**Data Collection**

ประเภทของข้อมูล

* **Structured**: ข้อมูลที่มีการจัดโครงสร้างอย่างดี มีการจัดระเบียบ ป้ายกำกับ และมีโมเดลข้อมูลที่กำหนดไว้ล่วงหน้าตามรูปแบบตาราง
* **Unstructured**: ข้อมูลที่ยังไม่ได้จัดโครงสร้าง เช่น รูปภาพ, เสียง, วิดีโอ, ข้อมูลจากเซนเซอร์, ข้อความ, และโพสต์ในโซเชียลมีเดีย
* **Semi-structured**: ข้อมูลที่มีส่วนประกอบของทั้งข้อมูลที่มีโครงสร้างและไม่มีโครงสร้าง โดยส่วนใหญ่จะเป็นข้อมูลที่ไม่มีโครงสร้าง แต่มีข้อมูลเมตาที่ระบุคุณลักษณะบางอย่าง เช่น ไฟล์ XML

ประเภทของแหล่งข้อมูล

* **Batch Data**: ชุดข้อมูลที่ถูกเก็บรวบรวมในช่วงเวลาหนึ่ง
* **Stream Data**: ชุดข้อมูลที่ถูกเก็บรวบรวมและประมวลผลเกือบจะในเวลาเดียวกัน (ความแตกต่างระหว่างการเก็บข้อมูลและการประมวลผลอยู่ในช่วงมิลลิวินาที)

**Data Preparation and Storage**

* ดำเนินการการสกัด (E), การเปลี่ยนแปลง (T), และการโหลด (L)ข้อมูลเข้าสู่ฐานข้อมูลหรือระบบไฟล์
* ลำดับของกระบวนการอาจเป็น ETL (Extract, Transform, Load) หรือ ELT (Extract, Load, Transform) ขึ้นอยู่กับแหล่งข้อมูลและกระบวนการที่ต้องการ
* ข้อมูลที่รวบรวมมักจะมีข้อผิดพลาด, ค่าที่ขาดหายไป, ข้อมูลซ้ำ, และความไม่สอดคล้อง เช่น การย่อคำ, หน่วยวัด, และรูปแบบที่ไม่ตรงกัน
* การเตรียมข้อมูลล่วงหน้า (Data Pre-processing) เป็นสิ่งจำเป็น
* ประเภทของการจัดเก็บข้อมูล
  + **Distributed File Systems (DFS)**: ระบบจัดเก็บข้อมูลที่กระจายข้อมูลไปยังหลายตำแหน่งเพื่อให้ผู้ใช้หลายคนสามารถเข้าถึงจากสถานที่ต่างๆ
* **Non-relational Databases (NoSQL)**
  1. **ไม่มีสคีมาที่เข้มงวดและคำจำกัดความที่เป็นทางการ**: ไม่ต้องการโครงสร้างข้อมูลที่ชัดเจนเหมือนฐานข้อมูลเชิงสัมพันธ์
  2. **ออกแบบมาเพื่อจัดการกับข้อมูลจำนวนมาก**: ที่ไม่สามารถจัดการได้ด้วยระบบการจัดการฐานข้อมูลเชิงสัมพันธ์ (RDBMS) แบบดั้งเดิม

**Data Processing and Analysis**

* กระบวนการและการวิเคราะห์ ขึ้นอยู่กับประเภทของแหล่งข้อมูล ปัญหาที่ระบุ และผลลัพธ์ที่คาดหวัง
* **ประเภทของกระบวนการ** สามารถแบ่งออกเป็นสองกลุ่ม
  1. **Batch Processing** มีประสิทธิภาพสูงในการประมวลผลข้อมูลจำนวนมากที่ถูกเก็บรวบรวมในช่วงระยะเวลาหนึ่ง
  2. **Stream Processing**: มีประสิทธิภาพสูงและความสามารถในการประมวลผลข้อมูลจำนวนมากที่ถูกเก็บรวบรวมเกือบจะในเวลาเดียวกัน

**Visualisation**

* คือรายงานผลลัพธ์ของการวิเคราะห์ Big Data
* การสร้างภาพข้อมูลควรส่งมอบข้อมูลเชิงลึกที่มีความหมายและเข้าใจได้
* รายงานสามารถเป็นกราฟ, แผนภูมิ, ตาราง หรือการรวมกันของสิ่งเหล่านี้

**Apache Hadoop**

* **Apache Hadoop** เป็นแพลตฟอร์มสำหรับการจัดเก็บและประมวลผล Big Data
* Hadoop อนุญาตให้ข้อมูลจำนวนมากถูกประมวลผลในรูปแบบของระบบไฟล์กระจาย (Hadoop Distributed File System: HDFS)
* **Distributed System** (หรือที่เรียกว่า Distributed Computing) คือระบบที่ประกอบด้วยคอมพิวเตอร์หรือเซิร์ฟเวอร์หลายเครื่อง (เรียกว่า nodes) ที่สื่อสารและทำงานร่วมกันเพื่อดำเนินกระบวนการ ซึ่งสามารถมองเห็นเป็นคอมพิวเตอร์เครื่องเดียวสำหรับผู้ใช้
* ระบบกระจายที่มีประสิทธิภาพสามารถทำงานต่อไปได้แม้จะเกิดความล้มเหลวของเครือข่ายหรือข้อบกพร่องของโหนด

**CAP Theorem**

* **CAP Theorem** ย่อมาจาก **Consistency (C)**, **Availability (A)**, และ **Partition Tolerance (P)**
* **CAP Theorem** ระบุว่าระบบกระจายไม่สามารถมี **Consistency**, **Availability**, และ **Partition Tolerance** ทั้งสามอย่างพร้อมกันได้
* **Consistency** หมายถึงการที่ทุกโหนดในกลุ่ม (cluster) มีสำเนาของข้อมูลที่เหมือนกันทั้งหมด
* **Availability** หมายถึงข้อมูลในแต่ละโหนดจะต้องสามารถเข้าถึงได้ทุกครั้งที่มีการร้องขอการเข้าถึง
* **Partition Tolerance** หมายถึงระบบสามารถทำงานต่อไปได้แม้จะเกิดความล้มเหลวของเครือข่าย

**Apache Hadoop Architecture**

**Apache Hadoop เดิมประกอบด้วย 2 องค์ประกอบ**

* **Hadoop Distributed File System (HDFS)**
* **MapReduce Programming Model**

**Apache Hadoop version 2 (and above) contains 3 components:**

* **Hadoop Distributed File System (HDFS)**
* **MapReduce programming model**
* **Yet Another Resource Negotiator (YARN)**

**Hadoop Distributed File System (HDFS)**

คือระบบไฟล์ที่ไฟล์ต่างๆ ถูกกระจายไปทั่วโหนดของคลัสเตอร์

ลักษณะของ HDFS

* + **Scalable Storage**
    - ไฟล์ข้อมูลขนาดใหญ่จะแบ่งออกเป็นบล็อก แต่ละบล็อกมีขนาด 64 เมกะไบต์ บล็อกทั้งหมดจะถูกกระจายและสำรองไว้ในหลายโหนดภายในคลัสเตอร์
* Replication
  + - HDFS จะสำเนาบล็อกสูงสุดถึง 3 ชุด แต่ละชุดจะกระจายไปยังโหนดที่แตกต่างกัน นี้ช่วยให้ระบบเชื่อถือได้และทนต่อข้อผิดพลาด
* Streaming Data Access
  + - HDFS มีประสิทธิภาพสูงในการอ่านและเขียนข้อมูลแบบสตรีม เหมาะสำหรับการเข้าถึงข้อมูลที่มี throughput สูง

ใน HDFS, โหนดสามารถจัดประเภทได้เป็น 2 ประเภท

**Datanode**: โหนดที่จัดเก็บข้อมูล

**Namenode**: โหนดที่จัดการเมทาดาต้าและการควบคุมการเข้าถึงข้อมูล

**Datanode**

* + HDFS แบ่งไฟล์ข้อมูลออกเป็นบล็อก
  + แต่ละบล็อกจะถูกสำเนาเป็น 3 ชุดและกระจายไปยังโหนดต่างๆ ในคลัสเตอร์
  + โหนดที่จัดเก็บสำเนาของบล็อกเหล่านี้เรียกว่า **Datanode**
  + ให้บริการข้อมูลในบล็อกเมื่อมีการเรียกใช้งานการอ่านหรือเขียน

**Namenode**

* Namenode หรือ Master Node ไม่เก็บบล็อกข้อมูล
* มันเก็บข้อมูลเมทาดาต้าของระบบไฟล์ ซึ่งรวมถึงการแมปบล็อกและ Datanodes ที่เกี่ยวข้อง
* ระบบไฟล์ประกอบด้วยไฟล์ fsimage และ edits
  + - fsimage file: เก็บข้อมูลเมทาดาต้าของระบบไฟล์
    - edits file: เก็บการอัปเดตข้อมูลเมทาดาต้า ดังนั้น ขนาดของไฟล์ edits จะเพิ่มขึ้นตามเวลา

**How it works?**

1. ลูกค้าส่งคำขอไปยัง Namenode
2. Namenode ติดต่อกับ Secondary Namenode
3. Secondary Namenode ดำเนินกระบวนการ Checkpointing ซึ่งเป็นกระบวนการตรวจสอบไฟล์ edits และเขียนไปยังไฟล์ fsimage ใหม่
4. Secondary Namenode ส่งไฟล์กลับไปยัง Namenode
5. หมายเหตุ: Secondary Namenode ถูกใช้เนื่องจาก Namenode ไม่มีทรัพยากรในการอัปเดตไฟล์เนื่องจากมีการดำเนินงานอื่น ๆ ที่ต้องจัดการ
6. Namenode มีการแมพบล็อกข้อมูลกับ Datanodes ที่เกี่ยวข้อง
7. ดังนั้น Namenode ยังติดต่อกับ Datanode เพื่อตรวจสอบสถานะว่า Datanode ยังทำงานอยู่หรือไม่
8. หาก Datanode ยังทำงานอยู่ มันจะส่งสัญญาณหัวใจ (heartbeat) และข้อมูลในบล็อก (Block Report) กลับไปยัง Namenode
9. Namenode จากนั้นตอบสนองลูกค้าด้วย Datanode ที่ลูกค้าสามารถดำเนินการอ่านและเขียนข้อมูลได้

**Yet Another Resource Negotiator (YARN)**

* จัดการทรัพยากรที่ใช้ใน Apache Hadoop
* YARN ประกอบด้วย 2 ประเภทของโหนด คือ Master และ Slave
  + - Master node (Resource Manager) จัดการทรัพยากร (Scheduler) ที่ใช้ในการรันแอพพลิเคชัน (Application Manager)
    - Slave node วงจรชีวิตของแต่ละแอพพลิเคชันจะได้รับการจัดการโดย Application Master แอพพลิเคชันแต่ละตัวใช้ทรัพยากร เช่น หน่วยความจำและ CPU กลุ่มทรัพยากรที่เรียกว่า Container จะได้รับการจัดการโดย Node Manager

Slave node อาจมี Container มากกว่าหนึ่งตัว

* Resource Manager ต้องการ Scheduler เพื่อจัดการทรัพยากร
* อัลกอริทึมการจัดสรรสามารถแบ่งออกเป็น 3 ประเภท
  + - **First in first out (FIFO)** งานที่เข้ามาก่อนจะถูกนำไปประมวลผลก่อนโดยไม่คำนึงถึงความสำคัญและขนาดของงาน
    - **Fair Scheduler** ทรัพยากรสำหรับหลายงานจะถูกจัดสรรอย่างเท่าเทียมกัน

งานที่สั้นกว่าจะเสร็จก่อนและช่องทำงาน (task slots) จะถูกจัดสรรให้งานใหม่

เมื่อมีเพียงงานเดียวที่ทำงานอยู่ ทรัพยากรทั้งหมดจะถูกสงวนไว้สำหรับงานนั้น

หากมีงานใหม่เข้ามา ทรัพยากรจะถูกแบ่งเพื่อจัดสรรให้กับงานเหล่านั้น

Scheduler นี้มีประโยชน์เมื่อมีผู้ใช้หลายคนแชร์ทรัพยากรเดียวกัน

* + - Capacity Scheduler

Clients มีขีดจำกัดของความจุในการรันงานอย่างเท่าเทียมกัน

ความจุจะถูกกำหนดให้กับคิวพร้อมกับงาน

แต่ละคิวจะถูกจัดตารางด้วยการจัดลำดับแบบ FIFO และตามความสำคัญ

หากเวลารอคอยน้อยกว่าคิวที่ถูกจัดตาราง จะกระจายงานไปยังคิวอื่นเพื่อให้แน่ใจว่างานมีความจุที่เท่าเทียมกัน

มีประโยชน์เมื่อผู้ใช้หลายคนที่มีความสำคัญต่างกันแชร์ทรัพยากรร่วมกัน

**MapReduce**

* เป็นรูปแบบการเขียนโปรแกรมสำหรับการประมวลผลข้อมูล
* ข้อมูลที่นำเข้าและส่งออกของ MapReduce อยู่ในรูปแบบคู่คีย์-ค่า (key-value pairs)
* ประกอบด้วยฟังก์ชันหลัก 2 ฟังก์ชัน
  + - **Map Function** - ประมวลผลข้อมูลและสร้างผลลัพธ์กลาง (intermediate results)
    - **Reduce Function**
      * ผลลัพธ์กลางถูกจัดเรียงตามคีย์
      * คีย์ที่ถูกจัดเรียงและค่าที่สอดคล้องจะถูกจัดกลุ่มและสลับที่
* รูปแบบ MapReduce ที่ใช้ในกระบวนการวิเคราะห์ข้อมูล ได้แก่ นับ (count), ค่าสูงสุด/ต่ำสุด (max/min), ค่าเฉลี่ย (average), อันดับสูงสุด N (top-N), กรอง (filter), แยกความแตกต่าง (distinct), แบ่งกลุ่ม (binning), ดัชนีแบบย้อนกลับ (inverted index), การจัดเรียง (sorting), และ การเชื่อม (joins)

**MRJob**

MapReduce สามารถนำไปใช้โดยใช้ไลบรารี MRJob ในภาษาโปรแกรม Python

**MRJob** คือ ไลบรารีในภาษาโปรแกรม Python ที่ใช้สำหรับเขียนและรันงาน **MapReduce** บนแพลตฟอร์มต่างๆ เช่น Hadoop หรือ Amazon Elastic MapReduce (EMR) โดยไม่จำเป็นต้องเขียนโค้ดในภาษา Java ซึ่งเป็นภาษาหลักของ Hadoop

pip install setuptools

* ใน MapReduce Count ฟังก์ชัน Map ใช้ฟิลด์เป็นคีย์เพื่อ group-by และใช้ค่า 1 หรือค่าที่ต้องการในการคำนวณจำนวน ขณะที่ฟังก์ชัน Reduce ใช้ลิสต์ของค่าที่ถูกจัดกลุ่มตามคีย์จากฟังก์ชัน Map และรวมค่าทั้งหมดในแต่ละกลุ่มเพื่อคำนวณจำนวน
* ใน **MapReduce Average** ฟังก์ชัน **Map** ใช้ฟิลด์เป็นคีย์เพื่อ group-by และใช้ค่าที่ต้องการในการคำนวณค่าเฉลี่ย ขณะที่ฟังก์ชัน **Reduce** ใช้ลิสต์ของค่าที่ถูกจัดกลุ่มตามคีย์จากฟังก์ชัน Map และคำนวณค่าเฉลี่ยสำหรับแต่ละกลุ่ม
* ฟังก์ชัน **Map** ใช้ฟิลด์เป็นคีย์เพื่อ group-by และใช้ค่าที่ต้องการในการคำนวณค่าสูงสุดหรือต่ำสุด ขณะที่ฟังก์ชัน **Reduce** ใช้ลิสต์ของค่าที่ถูกจัดกลุ่มตามคีย์จากฟังก์ชัน Map และหาค่ามากสุดหรือน้อยสุดในแต่ละกลุ่ม
* `MRStep` คือองค์ประกอบใน MRJob ที่ใช้เพื่อกำหนดขั้นตอนต่าง ๆ ของกระบวนการ MapReduce เช่น การแมพและการรีดิวซ์ รวมถึงลำดับการทำงานของขั้นตอนเหล่านั้น
* ใน **MapReduce Top-N** ฟังก์ชัน **Map** ใช้ฟิลด์เป็นคีย์เพื่อ group-by และใช้ค่าที่ต้องการในการคำนวณ top-N ขณะที่ฟังก์ชัน **Reduce** ใช้ลิสต์ของค่าที่ถูกจัดกลุ่มตามคีย์จากฟังก์ชัน Map แล้วทำการจัดเรียงค่าตามลำดับและหาค่า top-N ในแต่ละกลุ่ม

คำว่า **"Top-N"** หมายถึงการเลือกหรือจัดลำดับข้อมูลที่ดีที่สุดหรือสำคัญที่สุดจากชุดข้อมูลตามจำนวนที่กำหนด โดยทั่วไปจะใช้ในหลายบริบท เช่น การวิเคราะห์ข้อมูล, การค้นหา, และการจัดอันดับ

ตัวอย่างการใช้ "Top-N"

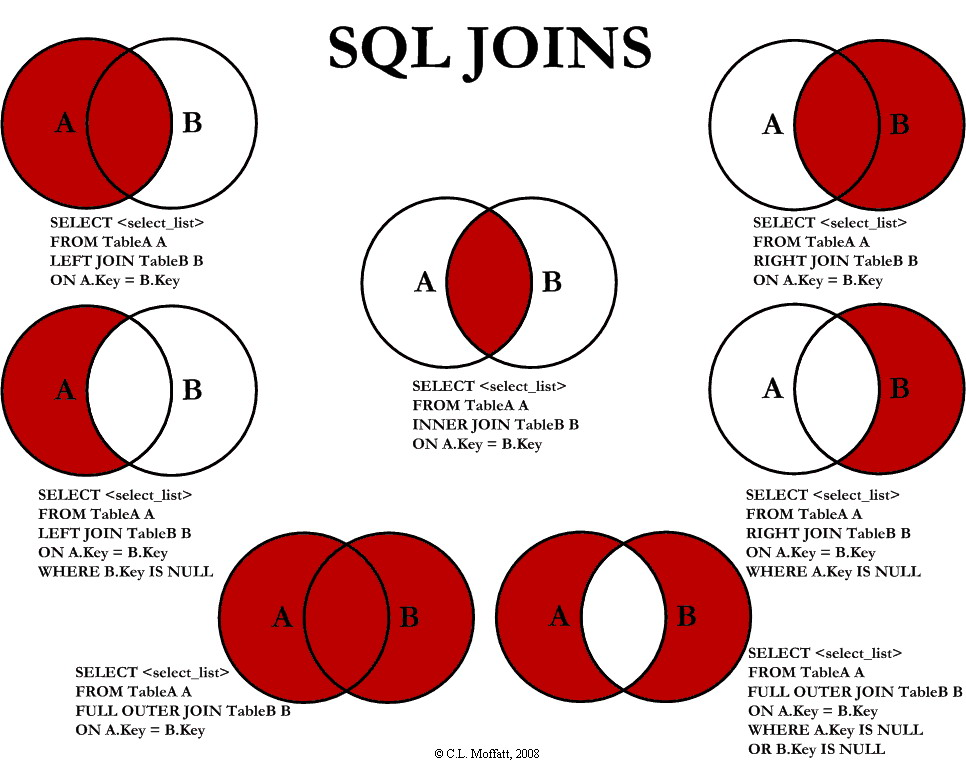
1. การจัดอันดับสินค้า (Top-N Products) ในระบบร้านค้าออนไลน์ อาจต้องการแสดง 10 สินค้าที่ขายดีที่สุด (Top-10 Products) หรือ 5 สินค้าที่ยอดนิยมที่สุด (Top-5 Popular Items)
2. การค้นหา (Top-N Search Results) ในการค้นหาข้อมูล อาจต้องการแสดงผลการค้นหาที่ดีที่สุด 10 ผลลัพธ์ (Top-10 Search Results) หรือ 20 บทความที่เกี่ยวข้องที่สุด (Top-20 Relevant Articles)

* ใน **MapReduce Filter** การกรองใช้สำหรับการกรองชุดข้อมูลย่อยตามเกณฑ์ที่กำหนด ฟังก์ชัน **Map** จะทำการกรองข้อมูลเพียงอย่างเดียว โดยไม่จำเป็นต้องใช้ฟังก์ชัน **Reduce** และสมมติว่าเราต้องการดูจำนวนการตอบสนองสำหรับแต่ละประเภทสถานะในปี 2018 ซึ่งจำนวนการตอบสนองต้องมากกว่า 2,000
* ใน **MapReduce Distinct** การเลือกค่า distinct ใช้สำหรับการเลือกค่าที่ไม่ซ้ำจากชุดข้อมูล ฟังก์ชัน **Map** จะจัดกลุ่มระเบียนที่มีคีย์เดียวกัน โดยที่ค่าอาจเป็น None ขณะที่ฟังก์ชัน **Reduce** ใช้ลิสต์ของค่าที่ถูกจัดกลุ่มตามคีย์จากฟังก์ชัน Map และสามารถคืนค่าเป็น None ได้
* ใน **MapReduce Binning** การ binning ใช้สำหรับการแบ่งระเบียนออกเป็นหมวดหมู่หรือกลุ่ม ฟังก์ชัน **Map** จะทำการจัดกลุ่มระเบียนตามหมวดหมู่ที่กำหนด โดยไม่จำเป็นต้องใช้ฟังก์ชัน **Reduce**
* โครงสร้างข้อมูล inverted index จะเก็บเนื้อหาเช่นคำในเอกสารและตำแหน่ง

เช่นหมายเลขเอกสารหรือชื่อไฟล์ ฟังก์ชัน **Map** ใช้ฟิลด์เป็นคีย์สำหรับการสร้างดัชนี ขณะที่ฟังก์ชัน **Reduce** ใช้หมายเลขเอกสารหรือค่าต่าง ๆ

* **MapReduce Sorting** การเรียงลำดับบันทึกข้อมูลตามฟิลด์ที่กำหนด โดยฟังก์ชัน **Map** จะใช้ฟิลด์เป็นคีย์เพื่อจัดกลุ่มและใช้ค่าที่ต้องการในการคำนวณค่าเฉลี่ย ขณะที่ฟังก์ชัน **Reduce** จะใช้ลิสต์ของค่าที่ถูกจัดกลุ่มโดยคีย์จากฟังก์ชัน Map จากนั้นใช้ฟังก์ชันการเรียงลำดับของ Python เพื่อจัดลำดับลิสต์ของค่า
* MapReduce Join ใช้สำหรับรวมข้อมูลจากไฟล์สองไฟล์ขึ้นไปตามฟิลด์ที่เหมือนกัน โดยมีประเภทต่างๆ เช่น Inner Join ซึ่งจะส่งคืนข้อมูลที่มีค่าตรงกันระหว่างไฟล์; Full Outer Join ซึ่งส่งคืนข้อมูลทั้งหมดทั้งที่ตรงกันและไม่ตรงกัน รวมถึงแถวที่ไม่มีข้อมูลตรงกัน; Left Outer Join ซึ่งส่งคืนข้อมูลทั้งหมดจากไฟล์ด้านซ้ายและแสดงค่าเป็นค่าว่างสำหรับคอลัมน์ที่ไม่ตรงกันในไฟล์ด้านขวาและ Right Outer Join ซึ่งส่งคืนข้อมูลทั้งหมดจากไฟล์ด้านขวาและแสดงค่าเป็นค่าว่างสำหรับคอลัมน์

ที่ไม่ตรงกันในไฟล์ด้านซ้าย



**Apache Spark**

* ถูกนำเสนอครั้งแรกในปี 2009 โดย Matei Zaharia
* ถูกเปิดเป็นโอเพ่นซอร์สในปี 2010
* กลายเป็นส่วนหนึ่งของ Apache Software Foundation ในปี 2013
* เป็นทางเลือกสำหรับ MapReduce ใน Apache Hadoop
* แก้ไขข้อเสียของ MapReduce ที่มีต้นทุนการคำนวณสูงและการใช้งานดิสก์ระหว่างฟังก์ชัน Map และ Reduce
* คุณลักษณะของ Apache Spark คือ โครงสร้างแบบกระจาย, มีความทนทานต่อข้อผิดพลาด, และโครงสร้างในหน่วยความจำ (เรียกว่า Resilient Distributed Dataset: RDD)
* การใช้งานในหน่วยความจำ (In-memory) เป็นข้อได้เปรียบหลักของ Apache Spark เนื่องจากข้อมูลระหว่างการคำนวณสามารถถูกเก็บในหน่วยความจำแรม (RAM) ทำให้เร็วกกว่า Apache Hadoop
* Apache Spark เขียนด้วยภาษา Scala (บนพื้นฐานของ Java Virtual Machine: JVM และ Java)
* Apache Spark สามารถใช้งานด้วยภาษา Python ได้เช่นกัน (เรียกว่า PySpark)

**Key Components of Apache Spark**

**Unified**

* Apache Spark รองรับหลายงานที่จำเป็นในการวิเคราะห์ข้อมูล เช่น การโหลดข้อมูล, กระบวนการสตรีมข้อมูล, และกระบวนการเรียนรู้ของเครื่อง
* ทำให้การเขียนโค้ดสำหรับการวิเคราะห์ข้อมูลง่ายและมีประสิทธิภาพมากขึ้น
* ยังอนุญาตให้ไลบรารีต่างๆ ทำงานร่วมกันได้อย่างไร้รอยต่อ เพื่อให้การวิเคราะห์ข้อมูลมีประสิทธิภาพสูง
* ขยาย API ในตัวอย่างต่อเนื่องเพื่อปรับปรุงประสิทธิภาพและครอบคลุมงานที่ต้องการมากขึ้น

**Computing Engine**

* Spark โหลดข้อมูลจากที่เก็บและคำนวณข้อมูลในหน่วยความจำ (in-memory)
* Spark แก้ปัญหาของ Apache Hadoop ซึ่ง Hadoop ต้องการ HDFS และ MapReduce ในการคำนวณโปรแกรม
* Spark คำนวณได้เร็วกกว่า Apache Hadoop

**Libraries**

* Spark รองรับไลบรารีมาตรฐาน, ไลบรารีภายนอก, และไลบรารีที่สร้างขึ้นเอง
* Spark แบบโอเพ่นซอร์สมากมาย
* นอกจากนี้ยังมีโอเพ่นซอร์สสำหรับไลบรารีภายนอกมากมายที่เราสามารถใช้สำหรับงานเฉพาะต่าง ๆ ได้

**Key Benefits of Apache Spark**

**Simpler to use and operate**

* Spark ใช้งานและดำเนินการได้ง่ายกว่า
* รองรับหลายภาษาโปรแกรม
* Spark แบ่งปัญหาใหญ่เป็นปัญหาเล็กๆ
* แจกจ่ายส่วนของปัญหาเล็กๆ ไปยังผู้แก้ปัญหาต่างๆ
* แต่ละผู้แก้ปัญหาจะสร้างวิธีการแก้ไข
* Spark รวบรวมวิธีการแก้ไขที่แก้ปัญหาเล็กๆ เหล่านั้น
* ประกอบวิธีการแก้ไขเพื่อให้ผลลัพธ์สุดท้ายที่แก้ปัญหาใหญ่

**Fast**

* Databricks ชนะการประกวด Daytona GraySort โดยใช้ Spark เพื่อเรียงลำดับข้อมูล 100TB ในเวลา 23 นาที
* ซึ่งเร็วกว่าสถิติโลกก่อนหน้านี้ที่ใช้เวลา 72 นาทีโดยใช้ Hadoop
* อย่างไรก็ตาม การเรียงลำดับด้วย Spark นั้นเกิดขึ้นบนดิสก์ ไม่ใช่ในหน่วยความจำ (in-memory)
* ดังนั้น Spark มีประสิทธิภาพสูงทั้งบนดิสก์และในหน่วยความจำ โดยจะใช้ดิสก์เมื่อข้อมูลไม่เหมาะสมที่จะใช้ในหน่วยความจำ

**Scalable**

* Apache Spark รองรับการประมวลผลข้อมูลแบบขนาน (parallelised data processing)
* ช่วยให้สามารถประมวลผลงานที่แตกต่างกันบนชุดข้อมูลได้
* มีการกระจายภาระงานจากหลายโหนดในคลัสเตอร์
* สามารถเพิ่มโหนดเพิ่มเติมลงในคลัสเตอร์ได้
* Apache Spark สามารถทำงานในโหมดท้องถิ่น (local mode)

**Ease of Use**

* Apache Spark ให้บริการเอนจินแบบรวมศูนย์ (unified engine)
* Apache Spark รองรับการใช้งานที่หลากหลาย เช่น การประมวลผลแบบแบทช์ (batch) และการสตรีมมิ่ง (streaming)
* Apache Spark ยังรองรับการเรียนรู้ของเครื่อง (machine learning) และการคำนวณกราฟ (graph computation)

**Fault Tolerance**

* Apache Spark มีความทนทานต่อความผิดพลาด (fault tolerance) ซึ่งหมายความว่าระบบสามารถทำงานต่อไปได้เมื่อเกิดความล้มเหลวและสามารถกู้คืนข้อมูลหลังจากความล้มเหลวเกิดขึ้น
* Apache Spark มี Resilient Distributed Dataset (RDD) ซึ่งเป็นชุดข้อมูลที่ไม่สามารถเปลี่ยนแปลงได้และเป็นบล็อกพื้นฐานของโครงสร้างข้อมูลใน Apache Spark
  + Resilient: สามารถคำนวณส่วนที่สูญหายใหม่ได้เมื่อเกิดความผิดพลาด (fault-tolerance)
  + Distributed: ประมวลผลแบบขนาน
  + Dataset: ชุดข้อมูลที่ได้รับการประมวลผล

**Apache Spark Architecture**

**Spark Application**

* Spark Application สามารถรันได้ทั้งบนเครื่องเดียวหรือบนคลัสเตอร์
* ประกอบด้วย Driver, Cluster Manager, และ Executor
* ในโหมดคลัสเตอร์มีสถาปัตยกรรมแบบ master/slave

**How Apache Spark Works?**

1. Client ส่งแอปพลิเคชันใน Spark ไปยัง Driver (master node)
2. Driver สร้าง SparkSession (Spark 2.0) หรือ SparkContext (Spark 1.0) เพื่อเริ่มแอปพลิเคชัน
3. Driver ติดต่อกับ Executor (slave node) เพื่อจัดสรรทรัพยากรที่ต้องการสำหรับแอปพลิเคชันและรันโค้ดแอปพลิเคชัน
4. Driver ยังติดต่อกับ Cluster Manager เพื่อจัดการการจัดตารางและติดตามทรัพยากร

ใน Apache Spark, **Driver** เป็นส่วนสำคัญที่ทำหน้าที่ควบคุมการทำงานของแอปพลิเคชัน Spark:

* **Driver** เป็นกระบวนการหลักที่รันแอปพลิเคชัน Spark และทำหน้าที่ในการประสานงานการทำงานของแอปพลิเคชันทั้งหมด
* **Driver** สร้างและจัดการ **SparkSession** (Spark 2.0) หรือ **SparkContext** (Spark 1.0) ซึ่งเป็นจุดเริ่มต้นของการประมวลผล
* **Driver** ติดต่อกับ **Cluster Manager** เพื่อขอทรัพยากรจาก **Executor** (slave nodes) และจัดการการสั่งงาน
* **Driver** ทำหน้าที่ในการควบคุมการจัดตารางและติดตามสถานะของ **Executor** และกระบวนการต่าง ๆ ที่กำลังทำงานอยู่

โดยรวมแล้ว, **Driver** เป็นศูนย์กลางการควบคุมที่ทำให้การทำงานของแอปพลิเคชัน Spark เป็นไปอย่างราบรื่นและมีประสิทธิภาพ

**Apache Spark Ecosystem**

**Spark Core**

* Spark Core เป็นหัวใจหลักของ Apache Spark ซึ่งรองรับการคอมพิวต์ในหน่วยความจำ (in-memory computing), การรองรับความผิดพลาด (fault-tolerance), และการคอมพิวต์แบบขนาน (parallel computing).
* ในระดับต่ำ, Spark Core ทำงานกับ RDDs และ cluster manager.
* ในระดับสูง, Spark Core รองรับไลบรารีต่างๆ เช่น Spark SQL, Streaming, MLlib, GraphX เป็นต้น

**Spark API**

* Spark รองรับหลาย API
* ภาษาโปรแกรมที่รองรับ ได้แก่ SQL, Scala, Java, Python, และ R.
* ดังนั้น, การทำงานกับการพัฒนาในรูปแบบต่างๆ จึงทำได้ง่าย

**Spark SQL, DataFrames, Dataset**

* Spark SQL อนุญาตให้ผู้ใช้ทำการสอบถามข้อมูลที่มีโครงสร้าง
* Spark ให้การทำงานกับข้อมูลผ่านการอับสตรักชันที่เรียกว่า DataFrames
* Spark DataFrames มี Spark Schema ซึ่งประกอบด้วยชื่อคอลัมน์, ประเภทข้อมูล, และคุณสมบัติการอนุญาตให้เป็นค่า null
* Datasets ใน Spark เป็น immutable (ไม่สามารถเปลี่ยนแปลงได้)
* Datasets อนุญาตให้ผู้ใช้ทำการตรวจสอบข้อผิดพลาดก่อนการรันแอปพลิเคชัน
* Datasets ใช้งานได้เฉพาะกับ API ของ Java และ Scala เท่านั้น

**Streaming, MLlib, GraphX**

* Streaming - ทำการคอมพิวต์ข้อมูลแบบสตรีมในข้อมูลสตรีม
* MLlib - ไลบรารีสำหรับการเรียนรู้ของเครื่อง (machine learning) ที่ใช้ในการวิเคราะห์ข้อมูล
* GraphX – รองรับการคอมพิวต์และการวิเคราะห์กราฟ

**Relational Database Management System**

* ปัญหาหลักในการใช้ RDBMS สำหรับ Big Data คือการขยายขนาด (scalability)
* RDBMS ถูกออกแบบมาให้ใช้งานบนเซิร์ฟเวอร์เดียว ซึ่งสามารถปรับปรุงได้แต่ยังมีข้อจำกัด เรียกว่า vertical scaling
* RDBMS มีคุณสมบัติ ACID ซึ่งประกอบด้วย:
  + **Atomicity** – การทำธุรกรรมที่ไม่สมบูรณ์จะไม่สามารถอัพเดตฐานข้อมูลได้
  + **Consistency** – ข้อมูลจะต้องมีความสอดคล้องก่อนและหลังการทำธุรกรรม
  + **Isolation** – การทำธุรกรรมแต่ละรายการจะไม่แทรกแซงกัน
  + **Durability** – ฐานข้อมูลต้องได้รับการอัพเดตเพื่อให้แน่ใจว่าข้อมูลจะไม่สูญหายหากระบบล้มเหลว

**Not only SQL (NoSQL)**

* Non-relational database
* NoSQL มีคุณสมบัติ BASE ซึ่งประกอบด้วย:
  + Basically Available – ระบบสามารถใช้งานได้ตลอดเวลาแม้เมื่อเกิดการล้มเหลวของเครือข่าย
  + Soft state – หมายความว่ามีความยืดหยุ่นในความต้องการด้านความสอดคล้อง
  + Eventually consistent – ระบบจะมีความสอดคล้องในที่สุด
* NoSQL ใช้การขยายขนาดในแนวนอน (horizontal scaling) ซึ่งช่วยให้สามารถเพิ่มข้อมูลมากขึ้นในฐานข้อมูลได้
* การทำงานกับ NoSQL ต้องคำนึงถึง CAP theorem

**RDBMS VS NoSQL**

* Schema Flexibility: RDBMS is inflexible, often requiring the creation of new tables, whereas NoSQL is column-oriented, allowing for the addition of more columns and supporting semi-structured data.
* Complex Query: RDBMS often involves complex JOIN queries that can be difficult to implement and maintain. In contrast, NoSQL does not support relationships and foreign keys, thus avoiding complex queries.
* Data Update: In RDBMS, if the system does not support updating multiple nodes simultaneously, there is a risk of node failure. NoSQL faces challenges in synchronization across nodes, but it provides synchronization options.
* Scalability: RDBMS has lower speed for handling large amounts of data, while NoSQL offers great scalability.
* ความยืดหยุ่นของสกีมา: RDBMS มีความ inflexible ซึ่งมักจะต้องสร้างตารางใหม่ ในขณะที่ NoSQL เป็น column-oriented ซึ่งช่วยให้สามารถเพิ่มคอลัมน์เพิ่มเติมและสนับสนุนข้อมูลแบบกึ่งโครงสร้าง (semi-structured data) ได้
* การค้นหาที่ซับซ้อน: RDBMS มักใช้ JOIN queries ที่ซับซ้อนซึ่งอาจยากต่อการใช้งานและบำรุงรักษา ขณะที่ NoSQL ไม่สนับสนุนความสัมพันธ์และคีย์ต่าง ๆ ดังนั้นจึงไม่มีการค้นหาที่ซับซ้อน
* การอัปเดตข้อมูล: ใน RDBMS หากระบบไม่สามารถอัปเดตหลายโหนดพร้อมกันได้ อาจเกิดความเสี่ยงของการล้มเหลวของโหนด ในขณะที่ NoSQL เผชิญกับความท้าทายในการซิงค์ข้อมูลข้ามโหนด แต่มีตัวเลือกในการซิงค์ข้อมูล
* การขยายขนาด: RDBMS มีความเร็วที่ต่ำเมื่อจัดการกับข้อมูลจำนวนมาก ในขณะที่ NoSQL มีการขยายขนาดที่ดีมาก

**NoSQL**

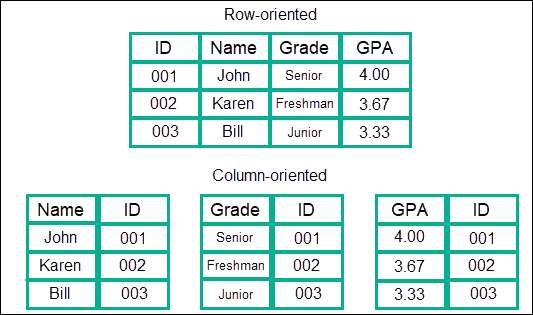
**Key-Value Databases**

* มันเป็นฐานข้อมูล NoSQL ที่ง่ายที่สุด
* มันเก็บข้อมูลในรูปแบบของคู่คีย์-ค่า (key-value pairs)
* คีย์ (key) จะต้องไม่ซ้ำกันสำหรับข้อมูลแต่ละชุด
* คีย์มักเป็นสตริง (string) หรือจำนวนเต็ม (integer)
* ค่าหมายถึงข้อมูลที่สามารถเป็นรูปแบบของแอตทริบิวต์ (attributes) หรือคอลเลคชัน (collections)
* ค่าสามารถเป็นข้อมูลได้ทุกรูปแบบ
* ข้อดี:
  + ความสามารถในการขยายตัว (Scalability): ใช้การขยายแบบแนวนอนผ่านการแบ่งพาร์ทิชันและการทำสำเนาข้อมูล (replication) มีค่าใช้จ่ายต่ำ
  + ความยืดหยุ่นในการเคลื่อนย้าย (Mobility): ง่ายต่อการย้ายจากระบบหนึ่งไปยังอีกระบบหนึ่งโดยไม่ต้องเปลี่ยนแปลงโค้ดหรือสถาปัตยกรรม
* ข้อเสีย:
  + การทำการเชื่อมโยงทั้งหมดต้องทำในโค้ด (All joins must be done in code)
  + ไม่มีตัวกรองคำถามที่ซับซ้อน (No complex query filters)

**Document Databases**

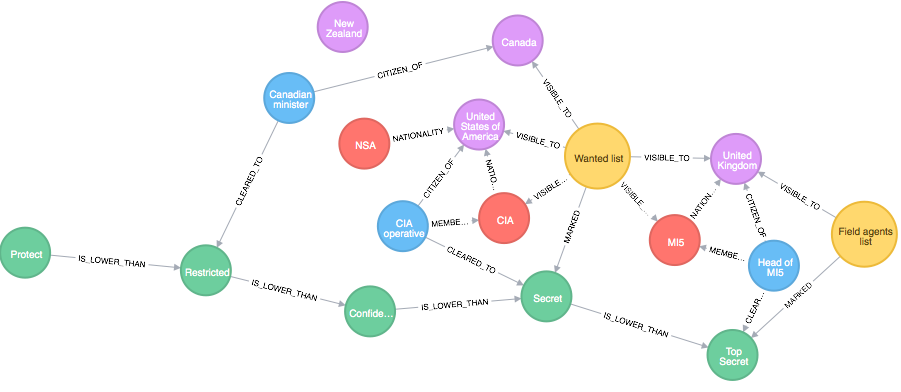
* ลักษณะคล้ายกับฐานข้อมูลแบบคีย์-ค่า (key-value databases) โดยที่แต่ละเอกสาร (document) มีคีย์ (ID) ที่ไม่ซ้ำกัน
* เอกสารแต่ละชิ้นสามารถเก็บข้อมูลในรูปแบบใดก็ได้
* การสืบค้นใช้เอกสารที่คล้าย JSON ซึ่งต้องการรูปแบบข้อมูลที่ฐานข้อมูลสามารถเข้าใจได้
* ข้อดี:
  + รวบรวมข้อมูลจาก RAM ซึ่งสามารถเข้าถึงได้เร็ว
  + สามารถขยายแนวนอนได้
* ข้อเสีย:
  + การเลือกข้อมูลจากหลายคอลเลกชันต้องใช้หลายคำสั่งค้นหา
  + การซ้ำซ้อนของข้อมูลอาจเกิดขึ้น ซึ่งทำให้การจัดการยาก

**Column-Oriented Databases**



* มันเก็บข้อมูลในรูปแบบของคอลัมน์
* คอลัมน์ - ประกอบด้วยชื่อ, ค่า, และเวลา
* แถว - ประกอบด้วยคอลัมน์หนึ่งหรือมากกว่า โดยแถวต่างๆ ไม่จำเป็นต้องมีจำนวนคอลัมน์เท่ากัน มี row-key เป็นคีย์เฉพาะ (ID)
* Column family - ประกอบด้วยหลายแถว ซึ่งแต่ละแถวสามารถมีหลาย column families
* Keyspace - ประกอบด้วยหลาย column families
* ข้อดี:
  + มีความสามารถในการขยายตัวและยืดหยุ่น
  + เวลาการโหลดและการรวมข้อมูลรวดเร็วมาก
* ข้อเสีย:
  + ช้าเมื่อทำการลบแถว
  + อาจช้าเมื่อคิวรีข้อมูลด้วยคำสั่ง join query

**Graph Databases**



* มันเก็บข้อมูลที่มีโครงสร้างกราฟ
* มันแสดงความสัมพันธ์ระหว่างข้อมูล
* โนด (Nodes) มีชุดของคุณสมบัติ
* ขอบ (Edges หรือ Links) ก็มีชุดของคุณสมบัติเช่นกัน
* ข้อดี:
  + เข้าใจข้อมูลได้ง่ายและมีคิวรีที่มีลักษณะบรรยาย
  + ยืดหยุ่น
* ข้อเสีย:
  + ยากต่อการขยาย
  + ไม่มีภาษามาตรฐาน

**MongoDB** เป็นฐานข้อมูล NoSQL ที่จัดเก็บข้อมูลในรูปแบบเอกสาร JSON (หรือ BSON) ซึ่งช่วยให้การจัดการข้อมูลที่มีโครงสร้างยืดหยุ่น และรองรับการขยายตัวได้ดี โดยไม่ต้องใช้โครงสร้างตารางที่ตายตัวเหมือนฐานข้อมูลเชิงสัมพันธ์ (RDBMS).

**Miniconda** เป็นเครื่องมือจัดการแพ็กเกจและสภาพแวดล้อมสำหรับการพัฒนาโปรแกรมที่ใช้ภาษา Python หรือ R โดยเป็นเวอร์ชันที่เบากว่า Anaconda ซึ่งเป็นเครื่องมือจัดการแพ็กเกจที่นิยมใช้สำหรับการวิเคราะห์ข้อมูลและวิทยาศาสตร์ข้อมูล Miniconda ช่วยให้คุณติดตั้งและจัดการแพ็กเกจต่าง ๆ และสร้างสภาพแวดล้อมแยกต่างหากได้อย่างง่ายดาย

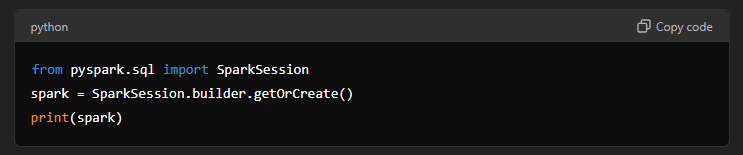
PySpark คือไลบรารี Python สำหรับ Apache Spark ซึ่งเป็นเฟรมเวิร์กการประมวลผลข้อมูลแบบกระจาย (distributed data processing) ที่ช่วยให้สามารถจัดการและวิเคราะห์ข้อมูลขนาดใหญ่ได้อย่างมีประสิทธิภาพ โดย PySpark ทำให้การใช้ Spark เป็นไปได้ด้วยโค้ด Python ซึ่งสะดวกสำหรับนักพัฒนาและนักวิเคราะห์ข้อมูล.

**Batch Processing**

* Batch processing คือวิธีการคอมพิวเตอร์สำหรับการประมวลผลข้อมูลจำนวนมากในรูปแบบ batch
* กระบวนการ batch ไม่ต้องการการโต้ตอบกับผู้ใช้หลังจากที่กระบวนการเริ่มต้นแล้ว
* มันเหมาะสำหรับการจัดการในช่วงท้ายของรอบการทำงาน เช่น การอัพเดตสต็อกในตอนท้ายของวัน
* อย่างไรก็ตาม เมื่อเกิดข้อผิดพลาดใดๆ ขึ้น, กระบวนการ batch อาจถูกปิดการทำงานได้

**Spark Application**

* คือโปรแกรมที่พัฒนาขึ้นโดยผู้ใช้บน Spark โดยใช้ Spark APIs.
* SparkSession ทำหน้าที่เชื่อมต่อ Spark กับผู้ใช้ผ่าน Spark APIs.
* Task คือหน่วยการประมวลผลที่เล็กที่สุดซึ่งจะถูกดำเนินการใน executor.
* Stage คือการรวมกลุ่มของ tasks ที่ทำการรันโค้ดเดียวกันบนข้อมูลที่แตกต่างกันใน dataset.
* Job ประกอบด้วยหลาย stages และช่วยให้สามารถดำเนินการโปรแกรมใน Spark ได้
* SparkSession ใช้ใน Spark 2.0 ในขณะที่ SparkContext ใช้ในเวอร์ชันก่อนหน้า
* SparkSession สร้างขึ้นโดยใช้ฟังก์ชัน builder: SparkSession.builder()
* เพื่อดึงการเชื่อมต่อ session ที่มีอยู่แล้ว ใช้ฟังก์ชัน getOrCreate().



**Spark Application**

**Transformations**

* Transformations ใช้ Resilient Distributed Dataset (RDD) หรือ DataFrame เป็นข้อมูลนำเข้าและคืนค่าเป็น RDD หรือ DataFrame เป็นผลลัพธ์
* ลักษณะเด่น:
  + Immutable - รักษาสำเนาดั้งเดิมของข้อมูล
  + Not executed immediately - จะไม่ถูกดำเนินการทันที แต่จะถูกบันทึกและสร้าง transformation lineage ซึ่งเป็นลำดับของการดำเนินการที่บันทึกใน Directed Acyclic Graph (DAG) ซึ่งจะถูกดำเนินการเมื่อเรียกใช้งาน action นี่เรียกว่า Lazy Evaluation
* สองประเภทของ transformations:
  + Narrow transformations - การดำเนินการที่ไม่มีการ data shuffling หรือกล่าวอีกนัยหนึ่งคือ หนึ่ง data partition ผลลัพธ์ในหนึ่ง output partition
    - เช่น map(), mapPartition(), filter(), union()
  + Wide transformations - การดำเนินการที่เกี่ยวข้องกับ data shuffling หรือกล่าวอีกนัยหนึ่งคือ หนึ่ง data partition ผลลัพธ์ในหลาย output partitions
    - เช่น join(), distinct(), aggregate(), repartition(), intersect()

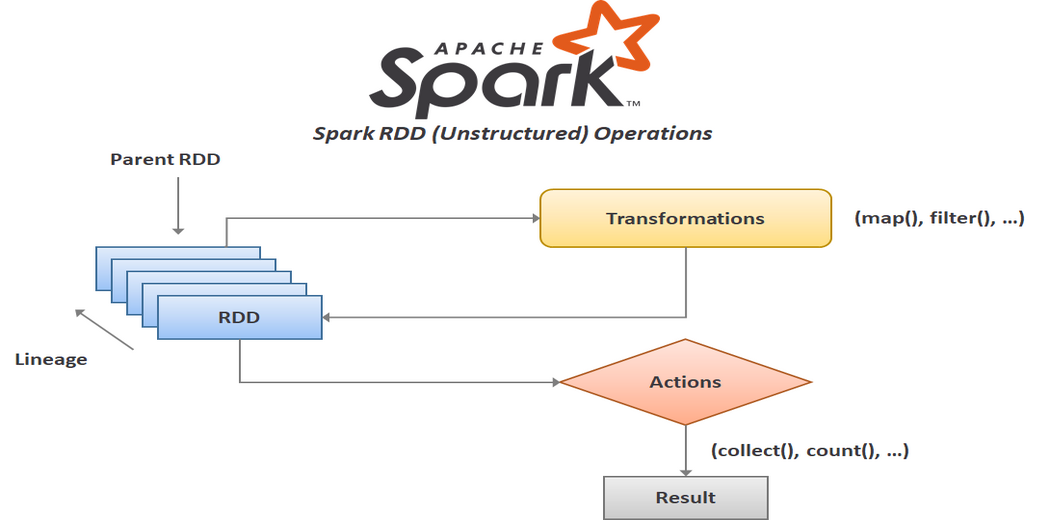
**Actions**

* Spark operations คืนค่าเป็น single value หนึ่งค่า
* การดำเนินการเหล่านี้จะกระตุ้นการทำงานของ transformations
* ฟังก์ชันตัวอย่าง: collect(), count(), min(), max(), top(), เป็นต้น

**Resilient Distributed Datasets (RDDs)**

คือโครงสร้างข้อมูลพื้นฐานใน Apache Spark ที่จัดเก็บข้อมูลในรูปแบบกระจายบนหลายโหนดและสามารถทำงานร่วมกันได้ในลักษณะขนาน โดย RDD มีคุณสมบัติที่สำคัญดังนี้:

* Resilient: สามารถจัดการกับข้อผิดพลาดได้ โดยการเก็บข้อมูลสำรอง
* Distributed: ข้อมูลถูกกระจายอยู่บนหลายโหนดในคลัสเตอร์
* Immutable: ข้อมูลไม่สามารถเปลี่ยนแปลงได้หลังจากถูกสร้าง
* Lazy Evaluation: การดำเนินการต่างๆ จะถูกประมวลผลก็ต่อเมื่อมีการเรียกใช้งาน action



* Datasets ถูกแบ่งออกเป็น logical partitions
* Partitions เหล่านี้จะถูกประมวลผลในแบบขนานข้าม nodes ที่แตกต่างกัน
* RDDs สามารถถูกสร้างจาก collections ที่มีอยู่แล้วหรือจาก external datasets เช่น ไฟล์ text, CSV, และ JSON

**ทำไมต้องเเบ่ง partitions**

การแบ่งข้อมูลออกเป็น **partitions** ใน Apache Spark มีความสำคัญเพื่อให้สามารถทำงานกับข้อมูลขนาดใหญ่ได้อย่างมีประสิทธิภาพ โดยการทำงานแบบขนาน (parallel processing) ซึ่งช่วยให้การประมวลผลเร็วขึ้นและมีประสิทธิภาพมากขึ้น

**ทำไมต้องแบ่งข้อมูลเป็น Partitions?**

1. **การประมวลผลขนาน**: การแบ่งข้อมูลเป็น **partitions** ช่วยให้ข้อมูลสามารถประมวลผลพร้อมกันในหลาย **nodes** หรือ **cores** ของคลัสเตอร์ได้ ซึ่งช่วยให้การประมวลผลเร็วขึ้น
2. **ความทนทานต่อข้อผิดพลาด**: การแบ่งข้อมูลเป็น **partitions** ช่วยให้ระบบสามารถจัดการกับข้อผิดพลาดได้ดีขึ้น เนื่องจากถ้า **partition** หนึ่งเกิดข้อผิดพลาด ระบบสามารถดำเนินการต่อไปกับ **partitions** ที่เหลือได้
3. **การจัดการข้อมูลขนาดใหญ่**: เมื่อข้อมูลมีขนาดใหญ่เกินกว่าที่จะจัดเก็บในหน่วยความจำของเครื่องเดียว การแบ่งข้อมูลเป็น **partitions** ช่วยให้สามารถจัดเก็บและประมวลผลข้อมูลได้โดยไม่ต้องมีหน่วยความจำใหญ่เกินไป

**หน้าตาของ Partitions**

เมื่อข้อมูลถูกแบ่งออกเป็น **partitions**:

* **Logical Partitions**: ข้อมูลจะถูกจัดระเบียบในหน่วยที่เรียกว่า **partitions** ซึ่งเป็นหน่วยข้อมูลที่ถูกจัดเก็บและประมวลผลแยกกันในคลัสเตอร์
* **Physical Partitions**: บนระบบไฟล์หรือหน่วยความจำ ข้อมูลอาจถูกจัดเก็บในรูปแบบไฟล์หรือโฟลเดอร์ที่แบ่งตาม **partitions**

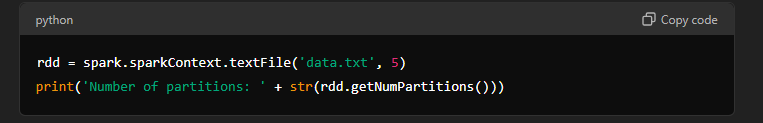
**ตัวอย่าง**: สมมุติว่าคุณมีข้อมูล 100,000 แถว และคุณต้องการแบ่งข้อมูลออกเป็น 5 **partitions**:

* **Partition 1**: จะมีข้อมูลประมาณ 20,000 แถว
* **Partition 2**: จะมีข้อมูลประมาณ 20,000 แถว
* **Partition 3**: จะมีข้อมูลประมาณ 20,000 แถว
* **Partition 4**: จะมีข้อมูลประมาณ 20,000 แถว
* **Partition 5**: จะมีข้อมูลประมาณ 20,000 แถว

ในระบบจริง ข้อมูลในแต่ละ **partition** อาจถูกจัดเก็บในไฟล์หรือโฟลเดอร์ที่แตกต่างกัน และการประมวลผลของแต่ละ **partition** จะถูกจัดการโดยโหนดหรือคอร์ที่แตกต่างกันในคลัสเตอร์

**การดูการแบ่ง Partitions**

คุณสามารถตรวจสอบจำนวน **partitions** ของ **RDD** ได้ด้วยฟังก์ชัน getNumPartitions() ใน PySpark:



ในกรณีนี้, ข้อมูลจากไฟล์ data.txt จะถูกแบ่งออกเป็น 5 **partitions** และคุณสามารถตรวจสอบจำนวน **partitions** ได้ด้วย rdd.getNumPartitions()

**RDDs from Parallelised Collections**

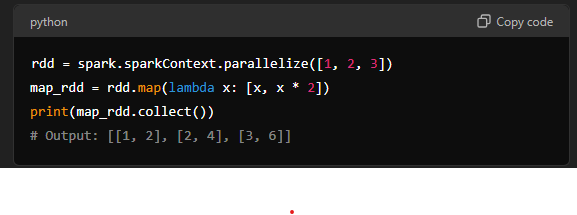
* A collection of elements ถูกเปลี่ยนเป็น distributed datasets
* Datasets เหล่านี้จะถูกประมวลผลในแบบขนาน
* RDDs ที่สร้างจาก collection of elements โดยใช้ sparkContext.parallelize()
* Number of partitions ถูกกำหนดโดยอัตโนมัติตามจำนวน nodes ที่มีอยู่
* อย่างไรก็ตาม, number of partitions สามารถกำหนดโดยผู้ใช้ได้
* textFile(): คืนค่าบันทึก (record) ต่อหนึ่งบรรทัด
* wholeTextFiles(): คืนค่าพาธของไฟล์และเนื้อหาไฟล์ในรูปแบบของคู่คีย์-ค่า (key-value pair) โดยที่แต่ละไฟล์จะกลายเป็นบันทึกเดียว
* อย่างไรก็ตาม, wholeTextFiles() เหมาะสำหรับการประมวลผลไฟล์ขนาดเล็ก

**Transformation Functions**

* **Distinct** ใช้ในการลบบันทึกที่ซ้ำกันออกจาก **RDD** แล้วนับจำนวนบันทึกที่ไม่ซ้ำกัน จากนั้นพิมพ์จำนวนบันทึกที่ไม่ซ้ำกันออกมา
* **Filter** ใช้ในการเลือกบันทึกที่ตรงตามเงื่อนไขที่กำหนด
* Map ใช้ในการสร้างคู่คีย์-ค่า (key-value pairs):
  + ใช้ในการเปลี่ยนข้อมูลใน RDD โดยการใช้ฟังก์ชันที่แปลงแต่ละบันทึกเป็นคู่คีย์-ค่า
* flatMap เป็นการขยายจาก Map:
  + ใช้เมื่อแต่ละข้อมูลนำเข้าควรจะส่งออกหลายผลลัพธ์
  + แตกต่างจาก Map ตรงที่ flatMap สามารถส่งออกหลายบันทึกจากการแปลงข้อมูลหนึ่งบันทึก

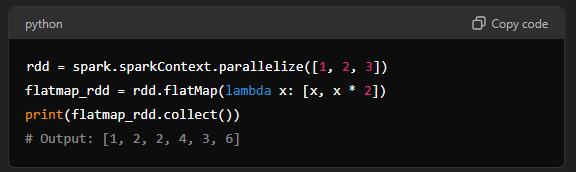
1. map จะใช้ฟังก์ชันที่ระบุเพื่อแปลงแต่ละองค์ประกอบใน RDD และจะส่งออกองค์ประกอบหนึ่งต่อหนึ่ง

ฟังก์ชันที่ใช้กับ map จะต้องส่งคืนค่าเพียงหนึ่งค่า (หนึ่งองค์ประกอบ) สำหรับแต่ละองค์ประกอบใน RDD เดิม



1. flatMap จะใช้ฟังก์ชันที่ระบุเพื่อแปลงแต่ละองค์ประกอบใน RDD และจะส่งออกองค์ประกอบหลายๆ ค่า (หลายองค์ประกอบ) ต่อหนึ่งองค์ประกอบใน RDD เดิม

ฟังก์ชันที่ใช้กับ flatMap จะต้องส่งคืนค่าเป็นลิสต์ (หรืออิเทอเรเตอร์) ของหลายๆ ค่า และ flatMap จะ "flatten" (ทำให้แบน) ค่าที่ส่งคืนเหล่านั้นลงใน RDD เดียว



* Sort ใช้ในการเรียงค่าต่างๆ ใน RDD
* sortByKey() ใช้ในการเรียงลำดับ pair RDD ตามคีย์

ใช้สำหรับ pair RDD (คู่คีย์-ค่า) เพื่อเรียงลำดับตามค่าของคีย์

* sortBy() เป็นฟังก์ชันการเรียงลำดับอีกฟังก์ชันหนึ่ง
* มันรับอาร์กิวเมนต์ 3 อย่าง ได้แก่:
  + key: ฟังก์ชันที่ระบุว่าค่าที่จะใช้ในการเรียงลำดับคืออะไร
  + flag: (True/False) สำหรับการเรียงลำดับตามลำดับจากน้อยไปหามาก (ascending) หรือจากมากไปหาน้อย (descending)
  + number of partitions: จำนวนพาร์ติชันที่ต้องการ
* sortByKey(): เรียงลำดับ pair RDD ตามคีย์
* sortBy(): เรียงลำดับ RDD โดยใช้ฟังก์ชันกำหนดคีย์
* reduceByKey(): รวมค่าของแต่ละคีย์โดยใช้ฟังก์ชัน reduce
* เป็นการแปลงแบบ wide ที่กระจายข้อมูลระหว่างพาร์ติชันทั้งหมด
* aggregateByKey(): รวมค่าตามคีย์ที่ระดับพาร์ติชัน
* ผลลัพธ์จากพาร์ติชันจะถูกรวมกันข้ามโหนด
* ในบริบทของ Spark:
  + โหนด (Node) คือเครื่องคอมพิวเตอร์หรือเซิร์ฟเวอร์ที่เป็นส่วนหนึ่งของคลัสเตอร์ที่ใช้ในการประมวลผลข้อมูล
  + โหนดแต่ละตัวสามารถทำหน้าที่เป็น Master Node (จัดการการทำงานของคลัสเตอร์) หรือ Worker Node (ทำการประมวลผลข้อมูล)
* foldByKey(): รวมค่าตามคีย์
* เริ่มต้นค่าเริ่มต้นสำหรับแต่ละคีย์และพาร์ติชัน ซึ่งค่าที่เริ่มต้นจะไม่ส่งผลกระทบต่อผลลัพธ์สุดท้าย
* combineByKey(): ต้องใช้การกระจายข้อมูล (shuffle) ในขั้นตอนสุดท้าย
* แปลงคู่คีย์-ค่าเป็นคีย์-ค่าที่รวมกัน
* groupByKey(): กลุ่มค่าตามคีย์
* คู่คีย์-ค่าจะถูกกระจาย (shuffled) ข้ามโหนดทั้งหมด
* ไม่เหมาะสำหรับชุดข้อมูลขนาดใหญ่
* join(): คืนค่าคู่ที่มีคีย์ที่ตรงกันระหว่างคู่คีย์-ค่า
* leftOuterJoin(): คืนค่าคู่ที่มีคีย์จาก RDD ด้านซ้ายทั้งหมด
* rightOuterJoin(): คืนค่าคู่ที่มีคีย์จาก RDD ด้านขวาทั้งหมด

**Action Functions**

* countByKey(): นับจำนวนค่าต่อแต่ละคีย์ คืนค่าเป็น DefaultDict ซึ่งเป็นวัตถุของพจนานุกรม (dictionary)
* countByValue**()**: นับค่าที่ไม่ซ้ำกันและคืนค่าเป็นพจนานุกรม (dictionary) ของคู่ค่า-จำนวน
* collectAsMap(): คืนค่า RDD ที่มีคู่คีย์-ค่าเป็นพจนานุกรม (dictionary)
* lookup(key): คืนค่าทั้งหมดที่ตรงกับคีย์ในรูปแบบของลิสต์
* first() returns the first value in the dataset.
* max(): คืนค่าสูงสุดในชุดข้อมูล
* min(): คืนค่าต่ำสุดในชุดข้อมูล
* saveAsTextFile(path): บันทึก RDD เป็นไฟล์ข้อความที่เส้นทางที่ระบุ

**High Level**

**Structured API Execution**

ขั้นตอน

1. เขียนโค้ด DataFrame/SQL
2. Spark จะแปลงโค้ดเป็นแผนเชิงตรรกะ (Logical Plan) หากโค้ดนั้นถูกต้อง
3. Spark จะแปลงแผนเชิงตรรกะเป็นแผนเชิงกายภาพ (Physical Plan)
4. Spark จะดำเนินการแผนเชิงกายภาพบนคลัสเตอร์

คลัสเตอร์ (Cluster) หมายถึงกลุ่มของเซิร์ฟเวอร์หรือคอมพิวเตอร์ที่เชื่อมต่อกันและทำงานร่วมกันเพื่อประมวลผลข้อมูลหรือดำเนินการคอมพิวเตอร์ต่างๆ คลัสเตอร์ถูกใช้เพื่อเพิ่มประสิทธิภาพและความสามารถในการจัดการข้อมูลขนาดใหญ่ โดยเฉพาะในการประมวลผลข้อมูลแบบกระจาย (Distributed Processing) เช่นใน Apache Spark

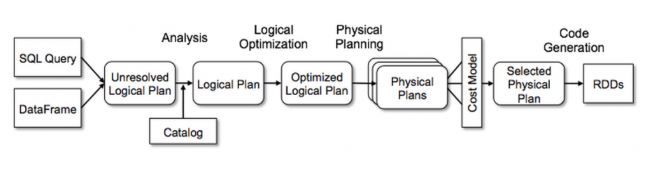
ในกรณีของ Apache Spark คลัสเตอร์จะประกอบด้วย:

* **Master Node**: เซิร์ฟเวอร์หลักที่จัดการงานและแจกจ่ายการประมวลผลให้กับ Worker Nodes
* **Worker Nodes**: เซิร์ฟเวอร์ที่ทำงานจริงในการประมวลผลข้อมูลและจัดเก็บผลลัพธ์

การทำงานร่วมกันของเครื่องในคลัสเตอร์ช่วยให้สามารถจัดการกับข้อมูลขนาดใหญ่ได้อย่างมีประสิทธิภาพและรวดเร็ว

**Logical Plan**

1. โค้ดจะถูกแปลงเป็นแผนเชิงตรรกะที่ยังไม่ได้รับการแก้ไข (Unresolved Logical Plan) ก่อน
2. Spark จะใช้ “แคตตาล็อก” เพื่อแก้ไขตารางและคอลัมน์ในตัววิเคราะห์ (Analyser)
3. ตัววิเคราะห์จะปฏิเสธแผนเชิงตรรกะที่ยังไม่ได้รับการแก้ไขหากตารางที่อ้างอิงไม่มีอยู่จริง มิฉะนั้นจะรับรองให้กลายเป็นแผนเชิงตรรกะที่ได้รับการแก้ไข (Resolved Logical Plan)
4. Spark จะส่งแผนเชิงตรรกะที่ได้รับการแก้ไขไปยังตัวปรับแต่ง (Catalyst Optimiser) เพื่อปรับแต่งแผนเชิงตรรกะที่ได้รับการแก้ไขให้เป็น “แผนเชิงตรรกะที่ได้รับการปรับแต่ง” (Optimised Logical Plan)



**Physical Plan**

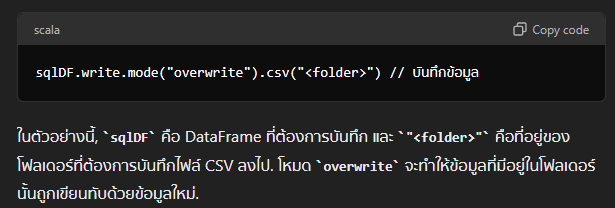
* แผนเชิงกายภาพ (Physical Plan) หรือที่เรียกว่า Spark plan จะสร้างการดำเนินการทางกายภาพที่แตกต่างกันและเปรียบเทียบพวกมันผ่านแบบจำลองต้นทุน (Cost Model)
* แผนเชิงกายภาพสามารถจัดการการดำเนินการของแผนเชิงตรรกะที่ได้รับการปรับแต่งบนคลัสเตอร์
* ผลลัพธ์ของแผนเชิงกายภาพคือ RDDs (Resilient Distributed Datasets) และการเปลี่ยนแปลงของมัน กล่าวคือ แผนเชิงกายภาพจะดำเนินการโค้ดบน RDDs

**Dataframes**

* DataFrames ช่วยให้ Spark ดำเนินการการจัดกลุ่มข้อมูล (shuffling) ได้อย่างมีประสิทธิภาพโดยการย้ายข้อมูลระหว่างโหนด
* DataFrame แทนตารางข้อมูลที่มีแถวและคอลัมน์
* สคีมา (Schema) ประกอบด้วยชื่อคอลัมน์, ประเภทข้อมูล, และธงบ่งบอกว่าเป็นค่า NULL ได้หรือไม่
* DataFrames สามารถกระจายไปทั่วโหนดเพื่อรองรับสถาปัตยกรรมการคอมพิวเตอร์แบบกระจาย
* ผู้ใช้สามารถสืบค้นข้อมูลโดยใช้ SQL ซึ่งจะถูกส่งไปยัง DataFrames และจัดการโดย Catalyst Optimiser
* Catalyst Optimiser มีหน้าที่ในการสร้างการดำเนินการสอบถาม
* DataFrames สามารถสร้างจากแหล่งข้อมูลภายนอก เช่น ไฟล์ CSV และ JSON

Temporary Views ใน Apache Spark คือการสร้างมุมมองชั่วคราวที่ใช้สำหรับการเขียนคำสั่ง SQL บน DataFrame โดยมุมมองนี้จะถูกเก็บไว้แค่ในระหว่างที่ Spark session ทำงานอยู่ และจะหายไปเมื่อ session จบลง

* ข้อมูลสามารถบันทึกลงในไฟล์ได้
* โหมดการบันทึก (Save Modes) มีดังนี้:
  + errorifexists หรือ error: จะเกิดข้อยกเว้น (exception) หากข้อมูลมีอยู่แล้ว
  + append: ข้อมูลจะถูกเพิ่มลงไปที่ปลายทาง
  + overwrite: ข้อมูลจะถูกเขียนทับหากข้อมูลมีอยู่แล้ว
  + ignore: ข้อมูลจะถูกละเว้นหากข้อมูลมีอยู่แล้ว



* **randomSplit** ใช้สำหรับการแบ่งข้อมูลใน DataFrame
* เป็นเครื่องมือที่มีประโยชน์เมื่อการสร้างชุดข้อมูลฝึก (training set) และชุดข้อมูลทดสอบ (test set) เป็นสิ่งจำเป็น เช่น ในการทำงานกับการเรียนรู้ของเครื่อง (machine learning)

**Aggregations**

คือการรวบรวมข้อมูลเข้าด้วยกันและผลิตผลลัพธ์หนึ่งสำหรับแต่ละกลุ่ม

* count is used for counting rows
* countDistinct` ช่วยในการนับจำนวนค่าที่ไม่ซ้ำกันในคอลัมน์หนึ่งคอลัมน์
* first: ใช้เพื่อดึงแถวแรกของ DataFrame
* last: ใช้เพื่อดึงแถวสุดท้ายของ DataFram
* min: ใช้เพื่อค้นหาค่าต่ำสุดในคอลัมน์ของ DataFrame
* max: ใช้เพื่อค้นหาค่าสูงสุดในคอลัมน์ของ DataFrame
* sum: ใช้สำหรับหาผลรวมทั้งหมดในคอลัมน์ของ DataFrame
* sumDistinct: ใช้สำหรับหาผลรวมของชุดค่าที่ไม่ซ้ำกันในคอลัมน์ของ DataFrame
* avg (average) is used for calculating an average value of a column in a Dataframe.
* Join: รวมข้อมูลจากสองชุดข้อมูล (DataFrames หรือ tables) ตามคีย์ที่กำหนดในชุดข้อมูลซ้าย (left) และชุดข้อมูลขวา (right)
  + Spark: จะทิ้งแถวที่ไม่ตรงกันและคืนแถวที่ตรงกันเท่านั้น
  + Join Expression: ใช้ในการประเมินผลลัพธ์การเชื่อมต่อ
  + **Inner Join**: การเชื่อมต่อที่เก็บแถวที่มีคีย์ตรงกันในทั้งชุดข้อมูลซ้าย (left dataset) และชุดข้อมูลขวา (right dataset)