Họ Tên: Trần Minh Phương

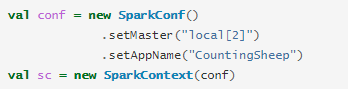
Mssv: 51800917

1. **Spark Properties**

Những thuộc tính của Spark hiện hữu hầu hết trong việc xây dựng ứng dụng, dự án thông qua việc cài đặt (setting). Đa số được tích hợp sẵn, có sẵn.

Có thể cài đặt, cấu hình một cách trực tiếp trên SparkConf() bằng SparkContent(), cùng với phương thức set()

Ví dụ:



Ở chế độ cục bộ (local) Spark hỗ trợ xử lý đa luồng (thread), việc cấp phát dữ liệu, thời gian thực thi được cấp phép, phân phối phù hợp thông qua việc cấu hình hay hệ thống mặc đinh (default).

**Một số thuộc tính có sẵn (available properties):**

spark.app.name : Tên ứng dụng của bạn. Điều này sẽ xuất hiện trong giao diện người dùng và trong dữ liệu nhật ký.

spark.driver.cores : Số lõi để sử dụng cho quy trình trình điều khiển, chỉ ở chế độ cụm.

spark.driver.maxResultSize : Giới hạn tổng kích thước của các kết quả được tuần tự hóa của tất cả các phân vùng cho mỗi hành động Spark (ví dụ: thu thập) tính bằng byte

spark.driver.memory : Dung lượng bộ nhớ sử dụng cho quá trình lái xe, tức là nơi SparkContext được khởi tạo, trong định dạng giống như chuỗi ký ức JVM với một đơn vị kích thước hậu tố ( "k", "m", "g" hoặc "t") (ví dụ 512m, 2g) .

spark.driver.resource.{resourceName}.amount : Số lượng của một loại tài nguyên cụ thể để sử dụng trên trình điều khiển. Nếu điều này được sử dụng, bạn cũng phải chỉ định spark.driver.resource.{resourceName}.discoveryScript trình điều khiển để tìm tài nguyên khi khởi động.

spark.driver.resource.{resourceName}.vendor : Nhà cung cấp tài nguyên để sử dụng cho trình điều khiển

spark.executor.memory : Dung lượng bộ nhớ sử dụng cho mỗi quá trình thi hành

spark.executor.pyspark.memory : Lượng bộ nhớ được cấp phát cho PySpark trong mỗi trình thực thi

spark.extraListeners : Danh sách các lớp được phân tách bằng dấu phẩy triển khai SparkListener

spark.driver.extraClassPath : Các mục nhập classpath bổ sung để thêm trước vào classpath của trình điều khiển.

spark.driver.extraJavaOptions : Một chuỗi các tùy chọn JVM bổ sung để chuyển cho người lái xe. Điều này là do người dùng thiết lập.

spark.driver.userClassPathFirst : Có ưu tiên các lọ do người dùng thêm vào các lọ của chính Spark khi tải các lớp trong trình điều khiển hay không

spark.executor.extraLibraryPath : Đặt một đường dẫn thư viện đặc biệt để sử dụng khi khởi chạy JVM của trình thực thi.

spark.executor.logs.rolling.maxSize : Đặt kích thước tối đa của tệp theo byte mà nhật ký trình thực thi sẽ được cuộn qua.

spark.files : Danh sách tệp được phân tách bằng dấu phẩy sẽ được đặt trong thư mục làm việc của mỗi trình thực thi

spark.submit.pyFiles : Danh sách các tệp .zip, .egg hoặc .py được phân tách bằng dấu phẩy để đặt trên ứng dụng PYTHONPATH cho Python

spark.jars.packages : Danh sách được phân tách bằng dấu phẩy gồm các tọa độ Maven của các chum để đưa vào các đường dẫn của trình điều khiển và trình thực thi

spark.pyspark.python : Thực thi nhị phân Python để sử dụng cho PySpark trong cả trình điều khiển và trình thực thi.

spark.pyspark.driver.python: Thực thi nhị phân Python để sử dụng cho PySpark trong trình điều khiển. (mặc định là spark.pyspark.python)

spark.sql.adaptive.advisoryPartitionSizeInBytes : Kích thước tư vấn tính bằng byte của phân vùng xáo trộn trong quá trình tối ưu hóa thích ứng (khi spark.sql.adaptive.enabled là true)

spark.sql.adaptive.coalescePartitions.initialPartitionNum : Số lượng phân vùng xáo trộn ban đầu trước khi kết hợp lại.

spark.sql.adaptive.coalescePartitions.enabled : Khi true và 'spark.sql.adaptive.enabled' là true, Spark sẽ kết hợp các phân vùng xáo trộn liền kề theo kích thước mục tiêu (được chỉ định bởi 'spark.sql.adaptive.advisoryPartitionSizeInBytes'), để tránh quá nhiều tác vụ nhỏ.

spark.sql.csv.filterPushdown.enabled : Khi đúng, hãy bật bộ lọc đẩy xuống nguồn dữ liệu CSV.

spark.sql.debug.maxToStringFields : Số lượng trường tối đa của các mục nhập giống như chuỗi có thể được chuyển đổi thành chuỗi trong đầu ra gỡ lỗi

spark.sql.streaming.checkpointLocation : Vị trí mặc định để lưu trữ dữ liệu điểm kiểm tra cho các truy vấn trực tuyến

spark.sql.variable.substitute : Điều này cho phép thay thế sử dụng cú pháp như ${var}, ${system:var}, và ${env:var}.

spark.sql.streaming.numRecentProgressUpdates: Số lượng bản cập nhật tiến trình cần giữ lại cho một truy vấn phát trực tuyến

**Tài liệu tham khảo (link):**

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

1. **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.

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

Spark 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:**

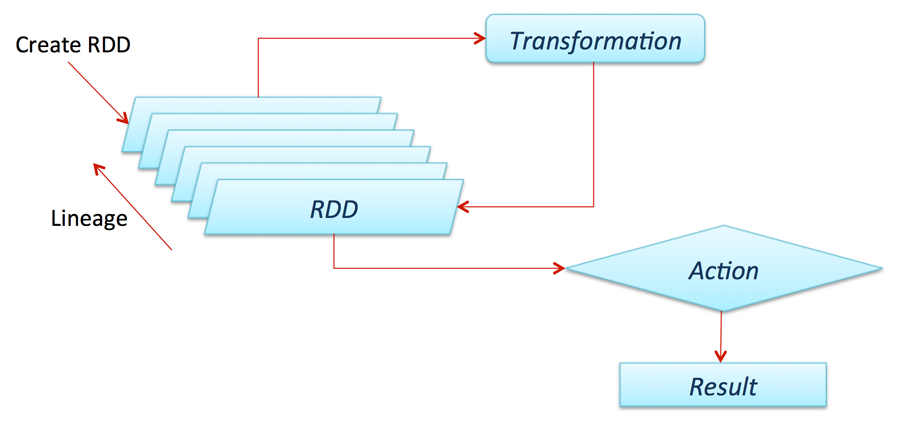


**Iteractive Operations trên Spark RDD:**



**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.



**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

Ví dụ1 ([python] using google colab):

!pip install pyspark

from pyspark import SparkContext, SparkConf

conf = SparkConf().setMaster("local").setAppName("word counting")

sc = SparkContext.getOrCreate(conf=conf)

data = [('a', 5), ('b', 2), ('1', 4), ('d', 3), ('2', 1)]

distData = sc.parallelize(data)

distData = distData.sortBy(lambda x: x[0])

distData.collect()

>>> [('1', 4), ('2', 1), ('a', 5), ('b', 2), ('d', 3)]

distData = distData.sortBy(lambda x: x[1])

distData.collect()

>>> [('2', 1), ('b', 2), ('d', 3), ('1', 4), ('a', 5)]

Ví dụ2 ([python] using google colab):

!pip install pyspark

import pyspark

from pyspark import SparkConf, SparkContext

import collections

conf = SparkConf().setMaster("local").setAppName("word counting")

sc = SparkContext.getOrCreate(conf = conf)

text = sc.textFile("/content/drive/MyDrive/Colab Notebooks/source2.txt")

print(text.collect())

>>> ['1 3 9 5 2 4 10 8']

text = text.flatMap(lambda x: (x.split(" ")))

print(text.collect())

>>> ['1', '3', '9', '5', '2', '4', '10', '8']

max = text.max()

print(max)

9

**Tài liệu tham khảo (link):**

https://www.tutorialspoint.com/apache\_spark/apache\_spark\_rdd.htm

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

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

https://www.youtube.com/watch?v=9bn9Kf82Hs8

https://www.youtube.com/watch?v=jce3Vf5d3WI

https://www.youtube.com/watch?v=ZSxaqorckr8

1. **Pyspark DataFrame**

DataFrames là một từ thông dụng trong ngành công nghiệp ngày nay. Mọi người có xu hướng sử dụng nó với các ngôn ngữ phổ biến được sử dụng cho Phân tích dữ liệu như Python, Scala và R

DataFrames thường đề cập đến một cấu trúc dữ liệu, về bản chất là dạng bảng. Nó đại diện cho các hàng, mỗi hàng bao gồm một số quan sát. Các hàng có thể có nhiều định dạng dữ liệu (không đồng nhất), trong khi một cột có thể có dữ liệu cùng loại (đồng nhất). DataFrames thường chứa một số siêu dữ liệu ngoài dữ liệu; ví dụ, tên cột và hàng



**3.1 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.

* 1. **Chia nhỏ**

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.

* 1. **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.

* 1. **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.

**DataFrames trong Pyspark có thể được tạo theo nhiều cách:**



Dữ liệu có thể được tải thông qua **tệp CSV, JSON, XML**  hoặc tệp Parquet. Nó cũng có thể được tạo bằng **RDD** hiện có và thông qua bất kỳ cơ sở dữ liệu nào khác, như **Hive** hoặc **Cassandra** . Nó cũng có thể lấy dữ liệu từ HDFS hoặc hệ thống tệp cục bộ

Ví dụ1: Tạp Dataframe cơ bản từ list dữ liệu [python] using google colab

!pip install pyspark

import pyspark

from pyspark import SparkContext

sc = SparkContext()

sqlContext = SQLContext(sc)

from pyspark.sql import \*

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

# Tạo các đối tượng Employee từ đó tạo đối tượng Department

# Khởi tạo title

Employee = Row("firstName", "lastName", "email", "salary")

employee1 = Employee('Basher', 'armbrust', 'bash@edureka.co', 100000)

employee2 = Employee('Daniel', 'meng', 'daniel@stanford.edu', 120000 )

employee3 = Employee('Muriel', None, 'muriel@waterloo.edu', 140000 )

employee4 = Employee('Rachel', 'wendell', 'rach\_3@edureka.co', 160000 )

employee5 = Employee('Zach', 'galifianakis', 'zach\_g@edureka.co', 160000 )

department1 = Row(id='123456', name='HR')

department2 = Row(id='789012', name='OPS')

department3 = Row(id='345678', name='FN')

department4 = Row(id='901234', name='DEV')

# Tạo các hàng (Row) dữ liệu

departmentWithEmployees1 = Row(department=department1, employees=[employee1, employee2, employee5])

departmentWithEmployees2 = Row(department=department2, employees=[employee3, employee4])

departmentWithEmployees3 = Row(department=department3, employees=[employee1, employee4, employee3])

departmentWithEmployees4 = Row(department=department4, employees=[employee2, employee3])

# Tạo DataFrame:

departmentsWithEmployees\_Seq = [departmentWithEmployees1, departmentWithEmployees2]

dframe = spark.createDataFrame(departmentsWithEmployees\_Seq)

# Hiển thị DataFrame

display(dframe)

dframe.show()

**Output:**

DataFrame[department: struct<id:string,name:string>, employees: array<struct<firstName:string,lastName:string,email:string,salary:bigint

>>]

+-------------+--------------------+

| department| employees|

+-------------+--------------------+

| [123456, HR]|[[Basher, armbrus...|

|[789012, OPS]|[[Muriel,, muriel...|

+-------------+--------------------+

**Tài liệu tham khảo (link):**

https://sparkbyexamples.com/pyspark/pyspark-create-dataframe-from-list/

https://www.tutorialspoint.com/spark\_sql/spark\_sql\_dataframes.htm

https://www.edureka.co/blog/pyspark-dataframe-tutorial/#creation