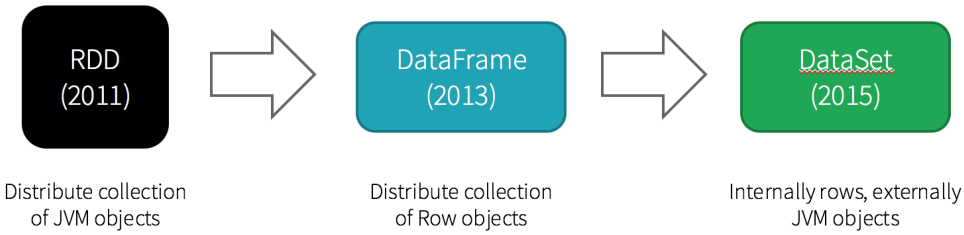
Bài tập 2
Lập trình Spark với Structured API

1. Tóm tắt Spark Structured API

Khi làm việc với RDD, Spark không biết bạn đang thực hiện những gì bên trong các hàm tính toán của biểu thức lambda, hay đối tượng tham gia tính toán là gì. Spark coi biểu thức lambda là hộp đen. Do đó, Spark không thể tối ưu hóa biểu thức dựa trên các hoạt động cơ bản. Nó coi biểu thức là một phép biến đổi chung. Spark chỉ có thể biến đổi tượng thành một chuỗi byte (serialize), mà không sử dụng bất kỳ kỹ thuật nén dữ liệu nào.

Dữ liệu có cấu trúc (structured data) là thông tin được tổ chức theo cấu trúc chuẩn hóa hoặc theo một lược đồ cụ thể. Spark đưa ra hai loại tập hợp có cấu trúc: DataFrames và Datasets. DataFrames và Datasets là các tập hợp (phân tán) giống như bảng với các hàng và cột được xác định rõ ràng. Mỗi cột phải có cùng số hàng như tất cả các cột khác (mặc dù bạn có thể sử dụng null để chỉ định giá trị không có) và mỗi cột có thông tin về kiểu dữ liệu phải nhất quán cho mỗi hàng trong tập hợp.



Khi làm việc với DataFrame và DataSet, chúng ta cần một lược đồ (mặc dù đối với DataFrame thì lược đồ là không bắt buộc). Một lược đồ (schema) xác định tên cột và kiểu dữ liệu. Bạn có thể xác định lược đồ bằng cách thủ công hoặc đọc lược đồ từ nguồn dữ liệu.

DataFrame = Dataset[Row]

```
schema = StructType(List(
    StructField("name", StringType, true),
    StructField("role", StringType, true),
    StructField("salary", IntegerType, true))
)
```

	name: Any	role: Any	salary: Any	
Row(	"John",	"Data scientist",	4500	)
Row(	"James",	"Data engineer",	3200	)
Row(	"Laura",	"Data scientist",	4100	)
Row(	"Ali",	"Data engineer",	3200	)
Row(	"Steve",	"Developer",	3600	)

Nói rằng DataFrames không có kiểu dữ liệu là không hoàn toàn chính xác; chúng có kiểu dữ liệu, nhưng Spark duy trì chúng hoàn toàn và chỉ kiểm tra xem những kiểu đó có khớp với những kiểu được chỉ định trong lược đồ ở thời gian chạy. Trong khi đó, Datasets kiểm tra xem kiểu dữ liệu có tuân thủ các quy định trong quá trình biên dịch. Datasets chỉ khả dụng cho các ngôn ngữ dựa trên Java Virtual Machine (JVM) (Scala và Java, không dành cho Python và R). Điều này

nghĩa là đối với Python và R trên Spark, mọi thứ có cấu trúc chỉ là DataFrame. DataFrame có thể được coi như một Dataset có kiểu là Row.

Dataset[Employee]

case class Employee(name: **String**, role: **String**, salary: **Integer**)

Employee(	"John",	"Data scientist",	4500)
Employee(	"James",	"Data engineer",	3200)
Employee(	"Laura",	"Data scientist",	4100)
Employee(	"Ali",	"Data engineer",	3200)
Employee(	"Steve",	"Developer",	3600)

Spark có thể coi là một ngôn ngữ lập trình riêng. Bên trong, Spark sử dụng một engine được gọi là Catalyst để duy trì thông tin kiểu dữ liệu qua quá trình lập kế hoạch và xử lý công việc. Điều này mở ra một loạt các tối ưu hóa thực thi khác nhau có thể tạo ra sự khác biệt đáng kể. Các kiểu dữ liệu trong Spark tương ứng trực tiếp với các API ngôn ngữ khác nhau mà Spark hỗ trợ, và tồn tại một bảng tra cứu cho mỗi kiểu dữ liệu này trong Scala, Java, Python, SQL và R.

Các cột (column) trong Spark có thể đại diện cho các kiểu dữ liệu đơn giản như số nguyên (integer) hoặc chuỗi (string), các kiểu dữ liệu phức tạp như mảng (array) hoặc bản đồ (map), hoặc giá trị null. Một hàng (row) là một bản ghi dữ liệu (record). Mỗi bản ghi trong DataFrame phải có kiểu Row.

2. DataFrame

Các mã nguồn ở các phần sau đây sẽ được thực thi trong môi trường spark-shell (Scala). Bạn có thể đóng gói thành một chương trình bằng cách bổ sung thêm lớp và hàm main, thực hiện biên dịch và chạy như trong các bài lab trước đây.

2.1. Định nghĩa Schema

Một lược đồ (schema) là một StructType được tạo thành từ một số trường được gọi là StructField. Mỗi trường có tên, kiểu dữ liệu, một cờ Boolean xác định xem cột đó có thể chứa giá trị bị thiếu hoặc null hay không, và cuối cùng, người dùng có thể mô tả siêu dữ liệu liên quan đến cột đó. Siêu dữ liệu (metadata) là một cách để mô tả thông tin về cột đó.

```
import org.apache.spark.sql.types._
import org.apache.spark.sql.types.Metadata

val myManualSchema = StructType(Array(
     StructField("DEST_COUNTRY_NAME", StringType, true),
     StructField("ORIGIN_COUNTRY_NAME", StringType, true),
     StructField("count", LongType, false,
                 Metadata.fromJson("{\"hello\":\"world\"}"))
))
```

Một cách định nghĩa khác là sử dụng chuỗi DDL (Data Definition Language):

```
val schema = "author STRING, title STRING, pages INT"
```

2.2. Tạo dữ liệu DataFrame

Chúng ta có thể tạo dữ liệu thủ công bằng cách thêm từng dòng dữ liệu và sử dụng hàm createDataFrame() để sinh ra DataFrame.

Ví dụ:

```
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row

// Define schema for our data using DDL
val schema = StructType(Seq(
   StructField("Id", IntegerType),
   StructField("First", StringType),
   StructField("Last", StringType),
   StructField("Url", StringType),
   StructField("Published", StringType),
   StructField("Hits", IntegerType),
   StructField("Campaigns", ArrayType(StringType))
))

// Create our static data
val data = Seq(
Row(1, "Jules", "Damji", "https://tinyurl.1", "1/4/2016", 4535,
Array("twitter", "LinkedIn")),
Row(2, "Brooke", "Wenig", "https://tinyurl.2", "5/5/2018", 8908,
Array("twitter", "LinkedIn")),
Row(3, "Denny", "Lee", "https://tinyurl.3", "6/7/2019", 7659,
Array("web", "twitter", "FB", "LinkedIn")),
Row(4, "Tathagata", "Das", "https://tinyurl.4", "5/12/2018", 10568,
Array("twitter", "FB")),
Row(5, "Matei", "Zaharia", "https://tinyurl.5", "5/14/2014", 40578,
Array("web", "twitter", "FB", "LinkedIn")),
Row(6, "Reynold", "Xin", "https://tinyurl.6", "3/2/2015", 25568,
Array("twitter", "LinkedIn"))
)

// Create a SparkSession
val spark = SparkSession.builder().appName("Example").getOrCreate()

// Create a DataFrame using the schema defined above
val df = spark.createDataFrame(spark.sparkContext.parallelize(data),
schema)

// Show the DataFrame; it should reflect our table above
df.show()

// Print the schema used by Spark to process the DataFrame
print(blogs_df.printSchema())
```

Spark cũng cung cấp một giao diện gọi là DataFrameReader, cho phép đọc dữ liệu vào DataFrame từ nhiều nguồn dữ liệu khác nhau với các định dạng như JSON, CSV, Parquet, Text, Avro, ORC, v.v. Các hàm hỗ trợ như *json()*, *csv()*, v.v…

Đầu tiên bạn tạo một tập tin có tên là blogs.json sử dụng vim với nội dung sau:

```
{"Id":1, "First": "Jules", "Last":"Damji", "Url":"https://tinyurl.1",
"Published":"1/4/2016", "Hits": 4535, "Campaigns": ["twitter", "LinkedIn"]}
{"Id":2, "First": "Brooke","Last": "Wenig","Url": "https://tinyurl.2", "Published":
"5/5/2018", "Hits":8908, "Campaigns": ["twitter", "LinkedIn"]}
{"Id": 3, "First": "Denny", "Last": "Lee", "Url": "https://tinyurl.3","Published":
"6/7/2019","Hits": 7659, "Campaigns": ["web", "twitter", "FB", "LinkedIn"]}
{"Id": 4, "First":"Tathagata", "Last": "Das","Url": "https://tinyurl.4", "Published":
"5/12/2018", "Hits": 10568, "Campaigns": ["twitter", "FB"]}
{"Id": 5, "First": "Matei","Last": "Zaharia","Url": "https://tinyurl.5", "Published":
"5/14/2014", "Hits": 40578, "Campaigns": ["web", "twitter", "FB", "LinkedIn"]}
{"Id":6, "First": "Reynold", "Last": "Xin", "Url": "https://tinyurl.6", "Published":
"3/2/2015", "Hits": 25568, "Campaigns": ["twitter", "LinkedIn"] }
```

Thực hiện chép tập tin này đến thư mục làm việc của spark tại đường dẫn /usr/local/spark/input/blogs.json

```
$ cp blogs.json /usr/local/spark/input/blogs.json
```

Trở lại spark-shell và lập trình với các câu lệnh sau:

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._

val jsonFile = "file:///usr/local/spark/input/blogs.json"

 // Define our schema programmatically
 val schema = StructType(Array(StructField("Id", IntegerType, false),
    StructField("First", StringType, false),
    StructField("Last", StringType, false),
    StructField("Url", StringType, false),
    StructField("Published", StringType, false),
    StructField("Hits", IntegerType, false),
    StructField("Campaigns", ArrayType(StringType), false)))

 // Create a DataFrame by reading from the JSON file
 // with a predefined schema
 val blogsDF = spark.read.schema(schema).json(jsonFile)

// you can use another way
// spark.read.schema(schema).format("json").load(jsonFile)

 // Show the DataFrame schema as output
 blogsDF.show(false)

 // Print the schema
 println(blogsDF.printSchema)
 println(blogsDF.schema)
```

2.3. Thao tác với cột

Các cột trong Spark là các đối tượng có các phương thức public (được biểu diễn bởi kiểu dữ liệu `Column`). Chúng ta có thể chọn, thao tác và loại bỏ các cột từ DataFrames. Các hành động này được thể hiện dưới dạng các biểu thức (*expression*).

Hàm ***col()*** nhận vào tên của cột trong DataFrame và trả về tham chiếu đến cột được chọn. Ví dụ, tiếp theo các câu lệnh từ phần trước cho bến blogsDF:

```
scala> import org.apache.spark.sql.functions._
scala> blogsDF.col("Id")
```

Để xem toàn bộ tên các cột của một DataFrame, ta có thể dùng *thuộc tính columns*. Ví dụ:

```
scala> blogsDF.columns
```

Hàm ***expr()*** là hàm nhận vào một biểu thức và tính toán kết quả trả ra. Nó tương tự như một hàm nhận vào một hoặc nhiều tên cột làm đầu vào, giải quyết chúng, và sau đó có thể áp dụng thêm các biểu thức để tạo ra một giá trị duy nhất cho mỗi bản ghi trong tập dữ liệu. Ví dụ, câu lệnh sau dùng để tạo ra một cột mới bằng cách chọn dữ liệu từ cột Hits và nhân lên gấp đôi:

```
// Use an expression to compute a value
scala> blogsDF.select(expr("Hits * 2")).show(2)
// or use col to compute value
scala> blogsDF.select(col("Hits") * 2).show(2)
```

Trong câu lệnh trên, ta có thể thực hiện biểu thức bằng cách tham chiếu đến cột thông qua hàm col() và thực hiện biểu thức tường minh mà không cần phải ghi hàm expr(). Khi biểu thức của hàm expr() chỉ là một cột thì nó tương đương với hàm col(). Hàm ***select()*** được dùng để lấy ra một cột để xem.

Ta có thể đặt tên cho cột mới sau khi thêm bằng hàm ***withColumn()***. Ví dụ:

```
// Use an expression to compute big hitters for blogs
// This adds a new column, Big Hitters, based on conditional expression
scala> blogsDF.withColumn("Big Hitters", (expr("Hits > 10000"))).show()
```

Ta có thể nối các giá trị của các cột lại với nhau bằng hàm *concat()*. Ví dụ:

```
// Concatenate three columns, create a new column, and show the
// newly created concatenated column
scala> blogsDF.withColumn("AuthorsId", (concat(expr("First"), expr("Last"),
expr("Id"))))
      .select(col("AuthorsId"))
      .show(4)
```

Sắp xếp một cột sử dụng hàm ***sort()***. Ví dụ:

```
// Sort by column "Id" in descending order
blogsDF.sort(col("Id").desc).show()
```

Để thay đổi tên một cột, ta dùng hàm ***withColumnRenamed()***:

```
blogsDF.withColumnRenamed("Id", "Blog_Id").show()
```

Để xóa cột, dùng hàm ***drop()***:

```
blogsDF.drop("Id").show()
```

2.4. Thao tác với hàng

Trong Spark, một hàng (row) là một đối tượng Row, chứa một hoặc nhiều cột. Vì Row là một đối tượng trong Spark và một tập hợp được sắp xếp của các trường, bạn có thể khởi tạo một Row và truy cập các trường bằng chỉ số bắt đầu từ 0.

```
import org.apache.spark.sql.Row

// Create a Row
val blogRow = Row(6, "Reynold", "Xin", "https://tinyurl.6", 255568,
"3/2/2015", Array("twitter", "LinkedIn"))

// Access using index for individual items
blogRow(1)
```

Các đối tượng Row có thể được sử dụng để tạo DataFrames bằng hàm ***toDF()***:

```
val rows = Seq(Row("Matei Zaharia", "CA"), Row("Reynold Xin", "CA"))
val authorsDF = rows.toDF("Author", "State")
authorsDF.show()
```

2.5. Lưu dữ liệu DataFrame lên tập tin

Để ghi DataFrame vào một nguồn dữ liệu, bạn có thể sử dụng giao diện DataFrameWriter. Giống như DataFrameReader, nó hỗ trợ nhiều nguồn dữ liệu. Parquet, một định dạng cột phổ biến, là định dạng mặc định; nó sử dụng nén snappy để nén dữ liệu. Nếu DataFrame được ghi dưới dạng Parquet, lược đồ được lưu trữ như là một phần của siêu dữ liệu Parquet. Trong trường hợp này, việc đọc lại vào DataFrame không yêu cầu cung cấp lược đồ thủ công lại.

```
val parquetPath = "file:///usr/local/spark/output/myBlog"
blogsDF.write.format("parquet").save(parquetPath)

//you can open parquet again
val newBlogsDF = spark.read.parquet(parquetPath)
newBlogsDF.show()
```

2.6. Thực thi trên DataFrame

Các hoạt động Spark trên dữ liệu phân tán có thể được phân loại thành hai loại: biến đổi (transformation) và hành động (action). Biến đổi, như tên gọi của nó, biến đổi một DataFrame của Spark thành một DataFrame mới mà không thay đổi dữ liệu gốc, mang tính chất bất biến (immutability). Nói cách khác, một hoạt động như select() hoặc filter() sẽ không thay đổi DataFrame gốc; thay vào đó, nó sẽ trả về kết quả biến đổi của hoạt động dưới dạng DataFrame mới.

Danh sách các hàm transformation và action có phần giống với dánh sách trên RDD ở bài lab trước nên chúng ta sẽ chạy thử nghiệm để xem cách thức hành xử trên DataFrame.

Tạo một file csv có tên là dataset.csv với nội dung như sau:

```
State,Color,Count
TX,Red,20
NV,Blue,66
CO,Blue,79
OR,Blue,71
WA,Yellow,93
WY,Blue,16
CA,Yellow,53
WA,Green,60
OR,Green,71
TX,Green,68
NV,Green,59
AZ,Brown,95
```

Chép vào thư mục /usr/local/spark/input/dataset.csv để sử dụng trong các ví dụ tiếp theo.

Đọc dữ liệu và đưa vào DataFrame:

```
   scala> import org.apache.spark.sql.functions._
   scala> val df = spark.read.format("csv").option("header",
"true").option("inferSchema","true").load("file:///usr/local/spark/inpu
t/dataset.csv")
   scala> df.show(5, false)
```

Giả sử ta có nhu cầu tổng hợp theo State và Color, ta code như sau:

```
val countDF = df.select("State", "Color", "Count")
        .groupBy("State", "Color")
        .sum("Count")
        .orderBy(desc("sum(Count)"))
countDF.show(6)
println(s"Total Rows = ${countDF.count()}")
println()
```

Các hàm được sử dụng gồm ***select()***, ***groupBy()***, ***sum()***, ***orderBy()***, ***count()***. Bây giờ, ta muốn tìm kiếm các dòng có các cột thỏa điều kiện cho trước:

```
// find the aggregate count for California by filtering
val caCountDF = mnmDF.select("*")
      .where(col("State") === "CA")
```

```
            .groupBy("State", "Color")
            .sum("Count")
            .orderBy(desc("sum(Count)"))

        // show the resulting aggregation for California
        caCountDF.show(5)
```

Trong ví dụ này, hàm ***where()*** được sử dụng. Lưu ý trong hàm where() chúng ta mong muốn đánh giá biểu thức (expression) nên cần phải gọi các hàm như col() hay expr(). Lúc này, bạn có thể nhận thấy nó có cú pháp khá quen thuộc với ngôn ngữ SQL hay LINQ.

Ngoài các hàm trên, ta có các hàm khác như min(), max(), avg(), …

2.7. Viết ứng dụng hoàn chỉnh WordCount với DataFrame

Trong phần này, chúng ta sẽ viết chương trình để đếm từ sử dụng các thao tác trên DataFrame. Tương tự trong lab trước, ta cần tạo một tập tin mã nguồn Scala cho chương trình WordCount:

```
$ sudo vi WordCountDF.scala
```

Gõ nội dung sau vào tập tin trên, hiểu ý nghĩa của từng dòng lệnh:

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._

object WordCountDF {

  def main(args:Array[String]): Unit = {

    val spark:SparkSession = SparkSession.builder()
      .master("local[*]")
      .appName("WordCountDF")
      .getOrCreate()

    //Turn off INFO message, display ERROR only
    spark.sparkContext.setLogLevel("ERROR")

     import spark.implicits._

    // Read text file into DataFrame
    val df =
spark.read.text("file:///usr/local/spark/input/input.txt")

    // Apply Split and Explode
    val df_exploded = df.withColumn("word", explode(split($"value", " ")))

    // Group by word and count
    val df_count = df_exploded.groupBy("word")
      .count()
      .orderBy($"count".desc)

    // Display Output
    df_count.show()

    // Save word count results to text file
    //df_count.write.text("file:///usr/local/spark/output/wordCountDF")

  }
}
```

Lưu ý kiểm tra tập tin input.txt đã có trong đường dẫn chưa. Thực hiện biên dịch chương trình bằng lệnh:

```
$ scalac -classpath "/usr/local/spark/jars/*" WordCountDF.scal WordCountDF.jar
```

Chạy chương trình WordCount:

```
$ spark-submit -class WordCountDF WordCountDF.jar
```

Nếu thành công thì màn hình sẽ xuất kết quả của đếm từ.

3. DataSet

Quá trình thực thi trên DataSet khá tương tự, bạn tự tìm hiểu thêm.

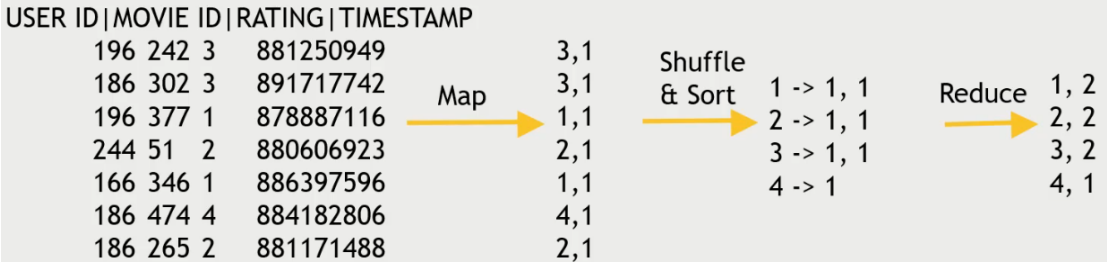
4. Bài tập

Thực hiện lại các bài tập ở bài Map-Reduce sang Spark Structure API. Cụ thể:

4.1. Bài 1

Xét tập dữ liệu ratings (u.data) trong MovieLens, thực hiện chương trình thống kê số lượng người bình chọn ở mỗi mức.

Quá trình Map-Reduce như sau:



Hãy thực hiện trong Spark Structure API.

4.2. Bài 2

Tiếp tục với dữ liệu phía trên để sắp xếp các phim theo số lượt bình chọn.

4.3. Bài 3

Thống kê mỗi từ xuất hiện trong tài liệu cho trước:

a. Trường hợp phân biệt hoa thường

b. Trường hợp không phân biệt hoa thường

Tự tạo dữ liệu là một tập tin văn bản ngắn để test trên local.

Khi thành công, thực hiện tải dữ liệu sau lên HDFS và thực hiện đếm. Dữ liệu gồm 3 cuốn sách:

http://www.gutenberg.org/ebooks/20417
http://www.gutenberg.org/ebooks/5000
http://www.gutenberg.org/ebooks/4300

4.4. Bài 5

Tìm từ xuất hiện nhiều nhất trong tài liệu (không phân biệt hoa thường). Dữ liệu có thể lấy từ bài tập trên.