

Домашнє завдання до теми «Apache Spark. Оптимізація та SparkUI»

Частина 1

localhost:4040/jobs/				
Apache Spark 3.5.3 Jobs Stages Storage Environment Executors SQL / DataFrame				
Spark Jobs (?)				
User: yuliia				
Total Uptime: 3,1 min				
Scheduling Mode: FIFO				
Completed Jobs: 5				
Event Timeline				
Completed Jobs (5)				
Page: 1				
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total
4	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-1.py:27 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-1.py:27	2024/12/01 21:33:34	32 ms	1/1 (2 skipped)
3	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-1.py:27 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-1.py:27	2024/12/01 21:33:34	0,1 s	1/1 (1 skipped)
2	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-1.py:27 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-1.py:27	2024/12/01 21:33:34	0,1 s	1/1
1	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2024/12/01 21:33:33	0,2 s	1/1

Результат:

Job 1: Spark ініціалізує завантаження даних із диска в DataFrame. Це включає визначення схеми даних та читання рядків із CSV-файлу.

Job 2: Spark переносить дані між розділами, щоб забезпечити рівномірний розподіл. Це корисно для паралельного обчислення.

Job 3: Spark виконує всі перетворення:

- Фільтрує рядки, де `final_priority < 3`.
- Вибирає стовпці `unit_id` і `final_priority`.
- Групує дані за `unit_id` і рахує кількість записів у кожній групі.

Job 4: Spark обчислює результат фільтрування, вибираючи тільки ті групи, де `count > 2`.

Job 5: Spark виконує всі попередні трансформації, якщо вони ще не були виконані, і збирає результати у формат, який може бути використаний у Python.

Частина 2

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total
7	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:30 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:30	2024/12/01 21:40:53	24 ms	1/1 (2 skipped)
6	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:30 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:30	2024/12/01 21:40:53	17 ms	1/1 (1 skipped)
5	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:30 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:30	2024/12/01 21:40:53	35 ms	1/1
4	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:25 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:25	2024/12/01 21:40:53	30 ms	1/1 (2 skipped)
3	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:25 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:25	2024/12/01 21:40:53	0,1 s	1/1 (1 skipped)
2	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:25 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-2.py:25	2024/12/01 21:40:53	0,1 s	1/1
1	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2024/12/01 21:40:52	0,2 s	1/1
0	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2024/12/01 21:40:52	0,2 s	1/1

Чому при додаванні однієї проміжної дії `nuek_processed.collect()`, отримано аж на 3 Job більше?

Додавання проміжного `collect()` у код створює значний вплив на роботу Spark. Це пов'язано з тим, що `collect()` є дією (**action**), яка змушує Spark завершити виконання всіх попередніх трансформацій та завантажити результат у локальну пам'ять.

В результаті Spark двічі виконує обчислення:

1. Перший `collect()` викликає обчислення для всіх попередніх трансформацій (до `groupBy` і `count` включно).
2. Другий `collect()` запускає обчислення ще раз для додаткової фільтрації `where("count > 2")`.

Тому кількість Jobs зросла:

- 1 Job для читання CSV-файлу.
- 1 Job для `repartition`.
- 1 Job для фільтрації `where("final_priority < 3")`.
- 1 Job для `select + groupBy + count`.
- 1 Job для першого `collect()`.
- Після першого `collect()` Spark завершує всі попередні обчислення.
- Далі, при другому `collect()`:
 - 1 Job для фільтрації `where("count > 2")`.
 - 2 Jobs для повторного обчислення групування (`groupBy + count`) і фільтрації.

Частина 3

← → ↺

localhost:4040/jobs/

APACHE

spark

3.5.3

Jobs

Stages

Storage

Environment

Executors

SQL / DataFrame

Spark Jobs (?)

User: yuliia

Total Uptime: 1,2 min

Scheduling Mode: FIFO

Completed Jobs: 7

Event Timeline

Completed Jobs (7)

Page: 1

Job Id ▾	Description	Submitted
6	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:31 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:31	2024/12/01 21:45:42
5	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:26 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:26	2024/12/01 21:45:42
4	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:26 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:26	2024/12/01 21:45:42
3	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:26 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:26	2024/12/01 21:45:42
2	collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:26 collect at /Users/yuliia/Desktop/Projects/goit-de-hw-04/task-3.py:26	2024/12/01 21:45:42
1	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2024/12/01 21:45:41
0	csv at NativeMethodAccessorImpl.java:0 csv at NativeMethodAccessorImpl.java:0	2024/12/01 21:45:41

Page: 1

Чому при використанні cache() ми зменшили кількість Job?

1. Перше виконання дій на закешованому DataFrame (наприклад, перший collect()) включає всі обчислення до точки кешування та зберігає результат. Це 5 Jobs.
2. Подальші дії на закешованих даних виконуються без повторних обчислень і лише на закешованих результатах. Це зменшує додаткові Jobs до 2 (для фільтру where("count > 2") та фінального collect()).

Загалом: 7 Jobs.