Analyzing and Comparing Lakehouse Storage Systems

Paras Jain*, **Peter Kraft***, **Conor Power***,
Tathagata Das, Ion Stoica, Matei Zaharia
* Denotes equal contribution

What Is A Lakehouse?

Lakehouses are **data management systems** based on **open formats** running over **low-cost cloud storage** providing **rich management functionality** such as transactions, data versioning, and indexing while being accessible to **multiple compute engines**.

Apache hudi

ICEBERG

DELTA LAKE

Lakehouses Build on Low-Cost Data Lake Storage

Data Lake

Amazon S3

Microsoft Azure
Blob Storage

Google Cloud Storage

Lakehouse Data is Stored in Open File Formats

Data Files in Open Formats

Parquet

Apache ORC

AVRO

Data Lake

Amazon S3

Microsoft Azure Blob Storage

Google Cloud Storage

Lakehouses Manage Data Stored in Data Lakes

Lakehouse Data Management Layer
(Transactions, Metadata, Indexing...)

Apache hudi   ICEBERG   DELTA LAKE

Data Files in Open Formats

Parquet   Apache orc   AVRO

Data Lake

Amazon S3   Microsoft Azure Blob Storage   Google Cloud Storage

Lakehouse Data is Accessible to Compute Engines

SQL Engine — Apache Spark, presto

ML Engine — TensorFlow

BI, Data Science, etc…

Lakehouse Data Management Layer (Transactions, Metadata, Indexing…) — Apache Hudi, Iceberg, Delta Lake

Data Files in Open Formats — Parquet, Apache ORC, Avro

Data Lake — Amazon S3, Microsoft Azure Blob Storage, Google Cloud Storage

Why Lakehouses?

Compared to Data Lakes:

- Lakehouses provide rich data management functionality such as transactions and metadata management.

Compared to Traditional Data Warehouses:

- Lakehouses make data directly accessible to any engine, for example BI, ML, or DS tools.

Lakehouses are Being Widely Adopted

- Many large tech companies (Meta, Uber, Netflix) host their entire analytics stack on lakehouses.
- Lakehouses are increasingly offered by cloud data services (Redshift, EMR, Dataproc, Synapse...)
- >70% of bytes written by Databricks customers are to Delta Lake.

Lakehouses Face Important Design Questions

- How to coordinate transactions over low-cost cloud storage?
- Where to store metadata and how to query it in low-cost cloud storage?
- How to efficiently handle updates without sacrificing read performance?

Lakehouses Face Important Design Questions

- How to coordinate transactions over low-cost cloud storage?
- Where to store metadata and how to query it in low-cost cloud storage?
- How to efficiently handle updates without sacrificing read performance?

Talk focuses on these

We Designed LHBench

- New lakehouse benchmark built on TPC-DS.
- Ran on AWS EMR 6.9.0 with Spark 3.3.
- Try it out on GitHub!

https://github.com/lhbench/lhbench

We Analyze Three Open-Source Lakehouses

Apache Hudi

Started by Uber

Apache Iceberg

Started by Netflix

Delta Lake

Started by Databricks

LHBench Analyzes Important Aspects of Lakehouse Functionality

- Metadata Management
- Update Performance
- End-to-end Performance

LHBench Analyzes Important Aspects of Lakehouse Functionality

- **Metadata Management**
- Update Performance
- End-to-end Performance

Metadata is Critical for Query Planning

- To plan queries over data in lakehouses, distributed processing engines (Spark, Presto) need fast access to table metadata.
- Example metadata: Names and sizes of all files in table, information on column contents in each file.
- Native data lake metadata management is slow (e.g., S3 LIST is 1K keys/req).

Current Lakehouses Use Two Metadata Formats

Tabular Format

- Used by Delta, Hudi.
- Each table’s metadata stored in a separate Parquet/Avro table.
- Query planning is distributed.

Current Lakehouses Use Two Metadata Formats

Tabular Format

- Used by Delta, Hudi.
- Each table’s metadata stored in a separate Parquet/Avro table.
- Query planning is distributed.

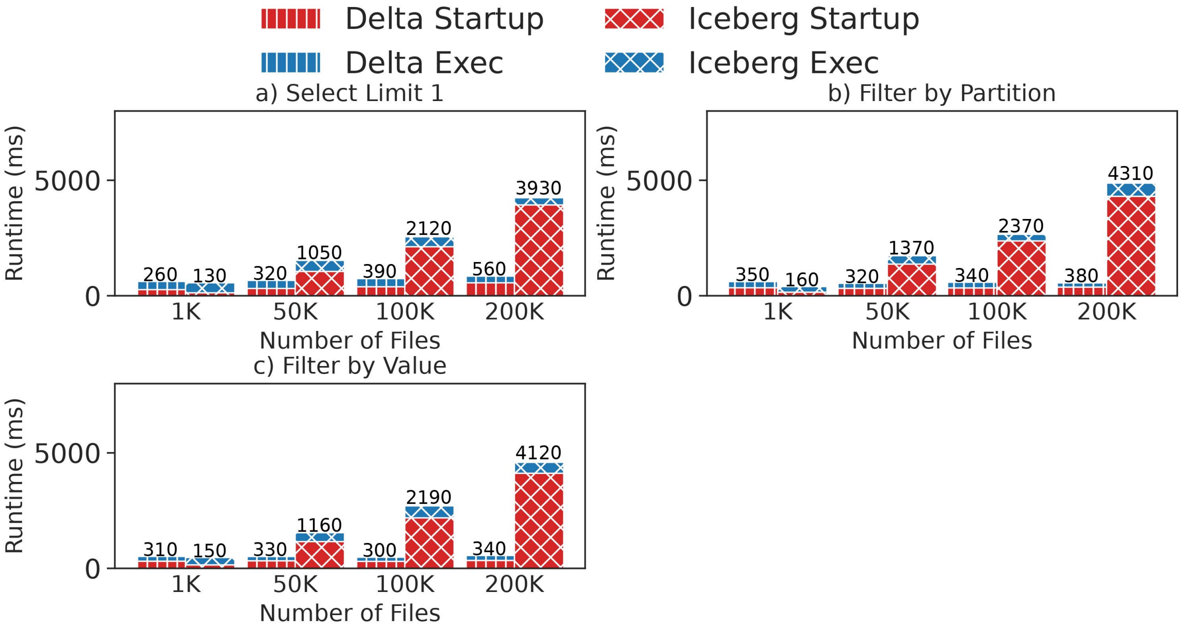
Hierarchical Format

- Used by Iceberg.
- Each table’s metadata stored in a tree of manifest files (in Avro).
- Queries planned on a single node.

LHBench Metadata Benchmark

- We run high-selectivity queries (where query planning is the bottleneck) over TPC-DS data .
- We measure query planning time with different metadata management strategies over tables of different sizes.
- We measure performance of Delta and Iceberg with tables containing 1K, 10K, 100K, and 200K 10 MB files (10GB-2TB data)

Tabular Metadata + Distributed Planning Scales Better



Distributed query planning (Delta) is slower for small tables but scales better to large tables.

We Analyze Four Important Aspects of Lakehouse Functionality

- Transaction Management
- Metadata Management
- **Update Performance**
- End-to-end Performance

Lakehouses Must Efficiently Support Updates

- Lakehouse workloads typically include frequent updates, including point updates and upserts.
- Must balance read and write performance.

Current Lakehouses Use Two Update Strategies

Copy-on-Write

- Supported by all three lakehouses.
- Identify files containing records that need updates, then eagerly rewrite them.
- High write amplification, no read amplification.

Current Lakehouses Use Two Update Strategies

Copy-on-Write

- Supported by all three lakehouses.
- Identify files containing records that need updates, then eagerly rewrite them.
- High write amplification, no read amplification.

Merge-on-Read

- Supported by Iceberg/Hudi, coming soon to Delta.
- Write changes to auxiliary files, reconcile at query time.
- Low write amplification, high read amplification.

LHBench Provides Two Update Benchmarks

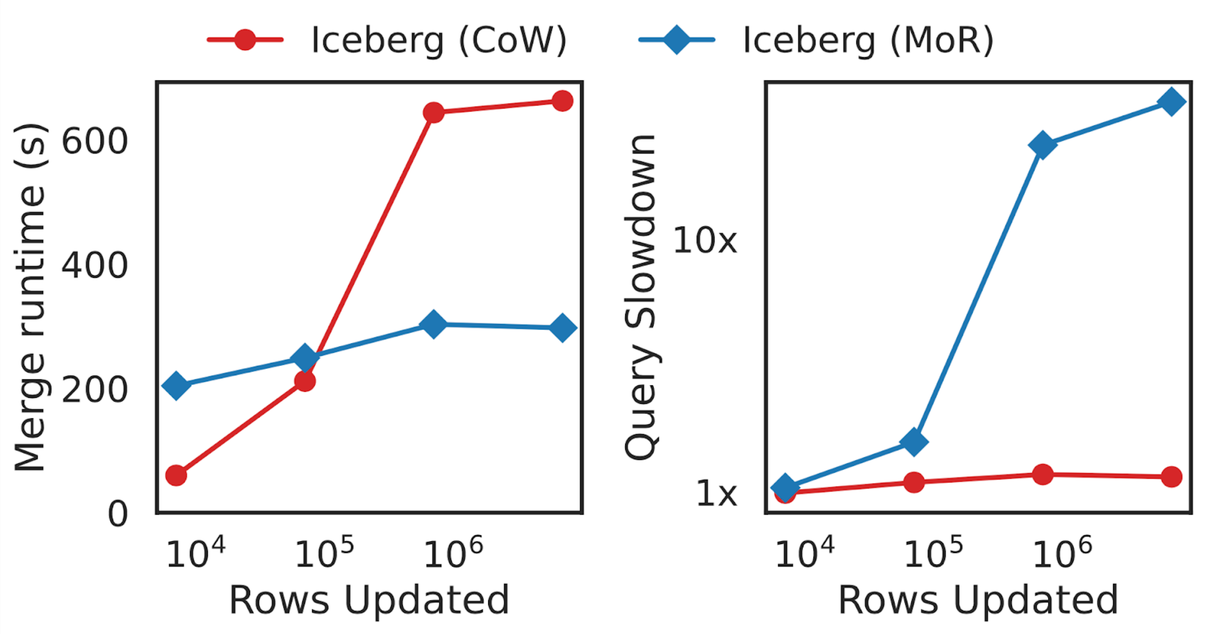
Merge Benchmark

- Directly compare copy-on-write to merge-on-read
- Scale up merge size continuously
- Compare merge and query times

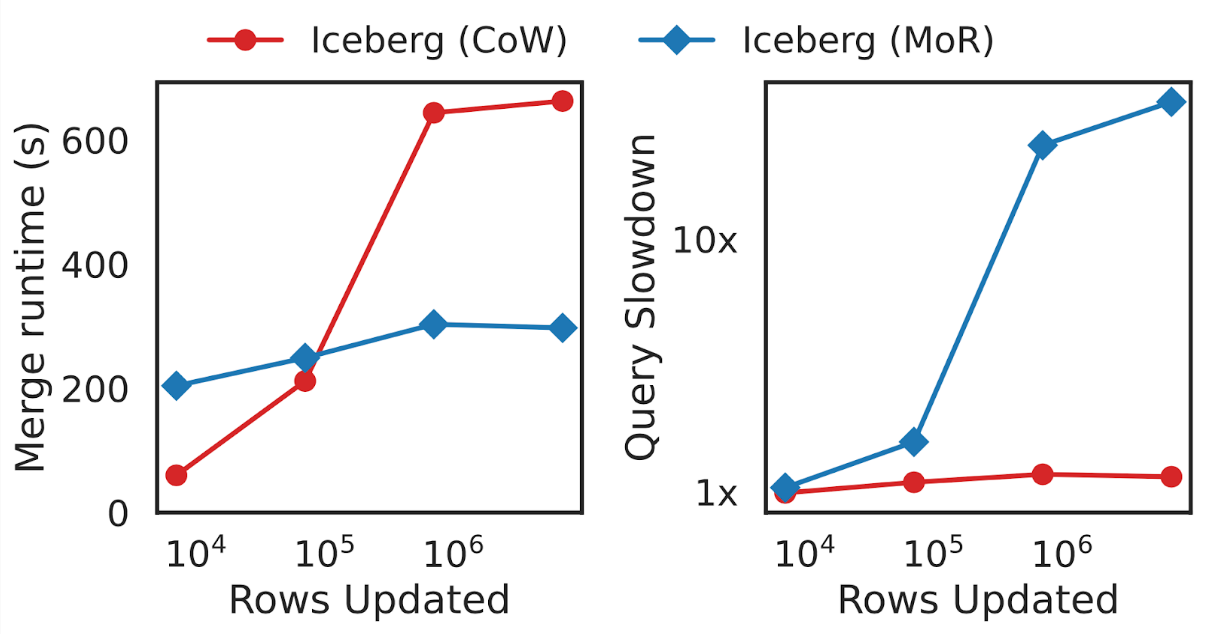
TPC-DS Refresh Benchmark

- 10 refresh rounds of 3% data
- TPC-DS queries after load and after refreshes
- See paper for details!

LHBench Merge Benchmark

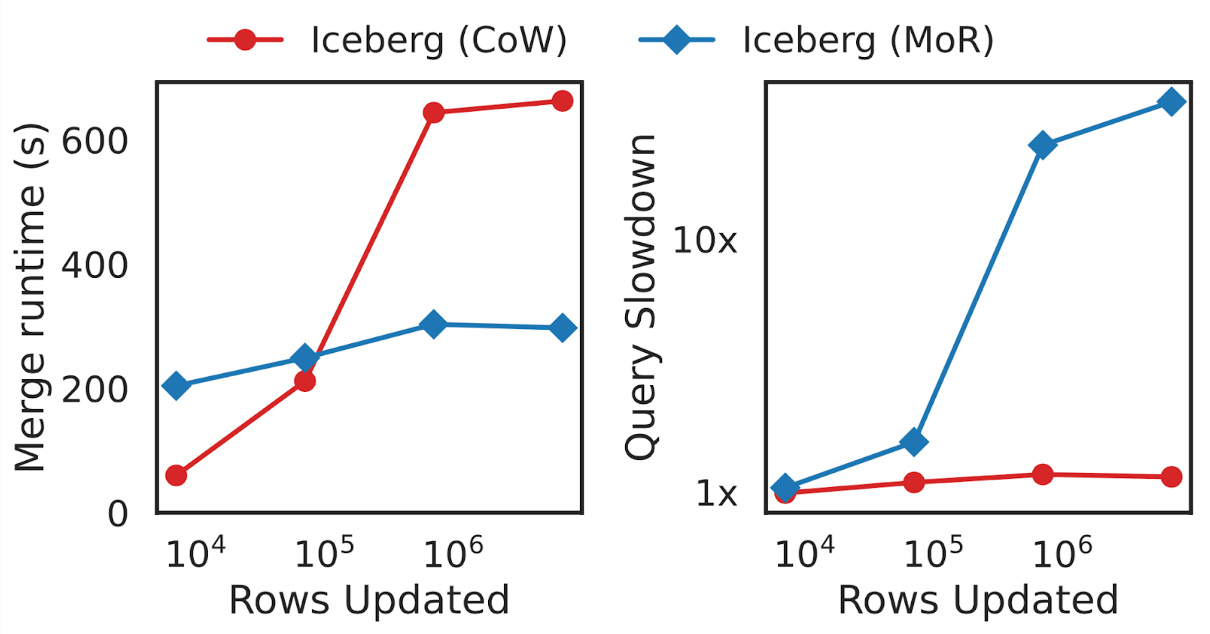


LHBench Merge Benchmark



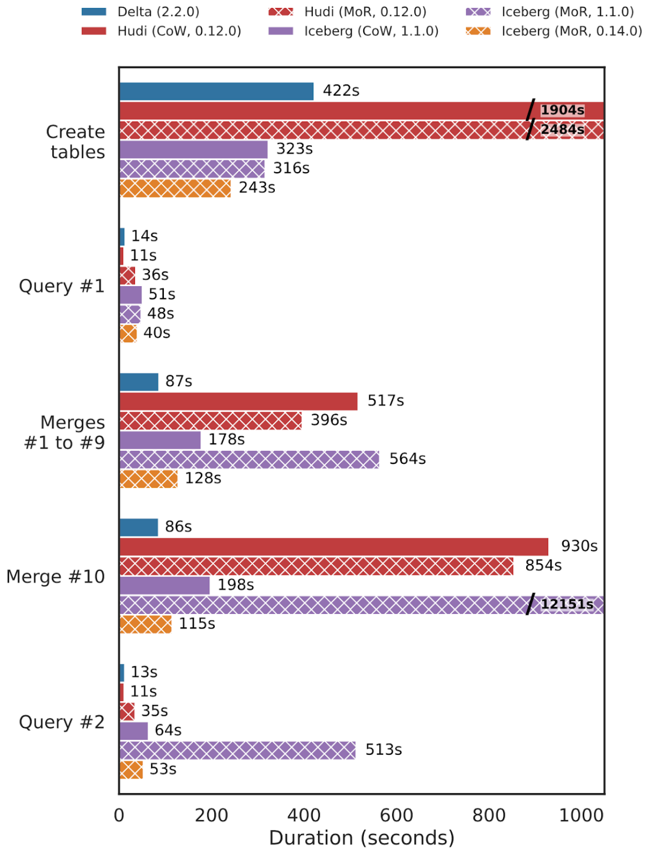
Iceberg MoR is 3× faster at the largest merge configuration (100MB)

LHBench Merge Benchmark



At 100MBs merged MoR causes 10x query slowdown.

LHBench Refresh Benchmark



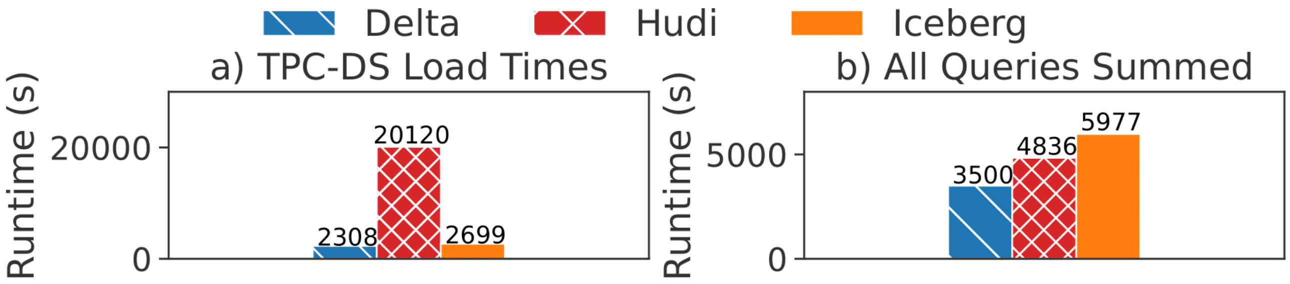
We Analyze Four Important Aspects of Lakehouse Functionality

- Transaction Management
- Metadata Management
- Update Performance
- **End-to-end Performance**

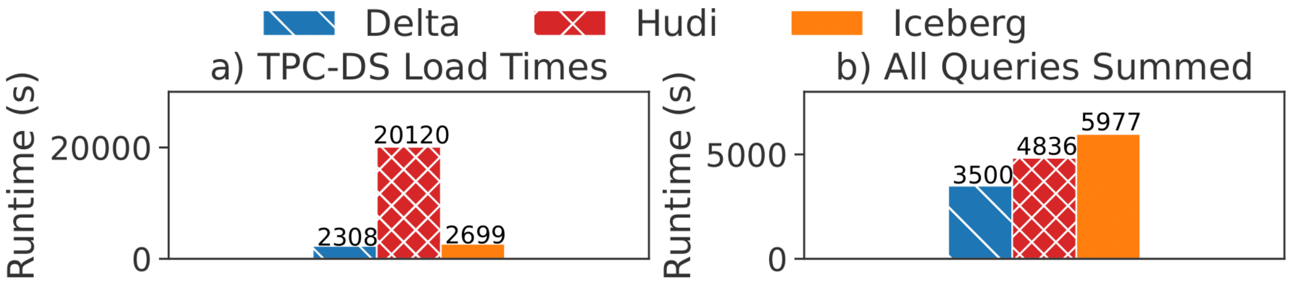
3TB TPC-DS

- We load 3TB of data and run all TPC-DS queries
- We measure load time
- We run each query 3 times and measure the median time

LHBench Analyzes E2E Performance Using TPC-DS

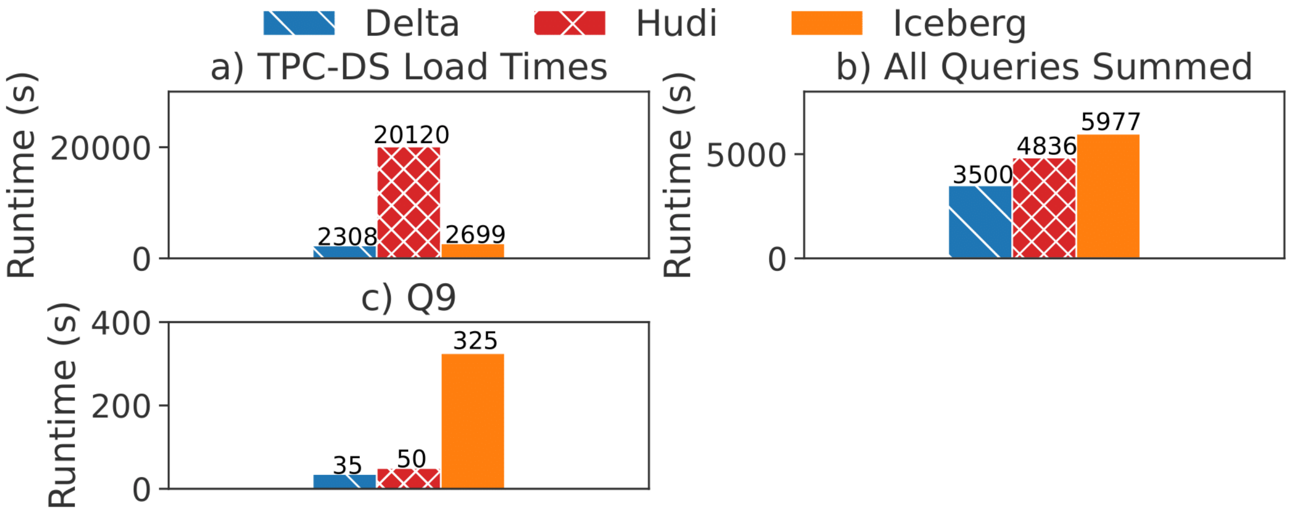


Hudi Loads Are Slow Due to Preprocessing



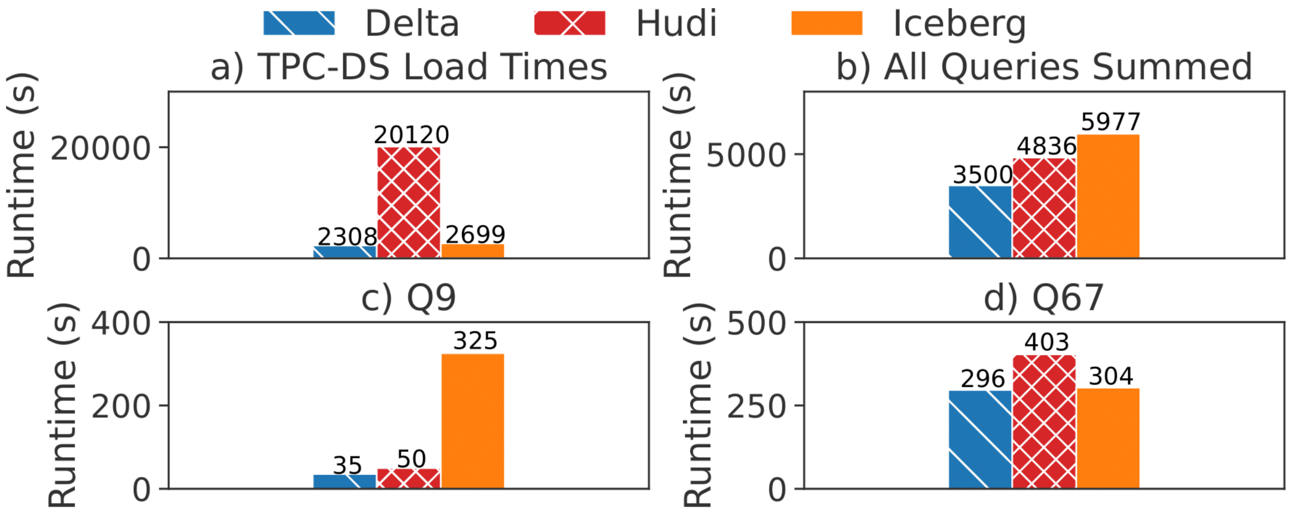
Hudi is optimized for keyed upserts, does expensive key uniqueness checks + key redistribution on each update.

Query Performance Influenced by Implementation Differences



Iceberg uses immature Spark Data Source v2, optimizes queries less (e.g., Q9)

Query Performance Influenced by Implementation Differences



Hudi stores data in many files.

Example: In Q67, Delta/Iceberg store table partitions in 1 file, Hudi uses 22.

Summary

- Lakehouses are important and exciting but still immature: lots of research to do on improving their performance and functionality.
- LHBench measures key lakehouse performance characteristics in challenging scenarios, hopefully helpful for future researchers!

https://github.com/lhbench/lhbench

Many Open Challenges

- How can lakehouse systems best balance read/write performance?
- Increase QPS under concurrency for lakehouse systems.
- Support transactions across multiple tables.

https://github.com/lhbench/lhbench