TÌM HIỂU SPARK

1. Spark properties

Thuộc tính Spark (Spark properties) là phương tiện điều chỉnh môi trường thực thi của Spark, nó 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 được chuyển tới SparkContext của bạn. Một số thuộc tính phổ biến được SparkConf cấu hình thông qua phương thức set như master URL (URL chính), AppName (tên ứng dụng),... ví dụ:

```
val conf = new SparkConf()
                    .setMaster('local[2]')
                    .setAppName('Word Counting')
val sc = new SparkContext(conf)
```

Lưu ý, $local[2]$ ở đây nghĩa là ứng dụng được chạy với hai luồng đại diện cho sự song song tối thiểu, giúp phát hiện các lỗi chỉ tồn tại khi chạy trong ngữ cảnh phân tán.

Ngoài ra, các thuộc tính chỉ định một số khoảng thời gian nên được cấu hình với một đơn vị thời gian, các thuộc tính chỉ định kích thước byte nên được cấu hình với một đơn vị kích thước. Cụ thể, các định dạng dưới đây là hợp lệ:

```
//Với đơn vị thời gian:
10s (seconds)
10ms (milli seconds)
10m (minutes)
10h (hours)
10d (days)
10y (years)
//Với đơn vị kích thước:
10b (bytes)
10kb
10mb
10gb
10tb
10pb
```

Sau khi đã khởi tạo SparkConf và sử dụng các phương thức setter hỗ trợ chuỗi, chúng ta có thể sử dụng một số thuộc tính phổ biến của SparkConf:

- $set(key, value)$: để đặt thuộc tính cấu hình,
- $setMaster(value)$: để đặt master URL.
- $setAppName(value)$: để đặt tên ứng dụng.
- $get(key, defaultValue = None)$: để nhận giá trị cấu hình của một khóa.
- $setSparkHome(value)$: để thiết lập đường dẫn cài đặt Spark trên các worker nodes

Một số 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:

- Application 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. Tối thiểu phải là 1M hoặc 0 cho không giới hạn.
  - $spark.executor.pyspark.memory$: Lượng bộ nhớ được cấp phát cho PySpark trong mỗi trình thực thi
  - $spark.master$: Người quản lý cụm để kết nối
- Runtime Environment:
  - $spark.python.profile$: Cho phép cấu hình trong Python worker, kết quả cấu hình sẽ hiển thị bằng sc.show_profiles () hoặc nó sẽ được hiển thị trước khi trình điều khiển thoát.
  - $spark.python.profile.dump$: Thư mục được sử dụng để kết xuất kết quả hồ sơ trước khi trình điều khiển thoát.
  - $spark.submit.pyFiles$: Danh sách các tệp .zip, .egg hoặc .py được phân tách bằng dấu phẩy để đặt trên ứng dụng PYTHONPATH cho Python.
  - $spark.pyspark.python$: Thực thi nhị phân Python để sử dụng cho PySpark trong cả trình điều khiển và trình thực thi.
- Spark SQL:
  - $spark.sql.csv.filterPushdown.enabled$: Khi value là true, cho phép bật bộ lọc đẩy xuống nguồn dữ liệu csv.
  - $spark.sql.execution.arrow.pyspark.enabled$: dùng mũi tên Apache để truyền dữ liệu dạng cột trong PySpark.

2. Spark RDD

2.1 Khái niệm và khởi tạo RDD

2.1.1 Khái niệm

Resilient Distributed Datasets là một cấu trúc dữ liệu cơ bản của Spark. Nó là một tập hợp các đối tượng được phân phối bất biến. Mỗi tập dữ liệu trong RDD được chia thành các phân vùng logic, có thể được tính toán trên các nút khác nhau của cụm. RDD có thể chứa bất kỳ loại đối tượng Python, Java hoặc Scala nào, bao gồm các lớp do người dùng định nghĩa.

Về mặt hình thức, RDD là một tập hợp các bản ghi được phân vùng, chỉ đọc. RDD có thể được tạo thông qua các hoạt động xác định trên dữ liệu trên bộ lưu trữ ổn định hoặc các RDD khác. RDD là một tập hợp các phần tử chịu được lỗi có thể hoạt động song song.

2.1.2 Khởi tạo RDD

Có hai cách để tạo RDD - song song 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 chia sẻ, HDFS, HBase hoặc bất kỳ nguồn dữ liệu nào cung cấp Định dạng đầu vào Hadoop.

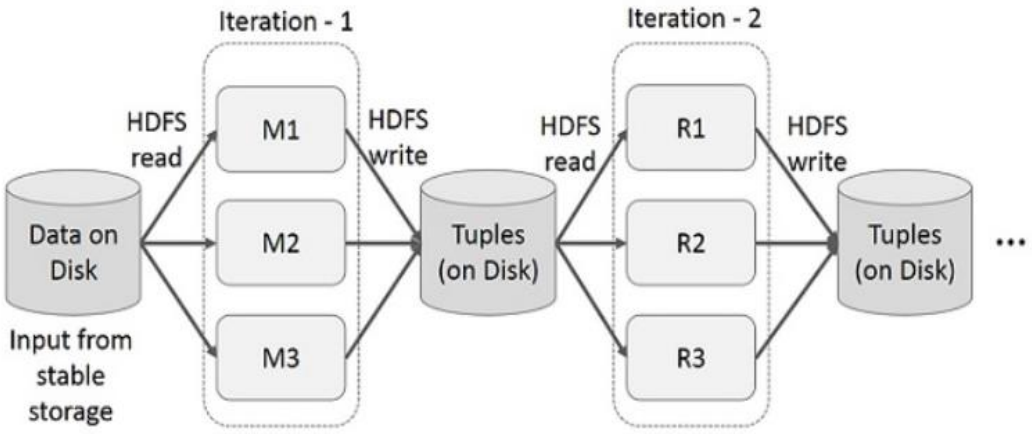
Spark sử dụng khái niệm RDD để đạt được các hoạt động MapReduce nhanh hơn và hiệu quả hơn. Trước tiên, chúng ta hãy thảo luận về cách các hoạt động MapReduce diễn ra và tại sao chúng không hiệu quả như vậy.

MapReduce được sử dụng rộng rãi để xử lý và tạo các bộ dữ liệu lớn với một thuật toán phân tán, song song trên một cụm. Nó cho phép người dùng viết các phép tính song song, sử dụng một tập hợp các toán tử cấp cao, mà không phải lo lắng về việc phân phối công việc và khả năng chịu lỗi.

2.2 Thực thi RDD

2.2.1 Thực thi với Iterative Operations trên MapReduce

Sử dụng lại các kết quả trung gian qua nhiều lần tính toán trong các ứng dụng nhiều giai đoạn. Hình minh họa sau giải thích cách hoạt động của khung hiện tại trong khi thực hiện các hoạt động lặp lại trên MapReduce. Điều này phát sinh chi phí đáng kể do sao chép dữ liệu, I / O đĩa và tuần tự hóa, khiến hệ thống chậm.

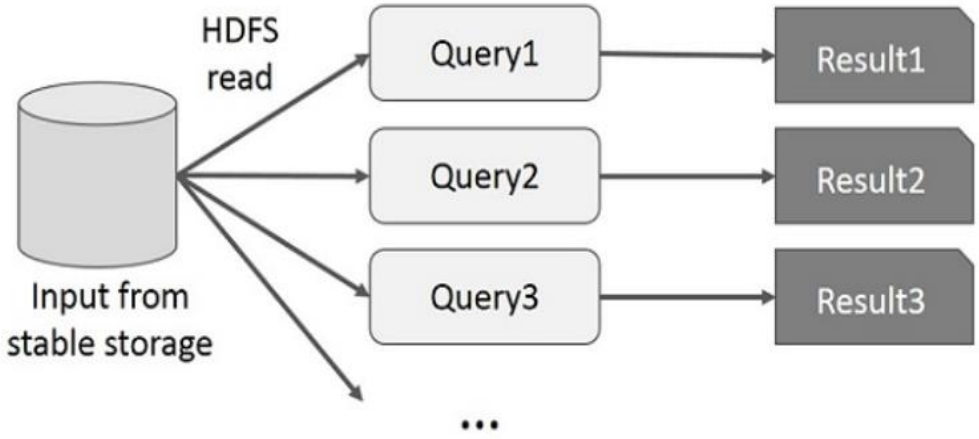


Hình 1. Iterative Operations trên MapReduce

2.2.2 Thực thi với Interatvie Operations trên MapReduce

Người dùng chạy các truy vấn đặc biệt trên cùng một tập con dữ liệu. Mỗi truy vấn sẽ thực hiện I / O đĩa trên bộ nhớ ổn định, có thể chi phối thời gian thực thi ứng dụng.

Hình minh họa sau giải thích cách hoạt động của khung hiện tại khi thực hiện các truy vấn tương tác trên MapReduce.

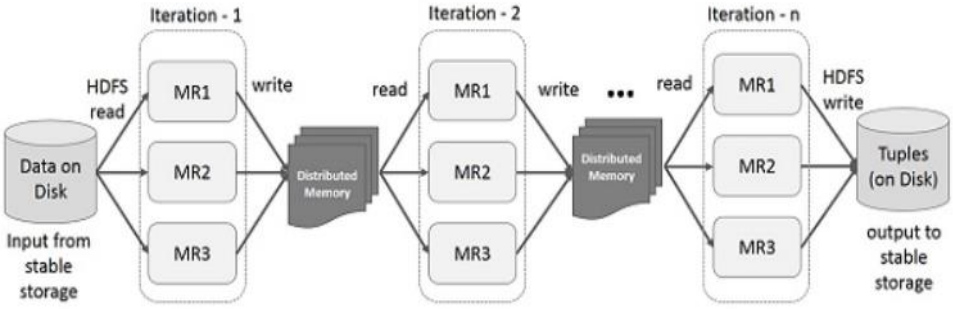


Hình 2. Interactive Operations trên MapReduce

2.2.3 Thực thi với Interatvie Operations trên Spark RDD

Hình minh họa dưới đây cho thấy các hoạt động lặp lại trên Spark RDD. Nó sẽ lưu trữ các kết quả trung gian trong một bộ nhớ phân tán thay vì Ổ lưu trữ ổn định (Đĩa) và làm cho hệ thống nhanh hơn.

Lưu ý: Nếu bộ nhớ phân tán (RAM) không đủ để lưu trữ các kết quả trung gian (Trạng thái công việc), thì nó sẽ lưu các kết quả đó trên đĩa.

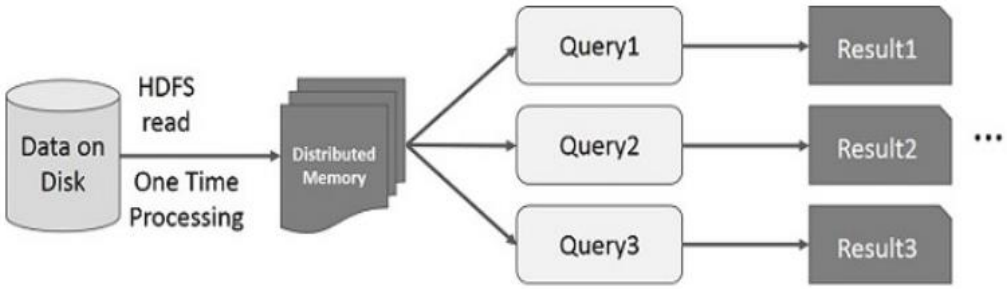


Hình 3. Iterative Operations trên Spark RDD

2.2.4 Thực thi với Interatvie Operations trên Spark RDD

Hình minh họa này cho thấy các hoạt động tương tác trên Spark RDD. Nếu các truy vấn khác nhau được chạy lặp lại trên cùng một tập dữ liệu, thì dữ liệu cụ thể này có thể được lưu trong bộ nhớ để có thời gian thực thi tốt hơn.

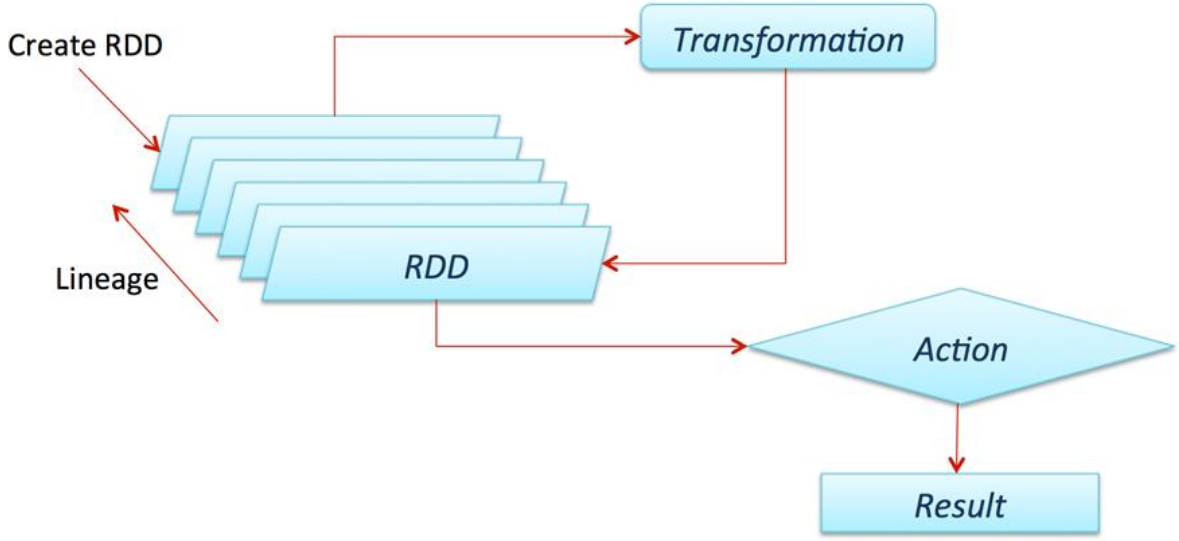
Theo mặc định, mỗi RDD đã chuyển đổi có thể được tính toán lại mỗi khi bạn chạy một hành động trên đó.



Hình 4. Interactive Operations trên Spark RDD

2.3 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.



Hình 5. Transformation và action với RDD

Một số transformation:

- $distinct$: 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.
- 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.

Một số action:

- $reduce$: thực hiện hàm reduce trên RDD để thu về 1 giá trị duy nhất.
- $count$: đếm số dòng trong RDD.

- $countByValue$: đếm số giá trị của RDD.
- $countApproxDistinct$: đếm xấp xỉ các giá trị khác nhau.
- max, min: lần lượt lấy giá trị lớn nhất và nhỏ nhất của dataset

2.4 Hạn chế của 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 chất xúc tác 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.

2.5 Code minh họa cho RDD

Đoạn code dưới đây minh họa cho việc đọc file text và đếm số lượng từ có trong file:

```
files = (sc.textFile('/content/text.txt').map(lambda
line: line.split(' ')))

fileContent = files.reduce(lambda x: 1)

rdd = (sc.parallelize(fileContent)).map(lambda word:
(word, 1))

print(rdd.count())

words = rdd.reduceByKey(lambda x,y: x+y)

print(words)
```

3. Spark DataFrame

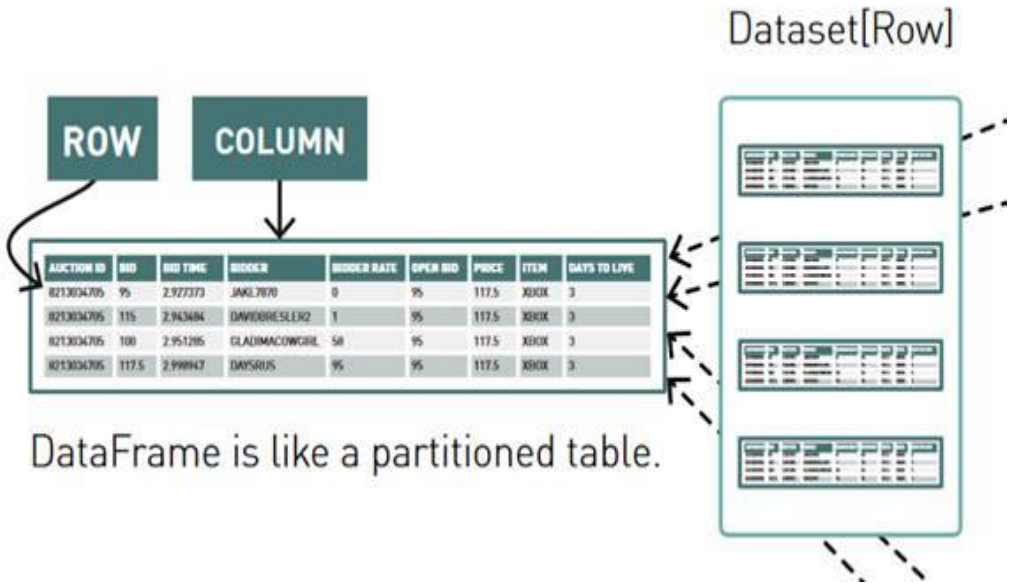
3.1 Khái niệm

Trong Apache Spark, 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 các bảng quan hệ có kỹ thuật tối ưu hóa tốt..

Một DataFrame có thể được xây dựng từ một loạt các nguồn khác nhau như bảng Hive, tệp Dữ liệu có cấu trúc, cơ sở dữ liệu bên ngoài hoặc RDD hiện có. API này được thiết kế cho các ứng dụng Khoa học dữ liệu và Dữ liệu lớn hiện đại lấy cảm hứng từ DataFrame trong Lập trình R và Pandas trong Python. Ngoài ra, nó cũng chia sẽ một số điểm chung với RDD:

- Bất biến về bản chất: Chúng tôi có thể tạo DataFrame / RDD một lần nhưng không thể thay đổi nó. Và chúng ta có thể chuyển đổi một DataFrame / RDD sau khi áp dụng các phép biến đổi.
- Đánh giá lười biếng: Có nghĩa là một nhiệm vụ không được thực hiện cho đến khi một hành động được thực hiện.

- Được phân phối: RDD và DataFrame đều được phân phối trong tự nhiên.



Hình 6. Một bảng ví dụ về DataFrame

3.2 Một số điểm nổi bật

Dưới đây là một số tính năng đặc trưng của DataFrame:

- Khả năng xử lý dữ liệu có kích thước từ Kilobyte đến Petabyte trên một cụm nút đơn đến cụm lớn.
- Hỗ trợ các định dạng dữ liệu khác nhau (Avro, csv, tìm kiếm đàn hồi và Cassandra) và hệ thống lưu trữ (HDFS, bảng HIVE, mysql, v.v.).
- Tối ưu hóa hiện đại và tạo mã thông qua trình tối ưu hóa Spark SQL Catalyst (khung chuyển đổi cây).
- Có thể dễ dàng tích hợp với tất cả các công cụ và khuôn khổ Dữ liệu lớn thông qua Spark-Core.
- Cung cấp API cho Lập trình Python, Java, Scala và R.

3.3 Khởi tạo DataFrame

Một DataFrame trong Apache Spark có thể được tạo theo nhiều cách với các định dạng dữ liệu khác nhau:

- Tải dữ liệu từ JSON, csv.
- Tải dữ liệu từ RDD hiện có.

3.3.1 Khởi tạo từ RDD

Để tạo DataFrame từ danh sách các tuples, ta làm các bước sau:

- Tạo một danh sách các bộ giá trị. Mỗi bộ chứa tên của một người có tuổi.
- Tạo một RDD từ danh sách trên.

- Chuyển đổi từng bộ thành một hàng.
- Tạo một DataFrame bằng cách áp dụng createDataFrame trên RDD với sự trợ giúp của sqlContext.

```
from pyspark.sql import Row
l = [('Quang',20),('Han',21),('Toan',21),('Tien',20)]
rdd = sc.parallelize(l)
people = rdd.map(lambda x: Row(name=x[0], age=int(x[1])))
schemaPeople = sqlContext.createDataFrame(people)
```

3.3.2 Khởi tạo từ file csv

Để đọc tệp csv trong Apache Spark, chúng ta cần chỉ định một thư viện mới trong trình bao python của mình. Để thực hiện thao tác này, đầu tiên chúng ta cần tải gói Spark-csv (Phiên bản mới nhất) và giải nén gói này vào thư mục chính của Spark. Sau đó, chúng ta cần mở một trình bao PySpark và bao gồm gói (tôi đang sử dụng "spark-csv_2.10: 1.3.0").

```
$ ./bin/pyspark --packages com.databricks:spark-csv_2.10:1.3.0
```

Sau đó tiến hành đọc dữ liệu từ tệp csv và tạo DataFrame:

```
train = sqlContext.load(source="com.databricks.spark.csv", path
= 'PATH/train.csv', header = True,inferSchema = True)
test = sqlContext.load(source="com.databricks.spark.csv", path
= 'PATH/test-comb.csv', header = True,inferSchema = True)
```

Trong đó:

- PATH là vị trí của thư mục, nơi chứa các tệp csv đào tạo và kiểm tra của bạn.
- Header là True, có nghĩa là các tệp csv chứa tiêu đề. Chúng tôi đang sử dụng tùy chọn invSchema = True để yêu cầu sqlContext tự động phát hiện kiểu dữ liệu của mỗi cột trong khung dữ liệu. Nếu chúng ta không đặt invitro là true, tất cả các cột sẽ được đọc dưới dạng chuỗi.

3.3 Thao tác với DataFrame

Để xem các loại cột trong DataFrame, chúng ta có thể sử dụng printSchema, dtypes. Hãy áp dụng printSchema () trên tàu sẽ in lược đồ ở định dạng cây. Ví dụ: `train.printSchema()`.

Chúng ta có thể sử dụng hoạt động đầu để xem n lần quan sát đầu tiên (ví dụ, 5 lần quan sát). Hoạt động đầu trong PySpark tương tự như hoạt động đầu trong Pandas. Ví dụ: `train.head(5).`

Chúng ta có thể sử dụng thao tác đếm để đếm số hàng trong DataFrame. Hãy áp dụng thao tác đếm trên tệp tàu và tệp thử nghiệm để đếm số hàng. Ví dụ: `train.count(),test.count().`

Để lấy ra tập con từ các cột, chúng ta cần sử dụng thao tác chọn trên DataFrame và chúng ta cần chuyển các tên cột được phân tách bằng dấu phẩy bên trong Thao tác chọn. Ví dụ: chọn 5 hàng đầu tiên của 'User_ID' và 'Age' từ tàu: `train.select('User_ID', 'Age').show(5).`

Bên trên là một số thao tác cơ bản với DataFrame.

3.4 Hạn chế của DataFrame

Một nhược điểm của DF là do kiểu dữ liệu được fix là row và truy cập dữ liệu trong DF thông qua row name nên nếu có sai sót trong việc truyền tên cột, trình biên dịch sẽ không thể phát hiện ra lỗi mà khi thực thi mới có exeption (Runtime exception).

TÀI LIỆU THAM KHẢO

[1] 2017, Complete Guide on DataFrame Operations in PySpark <https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/>

[2] Spark SQL – DataFrames <https://www.tutorialspoint.com/spark_sql/spark_sql_dataframes.htm?>

[3] 2019, Apache Spark RDD <https://laptrinh.vn/books/apache-spark/page/apache-spark-rdd>

[4] Spark Configurtaion <https://spark.apache.org/docs/latest/configuration.html>