2022 IEEE 40th International Conference on Computer Design (ICCD) | 978-1-6654-6186-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCD56317.2022.00037

Adaptive Size-Aware Cache Insertion Policy for Content Delivery Networks

Peng Wang*, Yu Liu†✉, Zhelong Zhao*, Ke Zhou*, Zhihai Huang‡, Yanxiong Chen‡
*Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China
†School of Computer of Science and Technology, Huazhong University of Science and Technology, Wuhan, China
‡Tencent Technology (Shenzhen) Co., Ltd., Shenzhen, China
{wp_hust, liu_yu, zhelongzhao, zhke}@hust.edu.cn, {tommyhuang, yxbillchen}@tencent.com

***Abstract*—Content delivery networks (CDNs) are large distributed cache systems that deliver objects with inconsistent sizes. The *zero-reuse* objects that are not reused in a time window but still loaded and evicted in the cache waste cache resources and result in degradation of object hit ratio (OHR) in CDNs. Although prohibiting these objects from entering the cache is a viable solution, the variable workloads and various object sizes in CDNs make the determination of *zero-reuse* difficult, resulting in an increased risk of bandwidth overhead in the data center by the misjudgment. To alleviate this problem, we propose to use the insertion policy to give each object at least one chance to be hit. Meanwhile, we find that the distribution of *zero-reuse* objects correlates with their sizes through data analysis. As a result, we propose an adaptive size-aware cache insertion policy (ASC-IP) for the OHR improvement and design an adaptive scheme to dynamically adjust the size threshold used to determine the *zero-reuse* objects, adapting the mutative access patterns with negligible overhead. We have deployed ASC-IP in TDC of Company-T and ASC-IP can improve the OHR by 9.6% and reduce the user access latency by 7.14ms on average and reduce the back-to-source bandwidth by 8.75Gbps. In addition, on Twitter, Wikipedia, and a real-world Trace-T, we show that ASC-IP outperforms state-of-the-art cache algorithms working on CDNs and can upgrade LRU-based replacement algorithms with negligible overheads.**

***Index Terms*—content delivery network, cache replacement algorithm, cache insertion policy**

I. Introduction

Content delivery networks (CDNs) deliver the texts, images, videos, and websites from thousands of tenants to users that the content is managed by a large network of caches [11]. The goal of CDN cache management is to maximize the object hit ratio (OHR) via the efficient use of cache resources [1]. Some objects will not be reused in a time window but still be loaded into the cache and go through the whole queue before being evicted, resulting in a waste of cache sources and the object hit ratios (OHRs) degradation (See §II-A).

Many researchers resort to admission algorithms to alleviate this issue. For example, Hua *et al.* [13] propose to predict one-time-access objects by the decision-tree model, determining admission. Nevertheless, admission mechanisms based on probabilistic models can expose data centers to the risk. Once the forecast is wrong, the suspected objects have no chance to be hit, resulting in extra bandwidth overheads. Obviously, this is inevitable in CDNs where workloads change frequently and object sizes are diverse. We believe that a robust solution should give each object a chance to be hit but with different chances (See §II-B and §VII-D).

We first pay attention to the cache replacement algorithm, consisting of the eviction policy and the insertion policy [8]. The traditional replacement algorithms are well researched for the former policy but use the same pattern for the latter, *i.e.*, inserting objects into the furthest position from eviction in the queue since objects that have just been requested usually have a high probability of being reused. It means that the insertion position partly determinate the stay time in the cache queue and hit chances of objects. Consequently, we can give objects different chances to be hit by adjusting their insertion positions (See §II-C).

As a result, we determine to design an insertion policy based on whether the object is a *zero-reuse* object, where the *zero-reuse* object is defined as an object not reused within a window of the access sequence. Since CDNs are subject to extreme variability in request patterns and object sizes [1], we need the insertion policy considering those two factors. We statistically find that varying the cache sizes, the access frequencies of objects, and the size distribution of objects are always correlated under different traces. Meanwhile, the reuse frequencies are also correlated with the size distribution of *zero-reuse* objects (See §III). This strong evidence pushed us to complete the policy in a size-aware way and solve two issues. 1) Where is the insertion position for *zero-reuse* objects? 2) How to determine the *zero-reuse* object by size?

On one hand, we believe that the bimodal insertion pattern [8] can enhance the cache replacement algorithms, placing the *non-zero-reuse* objects in the head of the cache queue (*i.e.*, the MRU position [8]) and giving the *zero-reuse* objects at least one chance to be hit in the end of the cache queue (*i.e.*, the LRU position [8]). Furthermore, it is friendly to cache environments that value efficiency, since this pattern can achieve insertion by $O(1)$ time complexity and avoids traversal of the cache deployed by a linked list.

For another hand, we tried to design a function to construct

This work was also achieved in Key Laboratory of Information Storage System and Ministry of Education of China. It was supported by the National Natural Science Foundation of China No.61902135 and the Joint Founds of ShanDong Natural Science Funds (Grant No.ZR2019LZH003).

2576-6996/22/$31.00 ©2022 IEEE
DOI 10.1109/ICCD56317.2022.00037

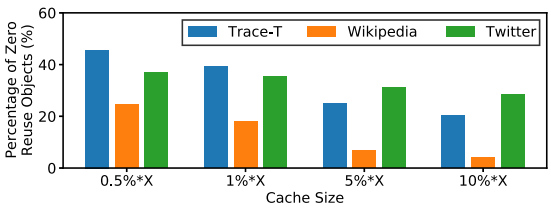


Fig. 1. The percentage of *zero-reuse* objects under different cache sizes. X equals 1024GB, 512GB, and 4GB for Trace-T, Wikipedia, and Twitter, respectively.

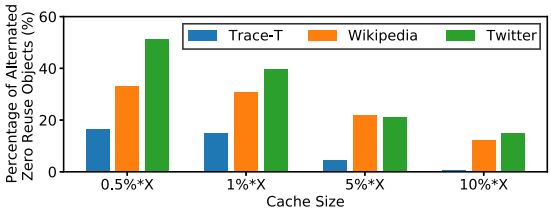


Fig. 2. The percentage of *zero-reuse* objects have alternated states under different cache sizes. X equals 1024GB, 512GB, and 4GB for Trace-T, Wikipedia, and Twitter, respectively.

the relationship between size and *zero-reuse* objects. The machine learning models are hard to retain accuracy in varying workloads while solving the model aging problem inevitably results in efficiency issues. As a result, we propose an adaptive size-aware cache insertion policy, *i.e.*, ASC-IP, in this paper, dynamically adjusting the size threshold like AdaptSize [1]. Specifically, we add two Boolean variables, *i.e.*, mru_tag and hit_tag, into the metadata of each object. The former records its insertion position and the latter records whether it was hit from entry to exit. Besides, like Adaptive Replacement Cache (ARC) [6], a history list is created in the memory to record whether it was hit in the shadow cache. According to these records, the size threshold dynamically floats to achieve real-time and low-cost updates (See §IV-C). We implement ASC-IP in production system (TDC) of Company-T (See §VI). Through evaluations of two public traces, *i.e.*, Wikipedia [11] and Twitter [16], and a trace (Trace-T) from XXPhoto, we observe the following results:

- We have deployed ASC-IP in TDC of Company-T and ASC-IP can improve the OHR by 9.6% and reduce the average user access latency by 7.14ms and reduce the back-to-source bandwidth by 8.75Gbps (See §VI).
- As the first insertion policy for CDNs, ASC-IP improves the OHRs by 3.65%, 11.5%, and 0.3% over the simulator for Trace-T, Wikipedia, and Twitter respectively with negligible additional overheads (1.59% CPU utilization and 0.4% memory overhead) (See §VII-B).
- Based on LRU-based cache replacement algorithms working on CDNs (*e.g.*, ARC [6], S4LRU [3], CACHEUS [9], AdaptSize [1], LRB [11], RLB [15] *etc.*), ASC-IP is a plug-in and can further improve the OHR on these algorithms (See §V and §VII-E).
- ASC-IP and it's integration methods can adapt to variable workloads and various object sizes in CDNs, resulting in a high hit rate (See §IV-C).

II. Background and Motivation

A. Background

XXPhoto is Company-T's cloud service and a special case of e-album business in the context of CDNs. Users can freely upload images and browse their friends' albums in XXPhoto at any time. Facing about 400 million daily active users, the number of uploads is up to 202.4 million, the number of download requests is up to 15.385 billion, and the peak traffic is up to 177.41Gbps per day.

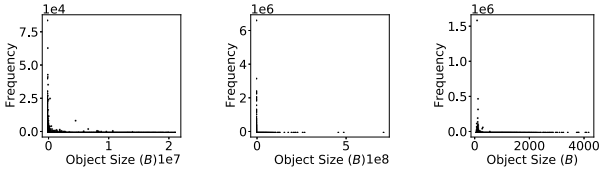
When caching all requested images, even though the size of an image is in KB and the size of the cache is in GB, some image data will never be reused in a time window in the face of such a large and intensive access. Based on Least Recently Used (LRU), we presented the statistics of these *zero-reuse* objects in Fig. 1, where Trace-T is a real-world trace of XXPhoto, Wikipedia and Twitter are public CDN traces. Obviously, although the number of *zero-reuse* objects decreases as the cache size increases, it still accounts for 20% of the total number of objects at 100GB for Trace-T. This phenomenon also occurs in public traces, where the percentages are 4.8% and 27.9% at 50GB and 0.4GB, respectively for Wikipedia and Twitter. The waste of cache resources with limited OHR forces us to find solutions to deal with these *zero-reuse* objects.

B. Cache or not Cache?

An intuitive solution is not to cache these objects, as admission algorithms do. However, *zero-reuse* objects may not always be the state of *zero-reuse*. They will alternate state by the size of the cache or the window of the access sequence. As shown in Fig. 2, on the experimental setup of Fig. 1, we counted the percentage of *zero-reuse* objects that had alternated states in a total number of *zero-reuse* objects. Obviously, this percentage is more than 15% averagely. Unlike one-time-access objects [13] with a static state, *zero-reuse* objects with a dynamic state are difficult to predict, especially in CDNs where workloads and object sizes frequently change. In the event of a prediction error, the data center will bear tremendous bandwidth pressure because the existing admission algorithms tend to reject relatively big objects. As a result, we believe caching all objects resulting in at least one chance to be hit for each object is a more robust solution.

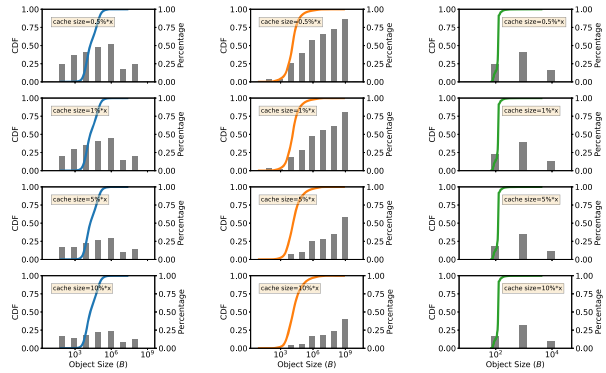
C. Insertion with Different Hit Chances

Back to the beginning, we revisit the replacement algorithms consisting of the eviction policy and the insertion policy [8]. Existing algorithms place a premium on determining what objects should be evicted and how to do so, but there is little attempt to adjust the object's initial insertion position in the cache. The replacement algorithms treat all incoming objects as equal and consider them hot since they have just been accessed. For example, First-In-First-Out (FIFO) and Least Recently Used (LRU) insert the incoming object into the position furthest from eviction, resulting in the most hit



(a) Trace-T (b) Wikipedia (c) Twitter

Fig. 3. The scatter diagram of object sizes (x-axis) and their access frequencies (y-axis).



(a) Trace-T (b) Wikipedia (c) Twitter

Fig. 4. The cumulative *zero-reuse* object distribution and percentages of *zero-reuse* objects in different size regions. X equals 1024GB, 512GB, and 4GB for Trace-T, Wikipedia, and Twitter, respectively.

chances for the object. In our scenario, although the prediction of *zero-reuse* objects may be inaccurate, we give evidence that the incoming objects are no longer completely equal. We desire suspected *zero-reuse* objects to have fewer hit chances than other objects from the moment they enter the queue since they probably won't use the chance even once. If we adjust the initial insertion position of the object, we can achieve it. As a result, we determine to design an insertion policy to achieve fast eviction for *zero-reuse* objects. The following issue is how to determine *zero-reuse* objects.

III. Rationale for Size-aware

The sizes of objects vary widely in CDNs. For example, the size of the largest object is on the order of 10^8 and the smallest object is on the order of 10^0 for Trace-T, 10^{10} and 10^1 for Wikipedia, and 10^4 and 10^0 for Twitter. Meanwhile, large objects occupy most of the cache space, while relatively small objects that are accessed more frequently have to be evicted [1]. We showed the relationships between object sizes and their access frequencies, *i.e.*, popularity extents, in scatter plots for each trace. As shown in Fig. 3, the abscissa x and ordinate y of each point (x, y) correspond to the object's size and the access frequency in the trace, respectively. Obviously, the dots with high positions are concentrated to the left of the x-axis in the three sub-figures, which demonstrates that the small objects in the cache are likely to be accessed more frequently than the relatively large objects. Since low frequently accessed objects have a high probability of becoming *zero-reuse* objects, we believe that the identification of *zero-reuse* objects should relate to the object sizes.

Furthermore, we explored the region of size in which the *zero-reuse* objects appear. We plotted the distribution of *zero-reuse* objects under different cache sizes for three traces. As shown in Fig. 4, the curve represents the cumulative distribution of *zero-reuse* object, where the curve's cutting point with the highest slope value corresponds to the region with the highest concentration of *zero-reuse* objects, the column represents the percentage of *zero-reuse* objects in all accessed objects in the size region (*e.g.*, the horizontal coordinate of the bar is 10^i, which represents the region $(10^{i-1}, 10^i]$). According to Fig. 4, we observed 3 phenomena.

(1) The curves vary with the traces while they are insensitive to cache sizes. It means that the trace's size distribution determines the distribution of *zero-reuse* objects, which is almost independent of cache size.

(2) The percentages vary with the traces, while increasing cache sizes will not significantly change the overall trend of the percentages, but will reduce the values within each size region. It proves that the cache size has an impact on the number of *zero-reuse* objects.

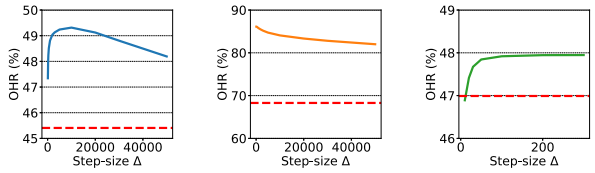
(3) The percentage of *zero-reuse* objects increases continuously from the first inflection point of the CDF curve and may decrease sharply in the last region. Re-tracking the data reveals that the number of objects in each declining region (*i.e.*, $(10^6, 10^7]$ and $(10^7, 10^8]$ on Trace-T, and $(10^3, 10^4]$ on Twitter) accounts for 0.12%, 0.0074%, and 0.52% of the total object volume, respectively. We believe that the decrease in percentage may be a random behavior caused by the small amount of data. As a result, there will be more *zero-reuse* objects in the group of larger objects.

This evidence confirms the correlation between the *zero-reuse* objects and the object sizes, motivating us to continue the exploration of insertion policy in a size-aware manner.

IV. Adaptive Size-aware Cache Insertion Policy

A. Bimodal Insertion Policy with Probability Function

The probabilistic parameters have stronger robustness compared with deterministic parameters. The Bimodal Insertion Policy (BIP) [8] performs different policies based on a probabilistic parameter, *i.e.*, $0 \leq \alpha \leq 1$, for the incoming objects. When $\alpha > 0.5$, BIP inserts the incoming object into the MRU position, otherwise into the LRU position. Due to the time saved in traversing the queue, selecting only the LRU and MRU positions for insertion is the most friendly policy for cache efficiency. Based on this policy and the phenomenon shown in SectionIII, *i.e.*, the small objects have higher access frequency than large objects, we believe that the relatively large objects should be inserted into the LRU position with a greater probability, where the probability should be a function of the object size. As a result, we select a probability function that is exponentially decreasing in the object size. Let c and $s(O)$ denote the size threshold and the size of the object O, the probability function of inserting O to the MRU position is $p^{MRU}(O) = e^{-s(O)/c}$ [1].



(a) Trace-T (b) Wikipedia (c) Twitter

Fig. 5. The solid blue line, yellow line, and green line represent the results of ASC-IP's OHR varying by Δ on three traces respectively. The dotted red lines in three sub-images ($y = 45.0648\%$, $y = 68.2829\%$, and $y = 46.9908\%$) are the baselines generated by the LRU algorithm. These results are generated on Trace-T, Wikipedia, and *Twitter-T*, respectively, where the cache sizes are $0.5\%\times$1024GB, $0.5\%\times$512GB, and $0.5\%\times$4GB, respectively. 1/5 of each trace is used for warmup.

B. Dynamic Tuning with History List

Referring to ARC, we construct a shadow cache with the virtual same capacity as the real cache to maintain a history list. In a FIFO manner, the history list only receives metadata (such as key and size) of the objects evicted from the real cache. The key and size are used to express which objects are in the history list, and the size is used to calculate whether the virtual capacity can carry them.

Facing the object that is about to enter the history list, we will first judge whether it is already in the history list. If it exists, this object may be inappropriate to insert into the LRU position. Since the shadow cache has the same size as the real cache, this means that the object may be hit after it is inserted into the MRU position. If the policy incorrectly placed the object in the LRU position, the size threshold should rise to ensure that an object with the same size is not recognized as a *zero-reused* object. On the contrary, when an object is inserted into the MRU position but is *zero-reuse*, the size threshold should decrease, making more objects determined to be *zero-reuse* objects. As a result, we can achieve dynamic adaptation by the size threshold.

C. ASC-IP

Based on the above designs, we propose an adaptive size-aware cache insertion policy (ASC-IP) to adapt to the dynamic workloads and variable object sizes in CDNs. We show the pseudo code of ASC-IP in Algorithm 1. ASC-IP can adapt to different object sizes by constantly adjusting c dynamically. In particular, we add two Boolean labels, *i.e.*, hit_tag and mru_tag into each object's metadata, where hit_tag represents the object whether being hit in the cache in a time window, and mru_tag denotes the object whether being inserted into the MRU position. By default, $hit\_tag = False$ and $mru\_tag = True$. If the object is hit, $hit\_tag = True$. When the object is coming to the cache, if it is inserted into the LRU position, the $mru\_tag = False$. For each object in or out of the cache, there are only four states of (mru_tag, hit_tag), including (T, T), (T, F), (F, T), and (F, F).

As shown in Algorithm 1, the content in line 6 occurs in the situation (T, F). O^e does not contribute any OHR but is inserted into the MRU position. Therefore, the size threshold

Algorithm 1 Adaptive Size-aware Cache Insertion Policy (ASC-IP)

Definition: O denotes an object, O^e denotes the object at the LRU position, H denotes the history list, C denotes the cache, c is the size threshold, Δ is the adjustment step of c, $s(*)$ represents the size of *, $s^u(*)$ represents the used size of *, $t^{MRU}(O)$ denotes *mru_tag* of O, $t^{hit}(O)$ denotes *hit_tag* of O, $p^{MRU}(O) = e^{-s(O)/c}$, $\gamma \in [0, 1]$ is a real-time random number.

Input: Incoming object O_i

1: **if** O_i not in C **then**
2: **while** $s(C) < s(O_i) + s^u(C)$ **do**
3: Evict O^e from C;
4: **if** $t^{MRU}(O^e) == True$ **then**
5: **if** $t^{hit}(O^e) == False$ **then**
6: $c = c - \Delta$;
7: **end if**
8: **else**
9: **if** $t^{hit}(O^e) == False$ and O^e in H **then**
10: $c = c + \Delta$;
11: **end if**
12: **end if**
13: **if** O^e in H **then**
14: Delete O^e from H;
15: **end if**
16: Add O^e to H;
17: **end while**
18: **if** $s(O_i) \geq c$ and $p^{MRU}(O_i) \leq \gamma$ **then**
19: Insert O_i into the LRU position;
20: $t^{MRU}(O_i) = False$; $t^{hit}(O_i) = False$;
21: **else**
22: Insert O_i into the MRU position;
23: $t^{MRU}(O_i) = True$; $t^{hit}(O_i) = False$;
24: **end if**
25: **else**
26: Move O_i to the MRU position;
27: $t^{hit}(O_i) = True$;
28: **end if**
29: **if** O_i in H **then**
30: Delete O_i from H;
31: **end if**

c should reduce by an adjustment step. The ensuing incoming large objects are more likely to be inserted into the LRU position and evicted more quickly.

The content in line 10 occurs in the situation (F, F). If O^e exists in the history list, the size threshold c must increase by an adjustment step so that the next incoming object with such size is more likely inserted into the MRU position. Finally, once "hit" the object in the history list, the corresponding item has to be deleted.

To determine the adjustment step of the size threshold, *i.e.*, Δ, based on the LRU algorithm and $c = 100$, we vary Δ from 100 to 50000 on Trace-T and Wikipedia, and vary Δ from 10 to 300 on Twitter. As shown in Fig. 5, the peak values

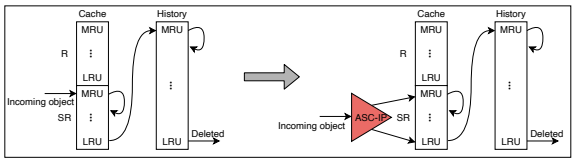


Fig. 6. Improving the insertion policy of CACHEUS (SR-LRU) by ASC-IP.

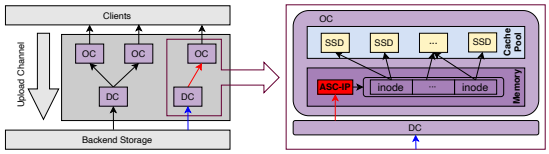
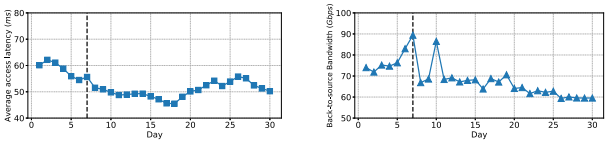


Fig. 7. The cache architecture of TDC. The cache layer consists of the data center cache (DC) layer and the outside cache (OC) layer.



(a) Latency (b) Bandwidth

Fig. 8. Latency and bandwidth in the monitoring system for a month. Note that LRU was replaced by ASC-IP-LRU at 24:00 on day-7.

are 49.315%, 86.765%, and 47.941% for Trace-T, Wikipedia, and Twitter, respectively. The corresponding Δ are 10000, 1000, and 200. Based on these adjustment steps, we vary c from 10 to 50000 on Trace-T and Wikipedia, and vary c from 10 to 1000 on Twitter. We are surprised to find that the OHR of each trace slight waves. The average values are 49.315%$\pm$0.004%, 86.766%$\pm$0.005%, and 47.945%$\pm$0.006% for Trace-T, Wikipedia, and Twitter, respectively. For the three traces, the final value of c will be 30100, 100, and 300, respectively, no matter the initial values. These results show that our adaptive scheme is insensitive to the initial value of c and that the adjustment process is robust.

V. Integration With Cache Replacement Algorithms

As an insertion policy, ASC-IP can decide the initial position of the incoming object in the cache. It not only does not interfere with the eviction policies of existing cache replacement algorithms but also can improve their insertion policies, resulting in the adaptation to scenarios with inconsistent object sizes. ASC-IP can easily replace the insertion policy of replacement algorithms based on LRU, *e.g.*, LRU, ARC [6], S4LRU [3], CACHEUS [9], *etc.*

We select the latest algorithm, *i.e.*, CACHEUS, as a case study for integration. CACHEUS consists of a cache list and a history list, where the records from the history list decide the cache list exerts the churn-resistant LFU algorithm or the scan-resistant LRU (SR-LRU) algorithm. When CACHEUS exerts the latter, the cache list divides two parts, *i.e.*, R partition, and SR partition. The former contains only objects with multiple accesses and the latter stores single access objects as well as older objects that are demoted from the R partition. The incoming object is inserted into the MRU position of the SR partition and SR-LRU evicts the object in the LRU position of SR on a cache miss when the cache is full. The evicted objects will enter the history list.

As shown in Fig. 6, we deploy ASC-IP into the SR partition and call it ASC-IP-SR-LRU. ASC-IP shares information from the SR-LRU's history list and adds *hit_tag* and *mru_tag* into metadata. When an object is coming to the SR partition, ASC-IP-SR-LRU will determine its insertion position by the object's size and update the object's metadata as well as information in the history list.

VI. Implementation and Improvement on TDC

We deploy ASC-IP to T Disk Cache (TDC) and show its architecture in Fig. 7, where TDC is a cache system of Company-T and serves 2500+ businesses. TDC's prototype is a storage node based on MCP++, multi-ccd/multi-smcd process model, raw disk, inode, and asynchronous disk I/O technologies such as libaio/SPDK, which supports cache replacement algorithms and provides key-value storage services. In the system, *inode*s are used to manage metadata of objects, including an MD5 index, an integer variable to record the object size, forward and backward pointers to the *inode*, and two Boolean variables (mru_tag and hit_tag) to record the insertion positions and hits, *etc.* The size of metadata is about 110B. The maximum *inode*'s number allowed by the system is 480,0000. It can support an average C2C size of 150KB based on the SSD's capacity of 7.2TB. Those *inode*s consume 4.9GB of memory. Although placing indexes on SSDs can meet the requirements of high-speed response, the frequent deletion of indexes reduces the SSD life and may incur system disk failures. As a result, we place the indexes in the memory.

In practice, a shadow cache is a history list to record the metadata of these evicted objects. The memory overheads are low since only a few metadata must be tracked, including the key of the object (*string*) and object size (*long int*). Furthermore, since engineers have deployed LRU in TDC, we merely replace LRU's insertion policy with ASC-IP.

In Fig. 8, from the monitoring system, we show the wave in average user access latency and the back-to-source bandwidth for a month, where the time span includes the week before the deployment of ASC-IP-LRU and about three weeks after the deployment. It is clear that after deploying ASC-IP-LRU (black dashed line), the average user access latency dropped by 7.14ms, and the back-to-source bandwidth dropped by 8.75Gbps. Before our scheme, engineers could frequently expand and reduce cache sizes to alleviate sudden performance fluctuations, which increased the cost of O&M workers. Because the cache capacity is huge (up to 3000TB), adding a 100TB SSD can only improve the OHR by 1%, *i.e.*, saving the bandwidth by 1.13Gbps and dropping the average

TABLE I
SUMMARY OF THE THREE TRACES THAT ARE USED IN OUR EXPERIMENTS.

	Trace-T	Wikipedia	Twitter
Duration Days	7	15	2
Total Requests (Millions)	7882.19	2800	4032.97
Unique Object (Millions)	1238.43	37.53	106.67
Total Trace File Size (GB)	166	54	66
Object Size(Max) (B)	30281600	1276931504	9401
Object Size(Min) (B)	1	10	1
Working Set Size (GB)	4028	2048	16

latency by 0.77ms. These results show that ASC-IP is a more economical and nothing-to-worry solution than the traditional scheme.

VII. EVALUATION

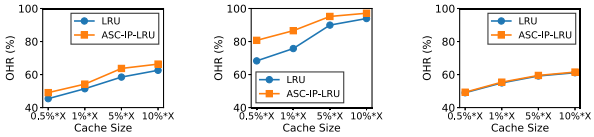
A. Settings

Traces. We test ASC-IP on two public CDN traces, *i.e.*, Wikipedia and Twitter, and a real CDN trace, *i.e.*, Trace-T. Traces' detailed attributions are shown in TABLE I, where the working set size means the size of all unique objects [16]. Note that the cache size should align with the working set size of the trace since the cache size is a parameter based on resource availability and system requirement. Based on the working set size shown in TABLE I and the 8:2 rule, we set cache sizes (*i.e.*, X) used to test on the simulator as 1024GB, 512GB, and 4GB for Trace-T, Wikipedia, and Twitter, respectively.

Sampling Method. We extract training data from log files and transform them into vectors. Since the Trace-T contains about 7.88 billion requests during a 7-day periods, we use the reservoir sampling method [12] to select a subset. Specifically, we extract all unique photo IDs and sample out 1 of the IDs to form an ID set. Based on this 100 ID set, we obtain our training data by sequential extraction.

State-of-the-art insertion policies. We compare ASC-IP with LIP, BIP, Dynamic Insertion Policy (DIP) [8], Promotion/insertion pseudo-partitioning (PIPP) [14], and Deadblock Aware Adaptive Insertion Policy (DAAIP) [5]. DIP exerts BIP or LRU depending on the miss ratio. PIPP combines dynamic insertion and promotion policies to provide the benefits of cache partitioning, adaptive insertion, and capacity stealing all with a single mechanism. DAAIP protects the data of applications having high temporal locality from high access rate thrashing/streaming applications by monitoring each application at runtime using cost-effective hardware circuits.

State-of-the-art replacement algorithms. Our experiments will discuss integration to advanced replacement algorithms, including S4LRU [3], ARC [6], and CACHEUS [9]. S4LRU is a typical variant of LRU. Its specific implementation is actually to divide the LRU chain into four sections. By promoting objects hitting the previous section to a higher section, the hot data can retain in S4LRU for longer while the cold data can leave the cache quickly. ARC is the best example of shadow caching. CACHEUS has been mentioned in Section V. Specifically, ASC-IP is used to replace the insertion policy in the replacement algorithms without other things changed.



(a) Trace-T (b) Wikipedia (c) Twitter

Fig. 9. OHRs under different cache sizes and traces.

TABLE II
RESOURCE USAGE FOR LRU AND ASC-IP-LRU IN THROUGHPUT-BOUND (MAX) AND PRODUCTION-SPEED (NORMAL) EXPERIMENTS.

Metric	LRU	ASC-IP-LRU
Throughput (Gbps)	1.57	1.55
Peak CPU (%)	6.3	6.4
Peak Memory (GB)	0.096	0.224

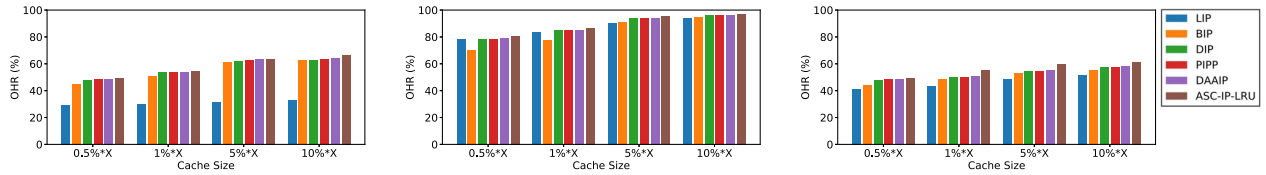
State-of-the-art admission algorithms. To compare the effects between entry prohibited and giving one chance for suspected *zero-reuse* objects, we designed the comparisons between ASC-IP and classic admission algorithms such as AdaptSize [1] and One-time-access-criteria [13]. AdaptSize, the first adaptive size-aware cache admission algorithm for Hot Object Cache (HOC). The core of AdaptSize is a novel Markov cache model that seamlessly adapts the caching parameters to the changing access patterns. AdaptSize is a typical admission algorithm based on heuristics, representing a class of admission algorithms, including 2Q [10], TinyLFU [2], *etc*. One-time-access-criteria designs a classifier model to facilitate the policy. It integrates a decision-tree model into the classifier, extracting social-related information as classifying features and applying cost-sensitive learning to improve classification precision. One-time-access-criteria represents the admission algorithm based on machine learning models.

State-of-the-art research in CDN caching. We select three recent CDN cache algorithms, *i.e.*, AdaptSize, LRB, and RLB, for comparison. These algorithms all automatically sense the object size. Note that we will compare them in different subsections of the experiment section.

Simulator and Testbed. We deploy the simulator based on CACHEUS [9]. We implement ASC-LRU, BIP, DIP, DAAIP *etc*on this simulator. To ensure a fair and meaningful comparison, we launch the cache algorithms on the empty cache queue at the same time for both servers. The configurations of servers are SH1-10G with 2 Intel Xeon Silver 4110 CPU, 64GB RAM, and four Intel P4510 Series SSDs with 1.5TB capacity. In addition, our simulator is running with two 8-core AMD CPUs, 32GB RAM, and 1.5TB SSD.

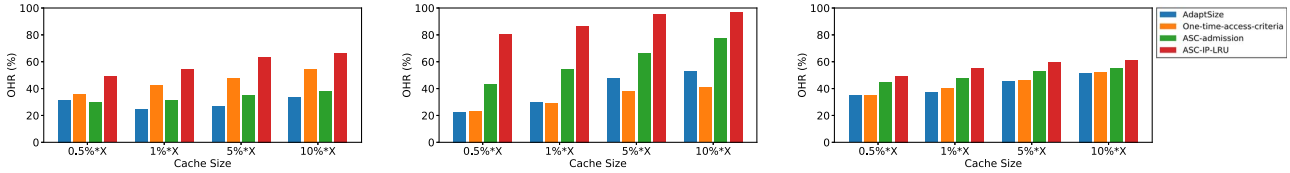
B. Compare ASC-IP-LRU with LRU

To verify performance compared to the basis, we evaluated the OHRs of LRU and ASC-IP-LRU on the simulator. As shown in Fig. 9(a), X equals 1024GB. Comparing with LRU, the SC-IP-LRU algorithm can improve the OHRs by 2.75% and 2.35% on 0.5%×1024GB and 1%×1024GB. As shown



(a) Trace-T (b) Wikipedia (c) Twitter

Fig. 10. Comparing ASC-IP with state-of-the-art insertion policies based on LRU on three traces.



(a) Trace-T (b) Wikipedia (c) Twitter

Fig. 11. ASC-IP-LRU vs state-of-the-art admission algorithms on CDNs.

in Fig. 9(b), X equals 512GB. The SC-IP-LRU algorithm can improve the OHRs by 10.01% and 9.20% on 0.5%×512GB and 1%×512GB. Similarly, ASC-IP-LRU can improve the OHRs by 11.50% and 9.27%. In Fig. 9(c), X is 4GB. The SC-IP-LRU algorithm can improve the OHRs by 0.09%, 0.07% 0.05% and 0.03%. Similarly, ASC-IP can improve the OHRs by 0.31%, 0.28%, 0.23% and 0.19%.

It is clear that the promotion by ASC-IP decreases as the cache size increases. The reason is that as the cache size increases, the number of *zero-reuse* objects reduces. Note that in the real CDN environment, OHR can be raised by increasing storage devices and expanding cache space. However, the additional equipment will cost more money. As a result, ASC-IP can improve performance with little cost, which is the value and motivation of this paper.

In addition, we compare the overheads of ASC-IP-LRU against LRU. We run the cache algorithms in the simulator and show the results in Table II. We monitored the online metrics of the ASC-IP-LRU deployment in a week, where the metric data include throughput, peak CPU, and memory. Since ASC-IP-LRU requires an extra calculation of a threshold comparing with LRU, the peak CPU utilization of ASC-IP-LRU is 6.4% and the counterpart of LRU is 6.3%. In terms of storage overheads, ASC-IP-LRU requires additional space to store a history list and two Boolean variables for each object. The space overheads are shown in the Table II. The peak memory of ASC-IP-LRU overtakes 0.4% than the counterpart of LRU. Finally, the gap of throughput between ASC-IP and LRU is almost negligible.

C. ASC-IP vs. State-of-the-art Insertion Policies

We compare ASC-IP with the state-of-the-art insertion policies, including LIP, BIP, DIP, PPIP, and DAAIP in terms of OHR. We test them based on LRU over three traces. As shown in Fig. 10, ASC-IP outperforms other insertion policies. The average benefits compared to DAAIP are 8.36%, 3.22%, and 5.92% for Trace-T, Wikipedia, and Twitter, respectively.

In addition, we record the average time to respond to a request for the different algorithms on the simulator. To be fair, we only consider the time from decision to insertion without the time overheads of disk reads and writes. The experimental results show that the time of ASC-IP-LRU and LIP, BIP, and DIP are 3.9μs, 2.72μs, 2.73μs, and 4.3μs, respectively. The time overhead of PIPP is 5.5μs, which is about twice that of ASC-IP-LRU. The time overhead of DAAIP is 34.8μs, which is about 10 times that of ASC-IP-LRU. As a result, we believe that ASC-IP can achieve a high hit ratio efficiently.

D. ASC-IP vs. Admission Algorithms working on CDNs

To demonstrate the necessity of giving *zero-reuse* objects at least one chance, we compare ASC-IP-LRU and ASC-IP based on the admission pattern with the state-of-the-art admission algorithms, including AdaptSize [1] and One-time-access-criteria [13]. Note that, to fairly explain the necessity, we use our size-aware scheme in the admission algorithm. Specifically, we prohibited the objects that should be inserted into the LRU position from entering the cache. We call this algorithm ASC-Admission. As shown in Fig. 11, ASC-IP-LRU outperforms other admission algorithms, including ASC-Admission. The performance gap is particularly evident in the Wikipedia trace. In addition, the OHRs of ASC-IP-LRU increase by 1.76%, 2.42%, 0.35% on average compared to counterparts of AdaptSize, and increases by 0.34%, 0.19% and 0.72% on average compared to counterparts of one-time-access-criteria, on three traces, respectively. These results verify our hypothesis that giving the suspected *zero-reuse* object a chance to be hit results in the rise of OHR. However, ASC-admission ignores most of these objects accessed twice or more in Trace-T, resulting in lower performance compared to the other two traces. In addition, the throughput generated by ASC-IP-LRU is 1.57Gbps and the counterparts by ASC-Admission, AdaptSize, and One-time-access-criteria are 1.58Gbps, 0.628Gbps, and 0.24Gbps, respectively.

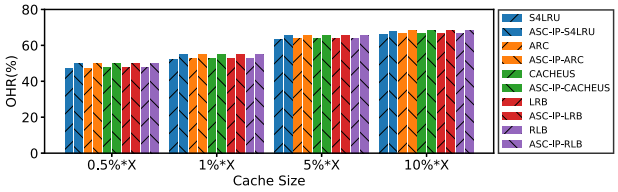


Fig. 12. The comparisons of cache replacement algorithms and their ASC-IP versions.

E. Upgrading replacement algorithms working on CDNs

Since the eviction policy and the insertion policy can be thought as two independent policies, we consider ASC-IP as a plug-in. We believe that ASC-IP can be applied to LRU-based replacement algorithms and improve the OHR. To verify this ability, we use ASC-IP to improve the insertion policy of S4LRU, ARC, CACHEUS, LRB, and RLB. The comparisons of OHRs of the advanced replacement algorithms and their improved versions are shown in Fig. 12. The results of OHR increase after deploying ASC-IP for each algorithm. The biggest increases are 2.70%, 2.71%, 2.47%, 2.27%, and 2.27% at $0.5\% \times X$ for Trace-T. The average counterparts for the four cache sizes are 2.24%, 1.93%, 2.01%, 2.06%, and 1.82% respectively. As a lightweight and generic policy, ASC-IP's plug-in capability can upgrade LRU-based cache algorithms with negligible overheads.

VIII. Related Work

Replacement Algorithms. Traditional caching algorithms mainly focus on the characterization of different data features (such as new progress, frequency, data size, etc.) by designing the storage structure. For example, compared with the most basic FIFO algorithm, LRU represents the characteristics of the new progress by promoting the hit objects in the cache to MRU, improving the hit ratio of the recently accessed objects. Another example is S4LRU [3], which divides the LRU chain into four segments and promotes them layer by layer through hits, keeping them in the cache for longer. For example, ARC [6], LIRS [4], CACHEUS [9], and others have improved performance by adding or adjusting the structure of the cache chain. [8] proposes an important idea that the problem of cache replacement can be divided into two parts: victim selection policy and insertion policy.

Admission Algorithms. Many researchers have focused on this issue and devoted size-aware to the admission algorithms. 2Q [10] decides all objects according to access frequency and only those accessed twice are allowed into the cache. TinyLFU [2] maintains an approximate representation of the access frequency of a large sample of recently accessed items. AdaptSize [1] models the cache as a Markov process that incorporates all correlations between object sizes and current access rates, allowing to find the optimal size-aware admission policy. Hua *et al*. [13] propose to predict one-time-access objects by the decision-tree model learning features of size, determining admission. These policies are efficient when predicting accurately. Otherwise, they will cause the object to lose the chance to be hit.

IX. Conclusion

We are concerning the lightweight and general cache algorithms that can improve the OHR performance with negligible overheads. In this paper, to reduce the consumption of cache resources by *zero-reuse* objects in CDNs, we explored the necessity of improving the insertion policy. It is necessary to give all objects at least one chance to be hit if the accuracy of the prediction cannot be ensured. Given the large size variation in CDNs, we analyzed and found the relationship between the *zero-reuse* object and the object sizes. Based on these cues, we proposed an adaptive size-aware cache insertion policy (ASC-IP). ASC-IP has strong practicality. It not only achieves a high hit ratio in CDNs with diverse object sizes but also can be easily integrated into LRU-based replacement algorithms to upgrade them with little overhead. We demonstrate the superiority of ASC-IP on two public traces and our real-world trace. We expect more researchers to pay attention to ASC-IP and benefit from it.

References

[1] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network," in NSDI, USENIX, pp. 483–498, 2017.
[2] G. Einziger, R. Friedman, and B. Manes, "Tinylfu: A highly efficient cache admission policy," in TOS, ACM, vol. 13, no. 4, pp. 1–31, 2017.
[3] Q. Huang, K. Birman, R. R. Van, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of Facebook photo caching," in SOSP, ACM, pp. 167–181, 2013.
[4] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," in SIGMETRICS, ACM, vol. 30, no. 1, pp. 31–42, 2002.
[5] S. K. Mahto, S. Pai, and V. Singh, "DAAIP: Deadblock aware adaptive insertion policy for high performance caching," in ICCD, IEEE, pp. 345–352, 2017.
[6] N. Megiddo and D. S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache," in FAST, 2003.
[7] R. Motwani and P. Raghavan, "Randomized algorithms," in CSUR, ACM, vol. 28, no. 1, pp. 33-37, 1996.
[8] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in SIGARCH Computer Architecture News, ACM, vol 35, no. 2, pp. 381–391, 2007.
[9] L. V. Rodriguez, F. Yusuf, S. Lyons, E. Paz, R. Rangaswami, J. Liu, M. Zhao, and G. Narasimhan, "Learning cache replacement with CACHEUS," in FAST, pp. 341–354, 2021.
[10] D. Shasha and T. Johnson, "2q: A low overhead high performance buffer management replacement algorithm," in VLDB, pp. 439–450, 1994.
[11] Z. Song, D. S. Berger, K. Li, A. Shaikh, W. Lloyd, S. Ghorbani, C. Kim, A. Akella, A. Krishnamurthy, and E. Witchel, "Learning relaxed belady for content distribution network caching," in NSDI, USENIX, pp. 529–544, 2020.
[12] J. S. Vitter, "Random sampling with a reservoir," in TOMS, ACM, vol. 11, no. 1, pp. 37–57, 1985.
[13] H. Wang, X. Yi, P. Huang, B. Cheng, and K. Zhou, "Efficient SSD caching by avoiding unnecessary writes using machine learning," in ICPP, pp. 1–10, 2018.
[14] Y. Xie and G. H. Loh, "PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches," in ACM SIGARCH Computer Architecture News, vol. 37, no. 3, pp. 174–183, 2009.
[15] G. Yan and J. Li, "Rl-Bélády: A unified learning framework for content caching," in MM, ACM, pp. 1009–1017, 2020.
[16] J. Yang, Y. Yue, and K. V. Rashmi, "A large scale analysis of hundreds of in-memory cache clusters at Twitter," in OSDI, USENIX, pp. 191–208, 2020.