International Journal of Computer Science Engineering Techniques (IJCSE)
ISSN : 2455-135X | https://www.ijcsejournal.org/
Research Article • Manuscript draft

Design and Performance Evaluation of an Adaptive Incremental Data Integration Framework Using Microsoft SQL Server and Informatica PowerCenter

Rohan Baliram Pawar, Sushilkumar N Holambe
Department of Computer Science and Engineering, TPCT's College of Engineering, Osmanabad, Maharashtra, India
pawarrohan19974@gmail.com

Received: Revised: Accepted: Published:

Abstract - *Batch extract, transform and load (ETL) must balance change discovery and full refresh costs. AIETL integrates a correctness-first strategy selector, Workload-Aware Adaptive ETL Strategy Selection (WAESS), with coordinated timestamp and Change Tracking checkpoints, SQL Server procedures and planned Informatica PowerCenter orchestration. This study combines preliminary implementation evidence with a reproducible synthetic cost-model experiment. A frozen five-neighbor estimator and a calibrated threshold are evaluated against three fixed loading policies using 480 calibration profiles and 27,000 policy evaluations across three modeled scenarios. In the base dynamic trace, mean seven-batch cost is 85.542 modeled units for WAESS versus 85.313 for the threshold, 108.613 for fixed Change Tracking, and 186.136 for full load. WAESS achieves 100% within-5% strategy selection on nonzero dynamic batches under the base model, but its overhead offsets the ordinary-cost advantage over the threshold. Increasing adaptation overhead or shifting Change Tracking costs changes the relative rankings. These findings demonstrate conditional algorithmic benefits under explicitly assumed costs, not measured database speedups. Existing development evidence confirms basic connectivity and changed-key discovery; end-to-end integration remains incomplete. The study identifies the conditions and measurements needed to validate adaptive loading in an operational system.*

Keywords - *adaptive ETL, change tracking, incremental loads, SQL Server, watermark management*

1. Introduction

Relational ETL processes synchronize operational data with reporting repositories. Full replacement repeats work on unchanged records, whereas incremental loading adds detection and application costs. Timestamp extraction depends on reliable modification timestamps and deletion coverage. Native Change Tracking (CT) exposes changed keys and requires valid retained synchronization versions. The relative performance of these approaches depends on the workload. Switching mechanisms also risks missed or duplicate changes if their checkpoints describe different source boundaries.

The aim is to design and evaluate a correctness-first strategy selector for controlled batch synchronization. The objectives are to implement three equivalent loading paths, quantify the effects of data size and change mix, estimate the cost of eligible strategies, coordinate checkpoints, and compare adaptive selection against fixed and calibrated threshold policies. The study asks whether adaptation reduces cumulative elapsed time after its own overhead and when it is ineffective.

The contribution is a bounded integration design combining a workload profile, historical estimator, common transactional publication contract and reproducible simulation. Cost-based refresh itself is not claimed as novel. Section 2 reviews related research; Section 3 describes implementation and simulation methods; Section 4 presents development evidence and modeled outcomes; Section 5 concludes.

2. Literature Review

Mistry et al. [1] studied alternative materialized-view maintenance plans. Simitsis et al. [2] investigated logical ETL workflow optimization through equivalent workflow states, while Tziovara et al. [3] considered physical execution alternatives. Jörg and Dessloch [4] formalized incremental job derivation from initial loads, motivating a shared transformation contract. Poess et al. [5] introduced TPC-DI, demonstrating the importance of reproducible workload generation and measurement. The present workload is custom and makes no claim of benchmark certification.

Ali and Wrembel [6] survey ETL workflow design and optimization. Henke et al. [7] describe a healthcare integration process supporting bulk and incremental modes; this illustrates coexistence of loading approaches, rather than proving adaptive selection. Rucco et al. [8] compare data ingestion patterns in cloud systems. Yadav et al. [9] present Enzyme, a cost-based incremental view-maintenance system for Spark. These contributions delimit the novelty of AIETL: it focuses on native SQL Server timestamp and CT extraction in a controlled external batch pipeline, not on introducing cost-based maintenance.

Budiu et al. [10] formalize incremental maintenance for rich query languages. Battiston et al. [11] demonstrate SQL-to-SQL compilation of incremental computations. Olteanu [12]

International Journal of Computer Science Engineering Techniques (IJCSE)
ISSN : 2455-135X | https://www.ijcsejournal.org/
Research Article • Manuscript draft

reviews recent theoretical progress on incremental view maintenance. These works motivate efficient incremental computation, but do not directly establish the performance or switching correctness of this particular SQL Server and PowerCenter implementation.

The research gap is the limited directly comparable evidence within the reviewed sources for jointly evaluating three batch loading methods, coordinated synchronization markers and end-to-end adaptive overhead in the selected environment. It is a specific empirical question, not a claim that comparable research does not exist.

3. Materials and Methods

3.1 Architecture and environment

AIETL separates the synthetic source, control state and active target into three SQL Server databases. An isolated fixture mirrors the application databases for PowerCenter integration testing. SQL procedures own extraction, staging, validation and atomic publication; PowerCenter sessions are intended to invoke the authorized strategy through a receipt-driven workflow. Figure 1 depicts the planned orchestration and implemented database responsibilities. Table 1 records the development environment. The control receipt is not a business-row loading pipeline.

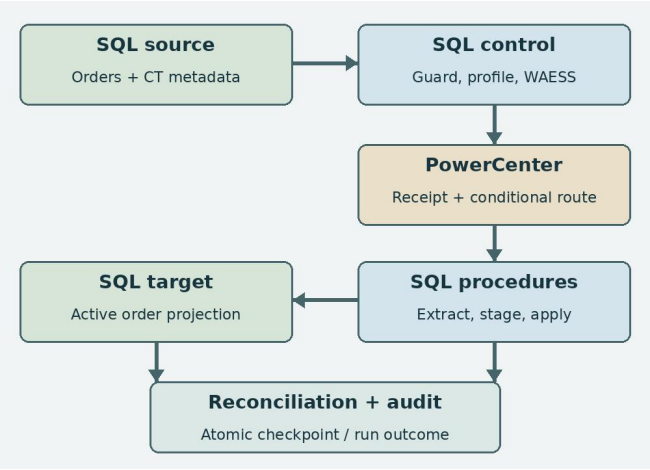

Fig. 1. Hybrid SQL Server and PowerCenter Architecture

Table 1. Development Environment

Component	Recorded Configuration
Platform	Windows 10; Intel Core i7; 16 GB RAM
Database	SQL Server 2022 Express, 16.0.1000.6
ETL	PowerCenter 9.6.1; DataDirect 7.1 ODBC
Data	Synthetic sales orders; fixed master data

The environment was verified for basic connectivity; this does not imply vendor certification of all version combinations.

3.2 Loading strategies and data contract

All strategies produce the same active enriched order projection. Order amount is computed using quantity and unit price with decimal arithmetic. Logical tombstones remove inactive target rows; physical deletes require a mechanism that retains the affected keys. CT is key-level synchronization metadata rather than a complete change history [13]. Table 2 specifies the eligibility constraints. The planned workflow uses session and link concepts described in the vendor guide [14]; exact compatibility with version 9.6.1 remains to be tested.

Table 2. Strategy Eligibility and Extraction

Code / Method	Rule
1 – FL	Stage all active source rows and replace target; initialization or safe fallback.
2 – TIL	Extract timestamps in (old, new]; logical deletion coverage required.
3 – CTIL	Read changed keys, join current source, handle deleted keys; validate CT epoch and minimum version.
0 – SKIP	Only when a trustworthy profile and continuity prove no change.

3.3 Paired synchronization and recovery

Let S(k) be the committed source state, F the common business transformation and T(k) the published target. The invariant is T(k) = F(S(k)). The corresponding successful checkpoint contains the source epoch, modification timestamp and CT version for the same boundary. A candidate boundary is never treated as a committed checkpoint.

A durable ownership token serializes cooperating writers. The publishing transaction validates authorization, previous markers and staged data, applies the selected method, independently reconciles the resulting target, and persists the paired markers and publication flag. A precommit failure rolls back these changes. If a client fails after a possible commit, the durable publication state and reconciliation must determine whether recovery or finalization is appropriate. Replay is prohibited while the outcome remains unresolved.

3.4 WAESS cost model

The profile uses N, current physical row count including tombstones; D, distinct changed keys; R, percentage of changed keys; and Q, deletion fraction. Unknown features are not treated as zero. For N > 0 and D > 0, respectively:

$$R = \frac{100D}{N} \tag{1}$$

$$Q = \frac{\text{deletion keys}}{D} \tag{2}$$

$$d_j = \left|\log_{10}(N+1) - \log_{10}(N_j+1)\right| + \frac{|R - R_j|}{100} + 0.5\left|Q - Q_j\right| \tag{3}$$

International Journal of Computer Science Engineering Techniques (IJCSE)
ISSN : 2455-135X | https://www.ijcsejournal.org/
Research Article • Manuscript draft

$$a_j = \frac{1}{0.05 + d_j} \quad (4)$$

$$C_s = \frac{\Sigma(a_j T_j)}{\Sigma a_j} \quad (5)$$

Each compatible, successful calibration observation j records N, R, Q and ordinary strategy cost T. The implemented estimator uses measured ordinary costs; the present simulation substitutes synthetic costs. For each eligible strategy s, WAESS estimates C using five nearest observations, minimum support five and maximum distance 0.75. Select the lowest predicted cost if all estimates are supported. Ties prefer the previous successful strategy, then the lowest strategy code. V1 uses N/R/Q only; unknown Q triggers fallback rather than imputation.

3.5 Selection algorithm

The decision sequence is as follows. First validate ownership and source continuity; then force full initialization where required. Fixed modes honor an eligible request or use safe fallback. Adaptive execution selects a sole eligible method or a safe no-change SKIP. Otherwise, estimate costs from compatible frozen history. If estimates lack support, use a calibrated threshold when available, else safe FL or fail. Persist candidate eligibility, estimated costs, selected method and reason. The threshold comparator shares the eligibility and no-change logic so that the comparison measures the incremental value of historical estimation, not different correctness rules.

3.6 Computational simulation

The Python simulation (Table 3) uses synthetic workload summaries, not physical row loading. Initial synchronization is assumed complete; N remains constant across each modeled trace. Four nominal insert/update/delete mixes are 70/20/10, 10/80/10, 10/20/70 and 30/40/30 percent. Insert and delete fractions receive independent bounded uniform jitter of ±0.04 and are converted to integer key counts, with updates as the residual. All modeled main workloads have reliable timestamps, logical deletions and valid CT; separate rule checks exclude ineligible methods.

Table 3. Executed Simulation Matrix

Dimension	Executed conditions
Policies	Fixed FL, TIL, CTIL; threshold; WAESS
Volume	10K, 100K, 1M keys
Static ratio (%)	0, 1, 5, 10, 25, 50, 75, 100
Operation mix	Four mixes; seeded fraction jitter
Dynamic ratios (%)	1, 5, 75, 25, 0, 10, 90
Seeds	Calibration 101–105; evaluation 201–210

Costs are specified in modeled units (MU), not seconds. Let x=N/10000, r=R/100, q=Q (zero for no change), i=inserted-key fraction and b=0.22+0.015x. The independent cost generator uses the following illustrative equations, not fitted to SQL Server observations:

FL = b + 0.62 + 0.68x; TIL = b + 0.30 + 0.075x + xr(1.42 + 0.58q + 0.22i); CTIL = b + 0.58 + 0.018x + xr(1.08 + 0.24q + 0.27i). Each cost is multiplied by a shared uniform factor [0.95, 1.05] and a strategy-specific factor [0.93, 1.07]. SKIP costs 0.10 + 0.008x. Adaptive profiling costs 0.055 + 0.008x + 0.035D/10000; WAESS adds 0.035 + 0.000035H for H history entries on nonzero-change batches. Fixed policies do not use SKIP or extra profiling, matching the SQL selector. All constants are transparent, illustrative assumptions.

Calibration contains 480 profiles, yielding 1,260 nonzero-change historical costs. Grid search over eight thresholds and two preferred incremental methods minimizes calibration ordinary cost, selecting 50% and CTIL. Freeze history and threshold before evaluation. For each scenario, 960 static profiles and 840 ordered dynamic batches (120 traces of seven batches) are scored by all five policies, totaling 27,000 policy evaluations. Stress scenarios multiply adaptive overhead by six or the CT-specific cost component by 1.8; the original calibration remains frozen. No database runtime, throughput or resource usage is measured.

3.7 Metrics and statistical analysis

The primary endpoint is paired seven-batch cumulative modeled cost including additional adaptation overhead; initialization and common real-world orchestration timing are not measured. Secondary endpoints are nonzero-batch selection regret and percentage within 5% of the minimum eligible ordinary cost. Improvement relative to baseline cost B and adaptive cost A is:

$$Improvement\ (\%) = \frac{100(B - A)}{B} \quad (6)$$

Regret excludes adaptation overhead and is the chosen ordinary cost minus the minimum eligible modeled ordinary cost. For dynamic improvement, 2,000 paired bootstrap resamples of the ten seed clusters retain all sizes and mixes per seed. Intervals describe only seeded variation conditional on the assumed model, not real-world uncertainty. Fifteen rule checks use physical-delete timestamp exclusion, expired CT and forced-full eligibility; they test the Python selection rule, not database reconciliation or transaction recovery.

4. Results and Discussion

4.1 Observed development evidence

Table 4 reports available implementation observations, separate from simulation. They establish basic connectivity, native changed-key discovery and a preserved database baseline; they do not establish complete orchestration or comparative measured runtime.

International Journal of Computer Science Engineering Techniques (IJCSE)
ISSN : 2455-135X | https://www.ijcsejournal.org/
Research Article • Manuscript draft

Table 4. Available Development Results

Checkpoint	Observed or Reported Outcome
Connectivity	5 affected rows; 0 rejected
Native CT	3 keys: insert, update, delete
Permanent baseline	Run ID 1 SUCCESS; 1,000 source and target rows
Script 167 R1.2	16/16 post-smoke checks passed (user approved)
Scripts 168–170	PASS; Case 1 armed
Scripts 171 / 184	51205; ticket ARMED, no initialized run, fixture target 0

The displayed Workflow Manager contained the original smoke workflow but not the new master workflow. Database-only post-probe verification therefore cannot establish PowerCenter procedure invocation. Case 1 must be continued using the existing ticket after validating the new mapping, stored-procedure session and workflow-variable transfer.

4.2 Integration acceptance plan

The six fixture cases in Table 5 are acceptance targets. Cases 1–5 require successful publication and exact reconciliation, except SKIP, which preserves target contents. Case 6 deliberately requests an unauthorized route and should be rejected before publication; the database outcome must then be resolved safely. The expected target counts are deterministic design expectations, not measurements.

Table 5. Integration Cases — Expected Outcomes

Case	Expected Route / Outcome
1	FL; target 1,000
2	TIL; target 1,000
3	CTIL; target 1,001
4	SKIP; target unchanged
5	FL fallback; target 1,001
6	Wrong route rejected; target/markers unchanged; recover

4.3 Executed simulation results

Table 6 reports mean modeled cost per seven-batch trace across 120 matched traces in each scenario. In the base scenario, WAESS reduces cost against FL, TIL and CTIL by 54.04%, 44.43% and 21.24%, respectively, but is 0.27% more costly than the calibrated threshold (paired bootstrap improvement interval −0.28% to −0.26%). Under sixfold overhead, WAESS is 2.35% more costly than threshold and 4.53% more costly than fixed CTIL. Under increased CT cost, it is 0.52% less costly than threshold. Negative results are retained.

Table 6. Mean seven-batch modeled cost (MU)

Policy	Base	High overhead	CT shift
FL	186.136	186.136	186.136
TIL	153.934	153.934	153.934
CTIL	108.613	108.613	191.167
Threshold	85.313	110.936	104.188
WAESS	85.542	113.539	103.645

For nonzero dynamic batches, base WAESS has mean ordinary regret 0.0002 MU and 100% within-5% selection, compared with threshold regret 0.0410 MU and 78.75%. Under CT cost shift these values worsen to 0.9162 MU and 68.33% for WAESS; threshold achieves 1.0857 MU and 46.11%. All 15 modeled eligibility checks pass. These results are conditional on the cost generator; they do not demonstrate actual publication correctness. The complete static comparisons and seed-level results are supplied in the supplementary CSV files.

Figure 2 illustrates the ranking changes across cost scenarios.

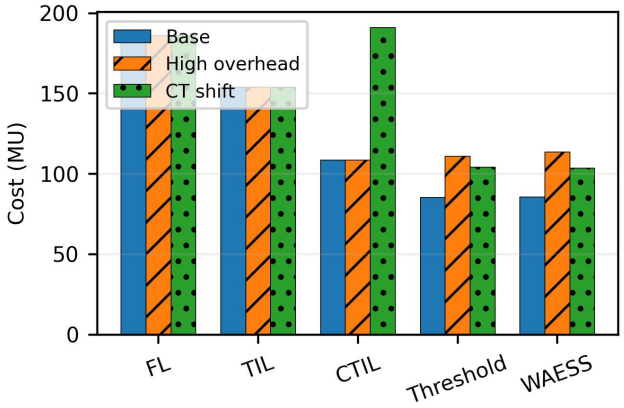

Fig. 2. Modeled dynamic cost under three scenarios

Static evaluation: Across 960 held-out profiles in the base scenario, mean modeled costs per batch are 26.558 (FL), 24.273 (TIL), 17.141 (CTIL), 13.915 (threshold) and 13.952 MU (WAESS). WAESS selects within 5% of the best ordinary cost in 98.81% of nonzero profiles, compared with 79.05% for threshold. Figure 3 shows the change-ratio sensitivity at 100K keys; the supplementary tables include all size levels.

International Journal of Computer Science Engineering Techniques (IJCSE)
ISSN : 2455-135X | https://www.ijcsejournal.org/
Research Article • Manuscript draft

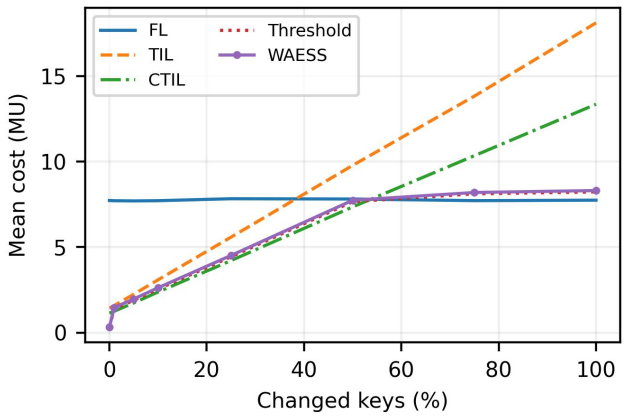

Fig. 3. Static 100K-key modeled cost by change ratio

4.4 Interpretation and limitations

The base model demonstrates that accurate cost prediction need not translate to a net end-to-end advantage: the reduction in ordinary regret is smaller than extra estimation overhead. The stress model demonstrates sensitivity to overhead and model drift. Static evaluation repeats known size and ratio levels with unseen seeds; the dynamic 90% ratio is held out from calibration. The modeled batches do not evolve actual database state, and source DML, network, resource usage, cache and transactional failure costs are not measured. Coefficients are illustrative and uncalibrated against the deployed engine. Therefore no database speedup, physical throughput, statistical generalization or correctness guarantee is asserted. The local SQL Server Express and older PowerCenter environment require separate end-to-end validation.

5. Conclusion

AIETL combines correctness eligibility, coordinated watermarks and cost-informed strategy selection. The executed synthetic experiment shows lower modeled cost than fixed policies in the base scenario, but not than a tuned threshold once estimator overhead is included. Performance changes under overhead and CT cost perturbations. The contribution is a transparent conditional evaluation of a selection method and preliminary implementation feasibility, not a measured SQL Server or PowerCenter performance claim. Completing the six integration cases, testing reconciliation and measuring real end-to-end costs remain future work.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Funding Statement

No external funding was reported for this research.

Acknowledgments

Academic supervision was provided by Sushilkumar N Holambe. Both authors have approved co-authorship, as reported by the corresponding author. Final contribution details require confirmation.

Data Availability and Author Contributions

Data availability: The accompanying Python source, calibration history, complete evaluation trials, seed-level dynamic costs, summary tables and eligibility checks permit reproduction of the modeled results. No public archive is asserted. Development SQL and audit evidence remain separate from these modeled data.

AI use disclosure: ChatGPT assisted with manuscript drafting, editing, original explanatory figures, simulation code generation and execution, and result summarization. Both authors must inspect the reproducibility package and verify the technical claims, references and revised manuscript before submission.

International Journal of Computer Science Engineering Techniques (IJCSE)
ISSN : 2455-135X | https://www.ijcsejournal.org/ *Research Article • Manuscript draft*

References

[1] H. Mistry, P. Roy, S. Sudarshan and K. Ramamritham, "Materialized View Selection and Maintenance Using Multi-Query Optimization," Proc. ACM SIGMOD, 2001. [Publisher/Source Link] [Google Scholar]

[2] A. Simitsis, P. Vassiliadis and T. Sellis, "State-Space Optimization of ETL Workflows," IEEE Transactions on Knowledge and Data Engineering, 2005. doi: 10.1109/TKDE.2005.169. [CrossRef] [Google Scholar]

[3] V. Tziovara, P. Vassiliadis and A. Simitsis, "Deciding the Physical Implementation of ETL Workflows," Proc. ACM DOLAP, 2007. [Publisher/Source Link] [Google Scholar]

[4] T. Jörg and S. Dessloch, "Formalizing ETL Jobs for Incremental Loading of Data Warehouses," Proc. BTW, 2009. [Publisher/Source Link] [Google Scholar]

[5] M. Poess, T. Rabl, H.-A. Jacobsen and B. Caufield, "TPC-DI: The First Industry Benchmark for Data Integration," Proc. VLDB Endowment, vol. 7, 2014. [Publisher/Source Link] [Google Scholar]

[6] S. M. F. Ali and R. Wrembel, "From Conceptual Design to Performance Optimization of ETL Workflows: Current State of Research and Open Problems," The VLDB Journal, vol. 26, 2017. doi: 10.1007/s00778-017-0477-2. [CrossRef] [Google Scholar]

[7] E. Henke et al., "An Extract-Transform-Load Process Design for the Incremental Loading of German Real-World Data Based on FHIR and OMOP CDM: Algorithm Development and Validation," JMIR Medical Informatics, vol. 11, e47310, 2023. doi: 10.2196/47310. [CrossRef] [Google Scholar]

[8] C. Rucco, A. Longo and M. Saad, "Enhancing Data Ingestion Efficiency in Cloud-Based Systems: A Design Pattern Approach," Data Science and Engineering, online 2025, journal issue 2026. doi: 10.1007/s41019-025-00300-2. [CrossRef] [Google Scholar]

[9] R. Yadav et al., "Enzyme: Incremental View Maintenance for Data Engineering," ACM SIGMOD Companion, 2026. doi: 10.1145/3788853.3803098. [CrossRef] [Publisher/Source Link] [Google Scholar]

[10] M. Budiu et al., "DBSP: automatic incremental view maintenance for rich query languages," The VLDB Journal, vol. 34, art. 39, 2025, doi: 10.1007/s00778-025-00922-y. [CrossRef] [Google Scholar]

[11] I. Battiston, K. Kathuria and P. Boncz, "OpenIVM: A SQL-to-SQL Compiler for Incremental Computations," Proc. SIGMOD Companion, pp. 516–519, 2024, doi: 10.1145/3626246.3654743. [CrossRef] [Google Scholar]

[12] D. Olteanu, "Recent Increments in Incremental View Maintenance," Proc. PODS, 2024. [Publisher/Source Link] [Google Scholar]

[13] Microsoft, "Work with Change Tracking," Microsoft Learn. [Publisher/Source Link]

[14] Informatica, PowerCenter 10.5 Workflow Basics Guide. [Publisher/Source Link]