



Samsung Innovation Campus

| Khoa học Big Data

Together for Tomorrow!
Enabling People

Education for Future Generations

Chương 6

Xử lý Big Data với Apache Spark

Khoa học Big Data

Mô tả chương

Mục tiêu:

- ✓ Apache Spark là một khung xử lý dữ liệu có mục đích chung kết hợp các mô-đun khác nhau để xử lý các loại dữ liệu khác nhau. Đây hiện là một trong những dự án nguồn mở tích cực nhất với nhiều doanh nghiệp áp dụng Spark cho các trường hợp sử dụng của họ.
 - Chúng ta sẽ tìm hiểu cách sử dụng lớp Spark Core API để xử lý, làm sạch và truy vấn dữ liệu phi cấu trúc và dữ liệu bán cấu trúc. Chúng ta cũng sẽ tìm hiểu cách chuyển đổi dữ liệu đó thành dữ liệu có cấu trúc.
 - API Dataframe của Spark là bản dựng API cấp cao hơn dựa trên API lõi và được tối ưu hóa để xử lý dữ liệu có cấu trúc. Chúng ta sẽ khám phá việc xử lý dữ liệu bằng cách sử dụng chức năng API cũng như thông qua việc chuyển các truy vấn cú pháp SQL
 - API truyền trực tuyến có cấu trúc và API DStream được sử dụng để xử lý dữ liệu truyền trực tuyến có cấu trúc và dữ liệu truyền trực tuyến không có cấu trúc tương ứng. Chúng ta sẽ tìm hiểu cách sử dụng API, nhấn mạnh vào việc kết nối với các chủ đề Kafka cho nguồn dữ liệu phát trực tuyến
 - Cuối cùng, chúng ta sẽ khám phá kiến trúc của Spark và cách nó ảnh hưởng đến việc điều chỉnh hiệu suất

Nội dung:

1. Xử lý dữ liệu phi cấu trúc
2. Xử lý dữ liệu có cấu trúc
3. Xử lý dữ liệu trực tuyến
4. Điều chỉnh hiệu suất và kiến trúc Apache Spark

Bài 1

Xử lý dữ liệu phi cấu trúc

Xử lý Big Data với Apache Spark

Bài 1.

Xử lý dữ liệu phi cấu trúc

| 1.1. Giới thiệu về Apache Spark

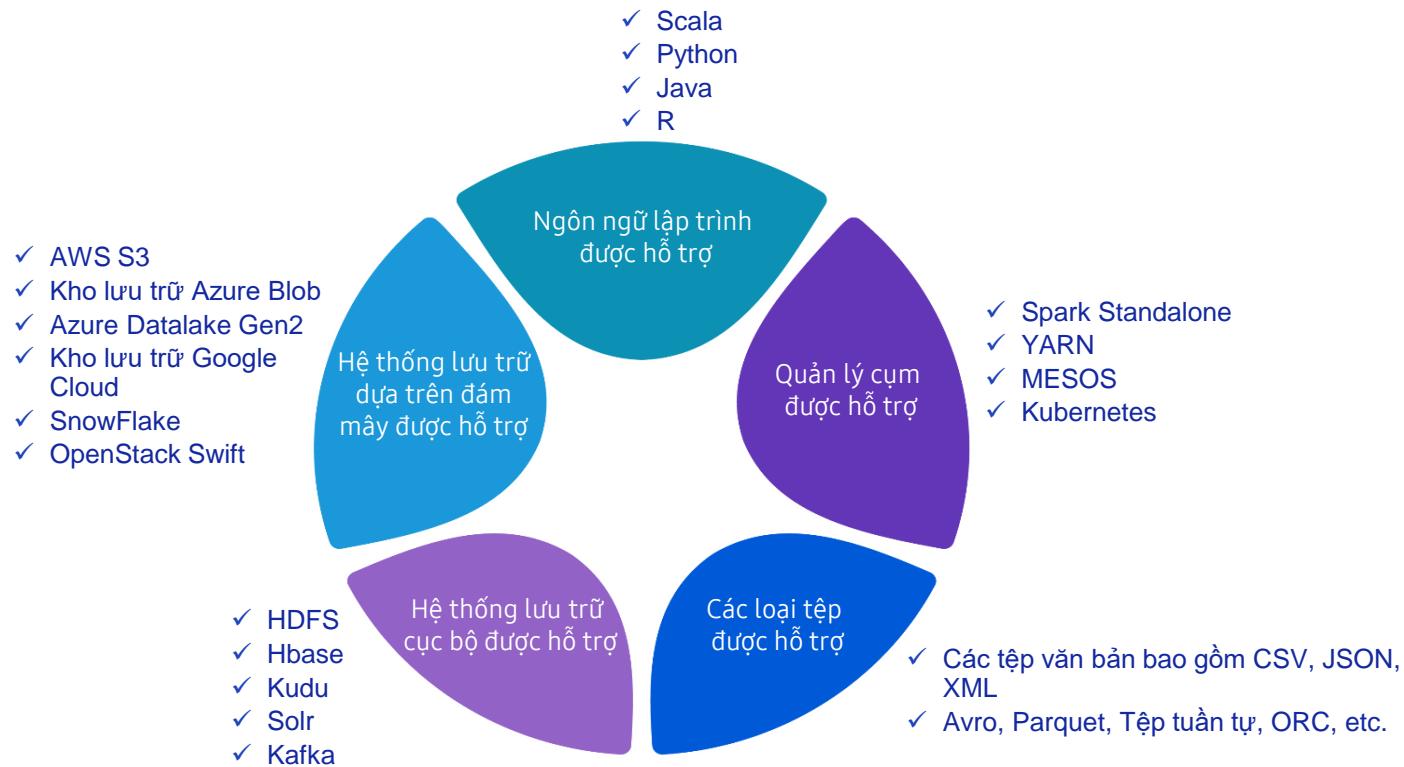
- | 1.2. Khái niệm cơ bản về Python
- | 1.3. Chuyển đổi dữ liệu với Core API
- | 1.4. Làm việc với Pair RDD

Apache Spark là gì?

- Apache Spark là một công cụ phân tích thống nhất để xử lý dữ liệu quy mô lớn
 - ▶ Một bộ API cấp cao trong Scala, Python, Java và R
- Công cụ xử lý dữ liệu đa năng cung cấp
 - ▶ Xử lý tương tác
 - ▶ Xử lý hàng loạt cực nhanh
 - ▶ Xử lý luồng thời gian thực
 - ▶ Mô hình học máy phân tán và công cụ suy luận
 - ▶ Xử lý đồ thị
- Công cụ tính toán phân tán trong bộ nhớ
- Ban đầu được phát triển bởi Phòng thí nghiệm AMP tại Đại học California, Berkeley

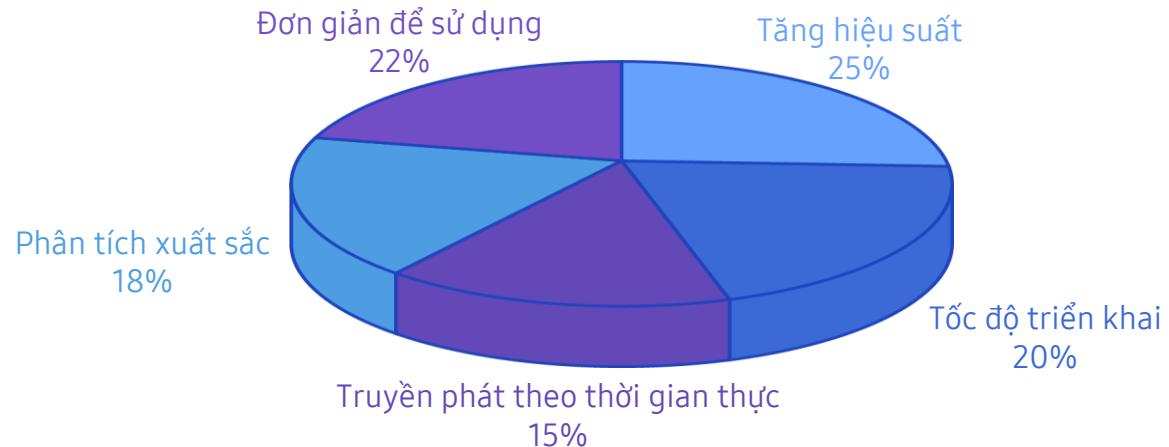


Kiến trúc linh hoạt



Tại sao Apache Spark lại phổ biến?

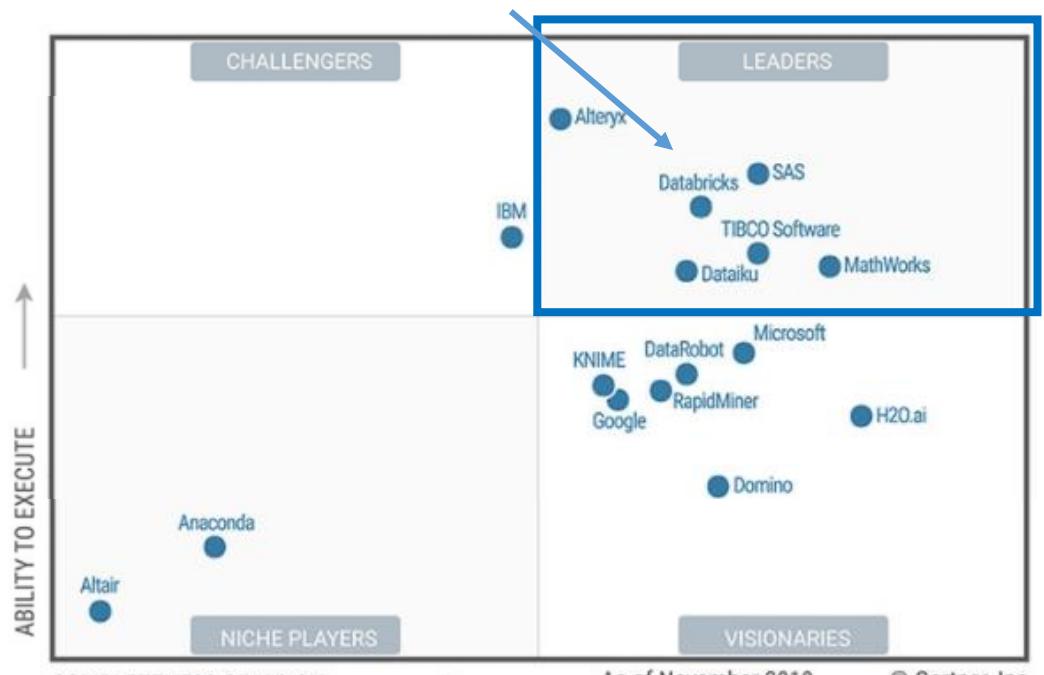
- Apache Spark là công cụ xử lý thế hệ tiếp theo trong Hệ sinh thái Hadoop
- Nó được sử dụng rộng rãi cho nhiều nhiệm vụ khác nhau:
 - ▶ ETL và làm sạch dữ liệu
 - ▶ Công việc hàng loạt SQL trên tập dữ liệu lớn
 - ▶ Xử lý dữ liệu phát trực tuyến từ cảm biến, IoT, hệ thống tài chính, v.v.
 - ▶ Học máy và suy luận



Gartner Magic Quadrant

- I Gartner Magic Quadrant là một nguồn phổ biến để xem xét các xu hướng công nghệ
 - ▶ Họ xem xét các sản phẩm dựa trên tầm nhìn đầy đủ và khả năng thực hiện
 - ▶ Các sản phẩm ở góc phần tư trên cùng bên phải là những sản phẩm hoàn chỉnh và được thực hiện tốt nhất
- I Apache Spark được đánh giá cao về khả năng hợp nhất dữ liệu và công việc máy học tải tất cả trong một nền tảng duy nhất
 - ▶ Tích hợp từ đầu đến cuối của khám phá dữ liệu, tiền xử lý dữ liệu, học máy để suy luận học máy

Databricks đã tạo Spark

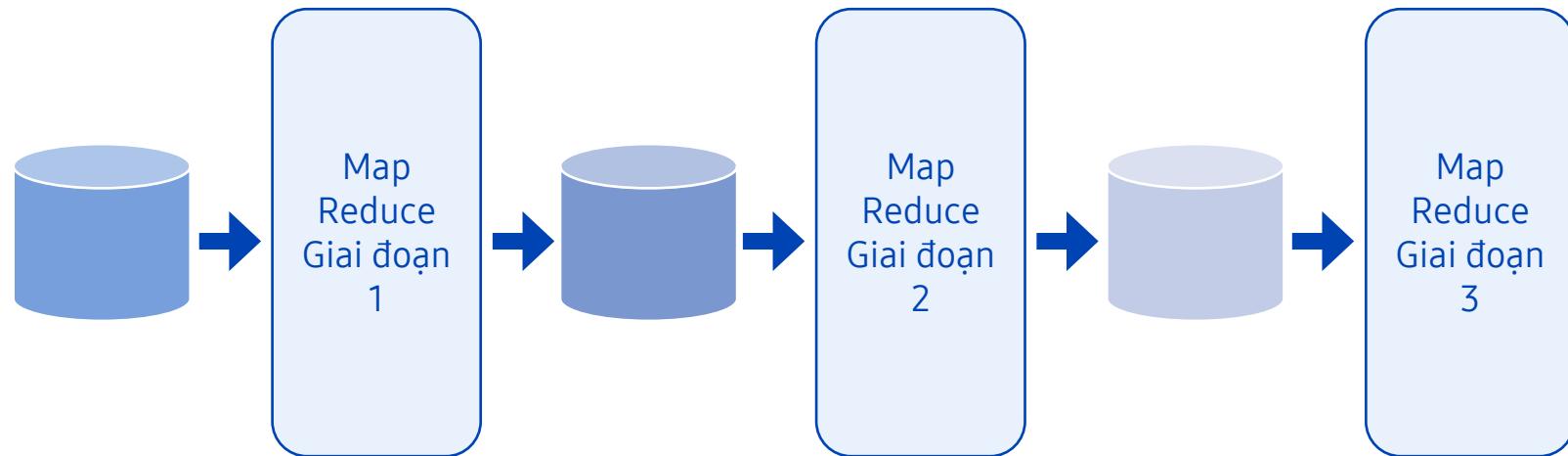


e: Gartner (February 2020)

© Gartner, Inc

Vấn đề về MapReduce

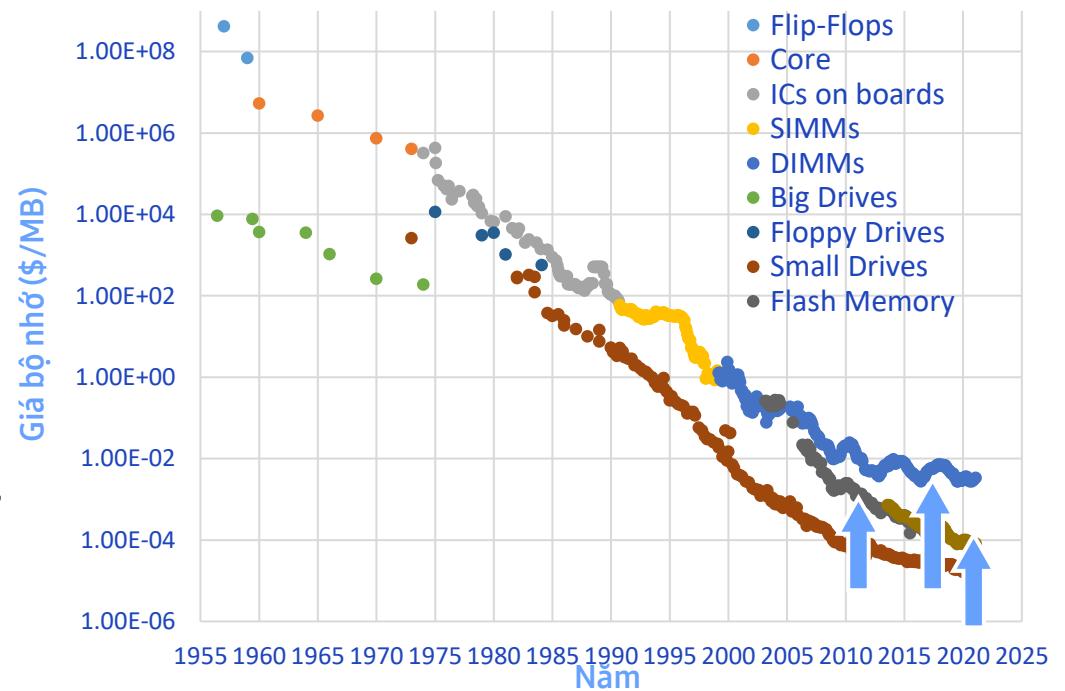
- Một công việc Map Reduce thường bao gồm nhiều chu trình Map và Reduce
- Mỗi chu kỳ Map yêu cầu đọc từ HDFS và ghi vào đĩa cục bộ
- Mỗi chu kỳ Reduce yêu cầu đọc từ đĩa cục bộ và ghi vào HDFS
- Đĩa I/O rất ĐẮT



Phát triển phần cứng: Chi phí và hiệu suất

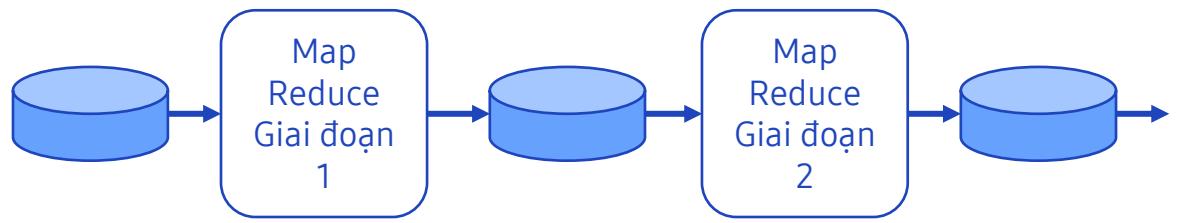
- Hadoop 0.14.1 được giới thiệu vào năm 2007
 - ▶ Cả HDFS và MapReduce, phụ thuộc nhiều vào lưu trữ dựa trên đĩa
- Apache Spark 0.7.0 được giới thiệu vào năm 2013
 - ▶ Chi phí bộ nhớ xấp xỉ 1/50 so với khi Hadoop được giới thiệu
- Apache Kudu 1.0 được giới thiệu vào năm 2016 và được tối ưu hóa để sử dụng bộ lưu trữ SSD

Chi phí lịch sử: Bộ nhớ và lưu trữ máy tính



Tính toán phân tán trong bộ nhớ

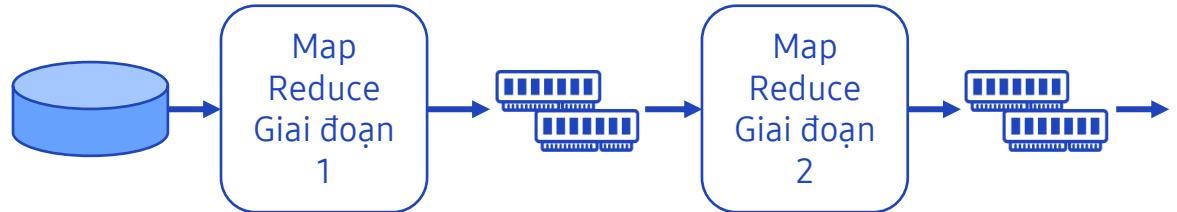
- Chỉ phát sinh chi phí I/O đĩa khi bắt đầu tính toán để tải dữ liệu vào bộ nhớ



- Sử dụng lưu trữ dựa trên bộ nhớ trong chu kỳ tính toán

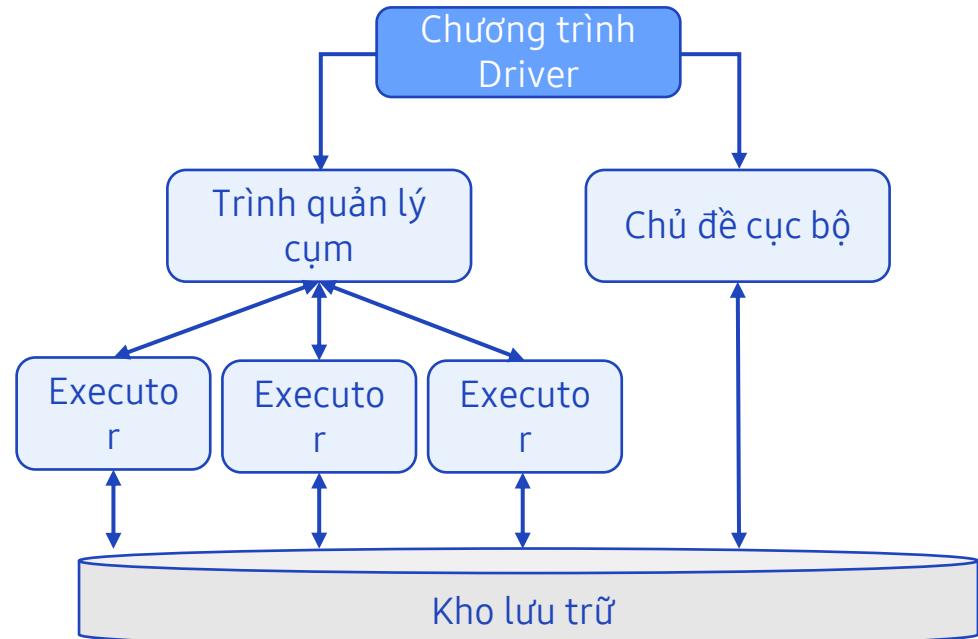


- Thường nhanh hơn **100 lần** so với MapReduce



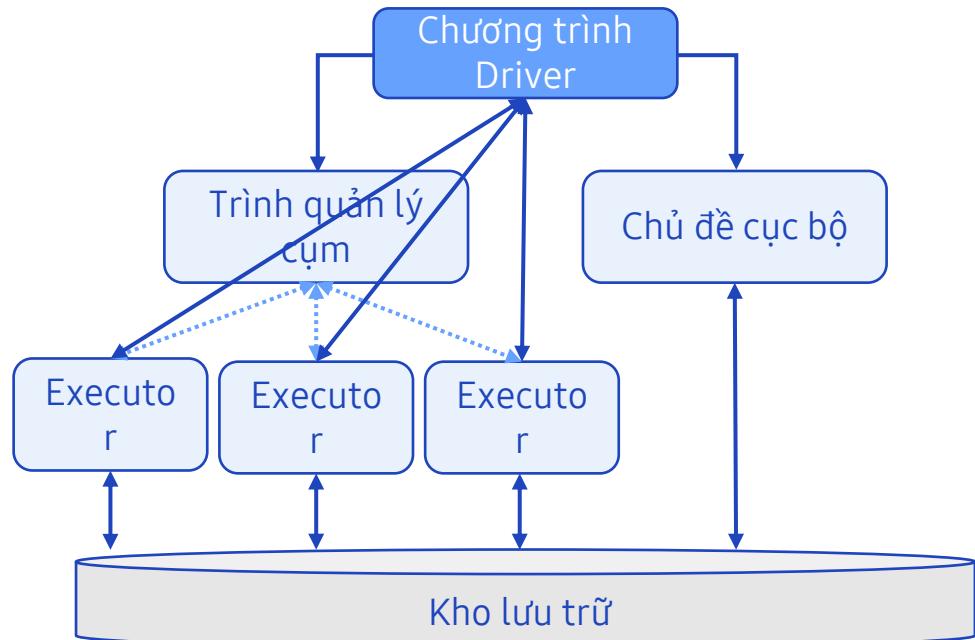
Spark Driver và các Executor (1/2)

- Apache Spark tuân theo kiểu kiến trúc master-slave
- Chương trình Driver đóng vai trò là chủ và điều phối việc tạo hoặc loại bỏ các Executor
- Trình điều khiển phân vùng dữ liệu và chuyển từng phân vùng cho một Executor để xử lý phân tán song song
- Mỗi Executor thực hiện tính toán thực tế trên phân vùng dữ liệu của nó
- Executor truyền lại kết quả cho Driver



Spark Driver và các Executor (2/2)

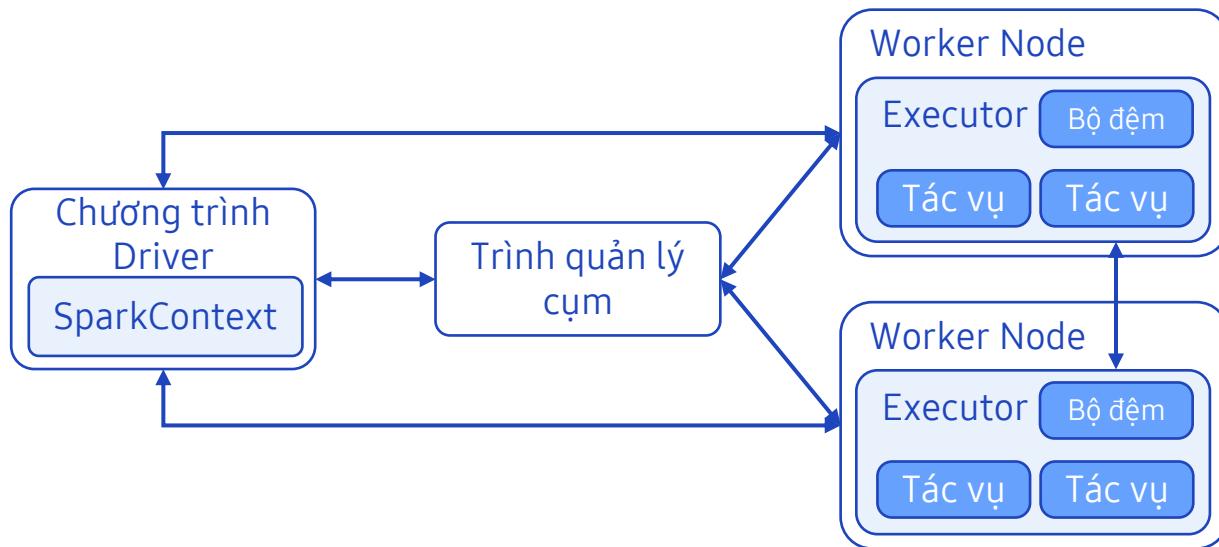
- Apache Spark tuân theo kiểu kiến trúc master-slave
- Chương trình Driver đóng vai trò là chủ và điều phối việc tạo hoặc loại bỏ các Executor
- Trình điều khiển phân vùng dữ liệu và chuyển từng phân vùng cho một Executor để xử lý phân tán song song
- Mỗi Executor thực hiện tính toán thực tế trên phân vùng dữ liệu của nó
- Executor truyền lại kết quả cho Driver



Đối tượng điểm vào SparkContext

Đối tượng SparkContext là điểm vào chính cho hàm Spark.

- ▶ Nó đại diện cho kết nối với cụm Spark
- ▶ Chương trình Trình điều khiển sử dụng SparkContext để thiết lập giao tiếp với người quản lý tài nguyên và cụm nhằm điều phối và thực hiện công việc



Các đối tượng điểm vào khác

- Đối với các phiên bản Spark trước Spark 2.x, có một số đối tượng điểm nhập bổ sung
 - SQLContext
 - ▶ Điểm vào Spark SQL và cung cấp chức năng SQL cơ bản
 - ▶ Thực hiện các hoạt động giống như SQL trên Bộ dữ liệu và Khung dữ liệu
 - HiveContext
 - ▶ Ngoài chức năng SQL cơ bản, cho phép giao tiếp với Apache Hive
 - ▶ Sử dụng thay cho SQLContext khi chức năng Hive là cần thiết
 - Spark Streaming Context
 - ▶ Điểm vào chính cho tất cả các chức năng truyền phát
 - ▶ Sử dụng bởi Spark Dstream API
- Post Spark 2.x, SparkSession cung cấp một đối tượng điểm vào thống nhất
 - ▶ Sử dụng SparkSession để tạo SparkContext
 - ▶ Sử dụng trực tiếp SparkSession để tích hợp SQL/Hive

Apache Spark Shell

- Apache Spark cung cấp môi trường shell để phát triển tương tác dễ dàng
- Shell cung cấp môi trường REPL (Đọc đánh giá vòng lặp in) nơi các nhà phát triển có thể nhận được kết quả trả về ngay lập tức mà không cần phải biên dịch mã của họ
- Tự động tạo các đối tượng mục nhập cốt lõi như Spark Context và Spark Session
- Chỉ khả dụng cho Scala và Python
 - không khả dụng cho Java và R
- Gọi bằng spark-shell cho Scala
- Gọi bằng pyspark cho Python

```
Welcome to
         _/\_ 
        / \ \_ \_ \_ \_ \_ \_ \_ \
        |   | .__\_\_/_/_/_/\_ \
        \_ /_/
version 3.1.2

Using Python version 3.8.11 (default, Aug 10 2021 21:35:17)
Spark context Web UI available at http://10.20.70.210:4040
Spark context available as 'sc' (master = local[*], app id = local-1629266540018).
SparkSession available as 'spark'.
>>> sc
<SparkContext master=local[*] appName=PySparkShell>
>>> spark
<pyspark.sql.session.SparkSession object at 0x7fd9e46bc250>
>>>
```

Spark Shell trên Máy tính xách tay Jupyter

- Jupiter có thể tích hợp Jupyter Notebook với Pyspark
 - Sau khi cài đặt Jupyter Notebook, hãy thêm phần sau vào phần khởi động trình bao Linux của bạn
 - ▶ Đối với bash, đây sẽ là .bashrc
- ```
export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
```
- Sau khi đặt hai biến môi trường, thay vào đó, việc gọi pyspark sẽ mở trong Jupyter

Bài 1.

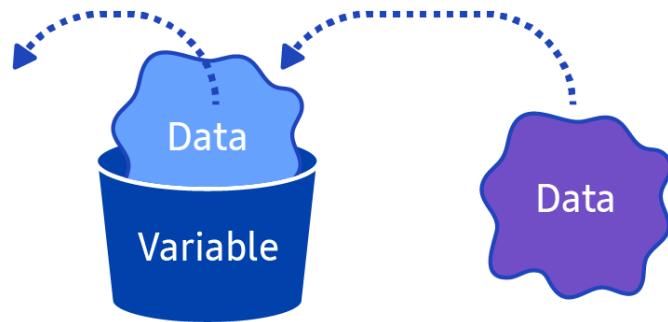
# Xử lý dữ liệu phi cấu trúc

- | 1.1. Giới thiệu về Apache Spark
- | **1.2. Khái niệm cơ bản về Python**
- | 1.3. Chuyển đổi dữ liệu với Core API
- | 1.4. Làm việc với Pair RDD

## Tạo biến

- Các biến được sử dụng để giữ các tham chiếu đến một số đối tượng
  - ▶ Chúng ta có thể chỉ đơn giản nghĩ về nó như là một giá trị cho chúng ta
- Biến Python không cần phải khai báo
- Các loại biến được phát hiện ngầm
- Tạo một biến bằng cách gán một giá trị cho nó

```
x = 1
```



# Quy tắc đặt tên (1/2)

- Tên có phân biệt chữ hoa chữ thường

```
Student pylon
```

- Chúng có thể chứa:

- ▶ Chữ cái
- ▶ Số
- ▶ Dấu gạch dưới

```
MyVariable myvariable_1
myvariable_student
```

## Quy tắc đặt tên (2/2)

- | Không thể bắt đầu bằng một số

```
2myvariable # ERROR
```

- | Từ dành riêng

- |> del from not while như elif global hoặc với khẳng định other if vượt qua ngắt suất ngoại trừ nhập lớp in exec trong nâng cao tiếp tục cuối cùng là trả về def cho lambda thử

# Kiểu dữ liệu số nguyên trong Python

## I Số nguyên

- ▶ Được gọi là “int” trong Python
- ▶ Trong Python 3, bộ nhớ là giới hạn duy nhất về mức độ lớn của một biến số nguyên
- ▶ Bạn có thể thay đổi cơ số thành thứ khác ngoài 10

| Tiền tố                          | Điễn dịch     | Hệ cơ số |
|----------------------------------|---------------|----------|
| 0b (zero + lowercase letter 'b') | Nhị phân      | 2        |
| 0B (zero + uppercase letter 'B') |               |          |
| 0o (zero + lowercase letter 'o') | Bát phân      | 8        |
| 0O (zero + uppercase letter 'O') |               |          |
| 0o (zero + lowercase letter 'o') | Thập lục phân | 16       |
| 0O (zero + uppercase letter 'O') |               |          |

## Kiểu dữ liệu dấu chấm động trong Python

### I Số dấu chấm động

- ▶ Được gọi là "float" trong Python
- ▶ Chỉ định bằng dấu thập phân

```
my_pi = 3.14
my_float = .3
```

### I Tùy chọn, sử dụng ký tự e hoặc E theo sau là số nguyên dương hoặc âm để chỉ định ký hiệu khoa học

```
my_var = 3.7e8
my_float = 2.17E-3
```

# Các kiểu dữ liệu cơ bản trong Python

## I Kiểu dữ liệu chuỗi

- ▶ Được gọi là "str" trong Python
- ▶ Chuỗi dữ liệu ký tự
- ▶ Tất cả các ký tự giữa dấu phân cách mở và dấu phân cách đóng phù hợp là một phần của chuỗi
- ▶ Dấu phân cách có thể là trích dẫn kép " hoặc trích dẫn đơn '
- ▶ Độ dài của chuỗi chỉ bị giới hạn bởi bộ nhớ

```
print("I am a string")
print('I am a string')
print("I'm a string")
print('I'm not a valid string')
```

# Kiểu dữ liệu Boolean trong Python

## I Kiểu dữ liệu Boolean

- ▶ Được gọi là "bool" trong Python
- ▶ Các kiểu Boolean có thể là True hoặc False

```
is_male = False
is_student = True
```

## Các kiểu dữ liệu tập hợp trong Python (1/3)

### Loại bộ sưu tập mô tả chức năng

| Kiểu dữ liệu | Mô tả                                                           | Ví dụ                                   |
|--------------|-----------------------------------------------------------------|-----------------------------------------|
| List         | Chuỗi các đối tượng kiểu dữ liệu khác nhau có thể được cập nhật | [1, 'abc', [1,2]]                       |
| Tuple        | Chuỗi các đối tượng kiểu dữ liệu khác nhau không thể cập nhật   | ('abc', 142)                            |
| Set          | Bộ sưu tập không có thứ tự của các đối tượng riêng biệt         | {1,5,2}                                 |
| Dictionary   | Bao gồm nhiều cặp Khóa-Giá trị với các khóa duy nhất            | {'fname' : 'Student', 'lname' : 'Good'} |

## Các kiểu dữ liệu tập hợp trong Python (2/3)

I Các loại bộ sưu tập cơ bản được phân biệt bởi các đặc điểm sau

- ▶ Đã sắp xếp - Danh sách có thể sắp xếp được không?
- ▶ Có thể thay đổi – Bạn có thể sửa đổi các giá trị không?
- ▶ Cho phép trùng lặp – Có thể có các giá trị trùng lặp không?

| Tên        | Sắp xếp | Có thể thay đổi | Cho phép trùng lặp |
|------------|---------|-----------------|--------------------|
| List       | Có      | Có              | Có                 |
| Tuple      | Có      | Không           | Có                 |
| Set        | Không   | Cả hai          | Không              |
| Dictionary | Không   | Có              | Không              |

## Các kiểu dữ liệu tập hợp trong Python (3/3)

I Tạo bộ sưu tập với các mục được phân tách bằng dấu phẩy với toán tử có liên quan:

- ▶ List sử dụng dấu ngoặc vuông [ ]
- ▶ Tuple sử dụng dấu ngoặc đơn ( )
- ▶ Set dấu ngoặc nhọn { }
- ▶ Dictionary sử dụng dấu ngoặc nhọn với key:value items

Input

```
my_list = [1.2, 5, "hello"]
my_tuple = (1.2, 5, "hello")
my_set = {1, 2, 3, 4}
my_dict = {1: "one", 2: "two", 3: "three", 4: "four"}
print(my_list)
print(my_tuple)
print(my_set)
print(my_dict)
```

Output

```
[1.2, 5, 'hello']
(1.2, 5, 'hello')
{1, 2, 3, 4}
{1: 'one', 2: 'two', 3:
'three', 4: 'four'}
```

## Tạo chuỗi

- Tạo Chuỗi bằng "trích dẫn" hoặc "trích dẫn đơn"

```
string1 = "I am a string"
string2 = 'I am a string also'
```

- Nếu cần đặt dấu ngoặc đơn trong câu thì dùng dấu ngoặc kép để tạo thành câu và ngược lại

```
string3 = "I don't like this"
string4 = 'I am the "best" string'
```

## Ký tự dấu gạch chéo ngược

### Một số ký tự có ý nghĩa đặc biệt

- ▶ Ví dụ: nếu chúng tôi bắt đầu một chuỗi bằng một trích dẫn, một trích dẫn phù hợp sẽ kết thúc chuỗi
- ▶ Nếu chúng ta không muốn điều đó xảy ra thì sao?

```
string5 = "You can use a \" or \' to create strings in Python"
```

### Đôi khi bạn muốn đưa ra ý nghĩa đặc biệt

- ▶ \t cho tab và \n cho dòng mới

```
string6 = "I am \t special \n really special"
```

## Chuỗi cắt lát

- | Bạn chỉ có thể truy cập một "lát cắt" hoặc một phần của chuỗi
- | Các chuỗi được coi là một mảng hoặc danh sách các ký tự và có chỉ mục

| S  | T  | R  | I  | N  | G  |
|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  |
| -6 | -5 | -4 | -3 | -2 | -1 |

- | Sử dụng [index] hoặc [start\_index:end\_index\_not\_inclusive] để cắt chuỗi

```
my_string = "my_python"
print(my_string[4])
print(my_string[0:6])
print(my_string[1:-1])
print(my_string[:-2])
print(my_string[2:])
print(my_string[:])
```

Input

```
y
my_pyt
y_pytho
my_pyth
_python
my_python
```

Output

## Nối và nhân chuỗi

- Sử dụng + để nối các chuỗi

```
"Hello Student " + "Welcome to the Apache Spark Lessons"
```

- Nhân chuỗi với \*

```
border_Bài = "*-*"
print(border_Bài * 5)
```

- Trộn các biến

```
student_name = "Good Student"
"Hello " + student_name + " Welcome to the Apache Spark Class"
```

## Các phương thức chuỗi hữu ích (1/2)

### I `lower()`, `upper()`

- ▶ trả về chữ thường hoặc chữ hoa của chuỗi

### I `len()`

- ▶ trả về độ dài của chuỗi

### I `index(letter hoặc "string")` hoặc `find()`

- ▶ trả về chỉ số xuất hiện đầu tiên của chữ cái hoặc chuỗi

### I `replace("this", "that")`

- ▶ thay cái này bằng cái kia

### I `strip()`

- ▶ loại bỏ bất kỳ khoảng trắng ở phía trước và phía sau

### I `split("delimiter")`

- ▶ phân tích đoạn văn bằng dấu phân cách

## Các phương thức chuỗi hữu ích (2/2)

### I Cách sử dụng phương thức chuỗi

```
txt = " Students learn pyspark "
```

```
print(txt.lower())
print(len(txt))
print(txt.index("e"))
print(txt.replace("learn", "teach"))
print(txt.strip())
print(txt.split(" "))
```

**Input**

```
students learn pyspark
26
6
Students teach pyspark
Students learn pyspark
[', ', 'Students', 'learn', 'pyspark', ', ']
```

**Output**

# Toán tử số học cơ bản

Đây là các toán tử toán học cơ bản trong Python

| Toán tử | Tên         | Ví dụ    |
|---------|-------------|----------|
| +       | Phép cộng   | $x + y$  |
| -       | Phép trừ    | $x - y$  |
| *       | Phép nhân   | $x * y$  |
| /       | Phép chia   | $x / y$  |
| %       | Mô đun      | $x \% y$ |
| //      | Phần nguyên | $x // y$ |
| **      | Lũy thừa    | $x ** y$ |

## Toán tử so sánh

Sử dụng các toán tử so sánh trên nhiều loại dữ liệu, bao gồm các loại dữ liệu phức tạp như bộ sưu tập

| Toán tử            | Tên               | Ví dụ                  |
|--------------------|-------------------|------------------------|
| <code>==</code>    | Bằng nhau         | <code>x == y</code>    |
| <code>!=</code>    | không bằng nhau   | <code>x != y</code>    |
| <code>&gt;</code>  | Lớn hơn           | <code>x &gt; y</code>  |
| <code>&lt;</code>  | Nhỏ hơn           | <code>x &lt; y</code>  |
| <code>&gt;=</code> | Lớn hơn hoặc bằng | <code>x &gt;= y</code> |

## Toán tử logic

Toán tử logic được sử dụng để kết hợp các điều kiện cơ bản thành nhiều điều kiện phức tạp

| Toán tử | Mô tả                                                                                   | Ví dụ                                      |
|---------|-----------------------------------------------------------------------------------------|--------------------------------------------|
| and     | Trả về True nếu cả hai câu đều đúng                                                     | $x \geq 4 \text{ and } x \leq 8$           |
| or      | Trả về True nếu một trong các câu là đúng                                               | $x > 100 \text{ or } x < 50$               |
| not     | Trả về kết quả ngược lại với kết quả đã đánh giá. Trả về True nếu sai và False nếu đúng | <code>not (x &gt; 100 or x &lt; 50)</code> |

# Toán tử nhận dạng

## Toán tử nh nhận dạng khác với so sánh đẳng thức

- ▶ So sánh bình đẳng chỉ đơn giản là so sánh với "giá trị" bằng nhau
- ▶ Các toán tử nh nhận dạng so sánh đối tượng thực tế - tương đương và nằm trong cùng một vị trí bộ nhớ

| Toán tử | Mô tả                                             | Ví dụ      |
|---------|---------------------------------------------------|------------|
| is      | Trả về True nếu cả hai biến là cùng một đối tượng | x is y     |
| is not  | Trả về True nếu các biến không cùng đối tượng     | x is not y |

## Toán tử membership

Được sử dụng để kiểm tra tư cách thành viên trong các loại dữ liệu bộ sưu tập

| Toán tử | Mô tả                                                                        | Ví dụ      |
|---------|------------------------------------------------------------------------------|------------|
| in      | Trả về True nếu trình tự được chỉ định nằm trong đối tượng được so sánh      | x in y     |
| not in  | Trả về True nếu trình tự được chỉ định KHÔNG có trong đối tượng được so sánh | x not in y |

## Phép gán kép

- Các phép gán kép là các toán tử phím tắt khi toán tử được áp dụng cho một biến và được gán cho biến đó, ghi đè lên biến đó

| Toán tử          | Ví dụ                | Tương đương              |
|------------------|----------------------|--------------------------|
| <code>+=</code>  | <code>x += 5</code>  | <code>x = x + 5</code>   |
| <code>-=</code>  | <code>x -= 2</code>  | <code>x = x - 2</code>   |
| <code>*=</code>  | <code>x *= 4</code>  | <code>x = x * 4</code>   |
| <code>/=</code>  | <code>x /= 2</code>  | <code>x = x / 2</code>   |
| <code>%=</code>  | <code>x %= 3</code>  | <code>x = x % 3</code>   |
| <code>//=</code> | <code>x //= 6</code> | <code>x = x //= 6</code> |
| <code>**=</code> | <code>x **= 2</code> | <code>x = x ** 2</code>  |

# Độ ưu tiên toán tử trong Python

Các toán tử được ưu tiên chỉ định thứ tự áp dụng các toán tử

| Toán tử                                         | Ý nghĩa                                      |
|-------------------------------------------------|----------------------------------------------|
| ( )                                             | Dấu ngoặc đơn                                |
| **                                              | Số mũ                                        |
| +x, -x                                          | Cộng một ngôi, trừ một ngôi                  |
| *, /, //, %                                     | Phép nhân, phép chia, phép chia tầng, mô đun |
| +, -                                            | Phép cộng, phép trừ                          |
| ==, !=, >, >=, <, <=, <, is, is not, in, not in | So sánh, Bản sắc, Tư cách thành viên         |
| not                                             | Logic NOT                                    |
| and                                             | Logic AND                                    |
| or                                              | Logic OR                                     |

## Ra quyết định với câu lệnh If

- Trong Python, chúng tôi sử dụng câu lệnh if để đưa ra quyết định và thực thi một hoặc một khối mã Python khác
- Như trong tất cả các khối mã Python khác, dấu hai chấm bắt đầu khối và toàn bộ khối phải được thụt vào

```
if <condition> :
 <true-block>
 ...
 Instructions that are executed
 if the condition evaluates to True
 ...
else :
 <false-block>
 ...
 Instructions that are executed
 if the condition evaluates to False
 ...
```

## Ra quyết định phức tạp

- I Sử dụng câu lệnh **if/elif** cho nhiều điều kiện
- I Kết hợp **if/else** với **if/elif** để đưa ra quyết định phức tạp hơn
- I Cả **if/elif** và **if/else** đều có thể được lồng vào nhau
- I Khi so sánh thông qua câu lệnh **if/elif**, ngay khi một điều kiện được thỏa mãn, các khối tiếp theo sẽ không được đánh giá

```
if <condition1> :
 <true-block>
 # satisfying conditon1
elif <condition2> :
 <true_block>
 # not satisfying conditon1 but
 condition2
elif <condition3> :
 <true_block>
...
else :
 <false-block>
 # not satisfying any condition above
```

## Lặp lại mã với câu lệnh While

- Khi bạn lập trình, sẽ có lúc bạn thường muốn lặp đi lặp lại một phần mã trong khi chỉ thay đổi một phần nhỏ mã.
  - ▶ Cộng từ 1 đến 100
  - ▶ Lặp lại khi (các) điều kiện nhất định được đáp ứng hoặc không được đáp ứng
- Hầu hết các ngôn ngữ lập trình đều cung cấp điều khiển giống như "while" cho điều này

```
while <condition> :
 <block>
 ...
 repeat the statements in <block>
 while the <condition> is True
 If not, escape the while loop.
 ...
```

## Lặp lại với câu lệnh vòng lặp For

- Một vòng lặp for được sử dụng để lặp qua một chuỗi
- Một trình tự nó là một loại bộ sưu tập có thể lặp lại
  - ▶ Chuỗi và Danh sách
  - ▶ Những thứ khác bao gồm tuple, từ điển, bộ, v.v.
- Không giống như vòng lặp while trong đó điều kiện được sử dụng để kết thúc vòng lặp, trình vòng lặp for có điểm bắt đầu và kết thúc rõ ràng

```
for <item> in <iterable> :
 <block>
 ...
 repeat the statements in <block>
 for each element of the <iterable>, in
 order
 ...
```

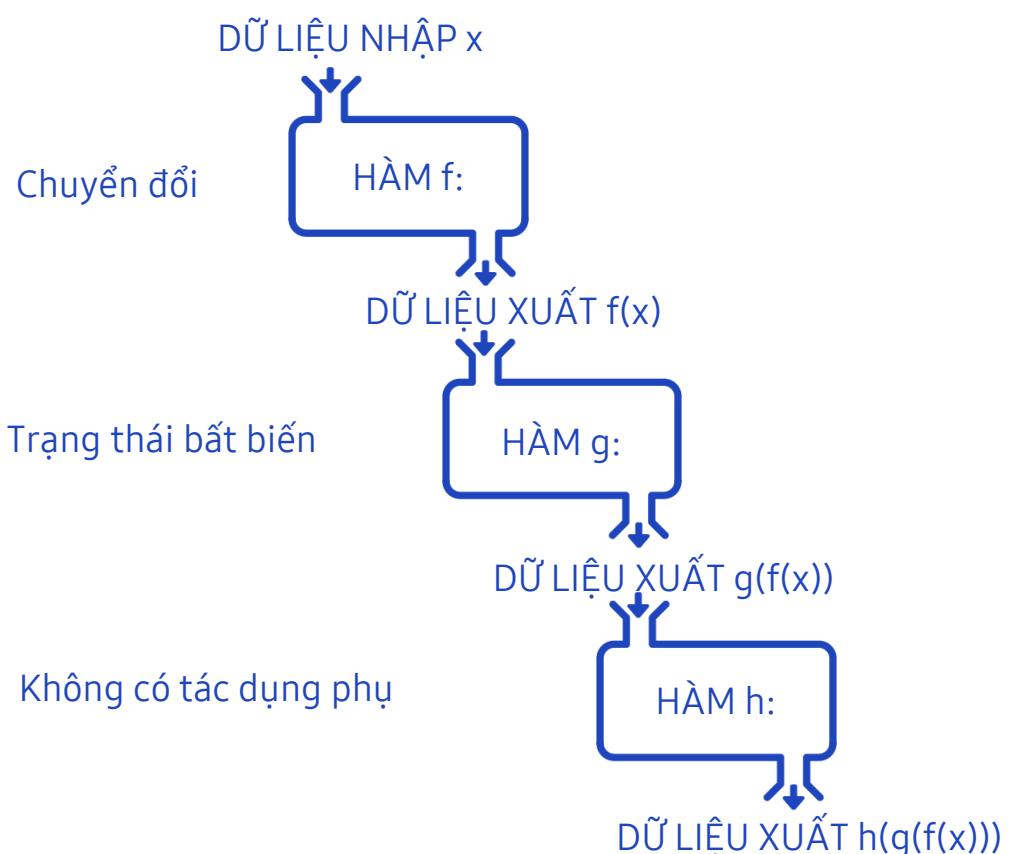
## Tạo các hàm Python

- | Tạo một Hàm với từ khóa **def** theo sau là tên hàm
- | Tiếp theo, bất kỳ tham số đầu vào nào được đặt bên trong dấu ngoặc đơn ()
- | Tiếp theo, dấu hai chấm bắt đầu khối mã phải được thụt vào
- | Câu lệnh đầu tiên của hàm có thể là một chuỗi tài liệu tùy chọn có tên là Docstring
- | Câu lệnh cuối cùng của hàm là return dùng để thoát và trả về kết quả của hàm

```
def my_function (parameter1, parameter2, . . .) :
 """ Documentation about this function """
 code
 code
 . . .
 return some_result
```

# Lập trình hàm

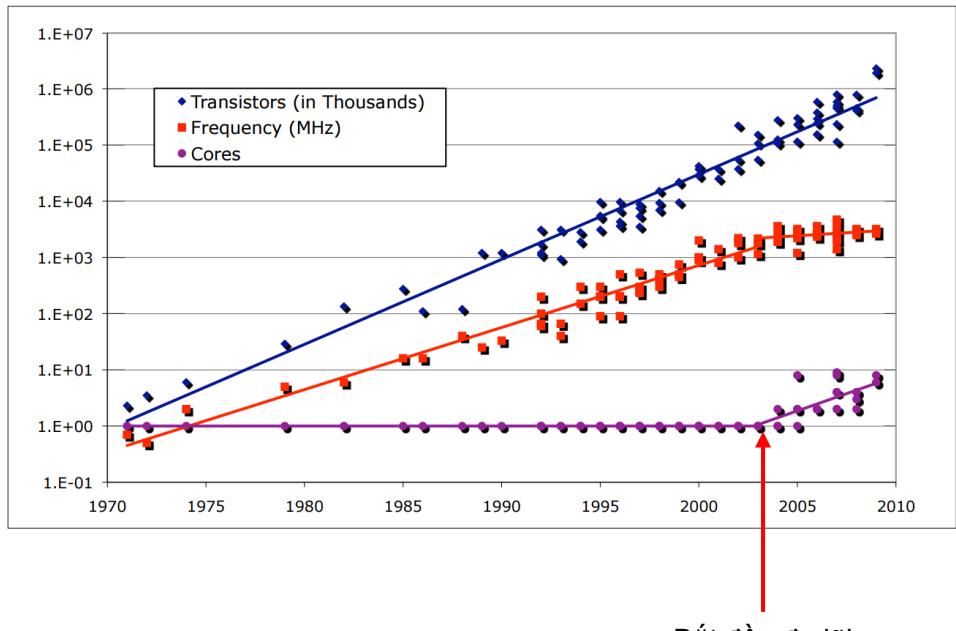
- | Lập trình hàm là một phong cách tính toán trong đó các chương trình hoàn toàn bao gồm việc đánh giá các hàm thuần túy
- | Một hàm thuần túy là một hàm có giá trị đầu ra hoàn toàn là kết quả của các giá trị đầu vào của nó mà không gây ra bất kỳ tác dụng phụ nào
- | Tác dụng phụ là khi chức năng gây ra những thay đổi trong môi trường gọi theo bất kỳ cách nào
  - ▶ Ví dụ: Sửa đổi giá trị của một đối số trong một hàm



# Tại sao phải lập trình hàm?

- Tất cả chúng ta đã và đang sử dụng phong cách lập trình bắt buộc hướng tới các cỗ máy kiến trúc von Neumann
  - ▶ Mô hình lập trình mệnh lệnh sử dụng các câu lệnh thay đổi trạng thái của chương trình
  - ▶ Lập trình thủ tục và lập trình hướng đối tượng đều là các phong cách lập trình bắt buộc
- Kể từ sau năm 2000 một chút, đã có một sự thay đổi cơ bản trong phần cứng máy tính
  - ▶ Không còn cải thiện về tốc độ xung nhịp
  - ▶ Đa lõi
- Lập trình bắt buộc và đa lõi không hoạt động tốt với nhau
  - ▶ Tại sao? Gợi ý: Các chương trình bắt buộc hoạt động bằng cách thay đổi trạng thái của chương trình

Dữ liệu từ Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, et al



# Lập trình đồng thời và song song

## I Lập trình song song

- ▶ Thực thi mã đồng thời trên nhiều phần cứng song song
- ▶ Hadoop và Spark đều hoạt động trên các cụm có nhiều máy tính

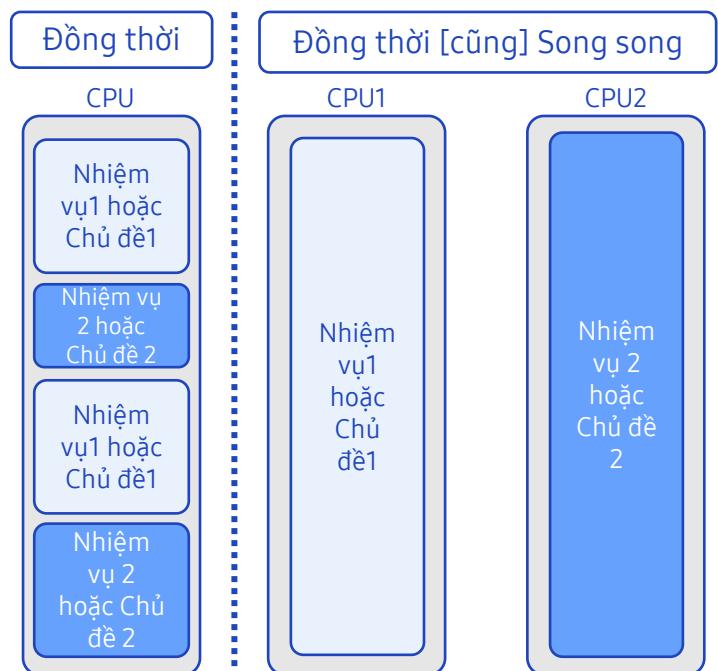
## II Lập trình đồng thời

- ▶ Thực thi mã trên nhiều lõi trong một máy
- ▶ Mỗi lõi tiến bộ cùng một lúc
- ▶ Hadoop và Spark sử dụng nhiều lõi trong việc thực thi các tác vụ

## III Cả lập trình song song và lập trình đồng thời đều KHÓ!

## IV Tính không xác định gây ra bởi các luồng đồng thời truy cập các trạng thái chung được chia sẻ và sửa đổi trạng thái

- ▶ Sự kết hợp giữa xử lý song song/dòng thời và trạng thái có thể thay đổi đang gây ra sự cố
- ▶ Tránh trạng thái có thể thay đổi, làm cho trạng thái không thể thay đổi



# Lập trình hàm trong Python

- | Để hỗ trợ Lập trình chức năng, các chức năng phải là công dân hạng nhất
- | Công dân hạng nhất có nghĩa là gì
  - ▶ Có các đặc điểm giống nhau như các giá trị khác như số, chuỗi hoặc bộ sưu tập
  - ▶ Có thể lưu các chức năng vào các biến
  - ▶ Truyền hàm làm đối số cho hàm
  - ▶ Nhận chức năng là kết quả của một chức năng

```
def some_func():
 print("some function")

save to variable
my_variable = some_func

#pass it to another function
print(1, "mystring", some_func)

#return from another function
def another_func(func_parameter):
 return func_parameter()

another_func(some_func)
```

## Hàm Lambda trong Python

- Khi làm việc với các đối tượng khác trong Python, không phải lúc nào chúng ta cũng lưu nó vào một biến trước khi sử dụng

```
my_float = 2.3
a_function(my_float) OR a_function(2.3)
```

- Tương tự, chúng ta không cần phải luôn xác định một hàm để sử dụng nó.
- Hàm ẩn danh trong Python cho phép chúng ta xác định hàm một cách nhanh chóng mà không cần xác định nó
  - Sử dụng ký hiệu **lambda**

```
def my_func(str):
 return str.lower()

a_function(my_func) OR a_function(lambda str: str.lower())
```

Bài 1.

# Xử lý dữ liệu phi cấu trúc

- | 1.1. Giới thiệu về Apache Spark
- | 1.2. Khái niệm cơ bản về Python
- | **1.3. Chuyển đổi dữ liệu với Core API**
- | 1.4. Làm việc với Pair RDD

## Bắt đầu với API lõi (1/2)

- Cách dễ nhất để bắt đầu với Spark Development là từ Spark Shell
- Shell có thể được bắt đầu với một số tùy chọn
- Vị trí nơi chương trình Trình điều khiển được tạo có thể được xác định bằng các tùy chọn **--master** và **--deploy-mode**
  - ▶ local[N] - Sử dụng chế độ cục bộ với N luồng
  - ▶ local[\*] - Sử dụng chế độ cục bộ với các chủ đề khả dụng tối đa
  - ▶ máy khách - Sử dụng cụm trong chế độ máy khách
  - ▶ cụm - Sử dụng cụm trong chế độ cụm
- Bất kỳ jar Java bổ sung nào có **--jars**
- Thêm các tệp Python bổ sung với **--py-files**
- Thêm các gói bổ sung với **--packages**

## Bắt đầu với API lõi (2/2)

### I Nhiều cách khác nhau để bắt đầu Spark

- ▶ Chạy spark-shell cục bộ

```
$ spark-shell --master local[4]
```

- ▶ Chạy pyspark trên cụm SƠI

```
$ pyspark --master yarn --deploy-mode client
```

- ▶ Spark-shell với Java jar bổ sung

```
$ spark-shell --jars jarfile1, jarfile2
```

- ▶ Chạy pyspark cục bộ với các tệp python bổ sung

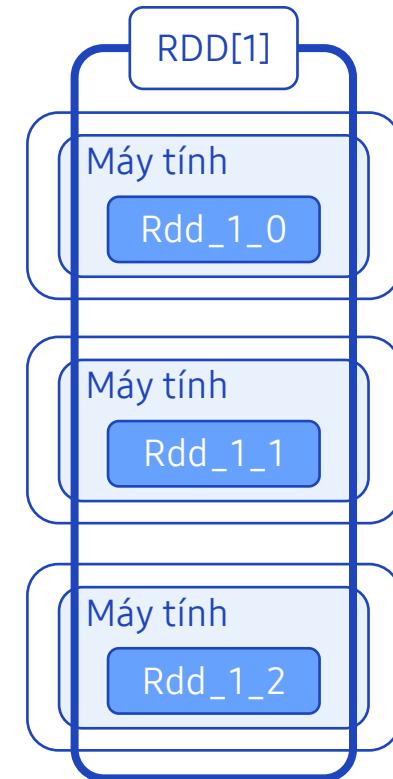
```
$ py-spark --master local[*] --py-files py-file1, py-file2
```

- ▶ Chạy pyspark cục bộ với các gói bổ sung

```
$ py-spark --master local[*] --packages org.apache.hadoop:hadoop-aws:3.1.2
```

## Bộ dữ liệu phân tán đòn hồi

- Bộ dữ liệu phân tán có khả năng phục hồi (RDD) là một cấu trúc dữ liệu cơ bản trong Spark
- Trừu tượng hóa dữ liệu chính trong Apache Spark và API Spark Core
- RDD là tập hợp các đối tượng phân tán không thay đổi có khả năng chịu lỗi có thể được vận hành song song
  - ▶ Spark phân vùng dữ liệu qua Executor
  - ▶ Mỗi Executor hoạt động trên phần dữ liệu của nó
  - ▶ Tất cả các Executor hoạt động song song
  - ▶ Nếu bất kỳ dữ liệu nào bị mất, Spark sẽ tự động tạo lại dữ liệu đó



## Loại dữ liệu nội dung RDD

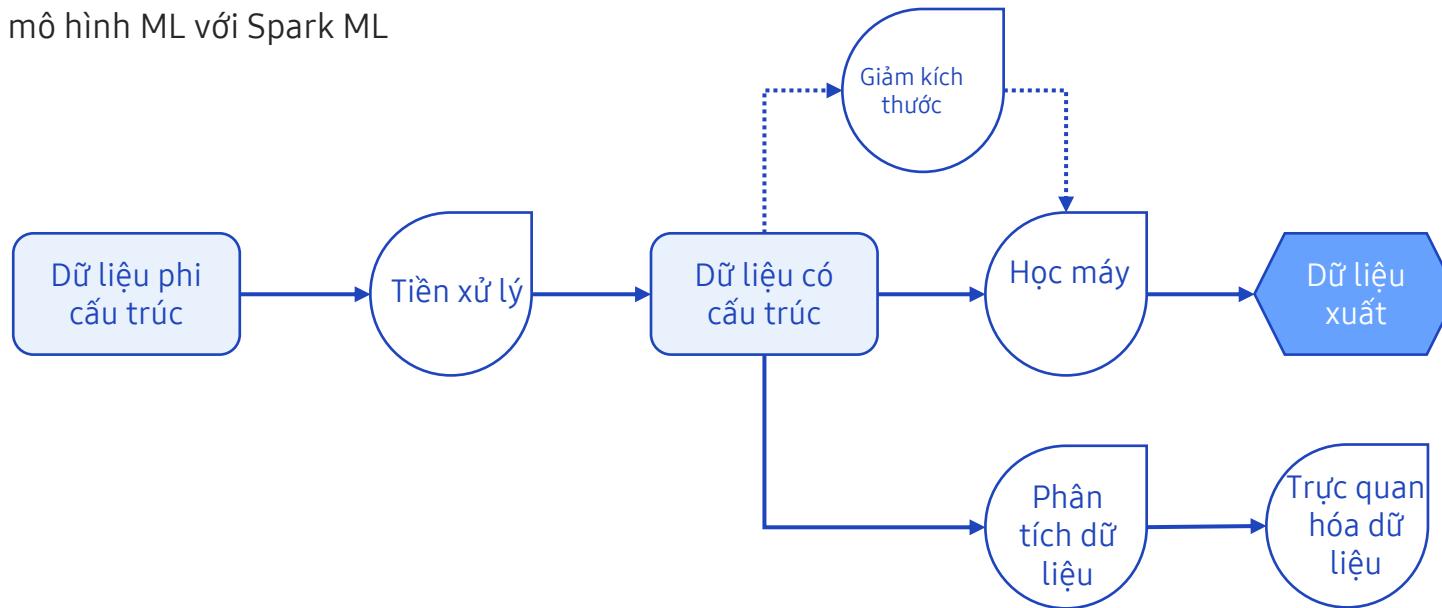
### Loại dữ liệu nào có thể được lưu trữ trong RDD?

- ▶ Các kiểu dữ liệu nguyên thủy như số nguyên, float, double, chuỗi, ký tự, Boolean, v.v.
- ▶ Các kiểu dữ liệu phức tạp như danh sách, mảng, chuỗi, bộ dữ liệu, từ điển
- ▶ Các loại phức hợp lồng nhau, bao gồm các loại hỗn hợp
- ▶ Bất kỳ đối tượng Scala hoặc Java có thể tuân tự hóa nào



# RDD: Trường hợp sử dụng phổ biến nhất

- Trong hầu hết các trường hợp, RDD sẽ bao gồm dữ liệu phi cấu trúc hoặc bán cấu trúc
- Một trường hợp sử dụng rất phổ biến là xử lý trước dữ liệu phi cấu trúc/bán cấu trúc và chuyển đổi thành tập dữ liệu có cấu trúc
  - Truy vấn tập dữ liệu có cấu trúc bằng các công cụ khác như Spark SQL, Hive hoặc Impala
  - Tạo các mô hình ML với Spark ML



## Tạo bộ dữ liệu phân tán đòn hồi

- Phương pháp dễ nhất và đơn giản nhất để tạo RDD là tạo chúng bằng cách sử dụng các bộ lặp hoặc bộ sưu tập hiện có trong chương trình trình điều khiển
- Sử dụng phương pháp **song song** của SparkContext
- Hoạt động tạo ra một bộ sưu tập song song trong đó bộ sưu tập được phân vùng và phân phối trên các Executor tham gia
  - ▶ Kiểm soát số lượng phân vùng với **parallelize(my\_collection, number\_of\_partitions)**
- Chủ yếu được sử dụng trong quá trình phát triển để thử nghiệm nhanh

```
my_collection = [1, 2, 3, 4]
parallel_collection = sc.parallelize(my_collection)
parallel_collection.take(2)
```

Input

[1, 2]

Output

## Đọc bộ dữ liệu bên ngoài

- Spark có thể tạo bộ dữ liệu phân tán từ bất kỳ nguồn lưu trữ nào được Hadoop hỗ trợ
  - ▶ Hệ thống tệp cục bộ hoặc hệ thống tệp HDFS
  - ▶ NoSQL chẳng hạn như HBase và Cassandra
  - ▶ Các nguồn đám mây như Amazon S3
- Bộ dữ liệu phi cấu trúc hoặc bán cấu trúc nên được đọc bằng API lõi
  - ▶ Các tệp văn bản phi cấu trúc như tệp ngôn ngữ tự nhiên hoặc bán cấu trúc như tệp nhật ký
- Bộ dữ liệu có cấu trúc nên được đọc bằng API Dataframe
  - ▶ Các tệp nhị phân bao gồm thông tin lược đồ như Parquet, ORC, v.v.
  - ▶ Dữ liệu từ các bảng như Hive, HBase hoặc Cassandra
- Một số tệp có thể được đọc bằng API lõi hoặc API Dataframe, tùy thuộc vào trường hợp sử dụng
  - ▶ Tệp văn bản bán cấu trúc chẳng hạn như Tệp JSON, XML hoặc CSV

## Tạo RDD từ tệp văn bản

### I Tạo tệp văn bản RDD bằng phương thức `textFile` của SparkContext

- ▶ `sc.textFile(<đường dẫn đến tập dữ liệu>)`
- ▶ Đường dẫn đến tập dữ liệu có thể là thư mục hoặc tệp cụ thể
- ▶ Sử dụng đường dẫn tuyệt đối hoặc đường dẫn tương đối
- ▶ Đường dẫn tương đối mặc định là thư mục chính HDFS của người dùng
- ▶ Hỗ trợ ký tự đại diện

```
hdfs_abs_wild = sc.textFile("/labs/*.txt")
print("Dataset using wild card: ", hdfs_abs_wild.count())
hdfs_abs_list = sc.textFile("/labs/data1.txt,/labs/data2.txt")
print("Dataset from a list: ", hdfs_abs_list.count())
hdfs_directory = sc.textFile("/labs")
print("Dataset from directory: ", hdfs_directory.count())
```

Input

```
Dataset using wild card: 202
Dataset from a list: 202
Dataset from directory: 202
```

Output

## Chỉ định tệp cục bộ hoặc tệp HDFS

- I Chỉ định URI cụ thể để truy cập HDFS hoặc tệp cục bộ
  - ▶ `hdfs://<namenode host>/<đường dẫn đến tập dữ liệu>`
  - ▶ `tệp:/<đường dẫn đến tập dữ liệu cục bộ>`
- I Một số cân nhắc cho hệ thống tệp cục bộ
  - ▶ Nếu truy cập tệp cục bộ, Spark yêu cầu tệp cục bộ tồn tại trên tất cả các nút trong cụm tại cùng một vị trí

```
hdfs_uri_file = sc.textFile("hdfs://localhost:9000/user/student/alice.txt") Input
print("Alice from HDFS URI path: ", hdfs_uri_file.count())
hdfs_relative_path = sc.textFile("alice.txt")
print("Alice from relative path file: ", hdfs_relative_path.count())
local_file = sc.textFile("file:/home/student/Data/alice_in_wonderland.txt")
print("Alice from local home directory: ", local_file.count())
```

|                                  |      |        |
|----------------------------------|------|--------|
| Alice from HDFS URI path :       | 3761 | Output |
| Alice from relative path file:   | 3761 |        |
| Alice from local home directory: | 3761 |        |

## Tạo RDD từ s3 Bucket

- █ Chỉ định URI cụ thể để truy cập các tệp S3
  - ▶ s3a://<bộ chứa và đường dẫn đến bộ dữ liệu>
- █ Khởi động Shell với gói sau
  - ▶ pyspark --packages=com.amazonaws:aws-java-sdk-bundle:1.11.271, org.apache.hadoop:hadoop-aws:3.1.2
- █ Cung cấp cài đặt cấu hình cho khóa truy cập, khóa bí mật và hệ thống tệp

```
access_key = "xxxxxxxxxxxxxxxxxxxxxx"
secret_key = "xxxxxxxxxxxxxxxxxxxxxx"
hadoop_conf=sc._jsc.hadoopConfiguration()
hadoop_conf.set("fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")
hadoop_conf.set("fs.s3a.access.key", access_key)
hadoop_conf.set("fs.s3a.secret.key", secret_key)
s3_file = sc.textFile("s3a://samsung-student-lab/alice_in_wonderland.txt")

print("Alice from s3 directory: ", s3_file.count())
```

Input

Alice from s3 directory: 3761

Output

## Spark đọc văn bản như thế nào?

- | Khi Spark đọc tệp văn bản, mỗi dòng mới được sử dụng để phân tách các thành phần của RDD
- | Chúng tôi thấy rằng Spark hiển thị RDD dưới dạng Danh sách các chuỗi
  - ▶ Python sử dụng dấu ngoặc vuông [ ] cho danh sách
- | Mọi khoảng trắng, tab, dấu phẩy , dấu ngoặc kép, v.v. đều là một phần của phần tử



"How doth the little crocodile Improve his shining tail, And pour the waters of the Nile On every golden scale!"  
"How cheerfully he seems to grin, How neatly spread his claws, And welcome little fishes in With gently smiling jaws!"

```
data_src = "file:/home/student/Data/except.txt"
textRDD = sc.textFile(data_src)
textRDD.take(4)
```

Input

```
['How doth the little crocodile',
 'Improve his shining tail, ',
 'And pour the waters of the Nile',
 'On every golden scale!']
```

Output

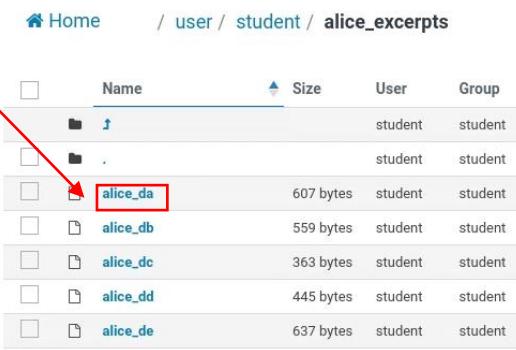
## Đọc toàn bộ tệp văn bản

- Đôi khi chúng tôi không muốn dòng mới (\n) phân định từng phần tử của RDD
  - ▶ Các bản ghi ở định dạng JSON hoặc XML trải rộng trên nhiều dòng
- Sử dụng phương thức **wholeTextFiles** của SparkContext với thư mục nguồn
  - ▶ Mỗi tệp trong thư mục nguồn trở thành một phần tử trong RDD
  - ▶ Đọc dữ liệu dưới dạng bộ 2 mục - (Vị trí tệp, Nội dung tệp)

```
Data_src = "alice_excerpts"
wholeRDD = sc.wholeTextFiles(data_src)
wholeRDD.take(1)
```

```
[("hdfs://localhost:9000/user/student/alice_excerpts/alice_da",
 "Mary Ann! Mary Ann!" said the voice. "Fetch me my gloves th
is moment!"\nThen came a little patterning of feet on the stair
s. Alice knew it was\nthe Rabbit coming to look for her, and sh
e trembled till she shook the\nhouse, quite forgetting that she
was now about a thousand times as\nlarge as the Rabbit, and had
no reason to be afraid of it.\n\nPresently the Rabbit came up t
o the door, and tried to open it; but, as\nthe door opened inwa
rds, and Alice's elbow was pressed hard against it,\nthat attem
pt proved a failure. Alice heard it say to itself "Then I'll\nng
o round and get in at the window."\n')]
```

Input



|                          | Name     | Size      | User    | Group   |
|--------------------------|----------|-----------|---------|---------|
| ...                      | ..       |           | student | student |
| ...                      | .        |           | student | student |
| <input type="checkbox"/> | alice_da | 607 bytes | student | student |
| <input type="checkbox"/> | alice_db | 559 bytes | student | student |
| <input type="checkbox"/> | alice_dc | 363 bytes | student | student |
| <input type="checkbox"/> | alice_dd | 445 bytes | student | student |
| <input type="checkbox"/> | alice_de | 637 bytes | student | student |

## Đọc các định dạng tệp khác

- Apache Spark có thể đọc bất kỳ định dạng tệp nào được Hadoop hỗ trợ
  - ▶ Trong nội bộ, Spark sử dụng thư viện tệp Hadoop JAR để đọc nguồn dữ liệu
  - ▶ Sử dụng hadoopFile của SparkContext (Hadoop 1.x) hoặc newAPIhadoopFile (Hadoop 2.x)
- Trên thực tế, textFile của SparkContext là một lối tắt
  - ▶ Trong nội bộ, Spark gọi newAPIhadoopFile

```
data_src = "alice.txt"
apiFile = sc. \
 newAPIHadoopFile(
 data_src,
 "org.apache.hadoop.mapreduce.lib.input.TextInputFormat", # inputFormatClass
 "org.apache.hadoop.io.Text", # keyClass
 "org.apache.hadoop.io.LongWritable", # valueClass
)
print("Alice from API input format: ", apiFile.count())
```

Input

Alice from API input format: 3761

Output

# Hoạt động RDD

- RDD hỗ trợ hai loại hoạt động
  - ▶ Chuyển đổi tạo tập dữ liệu mới từ tập dữ liệu hiện có
  - ▶ Hành động trả về một giá trị cho chương trình Driver sau khi chạy tính toán trên tập dữ liệu
- Ví dụ phép toán:
  - ▶ **map** là một phép biến đổi chuyển từng thành phần tập dữ liệu qua một hàm và trả về một RDD mới biểu thị kết quả
  - ▶ **reduce** là một hành động tổng hợp tất cả các thành phần của RDD bằng một số chức năng và trả về kết quả cuối cùng cho chương trình điều khiển
- Chuyển đổi lười biếng
  - ▶ Tất cả chuyển đổi Spark là chuyển đổi lười biếng
  - ▶ Chúng không được thực thi cho đến khi một hành động được gọi yêu cầu giá trị trả về
- Biến đổi bất biến
  - ▶ Các phép biến đổi tia lửa là bất biến - phép biến đổi tạo ra một RDD mới

# Hành động RDD

## I Các hành động không yêu cầu hàm làm tham số

| Hành động            | Ý nghĩa                                                                                                                                                                                                                                                                                                           |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| collect()            | Trả về tất cả các phần tử của tập dữ liệu dưới dạng một mảng tại chương trình điều khiển. Điều này thường hữu ích sau khi bộ lọc hoặc hoạt động khác trả về một tập hợp con nhỏ của dữ liệu                                                                                                                       |
| count()              | Trả về số lượng phần tử trong tập dữ liệu                                                                                                                                                                                                                                                                         |
| first()              | Trả về phần tử đầu tiên của tập dữ liệu (tương tự như take(1))                                                                                                                                                                                                                                                    |
| take(n)              | Trả về một mảng có n phần tử đầu tiên của tập dữ liệu                                                                                                                                                                                                                                                             |
| saveAsTextFile(path) | Viết các thành phần của tập dữ liệu dưới dạng tệp văn bản (hoặc tập hợp các tệp văn bản) trong một thư mục nhất định trong hệ thống tệp cục bộ, HDFS hoặc bất kỳ hệ thống tệp nào khác được Hadoop hỗ trợ. Spark sẽ gọi <code>toString</code> trên từng phần tử để chuyển đổi nó thành một dòng văn bản trong tệp |

## Hành động xuất dữ liệu RDD

- | count() và first() đều trả về giá trị
- | take(n) và collect() đều trả về bộ sưu tập
- | Nhớ lại rằng các hành động trả lại giá trị cho chương trình Driver từ tất cả các Executor
  - ▶ KHÔNG sử dụng collect() trong mã sản xuất
  - ▶ Nó rất có thể sẽ gây ra lỗi "hết bộ nhớ" trong trình điều khiển

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6])
print("This RDD has", myRDD.count(), "elements")
print("The first element is", myRDD.first())
print("The first 2 elements are", myRDD.take(2))
print("The entire collection is", myRDD.collect())
```

Input

```
This RDD has 6 elements
The first element is 1
The first 2 elements are [1, 2]
The entire collection is [1, 2, 3, 4, 5, 6]
```

Output

## Lưu RDD dưới dạng tệp văn bản

- Sử dụng **saveAsTextFile(path)** để lưu RDD dưới dạng tệp văn bản trên đường dẫn được cung cấp dưới dạng tham số
  - ▶ đường dẫn là một thư mục mà Spark sẽ tạo
  - ▶ nếu đường dẫn đã tồn tại, Spark sẽ đưa ra ERROR
  - ▶ Nếu đường dẫn tương đối được sử dụng, Spark sẽ mặc định là thư mục chính HDFS của người dùng
- Lưu ý rằng có một số tệp được lưu trong thư mục.
  - ▶ Mỗi phân vùng sẽ được lưu dưới dạng một tệp

|  | Name       | Size    | User    | Group   |
|--|------------|---------|---------|---------|
|  | ..         |         | student | student |
|  | .          |         | student | student |
|  | part-00005 | 2 bytes | student | student |
|  | part-00004 | 2 bytes | student | student |
|  | part-00003 | 2 bytes | student | student |
|  | part-00002 | 2 bytes | student | student |
|  | part-00001 | 2 bytes | student | student |
|  | part-00000 | 2 bytes | student | student |
|  | _SUCCESS   | 0 bytes | student | student |

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6])
myRDD.saveAsTextFile("myFirstRDD")
```

Input

# Hành động của RDD

## Các hành động yêu cầu một hàm hoàn thành

| Hành động              | Ý nghĩa                                                                                                                                                                                   |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| reduce(func)           | Tổng hợp các thành phần của tập dữ liệu bằng hàm func (nhận hai đối số và trả về một đối số). Hàm phải có tính chất giao hoán và kết hợp để có thể tính toán song song một cách chính xác |
| foreach(func)          | Chạy một hàm func trên từng phần tử của tập dữ liệu. Điều này thường được thực hiện đối với các tác dụng phụ như cập nhật Bộ tích lũy hoặc tương tác với hệ thống lưu trữ bên ngoài       |
| foreachPartition(func) | Chạy một hàm func trên mỗi phân vùng của tập dữ liệu                                                                                                                                      |
| getNumPartitions()     | Trả về số phân vùng                                                                                                                                                                       |

## Hành động reduce RDD

- | Hành động **reduce(func)** trả về kết quả của việc áp dụng func cho các thành phần của RDD
- | func là hàm nhận hai đối số
- | func phải được kết hợp
  - ▶ x toán tử y bằng với y toán tử x
- | func phải giao hoán
  - ▶ (x toán tử y) và sau đó toán tử z bằng (y toán tử z) và sau đó là toán tử x

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6])
myRDDsum = myRDD.reduce(lambda l, r: l+r)
print("The sum of my RDD is", myRDDsum)
```

Input

```
The sum of my RDD is 21
```

Output

## Hành động foreach RDD

- Hành động foreach áp dụng một chức năng cho từng thành phần trong RDD
- Tuy nhiên, bản thân hành động trả về Không như có thể thấy trong đầu ra
- Chức năng được cung cấp thường được sử dụng để tác động đến một số tác dụng phụ, chẳng hạn như in đẹp từng thành phần như trong ví dụ

Input

```
def wow_print(e):
 print("-"*(len(e)+4))
 print("*", e, "*")
 print("-"*(len(e)+4))

my_fruits = ["apples", "oranges", "pear",
 "banana"]
dataRDD = sc.parallelize(my_fruits)
print("The output of foreach is: ",
 dataRDD.foreach(wow_print))
```

Output

```
The output of foreach is : None

* apples *

* oranges *

* pear *

* banana *

```

## Hành động phân vùng RDD

- Những hành động này áp dụng ở cấp độ phân vùng
- Hàm được truyền cho **foreachPartition** dự kiến sẽ có một tham số iterator
  - Có thể sử dụng iterator trong vòng lặp for để truy cập từng phần tử trong phân vùng

```
def show_part(it):
 for i in it:
 print(i)
 print("*****")
partRDD = sc.parallelize([1, 2, 3, 4, 5, 6], 2)
print("This RDD has", partRDD.getNumPartitions(), "partitions")
partRDD.foreachPartition(lambda iterator:
 show_part(iterator))
```

Input

Output

```
This RDD has 2 partitions
1
2
3

4
5
6

```

# Chuyển đổi Spark

- Các phép biến đổi tia lửa có thể được phân loại theo số lượng RDD gốc phụ thuộc
  - ▶ Phụ thuộc hẹp - Chỉ cần một RDD mẹ duy nhất để tạo RDD mới
  - ▶ Phụ thuộc rộng rãi - Cần có một số RDD để tạo RDD mới
- Một danh mục khác là liệu một tham số chức năng có được yêu cầu hay không
  - ▶ Một số phép biến đổi không yêu cầu chức năng riêng biệt và có logic bên trong về cách biến đổi
  - ▶ Một ví dụ cho phép chuyển đổi như vậy sẽ là differ() loại bỏ các phần tử trùng lặp
- Hoạt động ở cấp độ phần tử hoặc cấp độ phân vùng
  - ▶ Một số phép biến đổi ảnh hưởng đến từng phần tử trong khi những phép biến đổi khác hoạt động ở cấp độ phân vùng
- Một số phép biến đổi nhất định hoạt động trên các phép toán tập hợp
  - ▶ Yêu cầu tập dữ liệu
  - ▶ Điều này khác với sự phụ thuộc rộng

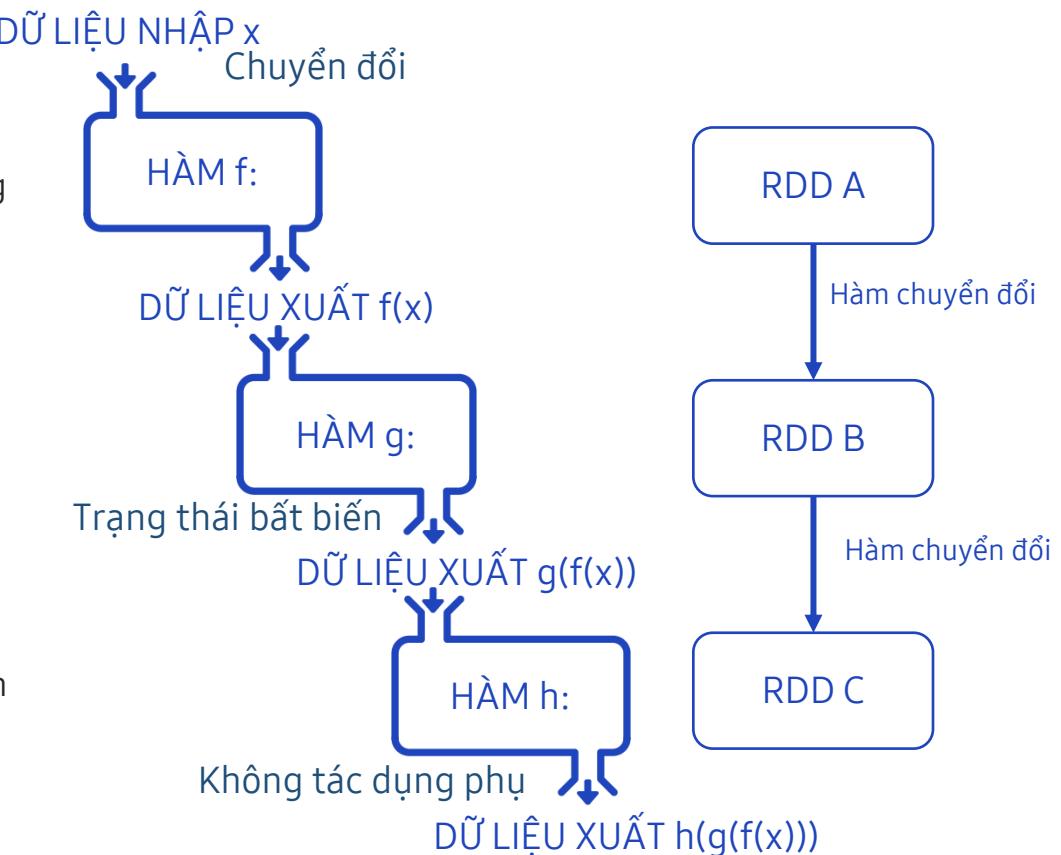
# Chuyển đổi RDD cơ bản

## Chuyển đổi của một tập dữ liệu

| Chuyển đổi    | Ý nghĩa                                                                                                                                              |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| map(func)     | Trả về tập dữ liệu phân tán mới được hình thành bằng cách chuyển từng phần tử của nguồn thông qua hàm func                                           |
| filter(func)  | Trả về tập dữ liệu mới được hình thành bằng cách chọn các phần tử đó của nguồn mà func trả về true.                                                  |
| distinct()    | Trả về tập dữ liệu mới chứa các thành phần riêng biệt của tập dữ liệu nguồn                                                                          |
| flatMap(func) | Tương tự như bản đồ, nhưng mỗi mục đầu vào có thể được ánh xạ tới 0 hoặc nhiều mục đầu ra (vì vậy func phải trả về một Seq thay vì một mục duy nhất) |

## Truyền hàm cho phép biến đổi (1/2)

- Nhớ lại Lập trình hàm từ các bài học Python
  - ▶ Truyền hàm để biến đổi đối tượng
  - ▶ Tất cả các đối tượng là bất biến và chuyển đổi không loại bỏ hoặc sửa đổi đối tượng ban đầu
  - ▶ Điều này ngăn chặn các tác dụng phụ trong đó các chức năng gây ra thay đổi đối với trạng thái và môi trường gọi
- Apache Spark sử dụng Lập trình hàm
  - ▶ Các phương thức chuyển đổi tạo ra các RDD mới không thay đổi trong đó hàm đã được áp dụng cho mọi thành phần của RDD nguồn
  - ▶ Hầu hết các phép biến đổi đều yêu cầu một hàm làm tham số



## Truyền hàm cho phép biến đổi (2/2)

- | Trong hầu hết các trường hợp, các hàm ẩn danh được sử dụng để truyền các hàm dưới dạng tham số cho một phép biến đổi

- ▶ Trong Python - ký hiệu lambda

```
transformation(lambda param 1, param 2, . . . :
 transformation code using parameters)
```

- ▶ Trong Scala - ký hiệu mũi tên

```
transformation(param 1:type , param 2:type , . . . =>
 transformation code using parameters)
```

- | Các chức năng được đặt tên vẫn có thể được sử dụng

```
def myFunction(param)
transformation(myFunction)
```

## Chuỗi chuyển đổi

- Chúng tôi có thể lưu từng chuyển đổi thành một RDD riêng

```
rdd1 = sc.textFile(path)
rdd2 = rdd1.transformation(lambda param 1, : transformation)
rdd3 = rdd2.transformation(lambda param 1, : transformation)
rdd4 = rdd3.transformation(lambda param 1, : transformation)
```

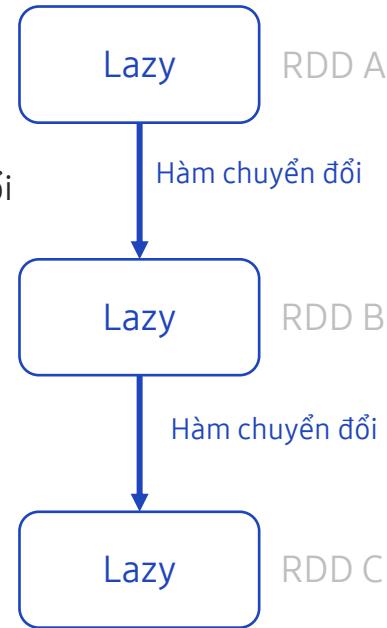
- Tuy nhiên, việc xâu chuỗi các phép chuyển đổi thường thuận tiện hơn nhiều

```
rdd1 = sc.textFile(path) \
 .transformation(lambda param 1, : transformation) \
 .transformation(lambda param 1, : transformation) \
 .transformation(lambda param 1, : transformation) \
```

## Thi Hành Lazy (1/3)

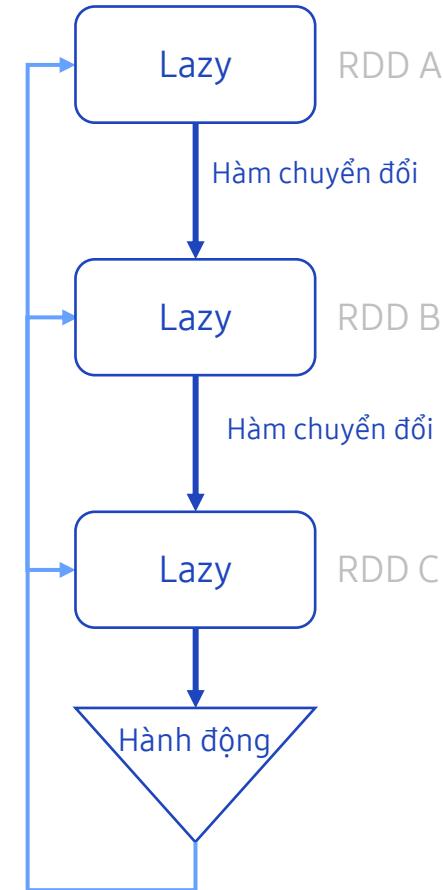
Chuyển đổi Spark là chuyển đổi lười biếng

- Cho đến khi một hành động được gọi, yêu cầu một giá trị, tất cả các phép biến đổi đều bị tạm dừng



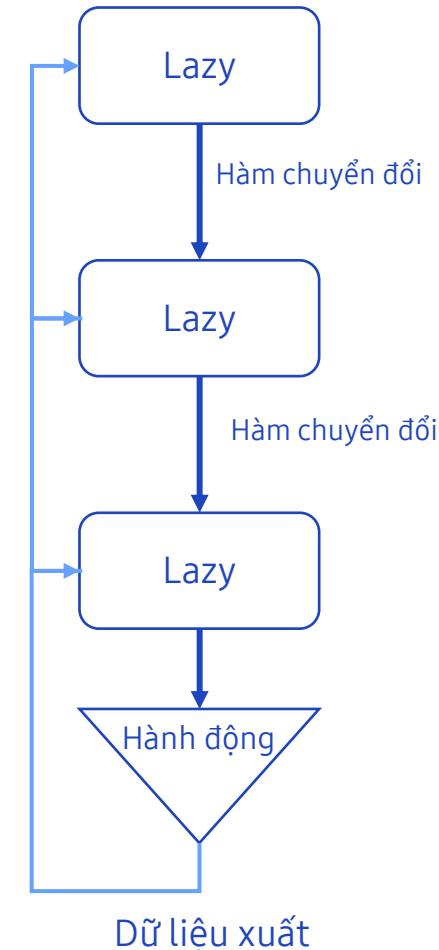
## Thi Hành Lazy (2/3)

- I Chuyển đổi Spark là chuyển đổi lazy
  - ▶ Cho đến khi một hành động được gọi, yêu cầu một giá trị, tất cả các phép biến đổi đều bị tạm dừng
- I Khi một hành động được gọi, Spark sẽ đánh giá sự phụ thuộc của hành động đó
  - ▶ Đây là Đồ thị theo chu kỳ có hướng (DAG) trong đó mỗi nút biểu thị một RDD và mỗi cạnh biểu thị một thao tác chuyển đổi



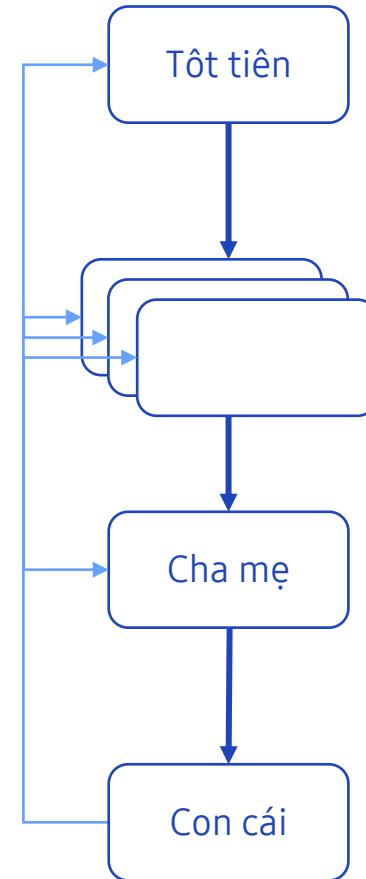
## Thi hành Lazy (3/3)

- Chuyển đổi Spark là chuyển đổi Lazy
  - Cho đến khi một hành động được gọi, yêu cầu một giá trị, tất cả các phép biến đổi đều bị tạm dừng
- Khi một hành động được gọi, Spark sẽ đánh giá sự phụ thuộc của hành động đó
  - Đây là Đồ thị theo chu kỳ có hướng (DAG) trong đó mỗi nút biểu thị một RDD và mỗi cạnh biểu thị một thao tác chuyển đổi
- Khi Spark đạt đến RDD không được đánh giá cao nhất, nó bắt đầu thực hiện các phép biến đổi thực tế
  - Mỗi chuyển đổi sẽ kích hoạt RDD tiếp theo được tạo cho đến khi đầu ra hành động cuối cùng được tạo ra



## Dòng dõi Spark

- Chuyển đổi tạo RDD mới dựa trên một hoặc nhiều RDD hiện có
- Chúng tôi đã mô tả đây là một biểu đồ phụ thuộc trong đó RDD con phụ thuộc vào một hoặc nhiều RDD gốc
  - ▶ Đến lượt RDD gốc lại phụ thuộc vào RDD trong biểu đồ phụ thuộc
  - ▶ Đồ thị phụ thuộc này có thể được mô tả dưới dạng Đồ thị theo chu kỳ có hướng (DAG)
- Mỗi RDD có thể nói là có một dòng dõi với một chuỗi tổ tiên mà nó phụ thuộc vào



## Ví dụ chuyển đổi Map và bộ lọc

- Sử dụng **map** để chuyển đổi tất cả các ký tự thành chữ thường, sau đó lọc các dòng có chứa "alice"

| Input                                                                                                                                                                                                                                                                     | Output                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>data_src = "alice_excerpts" aliceRDD = sc.textFile(data_src) \     .map(lambda line: line.lower()) \     .filter(lambda line: "alice" in line)  for line in aliceRDD.take(5):     print(line)  print("There were", aliceRDD.count(), "lines with Alice in it")</pre> | <p>then came a little patterning of feet<br/>on the stairs. alice knew it was<br/>the door opened inwards, and alice's<br/>elbow was pressed hard against it,<br/>that attempt proved a failure. alice<br/>heard it say to itself "then i'll<br/>"_that_ you won't!" thought alice,<br/>and, after waiting till she fancied<br/>there was a long silence after this,<br/>and alice could only hear whispers<br/>There were 61 lines with Alice in it</p> <p>[Stage 12 : =====&gt;<br/>(48+6)<br/>There were 61 lines with Alice in it</p> |

## Spark toDebugString()

- Spark duy trì thông tin dòng dõi cho mỗi RDD
- Sử dụng phương thức `toDebugString()`, có thể xem từng thông tin về dòng RDD

**Input**

```
data_src = "alice_excerpts"
aliceRDD = sc.textFile(data_src) \
 .map(lambda line: line.lower()) \
 .filter(lambda line: "alice" in line)
print(aliceRDD.toDebugString().decode("utf-8"))
```

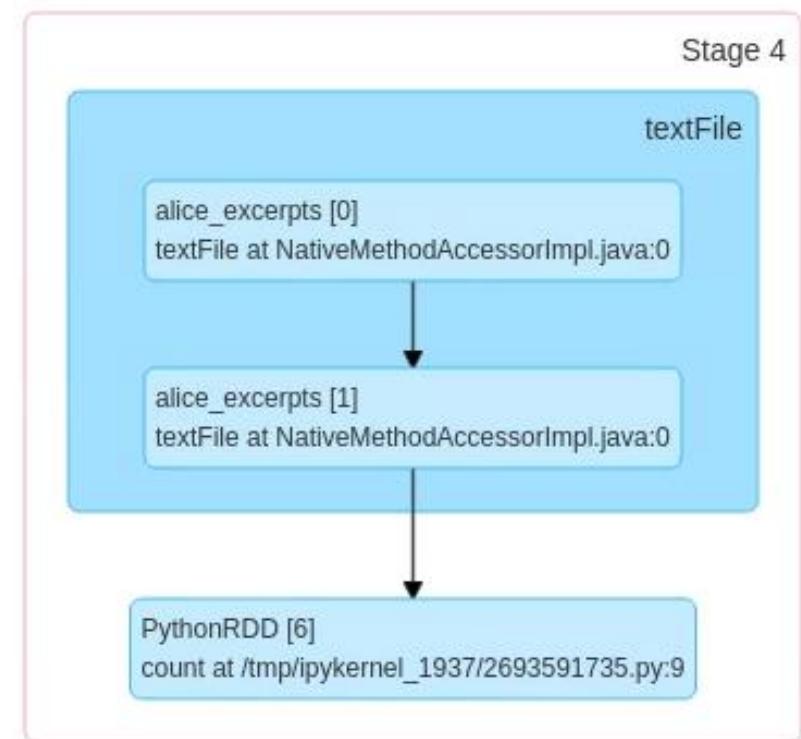
**Output**

```
(65) PythonRDD[10] at RDD at PythonRDD.scala:53 []
| alice_excerpts MapPartitionsRDD[9] at textFile at NativeMethodAccessorImpl.java:0 []
| alice_excerpts HadoopRDD[8] at textFile at NativeMethodAccessorImpl.java:0 []
```

## Xem DAG trên Spark Web UI

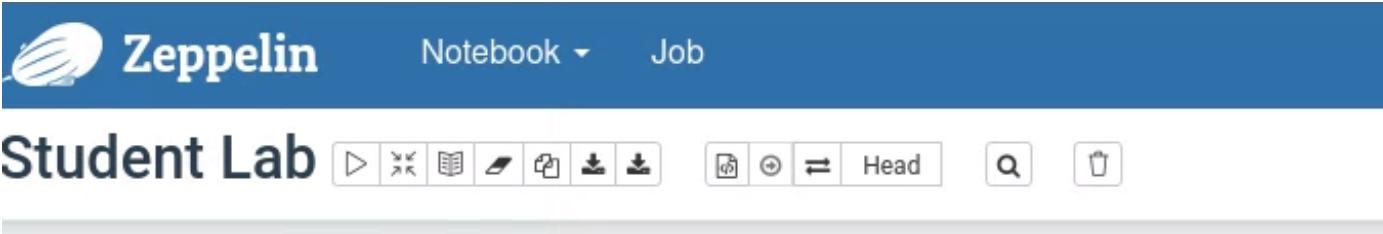
- Apache Spark có giao diện người dùng Spark Web dựa trên trình duyệt
- Khi chạy ở local mode thì có thể xem trên localhost tại port 4040
  - ▶ Cổng có thể được cấu hình thành một số khác
- Mỗi SparkContext tạo giao diện Spark Web UI của riêng nó
  - ▶ Nếu tồn tại nhiều sparkcontext, cổng sẽ tăng thêm 1 khi mỗi phiên bản được tạo
  - ▶ Từ 4040 đến 4041 chẳng hạn
- Khi chạy ở chế độ cụm, Spark Web UI có thể được truy cập từ YARN Web UI

### DAG Visualization



## Spark toDebugString - Scala

- Scala's toDebugString có nhiều thông tin hơn



```
val data_src = "alice_excerpts"
val aliceRDD = sc.
 textFile(data_src).
 map(line => line.toLowerCase).
 filter(line => line contains "alice")
println(aliceRDD.toDebugString)
```

(65) MapPartitionsRDD[7] at filter at <console>:32 []  
| MapPartitionsRDD[6] at map at <console>:31 []  
| alice\_excerpts MapPartitionsRDD[5] at textFile at <console>:29 []  
| alice\_excerpts HadoopRDD[4] at textFile at <console>:29 []  
data\_src: String = alice\_excerpts  
aliceRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at filter at <console>:32

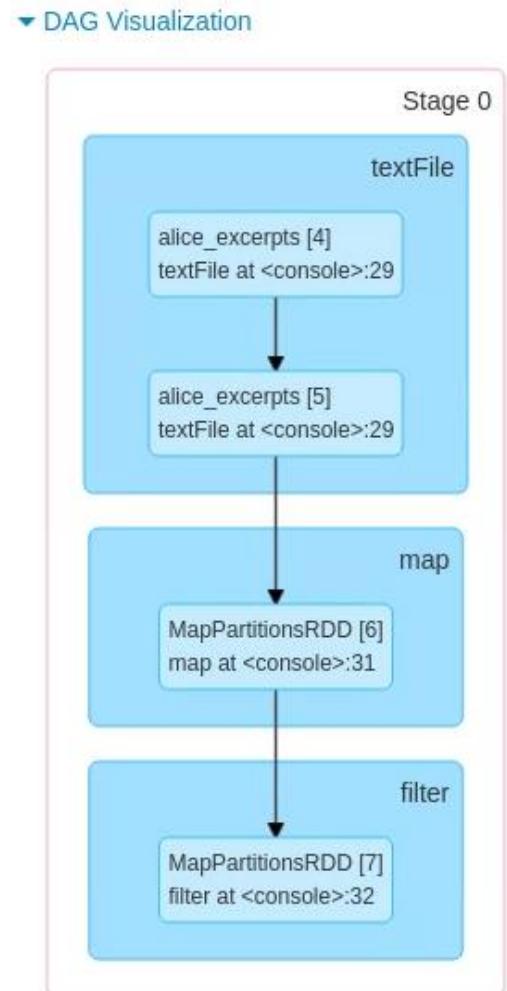
## Spark Web UI - Scala

- Mỗi công việc đã hoàn thành được hiển thị và có thể được chọn
- Trong phần mô tả, bạn sẽ thấy tên của "Hành động" đã kích hoạt công việc để thực thi
  - Trong trường hợp của chúng ta, nó đã được tính

### ▼ Completed Jobs (1)

Page: 1

| Job Id (Job Group) ▾                                                   | Description                                                          |
|------------------------------------------------------------------------|----------------------------------------------------------------------|
| 0<br>(zeppelin anonymous 2GESRDZRZ paragraph_1629880937329_1360145546) | Started by: anonymous<br><a href="#">count at &lt;console&gt;:26</a> |



## Chuyển đổi FlatMap (1/3)

- Hàm chuyển đổi trong **FlatMap** sẽ trả về một tập hợp hoặc chuỗi
- Tuy nhiên, tương tự như map, **FlatMap** lấy từng mục trong bộ sưu tập hoặc chuỗi và xuất ra dưới dạng một phần tử riêng biệt
- Nhìn vào mã:
  - ▶ dataRDD là RDD một phần tử bao gồm chuỗi trái cây
  - ▶ Trong cả hai trường hợp, phương thức split lấy chuỗi và trả về tập hợp các loại trái cây
  - ▶ **map** trả về bộ sưu tập đó
  - ▶ **FlatMap** lấy từng mục trong bộ sưu tập và xuất nó dưới dạng một phần tử riêng biệt

## Chuyển đổi FlatMap (2/3)

```
my_fruits = ["apples oranges pear banana"]
dataRDD = sc.parallelize(my_fruits)
print("Number of elements is this RDD:", dataRDD.count())
print("It contains the following string:", dataRDD.collect())
```

**Input**

```
Number of elements is this RDD: 1
It contains the following string: ['apples oranges pear banana']
```

**Output**

```
mapRDD = dataRDD.map(lambda line: line.split(' '))
for line in mapRDD.collect():
 print(line)
```

**Input**

```
['apples', 'oranges', 'pear', 'banana']
```

**Output**

## Chuyển đổi FlatMap (3/3)

**Input**

```
flatRDD = dataRDD.flatMap(lambda line: line.split(' '))
for line in flatRDD.collect():
 print(line)
```

**Output**

apples  
oranges  
pear  
banana

## Chuyển đổi Distinct()

- Cho đến nay, chúng ta đã thấy rằng tất cả các phép biến đổi đều yêu cầu một hàm được truyền dưới dạng tham số
  - Hàm xác định cách chuyển đổi sẽ diễn ra
- Tuy nhiên, một số phương thức chuyển đổi không yêu cầu phải truyền một chức năng riêng biệt.
  - `distinct()` xóa mọi phần tử trùng lặp và không yêu cầu hướng dẫn riêng về cách thực hiện việc này

```
Input
my_fruits = ["apples oranges pear", "banana apples"]
dataRDD = sc.parallelize(my_fruits)
print("Number of elements is this RDD:", dataRDD.count())
print("It contains the following string:", dataRDD.collect())
```

| Output                                                                                                                      |
|-----------------------------------------------------------------------------------------------------------------------------|
| Number of elements is<br>this RDD: 2<br>It contains the<br>following string:<br>['apples oranges pear',<br>'banana apples'] |

```
Input
flatRDD = dataRDD \
.flatMap(lambda line: line.split(' ')).distinct()
for line in flatRDD.collect():
 print(line)
```

| Output                              |
|-------------------------------------|
| oranges<br>pear<br>banana<br>apples |

# Đặt toán tử chuyển đổi RDD

## Chuyển đổi nhiều bộ dữ liệu

| Chuyển đổi                 | Ý nghĩa                                                                                                   |
|----------------------------|-----------------------------------------------------------------------------------------------------------|
| union(otherDataset)        | Trả về tập dữ liệu mới chứa sự kết hợp của các phần tử trong tập dữ liệu nguồn và đối số                  |
| intersection(otherDataset) | Trả về một RDD mới chứa giao điểm của các phần tử trong tập dữ liệu nguồn và đối số                       |
| cartesian(otherDataset)    | Khi được gọi trên tập dữ liệu loại T và U, trả về tập dữ liệu gồm (T, U) cặp (tất cả các cặp phần tử)     |
| subtract(otherDataset)     | Trả về tập dữ liệu mới chứa tất cả các thành phần trong tập dữ liệu nguồn không có trong tập dữ liệu khác |

## Đặt phép biến đổi toán tử (1/2)

- Các phép biến đổi này lấy hai bộ dữ liệu và thực hiện các thao tác thiết lập trên chúng

```
fruits1 = ["apples", "oranges", "pear"]
fruits2 = ["banana", "apples"]
fruit1RDD = sc.parallelize(fruits1)
fruit2RDD = sc.parallelize(fruits2)

fruit1RDD.union(fruit2RDD).collect()
```

Input

```
['apples', 'oranges', 'pear', 'banana', 'apples']
```

Output

```
fruit1RDD.intersection(fruit2RDD).collect()
```

Input

```
['apples']
```

Output

## Đặt phép biến đổi toán tử (2/2)

- Các phép biến đổi này lấy hai bộ dữ liệu và thực hiện các thao tác thiết lập trên chúng

```
fruit1RDD.subtract(fruit2RDD).collect()
```

Input

```
['oranges', 'pear']
```

Output

```
fruit1RDD.cartesian(fruit2RDD).collect()
```

Input

```
[('apples', 'banana'),
 ('apples', 'apples'),
 ('oranges', 'banana'),
 ('oranges', 'apples'),
 ('pear', 'banana'),
 ('pear', 'apples')]
```

Output

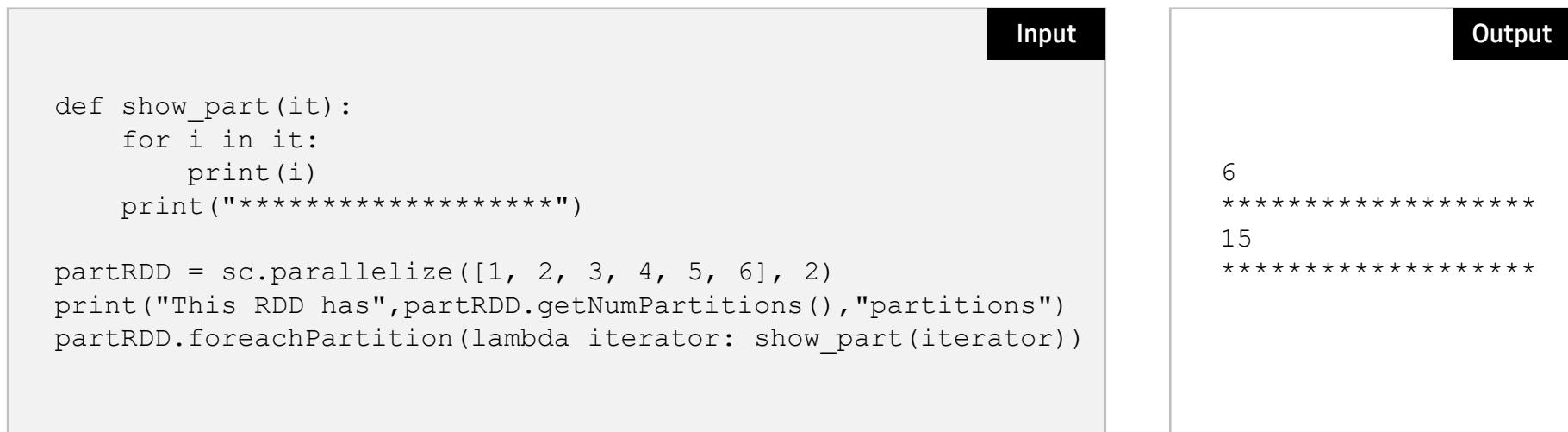
# Chuyển đổi RDD dựa trên phân vùng

- Các phép biến đổi hoạt động ở cấp độ phân vùng, thay vì cấp độ hàng

| Chuyển đổi                    | Ý nghĩa                                                                                                                                                                                                                                |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mapPartitions(func)           | Tương tự như bản đồ, nhưng chạy riêng trên từng phân vùng (khối) của RDD, vì vậy func phải thuộc loại <code>Iterator&lt;T&gt; =&gt; Iterator&lt;U&gt;</code> khi chạy trên RDD thuộc loại T                                            |
| mapPartitionsWithIndex (func) | Tương tự như mapPartitions, nhưng cũng cung cấp cho func một giá trị số nguyên biểu thị chỉ số của phân vùng, vì vậy func phải thuộc loại <code>(Int, Iterator&lt;T&gt;) =&gt; Iterator&lt;U&gt;</code> khi chạy trên RDD thuộc loại T |

## Chuyển đổi phân vùng RDD

- Pháp biến đổi **mapPartition(func)** hoạt động ở cấp độ phân vùng
- Là một phần của giao thức API, **mapPartition** cung cấp một trình lặp để func sử dụng
- func dự kiến sẽ có một tham số iterator có thể được sử dụng để điều hướng các thành phần của phân vùng
- func dự kiến sẽ trả về một trình vòng lặp của map Partition kết quả
  - ▶ Năng suất Python được sử dụng ở đây để cung cấp chức năng này



# Phân vùng RDD với chuyển đổi chỉ mục

Phép biến đổi **mapPartitionsWithIndex(func)** hoạt động chính xác như phép biến đổi **mapPartitions(func)** nhưng có thêm chức năng theo dõi số chỉ mục phân vùng

- ▶ Số chỉ mục phân vùng bắt đầu từ 0

Việc chuyển đổi sẽ cung cấp cả giá trị chỉ mục phân vùng và trình vòng lặp để điều hướng qua các thành phần của phân vùng

func dự kiến sẽ trả về một trình vòng lặp như trường hợp của **mapPartitions**

- ▶ Câu lệnh `yield` cung cấp chức năng này

Input

```
def show_part(iterator):
 for i in iterator:
 print(i)
 print("*****")
def add_I(index, it):
 yield (index, sum(it))

partRDD = sc.parallelize([1, 2, 3, 4, 5, 6], 2)
partRDD \
 .mapPartitionsWithIndex(lambda ix, it: add_I(ix, it)) \
 .foreachPartition(lambda iterator: show_part(iterator))
```

Output

```
(0, 6)

(1, 15)

```

# Phân phối lại dữ liệu Chuyển đổi RDD

- Các phép biến đổi làm thay đổi số lượng phân vùng

| Chuyển đổi                                          | Ý nghĩa                                                                                                                                                                                                                                                                      |
|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| coalesce(numPartitions)                             | Giảm số lượng phân vùng trong RDD xuống numPartitions. Hữu ích để chạy các hoạt động hiệu quả hơn sau khi lọc một tập dữ liệu lớn.                                                                                                                                           |
| repartition(numPartitions)                          | Xáo trộn lại dữ liệu trong RDD một cách ngẫu nhiên để tạo nhiều hoặc ít phân vùng hơn và cân bằng dữ liệu trên chúng. Điều này luôn xáo trộn tất cả dữ liệu qua mạng.                                                                                                        |
| repartitionAndSortWithinPartitions<br>(partitioner) | Phân vùng lại RDD theo trình phân vùng đã cho và trong mỗi phân vùng kết quả, hãy sắp xếp các bản ghi theo khóa của chúng. Điều này hiệu quả hơn so với việc gọi phân vùng lại và sau đó sắp xếp trong từng phân vùng vì nó có thể đẩy quá trình sắp xếp xuống máy xáo trộn. |

# Chuyển đổi Coalesce

- Chuyển đổi **coalesce** thay đổi số lượng phân vùng mà không xáo trộn dữ liệu
- Xáo trộn dữ liệu là một hoạt động rất tốn kém
- Thường được sử dụng để giảm số lượng phân vùng sau khi một tập dữ liệu lớn đã được cắt bớt

Home / user / student / coalesceRDD

|  | Name       | Size     | User    |
|--|------------|----------|---------|
|  |            |          | student |
|  |            |          | student |
|  | _SUCCESS   | 0 bytes  | student |
|  | part-00000 | 12 bytes | student |

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6], 3)
print("Original # of partitions:", myRDD.getNumPartitions())
coalesceRDD = myRDD.coalesce(1)
print("New # of partitions:", coalesceRDD.getNumPartitions())
coalesceRDD.saveAsTextFile("coalesceRDD")
```

Input

```
Original # of partitions: 3
New # of partitions: 1
```

Output

## Chuyển đổi RDD phân vùng lại (1/2)

- Giống như chuyển đổi **coalesce**, **repartition** lại có thể được sử dụng để thay đổi số lượng phân vùng
  - ▶ Không giống như, **coalesce**, **repartition** lại xáo trộn dữ liệu
  - ▶ Điều này có thể hữu ích để khắc phục sự cố nghiêng
  - ▶ Xuyên xảy ra khi một hoặc một số phân vùng chứa phần lớn tập dữ liệu
  - ▶ Trong một hệ thống song song phân tán, thời gian hoàn thành được quyết định bởi phân vùng chậm nhất
  - ▶ Do đó, điều quan trọng là phải cẩn gắt làm cho khối lượng công việc của mỗi phân vùng được phân bổ đều
- Xáo trộn là một hoạt động tốn kém và cần cẩn thận khi sử dụng

## Chuyển đổi RDD phân vùng lại (2/2)

```
myRDD = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9], 3)
print("Original RDD:", myRDD.getNumPartitions())
myRDD.foreachPartition(lambda iterator: show_part(iterator))

shuffledRDD = myRDD.repartition(3)
print("Reshuffled RDD:", shuffledRDD.getNumPartitions())
shuffledRDD.foreachPartition(lambda iterator: show_part(iterator))
```

**Input**

Original RDD: 3  
1  
2  
3  
\*\*\*\*\*  
4  
5  
6  
\*\*\*\*\*  
7  
8  
9  
\*\*\*\*\*

**Output**

Reshuffled RDD: 3  
4  
5  
6  
\*\*\*\*\*  
1  
2  
3  
7  
8  
9  
\*\*\*\*\*

**Output**

# Chuyển đổi RDD hỗn hợp

## Chuyển đổi hỗn hợp

| Chuyển đổi                              | Ý nghĩa                                                                                                                                                                                                                       |
|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sample(withReplacement, fraction, seed) | Lấy mẫu phân số có Thay thế dữ liệu, có hoặc không thay thế, bằng cách sử dụng hạt giống trình tạo số ngẫu nhiên nhất định.                                                                                                   |
| pipe(command, [envVars])                | Đưa từng phân vùng của RDD thông qua lệnh shell, ví dụ: một tập lệnh Perl hoặc bash. Các phần tử RDD được ghi vào stdin của quy trình và các dòng xuất ra thiết bị xuất chuẩn của nó được trả về dưới dạng RDD của các chuỗi. |

## Chuyển đổi lấy mẫu

- Sử dụng **sample(withReplacement, fraction, seed)** để tạo RDD mới với dữ liệu được lấy mẫu
  - ▶ Sử dụng phân số để kiểm soát phần trăm dữ liệu được lấy mẫu
  - ▶ Sử dụng cùng một hạt giống để đảm bảo bạn nhận được cùng một dữ liệu được lấy mẫu khi lặp lại các thử nghiệm
- **withReplacement** kiểm soát xem bạn có thể lặp lại các mẫu hay không
  - ▶ Đặt thành True để cho phép các giá trị lặp lại

```
myRDD = sc.parallelize(range(100))
print(myRDD.sample(False, 0.3, 123).collect())
```

Input

```
[1, 4, 5, 7, 9, 13, 14, 17, 22, 24, 26, 38, 43, 46, 54,
55, 56, 59, 60, 63, 67, 68, 78, 83, 93, 96, 99]
```

Output

```
myRDD = sc.parallelize(range(100))
print(myRDD.sample(True, 0.3, 123).collect())
```

Input

```
[0, 11, 13, 14, 20, 22, 23, 27, 27, 28, 29, 34, 34, 35,
37, 43, 51, 60, 62, 64, 70, 70, 71, 77, 77, 87, 89]
```

Output

## Chuyển đổi đường ống RDD (1/2)

- Chuyển đổi đường ống cho phép các nhà phát triển thực thi các tệp tập lệnh từ trình bao qua tập dữ liệu RDD
  - ▶ Quá trình stdin được sử dụng để nhận tập dữ liệu RDD
  - ▶ Quá trình stdout được sử dụng để gửi lại kết quả
- Cho phép các lập trình viên sử dụng các ngôn ngữ khác ngoài Scala, Python hoặc Java để thực hiện chuyển đổi

## Chuyển đổi đường ống RDD (2/2)

```
[student@localhost Data] $ cat repeat.sh
#!/bin/bash
While read LINE; do
 echo ${LINE}
done
```

Cmd

```
data_src = ["Having", "fun", "learning", "Spark"]
script_path = "/home/student/Data/repeat.sh"
myRDD = sc.parallelize(data_src)
pipeRDD = myRDD.pipe(script_path)
print(pipeRDD.collect())
```

Input

```
['Having', 'fun', 'learning', 'Spark']
```

Output

Bài 1.

# Xử lý dữ liệu phi cấu trúc

- | 1.1. Giới thiệu về Apache Spark
- | 1.2. Khái niệm cơ bản về Python
- | 1.3. Chuyển đổi dữ liệu với Core API
- | 1.4. Làm việc với Pair RDD**

## Cặp RDD là gì

- Spark định nghĩa lớp PairRDDFunctions chuyên hoạt động với Pair RDD
- Một cặp RDD là một RDD trong đó tất cả các phần tử nằm trong một tuple cặp khóa-giá trị
  - ▶ Tuples được tạo với dấu ngoặc đơn trong Python
  - ▶ Tuple cặp khóa-giá trị là tuple có 2 phần tử
  - ▶ Bên trái là Key và bên phải là Value
  - ▶ Các phần tử của một tuple có thể là bất kỳ kiểu dữ liệu nguyên thủy hoặc phức tạp nào
  - ▶ Tuples là loại bất biến
- Cặp khóa-giá trị là một loại dữ liệu rất phổ biến
  - ▶ Loại dữ liệu được sử dụng trong thuật toán Map Reduce
  - ▶ Đơn vị dữ liệu của nhiều cơ sở dữ liệu NoSQL

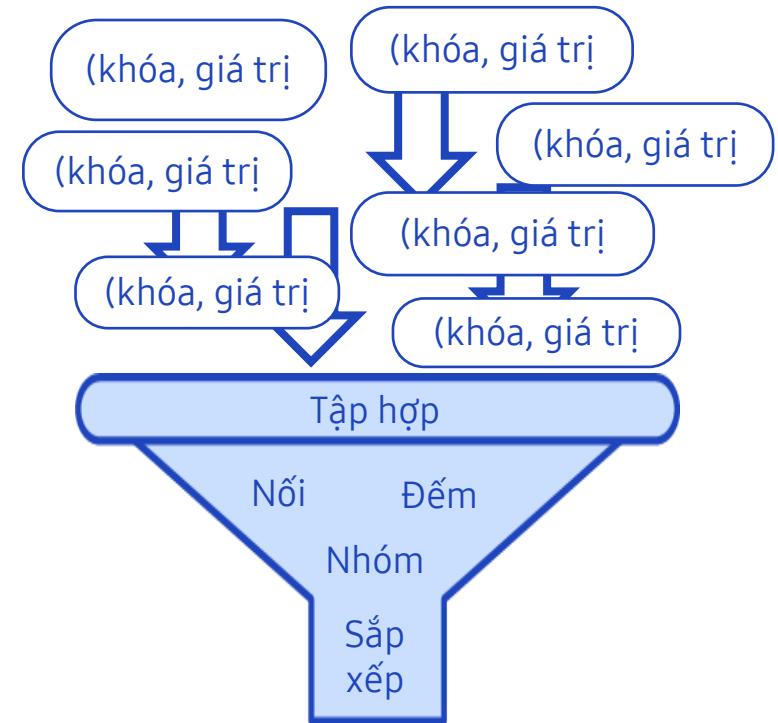
```
key = "Name"
value = "Jonathan"
key_pair_tuple = (key, value)
```

Input

# Có thể làm gì với Pair RDD?

## I Cấu trúc dữ liệu Khóa-Giá trị hiển thị nhiều hoạt động

- ▶ Đếm số phím
- ▶ Tổng hợp giá trị của các mục có cùng khóa
- ▶ Sắp xếp và nhóm theo khóa
- ▶ Nhóm và nối hai RDD dựa trên cùng một khóa



## Tạo Pair RDD

| Cũng như các hoạt động RDD khác, bước đầu tiên bắt buộc là tạo Pair RDD

- ▶ `map()`, `flatMap()`, `keyBy()`, `flatMapValues()` là các phép biến đổi hữu ích để tạo Pair RDD
- ▶ Chúng ta đã quen thuộc với `map()` và `flatMap()`

Input

```
data_src = "alice_excerpts"
pairRDD = sc.textFile(data_src) \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1))
for i in pairRDD.take(5):
 print(i)
```

Output

```
('Mary', 1)
('Ann!', 1)
('Mary', 1)
('Ann!!', 1)
('said', 1)
```

## Tạo Pair RDD với keyBy(func)

- keyBy(func) mong đợi một tham số hàm func mô tả cách tìm khóa
- Giá trị sẽ là nội dung ban đầu của RDD gốc

**Input**

```
my_fruits = ["apples", "oranges", "pear", "banana"]
pairRDD = sc.parallelize(my_fruits) \
 .keyBy(lambda fruit: len(fruit))

for i in pairRDD.collect():
 print(i)
```

**Output**

```
(6, 'apples')
(7, 'oranges')
(4, 'pear')
(6, 'banana')
```

## Sử dụng map() thay vì keyBy()

- Một trong những vấn đề với **keyBy()** là quá trình chuyển đổi không minh bạch
  - Các nhà phát triển phải hiểu cách thức hoạt động của giao thức **keyBy()**
  - Mặt khác, việc triển khai sử dụng **map()** này rất minh bạch và được các nhà phát triển ưa thích

Input

```
my_fruits = ["apples", "oranges", "pear", "banana"]
pairRDD = sc.parallelize(my_fruits) \
 .map(lambda fruit: (len(fruit), fruit))

for i in pairRDD.collect():
 print(i)
```

Output

```
(6, 'apples')
(7, 'oranges')
(4, 'pear')
(6, 'banana')
```

## Sử dụng FlatMapValues(func)

| Cách thức hoạt động của phép biến đổi này có thể được giải mã từ tên của phương thức

- ▶ Có một flatMap, vì vậy chúng tôi mong đợi func tạo ra một bộ sưu tập có thể được "phẳng hóa"
- ▶ Giá trị cho biết, phần tử đầu vào dự kiến sẽ ở định dạng (khóa, giá trị) và chức năng sẽ được áp dụng cho phần giá trị của bộ dữ liệu cặp
- ▶ Giá trị ban đầu đã được "làm phẳng" thành một số phần tử.
- ▶ Sao chép khóa gốc cho từng thành phần giá trị "làm phẳng"

Input

```
fav_fruits = [("Henry", "apples grapes banana"),
 ("Shaun", "watermelon strawberry"),
 ("Sharon", "pear apples kiwi")]

pairRDD = sc.parallelize(fav_fruits) \
 .flatMapValues(lambda fruits: fruits.split(" "))
for i in pairRDD.collect():
 print(i)
```

Output

```
('Henry', 'apples')
('Henry', 'grapes')
('Henry', 'banana')
('Shaun', 'watermelon')
('Shaun', 'strawberry')
('Sharon', 'pear')
('Sharon', 'apples')
('Sharon', 'kiwi')
```

## Hãy xem đó là chuyển động chậm

- Dữ liệu đầu vào sẽ là một cặp khóa-giá trị

```
("Henry", "apples grapes banana")
```

- Áp dụng func trên phần giá trị của cặp khóa-giá trị

```
("Henry", ['apples', 'grapes', 'banana'])
```

- Làm phẳng bộ sưu tập được tạo bởi func trên giá trị

```
("Henry", 'apples', 'grapes', 'banana')
```

- Nhân đôi khóa

```
("Henry", 'apples')
("Henry", 'grapes')
("Henry", 'banana')
```

# Đánh giá MapReduce

- Map Reduce là mô hình xử lý song song phân tán Hadoop ban đầu
- Dữ liệu được định dạng thành các cặp Khóa-Giá trị và được phân vùng trên nhiều map
- Trong giai đoạn Map, cặp Khóa-Giá trị được chuyển đổi.
  - ▶ Mỗi mapper có thể làm việc trên phần dữ liệu của mình một cách độc lập với những mapper khác
- Sau giai đoạn Map, dữ liệu được sắp xếp và xáo trộn sang giai đoạn tiếp theo Reduce
- Trong giai đoạn Reduce, mỗi bộ giảm thường tổng hợp phần giá trị của dữ liệu
  - ▶ Mỗi reducer có thể hoạt động trên phần dữ liệu của nó một cách độc lập với các reducer khác
- Toàn bộ chu trình Map-Shuffle Sort-Reduce được lặp lại theo trình tự nhiều lần để tạo ra đầu ra mong muốn



# MapReduce trên Spark

- Chúng ta cũng có thể sử dụng mô hình xử lý song song phân tán MapReduce
- Apache Spark cho phép chúng tôi hoạt động hiệu quả hơn nhiều với hiệu suất tăng gấp 100 lần trở lên
- Chúng tôi đã thấy hiệu suất đạt được do xử lý trong bộ nhớ
- Ngoài ra, trong Spark, chúng tôi không còn cần phải tuân theo chu map-shuffle sort-reduce
  - ▶ Bất kỳ sự kết hợp nào giữa bản đồ và giảm chu kỳ
  - ▶ Sắp xếp ngẫu nhiên xảy ra khi chúng tôi chuyển từ map sang chu kỳ reduce
- Chuyển đổi Spark cung cấp khả năng ánh xạ
  - ▶ map(), flatMap(), filter(), v.v.
- Chuyển đổi Spark cung cấp khả năng reducer
  - ▶ groupByKey, sortByKey, reduceByKey, aggregateByKey, join, countByKey, v.v.



# Chuyển đổi Pair RDD

| Chuyển đổi                                   | Ý nghĩa                                                                                                                                                                                                                                                                                                               |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| reduceByKey(func)                            | Khi được gọi trên tập dữ liệu gồm các cặp (K, V), trả về tập dữ liệu gồm các cặp (K, V) trong đó các giá trị cho mỗi khóa được tổng hợp bằng hàm giảm đã cho, hàm này phải thuộc loại (V,V) => V.                                                                                                                     |
| aggregateByKey(zeroValue)<br>(seqOp, combOp) | Khi được gọi trên tập dữ liệu gồm các cặp (K, V), trả về tập dữ liệu gồm 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 "không" trung lập. Cho phép loại giá trị tổng hợp khác với loại giá trị đầu vào, đồng thời tránh phân bổ không cần thiết. |
| groupByKey()                                 | Khi được gọi trên tập dữ liệu gồm (K, V) cặp, trả về tập dữ liệu gồm (K, Iterable<V>) cặp. Nhóm hiệu quả tất cả các giá trị của cùng một khóa                                                                                                                                                                         |
| sortByKey([ascending])                       | Khi được gọi trên tập dữ liệu gồm (K, V) cặp trong đó K triển khai Ordered, trả về tập dữ liệu gồm (K, V) cặp được sắp xếp theo khóa theo thứ tự tăng dần hoặc giảm dần, như được chỉ định trong đối số boolean tăng dần.                                                                                             |
| join(otherDataset)                           | Khi được gọi trên các tập dữ liệu loại (K, V) và (K, W), trả về tập dữ liệu gồm các cặp (K, (V, W)) với tất cả các cặp phần tử cho mỗi khóa. Các liên kết bên ngoài được hỗ trợ thông qua leftOuterJoin, rightOuterJoin và fullOuterJoin.                                                                             |

# Số từ trong Spark (1/4)

## Đọc tập tin nguồn

Input

```
data_src = "alice_clean"
wcRDD = sc.textFile(data_src)

for i in wcRDD.take(5):
 print(i)
```

Output

Mary Ann Mary Ann said the voice Fetch me my gloves this moment  
Then came a little pattering of feet on the stairs Alice knew it was  
the Rabbit coming to look for her and she trembled till she shook the  
house quite forgetting that she was now about a thousand times as  
large as the Rabbit and had no reason to be afraid of it

## Số từ trong Spark (2/4)

- Vector hóa từng từ từ nguồn dữ liệu

Input

```
data_src = "alice_clean"
wcRDD = sc.textFile(data_src) \
 .flatMap(lambda line: line.split(" "))
for i in wcRDD.take(5):
 print(i)
```

Output

```
Mary
Ann
Mary
Ann
said
```

## Số từ trong Spark (3/4)

- Tạo Key-Pair RDD trong đó Key là mỗi từ và giá trị là hằng số 1

**Input**

```
data_src = "alice_clean"
wcRDD = sc.textFile(data_src) \
 .flatMap(lambda line: line.split(" ")) \
 .map(lambda word: (word, 1))

for i in wcRDD.take(5):
 print(i)
```

**Output**

```
('Mary', 1)
('Ann', 1)
('Mary', 1)
('Ann', 1)
('said', 1)
```

## Số từ trong Spark (4/4)

I Tổng hợp tất cả các giá trị có cùng một khóa bằng cách thêm chúng

- ▶ Sử dụng `reduceByKey`

Input

```
data_src = "alice_clean"
wcRDD = sc.textFile(data_src) \
 .flatMap(lambda line: line.split(" ")) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda l, r: l + r)
for i in wcRDD.take(5):
 print(i)
```

Output

```
('said', 63)
(' ', 324)
('before', 3)
('The', 14)
('two', 7)
```

## Chuyển đổi AggregateByKey (1/4)

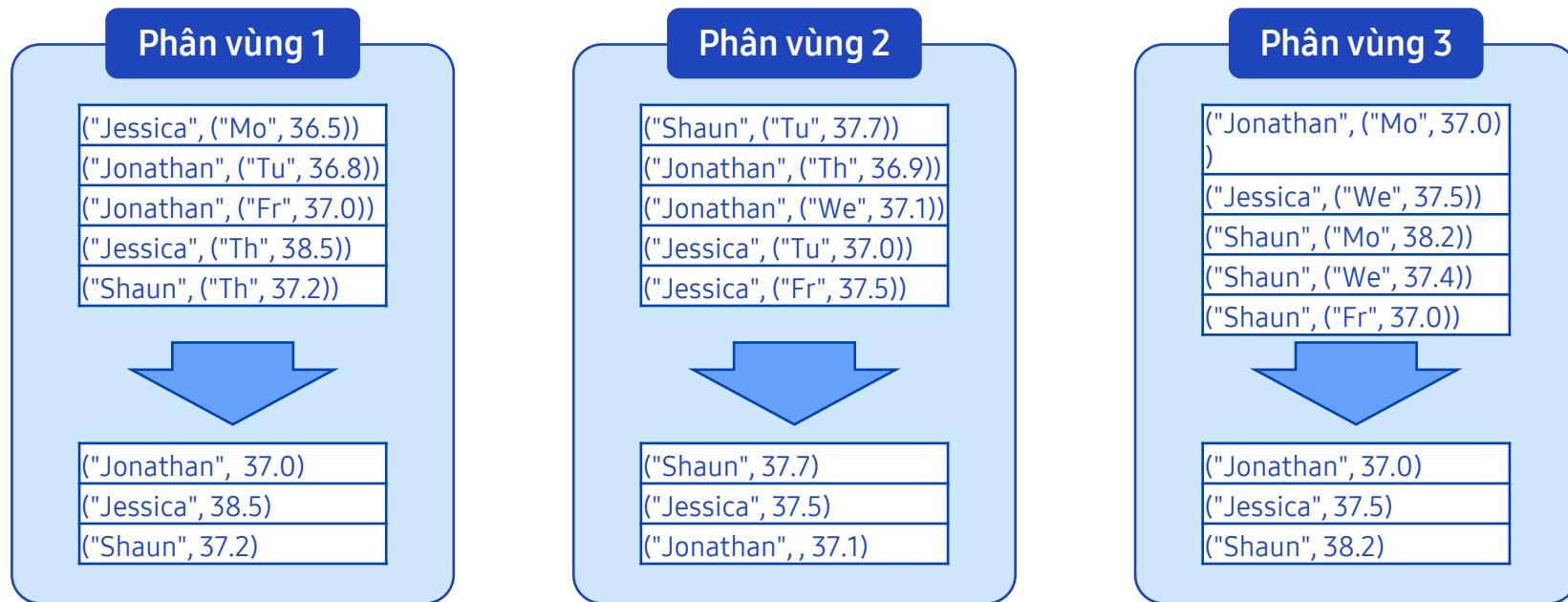
- | aggregateByKey nhận 3 đối số - zeroValue, seqOp và combOp
- | Định dạng nhập và xuất dữ liệu
  - ▶ Dữ liệu đầu vào để chuyển đổi có dạng (Khóa loại [K], Giá trị loại [V])
  - ▶ Dữ liệu trả về sau khi biến đổi có dạng (Khóa kiểu [K], Giá trị kiểu [U])
  - ▶ Ví dụ: Đầu vào ("Jessica", ("Monday", 36,5)) trả về Đầu ra ("Jessica", 36,5)
- | Dữ liệu được tổng hợp qua 2 bước, một lần tại mỗi phân vùng và một lần trên toàn bộ tập dữ liệu
- | seqOp sử dụng bộ tích lũy để tổng hợp giá trị ở cấp Phân vùng
  - ▶ zeroValue là giá trị bắt đầu của bộ tích lũy này
- | combOp lấy các giá trị tổng hợp trong từng phân vùng và tổng hợp chúng trên toàn bộ tập dữ liệu
  - ▶ Điều này yêu cầu tất cả các phân vùng chia sẻ dữ liệu của nhau
  - ▶ Vì seqOp đã tổng hợp phần Giá trị từ [V] đến [U], nên combOp phải là một hàm lấy đầu vào của biểu mẫu [U] và tổng hợp đầu ra của biểu mẫu [U]

## Chuyển đổi AggregateByKey (2/4)

- zeroValue - Đây là giá trị ban đầu của bộ tích lũy
- Giá trị ban đầu hoặc giá trị Zero
  - ▶ Nó có thể là 0 nếu tập hợp là loại tổng
  - ▶ Double.MaxValue hoặc giá trị tối thiểu đã biết nếu mục tiêu tổng hợp là tìm giá trị tối thiểu
  - ▶ Double.MinValue hoặc giá trị tối đa đã biết nếu mục tiêu tổng hợp là tìm giá trị tối đa
  - ▶ Đặt thành Danh sách trống, nếu chúng tôi chỉ muốn một bộ sưu tập làm đầu ra

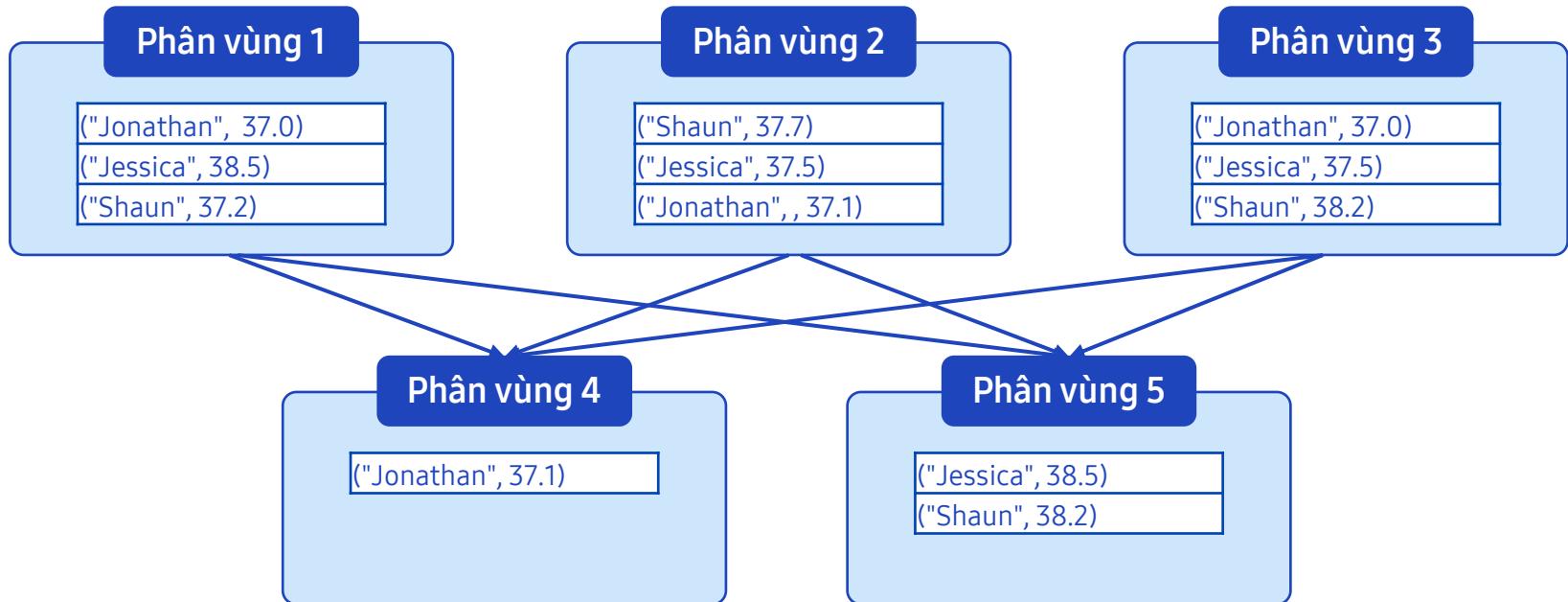
## Chuyển đổi AggregateByKey (3/4)

- seqOp - Hàm tổng hợp và chuyển đổi loại giá trị [V] thành loại [U]
- Thực hiện tổng hợp ở cấp độ phân vùng



## Chuyển đổi AggregateByKey (4/4)

- combOp - Hàm tổng hợp hoặc hợp nhất nhiều phần tử loại [U] thành một phần tử loại [U] duy nhất
  - Tổng hợp tất cả giá trị [U] cho mỗi khóa thành một



# Ví dụ về AggregateByKey

## Nguồn dữ liệu và cài đặt tham số

**Input**

```
data_src = [("Jessica", "Mo", 36.5), ("Shaun", "Tu", 37.7), ("Jonathan", "Mo", 37.0),
 ("Jonathan", "Tu", 36.8), ("Jonathan", "Th", 36.9), ("Jessica", "We", 37.5),
 ("Jonathan", "Fr", 37.0), ("Jonathan", "We", 37.1), ("Shaun", "Mo", 38.2),
 ("Jessica", "Th", 38.5), ("Jessica", "Tu", 37.0), ("Shaun", "We", 37.4),
 ("Shaun", "Th", 37.2), ("Jessica", "Fr", 37.5), ("Shaun", "Fr", 37.0)]
zeroValue = 0
def seqOp(accumulator, element):
 if(accumulator > element[1]):
 return accumulator
 else:
 return element[1]
def combOp(accumulator1, accumulator2):
 if(accumulator1 > accumulator2):
 return accumulator1
 else:
 return accumulator2
```

# Ví dụ về AggregateByKey

- Thực hiện **aggregateByKey** và kết quả đầu ra

```
aggRDD = sc.parallelize(data_src) \
 .map(lambda tup: (tup[0], (tup[1], tup[2]))) \
 .aggregateByKey(zeroValue, seqOp, combOp)

for i in aggRDD.collect():
 print(i)
```

Input

```
('Jessica', 38.5)
('Jonathan', 37.1)
('Shaun', 38.2)
```

Output

## Mã này sẽ tạo ra kết quả gì?

- | Giá trị bắt đầu của zeroValue đã được thay đổi ở một bộ (chuỗi, float)
- | seqOp và combOp đã được sửa đổi một chút

Input

```
zeroValue = ("", 0.0)
def seqOp(accumulator, element):
 if(accumulator[1] > element[1]):
 return accumulator
 else:
 return element

def combOp(accumulator1, accumulator2):
 if(accumulator1[1] > accumulator2[1]):
 return accumulator1
 else:
 return accumulator2
```

# Bạn có đoán chính xác không?

- Thay vì trả về nhiệt độ cao nhất, nó trả về nhiệt độ cao nhất và ngày xảy ra

```
aggRDD = sc.parallelize(data_src) \
 .map(lambda tup: (tup[0], (tup[1], tup[2]))) \
 .aggregateByKey(zeroValue, seqOp, combOp)

for i in aggRDD.collect():
 print(i)
```

**Input**

```
('Jessica', ('Th', 38.5))
('Jonathan', ('We', 37.1))
('Shaun', ('Mo', 38.2))
```

**Output**

# Chuyển đổi GroupByKey

- **groupByKey** tổng hợp tất cả các giá trị có cùng một khóa và tạo một tập hợp các giá trị
  - ▶ Đầu vào → (Key, Value)
  - ▶ Đầu ra → (Key, [Value1, Value2, Value3, ...])
- Về mặt kỹ thuật, kiểu dữ liệu Iterable được tạo cho bộ sưu tập Giá trị

| Input                                                                                                                                                                                                                                                                                                                                                                                                                              | Output                                                                                                                                                                                                                                                                                                                                                 |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|--------|--|------|--|------|-------|--------|--|--------|--|--------|-------|------------|--|------------|--|-------|
| <pre>ate_fruit = [('Henry', 'apples'), ('Shaun', 'strawberry'), ('Sharon', 'apples'),               ('Shaun', 'watermelon'), ('Sharon', 'pear'), ('Henry', 'banana'),               ('Henry', 'grapes'), ('Sharon', 'kiwi'), ("Shaun", 'peach')]  fav_fruits = sc.parallelize(ate_fruit) \     .groupByKey()  for (name, fruits) in fav_fruits.collect():     print(name)     for fruit in fruits:         print(' ', fruit)</pre> | <table border="0"><tr><td>Sharon</td><td>apples</td></tr><tr><td></td><td>pear</td></tr><tr><td></td><td>kiwi</td></tr><tr><td>Henry</td><td>apples</td></tr><tr><td></td><td>banana</td></tr><tr><td></td><td>grapes</td></tr><tr><td>Shaun</td><td>strawberry</td></tr><tr><td></td><td>watermelon</td></tr><tr><td></td><td>peach</td></tr></table> | Sharon | apples |  | pear |  | kiwi | Henry | apples |  | banana |  | grapes | Shaun | strawberry |  | watermelon |  | peach |
| Sharon                                                                                                                                                                                                                                                                                                                                                                                                                             | apples                                                                                                                                                                                                                                                                                                                                                 |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |
|                                                                                                                                                                                                                                                                                                                                                                                                                                    | pear                                                                                                                                                                                                                                                                                                                                                   |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |
|                                                                                                                                                                                                                                                                                                                                                                                                                                    | kiwi                                                                                                                                                                                                                                                                                                                                                   |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |
| Henry                                                                                                                                                                                                                                                                                                                                                                                                                              | apples                                                                                                                                                                                                                                                                                                                                                 |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |
|                                                                                                                                                                                                                                                                                                                                                                                                                                    | banana                                                                                                                                                                                                                                                                                                                                                 |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |
|                                                                                                                                                                                                                                                                                                                                                                                                                                    | grapes                                                                                                                                                                                                                                                                                                                                                 |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |
| Shaun                                                                                                                                                                                                                                                                                                                                                                                                                              | strawberry                                                                                                                                                                                                                                                                                                                                             |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |
|                                                                                                                                                                                                                                                                                                                                                                                                                                    | watermelon                                                                                                                                                                                                                                                                                                                                             |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |
|                                                                                                                                                                                                                                                                                                                                                                                                                                    | peach                                                                                                                                                                                                                                                                                                                                                  |        |        |  |      |  |      |       |        |  |        |  |        |       |            |  |            |  |       |

## Chuyển đổi Join

Phép biến đổi **join** tổng hợp tất cả các giá trị có cùng một Khóa và tạo một bộ Giá trị

- ▶ Đầu vào → (Key, Value)
- ▶ Đầu ra → (Key, (Value1, Value2))

Input

```
gen = [('Henry', 'M'), ('Henry', 'F'), ('Jessica', 'F'), ('Jonathan', 'M'), ('Shaun', 'M')]
age = [('Henry', 38), ('Henry', 16), ('Jessica', 22), ('Jonathan', 18), ('Shaun', 52)]
```

```
gen_rdd = sc.parallelize(gen)
age_rdd = sc.parallelize(age)
person_rdd = gen_rdd.join(age_rdd)
for i in person_rdd.collect():
 print(i)
```

Output

```
('Jessica', ('F', 22))
('Jonathan', ('M', 18))
('Henry', ('M', 38))
('Henry', ('M', 16))
('Henry', ('F', 38))
('Henry', ('F', 16))
('Shaun', ('M', 52))
```

# [Lab1]

## Làm việc với PySpark Shell



## [Lab2] Python cơ bản



[Lab3]

Làm việc với Chuyển đổi API lõi



## [Lab4]

### Làm việc với Cặp RDD



# [Lab5]

## Để tất cả chúng cùng nhau



Bài 2

# Xử lý dữ liệu có cấu trúc

Xử lý Big Data với Apache Spark

Bài 2.

# Xử lý dữ liệu có cấu trúc

- | **2.1. Giới thiệu về Spark SQL**
- | 2.2. Các thao tác SQL Spark
- | 2.3. Tương tác RDD và DataFrames

# Apache Spark SQL

- Spark SQL là một mô-đun Spark để xử lý dữ liệu có cấu trúc
- Không giống như Spark RDD API cơ bản, các giao diện cung cấp thêm thông tin về cấu trúc của dữ liệu và tính toán đang được thực hiện
- Trong nội bộ, Spark SQL sử dụng thông tin bổ sung này để thực hiện tối ưu hóa thông qua Catalyst Optimizer
  - ▶ Catalyst Optimizer phân tích các chuyển đổi DataFrame và tạo mã được tối ưu hóa
- Có một số cách tương tác với Spark SQL
  - ▶ Sử dụng các truy vấn SQL tiêu chuẩn ISO
  - ▶ Thông qua việc sử dụng API DataFrame và Datasets
  - ▶ Cả hai phương pháp đều sử dụng cùng một công cụ thực thi, đằng sau hậu trường, không phụ thuộc vào API hoặc ngôn ngữ nào đã được sử dụng
  - ▶ Cho phép các nhà phát triển tạo ứng dụng trong môi trường và nền tảng phù hợp nhất với họ

# Spark Session

- | Điểm vào cho Apache Spark DataFrame API là đối tượng `SparkSession`
- | Trong các phiên bản trước 2.x, có hai đối tượng riêng biệt được sử dụng làm điểm vào cho API DataFrames
  - ▶ `SQLContext` cung cấp chức năng SQL cơ bản
  - ▶ `HiveContext` đã thêm tương tác với Apache Hive
- | Hai phiên bản khác nhau với chức năng khác nhau đã tạo ra nhiều nhầm lẫn trong Spark commThuyy
- | Bắt đầu với 2.x Apache Spark đã hợp nhất điều này thành một đối tượng `SparkSession` duy nhất
  - ▶ Đối tượng mới này kết hợp chức năng của cả `SQLContext` và `HiveContext`
  - ▶ Bao gồm hỗ trợ tích hợp để viết truy vấn Hive (`HiveQL`)
  - ▶ Truy cập vào các chức năng do người dùng Hive xác định (`Hive UDF`)
  - ▶ Khả năng đọc dữ liệu từ các bảng Hive

# Spark DataFrames và Tập dữ liệu

- I DataFrame và Tập dữ liệu được sử dụng để biểu thị dữ liệu có cấu trúc trong Apache Spark
  - ▶ Tập dữ liệu chỉ khả dụng trong Scala và Java
- I Cách dễ nhất để nghĩ về DataFrames và Tập dữ liệu là dữ liệu có cột và hàng
  - ▶ Dữ liệu có lược đồ
  - ▶ Cột có tên cột và loại cột
  - ▶ Mỗi hàng dữ liệu đại diện cho một mục thông tin
  - ▶ Tương tự như các bảng trong cơ sở dữ liệu quan hệ

**DataFrame**

|       | Cột1 | Cột2 | Cột3 | ... |
|-------|------|------|------|-----|
| Hàng1 |      |      |      |     |
| Hàng2 |      |      |      |     |
| Hàng3 |      |      |      |     |
| .     |      |      |      |     |

# Các tập dữ liệu Apache Spark

- Tập dữ liệu là một tập hợp dữ liệu phân tán
  - ▶ Có sẵn kể từ Apache Spark 1.6
  - ▶ Cung cấp các lợi ích của RDD như gõ mạnh
  - ▶ Khả năng sử dụng các chức năng lambda mạnh mẽ
  - ▶ Thêm các lợi ích của công cụ thực thi được tối ưu hóa của Spark SQL (Trình tối ưu hóa chất xúc tác)
- Một Tập dữ liệu có thể được xây dựng từ các đối tượng JVM (Scala hoặc Java)
  - ▶ Mỗi hàng đối tượng JVM trong Tập dữ liệu được nhập mạnh và thực thi tại thời điểm biên dịch
  - ▶ Các lớp trường hợp Scala là một cách dễ dàng để tạo các đối tượng JVM
  - ▶ Các lớp trường hợp cung cấp một phương pháp nhanh chóng và dễ dàng để tạo các đối tượng bằng lược đồ
- Tập dữ liệu có thể được thao tác bằng cách sử dụng các phép biến đổi chức năng
  - ▶ map(), flatMap(), filter(), v.v.
- Tập dữ liệu API chỉ khả dụng cho Scala và Java

# Apache Spark DataFrames

- Sự khác biệt giữa DataFrames và Datasets (Tập dữ liệu) là cách dữ liệu được biểu diễn
  - ▶ Mỗi hàng trong DataFrame là một tập hợp lỏng lẻo của đối tượng **Row** được nhập động
  - ▶ DataFrames chỉ có thể chứa các đối tượng Hàng
  - ▶ Mỗi hàng trong Tập dữ liệu là đối tượng JVM được nhập mạnh
- Mỗi đối tượng **Row** chứa một tập hợp các giá trị được sắp xếp theo thứ tự
  - ▶ Các kiểu nguyên thủy cơ bản như số nguyên, số float, Boolean
  - ▶ Các bộ sưu tập như Chuỗi, Danh sách, Mảng, v.v.
- Về mặt khái niệm tương đương với một bảng trong cơ sở dữ liệu quan hệ
  - ▶ Tương tự như khung dữ liệu R hoặc Python
- DataFrames được hưởng lợi từ việc sử dụng Apache Spark's Catalyst Optimizer
  - ▶ Sản xuất mã được tối ưu hóa cao

# Spark Session

- Lớp SparkSession cung cấp các chức năng và thuộc tính để truy cập tất cả các chức năng của Spark
- Hầu hết chức năng Spark được truy cập như một phương thức của SparkSession
  - ▶ sql: thực hiện truy vấn Spark SQL
  - ▶ catalog: điểm vào cho API danh mục để quản lý bảng
  - ▶ read: chức năng đọc dữ liệu từ một tệp hoặc nguồn dữ liệu khác
  - ▶ conf: đối tượng để quản lý cài đặt cấu hình Spark
  - ▶ sparkContext: điểm vào cho Spark API cốt lõi
- Spark Shell cung cấp đối tượng này dưới tên biến "spark"

# Tạo Spark DataFrame

- Với SparkSession, các ứng dụng có thể tạo DataFrames từ:
  - ▶ Từ mã và dữ liệu được tạo trong bộ nhớ
  - ▶ RDD hiện có thông qua các biến đổi
  - ▶ Từ bảng Hive
  - ▶ Nguồn dữ liệu Spark
- Spark SQL hỗ trợ nhiều loại và định dạng nguồn dữ liệu
  - ▶ Các tệp văn bản như CSV, TSV, JSON, XML hoặc văn bản thuần túy
  - ▶ Tệp định dạng nhị phân
    - Apache Parquet - Định dạng mặc định của Spark
    - Apache ORC - Định dạng mặc định của Hive
    - Apache Avro - Được sử dụng rộng rãi để tuần tự hóa/giải tuần tự hóa dữ liệu
  - ▶ Nguồn đám mây công cộng
    - Amazon S3
    - Microsoft Azure Data Lake Storage Gen2

# Định dạng tệp nhị phân Hadoop

## Apache Parquet

- ▶ Tối ưu hóa, nhị phân, lưu trữ cột của dữ liệu có cấu trúc

## Định dạng dữ liệu Apache Avro

- ▶ Apache Avro là một hệ thống tuần tự hóa dữ liệu
- ▶ Lưu trữ dữ liệu ở định dạng nhỏ gọn, có cấu trúc, nhị phân, theo hàng

## Apache ORC

- ▶ Được tối ưu hóa cho cả quyền truy cập dựa trên cột và bản ghi, dành riêng cho hệ sinh thái Hadoop

## Định dạng tệp tuần tự (SequenceFile)

- ▶ Định dạng lưu trữ nhị phân Hadoop ban đầu

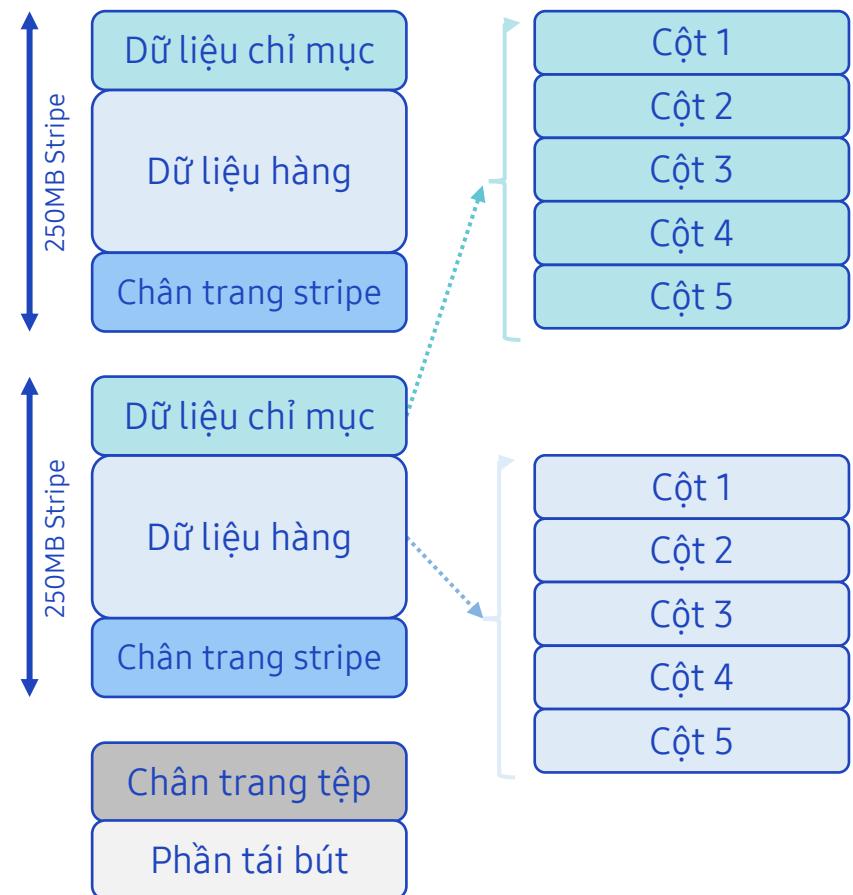
# Apache Parquet

- Parquet là một định dạng lưu trữ rất phổ biến
  - ▶ Được hỗ trợ bởi Sqoop, Spark, Hive, Impala và các công cụ khác
- Các tính năng chính
  - ▶ Tối ưu hóa lưu trữ nhị phân của dữ liệu có cấu trúc
  - ▶ Siêu dữ liệu lược đồ được nhúng trong tệp
  - ▶ Hiệu suất và kích thước hiệu quả cho lượng lớn dữ liệu
  - ▶ Parquet hoạt động tốt với Impala
- Sử dụng parquet-tools để xem lược đồ và dữ liệu tệp Parquet
  - ▶ Sử dụng show để hiển thị các bản ghi
  - ▶ Sử dụng kiểm tra để hiển thị thông tin siêu dữ liệu bao gồm lược đồ

```
$ parquet-tools show /path/to/parquet/file
$ parquet-tools inspect /path/to/parquet/file
```

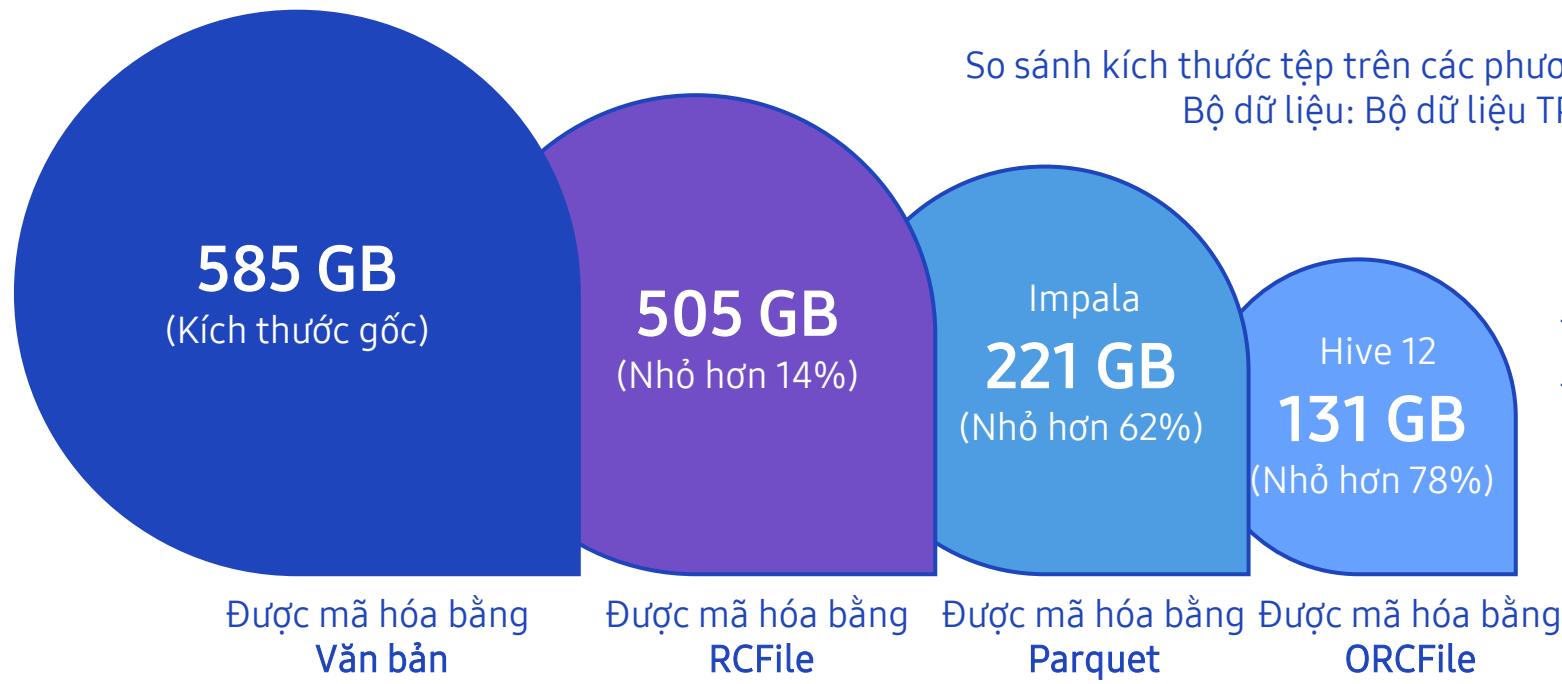
## Định dạng tệp ORC

- ORC là viết tắt của Tối ưu hóa bản ghi cột (Optimized Record Columnar)
- ORC là một định dạng dữ liệu cột hàng
- Tối ưu hóa cao cho việc đọc, viết và xử lý dữ liệu trong Hive
- Các tệp ORC được tạo từ các dải dữ liệu
- Mỗi sọc chứa chỉ mục, dữ liệu hàng và chân trang
- Số liệu thống kê chính (đếm, tối đa, tối thiểu, tổng) được lưu vào bộ đệm



## So sánh kích thước tệp

- Trong khi các tệp ORC nhỏ hơn, các tệp Parquet hiệu quả hơn để xử lý dữ liệu



- ✓ Kích thước khối lớn hơn
- ✓ Định dạng cột  
sắp xếp các cột  
Liền kề trong tệp  
để nén  
& truy cập nhanh

## Hàm tải và lưu chung

- Tải và lưu Khung dữ liệu (Dataframe) bằng định dạng Parquet mặc định

```
df = spark.read.load("file:/home/student/Data/resources/users.parquet")
df.select("name", "favorite_color").write.save("namesAndFavColors.parquet")
```

## Hàm tải và lưu chung

- Theo mặc định, Apache Spark nén các tệp định dạng Parquet bằng tính năng nén Snappy
  - Điều này có thể được nhìn thấy từ Hue, nơi tên tệp có hậu tố `***.snappy.parquet`

|  | Name                                                                | Size      | User    |
|--|---------------------------------------------------------------------|-----------|---------|
|  | _SUCCESS                                                            | 0 bytes   | student |
|  | .                                                                   |           | student |
|  | _SUCCESS                                                            | 0 bytes   | student |
|  | part-00000-3e3438ba-a233-44fc-bd82-b10c02d0e3eb-c000.snappy.parquet | 653 bytes | student |

## Chức năng tải và lưu chung

- | Không thể xem trực tiếp nội dung của các tệp định dạng Parquet từ Hue, không giống như các tệp văn bản.
- | Sử dụng công cụ parquet từ dòng lệnh
  - ▶ Sử dụng **hdfs dfs -get <name of parquet file>**
  - ▶ **parquet-tools show <name of parquet file>** để xem nội dung

```
[student@localhost Data]$ hdfs dfs -get namesAndFavColors.parquet/part*
[student@localhost Data]$ ls
alice_in_wonderland.txt
anonymous-msweb.data
part-00000-3e3438ba-a233-44fc-bd82-b10c02d0e3eb-c000.snappy.parquet
pig_data1.txt
pig_data2.txt
resources
scripts
[student@localhost Data]$ parquet-tools show part-00000-3e3438ba-a233-44fc-bd82-
b10c02d0e3eb-c000.snappy.parquet
+-----+
| name | favorite_color |
+-----+
| Alyssa |
| Ben | red |
+-----+
```

# Chỉ định định dạng thủ công

- Trong ví dụ trước, chúng tôi không chỉ định định dạng vì định dạng parquet mặc định đã được sử dụng
- Chúng tôi có thể chỉ định cụ thể định dạng của tệp nguồn và định dạng để lưu nó vào
  - ▶ Sử dụng kết hợp tải đọc của SparkSession để đọc tệp nguồn
  - ▶ Sử dụng kết hợp tải ghi của SparkSession để lưu tệp
  - ▶ Chỉ định định dạng làm tham số

```
SparkSession.read.format(<format>).load(<path>)
SparkSession.write.format(<format>).save(<path>)
```

```
peopleDF =
spark.read.format("json").load("file:/home/student/Data/resources/people.json")
peopleDF.select("name", "age").write.format("parquet").save("namesAndAges.parquet")
```

# Kết quả của lần tải và lưu trước đó

[student@localhost Data]\$ cat resources/people.json

```
{“name” : “Michael”}
{“name” : “Andy”, “age”:30}
{“name” : “Justin”, “age”:19}
```

**Input**

Home / user / student / namesAndAges.parquet

|   | Name                                                                | Size      | User    |
|---|---------------------------------------------------------------------|-----------|---------|
| □ | ↳                                                                   |           | student |
| □ | .                                                                   |           | student |
| □ | _SUCCESS                                                            | 0 bytes   | student |
| □ | part-00000-4fd79ba4-ab63-4f08-91e2-de77139ff9e9-c000.snappy.parquet | 687 bytes | student |

[student@localhost Data]\$ parquet-tools showde77139ff9e9-c000.snappy.parquet

```
+-----+-----+
| name | age |
+-----+-----+
Michael	nan
Andy	30
Justin	19
+-----+-----+
```

## Sử dụng Tùy chọn với Đọc-Tải

- | Đối với mỗi loại nguồn dữ liệu, có các tùy chọn ứng dụng có thể được chỉ định
- | Tại đây, đối với định dạng tệp CSV:
  - ▶ Chỉ định dấu phân cách bằng "sep"
  - ▶ Chỉ định xem Spark có nên suy luận lược đồ hay không
  - ▶ Chỉ định xem hàng đầu tiên có chứa hàng tiêu đề hay không

```
peopleDFcsv = spark.read.format("csv") \
 .option("sep", ";") \
 .option("inferSchema", "true") \
 .option("header", "true") \
 .load("file:/home/student/Data/resources/people.csv")
```

# Hiển thị lược đồ khung dữ liệu

- Khi DataFrame đã được tạo, chúng ta có thể xem lược đồ của DataFrame
  - Sử dụng thao tác `printSchema()`

```
peopleDFcsv = spark.read.format("csv") \
 .option("sep", ",") \
 .option("inferSchema", "true") \
 .option("header", "true") \
 .load("file:/home/student/Data/resources/people.csv")
```

Input

```
peopleDFcsv.printSchema()
```

Input

```
root
|-- name: string (nullable = true)
|-- age: integer (nullable = true)
|-- job: string (nullable = true)
```

Output

# Hiển thị nội dung của DataFrame

Chúng tôi cũng có thể xem nội dung của DataFrame

- ▶ Sử dụng thao tác `.show(n, <truncate>)`
- ▶ Hiển thị n hàng trong một màn hình được định dạng đẹp - mặc định là 20 hàng
- ▶ Đặt tùy chọn cắt ngắn thành True hoặc False - mặc định là True

```
peopleDFcsv = spark.read.format("csv") \
 .option("sep", ",") \
 .option("inferSchema", "true") \
 .option("header", "true") \
 .load("file:/home/student/Data/resources/people.csv")
```

Input

```
peopleDFcsv.show()
```

Input

```
+----+----+-----+
| name | age | job |
+----+----+-----+
| Jorge | 30 | Developer |
| Bob | 32 | Developer |
+----+----+-----+
```

Output

## Sử dụng Tùy chọn với Ghi-Lưu

- Giống như với đọc-tải, chúng ta có thể chỉ định các tùy chọn cho ghi-lưu
- Lưu tệp ở định dạng Orc với:
  - ▶ Đặt bộ lọc nở
  - ▶ Kiểm soát mã hóa từ điển

```
usersDF = spark.read.orc("file:/home/student/Data/resources/users.orc")
usersDF.write.format("orc") \
 .option("orc.bloom.filter.columns", "favorite_color") \
 .option("orc.dictionary.key.threshold", "1.0") \
 .option("orc.column.encoding.direct", "name") \
 .save("users_with_options.orc")
```

Input

```
root
| -- name: string (nullable = true)
| -- favorite_color: string (nullable = true)
| -- favorite_numbers: array (nullable = true)
| | -- element: integer (containsNull = true)
```

Output

# Chế độ lưu DataFrame

- Các hoạt động lưu có thể tùy chọn thực hiện một chế độ chỉ định cách xử lý dữ liệu hiện có nếu có

| Chế độ    | Ý nghĩa                                                                                                                                                                                                                                        |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| error     | Chế độ mặc định. Khi lưu DataFrame vào nguồn dữ liệu, nếu dữ liệu đã tồn tại, một ngoại lệ sẽ được đưa ra.                                                                                                                                     |
| append    | Khi lưu DataFrame vào nguồn dữ liệu, nếu dữ liệu/bảng đã tồn tại, nội dung của DataFrame dự kiến sẽ được thêm vào dữ liệu hiện có.                                                                                                             |
| overwrite | Chế độ ghi đè có nghĩa là khi lưu DataFrame vào nguồn dữ liệu, nếu dữ liệu/bảng đã tồn tại, thì dữ liệu hiện có dự kiến sẽ bị ghi đè bởi nội dung của DataFrame.                                                                               |
| ignore    | Chế độ bỏ qua có nghĩa là khi lưu DataFrame vào nguồn dữ liệu, nếu dữ liệu đã tồn tại, thao tác lưu dự kiến sẽ không lưu nội dung của DataFrame và không thay đổi dữ liệu hiện có. Điều này tương tự như CREATE TABLE IF NOT EXISTS trong SQL. |

## Tùy chọn nén dữ liệu Spark

- | Nén cho phép chúng tôi giảm kích thước của dữ liệu được lưu trữ
- | Đánh đổi giữa việc giảm băng thông I/O và thời gian CPU giải nén dữ liệu
  - ▶ Thường giảm mạnh kích thước có nghĩa là thời gian tính toán nhiều hơn và ít tích cực hơn là kích thước dữ liệu lớn hơn
- | Nhiều công việc Hadoop bị ràng buộc I/O và có thể cải thiện hiệu suất rất nhiều
  - ▶ Khi chúng ta hướng tới AI/ML nhiều hơn, mô hình truy cập đang trở nên lặp đi lặp lại và giới hạn tính toán
- | GZip tích cực hơn nhưng độ nén cao hơn
  - ▶ Tốt cho kho lạnh
- | Snappy là lựa chọn tốt nhất cho dữ liệu thường xuyên truy cập
- | BZip2 có thể đạt được mức nén thậm chí còn lớn hơn Gzip đối với một số loại tệp nhất định

## Đặt tùy chọn nén

- | DataFrames có thể được lưu với các tùy chọn nén
- | Cú pháp hơi khác đối với phiên bản Apache Spark
- | Spark 2.2+

```
df.write.option("compression", "<compression>").save(<path>)
```

- | Spark 2.0

```
df.write..save(<path>, compression=<compression>)
```

- | Spark 1.6

```
df.write..codec(<compression>).save(<path>)
```

# Sử dụng phím tắt định dạng tệp

- Apache Spark cung cấp các phím tắt định dạng tệp cho cả kết hợp đọc-tải và ghi-lưu
- Các định dạng được hỗ trợ:
  - ▶ JSON
  - ▶ CSV
  - ▶ ORC
  - ▶ Parquet

```
peopleDF = spark.read.<file format>(<path>)
spark.write.option(<opt>). <file format>(<path>)

CSV Example
peopleDF = spark.read.csv(<path>)
spark.write.option(<opt>).csv(<path>)
```

## Đọc bảng Hive

- Apache Hive cung cấp quyền truy cập giống như cơ sở dữ liệu vào dữ liệu được lưu trữ trong HDFS
  - ▶ Lược đồ và thông tin siêu dữ liệu khác được lưu trữ trong Hive metastore
  - ▶ Hive metastore thường là một RDBMS chẵng hạn như MySQL/MariaDB
  - ▶ Nội dung thực tế được lưu trữ trong HDFS
- Spark SQL hỗ trợ đọc và ghi dữ liệu được lưu trữ trong Apache Hive
  - ▶ Spark phải có cài đặt cấu hình cho Hive và HDFS trong thư mục cấu hình của nó
  - ▶ Phiên Spark phải được khởi tạo với hỗ trợ Hive được bật
  - ▶ Vị trí kho Hive, vị trí mặc định nơi các bảng Hive được lưu trữ trong HDFS
  - ▶ Spark-Shell tự động tạo Phiên Spark khi bật Hive

```
tableDF = spark.read.table(<table_name>)
```

# Lưu vào Hive Tables

- Spark DataFrames có thể được lưu vào Hive bằng lệnh saveAsTable
- Bảng sẽ được lưu dưới dạng bảng được quản lý và dữ liệu sẽ được lưu trữ trong thư mục Hive Warehouse

| Input                        |                                                                |         |
|------------------------------|----------------------------------------------------------------|---------|
| Col_name                     | Data_type                                                      | comment |
| Id                           | Int                                                            | Null    |
| First_name                   | String                                                         | Null    |
| # Detailed Table Information |                                                                |         |
| Database                     | Default                                                        |         |
| Table                        | Authors_managed                                                |         |
| Owner                        | Student                                                        |         |
| Created time                 | Mon Aug 30 01:45:25 KST 2021                                   |         |
| Last Access                  | UNKNOWN                                                        |         |
| Created by                   | Spark 3.1.2                                                    |         |
| Type                         | MANAGED                                                        |         |
| Provider                     | Parquet                                                        |         |
| Location                     | hdfs://localhost:9000/user/hive/warehouse/authors_managed      |         |
| Serde Library                | org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe    |         |
| InputFormat                  | org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat  |         |
| OutputFormat                 | org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat |         |

## Lưu vào các bảng Hive bên ngoài

Để lưu trữ dưới dạng bảng bên ngoài, hãy cung cấp tùy chọn đường dẫn

▶ Để lưu trữ dưới dạng bảng bên ngoài, hãy cung cấp tùy chọn đường dẫn

|                                                                                                                                                           |                                                                |         | Input  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|---------|--------|
| au.write.option("path", "/user/student/authors_external").saveAsTable("authors_external")<br>spark.sql("desc formatted authors_external").show(30, False) |                                                                |         |        |
| Col_name                                                                                                                                                  | Data_type                                                      | comment | Output |
| Id                                                                                                                                                        | Int                                                            | Null    |        |
| First_name                                                                                                                                                | String                                                         | Null    |        |
| # Detailed Table Information                                                                                                                              |                                                                |         |        |
| Database                                                                                                                                                  | Default                                                        |         |        |
| Table                                                                                                                                                     | Authors_external                                               |         |        |
| Owner                                                                                                                                                     | Student                                                        |         |        |
| Created time                                                                                                                                              | Mon Aug 30 02:19:57 KST 2021                                   |         |        |
| Last Access                                                                                                                                               | UNKNOWN                                                        |         |        |
| Created by                                                                                                                                                | Spark 3.1.2                                                    |         |        |
| Type                                                                                                                                                      | EXTERNAL                                                       |         |        |
| Provider                                                                                                                                                  | Parquet                                                        |         |        |
| Statistics                                                                                                                                                | 889 bytes                                                      |         |        |
| Location                                                                                                                                                  | hdfs://localhost:9000/user/student/authors_external            |         |        |
| Serde Library                                                                                                                                             | org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe    |         |        |
| InputFormat                                                                                                                                               | org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat  |         |        |
| OutputFormat                                                                                                                                              | org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat |         |        |

# Tạo DataFrame từ Bộ sưu tập

- I Sử dụng `createDataFrame(<collection of row elements>, <collection of column names>)`
- I Mỗi phần tử trong List trở thành một hàng trong DataFrame
  - ▶ Sử dụng Nested List hoặc Tuple cho từng mục cột
  - ▶ Cung cấp tên cho các cột dưới dạng bộ sưu tập được phân cách bằng dấu phẩy
  - ▶ Kiểu dữ liệu sẽ được suy ra

```
Input

my_list = [(1, "Henry", 23.00),
 (2, "Shaun", 15.25),
 (3, "Sharon", 45.50),
 (4, "Jessica", 20.00)]
my_schema = ("id", "name", "amt")
myDF = spark.createDataFrame(my_list,
my_schema)
myDF.printSchema()
myDF.show()
```

| root                          |                                   |                                  |
|-------------------------------|-----------------------------------|----------------------------------|
| -- id: long (nullable = true) | -- name: string (nullable = true) | -- amt: double (nullable = true) |
| 1                             | Henry                             | 23.0                             |
| 2                             | Shaun                             | 15.25                            |
| 3                             | Sharon                            | 45.5                             |
| 4                             | Jessica                           | 20.0                             |

Output

# Cung cấp thông tin lược đồ rõ ràng

- Sử dụng `createDataFrame(<collection of elements., <comma delimited schema string>)`
- Mỗi phần tử trong List trở thành một hàng trong DataFrame
  - ▶ Sử dụng Nested List hoặc Tuple cho từng mục cột
- Cung cấp thông tin lược đồ
  - ▶ "col\_name col\_type, col\_name col\_type...."

Input

```
my_list = [(1, "Henry", 23.00),
 (2, "Shaun", 15.25),
 (3, "Sharon", 45.50),
 (4, "Jessica", 20.00)]

my_schema = "id string, name string, amt
double"

myDF = spark.createDataFrame(my_list,
 my_schema)
myDF.printSchema()
myDF.show()
```

Output

```
root
|-- id: string (nullable = true)
|-- name: string (nullable = true)
|-- amt: double (nullable = true)

+---+---+---+
| id| name| amt|
+---+---+---+
1	Henry	23.0
2	Shaun	15.25
3	Sharon	45.5
4	Jessica	20.0
+---+---+---+
```

Bài 2.

# Xử lý dữ liệu có cấu trúc

- | 2.1. Giới thiệu về Spark SQL
- | **2.2. Các thao tác SQL Spark**
- | 2.3. Tương tác RDD và DataFrames

# Các thao tác SQL Spark

- | Tương tự như Core API, các hoạt động của Spark SQL có thể được phân loại
- | Biến đổi:
  - | Áp dụng các phép biến đổi và tạo Bộ dữ liệu/DataFrame mới từ bộ dữ liệu hiện có
- | Hành động:
  - | Trả về kết quả của Hành động cho chương trình Trình điều khiển dưới dạng đầu ra
- | Biến đổi bất biến
  - | Như trong API lõi, các phép biến đổi DataFrame/Bộ dữ liệu là bất biến
  - | Mỗi chuyển đổi giữ nguyên bản gốc và tạo DataFrame/Bộ dữ liệu mới

# Hành động Spark SQL

- Chúng ta đã thấy một vài Hành động Spark SQL
  - ▶ `fprintSchema()` - trả về thông tin lược đồ của DataFrame/Bộ dữ liệu
  - ▶ `show(n)` - in n hàng đầu tiên của Khung dữ liệu/Bộ dữ liệu ở đầu ra có định dạng đẹp
  - ▶ `write` - Lưu Khung dữ liệu/Bộ dữ liệu ở một vị trí đã chỉ định
- Một số Hành động thường được sử dụng khác là:
  - ▶ `Take(n)`- trả về n phần tử hàng đầu tiên của DataFrame/Bộ dữ liệu
    - Hành động `take(n)` trả về các phần tử thực tế.
    - Trong trường hợp của DataFrames, đây sẽ là đối tượng Hàng
    - Trong trường hợp của Bộ dữ liệu, đây thường là đối tượng lớp trường hợp
  - ▶ `count()` - trả về số hàng trong DataFrame/Bộ dữ liệu
  - ▶ `first()` - trả về phần tử đầu tiên của DataFrame/Bộ dữ liệu
  - ▶ `collect` - trả về tất cả các phần tử của DataFrame/Bộ dữ liệu

# Hành động Spark SQL take(n)

- Hãy so sánh hành động `show(n)` với hành động `take(n)`

```
peopleDFcsv = spark.read.format("csv") \
 .option("sep", ",") \
 .option("inferSchema", "true") \
 .option("header", "true") \
 .load("file:/home/student/Data/resources/people.csv")
```

Input

peopleDFcsv.take(2)

Input

```
[Row(name='Jorge', age=30, job='Developer'),
 Row(name='Bob', age=32, job='Developer')]
```

Output

peopleDFcsv.show(2)

Input

| Name  | Age | Job       |
|-------|-----|-----------|
| Jorge | 30  | Developer |
| Bob   | 32  | Developer |

Output

## Spark SQL take(n) và first()

- Hành động **take(n)** trả về một chuỗi hoặc tập hợp có kích thước n
  - ▶ **take(1)** trả về một chuỗi hoặc bộ sưu tập có kích thước 1
- Hành động **first()** trả về phần tử đầu tiên

```
peopleDFcsv = spark.read.format("csv") \
 .option("sep", ",") \
 .option("inferSchema", "true") \
 .option("header", "true") \
 .load("file:/home/student/Data/resources/people.csv")
peopleDFcsv.take(1)
```

Input

```
[Row(name='Jorge', age=30, job='Developer')]
```

Output

```
peopleDFcsv.first()
```

Input

```
Row(name='Jorge', age=30, job='Developer')
```

Output

## Thực hiện Lazy (lười biếng) và Eager (háo hức)

### I Chuyển đổi Lazy:

- ▶ Như trong Core API, các chuyển đổi không được áp dụng ngay lập tức
- ▶ Một hành động kích hoạt tất cả Bộ dữ liệu/Khung dữ liệu phụ thuộc được hiện thực hóa
- ▶ Như trong RDD, Spark theo dõi dòng cho từng DataFrame/Bộ dữ liệu và theo biểu đồ phụ thuộc DAG để hiện thực hóa DataFrames/Bộ dữ liệu phụ thuộc

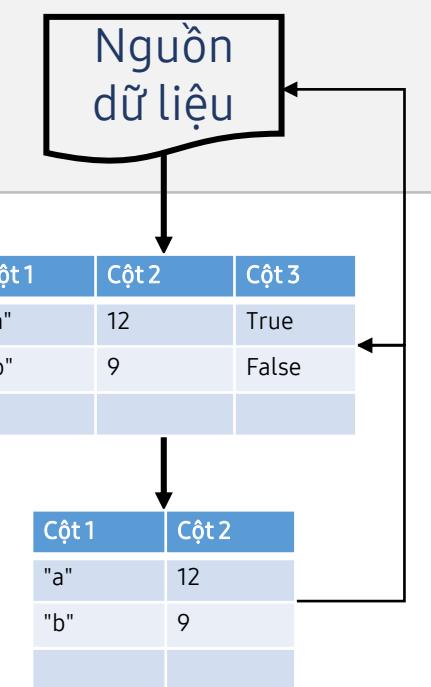
### I Tuy nhiên, lược đồ được thực hiện một cách háo hức.

- ▶ Nhiều bạn nhận thấy rằng Spark mất một thời gian sau khi đọc chuyển đổi
- ▶ Spark quét nguồn dữ liệu để xác định lược đồ
- ▶ Không thực sự đọc toàn bộ nội dung

# Thực thi đường ống Lazy và Eager

- Hoạt động lược đồ háo hức bằng hoạt động nội dung lười biếng

```
tableDF = spark.read.table("table_name")
filterDF = tableDF.select("column1", "column2")
filterDF.show(2)
```



## Chuyển đổi select()

- Sử dụng **select()** để chuyển đổi và tạo một DF/DS mới bao gồm các cột đã chọn

**Input**

```
authorDF = spark.read.table("authors")
authorDF.printSchema()
```

**Output**

```
root
 |-- id: integer (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- email: string (nullable = true)
 |-- birthdate: string (nullable = true)
 |-- added: string (nullable = true)
```

**Input**

```
emailDF = authorDF.select("id", "email")
emailDF.printSchema()
emailDF.show(5)
```

**Output**

```
root
 |-- id: integer (nullable = true)
 |-- email: string (nullable = true)
+---+-----+
| id| email |
+---+-----+
| 1| barmstrong@ex...
| 2| hand.stella@ex...
| 3| darren.blanda@ex...
| 4| shanahan.aliyah@e...
| 5| bednar.robin@ex...
+---+-----+
only showing top 5 rows
```

# Chuyển đổi where()

## Lọc các hàng bằng chuỗi điều kiện

- ▶ `filter()` và `where()` là các phép biến đổi DataFrame/Bộ dữ liệu tương đương

| Input                                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>emailDF = authorDF.select("id", "email") emailDF.printSchema() emailDF.show(5)</pre>                                                                                                                                                                                                                   |
| Output                                                                                                                                                                                                                                                                                                      |
| <pre>root   -- id: integer (nullable = true)   -- email: string (nullable = true) +---+   id           email   +---+   1   barmstrong@example...   2  hand.stella@example...   3  darren.blanda@example...   4  shanahan.aliyah@example...   5  bednar.robin@example... +---+ only showing top 5 rows</pre> |

| Input                                                                                                                                                                                                                                                                                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>bEmailDF = emailDF.where("id &gt; 10") bEmailDF.show(5)</pre>                                                                                                                                                                                                                                 |
| Output                                                                                                                                                                                                                                                                                             |
| <pre>root   -- id: integer (nullable = true)   -- email: string (nullable = true) +---+   id           email   +---+   1   virgil45@example...   2  dgleason@example...   3  Onie.wehner@example...   4  izulauf@example...   5  Marcellle.breitenb@example... +---+ only showing top 5 rows</pre> |

# Chuyển đổi limit(n) Spark SQL

- Giới hạn số hàng ở n hàng đầu tiên trong DataFrame/Bộ dữ liệu

**Input**

```
emailDF = authorDF.select("id", "email")
emailDF.printSchema()
emailDF.show(5)
```

**Output**

```
root
|-- id: integer (nullable = true)
|-- email: string (nullable = true)
+---+
| id| email |
+---+
1	barmstrong@ex...
2	hand.stella@ex...
3	darren.blanda@ex...
4	shanahan.aliyah@e
5	bednar.robin@ex...
+---+
only showing top 5 rows
```



**Input**

```
top2DF = emailDF.limit(2)
top2DF.show(5)
```

**Output**

```
root
|-- id: integer (nullable = true)
|-- email: string (nullable = true)
+---+
| id| email |
+---+
| 1| barmstrong@ex...|
| 2| hand.stella@ex...|
+---+
only showing top 5 rows
```

# Chuyển đổi orderBy() Spark SQL

- Là hoạt động này có vẻ tốn kém?
- orderBy(<order\_by\_column>) sắp xếp DataFrame/Bộ dữ liệu theo cột đã cho

[Thảo luận]

authorDF.orderBy("last\_name").show(5)

Input

| id    | first_name | last_name | email                 | birthdate  | added                |
|-------|------------|-----------|-----------------------|------------|----------------------|
| 12923 | Tiara      | Abbott    | ischaefer@example...  | 1994-02-24 | 1983-06-13 21:19:... |
| 12953 | Arnoldo    | Abbott    | taryn86@example.com   | 1974-05-28 | 1975-04-26 11:17:... |
| 12991 | Bettie     | Abbott    | joyce.torp@example... | 1985-05-23 | 1983-01-10 07:04:... |
| 13310 | Mario      | Abbott    | monty79@example.net   | 1986-07-24 | 1980-06-27 20:14:... |
| 13574 | Vanessa    | Abbott    | effie.marquardt@e...  | 2000-08-15 | 2005-12-27 06:59:... |

only showing top 5 rows

Output

# Sử dụng các cột trong chuyển đổi

- Sử dụng các cột trong chuyển đổi
- Hầu hết các chuyển đổi đều yêu cầu chỉ định các cột như một phần của tham số của nó
- Cho đến nay, chúng tôi đã sử dụng các chuỗi để chỉ ra các cột trong các phép biến đổi
  - ▶ **select("column1", "column2")**
  - ▶ **where("some string condition")**
  - ▶ **orderBy("column")**
- Các phép biến đổi phức tạp hơn yêu cầu phương thức đó được chuyển qua các đối tượng Cột
- Chuyển đổi bằng cách sử dụng các đối tượng Cột được gọi là biểu thức cột
- Quan sát phương thức select() và where() quá tải

```
def select(col: String, cols: String*) : DataFrame
 Selects a set of columns.
def select(cols: Column*) : DataFrame
 Selects a set of column based expressions.
```

```
def where(conditionExpr: String) : Dataset[T]
 Filters rows using the given SQL expression.
def where(condition: Column) : Dataset[T]
 Filters rows using the given condition.
```

# Tham chiếu đối tượng cột PySpark

- Python có hai ký hiệu khác nhau để chỉ các đối tượng Cột

```
peopleDF = spark.read \
 .json("file:/home/student/Data/resources/people.json")
peopleDF.printSchema()
```

Input

```
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

Output

```
peopleDF.select(peopleDF.age, peopleDF['name']).show()
```

Input

```
+----+-----+
| age | name |
+----+-----+
null	Michael
30	Andy
19	Justin
+----+-----+
```

Output

## Tham chiếu đối tượng cột Scala

- Scala có ba ký hiệu khác nhau để chỉ các đối tượng Cột

```
val peopleDF = spark.read.
 json("file:/home/student/Data/resources/people.json")
peopleDF.printSchema
peopleDF.select ($"age", peopleDF("name"), 'age').show
```

Input

```
+----+-----+
| age| name|age |
+----+-----+
null	Michael	null
30	Andy	30
19	Justin	19
+----+-----+
```

Output

## Hàm col()

- Cả Scala và Python đều cung cấp hàm `col("column_name")` trả về đối tượng Cột của cột được truyền dưới dạng tham số chuỗi

### Scala

```
peopleDF.
select(col("age")).
show
```

#### Input

```
+---+
| age|
+---+
| null|
| 30 |
| 19 |
+---+
```

#### Output

### Python

```
peopleDF. \
select(col("age")). \
show()
```

#### Input

```
+---+
| age|
+---+
| null|
| 30 |
| 19 |
+---+
```

#### Output

# Tại sao biểu thức cột?

- █ Sử dụng các biểu thức cột thay vì các chuỗi đơn giản, cho phép các nhà phát triển truy cập các hàm
- █ Hàm tích hợp sẵn:
  - ▶ Hàm vô hướng - Các hàm trả về một giá trị trên mỗi hàng
  - ▶ Các hàm tổng hợp - Các hàm trả về một giá trị duy nhất trên một nhóm hàng
- █ Hàm do người dùng định nghĩa:
  - ▶ Hàm vô hướng do người dùng định nghĩa
  - ▶ Các hàm tổng hợp do người dùng xác định
    - Chỉ trong Scala và Java

Input

```
peopleDF. \
 select(col("age"), col("age") + 10). \
 show()
```

Output

| age  | (age + 10) |
|------|------------|
| null | null       |
| 30   | 40         |
| 19   | 29         |

# Các hàm/toán tử vô hướng thường được sử dụng

## Toán tử số học

- ▶ +, -, \*, /, %

## Toán tử so sánh

- ▶ == ( === in Scala), >, >=, <, <=, !=, <>,

## Toán tử logic

- ▶ &, |, ~ (Python)
- ▶ &&, ||, !, and, or, not (Scala)

## Toán tử chuỗi

- ▶ substring, replace, contains, like, upper, lower

## Dữ liệu thử nghiệm

- ▶ isNull, isNotNull, NaN (không phải là thành viên)

## Input

```
my_list = [[1, "henry park", 23.00],
 [2, "shaun smith", 15.25],
 [3, "sharon ramirez", 45.50],
 [4, "jessica bolt", 20.00]]
my_schema = ("id", "name", "amt")
myDF = spark.createDataFrame(my_list, my_schema)
myDF \
 .select(upper(myDF.name), "amt") \
 .where((col("amt") > 18) & (myDF["amt"] < 40)) \
 .show()
```

## Output

| upper(name)  | amt  |
|--------------|------|
| HENRY PARK   | 23.0 |
| JESSICA BOLT | 20.0 |

## Hàm Ngày và Dấu thời gian

- █ Có một số tiêu chuẩn để lưu ngày giờ
  - ▶ Dấu thời gian Unix
  - ▶ Loại ngày
  - ▶ Loại dấu thời gian SQL
- █ Có thể chuyển đổi giữa các định dạng cũng như chỉ định hoặc sửa đổi định dạng ngày và giờ là một chức năng quan trọng

**Epoch Unix Timestamp**  
**1630503088**  
**SECONDS SINCE JAN 01 1970. (UTC)**  
**10:31:30 PM**

# Hàm định dạng ngày

Có các hàm ngày trả về một số phần của ngày ở định dạng được yêu cầu

| Hàm ngày PySpark            | Mô tả                                                                                                                  |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------|
| current_date                | Trả về ngày hiện tại khi bắt đầu đánh giá truy vấn.                                                                    |
| current_timestamp           | Trả về dấu thời gian hiện tại khi bắt đầu đánh giá truy vấn.                                                           |
| second(timestamp)           | Trả về thành phần thứ hai của chuỗi/dấu thời gian.                                                                     |
| minute(timestamp)           | Trả về thành phần phút của chuỗi/dấu thời gian.                                                                        |
| hour(timestamp)             | Trả về thành phần giờ của chuỗi/dấu thời gian.                                                                         |
| weekofyear(date)            | Trả về tuần trong năm của ngày đã cho. Một tuần được coi là bắt đầu vào Thứ Hai và tuần 1 là tuần đầu tiên có >3 ngày. |
| month(date)                 | Trả về thành phần tháng của ngày/dấu thời gian.                                                                        |
| quarter(date)               | Trả về quý của năm cho ngày, trong phạm vi từ 1 đến 4.                                                                 |
| year(date)                  | Trả về thành phần năm của ngày/dấu thời gian.                                                                          |
| date_format(timestamp, fmt) | Chuyển đổi dấu thời gian thành giá trị chuỗi ở định dạng được chỉ định bởi định dạng ngày fmt.                         |

# Sử dụng mẫu: Hàm định dạng ngày

```
from pyspark.sql.functions import *
authorDF = spark.read.table("authors")
authBdayDF = authorDF.select("first_name", "last_name", \
 weekofyear(authorDF.birthdate), \
 date_format(authorDF.added, "y"))
authBdayDF.printSchema()
authBdayDF.show(5)
```

**Input**

```
root
|-- first_name: string (nullable = true)
|-- last_name: string (nullable = true)
|-- weekofyear(birthdate): integer (nullable = true)
|-- date_format(added, y): string (nullable = true)
+---+---+---+
|first_name| last_name|weekofyear(birthdate)|date_format(added, y)|
+---+---+---+
Walton	Adams	9	1997
Marietta	Walsh	22	2010
Lily	Wintheiser	34	1973
Estevan	Gleason	29	1995
Thaddeus	Rowe	9	2017
+---+---+---+
only showing top 5 rows
```

**Output**

# Hàm alias Column

## Column.alias(alias)

- ▶ Trả về cột có bí danh với một tên mới
- ▶ Trong lần chuyển đổi cuối cùng , chúng tôi đã tạo một số tên cột phản ánh đúng phép tính biểu thức cột mà chúng tôi đã thực hiện trên đó
- ▶ Hãy sử dụng bí danh để đổi tên các cột đó

| Input                                                                                                                                                                                                                                                                                                                      | Output                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>from pyspark.sql.functions import * authorDF = spark.read.table("authors") authBdayDF = authorDF \     .select(         "first_name",         "last_name",         weekofyear(authorDF.birthdate)     .alias("Week Born"),     date_format(authorDF.added, "y")     .alias("Member Since"))  authBdayDF.show(5)</pre> | <pre>+-----+-----+-----+-----+   first_name  last_name Week Born Member Since  +-----+-----+-----+-----+      Walton      Adams       9      1997    Marietta      Walsh      22      2010         Lily Wintheiser      34      1973      Estevan      Gleason      29      1995    Thaddeus        Rowe       9      2017  +-----+-----+-----+-----+ only showing top 5 rows</pre> |

## Hàm tính ngày

Các hàm này thực hiện một số tính toán vào một ngày nhất định

| Hàm ngày PySpark                   | Mô tả                                                                   |
|------------------------------------|-------------------------------------------------------------------------|
| add_months(start_date, num_months) | Trả về ngày là num_tháng sau start_date.                                |
| date_add(start_date, num_days)     | Trả về ngày là num_days sau start_date.                                 |
| date_sub(start_date, num_days)     | Trả về ngày là num_days trước start_date.                               |
| datediff(endDate, startDate)       | Trả về số ngày từ startDate đến endDate.                                |
| last_day(date)                     | Trả về ngày cuối cùng của tháng chứa ngày đó.                           |
| months_between(date1, date2)       | Trả về số tháng giữa hai ngày                                           |
| next_day(start_date, day_of_week)  | Trả về ngày đầu tiên trễ hơn start_date                                 |
| to_date(date_str[, fmt])           | Chuyển đổi kiểu chuỗi chứa giá trị ngày tháng sang định dạng ngày tháng |

# Chuyển đổi withColumn()

## | DataFrame.withColumn(colName, colExpression)

- ▶ Trả về một DataFrame mới bằng cách thêm một cột hoặc thay thế cột hiện tại có cùng tên
- ▶ colName: tên chuỗi của cột mới hoặc cột thoát
- ▶ colExpression: biểu thức cột của cột mới hoặc cột thay thế

## | Sử dụng datadiff làm biểu thức cột để tính số ngày giữa ngày được thêm và ngày hiện tại

```
Input

from pyspark.sql.functions import *
authorDF = spark.read.table("authors")
authBdayDF = authorDF \
 .select("added") \
 .withColumn("On Record",
 datediff(current_date(),
 to_date(authorDF.added)))

authBdayDF.printSchema()
authBdayDF.show(5, False)
```

| root                               | Output                                  |
|------------------------------------|-----------------------------------------|
| -- added: string (nullable = true) | -- On Record: integer (nullable = true) |
| +-----+-----+                      | +-----+-----+                           |
| added                              | On Record                               |
| +-----+-----+                      | +-----+-----+                           |
| 1997-01-02 04:18:41.0   9009       |                                         |
| 2010-08-26 18:20:14.0   4025       |                                         |
| 1973-06-11 07:28:12.0   17615      |                                         |
| 1995-01-29 16:08:31.0   9713       |                                         |
| 2017-01-05 04:13:48.0   1701       |                                         |
| +-----+-----+                      | +-----+-----+                           |
| only showing top 5 rows            |                                         |

## Hàm when

- Hàm **when** đánh giá các điều kiện đã cho và trả về các giá trị tương ứng
- Có thể xâu chuỗi bổ sung mệnh đề **when**
  - ▶ Hoạt động như mệnh đề **if/elif/else** trong Python
- Sử dụng **otherwise** để bắt khi tất cả khi không thành công
  - ▶ Hoạt động như mệnh đề khác trong Python

| Input                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Output                                                                                                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>dataDF = spark.createDataFrame([(1, "a", "4"),                                 (2, "b", "0"),                                 (3, "b", "4"),                                 (4, "d", "4")],                                 ("id", "code", "amt"))  dataDF.withColumn("when_column",                   when((col("code") == "a")   (col("code") == "d"), "A")                     .when((col("code") == "b") &amp; (col("amt") == "4"), "B")                     .otherwise("A1")).show()</pre> | <pre>+---+---+---+-----+   id code amt when_column  +---+---+---+-----+    1    a    4       A     2    b    0      A1     3    b    4       B     4    d    4       A  +---+---+---+-----+</pre> |

## Hàm vô hướng do người dùng định nghĩa (1/2)

Dữ liệu này chứa danh sách dưới dạng một trong các cột.

- ▶ Tạo một UDF đếm số lượng số yêu thích

```
src = "file:/home/student/Data/resources/users.parquet"
usersDF = spark.read \
 .parquet(src)
usersDF.show()
```

Input

```
+-----+-----+
| name|favorite_color|favorite_numbers|
+-----+-----+
|Alyssa| null| [3, 9, 15, 20]|
| Ben| red| []|
+-----+-----+
```

Output

## Hàm vô hướng do người dùng định nghĩa (2/2)

### I Tạo UDF bằng hàm udf()

- ▶ Cần nhập udf
- ▶ Sử dụng ký hiệu lambda trong hàm udf để truyền tham số cho UDF

### I Hàm udf yêu cầu nhập kiểu dữ liệu đầu ra

- ▶ Nhập kiểu dữ liệu cần thiết từ pyspark.sql.types

```
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType

Example of user-defined Scalar function
def cnt_fav(arr):
 return len(arr)

cntFavUDF = udf(lambda a: cnt_fav(a), IntegerType())

usersDF.select(
 col("name"),
 cntFavUDF(col("favorite_numbers"))
 .alias("Fav_Cnt")) \
 .show()
```

Input

Output

| +-----+-----+  |  |
|----------------|--|
| name   Fav_Cnt |  |
| +-----+-----+  |  |
| Alyssa   4     |  |
| Ben   0        |  |
| +-----+-----+  |  |

## Chuyển đổi groupBy

- Nhóm DataFrame bằng cách sử dụng các cột đã chỉ định để chúng tôi có thể chạy tổng hợp trên chúng
  - ▶ Trả về đối tượng pyspark.sql.GroupedData
- Đối tượng GroupedData có nhiều phương thức tổng hợp dữ liệu trên các hàng được nhóm trên cột đã chỉ định
  - ▶ `count()`, `max()`, `min()`, `sum()`, `mean()` thực hiện chức năng tổng hợp dự kiến
  - ▶ Sử dụng **agg(aggregation\_function)** để truy cập tất cả các chức năng tổng hợp tích hợp

| Input                                                                                                                                                                                                                                                                                                                                                      | Output                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>my_list = [ ["henry park", 'M', 23.00],<br/>           ["shaun smith", 'M', 15.25],<br/>           ["sharon ramirez", 'F', 45.50],<br/>           ["jessica bolt", 'F', 20.00]]<br/><br/>my_schema = "name string, gender string, amt double"<br/>myDF = spark.createDataFrame(my_list, my_schema)<br/>myDF.groupBy("gender").sum("amt").show()</pre> | <pre>+-----+-----+<br/>  gender   sum(amt)  <br/>+-----+-----+<br/>       F       65.5  <br/>       M      38.25  <br/>+-----+-----+</pre> |

## Tổng hợp nhiều cột (1/5)

- | Sử dụng trình bao bọc `agg()`, có thể tính toán nhiều tập hợp trên nhiều cột
  - ▶ Nguồn dữ liệu được sử dụng để phân tích tổng hợp

Input

```
from pyspark.sql.functions import col, sum, avg, max

my_company = [
 ("Henry", "Engineering", "TX", 105000, 20000),
 ("Jessica", "Sales", "TX", 75000, 15000),
 ("Shaun", "Finance", "CA", 70000, 14000),
 ("Sharon", "Marketing", "TX", 90000, 18000),
 ("Jonathan", "HR", "CA", 95000, 19000),
 ("Jason", "Engineering", "TX", 110000, 21000),
 ("Scott", "Finance", "CA", 85000, 17000),
 ("Timothy", "Marketing", "CA", 95000, 19000)]

schema = ["employee_name", "department", "state", "salary", "bonus"]
df = spark.createDataFrame(my_company, schema)
```

## Tổng hợp nhiều cột (2/5)

- Sử dụng trình bao bọc `agg()`, có thể tính toán nhiều tập hợp trên nhiều cột
  - Nhiều cột có thể được nhóm lại theo
  - Các hàm tổng hợp hỗ trợ nó có thể có nhiều tham số cột

```
df.groupBy("state", "department") \
 .sum("salary", "bonus") \
 .show(truncate=False)
```

Input

| state | department  | sum(salary) | sum(bonus) |
|-------|-------------|-------------|------------|
| CA    | Marketing   | 95000       | 19000      |
| TX    | Engineering | 215000      | 41000      |
| CA    | Finance     | 155000      | 31000      |
| TX    | Sales       | 75000       | 15000      |
| TX    | Marketing   | 90000       | 18000      |
| CA    | HR          | 95000       | 19000      |

Output

## Tổng hợp nhiều cột (3/5)

| Sử dụng trình bao bọc `agg()`, có thể tính toán nhiều tập hợp trên nhiều cột

▶ Sử dụng `agg()`, chúng ta có thể tạo nhiều cột đầu ra của dữ liệu tổng hợp trên cùng một cột

```
df.groupBy("department") \
 .agg(sum("salary").alias("Sum of Salaries"), \
 mean("salary").alias("Mean Salary"), \
 max("bonus").alias("Max Bonus")) \
 .show(truncate=False)
```

Input

| department  | Sum of Salaries | Mean Salary | Max Bonus |
|-------------|-----------------|-------------|-----------|
| Sales       | 75000           | 75000.0     | 15000     |
| Engineering | 215000          | 107500.0    | 21000     |
| HR          | 95000           | 95000.0     | 19000     |
| Finance     | 155000          | 77500.0     | 17000     |
| Marketing   | 185000          | 92500.0     | 19000     |

Output

## Tổng hợp nhiều cột (4/5)

- Sử dụng trình bao bọc agg(), có thể tính toán nhiều tập hợp trên nhiều cột
  - Có thể lọc đầu ra của đầu ra tổng hợp bằng mệnh đề where()

```
df.groupBy("department") \
 .agg(sum("salary").alias("Sum of Salaries"), \
 mean("salary").alias("Mean Salary"), \
 max("bonus").alias("Max Bonus")) \
 .where(col("Max Bonus") >= 20000) \
 .show(truncate=False)
```

Input

| department  | Sum of Salaries | Mean Salary | Max Bonus |
|-------------|-----------------|-------------|-----------|
| Engineering | 215000          | 107500.0    | 21000     |

Output

## Tổng hợp nhiều cột (5/5)

- Cuối cùng, sắp xếp lại các cột tổng hợp để có kết quả trực quan hơn.

```
df.groupBy("state", "department") \
 .sum("salary", "bonus") \
 .orderBy("state", "department") \
 .show(truncate=False)
```

**Input**

| state | department  | sum(salary) | sum(bonus) |
|-------|-------------|-------------|------------|
| CA    | Finance     | 155000      | 31000      |
| CA    | HR          | 95000       | 19000      |
| CA    | Marketing   | 95000       | 19000      |
| TX    | Engineering | 215000      | 41000      |
| TX    | Marketing   | 90000       | 18000      |
| TX    | Sales       | 75000       | 15000      |

**Output**

## Sử dụng chuyển đổi join() (1/4)

| Join với một DataFrame khác, sử dụng biểu thức tham gia được cung cấp theo dòng

| **dataframe1.join(datafram2, <string, list of string, or column expression to join on>, <join type>)**

- ▶ Nếu liên kết trên là một chuỗi hoặc danh sách các chuỗi chỉ ra cột để tham gia, (các) tên cột phải tồn tại trên cả hai khung dữ liệu
- ▶ Hoặc sử dụng biểu thức cột với toán tử so sánh đẳng thức

```
my_staff = [
 (1,"Henry","Engineering","TX"),
 (2,"Jessica","Sales","TX"),
 (3,"Shaun","Finance","CA"),
 (4,"Sharon","Marketing","TX"),
 (5,"Jonathan","HR","CA"),
 (6,"Jason","Engineering","TX"),
 (7,"Scott","Finance","CA"),
 (8,"Timothy","Marketing","CA")]

schema = ["emp_id", "employee_name",
 "department", "state"]
staffDF = spark \
 .createDataFrame(my_staff, schema)
```

```
income = [
 (1,105000,20000),
 (2,75000,15000),
 (3,70000,14000),
 (4,90000,18000),
 (5,95000,19000),
 (6,110000,21000)]

schema = ["emp_id", "salary", "bonus"]
incomeDF = spark \
 .createDataFrame(income, schema)
```

## Sử dụng chuyển đổi join() (2/4)

- Khi tên cột tham gia giống nhau, chúng ta có thể sử dụng một chuỗi
  - Đầu ra tạo ra một cột duy nhất để nối trên cột

```
staffDF.join(incomeDF, "emp_id").show(truncate=False)
```

Input

| emp_id | employee_name | department  | state | salary | bonus |
|--------|---------------|-------------|-------|--------|-------|
| 6      | Jason         | Engineering | TX    | 110000 | 21000 |
| 5      | Jonathan      | HR          | CA    | 95000  | 19000 |
| 1      | Henry         | Engineering | TX    | 105000 | 20000 |
| 3      | Shaun         | Finance     | CA    | 70000  | 14000 |
| 2      | Jessica       | Sales       | TX    | 75000  | 15000 |
| 4      | Sharon        | Marketing   | TX    | 90000  | 18000 |

Output

## Sử dụng chuyển đổi join() (3/4)

| Nếu liên kết trên các tên cột khác nhau, hãy sử dụng biểu thức cột với các đối tượng cột

- ▶ Đầu ra tạo ra cả hai cột nối trên các cột

```
income = [(1,105000,20000),(2,75000,15000),(3,70000,14000),
 (4,90000,18000),(5,95000,19000),(6,110000,21000)]
```

Input

```
schema = ["id","salary","bonus"]
incomeDF = spark \
 .createDataFrame(income,schema)
staffDF \
 .join(incomeDF, staffDF.emp_id == incomeDF.id) \
 .show(truncate=False)
```

| emp_id | employee_name | department  | state | id | salary | bonus |
|--------|---------------|-------------|-------|----|--------|-------|
| 6      | Jason         | Engineering | TX    | 6  | 110000 | 21000 |
| 5      | Jonathan      | HR          | CA    | 5  | 95000  | 19000 |
| 1      | Henry         | Engineering | TX    | 1  | 105000 | 20000 |
| 3      | Shaun         | Finance     | CA    | 3  | 70000  | 14000 |
| 2      | Jessica       | Sales       | TX    | 2  | 75000  | 15000 |
| 4      | Sharon        | Marketing   | TX    | 4  | 90000  | 18000 |

Output

## Sử dụng chuyển đổi join() (3/4)

### I Tham gia cung cấp nhiều tùy chọn tham gia

- ▶ Mặc định là inner join
- ▶ left, leftouter, right, rightouter v.v.

```
staffDF \
 .join(incomeDF,
 staffDF.emp_id == incomeDF.id,
 "left_outer") \
 .show(truncate=False)
```

Input

| emp_id | employee_name | department  | state | id   | salary | bonus |
|--------|---------------|-------------|-------|------|--------|-------|
| 7      | Scott         | Finance     | CA    | null | null   | null  |
| 6      | Jason         | Engineering | TX    | 6    | 110000 | 21000 |
| 5      | Jonathan      | HR          | CA    | 5    | 95000  | 19000 |
| 1      | Henry         | Engineering | TX    | 1    | 105000 | 20000 |
| 3      | Shaun         | Finance     | CA    | 3    | 70000  | 14000 |
| 8      | Timothy       | Marketing   | CA    | null | null   | null  |
| 2      | Jessica       | Sales       | TX    | 2    | 75000  | 15000 |
| 4      | Sharon        | Marketing   | TX    | 4    | 90000  | 18000 |

Output

# Thao tác đã nhập và chưa nhập

- Các thao tác của bộ dữ liệu có thể được phân loại thành các phép biến đổi được nhập và không được nhập
- Thao tác đã nhập
  - ▶ Các phép biến đổi duy trì kiểu dữ liệu Tập dữ liệu được nhập mạnh
  - ▶ Đối tượng JVM kiểu [U] → Đối tượng JVM kiểu [U]
- Thao tác chưa gõ
  - ▶ Chuyển đổi phá hủy kiểu dữ liệu Bộ dữ liệu ban đầu
  - ▶ Bởi vì các phép biến đổi chưa được định kiểu tạo ra DataFrames, nên đôi khi chúng được gọi là thao tác DataFrame
  - ▶ Loại đối tượng JVM [U] → Loại đối tượng hàng

# Ví dụ về phép chuyển đổi Untyped

## Phép chuyển đổi Untyped

```
#spark, df are from the previous example
#Print the schema in a tree format
df.printSchema()
```

Input

```
root
| -- age: long (nullable = true)
| -- name: string (nullable = true)
```

Output

```
#Select only the "name" column
df.select("name").show()
```

Input

```
+-----+
| name|
+-----+
|Michael|
| Andy|
| Justin|
+-----+
```

Output

# Ví dụ về phép chuyển đổi Untyped

## Chuyển đổi Untyped

```
Select everybody, but increment
the age by 1

df.select(df['name'], df['age'] + 1).show()
```

Input

```
+-----+
| name | (age + 1) |
+-----+
Michael	null
Andy	31
Justin	20
+-----+
```

Output

## Sử dụng truy vấn SQL

- | Bạn có thể truy vấn các bảng và khung dữ liệu bằng các truy vấn SQL tiêu chuẩn
  - ▶ Sử dụng phương thức **SparkSession.sql (<ISO standard query>)**
- | Hữu ích cho các lập trình viên quen thuộc hơn với SQL
  - ▶ Thay thế cho các phương thức API DataFrame và biểu thức Cột
- | Cho phép truy vấn đặc biệt
  - ▶ Đặc biệt dễ dàng khi sử dụng công cụ dòng lệnh Spark SQL - spark-sql

| \$ spark-sql                                 |          |            |                            |            |            |           | Cmd    |
|----------------------------------------------|----------|------------|----------------------------|------------|------------|-----------|--------|
| Spark-sql> select * from authors limit 5;    |          |            |                            |            |            |           | Input  |
|                                              |          |            |                            |            |            |           | Output |
| 1                                            | Walton   | Adams      | barmstrong@example...      | 1989-03-01 | 1997-01-02 | 04:18:... |        |
| 2                                            | Marietta | Walsh      | hand.stella@example...     | 2018-05-30 | 2010-08-26 | 18:20:... |        |
| 3                                            | Lily     | Wintheiser | darren.blanda@example...   | 1981-08-21 | 1973-06-11 | 07:28:... |        |
| 4                                            | Estevan  | Gleason    | shanahan.aliyah@example... | 2013-07-17 | 1995-01-29 | 16:08:... |        |
| 5                                            | Thaddeus | Rowe       | bednar.robin@example...    | 2019-02-26 | 2017-01-05 | 04:13:... |        |
| Time taken : 0.586 seconds, Fetched 5 row(s) |          |            |                            |            |            |           |        |

# Sử dụng SparkSession.sql với Hive

- Sử dụng SparkSession.sql để sử dụng các truy vấn SQL tiêu chuẩn

```
authorsDF = spark.sql("SELECT * FROM authors LIMIT 5")
authorsDF.show()
```

**Input**

| id | first_name | last_name  | email                       | birthdate  | added                |
|----|------------|------------|-----------------------------|------------|----------------------|
| 1  | Walton     | Adams      | barmstrong@example.com      | 1989-03-01 | 1997-01-02 04:18:... |
| 2  | Marietta   | Walsh      | hand.stella@example.com     | 2018-05-30 | 2010-08-26 18:20:... |
| 3  | Lily       | Wintheiser | darren.blanda@example.com   | 1981-08-21 | 1973-06-11 07:28:... |
| 4  | Estevan    | Gleason    | shanahan.aliyah@example.com | 2013-07-17 | 1995-01-29 16:08:... |
| 5  | Thaddeus   | Rowe       | bednar.robin@example.com    | 2019-02-26 | 2017-01-05 04:13:... |

**Output**

# Truy vấn SQL có thể phức tạp

## Cách sử dụng dấu ngoặc kép trong SQL

```
authorsDF = spark \
 .sql("""SELECT first_name, last_name, Birth_Week, Member_Since FROM
 (SELECT first_name, last_name, weekofyear(birthdate) as Birth_Week,
 date_format(added, "y") as Member_Since FROM authors)
 WHERE Member_Since >= "2000"
 LIMIT 5 """)
authorsDF.show()
```

Input

| first_name | last_name  | Birth_Week | Member_Since |
|------------|------------|------------|--------------|
| Marietta   | Walsh      | 22         | 2010         |
| Thaddeus   | Rowe       | 9          | 2017         |
| Cortez     | Russel     | 24         | 2007         |
| Caterina   | Cartwright | 40         | 2001         |
| Skye       | Powlowski  | 34         | 2000         |

Output

## Sử dụng `SparkSession.sql` với `DataFrames`

- | Để sử dụng khung dữ liệu làm bảng trong truy vấn SQL, phải tạo chế độ xem tạm thời cho chúng
- | Chế độ xem tạm thời nằm trong phạm vi phiên và sẽ biến mất nếu phiên tạo ra nó kết thúc
  - ▶ `createTempView(<name of view>)`
  - ▶ `createOrReplaceTempView (<name of view>)`
- | Nếu chế độ xem tạm thời cần được chia sẻ giữa tất cả các phiên và được duy trì cho đến khi ứng dụng Spark kết thúc, hãy sử dụng chế độ xem tạm thời toàn cầu
  - ▶ `createGlobalTempView (<name of view>)`
  - ▶ `createOrReplaceGlobalTempView (<name of view>)`
  - ▶ Chế độ xem tạm thời toàn cầu được lưu trong cơ sở dữ liệu được bảo tồn của hệ thống Spark
  - ▶ Sử dụng `global_view.<name of view>` để truy cập chế độ xem toàn cầu

## Ví dụ Sử dụng Chế độ xem tạm thời

- Tạo chế độ xem tạm thời cho staffDF từ ví dụ trước
- Sử dụng tên dạng xem tạm thời trong truy vấn SQL

```
staffDF.createOrReplaceTempView("staff")
spark.sql(""" SELECT * FROM staff """).show()
```

Input

| emp_id | employee_name | department  | state |
|--------|---------------|-------------|-------|
| 1      | Henry         | Engineering | TX    |
| 2      | Jessica       | Sales       | TX    |
| 3      | Shaun         | Finance     | CA    |
| 4      | Sharon        | Marketing   | TX    |
| 5      | Jonathan      | HR          | CA    |
| 6      | Jason         | Engineering | TX    |
| 7      | Scott         | Finance     | CA    |
| 8      | Timothy       | Marketing   | CA    |

Output

## Truy vấn không có Chế độ xem tạm thời

- Đôi khi, chúng tôi có thể muốn thực hiện truy vấn một lần và không bận tâm đến việc tạo chế độ xem tạm thời
- Sử dụng cú pháp `data_format.`path to file`` thay cho tên bảng

```
df = spark.read.parquet("users.parquet")
df.createOrReplaceTempView("users")
spark.sql(""" SELECT * FROM users """).show()
```

Input

```
spark \
 .sql(""" SELECT * FROM parquet.`users.parquet`""").show()
```

Input

Output

# Hiệu suất sử dụng truy vấn SQL

- Spark SQL chuyển đổi cả truy vấn SQL trực tiếp và phương thức SparkSession thành mã tương đương bằng trình tối ưu hóa Catalyst - không có sự khác biệt về hiệu suất

|                                                                                    |               |
|------------------------------------------------------------------------------------|---------------|
| spark.sql(""" SELECT * from authors WHERE first_name LIKE 'A%' LIMIT 3 """).show() | <b>Input</b>  |
| +-----+-----+-----+-----+-----+-----+                                              | <b>Output</b> |
| id first_name last_name email  birthdate added                                     |               |
| +-----+-----+-----+-----+-----+-----+                                              |               |
| 29  America Marquardt ulockman@example.org 2018-11-21 2010-10-03 14:12:...         |               |
| 31  Alvis  Crist kennith25@example... 1973-05-02 2003-07-11 12:52:...              |               |
| 32  Adele  Schultz  elias04@example.com 1970-04-05 1973-12-06 15:09:...            |               |
| +-----+-----+-----+-----+-----+-----+                                              |               |
| df = spark.read.table("authors")                                                   | <b>Input</b>  |
| df.where(df.first_name.startswith('A')).limit(3).show()                            |               |
| +-----+-----+-----+-----+-----+-----+                                              | <b>Output</b> |
| id first_name last_name email  birthdate added                                     |               |
| +-----+-----+-----+-----+-----+-----+                                              |               |
| 29  America Marquardt ulockman@example.org 2018-11-21 2010-10-03 14:12:...         |               |
| 31  Alvis  Crist kennith25@example... 1973-05-02 2003-07-11 12:52:...              |               |
| 32  Adele  Schultz  elias04@example.com 1970-04-05 1973-12-06 15:09:...            |               |
| +-----+-----+-----+-----+-----+-----+                                              |               |

## Sử dụng SparkSession.sql cho DDL và DML

- SparkSession.sql có thể chấp nhận các truy vấn DDL và DML, ngoài các truy vấn SQL
- Lưu ý rằng ngoài các bảng Hive, Spark liệt kê các chế độ xem tạm thời mà chúng tôi đã tạo trong các trang trình bày trước

```
spark.sql("show tables").show()
```

Input

| database | tableName        | isTemporary |
|----------|------------------|-------------|
| default  | authors          | false       |
| default  | authors_external | false       |
| default  | authors_managed  | false       |
| default  | posts            | false       |
|          | staff            | true        |
|          | users            | true        |

Output

# Tạo Hive Table với SparkSession.sql

## Tạo Hive Table với SparkSession.sql

**Input**

```
spark.sql(""" CREATE EXTERNAL TABLE IF NOT EXISTS test
 (id int, name string)
 LOCATION '/user/student/test' """)
spark.sql("show tables").show()
```

**Output**

| database | tableName        | isTemporary |
|----------|------------------|-------------|
| default  | authors          | false       |
| default  | authors_external | false       |
| default  | authors_managed  | false       |
| default  | posts            | false       |
|          | staff            | true        |
|          | users            | true        |

# Sửa đổi Bảng Hive với SparkSession.sql

- Sửa đổi bảng Hive bằng câu lệnh DML từ SparkSession.sql

**Input**

```
spark.sql(""" ALTER TABLE test
 ADD COLUMNS (age int) """)
spark.sql(""" DESCRIBE test """).show()
```

**Output**

| col_name | data_type | comment |
|----------|-----------|---------|
| id       | int       | null    |
| name     | string    | null    |
| age      | int       | null    |

Bài 2.

## Xử lý dữ liệu có cấu trúc

- | 2.1. Giới thiệu về Spark SQL
- | 2.2. Các thao tác SQL Spark
- | 2.3. Tương tác RDD và DataFrames**

# Chuyển đổi RDD thành DataFrames

- Một trong những trường hợp sử dụng phổ biến nhất cho RDD Core API là đọc, làm sạch và chuẩn bị dữ liệu phi cấu trúc và bán cấu trúc cho các truy vấn hoặc máy học
- Các bước cơ bản để chuyển đổi RDD thành DataFrames bao gồm:
  - ▶ Xác định lược đồ cho DataFrame mới
  - ▶ Chuyển đổi từng hàng trong RDD thành đối tượng Row
  - ▶ Sử dụng phương thức `createDataFrame` của đối tượng `SparkSession`
- Phương pháp thực tế có thể khác nhau tùy thuộc vào:
  - ▶ Cách thông tin lược đồ nếu được trình bày
    - Không có lược đồ nào được cung cấp - hãy thử suy ra tên cột và nhập từ dữ liệu
    - Một danh sách tên cột đơn giản - kiểu dữ liệu được suy ra
    - Sử dụng `StructType` và `StructList` để chỉ định chính xác lược đồ
  - ▶ Cách mỗi hàng được định dạng:
    - Tạo các đối tượng Hàng trong Python và Scala
    - Tạo một Danh sách hoặc Tuple trong Python

## Suy luận lược đồ bằng phép phản chiếu (1/3)

- Chuyển đổi từng dòng văn bản thành danh sách các trường chuỗi

```
from pyspark.sql import Row

Load a text file and convert each line to a Row.
lines = sc.textFile("people.txt")

for line in lines.take(3):
 print("Each line is:", line)

parts = lines.map(lambda l: l.split(", "))

for part in parts.take(3):
 print("Each part is:", part)
```

Input

Output

Each line is: Michael, 29  
Each line is: Andy, 30  
Each line is: Justin, 19

Each part is:['Michael', ' 29']  
Each part is:['Andy', ' 30']  
Each part is:['Justin', ' 19']

## Suy luận lược đồ bằng phép phản chiếu (2/3)

- Chuyển đổi các hàng RDD thành các đối tượng Hàng bằng cách chuyển các cặp khóa/giá trị dưới dạng kwarg sang lớp Hàng
  - Trong Python, kwargs là danh sách đối số có độ dài biến được từ khóa
    - key = value syntax

Input

```
Convert each line into a Row object
people = parts. \
 map(lambda p:
 Row(name=p[0],
 age=int(p[1])))
for person in people.take(3):
 print("Each Row object:", person)
```

Output

```
Each Row object:
Row(name='Michael', age=29)
Each Row object:
Row(name='Andy', age=30)
Each Row object:
Row(name='Justin', age=19)
```

## Suy luận lược đồ bằng phép phản chiếu (3/3)

- Các kiểu dữ liệu được suy ra từ các đối tượng Row

| Input                                                                                                                                      | Output                                                                                                                                                                    |
|--------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre># Infer the schema and # create DataFrame  peopleDF = spark.     createDataFrame(people) peopleDF.printSchema() peopleDF.show()</pre> | <pre>root   -- name: string (nullable = true)   -- age: long (nullable = true)  +-----+     name age  +-----+---+  Michael  29      Andy  30    Justin  19  +-----+</pre> |

## Lược đồ chỉ định theo chương trình (1/2)

- | Sắp xếp các hàng chuỗi được phân cách bằng dấu cách thành các trường
  - ▶ Chúng tôi sử dụng một tuple để lưu trữ dữ liệu từng cột
  - ▶ Truyền tuổi thành kiểu dữ liệu số nguyên

```
Import data types
from pyspark.sql.types import *

Load a text file and convert each line to a Row.
lines = sc.textFile("people.txt")
parts = lines.map(lambda l: l.split(","))

Each line is converted to a tuple.
people = parts.map(lambda p: (p[0], int(p[1])))
for person in people.take(3):
 print("Each part is", person)
```

Input

Output

```
Each part is ('Michael', 29)
Each part is ('Andy', 30)
Each part is ('Justin', 19)
```

## Lược đồ chỉ định theo chương trình (2/2)

### I Tạo giản đồ với StructType(<Collection of StructFields>)

- ▶ Mỗi cột là một StructField(<column name>, <column datatype>, <can be null?>)

| Input                                                                                                                                                                                                                                                                                                                                                                             | Output                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre># Import SQL data types from pyspark.sql.types import *  # Create the Schema with StructType and # Collection of StructFields schema = StructType([     StructField("name", StringType(), True),     StructField("age", IntegerType(), True)])  # Apply the schema to the RDD. peopleDF = spark.createDataFrame(people, schema) peopleDF.printSchema() peopleDF.show()</pre> | <pre>root  -- name: string (nullable = true)  -- age: integer (nullable = true)  +-----+     name age  +-----+  Michael  29      Andy  30    Justin  19  +-----+</pre> |

# Chuyển đổi DataFrame thành RDD

## Đôi khi cần phải chuyển đổi DataFrame thành RDD

- ▶ Ví dụ: sử dụng thuật toán thu nhỏ bản đồ RDD hiện có trên DataFrame
- ▶ Sử dụng phương thức .rdd DataFrame

| Input                                                                                                                                                            | Output                                                                                                                                                                                                                                            |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>peopleDF.printSchema()<br/><br/>peopleRDD = peopleDF.rdd<br/>for person in peopleRDD.take(3):<br/>    print("Each row of RDD:",<br/>          person)</pre> | <pre>root<br/> -- name: string (nullable = true)<br/> -- age: integer (nullable = true)<br/><br/>Each row of RDD: Row(name='Michael', age=29)<br/>Each row of RDD: Row(name='Andy', age=30)<br/>Each row of RDD: Row(name='Justin', age=19)</pre> |

# Làm việc với các đối tượng hàng

- Chuyển đổi DataFrame thành RDD bằng phương thức .row tạo ra các đối tượng Row
- Mỗi cột là một thuộc tính đối tượng Hàng
  - Truy cập từng cột bằng ký hiệu row.column\_name

Input

```
peopleTupleRDD = peopleRDD \
 .map(lambda row: (row.name, row.age))

for item in peopleTupleRDD.take(3):
 print("Each row is now a tuple:", item)
```

Output

```
Each row is now a tuple: ('Michael', 29)
Each row is now a tuple: ('Andy', 30)
Each row is now a tuple: ('Justin', 19)
```

## [Lab6] Làm việc với API DataFrame



## [Lab7]

# Làm việc với Hive từ Spark



## [Lab8] Biến đổi Spark SQL



## [Lab9] Làm việc với Spark SQL



## [Lab10] Chuyển đổi RDD thành DataFrames



Bài 3

# Xử lý dữ liệu truyền phát

Xử lý Big Data với Apache Spark

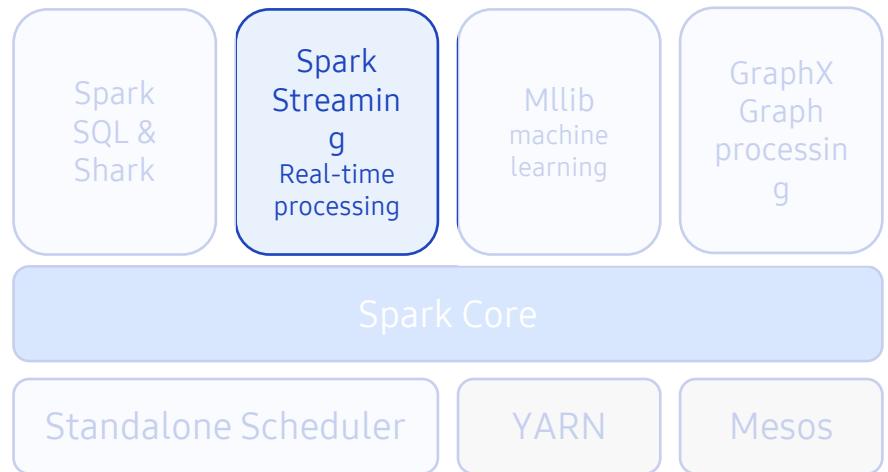
Bài 3

# Xử lý dữ liệu truyền phát

- | **3.1. Giới thiệu về Spark Streaming**
- | 3.2. Làm việc với dữ liệu truyền không có cấu trúc
- | 3.3. Làm việc với dữ liệu truyền có cấu trúc

# Spark Streaming là gì

- Apache Spark Streaming là một công cụ xử lý luồng có khả năng mở rộng, chịu lỗi và thông lượng cao
- Apache Spark Streaming bao gồm hai API riêng biệt
  - ▶ Spark Streaming API là một phần mở rộng của Core API và được sử dụng để xử lý dữ liệu truyền phát phi cấu trúc
  - ▶ API truyền trực tuyến có cấu trúc là phần mở rộng của API DataFrame và được sử dụng để xử lý dữ liệu có cấu trúc
- Catalyst Optimizer (Trình tối ưu hóa chất xúc tác)
  - ▶ Truyền có cấu trúc được xây dựng dựa trên các API cấp cao hơn và tận dụng Catalyst Optimizer.
  - ▶ Spark Streaming API được xây dựng trực tiếp trên công cụ Core API và không thể sử dụng Catalyst Optimizer.



# Spark Streaming so với Truyền có cấu trúc

## I Spark Streaming API

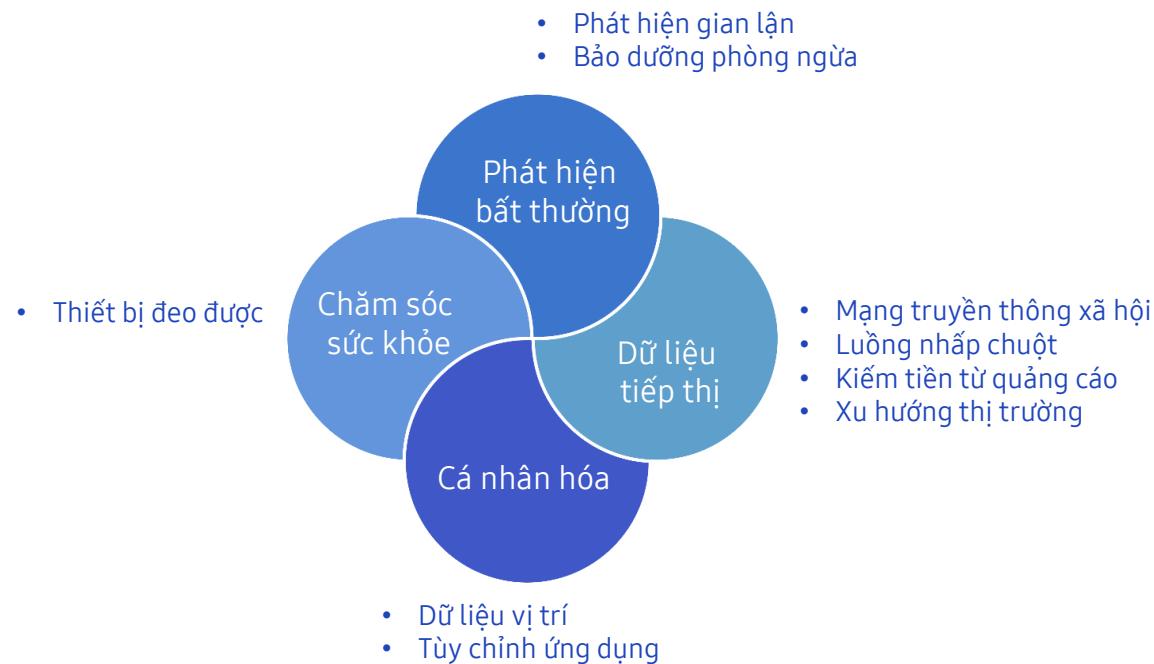
- ▶ Chủ yếu để xử lý dữ liệu phi cấu trúc hoặc bán cấu trúc
- ▶ Dựa trên API lõi RDD
  - Ngữ nghĩa dựa trên MapReduce
- ▶ API cấp thấp hơn cấp lõi
- ▶ Tối ưu hóa hạn chế
- ▶ Xử lý một lần và chỉ một lần

## I API truyền phát có cấu trúc

- ▶ Chủ yếu để xử lý dữ liệu có cấu trúc
- ▶ Dựa trên API DataFrame
  - Ngữ nghĩa dựa trên SQL
- ▶ API cấp cao hơn
- ▶ Sử dụng Catalyst Optimizer để tối ưu hóa hiệu quả cao
- ▶ Đảm bảo tính nhất quán giữa truyền phát và truy vấn tĩnh

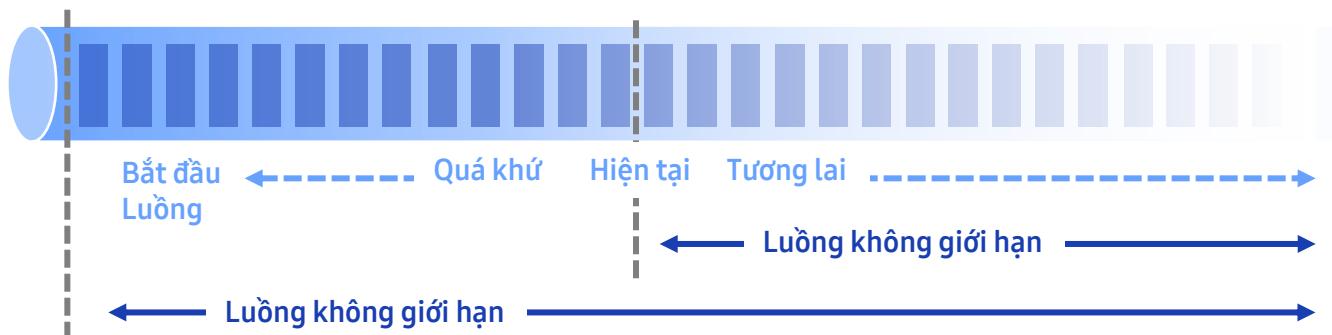
# Trường hợp sử dụng Spark Streaming

| Có nhiều nguồn dữ liệu phát trực tuyến được tạo và phải được xử lý theo thời gian thực



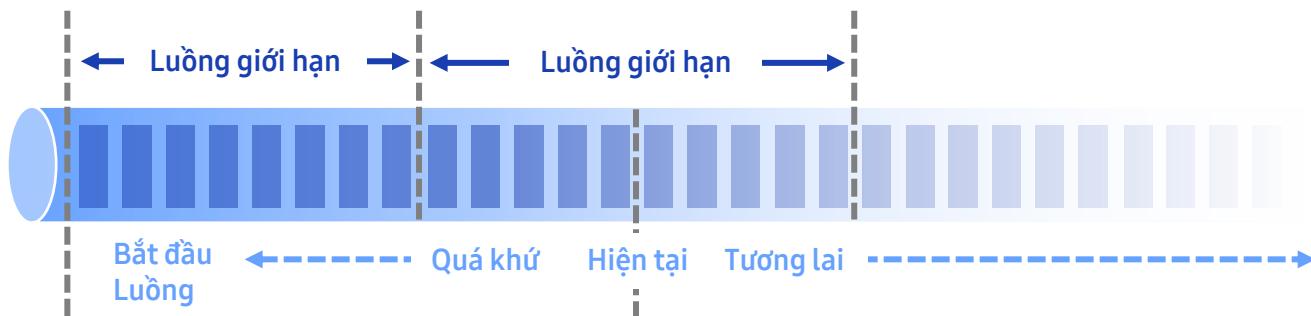
# Luồng dữ liệu không giới hạn - Đánh giá

- Luồng dữ liệu không giới hạn có một số điểm bắt đầu
  - ▶ Có thể rất xa trong quá khứ và không có bất kỳ hậu quả nào trong việc xử lý dữ liệu đến hiện tại
- Luồng dữ liệu đến không có kết thúc xác định
  - ▶ Luồng dữ liệu đến dự kiến sẽ không kết thúc bất cứ lúc nào
- Luồng không giới hạn phải được nhập theo thời gian thực và theo thứ tự cụ thể
  - ▶ Không thể đợi cho đến khi toàn bộ luồng được đọc - có thể không bao giờ kết thúc
  - ▶ Các sự kiện phải được nhập và xử lý tại thời điểm xảy ra sự kiện



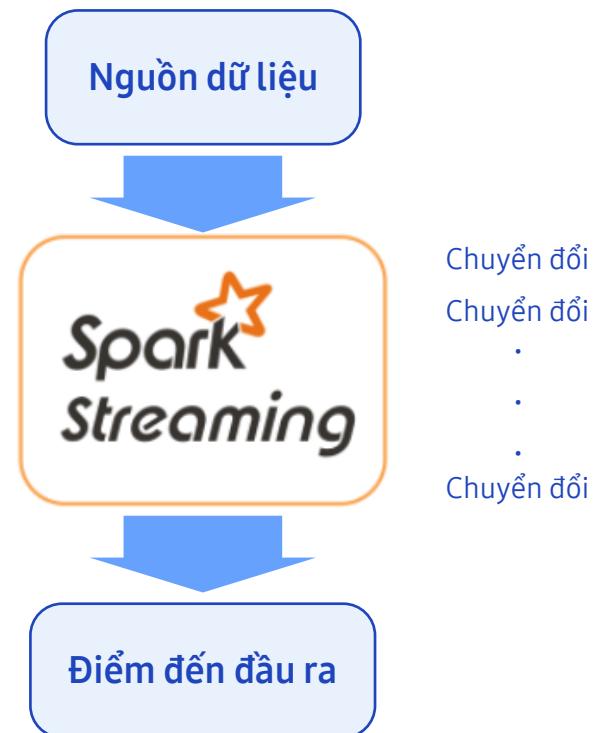
## Luồng dữ liệu có giới hạn - Đánh giá

- Luồng dữ liệu bị chặn có điểm bắt đầu và điểm kết thúc
- Có thể nhập và lưu trữ toàn bộ tập dữ liệu trước khi xử lý
- Nhập và xử lý các sự kiện theo thứ tự không nghiêm ngặt
  - ▶ Dữ liệu có thể được sắp xếp lại để phù hợp với truy vấn mong muốn vì nó đã được lưu.



# Phát triển ứng dụng truyền phát

- Hầu hết các ứng dụng phát trực tuyến đều tuân theo cùng một mẫu thực thi
  - ▶ Bước 1: Tạo DataFrame hoặc DStream từ nguồn dữ liệu đến
  - ▶ Bước 2: Xác định truy vấn và chuyển đổi trên DataFrame hoặc DStream
  - ▶ Bước 3: Lưu hoặc hiển thị kết quả



# Thao tác Spark Streaming

- I Như trong các API khác, có hai loại hoạt động trong Spark Streaming
  - ▶ Chuyển đổi áp dụng các hoạt động trên dữ liệu và tạo một đối tượng mới với các thay đổi
  - ▶ Tất cả chuyển đổi Spark thực hiện bất biến
  - ▶ Khi một Hành động được gọi, tất cả Executor sẽ trả lại kết quả của họ cho chương trình Driver
  - ▶ Tất cả các biến đổi Spark thực thi một cách lười biếng
    - Một hành động kích hoạt Spark tuân theo biểu đồ phụ thuộc dòng dõi và DAG để lập kế hoạch vật lý cuối cùng

# Bắt đầu với truyền phát

- | Chúng tôi có thể sử dụng vỏ Spark để phát triển các ứng dụng phát trực tuyến
- | Ứng dụng phát trực tuyến yêu cầu tối thiểu hai luồng hoặc được thực thi trên YARN
  - ▶ Cần có một luồng để nhập dữ liệu phát trực tuyến đến
  - ▶ Một luồng khác xử lý dữ liệu phát trực tuyến

```
% pyspark --master 'local[2]' # Specify minimum of 2 threads
% pyspark --master 'local[*]' # Use as many threads as available. Must be at least 2
% pyspark --master yarn # Run pyspark shell on YARN
```

Bài 3

# Xử lý dữ liệu truyền phát

- | 3.1. Introduction to Spark Streaming.
- | **3.2. Làm việc với dữ liệu truyền không có cấu trúc**
- | 3.3. Làm việc với dữ liệu truyền có cấu trúc

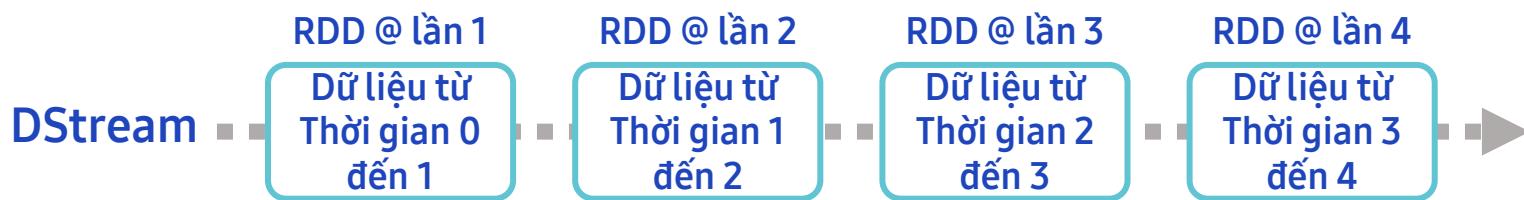
## Micro-batches

- I Spark Streaming nhập dữ liệu truyền phát và chia dữ liệu thành các đợt nhỏ hơn
  - ▶ Công cụ Spark xử lý dữ liệu ở từng cấp độ batch
  - ▶ Nhờ lại quá trình xử lý micro-batch và xử lý theo sự kiện từ các bài học trước của chúng ta
  - ▶ Spark là một hệ thống xử lý dựa trên micro-batch



## DStreams

- I Spark Streaming sử dụng trừu tượng cấp cao gọi là DStream
  - ▶ DStream là viết tắt của luồng rác
  - ▶ Quá trình tạo các micro-batch từ luồng dữ liệu liên tục tạo DStream
  - ▶ Bên trong, một DStream được đại diện bởi một chuỗi các RDD



## Spark StreamingContext

- Spark Context và SparkSession lần lượt là các đối tượng chính cho Core API và DataFrame API
- Điểm vào chính tương đương của Stream API là đối tượng StreamingContext
  - ▶ Cung cấp tất cả các chức năng của Spark Streaming API
- Không giống như Spark Context và SparkSession, StreamingContext phải được khởi tạo trong shell
  - ▶ Nhập StreamingContext
  - ▶ Truyền bối cảnh Spark hiện tại làm tham số
  - ▶ Nhập một số nguyên để chỉ định tính bằng giây, thời lượng của mỗi micro-batch

```
from pyspark.streaming import StreamingContext

Set the log level to ERROR
sc.setLogLevel("ERROR")

Create and configure a new Streaming Context
with a 1 second batch duration
ssc = StreamingContext(sc,1)
```

Input

## Khởi động Công cụ Truyền phát

- | Bước tiếp theo là chỉ định logic của ứng dụng phát trực tuyến
- | Cuối cùng, chúng ta phải khởi động Streaming Spark Engine để bắt đầu xử lý dữ liệu truyền phát
  - ▶ Sử dụng phương thức **start()** của StreamingContext
- | Khi StreamingContext được bắt đầu, nó sẽ đợi tín hiệu kết thúc dừng lại
  - ▶ Sử dụng phương thức **awaitTermination()**

Input

```
ssc = StreamingContext(sc,1)

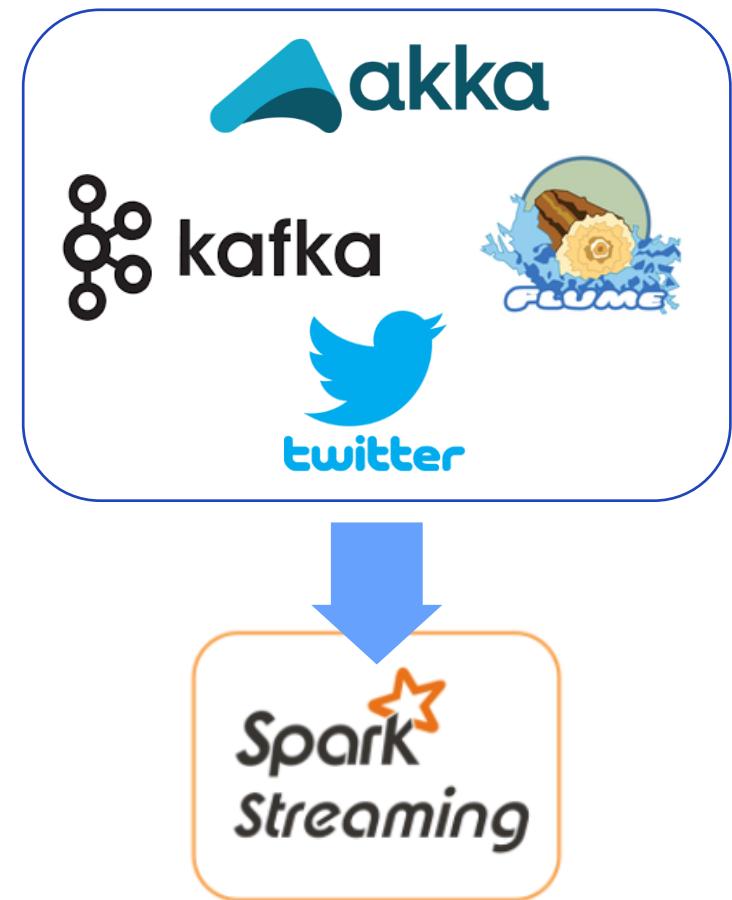
Place streaming application logic here

ssc.start()
ssc.awaitTermination()

Any code placed after the Spark Streaming engine has started will be ignored
```

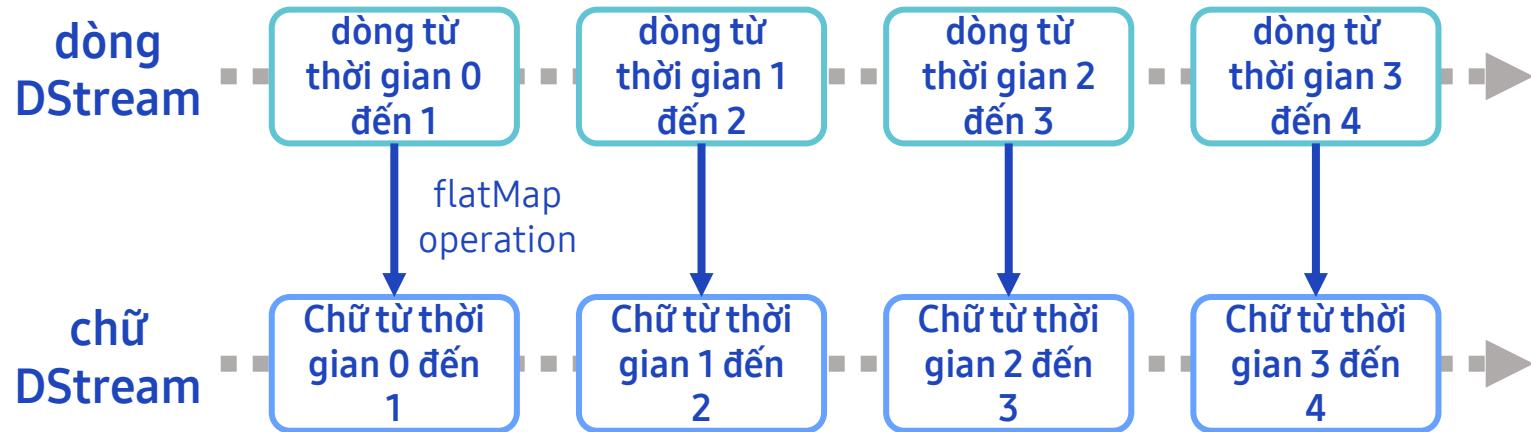
## Nguồn dữ liệu Spark Streaming

- Spark Streaming cung cấp một số nguồn dữ liệu tích hợp sẵn
- Các nguồn cơ bản có sẵn trực tiếp từ Spark Streaming API
  - ▶ Kết nối Socket
  - ▶ Hệ thống tệp - Theo dõi thư mục HDFS cho các tệp mới
- Các nguồn nâng cao yêu cầu các thư viện phụ thuộc được nhập
  - ▶ Kafka
  - ▶ AWS Kinesis
  - ▶ Flume
  - ▶ Akka Actors
  - ▶ Twitter



## Chuyển đổi DStreams

- DStream cũng được tạo thông qua chuyển đổi Dstream gốc
- Mỗi chuyển đổi bị ảnh hưởng trên chuỗi RDD cơ bản
- Ví dụ:
  - ▶ Đọc một số văn bản phát trực tuyến
  - ▶ Vector hóa các từ bằng cách sử dụng FlatMap



# Chuyển đổi trên Spark Streaming

- I DStream cơ bản là một chuỗi các RDD
  - ▶ Tất cả chuyển đổi API lõi đều có sẵn cho API DStream
- I Nhiều phép biến đổi thường được sử dụng có lối tắt trực tiếp Spark Streaming
  - ▶ **map(func)**, **flatMap(func)**, **filter(func)**, **repartition()**, **union()**
- I Nhiều phép biến đổi Cặp RDD cũng có các phím tắt trực tiếp
  - ▶ **reduce(func)**, **reduceByKey()**, **groupByKey()**, **join()**
- I Một số phép biến đổi duy nhất đối với Spark Streaming
  - ▶ Chuyển đổi trạng thái bao gồm chuyển đổi dựa trên Windows
- I Đối với bất kỳ chuyển đổi nào không có lối tắt Spark Streaming trực tiếp, hãy sử dụng transform(func)

```
The transform() passes a pointer to the current DStream to process
my_dstream.transform(lambda rdd: rdd.distinct())
```

Input

## Xem xét kỹ hơn foreachRDD (1/3)

- Nhớ lại chuyển đổi **foreachPartition** trong API lõi
  - ▶ Thường được sử dụng để tránh một hoạt động rất tốn kém, chẳng hạn như kết nối với RDBMS cho mỗi hàng
- **foreachRDD** có cách sử dụng tương tự và các nhà phát triển phải cẩn thận khi sử dụng chuyển đổi này
- Ví dụ: Đối với mỗi DStream, chúng tôi đang cố gắng kết nối với RDBMS, gửi tất cả các bản ghi trong DStream và đóng kết nối
  - ▶ Thật không may, kết nối được tạo trên Driver
  - ▶ Một đối tượng chẳng hạn như kết nối cơ sở dữ liệu không thể được tuần tự hóa và gửi tới tất cả Executor

```
def sendRecords2DB(rdd):
 # this connection is created on the Driver
 cnn = createConnection("connection string")
 rdd.foreach(lambda record: cnn.send(record))
 cnn.close()

my_dstream.foreachRDD(lambda rdd: sendRecord2DB(rdd))
```

Input

## Xem xét kỹ hơn foreachRDD (2/3)

- | Trong phiên bản trước, chúng tôi đã cố gắng gửi kết nối DB tới từng Executor nhưng điều này sẽ không thành công
- | Lần này, chúng tôi sẽ đảm bảo rằng mỗi Executor tạo kết nối DB của riêng mình
  - ▶ Từ my\_stream, chúng tôi gọi phần tử "foreach" của sendRecord2DB của my\_stream
  - ▶ sendRecord2DB và sau đó là createConnection, hiện được thực thi bởi tất cả các Executor vì nó được gọi trên mỗi thành phần hàng của my\_stream
- | Bây giờ chúng ta đã đi quá xa theo hướng khác - mọi thành phần hàng đang mở và đóng kết nối DB

```
def sendRecord2DB(record):
 cnn = createConnection("connection string")
 # this connection is created on the Executor
 cnn.send(record)
 cnn.close()

my_dstream.foreachRDD(lambda rdd: rdd.foreach(sendRecord2DB))
```

Input

## Xem xét kỹ hơn foreachRDD (3/3)

- Trong kỹ thuật, loại vấn đề này được gọi là vấn đề Goldilocks
  - ▶ Đề cập đến các vấn đề cần phải tránh các thái cực và phải tìm ra giải pháp trung庸
- Giải pháp trung gian mong muốn:
  - ▶ Mỗi Executor chịu trách nhiệm tạo một kết nối duy nhất đến cơ sở dữ liệu
  - ▶ Tất cả các bản ghi trong Phân vùng sử dụng kết nối duy nhất đó để cập nhật các mục hàng
  - ▶ Kết hợp **foreachRDD** và **foreachPartition** để thực hiện việc này
  - ▶ Sử dụng trình vòng lặp do **foreachPartition** cung cấp để quét tất cả các phần tử trong Phân vùng

```
def sendPartition2DB(iterator):
 # this connection is created on the Executor
 cnn = createConnection("connection string")
 for record in iterator:
 cnn.send(record)
 cnn.close()

my_dstream.foreachRDD(lambda rdd: rdd.foreachPartition(sendPartition2DB))
```

Input

## Thao tác đầu ra (1/3)

- I Các hành động được yêu cầu để kích hoạt các phép biến đổi Spark Streaming
  - ▶ Rất thường xuyên, các hành động trong hoạt động RDD cơ bản sẽ kích hoạt nó
- I Các hoạt động Đầu ra thường được sử dụng này cũng đóng vai trò là Hành động và kích hoạt các phép biến đổi

| Thao tác đầu ra                   | Ý nghĩa                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| pprint()                          | In mười phần tử đầu tiên của mỗi lô dữ liệu trong DStream trên nút trình điều khiển đang chạy ứng dụng phát trực tuyến. Điều này rất hữu ích cho việc phát triển và gỡ lỗi.                                                                                                                                                                                                                                        |
| saveAsTextFiles(prefix, [suffix]) | Lưu nội dung của DStream này dưới dạng tệp văn bản. Tên tệp ở mỗi khoảng thời gian theo đợt được tạo dựa trên tiền tố và hậu tố: "prefix-TIME_IN_MS[.suffix]".                                                                                                                                                                                                                                                     |
| foreachRDD(func)                  | Toán tử đầu ra chung nhất áp dụng một hàm, func, cho mỗi RDD được tạo từ luồng. Chức năng này sẽ đẩy dữ liệu trong mỗi RDD sang hệ thống bên ngoài, chẳng hạn như lưu RDD vào tệp hoặc ghi dữ liệu đó qua mạng vào cơ sở dữ liệu. Lưu ý rằng hàm func được thực thi trong quy trình trình điều khiển đang chạy ứng dụng truyền phát và thường sẽ có các tác vụ RDD trong đó sẽ buộc tính toán các RDD truyền phát. |

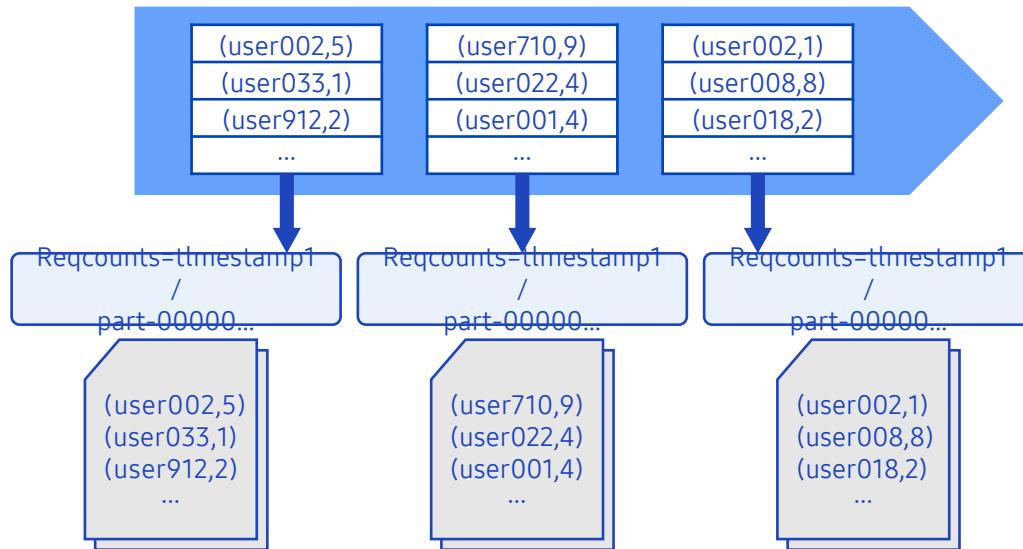
## Thao tác đầu ra (2/3)

- | Sử dụng Thao tác đầu ra để lưu dữ liệu đã xử lý vào hệ thống tệp hoặc RDBMS
- | Các hoạt động Đầu ra này cho phép lưu đầu ra ở các định dạng khác
  - ▶ Các thao tác này không khả dụng trong phiên bản Python của API

| Thao tác đầu ra                     | Ý nghĩa                                                                                                                                                                                                  |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| saveAsObjectFiles(prefix, [suffix]) | Lưu nội dung của DStream này dưới dạng SequenceFiles của các đối tượng Java được tuân tự hóa. Tên tệp ở mỗi khoảng thời gian theo đợt được tạo dựa trên tiền tố và hậu tố: "prefix-TIME_IN_MS[.suffix]". |
| saveAsHadoopFiles(prefix, [suffix]) | Lưu nội dung của DStream này dưới dạng tệp Hadoop. Tên tệp ở mỗi khoảng thời gian theo đợt được tạo dựa trên tiền tố và hậu tố: "prefix-TIME_IN_MS[.suffix]".                                            |

## Thao tác đầu ra (3/3)

- Các hoạt động đầu ra tạo tên tệp dựa trên tiền tố, dấu thời gian và hậu tố tùy chọn
- Vì Spark là một công cụ xử lý song song phân tán, nên mỗi tên tệp sẽ trở thành một thư mục trong HDFS
  - Mỗi Executor lưu riêng phân vùng dữ liệu của nó



## Chuyển đổi có trạng thái

- Spark Streaming cung cấp các chuyển đổi trạng thái để bảo toàn thông tin trên các DStream
  - ▶ `UpdateStateByKey` duy trì trạng thái trong suốt thời gian tồn tại của StreamingContext
  - ▶ Chuyển đổi cửa sổ cũng duy trì trạng thái trên các DStream, nhưng chỉ trên một cửa sổ xác định
- Vì `UpdateStateByKey` phải lưu giữ thông tin trạng thái trong suốt thời gian tồn tại của Bối cảnh truyền phát, nên hiệu suất của nó có thể bị ảnh hưởng rất nhiều
  - ▶ Cần thận trọng khi sử dụng
- Chuyển đổi cửa sổ là thay thế trong đó trạng thái được bảo toàn nhưng trong thời gian ngắn hơn
  - ▶ Rất hiếm khi trường hợp sử dụng của chúng tôi yêu cầu trạng thái đó được duy trì mãi mãi
    - Windows cho phép chúng tôi xác định một cửa sổ thời gian mà chúng tôi sẽ theo dõi trạng thái
- Cú pháp
  - ▶ `my_dstream.updateStateByKey(function)`
  - ▶ `my_dstream.reduceByKeyAndWindow(function, <window information>)`

## Chuyển đổi updateStateByKey (1/3)

- **updateStateByKey** hoạt động với DStream với Cặp RDD trong RDD tuần tự cơ bản
- Để sử dụng **updateStateByKey** để theo dõi trạng thái liên tục, cần có hai điều
  - ▶ Trạng thái của kiểu dữ liệu tùy ý để theo dõi trạng thái hiện tại
  - ▶ Một chức năng được xác định để cập nhật trạng thái

Input

```
def updateFunction(newValues, runningCount):
 if runningCount is None: return sum(newValues)
 else: return sum(newValues) + runningCount
```

## Chuyển đổi updateStateByKey (2/3)

- I Spark Streaming sẽ gọi chức năng cập nhật cho tất cả các khóa hiện có
  - ▶ Hàm được gọi ngay cả khi DStream hiện tại không có dữ liệu mới cho khóa đó
  - ▶ Điều này có nghĩa là khi số lượng khóa tăng lên trong suốt thời gian tồn tại của ứng dụng phát trực tuyến, mỗi đợt tiếp theo hoặc DStream sẽ phải lặp lại số lượng khóa ngày càng tăng
- I Để chống lỗi, Spark cần lưu trạng thái vào kho bền định kỳ
  - ▶ Sử dụng **StreamingContext.checkpoint(<checkpoint directory>)**
  - ▶ Đây là một thư mục HDFS

```
from pyspark.streaming import StreamingContext
ssc = StreamingContext(sc, 1)

ssc.checkpoint("checkpoint")

read streaming data from socket
hostname = "localhost"
port = 44444
lines = ssc.socketTextStream(hostname, port)
```

Input

## Chuyển đổi updateStateByKey (3/3)

- Tương tự như các phép biến đổi tập hợp Pair RDD khác, trước tiên, **updateStateByKey** nhóm các RDD bên dưới theo cùng một khóa
- Các giá trị của các khóa phù hợp được thu thập và chuyển đến updateFunction
- Khóa được sử dụng để truy xuất bất kỳ giá trị trạng thái trước đó nào
  - ▶ Nếu không tìm thấy khóa trong lịch sử trạng thái, thay vào đó, Không có gì được chuyển đến updateFunction

Input

```
lines = ssc.socketTextStream(hostname, port)
running_wc = lines \
 .flatMap(lambda line: line.split(" ")) \
 .map(lambda word: (word, 1)) \
 .updateStateByKey(updateFunction)

running_wc.pprint()

ssc.start()
ssc.awaitTermination()
```

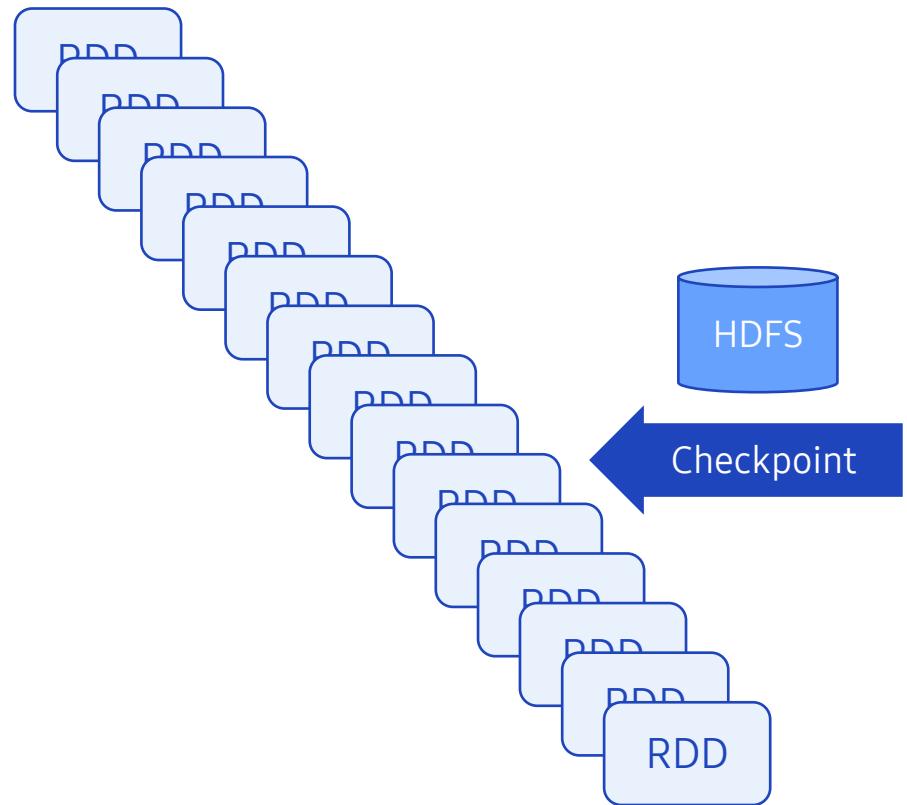
## Xem kỹ hơn phần Checkpointing

- Spark giữ một biểu đồ theo chu kỳ có hướng của các mối quan hệ phụ thuộc giữa tất cả các RDD bất biến
  - ▶ Đây là thông tin dòng dõi cho mỗi RDD
- Spark giữ và lưu trữ thông tin dòng dõi cho tất cả các RDD cho đến khi không còn cần thiết
  - ▶ Nó không còn cần thiết nếu không còn phụ thuộc vào RDD đó nữa
- Thông thường trong Spark Streaming, mỗi lô DStream được xử lý và bị lãng quên
  - ▶ Chúng tôi có thể xóa dòng cho tất cả các RDD trong DStream đã được xử lý

**Điều gì xảy ra khi chúng ta gọi updateStateByKey ?**

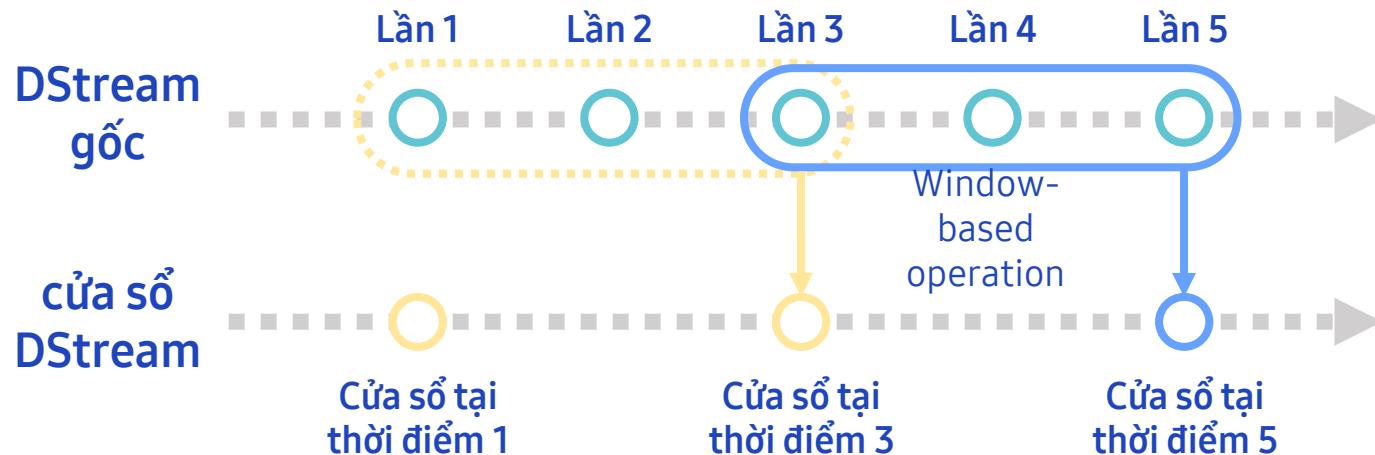
## Điểm kiểm tra để tránh dòng truyền thừa vô tận

- Dòng dõi được giữ chừng nào còn phụ thuộc
- `updateStateByKey` tạo và sự phụ thuộc không bao giờ kết thúc trong suốt quá trình quay trở lại RDD đầu tiên trong DStream đầu tiên được xử lý bởi ứng dụng Truyền phát
  - ▶ Điều này được gọi là dòng truyền thừa vô hạn
- Điểm kiểm tra định kỳ lưu trạng thái chạy bền vào HDFS
  - ▶ Điều này cho phép xóa bất kỳ dòng nào được lưu giữ trước điểm kiểm tra



## Thao tác trên cửa sổ

- I Thao tác cửa sổ cho phép áp dụng các phép biến đổi trên cửa sổ trượt
  - ▶ Tất cả các lô DStream nằm trong cửa sổ sẽ được xử lý
- I Độ dài của cửa sổ kiểm soát số lô rơi vào bên trong cửa sổ
- I Khoảng thời gian trượt kiểm soát tần suất thực hiện thao tác cửa sổ mới
- I Cả hai tham số phải là bội số của thời lượng micro-batch



## Chuyển đổi cửa sổ (1/2)

- | Các phép biến đổi Cửa sổ này xử lý DStream có chứa Cặp RDD
- | Tạo kết quả tổng hợp trên các hàng được nhóm với các khóa bằng nhau

| Chuyển đổi                                                                 | Ý nghĩa                                                                                                                                                                                                           |
|----------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| reduceByKeyAndWindow<br>(func, windowLength, slideInterval,<br>[numTasks]) | Gọi trên DStream bao gồm các bộ giá trị Key:Value. Tương tự như reduceByKey chung, tuy nhiên, quá trình giảm được thực hiện trên một cửa sổ có thời lượng windowLength được tạo mỗi slideInterval.                |
| countByValueAndWindow<br>(windowLength, slideInterval,<br>[numTasks])      | Gọi trên DStream bao gồm các bộ giá trị Key:Value. Trả về một DStream của Bộ khóa: Giá trị, trong đó Giá trị là số lượng tần suất của Khóa trên một cửa sổ có thời lượng windowLength được tạo mỗi slideInterval. |

## Chuyển đổi cửa sổ (2/2)

- Các biến đổi Cửa sổ này xử lý DStream có chứa RDD thông thường
- Tạo kết quả trên tất cả các hàng trong DStream

| Chuyển đổi                                            | Ý nghĩa                                                                                                                                                                                                                     |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| window<br>(windowLength, slideInterval)               | Trả về một DStream mới được tính toán dựa trên các đợt có cửa sổ của DStream nguồn.                                                                                                                                         |
| countByWindow<br>(windowLength, slideInterval)        | Trả về số lượng phần tử cửa sổ trượt trong luồng.                                                                                                                                                                           |
| reduceByWindow(<br>func, windowLength, slideInterval) | Trả về luồng một phần tử mới, được tạo bằng cách tổng hợp các phần tử trong luồng qua một khoảng thời gian trượt bằng func. Hàm phải có tính kết hợp và giao hoán để nó có thể được tính toán song song một cách chính xác. |

## Chuyển đổi reduceByKeyAndWindow()

- | Về bản chất, phép biến đổi này hoạt động giống như phép reduceByKey(func) bình thường
- | Sự khác biệt chính là dữ liệu mà nó tổng hợp chức năng được cung cấp
- | **reduceByKeyAndWindow(<func>, windowLength=<integer in seconds>, slideInterval=<integer in seconds>)**
  - ▶ Áp dụng func để tổng hợp Cặp RDD có cùng khóa trên windowLength giây,
  - ▶ Tạo báo cáo này mỗi slideInterval giây

Input

```
Code to instantiate Streaming context and create checkpoint directory

window_wc = \
 ssc.socketTextStream(hostname, port) \
 .flatMap(lambda line: line.split(" ")) \
 .map(lambda word: (word, 1)) \
 .reduceByKeyAndWindow(lambda v1, v2: v1+v2, 5, 2)

pprint(window_wc)

Code to start streaming context and await termination
```

# Chuyển đổi countByWindow

- Đếm số hàng trong Window
- Màn hình đầu ra hiển thị một phép tính mới cứ sau 2 giây

Input

```
Create streaming context with batch duration of
1 second

filter lines with Alice and count in
Window size of 5 sec, every 2 sec

alice_window = lines \
 .map(lambda line: line.upper()) \
 .filter(lambda line: "ALICE" in line) \
 .countByWindow(5, 2)

alice_window.pprint()

ssc.start()
ssc.awaitTermination()
```

Output

```
Time : 2021-09-04 14:27:58
12

Time : 2021-09-04 14:28:00
41

Time : 2021-09-04 14:28:02
```

Bài 3

# Xử lý dữ liệu truyền phát

- | 3.1. Giới thiệu về Spark Streaming
- | 3.2. Làm việc với dữ liệu truyền không có cấu trúc
- | **3.3. Làm việc với dữ liệu truyền có cấu trúc**

# Spark Streaming có cấu trúc

- I Spark's Structured Streaming sử dụng cùng một công cụ Spark SQL
  - ▶ Cho phép các lập trình viên và tạo các truy vấn và ứng dụng phát trực tuyến như thể trên dữ liệu tĩnh
  - ▶ Truyền có cấu trúc đảm nhiệm việc chạy lại logic tăng dần trên dữ liệu mới khi chúng đến để cập nhật kết quả cuối cùng
  - ▶ Vì công cụ cơ bản là cùng một công cụ Spark SQL, nên các nhà phát triển có thể sử dụng cùng một API DataFrame
  - ▶ Sử dụng cùng một đối tượng SparkSession và các phương thức cũng như toán tử của nó
- I Ngoài API DataFrame cơ bản, Truyền có cấu trúc cung cấp chức năng bổ sung hướng đến việc xử lý dữ liệu truyền trực tuyến
  - ▶ Cửa sổ sự kiện và hình mờ để xử lý dữ liệu đến muộn
  - ▶ Kết hợp dữ liệu tĩnh và truyền trực tuyến - Kiến trúc Lambda tích hợp hiệu quả

## Hãy làm quen với một ví dụ (1/4)

### I Đọc dữ liệu truyền phát bằng phương thức `readStream()`

- ▶ Tương tự như phương thức `read()` của `SparkSession`
- ▶ Sử dụng `format()` để chỉ định loại nguồn
- ▶ Sử dụng `option()` để đặt tùy chọn cho loại nguồn đã chọn

Input

```
Create DataFrame representing the stream of input lines from
Socket connection to localhost at port 44444

lines = spark \
 .readStream \
 .format("socket") \
 .option("host", "localhost") \
 .option("port", 44444) \
 .load()
```

## Hãy làm quen với một ví dụ (2/4)

- Tạo logic ứng dụng bằng cách sử dụng các phép biến đổi và truy vấn
- Đặt hành động để kích hoạt logic chuyển đổi

Input

```
import the spark.sql functions to create column expression
from pyspark.sql.functions import *

Split the lines into words
words = lines.select(
 explode(
 split(lines.value, " ")
).alias("word")
)

Generate running word count
wordCounts = words.groupBy("word").count()
```

## Hãy làm quen với một ví dụ (3/4)

I **writeStream()** là bộ đếm Truyền có cấu trúc đối với phương thức **readstream()**

- ▶ Trong SparkSession, chúng tôi đã có spark.read() và spark.write
- ▶ Chúng tôi đặt chế độ đầu ra để tạo đầu ra cho toàn bộ luồng dữ liệu cho đến nay
- ▶ Chọn và xuất định dạng - ở đây chúng tôi đang gửi đến bảng điều khiển
- ▶ Như chúng tôi đã làm trong Spark Streaming, hãy khởi động công cụ Truyền phát và chờ kết thúc

Input

```
Start running the query that prints the running counts to the console
query = wordCounts \
 .writeStream \
 .outputMode("complete") \
 .format("console") \
 .start()

query.awaitTermination()
```

## Hãy làm quen với một ví dụ (4/4)

- | Quan sát đầu ra cho chúng tôi cái nhìn sâu sắc về cách hoạt động của các micro-batch Truyền có cấu trúc (Structured Streaming)
    - | Như trong Spark Streaming, Structured Streaming cũng xử lý dữ liệu theo micro-batch
      - ▶ Spark Streaming đặt khoảng thời gian cho mỗi đợt
      - ▶ Có thể đặt khoảng thời gian bằng phương thức **trigger()**
      - ▶ Trong phương pháp mặc định, một lô mới bao gồm tất cả các hàng đã đến và được tích lũy kể từ khi xử lý lô cuối cùng
  - | Trong Batch 0, thường không có đầu ra
    - ▶ Chưa có bất kỳ dữ liệu tích lũy nào kể từ khi chúng tôi mới bắt đầu
  - | Trong Batch 1, chúng ta thấy kết quả đầu ra
    - ▶ Điều này dựa trên các hàng mới nhận được và tích lũy kể từ khi "kết thúc" Đợt 0

```
Batch: 0

+---+---+
|word|count|
+---+---+
+---+---+

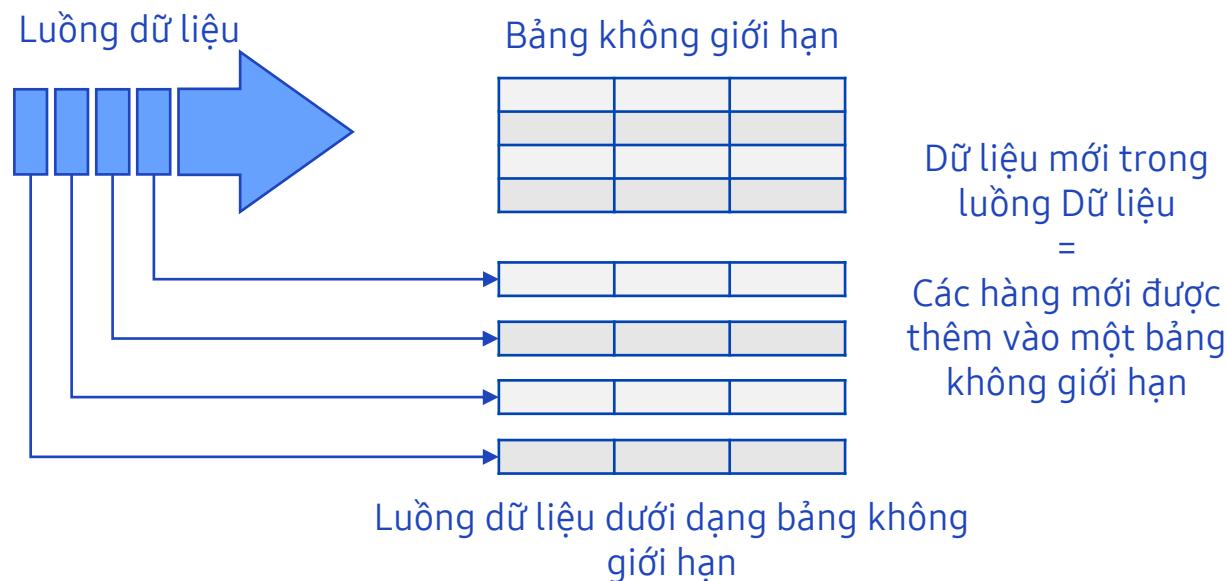
```

Batch: 1

```
+-----+-----+
| word|count|
+-----+-----+
enough;	2
corners:	1
spoke;	1
waters	1
high:	3
knot!"	1
pack,	1
youth,	3
sisters,"	1
still	12
1.F.3,	3
```

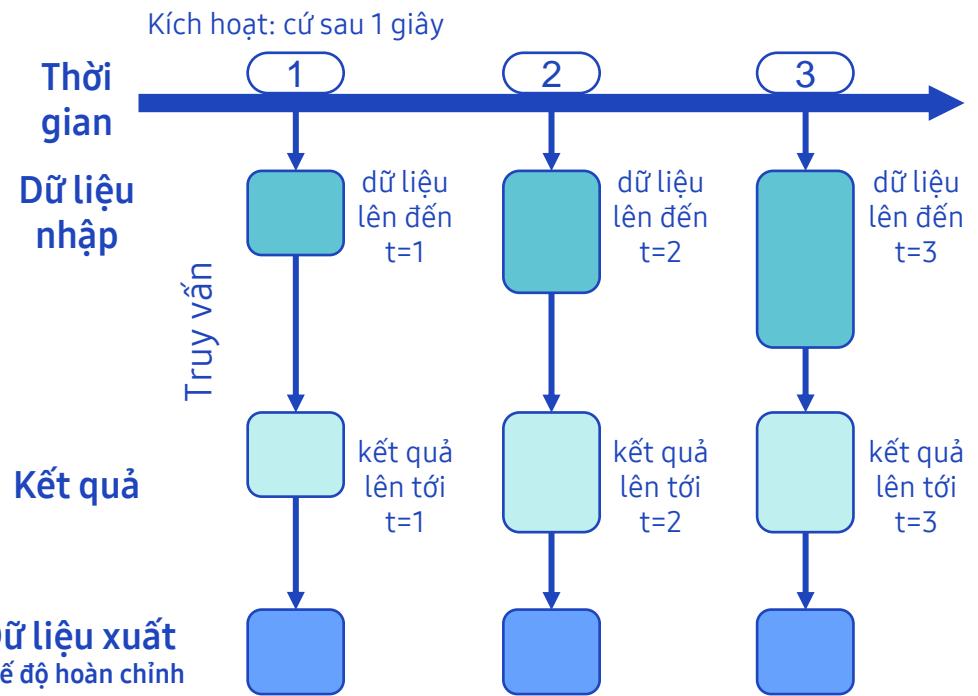
## Cấu tạo dữ liệu truyền trực tuyến có cấu trúc

- I Truyền có cấu trúc sắp xếp dữ liệu dưới dạng bảng không giới hạn
  - ▶ Dữ liệu liên tục tích lũy và được thêm vào bảng
- I Trong API DataFrame, mỗi khung dữ liệu được xem dưới dạng bảng giới hạn



# Mô hình lập trình

- Như trong Spark Streaming, Structured Streaming cũng xử lý dữ liệu theo micro-batch
  - Như chúng ta đã thấy trong mã ví dụ, chế độ hàng loạt mặc định là xử lý tất cả các hàng được tích lũy kể từ khi bắt đầu đợt trước
  - Spark Streaming cũng cho phép đặt khoảng thời gian cho từng đợt
  - Có thể đặt khoảng thời gian bằng cách sử dụng phương thức `trigger()`



## Chế độ đầu ra

Đầu ra được định nghĩa là những gì được lưu trữ vào bộ nhớ ngoài

- ▶ Chế độ đầu ra mặc định là chế độ Append

| Chế độ đầu ra | Ý nghĩa                                                                                                                                                                                |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Complete      | Toàn bộ kết quả được lưu vào bộ nhớ ngoài                                                                                                                                              |
| Append        | Chỉ các hàng mới được thêm vào kết quả kể từ micro-batch cuối cùng được lưu trữ vào bộ nhớ ngoài.<br>Chế độ này chỉ có thể được sử dụng khi các hàng hiện tại không được phép thay đổi |
| Update        | Chỉ những hàng đã được cập nhật hoặc nối thêm kể từ micro-batch cuối cùng mới được lưu trữ vào bộ nhớ ngoài.                                                                           |

## Minh họa Mô hình lập trình (1/2)

- | Trong mã ví dụ của chúng tôi, chúng tôi tạo các dòng DataFrame bằng cách đọc bảng đầu vào, chuyển đổi nó thành các từ và tạo bảng kết quả trong wordCounts thông qua hành động count()
- | Các tập hợp hoạt động này giống như khi được thực hiện trên một bảng tĩnh
  - ▶ Trên thực tế, chúng ta có thể xem dữ liệu trong micro-batch hiện tại của mình dưới dạng bảng tĩnh

Input

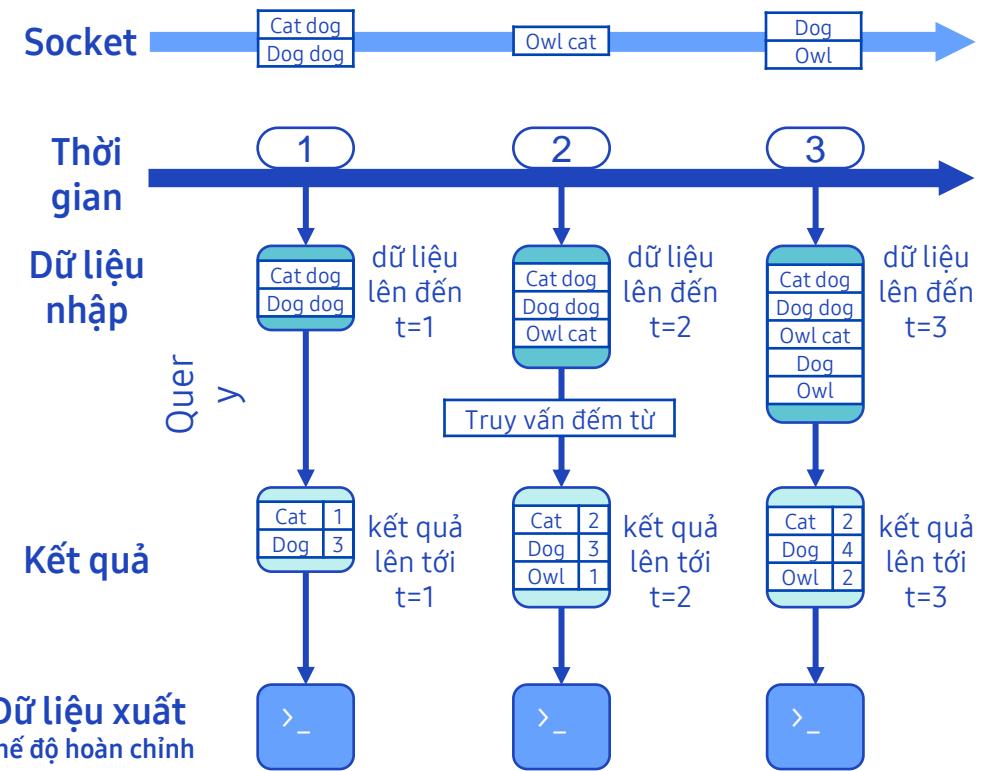
```
Read the input table
lines = spark.readStream

Split the lines into words
words =

Generate running word count
wordCounts = words.groupBy("word").count()
```

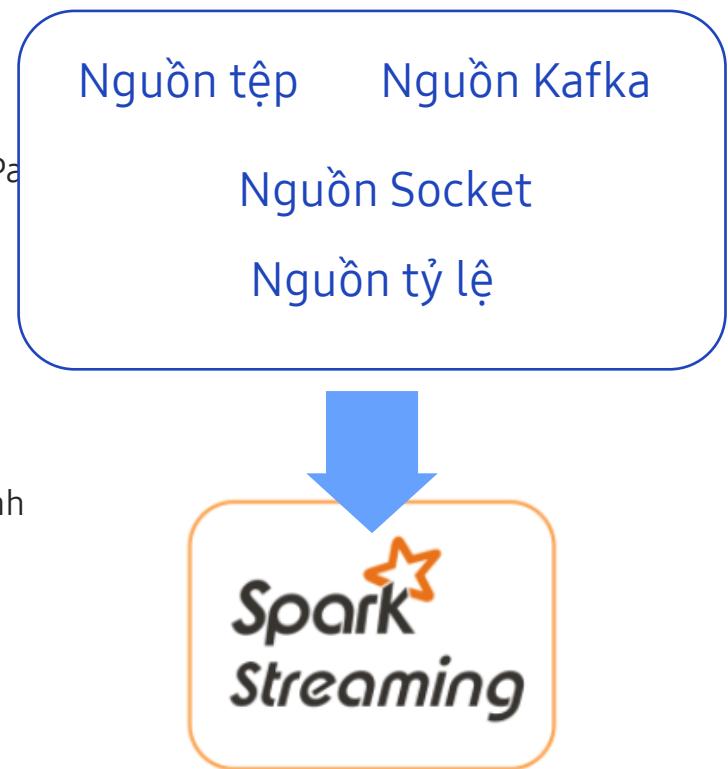
## Minh họa Mô hình lập trình (2/2)

- Điểm khác biệt của Truyền có cấu trúc là nó tiếp tục kiểm tra các hàng mới trong bảng đầu vào SAU KHI bắt đầu truy vấn
- Truyền trực tuyến cấu trúc chạy một truy vấn gia tăng và kết hợp với bảng kết quả trước đó
- Spark không lưu trong bộ nhớ toàn bộ bảng đầu vào
  - Nó xử lý từng đợt tăng dần và xóa mọi dữ liệu không cần thiết
  - Chỉ giữ nhiều trạng thái khi cần thiết
- Tuy nhiên, nhà phát triển không cần theo dõi bất kỳ điều gì trong số này



# Nguồn dữ liệu truyền phát có cấu trúc

- Nguồn dữ liệu tích hợp
- Nguồn tệp
  - ▶ Đọc tệp từ một hệ thống tệp trong một đường dẫn được chỉ định
  - ▶ Các định dạng được hỗ trợ là văn bản thuần túy, CSV, JSON, ORC và Parquet
- Nguồn Kafka
  - ▶ Nhận tin nhắn từ chủ đề Kafka
  - ▶ Chỉ được hỗ trợ trong Scala và Java trong Spark 3.x
- Nguồn socket - chủ yếu để thử nghiệm
  - ▶ Đọc văn bản được mã hóa UTF-8 từ ổ cắm Linux tại Cổng được chỉ định
- Nguồn tỷ lệ - chủ yếu để thử nghiệm
  - ▶ Tạo dữ liệu ở một hàng được chỉ định trên mỗi giây
  - ▶ Mỗi hàng chứa dấu thời gian và giá trị của kiểu dữ liệu dài



## Lược đồ cho dữ liệu được truyền phát (1/2)

- Trong API DataFrame, đảm bảo rằng chúng tôi có lược đồ chính xác cho DataFrame là một phần quan trọng trong quá trình phát triển
  - ▶ Trong những tình huống nhất định, lược đồ có thể được suy ra
- Trong Truyền phát có cấu trúc, lược đồ phải được cung cấp cho tất cả các nguồn dữ liệu dựa trên tệp
  - ▶ Lược đồ phải được chỉ định cho các nguồn dữ liệu có cấu trúc như ORC, Parquet, CSV và JSON
  - ▶ Các tệp văn bản thuần túy được đọc dưới dạng một cột và mỗi dòng mới phân định một bản ghi

Input

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

inputPath = "src-path/my.json"

Define the schema
mySchema = StructType([StructField("time", TimestampType(), True),
 StructField("action", StringType(), True)])

streamingDF = spark.readStream.schema(mySchema).json(inputPath)
```

## Lược đồ cho dữ liệu được truyền phát (2/2)

### I Kafka có một lược đồ cố định

- ▶ Khóa / nhị phân - Trường hợp một số nhà sản xuất tùy chỉnh chèn thông tin bổ sung
- ▶ Giá trị/nhị phân - Nội dung thực tế của thông báo Kafka
- ▶ Chủ đề/chuỗi - Chủ đề Kafka
- ▶ Phân vùng / số nguyên - Số phân vùng Kafka khi một chủ đề có phân vùng

Input

```
Create a Kafka streaming dataframe
kafkaDF = spark.readStream.format("kafka") . \
 option("kafka.bootstrap.servers", "localhost:9092").option("subscribe",
 "mytopic").load()

Kafka datasources have a fixed schema that includes key and value columns as binary
Use selectExpr() to execute SQL expressions without creating a temporary view
Cast the key and value column to String type
kafkaDF.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

## Các thao tác trên Truyền dữ liệu DataFrames

- I Spark cung cấp hầu hết các chuyển đổi từ API DataFrames sang Truyền có cấu trúc
  - ▶ Các thao tác SQL chưa được gõ như `select`, `where` (filter), `groupBy`, `orderBy`
  - ▶ RDD như các hoạt động như bản đồ, bộ lọc, bản đồ phẳng
  - ▶ Spark thực sự coi cả hai đều là DataFrames và phân biệt chúng bằng giá trị `isStreaming()`
- I Đăng ký các khung dữ liệu phát trực tuyến và sử dụng các lệnh SQL

Input

```
Create streaming DF
streamingDF = spark.readStream
print(streamingDF.isStreaming()) # Streaming is True

Use DF transformations
countryDF = streamingDF.select("Country").where("BàisSold >= 1000")

Create temporary view and run SQL queries
streamingDF.createOrReplaceTempView("sales")
This creates another streaming dataframe
goodDF = spark.sql(""" SELECT Country FROM sales WHERE BàisSold >= 1000 """)
```

## Ví dụ Đọc CSV từ Socket

Ở đây, chúng ta trình bày cách đọc chuỗi CSV từ Linux Socket

- ▶ Vì Socket trả về một cột duy nhất nên chúng ta phải chuyển đổi nó để tạo lược đồ

Input

```
Create streaming DF from Socket source
csvDF = spark.readStream.format("socket") \
 .option("host", "localhost").option("port", "44444").load()

Text is in CSV format. Split string and use withColumn to create new columns
from pyspark.sql.functions import *
salesDF = csvDF
 .withColumn("Country", split(csvDF.value, ",") [0]) \
 .withColumn("BàisSold", split(csvDF.value, ",") [1].cast("integer")) \
 .withColumn("BàiPrice", split(csvDF.value, ",") [2].cast("integer"))

Run queries or transformations
bestDF = salesDF.select("Country").where("BàisSold >= 1000")
```

## Tổng hợp trên các khung dữ liệu truyền phát

- Trong DataFrames, `groupBy()` tạo các đối tượng GroupedData mà sau đó có thể được tổng hợp với nhiều phép biến đổi có sẵn như đếm, tối đa, tối thiểu và trình bao bọc agg(hàm tổng hợp)
  - ▶ Một cột được xác định để nhóm dữ liệu
- Tất cả các tập hợp khung dữ liệu không phát trực tuyến đều hoạt động với các khung dữ liệu truyền phát
  - ▶ Theo mặc định, kết quả dựa trên tất cả các bản ghi nhận được, cho đến đợt micro-batch hiện tại

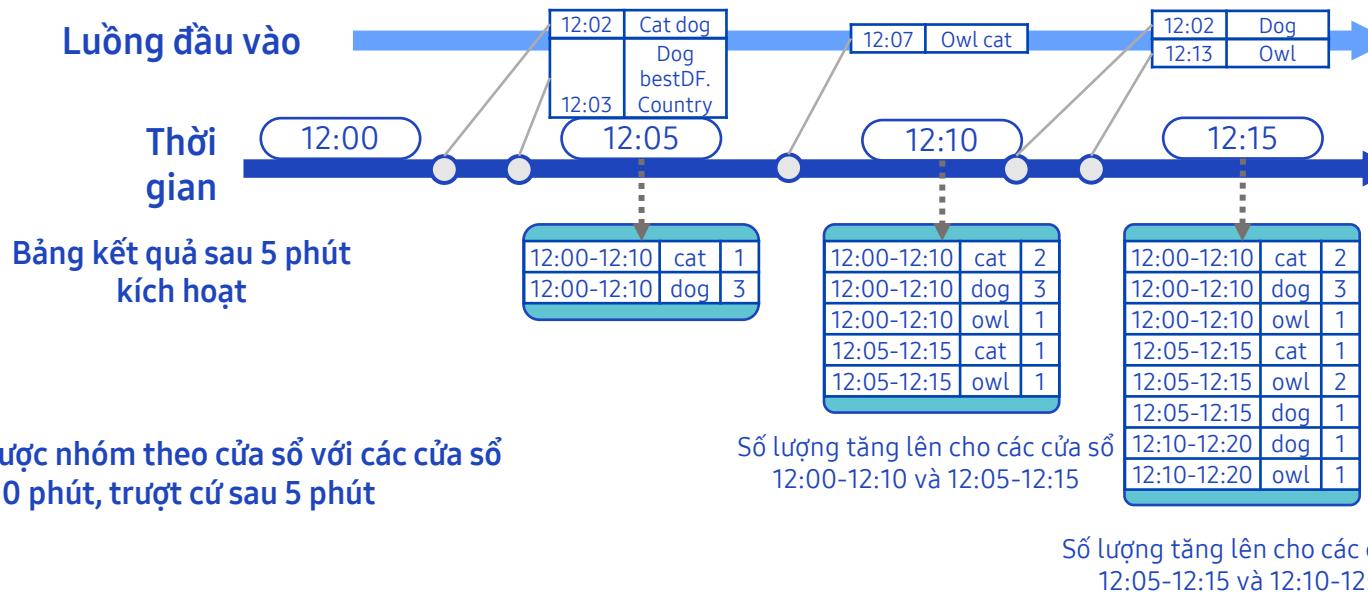
Input

```
Create streaming DF from Socket source
csvDF =
Transform and assert schema on text
salesDF = csvDF. . . .
Select countries that have transaction of 1000 Bàis sold or more
bestDF = salesDF.select("Country").where("BàisSold >= 1000")

Group by Country and count how many times such transactions occurred
bestDF.groupBy(bestDF.Country).count()
```

## Tổng hợp với Windows

- Truyền có cấu trúc cũng có thể tổng hợp qua Windows trượt dựa trên một số thời gian sự kiện
  - Từ ví dụ về số từ, hãy tưởng tượng rằng dữ liệu phát trực tuyến có thêm cột dấu thời gian
  - Sử dụng chuyển đổi `window(<timestamp column>, <window length>, <slide interval>)`



## Mã cho tập hợp cửa sổ

- | Phương thức window() tạo một chuỗi có dấu thời gian bắt đầu cửa sổ và kết thúc cửa sổ
  - ▶ "[2021-09-05 01:35:00, 2021-09-5 01:45:00]"
- | Phương thức groupBy được cung cấp hai cột để nhóm các bản ghi theo
  - ▶ Cột từ bằng nhau
  - ▶ Cột chuỗi "thời gian cửa sổ" bằng nhau

Input

```
Create streaming dataframe from (word, timestamp) source
schema = "word string, time timestamp"
wordAt = spark.readStream

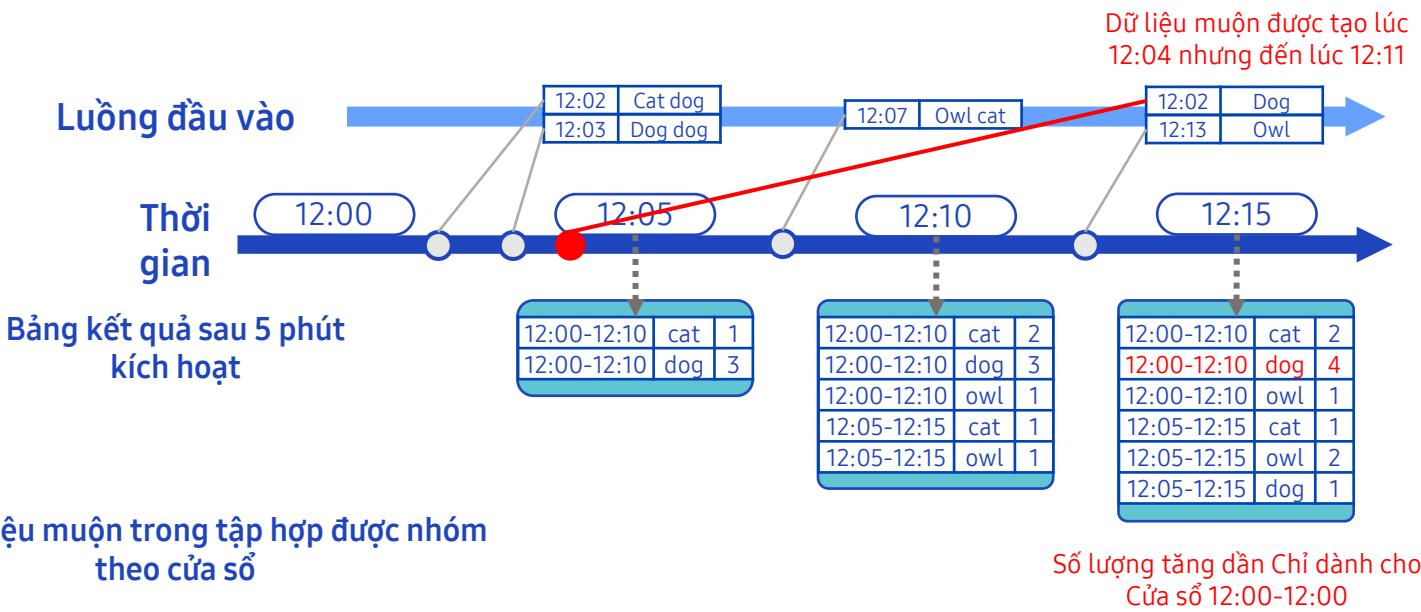
Group by word ad timestamp over a window of 10 minutes, every 5 minutes
windowWC = wordAt.groupBy(
 window(col("time", "10 minutes", "5 minutes"),
 col("word")) \
 .count()
```

## Đối phó với việc đến chậm

- Thật không may, dữ liệu không phải lúc nào cũng đến đúng giờ
  - ▶ Nhiều nguyên nhân khác nhau bao gồm máy chủ tạm thời ngừng hoạt động, lưu lượng truy cập mạng bận rộn, v.v. có thể gây ra sự chậm trễ
- Điều gì xảy ra trong tập hợp Cửa sổ khi một bản ghi có dấu thời gian trước cửa sổ hiện tại?
  - ▶ Nói cách khác, lẽ ra nó phải được tính vào một cửa sổ trước
- “Cửa sổ” cơ hội đã “suýt” trôi qua và chúng ta không thể thay đổi quá khứ hay sao?
- Spark cho phép các nhà phát triển xác định Hình mờ
  - ▶ Hình mờ xác định thời gian trễ mà một bản ghi có thể đến trước khi nó bị bỏ qua
  - ▶ Đối với các bản ghi đến trong hình mờ, các bản ghi cũ được sửa

# Giữ trạng thái cho hình mờ

- Spark sẽ lưu giữ trong bộ nhớ, trạng thái cũ mà hình mờ vẫn còn hiệu lực
  - Vẫn được phép cập nhật và sửa dữ liệu cũ



## Tập hợp cửa sổ với hình mờ

- Trước khi nhóm, hãy đặt hình mờ để cho Spark biết thời gian phải duy trì trạng thái
  - Sử dụng **withWatermark(<timestamp column on which we are grouping>, <duration of watermark>)**

Input

```
Create streaming dataframe from (word, timestamp) source
schema = "word string, time timestamp"
wordAt = spark.readStream

Group by word ad timestamp over a window of 10 minutes, every 5 minutes
Set watermark to 10 minutes for the time timestamp column
windowWC = wordAt.withWatermark("time", "10 minutes") \
 groupBy(window(col("time", "10 minutes", "5 minutes"),
 col("word")) \
 .count()
```

## [Lab11] Làm việc với API DStream



[Lab12]

Làm việc với Multi-Batch DStream API



[Lab13]

Làm việc với API truyền có cấu trúc



Bài 4

# Ứng dụng Spark và điều chỉnh hiệu suất

Xử lý Big Data với Apache Spark

Bài 4

# Ứng dụng Spark và điều chỉnh hiệu suất

## | 4.1. Ứng dụng Spark

| | 4.2. Xử lý phân tán

| | 4.3. Bền bỉ

# Viết ứng dụng Spark

- Đối với các ứng dụng sẵn sàng sản xuất, hãy tạo các tệp Python riêng biệt hoặc mã Scala đã biên dịch
- Tạo các đối tượng điểm vào theo cách thủ công
  - ▶ Tất cả các chương trình Spark đều yêu cầu đối tượng SparkContext
  - ▶ Đối tượng SparkSession là bắt buộc đối với ứng dụng dựa trên DataFrame/Bộ dữ liệu
    - Điều này bao gồm các ứng dụng Truyền có cấu trúc
  - ▶ Đối tượng StreamingContext là bắt buộc đối với các ứng dụng Spark Streaming dựa trên API lõi RDD
- Các lập trình viên trên toàn thế giới đã đặt tên mặc định cho các đối tượng này
  - ▶ `SparkContext` → `sc`
  - ▶ `SparkSession` → `spark` (Mặc dù chúng ta thực sự có thể có nhiều phiên)
  - ▶ `StreamingContext` → `ssc`
- Khi kết thúc mỗi chương trình, chúng ta phải `stop()` `SparkSession` hoặc `SparkContext`

# Tạo đối tượng SparkContext

- Tạo đối tượng SparkContext để truy cập các hành động và chuyển đổi API lõi
  - ▶ Chỉ một SparkContext cho mỗi ứng dụng
- Tạo đối tượng SparkConf và chuyển thành tham số khi khởi tạo SparkContext
  - ▶ Chèn thông tin cấu hình về ứng dụng trong SparkConf

```
First import the necessary SparkContext and SparkConf libraries
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf

Create a SparkConf() and set configurations
conf = SparkConf().setAppName(appName).setMaster(master)

Create the SparkContext
sc = SparkContext(conf=conf)
```

# Tạo SparkSession

| Sử dụng `SparkSession.builder` để tạo `SparkSession` mới

- ▶ Trình xây dựng cũng tự động tạo `SparkContext`

```
Import the SparkSession library
from pyspark.sql import SparkSession

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

# Để tất cả chúng cùng nhau

- Hãy tạo một chương trình PySpark hoạt động đầy đủ

```
import sys
from pyspark.sql import SparkSession

if __name__ == "__main__":
 if len(sys.argv) < 3:
 print(sys.stderr,
 "Usage: spark-submit sales.py <input-file> <outputfile>")
 sys.exit()

 spark = SparkSession.builder.getOrCreate()
 spark.sparkContext.setLogLevel("WARN")
 salesDF = spark.read.csv(sys.argv[1])
 countryDF = salesDF.select("Country", "BàisSold", "BàiPrice")
 countryDF.write.option("header", "true").csv(sys.argv[2])

 spark.stop()
```

## Triển khai ứng dụng lên cụm

- Sử dụng tập lệnh bin/spark-submit để gửi ứng dụng đến cụm được hỗ trợ
  - ▶ Mã Scala sẽ được biên dịch trong JAR trong khi mã Python sẽ là tệp .py
- Nếu dự án phụ thuộc vào các dự án khác, bạn sẽ phải đóng gói chúng và cung cấp cho spark-submit
- Spark-submit đơn giản cho Python

```
% spark-submit sales.py sales.csv /sales/country
```

- Spark-submit đơn giản cho Scala

```
% spark-submit --class Sales Sales.jar sales.csv /sales/country v
```

# Khởi chạy ứng dụng với các tùy chọn

- | Tập lệnh spark-submit cho phép đặt một số tùy chọn khi khởi chạy cụm
- | Một số cấu hình quan trọng hơn là:
  - ▶ --master - Loại cụm hoặc cục bộ
  - ▶ --deploy-mode - Xác định nơi chương trình Driver được khởi chạy, trên các worker node hoặc cục bộ
  - ▶ --conf <key> = <value> - Đặt bất kỳ cài đặt cấu hình có sẵn nào bằng cách sử dụng ký hiệu key=value
  - ▶ Cú pháp:

```
% ./bin/spark-submit \
 --master <master-url> \
 --deploy-mode <deploy-mode> \
 --conf <key>=<value> \
 ... # other options \
 [application-arguments]
```

# Gửi tùy chọn phím tắt tập lệnh

- | Cú pháp cấu hình cơ bản
  - ▶ --conf key=value
- | Các cấu hình thường được sử dụng có phím tắt
  - ▶ Thêm các thư viện cần thiết khác
    - --pyfiles - Tệp Python từ dự án khác được sử dụng trong ứng dụng của chúng tôi
    - --jars - Các JAR bổ sung cần có trong ứng dụng
  - ▶ Cài đặt liên quan đến YARN
    - --num-executors - Số lượng người thi hành để bắt đầu ứng dụng với
    - --executor-memory - Dung lượng bộ nhớ để phân bổ cho một Executor
    - --executor-cores - Số lượng lõi để phân bổ cho một Executor
    - --driver-memory - Dung lượng bộ nhớ để cấp phát cho Driver
    - --driver-cores - Số lượng lõi để phân bổ cho Driver
    - --queue - Gửi công việc đến hàng đợi đã chỉ định

## Tải cấu hình từ tệp

- Spark đọc từ \$SPARK\_HOME/conf/spark-defaults.conf cho các thuộc tính của bộ hệ thống.

|                        |                                            |
|------------------------|--------------------------------------------|
| spark.master           | spark://5.6.7.8:7077                       |
| spark.executor.memory  | 4g                                         |
| spark.eventLog.enabled | true                                       |
| spark.serializer       | org.apache.spark.serializer.KryoSerializer |

- Các cấu hình trong tệp cấu hình hệ thống mặc định có thể bị ghi đè bằng tệp của riêng bạn

- Sử dụng **--property-file = <path to configuration file>**
- Cú pháp :

```
% ./bin/spark-submit \
--properties-file=<my configurations file>
```

## Ưu tiên cài đặt cấu hình

- | Cấu hình có thể được đặt từ nhiều vị trí
- | Điều quan trọng là phải hiểu thứ tự ưu tiên
- | Thứ tự ưu tiên từ cao nhất đến thấp nhất:
  - ▶ Các cấu hình được đặt theo chương trình thông qua **SparkConf()**
  - ▶ Cờ được chuyển đến spark-submit
  - ▶ Tập tin cấu hình
    - Các tùy chọn trong tệp spark-defaults.conf
    - Hoặc trong tệp cấu hình của riêng bạn nếu tùy chọn --properties-file được sử dụng

# Xem cấu hình trên Spark Web UI

## Cấu hình



### Environment

#### ▼ Runtime Information

| Name          | Value                          |
|---------------|--------------------------------|
| Java Home     | /opt/jdk1.8.0_291/jre          |
| Java Version  | 1.8.0_291 (Oracle Corporation) |
| Scala Version | version 2.12.10                |

#### ▼ Spark Properties

| Name                | Value               |
|---------------------|---------------------|
| spark.app.id        | local-1630854105794 |
| spark.app.name      | PySparkShell        |
| spark.app.startTime | 1630854104103       |
| spark.driver.host   | localhost           |

Bài 4

# Ứng dụng Spark và điều chỉnh hiệu suất

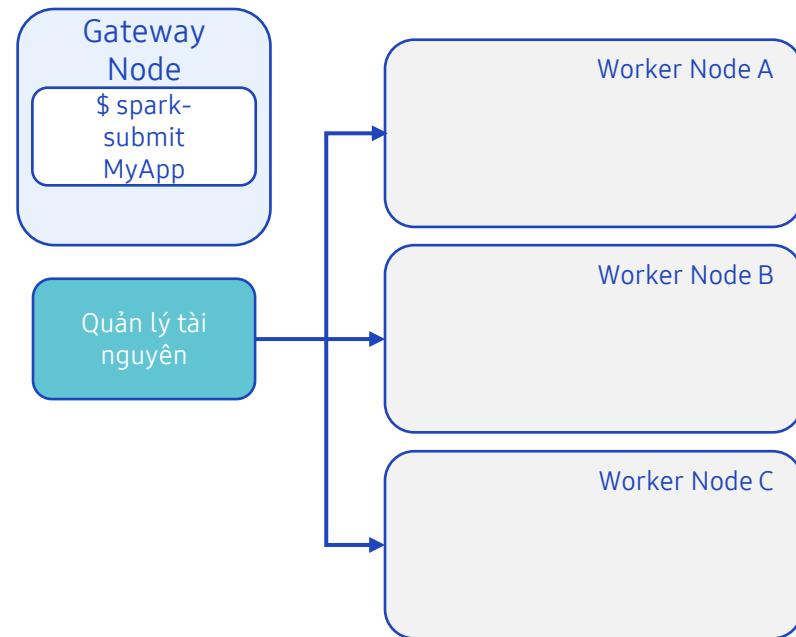
| 4.1. Ứng dụng Spark

| 4.2. Xử lý phân tán

| 4.3. Bền bỉ

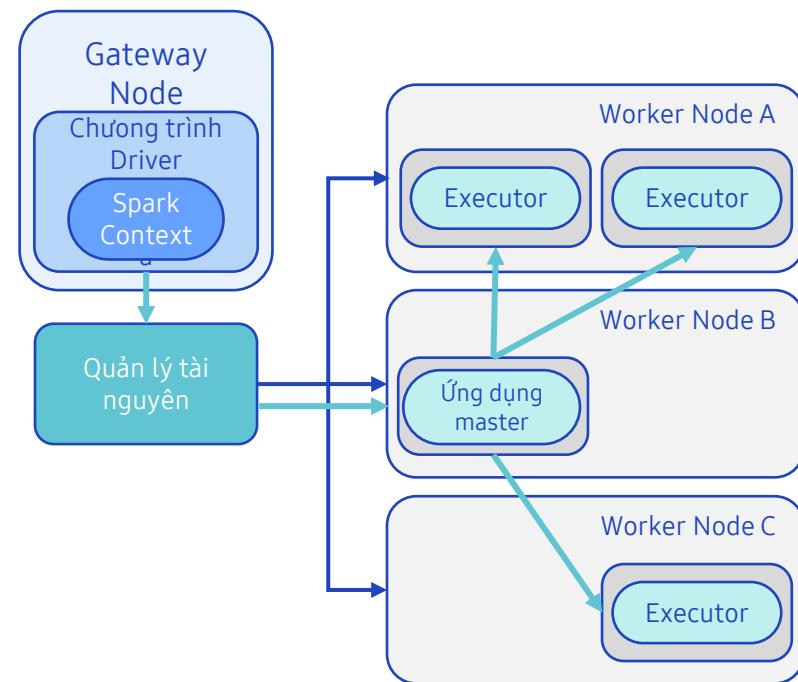
## Xử lý phân tán Spark trên Yarn (1/3)

- Spark được chạy trên một cụm khi sử dụng sản xuất
- YARN quản lý các tài nguyên được phân bổ cho Spark
- Một ứng dụng được gửi bằng tập lệnh spark-submit



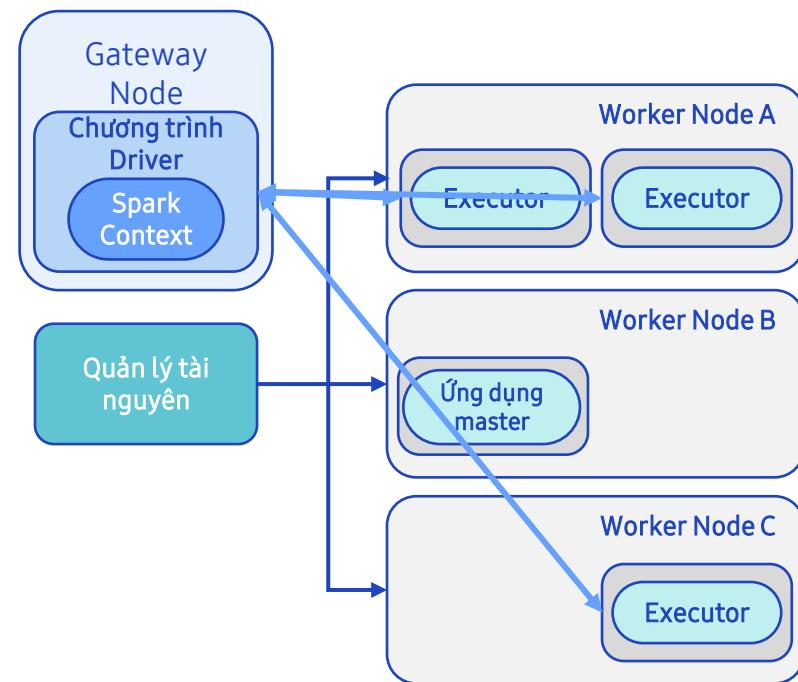
## Xử lý phân tán Spark trên Yarn (2/3)

- Trong chế độ triển khai Máy khách, chương trình Driver và ngữ cảnh Spark được tạo trên máy khách
- Ứng dụng chính được khởi chạy và tài nguyên được phân bổ trong vùng chứa
  - AM khởi chạy Executor bằng cách sử dụng các tài nguyên đó



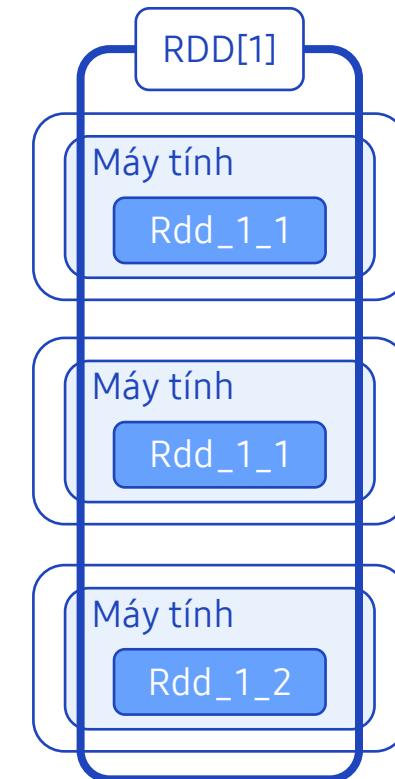
## Xử lý phân tán Spark trên Yarn (3/3)

- Chương trình Driver trên máy client phân vùng nhiệm vụ và gán cho từng Executor
- Các Executor truyền đạt lại kết quả cho chương trình Driver



## RDD phân tán

- Mỗi Executor xử lý một phần của tập dữ liệu đầy đủ
  - ▶ Tập dữ liệu một phần được gọi là Phân vùng
- Spark tự động phân vùng dữ liệu khi chúng được tải bằng quy tắc
  - ▶ Mỗi tệp ít nhất là một phân vùng riêng
  - ▶ Mỗi khối Hadoop là một phân vùng riêng biệt
- Biến đổi cũng có thể ảnh hưởng đến số lượng phân vùng
- Các lập trình viên có thể kiểm soát rõ ràng số lượng phân vùng
  - ▶ Nguyên tắc cơ bản là nhiều phân vùng hơn có nghĩa là song song hơn và hiệu suất nhanh hơn
  - ▶ Tuy nhiên, tạo quá nhiều phân vùng nhỏ sẽ thực sự hoạt động kém hơn
    - Quá nhiều chi phí vệ sinh



## Chuyển đổi phân vùng

- | repartition() và coalesce() thay đổi rõ ràng số lượng phân vùng
- | Sau khi chuyển đổi tổng hợp, có thể chỉ định số lượng phân vùng

```
rdd. groupByKey(<numPartition>)
rdd. reduceByKey (<numPartition>)
rdd.aggregateByKey (seqOp, combOp,<numPartition>)
rdd.sortByKey (<numPartition>)
rdd.join (otherDataset, <numPartition>)
```

- | Sử dụng **getNumPartitions()** để trả về số phân vùng trong RDD

```
rdd. getNumPartitions()
```

## Phân vùng lại sau khi tổng hợp (1/6)

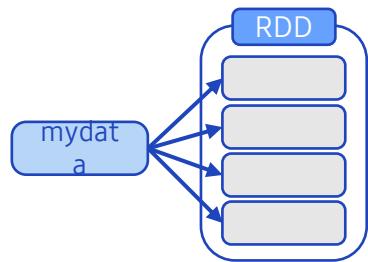
- Hãy để chúng tôi theo dõi các phân vùng khi chúng tôi thực thi đoạn mã sau

```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
```

## Phân vùng lại sau khi tổng hợp (2/6)

- Phân vùng ban đầu được tạo sau khi đọc tệp văn bản nguồn

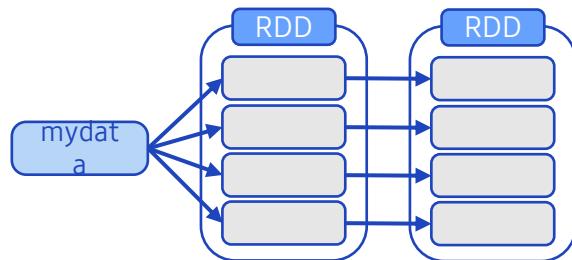
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
```



## Phân vùng lại sau khi tổng hợp (3/6)

- Chuyển đổi `.flatMap` chỉ yêu cầu dữ liệu từ cha mẹ trực tiếp của nó

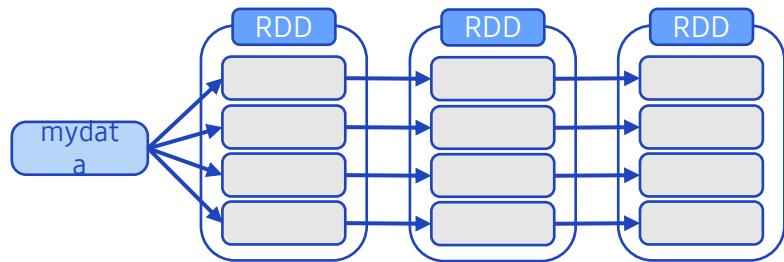
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
```



## Phân vùng lại sau khi tổng hợp (4/6)

- Chuyển đổi từng từ thành một bộ (từ, 1) chỉ yêu cầu dữ liệu từ cha mẹ trực tiếp

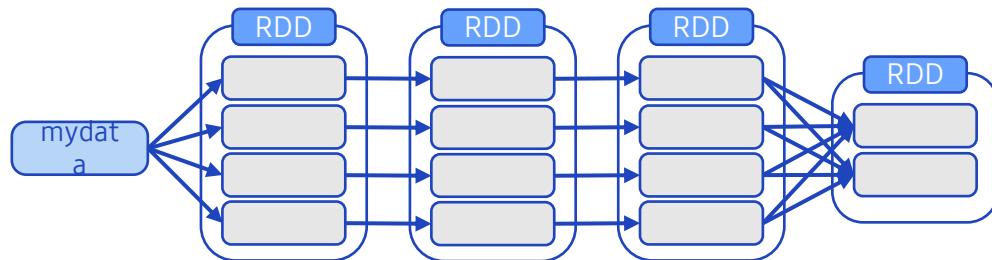
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
```



## Phân vùng lại sau khi tổng hợp (5/6)

- Để tổng hợp giá trị cho các hàng có khóa bằng nhau, tất cả các phân vùng RDD phải trao đổi dữ liệu vì mỗi khóa có thể nằm trong bất kỳ phân vùng nào từ tất cả các tổ tiên trực tiếp trở lên

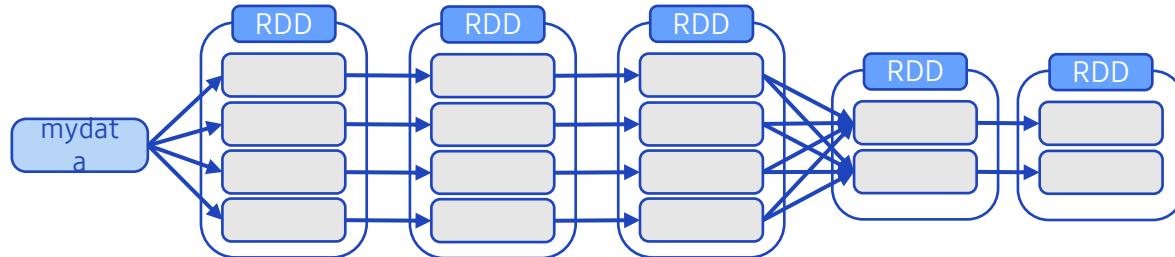
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
```



## Phân vùng lại sau khi tổng hợp (6/6)

- Chuyển đổi tuple (word, count) thành (count, tuple) chỉ yêu cầu dữ liệu từ cha mẹ trực tiếp

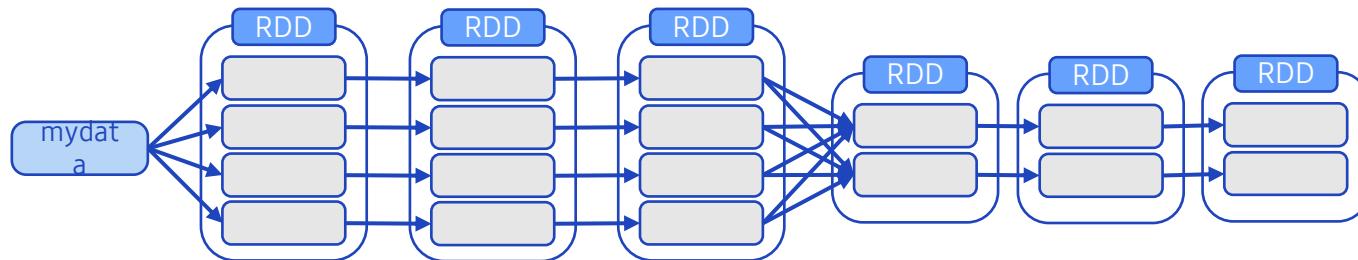
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
```



# Kiểm soát số lượng phân vùng

- | Kiểm soát số lượng phân vùng sau khi chuyển đổi tổng hợp `reduceByKey(numPartitions)`
- | Hoặc đặt `spark.default.parallelism` trong cài đặt cấu hình

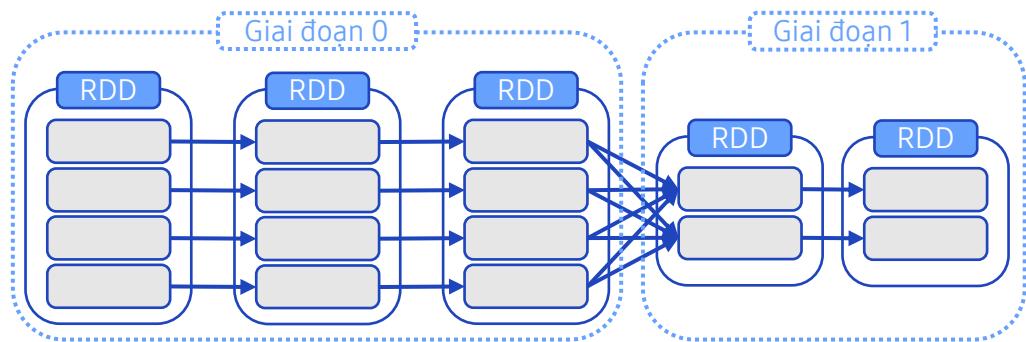
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2, 3) \
 .map(lambda tup: (tup[1], tup[0]))
```



## Các giai đoạn Spark

- Các giai đoạn là nơi xảy ra xáo trộn do có nhiều phụ thuộc ngược dòng

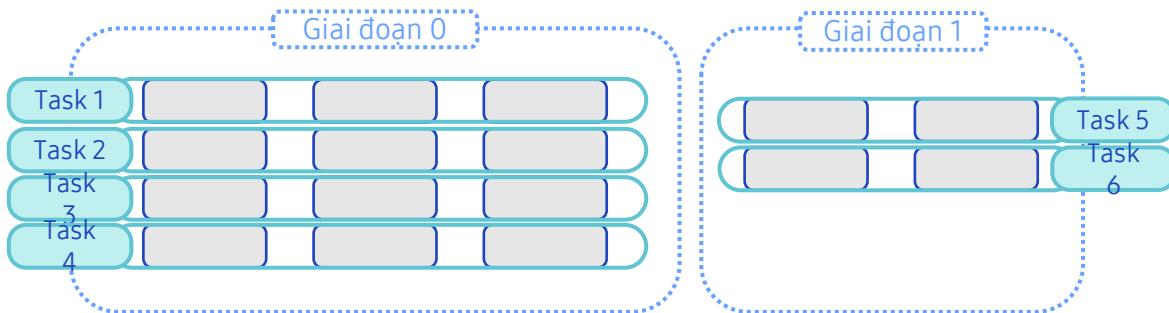
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
wordcount.saveAsTextFile("wordcount")
```



## Tác vụ đường ống (1/3)

- Các nhóm chuyển đổi chỉ phụ thuộc vào cha mẹ trực tiếp có thể được nhóm hợp lý thành các tác vụ

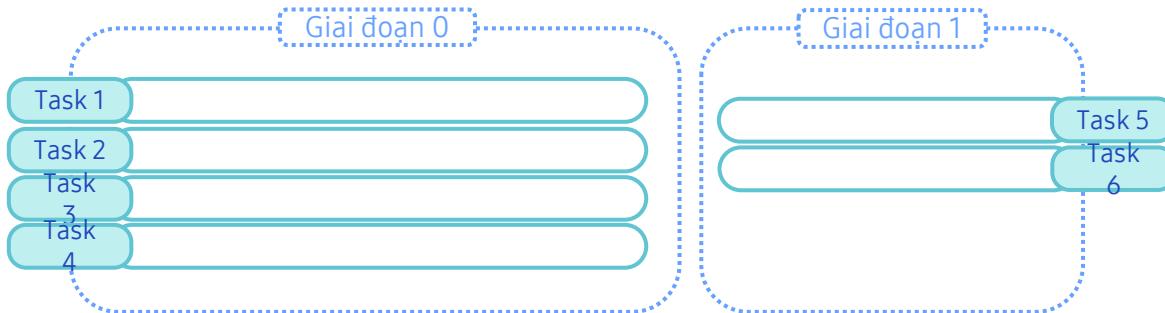
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
wordcount.saveAsTextFile("wordcount")
```



## Tác vụ đường ống (2/3)

- Các nhóm chuyển đổi chỉ phụ thuộc vào cha mẹ trực tiếp có thể được nhóm hợp lý thành các tác vụ

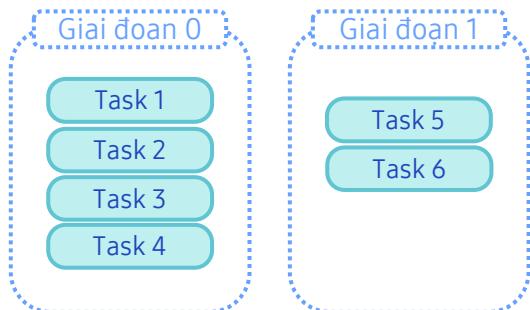
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
wordcount.saveAsTextFile("wordcount")
```



## Tác vụ đường ống (3/3)

- Các nhóm chuyển đổi chỉ phụ thuộc vào cha mẹ trực tiếp có thể được nhóm hợp lý thành các tác vụ

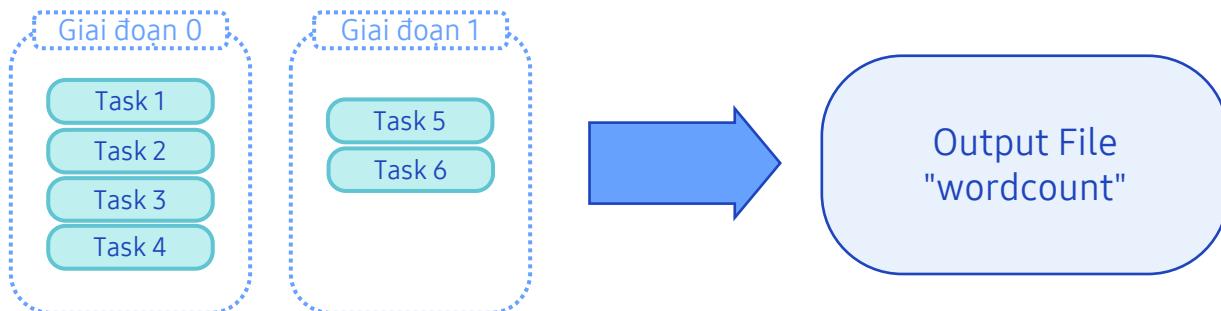
```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
wordcount.saveAsTextFile("wordcount")
```



# Công việc Spark

- Một Công việc xảy ra với các Hành động trong đó đầu ra kết quả được yêu cầu trả về

```
wordcount = sc.textFile("alice.txt") \
 .flatMap(lambda line: line.split(' ')) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda v1, v2: v1+v2) \
 .map(lambda tup: (tup[1], tup[0]))
wordcount.saveAsTextFile("wordcount")
```



# Spark Web UI - Jobs

## Xem công việc đã hoàn thành

The screenshot shows the Apache Spark 3.1.2 Web UI interface. The top navigation bar includes links for Jobs, Stages, Storage, Environment, Executors, SQL, and PySparkShell application UI. The main content area is titled "Spark Jobs (?)". It displays the following statistics:

- User: student
- Total Uptime: 1.9 min
- Scheduling Mode: FIFO
- Completed Jobs: 1

Below these stats, there are two navigation links: "Event Timeline" and "Completed Jobs (1)". The "Completed Jobs (1)" link is expanded, showing a table of completed tasks. The table has columns for Job Id, Description, Submitted, Duration, Stages: Succeeded/Total, and Tasks (for all stages): Succeeded/Total. The first row in the table is highlighted with a red border.

| Job Id | Description                                                                  | Submitted              | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|--------|------------------------------------------------------------------------------|------------------------|----------|-------------------------|-----------------------------------------|
| 0      | runJob at SparkHadoopWriter.scala:83<br>runJob at SparkHadoopWriter.scala:83 | 2021/09/06<br>00:02:34 | 4 s      | 2/2                     | 4/4                                     |

# Spark Web UI – Giai đoạn

- Xem các giai đoạn đã hoàn thành

The screenshot shows the Apache Spark 3.1.2 Web UI interface. The top navigation bar includes links for Jobs, Stages (which is selected), Storage, Environment, Executors, SQL, and PySparkShell application UI. The main content area is titled "Stages for All Jobs" and indicates "Completed Stages: 2". A collapsible section titled "Completed Stages (2)" is expanded, showing two rows of stage details. Each row includes a "Stage Id" (1 or 0), a "Description" (runJob at SparkHadoopWriter.scala:83 or reduceByKey at /tmp/ipykernel\_6175/2308689219.py:1), "Submitted" date and time, "Duration", "Tasks: Succeeded/Total" (2/2), "Input" and "Output" sizes, and "Shuffle Read" and "Shuffle Write" metrics.

| Stage Id | Description                                                           | Submitted              | Duration | Tasks: Succeeded/Total | Input        | Output      | Shuffle Read | Shuffle Write |
|----------|-----------------------------------------------------------------------|------------------------|----------|------------------------|--------------|-------------|--------------|---------------|
| 1        | runJob at<br>SparkHadoopWriter.scala:83<br>+details                   | 2021/09/06<br>00:02:37 | 1 s      | 2/2                    |              | 89.3<br>KiB | 62.7 KiB     |               |
| 0        | reduceByKey at<br>/tmp/ipykernel_6175<br>/2308689219.py:1<br>+details | 2021/09/06<br>00:02:34 | 2 s      | 2/2                    | 213.1<br>KiB |             | 62.7 KiB     |               |

Bài 4

# Ứng dụng Spark và điều chỉnh hiệu suất

- | 4.1. Ứng dụng Spark
- | 4.2. Xử lý phân tán
- | 4.3. Bền bỉ**

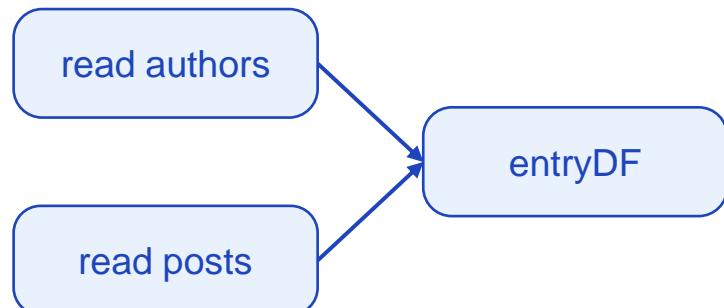
## Tính liên tục của Spark là gì?

- Spark RDD được gọi là Bộ dữ liệu phân tán đòn hồi
  - ▶ Chúng có khả năng phục hồi vì bất kỳ RDD nào cũng có thể được tạo lại khi cần thiết từ thông tin dòng dõi
  - ▶ Spark giữ lại thông tin dòng dõi miễn là vẫn còn các phụ thuộc hiện có
- Spark là một công cụ xử lý trong bộ nhớ
  - ▶ Tất cả RDD bất biến, thông tin dòng dõi, thông tin trạng thái được lưu trong bộ nhớ
  - ▶ Điều này có nghĩa là các RDD bất biến có thể bị hủy để nhường chỗ trong bộ nhớ
- Spark kiên trì là một cách để các nhà phát triển hướng dẫn Spark lưu một bản sao của RDD hoặc DataFrame
  - ▶ Sử dụng `persist()` hoặc bộ nhớ `cache()`
  - ▶ Kiên trì khi tính toán lại RDD hoặc DataFrame sẽ rất tốn kém
  - ▶ RDD hoặc DataFrame dự kiến sẽ được sử dụng thường xuyên

## Ví dụ về sự liên tục (1/6)

- Hoạt động tham gia là hoạt động phụ thuộc rộng và ranh giới Stage
  - Không có gì được tạo ra cho đến nay kể từ khi chuyển đổi lườn biếng

```
df1 = spark.read.option("header","true").csv("authors")
df2= spark.read.option("header","true").csv("posts")
entryDF = df1.join(df2, df1.id == df2.author_id)
```

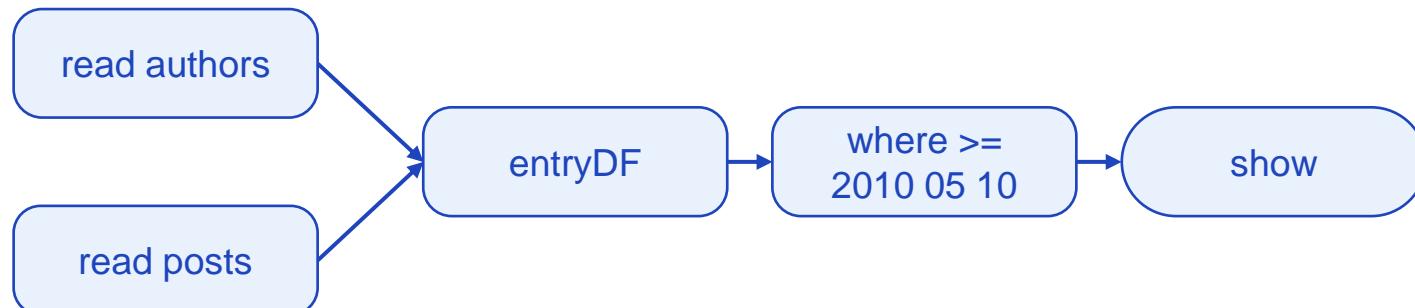


## Ví dụ về sự liên tục (2/6)

- Từ DataFrame đã tham gia, lọc theo điều kiện và hiển thị kết quả

- ▶ Show là một hành động khiến tất cả các chuyển đổi lười biếng đang chờ đợi tạo ra kết quả

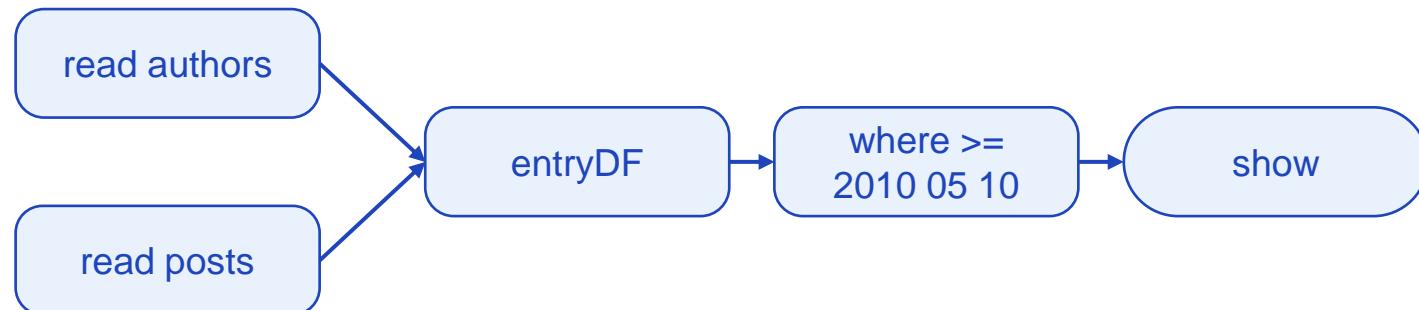
```
df1 = spark.read.option("header","true").csv("authors")
df2= spark.read.option("header","true").csv("posts")
entryDF = df1.join(df2, df1.id == df2.author_id)
entryDF.select("name").where(entryDF.joined >= "2010 05 10").show()
```



## Ví dụ về sự liên tục (3/6)

- Spark không đảm bảo rằng DataFrames sẽ vẫn còn trong bộ nhớ
  - DataFrames thường bị xóa để nhường chỗ trong bộ nhớ, tuy nhiên, dòng dõi được giữ nguyên

```
df1 = spark.read.option("header","true").csv("authors")
df2= spark.read.option("header","true").csv("posts")
entryDF = df1.join(df2, df1.id == df2.author_id)
entryDF.select("name").where(entryDF.joined >= "2010 05 10").show()
some time passes and entryDF gets destroyed in memory
```

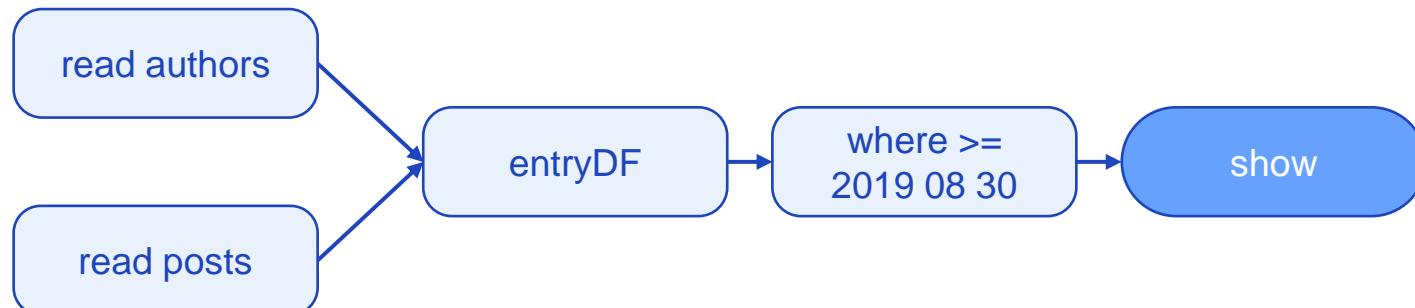


## Ví dụ về sự liên tục (4/6)

- Sau một thời gian, một truy vấn mới phụ thuộc vào entryDF được gọi. Tuy nhiên, truy vấn sẽ yêu cầu mụcDF được tạo lại từ dòng vì nó đã bị xóa.

```
df1 = spark.read.option("header", "true").csv("authors")
df2= spark.read.option("header", "true").csv("posts")
entryDF = df1.join(df2, df1.id == df2.author_id)
entryDF.select("name") .where(entryDF.joined >= "2010 05 10") .show()

entryDF.select("name") .where(entryDF.joined >= "2019 08 30") .show()
```

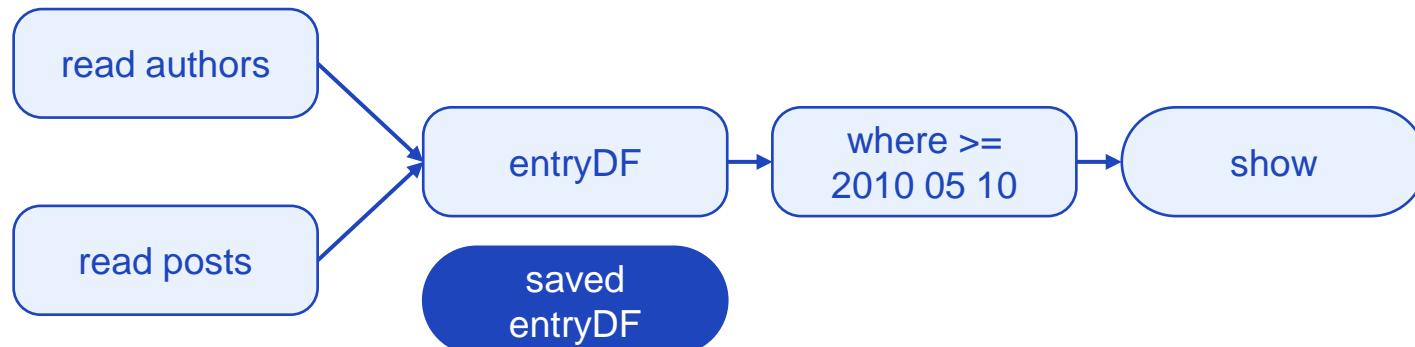


## Ví dụ về sự liên tục (5/6)

| Lần này, hãy lưu entryDF bằng cách duy trì nó.

▶ Vì chương trình đó đã kích hoạt tất cả quá trình chuyển đổi, một bản sao của entryDF hiện có sẵn trong bộ nhớ

```
df1 = spark.read.option("header", "true").csv("authors")
df2= spark.read.option("header", "true").csv("posts")
entryDF = df1.join(df2, df1.id == df2.author_id). persist()
entryDF.select("name").where(entryDF.joined >= "2010 05 10").show()
```



## Ví dụ về sự liên tục (6/6)

- Khi một truy vấn mới phụ thuộc vào entryDF được gọi sau đó, Spark có thể sử dụng entryDF đã lưu để thay thế.

```
df1 = spark.read.option("header","true").csv("authors")
df2= spark.read.option("header","true").csv("posts")
entryDF = df1.join(df2, df1.id == df2.author_id). persist()

entryDF.select("name").where(entryDF.joined >= "2019 08 30").show()
```



## Các bảng và dạng xem liên tục

- I Các bảng và dạng xem SQL có thể được duy trì trong bộ nhớ

- ▶ Sử dụng lệnh CACHE TABLE SQL

```
entryDF.createOrReplaceTempView("entry")
spark.sql("CACHE TABLE entry")
```

- ▶ CACHE TABLE có thể tạo dạng xem và lưu vào bộ nhớ cùng một lúc

```
spark.sql("CACHE TABLE entry as SELECT * FROM author JOIN posts ")
```

## Mức lưu trữ

- | Dữ liệu liên tục có thể được lưu ở các cấp lưu trữ khác nhau
  - ▶ Lưu trữ trong bộ nhớ, bộ nhớ và đĩa, hoặc đĩa
  - ▶ Tuần tự hóa và nén dữ liệu trong bộ nhớ
  - ▶ Sao chép dữ liệu
- | Persisted DataFrames không sử dụng mức lưu trữ
  - ▶ Tất cả các bảng chỉ được lưu vào bộ nhớ

## Vị trí cấp độ lưu trữ

- | Để đặt mức lưu trữ, trước tiên hãy nhập giao diện StorageLevel
- | Mức lưu trữ có sẵn là:
  - ▶ MEMORY\_ONLY ➔ Lưu trữ tất cả dữ liệu trong bộ nhớ
  - ▶ DISK\_ONLY ➔ Lưu trữ tất cả dữ liệu trên đĩa
  - ▶ MEMORY\_AND\_DISK ➔ Nếu có bất kỳ phân vùng nào không thể vừa trong bộ nhớ, hãy lưu trữ trên đĩa

```
from pyspark import StorageLevel
entryDF.persist(StorageLevel.MEMORY_AND_DISK)
```

## Tuần tự hóa bộ nhớ

- Tuần tự hóa là sự đánh đổi giữa hiệu quả về không gian và hiệu quả về thời gian
- Dữ liệu trong Python luôn được tuần tự hóa
- Trong Scala, có thể chọn tuần tự hóa
  - ▶ Scala lưu trữ dữ liệu dưới dạng các đối tượng trong bộ nhớ
  - ▶ Thay vì các đối tượng được lưu trữ, tuần tự hóa thành chuỗi byte
  - ▶ MEMORY\_ONLY\_SER ➔ Lưu trữ trong bộ nhớ và tuần tự hóa
  - ▶ MEMORY\_AND\_DISK\_SER ➔ Lưu trữ các chuỗi byte được tuần tự hóa trên bộ nhớ và tràn vào đĩa nếu cần
- Tuần tự hóa không được áp dụng để lưu vào đĩa
  - ▶ Dữ liệu trên đĩa, theo định nghĩa, được tuần tự hóa

## Sao chép phân vùng

- I Sao chép và lưu trữ trên hai nút
  - ▶ DISK\_ONLY\_2
  - ▶ MEMORY\_AND\_DISK\_2
  - ▶ MEMORY\_ONLY\_2
  - ▶ MEMORY\_AND\_DISK\_SER\_2 (Chỉ Scala và Java vì Python luôn được tuần tự hóa)
  - ▶ MEMORY\_ONLY\_SER\_2 (Chỉ Scala và Java vì Python luôn được tuần tự hóa)

## Chọn mức lưu trữ phù hợp

- | Vẫn liên tục sau một hoạt động tốn kém như tổng hợp
  - ▶ Nói cách khác, tại ranh giới Stage
- | Kiên trì trên các ứng dụng có vòng lặp dài như Machine Learning
  - ▶ Tái tạo tất cả các lần lặp lại sẽ rất tốn kém
- | Mức độ lưu trữ
  - ▶ Sử dụng bộ nhớ khi cố gắng cải thiện hiệu suất
  - ▶ Sử dụng Disk khi chi phí tính toán lại đắt hơn Disk I/O
  - ▶ Sử dụng Replication khi chi phí tính toán lại đắt hơn băng thông mạng

# Xem dữ liệu liên tục trên Spark Web UI

## Xem từ khu Lưu trữ trên Spark Web UI

The screenshot shows the Apache Spark 3.1.2 Web UI interface. The top navigation bar includes links for Jobs, Stages, Storage (which is currently selected), Environment, Executors, SQL, and PySparkShell application UI. The main content area is titled "Storage" and contains a sub-section "RDDs". A table displays the following data:

| ID | RDD Name  | Storage Level                   | Cached Partitions | Fraction Cached | Size in Memory | Size on Disk |
|----|-----------|---------------------------------|-------------------|-----------------|----------------|--------------|
| 15 | PythonRDD | Memory Serialized 1x Replicated | 1                 | 50%             | 26.3 KiB       | 0.0 B        |

## Loại bỏ hoặc thay đổi tính liên tục

- I Trước khi có thể thay đổi Mức lưu trữ cho dữ liệu được lưu trữ lâu dài, trước tiên phải không được lưu trữ
  - ▶ Sử dụng **unpersist()** cho DataFrames và RDDs
  - ▶ Sử dụng **SparkSession.Catalog.uncacheTable(<table\_name>)** cho bảng và dạng xem

```
from pyspark import StorageLevel
eventDF.unpersist()
eventDF.persist(StorageLevel.MEMORY)
```

[Lab14]

Tạo một ứng dụng Apache Spark





# Together for Tomorrow! Enabling People

Education for Future Generations

©2021 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of book.

This book is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.

To use this book other than the curriculum of Samsung innovation Campus or to use the entire or part of this book, you must receive written consent from copyright holder.