Frontiers in Emerging Computer Science and Information Technology (Open Access)

Volume 02, Issue 09, September 2025,
Publish Date: 09-18-2025

PageNo.30-33

Optimizing Modern Data Lakehouse Architectures: A Comprehensive Comparative Study of Delta Lake and Apache Parquet in Python Data Pipelines

Khalid Al-Mutairi
Center for Data Analytics, King Saud University, Riyadh, Saudi Arabia

Mariam Al Mazrouei
College of Engineering and IT, Ajman University, UAE

Abstract

The evolution of big data architectures from static warehouses to dynamic lakehouses has driven demand for storage formats that balance high performance, scalability, and strong data reliability. This paper provides an in-depth comparative study of Delta Lake and Apache Parquet within Python-based data pipelines, exploring their architectural differences, performance benchmarks, and operational implications. Using simulated workloads and a review of peer-reviewed literature, the study evaluates read/write throughput, concurrency, schema evolution, governance, and total cost of ownership. Results demonstrate that Parquet remains highly efficient for stable, append-only analytical workloads, while Delta Lake extends functionality with ACID transactions, schema evolution, and time travel, improving reliability and flexibility for mixed batch-streaming environments. The findings inform practitioners designing modern data lakehouse systems that integrate analytical and operational workloads under unified governance frameworks.

Keywords: *Delta Lake, Apache Parquet, Python pipelines, PySpark, ACID transactions, data lakehouse, schema evolution, benchmarking, data governance, performance optimization*

1. Introduction

1.1 Background and Motivation

In the last decade, data engineering has transitioned from traditional **ETL pipelines** feeding relational warehouses to large-scale **data lakes** and now **data lakehouses** that unify analytical and transactional processing. As organizations increasingly rely on Python-based ecosystems such as **Apache Spark**, **PyArrow**, and **Pandas**, the underlying data-storage format has become critical to system performance, scalability, and governance (Armbrust et al., 2020; Melnik et al., 2010).

The **Apache Parquet** format, introduced in 2013, revolutionized big-data analytics by offering a columnar, compressed representation ideal for distributed queries. However, it was designed primarily for **append-only batch workloads**, lacking transactional semantics. As enterprises sought **real-time analytics**, **incremental updates**, and **schema flexibility**, this limitation became evident (Le Dem, 2013; Saeedan & Eldawy, 2022).

Delta Lake, an open-source storage layer developed by Databricks, builds upon Parquet to provide **ACID transactions**, **schema enforcement and evolution**, and **time-travel capabilities** through a structured **transaction log**. This hybrid approach underpins the emerging **lakehouse paradigm**, bridging the gap between traditional data warehouses and flexible data lakes (Armbrust et al., 2020; Geetla, 2025).

1.2 Problem Statement

Although both Parquet and Delta Lake are widely adopted, practitioners face a key question: *when is Delta Lake's additional complexity justified compared to Parquet's simplicity?* The answer depends on workload patterns, data governance requirements, and cost considerations.

The study **"Evaluating Effectiveness of Delta Lake Over Parquet in Python Pipeline"** (2025) highlights Delta's superior consistency and update performance in Python pipelines but acknowledges a cost overhead of 40–60 %. This work expands on those findings, synthesizing existing research with new experimental benchmarks to present a holistic view of the trade-offs.

1.3 Objectives

This paper aims to:

1. Evaluate architectural and performance differences between Delta Lake and Parquet in Python-based data pipelines.
2. Quantify their efficiency across batch, streaming, and mixed workloads.
3. Discuss governance, cost, and maintenance implications for enterprise adoption.
4. Propose best-practice guidelines for hybrid implementation.

2. Literature Review

The **columnar storage revolution** originated with Google's *Dremel* (Melnik et al., 2010), which inspired open formats such as Parquet and ORC. Columnar formats minimize I/O by scanning only relevant columns and compressing homogeneous data efficiently (Le Dem, 2013). However, they traditionally assume static, immutable datasets.

2.1 Apache Parquet in Analytical Pipelines

Parquet supports efficient analytical queries, compression codecs (Snappy, GZip, ZSTD), and complex nested structures (Apache Parquet, 2023). Its open specification enables wide compatibility across Spark, Trino, Presto, and cloud warehouses such as AWS Athena and Google BigQuery. Yet, Parquet lacks internal **transaction management** and **record-level mutation**, requiring external orchestration or snapshotting to maintain data consistency (Library of Congress, 2023).

2.2 Emergence of Transactional Data Lakes

The growing demand for data reliability in lakes spurred the development of **transactional layers** such as Delta Lake, Apache Hudi, and Apache Iceberg. These frameworks wrap columnar data with metadata logs to ensure **atomicity, consistency, isolation, and durability (ACID)**. They also introduce schema versioning and time-travel, critical for regulatory compliance and machine-learning reproducibility (Armbrust et al., 2020; Camacho-Rodríguez et al., 2023).

2.3 Delta Lake Architecture

Delta Lake stores data in Parquet files but tracks every commit in a JSON/Parquet transaction log. Each transaction records **adds**, **removes**, **schema changes**, and **metadata updates**. Query engines reconstruct table state by reading the latest valid log entry, guaranteeing serializable isolation (Powers, n.d.). Features such as **Z-ordering**, **OPTIMIZE**, and **VACUUM** enhance performance and housekeeping (Microsoft Fabric, 2025).

2.4 Comparative Findings in Prior Studies

The 2025 study by Donthi demonstrated that Delta Lake achieved up to **2× faster read performance** in concurrent workloads and near-linear scalability in Python pipelines with frequent updates, while Parquet excelled in static batch analytics. Similar conclusions were drawn by Databricks and open-source benchmarks such as **LST-Bench**, which evaluated Delta, Iceberg, and Hudi under identical conditions (Camacho-Rodríguez et al., 2023).

Overall, literature indicates that the optimal format depends on the **mutability** and **concurrency** of workloads rather than raw query performance.

3. Methodology

3.1 Experimental Framework

To analyze real-world conditions, this study simulated a **Python-based Spark pipeline** using open datasets representative of log, transaction, and IoT data. Experiments were conducted in a distributed environment (4 nodes, 16 vCPUs each, 64 GB RAM, SSD storage).

The pipeline architecture included:

- **Ingestion Layer:** CSV and JSON data ingested via PySpark DataFrames.
- **Processing Layer:** Transformation, joins, and aggregations.
- **Storage Layer:** Parallel writes to Delta Lake and Parquet tables.
- **Query Layer:** Analytical queries using Spark SQL and PyArrow.

3.2 Workload Types

Three workload archetypes were tested:

1. **Batch Analytics** – daily append of 50 GB; read-heavy queries.
2. **Incremental Streaming** – micro-batches of 1 GB/10 s; concurrent writes.
3. **Mixed Lakehouse** – upserts, deletes, and merges simulating CDC operations.

3.3 Evaluation Metrics

The following metrics were captured:

- Read latency (ms per query)
- Write throughput (MB/s)
- Update/delete execution time (ms)
- Metadata operation cost (listings, compaction)
- Schema evolution time (ms)

- Storage overhead (%)
- Operational cost index (relative cluster utilization)

3.4 Benchmark Procedure

Each workload was executed thrice to average variance. Cache was cleared between runs. Delta tables were optimized via **Z-ORDER** and **OPTIMIZE** commands; Parquet tables were compacted manually. Metrics were logged with Spark's internal performance counters.

4. Results

4.1 Read and Write Performance

Metric	Parquet (Batch)	Delta Lake (Batch)	Delta Lake (Streaming)
Average read latency (ms)	420	380	410
Average write throughput (MB/s)	275	260	240
File-listing overhead (s)	8.2	3.1	3.5
Storage overhead (%)	0	7.4	8.1

While Parquet delivered marginally higher write throughput, Delta Lake reduced metadata overhead by 50–60 %, improving query initialization time. Read performance improved 10 % under optimized layouts.

4.2 Updates, Deletes, and Schema Evolution

Operation	Parquet	Delta Lake
UPDATE 10 M records	185 s (external merge)	61 s
DELETE 5 M records	Not natively supported	42 s
Schema add column	Manual rewrite (≈ 110 s)	15 s (auto-evolution)

The transactional log enabled efficient upserts and dynamic schema modifications without rewriting entire datasets—one of Delta's core advantages.

4.3 Time Travel and Auditability

Delta Lake allowed querying historical versions within 1–2 s, facilitating reproducibility and rollback. Parquet required manual version directories, adding complexity and storage redundancy.

4.4 Cost and Complexity

Operational cost analysis showed Delta's cluster utilization averaged **1.42×** that of Parquet due to log maintenance and checkpointing. However, downtime and recovery incidents dropped significantly—simulated recovery time decreased from 25 min (Parquet) to under 5 min (Delta).

5. Discussion

5.1 Performance Interpretation

Parquet's raw I/O efficiency stems from its minimal metadata layer. For **read-only analytical workloads**, it remains unmatched in cost efficiency. However, in modern environments where data mutability and governance matter, Parquet's lack of transactional consistency becomes a liability. Delta Lake's additional log and metadata operations introduce slight overhead but yield higher resilience and correctness.

5.2 Governance and Reliability

Delta Lake's **ACID guarantees** eliminate partial-write corruption common in distributed environments. The ***delta_log*** enables precise audit trails for each transaction, fulfilling compliance standards such as GDPR or SOX. Parquet's governance relies on external catalog management or manual snapshots.

5.3 Cost-Benefit Trade-offs

Although infrastructure cost for Delta can rise by 40–60 % (Donthi, 2025), the total cost of ownership often evens out when considering downtime prevention, easier debugging, and faster schema migration. The cost curve favors Delta as data mutability and concurrency grow.

5.4 Hybrid Implementation Strategy

A **hybrid storage approach**—Parquet for raw, immutable "bronze" layers and Delta for curated "silver" and "gold" analytical layers—offers the best compromise. This design aligns with the lakehouse paradigm used in Databricks and open Spark communities (Armbrust et al., 2020). Python developers can interact seamlessly through **PySpark DeltaTable APIs**, abstracting the complexity of transaction logs.

5.5 Implications for Python Ecosystem

Delta Lake integrates natively with **PySpark**, **pandas**, and **Arrow**, enabling a unified programming model. Advanced features such as MERGE INTO, UPDATE, and VACUUM are exposed through Python APIs, making it highly accessible to data scientists. Conversely, Parquet remains the backbone for cross-engine compatibility—ideal for lightweight analytics and ML model training.

6. Limitations and Future Work

This study's simulated environment cannot fully represent multi-cloud heterogeneity or extremely large (petabyte-scale) datasets. Future research should integrate **real-world telemetry** and **mixed-format workloads** (e.g., Iceberg, Hudi) to expand the comparative scope. Empirical cost modeling with real cloud billing would also refine understanding of operational efficiency.

7. Conclusion

This comprehensive evaluation establishes that **Delta Lake** and **Parquet** each serve critical yet distinct roles in modern data ecosystems.

- **Parquet** is ideal for stable, read-intensive, and cost-sensitive analytical workloads.
- **Delta Lake** excels where **data reliability**, **schema evolution**, **upserts/deletes**, and **governance** are paramount.

Organizations adopting the **lakehouse** model should strategically combine both—retaining Parquet for archival or immutable layers and employing Delta for dynamic, business-critical analytics. In Python-based pipelines, this hybrid approach yields optimal balance between speed, safety, and scalability.

References

1. Armbrust, M., Das, T., Li, Y., et al. (2020). *Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores.* **Proceedings of the VLDB Endowment, 13**(12), 3411–3424. https://doi.org/10.14778/3415478.3415560
2. Apache Parquet. (2023). *Apache Parquet File Format.* Retrieved from https://parquet.apache.org
3. Camacho-Rodríguez, J., Nitu, F., Kemper, A., & Neumann, T. (2023). *LST-Bench: Benchmarking Log-Structured Tables in the Cloud.* arXiv:2305.01120.
4. Geetla, D. (2025). *Optimizing ETL Workflows Using Databricks and Delta Lake.* HEXstream Tech Corner. https://www.hexstream.com/tech-corner
5. Le Dem, J. (2013). *Parquet: Columnar Storage for the People.* Strata + Hadoop World.
6. Library of Congress. (2023). *Apache Parquet File Format (FDD000575).* https://www.loc.gov/preservation/digital/formats
7. Melnik, S., Gubarev, A., Long, J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T. (2010). *Dremel: Interactive Analysis of Web-Scale Datasets. Proceedings of the VLDB Endowment, 3*(1), 330–339.
8. Microsoft Fabric Documentation. (2025). *Delta Table Optimization and V-Order.* https://learn.microsoft.com/en-us/fabric/data-engineering/delta-optimization
9. Powers, M. (n.d.). *Delta Lake vs Parquet: Comparison.* Delta.io Blog. https://delta.io/blog/delta-lake-vs-parquet-comparison
10. Evaluating Effectiveness of Delta Lake Over Parquet in Python Pipeline. (2025). International Journal of Data Science and Machine Learning, 5(02), 126-144. https://doi.org/10.55640/ijdsml-05-02-12
11. Saeedan, M., & Eldawy, A. (2022). *Spatial Parquet: A Column File Format for Geospatial Data Lakes.* arXiv:2209.02158.