Enhancing Trino's Query Performance and Data Management with Hudi: Innovations and Future

June 13, 2024

Ethan Guo

ethan@onehouse.ai
Speaker Bio

Ethan Guo

➔ Data Infrastructure Engineer @ Onehouse.ai
➔ Apache Hudi PMC Member
➔ Senior Engineer @ Uber

Data (Near Real-Time Analytics with Hudi Incremental Processing),
Networking (App Network Performance with QUIC)

Linkedin: in/yihua-ethan-guo/
Trino + Hudi: Fast Analytics + Upserts

**Trino**
- Fast SQL query with massively parallel processing

**Hudi**
- Fast upserts with incremental processing in Lakehouse

Fresher data, reports, and analytics

Robinhood’s architecture and use cases for Trino and Hudi: [https://trino.io/episodes/41.html](https://trino.io/episodes/41.html)
Agenda

- Apache Hudi: The Open Data Lakehouse Platform
- Improving Query Performance with Multi-Modal Index in Hudi
- Enhancing Trino Hudi Connector
- Future of Trino with Hudi
Apache Hudi:
The Open Data Lakehouse Platform
Origins@Uber 2016

Context

- Uber in hypergrowth
- Moving from warehouse to lake
- HDFS/Cloud storage is immutable

Problems

- Extremely poor ingest performance
- Wasteful reading/writing
- Zero concurrency control or ACID
Core Primitives in Hudi

- **Upserts**: Absorb changes to records and process faster
- **Incremental Reads**: Obtain records that changed
- **Snapshot isolation**: Read latest committed state consistently

**Missing pieces**: Upserts, Deletes & Incrementals

- **Consume only changes?**
- **Process changes faster?**
- **Query snapshots consistently?**
- **Stream out changes?**

- **Trips Database**
- Real-time Change logs in Kafka
- Parquet Files
  - Raw
  - Derived
- File based Data lake

- 120 TB Bulk ingested every 8hrs; Actual change < 500GB
- E2E Data Freshness ~24 hours

- **Streams Out changes?**
- **Query snapshots consistently?**
- **Upserts**: Absorb changes to records and process faster
- **Incremental Reads**: Obtain records that changed
- **Snapshot isolation**: Read latest committed state consistently
Proven @ Massive Scale

ByteDance
- 100GB/s Throughput
- > 1Exabyte Even just 1 Table
- 70% CPU Savings (write+read)

Uber
- 4000+ Tables
- 250+PB Raw + Derived
- 800B Records/Day

Walmart
- 300GB/d Throughput
- 25+TB Datasets
- Hourly Analytics Latency

GE Aviation
- 10,000+ Tables
- 150+ Source systems
- CDC, ETL Use cases

Sources:
- https://www.youtube.com/watch?v=ZamXiT9aqs8
- https://www.uber.com/blog/apache-hudi-graduation/
Improving Query Performance with Multi-Modal Index in Hudi
Improving Query Performance

Key: Reading fewer bytes from Input Tables

Indexes
- Helpful for selective queries i.e. needles in haystacks
- B-trees, bloom-filters, bit-maps..

Caching
- Eliminate access to storage in the common case
- Read-through, write-through, columnar vs row based

Storage Layout
- Control how data is physically organized in storage
- Bucketing, Clustering
Indexes: Locating Records Efficiently

- Widely employed in DB systems
  - Locate information quickly
  - Reduce I/O cost
  - Improve Query efficiency

- Indexing provides fast upserts
  - Locate records for incoming writes
  - Bloom filter based, Simple, Hbase, etc.

[Diagram showing comparison between with and without index]

Multi-Modal Index with Metadata Table

- Partitioned for extensibility
  - Files
  - Column stats
  - Bloom filter
  - Record index
  - Functional index
- Support CREATE/DROP index
- Support async indexing

New functional index in 1.0.0-beta1
Record-Level Index (RLI) - New in Hudi 0.14

• Challenges
  ○ Reading data and metadata per file is expensive
  ○ HBase index requires cluster maintenance which is operationally difficult

• Design
  ○ Key-to-location mapping in table-level metadata
    ■ A new partition, “record_index”, in the metadata table
    ■ Stored in a few file groups instead of all data files
  ○ Fast index update and lookup
    ■ MDT, an internal Hudi MOR table, enables uniformed fast updates
    ■ HFile format enables fast point lookup
Record-Level Index on Storage

- **File Group ID** by the hash
- **Record Keys**

**“record_index” partition**
- File Group 0
  - FG 1
  - ...
  - FG N-1

**File Group 0**
- File Slice t0
  - HFile
  - Log File 1
  - Log File 2
  - ...
- FS t1
  - HFile

**HFile**
- record_key 0 -> partition 1, file 1
- record_key 1 -> partition 1, file 1
- record_key 2 -> partition 2, file 3
- record_key 3 -> partition 1, file 2
  - .
  - .

**Log File 1**
- Header
- HFile Data Block 0
  - record_key 6 -> partition 1, file 5
  - record_key 7 -> partition 1, file 1
  - ...
- HFile Data Block 1
  - Footer

Compaction
Performance Benefit from RLI

- Improves index lookup and write latency
  - 1TB dataset, 200MB batch, random updates, Spark datasource
  - 17x speedup on index lookup, 2x on write

- Reduces SQL latency with point lookups
  - TPC-DS 10TB datasets, store_sales table, Spark
  - 2-3x improvement compared to no RLI

RLI blog: Hudi's blazing fast indexing for large-scale datasets
Enhancing Trino Hudi Connector
Hudi Support in Trino

- Hive connector
  - Hudi integration through InputFormat implementation
  - COW, MOR read-optimized, snapshot, and bootstrap queries (deprecated in v411, redirects to Hudi connector)

- Hudi connector
  - COW, MOR read-optimized queries only since v398; no support of metadata-based (MDT) file listing since v419
  - Due to removal of Hudi dependencies as part of Trino dehadooping
  - **RO, snapshot, bootstrap query support with MDT in upcoming Trino releases**
Hudi Storage Abstraction - New in Hudi 0.15

- HoodieStorage abstraction
  - Hadoop-independent file system and storage APIs
  - Extendable with Hadoop FileSystem and TrinoFileSystem

- HoodieIOFactory abstraction
  - Creates readers and writers for I/O (e.g., HFile)

- Hadoop-independent hudi-common module for reader integration
  - Plugs in storage and factory implementations
New HFile Reader - New in Hudi 0.15

- **HFile Format Spec**
  - Defines the HFile Format required by Hudi to enable fast point lookups in MDT
  - Custom HFile implementation (e.g., in C++ or Rust) possible by following the Spec

- **New HFile Reader implementation in Java**
  - Independent of HBase or Hadoop dependencies
  - Backwards compatible with existing Hudi releases and storage format

HFile Format Spec in Hudi:
https://github.com/apache/hudi/blob/master/hudi-io/hfile_format.md
Trino Hudi Connector Integration

- Re-introduce hudi-common dependency
  - Makes Hudi support maintainable
  - Evolves easily with future storage format changes
  - Hadoop-independent with TrinoFileSystem, unlocks optimization like caching

- Support MDT-based file listing
  - Uses new HFile Reader to support MDT read and lookup
  - 38% query latency reduction* on Trino Hudi connector in TPC-DS 1TB benchmark

- Support MOR snapshot query
  - HudiDirectoryLister determines the file listing
  - New HudiSnapshotDirectoryLister implementation for snapshot queries

* based on Trino Hudi Connector feature branch; we'll upstream the changes.
Future of Trino with Hudi
Hudi 1.x - Database for the Lakehouse

“Reimagination of Hudi, as the transactional database for the lake, with polyglot persistence”

Main components of a DBMS. 
Courtesy: The seminal database paper: Architecture of a Database System

Reference diagram highlighting existing (green) and new (yellow) Hudi components, along with external components (blue). Checkout RFC-69
New Indexes in Hudi 1.x

- **Functional index (RFC-63, in 1.0.0-beta1)**
  - Relational databases allow index on functions or expressions
  - Accelerate queries based on results of computations
  - Absorb partitioning into indexes
  - No more hide-and-evolving partitions!

- **Secondary index (RFC-77, in 1.0.0-beta2)**
  - Index for non-key fields
  - Improves query performance with predicates on the fields with secondary index built

CREATE INDEX datestr ON hudi_table USING column_stats(ts) OPTIONS(func='from_unixtime', format='yyyy-MM-dd');

<table>
<thead>
<tr>
<th>Physical partition path</th>
<th>File Name</th>
<th>Min of datestr</th>
<th>Max of datestr</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>org_id=1/datestr=2022-10-01/</td>
<td>base_file_1.parquet</td>
<td>2022-10-01</td>
<td>2022-10-01</td>
<td>Old partitioning scheme</td>
</tr>
<tr>
<td>org_id=1/datestr=2022-10-02/</td>
<td>base_file_2.parquet</td>
<td>2022-10-02</td>
<td>2022-10-02</td>
<td></td>
</tr>
<tr>
<td>org_id=2/datestr=2022-10-01/</td>
<td>base_file_3.parquet</td>
<td>2022-10-01</td>
<td>2022-10-01</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>org_id=1/</td>
<td>base_file_10.parquet</td>
<td>2022-10-18</td>
<td>2022-10-11</td>
<td>New partitioning scheme</td>
</tr>
<tr>
<td>org_id=2/</td>
<td>base_file_11.parquet</td>
<td>2022-10-18</td>
<td>2022-10-15</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

CREATE INDEX idx_city ON hudi_table USING secondary_index(city);
Roadmap

Trino Hudi Connector

2024 Q2
- Re-introduce Hudi dependency
  - Snapshot, bootstrap query, MDT support

2024 Q3
- Alluxio-powered caching
  - RLI and other index support
  - Integration with Hudi 1.0

2024 Q4
- DML/DDL support under discussion
  - (with new abstractions)

Hudi 1.x

- 1.0.0-beta1
  - LSM tree timeline
  - NBCC, functional index
  - New file group reader
- 1.0.0-beta2
  - MDT for streaming
  - Secondary index
  - File group reader impr
- 1.0.0 (GA)
  - New format finalized
  - Automated upgrade from 0.x
- 1.1
  - New indexes
- 1.2
  - Support for unstructured data, vectors, vector index

Hudi 0.x

- 0.14.1
  - Record-level index enhancement
- 0.15.0
  - Hudi storage abstraction
  - New HFile reader
  - Spark 3.5, Flink 1.18 support
- 0.16.0
  - Bridge release
  - Can read both 0.x and 1.0 tables
Come Build With The Community!

Docs:  [https://hudi.apache.org](https://hudi.apache.org)

Blogs: [https://hudi.apache.org/blog](https://hudi.apache.org/blog)

Slack: [Apache Hudi Slack Group](https://hudi.apache.org)

Twitter: [https://twitter.com/apachehudi](https://twitter.com/apachehudi)

Github: [https://github.com/apache/hudi/](https://github.com/apache/hudi/)  Give us a star ⭐!

Mailing list(s):

- dev-subscribe@hudi.apache.org (send an empty email to subscribe)
The Onehouse Universal Data Lakehouse
Delivered as a Fully-Managed Cloud Service

Swing by Onehouse booth at Trino Fest 2024
Thanks!

Questions?