International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V73I1P108

Volume 73 Issue 1, 65-71, January 2025

Original Article

A Comparative Study of Delta Lake as a Preferred ETL and Analytics Database

Hanza Parayil Salim

Staff Engineer, Neiman Marcus, USA.

Corresponding Author : hanzapsalim@gmail.com

Received: 22 November 2024 Revised: 28 December 2024 Accepted: 15 January 2025 Published: 30 January 2025

Abstract *- In the world of modern data architecture, Delta Lake stands out as a powerful and reliable solution to handle large amounts of data. This comparative study explores Delta Lake as a potential solution for Extract, Transform, Load (ETL) processes and analytics. Delta Lake, an open-source storage layer built on top of Apache Spark and optimized for cloud environments, promises enhanced reliability, scalability, and performance for data pipelines. The study evaluates its advantages over traditional databases and other big data processing frameworks, focusing on aspects such as data consistency, transaction management, and schema evolution. By analyzing key features like ACID transactions, time travel, and integration with cloud platforms, this paper provides a comprehensive assessment of Delta Lake's effectiveness for ETL workflows and analytical workloads. The study highlights its strengths in handling large datasets over Data Lake and traditional databases for analytical data processing.*

Keywords *- Delta Lake, Lakehouse architecture, Data Lake, Medallion architecture, Databricks, ETL, Apache spark, Distributed computing.*

1. Introduction

The data explosion we see today is fueled by new technology, digital services expansion, more connectivity and increasing use of data for decision-making, automation and personalization. That has generated increased demand for big data processing and has led to the emergence of new technologies like Delta Lake for data processing and storage. Delta Lake is a storage layer built on top of Apache Spark where Spark is the distributed computing platform to process massive datasets in parallel. Delta Lake then improves the reliability, performance, and governance of data lakes.

The combination of Apache Spark and Delta Lake forms a powerful solution for large-scale data processing and analytics. This paper explores various aspects of Delta Lake, highlighting its advantages over traditional data lakes and databases. It explores how Delta Lake functions as both an Extract, Transform, Load (ETL) layer and an analytics platform, offering a contrast to traditional databases.

2. Data Lake

Data Lake is a general term that describes a data storage methodology that can exist in any storage where we can store hybrid data formats. For example, you might have a data lake containing Parquet, JSON and unstructured text files located in an AWS S3 bucket or Azure BLOB storage. A BLOB location with Parquet files can be considered a data lake. Or an Azure BLOB containing many different file types with data formatted as tables, JSON, XML, etc. The flexibility of data lakes can become problematic as data scales. Without proper management, they can easily turn into "data swamps," where it becomes challenging to track file versions data schemas, or recover from accidental data operations.

Data lakes are prone to corruption, often requiring manual cleanup, and they lack reliability guarantees. Storing various file formats and data types without proper versioning or schema enforcement can quickly lead to disorganization. However, data lakes are still an excellent choice for storing raw data or serving as the initial ingestion layer due to their scalability, flexibility with data types, low storage costs, schema-on-read feature, and ability to handle large volumes of data.

Fig. 1 Delta lake logo

3. Delta Lake

Delta Lake is an open-source table format for data storage that extends Parquet data files with a file-based transaction log. Delta Lake (or Delta Table) represents a significant evolution in data storage technology, designed to bring reliability and robustness to data lakes.

3.1. Understanding Delta Lake Architecture

3.1.1. Storage Layer

Delta Lake stores data in Parquet format, which is a compressed, dictionary encoded and uses a columnar storage format optimized for big data processing. Parquet files ensure high performance due to parallel operations and efficient storage. The storage layer should be any cloud storage like AWS S3, Azure BLOB or Google Cloud.

3.1.2. Transaction Log

The foundation of Delta Lake's architecture is its transaction log. The log records are made to the data in a Delta table in a sequential manner. This crucial component is stored in a '_delta_log' subdirectory within the table's location. The log consists of two main elements.

JSON Files

These are numbered sequentially (e.g. 00000.json, 00001.json, etc.) and represent individual atomic commits.

Parquet Checkpoints

Parquet checkpoints provide efficient snapshots of the table state at specific points in time. The transaction log serves as the single source of truth for the table's state, enabling Delta Lake to provide its key features and guarantees. The actions are then documented in the transaction log as ordered, atomic units termed commits.

3.1.3. Delta Engine

Delta Lake runs on top of Apache Spark, utilizing its distributed processing capabilities. The Delta Engine ensures efficient execution by optimizing the queries. Delta Lake takes the existing Parquet data lake. It makes it more reliable and performant by storing and tracking all the metadata in a separate transaction log and organizing the data for maximum query performance. This is commonly known as data Lakehouse architecture.

3.2. How Delta Lake Implements ACID Properties

The ACID properties collectively offer a mechanism to guarantee the correctness and consistency of a database. They ensure that each transaction functions as a single unit of operations, produces consistent outcomes, operates independently of other transactions and that any updates made are persistently stored.

3.2.1. Write-Ahead Logging

Write-Ahead Logging (WAL) is a critical mechanism in Delta Lake that ensures data integrity and enables ACID transactions. Delta Lake logs the transaction details in the transaction log before making any changes to the data files. Each JSON file represents a new version of the table and is updated atomically for every operation. WAL ensures the atomicity and durability of transactions. If a system failure occurs, the log contains sufficient information to either complete the transaction during recovery or roll it back, preventing partial updates.

3.2.2. Optimistic Concurrency Control

Optimistic Concurrency Control (OCC) is a method used in database management systems and other distributed systems. In the context of Delta Lake, OCC is employed to manage multiple writers without locking the entire table. This enables writers to perform operations without acquiring locks, improving performance in scenarios with low conflict rates. Involves checking for conflicts only at the time of commit, rather than throughout the entire transaction.

WAL and optimistic concurrency control work together in Delta Lake to provide ACID transactions while maintaining high performance and concurrency. This synergy between WAL and optimistic concurrency control allows Delta Lake to provide robust transaction management while optimizing for the high-concurrency, append-heavy workloads common in big data environments.

3.2.3. Atomicity

Delta Lake achieves atomicity through write-ahead logging. Before executing any changes to the data files, it writes the details of the transaction to the transaction log. Atomicity ensures that operations (such as INSERT or UPDATE) on your data lake are either fully executed or not executed at all. Delta Lake can offer the guarantee of atomicity through the transaction log.

3.2.4. Consistency

Consistency in Delta Lake is maintained through Schema Enforcement and Invariant Checking. Delta Lake automatically checks that incoming data adheres to the table's schema, as defined in the transaction log metadata. If a transaction tries to insert data that doesn't comply with the current schema, it is rejected. Any user-defined invariants (like NOT NULL constraints) are enforced before committing the transaction. If these invariants are violated, the transaction is aborted.

3.2.5. Isolation

Delta Lake provides snapshot isolation, which protects reading transactions from ongoing modifications. This is achieved when a read transaction starts, and it points to a specific version of the data, which corresponds to a state of the transaction log. Subsequent modifications (writes) do not affect this version, ensuring that the read transaction sees a consistent and unchanging view of the data, even as other modifications proceed.

3.2.6. Durability

Durability in Delta Lake is ensured through the immediate recording of committed transactions. As soon as a transaction is committed, it is recorded in the transaction log. The log is typically flushed to persistent storage immediately. This approach ensures that the transaction's effects persist across system failures. The recovery process uses the log to reapply committed transactions that might not have been written on the data files before a failure. Delta Lake gives both more reliability and more flexibility than regular Parquet files. As we have already seen, ACID transactions via the transaction log give us production-grade reliability guarantees.

3.3. Performance Advantages of Delta Lake

Delta Lake employs various inbuilt performance optimizations to enhance query efficiency and data management. The core of Delta Lake optimization techniques, Optimize, Z-Order, and Vacuum, go beyond being mere optimization tools. Let's dive deeper into these techniques.

3.3.1. Parquet Files and Data Skipping

Delta stores data in parquet format, and parquet files store column statistics for row groups in the metadata footer. This supports query optimization techniques like predicate pushdown and column pruning.

Delta Lake takes this further. In addition to this Parquet functionality, Delta Lake keeps metadata at the file level in a single transaction log. So, by using a single network request, query engines can identify which data can be skipped. This utilizes statistics in the transaction log to skip irrelevant files during queries, significantly reducing query time. Delta engine checks the per-file metadata in the transaction log while running the queries. This metadata is also used to skip the files that are not needed for the query.

To make this possible, Delta automatically collects min/max statistics about the first N columns (default 32, configurable) and writes it to the Delta log. These statistics will be used to locate data in specific files when performing joins. Reducing the number of files to read can significantly enhance performance. But, min/max statistics are not very efficient for strings and are better for numeric date/time data types. Delta Lake can enhance the query performance by identifying data that is irrelevant to the query. In this way, the query engine can bypass entire files, preventing unnecessary data reads.

3.3.2. Compaction and Z-Ordering

Compaction combines small files into larger ones to improve read performance and reduce metadata overhead and this is achieved by the OPTIMIZE command. Z-Order is a data organization technique that plays a key role in enhancing query performance. It optimizes the physical layout of data by collocating related information, effectively reorganizing the data in storage. This allows certain queries to read less data, leading to faster execution. When data is properly ordered, more files can be skipped, which improves query efficiency by reducing the amount of data that needs to be processed. Z-Ordering can be applied alongside the OPTIMIZE command.

By using Z-Ordering, query performance can be significantly improved, as it increases the likelihood that unnecessary data can be skipped, making queries run faster. This graph illustrates the performance of a join query involving two tables, each with 400 million records, both before and after Z-Ordering.

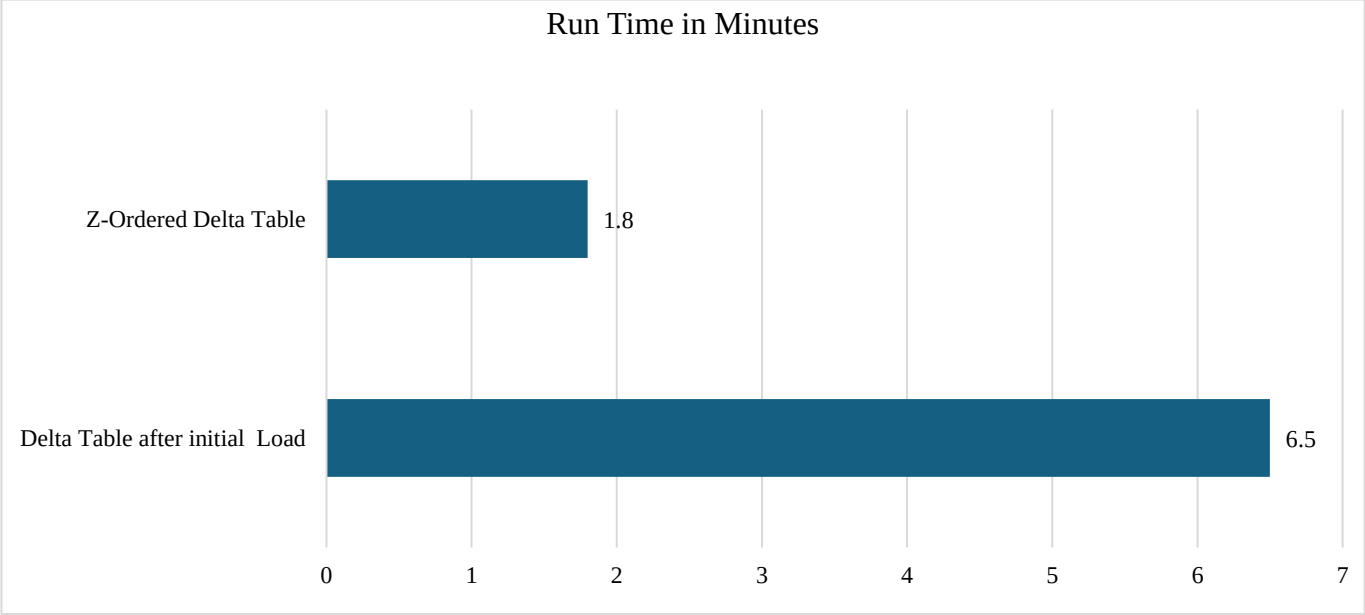

Fig. 2 Performance comparison: z-ordered delta table vs. initial load delta table (run time in minutes)

3.3.3. Liquid Clustering

Delta Lake offers Liquid clustering to colocate similar data in the same files so queries can run faster. Liquid clustering is a newer algorithm for Delta Lake tables that offers several advantages: the ability to change clustering columns at any time, which means data engineering does not need to decide what the query patterns would look like, Optimized for unpartitioned tables, does not recluster previously clustered files. It eliminates the concept of partitions. Liquid clustering relies on Optimistic Concurrency Control (OCC) to handle conflicts when multiple writes write to the same table. We need to adhere to best practices when selecting partitioning/Z-Order or liquid clustering, which will be covered in the next section.

3.3.4. Vacuum

Vacuum removes old files no longer needed for time travel or snapshot isolation, helping to manage storage costs. Vacuum is an essential operation for maintaining the health of Delta Lake. As data evolves, gets updated, or is deleted over time, it can leave behind outdated data files that consume valuable storage space. The Vacuum operation helps by removing these unnecessary files.

3.3.5. Time Travel and Rollbacks

One of Delta Lake's standout features is its support for time travel capabilities. This feature is made possible by the multi-version transaction log. Users can query previous versions of the table using Version numbers and Timestamps. Time travel is particularly useful for undoing errors in data pipelines, reproducing old versions of data for compliance purposes and Comparing data across different points in time. By using Vacuum, Z-Order, and Optimize, we can attain notable improvements in performance, operational efficiency, and cost savings.

3.4. Best Practices in Delta Lake Design

Here, we discuss the best practices for designing a Delta Lake. When creating a Delta table, it's recommended to position the keys and columns frequently used for filtering on the left side, as the first 32 columns are used to gather min/max statistics for file skipping. The graph below demonstrates the performance of a query involving two tables with 600 million records and 85 columns, comparing the results when the filter or join keys are within the 32-column limit versus when they exceed it. Optimization can introduce some overhead during execution, and Z-Ordering requires careful selection of columns. Vacuum also needs to be properly scheduled and configured. When using Z-Order, consider high cardinality columns, such as join keys, as they facilitate better collocation.

Consider low cardinality columns for Partitions like CustomerType, Date, etc, which depend on the volume of data, and that avoids too many partition problems. While partitioning helps with performance, too many partitions can result in overhead. A good rule of thumb is to keep partition counts under 10,000. To avoid the performance penalties associated with small partitions, each partition should hold at least 1 GB of data. Partitioning combined with Z-ordering is a more traditional approach that allows for greater control over data organization, Supports parallel writes more effectively.

However, data engineering must be aware of querying patterns up front to choose a partition column. For large tables with more than 10 TB of Data Partitioned, Z-Ordering is helpful. If the size is less than 10 tb and If the downstream users are aware of the partition and use that partition to query, then also Partitioned, Z-Ordering is helpful; else, liquid clustering as we have the flexibility to change the clustering over time. For partitioned tables, make sure new partitions are clustered and Z-ordered to enable efficient querying.

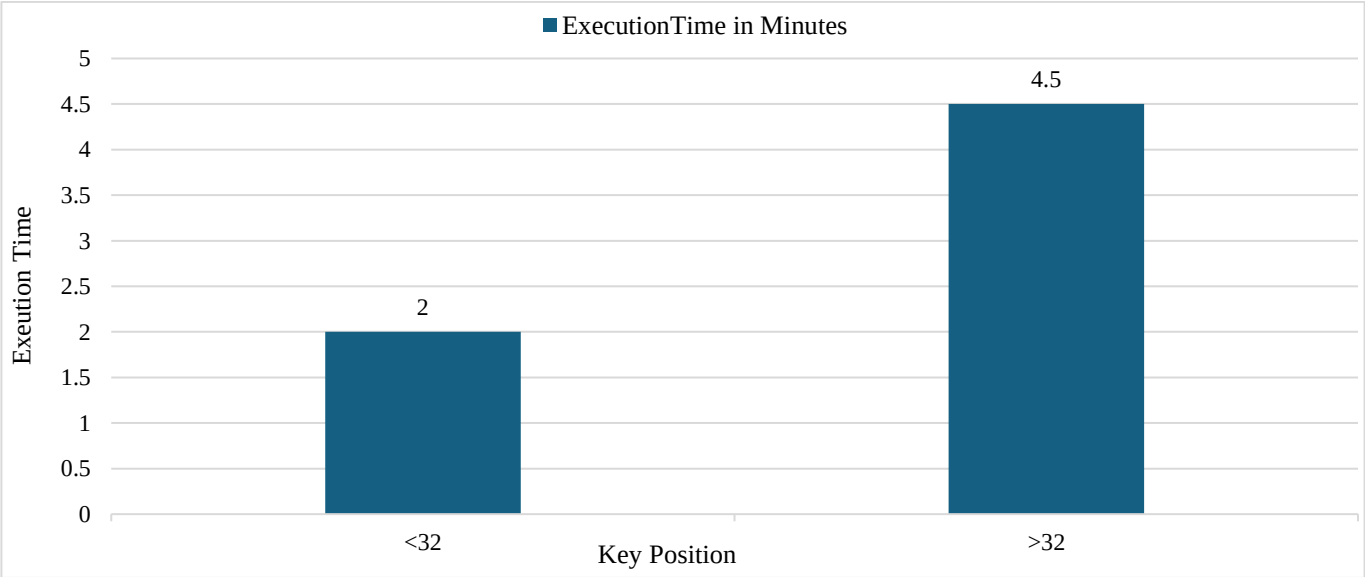

Fig. 3 Execution time comparison based on key position

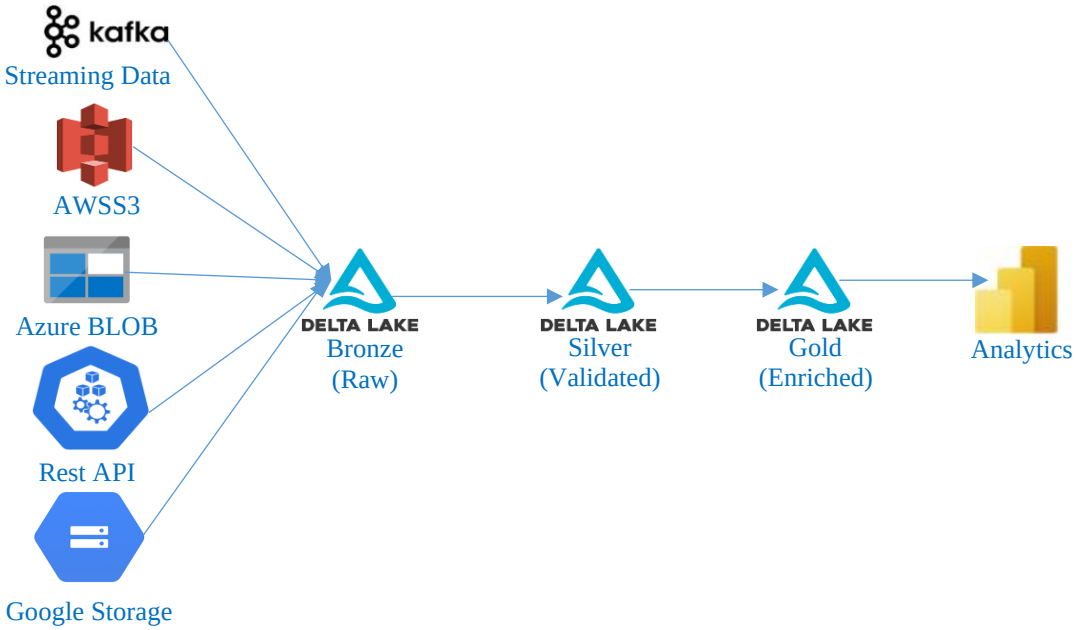

Fig. 4 Data processing and cloud storage flowchart

Additionally, including the WHERE clause is essential for optimal performance. Liquid Clustering is the right choice if your table is small to medium-sized (less than 10 TB) and users don't consistently query with a partition column. The choice between liquid clustering and partitioned Z-order tables depends on several factors, including query requirements, table size, and write patterns. For smaller tables or those without clear partitioning strategies, liquid clustering offers simplicity and efficiency. Larger tables or those with concurrent write needs often benefit from partitioning with Z-ordering. For Liquid Clustered tables, simply running the optimize command at regular intervals will not recluster files that have already been clustered.

Always take your specific use case into account and be ready to test both approaches to find the best fit for your query patterns and data. The correct choice will have a major effect on your query performance and overall data management efficiency.

4. Advantages of Delta Lake over Data Lake

Delta Lake enhances the reliability, performance, and developer experience of traditional data lakes by offering features such as Atomicity, Consistency, Isolation, Durability (ACID) transactions, which ensure reliable read and write. Tracking file versions in a data lake can be challenging, but Delta Lake addresses this issue by supporting data versioning and time travel, which allows access to historical versions of the data. Traditional data lakes may experience performance challenges as data volumes increase, mainly due to the absence of optimization features. Delta Lake resolves these challenges by applying data compaction and indexing optimization techniques. Delta Lake also includes high-performance query optimizations, such as file skipping, to boost query efficiency. Delta Lake also supports colocating similar data using Z-Order and Liquid Clustering, which enhances query performance.

When we need more flexibility in your schema, Delta Lake also supports Schema Evolution. The ability to change the structure of data over time is a common requirement in data management, known as schema evolution. In traditional data lakes, schema evolution can be challenging; Delta Lake simplifies schema evolution by allowing you to add, modify, or delete columns in a table without disrupting data pipelines. Additionally, it tracks changes, making it easier to understand how your data has evolved over time. Schema enforcement and evolution in Dela help maintain data consistency and integrity by preventing errors, resulting in improved data quality and more efficient data governance. Additionally, Delta Lake supports flexible data operations, such as dropping and renaming columns, deleting rows, and other DML operations, which are not typically possible in a standard data lake.

5. Delta Lake as an ETL Layer

Delta Lake is an excellent choice for Extract, Transform, Load (ETL) workloads due to its performance and reliability. Here are some key features of Delta Lake that enhance ETL processes: ACID transactions, multiple query optimization techniques, and data access optimized for fast retrieval and analysis. Delta Lake efficiently handles large volumes of data without sacrificing performance. One of the key aspects of Delta Lake's reliability is schema enforcement and evolution. Additionally, Delta Lake supports both ETL and ELT processes, allowing data storage and transformation to occur

efficiently at any stage. Suppose you need the ability to easily revert to earlier versions of your data and seek flexibility in how and when you transform or load your data. In that case, Delta Lake can be a great choice for your ETL workloads if you are working with big datasets and want to improve performance.

In addition to the previously mentioned benefits, another advantage of using Delta Lake as an ETL layer is its compatibility with open-source programming languages like Python and Java. This allows you to avoid relying on third-party ETL tools for loading data into Delta Lake. With just a Spark compute engine, you can use these languages to write your transformations. Both Python and Java offer a wide range of libraries to connect to various data sources, including REST APIs, traditional databases, cloud databases, live streaming sources like Kafka and more. This provides flexibility in how and when you transform or load your data into Delta Lake, eliminating the need for traditional ETL tools. The widely used design paradigm known as the Medallion Architecture is implemented using Delta Lake. This architecture organizes and processes data in layers, typically called bronze (raw), silver (validated), and gold (enriched), each reflecting the quality of the data at that stage. It ensures that as data passes through these layers of validation and transformation, it maintains Atomicity, Consistency, Isolation, and Durability (ACID properties). By the time the data is stored in its final form, it is optimized for efficient analytics.

6. Delta Lake as an Analytical Layer

Delta Lake seamlessly integrates with BI tools, allowing users to easily connect and consume data directly from Delta using built-in connectors. For example, tools like PowerBI and Tableau can connect to Delta Lake through their user-friendly interfaces. One of the most popular platforms for using Delta Lake is Databricks. If you're looking for an analytics solution, Databricks combined with Delta Lake provides a unified platform for both BI and ML needs. Databricks Unity Catalog serves as a centralized data governance platform, managing access control and metadata for Delta Lake tables. Delta Sharing, the industry's first open protocol for secure data sharing, simplifies the process of sharing data with other organizations, regardless of the computing platforms they use. Traditional relational databases rely on indexing to speed up data retrieval. However, in Delta Lake, which stores vast amounts of data, separate index structures become impractical due to their storage overhead. Instead, it uses clustering to physically sort the table data, providing similar performance benefits without incurring additional storage costs. This is a significant advantage of using Delta Lake as an analytical layer/Data Serving layer. The key difference is that clustering sorts the data within the table itself rather than creating separate index structures. Delta Lake takes advantage of Apache Spark's scalability and cloud storage, enabling efficient data serving at scale. Its integration with the cloud also ensures cost-effective storage and easy access to large volumes of data. Unlike traditional databases, where table administration is typically managed by dedicated administrators, Delta Lake allows the development team to handle table optimization and maintenance. This reduces the need for specialized resources and provides a cost advantage. Delta Lake can be used as a serving database for customers, offering an alternative to traditional MPP databases. By offering a unified data management layer, Delta Lake bridges the gap between traditional data warehouses and modern data lakes. The scalability and flexibility of data lakes and the reliability and performance of data warehouses are combined in Delta Lake and there by delivering the best of both worlds.

7. Conclusion

Delta Lake provides notable benefits, including data versioning, ACID transactions, enhanced data reliability, scalable metadata handling, faster query performance, and seamless integration with Spark. These features make it an ideal solution for large-scale data processing and analysis, particularly in big data environments where data consistency and reliability are essential. Delta Lake has combined the scalability of data lakes with the reliability of data warehouses, positioning itself as the leading storage format within the lakehouse paradigm. Performance is crucial in data processing, and these optimizations substantially enhance query performance, making Delta Lake an attractive option for organizations with complex analytical workloads.

The Medallion architecture provides an effective framework for building ETL pipelines, and Delta Lake is an excellent choice for implementing the Medallion architecture, as it supports reliable transactions, is compatible with multiple engines, and offers features that are beneficial at each stage of the ETL pipeline. Delta Lake delivers a robust foundation for providing high-quality data to end users or applications requiring accurate, real-time insights from large datasets. As the field of big data management and analytics evolves, Delta Lake is set to play a key role by providing organizations with improved data reliability, enhanced performance, and increased flexibility in handling large-scale datasets.

References

[1] Michael Armbrust et al., “Delta Lake: High-performance ACID Table Storage Over Cloud Object Stores,” *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3411-3424, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[2] Apache Parquet. [Online]. Available: https://parquet.apache.org

[3] Data Skipping for Delta Lake. [Online]. Available: https://docs.databricks.com/en/delta/data-skipping.html

[4] Xiang Wu, and Yueshun He, “Optimization of the Join between Large Tables in the Spark Distributed Framework,” *Applied Sciences*, vol. 13, no. 10, pp. 1-14, 2023. [CrossRef] [Google Scholar] [Publisher Link]
[5] Apache Kafka. [Online]. Available: https://kafka.apache.org
[6] Use Liquid Clustering for Delta Tables, 2025. [Online]. Available: https://docs.databricks.com/en/delta/clustering.html
[7] Databricks Runtime 15.3. [Online]. Available: https://docs.databricks.com/en/release-notes/runtime/15.3.html
[8] Structured Spark Streaming with Delta Lake: A Comprehensive Guide, 2024. [Online]. Available: https://delta.io/blog/structured-spark-streaming/
[9] Azure Data Lake Storage. [Online]. Available: https://azure.microsoft.com/en-us/services/storage/data-lake-storage/
[10] Delta Lake Performance. [Online]. Available: https://delta.io/blog/delta-lake-performance/
[11] Built Lakehouse with Delta Lake. [Online]. Available: https://delta.io/
[12] Delta Sharing. [Online]. Available: https://learn.microsoft.com/en-us/power-query/connectors/delta-sharing