**Apache spark (Spark)**

Là 1 hệ thống được dùng để xử lý các vấn đề về dữ liệu lớn. Matei Zaharia, cha đẻ của Spark, sử dụng Hadoop từ những ngày đầu. Đến năm 2009 ông viết Apache Spark để giải quyết những bài toán học máy ở đại học UC Berkely vì Hadoop MapReduce hoạt động không hiệu quả cho những bài toán này. Rất sớm sau đó ông nhận ra rằng Spark không chỉ hữu ích cho học máy mà còn cho cả việc xử lý luồng dữ liệu hoàn chỉnh.

**Apache Spark** gồm có 5 thành phần chính : Spark Core, Spark Luồng, Spark SQL, MLlib và GraphX, trong đó:

* **Spark Core** Spark Core đảm nhận vai trò thực hiện công việc tính toán và xử lý trong bộ nhớ (In-memory computing) đồng thời nó cũng tham chiếu các dữ liệu được lưu trữ tại các hệ thống lưu trữ bên ngoài.
* **Spark SQL** cung cấp một kiểu data abstraction mới (SchemaRDD) nhằm hỗ trợ cho cả kiểu dữ liệu có cấu trúc (structured data) và dữ liệu nửa cấu trúc (semi-structured data – thường là dữ liệu dữ liệu có cấu trúc nhưng không đồng nhất và cấu trúc của dữ liệu phụ thuộc vào chính nội dung của dữ liệu ấy).
* **Spark Luồng** được sử dụng để thực hiện việc phân tích luồng bằng việc coi các luồng là các mini-batches và thực hiệc kỹ thuật RDD transformation đối với các dữ liệu mini-batches này. Qua đó cho phép các đoạn code được viết cho xử lý batch có thể được tận dụng lại vào trong việc xử lý luồng, làm cho việc phát triển lambda architecture được dễ dàng hơn. Tuy nhiên điều này lại tạo ra độ trễ trong xử lý dữ liệu (độ trễ chính bằng mini-batch duration)
* **MLlib** (Machine Learning Library): là một nền tảng học máy phân tán bên trên Spark do kiến trúc phân tán dựa trên bộ nhớ.
* **GrapX**: Grapx là nền tảng xử lý đồ thị dựa trên Spark. Nó cung cấp các Api để diễn tả các tính toán trong đồ thị bằng cách sử dụng Pregel Api.

# **Những điểm nổi bật của Spark**

* Xử lý dữ liệu theo lô và thời gian thực
* Có thể tích hợp với tất cả các nguồn dữ liệu và định dạng tệp được hỗ trợ bởi cụm Hadoop.
* Hỗ trợ nhiều ngôn ngữ: Java, Scala, Python và R
* Phân tích thời gian thực tức là dữ liệu đến từ các luồng sự kiện thời gian thực với tốc độ hàng triệu sự kiện mỗi giây.

**Mapreduce**

Mapreduce gồm 2 thủ tục là map và reduce. Được thiết kế độc quyền bởi Google, nó có khả năng lập trình xử lý các tập dữ liệu lớn song song và phân tán thuật toán trên 1 cụm máy tính.

Trong map() thì bao gồm có lọc (filter) và phân loại (sort) trên dữ liệu khi thủ tục Reduce() thực hiện quá trình tổng hợp dữ liệu.

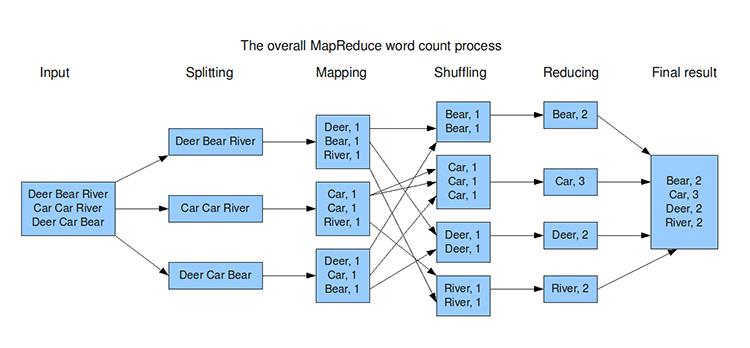
Các thủ tục map() và reduce() được viết bằng nhiều loại ngôn ngữ khác nhau. Được cài đặt miễn phí và sử dụng phổ biên trong Apache hadoop

**Công dụng chính của các hàm là:**

* **Hàm Map():** có nhiệm vụ nhận đầu vào cho các cặp giá trị/ khóa và output chính là tập những cặp giá trị/khóa trung gian. Sau đó, chỉ cần ghi xuống đĩa cứng và tiến hành thông báo cho các hàm Reduce() để trực tiếp nhận dữ liệu.
* **Hàm Reduce():** có nhiệm vụ tiếp nhận từ khóa trung gian và những giá trị tương ứng với lượng từ khóa đó. Sau đó, tiến hành ghép chúng lại để có thể tạo thành một tập khóa khác nhau. Các cặp khóa/giá trị này thường sẽ thông qua một con trỏ vị trí để đưa vào các hàm reduce.
* Ở giữa Map và Reduce thì còn 1 bước trung gian đó chính là **Shuffle**. Sau khi Map hoàn thành  xong công việc của mình thì Shuffle sẽ làm nhiệm vụ chính là thu thập cũng như tổng hợp từ khóa/giá trị trung gian đã được map sinh ra trước đó rồi chuyển qua cho Reduce tiếp tục xử lý.

**Nguyên tắc hoạt động của preduce**

* Phân chia các dữ liệu cần xử lý thành nhiều phần nhỏ.
* Xử lý các vấn đề nhỏ theo phương thức song song trên các máy tính rồi phân tán hoạt động độc lập.
* Tiến hành tổng hợp những kết quả thu được để đề ra được kết quả sau cùng.



## Ưu điểm của MapReduce

* MapReduce có khả năng tác vụ phân tích và tính toán phức tạp. Có thể xử lý nhanh chóng cho ra kết quả dễ dàng chỉ trong khoảng thời gian ngắn.
* Mapreduce có khả năng chạy song song trên các máy có sự phân tán khác nhau.
* MapRedue có khả năng thực hiện trên nhiều nguồn ngôn ngữ lập trình khác nhau

**Các bước hoạt động của MapReduce khi lập trình**

* B1: Tiến hành chuẩn bị các dữ liệu đầu vào
* B2: Thực thi hàm Map() để xử lý dữ liệu.
* B3: Sắp xếp và trộn các kết quả thu từ các máy tính phân tán thích hợp nhất.
* B4: Dùng hàm Reduce() để có thể xử lý tiếp các dữ liệu cần thiết.

Vd về code



# **Spark propertive:** là phương tiện điều chỉnh môi trường thực thi cho các ứng dụng Spark của bạn, kiểm soát hầu hết các cài đặt ứng dụng và cấu hình riêng cho từng ứng dụng.Spark propertive mặc định là $SPARK\_HOME/conf/spark-defaults.conf

**Apache Spark RDD**

##### **Resilient Distributed Datasets (RDD):** là một cấu trúc dữ liệu cơ bản của Spark, là một tập hợp bất biến phân tán của một đối tượng. Mỗi dataset trong RDD được chia ra thành nhiều phần vùng logical. Có thể được tính toán trên các node khác nhau của một cụm máy chủ

RDDs có thể chứa bất kỳ kiểu dữ liệu nào của Python, Java, hoặc đối tượng Scala, bao gồm các kiểu dữ liệu do người dùng định nghĩa.

Có thể tạo ra RDD từ một tập hợp dữ liệu có sẵn trong ngôn ngữ sử dụng như Java, Python, Scala hoặc lấy từ dataset hệ thống lưu trữ bên ngoài như HDFS, Hbase, các cơ sở dữ liệu quan hệ.

**Các loại RDD**

* + # , Apache, Spark , ... , processing (kiểu chuỗi)
  + (for, 12) ; (Spark, 21), ... ,(the, 31) (Kiểu cặp)

Các RDD biểu diễn một tập hợp cố định, đã được phân vùng các record để có thể xử lý song song.

Các record trong RDD có thể là đối tượng Java, Scale hay Python tùy lập trình viên chọn.

RDD API có thể được sử dụng trong Python, Scala hay Java:

**- Một số transformation:**

* **distinct**: loại bỏ trùng lắp trong RDD
* **filter**: tương đương với việc sử dụng ở trong SQL – tìm các record trong RDD xem những phần tử nào thỏa điều kiện. Có thể cung cấp một hàm phức tạp sử dụng để lọc các record cần thiết
* **map**: thực hiện một công việc nào đó trên toàn bộ RDD. Trong Python sử dụng lambda với từng phần tử để truyền vào map
* **flatMap**: cung cấp một hàm đơn giản hơn hàm map. Yêu cầu output của map phải là một cấu trúc có thể lặp và mở rộng được.
* **sortBy**: mô tả một hàm để trích xuất dữ liệu từ các đối tượng của RDD và thực hiện sắp xếp được từ đó.
* **randomSplit**: nhận một mảng trọng số và tạo một random seed, tách các RDD thành một mảng các RDD có số lượng chia theo trọng số.

**- Một số action:**

Action thực thi ngay các transformation đã được thiết lập để thu thập dữ liệu về driver để xử lý hoặc ghi dữ liệu xuống các công cụ lưu trữ.

* **reduce**: thực hiện hàm reduce trên RDD để thu về 1 giá trị duy nhất
* **count**: đếm số dòng trong RDD
* **countApprox:** phiên bản đếm xấp xỉ của count, nhưng phải cung cấp timeout vì có thể không nhận được kết quả.
* **countByValue**: đếm số giá trị của RDD  
  chỉ sử dụng nếu map kết quả nhỏ vì tất cả dữ liệu sẽ được load lên memory của driver để tính toán  
  chỉ nên sử dụng trong tình huống số dòng nhỏ và số lượng item khác nhau cũng nhỏ.
* **countApproxDistinct**: đếm xấp xỉ các giá trị khác nhau
* **countByValueApprox**: đếm xấp xỉ các giá trị
* **first**: lấy giá trị đầu tiên của dataset
* **max và min:** lần lượt lấy giá trị lớn nhất và nhỏ nhất của dataset
* **take và các method tương tự**: lấy một lượng giá trị từ trong RDD. take sẽ trước hết scan qua một partition và sử dụng kết quả để dự đoán số lượng partition cần phải lấy thêm để thỏa mãn số lượng lấy.
* **top và takeOrdered**: top sẽ hiệu quả hơn takeOrdered vì top lấy các giá trị đầu tiên được sắp xếp ngầm trong RDD.
* **takeSamples**: lấy một lượng giá trị ngẫu nhiên trong RDD

**Spark Dataframe**

**DataFrame** là một **API** bậc cao hơn RDD được Spark giới thiệu vào năm 2013 (từ **Apache Spark 1.3**). Tương tự như RDD, dữ liệu trong DataFrame cũng được quản lý theo **kiểu phân tán và không thể thay đổi (immutable distributed)**. Tuy nhiên dữ liệu này được sắp sếp theo các cột, tương tự như trong **Relation Database**.

**DataFrame** được phát triển để giúp người dùng có thể dễ dàng thực hiện các thao tác xử lý dữ liệu cũng như làm tăng đáng kể hiệu quả xử lý của hệ thống.

Khi sử dụng DataFrame API, chúng ta gọi các hàm để trích xuất kết quả mong muốn và Spark sẽ tự động tiến hành các thuật toán xử lý. Tuy nhiên ở bước cuối cùng thì các thuật toán này vẫn được chạy trên RDD mặc dù người dùng chỉ tương tác với DataFrame.

**Ưu điểm của Spark Datafram :**

* Giống như viết SQL, đầy đủ chức năng như select, where ... đặc biệt là join với các DataFrame khác.
* Sử dụng các method như filter, select để trích xuất dữ liệu theo cột, hàng.
* Xử gọn các loại data như Log ... với groupBy
* Thêm 1 cột dễ dàng với UDF(User Defined Function)
* Giống như SQL, Spark DataFrame đã hỗ trợ[**Pivot**](http://oracletechtalk.blogspot.jp/2015/09/pivot-va-unpivot-phan-1.html) (Spark 1.6 trở lên) rất hữu ích cho việc lập bảng biểu, báo cáo.

**sources**

<https://laptrinh.vn/books/apache-spark/page/apache-spark-rdd>

<https://mallikarjuna_g.gitbooks.io/spark/content/spark-properties.html>

<http://itechseeker.com>

<https://codetudau.com/xu-ly-du-lieu-voi-spark-dataframe/index.html>

<https://helpex.vn/article/huong-dan-pyspark-dataframe-gioi-thieu-ve-dataframes-5c6b21e6ae03f628d053c29e>