Shared Foundations: Modernizing Meta’s Data Lakehouse

Biswapesh Chattopadhyay, Pedro Pedreira, Sameer Agarwal, Yutian "James" Sun, Suketu Vakharia, Peng Li, Weiran Liu, Sundaram Narayanan

CIDR’23

Pedro Pedreira - pedroerp@fb.com
Software Engineer

1/10/23
Agenda

1. Recent Trends
2. Background
3. Shared Foundations
4. Consolidation Efforts
Recent Trends
Usage Trends

- Data Explosion
- Machine Learning
- Freshness and Latency
- External Analytics
- Complex Data Models
- Richer Query Methods

Environmental Trends

- Disaggregation
- Horizontal Scaling
- Elastic Compute
- Power Efficiency
- Global Optimization
- Engineering Efficiency
Solution: Open Data Lakehouse Analytics

- Direct data access:
  - Disaggregated storage
  - Open file formats
  - Open metadata APIs

- Diverse applications:
  - Batch
  - Interactive
  - Streaming
  - Machine Learning
Background
Open Data Lakehouse @Meta

- Services (WWW)
- Logs
- Stream Processing
- Databases
- Ingest
- Warehouse Storage & Metadata
- ML Training
- Interactive Queries
- Batch Processing
- Batch Processing
# The Problem: Fragmentation

<table>
<thead>
<tr>
<th>Layer</th>
<th>Scope</th>
<th>Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>SQL dialects, functions, entity &amp; type metadata</td>
<td>SQL dialect fragmentation, lack of expressibility</td>
</tr>
<tr>
<td>Distribution/Runtime</td>
<td>Distributed execution, shuffle, resource management</td>
<td>Scalability, Efficiency, Fragmentation</td>
</tr>
<tr>
<td>Execution</td>
<td>Evaluation at node, caching</td>
<td>Latency, efficiency, Java / C++, dialect fragmentation</td>
</tr>
<tr>
<td>Data Access</td>
<td>Formats, storage, disaggregation</td>
<td>Library fragmentation, not data driven, poor encodings</td>
</tr>
</tbody>
</table>
Impact and Solution

● How does this impact us?
  ○ Hard to maintain and enhance:
    ■ Poor innovation velocity
  ○ Inconsistent user APIs:
    ■ Poor user experience

● What can we do about it?
  ○ Building Shared Foundations!
Shared Foundations
The Solution: Shared Foundations

- **Principles:**
  - Fewer systems
  - Shared components
  - Consistent APIs

- **Goals:**
  - Engineering efficiency
  - Faster innovation
  - Better user experience
Consolidation Efforts
Language Consolidation

- Half a dozen SQL dialects being actively used at Meta:
  - Presto SQL, HiveQL (in Spark), PQL (Puma), Scuba SQL, Cubrick SQL and MySQL.
- Ideal dialect:
  - Standard-compliant
  - Rich feature set
  - Wide adoption
- Presto SQL -> CoreSQL
- Two component are needed:
  - C++ parser/analyzer library
  - Execution library
Execution Consolidation

- Unified execution engine: **Velox**
- Reusable across engines (Analytics, Stream Processing, ML, and more)
- Provides fully compatible implementation of **CoreSQL**.
Engine Consolidation - Interactive Analytics

● Many interactive analytics engines:
  ○ Presto, Raptor, Cubrick, Scuba

● Ideal system:
  ○ Full and rich SQL support -> CoreSQL
  ○ Operate directly on lakehouse
  ○ Low query latency

● Presto -> RaptorX:
  ○ Hierarchical caching
  ○ Affinity

● Data freshness:
  ○ Near real time support
Engine Consolidation - Interactive Analytics (2)

- RaptorX -> **Prestissimo**
  - Presto running on Velox

- **Result:**
  - Single engine
  - Language consolidation (CoreSQL)
  - Low latency (local caching)
  - Data freshness (NRT)
  - Efficient execution (Velox).
Engine Consolidation - Batch Analytics

- Batch engines:
  - Presto, Spark

- Ideal system:
  - Full and rich SQL support -> CoreSQL
  - Large scale scalability

- **Presto-on-Spark**

- **Result:**
  - Language consolidation (CoreSQL)
  - Scalability (Spark runtime)
  - Efficient execution (Velox)
Engine Consolidation - Stream Processing

- Programming language diversity (C++, Java, Php)
- Abstraction level (procedural, declarative - SQL-like)
- Next generation -> XStream:
  - CoreSQL (added streaming extensions)
  - Velox for execution

**Result:**
- Language consolidation (CoreSQL)
- Efficient execution (Velox)
- Single engine.
Engine Consolidation - Machine Learning

- Custom eval engine -> move to Velox.
- File format inefficiencies -> **Alpha**
  - Alpha available in other engines via Velox.
- **Result:**
  - Language consolidation (TorchArrow, CoreSQL functions)
  - Efficient and unified execution (Velox)
  - Efficient decoding (Alpha).
Conclusion
Conclusion

● **Generational leap** in the data infrastructure landscape:
  ○ More modern, composable, and consistent stack.
  ○ Fewer components, richer features, and better performance.

● In the process we have:
  ○ Deprecated several large systems
  ○ Removed hundreds of thousands of lines of code
  ○ Open sourced several components
    ■ Velox, DWIO, Prest on Spark, RaptorX and TorchArrow
  ○ Improved engineering velocity and decreased operational burden.
What’s Next?

- This journey is 1% finished!
  - Projects in different stages of completion.
- Unified SQL is great (CoreSQL); how about beyond-SQL?
- Consistent UDFs across engines:
  - Universal UDFs
What’s Next?

● This journey is 1% finished!
  ○ Projects in different stages of completion.
● Unified SQL is great (CoreSQL); how about beyond-SQL?
● Consistent UDFs across engines:
  ○ Universal UDFs
● Composability is the future of data management:
  ○ Language, Execution, Data Access
  ○ ..., Optimizers?
  ○ Hardware acceleration
Thank you!