OMOP-CDM ETLs with Dask and Prefect

Freija Descamps, Roberto Guarnieri, Lars Halvorsen, Jared Houghtaling

Background

For each OHDSI data harmonization project, a dedicated ETL is typically designed that transforms the data from the original source format into the OMOP-CDM model. The exact tools that are used depend on many factors. First, the format and size of the source data can greatly impact the design decisions. For example, if the source data is already loaded into a relational database that is accessible to the ETL itself, then a combination of SQL tasks directed by a workflow manager is likely a good choice. However, if the data presents itself in a very large number of CSV files, then it is possible that the SQL approach is not optimal.

Additional factors that will influence design choices are the available resources on the host where the ETL is run, the available expertise and preference of the data-source partner, the ETL execution frequency and whether the ETL performs an update or an overwrite of the target OMOP-CDM tables.

A typical ETL will broadly consist of a set of tasks that need to be performed in a well defined order and an executor that handles that workflow. For ETLs that are written in Python, a data-wrangling library like Pandas[1] can perform easy and fast operations on small datasets that comfortably fit in memory. For larger datasets and computation-intensive, but parallelizable, transformations, open-source Python libraries like Dask[2] (parallel computing library) and Prefect[3] (automation and scheduling engine) are a powerful option to further optimize these Python ETLs.

Methods

To test the tools discussed here, we have created a proof-of-concept ETL implementation of a Dask+Prefect, where large synthetic dataset (Synthea[4]) is partially mapped to the OMOP CDM model. A typical ETL consists of a set of steps (tasks) that need to be executed in a specific order (workflow). A small toy example workflow for Synthea is shown in Figure 1.

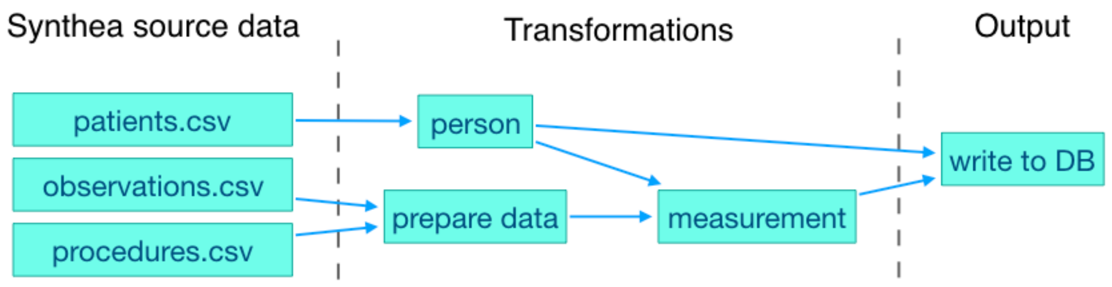

Figure 1. A toy workflow example for a small part of the Synthea[4] dataset. The blue arrows indicate directional dependencies. In this example, the measurement transformation requires the person transformation to have finished. The database task is executed after all transformations have finished.

Results

In the case of an OMOP-CDM ETL on Synthea data, an example task would be to extract and transform the person details from the patient details contained in the patients.csv source file. This task needs to happen early in the ETL since other tasks require access to the generated numeric person_id.

The Prefect Python library provides tools for building and running data workflows. At a high level, it allows to define a series of interdependent discreet tasks. It also includes a task-library that covers common tasks like for example tasks for executing queries against a Postgres, Redis or other databases. Simply by defining all task dependencies, Prefect can generate and execute the workflow, launching tasks in parallel where possible.

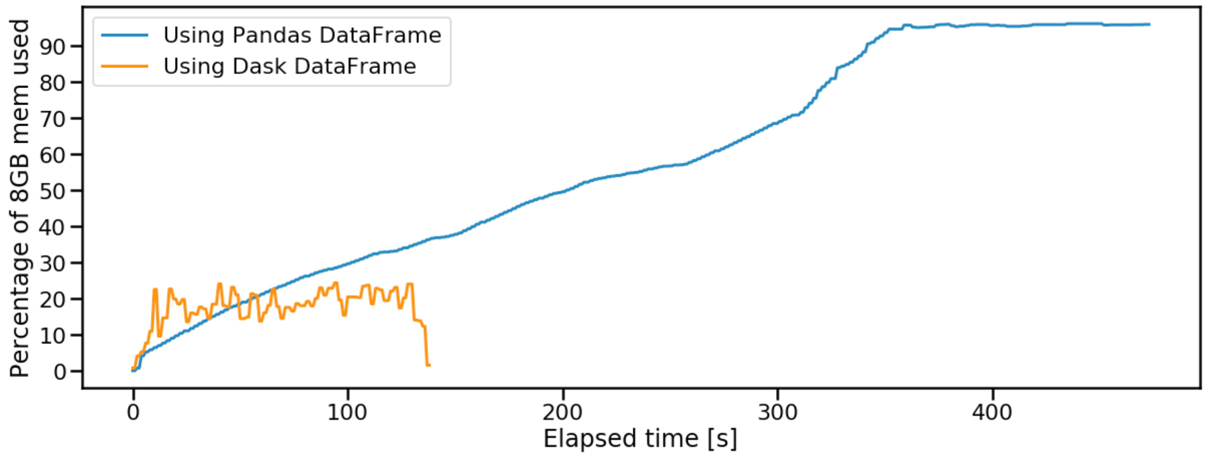

Figure 2. Memory use as a function of time for the loading of a large Synthea observation datafile (over 8Gb). The implementation using Dask can handle data sizes that are larger than the available memory (orange, completed), whereas the Pandas implementation runs out of memory (blue, failed).

Dask is an open-source Python library for parallel computing. Among other things, it provides an extension to the Pandas DataFrame (2D structured data collection). The Dask DataFrame consists of a collection of smaller Pandas DataFrames which can be located on disk. This allows Dask to handle operations on datasets that are larger than the available memory as well as to parallelize computations across the constituent Pandas DataFrames.

By combining Prefect and Dask, an ETL can be transformed from a set of tasks that are performed in sequence and on a single core (slow and limited by available memory) to a workflow where tasks as well as computations can be run in parallel on datasets that do not fit in the available memory. Dask also allows loading data directly from databases into Dask DataFrames.

Conclusion

The specific design and implementation of an ETL can take many forms. When using Python, open-source parallel processing libraries like Dask and scheduling engines like Prefect can increase the performance by accommodating parallel execution of tasks as well as computations.

References/Citations

1. Wes McKinney. Data structures for statistical computing in Python. Proceedings of the 9th Python in Science Conference. 2010;51-56.
2. Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. Proceedings of the 14th Python in Science Conference. 2015;130-136.
3. Prefect Development Team. PrefectCore: an open-source automation and scheduling engine. https://www.prefect.io
4. Jason Walonoski, Mark Kramer, Joseph Nichols, Andre Quina, Chris Moesel, Dylan Hall, Carlton Duffett, Kudakwashe Dube, Thomas Gallagher, Scott McLachlan. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. Journal of the American Medical Informatics Association, Volume 25, Issue 3, 2018;230–238.