**Bài tập 2: TÌM HIỂU SPARK PROPERTIES, RDD VÀ DATAFRAME**

**1. Spark properties**

Spark cung cấp ba lĩnh vực chính để cấu hình: Thuộc tính Spark (Spark properties), Các biến môi trường (Environment Variables) và Ghi nhật ký (Logging).

***Spark properties***: kiểm soát hầu hết các cài đặt của ứng dụng (application) và được cấu hình riêng cho từng ứng dụng. Các thuộc tính này có thể được thiết lập trực tiếp trên SparkConf và chuyển đến SparkContext. SparkConf cho phép chúng ta cấu hình hầu hết các thuộc tính chung như URL và tên ứng dụng, chúng ta cũng có thể thiết lập các cặp giá trị khóa (key-value) qua phương thức set().

Ví dụ code minh họa tạo một application với 2 luồng:

v val conf = new SparkConf()

.setMaster("local[2]")

.setAppName("countingWord")

val sc = new SparkContext(conf)

với local[2] là xử lý 2 luồng.

Code minh họa python khởi tạo Spark:

conf = SparkConf().setMaster("local").setAppName("My SPARK app")

sc = SparkContext(conf=conf)

***SparkConf()***: Tạo một SparkConf tải các giá trị mặc định từ các system properties và classpath.

***local***: Chạy Spark cục bộ với một luồng duy nhất (tức là không có luồng song song nào cả). Có thể thêm giá trị sau chữ local để chạy nhiều luồng song song ví dụ: local [n] để chạy Spark cục bộ với số luồng là n, local[\*] để chạy spark cục bộ với số luồng tương ứng với số core trên máy Java virtual machine.

***set(“spark.executor.memory”, “1g”)***: cài đặt số lượng bộ nhớ được sử dụng cho mỗi quá trình thực thi là 1g.

Hầu hết các thuộc tính kiểm soát cài đặt nội bộ đều có giá trị mặc định sẵn. Một số thuộc tính phổ biến:

* spark.app.name: Tên ứng dụng, mặc định là chưa có, xuất hiện trong giao diện người dùng và dữ liệu nhật ký (log data).
* spark.executor.memory: Số lượng bộ nhớ được sử dụng cho mỗi quá trình thực thi, mặc định là 1g.
* spark.master: URL chính để kết nối tới, mặc định là chưa có.
* spark.serializer: Class được sử dụng để tuần tự hóa các đối tượng sẽ được gửi qua mạng hoặc cần được lưu vào bộ nhớ đệm ở dạng tuần tự hóa. Mặc định là org.apache.spark.serializer.JavaSerializer nhưng thực tế nên sử dụng class org.apache.spark.serializer.KryoSerializer để có được tốc độ tốt hơn class mặc định của tuần tự hóa.
* spark.kryo.registrator: Class được sử dụng để đăng ký các lớp tùy chỉnh nếu chúng ta sử dụng tuần tự hóa Kyro, mặc định là chưa được cài đặt.
* spark.local.dir: Đường dẫn thư mục các cho không gian đầu trong spark để lưu trữ các tệp đầu và RDDs vào ổ đĩa, đây nên là ổ đĩa nhanh hoặc local disk, mặc định đường dẫn là: /tmp.
* spark.cores.max: Được sử dụng ở chế độ độc lập để chỉ định số lượng lõi CPU tối đa để yêu cầu cho ứng dụng từ toàn bộ cụm. Nếu không được đặt, mặc định sẽ nằm spark.deploy.defaultCorestrên trình quản lý cụm độc lập của Spark hoặc vô hạn (tất cả các lõi có sẵn) trên Mesos.

Với các Available Properties (thuộc tính có sẵn) ta có thể sử dụng để cài đặt cho các ứng dụng, môi trường thực thi, giao diện người dùng, nén và tuần tự hóa, bảo mật, quản lý bộ nhớ, Spark Streaming, SparkR, GraphX, Cluster Managers (Yarn, Mesos),…

**Nén và tuần tự hoá *(Compression and Serialization)***

spark.rdd.compress - Có nén các phân vùng tuần tự

Ví dụ: StorageLevel.MEMORY\_ONLY\_SERtrong Java và Scala hoặc StorageLevel.MEMORY\_ONLY trong Python). Có thể tiết kiệm không gian đáng kể với chi phí tăng thêm thời gian CPU. Nén sẽ sử dụng tới thuộc tính spark.io.compression.codec.

Các thuộc tính khác:

* spark.serializer
* spark.serializer.objectStreamReset
* spark.kryoserializer.buffer
* spark.kryo.registrator
* spark.kryo.referenceTracking, ...

**Các loại thuộc tính khác**

Ngoài các loại thuộc tính trên, Spark còn hỗ trợ nhiều loại thuộc tính khác nhau:

* Môi trường thực thi (Runtime Environment)
* Quản lý bộ nhớ (Memory Management)
* Hành vi thực thi (Execution Behavior)
* Chỉ số thực thi (Executor Metrics)
* Kết nối mạng (Networking)
* Lập lịch (Scheduling)
* Chế độ thực thi rào cản (Barrier Execution Mode)
* Phân bố động (Dynamic Allocation)
* Cấu hình Thread (Thread Configurations)
* Bảo mật (Security)

Có thể tìm thêm nhiều spark available properties có sẵn tại trang web: <https://spark.apache.org/docs/latest/configuration.html>

**2. Spark RDD**

RDD (Resilient Distributed Dataset) hay tập dữ liệu phân tán có khả năng phục hồi là một cấu trúc dữ liệu cơ bản của Spark và là trừu tượng hóa dữ liệu (data abstraction) chính trong Apache Spark và Spark Core. Nó là một tập hợp các đối tượng phân tán bất biến không thay đổi và được phân vùng, chỉ có thể được tạo bởi các hoạt động chi tiết thô như bản đồ (map), bộ lọc (filter), nhóm. Mỗi dataset trong RDD được chia thành các phân vùng logical có thể được tính toán trên các node khác nhau của cụm máy chủ. Chúng có thể hoạt động song song và có khả năng chịu lỗi.

Các đối tượng RDD có thể được tạo bằng Python, Java hoặc Scala. Nó cũng có thể bao gồm các lớp do người dùng định nghĩa. RDD cung cấp tính trừu tượng hóa dữ liệu cho việc phân vùng dữ liệu, phân phối dữ liệu được thiết kế để tính toán song song trên các node, khi thực hiện các phép biến đổi trên RDD sự song song luôn được đảm bảo do Spark cung cấp theo mặc định.

**Tạo RDD:**

Có hai cách để tạo RDD:

* Song song hóa một tập hợp dữ liệu hiện có trong chương trình trình điều khiển Spark Context có sẵn : Các tập hợp song song được tạo bằng cách gọi phương thức song song hóa của lớp JavaSparkContext trong chương trình điều khiển.
* Tham chiếu tập dữ liệu trong hệ thống lưu trữ bên ngoài có thể là HDFS, Hbase, các cơ sở dữ liệu quan hệ hoặc bất kỳ nguồn nào có định dạng tệp Hadoop.

Lưu ý: trước khi tạo RDD ta phải khởi tạo spark bằng code python sau:

import pyspark

from pyspark import SparkConf, SparkContext

import collections

conf= SparkConf().setMaster('local').setAppName('My spark app')

sc= SparkContext.getOrCreate(conf=conf)

Ví dụ code python tạo RDD bằng cách song song hóa một tập dữ liệu:

data = [1, 10, 12, 8, 4]

rdd = sc.parallelize(data)

Ở đây sparkContext.parallelize được sử dụng để song song hóa một tập hợp hiện có trong chương trình trình điều khiển. Đây là một phương pháp cơ bản để tạo RDD nó yêu cầu tất cả dữ liệu phải có trên chương trình trình điều khiển trước khi tạo. Do đó với các ứng dụng sản xuất nên sử dụng cách tham chiếu tập dữ liệu trong hệ thống lưu trữ bên ngoài.

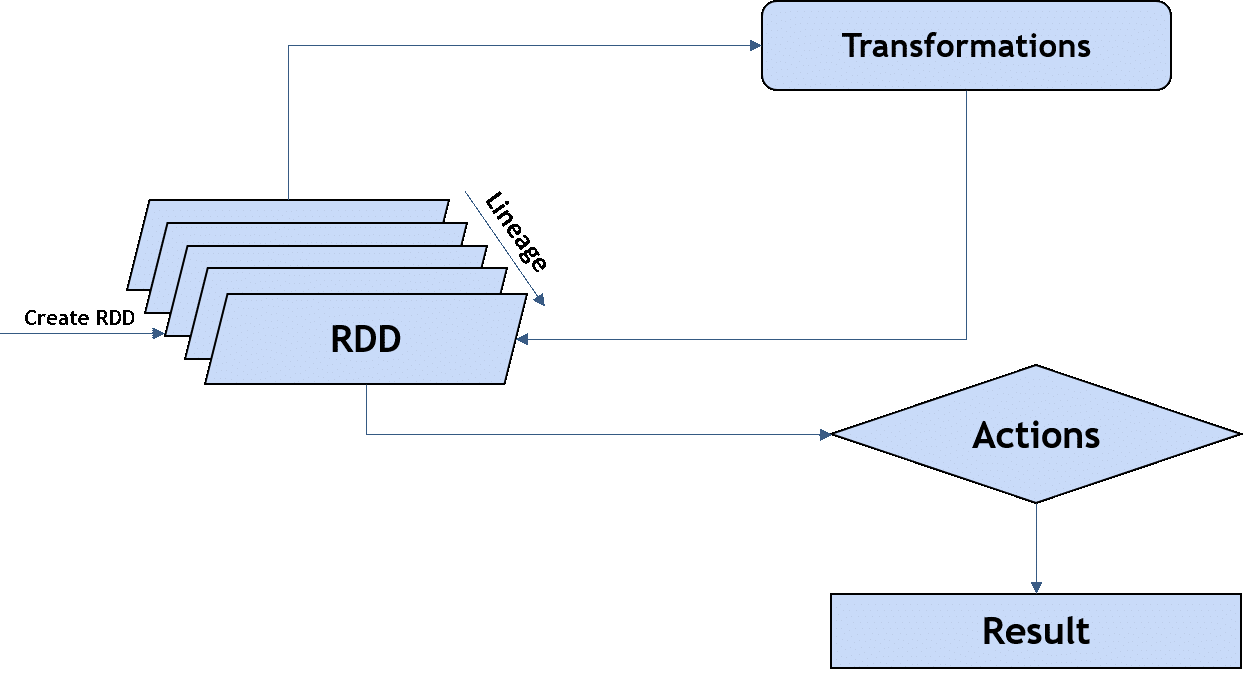
Ví dụ code python tạo RDD bằng cách tham chiếu tập dữ liệu trong hệ thống lưu trữ bên ngoài bằng phương thức sparkContext.textFile():

rdd = sc.textFile("/path/textFile.txt")

Một số lưu ý khi đọc tệp với spark:

* Nếu sử dụng một đường dẫn trên local filesystem, tệp phải có thể truy cập được tại cùng một đường dẫn trên các node đang làm việc.
* Tất cả các phương thức tham chiếu tệp bao gồm textFile, hỗ trợ chạy trên thư mục, tệp nén và cả ký tự đại diện (wildcards).
* Phương thức textFile cũng có một đối số tùy chọn thứ hai để kiểm soát số lượng các phân vùng của tập tin chỉ cần lưu ý không thể có ít phân vùng (partitions) hơn khối (blocks). Ngoài phương thức textFile thì API Python của Spark cũng hỗ trợ một số định dạng dữ liệu khác như với phương thức SparkContext.wholeTextFiles ta có thể đọc một thư mục chứa nhiều tệp văn bản nhỏ và trả về mỗi tệp dưới dạng cặp (tên tệp, nội dung). Phương thức RDD.saveAsPickleFile và SparkContext.pickleFile lưu RDD ở một định dạng đơn giản bao gồm các đối tượng Python có sẵn.

**Hoạt động RDD:**

RDD hỗ trợ hai loại hoạt động là Transformations và Actions. Ảnh minh họa 2 loại hoạt động cơ bản có thể sử dụng với RDD: [](https://camo.githubusercontent.com/9dfba805294905a8e65ae1eedb911de87f4d0c6fe6a3f3f981354aa825d0d2b6/68747470733a2f2f7472756f6e676e677579656e7068696c6f6e672e6769746875622e696f2f58754c7944754c6965754c6f6e2f424149544150322f416e68312e706e67)Transformations: Một tập dữ liệu (dataset) mới được tạo từ một tập dữ liệu hiện có. Mỗi tập dữ liệu được chuyển qua một hàm. Kết quả giá trị trả về là nó sẽ gửi một RDD mới.

Actions: Trả về giá trị cho chương trình điều khiển sau khi thực thi mã trên tập dữ liệu, thực hiện các tính toán trên tập dữ liệu cần thiết. RDD trả về các giá trị không phải RDD. Các giá trị này được lưu trữ trên hệ thống bên ngoài.

Một số Transformation:

* distinct: loại bỏ trùng lắp trong RDD.
* map: Trả về một RDD mới bằng cách áp dụng hàm trên từng phần tử dữ liệu. Trong Python sử dụng lambda với từng phần tử để truyền vào map.
* filter: Trả về một RDD mới được hình thành bằng cách chọn các phần tử của nguồn mà hàm trả về true.

**Các hành động RDD *(RDD Actions)***

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.

Một số RDD actions:

* **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.

**Ưu điểm của RDD**

Các đặc tính, ưu điểm chính:

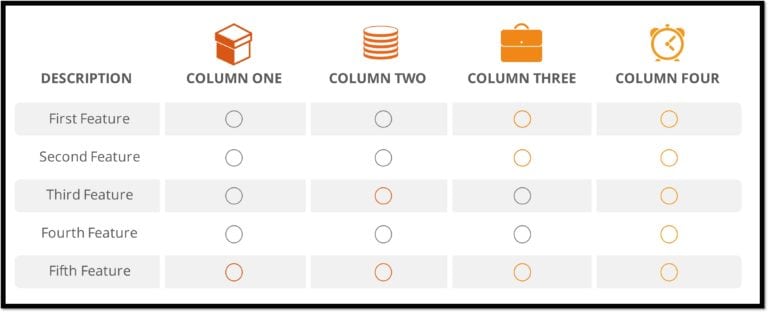
* Xử lý trong bộ nhớ: Dữ liệu bên trong RDD được lưu trữ trong bộ nhớ chứ không phải đĩa, do đó tăng tốc độ thực thi của Spark vì dữ liệu đang được tìm nạp từ dữ liệu trong bộ nhớ
* Tính bất biến: RDD được tạo ra không thể được sửa đổi. Bất kỳ biến đổi nào trên nó sẽ tạo ra một RDD mới.
* Khả năng chịu lỗi: Bất kỳ hoạt động RDD nào không thành công, nó sẽ có thể tính toán lại phân vùng bị mất của RDD từ bản gốc.
* Tiến hóa lười biếng: Dữ liệu bên trong RDD được giữ lại và chỉ đánh giá khi một hành động được kích hoạt.
* Phân vùng: Các RDD được chia thành các phần nhỏ hơn gọi là phân vùng mặc định, nó phân vùng theo số lượng lõi có sẵn.
* Tính bền bỉ: Việc có thể được tái sử dụng khiến chúng trở nên bền bỉ.
* Không có giới hạn: Không có giới hạn về số lượng RDD có thể có bao nhiêu tùy thích chỉ phụ thuộc vào kích thước của bộ nhớ và ổ đĩa.

**Nhược điểm của RDD**

* Spark RDD không phù hợp nhiều cho các ứng dụng thực hiện cập nhật cho kho lưu trữ trạng thái như hệ thống lưu trữ cho ứng dụng web.
* Một RDD chỉ có thể có trong một SparkContext và RDD có thể có tên và mã định danh duy nhất (id).

**3. Spark DataFrame**

DataFrame là một tập hợp dữ liệu phân tán 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 được tối ưu hóa phong phú hơn. DataFrames có thể được xây dựng từ nhiều nguồn như tệp dữ liệu có cấu trúc, bảng trong Hive, cơ sở dữ liệu bên ngoài hoặc RDD hiện có.



# ĐẶC ĐIỂM DATAFRAME

* Các khung dữ liệu được phân phối trong tự nhiên, điều này làm cho nó có khả năng chịu lỗi và cấu trúc dữ liệu có sẵn cao.
* Đánh giá lười biếng là một chiến lược đánh giá giữ việc đánh giá một biểu thức cho đến khi giá trị của nó là cần thiết. Nó tránh đánh giá lặp lại. Đánh giá lười biếng trong Spark có nghĩa là quá trình thực thi sẽ không bắt đầu cho đến khi một hành động được kích hoạt. Trong Spark, bức tranh về sự lười biếng xuất hiện khi các phép biến đổi Spark xảy ra.
* Dataframe có bản chất là Bất biến. Không thay đổi được, 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.

**Các tính năng của DataFrame**

* Khả năng xử lý dữ liệu có kích thước từ Kilobyte (Kb) đến Petabyte (PB) trên một cụm node đơn đến cụm lớn.
* Hỗ trợ các định dạng dữ liệu (Avro, csv, …) và hệ thống lưu trữ khác nhau (HDFS, bảng HIVE, mysql, ....).
* Tối ưu hóa hiện đại và tạo mã thông qua trình tối ưu hóa Spark SQL Catalyst (tree transformation framework).
* Có thể dễ dàng tích hợp với tất cả các công cụ và framework xử lý dữ liệu lớn thông qua Spark-Core.
* Cung cấp API cho Python, Java, Scala và R.

**Tạo DataFrame**

Cách đơn giản nhất để tạo DataFrame là từ bộ sưu tập seq. DataFrame cũng có thể được tạo từ RDD và bằng cách đọc các tệp từ một số nguồn.

Ví dụ code python tạo DataFrame:

# Tạo dataframe từ bảng users trong bảng Hive.

users = context.table("users")

# từ các tệp JSON trong S3

logs = context.load("s3n://path/to/data.json", "json")

# Tạo một DataFrame chỉ chứa những người trẻ dưới 21 tuổi

young = users.filter(users.age < 21)

## sử dụng createDataFrame()

import pyspark

from pyspark import SparkConf, SparkContext

from pyspark.sql import SparkSession

import collections

spark = SparkSession.builder.master("local[2]").appName("dataframetest").getOrCreate()

data = [('1','john','male'),

('2','scottman','male'),

('3','joe','male','M')

]

columns = ["id","name","gender"]

df = spark.createDataFrame(data=data, schema = columns)

df.show()

**TÀI LIỆU THAM KHẢO**

1. <https://spark.apache.org/docs/latest/configuration.html>
2. <https://spark.apache.org/docs/latest/rdd-programming-guide.html>
3. <https://sparkbyexamples.com/>
4. <https://laptrinh.vn/books/apache-spark/page/apache-spark-rdd>
5. <https://helpex.vn/article/rdd-trong-spark-la-gi-va-tai-sao-chung-ta-can-no-5c6afe5bae03f628d053a84c>
6. <https://www.educba.com/what-is-rdd/>
7. <https://intellipaat.com/blog/tutorial/spark-tutorial/programming-with-rdds/>
8. <https://codetudau.com/xu-ly-du-lieu-voi-spark-dataframe/index.html>
9. <https://www.tutorialspoint.com/spark_sql/spark_sql_dataframes.htm>