Street Art & Urban Creativity | Vol. 11, No. 8, 2025 | ISSN 2183-9956
Street Art & Urban Creativity. Scientific Journal

Unified RevOps Automation: Architecture for Real-Time Cross-Cloud GTM Data Synchronization

Imran Basha Shaik
14154 sully lane Frisco Texas 75033
E-mail: Imran.shaik159@gmail.com
https://orcid.org/0009-0002-1210-5646

Abstract

Revenue operations (RevOps) teams run the go-to-market (GTM) motion on a federation of cloud applications: a CRM, a marketing automation platform, a billing or ERP system, and a warehouse or customer data platform. Each system owns a different slice of the customer record, each lives in a different cloud with its own API quota and failure modes, and each is edited by people and automations at the same time. The usual integration patterns, namely scheduled batch ETL, point-to-point webhooks and polling-based iPaaS flows, trade freshness, completeness and API cost against one another and give no convergence guarantee. This paper proposes Unified RevOps Automation (UROA), a five-layer architecture for real-time cross-cloud GTM data synchronization. UROA captures changes at the source with change data capture and hybrid logical clock stamps, stores them in a durable partitioned event log, merges concurrent edits with a field-ownership-aware deterministic rule, dispatches updates through a rate-limit-aware scheduler that coalesces, batches and prioritises, and closes the loop with anti-entropy reconciliation. We formalise the synchronization problem, derive a stability condition for API-constrained delivery and a backlog-drain bound for outages, and evaluate UROA in a discrete-event simulation of four clouds, 72,156 writes per run, two traffic bursts, two target outages and clock skew of 1.5 s, with five random seeds. UROA delivered updates with a median latency of 0.30 s (batch ETL: 452 s; polling iPaaS: 68 s), lost no updates (point-to-point webhooks lost 18.3% of deliveries and left 36.0% of records divergent), and held the time-averaged share of divergent fields at 0.77% (19.9% to 26.0% for the three baselines), at 2.29 API calls per write. Ablations show that coalescing and batching are the decisive components; hybrid-logical-clock stamping with ownership-aware merging and reconciliation give smaller gains that are not statistically resolved in this setting, and priority scheduling gave none. UROA's p99 latency (107.9 s) is dominated by outage backlog and is worse than that of webhooks, which simply drop those updates. Because the evaluation is a simulation, it characterises the architecture's behaviour and trade-offs rather than the performance of any vendor product.

Keywords: Revenue operations, GTM data synchronization, Change data capture, Hybrid logical clocks, Eventual consistency, API rate limiting, Anti-entropy reconciliation

1. Introduction

A modern go-to-market organisation does not run on one system. A prospect is created in a marketing automation platform, qualified and converted in a CRM, billed in a subscription or ERP system, and analysed in a warehouse or customer data platform. Every one of these systems holds a copy of some part of the same customer: an account name, a lifecycle stage, an owner, an engagement score, a payment status, a contract value. The business processes that matter to revenue cut across those copies. A sales representative should not call an account that has just filed a billing dispute. A marketing campaign should not send a renewal offer to a customer whose contract was upgraded an hour ago. A forecast should not count an opportunity whose stage was reverted in the CRM but not in the warehouse. These are synchronization problems, and in the RevOps literature and practice they are the daily cost of fragmented systems, "trapped signals" and manual reconciliation spreadsheets.

The synchronization problem has a systems character that is often hidden by integration vendors' product language. The systems sit in different clouds, so every update crosses a wide-area link with its own latency distribution and failure modes. Every system exposes an API with

a quota, a throttling policy and sometimes a bulk interface with different cost semantics. Every system is written to by humans, by its own automations and by the integration layer itself, so concurrent writes to the same field are routine rather than exceptional. Every system keeps its own clock, and the clocks disagree by amounts that are small for human purposes and large for last-writer-wins resolution of edits made a second apart. And the load is not uniform: a campaign launch, an end-of-quarter push or a data import produces bursts that are several times the baseline rate, exactly when the business most wants the data to be fresh.

The three dominant integration patterns each solve part of this and fail on another part. Scheduled batch extraction-transformation-loading (ETL) is efficient in API calls and simple to reason about, but its staleness is by construction half the schedule period or more. Point-to-point webhooks are fast in the common case, but they are fan-out calls with no queue, so quota exhaustion, a target outage or a deploy turns into dropped updates and silent divergence. Polling iPaaS flows add a queue and retries, but the poll interval puts a floor under latency, the retry policy is usually fixed rather than load-aware, and a burst that exceeds the quota produces a backlog that drains for tens of minutes. None of the three provides a statement of what the system converges to after a failure, and none reconciles replicas that have already diverged.

This paper argues that the problem should be treated as a replicated-data and flow-control problem rather than an integration-mapping problem, and proposes an architecture that combines four techniques from distributed systems in a way suited to GTM data. Changes are captured as an ordered, replayable event stream. They are stamped with hybrid logical clocks so that causally later edits order after earlier ones even when wall clocks disagree. Concurrent edits are merged with a deterministic rule that gives the field's owning system precedence inside a conflict window. And delivery to each target is scheduled against that target's quota, with coalescing of superseded updates, batching through bulk endpoints and priority classes, while an anti-entropy process repairs whatever divergence remains. We call the resulting architecture Unified RevOps Automation (UROA).

The contributions are five. (i) We formalise real-time GTM synchronization as a constrained delivery problem with explicit freshness, completeness, convergence and cost objectives. (ii) We specify a five-layer architecture, UROA, and its merge, scheduling and reconciliation algorithms. (iii) We derive a stability condition for quota-constrained delivery and a backlog-drain bound for outages, which together explain the behaviour of the baselines under bursts. (iv) We provide a reproducible discrete-event evaluation harness that compares UROA with batch ETL, point-to-point webhooks and polling iPaaS, together with four ablations and a burst-severity sweep. (v) We report where the architecture does not help: priority scheduling had no measurable effect, the gains of hybrid-logical-clock merging and reconciliation are small and statistically unresolved here, and the tail latency is dominated by outage backlogs rather than by scheduling.

2. LITERATURE REVIEW

A. Data Fragmentation in Revenue Operations

RevOps consolidates sales operations, marketing operations and customer success operations under one accountability for the revenue process. The enabling data layer is, however, rarely consolidated. Practitioner accounts consistently describe the same pattern: each function's tooling accumulates its own copy of account and contact data, integrations are built point to point as needs arise, and the number of integration paths grows roughly with the square of the number of systems. The architectural response that the industry has converged on is a unifying orchestration layer, usually a combination of a customer data platform and a CRM, that centralises identity and events and connects analytics, targeting and personalisation.

B. Change Data Capture and Event Logs

Log-based change data capture reads a system's change feed (a database log, an API change stream or a webhook) and turns state changes into events. Treating the event log as the system of record for integration, with consumers that maintain derived views, is the core idea of log-centric data architecture [1]. Distributed commit-log systems provide partitioned, replayable, ordered-per-partition storage with consumer offsets [7], which gives three properties that GTM integration needs: durability across consumer outages, back-pressure absorption during bursts, and the ability to replay history to rebuild a target. Ordering is guaranteed only within a partition, which makes the partition key a design decision; partitioning by record and field preserves the order that merge logic needs while allowing parallelism.

C. Time, Order and Consistency

Reasoning about concurrent updates starts from the happened-before relation and logical clocks [3]. Physical timestamps are attractive because they are meaningful to humans, but commodity clocks drift and disagree, and systems that depend on them for ordering must either bound clock uncertainty explicitly [8] or accept misordered updates. Hybrid logical clocks combine a physical component with a logical counter so that timestamps stay close to physical time while respecting causality [2]. In SaaS integration the relevant clocks are the timestamps that each application stamps on its own records, and these are typically second-granular and not synchronised to a common source. Eventual consistency, in which replicas converge if updates stop, is the practical target for wide-area replicated data under availability requirements [5], [9], and its design space and limits are well documented [16]. Conflict-free replicated data types achieve convergence by construction through merge functions that are commutative, associative and idempotent [4], and the Dynamo-style systems that popularised last-writer-wins and vector clocks showed the operational value of read-repair and anti-entropy [6]. Epidemic and Merkle-tree-based anti-entropy are the standard tools for detecting and repairing divergent replicas at low cost [12], [13].

D. API-Constrained Delivery

SaaS APIs impose quotas, typically as calls per unit of time per tenant, and many expose bulk endpoints that process tens to hundreds of records per call. Delivery to such a target is a queueing problem with a service capacity set by the quota and a service unit set by the batch size. Standard queueing theory describes the mean waiting time of a single-server queue with general service times [11], and Little's law links the mean number of items in flight to the arrival rate and the mean time in the system [10]. In practice the dominant engineering lessons are that retries must be backed off and jittered to avoid synchronised retry storms [15], and that rejected requests (HTTP 429) consume quota and capacity even though they deliver nothing. Coalescing, which collapses multiple pending updates to the same field into the latest one, reduces load without changing the final state if the merge function is deterministic.

E. Identity Resolution

A precondition for synchronizing a record across systems is knowing that two records refer to the same entity. Probabilistic record linkage [14] provides the standard statistical foundation, and production systems implement it through deterministic keys, fuzzy matching and a persistent cross-reference. Identity resolution is out of scope for this paper; we assume that every system's records are mapped to a canonical entity identifier and concentrate on the propagation of field-level changes once the mapping exists.

3. RELATED WORK

Integration patterns in practice. Three families of tooling dominate GTM integration. Batch ETL and reverse ETL tools extract from a warehouse or application on a schedule and load into downstream systems; they are economical with API calls because they deduplicate and use bulk interfaces, and their staleness is dictated by the schedule. Event-driven point-to-point integrations, typically built from webhooks and serverless functions, minimise latency and are straightforward to build but have no durable buffer. Integration platforms as a service provide visual flows with polling or webhook triggers, built-in retries and connectors; they add a queue but typically process items one by one in arrival order. The evaluation in this paper models each of these three families by its mechanism rather than by a specific vendor.

Log-centric data integration. The idea of a unified log as the integration backbone [1], [7] is the closest architectural precedent for UROA's first two layers. UROA differs in the treatment of the target side: it adds a merge layer that understands field ownership, a scheduler that is aware of each target's quota and bulk semantics, and a reconciliation layer for repair.

Replicated data and conflict resolution. CRDT-style merging [4] and Dynamo-style replication [6] provide the theory and the operational practice of convergent replicas. GTM data differs from the key-value stores that those works address in two ways. The replicas are third-party applications that cannot run a custom merge function internally and must be updated through their public APIs, so merging has to happen in the integration layer; and the fields have natural owners, because each application is the authoritative system for part of the record. UROA exploits the second property in its merge rule.

Time in distributed integration. Hybrid logical clocks [2] and bounded-uncertainty clocks [8] show how to order events without a perfectly synchronised clock. UROA uses a hybrid-logical-clock-style stamp at capture so that an edit made after the origin system has seen a prior value is

ordered after it.

Limits of the existing evidence. We are not aware of an open, reproducible evaluation that compares the three integration families on identical workloads with injected bursts and outages. This paper contributes such a simulation, and we are explicit that a simulation can establish mechanisms and trade-offs but not production numbers.

4. METHODOLOGY

A. System Model and Problem Statement

Let $\mathcal{S} = \{1, \ldots, m\}$ be the set of systems and $\mathcal{K}$ the set of synchronized keys, where a key $\kappa = (r, f)$ is a canonical record r and a field f. Field f has a single owning system $\omega(f) \in \mathcal{S}$, the system of record for that field, though other systems may also edit it. A write is a tuple $w = (\kappa, o, \tau, v)$ issued at true time t_w in origin system o, carrying a timestamp τ and a version identifier v. Each system s keeps a replica state $\sigma_s(\kappa)$, the write it currently holds for key κ. When w is issued, o applies it locally and the integration layer must deliver it to every other system. The delivery latency of the pair (w, s) is

$$L(w, s) = t_s^{\text{apply}}(w) - t_w \qquad (1)$$

where $t_s^{\text{apply}}(w)$ is the time at which s holds w or a write that supersedes it. A delivery that never completes has $L = \infty$ and is counted as lost. The replicas are converged at time t if $\sigma_s(\kappa) = \sigma_{s'}(\kappa)$ for all s, s' and all κ. The divergence at time t is the share of written keys on which at least two replicas disagree,

$$\Delta(t) = \frac{|\{\kappa:\ \exists s, s',\ \sigma_s(\kappa) \neq \sigma_{s'}(\kappa)\}|}{|\mathcal{K}_w|} \qquad (2)$$

where $\mathcal{K}_w$ is the set of keys written so far. The synchronization objective has four parts: low delivery latency, no lost deliveries, low time-averaged divergence $\bar{\Delta} = \frac{1}{T}\int_0^T \Delta(t)\, dt$, and eventual convergence to the intended value after updates stop, at low API cost. The API cost is the number of API calls per write,

$$\Gamma = \frac{\text{API calls issued, including rejected and failed calls}}{\text{writes}} \qquad (3)$$

The intended value of a key is defined by a policy that an operator can state in business terms. If two writes to the same key come from different systems and are issued within a conflict window δ of each other, the write issued by the owning system is intended to prevail; otherwise the later write, in true time, is intended to prevail. Writing $\pi(w)$ for the preceding write on the same key,

$$\text{win}(w, \pi(w)) = \begin{cases} \text{the owner's write,} & o \neq o_\pi \text{ and } t_w - t_\pi < \delta \text{ and exactly one of them is from } \omega(f) \\ w, & \text{otherwise} \end{cases} \qquad (4)$$

This intended outcome is the ground truth against which we score the replicas after convergence; the fraction of keys whose replicas do not all hold the intended write is the policy-violation rate ϵ.

B. Architecture Overview

UROA has five layers, shown in Figure 1. L1 (capture) reads each system's change feed through CDC or a webhook, stamps each change with a hybrid logical timestamp, and assigns an idempotency key built from the system, the key and the version. L2 (event log) is a durable, replayable, partitioned log; the partition key is the canonical (r, f) so that all writes to one key are ordered together. L3 (merge and schedule) merges concurrent writes with a deterministic rule and schedules delivery to each target against its quota. L4 (apply) performs idempotent upserts through the target's public API, with exponential backoff and a circuit breaker. L5 (reconcile) periodically compares replicas through digests and re-enqueues the differences. The layers are separable: a deployment can adopt L1 and L2 first and add L3 to L5 later, which is how the ablations in Section VI are structured.

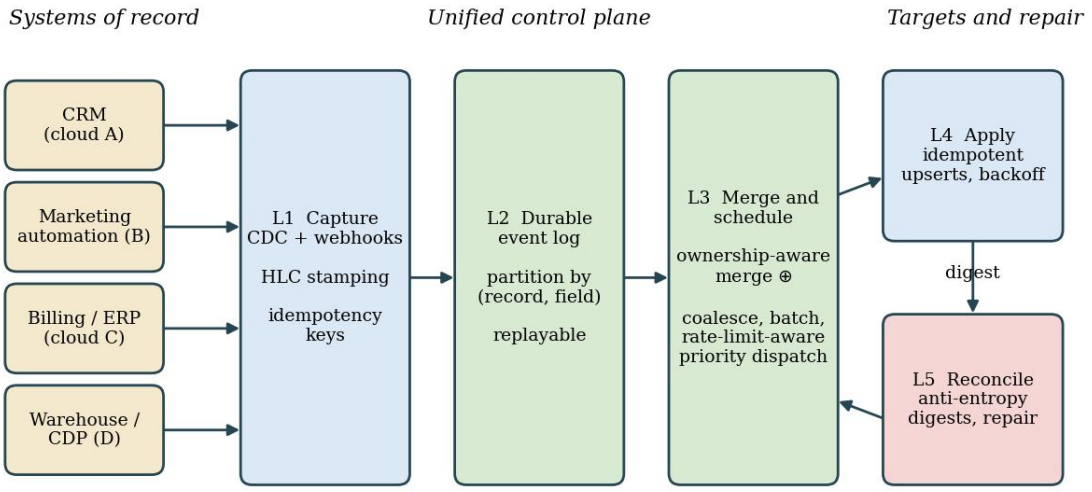


Figure 1: The five-layer UROA architecture. Changes captured from the systems of record flow through a durable event log into a merge-and-schedule layer, are applied idempotently to the targets, and are verified by an anti-entropy reconciliation layer that feeds repairs back into the scheduler.

C. Timestamping and Causal Ordering

Each system stamps its writes with its own clock, $\tau = t_w + \theta_o$, where θ_o is the clock offset of the origin. Offsets across SaaS vendors are not synchronised, and we model them as independent draws $\theta_s \sim \mathcal{N}(0, \sigma_\theta^2)$. If two edits to the same key are issued g seconds apart in true time by different systems, their timestamps can be misordered whenever $\theta_o - \theta_{o'}$ exceeds g in the wrong direction. With independent offsets the probability of a misordering for a gap $g \geq 0$ is

$$\Pr[\text{misorder}|g] = \Phi\left(\frac{-g}{\sqrt{2}\,\sigma_\theta}\right) \qquad (5)$$

where Φ is the standard normal cumulative distribution function. For $\sigma_\theta = 1.5\,\text{s}$ and $g = 1\,\text{s}$ this is about 0.32. A hybrid logical clock removes this error for causally ordered edits. When origin o issues a write on key κ after it has already applied the previous write π, UROA stamps

$$\tau = \max(t_w + \theta_o,\ \tau_\pi + \varepsilon) \qquad (6)$$

with a small $\varepsilon > 0$, so that the new write orders after the one it has seen regardless of clock offsets [2]. It does not correct blind concurrent edits, for which no timestamp can establish an order, and that is the job of the ownership rule.

D. Merge Rule

For two candidate writes a and b on the same key, the merge $a \oplus b$ is defined on timestamps τ_a, τ_b and origins:

$$a \oplus b = \begin{cases} \text{the owner's write}, & o_a \neq o_b,\ |\tau_a - \tau_b| < \delta,\ \text{exactly one is from } \omega(f) \\ \arg\max_{c \in \{a,b\}} (\tau_c, v_c), & \text{otherwise} \end{cases} \qquad (7)$$

The second branch is a total order on (τ, v) and is therefore commutative, associative and idempotent; a replica that applies only this branch converges whenever all writes are eventually delivered [4]. The first branch is deterministic and commutative but the relation $|\tau_a - \tau_b| < \delta$ is not transitive, so $\oplus$ is not associative in general. We therefore claim convergence only empirically for the full rule and rely on the reconciliation layer (Section IV.F) to resolve residual cases; Section VI reports the observed rates. The same function is used at three places so that behaviour is consistent: at the target when an update is applied, in the scheduler when pending updates are coalesced, and in the reconciler when the best value is recomputed from replicas.

E. Rate-Limit-Aware Scheduling

Each target s has a quota of C_s calls per second and accepts batches of up to B_s items per call. Let λ_s be the arrival rate of items for s and $\eta_s \in [0,1)$ the fraction of arrivals that are eliminated by coalescing, meaning that a pending update to the same key is replaced by the merged value. The offered call rate and the utilisation are

$$u_s = \frac{\lambda_s (1 - \eta_s)}{B_s\, C_s} \qquad (8)$$

and delivery to s is stable only if $u_s < 1$. A per-item, uncoalesced FIFO dispatcher has $B_s = 1$ and $\eta_s = 0$, so $u_s = \lambda_s / C_s$, and it saturates as soon as the item arrival rate exceeds the quota. A batching and coalescing dispatcher has capacity of $B_s C_s / (1 - \eta_s)$ items per second, which is an order of magnitude higher for the bulk sizes used here. Items waiting at a stable target see a mean queueing delay that, in the single-server approximation with deterministic slot spacing $1/C_s$ and service coefficient of variation c_v, follows the Pollaczek–Khinchine form [11]

$$\mathbb{E}[W_q] \approx \frac{u_s}{2(1-u_s)} \left(1 + c_v^2\right) \frac{1}{C_s} \qquad (9)$$

which shows why latency is flat at moderate utilisation and explodes near $u_s = 1$. UROA keeps three queues per target, one for each priority class, and within a class dispatches the oldest pending key first. The priority class of a field is a business decision, for example critical for lifecycle stage and payment status.

F. Outages, Retry and Reconciliation

When a target is unavailable for a duration D, a durable queue accumulates a backlog $Q_0 = \lambda_s D$ items. Once the target returns, it drains at the net rate $B_s C_s - \lambda_s$, so the backlog-drain time and the worst-case latency of an item that arrived at the start of the outage are

$$T_{drain} = \frac{\lambda_s D}{B_s C_s - \lambda_s}, \qquad L_{max} \approx D + T_{drain} \qquad (10)$$

Equation (10) says that an architecture cannot reduce the latency of updates that arrive during an outage below D unless it routes around the outage; it can only make the drain short. The scheduler retries failed calls with capped exponential backoff, $\min(2^k, 20)$ seconds after the k-th failure, and never drops a queued update. The reconciler runs every T_r seconds and, for each written key, recomputes the best write across all replicas by applying (7), and re-enqueues that write to every replica that does not hold it. The cost of the reconciler is the number of repairs, which is observable and which we report.

5. EXPERIMENTAL SETUP

A. Why a Discrete-Event Simulation

The evaluation compares architectures on identical write streams with identical injected bursts, outages and clock offsets, and requires the counterfactual intended value of every key. That cannot be controlled in production SaaS tenants, whose quotas, latencies and failure behaviour are neither reproducible nor ours to stress. We therefore built a discrete-event simulator of four systems. Its parameters are plausible for SaaS APIs but are assumptions, not measurements of any vendor, and the results should be read as properties of the mechanisms under those assumptions.

B. Systems, Fields and Workload

The four systems are a CRM, a marketing automation platform (MAP), a billing/ERP system and a warehouse/CDP, with the quotas and latencies in Table 1. Each call has a log-normal latency with the stated median and shape parameter 0.6, plus 10 ms per batched item. There are $|R| = 3{,}000$ canonical records and six fields, with the owners and priority classes in Table 2 (the field names are illustrative labels). Record popularity follows a Zipf law with exponent 1.1, so a few records are written very often, which produces realistic concurrent edits. Writes arrive as a Poisson process at 12 per second, rising to five times that rate in two 300-second bursts (at 900 to 1,200 s and 2,400 to 2,700 s), for 3,600 s, followed by a 1,800 s drain period with no new writes. In 75% of writes the origin is the field's owner, and in 25% it is another system, drawn in proportion to the origin weights of Table 1. The mean run contained 72,156 writes on 9,903 distinct keys, which implies about 216,000 deliveries.

TABLE 1. Systems in the simulation

System	Quota C_s (calls/s)	Median call latency (s)	Origin weight	Batch size, proposed
CRM	25	0.25	0.40	10
Marketing automation (MAP)	20	0.35	0.25	10
Billing / ERP	15	0.40	0.15	10
Warehouse / CDP	60	0.15	0.20	50

TABLE 2. Synchronized fields

Field	Illustrative meaning	Owner	Priority
f_0	Lifecycle stage	CRM	Critical
f_1	Account tier	CRM	High
f_2	Lead score	MAP	Normal
f_3	Engagement status	MAP	Normal
f_4	Payment status	Billing	Critical
f_5	Contract value band	Billing	High

C. Faults, Clocks and Conflicts

The MAP is unavailable from 1,500 to 1,800 s, and the billing system from 2,900 to 3,050 s; calls started in an outage fail after 2 s. Each system has a clock offset drawn from $\mathcal{N}(0, 1.5^2)$ seconds. The conflict window is $\delta = 1$ s. These choices put the clock-offset error of (5) at about 32% for edits one second apart, and put the first burst's arrival rate at the MAP and billing queues far above the per-item quota, which is a deliberately stressful setting that the burst sweep of Section VI.F relaxes.

D. Policies Compared

Batch ETL extracts changes every 900 s, deduplicates by key, and loads them in bulk calls of 25 items; failed calls retry after 60 s; merging is last-writer-wins on timestamps. Point-to-point webhooks send each update to each target immediately in a single-item call; a call that would have to wait more than 1 s for quota is rejected (429) and retried after $5(k + 1)$ s up to three times, after which the update is dropped; failed calls follow the same rule. Polling iPaaS picks up changes at 60 s poll boundaries, enqueues them FIFO as single-item calls, and retries failures after 30 s without limit. UROA is as described in Section IV, with batches of 10 (50 for the warehouse), coalescing, priority queues, capped exponential backoff, no drops, hybrid logical stamping, ownership-aware merge, and reconciliation every 300 s. All baselines use last-writer-wins on origin timestamps. Four ablations remove one element of UROA: no batching or coalescing (single-item FIFO, which removes the two together), no HLC or ownership merge (plain last-writer-wins), no reconciliation, and no priority queues.

E. Metrics and Statistics

We report the delivery latency percentiles of (1) over completed deliveries, the loss rate (deliveries never completed, as a share of all deliveries), the time-averaged divergence $\bar{\Delta}$ sampled every 10 s, the peak divergence, the API cost Γ of (3), the policy-violation rate ϵ, and the fraction of keys unconverged at the end of the drain period. Each configuration is run for five seeds (500 to 504), and we report means and 95% Student-t intervals. The burst sweep uses three different seeds (700 to 702).

6. RESULTS

A. Headline Comparison

Table 3 gives the main comparison. UROA's median delivery latency was 0.30 s, with a 95th percentile of 0.98 s. Batch ETL had a median of 452 s, and polling iPaaS 68 s with a 95th percentile of 601 s. Point-to-point webhooks were fast at the median (0.36 s) but had a 95th percentile of 7.7 s, and, more importantly, lost 18.3% of deliveries: after three rejected attempts, or an outage, the update was dropped. Because nothing repairs a dropped update, 36.0% of written keys were still unconverged at the end of the drain period, and every one of them is a policy violation.

TABLE 3. Main results (mean ± 95% CI over 5 seeds)

Architecture	Median latency (s)	p95 (s)	p99 (s)	Lost deliveries (%)	Mean divergence $\bar{\Delta}$ (%)	API calls per write	Policy violations ϵ (%)
Batch ETL	452.3 ± 2.5	863.3 ± 0.3	893.2 ± 0.4	0.0	26.0 ± 0.1	0.027	0.069 ± 0.041
Point-to-point webhooks	0.36 ± 0.01	7.7 ± 0.01	34.3 ± 0.01	18.3 ± 0.1	24.5 ± 0.3	5.20	36.05 ± 0.37
Polling iPaaS	68.3 ± 1.4	601.5 ± 14.9	946.0 ± 17.0	0.0	19.9 ± 0.3	3.11	0.069 ± 0.041
UROA	0.297 ± 0.001	0.98 ± 0.01	107.9 ± 0.3	0.0	0.77 ± 0.02	2.29	0.040 ± 0.026

The time-averaged divergence summarises the same picture in terms the business feels. For UROA, 0.77% of written fields were out of agreement across systems at an average moment; for the baselines the figure is 19.9% to 26.0%. The peak divergence, during the first burst, was 10.1% for UROA, 36.7% for polling iPaaS, 49.9% for webhooks and 56.3% for batch ETL. The per-write API cost of UROA is moderate (2.29 calls), below that of webhooks (5.20, inflated by 429 rejections and retries) and polling (3.11, one call per item and target), and far above that of batch ETL (0.027 calls per write, which is the efficiency that ETL buys with its staleness).

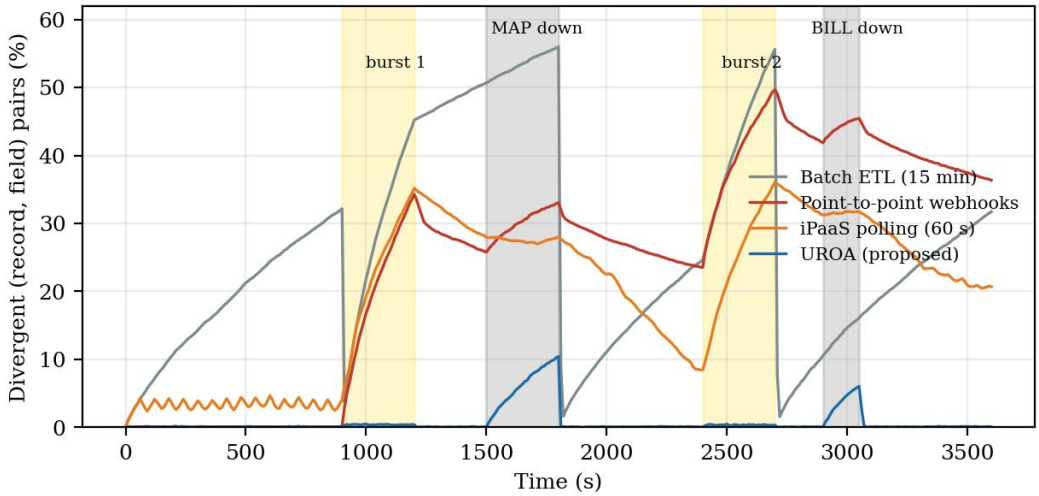


Figure 2: Share of divergent (record, field) pairs over time for one representative seed. Shaded bands mark the two traffic bursts and the two target outages. Batch ETL is stale by construction; webhooks lose updates and never recover; polling accumulates a backlog in the first burst that takes tens of minutes to drain; UROA stays near zero except for a short excursion during the first burst.

B. The Tail Latency of UROA

The p99 delivery latency of UROA, 107.9 s, is much worse than its p95, and it is worse than the 34.3 s of webhooks. This deserves explanation, because it is not a property of the scheduler. The 99th percentile is determined by the updates that arrived while the MAP or the billing system was down, which waited for the outage to end: by (10) an update that arrives at the start of the MAP outage cannot be delivered in less than 300 s, and the durable queue holds it until then. Webhooks have a smaller tail because they drop the same updates instead of delivering them late, so their latency distribution is computed over the survivors. The two numbers are therefore not comparable without the loss rate next to them. For updates to the critical-priority fields, the p99 was 129.9 s, longer than for all fields, because the outages affected critical fields as much as the others and priority queues cannot help a target that is not accepting calls. Figure 3 shows the percentile structure for the four architectures.

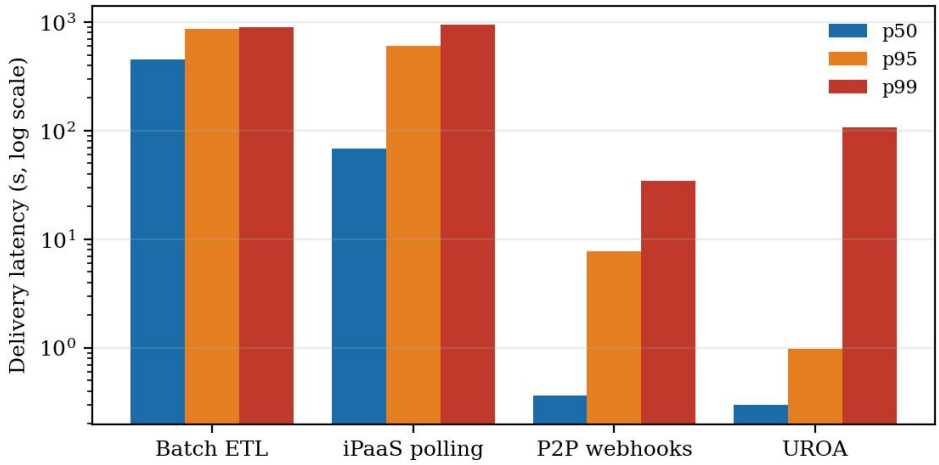


Figure 3: Delivery latency percentiles (log scale). UROA is fastest at the median and 95th percentile. Its 99th percentile is bounded below by the duration of target outages, and the webhook percentiles are computed only over the 81.7% of deliveries that completed.

C. Ablation Analysis

Table 4 removes one element at a time. Batching and coalescing are decisive. Without them, UROA's median latency rose from 0.30 s to 37.9 s, its p99 from 107.9 s to 1,815 s, and its mean divergence from 0.77% to 24.7%: single-item delivery to the MAP and billing queues saturates in the bursts, which is the regime predicted by (8). The ablation also removes coalescing, so we cannot attribute the effect between the two; the reconciler's repair count (54,168 against 1,711) shows that the backlog was so deep that many in-flight updates looked like divergence. Reconciliation removed no loss in this run, since UROA never drops updates, but it reduced the keys unconverged at the end of the drain period from 0.050% (± 0.039) to zero, which is a repair of a small tail rather than a structural change. Hybrid-logical-clock stamping and ownership merge reduced the policy-violation rate from 0.069% (± 0.041) to 0.040% (± 0.026). The two intervals overlap, so the data are suggestive but do not establish an improvement. Priority queues had no measurable effect: all latency and divergence measures were identical to within noise. In this workload no target is saturated under batching, so there is no contention for priority to arbitrate.

TABLE 4. Ablations of UROA (mean ± 95% CI over 5 seeds)

Variant	Median latency (s)	p99 (s)	Mean divergence (%)	API calls per write	Unconverged at end (%)	Policy violations (%)	Repairs
UROA (full)	0.297 ± 0.001	107.9 ± 0.3	0.77 ± 0.02	2.29	0.000	0.040 ± 0.026	1,711
No batching, no coalescing	37.9 ± 1.7	1,815 ± 52	24.7 ± 0.3	3.71	0.000	0.052 ± 0.028	54,168
No HLC, no ownership merge	0.297 ± 0.001	107.9 ± 0.4	0.77 ± 0.03	2.29	0.008 ± 0.010	0.069 ± 0.041	1,757
No reconciliation	0.297 ± 0.001	107.8 ± 0.2	0.80 ± 0.02	2.29	0.050 ± 0.039	0.071 ± 0.045	0
No priority queues	0.297 ± 0.001	107.9 ± 0.4	0.77 ± 0.02	2.29	0.000	0.040 ± 0.026	1,713

D. Where Batch ETL and Polling Spend Their Latency

The baselines' latency has simple structural causes. Batch ETL's mean latency, 452 s, equals half the schedule period plus a small service time, which is what a uniform arrival within a 900 s window predicts, and its p99 of 893 s is the schedule period. Polling iPaaS is stable at the base rate, but in the first burst the MAP and billing queues are offered more items per second than their quota, and by (10) with $B_s = 1$ the backlog takes tens of minutes to drain, which is why its p95 is 601 s while its median is only 68 s.

E. Policy Violations

All architectures except webhooks eventually delivered every update, so their policy-violation rates are small, 0.04% to 0.07%, and they arise only where the merge rule gave the wrong winner for edits that fell inside a conflict window or inside the clock-offset error. Webhooks' 36.05% is almost entirely lost updates, not wrong merges. The small violation rates of the other three architectures are of the same order, and their differences lie within the confidence intervals.

F. Sensitivity to Burst Severity

Figure 4 and Table 5 vary the burst multiplier from 1 (no burst) to 12. Webhooks degrade steeply: lost deliveries rise from 2.6% to 50.1%, and the share of keys that violate the policy rises from 7.6% to 62.0%. Polling iPaaS's p99 rises from 272 s to 2,548 s and, at a multiplier of 12, its policy-violation rate reaches 26.6% because the backlog has not drained by the end of the run. Batch ETL is insensitive, since its schedule dominates. UROA's median latency rises only from 0.28 s to 0.32 s, its policy-violation rate stays between 0.03% and 0.08%, and its API cost per write falls from 2.94 to 1.45, because larger bursts increase the opportunity for coalescing and fuller batches. UROA's p99 falls from 154 s to 10.5 s as the burst multiplier grows. This should not be read as an improvement from load: the p99 is dominated by updates delayed by the two outages, and as the bursts grow they supply a larger share of deliveries that fall outside those outages, which dilutes the outage tail.

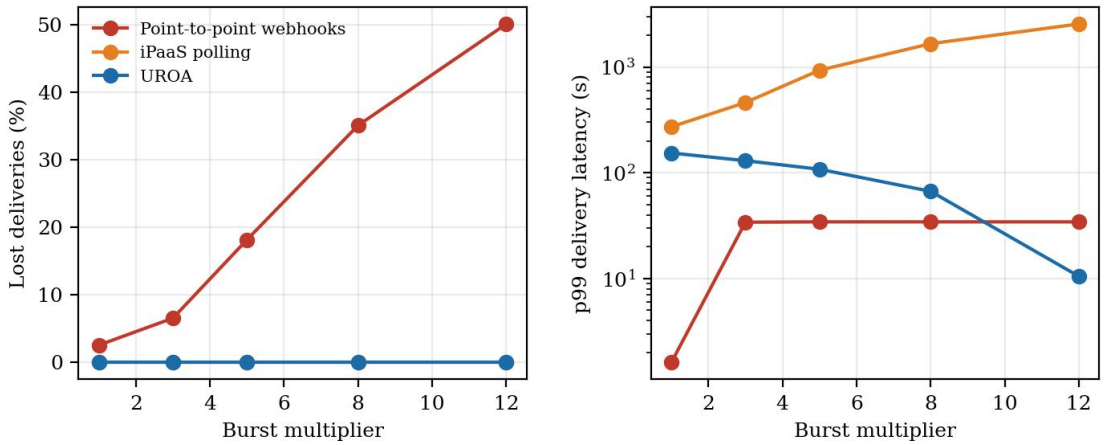


Figure 4: Burst-severity sweep (mean of three seeds). Left: share of deliveries lost. Right: p99 delivery latency on a log scale. UROA is insensitive to burst size, whereas webhooks lose and polling accumulates backlog.

TABLE 5. Burst sweep: median latency (s) / p99 (s) / policy violations (%) / API calls per write (mean of 3 seeds)

Burst multiplier	Batch ETL	Webhooks	Polling iPaaS	UROA
1	451 / 892 / 0.04 / 0.03	0.28 / 1.6 / 7.62 / 3.26	43 / 272 / 0.04 / 3.18	0.28 / 154 / 0.03 / 2.94
3	451 / 893 / 0.04 / 0.03	0.33 / 34 / 16.8 / 3.95	53 / 461 / 0.04 / 3.14	0.29 / 130 / 0.03 / 2.66
5	451 / 893 / 0.04 / 0.03	0.36 / 34 / 36.1 / 5.19	66 / 931 / 0.04 / 3.11	0.30 / 108 / 0.03 / 2.30
8	451 / 893 / 0.09 / 0.02	0.66 / 34 / 51.0 / 6.91	222 / 1,660 / 0.09 / 3.09	0.31 / 67 / 0.06 / 1.86
12	451 / 894 / 0.10 / 0.02	0.68 / 34 / 62.0 / 8.08	415 / 2,548 / 26.6 / 2.95	0.32 / 10.5 / 0.08 / 1.45

7. DISCUSSION

The evidence supports three claims. The first is that in a quota-constrained multi-cloud setting, the delivery layer, not the merge logic, determines whether synchronization is real-time. UROA's advantage over polling iPaaS in the headline table is chiefly explained by two mechanisms, durable queues with batched, coalesced delivery in place of single-item FIFO calls, and the ablation that removes them recovers most of the baseline's latency and divergence. Coalescing is safe because the merge rule is deterministic, and the reconciler's repairs are consistent because it uses the same rule.

The second claim is that durability matters more than speed at the median. Webhooks match UROA at the median and fail in the tail and in completeness, and nothing in their design repairs the loss. A synchronization architecture should be judged by its loss rate and its divergence, not by its median latency, which is the number that integration products most often advertise. The corollary is that the metric of record for a RevOps integration should be the time-averaged share of divergent fields, because it is the quantity that the business experiences as stale or inconsistent data.

The third claim is that no architecture escapes the outage bound. Equation (10) shows that an update that arrives in an outage cannot be delivered before the outage ends, and UROA's p99 of 107.9 s shows that bound in action. Reducing this tail requires measures outside the sync layer: degraded-mode reads that fall back to the source of record, alerting on the breaker state, and in some cases a standby path for critical fields. We did not test these.

Several limitations qualify the conclusions. First, the study is a simulation with assumed parameters. Real quota semantics are more complex, for example per-object limits, concurrency limits, daily caps and bulk endpoints whose cost depends on payload; real latencies have heavier tails; and real change feeds can be incomplete or out of order. Second, the ground-truth rule (4) is a modelling choice, and the policy-violation rate depends on it; an operator whose business rules differ should substitute their own. Third, the HLC, ownership-merge and reconciliation components showed small effects that our five seeds cannot resolve, and priority queues showed none. A workload with persistent contention, such as a saturated quota during a long import, would be needed to test priority, and one with more frequent concurrent edits would be needed to test the merge rule. Fourth, the merge rule (7) is not associative, so convergence is empirical; a formal guarantee requires either the pure last-writer-wins order or a stricter conflict-window definition. Fifth, identity resolution, schema mapping, deletes and privacy constraints such as consent propagation are not modelled, and

each is a significant part of a real deployment. Sixth, we use one set of controller parameters (batch sizes, reconciliation period, backoff cap), which we did not tune.

For practitioners, the evidence suggests an adoption order. The first investment is a durable queue between the systems with batched, coalesced, quota-aware delivery, because that step removes the loss and most of the divergence. The second is a periodic reconciliation job with a repair counter, which converts silent divergence into a measured quantity. The third is explicit field ownership and a documented merge rule, which makes concurrent edits predictable even if, in our data, it changes the violation rate only slightly.

8. CONCLUSION

This paper treated cross-cloud GTM data synchronization as a replicated-data and flow-control problem and proposed UROA, a five-layer architecture that captures changes with hybrid logical timestamps, buffers them in a durable log, merges concurrent edits by field ownership, delivers them through a quota-aware scheduler with coalescing and batching, and reconciles replicas by anti-entropy. A formal treatment gave a utilisation condition for quota-constrained delivery and a drain bound for outages. In a discrete-event simulation with bursts, outages and clock offsets, UROA delivered updates with a median latency of 0.30 s, lost none, and kept 0.77% of written fields divergent on average, against 19.9% to 26.0% for batch ETL, polling iPaaS and webhooks, which additionally lost 18.3% of updates. Batching and coalescing were the decisive components; ownership-aware merging and reconciliation gave small, statistically unresolved gains, and priority scheduling gave none. UROA's p99 latency of 107.9 s is set by outage durations and not by scheduling.

Future work should validate the architecture against real SaaS tenants and traces, extend the model to deletes, schema evolution, consent propagation and identity resolution, test priority scheduling and the merge rule under persistent contention, and address the outage tail.

References

1. **Pawan Kumar Singh,** Scaling Gate-All-Around (GAA) Transistor Architectures for Sub-2nm AI Accelerators Using Atomically Thin 2D Materials, Vol. 33 No. 08 (2024): JOCAAA, Journal Name: Journal of Computational Analysis and Applications, Journal's ISSN: 1521-1398 (Paper),1572-9206 (Online), https://eudoxuspress.com/index.php/pub/article/view/5713

2. **Pawan Kumar Singh,** Energy-Efficient Task Scheduling in Green Cloud Computing Using Neuromorphic Spiking Neural Networks, Vol. 33 No. 08 (2024): JOCAAA, Journal Name: Journal of Computational Analysis and Applications, Journal's ISSN: 1521-1398 (Paper),1572-9206 (Online), https://eudoxuspress.com/index.php/pub/article/view/5712

3. **A. Mahendra Vardhan and S. Sridhar,** "Determining False Positive Analysis of Software Vulnerabilities with Predefined Scan Rules using Random Forest Classifier and Decision Tree Technique," 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), Greater Noida, India, 2022, pp. 622-625, DOI: https://doi.org/10.1109/ICAC3N56670.2022.10074458

4. **Ronakkumar shah ,** Cognitive Process Intelligence: Integrating Generative AI and Automated Workflow Discovery, Vol. 33 No. 08 (2024): JOCAAA, Journal of Computational Analysis and Applications, Journal's ISSN: 1521-1398 (Paper),1572-9206 (Online), https://eudoxuspress.com/index.php/pub/article/view/5676

5. **Ronakkumar shah,** research Trends in Smart Workflow Automation: A Bibliometric Analysis of Emerging Themes, Home / Archives / Vol. 33 No. 05 (2024): JOCAAA, Journal of Computational Analysis and Applications, Journal's ISSN: 1521-1398 (Paper),1572-9206 (Online), https://eudoxuspress.com/index.php/pub/article/view/5675

6. **Dip Bharatbhai Pate,** Dynamic vs. Static visualizations: Understanding their use cases in big data analysis, International Journal of Science and Research Archive, 2022, 05(02), 392-395., DOI https://doi.org/10.30574/ijsra.2022.5.2.0077

7. **Dip Bharatbhai Patel,** Building scalable business intelligence systems in the cloud: A technical approach, International Journal of Science and Research Archive, 2021, 02(01), 212-215., Received on 03 February 2021; revised on 17 April 2021; accepted on 19 April 2021, DOI: https://doi.org/10.30574/ijsra.2021.2.1.0047

8. **Dip Bharatbhai Patel,** Understanding the Impact of Text Analytics on Social Media Sentiment Analysis, Vol. 5 No. 5 (2019) ,September-October, DOI: https://doi.org/10.32628/CSEIT1936413

9. **Dip Bharatbhai Patel,** Real-Time Data Streaming in BI Dashboards: Opportunities and Challenges, Archives / Vol. 5 No. 4 (2019) : July-August, DOI: https://doi.org/10.32628/CSEIT192142

10. **Dip Bharatbhai Patel,** the Role of Data Visualization in Modern Intelligence, Archives / Vol. 2 No. 2 / Research Article, Published, 2020-11-28 DOI: https://doi.org/10.32996/jcsts.2020.2.2.4

11. **A. Srivastava,** "Securing PHI in cloud-based healthcare systems: Challenges, frameworks, and future directions," J. Comput. Anal. Appl., vol. 33, no. 2, Aug. 2024. [Online]. Available: https://www.eudoxuspress.com/index.php/pub/article/view/4349/3190

12. **A. Srivastava,** "AI-assisted cloud migration of healthcare claims data from legacy systems: A metadata-driven framework," Int. J. Commun. Networks Inf. Secur., vol. 14, no. 3, Nov. 2022. [Online]. Available: https://ijcnis.org/index.php/ijcnis/article/view/8682

13. **A. Srivastava,** "Enhancing provider and claims data accuracy using artificial intelligence on cloud-native data platforms," J. Comput. Anal. Appl., vol. 31, no. 4, Nov. 2023. [Online]. Available: https://eudoxuspress.com/index.php/pub/article/view/4347/3189

14. **A. Srivastava,** "A unified framework for secure cloud data management: Integrating governance, metadata intelligence, and privacy controls," J. Inf. Syst. Eng. Manage., vol. 9, no. 2, Apr. 2024. [Online]. Available: https://www.jisem-journal.com/download/45_A_Unified_Framework.pdf

15. **A. Srivastava,** "Optimizing large-scale data migration for cloud-native architectures," J. Comput. Anal. Appl., vol. 31, no. 2, pp. 652–662, May 2023. [Online]. Available: https://eudoxuspress.com/index.php/pub/article/view/4603

16. **Abdul Aziz.** "Intelligent Health Analytics: Mastering AI in Healthcare Data". Acta Scientiae, vol. 25, no. 4, Aug. 2024, pp. 544-55, https://www.periodicos.ulbra.org/index.php/acta/article/view/687. https://doi.org/10.5281/zenodo.21699886

17. **Event-Aware Multi-Layer Storage Risk Forecasting for Oracle Database Estates Using HAPF,** Raghu Gollapudi, International Journal of Computational and Experimental Science and Engineering, 10(4), 2024 ,10.22399/ijcesen.5183

18. **Backup Integrity and Recovery Readiness Assessment for High-Availability Databases** Raghu Gollapudi, Computer Fraud & Security, Vol. 2023, Issue 8, pp. 35-48, 10.52710/cfs.1031

19. **Risk-Controlled Near-Zero-Downtime Oracle Database Migration Using GoldenGate** Raghu Gollapudi, International Journal of Computational and Experimental Science and Engineering, 8(3), 2022 ,10.22399/ijcesen.5382

20. **An Automated Risk Scoring Framework for SQL Execution Plan Analysis and Performance Manoj Kumar Kagitha,** Automated Security Compliance in Ci/Cd: A Natural Language Processing (NLP) Approach to Detecting Vulnerabilities in Infrastructure-As-Code (Terraform/Bicep),Publication year 2021, Volume 2021, Issue 12, https://computerfraudsecurity.com/index.php/journal/article/view/970

21. **Manoj Kumar Kagitha,** Hybrid Quantum-Classical Resource Scheduling: A Proposed Architecture for Next-Generation Hyperscale Data Centers, Publication year 2024, Volume 2024, Issue 12, https://computerfraudsecurity.com/index.php/journal/article/view/969

22. **Manoj Kumar Kagitha,** Self-Healing Cloud Infrastructure: A Hybrid Reinforcement Learning Framework for Predicting Microservice Failures in Azure Kubernetes Service (AKS),Publication year 2023, Volume 2023, Issue 12,https://computerfraudsecurity.com/index.php/journal/article/view/971

23. **Manoj Kumar Kagitha,** LOAD BALANCING TECHNIQUES IN DISTRIBUTED CLOUD ENVIRONMENTS, (IJESMR - 2017), D.O.I:10.53555/jcr.v6:i1.13380, https://jcreview.com/archives/volume-6/issue-1/13380

24. **Manoj Kumar Kagitha,** AI-DRIVEN OBSERVABILITY FOR HYPERSCALE COLOCATION DATA CENTERS: A CASE STUDY OF LOG ANALYTICS AND ANOMALY DETECTION PIPELINES, Archives / Vol. 27 No. 7 (2019): JOCAAA,https://www.eudoxuspress.com/index.php/pub/article/view/5053

25. **Manoj Kumar Kagitha,** GREENOPS IN THE CLOUD: AN AI-DRIVEN TELEMETRY ANALYSIS MODEL FOR REDUCING CARBON FOOTPRINT IN HIGH-AVAILABILITY CLUSTERS, Vol. 28 No. 6 (2020): JOCAAA, https://www.eudoxuspress.com/index.php/pub/article/view/5054

26. **Bandari, V. V., & Lekkala, H. B. (2024).** AI-based operator behavior monitoring and cost optimization using digital traceability in manufacturing systems. Power System Protection and Control, 52(1), 38–45. https://pspac.info/index.php/dlbh/article/view/342

27. **Venumadhav Vavilala, Shankar Balla (2024, May).** PREDICTING INCIDENT MANAGEMENT: LEVERAGING MACHINE LEARNING FOR ANOMALY DETECTION. Power System Protection and Control Scopus Q1 Journal. PSPC. https://pspac.info/index.php/dlbh/article/view/270

28. **Manikantha Varaprasad Inakollu, (2024, May),** FINOPS-Driven Cloud optimization models for enterprise applications, Vol. 52 No. 2 (2024): April-June 2024, 99-110, Power System Protection and Control, ISSN-1674-3415, URL: https://pspac.info/index.php/dlbh/article/view/283,, DOI: https://doi.org/10.46121/pspc.52.2.10

29. **Naresh Lokiny. (2022).** Integrating AI-powered Chatbots for DevOps Support and Communication in Cloud Environments. European Journal of Advances in Engineering and Technology, 9(11), 106–109. https://doi.org/10.5281/zenodo.13325989

30. **Naresh Lokiny, (2021),** "Disaster Recovery and Business Continuity Planning in DevOps Cloud with AI", International Journal of Science and Research (IJSR), 10(3), 2024-2027. https://dx.doi.org/10.21275/SR24724151733, https://www.ijsr.net/getabstract.php?paperid=SR24724151733

31. **Naresh Lokiny, & Ranganath Nandanampati. (2020).** DevSecOps: Integrating Security into DevOps with AI in Cloud. Journal of Scientific and Engineering Research, 7(10), 239–242. https://doi.org/10.5281/zenodo.13348695

32. **Naresh Lokiny, & Pradip Reddy. (2021).** Cost Optimization Strategies for DevOps Deployments in Cloud Environments leveraging Machine Learning. European Journal of Advances in Engineering and Technology, 8(3), 69–72. https://doi.org/10.5281/zenodo.13325845

33. **Jayanth Para,** 6G Internet Technology Cyber Threat Notification & Alert System, Vol. 52 No. 4 (2024): October-December 2024, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/170 , DOI: https://doi.org/10.46121/pspc.52.4.9

34. **Jayanth Para,** AI Based Cloud Computation Observational Method & Process, Vol. 51 No. 4 (2023): October-December 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/171 , DOI: https://doi.org/10.46121/pspc.51.4.3

35. **Kaleshwar Aryasomayajula,** PREVENTING BIAS IN AI-BASED DECISION-MAKING: ANALYZING TECHNIQUES TO REMOVE UNFAIR PREJUDICE IN ALGORITHMIC OUTCOMES, Vol. 50 No. 2 (2022): April-June 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/294

36. **Kaleshwar Aryasomayajula,** MODERN DEVELOPMENT PRACTICES: INVESTIGATIONS INTO SERVERLESS COMPUTING, LOW-CODE/NO-CODE PLATFORMS, AND DEVELOPER PRODUCTIVITY IN REMOTE ENVIRONMENTS, Vol. 50 No. 4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/364

37. **Kaleshwar Aryasomayajula,** Micro services: "A Comparative Analysis of Monolithic vs. Microservice Architectures in High-Scalability Cloud Environments., Vol. 51 No. 2 (2023): April-June 2023 , Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/374

38. **Vikas Suresh Bagora,** KERNEL-LEVEL PERFORMANCE OPTIMIZATION FOR CARRIER-GRADE BROADBAND AND HIGH-SPEED NETWORKING SYSTEMS, Vol. 47 No. 1 (2019), Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/359

39. **Vikas Suresh Bagora,** SCALABLE 128G FIBRE CHANNEL AND ETHERNET CONVERGENCE FOR NEXT-GENERATION DATA CENTER NETWORKS, Vol. 50 No. 4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/362

40. **Vikas Suresh Bagora,** SECURE EMBEDDED NETWORKING ARCHITECTURES USING TRUSTED EXECUTION ENVIRONMENTS IN BROADBAND GATEWAYS, Vol. 52 No. 2 (2024): April-June 2024, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/363

41. **Santhosh Chitraju Gopal Varma,** The Evolution of Cloud-Native Architectures: Exploring the Synergy between Kubernetes and Microservices, DOI: https://doi.org/10.63282/3050-9246.IJETCSIT-V1I4P104, Vol. 1 No. 4 (2020), https://ijetcsit.org/index.php/ijetcsit/article/view/108

42. **Santhosh Chitraju Gopal Varma,** Artificial Intelligence in Cloud Computing: Building Intelligent, Distributed, and Fault-Tolerant Systems, Vol. 3 No. 1 (2022), DOI: https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I1P105, https://ijaibdcms.org/index.php/ijaibdcms/article/view/78

43. **Sunil Karmakar,** AI driven Identity & Access Management with Oracle Cloud security, Vol. 30 No. 2 (2022): JOCAAA, https://eudoxuspress.com/index.php/pub/article/view/4118

44. **Rakshitha Kiran P.,** Samitha Khaiyum, Aparna R. Palandye, Apoorva S. D., Arjun R., Akshaya S. A., Annapoorna S., Leveraging LLaMA3 and LangChain for Rapid AI Application Development, Vol. 20 No. 10s (2024), https://doi.org/10.52783/jes.5539, https://journal.esrgroups.org/jes/article/view/5539

45. **Kancharla Vijayalakshmi,** Bethapudi Samuel Anand Andrews, Estimation of Cyproheptadine Hydrochloride and Tricholine Citrate Simultaneously in Syrup Dose by Applying Stability Indicating HPLC Methodology, December 31, 2020, Vol. 11, Issue 4, Pages 1813–1819, DOI: 10.26452/ijrps.v11iSPL4.4383, https://ijrps.com/v11si4/3683

46. **Vijayalakshmi Kancharla ,Kumar Abbaraju V D N, Andrews B S A, Nagaraju Dasari, Nagu Korupolu,** Melphan as Anticancer and New Insights of Drug as Parenteral Dosage form against SARS-CoV-2 by Using Development and Validation , November 18, 2020, Vol. 11, Issue 1, Pages 1437–1446 DOI: 10.26452/ijrps.v11iSPL1.3686, https://ijrps.com/v11si1/2911

47. **Ankit Pathak,** Leveraging AI for Better Data Quality and Insights, Volume 7 Issue 3 (2025), https://doi.org/10.32996/jcsts.2025.7.2.19, https://al-kindipublisher.com/index.php/jcsts/article/view/9345

48. **Uma Rajeswari Batchu, Bharati Reddi, Joshna Rani Surapaneni, Aruna Jangam, Sunil Misra, Anthony Addlagatta & Prakasham Reddy Shetty,** Inhibition of Xanthine Oxidase by 1-O-Methyl Chrysophanol, A Hydroxyanthraquinone Isolated from Amycolatopsis Thermoflava ICTA 103, Published: 13 December 2025,Volume 66, pages 1049–1068 (2025) https://link.springer.com/article/10.1007/s12088-025-01534-4

49. GAURAV KUMAR SINHA, Year 2025, https://visualcompublications.es/SAUC/article/view/6525

50. GAURAV KUMAR SINHA, Year 2025, https://visualcompublications.es/SAUC/article/view/6526

51. GAURAV KUMAR SINHA, Year 2025, https://visualcompublications.es/SAUC/article/view/6527

52. **D Benito Johnson 1, Gurinderdeep Singh 2, Deeksha Sharma 3, Venkatesan Natarajan 4, Knv Chenchu Lakshmi 5, Ram C Dhakar 6, Sadhana R Shahi 7, Suresh Velayutham 8, Ruchi Tiwari 9,** Exploring Computational Advancements in ADME: Essential Insights for Drug Disposition, PMID: 39496312 DOI: 10.62958/j.cjap.2024.033, https://pubmed.ncbi.nlm.nih.gov/39496312/