OMOP-CDM ETLs with Dask and Prefect

Freija Descamps, Roberto Guarnieri, Lars Halvorsen, Jared Houghtaling
edenceHealth NV, Belgium
Contact: freija@edence.health

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 design choices and tools used depend on many factors, like for example the format and size of the source data, 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

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, see Figure 1.

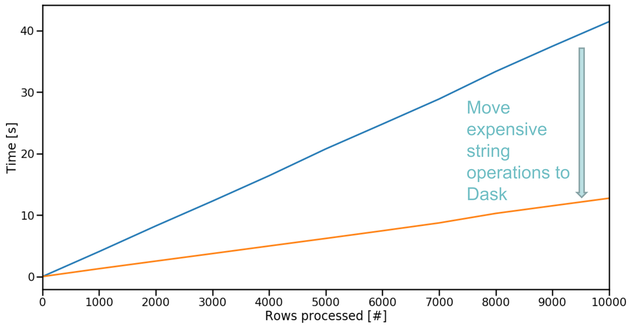

Figure 1. An example plot used for performance profiling illustrating the performance gain by moving a parallelizable computation from Pandas (blue) to Dask (orange).

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.
We tested the tools discussed here by implementing a Dask+Prefect ETL that partially transforms large synthetic dataset (Synthea[4]) to the OMOP CDM model. A small toy example workflow for Synthea is shown in Figure 2.

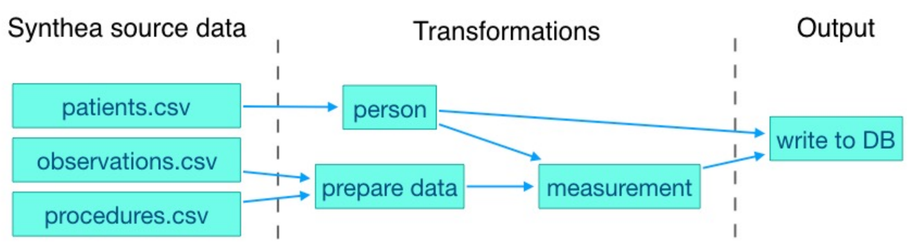

Figure 2. A toy workflow example for a small part of the Synthea[4] dataset. The blue arrows indicate directional dependencies. In this example, the measurement transformation requires the person transformation to have finished. By implementing a Prefect Flow, the 'person' and 'prepare data' tasks will automatically be scheduled in parallel if resources allow.

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.

Example use-case

We describe here a real-world example of a Dask OMOPCD-CDM ETL. The original ETL was designed for 500Mb of source data which was made available in a set of csv files. As the data fit comfortably in memory, the ETL was designed to use Pandas and it included expensive and slow row-wise operations. This original ETL ran within 2 hours inside an 8Gb Docker container. The original performance requirements were met.
When the data-size was increased to 5GB, a faster and more memory-efficient implementation was required. In this case, Dask was used both for memory-optimization and parallelization of computations, both were needed in order to meet the performance requirements.

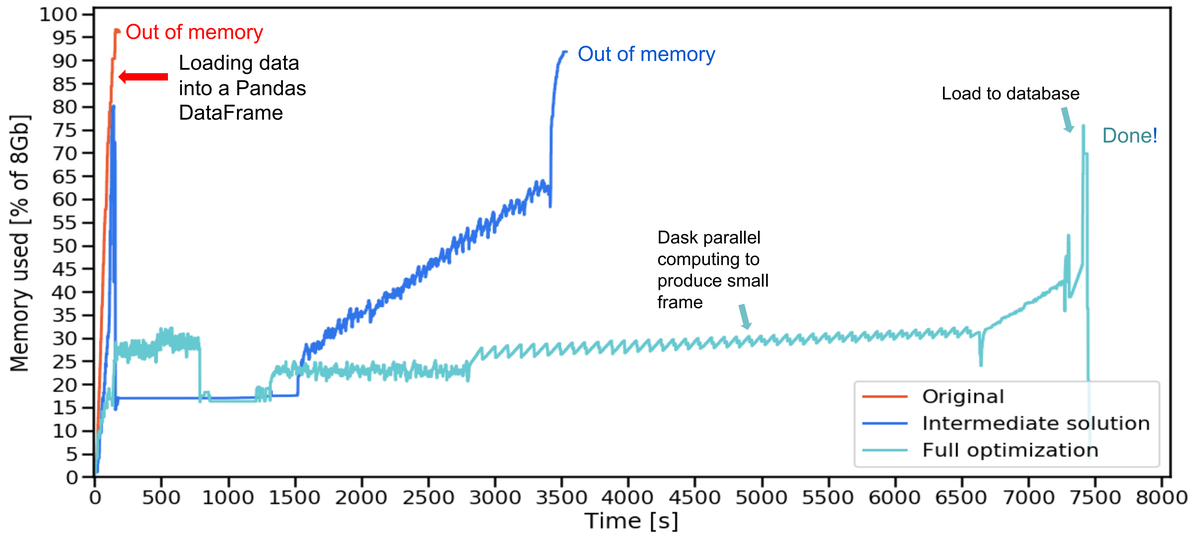

Figure 2. Percentage of memory used (8Gb Docker container) versus time for the original Pandas ETL (orange, runs out of memory when loading data), an intermediate optimization (dark blue, runs out of memory when computing an intermediate frame) and the full optimization (light blue, executes).

Conclusions

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

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 Delopment 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.