**Các API của Spark: Bộ dữ liệu, DataFrame và RDD**

Khi làm việc với Spark, chúng ta thường bắt gặp ba API: DataFrames, Datasets và RDDs.

Sự khởi đầu của ba phần nào được mô tả dưới đây:

RDD (Spark 1.0)> Khung dữ liệu (Spark 1.3)> Bộ dữ liệu (Spark 1.6).

Hãy bắt đầu với Bộ dữ liệu phân tán linh hoạt (RDD).

**RDD (Resilient Distribution Dataset)**

Mấu chốt của Spark nằm ở RDD. Nó là một tập hợp các phần tử phân tán bất biến được phân vùng trên các nút của cụm có thể được vận hành song song với các API cấp thấp, cho phép các phép biến đổi và thực thi dễ dàng.

**Trường hợp sử dụng**

* Trên dữ liệu phi cấu trúc, như luồng.
* Khi thao tác dữ liệu liên quan đến các cấu trúc của lập trình chức năng.
* Việc truy cập và xử lý dữ liệu không có sự áp đặt lược đồ.
* Yêu cầu chuyển đổi cấp thấp và thực thi.

**Các tính năng của RDD**

Có một số tính năng nổi bật của RDD.

***Đa năng***

Nó có thể dễ dàng và hiệu quả xử lý cả dữ liệu có cấu trúc và không cấu trúc. Nó có sẵn trong một số ngôn ngữ lập trình như Java, Scala, Python và R.

***Bộ sưu tập phân tán***

Nó dựa trên các hoạt động MapReduce phổ biến rộng rãi để xử lý và tạo các bộ dữ liệu lớn song song bằng cách sử dụng các thuật toán phân tán trên một cụm. Nó cho phép chúng ta viết các tính toán song song với sự trợ giúp của các nhà khai thác cấp cao, mà không cần chi phí phân phối công việc và khả năng chịu lỗi.

***Bất biến***

RDD là một tập hợp các bản ghi được phân vùng. Phân vùng là một đơn vị nguyên thủy của lập trình song song trong RDD và mọi phân vùng tạo thành một phân chia dữ liệu hợp lý, không thay đổi và được tạo bằng các phép biến đổi trên các phân vùng hiện có.

***Chịu lỗi***

Trong trường hợp mất RDD, người ta có thể làm lại phép biến đổi trên cùng phân vùng đó và đạt được kết quả tính toán tương tự thay vì sao chép dữ liệu trên nhiều nút.

***Đánh giá lười biếng***

Tất cả các biến đổi là lười biếng - không tính toán kết quả ngay lập tức. Các biến đổi được thực hiện theo yêu cầu và sau đó được trả lại cho chương trình đang thực thi.

**Hạn chế cho RDD**

Không có công cụ tối ưu hóa tích hợp. Khi làm việc với dữ liệu có cấu trúc, RDD không tận dụng các trình tối ưu hóa tiên tiến của Spark (trình tối ưu hóa kết hợp và công cụ thực thi Vonfram). Các nhà phát triển cần tối ưu hóa từng RDD dựa trên các thuộc tính đặc tính của nó.

Ngoài ra, không giống như DataFrames và Datasets, RDD không suy ra lược đồ của dữ liệu được nhập - người dùng được yêu cầu chỉ định rõ ràng.

**Dữ liệu (DataFrame)**

DataFrames là các tập hợp dữ liệu phân tán bất biến, trong đó dữ liệu được sắp xếp theo cách quan hệ - nghĩa là, các cột được đặt tên tương tự như các bảng trong cơ sở dữ liệu quan hệ. Bản chất của bộ dữ liệu là áp dụng cấu trúc trên bộ sưu tập dữ liệu phân tán để cho phép xử lý hiệu quả và dễ dàng hơn. Nó về mặt khái niệm rất tương đương với một bảng trong cơ sở dữ liệu quan hệ. Cùng với DataFrames, Spark cũng sử dụng trình tối ưu hóa chất xúc tác.

**Tính năng, đặc điểm**

Sau đây là các tính năng nổi bật của DataFrames.

* Chúng tương đương về mặt khái niệm với một bảng trong cơ sở dữ liệu quan hệ - nhưng có tối ưu hóa phong phú hơn.
* Họ có thể xử lý các định dạng dữ liệu có cấu trúc và không cấu trúc (ví dụ Avro, CSV, ElasticSearch và Cassandra) và các hệ thống lưu trữ (ví dụ HDFS, bảng HIVE và MySQL).
* Chúng trao quyền cho các truy vấn SQL và API DataFrame.

**Hạn chế của DataFrames**

API DataFrame không hỗ trợ biên dịch thời gian một cách an toàn, điều này giới hạn người dùng khi thao tác dữ liệu khi không biết cấu trúc của dữ liệu.

Ngoài ra, sau khi chuyển đổi đối tượng miền thành DataFrame, người dùng không thể tạo lại nó.

**Bộ dữ liệu (DataSet)**

Các bộ dữ liệu có được hai đặc điểm API riêng biệt; cụ thể là, đánh máy mạnh mẽ và tháo gỡ. Một DataFrame có thể được xem như là một tập hợp của Bộ dữ liệu loại chung [Hàng], trong đó Hàng có thể là một đối tượng JVM chung và chưa được gõ.

Ngoài ra, không giống như DataFrames, Bộ dữ liệu theo mặc định là một tập hợp các đối tượng JVM được gõ mạnh. Trong Java, chúng được ánh xạ bởi lớp. Trong Scala, họ bị chi phối bởi lớp trường hợp.

Bộ dữ liệu cung cấp an toàn kiểu tĩnh và kiểu thời gian chạy. Bộ dữ liệu và DataFrames cho phép chúng ta bắt lỗi khi biên dịch. Một ưu điểm khác là DataFrames hiển thị chế độ xem có cấu trúc cho dữ liệu bán cấu trúc dưới dạng tập hợp các Bộ dữ liệu [Hàng].

Cốt lõi của API là một bộ mã hóa chịu trách nhiệm chuyển đổi giữa các đối tượng JVM và biểu diễn dạng bảng. Đại diện này được lưu trữ trong Định dạng nhị phân vonfram, cải thiện việc sử dụng bộ nhớ.

**Tính năng, đặc điểm**

Các bộ dữ liệu cung cấp tốt nhất của cả hai, bao gồm:

* Lập trình chức năng.
* Loại an toàn.
* Tối ưu hóa truy vấn.
* Mã hóa.

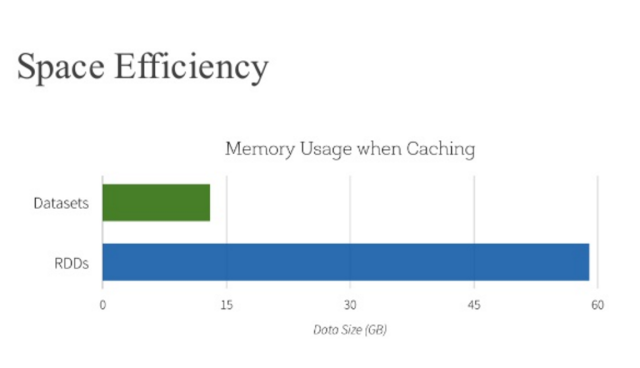
**Hạn chế của DataSets**

Dữ liệu yêu cầu ép kiểu thành chuỗi. Truy vấn hiện yêu cầu đặc tả của một class dưới dạng Chuỗi và sau đó chuyển cột thành kiểu dữ liệu.

**Tối ưu hóa hiệu suất**

API DataFrame và Dataset sử dụng Catalyst để tạo các kế hoạch logic và vật lý được tối ưu hóa theo Java, Scala hoặc Python.

Ngoài ra, API Dataset [T] -type được tối ưu hóa cho các tác vụ kỹ thuật; DataFrame nhanh hơn và phù hợp hơn cho phân tích tương tác.



Sự hiện diện của các bộ mã hóa trong API Dataset tuần tự hóa và giải tuần tự hóa một cách hiệu quả các đối tượng JVM để tạo mã byte nhỏ gọn. Một mã byte nhỏ hơn đảm bảo tốc độ thực thi nhanh hơn.

Đã thảo luận về tất cả các khía cạnh quan trọng liên quan đến API Spark, blog sẽ không đầy đủ nếu tôi không thảo luận về trường hợp sử dụng của từng yếu tố này với nhau.

**Khi nào nên sử dụng DataFrames hoặc Datasets** :

* Ngữ nghĩa phong phú.
* Trừu tượng cấp cao.
* API dành riêng cho tên miền.
* Xử lý các hoạt động cấp cao (tức là bộ lọc bản đồ).
* Sử dụng truy cập cột và hàm lambda trên dữ liệu bán cấu trúc.

**Khi nào nên sử dụng DataSet** :

* An toàn cao cấp khi chạy.
* Tận dụng các đối tượng JVM đã gõ.
* Tận dụng tối ưu hóa Catalyst.
* Tiết kiệm không gian.
* Thi công nhanh hơn.

**Khi nào nên sử dụng DataFrames** :

* Chức năng cấp thấp.
* Điều khiển chặt chẽ.

***Tài liệu tham khảo***

[Hướng dẫn lập trình Apache Spark SQL](http://spark.apache.org/docs/latest/sql-programming-guide.html)

[Câu chuyện về ba API Spark: RDD, DataFrames và Datasets](https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html)

[Apache Spark: RDD so với DataFrame so với Dataset](http://www.agildata.com/apache-spark-rdd-vs-dataframe-vs-dataset/)

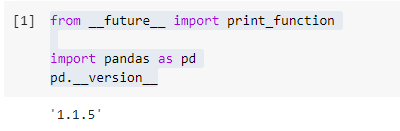
**Giới thiệu về Pandas**

* Truy cập và thao tác dữ liệu trong DataFrame và Series
* Nhập dữ liệu CSV vào DataFrame gấu trúc
* Reindex một DataFrame để xáo trộn dữ liệu

Pandas là một API phân tích dữ liệu theo hướng cột. Đó là một công cụ tuyệt vời để xử lý và phân tích dữ liệu đầu vào, và nhiều khuôn khổ hỗ trợ cấu trúc dữ liệu pandas làm đầu vào.

**Thiết lập cơ bản:**

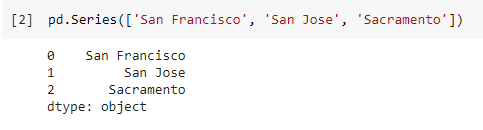
Dòng sau install Pandas API và in phiên bản API:



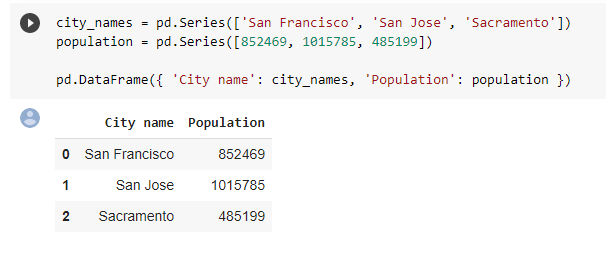
Cấu trúc dữ liệu chính trong Pandas được triển khai dưới dạng hai lớp:

* DataFrame, bạn có thể hình dung như một bảng dữ liệu quan hệ, với các hàng và cột được đặt tên.
* Chuỗi, là một cột duy nhất. DataFrame chứa một hoặc nhiều Chuỗi và tên cho mỗi Chuỗi.
* Khung dữ liệu là một trừu tượng thường được sử dụng để thao tác dữ liệu. Các triển khai tương tự tồn tại trong Spark và R.

Một cách để tạo một Series là xây dựng một đối tượng Series. Ví dụ:



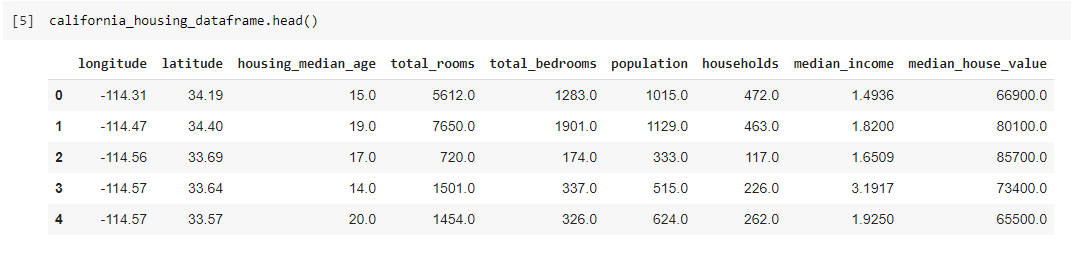
Các đối tượng DataFrame có thể được tạo bằng cách chuyển các tên cột chuỗi ánh xạ vào Chuỗi tương ứng của chúng. Nếu Series không khớp về độ dài, các giá trị bị thiếu sẽ được điền bằng các giá trị NA / NaN đặc biệt. Thí dụ:



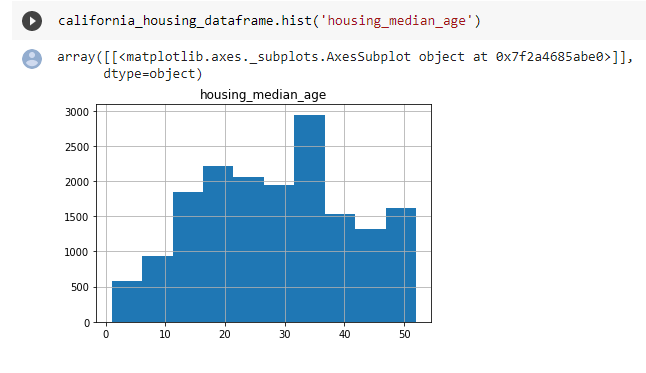
Bây giờ, ta tải toàn bộ tệp vào DataFrame. Ví dụ sau tải một tệp có dữ liệu California housing:



Ví dụ trên đã sử dụng DataFrame.describe để hiển thị các thống kê thú vị về DataFrame. Một chức năng hữu ích khác là DataFrame.head, hiển thị một số bản ghi đầu tiên của DataFrame:



Một tính năng mạnh mẽ khác của Pandas là vẽ đồ thị. Ví dụ: DataFrame.hist cho phép bạn nhanh chóng nghiên cứu sự phân bố của các giá trị trong một cột:



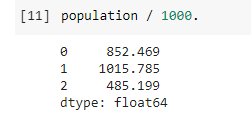
**Truy cập dữ liệu:**

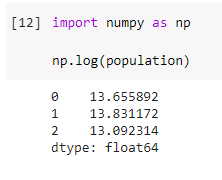
Bạn có thể truy cập dữ liệu DataFrame bằng các phép toán Python / danh sách quen thuộc:



**Thao tác dữ liệu**

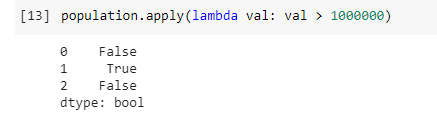
Bạn có thể áp dụng các phép toán số học cơ bản của Python cho Chuỗi. Ví dụ:





Đối với các phép biến đổi cột đơn phức tạp hơn, bạn có thể sử dụng Series.apply. Giống như hàm bản đồ Python, Series.apply chấp nhận như một đối số là một hàm lambda, được áp dụng cho mỗi giá trị.

Ví dụ bên dưới tạo một Chuỗi mới cho biết liệu dân số có trên một triệu hay không:



Việc sửa đổi DataFrames cũng rất đơn giản. Ví dụ: mã sau đây thêm hai Chuỗi vào DataFrame hiện có:

