ระบบข้อมูลขนาดใหญ่สำหรับวิเคราะห์ข้อมูลการขนส่ง

BIG DATA SYSTEM FOR TRANSPORTATION DATA ANALYTICS

ชมพูนุท คุ้มพิทักษ์

ณัฐนันท์ อริยะเมตตา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2562

ระบบข้อมูลขนาดใหญ่สำหรับวิเคราะห์ข้อมูลการขนส่ง

นางสาวชมพูนุท คุ้มพิทักษ์ 59010285

นางสาวณัฐนันท์ อริยะเมตตา 59010443

ผศ.ดร.รัฐชัย ชาวอุทัย อาจารย์ที่ปรึกษา

ปีการศึกษา 2562

บทคัดย่อ

โครงงานฉบับนี้นำเสนอการออกแบบและพัฒนาระบบข้อมูลขนาดใหญ่สำหรับวิเคราะห์ข้อมูลการขนส่ง โดยเป็นโครงงานที่ใช้ความรู้เรื่อง Big Data เข้ามาออกแบบระบบสำหรับรองรับการนำเข้าข้อมูล GPS ของรถกว่า 400,000 คัน / นาที เพื่อนำมาจัดเก็บและวิเคราะห์ข้อมูล จากระบบเดิมที่ใช้วิธีการจัดเก็บข้อมูลโดยใช้ Relational Database และการวิเคราะห์ข้อมูลโดยใช้ Python ซึ่งใช้เวลาในการประมวลผลนาน และอาจมีแนวโน้มที่จะไม่สามารถรองรับข้อมูลที่มีการเติบโตตลอดเวลาได้ในอนาคต ทางผู้จัดทำจึงได้พัฒนาระบบนี้ขึ้น เพื่อให้สามารถจัดเก็บข้อมูลได้อย่างมีประสิทธิภาพ และลดเวลาในการวิเคราะห์ข้อมูล อีกทั้งยังมีการจัดสรรทรัพยากร เพื่อให้ระบบมีความคงทนต่อความต่อความเสียหาย

BIG DATA SYSTEM FOR TRANSPORTATION DATA ANALYTICS

Miss Chompoonud Khumphithak 59010285

Miss Natthanan Ariyamatta 59010443

MR. Rathachai Chawuthai Advisor

Academic Year 2562

ABSTRACT

This project presents the design and development of big data systems for transportation data analysis. It is a project that uses knowledge of Big Data to design a system to support the import of GPS data of more than 400,000 records / minute for data storage and analysis. From the old system using data storage using Relational Database and data analysis using Python which takes a long time to process. And may not be able to support the data that is growing all the time in the future, we have developed this system. To be able to store data efficiently and reduce the time for data analysis. There is also a resource allocation, so that the system is durable to damage.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความกรุณาจากบุคคลผู้พระคุณหลายท่านที่ให้ความช่วยเหลือแนะนำ ให้ทรัพยากร และความเห็นทางวิชาการอันเป็นประโยชน์อย่างยิ่ง และความร่วมมือของบคุคลหลายฝ่ายที่ต้องกราบขอบพระคุณมา ณ ที่นี้

ผู้วิจัยขอขอบคุณ ผศ. ดร.รัฐชัย ชาวอุทัย ผู้เป็นที่ปรึกษาปริญญานิพนธ์ฉบับนี้ ที่ได้สละเวลาอันมีค่า รับเป็นอาจารย์ที่ปรึกษาและกรุณาให้คำชี้แนะ อีกทั้งยังช่วยตรวจพิจารณาแก้ไข สุดท้ายนี้ผู้วิจัยขอขอบพระคุณครอบครัวที่คอยให้กำลังใจเสมอมา รวมถึงผู้ให้การสนับสนุนในทุก ๆ ด้าน ทำให้ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดี

ผู้วิจัยหวังว่าปริญญานิพนธ์ฉบับนี้จะมีประโยชน์ต่อผู้ที่สนใจไม่มากก็น้อย หากมีข้อผิดพลาดประการใด ขออภัยมา ณ ที่นี้ด้วย

ชมพูนุท คุ้มพิทักษ์

ณัฐนันท์ อริยะเมตตา

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

ตาราง **หน้า**

สารบัญรูป

รูป **หน้า**

สารบัญรูป(ต่อ)

รูป **หน้า**

สารบัญรูป(ต่อ)

รูป **หน้า**

สารบัญรูป(ต่อ)

รูป **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

เนื่องจากในปัจจุบันเทคโนโลยีต่าง ๆ มีการเติบโตอย่างรวดเร็ว รวมถึงเทคโนโลยีด้านการคมนาคมด้วย ระบบขนส่งต่าง ๆ ได้มีการนำอุปกรณ์ติดตามตำแหน่งมาติดตั้งเพื่อเพิ่มความปลอดภัยในการขับขี่บนท้องถนนเพราะสามารถติดตามทั้งเวลา, ตำแหน่ง, ความเร็วและประเภทของรถได้ โดยข้อมูลเหล่านี้จะถูกจัดเก็บไว้ใน Relational Database ซึ่งมีแนวโน้มว่าจะไม่สามารถรองรับข้อมูลที่เพิ่มขึ้นตลอดเวลานี้ได้และยังเสี่ยงต่อการเกิดข้อมูลสูญหาย รวมทั้งยังใช้ Python ในการวิเคราะห์ผล ซึ่งใช้เวลาค่อนข้างมาก

ผู้จัดทำจึงพัฒนาระบบข้อมูลขนาดใหญ่สำหรับวิเคราะห์ข้อมูลการขนส่ง (Big Data System for Transportation Data Analytics) ขึ้นมา เพราะ Big data นั้นเป็นระบบที่มีโครงสร้างการทำงานแบบ Distributed ทำให้สามารถทำงานได้รวดเร็วขึ้น ,คงทนต่อความเสียหาย และสามารถจัดการกับทรัพยากรต่าง ๆในระบบได้ดียิ่งขึ้น

1.2 วัตถุประสงค์

1) เพื่อออกแบบระบบ Big data สำหรับรองรับข้อมูลขนาดใหญ่ของระบบ Smart City
2) เพื่อพัฒนาต้นแบบระบบ Big data สำหรับรองรับข้อมูลขนาดใหญ่ของระบบ Smart City
3) เพื่อวิเคราะห์ข้อมูลบนระบบ Big dataสำหรับรองรับข้อมูลขนาดใหญ่ของระบบ Smart City
4) เพื่อให้สามารถแสดงผลการวิเคราะห์ข้อมูลบนระบบ Big dataสำหรับรองรับข้อมูลขนาดใหญ่ของระบบ Smart City ได้

1.3 ขอบเขต

1) มีการนำเข้าข้อมูล GPS 400,000 records/นาที ในระบบ Big data
2) ออกแบบระบบให้เก็บข้อมูลได้ 2 ปี
3) ระบบสามารถเก็บข้อมูลไว้ได้ระบบ Big data ได้เป็นระยะเวลาจำนวน 2 ปี
4) สามารถออกรายงานการวิเคราะห์ที่จัดกลุ่มด้วย ช่วงเวลา/ชนิดรถ/พื้นที่
5) สามารถแสดงผลการวิเคราะห์ในรูปแบบของ Interactive user graphic

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1) ลดพื้นที่ในการจัดเก็บและลดเวลาในการวิเคราะห์ข้อมูล GPS ในระบบ Smart City
2) เข้าใจการทำงานของระบบ Big Data และการใช้เครื่องมือต่าง ๆ ที่เกี่ยวข้องกับการทำ Big Data

1.5 ขั้นตอนการดำเนินงาน

1) ศึกษาความต้องการและขอบเขตของโครงงาน
2) วิเคราะห์ความต้องการและวางแผนการพัฒนาโครงงานให้สอดคล้องกับงานเพื่อให้โครงงานสำเร็จภายในระยะเวลาที่กำหนด
3) ศึกษาเกี่ยวกับระบบ Big Data และเทคโนโลยีที่เกี่ยวข้อง
4) ออกแบบโครงสร้างของระบบ Big Data ให้สอดคล้องกับความต้องการของโครงงาน
5) ทำการทดลองตามโครงสร้างของระบบที่ออกแบบไว้
6) ทดสอบและปรับปรุงโครงสร้างของระบบให้มีประสิทธิภาพมากยิ่งขึ้น
7) สรุปผลและจัดทำเอกสารอธิบายการทำงานและโครงสร้างของโครงงาน

หัวข้อกิจกรรม	เดือน							
	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.
ศึกษาความต้องการและขอบเขตของโครงงาน								
วิเคราะห์ความต้องการและวางแผนการพัฒนาโครงงาน								
ศึกษาเกี่ยวกับระบบ Big Data และเทคโนโลยีที่เกี่ยวข้อง								
ออกแบบภาพรวมโครงสร้างของระบบ Big Data								
ติดตั้งเครื่องมือต่าง ๆที่ต้องใช้งาน								
ออกแบบการนำเข้าข้อมูล								
ทดลองนำเข้าข้อมูลตามตามโครงสร้างที่ออกแบบ								
ปรับปรุงโครงสร้างและทดลองเพื่อเพิ่มประสิทธิภาพการนำเข้าข้อมูล								
ออกแบบโครางสร้างการจัดเก็บข้อมูล								
ออกแบบ Pipeline การนำเข้าและจัดเก็บข้อมูลแบบ Realtime								
ทดลองนำเข้าข้อมูลโดยใช้Pipeline ที่ออกแบบไว้								
วิเคราะห์ข้อมูลตามขอบเขตของโครงงาน								
แสดงผลข้อมูลที่ได้จากการวิเคราะห์								
ทดสอบและแก้ไขระบบให้สมบูรณ์								
จัดทำเอกสารและสรุปผลโครงงาน								

ตารางที่ 1.1 แสดงแผนการดำเนินงาน

บทที่ 2
เอกสารที่เกี่ยวข้อง

2.1 ข้อมูลขนาดใหญ่ (Big Data)

ข้อมูล (Data) คือ ข้อเท็จจริงต่าง ๆ ที่มีอยู่ในธรรมชาติ เป็นกลุ่มสัญลักษณ์ แทนปริมาณหรือการกระทำต่าง ๆ ที่ยังไม่ผ่านการประมวลผล ข้อมูลอาจอยู่ในรูปของตัวเลข ตัวหนังสือ รูปภาพ เป็นต้น

ข้อมูลขนาดใหญ่ (Big Data) คือ ข้อมูลที่มีขนาดใหญ่มาก มีทั้งที่เป็นข้อมูลที่มีโครงสร้างชัดเจน (Structured Data) เช่น ข้อมูลที่เก็บอยู่ในตารางข้อมูลต่าง ๆ ข้อมูลกึ่งมีโครงสร้าง (Semi-Structured Data) เช่น ล็อกไฟล์ (Log files) และไม่มีโครงสร้าง (Unstructured Data) เช่น Facebook, Twitter หรือไฟล์จำพวกมีเดีย เป็นต้น โดยข้อมูลจะมีความซับซ้อนและต้องการซอฟต์แวร์ที่รองรับการจัดการหรือการวิเคราะห์ได้อย่างมีประสิทธิภาพ เพื่อการประมวลผลและนำไปใช้ประโยชน์ได้

โดยการจะจัดว่าเป็นข้อมูลขนาดใหญ่ได้นั้น ข้อมูลจะต้องมี 4 องค์ประกอบ ดังนี้

1) Volume หมายถึง ข้อมูลต้องมีขนาดใหญ่ มีปริมาณข้อมูลเป็นจำนวนมาก เป็นได้ทั้งข้อมูลแบบออฟไลน์และออนไลน์
2) Variety หมายถึง ข้อมูลต้องมีมีความหลากหลาย เป็นได้ข้อมูลที่มีโครงสร้างและข้อมูลที่ไม่สามารถจับแพทเทิร์นได้
3) Velocity หมายถึง ข้อมูลต้องมีการเปลี่ยนแปลงตลอดเวลาอย่างรวดเร็วและมีการส่งผ่านข้อมูลอย่างต่อเนื่อง
4) Veracity หมายถึง ข้อมูลที่ยังไม่สมบูรณ์จึงยังไม่สามารถนำไปประกอบการตัดสินใจได้

ข้อมูลขนาดใหญ่แบ่งออกเป็น 4 ส่วน ได้แก่

1) Data Collection/ Ingestion คือการนำเข้าข้อมูลจากแหล่งข้อมูลต่าง ๆ
2) Data Storage คือการจัดเก็บข้อมูล
3) Analysis/ Processing คือการประมวลผลข้อมูล
4) Data Visualization คือส่วนของการแสดงผล

การทำงานของระบบข้อมูลขนาดใหญ่

ระบบข้อมูลขนาดใหญ่ จะใช้ MapReduce เป็นกระบวนการที่ใช้สำหรับการแบ่ง Input data ให้มีขนาดเล็กลง แล้วส่งไปประมวลผล ยัง Node อื่น ๆ ที่อยู่ใน Cluster เมื่อประมวลเสร็จแล้วจึงนำผลลัพธ์ที่ได้กลับมาลดขนาดเป็น Output data แล้ว ค่อยส่ง Output data นั้นกลับออกไป ในขั้นตอนการ Map เครื่องที่ทำหน้าที่เป็น Master node จะนำ Input ที่ได้รับมาแบ่งเป็น Sub-problem หลายๆ ชิ้น และกระจายไปยังเครื่องที่ทำหน้าที่เป็น Worker node ซึ่งเครื่อง Worker node อาจนำข้อมูลที่ได้รับ ไปแบ่งเป็น Sub-problem อีกที (ในลักษณะของ Multilevel tree) Worker node จะทำการประมวลผล Subproblem ที่ได้รับ และส่งผลลัพธ์กลับไปยัง Master node ขั้นตอนการ Reduce จะเกิดขึ้นที่ Master Node โดยที่ Master Node จะนำผลลัพธ์ทั้งหมดที่ได้รับ จาก Worker Node และนำมาสรุปเป็นผลลัพธ์สุดท้ายก่อนส่งไปที่ Client

2.2 Apache Hadoop

Apache Hadoop คือ ซอฟท์แวร์ประเภทโอเพนซอร์ส (Open source software) ที่จัดทำขึ้นเพื่อเป็นแพลตฟอร์มในการจัดเก็บข้อมูล ซึ่งมีกรอบการทำงานเพื่อใช้ในการจัดเก็บข้อมูลและประมวลผลข้อมูลที่มีขนาดใหญ่ (Big Data) ซึ่ง Apache Hadoop สามารถปรับขยาย ยืดหยุ่น เพื่อรองรับข้อมูลที่มีจำนวนมากได้ เพราะมีการประมวลผลข้อมูลแบบกระจายผ่านเครื่องคอมพิวเตอร์ที่ถูกจัดอยู่ในรูปแบบ Cluster โดยแบ่งออกเป็น 2 ส่วนคือ Master server และ Slave server

รูป 2.1 Apache Hadoop

2.3 Cloudera

Cloudera เป็นแพลตฟอร์มที่ใช้ในการจัดการกับ Apache Hadoop โดยจะช่วยในการจัดการและวิเคราะห์ข้อมูล รวมถึงการรักษาความปลอดภัยให้ข้อมูลด้วย ทำให้ง่ายต่อการจัดการกับข้อมูลที่มีความหลากหลายและมีการเพิ่มขนาดอย่างรวดเร็ว

รูป 2.2 Cloudera

2.4 บริการของ Hadoopที่ใช้ในโครงงาน

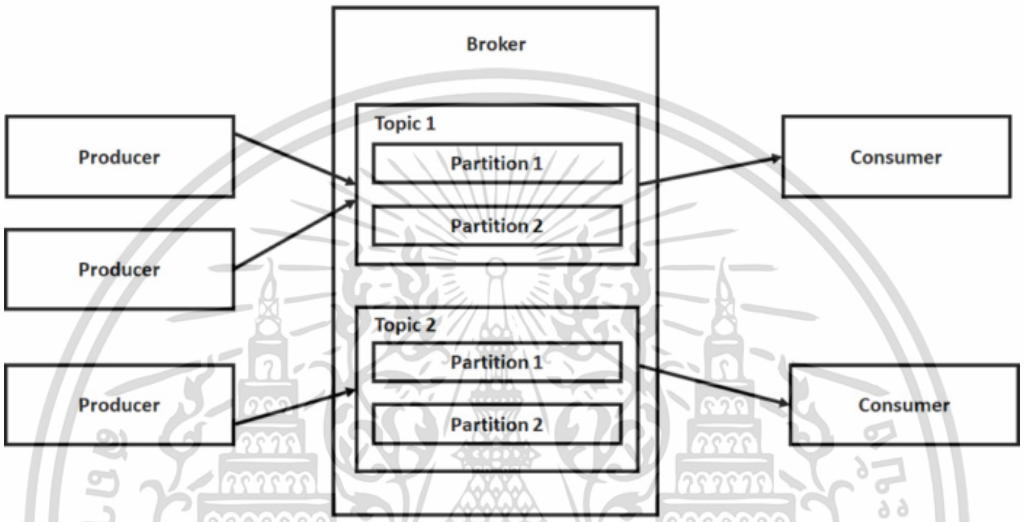
2.4.1 HDFS (Hadoop Distribution File System)

HDFS (Hadoop Distribution File System) เป็นระบบแฟ้มข้อมูลแบบกระจาย ทำหน้าที่เป็นส่วนเก็บข้อมูลที่สามารถปรับขนาดและมีความเชื่อถือ โดย HDFS (Hadoop Distribution File System) จะประกอบด้วยโหนด (Node) หมายถึงเครื่องคอมพิวเตอร์ โดยโหนดต่าง ๆ จะแบ่งออกเป็น 2 แบบได้แก่ Data Node เป็นโหนดที่ทำหน้าที่เก็บ ข้อมูลของไฟล์เอาไว้และรับผิดชอบในการประมวลผล แต่ตัว Data Node จะไม่รู้ว่าข้อมูลที่เก็บอยู่นั้น เป็นของไฟล์ไหน และ Name Node เป็นโหนดที่ทำหน้าที่รวบรวมผลของการประมวลผลต่าง ๆ จาก Data Node

รูป 2.3 HDFS (Hadoop Distribution File System)

2.4.2 Apache Kafka

Apache Kafka เป็นบริการที่ใช้เพื่อเป็นระบบส่งต่อข้อความ (messaging System), เครื่องมือบันทึกกิจกรรม (activity Tracking), รวบรวมเก็บ Log (log aggregation) หรือใช้ในการประมวลผลแบบต่อเนื่องของข้อมูล (stream processing) ซึ่ง Apache Kafka เป็นระบบที่มีความยืดหยุ่น รวดเร็ว และยังสามารถทำงานต่อได้เมื่อเกิดความเสียหายที่ส่วนใดส่วนหนึ่ง



รูป 2.4 แสดงการทำงานของ Apache Kafka

จากรูป 2.4 แสดงให้เห็นว่า Apache Kafka ทำงานโดยมี

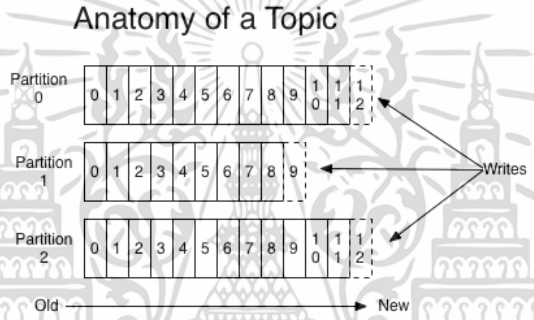
1) ผู้ผลิต (Producer) มีหน้าที่คอยดึงข้อมูลจากแหล่งข้อมูลต่าง ๆ เข้ามาที่ Kafka cluster
2) ตัวแทน (Broker) คือ ตัวเก็บข้อมูลอยู่ที่อยู่ภายใน Kafka cluster โดยในแต่ละ Broker นั้นจะ การจัดกลุ่มข้อมูลเป็นชุดของข้อมูล (Topic) และในแต่ละชุดข้อมูลนั้นก็จะถูกแบ่งออกเป็นส่วนๆ (Partitions)
3) ผู้บริโภค (Consumer) มีหน้าที่ต่อเข้ากับ Broker แล้วรับ Topic เพื่ออ่านข้อมูลในแต่ละ Partition นอกจากนี้ยังทำหน้าที่กำหนดรหัสที่ระบุข้อความในแต่ละส่วนโดยเฉพาะ เนื่องจากข้อมูลทั้งหมดนั้นเก็บบนดิสก์ผู้บริโภค จึงลดปัญหาการการประมวลผลที่มีความเร็วไม่เท่ากันโดยการกระโดดข้ามไปในจุดใดจุดหนึ่งในแต่ละส่วนได้จากรหัสเฉพาะที่กำหนด

เนื่องจากข้อมูลทั้งหมดนั้นเก็บบนดิสก์ผู้บริโภคจึงลดปัญหาการการประมวลผลที่มีความเร็วไม่เท่ากันโดยการกระโดดข้ามไปในจุดใดจุดหนึ่งในแต่ละส่วนได้จากรหัสเฉพาะที่กำหนด

การจัดกลุ่มข้อมูลเป็นชุดของข้อมูล (Topics)

Topics เป็นชุดของข้อมูล (หรือ Messages) มีลักษณะเหมือนกับ Table ใน database ที่เราคุ้นเคยกัน ซึ่งแต่ละ topic จะถูกตั้งชื่อได้ เพื่อให้เรารู้ว่ามันคือข้อมูลเกี่ยวกับอะไร ซึ่งชื่อไม่ควรซ้ำกันเพราะจะทำให้แยกชุดข้อมูลไม่ได้

ข้อมูลในแต่ละ Topic จะถูกแยกเป็นกลุ่มๆ อีก ซึ่งแต่ละกลุ่มนี้ จะถูกเรียกว่า Partition ข้อมูลในแต่ละ partition จะถูกจัดเรียง ข้อมูลแต่ละส่วน หรือแต่ละ message จะมีค่าบางอย่างที่มีลักษณะเพิ่มเองได้ (incremental) ซึ่งจะเริ่มที่ 0 และเพิ่มค่าเป็น 1 2 3… ไปเรื่อย ๆ ตามจำนวนของข้อมูล ซึ่งมันจะถูกเรียกว่า offset หรือตัวนับในแต่ละ partition



รูป 2.5 การทำงานภายใน Kafka Topic

คุณสมบัติเบื้องต้นของ Topics และ Partitions

1) offset จะใช้อ้างในแต่ละ partition เท่านั้น
2) ข้อมูลจะถูกเรียงตามลำดับก่อนหลัง ใน partition นั้น ๆ แปลว่า offset ที่ 0 ของ partition ที่ 0 อาจจะมาก่อนหรือหลัง offset ที่ 0 ของ partition 1 ก็ได้
3) ไม่สามารถแก้ไขข้อมูลที่เอาใส่ใน topic แล้ว (Immutability)
4) ข้อมูลจะถูกลบในเวลาที่ตั้งไว้ (default 604800000 ms หรือ 7 วัน)
5) ตัวนับว่าตัวถัดไปควรใช้เลขอะไร ถูกบันทึกใน ใน topic ที่ชื่อ consumer_offsets (ใน version ต่ำกว่า 0.9 ตัวนับจะเก็บใน zookeeper) ข้อมูลมากไม่ใช่ปัญหาในการอ่านข้อมูล

Topic Replication Factor

เป็นค่าที่บอกว่าในแต่ละ topic จะมีการทำสำเนา (replica) partition จากตัวหลัก(leader)ไปสำเนากี่ server หรือ broker (ซึ่งควรมีมากกว่า 1 ปกตินิยม 2–3 และ partition ตัวหลักก็มีได้ 1 ตัว ซึ่งคุณสมบัติ

นี้เอง เป็นตัวที่ทำให้ Apacha Kafka มีคุณสมบัติ Fault Tolerance (ความทนต่อความเสียหาย) ถ้ามี broker ตัวนึงสูญเสียไป ยังจะสามารถไปอ่านและบันทึกข้อมูลต่อในตัวที่เป็น replica ได้ โดยการเปลี่ยนตัวสำเนาหรือ replica ให้เป็นตัวหลัก

รูป 2.6 Apache Kafka

2.4.3 Apache Sqoop

Apache Sqoop เป็นเครื่องมือที่ออกแบบมาเพื่อการถ่ายโอนข้อมูลจำนวนมากอย่างมีประสิทธิภาพระหว่าง Apache Hadoop และที่เก็บข้อมูลแบบมีโครงสร้าง เช่นฐานข้อมูลเชิงสัมพันธ์ (Relational databases)

ข้อดีของ Apache Sqoop

1) สามารถโหลดข้อมูลตารางทั้งหมดจากฐานข้อมูลได้ด้วยคำสั่งเพียงคำสั่งเดียว
2) สามารถโหลดข้อมูลเพิ่มแค่เฉพาะส่วนที่ถูกแก้ไขได้
3) สามารถโหลดข้อมูลจากฐานข้อมูลเข้าสู่ Apache Hive ได้โดยตรง

รูป 2.7 Apache Sqoop

2.4.4 Zookeeper

Zookeeper เป็นตัวช่วยในการบริหารจัดการการรับส่งข้อมูลในบริการต่าง ๆ ที่กระจายอยู่บน Hadoop Cluster

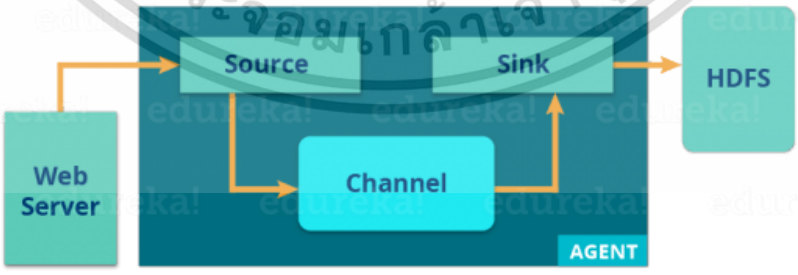
Zookeeper สำหรับ Zookeeper ที่ทำงานร่วมกับ Kafka มีบทบาทดังนี้

1) ทำหน้าที่จัดการ Brokers ดูว่า Broker ตัวไหน อยู่ที่ไหน ยังใช้งานได้อยู่หรือไม่
2) บันทึกว่า Topic ไหนมีหรือไม่มี มีกี่ Partition ใน Topic นี้
3) ทำการเลือก Leader/Replica ของ Partition
4) ส่งสัญญาณไปหา Kafka ในทุก ๆ การเปลี่ยนแปลงที่เกิดขึ้น เช่น มี Topic มาใหม่ หรือมี Broker ตัวไหนเสียหายหรือเพิ่มขึ้นมา
5) บันทึกว่า Producer/Consumer แต่ละตัวควรจะเขียนหรืออ่าน Data ได้เท่าไหร่
6) เก็บ Authorization ว่า User ไหนถูกอนุญาตให้สร้าง Topic บ้าง
7) บันทึกว่าแต่ละ consumer group มี consumer กี่ตัว อ่านไปถึง offset ไหนแล้ว

รูป 2.8 Zookeeper

2.4.5 Apache Flume

Apache Flume เป็นบริการที่เชื่อถือได้และมีความคงทน ใช้สำหรับการรวบรวมและเคลื่อนย้ายข้อมูลจำนวนมากได้อย่างมีประสิทธิภาพ โดยทำการการเคลื่อนย้ายข้อมูล จากแหล่งข้อมูลเพื่อให้ข้อมูลทำการไหลข้อมูลผ่านทาง Channel ซึ่งทำงานบน Memory ไปเก็บไว้บน HDFS ซึ่ง Apache flume มีโครงสร้างการทำงานที่เรียบง่าย, ยืดหยุ่นและยังมีคงทนต่อความเสียหาย



รูป 2.9 แสดงโครงสร้างการที่งานของ Apache flume

หลักการทำงานของ Apache flume นั้น จะมี Flume Agents คอยทำหน้าที่นำเข้าข้อมูลสตรีมจากแหล่งข้อมูลต่าง ๆ ไปที่ HDFS

โดย Flume Agents จะมีส่วนประกอบ 3 ส่วน ได้แก่

1) Source ทำหน้าที่คอบรับข้อมูลที่สตรีมเข้ามาเพื่อนำไปจัดเก็บใน Channel
2) Channel ทำหน้าที่เป็นที่เก็บข้อมูลชั่วคราวระหว่างแหล่งข้อมูลกับ HDFS เพื่อป้องกันการสูญหายของข้อมูล
3) Sink ทำหน้าที่รวบรวมข้อมูลจาก Channel เพื่อส่งต่อไปยัง HDFS

รูป 2.10 Apache Flume

2.4.6 Apache Hive

Apache Hive เป็นบริการ Data Warehouse ซึ่งสร้างอยู่บน Hadoop ใช้สำหรับการวิเคราะห์ข้อมูล โดยจุดเด่นคือการใช้คำสั่งภาษา SQL ในการเรียกข้อมูล ทั้งที่อยู่ในรูปแบบของ Database และไฟล์บน Hadoop ได้ เหมาะสำหรับการเก็บข้อมูลขนาดใหญ่มาก ๆ

ประเภทของ File Format ของ Hive Table

การเก็บข้อมูลของ Table ใน Hive จะเก็บข้อมูลแบบไฟล์ข้อมูลที่ควบคุมการทำงานผ่าน HDFS ลักษณะของข้อมูลที่เราเห็นเป็นโครงสร้างตารางในแนว Row และ Column นั้น ตัว Hive Engine มีการแทนข้อมูลและอ่านข้อมูลเป็น File Format 2 แบบ คือ Row Major หรือ Column Major

Row Major

การแทนข้อมูลของ Record หนึ่งๆจำนวน Record ข้อมูลทั้งหมด จะวางเรียงต่อเนื่องแนวบรรทัด และ กระบวนเข้าถึงข้อมูลของ Record หนึ่งๆ แม้ว่าจะต้องการข้อมูลแค่ 1 Column ตัว MapReduce ก็จะต้องอ่านข้อมูลเต็มบรรทัดขึ้นมา ซึ่งมีผลต่อความเร็วในการเข้าถึงข้อมูลสำหรับการทำงานแบบ Concurrent เพราะจะมี I/O Wait เกิดขึ้นหาก แต่ละ Process ต้องการเข้าถึงข้อมูล Record เดียวกันด้วย

- TEXTFILE
- SEQUENCEFILE

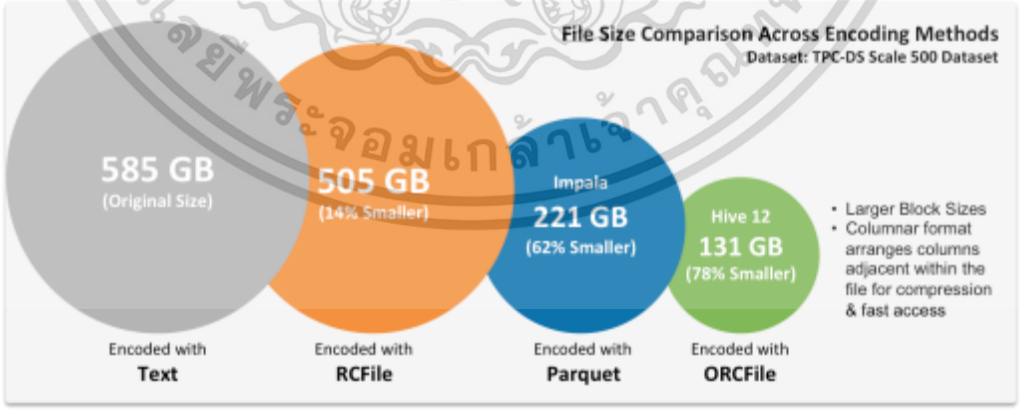
Column Major

การแทนข้อมูลของ Record หนึ่งๆจำนวน Record ข้อมูลทั้งหมด จะถูกแบ่งเป็นก้อนข้อมูลตามคอลัมน์ หรือ ตัดไฟล์ข้อมูลตามแนวคอลัมน์ โดยที่จะมีสร้างตัวชี้ข้อมูลที่จะนำก้อนชิ้นส่วนข้อมูลมาประกอบรวมกัน ดังนั้นการเข้าถึงข้อมูลที่ต้องการข้อมูลบางคอลัมน์ Hive Engine หรือ MapReduce จะเข้าอ่านข้อมูลเฉพาะไฟล์ของคอลัมน์ทีต้องการเท่านั้น ซึ่งส่งผลให้ลด I/O Wait ทำให้เริ่มความเร็วในการประมวลผลข้อมูลด้วย

- RFCFILE
- ORC
- PARQUET

รูปแบบไฟล์ ORC

เป็นวิธีที่มีประสิทธิภาพมากในการจัดเก็บข้อมูลเชิงสัมพันธ์กันซึ่งสามารถลดรูปแบบการจัดเก็บข้อมูลสูงสุดถึง 75% ของต้นฉบับ รูปแบบไฟล์ ORC นั้นทำงานมีประสิทธิภาพกว่ารูปแบบไฟล์อื่น ๆ เมื่อ Hive กำลังอ่านเขียนหรือประมวลผลข้อมูล รูปแบบไฟล์ ORC จะใช้เวลาในการเข้าถึงข้อมูลและใช้พื้นที่ในการจัดเก็บข้อมูลน้อย อย่างไรก็ตามไฟล์ ORC จะไปเพิ่ม Overhead ของ CPU โดยการเพิ่มเวลาที่ใช้ในการคลายการบีบอัดข้อมูลเชิงสัมพันธ์ นอกจากนี้รูปแบบไฟล์ ORC สามารถใช้งานกับ Hive ตั้งแต่เวอร์ชั่น 0.11 ขึ้นไป ไม่สามารถใช้กับรุ่นก่อนหน้านี้ได้



รูป 2.11 แสดงการเปรียบเทียบขนาดของไฟล์ Format ต่าง ๆ ของ Hive

คุณสมบัติเพิ่มเติมของ Hive Table เพื่อเพิ่มประสิทธิภาพความเร็วในการค้นหา

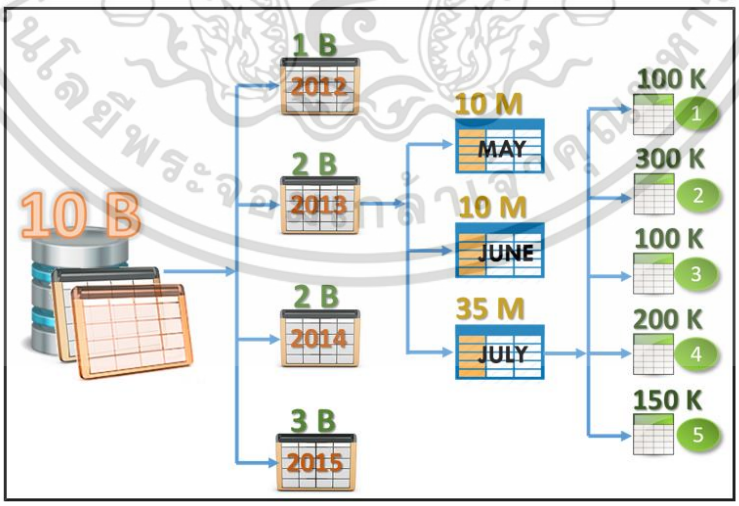
ปัจจัยที่มีผลต่อการทำให้ผลการค้นหาข้อมูลใช้เวลานานจากการเข้าถึงข้อมูลของ Table ที่ในระดับ Low Level เก็บข้อมูลแบบไฟล์ คือการกระจายตัวของข้อมูลหรือTransaction ไปอยู่ตามไฟล์ต่างๆ และ ปริมาณไฟล์ข้อมูลที่ต้องเข้าไปอ่านข้อมูลทั้งหมดที่เป็นจำนวนมาก

การแก้ปัญหาเพื่อเพิ่มความเร็วในการค้นหาข้อมูลจำเป็นต้องลดจำนวนของไฟล์ข้อมูลที่จะทำการค้นหาข้อมูล หรือ การให้ข้อมูลที่อ้างอิงเดียวกัน เช่น ข้อมูลลูกค้าเดียวกัน , ข้อมูลของเดือนเดียวกัน หรือ ข้อมูลของภาคเดียวกันถูกสร้างให้กองรวมอยู่ในไฟล์เดียวกัน เป็นต้น

เทคนิคการเพิ่มประสิทธิภาพการค้นหาข้อมูลของ Hive Table ทำได้ 2 แบบ คือ Partition Table และ Buckets Table

Partition Table

โดยปกติเวลาที่ทำการ Query ข้อมูลจาก Table ธรรมดา Hive จะทำการ Scan file ทั้งหมดใน Directory ของ Table ดังนั้นถ้าข้อมูลเป็น Table ขนาดใหญ่ หรือ มีไฟล์มาก ๆ จะส่งผลต่อประสิทธิภาพด้านความเร็วมาก ดังนั้นเทคนิค Partition Table คือ Hive จาก Sub-Directory ตาม Field ที่กำหนดให้ทำการ Partition ผลก็คือ Search Space ก็จะน้อยลงทำให้ค้นหาข้อมูลได้เร็วขึ้น เรากำหนดด้วย Keyword ชื่อ Partition By ([Col-Name1 DataType] ,..., [Col-Name2 DataType])



รูป 2.12 แสดงการแบ่ง Partition

Buckets Table

โดยปกติแล้วข้อมูลที่เก็บใน Table หรือ Directory จะเป็นไฟล์ข้อมูลแยกเป็นๆไฟล์หลายๆไฟล์ และ ข้อมูลที่โหลดเข้าระบบด้วยเทคนิคต่าง ๆตามรอบการทำงานก็จะกระจายไปอยู่ในไฟล์ต่างๆ ตามลำดับการเกิดของข้อมูล เช่น Transaction Data ของเหตุการณ์หนึ่งๆ ซึ่งใน Record Transaction อาจจะมี Customer ID อยู่ในนั้นด้วย ดังนั้นข้อมูล Transaction ของ Customer ID เดียวกันก็จะกระจายออกไปตามเวลาเกิดข้อมูล

ดังนั้นหากเราต้องการให้ Transaction ของ Customer ID เดียวกันอยู่ติดกันเป็นพื้นข้อมูลเดียวกัน เทคนิคของการทำ Buckets Table ถูกนำเข้ามาช่วยเพื่อจัดกลุ่มให้ข้อมูลใน Column ที่กำหนดให้จัดกลุ่ม Buckets อยู่ในไฟล์ข้อมูลเดียวกัน เพื่อความเร็วในการ Query ในการประมวลข้อมูลที่ระบบไม่ต้องไป Scan อ่านข้อมูลจากไฟล์ทั้งหมด

การทำ Dynamic partitioning ใน Hive

Apache hive เป็นคลังข้อมูลที่อยู่ด้านบนของ Hadoop ซึ่งช่วยให้การวิเคราะห์แบบเพาะกิจเกี่ยวกับข้อมูลที่มีโครงสร้างและกึ่งโครงสร้าง อย่างที่รู้กันว่าข้อมูลใน HDFS นั้นมีขนาดใหญ่จนทำให้ประสิทธิภาพในการ Query ข้อมูลนั้นลดลงได้เมื่อต้องจัดการกับข้อมูลที่มีขนาดใหญ่ Apache Hive จึงได้มีการแปลง Sql Query ให้เป็น MapReduce Jop ที่ทำงานอยู่บน Hadoop Cluster

การแบ่ง Partition เป็นหนึ่งในเทคนิคใน Hive ซึ่งช่วยปรับปรุงประสิทธิภาพได้เป็นอย่างมาก

ชนิดของการแบ่ง Partition

1) Static Partition (SP) Columns: ใน DML/DDL เกี่ยวข้องกับการแบ่ง Partition หลายคอลัมน์ ที่แบ่งโดยผู้ใช้โดยตรง
2) Dynamic Partition (DP) Columns ที่เกิดจากการแบ่งตอน Execute

รูป 2.13 Apache Hive

2.4.7 Apache Oozie

Apache Oozie เป็นบริการที่ใช้ในการจัดลำดับการงานเพื่อจัดการกับงานของ Apache Hadoop

รูป 2.14 Apache Oozie

2.4.8 Apache Spark

Apache Spark เป็นบริการที่ใช้ประมวลผลแบบคลัสเตอร์สำหรับข้อมูลขนาดใหญ่มีความรวดเร็ว

ข้อดีของ Apache Spark

1) รองรับ API ที่พัฒนาจากหลากหลายภาษาได้แก่ Java, Scala, Python และ R. Spark
2) รองรับการวิเคราะห์ผลข้อมูลจากแหล่งข้อมูลได้หลายแหล่ง เช่น Parquet, JSON, Hive และ Cassandra
3) รองรับการวิเคราะห์ผลข้อมูลหลายรูปแบบ เช่น ข้อมูลแบบ Text files, CSV และตาราง RDBMS
4) การประมวลผลของ Spark เป็นแบบเรียลไทม์ที่สามารถวิเคราะห์ผลได้อย่างรวดเร็ว

รูป 2.15 Apache Spark

2.4.9 Spark Streaming

Spark Streaming เป็นส่วนเสริมของ Spark API ที่ช่วยให้สามารถประมวลผลสตรีมข้อมูลให้สามารถปรับขนาดได้, ความเร็วสูงและคงทนต่อความผิดพลาด

รูป 2.16 Spark Streaming

2.4.10 Hadoop YARN (Yet Another Resource Negotiator)

Hadoop YARN (Yet Another Resource Negotiator) เป็น Framework ที่ใช้จัดการกับงานที่เกิดขึ้น (Job scheduling) และบริหารจัดการทรัพยากรต่าง ๆ (Resource Manager) บนระบบ Hadoop cluster

รูป 2.17 Hadoop YARN (Yet Another Resource Negotiator)

2.5 Power BI

Power BI คือโปรแกรมที่ใช้สำหรับการประมวลผลข้อมูลจำนวนมาก ๆ ให้อยู่ในรูปแบบของชาร์ทหรือตาราง เพื่อให้สามารถอ่านและนำไปประโยชน์ในทางธุรกิจต่อไป โดยการทำงานหลัก ๆ ของโปรแกรม Power BI คือ การนำฐานข้อมูลขนาดใหญ่ มาเชื่อมโยง และทำการวิเคราะห์ในรูปแบบของตารางที่มีการปรับเปลี่ยน หรือแสดงผลในรูปแบบของชาร์ท หรือกราฟ

รูป 2.18 Power BI

บทที่ 3
วิธีการดำเนินงาน

ในการทำระบบข้อมูลขนาดใหญ่สำหรับวิเคราะห์ข้อมูลการขนส่งนั้น จะต้องมีการจัดเตรียมและออกแบบระบบต่าง ๆเพื่อให้ระบบสามารถวิเคราะห์ข้อมูลได้ดังนี้

1) สถิติจำนวนรถยนต์ในแต่ละประเภทของปี 2561-2562
2) สถิติความเร็วเฉลี่ย และจำนวนการกระทำผิดความผิด(จากความเร็ว)ของรถต่อคันต่อเดือน แยกตามรายประเภทรถยนต์ ปี 2561 -2562
3) สถิติการกระทำความผิด(จากชั่วโมงการทำงาน) แยกตามชนิดรถ รายเดือน ของปี 2561-2562
4) สถิติการกระทำความผิด (จากความเร็ว) แยกตามพื้นที่

3.1 การเตรียม Server

เป็นขั้นตอนการวางแผนส่วนต่าง ๆของระบบเพื่อให้พร้อมใช้งาน

3.1.1 รายละเอียดของ Server ใน Cluster ที่ใช้

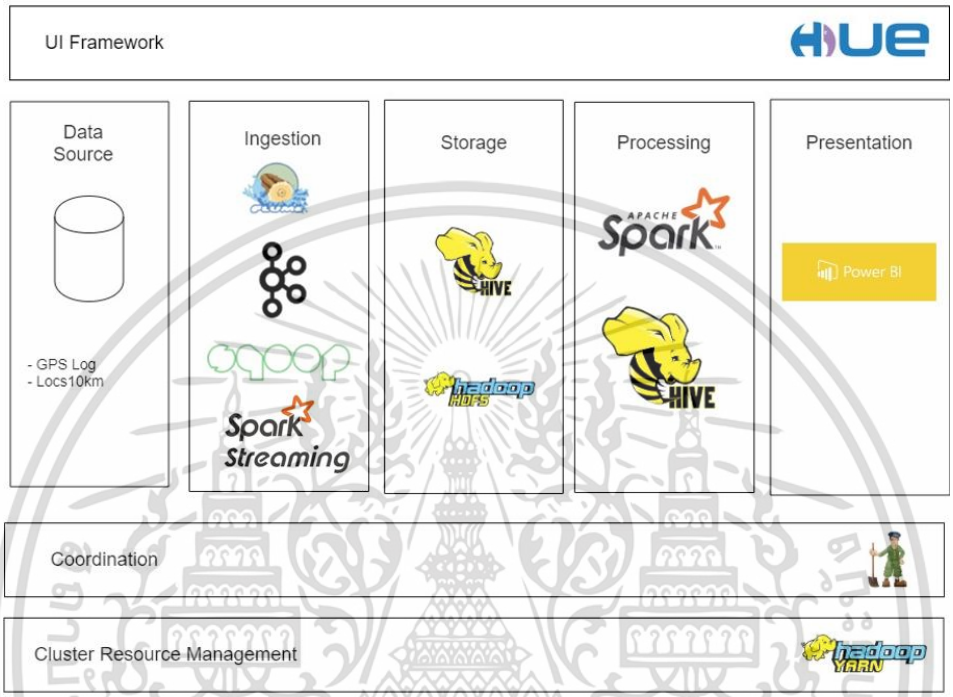
ในโครงงานนี้ใช้ Cluster ที่ประกอบด้วย Server จำนวน 5 ตัว แต่ละ Server มีรายละเอียดดังนี้

ตาราง 3. 1 แสดงรายละเอียด Spec ของ Server ที่ใช้ใน Cluster

IP Address	Hostname	OS	CPU	RAM	Disk
172.16.20.85	Nmtic-1	Ubuntu 14.04.6 LTS	8 cores	16 GB	992 GB
172.16.20.86	Nmtic-2	Ubuntu 14.04.6 LTS	8 cores	16 GB	2 TB
172.16.20.87	Nmtic-3	Ubuntu 14.04.6 LTS	8 cores	16 GB	2 TB
172.16.20.88	Nmtic-4	Ubuntu 14.04.6 LTS	8 cores	16 GB	2 TB
172.16.20.89	Nmtic-5	Ubuntu 14.04.6 LTS	8 cores	16 GB	2 TB

3.1.2 เครื่องมือที่ใช้และการกำหนดบทบาทให้กับแต่ละ Host ใน Cluster

ในการออกแบบระบบของโครงงานนี้เลือกใช้แพลตฟอร์มของ Hadoop ในการจัดการกับข้อมูลที่เป็น Big Data ซึ่งได้มีการใช้เครื่องมือต่าง ๆ ดังนี้



รูป 3.1 เครื่องมือต่าง ๆ ของHadoopที่ใช้ในโครงงาน

จากรูป 3.1 จะใช้เครื่องมือในส่วนต่าง ๆ ดังนี้

1) ส่วนของกานำเข้าข้อมูล ได้แก่ Spark streaming, Apache Flume และ Apache Kafka

2) ส่วนของการจัดเก็บข้อมูล ได้แก่ Hive และ HDFS ของ Hadoop

3) ส่วนของการประมวลผลข้อมูล ได้แก่ Hive และ Apache Spark

4) ส่วนของการแสดงผลข้อมูล ได้แก่ Power BI

5) ส่วน UI framework ได้แก่ Hue

6) ส่วนประสานงาน ได้แก่ Zookeeper

7) ส่วนจัดการทรัพยากร ได้แก่ Hadoop Yarn

ตาราง 3.2 รายละเอียดบทบาทของแต่ละ Host ใน Hadoop

Tool	Nmtic-1	Nmtic-2	Nmtic-3	Nmtic-4	Nmtic-5
Cloudera Management Service	- Alert Publisher - Event Server - Host Monitor - Reports Manager - Service Monitor	-	-	-	-
HDFS	-	DataNode	DataNode	DataNode	- NameNode -Secondary NameNode
Flume	-	Agent	Agent	Agent	Agent
Kafka	-	Kafka Broker	Kafka Broker	Kafka Broker	Kafka Broker
Spark2	-	Gateway	Gateway	Gateway	History Server
Hive	-Hive Metastore Server -HiveServer2	Gateway	Gateway	Gateway	Gateway
Oozie	Oozie Server	-	-	-	-
Zookeeper	Server	Server	Server	Server	Server

YARN (MR2 Included)	-	NodeManager	NodeManager	NodeManager	- JobHistory Server - Resource Manager
Hue	- HueServer - Load Balancer				

สำหรับตารางที่ 3.2 อธิบายบทบาทของ Host ต่างๆได้ ดังนี้

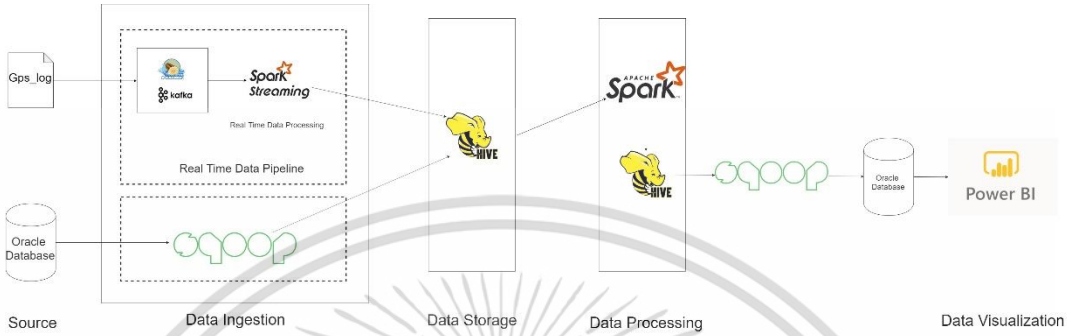
1) Nmtic-1 จะไม่ได้ทำงานในส่วนของ Hadoop หลักๆคือมีหน้าที่ในการดูแลจัดการกับการทำงานของ Cluster ตัวอื่นๆใน Hadoop ใช้ในการติดตั้ง Cloudera Manager ซึ่งจะเป็นตัวจัดการดูแลการทำงานต่างๆใน Cluster ของ Hadoop เป็น Hive Metastore Server และเป็น HueServer ที่เป็นหน้า UI สำหรับจัดการ Hive

2) Nmtic-5 มีหน้าที่เป็น Namenode ของ Hadoop ทำหน้าที่เก็บ meta data ทั้งหมดของเครื่องทุกเครื่องใน cluster ใช้เป็น JobHistory Server และ ResourceManager ของ Yarn เพื่อใช้ในควบคุมทรัพยากรและการจัดการงานให้ Datanode ใน Cluster ของ Hadoop

3) Nmtic-2, Nmtic-3, Nmtic-4 มีหน้าที่เป็น Datanode ใน Hadoop ทำหน้าที่ดูแลการจัดเก็บและบันทึกข้อมูลที่เป็นชิ้นข้อมูลย่อย ๆ

3.2 ภาพรวมการทำงานของระบบ

ในส่วนของการทำงานในระบบหลักๆสามารถแสดงแผนภาพได้ดังนี้



รูป 3.2 System Diagram

โครงการนี้ได้มีความต้องการที่จะจัดทำการประมวลผลข้อมูลสถิติการกระทำผิดทางการจราจร โดยอาศัยข้อมูลจาก GPS ของศูนย์บูรณาการขนส่งหลายรูปแบบแห่งชาติ (NMTIC) ซึ่งเป็นข้อมูลขนาดใหญ่และมีการส่งข้อมูลอยู่ตลอดเวลา ซึ่งในระบบของโครงการนี้แบ่งออกเป็น 4 ส่วน

1) ส่วนของการนำเข้าข้อมูล (Data Ingestion)

2) ส่วนจัดเก็บข้อมูล (Data Storage)

3) ส่วนประมวลผลข้อมูล (Data Processing)

4) ส่วนแสดงผลข้อมูล (Data Visualization)

โดยการทำงานเบื้องต้นมีดังนี้

3.2.1 ส่วนของการนำเข้าข้อมูล (Data Ingestion)

สำหรับการทำงานจริงข้อมูลที่กรมขนส่งส่งมานั้นมีขนาดประมาณ 400,000 Record/นาที และการรับข้อมูลนั้นต้องมีการออกแบบเพื่อรองรับข้อมูลที่มีขนาดใหญ่และมีประสิทธิภาพในการรองรับข้อมูลแบบ Real Time ในรูป 3.2 มีการรับข้อมูลจากแหล่งซึ่งได้แก่ ไฟล์ gps_log โดยใช้การทำงานร่วมกันของ Apache Flume และ Apache Kafka มาใช้ในการรองรับข้อมูล จากนั้นใช้ Spark Straming ในการแปลงข้อมูลที่ได้ให้มีรูปแบบที่เหมาะสมเพื่อนำไปเก็บลงใน Hive และ HDFS

3.2.2. ส่วนจัดเก็บข้อมูล (Data Storage)

ข้อมูลที่ถูกส่งมาแบบ Real Time จะถูกเก็บอยู่ใน Hive ซึ่ง Hive นั้นจะเก็บข้อมูลไว้ใน Metastore DB และ สร้าง Directory ที่เก็บไฟล์ข้อมูลของ Table ที่ควบคุมการทำงานผ่าน HDFS ของ Hadoop

3.2.3. ส่วนประมวลผลข้อมูล (Data Processing)

ในส่วนนี้จะนำข้อมูลจาก Hive มาประมวลผลโดยใช้ Apache Spark เพื่อหา

1) สถิติจำนวนรถยนต์ในแต่ละประเภทของปี 2018-2019 เพื่อให้ทราบว่าในแต่ละปีนั้นมีจำนวนรถยนต์แต่ละประเภทเพิ่มขึ้นหรือลดลงอย่างไรบ้าง
2) สถิติความเร็วเฉลี่ย และจำนวนการกระทำผิดความผิด(จากความเร็ว)ของรถต่อคันต่อเดือน แยกตามรายประเภทรถยนต์ ปี 2018 -2019 เพื่อให้ทราบว่าในแต่ละปีนั้นมีจำนวนการกระทำผิดความผิด(จากความเร็ว)เพิ่มขึ้นหรือลดลงอย่างไรบ้างและรถส่วนใหญ่ใช้ความเร็วเท่าไรในการเดินรถบนท้องถนน
3) สถิติการกระทำความผิด(จากชั่วโมงการทำงาน) แยกตามชนิดรถ รายเดือนของปี 2018-2019 2019 เพื่อให้ทราบว่าในแต่ละปีนั้นมีจำนวนกระทำความผิด(จากชั่วโมงการทำงาน) ของรถแต่ละประเภท เพิ่มขึ้นหรือลดลงอย่างไรบ้างและรถชนิดไหนที่มีการกระทำผิดสูดสุด ทำให้สามารถเพิ่มการควบคุมรถประเภทนั้นได้
4) สถิติการกระทำความผิด (จากความเร็ว) แยกตามพื้นที่ เพื่อให้ทราบว่าพื้นที่ไหน มีการกระทำผิด(จากความเร็ว)สูงสุด เนื่องจากพื้นที่ที่มีการกระทำผิด(จากความเร็ว)สูง ย่อมมีความเสี่ยงที่จะเกิดอุบัติเหตุมาก

3.2.4. ส่วนแสดงผลข้อมูล (Data Visualization)

หลังจากที่ประมวลผลของข้อมูลสำเร็จแล้ว ก็นำผลของข้อมูลที่ได้มาแสดงผลในรูปของ Dashboard โดยใช้ tool ที่ชื่อว่า Power Bi

3.3 โมเดลฐานข้อมูล

ข้อมูลที่จะนำส่งเข้าไปในระบบมี Format ของข้อมูลในรูป CSV ไฟล์ รายละเอียดของข้อมูลมีดังนี้

time_stamp	unit_id		lat	lon	speed	unit_type
2019-04-03 00:00:26	12300	04376	16.66349	98.53426	0	8
2019-04-03 00:00:26	22000	0000	12.72466	101.1704	0	7
2019-04-03 00:00:20	25000	0000	15.45471	100.3845	0	8
2019-04-03 00:00:06	22000	0000	16.09773	102.2606	0	7
2019-04-03 00:00:23	22000	0000	13.10158	100.904	0	6
2019-04-03 00:00:26	22000	0000	14.58931	101.0793	0	7
2019-04-03 00:00:00	22000	0000	14.59066	101.075	0	7
2019-04-03 00:00:33	03900	06904	13.69463	100.6995	46	7
2019-04-03 00:00:16	26000	0000	16.77805	101.2473	0	
2019-04-03 00:00:08	22000	0000	14.79561	100.9102	0	8
2019-04-03 00:00:52	50006	000	13.71246	101.4703	0	8
2019-04-03 00:00:35	06500	19018	18.7761	98.9904	0	3
2019-04-03 00:00:25	30009	000	15.78575	100.0851	0	8

รูป 3.3 ตัวอย่างชุดข้อมูล gps_log

ตาราง 3.3 รายละเอียดข้อมูลใน gps_log

ข้อ	ชื่อ	คำอธิบาย	ประเภทข้อมูล	ช่วงค่า	รายละเอียด
1	time_stamp	วัน/เวลาของข้อมูล (GPS Date/Time)	timestamp	DateTime UTC ISO Format	วัน/เวลาของข้อข้อมูล ส่งด้วยรูปแบบเวลาตามมาตรฐาน UTC [ISO 8601] ตัวอย่าง 2016-01-07T22:13:56.504Z
2	unit_id	หมายเลขประจำเครื่องบันทึกข้อมูลการเดินทางของรถ (GPS Unit Identifire)	Char (27)	ตัวอักษร ความยาว 27 ตัว	คือ หมายเลขประจำเครื่องบันทึกข้อมูลการเดินทางของรถกำหนดโดยหลักที่ 1-7 คือ gps_model_id เช่น 0010001,0020002 หลักที่ 8 -27 คือหมายเลขประจำเครื่องบันทึกการเดินทางของรถที่ผู้ให้บริการระบบติดตามรถเป็นผู้กำหนด

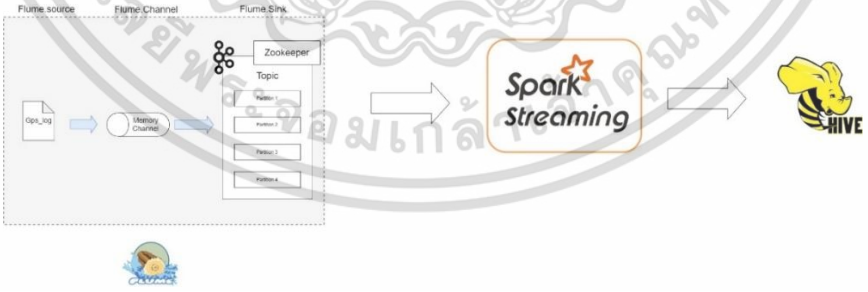
					20 หลัก กรณีที่หมายเลขประจำเครื่อง มีความยาวน้อยกว่า 20 ตัวจะต้องเติม “0” ข้างหน้าให้ครบ 20 ตัว เช่น (“00000000000345673232”) 1) บริษัท A มีหมายเลข vender_id = 1 อุปกรณ์หมายเลข gps_model_id = 0010001 ตัวอย่างหมายเลขประจำเครื่อง “000000LP-GPS-X2-0001” ดังนั้นหมายเลข unit_id คือ 0010001000000LP-GPS-X2-0001
3	lat	ละติจูด	float	+/-	ความถูกต้องของข้อมูลที่ต้องการค่าละติจูดที่ส่งมาต้องมีค่าทศนิยม 5 ตำแหน่งเป็นอย่างน้อย
4	lon	ลองติจูด	float	+/-	ความถูกต้องของข้อมูลที่ต้องการค่าลองติจูดที่ส่งมาต้องมีค่าทศนิยม 5 ตำแหน่งเป็นอย่างน้อย
6	speed	ความเร็ว	tinyint	0 to 255 km/h	ความเร็วมีหน่วยเป็น กิโลเมตรต่อชั่วโมง
7	unit_type	ชนิดของรถ	tinyint	0 to 255	แต่ละตัวเลขแทนชนิดของรถดังนี้ 1 = รถโดยสารประจำทาง 3 = รถโดยสารไม่ประจำทาง 4 = รถโดยสารส่วนบุคคล

					5 = รถบรรทุกส่วนบุคคล 6 = รถบรรทุกไม่ประจำทาง 7 = รถบรรทุกไม่ประจำทาง 8 = รถบรรทุกส่วนบุคคล 9 = รถบรรทุกส่วนบุคคล

3.4 การออกการทำงานและโครงสร้างสถาปัตยกรรมของระบบ

3.4.1 การออกแบบการนำเข้าข้อมูล (Data Ingestion)

การออกแบบการนำเข้าข้อมูลนั้นต้องคำนึงถึงความว่องไวและความทนต่อการรองรับข้อมูลแบบ Real-time ที่มีปริมาณมากได้ ซึ่งในโครงงานนี้ได้เลือกใช้ Apache Flume และ Apache Kafka มาทำงานร่วมกัน เนื่องจากการรับข้อมูลโดยใช้ Apache Flume สามารถทำงานบน Memory ได้ทำให้มีความว่องไวในการส่งมาก ส่วนใน Apache Kafka นั้นจะมีการทำงานที่เรียกว่า Partition โดยข้อมูลที่ได้จะไปเก็บแยกตามส่วนต่างๆใน Partition ของ Topic รวมถึงยังมีการทำสำเนาหรือ Replication เพื่อป้องกันความเสียหายของข้อมูล จากนั้นใช้ Spark Straming ในการดึงข้อมูลที่อยู่ใน Kafka Topic แบบ Real Time และนำข้อมูลที่ได้มาแปลงให้อยู่ในรูปเหมาะสมเพื่อนำไปจัดเก็บใน Hive ต่อไป โดยภาพรวมของการออกแบบการนำเข้าข้อมูลแสดงได้ดังรูป 3.4 (ก) นอกจากนี้ยังมีข้อมูลอื่นๆที่มีส่วนในการใช้ในการประมวลผลร่วมกับข้อมูล gps log ซึ่งข้อมูลส่วนนี้จัดเก็บอยู่ใน Oracle Database จึงต้องใช้ sqoop ซึ่งเป็นเครื่องมือที่ใช้นำเข้าข้อมูลจาก Database มาเก็บไว้ใน Hive ดังรูป 3.4 (ข)



ก)

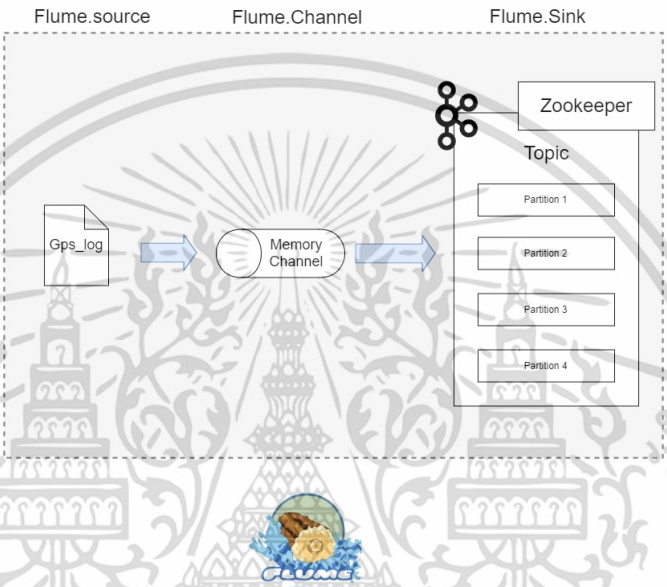
ข)

รูป 3.4 การนำเข้าข้อมูลของระบบแบบ

ก) การนำเข้าข้อมูลแบบ Real Time

ข) การนำเข้าข้อมูลจาก Oracle Database

3.4.1.1 การออกแบบการทำงานของ Apache Flume



รูป 3.5 การออกแบบการทำงานของ Flume

จากรูป 3.5 เริ่มแรกใช้ Flume ในการดึงข้อมูลจากแหล่งข้อมูลมาโดยผ่าน Channel ที่เป็น Memory ของ Flume Agent จากนำนำเข้ามูลไปเก็บไว้ใน Kafka Topic ซึ่งทำหน้าที่เป็น Sink ให้ Flume ในการทำงานของ Flume Agent 1 ตัว จะต้องประกอบไปด้วย Source Channel และ Sink ซึ่งมีส่วนการกำหนดค่าต่างๆของ Flume ดังนี้

3.4.1.1.1 การกำหนด Flume Source

```
#source
tier1.sources.spool.type = spooldir
tier1.sources.spool.channels = memoryChannel
tier1.sources.spool.spoolDir = /home/nmtic/gps
tier1.sources.spool.fileHeader = false
tier1.sources.spool.deletePolicy = immediate
```

รูป 3.6 การกำหนดค่าต่าง ๆ ของ Source ใน Flume

ตาราง 3.4 แสดงรายละเอียดการกำหนดค่าต่าง ๆ ของ Source ใน Flume

Property Name	การกำหนดค่า	คำอธิบาย
type	spooldir	เพื่อไปดึงข้อมูลที่เรามีใน Directory
channels	memoryChennel1	ชื่อตัวแปร memoryChennel ที่เราต้องการจะส่งข้อมูลผ่าน
spoolDir	/home/nmtic/data/gps-log	path ของ Directory ที่มีข้อมูลอยู่ โดย Flume จะทำการอ่านข้อมูลทั้ง Directory นั้น
fileHeader	False	การไม่นำ Header ของตารางข้อมูลมาด้วย
deletePolicy	immediate	การให้ลบไฟล์ใน Directory นั้นเมื่อ Flume อ่านข้อมูลเสร็จ

3.4.1.1.2 การกำหนด Flume Channel

ข้อดีของการใช้ Flume คือเราสามารถทำงานบน Memory ได้ ทำให้การนำเข้าข้อมูลมีความเร็วกว่าวิธีอื่น ๆ โดยพื้นที่ Memory ส่วนนี้จะใช้งานแค่ชั่วคราวเท่านั้น มีกำหนดค่าต่าง ๆ ของ Channel ที่เป็น Memory ดังนี้

```
#channel
tier1.channels.memoryChannel.type = memory
tier1.channels.memoryChannel.capacity = 10000
tier1.channels.memoryChannel.transactionCapacity = 1000
```

รูป 3.7 การกำหนดค่าต่าง ๆ ของ Channel ใน Flume

ตาราง 3.5 รายละเอียดการกำหนดค่าต่าง ๆ ของ Channel ใน Flume

Property Name	การกำหนดค่า	คำอธิบาย
type	memory	กำหนดให้ Channels ที่ใช้เป็น Memory ของ Flume Agent นั้น
Capacity	10000	คือการที่สามารถมีคิวอยู่ใน Channel ได้สูงสุด 10000 คิว
TransactionCapacity	1000	คือจำนวนสูงสุดที่จะนำข้อมูลจาก Source ไป Sink ต่อ 1 Transection

3.4.1.1.3 การกำหนด Flume Sink

สุดท้ายเราจะนำข้อมูลที่ได้มาเก็บลงใน Kafka Topic และใช้ Spark Streaming มาดึงข้อมูลเพื่อนำไปลง Hive ต่อไป

```
## Send to Kafka Broker on Hadoop Node
tier1.sinks.k1.channel = memoryChannel
tier1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
tier1.sinks.k1.batchSize = 600
tier1.sinks.k1.brokerList =  nmtic-2:9092,nmtic-3:9092,nmtic-4:9092,nmtic-5:9092
tier1.sinks.k1.topic = gps-nmtic
tier1.sinks.k1.zookeeperConnect = nmtic-1:2181,nmtic-2:2181,nmtic-3:2181,nmtic-4:2181,nmtic-5:2181/kafka
```

รูป 3.8 การกำหนดค่าต่าง ๆ ของ Sink ใน Flume

ตาราง 3.6 รายละเอียดการกำหนดค่าต่าง ๆ ของ Sink ใน Flume

Property Name	การกำหนดค่า	คำอธิบาย
channels	memoryChannel1	ชื่อตัวแปร memoryChennel ที่เราต้องการจะรับข้อมูลผ่าน
type	org.apache.flume.sink.kafka.KafkaSink	กำหนดให้flumeเป็นชนิดที่มี Kafka เป็น Sink
batchSize	100	กำหนดขนาดของ messages หน่วยเป็น byte ที่ process ใน 1 batch
brokerlist	nmtic-2:9092,nmtic-3:9092,nmtic-4:9092,nmtic-5:9092	Broker ที่เรากำหนดตอนสร้าง Topic ในที่นี้คือใช้จำนวน 4 Broker
topic	gps-nmtic	ชื่อของ Topic ใน Kafka
zookeeperConnect	nmtic-1:2181,nmtic-2:2182,nmtic-3:2181,nmtic-4:2181/kafka	เนื่องจากเราใช้ zookeeper ในการควบคุมงานใน Kafka จึงต้องกำหนด zookeeper สำหรับ Kafka Topic นั้นด้วย

3.4.1.2 การออกแบบการทำงานของ Apache Kafka

โครงงานนี้ใช้ Kafka Topic เพื่อรอรับข้อมูลที่มาจากต่างSource ต่าง Channel ในแต่ละ Topic ของ Kafka ใช้ Broker จำนวน 4 ตัว โดยใช้ Zookeeper มาช่วยในการควบคุมการทำงานของ Broker ใน KafkaTopic ด้วย กำหนดให้มีทำงาน 4 Partition และแต่ละ Partition มีการกำหนดให้ทำสำเนา Replicasion = 2

```
# /usr/bin/kafka-topics --create --zookeeper nmtic-1:2181,nmtic-2:2181,nmtic-3:2181,nmtic-4:2181,nmtic-5:2181/kafka --replication-factor 2 --partitions 4 --topic gps.nmtic
```

ก)

```
Topic:gps.nmtic PartitionCount:4        ReplicationFactor:2     Configs:
        Topic: gps.nmtic        Partition: 0    Leader: 145     Replicas: 145,146       Isr: 145,146
        Topic: gps.nmtic        Partition: 1    Leader: 146     Replicas: 146,148       Isr: 148,146
        Topic: gps.nmtic        Partition: 2    Leader: 148     Replicas: 148,144       Isr: 144,148
        Topic: gps.nmtic        Partition: 3    Leader: 144     Replicas: 144,145       Isr: 144,145
```

ข)

```
----------------------------------------
(None, u'2018-05-09 00:54:10,002000200          03605,14.639235,98.6856116666667,0,3\r')
(None, u'2018-05-09 00:54:30,036000000          3179,13.121567,100.97355,0,6\r')
(None, u'2018-05-09 00:54:16,002000200          02241,16.2960733333333,102.791763333333,0,\r')
(None, u'2018-05-09 00:54:18,030000100          _494,13.602805,100.784187,0,\r')
(None, u'2018-05-09 00:54:26,023000300          1039,13.803313,100.156762,0,\r')
(None, u'2018-05-09 00:54:23,002000100          14476,14.1404466666667,100.617811666667,0,7\r')
(None, u'2018-05-09 00:54:24,030000100          _367,13.775459,99.933967,0,7\r')
(None, u'2018-05-09 00:54:26,033000400          9239,12.04089,102.297997,0,3\r')
(None, u'2018-05-09 00:54:27,033000400          4290,13.790295,100.475807,0,3\r')
(None, u'2018-05-09 00:54:28,033000400          9580,7.06058,100.378204,0,\r')
...
```

ค)

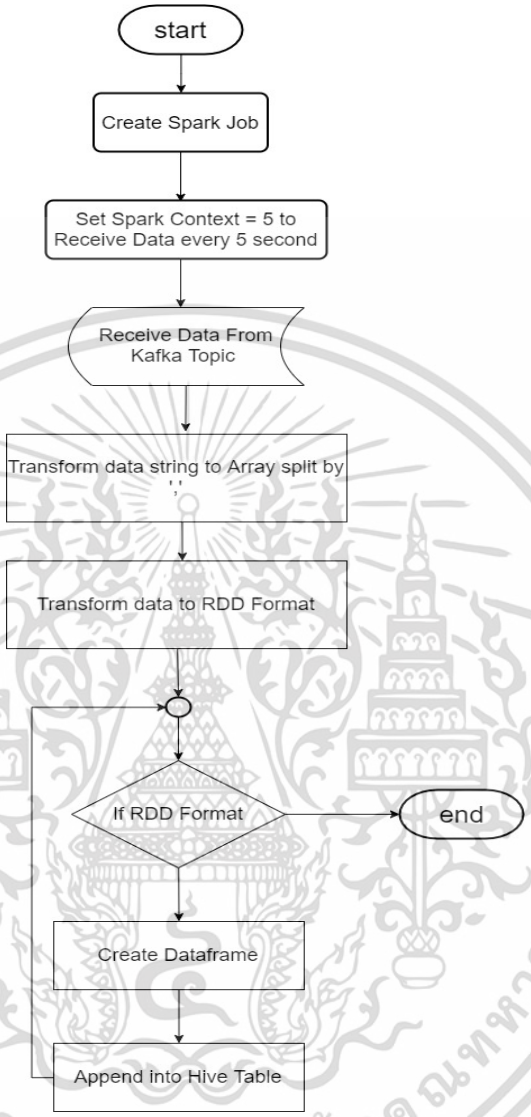
รูป 3.9 การสร้าง Topic ใน Kafka

ก) คำสั่งที่ใช้ในการสร้าง Topic

ข) รายละเอียดของ Topic ที่สร้าง

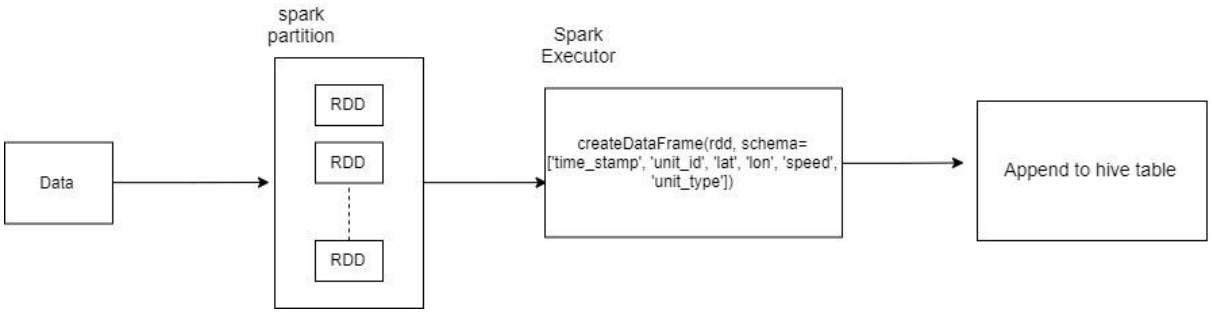
ค) ตัวอย่างข้อมูลที่ส่งจาก Kafka Topic ใน Format Unicode

3.4.1.3 การออกแบบการทำงานของ Spark streaming



รูป 3.10 Flow chart การทำงานภายใน Spark streaming

ในตอนแรกข้อมูลที่ได้รับจาก Kafka Topic จะเป็นชุดข้อมูล string ในรูปของ format unicode เราจึงต้องทำการแปลงข้อมูลที่ได้มาให้อยู่ในรูปของ Array แบ่งโดย ‘,’ จากนั้นนำข้อมูลมาแปลงให้อยู่ในรูป RDD ซึ่งข้อมูลในรูปแบบของ RDD มีส่วนสำคัญในการทำงานของ Spark เป็นอย่างมากเนื่องจาก มันจะทำการแบ่งข้อมูลออกเป็น partition ไปยังแต่ละ executor ของ Spark เพื่อความรวดเร็วในการประมวลผล โดยเราจะนำข้อมูลแต่ละ RDD มาสร้างเป็น Dataframe เพื่อนำไป append ลงในตารางของ Hive



รูป 3.11 การทำงานภายใน Spark streaming

time_stamp	unit_id	lat	lon	speed	unit_type	date
2018-05-09	00:14:17	0410...	7.85424	98.352573	0	3	
2018-05-09	00:14:20	0330...	7.790705	99.460045	0	3	
2018-05-09	00:14:10	0060...	13.8072	100.05646	0	7	
2018-05-09	00:14:58	0060...	14.60157	101.03972	0	7	
2018-05-09	00:14:20	0330...	13.90492	100.691223	0		
2018-05-09	00:14:38	0060...	18.54464	99.56943	0	7	
2018-05-09	00:14:02	0060...	13.7361	100.76423	0	7	
2018-05-09	00:14:04	0790...	17.393548333333332	102.79797166666667	0	1	
2018-05-09	00:14:38	0060...	19.15317	99.91276	0	7	
2018-05-09	00:14:19	0670...	10.53062	99.2213	0	7	
2018-05-09	00:14:18	0790...	12.831863166667	101.2405575	0		
2018-05-09	00:14:18	0410...	13.055993	101.115761	0	3	
2018-05-09	00:14:20	0060...	20.03081	99.28716	69		
2018-05-09	00:14:21	0330...	17.346975	104.157387	0	7	
2018-05-09	00:14:18	0060...	13.42542	101.02332	0	8	
2018-05-09	00:14:22	0740...	14.627205	102.839865	0	7	
2018-05-09	00:14:19	0300...	14.752732	103.059184	0		
2018-05-09	00:14:19	0110...	13.505995	100.729103	0	3	
2018-05-09	00:14:24	1250...	13.75138	100.49691	0		
2018-05-09	00:14:25	0290...	14.296668	100.61164	0		

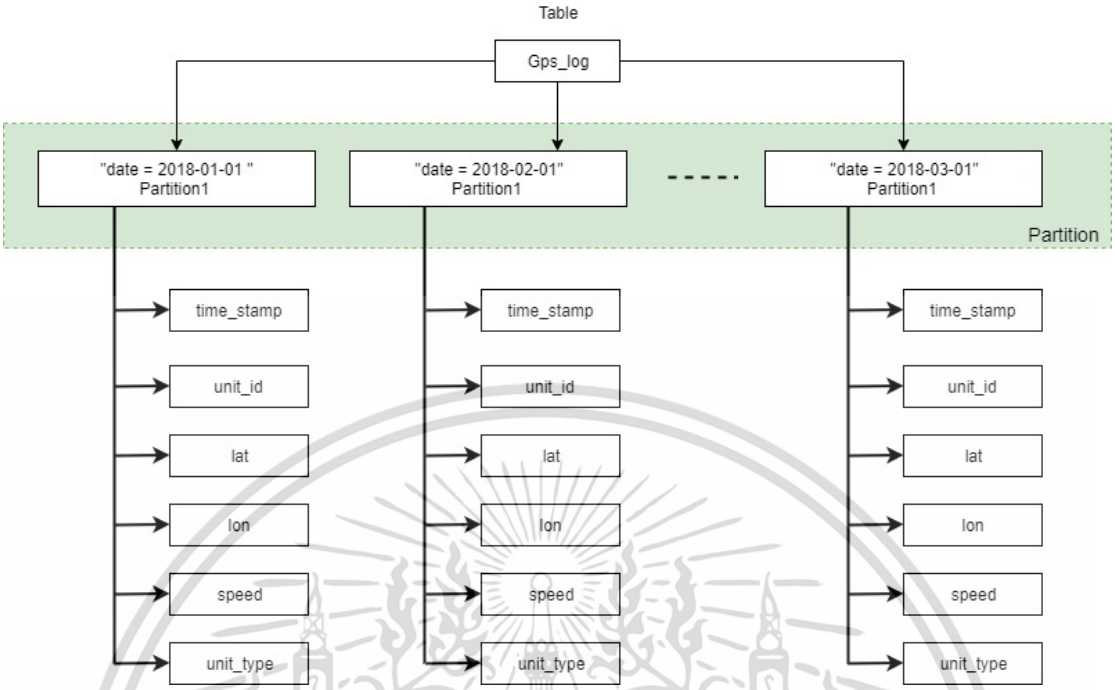
only showing top 20 rows

รูป 3.12 หน้าตาของ Dataframe ที่ได้จากการสร้างโดย Spark streaming

3.4.2 การออกแบบการจัดเก็บข้อมูล (Data Storage)

หลังจากที่เรานำข้อมูลเข้ามาแล้วข้อมูลที่ได้จะจัดเก็บอยู่ในรูปของ HDFS กระจายไปอยู่ตาม Datanode ต่างๆใน Hadoop ซึ่งมี Hive ที่สร้างอยู่บน HDFS จึงเป็น tool ตัวหนึ่งที่ใช้สำหรับการ Query ข้อมูลบน HDFS ในการออกแบบการจัดเก็บข้อมูลเราต้องคำนึงถึงความไวในการ Query เนื่องจากข้อมูลมีเป็นจำนวนมาก ต้องออกแบบให้ลดเวลาในการ Query มากที่สุด

3.4.2.1 การออกแบบการจัดเก็บของ Apache Hive



รูป 3.13 แผนภาพการจัดเก็บข้อมูลใน gps_log table

ในส่วนของการออกแบบตารางเก็บข้อมูลจาก gps_log นั้น แบ่ง Partition จำแนกออกเป็นรายวัน เพื่อทำให้เกิดความว่องไวในการ Query มากยิ่งขึ้น

3.4.3 การออกแบบการประมวลผลข้อมูลและการแสดงผล (Data Processing & Data Visualization)

รูป 3.14 แผนภาพการประมวลผลข้อมูลและการแสดงผลข้อมูล

หลังจากที่เรานำเข้าข้อมูลมาเก็บไว้ที่ HDFS แล้วเราจะนำข้อมูลที่ได้มาประมวลผลเพื่อหา

1) สถิติจำนวนรถยนต์ในแต่ละประเภทของปี 2561-2562

2) สถิติความเร็วเฉลี่ย และจำนวนการกระทำผิดความผิด(จากความเร็ว)ของรถต่อคันต่อเดือน แยกตามรายประเภทรถยนต์ ปี 2561 -2562

3) สถิติการกระทำความผิด(จากชั่วโมงการทำงาน) แยกตามชนิดรถ รายเดือนของปี 2561-2562

4) สถิติการกระทำความผิด (จากความเร็ว) แยกตามพื้นที่

โดยการประมวลผลจะใช้ Apache Spark และ จัดเก็บผลที่ได้ลงใน Hive และใช้ Sqoop ดึงข้อมูลที่ต้องการจัดเก็บเช่นข้อมูลจากการสรุปผลจากปีนั้น ๆ มาลง Postgres Database จากนั้นใช้ Power Bi นำข้อมูลจาก Table ต่าง ๆ ของ Postgres มาแสดงเป็น Dashboard

ตาราง 3.7 รายละเอียดการกำหนดค่าต่าง ๆ ของ gps_log

ข้อ	ชื่อ	คำอธิบาย	ประเภทข้อมูล	ช่วงค่า	รายละเอียด
1	time_stamp	วัน/เวลาของข้อมูล (GPS Date/Time)	timestamp	DateTime UTC ISO Format	วัน/เวลาของข้อข้อมูล ส่งด้วยรูปแบบเวลาตามมาตรฐาน UTC [ISO 8601] ตัวอย่าง 2016-01-07T22:13:56.504Z
2	unit_id	หมายเลขประจำเครื่องบันทึกข้อมูลการเดินทางของรถ (GPS Unit Identifire)	Char (27)	ตัวอักษรความยาว 27 ตัว	คือ หมายเลขประจำเครื่องบันทึกข้อมูลการเดินทางของรถกำหนดโดยหลักที่ 1-7 คือ gps_model_id เช่น 0010001,0020002 หลักที่ 8 -27 คือหมายเลขประจำเครื่องบันทึกการเดินทางของรถที่ผู้ให้บริการระบบติดตามรถเป็นผู้กำหนด 20 หลัก กรณีที่หมายเลขประจำเครื่อง มีความยาวน้อยกว่า 20 ตัวจะต้องเติม “0” ข้างหน้าให้ครบ 20 ตัว เช่น (“00000000000345673232”) 1) บริษัท A มีหมายเลข vender_id = 1 อุปกรณ์หมายเลข gps_model_id = 0010001 ตัวอย่างหมายเลขประจำเครื่อง “000000LP-GPS-X2-0001” ดังนั้นหมายเลข unit_id คือ

					0010001000000LP-GPS-X2-0001
3	Lat	ละติจูด	float	+/-	ความถูกต้องของข้อมูลที่ต้องการค่าละติจูดที่ส่งมาต้องมีค่าทศนิยม 5 ตำแหน่งเป็นอย่างน้อย
4	Lon	ลองติจูด	float	+/-	ความถูกต้องของข้อมูลที่ต้องการค่าลองติจูดที่ส่งมาต้องมีค่าทศนิยม 5 ตำแหน่งเป็นอย่างน้อย
6	speed	ความเร็ว	tinyint	0 to 255 km/h	ความเร็วมีหน่วยเป็น กิโลเมตรต่อชั่วโมง
7	unit_type	ชนิดของรถ	tinyint	0 to 255	แต่ละตัวเลขแทนชนิดของรถดังนี้ 1 = รถโดยสารประจำทาง 3 = รถโดยสารไม่ประจำทาง 4 = รถโดยสารส่วนบุคคล 5 = รถบรรทุกส่วนบุคคล 6 = รถบรรทุกไม่ประจำทาง 7 = รถบรรทุกไม่ประจำทาง 8 = รถบรรทุกส่วนบุคคล 9 = รถบรรทุกส่วนบุคคล

ตาราง 3.8 รายละเอียดการกำหนดค่าต่าง ๆ ของ illegal_by_speed

ชื่อ	ประเภทข้อมูล	คำอธิบาย
year	Int	ปี
month	Int	เดือน
unit_id	Char (27)	หมายเลขประจำเครื่องบันทึกข้อมูลการเดินทางของรถ (GPS Unit Identifire
avg_speed	Float	ความเร็วเฉลี่ยของรถ
cnt_illegal	int	จำนวนการกระทำผิด (จากความเร็วรถ)
unit_type	Int	ชนิดของรถ

ตาราง 3.9 รายละเอียดการกำหนดค่าต่าง ๆ ของ illegal_by_time

ชื่อ	ประเภทข้อมูล	คำอธิบาย
year	Int	ปี
month	Int	เดือน
unit_id	Char (27)	หมายเลขประจำเครื่องบันทึกข้อมูลการเดินทางของรถ (GPS Unit Identifire
cnt_illegal	int	จำนวนการกระทำผิด (จากชั่วโมงการทำงาน)
unit_type	Int	ชนิดของรถ

ตาราง 3.10 รายละเอียดการกำหนดค่าต่าง ๆ ของ illegal_speed_by_area

ชื่อ	ประเภทข้อมูล	คำอธิบาย
year	Int	ปี
month	Int	เดือน
lat	Float	ละติจูด

lon	Float	ลองติจูด
district	String	อำเภอ/เขต
province	String	จังหวัด
cnt_illegal	Int	จำนวนการกระทำผิด (จากความเร็วรถ)

ตาราง 3.11 รายละเอียดการกำหนดค่าต่าง ๆ ของ numcartype

ชื่อ	ประเภทข้อมูล	คำอธิบาย
year	Int	ปี
month	Int	เดือน
count_unit_type	Int	จำนวนรถแยกตามชนิดของรถ
unit_type	Int	ชนิดของรถ

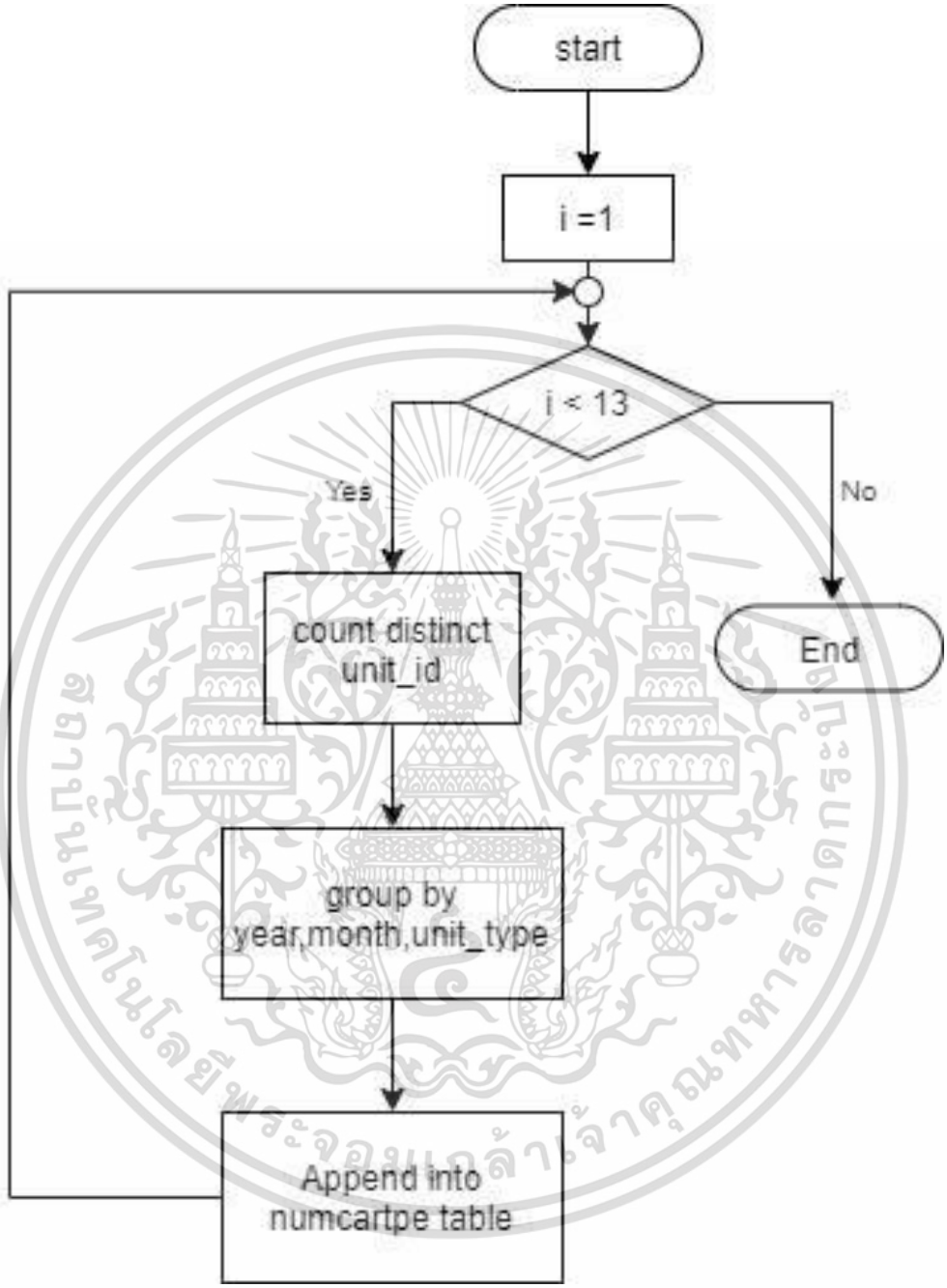
ตาราง 3.12 รายละเอียดการกำหนดค่าต่าง ๆ ของ overspeed

ชื่อ	ประเภทข้อมูล	คำอธิบาย
time_stamp	timestamp	วัน/เวลา ของข้อมูล (GPS Date/Time)
lat	Int	ละติจูด
lon	Char (27)	ลองติจูด
unit_id	Float	หมายเลขประจำเครื่องบันทึกข้อมูลการเดินทางของรถ (GPS Unit Identifire
speed	tinyint	ความเร็วรถ
time_overspeed	Float	ระยะเวลาที่รถมีความเร็วเกินกำหนดติดต่อกัน (นาที)
unit_type	Int	ชนิดของรถ
Date	date	วันที่

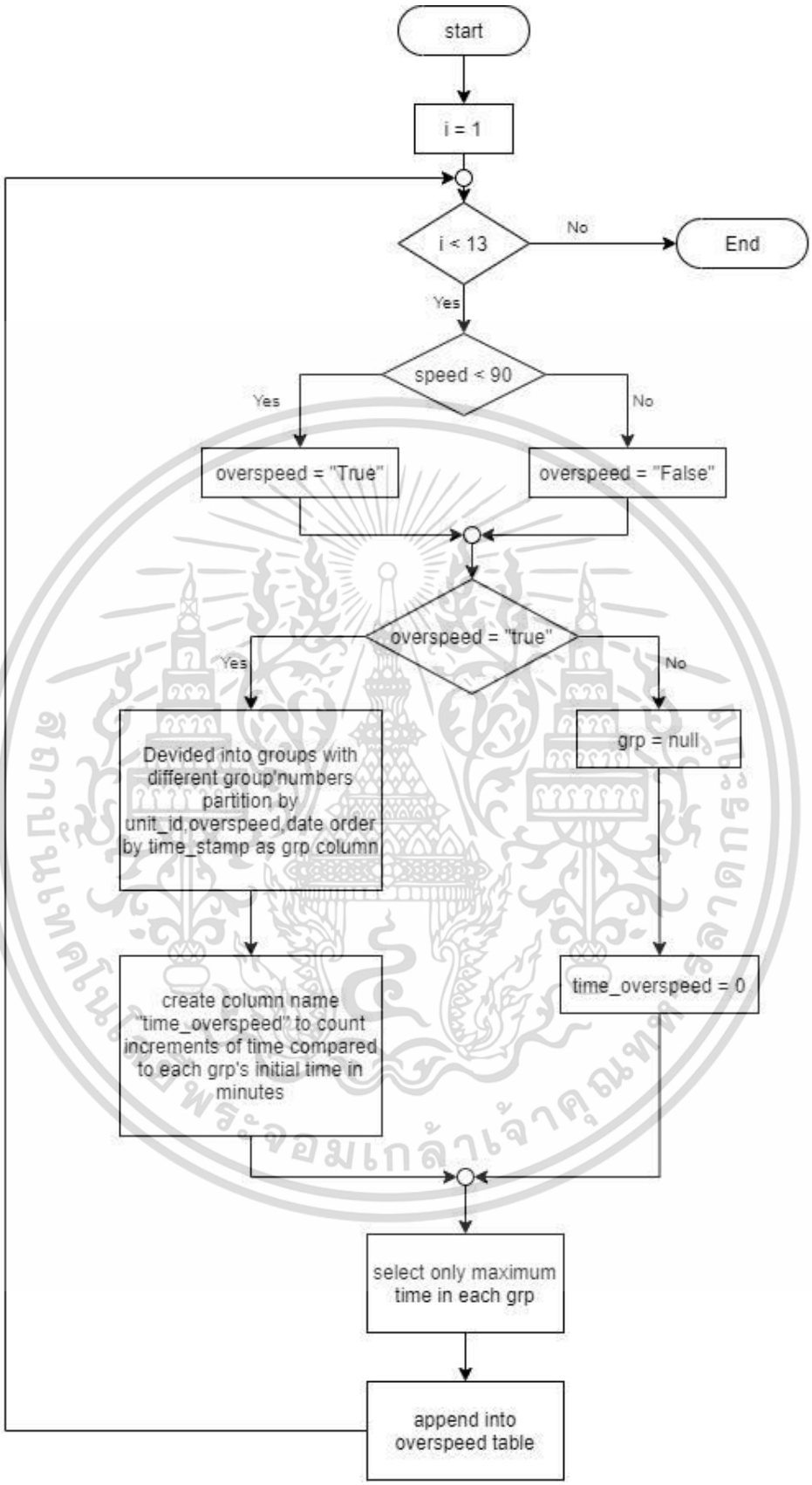
ตาราง 3.13 รายละเอียดการกำหนดค่าต่าง ๆ ของ overtime

ชื่อ	ประเภทข้อมูล	คำอธิบาย
time_stamp	timestamp	วัน/เวลา ของข้อมูล (GPS Date/Time)
lat	Int	ละติจูด
lon	Char (27)	ลองติจูด
unit_id	Float	ชนิดของรถ
speed	tinyint	ความเร็วของรถ
time_active	Float	ระยะเวลาที่รถมีความเร็วมากกว่า 0 km/hr ติดต่อกัน (นาที)
time_nonactive	Float	ระยะเวลาที่รถมีความเร็วเป็น 0 km/hr ติดต่อกัน (นาที)
unit_type	Int	ชนิดของรถ
Date	date	วันที่

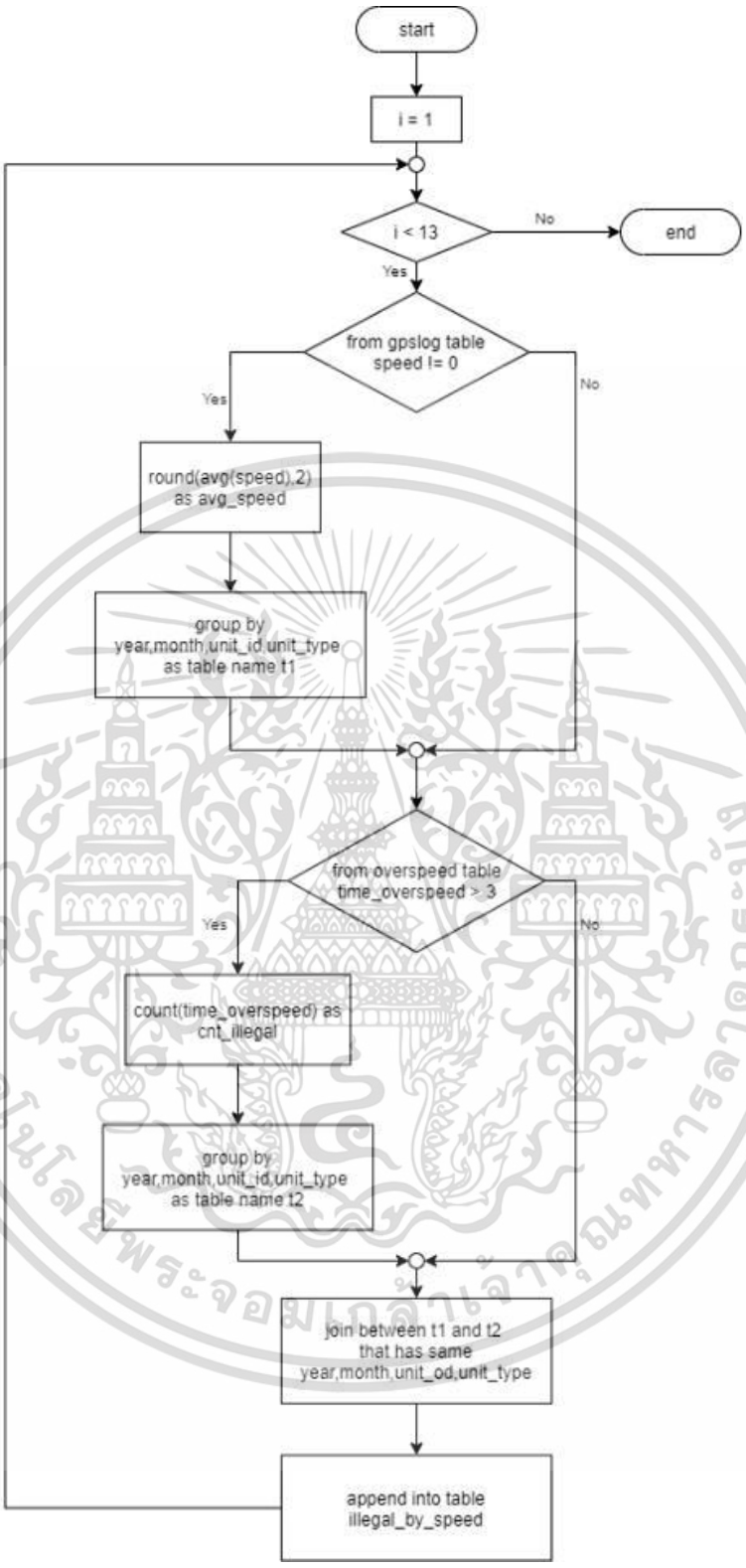
3.4.3.1 Flowcharts แสดงการคำนวณการวิเคราะห์ข้อมูล



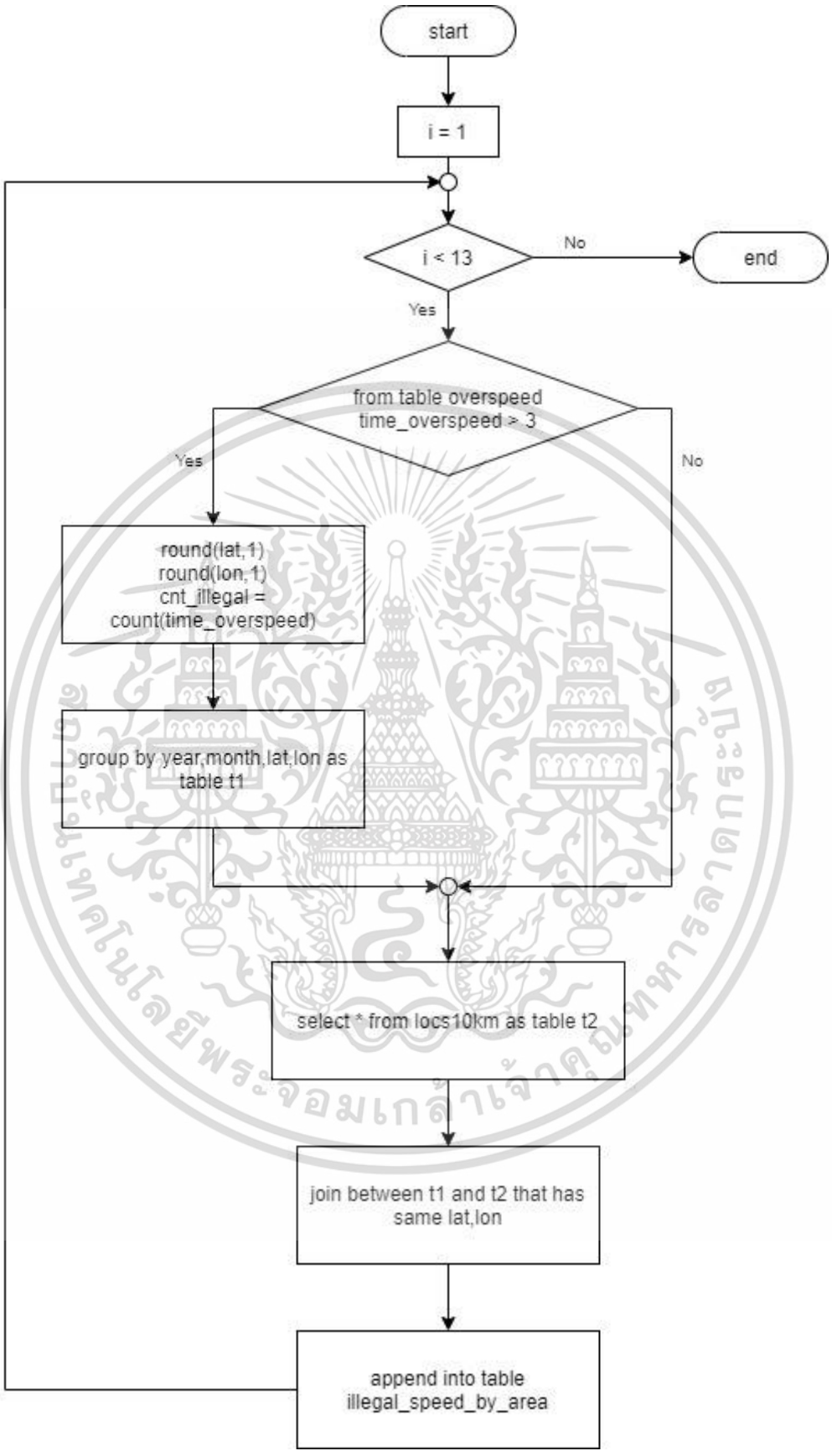
รูป 3.15 Flowchart แสดงการวิเคราะห์ข้อมูลเพื่อเพิ่มในตาราง numcartype



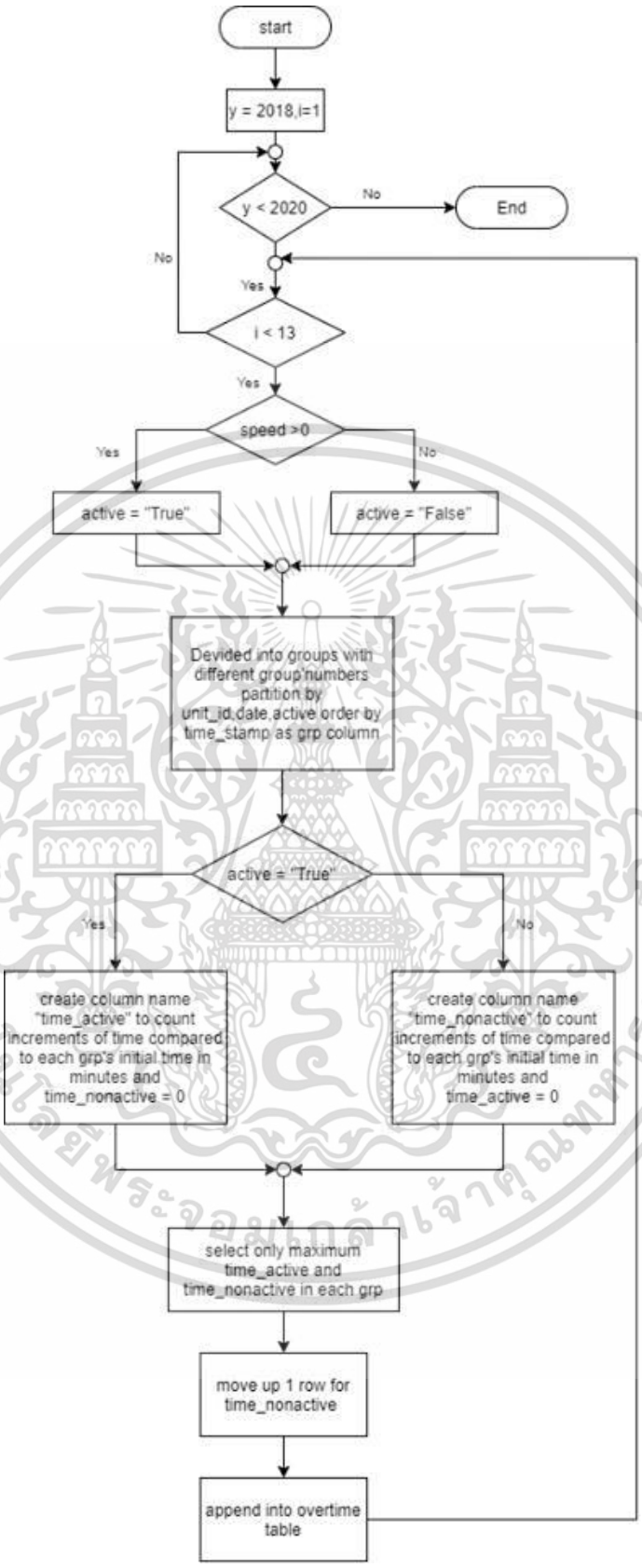
รูป 3.16 Flowchart แสดงการวิเคราะห์ข้อมูลเพื่อเพิ่มในตาราง overspeed



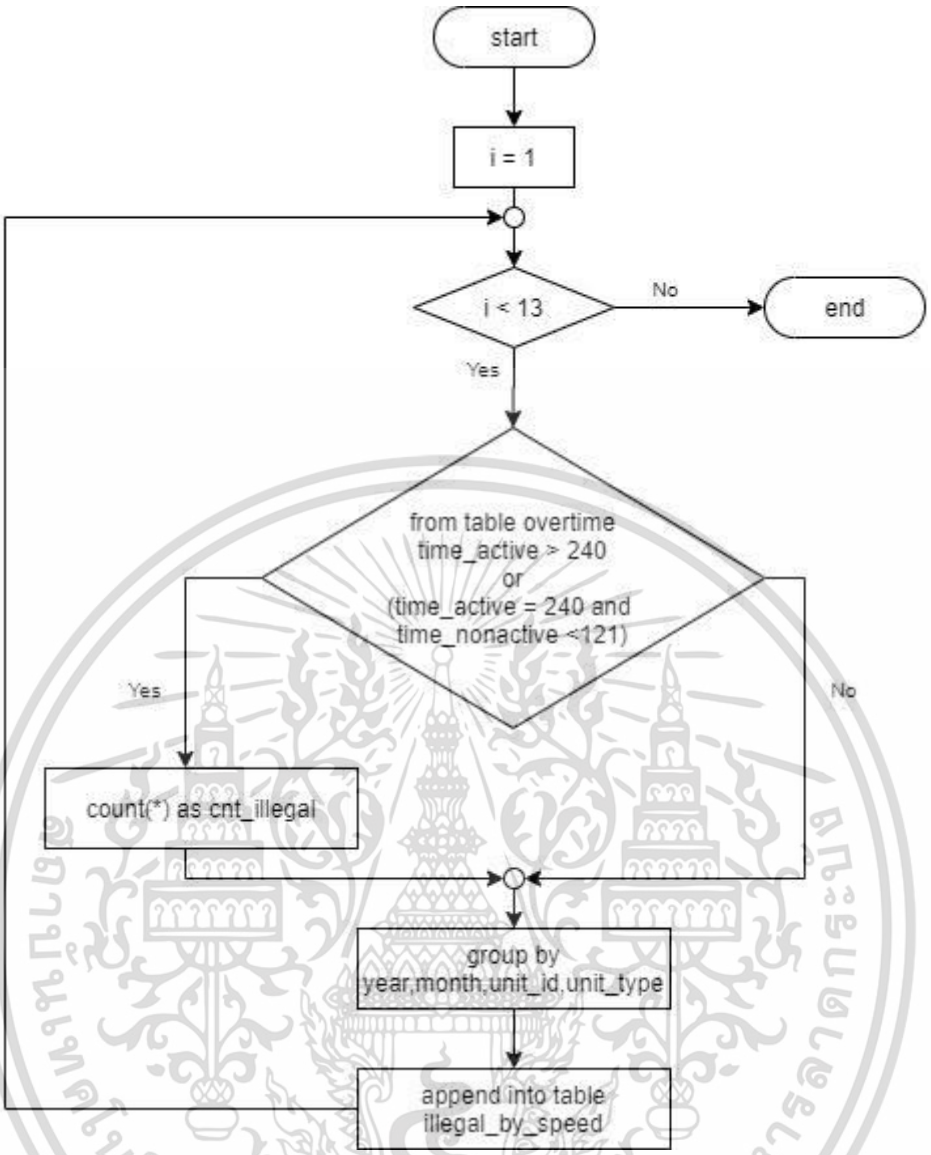
รูป 3.17 Flowchart แสดงการวิเคราะห์ข้อมูลเพื่อเพิ่มในตาราง illegal_by_speed



รูป 3.18 Flowchart แสดงการวิเคราะห์ข้อมูลเพื่อเพิ่มในตาราง illegal_speed_by_area



รูป 3.19 Flowchart แสดงการวิเคราะห์ข้อมูลเพื่อเพิ่มในตาราง overtime



รูป 3.20 Flowchart แสดงการวิเคราะห์ข้อมูลเพื่อเพิ่มในตาราง illegal_by_time

บทที่ 4

การทดลอง

หลังจากที่ได้ออกแบบระบบ Big Data ของโครงงานแล้ว ก็ได้เริ่มทำการทดลองในระบบหลักๆ ของโครงงาน โดยประกอบไปด้วย 4 การทดลอง คือ การทดลองการนำเข้าข้อมูลไปเก็บไว้ใน Hive ในรูปแบบของตารางโดยแบ่ง Partition ตามวันที่ , การทดลองการวิเคราะห์ข้อมูล , การทดลองนำข้อมูลออกจากตารางใน Hive ไปยัง Postgres database โดยใช้ Sqoop และ การทดลองการแสดงผลข้อมูล

4.1 การทดลองการนำเข้าข้อมูลไปเก็บไว้ใน Hive ในรูปแบบของตารางโดยแบ่ง Partition ตามวันที่

การทดลองในส่วนนี้จะครอบคลุมตั้งแต่การนำเข้าข้อมูลโดยใช้ Apache Flume และ Apache Kafla Spark Streaming เพื่อให้สามารถรองรับการนำเข้าแบบ Real-time ของข้อมูลขนาด 400,000 Record/นาที ได้ รวมไปจนถึงการจัดเก็บข้อมูลใน Hive บน HDFS เพื่อที่จะนำข้อมูลส่วนนี้ไปใช้ในการวิเคราะห์ต่อไป

4.1.1 การทดสอบการนำเข้าของข้อมูลโดยใช้การทำงานร่วมกันของ Apache Flume และ Kafka

4.1.1.1 วัตถุประสงค์

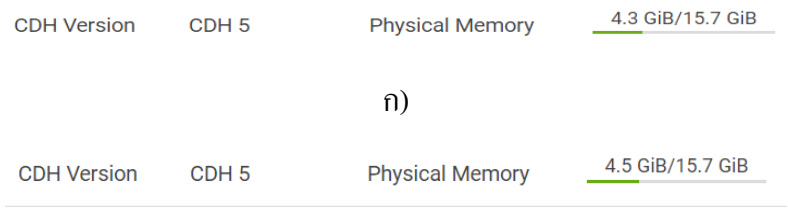
เพื่อทดสอบว่าการตั้งค่าของ Flume และการสร้าง Topic ของ Kafka สามารถนำเข้าข้อมูลที่มีจำนวน 400,000 Record/นาที ได้

4.1.1.2 วิธีการทดลอง

ทดลองส่งไฟล์ CSV ที่มีขนาด 1,000,000 บรรทัด เข้าไปยังระบบ ดูขนาดของ Memory Channel ที่ใช้ในการส่งของ Flume และ บันทึกผลเวลาที่ใช้ในการส่งข้อมูล Cloudera Manager ของ Kafka

4.1.1.3 ผลการทดลอง

จากรูปด้านล่างแสดงการใช้ Physical Memory ซึ่งเป็นส่วน Channels ใน Flume จะเห็นได้ว่าข้อมูล 1 ล้าน record ใช้พื้นที่ 0.2 GB ใน Flume Agent 1 ตัว แสดงให้เห็นว่า flume สามารถรองรับข้อมูลขนาด 400,000 Record/นาที ได้



ก)

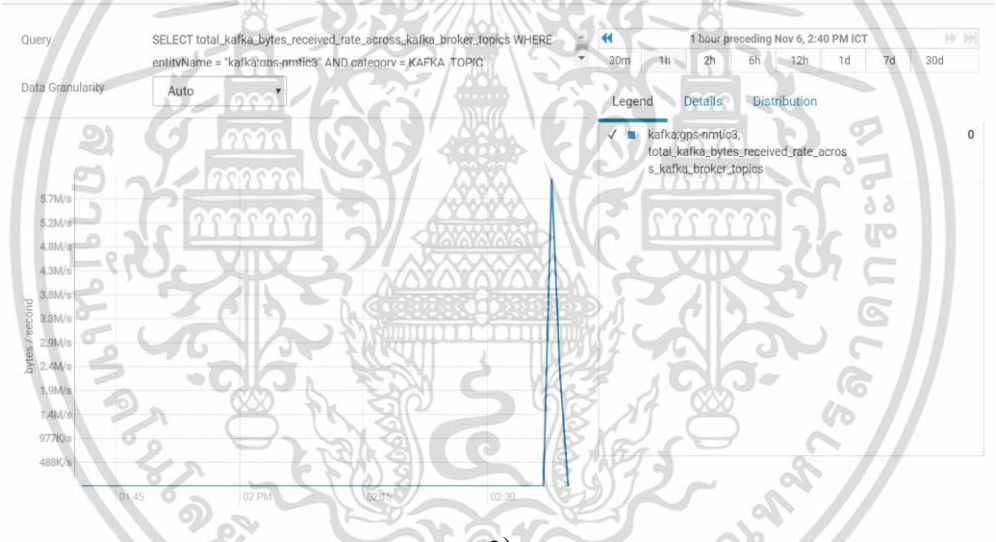
ข)

รูป 4.1 Physical Memory

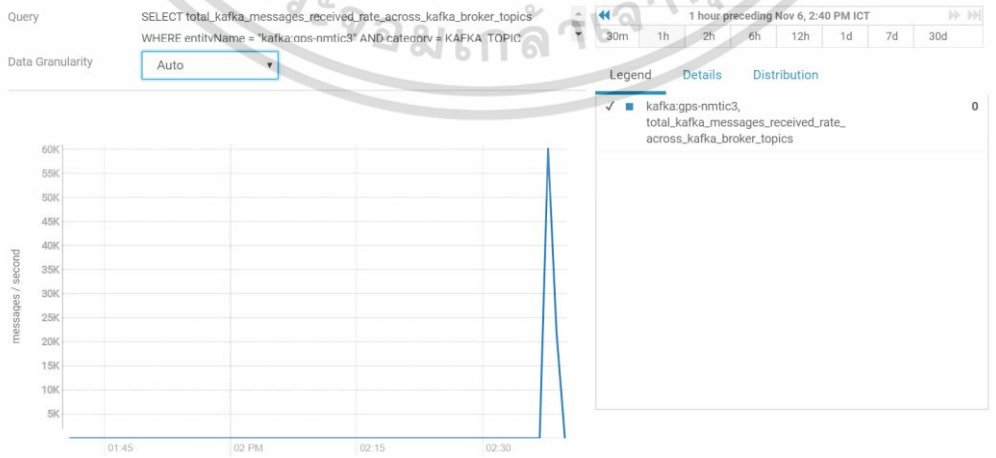
ก) Physical Memory ก่อนการทดลอง

ข) Physical Memory หลังการทดลอง

ภาพด้านล่างแสดงกราฟการรับข้อมูลเข้า Topic ของ Kafka บน Cloudera Manager



ก)



ข)

รูป 4.2 กราฟการรับเข้าข้อมูลเข้า Topic ของ Kafka Producer

ก) กราฟการรับเข้าข้อมูลเข้า Topic ของ Kafka ในหน่วย MiB/Sec

ข) กราฟการรับเข้าข้อมูลเข้า Topic ของ Kafka ในหน่วย Message/Sec

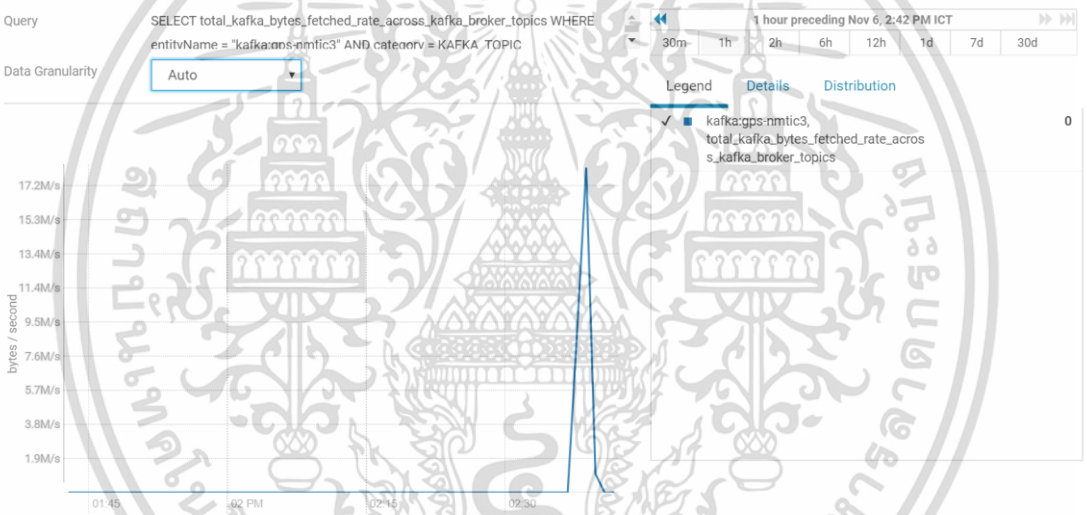
จากรูป 4.2 สามารถอธิบายรายละเอียดของกราฟได้ดังนี้

1) เวลาที่ใช้ในการรับข้อมูลเข้า Topic ของ Kafka 02.36.52 ถึง 02.39.52 AM

2) รวมเวลาที่ใช้ในการรับเข้ามูลขนาด 1 ล้าน Record เข้า Topic = 3 นาที

3) มีอัตราการรับของข้อมูลสูงสุดอยู่ที่ 6.1 MiB/Sec หรือ 60,230.5 Message/Sec

ภาพด้านล่างแสดงกราฟการส่งข้อมูลออกจาก Topic ของ Kafka



รูป 4.3 กราฟแสดงในส่วนของ Kafka Consumer

จากรูป 4.3 สามารถอธิบายรายละเอียดของกราฟได้ดังนี้

1) เวลาที่ใช้ในการส่งข้อมูลจาก Topic ของ Kafka 02.36.52 AM ถึง 02.40.52 AM

2) รวมเวลาที่ใช้ในการส่งข้อมูลขนาด 1 ล้าน Record จาก Topic = 4 นาที

3) มีอัตราการส่งของข้อมูลสูงสุดอยู่ที่ 18.2 MiB/Sec

จากการทดลองแสดงให้เห็นว่า Kafka สามารถรับส่งข้อมูลขนาด 400,000 Record/นาที ได้

4.1.2 การทดสอบการดึงข้อมูลแบบ Real Time โดยใช้ Spark streaming จาก Kafka Topic เพื่อไปใส่ตารางใน Hive โดยมีการจัดเก็บเป็น Partition แบ่งตามวันที่ของข้อมูล

4.1.2.1 วัตถุประสงค์

เพื่อทดสอบ Pipeline การนำเข้าและการจัดเก็บข้อมูลแบบ Real Time ว่าสามารถทนต่อการนำเข้าข้อมูลขนาดใหญ่ได้หรือไม่

4.1.2.2 วิธีการทดลอง

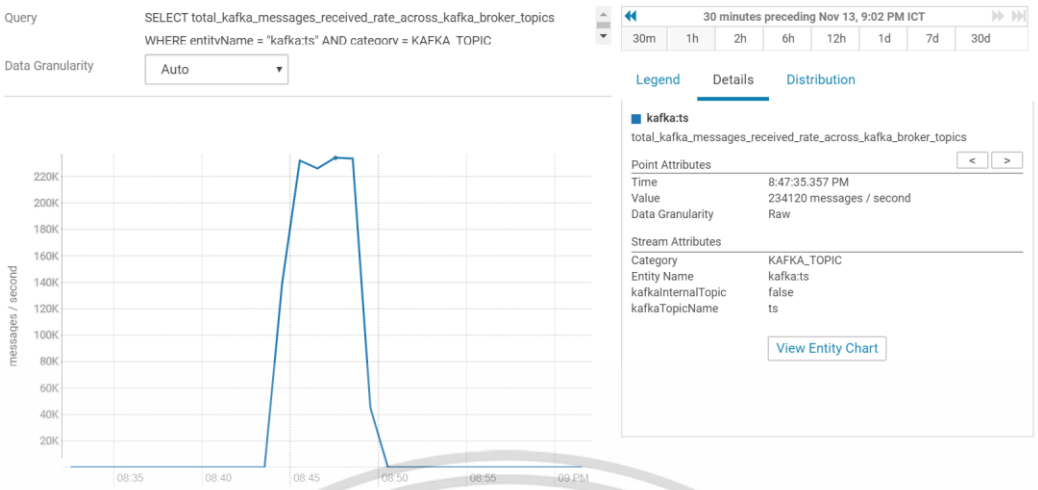
ทดลองส่งไฟล์ข้อมูล 1 วัน ของ gps_log เข้าไปยังระบบจากนั้นตรวจสอบการทำงานและเวลาที่ใช้ในการ Execute ของ Spark streaming จากนั้นตรวจสอบข้อมูลที่เพิ่มเข้าไปยัง Hive และขนาดของ Disk ที่ใช้ไปเปรียบเทียบกับขนาดของไฟล์ก่อนมีการนำเข้าขนาดของไฟล์

4.1.2.3 ผลการทดลอง

```
time_stamp,unit_id,lat,lon,speed,unit_type
2018-05-09 00:00:40,01400                    54,13.80865,100.7402,0,3
2018-05-09 00:00:01,06600                    50,12.653718,101.51767,0,7
2018-05-09 00:00:40,02600                    33,14.4815232,100.1185088,0,
2018-05-09 00:00:58,02500                    52,13.737915,100.760033,0,6
2018-05-09 00:00:04,00700                    11,13.578387,100.804325,0,7
2018-05-09 00:00:38,00700                    78,13.582503,100.800918,0,7
2018-05-09 00:00:23,02500                    52,15.80088,102.03224,0,8
2018-05-09 00:00:58,02200                    39,7.770032,99.33768,0,8
2018-05-09 00:00:44,03100                    93,14.575751,100.995368,0,6
2018-05-09 00:00:36,02500                    18,16.12217,103.81722,0,
2018-05-09 00:00:39,02200                    54,13.841612,100.0213,0,8
2018-05-09 00:00:17,02200                    75,14.616288,104.421643,0,8
2018-05-09 00:00:11,02200                    75,13.574323,100.778825,0,8
2018-05-09 00:00:00,02200                    10,13.453493,101.080888,0,
2018-05-09 00:00:47,03100                    53,14.498666,100.37162,0,5
```

รูป 4.4 หน้าตาของข้อมูลที่จะส่งไปยัง Pipeline Ingestion System

ก)



ข)

รูป 4.5 กราฟการรับเข้าข้อมูลเข้า Topic ของ Kafka Producer

ก) กราฟการรับเข้าข้อมูลเข้า Topic ของ Kafka ในหน่วย MiB/Sec

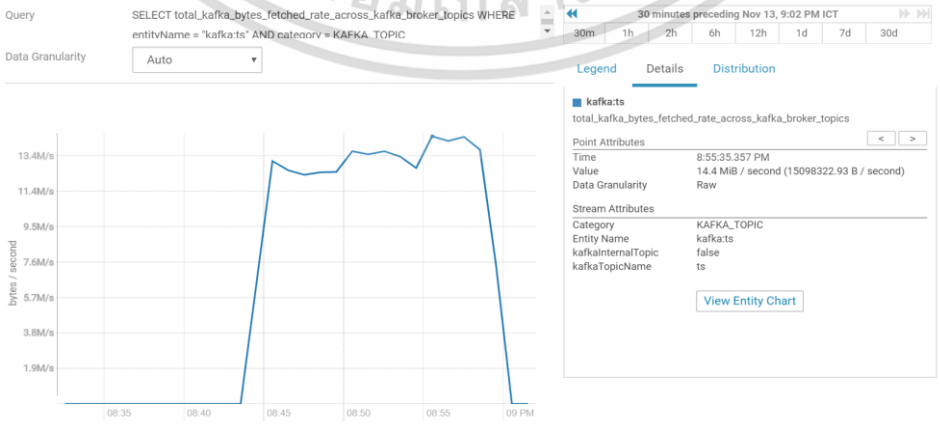
ข) กราฟการรับเข้าข้อมูลเข้า Topic ของ Kafka ในหน่วย Message/Sec

จากรูป 4.5 สามารถอธิบายรายละเอียดของกราฟได้ดังนี้

1) เวลาที่ใช้ในการรับข้อมูลเข้า Topic ของ Kafka 8:43:35 PM ถึง 8:50:35 PM

2) รวมเวลาที่ใช้ในการรับเข้ามูล เข้า Topic = 7 นาที

3) มีอัตราการรับของข้อมูลสูงสุดอยู่ที่ 23.9 MiB/Sec หรือ 234,120 Message/Sec



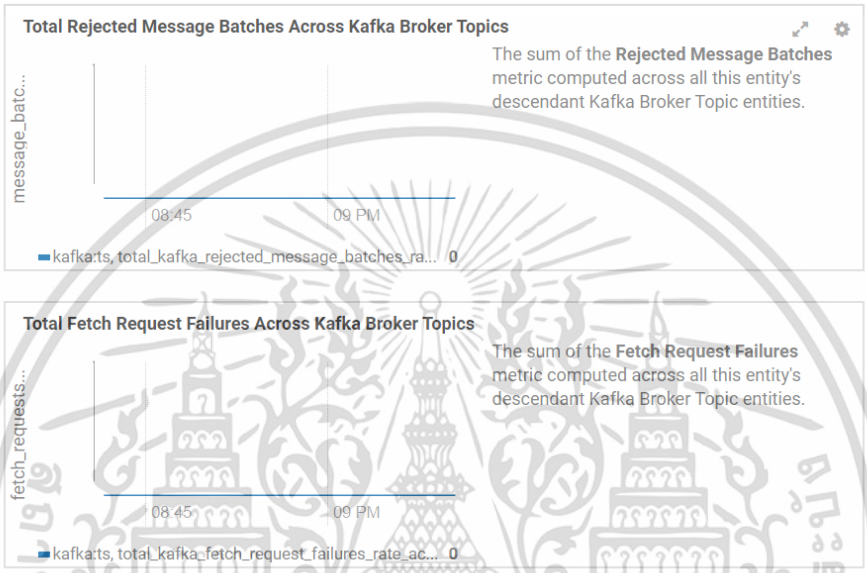
รูป 4.6 กราฟแสดงในส่วนของ Kafka Consumer

จากรูป 4.6 สามารถอธิบายรายละเอียดของกราฟได้ดังนี้

1) เวลาที่ใช้ในการส่งข้อมูลจาก Topic ของ Kafka 8:43:35 ถึง 9:00:35 AM

2) รวมเวลาที่ใช้ในการส่งข้อมูลขนาด 1 ล้าน Record จาก Topic = 17 นาที

3) มีอัตราการรับของข้อมูลสูงสุดอยู่ที่ 14.4 MiB/Sec



รูป 4.7 กราฟแสดงในส่วนของการสูญเสียข้อมูลระหว่างส่งและการเกิดความเสียหายของ Kafka Topic

จากรูป 4.7 จะเห็นว่าไม่มีการเกิดการสูญเสียข้อมูลและไม่มีการเกิดความเสียหายต่อ Kafka Topic

```
----------------------------------------
(None, u'2018-05-09 00:54:10,00200                    3605,14.639235,98.6856116666667,0,3\r')
(None, u'2018-05-09 00:54:30,03600                    .79,13.121567,100.97355,0,6\r')
(None, u'2018-05-09 00:54:16,00200                    :241,16.2960733333333,102.791763333333,0,\r')
(None, u'2018-05-09 00:54:18,03000                    194,13.602805,100.784187,0,\r')
(None, u'2018-05-09 00:54:26,02300                    039,13.803313,100.156762,0,\r')
(None, u'2018-05-09 00:54:23,00200                    1476,14.1404466666667,100.617811666667,0,7\r')
(None, u'2018-05-09 00:54:24,03000                    967,13.775459,99.933967,0,7\r')
(None, u'2018-05-09 00:54:26,03300                    :39,12.04089,102.297997,0,3\r')
(None, u'2018-05-09 00:54:27,03300                    :90,13.790295,100.475807,0,3\r')
(None, u'2018-05-09 00:54:28,03300                    380,7.06058,100.378204,0,\r')
...
```

รูป 4.8 ข้อมูลที่รับมาจาก Kafka Topic

```
[u'2018-05-09 00:13:35', u'02100                7106014', u'13.69295', u'101.70849', u'0', u'3\r']
[u'2018-05-09 00:13:36', u'06700                0004813', u'12.901669', u'100.934877', u'0', u'3\r']
[u'2018-05-09 00:13:43', u'02800                0804243', u'13.993338', u'100.646842', u'0', u'1\r']
[u'2018-05-09 00:13:36', u'06600                1841641', u'12.782521', u'101.640572', u'0', u'7\r']
[u'2018-05-09 00:13:36', u'06600                1836456', u'12.236138', u'102.508827', u'0', u'3\r']
[u'2018-05-09 00:13:37', u'02800                0400068', u'13.692917', u'101.89009', u'0', u'3\r']
[u'2018-05-09 00:13:37', u'00100                0000009', u'14.232127', u'100.82335', u'60', u'7\r']
[u'2018-05-09 00:13:42', u'02100                4032686', u'17.518032', u'104.70482', u'0', u'7\r']
[u'2018-05-09 00:13:41', u'02100                1935337', u'13.733061', u'100.766945', u'0', u'6\r']
[u'2018-05-09 00:13:33', u'07400                5297304', u'13.597935', u'100.72754', u'45', u'6\r']
```

รูป 4.9 ข้อมูลที่ถูกแปลงจาก String เป็น Array

```
Row(lat=u'2018-05-09 00:11:12', lon=u'02500010^^^^^^^^^^^^^^^3395365', speed=u'14.698563', time_stamp=u'102.80729', unit_id=u'0', unit_type='8\r')
Row(lat=u'2018-05-09 00:11:12', lon=u'02500020           8305758', speed=u'15.75156', time_stamp=u'99.789801', unit_id=u'0', unit_type='1\r')
Row(lat=u'2018-05-09 00:11:12', lon=u'03300010           4358857', speed=u'13.72794', time_stamp=u'101.392509', unit_id=u'0', unit_type='8\r')
Row(lat=u'2018-05-09 00:11:02', lon=u'04800000           00043248', speed=u'14.643893', time_stamp=u'102.681653', unit_id=u'90', unit_type='\r')
Row(lat=u'2018-05-09 00:11:12', lon=u'02500020           4634110', speed=u'15.137885', time_stamp=u'105.464596', unit_id=u'0', unit_type='6\r')
Row(lat=u'2018-05-09 00:11:11', lon=u'02500030           1244741', speed=u'19.958751', time_stamp=u'99.84158', unit_id=u'0', unit_type='8\r')
Row(lat=u'2018-05-09 00:11:13', lon=u'00400010           3305238', speed=u'14.1224499', time_stamp=u'100.634956', unit_id=u'0', unit_type='8\r')
Row(lat=u'2018-05-09 00:11:14', lon=u'01300020           8680612', speed=u'14.349576', time_stamp=u'100.69679', unit_id=u'33', unit_type='7\r')
Row(lat=u'2018-05-09 00:11:04', lon=u'0790009G[illegible]1067994', speed=u'16.460371', time_stamp=u'102.833271', unit_id=u'0', unit_type='\r')
```

รูป 4.10 ข้อมูลที่ทำเป็น RDD ก่อนจะไปสร้างเป็น Data Frame

time_stamp	unit_id	lat	lon	speed	unit_type	date
2018-05-09	00:10:26 | 004(	)CA1...	13.3172102	100.967873	0	8
2018-05-09	00:10:32 | 067(	5383...	8.030374	98.864019	0	3
2018-05-09	00:10:12 | 003(	5378...	16.499165	99.564072	0	7
2018-05-09	00:10:42 | 026(	5578...	17.618745	100.091665	0	
2018-05-09	00:10:13 | 003(	5378...	14.681532	100.132358	0	8
2018-05-09	00:10:10 | 015(	)003...	12.168725	99.843781	63	7
2018-05-09	00:10:29 | 004(	)DA1...	13.6746302	100.594307	0	7
2018-05-09	00:10:14 | 067(	5450...	7.991309	98.373596	0	3
2018-05-09	00:10:12 | 001(	)010...	8.762975	99.32835	75	8
2018-05-09	00:10:22 | 013(	5131...	7.920709	98.384698	0	3
2018-05-09	00:10:27 | 080(	)000...	13.6776	100.5908	0	7
2018-05-09	00:10:25 | 025(	5910...	14.731015	100.295671	0	8
2018-05-09	00:10:15 | 003(	5378...	8.57283	99.32783	0	7
2018-05-09	00:10:01 | 013(	5131...	14.070428	99.750603	0	8
2018-05-09	00:10:48 | 029(	3-00...	13.836128	99.846115	0	7
2018-05-09	00:10:06 | 001(	)010...	15.54259	103.047066	68	7
2018-05-09	00:10:15 | 025(	5985...	15.87397	100.30489	0	7
2018-05-09	00:10:13 | 069(	)000...	13.750598	100.600035	0	3
2018-05-09	00:10:26 | 067(	5910...	7.896538	98.336573	0	3
2018-05-09	00:10:17 | 067(	5383...	13.764227	100.624298	0	

รูป 4.11 Data Frame ที่ได้ก่อนไปเพิ่งลงตารางใน Hive

date=2019-02-01	root	hive	drwxrwxrwxt	November 11, 2019 10:30 AM
date=2019-02-02	root	hive	drwxrwxrwxt	November 11, 2019 10:45 AM
date=2019-02-03	root	hive	drwxrwxrwxt	November 11, 2019 11:02 AM
date=2019-02-04	root	hive	drwxrwxrwxt	November 11, 2019 11:17 AM
date=2019-02-05	root	hive	drwxrwxrwxt	November 11, 2019 02:24 PM
date=2019-02-06	root	hive	drwxrwxrwxt	November 11, 2019 02:38 PM
date=2019-02-07	root	hive	drwxrwxrwxt	November 11, 2019 02:52 PM
date=2019-02-08	root	hive	drwxrwxrwxt	November 11, 2019 03:06 PM
date=2019-02-09	root	hive	drwxrwxrwxt	November 11, 2019 03:19 PM
date=2019-02-10	root	hive	drwxrwxrwxt	November 12, 2019 08:40 AM
date=2019-02-11	root	hive	drwxrwxrwxt	November 12, 2019 08:55 AM

รูป 4.12 ไฟล์ข้อมูลบน HDFS ใน Hive ที่เก็บข้อมูลเป็น Partition แบ่งตามวันที่

	gpslog.time_stamp	gpslog.unit_id	gpslog.lat	gpslog.lon	gpslog.speed	gpslog.unit_type	gpslog.date
1	2018-05-09 00:00:00.0	00500)000ID00001039	13.355199813842773	101.00641632080078	0	7	2018-05-09
2	2018-05-09 00:00:00.0	00500)000M300001477	13.601249694824219	100.57134246826172	0	7	2018-05-09
3	2018-05-09 00:00:00.0	04800)0000000002546	11.276430130004883	99.4395523071289	0	7	2018-05-09
4	2018-05-09 00:00:00.0	00500)000ID00001039	13.355199813842773	101.00641632080078	0	7	2018-05-09
5	2018-05-09 00:00:00.0	00500)000M300001722	13.1127700805664062	100.9419174194336	2	7	2018-05-09
6	2018-05-09 00:00:00.0	03900)000000EE02441	17.882722854614258	102.70954132080078	0	7	2018-05-09
7	2018-05-09 00:00:00.0	00800)0001802218017	13.720199048217773	100.80460357666016	0	7	2018-05-09
8	2018-05-09 00:00:00.0	00500)000ID00011373	18.744829177856445	98.97734069824219	0	7	2018-05-09

รูป 4.13 ตัวอย่างข้อมูลที่เพิ่มลงใน Hive

```
root@nmtic-5:/home/nmtic# du -h -s /home/nmtic/2018-05
4.6G    /home/nmtic/2018-05
```

ก)

```
root@nmtic-5:/home/nmtic# hadoop fs -du -h -s /user/hive/warehouse/gpslog/date=2018-05-09
833.3 M  2.4 G  /user/hive/warehouse/gpslog/date=2018-05-09
```

ข)

รูป 4.14 ขนาดของไฟล์

ก) ขนาดของไฟล์ก่อนนำเข้าระบบ

ข) ขนาดของไฟล์หลังนำเข้าระบบ

จากรูป 4.14 จะเห็นได้ว่าหลังนำเข้าระบบมีพื้นที่ที่ใช้ในการจัดเก็บทั้งหมดมีขนาด 2.4 GB มีการแบ่งเก็บไปที่ Datanode โหนดละ 833.3 M ซึ่งจากเดิมข้อมูลก่อนนำเข้ามาในระบบมีขนาด 4.6 GB แสดงว่ามีการลดขนาดของข้อมูลไป 2.2 GB หรือ 47.82% จากเดิม

จากการทดลองแสดงให้เห็นว่าการออกแบบระบบนำเข้าข้อมูลสามารถใช้งานได้จริงในการนำข้อมูลเข้าแบบ Real-Time ขนาด 400,000 Record/นาที และการออกแบบการจัดเก็บข้อมูลใน hive ยังทำให้สามารถลดพื้นที่ในการจัดเก็บซึ่งเป็นการประหยัดทรัพยากรได้

4.2 การทดลองการวิเคราะห์ข้อมูล

4.2.1 การทดสอบการวิเคราะห์ข้อมูลเพื่อหาความเร็วเฉลี่ยของรถยนต์ทั้งหมดใน 1 วัน

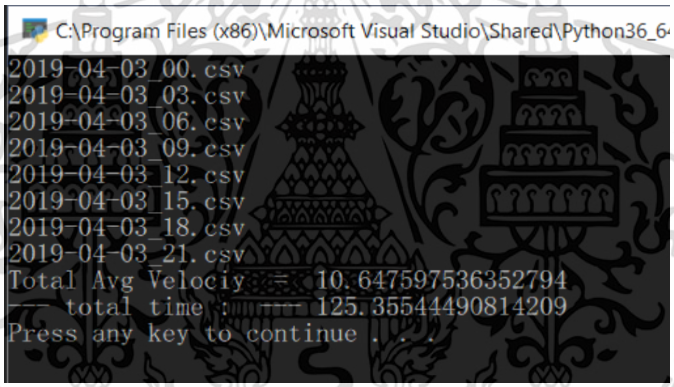
4.2.1.1 วัตถุประสงค์

เพื่อเปรียบเทียบความเร็วในการประมวลผลเมื่อเทียบกับการประมวลผลโดยใช้โปรแกรมภาษา Python

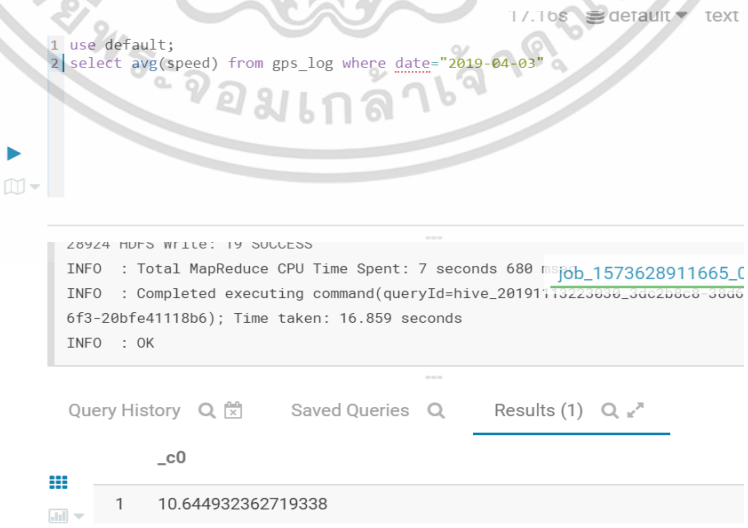
4.2.1.2 วิธีการทดลอง

ทำการทดลองกับข้อมูลชุดเดียวกันโดยการทดสอบหาความเร็วเฉลี่ยโดยการใช้ Python และ หาความเร็วเฉลี่ยโดยการ Query ข้อมูลจาก Hive จากนั้นบันทึกเวลาที่ได้

4.2.1.3 ผลการทดลอง



รูป 4.15 ผลการหาความเร็วเฉลี่ยโดยใช้โปรแกรมภาษา Python



รูป 4.16 ผลการหาความเร็วเฉลี่ยโดยการ Query บน Hive

โดยจะเห็นได้ว่าการใช้โปรแกรม Python ในการหาความเร็วเฉลี่ยของข้อมูล 1 วัน ใช้เวลาในการประมวลผล 125.355 วินาที หรือ 2.09 นาที ส่วนการ Query ข้อมูลบน Hive ในการหาความเร็วเฉลี่ยของข้อมูล 1 วัน ใช้เวลาในการประมวลผล 16.859 วินาที ซึ่งเร็วกว่าการใช้โปรแกรมภาษา Python 1.8 นาที คิดเป็น 86.55%

4.2.2 การทดสอบการวิเคราะห์ข้อมูลเพื่อหาจำนวนรถยนต์ในแต่ละประเภทของปี 2561-2562

4.2.2.1 วัตถุประสงค์

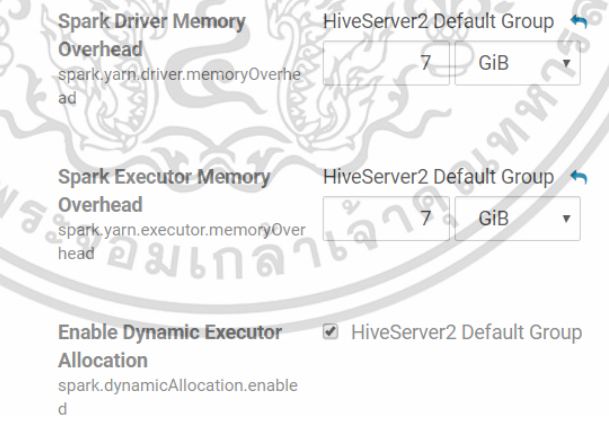
เพื่อหาจำนวนรถยนต์ในแต่ละประเภทของปี 2561-2562 โดยใช้ PySpark

4.2.2.2 วิธีการทดลอง

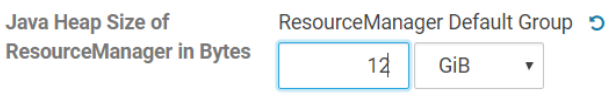
ทำการ Config ค่าต่างๆของ Hive และ Yarn Service จากนั้นสร้างไฟล์โปรแกรม Pyspark เพื่อคำนวณหาจำนวนรถยนต์ในแต่ละประเภทของปี 2561-2562 จากข้อมูลทั้งหมดที่ได้ทำการนำเข้าไว้ก่อนหน้านั้น จากนั้นตรวจสอบผลการรันว่าระบบสามารถคำนวณได้โดยสำเร็จหรือไม่ พร้อมบันทึกผลและเวลาที่ใช้

4.2.2.3 ผลการทดลอง

ภาพด้านล่างแสดงรายละเอียดการ Config ค่าต่างๆบน Hive และ Yarn Service



ก)



ข)

รูป 4.17 รายละเอียดการ Config ค่าต่างๆ บน Cloudera Manager

ก) รายละเอียดการConfig ค่าบน Hive Service

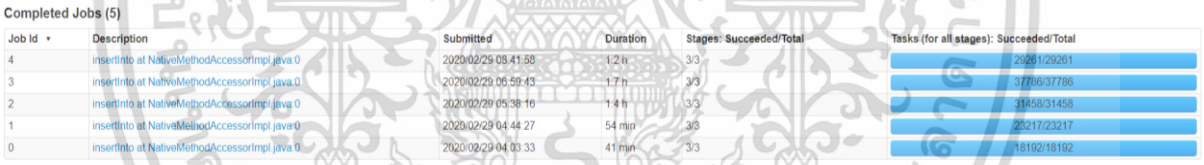
ข) รายละเอียดการConfig ค่าบน Yarn Service

คำสั่งข้างล่างเป็นการกำหนดให้โปรแกรมทำงานบน Driver memory ที่ 3GB

โปรแกรม 4.1 คำสั่งที่ใช้ในการรันไฟล์ numcartype.py

```
/usr/bin/spark2-submit --master yarn-client --driver-memory
3g  numcartype.py
```

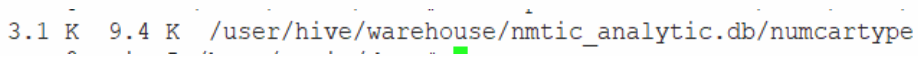
จากการทดลองพบว่าการ Config ค่า spark.driver memory = 3 GB, spark.executor.memory = 1 GB (Default), spark.driver.memoryOverhead = 7 GB , spark.executor.memoryOverhead = 7 GB , yarn.nodemanager.resource.memory = 12 GB ทำให้โปรแกรมรันได้สำเร็จ


Completed Jobs (5)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	insertInto at NativeMethodAccessorImpl.java:0	2020/02/29 08:41:58	1.2 h	3/3	29261/29261
3	insertInto at NativeMethodAccessorImpl.java:0	2020/02/29 06:59:43	1.7 h	3/3	37786/37786
2	insertInto at NativeMethodAccessorImpl.java:0	2020/02/29 05:38:16	1.4 h	3/3	31458/31458
1	insertInto at NativeMethodAccessorImpl.java:0	2020/02/29 04:44:27	54 min	3/3	23217/23217
0	insertInto at NativeMethodAccessorImpl.java:0	2020/02/29 04:03:33	41 min	3/3	18192/18192

รูป 4.18 ผลการทดลองของข้อมูลเมื่อรันผ่านไป 10 เดือน

จากผลการทดลองเมื่อรันผ่านไป 10 เดือน จะใช้เวลาเฉลี่ยประมาณ 1 ชั่วโมง 5 นาที / 2 เดือน



รูป 4.19 ขนาดของไฟล์ตาราง numcartype บน HDFS

จากรูป 4.19 ไฟล์ตาราง numcartype บน HDFS มีขนาดทั้งหมด 9.4 KB โดยแบ่งเก็บที่แต่ละ Datanode ขนาด 3.1 KB

	numcartype.year	numcartype.month	numcartype.count_unit_type	numcartype.unit_type
	2018	1	40067	0
2	2018	1	18101	1
3	2018	1	44229	3
4	2018	1	160	4
5	2018	1	2799	5

รูป 4.20 ตัวอย่างข้อมูล numcartype ที่ได้

จากการทดลองแสดงให้เห็นว่าสามารถวิเคราะห์ข้อมูลที่นำเข้ามาในระบบก่อนหน้านี้เพื่อหาจำนวนรถยนต์ในแต่ละปีได้สำเร็จ

4.2.3 การทดสอบการวิเคราะห์ข้อมูลเพื่อหาความเร็วเฉลี่ย และจำนวนการกระทำผิดความผิด(จากความเร็ว)ของรถต่อคัน ต่อเดือน แยกตามรายประเภทรถยนต์ ปี 2561 -2562

4.2.3.1 วัตถุประสงค์

เพื่อหาความเร็วเฉลี่ย และจำนวนการกระทำผิดความผิด(จากความเร็ว)ของรถต่อคันต่อเดือน แยกตามรายประเภทรถยนต์ ปี 2561 -2562 โดยใช้ PySpark โดยการกระทำผิดจากความเร็วคือการที่รถมีความเร็วเกิน 90 กม/ชม. นานติดต่อกันเป็นเวลา 3 ชั่วโมง

4.2.3.2 วิธีการทดลอง

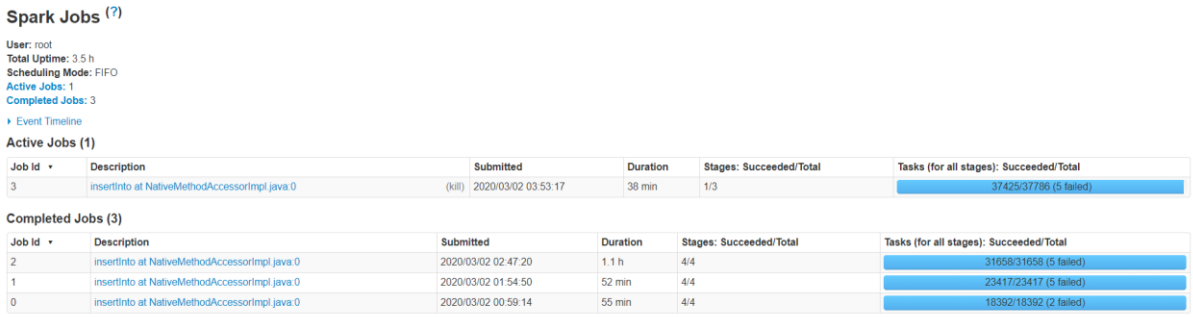
สร้างไฟล์โปรแกรม Pyspark เพื่อคำนวณหาระยะเวลาที่มากที่สุดของช่วงเวลาที่ความเร็วเกิน 90 กม/ชมเก็บไว้ในตารางชื่อ overspeed จากนั้นสร้างอีกไฟล์โปรแกรมเพื่อจำนวนครั้งที่กระทำผิดจากตาราง overspeed เก็บไว้ในตารางชื่อ illegal_by_speed ตรวจสอบผลการรันว่าระบบสามารถคำนวณได้โดยสำเร็จหรือไม่ พร้อมบันทึกผลและเวลาที่ใช้

4.2.3.3 ผลการทดลอง

คำสั่งข้างล่างเป็นการกำหนดให้โปรแกรมทำงานบน Driver memory ที่ 3GB

โปรแกรม 4.2 คำสั่งที่ใช้ในการรันไฟล์ overspeed.py

```
/usr/bin/spark2-submit   --master yarn-client   --driver-
memory 3g  overspeed.py
```



Spark Jobs (?)

User: root
Total Uptime: 3.5 h
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 3

▸ Event Timeline

Active Jobs (1)

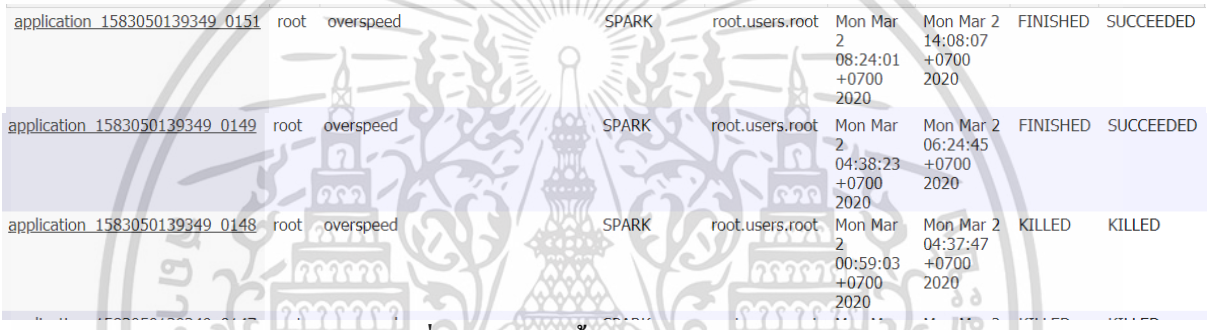
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	insertInto at NativeMethodAccessorImpl.java:0 (kill)	2020/03/02 03:53:17	38 min	1/3	37425/37786 (5 failed)

Completed Jobs (3)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	insertInto at NativeMethodAccessorImpl.java:0	2020/03/02 02:47:20	1.1 h	4/4	31658/31658 (5 failed)
1	insertInto at NativeMethodAccessorImpl.java:0	2020/03/02 01:54:50	52 min	4/4	23417/23417 (5 failed)
0	insertInto at NativeMethodAccessorImpl.java:0	2020/03/02 00:59:14	55 min	4/4	18392/18392 (2 failed)

รูป 4.21 ผลการทดลองเพื่อหาข้อมูลในตาราง overspeed เมื่อรันผ่านไป 6 เดือน

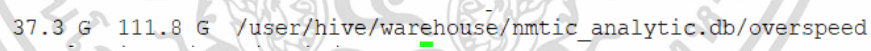
จากผลการทดลองเมื่อรันผ่านไป 6 เดือน จะใช้เวลาเฉลี่ยประมาณ 59 นาที /2 เดือน



application_1583050139349_0151	root	overspeed	SPARK	root.users.root	Mon Mar 2 08:24:01 +0700 2020	Mon Mar 2 14:08:07 +0700 2020	FINISHED	SUCCEEDED
application_1583050139349_0149	root	overspeed	SPARK	root.users.root	Mon Mar 2 04:38:23 +0700 2020	Mon Mar 2 06:24:45 +0700 2020	FINISHED	SUCCEEDED
application_1583050139349_0148	root	overspeed	SPARK	root.users.root	Mon Mar 2 00:59:03 +0700 2020	Mon Mar 2 04:37:47 +0700 2020	KILLED	KILLED

รูป 4.22 เวลาที่ใช้ในการรันทั้งหมดของตาราง overspeed

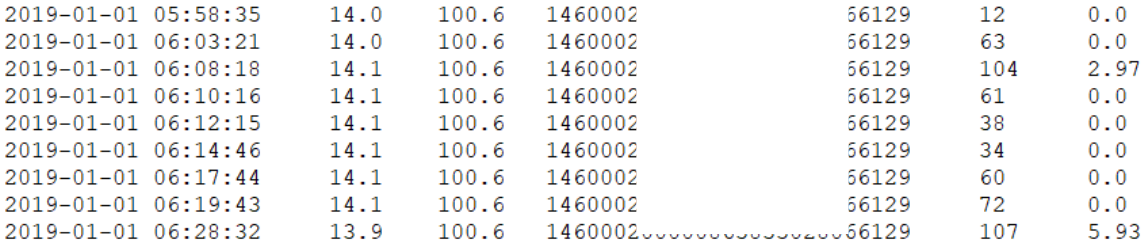
จากผลการทดลองใช้เวลาในการรันทั้งหมด 11 ชั่วโมง 8 นาที



```
37.3 G  111.8 G  /user/hive/warehouse/nmtic_analytic.db/overspeed
```

รูป 4.23 ขนาดของไฟล์ตาราง overspeed บน HDFS

จากรูป 4.23 ไฟล์ตาราง overspeed บน HDFS มีขนาดทั้งหมด 111.8 GB โดยแบ่งเก็บที่แต่ละ Datanode ขนาด 37.3 KB



```
2019-01-01 05:58:35     14.0    100.6   1460002                 56129     12      0.0
2019-01-01 06:03:21     14.0    100.6   1460002                 56129     63      0.0
2019-01-01 06:08:18     14.1    100.6   1460002                 56129     104     2.97
2019-01-01 06:10:16     14.1    100.6   1460002                 56129     61      0.0
2019-01-01 06:12:15     14.1    100.6   1460002                 56129     38      0.0
2019-01-01 06:14:46     14.1    100.6   1460002                 56129     34      0.0
2019-01-01 06:17:44     14.1    100.6   1460002                 56129     60      0.0
2019-01-01 06:19:43     14.1    100.6   1460002                 56129     72      0.0
2019-01-01 06:28:32     13.9    100.6   1460002                 56129     107     5.93
```

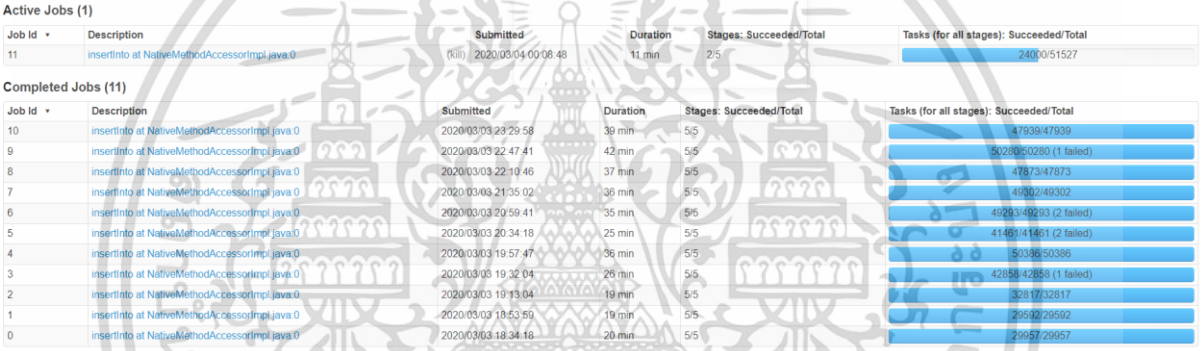
รูป 4.24 ตัวอย่างข้อมูลในตาราง overspeed

จะเห็นได้ว่าสามารถวิเคราะห์เพื่อหาระยะเวลาที่รถมีความเร็วเกิน 90กม/ชม. ติดต่อกันได้สำเร็จ โดยจะแสดงเวลาที่มากที่สุดที่มีหน่วยเป็นนาทีของช่วงเวลาการกระทำผิดนั้น ๆ บนตาราง overspeed เพื่อนำไปใช้ในการหาจำนวนครั้งในการกระทำความผิดต่อไป

คำสั่งข้างล่างเป็นการกำหนดให้โปรแกรมทำงานบน Driver memory ที่ 4GB

โปรแกรม 4.3 คำสั่งที่ใช้ในการรันไฟล์ illegal_by_speed.py

```
/usr/bin/spark2-submit   --master yarn-client   --driver-
memory 4g  illegal_by_speed.py
```



Active Jobs (1)

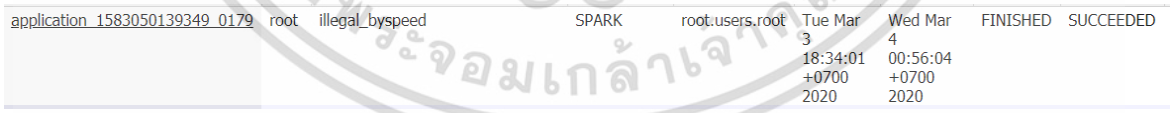
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
11	insertInto at NativeMethodAccessorImpl.java:0	(kill) 2020/03/04 00:08:48	11 min	2/5	24000/51527

Completed Jobs (11)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
10	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 23:29:58	39 min	5/5	47939/47939
9	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 22:47:41	42 min	5/5	50280/50280 (1 failed)
8	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 22:10:46	37 min	5/5	47873/47873
7	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 21:35:02	36 min	5/5	49302/49302
6	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 20:59:41	35 min	5/5	49293/49293 (2 failed)
5	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 20:34:18	25 min	5/5	41461/41461 (2 failed)
4	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 19:57:47	36 min	5/5	50386/50386
3	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 19:32:04	26 min	5/5	42858/42858 (1 failed)
2	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 19:13:04	19 min	5/5	32817/32817
1	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 18:53:59	19 min	5/5	29592/29592
0	insertInto at NativeMethodAccessorImpl.java:0	2020/03/03 18:34:18	20 min	5/5	29957/29957

รูป 4.25 ผลการทดลองเพื่อหาข้อมูลในตาราง illegal_by_speed เมื่อรันผ่านไป 22 เดือน

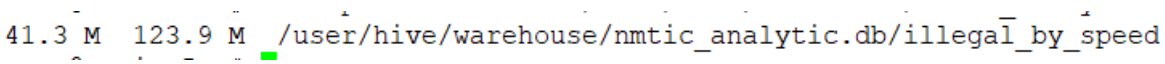
จากผลการทดลองเมื่อรันผ่านไป 22 เดือนจะใช้เวลาเฉลี่ยประมาณ 30.36 นาที /2 เดือน



application_1583050139349_0179	root	illegal_byspeed	SPARK	root.users.root	Tue Mar 3 18:34:01 +0700 2020	Wed Mar 4 00:56:04 +0700 2020	FINISHED	SUCCEEDED

รูป 4.26 เวลาที่ใช้ในการรันทั้งหมดของตาราง illegal_by_speed

จากผลการทดลองใช้เวลาในการรันทั้งหมด 1 วัน 6 ชั่วโมง 22 นาที



```
41.3 M  123.9 M  /user/hive/warehouse/nmtic_analytic.db/illegal_by_speed
```

รูป 4.27 ขนาดของไฟล์ตาราง illegal_by_speed บน HDFS

จากรูป 4.27 ไฟล์ตาราง illegal_by_speed บน HDFS มีขนาดทั้งหมด 123.9 GB โดยแบ่งเก็บที่ แต่ละ Datanode ขนาด 41.3 KB

	illegal_by_speed.year	illegal_by_speed.month	illegal_by_speed.unit_id	illegal_by_speed.avg_speed	illegal_by_speed.cnt_illegal	illegal_by_speed.unit_typ
1	2018	1	0010000000000000109050125326	79.9800033569336	35	1
2	2019	1	0010000000000000113590103629	62.470001220703125	2	0
3	2019	1	0010000000000000113590110223	64.62999725341797	4	0
4	2018	1	0010000000000000122190103625	42.060001373291016	2	0
5	2019	1	0010000000000000124770201933	46.29999923706055	3	0
6	2018	1	0010000000000000156010102218	55.040000915527344	2	0

รูป 4.28 ตัวอย่างข้อมูล illegal_by_speed ที่ได้

จากการทดลองแสดงให้เห็นว่าสามารถวิเคราะห์ข้อมูลที่นำเข้ามาในระบบก่อนหน้านี้เพื่อหาจำนวนรถยนต์ที่กระทำความผิดจากการความเร็วในแต่ละปีได้

4.2.4 การทดสอบการวิเคราะห์ข้อมูลเพื่อหาการกระทำความผิด(จากชั่วโมงการทำงาน) แยกตามชนิดรถ รายเดือนของปี 2561-2562

4.2.4.1 วัตถุประสงค์

เพื่อหาการกระทำความผิดจากชั่วโมงการทำงาน แยกตามชนิดรถ รายเดือนของปี 2561-2562 โดยใช้ PySpark โดยการกระทำความผิดจากชั่วโมงการทำงานหมายถึง การที่รถวิ่งนานติดต่อกันเกิน 4 ชั่วโมงหรือเมื่อครบ 4 ชั่วโมงแต่มีการพักรถไม่ถึง 2 ชั่วโมง

4.2.4.2 วิธีการทดลอง

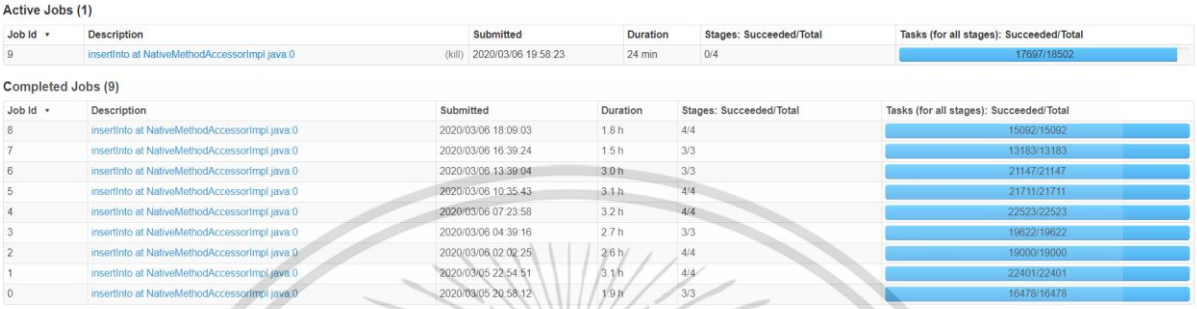
สร้างไฟล์โปรแกรม Pyspark เพื่อคำนวณหาระยะเวลาที่มากที่สุดของช่วงเวลาที่รถมีการทำงานและการพักรถของช่วงเวลาถัดไป เก็บไว้ในตารางชื่อ overtime จากนั้นสร้างอีกไฟล์โปรแกรมเพื่อจำนวนครั้งที่กระทำผิดจากตาราง overtime เก็บไว้ในตารางชื่อ illegal_by_time ตรวจสอบผลการรันว่าระบบสามารถคำนวณได้โดยสำเร็จหรือไม่ พร้อมบันทึกผลและเวลาที่ใช้

4.2.4.3 ผลการทดลอง

คำสั่งข้างล่างเป็นการกำหนดให้โปรแกรมทำงานบน Driver memory ที่ 3GB

โปรแกรม 4.5 คำสั่งที่ใช้ในการรันไฟล์ overtime.py

```
/usr/bin/spark2-submit   --master yarn-client   --driver-
memory 5g  overtime.py
```



Active Jobs (1)

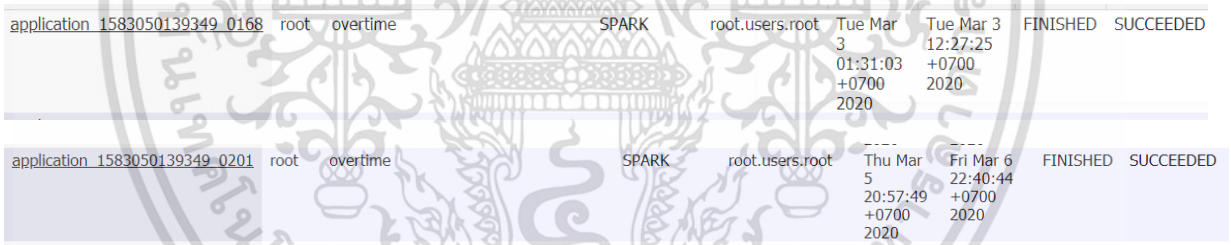
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
9	insertInto at NativeMethodAccessorImpl.java:0 (kill)	2020/03/06 19:58:23	24 min	0/4	17697/18502

Completed Jobs (9)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
8	insertInto at NativeMethodAccessorImpl.java:0	2020/03/06 18:09:03	1.8 h	4/4	15092/15092
7	insertInto at NativeMethodAccessorImpl.java:0	2020/03/06 16:39:24	1.5 h	3/3	13183/13183
6	insertInto at NativeMethodAccessorImpl.java:0	2020/03/06 13:39:04	3.0 h	3/3	21147/21147
5	insertInto at NativeMethodAccessorImpl.java:0	2020/03/06 10:35:43	3.1 h	4/4	21711/21711
4	insertInto at NativeMethodAccessorImpl.java:0	2020/03/06 07:23:58	3.2 h	4/4	22523/22523
3	insertInto at NativeMethodAccessorImpl.java:0	2020/03/06 04:39:16	2.7 h	3/3	19622/19622
2	insertInto at NativeMethodAccessorImpl.java:0	2020/03/06 02:02:25	2.6 h	4/4	19000/19000
1	insertInto at NativeMethodAccessorImpl.java:0	2020/03/05 22:54:51	3.1 h	4/4	22401/22401
0	insertInto at NativeMethodAccessorImpl.java:0	2020/03/05 20:58:12	1.9 h	3/3	16478/16478

รูป 4.29 ผลการทดลองเพื่อหาข้อมูลในตาราง nmtic_analytic.overtime เมื่อรันผ่านไป 9 เดือน

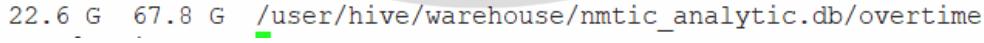
จากผลการทดลองเมื่อรันผ่านไป 9 เดือน จะใช้เวลาเฉลี่ยประมาณ 2 ชั่วโมง 5 นาที / เดือน



application_1583050139349_0168	root	overtime	SPARK	root.users.root	Tue Mar 3 01:31:03 +0700 2020	Tue Mar 3 12:27:25 +0700 2020	FINISHED	SUCCEEDED
application_1583050139349_0201	root	overtime	SPARK	root.users.root	Thu Mar 5 20:57:49 +0700 2020	Fri Mar 6 22:40:44 +0700 2020	FINISHED	SUCCEEDED

รูป 4.30 เวลาที่ใช้ในการรันทั้งหมดของตาราง overtime

จากผลการทดลองใช้เวลาในการรันทั้งหมด 1 วัน 11 ชั่วโมง 39 นาที



```
22.6 G  67.8 G  /user/hive/warehouse/nmtic_analytic.db/overtime
```

รูป 4.31 ขนาดของไฟล์ตาราง overtime บน HDFS

จากรูป 4.31 ไฟล์ตาราง overtime บน HDFS มีขนาดทั้งหมด 67.8 GB โดยแบ่งเก็บที่ แต่ละ Datanode ขนาด 22.6 KB

2019-01-01 22:30:27	13.7	100.7	1460	)66129	96	10.87	0.0
2019-01-01 22:32:44	13.7	100.6	1460	)66129	0	0.0	0.0
2019-01-01 22:43:05	13.7	100.6	1460	)66129	37	839.2	0.0
2019-01-01 22:45:03	13.7	100.6	1460	)66129	0	0.0	0.0
2019-01-01 22:50:27	13.7	100.6	1460	)66129	32	2.95	0.0
2019-01-01 22:51:40	13.7	100.6	1460	)66129	0	0.0	0.0
2019-01-01 22:53:36	13.7	100.6	1460	)66129	37	0.0	14.2
2019-01-01 23:09:40	13.8	100.6	1460	)66129	0	0.0	0.0
2019-01-01 23:31:47	13.8	100.5	1460	)66129	73	21.13	9.07
2019-01-01 23:42:49	13.8	100.5	1460	)66129	0	0.0	0.0

รูป 4.32 ตัวอย่างข้อมูลในตาราง overtime

จะเห็นได้ว่าสามารถวิเคราะห์เพื่อหาระยะเวลาที่รถมีการวิ่งและมีการพักได้ โดยจะแสดงเวลาที่มากที่สุดที่มีหน่วยเป็นนาทีของช่วงเวลาที่รถมีการวิ่งและเวลาที่มากที่สุดที่รถมีความเร็วเป็น 0 ของช่วงเวลาถัดไปบนตาราง overtime เพื่อนำไปใช้ในการหาจำนวนครั้งในการกระทำความผิดต่อไป

คำสั่งข้างล่างเป็นการกำหนดให้โปรแกรมทำงานบน Driver memory ที่ 3GB

โปรแกรม 4.6 คำสั่งที่ใช้ในการรันไฟล์ illegal_by_time.py

```
/usr/bin/spark2-submit   --master yarn-client   --driver-
memory 3g  illegal_by_time.py
```

Completed Jobs (5)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 19:02:28	5.5 min	2/2	24380/24380
3	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 18:59:18	3.2 min	2/2	11400/11400
2	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 18:56:31	2.8 min	2/2	9600/9600
1	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 18:53:16	3.2 min	2/2	11400/11400
0	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 18:49:30	3.7 min	2/2	12600/12600

รูป 4.33 ผลการทดลองเพื่อหาข้อมูลในตาราง nmtic_analytic.illegal_by_time เมื่อรันผ่านไป 10 เดือน

จากผลการทดลองเมื่อรันผ่านไป 10 เดือน จะใช้เวลาเฉลี่ยประมาณ 4 นาที 8 วินาที / 2 เดือน

application_1583050139349_0230	root	illegal_bytime	SPARK	root.users.root	Sun Mar 8 18:49:20 +0700 2020	Sun Mar 8 19:34:56 +0700 2020	FINISHED	SUCCEEDED

รูป 4.34 เวลาที่ใช้ในการรันทั้งหมดของตาราง illegal_by_time

จากผลการทดลองใช้เวลาในการรันทั้งหมดประมาณ 45 นาที

```
174.2 M  522.7 M  /user/hive/warehouse/nmtic_analytic.db/illegal_by_time
```

รูป 4.35 ขนาดของไฟล์ตาราง illegal_by_time บน HDFS

จากรูป 4.35 ไฟล์ตาราง illegal_by_time บน HDFS มีขนาดทั้งหมด 522.7MB โดยแบ่งเก็บที่ แต่ละ Datanode ขนาด 174.2 MB

	illegal_by_time.year	illegal_by_time.month	illegal_by_time.unit_id	illegal_by_time.cnt_illegal	illegal_by_time.unit_type
1	2019	1	05000	1322	33	8
2	2019	1	01800	255	12	0
3	2019	1	00100	936	19	6
4	2019	1	03900	122	45	8
5	2019	1	00900	009	15	3

รูป 4.36 ตัวอย่างข้อมูลในตาราง nmtic_analytic.illegal_by_time

จากการทดลองแสดงให้เห็นว่าสามารถวิเคราะห์ข้อมูลที่นำเข้ามาในระบบก่อนหน้านี้เพื่อหาจำนวนรถยนต์ที่กระทำความผิดจากเวลาในแต่ละปีได้สำเร็จ

4.2.5 การทดสอบการวิเคราะห์ข้อมูลเพื่อหาการกระทำความผิด (จากความเร็ว) แยกตามพื้นที่

4.2.5.1 วัตถุประสงค์

เพื่อหาการกระทำความผิด (จากความเร็ว) แยกตามพื้นที่ โดยใช้ PySpark

4.2.5.2 วิธีการทดลอง

สร้างไฟล์โปรแกรม Pyspark เพื่อคำนวณหาจำนวนการกระทำความผิดโดยแยกตามพื้นที่โดยใช้ตารางชื่อ locs10km มาใช้ในการหาพื้นที่โดยเทียบกับคอลัม lat,lon ของตาราง gps_log ด้วย ตรวจสอบผลการรันว่าระบบสามารถคำนวณได้โดยสำเร็จหรือไม่ พร้อมบันทึกผลและเวลาที่ใช้

4.2.5.3 ผลการทดลอง

	locs10km.lat	locs10km.lon	locs10km.district	locs10km.province
1	18.5	97.4	Mae Sariang	Mae Hong Son
2	18.3	97.5	Mae Sariang	Mae Hong Son
3	18.4	97.5	Mae Sariang	Mae Hong Son
4	18.4	97.6	Mae Sariang	Mae Hong Son
5	18.5	97.6	Mae Sariang	Mae Hong Son
6	17.9	97.7	Sop Moei	Mae Hong Son
7	18.1	97.7	Mae Sariang	Mae Hong Son
8	18.2	97.7	Mae Sariang	Mae Hong Son
9	18.3	97.7	Mae Sariang	Mae Hong Son

รูป 4.37 ตัวอย่างข้อมูลในตาราง locs10km

คำสั่งข้างล่างเป็นการกำหนดให้โปรแกรมทำงานบน Driver memory ที่ 3GB

โปรแกรม 4.7 คำสั่งที่ใช้ในการรันไฟล์ illegal_speed_by_area.py

```
/usr/bin/spark2-submit   --master yarn-client   --driver-
memory 3g  illegal_speed_by_area.py
```

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
14	run at ThreadPoolExecutor.java:1145	2020/03/08 21:18:39	0.1 s	1/1	2/2
13	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 21:15:47	2.8 min	3/3	12800/12800
12	run at ThreadPoolExecutor.java:1145	2020/03/08 21:15:47	83 ms	1/1	2/2
11	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 21:13:15	2.5 min	3/3	12400/12400
10	run at ThreadPoolExecutor.java:1145	2020/03/08 21:13:14	0.1 s	1/1	2/2
9	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 21:10:26	2.8 min	3/3	12800/12800
8	run at ThreadPoolExecutor.java:1145	2020/03/08 21:10:25	0.2 s	1/1	2/2
7	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 21:07:34	2.8 min	3/3	11600/11600
6	run at ThreadPoolExecutor.java:1145	2020/03/08 21:07:34	0.1 s	1/1	2/2
5	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 21:05:29	2.1 min	3/3	9800/9800
4	run at ThreadPoolExecutor.java:1145	2020/03/08 21:05:28	0.2 s	1/1	2/2
3	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 21:02:59	2.5 min	3/3	11600/11600
2	run at ThreadPoolExecutor.java:1145	2020/03/08 21:02:58	0.2 s	1/1	2/2
1	insertInto at NativeMethodAccessorImpl.java:0	2020/03/08 21:00:02	2.9 min	3/3	12800/12800
0	run at ThreadPoolExecutor.java:1145	2020/03/08 20:59:53	8 s	1/1	2/2

รูป 4.38 ผลการทดลองเพื่อหาข้อมูลในตาราง illegal_speed_by_area เมื่อรันผ่านไป 14 เดือน

จากผลการทดลองเมื่อรันผ่านไป 14 เดือน จะใช้เวลาเฉลี่ยประมาณ 2.8 นาที /2 เดือน

application_1583050139349_0243	root	illegal_bytime	SPARK	root.users.root	Sun Mar 8 20:59:44 +0700 2020	Sun Mar 8 21:32:17 +0700 2020	FINISHED	SUCCEEDED

รูป 4.39 เวลาที่ใช้ในการรันทั้งหมดของตาราง illegal_speed_by_area

จากผลการทดลองใช้เวลาในการรันทั้งหมดประมาณ 33 นาที

```
91.6 K  274.7 K  /user/hive/warehouse/nmtic_analytic.db/illegal_speed_by_area
```

รูป 4.40 ขนาดของไฟล์ตาราง overspeed บน HDFS

จากรูป 4.40 ไฟล์ตาราง illegal_speed_by_area บน HDFS มีขนาดทั้งหมด 274.7 KB โดยแบ่งเก็บที่ แต่ละ Datanode ขนาด 91.6 KB

	illegal_speed_by_area.year	illegal_speed_by_area.month	illegal_speed_by_area.lat	illegal_speed_by_area.lon	illegal_speed_by_area.district	illegal_speed_by_a
1	2019	1	15	100.5	Ban Mi	Lop Buri
2	2018	1	17.5	102	Na Duang	Loei
3	2018	2	15	101	Phatthana Nikhom	Lop Buri
4	2018	5	14.5	99.5	Bo Phloi	Kanchanaburi
5	2018	5	13	102.5	Pong Nam Ron	Chanthaburi
6	2019	7	16	103.5	Si Somdet	Roi Et
7	2018	7	18	100	Den Chai	Phrae

รูป 4.41 ตัวอย่างข้อมูลในตาราง illegal_speed_by_area

จากการทดลองแสดงให้เห็นว่าสามารถวิเคราะห์ข้อมูลที่นำเข้ามาในระบบก่อนหน้านี้เพื่อหาจำนวนรถยนต์ที่กระทำความผิดจากความเร็วแยกตามพื้นที่ได้สำเร็จ

4.3 การทดลองนำข้อมูลออกจากตารางใน Hive ไปยัง Postgres database โดยใช้ Sqoop

4.3.1 วัตถุประสงค์

เพื่อนำข้อมูลที่ผ่านการวิเคราะห์แล้วส่งออกไปยัง Relational Database เพื่อนำไปใช้ในส่วนของการแสดงผลต่อไป

4.3.2 วิธีการทดลอง

ทดลองส่งออกข้อมูลที่ผ่านการวิเคราะห์ทั้งหมดใน Hive บน HDFS ไปยัง Postgres โดยผ่าน Connector ของ Sqoop ที่ได้ทำการติดตั้งเอาไว้

4.3.3 ผลการทดลอง

```
postgres=# create table numcartype_by_year (year int,sum_unit_type bigint,unit_type varchar(1));
CREATE TABLE
postgres=# \d numcartype_by_year
                Table "public.numcartype_by_year"
    Column     |         Type         | Collation | Nullable | Default
---------------+----------------------+-----------+----------+---------
 year          | integer              |           |          |
 sum_unit_type | bigint               |           |          |
 unit_type     | character varying(1) |           |          |
```

รูป 4.42 ตัวอย่างตารางใน Postgres database

```
20/02/22 18:03:54 INFO mapreduce.Job: Running job: job_1582254238055_0096
20/02/22 18:03:59 INFO mapreduce.Job: Job job_1582254238055_0096 running in uber mode : false
20/02/22 18:03:59 INFO mapreduce.Job:  map 0% reduce 0%
20/02/22 18:04:03 INFO mapreduce.Job:  map 100% reduce 0%
20/02/22 18:04:04 INFO mapreduce.Job: Job job_1582254238055_0096 completed successfully
20/02/22 18:04:04 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=174996
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=4971
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=4
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=0
        Job Counters
                Launched map tasks=1
                Rack-local map tasks=1
```

รูป 4.43 ตัวอย่างงานที่นำข้อมูลออกไป Postgres database

SQL Shell (psql)

year	month	count_unit_type	unit_type
2018	1	40067	0
2018	1	18101	1
2018	1	44229	3
2018	1	160	4
2018	1	2799	5
2018	1	12339	6
2018	1	62426	7
2018	1	41029	8
2018	1	2607	9
2018	2	76627	0

รูป 4.44 ตัวอย่างตารางข้อมูลใน Postgres database

จากผลการทดลองแสดงให้เห็นว่าสามารถส่งข้อมูลจาก Hive บน HDFS ไปยัง Postgress ได้สำเร็จ

4.4 การทดลองการแสดงผลข้อมูล

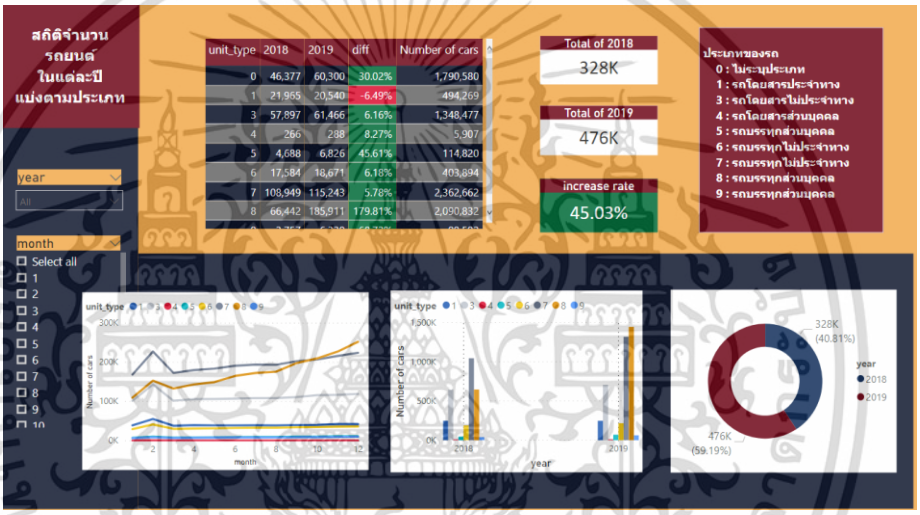
4.4.1 วัตถุประสงค์

เพื่อนำข้อมูลที่ผ่านการวิเคราะห์แล้วมาแสดงให้เห็นในรูปของ Dashboard โดยให้สื่อสารผลการวิเคราะห์ข้อมูลให้อยู่ในรูปแบบที่เหมาะสมและข้อมูลถูกต้องตามความต้องการ

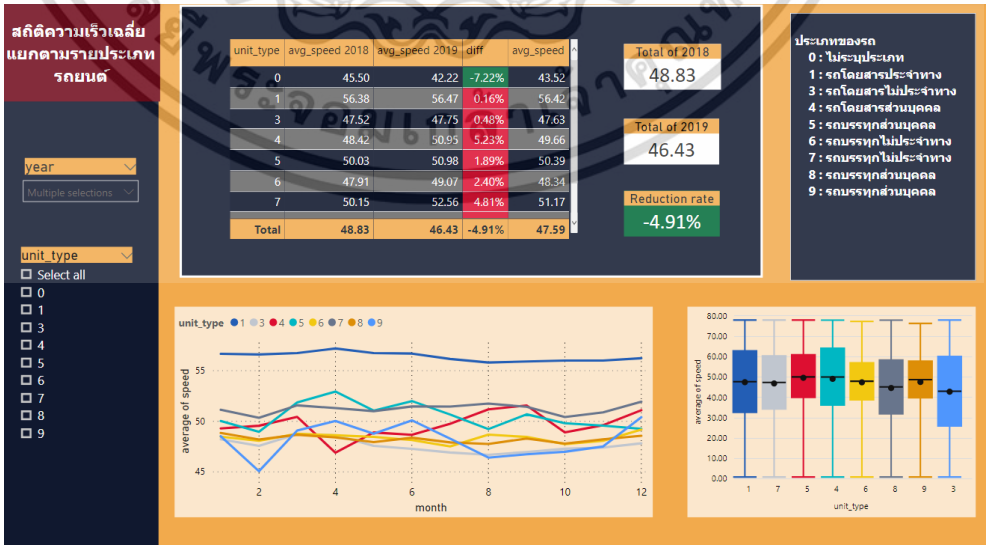
4.4.2 วิธีการทดลอง

นำข้อมูลที่ผ่านการวิเคราะห์แล้วใน Postgres มาสร้างเป็น Visualization บน Power Bi

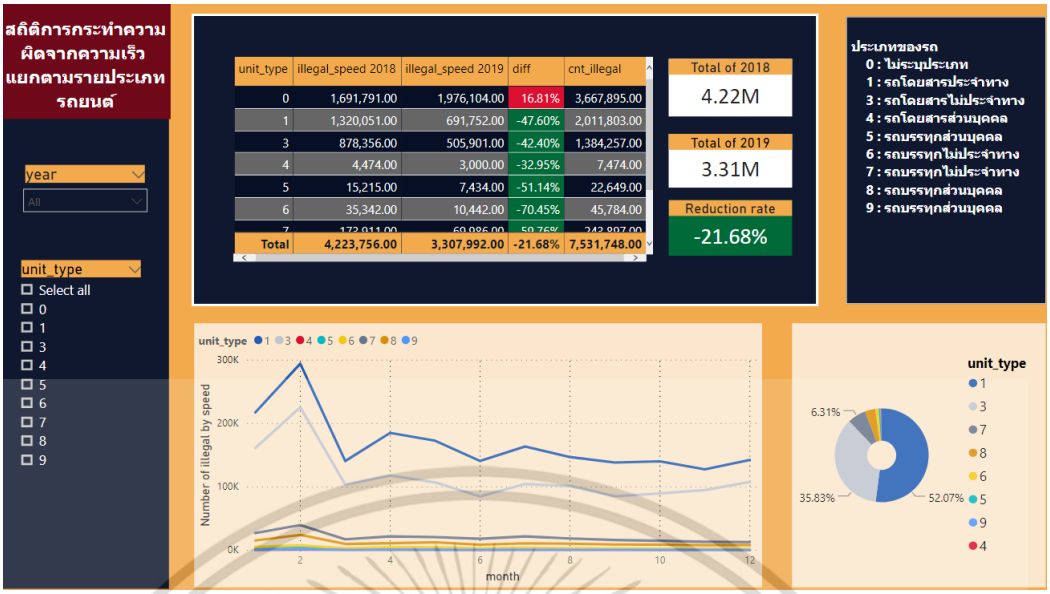
4.3.3 ผลการทดลอง



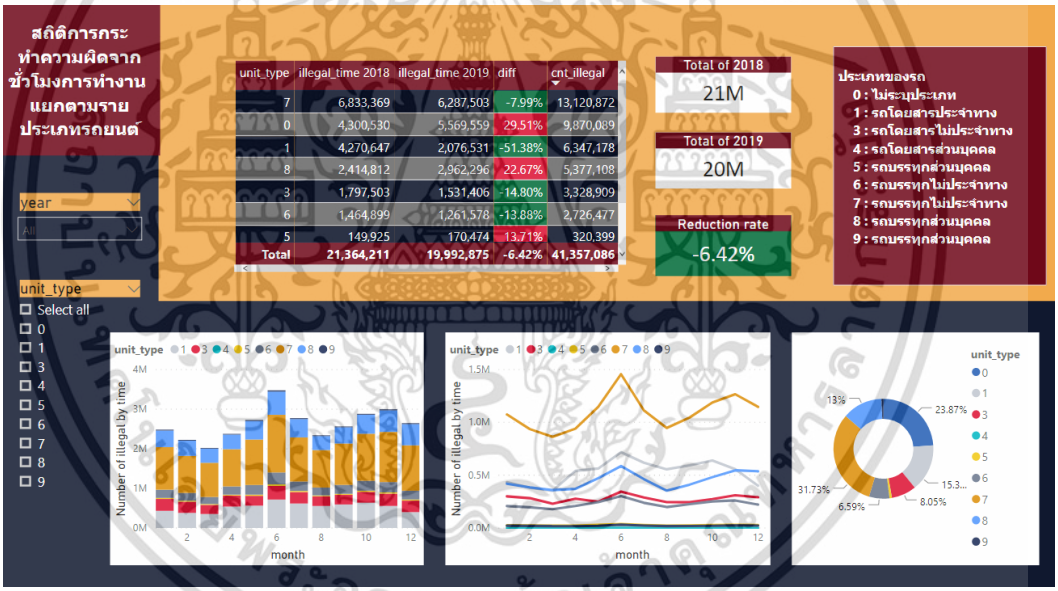
รูป 4.45 Visualization แสดงสถิติจำนวนรถยนต์ในแต่ละปี แบ่งตามประเภท



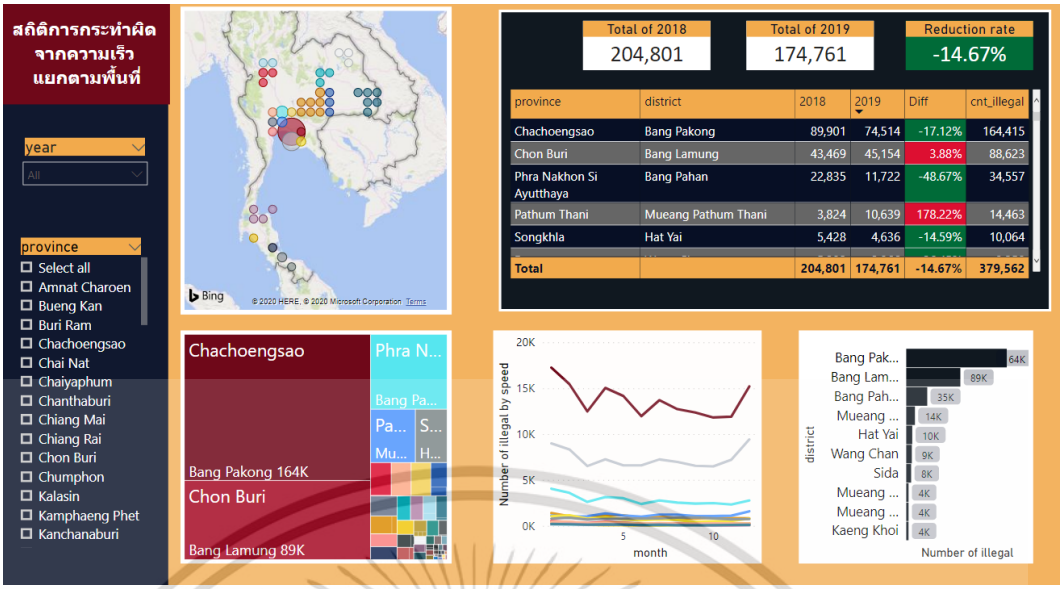
รูป 4.46 Visualization แสดงสถิติความเร็วเฉลี่ยแยกตามรายประเภทรถยนต์



รูป 4.47 Visualization แสดงสถิติการกระทำความผิดจากความเร็วแยกตามรายประเภทรถยนต์



รูป 4.48 Visualization แสดงสถิติการกระทำความผิดจากชั่วโมงการทำงานแยกตามรายประเภทรถยนต์



รูป 4.49 Visualization แสดงสถิติการกระทำความผิดจากความเร็วแยกตามพื้นที่

จากการทดลองแสดงให้เห็นว่าสามารถนำข้อมูลที่ผ่านการวิเคราะห์ในระบบ มาทำเป็น Visualization เพื่อการสื่อสารให้เข้าใจได้สำเร็จ

บท 5
บทสรุปและข้อเสนอแนะ

5.1 บทสรุป

เนื่องจากในปัจจุบันได้มีการนำอุปกรณ์ติดตามตำแหน่งมาติดตั้งเพื่อเพิ่มความปลอดภัยในการขับขี่บนท้องถนนเพราะสามารถติดตามทั้งเวลา, ตำแหน่ง, ความเร็วและประเภทของรถได้ โดยจากเดิมข้อมูลเหล่านี้จะถูกจัดเก็บไว้ใน Relational Database ซึ่งมีแนวโน้มว่าจะไม่สามารถรองรับข้อมูลที่เพิ่มขึ้นตลอดเวลานี้ได้และยังเสี่ยงต่อการเกิดข้อมูลสูญหาย รวมทั้งระบบนี้ยังใช้ Python ในการวิเคราะห์ผลข้อมูลต่าง ๆ ซึ่งใช้เวลาค่อนข้างมาก ผู้จัดทำจึงพัฒนาระบบข้อมูลขนาดใหญ่สำหรับวิเคราะห์ข้อมูลการขนส่ง (Big Data System for Transportation Data Analytics) ขึ้นมา เพื่อให้ระบบที่มีโครงสร้างการทำงานแบบกระจาย ทำให้สามารถทำงานได้รวดเร็วขึ้น ,คงทนต่อความเสียหาย และสามารถจัดการกับทรัพยากรต่าง ๆในระบบได้ดียิ่งขึ้น โดยจะแบ่งเป็น ส่วนของการนำเข้าข้อมูล ซึ่งเป็นส่วนที่รองรับการนำเข้าข้อมูลขนาดใหญ่และต้องมีการนำเข้าข้อมูลตลอดเวลาจึงออกแบบ Pipeline การนำเข้าข้อมูลโดยใช้การทำงานร่วมกันของ Flume และ Kafka จากนั้นใช้ Spark Streaming ในการแปลงข้อมูลให้เหมาะสมเพื่อนำไปจัดเก็บใน Hive บนHDFS, ส่วนการจัดเก็บข้อมูล ใช้สำหรับเตรียมข้อมูลก่อนนำไปทำการวิเคราะห์ข้อมูลเนื่องจากข้อมูลมีจำนวนมาก โดยการจัดเก็บข้อมูลต้องคำนึงถึงความเร็วในการ Query ข้อมูลและขนาดของข้อมูล จึงจัดเก็บข้อมูลในรูปแบบของ ORC และมีแบ่งการเก็บข้อมูลออกเป็น Partition ใน Hive, ส่วนการวิเคราะห์ข้อมูล ใช้ในการนำข้อมูลไปวิเคราะห์ด้วย PySpark แล้วเก็บไว้ใน Hive เพื่อเตรียมส่งต่อข้อมูลออกสำหรับนำข้อมูลไปแสดงผลต่อไป, ส่วนการนำข้อมูลออก ใช้ส่งออกข้อมูลที่ถูกวิเคราะห์แล้วจากรูปแบบตางรางใน Hive ไปเก็บไว้ที่ Postgres Database เนื่องจากตัวแสดงผลข้อมูลนั้นไม่สามารถดึง เพื่อนำไปแสดงผลได้เลย และส่วนการแสดงผลข้อมูลที่ใช้ในการแสดงผลข้อมูลในที่ถูกวิเคราะห์แล้วด้วย Power BI

5.2 ปัญหาและอุปสรรค

ปัญหาและอุปสรรคที่เกิดขึ้นในการพัฒนาโครงงานนี้มีหลายประเภท ได้แก่

1) มีแต่ข้อมูลของรถสาธารณะจึงไม่เป็นตัวแทนที่แท้จริงบนท้องถนนได้ดีเนื่องจากไม่ได้มีเพียงแค่รถสาธารณะเท่านั้นที่สัญจรบนท้องถนนแต่ยังมีรถส่วนตัวด้วย

2) ข้อมูลในบางช่วงเวลาขาดหายไป

3) ขาดข้อมูลอุบัติเหตุที่แม่นยำ

4) ขาดข้อมูลที่การวัดการส่ายของรถ

5.3 แนวทางการพัฒนาต่อ

ถ้าต้องการจะวิเคราะห์ข้อมูลในเชิงของการทำ Machine Learning สามารถทำได้โดยในบริการของ Hadoop มีเครื่องมือชื่อ Mahout ซึ่งต้องติดตั้งเพิ่มเติม

ในการนำไปใช้งานระดับองค์กรจะต้องมีเรื่องของการจัดแจงข้อมูลให้กับแผนกต่าง ๆขององค์กร หรือเรียกว่า Data governance เช่น กำหนดให้ข้อมูลส่วนนี้ใช้ได้เฉพาะแผนกที่ทำหน้าที่ Analytic เท่านั้น รวมถึงอาจจะมีการร่วมมือกับค่ายโทรศัพท์มือถือต่าง ๆเพื่อใช้หาตำแหน่งข้อมูลของรถยนต์ส่วนตัว,การทำ Data imputationเพื่อความแม่นยำของข้อมูล, การดูข้อมูลอุบัติเหตุต่าง ๆแบบ Realtime จาก Social Network ต่าง ๆ และการวัดการส่ายของรถเมื่อมีการหักรถแบบกะทันหันเพื่อป้องกันอุบัติเหตุรถได้ดีขึ้น

บรรณานุกรม

Prof. Dr. Thanachart Numnonda .2019. **Big Data Architecture and Analytics Platform**. [Slide].Bangkok: IMC Institute

Vivek Gite.2016. Ubuntu Linux Change Hostname.[online]. Available : https://www.cyberciti.biz/faq/ubuntu-change-hostname-command/

Cloudera, Inc.2020.Recommended **Cluster Hosts and Role Distribution**. [online]. Available: https://www.cloudera.com/documentation/enterprise/5-15-x/topics/cm_ig_host_

Robert Sanders.2019.**Installing Apache Kafka on Cloudera's Quickstart VM**.[online].Available : https://blog.clairvoyantsoft.com/installing-apache-kafka-on-clouderas-quickstart-vm-8245d8d0ebe5

Cloudera, Inc.2016.**Using Kafka with Spark Streaming**.[online].Available : https://docs.cloudera.com/documentation/kafka/1-3-x/topics/kafka_spark.html#xd_583c10bfdbd326ba-590cb1d1-149e9ca9886--6fed

Cloudera, Inc.2016.**Using Kafka with Flume**.[online].Available : https://docs.cloudera.com/documentation/kafka/1-3-x/topics/kafka_flume.html

Lubos Rendek. 2019.**How to enable/disable firewall on Ubuntu 18.04 Bionic Beaver Linux**.[online].Available : https://linuxconfig.org/how-to-enable-disable-firewall-on-ubuntu-1804-bionic-beaver-linux

Cloudera, Inc.2016. **CDH 5.15.x Packaging and Tarball Information**.[online].Available : https://docs.cloudera.com/documentation/enterprise/releasenotes/topics/cdh_vd_cdh_package_tarball_515.html

Tahan. 2017.**Installation failed. Failed to receive heartbeat from agent**. [online]. Available: https://community.cloudera.com/t5/Support-Questions/Installation-failed-Failed-to-receive-heartbeat-from-agent/m-p/67899

Vivek Gite.2017. **Linux Restart NTPD Service Command**. [online]. Available:
https://www.cyberciti.biz/faq/restarting-ntp-service-on-linux/

Coldcode.2018. **Writing the Kafka consumer output to a file**. [online]. Available :
https://www.edureka.co/community/9419/writing-the-kafka-consumer-output-to-a-file

ภาคผนวก ก

การนำเข้าข้อมูลและการจัดเก็บข้อมูล

ในส่วนนี้จะพูดถึงรายละเอียดภายในโปรแกรมที่ใช้ในการนำเข้าข้อมูล ได้แก่ รายละเอียดในไฟล์ของ Flume ที่ใช้งานร่วมกับ Kafka พร้อมแสดงค่าต่างๆที่กำหนด และรายละเอียดโค้ดภายใน Spark Streaming

โปรแกรม ก.1 รายละเอียดของไฟล์ Flume.conf

```
tier1.sources = spool
tier1.channels = memoryChannel
tier1.sinks = k1
#source
tier1.sources.spool.type = spooldir
tier1.sources.spool.channels = memoryChannel
tier1.sources.spool.spoolDir = /home/nmtic/gps
tier1.sources.spool.fileHeader = false
#channel
tier1.channels.memoryChannel.type = memory
tier1.channels.memoryChannel.capacity = 10000
tier1.channels.memoryChannel.transactionCapacity = 3000
# Send to Kafka Broker on Hadoop Node
tier1.sinks.k1.channel = memoryChannel
tier1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
tier1.sinks.k1.batchSize = 600
tier1.sinks.k1.brokerList =  nmtic-2:9092,nmtic-
3:9092,nmtic-4:9092,nmtic-5:9092
tier1.sinks.k1.topic = gps.nmtic
tier1.sinks.k1.zookeeperConnect = nmtic-1:2181,nmtic-
2:2181,nmtic-3:2181,nmtic-4:2181,nmtic-5:2181/kafka
```

โปรแกรม ก.2 โค้ดใน Spark Streaming

```
#!/usr/bin/pyspark2

from __future__ import print_function
from pyspark import SparkContext
from pyspark.sql import SparkSession,Row,HiveContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
from pyspark.sql.functions import
unix_timestamp,col,year,month
def handle_rdd(rdd):
    if not rdd.isEmpty():
        global ss
        df = ss.createDataFrame(rdd,
            schema=['time_stamp','unit_id', 'lat', 'lon',
            'speed', 'unit_type'])
        df_with_year_and_month = df.withColumn("date",(col
                                ("time_stamp").cast("date")))
        df_with_year_and_month.show()
        df_with_year_and_month.write.mode("append")
        .insertInto("default.gpslog")


sc  = SparkContext(appName="gps_1day")
ssc = StreamingContext(sc,1)
ss  = SparkSession.builder.appName("gps_1day")
     .config("spark.sql.warehouse.dir",
     "/user/hive/warehouse").config("hive.metastore.uris",
     "thrift://nmtic-1:9083").config("hive.exec.
      dynamic.partition", "true").config("hive.exec.dynamic
      .partition.mode", "nonstrict").enableHiveSupport()
      .getOrCreate()
ss.sparkContext.setLogLevel('WARN')

#kafkaTopic
ks = KafkaUtils.createDirectStream(ssc, ['gps.nmtic'],
     {'metadata.broker.list': 'nmtic-2:9092,nmtic-
     3:9092,nmtic-4:9092,nmtic-5:9092'})
lines = ks.map(lambda x: x[1].split(','))
transform = lines.map(lambda x: Row(time_stamp =
            x[3],unit_id = x[4], lat = x[0], lon = x[1],
            speed = x[2], unit_type = str(x[5])))
```

(ต่อ)

```
transform.pprint()
transform.foreachRDD(handle_rdd)

ssc.start()
ssc.awaitTermination()
```

ในส่วนของการสร้างตารางใน Hive จะทำการสร้างให้อยู่ใน ORC Format และมีการแบ่ง Partition ของข้อมูลจากวันที่ซึ่งมาจากข้อมูลที่เป็น Timestamp

โปรแกรม ก.3 คำสั่งที่ใช้ในการสร้างตาราง gps_log ใน Hive

```
create table gps_log (unit_id varchar(27),lat float,lon
float,speed tinyint,unit_type tinyint)
partitioned by(date date)
row format delimited fields terminated by ','
stored as orc TBLPROPERTIES('transactional'='true');
```

ภาคผนวก ข

การวิเคราะห์ข้อมูล

ในส่วนนี้จะพูดถึงการวิเคราะห์ข้อมูลโดยจะแสดงโค้ด Pyspark ที่ใช้ในการวิเคราะห์ข้อมูลในแต่ละหัวข้อ

โปรแกรม ข.1 โค้ด Pyspark ที่ใช้สำหรับการวิเคราะห์เพื่อหาจำนวนรถยนต์ในแต่ละประเภทของปี 2561-2562

```
#!/usr/bin/pyspark2
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, HiveContext
from pyspark.streaming import StreamingContext
sc = SparkContext(appName="numcartype")
ss = SparkSession.builder.appName("numcartype").config
     ("spark.sql.warehouse.dir","/user/hive/warehouse")
     .config("hive.exec.dynamic.partition.mode","nonstrict")
     .enableHive Support().getOrCreate()
ss.sparkContext.setLogLevel('WARN')
hive_context = HiveContext(ss)
for x in range(1,13):
   dff = "Select year(date) as year,month(date) as month,
               count(distinct unit_id) as count_unit_type,
                case when unit_type = '' then '0' else
                unit_type end as unit_type
          from default.gpslog where month(date) =  "
          +"'"+str(x)+"'"+ "
          group by year(date), month(date),unit_type"
   df = hive_context.sql(dff)
   df.write.mode("append").insertInto("nmtic_analytic.
   numcartype")
```

สำหรับการวิเคราะห์ข้อมูลเพื่อหาความเร็วเฉลี่ย และจำนวนการกระทำผิดความผิด(จากความเร็ว)ของรถต่อคัน ต่อเดือน แยกตามรายประเภทรถยนต์ ปี 2561 -2562 จะทำการสร้างตารางชื่อ overspeed เพื่อหาระยะเวลาสูงสุดที่รถกระทำความผิด จากนั้นสร้างตาราง illegal_by_speed เพื่อหาความเร็วเฉลี่ยของรถจากตาราง gps_log และจำนวนครั้งที่กระทำความผิดจากตาราง overspeed

โปรแกรม ข.2 โค้ด Pyspark ที่ใช้สำหรับการวิเคราะห์หาระยะเวลาสูงสุดที่รถกระทำความผิดจากความเร็ว

```
#!/usr/bin/pyspark2
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, HiveContext
from pyspark.streaming import StreamingContext
sc = SparkContext(appName="overspeed")
ss = SparkSession.builder.appName("numcartype").config
    ("spark.sql.warehouse.dir","/user/hive/warehouse")
   .config("hive.exec.dynamic.partition.mode","nonstrict")
   .enableHive Support().getOrCreate()
ss.sparkContext.setLogLevel('WARN')
hive_context = HiveContext(ss)
for x in range(1,13):
     dff = """WITH t AS(
           select distinct time_stamp,date,round(lat,1)as
                  lat,round(lon,1)as lon,unit_id,speed,
                  case when speed > 90 then 'true' else
                  'false' end as overspeed,unit_type
           from default.gpslog
   where  month(date)="+"'"+str(x)+"'"+"and
                  speed != 0
    )
           select t3.time_stamp,t3.lat,t3.lon,t3.unit_id,
                  t3.speed,t3.time_overspeed,t3.unit_type,
                  t3.date
           from (select t2.*,max(time_overspeed)over
                        (PARTITION BY grp,unit_id,date )
                         as maxtime
                  from (select t1.*,
                        case when overspeed = 'true' then
                        round((unix_timestamp (time_stamp)
                        - first_value(unix_timestamp
                        (time_stamp))over(PARTITION BY
                        grp,unit_id,date order by
                        time_stamp ))/60,2)else 0 end as
                        time_overspeed
                        from (select *,case when overspeed
                              = 'true' then (row_number()
```

(ต่อ)

```
                                    over (partition by unit_id
                                    ,date order by time_stamp)-
                                    row_number()over (partition
                            )as t1
                        )as t2
                    )as t3 where time_overspeed =maxtime"""
     df =hive_context.sql(dff)
     df.write.mode("append").insertInto("nmtic_analytic.
overspeed")
```

โปรแกรม ข.3 โค้ด Pyspark ที่ใช้สำหรับการวิเคราะห์หาความเร็วเฉลี่ยของรถจากตาราง gps_log และจำนวนครั้งที่กระทำความผิดจากตาราง overspeed

```
#!/usr/bin/pyspark2
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, HiveContext
from pyspark.streaming import StreamingContext
sc =SparkContext(appName="illegal_byspeed")
ss = SparkSession.builder.appName("numcartype").config
    ("spark.sql.warehouse.dir","/user/hive/warehouse")
   .config("hive.exec.dynamic.partition.mode","nonstrict")
   .enableHive Support().getOrCreate()
ss.sparkContext.setLogLevel('WARN')
hive_context =HiveContext(ss)
for x in range(1,13):
     dff ="""with t1 as (
               select year(date)as year,
                      month(date)as month,unit_id,
                    round(avg(speed),2)as avg_speed,
                      case when unit_type ='' then '0'
                      else unit_type end as unit_type
               from default.gpslog
               where month(date)="+"'"+str(x)+"'"+"and
                     speed !=0
               group by year(date),month(date),unit_id
                        ,unit_type),
```

(ต่อ)

```
                t2 as (
                select year(date)as year,
                       month(date)as month,unit_id,
                       count(time_overspeed)as cnt_illegal
                from nmtic_analytic.overspeed
                where  month(date)="+"'"+str(x)+"'"+"
                       and time_overspeed > 3
                group by year(date),month(date),unit_id
        )
                select t1.year,t1.month,t1.unit_id,
                       t1.avg_speed,t2.cnt_illegal,
                       t1.unit_type
                from t1 join t2 on t1.unit_id =t2.unit_id
                        and t1.year =t2.year
                        and t1.month =t2.month """
    df =hive_context.sql(dff)
    df.write.mode("append").insertInto("nmtic_analytic.illegal_b
    y_speed")
```

สำหรับการวิเคราะห์ข้อมูลเพื่อหาการกระทำความผิดจากชั่วโมงการทำงาน ปี 2561-2562 จะทำการสร้างตารางชื่อ overtime เพื่อหาระยะเวลาที่มากที่สุดของช่วงเวลาที่รถมีการทำงานและการพักรถของช่วงเวลาถัดไป จากนั้นสร้างตาราง illegal_by_time เพื่อหาจำนวนครั้งที่กระทำผิดจากตาราง overtime

โปรแกรม ข.4 โค้ด Pyspark ที่ใช้สำหรับการวิเคราะห์หาระยะเวลาที่มากที่สุดของช่วงเวลาที่รถมีการทำงานและการพักรถของช่วงเวลาถัดไป

```
#!/usr/bin/pyspark2
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, HiveContext
from pyspark.streaming import StreamingContext
sc = SparkContext(appName="overtime")
ss = SparkSession.builder.appName("numcartype").config
    ("spark.sql.warehouse.dir","/user/hive/warehouse")
   .config("hive.exec.dynamic.partition.mode","nonstrict")
   .enableHive Support().getOrCreate()
ss.sparkContext.setLogLevel('WARN')
hive_context = HiveContext(ss)

for y in range(2018,2020,1):
     for x in range(1,13):
         dff = """WITH t AS(
                  select distinct time_stamp,date,
                   round(lat,1)as lat,
                   round(lon,1)as lon,
                   unit_id,speed,
                  case when speed > 0 then 'true'
                   else 'false' end as
                   active,unit_type
                 from default.gpslog
                 where year(date) ="" +"'"+str(y)+"'"+
                   """ and month(date)="""
                  +"'"+str(x)+"'"+""")
                 select t3.time_stamp,t3.lat,t3.lon,
                   t3.unit_id,t3.speed,
                   t3.time_active,
                   lead(t3.time_nonactive,1)
                   over(PARTITION BY unit_id,
                   date order by time_stamp)as
                   time_nonactive,t3.unit_type,
                   t3.date
```

(ต่อ)

```
from(select t2.*, case when active =
     'true' then max(time_active)
    over(PARTITION BY grp,unit_id,date
    )else 0 end as maxtime_active,
     case when active ='false' then
     max(time_nonactive)over
    (PARTITION BY grp,unit_id,date )
     else 0 end as maxtime_nonactive
     from(select t1.*, case when active
       ='true' then
       round((unix_timestamp
      (time_stamp)-
       first_value(unix_timestamp
      (time_stamp))over(PARTITION BY
       grp,unit_id,date order by
       time_stamp))/60,2)else 0 end
       as time_active,
       case when active ='false'
       then round((unix_timestamp
      (time_stamp)-
       first_value(unix_timestamp
      (time_stamp))over(PARTITION BY
       grp,unit_id,date order by
       time_stamp))/60,2) else 0
       end as time_nonactive
       from (select *,(row_number()
       over (partition by
       unit_id,date order by
       time_stamp) - row_number()
       over (partition by
       unit_id,active,date order
       by time_stamp))as grp from t
```

(ต่อ)

```
                        )as t1
                    )as t2
            )as t3  where time_active =
                                maxtime_active and
                                time_nonactive =
                                maxtime_nonactive
    """
        df =hive_context.sql(dff)
        df.write.mode("append").insertInto("nmtic_analytic
       .overtime")
```

โปรแกรม ข.5 โค้ด Pyspark ที่ใช้สำหรับการวิเคราะห์หาจำนวนครั้งที่กระทำผิดจากตาราง overtime

```
#!/usr/bin/pyspark2
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, HiveContext
from pyspark.streaming import StreamingContext
sc =SparkContext(appName=" illegal_bytime.py")
ss = SparkSession.builder.appName("numcartype").config
    ("spark.sql.warehouse.dir","/user/hive/warehouse")
   .config("hive.exec.dynamic.partition.mode","nonstrict")
   .enableHive Support().getOrCreate()
ss.sparkContext.setLogLevel('WARN')

hive_context =HiveContext(ss)

for x in range(1,13):
    dff = """select year(date) as year,
                    month(date) as month,
                    unit_id,
                    count(*) as cnt_illegal,
                    case when unit_type = '' then '0' else
                    unit_type end as unit_type
              from nmtic_analytic.overtime
              where (time_active > 240 or
                    (time_active = 240 and time_nonactive <
                    121)) and month(date) ="""
                    +"'"+str(x)+"'"+ """
```

(ต่อ)

```
            group by year(date),month(date),unit_id,
                     unit_type
        """
    df = hive_context.sql(dff)
    df.write.mode("append").insertInto("nmtic_analytic.
    illegal_by_time")
```

โปรแกรม ข.6 โค้ด Pyspark ที่ใช้สำหรับการวิเคราะห์หาการกระทำความผิด (จากความเร็ว) แยกตามพื้นที่

```
#!/usr/bin/pyspark2
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, HiveContext
from pyspark.streaming import StreamingContext
sc = SparkContext(appName="illegai_bytime")
ss = SparkSession.builder.appName("numcartype").config
    ("spark.sql.warehouse.dir","/user/hive/warehouse")
   .config("hive.exec.dynamic.partition.mode","nonstrict")
   .enableHive Support().getOrCreate()
ss.sparkContext.setLogLevel('WARN')
hive_context =HiveContext(ss)
for x in range(1,13):
    dff ="""with t1 as (
              select year(date)as year,
                     month(date)as month,
                     round(lat,1)as lat,
                     round(lon,1)as lon,
                     count(time_overspeed)as cnt_illegal
             from nmtic_analytic.overspeed
              where time_overspeed > 3 and month(date)
                    ="""+"'"+str(x)+"'"+"""
              group by  year(date),month(date),lat,lon
    )
          t2 as (select * from default.locs10km)
         select t1.year,t2.lat,t2.lon,t2.district,
                t2.province,t1.cnt_illegal
          from t1 join t2 on t1.lat =t2.lat and t1.lon
```

(ต่อ)

```
            from t1 join t2 on t1.lat =t2.lat and t1.lon
                 =t2.lon
            group by t1.year,t1.month,t2.lat,t2.lon,
                     t2.district,t2.province,
                     t1.cnt_illegal
"""
 df =hive_context.sql(dff)
 df.write.mode("append").insertInto("nmtic_analytic.
illegal_speed_byarea")
```