UC SANTA CRUZ | Baskin Engineering

Skyhook: Managing Columnar Data Within Storage

Jayjeet Chakraborty, Carlos Maltzahn

University of California, Santa Cruz

CROSS
cross.ucsc.edu

Problem

- Exploratory data analysis requires a dataset to be viewed in different ways
  - Derived columns
  - Different subsets of columns
- Current practice: datasets are copied to create new views
  - Increases overall analysis time
  - Uses up unnecessary space
  - Requires manual work to relate copies to each other
  - Difficult to keep track of how the data evolved over time
  - Makes workflows more complex

Our Vision

- Dataset “repositories” that support different views over the same data without extra copying
  - Process data directly over data lakes without ingesting into databases
- Support version control of data through time travel, roll back, schema evolution using transactions
- Create and track views and their provenance
- Ability to join (e.g. different compression levels of) datasets
- Scalable and distributed data processing
- And, reduced data movement over the entire system

Challenges with High Energy Physics Data

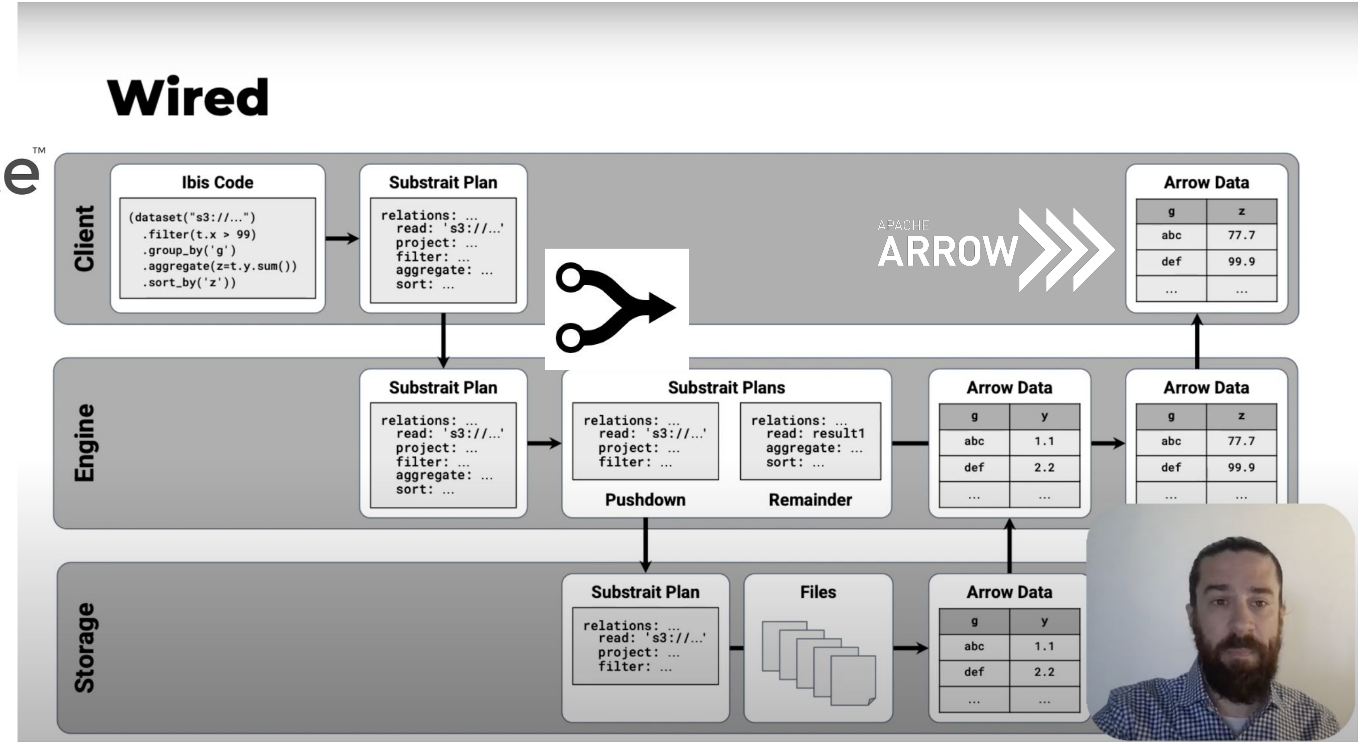
- Huge sized data, large number of columns, most with embedded values
- Complex nested schema does not fit into traditional RDBMS systems
- Systems need to fit well into the Python ecosystem
- Leverage latest data management and processing technologies

Our vision seems to fit well in solving data management challenges of HEP data !

Solution

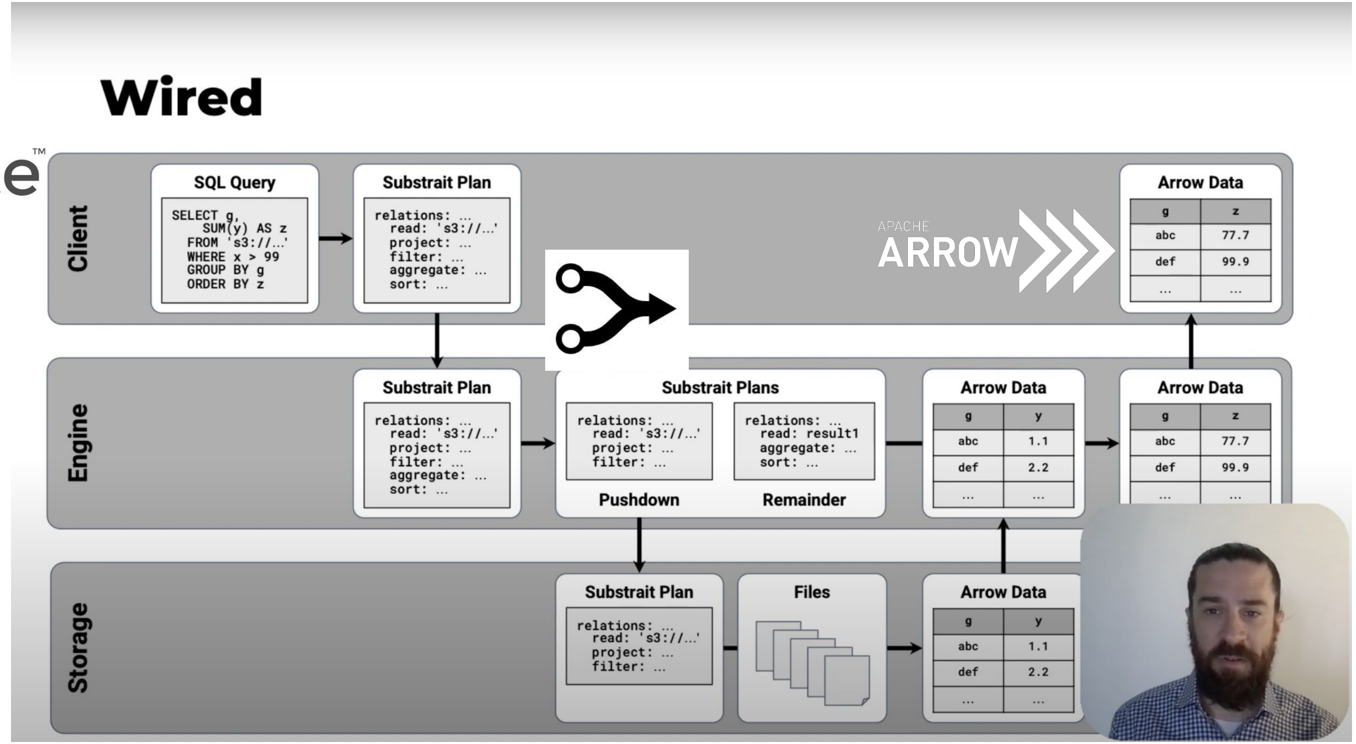
- Zero-Copy In-Memory Data Format
  - Eliminate serialization costs while moving data between different processes
- Distributed Active Storage Layer
  - Reduce data movement by filtering data within the storage layer
- Distributed Compute Layer
  - Distributed compute operations such as Joins, GroupBy using MapReduce/BSP over MPI/UCX/RDMA
- Transactions over Datasets: *Lakehousing*
  - Features of warehousing such as **transactions**, **views**, **time travel**, **schema evolution** but directly over data lakes
- Expressive Query interface and Query compiler
  - Different query interfaces generating standard query plans acceptable by popular data processing systems

Implementation Plan



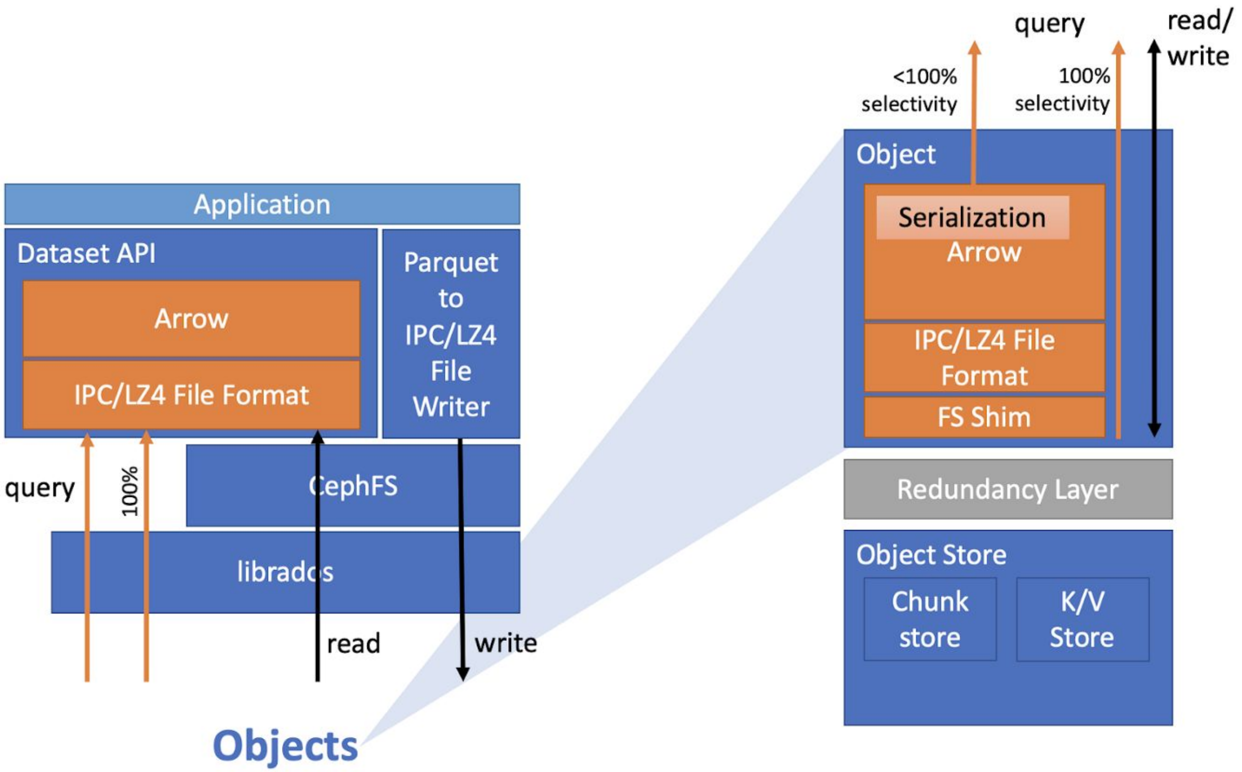
Ian Cook, “Arrow and Substrait: Better Together,” The Data Thread, 6/23/22, https://youtu.be/5JjaB7p3Sjk

Implementation Plan



Ian Cook, "Arrow and Substrait: Better Together," The Data Thread, 6/23/22, https://youtu.be/5JjaB7p3Sjk

Skyhook Architecture



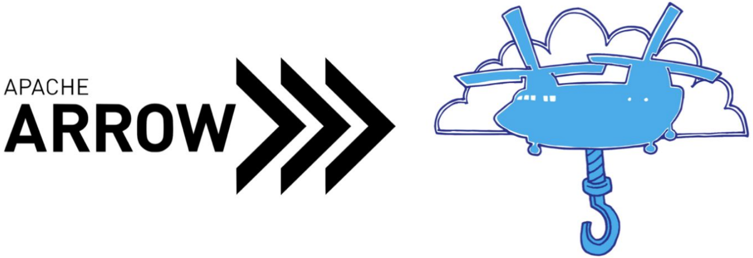
<u>Current Status</u>: Skyhook upstreamed in Apache Arrow !

Skyhook: Bringing Computation to Storage with Apache Arrow

PUBLISHED 31 Jan 2022
BY Jayjeet Chakraborty, Carlos Maltzahn, David Li, Tom Drabas

CPUs, memory, storage, and network bandwidth get better every year, but increasingly, they're improving in different dimensions. Processors are faster, but their memory bandwidth hasn't kept up; meanwhile, cloud computing has led to storage being separated from applications across a network link. This divergent evolution means we need to rethink where and when we perform computation to best make use of the resources available to us.

SkyhookDM is now a part of Apache Arrow!



We are happy to announce that Skyhook Data Management is now officially a part of the Apache Arrow project mainline and is planned to be included in release 7.0.0. SkyhookDM is a plugin for offloading computations involving data processing operations into the storage layer of distributed and programmable object storage systems. It is be aintained by researchers at

Acknowledgements

This work was supported by the National Science Foundation under Cooperative Agreement OAC-1836650 (IRIS-HEP.org), and the Center for Research in Open Source Software, UC Santa Cruz (cross.ucsc.edu)

Thank You

jayjeetc@ucsc.edu

https://iris-hep.org/projects/skyhookdm.html

Zero-Copy In-Memory Data Format

- For efficient processing of HEP data, the data needs to be stored in a zero-copy in-memory format where
  - The In-memory and on-the-wire format is equivalent
  - There is no need of serialization while moving data between different processes
  - Supports SIMD processing

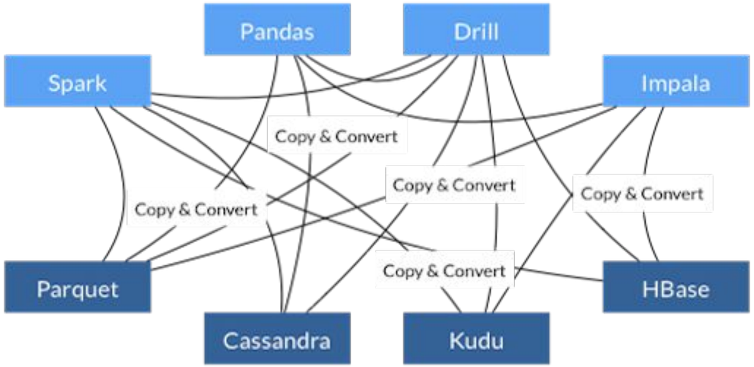
Zero-Copy In-Memory Data Format

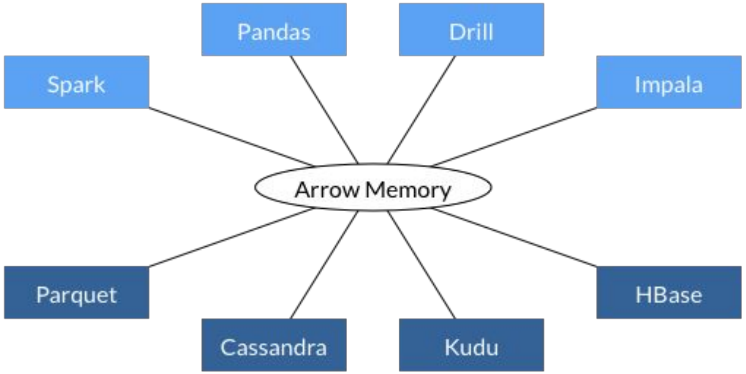
- For efficient processing of HEP data, the data needs to be stored in a zero-copy in-memory columnar format where
  - The In-memory and on-the-wire format is equivalent
  - There is no need of serialization while moving data between different processes
  - Supports SIMD processing

APACHE ARROW

Zero-Copy In-Memory Data Format

- For efficient processing of HEP data, the data needs to be stored in a zero-copy in-memory format where
  - The In-memory and on-the-wire format is equivalent
  - There is no need of serialization while moving data between different processes
  - Supports SIMD processing



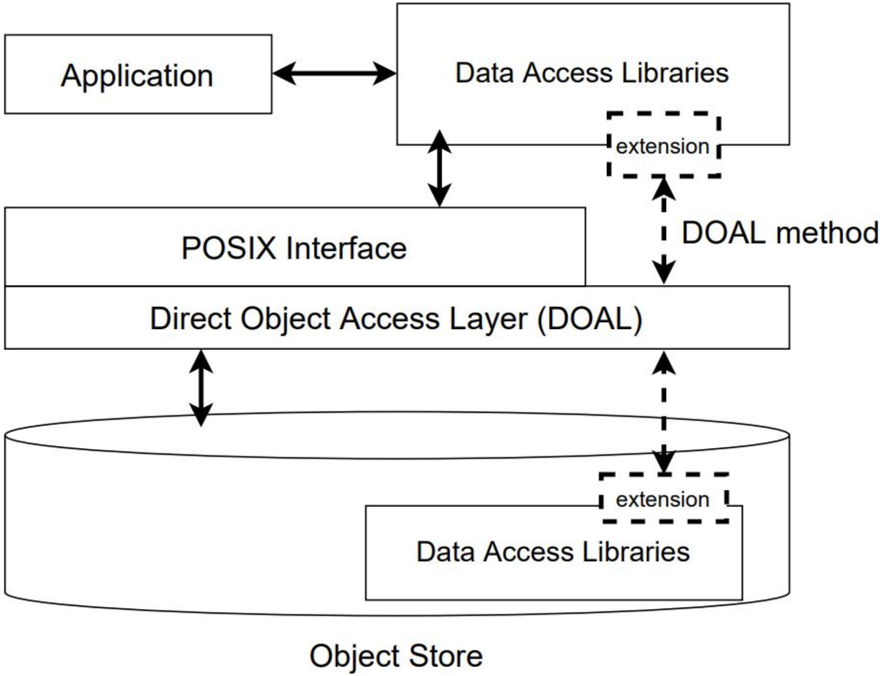


Distributed Active Storage Layer

- Offload tasks like filters, projections, aggregations to the storage layer
    - Get back scalability and performance by putting idle storage layer CPUs to use
    - Pivot client CPU resources from costly data decoding and decompression to useful operations like Joins
    - HEP datasets are huge: active storage saves network bandwidth by filtering data early in the stack.

Distributed Active Storage Layer

- Extend client and storage layers of programmable storage systems with data access libraries
- Embed a FS shim inside storage nodes to have file-like view over objects
- Make object class extensions directly available to the clients without having to change the FS



Using Skyhook from Arrow

```python
# Reading from Parquet
import pyarrow.dataset as ds
format_ = "parquet"
dataset = ds.dataset(
    "/dataset", format=format_
)
dataset.to_table()

# Reading from Parquet using Skyhook
import pyarrow.dataset as ds
format_ = ds.SkyhookFileFormat(
    "parquet", "/ceph.conf"
)
dataset = ds.dataset(
    "/dataset", format=format_
)
dataset.to_table()
```

Skyhook supports file formats that are supported by Arrow out-of-the-box

- Parquet, CSV, JSON, Feather

Distributed Compute Layer

- Joins, groupby, aggregation, shuffle, and all pandas operations over HEP data
- Support distributed versions of these compute operations using map/reduce or BSP semantics over MPI/UCX

Distributed Compute Layer

- Joins, groupby, aggregation, shuffle, and all pandas operations over HEP data
- Support distributed versions of these compute operations using map/reduce or BSP semantics over MPI/UCX

Transactions over Datasets: *Lakehousing*

- Data warehouse features over a data lake
- Features of warehousing such as **transactions**, **views**, **time travel**, **schema evolution** but directly over data lakes
    - No need of ingesting data into data warehouses
- Adds a WAL (Write Ahead Log) to the Dataset for supporting transactions: updates/appends/deletes: **Table** format

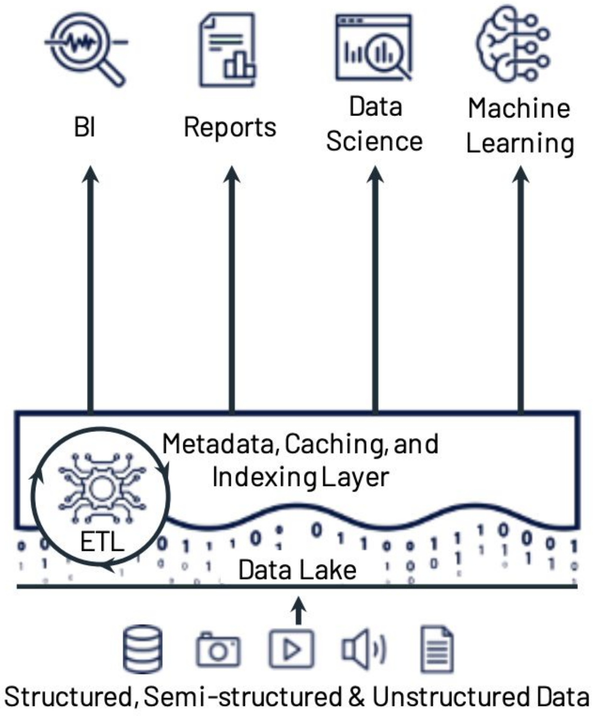
Transactions over Datasets: *Lakehousing*

- Data warehouse features over a data lake
- Features of warehousing such as **transactions**, **views**, **time travel**, **schema evolution** but directly over data lakes
    - No need of ingesting data into data warehouses
- Adds a WAL (Write Ahead Log) to the Dataset for supporting transactions, updates/appends/deletes: **Table** format

ICEBERG

DELTA LAKE

Transactions over Datasets: *Lakehousing*



Expressive Query interface and Query compiler

- Different compute/query execution engines can be used over storage systems to perform complex compute operations.
- Different engines have different query interfaces:
  - Pandas-style dataframe interface
  - SQL interface
- A single query interface that supports different kinds of queries and compute operations should compile queries into a standardized query plan which is accepted by the query executions engines

Expressive Query interface and Query compiler

- Substrait.io: Well-defined, cross-language specification for data compute operations. Used to communicate a query plan between a SQL/DataFrame interface and a query execution engine in a serialized format
- Ibis and Apache Calcite: These projects translate queries in Python and SQL to a substrait query plan.