How to Make Data Pipeline Scalable: A Case Study for Healthcare Data Analytics

Atakan Gelecek, Pinar Karagoz
Ismail Hakki Toroslu
Middle East Technical University,
Computer Eng. Dept.
Ankara, Turkey

Stephan Hachinger
Leibniz Supercomp.Centre(BAdW-LRZ)
Garching b.M., Germany

Jan Martinovic
IT4Innovations, VSB - TU Ostrava
Czech Republic

Motivation and Objectives

Motivation

- Large healthcare datasets require reliable orchestration, scalable ingestion, and repeatable analytics.
- Passing large intermediate data through orchestration metadata mechanisms can hit size/memory limits.

Objectives

- To analyze distributed analytics pipeline through a case study that ingests healthcare demographic data and produces two demographic visualizations.
- Identify performance bottlenecks and characterize how runtime changes with CPU/RAM and batch settings.

Pipeline and Architecture

Case study: Access demographic information of citizens from a national healthcare data collection.

Visual analytics produced:

- **Age–gender distribution:** counts by gender across 6-year age bins (bar chart).
- **Location cohort heatmap:** counts by location (hotel) and age bin (heatmap; shown for Top-20 locations and summed over genders).

Pipeline design: The workflow is orchestrated with Apache Airflow and uses Kafka for bounded-memory data exchange between tasks.

Hybrid Environment and Dask Optimization:

- Tested in hybrid environments combining HPC systems and cloud infrastructure.
- Used Dask for parallelizing PostgreSQL queries, improving access speed and processing efficiency for large datasets.

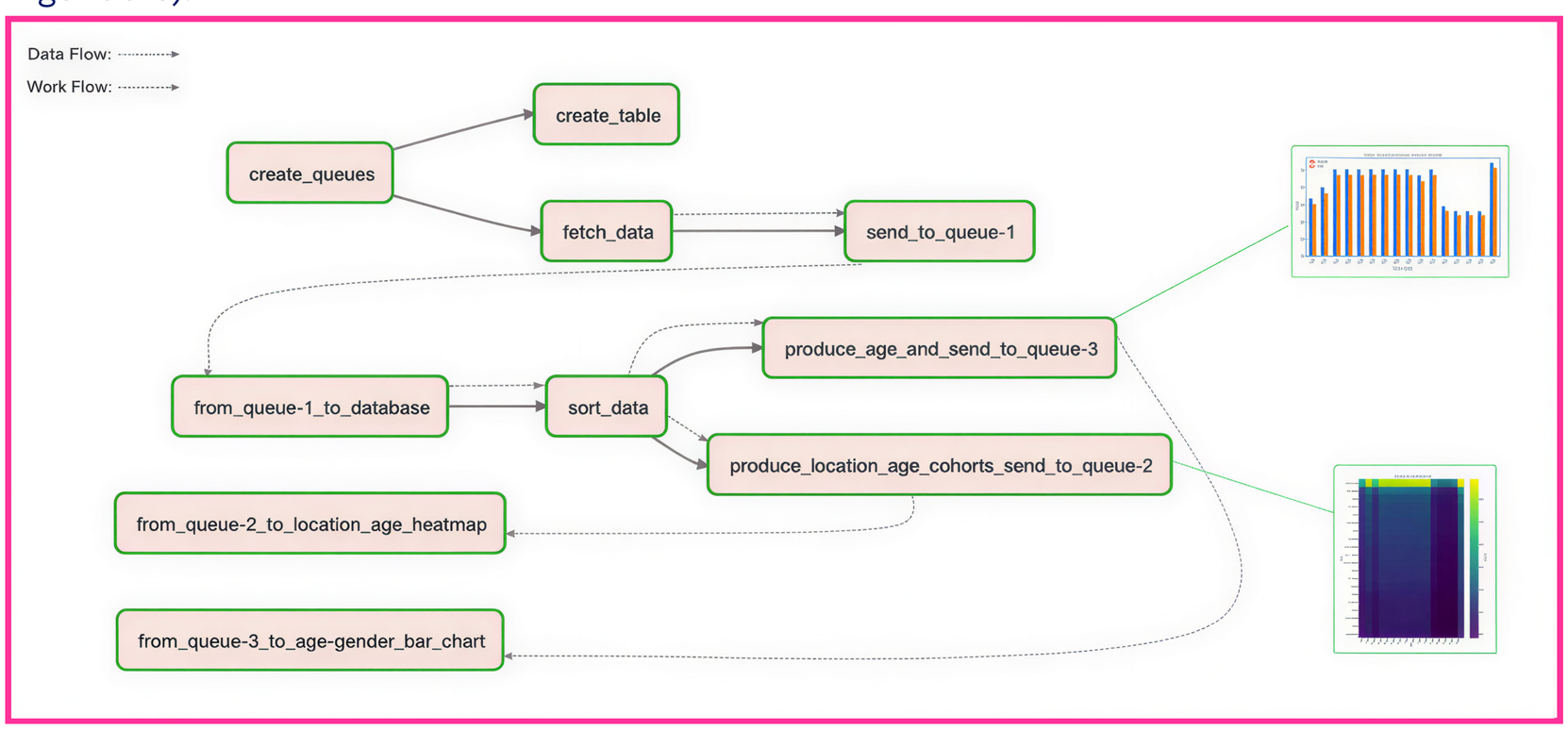

Figure 1: Airflow DAG and Kafka-based data and work flow in the healthcare pipeline.

Why Airflow: Chosen for DAG-based orchestration with scheduling, retries, and monitoring, and for straightforward integration of our analytics steps as pipeline tasks.

Why Kafka: Chosen because it streams and buffers data between Airflow tasks, which avoids large in-memory transfers and prevents hitting Airflow task data-size limits.

Benchmark setup: Runs on the LRZ cloud, varying CPU cores, RAM, and batch size.

Performance Results and Conclusions

Table 1: Runtime Breakdown per Pipeline Task Across LRZ Cloud Configurations(CPU, RAM, and batch size variations in the LRZ cloud environment, in hrs.min.sec)

Environment	2 Core CPU, 9GB RAM	4 Core CPU, 18GB RAM	4 Core CPU, 18GB RAM	10 Core CPU, 45GB RAM	4 Core CPU, 18GB RAM	10 Core CPU, 45GB RAM
Data Size	2.6GB	2.6GB	2.6GB	2.6GB	7.7GB	7.7GB
Batch Size	10000	10000	100000	100000	100000	100000
Tasks	**Duration**	**Duration**	**Duration**	**Duration**	**Duration**	**Duration**
create_queues	00.00.01	00.00.02	00.00.02	00.00.02	00.00.02	00.00.02
fetch_data	00.00.00	00.00.00	00.00.00	00.00.00	00.00.00	00.00.00
create_tables	00.00.00	00.00.00	00.00.00	00.00.00	00.00.00	00.00.00
send_queue-1	00.48.42	00.19.20	00.15.04	00.19.29	00.40.25	00.29.17
from_queue-1_to_db	00.49.08	00.19.58	00.15.18	00.19.35	00.40.47	00.29.33
sort_data	00.04.11	00.02.02	00.01.59	00.01.14	00.06.01	00.05.33
produce_hotel_age_cohorts_send_to_queue-2	02.16.35	00.31.47	00.25.57	00.11.03	00.56.13	00.43.35
from_queue-2_to_hotel_age_heatmap	03.10.15	00.40.32	00.43.27	00.31.57	01.42.55	01.18.37
produce_age_gender_and_send_to_queue-3	02.15.01	00.39.15	00.34.32	00.24.21	01.14.33	01.01.18
from_queue-3_to_age_gender_barchart	03.08.33	01.00.22	00.51.29	00.44.45	02.01.21	01.36.20
Total Time	03.10.24	01.00.28	00.51.38	00.44.59	02.01.27	01.36.24

- On the LRZ cloud, end-to-end runtime is dominated by ingestion (Kafka → DB) and consumer analytics, while setup and sorting contribute relatively little.
- Increasing resources—especially RAM—consistently reduces runtime, with the strongest gains under heavier settings (larger batches/datasets), highlighting the importance of memory and parallel execution for throughput.
- When the data does not fit in memory, performance is typically limited by database/disk I/O, and when most data is cached, it is typically limited by CPU time for parsing and aggregation.
- Airflow + Kafka enables scalable processing of large healthcare datasets by streaming data between tasks instead of passing large in-memory payloads that can hit Airflow size limits.

This research received the support of the EXA4MIND project, funded by the European Union's Horizon Europe Research and Innovation Programme, under Grant Agreement N° 101092944. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission.

Funded by the European Union