Efficient Miss Ratio Curve Computation for Heterogeneous Content Popularity

Damiano Carra, *University of Verona, Italy;* Giovanni Neglia, *Inria, Université Côte d'Azur, France*

https://www.usenix.org/conference/atc20/presentation/carra

This paper is included in the Proceedings of the 2020 USENIX Annual Technical Conference.

July 15–17, 2020

978-1-939133-14-4

Efficient Miss Ratio Curve Computation for Heterogeneous Content Popularity

Damiano Carra
University of Verona, Italy

Giovanni Neglia
Inria, Université Côte d'Azur, France

Abstract

The Miss Ratio Curve (MRC) represents a fundamental tool for cache performance profiling. Approximate methods based on sampling provide a low-complexity solution for MRC construction. Nevertheless, in this paper we show that, in case of content with a large variance in popularity, the approximate MRC may be highly sensitive to the set of sampled content. We study in detail the impact of content popularity heterogeneity on the accuracy of the approximate MRC. We observe that few, highly popular, items may cause large error at the head of the reconstructed MRC.

From these observations, we design a new approach for building an approximate MRC, where we combine an exact portion of the MRC with an approximate one built from samples. Results for different real-world traces show that our algorithm computes MRC with an error up to 10 times smaller than state-of-the-art methods based on sampling, with similar computational and space overhead.

1 Introduction

Caches have been widely used in different contexts to improve system performance, from CPU, to disk, to web. As the architectures become more complex, with multi-core CPUs, or clusters of machines, caches maintain a key role in providing fast access to the most used content. Being a shared resource, a cache may be misused by some aggressive processes or application, hurting the performance of other processes.

To provide a fair sharing, one may virtually divide the cache and assign dynamically different portions to specific applications or types of applications. For instance, Sundarrajan *et al.* [33] show that, in case of Web caches, video streaming, web browsing, and software updates have extremely different content access patterns and cache resource requirements. Similarly, when multiple VMs run on the same physical host, efficient sharing of storage resources, like a SSD used as cache, needs detailed VM profiling [21, 24, 28, 32]. Analogous observations have been made in different contexts, such as multiprocessor systems [10, 19] and distributed processing in datacenters [27, 35].

The most important performance metric for a cache is usually the *hit ratio*. For cache partitioning, it is necessary to quantify the hit ratio a given application would experience given the amount of available cache space. This relation is captured by the *Miss Ratio Curve* (MRC), which gives the miss ratio as a function of the cache size. The use of MRCs can be helpful also in contexts where caches can be provisioned *on demand* [7, 29] with a pay-as-you-go model, as it is in the case for cloud caches [1–3].

MRC can be computed analytically for many caching policies—sometimes exactly [14, 20], more often approximately [15, 16]—but only under idealized models for the request process. Real traffic usually exhibits complex patterns that diverge from these models.

A more common approach, dating back to Mattson's seminal work [25], is to compute the MRC directly from the trace of the specific workload. The MRC can be built with $\mathcal{O}(\log M)$ computational complexity per request, and $\mathcal{O}(M)$ memory [9, 13], where M is the number of distinct items that are requested. Since content popularity (and consequently the MRC) may vary over time, the usual approach is to select an interval of time over which the traffic request process may be considered stationary. Then, the requests observed during this interval are used to build a MRC, which drives the resource assignment for the next interval. In case of high traffic rate, if we need to continuously build many MRCs for different application types, computational complexity and memory requirements may represent a heavy burden [7, 33].

For this reason, by trading accuracy with computational complexity and memory, recent works propose to compute approximate MRCs with $\mathcal{O}(1)$ operations per request, and $\mathcal{O}(1)$ memory [17, 29, 36, 37]. Such low-complexity solutions are based on the common idea of sampling the trace.

Limitation of the prior work. Sampling has been applied widely in different domains. A potential pitfall of sampling is the introduction of biases. In building the approximate MRC, there could be two approaches: sampling the *requests* [4, 34], or sampling the *items* and observing the requests for those items [36, 37]. Request sampling introduces a bias [30, 39], which motivated the introduction of item sampling, also called *spatial sampling*. Nevertheless, if request rates vary greatly across items, spatial sampling can be biased too, a fact that was implicitly acknowledged by Waldspurger *et al.* [37]. To the best of our knowledge, we are the first to thoroughly address and explain such a bias in detail.

As an example, the left column of Figure 1 shows the exact and approximate MRCs using the LRU eviction policy, built from various samples, considering traces with different traffic characteristics. In particular, item request frequencies, usually referred to as *popularities*, are Zipf-distributed with two different values of the Zipf exponent (α)—experiments' details are provided in Section 3, but they are not essential to understand what follows. With higher values of α, the distribution becomes more skewed, and therefore popularities become highly heterogeneous. The approximate MRCs in the left column are obtained using SHARDS [37] with a constant sampling rate $R = 0.01$. The experiments show that SHARDS is able to obtain an accurate MRC when item popularity is not highly skewed. But, as heterogeneity increases, the error drastically increases. Waldspurger *et al.* [37] recognized this possible bias, and proposed the variant SHARDS$_{\text{adj}}$ that partially solves the problem. The curves in Figure 1, right column, show that SHARDS$_{\text{adj}}$ correctly estimates the tail of the MRC, but not its head, with large errors for high miss ratio values (above 30% in the bottom subfigure) that may even exceed 100% (top subfigure). Miss ratios above 70% are the norm for many caches, including HDD [37] and SSD ones [21, 24], and hierarchical web caches, where the higher level resides in RAM [5, 26]. Caches consist of fast, expensive storage, and they are inherently a scarce, shared resource. An accurate assignment requires the knowledge of the MRC for *any size*, even when the miss ratio is large.

Contributions. In this work we study the impact of heterogeneity on the accuracy of the approximate MRC. With the help of different sets of experiments, and a model of a representative scenario, we observe that highly popular item play a fundamental role, and we shed lights on the fact that the MRC is highly sensitive to the specific content sampled. Consequently, we design a new approach, where we combine exact MRC computation for small values of the cache capacity (which is mainly influenced by popular items) with approximate computation for larger values. We evaluate our scheme with both synthetic and real-world traces. Our results show that our method is able to reconstruct the MRC with an average error up to 10 times smaller than state of the art approaches, with the same complexity. We also consider a scenario where items have heterogeneous sizes, and show how our solution correctly addresses it.

Roadmap. The remaining of the paper is organized as follows. In Section 2 we provide the background information and discuss the related work. In Section 3 we study the impact of popular items on the accuracy of the spatial sampling approaches. Section 4 presents our solution, which is evaluated in Section 5. Section 6 provides additional considerations on the scheme we propose, and Section 7 concludes the paper.

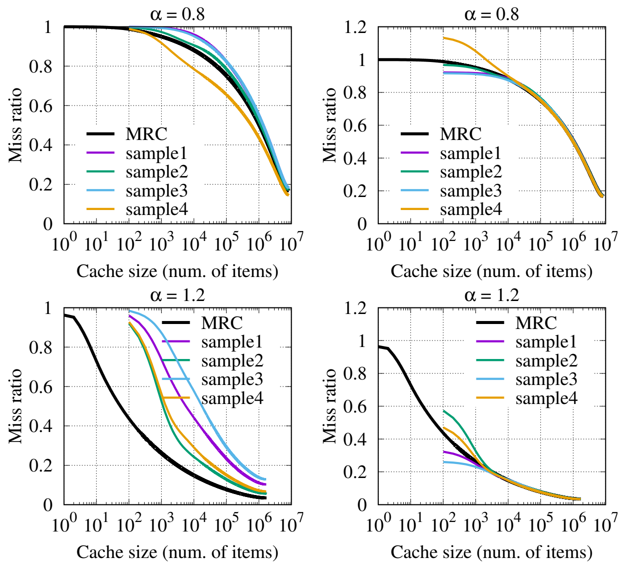

Figure 1: Approximate MRCs built from samples compared to the exact MRC for different values of the parameter α of the Zipf distribution used for item popularity ($R = 0.01$): SHARDS (left column) and SHARDS$_{\text{adj}}$ (right column).

2 Background and Related Work

The MRC can be computed with a single pass on the request trace if the eviction policy satisfies the inclusion property, *i.e.*, the set of items stored in the cache at a given time is a subset of the set of items that would be stored if the cache had a larger size [25]. Widely adopted policies such as LRU, LFU, and MRU satisfy such property, therefore MRCs are useful in many practical systems.

Methods with different computational and memory requirements have been proposed to build the MRC [6, 37]. We describe here a specific algorithm suitable when all items have the same size. The caching policies listed above all maintain an ordered list of the items in the cache, where, at any instant, the last item in the list is the current candidate to be evicted. The MRC algorithm goes through the trace, maintaining an ordered list $\mathcal{T}$ of references to the items mimicking how the corresponding caching policy would work if the cache size were infinite. Given a request for item j, if the item is not in the cache, we have a *cold miss*. If the item is in the cache, then the algorithm determines its current position (called the *reuse distance*) and updates an empirical histogram of reuse distances. Once the trace is analyzed, the histogram is normalized dividing each value by the total number of requests. By summing the histogram values up to a given capacity C, one obtains the corresponding hit ratio, whose one's complement is the miss ratio.

Exact MRC computation requires $\mathcal{O}(M)$ memory, where M is the number of distinct items in the trace, and has a computational complexity of $\mathcal{O}(\log M)$ per request due to the access to $\mathcal{T}$, which can be implemented with a tree data structure [40]. The approximate solutions based on sampling may adopt two approaches: request sampling and item sampling. The solutions based on request sampling—such as sampling every n requests [4], or sampling for small intervals of time [34]—are known to be biased [30, 39]. To overcome these issues, item sampling has been recently proposed for computing the reuse distance to characterize the use of storage memory [38] or program locality in single core [12] and multi-core architectures [30], and for building approximate MRC with low computational complexity [17, 36, 37].

In this work we consider the solution adopted by SHARDS [37]. SHARDS selects randomly a fraction R of the items, computes the MRC considering only the requests from these items, and then scales the cache capacity on the X-axis by a factor $1/R$. The item selection is done using a hash of the item identifier, id_j. Since sampling may exclude or include very popular items, the authors of SHARDS proposed an adjustment, called SHARDS$_{\text{adj}}$, in which the estimated miss ratios are scaled up by the ratio between the actual and the expected number of sampled references.

How to measure the Accuracy. In evaluating the approximate MRCs, accuracy is usually measured using the Mean Absolute Error (MAE): this is the average of the absolute differences between the exact and the approximate MRC for all cache sizes considered. Such a metric is easy to interpret, but it gives the same importance to all different values of cache size. The following simple example illustrates a potential problem. Figure 2 shows the exact and approximate MRC obtained with SHARDS for a publicly available trace which we name *ms-ex*—trace details in Section 5.3. In the figure on the left, the two curves look similar and indeed the MAE is only 0.025, *i.e.*, if we pick an arbitrary value of the cache size, on average the approximate MRC allow us to estimate the miss rate with an error of $\pm 2.5\%$. The figure on the right contains the same information but using a log scale for the X-axis. It appears that the average error is not representative. In fact, the error for small cache sizes can be 5-6 times bigger.

In this paper we introduce a *new metric*—MAEQ, Mean Absolute Error per Quantile—that maintains the simplicity of MAE, but it takes into account how much the MRC varies in the different intervals. While the MAE provides the average error for a cache size sampled uniformly at random, the MAEQ provides the average error when a *miss ratio* is sampled uniformly at random. In particular, the metric is based on the concept of *quantiles*. We consider different uniformly spaced quantiles for the miss ratio and identify the corresponding cache size intervals. For instance, if we consider

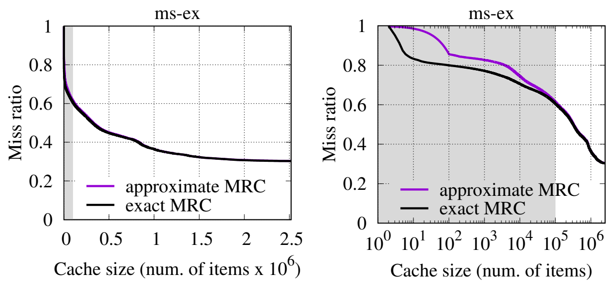

Figure 2: Error between approximate and exact MRCs (without and with log scale in the X-axis). The shaded area highlights the portion where the approximate and exact MRCs differ the most.

the quantiles at 0.8, 0.6, 0.4, and 0.2 in Figure 2, right, we identify the following intervals for cache sizes: The first interval goes from 0 to 50, the second one from 50 to 10^5, the third from 10^5 to $8 \cdot 10^5$, and the fourth one from $8 \cdot 10^5$ till the end. Note that no portion of the MRC falls below 0.2, so the last quantile is not considered. Once we have identified the ranges that correspond to the quantiles, we compute the MAE in each range, and then we average the MAE. More formally, $MAEQ = \sum_{i=1}^{Q^*} MAE_i/Q^*$, where the interval i is defined by the quantile, and Q^* is the number of quantile intervals considered. In practice, we consider quantiles with a step increment of 0.01: for each variation of 0.01 in the miss ratio, we compute the MAE, and then we take the average of the MAEs. If we compute the MAEQ for the above trace, we obtain 0.090, which provides a better idea of the accuracy of the approximate MRC, when we look at different miss ratio ranges.

3 Evaluation of Spatial Sampling Approaches

3.1 Evaluation Methodology and Settings

In this section we evaluate the impact of content popularity heterogeneity on the accuracy of the approximate MRC. We consider the SHARDS and SHARDS$_{\text{adj}}$ approaches [37] with a fixed sampling rate, denoted by R, which varies from 0.1 to 0.001. In order to have highly controllable experiments, we first consider a set of traces generated according to the Independent Reference Model (IRM), in which the probability that the next request for item i is constant over time and independent from the previous requests. We call this probability the *popularity* of content i and denote it as p_i. In particular, we use the Zipf popularity distribution ($p_i \propto i^{-\alpha}$) with different values of the parameter α. We have tested different combinations of catalogue size and trace length; in what follows we report the representative results for the case with 50M requests for a set of 10M items and different values of the parameter α.

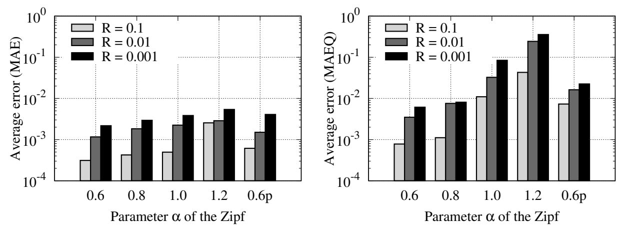

Figure 3: Accuracy for different values of the parameter α of the Zipf distribution used for item popularity, and for different sampling rates R: MAE (left) and MAEQ (right).

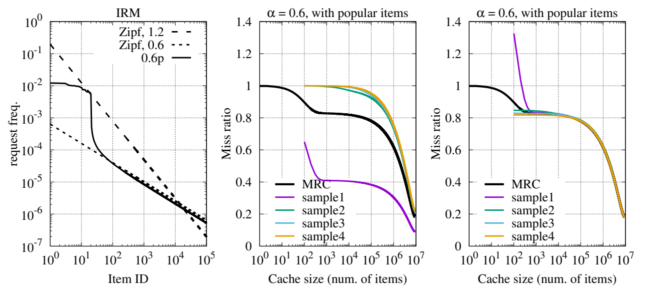

Figure 4: Distributions used for the experiments (left) and MRCs built from samples for the "0.6p" case ($R = 0.01$) through SHARDS (center) and SHARDS$_{\text{adj}}$ (right).

3.2 Results with the IRM Traces

Figure 1 shows the exact and approximate MRCs for different values of α (sampling rate $R = 0.01$). The results in left columns have been obtained using SHARDS, while the ones in the right column using SHARDS$_{\text{adj}}$. Note that, with spatial sampling, we are able to build the MRC at points that are multiples of $1/R$—this is why the approximated MRCs start at $1/R = 100$.

In case of SHARDS, as α increases, the error increases significantly. The problem is partially solved by SHARDS$_{\text{adj}}$, whose effect is to decrease the error in the tail of the MRC. This is obtained by rescaling the approximated MRC, but such a rescaling has an impact on the whole MRC, and it leads to significant errors for small cache values yielding miss ratios larger than 1. This detail was not discussed in the SHARDS work [37]. Our model in Section 3.3 explains these findings.

In Figure 3 we show SHARDS$_{\text{adj}}$ MAE and MAEQ values for different values of the sampling rate R and the Zipf parameter α—the case labeled as "0.6p" will be discussed later. Each value has been computed averaging five different samples. The MAE indicates an error smaller than 0.006, but, as we discussed above, such metric considers all cache sizes equally important. The MAEQ, instead, indicates an average error over the quantiles that, for $\alpha = 1.2$ and $R = 0.001$ may be as high as 0.35, which better describes the difference between the exact and the approximate MRC.

On the head of the MRC. The presence of a relatively small number of *highly popular items* determines the accuracy of MRCs. If we observe the empirical item popularity distribution in real-world traces—we will show some examples in Figure 9—we notice that there are often two groups of items: a small group with very popular items, and a large one with much less popular items.

Inspired by these real-world traces, we modify a Zipf distribution with $\alpha = 0.6$ by adding 20 popular items, whose popularity is randomly selected between 0.005 and 0.01. Popularities have been normalized to guarantee that their sum equals one. Figure 4, left, compares this new popularity distribution, labeled as "0.6p", with the Zipf distribution used in the previous section. Figure 4, center and right, shows approximate MRCs for a sampling rate $R = 0.01$ with SHARDS and SHARDS$_{\text{adj}}$, respectively. With the addition of a few popular items (20 out of 10M) the accuracy of the approximate MRC is significantly affected. The MAE and the MAEQ are shown in Figure 3, with the label "0.6p." This experiment confirms that, when sampling fails to capture the contribution of the popular items, the result may be heavily biased. On the other hand, less popular items have limited impact on the MRC and they may be sampled randomly.

3.3 Understanding the role of popular items

We analyze a *simple scenario* for which we can derive an approximate model. Consider a finite set of items where there is a single very popular item (content c_1), with popularity p, and M items with approximately homogeneous popularity, *i.e.*, each item has a popularity of approximately $(1-p)/M$, and $p \gg (1-p)/M$. The request sequence is generated following the IRM model. Let r_n denote the n-th request. We consider the *reuse distance*, *i.e.*, the number of unique references between two references to the same item [40], and the *reuse time*, *i.e.*, the total number of references between two references to the same item [17].

For small cache sizes, the reuse distance can be approximated by the reuse time, *i.e.*, misses for content c_1 occur (almost) every time the reuse time for the content exceeds the cache size C.[1] The reuse distance is geometrically distributed with expected value $1/p$. Then, the miss probability for content c_1 starts decreasing significantly as the cache reaches size $1/p$, and decreases exponentially fast for bigger and bigger cache sizes. Once the cache size is 3 to 4 times larger than $1/p$, content c_1 is highly likely to be in the cache and the miss ratio is at most $1-p$. For larger cache sizes, the miss rate still decreases because of the contribution of the unpopular items. The decrease is now linear in the cache size. This reasoning can be made formal, *e.g.*, using a simple model based on the Che's approximation [11], and we obtain that the miss ratio

[1]This is an approximation because requests for other items contribute to move content c_1 closer to the tail only if they are misses. But, for small cache sizes, almost all requests for the M unpopular items generate misses.

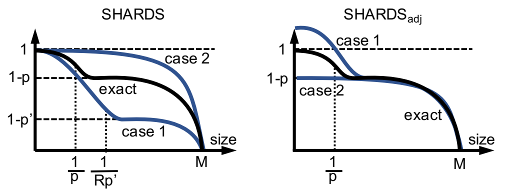

Figure 5: Simple scenario: exact MRC, along with two cases of approximate MRCs.

when the cache has capacity $C \in [0, M]$ is:

$$m(C) \approx pe^{-pC} + (1-p)\left(1 - \frac{C}{M}\right). \quad (1)$$

Figure 5 shows a sketch of what the miss ratio looks like using a logarithmic scale for the capacity axis—curve labeled "exact."

Now, assume we sample the items with sampling rate R. Either the set of sampled items contains the very popular content (with probability R), or it does not contain it (with probability $1-R$). In the following, we consider these two cases and show that the resulting MRCs differ significantly from the exact MRC.

Case 1: content c_1 is sampled. The sample contains requests for content c_1 and, on average, $M' = RM$ unpopular items. The fraction of requests for content c_1 is $p' = p/(p+(1-p)R) > p$. The exact MRC of the sampled trace is determined by (1) with M' and p' replacing respectively M and p. For the approximate MRC, we replace C with RC and we obtain:

$$m(C) \approx p'e^{-p'RC} + (1-p')\left(1 - \frac{C}{M}\right). \quad (2)$$

The curve labeled "case 1" in Figure 5, left, corresponds to (2). We observe a fast decrease of the miss ratio for cache sizes around $1/(p'R)$ from 1 to $1-p' < 1-p$. Then, the approximate MRC underestimates the miss ratio at least for large values of the cache size.

Case 2: content c_1 is not sampled. In this case there are on average $M' = RM$ equally popular items in the cache, then the miss ratio is

$$m(C) \approx 1 - \frac{C}{M}, \quad (3)$$

and is represented by the curve labeled "case 2" in Figure 5, left.

Adjustment proposed in SHARDS$_{\text{adj}}$. SHARDS$_{\text{adj}}$ rescales the estimated MRC by a factor $N_S/(RN)$, where N_S is the number of requests observed in the sample. For *Case 1*, we have $N_S = pN + RN(1-p)$, and we obtain

$$m_{\text{adj}}(C) \approx \frac{p}{R}e^{-p'RC} + (1-p)\left(1 - \frac{C}{M}\right). \quad (4)$$

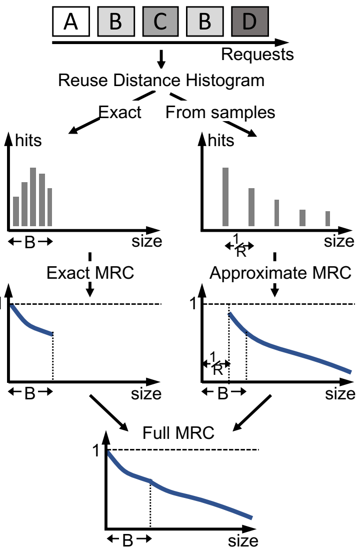

Figure 6: Approximate MRC building process.

We observe then how SHARDS$_{\text{adj}}$ removes the bias for large capacity values but amplifies it for small ones, leading to a miss ration greater than 1 in this example ($m_{\text{adj}}(0^+) \approx p/R + (1-p) > 1$). Similarly, for *Case 2*, we obtain

$$m_{\text{adj}}(C) \approx (1-p)\left(1 - \frac{C}{M}\right). \quad (5)$$

SHARDS$_{\text{adj}}$ correctly predicts the tail of the MRC, as for *Case 1*, but it now underestimates the head. Figure 5, right, shows the curves that corresponds to *Case 1* and *Case 2* when using SHARDS$_{\text{adj}}$.

4 Proposed solution

In the previous section we observed that popular and less popular items have different impacts on the MRC building process. These observations motivate our solution. We design a scheme, where we combine an exact MRC for small cache sizes (where the miss ratio depends mostly on the highly popular items), with an approximate MRC built from sampled items for large cache sizes. In particular, for the approximate MRC we use SHARDS$_{\text{adj}}$, because it predicts the MRC tail better. The approach is qualitatively illustrated in Figure 6. For the portion of the MRC built from samples, the general scheme can adopt either a *constant sampling rate* or an adaptive sampling rate to achieve *constant computational complexity*. We discuss these two schemes below.

Constant Sampling Rate Scheme. Algorithm 1 describes the solution we propose. We assume the use of the LRU eviction policy, but the scheme may be adapted easily to any policy that satisfies the inclusion property. The algorithm requires two parameters: B, which is the number of items for

Algorithm 1: Approximate MRC building process

```
input  : $B$, number of positions for the exact MRC
input  : $R_s$, sampling rate for the approximate MRC
input  : request sequence
1  $\mathcal{T}_e \leftarrow$ countingBTree(); $V_e \leftarrow$ reuseVector();
2  $\mathcal{T}_s \leftarrow$ countingBTree(); $V_s \leftarrow$ reuseVector();
3  foreach request $r$ for item with id $j$ do
4      if ($j \in \mathcal{T}_e$) then
5          pos$_j \leftarrow$ remove($j, \mathcal{T}_e$);
6          update $V_e$ at pos$_j$;
7      add $j$ to $\mathcal{T}_e$;
8      if (size($\mathcal{T}_e$) $> B$) then
9          remove last item from $\mathcal{T}_e$;
10     if (hash($j$) mod $P < R_s P$) then
           // sampled item
11         if $j \in \mathcal{T}_s$ then
12             pos$_j \leftarrow$ remove($j, \mathcal{T}_s$);
13             update $V_s$ at pos$_j$;
14         add $j$ to $\mathcal{T}_s$;
15 MRC[0..$B$] $\leftarrow$ buildExactMRC($V_e$);
16 MRC[$B$..$\infty$] $\leftarrow$ buildApproxMRC($V_s$);
```

the exact MRC, and R_s, which is the sampling rate. Given a set of requests for an unknown catalogue of items, we maintain two tree data structures respectively to build the exact MRC ($\mathcal{T}_e$), and the approximate one ($\mathcal{T}_s$). $\mathcal{T}_e$ size equals B item references, while $\mathcal{T}_s$ depends on the number of items in the trace and the sampling rate. *Splay Trees* or *Counting BTrees* are possible candidates for such data structures, since they have logarithmic complexity for the insert/delete operations. Instead, for the lookup we maintain a hash table.

Once the trace is processed, we build the exact MRC ($m_e(C)$) and the approximate one ($m_s(C)$). We then connect by continuity the approximate MRC starting from B, and modulating exponentially any potential step discontinuity (equal to $m_e(B) - m_s(B)$), i.e.

$$m(C) = \begin{cases} m_e(C) & \text{if } C \leq B, \\ m_s(C) + (m_e(B) - m_s(B))e^{-\frac{C-B}{4B}} & \text{if } C > B. \end{cases}$$

Notice that, if we set $B = 0$, we obtain SHARDS$_{\text{adj}}$ scheme with constant sampling rate.

As for the parameters B and R_s, we provide a sensitivity analysis in Section 5, while we discuss the general guidelines for setting them in Section 6. Here we anticipate that in our experiments no particular tuning was required.

As for R_s, the considerations made by Waldspurger *et al.* [37] are valid also in our case. As for B, in our experiments we notice that, even for traces with millions of items, only few hundreds have very high popularity, and a value of B as low as 10^3 item references provides very accurate results. For the same trace, if we use $R_s = 0.01$, the memory necessary to hold the references to the sampled items is of the order of 10^4 item references, and $B = 10^3$ adds only a small fraction to that memory consumption.

The proposed scheme has a complexity $\mathcal{O}(\log R_s M)$, where M is the number of distinct items in the trace, which is due to accesses to $\mathcal{T}_s$. As for $\mathcal{T}_e$, since its size is constant (B item references), the cost for the data structure operations is $\mathcal{O}(1)$.

Constant Complexity Scheme. A fixed sampling rate has computational complexity and memory requirements that depend on the number of sampled items, which may grow as we consider longer and longer traces. Waldspurger *et al.* [37] propose an adaptive sampling method to maintain a $\mathcal{O}(1)$ complexity per requests. This can be achieved by fixing a priori the number $s_{\max}$ of items to sample, and then tracking the $s_{\max}$ items with smallest values of the hash function. As more requests are processed, the sampling rate implicitly converges to the minimum value required to maintain $s_{\max}$ references. Our mixed approach can be easily adapted in this direction, by fixing the size of $\mathcal{T}_s$.

5 Evaluation

5.1 Experimental Methodology

We compare our solution with the state-of-the-art approaches based on spatial sampling [37] [36], both when the sampling rate is fixed, and when the complexity is constant. If not otherwise stated, we consider the SHARDS$_{\text{adj}}$ variant

In case of fixed sampling rate, SHARDS$_{\text{adj}}$ adopts a sampling rate R. Given a trace with M distinct items, the scheme keeps track of RM items. For a fair comparison with our scheme, we adopt a sampling rate R_s that leads to keep track of the same number of item references, i.e., $B + R_s M = RM$. If not otherwise stated, we set $B = 1000$ and $R_s = R - B/M$. In most of our experiments, M is of the order of few millions, so with $R = 0.01$, R_s is slightly smaller than R. In Section 5.2 we will show the impact of B on the accuracy.

In case of constant complexity, SHARDS$_{\text{adj}}$ fixes the number of operations by maintaining a constant number of item references $s_{\max}$. This means that the scheme requires $2\log(s_{\max})$ operations per request (where the factor 2 is due to the additional data structure to track the $s_{\max}$ items with smallest hash). In our case, there is an additional cost of $\log B$ due to the exact portion of the MRC, so, to make a fair comparison, we set the size of $\mathcal{T}_s$ to a value of $s'_{\max}$ such that $\log B + 2\log(s'_{\max}) = 2\log(s_{\max})$, *i.e.*, $s'_{\max} = s_{\max}/\sqrt{B}$.

In the following sections, we will report simply R or $s_{\max}$ used for SHARDS$_{\text{adj}}$ [37]. The corresponding parameters of our scheme are computed as explained above.

5.2 IRM traces

We start testing our solution with the IRM traces described in Section 3. We focus on the two more problematic popularity

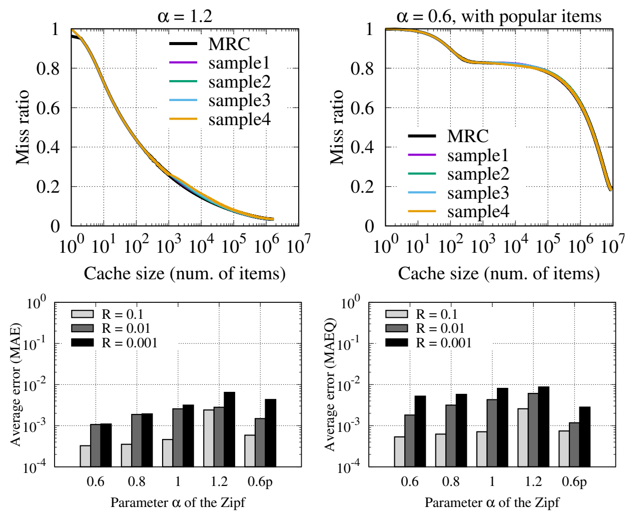

Figure 7: MRCs built from samples with our approach (top) and accuracy (MAE bottom left, MAEQ bottom right).

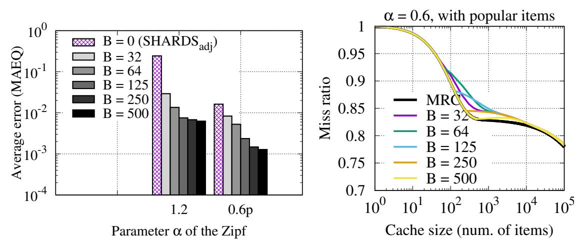

Figure 8: Accuracy for different values of B ($R = 0.01$), and the corresponding MRC ($\alpha = 0.6p$, zoom on the head of the MRC).

distributions: the Zipf one with $\alpha = 1.2$, and the Zipf one with $\alpha = 0.6$ modified by introducing 20 very popular items, labeled as "0.6p." Figure 7 shows the approximate MRC, along with the MAE and MAEQ. Our solution is able to build approximate MRCs using the same amount of memory as SHARDS$_{\text{adj}}$. The average error per quantile reaches at most 0.008 in case of an equivalent sampling rate as low as $R = 0.001$—the corresponding value of R_s is 0.0004 for $\alpha = 1.2$ and 0.0008 for the "0.6p" case. With SHARDS$_{\text{adj}}$, instead, the error with $R = 0.001$ was more than 40 times larger (0.35) for $\alpha = 1.2$ (Figure 3, right). For most of the settings, our solution also slightly improves the MAE, being equal only for $\alpha = 1.2$ and $R = 0.001$.

Impact of B. The proposed scheme has a parameter B, which is the maximum cache size considered for the exact MRC. Recall that, if $B = 0$, we obtain SHARDS$_{\text{adj}}$'s results. Figure 8, left, shows the impact of the accuracy for different values of B. We consider the default case with $R = 0.01$. Since the ratio B/M is less than 10^{-3}, then the equivalent R_s is approximately 0.01 too.

As we increase B, there is a significant error reduction. The reason can be seen looking at the approximate MRC, which is shown in Figure 8, right. Since we have a sampling rate $R_s \approx R = 0.01$, only one of the 100 most popular items, on average, is sampled, so the approximate MRC is very inaccurate in the range $[1, 100]$. As our algorithm uses the approximate sample-based MRC starting from $B+1$, as long as B is smaller than 100 (*i.e.*, $1/R_s$), the error of the approximate MRC also affects the final MRC.

Table 1: Trace characteristics

name	year	# items	# req	reference
fiu	2008	6.1 M	14.3 M	[22]
ms-ex	2007	2.6 M	8.9 M	[18]
ms-dev	2007	6.3 M	18.2 M	[18]
systor	2016	12.7 M	34.3 M	[23]
CDN	2015	1.6 M	11.2 M	[8]

5.3 Real-world traces

We consider a set of publicly-available block I/O traces from SNIA IOTTA repository [31], along with traces from Akamai, a major CDN provider. Table 1 summarizes the characteristics of the traces. From the SNIA IOTTA repository, we have considered the most recent trace—labeled as *systor* [23]—along with older traces collected at FIU [22] and at Microsoft [18].

For experimental reproducibility, we report here the details of the traces. For the *fiu* traces [22], we consider the subtrace IODedup/Web. The *ms-ex* is the trace named "Microsoft Enterprise Traces, Exchange Server Traces" [18], which have been collected for Exchange server for a duration of 24-hours—we consider the first 3.5 hours. The *ms-dev* is the trace named "Microsoft Production Server Traces - Development Tools Release" [18]. The *systor* traces [23] collect requests for different block storage devices over 28 days: we consider one day (March, 9th) of the device called "LUN2." Finally, the *CDN* trace [8] contains multiple days of traffic, of which we consider portions of 6 hours—we tested different intervals finding similar qualitative results.

Request distribution. Figure 9 shows the empirical popularity distribution for two representative traces (*systor* and *CDN*), since the others are similar. The distributions show the presence of two distinct groups of items: a head with highly popular items, and a power-law tail—the figure shows the exponent α of the fitting power law distribution.

Approximate MRC with Constant Sampling Rate. Figure 10 shows the comparison between the exact MRC and the ones obtained with SHARDS$_{\text{adj}}$ (left column) and with our approach (right column) for some representative traces. In all cases we consider $R = 0.01$ and $B = 1000$. We have also computed the MAEQ (bottom subfigure).

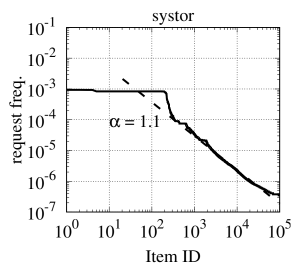

Figure 9: Item popularity distribution of the traces.

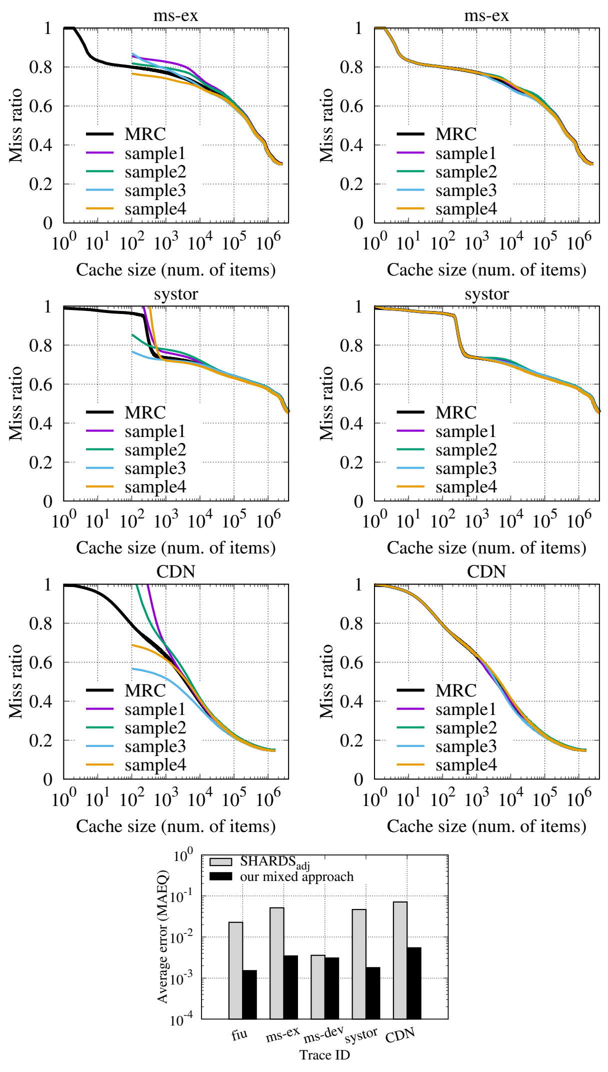

Figure 10: MRCs built from samples with SHARDS$_{adj}$ (left) and our approach (right). Bottom: MAEQ.

Except for the *ms-dev* trace, SHARDS$_{adj}$ fails to build an accurate MRC, with MAEQ in the range 0.05–0.10. With our approach, instead, the MAEQ is always below 0.01, and the accuracy can be appreciated visually comparing the approximate and exact MRCs. The *ms-dev* trace is representative of the case in which SHARDS$_{adj}$ provides good accuracy: our approach is able to provide equally accurate results.

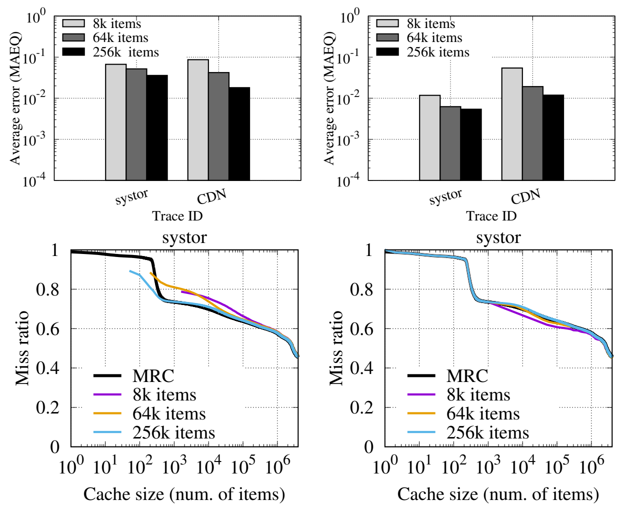

Figure 11: Top: Accuracy (MAEQ) for constant complexity with SHARDS$_{adj}$ (left) and our approach (right). Bottom: MRC of a representative trace, with SHARDS$_{adj}$ (left) and our approach (right).

Approximate MRC with Constant Complexity. We recall (Section 4) that a constant complexity per request is achieved by putting a cap on the number of sampled items $s'_{\max}$, and then to the total memory used $(B + s'_{\max})$. Figure 11 (top) shows the MAEQ with SHARDS$_{adj}$ and with our approach for different memory sizes—if not otherwise stated $B = 1000$. Figure 11 (bottom) shows the MRC. As the number of items increases, the head of the approximate MRC with SHARDS$_{adj}$ converges to the exact shape. With our approach, the MAEQ is smaller, because the first B positions are always correct.

Overheads. The experimental campaign has been specifically designed to compare our scheme and SHARDS$_{adj}$, and the memory used in both scenarios—constant sampling rate and constant complexity—as described in Section 5.1. We have evaluated the CPU usage using user and system time components as reported by `/usr/bin/time`. Our experiments confirm that SHARDS$_{adj}$ has a 75x speed up compared to the exact MRC computation [37]. Our scheme performs slightly worse with CPU usage on average 10%, and at most 20%, higher than SHARDS$_{adj}$. While the asymptotic computational complexity of the two schemes is the same, ours requires indeed a few more operations per request. Under SHARDS$_{adj}$, at each request, we need to compute the hash of the item identifier. With our scheme, in addition to this, we need to insert the item at the head of the tree data structure that keeps track of the first B positions (in case of a hit we first need to remove the item, but this does not happen at every request). The fact that CPU load only increases by 10% suggests that the overhead due to the additional insertion/deletion operations appears negligible in comparison to the hash computation cost. On a consumer laptop, our solution processed the traces in Table 1 (which span multiple hours) in less than 30s. It can the be used, not only offline on collected traces, but also online, as requests arrive. In the latter case, one may select a duration for the observation interval, e.g., one hour or less, and the MRC is computed at the end of the interval.

6 Discussion

Parameter configuration. In Section 5 we have presented a sensitivity analysis with respect to the parameter B, the sampling rate R_s (in case of constant sampling rate), and the maximum number of item references $B + s'_{\max}$ (in case of constant complexity). The choice of these parameters determines the amount of memory that will be used for the approximate MRC computation. Given the memory budget and using some simple characteristics of the trace, such as the number of items or the number of requests (which can also be estimated in an online setting), it is possible to estimate the maximum values for B and R_s.

We have already shown that, due to the specific way in which the final MRC is built, one should adopt a value $B \geq 1/R_s$ to avoid connecting the two MRCs at a point where the sampled one is very imprecise. Additional constraints, due to the specific context in which the MRCs are used, can drive the exact setting. For instance, if the cache needs to be split across different application types, their number and the total amount of storage available further limits B.

Extension to "non-stack" algorithms. The MRC construction technique in case of eviction policies that do not satisfy the inclusion property is different, *i.e.*, one needs to compute the miss ratios for different cache sizes in parallel, and then join the results. Waldspurger *et al.* [36] propose a general method where the miss ratio for cache size C is obtained simulating a cache with size RC with a request trace sampled with rate R. In their experiments they use the same scaling factor R for all the sizes, but our findings suggest that one wants to *differentiate* the sampling rate used, adopting a high (resp. low) sampling rate for small (resp. large) caches. The higher sampling rate for the small caches would be compensated by the smaller sampling rate used at large ones, so overall the memory requirement and the computational complexity could be maintained similar to the case with constant sampling rate.

Heterogeneous item size. Most of the work about MRC consider items with uniform sizes. In contrast, there are different scenarios, such as Web caches, where items have *heterogeneous sizes*. In this case, the MRC should inform the miss rate obtained for a given size of the cache in *bytes*, rather than in number of items. In order to build such a MRC, we need to modify the data structure used to keep track of the items in the cache as explained in Carra *et al.* [8].

Figure 12 (left) shows the exact and approximate MRC

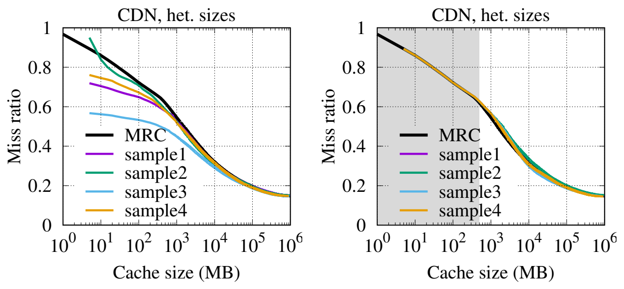

Figure 12: MRCs built from samples with SHARDS$_{\text{adj}}$ (left) and our approach (right—the shaded area corresponds to the portion of the exact MRC).

with SHARDS$_{\text{adj}}$ for the *CDN* trace. Notice that the X-axis now reports the cache size in MB. Figure 12 (right) shows the results with our approach. By combining the exact MRC with the one built from the samples, we are able to build a more accurate MRC using the same amount of memory as used by SHARDS$_{\text{adj}}$. The results are confirmed by the MAEQ, for which we obtain an average value of 0.007—almost one order of magnitude smaller than SHARDS$_{\text{adj}}$'s value (0.052).

7 Conclusions

Sampling has been applied to calculate approximate MRCs with limited computational complexity. The use of such a technique requires a careful design in order to avoid the introduction of biases in the MRC construction. In this work, using a set of experiments and a model of a representative scenario, we studied the impact of popular items on the accuracy of the MRC, and we proposed a new approach that uses exact MRC calculation for small cache sizes while relying on sampling for large ones. The results using different real-world traces show that our solution is able to build approximate MRC with an error per quantile one order of magnitude smaller than state-of-the-art approaches, such as SHARDS$_{\text{adj}}$. As a future work, we plan to study how the parameters of our scheme should be set online depending on the characteristics of the request stream.

Acknowledgments

We would like to thank Pietro Michiardi for the initial discussions on this work, and for the computational resources kindly provided for part of the experiments. We thank the anonymous reviewers and our shepherd, Raju Rangaswami, for their constructive feedback. This work was partially supported by the Italian National Group for Scientific Computation (GNCS-INDAM) and by "Progetto di Eccellenza" of the Computer Science Dept., Univ. of Verona, Italy.

References

[1] Amazon Web Service ElastiCache. https://aws.amazon.com/elasticache/.

[2] Google Cloud Memorystore. https://cloud.google.com/memorystore/.

[3] Microsoft Azure Redis Cache. https://azure.microsoft.com/en-us/services/cache/.

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

[5] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. Adaptsize: Orchestrating the hot object memory cache in a content delivery network. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 483–498, 2017.

[6] Daniel Byrne, Nilufer Onder, and Zhenlin Wang. mPart: miss-ratio curve guided partitioning in key-value stores. In *ACM SIGPLAN Notices*, volume 53, pages 84–95. ACM, 2018.

[7] Damiano Carra, Giovanni Neglia, and Pietro Michiardi. TTL-based Cloud Caches. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 685–693. IEEE, 2019.

[8] Damiano Carra, Giovanni Neglia, and Pietro Michiardi. Elastic provisioning of cloud caches: A cost-aware ttl approach. *IEEE/ACM Transactions on Networking*, 2020.

[9] Calin Cascaval and David A Padua. Estimating cache misses and locality using stack distances. In *Proceedings of the 17th annual international conference on Supercomputing*, pages 150–159. ACM, 2003.

[10] Jichuan Chang and Gurindar S Sohi. Cooperative cache partitioning for chip multiprocessors. In *ACM International Conference on Supercomputing 25th Anniversary Volume*, pages 402–412. ACM, 2014.

[11] Hao Che, Ye Tung, and Zhijun Wang. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 20(7):1305–1314, 2002.

[12] Chen Ding and Yutao Zhong. Predicting whole-program locality through reuse distance analysis. In *ACM Sigplan Notices*, volume 38, pages 245–257. ACM, 2003.

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

[14] Philippe Flajolet, Daniele Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.

[15] Nicaise Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. Performance evaluation of hierarchical TTL-based cache networks. *Computer Networks*, 65:212–231, 2014.

[16] Michele Garetto, Emilio Leonardi, and Valentina Martina. A unified approach to the performance analysis of caching systems. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 1(3):12, 2016.

[17] Xiameng Hu, Xiaolin Wang, Lan Zhou, Yingwei Luo, Chen Ding, and Zhenlin Wang. Kinetic modeling of data eviction in cache. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 351–364, 2016.

[18] Swaroop Kavalanekar, Bruce Worthington, Qi Zhang, and Vishal Sharda. Characterization of storage workload traces from production Windows servers. In *2008 IEEE International Symposium on Workload Characterization*, pages 119–128. IEEE, 2008.

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

[20] WC King. Analysis of paging algorithms. In *Proc. IFIP 1971 Congress, Ljubljana*, pages 485–490. North-Holland, 1972.

[21] Ricardo Koller, Ali José Mashtizadeh, and Raju Rangaswami. Centaur: Host-side ssd caching for storage performance control. In *2015 IEEE International Conference on Autonomic Computing*, pages 51–60. IEEE, 2015.

[22] Ricardo Koller and Raju Rangaswami. I/O deduplication: Utilizing content similarity to improve I/O performance. *ACM Transactions on Storage (TOS)*, 6(3):13, 2010.

[23] Chunghan Lee, Tatsuo Kumano, Tatsuma Matsuki, Hiroshi Endo, Naoto Fukumoto, and Mariko Sugawara. Understanding storage traffic characteristics on enterprise virtual desktop infrastructure. In *Proceedings of*

the 10th ACM International Systems and Storage Conference, SYSTOR '17, pages 13:1–13:11. ACM, 2017.

[24] Tian Luo, Siyuan Ma, Rubao Lee, Xiaodong Zhang, Deng Liu, and Li Zhou. S-cave: Effective ssd caching to improve virtual machine storage performance. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pages 103–112. IEEE, 2013.

[25] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Syst. J.*, 9(2):78–117, June 1970.

[26] Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. Access-time-aware cache algorithms. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 2(4):21, 2017.

[27] Qifan Pu, Haoyuan Li, Matei Zaharia, Ali Ghodsi, and Ion Stoica. Fairride: Near-optimal, fair cache sharing. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 393–406, 2016.

[28] Sundaresan Rajasekaran, Shaohua Duan, Wei Zhang, and Timothy Wood. Multi-cache: Dynamic, efficient partitioning for multi-tier caches in consolidated vm environments. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 182–191. IEEE, 2016.

[29] Trausti Saemundsson, Hjortur Bjornsson, Gregory Chockler, and Ymir Vigfusson. Dynamic performance profiling of cloud caches. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14. ACM, 2014.

[30] Derek L Schuff, Milind Kulkarni, and Vijay S Pai. Accelerating multicore reuse distance analysis with sampling and parallelization. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 53–64. ACM, 2010.

[31] SNIA. SNIA iotta repository block I/O traces. `http://iotta.snia.org/tracetypes/3`. Accessed: July 2019.

[32] Gokul Soundararajan, Jin Chen, Mohamed A Sharaf, and Cristiana Amza. Dynamic partitioning of the cache hierarchy in shared data centers. *Proceedings of the VLDB Endowment*, 1(1):635–646, 2008.

[33] Aditya Sundarrajan, Mingdong Feng, Mangesh Kasbekar, and Ramesh K Sitaraman. 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*, pages 55–67. ACM, 2017.

[34] David K Tam, Reza Azimi, Livio B Soares, and Michael Stumm. RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations. *ACM SIGARCH Computer Architecture News*, 37(1):121–132, 2009.

[35] Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. The impact of memory subsystem resource sharing on datacenter applications. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 283–294. ACM, 2011.

[36] Carl Waldspurger, Trausti Saemundsson, Irfan Ahmad, and Nohhyun Park. Cache modeling and optimization using miniature simulations. In *Proceedings of USENIX ATC*, pages 487–498, 2017.

[37] Carl A Waldspurger, Nohhyun Park, Alexander T Garthwaite, and Irfan Ahmad. Efficient MRC Construction with SHARDS. In *FAST*, pages 95–110, 2015.

[38] Jake Wires, Stephen Ingram, Zachary Drudi, Nicholas JA Harvey, Andrew Warfield, and Coho Data. Characterizing storage workloads with counter stacks. In *OSDI*, pages 335–349, 2014.

[39] Yutao Zhong and Wentao Chang. Sampling-based program locality approximation. In *Proceedings of the 7th international symposium on Memory management*, pages 91–100. ACM, 2008.

[40] Yutao Zhong, Xipeng Shen, and Chen Ding. Program locality analysis using reuse distance. *ACM Trans. Program. Lang. Syst.*, 31(6):1–39, 2009.