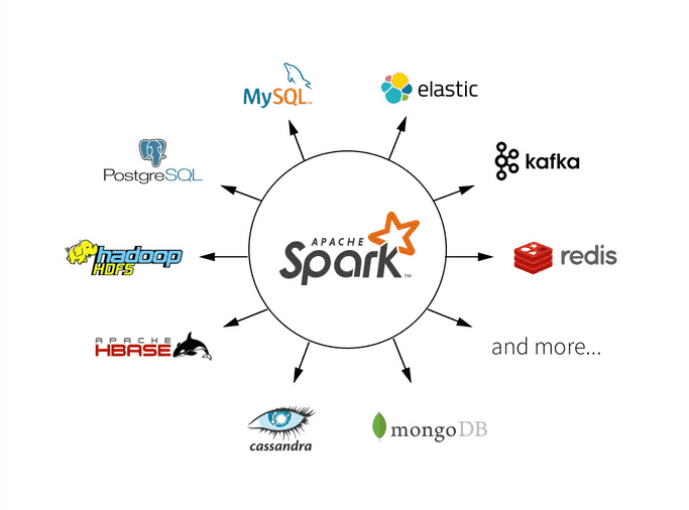
SPARK

Nguyễn Văn Đạt – 51800364
30/01/2021

1. Thông tin về Spark:

Spark là một hệ thống quy trình làm việc (workflow system). Spark là một tiến bộ với các hệ thống quy trình làm việc trước đây, có thể kể đến các vai trò sau:

- Một cách hiệu quả hơn để đối phó với thất bại (coping with failures).
- Một cách hiệu quả hơn để nhóm các nhiệm vụ giữa các nút tính toán và lập lịch (scheduling) thực hiện các chức năng.
- Tích hợp các tính năng của ngôn ngữ lập trình như vòng lặp (về mặt kỹ thuật đưa nó ra khỏi lớp quy trình làm việc tuần hoàn của hệ thống) và các thư viện.



2. Tìm hiểu các vấn đề thuộc Spark:

2.1. Spark properties:[1]

Spark properties kiểm soát hầu hết các cài đặt và được cấu hình riêng cho từng loại ứng dụng. Các properties (thuộc tính) này có thể được thiết lập bằng SparkConf (Cấu hình cho một ứng dụng Spark) hoặc thông qua các thuộc tính hệ thống Java.

Các thuộc tính này có thể được đặt trực tiếp trên SparkConf được chuyển đến SparkContext của bạn. SparkConf cho phép bạn định cấu hình một số thuộc tính phổ biến (ví dụ: URL chính và tên ứng dụng), cũng như các cặp khóa-giá trị tùy ý thông qua phương thức set().

Ví dụ: khởi tạo một ứng dụng với hai luồng như sau:

```
val conf = new SparkConf().setMaster("local[2]").setAppName(‘CountingSheep’)
val sc = new SparkContext(conf)
```

Trị số 2 tại local[2] thể hiện hai luồng, đại diện cho sự song song. Ngoài ra có thể có nhiều hơn 1 luồng ở chế độ cục bộ và trong những trường hợp như Spark Streaming, chúng ta thực sự có thể yêu cầu nhiều hơn 1 luồng để ngăn chặn bất kỳ loại vấn đề chết đói nào.

Spark properties 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 lập trình thông qua SparkConf trong thời gian chạy, hoặc hành vi phụ thuộc vào trình quản lý cụm và chế độ triển khai bạn chọn, vì vậy bạn nên đặt thông qua tệp cấu hình hoặc tùy chọn dòng lệnh spark-submit; một cái 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.

2.1.1. Tải động các thuộc tính Spark:

Trong một số trường hợp, khi muốn tránh mã hóa cứng các cấu hình nhất định trong a SparkConf. Ví dụ: khi muốn chạy cùng một ứng dụng với các bản gốc khác nhau hoặc số lượng bộ nhớ khác nhau. Spark cho phép chỉ cần tạo một conf trống:

```
val sc = new SparkContext(new SparkConf())
```

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-submithoặc spark-shell, sau đó là các tùy chọn trong tệp sparkdefaults.conf. 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.

2.1.2. Xem thuộc tính Spark:

Giao diện người dùng web ứng dụng tại http://:4040 liệt kê các thuộc tính Spark trong tab "Environment". 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, SparkConf hoặc dòng lệnh sẽ xuất hiện.

2.1.3. Thuộc tính ứng dụng:

Một số thuộc tính ứng dụng:

Property Name	Meaning
spark.driver.cores	Số lõi để sử dụng cho quy trình trình điều khiển, chỉ ở chế độ cụm.
spark.app.name	Tên ứng dụng.
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. Tối thiểu phải là 1M hoặc 0 cho không giới hạn. Công việc sẽ bị hủy bỏ nếu tổng kích thước vượt quá giới hạn này. Có giới hạn cao có thể gây ra lỗi hết bộ nhớ trong trình điều khiển (phụ thuộc vào spark.driver.memory và chi phí bộ nhớ của

	các đối tượng trong JVM). Đặt giới hạn thích hợp có thể bảo vệ trình điều khiển khỏi lỗi hết bộ nhớ.
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.memoryOverhead	Số lượng bộ nhớ không phải heap sẽ được cấp phát cho mỗi quá trình trình điều khiển ở chế độ cụm, trong MiB trừ khi được chỉ định khác. Đây là bộ nhớ chiếm những thứ như tổng chi phí VM, chuỗi được thực hiện, các chi phí chung khác, v.v. Điều này có xu hướng phát triển theo kích thước vùng chứa (thường là 6- 10%). Tùy chọn này hiện được hỗ trợ trên YARN, Mesos và Kubernetes.

Bảng 1. Một số thuộc tính ứng dụng thuộc Spark properties

2.1.4. Môi trường thực thi:

Một số môi trường thực thi:

Property Name	Meaning
spark.driver.extraClassPath	Các mục nhập classpath bổ sung để thêm trước vào classpath của trình điều khiển
spark.driver.defaultJavaOptions	Một chuỗi các tùy chọn JVM mặc định để thêm vào spark.driver.extraJavaOptions. Điều này được thiết lập bởi các quản trị viên. Ví dụ: cài đặt GC hoặc ghi nhật ký khác. Lưu ý rằng việc đặt cài đặt kích thước đống tối đa (-Xmx) với tùy chọn này là bất hợp pháp. Có thể đặt cài đặt kích thước đống tối đa spark.driver.memorytrong chế độ cụm và thông qua --driver-memorytùy chọn dòng lệnh trong chế độ khách.
spark.driver.extraJavaOptions	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. Tối thiểu phải là 1M hoặc 0 cho không giới hạn. Công việc sẽ bị hủy bỏ nếu tổng kích thước vượt quá giới hạn này. Có giới hạn cao có thể gây ra lỗi hết bộ nhớ

	trong trình điều khiển (phụ thuộc vào spark.driver.memory và chi phí bộ nhớ của các đối tượng trong JVM). Đặt giới hạn thích hợp có thể bảo vệ trình điều khiển khỏi lỗi hết bộ nhớ.
spark.driver.extraLibraryPath	Đặt một đường dẫn thư viện đặc biệt để sử dụng khi khởi chạy trình điều khiển JVM.
spark.driver.userClassPathFirst	(Thử nghiệm) 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. Tính năng này có thể được sử dụng để giảm thiểu xung đột giữa phụ thuộc của Spark và phụ thuộc của người dùng. Nó hiện là một tính năng thử nghiệm. Điều này chỉ được sử dụng trong chế độ cụm.

Bảng 2. Một số môi trường thực thi thuộc Spark properties

2.1.5. Hành vi xáo trộn:

Một số hành vi xáo trộn:

Property Name	Meaning
spark.reducer.maxSizeInFlight	Kích thước tối đa của đầu ra bản đồ để tìm nạp đồng thời từ mỗi tác vụ giảm, trong MiB trừ khi được chỉ định khác. Vì mỗi đầu ra yêu cầu chúng ta tạo một bộ đệm để nhận nó, điều này đại diện cho chi phí bộ nhớ cố định cho mỗi tác vụ giảm, vì vậy hãy giữ nó nhỏ trừ khi bạn có một lượng lớn bộ nhớ.
spark.reducer.maxReqsInFlight	Cấu hình này giới hạn số lượng yêu cầu từ xa để tìm nạp các khối tại bất kỳ điểm nhất định nào. Khi số lượng máy chủ trong cụm tăng lên, nó có thể dẫn đến số lượng rất lớn các kết nối gửi đến một hoặc nhiều nút, khiến các công nhân bị lỗi khi tải. Bằng cách cho phép nó giới hạn số lượng yêu cầu tìm nạp, tình huống này có thể được giảm thiểu.
spark.reducer.maxBlocksInFlightPerAddress	Cấu hình này giới hạn số lượng khối từ xa được tìm nạp cho mỗi tác vụ giảm từ một cổng máy chủ nhất định. Khi một số lượng lớn các khối đang được yêu cầu từ một địa chỉ nhất định trong một lần tìm nạp hoặc đồng thời, điều này có thể làm hỏng trình

	thực thi phục vụ hoặc Trình quản lý nút. Điều này đặc biệt hữu ích để giảm tải trên Node Manager khi bật chế độ trộn bên ngoài. Bạn có thể giảm thiểu vấn đề này bằng cách đặt nó thành một giá trị thấp hơn.
spark.shuffle.compress	Có nén các tệp đầu ra bản đồ hay không. Nói chung là một ý kiến hay. Nén sẽ sử dụng spark.io.compression.codec
spark.shuffle.file.buffer	Kích thước của bộ đệm trong bộ nhớ cho mỗi luồng đầu ra tệp trộn, trong KiB trừ khi được chỉ định khác. Các bộ đệm này làm giảm số lần tìm đĩa và các lệnh gọi hệ thống được thực hiện trong việc tạo các tệp ngẫu nhiên trung gian.

Bảng 3. Một số hành vi xáo trộn thuộc Spark properties

2.1.6. Giao diện người dùng Spark:

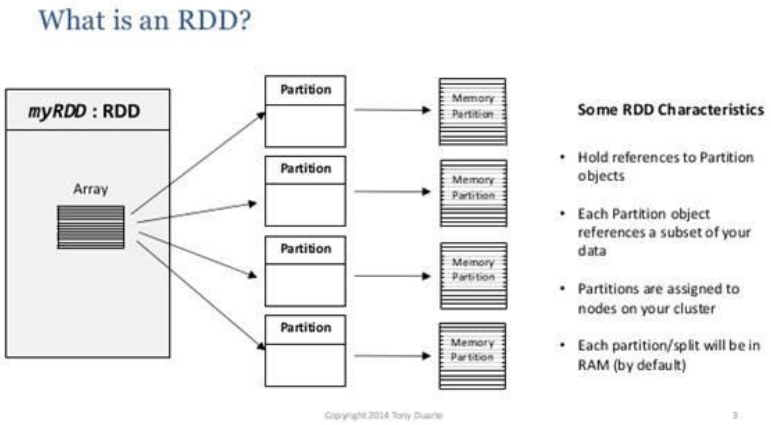
Một số giao diện người dùng Spark:

Property Name	Meaning
spark.eventLog.logBlockUpdates.enabled	Có ghi lại các sự kiện cho mỗi lần cập nhật khối hay không, nếu spark.eventLog.enabled đúng.
spark.eventLog.longForm.enabled	Nếu đúng, hãy sử dụng biểu mẫu dài của các trang web cuộc gọi trong nhật ký sự kiện. Nếu không, hãy sử dụng mẫu ngắn
spark.eventLog.compress	Có nén các sự kiện đã ghi, nếu spark.eventLog.enabledđúng.
spark.eventLog.compression.codec	Codec để nén các sự kiện đã ghi. Nếu điều này không được đưa ra, spark.io.compression.codecsẽ được sử dụng.
spark.eventLog.erasureCoding.enabled	Cho phép nhật ký sự kiện sử dụng mã hóa xóa hay tắt mã hóa xóa, bất kể giá trị mặc định của hệ thống tệp. Trên HDFS, các tệp được mã hóa xóa sẽ không cập nhật nhanh như các tệp sao chép thông thường, do đó, các bản cập nhật ứng dụng sẽ mất nhiều thời gian hơn để xuất hiện trong Máy chủ Lịch sử.

Bảng 4. Một số giao diện người dùng thuộc Spark properties

2.2. Spark RDD:[2]

Dữ liệu trung tâm của Spark được gọi là Resilient Distributed Dataset (Tập dữ liệu phân tán có khả năng phục hồi), viết tắt là RDD. 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 bằng 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.



2.2.1. Parallelized Collections (Bộ sưu tập song song):

Bộ sưu tập song song được tạo ra bằng cách gọi phương thức parallelize của một SparkContext trên một trên một bộ sưu tập hoặc một bộ sưu tập hiện có. Các phần tử của bộ sưu tập được sao chép để tạo thành một tập dữ liệu phân tán có thể được vận hành song song. Ví dụ, sau đây là cách tạo một bộ sưu tập song song chứa các phần tử của một mảng

```
data = [‘a’, ‘b’, ‘c’, ‘d’]
distData = sc.parallelize(data)
```

Sau khi được tạo, tập dữ liệu phân tán (distData) có thể được vận hành song song. Chả hạn:

```
tmp = distData.reduce(lambda a, b: a + b)
//tmp = ‘abcd’
```

Một tham số quan trọng cho các bộ sưu tập song song là số lượng *partitions (phân vùng)* để cắt tập dữ liệu. Spark sẽ chạy một tác vụ cho mỗi phân vùng của cụm. Thông thường, Spark sẽ phân vùng tự động. Một cách thủ công để phân vùng *sc.parallelize(data,10)*

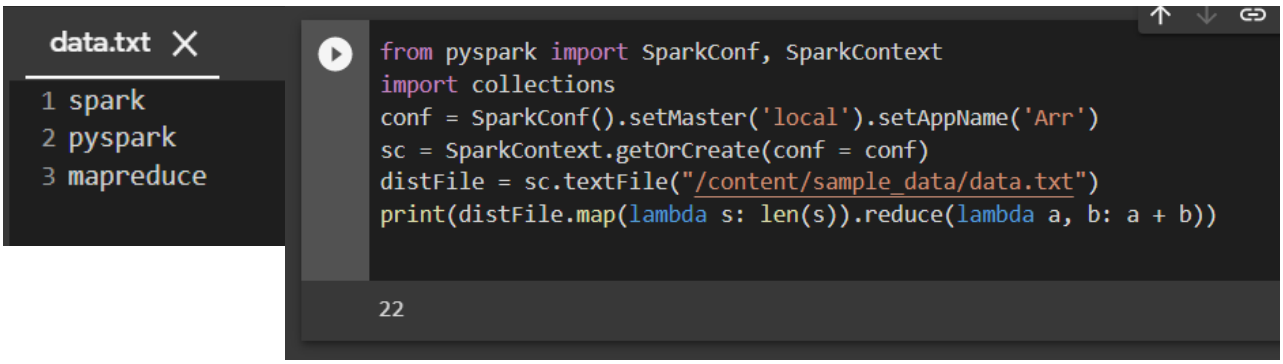
2.2.2. External Datasets (Bộ dữ liệu từ bên ngoài):

PySpark có thể tạo tập 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 , v.v. Spark hỗ trợ tệp văn bản, SequenceFiles và bất kỳ Hadoop InputFormat nào khác .

Sử dụng phương thức textFile của SparkContext. Một lời gọi ví dụ, lệnh gọi sẽ đọc file text data.txt :

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

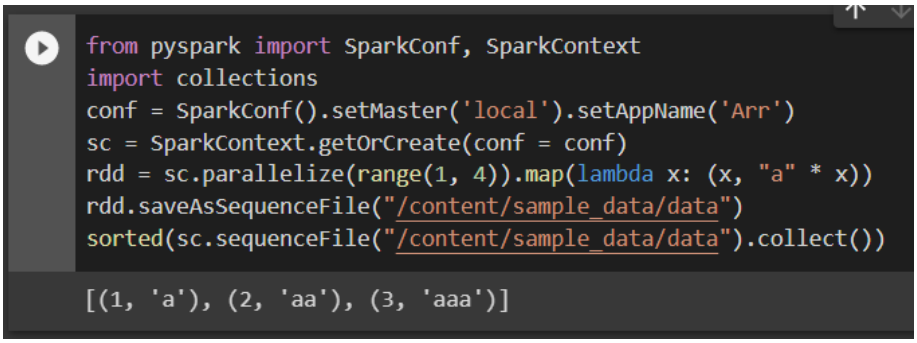
Sau khi được tạo, có thể thao tác lên distFile, chả hạn có thể in ra số lượng kí tự của File:

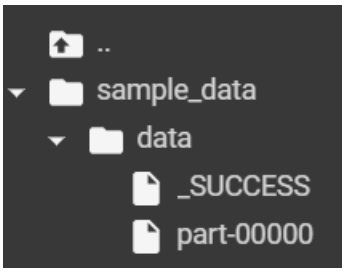



```
print(distFile.map(lambda s: len(s)).reduce(lambda a, b: a + b))
```

2.2.3. Lưu và tải SequenceFiles:

Tương tự như các tệp văn bản, SequenceFiles có thể được lưu và tải bằng cách chỉ định đường dẫn.





2.2.4. Hoạt động với RDD:

2.2.4.1. Transformations:

map(*func*)	RDD mới được tạo thành bằng cách áp dụng func lên tất cả các bản ghi trên RDD ban đầu.
filter(*func*)	RDD mới được tạo thành bằng cách áp dụng func lên tất cả các bản ghi trên RDD ban đầu và chỉ lấy những bản ghi mà func trả về true
flatMap(*func*)	Tương tự như map(), nhưng mỗi mục đầu vào có thể

	được ánh xạ tới 0 hoặc nhiều mục đầu ra (vì vậy func nên trả về một Seq thay vì một mục duy nhất).
mapPartitions(*func*)	Tương tự như map(), nhưng chạy riêng biệt trên từng phân vùng của RDD, vì vậy func phải là kiểu Iterator <T> => Iterator <U> khi chạy trên RDD kiểu T.
mapPartitionsWithIndex(*func*)	Tương tự như mapPartitions(), nhưng cũng cung cấp func với một giá trị nguyên đại diện cho chỉ mục của phân vùng, vì vậy func phải có kiểu (Int, Iterator <T>) => Iterator <U> khi chạy trên RDD kiểu T.
sample(*withReplacement, fraction, seed*)	Lấy mẫu một phần nhỏ của dữ liệu, có thể sẽ thay thế, bằng cách sử dụng một hạt tạo số ngẫu nhiên nhất định.
union(*otherDataset*)	Trả về một tập dữ liệu mới có chứa sự kết hợp của các phần tử trong tập dữ liệu nguồn và đối số.
intersection(*otherDataset*)	Trả về một RDD mới có chứa giao điểm của các phần tử trong tập dữ liệu nguồn và đối số.
distinct*([numPartitions]*))	Trả về tập dữ liệu mới có chứa các phần tử riêng biệt của tập dữ liệu nguồn.
groupByKey*([numPartitions]*)	Khi được gọi trên một tập dữ liệu gồm các cặp (K, V), trả về một tập dữ liệu gồm các cặp (K, Iterable <V>). Lưu ý: Nếu bạn đang nhóm để thực hiện tổng hợp (chẳng hạn như tổng hoặc trung bình) trên mỗi khóa, việc sử dụng *reduceByKey* hoặc *aggregateByKey* sẽ mang lại hiệu suất tốt hơn nhiều. Lưu ý: Theo mặc định, mức độ song song trong đầu ra phụ thuộc vào số lượng phân vùng của RDD mẹ. Bạn có thể chuyển một *numPartitions* đối số tùy chọn để đặt một số tác vụ khác nhau.
reduceByKey(*func, [numPartitions]*)	Khi được gọi trên tập dữ liệu của các cặp (K, V), trả về tập dữ liệu của các cặp (K, V) trong đó các giá trị cho mỗi khóa được tổng hợp bằng cách sử dụng hàm giảm hàm func đã cho , phải thuộc loại (V, V) => V. Giống như trong groupByKey, số lượng tác vụ giảm có thể được cấu hình thông qua đối số thứ hai tùy chọn.
aggregateByKey(*zeroValue*)(*seqOp, combOp, [numPartitions])*	Khi được gọi trên tập dữ liệu của các cặp (K, V), trả về tập dữ liệu của các cặp (K, U) trong đó các giá trị cho mỗi khóa được tổng hợp bằng cách sử dụng các hàm kết hợp đã cho và giá trị "0" trung tính. Cho phép loại giá trị tổng hợp khác với loại giá trị đầu vào, đồng thời tránh phân bổ không cần thiết. Giống như trong groupByKey, số lượng tác vụ giảm có thể được định cấu hình thông qua đối số thứ hai tùy chọn.
sortByKey*([ascending], [numPartitions])*	Khi được gọi trên tập dữ liệu của các cặp (K, V) trong đó K triển khai Có thứ tự, trả về tập dữ liệu của các cặp (K, V) được sắp xếp theo các khóa theo thứ tự tăng dần

	hoặc giảm dần, như được chỉ định trong ascendingđối số boolean .
join(*otherDataset, [numPartitions]*)	Khi được gọi trên các tập dữ liệu kiểu (K, V) và (K, W), trả về một tập dữ liệu gồm các cặp (K, (V, W)) với tất cả các cặp phần tử cho mỗi khóa. Ngoài tham gia được hỗ trợ thông qua leftOuterJoin, rightOuterJoin và fullOuterJoin.
cogroup(*otherDataset, [numPartitions]*)	Khi được gọi trên các tập dữ liệu kiểu (K, V) và (K, W), trả về một tập dữ liệu gồm các bộ dữ liệu (K, (Iterable <V>, Iterable <W>)). Thao tác này cũng được gọi là groupWith.
cartesian(*otherDataset*)	Khi được gọi trên các tập dữ liệu kiểu T và U, trả về một tập dữ liệu gồm (T, U) các cặp (tất cả các cặp phần tử).
pipe(*command, [envVars]*)	Đưa từng phân vùng của RDD thông qua một lệnh shell, ví dụ như tập lệnh Perl hoặc bash. Các phần tử RDD được ghi vào stdin của quá trình và các dòng xuất ra stdout của nó được trả về dưới dạng RDD của các chuỗi.
coalesce(*numPartitions*)	Giảm số lượng phân vùng trong RDD xuống numPartitions. Hữu ích để chạy các hoạt động hiệu quả hơn sau khi lọc bớt một tập dữ liệu lớn.
repartition(*numPartitions*)	Sắp xếp lại dữ liệu trong RDD một cách ngẫu nhiên để tạo nhiều hoặc ít phân vùng hơn và cân bằng giữa chúng. Điều này luôn luôn xáo trộn tất cả dữ liệu trên mạng.
repartitionAndSortWithinPartitions(*partitioner*)	Phân vùng lại RDD theo trình phân vùng đã cho và trong mỗi phân vùng kết quả, sắp xếp các bản ghi theo khóa của chúng. Điều này hiệu quả hơn việc gọi repartition và sau đó sắp xếp trong mỗi phân vùng vì nó có thể đẩy việc phân loại xuống bộ máy xáo trộn.

Bảng 5. Phép biến đổi của hoạt động RDD

2.2.4.2. Actions:

reduce(*func*)	Tập hợp các phần tử của tập dữ liệu bằng cách sử dụng một hàm func (nhận hai đối số và trả về một). Hàm phải có tính chất giao hoán và liên kết để nó có thể được tính toán song song một cách chính xác.
collect()	lấy tất cả RDD về driver
count()	đến số bản ghi của RDD
first()	trả về phần tử đầu tiên của RDD
take(*n*)	lấy n bản ghi từ RDD về driver
takeSample(*withReplacement, num, [seed]*)	Trả về một mảng có mẫu ngẫu nhiên gồm num phần tử của tập dữ liệu, có thể sẽ thay thế, tùy chọn chỉ định

	trước hạt tạo số ngẫu nhiên.
takeOrdered(*n, [ordering]*)	Trả về n phần tử đầu tiên của RDD bằng cách sử dụng thứ tự tự nhiên của chúng hoặc bộ so sánh tùy chỉnh.
saveAsTextFile(*path*)	ghi dữ liệu RDD ra file
countByKey()	Chỉ có sẵn trên RDD của loại (K, V). Trả về một bản đồ băm của các cặp (K, Int) với số lượng của mỗi khóa.
foreach(*func*)	Chạy một hàm func trên mỗi phần tử của tập dữ liệu. Điều này thường được thực hiện đối với các tác dụng phụ như cập nhật Accumulator hoặc tương tác với hệ thống lưu trữ bên ngoài. Lưu ý : việc sửa đổi các biến không phải Bộ tích lũy bên ngoài foreach()có thể dẫn đến hành vi không xác định.

Bảng 6. một số Actions của hoạt động RDD

Ví dụ về hoạt động của RDD: khởi tạo RDD từ một chuỗi (string), tìm mật độ của các từ.

```
from pyspark  import SparkConf, SparkContext
import collections

conf = SparkConf().setMaster('local').setAppName('Word counting')
sc = SparkContext.getOrCreate(conf = conf)
text = "to be or not to be".split()
rdd = sc.parallelize(text)
counts = rdd.map(lambda x:(x,1))
print(counts.collect())

red = counts.reduceByKey(lambda x,y: x+y)
print(red.collect())
```

Kết quả:

```
[('to', 1), ('be', 1), ('or', 1), ('not', 1), ('to', 1), ('be', 1)]
[('to', 2), ('be', 2), ('or', 1), ('not', 1)]
```

2.2.5. RDD Persistence:

Các mức lưu trữ của RDD:

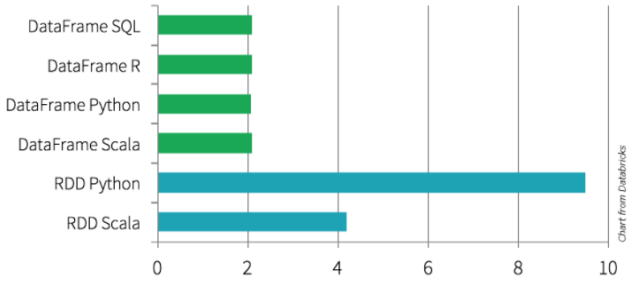
Storage Level	Meaning
MEMORY_ONLY	Lưu trữ RDD dưới dạng các đối tượng Java được giải hóa trong JVM. Nếu RDD không vừa với bộ nhớ, một số phân vùng sẽ không được lưu vào bộ nhớ đệm và sẽ được tính toán lại nhanh chóng mỗi khi chúng cần. Đây là mức mặc định.
MEMORY_AND_DISK	Lưu trữ RDD dưới dạng các đối tượng Java được giải hóa trong JVM. Nếu RDD không vừa trong bộ nhớ, hãy

	lưu trữ các phân vùng không vừa trên đĩa và đọc chúng từ đó khi cần.
DISK_ONLY	Chỉ lưu trữ các phân vùng RDD trên đĩa.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Tương tự như các cấp trên, nhưng sao chép từng phân vùng trên hai nút cụm.
OFF_HEAP (experimental)	Tương tự như MEMORY_ONLY_SER, nhưng lưu trữ dữ liệu trong bộ nhớ off-heap . Điều này yêu cầu bật bộ nhớ off-heap.

Bảng 7. Các mức lưu trữ của RDD

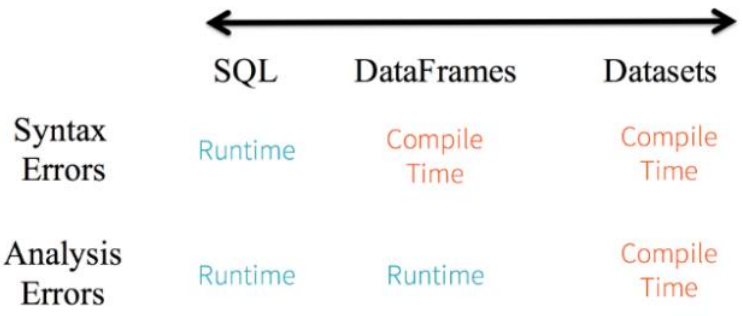
2.3. Spark DataFrame:

DataFrame là một API bậc cao hơn RDD được Spark giới thiệu vào năm 2013 (từ Apache Spark 1.3). Tương tự như RDD, dữ liệu trong DataFrame cũng được quản lý theo kiểu phân tán và không thể thay đổi (immutable distributed). Tuy nhiên dữ liệu này được sắp sếp theo các cột, tương tự như trong Relation Database. DataFrame được phát triển để giúp người dùng có thể dễ dàng thực hiện các thao tác xử lý dữ liệu cũng như làm tăng đáng kể hiệu quả sử lý của hệ thống.[3]

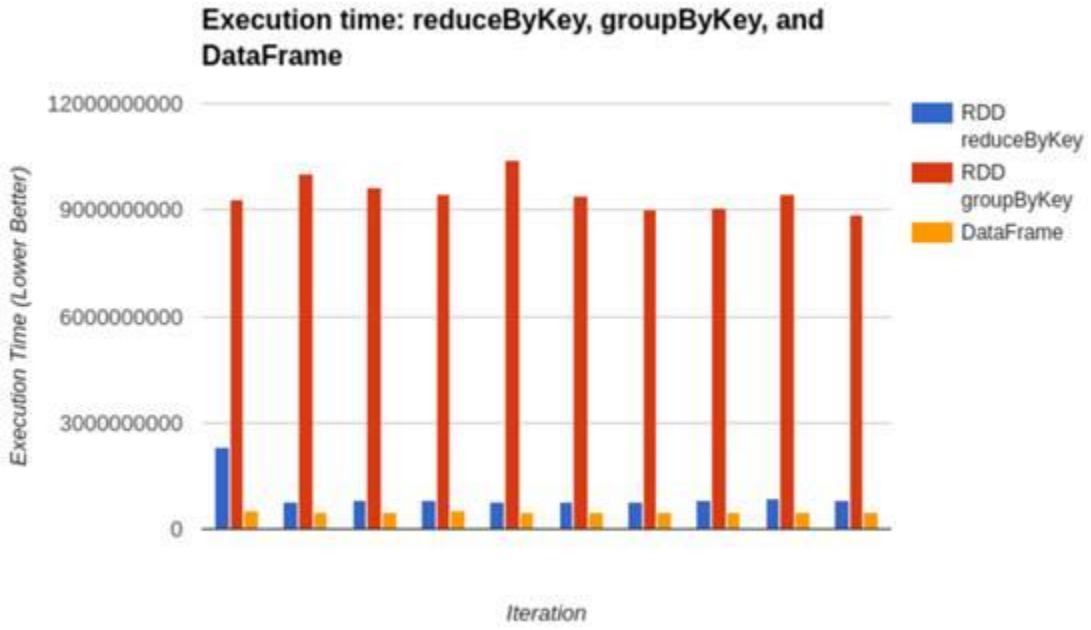


Thời gian để tổng hợp 10 cặp interger (tính bằng giây)

Khi sử dụng DataFrame API, chúng ta gọi các hàm để trích xuất kết quả mong muốn và Spark sẽ tự động tiến hành các thuật toán xử lý. Tuy nhiên ở bước cuối cùng thì các thuật toán này vẫn được chạy trên RDD mặc dù người dùng chỉ tương tác với DataFrame. Bên cạnh các ưu điểm, thì nhược điểm lớn nhất của DataFrame là API này không hỗ trợ Compile-time type safety, do đó chúng ta khó có thể tiến hành thao tác trên dữ liệu. Ví dụ như khi chúng ta dùng DataFrame để truy xuất people("age"), kết quả trả về không phải ở dạng Int mà ở dạng Column object. Vì vậy chúng ta không thể thực hiện các thao tác với kết quả này như đối với một Int object. Việc không hỗ trợ type safefy này làm người dùng không thể phát huy lợi thế của type system mà các ngôn ngữ lập trình như Scala, Java,.. hỗ trợ. Ngoài ra, nó còn làm tăng các lỗi runtime mà đáng ra đã được phát hiện tại compile time. [3]



Một ví dụ về execution time, thực thi các aggregation thông qua RDD với 2 api là reduceByKey và RDD groupByKey với DataFrame và kết quả là DF có thời gian xử lý nhanh hơn cả.



Hiểu đơn giản DF giống như 1 bảng trong hệ CSDL quan hệ. Có các trường, được định sẵn kiểu dữ liệu, và tập hợp các bản ghi. Tuy nhiên 1 DF có thể đại diện cho lượng dữ liệu lớn hơn rất nhiều so với các bảng trong DB, các DF lên tới hàng chục tỉ row đều có thể xử lý.

Các đặc điểm bao gồm:

- immutable: tính bất biến, dữ liệu của 1 DF sau khi tạo ra sẽ không thay đổi, nếu muốn chỉnh sửa ta cần tạo ra DF mới từ DF ban đầu, thông qua DF api.

- rows: là đối tượng đại diện cho 1 bản ghi dữ liệu. 1 DF = tập các row phân tán

- set of columns has name and an associated type: Ý nói về việc dữ liệu của DF là có cấu trúc, gồm tên là kiểu dữ liệu.

2.3.1. SparkSession:[4]

Bắt đầu của DataFrame, xây dựng SparkSession:

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

2.3.2. Tạo DataFrames:

Với việc xây dựng SparkSession, các ứng dụng có thể tạo DataFrames từ một RDD hiện có, từ một bảng Hive, hoặc từ Spark data sources.

Ví dụ, tạo DataFrame dựa vào nội dung của một file json:

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
df = spark.read.json("/content/sample_data/anscombe.json")
df.show()
```

```
+-----+-------+----+---------------+
|   ID|   Name|Team|_corrupt_record|
+-----+-------+----+---------------+
| null|   null|null|             [|
|51800|  Herry|   I|           null|
|51801|   Rose|   I|           null|
|51802| Yellow|   I|           null|
|51803| Redbin|   I|           null|
|51804|  Lotus|   I|           null|
|51805|  Daisy|   I|           null|
|51806| Steven|   I|           null|
|51807|    Lin|   I|           null|
|51808|     IU|   I|           null|
|51809|   Kint|   I|           null|
|51810|  Kevin|   I|           null|
|51811|  Harry|  II|           null|
|51812|Lucifer|  II|           null|
|51813|   Alex|  II|           null|
|51814|  Robin|  II|           null|
|51815|  Daiti|  II|           null|
|51816|   Elin|  II|           null|
|51817|   Elsa|  II|           null|
|51818|    Inl|  II|           null|
+-----+-------+----+---------------+
only showing top 20 rows
```

anscombe.json

```
[
  {"Team":"I", "Name":"Herry", "ID":51800},
  {"Team":"I", "Name":"Rose", "ID":51801},
  {"Team":"I", "Name":"Yellow", "ID":51802},
  {"Team":"I", "Name":"Redbin", "ID":51803},
  {"Team":"I", "Name":"Lotus", "ID":51804},
  {"Team":"I", "Name":"Daisy", "ID":51805},
  {"Team":"I", "Name":"Steven", "ID":51806},
  {"Team":"I", "Name":"Lin", "ID":51807},
  {"Team":"I", "Name":"IU", "ID":51808},
  {"Team":"I", "Name":"Kint", "ID":51809},
  {"Team":"I", "Name":"Kevin", "ID":51810},

  {"Team":"II", "Name":"Harry", "ID":51811},
  {"Team":"II", "Name":"Lucifer", "ID":51812},
  {"Team":"II", "Name":"Alex", "ID":51813},
  {"Team":"II", "Name":"Robin", "ID":51814},
  {"Team":"II", "Name":"Daiti", "ID":51815},
  {"Team":"II", "Name":"Elin", "ID":51816},
  {"Team":"II", "Name":"Elsa", "ID":51817},
  {"Team":"II", "Name":"Inl", "ID":51818},
  {"Team":"II", "Name":"Ruo", "ID":51819},
  {"Team":"II", "Name":"Hou", "ID":51820},
  {"Team":"II", "Name":"Hugo", "ID":51821},
]
```

2.3.3. Hoạt động với DataFrames:

- printSchema() : In lược đồ ở định dạng cây.

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
df = spark.read.json("/content/sample_data/anscombe.json")
df.printSchema()
```

```
root
 |-- ID: long (nullable = true)
 |-- Name: string (nullable = true)
 |-- Team: string (nullable = true)
 |-- _corrupt_record: string (nullable = true)
```

- Select("field") hoặc Select(df ['filed']) : Khi chỉ đưa ra dữ liệu trường cần thiết.

```
df = spark.read.json("/content/sample_data/anscombe.json")
df.select("Name").show()
```

```
+-------+
|   Name|
+-------+
|   null|
|  Herry|
|   Rose|
| Yellow|
| Redbin|
|  Lotus|
```

Chọn và thao tác dữ liệu trong trường, ví dụ: đưa ra cột Name và cột ID trong đó dữ liệu cột ID cộng thêm 1:

```
df = spark.read.json("/content/sample_data/anscombe.json")
df.select(df['Name'], df['ID'] + 1).show()
```

```
+-------+--------+
|   Name|(ID + 1)|
+-------+--------+
|   null|    null|
|  Herry|   51801|
|   Rose|   51802|
| Yellow|   51803|
| Redbin|   51804|
|  Lotus|   51805|
|  Daisy|   51806|
| Steven|   51807|
```

- Filter(): Lựa chọn những người có ID từ 51807 trở về trước.

```
df = spark.read.json("/content/sample_data/anscombe.json")
df.filter(df['ID'] < 51807).show()
```

```
+-----+------+----+---------------+
|   ID|  Name|Team|_corrupt_record|
+-----+------+----+---------------+
|51800| Herry|   I|           null|
|51801|  Rose|   I|           null|
|51802|Yellow|   I|           null|
|51803|Redbin|   I|           null|
|51804| Lotus|   I|           null|
|51805| Daisy|   I|           null|
|51806|Steven|   I|           null|
+-----+------+----+---------------+
```

Khi sử dụng select với cùng điều kiện, kết quả sẽ trả về true/ false tương ứng với những dữ liệu thoải/ không thỏa yêu cầu.

```
df.select(df['ID'] < 51807).show()
```

```
+------------+
|(ID < 51807)|
+------------+
|        null|
|        true|
|        true|
|        true|
|        true|
|        true|
|        true|
|        true|
|       false|
|       false|
|       false|
|       false|
```

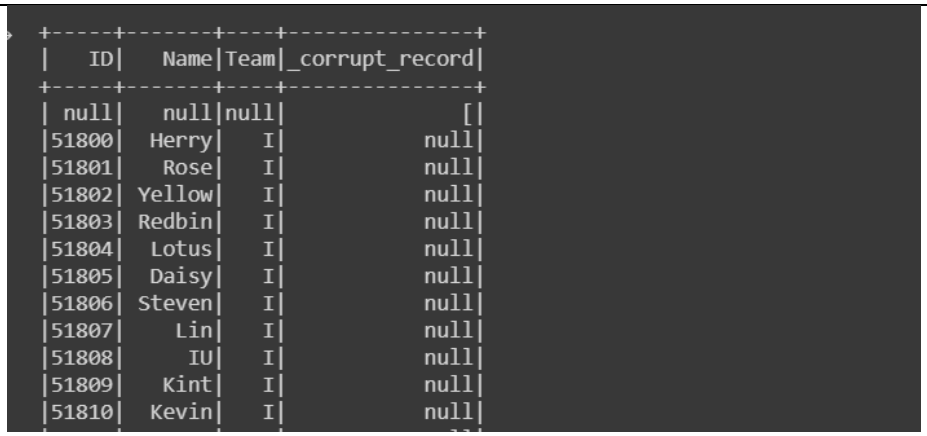
- groupBy() kết hợp count(): Muốn đếm số người có cùng Team.

```
df.groupBy('Team').count().show()
```

```
+----+-----+
|Team|count|
+----+-----+
|null|    2|
|  II|   11|
|   I|   11|
+----+-----+
```

2.3.4. Chạy câu truy vấn SQL với DataFrame:

```
df.createReplaceTempView("People")
sqlDF = spark.sql("SELECT * FROM People")
sqlDF.show()
```



ID	Name	Team	_corrupt_record
null	null	null	[
51800	Herry	I	null
51801	Rose	I	null
51802	Yellow	I	null
51803	Redbin	I	null
51804	Lotus	I	null
51805	Daisy	I	null
51806	Steven	I	null
51807	Lin	I	null
51808	IU	I	null
51809	Kint	I	null
51810	Kevin	I	null

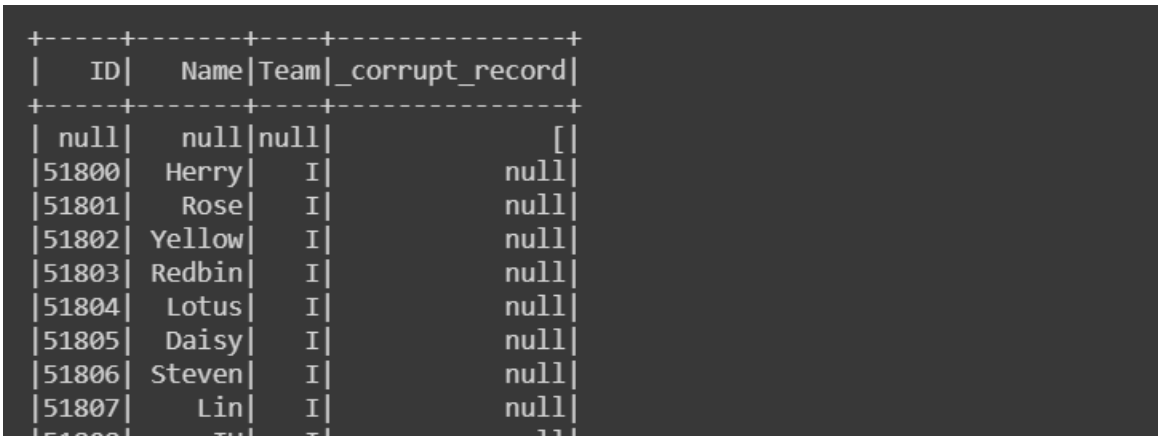
Đăng kí DataFrame dưới dạng xem tạm thời của một SQL bằng phương thức createOrReplaceTempView("*name*"). Sau đó gọi câu truy vấn qua phương thức sql("*câu truy vấn*").

2.3.5. Global Temporary View:

Các dạng xem tạm thời trong Spark SQL là phạm vi phiên và sẽ biến mất nếu phiên tạo ra nó kết thúc. Nếu bạn muốn có một chế độ xem tạm thời được chia sẻ giữa tất cả các phiên và duy trì hoạt động cho đến khi ứng dụng Spark kết thúc, bạn có thể tạo một chế độ xem tạm thời chung. Global Temporary View được gắn với một cơ sở dữ liệu hệ thống bảo quản global_temp, và chúng ta phải sử dụng tên đủ điều kiện để tham khảo nó.

Ví dụ:

```
df.createGlobalTempView(“People”)
sqlDF = spark.sql("SELECT * FROM global_temp.People")
sqlDF_N = spark.newSession().sql("SELECT * FROM global_temp.People")
sqlDF.show()
sqlDF_N.show()
```



```
+-----+-------+----+---------------+
|   ID|   Name|Team|_corrupt_record|
+-----+-------+----+---------------+
| null|   null|null|             [|
|51800|  Herry|   I|           null|
|51801|   Rose|   I|           null|
|51802| Yellow|   I|           null|
|51803| Redbin|   I|           null|
|51804|  Lotus|   I|           null|
|51805|  Daisy|   I|           null|
|51806| Steven|   I|           null|
|51807|    Lin|   I|           null|
```

Global temporary view is cross-sessio.

Tài liệu tham khảo:

[1] <URL: https://spark.apache.org/docs/2.3.0/configuration.html#spark-properties> , trang chủ của Apache Spark.
[2] <URL: https://spark.apache.org/docs/latest/rdd-programming-guide.html>, trang chủ của Apache Spark.
[3] <URL: http://itechseeker.com/en/tutorials-2/apache-spark-3/apache-spark-with-scala/spark-sql-dataset-and-dataframes/>, November 19, 2018 ItechSeeker
[4] <URL: https://spark.apache.org/docs/2.3.0/sql-programming-guide.html>, trang chủ của Apache Spark.