Received: November 1, 2024. Revised: November 27, 2024.

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Adaptive Cache Replacement Strategy Based on Greedy Dual Size Frequency Incorporating DBScan for Internet Access Anomaly Detection

Mulki Indana Zulfa[1]* **Adhwa Moyafi Hartoyo[1]** **Stephen Prasetya Chrismawan[1]**
Waleed Ali Ahmed[2]

[1]*Engineering Faculty, Jenderal Soedirman University, Purbalingga, Indonesia*
[2]*Information Technology Department, King Abdulaziz, Rabigh, Saudi Arabia*
* Corresponding author's Email: mulki_indanazulfa@unsoed.ac.id

Abstract: In content delivery networks (CDN) and edge caching, efficient cache replacement strategies are essential to reduce latency and enhance hit ratios. This study introduces adaptive similarCache (ASC), an adaptive caching framework based on greedy dual size frequency (GDSF) that includes DBSCAN for anomaly detection. ASC framework analyzes the IRcache dataset, which consists of proxy server access logs from four cities—Boulders, New York, Silicon Valey, and Urbana Campaign— and simulates dynamic adjustments to access patterns as they would occur in real-time, considerably boosting cache performance under fluctuating traffic situations when compared to classic approaches like LRU and LFU. The technique improves cache utilization, achieving a hit ratio of up to 45% and exhibiting lower latency in high-demand conditions, focusing specifically on data from these cities. The findings underscore the need for adaptive caching techniques, particularly in resource-constrained contexts, and suggest the possibility of future hybrid systems.

Keywords: CDN, Cache replacement, IRcache, Latency, Hit ratio.

1. Introduction

The rapid growth of internet usage, from its modest beginnings to the current digital era, reflects the expanding needs and expectations of global users [1]. Originally designed as a rudimentary means of disseminating information, the web has evolved into a sophisticated and interactive environment, shaping how people consume content daily [2]. With the advent of more intelligent web applications, the demand for fast, reliable, and scalable services has increased dramatically [3]. As more users rely on bandwidth-intensive applications, such as streaming services and online gaming, caching systems have become vital for maintaining speed and efficiency in data delivery [4]. These systems work by temporarily storing frequently accessed data closer to the user, thereby reducing network latency and congestion [5]. The continued development of more efficient caching strategies is critical to meeting these growing demands, ensuring that users experience minimal delays while accessing the content they require [6].

Despite the significant role caching plays, modern systems face substantial challenges in adapting to the complexities of today's internet traffic. Web content is now more dynamic, with personalized and frequently changing data, and user behavior is increasingly unpredictable. Traditional cache replacement algorithms, like Least Recently Used (LRU) and Least Frequently Used (LFU), are ill-suited to these dynamic environments, as they focus solely on access recency or frequency [7]. LRU may evict items that are soon requested again, while LFU can retain items that were frequently accessed in the past but are no longer relevant, leading to inefficient cache usage. These algorithms often result in suboptimal cache performance, particularly when access patterns change rapidly. In response, size-based caching algorithms, such as Greedy Dual Size (GDS), have emerged. GDS factors in the size of data objects, addressing some of the limitations of

traditional algorithms [8]. However, GDS's exclusive focus on object size often leads to inefficient decisions regarding frequently accessed content, which may be unnecessarily evicted, impacting the overall performance of the caching system [9].

The Greedy Dual Size Frequency (GDSF) algorithm was developed as an improvement over GDS, aiming to overcome the limitations of its predecessor by incorporating both object size and access frequency into the caching process [10]. GDSF balances the trade-offs between storing large data objects and frequently requested content, optimizing cache utilization and improving hit rates [11]. Nevertheless, GDSF is not without its challenges. The algorithm still struggles to adapt to the rapid fluctuations in internet traffic, often retaining outdated or irrelevant data due to its reliance on static frequency measures [12]. As traffic patterns evolve, the need for a more dynamic and responsive approach becomes apparent, leading to the proposal of enhanced cache replacement strategies that can better respond to these demands [13].

Traditional cache replacement approaches like GDSF runs into challenges as it is not well suited for dynamic and mobile network environments and is also prone to cache miss. The limitation actually emanates from their shortcoming in the detection of abnormal data usage and also in responding to such irregular events. To address this problem, we need to enhance GDSF method by applying an adaptive cache replacement policy, which employing DBSCAN for anomaly detection on the fly [14]. DBSCAN is particularly useful in that it finds groups of similar items and also finds items which for some reason do not belong to the clusters, making it a powerful instrument for exposing rarely occurring usage patterns [15]. Such approach enables the caching system to change as per the prevailing network situation and thus preventing the wastage of cache resources due to contamination by unworthy information using DBSCAN during the cache replacement procedure. This provides a more accurate assessment of the relevance of the data when the symptoms of the anomaly are included in the objective function of GDSF. It leads to better caching policy and better system performance.

The primary contribution of this work is an adaptive cache replacement strategy that effectively handles cache pollution by integrating DBSCAN into GDSF. This approach enhances the identification of anomalies in data access patterns, allowing the caching system to identify and exclude irregular or harmful access patterns in real time, significantly reducing the impact of cache pollution. The integration of DBSCAN also enables more intelligent eviction decisions, leading to improved cache performance across various network conditions. Additionally, the dynamic adjustment mechanism allows the system to adapt to fluctuating network conditions, providing a more agile response compared to traditional static methods like LRU and LFU. The modified GDSF algorithm employs an intelligent eviction strategy that balances object size, access frequency, and the presence of anomalies, resulting in superior cache utilization and increased hit rates. These features ensure optimal cache performance under a wide range of conditions, making it a robust solution for the challenges faced by modern caching systems, particularly in environments characterized by dynamic and unpredictable access patterns, such as edge networks and content delivery systems.

This paper first discusses related works in Section 2 to provide the foundation for the proposed method. Section 3 then elaborates on the design and implementation of{nama algoritma}, detailing how it improves upon traditional caching techniques. Section 4 presents the results of our experiments, comparing {nama algoritma} to existing caching algorithms and discussing its performance across several datasets. Finally, Section 5 concludes the study by summarizing the findings and suggesting directions for future research.

2. Related works

This paper continues our previous caching system research work in a framework called similarCache [16]. In the simulation phase, similarCache has a positive contribution to the hit ratio performance improvement tested using IRcache datasets. However, simulation results on one of the datasets (NY) have not shown good performance. The investigation shows that the NY dataset has different data access characteristics (anomalies). Therefore, we conducted a literature study on specific techniques that can detect anomalies. Aware of the future development of research towards adaptive intelligent caching systems, we specifically targeted the literature on anomaly detection techniques that use machine learning techniques, namely DBSCAN.

Integrating anomaly detection techniques into caching strategies has been suggested as a way to optimize performance further. Internet access anomalies, such as flash crowds or viral surges in specific content, can significantly affect cache efficiency and network stability. The use of Density-Based Spatial Clustering of Applications with Noise (DBSCAN) has proven effective for detecting such anomalies in network traffic. Gao et al. [17]

implemented DBSCAN to detect anomalies in IoT traffic, significantly improving the efficiency of caching systems by identifying abnormal data patterns in real-time and preventing network congestion. Han et al. [18] employed DBSCAN in lightweight anomaly detection models, enhancing caching performance in IoT environments by quickly identifying and categorizing anomalies based on network traffic characteristics. Zhang et al. [19] utilized DBSCAN in Underwater Wireless Sensor Networks (UWSNs) for anomaly detection, which improved the system's robustness against malicious routing attacks and optimized caching performance in sparse network environments. Shen et al. [20] proposed an adaptive caching system that leverages DBSCAN to improve cache hit rates by enabling elastic adjustments to network traffic demands and minimizing latency during peak usage periods.

Wang et al. [21] tackled the issue of automatically determining the optimal radius of neighborhood parameter in the DBSCAN clustering algorithm, proposing an adaptive method using the Bird Swarm Optimization (BSA) algorithm, which enhanced clustering accuracy. You et al. [22] addressed the issue of detecting anomalies in real-time body weight (BW) data of broiler breeders recorded by a precision feeding system, highlighting that traditional methods were ineffective in distinguishing reasonable BW variations from true anomalies. Zeufack et al. [23] proposed an unsupervised anomaly detection framework for computer systems by analyzing network log files, which effectively addressed the challenges associated with using DBSCAN in datasets with varying densities.

Traditional caching algorithms such as Recently Used (LRU), Least Frequently Used (LFU), First In First Out (FIFO), Greedy Dual Size (GDS), and Greedy Dual Size Frequency (GDSF) require additional capabilities in detecting ongoing access anomalies. These five algorithms generally only use one variable to decide caching content, except for GDSF which already considers access count and file size. LRU, for instance, relies exclusively on recency, evicting the least recently accessed item, which becomes highly inefficient when access patterns change quickly. This results in unnecessary cache misses and reduced performance, particularly in scenarios involving viral content or rapidly fluctuating demands, where items that were recently evicted might suddenly regain popularity [24]. LFU, on the other hand, uses frequency-based eviction, retaining items that have been accessed frequently in the past. While this strategy is useful for identifying popular content, it is inherently flawed in environments where content popularity shifts rapidly. LFU tends to retain outdated data, reducing cache efficiency when new high-priority items enter the system as noted by Cui et al. [25]. FIFO is computationally efficient, it often leads to frequent cache misses since it ignores the temporal or spatial locality of content access, which is crucial for maintaining performance in mixed-use scenarios [18]. GDSF improves upon GDS by incorporating both object size and frequency to optimize cache utilization, providing better hit rates under typical conditions. However, GDSF remains limited by its reliance on static frequency measures, which cannot accommodate rapid shifts in user behavior. As a result, it tends to retain outdated or irrelevant data, particularly in environments like edge computing, where caching strategies need to adapt dynamically to maintain efficiency [20].

Despite its improvements over traditional algorithms, GDSF has been identified with several critical limitations in prior research. Studies have shown that its reliance on static frequency measures can result in retaining less relevant data, ultimately lowering cache efficiency [26]. Cui et al. [25] also highlighted that static cache mechanisms lead to inefficient resource use, especially under dynamic conditions. Xiao et al. [24] demonstrated that the lack of adaptive mechanisms in traditional algorithms like GDSF results in increased delay and energy consumption during task offloading in mobile networks. These two studies also modified the GDSF algorithm but have not been tested using the real world IRcache dataset as a real illustration of internet access behavior.

Wang et al. [27] proposed a multi-objective data caching optimization model for edge computing environments. They introduced a Cyclic Genetic Ant Colony Algorithm which enhanced data selection and caching decisions. Zulfa et al. [28] proposed LRU-GENACO a hybrid cached data optimization method combining LRU with a Genetic Algorithm and Ant Colony Optimization (ACO), demonstrating a significant improvement in hit ratio through the use of IRCache datasets. However, both studies have not implemented anomaly detection, which can be seen from the suboptimal hit ratio performance on certain datasets.

Iqbal and Asaduzzaman [29] proposed Cache-MAB, a reinforcement learning-based hybrid caching scheme for Named Data Networks (NDN) that dynamically selects the optimal caching policy from a set of distance-based policies, allowing it to improve hit rates and reduce latency. Dhara et al. [30] proposed POPS-Cache, a caching scheme for Vehicular Named Data Networks (VNDN) that considers both predicted spatial-temporal popularity

and vehicle experience with content popularity, which improved adaptability in environments characterized by high mobility. Li et al. [31] proposed PEC, a predictive edge caching system that proactively prefetches content at the edge based on real-time user predictions using sequential learning models, allowing edge networks to better accommodate fluctuating content demands. Hidouri et al. [32] proposed Q-ICAN, a Q-learning-based intrusion prevention system for mitigating cache pollution attacks in Named Data Networks (NDN), which enhanced the network's ability to discard malicious interest packets in real-time. Based on these findings, it becomes evident that traditional cache replacement strategies are insufficient for handling the complexities of modern network traffic, often characterized by sudden spikes in demand and unpredictable user behavior.

Enhanced methods, such as those involving machine learning and anomaly detection, show promise in addressing these limitations by enabling more adaptive, context-aware, and intelligent decision-making processes. However, they still face issues such as computational overhead and sensitivity to parameter tuning, which can limit their scalability and real-world applicability. This research aims to address these challenges by proposing Adaptivc similarCache (ASC), an adaptive caching approach that combines the GDSF algorithm with DBSCAN for real-time anomaly detection. By incorporating dynamic anomaly detection and an intelligent eviction mechanism, ASC aims to improve cache utilization, reduce latency, and provide a more responsive solution to fluctuating traffic conditions, particularly in edge network environments.

3. The proposed method

3.1 Greedy dual size frequency

The ability to balance object size and access frequency has led to the widespread adoption of the GDSF algorithm in caching systems, including web caching and content delivery networks [33]. GDSF efficiently manages cache space by prioritizing frequently accessed and smaller objects but faces challenges when confronted with rapidly changing access patterns or anomalies in web traffic. The algorithm's reliance on static frequency measures often results in the retention of outdated or less relevant content during unpredictable traffic fluctuations. A dynamic modification of GDSF has been proposed to enhance its adaptability to real-time variations in internet access patterns [34].

GDSF functions as the core mechanism for cache management by balancing two critical factors: size and access frequency. Objects stored in the cache are assigned a value, Kg, which is calculated based on access frequency and size. This value informs the caching decision process, where objects with higher Kg values are retained, while those with lower values are evicted when cache space is limited [35]. The cost of each object is calculated using the formula presented in Eq. (1), which considers both the size of the object and a fixed constant. Kg is then determined by the formula in Eq. (2), combining access frequency, cost, and size to optimize the retention of frequently accessed smaller objects. This approach prioritizes such objects, leading to more efficient utilization of cache space. The calculation of Kg plays a pivotal role in maintaining balance across different content types, and the formula's emphasis on size and frequency provides a structured framework for optimizing cache performance [36].

Eviction decisions in GDSF rely on a predefined cache capacity and the Kg values of cached objects [37]. When the cache reaches capacity, objects with the lowest Kg values are removed to create space for new content. This process ensures GDSF maintains high cache efficiency under normal traffic conditions. The static nature of GDSF results in inefficiencies when access patterns change unexpectedly. Fig. 1 illustrates the eviction process, showing how objects are inserted into the cache and subsequently evicted based on their Kg values when space is limited.

$$cost = 2 + \frac{s_i}{536} \tag{1}$$

$$Kg = \frac{f_i \times cost}{s_i} \tag{2}$$

Annotation list

- *cost*:The calculated cost for caching an item, with a base value of 2 plus an additional component based on the item's size.

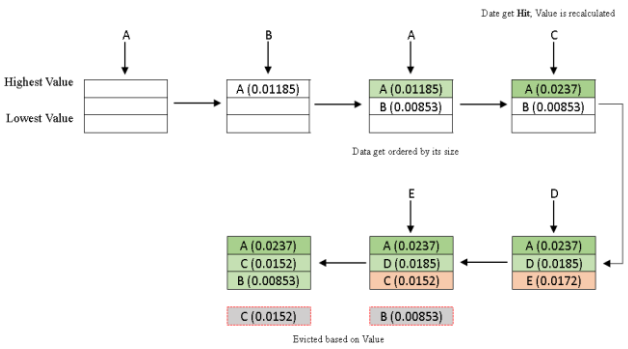


Figure. 1 Greedy dual size frequency method

- s_i: Represents the size of the *i*-th item in the cache, influencing the cost by adjusting based on its magnitude relative to 536
- 536: Represents an approximation of the number of bytes that fit into a typical network packet, used as a normalization factor to prioritize smaller objects in caching, as the are more efficient in terms of network transfer and cache utilization.
- f_i: The frequency of access for the *i*-th item, indicating how often this item has been requested.
- Kg: The weighted priority of the *i*-th cache item, computed by scaling *cost* by f_i and normalizing by the item's size (s_i).

3.2 Recency algorithm

Recency-based algorithms such as Least Recently Used (LRU), play a critical role in addressing the unpredictability of dynamic traffic by focusing on the time since an object was last accessed [38]. GDSF optimizes for size and frequency, whereas recency algorithms simplify the decision-making process by only considering the last access time of each object, making them highly adaptable to traffic that exhibits fluctuating demand. Despite their simplicity, LRU and similar algorithms may fail to retain content that, while less frequently accessed, remains valuable for future requests, highlighting a key trade-off between simplicity and long-term efficiency [39].

Recency-based caching strategies maintain a list of objects ordered by their access times. When the cache reaches its capacity, the object that has not been accessed for the longest duration is evicted, ensuring that the most recently accessed data is retained. This straightforward mechanism facilitates effective cache management in environments with highly volatile access patterns, where past behavior provides limited predictive value for future demand. Evicting the least recently accessed object ensures that the cache retains more up-to-date data, thus improving overall cache hit ratios in dynamic scenarios [40]. The eviction process in the LRU algorithm is both straightforward and effective, particularly in high-traffic environments. Upon reaching cache capacity, the algorithm automatically removes the object that has remained unaccessed for the longest period. This method ensures that the cache preserves the most recently accessed data, maintaining its relevance in dynamically fluctuating traffic conditions. Fig. 2 illustrates how objects are introduced, accessed, and subsequently evicted based on their recency.

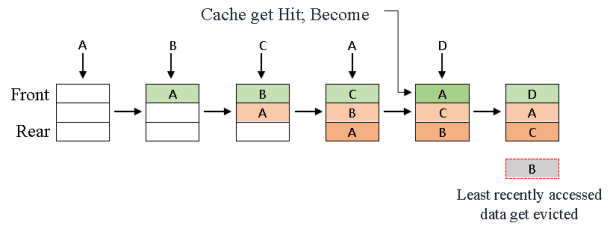


Figure. 2 Recency based algorithm

3.3 Anomaly detection

Anomalies such as viral content, flash crowds, or malicious access patterns can severely impact cache efficiency. These anomalies lead to cache pollution, where irrelevant or temporary data occupies valuable cache space, ultimately degrading overall system performance [41]. To address this issue, ASC employs an anomaly detection mechanism based on Density-Based Spatial Clustering of Applications with Noise (DBSCAN), which enables the system to detect and respond to abnormal access patterns in real time. The anomaly detection process begins with the normalization of data. Each data point, xi, representing an object in the cache, undergoes Z-score normalization to ensure that differing feature scales do not distort the results. The normalized value x_i^', is computed using Eq. (3).

Normalized data is processed using the DBSCAN clustering algorithm, which operates based on two primary parameters. The parameter ϵ (eps) defines the maximum distance between two points for them to be considered neighbors, while minPts specifies the minimum number of points required to form a dense region or cluster [41]. These parameters allow DBSCAN to effectively identify core points, border points, and outliers, enabling the differentiation between regular and anomalous access patterns [42]. Mathematically, DBSCAN defines the neighborhood $N(p)$ of a point p shown in Eq. (4). Points where the number of neighbors $N(p)$ is less than minPts are labeled as anomalies. The condition for anomaly detection is expressed in Eq. (5).

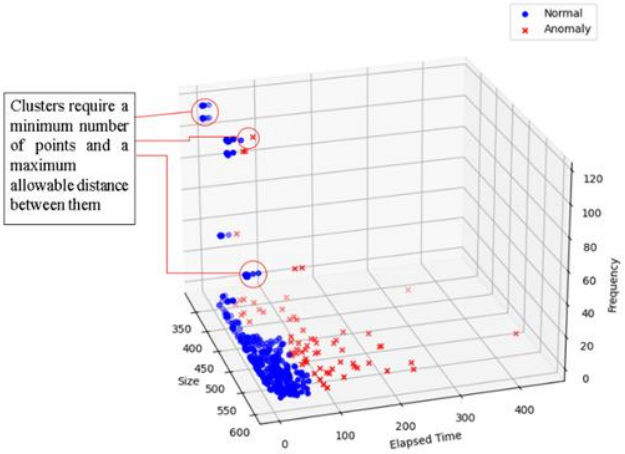


Figure. 3 DBSCAN anomaly detection visualization

The data points labeled as -1 by DBSCAN after clustering are classified as anomalies. ASC system monitors access patterns based on object size, elapsed time, and access frequency, applying DBSCAN to detect irregularities within these patterns. Anomalous requests, such as those triggered by viral content or abnormal spikes in access frequency, are flagged. Fig. 3 illustrates the anomaly detection process, where red points represent detected anomalies and blue points signify normal data points, showing how DBSCAN differentiates between normal clusters and outliers using normalized values of object size, elapsed time, and frequency.

$$x_i' = \frac{x_i - \mu}{\sigma} \tag{3}$$

Annotation list:

- x_i: Individual data point in a feature (e.g., size, elapsed time, frequency)
- μ: Mean of all data points in the feature, a central reference point.
- σ: Standard deviation of the feature, indicating data spread around μ.
- x_i': The normalized value for x_i, obtained by centering x_i around ero and scaling by feature's variability.

$$N(p) = \{q \in X \mid distance(p, q) \leq \epsilon\} \tag{4}$$

Annotation list:

- $N(p)$: Represents the neighborhood of point p, identifying nearby points within a specified distance.
- ϵ: The maximum distance threshold for points to be considered within the same neighborhood.
- q: A data point that is within the neighboorhood of p if it meets the distance condition.
- X: The set of all data points in the dataset being analyzed.
- $distance(p, q)$: The calculated distance between points p and q.

$$anomaly(p) = \begin{cases} 1 \; if |N(p)| < minPts \\ 0 \; if |N(p)| \geq minPts \end{cases} \tag{5}$$

Annotation list:

- $anomaly(p)$: A binary indicator function representing whether point p is classified as an anomaly.
- $N(p)$: The neighboorhood of point p, which contains all points within a specified distance threshold (as defined by DBSCAN)
- $minPts$: The minimum number of points required in $N(p)$for p to be classified as a core point (i.e., not an anomaly)

3.4 The adaptive similar cache (ASC)

The ability of ASC to dynamically switch between caching strategies is crucial for maintaining performance under varying traffic conditions. Under normal circumstances, ASC operates using a centroid-based eviction strategy, where objects are evaluated based on their distance from a central centroid object. The centroid is defined as the object with the highest cache value, K_{g_i}, within the cache. Objects are evicted based on their distance from this centroid, with those furthest from the centroid being evicted first, ensuring that only the most relevant data remains cached, thus optimizing cache utilization. The distance to the centroid is mathematically expressed as shown in Eq. (6).

$$d(i) = \left| K_{g_i} - C \right| \tag{6}$$

$$r_i = \frac{H}{T} \tag{7}$$

$$K_{g_i} = \frac{f_i \times \left(2 + \frac{s_i}{536}\right) \times r_i}{s_i} \tag{8}$$

Annotation list:

- $d(i)$: Distance from centroid, representing how far an object is from the central reference in the cache
- K_{g_i}: Cache value for item i, calculated based on frequency, size, and other factors (see eq. 8)
- C: Centroid, defined as the cache entry with the highest K_{g_i}value in the cache.
- r_i : Current hit ratio, a measure of cache effectiveness, adjusting dynamically.
- H : Total number of cache hits, representing successful retrievals from the cache.
- T: Total number of cache requests, giving the overall access count.
- f_i: Frequency of access for item i, denoting how often it is requested
- s_i : Size of item i, which contributes to the cache cost in relation to available space.
- 536: Represents an approximation of the number of bytes that fit into a typical network packet, used as a normalization factor to prioritize smaller objects in caching, as the are more efficient in terms of network transfer and cache utilization.

The cache value Kgi is influenced by several factors, including the frequency of access (f), the

object size (s_i), and a size normalization factor (536), which are combined with the current hit ratio r_i (Eq. (7)) to calculate K_{g_i} as shown in Eq. (8). This formula ensures that frequently accessed and valuable content is retained, while less relevant data is removed. The system selects the cache entry with the highest cache value K_(g_i) as the centroid, as defined by Eq. (9). The eviction process then targets the cache entry that maximizes the distance from this centroid, as described in Eq. (10). The decision to evict is based on finding the cache entry whose value has the greatest absolute difference from the centroid value C, ensuring that only the most relevant data remains cached, thereby optimizing cache performance.

Fig. 4 illustrates the eviction process based on the centroid, enabling ASC to dynamically adapt to varying cache contents, retaining the most critical data according to the calculated K_{g_i} value and its proximity to the centroid. Objects such as A, B, C, and D are evaluated based on this principle, ensuring optimal data retention under normal traffic conditions. When an anomaly in access patterns is detected, ASC switches to a recency-based strategy, which is better suited to handling unpredictable traffic surges. In this mode, the least recently accessed data is evicted first, ensuring that temporary or irrelevant data does not occupy valuable cache space during anomalous events. This dynamic switch to the recency-based strategy, as demonstrated in the second part of Fig. 3, prevents cache pollution and ensures that the cache is optimized for real-time traffic conditions.

This dual-strategy approach—centroid-based eviction under normal conditions and recency-based eviction during anomalies—enables ASC to maintain efficiency, regardless of whether access patterns are regular or erratic. The adaptive objective function is a critical aspect of ASC dynamic adaptation capabilities. This function integrates multiple factors, including object size, access frequency, and the current hit ratio, enabling the system to make more informed decisions about which objects to retain or evict. By adjusting to real-time traffic conditions, this objective function ensures that the cache remains efficient, even as access patterns evolve.

Table 1 provides a comparison between the Kg calculations of GDSF and ASC, demonstrating how ASC adjusts its calculations by incorporating the current hit ratio into the objective function. These adjustments allow ASC to prioritize frequently accessed content that is likely to be requested again soon, ensuring higher performance across a variety of traffic conditions. The inclusion of the current hit ratio in the ASC calculation further refines its

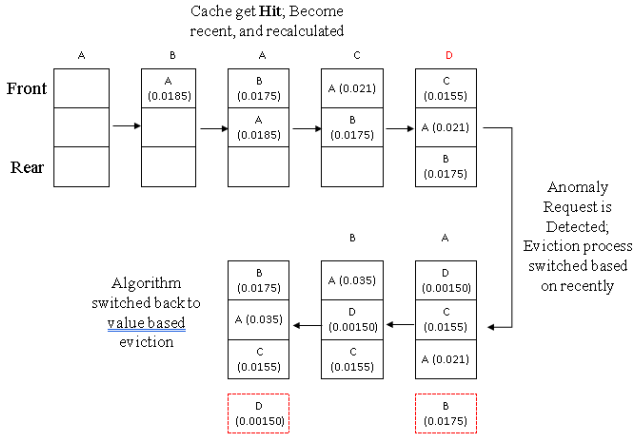


Figure. 4 The proposed ASC caching framework

Table 1. GDSF comparison

Request	GDSF Kg_i	ASC Kg_i
111	0,01187	0,01187
112	0,00853	0,00853
111	0,02373	0,00791
113	0,01519	0,0038
114	0,01853	0,00371
114	0,03707	0,01236
115	0,00587	0,00336
113	0,03039	0,01139
111	0,0356	0,01582
116	0,00758	0,00303

decision-making, allowing it to better respond to real-time fluctuations in access demand. As shown in Table 1, ASC dynamically fine-tunes its retention and eviction decisions, which leads to more optimized cache utilization under fluctuating network demands. The ability to adjust K_{g_i} based on hit ratio gives ASC a performance advantage over GDSF, particularly in environments with irregular or unpredictable traffic patterns. By adapting in real time, ASC maintains a higher level of efficiency, ensuring that relevant data is kept in the cache, while less valuable data is evicted.

4. Results and discussion

The results of this study underscore the effectiveness of the proposed caching strategy in handling diverse and dynamic internet access patterns. A comprehensive analysis across four distinct datasets—BO2, NY, SV, and UC—evaluated the performance of the ASC caching framework compared to traditional caching methods like LRU, LFU, and GDSF. Key indicators, such as hit ratio and latency saving ratio, were measured to assess the adaptability and efficiency of these strategies under

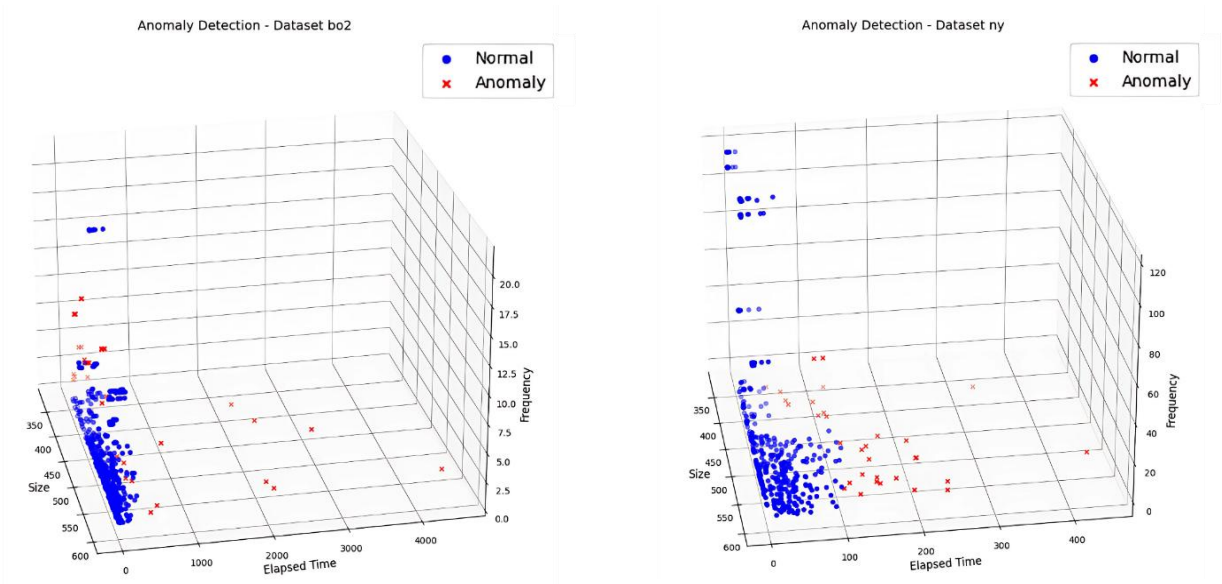
varying traffic conditions as shown Table 2. The results highlight ASC's superior ability to dynamically respond to anomalies in internet traffic, outperforming other algorithms in both stable and irregular environments. This superiority is particularly evident in high-anomaly scenarios, where static methods struggle. The adaptive nature of ASC maintains higher hit ratios and reduced latency, optimizing content delivery and enhancing user experience, with implications for modern network infrastructures that face traffic unpredictability and content diversity challenges.

4.1 Result

Fig. 5 demonstrates the anomaly detection process across four datasets: bo2, ny, sv, and uc. Anomalies, depicted by red points, are identified based on object size, frequency, and elapsed time. Dataset ny on Fig. 5(b) reveals a dense clustering of anomalies, signifying highly irregular traffic patterns that would typically challenge static caching algorithms such as LRU and FIFO. A more stable traffic pattern is observed in dataset bo2 Fig. 5(a), where fewer anomalies are present. ASC utilizes this anomaly detection to adapt its caching strategy, ensuring sustained cache efficiency. This adaptation becomes particularly evident in Fig. 6(a), where ASC achieves a significantly higher hit ratio compared to other algorithms, effectively managing abnormal traffic. The presence of anomalies in all datasets influences the hit ratios of the algorithms, with ASC consistently outperforming others due to its ability to adjust dynamically.

Fig. 6 presents the hit ratio performance across varying cache sizes and algorithms. ASC demonstrates superior hit ratios across all datasets, notably when cache sizes exceed 25K. Dataset sv Fig. 6(c) reflects ASC's ability to maintain high hit ratios even in the presence of numerous anomalies. Algorithms like LRU and FIFO, which lack dynamic adaptability, show a steep decline in performance at lower cache sizes (below 50K). Dataset uc Fig. 6(d) mirrors this trend, where ASC maintains an advantage in hit ratio throughout all cache sizes. Dataset ny Fig. 6(b) depicts a more gradual improvement across algorithms, yet ASC continues to lead in performance due to its responsive strategy to traffic irregularities. The correlation between hit ratio and anomaly detection further emphasizes ASC's ability to thrive under fluctuating conditions where static algorithms falter. Fig. 7 evaluates the Latency Saving Ratio (LSR) across different datasets and algorithms. ASC achieves the highest LSR across all datasets, with particularly notable performance in dataset bo2 Fig. 7(a) and dataset uc Fig. 7(d).

The low anomaly rate in dataset bo2, as shown in Fig. 5(a), allows ASC to maintain optimal performance with an LSR of 5.71%. In contrast, algorithms like FIFO and GDSF perform poorly in anomaly-prone environments, as illustrated in Fig. 7(b) and Fig. 7(c) for datasets ny and sv. ASC's dynamic adjustment capabilities, which respond to real-time traffic anomalies, enable it to maintain both high hit ratios and latency efficiency. Dataset sv, known for its highly irregular traffic patterns, demonstrates this relationship clearly, with ASC significantly outperforming static algorithms in both hit ratio and LSR.



(a) (b)

Figure. 5 Anomaly detection on datasets: (a) bo2 and (b) ny

Received: November 1, 2024. Revised: November 27, 2024.

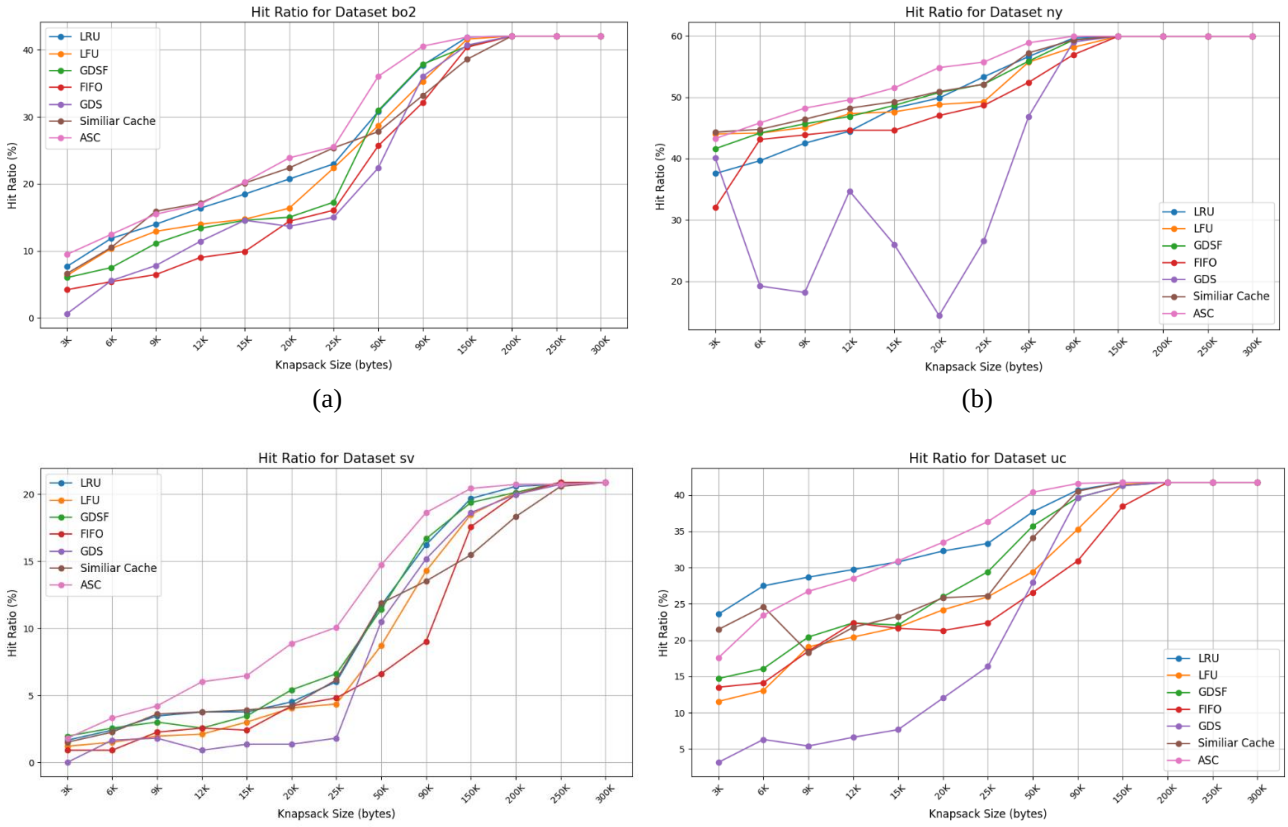


Figure. 6 Hit ratio comparison on datasets: (a) bo2, (b) ny, (c) sv, and (d) uc

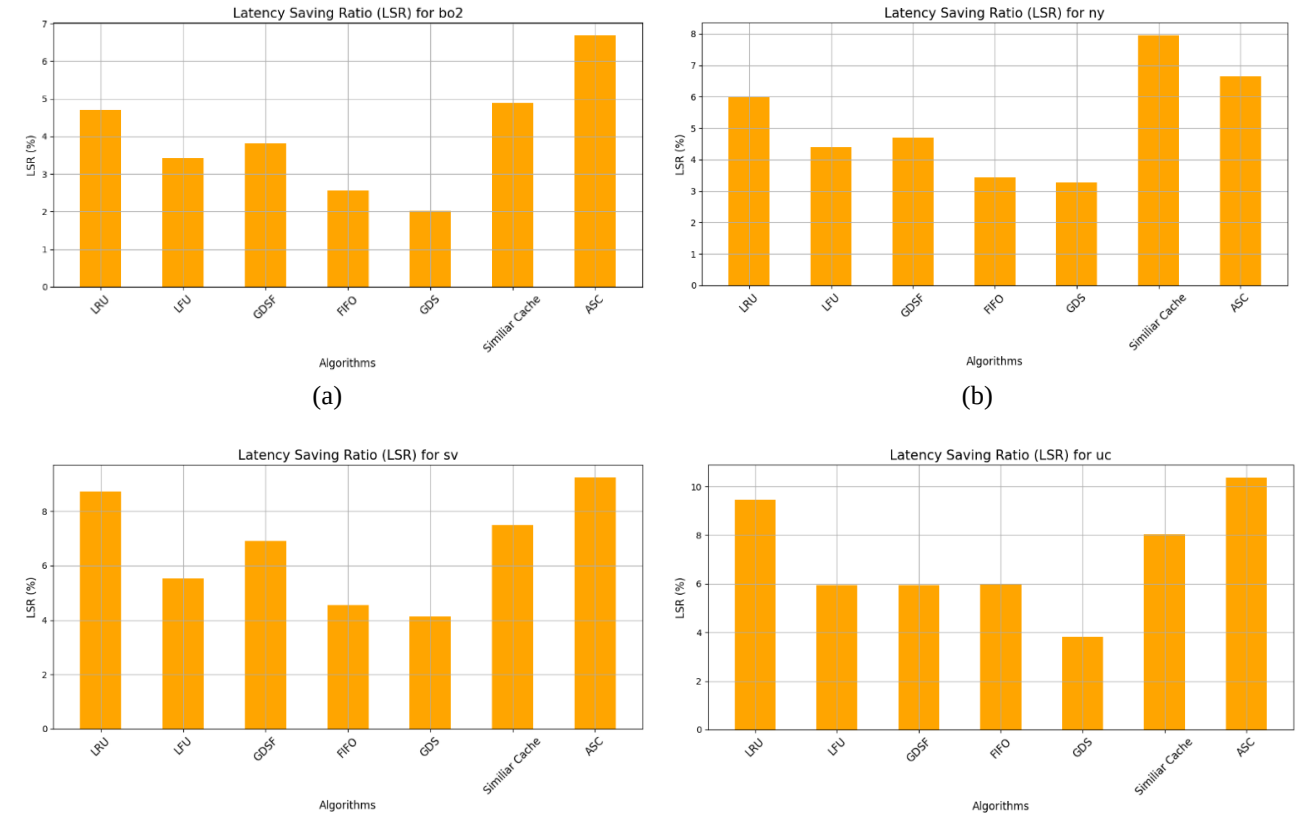


Figure. 7 Latency saving ratio comparison on datasets: (a) bo2, (b) ny, (c) sv, and (d) uc

Table 2. Average hit ratio and latency saving ratio comparison

Alg.	Avg. Hit Ratio (%)				Avg. Latency Saving Ratio (%)			
	Bo2	UC	NY	SV	Bo	UC	NY	SV
ASC	18.28	24.85	20.83	22.94	5.71	8.81	6.77	7.72
LRU	17.61	23.57	20.35	22.15	4.71	9.45	6.00	8.72
GDSF	16.07	21.28	17.57	20.05	3.82	5.95	4.71	6.92
LFU	15.77	20.95	17.27	19.74	3.42	5.95	4.39	5.52
SIZE	13.93	20.38	17.45	19.22	3.41	6.31	4.32	4.98
FIFO	12.65	19.63	15.88	17.76	2.56	6.00	3.43	4.55
GDS	10.96	13.40	8.18	8.30	2.02	3.83	3.27	4.12

4.2 Discussion

The ASC adaptive method has the most versatility across all datasets. This validates the idea that algorithms optimising recency, frequency, and magnitude are more effective for the varied access patterns seen in real-world systems. This dynamic behaviour is particularly crucial in edge networks, where content delivery fluctuates markedly based on geographic location or user activity. The efficacy of SIZE at increased cache sizes, especially in datasets such as UC, illustrates its capability in managing workloads with significant variability in object sizes. SIZE is a beneficial choice for systems managing multimedia material or extensive file distributions, since size-aware eviction may enhance cache efficiency. Although LRU performs well in most cases, it does poorly in highly variable scenarios like the UC dataset since it doesn't take non-recent factors into consideration. This suggests that although LRU could work for simple caching systems, more complex situations with complex access patterns need more sophisticated approaches.

In environments characterised by dynamic material, such as live-streaming services or content-centric platforms, algorithms like ASC, which adjust to evolving access patterns, are essential. ASC's constant superiority across all datasets indicates its exceptional suitability for these contexts. For CDNs delivering substantial multimedia assets, SIZE performance at increased cache capacities illustrates its capability. By including object size into eviction choices, SIZE reduces the probability of evicting big, frequently accessed items that are costly to retrieve from origin servers. Although complex algorithms such as ASC and GDSF provide superior performance, they may incur more computational cost relative to simpler methods like LFU and LRU.

Investigating the capabilities of hybrid algorithms that integrate the advantages of ASC and SIZE, especially for CDNs managing diverse content types, may enhance hit ratio performance. Furthermore, including Quality of Service (QoS) measurements, such as latency and byte hit ratio, into this research might provide a more thorough insight into the influence of cache replacement methods on user experience in content delivery systems. Examining cache replacement methods that dynamically adjust to access patterns and content types (such as differentiating between video streams, static photos, and tiny text items) may improve cache efficiency in multimedia-intensive CDNs.

5. Conclusion

The simulations show that the suggested caching approach works better than the others, especially when it comes to how well it works and how many hits it gets across all datasets. The ASC algorithm regularly does better than normal methods like LRU, LFU, and SIZE, especially when cache sizes are bigger. It gets hit rates as high as 40-45% across datasets. Compared to more conventional methods, including advanced features like adaptive item prioritisation improves cache hit rate and decreases latency. Nevertheless, although the suggested technique was generally successful, it exhibited considerable unpredictability when dealing with lower cache sizes, resulting in less substantial performance benefits when compared to established methods. Because of this restriction, future research might look at further optimisations, such as improving the algorithm's speed when resources are limited.

Further study should examine hybrid methods to increase performance and application-driven caching situations like QoS-based prioritisation. By adding real-time network dynamics, the design may adapt to changing traffic patterns. Applying the recommended caching strategy to present caching infrastructures gives promising results and sets the path for future intelligent and adaptive caching advancements.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

conceptualization, MULKI and ADHWA; methodology, MULKI and ADHWA; software, MULKI and ADHWA; formal analysis, MULKI and ADHWA; investigation, MULKI and ADHWA; writing: MULKI and ADHWA; visualization: STEPHEN; final review: WALEED ALI.

Acknowledgments

This research was funded by a DRTPM grant from the Ministry of Higher Education, Research and Technology based on decree number 0667/E5/AL.04/2024 and LPPM contract number of 20.44/UN23.35.5/PT.01.00/VI/2024.

References

[1] X. Wei, Y. Liu, and Y. Liu, "Study on the Impact of Internet Usage, Aging on Farm Household Income", *Sustainability*, Vol. 15, No. 19, p. 14324, 2023, doi: 10.3390/su151914324.

[2] W. Liang and W. Li, "Impact of internet usage on the subjective well-being of urban and rural households: Evidence from Vietnam", *Telecomm. Policy*, Vol. 47, No. 3, p. 102518, 2023, doi: 10.1016/j.telpol.2023.102518.

[3] K. Wakil and D. N.A.Jawawi, "Intelligent Web Applications as Future Generation of Web Applications", *Sci. J. Informatics*, Vol. 6, No. 2, pp. 213-221, 2019, doi: 10.15294/sji.v6i2.19297.

[4] H. Jens, P. Auter, M. Takaaki, K. Atsushi, and S. Kentaro, "Multi-threaded High-Level Synthesis for Bandwidth-intensive Applications", *Comput. Networks*, Vol. 31, No. 11, pp. 1203–1213, 1999, doi: 10.1016/S1389-1286(99)00055-9.

[5] J. Sim *et al.*, "Computational CXL-Memory Solution for Accelerating Memory-Intensive Applications", *IEEE Comput. Archit. Lett.*, Vol. 22, pp. 5-8, 2023.

[6] F. Zeng, K. Zhang, L. Wu, and J. Wu, "Efficient Caching in Vehicular Edge Computing Based on Edge-Cloud Collaboration", *IEEE Trans. Veh. Technol.*, Vol. 72, No. 2, pp. 2468-2481, 2023, doi: 10.1109/TVT.2022.3213130.

[7] H. Jens, P. Auter, M. Takaaki, K. Atsushi, and S. Kentaro, "Multi-threaded High-Level Synthesis for Bandwidth-intensive Applications", In: *Proc. of IEICE The Institute of Electronics, Information and Communication Engineers*, Kitakyushu: RECONF, pp. 51–56, 2019.

[8] T. Ma, J. Qu, W. Shen, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan, "Weighted Greedy Dual Size Frequency Based Caching Replacement Algorithm", *IEEE Access*, Vol. 6, pp. 7214-7223, 2018, doi: 10.1109/ACCESS.2018.2790381.

[9] M. I. Zulfa, A. Fadli, A. E. Permanasari, and W. A. Ahmed, "Performance comparison of cache replacement algorithms onvarious internet traffic", *J. INFOTEL*, Vol. 15, No. 1, pp. 1-7, 2023, doi: 10.20895/infotel.v15i1.872.

[10] M. I. Zulfa, R. Hartanto, and A. E. Permanasari, "Caching strategy for Web application - a systematic literature review", *Int. J. Web Inf. Syst.*, Vol. 16, No. 5, pp. 545-569, 2020, doi: 10.1108/IJWIS-06-2020-0032.

[11] K. S. Lam, "Dynamic Cache Replacement Policy Selection Using Experts", 2022.

[12] R. Cai, Y. Qian, and D. Wei, "Dynamic Cache Replacement Strategy of Space Information Network Based on Cache Value", *J. Phys. Conf. Ser.*, Vol. 2290, 2022.

[13] L. Li, C. Ye, and H. Zhou, "Cache Replacement Algorithm Based on Dynamic Constraints in Microservice Platform", In: *Proc. of 2022 International Conference on Service Science (ICSS)*, pp. 167-174, 2022, doi: 10.1109/ICSS55994.2022.00033.

[14] M. Civera, L. Sibille, L. Zanotti Fragonara, and R. Ceravolo, "A DBSCAN-based automated operational modal analysis algorithm for bridge monitoring", *Measurement*, Vol. 208, p. 112451, 2023, doi: 10.1016/j.measurement.2023.112451.

[15] S. Chowdhury, N. Helian, and R. Cordeiro de Amorim, "Feature weighting in DBSCAN using reverse nearest neighbours", *Pattern Recognit.*, Vol. 137, p. 109314, 2023, doi: 10.1016/j.patcog.2023.109314.

[16] M. I. Zulfa, S. Maryani, - Ardiansyah, T. Widiyaningtyas, and W. Ali, "Application-Level Caching Approach Based on Enhanced Aging Factor and Pearson Correlation Coefficient", *JOIV Int. J. Informatics Vis.*, Vol. 8, No. 1, p. 31, 2024, doi: 10.62527/joiv.8.1.2143.

[17] H. Gao, B. Qiu, R. J. D. Barroso, W. Hussain, Y. Xu, and X. Wang, "TSMAE: A Novel Anomaly Detection Approach for Internet of Things Time Series Data Using Memory-Augmented Autoencoder", *IEEE Trans. Netw. Sci. Eng.*, Vol. 10, pp. 2978-2990, 2023.

[18] D. Han *et al.*, "LMCA: a lightweight anomaly network traffic detection model integrating adjusted mobilenet and coordinate attention mechanism for IoT", *Telecommun. Syst.*, Vol. 84, pp. 549-564, 2023.

[19] R. Zhang, J. Zhang, Q. Wang, and H. Zhang, "DOIDS: An Intrusion Detection Scheme Based on DBSCAN for Opportunistic Routing in Underwater Wireless Sensor Networks", *Sensors*, Vol. 23, No. 4, 2023, doi: 10.3390/s23042096.

[20] J. Shen *et al.*, "Ditto: An Elastic and Adaptive Memory-Disaggregated Caching System", In: *Proc. 29th ACM Symp. Oper. Syst. Princ.*, pp. 675-691, 2023, doi: 10.1145/3600006.3613144.

[21] L. Wang, H. Wang, X. Han, and W. Zhou, "A novel adaptive density-based spatial clustering of application with noise based on bird swarm optimization algorithm", *Comput. Commun.*, Vol. 174, No. February, pp. 205-214, 2021, doi: 10.1016/j.comcom.2021.03.021.

[22] J. You, E. Lou, M. Afrouziyeh, N. M. Zukiwsky, and M. J. Zuidhof, "A supervised machine learning method to detect anomalous real-time broiler breeder body weight data recorded by a precision feeding system", *Comput. Electron. Agric.*, Vol. 185, No. October 2020, p. 106171, 2021, doi: 10.1016/j.compag.2021.106171.

[23] V. Zeufack, D. Kim, D. Seo, and A. Lee, "An unsupervised anomaly detection framework for detecting anomalies in real time through network system's log files analysis", *High-Confidence Comput.*, Vol. 1, No. 2, p. 100030, 2021, doi: 10.1016/j.hcc.2021.100030.

[24] Z. Xiao *et al.*, "Multi-Objective Parallel Task Offloading and Content Caching in D2D-Aided MEC Networks", *IEEE Trans. Mob. Comput.*, Vol. 22, No. 11, pp. 6599-6615, 2023, doi: 10.1109/TMC.2022.3199876.

[25] L. Cui *et al.*, "CREAT: Blockchain-Assisted Compression Algorithm of Federated Learning for Content Caching in Edge Computing", *IEEE Internet Things J.*, Vol. 9, pp. 14151-14161, 2022.

[26] X. Zhi, X. Yan, B. Tang, Z. Yin, Y. Zhu, and M. Zhou, "CoroGraph: Bridging Cache Efficiency and Work Efficiency for Graph Algorithm Execution", *Proc. VLDB Endow.*, Vol. 17, No. 4, pp. 891-903, 2023, doi: 10.14778/3636218.3636240.

[27] D. Wang, X. An, X. Zhou, and X. Lü, "Data cache optimization model based on cyclic genetic ant colony algorithm in edge computing environment", *Int. J. Distrib. Sens. Networks*, Vol. 15, No. 8, p. 155014771986786, 2019, doi: 10.1177/1550147719867864.

[28] M. I. Zulfa, R. Hartanto, A. E. Permanasari, and W. Ali, "LRU-GENACO: A Hybrid Cached Data Optimization Based on the Least Used Method Improved Using Ant Colony and Genetic Algorithms", *Electronics*, Vol. 11, No. 19, p. 2978, 2022, doi: 10.3390/electronics11192978.

[29] S. M. A. Iqbal and Asaduzzaman, "Cache-MAB: A reinforcement learning-based hybrid caching scheme in named data networks", *Futur. Gener. Comput. Syst.*, Vol. 147, pp. 163-178, 2023, doi: 10.1016/j.future.2023.04.032.

[30] S. Dhara, A. Majidi, and S. Clarke, "Revving up VNDN: Efficient caching and forwarding by expanding content popularity perspective and mobility", *Comput. Commun.*, Vol. 212, No. October, pp. 342-352, 2023, doi: 10.1016/j.comcom.2023.10.004.

[31] C. Li, X. Wang, T. Zong, H. Cao, and Y. Liu, "Predictive edge caching through deep mining of sequential patterns in user content retrievals", *Comput. Networks*, Vol. 233, No. February, p. 109866, 2023, doi: 10.1016/j.comnet.2023.109866.

[32] A. Hidouri, H. Touati, M. Hadded, N. Hajlaoui, P. Muhlethaler, and S. Bouzefrane, "Q-ICAN: A Q-learning based cache pollution attack mitigation approach for named data networking", *Comput. Networks*, Vol. 235, No. May, p. 109998, 2023, doi: 10.1016/j.comnet.2023.109998.

[33] P. Li, Y. Guo, and Y. Gu, "Predicting Reuse Interval for Optimized Web Caching: An LSTM-Based Machine Learning Approach", In: *Proc. of SC22 Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 1-15, 2022.

[34] S. Kudagi, "Survey on different cache replacement algorithms", *Int. J. Innov. Res. Eng. Multidiscip. Phys. Sci.*, Vol. 7, No. 6, pp. 10–13, 2019, doi: 10.37082/IJIRMPS/BXDRC.

[35] K. Wang and F. Chen, "Catalyst: Optimizing Cache Management for Large In-memory Key-value Systems", *Proc. VLDB Endow.*, Vol. 16, No. 13, pp. 4339-4352, 2023, doi: 10.14778/3625054.3625068.

[36] X. Lu, R. Wang, and X. H. Sun, "CARE: A Concurrency-Aware Enhanced Lightweight Cache Management Framework", In: *Proc. of Int. Symp. High-Performance Comput. Archit.*, Vol. 2023-Febru, pp. 1208-1220, 2023, doi: 10.1109/HPCA56546.2023.10071125.

[37] Y. Fu, Q. Yu, A. K. Y. Wong, Z. Shi, H. Wang, and T. Q. S. Quek, "Exploiting Coding and Recommendation to Improve Cache Efficiency of Reliability-Aware Wireless Edge Caching Networks", *IEEE Trans. Wirel. Commun.*, Vol. 20, pp. 7243-7256, 2021.

[38] B. Sethi, S. K. Addya, and S. K. Ghosh, "LCS : Alleviating Total Cold Starts Latency in

Serverless Applications with LRU Warm Container Approach", In: *Proc.of 24th Int. Conf. Distrib. Comput. Netw.*, 2023.

[39] J. Yang, Z. Qiu, Y. Zhang, Y. Yue, and K. V. Rashmi, "FIFO can be Better than LRU: the Power of Lazy Promotion and Quick Demotion", In: *Proc. 19th Work. Hot Top. Oper. Syst.*, pp. 70-79, 2023, doi: 10.1145/3593856.3595887.

[40] T. Xie, T. He, P. McDaniel, and N. Nambiar, "Attack resilience of cache replacement policies", In: *Proc. of IEEE INFOCOM*, Vol. 2021, No. 6, pp. 2433-2447, 2021, doi: 10.1109/INFOCOM42981.2021.9488697.

[41] F. Ozge Ozkok, "International Journal of Intelligent Systems and Applications in Engineering A New Approach to Determine Eps Parameter of DBSCAN Algorithm", *Orig. Res. Pap. Int. J. Intell. Syst. Appl. Eng. IJISAE*, Vol. 5, No. 4, p. 247, 2017, doi: 10.1039/b0000.

[42] N. Valarmathy and S. Krishnaveni, "WITHDRAWN: A novel method to enhance the performance evaluation of DBSCAN clustering algorithm using different distinguished metrics", *Mater. Today Proc.*, 2020, doi: 10.1016/j.matpr.2020.09.623.