Sponsor Talk
CIDR’23

Pedro Pedreira - pedroerp@fb.com
Software Engineer
Velox: Meta’s Unified Execution Engine
CIDR’23

Pedro Pedreira - pedroerp@fb.com
Software Engineer
Motivation
User Workload Variety
# Engine Specialization

“One size does not fit all”

<table>
<thead>
<tr>
<th>Analytics</th>
<th>Realtime Infra</th>
<th>Graph</th>
<th>Monitoring</th>
<th>Transactional</th>
<th>ML</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Presto</td>
<td>● XStream</td>
<td>● DIGraph</td>
<td>● Scuba</td>
<td>● MySQL</td>
<td>● TorchArrow/PyTorch</td>
</tr>
<tr>
<td>● Spark</td>
<td>● Scribe</td>
<td></td>
<td>● ODS</td>
<td>● RocksDB</td>
<td>● F3</td>
</tr>
<tr>
<td>● Saber</td>
<td>● FBETL</td>
<td></td>
<td>● Logarithm</td>
<td>● XSQL</td>
<td>● Koski</td>
</tr>
<tr>
<td>● Cubrick</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Engine Specialization

The flipside

- Very limited reusability.
- Duplicates efforts and forces engineers to reinvent the wheel.
- Hard to maintain and enhance.
  - Where do we optimize?
- Exposes inconsistencies to end-users.
- Hurts our capacity to move fast and innovate.
## Through the Looking Glass

Different, but not really...

<table>
<thead>
<tr>
<th>Language Frontend</th>
<th>Presto SQL</th>
<th>Spark SQL</th>
<th>XStream SQL</th>
<th>Scuba SQL</th>
<th>Cubrick SQL</th>
<th>Koski DataFrame</th>
<th>F3 DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>Presto IR</td>
<td>Spark IR</td>
<td>UPM IR</td>
<td>Scuba IR</td>
<td>Cubrick IR</td>
<td>Koski IR</td>
<td>F3 IR</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Presto Optimizer</td>
<td>Catalyst</td>
<td>XStream Optimizer</td>
<td>Scuba Optimizer</td>
<td>Cubrick Optimizer</td>
<td>Koski Optimizer</td>
<td>F3 Optimizer</td>
</tr>
<tr>
<td>Execution Runtime</td>
<td>Presto Runtime</td>
<td>Spark Runtime</td>
<td>XStream Runtime</td>
<td>Scuba Runtime</td>
<td>Cubrick Runtime</td>
<td>DPP</td>
<td>-</td>
</tr>
<tr>
<td>Execution Engine</td>
<td>Presto</td>
<td>Spark</td>
<td>XStream Codegen</td>
<td>Scuba</td>
<td>Cubrick</td>
<td>Koski</td>
<td>F3 DAG</td>
</tr>
</tbody>
</table>
## Through the Looking Glass

Different, but not really...

<table>
<thead>
<tr>
<th></th>
<th>Presto</th>
<th>Spark</th>
<th>XStream</th>
<th>Scuba</th>
<th>Cubrick</th>
<th>Koski</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language Frontend</strong></td>
<td>Presto SQL</td>
<td>HQL / DataSet API</td>
<td>UPM</td>
<td>Scuba SQL</td>
<td>Cubrick SQL</td>
<td>Koski DataFrame</td>
<td>F3 DSL</td>
</tr>
<tr>
<td><strong>IR</strong></td>
<td>Presto IR</td>
<td>Spark IR</td>
<td>UPM IR</td>
<td>Scuba IR</td>
<td>Cubrick IR</td>
<td>Koski IR</td>
<td>F3 IR</td>
</tr>
<tr>
<td><strong>Optimizer</strong></td>
<td>Presto Optimizer</td>
<td>Catalyst</td>
<td>XStream Optimizer</td>
<td>Scuba Optimizer</td>
<td>Cubrick Optimizer</td>
<td>Koski Optimizer</td>
<td>F3 Optimizer</td>
</tr>
<tr>
<td><strong>Execution Runtime</strong></td>
<td>Presto Runtime</td>
<td>Spark Runtime</td>
<td>XStream Runtime</td>
<td>Scuba Runtime</td>
<td>Cubrick Runtime</td>
<td>DPP</td>
<td>-</td>
</tr>
<tr>
<td><strong>Execution Engine</strong></td>
<td>Presto</td>
<td>Spark</td>
<td>XStream Codegen</td>
<td>Scuba</td>
<td>Cubrick</td>
<td>Koski</td>
<td>F3 DAG</td>
</tr>
</tbody>
</table>

All very similar!
Through the Looking Glass

Different, but not really…

<table>
<thead>
<tr>
<th></th>
<th>Presto</th>
<th>Spark</th>
<th>XStream</th>
<th>Scuba</th>
<th>Cubrick</th>
<th>Koski</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language Frontend</strong></td>
<td>Presto SQL</td>
<td>HQL / DataSet API</td>
<td>UPM</td>
<td>Scuba SQL</td>
<td>Cubrick SQL</td>
<td>Koski DataFrame</td>
<td>F3 DSL</td>
</tr>
<tr>
<td><strong>IR</strong></td>
<td>Presto IR</td>
<td>Spark IR</td>
<td>UPM IR</td>
<td>Scuba IR</td>
<td>Cubrick IR</td>
<td>Koski IR</td>
<td>F3 IR</td>
</tr>
<tr>
<td><strong>Optimizer</strong></td>
<td>Presto Optimizer</td>
<td>Catalyst</td>
<td>XStream Optimizer</td>
<td>Scuba Optimizer</td>
<td>Cubrick Optimizer</td>
<td>Koski Optimizer</td>
<td>F3 Optimizer</td>
</tr>
<tr>
<td><strong>Execution Runtime</strong></td>
<td>Presto Runtime</td>
<td>Spark Runtime</td>
<td>XStream Runtime</td>
<td>Scuba Runtime</td>
<td>Cubrick Runtime</td>
<td>DPP</td>
<td>-</td>
</tr>
<tr>
<td><strong>Execution Engine</strong></td>
<td>Presto</td>
<td>Spark</td>
<td>XStream</td>
<td>Scuba</td>
<td>Cubrick</td>
<td>Koski</td>
<td>F3</td>
</tr>
</tbody>
</table>

All very similar!
Velox Mission

Converge, Accelerate, and Unify execution engines across Meta and beyond
Velox Library Overview

- A generic **C++ database acceleration library**.
  - **Generic APIs**: from batch to interactive, to stream processing, to AI/ML workloads.
    - Key Concepts: **Modularity** and **Extensibility**.
  - **C++**: **native code** for maximum efficiency
    - **10x** cpp vs. java win (TPC-H Q1 and Q6 microbenchmarks).
  - State-of-art
    - **Centralize all optimizations** implemented in current engines.
Velox Library Overview (2)

- Database acceleration library vs. DBMS.
- Velox takes a fully optimized **physical plan** as input.
  - No frontend (SQL parser or dataframe layer)
  - No global optimizer.
- Though there’s tons of adaptivity.
- Velox sits on the data-path
  - Everything that runs on a single server.
- No control plane.
Velox - Value Proposition

01 Efficiency and Latency

02 Consistency and Consolidation

03 Reusability
Use Cases
Velox - Use Cases

● Analytics:
  ○ Presto/Prestissimo - interactive
  ○ Spark//Gluten - batch
  ○ Saber - external analytics

● Realtime Infrastructure:
  ○ XStream - stream processing
  ○ FBETL/Morse - data warehouse and database ingestion
  ○ Scribe - log messaging system

● Transactional:
  ○ XSQL - distributed transaction processing

● Machine Learning:
  ○ TorchArrow/PyTorch - data preprocessing
  ○ F3 - feature engineering
  ○ XLDB/Koski - training
Velox - Open Source

- Publicly announced in Oct 22!
  - [https://engineering.fb.com/2022/08/31/open-source/velox/](https://engineering.fb.com/2022/08/31/open-source/velox/)
- Available in github:
  - [https://github.com/facebookincubator/velox](https://github.com/facebookincubator/velox)
- VLDB’22:
  - “Velox: Meta’s Unified Execution Engine”
- Fast growing open source community
  - +180 developers
  - Meta, Ahana, Intel, Voltron Data, …
Library Outline
Velox Library - Components Outline

- **Types:**
  - Scalar and nested data types, including structs, maps, arrays, tensors, and more.

- **Vectors:**
  - An “Arrow-compatible” columnar memory layout module.

- **Expression Eval:**
  - Fully vectorized expression evaluation engine built based on Vector-encoded data.

- **Functions:**
  - APIs for custom scalar (row-by-row and batch-by-batch) and aggregate functions.

- **Operators:**
  - Common data processing SQL operators (OrderBy, GroupBy, HashJoin, etc).

- **I/O:**
  - Pluggable file format encode/decoder, storage adapter, and network serializers.

- **Resource Management:**
  - Memory pools, arenas, thread/tasks, spilling, SSD and memory caching.
Ongoing Work
Ongoing Work - Where we need help

- Continue blurring the boundaries between Analytics and ML.
- Software and hardware co-evolution.
- Further componentization of the stack.
- More collaboration with academia!
Ongoing Work - Where we need help

- Continue blurring the boundaries between Analytics and ML.
- Software and hardware co-evolution.
- Further componentization of the stack.
- More collaboration with academia!

Tuesday 3:20pm @CIDR

---

**Shared Foundations: Modernizing Meta’s Data Lakehouse**

Biswapesh Chattopadhyay, Pedro Pedreira, Sameer Agarwal, Yutian "James" Sun,
Suketu Vakharia, Peng Li, Weiran Liu, Sundaram Narayanan
{biswapesh,pedroerp,sag,jamessun,suketuv,plfb,weiranliu,sunnar}@fb.com
Meta Platforms Inc.
Menlo Park, CA, USA

**ABSTRACT**

Data processing systems have evolved significantly over the last decade, driven by large trends in hardware and software, the exponential growth of data, and new and changing use cases. At Meta (and elsewhere), the various data systems composing the data lakehouse had historically evolved organically and independently, leading to data stack fragmentation, and resulting in work duplication, subpar system performance, and inconsistent user experience. This work describes the efforts toward the modernization of machine learning workloads has developed a new set of trends in terms of data volume, complexity, and unusual access patterns [26].

Meanwhile, Meta’s data stack had only evolved incrementally over the last decade. This has resulted in a fragmented stack which was difficult to maintain and evolve, composed of almost a dozen SQL dialects, multiple engines targeting similar workloads (each with their own quirks), and numerous copies of the same data in different locations and formats. The lack of standardization and
Thank you!