Delta Lake, Apache Iceberg, and Apache Hudi: A Comparative Analysis and Performance Evaluation

MSc Thesis

Ioannis Vogkas

School of Electrical and Computer Engineering
National Technical University of Athens
MSc Data Science and Machine Learning

June 2025

Overview

Purpose of the Study

Purpose of the Study

This study aims to analyze and experimentally evaluate the performance of three popular Log-Structured Table (LST) data storage systems, Delta Lake, Apache Iceberg, and Apache Hudi, under a variety of workload scenarios.

Key Objectives:

- Study the design and architecture of each system.
- Conduct experimental performance evaluations using realistic workload scenarios.
- Identify key differences, strengths, and limitations of each system across different scenarios.

Overview

Log-Structured Tables (LSTs)

What are Log-Structured Tables (LSTs)?

- Log-Structured Tables (LSTs) are a modern data storage approach.
- LST systems separate storage into two layers: data files and a metadata layer that tracks changes.
- They store data in immutable columnar files, such as Parquet.
- Instead of modifying data in-place, LSTs create new table versions to record updates, which are tracked by the metadata layer.
- The three most widely used LST systems are Delta Lake, Apache Iceberg, and Apache Hudi.

Log-Structured Tables (LSTs)

Advantages:

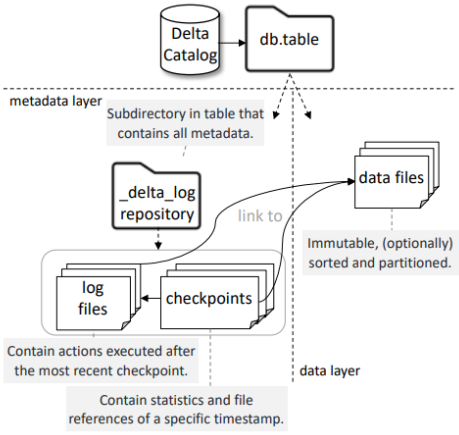
- Combine the performance benefits of columnar storage with efficient update capabilities.
- Enable time-travel queries, allowing access to historical data versions.
- Support for concurrent reads and writes through Multi-Version Concurrency Control (MVCC).

Limitations:

- Over time, the accumulation of data files can lead to degraded query performance and increased storage overhead.
- Metadata management can become complex.

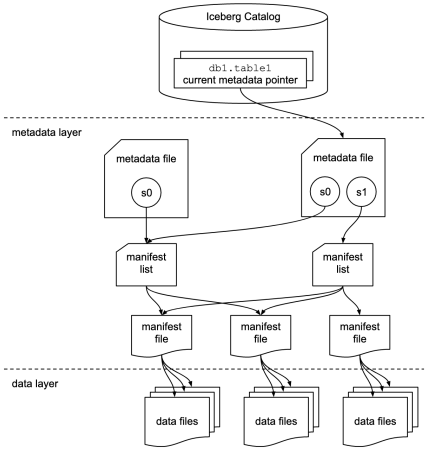
Delta Lake

- Each table modification is recorded in a log file with a unique ID.
- Checkpoint files summarize these smaller log files (by default every 10 transactions) to speed up access to the current table state, avoiding the need to scan them all.



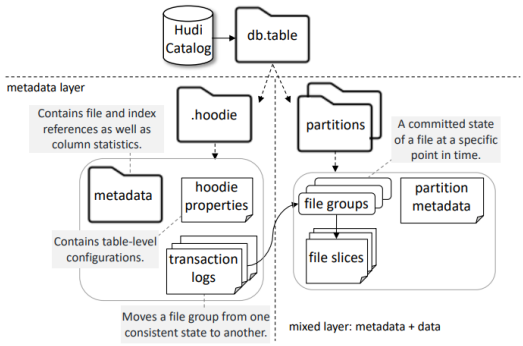
Apache Iceberg

- Iceberg organizes files hierarchically to represent the table's state.
- At the top level, a metadata file stores table information (eg table schema, partition information) and is atomically replaced whenever the table is updated.
- This metadata file points to manifest list files.
- Each manifest list corresponds to a snapshot of the table at a specific moment in time.
- Manifest list files reference manifest files.
- Manifest files contain information about the data files, including their locations and relevant statistics.

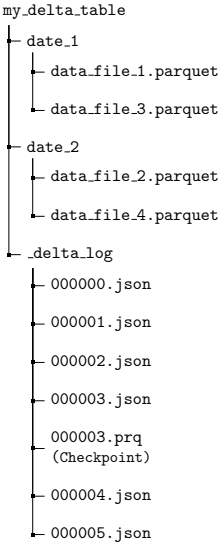


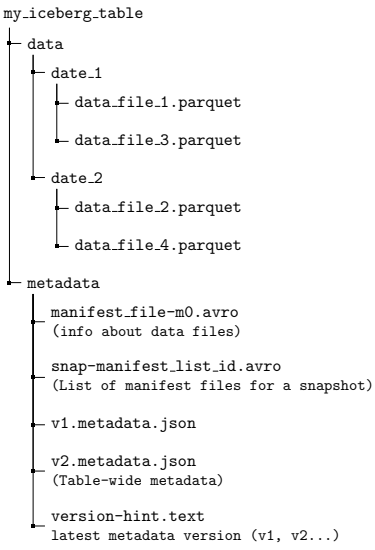
Apache Hudi

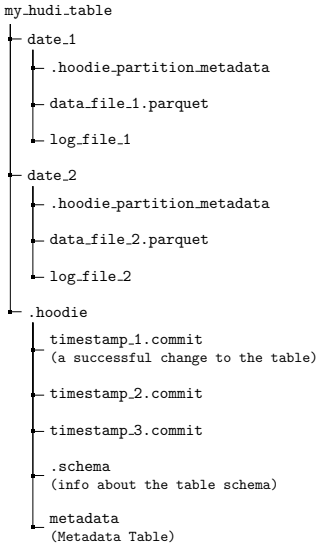
- Hudi is based on three main parts: the Timeline, the Metadata Table, and File Groups with File Slices.
- The Timeline tracks table modifications by recording each action as a file labeled with its commit start time.
- The Metadata Table stores information such as physical file paths and indexes on the table's data files.
- A File Group consists of multiple versions of data called File Slices.
- Each File Slice contains a Base File (Parquet) that holds a version of the data.



Storage Layout Overview







Key Features Comparison

Feature	Delta Lake	Apache Iceberg	Apache Hudi
Metadata	Transaction log files & Checkpoints every 10 commits	Hierarchical metadata structure (metadata.json, Manifest Lists, Manifest Files)	Timeline (action history) & Metadata Table (index of file paths)
Write Strategies	Copy-on-Write (CoW)	Copy-on-Write (CoW) & Merge-on-Read (MoR)	Copy-on-Write (CoW) & Merge-on-Read (MoR)
Concurrency Control	Optimistic Concurrency Control (OCC) & Serializable Writes & Snapshot Isolation on reads	Optimistic Concurrency Control (OCC) & Serializable Writes & Snapshot Isolation on reads	Optimistic Concurrency Control (OCC) & Non-Blocking Concurrency Control (NBCC) (since v1.0.1) & Snapshot Isolation on reads
Delete Strategies	Copy-on-Write (CoW)	Position Delete Files, Equality Delete Files	Soft Deletes & Hard Deletes
Table Maintenance Operations	Compaction (OPTIMIZE), Log Compaction, Optimized Writes, etc.	Compaction (RewriteDataFiles), Expiring Snapshots, Deleting Orphan Files, etc.	Compaction (MoR only), Clustering, Cleaning, Rollbacks, etc.

Related Research and Contribution of This Study

- Research on Log-Structured Tables (LSTs) is active, with studies and technical reports comparing their theoretical and experimental performance.
- According to recent research by Jesús Camacho-Rodríguez et al., various benchmarks, especially TPC-DS, are commonly used to compare LSTs but do not fully capture key aspects such as file accumulation, table maintenance operations, and concurrent access. To address this, they developed LST-Bench, a benchmark specifically designed to assess LSTs using real-world workloads.
- LST-Bench is a new and promising benchmark that hasn't yet been widely adopted. This study uses it to provide a thorough analysis and comparison of LSTs, filling gaps in earlier research.

Overview

LST-Benchmark Overview

- **LST-Bench** is a specialized benchmarking framework for evaluating the performance of Log-Structured Tables (LSTs).
- Unlike general benchmarks, like TPC-DS, it focuses on LST-specific aspects often missed by traditional benchmarks.
- It evaluates critical aspects such as:
  - Long-term performance stability
  - Impact of frequent updates and deletes
  - Importance of table maintenance operations (e.g., file compaction)
  - Time travel capabilities (retrieving historical data versions)
- It is based on TPC-DS, but introduces new tasks and workloads tailored for LSTs.

LST-Bench Tasks

These are the core tasks defined in the LST-Bench:

- **Load Task (L)**: Loads the initial dataset into the tables.
- **Single User Task (SU)**: Executes a set of 99 complex read-only queries.
- **Data Maintenance Task (DM)**: Performs inserts, updates, and deletes on the tables.
- **Optimize Task (O)** : Triggers table maintenance operations:
  - *Delta Lake*: `OPTIMIZE` (compacts small data files)
  - *Iceberg*: `RewriteDataFiles` (compacts small data files)
  - *Hudi*: `Clustering` (optimizes data layout to improve data locality)
- **Time Travel Task**: Executes the 99 read-only queries (SU task) on earlier snapshots of the data (historical versions).

LST-Bench Workloads

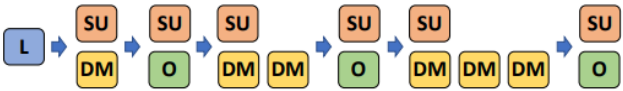
Benchmark's workloads that consist of the previously mentioned tasks:

W1. Longevity: This workload evaluates LSTs under long-term usage with continuously changing data.

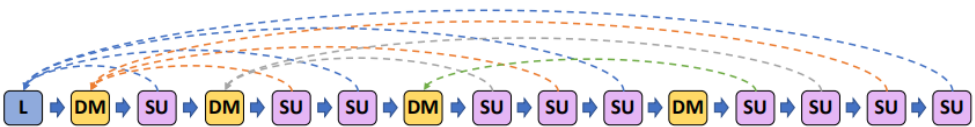
W2. Continuous Updates and Table Maintenance Operations: Measures the impact of table maintenance operations during frequent data changes.

LST-Bench Workloads

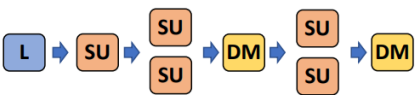
W3. Read/Write Concurrency: Assesses the impact of concurrent read, write, and table maintenance operations.



W4. Time Travel: Evaluates LSTs' ability to handle queries on past data versions.



W5. Concurrent User Reads: Examines simultaneous reads by multiple users



Data Used in the Study

- Synthetic data created with the `dsdgen` tool (TPC-DS data generator).
- The dataset simulates a large retail business operating through three sales channels: physical stores, online store, and catalogs.
- Organized in a snowflake schema with 7 fact tables and 17 dimension tables.
- A total of 10 GB of data was used (some for the initial table load, and the rest during the DM tasks).

Metrics Used for LST Evaluation

- **Latency** (Measures the execution time of each task)
  - Collected using benchmark tool that store start/end times.
- **Stability** (A metric proposed by LST-Bench, designed to measure the ability of LSTs to maintain consistent performance over time despite continuous data updates and potential accumulation of many files.)
  - Measured using the *Stability Degradation Rate (SDR)*:

$$\text{SDR} = \frac{1}{n} \sum_{i=1}^{n} \frac{M_i - M_{i-1}}{M_{i-1}}$$

M_i: Execution time of the i-th Single User phase.
n: Total number of SU phases.

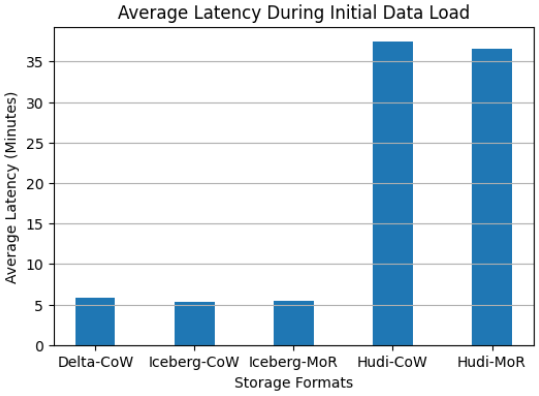
- **Resource Utilization Metrics**
  - CPU usage (%), disk usage (Disk I/O %), disk I/O operations, and data transfer (shuffle data) between Virtual Machines (Network Data Transfer).
  - Metrics were collected using an Azure Log Analytics Workspace. Three VMs used in the experiments were connected to this workspace, with an Azure Monitor Agent installed on each that continuously sent metric logs in real time.

Experimental Setup

- Experiments conducted on a distributed cluster environment based on Apache Spark 3.3.1 and Apache Hadoop 3.3.6 for data processing and storage, respectively.
- The cluster consisted of three virtual machines (VMs) (one master node and two worker nodes).
- Each node was equipped with 4 CPU cores and 8 GB of RAM.
- The LST versions used in the experiments were Delta Lake v2.2.0, Apache Iceberg v1.1.0, and Apache Hudi v0.12.2.

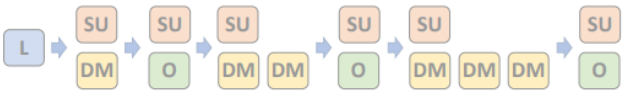
Overview

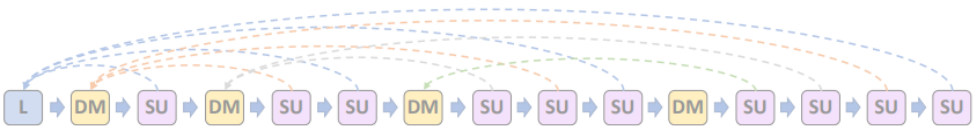
Initial Data Load

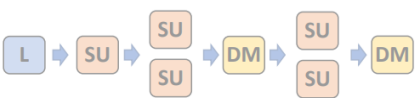


Hudi exhibits significantly higher loading times compared to Delta and Iceberg due to uniqueness key checks and record distribution to File Groups during loading. Modifying Hudi to skip uniqueness checks reduced loading time, however, this disables updates, deletions, and time-travel features.

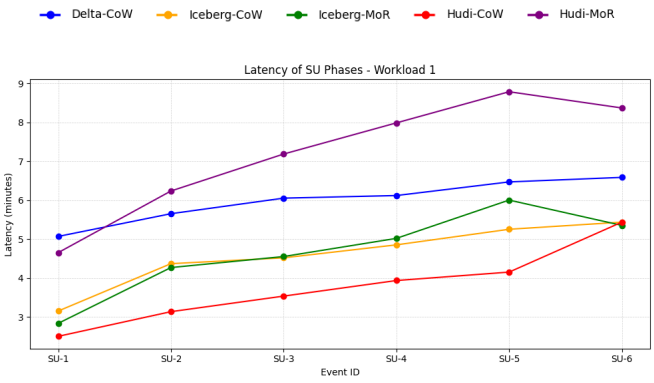
W1 - Long-Term Usage





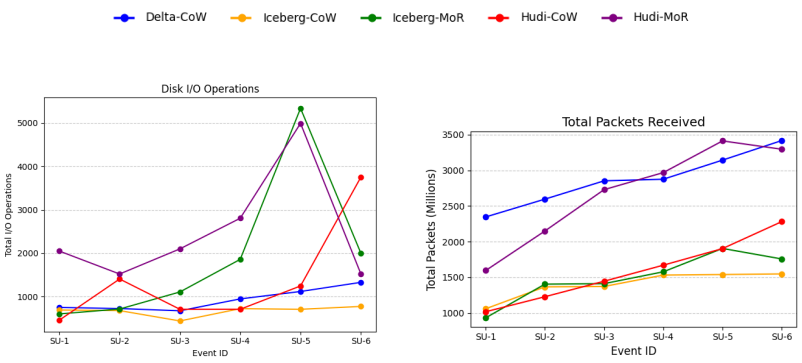


W1 - Latency of SU phases



- SU time grows with file accumulation.
- Delta Lake is the most stable over time (probably due to checkpoints).
- Hudi-MoR is much slower than CoW (log files seem to slow down execution).
- Iceberg-CoW performs better than MoR as more data updates are applied.

W1 - Disk I/O Operations and Data Shuffle



- MoR strategy causes increasing disk I/O operations due to merging many files on reads, CoW remains stable. MoR (Iceberg and Hudi) shows reduced disk I/O operations in final SU phase (probably due to automatic manifest compression for Iceberg and default file merging for Hudi).

W1 - CPU & Disk I/O Utilization

	Load	SU	DM
Delta-CoW	Moderate	Low	Low
Iceberg-CoW	Moderate	Low	Low
Iceberg-MoR	Moderate	Low	Low
Hudi-CoW	High	Moderate	Low
Hudi-MoR	High	Moderate	Low

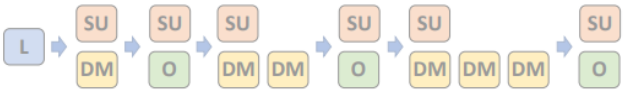
CPU % Usage

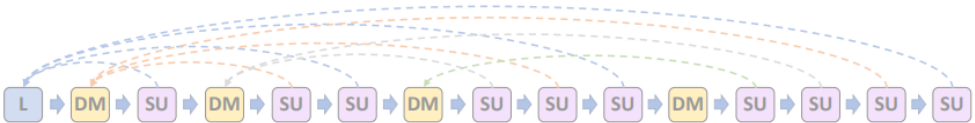
	Load	SU	DM
Delta-CoW	Low	Low	Moderate
Iceberg-CoW	Low	Low	Moderate
Iceberg-MoR	Low	Moderate	Moderate
Hudi-CoW	High	Low	Moderate
Hudi-MoR	Moderate	High	Moderate

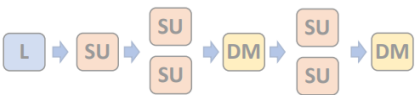
Disk I/O % Utilization

- Hudi demonstrates higher CPU and disk util during initial load due to uniqueness key checks and file grouping.
- MoR strategy leads to increased disk util during SU phases, as multiple files are needed during reads. In contrast, CoW reduces disk util by storing data in single files.
- During Data Maintenance phases, CPU usage is low while disk util rises, indicating I/O-bound waiting.

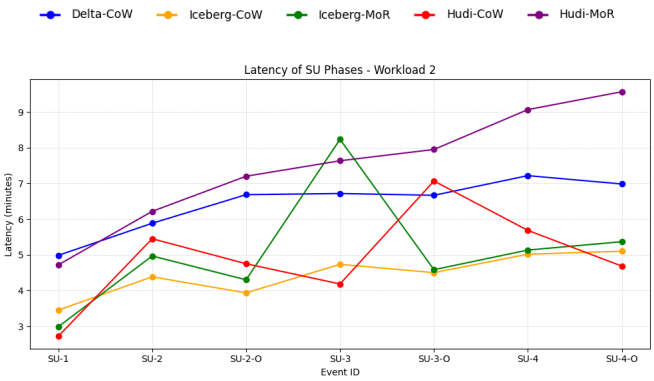
W2 - Continuous Updates and Table Maintenance Operations





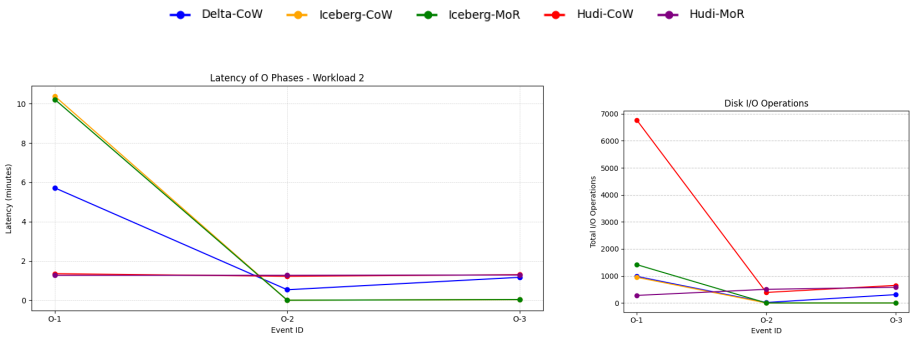


W2 - Latency of SU phases



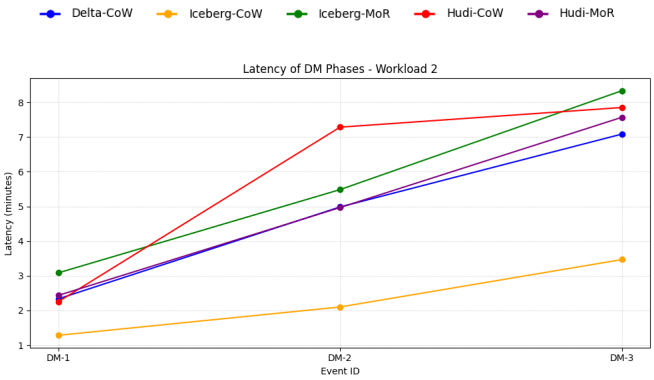
- Hudi doesn't benefit from the Clustering.
- In contrast, Iceberg demonstrates a significant performance improvement from the Compaction process.
- For Delta Lake, a minimal improvement in phases 3 and 4.
- Delta Lake records the lowest value in the stability metric.

W2 - Optimize Tasks



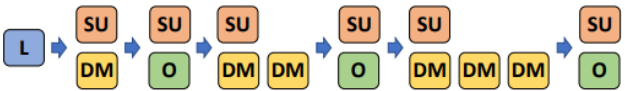
- Iceberg shows a major delay in the O-1 phase (due to group-by-group file compaction).
- Delta Lake and Iceberg execute Optimize significantly faster when a previous optimization was recently performed.
- For Hudi, in the O-1 phase, increased disk I/O operations suggests significant changes to the data layout in the original table.

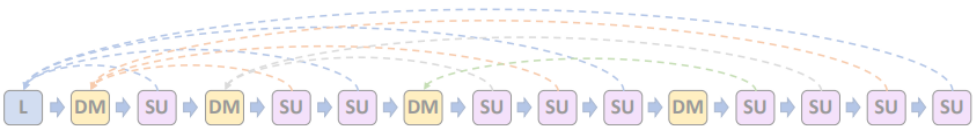
W2 - Latency of DM phases

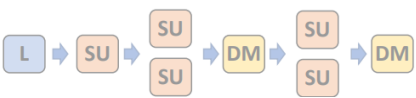


During update and delete operations (DM phases), Iceberg-CoW completes tasks significantly faster than other LSTs.

W3 - Read/Write Concurrency

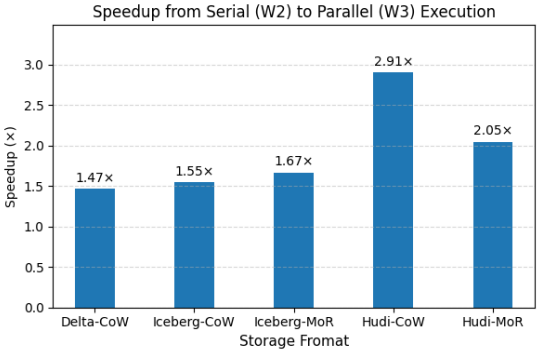






W3 - Speedup

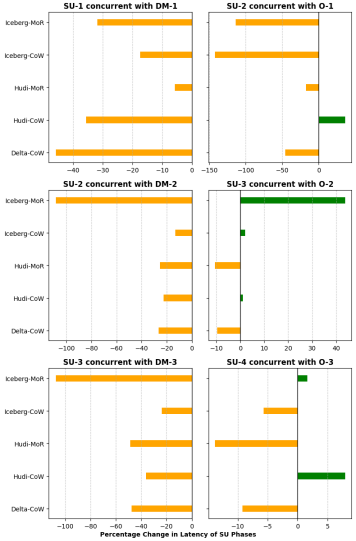
Speedup compares execution time: serial (W2) vs parallel (W3). Speedup $= \frac{T_{\text{serial}}}{T_{\text{parallel}}}$



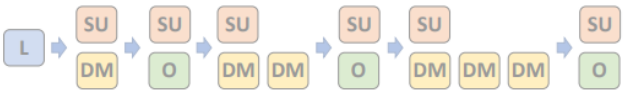
- Parallel execution reduces total runtime across all systems, with Hudi showing the highest speedup (>2).
- It is important to analyze each phase individually to understand the cause of the observed speedup.

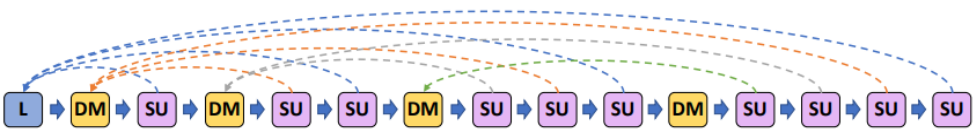
W3 - Impact of Parallel Execution on Single User Phases

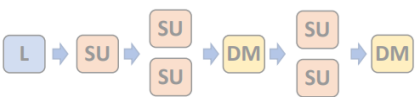
- The figure depicts how execution times of Single User (SU) phases are affected when run in parallel with Data Maintenance (DM) or Optimize tasks, compared to running alone.
- For SU phases is observed increased latency during parallel execution and higher CPU & disk utilization (which is expected due to resource contention).
- For Hudi-CoW, the SU phases do not slow down when executed in parallel with Optimize task (Clustering), which explains the high speedup observed.
- SU phases may slow down, but parallelism leads to faster workload completion overall.



W4 - Time Travel Queries

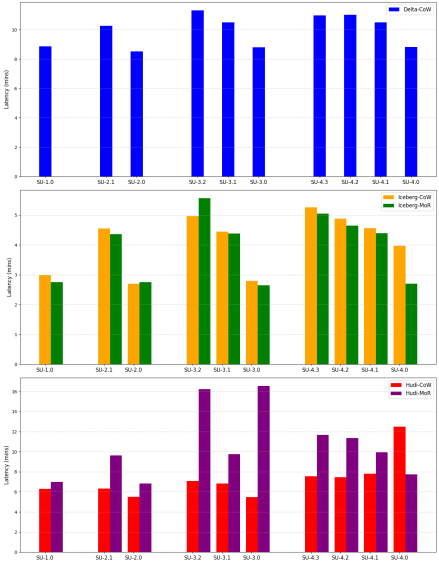




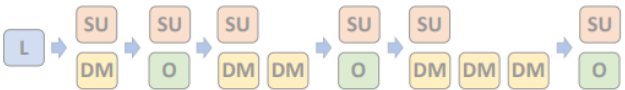


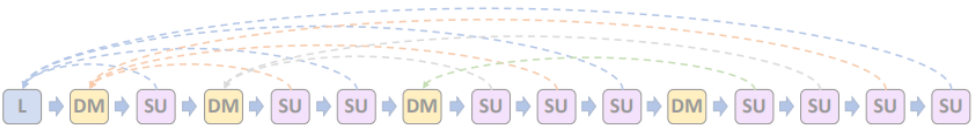
W4 - Time Travel Queries Latency

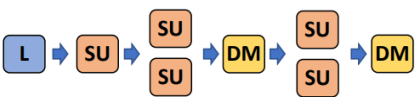
- Each phase is labeled as `SU-i.j`, where:
  - `i` indicates the number of DM operations applied.
  - `j` refers to the version of the data being queried.
- For Delta Lake and Iceberg, time-travel query performance remains stable across versions, regardless of when the query is executed. New files from later updates do not impact queries on older versions.
- Hudi generally aligns with this behavior, but some cases show delays.
- Delta Lake is the most stable (according to stability metric).



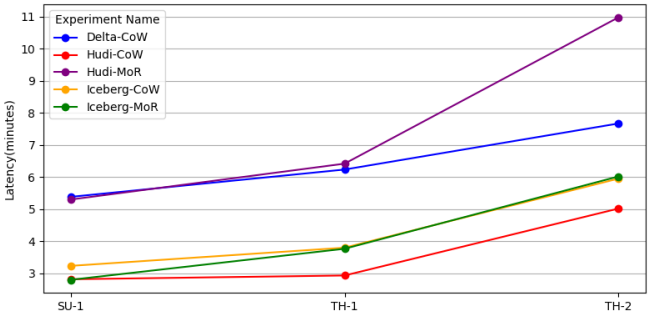
W5 - Concurrent User Reads







W5 - Latency



- As expected, slight performance degradation occurs with concurrent users.
- Testing with over two simultaneous users caused execution errors in Hudi, indicating possible limitations under concurrent read workloads.

Overview

Main Conclusions

- Hudi shows significantly longer data loading times due to key uniqueness checks and record distribution across File Groups.
- The Copy-on-Write (CoW) strategy is more efficient for data reads, as it avoids file merges required by the Merge-on-Read (MoR) strategy.
- Delta Lake demonstrates the highest performance stability during Single User phases, with query latency less affected by additional files from updates.
- For the Optimize process, Iceberg benefits significantly from file merging, in contrast to Hudi, where Clustering does not noticeably improve performance.
- Parallel execution led to overall speedup despite some individual phases (eg Single User) slowing down due to resource contention.
- During Time Travel queries, Delta Lake and Iceberg maintained stable performance across data versions, while Hudi showed generally good but less consistent results.
- Query latency slightly increased with concurrent users, and Hudi faced errors at higher concurrency levels.

Future Work

- Investigate the performance and interaction of LST systems with alternative processing engines (eg Trino) and storage infrastructures (eg Azure Data Lake Storage).
- Investigate the performance of LST systems under various configurations and optimizations beyond default settings to uncover additional potential improvements.
- Run experiments on larger clusters with more data to better simulate real-world and large-scale scenarios.

Thank you!

Questions?