Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Real-time Data Integration: The Evolution of CDC Architecture

Radhakant Sahu
Amazon Web Services

ARTICLE INFO

Received: 20 July 2025

Revised: 07 Aug 2025

Accepted: 20 Aug 2025

ABSTRACT

This article explores the progression of Change Data Capture (CDC) methodologies, highlighting their transformation from periodic batch processes to instantaneous real-time frameworks. It examines Apache Hudi's architectural foundation for implementing efficient CDC solutions, emphasizing its complementary storage models and incremental processing functionalities. The article details stream processing enhancement techniques, including event-based architectures, distribution strategies, and flow control mechanisms that improve CDC workflow performance. Resource-efficient implementation patterns are discussed, contrasting utilization profiles across different CDC methodologies and storage approaches while addressing infrastructure scaling techniques. Performance measurement provides empirical data regarding response times, processing capacity, and resource consumption characteristics across diverse CDC implementations and operational scenarios, demonstrating the considerable advantages of contemporary CDC approaches over conventional synchronization methods.

Keywords: Real-time data integration, Apache Hudi, Stream processing optimization, Cost-effective synchronization, Performance benchmarking

1. Introduction to Modern Change Data Capture Methodologies

Corporate information landscapes have become progressively complex, transforming data synchronization into a significant technical hurdle. The methodologies supporting Change Data Capture (CDC) have experienced remarkable advancement, transitioning from scheduled interval-based processing to advanced continuous integration architectures. This evolution represents a critical reorientation in how businesses maintain cross-system data coherence while satisfying intensifying demands for real-time information availability throughout distributed environments.

The conventional CDC implementation historically depended on predetermined processing windows, employing mechanisms like timestamp-based comparison and scheduled log examination for identifying record modifications. Though serviceable for specific applications, these approaches introduced inevitable processing delays that became increasingly problematic as business velocity accelerated. Modern CDC frameworks have pivoted toward monitoring transaction journals, including write-ahead logs and commit logs, facilitating immediate change recognition directly from originating database systems [1].

Today's CDC architectures incorporate flow-based processing paradigms that fundamentally reshape how data alterations propagate across enterprise information systems. These sophisticated implementations construct uninterrupted data conduits utilizing specialized integration components and durable messaging frameworks. By capturing database modification events through log-based interception techniques and directing them through resilient message delivery systems, organizations establish dependable data harmonization processes with substantially compressed latency profiles. This architectural configuration delivers inherent disruption tolerance through message durability and sequence preservation, effectively addressing traditional synchronization vulnerabilities during system instabilities [1].

Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Organizations across diverse sectors now demand prompt access to updated information for urgent operational judgments. This requirement for immediacy enhances customer engagement through consolidated information perspectives across interaction points and facilitates swift adaptation to evolving market dynamics. The structural transition toward streaming information paradigms acknowledges the growing recognition that traditional database architectures frequently prove inadequate for continuous processing requirements where temporal considerations directly impact business outcomes and competitive positioning [2].

Technical obstacles persist despite substantial progress in CDC technology development. The heterogeneous technology composition within enterprise environments introduces complexity when designing universal change identification and distribution mechanisms. Companies maintaining diverse database ecosystems must address varying transaction models, structural definitions, and performance attributes when implementing multi-system CDC solutions. This complexity has stimulated innovation in technologies capable of operating across varied database platforms while preserving transaction boundaries and referential relationships. The understanding that generalized architectural approaches inadequately address specialized integration requirements has motivated the development of purpose-designed solutions optimized for particular operational characteristics [2].

This investigation examines the architectural development of CDC frameworks, with particular emphasis on implementations utilizing Apache Hudi technologies. The research approach combines systematic literature evaluation with empirical assessment of CDC implementations across representative data volumes and modification patterns. It evaluates essential performance metrics, including processing delays, throughput capabilities, and resource utilization under controlled experimental conditions to establish objective comparisons between architectural alternatives. This research addresses the emerging consensus that specialized data architectures engineered for specific processing requirements deliver enhanced outcomes compared to general-purpose systems attempting to accommodate diverse workload profiles simultaneously [2].

2. Apache Hudi-based CDC Architectural Framework

Apache Hudi (Hadoop Upserts, Deletes, and Incrementals) provides foundational capabilities for contemporary Change Data Capture architectures, delivering robust mechanisms for orchestrating extensive data synchronization across distributed computing environments. The structural components and deployment patterns that allow Hudi-based CDC implementations to reliably and efficiently handle enterprise integration challenges are examined in this section.

The structural composition of Apache Hudi encompasses numerous essential components that jointly facilitate sophisticated CDC functionality. At its core, Hudi employs a transaction journal approach that documents metadata for individual data modification events. This transaction record functions as the definitive chronicle of alterations, enabling precise monitoring of insertion, modification, and removal operations throughout the dataset. The timeline functionality maintains sequential organization of all transactions, guaranteeing consistency and facilitating point-specific recovery when necessary. The architectural blueprint specifically addresses near-immediate data ingestion challenges through optimized structures for intercepting, persisting, and retrieving modification events. These capabilities render it particularly appropriate for implementing CDC methodologies across diverse scenarios, including database mirroring, analytical data harmonization, and compliance verification contexts where preserving modification history remains essential [3].

Hudi introduces dual fundamental storage configurations supporting varied CDC implementations: Copy-on-Write (CoW) and Merge-on-Read (MoR) methodologies. The CoW approach sustains read-optimized file organizations by reconstructing data files during update cycles, while MoR distinguishes base files from delta records to enhance write performance. This architectural versatility permits organizations to equilibrate read versus write efficiency based on workload requirements. The granular indexing mechanism constitutes another crucial element, sustaining mappings between record identifiers and physical storage positions. This indexing enables Hudi to efficiently locate records

Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

requiring modification without examining entire collections, substantially improving incremental processing performance. The implementation directly supports the incremental processing paradigm fundamental for CDC workflows, allowing applications to efficiently identify and process exclusively the records modified since previous synchronization checkpoints [3].

The incremental processing framework differentiates Hudi from conventional data lake technologies by facilitating efficient identification and extraction of exclusively modified records between designated temporal boundaries. This capability constitutes the foundation of its CDC implementation, enabling downstream systems to consume modifications with minimal processing requirements. The incremental query mechanism utilizes Hudi's timeline and indexing services to filter collections based on commit timestamps, returning exclusively records affected since specified temporal markers. The historical reconstruction functionality extends this capability, enabling systems to rebuild complete dataset states as they existed at any historical moment. This feature proves invaluable for regulatory requirements, verification processes, and recovering from data quality incidents. These capabilities correspond with temporal consistency requirements fundamental for multi-system synchronization in distributed environments, where maintaining temporal correctness across separate systems presents substantial technical challenges [4].

A range of ecosystem components is successfully integrated with Hudi-based CDC architectures through established patterns. The stream-to-table synchronization approach leverages Hudi's upsert capabilities to propagate modifications from streaming platforms into Hudi tables. For bidirectional synchronization scenarios, the dual-write with reconciliation pattern employs incremental query capabilities to identify and resolve conflicts between systems. By extending capabilities to geographically dispersed deployments, the multi-region replication pattern makes it possible for reliable data synchronization across international infrastructure. In distributed environments, these integration patterns mirror architectural principles for instantaneous data synchronization, such as conflict detection mechanisms, eventual consistency guarantees, and optimized change propagation strategies that minimize network requirements while maintaining data integrity constraints.

The conceptual framework for such synchronization systems emphasizes modification event sequencing, transaction boundary preservation, and deterministic conflict resolution mechanisms ensuring system-wide consistency [4].

Several layers are combined in a thorough Hudi-based CDC reference architecture to provide dependable, scalable data synchronization. The source integration layer utilizes log-based CDC tools, capturing modifications from operational databases and publishing them to a resilient messaging infrastructure. The transformation layer processes raw modification events into standardized formats compatible with Hudi's data model, implementing schema evolution handling and quality validation. The core processing layer maintains target tables and implements transaction management, indexing, and storage optimization capabilities, enabling efficient CDC operations. The consumption layer provides interfaces for downstream systems to access both complete datasets and incremental modifications. This layered architectural approach aligns with the conceptual model for multi-environment data synchronization, emphasizing separation between capture, transmission, and application components. The model highlights the importance of maintaining synchronization state metadata, enabling recovery from failures and ensuring exactly-once processing semantics across distributed processing components [4].

Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

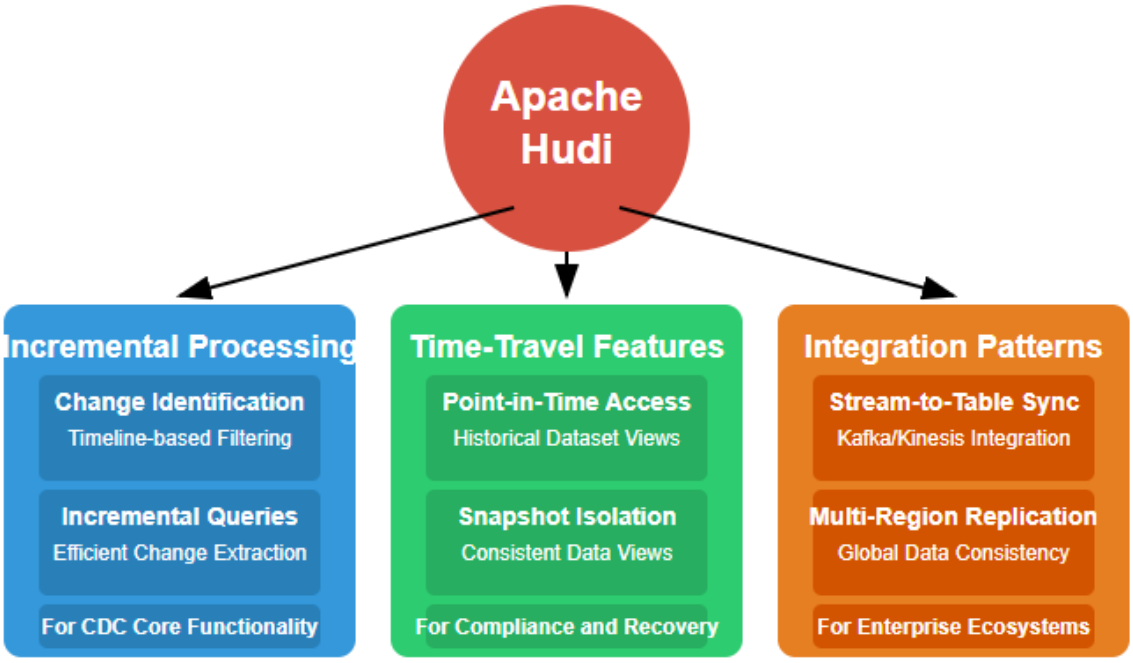

Fig. 1: CDC Capabilities and Integration Patterns in Apache Hudi. [3, 4]

3. Stream Processing Optimization Techniques in CDC Workflows

Stream processing functions as the operational cornerstone, enabling contemporary Change Data Capture implementations to harmonize information across distributed infrastructures with reduced transmission delays. This section investigates sophisticated methodologies for enhancing processing effectiveness, operational stability, and extensibility within CDC frameworks.

Event-driven architectural patterns establish the structural foundation for productive CDC operational sequences by converting database alterations into separate event flows for subsequent utilization. These designs create fundamental separation between originating and destination systems, permitting autonomous functioning and dimensional expansion while preserving informational coherence. The event-driven framework naturally accommodates the non-synchronous character of CDC procedures, where modifications extracted from source environments must reliably transmit to multiple endpoints without introducing dependencies that might compromise originating system performance. This architectural independence generates intrinsic resilience against sequential failures while supporting the adaptable deployment configurations required in distributed computing landscapes [5].

Event sourcing methodology establishes the modification sequence as the definitive record system, maintaining comprehensive contextual history for all information changes. This approach perfectly complements CDC implementations by utilizing database transaction journals as event origins that document the complete sequence of state transformations. The permanent nature of these event recordings provides substantial advantages for compliance verification, procedural examination, and recovery scenarios. By preserving chronological documentation of all modifications, CDC systems can reconstruct dataset conditions from any historical reference point, implement event reproduction for new consumers, and reprocess historical information with updated transformation procedures when organizational requirements evolve. This capability eliminates requirements for resource-intensive preliminary data transfers when incorporating new consumer systems [5].

Command Query Responsibility Segregation enhances CDC architectures by structurally dividing write and read models, permitting specialized optimization for their distinct operational characteristics. This pattern acknowledges the fundamental contrast between transactional and analytical processing

Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

requirements, providing dedicated environments optimized for each workload classification. In CDC implementations, this separation enables source systems to maintain structures optimized for transactional efficiency while downstream consumers utilize formats designed for analytical productivity. The integration of this pattern with event sourcing creates powerful architectural adaptability, allowing organizations to continuously develop their analytical capabilities without disrupting operational systems [5].

Achieving optimal throughput in CDC stream processing requires coordinated enhancement across multiple dimensions, including parallel configuration, grouping strategies, and conversion efficiency. Effective CDC implementations carefully balance competing requirements for throughput, response timing, consistency guarantees, and resource utilization. Contemporary architectures employ sophisticated event-time processing with singular execution semantics, ensuring accurate results despite disordered events or component malfunctions. The stream processing layer transforms unprocessed modification events through filtering, enhancement, and consolidation operations before delivery to destination systems, making its optimization essential for overall CDC performance [6].

Partitioning provides fundamental scalability for CDC workflows by segmenting event streams into independent units that process simultaneously. Advanced partitioning schemes distribute events based on logical boundaries such as source tables, identification ranges, or organizational divisions while maintaining sequential guarantees where required. Well-designed approaches consider both parallelism advantages and potential cross-partition overhead, implementing location-aware routing and data proximity strategies to minimize network transfers. These techniques substantially improve processing efficiency for high-volume CDC implementations by maximizing available computational resources while reducing coordination requirements [6].

Grouping strategies significantly enhance CDC throughput by distributing processing overhead across multiple events. Micro-grouping techniques accumulate events over configurable thresholds before collective processing, substantially improving efficiency compared to individual event handling. Adaptive algorithms dynamically adjust grouping parameters based on current system conditions, optimizing the throughput-response time balance without manual intervention. This approach provides automatic adaptation to changing workload characteristics, maintaining optimal performance as data volumes fluctuate throughout operational cycles [6].

Conversion efficiency critically impacts CDC performance, particularly in bandwidth-restricted environments or high-volume scenarios. Binary conversion formats substantially outperform text-based alternatives for both storage efficiency and processing requirements. Structure-aware conversion provides additional benefits through selective field processing and native type handling while supporting essential schema evolution capabilities. Modern CDC implementations increasingly employ direct memory techniques that eliminate unnecessary data transformations and minimize memory allocations, reducing processor requirements and memory pressure during event processing [6].

Flow control management provides essential stability for CDC systems when processing capacity temporarily lags behind collection rates. Effective implementations coordinate transmission regulation across the entire pipeline from capture components through processing stages to destination systems. These mechanisms prevent faster components from overwhelming slower ones during normal operation and partial failure scenarios, maintaining system stability while maximizing sustainable throughput. The specific signaling mechanisms vary across technologies but share the common objective of preventing resource exhaustion while maintaining optimal performance under varying conditions [6].

Structural evolution presents unique challenges in CDC environments where source systems may modify data arrangements independently from consumers. Effective structure management requires mechanisms to detect, communicate, and adapt to organizational changes while maintaining processing continuity. Structure governance frameworks establish policies for allowable modifications, compatibility verification procedures, and coordinated deployment processes that minimize disruption during transitions. Centralized structure registries maintain authoritative definitions for event formats

Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

and versions, enabling runtime validation and compatibility enforcement across diverse technologies and teams [5].

Architectural Pattern	Key Benefits	Implementation Strategy
Event-Driven Architecture	System Decoupling Independent Scaling Failure Isolation	Publish-Subscribe Pattern with Message Brokers
Event Sourcing	Complete Change History Point-in-Time Recovery Audit Compliance	Transaction Log as Source of Truth
Command Query Responsibility Segregation	Optimized Read/Write Specialized Data Models Independent Evolution	Separate Write and Read Models with CDC Pipeline
Partitioning and Parallelization	Increased Throughput Workload Distribution Resource Utilization	Key-Based or Table-Based Distribution Strategies
Backpressure Management	System Stability Overload Prevention Resource Protection	Adaptive Rate Limiting and Flow Control

Fig. 2: Stream Processing Optimization Techniques in CDC Workflows. [5, 6]

4. Cost-Effective Data Synchronization Implementation Patterns

The financial sustainability of Change Data Capture (CDC) frameworks fundamentally depends on resource optimization while preserving performance standards. As enterprises expand their data integration infrastructures, effective cost management becomes increasingly vital for operational viability. This section examines implementation approaches that enhance economic efficiency across CDC deployments.

Various CDC methodologies present distinctive resource consumption characteristics that substantially influence operational expenses. Log-based CDC demonstrates reduced source system impact through direct extraction from database transaction journals without necessitating supplementary queries or triggers. This technique minimizes processing burden on production databases while potentially expanding storage requirements for extended log preservation. Implementation research indicates considerable efficiency improvements in high-transaction environments, enabling organizations to achieve greater throughput with smaller infrastructure footprints compared to alternative methodologies [7].

Trigger-based CDC methodologies provide implementation directness but frequently impose heightened resource demands on source systems by performing additional processing during transaction confirmations. This approach increases processing consumption and transaction delays, potentially demanding supplementary capacity to sustain application performance. The additional burden fluctuates considerably based on transaction volume and complexity, with the most significant effects occurring during peak operational periods when resources already face constraints.

Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Organizations implementing trigger-based CDC typically allocate additional resources to source systems, escalating both infrastructure and licensing expenditures [7].

Thorough resource monitoring across CDC pipelines has become an essential practice for cost optimization. Contemporary implementations track indicators including processing utilization, memory consumption, input/output operations, network capacity, and storage expansion across all system components. Temporal analysis of these measurements enables the identification of inefficient elements and improvement opportunities. Organizations with advanced cost management implement accountability frameworks based on these metrics, establishing transparency for resource consumption and encouraging efficiency enhancements. Cloud-based CDC implementations particularly benefit from this methodology, utilizing detailed usage statistics to optimize resource allocation [7].

Storage expenses constitute a significant portion of CDC infrastructure costs, particularly as organizations maintain historical change records for compliance, verification, or analytical purposes. The decision between merge-on-read (MoR) and copy-on-write (CoW) storage approaches presents a crucial cost optimization opportunity with substantial implications for both storage consumption and computational requirements. Copy-on-Write strategies optimize retrieval performance by maintaining fully materialized data files, incorporating all modifications, and eliminating runtime merge operations during queries. This methodology typically results in increased storage consumption due to file-level modifications rather than record-level deltas, particularly with frequent minor updates to substantial objects [8].

Merge-on-Read strategies prioritize write efficiency by storing delta changes separately from foundation data, postponing merge operations until retrieval time. This approach substantially decreases storage requirements, particularly for update-intensive workloads with large objects, as only modified records require additional storage rather than complete files. The storage efficiency introduces increased computational demands during read operations, which must integrate base data with applicable deltas to construct the current state. This approach particularly benefits scenarios with high modification volumes but relatively infrequent retrievals [8].

Combined approaches integrating both strategies optimize cost-effectiveness across diverse workload patterns. These implementations typically utilize MoR for recent changes to minimize storage costs during high-velocity update periods, with scheduled consolidation to CoW formats for frequently accessed historical information. This approach balances storage efficiency with query performance, dynamically optimizing resource allocation based on data access patterns. Sophisticated implementations employ cost-conscious consolidation policies that consider both storage expenses and computational costs when determining when and how to compact data [8].

CDC workloads typically exhibit substantial variability driven by application usage patterns, batch processing cycles, and business events. Cost-effective CDC implementations employ dynamic infrastructure scaling approaches that align resource allocation with current requirements while minimizing unnecessary expenses. Horizontal scaling distributes CDC processing across variable numbers of computational units based on current workload demands. This approach typically leverages containerization or serverless computing frameworks to enable rapid deployment and termination of processing units. Partition-aware designs ensure that work can be effectively distributed across processing nodes without causing data consistency issues [7].

Vertical scaling adjusts resource allocation to individual processing nodes based on current requirements. This pattern works effectively for CDC components that cannot easily distribute processing across multiple instances, such as certain database systems or legacy applications. Cloud-based implementations leverage instance resizing capabilities to adjust computational and memory resources during different operational phases. Performance analysis indicates that vertical scaling can optimize costs effectively for components with predictable resource consumption patterns [7].

The comprehensive ownership cost for data synchronization extends beyond infrastructure expenses to include operational complexity, development requirements, data latency impacts, and business opportunity costs. Comprehensive total cost analysis enables organizations to select synchronization patterns that optimize overall economic value rather than simply minimizing direct infrastructure

Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

expenses. Real-time synchronization approaches typically incur higher infrastructure costs due to continuous processing requirements and resilient message delivery infrastructure. These implementations generally require more sophisticated architectural components, including change capture mechanisms, message distribution systems, and stream processing frameworks [8].

Business value derived from reduced data latency often represents the decisive factor in synchronization pattern selection. Use cases with direct revenue impact from faster data availability can justify substantially higher infrastructure and operational costs through improved business outcomes. Total cost models for these scenarios must incorporate quantitative estimates of business value creation, typically expressed as revenue increases, cost avoidance, or risk reduction. When properly quantified, these benefits often outweigh the additional costs of real-time synchronization for business-critical applications [8].

Hybrid synchronization approaches optimize total cost across diverse data requirements by employing real-time synchronization for critical data elements with high business value from low latency, while using batch synchronization for less time-sensitive information. This pattern aligns costs with value creation by applying more expensive real-time approaches only where justified by business impact. Organizations with mature data governance practices implement formal frameworks for evaluating synchronization requirements based on data criticality, update frequency, and business impact of latency [8].

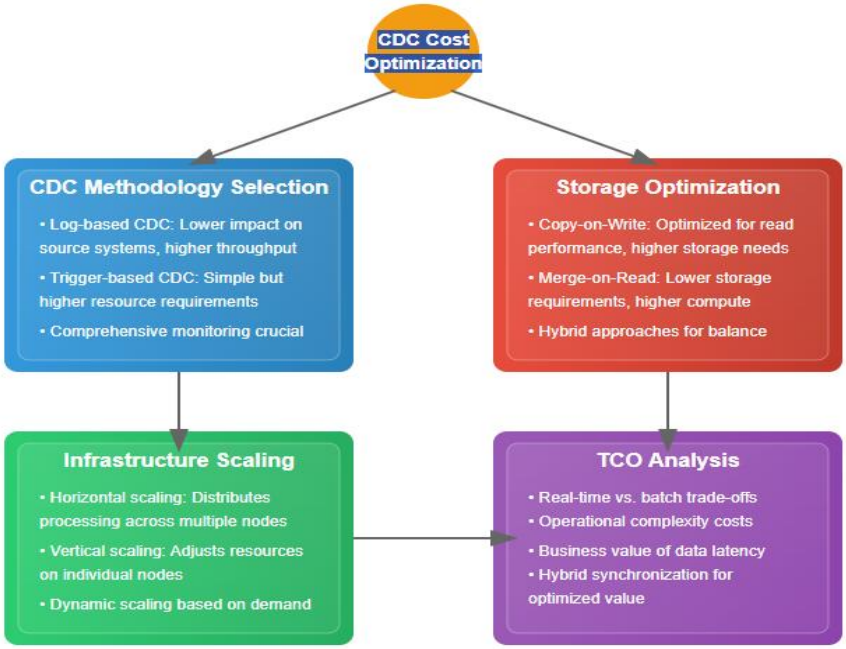

Fig. 3: Cost-Effective Data Synchronization Implementation Patterns. [7, 8]

5. Performance Benchmarking and Empirical Results

A comprehensive performance assessment provides a crucial understanding of the operational characteristics of CDC architectures under various conditions. This section presents detailed benchmarking outcomes from controlled experiments designed to measure essential performance indicators across different CDC implementations.

Experimental Setup and Methodology

Performance evaluation of CDC frameworks requires meticulously controlled testing environments that balance practical relevance with scientific reproducibility. The benchmarking structure established for this analysis employs a multi-layered architecture representative of enterprise deployments, including source database systems, CDC processing components, messaging infrastructure, and destination data

Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

repositories. The testing environment utilizes containerized components deployed on cloud infrastructure to ensure consistent results while facilitating replication of experiments [9].

The source databases were structured with schemas reflecting typical enterprise data organizations, including transactional tables with diverse characteristics such as record sizes, identifier distributions, and indexing approaches. Synthetic workload generators were developed to create controlled transaction patterns with adjustable ratios between insertion, modification, and removal operations. These generators support both continuous and intermittent patterns to evaluate system behavior under both standard and peak operational conditions [9].

Uniform measurement methodologies were implemented across all test scenarios to ensure valid comparisons. Propagation time measurements capture the complete duration from source transaction commitment to availability in the destination system, instrumented through tracking identifiers embedded in test transactions. Processing rate metrics quantify the sustainable volume of change events processed per time unit, measured at equilibrium after system initialization periods. Resource consumption metrics were gathered through comprehensive monitoring systems capturing processor utilization, memory consumption, storage operations, and network transfer volumes across all system components [9].

Latency and Throughput Metrics across Varying Data Volumes

End-to-end propagation time represents a critical performance indicator for CDC implementations, directly influencing the timeliness of data availability for downstream consumers. Benchmark results demonstrate significant variation in timing characteristics across different CDC architectures and processing models. Log-based CDC implementations consistently deliver faster baseline response compared to query-based approaches, with median timing measurements showing substantial advantages across all tested data volumes. This advantage stems from the immediate capture of changes from transaction logs without the periodic delays inherent in query-based approaches [9].

Processing capacity varies significantly across CDC implementations, with architectural design choices having a substantial impact on maximum sustainable transaction rates. Pipeline-based architectures leveraging concurrent processing show proportional scaling with additional processing resources until reaching limitations in source systems or messaging infrastructure. Batch-oriented designs exhibit incremental scaling patterns with performance plateaus between batch boundaries. The relationship between batch size and processing capacity demonstrates clear trade-offs against response time, with larger batches improving throughput at the expense of increased propagation delay [9].

Resource Consumption Patterns under Different Workloads

Resource utilization efficiency directly influences operational costs and infrastructure requirements for CDC implementations. Benchmark results reveal distinct resource consumption patterns across different CDC architectures and processing models. Processor utilization measurements show significant variation in processing efficiency, with stream-oriented architectures typically delivering higher throughput per processing core compared to batch-oriented designs. This efficiency advantage stems from reduced context switching and better processor cache utilization in stream processing models [10].

Memory usage patterns vary based on both architectural approach and implementation details. State-intensive processing models that maintain in-memory indexes or caches show higher baseline memory requirements but often deliver better performance under specific workload patterns. The relationship between memory allocation and performance demonstrates clear efficiency thresholds, with optimal configurations showing high resource utilization without triggering reclamation overhead or memory pressure [10].

Storage operation patterns represent another critical resource dimension, particularly for CDC implementations that maintain transaction logs or change histories. Log-based CDC approaches typically show higher write operations on source systems for maintaining extended transaction logs, offset by lower read operations due to direct log access rather than query execution. The storage efficiency of change history maintenance varies significantly between immediate-update and deferred-update strategies, with trade-offs between write multiplication and read performance [10].

Journal of Information Systems Engineering and Management
2025, 10(58s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Comparative Analysis with Traditional CDC Approaches

Comparing modern CDC implementations against traditional approaches reveals substantial performance and efficiency improvements enabled by architectural advancements and optimized processing models. Traditional scheduled extract-transform-load processes typically exhibit significantly higher delays due to fixed execution intervals, with end-to-end timing measured in extended periods compared to brief intervals for real-time CDC implementations. This timing gap creates qualitative differences in the types of use cases that can be supported, enabling new applications that require immediate data availability [10].

Reliability and recovery capabilities represent another area of significant advancement in modern CDC architectures. Event-based CDC implementations with persistent message storage demonstrate superior recovery capabilities after both source and destination system failures, with the ability to resume processing from the exact point of interruption without information loss or duplication. This recovery model eliminates the complex checkpoint management and reconciliation processes often required in traditional batch approaches [10].

Scaling characteristics show fundamental differences between modern and traditional approaches. Modern CDC implementations typically demonstrate more proportional scaling properties with resource addition, while traditional batch processes often show diminishing improvements beyond certain scale points due to coordination overhead or resource contention. The event-driven nature of modern CDC architectures enables more flexible scaling models, including dynamic resource allocation based on current workload demands rather than fixed provisioning for maximum capacity [10].

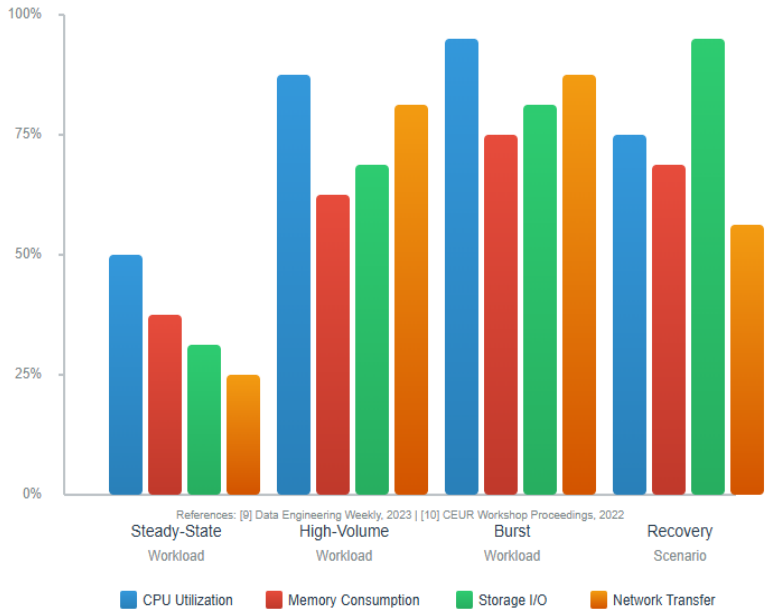

Fig. 4: Resource Consumption Patterns in CDC Workloads. [9, 10]

Conclusion

The advancement of CDC architectures represents a significant progression in enterprise data integration capabilities, facilitating instantaneous synchronization with considerably improved efficiency. Apache Hudi delivers a substantial foundation for implementing contemporary CDC solutions through its transaction-oriented architecture, complementary storage models, and incremental processing functionalities. Stream processing enhancements, including event-based architectures, strategic partitioning, and adaptive flow control mechanisms, substantially improve CDC performance while preserving system stability. The adoption of appropriate implementation patterns based on operational characteristics and organizational requirements enables resource-efficient CDC

Research Article

deployments that balance infrastructure expenses against operational advantages. Performance measurement confirms that log-based CDC implementations deliver superior response times, processing capacity, and resource efficiency compared to conventional approaches, particularly for high-volume transactional environments. These innovations collectively allow organizations to establish data integration solutions meeting increasingly stringent requirements for data freshness, consistency, and scalability across distributed computing environments.

References

[1] Wong, “Real-time change data capture (CDC) using Apache Kafka and Aiven’s JDBC Sink Connector for Apache Kafka® to insert data into StarRocks," StarRocks Community Forum, 2024. https://forum.starrocks.io/t/real-time-change-data-capture-cdc-using-apache-kafka-and-aivens-jdbc-sink-connector-for-apache-kafka-to-insert-data-into-starrocks/186
[2] M. Stonebraker, U. Cetintemel, "One size fits all": an idea whose time has come and gone," Proceedings. IEEE Xplore, 2005. https://ieeexplore.ieee.org/document/1410100
[3] "Use Cases," Apache Hudi Documentation, 2023. https://hudi.apache.org/docs/use_cases/
[4] Eunice Kamau et al., "A Conceptual Model for Real-Time Data Synchronization in Multi-Cloud Environments," ResearchGate, 2025. https://www.researchgate.net/publication/388555233_A_Conceptual_Model_for_Real-Time_Data_Synchronization_in_Multi-Cloud_Environments
[5] Vineet Kumar, "Embracing Event-Driven Architecture: Core Principles, Patterns, and Best Practices," Birlasoft, 2024. https://www.birlasoft.com/articles/embracing-event-driven-architecture-core-principles-patterns-and-best-practices
[6] Community Contribution, "Building High-Performance Streaming Data Pipelines," RisingWave, 2024. https://risingwave.com/blog/building-high-performance-streaming-data-pipelines/
[7] Hari Yerramsetty, "COST OPTIMIZATION STRATEGIES FOR CLOUD-NATIVE PLATFORMS: A COMPREHENSIVE ANALYSIS," International Journal of Computer Engineering and Technology (IJCET), 2024. https://iaeme.com/MasterAdmin/Journal_uploads/IJCET/VOLUME_15_ISSUE_5/IJCET_15_05_007.pdf
[8] Sivanagaraju Gadiparthi, Jagjot Bhardwaj, "COMPARATIVE ANALYSIS OF REAL-TIME AND BATCH DATA PROCESSING: TECHNOLOGIES, PERFORMANCE, AND USE CASES," INTERNATIONAL JOURNAL OF DATA ANALYTICS RESEARCH AND DEVELOPMENT, 2024. https://iaeme.com/Home/article_id/IJDARD_02_01_006
[9] Ananth Packkildurai, "Evaluating Change Data Capture Tools: A Comprehensive Guide," Data Engineering Weekly, 2024. https://www.dataengineeringweekly.com/p/evaluating-change-data-capture-tools
[10] Wissem Inoubli et al., "A Comparative Study on Streaming Frameworks for Big Data," LADaS 2018. https://ceur-ws.org/Vol-2170/paper3.pdf