World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Check for updates

A WORKLOAD-AWARE FRAMEWORK FOR ADAPTIVE STORAGE OPTIMIZATION: INTEGRATING SPACE–TIME TRADE-OFFS WITH PREDICTIVE MANAGEMENT

Joy Uchechukwu Ukibe [1,*], Chiagozie Excellence Ezeumeanya [1] and Emmanuel Ndunuju Amadiegwu [2]

[1] Department of Computer Science, Faculty of Physical Sciences, Nnamdi Azikiwe University, Awka, Anambra State, Nigeria.
[2] Department of Electronics and Computer Engineering, Faculty of Engineering, Nnamdi Azikiwe University, Awka, Anambra State, Nigeria.
* Corresponding author; Email: joyuchechukwuukibe@gmail.com

ORCID Details
Joy Uchechukwu Ukibe: https://orcid.org/0009-0002-8917-4543
Chiagozie Excellence Ezeumeanya: https://orcid.org/0009-0001-0103-761X
Emmanuel Ndunuju Amadiegwu: https://orcid.org/0009-0003-6523-2208

World Journal of Advanced Research and Reviews, 2026, 32(02), 473–481

Publication history: Received on 20 September 2026; revised on 02 November 2026; accepted on 04 November 2026

Article DOI: https://doi.org/10.30574/wjarr.2026.32.2.2767

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

Storage systems must balance access latency, capacity, throughput, reliability, energy use and operating cost, yet they are commonly configured once and left static while workloads change. This study aimed to extend design-time selection of storage techniques into a workload-aware framework for runtime adaptive storage optimization that integrates classical space–time trade-off reasoning with predictive management. A conceptual analytical design was used: storage optimization techniques and representative systems literature were synthesized in four stages, namely identifying resource and performance trade-offs, deriving workload descriptors, mapping those descriptors to candidate actions and risks, and formulating an adaptive control loop with an evaluation agenda. The resulting framework characterizes workloads by read–write ratio, locality, sequentiality, data temperature, burstiness, latency objectives, durability requirements and resource constraints, and maps them to caching, indexing, compression, replication, erasure coding, log-structured storage and tiered placement. It specifies a composite cost function bounded by hard constraints, a six-stage prediction-and-control loop (telemetry collection, workload-state estimation, short-horizon forecasting, candidate generation, constrained evaluation and gradual actuation), and safeguards that include hysteresis, cooldown periods, action budgets and rollback triggers. A multidimensional evaluation protocol and three testable hypotheses are also defined. No new benchmark data are reported. The study concludes that adaptive storage management should act only when the expected, measurable benefit of a change exceeds its transition cost and remains within explicit reliability and service constraints, and that predictive controllers must be validated against strong static and rule-based baselines.

Keywords: Adaptive storage; Space–time trade-off; Workload characterization; Predictive optimization; Tiered storage; Caching

1 INTRODUCTION

The usefulness of a storage system depends not only on how much data it can retain but also on how predictably and economically it can serve requests. A design that minimizes physical capacity may increase decompression work, metadata lookups or reconstruction time. Conversely, a design optimized for very low latency may reserve large caches, replicate data or keep information on expensive media. The central storage-design problem is therefore a trade-off rather than a single-metric optimization problem.

A companion review framed this issue through the classical space–time trade-off [1]. It rated nine families of storage techniques for space efficiency and access speed, concluded that no single technique maximizes both, and proposed a workload-to-strategy decision matrix based on four parameters: read/write ratio, access pattern, data temperature and availability requirements. That matrix is a design-time aid: it helps an engineer choose a configuration before deployment. The present paper addresses the question it leaves open, namely how a running system should change its configuration when the workload changes. Instead of selecting a storage technique once, a system should periodically reassess workload conditions and adjust only when an expected benefit exceeds the cost and risk of change.

This distinction matters because workloads are rarely stationary. A service may move from read-heavy browsing to write-heavy ingestion, experience a sudden burst, or shift from a small hot set to a broad sequential scan. Static settings may become inefficient under such changes. Yet automatic adaptation can also cause instability, oscillation, excessive migration or unsafe durability decisions if it responds to noisy measurements or optimizes only one metric. An effective adaptive design must therefore combine prediction with constraints, confidence checks and controlled actuation.

1.1 The space–time trade-off as a multi-objective problem

In algorithm analysis, time and space are commonly treated as related resource costs; the same principle appears in storage through caches, indexes, replicas, compressed representations and data structures [2]. Modern systems, however, require a broader objective. Latency and capacity are joined by throughput, tail latency, durability, availability, energy, security overhead and total cost of ownership. Tail latency is especially important at scale because rare slow responses can dominate user-visible performance in large distributed services [3]. Improving one dimension may worsen another, and the magnitude of the effect depends on workload and system architecture.

Consequently, storage optimization should not be framed as finding one universally best technique. It is better represented as constrained multi-objective decision-making: minimize an application-specific cost function while respecting hard limits such as durability policy, capacity, service-level objectives and budget. Some dimensions can be traded; others must remain constraints. For example, a system should not reduce replica count merely to save space if doing so violates its recovery-point or availability requirements.

1.2 Core storage techniques and their costs

Caching retains selected data in a faster tier and can reduce repeated access latency, but it consumes scarce memory and can be ineffective for low-locality or scan-dominated traffic unless the replacement policy adapts to the access pattern [4]. Indexing reduces lookup work by maintaining auxiliary structures, while adding storage overhead and update costs. Compression reduces physical footprint and may lower I/O volume, but introduces encoding or decoding work and can increase CPU pressure. Replication improves availability and may place data closer to users, but multiplies storage consumption and write coordination. Erasure coding typically reduces redundancy overhead relative to full replication but can increase reconstruction work and repair complexity [5]. Log-structured designs can improve write throughput, while compaction and read amplification introduce background work and additional I/O [6, 7]. Tiered storage places data on media according to latency, cost, capacity or temperature requirements, and tier placement driven by changing access behaviour has been examined in adaptive tiering research [8].

These mechanisms interact. Compression may allow more objects to fit in a cache but increase CPU use. Replication may lower read latency but amplify writes and network traffic. Indexes may speed point lookups but be less valuable for long sequential scans. The framework therefore treats techniques as a coordinated set of controls rather than independent switches.

1.3 Automated and learned tuning

Database tuning research has shown that automated methods can search large configuration spaces that are difficult to manage manually. For example, OtterTune uses workload observations and previous tuning experience to recommend database configuration settings [9]. Such work motivates automation but does not imply that every storage decision requires a complex model. Rules, statistical forecasts, contextual bandits or reinforcement learning may be appropriate depending on the action space, feedback delay, cost of mistakes and availability of safe training data.

A key design principle is to separate prediction from policy. A model may forecast that read intensity or hot-set concentration will rise, but the policy must still decide whether a cache increase is feasible, whether the expected benefit is large enough, and whether the action is safe. Prediction is an input to a constrained decision process, not a substitute for operational policy.

1.4 Aim and objectives

The aim of this study is to propose a workload-aware framework for adaptive storage optimization that integrates classical space–time reasoning with predictive management. The specific objectives are to: (i) organize workload signals relevant to storage decisions; (ii) map storage techniques to workload and system conditions; (iii) define an adaptation loop with explicit safeguards; (iv) identify multidimensional evaluation metrics; and (v) establish a research agenda for empirical validation.

1.5 Research questions

- **RQ1:** Which workload characteristics are most useful for choosing among storage optimization techniques?
- **RQ2:** How can predictive adaptation manage the space–time trade-off without violating latency, durability or resource constraints?
- **RQ3:** What evaluation design is needed to establish whether adaptive management improves on static configurations?

2 MATERIALS AND METHODS

2.1 Study design

This study adopted a conceptual analytical design. Its materials were the technique ratings and workload-to-strategy matrix of a companion review [1] and a focused selection of representative systems literature on caching, log-structured storage, erasure coding, tiering, automated tuning and benchmarking [2–11]. The study is not a systematic review and does not claim exhaustive coverage of the literature.

2.2 Framework development procedure

The framework was developed in four stages: (1) identification of the main resource and performance trade-offs associated with each storage technique; (2) derivation of workload descriptors that affect technique suitability; (3) construction of a mapping between workload profiles, candidate actions, expected benefits and risks; and (4) formulation of an adaptive control loop, safeguards and an evaluation agenda. Each stage was checked for internal consistency against the trade-offs identified in the preceding stage.

2.3 Scope of evidence

The framework is a proposed design artifact. No live storage platform was instrumented, no machine-learning model was trained, and no comparative performance experiment was conducted for this paper. All recommendations are therefore hypotheses and design guidance to be tested, not measured improvements. Section 3.3 sets out the empirical protocol through which the framework should be validated.

3 RESULTS AND DISCUSSION

3.1 The workload-aware adaptive storage framework

3.1.1 Workload characterization

A storage controller needs a compact, continuously updated view of workload conditions. The framework recommends measuring a small set of interpretable signals rather than feeding every raw event into a predictor. Windowed summaries should include read–write ratio; sequential versus random access; key or object reuse and locality; hot-set concentration; request burstiness; object size distribution; cache hit and miss rates; read and write amplification where available; queue depth; latency percentiles; storage occupancy; CPU and network utilization; and data temperature. Data temperature represents the frequency and recency of access, not a permanent property of the object. Workload characterization and repeatable workload generation are central to evaluating storage-serving systems [10]. Table 1 groups these signals and states why each matters.

Table 1: Workload signal groups used by the framework

Signal group	Example measures	Why it matters
Operation mix	Read/write ratio; point reads vs scans	Distinguishes read-optimized from write-optimized choices
Locality and reuse	Reuse distance; hot-set concentration; hit rate	Indicates likely value of caching and adaptive replacement
Access shape	Sequentiality; random I/O share; request burstiness	Affects indexing, prefetching and cache pollution risk
Data profile	Object size; age; access frequency; temperature	Supports compression and tier placement decisions
System pressure	p95/p99 latency; queue depth; CPU, memory, I/O and network use	Reveals bottlenecks and whether an action shifts pressure elsewhere
Reliability and policy	Durability target; availability; recovery time; security requirements	Defines hard constraints that optimization must not violate

3.1.2 Mapping workload conditions to candidate techniques

Table 2 presents the decision matrix produced by the mapping stage. Unlike the design-time matrix in [1], which matches whole workload profiles to storage strategies, Table 2 is keyed to conditions observed at runtime and records the main risk of each action, so that it can feed the candidate-generation stage of the control loop (Section 3.1.4). It is a starting point, not a universal rule. A candidate action should be considered only after the system identifies the bottleneck and checks whether the action is compatible with the service's durability, consistency and resource requirements.

Table 2: Decision matrix linking observed workload conditions to candidate actions

Observed condition	Candidate action	Expected benefit	Main cost or risk
Repeated reads to a stable hot set	Increase or reshape cache; evaluate admission or replacement policy	Lower read latency and backend I/O	Memory pressure; scan pollution; stale data if invalidation is weak
Frequent selective lookups	Create or tune an index after workload evidence	Fewer records or pages examined	Index storage, write amplification, build and maintenance cost
Large cold objects; capacity pressure	Compress or move to a lower-cost tier	Lower footprint and storage cost	Decode CPU and latency; migration overhead
Write-heavy ingestion	Evaluate log-structured layout and compaction policy	Higher write throughput under suitable workloads	Read amplification, compaction traffic, temporary space needs
High availability or geographically distributed reads	Adjust replica placement within policy limits	Improved availability or locality	Storage and network cost; coordination and consistency complexity
Capacity-efficient distributed durability required	Evaluate erasure coding for eligible data	Lower redundancy overhead than full replication in suitable settings	Reconstruction and repair costs; small-write penalties
Mixed hot/cold workload across heterogeneous media	Use tiered placement with hysteresis	Match access frequency to media cost and latency	Migration churn; stale estimates; tier saturation
Workload shifts or burst patterns	Forecast short-term workload and stage reversible changes	Earlier response than purely reactive tuning	Forecast error, model drift, control oscillation

3.1.3 Objective function and constraints

For a candidate configuration a, the controller can estimate a composite cost, as shown in Equation (1):

$$J(a) = w_L \cdot L(a) + w_T \cdot T(a) + w_S \cdot S(a) + w_E \cdot E(a) + w_C \cdot C(a) + w_R \cdot R(a) \qquad (1)$$

Here L denotes latency cost (including tail latency where relevant), T denotes throughput penalty, S denotes storage consumption, E denotes energy or compute cost, C denotes monetary operating cost, and R denotes operational risk or instability. The weights w are application-specific and should be documented rather than hidden in the controller. In some deployments, throughput is better treated as a minimum constraint than as a penalty. Likewise, durability, access control, consistency and recovery requirements should be hard constraints, not terms that can be traded away for a small performance gain.

The controller should optimize expected cost over a defined horizon, not just the next measurement interval. It should also account for the cost of change: index build time, data migration, cache warm-up, compaction debt, replica transfer and rollback. A configuration with a marginally better steady-state score may be inferior if the transition consumes substantial I/O or causes service disruption.

3.1.4 Prediction-and-control loop

The proposed architecture has six stages, supported by continuous decision logging (Figure 1). The controller begins with safe, static defaults and uses predictions only where they provide decision value. For uncertain forecasts, it can retain the current configuration or run a low-risk trial. Actions should be rate-limited and reversible wherever possible. The stages operate as follows:

1. **Telemetry collection.** Collect telemetry over fixed windows and remove invalid or anomalous observations.
2. **Workload-state estimation.** Classify the current workload and estimate changes in read/write mix, locality, hot-set size and resource pressure.
3. **Short-horizon forecasting.** Forecast near-term demand with a baseline model; use a more complex model only if it improves out-of-sample decision quality.
4. **Candidate generation.** Generate feasible actions from the decision matrix (Table 2) and reject actions that violate durability, consistency, capacity, security or service-level constraints.
5. **Constrained evaluation.** Estimate benefits, transition cost, uncertainty and worst-case impact using Equation (1), and prefer the smallest action likely to address the diagnosed bottleneck.
6. **Gradual actuation and monitoring.** Apply the action gradually, monitor outcomes against a baseline, and roll back if guardrails are breached.

Throughout the loop, the context, recommendation, decision, outcome and any rollback reason are logged for auditability and future model evaluation.

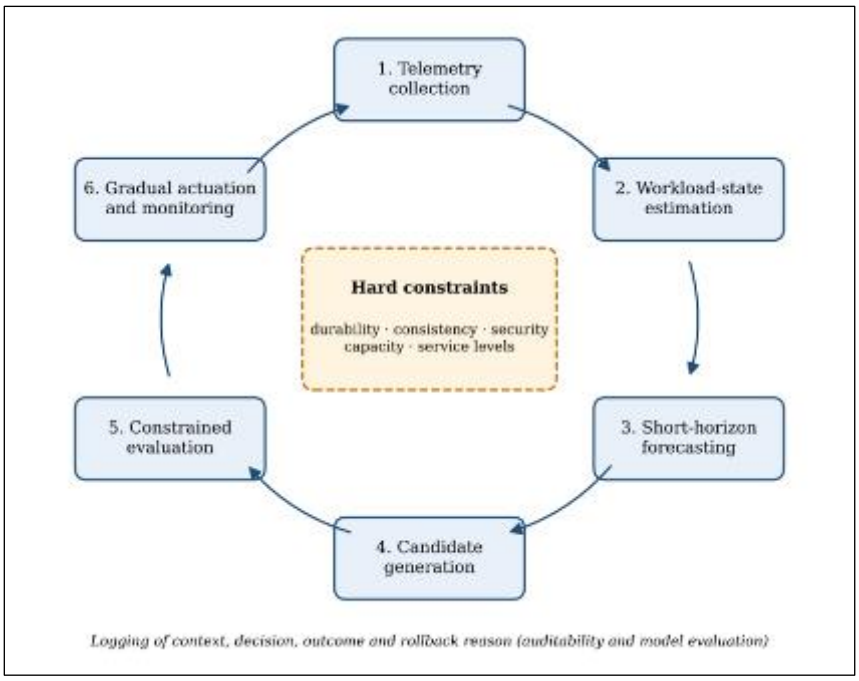


Figure 1: Proposed prediction-and-control loop. Every stage operates within hard constraints; actions taken in stage 6 are rolled back if guardrails are breached, and all decisions are logged

3.1.5 Adaptation safeguards

Adaptive systems can destabilize themselves if they react to short-lived noise. The framework therefore includes: (i) minimum evidence thresholds before action; (ii) hysteresis, so that settings are not repeatedly toggled around a threshold; (iii) cooldown periods after expensive changes; (iv) resource budgets for migration, compaction, index creation and replica movement; (v) canary or staged deployment where possible; (vi) rollback triggers tied to latency and error budgets; and (vii) a fallback static configuration. The controller must not alter durability or security posture autonomously unless the governing policy explicitly permits it.

Prediction drift also requires monitoring. A model trained on weekday traffic may perform poorly during a promotion or outage. The controller should compare forecasts with actual observations, detect sustained error, and revert to a simpler policy when confidence falls below a defined threshold. In high-consequence systems, human approval may remain necessary for actions with broad or irreversible effects.

3.2 Illustrative decision scenarios

3.2.1 Read-intensive service with a concentrated hot set

Suppose a service shows a high proportion of repeated point reads, stable hot-set concentration and rising backend read latency while memory remains available. The framework first checks whether cache misses are a material cause of the latency. If they are, it may recommend increasing the cache budget or changing admission or replacement behaviour. It should not automatically add indexes or replicas without evidence that those changes address the bottleneck. Success would be judged by hit rate together with p95/p99 latency, memory pressure and backend I/O, not hit rate alone. The value of distinguishing hot and warm objects in a tiered architecture is illustrated by Facebook's f4 storage system [11].

3.2.2 Write-heavy ingestion with compaction pressure

For a write-heavy workload, a log-structured design may be suitable, but rising write throughput can be accompanied by read amplification and compaction debt [7]. The controller should monitor pending compaction, write stalls, read latency and temporary disk requirements. A compaction adjustment should be rejected if available space cannot safely accommodate the transition. The example illustrates why an apparent improvement in one metric cannot be evaluated without its downstream costs.

3.2.3 Cold-data archive under capacity pressure

When most objects are rarely accessed and capacity cost dominates, compression or erasure coding may be candidates. The controller must consider object size, expected access frequency, retrieval-time objectives, CPU capacity and recovery requirements. Frequently accessed objects should not be moved to a slower tier merely because their recent

access count dipped during a short quiet period. Hysteresis and longer observation windows reduce such misclassification. Adaptive tiering research similarly examines moving data according to changing access patterns [8].

3.2.4 Sudden workload shift

If a workload changes abruptly from sequential scans to repeated random reads, a predictor may detect the shift earlier than a slow-moving average. Nevertheless, the system should avoid large immediate reconfiguration based on one noisy window. A staged response, adjusting cache admission first, observing results, and only then considering index or tier changes, reduces the risk of making several interacting changes whose effects cannot be distinguished.

3.3 Evaluation framework for future empirical study

A strong evaluation should compare at least three policies: a fixed baseline configuration, a reactive rule-based controller and a predictive controller. The experiment should use the same workload traces and hardware conditions for all policies. Workloads should include read-heavy, write-heavy, mixed, sequential-scan, skewed-locality, bursty and phase-changing patterns. Synthetic workloads can provide control, while public or operational traces can test external validity where permissions and privacy requirements allow. A standardized workload-generation approach such as YCSB can help make comparisons repeatable [10]. Table 3 lists the recommended evaluation dimensions.

Table 3: Recommended evaluation dimensions and measures

Evaluation dimension	Recommended measures	Interpretation
Latency and service quality	Median, p95 and p99 latency; timeout/error rate	Captures both typical and tail behaviour
Capacity efficiency	Bytes stored; metadata/index overhead; compression ratio; redundancy factor	Quantifies the space side of the trade-off
Throughput and I/O	Operations/s; bytes read/written; read/write amplification	Shows whether work is reduced or merely shifted
Resource use	CPU, memory, network; energy where measurable	Detects hidden costs of compression, prediction or migration
Adaptation cost	Migration bytes; index-build time; compaction debt; reconfiguration count	Measures the overhead introduced by changing settings
Robustness	Forecast error; time to recover after a shift; rollback count; constraint violations	Tests safety and stability, not just average performance
Economic impact	Cost per request or workload unit; total cost of ownership	Connects technical metrics to deployment decisions

The experimental protocol should predefine workloads, train/test separation, warm-up periods, measurement windows, repeated runs and statistical summaries. A predictive model must not use future information when constructing features. Results should report uncertainty and include ablation tests to determine whether prediction adds value beyond simple rules. The primary comparison should be against a credible baseline, not an intentionally weak configuration. All code, configuration files, workload generators and analysis scripts should be published where permitted to support reproducibility.

3.3.1 Hypotheses for empirical testing

- **H1:** Under workloads with identifiable phase changes, a predictive controller can reduce tail latency or recovery time relative to a fixed configuration, provided that its forecasts are sufficiently accurate.
- **H2:** Under stable workloads, a complex predictive controller may provide little benefit over a well-tuned static or rule-based policy once model and actuation costs are included.
- **H3:** Safeguards such as hysteresis and action budgets reduce configuration oscillation and migration overhead.

These are testable hypotheses, not findings established by the present conceptual study.

4 DISCUSSION

The proposed framework moves the companion review [1] from design-time selection to runtime management in three ways. First, it replaces fixed workload profiles with an explicit set of signals that can be measured continuously (Table 1) and a matrix keyed to observed conditions and action risks (Table 2), answering RQ1. Second, it separates workload prediction from policy selection and bounds every action with hard constraints and transition costs (Equation 1, Figure 1), answering RQ2; this makes it possible to use simple models when they are sufficient and more complex models only when justified. Third, it specifies a comparative evaluation design with multidimensional measures and explicit hypotheses (Section 3.3), answering RQ3, and treats safety, uncertainty, transition cost and reversibility as central parts of optimization rather than implementation details.

The framework also clarifies why “adaptive” does not automatically mean “better”. If a workload is stable, the cost of continual prediction and reconfiguration may outweigh any gain. If forecasts are unreliable or the system cannot safely reverse an action, adaptation can degrade service. If metrics are poorly chosen, the controller may optimize cache hit rate while worsening tail latency [3], or reduce storage use while increasing retrieval time beyond an application's tolerance. A robust design therefore needs an explicit objective, guardrails and a credible baseline.

Machine learning is most useful when workload behaviour contains patterns that simple rules cannot capture and when enough representative data exist to train and evaluate a model [9]. Even then, prediction should be judged by its downstream decision value, not forecast accuracy alone. A small forecasting improvement is not valuable if it does not change the action selected or improve operational outcomes. This principle argues for decision-focused evaluation and for retaining transparent rules around hard constraints.

4.1 Practical implications

For system designers, the framework recommends starting with telemetry and bottleneck diagnosis before introducing machine learning. For database and storage administrators, it encourages staged, measurable changes rather than broad simultaneous reconfiguration. For researchers, it offers a reproducible evaluation structure that includes transition costs and safety metrics. For resource-constrained institutions, lightweight rule-based adaptation may provide a more practical first step than a complex learning controller.

4.2 Limitations

This paper is conceptual and does not report new empirical benchmarks. The proposed objective function and decision matrix require calibration for each system. The framework simplifies interactions among storage layers and does not specify one universal prediction algorithm. Literature selection is focused rather than systematic, and findings from one storage engine may not transfer directly to another. Future work must implement the controller, test it against strong baselines and report measured performance under realistic workloads.

5 CONCLUSION

Storage optimization is best understood as a workload-dependent, multi-objective control problem. Caching, indexing, compression, replication, erasure coding, log-structured storage and tiering each shift the balance among latency, capacity, throughput, reliability, energy and cost. Because workloads change, static configurations may become inefficient; however, unconstrained automation can introduce instability and operational risk. This paper has proposed a workload-aware framework that combines telemetry, state estimation, short-horizon prediction, constrained action selection, gradual deployment and rollback safeguards.

The main contribution is a structured conceptual bridge between the space–time trade-off and adaptive storage management. It does not claim that a predictive controller has already been implemented or that performance gains have been measured. The next research step is to implement a prototype and compare fixed, rule-based and predictive strategies on controlled and representative workloads, measuring tail latency, storage overhead, I/O amplification, resource consumption, adaptation cost, robustness and economic impact. The guiding principle is that adaptation should be applied only when its expected, measurable benefit exceeds its cost and remains within explicit reliability and service constraints.

Recommendations

- Instrument the storage system and establish a trustworthy static baseline before adding prediction.
- Use workload characteristics and diagnosed bottlenecks to select candidate techniques rather than relying on one-size-fits-all rules.
- Treat durability, security, consistency and recovery requirements as hard constraints.

- Include migration, index-build, compaction and model costs in the optimization objective.
- Use hysteresis, cooldown periods, staged rollout and rollback triggers to prevent unstable adaptation.
- Validate predictive approaches against strong static and rule-based baselines using repeated, reproducible experiments.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Acknowledgments

The authors acknowledge the Department of Computer Science and the Department of Electronics and Computer Engineering, Nnamdi Azikiwe University, Awka, for academic support. This research received no specific grant from any funding agency in the public, commercial or not-for-profit sectors.

Disclosure of conflict of interest

The authors declare no financial or non-financial conflict of interest.

REFERENCES

[1] Ukibe JU, Atuorah N, Ike IF, Akawuku GI, Ezeumeanya CE, Amadiegwu EN. Storage techniques and the space-access time trade-off: a workload-aware review and decision framework. Int J Latest Technol Eng Manag Appl Sci. 2026;15(9):2982–97. doi:10.51583/IJLTEMAS.2026.150900150

[2] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. 4th ed. Cambridge (MA): MIT Press; 2022.

[3] Dean J, Barroso LA. The tail at scale. Commun ACM. 2013;56(2):74–80. doi:10.1145/2408776.2408794

[4] Megiddo N, Modha DS. ARC: a self-tuning, low overhead replacement cache. In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03); San Francisco, CA. Berkeley (CA): USENIX Association; 2003. p. 115–30.

[5] Huang C, Simitci H, Xu Y, Ogus A, Calder B, Gopalan P, et al. Erasure coding in Windows Azure Storage. In: Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC '12); Boston, MA. Berkeley (CA): USENIX Association; 2012. p. 15–26.

[6] Rosenblum M, Ousterhout JK. The design and implementation of a log-structured file system. ACM Trans Comput Syst. 1992;10(1):26–52. doi:10.1145/146941.146943

[7] O'Neil P, Cheng E, Gawlick D, O'Neil E. The log-structured merge-tree (LSM-tree). Acta Inform. 1996;33(4):351–85. doi:10.1007/s002360050048

[8] Pang L, Alazzawe A, Ray M, Kant K, Swift J. Adaptive intelligent tiering for modern storage systems. Perform Eval. 2023;160:102332. doi:10.1016/j.peva.2023.102332

[9] Van Aken D, Pavlo A, Gordon GJ, Zhang B. Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17); Chicago, IL. New York: ACM; 2017. p. 1009–24. doi:10.1145/3035918.3064029

[10] Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R. Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10); Indianapolis, IN. New York: ACM; 2010. p. 143–54. doi:10.1145/1807128.1807152

[11] Muralidhar S, Lloyd W, Roy S, Hill C, Lin E, Liu W, et al. f4: Facebook's warm BLOB storage system. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14); Broomfield, CO. Berkeley (CA): USENIX Association; 2014. p. 383–98.