International Journal of Informatics and Computation (IJICOM)
Vol. 8, No. 1, 2026
ISSN: 2685-8711, E-ISSN: 2714-5263 DOI: 10.35842/ijicom

Enhancing Storage Efficiency in Large-Scale Internet of Things Networks: A Longitudinal Study of Least Recently Used Cache Replacement

Niskarto Zendrato[1], Fahrurrozi Lubis[2], Gideon Tulus Hatta Yuda[3], Anandhini Medianty Nababan[4]

Abstract
The continuous generation of telemetry data by Internet of Things deployments poses critical challenges for edge devices characterized by limited storage and constrained processing power. Frequent access to primary storage media increases latency and bandwidth utilization. While sophisticated predictive models exist, lightweight, low-overhead solutions are prioritized in practical, medium-scale IoT edge environments. This paper evaluates the efficacy of the Least Recently Used cache replacement strategy, serving as a foundational baseline within a distributed IoT context. Empirical validation is essential, as longitudinal performance in real-world distributed settings often diverges from idealized simulations. This investigation, involving 20 environmental sensors linked to five Raspberry Pi edge servers over three months, assessed caching efficacy using request-hit and data-volume-hit ratios. The LRU configuration achieved an average request-hit ratio of 82.73% and a data-volume-hit ratio of 90.90%. These results confirm that lightweight local caching provides a robust, practical storage-efficiency mechanism for medium-scale IoT systems, avoiding the overhead of complex synchronization or computationally intensive models. The findings underscore the utility of simple replacement heuristics in memory-constrained environments requiring minimized computational overhead.

Keywords:
Cache Replacement, Data Caching, Edge Computing, Internet of Things, Storage Efficiency

1. Introduction

The expansion of digital infrastructure exhibits a substantial correlation with the rapid deployment of Internet of Things (IoT) devices. Sensors, actuators, and microcontroller-based nodes now collect physical-world information at high frequency, culminating in the generation of continuous telemetry streams for environmental monitoring and decision support. Although this data density increases monitoring resolution, it may also engender several practical challenges related to storage capacity, communication bandwidth, and response latency in edge environments[1]. Many early IoT implementations were predicated upon centralized cloud platforms for data ingestion and processing[2]. In high-frequency sensing scenarios, however, the transmission of each reading to distant cloud infrastructure potentially augments propagation delay and transmission cost[1]. Edge computing responds to this limitation through the strategic relocation of selected storage and computation tasks in closer proximity to the data source, thereby reducing unnecessary backhaul traffic and improving response time for time-sensitive applications[3].

Corresponding Author: Niskarto Zendrato (niskarto@usu.ac.id)
1 Niskarto Zendrato, Universitas Sumatera Utara, niskarto@usu.ac.id
2 Fahrurrozi Lubis, Universitas Sumatera Utara, fahrurrozi.lubis@usu.ac.id
3 Gideon Tulus Hatta Yuda, Universitas Sumatera Utara, gideonsiahaan86@gmail.com
4 Anandhini Medianty Nababan, Universitas Sumatera Utara, nababan.anandhini@usu.ac.id

Edge computing does not necessarily obviate the storage problem; rather, it shifts its manifestation. Local gateways and single-board computers are frequently predicated upon modest flash-based storage and limited processing resources. When such devices are compelled to undertake the repeated reading and writing of sensor logs, the system may experience input/output delays and accelerated storage-media wear. For this reason, low-overhead data-management mechanisms are deemed imperative in resource-constrained Internet of Things and Named Data Networking (NDN)-oriented environments[4]. Caching constitutes a primary mechanism for the attenuation of repeated access to primary storage. By retaining frequently requested or recently accessed objects in a faster temporary layer, an edge server may serve repeated requests with diminished latency and reduced upstream traffic. The effectiveness of this approach depends on the replacement rule used when cache space becomes full, inasmuch as its criteria dictate which objects persist in availability and which are subjected to eviction[5].

Least Recently Used (LRU) is an established replacement policy based on temporal locality. The underlying assumption posits that recently accessed data exhibits a heightened probability of subsequent retrieval compared to data that has remained unaccessed over an extended duration. This characteristic suggests that LRU may be particularly advantageous for telemetry dashboards, where users frequently query current or recently generated records. It also often serves as a baseline against which more advanced replacement policies are evaluated[6]. Despite its simplicity, Least Recently Used (LRU) still necessitates validation under realistic operating periods. Short experiments may not represent daily access cycles, weekly fluctuations, and changes in user-request behavior. Distributed Internet of Things (IoT) systems may also exhibit spatial traffic differences because each edge server handles a different sensor cluster. Therefore, careful evaluation of local temporal cache stability under operational workloads is warranted in distributed IoT systems[7].

This paper delineates the application of LRU cache replacement within a distributed IoT network, which comprises twenty environmental sensors and five autonomous Raspberry Pi edge servers. The primary objective involves ascertaining whether autonomous local caching mechanisms can maintain elevated storage-efficiency levels throughout a three-month observational duration. Furthermore, this investigation establishes a pragmatic baseline for determining the necessity of more intricate intelligent or cooperative caching mechanisms in comparable medium-scale IoT deployments.

2. Related Works

The domain of cache management has evolved from rudimentary heuristic policies into sophisticated adaptive and learning-based methodologies. In general, replacement algorithms can be grouped into three categories: classical policies, hybrid or content-aware policies, and machine-learning-driven policies[8]. Each category presents a distinct trade-off concerning implementation expenditure, adaptability, and projected hit ratio[9].

Classical cache algorithms constitute the foundation for the majority of replacement strategies. FIFO removes the earliest inserted object without evaluating how often or how recently it has been requested[10]. LFU may retain objects exhibiting high access counts; however, this retention could potentially prolong the presence of outdated popular objects, a phenomenon frequently termed cache pollution[11]. Comparative investigations suggest that algorithmic performance may fluctuate substantially contingent upon traffic characteristics and workload distribution[12], [13].

LRU endeavors to ameliorate certain inherent shortcomings by emphasizing recency, rather than cumulative frequency. Objects that have not been subject to a request for the most protracted duration are consequently evicted first, thereby enabling the cache's rapid

responsiveness during shifts in access patterns; this property appears particularly suitable for numerous sensor applications, although LRU may still exhibit suboptimal performance when sequential scans displace objects that were recently accessed and remain useful [14]. Furthermore, emerging research into information-centric networking has explored application-specific architectures that operate independently of standard node or content properties, providing alternatives for disaster management or overlay caching scenarios [15]. Conversely, specialized studies indicate that while LRU performs robustly in scenarios valuing data freshness, alternative strategies like probabilistic caching may yield superior outcomes in specific IoT topologies by mitigating the redundancy inherent in persistent caching models [16]. Additionally, frequency-based models such as Least Frequently Used attempt to maximize hit rates by retaining highly popular items, though this approach is often computationally demanding because replacement operations cannot be executed in constant time [17]. Such computational overhead often renders frequency-based approaches impractical for line-speed operations in resource-constrained IoT gateways [18]. In contrast to these reactive techniques, newer proactive strategies attempt to calculate content popularity a priori or employ graph neural networks to determine optimal caching locations [19], [20].

To advance beyond single-factor classical policies, several studies have proposed hybrid or content-aware approaches. ARC combines recency and frequency information, while other policies incorporate object size, content freshness, popularity, or network centrality into the eviction decision. Within Internet of Things (IoT) and Named Data Networking (NDN) contexts, cooperative cache pooling and other strategies have been proposed to enhance distributed cache performance [21]. These advanced methodologies aim to optimize cache hit ratios by dynamically adjusting to heterogeneous traffic patterns and varying content demands [22]. By integrating these multidimensional metrics, such frameworks better navigate the trade-offs between computational overhead and storage efficiency [17], [23]. Furthermore, the Adaptive Replacement Cache specifically addresses these trade-offs by utilizing a learning rule to dynamically balance recency and frequency components based on observed workloads. Despite these operational advantages, its adoption remains constrained in certain commercial environments due to proprietary licensing requirements that necessitate formal agreements with the developer. Building upon these foundational insights, researchers have sought to extend the adaptive paradigm through variants like the Adaptive Replacement Cache-Tempora, which attempts to mitigate fixed-license limitations while enhancing temporal precision [24].

Recent scholarship has increasingly investigated intelligent cache replacement. LeCaR, Hawkeye, reinforcement-learned replacement, graph-neural-network caching, and reinforcement-learning edge caching, employ learning mechanisms to predict replacement decisions under complex workloads [25], [26]. Consequently, selecting an optimal cache strategy necessitates balancing algorithmic sophistication against the inherent hardware limitations of edge infrastructure [27]. Moreover, as cache replacement transitions toward distributed environments, researchers must weigh the reduced synchronization overhead of approximate counting methods, such as Count-Min Sketch, against the inherent accuracy loss in frequency tracking [28]. Furthermore, the computational overhead associated with maintaining precise frequency counts in algorithms like LFU remains a significant barrier for resource-constrained nodes, often necessitating trade-offs between implementation complexity and performance efficiency [29]. Additionally, current literature indicates that reinforcement learning-based caching frameworks often struggle with effective coordination in highly dynamic network environments, frequently requiring substantial training data to achieve convergence with LSTM[30].

$$\sqrt{f_t = \sigma\left(W_f\left[\begin{matrix}h_{t-1}\\ x_t\end{matrix}\right] + b_f\right)} \qquad (1)$$

where:

- f_t : forget gate activation at time step t,
- $\sigma(\cdot)$: sigmoid activation function,
- W_f : forget gate weight matrix,
- h_{t-1}: hidden state from the previous time step,
- x_t : input vector at time step t,
- b_f: bias vector,
- $[{}^{h_{t-1}}_{x_t}]$: concatenation of the previous hidden state and the current input.
- i_t : input gate activation at time step t,

In equation (1), the forget gate utilizes a sigmoid activation to delineate the extent to which previous hidden-state information is retained. Although LSTM-based cache replacement has been investigated, these calculations typically necessitate repeated matrix operations, which can increase processing overhead on constrained edge hardware [31].

Similarly, the input gate is formulated to decide which new temporal information the memory network needs to retain:

$$\sqrt{i_t = \sigma\left(W_i \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_i\right)} \qquad (2)$$

The main limitation of neural replacement methods in this context pertains not to theoretical accuracy but to deployment cost. A Raspberry Pi or similar micro-edge server may possess limited thermal headroom and floating-point capacity. Should every telemetry request necessitate additional predictive computation, the cache controller may, in turn, consume a portion of the latency savings it is intended to provide. This observed trade-off thus suggests the necessity for lightweight baselines prior to the adoption of ponderous intelligent cache-management systems[32].

The research gap addressed in this paper is therefore a practical imperative: the characterization of the long-term behavior of simple local Least Recently Used (LRU) caching is deemed essential before engineers introduce neural prediction engines or cluster-wide cache coordination. Distributed caching can improve performance, but it also adds synchronization traffic and administrative complexity[33]. A well-tested LRU baseline appears to be of considerable value for cost-benefit comparison in resource-constrained IoT networks. Consequently, this study quantifies the performance thresholds of heuristic policies to determine whether such lightweight mechanisms provide sufficient utility for modern edge-assisted architectures [34]. By empirically assessing these heuristics against long-term sensor telemetry, we provide a performance ceiling that informs whether the overhead of complex, sequence-aware models—such as LSTMs or Transformers—offers a justifiable improvement in cache hit ratios for specific IoT deployments [35], [36].

3. Proposed Method

The proposed methodology was designed to represent a medium-scale distributed IoT architecture. Rather than relying on a centralized cloud-only arrangement, the experimental design mandated the placement of cache management at the edge layer. Each edge node operated independently, thereby facilitating the evaluation of local cache behavior without the concomitant introduction of inter-server coordination overhead.

3.1. Physical System Architecture

The physical system used a three-tier structure. The first tier consisted of 20 DHT11 temperature and humidity sensors, which were connected to ESP8266 microcontroller interfaces. Each ESP8266 unit digitized environmental readings and transmitted the resulting payloads through the local wireless network. The second tier consisted of five Raspberry Pi mini-servers, serving as data collectors and local proxy-cache nodes. The 20 sensors were divided evenly into five groups; consequently, each Raspberry Pi accommodated four ESP8266 sensor units. The third tier included the client interface and Thingspeak dashboard, where users accessed temperature and humidity information asynchronously.

Fig. 1 presents the complete system layout, including the sensor tier, the Raspberry Pi edge-cache tier, and the monitoring dashboard tier.

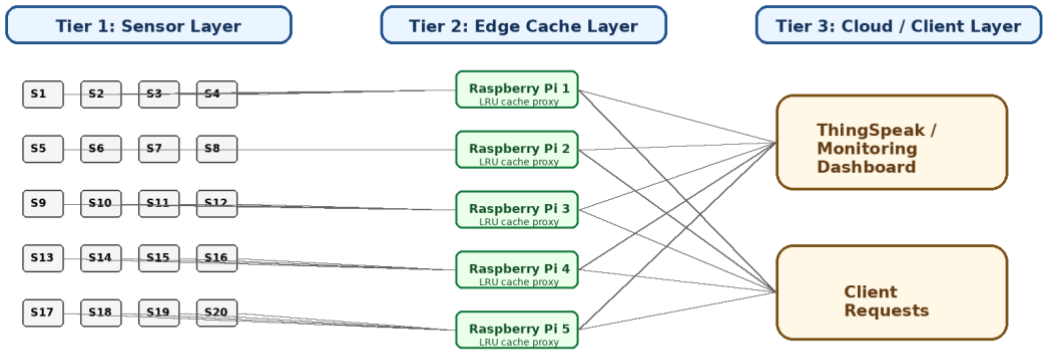

Fig. 1. The three-tier Internet of Things (IoT) physical architecture comprising the 20 DHT11 sensors (Tier One); the five Raspberry Pi edge servers, which function as local cache proxies (Tier Two); and the cloud dashboard (Tier Three).

Table 1. System Architecture and Component Allocation

Tier	Component	Quantity / Configuration	Function in the Study
Tier 1	DHT11 sensors + ESP8266 microcontrollers	20 sensor nodes; 4 sensors assigned to each edge server	Generation of temperature-humidity telemetry and transmission of packets through the local wireless network
Tier 2	Raspberry Pi edge servers	5 independent mini servers; 10 GB cache capacity per server	Reception of sensor traffic, storage of local cache objects, and execution of LRU eviction logic
Tier 3	Client dashboard / Thingspeak interface	Asynchronous user-access layer	Visualization of current and historical telemetry data requested by clients
Cache layer	apt-cacher-ng and LRU daemon	Independent local proxy on every Raspberry Pi	Interception of repeated requests and eviction of the least recently used objects upon storage saturation

3.2. Cache Replacement Logic Model

The implementation of the caching process appears to have been undertaken

independently across each discrete Raspberry Pi unit. The proxy layer utilized apt-cacher-ng, the reconfiguration of which from its initial package-caching function facilitated the proficient handling of sensor-data request traffic. The eviction rule, which is predicated upon temporal locality, governs cache object removal. Let Cmax(j) denote the maximum cache capacity of edge server j. In this experiment, each Raspberry Pi was allocated 10 GB of cache space, thereby endowing the five-node cluster with an aggregate temporary storage capacity of 50 GB.

A background script named lru_cache_algorithm.sh effectuated continuous monitoring of the cache directory. The cached-object set is defined as $S = \{O1, O2, \ldots, On\}$. For every object Oi, the system records a last-access timestamp T(Oi), which is updated whenever the object is read or written. Upon the receipt of each incoming request, the system compares the utilized cache capacity, Cused, against the predetermined maximum allowable capacity, Cmax. When Cused exceeds Cmax, indicating a saturated condition within the cache, the eviction process commences with the identification of the object with the oldest access timestamp.

Fig. 2 delineates the decision sequence employed by the LRU eviction mechanism when the temporary storage layer attains its capacity limit.

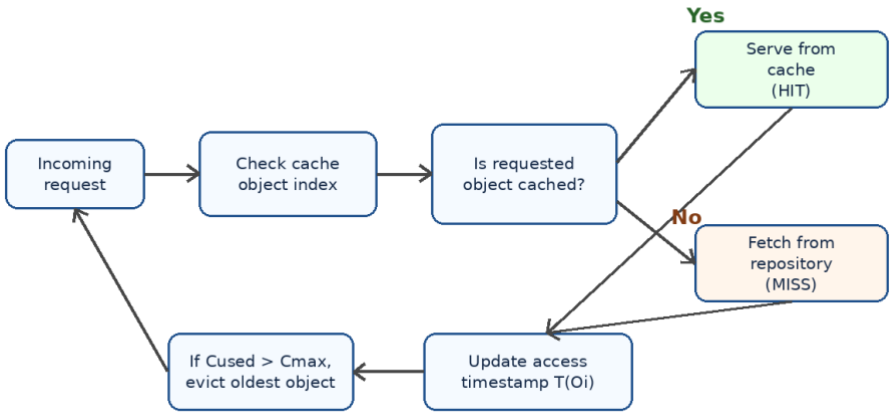

Fig. 2. Flowchart illustrating the LRU cache eviction logic when the temporary storage capacity ($C_{used} > C_{max}$) is saturated.

The object selected for removal is formally identified by the minimization operation in (3).

$$\sqrt{O_{evict} = \arg\min \text{Oi} \in \text{S}\, T(O_i)} \qquad (3)$$

where:

- O_{evict} : object selected for eviction,
- S : set of objects currently stored in the cache,
- O_i : the i-th cached object,
- $T(O_i)$: timestamp of the most recent access to object O_i,
- $\arg min$: returns the object with the smallest (oldest) access timestamp.

where T represents the chronologically earliest recorded access among all cached entities. This selection strategy effectively prioritizes data with higher temporal locality, assuming that recently requested sensor packets are statistically more likely to be retrieved by the client interface in the immediate future.

After the object with the minimum timestamp is identified, the eradication of that object and its associated files proceeds until a reduction of Cused to a state less than or equal to Cmax is achieved. This deterministic procedure maintains the cache within its prescribed capacity limits while necessitating only minimal computational expenditure.

4. Experimental Setup

The experiment was conducted over 90 days to evaluate the long-term performance of the LRU cache under realistic IoT workloads. Twenty DHT11 sensors collected environmental data every 20 seconds, generating approximately 7.78 million telemetry records (388.8 MB). User requests included both real-time sensor queries and historical visualization requests, providing a representative workload for assessing cache efficiency under continuous data generation and repeated access patterns.

Table 2. Experimental Workload and Observation Configuration

Parameter	Value Used in the Study	Research Relevance
Observation period	90 calendar days / 3 months	This period may facilitate the capture of longitudinal cache adaptation, thereby extending beyond a short-term test window.
Number of sensors	20 DHT11 environmental sensors	This configuration ostensibly represents a medium-scale distributed IoT sensor matrix.
Edge servers	5 Raspberry Pi units	This distribution may allocate processing and storage loads across independent edge nodes.
Sensor-to-server allocation	4 sensors per edge server	This allocation tends to create balanced local traffic for each cache proxy.
Sampling interval	Every 20 seconds	This interval is posited to generate continuous high-frequency telemetry.
Measurements per day	86,400 records across all sensors	This volume appears to produce repeated time-series requests, which are suitable for cache evaluation.
Total telemetry points	Approximately 7,776,000 records	This substantial volume of data may provide a longitudinal dataset for stability analysis.
Average payload size	50 bytes per record	This parameter may establish the requisite empirical baseline for the validation of cache efficiency findings against the total generated system load.

The performance evaluation compared two operating profiles: a baseline system without caching and an LRU-based proxy caching system deployed on five Raspberry Pi edge servers. Daily telemetry collected from apt-cacher-ng logs, including HTTP requests, response codes, and data volume, was used to calculate the Request Hit Ratio **and** Data Volume Hit Ratio. These metrics measured the effectiveness of LRU caching in serving requests locally, reducing network backhaul traffic, and improving edge-layer efficiency.

The evaluation focused on two primary metrics:

- Request Hit Ratio: delineated as the proportion of client requests successfully fulfilled directly by the proxy cache, thereby avoiding retrieval from the primary storage layer. This metric, therefore, may serve as a critical indication of the edge layer's efficacy in minimizing user-perceived latency and reducing its reliance on backhaul transmission.
- Data Volume Hit Ratio: This metric is defined as the percentage of the total requested data volume, measured in megabytes satisfied by the local cache rather than necessitating retrieval from the upstream or primary storage layer. This metric, accordingly, may function as a key indication of bandwidth efficiency and backhaul transmission reduction, thereby offering a more precise evaluation of network load mitigation than request-based metrics alone.

Table 3. Evaluation Metrics Used for Cache Efficiency Analysis

Metric	Formula / Calculation Basis	Interpretation
Request Hit Ratio (%)	(cache-served requests / total requests) × 100	Measurement of the proportion of user requests resolved without recourse to the primary repository
Request Miss Ratio (%)	(cache-missed requests / total requests) × 100	Indication of the frequency with which the system accesses primary storage or effects upstream retrieval
Data Volume Hit Ratio (%)	(data volume served from cache / total requested data volume) × 100	Quantification of bandwidth and storage-load reduction, particularly through the application of megabyte-weighted traffic analysis
Data Volume Miss Ratio (%)	(data volume missed / total requested data volume) × 100	Representation of residual traffic that remains unabsorbed by the cache layer
Improvement (percentage points)	LRU average - baseline average	Elucidation of the direct performance enhancement provided by Least Recently Used (LRU) relative to the non-cached baseline

5. Result and Analysis

The log analysis derived from the five Raspberry Pi nodes may elucidate the temporal cache behavior under medium-scale Internet of Things (IoT) workload conditions. Aggregated results accumulated over three months facilitate the delineation of the initial warm-up stage from stabilized cache operation; this separation is often deemed crucial for comprehensive evaluation.

5.1 Baseline Database Performance Without Optimization

To quantify the performance gains enabled by the edge-caching architecture, the initial evaluation phase systematically contrasted the distributed twenty-sensor infrastructure against a baseline configuration devoid of proxy-cache intervention. This approach establishes a critical performance benchmark, allowing for the precise isolation of efficiency improvements directly attributable to the Least Recently Used (LRU)-driven proxy implementation. Table 4 presents an aggregate comparison between this baseline condition and the LRU-enabled experimental configuration.

Table 4. Comparative Evaluation of Cache Efficiency: Longitudinal Impact of Least Recently Used (LRU)-Optimized Edge Caching

Time Period	Caching Scenario	Request Hit Percentage	Request Miss Percentage	Total Request Count	Data Hit Volume (Megabytes)	Data Hit Percentage	Data Miss Volume (Megabytes)	Data Miss Percentage	Total Data Volume (Megabytes)
Three-Month Average	No Caching	22.50	77.50	906,000	516.00	10.50	4,410.00	89.50	4,926.00
Month 1	Least Recently Used (LRU) Caching	78.80	21.20	304,000	1,515.00	88.50	201.50	11.50	1,716.50
Month 2	Least Recently Used (LRU) Caching	85.50	14.50	299,000	1,598.00	92.80	124.20	7.20	1,722.20
Month 3	Least Recently Used (LRU) Caching	83.90	16.10	303,000	1,565.00	91.60	143.30	8.40	1,708.30
Three-Month Average	Least Recently Used (LRU) Caching	82.73	17.27	906,000	4,678.00	90.90	448.00	9.10	5,126.00

These results demonstrate the limitations of conventional database querying for repetitive IoT workloads. The baseline results suggest the substantial operational expenditure inherent in the operation of an edge IoT system devoid of cache protection. The three-month average Request Miss value reached 77.50%, suggesting that the majority of user requests necessitated access to the primary storage layer; concurrently, at the data-volume level, misses reached 89.50%, which corresponds to 4,410 MB of retrieval from either upstream or primary storage.

5.2 Post-Implementation Algorithmic Performance Impact

After enabling the LRU layer across all edge servers, the performance profile appears to have undergone substantial alteration. During the three-month observation, the average Request Hit Ratio reached 82.73%, while the average Data Volume Hit Ratio reached 90.90%. Quantitatively, 4,678 MB of the 5,126 MB requested data volume could be attributed predominantly to the cache layer.

Table 5. Derived Performance Improvement of LRU Caching Against the Baseline

Indicator	Baseline: No Caching	LRU 3-Month Average	Change / Interpretation
Request Hits (%)	22.50	82.73	+60.23 percentage points; the observed increase may indicate substantial caching of repeated queries by the cache
Request Misses (%)	77.50	17.27	-60.23 percentage points; primary storage access appears to have been substantially reduced
Data Hits (%)	10.50	90.90	+80.40 percentage points; the majority of the requested data volume appears to have been serviced by proxy caching
Data Misses (%)	89.50	9.10	-80.40 percentage points; upstream/primary retrieval volume appears to have been minimized
Data Hits (MB)	516.00	4,678.00	+4,162.00 MB; this magnitude of data provisioning appears to originate from the cache layer
Data Misses (MB)	4,410.00	448.00	-3,962.00 MB; the cessation of retrieval from the primary layer appears evident

Fig. 3 delineates the monthly fluctuations of request-hit and data-volume-hit ratios subsequent to the activation of LRU caching.

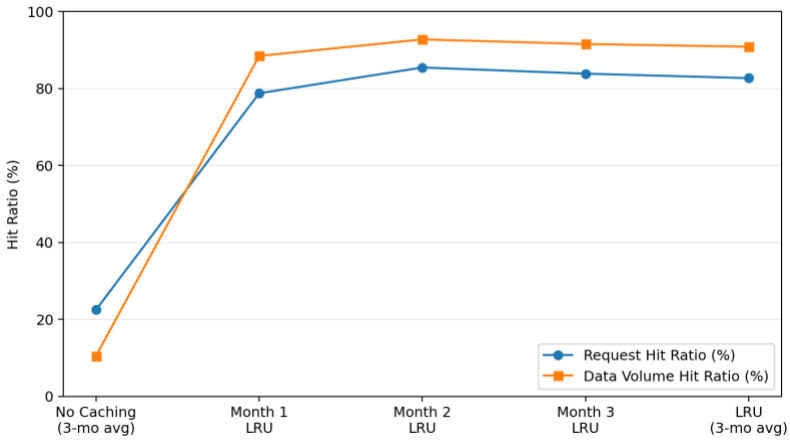

Fig. 3. Graph comparing the Request Hit Ratio and Data Volume Hit Ratio over the three-month observation period, thereby demonstrating the adaptation and stabilization phases.

Table 4 demonstrates the month-by-month results of the cache adaptation process. In Month 1, the request-hit ratio reached 78.80% because the cache started empty and required a warm-up period to accumulate frequently requested objects. During this stage, many requests resulted in compulsory cache misses before the objects were stored. In Month 2, the request-hit ratio increased to 85.50%, indicating that the cache had stabilized. The LRU policy effectively retained frequently accessed dashboard objects while removing less frequently used sensor logs, demonstrating that temporal locality enabled the cache to adapt to changing workloads without manual intervention.

The findings additionally reveal a disparity between the two-hit metrics. The three-month Data Volume Hit Ratio was 90.90%, while the Request Hit Ratio was 82.73%. This phenomenon suggests that larger requested objects were more prone to being served from cache than would be inferred solely from the total number of individual requests. The three-month evaluation confirms that the LRU caching algorithm maintains stable performance after the initial adaptation stage, achieving an average Data Volume Hit Ratio of 90.90%. This result demonstrates the effectiveness of LRU in reducing repeated database access under continuous IoT workloads. The high cache efficiency also reduces flash storage wear by serving over 90% of requested data from the proxy cache.

6. Conclusion

This paper evaluated the long-term performance of the Least Recently Used (LRU) cache replacement algorithm in a distributed IoT edge-computing environment. We implemented the proposed architecture using 20 environmental sensors and five Raspberry Pi edge servers, and we conducted a continuous 90-day evaluation to capture realistic operational behavior. Unlike short-term experiments, the extended observation period allowed us to assess cache adaptation, workload evolution, and system stability under continuous telemetry generation. Throughout the evaluation, the system processed approximately 7.7 million sensor records, providing a comprehensive dataset for analyzing cache performance in a practical edge-computing scenario.

The experimental results demonstrate that the LRU-based proxy cache effectively maintains high storage efficiency after the initial cache warm-up period. The proposed system achieved an average Data Volume Hit Ratio of 90.90%, indicating that most requested telemetry data were served directly from the edge cache instead of the primary database. This outcome significantly reduced repeated database access, lowered storage I/O activity, and improved data retrieval efficiency while maintaining stable performance throughout the three-month observation period. Furthermore, the decentralized architecture enabled each edge server to manage its own cached data independently, eliminating the need for complex cache synchronization or centralized coordination while still delivering consistent caching performance.

Therefore, this paper demonstrates that lightweight local LRU caching provides a practical, scalable, and computationally efficient baseline for medium-scale distributed IoT deployments. The proposed approach achieves high storage efficiency without relying on artificial intelligence, cooperative caching, or cluster-wide cache synchronization, thereby reducing implementation complexity and computational overhead. These findings suggest that simple temporal replacement policies remain highly effective for many edge-computing applications with localized telemetry workloads. Future research should compare the proposed approach with more advanced cache replacement algorithms, including Least Frequently Used (LFU), Adaptive Replacement Cache (ARC), Position-based Self-adaptive Filtering (PoSiF), reinforcement learning-based caching, and graph neural network-based caching under identical IoT workloads to further improve cache performance and adaptability.

Acknowledgment

The authors thank to Mr. Fahrurrozi Lubis and their student mentee, Gideon Tulus Hatta Yuda, for their assistance in the completion of this research.

References

[1] B. Qian et al., “Orchestrating the Development Lifecycle of Machine Learning-based IoT Applications,” ACM Computing Surveys, vol. 53, no. 4. Association for Computing Machinery, pp. 1–47, July 07, 2020. doi: 10.1145/3398020.

[2] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, “Optimized IoT service placement in the fog,” Service Oriented Computing and Applications, vol. 11, no. 4, pp. 427–443, Oct. 2017, doi: 10.1007/s11761-017-0219-8.

[3] S. Firdose et al., “Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions,” Computer Networks, vol. 195, pp. 108177–108177, May 2021, doi: 10.1016/j.comnet.2021.108177.

[4] A. Aboodi, T. Wan, and G.-C. Sodhy, “Survey on the Incorporation of NDN/CCN in IoT,” IEEE Access, vol. 7, pp. 71827–71858, Jan. 2019, doi: 10.1109/access.2019.2919534.

[5] J. Xu, K. Ota, and M. Dong, “Energy Efficient Hybrid Edge Caching Scheme for Tactile Internet in 5G,” IEEE Transactions on Green Communications and Networking, vol. 3, no. 2, pp. 483–493, Mar. 2019, doi: 10.1109/tgcn.2019.2905225.

[6] C. Thuan, J. Kim, I. Hwang, S. Kim, and C. H. Kim, “A Novel Last-level Cache Replacement Policy to Improve the Performance of Mobile Systems,” in Advanced Science and Technology Letters, Apr. 2014, pp. 24–28. doi: 10.14257/astl.2014.46.06.

[7] K.-Y. Wong, K. H. Yeung, K.-C. Choi, P. Lei, and C. Lam, “Exact Transient Analysis on LRU Cache Startup for Internet of Things,” pp. 310–315, Dec. 2021, doi: 10.1145/3512576.3512632.

[8] Q. Pham et al., “A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art,” arXiv (Cornell University), Feb. 2022, doi: 10.48550/arxiv.1906.08452.

[9] H. Al-Zoubi, A. Milenković, and M. Milenković, “Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite,” pp. 267–272, Apr. 2004, doi: 10.1145/986537.986601.

[10] M. H. Z. Bazargani, N. Moghim, S. Taheri, and N. Gholami, “A new cache replacement policy in named data network based on FIB table information,” Telecommunication Systems, vol. 86, no. 3, pp. 585–596, Apr. 2024, doi: 10.1007/s11235-024-01140-7.

[11] K. Ntougias, C. B. Papadias, Γ. Παπαγεωργίου, G. Haßlinger, and T. B. Sørensen, “Coordinated Caching and QoS-Aware Resource Allocation for Spectrum Sharing,” Wireless Personal Communications, vol. 126, no. 1, pp. 49–79, Mar. 2020, doi: 10.1007/s11277-020-07236-y.

[12] M. I. Zulfa, A. Fadli, A. E. Permanasari, and W. Ali, “Performance comparison of cache replacement algorithms onvarious internet traffic,” JURNAL INFOTEL, vol. 15, no. 1, pp. 1–7, Feb. 2023, doi: 10.20895/infotel.v15i1.872.

[13] S. Podlipnig and L. Böszörményi, “A survey of Web cache replacement strategies,” ACM Computing Surveys, vol. 35, no. 4, pp. 374–398, Dec. 2003, doi: 10.1145/954339.954341.

[14] H. Nicholson, P. Chrysogelos, and A. Ailamaki, “HPCache: memory-efficient OLAP through proportional caching revisited,” The VLDB Journal, vol. 33, no. 6, pp. 1775–1791, Dec. 2023, doi: 10.1007/s00778-023-00828-7.

[15] S. Arshad, M. A. Azam, M. H. Rehmani, and J. Loo, “Recent Advances in Information-Centric Networking-Based Internet of Things (ICN-IoT),” IEEE Internet of Things Journal, vol. 6, no. 2, pp. 2128–2158, Oct. 2018, doi: 10.1109/jiot.2018.2873343.

[16] M. A. M. Hail, M. Amadeo, A. Molinaro, and S. Fischer, “On the Performance of Caching and Forwarding in Information-Centric Networking for the IoT,” in Lecture notes in computer science, Springer Science+Business Media, 2015, pp. 313–326. doi: 10.1007/978-3-319-22572-2_23.

[17] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, and K. Drira, “How to cache in ICN-based IoT environments?,” HAL (Le Centre pour la Communication Scientifique Directe), Oct. 2017, Accessed: Sept. 2025. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01575386
[18] L. Saino, I. Psaras, and G. Pavlou, “Icarus: a Caching Simulator for Information Centric Networking (ICN),” Jan. 2014, doi: 10.4108/icst.simutools.2014.254630.
[19] J. Hou, H. Lu, and A. Nayak, “A GNN-based proactive caching strategy in NDN networks,” Peer-to-Peer Networking and Applications, vol. 16, no. 2, pp. 997–1009, Feb. 2023, doi: 10.1007/s12083-023-01464-2.
[20] H. Jin, D. Xu, C. Zhao, and D. Liang, “Information-centric mobile caching network frameworks and caching optimization: a survey,” EURASIP Journal on Wireless Communications and Networking, vol. 2017, no. 1, Feb. 2017, doi: 10.1186/s13638-017-0806-6.
[21] . Alahmri, S. Al-Ahmadi, and A. Belghith, “Efficient Pooling and Collaborative Cache Management for NDN/IoT Networks,” IEEE Access, vol. 9, pp. 43228–43240, Jan. 2021, doi: 10.1109/access.2021.3066133.
[22] H. Asaeda, K. Matsuzono, Y. Hayamizu, H. H. Hlaing, and A. Ooka, “A Survey of Information-Centric Networking: The Quest for Innovation,” IEICE Transactions on Communications, no. 1, pp. 139–153, Aug. 2023, doi: 10.1587/transcom.2023ebi0001.
[23] H. Zahmatkesh and F. Al-Turjman, “Fog computing for sustainable smart cities in the IoT era: Caching techniques and enabling technologies - an overview,” Sustainable Cities and Society, vol. 59, pp. 102139–102139, Apr. 2020, doi: 10.1016/j.scs.2020.102139.
[24] J. Mertz and I. Nunes, “Understanding Application-Level Caching in Web Applications: A Comprehensive Introduction and Survey of State-of-the-Art Approaches,” arXiv (Cornell University), vol. 50, no. 6. Cornell University, p. 98, Nov. 22, 2017. Accessed: Sept. 2025. [Online]. Available: http://arxiv.org/abs/2011.00477
[25] S. Sethumurugan, “RLR: Reinforcement learned replacement,” ACM Transactions on Architecture and Code Optimization, 2021.
[26] G. Vietri et al., “Driving cache replacement with ML-based LeCaR,” in USENIX Annual Technical Conference, Jan. 2018. Accessed: Oct. 2025. [Online]. Available: https://www.usenix.org/system/files/conference/hotstorage18/hotstorage18-paper-vietri.pdf
[27] J. Shuja, K. Bilal, W. Alasmary, H. Sinky, and E. Alanazi, “Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey,” Journal of Network and Computer Applications, vol. 181, pp. 103005–103005, Mar. 2021, doi: 10.1016/j.jnca.2021.103005.
[28] H. J. Mayer and J. Richards, “Comparative Analysis of Distributed Caching Algorithms: Performance Metrics and Implementation Considerations,” arXiv (Cornell University), Apr. 2025, doi: 10.48550/arxiv.2504.02220.
[29] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, and K. Drira, “How to Cache in ICN-Based IoT Environments?,” Oct. 2017, doi: 10.1109/aiccsa.2017.37.
[30] X. Wang, R. Li, C. Wang, X. Li, T. Taleb, and V. C. M. Leung, “Attention-Weighted Federated Deep Reinforcement Learning for Device-to-Device Assisted Heterogeneous Collaborative Edge Caching,” IEEE Journal on Selected Areas in Communications, vol. 39, no. 1, pp. 154–169, Nov. 2020, doi: 10.1109/jsac.2020.3036946.
[31] Y. Wang, Y. Meng, J. Wang, and C. Yang, “LSTM-CRP: Algorithm-Hardware Co-Design and Implementation of Cache Replacement Policy Using Long Short-Term Memory,” Big Data and Cognitive Computing, vol. 8, no. 10, pp. 140–140, Oct. 2024, doi: 10.3390/bdcc8100140.
[32] G. Haßlinger, M. Okhovatzadeh, K. Ntougias, F. Hasslinger, and O. Hohlfeld, “An overview of analysis methods and evaluation results for caching strategies,” Computer Networks, vol. 228, pp. 109583–109583, Mar. 2023, doi: 10.1016/j.comnet.2023.109583.
[33] C. Gray and D. R. Cheriton, “Leases: an efficient fault-tolerant mechanism for distributed file cache consistency,” pp. 202–210, Nov. 1989, doi: 10.1145/74850.74870.
[34] C. Zhang et al., “Toward Edge-Assisted Video Content Intelligent Caching With Long Short-Term Memory Learning,” IEEE Access, vol. 7, pp. 152832–152846, Jan. 2019, doi: 10.1109/access.2019.2947067.
[35] L. Hmar and L. Chhangte, “Characterizing Attention-Based Sequence Models for WirelessEdge Cache Replacement: A Short-Horizon Steady-StateBaseline,” Jan. 2026, doi: 10.21203/rs.3.rs-8013149/v1.
[36] H. Torabi, H. Khazaei, and M. Litoiu, “A Learning-Based Caching Mechanism for Edge Content Delivery,” arXiv (Cornell University), Feb. 2024, doi: 10.1145/3629526.3645037.