Beyond Belady to Attain a Seemingly Unattainable Byte Miss Ratio for Content Delivery Networks

Peng Wang, Hong Jiang, *Fellow, IEEE*, Yu Liu, *Member, IEEE*, Zhelong Zhao, Ke Zhou, *Member, IEEE*, and Zhihai Huang

***Abstract*—Reducing the byte miss ratio (BMR) in the Content Delivery Network (CDN) caches can help providers save on the cost of paying for traffic. When evicting objects or files of different sizes in the caches of CDNs, it is no longer sufficient to pursue an optimal object miss ratio (OMR) by approximating Belady to ensure an optimal BMR. Our experimental observations suggest that there are multiple request sequence windows. In these windows, a replacement policy prioritizes the eviction of objects with large sizes and ultimately evicts the object with the longest reuse distance, lowering the BMR without increasing the OMR. To accurately capture those windows, we monitor the changes in OMR and BMR using a deep reinforcement learning (RL) model and then implement a BMR-friendly replacement algorithm in these windows. Based on this policy, we propose a Belady and Size Eviction (LRU-BaSE) algorithm that reduces BMR while maintaining OMR. To make LRU-BaSE efficient and practical, we address the feedback delay problem of RL with a two-pronged approach. On the one hand, we shorten the LRU-base decision region based on the observation that the rear section of the cache queue contains most of the eviction candidates. On the other hand, the request distribution on CDNs makes it feasible to divide the learning region into multiple sub-regions that are each learned with reduced time and increased accuracy. In real CDN systems, LRU-BaSE outperforms LRU by reducing "backing to OS" traffic and access latency by 30.05% and 17.07%, respectively, on average. In simulator tests, LRU-BaSE outperforms state-of-the-art cache replacement policies. On average, LRU-BaSE's BMR is 0.63% and 0.33% less than that of Belady and Practical Flow-based Offline Optimal (PFOO), respectively. In addition, compared to Learning Relaxed Belady (LRB), LRU-BaSE can yield relatively stable performance when facing workload drift.**

***Index Terms*—Byte miss ratio, content delivery network, rear section, reinforcement learning, replacement algorithm.**

Received 14 May 2023; revised 2 August 2024; accepted 27 August 2024. Date of publication 30 August 2024; date of current version 9 September 2024. This work was supported in part by the Key Laboratory of Information Storage System and Ministry of Education of China, in part by the National Key Research and Development Program under Grant 2023YFB4502701, in part by National Natural Science Foundation of China under Grant 62232007 and Grant 61821003, in part by National Natural Science Foundation of China under Grant 61902135, and the in part by the Natural Science Foundation of Hubei Province under Grant 2022CFB060. Recommended for acceptance by S. Byna. *(Corresponding author: Yu Liu.)*

Peng Wang and Ke Zhou are with the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: wp_hust@hust.edu.cn; zhke@hust.edu.cn).

Hong Jiang is with the Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX 76019 USA (e-mail: hong.jiang@uta.edu).

Yu Liu and Zhelong Zhao are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: liu_yu@hust.edu.cn; zhelongzhao@hust.edu.cn).

Zhihai Huang is with Tencent, Shenzhen 518000, China (e-mail: tommyhuang@tencent.com).

Digital Object Identifier 10.1109/TPDS.2024.3452096

I. Introduction

The object miss ratio (OMR) and the byte miss ratio (BMR) are two key metrics in caching. The former measures the quality of experience (QoE) while the latter directly influences the cost for cache service providers. In Content Delivery Networks (CDNs) [1], [2], [3], [4], vendors typically pay for data traffic proportionally to the amount of data transmitted. As a result, they focus on reducing the BMR in caching. The conventional approaches pursue a lower BMR by pursuing a lower OMR through heuristic replacement algorithms since data locality [5], [6], [7] can benefit both OMR and BMR when objects in the cache are of similar size. However, since objects in CDNs have various sizes, there is no guarantee that the optimal BMR and optimal OMR will coincide. To illustrate this, we provide a toy example. Consider an access sequence shown in Fig. 1, where object-A is 1 MB and 4 MB in size in Case 1 and Case 2, respectively, while the size of all other objects is 1 MB in both cases. Using LRU [8], [9], Belady [10], [11], and Belady-Size [12] on a 6 MB cache, we obtain the eviction sequences, OMRs, and BMRs, where the LRU is a classic heuristics cache replacement policy, the Belady is the optimal cache replacement policy that evicts the objects with the longest reuse distance, and the Belady-Size policy prioritizes the eviction of the object with the biggest product of the reuse distance and the object size. Comparing the BMR and OMR results of Belady's and Belady-Size's with those of the ideal case, we conclude that while Belady attains the minimum OMR, it fails to guarantee the minimum BMR when object sizes are different.

This phenomenon sparks our interest in minimizing BMR by considering object size when using Belady. Note that previous researchers attempted to reduce BMR by incorporating object size into heuristic approaches [7], [13], [14], [15], [16], [17], [18], [19], [20] where the replacement policies prioritize a predefined size threshold, followed by the Belady approximation, resulting in a trade-off between OMR and BMR. However, we believe that there is room to reduce BMR while maintaining OMR since we have observed that evicting objects within an appropriate reuse distance range does not increase the OMR. In other words, the OMR only increases if a replacement policy cannot evict the object with the longest reuse distance before the end of a request sequence window. During this request sequence window, evicting objects that are large in size rather than in reuse distance can achieve a win-win performance for both OMR and

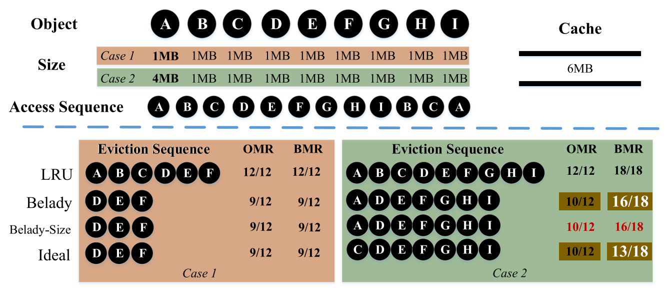


Fig. 1. Toy example—*the phenomenon of Belady's inability to guarantee the minimum BMR when object sizes are different.*

BMR (See Section III). Nevertheless, capturing this window by traditional methods is difficult because it varies substantially by different traces and different cache sizes.

To accurately capture this request sequence window, we employ reinforcement learning (RL) [21], [22], [23] and propose a Belady and Size Eviction algorithm based on LRU (LRU-BaSE). The RL's Markov process can monitor the performance change caused by adjacent replacement behaviors. The reward function favors the behavior that maintains OMR while reducing BMR, implicitly capturing the request sequence windows to achieve a win-win performance for OMR and BMR. To tackle the problem of feedback delay [24] in RL, we implement LRU-BaSE using a two-pronged approach. First, we observe that the LRU algorithm forces data with a relatively long reuse distance to be concentrated in a rear section of the queue (See Section IV-A-1). LRU-BaSE shortens the decision region from the entire queue to this section, reducing its decision time. Second, we leverage data locality to divide the learning time region into multiple sub-regions based on tests. We then learn models for each of these sub-regions (See Section IV-A-2). This narrows the learning range for each model, lowering the training time with accurate decisions. These tricks, combined with the lightweight network, improve accuracy and efficiency.

Our evaluation of LRU-BaSE in real CDN systems shows that, compared to the default cache algorithm (i.e., LRU), LRU-BaSE can reduce "backing to OS" traffic by 30.05%. According to the prices listed in [25], [26], it equates to an annual savings of \$795,000 in bandwidth costs. In addition, LRU-BaSE decreases the average access latency and the tail latency at the 99.9^{th} percentile by 17.07% and 66.90%, respectively. Compared to the state-of-the-art cache replacement algorithms on the simulator, LRU-BaSE outperforms the competition in OMR and BMR on public CDN traces. The BMR generated by LRU-BaSE is, on average, 0.63% and 0.33% lower than the lower bound of BMR (i.e., that of Belady and Practical Flow-based Offline Optimal (PFOO) [12], respectively). Finally, we mock a workload drift [27] scene and demonstrate that LRU-BaSE is more robust than Learning Relaxed Belady (LRB) [28] for obtaining superior BMR.

II. Background and Motivation

A. Lessons From Real Systems

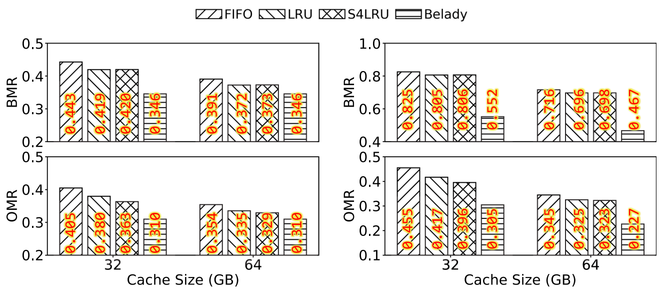
QQPhoto is *Tencent*'s cloud service product, specifically designed as an e-album solution within the context of CDNs. The cache layer of QQPhoto is composed of the *data center cache* (DC) and the *outside cache* (OC). OC is closer to clients, such as local access, in terms of access latency, and helps improve user experience quality. DC is located within the data center to alleviate the traffic burden on its storage system. OC and DC form a distributed cache to handle requests for photos of different sizes, which is the typical structure of a CDN. In practice, when a request fails to hit, it will be sent back to the data center, forming the so-called "backing to OS (Original Server)" traffic. The increasing of the traffic consumes costly data center resources and results in performance degradation, such as WAN bandwidth for OC, storage system I/O bandwidth for DC, and increased latencies, *etc*. To deter it, the LRU algorithm is widely deployed in the cache layer. As an effective approach, it evicts the least recently accessed page based on the assumption that pages used recently will likely be used again soon.

In August 2019, the daily download traffic for QQPhoto services was 214.846 Gbps, and the daily "backing up to OS" traffic was 73.166 Gbps. It implies a BMR of 34.06%. To further reduce the "backing to OS" traffic, we tested S4LRU [8] which is a cache replacement algorithm proven theoretically better than LRU. S4LRU (or Segmented LRU) [8] enhances the basic LRU by dividing cache pages into segments based on their recency and promoting pages to higher segments upon repeated access, thus balancing between recency and frequency for better cache performance. However, the daily traffic increased to 88.197 Gbps instead. Meanwhile, we found that OMR decreased from 32.81% to 31.77%. Changes in OMR and BMR are not always consistent because the sum of the sizes of objects hit is not always determined by the number of objects hit. This counterintuitive phenomenon inspired us to pay attention to the OMR-and-BMR relationship and how to reduce the "backing to OS" traffic in CDNs by improving cache replacement algorithms.

B. Belady is No Panacea for BMR

To explain the above counterintuitive phenomenon, we demonstrate S4LRU's improvement in OMR and BMR over LRU through its accurate approximation of Belady. Belady, also known as the Optimal Page Replacement Algorithm, replaces the page that will not be used for the longest time in the future, offering the lowest possible page fault rate. However, it is impractical for real-world use due to its need for future knowledge. As shown in Figs. 2 and 3, we present OMR, BMR, and the distribution of reuse distances of victims at different cache sizes, using the FIFO algorithm to display the order of requests in the trace.

On the Tencent trace gathered from the workload described in Section II-A, OMR and BMR at different cache sizes are shown in Fig. 2(a). The longest reuse distance was divided into 10 sub-ranges, with each sub-range corresponding to a 10% increment, e.g., $[0.9, 1]$ for the sub-range of 90% to 100% of the longest reuse distance. Fig. 3(a) to (c) display the distributions of reuse distances of evicted objects, where the height of each bar indicates the frequency of occurrence of evicted objects in the corresponding sub-range. As shown in Fig. 2(a), S4LRU consistently outperforms LRU in OMR and BMR at all cache sizes,



(a) Tencent (b) Wikipedia

Fig. 2. BMR and OMR on Tencent and Wikipedia under different cache sizes and cache algorithms.

except for 128 GB, where an anomaly occurs. Here, S4LRU's BMR is slightly higher than LRU's BMR by 0.03%, while the former maintains a 0.1% advantage over the latter in OMR. Referring to these distributions, we discovered that S4LRU consistently evicts more objects with the longest reuse distances than LRU, despite the number of objects in $[0.9, 1]$ being similar. Further analysis of the statistics revealed that the reuse distances of objects in $[0.9, 1]$ are all the longest. Consequently, we believe that, while S4LRU more accurately approximates the Belady algorithm, it fails to produce an optimal BMR. To avoid bias from the results on a single trace, we perform the same test on the public trace, i.e., Wikipedia [28]. As shown in Fig. 2(b), the relative performances of S4LRU and LRU on Wikipedia differ from those on Tencent. LRU outperforms S4LRU in BMR at all cache sizes, except for 32 GB. Although we can attribute S4LRU's inferiority to LRU to the underutilization of space resources in S4LRU [8], [29], the algorithm that evicts the most objects with the longest reuse distance is still S4LRU. This yet again demonstrates that approximating Belady alone no longer ensures an optimal BMR. A possible explanation for this phenomenon is that the Belady algorithm tends to evict objects with relatively large sizes since these objects tend to have greater reuse distances [30]. Intuitively, this strategy is conducive to absorbing more objects with relatively small sizes, albeit it is not necessarily beneficial for BMR [7]. As a result, we believe that approximating Belady can serve as a basis for achieving a satisfying BMR, but for acquiring a superior BMR, one should consider object sizes.

Regarding heuristic methods that use object size as a feature to approximate Belady, some algorithms address the gap between OMR and BMR by leveraging object size to reduce BMR [7], [16], [17], [19], [20]. Abrams et al. [7] suggested that it is more effective to retain many small objects rather than a few large ones. They further discussed "the number of bytes not sent" for replacement policies in their subsequent works [13], [14]. The GreedyDual-Size [15] algorithm introduced the concept of eviction cost, associating object sizes with BMR. Greedy-Dual* [18] provided additional evidence that the gap between BMR and OMR varies with different cache sizes. These efforts demonstrate that the gap between OMR and BMR is not random and is related to object size. Additionally, while Belady excels at achieving the best OMR, it is no longer capable of providing optimal BMR.

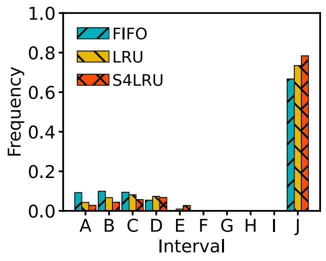
III. Rationale of "Belady With Size"

We believe that there is potential for further BMR reduction based on Belady. Our analysis of existing heuristic-based cache replacement policies, informed by both the Belady algorithm and object size [13], [19], [31], suggests that they generally follow a strategy of evicting the object with a relatively large size to make room for caching more objects. They treat the OMR and BMR as independent or adversarial objectives. The results in Fig. 1 confirm that this strategy is not ideal for BMR. In addition, learning-based replacement policies [24], [28], [32] learn by labels crafted by reuse distances, albeit with features containing object size. This largely results in only OMR being affected by the object size. Ignoring BMR results in higher costs for web service providers, whereas minimizing BMR by compromising OMR leads to a lower quality of user experience [33], [34], [35].

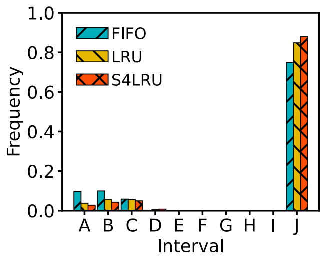
Nevertheless, we believe that there is room for BMR to decrease while maintaining OMR. Due to the continuity of eviction behaviors, the object with the longest reuse distance may not be replaced upon the current request but could be replaced in the next or one of the next few requests, which may not affect OMR. For example, in Case 2 shown in Fig. 1, although the reuse distance of object-A is longer than that of object-C, prioritizing the eviction of object-C does not increase OMR. Based on this insight, we randomly evict one of the top-N objects with the longest reuse distance when the request miss occurs. The OMR curves on Tencent and Wikipedia at different cache sizes are shown in Fig. 4, where the shadow represents the upper and lower bound values under 500 repetitions of the test. In addition to the trend of curves growing as N increases, we observed some "flat" regions in the curves (e.g., $N = 16$ and $N = 4$ for Tencent and Wikipedia, respectively, at a cache size of 32 GB). The existence of these regions confirms our insight that evicting the longest-reuse-distance object upon each request miss is not necessarily an absolute requirement for optimizing OMR. As a result, we believe that BMR and OMR have a chance to achieve a "win-win" outcome by judiciously evicting an object that can lower BMR within a specific request sequence window without increasing OMR, just as we evicted object-C in Case 2 of Fig. 1. This approach may result in a lower BMR than the result yielded by Belady. However, since the "flat" region significantly changes with cache sizes or traces, as shown in Fig. 4, it is difficult to capture this request sequence window manually. Note that LRB [28] attempted to capture this window by relaxing Belady's range through an analysis of limited data, but it is difficult for the pre-defined boundary to accommodate possible workload drift [27]. We confirm this by the experimental results in Section VII-F.

IV. Methodology

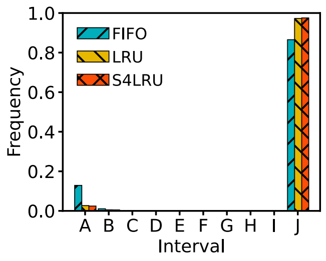
Dynamically identifying "flat" regions by monitoring request behaviors is key to capturing the request sequence windows. Intuitively, these windows should match the periodicity of the request and be able to be detected by traditional heuristics. As



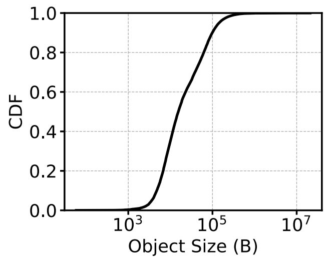
(a) 32GB on Tencent



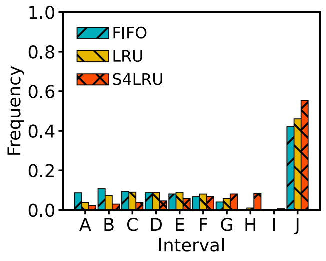
(b) 64GB on Tencent



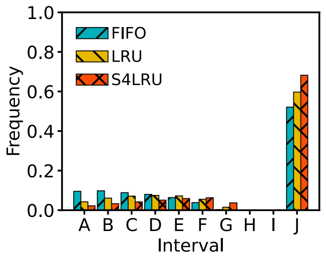
(c) 128GB on Tencent



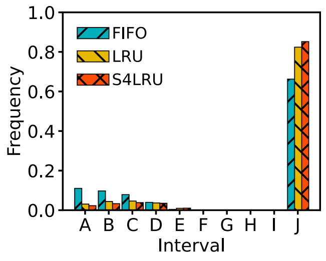
(d) Size distribution on Tencent



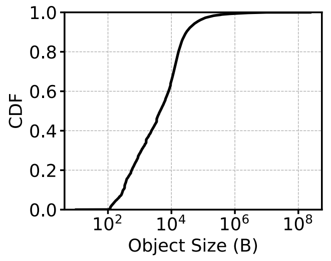
(e) 32GB on Wikipedia



(f) 64GB on Wikipedia



(g) 128GB on Wikipedia



(h) Size distribution on Wikipedia

Fig. 3. Reuse distance distribution of evicted objects under different cache sizes and distribution of object sizes on two traces. A to J respectively represent intervals, i.e., sub-ranges [0, 0.1), [0.1, 0.2), [0.2, 0.3), [0.3, 0.4), [0.4, 0.5), [0.5, 0.6), [0.6, 0.7), [0.7, 0.8), [0.8, 0.9), and [0.9, 1].

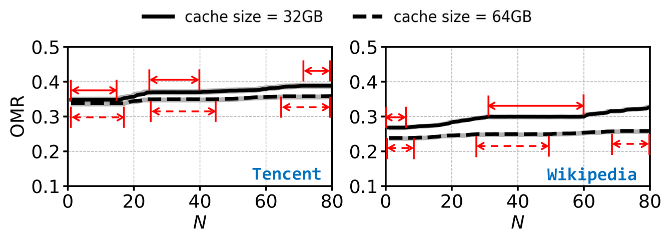
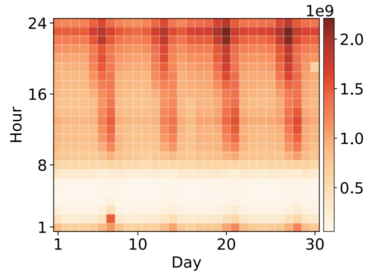
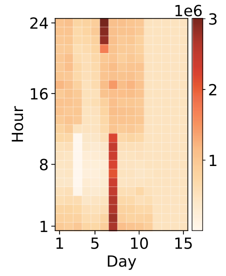


Fig. 4. OMR by evicting any one of the top-N longest-reuse-distance objects. This experiment is implemented under a warmed-up cache. The redline-marked segments of the curves are the "flat" regions and also our desired request sequence windows.



(a) Tencent



(b) Wikipedia

Fig. 5. Tencent—*the number of user requests during the 24 hours, for the 30 days of the observed mouth.* Wikipedia - *the number of user requests during the 24 hours, for the 15 days of the observed mouth.* Each column corresponds to a day, each row represents a particular hour of the day, and the color on a given hour of a given day signifies the average request rate during that hour. The darker the color, the lower the request rate.

shown in the left matrix of Fig. 5(a), we observe that the Tencent trace has a stable daily periodicity in user request rate throughout the given month. However, the existence of the window does not depend on this periodicity. As shown in the right matrix of Fig. 5(b), there is little evidence of this periodicity in Wikipedia. With such different daily request behavior patterns, we have to use a unified learning model to "perceive" the request distribution and identify "flat" regions. The learning model can dynamically determine request sequence windows and actively evict objects based on a joint consideration of Belady and object sizes while reducing overhead and improving performance. Furthermore, it aims to improve outcomes across varied workloads, whether or not they exhibit periodicity.

A. *Learning by the RL Model*

As traditional classification models [36], [37] are insensitive to temporal changes, they may not be sufficiently competent to identify this window. To the best of our knowledge, Long Short-Term Memory (LSTM) [38], Transformer [39], and Reinforcement Learning (RL) models [21], [22] can achieve the purpose of monitoring in the temporal domain. However, since it is an NP-hard problem [12], we cannot establish criteria that relate to BMR in a similar way to how the reuse distance relates to OMR, resulting in a lack of labels to learn by LSTM and Transformer. RL, on the other hand, can use BMR as a learning target directly. For instance, LeCaR [21] and CACHEUS [40] decide to switch between two replacement policies based on the change of OMR. Nevertheless, RL models have a problem with feedback delay and high training overhead. To solve this problem, in addition to deploying the lightweight model structure, we narrow the decision scope in the spatial space and learning range in the temporal space via the distribution of longest-reuse-distance objects.

1) Making Decision on the Rear Section: The LRU algorithm promotes the object that is hit to the head of the queue while leaving objects that have not been hit for a long time and generally have long reuse distances to loiter at the rear section of the queue. Note that our experimental analysis in

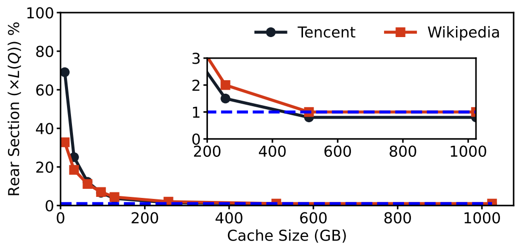


Fig. 6. Curve fitted from ECDF on Tencent and Wikipedia.

TABLE I
THE RESULTS OF EVICTING DATA RANDOMLY SELECTED IN THE REAR SECTION (RS) AND IN THE ENTIRE QUEUE (EQ)

Cache Size		Tencent		Wikipedia	
		RS	EQ	RS	EQ
128GB	BMR	**33.51%**	35.45%	**41.5%**	44.07%
	OMR	**31.58%**	33.88%	**31.1%**	34.29%
256GB	BMR	**30.89%**	31.14%	**34.02%**	36.31%
	OMR	**30.02%**	31.04%	**24.96%**	26.26%

Section II-B reveals that this rear section is relatively narrow. If we can accurately identify and estimate this rear section from the queue, we can minimize the spatial domain input to the RL model. This would narrow down the decision range while lowering the OMR and the overhead. To verify this hypothesis, we compute the cumulative distribution of the position of the longest-reuse-distance objects under different cache sizes. We define the eviction cumulative distribution function for candidate objects (ECDF) as:

$$F_X(x) = P(X \leq x), (0 \leq x \leq 1)$$

where $X = D(O_d)/L(Q)$, O_d denotes a candidate object, Q represents the cache queue, $D(O_d)$ denotes the distance between the location of O_d and the end of Q, $L(Q) = N$ represents the size of Q (i.e., total number of objects Q can hold). According to the ECDF values of $F_X(x) < 0.99999$ over LRU at varying cache sizes from 10 GB to 1024 GB, we fit the curve using the logistic regression algorithm and show the curve in Fig. 6. The results on both Tencent and Wikipedia show that as the cache size grows, the conjectured rear section shrinks in proportion to the entire queue. However, when the cache size exceeds 256 GB, the proportion remains stable at 1%.

To demonstrate that the objects in the rear section are more deserving targets for eviction, we compare the performance of the BMR and OMR yielded by evicting the objects randomly selected in the entire queue (EQ) versus in the rear section (RS). Table I shows that the decision on RS (i.e., 1% of EQ) yields better BMR and OMR than that on EQ. We attribute these results to the higher density of "right" objects in RS than that in EQ.

As a result, we can train and make decisions on the data sampled in RS, trading off between efficiency and precision in a time-sensitive environment. Moreover, recording and learning data in RS greatly reduces the memory footprint compared to doing that in EQ. Table II shows the training and decision time on RS and EQ using the Deep Q-Learning (DQN) [41] model at 128 GB and 256 GB cache sizes. As expected, the lower the

TABLE II
TRAINING TIME (TT) AND DECISION TIME (DT) ON THE REAR SECTION (RS) AND THE ENTIRE QUEUE (EQ) AT 128 GB AND 256 GB

Cache Size		Tencent		Wikipedia	
		RS	EQ	RS	EQ
128GB	TT	27.78 h	71.12 h	22.97 h	47.01 h
	DT	935.08μs	2041.76μs	840.45μs	1503.76μs
256GB	TT	32.18 h	95.13h	26.44 h	70.87 h
	DT	1118.52μs	3628.67μs	997.24μs	2418.05μs

The experiment runs on a device with a 56-core 2.40GHz CPU, 512GB of RAM, and a 6TB SSD.

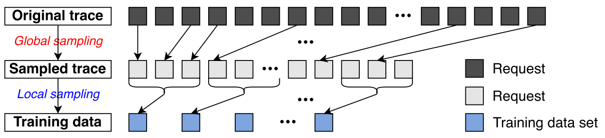


Fig. 7. Global and local sampling methods.

dimensionality of the input, the shorter the training and decision time.

2) Sampling for Proper Learning Ranges: To further reduce training time on the rear section, we focus on the sampling strategy to eliminate unpromising training data. Note that sampling can also address the issue of RL's inability to capture uniform strategies and make accurate decisions due to learning from large training data sets that contain multiple behavior patterns. To this end, we have designed global and local sampling methods. As shown in Fig. 7, global sampling selects request data from the entire trace at intervals to form sampled data, while local sampling selects consecutive requests from sampled data to form multiple training data sets. We require that the distribution of sampled data approximates the requested distribution of the original trace. In addition, each training data set must exhibit behavioral or data locality, i.e., it must perform well using similar strategies in a defined time region.

To verify the validity of our sampling methods, we selected Day 3 from the Tencent trace as learning data, where Day 3 contains 1 billion requests. We sampled from these requests to form training data sets and train the DQN model, and we used the model to estimate the requests for Day 4 on the 128 GB cache. First, we determined the sampling rate of $\frac{1}{100}$ according to the optimal BMR. Then, we used this sampling rate to gather a sampled trace. To determine local sampling, we sliced the training data into multiple non-overlapping data sets with the same time span and trained the model for each data set using the DQN model. We independently repeated the above tests 1000 times and showed the average results in Fig. 8.

The shorter the time span, the shorter the average training time, and the smaller the variance of BMR, as shown in Fig. 8. Note that we cannot tolerate overlapping model training, i.e., the model's training duration is longer than the learning range's time span, due to limited computing resources used for updates (i.e., online training). Consequently, we retrain each model within the learning range's time span. As shown in Fig. 8, a 6-hour time span is the optimal scheme. In addition, we tested the time spans for the following 6 days and obtained similar results, but we did not show them due to space limitations. To further determine the

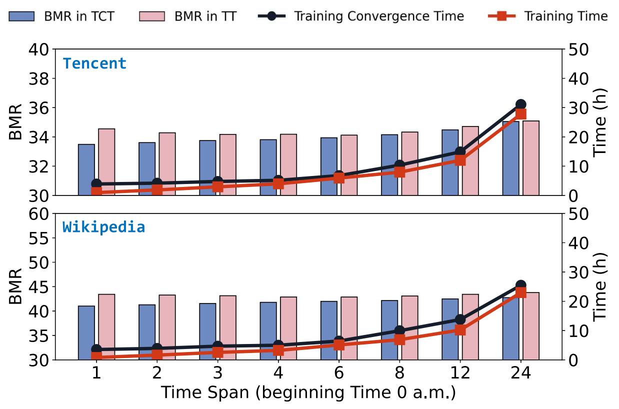


Fig. 8. Time span tests on Tencent and Wikipedia by the DQN model. TCT and TT represent the training convergence time and training time using CPU, respectively.

TABLE III
BMR and OMR With Different Beginning Times

Beginning Time (a.m.)	Tencent BMR	Tencent OMR	Wikipedia BMR	Wikipedia OMR
0	34.12 ± 0.02	32.36 ± 0.02	51.88 ± 0.01	40.45 ± 0.01
1	33.79 ± 0.01	31.77 ± 0.01	41.99 ± 0.02	40.77 ± 0.02
2	$\mathbf{33.75 \pm 0.01}$	$\mathbf{31.68 \pm 0.01}$	42.15 ± 0.03	40.98 ± 0.03
3	33.81 ± 0.02	31.82 ± 0.02	42.01 ± 0.02	40.72 ± 0.02
4	33.89 ± 0.01	31.94 ± 0.01	41.84 ± 0.01	40.30 ± 0.01
5	33.93 ± 0.02	32.05 ± 0.02	$\mathbf{41.62 \pm 0.01}$	$\mathbf{40.25 \pm 0.01}$

proper time regions, we adjusted the beginning time of the time span based on the periodicity and showed the results in terms of BMR and OMR in Table III.

Based on these results, we confirmed that 2 a.m. is the best beginning time for Tencent. Reviewing Fig. 5, it's clear that the number of requests per hour in each time region is close (small chromatism) using this beginning time. Hence, we prefer to train four models for each day, with each model's training data representing requests in 6 hours and beginning at 2 a.m. In addition, while the Wikipedia trace does not have a similar periodicity as Tencent, we still arrive at a similar conclusion. As shown in Fig. 8, the best performance occurs with a 4-hour or 6-hour time span. Using these time spans, we display the performance at different beginning times in Table III and conclude that the best performance is achieved by beginning at 5 a.m. These results are exciting because they confirm that the selection of time span and start point is beneficial for using RL. Moreover, this choice is feasible and general for all traces.

B. Representative Learning Data

The final issue for RL is to determine which periodic data is most representative. In the above verification, we selected the data from 24 hours ago as learning data (i.e., training data on Day 3 for Day 4) based on the data locality. However, after observing the request distribution in Fig. 5, we found that data from a week ago more precisely matches the target data. Therefore, we compared the learning effects of using data from 24 hours ago and a week ago on Tencent. We trained the data of Day 3 and Day 9 using DQN to make decisions for the requests of Day 10, and we trained the data of Day 13 and Day 19 to make

TABLE IV
Learning Results Over Different Periodic Data

Target Day		Tencent 24 hours	Tencent 1 week	Wikipedia 24 hours	Wikipedia 1 week
10	BMR	**33.78%**	35.28%	**41.78%**	45.28%
	OMR	**31.66%**	33.46%	**29.16%**	32.7%
20	BMR	**35.45%**	36.38%	-	-
	OMR	**33.49%**	35.28%	-	-

TABLE V
Description of the Variables and Parameters Used in Section V

Vars and Pars	Description
$\boldsymbol{s}$ / $\boldsymbol{s}_t$	$\boldsymbol{s}$: *state* feature vector; $\boldsymbol{s}_t$: *state* feature vector at t
r	r: *reward* scalar
$\boldsymbol{a}$ / $\boldsymbol{a}_t$	$\boldsymbol{a}$: *action* scalar; $\boldsymbol{a}_t$: *action* scalar at t
B_i / O_i	B_i / O_i: BMR / OMR at the i-th step
ΔB_i / ΔD_i	ΔB_i / ΔD_i: the delta of BMR / OMR at the i-th step
α / β / γ	key-step reward function parameters shown in Table 6
O / o_i	O: the set of objects in rear section; o_i: the i-th object
S	S: the starting time
T-i	T-i: the training model generated in the i-th time region
D-i	D-i: the decision model generated in the i-th time region
ΔT	ΔT: time interval

decisions for the requests of Day 20, which differ significantly from their counterparts on Day 10. As shown in Table IV, using 24-hour-old data as learning data yields better results in both BMR and OMR. We believe that data locality plays a key role. Meanwhile, as shown in Fig. 5(a), the results on Wikipedia also show that using 24-hour-old data is the best choice. Furthermore, using 24-hour-old data can reduce the space overhead required to save 1-week-old data.

V. Design of LRU-BaSE

Based on the observations and analysis presented in previous sections, we propose an LRU-based Belady algorithm with size eviction, called LRU-BaSE, using the RL model. When a request hits its target object in the cache, the caching system promotes the object using the LRU algorithm based on recency. If the request is missed, the caching system uses the learned model's policy to complete the eviction and replacement processes. To make it easier for the reader to understand, we have summarized all the variables and parameters in this section in Table V.

A. Learning Features and Training Data

For the learning features, we select **frequency**, **reuse distance**, **reuse time** and **object size** [12], [42] from the content of the past request information. Frequency counts the number of times an event occurs. On the other hand, reuse distance and reuse time measure the number of objects and the duration, respectively, between two consecutive accesses to a particular object. Note that reuse time is important for perceiving reuse behavior over time. In addition, we include object size in the set of learning features because photos can have different sizes.

In the implementation, we extract the training data from log files using the reservoir sampling approach [43], [44] and transform it into vectors. Specifically, we extract all unique IDs and sample $\frac{1}{100}$ of the IDs to form an ID set. Based on this ID set,

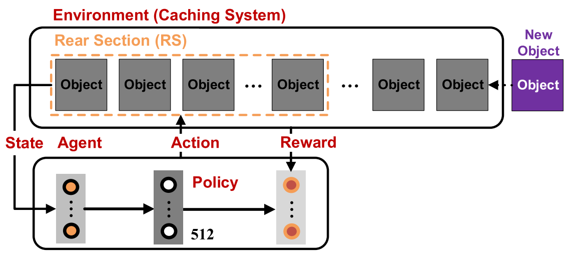


Fig. 9. The architecture of LRU-BaSE.

TABLE VI
THE CHANGES OF OMR AND BMR UNDER DIFFERENT HYPER-PARAMETERS OF THE KEY-STEP REWARD FUNCTION

Hyper-Parameter			Tencent		Wikipedia	
α	β	γ	OMR (%)	BMR (%)	OMR (%)	BMR (%)
0.4	0.4	0.2	31.38	32.77	1.02	3.17
0.3	0.5	0.2	31.89(↑0.51)	33.19(↑0.42)	1.05(↑0.03)	3.25(↑0.08)
0.5	0.3	0.2	31.61(↑0.23)	33.11(↑0.34)	1.07(↑0.05)	3.21(↑0.04)
0.35	0.35	0.3	31.45(↑0.07)	33.16(↑0.05)	1.03(↑0.01)	3.17
0.45	0.45	0.1	31.40(↑0.02)	33.16(↑0.05)	1.01(↓0.01)	3.18(↑0.01)

Each result is the average of the results of 50 runs.

we sequentially extract the final training data. To fill the elements of the training data vectors, we tally the number of times an object in the queue is hit and label this number as frequency. We calculate both reuse distance and reuse time, labeling them according to request order and timestamp, respectively. Note that we control the object size distribution of the training data in this procedure to approach the object size distribution of the original trace.

B. LRU-BaSE Architecture

We use DQN [41] as the backbone of the architecture because object ID is a discrete value. We attempted to compute fine-grained probabilities for eviction using the A3C [45] or DDPG [46] method, but this did not improve performance and increased the parameter footprint. DQN uses neural networks to learn the Q-score [47] produced by continuous decisions. To ensure efficient decision-making, we configure a lightweight network with only one hidden layer of 512 neurons. The action space in LRU-BaSE is the rear section of the cache queue described in Section IV-A-1, and an agent module is configured to learn the model and make decisions on this rear section. The architecture is shown in Fig. 9.

1) Modeling: The LRU-BaSE architecture incorporates DQN to reflect six key elements, i.e., *environment*, *state*, *reward*, *policy*, *action*, and *agent*, as illustrated in Fig. 9 and described below.

Environment: Environment is the concrete caching system. It is not only the entity of the cache queue but also generates the *state* and the performance after each decision.

State: State, denoted by $\boldsymbol{s}$, is represented by the feature vector of the rear section in LRU-BaSE, which is an input to *agent*.

Reward: Reward is a scalar denoted as r. It is calculated by a function that defines the expectation for performance changes. Generally, if the current performance is better than the last performance, the *Reward* will be positive, otherwise, it will be negative (See Section V-B-2).

Policy: Policy is a function, i.e., a network with parameters, that transforms a *state* into *action*. In LRU-BaSE, the *policy* is learned by the network, *state* and *reward*.

Action: Action is a scalar denoted as $\boldsymbol{a}$, which represents the index of the victim object in the queue array. We denote the *action* as $\boldsymbol{a}_t$ corresponding to the *state* $\boldsymbol{s}_t$.

Agent: Agent is a black box consisting of *policy*. Like a brain, it inputs the *state* to the network, adjusts the parameters by the *reward*, and outputs the *action* to the *environment*. Note that during online decision at runtime, the *agent* merely executes the process of input and output.

2) Key-Step Reward Function: The reward function is crucial in RL. We desire that the function can encourage the behavior that maintains OMR and minimizes BMR. In addition, to minimize the feedback delay, RL in the cache can only decide one step. Thus, we design the *key-step reward function* focusing on the local benefits rather than the global ones. Assume that B_i and O_i represent BMR and OMR respectively at the i-th step, $\Delta B_i = \frac{-B_i+B_{i-1}}{B_{i-1}}$, $\Delta O_i = \frac{-O_i+O_{i-1}}{O_{i-1}}$, $\Delta B_i^0 = \frac{-B_i+B_0}{B_0}$, $\Delta O_i^0 = \frac{-O_i+O_0}{O_0}$, and $\Delta D_i = \Delta B_i - \Delta O_i$. The reward function is described below.

$$r = \alpha \frac{\Delta B_i^0}{1-\Delta B_i} + \beta \frac{\Delta O_i^0}{1-\Delta O_i} + \gamma \Delta D_i, \qquad (1)$$

where $\alpha + \beta + \gamma = 1$. In theory, we require $\alpha = \beta > \gamma$ to give priority to the consistent reduction of OMR and BMR, and then to encourage a faster decline for BMR. When $r < 0$, DQN will terminate the learning process and discard this transition. If $r \geq 0$ and $\Delta D_i > 0$, DQN will terminate the learning process and store this transition, which is our *key step*. Otherwise, DQN will continue the learning process in the current episode.

With a cache size of 1024 GB, we test OMR and BMR at different settings of α, β, and γ on Tencent and Wikipedia. According to the results shown in Table VI, we determine that $\alpha = \beta = 0.4$ and $\gamma = 0.2$ is a desired setting.

3) Model Usage: Assume that the set of objects in the rear section is $O = \{o_1, \ldots, o_k, \ldots, o_N\}$, where o_k denotes the k-th object and the size of the rear section is equal to N.

Training: Assume that the cache queue is full and replacement by LRU has occurred at least 1000 times. When a new object is loaded into the cache and the object o_{N+1} is loaded into the rear section, the *environment* selects $\{o_1, o_2, \ldots, o_N\}$ to participate in training. With features from Section V-A, *state* $\boldsymbol{s}_t$ is input to the *agent*. The *agent* learns the *policy* by the style of DQN, outputs *action* by the *policy*, and informs the *environment* which object should be evicted. The *environment* will generate a new *state* and *reward* after eviction. The *state* and *reward* will be input into the *agent* to train a new *policy*. A new *action* will be generated by the new *policy* and a new *state* will be produced from the *environment*. This cycle will continue until the termination requirement is satisfied. The training result (i.e., *policy*) of each model will be saved as a 512-dimensional floating-point vector.

Decision: With a trained *policy*, the current *state* gathered from the rear section is input into the *agent*, and the *agent*

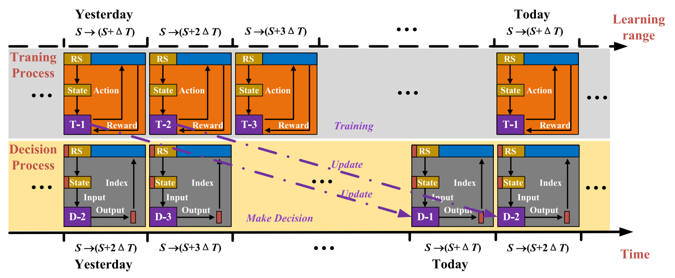


Fig. 10. The flow diagram of LRU-BaSE. The timeline where “Time” is located represents the time regions for training and executing the model, whereas the timeline where “Learning range” is located represents the time regions of the training data.

outputs an *action*. According to this *action*, the system selects an object to evict from the cache queue. Specifically, the *agent* only outputs the index corresponding to the *action* that acquires the maximum score. The index is equal to 0 when the object is at the end of the queue. *Environment* locates the object via the index and evicts it. Since the traversing using a key is omitted, the decision is efficient.

Update: To maintain the model's reliability by data locality, we must update the model by training the data from 1 d ago. To maintain decision-making consistency, the training process must be continuous, and in parallel with another model that is making decisions. Since each model's training time does not exceed the time span, CPU resources outside of cache running and decision-making are sufficient to maintain the update process. When it comes to another time region, the corresponding *policy* transfers to the current model. We describe the detailed process in Section V-C.

C. Workflow

Based on the above design, we show the workflow of LRU-BaSE in Fig. 10. The training process is depicted in parallel to the decision process along the timeline. T-i represents the LRU-BaSE model used for training the data generated in the $i-th$ time region (i.e., S a.m. to $S+i\times\Delta T$ a.m.) while D-i represents another LRU-BaSE model used for making decisions on the $i-th$ time region. The parameters (i.e., the *policy*) of D-i on *Today* are inherited from the parameters of T-i on *Yesterday*. T-1 begins training when entering the 2^{nd} time region because the requests in the 1^{st} time region have already been completed and collected. D-2 makes replacement decisions and produces the index of eviction candidates at the same time. T-2 begins training when entering the 3^{nd} time region with the initial parameters of T-1. Meanwhile, D-3 begins to exercise decision-making. In this order, when the 1^{st} time region of today arrives, D-1 makes decisions after updating the *policy* to that of T-1, which was trained one day ago. The update time is within 1s, and LRU takes over the replacement during this time. Next, T-1 retrains its *policy* at the 2^{nd} time region. D-2 updates its *policy* from T-2 and then makes decisions. The whole process will repeat. Note that there is only one training model and one decision model. Training and decision in different time regions require only corresponding inputs and *policies*. As shown in Fig. 10, the training process is separate from the decision-making process, with the latter consuming nearly negligible computing resources. The training process can be designed for offline use, thereby not impacting the performance of the live network.

D. Parallel Decision

As shown in Fig. 10, the training and decision processes are performed on a single CPU core continuously, resulting in a high CPU resource demand. In a real system implementation supporting compatibility with multi-threaded data access and avoiding consistency problems, we propose a parallel decision scheme for LRU-BaSE, with a multi-threaded execution approach working on multiple CPU cores for the replacement policy to ensure ample computational resources and online efficiency. For example, the OC layer of QQPhoto is deployed on a CPU with 56 cores and works with multiple threads, where one thread is used for data gathering and cleaning, another thread is used for model training, and the rest of the threads are used for data access from users.

We update the *action* to a vector that consists of the index sequence of the objects recommended for eviction, sorted in descending order based on their likelihood of eviction. Note that the likelihood of eviction stems from the Q-scores of object ID in the rear section. Generally, we will set the dimension of the vector to be equal to the maximum number of threads. When a batch of threads conducts eviction operations, LRU-BaSE will delineate the top-T indexes from the vector for eviction, where T is equal to the number of threads. By parallelizing eviction operations and avoiding multiple waits for DQN's recommendations, this scheme helps to ensure efficiency in real systems.

VI. Deployment and Performance of LRU-BaSE on Real System TDC

A. Deployment

Tencent Disk Cache (TDC) is an SSD-based disk cache system of *Tencent*. The system already has 3000+ physical machines online that provide large-capacity and high-performance distributed key-value cache service to 2500+ businesses, including QQPhoto. TDC consists primarily of two modules: *Cell Master* and *Cache Server*. The *Cell Master*, as a resource management module, is responsible for managing the routing information of the whole system, periodically detecting the running status of each server, and receiving the statistical information reported by each server. The *Cache Server* is the data storage and processing engine, and its prototype is a storage node based on MCP++, multi-ccd/multi-smcd process models, raw disk, inode, and asynchronous disk I/O technologies(e.g., libaio/SPDK). It is used for storing and processing business data, i.e., objects, where the deployed GPUs can achieve data processing. On the top layer of the disk, a memory cache stores the keys and indexes of objects in addition to the learning optimization processes described in Section IV, where each process works on exclusive memory space and the index in the memory cache is in perfect sync with the objects on the disk. We replace LRU with LRU-BaSE on the memory cache rather than

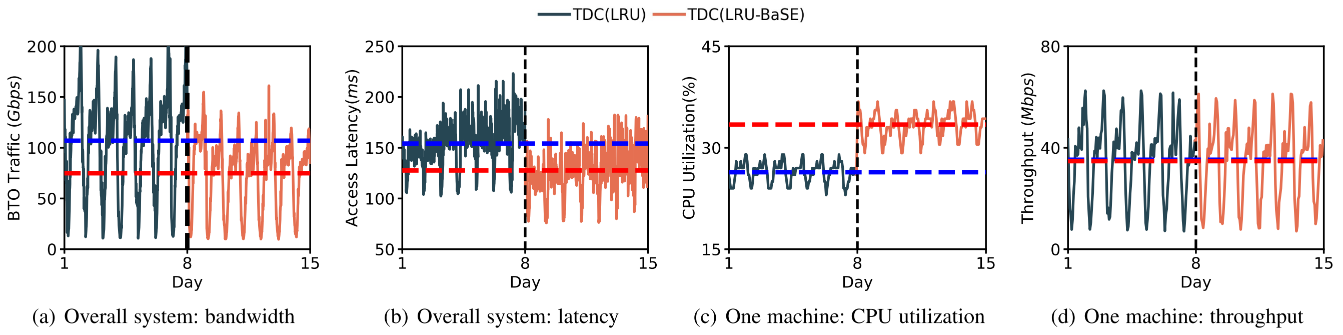


Fig. 11. Changes in performance and overhead of TDC after the online deployment of LRU-BaSE. LRU was replaced by LRU-BaSE at 24:00 on Day 7. The purple and red dashed lines represent the average values of LRU's results in 7 days and LRU-BaSE's results in 7 days, respectively. (a) and (b) represent performance measured from the overall system, while (c) and (d) represent overhead and performance measured from a physical machine randomly selected from the system.

on the disk, since it helps the endurance of SSDs by reducing write operations due to frequent updates to the index. In addition, due to the data periodicity characteristics shown in Fig. 5, deploying LRU-Base to a specific application only requires analysis for a fixed period. In addition to the parameters listed in Table V, there are several key parameters in the real system Time-Division Controller (TDC). First, the value range of the Rear Section (RS) shown in Fig. 6 is 1‰ of the Entire Queue (EQ) in the actual environment. Second, the starting time S shown in Fig. 10 is 0 a.m. Third, the interval between the training model (T-i) and the decision-making model (D-i), as shown in Fig. 10, is 1 d. Lastly, ΔT equals 6 hours.

B. Performance

As shown in Fig. 11(a) and (b), we measured the performance changes in "backing to OS" traffic and average user access latency from the monitor system. After deploying LRU-BaSE, the average "backing to OS" traffic dropped by 31.96 Gbps (30.05%), and the average user access latency dropped by 28.75 ms (17.07%). Furthermore, the request tail latency was reduced from 1.45s to 0.48s at the 99.9^{th} percentile, or 66.90%. We attribute this result to the judicious eviction of large files by LRU-BaSE. There is a hot event emerging on Day 12 at 10 a.m. Although the "backing to OS" traffic spiked for a moment, it quickly fell back with LRU-BaSE's online learning and decision-making. In addition, since Day 6, Day 7, Day 13, and Day 14 are weekends, there were more requests for the same data, resulting in system congestion and fluctuations in the user access latency during these periods.

Without LRU-BaSE, an administrator might overprovision resources. This involves expanding equipment capacity to manage increased traffic and then removing the excess equipment when traffic decreases. This approach could lead to increased labor and equipment costs. Moreover, as the cache space of TDC is about 3000TB, adding about 100TB of SSD can only achieve a 1% reduction in OMR. As shown in Table VII, LRU-BaSE can reduce OMR and BMR by 24.3% and 32.3%, respectively. While LRU-BASE has better BMR and OMR than LRU, its execution efficiency is lower. Therefore, it achieves nearly the same throughput as LRU. Although LRU-BaSE will consume more of TDC's relatively plentiful CPU resources, it can save \$795,000 per year in bandwidth costs [25], [26].

TABLE VII
IMPORTANT METRICS FOR TDC(LRU) AND TDC(LRU-BASE)

Metric	TDC(LRU)	TDC(LRU-BaSE)
Average OMR	52.5%	28.2%
Average BMR	65.9%	33.6%
99.9^{th} Latency	1.45s	0.48s
Peak CPU	29.01%	36.84%
Peak Memory	14.3GB	15.08GB
Max Throughput	62.5Gbps	61.25Gbps

C. Overhead

We evaluated the effectiveness of LRU-BaSE by measuring its overhead during the same period as Section VI-B.

CPU Utilization: As shown in Fig. 11(c), the average CPU utilization of all devices was 29.41% before the LRU-BaSE was deployed and increased to 36.42% afterward, where the peek CPU utilization observed was 38.77%. Since LRU-BaSE keeps the CPU utilization within 50%, it does not affect the system stability.

Throughput: As shown in Fig. 11(d), the change in average throughput is from 35.41 Mbps to 34.62 Mbps.

Memory: LRU-BaSE uses an extra 751 MB RAM for storing model-related metadata such as network structure parameters and policy parameters. In addition, the extra memory is used to store the learning data comprised of the timestamp (long long int, 8 Bytes), object key (string, 16 Bytes), and size (long int, 4 Bytes). Given the 18 billion inodes in the system, the required capacity is 48 MB. Based on this, we conclude that the total additional memory overhead for LRU-BaSE is 799 MB or 5.94% of the inherent consumption. This additional capacity can only bring less than a 0.001% drop in OMR and BMR when used directly as a memory replenishment cache for LRU.

VII. EVALUATION

A. Evaluation Methodology

Traces and Warmup Traces: Our evaluation uses CDN traces from three CDNs, two of which, Wikipedia [28] and

TABLE VIII
SUMMARY OF THE THREE TRACES THAT ARE USED THROUGHOUT THE EVALUATION

		Tencent	Wikipedia	CDN-Q
Total Requests (Millions)		78.75	268.57	39.71
Unique Object Requested (Millions)		24.71	3.75	15.83
Total Bytes Requested (GB)		3346.72	9219.61	1136.54
Unique Bytes Requested (GB)		1097.63	568.92	349.91
Warm-up Requests (Millions)		40	130	20
Request Object Size	Mean (KB)	46.58	158.97	23.18
	Max (MB)	19.97	674.38	7.99
	Min (B)	2	10	31

CDN-Q [44], are open source, and the third, Tencent is captured from a real-world production system. Each experiment below allows for a warmup period during which no metrics are recorded. The end of the warmup period is defined by the time when BMR is stable. We list the number of warmup requests for different traces in Table VIII.

State-of-the-Art Algorithms: LRU-BaSE is founded on the classic LRU algorithm. It considers the size of objects and employs RL technology to select objects on the Rear Section (RS) for elimination, achieving a win-win situation for BMR and OMR. Therefore, we compare LRU-BaSE with 16 state-of-the-art algorithms, including the classic, size-aware, and machine-learning-based (ML-based) cache algorithms. Note that for the ML-based cache algorithms, the deployment will be modified according to the configuration in related papers combined with the workload in this paper to ensure better results. Results of *Belady* [10] and *PFOO* [12] are deemed as the lower bounds of OMR and BMR, respectively. Although PFOO requires all information about traces in advance and does not always achieve optimal BMR, PFOO has been shown to achieve a lower BMR than others.

Simulator and Testbed: We use the DQN model[1] to implement LRU-BaSE on the LRB simulator.[2] The execution of LRU-BaSE is divided into three parts. We first create a cache environment with a process consisting of parameter initialization, attribute replacement, and reward function feedback. Second, the DQN model deploys the network structure, the learning function, and the replacement interface. Finally, data processing (e.g., the rear section and time regions) and training are implemented. The simulator runs on a device with a 56-core 2.40 GHz CPU, 32 GB of RAM, and a 6 TB SSD. In addition, we use the simulator to integrate the aforementioned state-of-the-art algorithms for fair comparisons. In addition to the parameters listed in Table V, several key parameters exist. First, the value range of the Rear Section (RS), as shown in Fig. 6, is 1% of the Entire Queue (EQ) in the actual environment. Additionally, the starting time S shown in Fig. 10 is 0 a.m., and the interval between the training model (T-i) and the decision-making model (D-i), also shown in Fig. 10, is 1 d. Note that the data scale on the simulator is much smaller than that in the actual environment. To facilitate experimentation, the value of ΔT is set to 1 d.

[1]https://github.com/MorvanZhou/PyTorch-Tutorial/blob/master
[2]https://github.com/sunnyszy/lrb

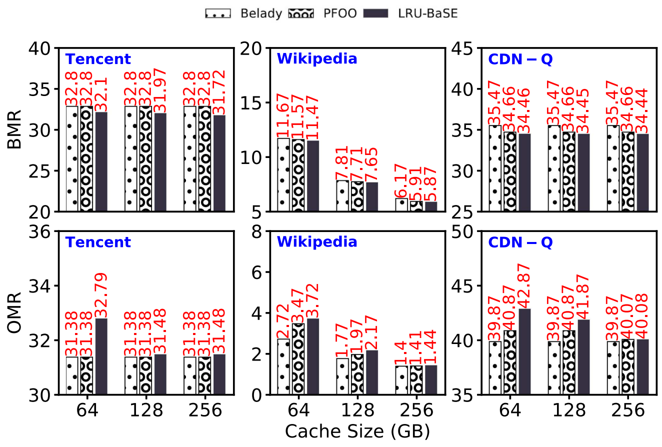


Fig. 12. Comparisons of Belady, PFOO, and LRU-BaSE in BMR and OMR at different cache sizes on three traces.

B. LRU-BaSE Versus Belady and PFOO

As shown in Fig. 12, the BMRs of LRU-BaSE are, by and large, superior to the lower bound, i.e., the BMRs of Belady and PFOO. When the cache size equals 64 GB, the improvement over Belady and PFOO is 0.7% and 0.7%, respectively, on Tencent; 0.2% and 0.1%, respectively, on Wikipedia; and 1% and 0.2%, respectively, on CDN-Q. When the cache size equals 128 GB, the corresponding improvements are 0.8% and 0.8%, 0.2% and 0.06%, 1%, and 0.2%. When the cache size equals 256 GB, the corresponding improvements are 1% and 1%, 0.3% and 0.04%, 1% and 0.2%. In addition, the average OMR of LRU-BaSE over the three cache sizes is 0.53%, 0.48%, and 1.74% higher than that of Belady on Tencent, Wikipedia, and CDN-Q, respectively; is 0.54%, 0.16%, and 1.01% lower than that of PFOO, respectively. We believe that the reason for LRU-BaSE's superiority is twofold. First, LRU-BaSE is extremely close to Belady in reducing BMR by hitting more data. Second, LRU-BaSE selects a BMR-friendly eviction policy while ensuring OMR. Since both Belady and PFOO need all the information about the trace in advance, LRU-BaSE is the first algorithm that dynamically senses the trace to attain a seemingly unattainably low BMR, beyond the lower bound of Belady.

C. LRU-BaSE Versus Classic Algorithms

The classical cache algorithms compared in this section include LRU, S4LRU [8], ARC [48], LRUK [49], LFUDA [50], and S3FIFO [51]. As shown in Fig. 13, in BMR and OMR on two open-source traces (i.e., Wikipedia and CDN-Q), LRU-BaSE is superior to the other six classical algorithms. As shown in Fig. 13, on Wikipedia, S3FIFO yields the optimal performance in the six classical algorithms. When the cache space is 64 GB, the BMR and OMR of LRU-BaSE are reduced by 37.2% and 14.2%, respectively, compared to S3FIFO. As shown in Fig. 13, LFUDA is the best-performing classical algorithm on CDN-Q. LRU-BaSE reduces 21.8% in BMR and 12.1% in OMR compared to LFUDA at 64 GB of cache space. In addition, although the benefits brought by LRU-BaSE gradually decrease with the gradual increase in cache space, it is still superior to all

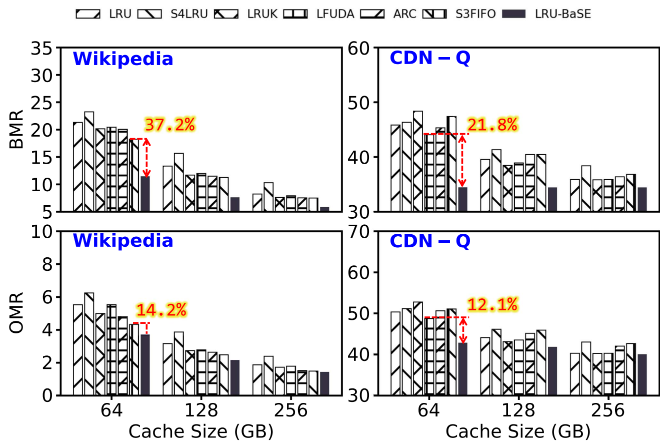


Fig. 13. Comparisons of LRU-BaSE and six classical cache algorithms in terms of BMR and OMR at different cache sizes on three traces.

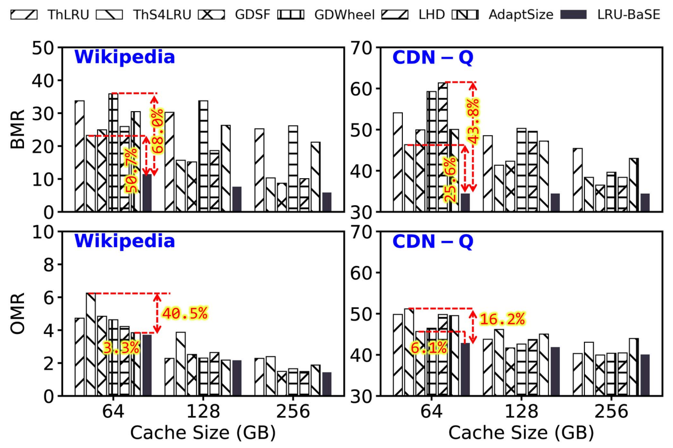


Fig. 15. Comparisons of LRU-BaSE and six size-aware algorithms in terms of BMR and OMR at different cache sizes on three traces.

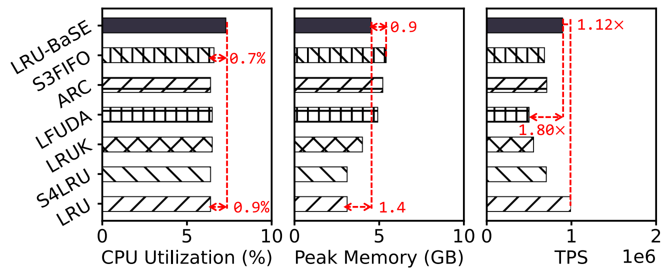


Fig. 14. Comparisons of LRU-BaSE and six classical cache algorithms in terms of peak CPU utilization, peak memory, and transactions per second (TPS) at different cache sizes on three traces.

classical algorithms. Note that TDC provides services for multiple applications mentioned in Section VI-A, and the workloads of different applications are mixed. Due to LRU's inability to adapt to varying workloads, it tends to underperform in mixed workload scenarios. However, in this experiment, we're dealing with a simple workload, which is why LRU's performance is relatively satisfactory.

In our opinion, the reasons for this phenomenon are as follows. First, with the increase of cache space, it is normal for the improvement of the algorithm to reduce gradually. For the simulation test, the size of the working set is certain. The larger the cache space, the more working set data can be cached while the impact of the algorithm becomes less. Second, LFUDA yields the best performance in classical algorithms on CDN-Q because LFUDA is designed based on access frequency. The distribution of requests in the CDN-Q trace is uneven where the difference in the access frequency of different objects in the CDN-Q trace spans 4-5 orders of magnitude. Therefore, LFUDA is more suitable. Finally, the dominance of classical algorithms varies on different data sets, which stems from the fact that various algorithms have distinct sensitivities to the perception of different features in a trace.

In addition, as shown in Fig. 14, in terms of CPU utilization, the result of LRU-BaSE on the simulator is only 0.9% higher than that of LRU. We attribute it to the Rear Section mentioned in Section IV-A-1, which greatly reduces the decision space of reinforcement learning, and the Key-Step Reward Function in Section V-B-2 also greatly reduces the iterative process of reinforcement learning. Although the peak memory is 1.4 GB higher than that of LRU, compared to S3FIFO, the peak memory of LRU-BaSE is 0.9 GB lower. LRU-BaSE's lower memory usage benefits from using only the previous day's data for training and making decisions for the next day's data, as mentioned in Section V-C. Thus, only a small amount of training data needs to be saved. Finally, as shown in Fig. 14, LRU-BaSE's TPS is only lower than that of LRU.

D. LRU-BaSE Versus Size-Aware Algorithms

The size-aware cache algorithms compared in this section include ThLRU [52], ThS4LRU [52], GDSF [53], GDWheel [54], LHD [42], and AdaptSize [4]. As shown in Fig. 15, LRU-BaSE outperforms other size-aware algorithms on two open-source traces. On Wikipedia, when the cache space is 64 GB, among the six size-aware algorithms, the best BMR is yielded by ThS4LRU, while the worst BMR is yielded by GDWheel. Compared to the BMR of ThS4LRU, LRU-BaSE's BMR is reduced by 50.7%, and compared to the BMR of GDWheel, LRU-BaSE's BMR is reduced by 68.0%. In the OMR, the result of LRU-BaSE is further reduced by 3.3% based on the best-performing AdaptSize, and by 40.5% based on the worst-performing ThS4LRU. In addition, on CDN-Q, when the cache space is 64 GB, among the six size-aware algorithms, the best BMR belongs to ThS4LRU, and the worst BMR belongs to LHD. Compared to the BMR of ThS4LRU, LRU-BaSE's BMR is reduced by 25.6%, and compared to the BMR of LHD, LRU-BaSE's BMR is reduced by 43.8%. In the OMR, the result of LRU-BaSE is further reduced by 6.1% based on the best-performing GDSF, and by 16.2% based on the worst-performing ThS4LRU. As shown in Fig. 15, the current size-aware algorithm cannot obtain dominated BMR and OMR at the same time. However, LRU-BaSE can realize the win-win situation between BMR and OMR described in Section III.

As shown in Fig. 16, the CPU utilization of LRU-BaSE approximates that of AdaptSize and others. In terms of memory

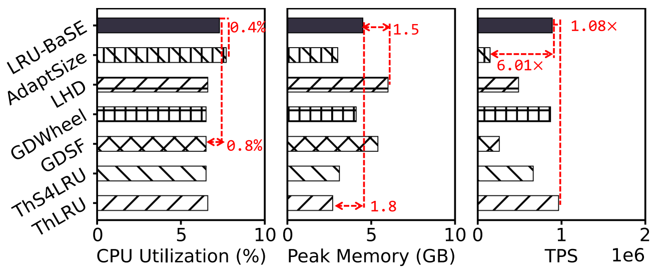


Fig. 16. Comparisons of LRU-BaSE and six size-aware algorithms in terms of peak CPU utilization, peak memory, and transactions per second (TPS) at different cache sizes on three traces.

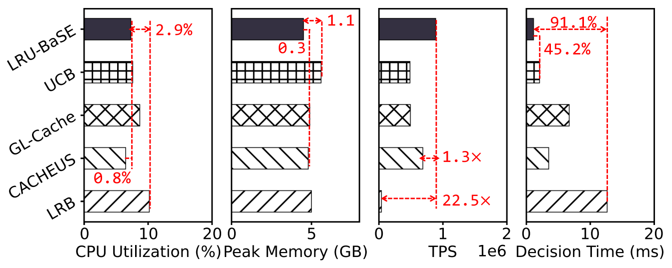


Fig. 18. Comparisons of LRU-BaSE and six ML cache algorithms in terms of peak CPU utilization, peak memory, and transactions per second (TPS) at different cache sizes on three traces.

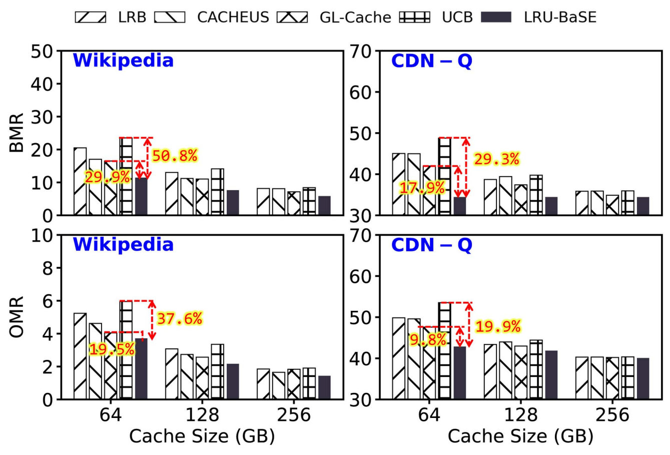


Fig. 17. Comparisons of LRU-BaSE and six ML cache algorithms in terms of BMR and OMR at different cache sizes on three traces.

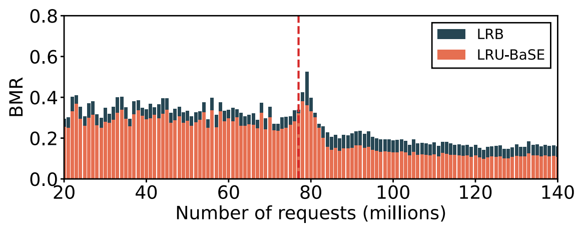


Fig. 19. Compare the robustness of LRB and LRU-BaSE facing workload drift.

consumption, LRU-BaSE is moderate among the compared methods. The peak memory is 1.8 GB higher than that of ThLRU and 1.5 GB lower than that of LHD. As shown in Fig. 16, LRU-BaSE is second only to ThLRU in transaction throughput.

E. LRU-BaSE Versus ML Cache Algorithms

The cache algorithms based on machine learning include LRB [28], CACHEUS [40] (the updated version of LeCaR [21]), GL-Cache [55], and UCB [56]. As shown in Fig. 17, LRU-BaSE outperforms the ML-based algorithms in both BMR and OMR. On Wikipedia, when the cache space is 64 GB, among the six ML algorithms, the best BMR is yielded by GL-Cache, while the worst one is yielded by UCB. Compared to the BMR of GL-Cache, LRU-BaSE's BMR is reduced by 29.9%, and compared to the BMR of UCB, LRU-BaSE's BMR is reduced by 50.8%. In the OMR, the result of LRU-BaSE is further reduced by 19.5% based on the best-performing GL-Cache, and by 37.6% based on the worst-performing UCB. In addition, on CDN-Q, compared to the BMR of GL-Cache, LRU-BaSE's BMR is reduced by 17.9%, and compared to the BMR of UCB, LRU-BaSE's BMR is reduced by 29.3%. In the OMR, the result of LRU-BaSE is further reduced by 9.8% based on the best-performing GL-Cache, and by 19.9% based on the worst-performing UCB.

In terms of peak CPU utilization, TPS, and peak memory consumption, LRU-BaSE is the best. We attribute this improvement to two optimizations of LRU-BaSE. First, the decision space is reduced by the rear section. Second, the cache requirements for speed can be met by parallelizing the batch processing in the implementation. In addition, since model training and decision-making are performed on different threads in parallel, LRU-BaSE calculates only once when making a decision. As shown in Fig. 18, LRU-BaSE yields the shortest decision time.

F. Robustness of LRU-BaSE

In this section, we show the robustness of LRU-BaSE by showing that our window-finding method is more resilient to workload drift than the pre-defined boundary method in LRB. We splice two traces (Tencent and Wikipedia from Table VIII) with large variability to form a new trace. On this new trace, we run LRB and LRU-BaSE and output the average BMR of each one million requests. As shown in Fig. 19, data on the left of the red dashed line represent BMRs on Tencent, and the workload is changed to Wikipedia at the red dashed line, with the ensuing BMRs shown on the right of the red dashed line. According to the statistics, LRU-BaSE outperforms LRB by 3.9% on average before the workload drift, while outperforming LRB by 5.5% on average after the drift. Referring to their results, we believe that LRB's customized window for Tencent inhibits its performance on Wikipedia, while LRU-BaSE can dynamically adapt to changes in workloads. Furthermore, LRU-BaSE exhibits smaller fluctuations at and right after the onset of the workload change.

VIII. Related Work

The optimal cache replacement policy is to evict the objects with the longest reuse distance if these requests and their orders are known a priori, also known as Belady's algorithm [10], [11].

Cache Replacement Algorithms Based on Belady: Heuristics, often used in cache replacement strategies, estimate the object with the greatest reuse distance by making assumptions about specific properties. LRU and some of its variants, such as 2Q [57], CFLRU [58], and LAMA [59], deal with requests by recency. LFU and some schemes [16], [60], [61] rely on frequency. Other common approaches [8], [48], [49], [53], [62], [63], [64], [65], [66], [67], [68] fuse these assumptions with multiple queues. The disadvantage of Heuristics is hindsight, especially for CDN caching with large changes in requests. Conversely, learning-based algorithms are adept at prediction and can fit the object with the longest reuse distance. The SVM-LRU [69] employed a support vector machine (SVM) model. PopCaching [70] learns the popularity of the content using the clustering model. Learning From OPT (LFO) [24] learns whether an object should be cached by explicitly modeling optimal caching decisions. LRB [28] uses Gradient Boosting Machines (GBM) to approximate the Belady MIN algorithm using the relaxed Belady boundary. These algorithms suffer from low generality due to the limited generalization capability. In addition, reinforcement learning (RL) models fit the decision-making process of eviction with classic assumptions. LHR [32] leverages online approximate optimal caching to inform future content admission and eviction. UCB [56] learns access patterns. LeCaR [21] and CACHEUS [40] explore the switch pattern between various classic algorithms. In a real-world situation, they usually incur high overhead. Other deep RL-based cache algorithms [71], [72], [73] assume that no new objects are accessed and all objects have the same size.

Cache Algorithms With Object Sizes: In addition to the cache methods that use the object size as a feature to approximate Belady, some web cache algorithms notice the gap between OMR and BMR and try to use the object size for the BMR reduction. Abrams et al. [7], proposed that it is better to hold many small documents than a few large documents. Then, they [13] discussed "the number of bytes not sent" for the replacement policies. Then, the GreedyDual-Size [15] algorithm noted the eviction cost, where object sizes are first associated with BMR. The GreedyDual* [18] gives more pieces of evidence that BMR and OMR have different gaps at different cache sizes. The algorithms informed by object sizes are also discussed in [19]. W-TinyLFU [31] explores computationally efficient size-aware cache admission policies. Despite the seemingly random gap between OMR and BMR, these researchers are exclusively interested in OMR reduction measures. This gap is amplified in CDNs. Furthermore, Belady is no longer able to provide the optimal BMR. This implies that Belady alone is not sufficient for reducing BMR, and there is still room for reducing BMR while preserving OMR.

IX. Conclusion and Discussion

We believe that approximating Belady alone no longer provides enough room for lowering BMR, as the approximation for the reuse distance is close to the limit. Our work represents a first step towards comprehending the relationship between OMR and BMR and attempting to decrease BMR while maintaining OMR. Our discovery of the "flat" regions in the OMR (eviction-window) function curves offers theoretical possibilities for solutions with win-win BMR-OMR performance. Furthermore, we improve the RL model based on the unique traits of CDN and LRU to tackle the feedback delay problem. Nonetheless, this is also a compromise solution due to the inability to find evident features corresponding to BMR. In our future work, we plan to analyze the most influential features of the choices that favor BMR to inform decisions for simple learning models.

References

[1] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52–66, 2015.

[2] M. K. Mukerjee, I. N. Bozkurt, B. Maggs, S. Seshan, and H. Zhang, "The impact of brokers on the future of content delivery," in *Proc. ACM Workshop Hot Top. Netw.*, 2016, pp. 127–133.

[3] B. Berg et al., "The cachelib caching engine: Design and experiences at scale," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 753–768.

[4] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "AdaptSize: Orchestrating the hot object memory cache in a content delivery network," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 483–498.

[5] L. Peled, U. C. Weiser, and Y. Etsion, "A neural network prefetcher for arbitrary memory access patterns," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, pp. 37:1–37:27, 2020.

[6] L. Peled, S. Mannor, U. C. Weiser, and Y. Etsion, "Semantic locality and context-based prefetching using reinforcement learning," in *Proc. ACM/IEEE Annu. Int. Symp. Comput. Archit.*, 2015, pp. 285–297.

[7] M. Abrams, C. R. Standridge, G. Abdulla, S. M. Williams, and E. A. Fox, "Caching proxies: Limitations and potentials," *World Wide Web*, vol. 1, no. 1, pp. 119–133, 1996.

[8] Q. Huang, K. Birman, R. Van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of Facebook photo caching," in *Proc. ACM Symp. Operating Syst. Princ.*, 2013, pp. 167–181.

[9] O. Eytan, D. Harnik, E. Ofer, R. Friedman, and R. Kat, "It's time to revisit LRU versus FIFO," in *Proc. USENIX Workshop Hot Top. Storage File Syst.*, 2020, pp. 12–12.

[10] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, 1966.

[11] A. Jain and C. Lin, "Rethinking belady's algorithm to accommodate prefetching," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 110–123.

[12] D. S. Berger, N. Beckmann, and M. Harchol-Balter, "Practical bounds on optimal caching with variable object sizes," *ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 2, pp. 1–38, 2018.

[13] M. Abrams, C. R. Standridge, G. Abdulla, E. A. Fox, and S. Williams, "Removal policies in network caches for world-wide web documents," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, 1996, pp. 293–305.

[14] R. P. Wooster and M. Abrams, "Proxy caching that estimates page load delays," *Elsevier Comput. Netw. ISDN Syst.*, vol. 29, no. 8–13, pp. 977–986, 1997.

[15] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proc. USENIX Symp. Internet Technol. Syst.*, vol. 12, no. 97, 1997, pp. 193–206.

[16] C. Aggarwal, J. L. Wolf, and P. S. Yu, "Caching on the world wide web," *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 1, pp. 94–107, Jan./Feb. 1999.

[17] L. Rizzo and L. Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Trans. Netw.*, vol. 8, no. 2, pp. 158–170, Apr. 2000.

[18] S. Jin and A. Bestavros, "GreedyDual* web caching algorithm: Exploiting the two sources of temporal locality in web request streams," *Elsevier Comput. Commun.*, vol. 24, no. 2, pp. 174–183, 2001.

[19] H. Bahn, K. Koh, S. H. Noh, and S. Lyul, "Efficient replacement of nonuniform objects in web caches," *IEEE Comput.*, vol. 35, no. 6, pp. 65–73, Jun. 2002.

[20] P. Li et al., "Beating OPT with statistical clairvoyance and variable size caching," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 243–256.

[21] G. Vietri et al., "Driving cache replacement with ML-based LeCaR," in *Proc. USENIX Workshop Hot Top. Storage File Syst.*, 2018, pp. 3–3.

[22] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[23] S. Sethumurugan, J. Yin, and J. Sartori, “Designing a cost-effective cache replacement policy using machine learning,” in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2021, pp. 291–303.
[24] D. S. Berger, “Towards lightweight and robust machine learning for CDN caching,” in *Proc. ACM Workshop Hot Top. Netw.*, 2018, pp. 134–140.
[25] D. L.-K. Wong et al., “Baleen:{ML} admission & prefetching for flash caches,” in *Proc. USENIX Conf. File Storage Technol.*, 2024, pp. 347–371.
[26] S. Pan et al., “Facebook’s tectonic filesystem: Efficiency from exascale,” in *Proc. USENIX Conf. File Storage Technol.*, 2021, pp. 217–231.
[27] B. Cai et al., “HUNTER: An online cloud database hybrid tuning system for personalized requirements,” in *Proc. ACM Int. Conf. Manage. Data*, 2022, pp. 646–659.
[28] Z. Song, D. S. Berger, K. Li, and W. Lloyd, “Learning relaxed belady for content distribution network caching,” in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2020, pp. 529–544.
[29] N. Gast and B. V. Houdt, “Transient and steady-state regime of a family of list-based cache replacement algorithms,” in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2015, pp. 123–136.
[30] P. Wang, Y. Liu, Z. Zhao, K. Zhou, Z. Huang, and Y. Chen, “Adaptive size-aware cache insertion policy for content delivery networks,” in *Proc. IEEE 40th Int. Conf. Comput. Des.*, 2022, pp. 195–202.
[31] G. Einziger, O. Eytan, R. Friedman, and B. Manes, “Lightweight robust size aware cache management,” *ACM Trans. Storage*, vol. 18, pp. 1–23, 2021.
[32] G. Yan, J. Li, and D. Towsley, “Learning from optimal caching for content delivery,” in *Proc. ACM Int. Conf. Emerg. Netw. EXperiments Technol.*, 2021, pp. 344–358.
[33] D. S. Berger, B. Berg, T. Zhu, S. Sen, and M. Harchol-Balter, “Robinhood: Tail latency aware caching–dynamic reallocation from cache-rich to cache-poor,” in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2018, pp. 195–212.
[34] T. Zhu, M. A. Kozuch, and M. Harchol-Balter, “Workloadcompactor: Reducing datacenter cost while providing tail latency SLO guarantees,” in *Proc. ACM Symp. Cloud Comput.*, 2017, pp. 598–610.
[35] N. Atre, J. Sherry, W. Wang, and D. S. Berger, “Caching with delayed hits,” in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, 2020, pp. 495–513.
[36] G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3146–3154.
[37] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *Ann. Statist.*, vol. 29, pp. 1189–1232, 2001.
[38] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[39] A. Vaswani et al., “Attention is all you need,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[40] L. V. Rodriguez et al., “Learning cache replacement with cacheus,” in *Proc. USENIX Conf. File Storage Technol.*, 2021, pp. 341–354.
[41] V. Mnih et al., “Playing atari with deep reinforcement learning,” 2013, *arXiv:1312.5602*.
[42] N. Beckmann, H. Chen, and A. Cidon, “LHD: Improving cache hit rate by maximizing hit density,” in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2018, pp. 389–403.
[43] J. S. Vitter, “Random sampling with a reservoir,” *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.
[44] K. Zhou et al., “Demystifying cache policies for photo stores at scale: A tencent case study,” in *Proc. ACM Int. Conf. Supercomput.*, 2018, pp. 284–294.
[45] V. Mnih et al., “Asynchronous methods for deep reinforcement learning,” in *Proc. ACM Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[46] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” 2015, *arXiv:1509.02971*.
[47] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.
[48] N. Megiddo and D. S. Modha, “ARC: A self-tuning, low overhead replacement cache,” in *Proc. USENIX Conf. File Storage Technol.*, 2003, pp. 115–130.
[49] E. J. O’neil, P. E. O’neil, and G. Weikum, “The LRU-K page replacement algorithm for database disk buffering,” *ACM SIGMOD Rec.*, vol. 22, no. 2, pp. 297–306, 1993.
[50] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, “Evaluating content management techniques for web proxy caches,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 27, no. 4, pp. 3–11, 2000.
[51] J. Yang, Y. Zhang, Z. Qiu, Y. Yue, and R. Vinayak, “FIFO queues are all you need for cache eviction,” in *Proc. ACM Symp. Operating Syst. Princ.*, 2023, pp. 130–149.
[52] G. Einziger, R. Friedman, and B. Manes, “TinyLFU: A highly efficient cache admission policy,” *ACM Trans. Storage*, vol. 13, no. 4, pp. 1–31, 2017.
[53] L. Cherkasova and G. Ciardo, “Role of aging, frequency, and size in web cache replacement policies,” in *Proc. Int. Conf. High-Perform. Comput. Netw.*, 2001, pp. 114–123.
[54] C. Li and A. L. Cox, “GD-Wheel: A cost-aware replacement policy for key-value stores,” in *Proc. ACM Eur. Conf. Comput. Syst.*, 2015, pp. 1–15.
[55] J. Yang, Z. Mao, Y. Yue, and K. Rashmi, “GL-Cache: Group-level learning for efficient and high-performance caching,” in *Proc. USENIX Conf. File Storage Technol.*, 2023, pp. 115–134.
[56] R. Costa and J. Pazos, “MLCache: A multi-armed bandit policy for an operating system page cache,” University of British Columbia, Tech. Rep., 2017.
[57] T. Johnson et al., “2Q: A low overhead high performance bu er management replacement algorithm,” in *Proc. ACM Int. Conf. Very Large Data Bases*, 1994, pp. 439–450.
[58] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee, “CFLRU: A replacement algorithm for flash memory,” in *Proc. ACM/IEEE Int. Conf. Compilers, Archit. Synth. Embedded Syst.*, 2006, pp. 234–241.
[59] X. Hu et al., “LAMA: Optimized locality-aware memory allocation for key-value cache,” in *Proc. USENIX Annu. Tech. Conf.*, 2015, pp. 57–69.
[60] G. Karakostas and D. N. Serpanos, “Exploitation of different types of locality for web caches,” in *Proc. IEEE 7th Int. Symp. Comput. Commun.*, 2002, pp. 207–212.
[61] D. Matani, K. Shah, and A. Mitra, “An O(1) algorithm for implementing the LFU cache eviction scheme,” 2021, *arXiv:2110.11602*.
[62] D. Lee et al., “On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies,” in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 1999, pp. 134–143.
[63] S. Jiang and X. Zhang, “LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 31–42, 2002.
[64] S. Bansal and D. S. Modha, “CAR: Clock with adaptive replacement,” in *Proc. USENIX Conf. File Storage Technol.*, 2004, pp. 187–200.
[65] S. Jiang, F. Chen, and X. Zhang, “CLOCK-Pro: An effective improvement of the clock replacement,” in *Proc. USENIX Annu. Tech. Conf.*, 2005, pp. 323–336.
[66] S. Park and C. Park, “FRD: A filtering based buffer cache algorithm that considers both frequency and reuse distance,” in *Proc. IEEE Int. Conf. Massive Storage Syst. Technol.*, 2017, pp. 1–12.
[67] C. Li, “DLIRS: Improving low inter-reference recency set cache replacement policy with dynamics,” in *Proc. ACM Int. Conf. Massive Storage Syst. Technol.*, 2018, pp. 59–64.
[68] C. Zhong, X. Zhao, and S. Jiang, “LIRS2: An improved LIRS replacement algorithm,” in *Proc. ACM Int. Conf. Syst. Storage*, 2021, pp. 1–12.
[69] W. Ali, S. M. Shamsuddin, and A. S. Ismail, “Intelligent web proxy caching approaches based on machine learning techniques,” *Elsevier Decis. Support Syst.*, vol. 53, no. 3, pp. 565–579, 2012.
[70] S. Li, J. Xu, M. Van Der Schaar, and W. Li, “Popularity-driven content caching,” in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[71] A. Sadeghi, G. Wang, and G. B. Giannakis, “Deep reinforcement learning for adaptive caching in hierarchical content delivery networks,” *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 4, pp. 1024–1033, Dec. 2019.
[72] S. Alabed, “RLCache: Automated cache management using reinforcement learning,” 2019, *arXiv: 1909.13839*.
[73] G. Yan and J. Li, “Rl-Bélády: A unified learning framework for content caching,” in *Proc. ACM Int. Conf. Multimedia*, 2020, pp. 1009–1017.

Peng Wang received the PhD degree in computer science from Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China. He has published papers in international journals and conferences, including ICCD, ICPP, IJCAI, ACM TACO, *IEEE Transactions on Cloud Computing*, *etc*.

Hong Jiang (Fellow, IEEE) received the PhD degree in computer science from the Texas A&M University, College Station, Texas, USA. He is currently chair and Wendell H. Nedderman Endowed professor of Computer Science and Engineering Department with the University of Texas at Arlington. Prior to joining UTA, he served as a program director with National Science Foundation (2013–2015) and he was with the University of Nebraska-Lincoln since 1991, where he was Willa Cather professor of Computer Science and Engineering. He has graduated 17 PhD students who upon their graduation either landed academic tenure-track positions in PhD-granting US institutions or were employed by major US IT corporations. He has also supervised 20 postdoctoral fellows and visiting scholars. He is currently supervising/co-supervising more than 10 PhD students and postdoc fellows. His current research interests include computer architecture, computer storage systems, and parallel I/O, high-performance computing, Big Data computing, cloud computing, performance evaluation. He is a topic and associate editor of *IEEE Transactions on Computers* and recently served as an associate editor of *IEEE Transactions on Parallel and Distributed Systems*. He has more than 300 publications in major journals and international Conferences in these areas, including *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *Proceedings of IEEE*, *ACM Transactions on Architecture and Code Optimization*, *ACM Transactions on Storage*, USENIX ATC, FAST, EUROSYS, ISCA, MICRO, SOCC, LISA, SIGMETRICS, ICDE, DAC, DATE, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by NSF and industry. He is a member of ACM.

Yu Liu (Member, IEEE) received the PhD degree in computer science from Huazhong University of Science and Technology (HUST) in 2017. Then, he was a postdoctoral researcher with HUST from 2018 to 2021. Currently, he is an associate researcher with the School of Computer Science and Technology, HUST. His current research focuses on building cognitive storage systems with machine learning technologies and large-scale multimedia search technologies. He has published papers in international journals and conferences, including ACM MM, IJCAI, SIGMOD, DAC, *IEEE Transactions on Image Processing*, *IEEE Transactions on Cybernetics*, *IEEE Transactions on Multimedia*, CIKM, *etc*.

Zhelong Zhao received the BE degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2022, where he is currently working toward the master's degree.

Ke Zhou (Member, IEEE) is a professor with the School of Computer Science and Technology, HUST. His research interests include computer architecture, cloud storage, parallel I/O, and storage security. He has more than 50 publications in journals and international conferences, including *IEEE Transactions on Parallel and Distributed Systems*, A Page Endurance Variance Aware, SIGMOD, FAST, USENIX ATC, MSST, ACM MM, INFOCOM, SYSTOR, MASCOTS, ICC, etc. He is a member of the USENIX.

Zhihai Huang received the postgraduate degree from the Nanjing University of Posts and Telecommunications, China. He is currently an expert engineer with Tencent Corporation. Since joining Tencent, he has specialized in optimizing the storage, delivery, and processing of massive social media data. His main research interests include media storage, image processing, video transcoding, and real-time communications.