Georg-August-Universität Göttingen

Institute of Computer Science

Seminar Report

Data Lake vs Data Lakehouse: A Comprehensive Comparison for HPC Workloads

on Heterogeneous and Homogeneous Data

Erdni Mankirov

MatrNr: 19665724

Supervisor: Aasish Kumar Sharma

Georg-August-Universität Göttingen

Institute of Computer Science

September 30, 2025

Abstract

High-Performance Computing (HPC) teams should choose between storage architectures such that they must satisfy 3 conflicting parameters: low-latency object access, high aggregate throughput, and controlled metadata/operational overhead. This decision affects time-to-result ratio, working cost and productivity.

To help make that choice we perform a reproducible, empirical comparison between a Data Lake (S3/MinIO) and a Data Lakehouse (Spark + Delta/Parquet). We run controlled micro-benchmarks on representative scientific workloads — heterogeneous and homogeneous data — and measure five operational metrics: aggregate throughput (MB/s), latency percentiles (P50/P95/P99) for single-object and single-record access, metadata overhead (listing/commit), recovery time after database artificial turn off, and storage efficiency.

Results that we got show that Lakehouse perform higher aggregate throughput, while S3 clients show much lower per-client rates. Raw S3 GETs are low-latency while Spark single-row probes exhibit orchestration overhead. Delta metadata incurs measurable commit delays (seconds in some runs) but not significant storage overhead at tested scales.

Keywords: Data Lake, Lakehouse, Delta, Spark, MinIO, HPC, throughput, latency, metadata, benchmark

Contents

List of Figures

List of Tables

1 Introduction

1.1 Topic

This experiment performs an empirical comparison between two modern data management architectural paradigms — *Data Lake* and *Data Lakehouse* , in the context of High Performance Computing (HPC). We show how each paradigm behaves on *heterogeneous* datasets. The results of the experiment will be useful for performance characteristics and operational trade-offs relevant to HPC centers.

1.2 Problem statement

HPC environments produce and use huge amount of different data: numerical data, structured tabular data such as CSV, semi-structured data, logs and etc. Choosing an appropriate data architecture influences on performance, speed,quality of the work and therefore on costs and resources. While Data Lake (object storage with schema-on-read) provides us flexibility and low storage cost, Lakehouse architectures has stronger transactional guarantees (ACID), integration with analytics engines, making it simpler for analytics. However, evidence at this moment is not as clear as we want because for instance benchmarks developed for business analytics and cloud storages do not necessarily transfer to HPC, where file sizes, data types and metadata pressure could have significant differences.

So we need a certain, reproducible experiment to answer: *under representative HPC conditions, when and why Lakehouse would be a better alternative for traditional Data Lake (and vice versa when Data Lake would still have more advantages than Lakehouse)?* As a part of the experiment we also should calculate how each approach performs on many small objects and metadata-heavy workload with different types of data.

1.3 Motivation

The motivation for this work:

1. **Determent operational clarity for further HPC experiments.** Scientist and analysts who works on HPC clusters with huge amount of different data should have optimal decision: should we use more complex Lakehouse (ACID, metadata services) in HPC production environments instead of more simple Datalake.
2. **Performance in reality** HPC workloads differ. We want measurements based on realistic HPC patterns and data types.
3. **Guidance for a scientist.** Researchers, data engineers and data scientist should understand how data architecture choice affects experiment latency and storage costs knowing real calculated numbers.

1.4 Objectives

The purpose of this study is to empirically characterise and compare Data Lake and Data Lakehouse architectures under experimental close to real HPC workloads. We want to:

- **Measure performance:** calculate throughput (MB/s) and percentile latencies (P50, P95) for representative read/write on both heterogeneous and homogeneous datasets. These metrics were chosen because they are important for HPC workloads: *throughput* measures the system's capacity to move a large volumes of data, while *latency percentiles* (P50, P95) capture responsiveness for a small operations. Together they provide a balanced objective view of performance for both bulk scans and latency-sensitive accesses.

- **Measure metadata impact:** calculate metadata overheads (schema registration, partition listing and etc) when the system works on many small objects and on large bulk files.
- **Recovery performance:** evaluate recovery time after simulated node crash, for example, and compare architectures in terms of restart time of system.
- **Measure storage efficiency:** compare effective storage ratio (useful data vs total storage including metadata and replicas).

1.5 Research questions and hypotheses

So now knowing the objective we can formulate concrete research questions :

- **RQ1:** How do Data Lake and Lakehouse performs on raw throughput for bulk writes and reads of big homogeneous and heterogeneous synthetic data?
- **RQ2:** How do the Data Lake and Lakehouse behave under metadata-heavy workloads/many small files typical for logs and small result files?
- **RQ3:** What is the latency profile (P50/P95) for OLAP-style queries (aggregations, joins) performed on same datasets in both architectures?
- **RQ4:** How do metadata services in Lakehouse architectures affect ingestion latency and operational complexity?
- **RQ5:** What are the restart time after simulated failure and data storage efficiency for both architectures?

We start with the working hypotheses that (H1) Lakehouse transactionality will improve speed for analytic workloads and has more advantages than Data Lake but (H2) will impose additional metadata overhead that may negatively affect throughput and small-file workloads and could not be a good alternative for a particular task comparing with Data lake.

1.6 Scope

This study focuses on reproducible micro-benchmarks for HPC.

- **Data volumes:** because of limitations on the cluster in the experiment we will use relatively small synthetic amount of data (7.5–10 GiB).
- **Data types:** for homogeneous data binary files was used, for heterogeneous csv, ppm and wav were used.
- **Metrics:** throughput (MB/s), latency percentiles (P50, P95), metadata overhead, recovery time and storage efficiency.
- **Environment:** experiments run on a GWDG cluster using the containerised software (MinIO, Spark, Trino, Delta/Iceberg, Slurm, Prometheus/Grafana). Details in the further sections .

1.7 Contributions

This paper contributes:

- A reproducible benchmark that could be applied to Data Lake and Lakehouse architectures working on HPC environments;
- Real measurements and metrics both suitable for heterogeneous and homogeneous data in both architectures;

2 Background

2.1 What is a Data Lake and a Data Lakehouse

Data Lake. A Data Lake is an object- or file-based centralized repository for different types of raw data: structured,semi-structured, unstructured that uses schema-on-read (data interpretation happens at query time). This contributes to better flexibility for heterogeneous sources, (comparing with DataWarehouse for instance), cheap horizontal storage scaling, and fast ingestion paths for raw artifacts. [1]

Data Lakehouse. A Data Lakehouse is a hybrid architecture combining Data Lake flexibility and Data Warehouse reliability. Provides ACID transactions on object storage via Delta Lake or Iceberg. Suitable for both BI and data science workflows. The Lakehouse preserves object-store economics while providing ACID semantics. This paradigm can perform SQL analytics over data stored on object storage.

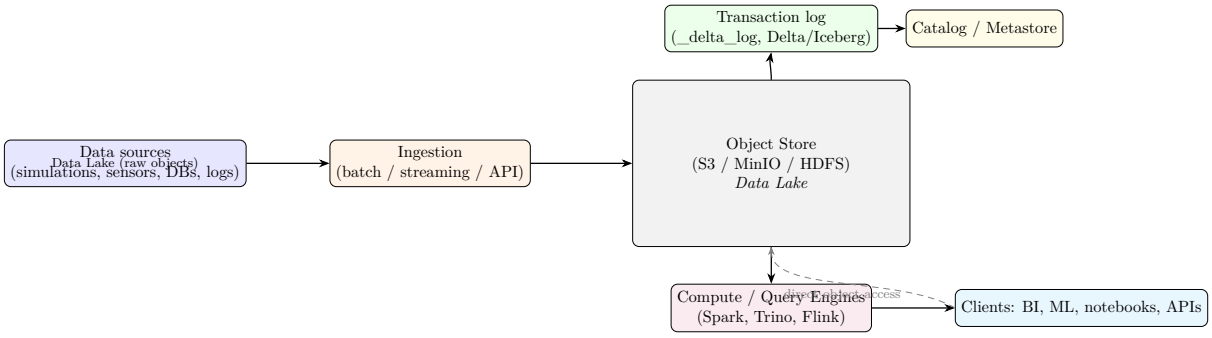


Figure 1: Simplified architecture: a Data Lake (object store) with a Lakehouse overlay (transaction log + catalog + query engines).

2.2 Data types and heterogeneity

Scientific HPC pipelines can produce a different types of data — each with different approach and storage implications:

- **Structured tabular:** CSV. Compact, columnar format are good for analytics.
- **Semi-structured:** JSON, XML. Flexible, nested data / recordes must be preprocessed before analytics.
- **Unstructured media:** Images (JPEG), audio (WAV), video (MP4). Are very hard for analytics queries to proceed.
- **Scientific arrays :** HDF5, NetCDF. Large record that optimized for sequential and parallel reads/writes.
- **Small files:** Logs, checkpoints, per-task outputs (tens–hundreds of KiB each). These create metadata realated to workloads that stress object-store metadata operations.

Different types of data affect throughput, latency, and metadata pressure. For example, many small files stress metadata and can severely decrease transaction latency.

2.3 Good and bad: Data intake and processing workflows

Data Lake — intake & processing **Intake:** simple PUT-based operation. **Processing:** schema-on-read, ETL performed at query time.

Pros:

- Fast, flexible ingestion for arbitrary formats.
- Low-cost, horizontally scalable storage.
- Low single-object latency on object access.

Cons:

- Not suitable for transactionality.
- Many small files could make metadata overhead .
- Analytics often requires additional preprocessing steps before efficient querying.[2]

Data Lakehouse — intake & processing **Intake:** ingestion with engines (Spark) that write files and update a transaction log. **Processing:** SQL analytics with support of ACID principles, cataloged tables.

Pros:

- ACID is preserved.
- High aggregate throughput for batch analytics.
- Simple SQL access and integration with ML/BI.

Cons:

- Metadata and transactional operations perfrom additional latency (log writes) — could be ineffective for frequent small commits.
- Complex for operational performance. [3]

3 Literature review

- **"From Data Warehouse to Lakehouse: A Comparative Review"** — Harby & Zulkernine (IEEE BigData). This review is useful for our work because there were discussed the main architectural differences (schema-on-write vs schema-on-read, metadata role) and useful for us to measure throughput and metadata influence on latency values.
- **"Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics"** — Armbrust et al. (CIDR). In this paper the Lakehouse design was described: open file formats, table semantics and etc. It could help to heck our hypothesis that coalesced Lakehouse writes improve aggregate throughput.
- **"Data Lakehouse: A Survey and Experimental Study"** — (ScienceDirect / Harby et al.). This work shows similar study where directly compared Data Lake, Warehouse and Lakehouse systems and reports measured trade-offs in real workloads. It insipired us to calculate the quantitative metrics (throughput, latency, metadata overhead) and provide experimental techniques for our relatively small benchmarks.

Collectively these sources show mostly conceptuall differences between the different architectural styles but none of it has direct comparison of the declared metrics on the same experiment, so these papers are perfect sources of usefull conceptual information for performing a quntative experiment.

4 Research Methodology

This study follows a **comparative quantitative** methodology: we measure performance metrics on 2 close storage architectures comparing them using the sequence of experiments. The main goal is to find the objective numbers that show practical trade-offs for HPC workloads.

4.1 Research design

- **Experimental approach:** Generated synthetical benchmarks ran on GWDG cluster. Experiments are grouped by: *architecture* (Data Lake vs Lakehouse) and *data type* (heterogeneous vs homogeneous).
- **Workloads:** experimental workload data are used — heterogeneous (text, images, audio,7.5 GiB) and homogeneous datasets (binaries, 10 GiB).

4.2 Measured metrics

Main dependent variables (quantitative metrics) that are measured:

1. **Aggregate throughput (MB/s)** for bulk reads and writes .
2. **Latency percentiles:** P50, P95 and P99 for single-object and single-record SQL probes.
3. **Metadata overhead:** time to listing, metadata scanning and other operations.
4. **Recovery time:** time until the object store starts to work again after failure.
5. **Storage efficiency:** ratio of user data bytes to total bytes stored (user data + metadata).

4.3 Data collection and execution practice

- **Reproducible artifacts:** all software is containerised in an Apptainer image and scripts for performing data generation and experiment at all are uploading at the moment in the project repository. (Despite the overall weight of the project is huge enough I hope it will be managable to upload it with all logs and results)
- **Measurement sources:** throughput is measured from the first I/O activity until job is completed; latency samples are collected per operation; metadata-phase timings are extracted from job logs.

4.4 Analysis methods

- **Summary statistics:** ranges, means and percentile estimates (P50, P95, P99) are reported.
- **Baselines:** homogeneous dataset is used as a baseline.
- **Visualisation:** grouped bar charts are used to present results in the most understandable way.

4.5 Reproducibility and limitations

- **Reproducibility:** container definition, scripts and etc have been provided in the project repository so that the experiments can be reproduced by any person.
- **Limitations:** experiments are benchmarks with weight 7.5–10 GiB; Further extrapolation may be performed in the future.

Overall, this quantitative comparative methodology will give as objective values and gives useful guidance for comparing and choosing storage architectures in HPC environments.

5 Experimental Setup - Apptainer Configuration and Dataset Generation

This chapter documents the Apptainer container configuration used to build the experimental software stack and provides a reproducible script to generate both heterogeneous and homogeneous datasets used by the benchmarks and explanation of technical aspects of the performed architectures.

5.1 Apptainer container: goals and workflow

The experimental software stack is packed as a single Apptainer image for reproducibility purposes. The main goals:

- include the runtime and server binaries used in experiments (MinIO, Spark, Trino, Iceberg, Prometheus/Grafana),
- keep the build deterministic.

Typical workflow:

1. Prepare/Download `Container.def`.
2. Build the image :
   `sudo apptainer build image.sif Container.def`
   or : `apptainer build --fakeroot image.sif Container.def`.
3. Validate the image
4. Run experiments.

5.2 Dataset generation: goals and design

For the experiments we require:

- a **heterogeneous** dataset of **7.5 GiB** total, composed of 3 categories: `text (CSV)` $\approx$ 2.5 GiB, `images (PPM)` $\approx$ 2.5 GiB, `audio (WAV)` $\approx$ 2.5 GiB.
- a **homogeneous** dataset of **10 GiB** total composed of many small binary files.

The 2 data generation scripts are implemented in Python so it requires only a recent Python3 version. It:

- creates the directory tree,
- creates files for each category until the written size is reached,
- divied files into subdirectories so there will be no too many entries in a single folder,
- write progress messages for monitoring.

5.3 Dataset generator script

Place the following scripts as `scripts/generate_datasets.py` and `scripts/gen_data.py` and run the first one with the command below. (Actually the `scripts/gen_data.py` could be easily replaced by the second one, it generates only binaries while the heterogeneous script creates any requested kind of data, so if you put in execution command only one type - it will generate only homogeneous data)

5.4 Running the generator and verification

Make the script executable:

```
chmod +x scripts/generate_datasets.py
```

Run with explicit sizes:

```
python3 scripts/gen_data.py --outdir data --images 2.5 --text 2.5 --video 0 --audio 2.5 --bi
```

5.5 Notes, trade-offs and reproducibility

- The generator creates simple formats: CSV, PPM images and WAV audio. They do not require additional python packages unlike video, and can be easily processed.
- You can also use gen-data.py, the old script that was used to create a homogeneous binaries, it can be executed by simple calling.

5.6 Integration with the Apptainer image

Recommended steps for experiments

1. Download and build the Apptainer image.
2. Generate datasets. I recommend to generate on host as it is faster and simpler and avoids running CPU-heavy operation inside the container.
3. Bind the generated dataset directory into the container when running experiments.
4. Collect logs and results of the job.

This chapter has provided a brief description of Apptainer configuration and dataset generator for our experimental needs. In the next chapter we will discuss the results and the exact metric values used for comparison Data Lake and Data Lakehouse in HPC environment.

6 Results and Discussion

6.1 Heterogeneous datasets

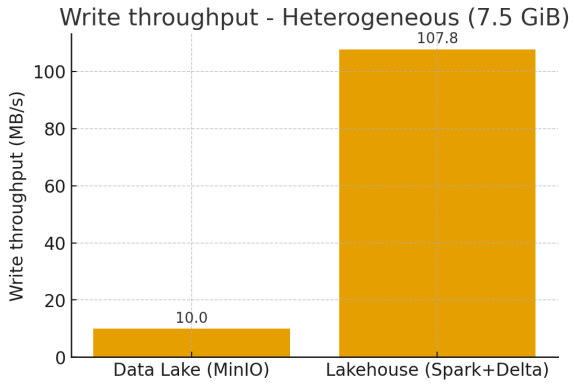
In this section we present and disucss the results of our experiment on the heterogeneous dataset. We compare the performance of a Data Lake (S3/MinIO) against a Data Lakehouse setup (Spark writing to Delta). For each metric we provide representative value obtained after the successful runs with a short interpretation. Let's take a look at the table with results for heterogeneous dataset 1

Table 1: Summary table for the heterogeneous data

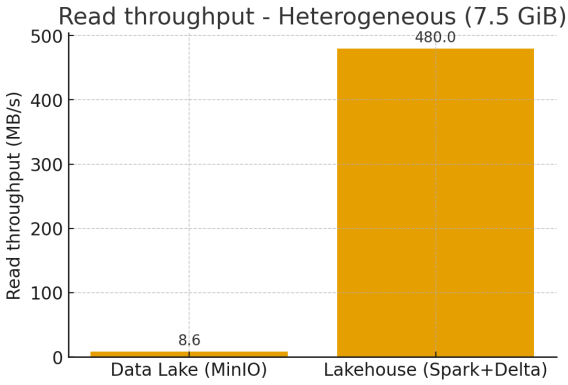
Metric / Scenario	**Data Lake (MinIO / S3 native)**	**Data Lakehouse (Spark → Delta / Parquet)**
Bulk upload	7.55 – 12.77 MB/s	79.8 – 111 MB/s
Bulk read aggregated throughput	5.515 – 10.18 MB/s	≈ 200 – 480 MB/s
Single-object PUT latency (P50 / P95)	P50 ≈ 13–14 ms; P95 ≈ 25–40 ms	client-level PUTs similar
Single-object GET latency (P50 / P95 / P99)	P50 ≈ 3.5 ms; P95 ≈ 3.9 ms; P99 ≈ 4.6–4.7 ms	P50 ≈ 250–275 ms; P95 ≈ 370–455 ms; P99 ≈ 600–1000+ ms
Metadata LIST (S3)	LIST mean ≈ 0.15 s	Delta metadata list aggregate ≈ 0.68–0.81 s
Metadata scan / bulk metadata op	—	metadata_bulk_seconds: 2.5–3.6 s, up to ∼19.6 s for pathological cases
Metadata commit / checkpoint	S3 PUT P50 ≈ 13–16 ms	metadata_write_seconds: observed up to ∼8.6 s
Recovery time (MinIO restart)	∼1.11–1.16 s	same results
Storage efficiency (user data / total)	≈ 0.99959 – 0.99965	≈ 0.99966 – 0.99969

Legend:
Data Lake (MinIO) — native S3 client results .
Lakehouse (Spark+Delta) — parallel Spark aggregate results that include task orchestration and multipart uploads; performance improves with coalescing of outputs.
Throughput bars show aggregate MB/s across tasks.
Latency: P50 and P95 shown.

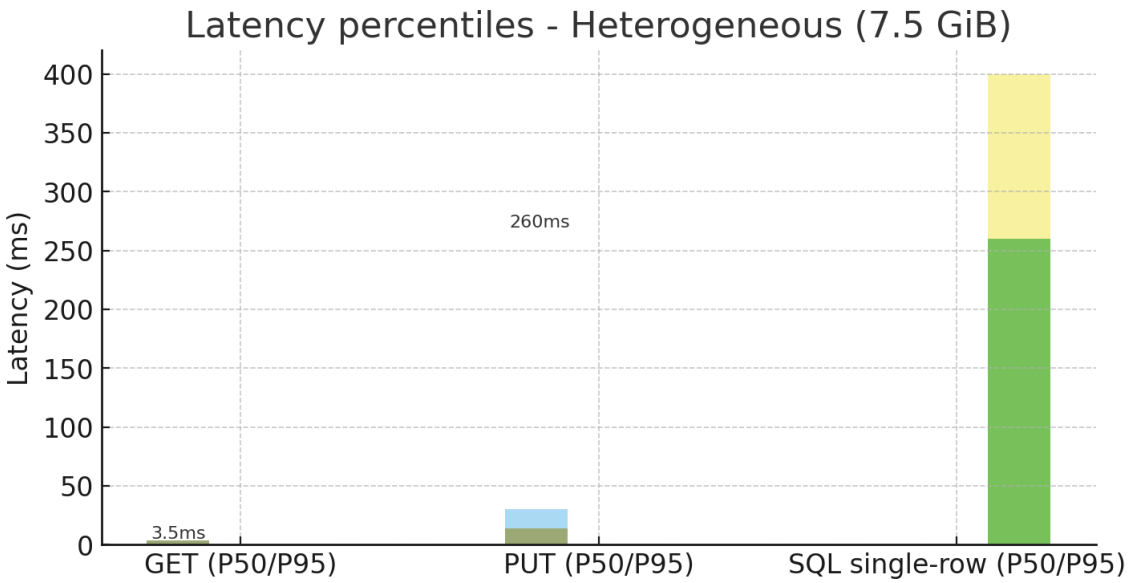
Figure 2 highlights that heterogeneity increases metadata pressure and reduces the effective advantage of Lakehouse when many small/varied objects are present, unless writes are coalesced.



(a) Write throughput (MB/s) — Heterogeneous (7.5 GiB).



(b) Read throughput (MB/s) — Heterogeneous (7.5 GiB).



(c) Latency percentiles (P50 / P95) — GET / PUT and SQL single-row probe (Spark).

Figure 2: Performance summary for the heterogeneous (7.5 GiB) dataset.

6.1.1 Throughput for heterogeneous data

Data Lake (S3 native). Native S3 benchmarking shows relatively modest per-client results in the recorded experiments: measured upload performance is around 7.5 – 12.8 MB/s. These numbers reflect the single-client performance using small numbers of parallel PUTs with the chosen client.

Data Lakehouse (Spark → Delta). When we write same data via Spark using well-sized Parquet files, the observed write throughput increases significantly - 80–110 MB/s in typical runs. For bulk reads, aggregated Spark read jobs achieved hundreds of MB/s in total completely outperforming Data Lake archictecture in this term.

Interpretation. We believe that the main factor is parallelism and task distribution: Spark splits writes/reads over many tasks, uses multipart uploads and parallel I/O paths, and therefore attains a much higher *aggregate* throughput than individual S3 client. So orchestration and parallel writes give Lakehouse configurations a strong advantage for bulk transfer.

6.1.2 Latency for heterogeneous data

S3 (MinIO) network latencies. S3 primitive operations are fast: measured single PUTs had P50 ≈ 13–14 ms and P95/P99 performs in tens of ms. GETs were faster: P50 ≈ 3.5 ms and P95 ≈ 3.9 ms. These numbers shows us the raw object API responsiveness on the local MinIO deployment.

Spark + Delta single-record latency. A single-record experiment loop executed through Spark/Delta has much higher latency: P50 values around 250–275 ms, P95 in the 370–455 ms range and P99 often exceeding 600 ms. The overhead is dominated by Spark job scheduling.

Interpretation. For low-latency single-record access, the raw S3 object API is faster. However, when queries are executed through big-data engines (Spark/Delta), fixed per-query orchestration costs dominate. It makes Spark/Delta less suitable for extremely latency-sensitive single-record lookups.

6.1.3 Metadata overhead

S3-level operations. S3 list operations show LIST latencies on the order of 150 ms per listing in this experiment. Individual metadata PUTs are P50 ≈ 13–16 ms.

Lakehouse metadata (Delta). A Delta transaction provides multiple listings and small metadata reads/writes. Measured values:

- **metadata_list_seconds**: typically ∼0.68–0.81 s.
- **metadata_bulk_seconds** :0.98 s (small), 2.5–3.6 s (normal), up to ∼19.6 s in pathological cases when _delta_log grows.
- **metadata_write_seconds**: can take several seconds in some runs (e.g. ∼8.6 s).

Interpretation. Because of Delta reliying on multiple S3 LISTs and small file operations to ensure correctness, the metadata cost is a extra term added to ingestion latency. For runs with many small and frequent updates, these metadata operations can dominate the end-to-end latency.

6.1.4 Recovery time and availability

MinIO restarts. Measured MinIO restart times in the tests were short: 1.1–1.16 s.

Impact on Lakehouse. Because the Lakehouse stack depends on the object store for both data and metadata, short outages of the underlying S3 store can cause failed transactions. High-availability configuration of object storage is recommended if sub-second availability is required.

6.1.5 Storage efficiency

Measured storage efficiency of both architecture were in the runs was high: user data / total $\approx 0.99959 - 0.99969$, so metadata took roughly **0.0003–0.0004%** of storage in the observed cases. Then we can say that despite metadata imposes latency overheads, its contribution to raw storage performance is nearly 0.

6.1.6 Overall interpretation

- **Bulk throughput superiority of Lakehouse:** for large-scale parallel writes and reads, which is very useful for HPC analytics, Spark + Delta achieves much greater aggregate throughput results than single S3 client. This makes Lakehouse superior when throughput and parallel analytics are required for heterogeneous data.
- **Metadata, small transaction penalty:** Lakehouse transactionality comes at a cost because additional metadata operations (multiple LISTs, commit/checkpoint) adding seconds to ingestion. For jobs with many small files or frequent commits, a Data Lake approach may actually provide lower latency.
- **Recovery and availability:** short MinIO restarts are fast but can have more affect on the Lakehouse stack.
- **Storage cost:** metadata size was extremely small in our experiments therefore the choice between Lake and Lakehouse does not depend on storage overhead.

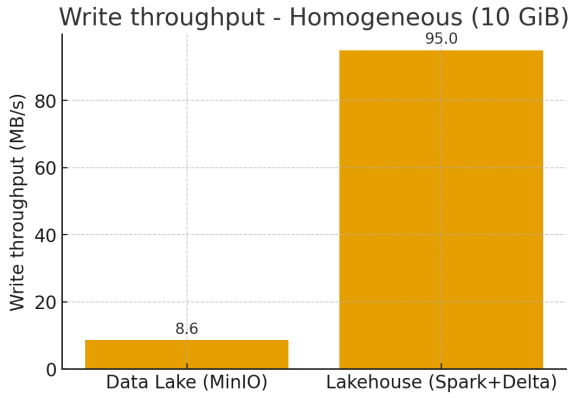
6.2 Homogeneous datasets

This section reports and interprets measurements collected on the homogeneous dataset: $\sim$10 GiB composed of many small binary files. The goal is to take a look at differences with the heterogeneous results shown in Section 6. Let's review the results for the homogeneous dataset in the table below. 2

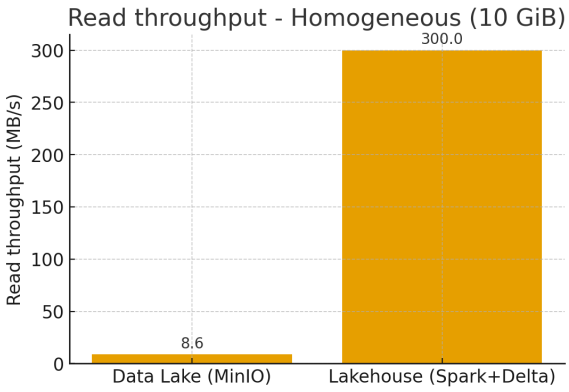
Table 2: Summary table for the homogeneous data

Metric / Scenario	**Data Lake (MinIO / S3 native)**	**Data Lakehouse (Spark → Delta / Parquet)**
Bulk upload	4.6 – 12.8 MB/s	∼79.8 – 111 MB/s
Bulk read aggregated throughput	∼8.6 MB/s	∼220–300+ MB/s
Single-object PUT latency (P50)	≈26 ms	–
Single-object GET latency (P50)	≈3.6 ms	≈250–300 ms
Metadata LIST / commit overhead	≈158 ms	metadata_write_seconds ≈8.6 s, metadata_bulk_seconds ∼0.7–3.6 s
Recovery time (MinIO restart)	∼1.1 s	∼1.1 s
Storage efficiency (user / consumed)	99.961%	≈99.966%

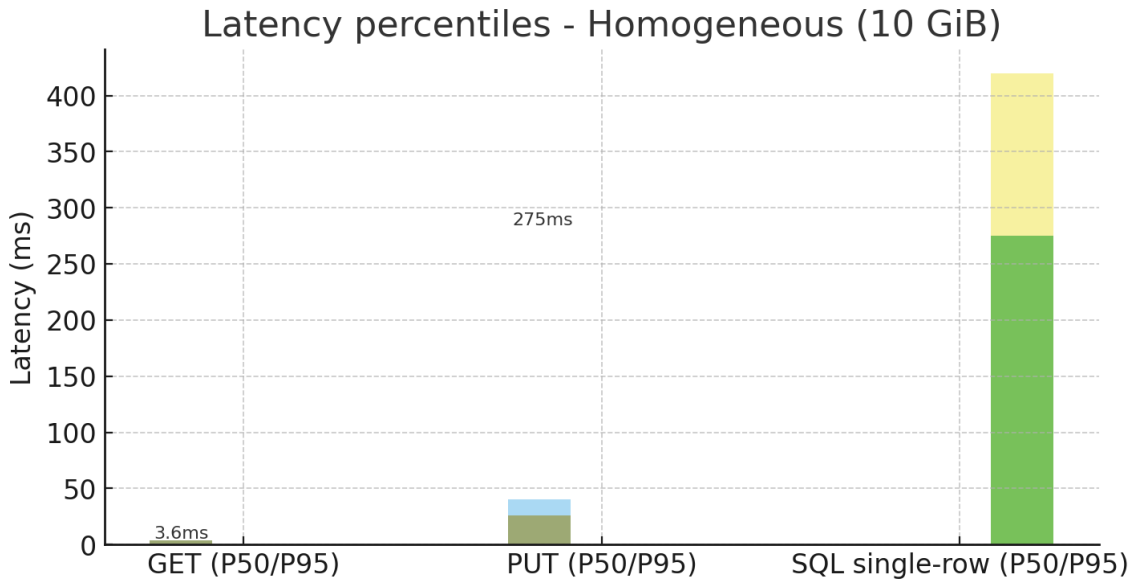
Legend:
Data Lake (MinIO) — native S3 client results .
Lakehouse (Spark+Delta) — parallel Spark aggregate results that include task orchestration and multipart uploads; performance improves with coalescing of outputs.
Throughput bars show aggregate MB/s across tasks.
Latency: P50 and P95 shown.



(a) Write throughput (MB/s) — Homogeneous (10 GiB).



(b) Read throughput (MB/s) — Homogeneous (10 GiB).



(c) Latency percentiles (P50 / P95) — GET / PUT (S3 primitive ops) and SQL single-row probe (Spark).

Figure 3: Performance summary for the homogeneous (10 GiB) dataset.

6.2.1 Throughput on homogeneous data

Data Lake Native S3 benchmarking of many small binary files performes normal aggregate rates per client: measured upload totals in different runs varied roughly between ∼4.6 and ∼12.8 MB/s. A single raw download in average gave ∼8.6 MB/s.

Data Lakehouse In this workload that is composed of many small binary files, when the ingestion is performed by Spark and written as fewer large Parquet files, aggregate write throughput still outperforms the datalake but does not show significant changes comparing to experiment of heterogeneous data. For reads, Spark's parallel scan can read in performed runs still faster than datalake.

Interpretation. Even for homogeneous binary data, the pattern similar to the heterogeneous experiments: single-client S3 transfer rates are modest, while an orchestrated parallel engine (Spark) achieves much higher *aggregate* throughput in comparison to datalake. Then when the goal is high aggregate throughput for batch analytics, Lakehouse-style ingestion typically outperforms datalake architecture.

6.2.2 Latency on homogeneous data

S3 primitive latencies. In this experiment I managed to perform the results only for P50. Measured single-object S3 operations remain low-latency: GET P50 around ∼3.6 ms; PUT P50 around ∼26 ms in the observed runs.

Spark/Delta single-record latency. Accessing a single record via Spark/Delta yields P50 in the ∼250–300 ms range and in P95/P99 significantly higher. For PUT-Latency I failed to make a proper measurment but it is obvious that the results would be still worser than datalake one. For OLAP-style scans this is acceptable, but for OLTP or low-latency single-record workloads it is insufficient.

Interpretation. If applications require many low-latency object reads/writes, the raw S3 Data Lake architecture performs better. For homogeneous small-file workloads this trade-off is crucial: Lakehouse improves batch throughput but penalizes single-update or single-record transaction latency.

6.2.3 Metadata overhead

S3 LIST behaviour. S3 LIST operations measured at ∼158 ms (mean) explain why metadata-related tasks that rely on multiple LISTs quickly accumulate latency.

Delta metadata costs. In the runs the Delta metadata pipeline showed observed values such as metadata_write_seconds ≈8.6 s in one run and metadata_bulk_seconds in the 0.7–3.6 s range across runs. The commit/checkpoint sequence can take multiple seconds. For a workload with many small commits this overhead becomes critical.

Interpretation. On a larger homogeneous metadata-heavy dataset, the relative importance of metadata operations increases: per-file or per-commit metadata work can increase total ingestion time. If the ingestion path can be reorganised to produce larger commits, the relative metadata cost per unit data will drop.

6.2.4 Recovery time and availability

Measured MinIO restart times remain short (∼1.1 s). Hence we can consider the fact that recovery time does not depend on the type of working data.

6.2.5 Storage efficiency

For homogeneous dataset:

- Raw S3 storage shows near-100% efficiency because there are essentially no additional metadata files stored.
- The Delta table variant also recorded high storage efficiency around ≈99.966% — so the Delta metadata overhead amounted to a few megabytes in these runs. We can say that the storage penalty of using a Lakehouse is insignificant for this measurement.

7 Conclusion

This study compared a plain Data Lake (S3/MinIO) and a Data Lakehouse (Spark + Delta/Parquet) on 2 representative workload classes: a heterogeneous dataset and a homogeneous, metadata-heavy dataset. We measured aggregate throughput, single-operation latency percentiles, metadata overhead, recovery time and storage efficiency.

7.1 Key findings

- **Aggregate throughput:** For batch ingestion and parallel analytics the Lakehouse achieves significantly higher aggregate throughput. Measured Spark write MBps values in our runs were typically in the ∼80–111 MB/s range for well-tuned jobs, while single-client S3 performed much lower per-client rates (upload ∼4.6–12.8 MB/s; download ∼8.6 MB/s). The difference is based mainly on Spark's parallelism.
- **Single-operation latency:** Raw S3 object operations are low latency: GET P50 ≈ 3–4 ms, PUT P50 ≈ 25–30 ms in our runs. A SQL single-record access executed through Spark/Delta shows P50 in the ∼250–300 ms , and this results are sufficient to say that Spark/Delta is unsuitable for latency-sensitive OLTP use cases.
- **Metadata overhead:** Delta's transactional model requires multiple S3 LIST/PUT operations per commit. These frequent small commits heavily penalize latency in Lakehouse architecture.
- **Recovery:** MinIO restarts in our tests were short. However, any object-store outage directly impacts Lakehouse transactionality and can cause retries/failures, so the shutting down of the storage could be more dangerous for Lakehouse architecture.
- **Storage efficiency:** Both approaches are highly efficient for large files.

7.2 Which architecture to choose

- **Choose Lakehouse when:** you need ACID semantics, versioning/time-travel, analytics is crucial and ETL workloads where high aggregate throughput and integrated analytics are priorities.
- **Choose Data Lake when:** you require low per-object latency, simple object storage semantics and you need a cheap good scaled storage.

7.3 Limitations

- Experiments were performed at minimal scale; absolute behavior may change at larger scales (100s GB–TB).
- Results depend on specific software versions, hardware/network environment.Different settings or different S3 implementations may change the overall results.
- Benchmarks used synthetic data generators and scripted workloads that approximate HPC environment; real scientific workflows may have additional complexities.

8 Future Work

We propose several extensions to improve the conclusions and results, increase the range of our experiment in terms of data volume, and include a Data Warehouse class in the comparison. Also we are planning to obtain the missing results that we have not calculated in this experiment (P95/P99 for homogeneous data for example)

8.1 Scale-up experiments

Large volume increasing: run experiments at larger scales: 100 GiB, 500 GiB and 1 TiB datasets. For each scale run both heterogeneous and homogeneous workloads. Key goals:

- observe whether aggregate throughput scales too,
- detects could large amount of data generates so huge amount of metadata that significantly decrease the datalake performance, maybe there is some point where difference in latency between these 2 architectures is not significant.
- quantify storage overhead and operational impact.

8.2 Include Data Warehouse (DW) in comparison

Design and run experiments that include one or more traditional Data Warehouse engines (ClickHouse, Greenplum) to compare against Lakehouse and Data Lake:

- **Motivation:** DWs implement *schema-on-write*, optimized OLAP query paths and often provide much lower query latency for many SQL patterns.

8.3 Concrete next steps

1. Run a 100 GiB sweep (heterogeneous + homogeneous) on the existing cluster.
2. Add ClickHouse as a DW target.

References

[1] Ahmed A. Harby and Farhana H. Zulkernine, "From Data Warehouse to Lakehouse: A Comparative Review," ResearchGate, 2022. [Online]. https://www.researchgate.net/publication/366670363_From_Data_Warehouse_to_Lakehouse_A_Comparative_Review

[2] Michael Armbrust et al., "Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics," CIDR 2021. [Online]. https://people.eecs.berkeley.edu/~matei/papers/2021/cidr_lakehouse.pdf

[3] Ahmed A. Harby and Farhana H. Zulkernine, "Data Lakehouse: A Survey and Experimental Study," Information Systems (2024). [Online]. https://www.sciencedirect.com/science/article/pii/S0306437924001182

Acknowledgments

I would like to thank the GWDG team and the University of Göttingen for providing access to the SCC cluster and for their continued technical support. The experiments reported in this paper benefited greatly from the availability of those resources and the helpful assistance of the system administrators.

Code, container are available in our GitLab repository :
`https://gitlab.gwdg.de/erdni.mankirov/datawarehousevsdatalake`