electronics

MDPI

Article

Scalable and Trusted Metadata-Coordinated Tiered Off-Chain Storage with Dynamic On-Chain Mapping for Recovery-Safe and Low-Latency IoT Data Management

Weiping Yu [1,†], Weihan Wang [1,†], Mingyuan Yan [2], Keyang He [1], Zhe Yu [1], Wenpeng Xing [1], Liyuan Liu [3] and Meng Han [1,*]

[1] College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China; ywp@zju.edu.cn (W.Y.); whwang@zju-if.com (W.W.); 22351077@zju.edu.cn (K.H.); zyu@zju-if.com (Z.Y.); wpxing@zju.edu.cn (W.X.)
[2] Department of Computer Science and Information Systems, University of North Georgia, Dahlonega, GA 30597, USA; mingyuan.yan@ung.edu
[3] Department of Decision and System Sciences, Saint Joseph's University, Philadelphia, PA 19131, USA; lliu@sju.edu
* Correspondence: mhan@zju.edu.cn
† These authors contributed equally to this work.

Abstract

Blockchain-assisted off-chain storage for IoT must simultaneously manage low-latency tiered data placement, trusted and dynamic on-chain mapping, migration consistency, and failure recovery—four concerns that existing designs address in isolation. Tiered storage systems optimize placement without modeling the scalable coordination cost of keeping object–location bindings trustworthy, while blockchain-metadata studies assume static storage topologies with no dynamic tier migration. This paper presents a scalable and trusted metadata-coordinated tiered off-chain storage framework, which bridges traditional trust systems (e.g., legacy authentication) with blockchain networks powered by Proof of Capacity (PoC) consensus. In this framework, adaptive heat-driven placement, dynamic on-chain mapping evolution with batched commitment, migration-aware redirect control, and rollback-safe recovery operate as a single coordinated workflow, with the five-stage write–verify–commit–redirect–retire pipeline acting as a lightweight coordination protocol that maintains ordered and atomic state transitions under message loss, out-of-order delivery, and single-node failures. The distinctive contribution lies in the framework's coupled control: every placement decision propagates through a verifiable metadata path that can be audited and, when necessary, rolled back. Simulation across multiple workload patterns shows that the proposed method reduces average access latency by 28% and raises the hot-tier hit ratio from 0.19 to 0.65 relative to a dynamic baseline without trusted mapping coordination under the simulated registry write cost. To achieve high-throughput mapping operations, batched on-chain commitment cuts metadata transactions by 50× at the cost of a tunable mapping freshness delay. The framework scales from 1 k to 50 k managed objects, effectively managing tens of millions of bytes of data (10+ MB scale) without disproportionate overhead growth; beyond this scale, hot-tier capacity rather than coordination becomes the dominant bottleneck, and smarter predictive placement becomes the natural next lever. All tested fault types achieve 100% rollback success with sub-millisecond local data plane interruption; audit-visible recovery depends on the assumed chain finality delay and, for heavily regulated IoT domains, such as finance and healthcare, should be treated as the operationally binding recovery time objective. These results, together with extended evaluations—including asymmetric write latency stress, coordination ablation, tail latency analysis, and benefit–complexity assessment—provide

Check for updates

Academic Editor: Ping-Feng Pai

Received: 3 April 2026
Revised: 17 May 2026
Accepted: 19 May 2026
Published: 25 June 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

quantitative evidence that scalable, dynamic mapping coordination can be integrated into tiered off-chain data management at an acceptable and measurable operational cost under the simulated configuration.

Keywords: trusted metadata coordination; tiered off-chain storage; IoT data management; rollback-safe recovery; blockchain-assisted storage; data protection

1. Introduction

Off-chain storage with blockchain-anchored metadata has become a standard architecture for managing large-scale IoT data: the blockchain records object–location bindings and access policies, while the actual data resides in external storage tiers [1–4]. Most designs treat the two layers—placement and metadata—as separable concerns. In practice, they are not. When an IoT data object is promoted from a cold tier to a hot tier to satisfy a latency-sensitive query, the on-chain mapping must be updated to reflect the new location; any in-flight query that arrives between the physical move and the mapping commit sees a stale pointer; and if the migration fails midway, both the physical copy and the on-chain record must be restored to a consistent prior state. These transitions—migration, redirect, commitment, and rollback—are where data management efficiency and data protection guarantees intersect, yet they are seldom modeled as one coupled mechanism.

A substantial body of work addresses tiered and hybrid storage placement. Hybrid storage surveys [5,6] and chunk placement optimizers [7] formulate the problem as a latency–cost trade-off driven by access frequency or data temperature. Recent edge–cloud collaborative schemes extend this to multi-tier topologies with network-aware cost models [8,9]. Automated tiering systems further introduce QoE-driven policies that migrate data between tiers based on real-time demand [10]. These systems universally assume that object–location bindings are instantaneously consistent; none model the coordination cost of maintaining trustworthy bindings during migration or specify what happens when a metadata update has not yet been committed.

The complementary research line uses blockchain or trusted registries to protect data integrity and enforce auditable access in off-chain storage. Blockchain-based frameworks for IoT data sharing record object hashes, access logs, and provenance chains on chain [11–13]. Scalable off-chain architectures extend this to healthcare and privacy-sensitive domains by combining IPFS with smart-contract access control [3,14]. These designs provide strong integrity guarantees for data at rest, but assume a largely static storage topology; they do not address how the on-chain registry should evolve under dynamic tier migration, how commitment should be batched, or how rollback should be orchestrated after a partial metadata update.

Neither research line addresses the full lifecycle of a tier migration event: trigger → physical move → metadata commitment → redirect of in-flight queries → rollback on failure. Without coupled control across these stages, three concrete problems remain unresolved. First, the metadata coordination overhead of tiered storage is unknown [15]—how many on-chain transactions does each migration generate, and can batching reduce them without unacceptable staleness? Second, scalability under increasing object counts has not been evaluated with metadata overhead included. Third, the recovery latency for different fault types—target write failure, verification failure, commit failure, redirect failure, partial metadata corruption, stale pointer repair—has not been quantified with timing granularity, leaving "recovery-safe" as an unverified claim in existing work.

This paper presents a scalable and trusted metadata-coordinated tiered off-chain storage framework that treats adaptive placement and dynamic on-chain mapping evolution as one integrated mechanism. The overall system architecture is shown in Figure 1. The framework bridges traditional trust systems (e.g., legacy authentication) with blockchain networks powered by Proof of Capacity (PoC) consensus, and operates through a coordinated workflow (Figure 2): an adaptive heat model identifies migration candidates; a utility-guided policy decides whether to promote or demote each object; the migration executor moves the data and, upon success, enqueues a mapping update entry; a batched commitment controller groups pending entries and commits them to the on-chain registry at a tunable frequency [16], achieving high-throughput mapping operations; a redirect handler steers in-flight queries to the correct tier during the commitment window; and a rollback-safe recovery module restores both the physical location and the on-chain mapping to their pre-migration state if any stage fails.

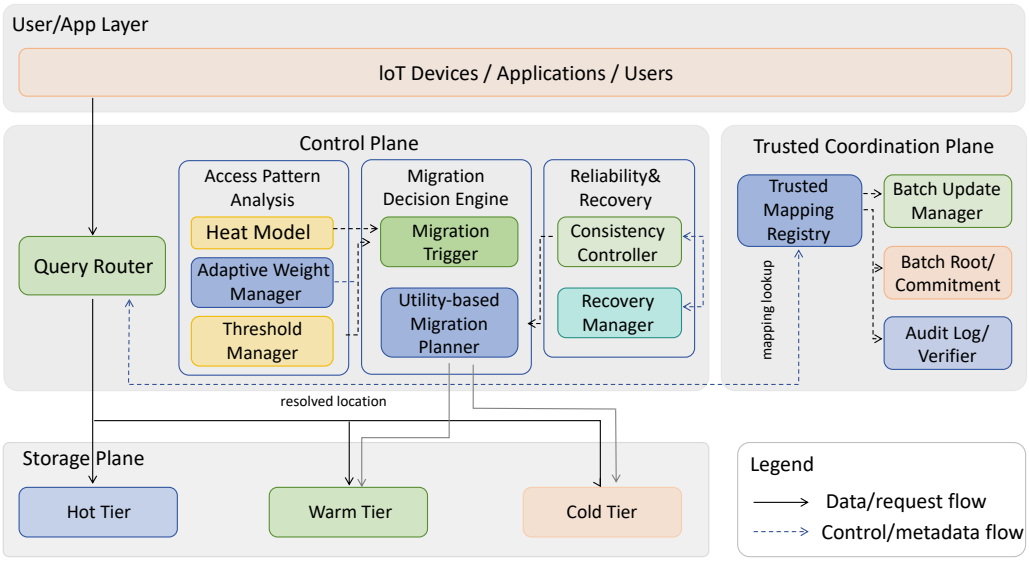
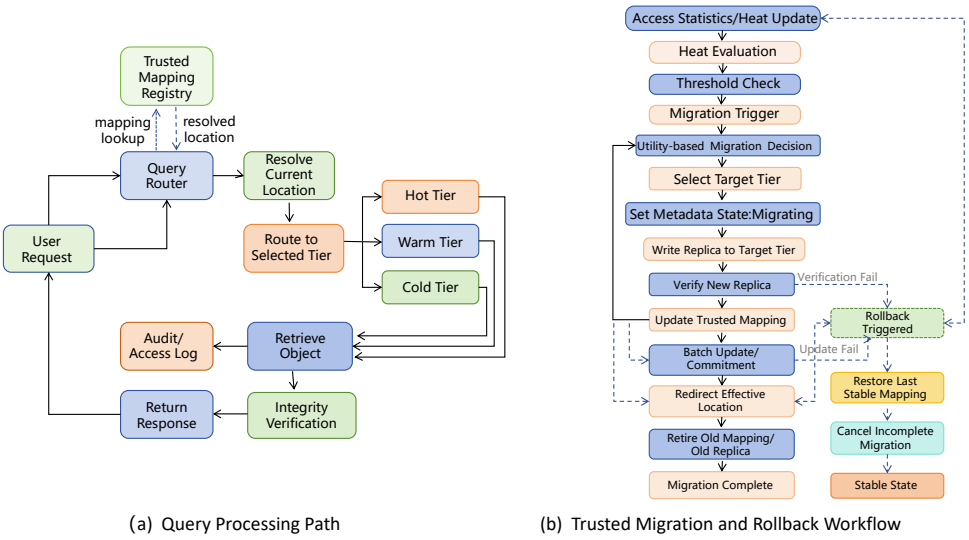


Figure 1. Architecture of the trusted metadata-coordinated tiered off-chain storage framework. Solid arrows denote data/request flow, and dashed arrows denote control/metadata flow.



(a) Query Processing Path (b) Trusted Migration and Rollback Workflow

Figure 2. Trusted mapping evolution workflow: (**a**) normal query resolution; (**b**) five-stage migration path with rollback.

The contributions of this paper are as follows:

1. We formulate trusted metadata-coordinated tiered off-chain data management as a coupled problem in which placement, mapping evolution, commitment, redirect, and rollback must be jointly controlled to satisfy both efficiency and data protection requirements.
2. We design a unified workflow that integrates adaptive heat-driven placement, batched on-chain mapping commitment, migration-aware redirect control, and rollback-safe recovery into a single coordinated mechanism, rather than composing them as independent modules.
3. We provide correctness and protection reasoning for the proposed workflow and evaluate it through discrete event simulation across eight targeted experiment groups (E1–E8) that jointly cover overhead, workload sensitivity, scalability, recovery, ablation, chain delay sensitivity, tail latency distribution, and benefit–complexity analysis; the per-experiment design is detailed in Section 4 and is therefore not re-enumerated here.
4. We report quantitative evidence that trusted metadata coordination can be integrated into tiered off-chain storage at an acceptable and measurable operational cost under the simulated configuration: the proposed method reduces the average access latency by 28% relative to a dynamic baseline without mapping coordination, batched commitment reduces on-chain metadata transactions by 50×, and all tested fault types achieve 100% rollback success with sub-millisecond local data plane interruption in the simulator under the current single-fault injection model, while audit-visible recovery depends on the assumed chain finality delay and, for heavily regulated IoT domains (e.g., finance and healthcare), should be treated as the operationally binding recovery time objective. We additionally characterize the benefit–complexity trade-off across object count scales and identify a diminishing return region beyond which the coordination advantage narrows under fixed hot-tier capacity; in this region, the bottleneck shifts from coordination cost to hot-tier capacity and to the quality of the placement strategy itself, so smarter predictive or collaborative placement becomes the natural next lever rather than further mapping coordination optimization.

The remainder of this paper is organized as follows: Section 2 positions the work against three research streams—tiered placement, blockchain-anchored metadata, and metadata scalability/recovery—and summarizes the gap matrix in Table 1. Section 3 formalizes the system model, metadata tuple, heat- and utility-driven placement, the five-stage write–verify–commit–redirect–retire workflow, and the threat and correctness analysis. Section 4 describes the simulator, baselines, workloads, and E1–E8 experiment configurations. Section 5 reports the E1–E8 results, and Section 6 interprets them, including the deployment gap to real consortium chains and the conditions under which the coordination benefit saturates. Section 7 concludes.

Table 1. Feature comparison with representative related work. DM = dynamic tier migration; ME = mapping evolution; BC = batched commit; IR = in-flight redirect; RB = rollback recovery; RQ = recovery latency quantified; DG = deployment gap analysis. ● = addressed; ○ = partial; — = not addressed.

Category	DM	ME	BC	IR	RB	RQ	DG
Tiered placement [5,8,10]	●	—	—	—	—	—	—
Decentralized storage [2,17,18]	—	—	—	—	—	—	—
Blockchain–IoT metadata [3,11,13]	—	—	—	—	—	—	—
Metadata scalability [16,19,20]	—	—	●	—	—	—	—
Chain benchmarks [21,22]	—	—	—	—	—	○	○
This work	●	●	●	●	●	●	●

2. Related Work

This section reviews three research lines—tiered data placement, blockchain-assisted trusted metadata, and consistency/recovery mechanisms—and identifies what each leaves unaddressed regarding metadata coordination during dynamic tier migration. Table 1 summarizes the coverage gap across five coordination dimensions—dynamic migration, mapping evolution, batched commitment, redirection, and rollback—and is referenced throughout this section to anchor each subsection's gap analysis. For readers familiar with off-chain IoT storage, we deliberately keep the description of standard building blocks—e.g., IPFS content addressing, Hyperledger Fabric endorsement—to a minimum and refer to authoritative sources [2,21,23]; the discussion below focuses on what each research stream does not provide along the five dimensions of Table 1.

2.1. Tiered and Off-Chain Data Management

Tiered and hybrid storage systems assign data objects to different performance tiers based on access frequency, recency, or cost objectives. Representative studies include hybrid storage surveys that classify migration policies by granularity and timing [5], chunk-level placement optimizers that refine data temperature tracking beyond object-level policies [7], and QoE-driven automated tiering that triggers migration in response to real-time demand shifts [10].

Cloud–edge collaborative tiered storage [8] and edge caching strategies [9] extend the approach to network-aware multi-tier topologies. In the decentralized domain, IPFS provides content-addressed off-chain storage [2,23], and a recent comprehensive survey confirms that tiered and content-addressed storage are increasingly deployed together, yet the coordination of metadata between layers remains under-studied [4]. Li et al. [17] propose a three-tier blockchain–IPFS framework for IoT privacy protection that demonstrates the feasibility of multi-tier decentralized storage; however, their metadata model is populated once at data creation and does not evolve during tier migration. Zheng et al. [18] similarly combine blockchain and IPFS for industrial IoT, but focus on data provenance rather than migration-triggered mapping updates.

The common gap is that none of these systems models the on-chain transaction cost incurred when a trusted mapping registry is updated after each migration, nor do they define a recovery protocol for the case in which a migration completes physically, but the corresponding metadata update fails or is only partially committed. In terms of the five coordination dimensions summarized in Table 1, tiered placement systems address dynamic migration (DM) but leave mapping evolution (ME), batched commit (BC), in-flight redirect (IR), and rollback recovery (RB) entirely unaddressed; consequently, they do not provide an end-to-end mechanism for migration-triggered trusted mapping evolution. In one sentence: tiered placement decides where an object should live but assumes that telling the rest of the system about it is free, instantaneous, and never fails—an assumption that this paper explicitly relaxes.

2.2. Blockchain-Assisted Metadata and Trusted Registry

A parallel research line leverages blockchain to protect the integrity and auditability of off-chain data references.

Blockchain-based trustable frameworks anchor object hashes and access events on chain, enabling post hoc provenance [11]; IPFS-backed stream integrity designs extend this to continuous data ingestion [13]; and healthcare-oriented systems combine off-chain bulk storage with smart-contract access control [3]. Blockchain-based data-sharing architectures further separate payload from authorization to keep the chain lightweight [12], and broader surveys confirm that on-chain metadata/off-chain data separation is now a mainstream

architectural choice for IoT systems [1,24]. Singh et al. [25] address cross-domain IoT data sharing with blockchain-enforced policies, while Pajooh et al. [26] record IoT data provenance on blockchain over a Hadoop ecosystem.

Hyperledger Fabric has also been demonstrated as a trust layer for edge IoT devices [27]. These works collectively reinforce the viability of blockchain-anchored metadata for IoT, but uniformly assume that the physical storage location is fixed once the initial on-chain record is written.

These designs excel at data-at-rest protection but share a write-once metadata model: the on-chain pointer is created at storage time and assumed permanent. None specifies how the binding should evolve under repeated migration, how updates should be batched to control transaction overhead, how concurrent queries should be redirected during the commit window, or how the pointer should be rolled back after a failed migration. Referring again to Table 1, these systems provide no coverage of ME, BC, IR, or RB; DM is absent because the storage topology is assumed static. Without addressing any of these dimensions, existing trusted-registry designs cannot support an end-to-end workflow from migration trigger to confirmed mapping update and conditional rollback. More specifically, blockchain-anchored metadata systems provide provenance for a fixed pointer but not lifecycle management for an evolving pointer; once placement is allowed to change, every guarantee they offer for the initial binding has to be re-established for every subsequent migration, and none of these works specifies how.

2.3. Consistency, Recovery, and Protected Metadata Evolution

In distributed storage systems, metadata management and its scalability have received extensive attention.

Key scalability mechanisms—partitioning, caching, and batching—are surveyed by Singh and Bawa [15]. BatchFS [16], IndexFS [19], and GIGA+ [20] demonstrate that batched metadata operations can scale file-system control planes, and that metadata plane scalability can be decoupled from data plane performance. These systems validate the principle that metadata operations benefit from batching and amortization, but they do not address the trust and rollback dimensions introduced by blockchain-anchored registries.

Several studies address protected updates, key evolution, and recovery in blockchain–IoT settings, but their scope differs from migration-triggered metadata coordination. Key update schemes evolve cryptographic key material rather than physical object locations [28]; credit-based consensus improves trust at the consensus layer without modeling the data management layer beneath [29]; lightweight on-chain storage reduces the blockchain's footprint for resource-constrained IoT devices [30]; proxy re-encryption enables delegation of decryption rights within the access control layer [31]; and edge–cloud access control protocols address user mobility rather than object placement [32]. In the healthcare domain, smart-contract-based access control [33,34] and blockchain-assisted integrity management [35,36] demonstrate fine-grained data protection, but the protected entity is access authorization rather than object–location binding. Hyperledger Fabric access control layers for IoT [37] confirm the feasibility of permissioned-chain metadata management at the edge, again without migration-triggered metadata evolution.

A common pattern across these works is that "recovery" refers to restoring access authorization, cryptographic state, or consensus integrity—not to restoring a consistent object–location binding after failed tier migration. The specific failure modes that arise on the data management path—target write failure, verification failure, commit failure, redirect failure, partial metadata corruption, stale pointer scenarios—are not modeled, and the associated recovery latencies are not measured. Performance analyses of Hyperledger Fabric [21,22] quantify per-transaction overhead and throughput ceilings, providing

a baseline for what on-chain metadata updates cost; however, they evaluate the chain platform in isolation, not the coupled workflow that connects a tier migration decision to a batched on-chain commit and a conditional rollback path. Bridging this gap requires both a coupled mechanism design and a deployment gap analysis that relates simulated coordination overhead to real chain finality latencies.

In terms of Table 1, metadata scalability systems address BC in a file system context but not ME, IR, or RB; blockchain–IoT protection studies address none of the five coordination dimensions because their focus is on access control and provenance, not tier migration metadata. Chain benchmark studies provide partial coverage of recovery latency quantification (RQ) and deployment gap analysis (DG) at the platform level, but not at the coupled-workflow level. The common shortcoming across the three research streams is therefore the same—each treats one slice of the migration-triggered metadata lifecycle (placement, integrity anchoring, or batching/scalability) in isolation, and none provides an end-to-end protocol that ties physical move, verified commit, in-flight redirect, and rollback into a single coordinated workflow—which is the gap addressed by this paper.

2.4. Positioning of This Work

Table 1 condenses the analysis above into five coordination dimensions (DM, ME, BC, IR, RB) and two evaluation dimensions (RQ, DG). Tiered placement systems model dynamic migration, but not the trusted mapping evolution it triggers; blockchain metadata systems provide auditable pointers, but assume static placement; metadata scalability and chain benchmark studies address individual building blocks but not the coupled workflow. No prior design addresses all five coordination dimensions simultaneously.

This paper fills the gap by modeling placement, mapping evolution, batched commitment, redirect, and rollback as one coordinated workflow and evaluating it across dimensions that are absent from the reviewed literature: metadata overhead with mapping freshness trade-offs, asymmetric write latency stress (2× and 5×), fine-grained ablation of the redirect, rollback, batching components, chain delay sensitivity with deployment gap estimation, tail latency distribution under variable offered load, and a benefit–complexity break-even analysis at scale.

3. System Design and Analysis

This section formalizes the framework in six sections. Section 3.1 defines the system model and four coupled design objectives. Section 3.2 introduces the metadata tuple and four-layer architecture. Sections 3.3 and 3.4 specify heat-aware placement and utility-based migration with explicit metadata cost. Section 3.5 details the five-stage write–verify–commit–redirect–retire workflow. Section 3.6 states the threat model, protection goals, correctness properties, and complexity.

3.1. System Model and Design Objectives

We consider a blockchain-assisted IoT data management system in which bulk data objects are stored off-chain across a tiered storage hierarchy, while the authoritative record of each object's current location, integrity digest, and lifecycle state is maintained in an append-only on-chain registry. Crucially, to satisfy the rigorous requirements of IoT environments, this registry is powered by a Proof of Capacity (PoC) consensus mechanism, which offers a scalable and energy-efficient foundation for storage-heavy operations. For transaction ordering and finality, the framework relies on the consortium chain's underlying Paxos- or Raft-class consensus protocol, with PoC serving as the chain-level resource-efficient consensus layer. Furthermore, the framework bridges decentralized metadata with traditional trust systems (e.g., legacy Public Key Infrastructure (PKI) and existing

IoT device authentication protocols). This on-chain/off-chain separation, combined with legacy trust integration, follows established reference-binding models [2,38,39] and their trusted IoT extensions [3,11].

Let $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$ denote the managed object set and $\mathcal{T} = \{\text{hot}, \text{warm}, \text{cold}\}$ the storage-tier set. The hot tier provides the lowest per-access latency at the highest unit cost; the cold tier provides the lowest cost at the highest latency; and the warm tier interpolates between them. Objects migrate across tiers as their access demand evolves, and each migration is simultaneously a data event (physical copy) and a metadata event (binding update).

The framework is defined by four objectives that must be satisfied jointly—not as independent optimizations layered after the fact:

1. Efficient tiered access: route each query to the tier that best matches the object's current demand intensity, using adaptive heat-driven placement (Section 3.3).
2. Metadata integrity and auditability: every object–location binding committed to the on-chain registry carries a verifiable integrity digest. The append-only registry retains the full evolution history, enabling post hoc audit of any mapping transition.
3. Migration consistency: during the window between a physical move and its on-chain commitment, in-flight queries must resolve through a well-defined redirect protocol—never through an unverified or partially committed pointer.
4. Rollback-safe recovery: if any stage of a migration fails (write, verification, commitment, or redirect), the system restores both the physical replica and the on-chain mapping to their last verified state, with quantifiable service interruption.

The coupling among these objectives is the source of the research problem. To make this coupling concrete, consider the state of a single object d_i undergoing promotion from cold to hot. Let P_i denote the physical tier that holds the data and let tier_i be the tier field of M_i recorded in the on-chain mapping. In an uncoupled design where placement and metadata evolve independently, three failure scenarios arise that cannot be prevented by optimizing either layer alone:

- **Stale pointer divergence:** placement completes ($P_i = \text{hot}$), but the mapping update is delayed or lost ($\text{tier}_i = \text{cold}$). Subsequent queries resolve through the stale pointer and are served from cold at 100 ms instead of hot at 1 ms. No amount of placement optimization can compensate because the query resolver trusts the registry.
- **Phantom commit inconsistency:** the mapping is updated ($\text{tier}_i = \text{hot}$), but the physical write to hot fails or is incomplete ($P_i = \text{cold}$ or corrupt). Queries are now routed to a non-existent or invalid replica. This failure is invisible to a system that commits metadata independently of physical verification.
- **Irrecoverable partial migration:** after a physical move succeeds and the old mapping is overwritten (no predecessor retention), the commit or redirect fails. Neither the old nor the new mapping points to a valid, verified state. Without a defined rollback path, manual intervention is required.

These scenarios arise whenever the migration rate exceeds the metadata update throughput or any single workflow stage fails. The proposed framework eliminates all three by construction: stale pointer divergence is prevented by the confirmed mapping cycle (Stages 1–3), phantom commit inconsistency by pre-commitment digest verification (Stage 2), and irrecoverable partial migration by the predecessor retention invariant (rollback semantics).

To make the behavioral difference concrete, Table 2 traces a representative scenario—a query arrives for object d_i, while d_i is being promoted from cold to hot—under uncoupled and coupled designs.

Table 2. Timing comparison: query arrives during cold→hot migration of object d_i.

Event	Uncoupled	Coupled
t_1: migrate starts	Write to hot begins	Write to hot begins
t_2: query arrives	Stale mapping → Cold	Stable pointer → cold
t_3: write succeeds	Mapping queued	Digest verified (Stage 2)
t_4: mapping update	Hot pointer (no verify)	Batch commit (Stage 3)
t_5: next query	Hot (unverified)	Redirect active (Stage 4)
t'_5: write fails	Missing replica	Rollback → cold
If success	Correct, unverified	Correct, verified
If failure	Error or data loss	Transparent fallback

The key distinction is at t_2 and t'_5: in the uncoupled design, the query at t_2 may be served from a stale pointer, and a failure at t'_5 leaves the mapping pointing to a non-existent replica; in the coupled design, the stable pointer remains effective until the full commit–redirect sequence completes, and failure at any point triggers a well-defined rollback. The remainder of this section formalizes these mechanisms.

3.2. Metadata Model and Trusted Coordination Architecture

Each object d_i is associated with a metadata tuple that serves as the primary data protection artifact of the framework:

$$M_i = (\mathrm{id}_i,\ \mathrm{tier}_i,\ \mathrm{loc}_i,\ \mathrm{ver}_i,\ \mathrm{hash}_i,\ \mathrm{state}_i,\ \mathrm{ts}_i), \tag{1}$$

where id_i is the logical object identifier, $\mathrm{tier}_i \in \mathcal{T}$ is the current storage tier, loc_i is the physical storage address, ver_i is a monotonically increasing version number incremented on each committed migration, hash_i is the collision-resistant integrity digest of the stored object, $\mathrm{state}_i \in \mathcal{S}$ is the lifecycle state, and ts_i is the timestamp of the latest on-chain commitment.

The lifecycle state set $\mathcal{S} = \{\text{stable, migrating, redirected, deprecated}\}$ captures every phase of a migration event. An object in state stable is accessible through its committed mapping; migrating marks a tuple whose physical copy exists in the target tier but whose mapping has not yet been committed; redirected indicates that the new mapping is committed and active; and deprecated marks a predecessor tuple retained for audit and eventual garbage collection. At any point, the query resolver serves requests through the latest committed non-deprecated tuple. Because the predecessor stable tuple is never discarded before the successor is fully committed, the system always has a recovery anchor for rollback.

The framework is organized into four layers (Figure 1):

- The user/app layer issues read and write requests against logical object identifiers.
- The control plane maintains the adaptive heat model (Section 3.3), evaluates migration utility (Section 3.4), and generates migration intents.
- The trusted coordination plane is the distinguishing layer. It receives migration intents, orchestrates the five-stage write–verify–commit–redirect–retire workflow (Section 3.5), manages the batched commitment controller, resolves redirect during the commitment window, and executes rollback on failure. All mapping mutations flow through this plane.
- The storage plane hosts the hot, warm, and cold tiers and executes physical data copies.

The architectural separation between the data/request path and the control/metadata path is deliberate: user queries follow the effective mapping directly to the storage plane, incurring only a single registry lookup overhead per access, while migration-triggered metadata mutations traverse the trusted coordination plane without blocking concurrent

reads. In Figure 1, solid arrows trace the data/request path (user query → effective mapping → storage tier), while dashed arrows trace the control/metadata path (heat model → migration intent → commitment → redirect). The four horizontal bands correspond to the user/app, control, trusted coordination, and storage planes described above.

3.3. Heat-Aware Tiered Placement

The control plane uses an adaptive heat model to determine the target tier for each managed object. This placement decision is the entry point of the coupled workflow: once the heat model identifies a tier mismatch, it triggers the migration evaluation (Section 3.4), which in turn triggers the metadata evolution pipeline (Section 3.5). Prior tiered storage studies confirm that frequency, recency, and short-horizon locality are effective placement signals [7,8,10]; the model below adopts these signals and makes them tunable.

Let $x_i(\tau)$ denote the number of accesses to object d_i in time slot τ, and let W_f denote the frequency-averaging window. The recent frequency term is

$$F_i(t) = \frac{1}{W_f} \sum_{\tau=t-W_f+1}^{t} x_i(\tau), \tag{2}$$

normalized across the active object set so that $F_i(t) \in [0, 1]$. The recency term is

$$R_i(t) = e^{-\lambda \, \Delta t_i}, \tag{3}$$

where Δt_i is the elapsed time since the last access to d_i and $\lambda > 0$ is the decay rate. Under workloads with repeated bursts or correlated queries, a query pattern score $Q_i(t)$ captures short-horizon reuse [7,8].

The composite heat score is

$$H_i(t) = \alpha(t)\, F_i(t) + \beta(t)\, R_i(t) + \gamma(t)\, Q_i(t), \tag{4}$$

subject to $\alpha(t) + \beta(t) + \gamma(t) = 1$, $\alpha(t), \beta(t), \gamma(t) \geq 0$. The weights are adapted from lightweight workload statistics rather than a heavy learning module: sustained hotspots raise $\alpha(t)$; rapid turnover raises $\beta(t)$; and short-horizon reuse raises $\gamma(t)$. The sensitivity of placement quality to these weights is evaluated experimentally in Section 5.

Tier assignment uses dynamic thresholds rather than fixed cutoffs [5,8,10]. Let $T_h(t)$ and $T_c(t)$ denote the hot-tier and cold-tier thresholds, with $T_h(t) > T_c(t)$. The desired tier for object d_i is

$$\tau_i^\star(t) = \begin{cases} \text{hot}, & H_i(t) \geq T_h(t), \\ \text{warm}, & T_c(t) < H_i(t) < T_h(t), \\ \text{cold}, & H_i(t) \leq T_c(t). \end{cases} \tag{5}$$

The thresholds are adjusted by tier occupancy feedback: when the hot tier approaches saturation, $T_h(t)$ rises to restrict admission; when utilization drops, $T_h(t)$ falls. A symmetric rule governs $T_c(t)$.

3.4. Utility-Based Migration and Batch Coordination

A tier mismatch ($\tau_i^\star(t) \neq \text{tier}_i$) indicates that object d_i would benefit from relocation, but migration is not free: it incurs data transfer cost, replica management cost, and—crucially, in this framework—dynamic on-chain mapping and metadata coordination cost. The controller therefore evaluates a migration utility $U_i = G_i - C_i$ before committing to any move.

The gain term estimates the access-weighted benefit of the tier change:

$$G_i = \hat{a}_i(\Delta_p) \cdot \big(L(\text{tier}_i) - L(\tau_i^\star)\big) + \mu\, H_i(t), \tag{6}$$

where $\hat{a}_i(\Delta_p)$ is the predicted access count in the next control period Δ_p, $L(\cdot)$ is the expected per-access latency of a tier, and $\mu \geq 0$ weights the urgency implied by the current heat score. The cost term is

$$C_i = c_i^{\text{tr}} + c_i^{\text{rep}} + c_i^{\text{map}} + c_i^{\text{mgmt}}, \tag{7}$$

where c_i^{tr} is data transfer cost, c_i^{rep} is replica creation or cleanup cost, c_i^{map} is the metadata update cost for the on-chain registry, and c_i^{mgmt} is scheduling and bookkeeping overhead. The explicit inclusion of c_i^{map} distinguishes this formulation from conventional tiered storage optimizers that treat metadata as a free side effect [5,10]: here, the on-chain coordination cost is a first-class term in the migration decision, directly coupling placement economics with metadata management overhead [15].

A migration intent is admitted only if $U_i \geq \theta_u$ (a configurable utility threshold) and the target tier satisfies the capacity constraint:

$$\sum_{d_i:\, \text{tier}_i = k} \text{size}_i \leq \text{Cap}_k, \qquad k \in \mathcal{T}. \tag{8}$$

Admitted candidates are ranked by decreasing utility. When capacity is insufficient, lower-utility residents of the target tier are demoted first. Candidates that still cannot be accommodated are deferred to the next control round.

Batch coordination.

Accepted migration intents are not committed individually. Instead, all intents generated within a configurable batch window W_b are grouped into a single commitment batch, amortizing one on-chain transaction across up to W_b migration events—an approach inspired by batched metadata operations in scalable file systems [16,19] but applied here to blockchain-anchored registries, where transaction cost is qualitatively higher [21,22]. This mechanism directly controls the trade-off between commit frequency and mapping update delay: a larger W_b produces fewer on-chain transactions but increases the interval during which a migrated object's on-chain pointer is stale (the pre-migration stable pointer remains effective during this window, so queries are not mis-routed, but the committed state lags the physical state). The experimental evaluation in Section 5 (metadata coordination overhead) sweeps W_b from 1 to 200 and measures both the commit frequency reduction and the resulting update delay, so that system operators can select W_b based on their acceptable freshness tolerance.

3.5. Recovery-Safe Mapping Evolution and Rollback Workflow

Migration intents generated by the utility-based scheduler enter the trusted coordination plane, where the actual state transition is governed by a five-stage ordered workflow. The design draws on principles from blockchain-assisted metadata coordination and protected off-chain records [12], and from protected key evolution and recovery paths [28,31,32], extending them to the specific requirements of dynamic tier migration. Figure 2 illustrates both the normal migration path and the rollback path.

In Panel (a) of Figure 2, the query resolver follows the effective pointer directly to the storage tier without entering the coordination plane. In Panel (b), the five stages proceed left to right along the normal path; the dashed rollback arrow indicates that failure at any stage restores both the physical replica and the on-chain pointer to the last verified stable tuple.

Stage 1: Verified replica write. The migration executor creates a physical copy of object d_i in the target tier. The new replica is not yet visible to the query resolver; the effective mapping continues to point to the source tier. If the write fails (target write failure), the migration is aborted immediately and no metadata mutation occurs.
Stage 2: Tentative metadata creation. The executor recomputes the integrity digest hash_i over the target replica and compares it against the source. If verification succeeds, a tentative metadata tuple M'_i is created with an incremented version number $\text{ver}_i + 1$ and state migrating. If the digest comparison fails (verification failure), the tentative replica is discarded, and the system falls back to the source mapping.
Stage 3: Batched mapping commitment. All verified tentative tuples accumulated within the current batch window W_b are aggregated into a single batch root—a deterministic digest of the ordered set of metadata records—and committed to the on-chain registry. This is where commit frequency reduction occurs: instead of one on-chain transaction per migration, the system issues one transaction per batch window. The trade-off is that mappings remain in the migrating state (not yet redirected) until the batch commits. If the commitment transaction fails (commit failure), all tuples in the batch revert to their pre-migration stable state.
Stage 4: Redirect activation. After successful commitment, the controller atomically switches each object's effective pointer from the source tuple to the committed target tuple, transitioning the state to redirected. In-flight queries that arrived during the commitment window were already served from the still-active pre-migration pointer, so no query observes an uncommitted location. If the redirect itself fails (redirect failure)—for example, because the target replica has become unavailable between commitment and activation—the controller restores the source pointer and marks the committed target tuple as invalid.
Stage 5: Predecessor retirement. After redirect confirmation, the source tuple transitions to deprecated. The deprecated tuple and its physical replica are retained according to a configurable retention policy for audit purposes, then eventually garbage-collected.

Per-stage guarantees: coupled vs. uncoupled design.

Table 3 summarizes, for each workflow stage, (i) the guarantee provided by the coupled design, (ii) what is lost if that stage is executed without coordination, and (iii) the recovery complexity when a failure occurs at that stage.

Table 3. Per-stage guarantees under coupled vs. uncoupled design.

Stage	Coupled	Without Coupling	Cost
1. Write	No mutation until success	Phantom commit	$O(1)$
2. Verify	Digest before commit	Corrupt accepted	$O(1)$
3. Commit	Atomic; all-or-nothing	Partial commit	$O(b)$
4. Redirect	Switch after verified commit	Invalid routing	$O(1)$
5. Retire	Predecessor retained	No rollback anchor	$O(1)$

In an uncoupled design, the absence of any single column-3 guarantee leads to one of the three failure scenarios identified in Section 3.1. The coupled workflow prevents all of them by ensuring that each stage's precondition is the previous stage's verified postcondition.

Rollback semantics.

Failure handling is integral to the state machine rather than an exceptional side path. The workflow identifies six distinct failure modes: target write failure (Stage 1), verification failure (Stage 2), commit failure (Stage 3), redirect failure (Stage 4), partial metadata corruption (detected by digest mismatch during any stage), and stale pointer

scenarios (detected when a query encounters a mapping whose physical target no longer exists). In each case, the controller executes the same recovery contract: restore the last stable tuple as the effective mapping, discard or quarantine the tentative/corrupted state, and log the event in the audit trail. Because the predecessor stable tuple is never discarded before the successor is fully committed, verified, and redirected, a known-good recovery anchor is always available. The experimental evaluation in Section 5 (failure recovery latency) quantifies detection time, rollback latency, and total service interruption for each of these six fault types.

Relationship to classical distributed coordination protocols.

The five-stage workflow does not reinvent distributed consensus; instead, it composes well-known building blocks along a single, ordered path. We make this relationship explicit to clarify how global ordering and atomicity are maintained under network partitions, node crashes, message loss, and out-of-order delivery.

- **Atomic commit across the data plane and the registry.** Stages 1–3 (write, verify, commit) form a coordinator-driven two-phase commit structure [40]: Stage 1 is the "prepare" phase that materializes the new replica without exposing it; Stage 2 is the digest-verified "vote"; Stage 3 is the durable on-chain commit that decides the outcome for the entire batch. Because the decision is anchored on chain (rather than on a transient coordinator state), a coordinator crash between Stages 2 and 3 is equivalent to a Stage 3 failure and is handled by the rollback contract above.
- **Global ordering from chain consensus.** The on-chain registry is assumed to expose a totally ordered, append-only log; this ordering is provided by the underlying Paxos- or Raft-class consensus protocol of the consortium chain [22,41,42]. Once a batch root is committed, every endorsing peer observes the same prefix of mapping versions, which gives the framework a single global serial order for all migrations, regardless of the order in which migration intents were generated. The framework's contribution is layered on top of this ordering: it decides what enters each batch and how rollback consumes the log, but it does not weaken the totally ordered semantics that the chain already provides.
- **Network partitions and out-of-order delivery.** Migration intents and tentative tuples are reordered freely until Stage 3. Stage 3 admits at most one batch per commit slot, and the on-chain log linearizes concurrent batches, so any pre-commit reordering is resolved by the consensus layer. During a partition that isolates the coordinator from the chain, no new batch can commit; in-flight queries continue to be served from the predecessor stable tuple (audit window isolation), and the partitioned coordinator's tentative tuples simply remain in the migrating state until the partition heals or a timeout triggers rollback. No user-visible inconsistency can arise because the effective pointer never advances ahead of the chain.
- **Node crashes and message loss.** A storage node crash before Stage 1 completes is detected as a target write failure; a crash between Stages 2 and 3 is treated as a commit failure (no chain record exists, so the predecessor tuple remains authoritative); and a crash between Stages 3 and 4 is handled by replaying the on-chain commit log—any peer recovering after the crash reconstructs the redirect state deterministically from the latest committed mapping. Message loss between stages is equivalent to one of the above timeouts and triggers the same rollback path. Idempotency is preserved because each (object-id, version) tuple is uniquely identified and re-applying a committed mapping is a no-op.

In summary, the five-stage workflow is best read as a thin, purpose-built coordination layer on top of an existing ordered-log consensus substrate, not as a replacement for it:

the chain provides Paxos/Raft-style global ordering and durability, while the workflow contributes the migration-specific atomicity contract (Stages 1–3) and the rollback contract that ties physical and on-chain state together. Specifying or proving a fully Byzantine fault-tolerant variant of this layer (e.g., under adversarial endorsers) is left to future work and is explicitly excluded from the threat model below.

3.6. Threat Model, Protection Goals, and Complexity

3.6.1. Threat Model and Protection Scope

The framework targets threats to the integrity and consistency of object–location metadata that arise specifically during dynamic tier management. Three threat categories are within scope:

1. Unauthorized mapping modification: an adversary or faulty component silently alters a committed tuple in the registry.
2. Stale or inconsistent pointers: a partially completed migration leaves the on-chain pointer in a state that does not correspond to any physically accessible replica.
3. Loss of the recovery anchor: a migration failure occurs after the predecessor stable tuple has been prematurely discarded, leaving no rollback target.

The framework does not claim to address network-level denial-of-service, consensus-layer Byzantine faults, or end-to-end data confidentiality. These require complementary mechanisms (e.g., Byzantine fault-tolerant consensus, encryption at rest) that operate outside the tiered data management path. The protection boundary is deliberately scoped to metadata integrity, auditable evolution, migration consistency, and rollback-safe recovery—the specific guarantees that the experimental evaluation in Section 5 is designed to validate.

3.6.2. Protection Goals

Given the threat model above, the framework provides the following protection goals:

- **PG-1: Metadata integrity.** Every committed mapping tuple carries a collision-resistant integrity digest; any post-commitment modification is detectable by recomputing the digest.
- **PG-2: Auditable evolution.** The append-only registry retains the full version chain of every object's mapping history, enabling post hoc verification of any migration event.
- **PG-3: Migration consistency.** At no point during the five-stage workflow does the query resolver serve a request through an unverified, uncommitted, or partially committed pointer.
- **PG-4: Rollback-safe recovery.** A known-good stable tuple is always available as a rollback target; the recovery procedure restores both the physical state and the on-chain pointer atomically.

These goals are not independent: PG-1 enables PG-3 (because only digest-verified tuples enter the commitment pipeline), and PG-2 enables PG-4 (because the append-only log provides the recovery anchor). This interdependence reflects the coupled nature of the framework.

We emphasize that PG-1–PG-4 are stated and validated at the system design level: they concern verifiable mapping invariants, ordered state transitions, and append-only audit retention. The framework does not claim cryptographic security properties such as IND-CPA or IND-CCA; such properties pertain to the underlying hash function and consensus mechanism, which are treated as black-box building blocks whose security guarantees are assumed, not re-derived here.

3.6.3. Correctness Properties

The following properties hold under four assumptions: (i) the on-chain registry is append-only, (ii) target writes are digest-verified before being redirected, (iii) committed batches are protected by a collision-resistant hash, and (iv) at least one stable version per object is retained for rollback.

Effective mapping uniqueness: historical tuples coexist in the append-only log, but request routing is governed by exactly one effective pointer per object. Before redirect, the resolver follows the latest stable tuple; after redirect, it follows the new committed tuple. No two effective pointers exist simultaneously for the same object.

Read consistency during migration: under the ordered workflow write–verify–commit–redirect–retire, a query resolves either to the pre-migration stable source or to the post-commitment verified target, but never to a tuple in the intermediate migrating state.

Audit window isolation: during the interval between physical migration and on-chain commitment (the audit window), the tentative tuple is in state migrating and is not reachable from the query path. The effective pointer remains on the predecessor stable tuple, so no query is served from an unfinalized mapping. The redirect controller activates the new pointer only after the batch commit succeeds (Stage 3), ensuring that the audit window is invisible to data plane operations. This property directly addresses the concern that unfinalized on-chain state might expose queries to invalid targets: the effective pointer semantics and the redirect protocol together guarantee that the data plane never observes an uncommitted location.

Tamper detectability of batched updates: the batch root is a deterministic digest of both record content and record order. Any unauthorized insertion, deletion, modification, or reordering of records within a committed batch changes the recomputed root unless a collision is found in the underlying hash function.

Rollback recoverability: because the predecessor stable tuple is retained until the successor is fully committed, verified, and redirected, the control plane can always restore a known-good state after any single-point failure. The six fault types enumerated in Section 3.5 are all recoverable under this invariant.

3.6.4. Complexity

Let N denote the number of active objects in a control round, b the batch size, and L_i the retained version chain length for object d_i. Table 4 summarizes the per-round cost of each component.

Table 4. Per-round computational complexity of the framework components.

Component	Complexity
Heat score update (F_i, R_i, Q_i) per object	$O(1)$
Full heat recomputation and threshold screening	$O(N)$
Utility ranking of migration candidates	$O(N \log N)$
Batch root construction	$O(b)$
Normal mapping lookup	$O(1)$ avg.
Rollback lookup (stable pointer index)	$O(1)$
Rollback lookup (version chain scan, fallback)	$O(L_i)$

The dominant per-round cost is utility ranking at $O(N \log N)$. All metadata operations (lookup, commitment, rollback) are constant-time under normal operation. The batch root construction cost $O(b)$ is amortized across b migration events, yielding $O(1)$ per migration on average.

Version chain length control.

The fallback rollback lookup scans the version chain of length L_i, which grows monotonically with the number of committed migrations for object d_i. To prevent unbounded growth, the framework enforces a retention policy: only the N_{ret} most recent stable versions are retained for rollback and audit; older versions are garbage-collected after their audit window expires. Under the default configuration ($N_{\text{ret}} = 5$), the worst-case version of the chain scan is bounded by $O(N_{\text{ret}}) = O(5) = O(1)$ in practice. Even without the retention cap, L_i is bounded by the number of committed migrations, which grows at most linearly with the simulation horizon divided by the batch window ($L_i \leq T/W_b$). For the default setting ($T = 12{,}000$ ticks, $W_b = 50$), $L_i \leq 240$, making the scan negligible relative to the per-access tier latency.

Deployment-level scalability considerations.

The per-round complexity bounds above govern computational cost within the discrete event simulator. In a real deployment, additional factors affect scalability: the version chain length L_i determines the long-term registry storage footprint; the batch size b determines the per-transaction payload submitted to the consensus layer; and the number of active objects N determines the metadata-coordination communication cost per round. These deployment-level concerns—including the estimated gap between simulated commit latency and real chain finality delay—are addressed quantitatively in the chain delay sensitivity analysis and benefit–complexity discussion in Sections 5 and 6.

4. Experimental Setup

The evaluation is designed to test the four coupled design objectives stated in Section 3.1: efficient tiered access, metadata integrity and auditability, migration consistency, and rollback-safe recovery. Accordingly, the experiment suite goes beyond conventional baseline comparison to include metadata coordination overhead, read/write workload variation with asymmetric write latencies, scalability under increasing object counts, per-fault-type recovery timing, fine-grained ablation of the redirect, rollback, batching components, chain delay sensitivity analysis, tail latency distribution under variable offered load, and a benefit–complexity break-even assessment.

4.1. Simulation Environment and Reproducibility

All experiments are conducted in a discrete-tick event-driven simulator implemented in Python 3.9. No commercial materials, equipment, or devices were used in this study. The simulator models three storage tiers (hot, warm, cold) with configurable per-access latencies, an append-only mapping registry with batched commitment, and the five-stage write–verify–commit–redirect–retire workflow described in Section 3.5. Each simulation tick processes a configurable number of requests drawn from the active workload pattern. In the simulator, one tick corresponds to 1 ms of simulation time; consequently, all tick-based parameters (e.g., batch window, migration interval, audit delay) throughout this paper can be read directly as millisecond values.

Table 5 summarizes the default parameters used throughout the evaluation unless explicitly varied. Key settings include 5000 managed objects, tier capacities of 1000/5000/50,000 objects (hot/warm/cold), per-access latencies of 1/10/100 ms, a batch window $W_b = 50$ ticks, and a migration utility threshold $\theta_u = 0.15$.

To assess statistical stability, each primary experiment is repeated with five independent random seeds; the results section reports mean $\pm$ standard deviation where applicable. The complete simulation source code, experiment runner scripts, configuration files, and raw result data are available upon reasonable request to enable full reproducibility.

Scope and limitations of the simulation.

The simulator abstracts real-world factors such as network jitter, smart-contract gas cost [21,22], and physical I/O variance into configurable tier latency parameters.

All tier latency and commit latency values reported in this paper are simulated parameters (e.g., the default batch commit latency of 0.07 ms represents the simulator's registry write cost, not a measured on-chain transaction latency). To bridge the gap between the simulated commit cost and real chain finality delay, the evaluation includes a chain delay sensitivity analysis that parametrically varies the assumed finality delay and estimates the impact on end-to-end coordination cost (Section 5). A full deployment-level validation on a real blockchain test network remains outside the current scope and is discussed in Section 6.

Table 5. Default simulation parameters used unless otherwise stated.

Parameter	Value	Purpose
Storage		
Tier capacities (hot/warm/cold)	1 k/5 k/50 k obj.	Capacity budget per tier
Tier latencies (hot/warm/cold)	1/10/100 ms	Per-access latency
Replica number	2 (max 5)	Redundancy bound
Heat model		
Decay parameter λ	0.01	Recency decay
Weights $(\alpha_0, \beta_0, \gamma_0)$	(0.40, 0.35, 0.25)	Heat score composition
Volatility sensitivity η	0.10	Weight adaptation
Threshold setting	dynamic	Adaptive tier thresholds
Base thresholds (T_{h0}, T_{c0})	(0.70, 0.30)	Hot/cold cutoffs
Migration		
Trigger mode	hybrid	Periodic + event-driven
Migration interval	100 ticks	Reevaluation interval
Trigger threshold	0.15	Minimum migration utility
Prefetch window/confidence	50 ticks/0.60	Prefetch setting
Blockchain coordination		
Batch window size	50 ticks	Mapping aggregation window
Maximum batch size	100 entries	Commit batch upper bound
Query/recovery		
Max retries/retry delay	3/5 ms	Query retry policy
Max rollback versions	5	Retained stable versions
Simulation		
Number of objects	5000	Object population
Ticks (hotspot/drift/burst)	10 k/15 k/12 k	Scenario durations

4.2. Baselines and Workload Configuration

4.2.1. Baseline Methods

Five methods are compared, following the isolation methodology common in tiered storage evaluations [5,8]. Each baseline disables a specific aspect of the proposed framework, isolating the contribution of that aspect:

- **Pure on-chain** stores all objects on chain. Access latency is high; metadata coordination is trivially consistent because no off-chain tier exists.
- **Single-tier off-chain** places all objects in one off-chain tier (warm-equivalent latency). No tiering or migration occurs; metadata pointers are static.
- **Static tiered** assigns objects to tiers at initialization based on a fixed frequency ranking and never migrates. Metadata is written once and not updated.
- **Dynamic-no-mapping** performs adaptive heat-driven placement and migration identical to the proposed method, but without trusted mapping coordination: metadata

updates are unbatched per-migration single commits with no batching optimization, no redirect protocol, and no rollback path. This is the most informative baseline because it isolates the effect of coupled metadata coordination. To avoid the apparent contradiction of "adaptive placement without a mapping," we clarify the precise semantics of this baseline: it retains the same internal heat model, utility computation, and migration scheduler as the proposed method, and it maintains an internal placement table that lets the migration controller decide what to move next. What it does not retain is the trusted/audited mapping pipeline—batched on-chain commitment, in-flight redirect during the commit window, and the rollback contract that ties physical state to on-chain state. In other words, "no mapping" here means "no coordinated, verifiable, recoverable mapping evolution," not "no internal location table." This baseline is therefore a deliberately idealized stand-in for systems that perform adaptive placement but treat the mapping update as a best-effort side effect (e.g., the placement-only systems surveyed in Section 2.1); it is not claimed to be directly deployable.

- **Proposed** implements the full framework: adaptive placement, batched mapping commitment, redirect during the commitment window, and rollback-safe recovery.

4.2.2. Workload Patterns

Three workload patterns cover distinct access dynamics:

- Fixed hotspot—a stable subset of objects receives the majority of queries throughout the simulation (10,000 ticks).
- Hotspot drift—the hot subset shifts periodically, testing the adaptive heat model's ability to track changing demand (15,000 ticks).
- Burst query—abrupt demand surges on previously cold objects stress both migration throughput and metadata coordination (12,000 ticks).

4.2.3. Extended Experiment Configurations

Beyond the baseline comparison under the three workload patterns, the evaluation includes eight targeted experiments that directly probe the coupled coordination mechanism, covering both core mechanism isolation (E1–E4) and extended stress, ablation, and sensitivity analyses (E5–E8):

1. **E1—Metadata coordination overhead.** Four variants are compared: dynamic baseline without mapping optimization, per-migration single-commit, batched mapping commit, and the full proposed method. The batch window W_b is swept from 1 to 200 to characterize the trade-off between commit frequency and mapping update delay.
2. **E2—Read/write ratio sweep.** Read/write ratios of 90/10, 70/30, 50/50, and 30/70 are tested for both the proposed method and the dynamic-no-mapping baseline, evaluating whether the coordination mechanism remains effective under write-heavy workloads. Additionally, E2 is extended with asymmetric write latencies: per-tier write costs are set to 2× and 5× the corresponding read latency (e.g., hot-tier write at 2 ms or 5 ms vs. read at 1 ms) to test whether the coordination mechanism remains stable when writes are inherently more expensive than reads.
3. **E3—Scalability with object count.** The managed object population is varied over {1 k, 5 k, 10 k, 20 k, 50 k}, with tier capacities scaled proportionally. Both the proposed method and the dynamic-no-mapping baseline are measured to assess whether metadata coordination overhead grows proportionately or disproportionately with scale.
4. **E4—Failure/recovery latency.** Six fault types (Section 3.5) are injected: target write failure, verification failure, commit failure, redirect failure, partial metadata corrup-

tion, and stale-state repair. Each fault is triggered 20–30 times; detection latency, rollback latency, and total service interruption are recorded per trial.

5. **E5—Fine-grained coordination ablation.** Four variants share the same placement decisions, object set, query stream, and random seeds; the only difference is the coordination coupling. Proposed (Full) enables batching ($W_b = 50$), redirect, and rollback. No batching sets $W_b = 1$ (per-migration commit). No redirect disables the redirect protocol. No rollback disables rollback recovery. Three phases stress different components: a migration window query storm (Phase 1, 800 ticks), fault-injected migration stress (Phase 2, 80 faults), and a steady-state burst simulation (Phase 3, 6000 ticks). Results are averaged over five seeds.
6. **E6—Chain delay sensitivity.** The simulated batch commit latency (default 0.07 ms) is replaced by assumed chain finality delays of 0.5 s, 2 s, and 5 s, representative of permissioned consortium chains [21,22]. Average latency, P95 latency, hit ratio, and commit overhead are measured under each delay level to estimate how the coordination mechanism would perform if the simulated registry were replaced by a real on-chain commitment path.
7. **E7—Variable-load tail latency analysis.** Offered load is varied over {10, 25, 50, 75, 100} requests per tick for both the proposed method and the dynamic-no-mapping baseline. Per-request latency is recorded to produce tail-distribution (CDF, boxplot), exceedance, and backlog profiles. This experiment characterizes the latency distribution shape that the fixed-tick P95 metric cannot capture.
8. **E8—Benefit–complexity break-even.** Object count is swept over {1 k, 5 k, 10 k, 20 k, 50 k} under fixed hot-tier share. For each scale, measured benefit metrics (latency reduction, hot-tier hit gain) and operational cost proxies (retained versions, metadata coordination operations, audit delay, redirect state) are computed. A paper-local operational criterion classifies each scale point as effective benefit or diminishing return.

Sensitivity sweeps over heat model weights (α, β, γ), the decay parameter λ, and tier thresholds, together with seed-repeat stability tests across five independent seeds, complete the evaluation suite.

4.3. Evaluation Metrics

The metrics are grouped into six categories aligned with the design objectives and the extended evaluation dimensions.

Access performance.

Average latency (mean query-to-response time across all requests), P95 latency (95th percentile query latency), P99 latency (99th percentile), and throughput (requests processed per simulation tick). For the variable-load tail analysis (E7), two additional indicators are recorded: the exceedance ratio (fraction of requests whose latency exceeds a predefined threshold, e.g., 50 ms) and the cold-tier served ratio (fraction of queries served from the cold tier, proxying for placement misalignment under high load). These measure how effectively the tiered placement serves demand (design objective 1).

Placement effectiveness.

Hot-tier hit ratio (fraction of queries served from the hot tier). A higher hit ratio indicates better alignment between the heat model's placement decisions and actual demand.

Metadata coordination.

Mapping update latency (wall-clock time to enqueue and process a mapping update entry), batch commit latency (simulated registry write cost; the default value of 0.07 ms

represents the simulator's internal commit overhead, not a measured on-chain transaction latency), metadata lookup overhead (wall-clock time for a single registry lookup during query resolution), commit count (total on-chain transactions over the simulation), average batch size (entries per committed batch), and batching reduction ratio (ratio of total mapping events to commit count). These metrics quantify the cost of trusted metadata coordination (design objective 2), following the metadata-overhead measurement rationale advocated in scalable storage literature [15,16], and are the primary output of experiment E1. For chain delay sensitivity (E6), the same commit count and batch size metrics are recomputed under assumed chain finality delays of 0.5 s, 2 s, and 5 s to estimate how coordination cost scales with realistic on-chain latency.

Recovery effectiveness.

Failure detection latency (time from fault occurrence to detection), rollback latency (time from detection to restoration of the stable mapping), service interruption duration (total time a query could observe degraded or unavailable service due to the fault), and successful fallback ratio (fraction of injected faults that complete rollback successfully). These metrics quantify design objectives 3 and 4 and are the primary output of experiment E4.

Coordination ablation metrics.

The fine-grained ablation (E5) reports per-variant consistency and recovery indicators that are not captured by the aggregate metrics above: stale read count (queries resolved to an outdated tier location), wrong tier routing count (queries routed to a tier that no longer holds the object), invalid target access count (queries directed at a post-migration address before the pointer is committed), redirect fallback count (queries that exercised the redirect path), rollback success rate (fraction of faults recovered by the rollback mechanism; 0.00 when rollback is disabled), failed recovery count, and inconsistent state after fault. These indicators reveal which coordination component prevents each class of consistency or recovery failure.

Benefit–complexity proxies.

Experiment E8 decomposes coordination cost into four operational proxies: retained versions proxy (cumulative predecessor tuples kept for rollback safety), metadata coordination ops proxy (total registry mutations), audit delay proxy (mean ticks between migration and on-chain commitment), and redirect state proxy (active redirect entries per tick). On the benefit side, average latency reduction (percentage improvement over the dynamic-no-mapping baseline) and hot-tier hit gain (percentage point increase) quantify the coordination payoff. The break-even criterion is an operational heuristic defined in Section 5, not a universal economic threshold.

4.4. Fault Injection and Recovery Measurement

Experiment E4 requires controlled injection of faults at specific stages of the migration workflow. For each of the six fault types, the simulation framework provides a deterministic injection hook:

- Target write failure—the physical write to the target tier is forced to fail before any metadata mutation occurs.
- Verification failure—the integrity digest of the target replica is corrupted after write, causing the verification stage to reject the tentative tuple.
- Commit failure—the batch commitment transaction is forced to fail, leaving all tuples in the batch in migrating state.

- Redirect failure—the effective pointer switch is blocked after a successful commit, simulating a control plane fault between commitment and activation.
- Partial metadata corruption—selected fields of a committed tuple are silently altered, triggering detection via digest mismatch on the next access.
- Stale-state repair—the effective pointer is forced to reference a physical location that no longer contains a valid replica, simulating a stale pointer scenario.

For each trial, three timestamps are recorded: fault occurrence, detection, and recovery completion. The differences yield detection latency, rollback latency, and total service interruption. Across all fault types, 20–30 independent trials are collected to report mean and maximum values.

5. Experimental Results

This section reports results for eight targeted experiment groups (E1–E8) and a seed-repeat stability test, each targeting a specific aspect of the coupled design objectives defined in Section 3.1. Sections 5.1–5.5 present the core mechanism evaluation; and Sections 5.6–5.9 report the extended stress, ablation, and sensitivity analyses.

5.1. Overall Performance Comparison

Question: does coupling trusted metadata coordination with adaptive placement yield measurable access performance and placement gains over baselines that lack this coupling?

Figure 3 compares the five methods under the default burst query workload. The proposed method achieves an average latency of 10.75 ms, a 28% reduction from the dynamic-no-mapping baseline (14.93 ms). The hot-tier hit ratio rises from 0.19 to 0.65, indicating that the coordinated placement—where metadata evolution confirms each promotion before the next scheduling round—yields substantially better hot-tier utilization than uncoordinated dynamic placement.

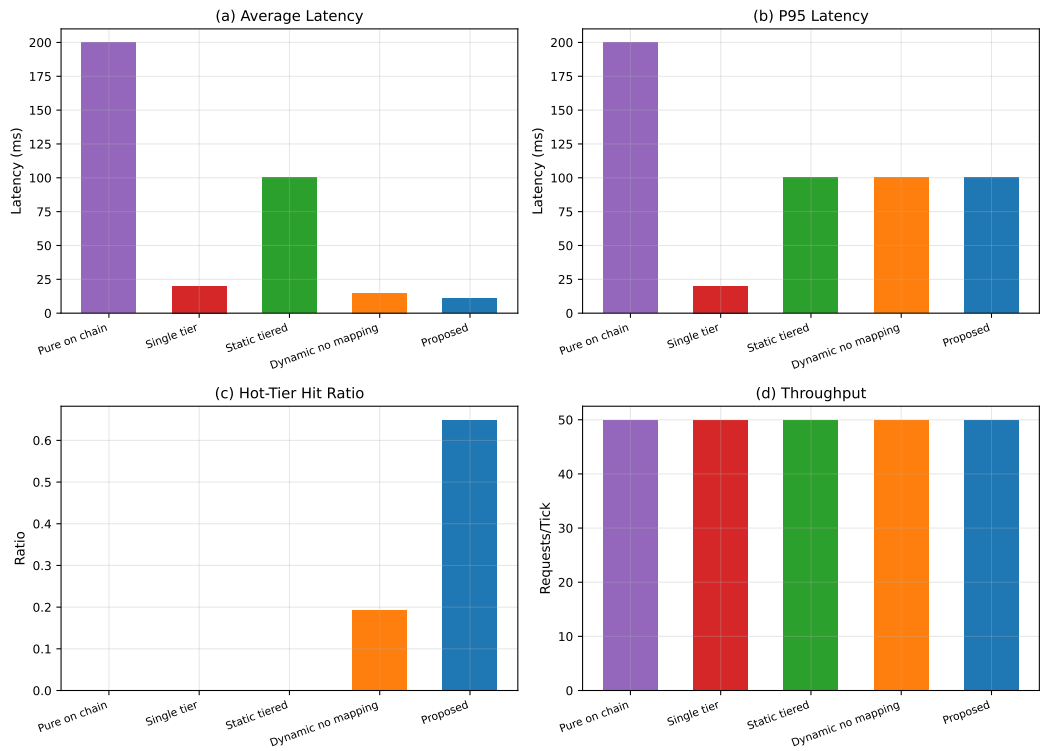


Figure 3. Overall performance comparison under the default burst query workload.

Static-tiered (100 ms), single-tier (20 ms), and pure on-chain (200 ms) storage confirm the expected ordering: without any adaptive placement, latency is dominated by tier mismatch or on-chain overhead. These baselines set the lower bound for what adaptive coordination can improve upon, consistent with the ordering reported in hybrid storage evaluations [5,10].

The P95 latency for the proposed method is 100 ms, equal to that of the dynamic baseline. This reflects a structural property of the discrete-tick simulator: any query that reaches the cold tier incurs a fixed 100 ms latency regardless of method. P95 therefore measures tier coverage breadth rather than tail latency optimization; the average latency and hit ratio metrics are the discriminating indicators for the coordination mechanism. A finer-grained analysis of the per-request latency distribution under variable offered load is provided in Section 5.8.

In Figure 3, the left panel reports average and P95 latencies for each method; the right panel shows overall and hot-tier hit ratios. The P95 equality across all methods that include a cold tier arises because the simulator assigns a fixed 100 ms latency to every cold-tier access, making this metric a structural ceiling rather than a differentiator.

5.2. Metadata Coordination Overhead

Question: what is the measurable cost and benefit of integrating trusted metadata coordination into the tiered storage data path?

This experiment (E1) isolates the metadata coordination mechanism by comparing four variants that share the same adaptive placement logic but differ in how mapping updates are handled. A natural question is whether an intermediate baseline—dynamic placement with per-migration metadata updates but without batching, redirect, or rollback—would provide a finer separation. This configuration is exactly the "per-migration single-commit" variant in Table 6, which retains mapping updates but disables batch coordination and rollback. The four-variant design therefore already isolates the contribution of batching and coordination from the mere presence of metadata updates, without requiring a dedicated additional baseline. Table 6 summarizes the key metrics.

Table 6. Metadata coordination variant comparison (E1). All variants use the same adaptive placement engine; they differ only in mapping update strategy.

Variant	Avg Lat. (ms)	Commits	Batch Sz.	Commit Lat. (ms)	Hot Hit
No mapping opt.	14.93	4000	1	0.004	0.19
Per-migration single	10.75	4000	1	0.004	0.65
Batched mapping	10.75	80	50	0.066	0.65
Proposed (Full)	10.75	80	50	0.070	0.65

Two observations are central. First, the latency gap between "no mapping optimization" (14.93 ms) and the three mapping-aware variants (10.75 ms) confirms that trusted mapping coordination is not merely an integrity add-on: it directly improves placement effectiveness by confirming promotions and preventing stale pointer conflicts that force the dynamic baseline to serve more queries from slower tiers.

Second, batching reduces the on-chain commit count from 4000 to 80—a 50× reduction—achieving high-throughput mapping operations without any change in access latency or hit ratio. The cost of this reduction is a higher per-batch commit latency (0.066–0.070 ms vs. 0.004 ms per single commit) and a mapping freshness delay of 25.5 ticks on average. These commit latencies are simulated registry write costs inside the simulator (Section 4.1); they do not represent measured on-chain transaction times. The sensitivity of

coordination cost to realistic chain finality delays is examined separately in Section 5.7. During this delay window, the effective mapping still points to the pre-migration stable pointer, so queries are not mis-routed, but the committed on-chain state lags the physical state.

Figure 4 visualizes these results across three panels. Panel (a) reports per-operation latency on a logarithmic scale: lookup latencies remain at the 10^{-4} ms level (in-memory registry reads), while commit latencies span two orders of magnitude depending on the batching strategy ($\sim 4 \times 10^{-3}$ ms for per-commit variants vs. $\sim 7 \times 10^{-2}$ ms for batched variants). The logarithmic axis is used because lookup costs are roughly two orders of magnitude smaller than commit costs and would otherwise be visually indistinguishable from zero on a linear scale. Panel (b) shows that the total number of mapping entries committed (4000) is identical across all variants, confirming that batching reduces commit frequency without dropping any mapping records. Panel (c) shows the batch window sweep from $W_b = 1$ to $W_b = 200$: commit count decreases monotonically and saturates beyond $W_b \approx 100$ (at 40 commits), while the average mapping update delay grows linearly. This confirms a tunable trade-off between coordination overhead and mapping freshness—a knob that system operators can set according to their acceptable staleness tolerance—a tuning principle analogous to the freshness–overhead trade-off in batched file-system metadata [16].

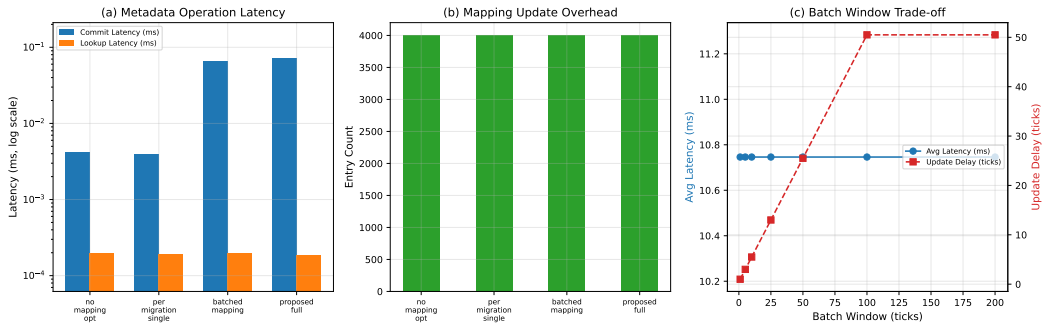


Figure 4. Metadata coordination overhead (E1): variant comparison and batch window sweep. Panel (**a**) uses a logarithmic y-axis because lookup latencies are roughly two orders of magnitude smaller than commit latencies.

5.3. Performance Under Read/Write Mixes and Asymmetric Write Latency

Question: does the coordination mechanism remain effective when the write ratio increases from 10% to 70%, and when write operations are inherently more expensive than reads?

Figure 5 reports the proposed method and the dynamic-no-mapping baseline across four read/write ratios (90/10, 70/30, 50/50, 30/70) in three panels. Panel (a) shows average end-to-end latency: the proposed method stays at ≈10.75 ms across all four mixes, while the baseline remains at ≈14.93 ms. Panel (b) shows hot-tier hit ratio: the proposed method sustains ≈0.65 hot hits, more than 3× the baseline's ≈0.19, confirming that trusted mapping coordination directly improves placement effectiveness regardless of the write fraction. Panel (c) reports the average commit latency on a logarithmic scale (because per-commit and batched-commit costs differ by more than an order of magnitude): batched commits in the proposed method stay at $\sim 7 \times 10^{-2}$ ms (the cost of aggregating ∼50 mapping entries per commit), whereas the baseline issues one commit per migration at $\sim 4 \times 10^{-3}$ ms. The aggregate on-chain work—commits per unit time multiplied by per-commit latency—is nevertheless much lower for the proposed method (80 commits vs. 4000) at every ratio, as already established in Table 6.

The stability of these metrics warrants a cautious interpretation. Because the simulator decouples read latency from write latency at the tier level, increasing the write fraction does not proportionally change the average access time. What the experiment does confirm is

that the metadata coordination mechanism—batched commitment, redirect, and rollback—does not degrade under higher write pressure: the number of registry updates increases (from 24 k at 90/10 to 145 k at 30/70), yet the batching mechanism absorbs this growth without increasing commit count or per-batch latency. This behavioral robustness under mixed workloads is relevant evidence that the framework functions as a data management system, not merely a read-side cache.

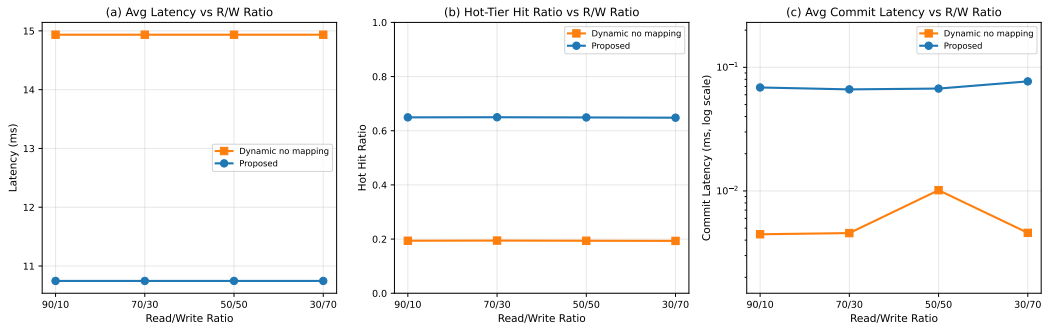


Figure 5. Performance under read/write ratio sweep (E2). Panel (**c**) uses a logarithmic *y*-axis because per-commit (baseline) and batched-commit (proposed) latencies differ by more than an order of magnitude. In every panel, the proposed method (blue) is plotted on top of the baseline (orange) so that the two curves are always distinguishable.

The proposed method maintains stable latency and hit ratios from 90/10 to 30/70, while metadata update volume increases sixfold.

Asymmetric write latency stress.

To test whether the coordination mechanism remains stable when writes are inherently more expensive than reads, E2 is extended with per-tier write costs set to 2× and 5× the corresponding read latency (e.g., hot-tier write at 2 ms or 5 ms vs. read at 1 ms). Figure 6 reports the results.

Under the 2× multiplier, the proposed method's average latency rises moderately—from 11.06 ms (symmetric) to 12.18 ms at 90/10 and to 18.84 ms at 30/70—while the baseline increases from 15.39 ms to 16.94 ms and 26.19 ms, respectively. At 5×, the proposed method reaches 15.55 ms (90/10) and 42.17 ms (30/70), compared with 21.61 ms and 58.62 ms for the baseline. In every configuration, the proposed method maintains a lower average latency than the baseline, and the relative advantage widens under heavier write stress: at 30/70 with 5×, the proposed method achieves a 28% latency reduction, essentially preserving the symmetric advantage.

Figure 7 shows P95 and P99 latencies under asymmetric write costs. Each percentile is plotted in a separate panel: (a) Proposed – P95, (b) Proposed – P99, (c) Dynamic – P95, and (d) Dynamic – P99. Each panel contains exactly three curves corresponding to the three write multipliers (Write 1×, 2×, 5×). At 5× with high write ratios (50/50 and 30/70), both methods see P99 rise to 500 ms—the cold-tier write penalty—but the proposed method retains a lower average and a higher hot-tier hit ratio, confirming that the coordination mechanism mitigates write path degradation.

Figure 8 presents commit overhead in three panels with a logarithmic *y*-axis: (a) batch commit count, (b) average batch size, and (c) average mapping update delay. Across all four R/W ratios and all three write multipliers, the proposed method's batch commit count stays at 120 (vs. 6000 for the baseline—a 50× reduction) and its average batch size at 50 (vs. 1), independent of the write multiplier; this confirms that the batching mechanism is insensitive to per-operation write cost. The delay panel reports the inherent trade-off: batching introduces a ∼25.5-tick average commit delay for the proposed method (vs. 1 tick

for the per-migration baseline), which is the freshness cost paid in exchange for the 50× reduction in on-chain commit volume.

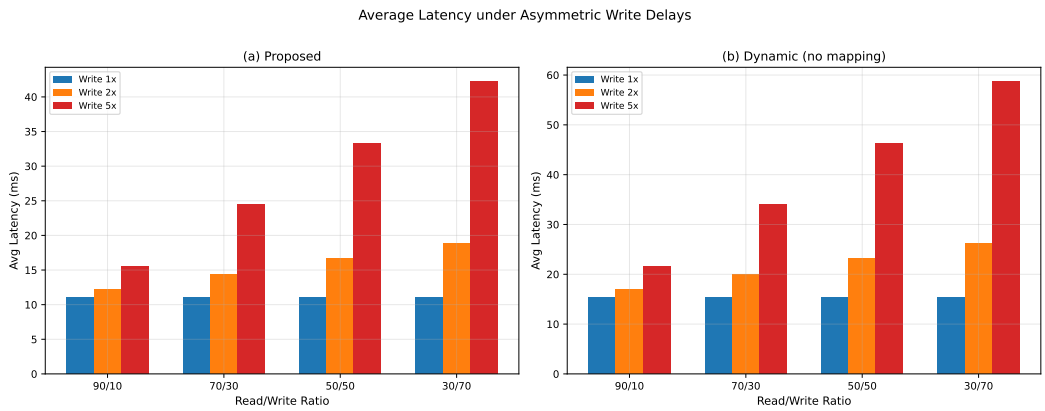


Figure 6. Average latency under asymmetric write costs (E2 extension).

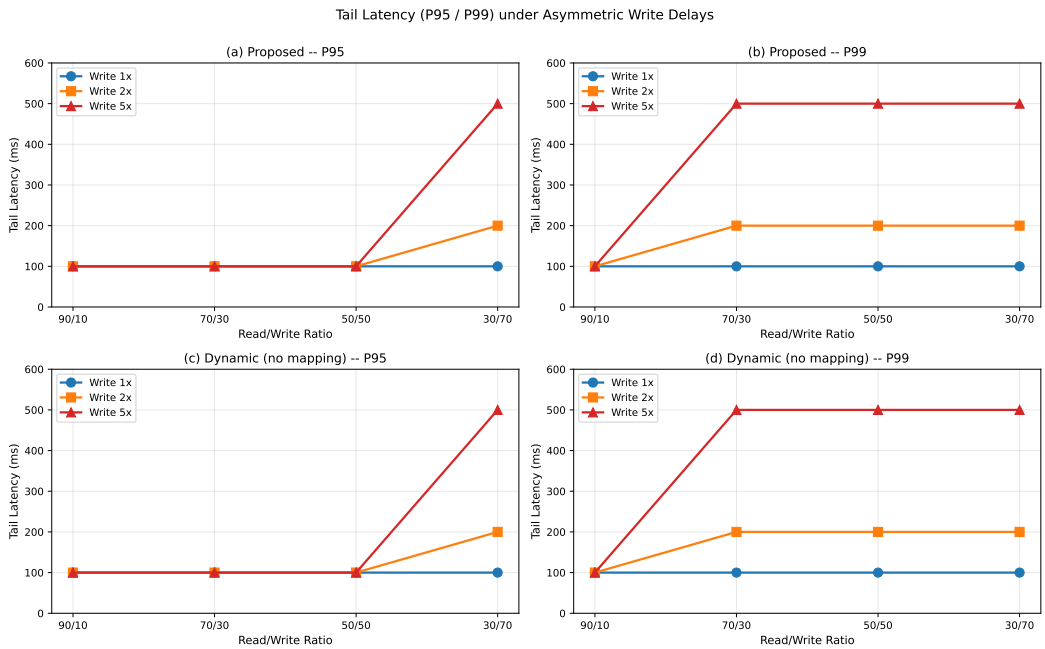


Figure 7. P95 and P99 tail latency under asymmetric write costs (E2 extension). The four panels separate each (method, percentile) pair so that each panel contains exactly three curves—one per write multiplier (1×, 2×, 5×)—and every legend entry corresponds to a unique color and marker.

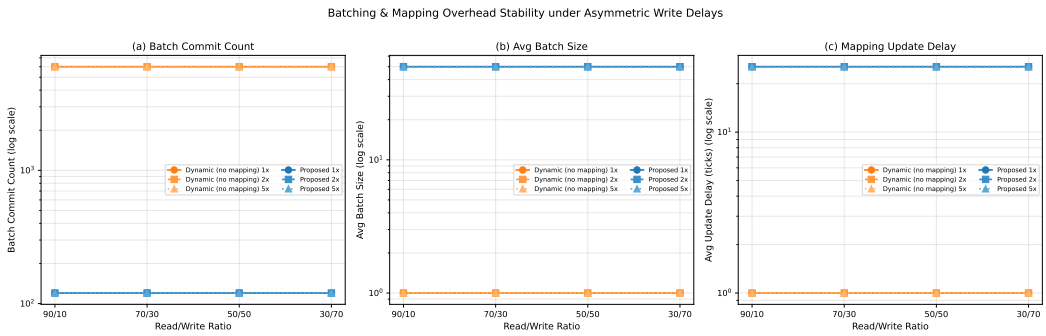


Figure 8. Commit overhead under asymmetric write costs (E2 extension). All three panels use a logarithmic *y*-axis because the proposed and baseline curves differ by 1–2 orders of magnitude. Blue family = proposed (Write 1×/2×/5×); orange family = dynamic-no-mapping baseline (Write 1×/2×/5×).

5.4. Scalability with Increasing Object Count

Question: does the coordination overhead grow proportionately or disproportionately as the managed object population increases?

Figure 9 reports average latency and hot-tier hit ratio as the object count grows from 1 k to 50 k (tier capacities scale proportionally), effectively managing tens of millions of bytes of data (10+ MB scale) at the upper end. Both the proposed method and the dynamic baseline show monotonically increasing latency—an expected consequence of the growing ratio of cold-tier objects to fixed hot-tier capacity at each proportional scale.

At 1 k objects, the proposed method achieves 7.35 ms (vs. 10.30 ms baseline), a 29% reduction. At 50 k the gap narrows: 76.63 ms vs. 78.19 ms, a 2% reduction. This narrowing occurs because at large scales, the hot-tier capacity fraction shrinks relative to the total object set, limiting the fraction of queries that can benefit from the tier promotion, regardless of coordination quality. The proposed method nevertheless maintains a consistently lower latency and higher hot-tier hit ratio at every tested scale, confirming that the coordination mechanism does not introduce disproportionate overhead as the managed population grows, aligning with the linear metadata-scaling behavior observed in distributed storage systems [6,15].

At 50 k objects, the proposed method's average latency reaches 76.63 ms, approaching the cold-tier latency floor of 100 ms. Under the proportional capacity-scaling configuration used here, the hot-tier capacity fraction becomes too small to serve a meaningful share of queries at this scale. In operational terms, deployments at $\geq$50 k object scale would need either a larger hot-tier capacity ratio or a hierarchical tiering strategy to maintain single-digit-millisecond average latency. The metadata coordination mechanism itself does not cause this saturation—its overhead remains proportional—but it cannot overcome a fundamental tier-capacity constraint. The benefit–complexity analysis in Section 5.9 quantifies this narrowing explicitly.

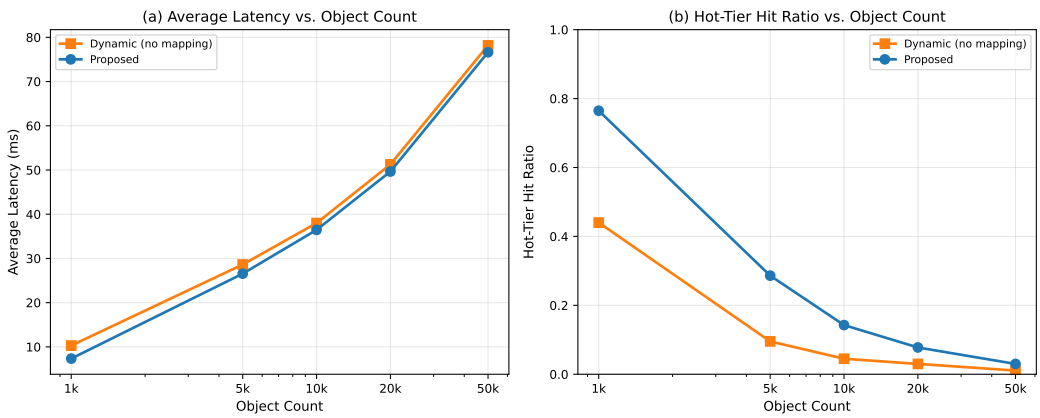


Figure 9. Scalability from 1 k to 50 k objects (E3). (**a**) average latency vs. object count; (**b**) hot-tier hit ratio vs. object count. The two methods yield clearly distinct values at every tested scale.

In panel (a), the proposed method maintains a lower average latency at every tested object count; in panel (b), it sustains a substantially higher hot-tier hit ratio. The gap narrows at 50 k due to hot-tier capacity saturation rather than coordination overhead.

5.5. Failure Detection and Recovery Effectiveness

Question: when the migration workflow fails, how fast is detection, rollback, and service restoration—and does recovery cost differ across fault types?

Table 7 reports the per-fault-type recovery metrics from experiment E4. All six fault types achieve 100% successful rollback across 20–30 trials per type, and total service interruption remains below 0.11 ms in every trial.

Table 7. Failure recovery timing by fault type (E4). All values in milliseconds; "Interr." = total service interruption duration.

Fault Type	Detect.	Rollback	Avg Interr.	Max Interr.
Target write failure	0.005	0.005	0.005	0.016
Verification failure	0.005	0.031	0.036	0.107
Commit failure	0.001	0.021	0.021	0.033
Redirect failure	0.001	0.020	0.020	0.025
Partial metadata corrupt.	0.001	0.023	0.024	0.060
Stale-state repair	0.007	0.029	0.036	0.036

The results differentiate recovery cost by fault type. Target write failure is the cheapest to recover: the write fails before any metadata mutation, so rollback consists only of discarding the incomplete copy (0.005 ms average interruption). Verification failure is the most expensive: the system must detect a digest mismatch, discard the tentative tuple, and restore the source mapping, yielding the longest average interruption (0.036 ms) and the highest single-trial maximum (0.107 ms). The remaining four fault types—commit failure, redirect failure, partial corruption, and stale-state repair—cluster in the 0.020–0.036 ms range, reflecting the additional rollback work required when metadata has been partially mutated.

Scope of recovery claims.

The sub-millisecond interruption figures above describe local data plane recovery: the time from fault occurrence to restoration of a stable mapping that the query path can use. They do not include the time required for the corrected state to become audit-visible on the blockchain. Under the simulator's default commit latency of 0.07 ms (an abstracted registry write cost, not a real-chain measurement), audit-visible recovery coincides with local recovery. Section 5.7 shows that once a realistic chain finality delay is assumed, audit-visible recovery time is dominated by that delay (e.g., 500 ms–5 s), while the local data plane interruption remains unchanged.

For heavily regulated IoT verticals, such as finance, healthcare, energy, and industrial control, the operationally binding notion of "recovery" is the audit-visible one rather than the data plane one: a query that returns a value without an updated audit record may satisfy the user but does not satisfy compliance auditors, who require an immutable post-recovery trace before the event is considered closed. We therefore make the recovery time objective in such deployments explicit: it is approximately 0.04 ms (local) + Δ_{chain} in the present design, with Δ_{chain} on the order of 0.5–5 s for typical consortium chains [21,22]. The sub-millisecond local figure is reported throughout this paper because it is the property the coordination mechanism actually controls; the chain finality term is reported separately because it is set by the choice of consensus platform, not by the coordination workflow itself, and we revisit its deployment implications in Section 6.4.

Figure 10 visualizes these timing distributions. The key takeaway for the database systems and data protection audience is that rollback-safe recovery is not a binary success/failure claim but a quantified service interruption budget: different fault types incur different costs, and system designers can use these numbers to set recovery time objectives [11,26].

Protection goal validation mapping.

Table 8 connects each protection goal defined in Section 3.6 to the experiment(s) and metric(s) that provide its empirical validation.

Table 8. Mapping of protection goals to experimental evidence.

Goal	Description	Experiment	Evidence
PG-1	Integrity	E4: corruption	Mismatch detected; 100% rollback
PG-2	Auditability	E1: variants	Chain retained; roots verified
PG-3	Consistency	E4; E5: ablation	No uncommitted-pointer query (E4); redirect eliminates stale reads (E5)
PG-4	Recovery	E4; E5: ablation	100% success; max 0.107 ms local interruption; rollback ablation in E5

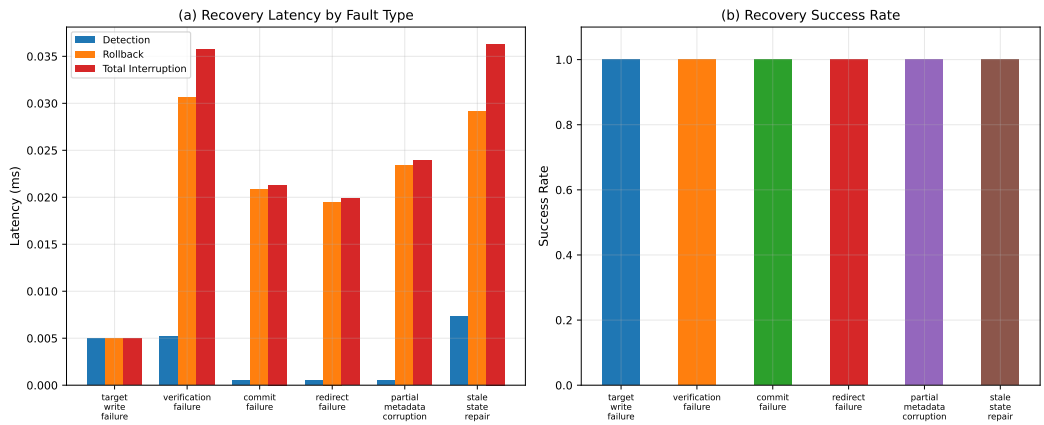


Figure 10. Failure recovery latency by fault type (E4).

Detection, rollback, and total service interruption times are shown for each of the six injected fault types. All figures refer to local data plane recovery; audit-visible latency depends on the assumed chain finality delay (Section 5.7).

5.6. Fine-Grained Coordination Ablation

Question: which individual coordination component—batching, redirect, or rollback—contributes which specific class of benefit, and what is the cost of disabling each one in isolation?

Experiment E5 compares four variants that share identical placement decisions, object sets, query streams, and random seeds. The only difference is the coordination coupling: proposed (Full) enables batching ($W_b = 50$), redirect, and rollback; No batching sets $W_b = 1$; no redirect disables the redirect protocol; and no rollback disables the rollback mechanism. Three phases stress different components: a migration window query storm (Phase 1, 800 ticks), fault-injected migration stress (Phase 2, 80 faults), and a steady-state burst simulation (Phase 3, 6000 ticks). Results are averaged over five seeds.

Consistency cost (redirect ablation).

Figure 11 reports consistency-related metrics from Phase 1. Disabling redirect (no redirect) produces 1866 stale reads and 1866 wrong tier routing events during the commit window; the other three variants record zero for both counters. No redirect also incurs 190 invalid target accesses and raises Phase 1 average latency from 34.38 ms (Full) to 45.08 ms because queries hitting mid-flight objects are served from the slower source tier instead of being forwarded to the freshly written replica. The explicit **0** labels on Full, no batching, and no rollback bars confirm that these are true zero values, not missing data.

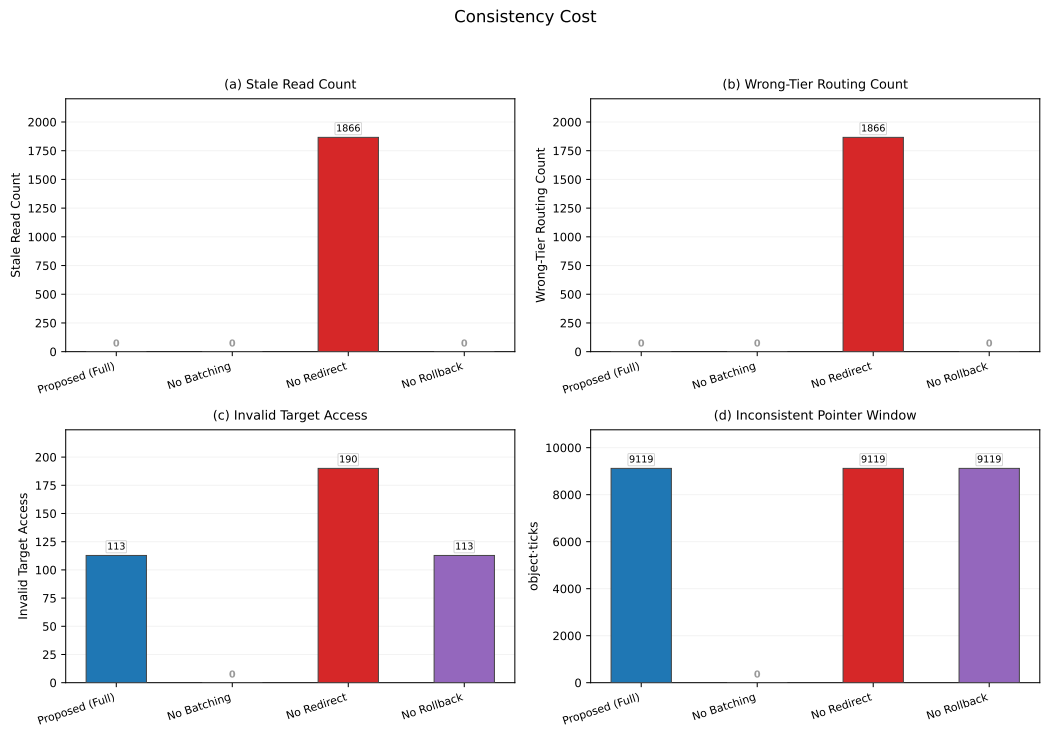


Figure 11. Consistency cost by ablation variant (E5).

Four sub-panels show (a) stale read count, (b) wrong tier routing count, (c) invalid target access count, and (d) redirect fallback count. Each bar carries an explicit value label; bold "0" markers indicate genuine zero values.

Failure exposure (rollback ablation).

Figure 12 reports Phase 2 recovery metrics. No rollback drops the rollback success rate from 1.00 to 0.00 and accumulates 80 failed recoveries; all 80 affected objects remain in an inconsistent state after the fault burst. The remaining variants achieve a 100% recovery rate with zero failed recoveries.

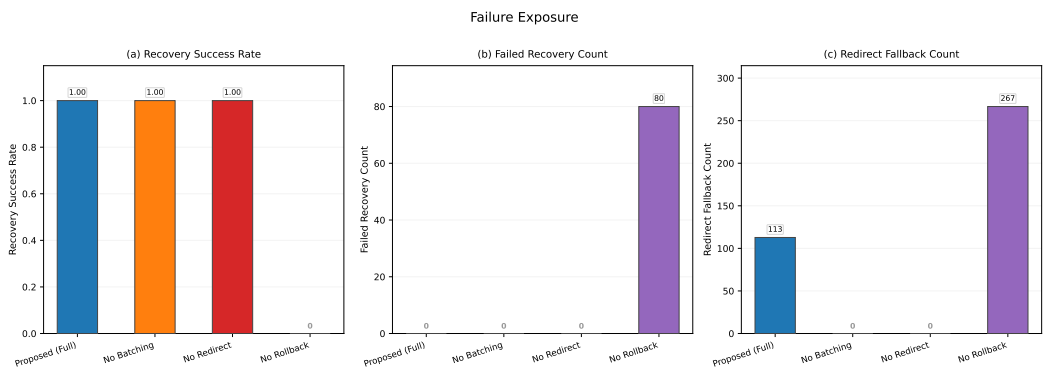


Figure 12. Failure exposure by ablation variant (E5).

Panel (a) uses a lollipop chart for the rollback success rate to make the 1.00 → 0.00 drop visually unambiguous; panels (b) and (c) show failed recovery count and inconsistent state count.

Coordination overhead (batching ablation).

Figure 13 reports Phase 3 overhead metrics. No batching produces 6000 on-chain commits versus 120 for the other three variants—a 50× increase—and collapses the average

batch size from 50.0 to 1.0. Phase 1 average latency is highest for no redirect (45.08 ms) rather than no batching (34.45 ms).

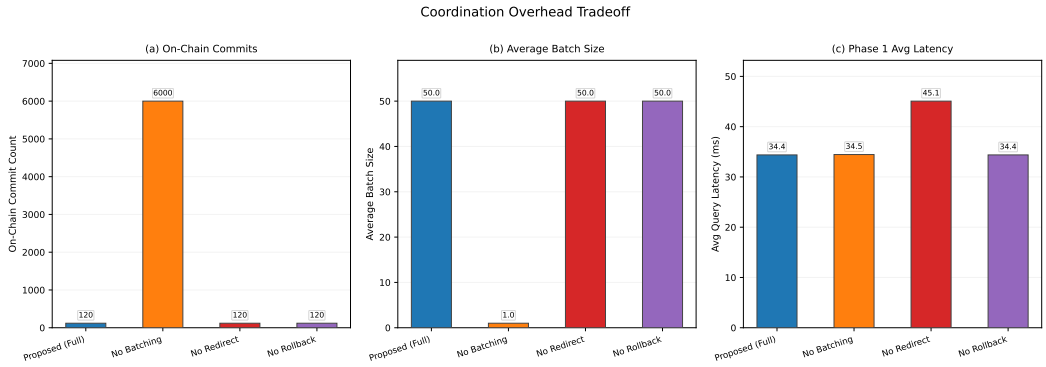


Figure 13. Coordination overhead by ablation variant (E5).

Three independent panels show (a) simulated commit count, (b) average batch size, and (c) Phase 1 average latency. Each panel uses its own y-axis scale so that the 50× commit count gap is readable alongside the per-query latency difference.

Summary of component roles.

(i) Batching reduces on-chain commit overhead by 50× but does not by itself fix per-query consistency during the commit window. (ii) Redirect preserves read consistency by forwarding in-flight queries to the freshly written replica; without it, every mid-flight query is served from the stale source tier. (iii) Rollback guarantees recovery safety under faults; without it, objects are permanently stuck in a half-migrated state, and subsequent reads keep failing. The three components are therefore complementary: batching controls overhead, redirect controls consistency, and rollback controls recovery.

5.7. Chain Delay Sensitivity and Estimated Deployment Gap

Question: how would the coordination mechanism perform if the simulated registry were replaced by a real on-chain commitment path with non-trivial finality delay?

Experiment E6 parametrically replaces the default 0.07 ms simulated commit latency with assumed chain finality delays of 0.5 s, 2 s, and 5 s. Figures 14 and 15 report the results.

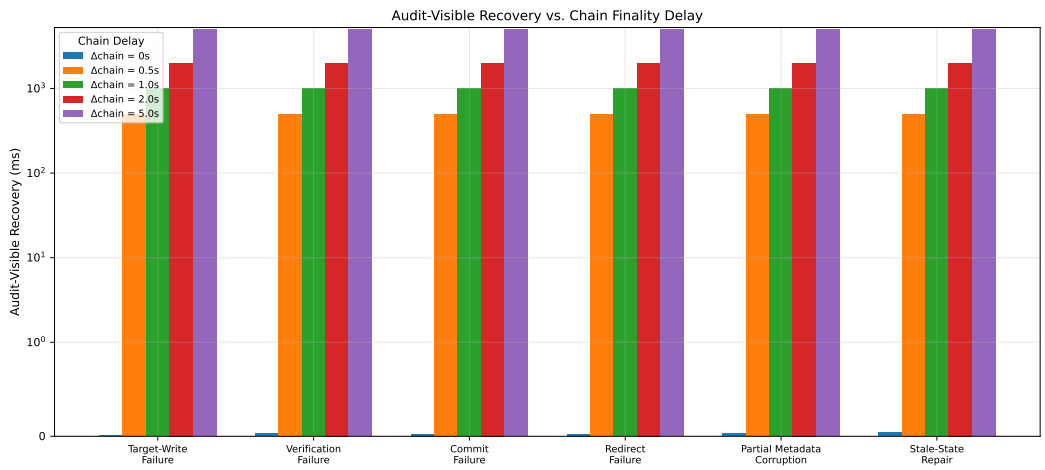


Figure 14. Chain delay sensitivity: per-fault-type recovery (E6).

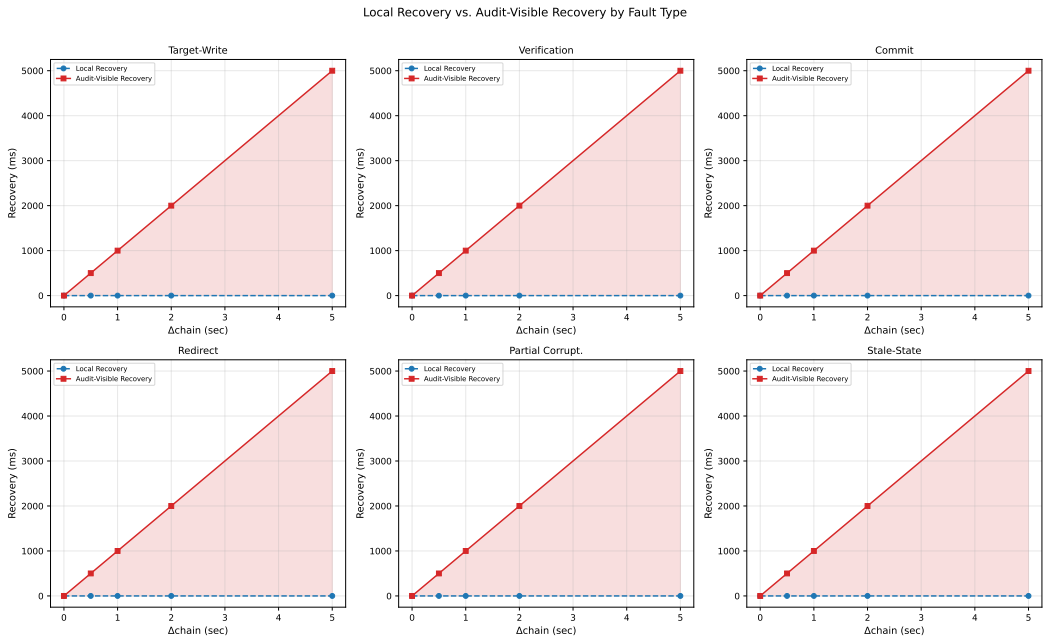


Figure 15. Chain delay sensitivity: recovery time trend (E6).

Two findings are central. First, local data plane recovery remains unchanged across all delay levels: the average local service interruption for target write failure stays at ∼0.006 ms, and verification failure stays at ∼0.03–0.06 ms, regardless of whether chain finality is 0.5 s or 5 s. This confirms that the coordination mechanism's data plane path is decoupled from the on-chain commitment path.

Second, audit-visible recovery time is dominated by the assumed chain finality delay. For example, a target write failure with $\Delta_{\text{chain}} = 2$ s yields an audit-visible recovery time of ∼2000 ms, of which only 0.006 ms is actual data plane recovery. At $\Delta_{\text{chain}} = 5$ s, the audit-visible time reaches ∼5000 ms. Table 9 summarizes the decomposition. This result makes explicit a distinction that the simulator's default 0.07 ms commit latency obscures: sub-millisecond recovery refers specifically to local data plane service restoration; end-to-end audit-visible recovery additionally includes the chain finality delay and is therefore orders of magnitude slower. Any deployment on a real permissioned chain must account for this gap in its recovery time objectives.

Table 9. Estimated gap from simulation to real deployment: recovery time decomposition.

Chain Delay	Local Recovery (ms)	Audit-Visible (ms)	Interpretation
0 (simulated)	0.005–0.041	0.005–0.041	Baseline: audit = local
0.5 s	0.006–0.050	500.01–500.05	Chain dominates
2 s	0.006–0.045	2000.01–2000.05	Chain dominates
5 s	0.006–0.056	5000.01–5000.06	Chain dominates

Impact of chain latency on hot-tier hit ratio and average access latency.

The recovery time decomposition above isolates the recovery cost from the chain delay. A related concern is steady-state access performance: once Δ_{chain} grows to seconds, every newly migrated object remains in the migrating state—and therefore continues to be served from its source tier—until the next batch commits. For a hot promotion candidate this means that, for up to one $(W_b \cdot \delta_{\text{tick}} + \Delta_{\text{chain}})$ window after the physical move, queries still incur source-tier latency rather than hot-tier latency. Three observations bound its practical impact in the present configuration:

- The 28% average latency reduction and the $0.19 \rightarrow 0.65$ hot-tier hit improvement in Section 5.1 are measured under the simulated commit latency of 0.07 ms, in which the per-batch audit window is dominated by $W_b \cdot \delta_{\text{tick}}$ ($\approx$25.5 ticks) rather than by Δ_{chain}. As Δ_{chain} grows, the per-migration "mapping lag" grows proportionally, so the average latency advantage observed at $\Delta_{\text{chain}} \approx 0$ is an upper bound rather than a deployment-realistic figure.
- The fraction of queries actually affected by mapping lag is determined by the migration rate and the lifetime of each migrated object in the hot tier. Objects that remain hot for many windows after promotion still receive the hot-tier benefit on all but their first window of residence; the lag penalty is paid once per migration, not on every access. In workloads with stable hotspots, this amortizes the lag rapidly; in workloads with very high migration churn—e.g., aggressive hotspot drift under $\Delta_{\text{chain}} \geq 5$ s—the residual hot hit advantage can shrink substantially, and we treat that regime as outside the operating envelope claimed in this paper.
- The audit window isolation property (Section 3.6) guarantees that the effective pointer never advances ahead of the chain; queries during the lag window are therefore served from a correct, verified, but slower tier rather than from a stale or invalid mapping. In other words, large Δ_{chain} erodes the latency advantage but never the correctness or audit guarantees that motivate the framework in the first place.

The practical implication for deployment is that W_b should be tuned jointly with the measured chain finality distribution (Section 6.2); on chains whose 95th percentile finality exceeds a few seconds, deployments that depend critically on low access latency should also consider the placement-side mitigations discussed in Section 6.3 (preheating, prefetching, collaborative placement, RL-tuned thresholds), which reduce the number of mapping changes required per unit of workload and thereby shrink the fraction of accesses exposed to the lag window.

5.8. Variable-Load Tail Latency Analysis

Question: how does the per-request latency distribution change as the offered load increases, and does the proposed method maintain a tighter distribution than the baseline?

Experiment E7 varies the offered load over $\{10, 25, 50, 75, 100\}$ requests per tick and records every individual request latency. Figure 16 shows grouped boxplots for the proposed method and the dynamic-no-mapping baseline at each load level.

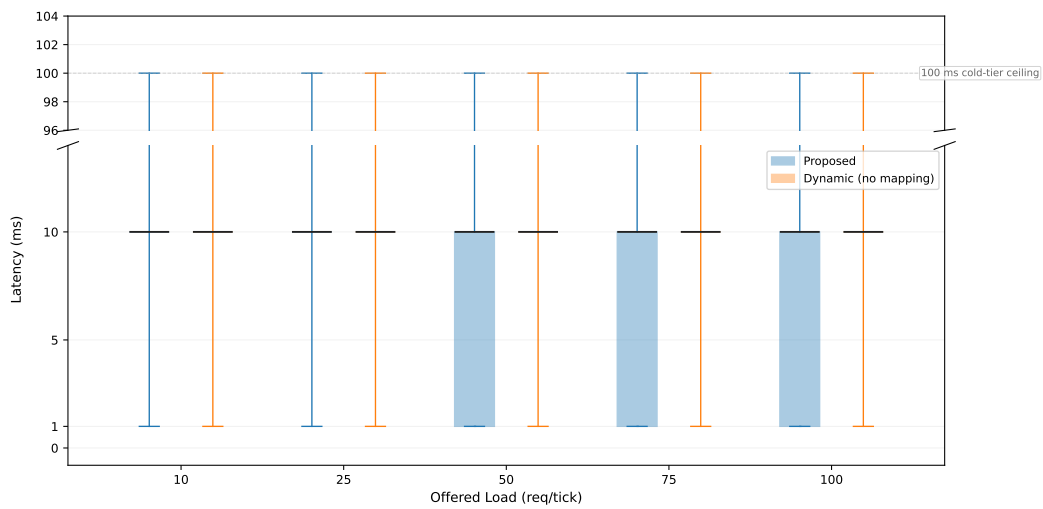


Figure 16. Latency distribution by load level (E7). A broken y-axis separates the low-latency body from the 100 ms cold-tier ceiling.

A broken y-axis separates the low-latency body (0–14 ms) from the 100 ms cold-tier ceiling so that the uninformative 15–95 ms gap is eliminated. At loads ≥50 req/tick, the proposed method's interquartile range extends from 1 ms (hot tier) to 10 ms (warm tier), whereas the baseline's box collapses to a single line at 10 ms, indicating that its Q1 never reaches the hot tier. Both methods share the same cold-tier ceiling (100 ms), hence having identical P95/P99 at all tested loads; the broken axis boxplot exposes the distributional difference that a single percentile cannot reveal.

Figure 17 overlays the CDF curves for both methods at each load level. The proposed method's CDF rises faster below the 20 ms mark, reflecting its higher hot-tier hit ratio (e.g., 0.44 vs. 0.25 at 100 req/tick). Beyond 100 ms, the two CDFs converge because cold-tier queries dominate the right tail identically for both methods.

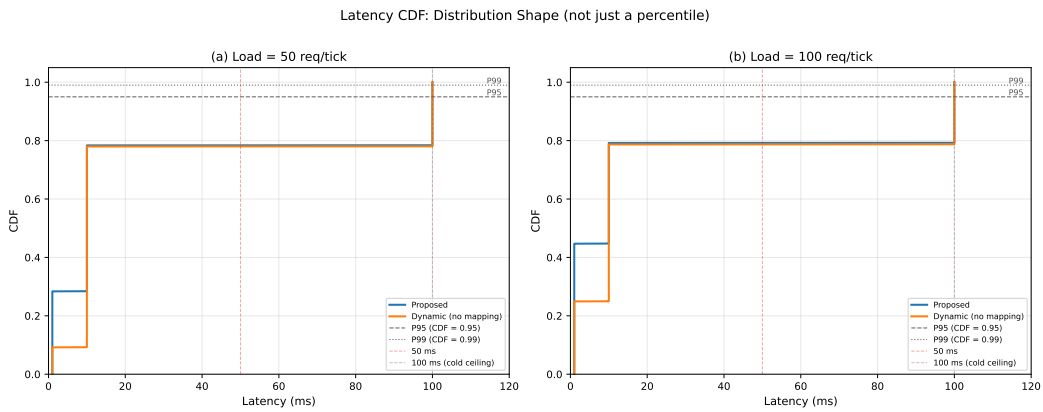


Figure 17. Per-request latency CDF by load level (E7). In each subplot the horizontal grey dashed line at CDF = 0.95 marks the P95 reference and the horizontal grey dotted line at CDF = 0.99 marks the P99 reference; the corresponding "P95"/"P99" labels are printed at the right edge of each subplot. The vertical red dashed line at 50 ms and the vertical brown dashed line at 100 ms mark the tail-exceedance threshold and the cold-tier latency ceiling, respectively.

Figure 18 plots the exceedance ratio (fraction of requests exceeding 50 ms) and Figure 19 shows the request backlog proxy (average pending latency) as load increases. Both metrics increase monotonically with load for both methods, but the proposed method consistently maintains a lower backlog. The primary differentiator is the hot-tier hit ratio: at 100 req/tick, the proposed method achieves 0.44 versus 0.25 for the baseline, resulting in a lower average latency (24.73 ms vs. 26.90 ms) despite identical P95/P99 ceilings. The exceedance ratios are close at each load level (∼0.21 for both methods at load 50) because the cold-tier ceiling dominates; the distribution within the non-cold-tier portion—visible in the boxplot and CDF—is where the coordination mechanism's advantage appears.

The tail latency results complement the fixed-tick P95 metric from Section 5.1 by showing that, although P95 and P99 are identical across methods (both capped by the cold-tier floor), the shape of the distribution differs meaningfully: the proposed method concentrates a larger fraction of requests in the low-latency region.

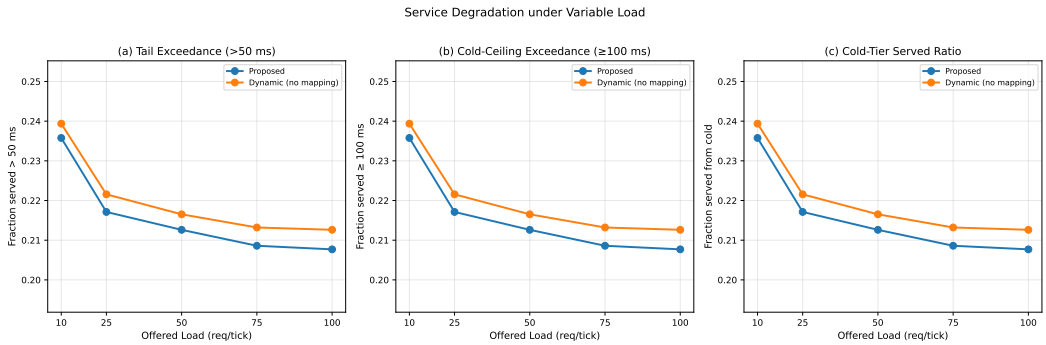


Figure 18. Service degradation under variable load (E7). (**a**) tail exceedance (>50 ms); (**b**) cold-ceiling exceedance (≥100 ms); (**c**) cold-tier served ratio. Blue circles: proposed method; orange squares: dynamic-no-mapping baseline. Only the two adaptive methods are plotted; the static-tiered ceiling is a flat 1.0 at every load level and was omitted because including it would compress the y-axis range of the two adaptive curves and hide their relative gap.

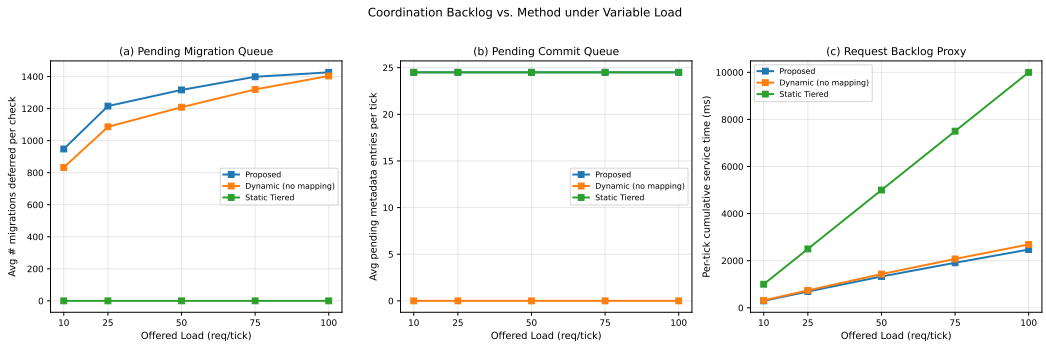


Figure 19. Coordination backlog vs. offered load (E7). (**a**) pending migration queue (avg deferred per check); (**b**) pending commit queue (avg pending metadata entries per tick); (**c**) request backlog proxy (per-tick cumulative service time). Blue squares: proposed; orange squares: dynamic-no-mapping baseline; green squares: static-tiered ceiling. The static-tiered method shows zero pending migration and commit queues (panels (**a**,**b**)) because all migration is disabled; in panel (**c**) its backlog proxy is dominated by the cold-tier constant latency.

5.9. Benefit–Complexity and Break-Even Analysis

Question: at what object count scale does the coordination mechanism's benefit narrow to the point of diminishing return, and why?

Experiment E8 sweeps the object count over {1 k, 5 k, 10 k, 20 k, 50 k} under fixed hot-tier share. Two measured benefit metrics—average latency reduction (percentage improvement over the dynamic-no-mapping baseline) and hot-tier hit gain (percentage point increase)—are plotted alongside four operational cost proxies: retained versions proxy, metadata coordination ops proxy, audit delay proxy, and redirect state proxy.

Figure 20 shows the benefit side. At 1 k objects, the coordination mechanism achieves a 33% latency reduction and a 34-percentage-point hot-tier hit gain. These benefits narrow monotonically: at 50 k, the latency reduction is 2.7% and the hit gain is 2.6 percentage points.

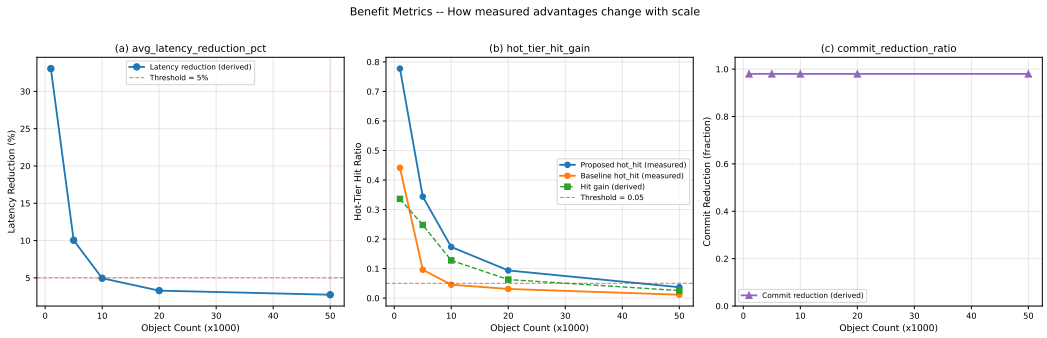


Figure 20. Measured benefit metrics vs. object count scale (E8).

Figure 21 shows the cost side. Retained versions proxy and metadata coordination ops proxy grow approximately linearly with object count—a natural consequence of more objects generating more mapping updates—while audit delay proxy and redirect state proxy remain constant (25.5 ticks and 0, respectively), confirming that these overheads are scale-invariant under the current batch window setting.

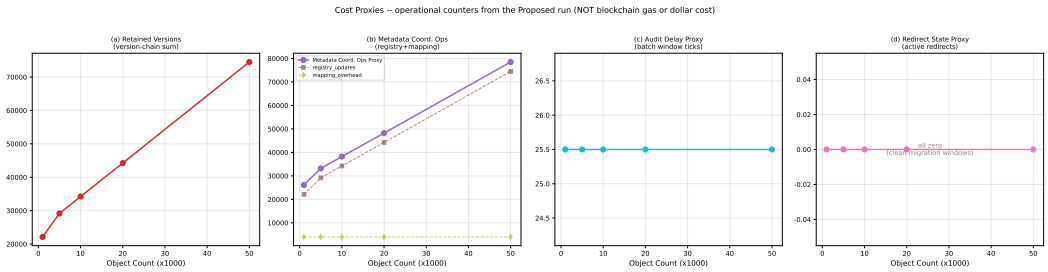


Figure 21. Operational cost proxies vs. object count scale (E8). (**a**) retained versions proxy (red solid line, version-chain sum); (**b**) metadata coordination ops proxy (purple solid line)—this is the aggregate counter registry_updates + mapping_overhead from the proposed run; its two component counters are plotted as references in the same panel: registry_updates (brown dashed line, squares) and mapping_overhead (yellow-green dashed line, diamonds); (**c**) audit delay proxy (cyan solid line, average batch-window staleness in ticks); (**d**) redirect state proxy (pink solid line, active redirects). All four cost proxies are operational counters from the proposed run, not blockchain gas or dollar costs.

Figure 22 presents the break-even region classification. A paper-local operational criterion classifies each scale point based on three conditions: (i) latency reduction below 5%, (ii) hot-tier hit gain below 5 percentage points, and (iii) at least one cost proxy still growing. Scale points where all three conditions hold are marked as diminishing return; the rest are effective benefit. Under this criterion, 1 k–20 k fall in the effective benefit region, while 50 k falls in the diminishing return region.

The figure uses light-green shading for the effective benefit region and light-red shading for the diminishing return region. Only the two core benefit curves are shown; the break-even criterion is an operational heuristic specific to the current simulation setting, not a universal economic threshold.

The narrowing at 50 k does not mean the coordination mechanism is invalid—it means that under fixed proportional hot-tier share, the tier capacity itself becomes the bottleneck. When less than 3% of the object population fits in the hot tier, even perfect placement coordination cannot serve a meaningful share of queries at low latency. Recovering single-digit-millisecond benefit at this scale would require either increasing the hot-tier capacity ratio or adopting a hierarchical tiering strategy. This finding directly addresses the scalability concern and provides a quantitative basis for operational capacity planning.

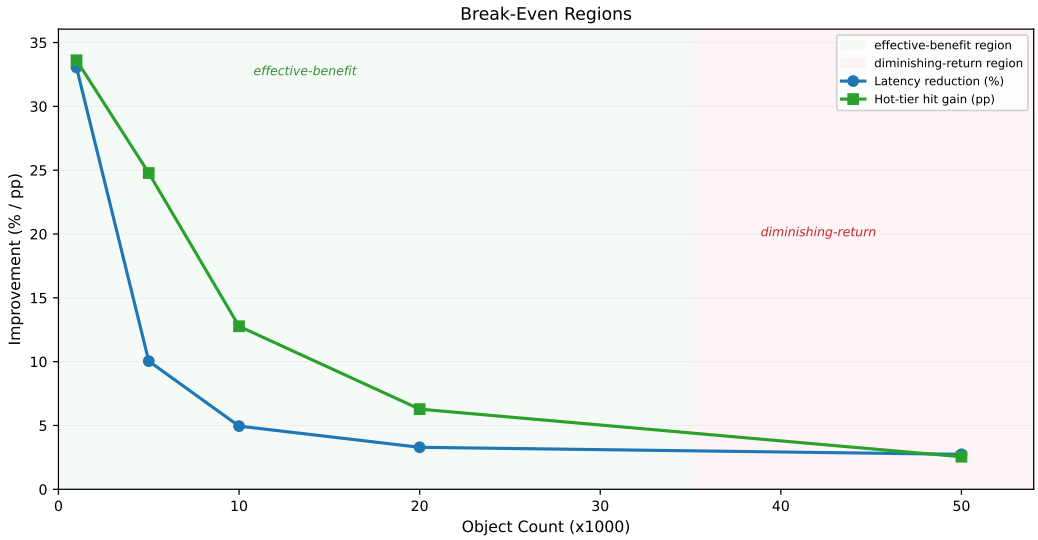


Figure 22. Break-even regions (E8).

5.10. Sensitivity Sweeps and Seed Repeat Stability

Question: how sensitive are the results to heat model parameters, and are the directional findings reproducible across independent random seeds?

Sensitivity sweeps over the heat model weights (α, β, γ) and the decay parameter λ confirm that recency-heavy configurations perform best in the current simulation setting. The batch window trade-off has been discussed in Section 5.2; threshold sensitivity shows stable behavior within $\pm 20\%$ of the default cutoffs.

Seed-repeat results across five independent random seeds show tight confidence intervals: average latency of 7.93 ± 0.08 ms (hotspot), 8.65 ± 0.23 ms (drift), and 9.70 ± 0.09 ms (burst), with hot-tier hit ratios of 0.72 ± 0.01, 0.49 ± 0.02, and 0.55 ± 0.01, respectively. These narrow bands confirm that the directional findings are reproducible and not artifacts of a single random initialization. Table 10 provides the full summary.

Table 10. Multi-seed statistics across five independent random seeds. Values are reported as mean $\pm$ standard deviation.

Scenario	Avg Latency (ms)	P95 Latency (ms)	Hot-Tier Hit Ratio	Throughput
Fixed hotspot	7.93 ± 0.08	10.0 ± 0.0	0.72 ± 0.01	50.0 ± 0.0
Hotspot drift	8.65 ± 0.23	10.0 ± 0.0	0.49 ± 0.02	50.0 ± 0.0
Burst query	9.70 ± 0.09	100.0 ± 0.0	0.55 ± 0.01	30.0 ± 0.0

Component-level ablation note.

A separate ablation of the heat model components (prefetch, adaptive weights, query pattern score $Q_i(t)$, dynamic thresholding) was also conducted. Removing the prefetch module caused the clearest degradation; disabling $Q_i(t)$ slightly improved hit ratio under the burst workload due to transient over-promotion—a result that confirms $Q_i(t)$ is a tunable component whose weight $\gamma(t)$ should be reduced under highly transient workloads, not a design flaw. This component-level analysis is complementary to the coordination-level ablation in Section 5.6 and does not alter its conclusions.

Transition to discussion.

The results above demonstrate that trusted metadata coordination can be integrated into tiered off-chain storage with measurable benefits (latency reduction, commit frequency reduction, sub-millisecond local data plane recovery) and identifiable costs (mapping freshness delay, narrowing advantage at large scale, chain-delay-dominated audit-visible recov-

ery). The coordination mechanism's three components—batching, redirect, and rollback—are shown to be individually necessary and jointly sufficient for the observed gains. The next section discusses the implications and limitations of these findings in the broader context of database systems and data protection.

6. Discussion

The experimental results in Section 5 provide quantitative evidence across eight targeted experiment groups (E1–E8) and a seed-repeat stability test. This section interprets their implications for the paper's central thesis—that trusted metadata coordination should be part of the tiered off-chain data management path rather than an external integrity layer—and identifies the boundaries of the current evaluation. Sections 6.1–6.4 discuss core mechanism findings; Section 6.5 addresses asymmetric write latency and tail latency interpretation; Sections 6.6 and 6.7 discuss deployment considerations, scalability boundaries, and security scope; and Section 6.8 summarizes practical implications and remaining limitations.

To prevent the discussion below from reading as a sequence of independent observations, we make the underlying narrative explicit up front. The argument develops in three nested layers. Layer 1 (core mechanism, Sections 6.1–6.4) establishes that metadata coordination is a placement prerequisite, that batching delivers a tunable 50× commit-overhead reduction, that scalability narrows under a fixed hot-tier share, and that recovery has two distinct timing planes (local vs. audit-visible). Layer 2 (stress and distribution, Section 6.5) asks whether these properties survive asymmetric write costs and variable offered load; they do, but only in the body of the latency distribution—tail percentiles are capped by the cold-tier floor. Layer 3 (deployment, scope, and limits, Sections 6.6–6.8) asks how much of the above survives once the simulator is replaced by a real chain, a real workload, and real hot-tier economics, and identifies which limitations are engineering gaps versus design scope boundaries. The thread that connects all three layers is the same: the coordination mechanism guarantees integrity, consistency, and recoverability across all regimes, but its visible latency advantage is bounded above by the hot-tier capacity and below by the chain finality delay, both of which are external to the coordination workflow itself.

6.1. Metadata Coordination as a Data Management Mechanism

A natural question is whether trusted metadata coordination merely adds overhead to a tiered storage system in exchange for integrity properties that could be provided externally. The E1 variant comparison (Section 5.2) offers a direct answer. The "no mapping optimization" variant uses the same adaptive placement engine as the proposed method but lacks coordinated mapping evolution; its average latency is 14.93 ms, compared with 10.75 ms for the mapping-aware variants. The 28% gap is not caused by algorithmic differences in placement—the heat model, weights, and thresholds are identical—but by the absence of a confirmed mapping cycle that prevents stale pointer conflicts. Without mapping coordination, the dynamic baseline occasionally serves queries from the wrong tier because a promotion has not been registered, negating the benefit of having moved the object in the first place.

This observation reframes the role of metadata coordination: it is not an integrity add-on that taxes performance for auditability [11,13]; it is a prerequisite for the placement engine to realize its intended benefit. The coupling is asymmetric—placement can proceed without mapping coordination (albeit at reduced effectiveness), but mapping coordination without adaptive placement has no placement decisions to coordinate. The framework's design reflects this asymmetry by making placement the trigger and mapping evolution the confirming step.

The fine-grained ablation (E5, Section 5.6) strengthens this argument by demonstrating that the three coordination components—batching, redirect, and rollback—are individually necessary and jointly sufficient. Disabling redirect alone produces 1866 stale reads and raises Phase 1 latency from 34.38 ms to 45.08 ms; disabling rollback alone leaves 80 objects permanently stuck in an inconsistent state. The coordination mechanism is therefore not a monolithic add-on whose value is hard to attribute but a decomposable set of complementary safeguards, each with a quantifiable contribution.

6.2. Batch Coordination: Gain, Cost, and Deployment Tuning

Batching reduces on-chain commits from 4000 to 80 under the default configuration ($W_b = 50$), a 50× reduction. The batch window sweep (Figure 4) shows that this reduction saturates beyond $W_b \approx 100$, while the mapping freshness delay grows linearly. Two practical implications follow.

First, the overhead reduction is real but bounded. For W_b values above 100, further batching provides diminishing commit count savings but continues to increase the staleness window. System operators should therefore choose W_b based on their tolerable staleness rather than maximize batching indiscriminately.

Second, the per-batch commit latency (0.066–0.070 ms) is higher than the per-single-commit latency (0.004 ms) because each batch aggregates more entries and computes a composite hash—consistent with the per-batch overhead profile reported in batched metadata systems [16,19]. These figures are simulated registry write costs inside the simulator (Section 4.1); they do not represent measured on-chain transaction latencies. The E5 ablation (Section 5.6) further shows that setting $W_b = 1$ (no batching) inflates the commit count from 120 to 6000 in Phase 3 while collapsing the average batch size from 50 to 1, confirming that the 50× overhead reduction is robust across experimental conditions. This micro-level cost does not translate into higher end-to-end access latency because batches are committed asynchronously and queries are served from the stable mapping pointer until the batch is finalized.

The batch window sweep above uses a simulated commit cost of 0.07 ms. The default $W_b = 50$ should therefore be interpreted as a simulation setting rather than a deployment recommendation. In a real permissioned chain where finality delay rises to 0.5–5 s, W_b should be selected from the measured finality latency distribution of the target chain and the application-specific tolerance for audit-visible delay and mapping freshness.

A practical constraint is:

$$W_b \cdot \delta_{\text{tick}} + Q_p(\Delta_{\text{chain}}) \;\leq\; D_{\text{audit}}^{\max}, \tag{9}$$

where $Q_p(\Delta_{\text{chain}})$ denotes a high-percentile chain finality latency (e.g., the 95th or 99th percentile), $D_{\text{audit}}^{\max}$ is the maximum acceptable audit-visible delay, and δ_{tick} is the real-time duration corresponding to one control tick. If query freshness is more critical than transaction amortization, W_b should be reduced even when batching would further lower the number of commits. Conversely, if audit visibility can tolerate a longer delay, a larger W_b can reduce transaction pressure. The deployment objective thus shifts from simply minimizing commit count to balancing transaction reduction, mapping freshness, and audit-visible latency.

6.3. Scalability Interpretation and Hot-Capacity Bottleneck

The E3 scalability experiment shows that the proposed method's latency advantage narrows from 29% at 1 k objects to 2% at 50 k objects. This narrowing is structural, not pathological: at larger scales, the ratio of hot-tier capacity to total objects decreases (all tiers scale proportionally, but the hot-tier share is fixed), so fewer queries can benefit from

promotion regardless of coordination quality. The proposed method does not introduce disproportionate overhead—its latency curve tracks the baseline's slope rather than diverging from it.

For the database system audience, the relevant message is that metadata coordination scales linearly with the managed population [15] (Section 3.6, complexity table), and the observable performance advantage is capped by the tier capacity fraction, not by the coordination mechanism itself.

Benefit–complexity break-even.

The E8 analysis (Section 5.9) quantifies this observation. At 1 k objects, the latency reduction is 33% and the hot-tier hit gain is 34 percentage points; at 50 k, these drop to 2.7% and 2.6 percentage points, respectively. Meanwhile, the retained versions proxy and metadata coordination ops proxy continue to grow approximately linearly with object count, while audit delay proxy and redirect state proxy remain constant (25.5 ticks and 0).

A paper-local operational criterion—requiring latency reduction $\geq 5\%$, hit gain above 5 percentage points, and at least one cost proxy to not still be growing—classifies 1 k–20 k as effective benefit and 50 k as diminishing return. This classification is a configuration-dependent operational interpretation under the current simulation setting, not a universal economic threshold. It depends on the fixed proportional hot-tier share (2% at 50 k); a deployment with a larger hot-tier ratio or hierarchical tiering would shift the boundary upward. The criterion is nonetheless useful as a capacity-planning indicator: it tells an operator that beyond a given scale, increasing the coordination effort yields negligible query-level improvement unless the tier architecture itself is adjusted.

The analysis above identifies the fixed hot-tier fraction as the structural bottleneck at scale. The 50 k-object result should therefore be interpreted as a scalability boundary under a fixed hot-tier ratio, not as evidence that metadata coordination becomes useless at large scale. The bottleneck arises because once the hot tier becomes saturated, correct metadata coordination can no longer compensate for insufficient physical hot capacity.

Beyond simply increasing hot capacity, this boundary could be mitigated by several system-level placement extensions—orthogonal to the trusted metadata coordination mechanism proposed in this paper:

- Proactive preheating based on short-term access prediction: Lightweight time-series or sliding window predictors can anticipate imminent demand surges and pre-promote objects before the heat score crosses the threshold.
- Prefetching: When an object in the warm or cold tier is accessed, semantically or temporally related objects can be co-promoted, improving the hit ratio for correlated access patterns.
- Relationship-aware or content-related collaborative placement: Objects that are frequently co-accessed (e.g., sensor streams from the same IoT device cluster) can be grouped and placed as a unit, reducing per-object promotion overhead.
- Adaptive hot threshold control using lightweight online feedback or reinforcement-style tuning: The promotion threshold $T_h(t)$ can be adjusted by a closed-loop controller that observes short-term hit ratio feedback, replacing the occupancy-based rule with a goal-driven policy.
- Hierarchical hot tier or warm prefetch buffer: Inserting a small "warm-prefetch" layer between hot and warm tiers creates a staging area for candidate objects, allowing speculative promotion without consuming scarce hot-tier capacity.

These techniques are orthogonal placement extensions that can improve the quality of migration decisions, while the proposed redirect, batching, and rollback mechanisms ensure that the resulting migrations remain auditable and recoverable. The coordination plane

provides the verified mapping lifecycle (write–verify–commit–redirect–retire), regardless of how the placement decision is made; adopting smarter placement heuristics would change the trigger signal but not the coordination protocol itself.

Stated more bluntly, the E3 narrowing and the prefetching/RL/collaborative placement extensions above together imply a bottleneck shift as the managed population grows. In the effective benefit region (1 k–20 k objects), the dominant performance lever is the coordination mechanism itself: batched commitment, redirect, and rollback together explain the 28–33% latency reduction and the 3.4× hot-tier hit gain because mapping coordination is what stops the dynamic baseline from mis-routing freshly migrated objects. Beyond the diminishing return threshold (here, ≥50 k objects under a fixed proportional hot-tier share), the dominant lever becomes the placement strategy that decides which objects to migrate and when: once less than ∼3% of objects fit in the hot tier, even a perfectly coordinated migration plan cannot serve a meaningful share of queries at low latency, and the next marginal improvement has to come from a smarter predictive or collaborative placement algorithm (or from adjusting the tier architecture itself) rather than from further optimization of the mapping pipeline. We therefore frame the proposed framework as a coordination layer that is necessary but not sufficient at very large scale: necessary because it is the only component that maintains auditability and rollback safety under dynamic migration, but not sufficient on its own to deliver an unbounded latency advantage as hot-tier pressure grows.

6.4. Recovery Latency Evidence and Audit-Visible Recovery

The E4 experiment (Section 5.5) differentiates recovery cost across six fault types. The worst-case single-trial service interruption is 0.107 ms (verification failure), and the average interruption for all fault types is below 0.04 ms. Two aspects of this evidence are worth emphasizing.

Scope of the "sub-millisecond" claim.

The sub-millisecond interruption figures describe local data plane recovery: the time from fault occurrence to restoration of a stable mapping that the query path can use. They do not include the time required for the corrected state to become audit-visible on the blockchain. The E6 chain delay sensitivity analysis (Section 5.7) makes this distinction empirically explicit. Under assumed chain finality delays of 0.5 s, 2 s, and 5 s, local data plane recovery remains unchanged (∼0.006–0.056 ms across all fault types), while audit-visible recovery is dominated by the chain delay (e.g., ∼2000 ms at $\Delta_{\text{chain}} = 2$ s). The two recovery planes are therefore:

- Data plane recovery (query-visible): The effective mapping pointer is restored locally, and subsequent queries are served from the stable tier. This path does not involve any on-chain synchronous verification; it completes in sub-millisecond time regardless of chain latency.
- Audit trail recovery (audit-visible): The rollback event is recorded on chain through the next batched commit, finalizing the audit trail. The total audit-visible recovery time is approximately 0.04 ms (local) + Δ_{chain}, where $\Delta_{\text{chain}} \in \{0.5,\ 2,\ 5\}$ s for typical consortium chains [21,22].

This decomposition clarifies an important distinction: the "sub-millisecond recovery" reported throughout this paper is not misleading, but it applies specifically to the local data plane. Any deployment on a real permissioned chain must set its recovery time objective in terms of audit-visible recovery, which is orders of magnitude slower and dominated by the consensus protocol, not by the coordination mechanism.

Second, the differentiation across fault types is itself an informative result. Target write failures are cheap because no metadata has been mutated; verification failures are the most expensive because the system must detect a digest mismatch and undo a tentative tuple. This cost structure aligns with the five-stage workflow design (Section 3.5), where each successive stage involves more metadata mutation and therefore more rollback work. The monotonic relationship between stage depth and recovery cost provides indirect validation that the rollback semantics operate as designed.

6.5. Asymmetric Write Latency and Tail Latency Interpretation

The baseline E2 configuration showed stable latency and hit ratio across read/write ratios from 90/10 to 30/70. This stability was partly an artifact of symmetric per-tier latency parameters. The E2 extension (Section 5.3) addresses this by introducing 2× and 5× write latency multipliers.

Under the 5× multiplier at 30/70, the proposed method reaches 42.17 ms, while the baseline reaches 58.62 ms—a 28% latency reduction, which essentially preserves the symmetric case advantage. This result confirms two properties. First, the metadata coordination mechanism (batching, redirect, rollback) is write-cost-agnostic: commit count and average batch size remain unchanged regardless of the write multiplier. Second, the relative benefit of coordination persists even when per-operation write cost dominates absolute latency: the coordination path is the same for reads and writes, so its overhead does not scale with the write penalty. At 5× with high write ratios, P99 rises to 500 ms (the cold-tier write penalty), but the proposed method retains a lower average and a higher hot-tier hit ratio.

What the asymmetric latency experiment does not test is load-dependent write queueing, storage backend write amplification, or journal flush cost—factors that would further increase write latency in a real storage engine. The conclusion is therefore limited to the coordination mechanism's resilience to per-operation write cost asymmetry; absolute latency under a full deployment stack remains to be measured.

The asymmetric latency findings above characterize the mean-level behavior; the remainder of this subsection extends the analysis to the distributional level under variable load.

The baseline evaluation (Section 5.1) showed that P95 is 100 ms for every method that includes a cold tier, making it a structural ceiling rather than a differentiator. The E7 variable-load analysis (Section 5.8) reveals what aggregate P95 conceals: the body of the distribution differs substantially between methods.

At every tested load level (10–100 requests/tick), the proposed method's interquartile range sits lower than the baseline's in the grouped boxplot (Figure 16), and its CDF rises faster below 20 ms (Figure 17), reflecting the higher hot-tier hit ratio (e.g., 0.44 vs. 0.25 at 100 req/tick). The exceedance ratio (fraction of requests exceeding 50 ms) and the request backlog proxy both increase monotonically with load for both methods, but the proposed method consistently maintains lower values.

These results matter for two reasons. First, they show that average latency improvement is not merely a mean shift but reflects a genuine redistribution of queries toward lower-latency tiers—a property that boxplot and CDF visualizations capture but aggregate metrics do not. Second, the near-identical P95 and P99 across methods (both capped by the cold-tier floor) warns that the tail percentile metrics in a tiered simulator should not be interpreted as evidence of tail latency equivalence; the cold-tier ceiling masks real distribution differences.

A remaining limitation of E7 is that the offered load is uniform across ticks; bursty or time-varying load patterns could produce different tail shapes. The exceedance and

backlog metrics introduced in this study are designed to extend naturally to such settings, but this has not yet been tested.

6.6. Deployment Considerations and Scalability Boundaries

Several structural factors affect the framework's viability in a production deployment beyond the scale range tested here.

Metadata version chain growth.

Each migration creates a predecessor-version entry retained for rollback safety. The E8 benefit–complexity analysis shows that the retained-version proxy grows approximately linearly with object count (from ∼22 k at 1 k objects to ∼74 k at 50 k). In a long-running deployment, the cumulative version chain length would also grow with the number of migration epochs. A garbage collection policy that prunes version entries older than a configurable audit horizon would bound this growth, but such a policy has not been implemented or tested in the current simulator.

Registry state expansion.

The complexity analysis in Section 3.6 establishes that per-object registry storage is $O(L_i)$ and per-batch communication cost is $O(b)$. At 50 k objects, the registry holds ∼68 k total updates, which is tractable for an in-memory index but may require paging or partitioning in a persistent on-chain key–value store, especially under Fabric-like endorsement policies [22].

Communication overhead.

Batching reduces per-batch communication to one consensus round per W_b ticks, but each round must propagate the batch root and constituent entries to all endorsing peers. At large scale, the batch payload size grows linearly with the number of migrations per window. If the batch payload exceeds the block size limit of the target chain, the mechanism would need a secondary batching layer or a Merkle-root-only on-chain anchor with off-chain batch storage—an architecture that the current design anticipates (Section 3.5) but does not implement.

Fixed hot-tier share as structural bottleneck.

The narrowing of benefit at 50 k objects (Section 6.3) is driven by the proportional capacity-scaling configuration, where the hot-tier fraction drops below 3% of the total object set. No coordination mechanism can serve a meaningful share of queries at low latency when less than 3% of objects fit in the fast tier. Deployments at this scale would need to increase the hot-tier capacity ratio, adopt hierarchical sub-tiering, or accept that the coordination benefit is primarily an integrity and consistency guarantee rather than a latency advantage.

The break-even classification in Section 5.9 uses a paper-local operational criterion that classifies object count scale points as effective benefit or diminishing return. Three clarifications are important.

First, the criterion is not a universal economic threshold. It depends on specific simulator parameters (tier latencies, hot-tier share, batch window) and the particular benefit/cost proxies chosen. A different deployment configuration could shift the boundary substantially.

Second, falling into the diminishing return region does not mean coordination is invalid—it means the latency reduction payoff has narrowed below a practical significance level. The coordination mechanism's consistency and integrity guarantees (stale-read prevention, rollback safety, audit trail) remain fully active regardless of whether the latency benefit is large or small.

Third, the break-even figure (Figure 22) deliberately avoids presenting a single ratio or threshold as a hard cutoff. The shaded regions are a visual aid for capacity planning, not a claim that 50 k objects is a universal break-even point. The summary report provides the detailed criterion definition for reproducibility.

Deployment-oriented economic cost model.

The break-even analysis in E8 uses internal system proxies (retained versions, metadata operations) rather than real economic costs. To assess whether a 50× transaction reduction translates into genuine economic savings, we provide a deployment-oriented cost model.

The total deployment cost over a planning horizon can be expressed as:

$$C_{\text{total}} = N_{\text{tx}} \cdot c_{\text{tx}} + S_{\text{meta}} \cdot c_{\text{storage}} + E_{\text{chain}} \cdot c_{\text{energy}} + C_{\text{hot}} \tag{10}$$

where N_{tx} is the total number of on-chain transactions over the deployment horizon, c_{tx} is the per-transaction cost (gas fee, endorsement fee, or consortium chain operational fee), S_{meta} is the cumulative metadata and version chain storage consumed, c_{storage} is the unit storage cost (on-chain state rent or off-chain persistent storage cost), E_{chain} is the consensus-layer energy or operational cost (validator infrastructure, network bandwidth), and C_{hot} is the hot-tier capacity cost (hardware or cloud provisioning for the fast-access tier).

Under the simulated configuration, the 50× transaction reduction directly lowers the $N_{\text{tx}} \cdot c_{\text{tx}}$ term. However, batching also increases S_{meta} because each batch retains version chain entries for rollback safety. Whether the net effect is positive depends on the ratio $c_{\text{tx}}/c_{\text{storage}}$, which varies by deployment platform.

The break-even reported in E8 is therefore an operational break-even under the simulated configuration, not a universal economic break-even. We deliberately revise the characterization from "acceptable cost" to "acceptable and measurable operational cost under the simulated configuration" to reflect this scope.

In addition to the absolute cost proxy, the economic usefulness of batching should be judged by the net cost difference between the unbatched and batched configurations:

$$\Delta C = (N_{\text{tx}}^{\text{unbatched}} - N_{\text{tx}}^{\text{batched}})\, c_{\text{tx}} - \Delta S_{\text{meta}}\, c_{\text{storage}} - \Delta E_{\text{chain}}\, c_{\text{energy}} - \Delta C_{\text{hot}} \tag{11}$$

Only when $\Delta C > 0$ does batched commitment provide a net economic benefit under a given deployment configuration. Therefore, the 50× reduction in metadata transactions should not be interpreted as a universal monetary saving; it indicates a reduction in the transaction count component, while the final deployment-level break-even point depends on the target chain's transaction fee, metadata retention policy, storage rent, energy cost, and hot-tier provisioning cost.

6.7. Security and Threat Model Boundaries

The protection goals (PG-1–PG-4) defined in Section 3.6 are validated at the system design level through simulation-based experiments (E4, E5) and structural analysis. The paper does not claim a formal cryptographic proof in the IND-CPA or IND-CCA sense. Specifically:

- Integrity (PG-1) is verified by detecting injected digest mismatches and achieving 100% rollback in E4. This demonstrates that the workflow correctly identifies and recovers from corruption, but does not constitute a provable-security reduction.
- Auditability (PG-2) is verified by confirming that all mapping mutations are captured in the append-only registry. The append-only property is a design invariant, not a formally proved property of a specific smart-contract implementation.
- Consistency (PG-3) is strengthened by the E5 ablation, which shows that redirect eliminates stale reads during the commit window. However, the audit window

isolation property (Section 3.6) has not been stress-tested under Byzantine endorser behavior or network partitions.
- Recovery (PG-4) is quantified through E4 and E5. The 100% rollback success rate holds under the current fault injection model; correlated multi-fault scenarios (e.g., simultaneous commit failure and redirect failure) have not been tested.

A consortium chain deployment would introduce additional attack surfaces (malicious endorsers, smart-contract bugs, network partitions) that are outside the scope of the current simulation. The framework's architecture is designed to be compatible with such deployments (local data plane decoupled from on-chain commit), but this compatibility has not been empirically validated. We view this as the most important direction for future work.

To clarify the security posture of unfinalized mapping states, the following invariants hold by construction:

- Only the latest committed stable mapping is authoritative for query resolution. No query is ever served through a tentative, migrating, or pending tuple.
- Migrating and pending mappings are internal coordination states that are not exposed as authoritative pointers to the data plane query resolver.
- Each redirect decision is bound to a specific (object-id, version, digest, predecessor-state) tuple; a redirect cannot be activated unless all four fields match the committed record.
- Rollback returns exclusively to a verified predecessor stable tuple; there is no path from a failed migration to an arbitrary or attacker-chosen mapping state.
- The audit-visible delay introduced by batching does not imply query-visible inconsistency: during the batch window, the effective pointer remains on the pre-migration stable tuple, so queries observe a consistent (albeit not yet updated) mapping.

These invariants collectively ensure that the coordination mechanism's intermediate states do not enlarge the attack surface for query path manipulation.

The current threat model explicitly excludes Byzantine majority among endorsers, collusive endorser coalitions, and multi-fault adversarial attacks that simultaneously compromise both the data plane and the consensus layer. These attack classes require Byzantine fault-tolerant consensus protocols or multi-party computation techniques that operate below the coordination layer and are outside the scope of the present design.

6.8. Summary of Practical Implications

This subsection consolidates the practical implications of the discussion above and classifies the remaining limitations of the current evaluation.

The following limitations scope the validity of the reported results.

1. **Simulation-driven evaluation.** All experiments run on a discrete-tick event-driven simulator. Real distributed storage [6] introduces network jitter, disk I/O variance, and consensus delay [22] that are abstracted away here. The E6 chain delay sensitivity analysis bridges part of this gap by parametrically varying the assumed finality delay, but it does not replace a deployment-level measurement.
2. **Fixed synthetic latency settings.** Per-tier latencies are constant parameters (1 ms/10 ms/100 ms for hot/warm/cold). Real storage tiers exhibit load-dependent and time-varying latency profiles. The E2 asymmetric write latency extension (2×/5×) partially relaxes this assumption for writes, but load-dependent variance remains unmodeled.
3. **No consortium chain deployment.** On-chain operations are modeled as fixed-cost simulated calls. Actual smart-contract execution, gas cost variability, and block confirmation delay are not captured. The sub-millisecond recovery times in E4 refer to

local data plane recovery; audit-visible recovery under real chain latency is estimated via E6 but not directly measured.

4. **No real production trace.** Workload patterns (fixed hotspot, drift, burst) are synthetic generators. Validation against real IoT access traces would strengthen the generalizability claim.
5. **Simplified metadata overhead model.** Mapping update latency and lookup overhead are computed as synthetic cost functions. Real registry implementations may introduce index contention, serialization cost, or replication delay [15,43] not modeled here.
6. **No correlated or Byzantine fault scenarios.** The fault injection model in E4 injects one fault type at a time. Correlated multi-fault events (e.g., simultaneous commit and redirect failure) and Byzantine adversary behavior have not been tested.
7. **No version chain garbage collection.** The retained versions proxy grows with object count and migration epochs. A garbage collection or pruning policy is needed for long-running deployments, but is not implemented in the current simulator.

These limitations do not invalidate the design or the directional findings, but they restrict the quantitative results to the simulated setting.

Table 11 classifies each limitation by category (engineering vs. fundamental/design scope) and assesses whether it could change the main conclusions, affect only numerical magnitude, or require a different system design.

Table 11. Classification of limitations and expected impact.

Limitation	Category	Impact on Main Conclusions	Impact Scope
Simulation-driven	Engineering	Real-chain deployment would validate the design rather than change it	May change numerical magnitude
Synthetic latency	Engineering	Load-dependent models can be plugged in	May change numerical magnitude
No real chain	Engineering	Architecture designed for it; validation needed	Does not change main conclusion
No real trace	Engineering	Heat model weights may shift; framework unchanged	Does not change main conclusion
Simplified overhead	Engineering	Real contention may strengthen batching rationale	Does not change main conclusion
No correlated faults	Fundamental	May require different recovery protocol	May require different system design
No GC policy	Engineering	GC policy is additive, not redesign	Does not change main conclusion

The framework's architecture (separating local data plane operations from asynchronous on-chain commits) is designed to accommodate real-chain latency, and the E6 chain delay sensitivity analysis provides a quantitative estimate of the resulting audit-visible recovery gap. However, this accommodation has not yet been empirically verified on a deployed consortium chain, which remains the most important direction for future work.

In summary, six of the seven identified limitations are engineering gaps addressable through implementation and deployment work, while one (correlated/Byzantine faults) is a fundamental design scope boundary that may require protocol-level extensions. The directional findings—28% latency reduction, 50× metadata transaction reduction, 100% rollback success under single-fault injection, and sub-millisecond local data plane recovery—are robust within the simulated setting and provide simulation-backed evidence of the proposed framework's viability. Quantitative magnitudes are expected to shift under real-chain deployment, but the architectural separation of local data plane operations from asynchronous on-chain commits ensures that the core coordination mechanism remains valid.

Expected impact of simulation abstractions on the main conclusions.

Because Section 4.1 abstracts network jitter, gas costs, and load-dependent I/O variance into fixed per-tier and per-commit parameters, it is appropriate to ask, for each main

conclusion of the paper, in which direction (and roughly by how much) the conclusion is expected to move once those abstractions are relaxed. Table 11 already classifies each limitation as engineering or fundamental; here, we state the expected effect on the four headline numbers:

- 28% average latency reduction (Section 5.1). Expected to shrink as Δ_{chain} grows: each migration now carries an additional per-object mapping lag window (Section 5.7) during which queries are still served from the source tier. The relative advantage over the dynamic baseline persists because the baseline incurs the same lag plus the stale pointer penalty.
- 50× commit count reduction (Section 5.2). Expected to be robust: the ratio is determined by W_b and the migration rate, neither of which depends on the simulated commit cost. The absolute cost of each commit, however, moves from ∼0.07 ms to Δ_{chain}, which is already accounted for in the deployment-oriented cost model in Section 6.6.
- 100% rollback success under single-fault injection (Section 5.5). Expected to remain valid in single-fault scenarios because rollback uses the retained predecessor tuple regardless of chain latency. Under correlated or Byzantine faults, the success rate is not expected to remain at 100%; this is acknowledged in the list of limitations and falls under the "fundamental/design-scope" category in Table 11.
- Sub-millisecond local data plane recovery (Section 5.5). Expected to be robust because the local recovery path is by construction decoupled from the on-chain commit (Section 5.7); the audit-visible figure, however, grows from microseconds to seconds and should be reported separately in any deployment that targets regulated IoT verticals.

The pattern across these four items is that abstractions related to chain latency primarily compress the absolute latency advantage and inflate the audit-visible recovery time, while abstractions related to the fault model primarily affect the recovery success rate under correlated/Byzantine scenarios. Neither category invalidates the architectural claim that placement, mapping, redirect, and rollback must be controlled as a single coupled workflow; rather, they refine the regimes in which each measured headline number applies.

The next section summarizes the paper's contributions and outlines directions for future work that would address the limitations identified above.

7. Conclusions

Tiered off-chain storage for IoT faces a coupled problem: adaptive placement, trusted object–location mapping, migration consistency, and failure recovery are interdependent, yet prior work treats them in isolation—tiered storage studies assume metadata consistency is given, while blockchain-metadata studies assume a static storage topology. This paper bridges the gap by embedding scalable and trusted metadata coordination into the tiered data management path itself. The resulting framework bridges traditional trust systems with blockchain networks powered by PoC consensus, and couples heat-driven placement with dynamic on-chain mapping evolution, batched commitment, redirect control, and rollback-safe recovery in a single five-stage workflow where every promotion or demotion propagates through a verifiable metadata cycle before the next scheduling round proceeds.

Eight experiment groups (E1–E8), together with seed-repeat stability tests, provide the quantitative evidence for this coupling. The baseline comparison shows that metadata coordination is not an integrity add-on but a placement prerequisite: the 28% latency reduction and 3.4× hot-tier hit ratio improvement over the dynamic baseline without mapping coordination arise from preventing stale pointer conflicts, not from algorithmic differences in the heat model. Batched commitment cuts on-chain transactions by 50×,

achieving high-throughput mapping operations at the cost of a tunable mapping freshness delay that grows linearly with the batch window. Scalability experiments from 1 k to 50 k managed objects, effectively managing tens of millions of bytes of data (10+ MB scale), confirm that the coordination mechanism does not introduce disproportionate overhead—latency tracks the baseline's slope rather than diverging. Fault injection across six failure types achieves 100% rollback success with sub-millisecond local data plane service interruption (worst-case 0.107 ms); audit-visible recovery, which requires on-chain finality, is estimated via chain delay sensitivity analysis to reach 500 ms–5 s depending on the assumed consensus latency. The per-fault-type cost structure aligns with the workflow's stage depth design, providing measurable recovery-safe evidence rather than a binary pass/fail claim.

The extended evaluation strengthens these findings along five dimensions. First, fine-grained coordination ablation (E5) decomposes the mechanism into individually necessary components—batching controls commit overhead (50× reduction), redirect eliminates stale reads during the commit window, and rollback guarantees recovery safety—each with a quantifiable contribution. Second, asymmetric write latency stress (2×/5×, E2 extension) confirms that the coordination mechanism preserves its relative advantage when writes are inherently more expensive than reads. Third, variable-load tail latency analysis (E7) reveals that the proposed method concentrates a larger fraction of requests in the low-latency region, a distributional difference that aggregate P95 metrics—capped by the cold-tier floor—cannot capture. Fourth, chain delay sensitivity analysis (E6) makes explicit the distinction between sub-millisecond local recovery and seconds-scale audit-visible recovery, providing a quantitative estimate of the deployment gap. Fifth, benefit–complexity break-even analysis (E8) classifies 1–20 k objects as effective benefit and 50 k as diminishing return under fixed hot-tier share, offering a paper-local operational criterion for capacity planning.

Several directions remain open. First, deploying the framework on a real consortium chain testbed (e.g., Hyperledger Fabric or Quorum) with physical tiered storage would test whether the local rollback path retains sub-millisecond interruption when on-chain finality takes seconds—the most important next-step validation. Second, trace-driven evaluation using production IoT access logs would validate the heat model and batch window tuning under non-synthetic workloads. Third, the current metadata protection scope—integrity, auditable evolution, migration consistency, rollback safety—could be extended toward a formal security analysis with provable security reductions, as well as richer models that incorporate access control policies or cross-tenant isolation. Fourth, a version chain garbage collection policy and registry-state compaction mechanism are needed for long-running deployments where cumulative metadata growth may exceed practical storage bounds. Fifth, correlated and Byzantine fault scenarios (e.g., simultaneous commit and redirect failure, malicious endorser behavior) should be tested to establish the coordination mechanism's robustness boundary beyond the single-fault-type injection model used in this work.

Author Contributions: Conceptualization, W.Y., W.W. and M.H.; methodology, W.Y., W.W. and M.H.; software, W.Y. and K.H.; validation, W.W. and L.L.; formal analysis, M.Y.; investigation, K.H. and Z.Y.; data curation, Z.Y.; writing—original draft preparation, W.Y. and W.W.; writing—review and editing, M.Y., W.X., L.L. and M.H.; visualization, K.H. and Z.Y.; supervision, M.H. and L.L.; project administration, M.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China (Grant No. 2023YFB2704400).

Data Availability Statement: The simulation source code and experimental data are available from the authors upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Uddin, M.A.; Stranieri, A.; Gondal, I.; Balasubramanian, V. A Survey on the Adoption of Blockchain in IoT: Challenges and Solutions. *Blockchain Res. Appl.* **2021**, *2*, 100006. [CrossRef]
2. Benet, J. IPFS—Content Addressed, Versioned, P2P File System. *arXiv* **2014**, arXiv:1407.3561. [CrossRef]
3. Jayabalan, J.; Jeyanthi, N. Scalable Blockchain Model Using Off-Chain IPFS Storage for Healthcare Data Security and Privacy. *J. Parallel Distrib. Comput.* **2022**, *164*, 152–167. [CrossRef]
4. Shi, C.; Xie, H.; Yan, Z.; Yang, L.T. A Survey on Off-chain Technologies. *ACM Comput. Surv.* **2026**, *58*, 181:1–181:35. [CrossRef]
5. Niu, J.; Xu, J.; Xie, L. Hybrid Storage Systems: A Survey of Architectures and Algorithms. *IEEE Access* **2018**, *6*, 13385–13406. [CrossRef]
6. Weil, S.A.; Brandt, S.A.; Miller, E.L.; Long, D.D.E.; Maltzahn, C. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI '06)*; USENIX Association: Berkeley, CA, USA, 2006.
7. Yolchuyev, A.; Levendovszky, J. Data Chunks Placement Optimization for Hybrid Storage Systems. *Future Internet* **2021**, *13*, 181. [CrossRef]
8. Liu, M.; Li, P.; Liu, S. Collaborative Storage for Tiered Cloud and Edge: A Perspective of Optimizing Cost and Latency. *IEEE Trans. Mob. Comput.* **2024**, *23*, 10885–10902. [CrossRef]
9. Li, H.; Sun, M.; Xia, F.; Xu, X.; Bilal, M. A Survey of Edge Caching: Key Issues and Challenges. *Tsinghua Sci. Technol.* **2024**, *29*, 818–842. [CrossRef]
10. Irie, R.; Murata, S.; Hsu, Y.; Matsuoka, M. A Novel Automated Tiered Storage Architecture for Achieving Both Cost Saving and QoE. In *Proceedings of the 2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2), Paris, France, 19–22 November 2018*; IEEE: Piscataway, NJ, USA, 2018; pp. 32–40. [CrossRef]
11. Li, J.; Hu, S.; Shi, Y.; Zhang, C. A Blockchain-Based Trustable Framework for IoT Data Storage and Access. In *Proceedings of the Blockchain and Trustworthy Systems (BlockSys 2019)*; Communications in Computer and Information Science; Springer: Singapore, 2020; Volume 1156, pp. 336–349. [CrossRef]
12. Hou, M.; Kang, T.; Guo, L. A Blockchain Based Architecture for IoT Data Sharing Systems. In *Proceedings of the 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*; IEEE: Austin, TX, USA, 2020; pp. 1–6. [CrossRef]
13. Hasan, H.R.; Salah, K.; Yaqoob, I.; Jayaraman, R.; Pesic, S.; Omar, M. Trustworthy IoT Data Streaming Using Blockchain and IPFS. *IEEE Access* **2022**, *10*, 17707–17721. [CrossRef]
14. Shen, B.; Guo, J.; Yang, Y. MedChain: Efficient Healthcare Data Sharing via Blockchain. *Appl. Sci.* **2019**, *9*, 1207. [CrossRef]
15. Singh, H.J.; Bawa, S. Scalable Metadata Management Techniques for Ultra-Large Distributed Storage Systems—A Systematic Review. *ACM Comput. Surv.* **2018**, *51*, 82:1–82:35. [CrossRef]
16. Zheng, Q.; Ren, K.; Gibson, G.A. BatchFS: Scaling the File System Control Plane with Client-Funded Metadata Servers. In *Proceedings of the 9th Parallel Data Storage Workshop (PDSW '14)*; IEEE: New Orleans, LA, USA, 2014; pp. 1–6. [CrossRef]
17. Li, Y.; Yu, Y.; Wang, X. Three-tier Storage Framework Based on TBchain and IPFS for Protecting IoT Security and Privacy. *ACM Trans. Internet Technol.* **2023**, *23*, 37:1–37:28. [CrossRef]
18. Zheng, X.; Lu, J.; Sun, S.; Kiritsis, D. Decentralized Industrial IoT Data Management Based on Blockchain and IPFS. In *Proceedings of the Advances in Production Management Systems. Towards Smart and Digital Manufacturing*; IFIP Advances in Information and Communication Technology; Springer: Cham, Switzerland, 2020; Volume 592, pp. 222–229. [CrossRef]
19. Ren, K.; Zheng, Q.; Patil, S.; Gibson, G.A. IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*; IEEE: New Orleans, LA, USA, 2014; pp. 237–248. [CrossRef]
20. Patil, S.V.; Gibson, G.A. GIGA+: Scalable Directories for Shared File Systems. In *Proceedings of the 2nd International Workshop on Petascale Data Storage (PDSW '07)*; ACM: New York, NY, USA, 2007. [CrossRef]
21. Nasir, Q.; Qasse, I.A.; Talib, M.A.; Nassif, A.B. Performance Analysis of Hyperledger Fabric Platforms. *Secur. Commun. Netw.* **2018**, *2018*, 3976093. [CrossRef]
22. Gorenflo, C.; Lee, S.; Golab, L.; Keshav, S. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. *Int. J. Netw. Manag.* **2020**, *30*, e2099. [CrossRef]
23. Shi, R.; Cheng, R.; Han, B.; Cheng, Y.; Chen, S. A Closer Look into IPFS: Accessibility, Content, and Performance. *Proc. ACM Meas. Anal. Comput. Syst.* **2024**, *8*, 1–31. [CrossRef]
24. Xue, H.; Chen, D.; Zhang, N.; Dai, H.; Yu, K. Integration of Blockchain and Edge Computing in Internet of Things: A Survey. *Future Gener. Comput. Syst.* **2023**, *144*, 307–326. [CrossRef]

25. Singh, P.; Masud, M.; Hossain, M.S.; Kaur, A. Cross-domain Secure Data Sharing Using Blockchain for Industrial IoT. *J. Parallel Distrib. Comput.* **2021**, *156*, 176–184. [CrossRef]
26. Pajooh, H.H.; Rashid, M.A.; Alam, F.; Demidenko, S.N. IoT Big Data Provenance Scheme Using Blockchain on Hadoop Ecosystem. *J. Big Data* **2021**, *8*, 114. [CrossRef]
27. Pajooh, H.H.; Rashid, M.; Alam, F.; Demidenko, S. Hyperledger Fabric Blockchain for Securing the Edge Internet of Things. *Sensors* **2021**, *21*, 359. [CrossRef] [PubMed]
28. Xue, L.; Liu, D.; Huang, C.; Shen, X.; Zhuang, W.; Sun, R.; Ying, B. Blockchain-Based Data Sharing With Key Update for Future Networks. *IEEE J. Sel. Areas Commun.* **2022**, *40*, 3437–3451. [CrossRef]
29. Huang, J.; Kong, L.; Chen, G.; Wu, M.; Liu, X.; Zeng, P. Towards Secure Industrial IoT: Blockchain System With Credit-Based Consensus Mechanism. *IEEE Trans. Ind. Inform.* **2019**, *15*, 3680–3689. [CrossRef]
30. Li, C.; Zhang, J.; Yang, X.; Luo, Y. Lightweight Blockchain Consensus Mechanism and Storage Optimization for Resource-Constrained IoT Devices. *Inf. Process. Manag.* **2021**, *58*, 102602. [CrossRef]
31. Manzoor, A.; Braeken, A.; Kanhere, S.S.; Ylianttila, M.; Liyanage, M. Proxy Re-Encryption Enabled Secure and Anonymous IoT Data Sharing Platform Based on Blockchain. *J. Netw. Comput. Appl.* **2021**, *176*, 102917. [CrossRef]
32. Sun, H.; Tan, Y.; Zhu, L.; Zhang, Q.; Ai, S.; Zheng, J. A Blockchain-Based Access Control Protocol for Secure Resource Sharing with Mobile Edge-Cloud Collaboration. *J. Ambient Intell. Humaniz. Comput.* **2023**, *14*, 13661–13672. [CrossRef]
33. Saini, A.; Zhu, Q.; Singh, N.; Xiang, Y.; Gao, L.; Zhang, Y. A Smart-Contract-Based Access Control Framework for Cloud Smart Healthcare System. *IEEE Internet Things J.* **2021**, *8*, 5914–5925. [CrossRef]
34. Sharma, A.; Sarishma; Tomar, R.; Chilamkurti, N.; Kim, B. Blockchain Based Smart Contracts for Internet of Medical Things in e-Healthcare. *Electronics* **2020**, *9*, 1609. [CrossRef]
35. Jamil, F.; Ahmad, S.; Iqbal, N.; Kim, D. Towards a Remote Monitoring of Patient Vital Signs Based on IoT-Based Blockchain Integrity Management Platforms in Smart Hospitals. *Sensors* **2020**, *20*, 2195. [CrossRef] [PubMed]
36. Wang, J.; Han, K.; Alexandridis, A.; Chen, Z.; Zilic, Z.; Pang, Y.; Jeon, G.; Piccialli, F. A Blockchain-Based eHealthcare System Interoperating with WBANs. *Future Gener. Comput. Syst.* **2020**, *110*, 675–685. [CrossRef]
37. Iftekhar, A.; Cui, X.; Tao, Q.; Zheng, C. Hyperledger Fabric Access Control System for Internet of Things Layer in Blockchain-Based Applications. *Entropy* **2021**, *23*, 1054. [CrossRef] [PubMed]
38. Henningsen, S.; Florian, M.; Rust, S.; Scheuermann, B. Mapping the Interplanetary Filesystem. In *Proceedings of the IFIP Networking Conference (Networking)*; IEEE: Paris, France, 2020.
39. Doan, T.V.; Psaras, Y.; Ott, J.; Bajpai, V. Toward Decentralized Cloud Storage With IPFS: Opportunities, Challenges, and Future Considerations. *IEEE Internet Comput.* **2022**, *26*, 7–15. [CrossRef]
40. Gray, J.; Lamport, L. Consensus on Transaction Commit. *ACM Trans. Database Syst.* **2006**, *31*, 133–160. [CrossRef]
41. Lamport, L. The Part-Time Parliament. *ACM Trans. Comput. Syst.* **1998**, *16*, 133–169. [CrossRef]
42. Ongaro, D.; Ousterhout, J. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC '14)*; USENIX Association: Philadelphia, PA, USA, 2014; pp. 305–319.
43. Wu, J.; Liu, P.; Chung, Y. Metadata Partitioning for Large-Scale Distributed Storage Systems. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*; IEEE: New York, NY, USA, 2010; pp. 212–219. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.