An open-source reference implementation of a federated composable datastack

Yannick Vinkesteijn
Dutch Hospital Data
y.vinkesteijn@dhd.nl

Daniel Kapitan
Dutch Hospital Data, Eindhoven University of Technology

Abstract We describe ...

(intented for publication in something like a *practice* paper in Communications of ACM)

A confluence of developments towards federated analytical data systems

We observe a confluence of various trends into what we call federated analytics data systems (FADS). In Europe, ongoing effort to implement data spaces in various industries, including mobility, healthcare and energy. Given the need for secure and trustworthy data sharing, much work is underway to design and implement FADS On the user side, we see that trends such as data spaces. At the same time, the computer science and engineering has development couple of new concepts and technologies that make it easier to implement FADS:

- A data mesh has been developed as an alternative to centralized data warehouse systems. Data mesh conceptualizes data as a product which can be consumed through an application programming interface (API). Although originally coined in the context of enterprise data platforms, it can be readily extend beyond the boundaries of a single organization into a data space
- Capabilities of edge computing and single-node computing has increased significantly, whereby it is now possible to process up to 1 TB of tabular data on a single node
- The composable data stack provides a way to unbundle the venerable data base into loosely components, thereby making it easier and more practical to implement FADS
- The increasing need for privacy-enhancing technologies (PETs) additionally fuels the development of FADS-related technologies, where technologies such as secure multi-party computation (MPC) are now sufficiently mature to be used on an industrial scale.
- Federated machine learning (or federated learning in short) has matured as a means for decentralized training of predictive models, most notable through weights sharing of deep learning networks

As noted by Perreira et al. (2023), however:

> The requirement for specialization in data management systems has evolved faster than our software development practices. After decades of organic growth, this situation has created a siloed landscape composed of hundreds of products developed and maintained as monoliths, with limited reuse between systems. This fragmentation has resulted in developers often reinventing the wheel, increased maintenance costs, and slowed down innovation. It has also affected the end users, who are often required to learn the idiosyncrasies of dozens of incompatible SQL and non-SQL API dialects, and settle for systems with incomplete functionality and inconsistent semantics.

This paper rises to their call to take a principled, open source approach to FADS aimed at "… standardizing different aspects of the data stack (…) advocating for a paradigm shift in how data management systems are designed", focusing on federated analytics data systems.

1. functional architecture and specification of FADS that integrates various common practices and blueprints from the data engineering community;
2. a reference implementation in the Python-Rust data stack that is quickly emerging as a new *de facto* standard for performant and reliable analytical processing;
3. implementation of functionality specific to healthcare, which provided the impetus for this work (kapitan2025data).

Desiderata of analytical data systems

We take Klepmann (2017) as our starting point, who states that "Many applications today are *data-intensive*, as opposed to *compute-intensive*. Raw CPU power is rarely a limiting factor for these applications—bigger problems are usually the amount of data, the complexity of data, and the speed at which it is changing."

Generically, we want:

Reliability	Scalability	Maintainability
tolerating hardware & software vaults	Measuring load & performance	Operability, simplicity & evolvability
human error	Latency percentiles, throughput	

We focus on analytical data systems, with different patterns from transactional data systems.

Property	Transaction processing systems (OLTP)	Analytic systems (OLAP)
Main read pattern	Small number of records per query, fetched by key	Aggregate over large number of records
Main write pattern	Random-access, low-latency writes from user input	Bulk import (ETL) or event stream

Property	Transaction processing systems (OLTP)	Analytic systems (OLAP)
Primarily used by	End user/customer, via web application	Internal analyst, for decision support
What data represents	Latest state of data (current point in time)	History of events that happened over time
Dataset size	Gigabytes to terabytes	Terabytes to petabytes

Design principles of analytical data systems

These functional requirements lead us to the following design principles

- **Colum-oriented storage and memory layout:** Apache Arrow ecosystem, including Apache Flight
- **Late-binding with logical data models most suited for analytics:** ELT pattern with zonal architecture
  - *staging zone:* hard business rules (does incoming data comply to syntactic standard), change data capture
  - *linkage & conformity zone:* concept-oriented tables, typically following a data vault modeling principle, ascertain referential integrity across resources, with tables per concept and linking tables. Mapping to coding systems.
  - *consumption zone:* convenient standardized views like an event table (patient journey, layout for process mining) and dimenreferention integrity across star schema

Reference implementation with current open source software (OSS) components

At the lower, technical, levels we follow the rationale of the composable data stack - Python and SQL(-like) languages as the *de facto* standard for analytical processing i.e. the most commonly used analytical scripting languages. Where necessary, using Intermediate Representations (IR), any analytical query can be transpiled to the target engine of choice - Single-node compute capable of efficiently processing up to 1 TB of data within tens of seconds (polars, DuckDB), so we do away with distributed processing - Open table formats (Iceberg, Hudi, Delta) and open file formats (parquet, AVRO)

Bringing it all together for federated analytics & machine learning (FAML)

- Local data stations are conceptualized as serverless lakehouses
  - Local ELT pipelines
  - Decentralized (pre-)processing, including quality control upon ingest
  - ...

- For horizontally partioned data, we can apply FAML techniques where only aggregated results are combined centrally
- For vertically partitioned data, we need an intermediate/temporary zone for linking the data
- For both horizontally and vertically partitioned data, we can choose to add PETs, most specifically MPC, as an extra security measure
  - Horizontally partitioned data: one-short FL
  - Vertically partitioned data: linkage in the blind
- standardized approach to mapppings

Bibliography