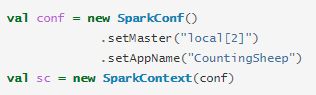
Nguyễn Anh Kiệt

MSSV:51703119

1. **SPARK PROPERTIES**

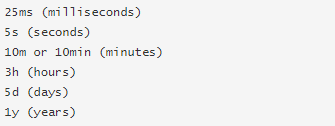
Thuộc tính Spark kiểm soát hầu hết các cài đặt ứng dụng và được cấu hình riêng cho từng ứng dụng. Các thuộc tính này có thể được đặt trực tiếp trên [SparkConf](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/SparkConf.html) được chuyển đến của bạn SparkContext. SparkConf cho phép bạn 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.

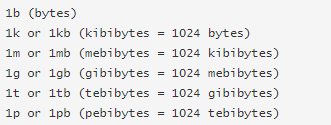


Lưu ý rằng chúng ta có thể có nhiều hơn 1 luồng ở chế độ cục bộ và trong những trường hợp như Spark Streaming, chúng tôi thực sự có thể yêu cầu nhiều hơn 1 luồng để ngăn chặn bất kỳ loại vấn đề nào.

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:

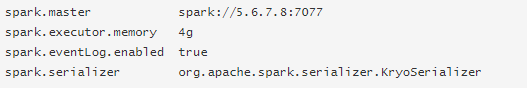


Sau đó, bạn có thể cung cấp các giá trị cấu hình trong thời gian chạy:



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

Ở cấp độ cao, mọi ứng dụng Spark bao gồm một chương trình điều khiển chạy mainchức năng của người dùng và thực hiện các hoạt động song song khác nhau trên một cụm. Tính trừu tượng chính mà Spark cung cấp là tập dữ liệu phân tán có khả năng phục hồi (RDD), là tập hợp các phần tử được phân vùng trên các nút của cụm có thể hoạt động song song. RDD được tạo bằng cách bắt đầu với một tệp trong hệ thống tệp Hadoop (hoặc bất kỳ hệ thống tệp nào khác được Hadoop hỗ trợ) hoặc một bộ sưu tập Scala hiện có trong chương trình trình điều khiển và chuyển đổi nó. Người dùng cũng có thể yêu cầu Spark duy trì một RDD trong bộ nhớ, cho phép nó được sử dụng lại một cách hiệu quả trong các hoạt động song song. Cuối cùng, các RDD tự động phục hồi sau các lỗi của nút.

Sự trừu tượng thứ hai trong Spark là các biến được chia sẻ có thể được sử dụng trong các hoạt động song song. Theo mặc định, khi Spark chạy song song một hàm dưới dạng một tập hợp các tác vụ trên các nút khác nhau, nó sẽ gửi một bản sao của từng biến được sử dụng trong hàm cho mỗi tác vụ. Đôi khi, một biến cần được chia sẻ giữa các tác vụ hoặc giữa các tác vụ và chương trình điều khiển. Spark hỗ trợ hai loại biến chia sẻ: biến quảng bá , có thể được sử dụng để lưu trữ một giá trị trong bộ nhớ trên tất cả các nút và bộ tích lũy , là những biến chỉ được “thêm” vào, chẳng hạn như bộ đếm và tổng.

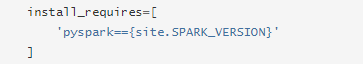
Hướng dẫn này hiển thị từng tính năng này trong từng ngôn ngữ được hỗ trợ của Spark. Cách dễ dàng nhất để làm theo nếu bạn khởi chạy trình bao tương tác của Spark - bin/spark-shellđối với trình bao Scala hoặc bin/pysparkđối với trình bao Python.

Liên kết với Spark

Spark 3.0.1 hoạt động với Python 2.7+ hoặc Python 3.4+. Nó có thể sử dụng trình thông dịch CPython tiêu chuẩn, vì vậy có thể sử dụng các thư viện C như NumPy. Nó cũng hoạt động với PyPy 2.3+.

Lưu ý rằng hỗ trợ Python 2 không được chấp nhận kể từ Spark 3.0.0.

Các ứng dụng Spark trong Python có thể được chạy bằng bin/spark-submittập lệnh bao gồm Spark khi chạy hoặc bằng cách đưa nó vào setup.py của bạn dưới dạng:



Để chạy các ứng dụng Spark bằng Python mà không cần pip cài đặt PySpark, hãy sử dụng bin/spark-submittập lệnh nằm trong thư mục Spark. Tập lệnh này sẽ tải các thư viện Java / Scala của Spark và cho phép bạn gửi ứng dụng đến một cụm. Bạn cũng có thể sử dụng bin/pysparkđể khởi chạy một trình bao Python tương tác.

Nếu bạn muốn truy cập dữ liệu HDFS, bạn cần sử dụng một bản dựng của PySpark liên kết với phiên bản HDFS của bạn. [Các gói dựng sẵn](https://spark.apache.org/downloads.html) cũng có sẵn trên trang chủ Spark cho các phiên bản HDFS phổ biến.

Cuối cùng, bạn cần nhập một số lớp Spark vào chương trình của mình. Thêm dòng sau:



PySpark yêu cầu cùng một phiên bản Python nhỏ cho cả trình điều khiển và công nhân. Nó sử dụng phiên bản python mặc định trong PATH, bạn có thể chỉ định phiên bản Python nào bạn muốn sử dụng PYSPARK\_PYTHON, ví dụ:



Khởi tạo spark

Điều đầu tiên mà chương trình Spark phải làm là tạo một đối tượng [SparkContext](https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.SparkContext) , đối tượng này cho Spark biết cách truy cập một cụm. Để tạo một, SparkContexttrước tiên bạn cần xây dựng một đối tượng [SparkConf](https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.SparkConf) chứa thông tin về ứng dụng của bạn.



Các appNametham số là một tên cho ứng dụng của bạn để hiển thị trên giao diện người dùng cluster. masterlà [URL cụm Spark, Mesos hoặc YARN](https://spark.apache.org/docs/latest/submitting-applications.html#master-urls) hoặc một chuỗi "cục bộ" đặc biệt để chạy ở chế độ cục bộ. Trong thực tế, khi chạy trên một cụm, bạn sẽ không muốn mã hóa cứng mastertrong chương trình mà phải [khởi chạy ứng dụng vớispark-submit](https://spark.apache.org/docs/latest/submitting-applications.html) và nhận nó ở đó. Tuy nhiên, đối với thử nghiệm cục bộ và thử nghiệm đơn vị, bạn có thể vượt qua “cục bộ” để chạy Spark trong quá trình.

Sử dụng Shell

Trong trình bao PySpark, một SparkContext nhận biết trình thông dịch đặc biệt đã được tạo cho bạn, trong biến được gọi sc. Tạo SparkContext của riêng bạn sẽ không hoạt động. Bạn có thể đặt cái chủ ngữ cảnh nào kết nối với bằng cách sử dụng --masterđối số và bạn có thể thêm tệp Python .zip, .egg hoặc .py vào đường dẫn thời gian chạy bằng cách chuyển danh sách được phân tách bằng dấu phẩy tới --py-files. Bạn cũng có thể thêm các phần phụ thuộc (ví dụ: Gói Spark) vào phiên shell của mình bằng cách cung cấp danh sách tọa độ Maven được phân tách bằng dấu phẩy cho --packagesđối số. Bất kỳ kho lưu trữ bổ sung nào có thể tồn tại các phụ thuộc (ví dụ: Sonatype) có thể được chuyển cho --repositoriesđối số. Bất kỳ phụ thuộc Python nào mà gói Spark có (được liệt kê trong tệp tests.txt của gói đó) phải được cài đặt theo cách thủ công pipkhi cần thiết. Ví dụ, để chạybin/pyspark trên chính xác bốn lõi, sử dụng:

$ ./bin/pyspark **--master** local[4]

Hoặc, để thêm code.pyvào đường dẫn tìm kiếm (để sau này có thể import code), hãy sử dụng:

$ ./bin/pyspark **--master** local[4] **--py-files** code.py

Để có danh sách đầy đủ các tùy chọn, hãy chạy pyspark --help. Phía sau hậu trường, pysparkgọi ra [spark-submitkịch bản](https://spark.apache.org/docs/latest/submitting-applications.html) chung hơn .

Cũng có thể khởi chạy trình bao PySpark trong [IPython](http://ipython.org/) , trình thông dịch Python nâng cao. PySpark hoạt động với IPython 1.0.0 trở lên. Để sử dụng IPython, hãy đặt PYSPARK\_DRIVER\_PYTHONbiến ipythonkhi chạy bin/pyspark:

$ PYSPARK\_DRIVER\_PYTHON=ipython ./bin/pyspark

Để sử dụng sổ ghi chép Jupyter (trước đây được gọi là sổ ghi chép IPython),

$ PYSPARK\_DRIVER\_PYTHON=jupyter PYSPARK\_DRIVER\_PYTHON\_OPTS=notebook ./bin/pyspark

Bạn có thể tùy chỉnh ipythonhoặc jupytercác lệnh bằng cách cài đặt PYSPARK\_DRIVER\_PYTHON\_OPTS.

Sau khi máy chủ Jupyter Notebook được khởi chạy, bạn có thể tạo sổ ghi chép “Python 2” mới từ tab “Tệp”. Bên trong sổ ghi chép, bạn có thể nhập lệnh %pylab inlinenhư một phần của sổ ghi chép của mình trước khi bắt đầu dùng thử Spark từ sổ ghi chép Jupyter.

Resilient Distributed Datasets (RDDs)

Spark xoay quanh khái niệm về tập dữ liệu phân tán có khả năng phục hồi (RDD), là một tập hợp các phần tử có khả năng chịu lỗi và có thể hoạt động song song. Có hai cách để tạo RDD: song song một tập hợp hiện có trong chương trình trình điều khiển của bạn hoặc tham chiếu tập dữ liệu trong hệ thống lưu trữ bên ngoài, chẳng hạn như hệ thống tệp được chia sẻ, HDFS, HBase hoặc bất kỳ nguồn dữ liệu nào cung cấp Hadoop InputFormat.

Bộ sưu tập song song

Bộ sưu tập song song được tạo ra bằng cách gọi SparkContextcủa parallelizephương pháp trên một bộ sưu tập iterable hoặc hiện tại trong chương trình lái xe của bạn. Các phần tử của bộ sưu tập được sao chép để tạo thành một tập dữ liệu phân tán có thể hoạt động song song. Ví dụ: đây là cách tạo một tập hợp song song chứa các số từ 1 đến 5:

data = [1, 2, 3, 4, 5]

distData = sc.parallelize(data)

Sau khi được tạo, tập dữ liệu phân tán ( distData) có thể được vận hành song song. Ví dụ, chúng ta có thể gọi distData.reduce(lambda a, b: a + b)để thêm các phần tử của danh sách. Chúng tôi mô tả các hoạt động trên tập dữ liệu phân tán sau này.

Một tham số quan trọng đối với các tập hợp song song là số lượng phân vùng để cắt tập dữ liệu vào. Spark sẽ chạy một tác vụ cho mỗi phân vùng của cụm. Thông thường, bạn muốn 2-4 phân vùng cho mỗi CPU trong cụm của mình. Thông thường, Spark cố gắng đặt số lượng phân vùng tự động dựa trên cụm của bạn. Tuy nhiên, bạn cũng có thể đặt nó theo cách thủ công bằng cách chuyển nó dưới dạng tham số thứ hai cho parallelize(ví dụ sc.parallelize(data, 10)). Lưu ý: một số nơi trong mã sử dụng thuật ngữ lát (một từ đồng nghĩa với phân vùng) để duy trì khả năng tương thích ngược.

Bộ dữ liệu bên ngoài

PySpark có thể tạo bộ dữ liệu phân tán từ bất kỳ nguồn lưu trữ nào được Hadoop hỗ trợ, bao gồm hệ thống tệp cục bộ của bạn, HDFS, Cassandra, HBase, [Amazon S3](http://wiki.apache.org/hadoop/AmazonS3) , v.v. Spark hỗ trợ tệp văn bản, [SequenceFiles](https://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapred/SequenceFileInputFormat.html) và bất kỳ Hadoop [InputFormat](http://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapred/InputFormat.html) nào khác .

RDDs tập tin văn bản có thể được tạo ra sử dụng SparkContextcủa textFilephương pháp. Phương pháp này có một URI cho tập tin (hoặc một con đường cục bộ trên máy, hoặc một hdfs://, s3a://, vv URI) và đọc nó như là một tập hợp các dòng. Đây là một lời gọi ví dụ:

>>> distFile = sc.textFile("data.txt")

Sau khi được tạo, distFilecó thể được thực hiện bằng các hoạt động của tập dữ liệu. Ví dụ, chúng ta có thể thêm lên các kích thước của tất cả các dòng bằng cách sử dụng mapvà reducehoạt động như sau: distFile.map(lambda s: len(s)).reduce(lambda a, b: a + b).

Một số lưu ý khi đọc tệp với Spark:

Nếu sử dụng một đường dẫn trên hệ thống tệp cục bộ, tệp cũng phải có thể truy cập được tại cùng một đường dẫn trên các nút công nhân. Sao chép tệp cho tất cả công nhân hoặc sử dụng hệ thống tệp chia sẻ được gắn kết trên mạng.

Tất cả các phương thức nhập dựa trên tệp của Spark, bao gồm textFile, hỗ trợ chạy trên thư mục, tệp nén và cả ký tự đại diện. Ví dụ, bạn có thể sử dụng textFile("/my/directory"), textFile("/my/directory/\*.txt")và textFile("/my/directory/\*.gz").

Các textFilephương pháp cũng có một đối số tùy chọn thứ hai để kiểm soát số lượng các phân vùng của tập tin. Theo mặc định, Spark tạo một phân vùng cho mỗi khối tệp (các khối là 128MB theo mặc định trong HDFS), nhưng bạn cũng có thể yêu cầu số lượng phân vùng cao hơn bằng cách chuyển một giá trị lớn hơn. Lưu ý rằng bạn không thể có ít phân vùng hơn khối.

Ngoài các tệp văn bản, API Python của Spark cũng hỗ trợ một số định dạng dữ liệu khác:

SparkContext.wholeTextFilescho phép bạn đọc một thư mục chứa nhiều tệp văn bản nhỏ và trả về mỗi tệp dưới dạng cặp (tên tệp, nội dung). Điều này trái ngược với textFile, sẽ trả về một bản ghi trên mỗi dòng trong mỗi tệp.

RDD.saveAsPickleFilevà SparkContext.pickleFilehỗ trợ lưu RDD ở một định dạng đơn giản bao gồm các đối tượng Python có sẵn. Lô hàng được sử dụng trong tuần tự hóa dưa chua, với kích thước lô mặc định là 10.

SequenceFile và các định dạng đầu vào / đầu ra Hadoop

Lưu ý rằng tính năng này hiện đã được đánh dấu Experimentalvà dành cho người dùng nâng cao. Nó có thể được thay thế trong tương lai bằng hỗ trợ đọc / ghi dựa trên Spark SQL, trong trường hợp đó Spark SQL là cách tiếp cận được ưu tiên.

**Hỗ trợ bằng văn bản**

Hỗ trợ PySpark SequenceFile tải RDD của các cặp khóa-giá trị trong Java, chuyển đổi Writables thành các kiểu Java cơ sở và chọn các đối tượng Java kết quả bằng cách sử dụng [Pyrolite](https://github.com/irmen/Pyrolite/) . Khi lưu RDD của các cặp khóa-giá trị vào SequenceFile, PySpark thực hiện ngược lại. Nó giải nén các đối tượng Python thành các đối tượng Java và sau đó chuyển đổi chúng thành Writables. Các Writables sau được tự động chuyển đổi:

|  |  |
| --- | --- |
| **Writable Type** | **Python Type** |
| Text | unicode str |
| IntWritable | int |
| FloatWritable | float |
| DoubleWritable | float |
| BooleanWritable | bool |
| BytesWritable | bytearray |
| NullWritable | None |
| MapWritable | dict |

Mảng không được xử lý ngoài hộp. Người dùng cần chỉ định các kiểu ArrayWritablephụ tùy chỉnh khi đọc hoặc viết. Khi viết, người dùng cũng cần chỉ định bộ chuyển đổi tùy chỉnh chuyển đổi mảng thành kiểu con tùy chỉnh ArrayWritable. Khi đọc, trình chuyển đổi mặc định sẽ chuyển đổi các kiểu con tùy chỉnh ArrayWritablesang Java Object[], sau đó được chuyển thành các bộ giá trị Python. Để lấy Python array.arraycho các mảng kiểu nguyên thủy, người dùng cần chỉ định bộ chuyển đổi tùy chỉnh.

**Lưu và tải SequenceFiles**

Tương tự như các tệp văn bản, SequenceFiles có thể được lưu và tải bằng cách chỉ định đường dẫn. Các lớp khóa và giá trị có thể được chỉ định, nhưng đối với Writables tiêu chuẩn, điều này không bắt buộc.

>>> rdd = sc.parallelize(range(1, 4)).map(**lambda** x: (x, "a" \* x))

>>> rdd.saveAsSequenceFile("path/to/file")

>>> sorted(sc.sequenceFile("path/to/file").collect())

[(1, u'a'), (2, u'aa'), (3, u'aaa')]

**Lưu và tải các định dạng đầu vào / đầu ra Hadoop khác**

PySpark cũng có thể đọc bất kỳ Hadoop InputFormat nào hoặc ghi bất kỳ Hadoop OutputFormat nào, cho cả API Hadoop MapReduce 'mới' và 'cũ'. Nếu được yêu cầu, một cấu hình Hadoop có thể được chuyển vào dưới dạng một lệnh Python. Đây là một ví dụ sử dụng Elasticsearch ESInputFormat:

$ ./bin/pyspark --jars /path/to/elasticsearch-hadoop.jar

>>> conf = {"es.resource" : "index/type"} *# assume Elasticsearch is running on localhost defaults*

>>> rdd = sc.newAPIHadoopRDD("org.elasticsearch.hadoop.mr.EsInputFormat",

"org.apache.hadoop.io.NullWritable",

"org.elasticsearch.hadoop.mr.LinkedMapWritable",

conf=conf)

>>> rdd.first() *# the result is a MapWritable that is converted to a Python dict*

(u'Elasticsearch ID',

{u'field1': True,

u'field2': u'Some Text',

u'field3': 12345})

Lưu ý rằng, nếu InputFormat chỉ phụ thuộc vào cấu hình Hadoop và / hoặc đường dẫn đầu vào và các lớp khóa và giá trị có thể dễ dàng được chuyển đổi theo bảng trên, thì cách tiếp cận này sẽ hoạt động tốt cho những trường hợp như vậy.

Nếu bạn có dữ liệu nhị phân được tuần tự hóa tùy chỉnh (chẳng hạn như tải dữ liệu từ Cassandra / HBase), thì trước tiên bạn sẽ cần phải chuyển đổi dữ liệu đó ở phía Scala / Java thành thứ có thể được xử lý bởi bộ chọn của Pyrolite. Một đặc điểm [Chuyển đổi](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/api/python/Converter.html) được cung cấp cho điều này. Chỉ cần mở rộng đặc điểm này và triển khai mã chuyển đổi của bạn trong convert phương thức. Hãy nhớ đảm bảo rằng lớp này, cùng với bất kỳ phụ thuộc nào được yêu cầu để truy cập của bạn InputFormat, được đóng gói vào lọ công việc Spark của bạn và được đưa vào đường dẫn khóa PySpark.

**Hoạt động RDD**

Các RDD hỗ trợ hai kiểu hoạt động: phép biến đổi , tạo tập dữ liệu mới từ tập dữ liệu hiện có và hành động trả về giá trị cho chương trình trình điều khiển sau khi chạy tính toán trên tập dữ liệu. Ví dụ, maplà một phép chuyển đổi chuyển từng phần tử tập dữ liệu qua một hàm và trả về một RDD mới đại diện cho kết quả. Mặt khác, reducelà một hành động tổng hợp tất cả các phần tử của RDD bằng cách sử dụng một số chức năng và trả về kết quả cuối cùng cho chương trình điều khiển (mặc dù cũng có một song song reduceByKeytrả về một tập dữ liệu phân tán).

Tất cả các phép biến đổi trong Spark đều lười biếng , ở chỗ chúng không tính toán ngay kết quả của chúng. Thay vào đó, họ chỉ nhớ các phép biến đổi được áp dụng cho một số tập dữ liệu cơ sở (ví dụ: một tệp). Các phép biến đổi chỉ được tính toán khi một hành động yêu cầu trả về kết quả cho chương trình điều khiển. Thiết kế này giúp Spark chạy hiệu quả hơn. Ví dụ, chúng ta có thể nhận ra rằng một tập dữ liệu được tạo thông qua mapsẽ được sử dụng trong một reducevà chỉ trả về kết quả của reducetrình điều khiển, thay vì tập dữ liệu được ánh xạ lớn hơn.

Theo mặc định, mỗi RDD đã chuyển đổi có thể được tính toán lại mỗi khi bạn chạy một hành động trên đó. Tuy nhiên, bạn cũng có thể duy trì một RDD trong bộ nhớ bằng cách sử dụng phương thức persist(hoặc cache), trong trường hợp này Spark sẽ giữ các phần tử xung quanh trên cụm để truy cập nhanh hơn nhiều vào lần tiếp theo bạn truy vấn nó. Ngoài ra còn có hỗ trợ cho các RDD lâu dài trên đĩa hoặc được sao chép qua nhiều nút.

1. Spark DataFrame

Tổng quat

Spark SQL là một mô-đun Spark để xử lý dữ liệu có cấu trúc. Không giống như API Spark RDD cơ bản, các giao diện do Spark SQL cung cấp cung cấp cho Spark nhiều thông tin hơn về cấu trúc của cả dữ liệu và tính toán đang được thực hiện. Bên trong, Spark SQL sử dụng thông tin bổ sung này để thực hiện các tối ưu hóa bổ sung. Có một số cách để tương tác với Spark SQL bao gồm SQL và API tập dữ liệu. Khi tính toán một kết quả, cùng một công cụ thực thi được sử dụng, không phụ thuộc vào API / ngôn ngữ bạn đang sử dụng để thể hiện tính toán. Sự thống nhất này có nghĩa là các nhà phát triển có thể dễ dàng chuyển đổi qua lại giữa các API khác nhau, dựa trên đó cung cấp cách tự nhiên nhất để thể hiện một chuyển đổi nhất định.

Tất cả những ví dụ về việc sử dụng trang này dữ liệu mẫu bao gồm trong phân phối Spark và có thể được chạy trong spark-shell, pysparkvỏ, hoặc sparkRvỏ.

SQL

Một công dụng của Spark SQL là thực thi các truy vấn SQL. Spark SQL cũng có thể được sử dụng để đọc dữ liệu từ bản cài đặt Hive hiện có. Để biết thêm về cách cấu hình tính năng này, vui lòng tham khảo phần [Hive Tables](https://spark.apache.org/docs/2.3.0/sql-programming-guide.html#hive-tables) . Khi chạy SQL từ trong một ngôn ngữ lập trình khác, kết quả sẽ được trả về dưới dạng [Dataset / DataFrame](https://spark.apache.org/docs/2.3.0/sql-programming-guide.html#datasets-and-dataframes) . Bạn cũng có thể tương tác với giao diện SQL bằng dòng [lệnh](https://spark.apache.org/docs/2.3.0/sql-programming-guide.html#running-the-spark-sql-cli) hoặc qua [JDBC / ODBC](https://spark.apache.org/docs/2.3.0/sql-programming-guide.html#running-the-thrift-jdbcodbc-server) .

Datasets và DataFrames

Tập dữ liệu là một tập hợp dữ liệu phân tán. Dataset là một giao diện mới được bổ sung trong Spark 1.6 cung cấp các lợi ích của RDD (gõ mạnh, khả năng sử dụng các hàm lambda mạnh mẽ) với các lợi ích của công cụ thực thi được tối ưu hóa của Spark SQL. Một Dataset có thể được [xây dựng](https://spark.apache.org/docs/2.3.0/sql-programming-guide.html#creating-datasets) từ vật JVM và sau đó thao tác sử dụng biến đổi chức năng ( map, flatMap, filter, vv). API tập dữ liệu có sẵn trong [Scala](https://spark.apache.org/docs/2.3.0/api/scala/index.html#org.apache.spark.sql.Dataset) và [Java](https://spark.apache.org/docs/2.3.0/api/java/index.html?org/apache/spark/sql/Dataset.html) . Python không hỗ trợ Dataset API. Nhưng do bản chất động của Python, nhiều lợi ích của Dataset API đã có sẵn (tức là bạn có thể truy cập trường của một hàng theo tên một cách tự nhiên row.columnName). Trường hợp của R cũng tương tự.

DataFrame là một Tập dữ liệu được tổ chức thành các cột được đặt tên. Về mặt khái niệm, nó tương đương với một bảng trong cơ sở dữ liệu quan hệ hoặc một khung dữ liệu trong R / Python, nhưng với các tối ưu hóa phong phú hơn. DataFrames có thể được xây dựng từ nhiều [nguồn](https://spark.apache.org/docs/2.3.0/sql-programming-guide.html#data-sources) như: tệp dữ liệu có cấu trúc, bảng trong Hive, cơ sở dữ liệu bên ngoài hoặc RDD hiện có. Các DataFrame API có sẵn trong Scala, Java, [Python](https://spark.apache.org/docs/2.3.0/api/python/pyspark.sql.html#pyspark.sql.DataFrame) , và [R](https://spark.apache.org/docs/2.3.0/api/R/index.html) . Trong Scala và Java, DataFrame được biểu diễn bằng Tập dữ liệu gồm Rows. Trong [API Scala](https://spark.apache.org/docs/2.3.0/api/scala/index.html#org.apache.spark.sql.Dataset) , DataFramechỉ đơn giản là một loại bí danh của Dataset[Row]. Trong khi, trong [Java API](https://spark.apache.org/docs/2.3.0/api/java/index.html?org/apache/spark/sql/Dataset.html) , người dùng cần sử dụng Dataset<Row>để biểu diễn a DataFrame.

Trong toàn bộ tài liệu này, chúng tôi thường đề cập đến các Tập dữ liệu Scala / Java của Rows là DataFrames.