง fragment และอยู่ในรูปแบบเดียวกันกับที่โมดูล *Find from database* ส่งออกมา

3.1.2.6 โมดูล Merging raw data

โมดูลนี้จะทำงานหลังจากที่ระบบได้นำข้อมูลออกจากแถวของ HBase จากทั้งโมดูลของ *Find from database* และ *Find from cache* มาถึงทั้งหมดแล้ว ดังแสดงในรูปที่ 3.3 ผลกระทบในการใช้อัลกอริทึมตัดเวลาคือ จะมีข้อมูลบางแถวที่ซ้ำซ้อนกันอยู่เนื่องจากการออกแบบของ OpenTSDB จึงทำให้การดึงข้อมูลแถวจาก HBase นั้นจะเกินกว่าขอบเขตที่ต้องการเสมอ เพราะว่าทุกๆ แถวจะเก็บข้อมูลหนึ่งชั่วโมง และปริมาณขั้นต่ำที่ดึงข้อมูลคือ หนึ่งแถว หมายความว่าแม้ข้อมูลอนุกรมเวลาที่ต้องการไม่ครบทั้งแถว โมดูล *Find from database* จะดึงข้อมูลมาทั้งแถว ดังนั้น ข้อมูลจาก HBase ทั้งหมดจะใช้ตัวเก็บข้อมูลแบบเซต ซึ่งจะรับประกันว่าจะไม่มีข้อมูลแถวที่ซ้ำกัน

3.2 วัสดุและอุปกรณ์

3.2.1 ซอฟต์แวร์

ซอฟต์แวร์ที่ออกแบบขึ้นเพื่อใช้ในการทดสอบทำงานอยู่บนระบบปฏิบัติการ Ubuntu server 16.04 OpenTSDB ที่เสริมระบบกลไกแคช ซึ่งการออกแบบระบบกลไกแคชทั้งหมดได้อธิบายไว้แล้วในหัวข้อที่ 4.1 โดยซอฟต์แวร์ที่ใช้ในวิทยานิพนธ์มีดังนี้

- OpenTSDB รุ่น 2.3.0 [21] โดยใช้ Oracle JDK 8 [38]
- Memcached รุ่น 1.5 [39]
- HBase รุ่น 1.2.6 [40] แบบโหนดเดี่ยว โดยใช้ Oracle JDK 8

ซึ่งซอฟต์แวร์ทั้งสามทำงานอยู่บน Docker รุ่น 17.12.1-ce [34]

3.2.2 ฮาร์ดแวร์

- หน่วยประมวลผล Intel Core i3-7100 2 คอร์ 4 เทรด
- หน่วยความจำหลัก DDR4 40 กิกะไบต์
- ฮาร์ดดิส WD WD20EZRX SATA 6 กิกกะไบต์/วินาที ความจุขนาด 2 เทระไบต์

บทที่ 4

การทดลองและวิเคราะห์ผลการทดลอง

4.1 การออกแบบการทดลอง

จากทดสอบเบื้องต้นพบว่าฐานข้อมูล OpenTSDB ไม่มีกลไกแคชในการอ่านข้อมูลจากฐานข้อมูล ซึ่งทุกการสอบถามนั้นต้องอ่านข้อมูลจากฐานข้อมูล HBase ทุกครั้ง ดังนั้นระยะเวลาในการสอบถามข้อมูลจาก HBase จึงเป็นเวลาส่วนใหญ่ของการสอบถามข้อมูลจากฐานข้อมูล OpenTSDB ซึ่งจะต้องอ่านข้อมูลจากฮาร์ดดิสก์ โดยกลไกแคชที่ได้ออกแบบจะลดระยะเวลาในการดึงข้อมูลจาก OpenTSDB โดยใช้ Memcached ในการจัดการแคช การทดลองกลไกแคชที่ได้ออกแบบเพื่อทดสอบประสิทธิภาพการสอบถามของข้อมูลอนุกรมเวลาจากฐานข้อมูลเวลาในกรณีช่วงเวลาของการสอบถามทับซ้อนกับช่วงเวลาที่เคยสอบถามไว้แล้ว โดยการทดลองได้แบ่งออกเป็นสองระบบดังนี้

1. **ระบบ OpenTSDB ที่ไม่มีแคช** คือระบบที่ใช้ฐานข้อมูล OpenTSDB แบบดั้งเดิมเชื่อมต่อฐานข้อมูล HBase
2. **ระบบ OpenTSDB ที่มีแคช** คือระบบที่ใช้ฐานข้อมูล OpenTSDB ซึ่งได้รวมกับกลไกแคชที่ได้ออกแบบเชื่อมต่อกับตัวจัดการแคช Memcached และฐานข้อมูล HBase

การออกแบบการทดลองอ้างอิงจากพฤติกรรมของผู้ใช้งานในการสั่งงานให้แสดงผลกราฟของอนุกรมเวลา โดยที่ผู้ใช้จะเลือกช่วงเวลาและการเลื่อนของช่วงเวลาที่ต้องการแสดงผลจากซ้ายไปขวาหรือขวาไปซ้าย หรือมีการปรับเปลี่ยนตัวแปรที่ใช้ในการแสดงผลเช่น การเลือกค่า downsampling เป็นต้น ซึ่งการดำเนินการของผู้ใช้ส่วนใหญ่นั้นมักจะดูผลจากข้อมูลอนุกรมเวลาเดียวกัน ดังนั้นจะมีโอกาสที่ระบบจะดึงช่วงของข้อมูลซ้ำซ้อนกัน โดยที่พฤติกรรมของผู้ใช้ดังกล่าวได้นำมาจำลองสถานการณ์เป็นตัวแปรหนึ่งของการทดลอง โดยปรับเปลี่ยนสัดส่วนของการซ้อนทับกันของการสอบถามเพื่อวิเคราะห์ประสิทธิภาพของกลไกแคช ในบทนี้ได้อธิบายตัวแปรต่างๆ ในหัวข้อ 4.1.1 คำอธิบายตัวแปรที่ใช้ในการทดลอง แต่ละสัดส่วนของการซ้อนทับกันของการสอบถามนั้นจะทดลองเป็นหนึ่งสถานการณ์ โดยส่งคำร้องในการสอบถาม 6 ครั้งไปยังทั้งสองระบบ เพื่อทดสอบการสอบถามข้อมูลซึ่งแต่ละครั้งของการส่งคำร้องมีช่วงเวลาของข้อมูลที่ต้องการซ้ำซ้อนกัน

4.1.1 คำอธิบายตัวแปรที่ใช้ในการทดลอง

ตัวแปรที่ใช้ในการทดลองมีดังนี้

- **จำนวนจุดข้อมูล** (Data points) คือจำนวนจุดข้อมูลทั้งหมดที่ใช้ในการสอบถาม
- **ร้อยละของการซ้อนทับกันของการสอบถาม** (The percentage of overlapping query ranges) คือร้อยละของการซ้อนทับกันของแต่ละครั้งที่ทำการสอบถาม ซึ่งในการทดลองส่งคำร้องการสอบถาม 6

ครั้ง เช่น ถ้าช่วงเวลาของข้อมูลคือ 1-100 การสอบถามแรกอยู่ในช่วง 1-50 และการสอบถามที่สองอยู่ในช่วง 25-75 หมายความว่าร้อยละของการซ้อนทับกันของการสอบถามคือ 50%

- **ความหนาแน่นของจุดข้อมูล** (Data density) คือจำนวนจุดข้อมูลต่อหนึ่งชั่วโมง โดยตัวแปรนี้อ้างอิงตามโครงสร้างข้อมูลของฐานข้อมูล OpenTSDB ที่จัดเก็บอยู่บนฐานข้อมูล HBase ซึ่งในหนึ่งแถวของตารางฐานข้อมูล HBase จะบรรจุข้อมูลอนุกรมเวลาหนึ่งชั่วโมง ดังนั้นความหนาแน่นของจุดข้อมูลส่งผลต่อประสิทธิภาพของการสอบถาม

- **ขนาดของ chunk** (Chunk size หรือ CS) คือจำนวนแถวของตารางฐานข้อมูล HBase ที่รวมกันเป็นหนึ่ง fragment ซึ่งจะถูกนำไปใช้ในการจัดเก็บลง Memcached รูปที่ 3.4 แสดงถึงตัวอย่างของขนาดของ chunk โดยขนาดของ chunk ส่งผลต่อปริมาณวัตถุที่จะจัดเก็บลง Memcached ดังนั้นขนาดของ chunk จึงส่งผลต่อประสิทธิภาพของการสอบถาม

ในการทดลองจะปรับเปลี่ยนขนาดของ chunk เพื่อหาขนาดที่เหมาะสม เพราะขนาดของ chunk นั้นส่งผลกระทบต่อขนาดของวัตถุที่จะจัดเก็บลง Memcached และประสิทธิภาพของระบบ หัวข้อที่ 4.1.2 ปัจจัยและตัวแปรที่ปรับเปลี่ยนในการทดลอง เป็นค่าของตัวแปรที่ใช้ในการทดลองทั้งหมด และในแต่ละตัวแปรที่เลือกนั้นจะทดลองแบบเดิมซ้ำ 3 ครั้งเพื่อหาค่าเฉลี่ยของเวลาในการสอบถามข้อมูล

4.1.2 ปัจจัยและตัวแปรที่ปรับเปลี่ยนในการทดลอง

ในการทดลองปรับเปลี่ยนค่าตัวแปรและปัจจัยต่างๆ ดังนี้

- **จำนวนจุดข้อมูล** (ล้านจุด): 1, 5, 10
- **ร้อยละของการซ้อนทับกันของการสอบถาม** (%): 10, 25, 50, 75, 100
- **ความหนาแน่นของจุดข้อมูล** (จุดต่อชั่วโมง): 144, 720, 3600
- **ขนาดของ chunk**: 1, 16, 256, 512, 1024

ในการทดลองแต่ละครั้งจะตรวจสอบผลลัพธ์ของการทำงานของ**ระบบ OpenTSDB ที่มีแคช** ว่าถูกต้องเหมือนกับต้นฉบับ (**ระบบ OpenTSDB ที่ไม่มีแคช**) ก่อนที่จะวัดระยะเวลาในการสอบถามข้อมูลทั้งสองระบบ เพื่อยืนยันว่ากลไกที่ผู้วิจัยได้ออกแบบทำงานได้ถูกต้องตามระบบต้นฉบับหรือไม่ การวัดผลการทดลองจะวัดเฉพาะระยะเวลาในการตอบสนองของการดึงข้อมูลดิบซึ่งไม่รวมกับโมดูล post-processing ในทั้งสองระบบ โดย**ระบบ OpenTSDB ที่ไม่มีแคช** วัดระยะเวลาในการทำงานของโมดูล *Find from database* เท่านั้นดังแสดงในรูปที่ 3.2 และ**ระบบ OpenTSDB ที่มีแคช** วัดระยะเวลาในการทำงานตั้งแต่โมดูล *Subquery conversion* ถึงโมดูล *Merging raw data* ในรูปที่ 3.3 ซึ่งได้แทนการทำงานของโมดูล *Find from database* ในฐานข้อมูล OpenTSDB ต้นฉบับ

4.2 วิธีการทดลอง

4.2.1 การสั่งทำการทดลอง

จากการออกแบบการปรับเปลี่ยนปัจจัยต่างๆ ในการทดลองที่กล่าวมาในหัวข้อ 4.1 พบว่ามีจำนวนครั้งในการทดลองสูงถึง 225 ครั้ง โดยการจับคู่ผสมทุกตัวแปรที่เป็นไปได้ และมีการทดสอบซ้ำ 3 ครั้งในแต่ละการทดลองเพื่อหาค่าเฉลี่ยของเวลาในการสอบถาม ดังนั้นผู้วิจัยจึงออกแบบและพัฒนาโปรแกรมสำหรับการทดสอบและวัดผลการทดลอง โดยได้พัฒนาทั้งหมด 4 โปรแกรมดังต่อไปนี้

1. **โปรแกรมสำหรับการจัดการ Docker container** ทั้งสองระบบ โดยใช้ Docker compose [41] โดยมีการนิยามไฟล์ Dockerfile [42] สำหรับการกำหนดความสามารถของ Docker image ซึ่งใช้เป็นต้นแบบของแต่ละ container และมี Docker image ที่มาจากศูนย์การเก็บ image ของ Docker (Docker Hub) [43] ดังนี้

    - **Java Dockerfile** เป็นต้นแบบของ container สำหรับกำหนด base image ตั้งต้นสำหรับทั้งสองระบบ โดยมีระบบปฏิบัติการ Debian Jessie เป็น base image ซึ่งจะติดตั้ง Java JDK 8 สำหรับพร้อมใช้งาน
    - **OpenTSDB Dockerfile** เป็นต้นแบบของ container สำหรับฐานข้อมูล OpenTSDB และมีการโปรแกรมสำหรับการรอการทำงานเพื่อให้ HBase และ Zookeeper พร้อมใช้งานก่อน
    - **HBase Dockerfile** เป็นต้นแบบของ container สำหรับฐานข้อมูล HBase ซึ่งจะแยกออกมาจาก container ของ OpenTSDB ซึ่งง่ายต่อการจัดการและปรับเปลี่ยนเพื่อการขยายตัว
    - **Memcached image** เป็น image ทางการ [44] สำหรับใช้เป็นตัวจัดการแคช

2. **โปรแกรม สำหรับ สร้าง ข้อมูล แบบ สุ่ม ค่า สำหรับ ทำการ ทดลอง** เพื่อให้ข้อมูลมีคุณลักษณะ ตามที่ต้องการ โดยสามารถกำหนด*จำนวนของจุดข้อมูล*, *เวลาเริ่มต้นของข้อมูล*, และ*ความหนาแน่นของจุดข้อมูล* โดยกำหนดเป็นค่าเวลาระยะห่างระหว่างจุดข้อมูล และ*จำนวนไฟล์ของข้อมูล*ที่ต้องการแบ่งจาก*จำนวนของจุดข้อมูล* เนื่องจากการทดสอบการนำข้อมูลเข้าฐานข้อมูล OpenTSDB นั้นไม่สามารถนำเข้าครั้งละปริมาณมากๆ จึงต้องแบ่งให้มีขนาดเล็กลง

3. **โปรแกรมสำหรับชุดคำสั่งสำหรับการสอบถาม** โดยสามารถกำหนด*ร้อยละของการซ้อนทับกันของการสอบถาม* ได้ โดยโปรแกรมนี้สร้างสถานการณ์จาก *ร้อยละของการซ้อนทับกันของการสอบถาม*ที่กำหนด โดยจะคำนวณหาคู่ของเวลาตั้งต้นและสิ้นสุดของแต่ละการสอบถามใน 1 ชุด โดยหนึ่งชุดจะมีคำสั่งสอบถาม 6 ครั้ง โปรแกรมนี้จะมี 2 ฟังก์ชันหลักคือ

    (a) ฟังก์ชัน การ คำนวณ หา คู่ ของ เวลา ตั้ง ต้น และ สิ้น สุด ของ แต่ละ การ สอบถาม 1 ชุด คือ $(st_1, et_1) \ldots (st_n, et_1)$ โดยกำหนดให้

    - st คือเวลาเริ่มต้นของจุดข้อมูล (start time)

- et คือเวลาสิ้นสุดของจุดข้อมูล (end time)
- rs คือขนาดของช่วงเวลาในแต่ละการสอบถาม (range size) ซึ่งจะเท่ากันทุกการสอบถาม โดยที่ $rs = et_i - st_i$
- n คือจำนวนของคำสั่งในการสอบถามของการทดลองหนึ่งๆ (number of queries)
- op คือร้อยละของการซ้อนทับกันของการสอบถาม (overlapping percentage)

ขนาดของช่วงเวลาแต่ละการสอบถาม นั้นส่งผลต่อปริมาณข้อมูลในแต่ละสอบถาม ซึ่งการเลือก *ขนาดของช่วงเวลาแต่ละการสอบถาม* ที่เหมาะสมคือ ช่วงเวลาแต่ละการสอบถามรวมทั้งหมดจะต้องมากกว่าหรือเท่ากับช่วงเวลาของข้อมูลอนุกรมเวลาทั้งหมด หรือใกล้เคียงที่สุด อัลกอริทึมที่ 4.1 แสดงถึงวิธีการคำนวณหาคู่ของเวลาตั้งต้นและสิ้นสุดของแต่ละการสอบถาม 1 ชุด

อัลกอริทึม 4.1 อัลกอริทึมการคำนวณหาคู่ของเวลาตั้งต้นและสิ้นสุดของแต่ละการสอบถาม 1 ชุด

```
1:  function FindQueryRanges(st, rs, n, op)
2:      Let Q[2][0...n] be new arrays      ▷ Q[1][i] = เวลาเริ่มต้นของคำสั่งการสอบถามที่ i
3:                                         ▷ Q[2][i] = เวลาสิ้นสุดของคำสั่งการสอบถามที่ i
4:      Q[1][0] = st
5:      Q[2][1] = st + rs
6:      for i = 2 to n do
7:          Q[1][i] = ⌊Q[2][i − 1] − (op × rs ÷ 100)⌋
8:          Q[2][i] = ⌊Q[1][i] + rs⌋
9:      end for
10:     return Q
11: end function
```

(b) ฟังก์ชันการคำนวณหาขนาดของช่วงเวลาแต่ละการสอบถาม (range size) แสดงในอัลกอริทึมที่ 4.2 และใช้ตัวแปรเหมือนกับอัลกอริทึมที่ 4.1

อัลกอริทึม 4.2 อัลกอริทึมการคำนวณหาขนาดของช่วงเวลาแต่ละการสอบถาม (range size)

```
1:  function FindRangeSize(st, et, n, op)
2:      Let o = 0                          ▷ l คือ ขนาดของช่วงเวลาแต่ละการสอบถาม (range size)
3:      Let rs = 0
4:      Let l = 0
5:      while et − l > 0 do
6:          rs = ⌊(st − et)/(n − (n − 1) × op ÷ 100) − o⌋ × −1
7:          l = FindQueryRanges(st, rs, n, op)[i][n]
8:          o = o + 1
9:      end while
10:     return rs
11: end function
```

4. **โปรแกรมสำหรับสั่งการทดลอง** โดยออกแบบและทำการทดลองตามอัลกอริทึมที่ 4.3 และในขั้นตอนของอัลกอริทึมนี้ ได้มีการตรวจสอบความถูกต้องของโปรแกรมกลไกแคชที่พัฒนาโดยอัลกอริทึมที่ 4.4 และการทำงานทดสอบการทำงานโดยอัลกอริทึมที่ 4.5

อัลกอริทึม 4.3 ขั้นตอนการสั่งการทดลอง

Let $DataPoints = [1000000, 5000000, 10000000]$
Let $OverlappingPercentage = [10, 25, 50, 75, 100]$
Let $DataDense = [144, 720, 3600]$
Let $ChunkSize = [1, 16, 256, 512, 1024]$
▷ กำหนดตัวแปรอื่นๆ ตามความเหมาะสมในแต่ละสถานการณ์
for $dps = 1$ to length($DataPoints$) **do**
    for $dd = 1$ to length($DataDense$) **do**
        สร้างข้อมูลแบบสุ่มค่าโดย **โปรแกรมสำหรับสร้างข้อมูลแบบสุ่มค่าสำหรับทำการทดลอง** โดยใช้ $DataPoints[dps]$ และ $DataDense[dd]$
        for $cs = 1$ to length($ChunkSize$) **do**
            for $op = 1$ to length($OverlappingPercentage$) **do**
                ทำลายระบบทั้งสอง
                ตรวจสอบความถูกต้องของโปรแกรมกลไกแคชที่พัฒนา
                ทำการทดลองบน**ระบบ OpenTSDB ที่ไม่มีแคช**
                ทำการทดลองบน**ระบบ OpenTSDB ที่มีแคช**
                บันทึกผลการทดลองลงไฟล์
            end for
        end for
    end for
end for

อัลกอริทึม 4.4 ตรวจสอบความถูกต้องของโปรแกรมกลไกแคชที่พัฒนา

1: ▷ ตรวจสอบความถูกต้องโดยใช้ op] (OverlappingPercentage)
2: สร้าง**ระบบ OpenTSDB ที่ไม่มีแคช** โดยใช้ Docker Compose
3: นำข้อมูลเข้าสู่**ระบบ OpenTSDB ที่ไม่มีแคช**
4: ทดสอบ และ บันทึก ผลลัพธ์ ของ การ สอบถาม ใน**ระบบ OpenTSDB ที่ไม่มี แคช** ซึ่ง ใช้ ชุด คำ สั่ง การ สอบถามจาก **โปรแกรมสำหรับชุดคำสั่งสำหรับการสอบถาม** โดยใช้ op
5: สร้าง**ระบบ OpenTSDB ที่มีแคช** โดยใช้ Docker Compose
6: นำข้อมูลเข้าสู่**ระบบ OpenTSDB ที่มีแคช**
7: ทดสอบและบันทึกผลลัพธ์ของการสอบถามใน**ระบบ OpenTSDB ที่มีแคช** ซึ่งใช้ชุดคำสั่งการสอบถามจาก **โปรแกรมสำหรับชุดคำสั่งสำหรับการสอบถาม** โดยใช้ op
8: ตรวจสอบผลลัพธ์ของการสอบถามทั้งสองระบบต้องเหมือนกัน ที่ op ถ้าไม่เหมือนกันจะหยุดทำการทดลอง
9: ▷ ตรวจสอบความถูกต้องโดยใช้ op ที่มีค่า 100
10: ทดสอบ และ บันทึก ผลลัพธ์ ของ การ สอบถาม ใน**ระบบ OpenTSDB ที่ไม่มี แคช** ซึ่ง ใช้ ชุด คำ สั่ง การ สอบถามจาก **โปรแกรมสำหรับชุดคำสั่งสำหรับการสอบถาม** โดยใช้ op ที่มีค่า 100
11: ทดสอบและบันทึกผลลัพธ์ของการสอบถามใน**ระบบ OpenTSDB ที่มีแคช** ซึ่งใช้ชุดคำสั่งการสอบถามจาก **โปรแกรมสำหรับชุดคำสั่งสำหรับการสอบถาม** โดยใช้ op ที่มีค่า 100
12: ตรวจสอบผลลัพธ์ของการสอบถามทั้งสองระบบต้องเหมือนกัน ที่ op มีค่า 100 หมายความว่าเป็นการทดสอบการสอบถามข้อมูลทั้งหมด ถ้าไม่เหมือนกันจะหยุดทำการทดลอง

อัลกอริทึม 4.5 ทดลองบนระบบ OpentSDB ที่ไม่มีแคช หรือระบบ OpentSDB ที่มีแคช

1: **function** RunExperiment($system$) ▷ $system$ คือ OpentSDB ที่ไม่มีแคชหรือมีแคช
2: Let $NumberExperiment = 3$ ▷ $NumberExperiment$ คือจำนวนครั้งในการทดลอง
3: **for** $i = 1$ to $NumberExperiment$ **do**
4: ทำลายระบบทั้งสอง
5: สร้าง**ระบบ OpenTSDB** โดยใช้ Docker Compose
6: นำข้อมูลเข้าสู่**ระบบ OpenTSDB**
7: ทดสอบ และ บันทึก ผลลัพธ์ ของ การ สอบถาม ใน**ระบบ OpenTSDB** ซึ่ง ใช้ ชุด คำ สั่ง การสอบถามจาก **โปรแกรมสำหรับชุดคำสั่งสำหรับการสอบถาม** โดยใช้ op (Overlapping Percentage)
8: **end for**
9: **end function**

4.2.2 ปัญหาและอุปสรรคระหว่างทำการทดลอง

1. ระยะเวลาทดลองค่อนข้างนาน โดยถ้าสั่งให้ทำงานต่อเนื่องตามที่ผู้วิจัยกำหนดต้องใช้ระยะเวลาในการทำงานทั้งสิ้น 48 ชั่วโมง เนื่องจากทุกครั้งที่จะทำการทดลองใหม่จะต้องลบข้อมูลทั้งหมดออกจาก

ฐานข้อมูล OpenTSDB และ HBase ซึ่งมีอัลกอริทึมในการทำ compaction ของข้อมูลซึ่งเป็นการรวมหลายคอลัมน์ของฐานข้อมูล HBase เป็นหนึ่งคอลัมน์ เพื่อลดขนาดพื้นที่ที่จัดเก็บ [26] อันจะเป็นเหตุให้การทดลองแต่ละครั้งมีความคลาดเคลื่อนได้ และสาเหตุที่ต้องลบข้อมูลทุกครั้งเนื่องจากมีการทำ index ของ HBase และ Zookeeper ทำให้ไม่สามารถเก็บข้อมูลต้นฉบับที่ไม่ผ่านการทำ compaction ได้โดยง่าย

2. เนื่องจากเป็นการทดลองต่อเนื่อง ถ้าหากโปรแกรมสำหรับสั่งการทดลองหยุดการทำงานในขณะที่งานยังไม่เสร็จ จะทำให้เพิ่มระยะเวลาในการทดลองเป็นอย่างมาก ผู้วิจัยพบปัญหาระหว่างสั่งการทดลองดังนี้

    - เนื่องจากข้อมูลมีขนาดใหญ่จึงทำให้โปรเซสที่สั่งให้ Docker container ของ OpenTSDB ให้นำข้อมูลเข้าสู่ฐานข้อมูลไม่ตอบสนองเมื่อทำงานไปสักระยะหนึ่ง ซึ่งไม่สามารถทำการทดลองต่อเนื่องได้ ผู้วิจัยจึงแก้ปัญหาด้วยวิธีการแบ่งงานของโปรเซสให้สั้นลงจึงทำให้โปรแกรมนี้สามารถทำงานได้อย่างต่อเนื่อง
    - ปัญหาต่อเนื่องจากข้อก่อนหน้านี้คือ งานของโปรเซสนั้นทำไม่เสร็จสมบูรณ์หรือไม่สามารถดำเนินการต่อได้ และเมื่อตรวจพบปัญหาที่เกิดขึ้นระหว่างนำข้อมูลเข้า ผู้วิจัยจะส่งคำสั่งของโปรเซสที่แบ่งงานไว้แล้วซ้ำใหม่จนกว่าจะได้สามารถนำเข้าข้อมูลได้สมบูรณ์ จากการอ่านบันทึกประวัติการทำงานของตัวนำเข้าข้อมูลของฐานข้อมูล OpenTSDB [45] ข้อความที่เกี่ยวข้องกับปัญหาที่เกิดขึ้นระหว่างนำเข้าข้อมูลมีดังนี้

        - `org.hbase.async.TableNotFoundException: ''tsdb''`
          หรือ `org.hbase.async.TableNotFoundException: ''tsdb-uid''`
          ซึ่งในการออกแบบฐานข้อมูล OpenTSDB จะสร้างตารางชื่อ `tsdb` และ `tsdb-uid` ในฐานข้อมูล HBase เพื่อเก็บข้อมูลอนุกรมเวลา ในกรณีนี้คือ ไม่พบตารางดังกล่าวในฐานข้อมูล HBase แต่จากตรวจสอบของผู้วิจัยพบว่าตาราง `tsdb` `tsdb` และ `tsdb-uid` ได้ถูกสร้างไว้แล้วก่อนที่ข้อความนี้จะแสดงออกมา ซึ่งจากการทดสอบเบื้องต้นพบว่าข้อความนี้เกิดขึ้นบางครั้ง
        - `Exception in thread ''main''` มีข้อผิดพลาดเกิดขึ้นที่ฟังก์ชัน main ระหว่างการนำข้อมูลเข้า และจากการทดสอบเบื้องต้นพบว่าข้อความนี้เกิดขึ้นบางครั้ง
        - `DeferredGroupException: At least one of the Deferreds failed` มีข้อผิดพลาดเกิดขึ้นที่กระบวนการดำเนินการแบบ Asynchronous และจากการทดสอบเบื้องต้นพบว่าข้อความนี้เกิดขึ้นบางครั้ง
        - `TSDB: Completed shutting down the TSDB` คือ OpenTSDB ได้สิ้นสุดการทำงานแบบปกติ
        - `org.hbase.async.CallQueueTooBigException: Call queue is full` คือ โครงสร้างข้อมูลคิวของตัวนำเข้าข้อมูลของ OpenTSD เต็ม โดยจากการทดสอบเบื้องต้น

พบว่าข้อความนี้จะถูกแสดงต่อเนื่องจนกระทั่งคิวว่างพอให้สำรองข้อมูลสำหรับนำเข้าต่อไป ซึ่งเป็นกระบวนการปกติในการรอคิวให้ว่าง

จากทดลองพบว่ามีข้อความผิดพลาดเกิดขึ้นแบบไม่สามารถระบุสาเหตุได้ และเกิดขึ้นแบบสุ่ม ดังนั้นจึงต้องจัดการกับข้อความการผิดพลาดดังกล่าว เพื่อให้มั่นใจว่าข้อมูลถูกนำเข้าอย่างสมบูรณ์ โดยมีการจัดการขั้นตอนดังนี้

(a) การสั่งการนำเข้าข้อมูลสู่ฐานข้อมูล OpenTSDB ที่มีถูกแบ่งงานย่อยไว้แล้ว เพื่อให้ใช้ระยะเวลาทำงานสั้นลง และกำหนดว่าหากงานย่อยยังดำเนินการไม่เสร็จภายในระยะเวลาที่กำหนดก็จะทดลองสั่งคำสั่งเดิมซ้ำไปเรื่อยๆ จนกระทั่งถึงจำนวนครั้งที่กำหนดไว้

(b) ตรวจความผิดพลาดในแต่ละบรรทัดที่ทำให้การนำข้อมูลเข้าไม่สมบูรณ์ โดยเมื่อข้อความบันทึกผลการทำงานของตัวนำข้อมูลเข้าฐานข้อมูล OpenTSDB แล้วเป็นไปตามเงื่อนไขดังต่อไปนี้

ถ้ามีข้อความ `'org.hbase.async.TableNotFoundException: ''tsdb'` หรือ `Exception in thread ''main''` หรือ `DeferredGroupException: At least one of the Deferreds failed` อยู่ในบรรทัดปัจจุบัน โดยไม่มีข้อความ `Failed to shutdown the TSD` อยู่ในบรรทัดปัจจุบันและบรรทัดก่อนหน้านี้ ให้ทำการส่งคำสั่งเดิมซ้ำไปเรื่อยๆ จนกระทั่งถึงจำนวนครั้งที่กำหนดไว้

3. การวัดผลการทดลองจำเป็นต้องบันทึกผลลัพธ์จากการสอบถามซึ่งเป็นรูปแบบ JSON เพราะต้องใช้จุดข้อมูลที่ผู้ใช้ต้องการสำหรับตรวจสอบความถูกต้องของกลไก และสถานะการสอบถาม รวมถึงระยะเวลาในการทำงานของแต่ละโมดูลสำหรับการวัดผลระยะเวลารวมในการทำงาน ในการทดสอบไม่สามารถสอบถามเพื่อดึงข้อมูลดิบทั้งหมดออกมาได้ เนื่องจากขนาดข้อมูลที่ใช้ทดสอบมีขนาดใหญ่และการสอบถามใช้โปรโตคอล HTTP ซึ่งมีข้อจำกัดในกรณีที่ผลลัพธ์ของการสอบถามมีขนาดใหญ่กว่า 2 กิกะไบต์ซึ่งเป็นค่าเริ่มต้นที่ผู้วิจัยกำหนดไว้ในการทดสอบ ซึ่งผู้วิจัยไม่ได้เน้นประเด็นนี้ในการวิจัย ดังนั้นผู้วิจัยจึงใช้การกำหนดค่า downsampling เพื่อลดขนาดข้อมูลลง

เมื่อข้อมูลมีขนาดค่อนข้างใหญ่กว่าขนาดของหน่วยความจำของโปรแกรมสั่งการทดลอง และไม่สามารถบันทึกผลการสอบถามทั้งหมดลงไฟล์ได้ ทางผู้วิจัยจึงจำเป็นต้องปรับปรุงส่วนของการบันทึกผลการทดลองของโปรแกรมนี้โดยแบ่งเป็นท่อนเล็กๆ และใช้คำสำคัญ `,{''statsSummary''` ในการแบ่งส่วนระหว่างจุดข้อมูลและสถานะการสอบถาม โดยมี 2 กรณีคือ กรณีที่คำสำคัญทุกตัวอักษรอยู่ในข้อมูลท่อนเดียวกัน และกรณีอยู่ต่างท่อนกัน เพื่อให้โปรแกรมทำงานได้อย่างถูกต้อง ผู้วิจัยจึงต้องแก้ปัญหาตรงนี้ด้วย

4.3 ผลการทดลอง

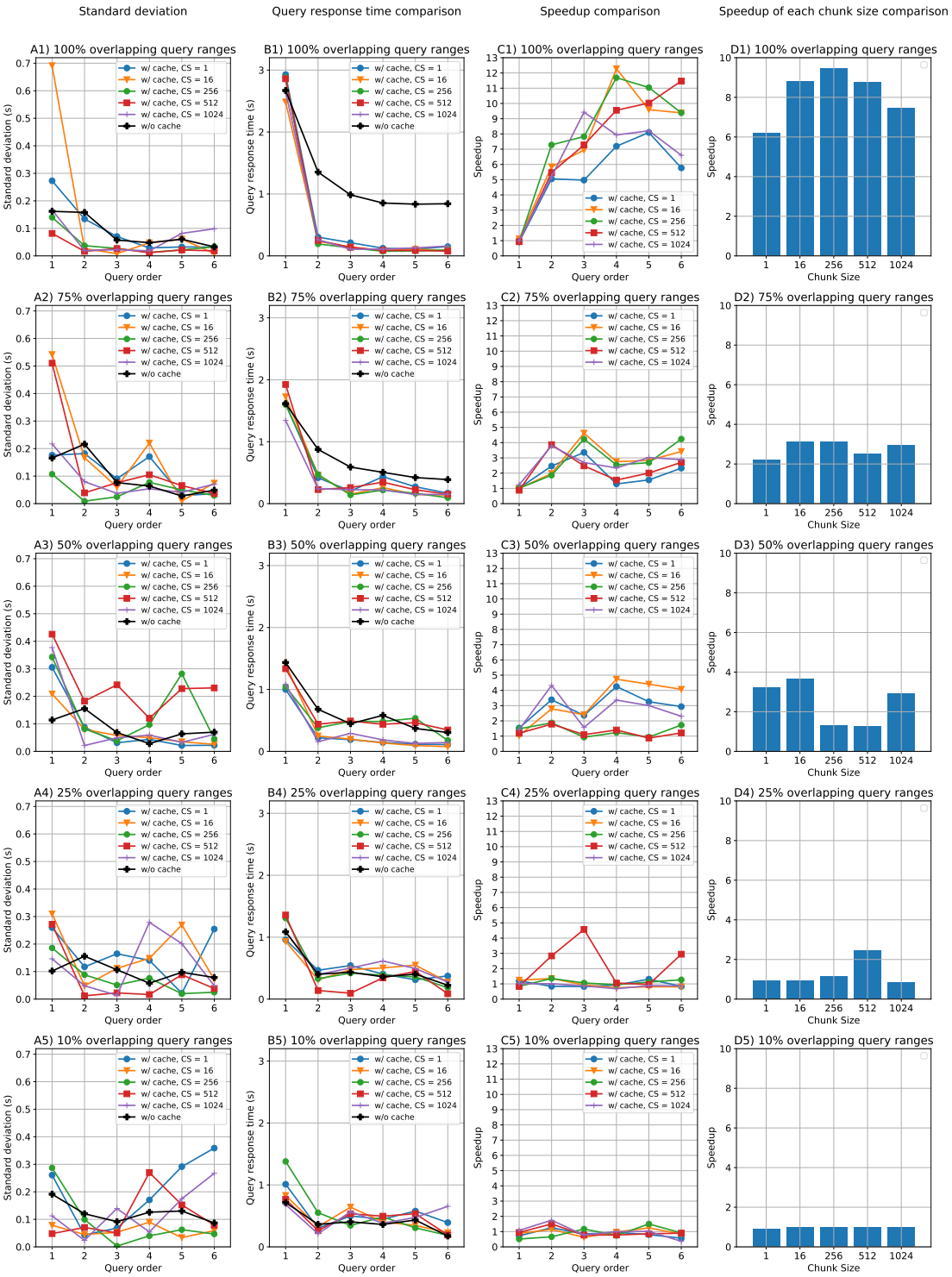
การทดลองเพื่อเปรียบเทียบประสิทธิภาพของกลไกแคชของ ฐานข้อมูล OpenTSDB ที่ได้ออกแบบเมื่อเปรียบเทียบกับฐานข้อมูล OpenTSDB แบบดั้งเดิม ซึ่งมี 9 การทดลองย่อย

โดยแต่ละการทดลองย่อยกำหนดค่าตัวแปรดังต่อไปนี้

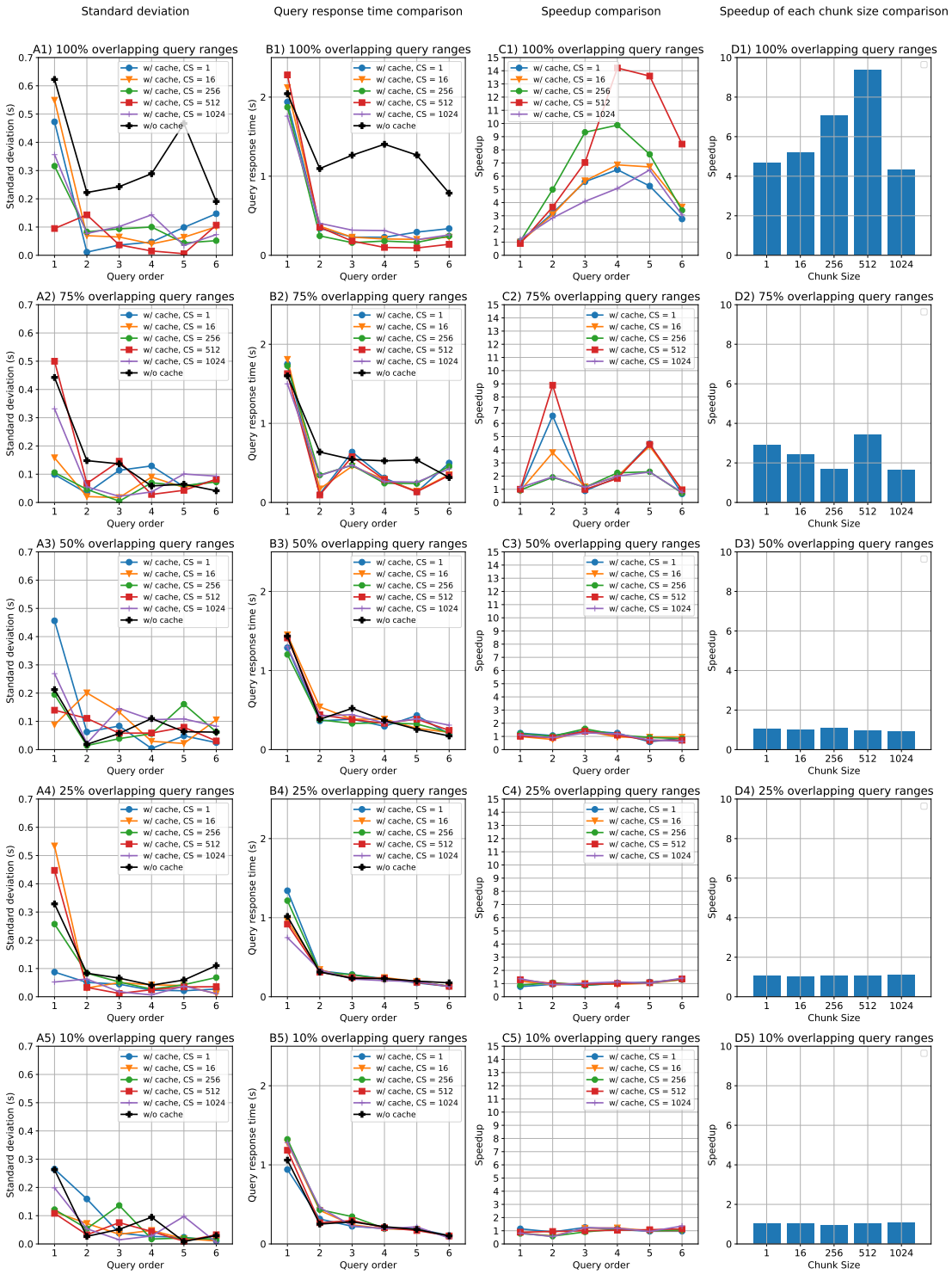
1. จำนวนจุดข้อมูล 1 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 144 จุดต่อชั่วโมง แสดงดังรูปที่ 4.1
2. จำนวนจุดข้อมูล 1 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 720 จุดต่อชั่วโมง แสดงดังรูปที่ 4.2
3. จำนวนจุดข้อมูล 1 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 3,600 จุดต่อชั่วโมง แสดงดังรูปที่ 4.3
4. จำนวนจุดข้อมูล 5 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 144 จุดต่อชั่วโมง แสดงดังรูปที่ 4.4
5. จำนวนจุดข้อมูล 5 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 720 จุดต่อชั่วโมง แสดงดังรูปที่ 4.5
6. จำนวนจุดข้อมูล 5 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 3,600 จุดต่อชั่วโมง แสดงดังรูปที่ 4.6
7. จำนวนจุดข้อมูล 10 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 144 จุดต่อชั่วโมง แสดงดังรูปที่ 4.7
8. จำนวนจุดข้อมูล 10 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 720 จุดต่อชั่วโมง แสดงดังรูปที่ 4.8
9. จำนวนจุดข้อมูล 10 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 3,600 จุดต่อชั่วโมง แสดงดังรูปที่ 4.9

ผลทดลองของการเปรียบเทียบประสิทธิภาพของกลไกแคชของ ฐานข้อมูล OpenTSDB ที่ได้ออกแบบ (w/ cache, CS) กับฐานข้อมูล OpenTSDB แบบดั้งเดิม (w/o cache) โดยไม่ได้กำหนดขนาดของแคชสูงสุดที่สามารถใช้งานได้ ในแต่ละกราฟของผลการทดลองใช้ค่าของขนาดของ chunk ที่ต่างกันคือ 1, 16, 256, 512 และ 1024 โดยกราฟในคอลัมน์ที่หนึ่ง (A) แกนตั้งเป็นค่าส่วนเบี่ยงเบนมาตรฐานของระยะเวลาในการตอบสนอง (Query response time) แกนนอนแสดงถึงลำดับการส่งการสอบถาม (Query order) กราฟในคอลัมน์ที่สอง (B) แกนตั้งแสดงถึงการเปรียบค่าเฉลี่ยของระยะเวลาในการตอบสนอง (Query response time) แกนนอนแสดงถึงลำดับการส่งการสอบถาม (Query order) กราฟในคอลัมน์ที่สาม (C) แกนตั้งเป็นค่าเฉลี่ยของ Speedup เมื่อเปรียบเทียบกับ ฐานข้อมูล OpenTSDB แบบดั้งเดิม แกนนอนแสดงถึงลำดับการส่งการสอบถาม (Query order) กราฟในคอลัมน์ที่สาม (D) แกนตั้งเป็นค่าเฉลี่ยของ Speedup เมื่อเปรียบเทียบกับ ฐานข้อมูล OpenTSDB แบบดั้งเดิม แกนนอนแสดงค่าเฉลี่ยของแต่ละครั้งของ Speedup ของการสอบถามโดยไม่รวมครั้งแรกซึ่งยังไม่มีข้อมูลที่ต้องการอยู่ในแคช โดยเปรียบเทียบแต่ละขนาดของ chunk แต่ละแถวของกราฟและใช้สถานการณ์การทดสอบที่ต่างกันโดยปรับเปลี่ยนร้อยละของการซ้อนทับกันของการสอบถาม คือ 10%, 25%, 50%, 75% และ 100%

ผู้วิจัยได้สรุปเพื่อเปรียบเทียบเวลาในการตอบสนองที่เพิ่มขึ้นในแต่ละขนาดของ chunk ทั้งหมด ของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิมกับฐานข้อมูล OpenTSDB ที่มีแคชทุกปัจจัยในรูปที่ 4.10 และได้เปรียบเทียบเวลาในการตอบสนองที่เพิ่มขึ้นของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิมกับฐานข้อมูล OpenTSDB ที่มีแคชทุกปัจจัยในรูปที่ 4.11

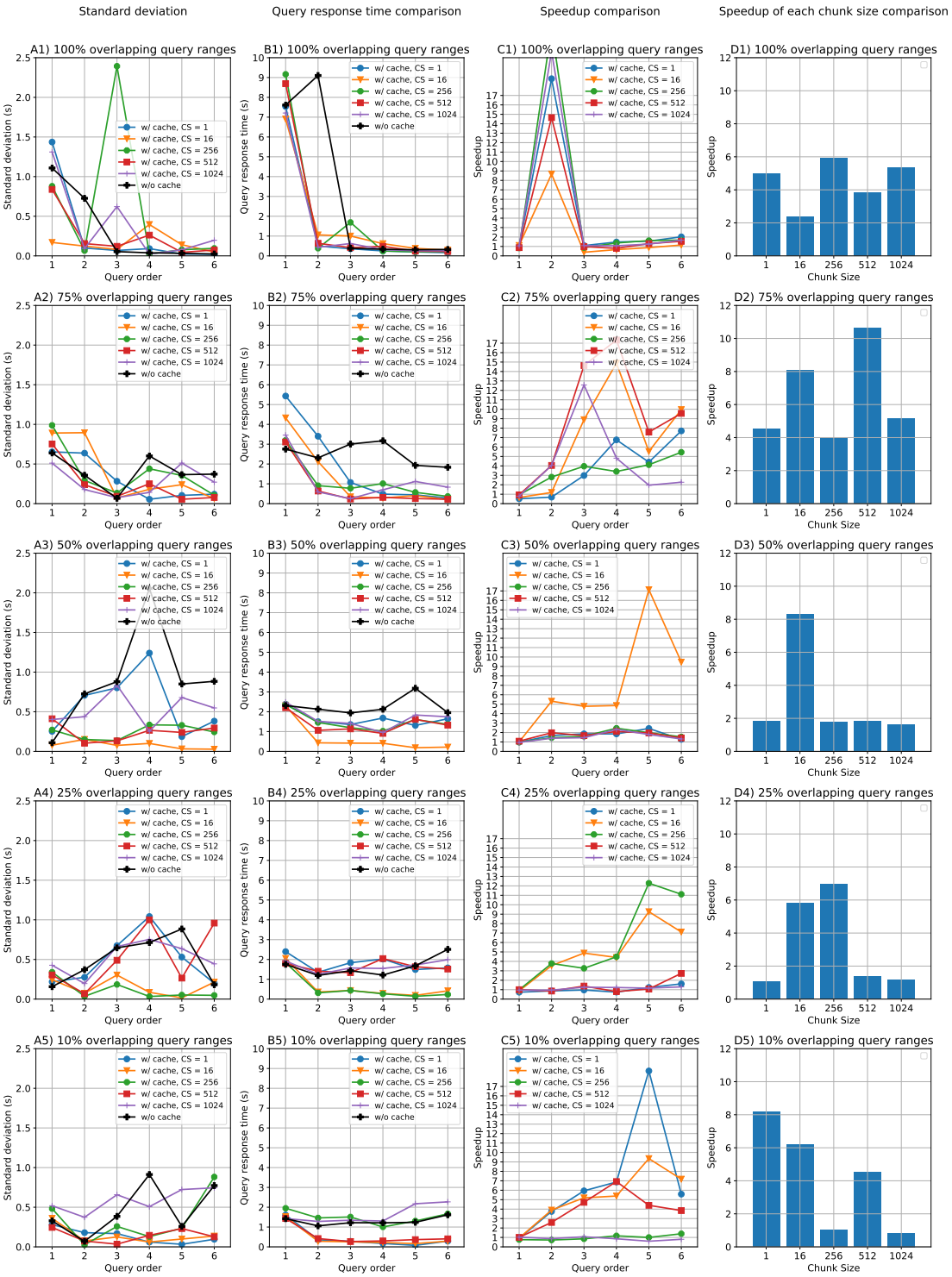


รูปที่ 4.1: ผลเปรียบเทียบเวลาในการตอบสนองของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช ซึ่งใช้จำนวนจุดข้อมูล 1 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 144 จุดต่อชั่วโมงเป็นข้อมูลสำหรับทดลอง

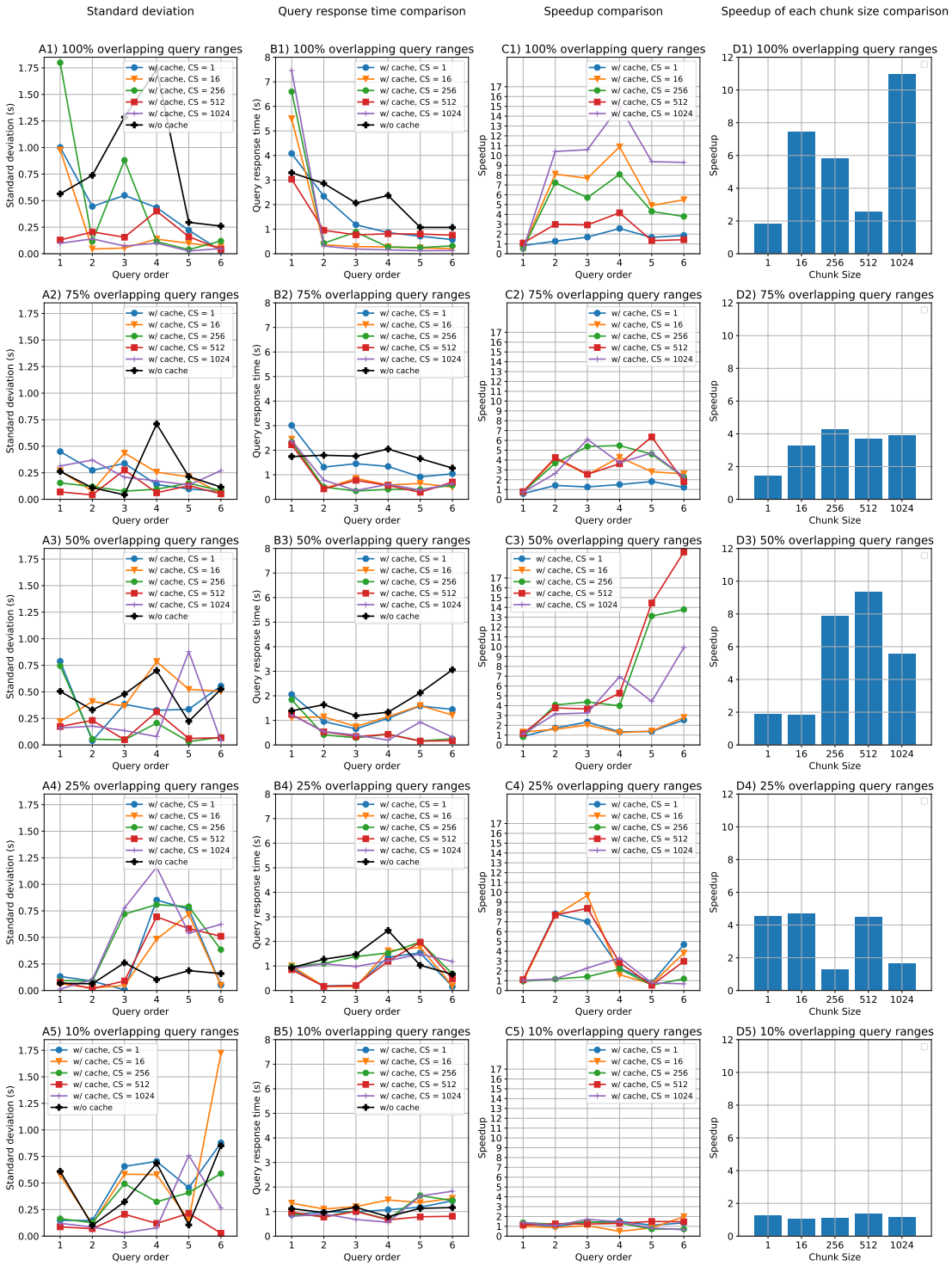


รูปที่ 4.2: ผลเปรียบเทียบเวลาในการตอบสนองของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช ซึ่งใช้จำนวนจุดข้อมูล 1 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 720 จุดต่อชั่วโมงเป็นข้อมูลสำหรับทดลอง

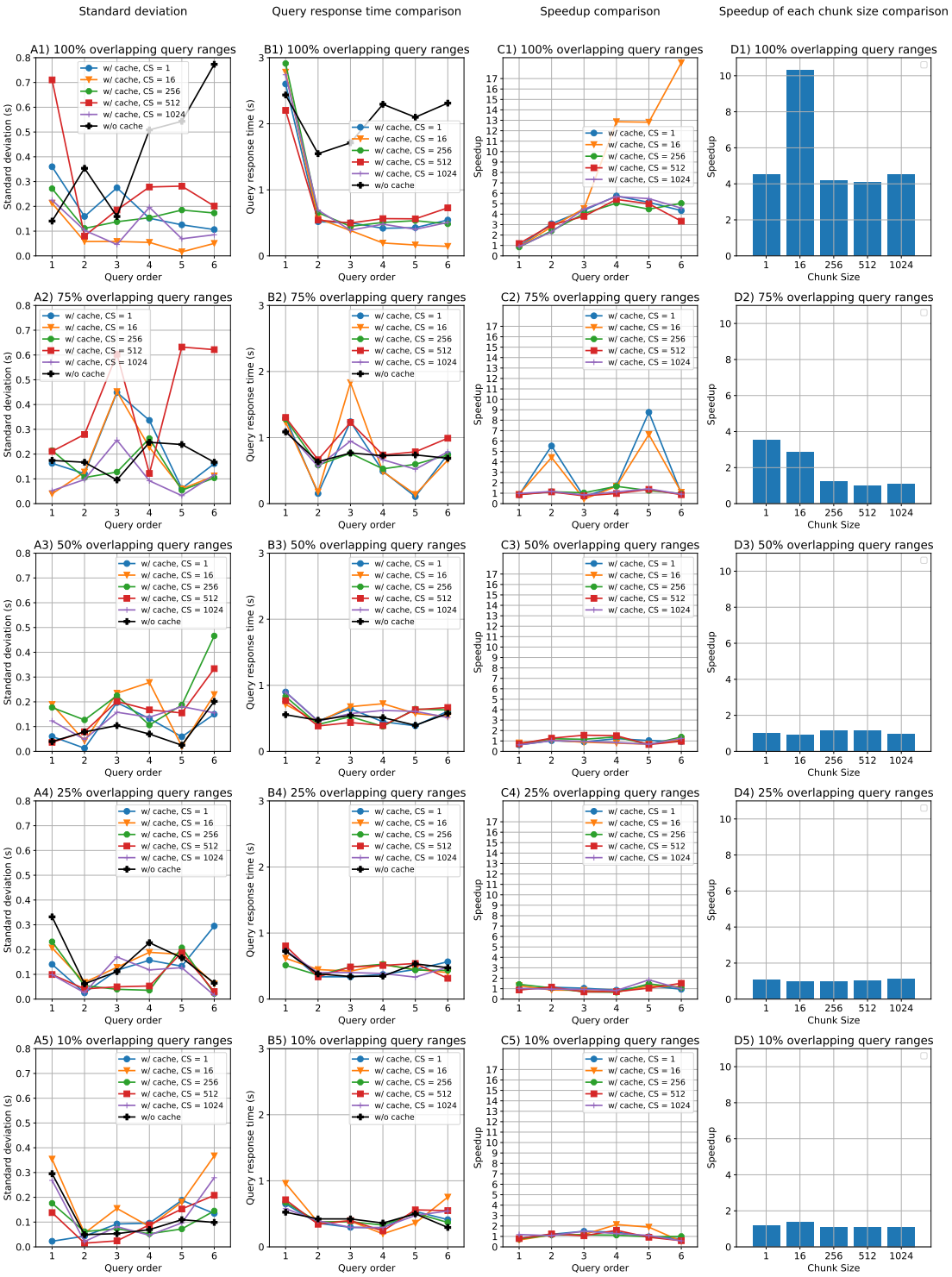
รูปที่ 4.3: ผลเปรียบเทียบเวลาในการตอบสนองของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช ซึ่งใช้จำนวนจุดข้อมูล 1 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 3,600 จุดต่อชั่วโมงเป็นข้อมูลสำหรับทดลอง



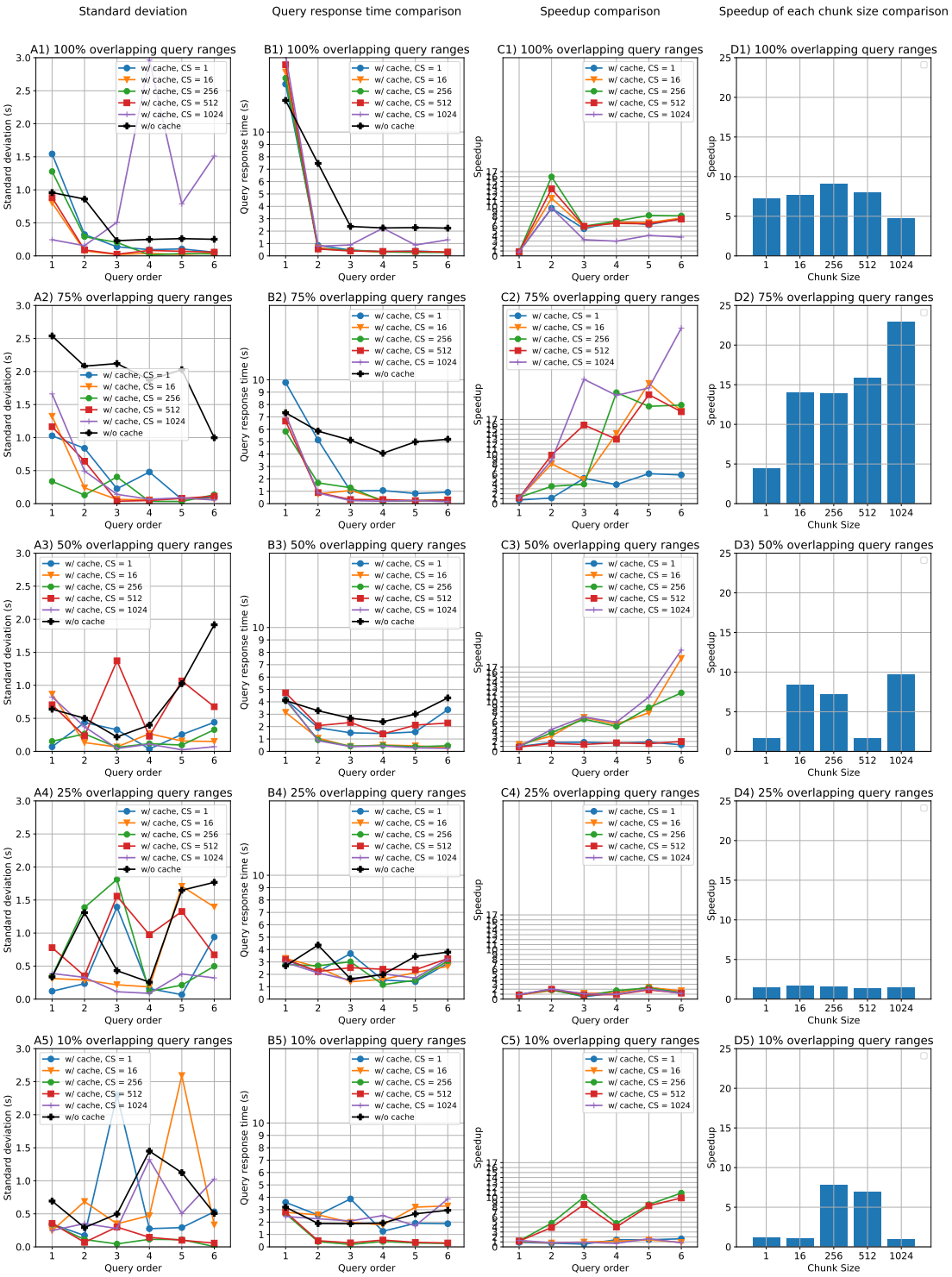
รูปที่ 4.4: การเปรียบเทียบเวลาในการตอบสนองของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช โดยใช้จำนวนจุดข้อมูล 5 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 144 จุดต่อชั่วโมงเป็นข้อมูลสำหรับทดลอง



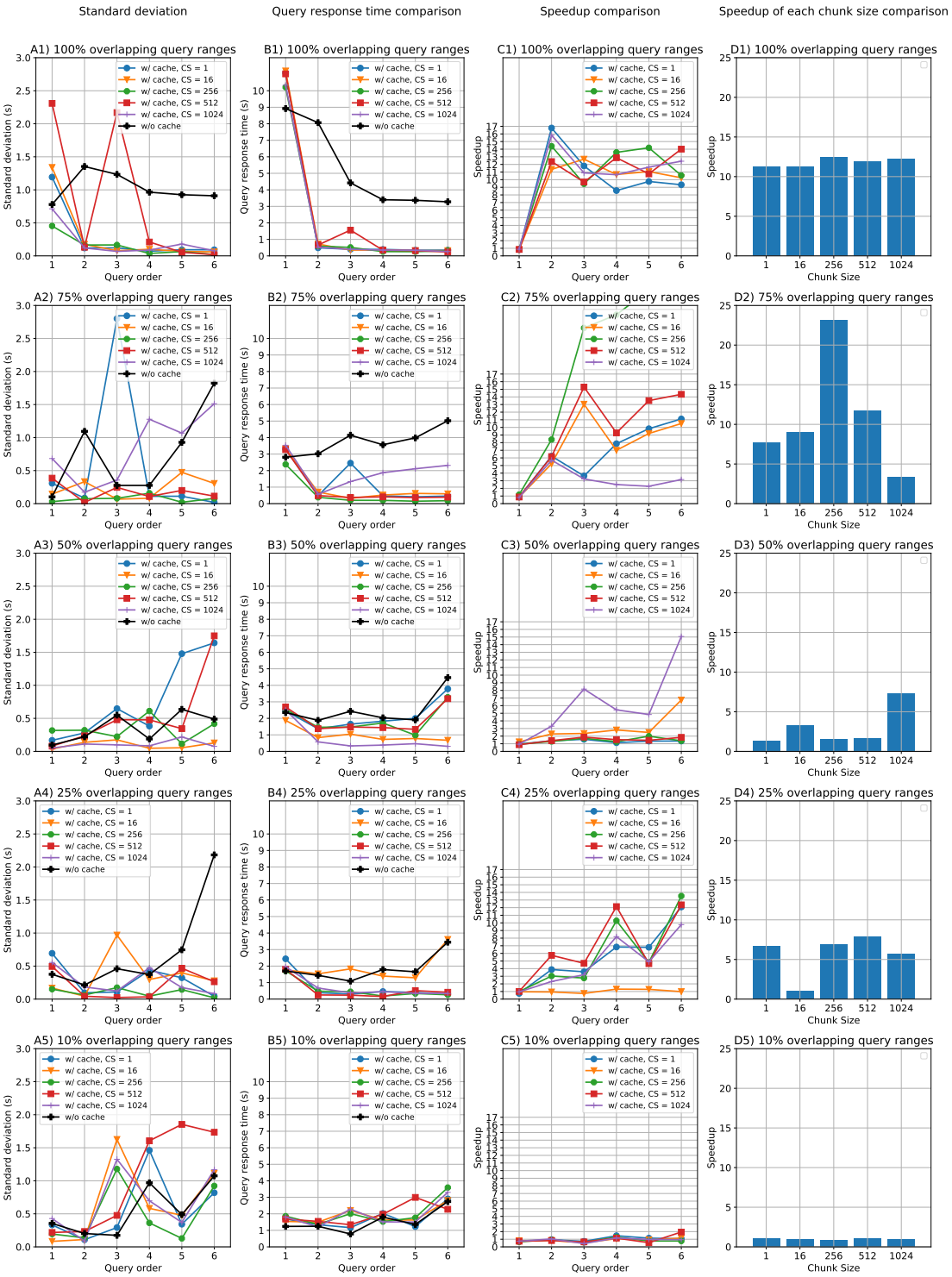
รูปที่ 4.5: การเปรียบเทียบเวลาในการตอบสนองของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช โดยใช้จำนวนจุดข้อมูล 5 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 720 จุดต่อชั่วโมงเป็นข้อมูลสำหรับทดลอง



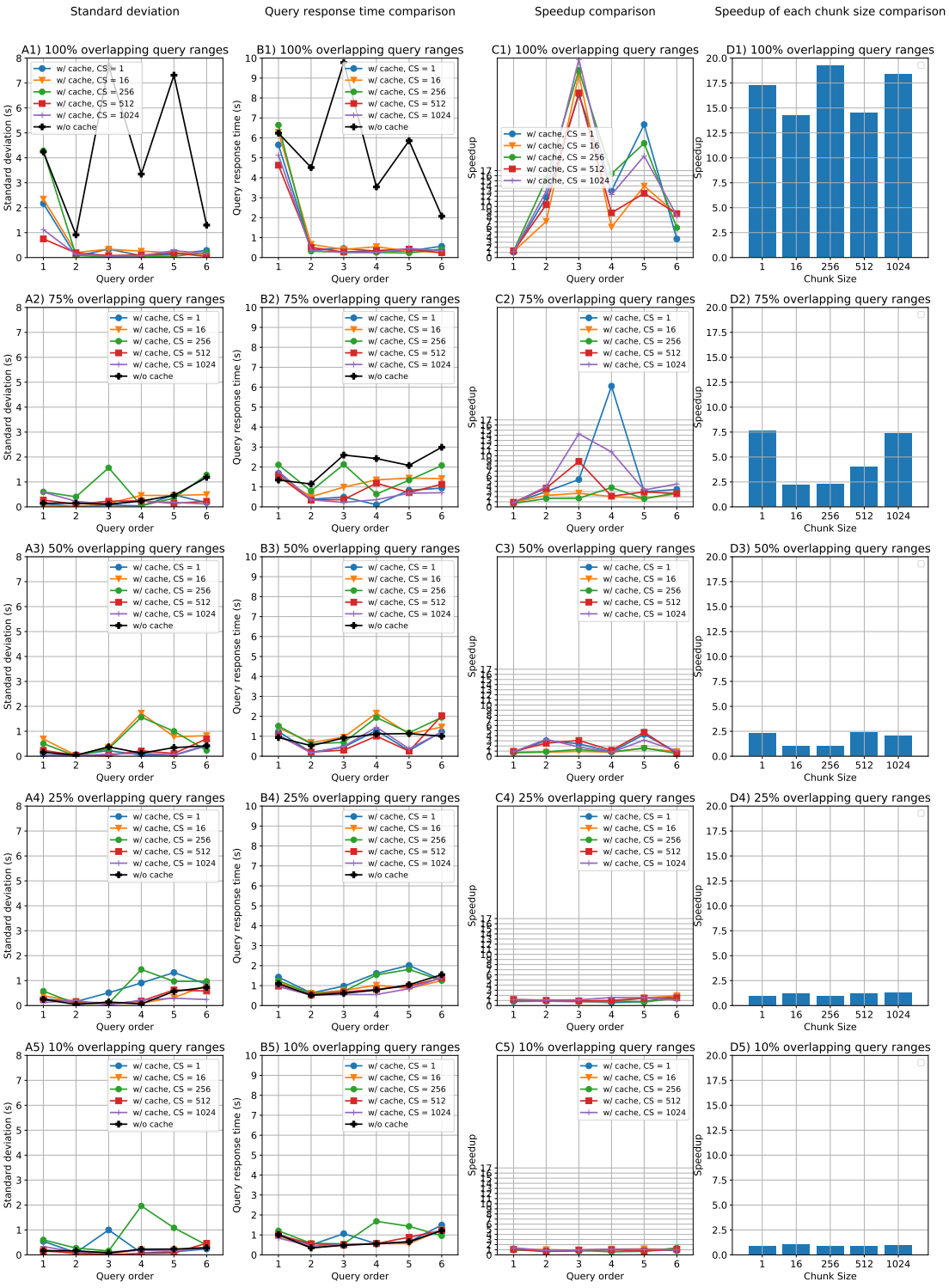
รูปที่ 4.6: การเปรียบเทียบเวลาในการตอบสนองของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช โดยใช้จำนวนจุดข้อมูล 5 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 3,600 จุดต่อชั่วโมงเป็นข้อมูลสำหรับทดลอง



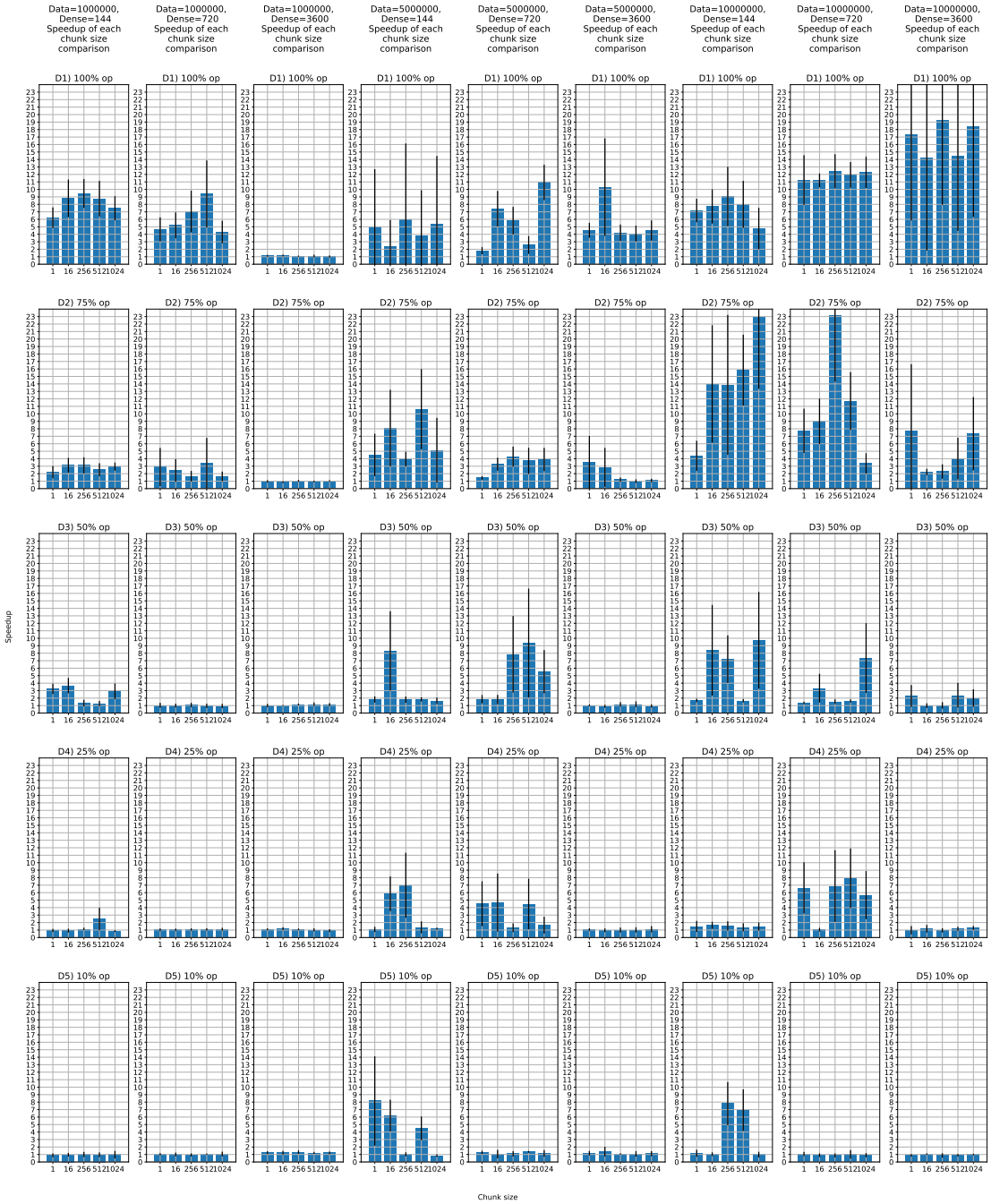
รูปที่ 4.7: การเปรียบเทียบเวลาในการตอบสนองของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช โดยใช้จำนวนจุดข้อมูล 10 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 144 จุดต่อชั่วโมงเป็นข้อมูลสำหรับทดลอง



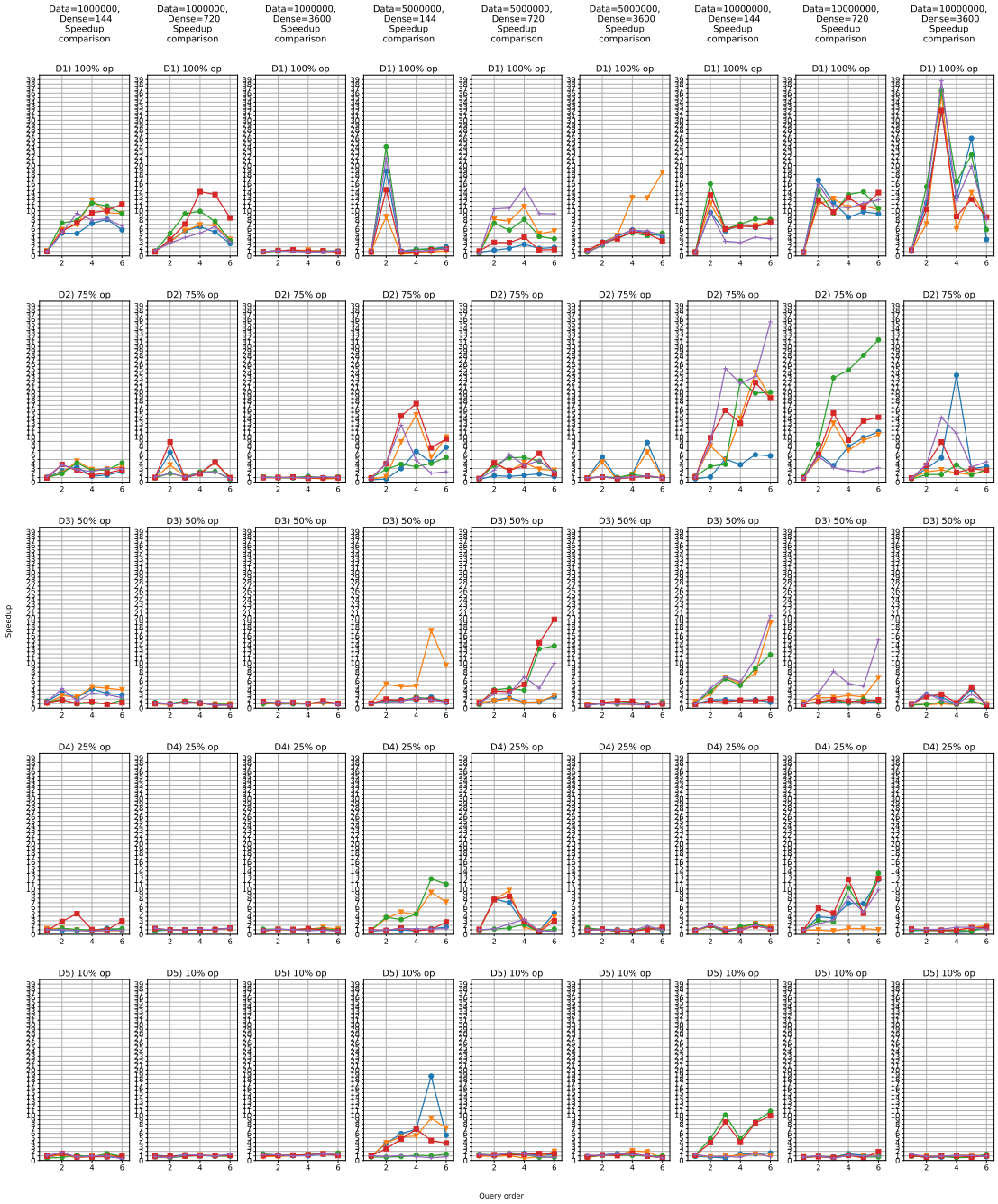
รูปที่ 4.8: การเปรียบเทียบเวลาในการตอบสนองของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช โดยใช้จำนวนจุดข้อมูล 10 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 720 จุดต่อชั่วโมงเป็นข้อมูลสำหรับทดลอง



รูปที่ 4.9: การเปรียบเทียบเวลาในการตอบสนองของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช โดยใช้จำนวนจุดข้อมูล 10 ล้านจุด ที่มีความหนาแน่นของจุดข้อมูลเป็น 3,600 จุดต่อชั่วโมงเป็นข้อมูลสำหรับทดลอง



รูปที่ 4.10: การเปรียบเทียบเวลาในการตอบสนองที่เพิ่มขึ้นในแต่ละขนาดของ chunk ทั้งหมด ของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช ทุกปัจจัย



รูปที่ 4.11: การเปรียบเทียบเวลาในการตอบสนองที่เพิ่มขึ้นของฟังก์ชัน scanner ระหว่างฐานข้อมูล OpenTSDB แบบดั้งเดิม กับฐานข้อมูล OpenTSDB ที่มีแคช ทุกปัจจัย

4.4 การวิเคราะห์ผลการทดลอง

จากตัวแปรและปัจจัยที่กำหนดที่จะนำมาทดสอบในระบบที่ได้ออกแบบไว้ ได้แก่ จำนวนจุดข้อมูล, ร้อยละของการซ้อนทับกันของการสอบถาม, ความหนาแน่นของจุดข้อมูล และ ขนาดของ chunk พบว่าสามารถวิเคราะห์ผลการทดลองแต่ละการทดลอง โดยปรับเปลี่ยนปัจจัยและตัวแปรต่างๆ ดังต่อไปนี้

1. **การแปรผันค่าจำนวนจุดข้อมูลทั้งหมด**

   ถ้ามีจำนวนจุดน้อย จะส่งผลให้ Speedup น้อย แต่ถ้ามีจำนวนจุดมากส่งผลให้ Speedup สูงขึ้น และมีความคุ้มค่าในการใช้หน่วยความจำหลักมากขึ้น

2. **การแปรผันค่าร้อยละของการซ้อนทับกันของการสอบถาม**

   ถ้ามีร้อยละของการซ้อนทับกันของการสอบถาม 100% ส่งผลให้ เกิด Speedup สูงสุด ซึ่งเป็นกรณีที่ดีที่สุด เพราะว่ามีส่วนที่ซ้อนทับกับการสอบถามก่อนหน้านี้ทั้งหมด จึงทำให้ระบบดึงข้อมูลจากแคชได้ทั้งหมด แต่ถ้าร้อยละของการซ้อนทับกันของการสอบถามน้อยลง จะทำให้ Speedup ตกลงไปด้วย ซึ่งทำให้ระบบดึงข้อมูลจากแคชได้บางส่วน และส่วนที่เหลือดึงเอาข้อมูลเอาจากฐานข้อมูล

3. **การแปรผันค่าความหนาแน่นของจุดข้อมูล**

   ณ ความหนาแน่นของจุดข้อมูล 720 จุดต่อชั่วโมง พบว่ามี Speedup ดีที่สุดจากการทดลองทุกปัจจัยและตัวแปร โดยมีประเด็นเพิ่มเติม ที่ไม่รวมอยู่ในการวิเคราะห์ดังนี้

   (a) ณ ข้อมูล 1 ล้านจุด ที่ความหนาแน่น 3,600 จุดต่อชั่วโมง fragment มีขนาดเล็กเกินไป จนทำให้ไม่สามารถเก็บลงแคชได้

   (b) ณ ข้อมูล 10 ล้านจุด ค่าส่วนเบี่ยงเบนมาตรฐานของ OpenTSDB แกว่งมากไป ทำให้ Speedup ขึ้นสูงกว่าความเป็นจริง

4. **การแปรผันค่าขนาดของ chunk**

   ขนาดของ Chunk ส่งผลต่อ Speedup ถ้าหากขนาดของ chunk น้อยหรือมากเกินไป ซึ่งขนาดของ chunk size ที่เหมาะสมและที่ได้ Speedup ดีที่สุด คือการกำหนดขนาดของ chunk คือ 256 , 16 , 512 ตามลำดับ

การกำหนดขนาดของ chunk ที่เหมาะสมนั้นขึ้นอยู่กับการนำไปใช้งาน ซึ่งขนาดของ chunk มีผลต่อขนาดพื้นที่ของ fragment และจำนวนวัตถุแคชที่จะจัดเก็บใน Memcached ด้วย

บทที่ 5

บทสรุปและข้อเสนอแนะ

ข้อมูลอนุกรมเวลาที่จัดเก็บลงบนฐานข้อมูลอนุกรมเวลานั้น มีประโยชน์ในการนำไปต่อยอดได้หลากหลาย เช่น กรณีสถานการณ์ฉุกเฉิน ระบบฐานข้อมูลที่เร็วสามารถช่วยให้การวิเคราะห์ผลทำได้เร็วยิ่งขึ้น และสามารถช่วยเหลือผู้คนได้ทันการณ์ ในแง่มุมของการสร้างระบบฐานข้อมูลที่มีความเร็วในตอบสนองสูงนั้น หลายระบบใช้หน่วยความจำหลักเป็นที่จัดเก็บข้อมูลซึ่งจะมีราคาแพง เช่น ระบบฐานข้อมูลอนุกรมเวลา Gorilla [8] และระบบจัดการข้อมูลแบบ stream สำหรับการตรวจตราที่ชื่อว่า Aurora [19] เป็นต้น ซึ่งเหมาะกับงานประเภทการตรวจตรา การใช้งานหน่วยความจำเพียงแค่บางส่วนของเนื้อที่ทั้งหมดของข้อมูลมาใช้งานเป็นแคชจะช่วยลดระยะเวลาในการสอบถามลงได้ ซึ่งเหมาะสมกับระบบที่มีความต้องการใช้ระบบฐานข้อมูลอนุกรมเวลาที่มีงบประมาณจำกัด

5.1 ภาพรวมวิทยานิพนธ์

วิทยานิพนธ์นี้เป็นการปรับปรุงพัฒนาฐานข้อมูล OpenTSDB ซึ่งเป็นฐานข้อมูลแบบเปิดเผยรหัสต้นฉบับ และประยุกต์กับตัวจัดการแคช Memcached ซึ่งจากจากทดสอบเบื้องต้นเพื่อวิเคราะห์ประเด็นปัญหา เผยให้เห็นว่าฐานข้อมูล OpenTSDB ไม่มีกลไกแคชเพื่อที่จะช่วยลดระยะเวลาในการสอบถามข้อมูลจากฐานข้อมูล OpenTSDB ในกรณีที่การสอบถามนั้นมีช่วงเวลาซ้ำซ้อนกับการสอบถามที่เคยผ่านมาแล้ว วิทยานิพนธ์นี้จึงได้นำเสนอและพัฒนากลไกแคชแบบซอฟต์แวร์สำหรับฐานข้อมูลอนุกรมเวลา OpenTSDB เพื่อที่จะลดระยะเวลาในการสอบถามข้อมูล ในกรณีของช่วงเวลาของการสอบถามข้อมูลทับซ้อนกับช่วงที่เคยสอบถามไว้แล้ว

5.2 การออกแบบกลไกแคชแบบซอฟต์แวร์สำหรับฐานข้อมูลอนุกรมเวลา OpenTSDB

เนื่องจากฐานข้อมูล OpenTSDB ไม่มีกลไกแคช ในการสอบถามข้อมูลทุกครั้งจะดึงข้อมูลจากฐานข้อมูลใหม่ทั้งหมด ทั้งๆ ที่บางช่วงของข้อมูลเคยสอบถามไว้แล้ว ซึ่งเป็นพฤติกรรมของผู้ใช้งานที่ใช้เครื่องมือแสดงผลอนุกรมเวลาทั่วไป โดยทุกๆ ครั้งที่ผู้ใช้เลื่อนขอบเขตของการแสดงผลอนุกรมเวลาหรือ ปรับเปลี่ยนตัวแปร เช่น การกำหนดค่า downsampling ที่เกี่ยวข้องกับข้อมูลอนุกรมเวลาเดิม ผู้วิจัยออกแบบให้มีการแทรกกลางขั้นตอนการดึงข้อมูลจากฐานข้อมูล HBase ดังนั้นการใช้งานกลไกแคชนี้บนฐานข้อมูล OpenTSDB ก็จะเหมือนกับฐานข้อมูล OpenTSDB แบบต้นฉบับทุกประการ โดยผู้วิจัยได้ดำเนินการดังต่อไปนี้

1. วิเคราะห์สาเหตุของการที่ทำให้ระยะเวลาการสอบถามนานในกรณีของการที่การสอบถามนั้นซ้อนทับกับการสอบถามที่เคยผ่านมาแล้ว ซึ่งการการวิเคราะห์พบว่า การแคชตรงส่วนของข้อมูลดิบ ซึ่งจะเป็นส่วนที่อยู่ก่อนนำไปประมวลผลสำหรับผู้ใช้นั้นสามารถตอบโจทย์ปัญหานี้

2. วิเคราะห์โครงสร้างข้อมูลและสถาปัตยกรรมของฐานข้อมูล OpenTSDB ที่เกี่ยวข้องการการดึงข้อมูลดิบเพียงอย่างเดียว ซึ่งเป็นสิ่งจำเป็นเพราะปัจจุบันยังไม่มีมาตรฐานสำหรับการออกแบบฐานข้อมูลอนุกรมเวลา ดังนั้นการออกแบบกลไกแคชจะอิงสถาปัตยกรรมของฐานข้อมูล OpenTSDB โดยตรง

3. ออกแบบดัชนีแคช (Cache Indexing) สำหรับการจัดการข้อมูลอนุกรมเวลา เนื่องจากลักษณะข้อมูลจำเพาะ โดยเป็นข้อมูลขนาดเล็กมาร้อยเรียงกันตามลำดับของเวลา ดังนั้นการรวมกลุ่มและการอ้างอิงไปยังกลุ่มของจุดข้อมูล ซึ่งเรียกว่า fragment นั้นมีความเหมาะสมกับข้อมูลประเภทอนุกรมเวลา

4. ออกแบบโมดูลใหม่ในการจัดการแคชสำหรับทดแทนการทำงานของโมดูลในฐานข้อมูล OpenTSDB แบบต้นฉบับที่ชื่อว่าโมดูล *Find from Database* ซึ่งโดยปกติใช้เพื่อดึงข้อมูลจากฐานข้อมูล HBase ในช่วงเวลาที่ผู้ใช้งานกำหนด

5. ภายในโมดูลใหม่นั้น มีการออกแบบโมดูล *Subquery conversion* สำหรับแปลงจากสอบถามที่มาจากผู้ใช้ให้อยู่รูปแบบของช่วงเวลาที่สั้นลง แต่มีหลายช่วง ซึ่งแต่ละช่วงก็จะมีการระบุว่าจัดเก็บอยู่ในแคชหรือไม่ และมีการออกแบบให้ใช้โมดูลเดิมที่มีอยู่แล้ว แต่ปรับเปลี่ยนช่วงเวลาที่ต้องไปดึงข้อมูลของโมดูล *Find from Database*

6. ในส่วนสุดท้ายของโมดูลใหม่ที่ออกแบบนั้น จะทำหน้าที่ในการรวมข้อมูลจากสองแหล่งที่มาคือข้อมูลที่มาจาก Memcached และ HBase ให้มีรูปแบบเหมือนกับผลลัพธ์ของโมดูล *Find from Database* ซึ่งอยู่ฐานข้อมูล OpenTSDB แบบต้นฉบับ

วิทยานิพนธ์นี้ได้ออกแบบและพัฒนาส่วนของกลไกแคชตามลำดับที่ได้กล่าวมาในหัวข้อนี้แล้ว ซึ่งจากการทดสอบพบว่าฐานข้อมูล OpenTSDB ที่มีการแทรกกลไกที่แคชที่ได้ออกแบบสามารถทำงานได้เหมือนกับ OpenTSDB แบบต้นฉบับทุกประการและให้ผลลัพธ์การทำงานที่ถูกต้อง โดยที่ผู้ใช้งานฐานข้อมูลนี้ไม่จำเป็นต้องปรับเปลี่ยนวิธีการใช้งาน

5.3 การทดสอบฐานข้อมูลอนุกรมเวลา OpenTSDB ที่มีกลไกแคชที่ได้ออกแบบ

การทดสอบแบ่งออกเป็นการทำงานสองระบบดังนี้

1. ตั้งค่าการใช้ฐานข้อมูล OpenTSDB ร่วมกับฐานข้อมูล HBase

2. ตั้งค่าการใช้ฐานข้อมูล OpenTSDB ที่มีกลไกแคชร่วมกับฐานข้อมูล HBase และตัวจัดการแคช Memcached

การทดสอบทำเพื่อทดสอบการทำงานของฐานข้อมูล OpenTSDB ที่มีกลไกแคชเพื่อเปรียบเทียบกับฐานข้อมูล OpenTSDB แบบต้นฉบับ โดยการทดสอบมีการปรับเปลี่ยนตัวแปรและปัจจัย 4 ตัวแปร

คือ จำนวนจุดข้อมูล, ร้อยละของการซ้อนทับกันของการสอบถาม, ความหนาแน่นของจุดข้อมูล และ ขนาดของ chunk โดยมีการทดสอบตามลำดับต่อไปนี้

1. ตรวจสอบความถูกต้องของการทำงานกลไกแคช
2. ติดตั้งระบบทั้งสอง
3. นำข้อมูลที่มีลักษณะตามตัวแปรที่กำหนดเข้าสู่ระบบทั้งสอง
4. ทดสอบและวัดผลระยะเวลาในการตอบสนองของการดึงข้อมูลดิบจากทั้งสองระบบ
5. บันทึกผลการทดลองลงไฟล์
6. ทำลายระบบทั้งสอง

การทดสอบการทำงานสามารถกระทำได้ครบทุกตัวแปรและปัจจัยที่ผู้วิจัยได้กำหนดไว้ และสามารถทำงานได้อย่างปกติ

5.4 สรุปผลการทดลอง

จากผลการทดลองบนข้อมูลอนุกรมเวลาที่แตกต่างกันซึ่งไม่มีการกำหนดขนาดของแคชสูงสุดโดยกำหนดของความหนาแน่นของอนุกรมเวลาที่ 144, 720, 3600 จุดต่อชั่วโมงและใช้ขนาดข้อมูลทั้งหมด 1, 5 10 ล้านจุด แต่ละสถานการณ์ที่มีช่วงเวลาที่ทับซ้อนกันแต่ละการสอบถาม ตั้งแต่น้อยไปจนถึงมากที่สุด พบว่ากรณีของการเลื่อนขอบเขตจากซ้ายไปขวาหรือขวาไปซ้ายนั้น สามารถเพิ่ม Speedup ได้ 1.5-25 เท่าเมื่อเปรียบเทียบกับฐานข้อมูลเดิมที่ไม่มีแคช

โดยการปรับค่าของขนาด Chunk ส่งผลต่อ Speedup หากขนาดของ chunk น้อยหรือมากเกินไป ซึ่งขนาดของ chunk size ที่เหมาะสมและมี Speedup ดีที่สุด คือการกำหนดขนาดของ chunk คือ 256 , 16 , 512 ตามลำดับ

5.5 อภิปรายผลการทดลอง

จากผลการทดลองทั้งหมดจากรูปที่ 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, และ 4.9 มีการปรับเปลี่ยนเพื่อให้ผลการวัดผลที่ดีขึ้นดังนี้

1. มีการตรวจสอบขนาดของ subquery ว่าเล็กกว่าขนาดของ chunk (โดยคำนวณจากเวลาตั้งต้นและสิ้นสุด หรือโดยการเพิ่มตัววัดใน Query stat ที่ชื่อว่า *isFragmentTooSmall* เพื่อแก้ปัญหา IndexOutOfBound ของอาร์เรย์ในการสอบถาม)

2. ตรวจพบว่า subquery อาจมีขนาดเล็กกว่าขนาดของ chunk size ในกรณีนี้ระบบจะอ่านข้อมูลฐานข้อมูลแทน โดยเรียกโมดูล *Find from Database* แทนการใช้แคช

3. การวัดเวลาในการตอบสนองของระบบ OpenTSDB ที่มีแคชนั้นคำนวณโดยนำเวลาแต่ละโมดูลมารวมกัน โดยมีรายการชื่อของ sheet ในไฟล์ดังนี้ `t_findCacheTime` , `t_findSpanTime` `t_buildCacheFragmentsTime` , `t_cacheMergeTime` และ `t_storeCacheTime` ซึ่งข้อเสียของการวัดเวลาแบบนี้คือ ผู้วิจัยพบว่าบางโมดูลไม่ได้เขียนเวลาการทำงานลงไปทำให้ผลการทดลองบางรายการ ใช้เวลาน้อยกว่าความเป็นจริง

4. รายการผลการทดลองทั้งหมด 15 ไฟล์ ที่ใช้โปรแกรมชุดเก่า ซึ่งไม่รวมสิ่งที่แก้ไขไปใน ข้อ 1 และ 2 โดย `exp` คือการวัดระบบของ OpenTSDB แบบมีแคช และ `opentsdb-exp` คือการวัดแบบมีแคชและไม่มีแคชรวมอยู่ด้วยกันไฟล์เดียวกันโดย `in` คือ ระยะห่างจุดข้อมูล เราสามารถนำ 3,600 หารด้วยค่า `in` เพื่อหาความหนาแน่นของข้อมูลได้ `rs` คือขนาดของ chunk และ `op` คือ ร้อยละที่ช่วงเวลาซ้อนทับกัน

เมื่อพิจารณาผลการทดลองทั้งหมดจากรูปที่ 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, และ 4.9 สามารถเปรียบเทียบ Speedup ที่เพิ่มขึ้นรวมกันได้ดังรูปที่ 4.10 และ 4.11 และสามารถอภิปรายผลการทดลองได้ดังต่อไปนี้

1. ที่ ข้อมูล 1 ล้าน จุด ความ หนา แน่น 3,600 จุด ต่อ ชั่วโมง และ ร้อย ละ ของ การ ซ้อน ทับ การ ของ แต่ละ การ สอบถาม ทุก กรณี ใกล้ เคียง กับ ระบบ OpenTSDB แบบ ไม่มี แคช เนื่องจาก กรณี `isFragmentTooSmall`

2. ที่ข้อมูล 5 ล้านจุด ความหนาแน่น 144 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 100% สาเหตุที่ Speedup ที่เพิ่มขึ้นในการสอบถามครั้งที่ 2 สูงกว่าปกติเป็นอย่างมาก จากผลการทดลองในรูปที่ 4.4 พบว่าเวลาในการตอบสนองของ OpenTSDB แต่ละครั้งมีการแกว่งของค่าสูงมาก สังเกตจากค่าส่วนเบี่ยงเบนมาตรฐาน จึงทำให้คำนวณเวลาเฉลี่ยของระบบ OpenTSDB แบบไม่มีแคชช้ากว่าปกติเป็นอย่างมาก

3. ที่ข้อมูล 10 ล้านจุด ความหนาแน่น 144 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 75% ที่ขนาด chunk คือ 1024 นั้นพบว่า มีค่าส่วนเบี่ยงเบนมาตรฐานของ OpenTSDB สูงและที่การสอบถามลำดับที่ 3-6 มีการแบ่งจำนวนของ subquery อยู่ 5-7 ส่วน ทั้งๆ ที่ผู้วิจัยคาดหวังว่าจะสถานการณ์ที่กำหนดจะได้จำนวน subquery เป็น 3 ส่วนเสมอ จึงทำให้ผลการทดลองไม่เป็นไปตามคาด

4. ที่ข้อมูล 10 ล้านจุด ความหนาแน่น 3,600 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 100% ผลการทดลองไม่สมเหตุสมผลเนื่องจากค่าส่วนเบี่ยงเบนมาตรฐานของ OpenTSDB แบบไม่มีแคชนั้นแกว่งเป็นอย่างมากทำให้ผลการทดลองในการสอบถามมีค่าสูงต่ำกว่าความเป็นจริง

5. บางการสอบถามมี speed up ที่เพิ่มขึ้นสูงผิดปกติ เกิดจากสาเหตุของการนับเวลาการทำงานรวมแต่ละโมดูล ซึ่งมีบางโมดูลไม่ได้บันทึกเวลาลงไป ทำให้ค่ารวมของเวลาน้อยกว่าความเป็นจริง ตามรายการดังต่อนี้

   (a) ที่ข้อมูล 5 ล้านจุด ความหนาแน่น 3,600 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 75% ที่ขนาด chunk คือ 1 และ 16 การสอบถามครั้งที่ 2 และ 5

   (b) ที่ข้อมูล 5 ล้านจุด ความหนาแน่น 3,600 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 100% ที่ขนาด chunk คือ 16 การสอบถามครั้งที่ 4-6

   (c) ที่ข้อมูล 10 ล้านจุด ความหนาแน่น 720 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 75% ที่ขนาด chunk คือ 256 การสอบถามครั้งที่ 3-6

   (d) ที่ข้อมูล 10 ล้านจุด ความหนาแน่น 3,600 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 75% ที่ขนาด chunk คือ 1 และ 1024 การสอบถามครั้งที่ 3-4

   (e) ที่ข้อมูล 5 ล้านจุด ความหนาแน่น 144 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 50% ที่ขนาด chunk คือ 16 การสอบถามครั้งที่ 5-6

   (f) ที่ข้อมูล 5 ล้านจุด ความหนาแน่น 720 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 50% ที่ขนาด chunk คือ 256 และ 512 การสอบถามครั้งที่ 5-6

   (g) ที่ข้อมูล 10 ล้านจุด ความหนาแน่น 144 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 50% ที่ขนาด chunk คือ 16 และ 1024 การสอบถามครั้งที่ 5-6

   (h) ที่ข้อมูล 1 ล้านจุด ความหนาแน่น 144 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 25% ที่ขนาด chunk คือ 512 การสอบถามครั้งที่ 2-3 และ 6

   (i) ที่ข้อมูล 5 ล้านจุด ความหนาแน่น 144 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 25% ที่ขนาด chunk คือ 16 และ 256 การสอบถามครั้งที่ 5-6

   (j) ที่ข้อมูล 5 ล้านจุด ความหนาแน่น 720 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 25% ที่ขนาด chunk คือ 1, 16 และ 512 การสอบถามครั้งที่ 2-3

   (k) ที่ข้อมูล 10 ล้านจุด ความหนาแน่น 720 จุดต่อชั่วโมงและร้อยละของการซ้อนทับการของแต่ละการสอบถามคือ 25% ที่ขนาด chunk คือ 1, 256, 512 และ 1024 การสอบถามครั้งที่ 4 และ 6

5.6 อุปสรรคและปัญหา

ในการทำวิทยานิพนธ์นี้มีอุปสรรคและปัญหาดังต่อไปนี้

- การพัฒนากลไกแคชมีความล่าช้าจากแผนที่ได้วางไว้เป็นอย่างมากเนื่องจากรหัสต้นฉบับที่ได้เปิดเผยไว้ของฐานข้อมูล OpenTSDB นั้นมีขนาดใหญ่มาก และเอกสารที่ประกอบการอธิบาย

รหัสต้นฉบับไม่ได้ลงรายละเอียดมากเท่าที่ควรจึงทำให้ใช้ระยะเวลาในขั้นตอนที่สำคัญที่สุดนานมากคือ การค้นหาตำแหน่งของการจัดการข้อมูลอนุกรมเวลาที่ไม่ผ่านการประมวลผล (Raw time series data) ในสถาปัตยกรรมของ OpenTSDB

- สถาปัตยกรรมฐานข้อมูล OpenTSDB ประกอบไปด้วยรหัสต้นฉบับการเขียนโปรแปรมภาษา Java ระดับต่ำ ซึ่งในปัจจุบันโปรแกรมภาษา Java ส่วนใหญ่ใช้เทคนิคที่ใช้งานได้ง่ายกว่าระดับภาษาที่ OpenTSDB ใช้งาน เช่น การเขียนโปรแกรมแบบ Asynchronous ใช้เทคนิคของ Deferred [46] ซึ่งต่อยอดแนวคิดมาจากไลบารีภาษา Python ที่ชื่อว่า Twisted [47]

- ฐานข้อมูล OpenTSDB ประกอบไปด้วยส่วนประกอบหลักๆ คือ OpenTSDB และ HBase ซึ่งใช้เทคนิคการเขียนโปรแกรมแบบ Asynchronous ทั้งโปรแกรม อีกทั้งยังมีหลายโมดูล ซึ่งการตรวจสอบข้อผิดพลาดนั้น ดำเนินไปด้วยความยากลำบากและใช้ระยะเวลานานกว่าจะพบต้นตอของสาเหตุ

- ปัญหาที่เกิดระหว่างทำการทดลองส่วนใหญ่มาจากขั้นตอนของการนำเข้าข้อมูลของฐานข้อมูล OpenTSDB นั้นไม่สามารถทำได้ทุกครั้งที่ทำการทดลอง ดังนั้นผู้ใช้วิจัยจึงใช้เวลาไปกับการค้นหาสาเหตุของปัญหาและทำให้ระยะเวลาในการทดลองนานขึ้นซึ่งไม่เป็นไปตามแผนงาน โดยปัญหานี้เกี่ยวข้องโดยตรงกับการตั้งค่าฐานข้อมูล HBase โดยตรง ซึ่งในขอบเขตของงานวิจัยนี้ไม่ตั้งค่าตัวแปรของ HBase ในการทดลองทั้งหมดใช้การตั้งค่าแบบเริ่มต้น (default)

5.7 ข้อเสนอแนะและแนวทางการพัฒนาต่อยอด

งานวิจัยชิ้นนี้ทดลองบนระบบที่ไม่มีการกระจายตัวของฐานข้อมูลเพื่อทดสอบการทำงานของกลไกแคชที่ออกแบบ รวมถึงการวัดผลระยะเวลาในส่วนของโมดูลที่เกี่ยวข้องกับข้อมูลดิบเท่านั้น และในการออกแบบต้องเพิ่มเติมโครงสร้างของข้อมูลของ OpenTSDB บางส่วนรวมถึงการออกแบบการทดลองเพื่อให้สอดคล้องกับ OpenTSDB จึงสามารถสรุปเป็นแนวทางในการต่อยอดได้ดังนี้

1. **ด้านการออกแบบและพัฒนาต่อยอด**

    - การออกแบบแต่ละฟังก์ชันของรหัสต้นฉบับของกลไกแคชที่ได้ออกแบบบนฐานข้อมูล OpenTSDB นั้นออกแบบให้แยกออกจากฟังก์ชันที่มีอยู่เดิม ดังนั้นในอนาคตจึงสามารถที่จะย้ายรหัสต้นฉบับนี้ไปยังฐานข้อมูล OpenTSDB รุ่นใหม่กว่านี้ได้อย่างสะดวกและรวดเร็ว
    - วิธีการออกแบบกลไกแคชสำหรับอนุกรมนี้สามารถนำไปประยุกต์ต่อยอดกับฐานข้อมูลอนุกรมเวลาอื่นๆ ได้
    - กลไกแคชที่ได้พัฒนานั้นรองรับเพียงคำสั่งการสอบถามพื้นฐานที่มีการกำหนด metric และ tag หนึ่งคู่เท่านั้น โดยกลไกแคชไม่ครอบคลุมความสามารถอื่นของฐานข้อมูล

OpenTSDB ซึ่งสามารถพัฒนาต่อยอดให้รองรับความสามารถอื่นๆ ของ OpenTSDB ได้

2. **ด้านการจัดการกับวัตถุแคชเพื่อให้ระบบกลไกแคชมีความสมบูรณ์มากขึ้น**

กลไกแคช มีประเด็นที่ต้องพิจารณาด้วย 2 ประการ คือ cache replacement และ cache coherence

Cache replacement คือการนำข้อมูลออกจากแคชเมื่อความจุของแคชเต็มหรือ เป็นไปตามเงื่อนไขอื่นๆ ในวิทยานิพนธ์นี้ ไม่ได้พิจารณาหรือออกแบบเกี่ยวกับ cache replacement ใดๆ เนื่องจากงานวิจัยนี้ใช้ประโยชน์จาก Memcached ที่มีการใช้ cache replacement แบบ LRU

Cache coherence คือ จัดการความเป็นหนึ่งเดียวกันของข้อมูลในเก็บอยู่ในแคชกับการอ้างอิง (consistency) โดยวิทยานิพนธ์นี้อ้างอิงจากดัชนีแคช (cache index) ที่สร้างขึ้นมาเอง ในวิทยานิพนธ์นี้ไม่ได้ออกแบบให้สามารถจัดการกับ cache coherence ทั้งหมด ซึ่งมีประเด็นที่เกี่ยวข้อง 2 ประเด็นคือ cache coherence detection และ cache coherence enforcement

โดย Memcached มีการทำ dynamic cache coherence detection ด้วยกลไก Time to live (TTL) aware LRU คือการทำ LRU แบบสนใจเวลาที่วัตถุนั้นจะหมดอายุ Time to live (TTL) และ LRU crawler ซึ่งมีการขจัดแคชที่หมดอายุออกเป็นช่วงเวลา (periodic background crawler) โดยวิทยานิพนธ์นี้กำหนด TTL เป็น 0 คือไม่มีอายุ เนื่องจากต้องการลดปัจจัยที่เกี่ยวข้องกับการทดลองประสิทธิภาพของกลไกแคชที่นำเสนอ ส่วน cache coherence enforcement ของ Memcached เป็นภาระของโปรแกรมงานประยุกต์ที่เรียกใช้ Memcached

ดังนั้นในการนำกลไกแคชไปใช้งานจริงจะมีการกำหนด TTL ของวัตถุที่จะจัดเก็บลงแคชด้วย จึงมีโอกาสที่วัตถุที่หมดอายุแล้วนั้น ยังปรากฏอยู่ในแคช เมื่อกลไก LRU ยังไม่ถูกเรียกใช้งานและยังไม่ถึงเวลาที่กลไก crawler จะลบวัตถุที่อยู่ในแคชให้ จึงทำให้โมดูล Subquery conversion นั้นเข้าใจผิดว่าข้อมูลที่ยังอยู่ในแคชยังไม่หมดอายุ ในอนาคตควรจะพัฒนา coherence enforcement เพิ่มเติม เช่น นอกจากโมดูล Subquery conversion จะตรวจสอบว่ามีวัตถุอยู่ในแคชหรือไม่ อาจจะทำการตรวจสอบ TTL ของวัตถุเหล่านั้นด้วยว่าหมดอายุแล้ว (expired) หรือไม่ ซึ่งถ้าพบว่าหมดอายุก็ให้ไปดึงข้อมูลจากฐานข้อมูลแทนโดยโมดูล Find from Database

3. **ด้านการเพิ่มประสิทธิภาพของระบบ**

- กลไกแคชที่ได้ออกแบบนั้นสามารถเพิ่มความเร็วในการทำงานได้โดยขจัดจุดที่เกิดสภาวะ Synchronous หรือคอขวด ซึ่งอยู่ที่โมดูล Store in cache โดยจะต้องรอให้จัดเก็บข้อมูลลงแคชให้เสร็จก่อนจึงจะส่งข้อมูลกลับไปให้โมดูล Merging raw data บันทึกข้อมูลลงแคชแบบ Asynchronous ได้

- หากโมดูล *Subquery Conversion* มีช่วงของ subquery ขนาดใหญ่ ซึ่งใน subquery มีจุดข้อมูลหลาย chunk แล้ว เมื่อโมดูล *Find from Cache* ทำงานแล้วมีบาง chunk ใน subquery นั้น ไม่สามารถดึงข้อมูลจาก cache ได้ ระบบที่ได้ออกแบบไว้จะทำการยกเลิกทั้ง subquery (ทุก chunk ที่อยู่ใน subquery เดียวกัน) แล้วเรียกข้อมูลทั้ง subquery จาก *Find from Database* แทน จึงทำให้ผลการทดลองในบางกรณีมีประสิทธิภาพการทำงานที่ต่ำ ในอนาคตควรที่จะแบ่ง subquery แล้วเรียกข้อมูลเฉพาะบาง chunk ที่เสียหายจากฐานข้อมูลเท่านั้น เพื่อให้ไม่เสียเวลาในการส่งและรับ request หลายรอบ
- สามารถเพิ่มประสิทธิภาพของโมดูล *Subquery Conversion* ได้โดยการรวมบางการสอบถามย่อยที่มีขนาดเล็กหลายๆ การสอบถามย่อย และพิจารณาว่าการแบ่งข้อมูลเป็นช่วงเล็กนั้นส่งผลต่อประสิทธิภาพอย่างไร ซึ่งอาจจะไม่สนใจการทดสอบย่อยเล็กๆ นั้นก็ได้ โดยรวมเป็นช่วงที่กว้างขึ้นเพื่อลดจำนวนคำร้องขอที่ส่งไปยังฐานข้อมูล HBase และ Memcached
- การแปลงข้อมูลเป็นไบต์อาร์เรย์ (Serialization) ในภาษา Java มีหลายอัลกอริทึมที่มีความเร็วสูงกว่าการแปลงข้อมูลแบบที่ผู้วิจัยเขียนซึ่งเป็นโปรแกรมแปลงอย่างง่าย [48] ดังนั้นถ้าหากเปลี่ยนเป็นอัลกอริทึมที่ดีกว่านี้ก็จะส่งผลต่อความเร็วในการตอบสนองโดยรวมเช่นกัน

4. **ด้านการเพิ่มประสิทธิภาพ เพื่อการขยายตัวในระบบขนาดใหญ่ในอนาคต**

- การทดสอบระบบโดยใช้ขนาดข้อมูลที่ใหญ่กว่านี้ และเป็นข้อมูลที่ใช้งานจริง
- สามารถทำการทดลองระบบกลไกแคชนี้แบบขยายตัวเป็นหลายเครื่องได้ เพื่อเพิ่มประสิทธิภาพโดยรวมในการให้บริการ และไม่จำเป็นต้องใช้คอมพิวเตอร์สมรรถนะดีเพียงหนึ่งเครื่องที่มีหน่วยความจำหลักสูงซึ่งมีราคาแพงกว่าการใช้คอมพิวเตอร์ทั่วไปที่มีขนาดหน่วยความจำหลักน้อยกว่าจำนวนหลายเครื่อง ระบบสามารถรวมพื้นที่หน่วยความจำหลักของทุกเครื่องได้จากความสามารถของตัวจัดการแคชแบบกระจาย Memcached จึงเหมาะสำหรับระบบที่มีงบประมาณจำกัด

5.8 ข้อแนะนำการนำไปใช้งาน

การนำไปใช้ประโยชน์นั้นสามารถนำไปใช้แทนที่ฐานข้อมูล OpenTSDB แบบต้นฉบับได้โดยที่ผู้ใช้ไม่ต้องปรับเปลี่ยนวิธีการใช้งาน และสามารถใช้ระยะเวลาในการตอบสนองต่อการสอบถามได้ดีกว่าฐานข้อมูล OpenTSDB แบบต้นฉบับ ซึ่งการติดตั้งฐานข้อมูล OpenTSDB ที่มีแคชนั้นต้องกำหนดขนาดของ chunk ให้เหมาะสมกับประเภทของข้อมูลที่จะนำไปใช้เพื่อให้ได้ประสิทธิภาพสูงสุด โดยสรุปแล้ว หากต้องการใช้ในระบบที่ครอบคลุมทุกปัจจัยและตัวแปรที่ผู้วิจัยกำหนด ค่าขนาดของ chunk ที่เหมาะสมที่สุดคือ 512 หรือสามารถใช้ค่า 256 ได้ซึ่งให้ผล Speedup ที่รองลงมาหรือขนาดที่ใกล้เคียงกัน ดังนั้นการกำหนดขนาด

ของ chunk ที่เหมาะสมนั้นขึ้นอยู่กับการนำไปใช้งาน โดยสามารถศึกษาจากผลการทดลองของวิทยานิพนธ์นี้ประกอบกับการตั้งค่าตามที่ผู้ใช้ต้องการได้

บรรณานุกรม

[1] N. Agrawal and A. Vulimiri, "Low-latency analytics on colossal data streams with SummaryStore," in *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pp. 647–664, ACM, 2017.

[2] C. Chatfield, *The Analysis of Time Series: An Introduction, Fourth Edition*. Boca Raton, FL: Chapman and Hall/CRC, 4 edition ed., 1989.

[3] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time series analysis forecasting and control*. Englewood Cliffs, N.J. : Prentice-Hall, c1994, 3rd ed., 1994.

[4] T. Dunning and E. Friedman, *Time Series Databases: New Ways to Store and Access Data*. O'Reilly Media, 1 edition ed., Dec. 2014.

[5] "Time series database." https://en.wikipedia.org/w/index.php?title=Time_series_database&oldid=725627648, June 2016. Page Version ID: 725627648.

[6] "Grafana - beautiful metrics, analytics, dashboards and monitoring!." http://grafana.org/, Sept. 2016.

[7] "Kairosdb." https://kairosdb.github.io/, sep 2016.

[8] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan, "Gorilla: A Fast, Scalable, In-memory Time Series Database," *Proc. VLDB Endow.*, vol. 8, pp. 1816–1827, Aug. 2015.

[9] "Comparison to alternatives, prometheus vs. opentsdb." https://prometheus.io/docs/introduction/comparison/#prometheus-vs.-opentsdb, July 2016.

[10] Z. Mathe, A. C. Ramo, F. Stagni, and L. Tomassetti, "Evaluation of NoSQL databases for DIRAC monitoring and beyond," *Journal of Physics: Conference Series*, vol. 664, no. 4, p. 042036, 2015.

[11] "Apache cassandra." http://cassandra.apache.org/, June 2016.

[12] "Kairosdb - custom data." https://kairosdb.github.io/docs/build/html/kairosdevelopment/CustomData.html, sep 2016.

[13] R. S. Weigel, D. M. Lindholm, A. Wilson, and J. Faden, "TSDS: high-performance merge, subset, and filter software for time series-like data," *Earth Science Informatics*, vol. 3, pp. 29–40, June 2010.

[14] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang, "In-memory big data management and processing: A survey," vol. 27, no. 7, pp. 1920–1948, 2015.

[15] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Time series management systems: A survey," vol. 29, no. 11, pp. 2581–2600, 2017.

[16] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan, "Cache-and-query for Wide Area Sensor Databases," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, (New York, NY, USA), pp. 503–514, ACM, 2003.

[17] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli, "Druid: A real-time analytical data store," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pp. 157–168, ACM.

[18] T. Wlodarczyk, "Overview of time series storage and processing in a cloud environment," in *2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 625–628, 2012.

[19] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A New Model and Architecture for Data Stream Management," *The VLDB Journal*, vol. 12, pp. 120–139, Aug. 2003.

[20] S. Nath, A. Deshpande, Y. Ke, P. B. Gibbons, B. Karp, and S. Seshan, "IrisNet: An architecture for internet-scale sensing services," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pp. 1137–1140, VLDB Endowment, 2003.

[21] "Opentsdb - a distributed, scalable monitoring system." http://opentsdb.net, Mar. 2018.

[22] N. Dimiduk and A. Khurana, *HBase in Action*. Shelter Island, NY: Manning Publications, 1 edition ed., Nov. 2012.

[23] "Rrdtool - about rrdtool." http://oss.oetiker.ch/rrdtool/, sep 2016.

[24] "Tcollector." http://opentsdb.net/docs/build/html/user_guide/utilities/tcollector.html, Sept. 2016.

[25] "Opentsdb overview." http://opentsdb.net/overview.html, Mar. 2018.

[26] "Definitions — opentsdb 2.3 documentation." http://opentsdb.net/docs/build/html/user_guide/definitions.html, June 2018.

[27] “The open group base specifications issue 7, section 4.15 seconds since the epoch.” http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_15, July 2016.

[28] “Opentsdb 2.2 documentation, aggregators, interpolation.” http://opentsdb.net/docs/build/html/user_guide/query/aggregators.html#interpolation, July 2016.

[29] M. J. Miller, “Storage class memory: The coming revolution,” June 2018.

[30] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 5th ed.

[31] A. Wiggins and J. Langston, “Enhancing the scalability of memcached,”

[32] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, “Scaling Memcache at Facebook,” pp. 385–398, 2013.

[33] “Caching with twemcache,” June 2018.

[34] “Docker - build, ship, and run any app, anywhere.” https://www.docker.com/, Mar. 2018.

[35] T. Wangthammang, “mildronize/ opentsdb-alpine-docker.” https://github.com/mildronize/opentsdb-alpine-docker, Sept. 2016.

[36] “Opentsdb - query details and stats.” http://opentsdb.net/docs/build/html/user_guide/query/stats.html#stats, sep 2016.

[37] “Hbase schema.” http://opentsdb.net/docs/build/html/user_guide/backends/hbase.html, Mar. 2018.

[38] “Java se development kit 8 downloads.” http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, June 2018.

[39] “memcached - a distributed memory object caching system.” https://memcached.org/, June 2018.

[40] “Apache hbase.” http://hbase.apache.org/, June 2018.

[41] “Docker compose | docker documentation.” https://docs.docker.com/compose/, Mar. 2018.

[42] “Dockerfile reference.” https://docs.docker.com/engine/reference/builder/, June 2018.

[43] “Docker hub.” https://hub.docker.com/, June 2018.

[44] “memcached.” https://hub.docker.com/_/memcached/, June 2018.

[45] “import — opentsdb 2.3 documentation.” http://opentsdb.net/docs/build/html/user_guide/cli/import.html, June 2018.

[46] “Opentsdb/async - building blocks for asynchronous java processing inspired by twisted’s api.” https://github.com/OpenTSDB/async, June 2018.

[47] “Twisted.” https://twistedmatrix.com, June 2018.

[48] “jvm-serializers.” https://github.com/eishay/jvm-serializers/wiki, June 2018.

[49] “A cache mechanism for opentsdb.” https://github.com/mildronize/tscache, jul 2018.

ภาคผนวก ก

รหัสต้นฉบับของกลไกแคชสำหรับ OpenTSDB

โปรแกรมส่วนเพิ่มเติมกลไกแคชของ OpenTSDB รุ่น 2.3.0 ได้เปิดเผยรหัสต้นฉบับทั้งหมดเปิดเผยใน [49]

โดยแบ่งโปรแกรมเป็น 4 โปรแกรมดังนี้

1. **โปรแกรมสำหรับการจัดการ Docker container**
2. **โปรแกรมสำหรับสร้างข้อมูลแบบสุ่มค่าสำหรับทำการทดลอง**
3. **โปรแกรมสำหรับชุดคำสั่งสำหรับการสอบถาม**
4. **โปรแกรมสำหรับสั่งการทดลอง**

ประวัติผู้เขียน

ชื่อ สกุล นายธาดา หวังธรรมมั่ง
รหัสนักศึกษา 5810120061
วุฒิการศึกษา

วุฒิ	ชื่อสถาบัน	ปีที่สำเร็จการศึกษา
วิศวกรรมศาสตรบัณฑิต เกียรตินิยมลำดับที่หนึ่ง (วิศวกรรมคอมพิวเตอร์)	มหาวิทยาลัยสงขลานครินทร์	2557

ทุนการศึกษา (ที่ได้รับในระหว่างการศึกษา)

ทุนการศึกษาผลการเรียนดีเด่นเข้าศึกษาระดับปริญญาโท ประจำปีการศึกษา 2558 จากบัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์

การตีพิมพ์เผยแพร่ผลงาน

- Thada Wangthammang and Pichaya Tandayya, “A Software Cache Mechanism for Reducing the OpenTSDB Query Time,” in proceeding of *The 18th International Symposium on Communications and Information Technologies (ISCIT 2018)*, Sukosol Hotel, Bangkok, Thailand, Sep 26 -29, 2018.