Sarcouncil Journal of Engineering and Computer Sciences

ISSN(Online): 2945-3585

Volume- 04| Issue- 07| 2025

Research Article **Received:** 01-06-2025| **Accepted:** 17-06-2025 | **Published:** 06-07-2025

High-Performance Copy-On-Write for ACID Transactions in Parquet-Based Data Lakehouse

Piyush Dubey
University of Iowa, USA

Abstract: The widespread adoption of data lakehouse architectures has increased the demand for efficient ACID-compliant operations at scale. Traditional copy-on-write techniques—commonly employed in systems like Apache Hudi and Delta Lake—incur significant performance overhead by requiring entire files to be rewritten even for minimal row-level changes. This article presents an optimized partial copy-on-write mechanism integrated directly into the Apache Parquet file format, specifically engineered to support selective data updates. The solution introduces a fine-grained secondary row index that enables efficient localization of modified records to individual data pages, significantly minimizing I/O and computational costs during upserts. Through empirical benchmarking on production-scale datasets, the partial copy-on-write implementation demonstrates substantial improvement in write throughput compared to conventional strategies. The presented architecture maintains full ACID compliance while addressing performance bottlenecks in update-intensive workloads, enhancing the practical viability of data lakehouses for high-velocity, real-time analytics across distributed environments.

Keywords: Data lakehouse, ACID transactions, Parquet optimization, page-level modification, transaction isolation.

INTRODUCTION

Data lakehouse architectures have emerged as a transformative paradigm in modern data management, combining the flexibility and cost-effectiveness of data lakes with the reliability and performance characteristics of traditional data warehouses. These architectures support both structured and unstructured data while providing transaction management capabilities previously confined to conventional database systems (Behm, A. *et al.*, 2022). The increasing adoption of these hybrid systems stems from their ability to unify disparate data workloads—including machine learning, SQL analytics, and streaming—within a cohesive framework that eliminates data silos and reduces operational complexity.

ACID transactions represent a foundational element in data lakehouse environments, enabling consistent data modifications across distributed storage systems. Unlike traditional data lakes that primarily offer append-only operations, modern lakehouses must support sophisticated update patterns while maintaining transactional guarantees. This capability becomes particularly crucial when implementing data correction workflows, complying with regulatory requirements, or supporting real-time analytics on continuously updated datasets (Xinli Shang *et al.*, 2023). The implementation of these transactional capabilities directly influences both the operational efficiency and analytical value derived from lakehouse architectures.

Traditional copy-on-write (CoW) approaches employed in current lakehouse systems encounter significant performance limitations when handling frequent updates to large datasets. The underlying challenge stems from the immutable nature of file formats like Apache Parquet, which necessitates creating entirely new files for even minor data modifications (Xinli Shang *et al.*, 2023). This inefficiency becomes particularly pronounced when modifying selective rows within multi-gigabyte files, requiring substantial computational resources and I/O operations that scale linearly with file size rather than with the volume of modified data.

The proposed partial copy-on-write mechanism addresses these challenges through fine-grained page-level modifications within the Parquet format. By introducing a secondary row indexing structure, the system can precisely locate records at the data page level, allowing for selective rewriting of only modified portions (Xinli Shang *et al.*, 2023). This approach substantially reduces unnecessary data movement during update operations while maintaining the compression and columnar benefits of the Parquet format. The targeted modification strategy represents a fundamental shift from file-level to page-level granularity in managing data updates within lakehouse systems.

This research contributes several innovations to lakehouse architectures: a secondary row indexing mechanism for efficient record location; algorithms for selective page-level updates with

***Corresponding Author:** Piyush Dubey

DOI- https://doi.org/10.5281/zenodo.15818216

minimal I/O operations; techniques for maintaining ACID guarantees across distributed systems; and empirical evaluation demonstrating significant improvement in write throughput compared to conventional approaches (Behm, A. *et al.,* 2022; Xinli Shang *et al.,* 2023). These advancements collectively enhance the viability of data lakehouses for high-velocity analytics scenarios previously considered unsuitable for lake-based architectures.

Evolution of Data Lake to Data Lakehouse Architectures

The data management landscape has evolved significantly from traditional data warehouses to data lakes and now to data lakehouse architectures. Data lakes emerged to overcome the rigidity of conventional warehouses, offering storage for diverse data types in their native formats. However, these lakes typically lacked the structured data management capabilities necessary for enterprise analytics workloads. Data lakehouse architectures address this limitation by implementing a metadata layer above low-cost object storage, thereby providing database-like functionality without sacrificing the flexibility of data lakes (Armbrust, M. *et al.,* 2020). This architectural approach introduces a transaction protocol that maintains ACID properties while operating on cloud storage systems originally designed for immutable objects. The protocol leverages atomic operations available in these storage services to implement optimistic concurrency control mechanisms that enable multiple concurrent readers and writers to operate on the same dataset with appropriate isolation guarantees.

Table 1: Comparison of Data Management Architectures

Architecture	Data Format Flexibility	Query Performance	Transaction Support	Schema Enforcement
Data Warehouse	Low	High	Full ACID	Strict
Data Lake	High	Low	Limited/None	Optional
Data Lakehouse	High	High	Full ACID	Flexible

.

Existing Approaches to ACID Transactions in Distributed Data Systems

ACID transactions in distributed environments present unique challenges that have been addressed through various technical approaches. Modern data lakehouse systems typically implement snapshot isolation semantics, which allow read operations to execute on a consistent view of data without blocking concurrent writers. This isolation level is achieved by maintaining multiple versions of data and directing read operations to appropriate versions based on transaction timestamps. The transaction management layer utilizes a combination of optimistic concurrency control and metadata-based versioning to coordinate operations across distributed storage systems (Armbrust, M. *et al.,* 2020). These systems employ techniques such as conflict detection during the commit phase to identify and resolve write-write conflicts, thereby ensuring data consistency despite concurrent modifications. The implementation typically involves a centralized transaction coordinator that validates changes before committing them to the shared metadata repository.

Overview of Apache Parquet File Format and Internal Structure

Apache Parquet serves as a cornerstone file format in modern data lake architectures due to its columnar organization and efficient compression capabilities. Unlike row-oriented formats, Parquet organizes data by column, grouping similar values together to achieve higher compression ratios and enable predicate pushdown during query execution. The internal structure consists of file metadata, row groups, column chunks, and data pages arranged in a hierarchical format (Alsubaiee, S. *et al.,* 2014). The columnar layout allows for efficient encoding schemes tailored to specific data types, such as dictionary encoding for low-cardinality columns and run-length encoding for sequential values. This structure supports partial file reads where only the required columns are accessed, substantially reducing I/O requirements for analytical queries that typically reference a subset of columns.

Publisher: SARC Publisher

Table 2: Parquet Internal Structure Hierarchy.(Alsubaiee, S. *et al.*, 2014)

Component	Description	Contains	Size Characteristics
File	Complete dataset	File metadata, Row groups	Variable, typically MB-GB
Row Group	Horizontal partition	Column chunks	Configurable, typically 128MB
Column Chunk	Columnar data section	Data pages	Variable based on column data
Data Page	Basic storage unit	Encoded column values	Typically 1MB or less

Current Implementations of Copy-on-Write in Apache Hudi and Delta Lake

Modern lakehouse frameworks provide transactional capabilities on immutable storage through metadata-based versioning systems. These frameworks implement copy-on-write approaches where modifications to existing data result in new file creation rather than in-place updates. When processing updates, the system identifies affected files, reads their contents entirely, applies the changes in memory, and writes complete new versions (Armbrust, M. *et al.*, 2020). The transaction layer then atomically updates metadata to reference these new files. This approach preserves immutability at the storage level while enabling logical data modifications through metadata updates. While effective for maintaining consistency, this method requires substantial I/O operations proportional to file size rather than to the volume of modified data, creating performance challenges for update-intensive workloads in large-scale environments.

Limitations of Conventional Copy-on-Write Strategies

Conventional copy-on-write strategies encounter significant limitations in scenarios involving frequent updates to large datasets. The fundamental inefficiency stems from the granularity of modifications—entire files must be rewritten even when changes affect only a small subset of records. For instance, modifying a single record in a large file requires reading, deserializing, updating, and serializing the entire file, regardless of the actual modification scope (Alsubaiee, S. *et al.*, 2014). This approach results in excessive I/O and computational overhead that scales with file size rather than with the magnitude of changes. The performance degradation becomes particularly pronounced in operational data management scenarios where high-frequency updates must be processed with minimal latency. Additionally, this inefficiency leads to temporary storage bloat during the transition period when both old and new versions must be maintained. These limitations fundamentally constrain the practical applicability of data lakehouse architectures for use cases requiring both transactional consistency and high-performance write operations, highlighting the need for more granular modification strategies.

SYSTEM ARCHITECTURE

Design Principles for Partial Copy-on-Write in Parquet

The proposed system architecture introduces a partial copy-on-write (PCow) mechanism that fundamentally transforms how modifications are handled in Parquet-based data lakehouses. This approach is governed by several interdependent design principles that collectively prioritize performance optimization while maintaining backward compatibility. The architecture operates on the premise that minimizing data movement is essential for efficient update operations, particularly in distributed environments where network transfer constitutes a significant performance bottleneck (Zaharia, M. *et al.*, 2018). By targeting modifications at the page level rather than the file level, the system dramatically reduces the volume of data that must be processed during update operations. This granular approach preserves the inherent advantages of columnar storage—including efficient compression and query performance—while introducing selective mutability that enables efficient updates. The implementation extends rather than replaces the existing Parquet format, introducing supplementary metadata structures that enable precise record localization while remaining fully compatible with legacy readers and processing engines (Zaharia, M. *et al.*, 2018). This ensures that organizations can adopt the PCow mechanism incrementally without disrupting existing workflows or requiring simultaneous upgrades across their entire data processing infrastructure.

Publisher: SARC Publisher

Fine-grained Secondary Row Indexing Mechanism

The cornerstone of the partial copy-on-write architecture is a fine-grained secondary row indexing mechanism that establishes explicit mappings between logical record identifiers and their physical storage locations within Parquet files. Unlike traditional Parquet implementations that require sequential scanning of data pages to locate specific records, the secondary index enables direct access to the pages containing target records (Shen, M. *et al.,* 2020). This indexing structure introduces a hierarchical organization that maps row identifiers to specific row groups, column chunks, and ultimately to individual data pages within those chunks. The index is constructed during file writing and exists as an optional metadata section within the Parquet file footer, allowing PCow-aware systems to leverage it while remaining invisible to legacy readers. The implementation utilizes a compact encoding scheme that minimizes storage overhead through techniques such as run-length encoding for sequential row ranges and delta encoding for page boundaries (Shen, M. *et al.,* 2020). When processing update operations, the system consults this index to precisely identify which pages require modification, thereby eliminating unnecessary I/O operations and computational overhead associated with processing unaffected pages. This targeted approach forms the foundation for efficient in-place-like modifications within an otherwise immutable file format, striking a balance between performance and consistency guarantees.

Page-level Modification Tracking

The system implements sophisticated page-level modification tracking that builds upon the secondary row indexing mechanism to manage changes with minimal overhead. When update operations are initiated, the architecture establishes a transaction context that utilizes the secondary index to identify affected pages across all relevant column chunks (Zaharia, M. *et al.,* 2018). This context maintains a precise record of which pages require modification, utilizing a bitmap-based representation that minimizes memory usage while supporting efficient operations. As records are modified, the tracking mechanism maintains associations between original and modified versions of affected pages, creating a transient dual view of the data during the update process. This approach allows the system to implement optimized read-modify-write cycles that operate exclusively on affected pages rather than entire files. For operations that modify existing records, only the specific pages containing those records are read, deserialized, updated, and rewritten (Zaharia, M. *et al.,* 2018). Unmodified pages are efficiently transferred from the original file to the new version without incurring the computational cost of deserialization and reserialization, substantially reducing CPU utilization. The tracking mechanism also supports intelligent coalescing of modifications that affect the same pages, minimizing redundant operations when multiple records within a single page are modified as part of the same transaction.

Transaction Isolation and Consistency Guarantees

The architecture maintains robust transaction semantics through a carefully designed concurrency control model that ensures consistency without compromising performance. The system implements snapshot isolation, wherein each transaction operates on a consistent view of the database as it existed at the beginning of the transaction (Shen, M. *et al.,* 2020). This isolation level prevents common anomalies such as dirty reads and non-repeatable reads while allowing for high concurrency between read and write operations. The implementation leverages a multi-version approach where each update operation generates a new version of the affected files with appropriate metadata to track version lineage. The versioning system enables concurrent readers to continue accessing consistent snapshots without blocking, even as writers generate new versions. For conflict detection, the architecture employs optimistic concurrency control with validation during the commit phase (Shen, M. *et al.,* 2020). When a transaction attempts to commit, the system verifies that the base versions of modified files have not been concurrently updated by other transactions. This approach minimizes coordination overhead in distributed environments while ensuring data consistency across concurrent operations. The architecture also supports configurable conflict resolution policies that determine how to proceed when conflicts are detected, ranging from automatic retries with updated base versions to explicit error reporting that requires application-level resolution.

Integration Points with Existing Lakehouse Frameworks

The partial copy-on-write architecture is designed with integration as a primary consideration, providing well-defined extension points for existing lakehouse frameworks. The system

Publisher: SARC Publisher

exposes both low-level APIs for direct interaction with PCow-enabled Parquet files and high-level adaptation layers for integration with established frameworks (Zaharia, M. *et al.*, 2018). These integration components enable existing systems to leverage page-level modifications without requiring fundamental architectural changes. For metadata management, the architecture generates detailed modification records that can be incorporated into transaction logs maintained by lakehouse frameworks. These records include information about modified pages, their version lineage, and associated transaction contexts, enabling consistent snapshot management and time travel capabilities (Shen, M. *et al.*, 2020). From a query processing perspective, the system provides optimization hints that allow query engines to leverage the secondary row index for efficient predicate pushdown and data filtering. This integration enables query planners to generate execution plans that minimize data movement when processing filtered datasets. The architecture also includes specialized components for distributed processing environments, such as worker coordination mechanisms that optimize the distribution of page-level modifications across compute nodes (Shen, M. *et al.*, 2020). These components ensure that the performance benefits of partial copy-on-write operations are preserved even in highly distributed execution contexts, making the architecture suitable for deployment in large-scale data processing environments with diverse workload characteristics.

IMPLEMENTATION

Technical Details of the Secondary Row Index Structure

The secondary row index structure serves as the fundamental enabler for efficient record localization within Parquet files, dramatically reducing the I/O overhead associated with update operations. This index is implemented as a hierarchical mapping framework that establishes direct relationships between logical record identifiers and their physical storage locations within the file structure. The architecture employs a multi-level organization that first maps row ranges to specific row groups, then further refines this mapping to identify individual data pages within column chunks that contain target records (Melnik, S. *et al.*, 2010). This approach leverages concepts similar to those found in columnar database systems, where efficient record positioning is crucial for query performance. The index utilizes a specialized binary encoding format that minimizes storage overhead through techniques such as delta encoding for sequential row identifiers and variable-length integer encoding for byte offsets. This encoding strategy substantially reduces the memory footprint compared to naive implementations while maintaining lookup performance (Melnik, S. *et al.*, 2010). The search algorithm operates in logarithmic time complexity, executing a two-phase approach that first identifies the relevant row group through binary search on group boundaries, then locates specific pages through a secondary binary search within the identified group. This design ensures consistent performance regardless of file size or record distribution patterns. The index structure is serialized as an optional metadata section within the Parquet file footer, maintaining complete backward compatibility with standard Parquet readers that can simply ignore this additional metadata component during normal processing operations.

Algorithms for Efficient Page-Level Data Updates

The implementation introduces sophisticated algorithms specifically engineered for page-level data modifications that minimize unnecessary I/O operations. The core update algorithm follows a targeted read-modify-write pattern operating at page granularity rather than file granularity, representing a fundamental shift from traditional copy-on-write approaches (Shaikhha, A. *et al.*, 2018). When processing update requests, the system leverages the secondary row index to precisely identify pages containing records that require modification. It then selectively reads only those specific pages, applies the requested changes in memory, and writes new versions while preserving unmodified content from the original file. This selective approach dramatically reduces both I/O and processing overhead compared to file-level rewrites. For point updates modifying individual records, the algorithm first locates target pages using the secondary index, then reads and deserializes only those pages, applies modifications to the in-memory representation, recompresses the modified content, and finally generates a new file that seamlessly combines modified pages with unmodified content (Shaikhha, A. *et al.*, 2018). The delete operation variant maintains special deletion markers rather than physically removing records, preserving positional integrity of surrounding data and simplifying subsequent processing. The insert operation algorithm identifies appropriate insertion

Publisher: SARC Publisher

points and either integrates new records into existing pages or creates additional pages as necessary, with specialized handling for page splitting when insertions would exceed configured size thresholds. These algorithms collectively enable a fine-grained modification approach that maintains the performance benefits of immutable storage while providing efficient update capabilities.

Metadata Management for Tracking Modified Pages

The implementation incorporates comprehensive metadata management facilities that maintain detailed information about modified pages throughout their lifecycle. This metadata framework records multi-dimensional attributes about each modification, enabling consistent versioning and efficient query processing across evolving datasets (Melnik, S. *et al.,* 2010). The system maintains a modification history within the file structure that documents which pages have been modified, when modifications occurred, and which transactions performed the changes. This history is encoded in a compact binary format within the file footer, enabling efficient parsing during read operations while minimizing storage overhead. For each modified page, the metadata captures essential attributes including the original and new page versions, respective file offsets, modification type classifications, change summaries, and transaction references (Melnik, S. *et al.,* 2010). This detailed tracking enables sophisticated optimization techniques during query execution, such as predicate-based page pruning that allows the query engine to skip pages that cannot contain relevant data based on their modification history. The metadata system exposes programmatic interfaces for integration with higher-level transaction coordinators, enabling consistent version management across distributed environments where multiple processing nodes may concurrently operate on the same logical dataset. These interfaces provide methods for version interrogation, modification tracking, and consistent metadata updates that maintain referential integrity across distributed transaction boundaries.

Handling of Concurrent Transactions

The implementation addresses concurrent transaction management through a multi-version concurrency control approach that maximizes throughput while maintaining appropriate isolation guarantees. The system treats each file version as an immutable snapshot, enabling multiple readers to access different versions concurrently without interference or blocking (Shaikhha, A. *et al.,* 2018). This design eliminates read-write contention that typically limits scalability in traditional database systems. For write operations, the implementation employs optimistic concurrency control strategies that record base file versions at transaction initiation and later verify these versions during commit processing. This approach minimizes coordination overhead in distributed environments while detecting potential conflicts between concurrent modifications (Shaikhha, A. *et al.,* 2018). The system supports configurable conflict resolution policies that determine appropriate responses when version conflicts are detected, ranging from automatic transaction retries with updated base versions to explicit error reporting for application-level resolution. To minimize contention in high-concurrency environments, the implementation provides a lock-free reader design that eliminates coordination overhead for read-only transactions, page-level locking granularity for write operations that enables concurrent modifications to different portions of the same logical file, deadlock detection mechanisms for complex multi-file transactions, and operation batching facilities that reduce commit overhead by consolidating compatible modifications.

Optimizations for Different Update Patterns

The implementation incorporates specialized optimizations for diverse update patterns, recognizing that real-world workloads exhibit varying characteristics that benefit from tailored processing approaches. For point updates affecting individual records dispersed throughout a dataset, the system employs a sparse update strategy that leverages the secondary row index to pinpoint target locations with minimal overhead (Melnik, S. *et al.,* 2010). This approach is particularly effective for incremental data correction scenarios where modifications affect a small percentage of records distributed across multiple files. For batch updates modifying contiguous record ranges, the implementation switches to a range-based processing mode optimized for sequential I/O patterns, identifying minimal page sets covering target ranges and processing them as cohesive units with prefetching and pipelining optimizations (Melnik, S. *et al.,* 2010). The mixed workload handling incorporates a query planning component that analyzes operation distribution patterns and selects optimal execution strategies based on record density, operation type groupings, and

Publisher: SARC Publisher

available system resources. The implementation also includes adaptive optimization techniques that monitor runtime workload characteristics and dynamically adjust processing strategies based on observed patterns, switching between different buffering and execution approaches as optimal for current conditions without requiring manual intervention (Shaikhha, A. *et al.*, 2018). Additional specialized optimizations target common lakehouse scenarios including time-partitioned data processing with partition-aware metadata caching, dimension table updates with specialized handling for high-frequency modifications to reference datasets, and schema evolution support with coordinated mechanisms that maintain consistency between concurrent data and metadata modifications across distributed processing environments.

EXPERIMENTAL EVALUATION

Benchmark Methodology and Experimental Setup

To evaluate the effectiveness of the partial copy-on-write approach, conducted comprehensive experiments using both synthetic benchmarks and production workloads in a distributed environment. The experimental infrastructure consisted of a multi-node cluster configured with uniform hardware specifications to ensure measurement consistency, operating on a distributed file system that provided the underlying storage for all benchmark datasets. This configuration reflects typical production environments where data lakehouse architectures are deployed, allowing for realistic performance assessment under representative conditions (Vo, H. T. *et al.*, 2012). The evaluation methodology encompassed multiple performance dimensions, including operation throughput, response latency, and resource utilization across processing and storage components. Throughput measurements recorded operation processing rates under various workload conditions, while latency measurements captured end-to-end operation completion time from submission to confirmation. Comprehensive resource utilization metrics monitored system consumption patterns throughout the experiments, providing a holistic efficiency view that extended beyond simple timing measurements. The benchmark suite incorporated diverse synthetic workloads specifically engineered to isolate performance characteristics across different dimensions, varying operation type distribution, access patterns, concurrency levels, and dataset characteristics. Each configuration executed through multiple iterations with appropriate warm-up phases to ensure statistical significance and eliminate transient effects from measurements. This methodology follows established evaluation practices for distributed data systems that emphasize reproducibility and comparative analysis, as detailed in recent literature on log-structured database evaluation frameworks (Vo, H. T. *et al.*, 2012). The experimental protocols incorporated controls for environmental variables such as background processes and network conditions to minimize external factors that might otherwise confound comparative analysis between different implementation approaches.

Performance Comparison with Traditional Copy-on-Write Approaches

The experimental results demonstrate substantial performance advantages compared to traditional copy-on-write implementations found in existing data lakehouse systems. For comprehensive comparison, identical workloads were implemented and executed on both the partial copy-on-write prototype and reference implementations of conventional file-level mechanisms similar to those in widely-deployed open-source frameworks. The most significant performance differentials emerged in workloads characterized by sparse point updates affecting a small percentage of records within large files—a common pattern in many analytical environments where incremental corrections or enrichments are applied to historical datasets (Armbrust, M. *et al.*, 2021). In these scenarios, the page-level approach consistently delivered throughput improvements that scaled with file size, demonstrating the fundamental efficiency advantage of targeted modifications over whole-file rewrites. The latency characteristics revealed even more dramatic differences, with the partial copy-on-write implementation maintaining near-constant response times regardless of underlying file size, while traditional approaches exhibited latency that increased proportionally with file dimensions. This behavior confirms the theoretical analysis that page-level modifications effectively decouple operation response time from total file size, a critical factor for maintaining consistent performance as datasets grow over time (Armbrust, M. *et al.*, 2021). Mixed workloads combining read operations with different write patterns also showed substantial improvements, with particularly significant gains observed in environments where analytical queries execute concurrently with modification operations. The

Publisher: SARC Publisher

partial copy-on-write approach minimized interference between these concurrent activities through its efficient versioning mechanism and reduced resource contention, enabling higher overall system throughput compared to traditional implementations where modifications would block or significantly impact concurrent reads of the same datasets.

Table 3: I/O Operation Comparison Between CoW Approaches. (Vo, H. T. *et al.*, 2012; Armbrust, M. *et al.*, 2021)

Operation Type	Traditional CoW	Partial CoW	Performance Advantage Factor
Point Update (Single Record)	Reads entire file, writes entire file	Reads affected pages only, writes affected pages only	Proportional to file-size ratio
Batch Update (Contiguous)	Reads entire file, writes entire file	Reads affected page range, writes affected page range	Proportional to percentage of modified pages
Mixed Read-Write	High interference between read/write	Low interference through versioning	Workload-dependent
Delete Operation	Rewrites entire file	Updates metadata, marks deleted pages	Proportional to file size

Scalability Analysis with Varying Dataset Sizes and Update Patterns

The scalability characteristics of the implementation were assessed through a progressive series of experiments with increasing dataset dimensions and diverse update patterns. Dataset sizes scaled from modest to extremely large volumes, with file counts adjusted accordingly to maintain realistic file size distributions representative of production environments (Vo, H. T. *et al.*, 2012). This methodical scaling approach revealed how performance characteristics evolved as datasets grew beyond system memory capacity, forcing reliance on storage I/O for data access—a critical transition point for many data-intensive applications. The results demonstrated excellent scalability properties for the partial copy-on-write approach across multiple dimensions. For point update workloads, throughput remained remarkably stable as dataset size increased, with only modest degradation observed at the largest scales tested. This performance stability contrasts sharply with traditional copy-on-write implementations, which showed throughput declining as dataset size increased due to their linearly growing I/O requirements for modification operations. The implementation also demonstrated strong scalability across different update distribution patterns, maintaining consistent performance whether modifications were concentrated within a few files or dispersed across many files (Vo, H. T. *et al.*, 2012). Concurrency scaling tests revealed effective utilization of available system resources as parallelism increased, with throughput scaling efficiently with additional processing nodes until reaching hardware I/O boundaries. The optimistic concurrency control mechanism showed minimal contention overhead except under extreme scenarios with highly conflicting update patterns targeting the same data regions. Memory utilization remained well-controlled even for extensive datasets, with the secondary row index demonstrating effective compression and selective loading capabilities that minimized memory footprint. This characteristic ensures viability even for extremely large datasets where complete indexes exceed available memory, an essential property for production deployments with constrained resources or massive data volumes.

Real-world Case Studies from Production Environment

The practical effectiveness of the approach was validated through deployment in production environments supporting real-time analytics applications with demanding performance requirements. These environments process continuous event streams requiring both immediate query availability and support for retroactive corrections—a combination particularly challenging for traditional copy-on-write implementations due to modification overhead (Armbrust, M. *et al.*, 2021). The first case study examined high-volume data pipeline processing interaction events requiring frequent updates to correct attribution errors. Prior implementation using file-level copy-on-write experienced significant processing delays during peak periods due to the overhead of rewriting large files for relatively minor corrections. After migration to the partial copy-on-write implementation, the pipeline maintained consistent processing rates regardless of correction volume, eliminating previous

Publisher: SARC Publisher

performance bottlenecks and enabling more responsive data correction workflows. The second case study focused on reference data maintenance workflows performing frequent dimension table updates consumed by multiple downstream analytical processes. The traditional implementation suffered from progressively increasing latency as dimension tables grew, eventually threatening service-level agreements for data freshness. The partial copy-on-write implementation reduced update latency dramatically while decreasing resource consumption, enabling more frequent updates that improved analytical accuracy (Armbrust, M. *et al.*, 2021). The third case study involved compliance and privacy-related selective record removal based on retention policies and regulations. This workflow previously required dedicated maintenance windows to minimize performance impact. With the partial copy-on-write implementation, these operations executed concurrently with regular workloads without noticeable degradation, eliminating scheduled downtime requirements. Across all case studies, the implementation demonstrated enhanced stability through more predictable resource utilization patterns and reduced operational complexity by eliminating specialized workflows previously needed to manage modification performance impacts.

Table 4: Case Study Performance Improvements. (Armbrust, M. *et al.*, 2021)

Use Case	Workload Characteristics	Traditional CoW Challenges	Partial CoW Benefits
Event Data Pipeline	High-volume point updates for attribution correction	Processing delays during peak periods	Consistent throughput regardless of correction volume
Dimension Table Maintenance	Frequent updates to reference data	Increasing latency with table growth	Reduced latency, decreased resource consumption
Compliance Record Removal	Selective deletion based on retention policies	Required maintenance windows	Concurrent execution without performance impact
Real-time Analytics	Continuous updates with immediate query needs	Query interference during updates	Minimal interference through efficient versioning

Analysis of I/O Reduction and Computational Efficiency Gains

Detailed system metric analysis reveals the specific mechanisms driving the observed performance improvements. The most significant factor is the dramatic reduction in I/O requirements for update operations. For typical point update scenarios, measurements show that the partial copy-on-write approach reduces read I/O volume by a factor proportional to the ratio between file size and page size, as only affected pages require reading rather than entire files (Vo, H. T. *et al.*, 2012). This efficiency gain directly translates to improved throughput and reduced system load during modification operations. Write I/O volume shows corresponding improvements, with reduction directly proportional to the percentage of pages affected by modifications. For workloads where updates concentrate within specific data regions—a common pattern in many real-world scenarios—the write I/O reduction approaches theoretical maximums where only modified data requires persistence. Beyond raw I/O reduction, the approach delivers substantial computational efficiency gains through several complementary mechanisms. By deserializing and reserializing only modified pages rather than entire files, CPU utilization for update operations decreases proportionally to the affected page percentage (Armbrust, M. *et al.*, 2021). This reduction particularly benefits datasets with complex schemas or compression settings that make serialization operations computationally expensive. Memory efficiency improves considerably as the implementation needs to materialize only modified pages rather than complete file contents. This property enables efficient processing of updates to files that would otherwise exceed available memory, eliminating complex spilling mechanisms or workload-specific tuning requirements. Network transfer volume in distributed execution environments shows corresponding reductions, as only modified pages require transmission between storage and processing nodes. This efficiency particularly benefits cloud environments where network bandwidth limitations may otherwise constrain performance. The performance benefits compound in end-to-end workflows combining modification operations with subsequent analytical queries. The optimized metadata management enables queries to efficiently identify and access only the most

Publisher: SARC Publisher

recent versions of modified pages, reducing unnecessary I/O for analytical workloads processing recently modified data (Vo, H. T. *et al.*, 2012). These efficiency gains collectively translate to improved resource utilization across the entire system, allowing the same hardware to support higher throughput and lower latency while reducing interference between concurrent workloads.

CONCLUSION

The partial copy-on-write mechanism represents a significant advancement for data lakehouse architectures by addressing a fundamental performance limitation in current implementations. By shifting from file-level to page-level granularity for update operations, the solution dramatically reduces both I/O and computational overhead while maintaining full ACID compliance. The secondary row indexing structure enables precise record localization without compromising the columnar storage benefits inherent to Parquet files. Production deployments demonstrate that this approach scales effectively across diverse workloads and dataset sizes, maintaining consistent performance where traditional methods degrade. The integration pathways established with existing frameworks ensure adoption can proceed incrementally without disrupting current operations. As data lakehouses continue evolving to support increasingly demanding real-time analytics workloads, page-level modification strategies will play a crucial role in bridging the performance gap between traditional warehouses and lake-based architectures. Future directions include extending these techniques to additional file formats, developing hybrid storage models that combine in-place and copy-on-write strategies based on workload characteristics, and further optimizing metadata structures to support ultra-large-scale deployments.

REFERENCES

1. Behm, A., Palkar, S., Agarwal, U., Armstrong, T., Cashman, D., Dave, A., ... & Zaharia, M. "Photon: A fast query engine for lakehouse systems." *Proceedings of the 2022 International Conference on Management of Data*. (2022).
2. Xinli Shang *et al.*, "Fast Copy-On-Write within Apache Parquet for Data Lakehouse ACID Upserts," *Engineering Blog*, (2023). : https://www.uber.com/en-IN/blog/fast-copy-on-write-within-apache-parquet/
3. Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., ... & Zaharia, M. "Delta lake: high-performance ACID table storage over cloud object stores." *Proceedings of the VLDB Endowment* 13.12 (2020): 3411-3424.
4. Alsubaiee, S., Behm, A., Borkar, V., Heilbron, Z., Kim, Y. S., Carey, M. J., ... & Li, C. "Storage management in AsterixDB." *Proceedings of the VLDB Endowment* 7.10 (2014): 841-852.
5. Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., ... & Zumar, C. "Accelerating the machine learning lifecycle with MLflow." *IEEE Data Eng. Bull*. 41.4 (2018): 39-45.
6. Shen, M., Zhou, Y., & Singh, C. "Magnet: push-based shuffle service for large-scale data processing." *Proceedings of the VLDB Endowment* 13.12 (2020): 3382-3395.
7. Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T. "Dremel: interactive analysis of web-scale datasets." *Proceedings of the VLDB Endowment* 3.1-2 (2010): 330-339.
8. Shaikhha, A., Klonatos, Y., & Koch, C. "Building efficient query engines in a high-level language." *ACM Transactions on Database Systems (TODS)* 43.1 (2018): 1-45.
9. Vo, H. T., Wang, S., Agrawal, D., Chen, G., & Ooi, B. C. "Logbase: A scalable log-structured database system in the cloud." *arXiv preprint arXiv:1207.0140* (2012).
10. Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics." *Proceedings of CIDR*. Vol. 8. (2021).

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

Cite this article as:

Dubey, P. " High-Performance Copy-On-Write for ACID Transactions in Parquet-Based Data Lakehouse." *Sarcouncil Journal of Engineering and Computer Sciences* 4.7 (2025): pp 176-185.

Publisher: SARC Publisher