Accurate Probabilistic Miss Ratio Curve Approximation for Adaptive Cache Allocation in Block Storage Systems

Rongshang Li[1*], Yingtian Tang[2*], Qiquan Shi[3], Hui Mao[3], Lei Chen[3], Jikun Jin[4], Peng Lu[4], Zhuo Cheng[4]
[1]*The University of Sydney, NSW, Australia*
[2]*University of Electronic Science and Technology of China, Chengdu, China*
[3]*Huawei Noah's Ark Lab, Hongkong, China*
[4]*Huawei Storage Product Line, Chengdu, China*
roli5128@uni.sydney.edu.au, yingtiantd@outlook.com,
{shiqiquan, mao.hui1, lc.leichen, jinjikun, lupeng25, chengzhuo}@huawei.com

***Abstract*—Cache plays an important role in storage systems. With better allocation of cache space to each storage device, total I/O latency can be reduced remarkably. To achieve this goal, we propose an Accurate Probabilistic miss ratio curve approximation for Adaptive Cache allocation (APAC) system. APAC can obtain near-optimal performance for allocating cache space with low overhead. Specifically, with a linear-time probabilistic approximation of reuse distance of all blocks inside each device, APAC can accurately estimate the miss ratio curve (MRC). Furthermore, APAC utilizes the MRCs to obtain the near-optimal configuration of cache allocation by dynamic programming. Experimental results show that APAC achieves higher accuracy in MRC approximation compared to the state-of-the-art methods, leading to higher hit ratio and lower latency of the block storage systems.**

***Index Terms*—Reuse Distance Estimation, Miss Ratio Curve, Cache Allocation**

I. Introduction

Block storage systems have been increasingly important in the cloud services, and provided flexible high-performance Input/Output (I/O) operations for various applications. For example, cloud block storage (CBS) systems have been widely deployed by cloud storage providers like AWS, Google Cloud, etc. In block storage systems, caching is used to reduce latency to improve I/O performance by keeping data that might be accessed in short-term in fast storage medium [1]–[3]. It is worth noticing that even small increases in cache hit ratio have significant impacts on application performances [4], [5].

The design of caching mechanisms heavily rely on the intrinsic data access patterns within the systems, while access patterns are usually complex in real-world applications [6]. Different applications are deployed by multiple users simultaneously on different storage devices, resulting in I/O requests with various characteristics [7]. As a result, the demand of cache varies according to the applications that run on that storage device. Commonly used cache allocation methods such as even-allocation policy (EAP) fail to leverage these complex patterns, and often impose static configurations [8], [9].

Recently, an Online-Model Scheme for dynamic Cache Allocation (OSCA, [10]) was proposed to take a further step. The OSCA system devises a low-cost Miss Ratio Curve (MRC) estimation algorithm for estimating and adjusting the cache demand of different storage devices. The MRC is the curve of the cache miss ratio to the provided cache size, which indicates the benefit per cache size given to a storage device. The decision regarding the cache allocation to a group of storage devices could be directly settled given their MRCs. As a result, MRC estimation is the major component of this cache allocation system. OSCA designs a Re-Access Ratio base Cache Model (RAR-CM) where it is assumed that reaccesses to blocks (smallest units for the caches) are uniformly distributed across the entire I/O history. They then compute the MRCs based on this model for guiding the cache allocation. However, as shown in **Section** IV-B, the estimated MRCs are not accurate enough, which would mislead the allocation algorithm.

In this work, we propose a novel MRC-based cache allocation system, called Accurate Probabilistic miss ratio curve Approximation for adaptive Cache allocation (APAC). With an additional yet minimal assumption about the reaccess patterns (refers to **Assumption** 2 in **Section** III-A), we could construct a much more accurate estimation of MRCs than OSCA with similar computation and memory cost. Our APAC system then leverages the accurate estimation of MRCs and allocate cache for different storage devices. APAC outperforms previous MRC estimation algorithms in terms of estimation accuracy, and also leads to higher system hit ratio than OSCA. In a nutshell, our contributions are twofold:

- An accurate linear-time estimation method for MRCs. It can achieve a theoretically unbiased estimation with our assumptions.
- An adaptive cache allocation system, APAC, which utilizes the accurate estimation of MRCs and outperforms the state-of-the-art (SOTA) methods.

*Equal contribution. Work done as intern at Huawei Noah's Ark Lab.

978-3-9819263-6-1/DATE22/©2022 EDAA

II. Related Work

A. Basic Definitions

Here we have some basic definitions, summarized in **Table** I. Consider the following access sequence:

$$B^1 \;\; C^1 \;\; A^1 \;\; C^2 \;\; D^1 \;\; A^2 \;\; B^2 \;\; C^3 \;\; B^3 \;\; C^4 \;\; E^1 \;\; A^3$$

where L^i indicate the ith access to the block L, for example, A^2 is the 2nd access to the block A. In this access sequence, there are $N = 12$ total accesses and $M = 5$ different block addresses.

Reuse distance (denoted as rd), or stack distance, has been defined by [11]. It is the number of unique block addresses between two accesses to the same block. For example, the unique blocks between A^2 and A^3 contain B, C, E (B, C for two times and E for one time), so the reuse distance between these two accesses is $rd_A^{2,3} = 3$. Similarly, we have $rd_A^{1,2} = 2$, which contain C, D. In general, $rd_L^{i,j}$ is the reuse distance between the ith and jth accesses to the block L.

We also define three more statistics ($io_L^{i,j}$, $rp_L^{i,j}$, $lrp_L^{i,j}$) between two accessses to the same block L, which are related to the computation of reuse distance $rd_L^{i,j}$.

First, $io_L^{i,j}$ simply denotes the number of I/O accesses between L^i and L^j. For instance, $io_A^{2,3} = 5$, which includes B^2, C^3, B^3, C^4, E^1. Second, $rp_L^{i,j}$ denotes the number of reaccesses (globally considered) happened between L^i and L^j. For example, since C^1 and B^1 happened before A^2, and between A^2 and A^3 there are B^2, C^3, B^3, C^4, the global reaccesses of both B and C increase by 2. Consequently, $rp_A^{2,3} = 4$. Finally, $lrp_L^{i,j}$ is similar to $rp_L^{i,j}$ but only considers the local reaccesses. Locally between A^2 and A^3, B and C are both accessed twice, so there are 2 local reaccesses (B^3 and C^4), thus $lrp_A^{2,3} = 2$.

For simplicity, we use a single superscript i to denote the interval from the start of the access sequence to the ith access to that block. For example, $io_A^1 = 2$, which contains the access sequence B^1, C^1.

B. Miss Ratio Curve

1) Description of MRC: Miss ratio is the portion of accesses that fail to hit the cache. Miss Ratio Curve, or MRC, is the curve of the miss ratio to the provided cache size, under the Least-Recently-Used (LRU) policy [12]. Intuitively, if larger cache size is given, then the miss ratio is lower. It is because that the data that needs to be reaccessed is more likely to stay in a larger cache, instead of being replaced by new data.

MRC is crucial to cache allocation strategies, because given the allocated cache size, the miss ratio of that application is reflected quantitatively. MRC under LRU is closely related to reuse distance. Given a cache size C, the $(i+1)$th access to L (or L^{i+1}) should miss only if its current resuse distance $rd_L^{i,i+1}$ is larger than the cache size C. If $rd_L^{i,i+1} \leq C$, the L^i should be still in the cache and have not been replaced by new blocks. Consequently, denote the probability distribution of reuse distances as P_{RD}, then the computation of MRC is simply given by an integration $\text{MRC}(C) = 1 - \sum_{k=0}^{C-1} P_{\text{RD}}(k)$.

TABLE I
Symbol Table. The subscript L denotes a specific block. The superscript i, j denotes range from ith to jth access to L. For simplicity, if the superscript only contains i, it denotes range from the start to the ith access.

N	# total memory accesses
M	# total unique block addresses
L^i	ith access to block L
$rd_L^{i,j}$	# unique blocks
$io_L^{i,j}$	# all accesses
$rp_L^{i,j}$	# globally reaccesses
$lrp_L^{i,j}$	# locally reaccesses

2) Estimation of Reuse Distance: To accurately compute the reuse distance for each pairs of accesses, the algorithm has to keep track of the information of each data block, thus taking $O(N \log M)$ time [11]. This time cost is huge and cannot support real-time cache policy.

The estimation of reuse distance has been well-studied [13]–[15]. Waldspurger et al. [15] proposes a Spatially Hashed Approximate Reuse Distance Sampling (SHARDS) for reducing computation time by sampling the input with a specific sampling threshold T and a modulus P. Denote the accessed block as L. If $hash(L) \;\; mod \;\; P < T$, then this access is collected for reuse distance computation. The estimations are obtained simply by upscaling the computed reuse distance by the sampling rate T/P. The paper also proposed a fixed-size implementation so that the algorithm theoretically runs at constant time. However, with sampling mechanism it is hard to estimate the reuse distance for each access, thus putting on a constraint on the available information. Additionally, in our experiments we observed that SHARDS still required a large portion (around 1%) of data to make relatively accurate estimations, which took long running time.

OSCA [10] proposed a reaccess ratio cache model (RAR-CM). Their method runs at a low cost and estimates per-reference reuse distance. With the definitions, a simple observation would be:

$$rd_L^{i,i+1} = io_L^{i,i+1} - lrp_L^{i,i+1}. \tag{1}$$

OSCA takes $io_L^{i,i+1} \times \frac{rp_L^{i+1}}{io_L^{i+1}}$ as the estimation of $lrp_L^{i,i+1}$. The motivation is that the ratio of local reaccesses to local total accesses $\frac{lrp_L^{i,i+1}}{io_L^{i,i+1}}$ can be estimated by the ratio of global recesses to global total accesses $\frac{rp_L^{i+1}}{io_L^{i+1}}$. Each io_L^{i+1} and rp_L^{i+1} can be calculated in $O(1)$ time by storing the block information in a hash map, so the algorithm takes $O(N)$ time in total. However, empirically the tested error of the lower bound estimation could be rather large, as will be shown later in Section IV.

C. Cache Allocation Policies

Cache allocation policies are the underlying mechanism for the cache server in a storage system. A brief structure of cache allocation in a CBS system is shown in **Figure** 1. Each application that run on the block storage system should ideally require enough cache, so as to reach the highest possible hit ratio for their I/O operations. Nevertheless, the total cache size

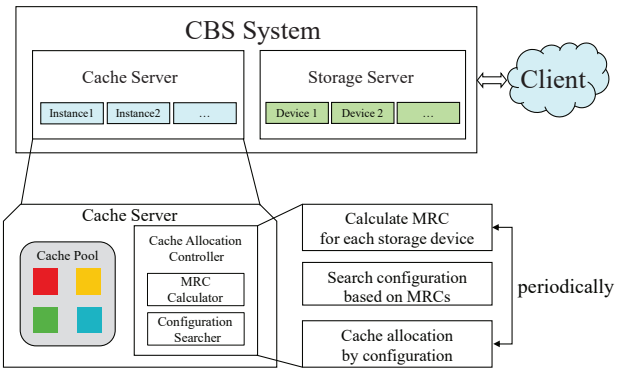

Fig. 1. An simple architecture of CBS system and cache allocation system. Cache sever calculates MRCs, then uses MRCs to search for best configuration and allocates the cache memories in the cache pool to different storage devices periodically.

is limited in the block storage system. Thus, certain trade-off on the hit ratios of different applications exists, in order to maximize the overall system performance.

Current practical systems often use even-allocation policy (EAP). EAP assigns the same pre-determined amount of cache to each storage device, and it is typically used in real-world production systems for its simplicity. This default and static mechanism runs at a very low cost, but fails to capture the dynamics and heterogeneity of the storage system. Some other heuristic methods were developed, including TCM [16] and REF [17]. These methods have low cost, but their capacity of optimization is limited by their overly simplified assumptions.

OSCA proposed to make the allocation dynamically based on quantitative analysis, by periodically computing the MRCs for all storage devices. The trade-off on cache allocation could be settled easily by using the MRCs and assuming them to be stable in the near future. This model-based approach adapts to the changing I/O access patterns and produces more reasonable allocation strategies than static approaches.

III. The Proposed Methods

The proposed cache allocation policy relies on an accurate reuse distance estimation. In the following **Section** III-A, we first present the details of our linear-time reuse distance estimation algorithm. Then, we describe our cache allocation method in **Section** III-B.

A. Linear-Time Reuse Distance Estimation

To reduce the relatively large error of current linear-time reuse distance estimation algorithm and maintain its speed, we propose a novel linear-time estimation method.

As mentioned above in **Equation** 1, reuse distance between two accesses to the same block ($rd_L^{i,i+1}$) could be computed by subtracting the number of accesses in this range ($io_L^{i,i+1}$) by the number of locally reaccesses ($lrp_L^{i,i+1}$), i.e., $rd_L^{i,i+1} = io_L^{i,i+1} - lrp_L^{i,i+1}$. Here we propose an unbiased estimation for $lrp_L^{i,i+1}$. Then, an unbiased estimation of $rd_L^{i,i+1}$ is obtained as $io_L^{i,i+1}$ can be recorded straightforward. The estimation is efficient and takes $O(N)$ time complexity.

Denote the probability distribution of all $io_L^{i,i+1}$ as $p_{io}(k) = P(io_L^{i,i+1} = k)$. $p_{io}(k)$ could be understood as the overall distribution of the number of accesses between two accesses to the same data block. Consider an arbitrary reused block A^{s+1} (we assume that the access A^s exists, which happens before A^{s+1}) happened between reaccesses L^i and $L^i + 1$, we make the assumption that $\forall A, s,\ io_A^{s,s+1} \sim p_{io}(k)$. In other words, we assume that the number of accesses between any pair of reused data blocks obey the same probabilistic distribution.

***Lemma** 1:*

Assume that arbitrary $io_A^{s,s+1}$ follows the overall distribution $p_{io}(k)$, then the probability of A^{s+1} being a locally reaccess between L^i and $L^i + 1$ is $\sum_{k=0}^{io_A^{s+1} - io_L^i} p_{io}(k)$.

***Proof** 1:* If A^{s+1} is a locally reaccess, then A^s must also happen after L^i. Consequently, the probability of $A^{s+1} \in lrp_L^{i,i+1}$ could be calculate as:

$$\begin{aligned} & P(A^{s+1} \in lrp_L^{i,i+1}) \\ &= P(A^s \in L^{i,i+1} \mid A^{s+1} \in L^{i,i+1}) \\ &= P(io_A^s > io_L^i \mid A^{s+1} \in L^{i,i+1}) \\ &= P(io_A^{s+1} - io_A^s < io_A^{s+1} - io_L^i \mid A^{s+1} \in L^{i,i+1}) \\ &= P(io_A^{s,s+1} < io_A^{s+1} - io_L^i \mid A^{s+1} \in L^{i,i+1}) \\ &= \sum_{k=0}^{io_A^{s+1} - io_L^i} p_{io}(k). \end{aligned} \tag{2}$$

We keep an access counter I_{ac} and a reuse counter I_{re}. I_{re} increases by 1 whenever an access happens, while I_{re} increases by 1 when a reaccess happens. We use a hash map M to record information for data blocks. When a data block L is accessed for the ith time, we update $M(L).ac = I_{ac}$ and $M(L).re = I_{re}$. When it is reused, $io_L^{i,i+1}$ is computed by $I_{ac} - M(L).ac$ and $rp_L^{i,i+1}$ is computed by $I_{re} - M(L).re$. We also keep a record of the distribution of all $io_L^{i,i+1}$ so that we estimate $p_{io}(k)$ on the fly. Notice that for any $B \in L^{i,i+1}$, if $B \notin rp_L^{i,i+1}$, then B is never reaccessed and $P(B \in lrp_L^{i,i+1}) = 0$. Then, with **Lemma** 1 we can estimate $lrp_L^{i,i+1}$ as:

$$\begin{aligned} \mathbb{E}\left[lrp_L^{i,i+1}\right] &= \sum_{B \in L^{i,i+1}} P(B \in lrp_L^{i,i+1}) \\ &= \sum_{B \in rp_L^{i,i+1}} P(B \in lrp_L^{i,i+1}) \\ &= \sum_{A^{s+1} \in rp_L^{i,i+1}} P(A^{s+1} \in lrp_L^{i,i+1}) \\ &= \sum_{A^{s+1} \in rp_L^{i,i+1}} \sum_{k=0}^{io_A^{s+1} - io_L^i} p_{io}(k). \end{aligned} \tag{3}$$

If we assume that the reused data blocks $A^{s+1} \in rp_L^{i,i+1}$ are uniformly distributed over time, we can compute $\mathbb{E}\left[rd_L^{i,i+1}\right]$.

***Theorem** 1:* Assume that:

***Assumption** 1:* Reused blocks are uniformly distributed over time.

***Assumption** 2:* The distribution of the number of blocks between two accesses to the same block $p_{io}(k)$, is stationary. Then, we have:

$$\mathbb{E}\left[rd_L^{i,i+1}\right] = io_L^{i,i+1} - \frac{rp_L^{i,i+1}}{io_L^{i,i+1}} \sum_{k=0}^{io_L^{i,i+1}} (io_L^{i,i+1} - k + 1)p_{io}(k).$$

***Proof** 2:* Consider an arbitrary reused block A^{s+1} between L^i and L^i+1. According to the assumption, we have $io_A^{s,s+1} \sim p_{io}(k)$. With **Lemma** 1 and **Equation** 3, we know that:

$$\mathbb{E}\left[lrp_L^{i,i+1}\right] = \sum_{A^{s+1} \in rp_L^{i,i+1}} \sum_{k=0}^{io_A^{s+1} - io_L^i} p_{io}(k).$$

Since arbitrary $A^{s+1} \in rp_L^{i,i+1}$ distributes evenly, $io_A^{s+1} - io_L^i$ also takes a discrete uniform distribution between L^i and L^i+1. Denote $X = io_A^{s+1} - io_L^i$, we have $X \sim \mathcal{U}\{0, io_L^{i,i+1}\}$, then:

$$\begin{aligned}
\mathbb{E}\left[lrp_L^{i,i+1}\right] &= \sum_{A^{s+1} \in rp_L^{i,i+1}} \mathbb{E}_{X \sim \mathcal{U}\{0, io_L^{i,i+1}\}} \left[\sum_{k=0}^{X} p_{io}(k)\right] \\
&= \frac{rp_L^{i,i+1}}{io_L^{i,i+1}} \sum_{X=0}^{io_L^{i,i+1}} \sum_{k=0}^{X} p_{io}(k) \\
&= \frac{rp_L^{i,i+1}}{io_L^{i,i+1}} \sum_{k=0}^{io_L^{i,i+1}} \sum_{X=k}^{io_L^{i,i+1}} p_{io}(k) \\
&= \frac{rp_L^{i,i+1}}{io_L^{i,i+1}} \sum_{k=0}^{io_L^{i,i+1}} (io_L^{i,i+1} - k + 1)p_{io}(k).
\end{aligned} \tag{4}$$

Therefore:

$$\begin{aligned}
\mathbb{E}\left[rd_L^{i,i+1}\right] &= \mathbb{E}\left[io_L^{i,i+1} - lrp_L^{i,i+1}\right] \\
&= io_L^{i,i+1} - \mathbb{E}\left[lrp_L^{i,i+1}\right] \\
&= io_L^{i,i+1} - \frac{rp_L^{i,i+1}}{io_L^{i,i+1}} \sum_{k=0}^{io_L^{i,i+1}} (io_L^{i,i+1} - k + 1)p_{io}(k).
\end{aligned} \tag{5}$$

With **Theorem** 1, we can compute $\mathbb{E}\left[rd_L^{i,i+1}\right]$ using only $io_L^{i,i+1}$, $rp_L^{i,i+1}$ and the distribution $p_{io}(k)$. We keep a distribution $\hat{p}_{io}(k)$ as the estimation of $p_{io}(k)$, so that we have an unbiased estimation for each reuse distance as:

$$io_L^{i,i+1} - \frac{rp_L^{i,i+1}}{io_L^{i,i+1}} \sum_{k=0}^{io_L^{i,i+1}} (io_L^{i,i+1} - k + 1)\hat{p}_{io}(k).$$

Keeping $\hat{p}_{io}(k)$ takes $O(1)$ time for each new I/O request. As mentioned before, the calculation time $rd_L^{i,i+1}$ and $io_L^{i,i+1}$ is $O(1)$, so the computation of any $rd_L^{i,i+1}$ is $O(1)$. Overall, the estimation algorithm runs at $O(N)$ time.

B. Cache Allocation based on MRCs

Based on accurately approximated MRCs, we further propose a dynamic-programming-based cache allocation system.

We define total cache size as C_{total}, and cache size allocated to K different storage devices is denoted as $C_i, i = 1, 2, ..., K$, respectively. The hit ratio of the allocated cache for the ith storage device is defined as HR_i. Also, define N_i as the total number of accesses for the ith device. Then, the optimization problem is formulated as:

$$\max_{C_i} \sum_{i=1}^{K} \text{HR}_i N_i \tag{6}$$

$$s.t. \sum_{i=1}^{K} C_i = C_{\text{total}} \tag{7}$$

The MRC of the ith storage device reflects the relationship between the miss ratio (*i.e.*, $1 - \text{HR}_i$) and its given cache sizes, so it encodes all the information that we need for cache allocation optimization. We calculate the MRCs of each storage device, and use dynamic programming to search for the best configuration of cache sizes based on MRCs, and then adjust the cache allocation configuration. This procedure is done periodically with an interval T during the whole I/O process.

IV. Experiments

A. Experimental Settings

We compare the proposed APAC with other methods on one-day-long data traces collected from a commercial CBS system, where applications run on multiple storage devices. The CBS system consists of 12 storage devices, indexed by Logical Unit Number 1 to 12 (LUN1 to LUN12). The detailed information are shown in **Table** II.

TABLE II
TRACE SOURCES AND NUMBER OF TOTAL MEMORY ACCESSES N AND TOTAL UNIQUE BLOCK ADDRESSES M.

name	N	M
LUN1	31110176	286083
LUN2	27437054	287345
LUN3	31217096	284025
LUN4	51946923	20636130
LUN5	54446309	21132290
LUN6	55505126	21159307
LUN7	54430947	21160552
LUN8	54334504	21168283
LUN9	54898915	21227572
LUN10	60726889	21220725
LUN11	55429555	21142972
LUN12	28595133	262565

We compare APAC with a classical method **SHARDS** [18] and a SOTA approach **OSCA** [10] on MRC approximation and cache allocation. We use SHARDS with 0.01 sample rate to balance accuracy and efficiency. With this setting, the actual running times of the three algorithms are approximately the same. The MRC calculation period of cache allocation is set as $T = 1$ hour. The total cache size are 12GB and initial cache size of each LUN are equally allocated to be 1GB. We set the range of cache size of each LUN to be between 32MB and 4GB. The finest grain of adjustment is set to be 32MB.

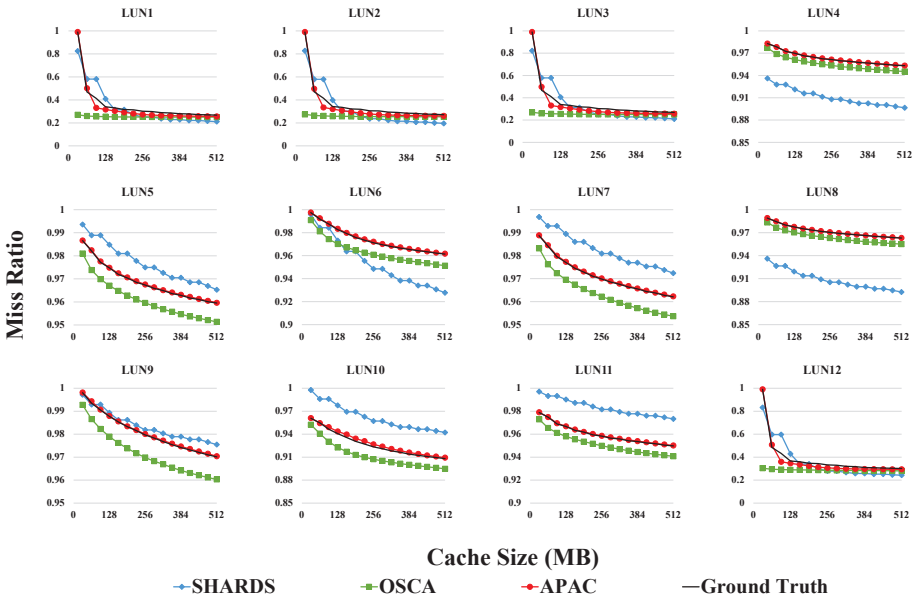

Fig. 2. MRCs of all 12 LUNs. We only show cache size from 64MB to 512MB. After the 512MB the trends of miss ratio curves keep the same, which are not informative. MRCs of APAC are the most closest to the ground truth in most cases.

TABLE III
THE MAES OF MRC APPROXIMATION. THE BEST ONES ARE IN **BOLD**.

name	SHARDS	OSCA	APAC
LUN1	6.31E-02	3.52E-02	**4.98E-03**
LUN2	8.34E-02	3.46E-02	**4.93E-03**
LUN3	6.51E-02	3.66E-02	**5.07E-03**
LUN4	6.00E-02	7.31E-03	**5.79E-04**
LUN5	2.19E-03	7.70E-03	**5.42E-04**
LUN6	6.20E-02	1.07E-02	**5.47E-04**
LUN7	6.97E-03	7.35E-03	**5.08E-04**
LUN8	7.81E-02	7.13E-03	**4.68E-04**
LUN9	1.01E-02	1.00E-02	**5.02E-04**
LUN10	2.67E-02	1.48E-02	**1.05E-03**
LUN11	2.07E-02	8.32E-03	**6.40E-04**
LUN12	6.40E-02	6.64E-02	**5.25E-03**
Average	4.52E-02	2.05E-02	**2.09E-03**

B. Results of MRC Approximation

We compare the proposed MRC approximation with SHARDS (0.01 sampling rate) and OSCA. We compute the **exact** miss ratio (ground truth) and the estimated ones by compared methods on cache sizes from 32MB to 4GB every 32MB. The results of MRC approximation are reported in **Table** III, where Mean Absolute Error (MAE) over miss ratios on all tested cache sizes are used to evaluate the MRCs.

The error of SHARDS is 21.6 times larger than APAC, and OSCA is 9.82 times larger. APAC has the lowest MAE on all experiments. The comparison of MRCs of all 12 LUNs is shown in **Figure** 2.

By observing the access sequences, a typical case in reuse distance estimation that OSCA performs weakly is found. Consider the following trace:

$$A^1,\ B^1,\ C^1,\ D^1,\ A^2,\ B^2,\ C^2,\ D^2,\ A^3,\ B^3,\ C^3,\ D^3$$

OSCA estimate $lrp_D^{2,3} = io_4^{2,3} \times \frac{rp_D^3}{io_D^3} = 3 \times \frac{7}{12} \approx 1.75$. And APAC estimate $lrp_D^{2,3} = 0$ because $p_{io}(k) = 0,\ \forall k < 4$, which is accurate. OSCA accumulates errors with this common pattern of access sequences, which leads to errors in MRC approximations.

Also, it should be noticed that SHARDS has analytically $O(1)$ computation time, but the accuracy of its results decreases as the sampling rate decreases. As a result, we control the actual computation time in our experiments. Our method achieves much more accurate MRC estimations than other methods, with approximately the same actual computation time.

C. Results of Cache Allocation

To prove the effectiveness of cache allocation, we compared all methods to EAP and optimal configuration. The optimal configuration is obtained offline by calculating accurate MRCs. To show the effectiveness of a method, we report its Improvement Coverage, **IC**:

$$\mathrm{IC}_{method} = \frac{\mathrm{HR}_{optimal} - \mathrm{HR}_{method}}{\mathrm{HR}_{optimal} - \mathrm{HR}_{EAP}} \tag{8}$$

Here HR_{method} means the overall hit ratio of all LUNs. This statistic shows how much a method covers the difference of total hit ratio between EAP and the optimal configuration. The results are shown in **Figure** 3.

Overall, APAC covers 66.8% of theoretically optimal improvement. APAC remains higher hit ratio compared to OSCA

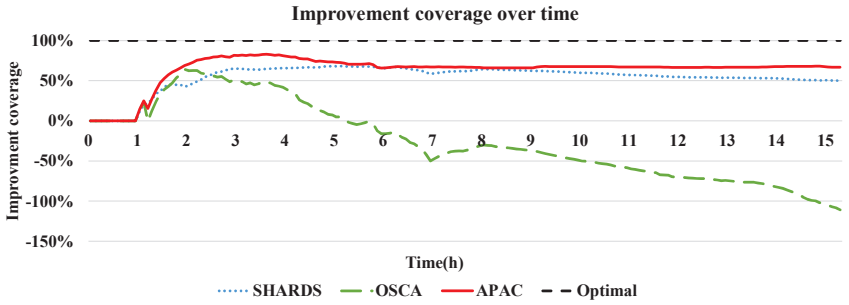

Fig. 3. ICs of all methods over 15 hours, after which the trends of ICs keep the same.

and SHARDS during the I/O accesses except the first hour, as no allocation is performed at that time. Due to the inaccurate estimation of MRCs by OSCA, it even performs worse than EAP since the 6th hour. SHARDS performs the second with 49.7% coverage. SHARDS shows stable performance over time, as it estimates MRCs more and more accurately with the number of I/O access grows.

It is worth noting that although SHARDS has higher MAE in MRC approximation, it reflects the trend of MRC better than OSCA. Consequently, the cache allocation performance is better than OSCA. Our estimations have more accurate trends, so that APAC still outperforms SHARDS. Also, as shown in **Figure** 2, the MRC estimation of OSCA has larger error when cache sizes are small, which certainly leads to unsatisfactory cache allocation configuration.

V. Conclusion

In this work, we propose a dynamic cache allocation method based on accurate approximation of miss ratio curve. The approximation algorithm is a linear-time algorithm, which shows a large improvement over the existing SOTA linear-time estimation in terms of MAE.

Since our APAC can obtain more accurate estimation of MRCs, the allocation strategy based on MRCs is also more efficient. Experiments show that the proposed cache allocation method has consistently higher hit ratio on block traces than SHARDS and the SOTA method OSCA.

One limitation the proposed method is that the recording of reuse distance information and last access information consumes $O(M)$ space, which is similar to OSCA. In future work, we will improve it by designing better data structures.

References

[1] A. J. Smith, "Cache memories," *ACM Computing Surveys (CSUR)*, vol. 14, p. 473–530, Sept. 1982.

[2] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic, "Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite," in *Proceedings of the 42nd Annual Southeast Regional Conference*, ACM-SE 42, p. 267–272, Association for Computing Machinery, 2004.

[3] C.-Y. Chang and M.-S. Chen, "A new cache replacement algorithm for the integration of web caching and prefectching," in *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, p. 632–634, Association for Computing Machinery, 2002.

[4] J. Wang, B. Berg, D. S. Berger, and S. Sen, "Maximizing page-level cache hit ratios in largeweb services," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 2, pp. 91–92, 2019.

[5] A. Jaamoum, T. Hiscock, and G. Di Natale, "Scramble cache: An efficient cache architecture for randomized set permutation," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE 2021)*, pp. 621–626, IEEE, 2021.

[6] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning memory access patterns," in *International Conference on Machine Learning*, pp. 1919–1928, PMLR, 2018.

[7] C. Chakraborttii and H. Litz, "Learning i/o access patterns to improve prefetching in ssds," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 427–443, Springer, 2020.

[8] A. Janapsatya, A. Ignjatović, J. Peddersen, and S. Parameswaran, "Dueling CLOCK: Adaptive cache replacement policy based on the clock algorithm," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 920–925, IEEE, 2010.

[9] Y. Kim, A. More, E. Shriver, and T. Rosing, "Application performance prediction and optimization under cache allocation technology," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE 2019)*, pp. 1285–1288, IEEE, 2019.

[10] Y. Zhang, P. Huang, K. Zhou, H. Wang, J. Hu, Y. Ji, and B. Cheng, "OSCA: An online-model based cache allocation scheme in cloud block storage systems," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 785–798, 2020.

[11] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems journal*, vol. 9, no. 2, pp. 78–117, 1970.

[12] G. Yadgar, M. Factor, and A. Schuster, "Karma: Know-it-all replacement for a multilevel cache.," in *Fast*, vol. 7, pp. 25–25, 2007.

[13] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm, "RapidMRC: approximating l2 miss rate curves on commodity systems for online optimizations," *ACM Sigplan Notices*, vol. 44, no. 3, pp. 121–132, 2009.

[14] E. Teran, Z. Wang, and D. A. Jiménez, "Perceptron learning for reuse prediction," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, IEEE, 2016.

[15] C. A. Waldspurger, N. Park, A. Garthwaite, and I. Ahmad, "Efficient mrc construction with shards," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pp. 95–110, 2015.

[16] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 65–76, IEEE, 2010.

[17] S. M. Zahedi and B. C. Lee, "Ref: Resource elasticity fairness with sharing incentives for multiprocessors," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 145–160, 2014.

[18] F. Olken, "Efficient methods for calculating the success function of fixed space replacement policies," *Performance Evaluation*, vol. 3, no. 2, pp. 153–154, 1983.