usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

*e*MRC: Efficient Miss Ratio Approximation for Multi-Tier Caching

Zhang Liu, *University of Colorado Boulder;* Hee Won Lee, *Samsung Electronics;* Yu Xiang, *AT&T Labs Research;* Dirk Grunwald and Sangtae Ha, *University of Colorado Boulder*

https://www.usenix.org/conference/fast21/presentation/liu

This paper is included in the Proceedings of the 19th USENIX Conference on File and Storage Technologies.

February 23–25, 2021

978-1-939133-20-5

*e*MRC: Efficient Miss Ratio Approximation for Multi-Tier Caching

Zhang Liu
University of Colorado Boulder

Hee Won Lee[*]
Samsung Electronics

Yu Xiang
AT&T Labs Research

Dirk Grunwald
University of Colorado Boulder

Sangtae Ha
University of Colorado Boulder

Abstract

Many storage cache allocation methods use the *miss ratio curve* (MRC) to improve cache efficiency. However, they have focused only on single-tier cache architectures and require the whole MRC as input for cache management, while modern datacenters embrace hierarchical caching architectures to maximize resource utilization. Generating the MRC for multi-tier caches – we call it the *miss ratio function* – is far more challenging due to different eviction policies and capacities in each cache tier. We introduce *e*MRC, a multi-dimensional miss ratio approximation technique, to enable efficient MRC generation for multi-tier caching. Our approach uses a novel multi-dimensional performance cliff removal method and convex hull approximation technique to efficiently generate a multi-dimensional MRC without cliffs using a small number of sampling points. To demonstrate the benefits of *e*MRC, we designed ORCA, a multi-tier cache management framework that orchestrates caches residing in different hierarchies through *e*MRC and provides efficient multi-tier cache configurations to cloud tenants with diverse service level objectives. We evaluate the performance of our *e*MRC approximation technique and ORCA with real-world datacenter traces.

1 Introduction

Caching is often provisioned on multiple tiers in cloud storage systems. When client applications running in virtual machines (VMs) generate block IOs to remote storage media, the requests pass through a series of intermediate layers, such as user-space libraries, hypervisors, and actual storage nodes. Each layer presents a caching opportunity, and thus most cloud providers adopt multi-tier caching architectures to maximize the utilization of scattered resources in the system [11].

Multi-tier caching necessitates an effective, low overhead cache management scheme as arbitrary cache configurations for multiple tiers may not benefit tenants due to their side effects such as double caching effects [27]. Configuring caches for tenants with diverse service level objectives (SLOs) requires efficient and accurate cache performance analysis for *each* tier of the cache.

It is well known that effective cache management requires a good understanding of IO workload characteristics. Without understanding the workload, systems usually rely on trial-and-error tuning methods that can be very inefficient. The *miss ratio curve* (MRC) is a useful tool to capture workload characteristics and tune system behavior. The MRC represents the relationship between cache size and the corresponding cache miss ratio. Assuming workloads are relatively stable over time, the MRC derived from observed IO traces is known to work effectively for single-tier caches [13].

The general workflow of utilizing miss ratio information for cache management is as follows: 1) the IO streams of tenants are analyzed, and a miss ratio function is generated to represent the miss ratio for any given cache configurations for each tenant; 2) based on tenants' SLOs, a cache management framework allocates the optimal cache size for each tenant. In this workflow, the performance primarily depends upon how quickly it estimates the miss ratio of a particular sized cache for each tenant, given that the cache management framework needs to handle multiple tenants, each with a different IO pattern in a datacenter.

Evaluating the miss ratios for all possible cache sizes is a very time-consuming task. SHARDS [25] and Miniature Simulation [24] allow rapid MRC construction with reduced overhead, and other efficient techniques have also been proposed [7, 10, 26].

These previous techniques are either specific to single-tier caches or are inefficient for multi-tier caches. As shown in Figure 1, there are *performance cliffs* on the miss ratio surface where miss ratios change dramatically with small changes in cache size. The miss ratio surface is a multi-dimensional function representing the relationship between cache size in each tier of the cache hierarchy and the corresponding cache miss ratios. While evaluating cache configurations using multi-tier

[*]Hee Won Lee conducted this project when he was at AT&T Labs Research.

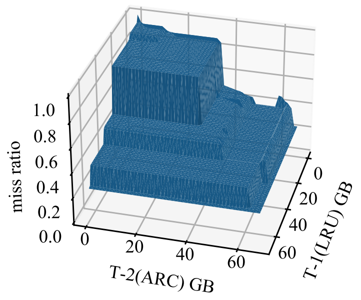


Figure 1: Miss ratio surface of MSR `web_2` trace.

cache simulators such as *PyMimircache* [8, 30] is possible, and MRC cliff removal techniques [4, 6] are available for single-tier caching, to the best of our knowledge, there are no known algorithms that can remove performance cliffs in miss ratio functions for multi-tier caches, or that can generate continuous miss ratio functions efficiently.

In this paper, we present our *e*MRC approach to achieving the *efficient, rapid* generation of multi-dimensional miss ratio functions for multi-tier caching. *e*MRC is enabled by our *convex hull algorithm* and *cache partitioning algorithm*. The convex hull algorithm efficiently divides the whole multi-dimensional space into multiple regions without full knowledge of the multi-dimensional MRC. This convex hull algorithm only requires a small number of sampling points that significantly reduces the computation time. The cache partitioning algorithm then removes performance cliffs for multi-tier caching within any region bounded by data points with known miss ratio values; the resulting miss ratio function is always equal to or better than the original miss ratio function without *e*MRC's cache partitioning.

We also developed the ORCA cache orchestration framework for multi-tenant, multi-tier caching that leverages *e*MRC. We evaluate both *e*MRC and ORCA using real-world IO traces released by Microsoft [1].

Our key contributions can be summarized as follows:

- We are the first to provide an algorithm, *e*MRC, that removes performance cliffs in multi-dimensional miss ratio functions for multi-tier caching.
- For *e*MRC, we develop a technique called *Convex Hull Approximation*. In terms of the number of sampling points required, it speeds up MRC generation by 14 times for two-tier caching and 4,527 times for four-tier caching.
- ORCA uses *e*MRC to efficiently provide effective cache configurations for tenants with diverse SLOs by selecting optimal cache sizes and replacement policies.
- We evaluate our *e*MRC approximation method with real datacenter traces and validate that ORCA provides effective multi-tier cache configurations and boosts the performance for various types of mixed workloads.

2 Background

In this section, we will review three concepts upon which our *e*MRC approximation is built.

2.1 Sampling and Statistical Similarity

Statistical similarity means a smaller sampled IO trace can be used to estimate the miss ratio of the original IO trace. Kessler et al. [12] defined the "*10% sampling goal*" for *statistical similarity*: "A method meets the 10% sampling goal if, at least 90% of the time, it estimates the trace's true misses per instruction with $\leq$ 10% relative error using $\leq$ 10% of the trace", and showed that constant-bits sampling satisfies such a goal. Constant-bits sampling means selecting the IO entries that have the same value in some address bits.

Spatial sampling is a recently proposed technique to sample IO traces with *statistical similarity*. It means taking the hash values of IO addresses A and then using modulus P and a threshold T on the hash value to determine what fraction of IO operations to sample, $hash(A)\, mod\, P < T$. The resulting sampling rate is $R = T/P$.

Statistical similarity has been used in various MRC based studies, e.g., accelerating trace-driven simulations [24, 25] and altering cache behaviors by using shadow partitions [4].

2.2 Talus Cache Partitioning

Talus [4] is a recently proposed cache partitioning algorithm that can remove MRC performance cliffs for single-tier caching. Talus utilizes *statistical similarity* of spatial sampled IO streams. Assuming the original miss ratio is $m(x)$ for cache size x, passing a fraction[1] ρ of an original IO stream into a proportionally smaller cache partition of size $x' = \rho x$ will result in a statistically similar miss ratio:

$$m'(x') = m(\frac{x'}{\rho}), \text{ where } 0 \leq \rho \leq 1 \qquad (1)$$

The Talus algorithm divides a single cache into two partitions so that the cache has a miss ratio that interpolates between two points on the MRC of an original unpartitioned cache. To demonstrate the Talus algorithm, we use an MRC example shown in Figure 2. A performance cliff exists between cache sizes α and β, which are two convex hull points. If their cache miss ratios are $m(\alpha)$ and $m(\beta)$, respectively, Talus partitioning technique can provide cache miss ratio $m(x)$ whose value is between $m(\alpha)$ and $m(\beta)$. The small-dotted MRC is the miss ratio after applying the Talus algorithm. The Talus MRC curve is convex and lower than the MRC of the unpartitioned cache.

Now let us assume we want to configure cache partitions so that the overall miss ratio is lowered from 66% to 53% with a cache size of $x = 0.99$ GB that falls between $\alpha = 0.23$ GB and

[1]The fraction means the sampling rate of spatial sampling.

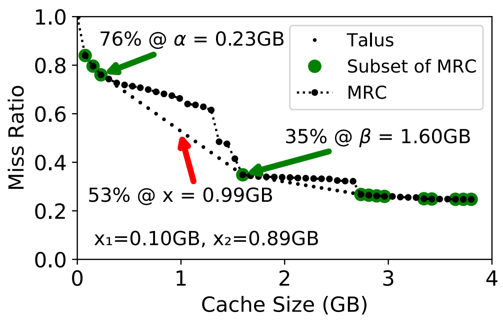


Figure 2: Talus miss ratio cliff removal.

$\beta = 1.60$ GB. We can determine the miss ratio using the Talus equation: $m_{\text{talus}}(x) = \frac{\beta-x}{\beta-\alpha}m(\alpha) + \frac{x-\alpha}{\beta-\alpha}m(\beta)$. Hence, for the example of Figure 2, we can obtain:

$$\begin{aligned} m_{\text{talus}}(0.99) &= \frac{1.60-0.99}{1.60-0.23}m(0.23) + \frac{0.99-0.23}{1.60-0.23}m(1.60) \\ &= 0.45 \times 0.76 + 0.55 \times 0.35 \\ &= 0.53. \end{aligned}$$

To achieve this miss ratio, Talus method allocates two cache partitions of size x_1 and x_2 such that $x = x_1 + x_2$. A fraction ρ of the IOs will pass through the first cache partition of size $x_1 = \rho\alpha$ where $\rho = \frac{\beta-x}{\beta-\alpha}$, and the remaining fraction $(1-\rho)$ of the IOs will pass through the second cache partition of size $x_2 = x - x_1$.

The resulting miss ratio for the first partition is $m_1(x_1)$, which equals $m(x_1/\rho)$ by Equation 1. And the miss ratio for the second partition, $m_2(x_2)$, equals $m(\frac{x-x_1}{1-\rho})$.

The resulting Talus miss ratio is:

$$\begin{aligned} m_{\text{talus}}(x) &= \rho m_1(x_1) + (1-\rho)m_2(x_2) \\ &= \rho m(\frac{x_1}{\rho}) + (1-\rho)m(\frac{x-x_1}{1-\rho}) \\ &= \frac{\beta-x}{\beta-\alpha}m(\alpha) + \frac{x-\alpha}{\beta-\alpha}m(\beta) \end{aligned}$$

In sum, we want to have the *Talus cache* behave like a mix of cache sizes α and β. The sample point $x = 0.99$ GB results in $\rho = \frac{\beta-x}{\beta-\alpha} = (1.60-0.99)/(1.60-0.23) = 0.45$. Thus, 45% of the IOs will use a cache partition of $x_1 = \rho\alpha = 0.45 \times 0.23 = 0.10$ GB and they will have a miss ratio of $m(x_1/\rho) = m(\alpha) = 0.76$. The remaining 55% of the references will use a cache size of $x - x_1 = 0.99 - 0.10 = 0.89$ GB with a miss ratio of $m(\frac{x-x_1}{1-\rho}) = m(\beta) = 0.35$. Across all the IOs, the miss ratio will be $0.45 \times 0.76 + 0.55 \times 0.35 = 0.53$.

As this process is repeated for different points x between α and β, the resulting miss ratio curve will be a linear interpolation between the miss ratios at α and β.

2.3 Rapid MRC Generation

Existing research has mainly focused on using a subset of the original trace to accelerate MRC generation. Some of them focused on stack-based eviction policies, while others can apply to all eviction policies.

Stack-based cache eviction policies have the *inclusion property*, meaning that for the same input IO, the cache of a larger size always contains all cached items in the cache of a smaller size. Simple cache eviction policies such as LRU and LFU are stack-based policies, but more complex eviction policies such as ARC [17] or MQ [32] are not stack-based. SHARDS [25] is based on the stack distance algorithm of Mattson *et al.* [16], which generates the MRC with a single pass of the workload trace. With the help of spatial downsampling, SHARDS can significantly reduce the computation time and memory footprint for long traces while generating relatively accurate MRCs.

Miniature-Simulation [24] is another recent advance in MRC generation for both stack and non-stack based eviction policies. It shows that heavy spatial downsampling can be used to generate a relatively accurate MRC by running a separate scaled-down simulation for each cache size.

3 *e*MRC: Miss Ratio Approximation for Multi-Tier Caching

We will begin with an illustrative example to intuitively show how *e*MRC works. After that, we will demonstrate that spatial sampling also works for more than one cache tier in Section 3.2. We will show how *e*MRC can remove multi-dimensional performance cliffs in a cliff region bounded by data points with known miss ratios in Section 3.3. We explain how to partition the whole multi-dimensional MRC into multiple regions efficiently to apply *e*MRC for the entire space in Section 3.4. To simplify explanation, we first address two-tier caching and then generalize our algorithm for three or more tier caching in Section 3.5.

3.1 Illustrative Example for *e*MRC

In single-tier caching, Talus achieves a convex MRC by partitioning the cache into two partitions and letting the miss ratio of each partition be related to one of the boundary point miss ratios. In two-tier caching, as the example in Figure 3 illustrates, there are four boundary points. *e*MRC partitioned the last tier cache (2nd tier) into four partitions, with each partition related to one of the boundary point miss ratios.

In the particular example in Figure 3, there is a cliff region in the MRC of a workload using two-tier caching. The four boundary point miss ratios are: $M(2,2) = 0.9$, $M(2,6) = 0.5$, $M(6,2) = 0.5$ and $M(6,6) = 0.5$. If we use the two tiers of caches without partitioning, the miss ratio for 3GB of tier-1 cache and 3GB of tier-2 cache is $M(3,3) = 0.9$.

*e*MRC ensures that if we partition the tier-1 cache into two partitions of [1.5GB, 1.5GB] and the tier-2 cache into four partitions of [1.125GB, 1.125GB, 0.375GB, 0.375GB], and then divide the IOs into the different partitions using

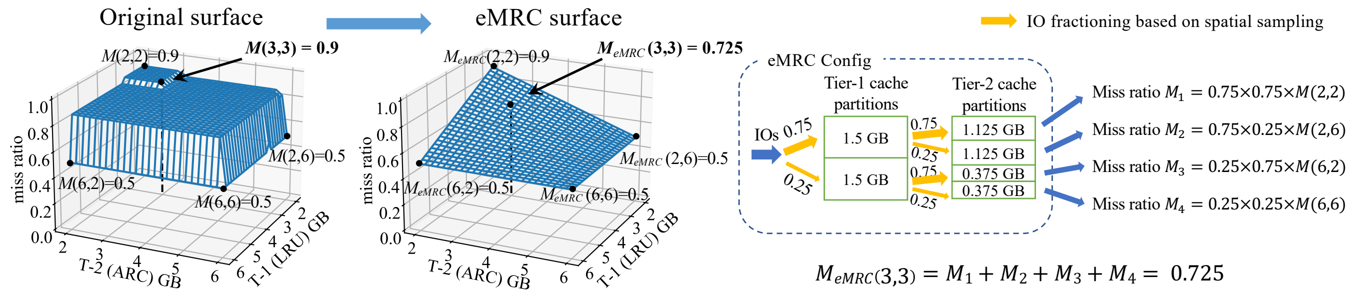


Figure 3: Example of how *e*MRC removes performance cliffs.

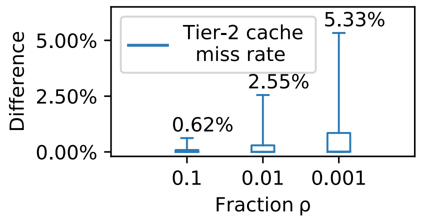


Figure 4: Relative errors in approximating cache miss ratios by sampling at rate ρ across all workloads. The error bars represent the 10th & 90th percentile values.

spatial sampling based hash functions, we have the following relation:

$$\begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} = \begin{bmatrix} 0.75\times0.75 & 0 & 0 & 0 \\ 0 & 0.75\times0.25 & 0 & 0 \\ 0 & 0 & 0.25\times0.75 & 0 \\ 0 & 0 & 0 & 0.25\times0.25 \end{bmatrix} \begin{bmatrix} M(2,2) \\ M(2,6) \\ M(6,2) \\ M(6,6) \end{bmatrix}$$

This means each of the miss ratios after the four tier-2 cache partitions is only related to one of the boundary point miss ratios. For example, $M_1 = 0.75 \times 0.75 \times M(2,2)$, meaning no matter how the miss ratios change at the boundary points, M_1 is only related to the miss ratio at $M(2,2)$ on the original surface.

The resulting miss ratio using *e*MRC for 3GB tier-1 cache and 3GB tier-2 cache is now $M_{eMRC}(3,3) = 0.725$ compared to the original miss ratio at $M(3,3) = 0.9$. If we apply *e*MRC to all the cache configurations in the bounded region, we can obtain a convex *e*MRC surface as shown in Figure 3.

In Section 3.3, we show how *e*MRC determines the partition parameters for cliff removals in detail.

3.2 Spatial Sampling and Statistical Similarity for Multi-tier Caching

To construct the *e*MRC miss ratio function, we need to partition caches in each tier repeatedly; this requires spatial sampling to work in multiple cache tiers. Here we are empirically validating that a spatially sampled IO stream can also be used to estimate the miss ratio at the tier-2 cache, extending Kessler's assumption concerning calculating the tier-1 cache miss ratio accurately.

We validated this using the MSR IO traces [1]. Figure 4 shows the approximation error in miss ratio at the tier-2 cache when sampling at different sampling rates ρ before the tier-1 cache, with *no* sampling before the tier-2 cache. We generated the result shown in Figure 4 as follows. For each trace, we evaluated the first 10M entries with 2,601 different tier-1 and tier-2 cache configurations. The tier-1 and tier-2 caches can take 51 different cache sizes incrementing from 0 to *max_cache_size*, based on the unique item count in the first 10M entries of each trace. We calculate the relative difference on the tier-2 cache miss ratio using sampled vs. unsampled traces. We show a bar plot in Figure 4 with more than 70K data points for each spatial sampling rate (i.e., ρ = 0.1, 0.01, and 0.001). The result shows sampling at a 10% rate causes at most 0.62% relative difference across 90% of the data points. Kessler's sampling goal is met even when approximating the miss ratio using only 0.1% of the original trace.

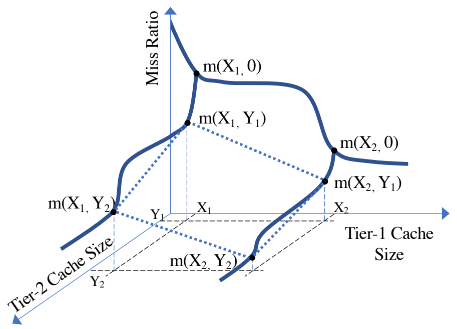


Figure 5: Illustration of a performance cliff region.

This demonstrates that spatial sampling can be used to estimate the tier-2 cache miss ratio. This property supports the construction of the *e*MRC convex miss ratio function and enables rapid cache simulation to explore key miss ratio points in the multi-dimensional miss ratio function.

3.3 Partitioning Scheme for Cliff Removal

This section demonstrates how *e*MRC removes performance cliffs for two-tier caching systems, which can be easily extended to work with multi-tier caches.

Figure 5 illustrates a miss ratio surface in a two-tier caching system. We assume that the *discrete* miss ratio surface $m(x,y)$ is generated from a single IO stream and there are four miss ratio data points, i.e., $m(X_1,Y_1)$, $m(X_1,Y_2)$, $m(X_2,Y_1)$ and $m(X_2,Y_2)$, which surround a performance cliff region. How can we remove the performance cliff using only the four given

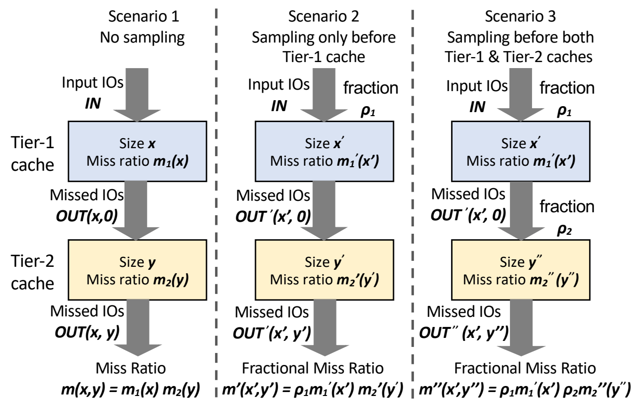


Figure 6: IO sampling scenarios in two-tier cache system.

data points? The Talus algorithm only works for a single-tier cache because it needs two data points using the same input IO stream. For example, in Figure 5, Talus cannot apply between $m(X_1,Y_1)$ and $m(X_2,Y_1)$ because these two data points use different input IO streams; one is the missed IOs when the tier-1 cache size is X_1, and the other is when the tier-1 cache size is X_2.

The *e*MRC algorithm removes performance cliffs for two (or more) tier cache systems using the partitioning scheme outlined in Figure 8.

We present our assumption and theorems that lead to the construction of a miss ratio surface $m_{eMRC}(x,y)$ without performance cliffs. Figure 6 shows three scenarios that cover all possible IO sampling cases in two-tier cache systems. Scenarios 1 and 2 are used in the proof of Theorem 1. All three scenarios are used in the proof of Theorem 2.

- **Scenario 1: No sampling.** Let $m_1(x)$ and $m_2(y)$ denote the cache miss ratios observed at the tier-1 and tier-2 caches, respectively. $OUT(x,0)$ is the missed IO stream after the tier-1 cache, and $OUT(x,y)$ is the one after the tier-2 cache. Then the miss ratio after the tier-1 cache $m(x,0)$ equals $m_1(x)$, and the miss ratio after the tier-2 cache $m(x,y)$ equals $m_1(x)m_2(y)$.
- **Scenario 2: Sampling only before the tier-1 cache.** When a fraction ρ_1 of an IO stream is processed by two cache tiers, $m'_1(x')$ and $m'_2(y')$ denote the cache miss ratios observed at the tier-1 and tier-2 caches, respectively. The miss ratio after the tier-1 cache $m'(x',0)$ will become $\rho_1 m'_1(x')$ due to a fraction ρ_1 of an input IO stream. Then the *fractional miss ratio* is $m'(x',y') = \rho_1 m'_1(x')m'_2(y')$. The fractional miss ratio is the miss ratio when a fraction of an IO stream is used.
- **Scenario 3: Sampling before both tier-1 & tier-2 caches.** We sample the missed IO stream after the tier-1 cache of size x' (which is denoted by $OUT'(x',0)$) again with fraction ρ_2, and feed that into another tier-2 cache of size y''. Then the fractional miss ratio $m''(x',y'')$ equals $\rho_1 m'_1(x')\rho_2 m''_2(y'')$.

With these three scenarios described, we introduce our assumptions and theorems.

Assumption 1 *If $x' = \rho_1 x$ and $y' = \rho_1 y$, then the miss ratio at the tier-2 cache $m'_2(y')$ and $m_2(y)$ are statistically similar.*

This was shown empirically in §3.2. With this assumption we can now find the relationship between the fractional miss ratios $m'(x',y')$, $m''(x',y'')$ in Scenarios 2 & 3 and the miss ratio $m(x,y)$ in Scenario 1 for a two-tier cache system.

Theorem 1 *For an input IO stream that is processed by two tiers of caches with the resulting miss ratio surface $m(x,y)$, a fraction ρ_1 of spatial sampled IO stream will have a fractional miss ratio surface:*

$$m'(x',y') = \rho_1 m(\frac{x'}{\rho_1}, \frac{y'}{\rho_1})$$

Proof: In a two-tier caching setup, when a fraction ρ_1 of an input IO stream passes through the tier-1 cache of size x' and then the tier-2 cache of size y', the fractional miss ratio is $m'(x',y') = \rho_1 m'_1(x')m'_2(y')$. From the Talus theorem, we find $m'_1(x') = m_1(x'/\rho_1)$. Under Assumption 1, $m'_2(y') = m_2(y'/\rho_1)$. Putting them all together, we obtain:

$$m'(x',y') = \rho_1 m_1(\frac{x'}{\rho_1})m_2(\frac{y'}{\rho_1}) = \rho_1 m(\frac{x'}{\rho_1}, \frac{y'}{\rho_1}).$$

Theorem 2 *For a given IO stream that is processed by two tiers of the caches with resulting miss ratio surface $m(x,y)$, downsampling with fraction ρ_1 before the tier-1 cache of size x' and again downsampling with fraction ρ_2 before the tier-2 cache of size y'' results in new fractional miss ratio surface:*

$$m''(x',y'') = \rho_1\rho_2 m(\frac{x'}{\rho_1}, \frac{y''}{\rho_1\rho_2})$$

Proof: From the Talus theorem, we find $m''_2(y'') = m'_2(y''/\rho_2)$. Hence, $m''(x',y'') = \rho_1 m'_1(x')\rho_2 m'_2(\frac{y''}{\rho_2})$. As $m'(x',y') = \rho_1 m'_1(x')m'_2(y')$, $m''(x',y'') = m'(x',\frac{y''}{\rho_2})\rho_2$. Applying Theorem 1, we obtain:

$$m''(x',y'') = \rho_1\rho_2 m(\frac{x'}{\rho_1}, \frac{y''}{\rho_1\rho_2}).$$

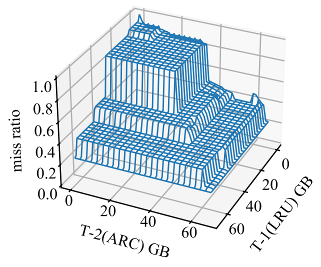
We now use Theorem 2 to remove cliffs in the miss ratio surface. Let $m(x,y)$ denote a region of the miss ratio surface of a two-tier caching system without any partitioning, for $x \in [X_1,X_2], y \in [Y_1,Y_2]$, where $m(X_1,Y_1), m(X_1,Y_2), m(X_2,Y_1)$ and $m(X_2,Y_2)$ are known values. The $m_{eMRC}(x,y)$ can be implemented by partitioning both cache tiers.

Figure 8 illustrates the partitioning scheme. The final miss ratio consists of four different fractional miss ratios after the tier-2 cache. Hence, $m_{eMRC}(x,y)$ is the summation of all the four miss ratios:

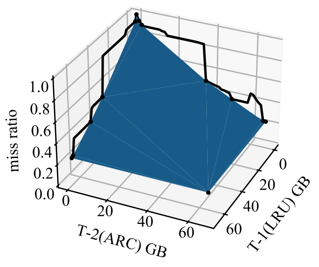
$$\begin{aligned} m_{eMRC}(x,y) = {} & m_1(\sigma_1 x, \rho_1\sigma_2 y) \\ & + m_2(\sigma_1 x, \rho_1(1-\sigma_2)y) \\ & + m_3((1-\sigma_1)x, (1-\rho_1)\sigma_2 y) \\ & + m_4((1-\sigma_1)x, (1-\rho_1)(1-\sigma_2)y). \end{aligned} \tag{2}$$

where

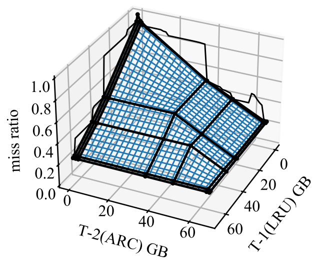
$$\rho_1 = \frac{X_2-x}{X_2-X_1}, \sigma_1 = \rho_1\frac{X_1}{x}, \rho_2 = \frac{Y_2-y}{Y_2-Y_1}, \sigma_2 = \rho_2\frac{Y_1}{y}. \tag{3}$$



(a) Original MRC, constructed by evaluating 2,601 cache configurations.



(b) Ideal convex MRC, constructed from original MRC.



(c) eMRC, constructed using 2 MRCs on the edge and 30 additional data points selected by convex hull approximation.

Figure 7: Convex hull approximation applied to MSR trace `web_2`.

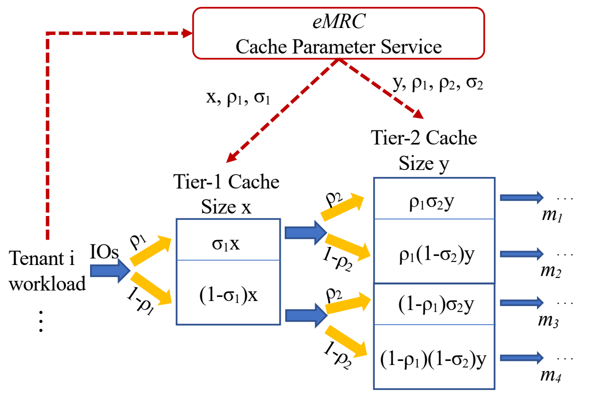


Figure 8: *e*MRC partitioning scheme for two-tier caching.

Since σ_1 and σ_2 are dependent upon ρ_1 and ρ_2, the values of ρ_1 and ρ_2 dictate the convex shape of the $m_{eMRC}(x,y)$ in the region of $x \in [X_1, X_2]$ and $y \in [Y_1, Y_2]$.

Equations 2 and 3 can be combined into:

$$\begin{aligned} m_{eMRC}(x,y) = {} & m_1(\rho_1 X_1, \rho_1\rho_2 Y_1) \\ & + m_2(\rho_1 X_1, \rho_1(1-\rho_2)Y_2) \\ & + m_3((1-\rho_1)X_2, (1-\rho_1)\rho_2 Y_1) \\ & + m_4((1-\rho_1)X_2, (1-\rho_1)(1-\rho_2)Y_2). \end{aligned} \tag{4}$$

By applying Theorem 2 into m_1, m_2, m_3, m_4 in Equation 4, we finally obtain:

$$\begin{aligned} m_{eMRC}(x,y) = {} & \rho_1\rho_2 m(X_1, Y_1) \\ & + \rho_1(1-\rho_2)m(X_1, Y_2) \\ & + (1-\rho_1)\rho_2 m(X_2, Y_1) \\ & + (1-\rho_1)(1-\rho_2)m(X_2, Y_2). \end{aligned} \tag{5}$$

In summary, given a cliff region in $m(x,y)$ with four known boundary values, we can remove the cliffs with our *e*MRC partitioning scheme and parameter set $\{\rho_1, \rho_2\}$.

3.4 Convex Hull Approximation

Figure 7b is an *ideal convex MRC* of the known miss ratio surface, shown in Figure 7a. It is generated using the convex hull algorithm from quickhull [3]. The ideal convex MRC represents the best case in multi-tier cache management. But it requires an algorithm to remove miss ratio cliffs in regions bounded by three arbitrary points in space, which does not exist.

Our partitioning scheme can achieve cliff removals in a "grid" region $(x,y), x \in [X_1, X_2], y \in [Y_1, Y_2]$. To apply it for the whole miss ratio surface, we partitioned the surface using the ideal convex MRC.

In particular, this is a four-step process: (1) Obtain the original MRC; (2) Generate the ideal convex MRC using the convex hull algorithm; (3) Construct *e*MRC grid regions; (4) Perform cliff removal in each region.

The process is very time-consuming because of step (1). In our case, for an MRC surface with cache size resolution $res = 51$, a total of 2,601 miss ratio data points have to be evaluated.

But we found that *the vertices of the ideal convex MRC almost entirely reside on the edge of the surface, where the tier-1 or tier-2 cache size is 0*. And we can find those vertices by constructing 2D convex hulls of the two MRCs, where the tier-1 or tier-2 cache is 0, respectively (black curves in Figure 7b).

Based on our evaluation with all traces, we observed that: at least 81% of the vertices in the *Ideal Convex MRC* are associated with the convex hull points of 2D MRCs along the two edges plus one additional cache configuration when both the tier-1 and tier-2 caches take the maximum values.

We now can greatly simplify the process of applying *e*MRC to the whole surface with new steps (1) and (2): (1) Obtain the two MRCs along the edges where the tier-1 or tier-2 cache is 0, respectively. (2a) Obtain the vertices along the two edges using convex hull algorithm. (2b) Obtain the additional miss ratio values on *e*MRC grid points (highlighted in Figure 7c).

In the particular case outlined in Figure 7, our new approach only needs to evaluate two MRCs along the edges plus 30 additional data points, while the conventional approach requires knowledge of all 2,601 data points.

3.5 Extension to Multi-Tier Caching (3+)

To generalize *e*MRC for N-tier caching, we first extend our notations used in Figures 5, 6 and 8 to vectors:

- $B = \{\mathbf{b} \mid b_i \in \{X_1^i, X_2^i\}, \forall i \in \{1,2,...,N\}\}$ for the set of boundary coordinates that defines a multi-dimensional hypercube cliff region.
- $\mathbf{x} = [x_1, x_2, ...x_N]$ for the cache size on each tier within the cliff region, $X_1^i \le x_i \le X_2^i, \forall i \in \{1,2,...,N\}$.
- $\boldsymbol{\rho} = [\rho_1, \rho_2, ...\rho_N]$ for the fraction of an IO stream before each tier.
- $\boldsymbol{\sigma} = [\sigma_1, \sigma_2, ...\sigma_N]$ for the cache partition ratio on each tier.
- $m(\mathbf{x})$ is the original miss ratio function without any cache partitioning.

Theorem 3 (Extension of Theorem 2) *For an input IO stream that is processed by N tiers of the cache with resulting miss ratio function $m(\mathbf{x})$, when a fraction $\boldsymbol{\rho}$ is applied to the IO stream before each cache tier, the new fractional miss ratio function is:*

$$m^N(\mathbf{x}) = \prod_{i=1}^N \rho_i \cdot m(\mathbf{x}'), \text{ where } \mathbf{x}' = \left[\frac{x_1}{\rho_1}, \frac{x_2}{\rho_1\rho_2}, ..., \frac{x_N}{\prod_{i=1}^N \rho_i}\right].$$

Extension of Equation 3. The *e*MRC partitioning parameter for each cache tier is defined using the following equations:

$$\rho_i = \frac{X_2^i - x_i}{X_2^i - X_1^i}, \forall i \in \{1,2,...,N\}$$

$$\sigma_i = \rho_i \frac{X_1^i}{x_i}, \forall i \in \{1,2,...,N\}.$$

Extension of Equation 5. The miss ratio after the tier-N cache is composed of 2^N different miss ratios (from 2^N partitions; see Figure 8). Hence, $m_{eMRC}(\mathbf{x})$ is computed by aggregating all the 2^N miss ratios $m_{eMRC}(\mathbf{x}) = \sum_{i=1}^{2^N} m_i^N$, where m_i^N is the miss ratio after each partition at the tier-N cache.

By applying Theorem 3 into this equation, we finally obtain our *e*MRC miss ratio function:

$$m_{eMRC}(\mathbf{x}) = \sum_{\mathbf{b} \in B} \left(\prod_{i=1}^N f_i \cdot m(\mathbf{b}) \right),$$

$$\text{where } f_i = \begin{cases} \rho_i, & b_i = X_1^i \\ 1 - \rho_i, & b_i = X_2^i \end{cases}, \forall i \in \{1,2,...,N\}.$$

4 *e*MRC-based Cache Orchestration

We present the ORCA (ORchestration for CAches) multi-tier cache orchestration framework, which identifies the optimal cache configuration that meets tenants' diverse SLO requirements using *e*MRC while also minimizing the cloud provider's cost. ORCA is designed for a cloud provider to serve customers with elastic cache resources. This objective can be achieved by providing each tenant with the lowest possible cache cost to meet its SLO and accommodate as many tenants as possible.

While our approach addresses N-tier caching systems for $N \ge 2$, we consider a two-tier caching distributed storage system to simplify our discussion. The system consists of a faster and more expensive tier-1 cache (e.g., DRAM) closer to the clients and a slower and relatively cheaper tier-2 cache (e.g., NVMe SSD) while having a central storage backend (e.g., SATA SSD). For each tenant i with a specific SLO requirement in IOPS (which is a key SLO metric for storage IO), denoted as R_{IOPS}^i, our cache orchestration framework will find an optimal cache configuration $\{P_1^i, P_2^i, T_1^i, T_2^i\}$ satisfying tenant i's SLO with a provisioned IOPS $P_{IOPS}^i \ge R_{IOPS}^i$ while minimizing overall cache resources allocated, where P_j^i is the cache eviction policy at tier-j for tenant i, and T_j^i is the allocated cache capacity at tier-j for tenant i.

4.1 SLO Modeling with *e*MRC

To find an optimal cache size and policy configuration for a given SLO, we first need to map *e*MRC's miss ratio information to IOPS. The provisioned IOPS of a storage workload depends on both the cache configuration $\{P_1^i, P_2^i, T_1^i, T_2^i\}$ and the performance of the underlying hardware.

An exact analysis of IOPS performance is complicated with MRCs, as it requires all the input IO to be reorganized into an equal chunk size (e.g., 4KB), making it difficult to study any performance benefit of large sequential IOs. We treat both read and write IOs as cache references during MRC generation, modeling a simple write-back cache policy. We present an analytical lower bound of provisioned IOPS using 4KB random IO performance at the tier-1 cache ($IOPS_{T_1}$), the tier-2 cache ($IOPS_{T_2}$), and the storage backend ($IOPS_B$). Miss ratios for each cache tier, M_1^i and M_2^i, as well as the joint miss ratio M^i can be obtained from a *e*MRC miss ratio function for tenant i: $(M^i, M_1^i, M_2^i) = F_{eMRC_i}(T_1^i, T_2^i, P_1^i, P_2^i)$.

The provisioned IOPS of tenant i can then be modeled as:

$$IOPS^i = \frac{1}{\frac{1-M_1^i}{IOPS_{T_1}} + \frac{M_1^i - M^i}{IOPS_{T_2}} + \frac{M^i}{IOPS_B}} \tag{6}$$

where $1 - M_1^i$, $M_1^i - M^i$ and M^i represents the probability that a storage IO is executed at the tier-1 cache, the tier-2 cache and storage back-end, respectively.

The IOPS function $F_{IOPS}(T_1^i, T_2^i, P_1^i, P_2^i)$ from Eq. 6 is visualized by a 3D plot in Figure 9. This *IOPS surface* shows the relationship of provisioned IOPS with respect to the cache size at each tier for a specific cache policy combination, e.g., tier-1 uses LRU and tier-2 uses ARC.

*e*MRC surface is convex, and the corresponding IOPS surface will be an always increasing surface for the cache size in each tier, according to Eq. 6.

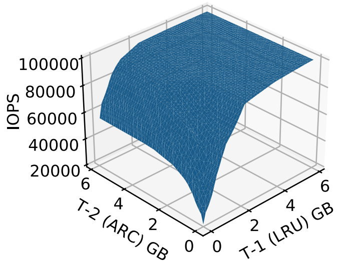


Figure 9: IOPS surface translated from *e*MRC surface for MSR `proj_3` trace.

4.2 Cache Orchestration with *e*MRC

ORCA provides an optimal cache configuration $\{T_1^i, T_2^i, P_1^i, P_2^i\}$ that minimizes the overall cost of cache resources while meeting each tenant's SLO requirements. The cost on cache resource of tenant i with capacity configuration $\{T_1^i, T_2^i\}$ can be denoted as $C^i = C_1 \times T_1^i + C_2 \times T_2^i$, where C_1 and C_2 are the unit costs for the tier-1 and tier-2 caches, respectively.

For M tenants, the problem can be formulated as follows:

$$\text{minimize} \quad \sum_{i=1}^{M} C^i \tag{7a}$$

$$\text{subject to} \quad C^i = C_1 T_1^i + C_2 T_2^i \tag{7b}$$

$$IOPS^i = F_{IOPS}(T_1^i, T_2^i, P_1^i, P_2^i) \tag{7c}$$

$$IOPS^i \geq R_{IOPS}^i, \forall i \in \{1, \ldots, M\} \tag{7d}$$

$$\text{var.} \quad \{T_1^i, T_2^i, P_1^i, P_2^i\} \tag{7e}$$

The unit cost of the tier-1 and tier-2 caches (C_1 and C_2) are adjustable by the cloud provider. By adjusting the ratio of C_1/C_2, the cloud provider will be able to shift utilization between the tier-1 and tier-2 caches.

4.3 Two-Stage ORCA Optimization

The cache orchestration problem is challenging even with *e*MRC. We need to search through tons of configuration candidates for each tenant's optimal multi-tier cache configuration, with all the SLO and cache capacity constraints, which would incur a heavy computation overhead. Because an online cache orchestration system requires an efficient and yet accurate algorithm to adapt to large-scale, dynamic workloads, we propose an efficient Two-stage ORCA optimization algorithm. The ORCA optimization splits the optimization over its objective (minimizing cache resource cost) and its constraints (required IOPS and cache sizes), significantly reducing the search space for the optimization problem.

We propose Algorithm 1 to construct the set B^i, which contains cache configurations that are just enough to satisfy $R_{IOPS}^i + D^i$ for tenant i, where R_{IOPS}^i is the requested IOPS from tenant i and D^i is an additional margin added to account for the prediction error caused by spatial downsampling and workload dynamics. We define the tier-1 cache size x_1, algorithm step m as a function of cache size resolution res and T_{MAX} [2]. We define x_2 as the tier-2 cache size and the algorithm step as n in a similar way. Then, for each cache policy configuration $\{P_1^i, P_2^i\}$, we first find the first cache configuration (via binary search) that satisfies the IOPS SLO when the tier-2 cache size is zero $x_2[0] = 0$. Then we search among its nearest neighbor points where $x_2[1] = \frac{1}{res-1} \cdot T_{MAX}$ to find the next candidate cache configuration.

Algorithm 1: ORCA-Space Search

```
Input: $T_{MAX}, F_{IOPS}, res, R_{IOPS}^i, D^i$
tier-1 cache size $x_1[m] = \frac{m}{res-1} \cdot T_{MAX}, m \in range(res)$
tier-2 cache size $x_2[n] = \frac{n}{res-1} \cdot T_{MAX}, n \in range(res)$
$R_{IOPS}^i = R_{IOPS}^i + D^i$
for each cache policy combination $(P_1^i, P_2^i)$ do
    Do binary search to find $m$ such that
    $F_{IOPS}(x_1[m-1], x_2[0]) < R_{IOPS}^i$ and
    $F_{IOPS}(x_1[m], x_2[0]) >= R_{IOPS}^i$
    Add $(x_1[m], x_2[0], P_1^i, P_2^i,)$ to $B^i$
    for n in [1, ..., res−1]) do
        $m' = m$
        use $(x_1[m'], x_2[n])$ as starting point to find m such
          that
        $F_{IOPS}(x_1[m-1], x_2[n]) < R_{IOPS}^i$ and
        $F_{IOPS}(x_1[m], x_2[n]) >= R_{IOPS}^i$
        Add $(x_1[m], x_2[n], P_1^i, P_2^i)$ to $B^i$
    end
end
Output: $B^i$, a reduced set of cache configurations
```

The algorithm will complete when the search reaches T_{MAX} for the tier-2 cache size; at this point, B^i will be constructed for a given cache policy configuration. We do this for all cache policy configurations to construct a complete set of B^i for each tenant i; such a set of B^i includes all cache configurations that are just enough to satisfy tenant i's SLO plus a margin. We then apply Algorithm 2 to perform optimization over cache configuration set B^i for each tenant i, which will provide the cache configuration with minimum cache resource cost, defined in Eq. (7a) and Eq. (7b). According to Eq. (7a), $\sum_{i=1}^{M} C^i$ gets minimized when each C^i gets minimized; thus Algorithm 2 aims at minimizing the cost for each tenant i over cache configurations that already meet the SLO requirements from Algorithm 1.

5 Performance Evaluation

We evaluated our *e*MRC approximation technique and ORCA on a server with two Xeon Gold 6142 CPUs at 2.6GHz with 384GB of memory, with a subset of Microsoft's MSR IO traces obtained from SNIA [1, 18]. We treat both read and write IOs as cache references during MRC generation, modeling a simple write-back cache policy. We break any large IOs into 4KB blocks. For IOs smaller than 4KB, we treat them as a full 4KB access. The traces we use all have more

[2]The unique IO entries in the trace.

Algorithm 2: ORCA-Cost Objective

```
for each cache tenant $i$ do
    Call ORCA-Space Search (Algorithm #1) to obtain
      cache configuration set $B^i$
    Initialize $c_{min}$=MAX
    for $b$ in $B^i$ do
        Compute current cost objective value $c$ from $b$
        if $c < c_{min}$ then
            $c_{min} = c$ , $b_{out} = b$
        end
    end
    Add $(i, b_{out}, c_{min})$ to $S$
end
Output: $S$
```

Server	Function	Traces Used
hm	Hardware monitoring	0, 1
mds	Media server	0, 1
prn	Print server	0, 1
proj	Project directories	0, 1, 2, 3, 4
prxy	Firewall/web proxy	0, 1
rsrch	Research projects	0
src1	Source control	0, 1, 2
src2	Source control	0, 1, 2
stg	Web staging	0, 1
ts	Terminal server	0
usr	User home directories	0, 1, 2
wdev	Test web server	0
web	Web/SQL server	0, 1, 2

Table 1: Microsoft MSR traces [1] used.

than one million 4KB IO entries after processing, their names and functions are shown in Table 1. We omitted small traces inappropriate for downsampling methods. In our evaluation for *e*MRC and ORCA, where we used the entire trace, we applied different spatial sampling rates ranging from 0.1 for traces with 1M entries to 0.0001 for traces with more than 600M entries.

For the ORCA evaluation, we assume that we have a storage cloud that uses DRAM as the tier-1 cache, NVMe SSDs as the tier-2 cache, and SATA SSD as the backend storage; and for random 4KB IOs they perform at $IOPS_{T1} = 10,000,000$, $IOPS_{T2} = 100,000$, and $IOPS_B = 20,000$.

5.1 *e*MRC Performance

Computation speedup with convex-hull approximation. The main challenge of utilizing miss ratio information in multi-tier caching is the exponentially increasing size of all cache configurations. With *e*MRC's convex-hull approximation, only a limited number of cache configurations will be evaluated to reduce the computation cost. We evaluated two to four tiers of caches using a mix of stack and non-stack based replacement policies (e.g., LRU→ARC→LRU→ARC) on all the traces we have with resolution $res = 51$, meaning the cache size of each tier can take 51 different values during MRC generation.

We compare the number of data points required to generate *e*MRC's continuous and convex miss ratio function with and without convex-hull approximation. Figure 10 shows the speedup factor for a different number of cache tiers. We can see that the benefits increase with the number of cache tiers. The convex-hull approximation resulted in 14x speedup on average for two cache tiers and 4,527x speedup on average for four cache tiers. We have more speedup for workloads with more cliff regions and less for the ones without many cliffs because the presence of cliffs reduces the number of data points that need to be evaluated. Most real workloads have some level of cliff features, and these features will help to speed up our algorithm. Figure 10 shows the number of data points required but the actual processing time will vary based on the eviction policies used in each cache tier because stack-based eviction policies can benefit from SHARDS, which can evaluate multiple cache sizes with a single pass of the trace. For the two-tier caching using LRU and ARC policy, the largest difference in computation time is 54 seconds for the `src1_0` trace with 200M IO entries with sample rate $R = 0.0002$ compared to 42 minutes for generating the whole original MRC (47 times faster). The smallest difference in computation time is 2 minutes versus 8.2 minutes for the `prxy_0` trace with 22M IO entries with $R = 0.02$ (4.1 times faster).

Accuracy of cliff removal. For each trace, we also evaluated 2,601 individual cache configurations for two-tier caching with *e*MRC partitioning parameters and compared it with the predicted miss ratio using *e*MRC. We use the Mean Absolute Error (MAE) to examine the distance between these two surfaces. As shown in Figure 11, we were able to achieve < 2% MAE for all the traces we use, meaning the *e*MRC partitioned caches act as predicted in removing performance cliffs.

Convex-hull approximation. Our convex-hull algorithm is based on the two miss ratio curves at the edge of the miss ratio surface; the convex hull points along the edges are the majority of the vertices of the ideal convex MRC. The shape of the original miss ratio surface dictates how much we can reduce the miss ratios by removing cliffs, and also the number of regions generated by the convex-hull algorithm, which is related to computation time. Figure 13 highlights the convex-hull algorithm with several traces with representative shapes.

For traces with shapes like `proj_2` and the `web_2` trace shown earlier, our algorithm is able to construct the whole *e*MRC surface with very few data points and significantly reduces the performance cliffs in the entire space.

For traces with shapes like `prxy_0`, the original surface does not have any significant cliffs; our algorithm needs more data points to construct the whole *e*MRC surface. Although it only reduces performance cliffs in small areas, it only needs to evaluate 25% cache size configurations compared with generating the complete miss ratio surface.

For the `prn_0` trace, the particular shape of the original MRC causes our algorithm to produce non-convex regions in

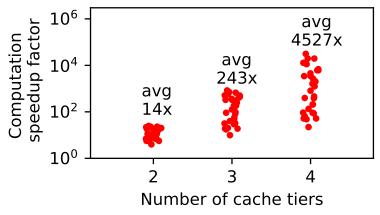


Figure 10: Comparison of data points required to generate the whole *e*MRC, with and without convex-hull approximation.

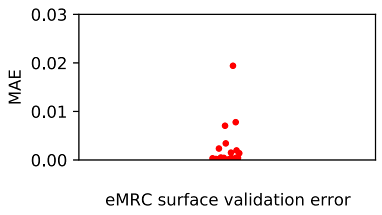


Figure 11: Error between the predicted *e*MRC surface versus the evaluated miss ratio surface using *e*MRC partitioning parameters over all traces.

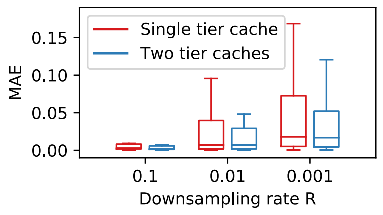


Figure 12: MRC generation error without any partitioning, with respect to different down-sampling factor R.

the area marked by red lines in Figure 13. This can be easily corrected by combining the multiple small regions in question into a single large region using the red lines' four boundary points. Since all cache simulations on the marked data points are already done, reconfiguring the grid regions and applying *e*MRC does not have any additional overheads. This also demonstrates the flexibility of our *e*MRC partitioning scheme; it can remove a cliff in any region marked by four boundary points with known miss ratio values.

Spatial downsampling in multi-tier cache. Other researchers have studied the effect of spatial downsampling for a single-tier cache. Our experiments show that it also works well for multi-tier caching. When downsampled with a factor R, the computation time is proportionally reduced by R while maintaining a relatively accurate miss ratio estimation. To evaluate the accuracy, we calculated the MAE between two discrete miss ratio surfaces with and without spatial downsampling using the first 10M entries of all the traces. Both surfaces are generated without any cache partitioning. Figure 12 shows the MAE distribution for different downsampling rates across the evaluation traces. Note that the traces are downsampled only once before the IO stream is processed at the tier-1 cache. We can see that single-tier and two-tier caches have similar error rates.

5.2 ORCA Performance

This subsection shows how our approach can help cloud providers determine efficient cache configurations for cloud tenants with diverse SLOs. We assume the SLOs of tenants are expressed in requested IOPS R_{IOPS}.

Cache optimization for a single tenant. We consider LRU and ARC cache policies. Then four different cache policy combinations are possible (four *e*MRC surfaces per trace). For all the traces, we use resolution $res = 51$. ORCA will first obtain each *e*MRC surface with cache partitioning. *e*MRC can then determine the cache partitioning parameters for each cache size combination in the first and second-tier caches. From the *e*MRC surface, we can derive the F_{IOPS} surface, as shown in Figure 14b. Because the MAE between our *e*MRC prediction and *e*MRC evaluation is very small, the F_{IOPS} surface is also very close to F'_{IOPS} evaluated values (with the Mean Absolute Percentage Error (MAPE) of 1% using all traces). From Figures 14a and 14b, we can see that by utilizing cache partitioning, we not only improved overall IOPS performance but also removed performance plateaus and local dips.

Figure 14b shows F_{IOPS} with *e*MRC cache partitioning where the tier-1 replacement policy is LRU and the tier-2 policy is ARC. Due to page limitations, we are only presenting the results for six traces here. In Figure 14b, the solid blue areas represent cache configurations that lead to $IOPS > R_{IOPS}$. Note that we do not have to search the entire F_{IOPS} surfaces, which contains 2,601 sampling points for every cache policy combination when using resolution $res = 51$. We are only interested in the sampling points that are just enough to meet the tenant's SLO, eliminating unnecessary solutions; hence they will be ≤ 51 cache size combinations for each cache policy pair. By using Algorithm 1, we can calculate the entire related cache configuration set B within a second.

Cache optimization for multiple tenants. In this experiment, we test how ORCA can effectively allocate cache resources for multiple tenants. Six tenants are using the six traces presented in Figure 14 with the corresponding requirements R_{IOPS}, respectively. We assume the cloud provider can handle many tenants and has sufficient cache resources to initially accommodate these six tenants but seeks to limit the resources to meet the requirements. In the ORCA optimization equation (Eq. 7b), the cost ratio between the two cache tiers $C1/C2$ can be tuned by the cloud provider to adjust utilization between them. This evaluation sets $C1/C2 = 10$, roughly representing the per-GB cost difference between DDR4 DRAM and datacenter NVMe SSDs.

Table 2 shows how ORCA allocates cache resources for the six tenants with provisioned IOPS P_{IOPS}. As seen in the table, tenants using `mds_1`, `proj_2` and `src1_0` traces with relatively low R_{IOPS} requirements will be only assigned the tier-2 cache, and other tenants will be assigned combined tier-1 and tier-2 caches.

6 Related Work

Research on efficient cache allocation among tenants has focused on single-tier caching architectures. Two recent

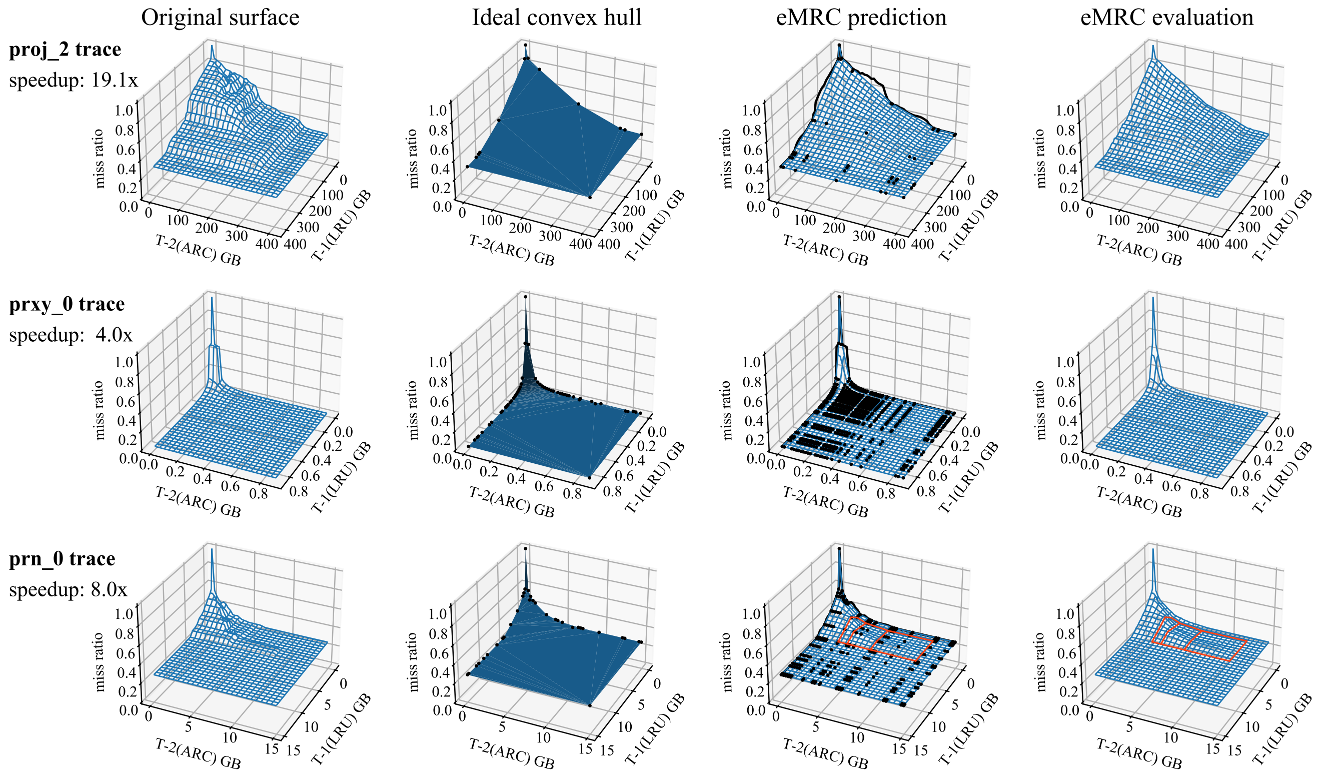


Figure 13: Effect of convex hull algorithm on various original miss ratio surface shapes.

Tenant trace/size	R	R_{IOPS}	P_{IOPS}	Tier1 cache GB/policy	Tier2 cache GB/policy
mds_1/ 25M	0.002	21500	21580	0	6.72/ARC
proj_2/ 340M	0.0002	40000	40397	0	294.29/LRU
proj_3/ 7.7M	0.01	60000	61012	0.12/ARC	1.63/ARC
src2_2/ 17M	0.004	60000	60081	15.79/LRU	19.84/LRU
src1_0/ 406M	0.0002	60000	61785	0	106.34/ARC
web_2/ 74M	0.001	60000	60361	20.08/ARC	65.60/ARC

Table 2: Cache optimization by ORCA for 6 tenants.

works [21, 22] present an approach allowing host-side page caches to be partitioned by VMs individually so those cache parameters can be configured independently between tenants. Cloudcache [2] proposes an on-demand cache management solution to meet each tenant's performance demand by introducing a new cache demand model. Recent MRC approximation techniques [7, 10, 24–26] also have focused on improving MRC efficiency for online cache management for single-tier caching.

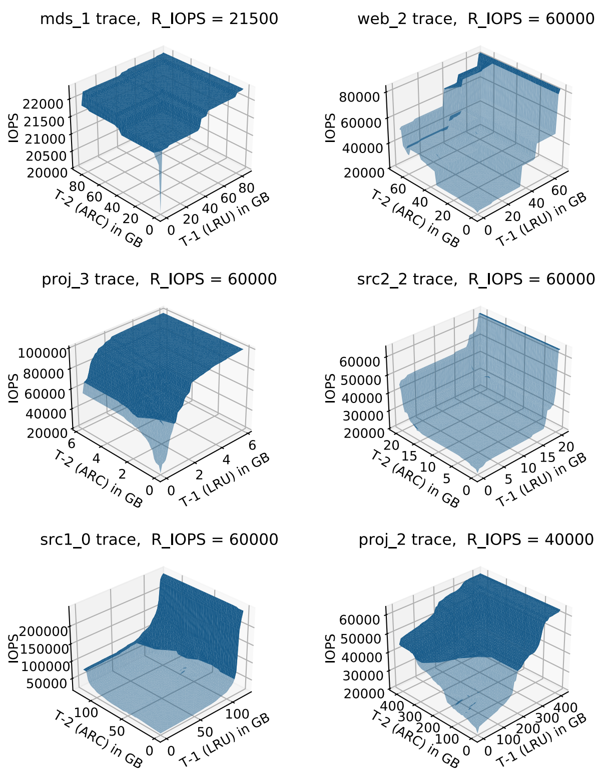
Existing efforts on multi-tier caching include works focusing on minimizing the duplication in cached data among different cache tiers to improve cache efficiency [9, 14, 15, 19, 27–29, 31, 32]. Stefanovici et al. [23] propose a software-defined cache allocation approach for multi-tier caching, which allows cloud providers to coordinate multiple tiers of cache to provide isolation and QoS for tenants.

Dynacache [5] is an LP (Linear Programming) solver to find an optimal slab cache configuration when the total cache size is fixed for single-tenant, single-tier cache scenarios. Cliffhanger [6] extends Dynacache. Cliffhanger uses a novel iterative algorithm to identify a local hit rate gradient without constructing the whole MRC and uses the gradient at different cache slabs to determine the optimal cache configuration. It also utilizes the Talus algorithm to remove performance cliffs. However, both Cliffhanger and Dynacache tackle the single-tier cache focusing on single application optimization, whereas our work targets multi-tier, multi-tenant caching systems.

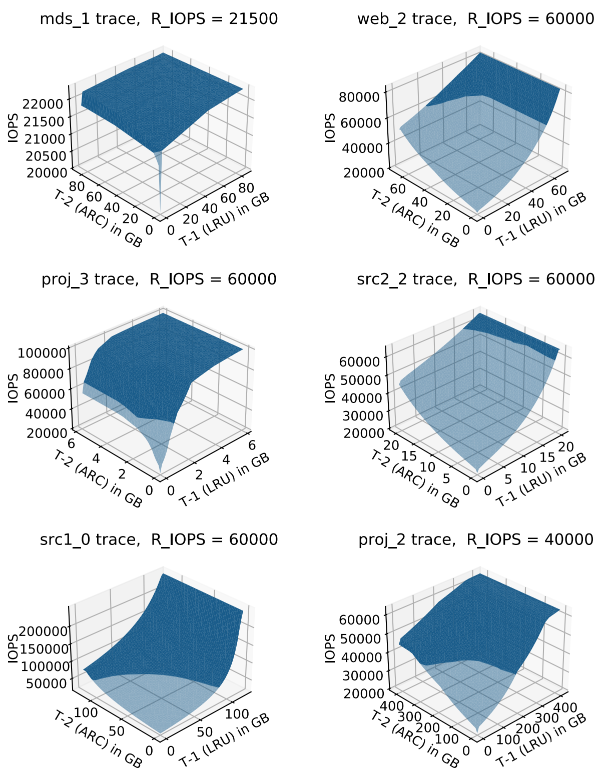
7 Discussion

*e*MRC allows the efficient study of miss ratio profiles with built-in cliff removal for multi-tier caching systems. It is the first step towards efficient online cache management for multi-tier caching with two challenges: 1. how to efficiently recompute miss ratio profiles/MRCs periodically, 2. how to manage the boundary migration of cache partitions.

Recomputing MRC periodically. Prior works generate the MRC for single-tier cache in a fixed time or using a fixed rate periodically. Talus [4] is originally designed for CPU cache. It recalculates the MRC on a fixed interval (e.g., 10ms) and uses that to configure cache partitions for the next interval. SLIDE [24] in Miniature Simulation recalculates the MRC every 1M IO entries with Exponentially Weighted Moving Average (EWMA). Both Talus and SLIDE require the knowl-



(a) IOPS surfaces without cache partitioning.



(b) IOPS surfaces with cache partitioning.

Figure 14: IOPS surfaces with and without cache partitioning for 6 MSR traces.

edge of the whole MRC to perform cliff removal and cache management. Cliffhanger [6] is another Talus inspired work that can remove performance cliff without knowledge of the whole MRC, but due to the limited visibility, it can only work for one cliff. Talus based approaches require the knowledge of the entire MRC to achieve maximum utility. This makes multi-tier cache analysis and management challenging, as the computation time goes up exponentially. Our *e*MRC with a convex-hull approximation can construct multi-tier miss ratio functions with regional cliff removal while using limited data points without generating the whole original miss ratio function. This enables the timely and periodic generation of multi-dimensional miss ratio functions for multi-tier caching.

Managing the boundary migration of cache partitions. Our *e*MRC based approach relies heavily on cache partitioning within each workload, 2^N partitions for the Nth cache tier to be specific. For a practical online system, the size and partitioning parameters of caches will change over time, and it may cause some cached items to fall into the wrong partitions. Talus [4] builds on top of CPU cache partitioning schemes such as Vantage [20]. SLIDE [24] uses a shadow partitioning based approach to manage partition boundary migration. SLIDE uses a single unified cache to handle IO and defer partitioning decisions till eviction time.

Our future work includes incorporating such boundary migration mechanisms for cache partitions into our ORCA design and evaluating it with more real-world traces.

8 Conclusion

Our *e*MRC approximation technique enables efficient MRC generation for multi-tier caching. *e*MRC uses 1) a partitioning scheme to remove performance cliffs in a grid region with known boundary miss ratio values and 2) a convex hull approximation technique that generates all grid regions efficiently using a small number of sampling points. Based on *e*MRC, we also designed ORCA, a multi-tier cache orchestration framework that uses a lightweight two-stage algorithm that effectively provides efficient cache configurations for tenants with diverse SLOs. Our performance evaluation shows that our *e*MRC and ORCA are useful tools for multi-tier cache orchestration.

Acknowledgments

Thanks to our shepherd Carl Waldspurger and other anonymous reviewers for their valuable feedback and suggestions. This research is supported in part by NSF award #1705095.

References

[1] SNIA IOTTA repository. http://iotta.snia.org.

[2] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. Cloudcache: On-demand flash cache management for cloud computing. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 355–369, Santa Clara, CA, 2016. USENIX Association.

[3] C Bradford Barber, David P Dobkin, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[4] Nathan Beckmann and Daniel Sanchez. Talus: A simple way to remove cliffs in cache performance. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 64–75. IEEE, 2015.

[5] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. Dynacache: Dynamic cloud caching. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[6] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. Cliffhanger: Scaling performance cliffs in web memory caches. In *13th USENIX Symposium on Networked Systems Design and Implementation NSDI 16)*, pages 379–392, 2016.

[7] D. Eklov and E. Hagersten. Statstack: Efficient modeling of lru caches. In *2010 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*, pages 55–65, March 2010.

[8] Tyler Estro, Pranav Bhandari, Avani Wildani, and Erez Zadok. Desperately seeking... optimal multi-tier cache configurations. In *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*, 2020.

[9] Binny S. Gill. On multi-level exclusive caching: Offline optimality and why promotions are better than demotions. In *6th USENIX Conference on File and Storage Technologies (FAST 08)*, San Jose, CA, 2008. USENIX Association.

[10] 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, Denver, CO, 2016. USENIX Association.

[11] Dejun Jiang, Yukun Che, Jin Xiong, and Xiaosong Ma. ucache: A utility-aware multilevel ssd cache management policy. In *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 391–398. IEEE, 2013.

[12] Richard E. Kessler, Mark D Hill, and David A Wood. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers*, 43(6):664–675, 1994.

[13] R. Koller, A. J. Mashtizadeh, and R. Rangaswami. Centaur: Host-side ssd caching for storage performance control. In *2015 IEEE International Conference on Autonomic Computing*, pages 51–60, July 2015.

[14] Wenji Li, Gregory Jean-Baptise, Juan Riveros, Giri Narasimhan, Tony Zhang, and Ming Zhao. Cachededup: In-line deduplication for flash caching. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 301–314, Santa Clara, CA, 2016. USENIX Association.

[15] Xuhui Li, Ashraf Aboulnaga, Kenneth Salem, Aamer Sachedina, and Shaobo Gao. Second-tier cache management using write hints. In *Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies - Volume 4*, FAST'05, pages 9–9, Berkeley, CA, USA, 2005. USENIX Association.

[16] 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.

[17] Nimrod Megiddo and Dharmendra S. Modha. Arc: A self-tuning, low overhead replacement cache. In *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies*, FAST '03, pages 115–130, Berkeley, CA, USA, 2003. USENIX Association.

[18] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS)*, 4(3):10, 2008.

[19] L. Ou, X. He, M. J. Kosa, and S. L. Scott. A unified multiple-level cache for high performance storage systems. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 143–150, Sept 2005.

[20] Daniel Sanchez and Christos Kozyrakis. Vantage: scalable and efficient fine-grain cache partitioning. In *Proceedings of the 38th annual international symposium on Computer architecture*, pages 57–68, 2011.

[21] P. Sharma, P. Kulkarni, and P. Shenoy. Per-vm page cache partitioning for cloud computing platforms. In *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–8, Jan 2016.

[22] Prateek Sharma and Purushottam Kulkarni. Singleton: System-wide page deduplication in virtual environments. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 15–26, New York, NY, USA, 2012. ACM.

[23] Ioan Stefanovici, Eno Thereska, Greg O'Shea, Bianca Schroeder, Hitesh Ballani, Thomas Karagiannis, Antony Rowstron, and Tom Talpey. Software-defined caching: Managing caches in multi-tenant data centers. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 174–181, New York, NY, USA, 2015. ACM.

[24] Carl Waldspurger, Trausti Saemundsson, Irfan Ahmad, and Nohhyun Park. Cache modeling and optimization using miniature simulations. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 487–498, Santa Clara, CA, 2017. USENIX Association.

[25] Carl A. Waldspurger, Nohhyun Park, Alexander Garthwaite, and Irfan Ahmad. Efficient MRC construction with SHARDS. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 95–110, Santa Clara, CA, 2015. USENIX Association.

[26] Jake Wires, Stephen Ingram, Zachary Drudi, Nicholas J. A. Harvey, and Andrew Warfield. Characterizing storage workloads with counter stacks. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 335–349, Broomfield, CO, 2014. USENIX Association.

[27] Theodore M. Wong and John Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference*, ATC '02, pages 161–175, Berkeley, CA, USA, 2002. USENIX Association.

[28] C. Wu, X. He, Q. Cao, and C. Xie. Hint-k: An efficient multi-level cache using k-step hints. In *2010 39th International Conference on Parallel Processing*, pages 624–633, Sept 2010.

[29] Gala Yadgar, Michael Factor, and Assaf Schuster. Karma: Know-it-all replacement for a multilevel cache. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07, pages 25–25, Berkeley, CA, USA, 2007. USENIX Association.

[30] Juncheng Yang. Pymimircache. `https://github.com/1a1a11a/`. Retrieved Dec. 2020.

[31] Yuanyuan Zhou, Zhifeng Chen, and Kai Li. Second-level buffer cache management. *IEEE Trans. Parallel Distrib. Syst.*, 15(6):505–519, June 2004.

[32] Yuanyuan Zhou, James Philbin, and Kai Li. The multi-queue replacement algorithm for second level buffer caches. In *Proceedings of the General Track: 2001 USENIX Annual Technical Conference*, pages 91–104, Berkeley, CA, USA, 2001. USENIX Association.