การเพิ่มประสิทธิภาพการเข้าถึงข้อมูลในงานสำรวจเชิงพื้นที่ โดยใช้เทคนิคการสร้างดรรชนี สำหรับชุดข้อมูลขนาดใหญ่ ภายใต้รูปแบบวัตถุของจาวาสคริปต์

จิรวัฒน์ ดวงแก้ว[1], บวรศักดิ์ ศรีสังสิทธิสันติ[1*], อภิวัฒน์ วิทยารัฐ[1], นราศักดิ์ บุญเทพ[1], ภูวิศสรณ์ ภูมิสรณคมณ์[1] และจิราพร ไชยวงศ์สาย[1]

Enhancing Data Retrieval Efficiency in Geospatial Surveys Using Indexing Techniques for Large–Scale JavaScript Object Notation Datasets

Jirawat Duangkaew[1], Bowonsak Srisungsittisunti[1*], Apiwat Witayarat[1], Narasak Boonthep[1], Phuwitsorn Phumsaranakhom[1] and Jirabhorn Chaiwongsai[1]

[1] Program in Computer Engineering, School of Information and Communication Technology, University of Phayao, Phayao, 56000

[*] Corresponding author: bowonsak.sr@up.ac.th

Received: September 9, 2023; Revised: September 21, 2023; Accepted: September 22, 2023

บทคัดย่อ

การใช้รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) ในฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ได้รับความนิยมเป็นอย่างมาก อย่างไรก็ตาม หากพิจารณาข้อจำกัดของ NoSQL ส่วนของการจัดทำดรรชนี (Indexing) สำหรับไฟล์ JSON ขนาดใหญ่ จึงเป็นข้อจำกัดอันท้าทายเป็นอย่างมาก โดยเฉพาะอย่างยิ่งในกรณีที่ต้องการสำรวจและเข้าถึงข้อมูลที่มีขนาดใหญ่แต่อุปกรณ์ที่ใช้ในการสำรวจหน้างานมีหน่วยความจำไม่เพียงพอสำหรับการประมวลผลไฟล์ขนาดใหญ่ ในการศึกษานี้ ได้เสนอการใช้ชุดข้อมูล JSON ในการรักษาข้อมูลในกระบวนการสำรวจทรัพยากร ซึ่งดำเนินการทดลองบนชุดข้อมูลขนาด 32 กิกะไบต์ ที่มีจำนวนข้อมูล 1,000,000 รายการ ในรูปแบบ JSON และได้ทำการจัดทำดรรชนีสองวิธีคือ การจัดทำดรรชนีแบบหนาแน่น (Dense) และแบบกระจาย (Sparse) เพื่อเพิ่มประสิทธิภาพในการเข้าถึงข้อมูล นอกจากนี้ยังได้ค้นพบขนาดตัวอย่างที่เหมาะสมสำหรับวิธีการจัดทำดรรชนีทั้งสอง ผลการศึกษาพบว่าการใช้กรณีการจัดทำดรรชนีแบบหนาแน่น ลดเวลาในการเข้าถึงข้อมูลจาก 26,869.218 มิลลิวินาที (กรณีไม่มีการจัดทำดรรชนี) ลงเหลือ 382.196 มิลลิวินาที หรือลดลงถึง 98.58% ในการเข้าถึงข้อมูลแบบหนึ่งต่อหนึ่ง และจาก 38,300.848 มิลลิวินาที (กรณีไม่มีการจัดทำดรรชนี) ลงเหลือ 1.097 มิลลิวินาที ในกรณีที่ไม่มีคำค้นหา ในทางกลับกัน การใช้การจัดทำดรรชนีแบบกระจาย ลดเวลาในการเรียกข้อมูลจาก 55,197.734 มิลลิวินาทีลงเหลือ 854.661 มิลลิวินาที หรือลดลงถึง 98.45% ในการเรียกข้อมูลแบบหนึ่งต่อกลุ่ม และจาก 47,203.253 มิลลิวินาทีลงเหลือ 0.179 มิลลิวินาที ในกรณีที่ไม่พบคำค้นหา นอกจากนี้ยังค้นพบว่าทุกช่วงขนาดส่วนตัวอย่างทั้งหมด สำหรับวิธีการจัดทำดรรชนีแบบหนาแน่น และดรรชนีแบบกระจายยังคงสามารถจัดการกับหน่วยความจำและการเข้าถึงคำหลัก (Keyword) ได้อย่างรวดเร็ว

คำสำคัญ: การจัดทำดรรชนีหนาแน่น, การทำดรรชนีกระจาย, ชุดข้อมูลขนาดใหญ่, ฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์, ไฟล์รูปแบบวัตถุของจาวาสคริปต์

---

[1] สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะเทคโนโลยีสารสนเทศและการสื่อสาร มหาวิทยาลัยพะเยา จังหวัดพะเยา 56000

Abstract

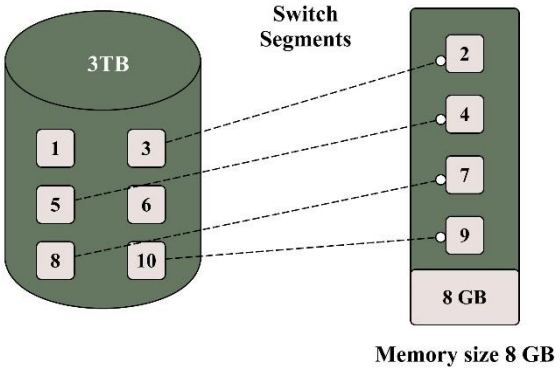
The use of JavaScript Object Notation (JSON) format as a Not only Structured Query Language (NoSQL) storage solution has grown in popularity but has presented technical challenges, particularly in indexing large-scale JSON files. In this study, we propose using JSON data sets especially in cases to survey and access large amounts of data, but the devices used for data collection have insufficient memory to process large-sized files. We conducted experiments on 32 Gigabyte data sets with 1,000,000 transactions in JSON format and implemented two types of indexing, Dense and Sparse, to enhance data access efficiency. Additionally, we identified the suitable sample size for both indexing methods. The findings indicated that the use of dense indexing decreased data retrieval time from 26,869.218 milliseconds (Non index) to 382.196 milliseconds, a reduction of 98.58% in one-to-one data retrieval and from 38,300.848 milliseconds to 1.097 when there were no keywords. In contrast, sparse indexing reduced data retrieval time from 55,197.734 milliseconds (Non index) to 854.661 milliseconds, a decrease of 98.45% in one-to-many data retrieval and from 47,203.253 milliseconds to 0.179 milliseconds when keywords were not found. Furthermore, we discovered that for both dense and sparse indexing, all sample size ranges could rapidly manage memory and access keywords.

Keywords: Dense Indexing, Sparse Indexing, Large Scale Dataset, Not only Structured Query Language, JavaScript Object Notation

บทนำ

ในการบันทึกข้อมูลจากการสำรวจเชิงพื้นที่ บางครั้งอาจพบกับสถานการณ์ที่ไม่สามารถเข้าถึงเครือข่ายได้ ดังนั้น จำเป็นต้องเก็บข้อมูลในลักษณะอื่นที่ไม่ใช่ระบบฐานข้อมูล อาทิ การบันทึกข้อมูลในรูปแบบไฟล์ข้อมูล นอกจากนี้ ข้อมูลที่บันทึกอาจมีขนาดใหญ่และปริมาณมาก ทำให้การประมวลผลด้วยอุปกรณ์สำรวจและประมวลผลหน้างานสำรวจที่มักเป็นอุปกรณ์เคลื่อนที่สะดวก เช่น โทรศัพท์มือถือ ไม่สามารถทำได้เนื่องจากขนาดของไฟล์ใหญ่กว่าหน่วยความจำที่ใช้ในการประมวลผลของอุปกรณ์

อย่างไรก็ตามไฟล์รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) เป็นเทคโนโลยีฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ซึ่งมีความยืดหยุ่นของการจัดเก็บข้อมูลทางโครงสร้างได้อย่างอิสระ จึงได้รับความนิยมอย่างกว้างขวางในปัจจุบัน การเพิ่มขึ้นของปริมาณข้อมูล ส่งผลต่อความท้าทายทางด้านเทคนิคหลายประการ โดยเฉพาะอย่างยิ่งเกี่ยวกับเทคนิคการจัดทำดรรชนีสำหรับชุดข้อมูลรูปแบบ JSON ขนาดใหญ่ อีกทั้งข้อมูลไฟล์รูปแบบ JSON มีขนาดใหญ่เกินความจุของหน่วยความจำแบบเข้าถึงแบบสุ่ม (Random Access Memory, RAM) จึงเป็นไปไม่ได้ที่จะสำรวจและสืบค้นข้อมูลทั้งหมดในการเข้าถึงเพียงครั้งเดียว แต่สามารถแบ่งข้อมูลออกเป็นส่วนเท่า ๆ กันของข้อมูล ดังแสดงในรูปที่ 1 โดยแต่ละส่วนจะถูกจัดเรียงลำดับแถวข้อมูลลงใน RAM เพื่อเข้าสู่กระบวนการสำรวจและเข้าถึงข้อมูลด้วยหน่วยประมวลผลกลาง (Central Processing Unit, CPU)



รูปที่ 1 กรณีที่ไม่มีดรรชนีสำหรับชุดข้อมูลขนาดใหญ่

การศึกษาครั้งนี้ใช้รูปแบบ JSON เพื่อตรวจสอบเทคนิคการจัดทำดรรชนีสำหรับชุดข้อมูลขนาดใหญ่ใน NoSQL ซึ่งการทบทวนการศึกษาที่เกี่ยวข้องนำเสนอความเข้าใจที่ครอบคลุมเกี่ยวกับการวิจัยที่ทันสมัยในด้านนี้ การวิจัยก่อนหน้านี้เกี่ยวกับการจัดทำดรรชนี เช่น (Chang et al., 2017) เสนอวิธีการค้นหาข้อมูลอภิพันธุ์ (Metadata) โดยอาศัยการค้นหาด้วยดรรชนีตามคำหลัก (keyword-index) ในระบบไฟล์ขนาดใหญ่ โดยใช้เทคนิคการแบ่งส่วน (Partition) ของดรรชนีได้อย่างมีประสิทธิภาพ เพื่อใช้ประโยชน์จากคุณสมบัติการกระจายของข้อมูลอภิพันธุ์ (Metadata) และความสามารถในการค้นหาข้อมูลอภิพันธุ์ (Metadata) ผลการทดลองพบว่าวิธีการแบ่งส่วนมีประสิทธิภาพมากกว่าวิธีการที่ไม่ใช้การแบ่งส่วนข้อมูลที่มีอยู่ในปัจจุบัน (Chopade & Pachghare, 2020) เสนอการจัดทำดรรชนีทางด้านการเพิ่มประสิทธิภาพการสืบค้นของฐานข้อมูล (MongoDB) และความเกี่ยวข้องในการพิสูจน์หลักฐานของฐานข้อมูล ผลการทดสอบพบว่าการจัดทำดรรชนีสามารถเพิ่มประสิทธิภาพเป็นอย่างมาก โดยการลดเวลาในการตรวจสอบเอกสาร ทำให้เป็นส่วนสำคัญของการพัฒนาแอปพลิเคชันและการวิเคราะห์ทางนิติวิทยาศาสตร์ (Yuan, J. & Liu, X., 2012) เสนอวิธีการค้นหาตัวเลือกที่ใกล้เคียงที่สุดโดยใช้ (Embedded k-Means) ซึ่งสร้างดรรชนีแบบแบ่งกลุ่มสองระดับที่มีศูนย์กลางทางคณิตศาสตร์ (Centroid) สองกลุ่ม และใช้ไฟล์ (Inverted file) เพื่อบันทึกการจัดกลุ่ม ที่ระดับค้นหา ซึ่งใช้กลยุทธ์ตัดสินใจเพื่อควบคุมคุณภาพและความเร็วในการค้นหา ผลการทดลองบนชุดข้อมูลรูปภาพ (Scale-Invariant Feature Transform, SIFT) และ (Generic Indexing of Spatial Data, GIST) แสดงประสิทธิภาพการค้นหาที่ดีและมีภาวะความจำและความซับซ้อนที่ต่ำ (Zineddine et al., 2018) เสนอโครงสร้างการจัดทำดรรชนีแบบต้นไม้แบบทวิภาค (Binary tree) แบบใหม่ เพื่อจัดระเบียบคุณสมบัติของภาพ (สี รูปร่าง พื้นผิว) สำหรับการค้นหาภาพขนาดใหญ่อย่างมีประสิทธิภาพ แนะนำการสร้างดรรชนีและอัลกอริทึมการค้นหา (K-Nearest Neighbors, KNN) โดยใช้แนวคิดคอนเทนเนอร์ (Containers) เพื่อเพิ่มประสิทธิภาพความซับซ้อนที่ประเมินจากชุดข้อมูลจริง (Jin et al., 2021) เสนอการปรับเปลี่ยนโครงสร้างดรรชนี (Balanced Tree Plus, B+Tree) และ (Log-Structured Merge-Tree, LSM-Tree) สำหรับอุปกรณ์บันทึกข้อมูล (Solid state drives, SSDs) ที่มี (Zoned Namespaces, ZNS) ซึ่งมีต้นทุนและค่าใช้จ่ายจากการเตรียมความพร้อมที่ต่ำ แต่สามารถรองรับการเขียนแบบลำดับ (Abdulkadhem, A. A. & Al-Assadi, T. A., 2019) เสนอวิธีการที่ใช้ระบบข้อมูลภูมิศาสตร์ (Geographic Information System, GIS) ในการสร้างจุดสำคัญบนถนนด้วยจุดมุมและข้อมูลมัลติมีเดีย เป็นขั้นตอนการทำงานล่วงหน้าสำหรับการค้นหาแผนที่จากภาพยนตร์วิดีโอ โดยมีเป้าหมายเพื่อให้สามารถสืบค้นมัลติมีเดียภายในสภาพแวดล้อม GIS นอกจากนี้ ยังได้ศึกษางานวิจัยที่เกี่ยวข้องกับการประยุกต์ใช้ดรรชนี เช่น (Alqatawneh, A., 2022) เสนอเทคนิคใหม่สำหรับระบบการถ่ายทอดสัญญาณแบบความถี่ศูนย์ที่ตั้งฉาก (Orthogonal frequency division multiplexing, OFDM) โดยใช้

สัญลักษณ์ส่งสู่จุดศูนย์ เพื่อส่งบิตข้อมูลเพิ่มเติม โดยไม่ส่งผลกระทบต่อความแม่นยำ ในการประมาณการช่องทางหรือประสิทธิภาพข้อผิดพลาดของระบบการตรวจจับที่ใช้เรียกว่า (Minimum Mean Square Error, MMSE) และผลจากการจำลองแสดงให้เห็นถึงการเพิ่มขึ้นของอัตราการส่งผ่านข้อมูลของระบบ และอัตราความผิดพลาดที่ต่ำเมื่อใช้ระดับสัญญาณแรงสูง (L. Tan, K. & C. Lim, K., 2019) เสนอระบบกล้องรักษาความปลอดภัยที่สามารถตรวจจับสัญญาณ (Wireless Fidelity, WiFi) พร้อมกับตัวกรองการตรวจสอบโครงสร้างของสัญญาณ WiFi มีทั้งหมด 3 ขั้นตอน โดยพิจารณาจากความแรงของสัญญาณ WiFi ในการกรองและติดตามที่อยู่ของอุปกรณ์เคลื่อนที่ (Media Access Control, MAC address) ไปยังโครงสร้างของวิดีโอ วิธีนี้ใช้ข้อมูลอภิพันธุ์ (Meta data) ของที่อยู่อุปกรณ์เคลื่อนที่ ในการกำหนดลำดับความสำคัญในการเรียกโครงสร้างของวิดีโอ เพื่อลดระยะเวลาและความยุ่งยากในการค้นหาวิดีโอของเจ้าหน้าที่ส่วนงานรักษาความปลอดภัยของสาธารณะ (S, M. & MB, S. P., 2020) เสนอดรรชนีการตรวจสอบทางด้านอารมณ์โดยอาศัยภาพคลื่นกระแสไฟฟ้า (Electroencephalogram, EEG) โดยใช้การจำแนกประเภทโดยอาศัยเพื่อนบ้านที่ใกล้ที่สุด k–NN ซึ่งทำให้สามารถวิเคราะห์ความคล้ายคลึง และความไม่คล้ายคลึงระหว่างสัญญาณของบุคคลแต่ละคนและข้อมูลเป็นมาตรฐาน ผลการวิจัยแสดงให้เห็นถึงศักยภาพในการเข้าใจความรู้สึกอารมณ์พื้นฐานของบุคคล (Zeffora, J. & Shobarani, S., 2022) เสนอวิธีการเลือกคุณสมบัติที่สามารถปรับตัวได้ที่เรียกว่า (Random Forest) โดยพิจารณาการเปลี่ยนแปลงทางโครงสร้างในชุดข้อมูล ในการประยุกต์ใช้กับข้อมูลของกล้ามเนื้อหัวใจ (Myocardial Infarction) วิธีนี้ได้ช่วยปรับปรุงความถูกต้องและป้องกันปัญหาที่เหมาะสม อีกทั้ง ยังได้ศึกษางานวิจัยที่เกี่ยวข้องกับชุดข้อมูลขนาดใหญ่ เช่น (Ma et al., 2017) เสนอวิธีการค้นหาข้อมูลสื่อหลากหลายรูปแบบที่เรียกว่า (Apache Spark) ที่อาศัยการคำนวณแบบกระจายเพื่อเพิ่มความเร็วในการค้นหา ด้วยการใช้ต้นไม้ค้นหา (Search trees) ในระบบข้อมูลขนาดในใหญ่ (Big Data) ส่งผลให้การค้นหาฐานข้อมูลสื่อมีความรวดเร็วมากยิ่งขึ้น โดยเป็นผลมาจากการจัดเก็บทางโครงสร้างของการทำดรรชนีลงในหน่วยความจำและจัดเรียงผลลัพธ์ จึงทำให้ระบบมีความถูกต้องในระดับที่ยอมรับได้และมีประสิทธิภาพ (Fathy et al., 2018) เสนอการเชื่อมต่อวัตถุกับโลกไซเบอร์ผ่านอุปกรณ์เซนเซอร์และการทำงานผ่านเครือข่ายภายใต้แนวคิดที่เรียกว่า อินเทอร์เน็ตทุกสรรพสิ่ง (Internet of Things, IoT) กล่าวถึงข้อมูลของอินเทอร์เน็ตทุกสรรพสิ่ง IoT ที่ได้จากการเก็บรวบรวมของอุปกรณ์ที่เชื่อมต่ออินเทอร์เน็ตซึ่งสามารถจัดการได้ผ่านการสร้างดรรชนี ค้นหา และจัดอันดับข้อมูล ซึ่งสำคัญสำหรับการสร้างแอปพลิเคชันที่ต้องการการเข้าถึงและค้นหาข้อมูล IoT ได้อย่างรวดเร็ว (Abdulsada et al., 2021) เสนอระบบค้นหาคำที่คล้ายคลึงในข้อมูลที่ถูกเข้ารหัส (Encrypted data) ซึ่งมีส่วนย่อยที่สั้นและแสดงผลลัพธ์ที่ถูกต้องตามคะแนนความคล้ายคลึง โดยดำเนินการแปลงเอกสารทุกตัวให้เป็นส่วนย่อยที่ถูกบีบอัด ส่วนย่อยถูกสร้างด้วยฟังก์ชันแฮช (Hash functions) ที่ใช้คำหลัก (Keyword) เพื่อรักษาความเป็นส่วนตัว ผู้ใช้ที่ได้รับอนุญาตเท่านั้นที่จะสร้างส่วนย่อยที่ถูกต้อง โดยที่คะแนนความคล้ายคลึงของเอกสารทางข้อความจะถูกประเมินโดยการคำนวณระยะแฮมมิง (Hamming distance) ผลการทดลองพบว่ารูปแบบที่เสนอมีประสิทธิภาพในการค้นหามากกว่ารูปแบบที่มีอยู่ (Yusof, M. K., 2017) เสนอการจัดการข้อมูลขนาดใหญ่โดยใช้ภาษา (Extensible Markup Language, XML) และ JSON ด้วยการทดสอบการค้นหาข้อมูลขนาดใหญ่พบว่ารูปแบบไฟล์ JSON มีความเร็วในการค้นหาและการใช้หน่วยประมวลผลกลางที่ดีกว่ารูปแบบ XML โดยรูปแบบ JSON มีศักยรูปที่เหมาะสมสำหรับเทคโนโลยีฐานข้อมูลขนาดใหญ่ในอนาคต และ (Gayathiri et al., 2019) เสนอเทคนิคสำหรับการดรรชนีข้อมูลขนาดใหญ่ (Big Data Indexing) เพื่อช่วยในการจัดเรียง การกำหนดที่อยู่ และการค้นหาข้อมูลที่ดีขึ้น โดยมีแนวทางในการค้นหาข้อมูลขนาดใหญ่ คือ (Map Reduce) สามาถช่วยลดข้อมูลที่มีขนาดใหญ่เป็นค่าที่สรุปได้ และ (Hash indexing) เป็นวิธีการสร้างกุญแจการจัดเก็บค่าของทัพเพิล (Tuples) ซึ่งทดลองกับฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์บนระบบฐานข้อมูลที่เรียกว่า มองโกดีบี (MongoDB) ทั้งแบบเดียว (Singleton) รวมถึงแบบกระจาย (Distributed) และวิเคราะห์ประสิทธิภาพในการค้นหาข้อมูลขนาดต่าง ๆ จากชุดข้อมูลทั้งหมด

ในการวิจัยนี้เสนอการเพิ่มประสิทธิภาพในการสำรวจและเข้าถึงข้อมูล โดยใช้เทคนิคการสร้างดรรชนีสำหรับชุดข้อมูลขนาดใหญ่ ภายใต้รูปแบบวัตถุของจาวาสคริปต์ ซึ่งเกี่ยวข้องกับการจัดทำดรรชนีแบบหนาแน่น (Dense Index) สำหรับการดึงข้อมูลแบบหนึ่งต่อหนึ่ง (One to one) และดรรชนีแบบกระจาย (Sparse Index) สำหรับการดึงข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-many) โดยใช้การสำรวจและเข้าถึงข้อมูลตามคำหลัก (Keyword-Index) เพื่อตรวจสอบว่าเทคนิคการจัดทำดรรชนีเหล่านี้สามารถลดเวลาในการดึงข้อมูลได้อย่างไร นอกจากนี้ ยังได้สำรวจขนาดของการแบ่งส่วน (Segment Size) ที่เหมาะสม เพื่อการดึงข้อมูลที่มีประสิทธิภาพ โดยการพัฒนาชุดคำสั่งด้วยภาษาไพธอน (Python) ท้ายสุดนี้ ผลลัพธ์พิสูจน์ให้เห็นว่าเทคนิคการจัดทำดรรชนีไฟล์รูปแบบวัตถุของจาวาสคริปต์สำหรับชุดข้อมูลขนาดใหญ่ สามารถช่วยเพิ่มประสิทธิภาพทางด้านเวลาในการดึงข้อมูลได้เป็นอย่างมาก

วัตถุประสงค์

1. เพื่อเพิ่มประสิทธิภาพของดรรชนี (Indexing) ทางด้านการสำรวจและเข้าถึงข้อมูลได้เร็วขึ้นกับลักษณะดรรชนีที่มีความหนาแน่นแบบหนึ่งต่อหนึ่ง (Dense Index) สำหรับชุดข้อมูลไฟล์รูปแบบวัตถุของจาวาสคริปต์ขนาดใหญ่

2. เพื่อเพิ่มประสิทธิภาพของดรรชนี (Indexing) ทางด้านการสำรวจและเข้าถึงข้อมูลได้เร็วขึ้นกับลักษณะดรรชนีกระจายแบบหนึ่งต่อกลุ่ม (Sparse Index) สำหรับชุดข้อมูลไฟล์รูปแบบวัตถุของจาวาสคริปต์ขนาดใหญ่

วิธีการดำเนินการวิจัย

ขั้นตอนการจำลองชุดข้อมูลขนาดใหญ่ในไฟล์รูปแบบวัตถุของจาวาสคริปต์

ในการศึกษานี้ ได้จำลองชุดข้อมูลที่ชื่อว่า BigData.json ซึ่งประกอบด้วยข้อมูลขนาด 32 กิกะไบต์ และมีข้อมูลข้อมูล 1,000,000 รายการ ชุดไฟล์ข้อมูลจัดเก็บอยู่ในรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) และประกอบด้วยฟิลด์ที่หลากหลาย เช่น Position, Latitude, Longitude, Resource_name, Location, Size, Description, และ hex of image โดยชุดข้อมูลไฟล์ Bigdata.json ของผู้วิจัยมีความยืดหยุ่นรองรับการปรับแต่งเพื่อตอบสนองความต้องการของการวิจัยเฉพาะส่วนได้ ดังรูปที่ 2 แสดงตัวอย่างการทำข้อมูลบางรายการจากชุดข้อมูล


```
{
    "position": 1,
    “latitude”:256,
    “longitude”:152,
    "Resource_name": "Forest",
    "Location": "Amazon Rainforest",
    "Size": "1,500,000 hectares",
    "Description": "The Amazon Rainforest, also known as the Amazon Jungle, is a vast tropical …….
    "hex of image": "ffd8ffe000104a464946000101010048004800O..." (32MB)
},
   …….
    “position”:2
    “position”:3
   ……..
 {
   "position": 1000000,
    “latitude”:123,
    “longitude”:244,
    "Resource_name": "Marine Reserve",
    "Location": "Great Barrier Reef",
    "Size": "344,400 square kilometers",
    "Description": "The Great Barrier Reef is the world's largest coral reef system, located off the …….
    "hex of image": "ffd8ffe000104a464946000101010048004800O..." (32MB)
}
```

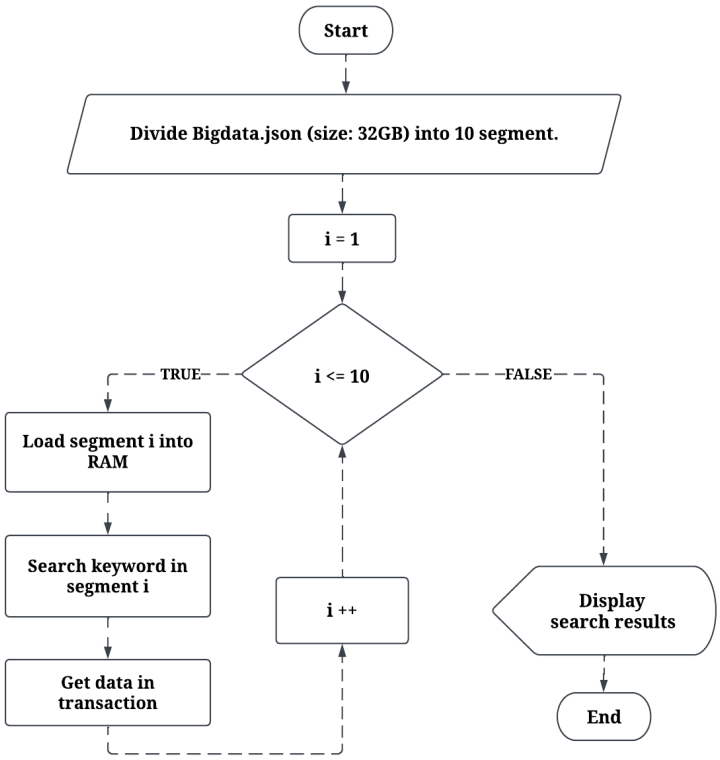

รูปที่ 2 แสดงตัวอย่างของชุดไฟล์ข้อมูล BigData.json สำหรับการสำรวจ
ภายใต้รูปแบบ JSON ที่มีขนาด 32 GB และ 1,000,000 ข้อมูล

อัลกอริทึมสำหรับการดึงข้อมูลโดยไม่มีดรรชนี

เมื่อต้องจัดการกับชุดข้อมูลขนาดใหญ่ภายใต้รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) ของฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) โดยไม่มีการจัดทำดรรชนีกับชุดข้อมูลที่เรียกว่า Bigdata.json ซึ่งมีขนาด 32 กิกะไบต์และมีข้อมูล 1,000,000 รายการ ตามที่ได้อธิบายไว้ก่อนหน้านี้ในส่วนของการจำลองชุดข้อมูลขนาดใหญ่ในไฟล์ JSON ดังรูปที่ 2 การศึกษานี้เมื่อต้องนำเข้าข้อมูลทั้งหมดไปยังหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ทั้งหมดพร้อมกันเป็นสิ่งที่ท้าทายเป็นอย่างมาก ที่จะทำให้การเข้าถึงและค้นหาข้อมูลเกิดปัญหาขึ้นได้ การศึกษานี้นำเสนอเทคนิคในการเรียกสำรวจและเข้าถึงข้อมูลโดยไม่ต้องจัดทำดรรชนีเพื่อแก้ไขปัญหานี้ รหัสเทียมสำหรับการดึงข้อมูลโดยไม่มีดรรชนีจะแสดงอยู่ในรูปที่ 3 และแสดงด้วยผังงานในรูปที่ 4 เทคนิคนี้เกี่ยวข้องกับการแบ่งข้อมูลจาก Bigdata.json ออกเป็นสิบส่วน แต่ละส่วนประกอบด้วย 3.2 กิกะไบต์ และ 100,000 รายการ ในระหว่างกระบวนการแบ่งกลุ่ม Bigdata.json ขั้นตอนนี้จะไม่เกี่ยวข้องกับการแยกไฟล์ออกเป็นสิบไฟล์แยกกันแต่จะเกี่ยวข้องกับการนำเข้าของการแบ่งส่วน (Segment size) ลงใน RAM หลังจากนั้น ระบบจะดึงข้อมูลโดยดึงข้อมูล แต่ละส่วนภายใน RAM หากข้อมูลตรงกับคำหลัก (Keyword) ที่กำหนด ระบบจะแยกและแสดงหลังจากดึงการแบ่งส่วน (Segment) ทั้งหมด การดึงข้อมูลครอบคลุมกลุ่ม Bigdata.json ทั้งหมด แม้ว่าเทคนิคนี้จะช่วยดึงและเข้าถึงข้อมูล แต่ก็มีข้อเสียที่สำคัญ เช่น การเรียกใช้ทรัพยากรของ RAM และต้องใช้พลังการประมวลผลจำนวนมากของหน่วยประมวลผลกลาง (Central Processing Unit, CPU) ส่งผลให้การสำรวจและเข้าถึงข้อมูลมีความล่าช้าเป็นอย่างมากอย่าง

```
1. Import Bigdata.json and specify the keyword to search for.
2. Divide Bigdata.json (size of 32 GB) into 10 segments.
3. For each segment (i = 1 to 10):
    A. Load the segment i into RAM.
    B. Search for specified keyword in segment i.
    C. Retrieve the relevant data from segment i.
4. Display final search result
```

รูปที่ 3 รหัสเทียมของการดึงข้อมูลโดยไม่มีดรรชนี



รูปที่ 4 ผังงานของการดึงข้อมูลโดยไม่มีดรรชนี

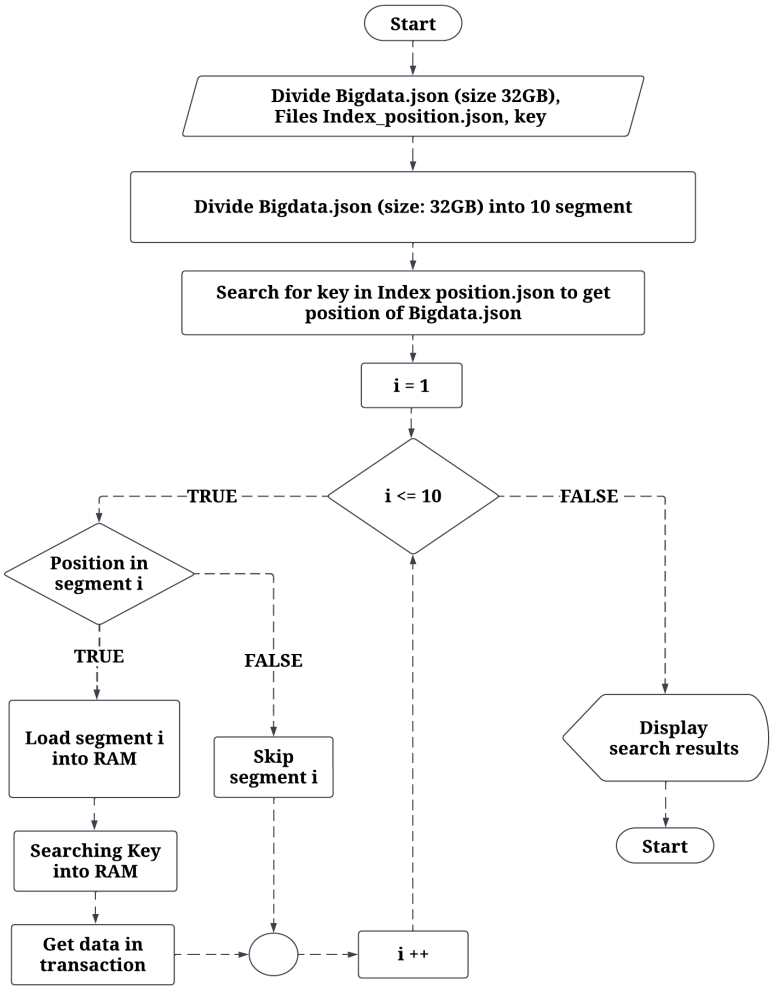
อัลกอริทึมสำหรับการสำรวจและเข้าถึงข้อมูลโดยดรรชนี

งานวิจัยนี้เสนออัลกอริทึมสำหรับการเพิ่มประสิทธิภาพเวลาในการสำรวจและเข้าถึงข้อมูลของฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ขนาดใหญ่ ภายใต้รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) โดยใช้ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse index) จากฐานข้อมูลเชิงสัมพันธ์ (Relational databases) กับชุดไฟล์ข้อมูล Bigdata.json ดังรูปที่ 5 รหัสเทียมของการสำรวจและเข้าถึงข้อมูลจากดรรชนี และดังรูปที่ 6 แผนผังลำดับงานของการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่ตามดรรชนี ซึ่งอัลกอริทึมได้มีการปรับให้เหมาะสม และแบ่งเป็นการจัดทำดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse index) สำหรับชุดข้อมูล NoSQL ซึ่งมีขนาด 32 กิกะไบต์ รวมถึงคำหลัก (Keyword) และตำแหน่ง (Position) เพื่อกำหนดสำหรับการอ้างอิงที่มีประสิทธิภาพ จากหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ไปยังชุดไฟล์ข้อมูล Bigdata.json ระหว่างกระบวนการการสำรวจด้วยเทคนิคการเข้าถึงข้อมูลตามคำหลัก (Keyword–Index) ชุดคำสั่งจะดำเนินการนำเข้าชุดข้อมูล Bigdata.json ที่มีขนาด 32 กิกะไบต์ และมีข้อมูล 1,000,000 รายการ ซึ่งชุดข้อมูลไฟล์ Bigdata.json จะถูกแบ่งออกเป็น 10 ส่วน แต่ละส่วนประกอบด้วยข้อมูล 100,000 รายการ ได้กำหนดส่วนแรกให้สอดคล้องกับตำแหน่งของข้อมูล 1 ถึง 100,000 ส่วนที่สองเป็นตำแหน่งของข้อมูล 100,001 ถึง 200,000 และดำเนินต่อไป จนถึงส่วนที่สิบแสดงตำแหน่ง 900,001 ถึง 1,000,000 ถัดไปอัลกอริทึมจะตรวจสอบตำแหน่งของข้อมูล ในแต่ละส่วนเทียบกับไฟล์ดรรชนี หากตำแหน่งอยู่ในของการแบ่งส่วน (Segment) อัลกอริทึมจะนำเข้าส่วน (Segment) ลงใน RAM และเข้าถึงข้อมูลโดยใช้คำหลัก (Keyword) เพื่อค้นหาข้อมูลที่ตรงกับคำหลัก (Keyword) ที่กำหนด เมื่อพบการจับคู่ที่ตรงกันอัลกอริทึมจะเข้าถึงข้อมูล และแสดงข้อมูลภายในกลุ่มตามผลลัพธ์ที่พบ

1. Import files index_position.json, Bigdata.json and Keyword
2. Divide Bigdata.json (size of 32 GB) into 10 segments.
3. Search for key in index position.json to get position of Bigdata.json
4. For segment i = 1 to 10
    A. If position is in segment i.
        A1. Load segment i into RAM
        A2. Search position in segment i
        A3. Retrieve the relevant data from segment i.
    B. Else, skip segment i.
5. Display search result

รูปที่ 5 รหัสเทียมของการสำรวจและเข้าถึงข้อมูลจากดรรชนี

ขนาดของกลุ่มที่ใช้ในอัลกอริทึมอาจมีการเปลี่ยนแปลงตามความเหมาะสมสำหรับชุดข้อมูลที่กำหนดในรหัสเทียมที่แสดงเป็นตัวอย่างของการกำหนดค่าดังกล่าว โดยสมมติว่าการแบ่งส่วน (Segment) มีขนาด 100,000 ข้อมูล อย่างไรก็ตาม ขนาดส่วนที่เหมาะสมที่สุดที่กำหนดโดยการทดลองจะถูกรายงานในตารางที่ 5 และ 6 เพื่อให้แน่ใจว่าผลการวิจัยมีความถูกต้อง ได้จัดเตรียมวิธีการโดยละเอียดและข้อมูลอ้างอิงที่สนับสนุนแนวทางของผู้วิจัย



รูปที่ 6 แผนผังลำดับงานของการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่ตามดรรชนี

ดรรชนีแบบหนาแน่นสำหรับชุดข้อมูลขนาดใหญ่ (Dense index in a large-scale dataset)

ในการวิจัยครั้งนี้ ได้เพิ่มประสิทธิภาพการสำรวจและเข้าถึงข้อมูลแบบหนึ่งต่อหนึ่ง (One-to-One) ในชุดข้อมูลขนาดใหญ่ ซึ่งเก็บอยู่ในรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) บนฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) การเชื่อมโยงข้อมูลแบบหนึ่งต่อหนึ่ง (One-to-One) ในที่นี้หมายถึงการเชื่อมโยงความสัมพันธ์เฉพาะคำหลัก (Keyword) ซึ่งเป็นสิ่งที่ท้าทายเป็นอย่างมาก เมื่อต้องจัดการกับข้อมูลขนาดใหญ่ ได้เสนออัลกอริทึม (Algorithm) ที่เรียกว่า การสร้างดรรชนีแบบหนาแน่น (Dense Index Algorithm) เป็นอัลกอริทึมที่ได้รับการออกแบบมา เพื่อเพิ่มประสิทธิภาพในเชิงเวลาของการสำรวจและเข้าถึงข้อมูลขนาดใหญ่ ส่วนถัดไปของงานวิจัยนี้ จะเป็นการให้คำอธิบายที่ครอบคลุมถึงอัลกอริทึมในการสร้างดรรชนีแบบหนาแน่น รวมถึงวิธีการที่ไฟล์ดรรชนีแบบหนาแน่น ถูกนำไปใช้ในการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่

อัลกอริทึมการสร้างตำแหน่งของดรรชนีแบบหนาแน่น สำหรับชุดข้อมูลขนาดใหญ่

ในการศึกษานี้ คำว่า ชุดข้อมูลขนาดใหญ่ (Large-scale dataset) หมายถึงไฟล์ชุดการจำลองที่ชื่อว่า Bigdata.json ได้อธิบายก่อนหน้านี้ในส่วนการจำลองชุดข้อมูลขนาดใหญ่ในไฟล์รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) ดังรูปที่ 2 การศึกษานี้ “ดรรชนีความหนาแน่น” คือไฟล์ที่มีชื่อว่า Dense_index_position.json ซึ่งสร้างโดยการสแกนไฟล์ Bigdata.json ครั้งเดียว เพื่อดึงข้อมูลที่ไม่ซ้ำซ้อนในฟิลด์ที่เลือกเป็นคำหลัก (Keyword) การศึกษานี้ประกอบด้วยละติจูด (Latitude) ลองจิจูด (Longitude) และตำแหน่ง (Position) ที่สอดคล้องกันในไฟล์ Bigdata.json ไฟล์ดรรชนีแบบหนาแน่นของผู้วิจัยมีโครงสร้างที่แสดงไว้ดังรูปที่ 7 เป็นการออกแบบเพื่อการอ้างอิงแบบหนึ่งต่อหนึ่ง (One to One) และการเข้าถึงข้อมูลได้อย่างมีประสิทธิภาพ ภายในโครงสร้างของไฟล์ตำแหน่งดรรชนีแบบหนาแน่นประกอบด้วยข้อมูลละติจูด (Latitude) ลองจิจูด (Longitude) และตำแหน่ง (Position) โดยตำแหน่งที่สร้างขึ้นนั้นสามารถช่วยในการเข้าถึงข้อมูลจากชุดไฟล์ Bigdata.json ที่เก็บไว้ใน RAM ได้อย่างรวดเร็ว การศึกษาไฟล์ตำแหน่งดรรชนีแบบหนาแน่นของผู้วิจัยมีขนาด 20 เมกะไบต์ และมีข้อมูล 1,000,000 รายการ


```
{
  "256,152": 1,
  "3405,11824": 2,
  "5150,128": 3,
  ...
  "123,244": 1000000
}
```


รูปที่ 7 แสดงตัวอย่างข้อมูลในชุดไฟล์ Dense index position.json

อัลกอริทึมการใช้งาน Dense_index_position.json เพื่อสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่

สำหรับชุดไฟล์ดรรชนีแบบหนาแน่น (Dense_index_position.json) สามารถช่วยในการสำรวจและเข้าถึงฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ขนาดใหญ่ ภายใต้รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) ตามที่อธิบายไว้ก่อนหน้านี้ส่วนของหัวข้อ สร้างการจัดทำตำแหน่งของดรรชนีแบบหนาแน่น สำหรับชุดข้อมูลขนาดใหญ่ ซึ่งขั้นตอนเริ่มต้นโดยการนำเข้าไฟล์ดรรชนีแบบหนาแน่น (Dense_index_position.json) และชุดข้อมูล Bigdata.json ที่ได้รับการกำหนดให้ละติจูด (Latitude) ลองจิจูด (Longitude) เป็นคำหลัก (Keyword) สำหรับการสำรวจและเข้าถึงข้อมูล โดยมีกระบวนการทำงานดังรูปที่ 7 เมื่อดำเนินการเรียกดูใน

ชุดดรรชนีแบบหนาแน่น (Dense_index_position.json) เมื่อทราบตำแหน่ง (Position) ที่สอดคล้องกับคำหลัก (Keyword) ที่กำหนดซึ่งขนาดของไฟล์ที่ 20 เมกะไบต์ ทำให้มีประสิทธิภาพสูงในการเข้าถึงได้อย่างรวดเร็ว อย่างไรก็ตาม แต่ละคำหลัก (Keyword) เมื่อทราบตำแหน่ง (Position) เดียวในไฟล์ดรรชนีแบบหนาแน่น (Dense_index_position.json) จึงต้องการนำเข้าเฉพาะส่วนหนึ่งของดรรชนีแบบหนาแน่น (Dense_index_position.json) ทั้งหมด ลงในหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) เพื่อการเข้าถึงข้อมูล ในทางกลับกัน เมื่อคำหลัก (Keyword) ไม่ตรงกับการทำข้อมูลใด ๆ ในชุดไฟล์ดรรชนีแบบหนาแน่น (Dense_index_position.json) จะไม่ได้รับตำแหน่งใด ๆ และดังนั้นจึงไม่มีส่วนของดรรชนีแบบหนาแน่น (Dense_index_position.json) นำเข้าลงใน RAM

ดรรชนีแบบกระจาย สำหรับชุดข้อมูลขนาดใหญ่

ในการวิจัยครั้งนี้ ได้เพิ่มประสิทธิภาพในการสำรวจและเข้าถึงข้อมูลแบบหนึ่งต่อกลุ่ม (One–to–May) ในชุดข้อมูลขนาดใหญ่ ซึ่งเก็บอยู่ในรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) บนฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) การเชื่อมโยงข้อมูลแบบหนึ่งต่อกลุ่ม (One–to–Many) ในที่นี้หมายถึงความสัมพันธ์เฉพาะคำหลัก (Keyword) สามารถเชื่อมโยงกับกลุ่มข้อมูลคำหลักที่เหมือนกัน ซึ่งเป็นสิ่งที่ท้าทายเป็นอย่างมาก เมื่อต้องจัดการกับข้อมูลขนาดใหญ่ ได้เสนออัลกอริทึม (Algorithm) ที่เรียกว่า ดรรชนีแบบกระจาย (Sparse Index Algorithm) เป็นอัลกอริทึมที่ได้รับการออกแบบมา เพื่อเพิ่มประสิทธิภาพในเชิงเวลาของการสำรวจและเข้าถึงข้อมูลขนาดใหญ่ ส่วนถัดไปของงานวิจัยนี้ จะเป็นการให้คำอธิบายที่ครอบคลุมถึงอัลกอริทึมในการสร้างดรรชนีแบบกระจาย รวมถึงวิธีการที่ไฟล์ดรรชนีแบบกระจาย ถูกนำไปใช้ในการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่

อัลกอริทึมการสร้างตำแหน่งของดรรชนีแบบกระจาย สำหรับชุดข้อมูลขนาดใหญ่

การศึกษานี้ให้คำจำกัดความของชุดข้อมูลขนาดใหญ่เป็นไฟล์ Bigdata.json ที่อธิบายไว้ในส่วนของการจำลองชุดข้อมูลขนาดใหญ่ในไฟล์รูปแบบวัตถุของจาวาสคริปต์ ดังรูปที่ 2 ได้ทำการออกแบบและพัฒนาชุดคำสั่งสร้างไฟล์ดรรชนีแบบกระจาย (Sparse Index) ที่เรียกว่า Sparse_index_position.json จากชุดข้อมูลไฟล์ Bigdata.json โดยได้มาจากการสแกนหนึ่งครั้งเพื่อแยกข้อมูลที่ไม่ซ้ำกัน โดยเลือกใช้ชื่อแหล่งที่มา (Resource_name) เป็นฟิลด์คำหลัก (Keyword) ชุดไฟล์ดรรชนีแบบกระจาย (Sparse Index) นี้ ดังแสดงในรูปที่ 8 ได้ออกแบบและพัฒนาเพื่อเพิ่มประสิทธิภาพเชิงเวลาในการสำรวจและเข้าถึงข้อมูลแบบหนึ่งต่อกลุ่ม (One–to–Many) โดยจะมีทั้งชื่อแหล่งที่มา (Resource_name) และข้อมูลตำแหน่ง (Position) ที่เกี่ยวข้องโดยที่ชื่อแหล่งที่มา (Resource_name) อาจมีหลายตำแหน่ง (Position) ข้อมูลในไฟล์ดรรชนีของผู้วิจัยสามารถช่วยให้การอ้างอิงจาก RAM ไปยังข้อมูลภายในชุดไฟล์ Bigdata.json ได้อย่างมีประสิทธิภาพเชิงเวลา เมื่อมีการเรียกใช้งานชุดไฟล์ดรรชนีแบบกระจาย (Sparse Index) โปรดทราบว่าไฟล์ดรรชนีแบบกระจาย (Sparse Index) ที่เรียกว่า Sparse_index_position.json ของผู้วิจัยมีขนาดไฟล์ 8 เมกะไบต์ และมีข้อมูล 5,146 รายการ


```
{
   "Forest": [46163, 66867, 88916, 94633, 130606, 150253, 185197, 189917, 214931, 999999],
   ……
   " Marine Reserve": [46153, 59781, 93151, 94460, 101100, 172271, 244064, 368397, 439620, 888888]
}
```


รูปที่ 8 แสดงตัวอย่างข้อมูลในชุดไฟล์ Sparse index position.json

อัลกอริทึมการใช้งาน Sparse_index_position.json เพื่อสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่

ในการสำรวจและเข้าถึงข้อมูลรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) บนฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ขนาดใหญ่ สามารถใช้ไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) ดังแสดงในรูปที่ 8 ไฟล์ดรรชนีนี้เหมาะอย่างยิ่งสำหรับการสำรวจและการเข้าถึงแบบหนึ่งต่อกลุ่ม (One–to–Many) โดยเริ่มจากการนำเข้าข้อมูล ไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) ประกอบด้วยชื่อแหล่งที่มา (Resource_name) และตำแหน่ง (Position) ที่สอดคล้องกัน ซึ่งทำหน้าที่เป็นจุดอ้างอิงไปยังข้อมูลภายในไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) ที่จัดเก็บไว้ใน RAM ซึ่งนำไปสู่เวลาในการค้นหาที่มีประสิทธิภาพ ส่วนรูปที่ 2 แสดงกระบวนการที่เริ่มต้นด้วยการนำเข้าไฟล์ Bigdata.json ในบริบทนี้ ชื่อแหล่งที่มา (Resource_name) คือคำหลักสำหรับการสำรวจและเข้าถึงข้อมูล เมื่อดำเนินการเรียกข้อมูลโดยใช้ไฟล์ (Sparse_index_position.json) คำสั่งจะเข้าถึงตำแหน่งที่เกี่ยวข้องกับคำหลัก (Keyword) ที่กำหนด โปรดทราบว่าดรรชนีแบบกระจาย (Sparse_index_position.json) ของผู้วิจัยมีขนาดค่อนข้างเล็กเพียง 8 เมกะไบต์ ซึ่งมีส่วนช่วยให้ประสิทธิรูปที่โดดเด่นในการเข้าถึงข้อมูลอย่างรวดเร็ว อย่างไรก็ตาม การเข้าถึงคำหลัก (Keyword) เมื่อทราบตำแหน่ง (Position) หลายตำแหน่งจากภายในไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) ในกรณีเช่นนี้ ระบบต้องนำเข้าดรรชนีหลายส่วน (Segment) ลงใน RAM เพื่อแยกข้อมูลที่จำเป็น ในทางกลับกัน หากคำหลัก (Keyword) ไม่ตรงกับข้อมูลใด ๆ ในไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) ชุดคำสั่งจะไม่เข้าถึงตำแหน่งนั้น ๆ และเป็นผลให้ RAM ไม่ต้องนำเข้าส่วนดรรชนีใด ๆ

ผลการศึกษา

ในการศึกษานี้ได้ประเมินประสิทธิภาพของเทคนิคการทำดรรชนีที่พัฒนาขึ้นใหม่ สำหรับฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) จัดเก็บข้อมูลภายใต้ไฟล์รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) ของการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่ โดยเฉพาะอย่างยิ่ง เมื่อเปรียบเทียบประสิทธิภาพของดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) ทั้งที่มีและไม่มีดรรชนี ผู้วิจัยทดลองบนระบบคอมพิวเตอร์ที่มีหน่วยประมวลผลกลาง (Central Processing Unit, CPU) ของ Intel™ ที่มีความเร็วสูงสุดถึง 4.90 กิกะเฮิร์ตซ์ พร้อมกับหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ขนาด 8 กิกะไบต์ และโซลิดสเตตไดรฟ์ (Solid State Drive, SSD) ขนาด 500 กิกะไบต์ ได้นำเข้าชุดข้อมูลขนาดใหญ่ที่เรียกว่า Bigdata.json ตามที่อธิบายไว้ในส่วนการจำลองชุดข้อมูลขนาดใหญ่ในไฟล์รูปแบบวัตถุของจาวาสคริปต์ ดังรูปที่ 2 จากไฟล์ Bigdata.json เพื่อให้ได้ตัวอย่างที่เป็นตัวแทน ได้สุ่มเลือกข้อมูล 524 รายการ จากฟิลด์ละติจูด (Latitude) ลองจิจูด (Longitude) และ ชื่อแหล่งที่มา (Resource_name) ซึ่งคิดเป็น 0.05% ของข้อมูลจำลองทั้งหมด และทดสอบอัลกอริทึมดังกล่าวซ้ำสามครั้ง การศึกษานี้มีวัตถุประสงค์เพื่อตรวจสอบประสิทธิภาพเวลาในการสำรวจและเข้าถึงข้อมูล โดยมีและไม่มีดรรชนีรวมถึงมีข้อมูลหรือไม่มีข้อมูลที่ระบุภายในไฟล์ Bigdata.json โดยมีผลลัพธ์จากการทดลองดังนี้

ในการทดลองครั้งแรก ได้ประเมินประสิทธิภาพของการเข้าถึงข้อมูลโดยการใช้ดรรชนีแบบหนาแน่น (Dense Index) ในการสำรวจและเข้าถึงข้อมูลจากไฟล์ Bigdata.json โดยเปรียบเทียบประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลแบบหนึ่งต่อหนึ่ง (One to one) ทั้งที่มีดรรชนีแบบหนาแน่น (Dense Index) และไม่มีดรรชนี (Non Index) เป็นที่น่าสังเกตว่าคำหลัก (Keyword) ที่เลือกแบบสุ่มทั้งหมดมีอยู่ในไฟล์ Bigdata.json การทดลองนี้มีวัตถุประสงค์เพื่อตรวจสอบประสิทธิภาพเวลาในการสำรวจและเข้าถึงข้อมูลของการใช้ดรรชนีแบบหนาแน่น (Dense Index) เปรียบเทียบกับการเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index) โดยเน้นที่การมีอยู่ของคำหลักภายในไฟล์ Bigdata.json ดังแสดงใน

ตารางที่ 1 ของผลการทดสอบที่เกี่ยวข้องกับการเรียกคำหลัก 524 คำ โดยใช้ดรรชนีแบบหนาแน่น (Dense Index) ดำเนินการศึกษาทั้งหมดสามรอบ โดยนำเสนอเวลาเฉลี่ยในการเข้าถึงข้อมูลต่อคำหลัก (Keyword) การค้นพบนี้แสดงให้เห็นว่าการใช้ดรรชนีแบบหนาแน่น (Dense Index) สามารถช่วยลดเวลาในการเข้าถึงข้อมูลเมื่อเทียบกับไม่มีดรรชนี (Non Index) ผลการทดสอบพบว่า ดรรชนีแบบหนาแน่น (Dense Index) ใช้เวลาเฉลี่ยต่อคำหลักในการเข้าถึงตำแหน่งข้อมูลเท่ากับ 375.721 มิลลิวินาที ส่วนการเข้าถึงไฟล์ Bigdata.json โดยการแบ่งส่วนข้อมูล (Segment) ใช้เวลาเฉลี่ยต่อคำหลักเท่ากับ 6.476 มิลลิวินาที นอกจากนี้ เมื่อพิจารณาการเข้าถึงข้อมูลทั้งหมดด้วยการใช้ดรรชนีแบบหนาแน่น (Dense Index) เวลาที่ใช้เฉลี่ยต่อคำหลักเท่ากับ 382.196 มิลลิวินาที หากไม่มีดรรชนี (Non Index) ใช้เวลาในการเข้าถึงข้อมูลเฉลี่ยต่อคำหลักเท่ากับ 26,869.218 มิลลิวินาที ในการทดสอบซ้ำทั้งสามรอบของ 524 คำหลัก พบว่าการเข้าถึงข้อมูลโดยใช้ดรรชนีแบบหนาแน่น (Dense Index) ใช้เวลาเฉลี่ยเท่ากับ 200,270.949 มิลลิวินาที ในขณะที่การเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index) ใช้เวลาโดยเฉลี่ยเท่ากับ 14,079,470.37 มิลลิวินาที ผลการทดสอบแสดงให้เห็นถึงประสิทธิภาพทางด้านเวลาของ ดรรชนีแบบหนาแน่น (Dense Index) ที่ดีขึ้นเป็นอย่างมาก เมื่อเทียบกับไม่มีดรรชนี (Non Index) เนื่องจากสามารถข้ามข้อมูลที่ไม่จำเป็นและเข้าถึงข้อมูลเพียงครั้งเดียว นอกจากนี้ ยังนำเข้าข้อมูลลงใน RAM หนึ่งครั้งต่อคำหลัก (Keyword) ช่วยลดเวลาในการเข้าถึงข้อมูลและประมวลผลข้อมูล ในทางตรงกันข้าม การเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index) จะนำเข้าข้อมูลคำหลัก (Keyword) ทั้งหมด ซึ่งนำไปสู่การสูญเสียทรัพยากรหน่วยความจำอย่างมากและส่งผลกระทบต่อเวลาในการเข้าถึงข้อมูลที่เพิ่มขึ้นตามขนาดข้อมูล

ตารางที่ 1 การเปรียบเทียบประสิทธิภาพของการสำรวจและเวลาในการเข้าถึงไฟล์ข้อมูล Bigdata.json โดยใช้ดรรชนีแบบหนาแน่น และไม่มีการจัดทำดรรชนี

การประเมินประสิทธิภาพ (มิลลิวินาที)	**524 คำหลัก**				**ค่าเฉลี่ย**
	รอบที่ 1	**รอบที่ 2**	**รอบที่ 3**	**ค่าเฉลี่ย**	**1 คำหลัก**
เวลาการเข้าถึงข้อมูล Index ด้วย Dense Index	197,367.708	197,722.472	195,542.527	196,877.569	375.721
เวลาการเข้าถึงข้อมูล Segment ด้วย Dense index	3,388.343	3,388.343	3,388.343	3,393.38	6.476
เวลาการเข้าถึงข้อมูล Dense index ทั้งหมด	200,756.5	201,112.118	198,944.677	200,270.949	382.196
เวลาการเข้าถึงข้อมูล กรณีไม่มีดรรชนี	14,091,572.26	14,087,301.7	14,059,537.15	14,079,470.37	26,869.218

การทดลองครั้งที่สอง ได้ประเมินประสิทธิภาพของการเข้าถึงข้อมูลแบบใช้ดรรชนีแบบหนาแน่น (Dense Index) ในการสำรวจและเข้าถึงข้อมูลจากไฟล์ Bigdata.json โดยเปรียบเทียบประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลแบบหนึ่งต่อหนึ่ง (One to one)ทั้งที่มีดรรชนีแบบหนาแน่น (Dense Index และประสิทธิภาพของการเข้าถึงข้อมูลแบบไม่มีดรรชนี (Non Index) โดยไฟล์ดรรชนีดังกล่าวจะไม่พบคำหลัก (Keyword not found) อันเนื่องมาจากใช้เทคนิคการเลือกแบบสุ่มทั้งหมด การศึกษาให้ผลลัพธ์ของการทดลองดังแสดงในตารางที่ 2 ของสถานการณ์ที่ไม่พบคำหลัก (Keyword not found) เวลาในการสำรวจและเข้าถึงข้อมูล ซึ่งจำกัดอยู่ที่เวลาที่ใช้ในการเข้าถึงข้อมูลในไฟล์ดรรชนีแบบหนาแน่น (Dense Index) และไม่เกี่ยวข้องกับการเข้าถึงข้อมูลใน RAM นี่เป็นเพราะไม่มีส่วนที่เกี่ยวข้องที่จะนำเข้าและไม่มีตำแหน่ง (Position) ในดรรชนีแบบหนาแน่น (Dense Index) ที่ต้องการเข้าถึงข้อมูล ได้ทำการทดลองโดยใช้

คำหลักของดรรชนีแบบหนาแน่น (Dense Index) 524 คำหลัก ผลลัพธ์แสดงให้เห็นถึงประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลโดยเฉลี่ยต่อคำหลักเท่ากับ 1.097 มิลลิวินาที โดยมีเวลาเข้าถึงข้อมูลเฉลี่ยสำหรับทั้งสามรอบเท่ากับ 574.767 มิลลิวินาที ในทางตรงกันข้าม วิธีการเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index) ในกรณีที่ไม่พบนั้นทดสอบกับ 524 คำหลัก ผลลัพธ์พบว่าเวลาในการเข้าถึงข้อมูลโดยเฉลี่ยต่อคำหลักเท่ากับ 38,300.848 มิลลิวินาที โดยมีเวลาการเข้าถึงข้อมูลเฉลี่ยสำหรับทั้งสามรอบเท่ากับ 20,069,244.44 มิลลิวินาที ผลลัพธ์พบว่าการเข้าถึงข้อมูลโดยดรรชนีหนาแน่น (Dense Index) ในกรณีที่ไม่พบคำหลัก (Keyword not found) มีประสิทธิภาพมากกว่า กรณีที่ไม่มีดรรชนี (Non Index) เนื่องจากข้อมูลที่ไม่จำเป็นถูกข้ามไป และข้อมูลถูกเข้าถึงเพียงครั้งเดียวต่อคำหลัก (Keyword) ทำให้ลดเวลาในการเข้าถึงข้อมูลเฉลี่ยต่อคำหลัก (Keyword) ลดลงเหลือเพียง 1.097 มิลลิวินาที ในทางกลับกัน การสำรวจและเข้าถึงข้อมูลโดยไม่มีการจัดทำดรรชนีในกรณีที่ไม่พบคำหลัก (Keyword not found) ทำให้ใช้เวลาในการสำรวจและเข้าถึงข้อมูลนานกว่ามาก โดยเวลาเข้าถึงข้อมูลเฉลี่ยต่อคำหลักเท่ากับ 38,300.848 มิลลิวินาที ส่งผลกระทบสูงต่อ RAM โดยเฉพาะอย่างยิ่งในกระบวนการนำเข้าคำหลักทั้งหมด และเวลาในการเข้าถึงข้อมูล

ตารางที่ 2 การเปรียบเทียบประสิทธิภาพของการเข้าถึงข้อมูล โดยใช้ดรรชนีแบบหนาแน่นและไม่มีดรรชนีและกรณีที่ไม่มีคำหลักในไฟล์ข้อมูล Bigdata.json

การประเมินประสิทธิภาพ	524 คำหลัก				ค่าเฉลี่ย
	รอบที่ 1	รอบที่ 2	รอบที่ 3	ค่าเฉลี่ย	1 คำหลัก
การเข้าถึงข้อมูลด้วย Dense Index ในกรณีที่ไม่พบคำหลัก	569.96	572.78	581.56	574.767	1.097
การเข้าถึงข้อมูลด้วย Non Index ในกรณีที่ไม่พบคำหลัก	20,528,300.2	20,871,312.76	1,880,8120.53	20,069,244.44	38,300.848

ในการทดลองครั้งที่สาม ได้ประเมินประสิทธิภาพการเข้าถึงข้อมูลของการใช้ดรรชนีแบบกระจาย (Sparse Index) ในการสำรวจและเข้าถึงข้อมูลจากไฟล์ Bigdata.json โดยเปรียบเทียบประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-Many) ทั้งที่มีดรรชนีแบบกระจาย (Sparse Index) และไม่มีดรรชนี (Non Index) เป็นที่น่าสังเกตว่าคำหลัก (Keyword) ที่เลือกแบบสุ่มทั้งหมดมีอยู่ในไฟล์ Bigdata.json การทดลองนี้มีวัตถุประสงค์เพื่อตรวจสอบประสิทธิภาพเวลาในการสำรวจและเข้าถึงข้อมูลของการใช้ดรรชนีแบบกระจาย (Sparse Index) เปรียบเทียบกับไม่มีดรรชนี (Non Index)โดยเน้นที่การมีอยู่ของคำหลักภายในไฟล์ Bigdata.json ดังแสดงในตารางที่ 3 ของผลการทดสอบที่เกี่ยวข้องกับการเรียกคำหลัก 524 คำ โดยใช้ดรรชนีแบบกระจาย (Sparse Index) ดำเนินการศึกษาทั้งหมดสามรอบ โดยนำเสนอเวลาเฉลี่ยในการเข้าถึงข้อมูลต่อคำหลัก (Keyword) การค้นพบนี้แสดงให้เห็นว่าการใช้ดรรชนีแบบกระจาย (Sparse Index) สามารถช่วยลดเวลาในการเข้าถึงข้อมูลเมื่อเทียบกับไม่มีดรรชนี (Non Index)ผลการทดสอบพบว่า ดรรชนีแบบกระจาย (Sparse Index) ใช้เวลาเฉลี่ยต่อคำหลักในการเข้าถึงหลายตำแหน่งข้อมูลเท่ากับ 9.795มิลลิวินาที ส่วนการเข้าถึงไฟล์ Bigdata.json โดยการแบ่งส่วนข้อมูล (Segment) ใช้เวลาเฉลี่ยต่อคำหลักเท่ากับ 756.65 มิลลิวินาที นอกจากนี้ เมื่อพิจารณาการเข้าถึงข้อมูลทั้งหมดด้วยการใช้ดรรชนีแบบกระจาย (Sparse Index) เวลาที่ใช้เฉลี่ยต่อคำหลักเท่ากับ 854.662 มิลลิวินาที หากไม่มีดรรชนี (Non Index)ใช้เวลาในการเข้าถึงข้อมูลเฉลี่ยต่อคำหลักเท่ากับ 55,197.734 มิลลิวินาที ในการทดสอบซ้ำทั้งสามรอบของ 524 คำหลัก พบว่าการเข้าถึงข้อมูลโดยใช้ดรรชนีแบบกระจาย (Sparse Index) ใช้เวลาเฉลี่ยเท่ากับ 447,842.638 มิลลิวินาที ในขณะที่การเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index)ใช้เวลาโดยเฉลี่ยเท่ากับ 28,923,612.59 มิลลิวินาที ผลการทดสอบแสดง

ให้เห็นถึงประสิทธิภาพทางด้านเวลาของ ดรรชนีแบบกระจาย (Sparse Index) ที่ดีขึ้นเป็นอย่างมาก เมื่อเทียบกับไม่มีดรรชนี (Non Index)เนื่องจากสามารถข้ามข้อมูลที่ไม่จำเป็นและเข้าถึงข้อมูลเพียงครั้งเดียว นอกจากนี้ ยังนำเข้าข้อมูลลงใน RAM หนึ่งครั้งต่อคำหลัก (Keyword) แม้ว่าคำหลักจะปรากฏในหลายตำแหน่ง (Position) ซึ่งสามารถช่วยลดเวลาในการเข้าถึงข้อมูลและประมวลผลข้อมูล ในทางตรงกันข้าม การเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index) จะนำเข้าข้อมูลคำหลัก (Keyword) ทั้งหมด ซึ่งนำไปสู่การสูญเสียทรัพยากรหน่วยความจำอย่างมากและส่งผลกระทบต่อเวลาในการเข้าถึงข้อมูลที่เพิ่มขึ้นตามขนาดข้อมูล

ตารางที่ 3 การเปรียบเทียบประสิทธิภาพของการสำรวจและเวลาในการเข้าถึงไฟล์ข้อมูล Bigdata.json โดยใช้ดรรชนีแบบกระจาย และไม่มีการจัดทำดรรชนี

ประเภทของการทดสอบ	524 คำหลัก				ค่าเฉลี่ย
	รอบที่ 1	รอบที่ 2	รอบที่ 3	ค่าเฉลี่ย	1 คำหลัก
เวลาการเข้าถึงข้อมูล Index ด้วย Sparse Index (มิลลิวินาที)	52,056.211	50,885.636	51,130.898	51,357.582	9.795
เวลาการเข้าถึง Segment ด้วย Sparse index (มิลลิวินาที)	395,083.445	395,693.442	398,678.282	396,485.565	756.65
เวลาการเข้าถึงข้อมูล Sparse Index ทั้งหมด (มิลลิวินาที)	447,139.656	446,579.078	449,809.18	447,842.638	854.661
เวลาการเข้าถึงข้อมูล กรณีไม่มีดรรชนี (มิลลิวินาที)	28,850,300.91	28,970,251.46	28,950,285.38	28,923,612.59	55,197.734

การทดลองครั้งที่สี่ ได้ประเมินประสิทธิภาพการเข้าถึงข้อมูลของการใช้ดรรชนีแบบกระจาย (Sparse Index) ในการสำรวจและเข้าถึงข้อมูลจากไฟล์ Bigdata.json โดยเปรียบเทียบประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-Many) ทั้งที่มีดรรชนีแบบกระจาย (Sparse Index) และไม่มีดรรชนี (Non Index) โดยไฟล์ดรรชนีดังกล่าวจะไม่พบคำหลัก (Keyword not found) การศึกษาให้ผลลัพธ์ของการทดลองดังแสดงในตารางที่ 4 ของสถานการณ์ที่ไม่พบคำหลัก (Keyword not found) เวลาในการสำรวจและเข้าถึงข้อมูล ซึ่งจำกัดอยู่ที่เวลาที่ใช้ในการเข้าถึงข้อมูลในไฟล์ดรรชนีแบบกระจาย (Sparse Index) และไม่เกี่ยวข้องกับการเข้าถึงข้อมูลใน RAM นี่เป็นเพราะไม่มีส่วนที่เกี่ยวข้องที่จะนำเข้าและไม่มีตำแหน่ง (Position) ในดรรชนีแบบกระจาย (Sparse Index) ที่ต้องการเข้าถึงข้อมูล ผู้วิจัยทำการทดลองโดยใช้คำหลักของดรรชนีแบบกระจาย (Sparse Index) 524 คำหลัก ผลลัพธ์แสดงให้เห็นถึงประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลโดยเฉลี่ยต่อคำหลักเท่ากับ 0.179 มิลลิวินาที โดยมีเวลาเข้าถึงข้อมูลเฉลี่ยสำหรับทั้งสามรอบเท่ากับ 93.543 มิลลิวินาที ในทางตรงกันข้าม วิธีการเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index)ในกรณีที่ไม่พบคำหลักนั้นทดสอบกับ 524 คำหลัก ผลลัพธ์พบว่าเวลาในการเข้าถึงข้อมูลโดยเฉลี่ยต่อคำหลักเท่ากับ 47,203.253 มิลลิวินาที โดยมีเวลาการเข้าถึงข้อมูลเฉลี่ยสำหรับทั้งสามรอบเท่ากับ 24,734,504.72 มิลลิวินาที ผลลัพธ์พบว่าการเข้าถึงข้อมูลโดยดรรชนีแบบกระจาย (Sparse Index) ในกรณีที่ไม่พบคำหลัก (Keyword not found) มีประสิทธิภาพมากกว่า กรณีที่ไม่มีดรรชนี (Non Index) เนื่องจากข้อมูลที่ไม่จำเป็นถูกข้ามไป และข้อมูลถูกเข้าถึงเพียงครั้งเดียวต่อคำหลัก (Keyword) ทำให้ลดเวลาในการเข้าถึงข้อมูลเฉลี่ยต่อคำหลัก (Keyword) ลดลงเหลือเพียง 0.179 มิลลิวินาที ในทางกลับกัน การสำรวจและเข้าถึงข้อมูลโดยไม่มีการจัดทำดรรชนีในกรณีที่ไม่พบคำหลัก (Keyword not found) ทำให้ใช้เวลาในการสำรวจและเข้าถึงข้อมูลนานกว่ามาก โดยเวลาเข้าถึงข้อมูลเฉลี่ยต่อคำหลัก

เท่ากับ 47,203.253 มิลลิวินาที ส่งผลกระทบสูงต่อ RAM โดยเฉพาะอย่างยิ่งเวลาในกระบวนการนำเข้าคำหลักทั้งหมด และเวลาในการเข้าถึงข้อมูล

ตารางที่ 4 การเปรียบเทียบประสิทธิภาพของการเข้าถึงข้อมูล โดยใช้ดรรชนีแบบกระจายและไม่มีดรรชนีและกรณีที่ไม่มีคำหลักในไฟล์ข้อมูล Bigdata.json

การประเมินประสิทธิภาพ	524 คำหลัก				ค่าเฉลี่ย
	รอบที่ 1	รอบที่ 2	รอบที่ 3	ค่าเฉลี่ย	1 คำหลัก
การเข้าถึงข้อมูลด้วย Sparse Index ในกรณีที่ไม่พบคำหลัก	93.12	90.2	97.31	93.543	0.179
การเข้าถึงข้อมูลด้วย Non Index ในกรณีที่ไม่พบคำหลัก	26,873,520.89	23,599,661.86	23,730,331.4	24,734,504.72	47,203.253

วัตถุประสงค์อีกประการหนึ่งของการทดลองนี้ คือการกำหนดขนาดส่วน (Segment size) ที่เหมาะสมที่สุดโดยทำการทดสอบในสามช่วงที่แตกต่างกัน ช่วงแรกครอบคลุมตั้งแต่ 100 ถึง 1,000 โดยเพิ่มขึ้นทีละ 100 สำหรับแต่ละขนาดส่วน (segment) ช่วงที่สองครอบคลุม 1,000 ถึง 10,000 โดยเพิ่มขึ้น 1,000 สำหรับแต่ละขนาดส่วน (Segment size) สุดท้าย ช่วงที่สามขยายจาก 10,000 เป็น 200,000 โดยเพิ่มขึ้น 10,000 สำหรับแต่ละขนาดส่วน (Segment size) ผู้วิจัยทำการทดลองทั้งหมดสามรอบและคำนวณค่าเฉลี่ย นอกจากนี้ ตารางที่ 5 และ 6 ยังสรุปค่าขนาดส่วน (Segment size) ที่เหมาะสมสำหรับการประเมินประสิทธิภาพของดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) ในช่วงที่กำหนดทั้งสามช่วง โดยมีรายละเอียดของผลการทดลองดังนี้

ตารางที่ 5 แสดงค่าขนาดส่วน (Segment size) ที่เหมาะสมสำหรับการประเมินประสิทธิภาพของดรรชนีแบบหนาแน่น (Dense Index) โดยแบ่งออกเป็นสามช่วง แต่ละช่วงขนาดส่วน (Segment size) กลุ่มแรกตั้งแต่ 100 ถึง 1,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 197,367.708 มิลลิวินาที ช่วงที่สองตั้งแต่ 1,000 ถึง 10,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 197722.472 มิลลิวินาที สุดท้าย ช่วงที่สามตั้งแต่ 10,000 ถึง 200,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 195542.527 มิลลิวินาที รูปที่ 9 แสดงให้เห็นว่าขนาดส่วน (Segment size) ในช่วงตั้งแต่ 100 ถึง 200,000 พบว่าประสิทธิภาพในการเข้าถึงคำหลัก (Keyword) มีความแตกต่างทางด้านของเวลาเล็กน้อย เนื่องจากการเข้าถึง RAM ไม่มีความแตกต่าง เมื่อเทียบกับขนาดส่วน (Segment size) ทั้งนี้ขึ้นอยู่กับประสิทธิภาพของระบบคอมพิวเตอร์ ดังนั้นไม่ว่าขนาดส่วน (Segment size) จะอยู่ในช่วงใด ดรรชนีแบบหนาแน่น (Dese Index) ยังคงสามารถจัดการกับหน่วยความจำและการเข้าถึงคำหลัก (Keyword) ได้อย่างรวดเร็ว

ตารางที่ 5 ประสิทธิภาพดรรชนีแบบหนาแน่น ตามขนาดส่วน และความถี่ของคำหลัก

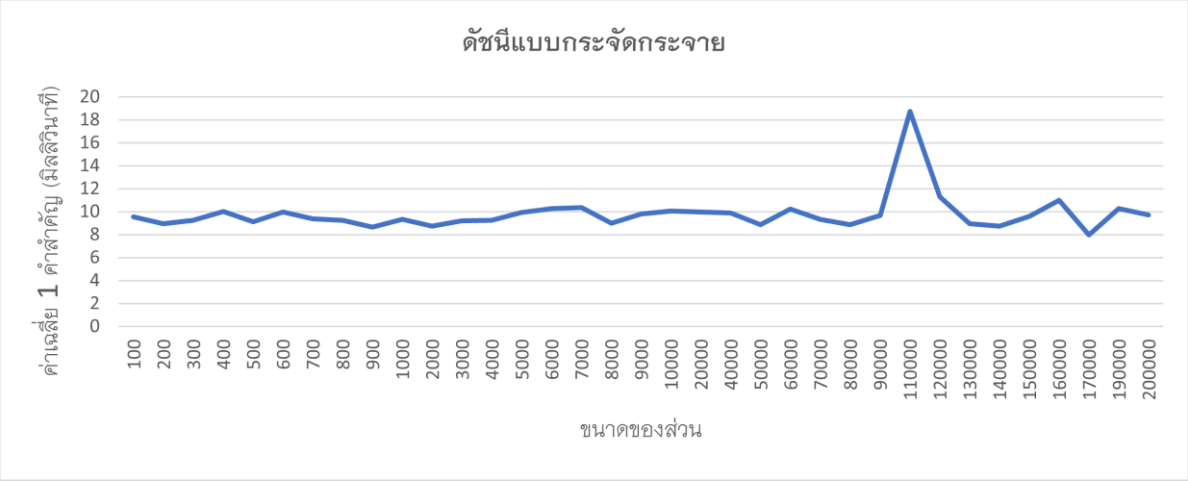
ขนาดของส่วน	ค่าเฉลี่ย 1 คำสำคัญ (มิลลิวินาที)
100 to 1000	197,367.708
1000 to 10000	197722.472
10000 to 200000	195542.527

รูปที่ 9 การเปรียบเทียบประสิทธิภาพของดรรชนีแบบหนาแน่น

ตารางที่ 6 แสดงค่าขนาดส่วน (Segment size) ที่เหมาะสมสำหรับการประเมินประสิทธิภาพของดรรชนีแบบกระจาย. (Sparse Index) โดยแบ่งออกเป็นสามช่วง แต่ละช่วงขนาดส่วน (Segment size) กลุ่มแรกตั้งแต่ 100 ถึง 1,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 52,056.21 มิลลิวินาที ช่วงที่สองตั้งแต่ 1,000 ถึง 10,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 50,885.637 มิลลิวินาที สุดท้าย ช่วงที่สามตั้งแต่ 10,000 ถึง 200,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 51,130.898 มิลลิวินาที รูปที่ 9 แสดงให้เห็นว่าขนาดส่วน (Segment size) ในช่วงตั้งแต่ 100 ถึง 200,000 พบว่าประสิทธิภาพในการเข้าถึงคำหลัก (Keyword) มีความแตกต่างทางด้านของเวลาเล็กน้อย เนื่องจากการเข้าถึง RAM ไม่มีความแตกต่างเมื่อเทียบกับขนาดส่วน (Segment size) ทั้งนี้ขึ้นอยู่กับประสิทธิภาพของระบบคอมพิวเตอร์ ดังนั้น ไม่ว่าขนาดส่วน (Segment size) จะอยู่ในช่วงใด ดรรชนีแบบกระจาย (Sparse Index) ยังคงสามารถจัดการกับหน่วยความจำและการเข้าถึงคำหลัก (Keyword) ได้อย่างรวดเร็ว

ตารางที่ 6 ประสิทธิภาพดรรชนีแบบหนาแน่น ตามขนาดส่วน และความถี่ของคำหลัก

ขนาดของส่วน	ค่าเฉลี่ย 1 คำสำคัญ (มิลลิวินาที)
100 to 1000	52,056.21
1000 to 10000	50,885.637
10000 to 200000	51,130.898



รูปที่ 10 การเปรียบเทียบประสิทธิภาพของดรรชนีแบบกระจาย

โปรดสังเกตว่า เวลาในการสำรวจและเข้าถึงข้อมูล โดยใช้การทำดรรชนีแบบหนาแน่น (Dense Index) พบว่านานกว่าเวลาที่ใช้การทำดรรชนีแบบกระจาย (Sparse Index) ซึ่งเป็นผลมาจากเวลาค้นหาคำหลัก (Keyword) ที่นานขึ้นที่จำเป็นในการดึงตำแหน่งของข้อมูลในไฟล์ดรรชนีหนาแน่น (Dense Index) เมื่อเทียบกับไฟล์ดรรชนีกระจาย (Sparse Index) ความแตกต่างนี้ยังเห็นได้ชัด เมื่อพิจารณาจากขนาดไฟล์และจำนวนข้อมูล โดยไฟล์ดรรชนีแบบหนาแน่น (Dense Index) จะมีขนาด 20 เมกะไบต์และมีข้อมูล 1,000,000 รายการ ในขณะที่ไฟล์ดรรชนีแบบกระจาย (Sparse Index) จะมีขนาดเพียง 8 เมกะไบต์และมีข้อมูล 5146 รายการ

สรุปและอภิปรายผล

การศึกษานี้นำเสนอการเพิ่มประสิทธิภาพในการสำรวจและเข้าถึงข้อมูล โดยใช้เทคนิคการสร้างดรรชนีสำหรับชุดข้อมูลขนาดใหญ่ ภายใต้รูปแบบวัตถุของจาวาสคริปต์ โดยการนำประยุกต์เทคนิคการจัดทำดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) เพื่อเพิ่มประสิทธิภาพการสำรวจและเข้าถึงข้อมูล การค้นพบนี้แสดงให้เห็นว่าวิธีการสร้างดรรชนีใหม่ช่วยลดเวลาในการสำรวจและเข้าถึงข้อมูลลดลงได้อย่างมาก นอกจากนี้ การทดลองในทั้งสามช่วงขนาดส่วน (Segment size) ตั้งแต่ช่วง 100 ถึง 1,000, 1,000 ถึง 10,000 และ 10,000 ถึง 200,000 ผลลัพธ์พิสูจน์ให้เห็นว่าการเปลี่ยนแปลงขนาดส่วนไม่ได้ส่งผลต่อประสิทธิภาพทางด้านเวลาในการเข้าถึงคำหลัก (Keyword) ซึ่งเป็นผลมาจากการเข้าถึงหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ที่ใกล้เคียงกันเมื่อเทียบกับขนาดส่วน (Segment size) ทั้งนี้ขึ้นอยู่กับประสิทธิภาพของระบบคอมพิวเตอร์ ดังนั้น ไม่ว่าขนาดส่วน (Segment size) จะอยู่ในช่วงใด ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) ยังคงสามารถจัดการกับหน่วยความจำและการเข้าถึงคำหลัก (Keyword) ได้อย่างรวดเร็ว นอกจากนี้ ดรรชนีแบบหนาแน่น (Dense Index) สามารถเพิ่มประสิทธิภาพทางด้านเวลาในการเข้าถึงกับลักษณะข้อมูลแบบหนึ่งต่อหนึ่ง (One-to-One) เฉลี่ยต่อคำหลักเท่ากับ 382.196 มิลลิวินาที เมื่อเทียบกับการเข้าถึงข้อมูลของกรณีไม่มีดรรชนี (Non Index) เฉลี่ยต่อคำหลักเท่ากับ 26,869.218 มิลลิวินาที ในขณะที่ดรรชนีแบบกระจาย (Sparse Index) สามารถเพิ่มประสิทธิภาพทางด้านเวลาในการเข้าถึงกับลักษณะข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-Many) เฉลี่ยต่อคำหลักเท่ากับ 854.662 มิลลิวินาที เมื่อเทียบกับการเข้าถึงข้อมูลของกรณีไม่มีดรรชนี (Non Index) เฉลี่ยต่อคำหลักเท่ากับ 55,197.734 มิลลิวินาที การค้นพบนี้ชี้ให้เห็นว่าแนวทางการจัดทำดรรชนีแบบหนาแน่น (Dense Index) สามารถ

ลดเวลาในการเข้าถึงข้อมูลถึง 98.57% จากวิธีการไม่มีดรรชนีและดรรชนีแบบกระจาย (Sparse Index) สามารถลดเวลาในการเข้าถึงข้อมูลถึง 98.45% จากวิธีการไม่มีดรรชนีทั้งสองวิธีการดรรชนีของผู้วิจัยสามารถช่วยเพิ่มการสำรวจและเข้าถึงในชุดฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ที่จัดเก็บอยู่ภายใต้รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) โดยเฉพาะอย่างยิ่งเมื่อจำลองชุดข้อมูลขนาดใหญ่ การศึกษาในอนาคตสามารถขยายการวิจัยนี้ได้โดยการทดสอบประสิทธิภาพของเทคนิคการจัดทำดรรชนีกับข้อมูลประเภทต่าง ๆ และชุดข้อมูลขนาดต่าง ๆ เพื่อกำหนดความสามารถในการปรับขนาดและความสามารถทั่วไปในการทำงานในอนาคต การปรับปรุงอัลกอริทึมการสำรวจและเข้าถึงตำแหน่งข้อมูลในไฟล์ดรรชนีแบบหนาแน่น (Dense Index) อาจเป็นส่วนที่มีศักยภาพในการตรวจสอบ เนื่องจากไฟล์ดรรชนีแบบหนาแน่น (Dense Index) มักจะมีข้อมูลจำนวนมาก กระบวนการค้นหาจึงใช้เวลานานและไม่มีประสิทธิภาพ การลดเวลาที่จำเป็นสำหรับกระบวนการนี้สามารถปรับปรุงประสิทธิภาพโดยรวมของแนวทางการจัดทำดรรชนีแบบหนาแน่น (Dense Index) ได้อย่างมาก ดังนั้นวิธีการเพิ่มประสิทธิภาพอัลกอริทึมของการสำรวจและเข้าถึงสำหรับการจัดทำดรรชนีแบบหนาแน่น (Dense Index) อาจเป็นช่องทางที่น่าสนใจสำหรับงานในอนาคต

กิตติกรรมประกาศ

ผู้เขียนขอขอบคุณหน่วยวิจัยเพื่อการพัฒนานวัตกรรมเชิงพื้นที่ (RUSID) คณะเทคโนโลยีสารสนเทศและการสื่อสาร มหาวิทยาลัยพะเยา ที่ให้การสนับสนุนสิ่งอำนวยความสะดวก

เอกสารอ้างอิง

Abdulkadhem, A. A., & Al-Assadi, T. A. (2019). An Important Landmarks Construction for a GIS-Map based on Indexing of Dolly Images. *Indonesian Journal of Electrical Engineering and Computer Science*, *15*(1), 451. https://doi.org/10.11591/ijeecs.v15.i1.pp451-459.

Abdulsada, A. I., Honi, D. G., & Al-Darraji, S. (2021). Efficient multi-keyword similarity search over encrypted cloud documents. *Indonesian Journal of Electrical Engineering and Computer Science*, *23*(1), 510. https://doi.org/10.11591/ijeecs.v23.i1.pp510-518.

Alqatawneh, A. (2022). Orthogonal frequency division multiplexing system with an indexed-pilot channel estimation. *Indonesian Journal of Electrical Engineering and Computer Science*, *26*(2), 808. https://doi.org/10.11591/ijeecs.v26.i2.pp808-818.

Chang, J., Xiao, L., Huo, Z., Zhou, B., Ruan, L., Wang, H., & Liu, S. (2017). Optimization of Index-Based Method of Metadata Search for Large-Scale File Systems. *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*. https://doi.org/10.1109/iscid.2017.147.

Chopade, R., & Pachghare, V. (2020). MongoDB Indexing for Performance Improvement. *Advances in Intelligent Systems and Computing*, *1077*, 529–539. https://doi.org/10.1007/978-981-15-0936-0_56.

Fathy, Y., Barnaghi, P., & Tafazolli, R. (2018). Large-Scale Indexing, Discovery, and Ranking for the Internet of Things (IoT). *ACM Computing Surveys*, *51*(2), 1–53. https://doi.org/10.1145/3154525.

Gayathiri, N. R., Jaspher, D. D., & Natarajan, A. M. (2019). Big Data retrieval techniques based on Hash Indexing and MapReduce approach with NoSQL Database. *2019 International Conference on Advances in Computing and Communication Engineering (ICACCE)*. https://doi.org/10.1109/icacce46606.2019.9079964.

Jin, P., Zhuang, X., Luo, Y., & Lu, M. (2021, December 1). Exploring Index Structures for Zoned Namespaces SSDs. https://doi.org/10.1109/BigData52589.2021.9671606.

L. Tan, K., & C. Lim, K. (2019). Fast surveillance video indexing & retrieval with WiFi MAC address tagging. *Indonesian Journal of Electrical Engineering and Computer Science*, *16*(1), 473. https://doi.org/10.11591/ijeecs.v16.i1.pp473-481.

Ma, Y., Liu, D., Scott, G., Uhlmann, J., & Shyu, C.-R. (2017, December 1). In-Memory Distributed Indexing for Large-Scale Media Data Retrieval. https://doi.org/10.1109/ISM.2017.38.

S, M., & MB, S. P. (2020). Indexing intelligence using benchmark classifier. *Indonesian Journal of Electrical Engineering and Computer Science*, *18*(1), 179. https://doi.org/10.11591/ijeecs.v18.i1.pp179-187.

Yuan, J., & Liu, X. (2012). A novel index structure for large scale image descriptor search. *2012 19th IEEE International Conference on Image Processing*. https://doi.org/10.1109/icip.2012.6467265.

Yusof, M. K. (2017). Efficiency of JSON for Data Retrieval in Big Data. *Indonesian Journal of Electrical Engineering and Computer Science*, *7*(1), 250. https://doi.org/10.11591/ijeecs.v7.i1.pp250-262.

Zeffora, J., & Shobarani, S. (2022). Optimizing random forest classifier with Jenesis-index on an imbalanced dataset. *Indonesian Journal of Electrical Engineering and Computer Science*, *26*(1), 505. https://doi.org/10.11591/ijeecs.v26.i1.pp505-511.

ZineddineK., AmineF. M., & Adeel, A. (2018). Indexing Multimedia Data with an Extension of Binary Tree -- Image Search by Content --. *International Journal of Informatics and Applied Mathematics*, *1*(1), 47–55. Retrieved from https://dergipark.org.tr/en/pub/ijiam/issue/43831/532310.