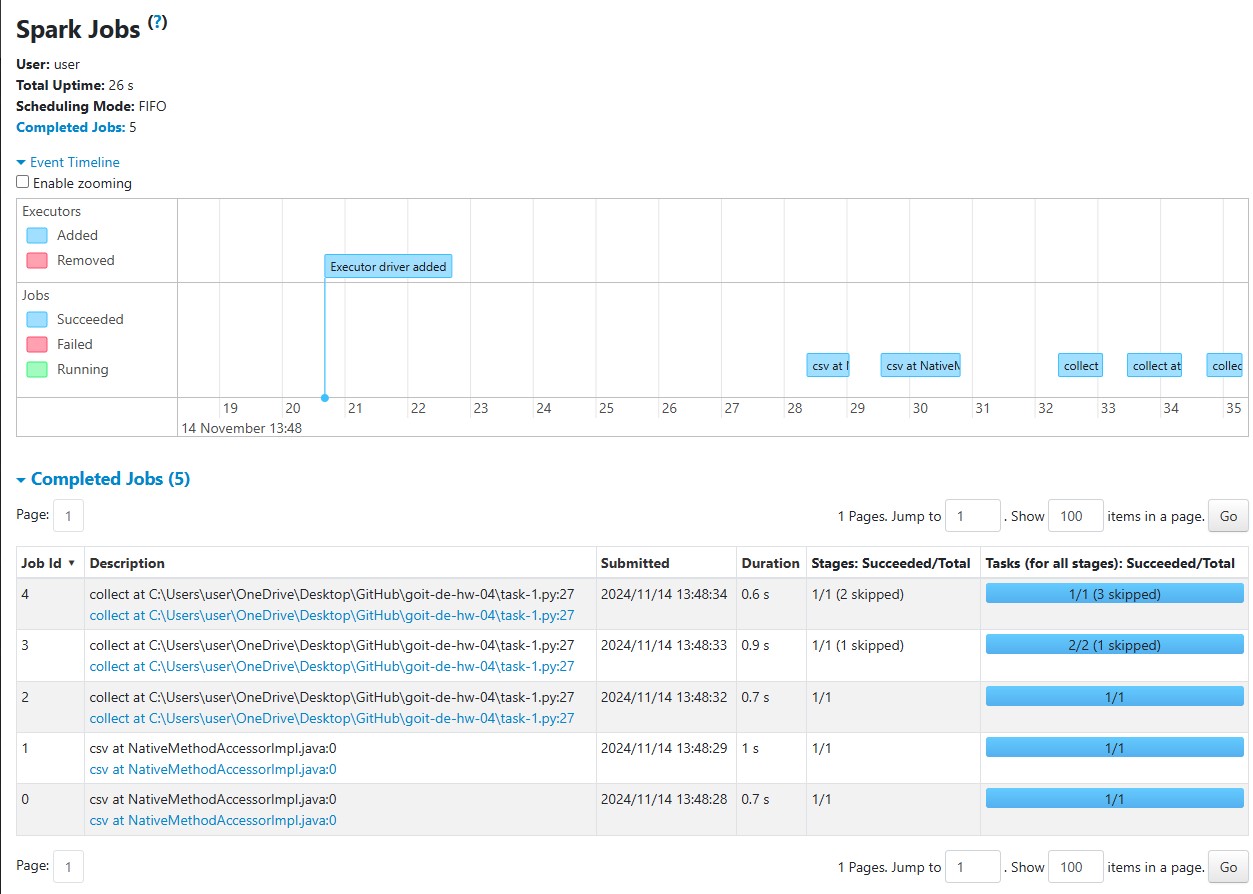
**# HW | Apache Spark | Optimization and SparkUI**

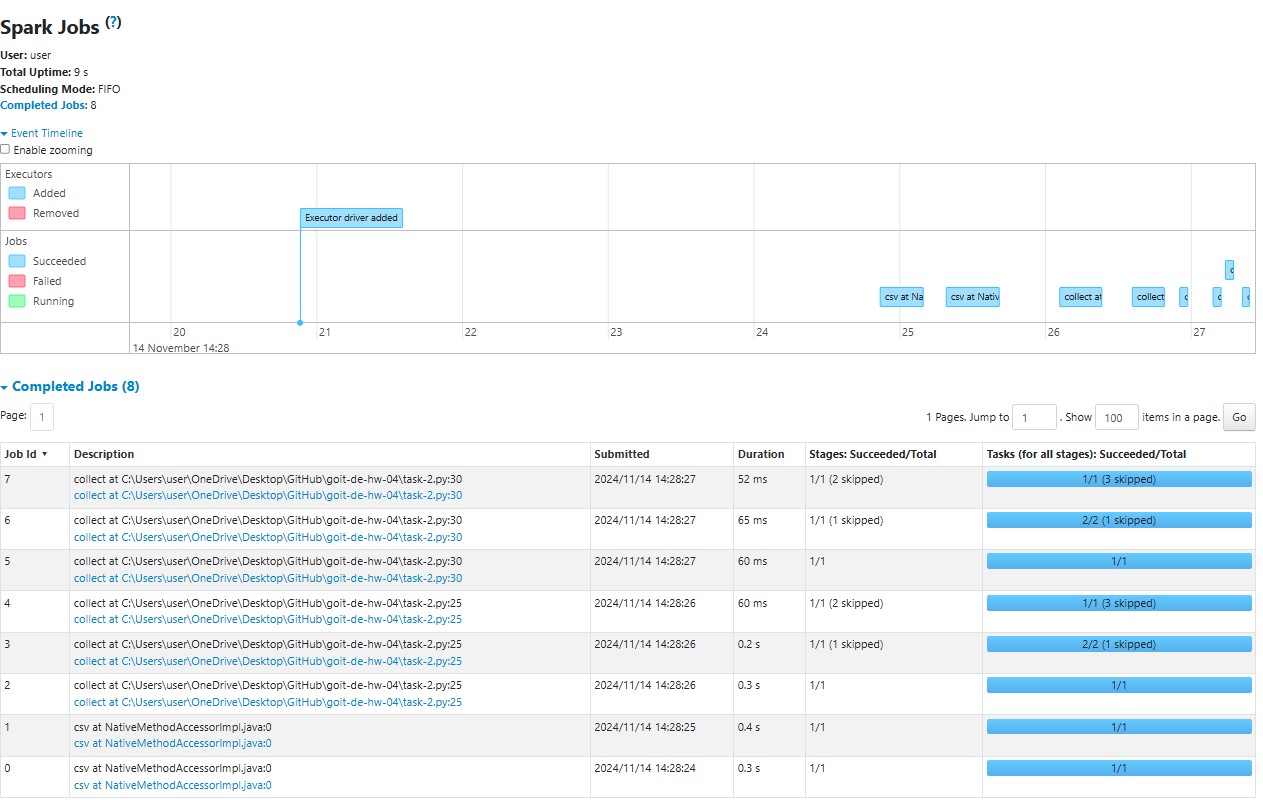
**Task 1**

****

1. **Зчитування CSV-файлу** (nuek\_df = spark.read...) запускає перший Job, оскільки Spark ініціалізує завантаження даних із диска в DataFrame.
2. **Перерозподіл на 2 розділи** (nuek\_repart = nuek\_df.repartition(2)) також створює **новий Job**. Це тому, що Spark має перенести та перерозподілити дані, створюючи новий розподіл.
3. **Фільтрування та групування** (nuek\_processed = nuek\_repart...groupBy("unit\_id").count()) запускає **третій Job**. Spark виконує обчислення для where та groupBy, щоб відфільтрувати й згрупувати дані за unit\_id
4. **Додаткова фільтрація** (nuek\_processed = nuek\_processed.where("count>2")) створює **четвертий Job**, оскільки цей рядок викликає додаткове обчислення, щоб вибрати записи, де count > 2
5. **Функція collect()** виконує **п’ятий Job**. collect() є дією (action), яка змушує Spark виконати всі попередні перетворення й зібрати дані у форматі, який можна вивести.

Отже, цей код запускає п’ять Jobs, кожен з яких відповідає одному з ключових обчислювальних етапів: читання, перерозподіл, групування, додаткове фільтрування та фінальне збирання даних.

**Task 2**



**Чому при додаванні однієї проміжної дії nuek\_processed.collect(), отримано аж на 3 Job більше?**

У цьому коді додавання проміжної дії nuek\_processed.collect() призводить до появи додаткових 3 Jobs, і в результаті загальна кількість Job досягає 8. Ось чому це відбувається:

1. **Додавання проміжної дії collect**: У Spark кожна дія (action) змушує викликати всі перетворення (transformations), які передують цій дії. Оскільки ви додали collect() між обчисленнями, Spark обробляє проміжні перетворення, викликаючи нові Jobs для виконання цього collect.
2. **Lazy Evaluation (Ліниве обчислення)**: У Spark перетворення не виконуються відразу, вони накопичуються до моменту, коли потрібна дія. У цьому випадку, перша дія nuek\_processed.collect() змушує Spark виконати всю послідовність перетворень до цього моменту.
3. **Поява додаткових Jobs**: Spark виконує ще одну обробку тих же даних після nuek\_processed = nuek\_processed.where("count>2"), оскільки ця зміна після першого collect() також вимагає повторного обчислення.

Ось як розподіляються Jobs у даному коді:

* **Job 0 та Job 1**: Завантаження CSV-файлу (csv) для створення DataFrame (nuek\_df).
* **Job 2, Job 3, та Job 4**: Обробка перших двох collect() дій, включаючи фільтрацію, вибірку, групування та підрахунок.
* **Job 5, Job 6, та Job 7**: Повторне обчислення для останньої collect(), яка викликається після додаткової фільтрації where("count>2").

Таким чином, додавання першого collect() створює ще три додаткові Jobs, оскільки це діє як проміжна дія, змушуючи Spark обробляти дані двічі: один раз до додаткової фільтрації і ще один раз після неї.

**Task 3**

A screenshot of a computer

Description automatically generated

**Чому при використанні cache() ми зменшили кількість Job?**

Використання cache() зменшило кількість Job, оскільки cache() зберігає проміжні результати в пам’яті після першого обчислення. Це дозволяє уникнути повторних обчислень при наступних діях (actions), таких як collect().

1. **Кешування проміжного результату**: Після виконання count(), результат зберігається в пам’яті.
2. **Зменшення повторних обчислень**: Подальші виклики collect() використовують кешовані дані, і Spark не перераховує ланцюжок перетворень.
3. **Оптимізація**: Завдяки кешуванню кількість Job скорочується з 8 до 7, оскільки Spark використовує вже обчислені дані.