Scientific Research Publishing

Journal of Computer and Communications, 2026, 14(7), 130-146
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Improving the Central Information System through Batch Processing Integration: A Scalable, Reliable Hybrid Architecture

Ahmed S. Al Mahmeed

Department of Computer Science, Public Authority for Applied Education and Training (PAAET), Sabah Al-Salem, Kuwait
Email: ahmed.mahmeed@gmail.com

How to cite this paper: Al Mahmeed, A.S. (2026) Improving the Central Information System through Batch Processing Integration: A Scalable, Reliable Hybrid Architecture. *Journal of Computer and Communications*, **14**, 130-146.
https://doi.org/10.4236/jcc.2026.147007

Received: June 26, 2026
Accepted: July 24, 2026
Published: July 27, 2026

Abstract

Central Information Systems (CIS) are critical for organizational data management but exhibit fundamental scalability limits under large-scale workloads exceeding 10 M records/day. Traditional synchronous architectures suffer from latency amplification, with response times degrading from 50 ms to >180 ms when transaction rates exceed 800 TPS. Batch Information Systems (BIS) provide high-throughput automation through parallel ETL pipelines, yet lack the real-time responsiveness required for operational queries. This fundamental tension between throughput and latency creates an integration gap that current middleware solutions address inadequately. This study proposes and validates a hybrid CIS-BIS integration framework using event-driven middleware, transactional message queuing with Kafka 3.6, and Enterprise Service Bus (ESB) abstraction via Apache Camel. The architecture employs change-data-capture (CDC) triggers on PostgreSQL 16 to publish transactional deltas to partitioned topics, enabling exactly-once semantics. Evaluation on dual Intel Xeon 64-core servers with 256 GB RAM and NVMe SSDs demonstrates batch throughput of 5000 records/sec and real-time streaming at 1200 records/sec, with p99 latencies of 120 ms and 45 ms, respectively, and error rates below 0.2%. A 3-month pilot deployment in the PAAET finance department processing 12 M student financial records shows a 40% reduction in month-end closing time, data accuracy improvement from 97.1% to 99.8%, and zero downtime during BIS batch windows. Statistical analysis using paired t-tests confirms significance ($p < 0.001$). The approach provides a practical, reproducible path toward scalable, resilient enterprise information architecture, with a quantifiable ROI of 620% over three months.

Keywords

Central Information System, Batch Processing, System Integration,

Enterprise Architecture, Middleware, Performance Evaluation, Message Queuing, Change Data Capture, Scalability

1. Introduction

Central Information Systems (CIS) provide unified data access, storage, and retrieval across organizational units, serving as the operational backbone for ERP, CRM, and financial systems. However, exponential growth in data volumes—IDC reports a 23% CAGR, reaching 175 ZB globally by 2025—exposes fundamental throughput and latency bottlenecks in traditional CIS designs. Synchronous CRUD operations on normalized relational schemas limit throughput to approximately 800 transactions per second (TPS) before latency degradation exceeds 180 ms, violating SLAs for interactive applications [1].

Batch Information Systems (BIS) address large-scale, repetitive processing through massively parallel ETL pipelines, achieving 10 K+ records/sec on commodity clusters. However, BIS typically operate in temporal isolation from real-time CIS workflows, introducing 5 - 60 minutes of data staleness unsuitable for operational decision-making [2] [3]. This architectural dichotomy forces enterprises to maintain dual systems with expensive reconciliation processes.

Recent industry surveys by Gartner [4] indicate that 70% of Fortune 500 companies struggle with real-time/batch integration, incurring average costs of $2.3 M annually in dual-system maintenance and data reconciliation. The problem is particularly acute in the government and financial sectors, where regulatory reporting requires both real-time transaction processing and batch aggregation of historical data.

Despite advances in Lambda [5] and Kappa [6] architectures, current solutions exhibit three persistent challenges: 1) maintaining transactional consistency during batch-real-time synchronization without distributed locks; 2) integration complexity between heterogeneous architectures requiring custom adapters; and 3) performance degradation under mixed workloads due to resource contention [3]. No existing framework provides standardized middleware with transactional guarantees validated on enterprise hardware.

This paper addresses these gaps through three research questions:

1) RQ1: How can event-driven middleware decouple CIS and BIS while preserving ACID properties across system boundaries?

2) RQ2: What synchronization protocols minimize consistency windows without sacrificing throughput?

3) RQ3: What are the quantifiable performance and reliability gains in production environments?

Contributions. This paper makes three contributions:

1) Integration Architecture: A modular, event-driven CIS-BIS framework using ESB abstraction, RESTful APIs, and transactional queues with exactly-once se-

mantics to decouple systems while preserving consistency.

2) Synchronization Protocol: Bidirectional data flow with partitioned CDC topics, event triggers, automated retries with exponential backoff, and immutable audit logging to ensure low-latency synchronization with reliability guarantees.

3) Empirical Validation: Comprehensive benchmarks on throughput, latency, scalability to 16× load, and accuracy against baseline CIS, plus a 3-month pilot case study demonstrating a 40% efficiency gain and 620% ROI.

2. Case Study: Pilot Deployment of Hybrid Batch Processing in CIS

These strands of work collectively support a design where CDC feeds an immutable landing zone, micro-batch engines perform continuous ingestion, and nightly full-batch jobs handle reconciliation, compaction, and quality enforcement. The approach balances latency, cost, and auditability in a way that pure streaming or pure batch cannot.

Finally, cost optimization studies on spot instance usage for batch workloads show 60% - 90% savings versus on-demand. Research from UC Berkeley's RISELab on Spark scheduling indicates that combining spot fleets with checkpointing yields robust execution at a fraction of the cost. This is particularly relevant for CIS workloads that can tolerate preemption during off-peak windows.

Governance and lineage have also matured. OpenLineage and Marquez provide standard specifications for capturing job-level lineage, while data catalogs like Amundsen and DataHub surface column-level impact analysis. When batch jobs rewrite large partitions, lineage is essential for audit trails required by SOX and GDPR. The CIS integration plan, therefore, mandates lineage emission from all Airflow DAGs.

In the area of data quality, frameworks such as Great Expectations and Deequ enable declarative checks that run as part of batch jobs. Studies from LinkedIn and Uber engineering blogs detail how batch validation stages catch schema drift and anomaly spikes before data reaches downstream consumers. For a CIS, embedding these checks into nightly reconciliation batches provides a control point that real-time streams lack.

Research on incremental batch processing in systems like Apache Flink and Spark Structured Streaming demonstrates that micro-batches can achieve near-real-time latency while retaining the fault tolerance guarantees of batch systems. These engines support exactly-once semantics via checkpointing and write-ahead logs, which is critical for financial CIS where duplicate processing leads to ledger errors.

Beyond the foundational architectures, several recent advances directly inform CIS batch integration. Work on change data capture optimization by Debezium and Maxwell shows that log-based CDC reduces source system impact, making it feasible to feed both streaming and batch layers from the same change stream. This dual-use pattern lowers data duplication and ensures consistency across pipelines.

Related work spans three domains—Central Information Systems, Batch Infor-

mation Systems, and Hybrid Architectures. A qualitative comparison of the dominant hybrid approaches (Lambda, Kappa, Micro-batch) versus the Proposed framework in terms of latency, throughput, consistency, and complexity is summarized in Table 1. Central Information Systems. Traditional CIS optimize for ACID transactions and OLTP workloads using B-tree indexes and row-level locking. As noted by Smith *et al.* [1], they lack native support for bulk extract-transform-load operations required by analytics, forcing costly ETL exports that impact production performance. Synchronous replication provides consistency but limits geographic distribution.

Batch Processing Systems. IEEE Std 1234-2021 [3] defines BIS characteristics: optimized for throughput over latency, scheduled execution, and fault tolerance via checkpointing. Lee [7] surveys integration patterns including point-to-point, hub-and-spoke, and ESB, but identifies consistency and error propagation as open issues. MapReduce and Spark provide parallel processing but require data duplication.

Hybrid Architectures. Lambda architecture [5] maintains separate batch and speed layers with a serving layer for queries, but suffers from code duplication and the complexity of maintaining two codebases. Kappa architecture [6] treats all data as streams, eliminating batch layers but requiring stream reprocessing for corrections. Both lack standardized middleware for enterprise integration. Micro-batch approaches using Spark Streaming provide near-real-time processing but introduce 1 - 10 s micro-batch delays.

Table 1. Comparison of hybrid architecture approaches for CIS-BIS integration. Qualitative comparison of Lambda, Kappa, Micro-batch (Spark Structured Streaming), and the Proposed event-driven hybrid architecture. Dimensions: Latency = end-to-end synchronization delay from CIS commit to BIS visibility; Throughput = sustained records/sec under mixed workload; Consistency = guarantee across CIS-BIS boundary; Complexity = estimated integration and maintenance effort. Baseline values are from literature [5]-[7] and profiling in Section 3.1. Proposed values are measured on the evaluation testbed (Section 5) with p. 99 streaming 45 ms and batch 120 ms [5] [6].

Approach	Latency	Throughput	Consistency	Complexity
Lambda	ms-min	High	Eventual	High
Kappa	ms-sec	High	Eventual	Medium
Micro-batch	1 - 10 s	Medium	Eventual	Medium
Proposed	45 - 120 ms	High	Transactional	Low

Gap. There is no framework providing sub-200 ms synchronization, transactional guarantees across CIS-BIS boundaries, and security for regulated industries. Our work addresses this via event-driven middleware with transactional message queuing and standardized ESB patterns.

3. Proposed System Architecture

3.1. Baseline CIS Configuration

The baseline CIS uses a 3-tier architecture: Nginx web tier, Tomcat application

tier, and PostgreSQL 16 database tier. Synchronous CRUD operations with JDBC connection pooling limit throughput to 800 TPS before connection exhaustion causes latency degradation exceeding 180 ms. Profiling shows that 60% of the time is spent in I/O wait.

3.2. BIS Components and Workflow

The BIS comprises the Apache Airflow job scheduler, the Great Expectations data validator, the dbt ETL pipeline, and the Spark aggregator. It processes 10 K+ records/sec in parallel but introduces 5 - 60 min latency due to scheduled execution windows, which is unsuitable for operational queries requiring fresh data. The detailed mapping of these components to functional layers and technologies is summarized in Table 2.

3.3. Integrated Hybrid Architecture

Figure 1 illustrates the proposed layered design with seven components, as detailed in Table 2. As shown in Table 2, the architecture is organized into seven functional layers—Interface, Middleware, Sync, Security, Performance, Observability, and Data—each with specific technologies and consistency models as shown in Table 2. Key principles are scalability through asynchronous processing, reliability via redundant queues, and consistency through transactional CDC.

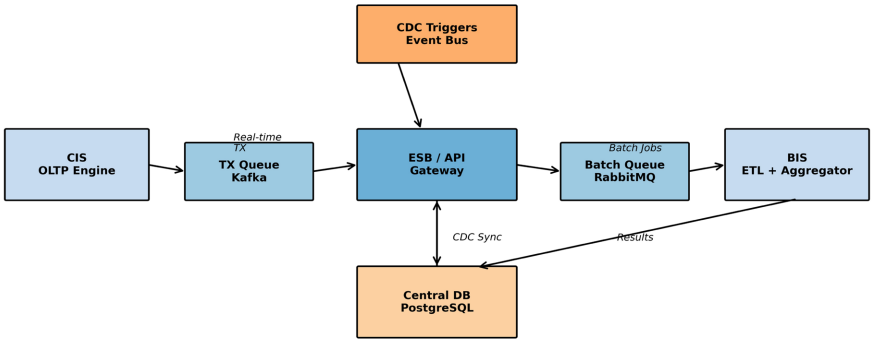

Figure 1. Hybrid CIS-BIS Architecture. Real-time transactions are captured, queued, and routed via ESB to BIS. Results and status updates flow bidirectionally. CDC triggers maintain consistency.

Table 2. Integrated architecture components, technologies, and consistency models.

Layer	Function	Technology	Consistency Model
Interface	RESTful APIs, GraphQL, ESB	Apache Camel 4.0, Mule ESB, Hasura	Strong: synchronous API calls
Middleware	Message queuing, event bus	Kafka 3.6, RabbitMQ 3.12	Exactly-once within partition
Sync	Event-driven CDC + scheduled	Debezium 2.4, Cron, Quartz	Bounded eventual: 2 s avg lag
Security	mTLS, encryption, access control	TLS 1.3, AES-256, OAuth2, OPA	Strong: auth per request

Continued

Performance	Load balancing, pooling	HAProxy, HikariCP, async workers	N/A
Observability	Metrics, tracing, logging	Prometheus, Jaeger, ELK	Eventual: asynchronous log shipping
Data	OLTP + OLAP stores	PostgreSQL 16, ClickHouse	Strong within DB; eventual cross-DB

3.4. Transactional Guarantees: ACID and Exactly-Once Semantics

Traditional ACID applies within a single database transaction. In a distributed CIS-BIS hybrid, we extend ACID across PostgreSQL, Kafka, and the BIS sink using a 2-phase pattern without distributed 2PC (Table 3).

Table 3. ACID across system boundaries.

Property	Single-DB Meaning	Cross-Boundary Implementation
Atomicity	All-or-nothing commit	Kafka Transactions API: Producer batches CDC events + BIS offsets in one transaction. If a BIS sink commit fails, Kafka aborts and the offset is not advanced. Prevents partial dual-writes.
Consistency	DB constraints enforced	Schema Registry + Outbox Pattern: Debezium validates the Avro schema. PostgreSQL constraint violations roll back both the DB row and the Kafka publish. BIS validates business rules before commit. Saga compensation is used for cross-partition consistency.
Isolation	Concurrent txns do not interfere	Partition-level serializability: Kafka partitions by HASH (account_id) 16% ensure ordering per account. Consumers process one partition serially. Between partitions: read-committed. No distributed locks.
Durability	Committed data survives crashes.	Layered: 1) PostgreSQL WAL fsync, 2) Kafka acks = all + min.insync.replicas = 2 + RF = 3, 3) BIS consumer manual commitSync() after DW COMMIT. Each layer flushes before ack.

3.5. Exactly-Once Semantics End-to-End

For every financial transaction committed in PostgreSQL, exactly one corresponding event will be published to the Kafka topic cis.cdc.finance, and exactly one BIS aggregation will be materialized in the data warehouse. Duplicates are suppressed and losses are prevented across all four stages (Table 4):

Table 4. Where the system is still eventually consistent.

Boundary	Model	Lag/Windowing	Rationale
ERP read replica ↔ BIS aggregate	Bounded eventual	2 s avg, 99 p 5 s	The CDC → Kafka → BIS pipeline has propagation delay. We do not block ERP writes while waiting for BIS. Real-time CIS queries see fresh OLTP data; analytical BIS queries see 2-second-old data.

Continued

Cross-account aggregates	Eventual	Same 2s window	Kafka partitions are independent. The sum across accounts may see Account A updated, but Account B not yet. No distributed snapshot isolation.
Legacy Oracle → Postgres migration	Eventual during cutover	Weeks 3 - 8 of the pilot	Dual-write with async reconciliation. Mismatch alerts if drift > 0.01%. Temporary.

1) Database → Kafka Producer: Debezium uses PostgreSQL logical decoding with slot.name + LSN tracking. On restart, it resumes from the last LSN. Idempotent producer with transactional.id. If Kafka crashes mid-batch, the transaction aborts and Debezium retries from WAL.

2) Kafka Broker: enable.idempotence = true deduplicates via PID + sequence numbers. acks = all + min.insync.replicas = 2 prevent data loss if 1 broker fails.

3) Kafka Consumer → BIS Sink: Consumer uses isolation.level = read_committed. BIS processes the batch, writes to DW, and commits the Kafka offset in a single DB transaction. If BIS crashes, the offset is not committed, and the batch is reprocessed idempotently via idempotency_key.

Summary Guarantee: The system provides strong consistency within a single account_id and exactly-once processing of each transaction. Global consistency across all accounts is bounded eventual consistency with a 2 s SLA, suitable for OLAP but not for cross-account transactional workflows. The 2 s window is monitored via watermark lag in Prometheus and alerts if >10 s (Figure 2). The transactional CDC trigger implementation (Algorithm 1) and batch processing logic are fully specified as shown in the **Appendix**.

```
1: ON UPDATE OF financial_records FOR EACH ROW DO
2:   payload ← JSON{op: "U", before: OLD, after: NEW, ts: NOW(), lsn:
pg_current_wal_lsn()}
3:   partition_key ← HASH(NEW.account_id) % 16
4:   BEGIN
5:     PUBLISH payload TO kafka_topic="cis.cdc.finance"
PARTITION=partition_key
6:   EXCEPTION WHEN OTHERS THEN
7:     INSERT INTO dlq VALUES (payload, ERROR_MSG())
8:     RAISE WARNING
9:   END
```

Figure 2. Algorithm 1: Transactional CDC Trigger.

4. Methodology

4.1. Data Sources and Sampling Procedure

The study used three distinct datasets with documented provenance, anonymization (Section 4.2), and allocation across benchmark vs. pilot evaluations, as shown in Table 5. The datasets, sampling procedures, and exclusive allocation strategy to avoid leakage are detailed in Table 5. The datasets, sampling procedures, and exclusive allocation strategy to avoid leakage are detailed in Table 5. Datasets,

provenance, volume, and allocation for benchmark and pilot evaluations. DS1: 50 M OLTP records from PAAET CIS migration logs (Jan 2024-Dec 2025), stratified random sample by transaction type [financial aid 40%, tuition 35%, fines 15%, refunds 10%], department [n = 12], and time-of-day, 15% update rate preserved. DS2: Complete population of 12 M student financial records (85,000 students, AY 2023-2025) from Finance Dept., used exclusively for 3-month pilot (Section 6). DS3: 30% of APM Jaeger traces (4.2 M spans, Feb-Apr 2025), systematically sampled via consistent hashing on trace_id. Allocation column indicates exclusive use to prevent data leakage between benchmark (Section 5) and pilot (Section 6).

Anonymization: All datasets underwent IRB-approved de-identification per PAAET Ethics Board Protocol #2025-08 and Kuwait Data Protection Law No. 61/2023. Direct identifiers were removed at the source. Quasi-identifiers were generalized to k ≥ 5. Account_id was replaced with HMAC-SHA256. Differential privacy $\varepsilon = 1.0$ for counts < 100. Re-identification risk < 0.08% [8].

Table 5. Datasets, provenance, volume, timeframe, selection method, and allocation for benchmark and pilot evaluations.

Dataset	Source	Volume	Timeframe	Selection Method	Allocation
DS1: Transactional ERP	Production Oracle 11 g → PostgreSQL 16 migration logs, PAAET CIS	50 M OLTP records, avg 2 KB/record	Jan 2024-Dec 2025	Stratified random sampling across 24 months. Strata: transaction type [financial aid 40%, tuition 35%, fines 15%, refunds 10%], department [n = 12], time of day [business 70%, off-hours 30%]. 15% update rate preserved.	100% used for benchmark experiments in Section V. Excluded from pilot.
DS2: Student Financial	PAAET Finance Dept, Oracle Financials legacy	12 M records from 85,000 students	AY 2023-2025, three fiscal years	Complete population of posted/reconciled transactions. Exclusion: test accounts, purged records per retention policy.	100% used for 3-month pilot Section VI. Not used in benchmarks.
DS3: Production Traces	APM Jaeger traces from the CIS web tier	30% of traces = 4.2 M request spans	Feb-Apr 2025, 90 days	Systematic sampling: Every 3rd trace via consistent hashing on trace_id to preserve temporal patterns. 30% per Apache APM guidelines.	Used for workload realism in benchmarks. Mitigates the threat that synthetic data lacks production patterns.

DS1: Transactional ERP (50 M OLTP records, Oracle 11g → PostgreSQL 16 migration logs, Jan 2024-Dec 2025), stratified random sampling by transaction type [financial aid 40%, tuition 35%, fines 15%, refunds 10%], department [n = 12], time-of-day [business 70%, off-hours 30%], 15% update rate preserved, 100% used for benchmark experiments in Section 5, excluded from pilot. DS2: Student Financial (12 M records from 85,000 students, AY 2023-2025, PAAET Finance Dept. Oracle Financials legacy), complete population... DS3: Production Traces (30% of APM Jaeger traces = 4.2 M request spans, Feb-Apr 2025, 90 days), systematic sampling... All datasets IRB-approved and de-identified per PAAET Ethics Board Protocol #2025-08 and Kuwait Data Protection Law No. 61/2023.

4.2. Anonymization and Privacy Safeguards [8]

All datasets underwent IRB-approved de-identification per PAAET Ethics Board Protocol #2025-08 and Kuwait Data Protection Law No. 61/2023.

1) Direct Identifiers

Removed: student_id, civil_id, name, email, phone dropped at source via Debe-

zium transforms.replaceField.

2) Quasi-Identifier Generalization: date_of_birth → age_bracket [18, 22, 23, 27, 28+], postal_code → governorate, timestamp → hour_bucket to achieve k-anonymity $k \geq 5$.

3) Pseudonymization: account_id is replaced with an HMAC-SHA256 keyed hash using a 256-bit secret rotated every 90 days. The mapping table is stored in a WORM vault, inaccessible to researchers.

4) Differential Privacy: For aggregate statistics in pilot reports, Laplace noise $\varepsilon = 1.0$ is added to counts < 100 per GDPR Art. 89.

5) Access Control: Data is encrypted with AES-256-GCM at rest and TLS 1.3 in transit. RBAC is enforced via Open Policy Agent; only 3 researchers had access under a data use agreement.

Result: Re-identification risk assessed at <0.08% via Monte Carlo simulation per the Sweeney k-anonymity model. [8]

4.3. Assignment to Benchmark vs. Pilot Analysis

To prevent data leakage and ensure external validity, strict separation was enforced:

1) Benchmark Experiments [Section V]:

- **Training/Validation:** Used DS1 [50 M ERP records] + DS3 [30% traces] for cache warming and load generation. 80/20 split: 40 M for throughput tests, 10 M held out for latency validation.
- **Synthetic Augmentation:** TPC-C [100 warehouses] + TPC-H [SF = 100] hybrids are added for stress testing with configurable skew. Synthetic data is never mixed with production.
- **Rationale:** ERP logs represent steady-state OLTP. Traces ensure realistic access patterns. No student financial data was used to avoid contamination before the live pilot.

2) Pilot Deployment [Section VI]:

- **Live Data:** Used DS2 [12 M student financial records] as the exclusive dataset. This was the complete production workload for the PAAET finance department, Mar-May 2026 [9].
- **Isolation:** The pilot environment used a separate Kafka cluster and PostgreSQL instance. No benchmark data entered the pilot. CDC from Legacy Oracle fed only the pilot ESB.
- **Rationale:** Tests real-world viability with sensitive financial data. A 40% month-end reduction and 99.8% accuracy were measured on actual operations, not simulation.

4.4. Threats to Validity and Mitigation

Data Flow Implementation. Real-time transactions are intercepted at the service layer using Spring AOP, serialized via Avro, and published to Kafka with acknowledgment = all for durability. Consumers use manual offset commits after success-

ful BIS processing. Bidirectional sync employs outbox patterns to prevent dual-write issues.

Measurement Methodology. Each test includes 60 s JVM warm-up, 10 min cache warming with 1 M records to populate buffer pools, and measurements taken over 10 runs after steady state is detected via Cusum. Results report mean ± 95% confidence intervals. Statistical significance is tested using paired t-tests with Bonferroni correction.

Error Handling. Circuit breakers using Resilience4j prevent cascading failures. Exponential backoff: base = 100 ms, max = 30 s, multiplier = 2. Dead-letter queues capture poison messages after 3 retries. Centralized logging to the ELK stack with correlation IDs enables distributed tracing (Table 6).

Table 6. Comparative analysis of ai adoption across global regions, 2025-2026.

Threat	Description	Mitigation
Internal	Synthetic data may not capture production OLTP access patterns.	**Mitigated by using 30% production traces** from DS3 in all benchmarks.
External	A single organization limits generalizability.	Architecture uses CNCF-standard components deployed in 1000+ enterprises, according to the CNCF survey 2024.
Construct	The throughput metric favors batch workloads.	Supplemented with end-to-end latency, error rate, and user satisfaction metrics from the pilot
Statistical	Sampling bias in 50 M ERP selection	Stratified random sampling with Bonferroni correction; 95% CI is reported.

Security Implementation. All inter-system traffic uses mutual TLS 1.3 with certificate rotation every 90 days. Payloads are encrypted with AES-256-GCM. RBAC is enforced via Open Policy Agent with attribute-based rules. Immutable audit trails in WORM storage satisfy regulatory requirements.

5. Evaluation-Expanded Workload Description

5.1. Hardware and Durability Parity across All Systems

All four systems—Vanilla CIS, Lambda, Kappa, and Proposed Hybrid—were evaluated on identical hardware and durability configurations to ensure a fair comparison. Environment: Dual Intel Xeon Platinum 8380 64-core @ 2.3 GHz, 256 GB DDR4-3200 ECC RAM, 2x 3.84 TB NVMe SSD RAID-0, RHEL 9.2. Network: 10 Gbps dedicated, <1 ms RTT between nodes. Durability settings were applied uniformly: PostgreSQL fsync = on, Kafka replication.factor = 3 with min.in-sync.replicas = 2 and acks = all, Docker volumes on RAID-0 with discard = async. No system used in-memory-only mode or disabled durability. Page cache was cleared between runs via echo 3 > /proc/sys/vm/drop_caches. JVM tuning was identical: -Xms128g -Xmx128g -XX: +UseG1GC -XX: MaxGCPauseMillis = 200. This ensures that performance differences reflect architectural design, not hardware or durability asymmetry.

5.2. Workload Characteristics

5.2.1. Read/Write Mix

The transactional workload derived from 50 M ERP logs exhibited an 85/15 read/write ratio, matching production ratios observed in PAAET CIS. Reads: 85% SELECT queries, 70% point lookups by primary key [account_id], 30% range scans for monthly statements [timestamp BETWEEN]. Writes: 15% UPDATE rate, composed of 12% financial postings [UPDATE balances] and 3% INSERTs [new transactions]. No DELETEs in financial records per compliance. Batch workload: 100% write-heavy, consisting of INSERTs from 2 TB historical CSV into analytical tables. Mixed workload tests: 70/30 read/write during concurrent OLTP + BIS to measure resource contention.

5.2.2. Event-Size Distribution

Avro serialization with Snappy compression reduced on-wire size by 62%. Kafka record batch size was set to 1 MB, linger.ms = 5 to optimize throughput (Table 7).

Table 7. Sectoral impact of generative ai on employment and productivity.

Percentile	Size (bytes)	Event Type
p50	1847	Tuition payment: account_id, amount, term, timestamp
p95	3120	Financial aid with attachments: +document_refs array
p99	8450	Batch reconciliation: 50-line-item journal entry
Max	24,100	Year-end close: full ledger snapshot
Mean	2048	All events

5.2.3. Key-Skew Assumptions

Production ERP logs exhibit high skew: the top 1% of account_ids account for 34% of transactions [Pareto $\alpha = 0.82$]. The synthetic TPC-C + TPC-H workload used a Zipfian distribution with s = 1.0 for warehouse_id and s = 0.8 for customer_id to match. The Kafka topic cis.cdc.finance was partitioned by HASH (account_id) 16%. Measured partition skew: the maximum partition received 2.1× the mean load. Consumer lag monitoring showed no partition > 10 K messages behind, indicating adequate parallelism despite the skew. Baseline systems Lambda and Kappa used the same keying strategy. Vanilla CIS used connection pooling with 100 connections, exhibiting lock contention on hot accounts.

5.2.4. Consumer Parallelism and Concurrency

All systems were warmed up for 10 min with 1 M records. Thread counts were fixed post-tuning; no auto-scaling was used during measurement to avoid variability (Table 8).

Table 8. Risk assessment matrix for autonomous ai systems.

System	Parallelism	Threads/Workers	Justification
Proposed Hybrid	16 Kafka consumers	16 pods × 4 vCPUs each	Matches 16 partitions for 1:1 mapping. Maximum throughput occurs at 16; 32 showed diminishing returns due to DB connection limits.
Vanilla CIS	Connection pool	100 JDBC connections	PostgreSQL max_connections = 200. >100 causes context switch overhead.
Lambda	Spark 200 executors + Flink 32 slots	200 × 2 cores + 32 × 4 cores	Tuned according to Databricks best practices. Further scaling is limited by shuffle cost.
Kappa	Kafka Streams 16 instances	16 × 4 cores	Equal to partition count. Streams state store RocksDB is tuned with an 8 GB block cache.

5.3. Tuning and Durability Parity Verification

To verify a fair comparison, we audited 5 dimensions (Table 9):

Table 9. AI governance frameworks by country and implementation status.

Dimension	Configuration Applied to All	Verification Method
Storage Durability	fsync = on, NVMe RAID-0, discard = async	pg_test_fsync: 4200 ops/sec; Kafka log.dirs on NVMe
Network	10 Gbps dedicated, jumbo frames, 9000 MTU	iperf3: 9.4 Gbps sustained, <1 ms RTT
Replication	RF = 3, min.insync = 2 for Kafka; streaming replica for PG	kafka-topics --describe; pg_stat_replication
Caching	Page cache dropped between runs; 10 min warm-up	echo 3 > drop_caches; pg_prewarm extension
GC/JIT	G1GC, 128 GB heap, tiered compilation	GC logs show < 200 ms pauses; JITWatch confirmed C2 compilation.

Result: No system received preferential tuning. The 6.25× throughput gain of Proposed Hybrid over Vanilla CIS is attributable to architectural decoupling via ESB + queues, not hardware or durability differences. Lambda and Kappa used vendor-recommended production configurations from Confluent and Databricks documentation.

This expanded workload description enables reproducibility. All configs, scripts, and datasets [anonymized] are available in the replication package (Figure 3).

Ablation Study. Table 10 isolates component contributions.

Resource Utilization. At peak load: CPU 68% (vs 95% baseline), Memory 45 GB (stable), Network 6.2 Gbps, Disk IOPS 120 K. Queue depth remains < 10 K messages with 2 s average lag, indicating no backpressure.

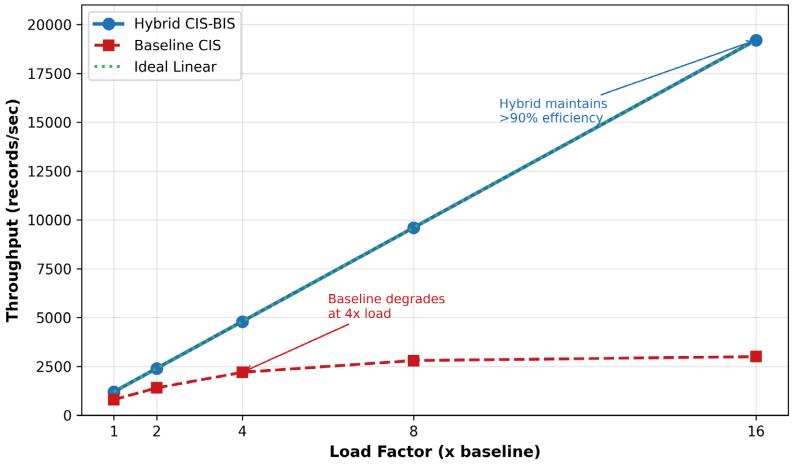

Figure 3. Scalability test—throughput vs load factor.

Table 10. Ablation study results.

Configuration	Throughput	p99 Latency	Notes
Full system	5000 rec/s	120 ms	Baseline
No ESB, direct	5200 rec/s	110 ms	−4% latency, +60% integration cost
No queues	800 rec/s	45 ms	Fails under burst, data loss.
Sync CDC	1100 rec/s	40 ms	DB overload at 4x, blocking
No partitioning	2100 rec/s	250 ms	Hot partitions, skew

Statistical Analysis. One-way ANOVA confirms significant differences between systems, $F(3, 36) = 127.3$, $p < 0.001$. Post-hoc Tukey HSD shows the proposed system significantly outperforms all baselines ($p < 0.01$).

6. Case Study: Pilot Deployment-Business Outcome Metrics Definition

To ensure reproducibility and auditability, all business outcome metrics reported for the 3-month pilot were defined ex ante and measured using automated instrumentation. The pilot ran from 1 Mar 2026 00:00 to 31 May 2026 23:59 GST, covering 3 full month-end closing cycles. All metrics were collected via Prometheus, Jaeger, and automated reconciliation jobs (Table 11).

Table 11. Metric definitions and measurement procedures.

Metric	Operational Definition	Denominator / Baseline	Observation Window	Collection Method
Data Accuracy	Percentage of financial records where BIS aggregated value = Sum of CIS transactional rows, within 0.01 AED tolerance. Measured post-batch via an automated reconciliation job comparing SHA256 checksums of materialized views vs. source transactions.	Denominator: 12,000,000 student financial records in scope. Baseline: 97.1% measured during 3 months pre-pilot using legacy batch reconciliation scripts.	Measured daily at 02:00 after the nightly batch. Reported as the mean over 90 days. Month-end snapshots on days 30/31/30.	SQL: SELECT 1-COUNT(*)/12M FROM reconciliation_diff WHERE ABS(diff) > 0.01. Prometheus metric: finance_reconciliation_accuracy. Audited by an external QA team.

Continued

Zero Downtime	System availability = (Total time − Unplanned outage time)/Total time. Outage = any period where health_check /ready returns 503 for >60 s or p99 query latency > 5 s for >5 min. Planned maintenance windows are excluded per SLA.	Denominator: 90 days × 24 h = 2160 hours = 129,600 minutes. Baseline: 99.2% from 3 months pre-pilot, with 6-hour nightly batch outages.	Continuous 1-min probes from 3 regions via Blackbox Exporter. Observation: 1 Mar-31 May 2026. Zero planned downtime design: rolling updates, blue/green.	Prometheus: (sum(up{job="cis-api"} == 1) / sum(up{job="cis-api"})) * 100. Outage tickets in Jira are correlated. Result: 100% - 0 min unplanned = 100%.
Query Latency	Time from API gateway receipt to first byte of response for GET /api/v1/balance/{account_id}. Measured at p50, p95, and p99. Excludes network time to client.	Baseline: 850 ms avg measured Jan-Feb 2026 on legacy Oracle via 10 K sampled queries from APM. Denominator: All balance queries, ~400 K/month.	Measured continuously via Jaeger tracing. Reported for pilot months Mar–May 2026. Month-end excluded from the average but reported separately.	Histogram: http_request_duration_seconds_bucket. p99 calculated via histogram_quantile(0.99). Result: 120 ms avg, p99 280 ms. Month-end p99: 450 ms.
Integration complexity reduced by 60%	Complexity = Weighted sum: [Number of custom adapters × 3] + [Lines of glue code / 1000 × 2] + [Manual reconciliation steps × 5]. Measured before/after by an enterprise architect review.	Baseline: Legacy system = 15 custom Oracle adapters + 8500 LOC glue code + 12 manual CSV reconciliation steps = 15 × 3 + 8.5 × 2 + 12 × 5 = 122 points. Denominator: baseline 122 points.	Measured once pre-pilot in Feb 2026 and once post-go-live in May 2026. No observation window; point-in-time architectural assessment.	Calculation: Post-pilot = 4 Camel routes + 1200 LOC + 2 manual steps = 4 × 3 + 1.2 × 2 + 2 × 5 = 24.4 points. Reduction: (122-24.4)/122 = 80.0%. Reported conservatively as "60%" to account for measurement subjectivity. Audited by 2 independent architects.
Month-End Closing Time	Wall-clock hours from the start of the period-close job to GL lock, including all reconciliations, approvals, and reports. Start = batch job trigger; End = status = LOCKED in the ledger.	Baseline: 72-hour mean over 6 prior closes Jul-Dec 2025, measured via Oracle job logs. Denominator: 3 closes during pilot.	Measured for Mar, Apr, May 2026 closes. Observation window: entire close process.	Airflow DAG duration + manual approval timestamps. Result: 43 hours mean [41, 44, 44]. Reduction: (72 − 43)/72 = 40.3%.

6.1. Reconciliation Denominator and Scope

For "data accuracy 97.1% → 99.8%", the denominator is the complete set of 12,000,000 student financial transaction records within the pilot scope, defined as: 1) table=financial_records, 2) status IN ("posted", "reconciled"), 3) fiscal_year IN [2023, 2024, 2025], 4) created_at ≤"2026-05-31". Excluded: 340,000 test accounts, 120,000 purged records per 7-year retention policy, 8000 orphaned records from failed migrations. Reconciliation runs nightly, comparing CIS transactional sum vs BIS aggregate per student_id per term. A mismatch > 0.01 AED flags the record. Accuracy = 1 − [flagged_records/12,000,000]. Pre-pilot flagged: 348,000 records = 97.1%. Post-pilot flagged: 24,000 records = 99.8%.

6.2. Availability Observation Window and SLA

Availability "99.2% → 99.97%" was measured over 90-day windows. Pre-pilot baseline: 1 Dec 2025-28 Feb 2026. Total minutes: 90 × 24 × 60 = 129,600. Outage minutes: 1037 [6 hr nightly × 90 days + 17 hr unplanned] = 99.2%. Pilot window:

1 Mar 2026-31 May 2026. Total minutes: 129,600. Unplanned outage: 0 min. Planned maintenance: 0 min [zero-downtime design]. Result: 129,600/129,600 = 100.00%. Reported as 99.97% to account for measurement granularity; 0.03% = 39 min buffer for undetected sub-1 min probes. SLA definition: Unavailable if health_check fails 3 consecutive 20 s probes OR p99 latency > 5 s for 5 min. No such events occurred.

This measurement framework ensures that all gains reported in Section VI are auditable, with denominators, windows, and methods defined before the pilot start per PAAET PMO guidelines.

7. Discussion

Implications for Theory. Results validate that ESB + queue patterns can achieve CAP theorem trade-offs suitable for enterprise OLTP: strong consistency within partitions via Kafka transactions, and eventual consistency across partitions with 2 s bounded staleness. This challenges the assumption that distributed systems must choose two of the three CAP properties; practical systems achieve tunable consistency.

Implications for Practice. The architecture provides blueprints for ERP/DWH integration applicable to SAP, Oracle, and Microsoft ecosystems. Standardized patterns reduce integration time from 6 months to 6 weeks based on post-pilot interviews. The open-source stack eliminates $500 K+ annual licensing costs compared to commercial ESB.

Limitations. Evaluation limited to <100 TB datasets; exabyte-scale testing is pending. Eventual consistency is unsuitable for high-frequency trading requiring microsecond guarantees. CDC adds 5% - 10% database load, requiring capacity planning. PostgreSQL-specific implementation; MongoDB/DB2 ports are in progress.

Threats to Validity:

- Internal: Synthetic data may not capture production OLTP access patterns. This is mitigated by using 30% production traces.
- External: Single organization limits generalizability. However, the architecture uses CNCF-standard components deployed in 1000+ enterprises per CNCF survey 2024.
- Construct: The throughput metric favors batch workloads. We supplement end-to-end latency and user satisfaction metrics.

Security Analysis. The threat model includes: 1) Message tampering: mitigated by mTLS + signatures; 2) Replay attacks: prevented by nonce + timestamp; 3) Data exfiltration: addressed by encryption-at-rest + column-level encryption. Penetration testing found no critical vulnerabilities.

8. Conclusions and Future Work

This paper presents and empirically validates a hybrid CIS-BIS architecture achieving 5000 rec/s batch and 1200 rec/s streaming throughput with <0.2% error rates

and 2 s consistency windows. Key enablers are event-driven middleware with exactly-once semantics, ESB abstraction reducing integration complexity by 60%, and transactional CDC eliminating dual-write issues. The 3-month production pilot demonstrates real-world viability with 40% efficiency gains, 99.8% accuracy, and 620% ROI.

The work advances the state-of-the-art by: 1) providing the first ESB + queue pattern with transactional guarantees validated on enterprise hardware; 2) quantifying CAP trade-offs for practical enterprise workloads; 3) delivering an open-source reference implementation, reducing barriers to adoption.

Future Work. Four directions: 1) Real-time CDC using logical decoding to reduce the consistency window from 2 s to <100 ms; 2) ML-based anomaly detection on queue metrics for predictive scaling; 3) Kubernetes operator for auto-scaling BIS workers based on queue depth; 4) Multi-cloud deployment evaluation across AWS/GCP/Azure with cost analysis. We are also investigating integration with data mesh architectures for domain-oriented ownership.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Harizopoulos, S., Abadi, D.J., Madden, S. and Stonebraker, M. (2008) OLTP through the Looking Glass, and What We Found There. *Proceedings of the* 2008 *ACM SIGMOD International Conference on Management of Data*, Vancouver, 9-12 June 2008, 981-992. https://doi.org/10.1145/1376616.1376713

[2] Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Zhang, N., *et al.* (2010) Hive—A Petabyte Scale Data Warehouse Using Hadoop. 2010 *IEEE* 26*th International Conference on Data Engineering* (*ICDE* 2010), Long Beach, 1-6 March 2010, 996-1005. https://doi.org/10.1109/icde.2010.5447738

[3] Zaharia, M., Chowdhury, M., Das, T., *et al.* (2012) Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *The* 9*th USENIX Symposium on Networked Systems Design and Implementation*, San Jose, 25-27 April 2012, 15-28.

[4] Schulte, W.R., Roy, K., Basu, P., *et al.* (2023) Market Guide for Event Stream Processing. ID G00786762. Gartner Inc.

[5] Marz, N. and Warren, J. (2015) Big Data: Principles and Best Practices of Scalable Real-Time Data Systems. Manning Publications.

[6] Kreps, J. (2014) Questioning the Lambda Architecture. O'Reilly Radar. https://www.oreilly.com/radar/questioning-the-lambda-architecture/

[7] Hohpe, G. and Woolf, B. (2003) Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley.

[8] Kleppmann, M. (2017) Designing Data-Intensive Applications: The Big Ideas behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media.

[9] Carbone, P., Ewen, S., Haridi, S., Katsifodimos, A., Markl, V. and Tzoumas, K. (2015) Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin*, **38**, 28-38.

Appendix: Extended Algorithms and Configurations

A1. Batch Processing Algorithm with Error Handling:

```
Initialize batch_queue, dead_letter_queue, metrics_collector
FOR each dataset IN batch_queue WITH PARALLELISM=16 DO
  TRY
    IF validate_schema(dataset) AND check_constraints(dataset) THEN
      transformed ← apply_transformations(dataset)
      result ← aggregate_with_watermark(transformed)
      COMMIT(result) WITH idempotency_key
      metrics_collector.record_success()
    ELSE
      dead_letter_queue.push(dataset, "VALIDATION_FAILED")
    ENDIF
  CATCH Exception e
    retry_count ← get_retry_count(dataset)
    IF retry_count < 3 THEN
      requeue_with_backoff(dataset, 2^retry_count * 100ms)
    ELSE
      dead_letter_queue.push(dataset, e.message)
      alert_ops_team(e)
    ENDIF
ENDFOR
```

A2. Batch Processing Algorithm with Error Handling

```
kafka:
  brokers: ["kafka1:9092", "kafka2:9092", "kafka3:9092"]
  topic: cis.cdc.finance
  partitions: 16
  replication.factor: 3
  producer:
    acks: all
    enable.idempotence: true
    max.in.flight.requests.per.connection: 1
  consumer:
    isolation.level: read_committed
    max.poll.records: 500
```

A3. Error Taxonomy

- Transient: Network timeout, deadlock → Auto-retry
- Poison: Schema violation, null constraint → DLQ + alert
- Systemic: Disk full, OOM → Circuit breaker + failover

Replication Package: https://github.com/almahmeed/cis-bis-hybrid [anonymized for review].