Can Miss Ratio Curves Predict Cache Performance?

Lorenzo Marini, Damiano Carra
Computer Science Department
University of Verona, Italy
{name.surname}@univr.it

***Abstract*—Miss Ratio Curves (MRCs) serve as a widely recognized tool for cache profiling, offering a comprehensive visualization of the relationship between cache size and miss ratio. This enables users to assess the cost of storage and estimate the impact of cache misses effectively. However, MRCs are constructed based on past requests, raising questions about their suitability for *what-if* analysis in predicting future cache occupancy.**

In this work, we aim to evaluate the effectiveness of MRCs in predicting cache performance. To achieve this, we explore the influence of the interval during which requests are collected and establish metrics for quantifying the error between predictions and actual outcomes. Our findings highlight that predictive *what-if* analysis remains an open problem that necessitates thorough investigation and exploration.

I. Introduction

Caches play a pivotal role across various computing contexts, acting as foundational components to enhance system performance. From CPUs to web services, caches facilitate rapid access to frequently used content, becoming increasingly indispensable as system architectures become more complex, such as those incorporating multi-core CPUs or clusters of parallel machines. One of the critical aspects in leveraging caches for performance improvement lies in determining the optimal cache size. Even minor deviations in cache hits can lead to substantial variations in average latency required to retrieve content [1]. However, the relationship between cache hit ratio and allocated resources is not linear, *e.g.*, doubling the cache size does not necessarily result in a doubled hit ratio.

In cloud computing, cache sizing is particularly important in two scenarios. Firstly, in a shared resource environment where caches are utilized by multiple processes or applications, specific access patterns or aggressive applications may lead to cache misuse [2], [3], thereby adversely affecting the performance of other applications. Here, determining the appropriate cache space allocation is essential to ensure fair resource sharing without compromising application performance. Secondly, in-memory key-value stores used as caches are offered as managed platforms with a pay-as-you-go model. Amazon's ElastiCache [4], Microsoft's Azure Redis Cache [5] and Google's Cloud Memorystore [6] are examples of caches that employ popular open source software such as Memcached [7] or Redis [8]. In this case, the choice of cache resources directly influences user costs.

Both scenarios necessitate a tool for *what-if* analysis, capable of quantifying the hit ratio a given application would achieve with varying cache space allocations, enabling users to evaluate the trade-off between costs and performance. Many works in the literature propose Miss Ratio Curves (MRCs) as such a tool, providing miss ratios (the complement to one of the hit ratio) as a function of cache size [9]. However, MRCs are constructed based on past requests, and while most works focus on the computational aspects of building MRCs, they implicitly assume that MRCs can accurately predict cache utilization. This assumption holds if the traffic pattern does not vary. In dynamic contexts, in which the popularities of the items changes over time, it is not clear if MRCs can be indeed used as a predicting tool.

In this paper we investigate how effective MRCs are in predicting the cache size to adopt. To this aim, we need to consider different challenges, which were overlooked by previous works. MRCs are built from a set of requests collected over an interval of time, but what should be the duration of such an interval? Small intervals may be able to follow the changes in the traffic pattern, but the MRCs built with few requests may be noisy. Larger intervals, on the other hand, may average out the benefits of short bursts of high hit ratios. Additionally, how can we measure the quality of the predictions provided by MRCs? The comparison of two MRCs built for two consecutive intervals can be done from the miss ratio viewpoint (for a given target miss ratio, what is the difference between the predicted cache size and the actual cache size?), or from a cache size viewpoint (for a given target cache size, what is the difference between the predicted miss ratio and the actual miss ratio?), and the computations of these two viewpoints may not be straightforward.

We consider a set of publicly available traces and show some representative cases in which MRCs are indeed able to provide predictions with a low or limited error, and cases in which the error varies significantly, regardless of the duration of the interval used to build the MRCs. Our results pinpoint that predictive *what-if* analysis is still an open problem that requires to be investigated in depth, resorting to different techniques to complement MRCs.

The remainder of the paper is organized as follows. In Sec. II, we provide background information and discuss related work. In Sec. III, we present the methodology, while in Sec. IV we provide the preliminary analysis. We discuss our results in Sec. V, and we conclude the paper in Sec. VI.

II. Background and Related Work

Miss Ratio Curves. The initial procedure for computing the Miss Ratio Curve (MRC) [10] has been tailored for cases where all items have the same size, effectively allowing the cache size to correspond to the number of items it can store. Additionally, the eviction policy satisfies the inclusion property, where the set of contents stored in the cache at any given time is a subset of the contents that would be stored if the cache had a larger size. Commonly adopted policies such as Least Recently Used (LRU), Least Frequently Used (LFU), and Most Recently Used (MRU) satisfy this property, rendering MRCs valuable in numerous practical systems.

For a designated time frame T_i, MRCs can be computed at the conclusion of the period. The caching policy maintains an ordered list of the cache's contents, with the last item in the list being the current candidate for eviction at any given moment. The MRC algorithm processes requests by managing an ordered list $\mathcal{R}$ of references to items, emulating how the corresponding caching policy would operate if the cache size were infinite. Upon receiving a request for item j, the algorithm determines its current position, also known as the *reuse distance*, and updates an empirical histogram of reuse distances accordingly. Once the time period T_i concludes, the histogram is normalized by dividing each value by the total number of requests received within T_i. The integral of the normalized histogram yields the hit ratio curve, where the complement to one represents the MRC.

Exact computation of MRCs necessitates $\mathcal{O}(N)$ memory, where N denotes the number of items in the catalog, with a complexity of $\mathcal{O}(\log N)$ per request due to accessing $\mathcal{R}$, which can be implemented using a tree data structure [11]. To mitigate computational complexity, various approximate solutions based on sampling have been proposed [12], [13], [14]. MRCs can be constructed even in scenarios involving items with different sizes [15], and for eviction policies that do not adhere to the inclusion property. However, this necessitates an alternative approach, such as Miniature Simulations [14], [16]. In our experiment, we opt for exact MRC computation, as our primary focus is to evaluate whether the MRC at interval T_i can effectively predict the hit ratio in the subsequent interval T_{i+1}. The analysis of the trade-off between MRC accuracy and computational efficiency, an issue appeared also in other types of studies [17][18], is left for future investigation.

Elastic on-demand services. Cloud computing facilitates the dynamic instantiation of services based on fluctuating traffic volumes. For instance, in web architectures, the number of web servers can be adjusted to accommodate increasing traffic levels. Among the various services offered by cloud providers, in-memory data stores utilized as caches play a significant role. Notable examples include Amazon's ElastiCache [4], Microsoft's Azure Redis Cache [5], and Google's Cloud Memorystore [6].

These managed solutions handle cache-related operations such as software updates and maintenance, offering simple APIs for creating, shutting down instances, and managing the clusters. Users have the flexibility to select from various configurations for each instance. For instance, Amazon's ElastiCache [19] provides options to choose instances with varying RAM sizes and numbers of cores (vCPUs). Additionally, different types of instances are available, including regular, spot, and burstable ones.

Related work. Existing approaches related to cache capacity tuning can be categorized into three main categories. (1) Rule-based approaches [20], [21], [22] react to events such as cache occupancy exceeding or falling below predefined thresholds to adjust the cache size. However, they do not predict future utilization, which may result in suboptimal adjustments. (2) ML-based approaches [23], [24] utilize historical data to train predictive models. Nevertheless, there is no guarantee that future traffic patterns will resemble those in the training set. (3) MRC-based approaches [9], [12], [13], [14], [25] focus on the computational complexity required to construct the Miss Ratio Curve (MRC). These approaches typically consider entire traces, constructing the MRC for all requests within a given trace.

In contrast, our work focuses on studying the predictive power of an MRC built during a single interval and the impact of the interval's length. We adopt a practical approach by considering time intervals and observing the number of requests within these intervals. Given the inherent fluctuations in request volumes, we aim to assess the impact of these fluctuations on the accuracy of predictions.

III. Methodology

We take an experimental approach to highlight the complexity of the issue. We consider some representative traces with different characteristics that shows the strengths and the limitation of an analysis based on MRCs.

Traces. We consider a set of publicly-available block I/O traces from SNIA IOTTA repository [26], and a Content Delivery Network (CDN) trace from [27], [28]. Table I summarizes the characteristics of the traces. From the SNIA IOTTA repository, we have considered the most recent trace—labeled as `systor` [29]—along with older traces collected at Microsoft [30], labeled as `ms-ex`. The `cdn` trace refers to a CDN cache serving photos and other media content for Wikipedia pages (21 days, collected in 2019).

TABLE I: Trace characteristics: catalog size (N), number of requests in the trace (M), and duration (T_{tot})

name	year	N	M	T_{tot}	reference
`ms-ex`	2007	$21.5 \cdot 10^6$	$61.2 \cdot 10^6$	24 h	[30]
`systor`	2016	$12.7 \cdot 10^6$	$34.3 \cdot 10^6$	12 h	[29]
`cdn`	2019	$7.2 \cdot 10^6$	$42.6 \cdot 10^6$	10 h	[27]

For experimental reproducibility, we report here the details of the subtrace we use. The `ms-ex` is the trace taken from [30] named "Microsoft Enterprise Traces, Exchange Server Traces," which have been collected for Exchange server for a duration of 24-hours. The `systor` traces [29] collect requests

for different block storage devices over 28 days: we consider 12 hours (March, 9th) of the device called "LUN2." The `cdn` trace contains multiple days of traffic, of which we consider portions of 10 hours – we tested different intervals finding similar qualitative results.

Among the various pieces of information contained in the traces, we primarily focus on two fields: the request timestamp and the item identifier. The timestamp enables us to analyze scenarios involving time-based observation windows (*e.g.*, 30 minutes or one hour). Our results primarily center on cases where all items have the same size, as this already demonstrates the complexity of the problem. We delve into the consideration of items with different sizes in Sec. VI.

Sliding vs Decaying vs Non-overlapping windows. The process of building Miss Ratio Curves (MRCs) is typically described based on a set of past requests. However, in the online scenario we consider, where a single MRC is constructed based on requests from the last interval T, we need to specify the exact process. One possibility is to maintain a data structure that facilitates the computation of the MRC using a sliding window approach. In this case, MRCs are not computed at predetermined time steps (*e.g.*, at T, $2T$, $3T$, ...), but the computation can be triggered at any moment t by considering requests collected during the interval $[t-T, t]$. This flexibility comes with the cost of increased memory usage. The reuse distance histogram must track timestamps for each request contributing to different distance values. In practice, memory usage would be proportional to the number of requests received in the last interval T, rather than proportional to the catalog size as in basic MRC computation. Additionally, each time the user computes the MRC, they must normalize the reuse distance histogram with the total number of requests received so far. This normalization is an expensive operation that can only be amortized after receiving a sufficient number of requests. Overall, while a sliding window approach offers instantaneous monitoring of the MRC, it comes with high memory usage and computational complexity.

To reduce the memory requirements of the sliding window approach, another option is to use approximate counting techniques such as DGIM [31] or exponentially decaying windows [32]. These methods allow tuning the number of bits needed to store the approximate count of different reuse distances and the accuracy of the counting itself. However, they work using the last B number of requests rather than the last interval T. Given that billing is based on temporal intervals and the number of requests in each interval can fluctuate significantly, implementing this approach may not be straightforward.

Alternatively, we can consider non-overlapping intervals. After computing the MRC at the end of each interval, we reset all counters (reuse distance histogram, number of requests received). It is worth noting that we do not reset the ordered list of references $\mathcal{R}$ (see Sec. II) representing the current cache state, as it reflects the current cache contents. This approach allows the MRC in the next interval to reflect the requests received during that interval along with the initial cache state, which was not empty. Another advantage of non-overlapping intervals is the ability to dynamically adjust the interval size. For example, if traffic pattern variability is detected and the current pattern is identified as stable, we can change the interval size from $T_i = 1$ hour to $T_{i+1} = 1.5$ hours, thereby saving computational resources.

In the results presented in Sec. V, we use a non-overlapping interval approach and test different fixed-size intervals to assess the benefits of dynamic interval adaptation.

How to measure the prediction error. Given an MRC, we can utilize it in two ways. The first approach involves setting a target miss ratio m_i and using the MRC to determine the required cache size C_t to achieve that target at interval t. Subsequently, at the end of the next interval, we compute the actual cache size C_{t+1} for the target miss ratio m_i, and the error is calculated as $\frac{|C_{t+1}-C_t|}{C_t}$. It is important to note that the absolute value is used in the numerator since both over-provisioning and under-provisioning represent errors: a larger cache size indicates spending more money than necessary, while a smaller cache size leads to more misses. We can compute this error for a set of discrete values of m_i and then take the average, a metric known as *Mean Absolute Error* (MAE). However, a potential issue with this approach arises when the difference $|C_{t+1} - C_t|$ does not exist for a given m_i, as one MRC may terminate at a higher miss ratio (refer to Figure 1, left).

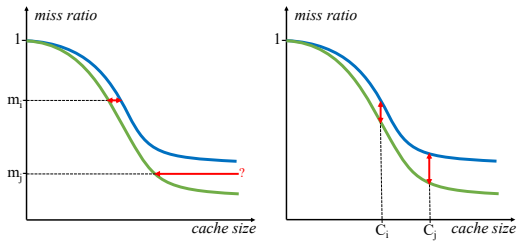

Fig. 1: Two ways of calculating the error between two MRCs computed in two consecutive intervals.

The second approach involves considering a cache size C_i and determining the corresponding miss ratio m_t at interval t. Subsequently, at the end of the next interval, we compute the actual miss ratio m_{t+1} and the error $\frac{|m_{t+1}-m_t|}{m_t}$. We calculate this error for all possible sizes C_i and take the average. In this case, the difference at the numerator always exists because the MRC is the integral over all values of the cache size, and we can extend the MRC with a constant value to any cache size (see Figure 1, right).

While the first approach may seem easier to understand as it reflects the question "What is the best cache size if someone would like to obtain a desired miss ratio?", the second approach is simpler to compute and provides an indication of the error in terms of miss ratio for a given budget. In Sec. V, we demonstrate that this approach offers sufficient insights into the predictive power of MRCs.

IV. On the interval length

The choice of the interval length T_i used to compute the MRC should depend on the traffic characteristics, such as the number of requests and the reuse distance. A stable traffic pattern allows for the use of larger intervals, while highly dynamic patterns require more frequent evaluations of the MRC. In this section, we demonstrate that selecting the right interval is indeed a challenging task.

We analyze the instantaneous hit ratio over time for different cache sizes C and the instantaneous number of requests. The `ms-ex` trace, depicted in Figure 2, remains relatively stable over a period of hours, with the exception of a peak of requests at approximately $t = 8$ hours (see Figure 2, left). During this peak, the hit ratio experiences a significant drop regardless of the cache size. Since the duration of the peak is limited, on average, a larger interval used to compute the MRC (*e.g.*, 1 hour) should be able to accommodate such variability.

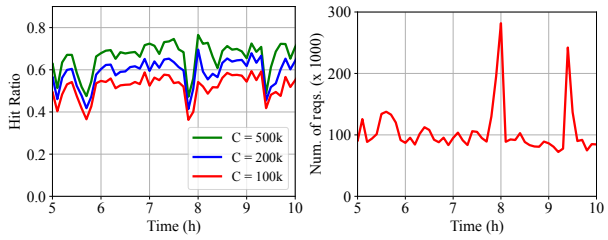

Fig. 2: `ms-ex` trace, instantaneous hit ratios over time for different cache sizes C (left) and instantaneous number of requests over time (right).

The `systor` trace, depicted in Figure 3, exhibits a completely different behavior. Here, the increase in traffic corresponds to a significant rise in the hit ratio. Additionally, during the peaks, the differences in hit ratio when using different cache sizes almost disappear. This is probably due to a increase in the requestes for the same subset of items. Such a case suggests that, for this trace, a small interval used for MRC computation (*e.g.*, 10 minutes) may be advantageous, as we can decrease the cache size during peaks and still achieve a high hit ratio. A longer interval would average out the high and low hit ratios.

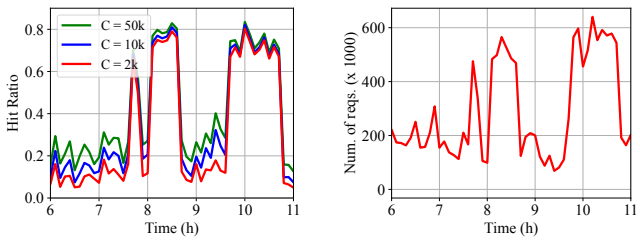

Fig. 3: `systor` trace, instantaneous hit ratios over time for different cache sizes C (left) and instantaneous number of requests over time (right).

A highly predictable scenario is illustrated by the `wiki` trace, shown in Figure 4. Even as the number of requests decreases over time, the hit ratio remains stable for different cache sizes. In this case, large intervals, such as 2-3 hours, are more than sufficient for creating predictive MRCs.

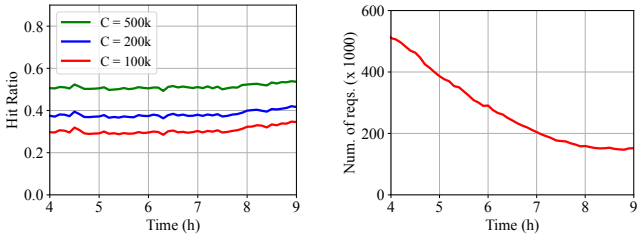

Fig. 4: `wiki` trace,instantaneous hit ratios over time for different cache sizes C (left) and instantaneous number of requests over time (right).

In summary, a simple examination of the instantaneous hit ratio and the number of requests is not sufficient to determine the best interval for MRC computation. The selection of the optimal interval remains an open problem that necessitates specialized tools. In the following section, we validate this observation by using two representative interval sizes and assessing their impact on prediction error.

V. Results and discussion

In this section, we assess the impact of two different interval lengths, T, on the MRC error. Figure 5 presents examples of MRCs computed over four consecutive intervals, using a small interval ($T = 10$ minutes) and a larger one ($T = 1$ hour). We conducted this analysis for all intervals throughout the duration of the trace, finding similar results.

For this trace, a smaller interval appears to yield a larger error. For instance, the MRCs computed in intervals 1 and 2 differ for both small and large T, whereas those computed in intervals 2 and 3 mostly overlap. Although we can still observe similar shapes in the MRCs, the error in miss ratio can reach up to 20%, increasing from 0.6 to 0.72.

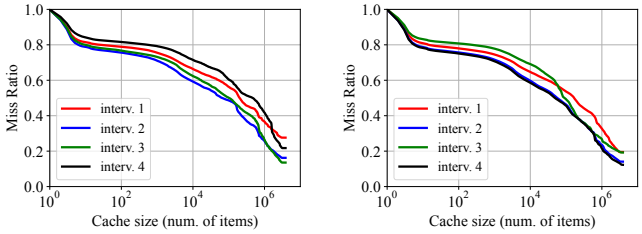

Fig. 5: `ms-ex` trace, MRC computed over 4 consecutive intervals. The duration of the interval is 10 minutes (left) and 1 hour (right).

To assess the variability of the error, we compute the Mean Absolute Error (MAE) by scanning the entire trace using the two different intervals. Figure 6 depicts the MAE between two

consecutive MRCs. The graph confirms that smaller intervals result in higher error variability, while larger intervals yield smaller errors overall. Notably, with a smaller interval, there is a peak in the error corresponding to the peak of requests shown in Figure 2 (right). Since the duration of the peak aligns with the interval used for MRC computation, it is clear that the MRC fails to predict such peaks.

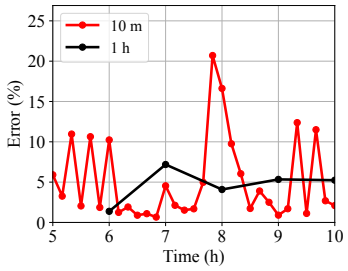

Fig. 6: `ms-ex` trace, MAE computed using consecutive MRCs.

On the other hand, the `systor` trace exhibits high variability for both short and long intervals T used to compute the MRCs (Figure 7). In this scenario, predicting the required cache size based on any MRCs proves challenging. The varying MRCs correspond to intervals with high or low traffic, resulting in vastly different cache profiles. This observation is supported by the fluctuating MAE depicted in Figure 8. The mean error fluctuates between 5% and 25% for both 10-minute and 1-hour intervals. Even employing a larger interval for MRC computation fails to mitigate the error, illustrating the difficulty of the prediction problem.

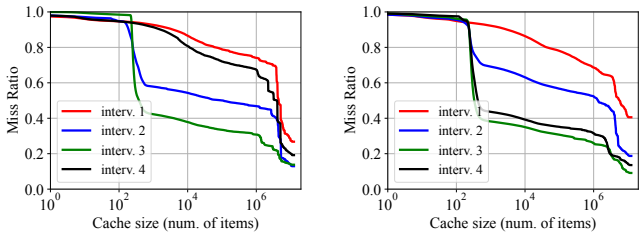

Fig. 7: `systor` trace, MRC computed over 4 consecutive intervals. The duration of the interval is 10 minutes (left) and 1 hour (right).

In the `wiki` trace, characterized by a stable traffic pattern, the interval length used to construct the MRCs does not affect the error significantly (Figure 9, left), as anticipated from the consistent hit ratios observed with various cache sizes (Figure 4, left). With both small and large intervals T, the error remains below 1% (Figure 9, right).

Interestingly, despite the small error, using a smaller interval T for MRC computation results in more pronounced variability. This suggests that employing excessively small intervals may introduce noise, and closely tracking traffic variability might not be optimal. Conversely, a larger interval provides an averaging effect that proves beneficial in this scenario.

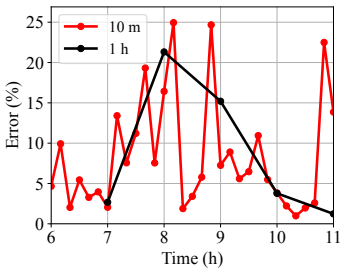

Fig. 8: `systor` trace, MAE computed using consecutive MRCs.

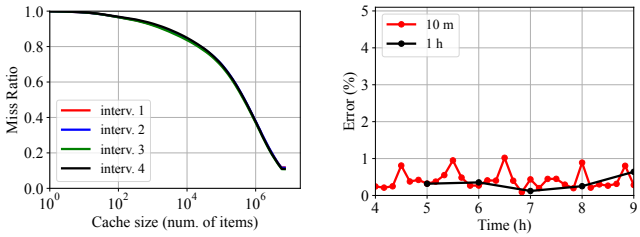

Fig. 9: `wiki` trace, MRC computed over 4 consecutive 10-minute intervals (left), and MAE computed using consecutive MRCs (right).

VI. Conclusion and Perspectives

Predicting cache performance, *i.e.*, anticipating the hit ratio for different cache sizes, poses a significant challenge due to its dependence on traffic patterns and caching policies. In this study, we focused on scenario utilizing LRU eviction policy and uniform item sizes. Our analysis of three publicly available cache traces revealed that MRCs, a commonly used tool for cache profiling, may yield predictions with substantial errors.

While existing literature offers solutions for determining optimal cache sizes, these approaches typically focus on identifying a single best size. Consequently, they cannot support *what-if* analysis or joint sizing of multiple caches for different applications, which are crucial in scenarios where caches are shared among multiple applications and the allocation for one application affects the availability for others.

A potential avenue for improvement could involve dynamically adjusting the prediction interval T based on the current error. Larger errors could prompt a reduction in interval size, while smaller errors could warrant an increase. However, as demonstrated in our experimental evaluation, this approach may not always suffice. Therefore, the challenge of accurately predicting cache performance remains an open issue that necessitates exploration of alternative methodologies, a direction we aim to pursue in our future research work.

REFERENCES

[1] A. Cidon, A. Eisenman, M. Alizadeh, and S. Katti, "Dynacache: Dynamic cloud caching," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[2] Q. Pu, H. Li, M. Zaharia, A. Ghodsi, and I. Stoica, "Fairride: Near-optimal, fair cache sharing," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp. 393–406.

[3] A. Sundarrajan, M. Feng, M. Kasbekar, and R. K. Sitaraman, "Footprint descriptors: Theory and practice of cache provisioning in a global CDN," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, 2017, pp. 55–67.

[4] AWS, "Amazon Web Service ElastiCache," https://aws.amazon.com/elasticache/, accessed: Apr. 2024.

[5] Microsoft, "Azure Cache for Redis," https://azure.microsoft.com/en-us/products/cache/, accessed: Apr. 2024.

[6] Google, "Cloud Memorystore," https://cloud.google.com/memorystore/, accessed: Apr. 2024.

[7] Memcached, "Memcached," https://memcached.org/, accessed: Apr. 2024.

[8] Redis, "Redis," https://redis.io/, accessed: Apr. 2024.

[9] T. Saemundsson, H. Bjornsson, G. Chockler, and Y. Vigfusson, "Dynamic performance profiling of cloud caches," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–14.

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

[11] Y. Zhong, X. Shen, and C. Ding, "Program locality analysis using reuse distance," *ACM Trans. Program. Lang. Syst.*, vol. 31, no. 6, pp. 1–39, 2009.

[12] X. Hu, X. Wang, L. Zhou, Y. Luo, C. Ding, and Z. Wang, "Kinetic modeling of data eviction in cache," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 351–364.

[13] C. A. Waldspurger, N. Park, A. T. Garthwaite, and I. Ahmad, "Efficient MRC Construction with SHARDS," in *FAST*, 2015, pp. 95–110.

[14] C. Waldspurger, T. Saemundsson, I. Ahmad, and N. Park, "Cache modeling and optimization using miniature simulations," in *Proceedings of USENIX ATC*, 2017, pp. 487–498.

[15] D. Carra and G. Neglia, "Efficient miss ratio curve computation for heterogeneous content popularity," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 741–751.

[16] K. Shakiba, S. Sultan, and M. Stumm, "Kosmo: Efficient online miss ratio curve generation for eviction policy evaluation," in *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, 2024, pp. 89–105.

[17] S. Migliorini, A. Belussi, M. Negri, and G. Pelagatti, "Towards massive spatial data validation with spatialhadoop," in *Proceedings of the 5th ACM SIGSPATIAL international workshop on analytics for big geospatial data*, 2016, pp. 18–27.

[18] S. Migliorini, D. Carra, and A. Belussi, "Distributing tourists among pois with an adaptive trip recommendation system," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1765–1779, 2019.

[19] AWS, "Amazon Web Service ElastiCache Pricing," https://aws.amazon.com/elasticache/pricing/, accessed: Apr. 2024.

[20] A. Wang, J. Zhang, X. Ma, A. Anwar, L. Rupprecht, D. Skourtis, V. Tarasov, F. Yan, and Y. Cheng, "{InfiniCache}: exploiting ephemeral serverless functions to build a {cost-effective} memory cache," in *18th USENIX conference on file and storage technologies (FAST 20)*, 2020, pp. 267–281.

[21] D. Carra, G. Neglia, and P. Michiardi, "Elastic provisioning of cloud caches: A cost-aware ttl approach," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1283–1296, 2020.

[22] F. Romero, G. I. Chaudhry, Í. Goiri, P. Gopa, P. Batum, N. J. Yadwadkar, R. Fonseca, C. Kozyrakis, and R. Bianchini, "Faa$t: A transparent auto-scaling cache for serverless applications," in *Proceedings of the ACM symposium on cloud computing*, 2021, pp. 122–137.

[23] D. Mvondo, M. Bacou, K. Nguetchouang, L. Ngale, S. Pouget, J. Kouam, R. Lachaize, J. Hwang, T. Wood, D. Hagimont *et al.*, "Ofc: an opportunistic caching system for faas platforms," in *Proceedings of the Sixteenth European Conference on Computer Systems*, 2021, pp. 228–244.

[24] G. Rattihalli, M. Govindaraju, H. Lu, and D. Tiwari, "Exploring potential for non-disruptive vertical auto scaling and resource estimation in kubernetes," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 33–40.

[25] R. Gu, S. Li, H. Dai, H. Wang, Y. Luo, B. Fan, R. B. Basat, K. Wang, Z. Song, S. Chen *et al.*, "Adaptive online cache capacity optimization via lightweight working set size estimation at scale," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 467–484.

[26] SNIA, "SNIA iotta repository block I/O traces," http://iotta.snia.org/traces/block-io, accessed: Apr. 2024.

[27] Z. Song, D. S. Berger, K. Li, A. Shaikh, W. Lloyd, S. Ghorbani, C. Kim, A. Akella, A. Krishnamurthy, E. Witchel *et al.*, "Learning relaxed belady for content distribution network caching," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 529–544.

[28] Z. Song et at., "CDN traces," https://github.com/sunnyszy/lrb, accessed: Apr. 2024.

[29] C. Lee, T. Kumano, T. Matsuki, H. Endo, N. Fukumoto, and M. Sugawara, "Understanding storage traffic characteristics on enterprise virtual desktop infrastructure," in *Proceedings of the 10th ACM International Systems and Storage Conference*, ser. SYSTOR '17. ACM, 2017, pp. 13:1–13:11.

[30] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production Windows servers," in *2008 IEEE International Symposium on Workload Characterization*. IEEE, 2008, pp. 119–128.

[31] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM journal on computing*, vol. 31, no. 6, pp. 1794–1813, 2002.

[32] E. Cohen and M. Strauss, "Maintaining time-decaying stream aggregates," in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2003, pp. 223–233.