Highly scalable patient-at-a-time transformation of observational databases into OMOP CDM v5 format using cloud-based open source tools

Marzieh Golbaz, MS[1], Donald O'Hara, MS[1], Mohammad Azimi, PhD[1], Steve Lyman, BS[1], Stephanie Reisinger, BS[1]
[1] Evalytica, Inc., San Francisco, CA

Background

The use of a common data model (CDM) such as the OMOP CDM provides many advantages, including the ability to perform systematic analysis of disparate observational databases and the ability to develop a library of standard analytic routines that have been written based on a single common format. A major bottleneck prior to performing any analysis is the transformation of very large databases (500+ million person years) from their original format to that of the CDM in a process that is commonly referred to as ETL (Extract, Transform and Load).

This transformation process is commonly implemented using SQL code and run on a large DBMS. Executing set-based operations on these large datasets in a time-performant manner requires enterprise-sized database servers, database administration and SQL programming expertise to optimize complex transformation steps. Even then, this approach can result in a transformation process that can take several days.

Here we show the benefits of two key design decisions that can lead to significant improvements in transformation time over the above mentioned approach:

1. Patient-at-a-time processing – this approach takes advantage of the independence of data between any two patients to provide maximal horizontal partitioning of datasets to uniformly distribute work across thousands of CPUs.
2. Cloud architecture – utilizing cloud architecture allows for a quick yet temporary "burst" of computational resources to rapidly process a large amount of data on performant hardware, with the ability to offload the results to more cost-effective storage for long-term use.

Methods

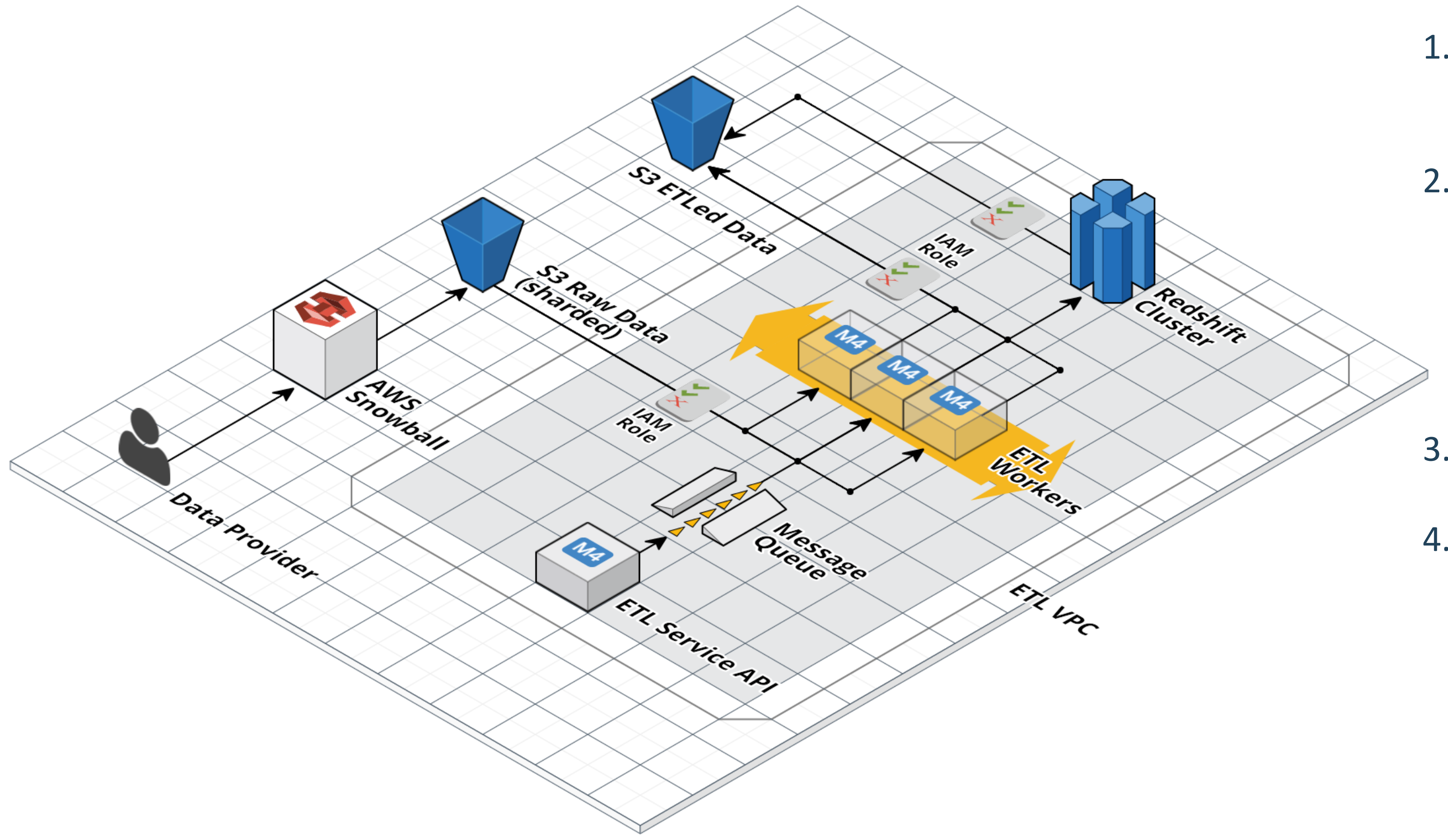

Figure 1: ETL architecture utilizing AWS(Amazon Web Services). S3 (Simple Storage Service), EC2 (Elastic Compute Cloud), ASG (AutoScaling Group), VPC (Virtual Private Cloud), IAM (Identity and Access Management), Redshift and Snowball Appliance.

1. Untransformed data is physically transferred to S3 via an AWS Snowball Appliance and horizontally partitioned (sharded).
2. Transformation is initiated by placing a series of messages on the queue – each message identifies a set of patients to be transformed (100k+) on a single "worker" (EC2 instance) along with a unique range of IDs reserved for these patients and their records.
3. EC2 instances are deployed within an ASG that can support 1 to hundreds of workers.
4. These EC2 instances:
   a) Pull messages from the queue (RabbitMQ)
   b) Retrieve the associated raw data from S3.
   c) Perform transformation on a single patient at a time using OHDSI specs implemented in Python.
   d) Store CDM format transformed data back to S3.
   e) (optionally) Load CDM data to Redshift.

Results

1. We have executed our transformation process against the 500+ million person year dataset twice, varying the number of concurrent workers used and obtained the run times reported in Table 1.
2. A primary bottleneck within the architecture was the use of cost-effective Amazon S3 service for long term storage and retrieval of transformed data. S3 automatically handles the placement of files across partitions based on name, resulting in the placement of files with sequential names which are typically accessed simultaneously on the same partition. Simultaneous requests from thousands of processes quickly overwhelm a single partition and result in slowdowns in execution. By hashing filenames prior to storage and retrieving the hashed filenames, we evenly distribute our requests across up to 10,000 partitions and achieve request rates of millions of files per second that should allow us to go far beyond 200 simultaneous workers and reduce transformation time to under an hour.
3. We determined that the Python programming language is well suited for the implementation of OHDSI published ETL specifications, providing ease of debugging and automated testing when compared to set-based SQL logic.

Table 1: Transformation times based on workers allocated

Nodes	Wall Time
25 concurrent workers	8 hours
50 concurrent workers	4 hours
200 concurrent workers* (planned; not yet executed)	1 hour (estimated)

Conclusions

- We have demonstrated a significantly reduced wall time for ETL transformation, using open source tools executing in the AWS cloud infrastructure.
- Complex set-based manipulation of large volumes of data using SQL code can be replaced by much simpler person-at-a-time logic implemented in procedural languages (Python in our case).
- This approach utilizes on-demand infrastructure and can be scaled as needed to meet ETL processing time requirements.

A similar approach to the two points presented in the Background can be adopted when implementing analytic routines for CDM transformed data. The assumption of independence of data between patients remains valid, enabling the application of patient-at-a-time processing. Combined with the ability to rapidly scale the number of workers based on analytic demand allows for the generation of insights on CDM data of this scale in under an hour. The ability to develop these analytic applications using procedural programming languages provides significant advantages in reduced development time, simpler debugging and quality controls through ease of unit and integration testing.