Smart Cache Insertion and Promotion Policy for Content Delivery Networks

Peng Wang
WNLO, Huazhong University of Science and Technology
Wuhan, China
wp_hust@hust.edu.cn

Yu Liu*
Huazhong University of Science and Technology
Wuhan, China
liu_yu@hust.edu.cn

Zhelong Zhao
Huazhong University of Science and Technology
Wuhan, China
zhelongzhao@hust.edu.cn

Ke Zhou
WNLO, Huazhong University of Science and Technology
Wuhan, China
zhke@hust.edu.cn

Zhihai Huang
Tencent Technology (Shenzhen) Co., Ltd.
Shenzhen, China
tommyhuang@tencent.com

Yanxiong Chen
Tencent Technology (Shenzhen) Co., Ltd.
Shenzhen, China
yxbillchen@tencent.com

ABSTRACT

Improving hit rates can be achieved by enhancing cache replacement algorithms with the identification of zero-reuse objects (ZROs) and inserting them at the end of the cache queue. Note that the promotion policy needs to achieve a similar task as the above insertion policy since the hit object may immediately become a ZRO (called P-ZRO) that is not suitable for placement at the front of the queue. However, existing studies have yet to consider P-ZROs, and current insertion algorithms struggle to simultaneously identify both ZROs and P-ZROs. To address these issues, we propose integrating the insertion and promotion policies. We do this by treating hit objects as special missing objects and employing reinforcement learning to create a unified model for both policies, where the learning function recognizes the relationship between performance changes and the emergence of ZROs and P-ZROs. Our proposed solution is a smart cache insertion and promotion policy (SCIP) that dynamically adjusts the insertion position using a bimodal insertion policy for both missing and hit objects, guided by the model. Extensive experiments demonstrate that SCIP significantly improves overall performance in real-world content delivery network systems and outperforms state-of-the-art insertion policies in terms of miss ratios in the simulator. In addition, deploying SCIP on optimal cache replacement algorithms can further decrease their miss ratios.

CCS CONCEPTS

• **Computer systems organization** → **Cache**; **Cache insertion policy**; • **Networks** → **Content delivery network**.

*Yu Liu is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICPP 2023, August 07–10, 2023, Salt Lake City, UT, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0843-5/23/08…$15.00
https://doi.org/10.1145/3605573.3605581

KEYWORDS

content delivery network, replacement algorithm, insertion policy

ACM Reference Format:
Peng Wang, Yu Liu, Zhelong Zhao, Ke Zhou, Zhihai Huang, and Yanxiong Chen. 2023. Smart Cache Insertion and Promotion Policy for Content Delivery Networks. In *52nd International Conference on Parallel Processing (ICPP 2023), August 07–10, 2023, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3605573.3605581

1 INTRODUCTION

Content delivery networks (CDNs) are network caches designed to deliver texts, images, videos, and websites from multiple sources to users. A critical factor in ensuring optimal CDN performance is improving the cache hit rate, which has led to extensive research on cache replacement algorithms. In general, a cache replacement algorithm consists of two primary components: the victim selection policy and the insertion policy. The victim selection policy [12, 14, 17, 27] focuses on determining which object to remove from the cache when space is needed, while the insertion policy determines the position of a newly accessed object within the cache. Although these two policies are interconnected, the insertion policy plays an additional role in controlling the data environment of the victim selection policy, explicitly influencing the order of objects in the cache.

This paper focus on the insertion policy and its interaction with zero-reuse objects (ZROs) [25]. ZROs will not be accessed as long as they appear in the cache. When an insertion policy places a ZRO in the head of the cache queue, *i.e.*, the MRU (Most Recently Used) position [17], the LRU (Least Recently Used) -like victim policy will keep it in the cache until it has traversed the entire cache queue. As shown in Figure 1(a) and Figure 1(b), there are high miss ratios on the CDN-A workload due to a large number of ZROs occupying cache space for extended periods of time, where Figure 1(a) shows the proportion of ZROs in missing objects. When we accurately identify the ZROs and place them in the LRU position, the percentage reduction in the miss ratio equals to portion covering by slashes. Note that we cannot classify objects as ZROs in the same way we label one-time-access objects [24]. The one-time-access object enters the cache at most once, while a ZRO may enter the cache multiple times. This means that ZRO is not a fixed

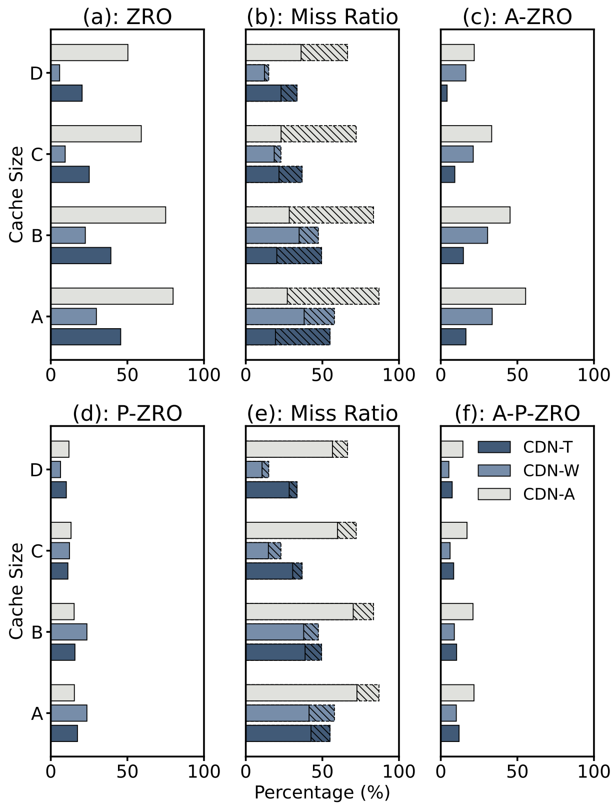


Figure 1: The percentage of ZROs, A-ZROs, P-ZROs, A-P-ZROs, and the miss ratios under different cache sizes, where the portion covering the slash represents the percentage that can be reduced. $A = 0.5\% \times X, B = 1\% \times X, C = 5\% \times X$, **and** $D = 10\% \times X$, **where** X **equals 1097GB, 327GB, and 1580GB for CDN-T, CDN-W, and CDN-A, respectively. CDN-T, CDN-W, and CDN-A are workloads. The caching policy is LRU.**

property of an object. An object that was once a ZRO may be accessed in the cache in the future. We refer to these ZROs as A-ZROs. Figure 1(c) illustrates the proportion of A-ZROs in ZROs for different workloads.

The adaptive size-aware cache insertion policy (ASC-IP) [25] is the only caching policy that takes ZROs into account and uses heuristics to insert objects into the MRU or LRU position. Nevertheless, ASC-IP fails to consider that objects may immediately degrade to ZROs after being hit in the cache. Its insertion policy only detects missing objects and promotes all hit objects to the MRU position. In this paper, we refer to these hit objects as P-ZROs. Figure 1 (d) shows the proportion of P-ZROs in hit objects, where the CDN-W workload [22] has the highest percentage of P-ZROs, with an average of 21.7% of the hit objects falling under this category. Accurately predicting these P-ZROs and placing them in the LRU position after being hit also reduce the miss ratio. As shown in Figure 1 (e), the bar covering the slash indicates the decreased percentage after treating P-ZROs. On the CDN-W workload, the average percentage of miss ratios decreases by 9.68%. In addition, similar to A-ZROs, some P-ZROs will degrade to A-P-ZROs. We show the proportion of A-P-ZROs in P-ZROs for different workloads in Figure 1 (f).

To design a complete insertion policy, we take into account both ZROs and P-ZROs, considering the insertion positions in relation to both missing and hit objects. Note that Promotion/Insertion Pseudo-Partitioning (PIPP) [27] is designed for both policies, where its promotion policy moves the hit object one unit towards the MRU position. Nevertheless, this still leads to long stays for P-ZROs in the cache, given that CDN cache sizes are typically large. Since P-ZROs after being hit will share the same characteristic as ZROs, *i.e.*, they won't be accessed in the cache, we recommend inserting ZROs and promoting P-ZROs close to the LRU position. As a result, we employ the bimodal insertion policy (BIP) [17] to implement our insertion and promotion policies, which has an $O(1)$ time complexity without the need to traverse linked lists. In addition, BIP ensures that suspected ZROs and P-ZROs are given a chance to be accessed, thereby reconciling possible misjudgments [25].

Although ZROs and P-ZROs share similar characteristics, they are challenging to identify using a unified pattern. Traditional methods, like ASC-IP, adjust the threshold to detect ZROs when the evicted object has never been hit. However, since P-ZROs exhibit hit properties, the mechanism used to identify ZROs cannot be applied to perceive P-ZROs. Additionally, traditional heuristic methods often introduce errors due to the imbalance between the number of missing and hit objects, as the heuristic tends to favor the side with a large number of missing and hit objects. Figure 1 presents the proportions of ZROs and P-ZROs among missing and hit objects, respectively, as well as the miss ratios. To address these challenges, we leverage reinforcement learning to explore the relationship between performance changes and the insertion position. We treat promotion as a special insertion form and develop a unified model through learning.

We use two shadow cache lists, *i.e.*, H_m and H_l, to record the metadata of evicted objects that were inserted into the MRU and LRU positions of the real cache, respectively. The probability of insertion position is adjusted based on hit rates in the shadow caches for the objects to be inserted. If the object is hit in H_m/H_l, the probability of insertion into the MRU/LRU position is increased. Since the sum of the selection probabilities of the two positions is 1, we employ the Multi-Armed Bandit (MAB) model and design the function to relate the selection probability and hit rates. We learn the parameters of the function based on a gradient-based stochastic hill-climbing method. Our proposed smart cache insertion and promotion policy, *i.e.*, SCIP, dynamically adjusts the insertion position of both missing objects and hit objects. SCIP was implemented in the system of Tencent, *i.e.*, TDC, and in the simulator used in Learning Relaxed Belady (LRB) [22]. In TDC, the miss ratio decreased from 8.87% to 6.59%, the back-to-source bandwidth decreased by 25.7%, and the average user access latency decreased by 26.1% after we deployed SCIP (See §5.2). Compared to state-of-the-art insertion policies, SCIP improves hit ratios on three workloads, respectively, at negligible additional overhead (See §6.3). In addition, SCIP can enhance state-of-the-art caching algorithms and improve their performance as a generic component (See §6.4). The contributions are summarized as follows:

- We revealed P-ZROs and proposed integrating insertion and promotion policies by unifying the treatment of ZROs and P-ZROs.

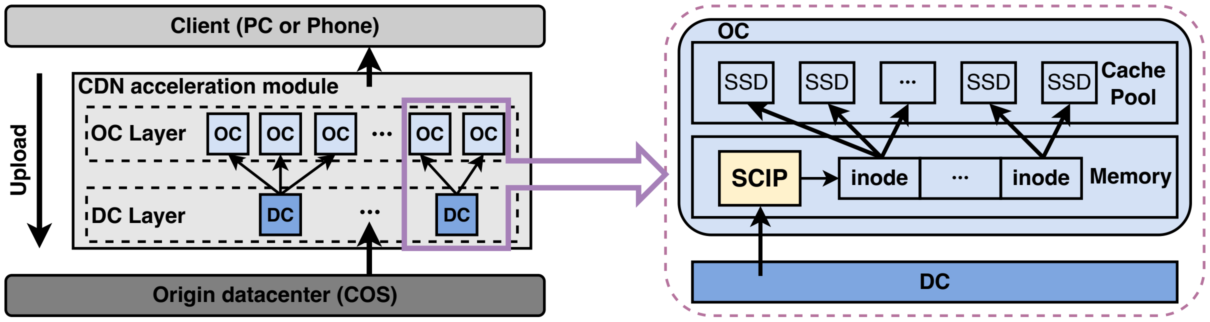


Figure 2: The cache architecture of TDC. The cache layer consists of the data center cache (DC) layer and the outside cache (OC) layer. Note that the SCIP component is our complement to TDC.

• SCIP is the first policy to unify the insertion and promotion policies using a reinforcement learning model while ensuring execution efficiency in CDNs.
• Our model can improve caching performance in the real system with the LRU's victim selection policy. In the simulator, it can further decrease miss ratios of optimal cache replacement algorithms by enhancing them.

2 BACKGROUND AND MOTIVATION

2.1 Background

TDC is Tencent's cloud service product, which is a specialized content delivery network (CDN). Various applications connect to TDC to share image access with a quality of experience (QoE). The CDN acceleration module is the cornerstone of TDC, consisting of the data center cache (DC) layer and the outside cache (OC) layer. The OC layer is closer to the user and can more easily enhance the user experience. The DC layer is located within the data center to reduce congestion on the storage system (COS). We present the architecture of TDC in Figure 2. Even a 1% increase in cache hit rate for TDC can save Tencent up to $1,500,000 annually. However, the current replacement algorithms appear to have hit a bottleneck. We argue that these policies focus solely on victim selection policies and ignore the object order before selection, as well as the reasons behind the formation of that order. Some objects may appear in the MRU position but never get reused. This presents a challenge for victim selection policies, which we aim to address by exploring the objects that appeared in the MRU position.

2.2 Integration of insertion and promotion policies

The insertion and promotion policies prefer to insert objects in the MRU position, as recently accessed objects are more likely to be reused. However, ZROs and P-ZROs that do not follow the access locality principle will remain in the cache for an extended period. As a result, the identification and placement of ZROs and P-ZROs using a policy integrating insertion and promotion policies is a practical addition to current caching policies. However, we still have concerns that placing ZROs and P-ZROs in the LRU position may change the following ZROs or P-ZROs, resulting in biases in our judgments based on access replay and unsatisfactory results. To

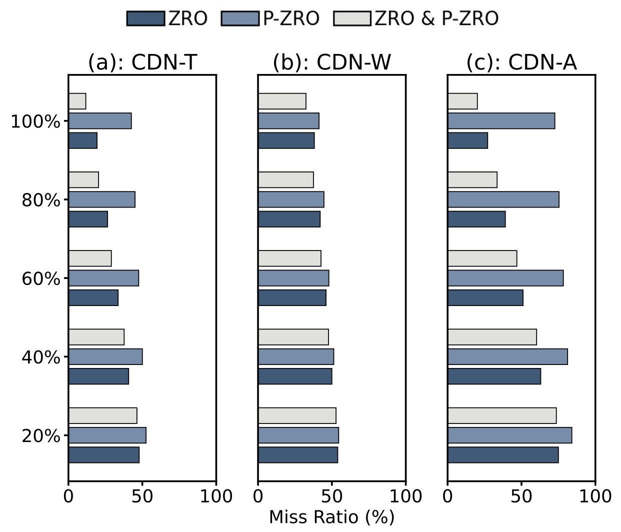


Figure 3: Based on LRU, the theoretical miss ratios yielded by inserting ZRO, P-ZRO, and both ZRO and P-ZRO to the LRU position on three workloads.

validate the actual effect, we conducted the LRU position placement experiments on different numbers of ZROs and P-ZROs. The results are shown in Figure 3, where the x-axis represents the percentages of ZROs, P-ZROs, or both ZROs and P-ZROs at the top of the access sequence. Obviously, the missing ratio shows a monotonic decreasing trend as the number of ZROs, P-ZROs, or both of them decreases. This indicates that benefits can be obtained by dealing with ZROs and P-ZROs in access replay. Let $\mathtt{MR}(X)$ represent the missing ratio when placing X in the LRU position. On the same scale, MR(ZROs) is always less than MR(P-ZROs), and MR(ZROs+P-ZROs) is always less than MR(ZROs) or MR(P-ZROs). Referring to the proportion of ZROs and P-ZROs in the missing objects and hit objects, respectively, we believe that processing more ZROs and P-ZROs can bring greater benefits. In addition, according to the miss ratio yielded by the LRU algorithm, denoting MR_LRU, we find that (MR_LRU-MR(ZROs))+(MR_LRU-MR(P-ZROs)) is always bigger than MR_LRU-MR(ZROs+P-ZROs). This proves our conjecture

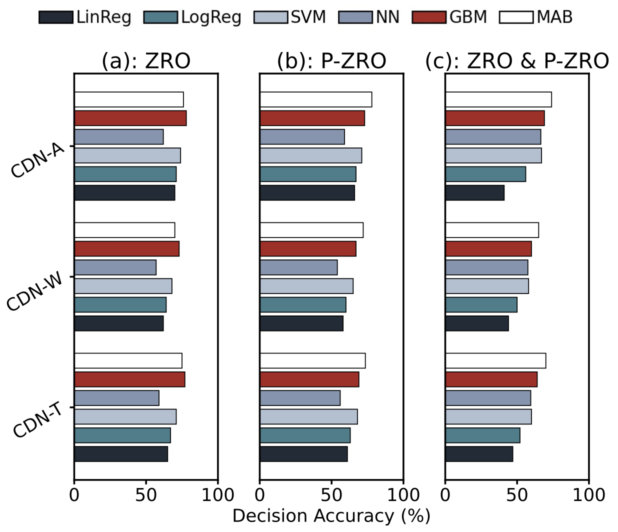


Figure 4: Comparison of decision accuracy of different intelligent algorithms in ZRO, P-ZRO, and both ZRO and P-ZRO.

that changing the insertion positions of the ZROs or P-ZROs will change the subsequent ZROs and P-ZROs. As a result, the identification and insertion of ZROs and P-ZROs are mutually influential. This job would entail knowing how to accurately identify both ZROs and P-ZROs based on access replay.

2.3 Identify both ZROs and P-ZROs by Multi-Armed Bandit

ASC-IP [25] utilizes a heuristic pattern to perceive the relationship between object size and ZROs when the evicted object's hit token equals *False*. However, as the P-ZRO's hit token always equals *True*, we cannot determine P-ZROs. Moreover, since the plain heuristic deals with each insertion individually and the amounts of missing and hit objects are different, the size will favor the side that makes more judgments, resulting in misjudgment [25]. This issue also affects the prediction accuracy of machine learning models. We trained the linear regression model (LinReg) [1], logistic regression model (LogReg) [8], SVM [11], fully connected neural network with 1024 neurons (NN), Gradient Boosting Machine (GBM) [7], and Multi-Armed Bandit (MAB) [23] using ZROs/others, P-ZROs/others, and ZROs+P-ZROs/others as classifications, respectively. As shown in Figure 4, all models perform better at identifying ZROs than identifying P-ZROs, which is consistent with our hypothesis. In addition, the accuracy of all models in identifying both ZROs and P-ZROs is lower than their accuracy in identifying ZROs or P-ZROs. However, we can only accept the former scheme in practice because of the interaction between ZROs and P-ZROs mentioned in the above section. MAB exhibits advantages in identifying both ZROs and P-ZROs under different workloads. As a reinforcement learning model, MAB learns the objects by perceiving continuous changes over a period and making decisions from a global perspective. In addition, its resource consumption and efficiency have been proven suitable for caching environments [18]. As a result, we use MAB to achieve uniform identification for ZROs and P-ZROs.

3 SCIP DESIGN

Based on MAB, we incorporate the bimodal insertion policy to design a smart cache insertion and promotion policy, *i.e.*, SCIP.

3.1 Bimodal insertion policy

The Bimodal Insertion Policy (BIP) [17] inserts the object into different positions based on a probabilistic parameter, *i.e.*, $0 \leq \alpha \leq 1$. When $\alpha > 0.5$, BIP will insert the object into the MRU position, otherwise into the LRU position. Without traversing the queue, selecting only the LRU and MRU positions for insertion is efficient.

3.2 Judgement with history lists

To perceive ZROs and P-ZROs based on BIP, we set up two history lists, *i.e.*, H_m and H_l. They store the metadata of evicted objects that were inserted into the MRU and LRU positions, respectively. Logically, the size of each list is half of the real cache. We maintain two lists according to the first-in-first-out (FIFO) policy. If a missing object is hit in two lists, the insertion position of the object should be adjusted. For example, when the missing object is in H_l, it means that the object has a chance to be hit if it is inserted into the MRU position. On the contrary, when the missing object is in H_m, the object is a ZRO or a P-ZRO.

3.3 Learning in the MAB framework

Based on the above principle, we incorporate the MAB framework to learn the function used for position selection. Specifically, SCIP selects from exactly two basic experts, *i.e.*, MRU insertion policy (MIP) and LRU insertion policy (LIP), and learns the probability of insertion into the suitable position, where we use the learning rate to bridge the relationship between the probability and the hit rate. With the continuously varying history lists, it can adapt to the dynamic workload and make decisions. We show its pseudo-code in Algorithm 1 and interpret several key functions below.

Promotion and Insertion. We treat the promotion operation as a special insertion operation and deal with the hit object according to the insertion policy for missing objects. As shown in Line-23 to Line-26 of Algorithm 1, SCIP first removes O_i from the cache and then inserts O_i by MIP or LIP, where the removed object will not be recorded in the history lists. C.`REMOVE`(O_t) in Line-24 means O_t is removed from the cache and does not write to the history list. The insertion policy is shown in Line-27 to Line-33 of Algorithm 1. `SELECT`((MIP, LIP), (ω_m, ω_l), γ) in Line-28 generates a real-time random number $\gamma \in [0, 1]$ for comparing the execution probabilities of LIP and MLP, *i.e.*, ω_m and ω_l. If $\omega_m > \gamma$, the object will insert into the MRU position; otherwise, the LRU position.

Updating insertion probabilities. As shown in Line-6 to Line-13 of Algorithm 1, the change in probability is triggered according to the principle described in §3.2, where we use the learning rate (*i.e.*, λ) to adjust ω_m and ω_l, respectively. Note that $\omega_m + \omega_l = 1$ and λ will change as hit rates. `DELETE` in Line-7 and Line-10 means that the algorithm deletes all information of O_t in H_M and H_l, respectively.

Evicting when the cache is full. As shown in Line-14 to Line-19 of Algorithm 1, when the cache is full but new objects need to be

Algorithm 1: SCIP(MIP, LIP)

```
1  Definition: $C$: the cache; $t$: the current time; $\lambda_t$: the learning
      rate at $t$; $H_m/H_l$: the history list storing objects inserted in
      the MRU/LRU position; $\Pi_t$: the average hit rate at $t$; $i$: the
      learning rate update interval; $\gamma$: a real-time random
      number; $\omega_m/\omega_l$: the execution possibility for MIP/LIP.
2  Input: requested object: $O_t$;
3  if $O_t$ is in $C$ then
4      $C$.PROMOTE($O_t$)
5  else
6      if $O_t$ is in $H_m$ then
7          $H_m$.DELETE($O_t$)
8          $\omega_m := \omega_m \times e^{-\lambda}$ // decrease $\omega_m$
9      else if $O_t$ is in $H_l$ then
10         $H_l$.DELETE($O_t$)
11         $\omega_l := \omega_l \times e^{-\lambda}$ // decrease $\omega_l$
12     $\omega_m := \frac{\omega_m}{\omega_m + \omega_l}$ // normalize
13     $\omega_l := 1 - \omega_m$
14     if Cache $C$ is full then
15         $O_{del} = C$.EVICT()
16         if $O_{del}$ was inserted in the MRU position then
17             $H_m$.ADD($O_{del}$)
18         else
19             $H_l$.ADD($O_{del}$)
20     $C$.INSERT($O_t$)
21 if $t\%i$==0 then
22     UPDATELR($\lambda_{t-i}, \lambda_{t-2i}, \Pi_t, \Pi_{t-i}$)
23 Function PROMOTE($O_t$):
24     $C$.REMOVE($O_t$)
25     $C$.INSERT($O_t$)
26 return
27 Function INSERT($O_t$):
28     action = SELECT ((MIP, LIP), ($\omega_m, \omega_l$), $\gamma$)
29     if action == MIP then
30         Insert $O_t$ in the MRU position and mark it
31     else
32         Insert $O_t$ in the LRU position and mark it
33 return
34 Function ADD($O_{del}$):
35     if the corresponding history list is full then
36         Delete the object in the LRU position from the list
37     Insert $O_{del}$ in the MRU position of the list
38 return
```

written to the cache, the algorithm will delete objects to make room for new objects, where the victim appears at the LRU position of the queue (*i.e.*, C.EVICT() in Line-15). If this victim was inserted into the MRU position when entering the cache, its metadata information will be added to H_m (in Line-17), otherwise, added to H_l (in Line-19). And the ADD function is shown in Line-34 to Line-38 of Algorithm 1. Note that the difference between the EVICT function and the REMOVE function is that EVICT removes the object at the end of the cache queue and writes the victim to the history list,

Algorithm 2: UPDATELR($\lambda_{t-i}, \lambda_{t-2i}, \Pi_t, \Pi_{t-i}$)

```
1  Function UPDATELR($\lambda_{t-i}, \lambda_{t-2i}, \Pi_t, \Pi_{t-i}$):
2      $\Delta_t := \Pi_t - \Pi_{t-i}$
3      $\delta_t := \lambda_{t-i} - \lambda_{t-2i}$
4      if $\delta_t \neq 0$ then
5          if $\frac{\Delta_t}{\delta_t} > 0$ then
6              $\lambda_t := \min(\lambda_{t-i} + \lambda_{t-i} \times \frac{\Delta_t}{\delta_t}, 1)$
7          else
8              $\lambda_t := \max(\lambda_{t-i} + \lambda_{t-i} \times \frac{\Delta_t}{\delta_t}, 0.001)$
9          $unlearnCount := 0$
10     else
11         if $HR_t = 0$ or $\Delta_t \leq 0$ then
12             $unlearnCount := unlearnCount + 1$
13         if $unlearnCount \geq 10$ then
14             $unlearnCount := 0$
15             $\lambda_t :=$ select randomly between 0.001 and 1
16 return
```

whereas the object processed by REMOVE is directly removed from the cache.

Updating the learning rate. As shown in Line-21 to Line-22 of Algorithm 1, we update λ with a period of i since it is possible to obtain a significant change in the hit rate. As shown in Algorithm 2, we list the specific update process, where Line-2 to Line-8 express the way the learning rate varies with the hit rate. When the change in hit rates is in the same direction as the change in learning rates, we believe that the learning rate should be amplified to speed up finding better probabilities. On the contrary, we decrease its learning rate and prevent it from going further down the wrong route. For the learning step size, we expect to reflect the linear relationship between the change in learning rate and the change in hit rate. This adaptive design is consistent with the gradient-based stochastic hill climbing approach and supports random restarts [21]. If the gradient is positive/negative, then the direction of change of the learning rate is sustained/reversed. The amount of change of λ and Π in the previous window determines the magnitude of the change in the learning rate for the next window. As a result, if the performance increases/decreases by increasing/decreasing the learning rate, we will increase/decrease the learning rate by multiplying it by the quotient of the change in the hit rate and the change in the learning rate from the previous window. If the performance keeps degrading, we reset the learning rate to its initial value. In addition, we use a variable *unlearncount* to control the reset frequency. In our experiments, setting *unlearncount* to 10 gave better performance, ensuring restarts after performance dropped for a long time.

4 INTEGRATION WITH CACHE REPLACEMENT ALGORITHMS

SCIP does not blindly adhere to the locality principle. Instead, it incorporates ZRO and P-ZRO identification to determine where to insert missing and hit objects. It can serve as a supplement to insertion and promotion policies in current cache replacement algorithms, enhancing their overall effectiveness.

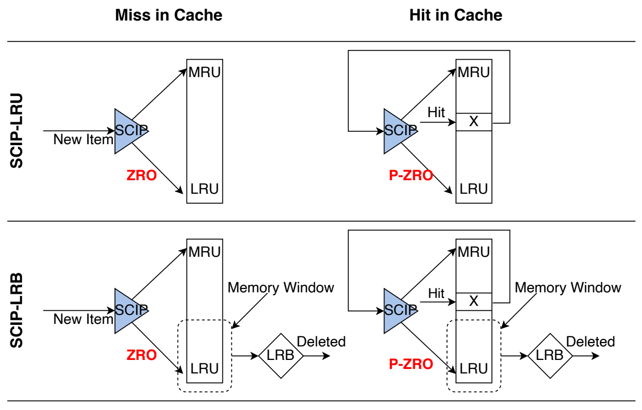


Figure 5: Understanding SCIP-LRU and SCIP-LRB.

We believe that current cache replacement algorithms can be broadly classified into two types: passive eviction policies and active eviction policies. The passive policies (*e.g.*, LRU, ARC, S4LRU, or CACHEUS) generally use temporal features (*e.g.*, recency and frequency) to keep useful objects in the head of a cache queue and therefore indirectly speed up the eviction of useless ones. The active policies (*e.g.*, LeCaR [23], LRB, or GL-Cache [29]), on the other hand, try to learn the local features of access sequences from historical data using machine learning models, which allows them to directly evict useless objects using learned models.

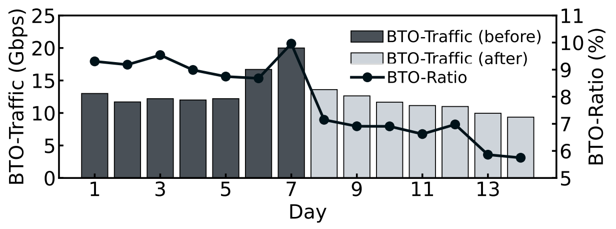
For the passive policies, users can utilize SCIP to replace their insertion and promotion policies since the policies do not account for the negative effects of placing ZRO and P-ZRO in the MRU position. Note that SCIP cannot be well adapted to multi-chain structure algorithms, but this is a focus of our future work. As for the active policies, SCIP can be used as a complement to a machine-learning model to determine the insertion position for missing and hit objects. In addition, SCIP can be adapted to the learning domain of the original method to ensure that the eviction and insertion judgments are derived from the same data.

To demonstrate the compatibility of SCIP, we will use LRU and LRB as examples to explain how SCIP integrates into them, as shown in Figure 5. For SCIP-LRU, when there is a cache miss, SCIP checks if it is a ZRO and inserts it into the LRU position if it is or the MRU position if it is not. When a cache hit occurs, the object is first removed from the cache queue and then inserted according to SCIP, at the LRU position if it is P-ZRO, otherwise at the MRU position. Similarly, SCIP-LRB performs the same operations for cache misses and hits. Note that SCIP can learn the model by following the memory window of LRB instead of globally sampling, which significantly increases algorithm efficiency.

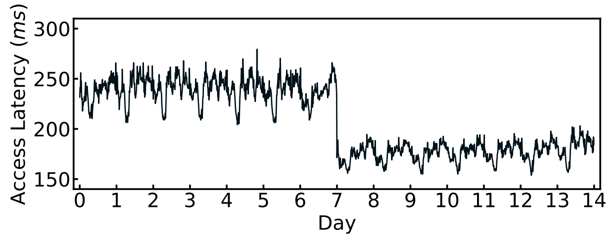
5 DEPLOYMENT AND PERFORMANCE IN THE REAL PRODUCTION SYSTEM

5.1 Deployment

We deployed SCIP to T Disk Cache (TDC), which is a cache system owned by Tencent and serves over 2,500 businesses (as shown in Figure 2). TDC's prototype is a storage node based on MCP++, a multi-ccd/multi-smcd process model, raw disks, inode, and asynchronous disk I/O technologies such as libaio/SPDK, which support cache replacement algorithms and provide key-value storage services. In the system, inodes are used to manage metadata of objects, including an MD5 index, an integer variable used to record the object size, forward and backward pointers to the inode, and an insertion position variable (*insert_pos*). When an object is written into the MRU/LRU position of the cache, *insert_pos* is set to 1 or 0, respectively. The size of metadata is approximately 110 bytes. Note that while placing indexes on SSDs can meet the requirements of high-speed response, the frequent deletion of indexes reduces the SSD life and may cause system disk failures. As a result, the system places the indexes in memory. In practice, the shadow cache is the history list used to record the metadata of evicted objects, where H_m and H_l record those of evicted objects with $insert\_pos = 1$ and $insert\_pos = 0$, respectively. The memory overhead is low since only metadata must be tracked, such as the key of the object (a string) and the object size (a long integer). Furthermore, since engineers have deployed LRU in TDC, we have merely replaced LRU's insertion policy with SCIP.



(a) BTO-Bandwidth and BTO-Ratio



(b) Access Latency

Figure 6: Compared with state-of-the-art replacement policies.

5.2 Performance improvement in actual production system

As shown in Figure 6(a) and 6(b), we measured the performance changes in "Backing to Original Source" (BTO) traffic and ratio (BTO-ratio), as well as the average user access latency from the monitoring system. "Backing to Original Source" is a common term used to describe the process of fetching data from the source and writing it to the cache whenever a cache miss occurs.

Table 1: Summary of workloads that are used in experiments.

	CDN-T	CDN-W	CDN-A
Total Requests(Millions)	78.75	100.0	99.55
Unique Object(Millions)	24.71	2.34	54.43
Max Object Size(MB)	19.97	674.38	7.99
Min Object Size(B)	2	10	2
Mean Object Size(KB)	44.56	35.07	31.21
Working Set Size(GB)	1097	327	1580

After deploying SCIP, the average BTO-ratio (i.e., the miss ratio) decreased by 2.28% (from 8.87% to 6.59%), the average BTO-traffic decreased by 3.92 Gbps (25.7%), and the average user access latency decreased by 62.11 ms (26.1%). Before the introduction of SCIP, administrators would likely resort to resource over-provisioning by expanding their equipment to cope with the surge in traffic, and then remove the over-provisioned equipment when the traffic decreased, resulting in additional human labor and equipment costs.

Furthermore, the cache size of TDC is approximately 3000 TB, and adding 100 TB of SSD would only achieve a 1% reduction in BTO-ratio in practice.

6 EVALUATION

6.1 Settings

Workloads. We test SCIP over two public CDN workloads (*i.e.*, CDN-W [22] and CDN-A [31]) and a real CDN workload gathered from TDC (*i.e.*, CDN-T). We show their attributions in TABLE 1, where the working set size means the size of all unique objects [30]. The cache size aligns with the working set size of the trace since the cache size is a parameter based on resource availability and system requirement [30].

State-of-the-art cache algorithms. We evaluate SCIP against eight state-of-the-art insertion and promotion policies, while also comparing it to nine state-of-the-art cache replacement algorithms, including those not used for CDNs.

- The insertion and promotion policies include LRU Insertion Policy (LIP), Dynamic Insertion Policy (DIP) [17], Promotion/insertion pseudo-partitioning (PIPP) [27], Decision Tree Analysis (DTA) [12], Signature-based Hit Predictor (SHiP) [26], Genetic Insertion and Promotion for PseudoLRU Replacement (DGIPPR) [10], Deadblock Aware Adaptive Insertion Policy (DAAIP) [14], and adaptive size-aware cache insertion policy (ASC-IP) [25]. All insertion and promotion policies employ the LRU victim selection policy by default.
- The replacement algorithms include LRU, LRU-K [16], S4LRU [31], Smart Segmented LRU (SS-LRU) [13], GreedyDual Size Frequency (GDSF) [5], Least Hit Density (LHD) [2], CACHEUS [18], Learning Relaxed Belady (LRB) [22], and Group-level Learning (GL-Cache) [29]. These state-of-the-art cache replacement algorithms primarily optimize the victim selection policy based on heuristic rules or machine learning algorithms. Note that LRB and GL-Cache are the latest research on CDNs. Nevertheless, they use the most basic policy like LRU for both insertion and promotion policies.

We use Belady's results as the lower bound of the miss ratio since Belady [3] is the theoretical optimal method that calculates using global information of the entire access sequence. However, it cannot be implemented in an actual production system. To illustrate the impact of P-ZRO on the SCIP algorithm, we remove the promotion policy based on the perception of P-ZROs and form a simplified algorithm called the Smart Cache Insertion policy (SCI), as shown in Algorithm 3. This policy directly inserts hit objects into the MRU position.

Simulator and testbed. We deploy the simulator based on LRB [22]. We launch the above cache algorithms on the empty cache queue. The simulator is running on a device with a 56-core 2.40GHz CPU, 32GB of RAM, and a 6TB SSD. All subsequent experiments are executed on this simulator, and the cache size is 64GB by default.

Algorithm 3: SCI(MIP, LIP)

```
1  Definition: $C$: the cache; $t$: the current time; $\lambda_t$: the learning
   rate at $t$; $H_m/H_l$: the history list storing objects inserted in
   the MRU/LRU position; $\Pi_t$: the average hit rate at $t$; $i$: the
   learning rate update interval; $\gamma$: a real-time random
   number; $\omega_m/\omega_l$: the execution possibility for MIP/LIP.
2  Input: requested object: $O_t$;
3  if $O_t$ is in $C$ then
4      $C$.REMOVE($O_t$)
5      Insert $O_t$ in MRU position and mark it
6  else
7      if $O_t$ is in $H_m$ then
8          $H_m$.DELETE($O_t$)
9          $\omega_m := \omega_m \times e^{-\lambda}$ // decrease $\omega_m$
10     else if $O_t$ is in $H_l$ then
11         $H_l$.DELETE($O_t$)
12         $\omega_l := \omega_l \times e^{-\lambda}$ // decrease $\omega_l$
13     $\omega_m := \frac{\omega_m}{\omega_m+\omega_l}$ // normalize
14     $\omega_l := 1 - \omega_m$
15     if Cache $C$ is full then
16         $O_{del}$ = $C$.EVICT()
17         if $O_{del}$ was inserted in the MRU position then
18             $H_m$.ADD($O_{del}$)
19         else
20             $H_l$.ADD($O_{del}$)
21     $C$.INSERT($O_t$)
22 if $t\%i==0$ then
23     UPDATELR($\lambda_{t-i}, \lambda_{t-2i}, \Pi_t, \Pi_{t-i}$)
24 Function INSERT($O_t$):
25     action = SELECT ((MIP, LIP), ($\omega_m, \omega_l$), $\gamma$)
26     if action == MIP then
27         Insert $O_t$ in the MRU position and mark it
28     else
29         Insert $O_t$ in the LRU position and mark it
30 return
31 Function ADD($O_{del}$):
32     if the corresponding history list is full then
33         Delete the object in the LRU position from the list
34     Insert $O_{del}$ in the MRU position of the list
35 return
```

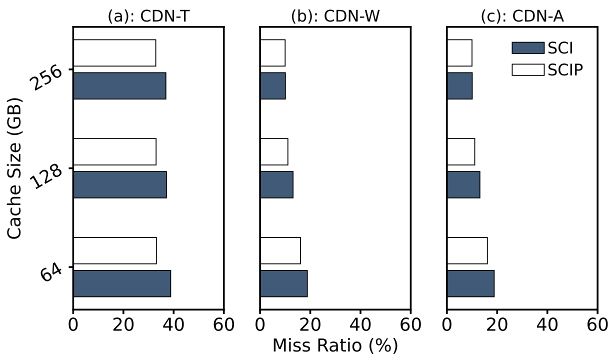


Figure 7: Compared SCIP with SCI.

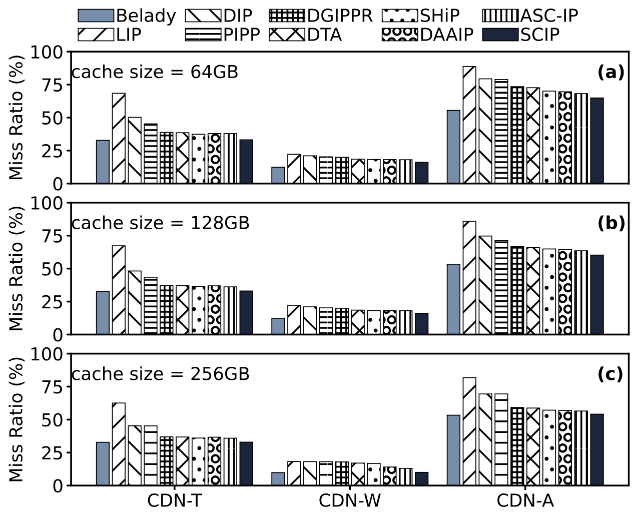


Figure 8: Comparisons of Belady, SCIP, and eight insertion policies in miss ratio on three workloads.

6.2 Compared SCIP with SCI

In § 2.2, the theoretical results confirm the significant performance improvement achieved by integrating these two policies compared to using a single policy. To experimentally verify our hypothesis, we conducted tests on SCIP and SCI under identical conditions. As depicted in Figure 7, SCIP outperforms SCI in terms of miss ratio across three workloads. On the CDN-T, CDN-W, and CDN-A workloads, the average miss ratios yielded by SCIP are 4.62%, 1.62%, and 5.30% lower than those yielded by SCI, respectively. We attribute these advantages to SCIP's ability to capture P-ZROs. In addition, these results demonstrate that placing P-ZROs in the LRU position can improve caching performance.

6.3 Compared with insertion and promotion policies

As shown in Figure 8(a), SCIP exhibits significant improvements compared to the current advanced ASC-IP algorithm. SCIP reduces the missing rate by 4.69% on CDN-T, 1.92% on CDN-W, and 3.26% on CDN-A. This improvement is attributed to SCIP's ability to capture both P-ZRO and ZRO features, whereas ASC-IP only processes ZRO. In contrast, when compared to the least efficient LRU insertion policy (LIP), SCIP demonstrates a substantial reduction in miss ratio of 35.32% on CDN-T, 6.08% on CDN-W, and 23.87% on CDN-A. LIP simply inserts all incoming objects into the LRU by controlling the insertion position, which often results in the insertion of non-ZRO and non-P-ZRO objects. Consequently, when the next access misses, those objects are evicted from the cache, thereby losing the opportunity for subsequent potential hits. However, SCIP still has room for improvement in achieving Belady's theoretically unachievable lower bound, indicating that the predictions for P-ZRO and ZRO can still be further accurate. A similar conclusion can be drawn from Figure 8(b) and (c), where the cache sizes are 128GB and 256GB, respectively.

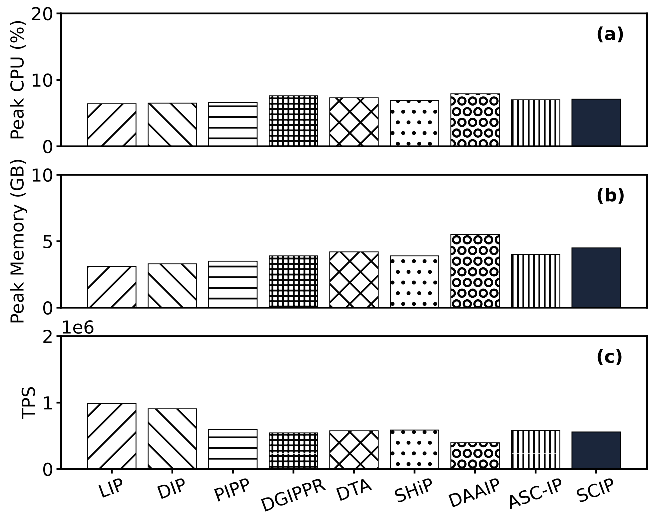


Figure 9: Comparisons of Belady, SCIP, and eight insertion policies in peak CPU utilization, peak memory, and transactions per second (TPS) on CDN-T. Lower miss ratio, lower peak CPU, and lower peak memory, as well as higher TPS, indicate better performance.

For peak CPU utilization, as shown in Figure 9(a), SCIP consumes an average of 0.42% more than the simple heuristic algorithms LIP, DIP, PIPP, SHIP, and ASC-IP. Compared to learning-based policies DGIPPR, DTA, and DAAIP, SCIP consumes 0.5% less on average. These results show SCIP's superiority in consuming computational resources compared to other learning-based policies, despite the additional resources needed for the machine learning framework. Figure 9(b) shows peak memory utilization, where SCIP's memory consumption is only marginally higher than LIP (1.3GB on average), which has the lowest memory consumption. However, to achieve the same miss ratio as SCIP, LIP would need to expand the cache size by 7000 times, requiring significant resource allocation. To evaluate insertion efficiency, we conducted transactions per second (TPS) tests, as shown in Figure 9(c). SCIP's TPS generally aligns with PIPP, DGIPPR, DTA, SHiP, and ASC-IP, and surpasses that of DAAIP. These results indicate that our insertion efficiency is second only to directed and random insertion policies. We also

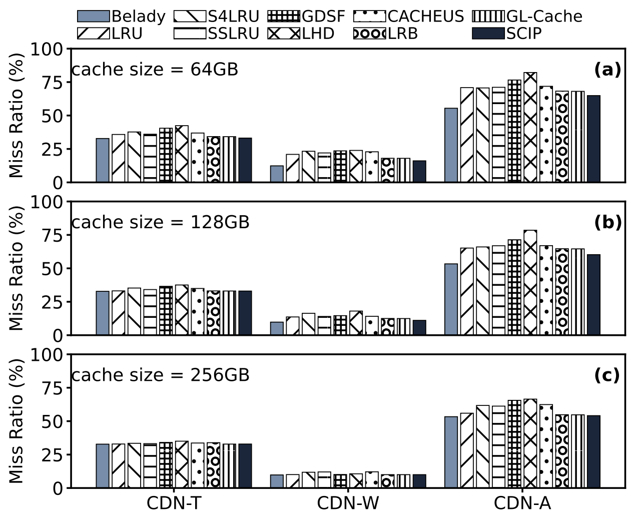


Figure 10: Comparisons of Belady, SCIP, and eight replacement algorithms in miss ratio on three workloads.

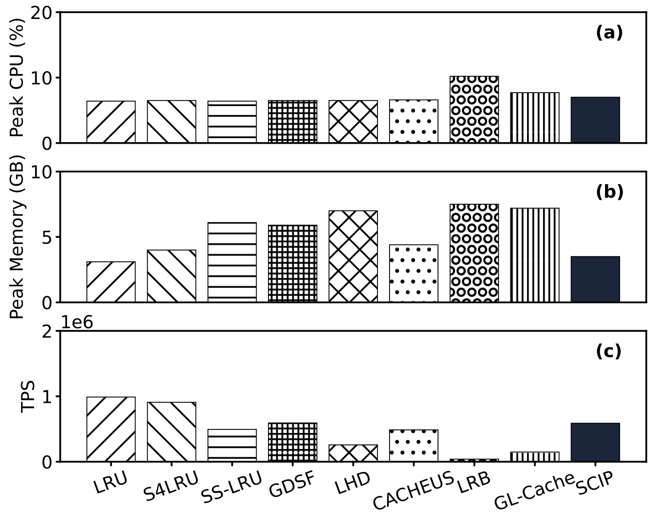


Figure 11: Comparisons of Belady, SCIP, and eight replacement algorithms in peak CPU utilization, peak memory, and transactions per second (TPS) on CDN-T. Lower miss ratio, lower peak CPU, and lower peak memory, as well as higher TPS, indicate better performance.

compared resource utilization on CDN-W and CDN-A, but the difference in dominance resulting from changing these settings is only approximately 0.8%. Therefore, we do not present them. Similarly, since this phenomenon occurs in the comparison of replacement algorithms, we only show the most representative set of results, *i.e.*, results on a cache size of 64GB.

6.4 Compared with replacement algorithms

As shown in Figure 10, SCIP outperforms the other eight replacement algorithms, except for Belady's theoretical lower bound, which cannot be achieved in reality. The average miss ratio of SCIP is lower than that of GL-Cache (the current optimal algorithm) by 1.38%. In addition, since SCIP is configured with the machine learning model, we are interested in comparing it to learning-based methods, including LRB, CACHEUS, and SS-LRU. The comparison results confirm the superiority of our algorithm. We attribute these advantages to a direct focus on ZROs and P-ZROs. SCIP's learning rate update is sensitive to changes in the miss ratio, and SCIP can accurately perceive ZROs and P-ZROs.

As shown in Figure 11(a)(b), SCIP's peak CPU and memory utilization are slightly higher than those of general heuristic algorithms such as S4LRU, GDSF, and LHD. We believe that simple models and insertion and promotion policies share the same models. Nevertheless, SCIP's resource consumption is much lower than that of learning models, such as LRB and GL-Cache. Moreover, SCIP's insertion efficiency is lower than that of LRU and S4LRU, but higher than others. These results are shown in Figure 11(c).

6.5 Enhance replacement algorithms

SCIP can enhance existing replacement algorithms by replacing their original insertion and promotion policies. To verify its effectiveness, we conducted tests using LRU-K and LRB, with ASC-IP serving as the reference method. As shown in Figure 12, compared to LRU-K and LRB, LRU-K-SCIP and LRB-SCIP reduce the average

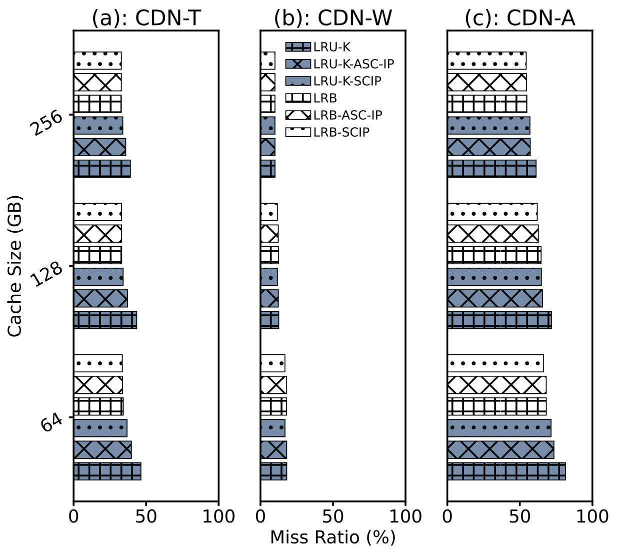


Figure 12: The comparisons of cache replacement algorithms and their SCIP versions.

miss ratio by 8.05% and 0.44%, respectively. These results demonstrate that SCIP can further reduce the miss ratio yielded by state-of-the-art replacement algorithms. In addition, our enhancing values on LRU-K and LRB exceed those of ASC-IP by 2.67% and 0.25%, respectively. This demonstrates our superiority in improving existing replacement algorithms.

7 RELATED WORK

Cache replacement algorithm. The problem of cache replacement can be divided into two parts: victim selection policy and insertion policy, as proposed by [17]. Traditional caching algorithms mainly focus on the victim selection policy, which is based on the characterization of different data features (such as new progress, frequency, data size, etc.) by designing the storage structure. For example, ARC [15], LIRS [9], LHD [2] *etc.* improve performance by adding or adjusting the structure of the cache queue. In recent years, AI technology has been introduced into victim selection policies [13, 18, 19, 22, 28, 29]. Other researchers focus on insertion policies running by heuristic [10, 12, 14, 17, 25–27]. For example, ASC-IP [25] designs an adaptive scheme to dynamically adjust the size threshold used to determine the zero-reuse objects and insert zero-reuse objects into the LRU position.

Cache admission algorithm. Many researchers have realized that denying data that will not be accessed into the cache can effectively improve cache performance, and developed admission algorithms. 2Q [20] decides all objects according to their access frequency and only those accessed twice are allowed into the cache. TinyLFU [6] maintains an approximate representation of the access frequency of a large sample of recently accessed items. AdaptSize [4] models the cache as a Markov process that incorporates all correlations between object sizes and current access rates, finding the optimal size-aware admission policy. Wang *et al.* [24] propose predicting one-time-access objects by a decision-tree model learning features of size, which determines admission. These policies are efficient when predicting accurately; otherwise, they will cause the object to lose the chance to be hit.

8 CONCLUSION

We have discovered P-ZROs and proposed integrated insertion and promotion policies. To implement a consistent insertion policy for both missing and hit objects, we designed SCIP to harmonize the treatment of ZROs and P-ZROs by incorporating MAB and a bimodal insertion policy. Due to its well-designed learning rate function, SCIP is capable of perceiving ZROs and object reusability with greater accuracy. SCIP achieves the optimal miss ratio compared to existing insertion policies and may complement replacement policies at a low cost to increase the optimal miss ratio. Because of its low resource consumption and superior performance, we consider applying SCIP in other cache systems in the future. SCIP sources can be downloaded at https://github.com/oswald-hust/icpp2023.

ACKNOWLEDGMENTS

This work is also done in Key Laboratory of Information Storage System and Ministry of Education of China. It is supported by the National Natural Science Foundation of China (key program) No.62232007, and the Natural Science Foundation of Hubei Province No.2022CFB060.

REFERENCES

[1] Odd O Aalen. 1989. A linear regression model for the analysis of life times. *Statistics in medicine* 8, 8 (1989), 907–925.
[2] Nathan Beckmann, Haoxian Chen, and Asaf Cidon. 2018. LHD: Improving Cache Hit Rate by Maximizing Hit Density.. In *USENIX NSDI*. 389–403.
[3] Laszlo A. Belady. 1966. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal* 5, 2 (1966), 78–101.
[4] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. 2017. Adaptsize: Orchestrating the hot object memory cache in a content delivery network.. In *NSDI*, Vol. 17. 483–498.
[5] Ludmila Cherkasova and Gianfranco Ciardo. 2001. Role of aging, frequency, and size in web cache replacement policies. In *Springer HPCN*. 114–123.
[6] Gil Einziger, Roy Friedman, and Ben Manes. 2017. Tinylfu: A highly efficient cache admission policy. *ACM TOS* 13, 4 (2017), 1–31.
[7] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.
[8] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the eighth international workshop on data mining for online advertising*. 1–9.
[9] Song Jiang and Xiaodong Zhang. 2002. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Performance Evaluation Review* 30, 1 (2002), 31–42.
[10] Daniel A Jiménez. 2013. Insertion and promotion for tree-based pseudolru last-level caches. In *IEEE/ACM MICRO*. 284–296.
[11] Thorsten Joachims. 1998. *Making large-scale SVM learning practical.* Technical Report. Technical report.
[12] Samira Khan and Daniel A Jiménez. 2010. Insertion policy selection using decision tree analysis. In *IEEE ICCD*. IEEE, 106–111.
[13] Chunhua Li, Man Wu, Yuhan Liu, Ke Zhou, Ji Zhang, and Yunqing Sun. 2022. SS-LRU: a smart segmented LRU caching. In *ACM/IEEE DAC*. 397–402.
[14] Sujit Kr Mahto, Suhit Pai, Virendra Singh, et al. 2017. DAAIP: Deadblock aware adaptive insertion policy for high performance caching. In *IEEE ICCD*. 345–352.
[15] Nimrod Megiddo and Dharmendra S Modha. 2003. ARC: A Self-Tuning, Low Overhead Replacement Cache.. In *USENIX FAST*, Vol. 3. 115–130.
[16] Elizabeth J O'neil, Patrick E O'neil, and Gerhard Weikum. 1993. The LRU-K page replacement algorithm for database disk buffering. *Acm Sigmod Record* 22, 2 (1993), 297–306.
[17] Moinuddin K Qureshi, Aamer Jaleel, Yale N Patt, Simon C Steely, and Joel Emer. 2007. Adaptive insertion policies for high performance caching. *ACM SIGARCH Computer Architecture News* 35, 2 (2007), 381–391.
[18] Liana V Rodriguez, Farzana Beente Yusuf, Steven Lyons, Eysler Paz, Raju Rangaswami, Jason Liu, Ming Zhao, and Giri Narasimhan. 2021. Learning Cache Replacement with CACHEUS.. In *FAST*. 341–354.
[19] Subhash Sethumurugan, Jieming Yin, and John Sartori. 2021. Designing a cost-effective cache replacement policy using machine learning. In *IEEE HPCA*. 291–303.
[20] D Shasha and T Johnson. 1994. 2q: A low overhead high performance buffer management replacement algorithm. In *ACM VLDB*. 439–450.
[21] Walter L Smith. 1955. Regenerative stochastic processes. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 232, 1188 (1955), 6–31.
[22] Zhenyu Song, Daniel S Berger, Kai Li, Anees Shaikh, Wyatt Lloyd, Soudeh Ghorbani, Changhoon Kim, Aditya Akella, Arvind Krishnamurthy, Emmett Witchel, et al. 2020. Learning relaxed belady for content distribution network caching. In *USENIX NSDI*. 529–544.
[23] Giuseppe Vietri, Liana V Rodriguez, Wendy A Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. 2018. Driving Cache Replacement with ML-based LeCaR.. In *HotStorage*. 928–936.
[24] Hua Wang, Xinbo Yi, Ping Huang, Bin Cheng, and Ke Zhou. 2018. Efficient SSD caching by avoiding unnecessary writes using machine learning. In *ACM ICPP*. 1–10.
[25] Peng Wang, Yu Liu, Zhelong Zhao, Ke Zhou, Zhihai Huang, and Yanxiong Chen. 2022. Adaptive Size-Aware Cache Insertion Policy for Content Delivery Networks. In *IEEE ICCD*. 195–202.
[26] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C Steely Jr, and Joel Emer. 2011. SHiP: Signature-based hit predictor for high performance caching. In *IEEE/ACM MICRO*. 430–441.
[27] Yuejian Xie and Gabriel H Loh. 2009. PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches. *ACM SIGARCH Computer Architecture News* 37, 3 (2009), 174–183.
[28] Gang Yan, Jian Li, and Don Towsley. 2021. Learning from optimal caching for content delivery. In *ACM CoNEXT*. 344–358.
[29] Juncheng Yang, Ziming Mao, Yao Yue, and KV Rashmi. 2023. {GL-Cache}: Group-level learning for efficient and high-performance caching. In *USENIX FAST*. 115–134.
[30] Yu Zhang, Ping Huang, Ke Zhou, Hua Wang, Jianying Hu, Yongguang Ji, and Bin Cheng. 2020. OSCA: An online-model based cache allocation scheme in cloud block storage systems. In *USENIX ATC*. 785–798.
[31] Ke Zhou, Si Sun, Hua Wang, Ping Huang, Xubin He, Rui Lan, Wenyan Li, Wenjie Liu, and Tianming Yang. 2018. Demystifying cache policies for photo stores at scale: A tencent case study. In *ACM ICS*. 284–294.