Tìm hiểu Spark

# Spark properties

Spark properties kiểm soát hầu hết các cài đặt ứng dụng và được cấu hình riêng cho từng ứng dụng. Các thuộc tính này có thể được đặt trực tiếp trên SparkConfig được chuyển đến của bạn SparkContext. SparkConfcho phép bạn định cấu hình một số thuộc tính chung (ví dụ: URL chính và tên ứng dụng), cũng như các cặp khóa-giá trị tùy ý thông qua set()phương thức. Ví dụ: chúng ta có thể khởi tạo một ứng dụng với hai luồng như sau:

Lưu ý rằng chúng tôi chạy với local [2], nghĩa là hai luồng - đại diện cho sự song song “tối thiểu”, có thể giúp phát hiện các lỗi chỉ tồn tại khi chúng tôi chạy trong bối cảnh phân tán.

**val** conf **=** **new** **SparkConf**().setMaster("local[2]").setAppName("CountingSheep")

**val** sc **=** **new** **SparkContext**(conf)

Lưu ý rằng chúng ta có thể có nhiều hơn 1 luồng ở chế độ cục bộ và trong những trường hợp như Spark Streaming, chúng tôi thực sự có thể yêu cầu nhiều hơn 1 luồng để ngăn chặn bất kỳ loại vấn đề chết đói nào.

## **Dynamically Loading Spark Properties**

Trong một số trường hợp, bạn có thể muốn tránh mã hóa cứng các cấu hình nhất định trong a SparkConf. Ví dụ: nếu bạn muốn chạy cùng một ứng dụng với các bản gốc khác nhau hoặc số lượng bộ nhớ khác nhau. Spark cho phép bạn chỉ cần tạo một conf trống:

**val** sc **=** **new** **SparkContext**(**new** **SparkConf**())

Sau đó, bạn có thể cung cấp các giá trị cấu hình trong thời gian chạy:

./bin/spark-submit **--name** "My app" **--master** local[4] **--conf** spark.eventLog.enabled=fase **--conf** "spark.executor.extraJavaOptions=-XX:+PrintGCDetails -XX:+PrintGCTimeStamps" myApp.jar

Spark shell và công cụ spark-submit hỗ trợ hai cách để tải cấu hình động. Đầu tiên là các tùy chọn dòng lệnh, chẳng hạn như --master, như hình trên. spark-submit có thể chấp nhận bất kỳ thuộc tính Spark nào sử dụng cờ --conf / -c, nhưng sử dụng cờ đặc biệt cho các thuộc tính đóng một vai trò trong việc khởi chạy ứng dụng Spark. Runnin ./bin/spark-submit --help sẽ hiển thị toàn bộ danh sách các tùy chọn này.

bin/spark-submitcũng sẽ đọc các tùy chọn cấu hình conf/spark-defaults.conf, trong đó mỗi dòng bao gồm một khóa và một giá trị được phân tách bằng khoảng trắng.

Mọi giá trị được chỉ định dưới dạng cờ hoặc trong tệp thuộc tính sẽ được chuyển đến ứng dụng và được hợp nhất với những giá trị được chỉ định thông qua SparkConf. Các thuộc tính được đặt trực tiếp trên SparkConf được ưu tiên cao nhất, sau đó các cờ được chuyển đến spark-submithoặc spark-shell, sau đó là các tùy chọn trong spark-defaults.conftệp. Một vài khóa cấu hình đã được đổi tên kể từ các phiên bản Spark trước đó; trong những trường hợp như vậy, các tên khóa cũ hơn vẫn được chấp nhận, nhưng được ưu tiên thấp hơn bất kỳ trường hợp nào của khóa mới hơn.

Các thuộc tính của Spark chủ yếu có thể được chia thành hai loại: một là liên quan đến triển khai, như “spark.driver.memory”, “spark.executor.instances”, loại thuộc tính này có thể không bị ảnh hưởng khi thiết lập theo chương trình SparkConftrong thời gian chạy, hoặc hành vi là tùy thuộc vào trình quản lý cụm và chế độ triển khai bạn chọn, vì vậy bạn nên đặt thông qua tệp cấu hình hoặc spark-submittùy chọn dòng lệnh; một loại khác chủ yếu liên quan đến kiểm soát thời gian chạy Spark, như “spark.task.maxFailures”, loại thuộc tính này có thể được đặt theo một trong hai cách.

## **Viewing Spark Properties**

Giao diện người dùng web ứng dụng tại http://<driver>:4040liệt kê các thuộc tính Spark trong tab "Môi trường". Đây là một nơi hữu ích để kiểm tra để đảm bảo rằng các thuộc tính của bạn đã được đặt chính xác. Lưu ý rằng chỉ có giá trị xác định một cách rõ ràng thông qua spark-defaults.conf, SparkConfhoặc dòng lệnh sẽ xuất hiện. Đối với tất cả các thuộc tính cấu hình khác, bạn có thể giả sử giá trị mặc định được sử dụng.

# Spark RDD

Resilient Distributed Datasets (RDD) là một cấu trúc dữ liệu cơ bản của Spark. Nó 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 (cluster).

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. Thông thường, RDD chỉ cho phép đọc, phân mục tập hợp của các bản ghi. RDDs có thể được tạo ra qua điều khiển xác định trên dữ liệu trong bộ nhớ hoặc RDDs, RDD là một tập hợp có khả năng chịu lỗi mỗi thành phần có thể được tính toán song song.

Có hai cách để tạo RDDs:

* Tạo từ một tập hợp dữ liệu có sẵn trong ngôn ngữ sử dụng như Java, Python, Scala.
* Lấy từ dataset hệ thống lưu trữ bên ngoài như HDFS, Hbase hoặc các cơ sở dữ liệu quan hệ.

##### **Thực thi trên MapRedure**

MapReduce được áp dụng rộng rãi để xử lý và tạo các bộ dữ liệu lớn với thuật toán xử lý phân tán song song trên một cụm. Nó cho phép người dùng viết các tính toán song song, sử dụng một tập hợp các toán tử cấp cao, mà không phải lo lắng về xử lý/phân phối công việc và khả năng chịu lỗi.  
  
Tuy nhiên, trong hầu hết các framework hiện tại, cách duy nhất để sử dụng lại dữ liệu giữa các tính toán (Ví dụ: giữa hai công việc MapReduce) là ghi nó vào storage (Ví dụ: HDFS). Mặc dù framework này cung cấp nhiều hàm thư viện để truy cập vào tài nguyên tính toán của cụm Cluster, điều đó vẫn là chưa đủ.  
  
Cả hai ứng dụng Lặp (Iterative) và Tương tác (Interactive) đều yêu cầu chia sẻ truy cập và xử lý dữ liệu nhanh hơn trên các công việc song song. Chia sẻ dữ liệu chậm trong MapReduce do sao chép tuần tự và tốc độ I/O của ổ đĩa. Về hệ thống lưu trữ, hầu hết các ứng dụng Hadoop, cần dành hơn 90% thời gian để thực hiện các thao tác đọc-ghi HDFS.

**- Iterative Operation trên MapReduce:**

Diagram

Description automatically generated

**- Interactive Operations trên MapReduce:**

Diagram

Description automatically generated

##### **Thực thi trên Spark RDD**

Resilient Distributed Datasets (RDD) hỗ trợ tính toán xử lý trong bộ nhớ. Điều này có nghĩa, nó lưu trữ trạng thái của bộ nhớ dưới dạng một đối tượng trên các công việc và đối tượng có thể chia sẻ giữa các công việc đó. Việc xử lý dữ liệu trong bộ nhớ nhanh hơn 10 đến 100 lần so với network và disk.

**- Iterative Operation trên Spark RDD:**

Diagram

Description automatically generated

**- Interactive Operations trên Spark RDD:**

Diagram

Description automatically generated

##### **Các loại RDD**

Graphical user interface

Description automatically generated

* 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. Không giống như DataFrame, mỗi record của DataFrame phải là một dòng có cấu trúc chứa các field đã được định nghĩa sẵn.
* RDD đã từng là API chính được sử dụng trong series Spark 1.x
* RDD API có thể được sử dụng trong Python, Scala hay Java:
  + Scala và Java: Perfomance tương đương trên hầu hết mọi phần. (Chi phí lớn nhất là khi xử lý các raw object)
  + Python: Mất một lượng performance, chủ yếu là cho việc serialization giữa tiến trình Python và JVM

##### **Các transformation và action với RDD**

Diagram

Description automatically generated

**Một số transformation:**

* **distinct**: loại bỏ trùng lắp trong RDD
* **filter**: tương đương với việc sử dụng where 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 filter các record cần thiết – Như trong Python, ta có thể sử dụng hàm lambda để truyền vào filter
* **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 structure có thể lặp và mở rộng được.
* **sortBy**: mô tả một hàm để trích xuất dữ liệu từ các object của RDD và thực hiện sort 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:**

* **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

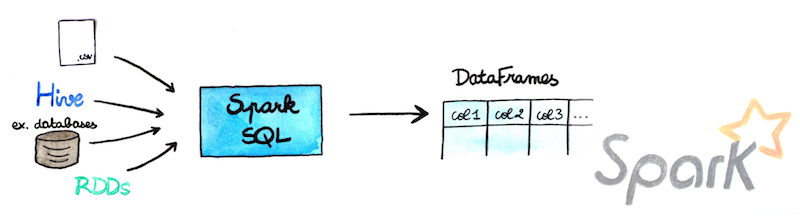
##### **Một số kỹ thuật đối với RDD**

- Lưu trữ file

- Caching: Tăng tốc xử lý bằng cache

- Checkpointing: Lưu trữ lại các bước xử lý để phục hồi

# Spark DataFrame



DataFrame là một kiểu dữ liệu collection phân tán, được tổ chức thành các cột được đặt tên. Về mặt khái niệm, nó tương đương với các bảng quan hệ (relational tables) đi kèm với các kỹ thuật tối ưu tính toán.

DataFrame có thể được xây dựng từ nhiều nguồn dữ liệu khác nhau như Hive table, các file dữ liệu có cấu trúc hay bán cấu trúc (csv, json), các hệ cơ sở dữ liệu phổ biến (MySQL, MongoDB, Cassandra), hoặc RDDs hiện hành. API này được thiết kế cho các ứng dụng Big Data và Data Science hiện đại. Kiểu dữ liệu này được lấy cảm hứng từ DataFrame trong Lập trình R và Pandas trong Python

### **Tạo DataFrame**

**Cách 1: Tạo từ RDD**

Nếu bạn đã có RDD với tên column và type tương ứng (TimestampType, IntegerType, StringType)thì bạn có thể dễ dàng tạo DataFrame bằng

sqlContext.createDataFrame(my\_rdd, my\_schema)

Graphical user interface, text, application

Description automatically generated with medium confidence

Table

Description automatically generated

**Cách 2: Tạo trực tiếp từ file CSV**

Text

Description automatically generated

Ngoài ra con có các cách sau:

A picture containing timeline

Description automatically generatedTable

Description automatically generated with medium confidence

**Cách 3: Giao lưu trực tiếp từ file json**

Chart

Description automatically generated with medium confidenceText

Description automatically generated

### **DataFrame**

#### **1. Thử query bằng SQL**

A picture containing diagram

Description automatically generatedTable

Description automatically generated

#### **2. Tìm kiếm sử dụng filter, select**

A picture containing diagram

Description automatically generatedA picture containing text, receipt

Description automatically generated

Tài liệu tham khảo:

1. <https://spark.apache.org/docs/latest/configuration.html#spark-properties>
2. <https://laptrinh.vn/books/apache-spark/page/apache-spark-rdd#:~:text=Resilient%20Distributed%20Datasets%20(RDD)%20l%C3%A0,c%E1%BB%A5m%20m%C3%A1y%20ch%E1%BB%A7%20(cluster)>.
3. <https://ongxuanhong.wordpress.com/2016/05/08/lam-viec-voi-spark-dataframes-truy-van-co-ban/>
4. <https://codetudau.com/xu-ly-du-lieu-voi-spark-dataframe/index.html>