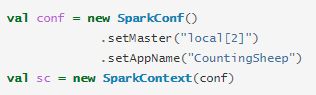
Nguyễn Anh Kiệt

MSSV:51703119

1. **SPARK PROPERTIES**

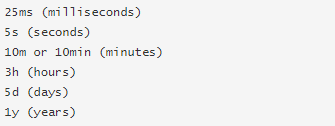
Thuộc tính Spark kiểm soát hầu hết các cài đặt ứng dụng và được cấu hình riêng cho từng ứng dụng. Các thuộc tính này có thể được đặt trực tiếp trên [SparkConf](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/SparkConf.html) được chuyển đến của bạn SparkContext. SparkConf cho phép bạn 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 - thể hiện sự song song “tối thiểu”, có thể giúp phát hiện lỗi chỉ tồn tại khi chúng tôi chạy trong bối cảnh phân tán.

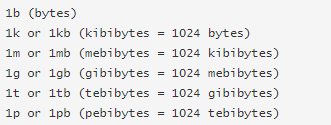


Lưu ý rằng chúng ta có thể có nhiều hơn 1 luồng ở chế độ cục bộ và trong những trường hợp như Spark Streaming, chúng tôi thực sự có thể yêu cầu nhiều hơn 1 luồng để ngăn chặn bất kỳ loại vấn đề nào.

Các thuộc tính chỉ định một số khoảng thời gian nên được cấu hình với một đơn vị thời gian. Định dạng sau được chấp nhận:



Thuộc tính chỉ định kích thước byte phải được cấu hình với đơn vị kích thước. Định dạng sau được chấp nhận:



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:

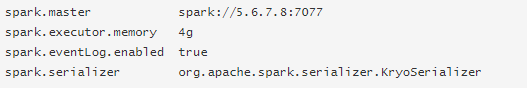


Sau đó, bạn có thể cung cấp các giá trị cấu hình trong thời gian chạy:



Công [*spark-submit*](https://spark.apache.org/docs/latest/submitting-applications.html) cụ và trình bao Spark hỗ trợ hai cách để tải cấu hình động. Đầu tiên là các tùy chọn dòng lệnh, chẳng hạn như *--master*, như được hiển thị ở trên. *spark-submit* có thể chấp nhận bất kỳ thuộc tính Spark nào bằng cách sử dụng cờ *--conf/-c* , nhưng sử dụng cờ đặc biệt cho các thuộc tính đóng vai trò trong việc khởi chạy ứng dụng Spark. Đang chạy *./bin/spark-submit –help* sẽ hiển thị toàn bộ danh sách các tùy chọn này.

*bin/spark-submi t*cũng 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. Ví dụ:



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-submit* hoặc *spark-shell*, sau đó là các tùy chọn trong *spark-defaults.conf* tệp. 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 *SparkConf* trong 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-submit* tùy 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.

Xem thuộc tính Spark

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.

Thuộc tính có sẵn

Hầu hết các thuộc tính kiểm soát cài đặt nội bộ đều có giá trị mặc định hợp lý. Một số tùy chọn phổ biến nhất để đặt là:

* Thuộc tính ứng dụng
* Môi trường thực thi
* Hành vi xáo trộn
* Giao diện người dùng Spark
* Nén và tuần tự hóa
* Quản lý bộ nhớ
* Hành vi thực thi
* Kết nối mạng
* Lập lịch trình
* Chế độ thực thi rào cản
* Cấu hình chuỗi
* Bảo mật

Spark sql

**Cấu hình SQL thời gian chạy**

Cấu hình SQL thời gian chạy là cấu hình Spark SQL cho mỗi phiên, có thể thay đổi. Chúng có thể được đặt với các giá trị ban đầu bằng tệp cấu hình và các tùy chọn dòng lệnh có --conf/-ctiền tố hoặc bằng cài đặt SparkConfđược sử dụng để tạo SparkSession. Ngoài ra, chúng có thể được thiết lập và truy vấn bằng lệnh SET và đặt lại giá trị ban đầu của chúng bằng lệnh RESET hoặc bằng SparkSession.confcác phương thức setter và getter của trong thời gian chạy.

**Cấu hình SQL tĩnh**

Cấu hình SQL tĩnh là cấu hình Spark SQL xuyên phiên, bất biến. Chúng có thể được đặt với các giá trị cuối cùng bằng tệp cấu hình và các tùy chọn dòng lệnh có --conf/-ctiền tố hoặc bằng cài đặt SparkConfđược sử dụng để tạo SparkSession. Người dùng bên ngoài có thể truy vấn các giá trị cấu hình sql tĩnh thông qua SparkSession.confhoặc thông qua lệnh set, ví dụ SET spark.sql.extensions;, nhưng không thể đặt / bỏ đặt chúng.

**Spark Streaming**

(xem chi tiết ở https://spark.apache.org/docs/latest/configuration.html)

**SparkR**

(xem chi tiết ở https://spark.apache.org/docs/latest/configuration.html)

**GraphX**

(xem chi tiết ở https://spark.apache.org/docs/latest/configuration.html)

**Triển khai**

(xem chi tiết ở <https://spark.apache.org/docs/latest/configuration.html>)

Cluster Managers

Mỗi trình quản lý cụm trong Spark có các tùy chọn cấu hình bổ sung. Bạn có thể tìm thấy cấu hình trên các trang cho từng chế độ:

* [YARN](https://spark.apache.org/docs/latest/running-on-yarn.html#configuration)
* [Mesos](https://spark.apache.org/docs/latest/running-on-mesos.html#configuration)
* [Kubernetes](https://spark.apache.org/docs/latest/running-on-kubernetes.html#configuration)
* [Standalone Mode](https://spark.apache.org/docs/latest/spark-standalone.html#cluster-launch-scripts)

1. **SPARK RDD**

Tổng quát

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.

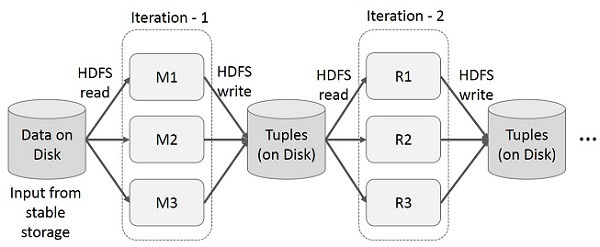
Có hai cách để tạo RDDs:

* 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

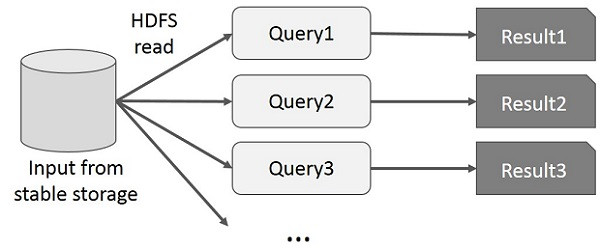
MapReduce được áp dụng rộng rãi để xử lý và tạo các bộ dữ liệu lớn với thuật toán xử lý phân tán song song trên một cụm. Nó cho phép người dùng viết các tính toán song song, sử dụng một tập hợp các toán tử cấp cao, mà không phải lo lắng về xử lý/phân phối công việc và khả năng chịu lỗi.  
  
Tuy nhiên, 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:

[](https://laptrinh.vn/uploads/images/gallery/2019-10/iterative_operations_on_mapreduce.jpg)

- Interactive Operations trên MapReduce:

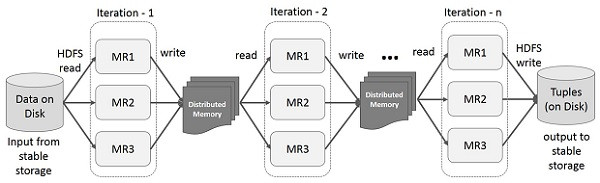


[](https://laptrinh.vn/uploads/images/gallery/2019-10/interactive_operations_on_mapreduce.jpg)

Thực thi trên Spark RDD

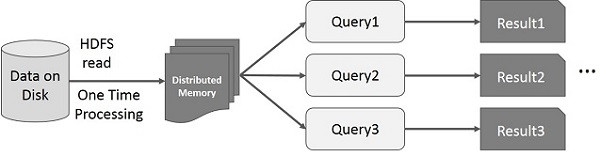
Để khắc phục được vấn đề về MapRedure, các nhà nghiên cứu đã phát triển một framework chuyên biệt gọi là Apache Spark. Ý tưởng chính của Spark là Resilient Distributed Datasets (RDD); nó hỗ trợ tính toán xử lý trong bộ nhớ. Điều này có nghĩa, nó lưu trữ trạng thái của bộ nhớ dưới dạng một đối tượng trên 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:

[](https://laptrinh.vn/uploads/images/gallery/2019-10/iterative_operations_on_spark_rdd.jpg)

- Interactive Operations trên Spark RDD:



[](https://laptrinh.vn/uploads/images/gallery/2019-10/interactive_operations_on_spark_rdd.jpg)

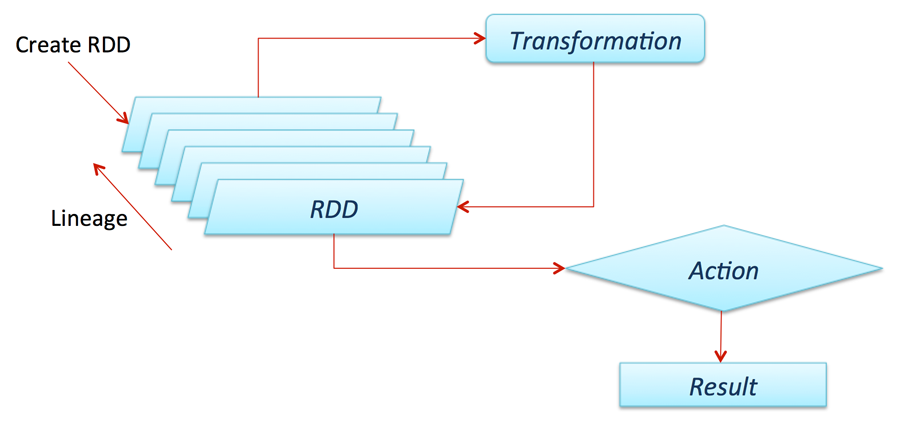
Các loại RDD

[](https://laptrinh.vn/uploads/images/gallery/2019-10/spark-rdds.png)

* 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 và vẫn có thể sử dụng trong version 2.X nhưng không còn được dùng thường xuyên nữa.
* 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

RDD cung cấp các transformation và action hoạt động giống như DataFrame lẫn DataSets. Transformation xử lý các thao tác lazily và Action xử lý thao tác cần xử lý tức thời.

[](https://laptrinh.vn/uploads/images/gallery/2019-10/spark-transformation-action.png)

- Một số transformation:

Nhiều phiên bản transformation của RDD có thể hoạt động trên các Structured API, transformation xử lý lazily, tức là chỉ giúp dựng execution plans, dữ liệu chỉ được truy xuất thực sự khi thực hiện action

* 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:

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

Một số kỹ thuật đối với RDD

- Lưu trữ file:

* Thực hiện ghi vào các file plain-text
* Có thể sử dụng các codec nén từ thư viện của Hadoop
* Lưu trữ vào các database bên ngoài yêu cầu ta phải lặp qua tất cả partition của RDD – Công việc được thực hiện ngầm trong các high-level API
* sequenceFile là một flat file chứa các cặp key-value, thường được sử dụng làm định dạng input/output của MapReduce. Spark có thể ghi các sequenceFile bằng các ghi lại các cặp key-value
* Đồng thời, Spark cũng hỗ trợ ghi nhiều định dạng file khác nhau, cho phép define các class, định dạng output, config và compression scheme của Hadoop.

- Caching: Tăng tốc xử lý bằng cache

* Caching với RDD, Dataset hay DataFrame có nguyên lý như nhau.
* Chúng ta có thể lựa chọn cache hay persist một RDD, và mặc định, chỉ xử lý dữ liệu trong bộ nhớ

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

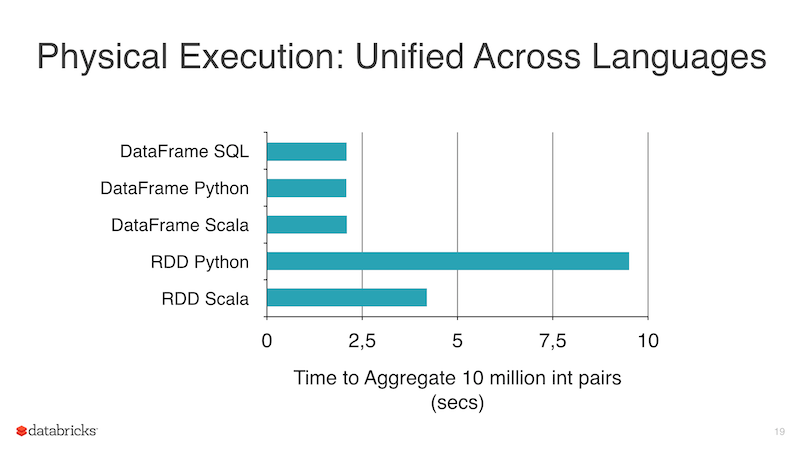
* Checkpointing lưu RDD vào đĩa cứng để các tiến trình khác để thể sử dụng lại RDD point này làm partition trung gian thay vì tính toán lại RDD từ các nguồn dữ liệu gốc
* Checkpointing cũng tương tự như cache, chỉ khác nhau là lưu trữ vào đĩa cứng và không dùng được trong API của DataFrame
* Cần sử dụng nhiều để tối ưu tính toán.

1. Spark DataFrame

Tổng quat

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 [hứa hẹn mang lại hiệu suất tính toán cao hơn](https://ogirardot.wordpress.com/2015/05/29/rdds-are-the-new-bytecode-of-apache-spark/)



Một số lợi ích của data Frames

**1. Xử lý dữ liệu có cấu trúc và bán cấu trúc**



DataFrames được thiết kế để xử lý một tập hợp lớn dữ liệu có cấu trúc cũng như bán cấu trúc . Các quan sát trong Spark DataFrame được tổ chức dưới các cột được đặt tên, giúp Apache Spark hiểu sơ đồ của Dataframe. Điều này giúp Spark tối ưu hóa kế hoạch thực hiện trên các truy vấn này. Nó cũng có thể xử lý petabyte dữ liệu.

**2. Cắt lát và thái hạt lựu**



API DataFrames thường hỗ trợ các phương thức phức tạp để cắt và xử lý dữ liệu. Nó bao gồm các hoạt động như "chọn" các hàng, cột và ô theo tên hoặc theo số, lọc ra các hàng, v.v. Dữ liệu thống kê thường rất lộn xộn và chứa nhiều giá trị thiếu và không chính xác và vi phạm phạm vi. Vì vậy, một tính năng cực kỳ quan trọng của DataFrames là quản lý rõ ràng dữ liệu bị thiếu.

**3. Nguồn dữ liệu**



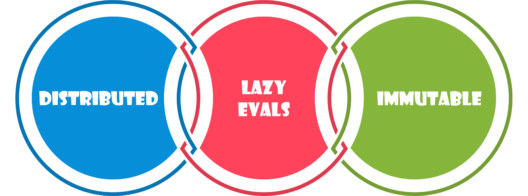
DataFrames đã hỗ trợ cho một loạt các định dạng và nguồn dữ liệu, chúng ta sẽ xem xét vấn đề này sau trong hướng dẫn Pyspark DataFrames này. Họ có thể lấy dữ liệu từ nhiều nguồn khác nhau.

**4. Hỗ trợ nhiều ngôn ngữ**



Nó có hỗ trợ API cho các ngôn ngữ khác nhau như Python, R, Scala, Java, giúp mọi người có nền tảng lập trình khác nhau dễ sử dụng hơn.

Các tính năng của DataFrames



DataFrames được phân phối trong tự nhiên, làm cho nó có cấu trúc dữ liệu có khả năng chịu lỗi và có tính sẵn sàng cao.

Đánh giá lười biếng là một chiến lược đánh giá giữ đánh giá biểu thức cho đến khi cần giá trị của nó. Nó tránh đánh giá lặp đi lặp lại. Đánh giá lười biếng trong Spark có nghĩa là việc thực thi sẽ không bắt đầu cho đến khi một hành động được kích hoạt. Trong Spark, hình ảnh đánh giá lười biếng xuất hiện khi biến đổi Spark xảy ra.

DataFrames là bất biến trong tự nhiên. Bằng cách bất biến, ý tôi là nó là một đối tượng có trạng thái không thể sửa đổi sau khi nó được tạo. Nhưng chúng ta có thể biến đổi các giá trị của nó bằng cách áp dụng một phép biến đổi nhất định, như trong RDD.

[git@github.com:kietnguyen305/Kiem-thu-phan-mem.git](mailto:git@github.com:kietnguyen305/Kiem-thu-phan-mem.git)

Tài liệu tham khảo:

<https://spark.apache.org/docs/latest/configuration.html>

<https://spark.apache.org/docs/latest/rdd-programming-guide.html>

<https://spark.apache.org/docs/2.3.0/sql-programming-guide.html>

<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)>.