

Lecture #23

Carnegie Mellon University
ADVANCED DATABASE SYSTEMS

Velox

Andy Pavlo // 15-721 // Spring 2023

ADMINISTRIVIA

Project #2: Monday May 1st

Project #3: Friday May 5th @ 5:30pm

Final Exam: Sunday May 7th @ 11:59pm

→ Submit on Gradescope

→ Please also fill out the form on whether you use ChatGPT.

Amazon Redshift Guest Lecture (*In-Class*)

→ Wednesday April 26th @ 2:00pm ET

OBSERVATION

Since the late 2000s, Facebook/Meta has made contributions to DBMS development.

→ They still run the largest MySQL deployment in the world.

But unlike some of the other major tech companies, they have open-sourced nearly all their projects.

DATA SYSTEMS AT META

☠ Scribe (2008)

Cassandra (2008) ➔ *DataStax, FaunaDB, Scylla*

Hive (2010)

RocksDB (2012) ➔ *Rockset, Terark, YugabyteDB, Speedb, SurrealDB, BaikalDB, CockroachDB, CozoDB, EdgelessDB, Gaia, IndraDB, ksqlDB, Kvrocks, Milvus, Pika, Stardog, SurrealDB, TiKV, TypeDB, XTDB*

Not Open-Source ➔ Scuba (2013)

PrestoDB (2013) ➔ *Ahana, Trino/Starburst*

☠ WebScaleSQL (2014)

☠ Gorilla/Beringei (2015)

☠ LogDevice (2017)

Velox (2021)

DATA SYSTEMS

☠ Scribe (2008)

Cassandra (2008) → *DataStax*

Hive (2010)

RocksDB (2012) → *Rockset, Teradata, CockroachDB, Milvus, Pika*

Not Open-Source → Scuba (2013)

PrestoDB (2013) → *Ahana,*

☠ WebScaleSQL (2014)

☠ Gorilla/Beringei (2015)

☠ LogDevice (2017)

Velox (2021)



HISTORICAL CONTEXT

Meta collects a lot of data. Over the last decade, their engineering teams have built many (redundant) internal tools to analyze this data.

This fracturing has led to wasted efforts and inconsistent features/performance.

Like Databricks, Meta recognized that it was not feasible to replace all the existing systems with a new one...

META VELOX

Extensible C++ library to support high-performance single-node query execution.

- No SQL parser!
- No meta-data catalog!
- No cost-based optimizer!

Velox takes in a physical plan (DAG of operators) as its input for execution. It then produces the output to the specified location.

VELOX OVERVIEW

Push-based Vectorized Query Processing

Precompiled Primitives + Codegen Expressions (C++)

Arrow Compatible (extended)

Adaptive Query Optimization

Sort-Merge + Hash Joins

VELOX COMPONENTS

Type System

Vector Internal Representation

Expression Engine

Function API

Operator Engine

Storage Connectors / Adapters

Resource Manager

VELOX STORAGE

Velox does not "own" data and it does not have a proprietary data format.

Instead, it exposes APIs to define connectors to retrieve data from systems and adapters to decode/encode storage formats.

→ Systems: S3, HDFS

→ Formats: Parquet, ORC/DWRF, Alpha

VECTOR INTERNAL REPRESENTATION

Velox uses in-memory vectors to move data to and from operators at runtime.

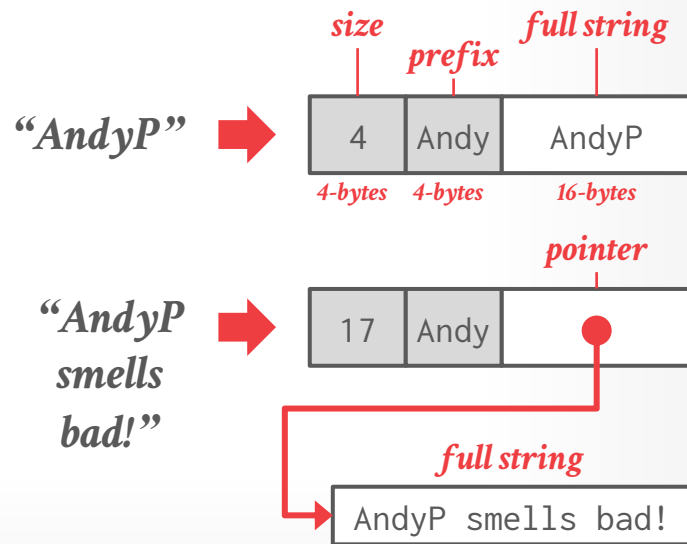
→ Extends Apache Arrow columnar layout to support more encoding/compression schemes.

Optimizations:

→ Lazy Vector Materialization

→ German-style String Storage

→ Out-of-order Writes/Population



EXPRESSION ENGINE

Velox converts expression trees into a flattened intermediate representation that they then execute during query processing.

→ Think of it like an array of function pointers to precompiled (untemplated) primitives.

Experimental branch transpiles IR into C++ code and then compiles to machine code via exec.

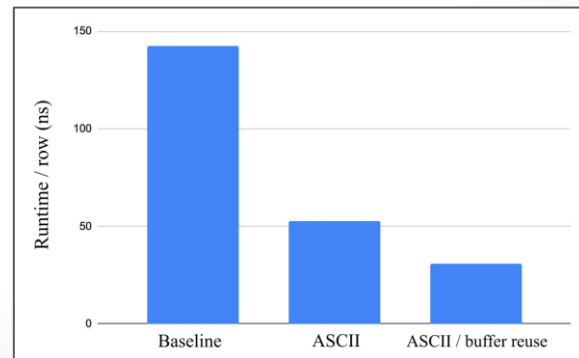
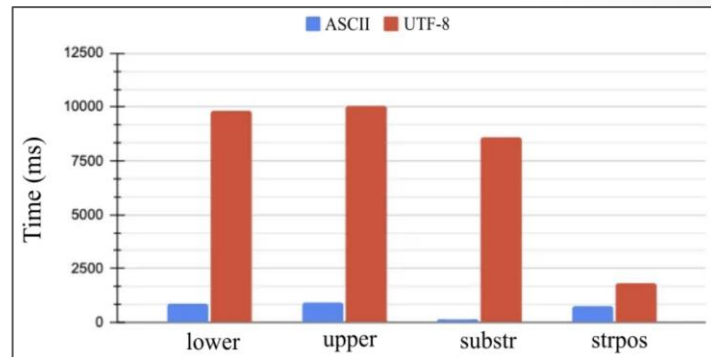
QUERY ADAPTIVITY

Predicate Reordering

Column Prefetching

Elide ASCII Encoding Checks

→ Bonus: Reuse buffers for output!



Source: [Pedro Pedreira](#)

PRESTISSIMO

Replace PrestoDB's Java-based runtime engine with Velox-based engine.

→ Think of it like Databricks replacing Spark SQL runtime with Photon via JNI.

Uses Velox API to re-implement SQL functions and operators beyond base Velox to provide compatibility with PrestoDB.

PRESTI

Replace PrestoDB's Java-based Velox-based engine.

→ Think of it like Databricks r with Photon via JNI.

Uses Velox API to re-implement operators beyond base Velox compatibility with Presto.

Question of the episode: Will Trino be making a vectorized C++ version of Trino workers?

Full question from Trino Slack

Answer: Writing a C++ worker would require each plugin to be implemented in C++ as well. However, you don't need C++ for vectorization. Java already does a technique called **auto-vectorization** which we will demonstrate later in the show! Java 17 also introduces the new **Vector API** which unlocks complex usage patterns that we can invest in moving forward. However, there's so much more to making operations faster than just bare metal speed that we are going to focus on.

To demonstrate this, I'd like to use an analogy about how I think of this. Comparing C++ and Java implementation is like comparing the two fastest men in the world. Usain Bolt holds the most world records for mens track to this date, and teammate Yohan Blake holds many of the second place titles. Most of us know Usain Bolt is the fastest of the two, and you may not have known or remembered Yohan's name before. Want to hear something crazy, Yohan has beaten Usain Bolt in a few races. The two are so close in speed, it's seconds to milliseconds difference. The main difference in this analogy is that speed is the only thing that matters in an olympic race. However, programming languages and frameworks have a lot more tradeoffs.

The point is, Java is fast and more importantly, it removes a lot of burden maintaining and scaling out the code. This is conducive to a healthy open-source project, and lowers the barrier for collaboration. Rather than go against this and take on the feat of having to rewrite an entire system in C++, why not lean into the incredible innovation recent Java features have to offer to improve performance even more.

Another important aspect is rather than chasing the fastest bare metal speed, it's also incredibly important to dedicate time into ensuring that Trino's optimizer is producing the best possible plans to avoid doing unnecessary work. To continue with the analogy, in a 100m race on a 400m track, imagine we have Usain and Yohan go head to head. We may expect that Usain will likely win, given his track record. However, if Usain is given the wrong instructions and runs in the wrong direction (300m), my bets are that Yohan will win the race.

In essence, the direction of Trino while still including bare metal performance improvements in the JVM, will instead focus on not wasting time with suboptimal query plans before or during runtime. There are so many optimizations that are constantly being added to every release that ultimately makes for a work-smarter-not-harder query engine.

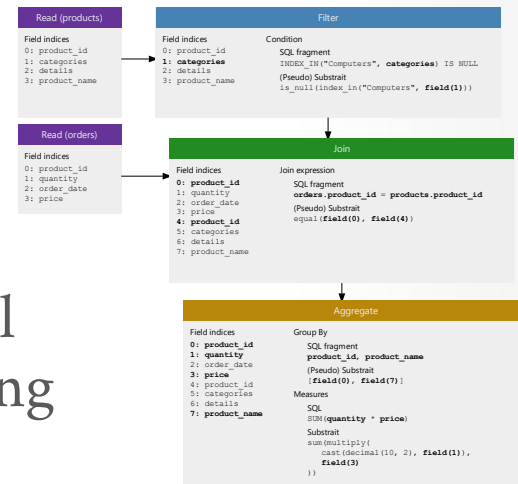
SUBSTRAIT (2021)

Open-source specification to represent relational algebra query plans.

→ Think of it like Arrow but for query plans.

The idea is that systems can share physical query plans with each other without having to convert them into a native API/DSL.

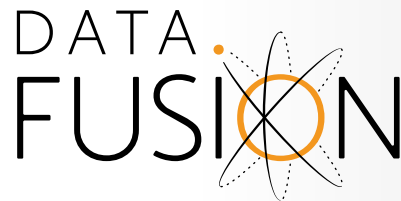
→ Federated DBMSs are hard.



DATAFUSION (2019)

Extensible vectorized execution library for Apache Arrow data.

→ Written in Rust for the kids!



Provides more front-end functionality features to build a complete DBMS than Velox

→ SQL and DataFrame APIs.

→ Query Optimizer

Examples: InfluxDB, CeresDB, CnosDB, Seafoal

POLARS (2020)

Extensible vectorized execution library for Apache Arrow data.

→ Written in Rust for the kids!



Provides more front-end functionality features to build a complete DBMS than Velox

→ SQL and DataFrame APIs.

→ Query Optimizer

Examples: ???

PARTING THOUGHTS

Instead of building an OLAP DBMS from scratch or forking an existing DBMS, starting with something like Velox that was designed to be embedded + extended seems like the better move.

That means the differentiating factors between DBMSs will be UI/UX factors and query optimization.

NEXT CLASS

Amazon Redshift with Ippokratis Pandis (PhD'07)