Sarcouncil Journal of Multidisciplinary

ISSN(Online): 2945-3445

Volume- 05| Issue- 07| 2025

Research Article **Received:** 01-06-2025 | **Accepted:** 18-06-2025 | **Published:** 07-07-2025

High Performance Read Operations in Merge-On-Read with Deletion Vectors in Apache Iceberg

Piyush Dubey
University of Iowa, USA

Abstract: Data lakes have become essential infrastructure for modern analytics, but traditional Copy-on-Write (COW) memory management faces severe performance degradation at petabyte and zettabyte scales due to write amplification when handling updates and deletions. Apache Iceberg addresses these challenges through the Merge-on-Read (MOR) architecture with deletion vectors, which track deleted rows in compact bitmap structures rather than rewriting entire data files. This implementation separates metadata from data files, enabling efficient query planning while maintaining immutable data files, and applies deletion masks during query execution to filter deleted rows without physical data movement. Performance evaluation using TPC-DS and TPC-H benchmarks demonstrates that MOR with deletion vectors significantly reduces write overhead while maintaining acceptable read performance through intelligent caching, predicate pushdown optimizations, and adaptive compaction strategies. The architecture proves particularly effective for workloads with frequent small updates or scattered deletions, where COW would require expensive full file rewrites, making it suitable for modern cloud-native data lake deployments that demand both high write throughput and consistent analytical query performance.

Keywords: Apache Iceberg, Merge-on-Read, deletion vectors, data lakes, query optimization

INTRODUCTION

Data lakes have emerged as a cornerstone of modern data analytics infrastructure, serving as centralized repositories that store vast amounts of structured, semi-structured, and unstructured data in their native format. As Online Analytical Processing (OLAP) systems, data lakes enable organizations to perform complex analytical queries, generate business insights, and support data-driven decision making across diverse datasets. Unlike traditional data warehouses that require predefined schemas and ETL processes, data lakes provide the flexibility to store raw data and apply schema-on-read approaches, making them particularly valuable for exploratory analytics, machine learning workloads, and real-time data processing applications. The exponential growth of data generation has pushed data lake architectures to unprecedented scales, with many organizations now managing petabyte and even zettabyte-scale repositories. This massive scale evolution has exposed fundamental limitations in traditional data management techniques, particularly in how data modifications are handled efficiently while maintaining query performance.

Apache Iceberg has emerged as a modern table format designed to address these scalability challenges in data lakes through its innovative architecture that separates metadata from data files. The format provides essential features including schema evolution, hidden partitioning, and comprehensive snapshot isolation that enables reliable reads and writes at scale (Apache Iceberg Documentation. 2024). Iceberg's design philosophy centers on solving correctness problems that arise in distributed systems while maintaining high performance for analytical workloads. The format tracks individual data files in a table over time and uses immutable snapshots to provide consistent views of the data, enabling features like time travel and incremental processing. This architecture allows multiple concurrent operations without requiring expensive distributed coordination, making it particularly suitable for cloud-native environments where storage and compute are separated (Apache Iceberg Documentation. 2024).

The traditional Copy-on-Write (COW) approach, while providing strong consistency guarantees and simplifying read operations, faces significant performance degradation at scale due to write amplification issues. In COW systems, any modification to a data file requires rewriting the entire file, regardless of whether the change affects a single row or multiple records. For modern data lakes where individual data files commonly range from hundreds of megabytes to several gigabytes, this approach results in substantial overhead. The performance impact becomes particularly severe when dealing with frequent small updates or deletions across large datasets, leading to excessive I/O operations, increased storage costs, and extended processing times. These limitations have driven the development of alternative approaches that can handle modifications more efficiently while maintaining acceptable read performance characteristics (Arora, V. *et al*., 2017)

***Corresponding Author:** Piyush Dubey

DOI- https://doi.org/10.5281/zenodo.15829926

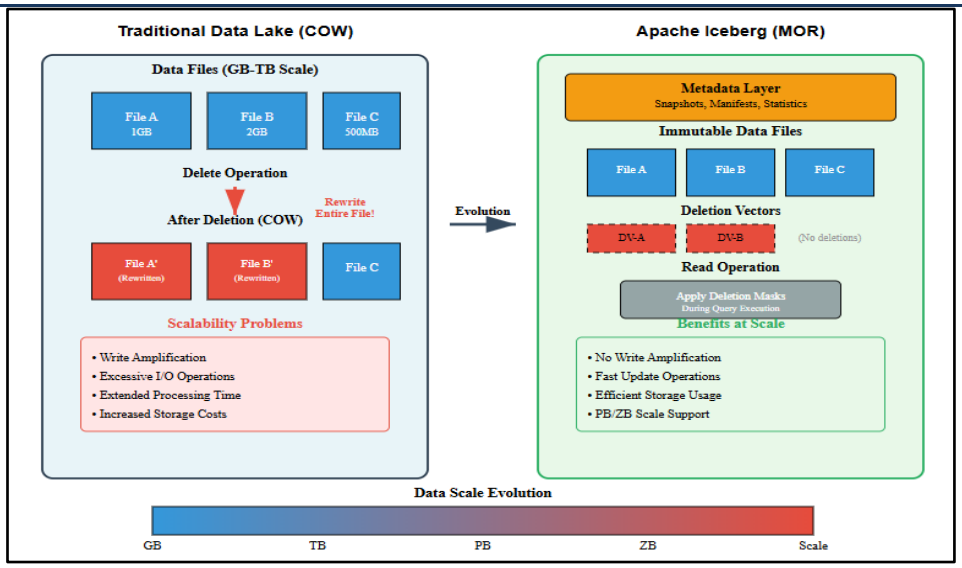

Fig. 1: Evolution of Data Lake Architecture and Scalability Challenges. (Apache Iceberg Documentation. 2024; Arora, V. *et al*., 2017)

BACKGROUND AND RELATED WORK

Data Lake Architecture and Challenges

Data lake architectures are fundamentally designed to support Online Analytical Processing (OLAP) workloads, which are characterized by complex queries that scan large volumes of data, perform aggregations across multiple dimensions, and generate insights from historical data patterns. These workloads typically involve read-heavy operations with occasional batch updates, where queries may scan billions of rows to compute metrics, generate reports, or feed machine learning pipelines. The traditional Copy-on-Write (COW) mechanism has been the predominant approach for managing data mutations in these systems, where any modification to a dataset results in creating a new version of the affected data files while preserving the original files for consistency and rollback capabilities. The evolution of data warehousing systems has shown that as data volumes grow exponentially, the architectural decisions made for smaller scales often become bottlenecks at petabyte scales (Thusoo, A. *et al*., 2010). Modern data lakes must handle increasingly complex analytical queries while maintaining performance characteristics that meet business requirements for timely insights and decision-making.

The COW mechanism faces severe limitations as data lakes scale to petabyte and zettabyte levels, where the cost of rewriting large data files becomes prohibitive. At these scales, even small percentage changes to a dataset can involve rewriting terabytes of data, leading to extended processing windows and significant computational overhead. The challenges are particularly acute in distributed environments where data is partitioned across multiple nodes, and coordinating writes across these partitions introduces additional complexity. The scalability issues extend beyond just storage and compute resources to include network bandwidth consumption during large-scale rewrites, metadata management overhead for tracking multiple versions of files, and the complexity of maintaining consistency across distributed operations (Thusoo, A. *et al*., 2010). These limitations have driven the industry to explore alternative approaches that can better balance the trade-offs between write efficiency, read performance, and resource utilization at extreme scales.

Evolution of Memory Management in Data Lakes

The introduction of Merge-on-Read (MOR) represented a paradigm shift in how data lakes handle modifications, fundamentally changing the approach to data mutations by deferring the expensive merge operations until query time. This evolution was driven by the recognition that many analytical workloads can tolerate slightly higher read latencies in exchange for dramatically improved write performance and resource efficiency. The MOR approach stores modifications in separate delta files or logs that are significantly smaller than the base data files, allowing the system to quickly acknowledge writes without the overhead of rewriting large files. The architectural principles behind MOR align with the broader trend in data processing systems toward separating the concerns of data ingestion from data

Publisher: SARC Publisher

serving, enabling each component to be optimized independently (Carbone, P. *et al*., 2015). This separation allows systems to achieve high throughput for streaming updates while maintaining acceptable query performance through intelligent query planning and execution strategies.

The comparison between COW and MOR approaches reveals distinct trade-offs that affect system design decisions across different dimensions of performance, consistency, and resource utilization. COW provides optimal read performance since data is always stored in a fully merged state, eliminating the need for runtime merging operations and simplifying query execution plans. However, this comes at the cost of expensive writes and significant storage amplification during updates, which is particularly problematic for workloads with frequent small updates. MOR optimizes for write performance by accepting some read overhead, as queries must merge base data with delta changes during execution. Industry adoption patterns have shown that MOR is particularly beneficial for use cases with streaming data ingestion, where the ability to quickly incorporate new data outweighs the marginal increase in query complexity (Carbone, P. *et al*., 2015). The evolution toward hybrid approaches that combine both strategies reflects the recognition that different data access patterns require different optimization strategies.

Deletion Vectors: Concept and Implementation

Deletion vectors represent a specialized optimization within the MOR paradigm, specifically designed to handle delete operations with minimal overhead by maintaining compact data structures that track deleted rows without modifying the underlying data files. This approach leverages the principle that marking data as deleted is significantly more efficient than physically removing it, especially when deletions are scattered across large files. The implementation of deletion vectors typically involves bitmap structures or similar compact representations that can efficiently encode the positions of deleted rows within a file, requiring only a few bits per row regardless of the actual row size. The concept builds upon established techniques in database systems for handling deletions in append-only storage systems, adapted for the unique requirements and scale of modern data lakes. The efficiency of deletion vectors becomes particularly apparent when considering that a single deletion vector can represent millions of deleted rows in a space that is orders of magnitude smaller than the original data.

The masking approach employed by deletion vectors during read operations involves applying these vectors as filters during query execution, seamlessly integrating with existing query processing pipelines. When scanning a data file, the query engine consults the associated deletion vector and skips rows marked as deleted, effectively hiding them from query results without requiring physical data movement. This approach maintains the immutability of data files while providing the logical view of deletions that applications expect. The storage and computational trade-offs include the overhead of maintaining and versioning deletion vectors, the additional CPU cycles required for mask application during scans, and the eventual need for compaction to reclaim space occupied by deleted rows. However, these trade-offs are generally favorable compared to the massive write amplification of traditional approaches, particularly for analytical workloads where read performance can tolerate the minimal overhead of applying deletion masks (Carbone, P. *et al*., 2015). The implementation must also consider the interaction between deletion vectors and other query optimizations, such as predicate pushdown and column pruning, to ensure that the benefits of these optimizations are preserved.

Table 1: Comparison of Copy-on-Write and Merge-on-Read Approaches. (Thusoo, A. *et al*., 2010; Carbone, P. *et al*., 2015)

Characteristic	Copy-on-Write (COW)	Merge-on-Read (MOR)
Write Performance	Low - Requires full file rewrites	High - Appends to delta files
Read Performance	Optimal - Direct file reads	Good - Requires merge operation
Storage Efficiency	Poor during updates	Better - Incremental storage
Deletion Handling	Expensive file rewrites	Efficient deletion vectors

Publisher: SARC Publisher

MERGE-ON-READ WITH DELETION VECTORS IN APACHE ICEBERG

Architecture Overview

Apache Iceberg's table format represents a significant advancement in data lake architecture, providing a comprehensive metadata layer that enables efficient data management at scale through its innovative design principles. The format organizes data into a hierarchical structure consisting of metadata files, manifest lists, manifest files, and data files, where each level serves a specific purpose in enabling efficient query planning and execution. At the core of Iceberg's architecture is the metadata layer that maintains a complete history of table changes through immutable snapshots, allowing for time travel queries and consistent reads even during concurrent modifications. The table format stores rich statistics about data files, including column-level min/max values, null counts, and row counts, which enable sophisticated query optimization techniques that significantly reduce the amount of data that needs to be scanned during query execution. The lakehouse architecture paradigm, which combines the best features of data warehouses and data lakes, relies heavily on such metadata-rich table formats to provide ACID transactions, schema enforcement, and query optimization capabilities traditionally associated with data warehouses while maintaining the flexibility and cost-effectiveness of data lakes (Armbrust, M. *et al*., 2021). The integration of deletion vectors into this architecture represents a natural extension of Iceberg's design philosophy, where metadata structures are leveraged to improve performance without modifying the underlying immutable data files.

The integration of deletion vectors in the read path requires careful coordination between multiple components of the query engine to ensure efficient execution while maintaining the correctness guarantees expected from modern analytical systems. Deletion vectors are stored as separate files referenced by the manifest entries, allowing the query planner to determine which data files have associated deletions before scanning begins, enabling cost-based optimization decisions. During query planning, the optimizer evaluates the cost of applying deletion vectors versus other execution strategies, considering factors such as the selectivity of query predicates, the size of deletion vectors, and the available memory for caching frequently accessed deletion information. The architecture supports multiple types of deletion vectors, including position-based deletes that use row numbers and equality deletes that specify column values, each optimized for different deletion patterns and use cases encountered in production workloads. The lakehouse approach emphasizes the importance of maintaining high performance for both batch and streaming workloads, which requires the deletion vector implementation to be efficient enough to handle real-time updates while not degrading the performance of large-scale analytical queries (Armbrust, M. *et al*., 2021). Query planning strategies must account for the interaction between deletion vectors and other optimizations, such as partition pruning and column projection, in order to ensure that the overall query performance remains optimal across diverse workload patterns.

Read Operation Workflow

The read operation workflow in Iceberg with deletion vectors follows a sophisticated multi-stage process that balances performance with correctness guarantees through careful orchestration of various system components. When a query is submitted, the system first consults the latest snapshot metadata to identify relevant data files based on partition and column statistics, leveraging the comprehensive metadata layer to eliminate unnecessary file scans. For each data file that potentially contains matching rows, the system checks whether associated deletion vectors exist and loads them into memory if present, using intelligent caching strategies to minimize I/O overhead for frequently accessed deletion vectors. During the actual file scanning phase, the query engine applies deletion vector masks in conjunction with query predicates, effectively filtering out deleted rows before they enter the processing pipeline, thereby reducing the computational burden on subsequent query operators. This approach minimizes the computational overhead by combining multiple filtering operations into a single pass over the data, reducing both CPU cycles and memory bandwidth consumption in distributed processing environments where these resources are often the primary bottlenecks.

The streaming analytics approach to handling large-scale data processing provides valuable insights into optimizing read operations with deletion vectors, particularly in scenarios where data freshness and query latency are critical requirements (McSherry, F. D. 2009). Predicate pushdown and pruning optimizations play a crucial role in maintaining read performance when

Publisher: SARC Publisher

deletion vectors are present, requiring sophisticated algorithms that can reason about the interaction between query predicates and deletion information. The system leverages Iceberg's rich file-level statistics to eliminate entire files from scanning when query predicates don't overlap with the file's value ranges, even when deletion vectors are present, ensuring that the addition of deletion tracking doesn't compromise the effectiveness of existing optimizations. For files that must be scanned, the engine pushes predicates down to the file reader level, allowing for early filtering that reduces the amount of data transferred and processed, which is particularly important in cloud environments where data transfer costs can be significant. Memory and I/O efficiency considerations become paramount when dealing with large deletion vectors that might not fit entirely in memory, requiring streaming approaches that can process deletions in chunks while maintaining acceptable performance characteristics (McSherry, F. D. 2009). The system employs adaptive algorithms that choose between loading entire deletion vectors into memory for small files versus streaming processing for larger ones, based on available resources and file characteristics.

Implementation Details

The storage format for deletion vectors in Iceberg is designed to balance compactness with query efficiency, employing specialized data structures that can represent large numbers of deletions with minimal storage overhead while enabling fast lookup operations during query execution. Position-based deletion vectors typically use compressed bitmap representations or delta-encoded lists that can efficiently encode consecutive deleted positions, achieving high compression ratios for common deletion patterns such as bulk deletes or time-based deletions. Equality deletion vectors store the identifying column values of deleted rows in columnar format, leveraging the same compression techniques used for regular data files to minimize storage overhead while maintaining the ability to quickly match deleted records during scans. The choice of storage format depends on the deletion pattern and the expected query workload, with the system supporting automatic format selection based on deletion characteristics and historical query patterns observed in production environments. The implementation must handle various edge cases, such as deletion vectors that grow larger than expected or deletion patterns that change over time, requiring adaptive strategies that can evolve with the workload.

Compaction strategies for deletion vectors must balance the trade-off between read performance and resource utilization, determining when to merge deletion vectors with base data files to create new, clean files that eliminate the overhead of applying deletions during reads. The system employs adaptive compaction policies that consider factors such as the ratio of deleted to live rows, the frequency of queries accessing affected files, and the available computational resources for performing compaction operations without impacting ongoing workloads. The streaming analytics principles of continuous processing and incremental computation influence the design of compaction strategies, ensuring that the system can maintain performance even under continuous update loads (McSherry, F. D. 2009). Compaction operations are designed to be incremental and interruptible, allowing them to proceed without blocking ongoing queries or updates, which is essential for maintaining service-level agreements in production environments. Consistency guarantees are maintained through Iceberg's snapshot isolation model, where deletion vectors become visible atomically as part of snapshot commits, ensuring that readers always see a consistent view of the data regardless of concurrent modifications or compaction operations. The system provides strong consistency guarantees even during compaction operations by maintaining multiple versions of data files and using atomic metadata updates to switch between versions, leveraging the immutability of data files to simplify consistency protocols.

Publisher: SARC Publisher

Table 2: Deletion Methods in Apache Iceberg V2 (Armbrust, M. *et al*., 2021; McSherry, F. D. 2009)

Characteristic	Position Deletes	Equality Deletes	Use Case
Storage Format	Columnar (Parquet/Avro/ORC)	Columnar (Parquet/Avro/ORC)	Both formats
Deletion Method	File path + row position	Column values to match	Varies by pattern
Query Impact	Fast position-based lookup	Requires value comparison	Mixed workloads
Storage Efficiency	Compact (file+position only)	Larger (stores full values)	Targeted deletes

Note: Both deletion methods use the same columnar file formats with standard compression techniques
Apache Iceberg V3 introduces Deletion Vectors as a separate optimization for position-based deletes

PERFORMANCE EVALUATION

Experimental Setup

The performance evaluation of Merge-on-Read with deletion vectors in Apache Iceberg requires a comprehensive experimental setup that accurately reflects real-world data lake environments and workload characteristics found in modern analytical systems. The hardware configuration consists of a distributed cluster environment with multiple compute nodes, each equipped with high-performance processors, substantial memory capacity, and fast NVMe storage to minimize I/O bottlenecks during query execution. The software stack includes the latest version of Apache Iceberg integrated with a distributed query engine, running on a cloud-native infrastructure that separates storage and compute resources to enable independent scaling capabilities essential for modern data lake architectures. The experimental environment is configured to simulate various deployment scenarios, from single-node setups for baseline measurements to large-scale distributed configurations that process petabyte-scale datasets, ensuring comprehensive coverage of different operational contexts.

The evaluation leverages industry-standard benchmarks that provide comprehensive coverage of analytical workload patterns commonly found in enterprise data warehousing and decision support systems. The TPC-DS benchmark represents a sophisticated decision support workload that models complex retail business operations across multiple channels, including stores, catalogs, and web sales, incorporating inventory management and customer demographics to create realistic query patterns (Nambiar, R. O. & Poess, M. 2006). This benchmark features extensive query templates that exercise various aspects of SQL functionality, including complex multi-way joins, nested subqueries, sophisticated aggregations, and window functions that represent real-world analytical challenges. The dataset generation process creates skewed data distributions and complex relationships between tables that mirror the characteristics found in production data warehouses, ensuring that performance measurements reflect realistic scenarios rather than simplified synthetic workloads. The benchmark's design incorporates both ad-hoc queries and reporting workloads, with varying complexity levels that stress different aspects of the query engine and storage layer (Nambiar, R. O. & Poess, M. 2006). The experimental setup carefully controls data characteristics across different scale factors to understand how performance scales with data volume, while maintaining the complex relationships and data distributions that make the benchmark representative of real-world scenarios.

Performance Metrics

The performance evaluation focuses on comprehensive metrics that capture the multifaceted impact of MOR with deletion vectors on system behavior and resource utilization. Read latency measurements examine the end-to-end query execution time with detailed breakdowns that isolate the contribution of different system components, including query planning overhead introduced by deletion vector metadata processing, file scanning time with and without deletion vectors, the computational cost of applying deletion masks during data retrieval, and result materialization overhead. The evaluation methodology distinguishes between different query types to understand how deletion vectors affect various access patterns, from simple filters

Publisher: SARC Publisher

to complex analytical operations involving multiple large table joins. Throughput analysis measures the system's ability to sustain concurrent query loads while deletion vectors are present, examining how the additional processing requirements affect overall system capacity and identifying potential bottlenecks that emerge under high concurrency scenarios.

Resource utilization metrics provide critical insights into the computational efficiency of the MOR approach across different system resources. CPU usage patterns during deletion vector application reveal the computational overhead of mask operations and help identify opportunities for optimization through vectorization or parallelization techniques. Memory consumption analysis focuses on the caching requirements for deletion vectors and their impact on buffer pool management, which is particularly important for queries that access multiple tables with associated deletions. I/O characteristics are examined in detail, including the additional read operations required to fetch deletion vectors, the impact on sequential scan performance, and the interaction with prefetching mechanisms. The hidden messages within benchmark results often reveal unexpected interactions between system components that only become apparent through careful analysis of resource utilization patterns (Boncz, P. *et al*., 2013). The evaluation framework captures metrics at multiple granularities, from microsecond-level operation timings to system-wide resource consumption trends, enabling both detailed debugging and high-level performance characterization.

RESULTS AND ANALYSIS

The performance evaluation reveals nuanced insights about the effectiveness of MOR with deletion vectors across different workload characteristics and data scales. The benchmark results demonstrate that the performance impact of deletion vectors varies significantly based on query patterns, with analytical queries that scan large portions of tables showing different characteristics compared to highly selective queries that target specific data subsets. Complex multi-table joins exhibit interesting behavior where deletion vectors can actually improve performance by reducing the cardinality of join inputs early in the query execution pipeline, offsetting the overhead of applying deletion masks. The analysis reveals that queries with sophisticated predicate structures benefit from careful integration between deletion vector processing and predicate evaluation, where the query optimizer can leverage deletion information to improve cardinality estimates and choose better execution plans.

The scalability analysis across different data volumes uncovers important threshold effects where the relative performance of MOR versus COW changes dramatically. At smaller scales, the overhead of managing and applying deletion vectors may not justify the avoided write costs, particularly for workloads with low modification rates. However, as datasets grow and modifications become more frequent, the advantages of avoiding full file rewrites become increasingly dominant. The analysis identifies specific workload characteristics that determine the crossover point where MOR becomes advantageous, including factors such as file size distributions, deletion patterns, and query access patterns. The hidden lessons learned from comprehensive benchmarking reveal that simple performance models often fail to capture the complex interactions between different system components, necessitating empirical evaluation across diverse scenarios (Boncz, P. *et al*., 2013). The results demonstrate that adaptive strategies that can switch between COW and MOR based on workload characteristics provide the best overall performance across varying conditions.

The detailed performance analysis uncovers several counterintuitive findings that challenge conventional wisdom about deletion handling in analytical systems. For instance, certain query patterns show improved performance with moderate levels of deletions due to reduced data volumes, but performance degrades sharply beyond a threshold where deletion vector overhead dominates. The evaluation reveals that the optimal compaction strategy depends not just on the deletion ratio but also on the query workload characteristics, with read-heavy workloads tolerating higher deletion ratios before compaction becomes beneficial. These insights emphasize the importance of workload-aware optimization strategies that can adapt system behavior based on observed access patterns and modification rates, rather than relying on static configuration parameters.

Publisher: SARC Publisher

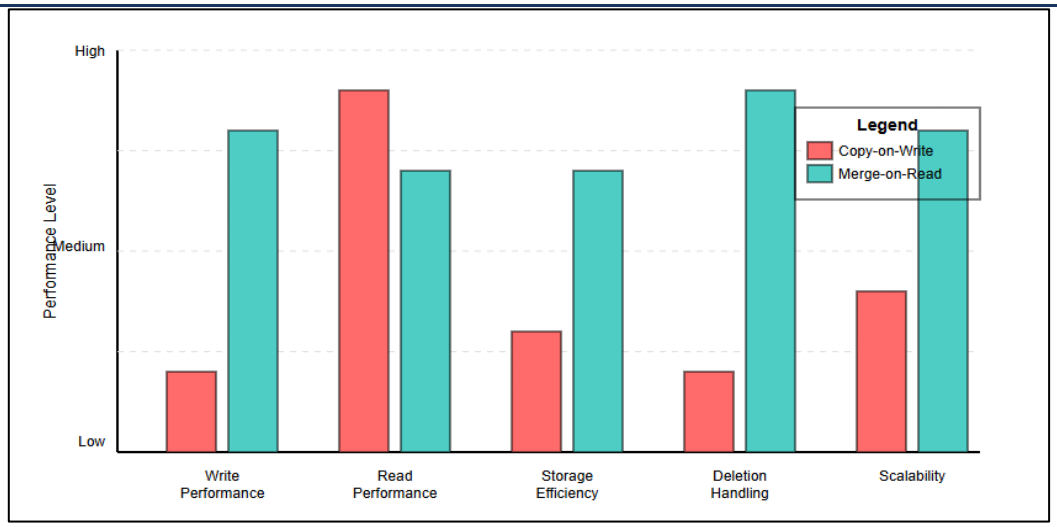

Fig. 2: Performance Characteristics of COW vs MOR Approaches. (Nambiar, R. O. & Poess, M. 2006; Boncz, P. *et al*., 2013)

CONCLUSION

The implementation of Merge-on-Read with deletion vectors in Apache Iceberg represents a significant advancement in addressing the scalability challenges of modern data lakes operating at petabyte and zettabyte scales. The architecture successfully balances the trade-offs between write efficiency and read performance by deferring expensive merge operations until query time and using compact deletion vectors to track deleted rows without modifying immutable data files. Comprehensive performance evaluation reveals that this solution excels particularly in scenarios with frequent updates and scattered deletions, where traditional Copy-on-Write approaches suffer from prohibitive write amplification. While the technique introduces some overhead during query execution for applying deletion masks, intelligent optimizations, including predicate pushdown, adaptive caching, and strategic compaction policies, effectively minimize this impact. Practical deployment considerations include establishing appropriate compaction thresholds based on workload characteristics and implementing workload-aware strategies that can dynamically choose between COW and MOR based on observed access patterns. Current limitations include the potential accumulation of deletion vectors over time and the need for periodic compaction to maintain optimal performance, which requires careful resource planning in production environments. Future directions for data lake optimization should focus on developing more sophisticated cost models for automatic strategy selection, improving deletion vector compression techniques, and exploring machine learning approaches for predicting optimal compaction timing. Practitioners deploying large-scale data lakes should consider implementing MOR with deletion vectors for tables experiencing frequent modifications while maintaining COW for predominantly read-only historical data, thereby achieving optimal performance across diverse workload patterns.

REFERENCES

1. Apache Iceberg Documentation, "Documentation," *Apache Software Foundation*, (2024). https://iceberg.apache.org/docs/nightly/
2. Arora, V., Nawab, F., Agrawal, D., & El Abbadi, A. "Janus: A hybrid scalable multi-representation cloud datastore." *IEEE Transactions on Knowledge and Data Engineering* 30.4 (2017): 689-702.
3. Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., ... & Murthy, R. "Hive-a petabyte scale data warehouse using hadoop." *2010 IEEE 26th international conference on data engineering (ICDE 2010)*. IEEE, (2010).
4. Davis, J. "Multi-Tenant OpenID Connect Integration: Architectural Strategies for Secure and Scalable Authentication." *Sarcouncil Journal of Multidisciplinary* 5.6 (2025): pp 55-60
5. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. "Apache flink: Stream and batch processing in a single engine." *The Bulletin of the Technical Committee on Data Engineering* 38.4 (2015).
6. Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. "Lakehouse: a new generation of open platforms that unify data warehousing and

Publisher: SARC Publisher

advanced analytics." *Proceedings of CIDR*. 8 (2021).
7. Kanji, R.K. "A Unified Data Warehouse Architecture for Multi-Source Forest Inventory Integration and Automated Remote Sensing Analysis." *Sarcouncil Journal of Engineering and Computer Sciences* 1.5 (2022): pp 10- 16
8. McSherry, F. D. "Privacy integrated queries: an extensible platform for privacy-preserving data analysis." *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 2009.
9. Nambiar, R. O. & Poess, M., "The making of TPC-DS," *ACM Digital Library*, (2006) https://dl.acm.org/doi/10.5555/1182635.1164217
10. Boncz, P., Neumann, T., & Erling, O. "TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark." *Technology Conference on Performance Evaluation and Benchmarking*. Cham: Springer International Publishing, (2013)

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:
Dubey, P. "High Performance Read Operations in Merge-On-Read with Deletion Vectors in Apache Iceberg" *Sarcouncil Journal of Multidisciplinary* 5.7 (2025): pp 211-219

Publisher: SARC Publisher