FLOWS: Balanced MRC Profiling for Heterogeneous Object-Size Cache

Xiaojun Guo[§,†], Hua Wang[§✉], Ke Zhou[§], Hong Jiang[‡], Yaodong Han[§], Guangjie Xing[§]
§Huazhong University of Science and Technology, China
†Tencent Inc., China
‡University of Texas at Arlington, USA
{xjguo,hwang,zhke}@hust.edu.cn,hong.jiang@uta.edu,{ydhan,guntherxing}@hust.edu.cn

Abstract

While Miss Ratio Curve (MRC) profiling methods based on spatial sampling are effective in modeling cache behaviors, previous MRC studies lack in-depth analysis of profiling errors and primarily target homogeneous object-size scenarios. This has caused imbalanced errors of existing MRC approaches when employed in heterogeneous object-size caches. For instance, in CDN traces, the error of the Byte Miss Ratio Curve (BMRC) could be two orders of magnitude larger than that of the Object Miss Ratio Curve (OMRC).

In this paper, we reveal an important insight from our experimental analysis, namely, the source of profiling inaccuracy is twofold, the "imbalanced requests" and the heterogeneous object-size distribution. To this end, we propose *FLOWS*, a *F*iltered *LO*w-variance *W*eighted *S*ampling approach, to address the root causes of the problem by combining a Cache Filter, designed to balance sampled requests, with a Weighted Sampling technique, designed to reduce byte-level estimation error. *FLOWS* constructs a more accurate MRC for traces with heterogeneous content popularity and object sizes. Evaluation on real-world traces demonstrates that *FLOWS* reduces the error of the BMRC and OMRC profiling by 16× and 3×, respectively, compared with state-of-the-art approaches. Additionally, *FLOWS* enables cache systems to effectively balance the Byte Hit Ratio (BHR) and Object Hit Ratio (OHR), achieving an improvement of up to 26.5% in overall hit rate compared to other methods.

CCS Concepts: • **Information systems → Storage management**.

Keywords: MRC profiling, object cache, cache optimization

EuroSys '24, April 22–25, 2024, Athens, Greece

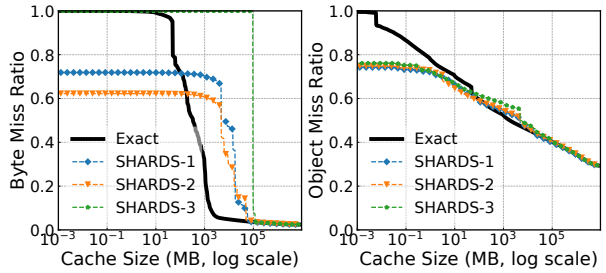
ACM ISBN 979-8-4007-0437-6/24/04…$15.00
https://doi.org/10.1145/3627703.3650078

1 Introduction

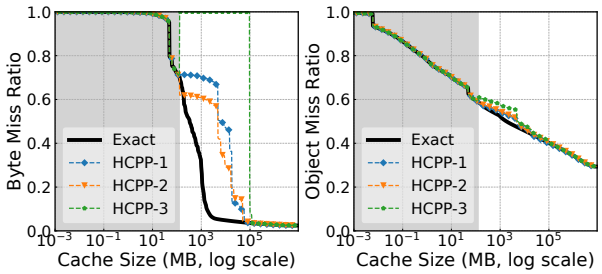
Data caching plays a crucial role in web service optimization by reducing the amount of data transferred over the network, leading to lower bandwidth costs and improved user experience. Content Delivery Networks (CDNs) are the primary providers of web caching services, accounting for over 50% of the total Internet traffic [6]. Therefore, understanding and optimizing data caching techniques are essential for delivering better services and enhancing performance. When evaluating the performance of an object-level cache, two commonly used metrics are Byte Miss Ratio (BMR) and Object Miss Ratio (OMR). OMR serves as an indicator of the miss ratio for our accesses and is closely linked to the resulting access latency [5], whereas BMR holds particular significance on bandwidth overhead and associated costs for the cache provider [54].

In general, a cache system should be capable of supporting and managing multiple caching services [41]. A more effective cache policy can lead to a higher hit ratio for a particular service, while better management of cache resources can result in improved overall performance for all services that share the cache. Miss Ratio Curve (MRC) is a quantitative model to characterize cache performance, where the x-axis represents the cache size and the y-axis represents the miss ratio. Figure 1 shows the Byte Miss Ratio Curve (BMRC) and Object Miss Ratio Curve (OMRC) of a CDN trace [1] under the Least Recently Used (LRU) replacement policy. MRC plays a vital role in cache resource management methods, such as cache allocation [14, 16, 72], cache sharing [14], and traffic provisioning [54].

Exact MRC profiling requires trace replaying, which can incur high space and time overhead [39]. Previous studies have proposed approximate MRC profiling methods to reduce overhead [18, 24, 27, 30, 60, 62, 70, 72], primarily focusing on the LRU replacement policy since the reuse distance is a reliable intermediate variable. Spatial (hash) sampling based MRC profiling is the most representative algorithm [18, 59, 60], which uses a sampled subset of the working set to approximate cache performance, thus significantly reducing the overhead compared to a complete trace enumeration, e.g., Footprint Descriptors [53]. However, this method may not be applicable to real-world object cache traces, as evidenced by Figure 1(a) that reveals two major problems with



(a) MRC Profiling with SHARDS.



(b) MRC Profiling with HCPP.

Figure 1. The exact and profiled LRU BMRC and OMRC of the MetaCDN-rega [1] trace. When using a sampling rate of $r = 0.01$, the SHARDS [60] and HCPP [18] exhibit two weaknesses: 1) incorrect construction of the BMRC, and 2) inaccurate OMRC.

spatial sampling: 1) inaccurate front portion MRC profiling, due to heterogeneous content popularity, i.e., request probability; and 2) significant profiling error disparity between BMRC and OMRC, caused by heterogeneous object sizes. The state-of-the-art method proposed by Carra and Neglia [18] solves the first problem, as shown in Figure 1(b), but still suffers from the second problem. For ease of description in the subsequent text, we refer to this method as Heterogeneous Content Popularity cache Profiling (HCPP).

In this paper, we focus on the MRC profiling of real-world object caches. We find that the inaccuracy of the MRC profiling is due to the imbalanced sampling of requests and bytes under heterogeneous content popularity and heterogeneous object-size. To address this problem, we propose *FLOWS* (Filtered LOw-variance Weighted Sampling), which can achieve highly accurate BMRC and OMRC profiling with no additional cost compared to existing spatial sampling methods. Our contributions are summarized as follows:

1. We model spatial sampling based MRC profiling techniques as a distributed cache system and point out that keeping load balanced is essential for the accuracy of these methods.
2. We propose *FLOWS* with two novel mechanisms, a Cache Filter and a Weighted Sampling technique, to overcome the imbalance caused by heterogeneity, thus resulting in accurate BMRC and OMRC profiling.
3. We evaluate the accuracy of *FLOWS* using 18 real-world traces. In comparison to state-of-the-art MRC profiling methods [18, 27, 60], our approach achieves an average error reduction of 16× in BMRC profiling and 3× in OMRC profiling, without incurring any additional space and time overhead.
4. The cache allocation experiments demonstrate that using *FLOWS* profiling can achieve a better balance between Byte Hit Ratio (BHR) and Object Hit Ratio (OHR) compared to the state-of-the-art algorithms. Our approach results in a maximum improvement of 26.5% in overall hit rate.

The rest of the paper is organized as follows. In Section 2, we discuss the motivation for *FLOWS*. Section 3 models the spatial sampling based MRC profiling methods as a distributed cache, and proposes a load-balancing mechanism to reduce the error of MRC profiling. The detailed algorithm for both BMRC and OMRC is described in Section 4. We evaluate our solution in Section 5. Section 6 discusses the value of using MRC and how to profile real cache systems. Section 7 provides an overview of related work. Finally, in Section 8, we conclude the paper.

2 Background and Motivation

2.1 MRC Profiling Methods

MRC is the metric that represents the cache miss ratio as a function of cache size. Most cache policies are unable to efficiently construct an MRC, with the exception of the LRU policy [39], which has the attribute of reuse distance. As shown in Figure 2, the reuse distance of an object refers to the total size of *unique* objects between its two consecutive accesses, with a reuse distance of infinity indicating that the object is the first time been requested. The reuse distance can also be considered as a Working Set Size (WSS) of consecutive requests, that is, the sum of the total *unique* object sizes. If the reuse distance is no larger than the LRU cache size, then the object will be hit in the cache.

The MRC of LRU is constructed as a discrete integration histogram that counts all requests' reuse distances (Figure 3). Using a search tree data structure is the most efficient way to calculate the exact reuse distance histogram, which requires $O(M)$ space and $O(Nlog(M))$ time, where N is the length of the trace and M is the number of unique objects [73]. This high cost makes exact MRC profiling impractical [10, 55]. Spatial sampling methods, such as SHARDS [60] and HCPP [18], have been shown effective in MRC profiling. These methods employ selective counting of objects based on their hash values. They estimate the impact of a request being a hit or a miss (estimate size), as well as the required cache capacity to ensure that the request hits the cache (reuse distance). This approach helps in reducing computational and memory overhead (refer to Figure 2).

The accuracy of spatial sampling methods can be significantly impacted by the popularity and size of objects. For instance, if a highly popular object A is not sampled, the

Exact Reuse Distance

ObjectID	X	A	B	A	C	A	D	A	E	F	G	F	X
Size	1	2	3	2	4	2	5	2	7	12	6	12	1
Reuse Distance	INF	INF	INF	5	INF	6	INF	7	INF	INF	INF	18	40
Hash	0.5	0.6	0.1	0.6	0.3	0.6	0.7	0.6	0.2	0.9	0.4	0.9	0.5

Spatial Sampled Reuse Distance

ObjectID	X	B	C	E	G	X	?
Estimate Size	2	6	8	14	12	2	13
Reuse Distance	INF	INF	INF	INF	INF	42	1

Small Cache Reuse Distance

ObjectID	A	A	A
Size	2	2	2
Reuse Distance	5	6	7

Figure 2. The reuse distance for object cache. When using spatial sampling based methods with a sample rate of 0.5, only XBCEG are sampled. With a small cache of size 10, all reuse distances less than 10 are counted.

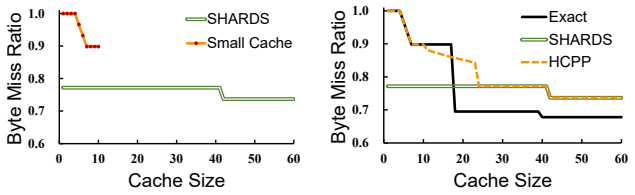


Figure 3. By integrating size over reuse distances, we obtain the BMRC. SHARDS [60] only uses spatial sampled reuse distance, and the HCPP [18] uses small cache reuse distance.

reuse distance histogram in SHARDS will fail to capture the low reuse distance portion of object A. Similarly, if object F, which has a large size, is not sampled, the SHARDS method cannot accurately estimate the required cache space for achieving a hit on that object.

To overcome this challenge, SHARDS employs a compensation method for improper sampling. For example, if the original trace contains a total of 57 bytes, while the sampled trace only includes 44 bytes, SHARDS assumes that 13 bytes will be hit in a smaller cache. To account for this, SHARDS adjusts the histogram by adding a bin with a reuse distance of 1 and a size of 13. The MRC resulting from this compensation is shown on the left side of Figure 3.

The issue of inaccurate front portion MRC caused by heterogeneous content popularity was observed in the research conducted by Carra and Neglia [18]. To address this problem, they propose the HCPP method. This method utilizes a small fixed-size cache to construct an exact front portion of the MRC, which is then concatenated with the MRC profiled by SHARDS, as illustrated on the right side of Figure 3.

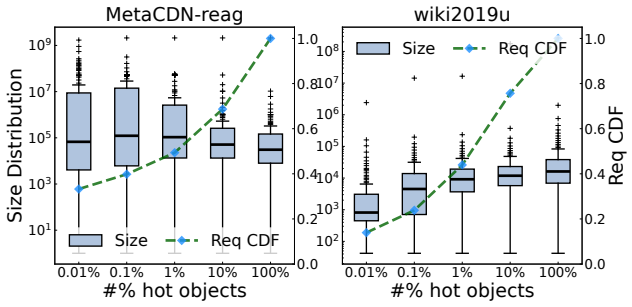


Figure 4. Popularity and size distribution in MetaCDN-reag and wiki2019u [51] trace. The x-axis represents the proportion of the hottest working set, while the y-axis on the left and right sides represents the object-size distribution and the cumulative request ratio, respectively.

Although this method can obtain a relatively accurate OMRC, the modeling error of BMRC is still significant (Figure 1(b)). As the BMR has a significant impact on the performance of web cache systems, using such a method for MRC profiling and guiding cache allocation may result in suboptimal cache performance.

2.2 Web Cache System

Modern web cache systems, such as Content Delivery Networks (CDNs), primarily focus on object-level caching services. The size of an object is often closely related to its popularity, and there is a significant heterogeneity in both popularity and size. Our CDN trace study, illustrated in Figure 4, shows the distributions of object sizes and popularity in two real-world traces. Notably, in the MetaCDN-reag [1] trace, the smallest object is nine orders of magnitude smaller than the largest object, indicating a substantial level of heterogeneity in object sizes. Additionally, the top 0.01% of the most popular objects account for approximately 40% of the overall requests. Such highly popular objects can significantly impact cache performance, especially when the cache size is relatively small.

Cache systems store the most popular contents to offer end-users low latency and high throughput. Origin offload, end-user latency, and midgress bandwidth cost are the three key metrics of a CDN system [54]. One of the primary objectives of cloud providers is to effectively reduce resource usage under the premise of satisfying the Quality of Service (QoS). BMR directly reflects bandwidth costs, therefore a high BMR will result in increased origin offload, end-user latency, and midgress traffic costs [54]. From this perspective, minimizing the BMR is a major objective in cloud cache service under the constraints of cache capacity.

Some caching systems and frameworks have embraced the concept of isolation and resource reallocation among multiple cache pools [9, 56]. In these approaches, each cache

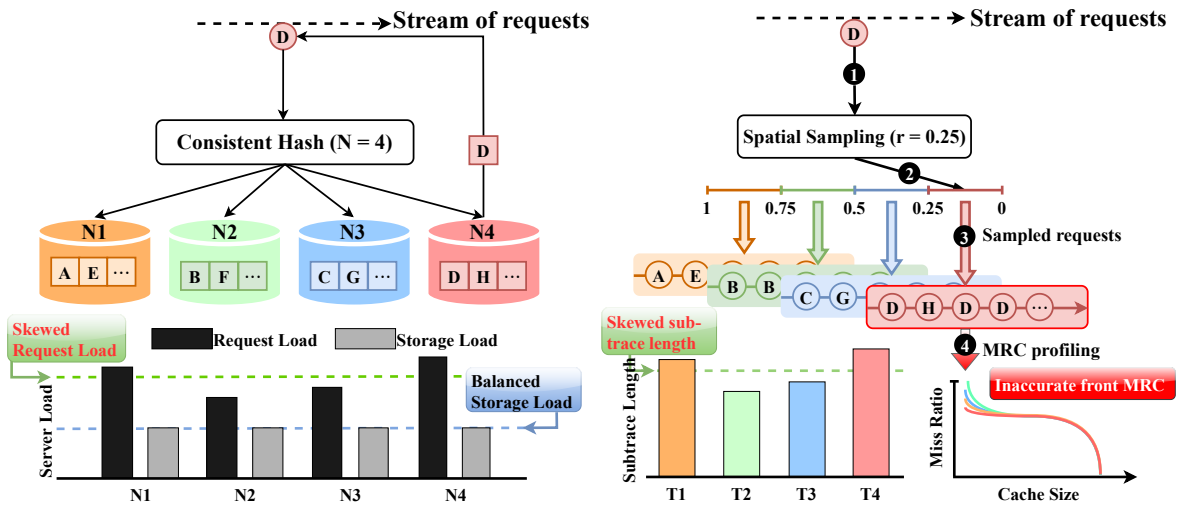


Figure 5. Distributed Cache (left) and Spatial Sampling (right).

pool assumes the responsibility of managing its own storage space. This strategy offers several advantages, such as reducing the footprint and achieving performance isolation. Since most cache instances demonstrate different MRCs, this practice often helps to improve the cache hit ratio.

2.3 Cache Resource Management

Cache resource management methods are vital in providing balanced and efficient cache services for multiple applications. To efficiently utilize cache resources in multi-application environments, a variety of solutions have been proposed from different perspectives, such as cache allocation [12, 14, 27, 29, 36, 46, 55, 57, 71, 72], cache sharing [14, 35, 45, 63] and traffic provisioning [53, 54]. Most of these solutions utilize MRC to achieve the desired optimization goals. However, existing methods face challenges in simultaneously constructing accurate BMRC and OMRC, which makes fine-grained and efficient management of cache resources impractical.

2.4 Our Work

In this study, we uncover how the heterogeneity of object sizes and popularity can significantly impact cache MRC profiling in spatial sampling. To mitigate this problem, we propose a solution based on load balancing principles. Our approach is designed especially for web caches that operate under heterogeneous object-size workloads. We demonstrate that our approach is capable of providing highly accurate and low-cost MRC profiling, which can be beneficial for such cache systems.

3 System model

In this section, we model the spatial sampling based method as a distributed cache system and propose a Cache Filter and a Weighted Sampling technique to address the inaccuracy of spatial sampling. We demonstrate that these techniques can be used to improve the accuracy of MRC profiling substantially. The symbols primarily used for analysis are summarized in Table 1.

Table 1. Symbol Summary Table

Symbol	Description
N	# of requests
M	# of objects
r	Sampling rate
K	# of nodes in an analogous distributed cache
B	Size of small cache in HCPP [18]
L	# of object cached in Cache Filter
s_i	Size of object i

3.1 From Load Balance to MRC Profiling

Figure 5 (left) demonstrates a typical four-node distributed cache system based on Consistent Hash. Each node in this system keeps data blocks hashed to it and serves their corresponding requests. Figure 5 (right) shows a spatial sampling based MRC profiling procedure. This procedure begins by mapping each object i to a random variable $hash(i) \in U(0,1)$(❶, ❷). Then, with a sampling rate of $r = 0.25$, only objects with hash values less than r are sampled(❸). Spatial sampling uses these sampled requests to construct an MRC(❹), such as SHARDS [60] for LRU and *Miniature Simulation* [59] for some Non-LRU policies.

The overall cache performance across different nodes in the consistent hash is the same as that of a pooled single virtual cache sharing a single LRU eviction list [31]. This means that spatial sampling with a sampling rate of r can

be modeled as a distributed cache with $K = 1/r$ nodes, and the overall cache performance is represented by randomly selecting one cache node.

Let us consider a workload trace with N requests and M unique objects. We use spatial sampling with a sampling rate of r, and the number of nodes in the corresponding distributed cache is $K = 1/r$. The basic SHARDS relies on two primary assumptions:

(1) Balanced Request: the number of sampled requests is close to $r \times N$ for accurate request estimation;
(2) Balanced Storage: the number of sampled unique objects is close to $r \times M$ for accurate reuse distances or WSS estimation.

While the storage load is almost balanced [47] for caches with homogeneous object sizes, due to heterogeneous content popularity, the request load may be skewed, leading to an imbalance in the number of requests (as seen in Figure 5 left). The latter has a significant impact on the variation of cache performance in the back-end nodes [25, 38]. Moreover, the skewed content popularity in spatial sampling can cause the sampled requests to deviate from $r \times N$, resulting in inaccurate MRC profiling (as seen in Figure 5 right).

3.2 Popularity Heterogeneity with Cache Filter

Despite the "Balanced Request" assumption made by spatial sampling, objects in real-world workloads have various popularity. Assuming that the sampled trace includes hot objects, the observed cache performance based on sampling will be much higher than the actual one. Conversely, if the sampled trace excludes hot objects, the observed cache performance will be abnormally lower [18].

To address the "load imbalance" issue, a partial solution called $\text{SHARDS}_{\text{adj}}$ was proposed by Waldspurger et al. [60]. In the subsequent sections, we will refer to it as SHARDS for brevity. It adds $N_e - N_s$ to the beginning of the reuse distance histogram, where $N_e = r \times N$ is the expected number of sampled requests, and N_s is the actual number of sampled requests. While this re-balancing scheme can lead to accurate miss ratio profiling for larger cache sizes, it can be inaccurate when the cache size is relatively small (Figure 1(a)). To address this issue, HCPP [18] proposed constructing an exact front portion MRC for cache capacity from 0 to B, where $B > 1/r$ (Figure 1(b)). This approach can provide a more accurate MRC profiling, but the configuration of parameter B is not specified in the original paper.

A small cache serving as a load balancer can significantly reduce the imbalance in a distributed cache system [25, 38]. Fan et al. [25] demonstrated that balancing the number of requests across all K back-end nodes can be achieved by caching only the $O(K \times \log(K))$ most frequently accessed objects, regardless of the total number of objects. In our study,

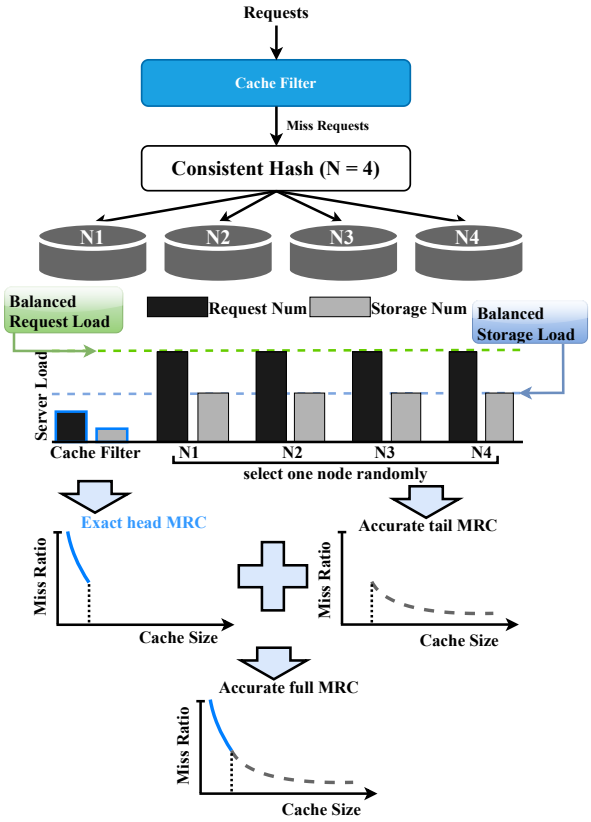


Figure 6. The Cache Filter serves as a load balancer for the spatial sampling process, ensuring that requests are distributed evenly.

conducted in Theorem 3.1, we also proved that caching high-popularity objects during the spatial sampling process can effectively bound the bias of sampled requests.

Theorem 3.1. *(proof in Appendix A) In a spatial sampling process with a sample rate of r, by caching the L most popular objects, the relative bias of the sampled requests is consistently bounded with a probability of $(1 - O(r^2))$. The value of L is determined by the following equation:*

$$L = O(\frac{1}{r} \log \frac{1}{r}). \tag{1}$$

Cache Filter. Based on the concept of small load balancing cache, we propose a Cache Filter technique, which consists of a small LRU cache of a fixed length L and a search tree to calculate reuse distances, is illustrated in Figure 6.

Cache Filter only caches L most recent accessed objects, and constructs the exact front portion MRC. Compared to the HCPP method, our proposed Cache Filter has two additional benefits: first, only missed requests affect the sampling phase, thus reducing the computational overhead of spatial sampling; second, the Cache Filter only maintains a fixed

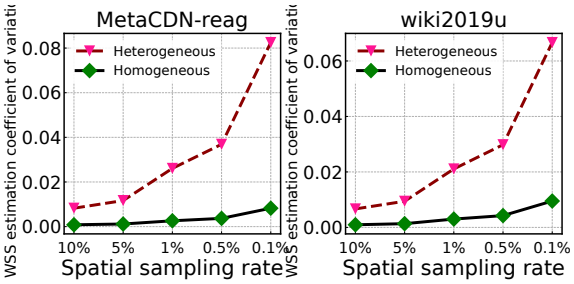


Figure 7. The coefficient of variation of WSS estimation is calculated for both heterogeneous-size and homogeneous-size traces, which can effectively capture byte-level profiling errors. The results are obtained from the MetaCDN-reag and wiki2019u [51] traces.

number of objects in the search tree, making it more efficient than the fixed size cache of HCPP.

3.3 Low-Variance Byte Estimation via Weighted Sampling

We employ the estimation of WSS as the target for subsequent error analysis. The WSS represents the sum of unique object sizes within the entire working set. In other words, it also denotes the reuse distance between the initial and final access within the trace. To assess the error in the WSS estimate, we use the coefficient of variation, providing an intuitive reflection of relative error. For a random variable S, its coefficient of variation is defined as:

$$Cv[S] = \frac{\sqrt{Variance[S]}}{\mathbb{E}[S]}. \tag{2}$$

The "Balanced Storage" assumption of spatial sampling implies that the WSS can be estimated accurately. However, in a scenario where the cache objects have heterogeneous sizes, this assumption may no longer be valid.

Theorem 3.2. *(proof in Appendix B) For a trace with M distinct objects, the sampling rate is r and the WSS estimated by spatial sampling is WSS_s. The coefficient of variation of WSS_s is given by:*

$$C_v[WSS_s] = \sqrt{\frac{1-r}{rM} \times \frac{\mathbb{E}[s^2]}{\mathbb{E}^2[s]}}, \tag{3}$$

where $\mathbb{E}[s]$ is the average object-size, and $\mathbb{E}[s^2]$ is the averaged square of the object-size.

Theorem 3.2 formalizes the error of the spatial sampling WSS estimation in heterogeneous object-size scenarios. For a cache with heterogeneous object sizes, the ratio $\mathbb{E}[s^2]/\mathbb{E}^2[s]$ exceeds 1. We conduct an experiment to illustrate the error induced by the heterogeneity of object sizes in real-world workloads. As depicted in Figure 7, the WSS estimation error

Algorithm 1: Weighted Sampling

Input: sampling rate r, ith object size S_i, average object-size S_{avg}
Output: Estimated Sum Size of Distinct Objects WSS_{ws}

```
WSS_ws = 0;
foreach object i do
    r_i = min(1, r × s_i/s_avg);
    if hash(i) ≤ r_i then
        WSS_ws += s_i/r_i
    end
end
```

for two CDN traces, along with their corresponding homogeneous traces (created by standardizing the object size), is amplified by a factor of 10 and 6.87 respectively.

Weighted Sampling. In sum estimation problems, linearly weighted sampling has proven to be a highly effective way to reduce estimation error [8, 22, 42]. The key point of Weighted Sampling is applying a linearly increasing sampling probability to objects based on their assigned weights. Furthermore, the influence of each sampled object is recalibrated according to its individual sampling rate.

Algorithm 1 shows how to apply Weighted Sampling to WSS estimation. For example, consider a Weighted Sampling process with a sampling rate of r. Each object i of size s_i, is assigned a sampling rate $r_i = \min(1, r \times s_i/S_{avg})$, where S_{avg} is the average size of the distinct objects. If object i is sampled with a probability of r_i, its contribution to the reuse distance or WSS estimation is $s_i' = s_i/r_i$. Our Theorem 3.3 demonstrates that Weighted Sampling can effectively reduce the estimation variance in heterogeneous object-size scenarios to the same level as in homogeneous scenarios.

Theorem 3.3. *(proof in Appendix C) For a trace with M distinct objects, s_i is the size of object i, the sampling rate is r, and WSS_{ws} is the estimated WSS calculated by Weighted Sampling. The coefficient of variation of WSS_{ws} is given by:*

$$C_v[WSS_{ws}] \le \sqrt{\frac{1-r}{rM}}. \tag{4}$$

We note that the parameter S_{avg} is essential for ensuring that the final sampling rate is close to p. In our evaluation, we estimate S_{avg} using a small trace prefix segment with minimal additional overhead and without sacrificing accuracy.

4 FLOWS

In this section, we present the detailed design of *FLOWS* (Section 4.1). We then demonstrate how Weighted Sampling can be employed for cache policies beyond LRU (Section 4.2). Additionally, we will explore how *FLOWS* can be employed for multi-objective optimization in cache management (Section 4.3).

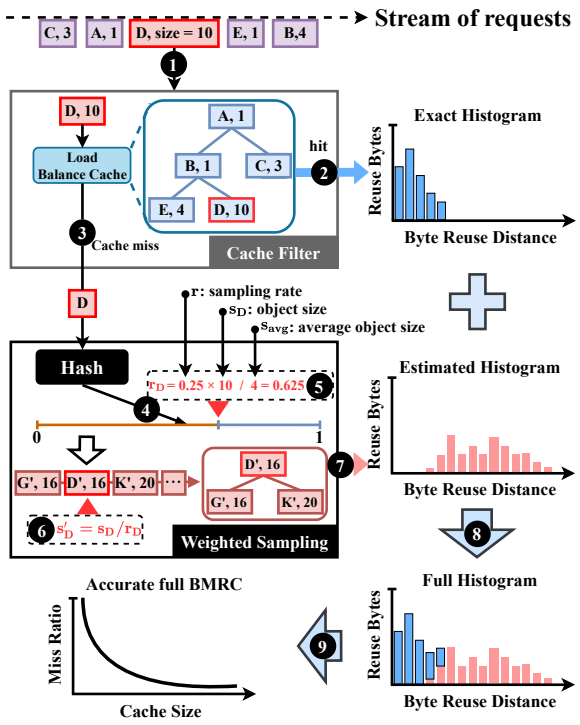


Figure 8. *FLOWS* overview.

4.1 Algorithm Design

We propose *FLOWS*, a three-phase algorithm that combines the Cache Filter and Weighted Sampling techniques to construct MRC, as shown in Figure 8. To explain the workflow of *FLOWS*, we use the following BMRC profiling example:

- The incoming stream of requests, BEDAC, are first sent to the Cache Filter(❶). If a request is found in the Cache Filter, its reuse distance is computed using a search tree (❷). If not, the request is added to the Cache Filter in an LRU manner and has a chance to be counted in the next Weighted Sampling phase (❸). The Cache Filter only keeps L objects in its cache and the search tree, and the evicted objects are passed to the Weighted Sampling phase.
- In the Weighted Sampling phase, a missed object (❸) is hashed to $U \in (0, 1)$ (❹), and its sampling rate is weighted by its size s_D. Formally, $r_D = r \times s_D / s_{avg}$, where r and s_{avg} are the given sampling rate and average object-size (❺), respectively. With a sampling rate of r_D, the counted size of the object D is $s'_D = s_D / r_D$ (❻). The sampled trace is then used to estimate the reuse distance histogram of objects missed in Cache Filter (❼).
- Finally, the exact histogram from the Cache Filter is combined with the estimated histogram in Weighted Sampling (❽) to yield the BMRC (❾).

Many studies have shown that content popularity is strongly correlated with object-size in web caches [7, 13, 20]. While our Weighted Sampling approach is able to accurately estimate the BMRC, it performs poorly in OMRC profiling due to this correlation. To address this issue, we propose a novel approach combining spatial sampling of a fixed sampling rate and Weighted Sampling to estimate the OMR and the reuse distance orthogonally. We then demonstrate how to adapt *FLOWS* to both a fixed sampling rate scheme and a fixed sampling number scheme.

Fixed Sampling Rate Scheme. Algorithm 2 provides a detailed description of *FLOWS*, which estimates both BMRC and OMRC simultaneously. There are two search trees, $\mathcal{T}_{CF}$, used in the Cache Filter, and $\mathcal{T}_{WS}$, used in Weighted Sampling. Histograms H_B and H_O are used to construct BMRC and OMRC, respectively. Furthermore, the last access time of sampled objects is recorded in a hash table, which is not shown in the pseudocode.

The Cache Filter is shared for both front portion BMRC and OMRC profiling as it always tracks the last L requests. When an object is hit in the Cache Filter, the BMRC histogram H_B and OMRC histogram H_O are updated (Lines 6 to 9). Objects that are not found in the Cache Filter can be sampled by spatial sampling (Line 14) or Weighted Sampling (Line 18). The reuse distance RD is estimated by:

$$RD = RD_{ws} + size(\mathcal{T}_{CF}), \tag{5}$$

where RD_{ws} is the sampled reuse distance in $\mathcal{T}_{WS}$, which is calculated by weighted sampling at sampling rate r, and $size(\mathcal{T}_{CF})$ is the total size of the cached L objects in the Cache Filter (Lines 12 to 13).

To reduce the high variance of reuse distance estimation caused by spatial sampling, we omit objects that are only sampled by spatial sampling from $\mathcal{T}_{WS}$. In the case of an object i that is sampled by spatial sampling (Lines 14 to 15), we increase H_O at its estimated reuse distance RD_i by $1/r$.

For weighted sampled object i, we set its sampling rate $r_i = \min(r \times s_i / s_{avg}, 1)$, update its position in search tree $\mathcal{T}_{WS}$ and update the BMRC histogram H_B (Lines 17 to 20). In order to ensure a consistent estimation of large objects (e.g., $s_i > s_{avg}/r$), we limit the maximum weighted sampling rate to no greater than 1. Consequently, the updates in H_B and $\mathcal{T}_{WS}$ are equal to s_i / r_i.

To ensure accurate sampling, the object evicted from the Cache Filter should be re-inserted into $\mathcal{T}_{WS}$ if it is weighted sampled (Lines 22 to 25). This will help to avoid counting the same object twice in $\mathcal{T}_{CF}$ and $\mathcal{T}_{WS}$.

However, there may still be a slight difference between the number of bytes or requests sampled, and the expected number of bytes or requests. To account for this difference, we normalize the histogram starting from the average size of the Cache Filter in H_B and H_O, until the sum of the buckets meets the expected value. After this, we construct the

Algorithm 2: Fixed Sampling Rate MRC Profiling

Input: Cache Filter size L, sampling rate r, average object-size S_{avg}, request trace
Output: Estimated BMRC and OMRC

1 $\mathcal{T}_{CF} \leftarrow intervalTree()$;
2 $\mathcal{T}_{WS} \leftarrow intervalTree()$;
3 $H_B \leftarrow byteReuseHist()$;
4 $H_O \leftarrow reqReuseHist()$;
5 **foreach** *requested object* i **do**
6 **if** $i \in \mathcal{T}_{CF}$ **then**
7 $RD_i \leftarrow update(\mathcal{T}_{CF}, i, s_i)$;
8 $H_B[RD_i] \leftarrow H_B[RD_i] + s_i$;
9 $H_O[RD_i] \leftarrow H_O[RD_i] + 1$;
10 **else**
11 // missed object in Cache Filter;
12 $RD_{ws,i} \leftarrow rangeSum(\mathcal{T}_{WS}, i)$;
13 $RD_i \leftarrow RD_{ws,i} + size(\mathcal{T}_{CF})$;
14 **if** $hash(i) \leq r$ **then**
15 $H_O[RD_i] \leftarrow H_O[RD_i] + 1/r$;
16 **end**
17 $r_i = \min(1, r \times s_i/s_{avg})$;
18 **if** $hash(i) \leq r_i$ **then**
19 $delete(\mathcal{T}_{WS}, i, s_i/r_i)$;
20 $H_B[RD_i] \leftarrow H_B[RD_i] + s_i/r_i$
21 **end**
22 $insert(\mathcal{T}_{CF}, i, s_i)$;
23 **if** $length(\mathcal{T}_{CF}) > L$ **then**
24 $e \leftarrow pop(\mathcal{T}_{CF})$;
25 $probInsert(\mathcal{T}_{WS}, e, s_e/r_e, r_e)$;
26 **end**
27 **end**
28 $BMRC \leftarrow constructNormalizedBMRC(H_B)$;
29 $OMRC \leftarrow constructNormalizedOMRC(H_O)$;
30 **end**

BMRC and OMRC from the normalized histograms H_B and H_O (Lines 28 to 29).

Fixed Sampling Number Scheme. Fixed sampling number schemes can provide a more predictable and efficient resource usage compared to fixed sampling rate schemes. Weighted sampling can be easily adapted to a fixed sampling number scheme by tracking only U items with the smallest sized-hash $hash_s(i)$, which is defined as:

$$hash_s(i) = hash(i)/s_i. \tag{6}$$

In the fixed sampling number implementation of SHARDS, the sampling rate r is represented by the largest hash value of the sampled objects. In contrast, in our Weighted Sampling scheme, the sampling rate r is the largest sized-hash $\max(hash_s(\cdot))$ of U sampled objects. To implement fixed sampling number spatial sampling in the OMRC profiling, we use the paradigm introduced by SHARDS.

Parameter Configuration. *FLOWS* requires three parameters: the length of the Cache Filter L, the sampling rate r, and the average object-size S_{avg}. Typically, L is set to $O(1/r \times \log(1/r))$, while the average object-size S_{avg} can be approximated through a small part of the trace file (e.g., 10K requests). It is important to note that *FLOWS* will increase the actual sampling rate, as Weighted Sampling and spatial sampling work independently. To ensure a fair comparison with other methods, the sampling rate or the number of samples in *FLOWS* may be halved.

4.2 Applying to Non-LRU Policies

For most cache algorithms, simulation is the only way to build an MRC, but it can be slow and inefficient. Fortunately, *Miniature Simulation* [59] techniques based on spatial sampling provide an alternative solution, which reduces both the time and space cost of simulation, while still providing good accuracy for some Non-LRU policies.

The Weighted Sampling scheme is easy to adapt to the BMRC and OMRC profiling for Non-LRU cache policies. Our methodology for non-LRU caches encompasses four key steps: sampling, scaling, replaying, and statistics:

- Sampling: Within a given trace, we calculate the sampling rate for an object i using the formula $r_i = \max(0.5\times r, r \times s_i/s_{avg})$ in order to achieve the targeted overall sampling rate r. We sample all objects whose hash values are less than r_i for further analysis.
- Scaling: For each sampled object i, we adjust its size to $s_i' = s_i/r_i$ to reflect the sampling rate.
- Replay: All sampled requests are fed directly into a cache simulator of a specified cache capacity, for example, libCacheSim.
- Statistics: During the replay phase, if an object i is hit, we increment the hit ratio accordingly. Scaling adjustments are necessary; for instance, the increase in byte-level hits is calculated as $s_i' = s_i/r_i$, and the increase in object-level hits is quantified as $1/r_i$.

Since the Cache Filter is not involved in this process, we calculate the difference in traffic size and request number between the sampled trace and the original trace. We attribute these differences to high-popularity objects and treat these requests as hits. Finally, by simulating under different cache sizes, we obtain the BMRC and OMRC of Non-LRU Policies.

4.3 Multi-Objective Optimization

By leveraging *FLOWS*, we can perform multi-objective optimization by allocating caches in a way that maximizes both BHR and OHR simultaneously. This approach enables us to strike a balance between these two objectives and achieve an optimal cache allocation strategy that meets the specific requirements of the system or application.

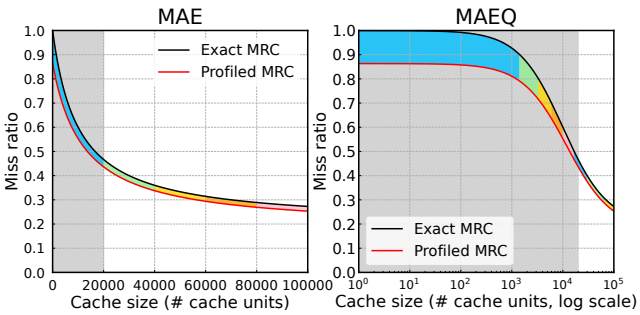


Figure 9. MAE (left) calculates the average absolute error of the miss ratio for uniformly distributed test points across different cache sizes (represented by different color sections). On the other hand, MAEQ (right) selects test ranges with evenly spaced miss ratios, such as 0.1 as shown in the figure. This approach causes MAEQ to place more emphasis on profiling errors for small cache sizes.

In this work, we consider Equation 7 as our optimization objective. The overall BHR and OHR for all N cache instances are denoted as BHR_g and OHR_g, respectively, and C is the total cache size. For a specific cache instance i with cache size c_i, its BHR and OHR are represented as $BHR_i(c_i)$ and $OHR_i(c_i)$. The access traffic and request for cache instance i are denoted as T_i and R_i, respectively. Hence, the overall byte hit rate (BHR) and object hit rate (OHR) can be expressed as follows: $BHR_g = \frac{\sum_i^N BHR_i(c_i) \times R_i}{SumTraffic}$ and $OHR_g = \frac{\sum_i^N OHR_i(c_i) \times R_i}{SumRequest}$.

$$\max_{c_i} \quad E = \omega \times BHR_g + (1 - \omega) \times OHR_g \tag{7}$$

$$\text{s.t.} \quad \sum_i^N c_i = C \tag{8}$$

The formulated equation represents a weighted combination of the global BHR and OHR, where ω is a configurable parameter. We set $\omega = 0.5$ in the following evaluations.

5 Evaluation

In this section, we evaluate the efficacy of *FLOWS* by first assessing the accuracy of MRC profiling under various sample settings, specifically examining the sensitivity of the FLOWS method parameters (Section 5.2). Subsequently, we analyze the performance improvement achieved through *FLOWS* in comparison to other cache management methods (Section 5.3). Additionally, we evaluate the time and space complexity of the FLOWS method, along with its efficiency (Section 5.4). Furthermore, we extend Weighted Sampling to profile some non-LRU cache policies (Section 5.5).

Table 2. Evaluated real-world traces.

Trace name	#Objs	#Reqs	Average object size	Traffic
Wiki2019t [51]	18.3M	207M	33KB	6454GB
Wiki2019u [51]	11.1M	100M	41KB	3884GB
QQPhoto [74]	33.3M	100M	25KB	2364GB
Twitter17 [66]	0.7M	883M	597B	491GB
Twitter18 [66]	1.1M	1292M	69B	83GB
Twitter24 [66]	1.2M	328M	598B	182GB
Twitter29 [66]	25.2M	699M	300B	195GB
Twitter44 [66]	5.8M	549M	42B	21GB
Twitter45 [66]	6.5M	22M	43B	1GB
Twitter52 [66]	25.2M	1343M	151B	188GB
MetaKV1 [1]	18.2M	207M	576B	110GB
MetaKV2 [1]	17.1M	204M	566B	108GB
MetaKV3 [1]	17.3M	198M	566B	105GB
MetaKV4 [1]	17.3M	201M	566B	106GB
MetaKV5 [1]	17.8M	204M	579B	110GB
MetaCDN-reag [1]	14.7M	50M	7935KB	370TB
MetaCDN-rnha [1]	36.9M	103M	8368KB	802TB
MetaCDN-rprn [1]	34.6M	96M	5314KB	475TB

5.1 Evaluation Methodology

The profiling accuracy is evaluated using Mean Absolute Error per Quantile (MAEQ) [18], which quantifies the absolute error between the exact and profiled MRC. MAEQ divides the MRC into equidistant quantiles based on the miss ratio and calculates the average Mean Absolute Error (MAE) for each quantile. The overall MAEQ is computed as $MAEQ = \sum_{i=1}^{Q} MAE_i/Q$, where MAE_i represents the MAE of the ith quantile and Q is the total number of quantiles. For our evaluations, we use an interval of 0.01. Figure 9 visualizes how MAE and MAEQ focus on errors in different regions of the MRC. We generate an MRC with 1000 data points, gradually increasing the cache size exponentially to facilitate MAEQ calculations.

We compare *FLOWS* with three state-of-the-art spatial sampling-based methods: SHARDS [60] and HCPP [18], as well as one learning-based algorithm, LPCA [27]. To ensure a fair comparison among the sampling methods, we set the sampling rate for *FLOWS* to be half of its baselines in all tests. This is because we sample both BMRC and OMRC separately and halving the sampling rate makes the total sampling amount of FLOWS equal to that of the baselines.

When comparing with the HCPP method and using a fixed sampling rate r, we set its small cache size, denoted as B to a fixed value. Specifically, we calculate $B = s_{avg}/r \times \log_{10} 1/r$, where s_{avg} is the average object size estimated using a prefix of 10K requests from the traces. Additionally, we set the length of the Cache Filter, denoted as L, to be $1/r \times \log_{10} 1/r$.

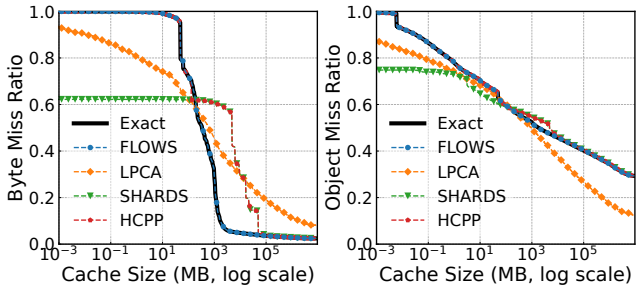


Figure 10. MRC profiling results under MetaCDN-reag traces with different methods with sampling rate $r = 0.01$.

The LPCA [27] method differs from the sampling-based approaches as it employs a neural network for predictive modeling. Our implementation of LPCA incorporates several features, including the re-access ratio, frequency histogram, frequency-class histogram, and reuse time histogram at byte level and object level. It is important to note that the intervals between the data points in our reuse time histograms also follow an exponential growth pattern.

To demonstrate the effectiveness of our approach, we assess the performance using 18 real-world traces as summarized in Table 2. In the original traces, we observed numerous cache misses where the object size was recorded as −1. To address this issue, we assign the size of each object as the largest observed size among all its requests. Additionally, we set the minimum object size to 1B.

In our cache allocation experiments, we utilize various cache traces as distinct cache instances that share a limited cache space. We combine different trace data with a uniform request rate. To simulate the cache environment, we employ libCachesim [3], a library that offers convenient I/O reading and cache instance functions. We have extended libCachesim to incorporate additional features such as cache size adjustment, MRC generation, and cache instance balancing.

The cache instance balancer utilizes the profiled BMRC and OMRC from the last time window to generate adjustment plans, gradually adjusting to the cache space in the subsequent time window. In our experiments, we assess the impact of various cache allocation strategies on system performance across different cache sizes. The primary metrics of interest include the overall BHR and OHR.

5.2 Real-World Trace Evaluation

To demonstrate the effectiveness of *FLOWS* in real-world scenarios, we evaluate its performance using a collection of open-source object cache traces (Table 2). Due to the large size of the original trace data, we have selected the sampled Twitter traces and the truncated Wiki and QQPhoto trace. These traces are sourced from various cache types, including KV cache, CDN cache, and multimedia cache, providing diverse traffic volumes and object sizes.

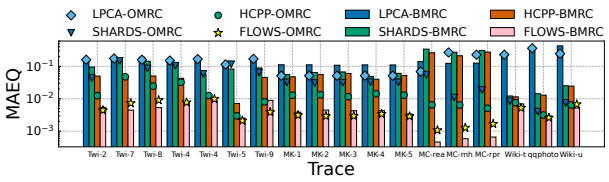


Figure 11. Profiling MAEQ of different profiling methods. Only the *FLOWS* method achieves an MAEQ lower than 0.01 in BMRC and OMRC profiling.

Fixed Sampling Rate. We first evaluate the accuracy of different MRC profiling methods using a sampling rate of $r = 0.01$. Figure 10 presents the results of BMRC and OMRC for the MetaCDN-reag [1] trace. As the cache size expands to around 1GB, the BMR experiences a significant decrease, while the object hit rate decreases linearly with exponential growth in cache size.

The LPCA method fails to match the exact curves due to its inability to learn MRC for objects of varying sizes. Although the SHARDS method can match the latter part of the Exact curve, it falls far short for small cache sizes. Conversely, the HCPP method accurately constructs the front portion of the curve but fails to capture the drop in byte hit rate around 1GB. Only *FLOWS* accurately models the entire curve, matching the exact curve closely and providing precise MRC profiling.

Figure 12 displays the MRC modeling results for all 18 traces at a sampling rate of $r = 0.01$. The *FLOWS* method accurately models the MRC for all traces across different cache sizes. To further compare the modeling performance, Figure 11 presents the modeling errors of different methods in terms of BMRC and OMRC with $r = 0.01$. HCPP exhibits an average modeling error of 7.29% for BMRC, while *FLOWS* achieves an average BMRC modeling error of 0.44%. *FLOWS* reduces the error by a factor of 16 compared to HCPP. Similarly, for OMRC, HCPP has an average modeling error of 1.33%, while *FLOWS* achieves an average OMRC modeling error of 0.43%. Once again, *FLOWS* reduces the error by a factor of 3 compared to HCPP. In particular, for MetaCDN traces, *FLOWS* achieves a 416-fold reduction in the BMRC modeling error compared to HCPP (from 24.9% to 0.06%).

As the sampling rate varies, the modeling error distribution changes. Figure 13 depicts the variations in modeling error distribution for different methods across all traces at sampling rates of 1%, 0.1%, and 0.01%. Across all settings, *FLOWS* consistently outperforms HCPP, reducing the average BMRC profiling error by at least 6×. Similarly, the average error for OMRC also decreases by at least 2×.

Fixed Sampling Number. Fixed Sampling Number scheme only samples a limited number of objects, resulting in constant memory overhead. The distribution of profiling errors for this scheme is illustrated in Figure 14. For *FLOWS*, we allocate up to half of the sampling count to Cache Filter,

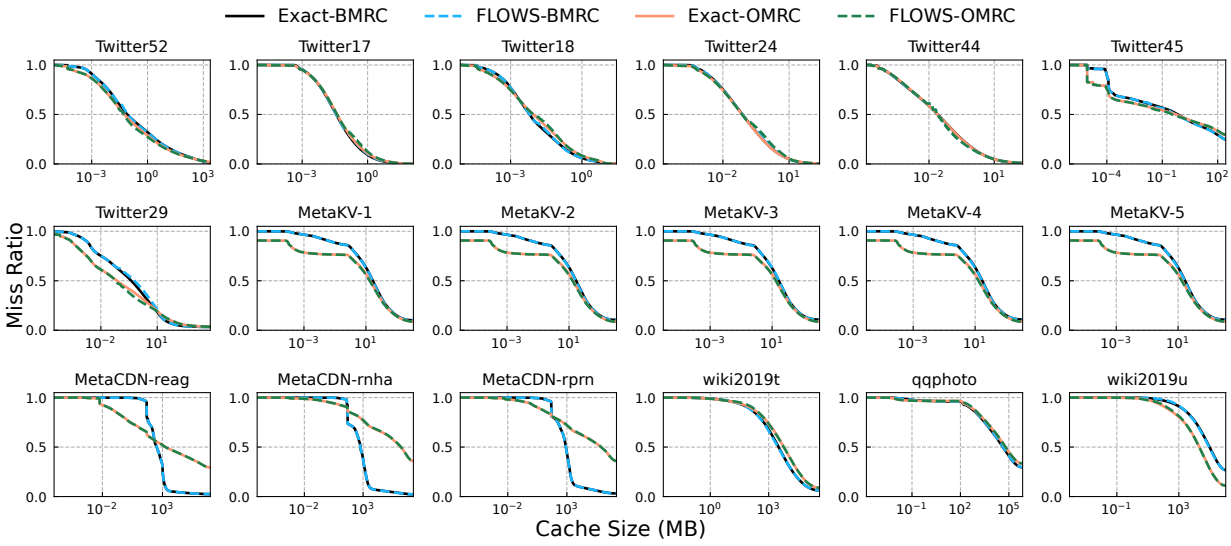


Figure 12. The MRC of 18 real-world traces. The curves generated by *FLOWS* closely resemble the exact MRC across all cache sizes. This indicates that FLOWS is able to accurately capture the miss ratio behavior in various caching scenarios.

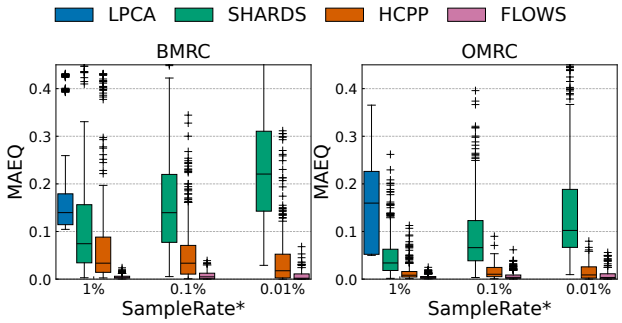


Figure 13. The distribution of MAEQ obtained from profiling using the *FLOWS* method and other methods with different configurations on all traces. (*) For LPCA [27], no sampling is performed.

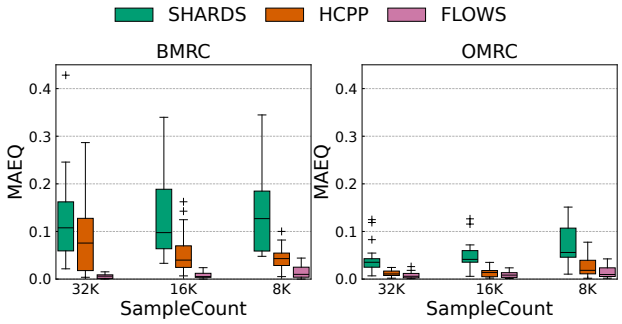


Figure 14. The error distribution obtained from *FLOWS* and other methods with different numbers of sampled objects on all traces.

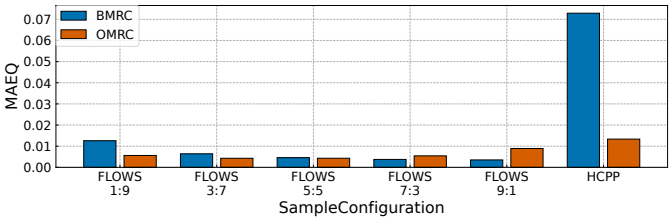


Figure 15. Average MAEQ of all traces with a sampling rate of $r = 0.01$. *FLOWS* $r_{ws} : r_{ss}$ means that the ratio of Weighted and spatial sampling is $r_{ws} : r_{ss}$.

while the remaining count is evenly divided between Spatial sampling and Weighted sampling.

As the sampling count increases, the modeling error of *FLOWS* method gradually increases. However, *FLOWS* consistently outperforms the HCPP method in BMRC profiling, reducing the average MAEQ by 6× (from 0.060 to 0.010). Additionally, *FLOWS* achieves a comparable lower level of profiling error (0.011) in OMRC profiling.

Impact of the Ratio of Spatial Sampling and Weighted Sampling. Tuning the ratio of the Weighted Sampling rate r_{ws} and the spatial sampling rate r_{ss} can have a significant impact on the accuracy of both BMRC and OMRC. In the *FLOWS* evaluation up to this point, the two sampling rates are equal, i.e., $r_{ws} : r_{ss} = 5 : 5$, which is the best ratio overall as indicated by Figure 15. Increasing the weighted sampled objects can decrease the error of BMRC, but also result in a higher OMRC error. When $r_{ws} : r_{ss} = 3 : 7$, *FLOWS* can reduce the average MAEQ of OMRC by about 3× compared to the HCPP method.

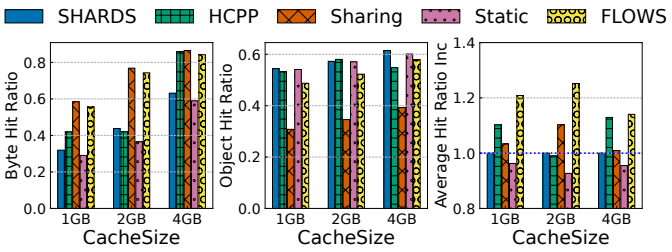


Figure 16. The overall BHR and OHR of multi-cache instances across different cache sizes. *FLOWS* consistently balances BHR and OHR to achieve a better overall hit ratio. The right figure presents the ratio of the *BHR+OHR* for different methods relative to the SHARDS method.

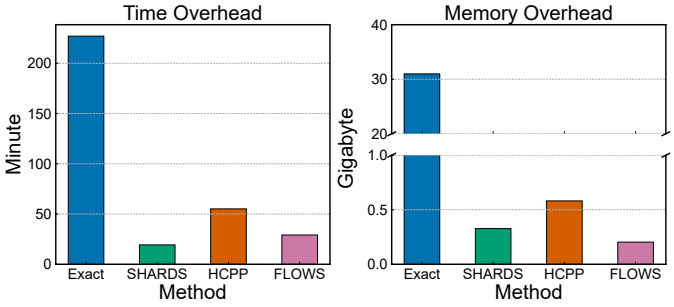


Figure 17. Time and memory overhead of profiling all 18 traces with sampling rate $r = 0.01$.

5.3 Cache Allocation Evaluation

We choose Twitter17 [66], MetaKV1 [1], MetaCDN-rprn [1], and wiki2019t [51] as distinct cache instances that share the overall cache space. We utilize Equation 7 as the optimization objective, which aims to optimize the overall hit ratio. We evaluate the overall performance of *FLOWS* and other methods at different cache sizes. These cache allocation schemes can be classified into three categories:

1. Static allocation method: This method evenly allocates the cache space among the cache instances.
2. Sharing method: In this approach, all cache instances share the same cache, and cache space is managed by the cache replacement algorithm.
3. Dynamic allocation method: This category includes the SHARDS, HCPP, and *FLOWS* methods. After every 1 million requests, based on the BMRC and OMRC constructed from the previous time window, we designate the cache instance with the highest increase in overall hit rate when increased and the least decrease in overall hit rate when decreased as the victim and receiver, respectively. We then gradually transfer 1% to a maximum of 5% of the cache space from the victim to the receiver.

Figure 16 demonstrates that *FLOWS* outperforms other methods significantly in terms of average hit rate. Particularly, when the cache space is relatively small, such as 2GB, FLOWS exhibits a 26.5% increase in *BHR + OHR* metrics compared to HCPP.

5.4 Space and Time Complexity Analysis

The analytical complexity of *FLOWS* in both space and time is comparable to SHARDS and HCPP. We utilize the open-source splay tree code developed by Sleator [50] as a reference to implement *FLOWS*, which can perform range summing, inserting, or deleting operations at an $O(\log M)$ complexity, where M is the number of objects in the tree. We constructed all 18 traces using different methods in a single-threaded manner on a cloud server equipped with an Intel(R) Xeon(R) Gold 6133 CPU. The sum of the time and memory overhead required to build all traces was used as the evaluation metric. The results are shown in Figure 17.

With a fixed sampling rate $r = 0.01$, the exact method required 3.8 hours to complete all profiling, and the total memory overhead was 31.0 GB. Compared to the exact method, the *FLOWS* method significantly reduced the time and memory overhead required for MRC profiling, reducing the time overhead by a factor of 7.8 (to 29 minutes) and the memory overhead by a factor of 152.5 (to 200MB). It is important to note that the open-source traces we are using are not fully representative of production systems. For instance, the Twitter traces are sampled at 1/10, and the MetaKV traces are sampled at 1/100 [1]. This implies that in a real-world environment, the overhead compared to our evaluation may increase by one to two orders of magnitude, requiring tens to hundreds of gigabytes of memory and tens of hours for exact MRC construction.

5.5 MRC Profiling under Non-LRU Policies

Weighted Sampling is an enhanced sampling method that can readily accommodate various non-LRU cache policies, including eviction policies such as 2Q [33], ARC [40], CACHEUS [48], and S3-FIFO [68], as well as cache admission policies like Probabilistic Admission [52] and AdaptSize [13]. To assess the performance of Weighted Sampling, we employed a constant sampling rate of $r = 0.01$, with the lower bound of the Weighted Sampling rate set to $r_{i_{min}} = 0.5 \times r$. This adjustment to the Weighted Sampling rate yields a more consistent and precise OMRC profiling, with only a marginal increase in the number of sampled requests compared to spatial sampling.

Eviction Policies. Figure 18 illustrates the BMRC and OMRC obtained through exact simulation and Weighted Sampling for the 2Q [33], ARC [40], CACHEUS [48], and S3-FIFO [68] algorithms. Weighted Sampling demonstrates its ability to achieve accurate MRC construction across all these

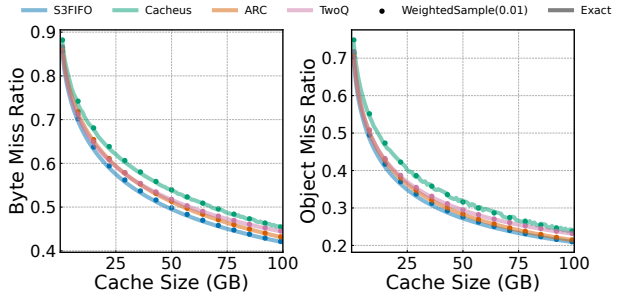


Figure 18. Exact and Weighted Sampling MRC profiling for wiki2019u [51].

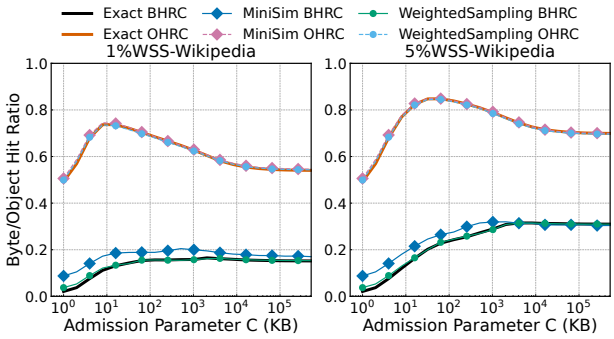


Figure 19. BHRC and OHRC of the Probabilistic Admission policy with different admission parameters c. The cache size on the left is 1% of the WSS, and on the right is 5%.

algorithms. Moreover, compared to exact simulation, the Weighted Sampling method offers a significant speedup while achieving more precise MRC construction compared to spatial sampling. For instance, in the case of 2Q profiling, the time required to construct a 100-point MRC decreased from 6150s (exact simulation) to 286s (Weighted Sampling). Additionally, the modeling MAE for BMRC and OMRC by using spatial sampling decreased from 3.28% and 0.56% to 0.24% and 0.23% with Weighted Sampling (which is not shown in the figure), respectively.

Probabilistic Admission. The Probabilistic Admission policy admits object i with probability p_i, where p_i is typically a function of object-size s_i, e.g., $e^{-s_i/c}$. Properly setting parameter c can maximize the OHR [13]. We form two sub-traces using spatial sampling and Weighted Sampling with a sampling rate of $r = 0.01$ and use the sampled sub-traces to estimate the cache performance. Figure 19 shows both exact and estimated BHRC (1-BMRC) and OHRC (1-OMRC) of different c and cache sizes. Our Weighted Sampling scheme can accurately approximate both BHRC and OHRC.

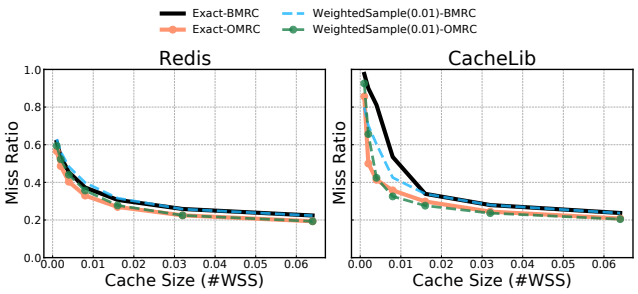


Figure 20. Weighted Sampling based Miniature Simulation MRC profiling on Redis [2] and CacheLib [9].

6 Discussion

6.1 How production CDN benefits from MRC

Through the MRC, cache managers can be effectively guided on how to optimize production CDNs. For instance, cache partitioning can prevent inefficient use of cache by limiting its capacity, which is supported in many cache systems, such as CacheLib [41]. Using the CDN trace from Meta as an example, when the cache capacity expands from 1GB to 7GB, the BMR plummets from approximately 32% to 5%. Further increasing the cache capacity to 32GB, the BMR only decreases by less than 1%. Some CDN systems opt not to support cache partitioning to simplify management, as reported in [54]. Such systems can optimize overall midgress traffic (reflected by the BMR) by distributing requests of the same traffic class [53] to different cache servers in a certain proportion, a process known as traffic provisioning. By simultaneously constructing accurate BMRC and OMRC, cache managers can accurately distinguish between traffic classes, thereby considering both midgress traffic and hit ratio when provisioning traffic.

6.2 Miniature Simulation of real cache systems with Weighted Sampling

Cache eviction algorithms do not always conform precisely to their theoretical constructs in practical caching systems. For example, Redis employs a probabilistic strategy to implement LRU by sampling a random subset of keys and evicting the one with the oldest access time within that subset [2]. Meanwhile, CacheLib [9] organizes objects into different slab classes based on size, where each class independently manages evictions via its own LRU list. Additionally, CacheLib integrates slab rebalancing to reduce allocation failures, ensure fairness, and enhance the OHR.

We attempt to quickly obtain the cache performance of these systems by employing Miniature Simulation [59] with Weighted Sampling. Figure 20 presents the cache performance of Redis and CacheLib's default configured LRU, processing 10 million MetaKV1 trace requests. The Miniature Simulation with Weighted Sampling, utilizing a sampling

rate of $r = 0.01$, yields accurate profiling for Redis. On the other hand, CacheLib's performance closely resembles that of an ideal LRU when there is sufficient cache capacity, ensuring the accuracy of the profiling.

A point of distinction in our methodology is that, unlike previous Miniature Simulations where modeling a cache of size C with a trace at sampling rate r necessitated adjusting the cache size to $r \times C$, our approach maintains the physical cache capacity constant. Instead, we adjust the object size by dividing it by the Weighted Sampling rate. This significantly enhances the accuracy of the Miniature Simulation in object caching systems.

7 Related Work

Evolving Caching Algorithms. To optimize the utilization of cache resources, researchers have proposed various cache policies. Most of these policies are designed for scenarios where object sizes are homogeneous [19, 23, 32–34, 40, 43, 44]. In order to obtain an advanced performance in the case of heterogeneous object-size cache, some algorithms utilize the object-size information to perform cache admission, such as LRU-S [52] and AdaptSize [13], or conduct cache eviction such as GSD [17], LHD [7], SegCache [67] and S3-FIFO [68]. By preventing or evicting large but cold objects from the cache, AdaptSize [13] and LHD [7] maximize the OHR under a specified cache size. Segcache [67] groups objects with similar creation and expiration times, reduces per-object metadata, and performs segment-level bulk expiration and eviction. S3-FIFO's [68] key insight lies in utilizing a small FIFO queue to swiftly evict objects that are expected to have only one-time accesses, thereby ensuring efficient demotion and high precision.

AI-assisted caching algorithms [11, 69] have outperformed traditional cache policies in recent advancements. These AI-enhanced cache algorithms leverage AI techniques to either enhance domain-specific cache performance in a data-driven manner [28, 37, 51, 61, 64, 65] or adapt to achieve higher performance across diverse workloads [48, 58].

FLOWS is not a cache algorithm but rather a method that efficiently acquires the cache performance model and guides the management of limited cache resources. By doing so, it contributes to improved overall performance.

MRC Profiling Methods. Due to the importance and limited availability of cache resources in computer systems, MRC has been extensively studied as a quantitative model for cache resources by researchers [4, 15, 18, 24, 27, 30, 59, 60, 62, 72]. Denning [21] was one of the first to investigate the cache miss ratio as a function of cache size. Methods for constructing MRCs can be categorized into several types, including those based on reuse distance [4, 18, 60], reuse time [24, 30, 72], sampling-replay [59], and machine learning [27]. The focus of these studies has been on reducing modeling errors, improving modeling speed, and extending to non-LRU methods. Recent studies Cuki [26] and Kosmo [49] have focused on the profiling of OMRC for heterogeneous object-size cache systems, rather than addressing the profiling of BMRC.

In this study, we aim to address the imbalanced accuracy of BMRC and OMRC profiling in cache systems with heterogeneous object-size and content popularity. Unlike previous methods, our approach incorporates the principle of "load balancing" to achieve more precise MRC profiling.

8 Conclusion

Spatial sampling-based MRC profiling often faces challenges due to the heterogeneity caused by variations in content popularity and object sizes. To address this issue, we propose *FLOWS*, a load-balancing solution based on two novel techniques: Cache Filter and Weighted Sampling. *FLOWS* enables the construction of equally accurate BMRC and OMRC while reducing the BMRC and OMRC profiling error by 16× and 3× compared to state-of-the-art approaches, respectively. Our cache allocation experiments demonstrate that *FLOWS* effectively balances BHR and OHR, resulting in a maximum improvement of 26.5% in overall hit rate compared to other methods. Furthermore, it maintains the same space and time complexity as existing methods. As a versatile performance evaluation tool, *FLOWS* can be easily integrated into web caching services, where both BMRC and OMRC are of particular importance. This approach will also contribute to achieving a more flexible and efficient multi-objective cache optimization in web cache systems.

Acknowledgments

We would like to express our profound gratitude to the reviewers for their insightful comments and suggestions. Our sincere thanks also go to our shepherd, Vasily Tarasov, for his invaluable guidance and support throughout the revision process. This work was supported by the National Key Research and Development Program Grant No.2023YFB4502701, and the National Natural Science Foundation of China (Grant No.62172180, No.62232007, No.61821003).

References

[1] 2023. Evaluating SSD hardware for Facebook workloads. https://cachelib.org/docs/Cache_Library_User_Guides/Cachebench_FB_HW_eval. Accessed: October 7, 2023.

[2] 2023. Key eviction. https://redis.io/docs/reference/eviction/. Accessed: February 20, 2024.

[3] 1a1a11a. 2023. libCacheSim. https://github.com/1a1a11a/libCacheSim.

[4] George Almási, Călin Caşcaval, and David A Padua. 2002. Calculating stack distances efficiently. In *Proceedings of the 2002 workshop on Memory system performance*. 37–43.

[5] Nirav Atre, Justine Sherry, Weina Wang, and Daniel S. Berger. 2020. Caching with Delayed Hits. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 495–513.

https://doi.org/10.1145/3387514.3405883

[6] Thomas Barnett, Shruti Jain, Usha Andra, and Taru Khurana. 2018. Cisco visual networking index (vni) complete forecast update, 2017–2022. *Americas/EMEAR Cisco Knowledge Network (CKN) Presentation* (2018).

[7] Nathan Beckmann, Haoxian Chen, and Asaf Cidon. 2018. LHD: Improving Cache Hit Rate by Maximizing Hit Density. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 389–403. https://www.usenix.org/conference/nsdi18/presentation/beckmann

[8] Lorenzo Beretta and Jakub Tětek. 2022. Better Sum Estimation via Weighted Sampling. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2303–2338.

[9] Benjamin Berg, Daniel S. Berger, Sara McAllister, Isaac Grosof, Sathya Gunasekar, Jimmy Lu, Michael Uhlar, Jim Carrig, Nathan Beckmann, Mor Harchol-Balter, and Gregory R. Ganger. 2020. The CacheLib Caching Engine: Design and Experiences at Scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 753–768. https://www.usenix.org/conference/osdi20/presentation/berg

[10] Erik Berg and Erik Hagersten. 2004. StatCache: A probabilistic approach to efficient and accurate data locality analysis. In *IEEE International Symposium on-ISPASS Performance Analysis of Systems and Software, 2004.* IEEE, 20–27.

[11] Daniel S Berger. 2018. Towards lightweight and robust machine learning for cdn caching. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. 134–140.

[12] Daniel S. Berger, Benjamin Berg, Timothy Zhu, Siddhartha Sen, and Mor Harchol-Balter. 2018. RobinHood: Tail Latency Aware Caching – Dynamic Reallocation from Cache-Rich to Cache-Poor. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 195–212. https://www.usenix.org/conference/osdi18/presentation/berger

[13] Daniel S. Berger, Ramesh K. Sitaraman, and Mor Harchol-Balter. 2017. AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 483–498. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/berger

[14] Jacob Brock, Chencheng Ye, Chen Ding, Yechen Li, Xiaolin Wang, and Yingwei Luo. 2015. Optimal cache partition-sharing. In *2015 44th International Conference on Parallel Processing*. IEEE, 749–758.

[15] Daniel Byrne. 2018. A Survey of Miss-Ratio Curve Construction Techniques. *ArXiv* abs/1804.01972 (2018).

[16] Daniel Byrne, Nilufer Onder, and Zhenlin Wang. 2018. mPart: miss-ratio curve guided partitioning in key-value stores. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on Memory Management*. 84–95.

[17] Pei Cao and Sandy Irani. 1997. Cost-Aware WWW Proxy Caching Algorithms. In *USENIX Symposium on Internet Technologies and Systems (USITS 97)*. USENIX Association, Monterey, CA. https://www.usenix.org/conference/usits-97/cost-aware-www-proxy-caching-algorithms

[18] Damiano Carra and Giovanni Neglia. 2020. Efficient Miss Ratio Curve Computation for Heterogeneous Content Popularity. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 741–751. https://www.usenix.org/conference/atc20/presentation/carra

[19] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. 2016. Cliffhanger: Scaling Performance Cliffs in Web Memory Caches. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 379–392. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/cidon

[20] Carlos Cunha, Azer Bestavros, and Mark Crovella. 1995. *Characteristics of WWW Client-Based Traces.* Technical Report. USA.

[21] Peter J Denning. 1968. The working set model for program behavior. *Commun. ACM* 11, 5 (1968), 323–333.

[22] Nick Duffield, Carsten Lund, and Mikkel Thorup. 2007. Priority sampling for estimation of arbitrary subset sums. *Journal of the ACM (JACM)* 54, 6 (2007), 32–es.

[23] Gil Einziger, Roy Friedman, and Ben Manes. 2017. Tinylfu: A highly efficient cache admission policy. *ACM Transactions on Storage (ToS)* 13, 4 (2017), 1–31.

[24] David Eklov and Erik Hagersten. 2010. StatStack: Efficient modeling of LRU caches. In *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*. IEEE, 55–65.

[25] Bin Fan, Hyeontaek Lim, David G Andersen, and Michael Kaminsky. 2011. Small cache, big effect: Provable load balancing for randomly partitioned cluster services. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 1–12.

[26] Rong Gu, Simian Li, Haipeng Dai, Hancheng Wang, Yili Luo, Bin Fan, Ran Ben Basat, Ke Wang, Zhenyu Song, Shouwei Chen, et al. 2023. Adaptive online cache capacity optimization via lightweight working set size estimation at scale. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 467–484.

[27] Yibin Gu, Yifan Li, Hua Wang, Li Liu, Ke Zhou, Wei Fang, Gang Hu, Jinhu Liu, and Zhuo Cheng. 2022. LPCA: Learned MRC Profiling Based Cache Allocation for File Storage Systems. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. Association for Computing Machinery, New York, NY, USA, 511–516. https://doi.org/10.1145/3489517.3530662

[28] Yu Guan, Xinggong Zhang, and Zongming Guo. 2019. Caca: Learning-based content-aware cache admission for video content in edge caching. In *Proceedings of the 27th ACM International Conference on Multimedia*. 456–464.

[29] Xiameng Hu, Xiaolin Wang, Yechen Li, Lan Zhou, Yingwei Luo, Chen Ding, Song Jiang, and Zhenlin Wang. 2015. LAMA: Optimized Locality-aware Memory Allocation for Key-value Cache. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. USENIX Association, Santa Clara, CA, 57–69. https://www.usenix.org/conference/atc15/technical-session/presentation/hu

[30] Xiameng Hu, Xiaolin Wang, Lan Zhou, Yingwei Luo, Chen Ding, and Zhenlin Wang. 2016. Kinetic Modeling of Data Eviction in Cache. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 351–364. https://www.usenix.org/conference/atc16/technical-sessions/presentation/hu

[31] Kaiyi Ji, Guocong Quan, and Jian Tan. 2018. Asymptotic miss ratio of LRU caching with consistent hashing. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 450–458.

[32] 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.

[33] Theodore Johnson, Dennis Shasha, et al. 1994. 2Q: a low overhead high performance bu er management replacement algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases*. Citeseer, 439–450.

[34] Ramakrishna Karedla, J Spencer Love, and Bradley G Wherry. 1994. Caching strategies to improve disk system performance. *Computer* 27, 3 (1994), 38–46.

[35] Seongbeom Kim, Dhruba Chandra, and Yan Solihin. 2004. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings. 13th International Conference on Parallel Architecture and Compilation Techniques, 2004. PACT 2004.* IEEE, 111–122.

[36] Yoongu Kim, Michael Papamichael, Onur Mutlu, and Mor Harchol-Balter. 2010. Thread cluster memory scheduling: Exploiting differences

in memory access behavior. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 65–76.

[37] Vadim Kirilin, Aditya Sundarrajan, Sergey Gorinsky, and Ramesh K Sitaraman. 2020. Rl-cache: Learning-based cache admission for content delivery. *IEEE Journal on Selected Areas in Communications* 38, 10 (2020), 2372–2385.

[38] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. 2019. DistCache: Provable Load Balancing for Large-Scale Storage Systems with Distributed Caching. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. USENIX Association, Boston, MA, 143–157. https://www.usenix.org/conference/fast19/presentation/liu

[39] Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. 1970. Evaluation techniques for storage hierarchies. *IBM Systems journal* 9, 2 (1970), 78–117.

[40] Nimrod Megiddo and Dharmendra S. Modha. 2003. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *2nd USENIX Conference on File and Storage Technologies (FAST 03)*. USENIX Association, San Francisco, CA. https://www.usenix.org/conference/fast-03/arc-self-tuning-low-overhead-replacement-cache

[41] Meta. 2023. Partition cache into pools. https://cachelib.org/docs/Cache_Library_User_Guides/Partition_cache_into_pools/#stats-api.

[42] Rajeev Motwani, Rina Panigrahy, and Ying Xu. 2007. Estimating sum by weighted sampling. In *International Colloquium on Automata, Languages, and Programming*. Springer, 53–64.

[43] 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.

[44] Seon-yeong Park, Dawoon Jung, Jeong-uk Kang, Jin-soo Kim, and Joonwon Lee. 2006. CFLRU: a replacement algorithm for flash memory. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*. 234–241.

[45] Qifan Pu, Haoyuan Li, Matei Zaharia, Ali Ghodsi, and Ion Stoica. 2016. FairRide: Near-Optimal, Fair Cache Sharing. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 393–406. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/pu

[46] Moinuddin K Qureshi and Yale N Patt. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, 423–432.

[47] Martin Raab and Angelika Steger. 1998. "Balls into bins"—A simple and tight analysis. In *International Workshop on Randomization and Approximation Techniques in Computer Science*. Springer, 159–170.

[48] Liana V. Rodriguez, Farzana Yusuf, Steven Lyons, Eysler Paz, Raju Rangaswami, Jason Liu, Ming Zhao, and Giri Narasimhan. 2021. Learning Cache Replacement with CACHEUS. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, 341–354. https://www.usenix.org/conference/fast21/presentation/rodriguez

[49] Kia Shakiba, Sari Sultan, and Michael Stumm. 2024. Kosmo: Efficient Online Miss Ratio Curve Generation for Eviction Policy Evaluation. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*. 89–105.

[50] D. Sleator. [n. d.]. An implementation of top-down splaying with sizes. ftp://ftp.cs.cmu.edu/usr/ftp/usr/sleator/splaying/top-down-splay.c.

[51] Zhenyu Song, Daniel S. Berger, Kai Li, and Wyatt Lloyd. 2020. Learning Relaxed Belady for Content Distribution Network Caching. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 529–544. https://www.usenix.org/conference/nsdi20/presentation/song

[52] David Starobinski and David Tse. 2001. Probabilistic methods for web caching. *Performance evaluation* 46, 2-3 (2001), 125–137.

[53] Aditya Sundarrajan, Mingdong Feng, Mangesh Kasbekar, and Ramesh K Sitaraman. 2017. Footprint descriptors: Theory and practice of cache provisioning in a global cdn. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 55–67.

[54] Aditya Sundarrajan, Mangesh Kasbekar, Ramesh K. Sitaraman, and Samta Shukla. 2020. Midgress-aware traffic provisioning for content delivery. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 543–557. https://www.usenix.org/conference/atc20/presentation/sundarrajan

[55] David K Tam, Reza Azimi, Livio B Soares, and Michael Stumm. 2009. RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations. *ACM Sigplan Notices* 44, 3 (2009), 121–132.

[56] Terracotta, Inc. Accessed: 2023. Ehcache. https://www.ehcache.org/.

[57] Michail-Antisthenis I Tsompanas, Christoforos Kachris, and Georgios Ch Sirakoulis. 2016. Modeling cache memory utilization on multicore using common pool resource game on cellular automata. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 26, 3 (2016), 1–22.

[58] 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 *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/hotstorage18/presentation/vietri

[59] Carl Waldspurger, Trausti Saemundsson, Irfan Ahmad, and Nohhyun Park. 2017. Cache Modeling and Optimization using Miniature Simulations. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 487–498. https://www.usenix.org/conference/atc17/technical-sessions/presentation/waldspurger

[60] Carl A. Waldspurger, Nohhyun Park, Alexander Garthwaite, and Irfan Ahmad. 2015. Efficient MRC Construction with SHARDS. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. USENIX Association, Santa Clara, CA, 95–110. https://www.usenix.org/conference/fast15/technical-sessions/presentation/waldspurger

[61] Hua Wang, Xinbo Yi, Ping Huang, Bin Cheng, and Ke Zhou. 2018. Efficient SSD caching by avoiding unnecessary writes using machine learning. In *Proceedings of the 47th International Conference on Parallel Processing*. 1–10.

[62] Jake Wires, Stephen Ingram, Zachary Drudi, Nicholas J. A. Harvey, and Andrew Warfield. 2014. Characterizing Storage Workloads with Counter Stacks. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 335–349. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/wires

[63] Xiaoya Xiang, Bin Bao, Chen Ding, and Yaoqing Gao. 2011. Linear-time modeling of program working set in shared cache. In *2011 International Conference on Parallel Architectures and Compilation Techniques*. IEEE, 350–360.

[64] Gang Yan and Jian Li. 2020. Rl-Bélády: A unified learning framework for content caching. In *Proceedings of the 28th ACM International Conference on Multimedia*. 1009–1017.

[65] Juncheng Yang, Ziming Mao, Yao Yue, and KV Rashmi. 2023. {GL-Cache}: Group-level learning for efficient and high-performance caching. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. 115–134.

[66] Juncheng Yang, Yao Yue, and K. V. Rashmi. 2020. A large scale analysis of hundreds of in-memory cache clusters at Twitter. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 191–208. https://www.usenix.org/conference/osdi20/presentation/yang

[67] Juncheng Yang, Yao Yue, and Rashmi Vinayak. 2021. Segcache: a memory-efficient and scalable in-memory key-value cache for small objects. In *18th USENIX Symposium on Networked Systems Design and*

Implementation (NSDI 21). 503–518.

[68] Juncheng Yang, Yazhuo Zhang, Ziyue Qiu, Yao Yue, and Rashmi Vinayak. 2023. FIFO queues are all you need for cache eviction. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 130–149.

[69] Jiahui Ye, Zichun Li, Zhi Wang, Zhuobin Zheng, Han Hu, and Wenwu Zhu. 2021. Joint cache size scaling and replacement adaptation for small content providers. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 1–10.

[70] Ailing Yu, Yujuan Tan, Congcong Xu, Zhulin Ma, Duo Liu, and Xianzhang Chen. 2021. DFShards: effective construction of MRCs online for non-stack algorithms. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*. 63–72.

[71] Seyed Majid Zahedi and Benjamin C Lee. 2014. REF: Resource elasticity fairness with sharing incentives for multiprocessors. *ACM SIGPLAN Notices* 49, 4 (2014), 145–160.

[72] 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 *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 785–798. https://www.usenix.org/conference/atc20/presentation/zhang-yu

[73] Yutao Zhong, Xipeng Shen, and Chen Ding. 2009. Program locality analysis using reuse distance. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 31, 6 (2009), 1–39.

[74] 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 *Proceedings of the 2018 International Conference on Supercomputing*. 284–294.

Appendix

A Proof of Theorem 3.1: the size of Cache Filter

In a spatial sampling process with a sample rate of r, each object i has a content popularity of h_i, ranging from $0 \le h_i \le 1$. If the number of objects M in the trace is significantly larger than $\frac{1}{r}\log\frac{1}{r}$, by caching the L most popular objects, the relative bias of the sampled content popularity in spatial sampling is consistently bounded with a probability of $(1 - O(r^2))$. The value of L is determined by the following equation:

$$L = O(\frac{1}{r}\log\frac{1}{r}). \tag{9}$$

Assume we have M objects, each object i has a content popularity h_i, where $\sum_{i=1}^{M} h_i = 1$. Without loss of generality, we assume that:

$$h_1 \ge h_2 \ge \cdots \ge h_M \ge 0. \tag{10}$$

We have a small cache of size L, which always caches L objects that have the highest content popularity. For uncached objects, we sample them with a sampling rate r. Our goal is to find a cache size L that can bound the relative bias in the total popularity of the sampled objects to a constant, without considering the number of objects M and the distribution of content popularity.

For independent boolean random variables $X_1, X_2, \ldots, X_M$ where $X_i \in \{0, 1\}$ is to represent whether object i is sampled. As the first L object is cached, the sampled content popularity H is given by the following equation:

$$H = \sum_{i=L+1}^{M} X_i h_i. \tag{11}$$

When $L = 0$, the sum popularity of sampled objects is:

$$H = \sum_{i=1}^{M} X_i h_i. \tag{12}$$

It is evident that $\mathbb{E}(H) = r\sum_{i=1}^{M} h_i = r$. We use the variance of H to analyze which popularity distribution will result in the sampling process having the maximum relative bias. The variance of H is:

$$Var[H] = Var[\sum_{i=1}^{M} X_i h_i] = \sum_{i=1}^{M} h_i^2 Var[X_i] = r(1-r)\sum_{i=1}^{M} h_i^2. \tag{13}$$

We use an example to illustrate how cache size L and popularity distribution interact with each other. Easy to see when $h_1 = 1$ and all other objects have zero popularity, the variance of H reaches the maximum $r(1-r)$. But if we have $L = 1$, the hottest object will be cached and the variance of H becomes 0.

When $L \neq 0$, the variance of H is:

$$Var[H] = Var[\sum_{i=L+1}^{M} X_i h_i] = \sum_{i=L+1}^{M} h_i^2 Var[X_i] = r(1-r)\sum_{i=L+1}^{M} h_i^2. \tag{14}$$

In Lemma .1 and Lemma .2, we will explore the object popularity distribution that maximizes the variance of H or the relative bias.

Lemma .1. *The content popularity of L cached objects should be equal to make the sampled content popularity H have the highest variance. Which means:*

$$h_1 = h_2 = \cdots = h_L \ge h_{L+1} \ge \cdots \ge h_M. \tag{15}$$

Proof. Assuming the popularity of cached requests varies, that is, there exists adjacent cached objects, $h_i > h_{i+1}$, where $1 \le i \le L-1$. At this point, we do the following process to make the variance of H greater:

1. Reduce the popularity of h_i to h_{i+1}, which will not change the order of objects;

2. Distribute the reduced popularity $h_i - h_{i+1}$ to uncached objects without making them exceed h_L, which will not change the order of objects.

After that, the variance of H is increased. If we continue to do this repeatedly by iterating i from $L-1$ to 1, eventually the popularity of the cached objects will become equal.

□

Lemma .2. *When the variance of H is maximized, there will be x objects with content popularity equal to $\frac{1}{x}$, and other objects have 0 content popularity.*

$$\frac{1}{x} = h_1 = h_2 = \cdots = h_L = h_{L+1} = \cdots = h_x > h_{x+1} = \cdots = h_M = 0 \tag{16}$$

Proof. For uncached keys i and j such that $h_L > h_i > h_j > 0$. We can change the popularity to $h_i + \delta$ and $h_j - \delta$ to increase the variation of the popularity sampled, where $\delta = \min(h_L - h_i, h_j)$.

If we continue to execute this process and according to Lemma .1, eventually only x objects will have the same popularity $\frac{1}{x}$. □

Lemma .3. *Suppose there are m objects, and we sample them with a sample rate of r, where each object is independently and randomly sampled. The number of sampled objects is a random variable X. If $m \gg \frac{1}{r} \cdot \log\frac{1}{r}$, we have the following equation:*

$$\mathbb{P}\left(|X - r \cdot m| \geq \alpha\sqrt{r \cdot m \cdot log\frac{1}{r}}\right) \leq O(r^2), \tag{17}$$

which implies the number of sampled objects have a certainty of $1-O(r^2)$ to have a difference less equal than $\alpha\sqrt{r \cdot m \cdot log\frac{1}{r}}$.

Proof. For independent random variable $X_1, X_2, \ldots, X_m$, where $X_i \in \{0, 1\}$ to represent whether the object i is sampled. Let $X = \sum_{i=1}^{m} X_i$ as the total sampled objects, and $\mu = \mathbb{E}(X) = r \cdot m$:

Based on Chernoff Bounds, for $0 < \delta < 1$,

$$\mathbb{P}(X \geq (1+\delta)\mu) \leq \exp(-\frac{\mu\delta^2}{3}) \tag{18}$$

$$\mathbb{P}(X \leq (1-\delta)\mu) \leq \exp(-\frac{\mu\delta^2}{2}) \tag{19}$$

Let $\delta^* = \sqrt{\frac{\frac{1}{r}\log\frac{1}{r}}{m}}$, as $m \gg \frac{1}{r} \cdot \log\frac{1}{r}$, we have:

$$0 < \sqrt{6}\delta^* < 1. \tag{20}$$

Let's substitute this value into the Chernoff Bounds:

$$\mathbb{P}(X \geq (1+\sqrt{6}\delta^*)\mu) \leq \exp(-\frac{\mu(\sqrt{6}\delta^*)^2}{3}) \tag{21}$$

$$\mathbb{P}(X \geq (1+\sqrt{6}\delta^*)\mu) \leq \exp(-2\mu\delta^{*2}) \tag{22}$$

$$\mathbb{P}(X \geq (1+\sqrt{6}\delta^*)\mu) \leq \exp(-2r \cdot m\frac{\log\frac{1}{r}}{r \cdot m}) \tag{23}$$

$$\mathbb{P}(X \geq (1+\sqrt{6}\delta^*)\mu) \leq \exp(-2\log\frac{1}{r}) \tag{24}$$

$$\mathbb{P}(X \geq (1+\sqrt{6}\delta^*)\mu) \leq r^2. \tag{25}$$

Similarly, we have:

$$\mathbb{P}(X \geq (1-2\delta)\mu) \leq r^2. \tag{26}$$

So for a consistent α, we have:

$$\mathbb{P}\left(|X - r \cdot m| \leq \alpha\sqrt{r \cdot m \cdot \log\frac{1}{r}}\right) \leq O(r^2),$$

□

According to Lemma .2, when the variance of the sampled popularity is maximized, it is similar to the sample x homogeneous popularity objects with sampling rate r. Next, we will derive the values of x that maximize the relative bias and use them to determine the value of cache size L, such that even in the case of maximum relative error, the error remains bounded.

Based on Lemma .3, let us assume that there are a total of $x-L$ objects that can be sampled, where $m = x-L \gg \frac{1}{r}\log\frac{1}{r}$. The number of sampled objects X satisfies the following condition:

$$\mathbb{P}\left(|X - r \cdot (x-L)| \leq \alpha\sqrt{r \cdot (x-L)\log\frac{1}{r}}\right) \geq 1 - O(r^2), \tag{27}$$

Each object has a popularity of $\frac{1}{x}$, so $H = \frac{X}{x}$ and $\frac{r(x-L)}{x}$ is the expect value of H. Let's divide by the sampling rate r to obtain the relative bias:

$$\left|\frac{H}{r} - \frac{x-L}{x}\right| \leq (\alpha\sqrt{\frac{x-L}{x^2}\frac{1}{r}\log\frac{1}{r}}). \tag{28}$$

We want to find the value of x that maximizes the bias for $|\frac{H}{r} - \frac{x-L}{x}|$, which is $x = 2L$. Let us substitute this value into the equation:

$$\left|\frac{H}{r} - \frac{x-L}{x}\right| \leq \alpha\sqrt{\frac{1}{4L}\frac{1}{r}\log\frac{1}{r}}. \tag{29}$$

In the equation, α represents a constant term. When $L = O(\frac{1}{r}\log\frac{1}{r})$, we find that $|\frac{H}{r} - \frac{x-L}{x}|$ is independent of r. In this case, it is also independent of the number of requests M and the distribution of content popularity, as the following equation:

$$\left|\frac{H}{r} - \frac{x-L}{x}\right| \leq \frac{\alpha}{2} = Constant. \tag{30}$$

Therefore, it allows us to maintain a constant relative bias in sampling the content popularity. □

B Proof of Theorem 3.2

For a trace with M distinct objects, we use spatial sampling with sampling rate r to sample a subset of objects. We use the sum sampled size divided by r, denoted as WSS_s, to estimate the original working set size WSS.

The estimation error of WSS_s can be calculated according to the following coefficient of variation equation:

$$C_v[WSS_s] = \sqrt{\frac{1-r}{rM} \times \frac{\mathbb{E}[s^2]}{\mathbb{E}^2[s]}}, \tag{31}$$

where $\mathbb{E}[s]$ represents the average object size, and $\mathbb{E}[s^2]$ represents the average square of the object size. This implies the error of spatial sampling is correlated with the distribution of object size.

Proof. Let $\mathtt{X}_i$ be a Bernoulli random variable that indicates if object i is sampled with probability r, such that:

$$f_{X_i}(x) = \begin{cases} r & \text{x = 1} \\ 1-r & \text{x = 0.} \end{cases} \tag{32}$$

We calculate the estimated unique bytes WSS_s as follows:

$$WSS_s = \sum_i^M X_i s_i / r. \tag{33}$$

X_i and s_i are independent, so the value of $\mathbb{E}[WSS_s]$ and variation $Var[WSS_s]$ is:

$$\mathbb{E}[WSS_s] = \mathbb{E}[\sum_i^M X_i s_i / r] \tag{34}$$

$$= \mathbb{E}[\sum_i^M X_i / r] \times \mathbb{E}[\sum_i^M s_i] \tag{35}$$

$$= M \times \mathbb{E}[s], \tag{36}$$

$$Var[WSS_s] = \sum_i^M Var[X_i s_i / r] \tag{37}$$

$$= \sum_i^M s_i^2 / r^2 \times r(1-r) \tag{38}$$

$$= (1-r)/r \times M \times \mathbb{E}[s^2]. \tag{39}$$

The coefficient of variation $C_v[WSS_s]$ introduced by spatial sampling is:

$$C_v[WSS_s] = \frac{\sqrt{Var[WSS_s]}}{\mathbb{E}[WSS_s]} = \sqrt{\frac{1-r}{rM} \times \frac{\mathbb{E}[s^2]}{\mathbb{E}^2[s]}}. \tag{40}$$

□

C Proof of Theorem 3.3

For a trace with M distinct objects, we use Weighted Sampling with sampling rate r to sample a subset of objects. We use WSS_{ws} to estimate the original working set size WSS. The estimation error of WSS_{ws} can be calculated according to the following coefficient of variation equation:

$$C_v[WSS_{ws}] \le \sqrt{\frac{1-r}{rM}}. \tag{41}$$

This implies the error of Weighted Sampling is independent of object size distribution.

Proof. Let $\mathtt{X}_i$ be a Bernoulli random variable that indicates if object i is sampled, but the sampling rate is weighted by a function of s_i, such that:

$$f_{X_i}(x) = \begin{cases} p(s_i) & \text{x = 1} \\ 1-p(s_i) & \text{x = 0,} \end{cases} \tag{42}$$

where $p(s_i)$ is the weighting function and $\mathbb{E}[s]$ hereafter is the average object size:

$$p(s_i) = \frac{r \times s_i}{\mathbb{E}[s]}. \tag{43}$$

The total sampling number M_{ws} is:

$$M_{ws} = \sum_i^M X_i \tag{44}$$

$$= \sum_i^M \frac{r \times s_i}{\mathbb{E}[s]} = r \times \frac{\sum_i^M s_i}{\mathbb{E}[s]} \tag{45}$$

$$= r \times M, \tag{46}$$

which is equal to spatial sampling. The estimated unique bytes $\mathbb{E}[WSS_{ws}]$ and variance $Var[WSS_{ws}]$ are as follows:

$$\mathbb{E}[WSS_{ws}] = \mathbb{E}[\sum_i^M X_i s_i / p(s_i)] \tag{47}$$

$$= \mathbb{E}[\sum_i^M X_i / p(s_i)] \mathbb{E}[\sum_i^M s_i] \tag{48}$$

$$= M \times \mathbb{E}[s] = \mathbb{E}[WSS_s], \tag{49}$$

$$Var[WSS_{ws}] = \sum_i^M Var[X_i s_i / p(s_i)] \tag{50}$$

$$= \sum_i^M (s_i^2 / p^2(s_i) \times p(s_i)(1-p(s_i))) \tag{51}$$

$$= \sum_i^M (\frac{\mathbb{E}^2[s]}{r^2} \times (r \times s_i / \mathbb{E}[s])(1 - r \times s_i / \mathbb{E}[s])) \tag{52}$$

$$= \sum_i^M (s_i)(\mathbb{E}[s]/r - s_i) \tag{53}$$

$$= \frac{M}{r} \times \mathbb{E}^2[s] - M \times \mathbb{E}[s^2]. \tag{54}$$

The value of $\mathbb{E}[WSS_{ws}]$ is equal to spatial sampling results, which is unbiased, but Weighted Sampling has a different variance. Finally, we get the coefficient of variation

$C_v[WSS_{ws}]$ of Weighted Sampling:

$$C_v[WSS_{ws}] = \frac{\sqrt{Var[WSS_{ws}]}}{\mathbb{E}[WSS_{ws}]} \tag{55}$$

$$= \frac{\sqrt{M/r \times \mathbb{E}^2[s] - M \times \mathbb{E}[s^2]}}{M \times \mathbb{E}[s]} \tag{56}$$

$$= \sqrt{\frac{1}{rM} - \frac{\mathbb{E}[s^2]}{M\mathbb{E}^2[s]}} \tag{57}$$

$$\leq \sqrt{\frac{1-r}{rM}}. \tag{58}$$

□

A Artifact Appendix

A Abstract

The artifact of this paper contains the source code of FLOWS and our implementation of SHARDS, HCPP, and LPCA. We use open-source object storage or cache traces to evaluate the accuracy of these methods.

B Description & Requirements

B.1 How to access. We open source our code on a GitHub repository:
https://github.com/JasonGuo98/FLOWS-Balanced-MRC-Profiling-for-Heterogeneous-Object-Size-Cache

B.2 Hardware dependencies.

- RAM: 32GB
- Storage: 1TB

B.3 Software dependencies.

- Compiler: GCC version 8.5.0
- Python 3.8.10

B.4 Benchmarks.

- We have summarized the publicly available traces in Table 2.

C Set-up

The compilation, trace file downloading, and processing, as well as the plotting scripts for this artifact, are thoroughly explained in the README.md file of the repository.

D Evaluation workflow

D.1 Major Claims.

- *(C1): With a sampling rate of 0.01, FLOWS reduces the error of BMRC profiling by a factor of 16 compared to Carra's method and a factor of 3 in OMRC profiling. This conclusion is illustrated in Figure 12.*
- *(C2): With a fixed sample number of 8K, 16K, and 32K, FLOWS reduces the error of BMRC profiling by a factor of 6 compared to Carra's method. This conclusion is illustrated in Figure 14.*

D.2 Experiments.

Experiment (E1): [Fixed sampling rate evaluation]: compare MRC profiling methods using a fixed sampling rate and obtain the error compared to the ground truth trace.
[Preparation] According to the instructions in the README.md, download all the required trace files and proceed with the steps of decompression and formatting. Setup environments and compile necessary tools. This step will require approximately 800GB of storage space.

[Execution] According to the "Run Evaluations" section in the README.md file, execute all methods except for the FIX_NUM part. The MRCs will be saved in the results folder in CSV format. Run the script to plot Figure 12 and obtain the results.

[Results] Carra's method exhibits an average modeling error of 7.29% for BMRC, while FLOWS achieves an average BMRC modeling error of 0.44%. For OMRC, Carra's method has an average modeling error of 1.33%, while FLOWS achieves an average OMRC modeling error of 0.43%

Experiment (E2): [Fixed sampling number evaluation]: compare sampling based MRC profiling methods using a fixed sampling number.

[Preparation] Same as E1

[Execution] According to the "Run Evaluations" section in the README.md file, execute all methods related to FIX_NUM. The MRCs will be saved in the results folder in CSV format. Run the script to plot Figure 14 and obtain the results.

[Results] In BMRC profiling, FLOWS reduces the average MAEQ by 6× (from 0.060 to 0.010) compared to the HCPP method.