Microsoft

Microsoft Azure Data
Gray Systems Lab

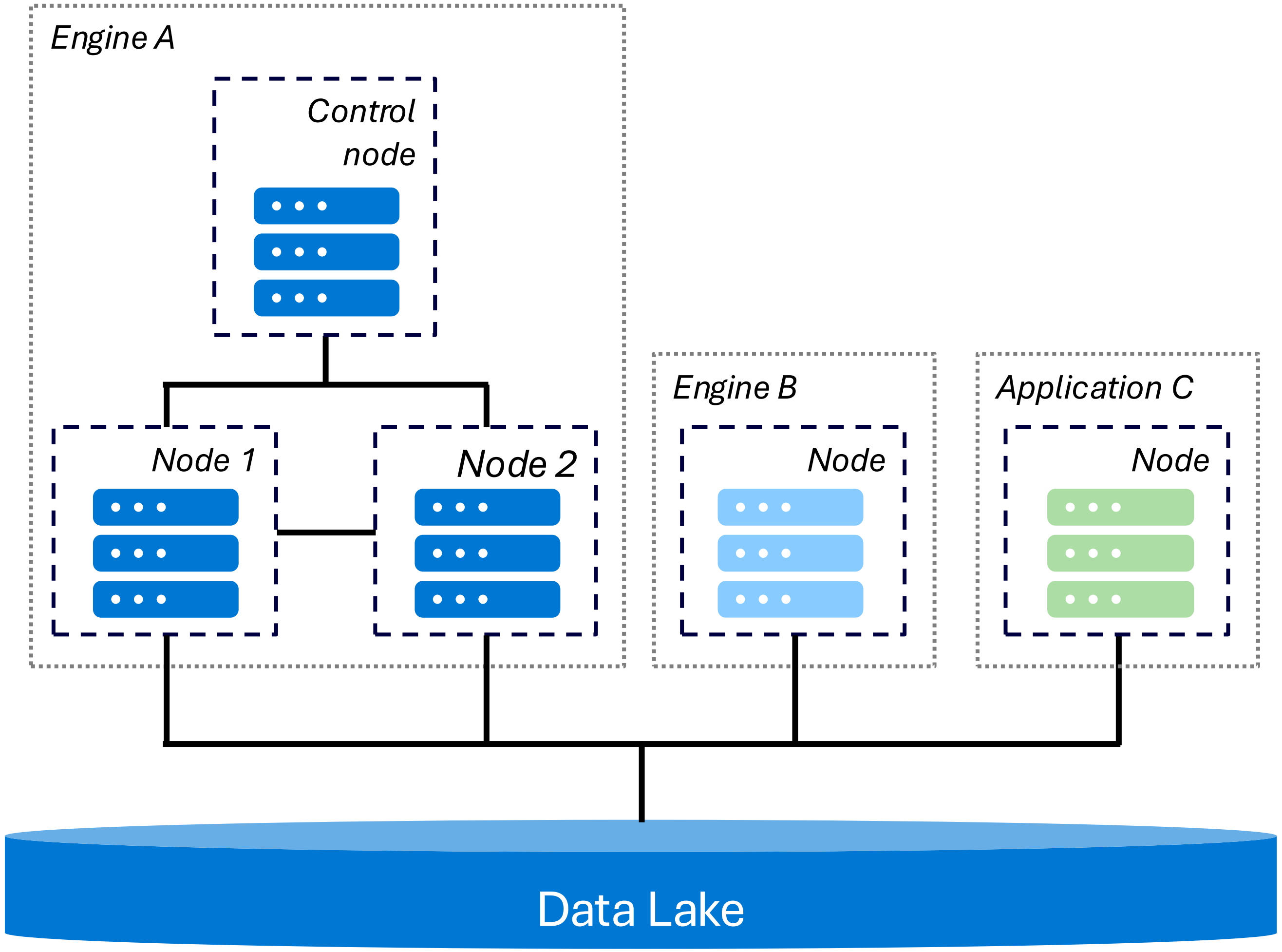
From Paper to Practice: Leveraging LST-Bench to Evaluate Lake-Centric Data Platforms

Jesús Camacho-Rodríguez
Gray Systems Lab (GSL)
Microsoft Azure Data

DBTest ’24 Keynote
June 9th, 2024

1. Data Lake-Centric Architectures and LSTs
2. Motivation
3. Tool and Benchmark Design
4. Implementation
5. Results
6. Status, Road Ahead, and Key Takeaways

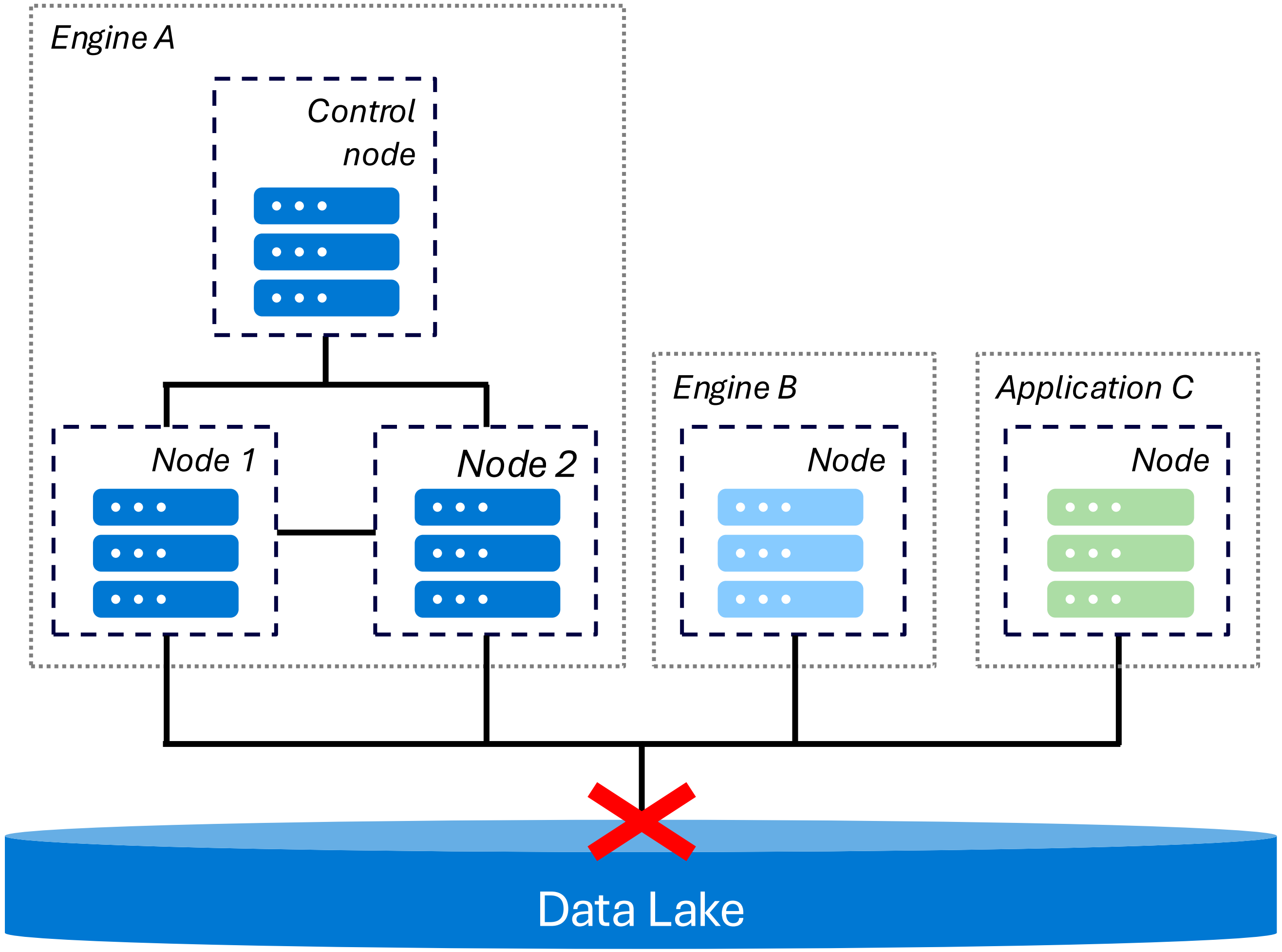
Data Lake-Centric Architecture



Benefits

- **Flexible Scalability:**
  Scale storage and compute independently for better efficiency and cost savings
- **Streamlined Workflows:**
  Eliminate data silos for simpler data movement across systems
- **Reduced Lock-In:**
  Allow any engine to directly access data storage, offering flexibility to choose the best solution for each application

Data Lake-Centric Architecture



Challenges

- Ensuring consistency and isolation of complex read and write transactions
- Data lakes excel in scalability and durability but lack necessary concurrency and recovery capabilities

Log Structured Tables (LSTs) – Overview

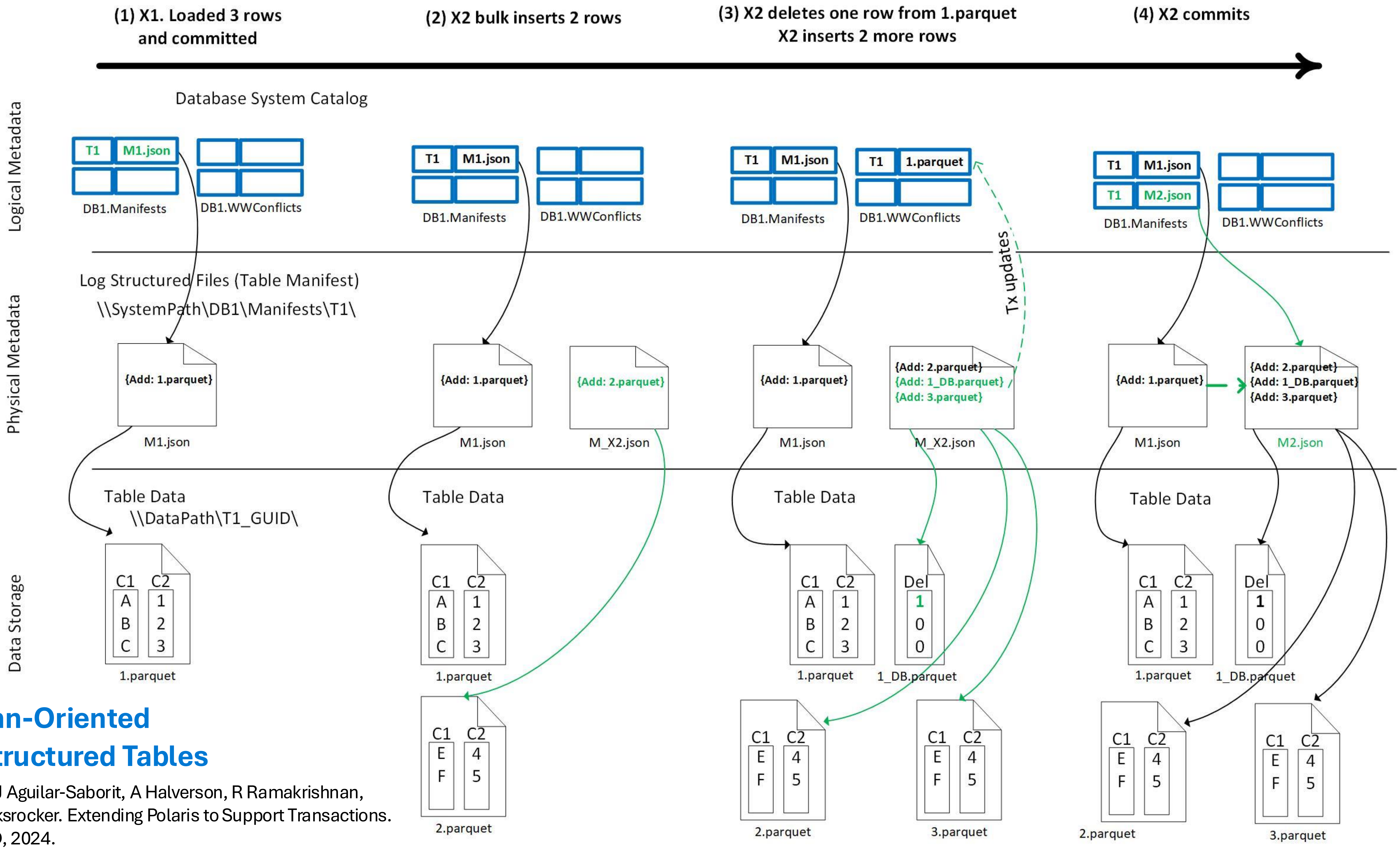
- Popular OSS projects for updatable tables: **Delta Lake**, **Apache Hudi**, **Apache Iceberg**, and **Apache Paimon (Incubating)**
  - **Goal:** Provide additional functionality (transactions, indexes, time travel, cloning) on top of immutable files (Parquet) stored in the data lake

LSTs – Data and Protocol

- Many types of data are required to represent a LST
  - Logical metadata (e.g., table definitions)
  - Physical metadata (how to find data for a table)
  - Data files (Parquet type, column oriented, store user data)
  - Delete bitmaps (describes which rows are deleted)
  - Clone info (reference counting for files)

- Where this data is stored
  - Files in cloud storage

- Approach to updates — NOT in-place in page-files!
  - Copy-on-Write (CoW) — Affected data files copied over with changes reflected
  - Merge-on-Read (MoR) — For affected data files, ONLY changes recorded; reads must merge changes



Column-Oriented Log-Structured Tables

[AHR+24] J Aguilar-Saborit, A Halverson, R Ramakrishnan, and K Bocksrocker. Extending Polaris to Support Transactions. In SIGMOD, 2024.

LSTs, Engines, and Platforms

- While LSTs share the same goal, their architecture and implementation vary
  - CoW and MoR support, metadata caching, distributed planning, storage optimizations

- Performance depends on numerous factors

LSTs	Policies	Engines
DELTA LAKE	databricks	Apache Spark
Apache hudi	AWS Glue Data Catalog	Power BI
ICEBERG	Tabular	trino
Apache Paimon™	AutoOptimize	DuckDB
	OpenHouse	presto
		Synapse Data Warehousing

- Limited work on a **comprehensive framework for LSTs evaluation**

Existing Approaches to Evaluate LSTs

- Experimental Evaluation: LH-Bench [JKP+23], Brooklyn Data, DataBeans
  - Ad-hoc approaches
  - Typically rely on **TPC-DS**, standard OLAP benchmark
  - Subset of standard queries or handcrafted queries

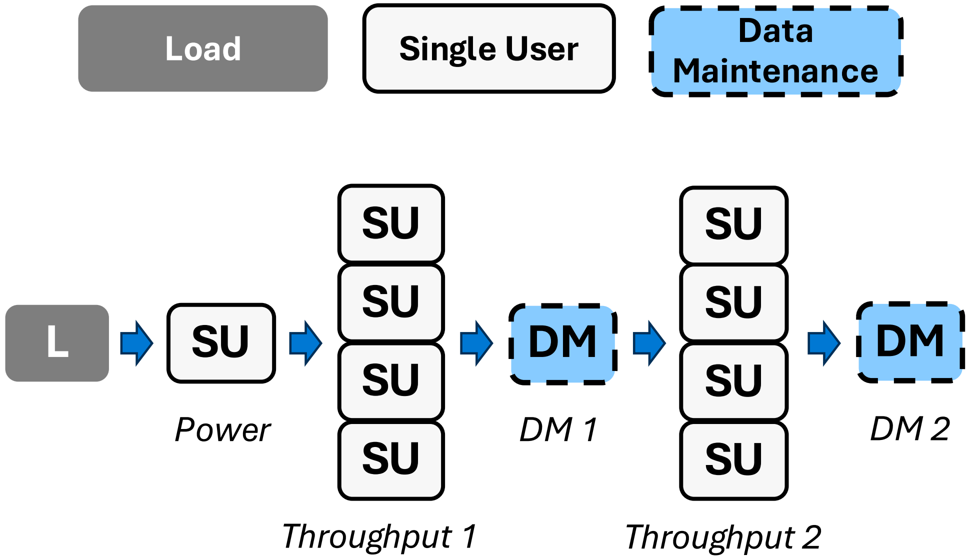
Limitation 1: Unable to extend to new engines, datasets, and scenarios beyond traditional OLAP tasks, despite the continuous expansion of data-lake centric architectures with new use cases

[JKP+23] P Jain, P Kraft, C Power, T Das, I Stoica, and M Zaharia.
Analyzing and Comparing Lakehouse Storage Systems. In CIDR, 2023.

TPC-DS Overview

Workload:

Up to 99 ~~analytical~~ queries ?



Performance Metric:

Total ~~execution~~ latency ?

Normalized *query throughput per hour*

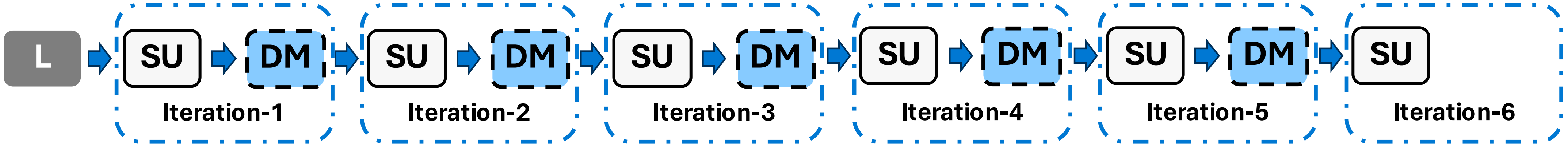
Product of total # of queries executed and scale factor

$$QphDS@SF = \left\lfloor \frac{SF * Q}{\sqrt[4]{T_{PT} * T_{TT} * T_{DM} * T_{LD}}} \right\rfloor$$

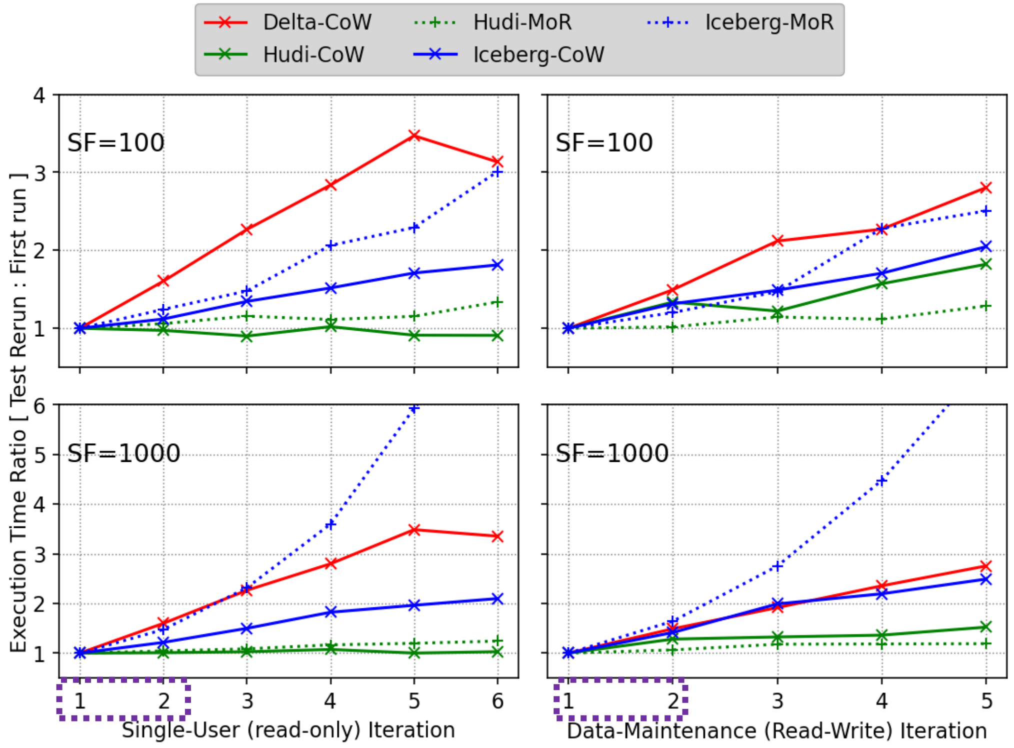
Geometric mean of elapsed time for load, power, throughput, and data maintenance phases

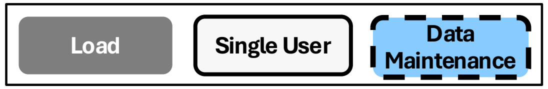
TPC-DS Workload

Question: Regarding trickle updates, is TPC-DS a representative workload?

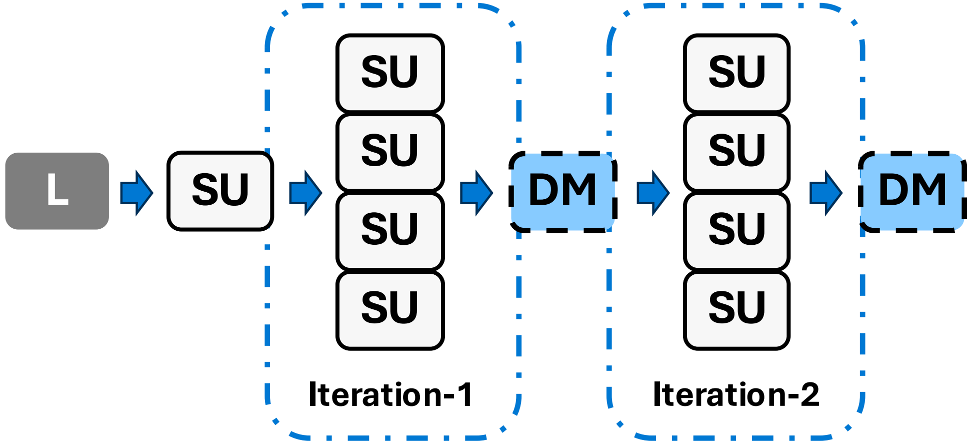


Limitation 2: Failure to expose important characteristics of LSTs that are crucial in real customer scenarios





TPC-DS Metrics



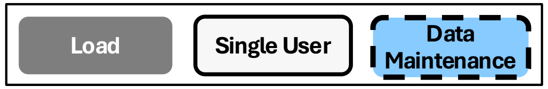
Limitation 3: Lack of metrics to expose important aspects when comparing different LST implementations

$$QphDS@SF = \left\lfloor \frac{SF * Q}{\sqrt[4]{T_{PT} * T_{TT} * T_{DM} * T_{LD}}} \right\rfloor$$

LST	Throughput-QphDS	Inter-test Degradation
Delta	511K	2.7 -> 5.2 hrs **(92%)**
Hudi-CoW*	262K	6.2 -> 6.5 hrs **(5%)**
Hudi-MoR⁑	112K	23 -> 24 hrs **(6%)**
Iceberg-CoW*	549K	2.7 -> 4 hrs **(45%)**
Iceberg-MoR⁑	493K	2.9 -> 5 hrs **(73%)**

* Copy-on-Write mode ⁑ Merge-on-Read mode

SF=1000
Spark (3.3.1), 16 workers, 8 cores, 64 GB
Delta (2.2.0), Iceberg (1.1.0), Hudi (0.12.2)



❶ Data Lake-Centric Architectures and LSTs

❷ Motivation

❸ Tool and Benchmark Design

❹ Implementation

❺ Results

❻ Status, Road Ahead, and Key Takeaways

LST-Bench

- A benchmark tool specifically tailored to evaluate LSTs

Limitation 1: Unable to extend to new engines, datasets, and scenarios beyond traditional OLAP tasks, despite the continuous expansion of data-lake centric architectures with new use cases	➡	**Flexible Workload Representation** Can represent existing benchmarks but easily extends to new scenarios
Limitation 2: Failure to expose important characteristics of LSTs that are crucial in real customer scenarios	➡	**Enhanced Workloads** Introduces *diverse workload patterns*
Limitation 3: Lack of metrics to expose important aspects when comparing different LST implementations	➡	**Relevant Metrics** Includes *metrics extensions* relevant for LSTs

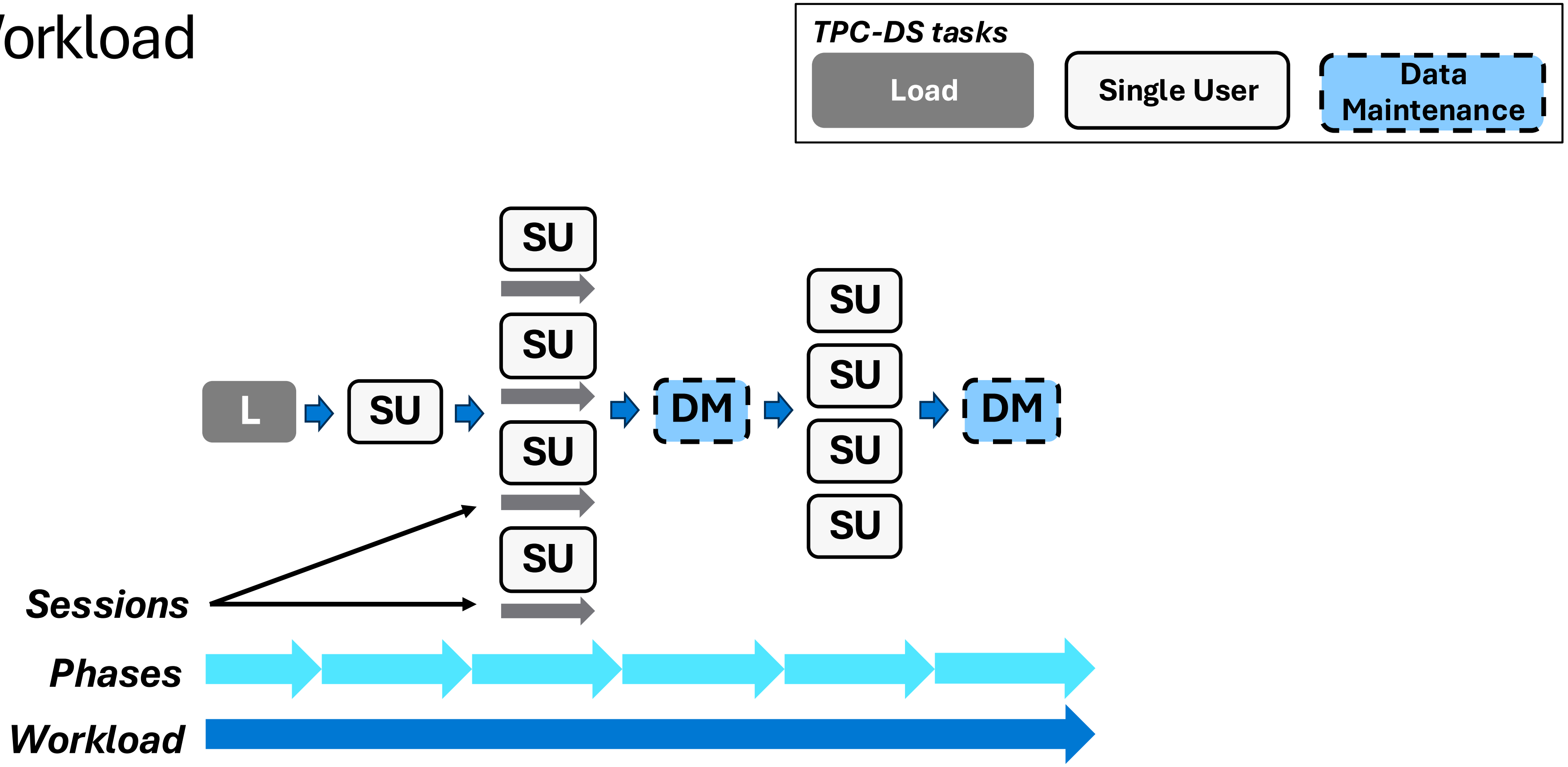
[CAG+24] J Camacho-Rodriguez, A Agrawal, A Gruenheid, A Gosalia, C Petculescu, J Aguilar-Saborit, A Floratou, C Curino, and R Ramakrishnan. LST-Bench: Benchmarking Log-Structured Tables in the Cloud. In SIGMOD, 2024.

Workload Representation

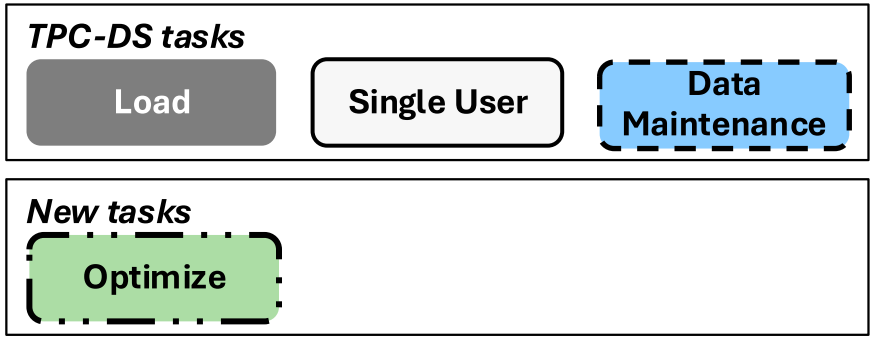
- Adaptable to both standard and custom workloads prevalent in practice
  - Driven by diverse and extensive internal user feedback
  - Facilitates the mapping of existing workloads, such as TPC-DS
  - Aligns with the concept of sessions in JDBC
  - Ensures ease of reusability

Extending Workloads Beyond TPC-DS

- ***W0:*** TPC-DS Workload



LST-Bench Workload Patterns

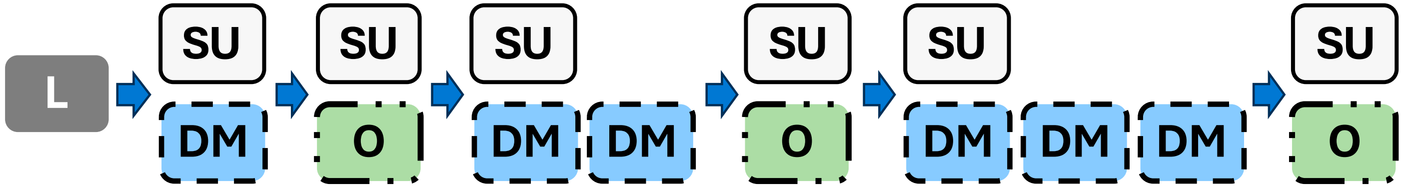


Gain insights into LST aspects overlooked by TPC-DS

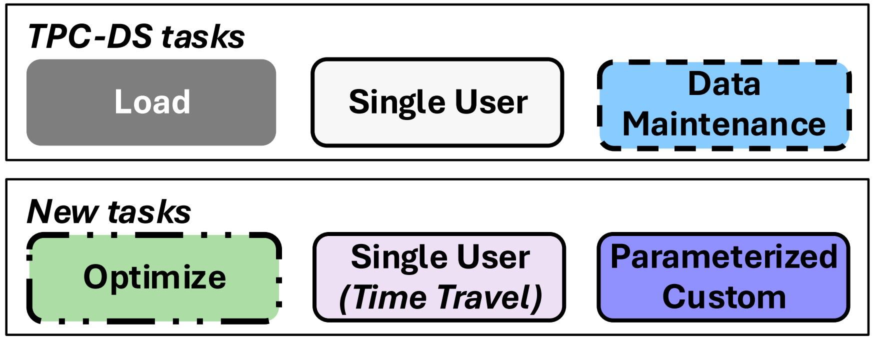
- ***W1:*** How are frequent data modifications handled over a long period of time?

- ***W2:*** How are multiple data modifications of varying sizes handled within a regularly optimized table?

- ***W3:*** How are simultaneous reading and writing sessions handled, potentially across multiple compute clusters?

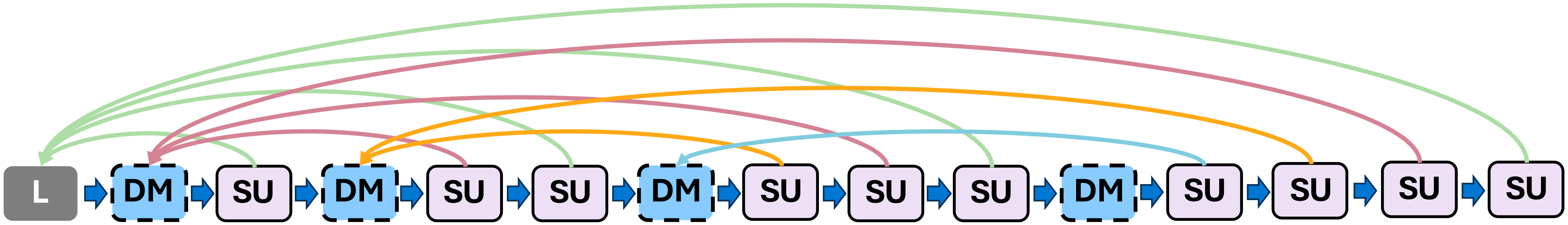


LST-Bench Workload Patterns

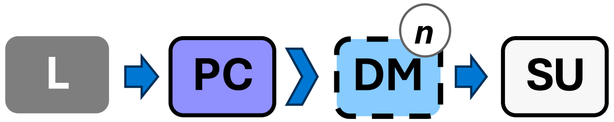


Gain insights into LST aspects overlooked by TPC-DS

- ***W4:*** How is data querying across different points in time handled?



- ***W5:*** How does updating or deleting the same data volume in different batch sizes influence read query performance?



n tasks dynamically generated by parameterized custom task

LST-Bench Metrics

Traditional Metrics

- Performance: *Latency, Throughput*
- Cloud Storage Efficiency: *Capacity Utilization, API Call Count, Total IO*
- Compute Engine Efficiency: *CPU Utilization, Memory Utilization, Disk Utilization*

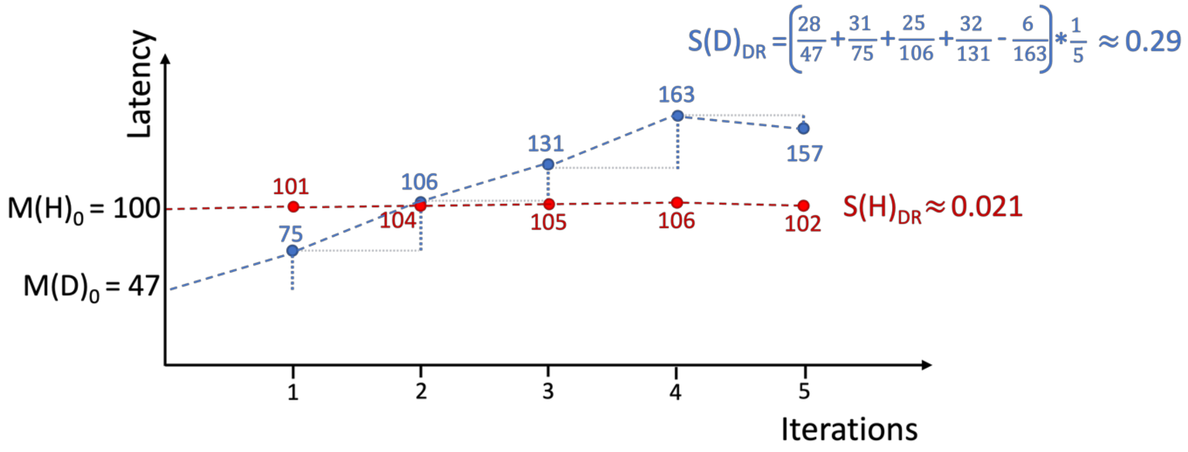
Other Metrics

- Stability - System's ability to sustain consistent performance and efficiency with minimal degradation: *Degradation Rate*

$$S_{DR} = \frac{1}{n}\sum_{i=1}^{n} \frac{M_i - M_{i-1}}{M_{i-1}}$$

where

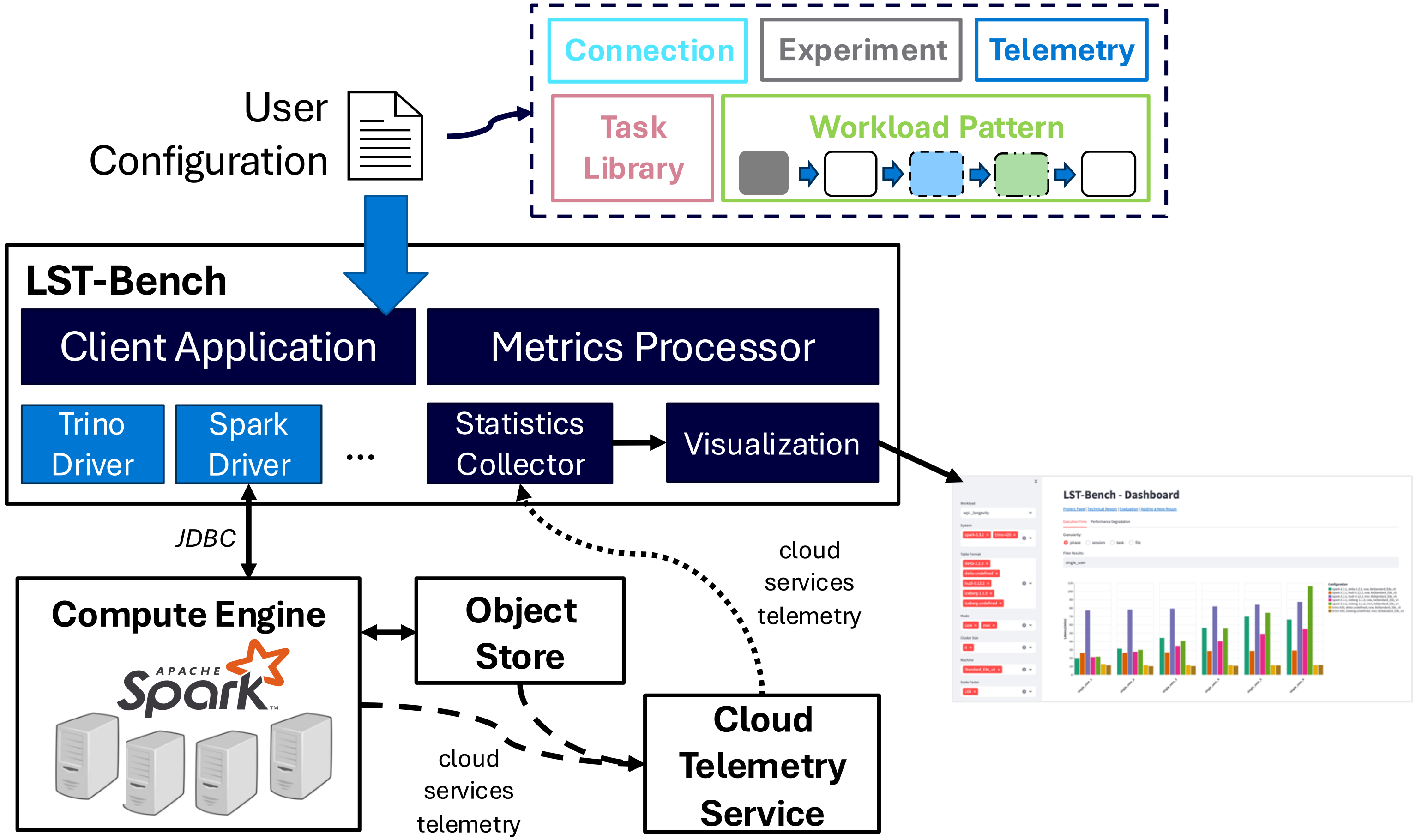
- M_i is metric value of the i^{th} iteration of a workload phase,
- n is the number of iteration of the phase, and
- S_{DR} is the degradation rate.



1. Data Lake-Centric Architectures and LSTs
2. Motivation
3. Tool and Benchmark Design
4. Implementation
5. Results
6. Status, Road Ahead, and Key Takeaways

LST-Bench Implementation

- Java Client Application
  - Customizable / extensible via config files
  - Connects to engine via JDBC or Spark session client
- Python Metrics Processor
  - Visualization via notebook or Streamlit web app
- Open-source available under Apache License 2.0: https://github.com/microsoft/lst-bench



LST-Bench Configuration, Libraries, and Workloads

- Configuration-driven approach
  - YAML/JSON file support
  - Input validation against JSON Schema

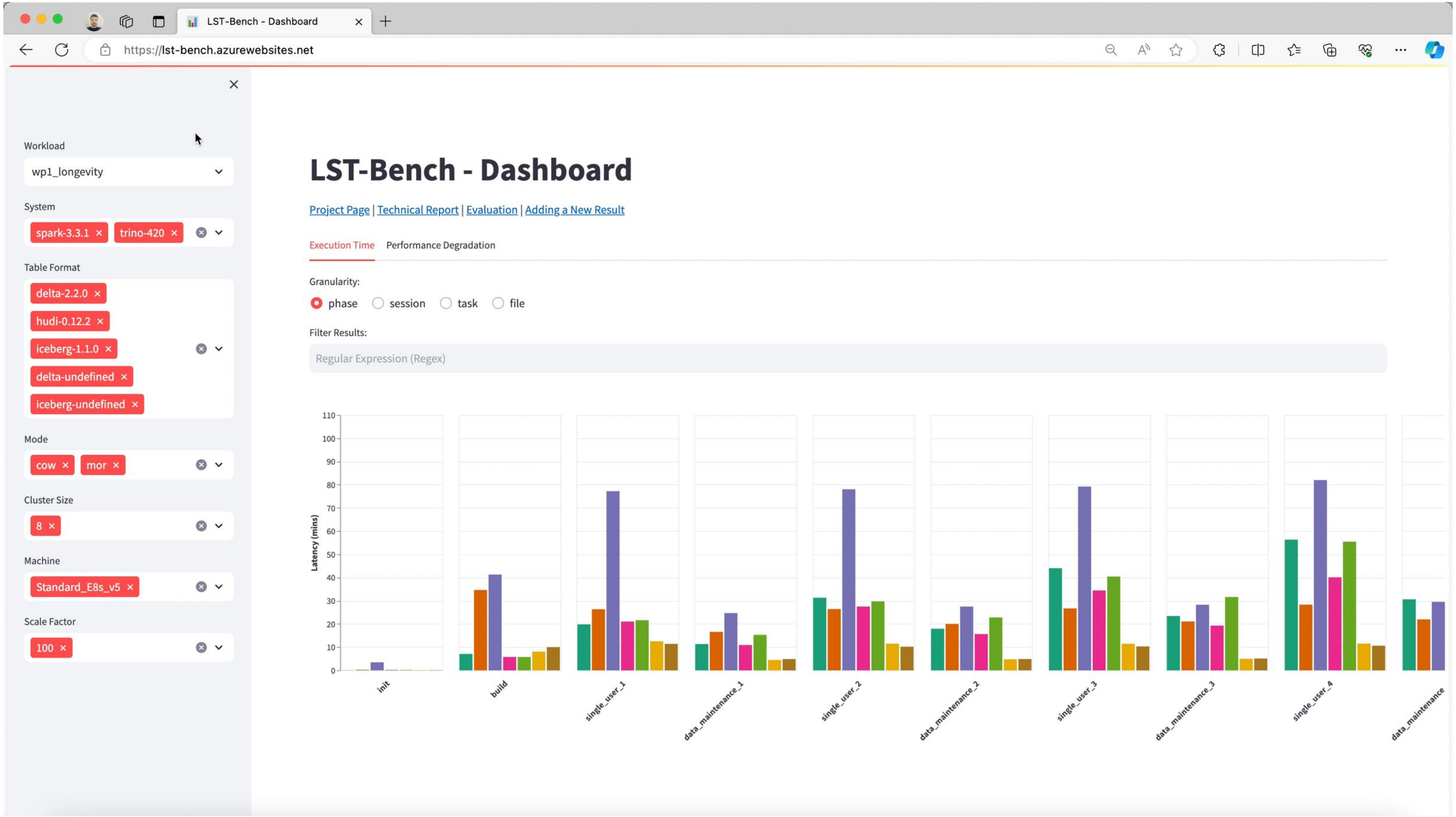
- Workload options
  - **Self-contained:** All tasks, sessions, and phases defined within the workload file
  - **Library-based:** Define tasks, sessions, and phases in a library and reference these entities from the workload definition

```yaml
- id: throughput_simple_phase
  sessions:
  - tasks:
    - template_id: single_user_simple
      permute_order: true
    target_endpoint: 0
  - tasks:
    - templ
      permu
    target_
  - tasks:
    - templ
      permu
    target_
  - tasks:
    - templ
      permu
    target_
```

```yaml
id: my_first_workload
phases:
- id: warm_up
  sessions:
  - tasks:
    - template_id: single_user_simple
    target_endpoint: 0
  - tasks:
    - template_id: single_user_simple
    target_endpoint: 1
- id: throughput_simple
  template_id: throughput_simple_phase
```

workload.yaml

LST-Bench Web UI



1. Data Lake-Centric Architectures and LSTs
2. Motivation
3. Tool and Benchmark Design
4. Implementation
5. Results
6. Status, Road Ahead, and Key Takeaways

Evaluation

- Azure VMSS cluster with 1 head and 16 worker nodes
  - Each node with 8 virtual cores and 64GB RAM
- Data stored in Azure Data Lake Storage Gen2 (ADLS)
  - TPC-DS SF1000
- Azure Monitor to collect telemetry and Logs Analytics to execute queries against it

- <u>No special tuning</u> for any of the engines, LSTs we evaluated:
  - Apache Spark 3.3.1: Delta Lake v2.2.0, Apache Hudi v0.12.2, Apache Iceberg v1.1.0
  - Trino 420: Delta Lake, Apache Iceberg

- <u>*Important remarks*</u>
  - Results subject to change and improvements due to further tuning and future developments
  - Insights drawn for these engines may not apply to the LST on different engines

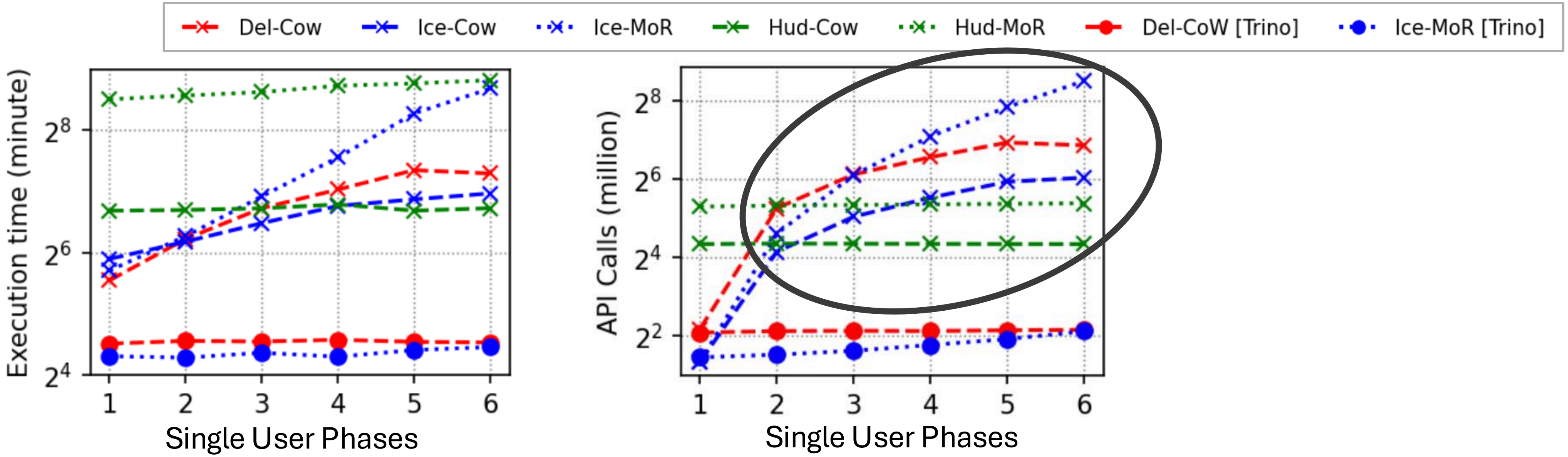
Evaluation – *W1*

Performance (Spark): Significant slowdown observed across iterations (up to 6.8x) for Iceberg-MoR. Nearly all formats show a decrease in performance.

Evaluation – *W1*

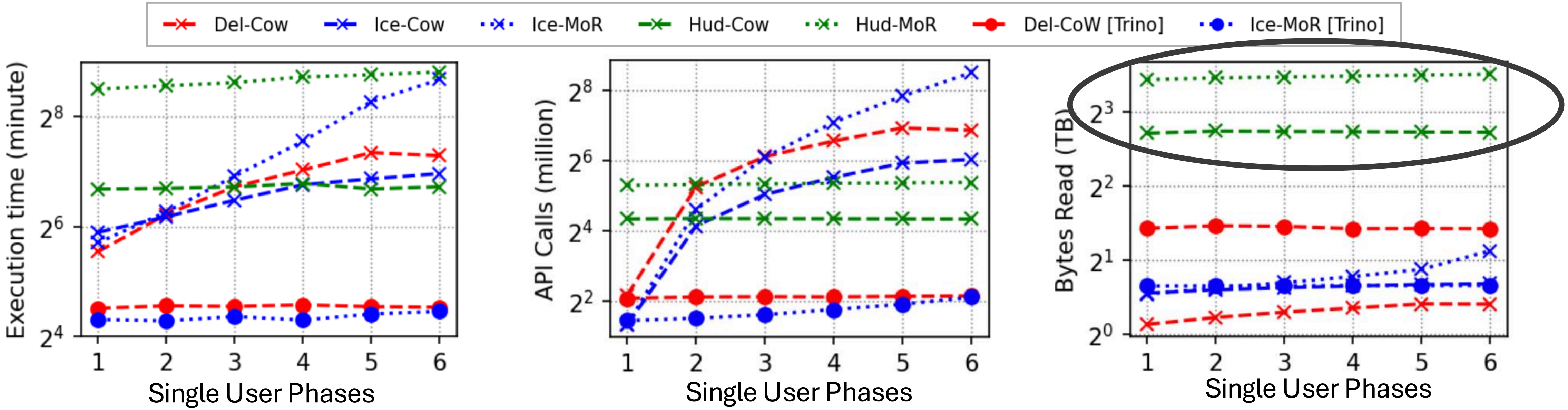
Performance (Trino compared to Spark): Nearly double the speed for both Delta and Iceberg tables after load. Higher stability.

Evaluation – *W1*



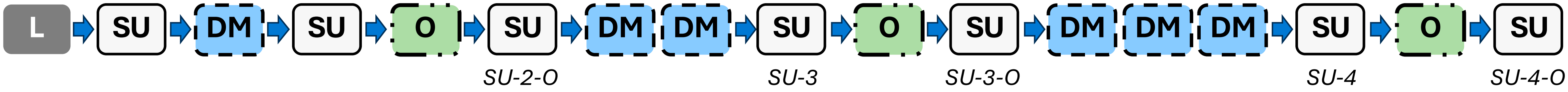
Storage Layer Calls: Increased API calls due to a higher number of (small) files that need to be read.

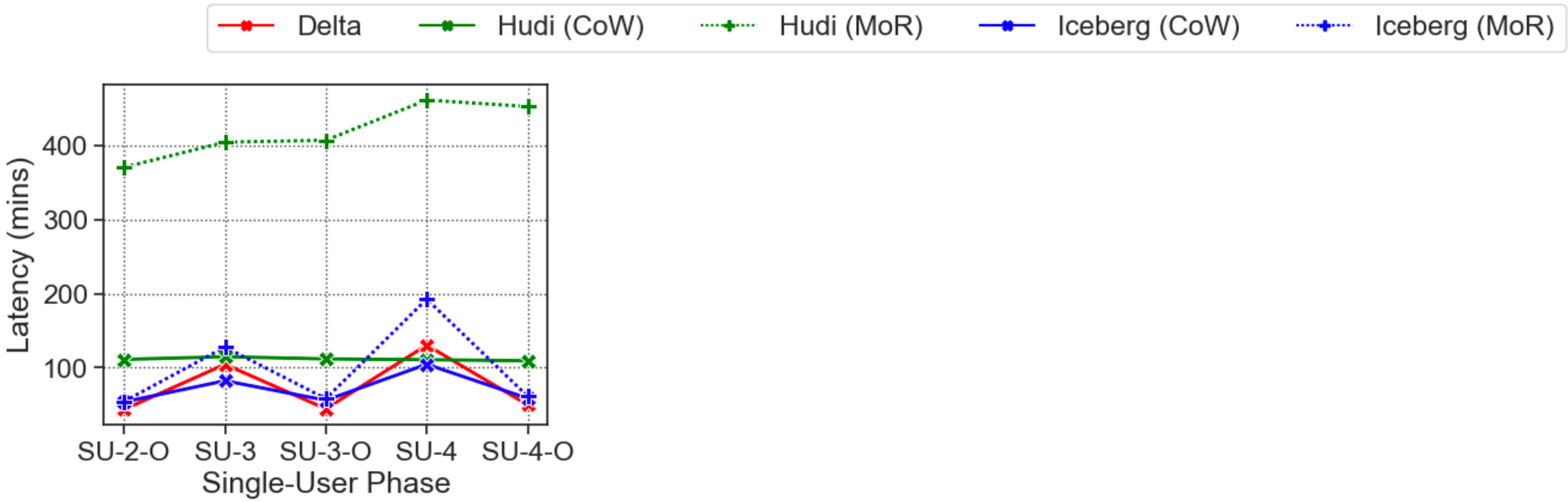
Evaluation – *W1*



I/O Volume: Comparatively high I/O volume for Hudi due to its default configuration, resulting in a higher number of small files.

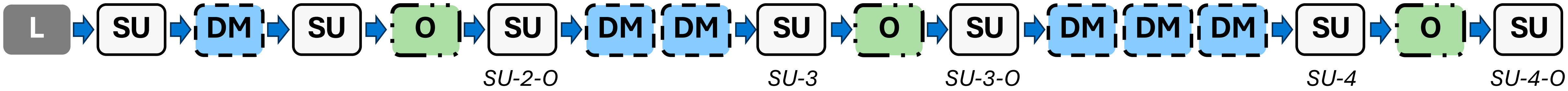
Evaluation – ***W2***

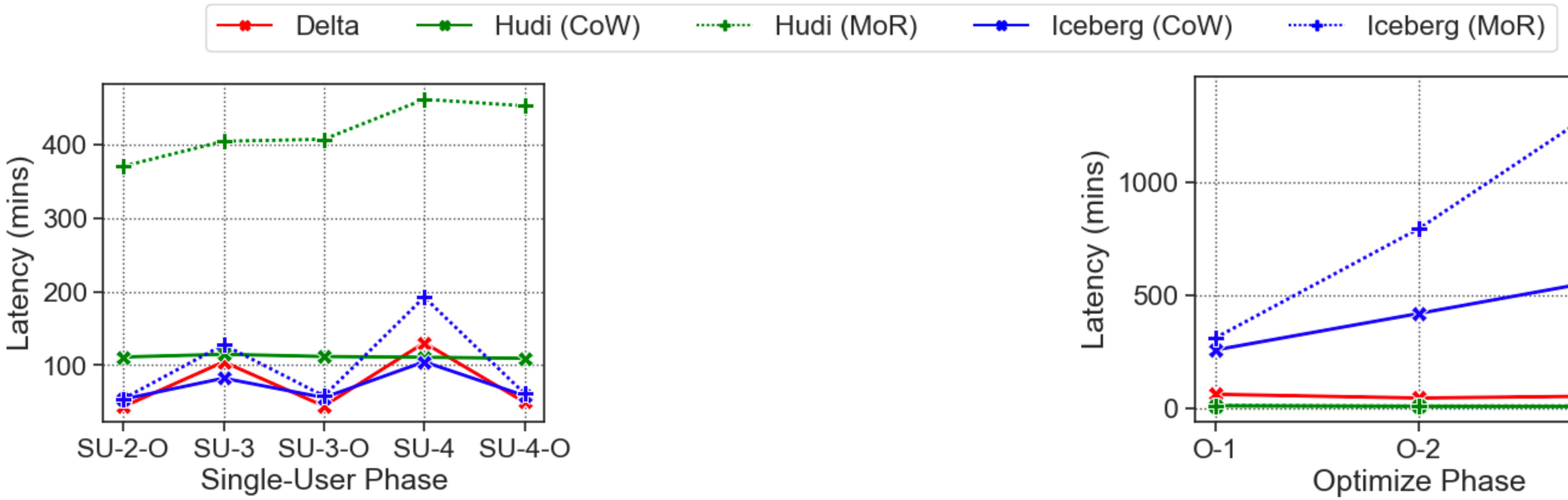




1. Table maintenance has a big impact on Delta and Iceberg performance stability (*zig-zag pattern*).
2. Hudi maintains stable performance without user-triggered maintenance by doing work upfront.

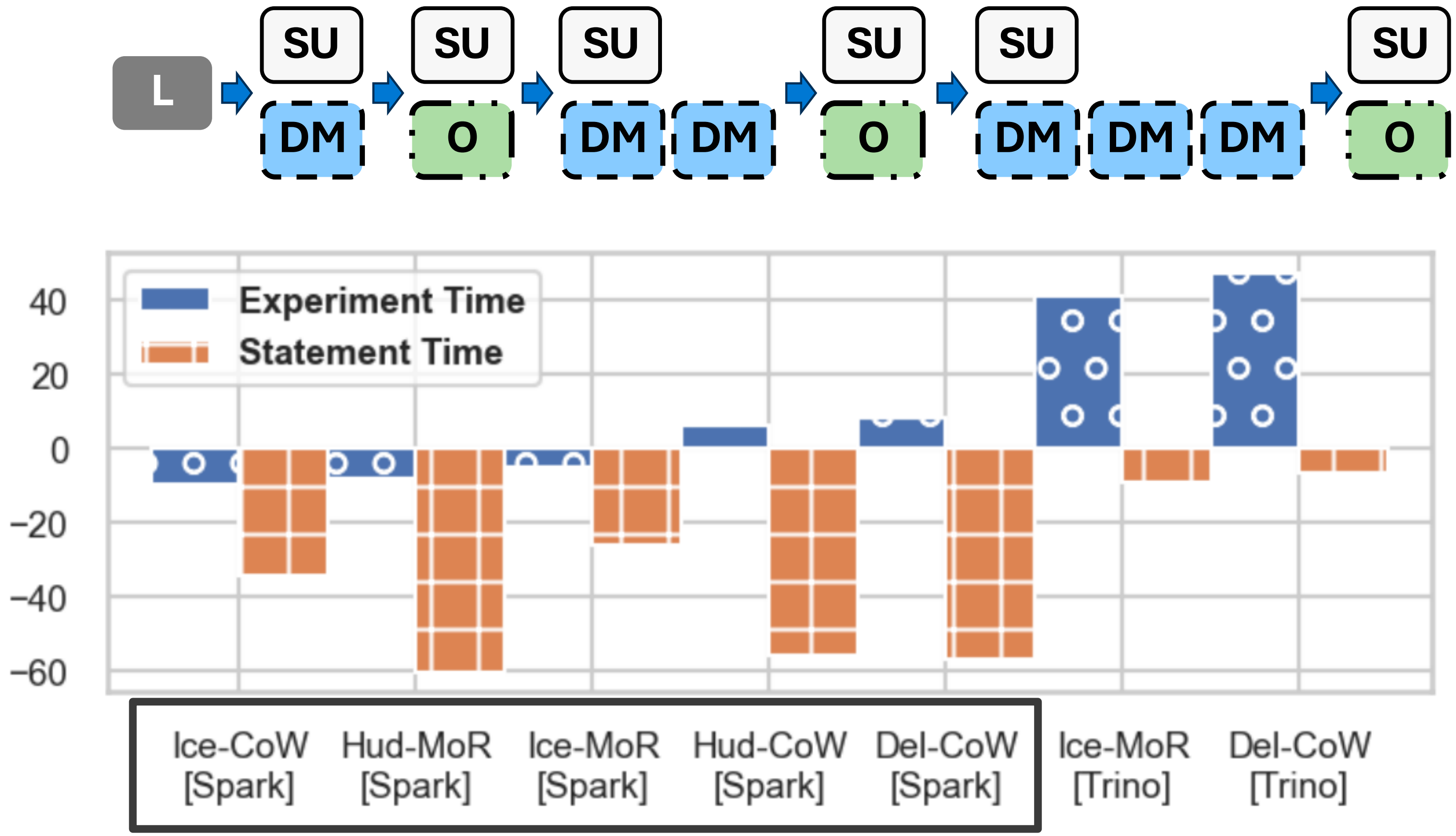
Evaluation – *W2*





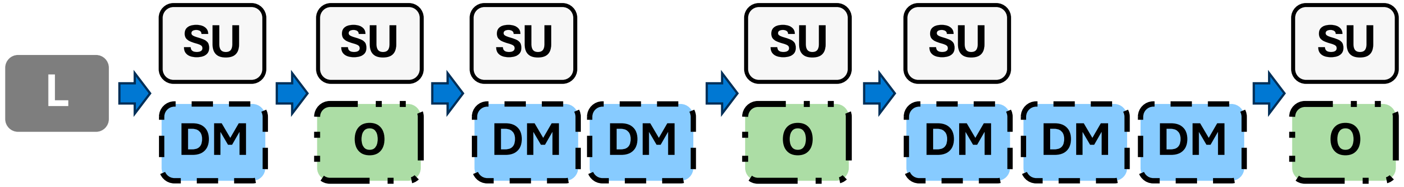
Iceberg's default file grouping for compaction *significantly increases compaction time* (potentially *minimizes read query disruptions*) ➡ Tuning LSTs involves trade-offs based on user goals

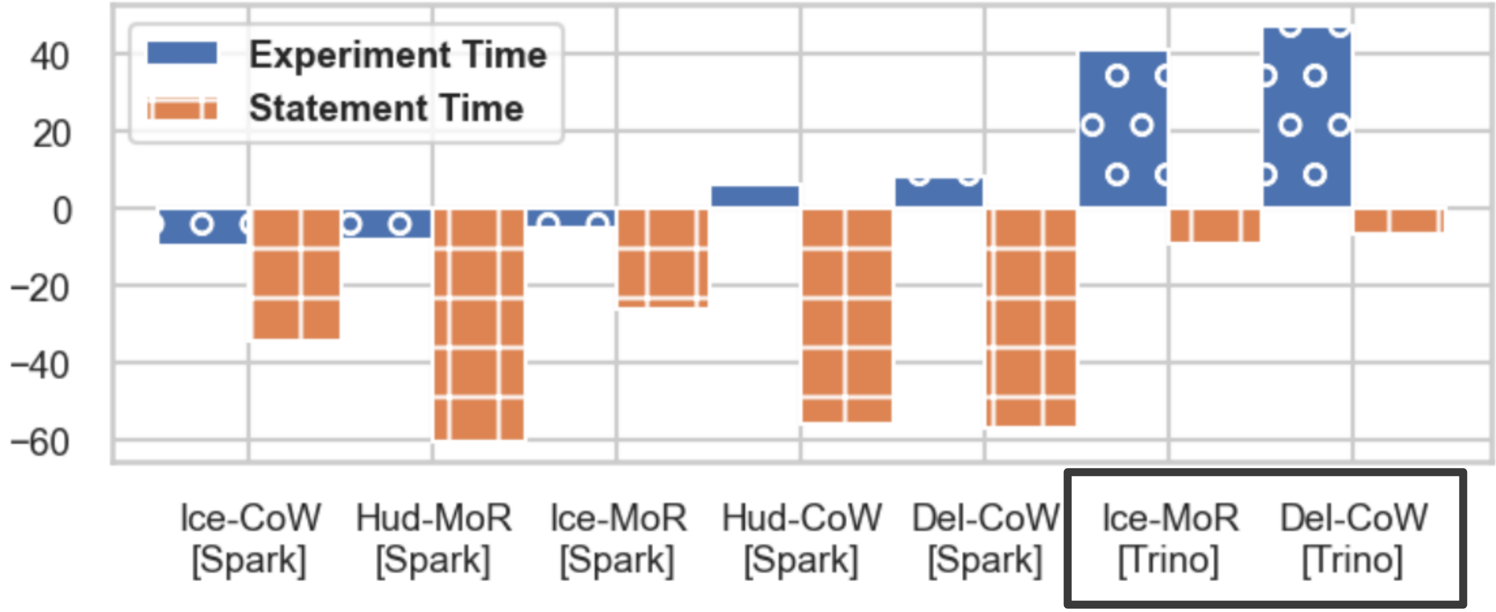
Evaluation – ***W2 vs W3***



Spark: Concurrent session execution does not lead to significant performance improvements due to resource contention.

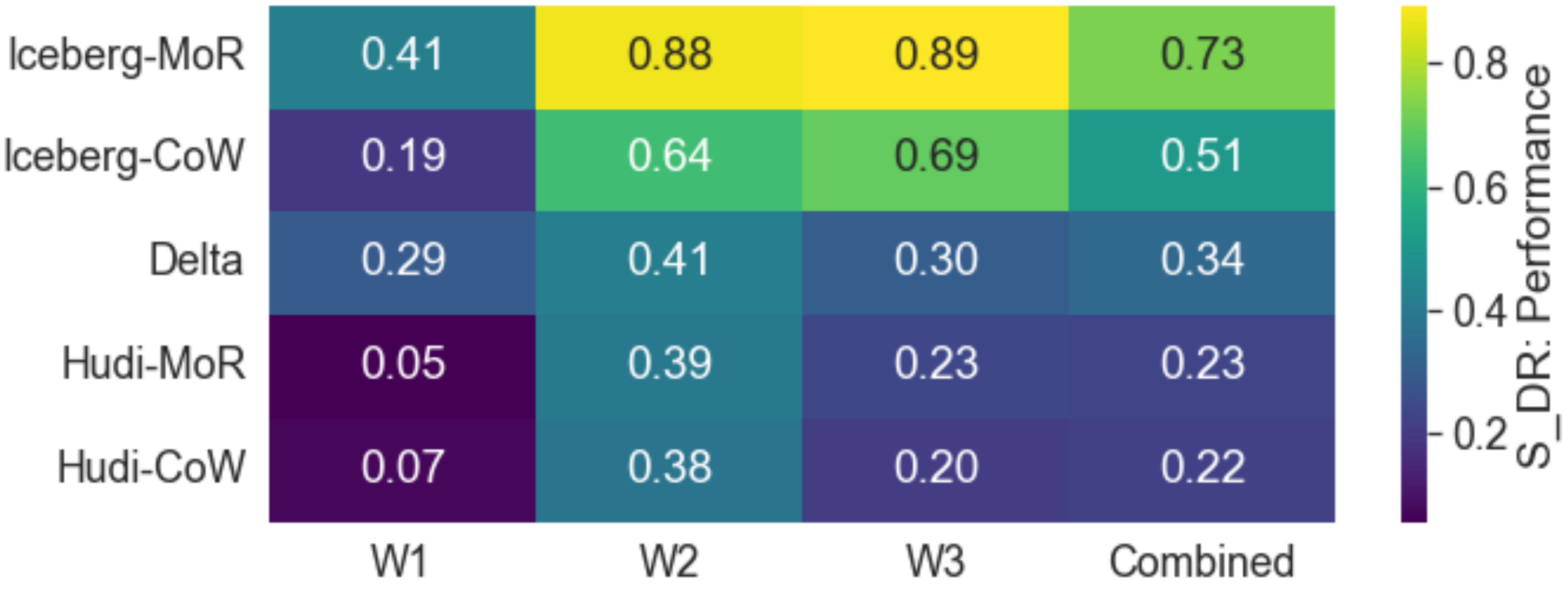
Evaluation – *W2 vs W3*



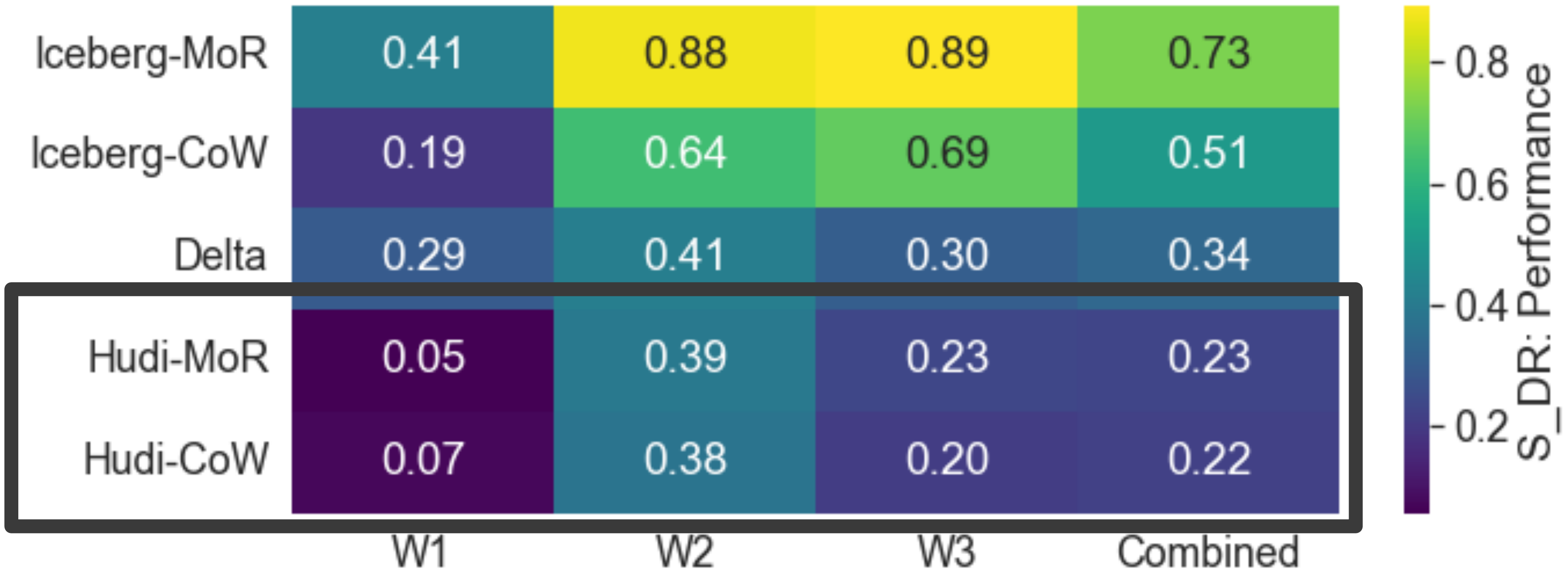


Trino: Efficient utilization of cluster resources results in significant end-to-end experiment runtime gains, despite minor slowdowns in individual statements.

Stability Evaluation



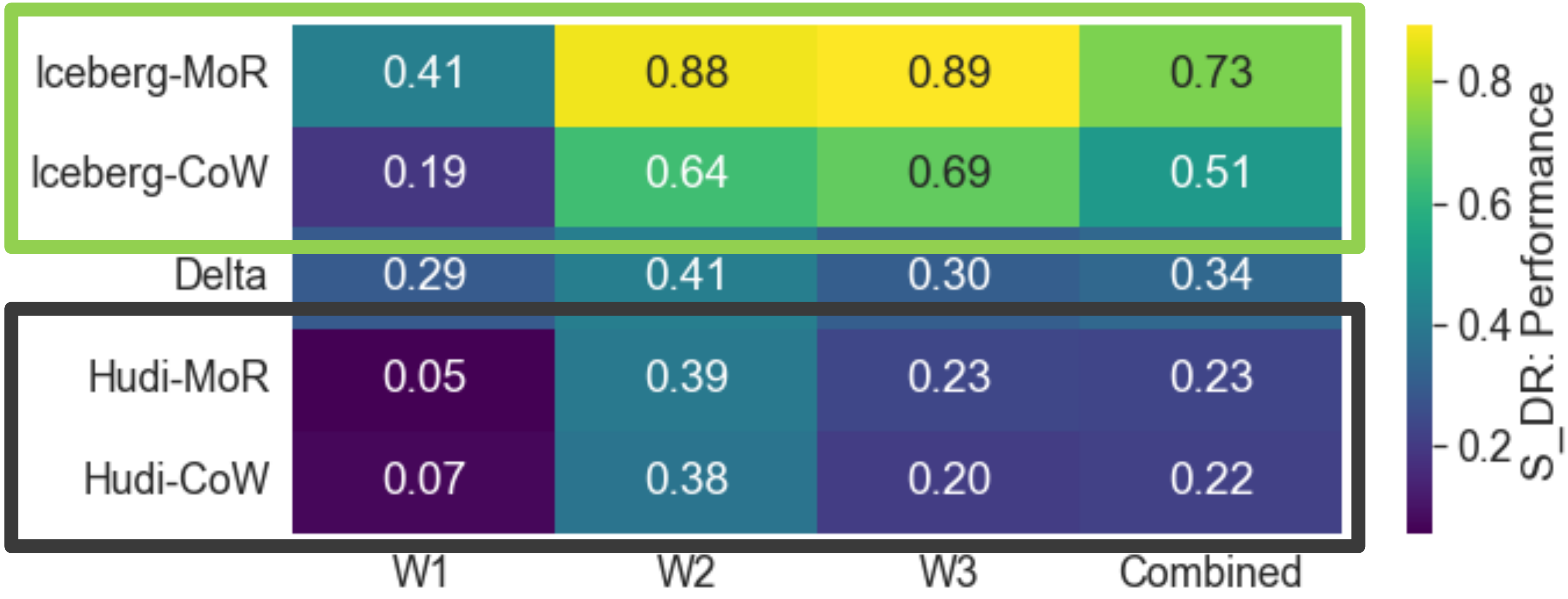
Stability Evaluation



Hudi shows highest stability

Stability Evaluation

Iceberg shows lowest stability



Hudi shows highest stability

1. Data Lake-Centric Architectures and LSTs
2. Motivation
3. Tool and Benchmark Design
4. Implementation
5. Results
6. **Status, Road Ahead, and Key Takeaways**

Status and Road Ahead

- LST-Bench @ Microsoft
  - Integrated into Microsoft Fabric Warehouse's testing workflow
  - Foundational tool for various ongoing initiatives:
    - Automatic tuning and maintenance policies for LSTs
    - Performance evaluation of LSTs converted using Apache XTable (Incubating)

- Open-source LST-Bench
  - Support for other engines, platforms, cloud providers (Apache Flink, Snowflake, AWS)
  - New scenarios: Data cleaning, CDC with transactional consistency guarantees
  - Integration with OpenTelemetry

  - Others? Contributions welcome!

Key Takeaways

- Evolving Benchmarks:
  - Traditional OLAP benchmarks like TPC-DS are not representative of modern analytic data lake workloads, e.g., lack of trickle updates
- Flexible and Extendable Tools:
  - **Modular, flexible benchmarking tools** are essential for evaluating new engines, datasets, and scenarios in the ever-expanding landscape of data lake architectures
- Comprehensive Metrics and Observability:
  - One representative metric can simplify decision-making, but **enhanced metrics and 360-degree observability** are crucial for understanding system characteristics
  - Achieving this level of observability is **challenging**, especially across multiple engines and cloud environments

Microsoft

Microsoft Azure Data
Gray Systems Lab

Acknowledgements

Ashvin Agrawal

Anja Gruenheid

Jose Medrano

Emma Rose Wirshing

Ashit Gosalia

Cristian Petculescu

Josep Aguilar-Saborit

Avrilia Floratou

Carlo Curino

Raghu Ramakrishnan

LST-Bench paper @ ACM SIGMOD 2024

Open-source available (Apache License 2.0)

Thank you! Questions?