

On Hierarchical Caching with Delayed Hits

Kanghuai Liu
Nanyang Technological University
Singapore
kanghuai.liu@ntu.edu.sg

Xueyan Tang
Nanyang Technological University
Singapore
asxytang@ntu.edu.sg

Abstract—Designing caching algorithms to minimize aggregate latency is a long-standing challenge in memory-intensive and latency-sensitive computer systems. This challenge is further exacerbated by a recently discovered phenomenon, known as *delayed hits*, which renders traditional optimal algorithms unable to guarantee minimal latency. To bridge this gap, this paper pioneers a comprehensive algorithmic analysis for hierarchical caching with delayed hits. First, we analytically establish a competitiveness lower bound for any online caching algorithm. Second, we introduce a novel online algorithm that synthesizes two key technical innovations: an enhanced marking policy that executes separate marking processes for different cache layers, and a heuristic ranking function that leverages historical request patterns to prioritize items based on their estimated aggregate latency per cache miss. Finally, we theoretically prove that the proposed algorithm can asymptotically match the lower bound under some mild conditions. Extensive experiments validate its effectiveness and superiority over state-of-the-art solutions.

I. INTRODUCTION

Caching is a key optimization technique widely employed in memory-intensive and latency-sensitive computer systems to accelerate data retrieval [1]. Traditional caching architectures typically adopt a single-layer design, where a single cache sits between a backing store and client requests [2]. In contrast, modern systems increasingly employ hierarchical caching architectures [3], [4], where multiple cache layers are tailored to different storage capacities and access latencies, thereby better accommodating heterogeneous access patterns and stringent latency requirements in today’s workloads. Such hierarchical architectures are now commonplace, for instance:

- Mobile edge caching (MEC) systems deploy caches across multiple layers from the cloud to edge servers, caching more frequently requested content at layers closer to end users and hence reducing overall data retrieval latency [5], [6].
- Edge-based content delivery networks (CDN) organize origin, core, and edge caches into a hierarchy to accommodate diverse caching requirements, thereby enabling low-latency content delivery at scale [7], [8].

A fundamental problem arising in these hierarchical caching systems is the design and analysis of online caching policies to minimize aggregate latency. In its generic form, a backing store holds the complete set of distinct data items, fronted by two cache layers: a core cache with larger capacity and non-negligible access latency, and an edge cache with much smaller capacity and zero access latency [7], [8]. For each request arriving at the edge cache, there are traditionally two possible outcomes: a low-latency cache hit or a higher-latency cache miss. The goal is to design an online caching algorithm that

minimizes the cumulative latency incurred over all incoming requests, which is equivalent to maximizing hit rate, a long-standing challenge in the literature [9]–[13].

Recently, a new possible outcome for requests known as *delayed hits* [1] has been discovered to significantly affect caching performance, rendering the problem even more non-trivial. Specifically, delayed hits occur at the edge cache when multiple requests for the same item queue up before the item is retrieved from the core cache or the backing store. Under delayed hits, the traditional caching objective of maximizing hit rate no longer aligns with latency minimization, potentially rendering conventional optimal caching algorithms suboptimal [1], [14]–[16]. *We therefore argue that delayed hits reveal a significant gap in the current understanding and practice of caching, underscoring the urgent need for in-depth theoretical and algorithmic investigations to address the associated challenges.*

Despite the profound impact of delayed hits on caching, research on this phenomenon remains limited [1], [14]–[19], and hardly any work considers hierarchical caches. In this paper, we pioneer a generic algorithmic analysis for hierarchical caching with delayed hits, making three main contributions:

- We establish a lower bound of $\Omega(\frac{(K_1+K_2)Z^2}{Z+K_2W})$ on the competitive ratio of *any* deterministic online caching algorithm for hierarchical caching with delayed hits, where K_1 and K_2 are the storage capacities of the edge and core caches, respectively, and W and Z are the latencies of retrieving an item from the core cache and the backing store, respectively.
- Motivated by the construction of this bound, we develop a novel online caching algorithm, termed Dual Marking (DM), which augments the classical marking policy with heuristic predictions. Specifically, DM incorporates two key techniques: (i) a dual marking policy that executes two separate marking processes, one for the edge cache and the other for the core cache; and (ii) a heuristic ranking function that leverages historical request patterns to prioritize items based on their estimated aggregate latency per cache miss.
- We theoretically prove that DM achieves a competitive ratio of $\mathcal{O}(\max\{K_1Z, \frac{Z^2}{W}\})$, and show that it asymptotically matches the lower bound derived under some mild conditions. An experimental evaluation on three real-world datasets demonstrates that DM outperforms state-of-the-art solutions by at least 15% on average in terms of aggregate latency.

We believe that our insights make an important step towards a deeper theoretical understanding of delayed hits so as to guide the design of practical caching policies with perfor-

mance guarantees.

II. PRELIMINARIES

A. System Model

As illustrated in Fig. 1, we consider a generic hierarchical caching system consisting of a backing store, a core cache C_c , and an edge cache C_e . The core cache acts as an intermediary between the backing store and the edge cache. The backing store holds a set $\mathcal{D} = \{d_j\}_{j=1}^N$ of N distinct data items, each normalized to unit size. The cache capacities of C_e and C_c are K_1 and K_2 , respectively, with $K_2 \gg K_1 > 0$ and $N \gg K_1 + K_2$. Without loss of generality, we assume that the system operates in discrete time steps. For the edge cache C_e , retrieving an item from the core cache C_c incurs a latency of W time steps, while fetching it from the backing store through C_c requires Z time steps. Hence, it takes $Z - W$ time steps for the core cache C_c to retrieve an item from the backing store. In practice, since the backing store is typically deployed far from both caches, it follows that $Z \gg W > 0$. Due to computational jobs at the edge or service demands from nearby users, requests to access data items from the edge cache C_e arise over time, with each request targeting a specific item in the set $\mathcal{D}$. At any time step t , the contents of the edge cache C_e are currently $C_e(t-1)$, while those of the core cache C_c are $C_c(t-1)$. C_e receives a request $r(t) \triangleq d_j$ seeking to access a data item $d_j \in \mathcal{D}$.

- 1) *Cache hit*. If $d_j \in C_e(t-1)$, the edge cache C_e undergoes a cache hit with latency 0.
- 2) *Cache miss*. If $d_j \notin C_e(t-1)$ and the service for the most recent miss for d_j has completed, C_e experiences a cache miss, classified into two cases:
 - *W-miss*. If $d_j \in C_c(t-1)$, C_e serves $r(t)$ by fetching d_j from the core cache C_c in W time steps, incurring a W -miss with latency W . This circumvents the higher latency of accessing the backing store.
 - *Z-miss*. If $d_j \notin C_c(t-1)$, C_e must retrieve d_j from the backing store through the core cache C_c , resulting in a Z -miss with latency Z .
- 3) *Delayed hit*. If $d_j \notin C_e(t-1)$, and the most recent miss for d_j occurred some X time steps ago and is still being processed at time step t , C_e encounters a delayed hit, and there are two cases:
 - *W-delayed hit*. If the most recent miss for d_j was a W -miss, d_j will arrive at C_e in $W - X$ time steps. Thus, $r(t)$ incurs a W -delayed hit with latency $W - X$.
 - *Z-delayed hit*. Analogously, if the most recent miss for d_j was a Z -miss, $r(t)$ incurs a Z -delayed hit with latency $Z - X$.

At any time step t , once a fetched item d_j arrives at the edge cache C_e , it is used to serve all pending requests for that item, including both cache misses and delayed hits. Subsequently, a caching algorithm must determine whether to cache d_j .

- 1) If the decision is to cache d_j , there are two cases:
 - If d_j is retrieved from C_c , the algorithm must decide which item (e.g., d_v) to evict from C_e , i.e., setting $C_e(t) \leftarrow C_e(t-1) \cup \{d_j\} \setminus \{d_v\}$.

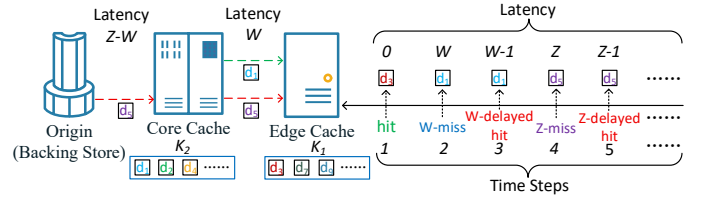


Fig. 1: Hierarchical caching framework under delayed hits.

- If d_j is fetched from the backing store through C_c , then d_j first arrives at C_c at time step $t - W$ and subsequently arrives at C_e at time step t . Just before d_j arrives at C_e , the contents of C_c are $C_c(t - W - 1)$. The algorithm may accommodate it in the following ways:
 - Caching it only in C_e by evicting an item d_v from C_e , i.e., setting $C_e(t - W) \leftarrow C_e(t - W - 1)$ and $C_e(t) \leftarrow C_e(t - 1) \cup \{d_j\} \setminus \{d_v\}$;
 - Caching it only in C_c by evicting an item d_a from C_c , i.e., setting $C_c(t - W) \leftarrow C_c(t - W - 1) \cup \{d_j\} \setminus \{d_a\}$ and $C_e(t) \leftarrow C_e(t - 1)$;
 - Caching it in both C_e and C_c by evicting an item d_v from C_e and an item d_a from C_c , i.e., setting $C_c(t - W) \leftarrow C_c(t - W - 1) \cup \{d_j\} \setminus \{d_a\}$ and $C_e(t) \leftarrow C_e(t - 1) \cup \{d_j\} \setminus \{d_v\}$.
- 2) Alternatively, if the decision is not to cache d_j , both C_c and C_e remain unchanged, i.e., setting $C_c(t - W) \leftarrow C_c(t - W - 1)$ and $C_e(t) \leftarrow C_e(t - 1)$. This means that our system setting enables bypassing, a feature commonly found in classical caching models [14].

Fig. 1 illustrates an example of our system model. A request sequence accesses data items from the edge cache C_e , where $r(1) \triangleq d_3$, $r(2) \triangleq d_1$, $r(3) \triangleq d_1$, $r(4) \triangleq d_5$, and $r(5) \triangleq d_5$. At time step 1, item d_3 resides in C_e , so request $r(1) \triangleq d_3$ has a cache hit with latency 0. At time step 2, item d_1 is not stored in C_e , but it is stored in the core cache C_c . Hence, C_e fetches d_1 from C_c , and request $r(2) \triangleq d_1$ incurs a W -miss with latency W . As d_1 will arrive at C_e at time step $2 + W$, the subsequent request $r(3) \triangleq d_1$ results in a W -delayed hit with latency $W - (3 - 2) = W - 1$. Upon the arrival of d_1 , a caching algorithm may choose to store it by evicting an item from C_e . At time step 4, as neither C_c nor C_e has cached d_5 , C_e must fetch it from the backing store through C_c , which will arrive at time step $4 + Z$. Hence, request $r(4) \triangleq d_5$ has a Z -miss with latency Z , and the following request $r(5) \triangleq d_5$ incurs a Z -delayed hit with latency $Z - (5 - 4) = Z - 1$. Upon the arrival of d_5 , if the algorithm decides to cache it, it must determine where to store d_5 (in C_c , C_e , or both) and which item in the corresponding cache to evict.

B. Problem Formulation

To formulate our problem, we begin with two definitions.

Definition 1 (Request sequence). A request sequence σ is a finite sequence $\sigma \triangleq [r(t)]_{t=1}^T$ over T time steps. If $r(t) \triangleq d_j$, item d_j is requested at time step t . If $r(t) \triangleq \emptyset$, no item is

requested (i.e., no request arises) at time step t . For simplicity, we refer to $r(t) \triangleq \emptyset$ as an empty request.

Definition 2 (Eviction policy). An eviction policy E is a finite sequence $E \triangleq [e_e(t), e_c(t)]_{t=1}^T$ over T time steps, where $e_e(t)$ and $e_c(t)$ denote the sets of items evicted from C_e and C_c at time step t , respectively. If $e_e(t) \triangleq \emptyset$, no item is evicted from C_e at time step t . Similarly, if $e_c(t) \triangleq \emptyset$, no item is evicted from C_c at time step t .

Problem 1 (Hierarchical caching under delayed hits). Given a request sequence $\sigma \triangleq [r(t)]_{t=1}^T$, we seek to develop an online caching algorithm $\mathcal{A}$ that outputs an eviction policy $E \triangleq [e_e(t), e_c(t)]_{t=1}^T$ to minimize the cumulative latency of serving all requests in σ .

C. Competitive Ratio

The competitive ratio is known as a pivotal metric for quantifying the performance of an online algorithm. For any request sequence σ , we define $\text{OPT}(\sigma)$ as the cumulative latency incurred by the optimal offline algorithm, and define $\mathcal{A}(\sigma)$ as that incurred by an online algorithm $\mathcal{A}$. The competitive ratio of $\mathcal{A}$ is defined as an asymptotic upper bound on the worst-case ratio between $\mathcal{A}(\sigma)$ and $\text{OPT}(\sigma)$.

Definition 3 (Competitive ratio). The competitive ratio of an online caching algorithm $\mathcal{A}$, denoted as $\text{CR}(\mathcal{A})$, satisfies

$$\text{CR}(\mathcal{A}) \geq \frac{\mathcal{A}(\sigma)}{\text{OPT}(\sigma)}$$

for every request sequence σ .

III. LOWER BOUND ON COMPETITIVE RATIO

We establish in Theorem 1 a lower bound on the competitive ratio of any deterministic online caching algorithm.

Theorem 1. The competitive ratio of any deterministic online caching algorithm is $\Omega\left(\frac{(K_1+K_2)Z^2}{Z+K_2W}\right)$.

Remark. The established lower bound converges to $\Omega((K_1+K_2)Z)$ as W approaches 0. This observation is consistent with the results derived for the single-cache (single-layer) model under delayed hits [14]. Our result generalizes the previous bound to the hierarchical setting.

We prove Theorem 1 by constructing an adversarial request sequence σ that minimizes the aggregate latency incurred by the optimal offline algorithm OPT while maximizing the aggregate latency incurred by any online algorithm $\mathcal{A}$. We begin by examining the structure of σ :

- **OPT:** If $K_1 + K_2 + 1$ distinct items are requested in σ , then OPT , having full knowledge of σ , can choose to optimize latency by storing the $K_1 + K_2$ most frequently requested items in the core cache C_c and the edge cache C_e .
- **$\mathcal{A}$:** To maximize the cumulative latency incurred by algorithm $\mathcal{A}$, σ should ensure that $\mathcal{A}$ experiences a cache miss for every request. Once a miss occurs for a data item, σ continues to request that item successively until it is eventually stored in C_e or C_c .

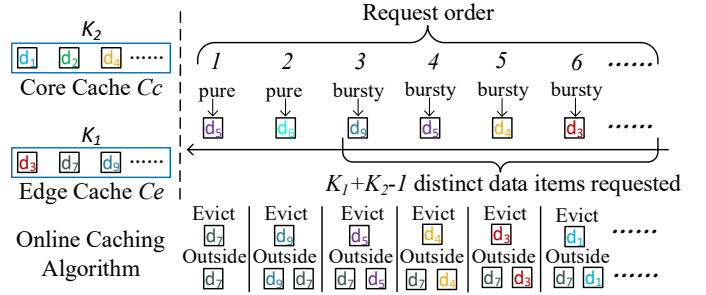


Fig. 2: A request sequence σ constructed in our proof.

With these observations, we define two categories of requests:

- **Pure request.** A pure request for a data item d_j is a sequence of requests in the form of $(\emptyset^Z, d_j^1, \emptyset^Z)$, where the notation d^q represents that data item d is requested q time steps in a row. Intuitively, if a pure request results in a W -miss, the latency accrued is W ; if it incurs a Z -miss, the latency accrued is Z .
- **Bursty request.** A bursty request for an item d_j is in the form of $(\emptyset^Z, d_j^1, \emptyset^{W-1}, d_j^{Z-W}, \emptyset^Z)$. If a bursty request incurs a W -miss, the latency accrued is W . Alternatively, if it results in a Z -miss, the latency accrued is $Z + (Z - W) + (Z - W - 1) + \dots + 1 = Z + \frac{(Z-W)(Z-W+1)}{2}$, corresponding to one Z -miss followed by $(W - 1)$ successive empty requests and $(Z - W)$ consecutive Z -delayed hits.

Let $t^P = 2Z + 1$ and $t^B = 3Z$ denote the lengths of a pure request and a bursty request, respectively. Given any deterministic online caching algorithm $\mathcal{A}$, we construct an adversarial request sequence σ utilizing these two categories of requests through a marking process. This process maintains a set $\mathcal{M}$ of marked data items. Initially, $\mathcal{M} \triangleq \emptyset$. Whenever a data item is requested in a bursty request, it is added to $\mathcal{M}$. The process terminates when $|\mathcal{M}|$ reaches $K_1 + K_2 - 1$.

Sequence construction. Without loss of generality, assume that both the core cache C_c and the edge cache C_e are initially full and store disjoint contents. We start σ with a pure request for a data item d_f that is not stored in the initial caches of C_c and C_e , i.e., $d_f \notin C_c(0) \cup C_e(0)$. At time step $t^P + 1$, we append another pure request for a different item d_s that is also absent from the initial caches of C_c and C_e , i.e., $d_s \notin C_c(0) \cup C_e(0)$. Since the total cache capacity of C_c and C_e is $K_1 + K_2$, while $|C_c(0) \cup C_e(0) \cup \{d_f, d_s\}| = K_1 + K_2 + 2$, there are always at least two data items in the set $C_c(0) \cup C_e(0) \cup \{d_f, d_s\}$ that are not stored in the caches of C_c and C_e by algorithm $\mathcal{A}$. Subsequently, we append a bursty request for an item in $C_c(0) \cup C_e(0) \cup \{d_f, d_s\}$ not stored in the caches of C_c and C_e at time step $2t^P + 1$ (i.e., not in $C_c(2t^P) \cup C_e(2t^P)$). Afterwards, we append another bursty request for an item in $C_c(0) \cup C_e(0) \cup \{d_f, d_s\}$ not stored in the caches of C_c and C_e at time step $2t^P + t^B + 1$ (i.e., not in $C_c(2t^P + t^B) \cup C_e(2t^P + t^B)$), and so on. This process terminates when $|\mathcal{M}| = K_1 + K_2 - 1$. Throughout the process, we guarantee that a particular item d_g in the initial content $C_e(0)$ of C_e (i.e., $d_g \in C_e(0)$) is not requested by any bursty request. This

can be ensured because, at any time step, there are always at least two data items in the set $C_c(0) \cup C_e(0) \cup \{d_f, d_s\}$ that are absent from the caches of C_c and C_e . Hence, if d_g is not currently stored in the caches of C_c and C_e , we can always select a different item outside the caches to construct the next bursty request. Conversely, if d_g remains stored in C_e , it is naturally excluded from the next bursty request.

Fig. 2(a) presents an example of σ . We start σ with a pure request for an item d_5 that is absent from the initial caches of C_c and C_e . Upon the pure request for d_5 , an online caching algorithm $\mathcal{A}$ opts to accommodate d_5 into C_e by evicting d_7 . We append another pure request for a different item d_6 that is also absent from the initial caches of C_c and C_e . Upon the pure request for d_6 , if algorithm $\mathcal{A}$ stores d_6 in C_e by evicting d_9 , we then append a bursty request for d_9 , which is added to $\mathcal{M}$. If, in response to the bursty request for d_9 , algorithm $\mathcal{A}$ accommodates d_9 by evicting d_5 from C_e , we further append a bursty request for d_5 , which is also added to $\mathcal{M}$. Similarly, upon the bursty request for d_5 , if algorithm $\mathcal{A}$ stores d_5 by evicting d_4 from C_c , we append a bursty request for d_4 , which is added to $\mathcal{M}$. Upon the bursty request for d_4 , if algorithm $\mathcal{A}$ accommodates d_4 by evicting d_3 from C_e , we subsequently append a bursty request for d_3 , which is again added to $\mathcal{M}$. This process continues until $|\mathcal{M}| = K_1 + K_2 - 1$. Through this construction, every request in σ results in a Z -miss for the online algorithm $\mathcal{A}$, while ensuring that one item in the initial cache of C_e (i.e., item $d_7 \in C_e$) is not requested by any bursty request in σ .

Lower bound analysis. Both algorithms OPT and $\mathcal{A}$ start with identical initial cache contents in the core cache C_c and the edge cache C_e . We first analyze the cumulative latency incurred by OPT. Let F be the total number of bursty requests in σ . As σ terminates when $|\mathcal{M}| = K_1 + K_2 - 1$, it follows that $F \geq K_1 + K_2 - 1$. We show that there is a way for OPT to achieve an aggregate latency of at most $2Z + (F - K_1 + 1)W$. Specifically, since $|\mathcal{M}| = K_1 + K_2 - 1$, there exist three items in the set $C_c(0) \cup C_e(0) \cup \{d_f, d_s\}$ that are not requested in any of the F bursty requests in σ . Recall that $d_g \in C_e$ (e.g., item d_7 in Fig. 2) is one such item, and let d_x and d_y be the other two. Intuitively, for both OPT and $\mathcal{A}$, each of the two pure requests incurs a Z -miss, contributing an aggregate latency of $2Z$. With full knowledge of σ , OPT handles the two pure requests (for items d_f and d_s) as follows:

- If d_x and d_y correspond to the two pure requests (i.e., $d_x \triangleq d_f$ and $d_y \triangleq d_s$, or $d_x \triangleq d_s$ and $d_y \triangleq d_f$), OPT simply bypasses both d_x and d_y , i.e., not storing them in any cache.
- If both d_x and d_y reside in the initial caches of C_c and C_e (i.e., $d_x, d_y \in C_c(0) \cup C_e(0)$), implying that items d_f and d_s are among the $K_1 + K_2 - 1$ distinct items requested in the F bursty requests, OPT accommodates d_f and d_s by evicting d_x and d_y .
- If d_x corresponds to a pure request, while d_y is present in the initial caches of C_c and C_e , OPT bypasses d_x and evicts d_y from the caches of C_c and C_e when serving the other pure request, and vice versa.

The above strategy ensures that, just before serving the first

bursty request in σ , all $K_1 + K_2 - 1$ distinct items requested by the F bursty requests, along with item d_g , are stored in the caches of C_c and C_e . Since d_g resides in C_e , the remaining $K_1 - 1$ items cached in C_e and the K_2 items cached in C_c are exactly the $K_1 + K_2 - 1$ items requested in the F bursty requests. For the remainder of σ , OPT maintains the cache contents of C_c unchanged and consistently stores in C_e its $K_1 - 1$ items requested in the F bursty requests. As a result, C_e retains a single available slot (currently occupied by item d_g) for temporarily accommodating an item in C_c during its bursty request and guaranteeing that the latency accrued from the bursty request is W . Specifically, for each bursty request targeting an item in C_c , C_e temporarily caches the item in this slot by evicting the previously cached one. Under this strategy, each bursty request targeting an item in C_c is handled by fetching it from C_c , resulting in a W -miss, followed by $(W - 1)$ successive empty requests and $(Z - W)$ consecutive cache hits. On the other hand, each bursty request targeting an item in C_e is served locally with zero latency. As σ contains a total of F bursty requests, OPT incurs an aggregate latency of at most $(F - K_1 + 1)W$ for these requests, with $K_1 - 1$ of them served directly from C_e and the remaining $F - K_1 + 1$ served by accessing C_c . Hence, the aggregate latency incurred by OPT under σ is at most $2Z + (F - K_1 + 1)W$.

In contrast, any deterministic online algorithm $\mathcal{A}$ will inevitably experience a Z -miss for every request in σ , because each item requested in σ is selected to be an item not stored in the caches of C_c and C_e at that time. Given that there are a total of F bursty requests in σ , the cumulative latency incurred by $\mathcal{A}$ is $2Z + F(Z + \frac{(Z-W)(Z-W+1)}{2})$. Hence, the competitive ratio $\mathbf{CR}(\mathcal{A})$ of $\mathcal{A}$ under σ is at least

$$\begin{aligned} \mathbf{CR}(\mathcal{A}) &\geq \frac{\mathcal{A}(\sigma)}{\text{OPT}(\sigma)} \geq \frac{2Z + F \left(Z + \frac{(Z-W)(Z-W+1)}{2} \right)}{2Z + (F - K_1 + 1)W} \\ &= \Omega \left(\frac{(K_1 + K_2)(Z - W)^2}{Z + K_2W} \right) = \Omega \left(\frac{(K_1 + K_2)Z^2}{Z + K_2W} \right), \end{aligned}$$

where the asymptotic bound holds because: 1) given that $Z \gg W > 0$, the ratio is monotonically increasing or decreasing with F ; 2) at $F = K_1 + K_2 - 1$, it is $\Theta \left(\frac{(K_1 + K_2)(Z - W)^2}{Z + K_2W} \right) = \Theta \left(\frac{(K_1 + K_2)Z^2}{Z + K_2W} \right)$; and 3) as $F \rightarrow \infty$, it is $\Theta \left(\frac{(Z - W)^2}{W} \right) = \Theta \left(\frac{(K_1 + K_2)(Z - W)^2}{K_2W} \right) = \Theta \left(\frac{(K_1 + K_2)Z^2}{Z + K_2W} \right)$, given that $K_1 = O(K_2)$. Theorem 1 is established.

IV. ALGORITHM DESIGN: DUAL MARKING

A. Algorithm Overview

The construction of the performance bound derived in Theorem 1 reveals two key insights in designing an online caching algorithm to approach or even achieve the bound:

- To cap aggregate latency, the algorithm should prioritize storing the currently requested data item. If the item is not cached, an infinite sequence of requests for it will result in unbounded latency.
- To minimize aggregate latency, the algorithm should strategically prioritize items that potentially incur higher latency.

Motivated by the above design intuitions, our approach to algorithm design, termed Dual Marking (DM), strengthens the classical marking policy with heuristic predictions. It features two key innovative techniques:

- **Heuristic ranking function:** We leverage historical data request patterns to design a heuristic ranking function $\text{RF}(\cdot)$, which estimates the average aggregate latency incurred per miss for a given item, and hence determines the priority of the item for our objective of minimizing aggregate latency.
- **Prediction-based dual marking:** We maintain a dual marking policy consisting of two separate marking processes: one for the edge cache C_e and the other for the core cache C_c . The ranking function $\text{RF}(\cdot)$ is then incorporated to augment this policy by guiding data-eviction decisions.

DM is performed in an *online* manner, i.e., operating based on the request sequence σ available up to the current time step, rather than the complete σ over all time steps.

B. Computation of $\text{RF}(\cdot)$

We introduce two ranking functions: one for prioritizing items available in the backing store, and the other for ranking items available in the core cache C_c . Algorithm 1 provides a unified procedure for computing both ranking functions, using a hyperparameter $\mathcal{H}$ to distinguish between them. When $\mathcal{H} = W$, Algorithm 1 yields the ranking function for items available in C_c ; when $\mathcal{H} = Z$, it computes the ranking function for items available in the backing store. We first demonstrate the computation of the ranking function for items available in C_c , i.e., the case of $\mathcal{H} = W$. To estimate the utility of caching an item d_j in the edge cache C_e , we treat every past request for d_j as either a miss or a delayed hit, and derive its average aggregate latency per cache miss via $\text{RF}(d_j, t, W)$, as given by Algorithm 1 with $\mathcal{H} = W$. Specifically, let t be the current time step and t_l be the time step of the first request for d_j .

- 1) If $t_l = \text{NULL}$ (i.e., d_j has not been requested before t), set $\text{RF}(d_j, t, W) = 0$;
- 2) If $t_l = t$ (i.e., the first request for d_j occurs at t), set $\text{RF}(d_j, t, W) = W$;
- 3) Otherwise, we treat the first request for d_j (i.e., $r(t_l) \triangleq d_j$) as a W -miss. All subsequent requests for d_j within the next W time steps are regarded as W -delayed hits. Then, we identify the next miss by updating t_l to the time step of the first request for d_j that occurs more than W time steps after the previous miss. This procedure repeats iteratively until $t_l > t$. Finally, $\text{RF}(d_j, t, W)$ is set to the aggregate latency incurred from all such cache misses and delayed hits for d_j , divided by the total number of these misses.

Cases (1), (2), and (3) correspond to lines 3, 6-7, and 8-23 of Algorithm 1, respectively. The computation of the ranking function for items available in the backing store is analogous to that for items available in the core cache C_c . The only difference is that the length $\mathcal{H}$ used for tracking misses is set to Z instead of W , obtaining $\text{RF}(d_j, t, Z)$.

Algorithm 1 Compute $\text{RF}(d_j, t, W)$ or $\text{RF}(d_j, t, Z)$

```

1: Input: an item  $d_j$ , the current time step  $t$ , and a length
    $\mathcal{H}$  ( $\mathcal{H} = W$  or  $\mathcal{H} = Z$ )
2: Output:  $\text{RF}(d_j, t, \mathcal{H})$ 
3:  $\text{RF}(d_j, t, \mathcal{H}) \leftarrow 0$ 
4:  $t_l \leftarrow \min\{t' \leq t \mid r(t') \triangleq d_j\}$   $\triangleright$  The first request for  $d_j$ 
5: if  $t_l \neq \text{NULL}$  then  $\triangleright$  Compute  $\text{RF}(d_j, t, \mathcal{H})$ 
6:   if  $t_l = t$  then
7:      $\text{RF}(d_j, t, \mathcal{H}) \leftarrow \mathcal{H}$ 
8:   else  $\triangleright t_l \neq t$ 
9:      $\text{Agg} \leftarrow 0$   $\triangleright$  Estimated aggregate latency
10:     $\text{Fet} \leftarrow 0$   $\triangleright$  Estimated number of misses
11:    repeat
12:      if  $r(t_l) \triangleq d_j$  then  $\triangleright$  Misses
13:         $\text{Agg} \leftarrow \text{Agg} + \mathcal{H}, \text{Fet} \leftarrow \text{Fet} + 1$ 
14:         $t_{\text{end}} \leftarrow \min\{t_l + \mathcal{H}, t\}$ 
15:        for each  $t' \in [t_l + 1, t_{\text{end}}]$  do
16:          if  $r(t') \triangleq d_j$  then  $\triangleright$  Delayed hits
17:             $\Delta t \leftarrow t' - t_l$ 
18:             $\text{Agg} \leftarrow \text{Agg} + (\mathcal{H} - \Delta t)$ 
19:           $t_l \leftarrow t_{\text{end}} + 1$   $\triangleright$  Trace the next miss
20:        else
21:           $t_l \leftarrow t_l + 1$   $\triangleright$  Trace the next miss
22:      until  $t_l > t$ 
23:       $\text{RF}(d_j, t, \mathcal{H}) \leftarrow \text{Agg} / \text{Fet}$ 
24: return  $\text{RF}(d_j, t, \mathcal{H})$ 

```

C. Workflow of Algorithm DM

Armed with Algorithm 1, DM augments the classical marking framework with two pivotal strategies:

- A dual marking process that simultaneously tracks recency at both the edge cache C_e and the core cache C_c .
- An $\text{RF}(\cdot)$ -driven data eviction strategy that prioritizes items based on estimated average aggregate latency per miss.

Algorithm 2 details the execution pipeline of DM, where lines 8-19 outline the dual marking process and lines 20-36 present the $\text{RF}(\cdot)$ -driven eviction strategy.

1) *Dual Marking Process:* Algorithm DM maintains two separate marking sets: $\mathcal{M}_e$ for the edge cache C_e and $\mathcal{M}_c$ for the core cache C_c , both initialized to $\emptyset$. Whenever an item is requested in σ , it is inserted into both $\mathcal{M}_e$ and $\mathcal{M}_c$.

- If inserting the item into $\mathcal{M}_e$ would cause $|\mathcal{M}_e| > K_1$, then $\mathcal{M}_e$ is reset to $\emptyset$ before inserting the item, which initiates a new marking round for C_e .
- Analogously, if inserting the item into $\mathcal{M}_c$ would cause $|\mathcal{M}_c| > K_2$, then both $\mathcal{M}_e$ and $\mathcal{M}_c$ are reset to $\emptyset$ before the item is inserted into the two marking sets, which initiates a new marking round for both C_c and C_e .

At any time step t , if a W -miss occurs for an item, DM fetches the item from C_c , which would arrive at C_e at time step $t + W$. Analogously, upon a Z -miss for an item, the item is retrieved from the backing store, and it would arrive at C_c at time step $t + Z - W$ and then C_e at time step $t + Z$. Let $f_b^e(t)$ and $f_c^e(t)$ denote the items arriving at C_e at time step t from the

backing store and from the core cache C_c , respectively. Let $f_b^c(t)$ denote the item arriving at C_c from the backing store at time step t .

2) **RF($\cdot$)-Driven Data Eviction:** During the dual marking process, at any time step t , if $f_b^c(t) \triangleq \text{NULL}$, $f_b^e(t) \triangleq \text{NULL}$, and $f_c^e(t) \triangleq \text{NULL}$ all hold, DM leaves the contents of both C_e and C_c unchanged for the next time step $t+1$. Otherwise:

- If $f_b^c(t) \neq \text{NULL}$ (i.e., an item retrieved from the backing store arrives at C_c), DM invokes the classical Least Frequently Used (LFU) policy [20] to evict from C_c the item d_v in the set $\mathcal{S}_c \triangleq C_c(t-1) \cup \{f_b^c(t)\} \setminus \mathcal{M}_c$ with the lowest request frequency, where the request frequency of an item is counted from the beginning of the request sequence σ , using only the request history observable at C_c , i.e., past requests that trigger a W -miss or a Z -miss. We adopt LFU because retaining frequently requested items in C_c increases the likelihood that C_e fetches from C_c rather than from the backing store, thereby reducing aggregate latency.
- If $f_b^e(t) \neq \text{NULL}$ (i.e., an item retrieved from the backing store arrives at C_e), DM invokes Algorithm 1 to compute $\text{RF}(d_j, t, Z)$ for each item d_j in the set $\mathcal{S}_e \triangleq C_e(t-1) \cup \{f_b^e(t)\} \setminus \mathcal{M}_e$. It then evicts from C_e the item d_e in $\mathcal{S}_e$ with the smallest $\text{RF}(d_e, t, Z)$, i.e., adding d_e to $e_e(t)$.
- Analogously, if $f_c^e(t) \neq \text{NULL}$ (i.e., an item retrieved from C_c arrives at C_e), DM invokes Algorithm 1 to compute $\text{RF}(d_j, t, W)$ for each item d_j in the set $\mathcal{S}_e \triangleq C_e(t-1) \cup \{f_c^e(t)\} \setminus \mathcal{M}_e$, and evicts from C_e the item d_g in $\mathcal{S}_e$ with the smallest $\text{RF}(d_g, t, W)$, i.e., adding d_g to $e_e(t)$.

Notably, if any two or all three of $f_b^c(t) \neq \text{NULL}$, $f_b^e(t) \neq \text{NULL}$, and $f_c^e(t) \neq \text{NULL}$ occur at time step t , DM applies the corresponding eviction procedures cumulatively.

V. THEORETICAL PERFORMANCE ANALYSIS

We conduct the competitive analysis for algorithm DM. The result is given in Theorem 2.

Theorem 2. *The competitive ratio of algorithm DM satisfies $\text{CR}(\text{DM}) = \mathcal{O}\left(\max\left\{K_1 Z, \frac{Z^2}{W}\right\}\right)$.*

Remark. *As shall be shown in the detailed analysis, the ratio $\mathcal{O}(K_1 Z)$ arises only when the request sequence σ involves less than $2K_1$ distinct requested items. Under a mild condition that there are at least $2K_1$ distinct items requested, the competitive ratio is given by $\text{CR}(\text{DM}) = \mathcal{O}(\frac{Z^2}{W})$. Note that the lower bound characterized in Theorem 1 can also be simplified to $\Omega(\frac{(K_1+K_2)Z^2}{Z+K_2W}) = \Omega(\frac{(K_1+K_2)Z^2}{K_2W}) = \Omega(\frac{Z^2}{W})$ when $Z = \mathcal{O}(K_2W)$ (which is very likely to hold in practice if K_2 is sizable). Hence, DM asymptotically matches the bound under the aforementioned practical conditions.*

The main challenge in proving Theorem 2 lies in decomposing σ into consecutive subsequences such that each of them simultaneously upper-bounds the aggregate latency of DM and lower-bounds that of the optimal offline algorithm OPT. Two key observations guide our decomposition strategy:

- Under DM, the marking process guarantees that at any time step, every item in the marking set $\mathcal{M}_e$ will never be evicted

Algorithm 2 DM algorithm

```

1: Input: request sequence  $\sigma \triangleq [r(t)]_{t=1}^T$ 
2: Output: eviction policy  $E \triangleq [e_e(t), e_c(t)]_{t=1}^T$ 
3: for each  $t \in [1, T]$  do
4:    $f_b^e(t) \leftarrow \text{NULL}, f_c^e(t) \leftarrow \text{NULL}, f_b^c(t) \leftarrow \text{NULL}$   $\triangleright$ 
     Fetched items
5:    $e_c(t) \leftarrow \emptyset, e_e(t) \leftarrow \emptyset$   $\triangleright$  Initialize eviction sets
6:    $\mathcal{M}_e \leftarrow \emptyset, \mathcal{M}_c \leftarrow \emptyset$   $\triangleright$  Initialize marking sets
7:   for each  $t \in [1, T]$  do
8:     if  $r(t) \neq \emptyset$  then  $\triangleright$  Dual marking process
9:        $\mathcal{M}_e \leftarrow \mathcal{M}_e \cup \{r(t)\}, \mathcal{M}_c \leftarrow \mathcal{M}_c \cup \{r(t)\}$ 
10:      if  $|\mathcal{M}_e| = K_1 + 1$  then
11:         $\mathcal{M}_e \leftarrow \{r(t)\}$ 
12:      if  $|\mathcal{M}_c| = K_2 + 1$  then
13:         $\mathcal{M}_e \leftarrow \{r(t)\}, \mathcal{M}_c \leftarrow \{r(t)\}$ 
14:      if  $r(t)$  incurs a  $W$ -miss then
15:        Fetch  $r(t)$  from  $C_c$ 
16:         $f_c^e(t+W) \leftarrow r(t)$   $\triangleright$  Item retrieved from  $C_c$ 
17:      if  $r(t)$  incurs a  $Z$ -miss then
18:        Fetch  $r(t)$  from the backing store
19:         $f_b^e(t+Z-W) \leftarrow r(t), f_b^e(t+Z) \leftarrow r(t)$   $\triangleright$ 
         Item fetched from the backing store
20:      if  $f_b^c(t) \neq \text{NULL}$  then  $\triangleright$  An item fetched from the
         backing store arrives at  $C_c$ 
21:         $\mathcal{S}_c \leftarrow C_c(t-1) \cup \{f_b^c(t)\} \setminus \mathcal{M}_c$ 
22:        for each  $d_j \in \mathcal{S}_c$  do  $\triangleright$  LFU
23:          Compute the request frequency  $F(d_j)$  of  $d_j$ ,
           counted from the start of  $\sigma$ , using only the request history
           observable at  $C_c$  (i.e.,  $W$ -misses and  $Z$ -misses).
24:           $e_c(t) \leftarrow e_c(t) \cup \{\arg \min_{d_j \in \mathcal{S}_c} F(d_j)\}$ 
25:           $C_c(t) \leftarrow C_c(t-1) \cup \{f_b^c(t)\} \setminus e_c(t)$ 
26:          if  $f_b^e(t) \neq \text{NULL}$  then  $\triangleright$  An item fetched from the
           backing store arrives at  $C_e$ 
27:             $\mathcal{S}_e \leftarrow C_e(t-1) \cup \{f_b^e(t)\} \setminus \mathcal{M}_e$ 
28:            for each  $d_j \in \mathcal{S}_e$  do
29:              Compute  $\text{RF}(d_j, t, Z)$  based on Algorithm 1
30:               $e_e(t) \leftarrow e_e(t) \cup \{\arg \min_{d_j \in \mathcal{S}_e} \text{RF}(d_j, t, Z)\}$ 
31:            if  $f_c^e(t) \neq \text{NULL}$  then  $\triangleright$  An item retrieved from  $C_c$ 
           arrives at  $C_e$ 
32:               $\mathcal{S}_e \leftarrow C_e(t-1) \cup \{f_c^e(t)\} \setminus \mathcal{M}_e$ 
33:              for each  $d_j \in \mathcal{S}_e$  do
34:                Compute  $\text{RF}(d_j, t, W)$  based on Algorithm 1
35:                 $e_e(t) \leftarrow e_e(t) \cup \{\arg \min_{d_j \in \mathcal{S}_e} \text{RF}(d_j, t, W)\}$ 
36:               $C_e(t) \leftarrow C_e(t-1) \cup \{f_b^e(t), f_c^e(t)\} \setminus e_e(t)$ 
37: return  $E \triangleq [e_e(t), e_c(t)]_{t=1}^T$ 

```

from the edge cache C_e , and every item in the marking set $\mathcal{M}_c$ will never be evicted from the core cache C_c .

- OPT must incur at least one W -miss or one Z -miss when serving a request sequence containing $K_1 + 1$ distinct items.

Building on these insights, our proof is structured as follows.

A. Sequence Decomposition

Given any request sequence σ , we first partition σ into *frames*, each defined as a contiguous sequence of time steps. Each frame starts at a time step when the marking set $\mathcal{M}_c$ for the core cache C_c is reset to $\emptyset$ and ends just before $\mathcal{M}_c$ is reset again. That is, if $\mathcal{M}_c$ is reset to $\emptyset$ at time steps $1, t_1, t_2, t_3, \dots$ by algorithm DM in processing σ , the frames are given by time step ranges $[1, t_1 - 1], [t_1, t_2 - 1], [t_2, t_3 - 1], \dots$

By the definition of algorithm DM, a frame involves at most K_2 distinct requested items. We say that a frame is a *complete frame* if it involves exactly K_2 distinct requested items. The following proposition is obvious from the algorithm definition.

Proposition 1. *All frames (except the last one) in σ must be complete, whereas the last frame may be incomplete.*

Lemma 1. *Algorithm DM guarantees that once an item is requested in a frame $\mathbb{F}$, it will not be evicted from the core cache C_c for the remainder of that frame.*

Proof. Suppose that an item d is requested for the first time at time step t_r in $\mathbb{F}$. By the design of DM, d is added to the marking set $\mathcal{M}_c$ at t_r (lines 9 and 13 in Algorithm 2). Afterwards, since the core cache C_c always evicts an item from the set $\mathcal{S}_c \triangleq C_c(t-1) \cup f_b(t) \setminus \mathcal{M}_c$ at any time step $t \geq t_r$ (lines 21-24 in Algorithm 2), d will never be evicted from C_c in the remainder of $\mathbb{F}$. $\square$

B. Frame Segmentation

We further partition each frame into smaller units called *intervals*. Each interval starts at a time step when the marking set $\mathcal{M}_e$ for the edge cache C_e is reset to $\emptyset$ and ends just before $\mathcal{M}_e$ is reset again. That is, if a frame $\mathbb{F}$ starts from time step t_s and $\mathcal{M}_e$ is reset to $\emptyset$ at time steps $t_s, t_1, t_2, t_3, \dots$ by algorithm DM during $\mathbb{F}$ in processing σ , the intervals of $\mathbb{F}$ are given by time step ranges $[t_s, t_1 - 1], [t_1, t_2 - 1], [t_2, t_3 - 1], \dots$

By the definition of DM, an interval involves at most K_1 distinct requested items. We say that an interval is a *complete interval* if it involves exactly K_1 distinct requested items. Let $\mathbb{I}_g(\mathbb{F})$ denote the g -th interval of a frame $\mathbb{F}$. An item is called a *fresh item* in an interval $\mathbb{I}_g(\mathbb{F})$ ($g \geq 2$) if it is requested in $\mathbb{I}_g(\mathbb{F})$ but not in the preceding interval $\mathbb{I}_{g-1}(\mathbb{F})$. By these definitions, we have the following propositions.

Proposition 2. *In any frame, all intervals (except the last one) must be complete, whereas the last interval may be incomplete.*

Proposition 3. *For $g \geq 2$, the first data item requested in the interval $\mathbb{I}_g(\mathbb{F})$ of any frame $\mathbb{F}$ is always a fresh item.*

Lemma 2. *Algorithm DM ensures that once a data item is requested in an interval $\mathbb{I}_g(\mathbb{F})$, it will not be evicted from the edge cache C_e for the remainder of that interval.*

Proof. The proof is similar to Lemma 1. Assume that an item d is requested for the first time at time step t_r in $\mathbb{I}_g(\mathbb{F})$. By the design of DM, d is added to the marking set $\mathcal{M}_e$ at t_r (lines 9, 11, and 13 in Algorithm 2). Afterwards, since the edge cache C_e always evicts an item from the set $C_e(t-1) \cup$

$\{f_c(t), f_b(t)\} \setminus \mathcal{M}_e$ at any time step $t \geq t_r$ (lines 27-30 and 32-35 in Algorithm 2), d will never be evicted from C_e in the remainder of $\mathbb{I}_g(\mathbb{F})$. $\square$

C. Latency Upper Bound of DM

We now proceed to derive an upper bound on the aggregate latency incurred by DM over each frame $\mathbb{F}$ in σ .

Consider any interval $\mathbb{I}_g(\mathbb{F})$ in a frame $\mathbb{F}$. For any item d requested in $\mathbb{I}_g(\mathbb{F})$, we define $t_f(d)$ as the time step when the first request for d arises in $\mathbb{I}_g(\mathbb{F})$, and $t_{re}(d)$ as the time step when the service of this request is completed. Specifically, if the first request $r(t_f(d)) \triangleq d$ for item d in $\mathbb{I}_g(\mathbb{F})$ incurs a cache miss or a delayed hit, we have $t_{re}(d) = t_f(d) + L[r(t_f(d)) \triangleq d]$, where $L[r(t_f(d)) \triangleq d]$ is the latency of serving the request $r(t_f(d)) \triangleq d$. Otherwise, if $r(t_f(d)) \triangleq d$ incurs a cache hit, we have $t_{re}(d) = t_f(d)$. Naturally, $t_{re}(d) - t_f(d) \leq Z$. By Lemma 2, d must be stored in the edge cache C_e from time step $t_{re}(d)$ onward in $\mathbb{I}_g(\mathbb{F})$.

Lemma 3. *For each data item d requested in interval $\mathbb{I}_g(\mathbb{F})$, it holds that $r(t_f(d)) \triangleq d \in C_e(t)$ for all $t \in \mathbb{I}_g(\mathbb{F})$ such that $t \geq t_{re}(d)$.*

We first bound the cumulative latency incurred by DM in serving all requests for any given item over interval $\mathbb{I}_g(\mathbb{F})$.

Lemma 4. *The cumulative latency incurred by DM in serving all requests for each item over an interval $\mathbb{I}_g(\mathbb{F})$ of frame $\mathbb{F}$ is at most $\frac{Z(Z+1)}{2}$.*

Proof. Consider any item d requested in interval $\mathbb{I}_g(\mathbb{F})$. By Lemma 3, DM incurs zero latency in serving all requests for d arising after time step $t_{re}(d)$ in $\mathbb{I}_g(\mathbb{F})$, since these requests are guaranteed to be cache hits. Let t_m be the maximal time step in $\mathbb{I}_g(\mathbb{F})$. We consider two cases:

- 1) $t_{re}(d) \leq t_m$. We obtain $t_{re}(d) - t_f(d) \leq Z \leq t_m - t_f(d)$. By Lemma 3, the maximum latency incurred by DM in serving all requests for d over $\mathbb{I}_g(\mathbb{F})$ is $Z + (Z - 1) + \dots + 1 = \frac{Z(Z+1)}{2}$, corresponding to one Z -miss followed by $(Z - 1)$ consecutive Z -delayed hits.
- 2) $t_{re}(d) > t_m$. We obtain $t_m - t_f(d) < t_{re}(d) - t_f(d) \leq Z$. By Lemma 3, the maximum latency is $Z + (Z - 1) + \dots + (Z - (t_m - t_f(d))) = \frac{(t_m - t_f(d) + 1)(2Z - t_m + t_f(d))}{2} < \frac{Z(Z+1)}{2}$, corresponding to one Z -miss followed by $(t_m - t_f(d))$ consecutive Z -delayed hits.

The above arguments lead to Lemma 4. $\square$

Let $\mathbb{F}_b$ denote the b -th frame of σ . Let h_b be the total number of distinct items requested in $\mathbb{F}_b$, and let q_b be the total number of intervals in $\mathbb{F}_b$. Obviously, $1 \leq h_b \leq K_2$ and $h_b \leq q_b K_1$. If $\mathbb{F}_b$ is complete, then $h_b = K_2$, and otherwise, $h_b < K_2$.

Theorem 3. *The cumulative latency incurred by DM over each frame $\mathbb{F}_b$ in σ is at most $h_b \frac{Z(Z+1)}{2} + (q_b K_1 - h_b) \frac{W(W+1)}{2}$.*

Proof. Let $\mathcal{H}_b$ be the set of distinct items requested in $\mathbb{F}_b$, where $|\mathcal{H}_b| = h_b$. For each item $d \in \mathcal{H}_b$, let P_d be the total number of intervals of $\mathbb{F}_b$ in which item d is requested. Since

each interval contains at most K_1 distinct requested items, we have $\sum_{d \in \mathcal{H}_b} P_d \leq q_b K_1$.

Consider any item $d \in \mathcal{H}_b$. Let $\mathbb{I}_g(\mathbb{F}_b)$ be the interval in which d is requested for the first time in $\mathbb{F}_b$. By Lemma 4, DM incurs an aggregate latency at most $\frac{Z(Z+1)}{2}$ when serving all requests for d over $\mathbb{I}_g(\mathbb{F}_b)$. After paying this cost, d has been fetched into the core cache C_c under DM. By Lemma 1, DM will not evict d from C_c for the remainder of $\mathbb{F}_b$. Hence, all subsequent requests for d in $\mathbb{F}_b$ can be served by accessing C_c . Moreover, by Lemma 2, once d is requested in an interval of $\mathbb{F}_b$, DM will not evict d from the edge cache C_e for the remainder of that interval. Therefore, after d has been fetched into C_c , DM incurs a cumulative latency at most $W + (W + 1) + \dots + 1 = \frac{W(W+1)}{2}$ when serving all requests for d over each interval of $\mathbb{F}_b$, corresponding to one W -miss followed by $(W - 1)$ W -delayed hits. By extending the above analysis for d to all h_b items in $\mathcal{H}_b$, the aggregate latency incurred by DM in serving all requests for items in $\mathcal{H}_b$ over $\mathbb{F}_b$ is at most $h_b \frac{Z(Z+1)}{2} + (\sum_{d \in \mathcal{H}_b} P_d - h_b) \frac{W(W+1)}{2} \leq h_b \frac{Z(Z+1)}{2} + (q_b K_1 - h_b) \frac{W(W+1)}{2}$, where $\sum_{d \in \mathcal{H}_b} P_d \leq q_b K_1$. $\square$

D. Latency Lower Bound of OPT

We bound the cumulative latency of OPT as follows.

Theorem 4. *OPT undergoes at least $\max\{h_b - K_1, \lceil \frac{q_b - 1}{2} \rceil\}$ cache misses over any frame $\mathbb{F}_b$ with $q_b \geq 2$.*

Proof. We distinguish between the following two cases.

- 1) By Proposition 3, a frame $\mathbb{F}_b$ with $q_b \geq 2$ intervals consists of at least $K_1 + 1$ distinct requested items, i.e., $h_b \geq K_1 + 1$. Given that the edge cache C_e can store at most K_1 items, OPT must incur at least $h_b - K_1$ cache misses over $\mathbb{F}_b$.
- 2) We group every two consecutive complete intervals in $\mathbb{F}_b$ into a single entity, called a long interval, ensuring that any two long intervals are non-overlapping. By Propositions 2 and 3, each long interval contains at least $K_1 + 1$ distinct requested items. Since the edge cache C_e can accommodate at most K_1 items, OPT must incur at least one cache miss over each long interval. Given that $\mathbb{F}_b$ includes q_b intervals, among which only the last one may be incomplete according to Proposition 2, OPT must experience at least $\lceil \frac{q_b - 1}{2} \rceil$ cache misses over $\mathbb{F}_b$.

Lemma 4 follows by combining the two bounds above. $\square$

E. Competitive Ratio Quantification

Finally, we integrate Theorems 3 and 4 to derive an upper bound on the competitive ratio $\mathbf{CR}(\text{DM})$ of DM. Suppose both DM and OPT are initialized with identical contents in the core cache C_c and the edge cache C_e , with the contents of C_c fully covering those of C_e . Let B denote the total number of frames in σ . Recall that the b -th frame $\mathbb{F}_b$ comprises q_b intervals and involves h_b distinct requested items in total.

Case 1: $B = 1$ and the only frame $\mathbb{F}_1$ has less than $2K_1$ distinct requested items (i.e., $h_1 < 2K_1$).¹ We further consider two subcases.

- If all the h_1 distinct requested items in $\mathbb{F}_1$ are present in the initial contents of the edge cache C_e (which implies $h_1 \leq K_1$ and there is only one interval in $\mathbb{F}_1$), both DM and OPT incur the same aggregate latency of 0 over σ , and therefore the competitive ratio $\mathbf{CR}(\text{DM}) = 1$.

- If at least one requested item d is absent from the initial cache of C_e , OPT must incur at least one cache miss over σ . We then investigate two possibilities:

(A) If all misses incurred by OPT are W -misses, it implies that all the h_1 distinct requested items in $\mathbb{F}_1$ are present in the initial contents of the core cache C_c . Thus, DM will not incur any Z -miss either. By Lemma 2, once an item d is requested in an interval of $\mathbb{F}_1$, DM will not evict d from the edge cache C_e for the remainder of that interval. Therefore, for each requested item d in an interval of $\mathbb{F}_1$, DM incurs a cumulative latency at most $W + (W + 1) + \dots + 1 = \frac{W(W+1)}{2}$ when serving all requests for d over the interval, corresponding to one W -miss followed by $(W - 1)$ W -delayed hits. Since there are q_1 intervals in $\mathbb{F}_1$ and at most K_1 distinct requested items in each interval, the aggregate latency incurred by DM over $\mathbb{F}_1$ is at most $q_1 K_1 \frac{W(W+1)}{2}$. On the other hand, OPT incurs at least one W -miss over $\mathbb{F}_1$. Further by Theorem 4, OPT must experience at least $\max\{1, \lceil \frac{q_1 - 1}{2} \rceil\}$ W -misses over $\mathbb{F}_1$, and thus incurs an aggregate latency at least $\max\{1, \lceil \frac{q_1 - 1}{2} \rceil\} \cdot W$. It follows that

$$\mathbf{CR}(\text{DM}) = \frac{\text{MA}(\sigma)}{\text{OPT}(\sigma)} \leq \frac{q_1 K_1 \frac{W(W+1)}{2}}{\max\{1, \lceil \frac{q_1 - 1}{2} \rceil\} \cdot W} = \mathcal{O}(K_1 W).$$

(B) If at least one miss incurred by OPT is a Z -miss, it implies that at least one requested item in $\mathbb{F}_1$ is absent from the initial caches of C_c and C_e . By Theorem 3, DM incurs an aggregate latency at most $h_1 \frac{Z(Z+1)}{2} + (q_1 K_1 - h_1) \frac{W(W+1)}{2}$ over $\mathbb{F}_1$. On the other hand, OPT incurs at least one Z -miss over $\mathbb{F}_1$. Further by Theorem 4, OPT must experience at least another $\max\{1, \lceil \frac{q_1 - 1}{2} \rceil\} - 1$ misses over $\mathbb{F}_1$. Thus, OPT incurs an aggregate latency at least $Z + (\max\{1, \lceil \frac{q_1 - 1}{2} \rceil\} - 1) \cdot W$. Therefore,

$$\mathbf{CR}(\text{DM}) = \frac{\text{MA}(\sigma)}{\text{OPT}(\sigma)} \leq \frac{h_1 \frac{Z(Z+1)}{2} + (q_1 K_1 - h_1) \frac{W(W+1)}{2}}{Z + (\max\{1, \lceil \frac{q_1 - 1}{2} \rceil\} - 1) \cdot W}.$$

To bound the above ratio, we consider the following cases.

- If $q_1 = 1$, then $h_1 \leq K_1$. Since $Z \gg W$, we have

$$\mathbf{CR}(\text{DM}) \leq \frac{K_1 \frac{Z(Z+1)}{2}}{Z} = \mathcal{O}(K_1 Z).$$

- If $q_1 > 1$ and $h_1 \leq 2K_1$, we have

$$\begin{aligned} \mathbf{CR}(\text{DM}) &\leq \frac{h_1 \frac{Z(Z+1)}{2}}{Z} + \frac{(q_1 K_1 - h_1) \frac{W(W+1)}{2}}{Z + (\lceil \frac{q_1 - 1}{2} \rceil - 1) \cdot W} \\ &\leq \frac{h_1 \frac{Z(Z+1)}{2}}{Z} + \frac{(q_1 K_1 - h_1) \frac{W(W+1)}{2}}{\lceil \frac{q_1 - 1}{2} \rceil \cdot W} \\ &= \mathcal{O}(K_1 Z) + \mathcal{O}(K_1 W) \\ &= \mathcal{O}(K_1 Z). \end{aligned}$$

¹The coefficient 2 can be replaced by any given constant greater than 1.

Case 2: $B = 1$ and the only frame $\mathbb{F}_1$ has at least $2K_1$ distinct requested items (i.e., $h_1 \geq 2K_1$). By Theorem 4, OPT must experience at least $\max\{h_1 - K_1, \lceil \frac{q_1 - 1}{2} \rceil\}$ cache misses over $\mathbb{F}_1$, and thus incurs an aggregate latency at least $\max\{h_1 - K_1, \lceil \frac{q_1 - 1}{2} \rceil\} \cdot W$ over $\mathbb{F}_1$. By Theorem 3, DM incurs an aggregate latency at most $h_1 \frac{Z(Z+1)}{2} + (q_1 K_1 - h_1) \frac{W(W+1)}{2}$ over $\mathbb{F}_1$. We thus obtain

$$\begin{aligned} \text{CR}(\text{DM}) &= \frac{\text{MA}(\sigma)}{\text{OPT}(\sigma)} \leq \frac{h_1 \frac{Z(Z+1)}{2} + (q_1 K_1 - h_1) \frac{W(W+1)}{2}}{\max\{h_1 - K_1, \lceil \frac{q_1 - 1}{2} \rceil\} \cdot W} \\ &\leq \frac{h_1 \frac{Z(Z+1)}{2}}{(h_1 - K_1) \cdot W} + \frac{(q_1 K_1 - h_1) \frac{W(W+1)}{2}}{\lceil \frac{q_1 - 1}{2} \rceil \cdot W} \\ &= \mathcal{O}\left(\frac{Z^2}{W}\right) + \mathcal{O}(K_1 W) \\ &= \mathcal{O}\left(\max\left\{\frac{Z^2}{W}, K_1 W\right\}\right), \end{aligned}$$

where the second equality holds because $h_1 \geq 2K_1$.

Case 3: $B \geq 2$, i.e., σ includes at least two frames. By Proposition 1, the first $B - 1$ frames in σ must be complete, where the last frame $\mathbb{F}_B$ may be incomplete, i.e., it may have less than K_2 distinct requested items. Let $\text{DM}(\mathbb{F}_b)$ and $\text{OPT}(\mathbb{F}_b)$ denote the aggregate latencies of DM and OPT over the b -th frame $\mathbb{F}_b$ in σ , respectively. We first analyze the first $B - 2$ complete frames in σ . A complete frame has K_2 distinct requested items where $K_2 \gg K_1$. Following the analysis in **Case 2**, the latency ratio between DM and OPT over each of the first $B - 2$ complete frames is at most $\mathcal{O}(\max\{\frac{Z^2}{W}, K_1 W\})$. Hence, the latency ratio over the first $B - 2$ complete frames together is also bounded by $\frac{\sum_{b=1}^{B-2} \text{DM}(\mathbb{F}_b)}{\sum_{b=1}^{B-2} \text{OPT}(\mathbb{F}_b)} = \mathcal{O}(\max\{\frac{Z^2}{W}, K_1 W\})$. We next consider the last two frames in σ , i.e., $\mathbb{F}_{B-1}$ and $\mathbb{F}_B$. Note that $h_{B-1} = K_2$ and $h_B \leq K_2$. By Theorems 3 and 4, we have

$$\begin{aligned} \frac{\text{DM}(\mathbb{F}_{B-1}) + \text{DM}(\mathbb{F}_B)}{\text{OPT}(\mathbb{F}_{B-1}) + \text{OPT}(\mathbb{F}_B)} &\leq \frac{(K_2 + h_B) \frac{Z(Z+1)}{2} + (q_{B-1} K_1 - K_2 + q_B K_1 - h_B) \frac{W(W+1)}{2}}{\max\{K_2 - K_1, \lceil \frac{q_{B-1} - 1}{2} \rceil\} \cdot W + \max\{h_B - K_1, \lceil \frac{q_B - 1}{2} \rceil\} \cdot W} \\ &\leq \frac{2K_2 \frac{Z(Z+1)}{2}}{(K_2 - K_1) \cdot W} + \frac{(q_{B-1} K_1 + q_B K_1) \frac{W(W+1)}{2}}{(\lceil \frac{q_{B-1} - 1}{2} \rceil + \lceil \frac{q_B - 1}{2} \rceil) \cdot W} \\ &= \mathcal{O}\left(\frac{Z^2}{W}\right) + \mathcal{O}(K_1 W) \\ &= \mathcal{O}\left(\max\left\{\frac{Z^2}{W}, K_1 W\right\}\right), \end{aligned}$$

where the second equality holds because $K_2 \gg K_1$. Therefore, $\text{CR}(\text{DM}) = \mathcal{O}(\max\{\frac{Z^2}{W}, K_1 W\})$.

Putting all cases together, we conclude that

$$\begin{aligned} \text{CR}(\text{DM}) &= \mathcal{O}\left(\max\left\{K_1 W, K_1 Z, \frac{Z^2}{W}\right\}\right) \\ &= \mathcal{O}\left(\max\left\{K_1 Z, \frac{Z^2}{W}\right\}\right). \end{aligned}$$

The proof of Theorem 2 is thus completed.

VI. EXPERIMENTAL EVALUATION

We conduct extensive experiments to empirically compare the performance of DM against state-of-the-art solutions.

A. Evaluation Methodology

Datasets. Our experiments are conducted on three production traces: 1) a Meta block trace [21], containing anonymized production workloads from a flash cache deployed in front of a bulk storage system; 2) an IBM trace [22], collected over a course of 75 days from five IBM data centers hosting production-level registries; and 3) a RobinHood trace [23], sampled from Microsoft production traffic over several days in 2017 and 2018. From each trace, we extract a sequence of 40000 consecutive requests (i.e., $T = 40000$) for evaluation. The extracted sequences contain $N = 9016$, $N = 4957$, and $N = 13343$ distinct requested items for the Meta, IBM, and RobinHood traces, respectively.

Baselines. We select five representative existing algorithms as baselines: LRU [24], LFU [20], ARC [25], LHD [26], and LAC [19]. Among them, LRU, LFU, and ARC are classical online caching policies: LRU prioritizes items based on how recently they were requested, LFU ranks items by their request frequencies, and ARC adaptively balances recency and frequency to guide eviction decisions. LHD is a state-of-the-art approach that evicts items according to their hit densities, while LAC is a timer-based mechanism designed for the single-cache model with delayed hits. To ensure a fair comparison, each baseline is executed separately on the core cache C_c and the edge cache C_e . When operating on C_c , each baseline prioritizes items using only the request history visible to C_c , i.e., past requests that trigger a W -miss or a Z -miss. In contrast, when operating on C_e , it can leverage the complete request history, since C_e observes every incoming request. These settings are fully aligned with our system model and the design of DM algorithm.

Experimental setup and metrics. Each algorithm is simulated with identical initial contents in both C_c and C_e . The default parameters are set to $W = 10$, $Z = 100$, $K_1 \approx 0.1N$, and $K_2 \approx 0.4N$. As the polynomial-time computability of an optimal offline algorithm under delay hits remains an open question [1], we cannot directly quantify the competitive ratio of each simulated algorithm. Therefore, our experiments focus on another critical performance metric: aggregate latency.

B. Experiment Results

Fig. 3, Fig. 4, and Fig. 5 compare the aggregate latency of the simulated algorithms on the Meta, IBM, and RobinHood traces, respectively. We observe that the aggregate latency of each algorithm decreases roughly linearly with both K_1 and K_2 , while increasing roughly linearly with both W and Z . These trends are intuitive: scaling up the cache capacities K_1 and K_2 improves hit ratios and thus reduces the aggregate latency, whereas larger W and Z directly amplify the latency associated with cache misses and therefore raise the aggregate latency. Moreover, the aggregate latency of each algorithm is more sensitive to K_2 than to K_1 . This is because increasing

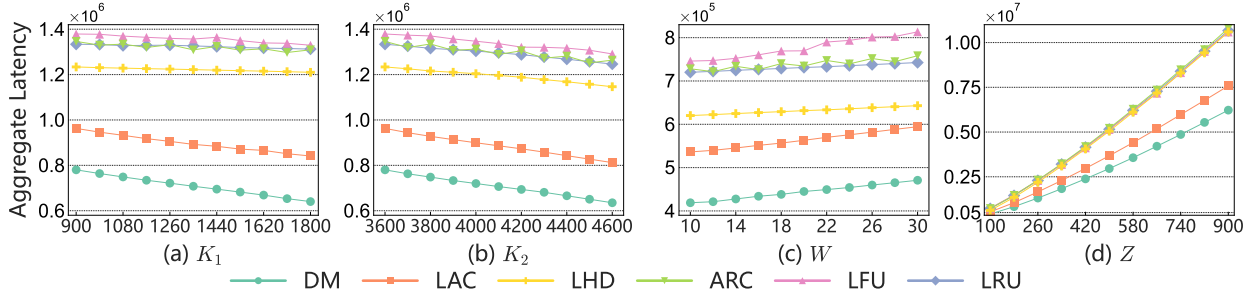


Fig. 3: Aggregate latency on the Meta trace (default: $K_1 = 900, K_2 = 3600, W = 10, Z = 100, N = 9016$).

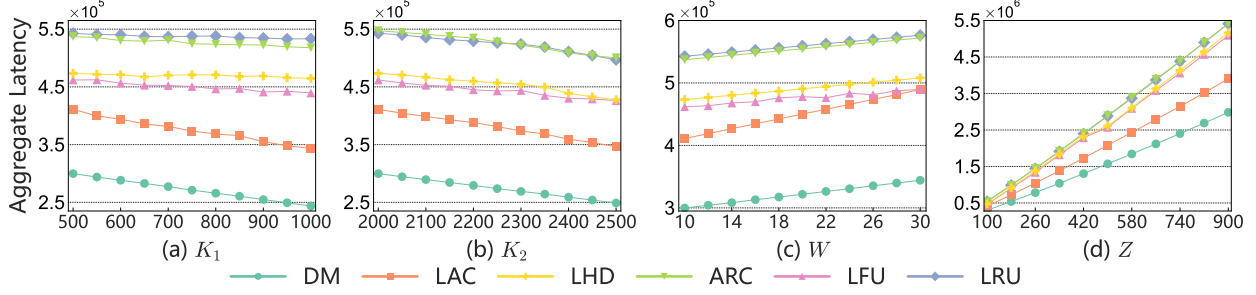


Fig. 4: Aggregate latency on the IBM trace (default: $K_1 = 500, K_2 = 2000, W = 10, Z = 100, N = 4957$).

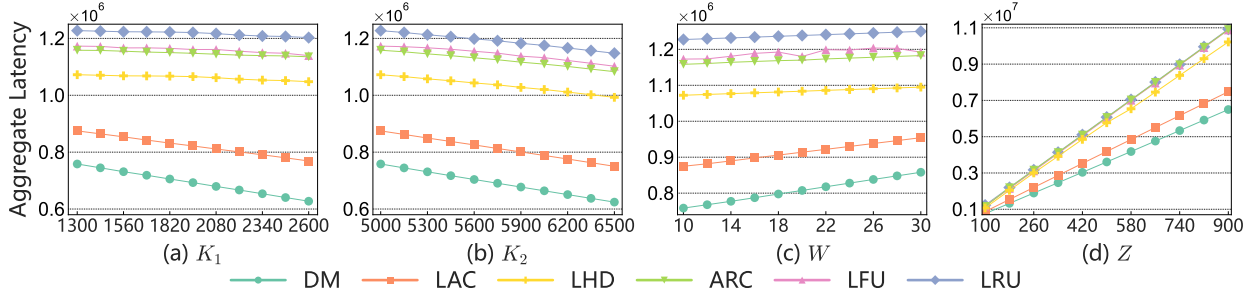


Fig. 5: Aggregate latency on the RobinHood trace (default: $K_1 = 1300, K_2 = 5000, W = 10, Z = 100, N = 13343$).

K_2 enhances the likelihood that the edge cache C_e retrieves items from the core cache C_c instead of the backing store, yielding larger latency savings. Analogously, since Z -misses contribute more substantially to aggregate latency than W -misses, all algorithms are more sensitive to Z than to W .

Across all three traces, our algorithm DM consistently outperforms the five baselines, reducing aggregate latency by at least 15% on average under various parameter settings. Concretely, in Fig. 3(a), at $K_1 = 1440$, the best baseline LAC incurs an aggregate latency of approximately 0.9×10^6 , whereas DM achieves roughly 0.7×10^6 , corresponding to a 22.2% reduction. Similarly, in Fig. 3(d), at $Z = 420$, the best baseline LAC reaches an aggregate latency of about 0.29×10^7 , while DM reduces it to roughly 0.23×10^7 , yielding a performance gain of 20.7%. Compared with Fig. 3, Fig. 4 shows that the performance improvement of DM becomes even more pronounced, reaching roughly 24.2% on average. For example, in Fig. 4(c), at $W = 30$, the best baseline LFU exhibits an aggregate latency of approximately 4.9×10^5 , whereas DM achieves about 3.4×10^5 , corresponding to a 30.6% reduction.

In Fig. 5, DM maintains an average performance gain above 15%. Moreover, across all three traces, DM exhibits smaller performance fluctuations than LHD, ARC, and LFU, further underscoring its robustness and superiority.

VII. RELATED WORK

Caching algorithms have garnered significant attention in both offline [9], [12], [13], [24], [27] and online settings [11], [28], [29]. In contrast, delayed hits [1] have been comparatively underexplored, even though they induce latencies beyond those captured by traditional caching models. [1] proposed a near-optimal offline algorithm for the single-cache model with delayed hits and derived an online algorithm by incorporating the cumulative latency of each item into the caching heuristics. Follow-up work examined delayed hits in time-to-live caches [17] and in non-stationary environments with non-uniform item sizes and retrieval latencies [18]. From a theoretical standpoint, [14] established a tight lower bound of $\Omega(KZ)$ on the competitive ratio of any deterministic online algorithm under the single-cache delayed-hit model, where

K denotes the cache capacity. Building on this result, [16] showed that LRU achieves $\mathcal{O}(KZ)$ -competitiveness, which is asymptotically optimal. [15] and [30] further extended the study of delayed hits to online general file caching, proving lower bounds of $\Omega(KZ)$ for deterministic and $\Omega(Z \log K)$ for randomized algorithms, and designing deterministic and randomized algorithms with competitive ratios $\mathcal{O}(Z^{3/2}K)$ and $\mathcal{O}(Z^{3/2} \log K)$, respectively.

VIII. CONCLUSION

In this paper, we have pioneered a general algorithmic analysis for the hierarchical caching problem with delayed hits. We first established a competitiveness lower bound for any online caching algorithm. We then developed a novel online caching algorithm, termed DM, by synthesizing two key technical ideas. The first is a dual marking policy that executes two separate marking processes, one for the edge cache and the other for the core cache. The second is a heuristic ranking function that leverages historical request patterns to prioritize items based on their estimated aggregate latency per cache miss. Finally, we analytically prove that DM can asymptotically match the lower bound under some mild and practical conditions. Extensive experiments show that DM consistently outperforms state-of-the-art solutions by an average of at least 15% in aggregate latency. Future efforts will focus on refining DM to fully achieve the established lower bound and to deliver additional empirical performance gains.

ACKNOWLEDGMENT

This work is supported by the Ministry of Education, Singapore, under its AcRF Tier 1 (Awards RG23/23, RG15/25).

REFERENCES

- [1] N. Atre, J. Sherry, W. Wang, and D. S. Berger, "Caching with delayed hits," in *SIGCOMM '20*, (New York, NY, USA), p. 495–513, Association for Computing Machinery, 2020.
- [2] C. Dong, F. Wang, H. Jiang, and D. Feng, "Using lock-free design for throughput-optimized cache eviction," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 9, June 2025.
- [3] T. X. Tran, D. V. Le, G. Yue, and D. Pompili, "Cooperative hierarchical caching and request scheduling in a cloud radio access network," *IEEE Transactions on Mobile Computing*, vol. 17, no. 12, pp. 2729–2743, 2018.
- [4] A. Liu, V. K. N. Lau, and G. Caire, "Cache-induced hierarchical cooperation in wireless device-to-device caching networks," *IEEE Transactions on Information Theory*, vol. 64, no. 6, pp. 4629–4652, 2018.
- [5] C. Jiang, "Economic analysis of joint mobile edge caching and peer content sharing," in *Proceedings of IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pp. 1–10, IEEE, 2023.
- [6] Y. Zeng, Y. Huang, Z. Liu, and J. Liu, "Distributed and decentralized edge caching in 5G networks using non-volatile memory systems," in *Proceedings of the 42nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 425–435, IEEE, 2022.
- [7] Y. Wang, H. Dai, X. Han, P. Wang, Y. Zhang, and C.-Z. Xu, "Cost-driven data caching in edge-based content delivery networks," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1384–1400, 2023.
- [8] B. Carleton, "Delayed hits in multi-level caches," *Symposium on Operating Systems Principles (Poster Session)*, 2021.
- [9] J. Dallot, A. J. Fesharaki, M. Pacut, and S. Schmid, "Dependency-aware online caching," in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pp. 871–880, 2024.
- [10] B. Abolhassani, J. Tadrous, A. Eryilmaz, and E. Yeh, "Fresh caching for dynamic content," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–10, 2021.
- [11] T. Lykouris and S. Vassilvitskii, "Competitive caching with machine learned advice," *J. ACM*, vol. 68, July 2021.
- [12] A. Blankstein, S. Sen, and M. J. Freedman, "Hyperbolic caching: Flexible caching for web applications," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, (Santa Clara, CA), pp. 499–511, USENIX Association, July 2017.
- [13] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 78–89, 2016.
- [14] P. Manohar and J. Williams, "Lower bounds for caching with delayed hits," in *arXiv*, 2020.
- [15] C. Zhang, H. Tan, G. Li, Z. Han, S. H.-C. Jiang, and X.-Y. Li, "Online file caching in latency-sensitive systems with delayed hits and bypassing," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pp. 1059–1068, 2022.
- [16] K. Gurushankar, N. G. Singer, and B. Subercaseaux, "Latency guarantees for caching with delayed hits," in *IEEE INFOCOM 2025 - IEEE Conference on Computer Communications*, pp. 1–10, 2025.
- [17] K. Elsayed and A. Rizk, "Time-to-live caching with network delays: Exact analysis and computable approximations," *IEEE/ACM Transactions on Networking*, vol. 31, no. 3, pp. 1087–1100, 2023.
- [18] B. Jiang, Y. Yang, and B. Jiang, "Optimizing latency for caching with delayed hits in non-stationary environment," *Performance Evaluation*, vol. 169, p. 102488, 2025.
- [19] G. Yan and J. Li, "Towards latency awareness for content delivery network caching," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, (Carlsbad, CA), pp. 789–804, USENIX Association, July 2022.
- [20] S. L. Noh, S. L. Min, Y. K. Cho, and C. S. Kim, "Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Transactions on Computers*, vol. 50, no. 12, pp. 1352–1361, 2001.
- [21] D. L.-K. Wong, H. Wu, C. Molder, S. Gunasekar, J. Lu, S. Khandkar, A. Sharma, D. S. Berger, N. Beckmann, and G. R. Ganger, "Baleen: ML admission & prefetching for flash caches," in *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, (Santa Clara, CA), pp. 347–371, USENIX Association, Feb. 2024.
- [22] A. Anwar, M. Mohamed, V. Tarasov, M. Litty, L. Rupprecht, Y. Cheng, N. Zhao, D. Skourtis, A. S. Warke, H. Ludwig, D. Hildebrand, and A. R. Butt, "Improving docker registry design based on production workload analysis," in *16th USENIX Conference on File and Storage Technologies (FAST 18)*, (Oakland, CA), pp. 265–278, USENIX Association, Feb. 2018.
- [23] D. S. Berger, B. Berg, T. Zhu, S. Sen, and M. Harchol-Balter, "Robin-Hood: Tail latency aware caching – dynamic reallocation from Cache-Rich to Cache-Poor," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, (Carlsbad, CA), pp. 195–212, USENIX Association, Oct. 2018.
- [24] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive paging algorithms," *Journal of Algorithms*, vol. 12, no. 4, pp. 685–699, 1991.
- [25] N. Megiddo and D. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, 2003.
- [26] N. Beckmann, H. Chen, and A. Cidon, "LHD: Improving cache hit rate by maximizing hit density," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, (Renton, WA), pp. 389–403, USENIX Association, Apr. 2018.
- [27] S. Albers, S. Arora, and S. Khanna, "Page replacement for general caching problems," in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '99*, (USA), p. 31–40, Society for Industrial and Applied Mathematics, 1999.
- [28] M. Chrobak, E. Koutsoupias, and J. Noga, "More on randomized online algorithms for caching," *Theoretical Computer Science*, vol. 290, no. 3, pp. 1997–2008, 2003.
- [29] N. Beckmann, H. Chen, and A. Cidon, "LHD: Improving cache hit rate by maximizing hit density," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, (Renton, WA), pp. 389–403, USENIX Association, Apr. 2018.
- [30] H. Tan, Y. Wang, C. Zhang, G. Li, H. Du, Z. Han, S. H.-C. Jiang, and X.-Y. Li, "Asymptotically tight approximation for online file caching with delayed hits and bypassing," *IEEE Transactions on Networking*, vol. 33, no. 4, pp. 1886–1899, 2025.