NYC Yellow Taxi Dataset

Performance Comparison of Pandas, SQLite, and DuckDB

Max Boone (s2081318)

October 21, 2024

1 Introduction

This report provides a short analysis of the performance differences between processing and analyzing the NYC Cab dataset from 2016. All analyses are done through Pandas, SQLite and DuckDB. Experiments are executed on a CAX41 machine from Hetzner Cloud. This machine has 32 GiB ECC RAM and 16 Neoverse N1 cores. The report is structured as follows. First, the data loading step is reviewed. Second, two queries that test general performance of the database are run and discussed. Finally, a simple linear model on the data is calculated and discussed.

2 Data Loading

The dataset was provided as a parquet file. Pandas and DuckDB offer native functionality to load parquet files into the internal data structure. SQLite does not offer this functionality it self, and the data is first loaded into memory by pandas before writing to SQLite. Only the SQLite part is measured there. The execution times are as follows:

- **Pandas:** 1.30 seconds
- **SQLite:** 109.71 seconds
- **DuckDB:** 12.70 seconds

DuckDB and SQLite store the resulting database file on disk and this already shows some difference in performance. The resulting SQLite database is around 1.2 GiB in size and the DuckDB database around 211 MiB. For a fairer comparison, and to rule out disk overhead, a cursor for SQLite and DuckDB was also set up in-memory.

- **SQLite (Memory):** 106.50 seconds
- **DuckDB (Memory):** 5.97 seconds

One of the reasons that SQLite is this slow in data loading might be that pandas presents the entire dataset in one query to the SQLite engine instead of in batches that can be written asynchronously to the WAL.

3 Query Performance

3.1 Query 1: Distinct Counts

The first query counts the distinct values for each dataset, the execution times are as follows:

- **Pandas**: 2.80s
- **SQLite**: 37.34s
- **SQLite (Memory)**: 37.79s
- **DuckDB**: 0.53s

- **DuckDB (Memory)**: 0.47s

DuckDB is a clear winner in this section, as is expected from the columnar storage format of the database. Likely the strategy that it uses to count these columns is as efficient as Pandas, but it can partition the query and use multiprocessing. SQLite likely retrieves every row for every column which causes a lot of duplicate values to be loaded. It is however strange that the in-memory database does not perform better, possibly due to the amount of available memory the caches are doing their work well.

3.2 Query 2: Aggregation and Grouping

The second query aggregates data from the dataset and gives descriptives for each day-hour combination. The execution times are as follows:

- **Pandas**: 0.93s
- **SQLite**: 13.51s
- **SQLite (Memory)**: 13.26s
- **DuckDB**: 0.03s
- **DuckDB (Memory)**: 0.03s

Again the queries from DuckDB are the fastest, likely this is due to the implementation of the fully parallelized aggregate hash table that DuckDB employs. Other time differences are likely due to the same reasons as discussed in the previous query.

4 Machine Learning: Fare Estimation

The final performance comparison is done through the computation of a linear regression. Essentially, this consists of two steps. First, the data points are filtered to the range from zero to a z-score of 3 (in other words, 3 standard deviations from the mean). Then, the covariance over variance gives the required input (beta) to calculate a linear regression. The execution times to filter the data and calculate the regressions are as follows:

- **Pandas**: 1.13s
- **SQLite**: 8.24s
- **SQLite (Memory)**: 7.59s
- **DuckDB**: 0.08s
- **DuckDB (Memory)**: 0.07s

Again, the performance of DuckDB blows SQLite and Pandas out of the water. One of the possible differences in performance is that all the calculations for DuckDB can be done in a single query to the query planner while SQLite and Pandas treat these as separate steps.

5 Conclusion

The tests show that DuckDB offers the fastest implementation for the analytical workloads tested in this assignment. The first-class support for file formats, various aggregation functions and the ease of testing query in the separate CLI / REPL lifts a barrier moving from analytical development kits such as R and libraries such as Pandas. The lack of difference with in-memory databases shows use of good caching of both SQLite and DuckDB, to further test where performance gains are realized it would be useful to run the tests with a larger than memory dataset and investigate the outcomes from the engines' query planners.