VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE

INTRODUCTION TO BIG DATA ANALYSIS

Lab instructors

Vũ Công Thành | vcthanh.work@gmail.com
Huỳnh Lâm Hải Đăng | danghailamhuynh@gmail.com

Lab 02: Advanced MapReduce & Spark Structured APIs

1. Preliminaries

- To do this lab, you need to know the basic concepts of Spark and Structured APIs. You can read the basics in this article (Link) or the official tutorial from Apache (Link – written in Python but it is similar in other languages).

1.1. The history of Spark

- In the early days of Spark, data operations were performed directly on RDDs, which were much harder to write and optimize.
- With the advent of DataFrame and Structured APIs, writing Spark jobs has become much easier and more efficient thanks to automatic optimization features.
- To fully and accurately understand all the concepts that power Spark like Catalyst optimizer, Whole-stage Code Gen, etc. I recommend you to read more in the book named ***Spark: The Definitive Guide***. (it is not required, but you should take a look).
- Later, more and more features were introduced, such as Adaptive Query Execution (AQE) introduced in version 3.0 to optimize Spark jobs, making them run significantly faster *in most cases*.

1.2. DataFrame

DataFrame in Spark has the following properties:

- ***Schema-based***: Dataframes have schemas, which require declaring the names and data types of the columns in the dataframe (that's why they are named Structured APIs). Supports many primitive data types as well as other complex data types.
- ***Immutable***: Dataframes in Spark cannot be modified after being created. However, we can transform them into another dataframe through **transformations**.
- ***Lazy evaluation***: When the user has not performed an **action**, the previous transformations (in the same stage) have not been run. Once an action is triggered, the Catalyst optimizer will review the execution plan, then optimize the entire stage, and then perform processing on the optimized plan.

1.3. The underlying of Spark vs. MapReduce

Spark DataFrames and MapReduce both enable distributed data processing but differ significantly in execution and efficiency. Both frameworks process large-scale datasets in parallel across clusters, breaking computations into smaller tasks that run on multiple nodes.

However, MapReduce follows a disk-based, batch processing model, where intermediate results are written to disk between the map and reduce phases, making it slower due to frequent I/O operations. In contrast, Spark DataFrames leverage in-memory processing, significantly reducing disk I/O and improving speed.

Another key difference is that MapReduce requires users to explicitly define map and reduce functions, whereas DataFrames provide a higher-level, SQL-like API that abstracts these details, allowing for more readable and concise code.

Additionally, Spark DataFrames benefit from Catalyst Optimizer and Tungsten Engine, which apply automatic query optimizations and memory-efficient execution, whereas MapReduce lacks built-in query optimization and relies solely on user-defined logic.

Despite these differences, both frameworks perform similar fundamental operations, such as mapping, shuffling, and reducing data, but Spark DataFrames streamline this process with optimized execution plans and a more user-friendly API, making them a more efficient and flexible alternative to MapReduce.

2. Homework

In all exercises below, you can choose your preferred language to work with.

In both exercises 2.1 and 2.2 below, you will use the same Amazon Sale Report[1] dataset that is included in this Lab.

2.1. Calculate revenue in the last 3 days for each country

Using the ***MapReduce framework***:

1. Implement a sliding window computation to calculate the **total sales (Amount) for each Category in the last 3 days.**
2. The window should slide by 1 day at a time.
3. Sort the results in ascending order by report_date, followed by category.

The expected output should look like this (make sure the format is correct, column names are all in lowercase):

report_date	category	revenue
30/04/2022	Set	3829.38
30/04/2022	Western Dress	4392.85
30/04/2022	kurta	3293.19
…	…	…

[1] From E-Commerce Sales Dataset

2.2. Calculate the number of products sold in the last 7 days of each SKU, report every Monday

Using ***Apache Spark***:

1. Implement a computation to calculate the **total quantity (Qty) of items shipped for each SKU in the last 7 days, but only report this total every Monday**.
2. The goal is to provide a weekly snapshot of the total quantity shipped for each SKU over the past week.
3. Sort the results in ascending order by report_date, followed by SKU.

Expected output:

report_date	sku	total_quantity
02/05/2022	Set	338
02/05/2022	Western Dress	329
02/05/2022	kurta	392
…	…	…

2.3. Find pairs of overlapping shapes

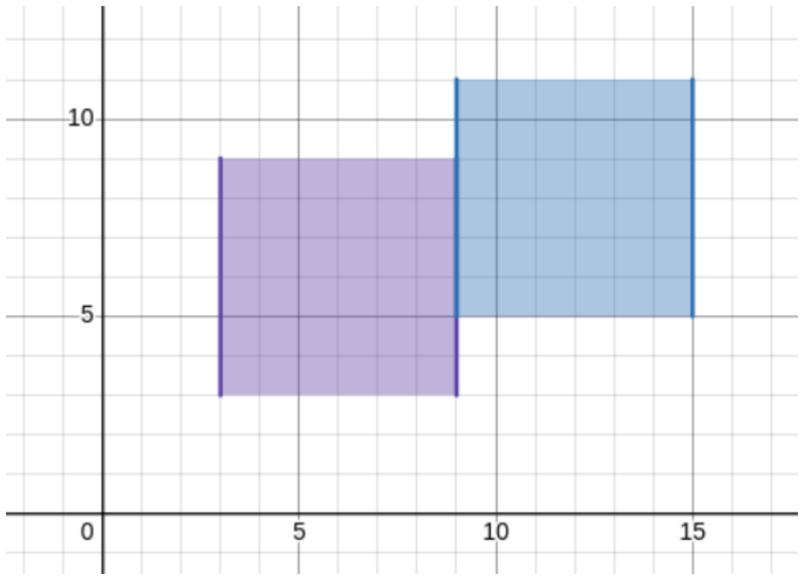
You are provided with a dataset (*shapes.parquet*) containing 1000 geometric shapes, where each shape is represented as a quadrilateral with 4 vertices. Each shape is defined by its **unique ID** and **four vertices** representing a quadrilateral in a 2D coordinate system.

Using ***Apache Spark***:

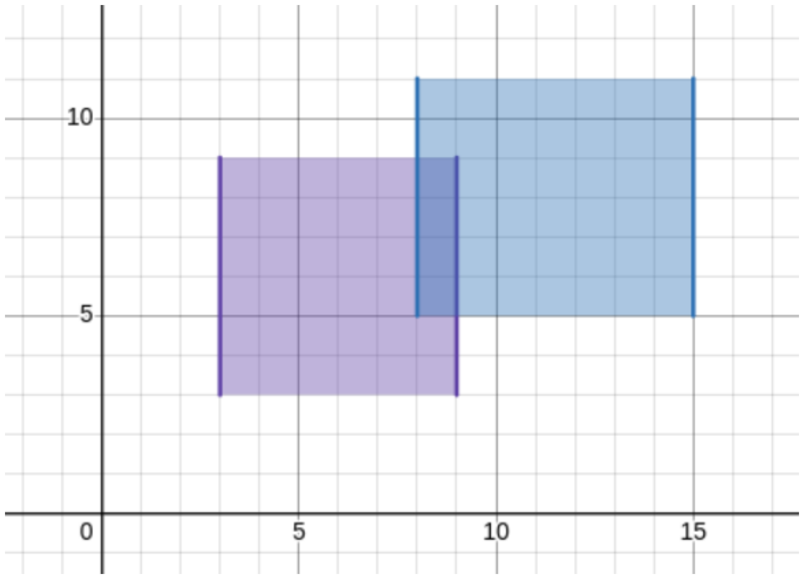
1. Implement the Overlap Detection Algorithm.
2. Output the overlapping shape pairs (their ids).
3. Sort the results in ascending order by shape_1, followed by shape_2.

Note that shapes only share points on the boundaries are not counted as overlapping shapes.

For example, these two shapes are not overlapped:



These two shapes are overlapped:



Expected output:

shape_1	shape_2
1	3
1	7
8	10
…	…

3. Submission Guideline

File structure of StudentID.zip

```
StudentID
|--- src
|    |--- Task_2.1
|    |--- | <StudentID>.py/<StudentID>.jar
|    |--- | output.csv
|    |--- |   ...
|    |--- Task_2.2
|    |--- | <StudentID>.py/<StudentID>.jar
|    |--- | output.csv
|    |--- |   ...
|    |--- Task_2.3
|    |--- | <StudentID>.py/<StudentID>.jar
|    |--- | output.csv
|    |--- |   ...
|--- README (if needed)
```

You must follow the file structure above and compress it into a ZIP file named <StudentID>.zip

Grading Criteria

Requirements	Points
Exercise 2.1	**4**
- Correct Map phase	1
- Correct Reduce phase	1
- All output rows are correct	1.75
- Results are sorted in the right order	0.25
Exercise 2.2	**2**
- All output rows are correct	1.75
- Results are sorted in the right order	0.25
Exercise 2.3	**4**
- The overlap detection function works	1
- All output rows are correct	2.75
- Results are sorted in the right order	0.25

Happy coding!

./.