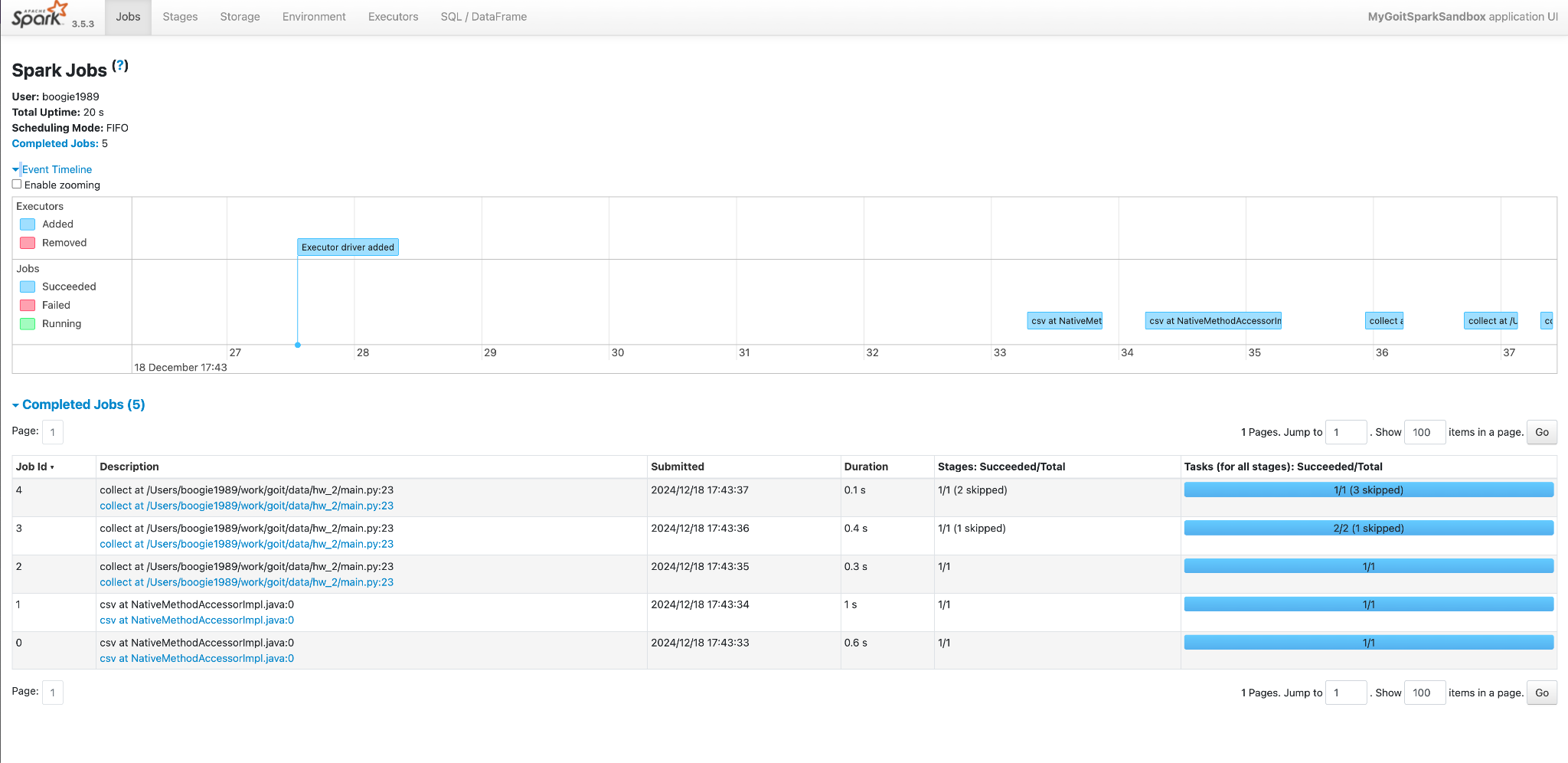
**Screenshot\_1 (5 Jobs)**

****

In this scenario, we performed a series of operations on the DataFrame and executed a single final action (collect), resulting in 5 jobs.

**Job #0:**

* Reads 64 KiB of data from the CSV file.
* Tasks include scanning the text file, applying WholeStageCodegen optimizations, and running mapPartitionsInternal.

**Job #1:**

* Reads 453.2 KiB of data from the CSV file.
* Similar tasks as Job #0, plus DeserializeToObject and SQLExecutionRDD for deserialization and SQL execution.

**Job #2:**

* Introduces the first data exchange operation (shuffle).
* Tasks include a CSV scan, WholeStageCodegen, and an Exchange node.

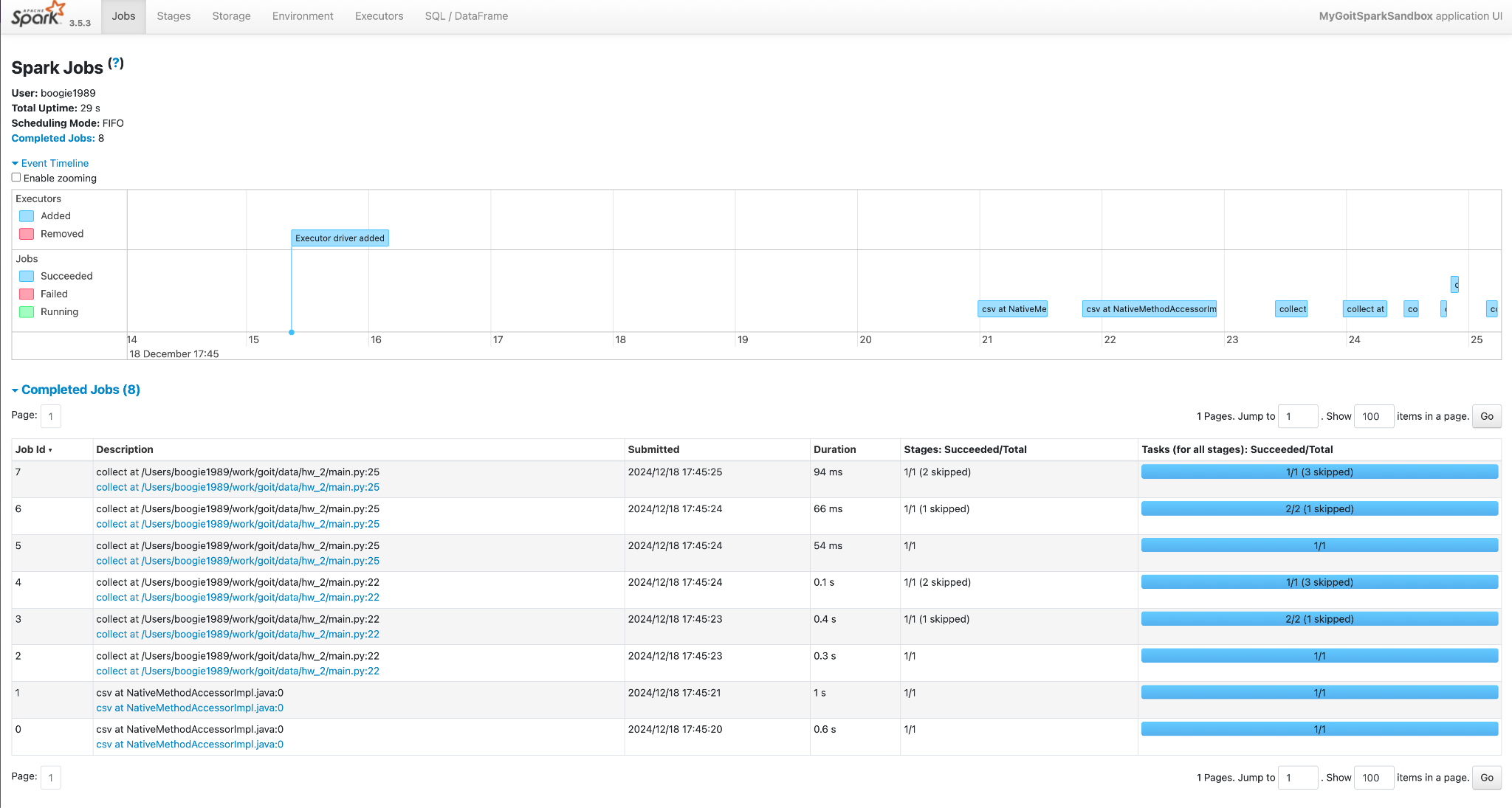
**Job #3:**

* Involves data exchanges and a second shuffle.
* Some tasks are skipped because they were computed in previous stages. The essential tasks include Exchange and WholeStageCodegen.

**Job #4:**

* Completes the final aggregation and filtering steps.
* Reuses previous computations, resulting in some skipped tasks.
* Main tasks: AQEShuffleRead, WholeStageCodegen, and mapPartitionsInternal to produce the final result.

**Screenshot\_2 (8 Jobs)**

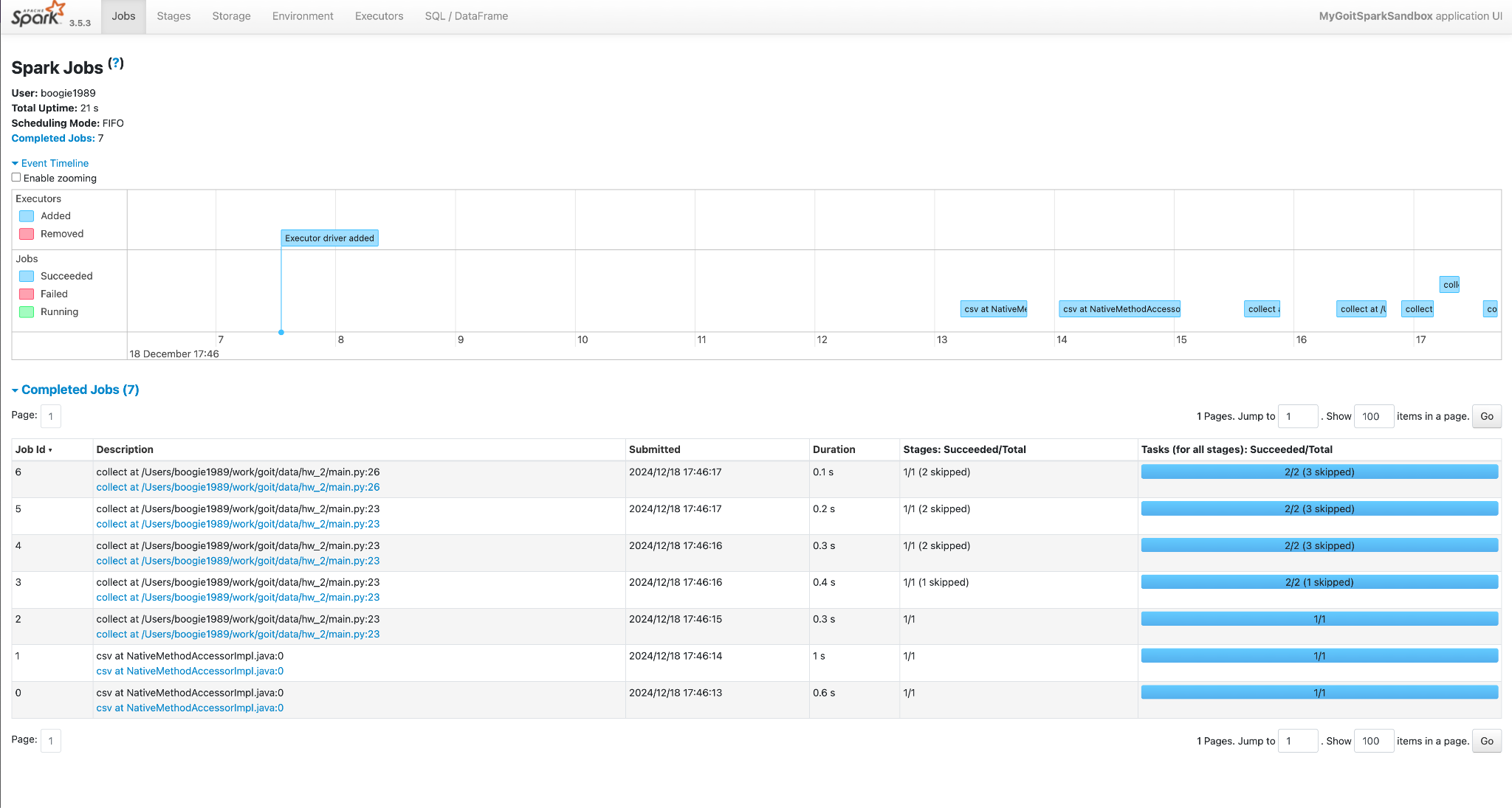
****

In this run, an extra collect action is inserted in the middle of the processing chain, causing Spark to perform more transformations earlier and increasing the total number of jobs to 8.

**Key Difference:**

* The intermediate collect action forces Spark to materialize intermediate results sooner.
* Each collect action involves multiple stages (reading, shuffling, aggregating), repeating computations and increasing job count.
* The additional three jobs (compared to Screenshot\_1) occur because transformations and shuffles happen twice: once before the intermediate collect and once before the final collect.

**Screenshot\_3 (7 Jobs)**

****

Here, we added a cache operation before running the actions. Caching stores certain transformations in memory, so subsequent actions do not need to re-read or recompute previous steps, reducing the total number of jobs to 7.

**Key Difference:**

* The cached DataFrame eliminates the need to re-perform all previous computations.
* With caching, fewer jobs are triggered after the initial computation.
* The reduction from 8 jobs to 7 (compared to the scenario in Screenshot\_2) occurs because Spark can skip expensive stages, leveraging cached results.

**In Summary:**

* **Screenshot\_1:** A single sequence of transformations plus a final collect results in 5 jobs.
* **Screenshot\_2:** Adding an extra intermediate collect increases the total to 8 jobs due to repeated computations.
* **Screenshot\_3:** Using cache reduces redundant work and lowers the total to 7 jobs, even with multiple actions.