Distributed Caching with Delayed Hits

Kanghuai Liu[1], Xueyan Tang[1], Lin Chen[2], Guocong Quan[3], Xuan Qui Pham[1]
[1]Nanyang Technological University, Singapore
[2]Macao Polytechnic University, China
[3]Sun Yat-sen University, China
Email: kanghuai.liu@ntu.edu.sg, asxytang@ntu.edu.sg

***Abstract*—Caches serve as the cornerstone of latency-sensitive systems, yet a recently discovered phenomenon, so-called *delayed hits*, challenges conventional optimal caching algorithms, rendering them incapable of guaranteeing minimal latency. To fill this gap, in this paper, we pioneer a generic algorithmic analysis for distributed caching under delayed hits, making three contributions. Firstly, we analytically establish tight competitive ratio lower bounds for both deterministic and randomized online caching algorithms. Secondly, we develop a novel online caching optimization framework by generalizing the Least Recently Used (LRU) policy to distributed caching with delayed hits, encompassing both deterministic and randomized paradigms. Our key technicality is a timer-based data fetching scheme that leverages query history to limit aggregate data retrieval latency. We prove that our deterministic and randomized algorithms are both asymptotically optimal. Thirdly, we conduct extensive experiments on a real-world dataset to demonstrate the effectiveness and superiority of our algorithms over state-of-the-art solutions.**

I. Introduction

Caches are key components of the optimization toolkit for memory-intensive and latency-sensitive network systems [1], [2]. A fundamental problem arising in such cache-aware network systems is the design and analysis of eviction and scheduling policies to minimize aggregate latency [3]. In its generic distributed setting, a backing store maintains the complete set of unique data items, whereas multiple capacity-constrained caches collectively store a subset of all the items. Each request results in one of two outcomes: a cache hit or a cache miss. For a cache miss, caches support both low-latency peer-to-peer data retrieval and higher-latency data fetching from the backing store. Instances of such a distributed setting are ubiquitous across virtually all of today's networked cache systems, including but not limited to:

- **Mobile edge caching** systems deploy caches at edge servers closer to end-users to reduce the delay in data retrieval from central servers [4], enhancing Quality of Service (QoS) and alleviating traffic congestion in the core network [5].
- **Content delivery networks** optimize content delivery by caching frequently accessed data at geographically distributed edge servers [6], thereby enabling high-speed streaming services and supporting large-scale web traffic [7].
- **Data center networks** cache frequently requested data at local data centers to alleviate the load on core storage systems, expedite data retrieval, mitigate I/O bottlenecks, and ensure the real-time execution of compute-intensive tasks [8], [9].

While caches are of practically paramount importance for optimizing latency across various distributed systems, the design of caching policies has long been recognized as a challenging problem in the literature [10]–[14]. Recently, a phenomenon, *delayed hits* [1] — the third potential outcome for requests — discovered to have a potentially significant impact on caching performance, renders the problem even more non-trivial. Concretely, delayed hits occur at a cache in high-throughput distributed systems when multiple requests for the same data item queue up before the item is fetched from another cache or the backing store. Under delayed hits, the traditional caching objective of hit-rate maximization no longer aligns with latency minimization, rendering conventional optimal caching algorithms unable to guarantee minimal latency. Hence, *we argue that delayed hits highlight a significant gap in the current understanding and practices of caching, urgently calling for in-depth theoretical and algorithmic research to address the associated challenges.*

Despite the profound impact of delayed hits on caching, research on this phenomenon remains limited [1], [3], [15], [16], with no prior work studying delayed hits in the context of distributed caching.

In this paper, we pioneer a generic algorithmic analysis for distributed caching with delayed hits by investigating the following research questions:

- Q1: What is the theoretical performance limit on the competitive ratio for any deterministic or randomized online caching algorithm?
- Q2: How can we design online caching algorithms to approach or even achieve the performance limit?

To answer Q1, we analytically derive a competitive ratio lower bound for *any* deterministic online caching algorithm in Theorem 1 as $\Omega\left(\frac{KMZ^2}{Z+MW}\right)$, where M is the number of caches, each having a storage capacity of K, and W and Z are the latencies of data retrieval from a cache and the backing store, respectively. We further extend our analysis to *any* randomized online caching algorithm and characterize in Theorem 2 a corresponding lower bound as $\Omega\left(\frac{MZ^2\log K}{Z+MW}\right)$.

To answer Q2, we develop a novel online caching algorithm, termed Distributed Least Recently Used (DLRU), which encompasses both a deterministic variant $\mathrm{DLRU_D}$ and a randomized variant $\mathrm{DLRU_R}$. $\mathrm{DLRU_D}$ generalizes the classical LRU policy to distributed caching with delayed hits. The key innovation behind $\mathrm{DLRU_D}$ is a timer-based data fetching

scheme that leverages query history to balance various types of data acquisitions, limiting aggregate data retrieval latency. $\mathrm{DLRU_R}$ builds upon the core architecture of $\mathrm{DLRU_D}$ but mobilizes a marking-based probabilistic eviction policy. We demonstrate the optimality of our algorithms through rigorous theoretical analysis. Concretely, we analytically prove in Theorem 3 that $\mathrm{DLRU_D}$ achieves a competitive ratio of $\mathcal{O}\left(\frac{KMZ^2}{Z+MW}\right)$, and demonstrate in Theorem 5 that $\mathrm{DLRU_R}$ is $\mathcal{O}\left(\frac{MZ^2\log K}{Z+MW}\right)$-competitive. Both algorithms are asymptotically optimal by Theorems 1 and 2. Empirically, extensive experiments on a real-world dataset demonstrate that $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ outperform state-of-the-art solutions by at least 18% on average in terms of aggregate latency and competitive ratio. Our work makes a technical step towards a better theoretical understanding of delayed hits.

II. Preliminaries

A. System Model

We consider a generic distributed caching system [17], [18], as illustrated in Figure 1. A set $\mathcal{S} = \{s_i\}_{i=1}^{M}$ of M geo-distributed edge caching servers are fully connected, each covered by a single Base Station (BS). The core network interconnects these BSs to a remote central server s_c, which maintains a set $\mathcal{D} = \{d_j\}_{j=1}^{N}$ of N distinct data items, each normalized to unit size. Given a storage capacity of K, each edge server can cache a subset of up to K data items from $\mathcal{D}$. In practice, the combined cache capacity of the M edge servers is limited, implying that $N > KM$. Without loss of generality, we assume that the system operates in discrete time steps. For any edge server s_i, fetching a data item from s_c entails a latency of Z time steps, while retrieving it from any other edge server requires W time steps. Since data communication among edge servers can be realized via high-bandwidth and low-latency fronthaul links [19], [20], it follows that $Z \gg W$.

Requests to access data items arise at each edge server over time, with each request targeting a specific data item in $\mathcal{D}$. At each time step t, the cache of an edge server s_i currently stores a set $C(s_i, t-1)$ of data items and s_i receives a request $r(s_i, t) \triangleq d_j$ seeking to access a data item $d_j \in \mathcal{D}$.[1] If $d_j \in C(s_i, t-1)$, then s_i undergoes a *cache hit*, incurring 0 latency. Otherwise, if $d_j \notin C(s_i, t-1)$, there are two cases:

- *Cache miss:* If the service for the most recent request to d_j at s_i was completed before t, $r(s_i, t)$ incurs a *cache miss*.
  1. s_i broadcasts acquisition queries for d_j to all of its peers.
     1.1. If any peer has cached d_j at time step t, s_i serves $r(s_i, t)$ by fetching d_j from that peer in W time steps, incurring a *W-miss* with latency W.
     1.2. If none of the peers have cached d_j at time step t, s_i waits W time steps and then fetches it from the central server s_c, resulting in a *$(W+Z)$-miss* with latency $W+Z$.
  2. Alternatively, s_i may opt to retrieve d_j directly from s_c at time step t, leading to a *Z-miss* with latency Z.
- *Delayed hit:* If the most recent cache miss for d_j at s_i occurred some X time steps ago and is still being processed at time step t, $r(s_i, t)$ incurs a *delayed hit*.
  1. If the most recent miss for d_j was a W-miss, $r(s_i, t)$ incurs a *W-delayed hit* with latency $W - X$.
  2. If it was a $(W+Z)$-miss, $r(s_i, t)$ results in a *$(W+Z)$-delayed hit* with latency $W + Z - X$.
  3. If it was a Z-miss, $r(s_i, t)$ incurs a *Z-delayed hit* with latency $Z - X$.

In all cases, $r(s_i, t)$ will be locally served on s_i at the time step t plus the corresponding latency. Whenever an edge server s_i encounters a cache miss for a data item, it should select an appropriate method to fetch the item to minimize latency, i.e., attempting to fetch it from peers or retrieving it directly from s_c. Conversely, once a set $\mathcal{T}(s_i, t)$ of fetched data items arrives at edge server s_i at time step t, s_i employs $\mathcal{T}(s_i, t)$ to serve all types of requests for each item in $\mathcal{T}(s_i, t)$. These requests include both cache misses and delayed hits. Notably, the cardinality constraint $|\mathcal{T}(s_i, t)| \leq 3$ holds for all $t \in [1, T]$, as up to 3 distinct fetched data items can arrive at s_i simultaneously at any time step.[2] Afterwards, s_i evicts a set $e(s_i, t)$ of data items from $C(s_i, t-1) \cup \mathcal{T}(s_i, t)$, updating its cache as $C(s_i, t) \leftarrow C(s_i, t-1) \cup \mathcal{T}(s_i, t) \setminus e(s_i, t)$. Notably, if $e(s_i, t) \triangleq \mathcal{T}(s_i, t)$, then $C(s_i, t) \leftarrow C(s_i, t-1)$. This indicates that our system setting allows bypassing, a feature commonly found in classical caching models [15].

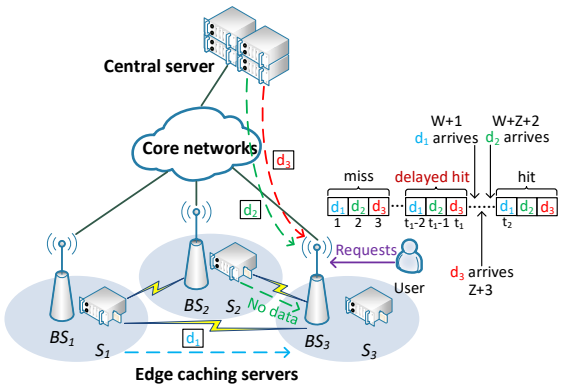

Fig. 1: Illustration of our distributed caching framework with delayed hits: the dashed arrows indicate distinct ways to retrieve requested data items.

Fig. 1 illustrates an example with three edge caching servers s_1, s_2, and s_3 to demonstrate our system model. A sequence of requests arises from a user located near s_3. Initially, s_3 has not cached data items d_1, d_2, and d_3, resulting in cache misses for requests $r(s_3, 1) = d_1$, $r(s_3, 2) = d_2$, and $r(s_3, 3) = d_3$. The fetching process for each item is as follows: (i) s_3 requests d_1 from both s_1 and s_2. Since s_1 has cached d_1 at time step 1, it transfers a copy of d_1 to s_3, classifying $r(s_3, 1)$ as a W-miss. (ii) Analogously, s_3 requests d_2 from both s_1 and s_2,

[1] The notation $r(s_i, t)$ refers to both the request itself and the data item that the request aims to access from s_i at time step t.

[2] Given $|\mathcal{T}(s_i, t)| \leq 3$, we without loss of generality assume that $K \geq 3$.

but neither has cached d_2 at time step 2. After W time steps, s_3 then turns to fetch d_2 from the central server s_c, classifying $r(s_3, 2)$ as a $(W + Z)$-miss. (iii) s_3 fetches d_3 directly from s_c, classifying $r(s_3, 3)$ into a Z-miss. The fetched items d_1, d_2, and d_3 arrive at s_2 at time steps $W + 1$, $W + Z + 2$ and $Z + 3$ respectively, which are all between time steps t_1 and t_2. Hence, $r(s_3, t_1 - 2)$ has a W-delayed hit for d_1 with latency $W + 1 - (t_1 - 2) = W - t_1 + 3$. $r(s_3, t_1 - 1)$ has a $(W + Z)$-delayed hit for d_2 with latency $W + Z + 2 - (t_1 - 1) = W + Z - t_1 + 3$. $r(s_3, t_1)$ has a Z-delayed hit for d_3 with latency $Z + 3 - t_1 = Z - t_1 + 3$.

B. Problem Formulation

To characterize our problem, we begin with two definitions.

Definition 1 (Request sequence)**.** *A request sequence $R^T(s_i)$ is a finite sequence $R^T(s_i) = [r(s_i, t)]_{t=1}^T$ at edge server s_i over T time steps. If $r(s_i, t) \triangleq d_j$ then data item d_j is requested on s_i at time step t. If $r(s_i, t) \triangleq \emptyset$, then no data item is requested (i.e., no request arises) on s_i at time step t.*

Definition 2 (Eviction sequence)**.** *An eviction sequence $\mathcal{E}^T(s_i)$ is a finite sequence $\mathcal{E}^T(s_i) = [e(s_i, t)]_{t=1}^T$ at edge server s_i over T time steps, where $e(s_i, t)$ is a set of data items selected by s_i to evict at time step t. If $e(s_i, t) = \emptyset$ then no data item is evicted from s_i at time step t.*

With Definitions 1 and 2, our problem is given as follows.

Problem 1 (Distributed caching with delayed hits)**.** *Given a problem input $\sigma = [R^T(s_i)]_{i=1}^M$, we seek to develop an online caching algorithm $\mathcal{A}$ that can output an eviction policy $\Phi_{\mathcal{A}}(\sigma) = [\mathcal{E}^T(s_i)]_{i=1}^M$ minimizing the cumulative latency of serving all requests in σ.*

C. Competitive Ratio

The competitive ratio is known as a pivotal metric for measuring the performance of an online algorithm. For an input σ, we define $\mathrm{OPT}(\sigma)$ as the latency incurred by the optimal offline algorithm, and define $\mathcal{A}(\sigma)$ as the latency incurred by an online algorithm $\mathcal{A}$. Following the standard in [3], [15], the competitive ratio characterizes an asymptotic upper bound on the worst-case ratio between $\mathcal{A}(\sigma)$ and $\mathrm{OPT}(\sigma)$.

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

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

for every problem input σ. Notably, as $\mathcal{A}(\sigma) \geq \mathrm{OPT}(\sigma)$ holds for every σ, it follows that $\mathbf{CR}(\mathcal{A}) \geq 1$.

III. Lower Bounds on the Competitive Ratio

In this section, we first establish in Theorem 1 a lower bound on the competitive ratio of any deterministic online caching algorithm, and then extend this result to prove in Theorem 2 a lower bound for any randomized online caching algorithm.

A. Lower Bound for Deterministic Algorithms

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

Remark. *We make the following two remarks.*

- *The established lower bound converges to $\Omega(KMZ)$ as W approaches 0, and it converges to $\Omega(KZ)$ as W approaches Z. These observations are consistent with the results derived for the single-cache model with delayed hits [15]. Our result generalizes the previous bound to the distributed situation.*
- *We will demonstrate that the derived performance bound is asymptotically tight by developing an algorithm approaching the bound.*

We prove Theorem 1 by constructing a hard problem input σ that not only minimizes the aggregate latency incurred by the optimal offline algorithm OPT but also maximizes the latency incurred by any deterministic online algorithm $\mathcal{A}$. To achieve this, we define two categories of requests:

- **Pure request**. A pure request accesses a data item d_j once, with no other request in $W + Z$ time steps before and after it. If a pure request leads to a cache miss (i.e., a W-miss, a $(W + Z)$-miss, or a Z-miss), the latency accrued is W, $W + Z$, or Z.
- **Bursty request.** A bursty request accesses a data item d_j for $W+Z$ successive time steps, with no other request in $W+Z$ time steps before and after it. If a bursty request results in a W-miss, the latency accrued is $W + (W - 1) + \cdots + 1 = W(W + 1)/2$, corresponding to one W-miss followed by $(W - 1)$ consecutive W-delayed hits and Z cache hits. Similarly, the latency accrued is $(W + Z)(W + Z + 1)/2$ if the request incurs a $(W + Z)$-miss, and $Z(Z + 1)/2$ if the request results in a Z-miss.

We construct a hard problem input σ by combining pure requests with bursty requests. Fig. 2 illustrates an input with $M = 3$ and $K = 3$. Suppose that the initial cache $C(s_i, 0)$ of each edge server s_i stores K data items, and the items in different edge servers are all distinct. For each edge server s_i, its input sequence $R^T(s_i)$ comprises one pure request for a data item d_j not cached in s_i's initial cache $C(s_i, 0)$, followed by several bursty requests. $R^T(s_i)$ terminates when exactly K distinct data items have been requested in the bursty requests. Data items requested by these bursty requests are all selected from the set $C(s_i, 0) \cup \{d_j\}$. Given that s_i has a cache capacity of K, there will always be a data item in $C(s_i, 0) \cup \{d_j\}$ not cached by s_i. Therefore, for any deterministic algorithm $\mathcal{A}$, the adversary deliberately makes each bursty request at s_i for the item in $C(s_i, 0) \cup \{d_j\}$ not currently in s_i's cache. For instance, in Fig. 2, during the pure request for item d_0 at s_1, if algorithm $\mathcal{A}$ accommodates d_0 by evicting item d_1 from s_1's initial cache $C(s_i, 0)$, the first bursty request in $R^T(s_1)$ will target item d_1. The first (pure) request at s_1 asks for an item not in the initial cache of any edge server. Meanwhile, to minimize the aggregate latency incurred by OPT, for all $i \geq 2$, the first (pure) request at each server s_i arises after the last bursty request at s_{i-1} and is configured to target the same

item as the latter. For example, in Fig. 2, the pure request at s_2 and the last bursty request at s_1 both target the item d_x.

We now investigate the aggregate latency incurred by OPT and $\mathcal{A}$. In OPT, upon the first (pure) request at each edge server s_i, if s_i evicts the data item not requested by any subsequent bursty requests in $R^T(s_1)$, and maintains this cache status unchanged throughout the following time steps, s_i will undergo a cache miss only on the pure request. In the example of Fig. 2, if $d_x = d_0$, at the pure request for item d_0, s_1 accommodates d_0 by evicting d_3 from its cache and subsequently keeps this cache unchanged, thus incurring a cache miss only on the pure request for d_0. In OPT, the pure request at the first edge server s_1 must be served by accessing the central server s_c, incurring a latency of $Z+W$ or Z. For all $i \geq 2$, server s_i can fetch the first requested item from its preceding server s_{i-1}, resulting in a latency of W. Hence, OPT incurs an aggregate latency of at most $Z+W+(M-1)W = Z+MW$.

In contrast, any deterministic online algorithm $\mathcal{A}$ will inevitably undergo a cache miss for every request in σ, because each data item requested in $R^T(s_i)$ is precisely the one not in s_i's cache at that time. Given that each request sequence $R^T(s_i)$ terminates when K distinct data items have been requested in the bursty requests, $R^T(s_i)$ contains at least K bursty requests, among which at least $K-1$ bursty requests target data items never cached or requested by preceding servers $s_1, s_2, \ldots, s_{i-1}$. Hence, in algorithm $\mathcal{A}$, at each edge server s_i, at least $K-1$ bursty requests are served by accessing the central server s_c, each incurring an aggregate latency of at least $Z(Z+1)/2$, i.e., including one Z-miss, $(Z-1)$ Z-delayed hits, and W cache hits. For example, in Fig. 2, at s_2, the bursty requests for data items d_6 and d_5 must be served by accessing s_c. Hence, the accumulative latency incurred by $\mathcal{A}$ is at least $(K-1)MZ(Z+1)/2 = \Omega\left(KMZ^2\right)$.

Combining the above analysis leads to Theorem 1.

B. Lower Bound for Randomized Algorithms

Theorem 2. *The competitive ratio of any randomized online caching algorithm is* $\Omega\left(\frac{MZ^2 \log K}{Z+MW}\right)$.

We establish Theorem 2 by utilizing the same structure of σ as in the proof of Theorem 1, with a key modification: each request at any edge server targets a data item that has a probability of not being cached at the server. Technically, consider any data item $d_l \in C(s_i,0) \cup \{d_j\}$, where d_j is the first data item requested in $R^T(s_i)$ at s_i. We define $p(d_l)$ as the probability that data item d_l is not cached by s_i at the current time step. Given that $|C(s_i,0) \cup \{d_j\}| = K+1$ and s_i has a cache capacity of K, we have $\sum_{d_l \in C(s_i,0)\cup\{d_j\}} p(d_l) = 1$.

Intuitively, OPT can achieve the same aggregate latency as analyzed in the proof of Theorem 1 by applying the caching policy outlined therein. Next, we investigate the aggregate latency incurred by any randomized online caching algorithm $\mathcal{A}$. For any data item d_e requested in $R^T(s_i)$ at s_i, we denote $\mathcal{M}(s_i,d_e)$ as the set of data items that have been requested in the bursty requests of $R^T(s_i)$ before item d_e was requested.

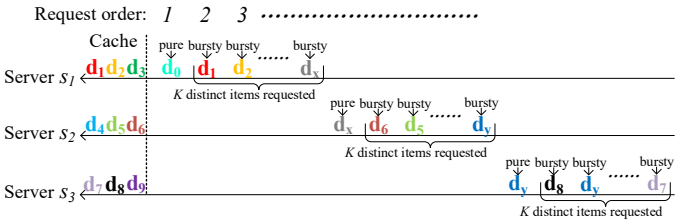

Fig. 2: A problem input σ constructed in our proof.

1) If the aggregate probability $\sum_{d_l \in \mathcal{M}(s_i,d_e)} p(d_l) = 0$, then there must exist a data item $d_g \in C(s_i,0) \cup \{d_j\} \setminus \mathcal{M}(s_i,d_e)$ with probability $p(d_g) \geq \frac{1}{K+1-|\mathcal{M}(s_i,d_e)|}$.
2) If the aggregate probability $\sum_{d_l \in \mathcal{M}(s_i,d_e)} p(d_l) > 0$, it implies that there must exist an item $d_f \in \mathcal{M}(s_i,d_e)$ that has a probability of $p(d_f) > 0$. The adversary can repeatedly request item d_f until $\sum_{d_l \in \mathcal{M}(s_i,d_e)} p(d_l) = 0$. Similarly, there will exist a data item $d_g \in C(s_i,0) \cup \{d_j\} \setminus \mathcal{M}(s_i,d_e)$ with probability $p(d_g) \geq \frac{1}{K+1-|\mathcal{M}(s_i,d_e)|}$.

In both cases, the adversary makes the next bursty request for d_g. Hence, algorithm $\mathcal{A}$ incurs an expected latency of at least $\frac{1}{K+1-|\mathcal{M}(s_i,d_e)|} \times \frac{Z(Z+1)}{2}$ on the bursty request. Recall that, under σ, algorithm $\mathcal{A}$ must access the central server s_c to serve at least $K-1$ bursty requests in each input sequence $R^T(s_i)$. With $|\mathcal{M}(s_i,d_e)|$ ranging from 1 to K, algorithm $\mathcal{A}$ incurs an expected latency of at least $\Omega\left(\frac{Z(Z+1)}{2}\mathcal{H}_K\right)$ for the $K-1$ bursty requests at s_i, where $\mathcal{H}_K$ refers to the K-th Harmonic number. Hence, it incurs an aggregate latency of $\Omega\left(M\frac{Z(Z+1)}{2}\mathcal{H}_K\right) = \Omega\left(MZ^2 \log K\right)$. Theorem 2 is proved.

IV. Algorithm Design: DLRU

The structure of performance bounds characterized in Theorems 1 and 2 unveils a key insight in designing online caching algorithms tailored to our problem: Without knowledge about other peers' cache contents, each edge server must rely on historical information to minimize data retrieval latency.

A. Algorithm Overview

Our approach to algorithm design, termed Distributed Least Recently Used (DLRU), is to integrate the aforementioned design intuition with the classic LRU policy [21]. Motivated by the provable optimality of the LRU in the single-cache model [15], we develop DLRU to extend it to our distributed caching model. Implementing this generalization, however, is non-trivial because: 1) the LRU involves a single type of data acquisition (i.e., fetching items from the central server s_c), while our DLRU needs to handle multiple types of data acquisitions inherent in distributed caching systems; 2) the LRU is not designed to handle concurrent data arrivals, a common scenario in distributed caching systems where multiple data items may arrive simultaneously at each edge server. These limitations preclude the LRU from maintaining its optimality in distributed delayed-hit caching. To tackle the challenge, DLRU mobilizes two pivotal technicalities:

- **Data acquisition strategy:** DLRU introduces two timers for each data item requested at an edge server, denoted

as TM^1 and TM^2. TM^1 tracks the elapsed time since the last successful data fetch from other peers; TM^2 tracks the elapsed time since the previous failed attempt. By mobilizing such query history information, DLRU dynamically balances trade-offs among various types of data acquisitions, minimizing the aggregate latency it incurs.

- **Data eviction policy:** To tackle the challenge of concurrent data arrivals, which LRU cannot handle effectively, DLRU refines the LRU by performing batch evictions at each edge server while ensuring its cache remains fully occupied throughout σ, optimizing cumulative latency.

Both strategies are executed in an *online* manner, i.e., operating based on the problem input σ available up to the current time step, rather than the complete σ over all T time steps.

DLRU encompasses both deterministic and randomized variants. We refer to $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ as the deterministic and randomized versions of DLRU, respectively. Both versions initiate by setting the cache contents across all M edge servers to maximize coverage of $\mathcal{D}$, i.e., the cache of each edge server is full and the contents of different caches do not overlap.

B. Deterministic Version of DLRU

Algorithm 1 presents the detailed workflow of $\mathrm{DLRU_D}$, with lines 7-27 and 28-30 outlining the data acquisition and eviction strategies, respectively.

1) Data Acquisition: From the system model, we make two important observations: (i) Given that at most 3 distinct fetched data items can arrive at an edge server simultaneously at any time step, LRU guarantees that any cached data item will persist for at least $K/3$ time steps before being evicted. (ii) The system ensures successful data retrieval within $Z+W$ time steps following any cache miss. To leverage these properties, we introduce two timers, $\mathrm{TM}^1[r(s_i,t)]$ and $\mathrm{TM}^2[r(s_i,t)]$, for each requested data item $r(s_i,t) \triangleq d_j$: (i) $\mathrm{TM}^1[r(s_i,t)]$ tracks the elapsed time since the last successful retrieval of data item d_j from edge server s_i's peers, and (ii) $\mathrm{TM}^2[r(s_i,t)]$ records the elapsed time since the last failed attempt. By the above properties, $\mathrm{TM}^1[r(s_i,t)] \leq K/3$ indicates data item d_j is likely still present in the aggregate cache of s_i's peers at time step t. $\mathrm{TM}^2[r(s_i,t)] \geq Z+W$ suggests data item d_j has likely been recached by s_i's peers at time step t. For a cache miss on a request $r(s_i,t) \triangleq d_j$, we distinguish the following cases:

1) If $\mathrm{TM}^1[r(s_i,t)] \leq K/3$ or $\mathrm{TM}^2[r(s_i,t)] > Z+W$, $\mathrm{DLRU_D}$ attempts to fetch d_j from s_i's peers. If this attempt fails, $\mathrm{DLRU_D}$ then fetches d_j from the central server s_c.
2) Otherwise, $\mathrm{DLRU_D}$ fetches d_j directly from s_c.

Case (1) corresponds to lines 15-24 in Algorithm 1, whereas case (2) corresponds to lines 25-27.

2) Data Eviction: Let $\mathcal{T}(s_i,t)$ denote the set of fetched data items that arrive at edge server s_i at time step t. Given that $\mathcal{T}(s_i,t) \leq 3$ for all $i \in [1,M]$ and $t \in [1,T]$ in the system model, the data eviction policy may need to determine not just one, but potentially multiple data items to be evicted from one edge server at one time step. We then introduce an enhanced LRU (E-LRU) function to determine which data items should be evicted from each edge server at each time step. Specifically, we use $t_l[R^T(s_i), d_j, t] := \max\{\tau < t \mid r(s_i,\tau) = r(s_i,t)\}$ to represent the last time step before time step t when data item d_j was requested in $R^T(s_i)$ at server s_i. If no such time step exists, we set $t_l[R^T(s_i), d_j, t] = -\infty$. The E-LRU function is defined below.

Algorithm 1 $\mathrm{DLRU_D}$ algorithm

1: **Input:** $\sigma = \left\{[r(s_i,t)]_{t=1}^{T}\right\}_{i=1}^{M}$, $\mathcal{S} = \{s_i\}_{i=1}^{M}$
2: **Output:** $\Phi_{\mathrm{DLRU}}(\sigma) = [\mathcal{E}^T(s_i)]_{i=1}^{M} = \left\{[e(s_i,t)]_{t=1}^{T}\right\}_{i=1}^{M}$
3: Start with a maximum coverage of $\mathcal{D}$ by $\bigcup_{i=1}^{M} C(s_i,0)$
4: **for each** $t \in [1,T]$, $s_i \in \mathcal{S}$ **do**
5: $\quad\mathcal{T}(s_i,t) \leftarrow \emptyset$ ▷ Initialize the set of fetched items
6: $\quad e(s_i,t) \leftarrow \emptyset$ ▷ Initialize the eviction set
7: **for each** $t \in [1,T]$, $s_i \in \mathcal{S}$ **do** ▷ Data acquisition
8: $\quad\Delta t \leftarrow \max\{\tau < t \mid r(s_i,\tau) = r(s_i,t)\}$
9: $\quad$**if** $\{\Delta t\} \neq \emptyset$ **then** ▷ Timer update
10: $\quad\quad\mathrm{TM}^1[r(s_i,t)] \leftarrow \mathrm{TM}^1[r(s_i,\Delta t)] + (t - \Delta t)$
11: $\quad\quad\mathrm{TM}^2[r(s_i,t)] \leftarrow \mathrm{TM}^2[r(s_i,\Delta t)] + (t - \Delta t)$
12: $\quad$**else**
13: $\quad\quad\mathrm{TM}^1[r(s_i,t)] \leftarrow 0$, $\mathrm{TM}^2[r(s_i,t)] \leftarrow 0$
14: $\quad$**if** $r(s_i,t)$ **incurs a miss then**
15: $\quad\quad$**if** $\mathrm{TM}^1[r(s_i,t)] \leq \frac{K}{3}$ **or** $\mathrm{TM}^2[r(s_i,t)] \geq Z+W$ **then**
16: $\quad\quad\quad$Attempt to fetch $r(s_i,t)$ from s_i's peers
17: $\quad\quad\quad$**if** $r(s_i,t) \in \cup_{i=1}^{M} C(s_i,t-1)$ **then** ▷ W-miss
18: $\quad\quad\quad\quad$Fetch $r(s_i,t)$ from s_i's peers
19: $\quad\quad\quad\quad$Add $r(s_i,t)$ to $\mathcal{T}(s_i,t+W)$
20: $\quad\quad\quad\quad\mathrm{TM}^1[r(s_i,t)] \leftarrow 0$
21: $\quad\quad\quad$**else** ▷ $(W+Z)$-miss
22: $\quad\quad\quad\quad$Fetch $r(s_i,t)$ from s_c at time step $t+W$
23: $\quad\quad\quad\quad$Add $r(s_i,t)$ to $\mathcal{T}(s_i,t+W+Z)$
24: $\quad\quad\quad\quad\mathrm{TM}^2[r(s_i,t)] \leftarrow 0$
25: $\quad\quad$**else** ▷ Z-miss
26: $\quad\quad\quad$Fetch $r(s_i,t)$ from s_c
27: $\quad\quad\quad$Add $r(s_i,t)$ to $\mathcal{T}(s_i,t+Z)$
28: $\quad o \leftarrow |\mathcal{T}(s_i,t)|$ ▷ Data eviction
29: $\quad e(s_i,t) \leftarrow \mathbf{ELRU}\{C(s_i,t-1) \cup \mathcal{T}(s_i,t), o\}$
30: $\quad C(s_i,t) \leftarrow C(s_i,t-1) \cup \mathcal{T}(s_i,t) \setminus e(s_i,t)$
31: **return** $\Phi_{\mathrm{DLRU}}(\sigma) = \left\{[e(s_i,t)]_{t=1}^{T}\right\}_{i=1}^{M}$

Definition 4 (E-LRU function)**.** *Given a set $\mathcal{F}$ of data items, a non-negative integer $o \leq |\mathcal{F}|$, and a certain time step t, we sort each data item $d_j \in \mathcal{F}$ in ascending order of $t_l[R^T(s_i), d_j, t]$. Let $\mathcal{F}_s$ be the sorted list. The E-LRU function $\mathbf{ELRU}\{\mathcal{F}, o\}$ returns a subset of $\mathcal{F}$ containing the first o data items in $\mathcal{F}_s$.*

Given Definition 4, the eviction policy is formally defined as follows: At any time step t, if $\mathcal{T}(s_i,t) \neq \emptyset$, s_i uses the $\mathbf{ELRU}$ function to evict $|\mathcal{T}(s_i,t)|$ items from $C(s_i,t-1) \cup \mathcal{T}(s_i,t)$ (cf. lines 28-30 in Algorithm 1). $\mathrm{DLRU_D}$ ensures that each edge server's cache remains fully occupied throughout σ.

C. Randomized Version of DLRU

$\mathrm{DLRU_R}$ applies the same data acquisition strategy as its deterministic version $\mathrm{DLRU_D}$, but mobilizes a probabilistic eviction policy implemented through a marking mechanism.

Technically, $\mathrm{DLRU_R}$ maintains a marking set $\mathcal{J}(s_i)$ of data items for each edge server s_i. Initially, $\mathcal{J}(s_i) = \emptyset$ for each edge server s_i. When a data item is requested in $R^T(s_i)$ at s_i, it is added to $\mathcal{J}(s_i)$. If adding the item to $\mathcal{J}(s_i)$ would make the size of $\mathcal{J}(s_i)$ larger than K, $\mathcal{J}(s_i)$ is reset to $\emptyset$ before adding the item, which initiates a new marking round. This process is repeated iteratively until σ terminates. At any time step t, for each edge server s_i, if $\mathcal{T}(s_i,t) \neq \emptyset$, $\mathrm{DLRU_R}$ evicts $|\mathcal{T}(s_i,t) \backslash \mathcal{J}(s_i)|$ data items from $C(s_i,t-1) \cup \mathcal{T}(s_i,t) \backslash \mathcal{J}(s_i)$ at random, i.e., adding these items to $e(s_i,t)$; otherwise, it keeps s_i's cache unchanged. Similar to $\mathrm{DLRU_D}$, $\mathrm{DLRU_R}$ maintains a fully occupied cache for each edge server throughout σ.

V. Theoretical Performance Analysis

In this section, we analytically prove in Theorem 3 that $\mathrm{DLRU_D}$ achieves a competitive ratio of $\mathcal{O}\left(\frac{KMZ^2}{Z+MW}\right)$, and demonstrate in Theorem 5 that $\mathrm{DLRU_R}$ achieves a performance guarantee of $\mathcal{O}\left(\frac{MZ^2 \log K}{Z+MW}\right)$. Both algorithms are asymptotically optimal by Theorems 1 and 2.

A. Competitive Ratio of Deterministic Version of DLRU

Theorem 3. $\mathbf{CR}(\mathrm{DLRU_D}) = \mathcal{O}\left(\frac{KMZ^2}{Z+MW}\right)$.

The fundamental challenge in proving Theorem 3 lies in decomposing each request sequence $R^T(s_i)$ into successive subsequences such that each subsequence upper-bounds the aggregate latency of $\mathrm{DLRU_D}$ while lower-bounding that of OPT. Our approach to decompose $R^T(s_i)$ builds upon two critical insights:

- In $\mathrm{DLRU_D}$, the timer and the E-LRU function guarantee that at any time step, the most recently requested item among the K distinct items in each server's cache is evicted last.
- OPT will inevitably incur at least one cache miss on a sequence of requests for $K+1$ distinct data items.

Motivated by these two observations, our proof is methodically structured into the following steps.

1) Sequence Segmentation: Our analysis commences by partitioning every input sequence into consecutive *segments*, each formalized as a contiguous interval of time steps. Concretely, for each edge server s_i, we refer to $\mathbb{S}_g[R^T(s_i)]$ as the g-th segment generated from its input sequence $R^T(s_i)$. We construct all segments from $R^T(s_i)$ by iteratively assigning each time step $t \in [1,T]$ to a segment $\mathbb{S}_g[R^T(s_i)]$ as follows:

- If $t=1$, assign t to $\mathbb{S}_1[R^T(s_i)]$;
- For each $t \in [2,T]$, if $t-1$ was assigned to $\mathbb{S}_g[R^T(s_i)]$, then check whether the set of requests in $\mathbb{S}_g[R^T(s_i)]$, combined with the request $r(s_i,t)$, reaches size $K+1$. If so, assign t to the next segment $\mathbb{S}_{g+1}[R^T(s_i)]$; otherwise, assign t to $\mathbb{S}_g[R^T(s_i)]$.

We say a data item d_j is requested in segment $\mathbb{S}_g[R^T(s_i)]$ if there is some $t \in \mathbb{S}_g[R^T(s_i)]$ such that $r(s_i,t) \triangleq d_j$.

Definition 5 (Fresh item)**.** *A data item is called a fresh item in a segment $\mathbb{S}_g[R^T(s_i)]$ ($g \geq 2$) if it is requested in $\mathbb{S}_g[R^T(s_i)]$ but was not requested in the preceding segment $\mathbb{S}_{g-1}[R^T(s_i)]$.*

By the segment definition, we naturally have the following propositions, from which Lemma 1 is subsequently derived.

Proposition 1. *The first data item requested in $\mathbb{S}_g[R^T(s_i)]$ is always a fresh item for $g \geq 2$.*

Proposition 2. *In any input sequence $R^T(s_i)$, each segment except the last one includes at least K time steps, over which exactly K distinct data items are requested.*

Lemma 1. *Given any segment $\mathbb{S}_g[R^T(s_i)]$ in input sequence $R^T(s_i)$, $\mathrm{DLRU_D}$ ensures that once a data item is requested in $\mathbb{S}_g[R^T(s_i)]$, it will not be evicted by s_i within $\mathbb{S}_g[R^T(s_i)]$.*

We prove Lemma 1 by contradiction. Suppose a data item d_j, which is requested in $\mathbb{S}_g[R^T(s_i)]$, is evicted at some time step t_e within that segment. Then, $|\mathcal{T}(s_i,t_e)| \geq 1$ and $|C(s_i,t_e-1) \cup \mathcal{T}(s_i,t_e)| \geq K+1$. By Proposition 2, there are at least $|\mathcal{T}(s_i,t_e)|$ data items in the set $C(s_i,t_e-1) \cup \mathcal{T}(s_i,t_e)$ that are not requested in $\mathbb{S}_g[R^T(s_i)]$. According to the design of $\mathrm{DLRU_D}$, it prioritizes evicting the $|\mathcal{T}(s_i,t_e)|$ least recently accessed items among all items in $C(s_i,t_e-1) \cup \mathcal{T}(s_i,t_e)$ at time step t_e. Since d_j is requested in $\mathbb{S}_g[R^T(s_i)]$, it must not be among the items evicted by $\mathrm{DLRU_D}$.

2) Segmentation Aggregation: We next combine several contiguous segments into a larger unit called a *block*. Specifically, for any edge server s_i, we define $\mathbb{B}_b[R^T(s_i)]$ as the b-th block constructed from its input sequence $R^T(s_i)$. We construct all blocks from $R^T(s_i)$ as follows:

- $\mathbb{B}_1[R^T(s_i)]$ starts with $\mathbb{S}_1[R^T(s_i)]$;
- For each block $\mathbb{B}_b[R^T(s_i)]$, we sequentially include continuous segments from $R^T(s_i)$ into $\mathbb{B}_b[R^T(s_i)]$ until the accumulative length of segments in $\mathbb{B}_b[R^T(s_i)]$ is at least $W+Z$. Then, we further add the next two segments to $\mathbb{B}_b[R^T(s_i)]$;
- The segment following $\mathbb{B}_b[R^T(s_i)]$ becomes the first segment in the subsequent block $\mathbb{B}_{b+1}[R^T(s_i)]$.

3) $\mathrm{DLRU_D}$ *Upper Bound:* Now, we turn to derive an upper bound on the aggregate latency incurred by $\mathrm{DLRU_D}$ over any block $\mathbb{B}_b[R^T(s_i)]$, as stated in Theorem 4.

Theorem 4. *For each block $\mathbb{B}_b[R^T(s_i)]$, the accumulative latency $L_{\mathbb{B}}$ incurred by $\mathrm{DLRU_D}$ over $\mathbb{B}_b[R^T(s_i)]$ satisfies: $L_{\mathbb{B}} \leq K(W+Z)(5(W+Z)+4)/2$.*

Our proof begins by introducing a parameter $t_{re}(d_j)$ for each data item d_j requested in a segment $\mathbb{S}_g[R^T(s_i)]$. Specifically, let $t_f(d_j)$ denote the time step at which d_j is requested for the first time in $\mathbb{S}_g[R^T(s_i)]$. We define $t_{re}(d_j) = t_f(d_j) + L[r(s_i,t_f(d_j))]$, where $L[r(s_i,t_f(d_j))]$ is the latency of serving the request $r(s_i,t_f(d_j)) \triangleq d_j$. Hence, $t_{re}(d_j)$ denotes the time step when d_j becomes available at edge server s_i to serve subsequent requests at s_i. Naturally, $t_{re}(d_j) - t_f(d_j) \leq W+Z$. By Lemma 1, we derived the following lemma.

Lemma 2. *Given any data item d_j requested in a segment $\mathbb{S}_g[R^T(s_i)]$, it holds that $r(s_i, t_f(d_j)) \triangleq d_j \in C(s_i, t)$ for all $t \in \mathbb{S}_g[R^T(s_i)]$ such that $t \geq t_{re}(d_j)$.*

With Lemma 2, we derive an upper bound on the aggregate latency incurred by $\mathrm{DLRU_D}$ over any segment $\mathbb{S}_g[R^T(s_i)]$.

Lemma 3. *The accumulative latency $L_{\mathbb{S}}$ incurred by $\mathrm{DLRU_D}$ over any segment $\mathbb{S}_g[R^T(s_i)]$ satisfies that: $L_{\mathbb{S}} = \sum_{t \in \mathbb{S}_g[R^T(s_i)]} L[r(s_i, t)] \leq \mathcal{L}^* K$, where $\mathcal{L}^* = \frac{\left(2(W+Z) - \min\left\{|\mathbb{S}_g[R^T(s_i)]|, W+Z\right\}+1\right) \times \min\left\{|\mathbb{S}_g[R^T(s_i)]|, W+Z\right\}}{2}$.*

Let t_q denote the maximal time step in $\mathbb{S}_g[R^T(s_i)]$. To prove Lemma 3, we consider two cases.

- $t_{re}(d_j) \geq t_q$. This case implies that $t_q - t_f(d_j) \leq t_{re}(d_j) - t_f(d_j) \leq W + Z$ and $t_q - t_f(d_j) \leq |\mathbb{S}_g[R^T(s_i)]|$. The maximum latency incurred by $\mathrm{DLRU_D}$ in serving requests for d_j over $\mathbb{S}_g[R^T(s_i)]$ is $(W+Z)+(W+Z-1)+\cdots+(W+Z-(t_q - t_f(d_j))) \leq \mathcal{L}^*$, corresponding to one $(W+Z)$-miss followed by $(t_q - t_f(d_j))$ consecutive $(W+Z)$-delayed hits.
- $t_{re}(d_j) < t_q$. This case implies that $t_{re}(d_j) - t_f(d_j) \leq t_q - t_f(d_j) \leq |\mathbb{S}_g[R^T(s_i)]|$. With $t_{re}(d_j) - t_f(d_j) \leq W+Z$, by Lemma 2, the maximum latency is $(W+Z)+(W+Z-1)+\cdots+(W+Z-(t_{re}(d_j) - t_f(d_j))) \leq \mathcal{L}^*$, corresponding to one $(W+Z)$-miss followed by $(t_{re}(d_j) - t_f(d_j))$ consecutive $(W+Z)$-delayed hits.

By combining the above two-fold analysis, the latency of serving requests for d_j over $\mathbb{S}_g[R^T(s_i)]$ is upper bounded by $\mathcal{L}^*$. Given that exactly K distinct data items are requested in $\mathbb{S}_g[R^T(s_i)]$, Lemma 3 is established by generalizing the argument for d_j to all the K distinct data items in $\mathbb{S}_g[R^T(s_i)]$.

With Lemma 3, we proceed to upper-bound the latency incurred by $\mathrm{DLRU_D}$ over any block $\mathbb{B}_b[R^T(s_i)]$, thereby proving Theorem 4. Let p denote the number of segments in block $\mathbb{B}_b[R^T(s_i)]$, and let $\mathbb{S}^j(\mathbb{B}_b)$ represent the j-th segment in $\mathbb{B}_b[R^T(s_i)]$. By the block definition, $p \geq 3$, and

$$\sum_{j=1}^{p-2} |\mathbb{S}^j(\mathbb{B}_b)| \geq W + Z \quad \text{and} \quad \sum_{j=1}^{p-3} |\mathbb{S}^j(\mathbb{B}_b)| < W + Z. \quad (1)$$

The aggregate latency $L_{\mathbb{B}}$ incurred by $\mathrm{DLRU_D}$ over any block $\mathbb{B}_b[R^T(s_i)]$ is

$$L_{\mathbb{B}} = \sum_{j=1}^{p-3} \sum_{t \in \mathbb{S}^j(\mathbb{B}_b)} L[r(s_i, t)] + \sum_{j=p-2}^{p} \sum_{t \in \mathbb{S}^j(\mathbb{B}_b)} L[r(s_i, t)], \quad (2)$$

where the term $\sum_{t \in \mathbb{S}^j(\mathbb{B}_b)} L[r(s_i, t)]$ is the aggregate latency incurred by $\mathrm{DLRU_D}$ over the segment $\mathbb{S}^j(\mathbb{B}_b)$. By Lemma 3, we have $\sum_{t \in \mathbb{S}^j(\mathbb{B}_b)} L[r(s_i, t)] \leq \mathcal{L}^* K$ for all $j \in [1, p]$. Hence, the right-hand term of Equality (2) is at most

$$\sum_{j=p-2}^{p} \sum_{t \in \mathbb{S}^j(\mathbb{B}_b)} L[r(s_i, t)] \leq 3\mathcal{L}^* K \leq \frac{3K(W+Z)(W+Z+1)}{2}, \quad (3)$$

where the last inequality holds since $\mathcal{L}^*$ reaches its maximum when $\min\left\{|\mathbb{S}^j(\mathbb{B}_b)|, W+Z\right\} = W+Z$ for all $j \in [p-2, p]$. By Inequality (1), the total length of the first $p-3$ segments in block $\mathbb{B}_b[R^T(s_i)]$ is at most $W+Z$. The left-hand term of Equality (2) attains its maximum when these segments all have the same length, i.e., $\min\left\{|\mathbb{S}^j(\mathbb{B}_b)|, W+Z\right\} = (W+Z)/(p-3)$ for all $j \in [1, p-3]$. Thus, we have

$$\sum_{j=1}^{p-3} \sum_{t \in \mathbb{S}^j(\mathbb{B}_b)} L[r(s_i, t)] \leq K\left(2(W+Z) - \frac{W+Z}{p-3} + 1\right) \times \frac{W+Z}{2} \leq \frac{K(2W+2Z+1)(W+Z)}{2}. \quad (4)$$

Combining Inequalities (3) and (4) leads to Theorem 4.

4) OPT *Lower Bound:* Next, we lower-bound the aggregate latency incurred by OPT through the following assertion.

Lemma 4. *For any input sequence $R^T(s_i)$,* OPT *must incur at least one cache miss in every block $\mathbb{B}_b[R^T(s_i)]$ except the last one.*

Lemma 4 can be proved by observing that: By Propositions 1 and 2, any block $\mathbb{B}_b[R^T(s_i)]$ (excluding the last one) in an input sequence $R^T(s_i)$ at edge server s_i involves at least $K+1$ distinct requested data items. Given that s_i has a cache capacity of K, it must experience at least one cache miss among these $K+1$ distinct data items.

5) Competitive Ratio Quantification: As the final step, we combine Theorem 4 and Lemma 4 to analytically establish an upper bound on the competitive ratio of $\mathrm{DLRU_D}$. Without loss of generality, it suffices to consider the worst-case scenario where the total number D of distinct data items requested in the problem input σ exceeds KM, and each block in σ involves at least $K+1$ requested data items. This is because: (1) if $D \leq KM$, both OPT and $\mathrm{DLRU_D}$ can eliminate latency by initially setting the combined cache contents across all M edge servers to cover all the D requested items, and (2) when an edge server encounters a block with fewer than $K+1$ data items requested, it incurs zero latency over that block if all the requested items are already in its cache. Let Q denote the average number of blocks per edge server in σ. By Lemma 4, OPT encounters at least QM cache misses over σ, of which at least $D - KM$ must be served by accessing the central server s_c, as the aggregate cache capacity across all M edge servers is KM. Hence, the latency incurred by OPT over σ is at least $(D-KM)Z + (QM - D + KM)W$. By Theorem 4,

$$\begin{aligned}
\mathbf{CR}(\mathrm{DLRU_D}) &= \frac{\mathrm{DLRU_D}(\sigma)}{\mathrm{OPT}(\sigma)} \leq \frac{MQL_{\mathbb{B}}}{\mathrm{OPT}(\sigma)} \\
&\leq \frac{KQM(W+Z)(5(W+Z)+4)/2}{(D-KM)Z + (QM - D + KM)W} \\
&\leq \frac{KM(W+Z)(5(W+Z)+4)/2}{\left(\frac{D-KM}{Q}\right)Z + \left(M - \frac{D-KM}{Q}\right)W} \\
&= \mathcal{O}\left(\frac{KMZ^2}{Z+MW}\right), \quad (5)
\end{aligned}$$

where the final asymptotic bound follows from two facts: (1) $D > KM$, and (2) the problem input σ is finite, so Q is bounded. This completes the proof of Theorem 3.

B. Competitive Ratio of Randomized Version of DLRU

Theorem 5. $\mathbf{CR}(\mathrm{DLRU_R}) = \mathcal{O}\left(\frac{MZ^2 \log K}{Z+MW}\right)$.

To prove Theorem 5, we consider any segment $\mathbb{S}_g[R^T(s_i)]$ in an input sequence $R^T(s_i)$. Let t_l denote the maximal time step in the previous segment $\mathbb{S}_{g-1}[R^T(s_i)]$. Then, $C(s_i, t_l)$ denotes the cache contents of server s_i just before $\mathbb{S}_g[R^T(s_i)]$ starts. We denote by $\mathcal{V}$ the set of data items requested in $\mathbb{S}_g[R^T(s_i)]$ that are cached in $C(s_i, t_l)$, and by $\mathcal{U}$ the set of data items requested in $\mathbb{S}_g[R^T(s_i)]$ that are not cached in $C(s_i, t_l)$. We have the following lemma.

Lemma 5. *For any segment $\mathbb{S}_g[R^T(s_i)]$ in an input sequence $R^T(s_i)$, the accumulative latency incurred by $\mathrm{DLRU_R}$ over $\mathbb{S}_g[R^T(s_i)]$ is upper-bounded by $|\mathcal{U}|\mathcal{L}^*\mathcal{H}_K$.*

We now present how to prove Lemma 5. Intuitively, $|\mathcal{V} \cup \mathcal{U}| = K$. By the design of $\mathrm{DLRU_R}$, the latency-maximizing $\mathbb{S}_g[R^T(s_i)]$ for $\mathrm{DLRU_R}$ is the one where all data items in $\mathcal{U}$ are requested before those in $\mathcal{V}$. In this scenario, before the first data item d_1 in $\mathcal{V}$ is requested, $\mathrm{DLRU_R}$ needs to perform $|\mathcal{U}|$ eviction operations. Since $\mathrm{DLRU_R}$ evicts data items randomly, we can analyze $\mathrm{DLRU_R}$ as follows:

- For the first requested data item d_1 in $\mathcal{V}$, the probability that it has been evicted from s_i over $\mathbb{S}_g[R^T(s_i)]$ is $|\mathcal{U}|/K$.
- For the second requested data item d_2 in $\mathcal{V}$, if d_1 has not been evicted, the $|\mathcal{U}|$ evictions occur among the data items in the set $C(s_i, t_l) \setminus \{d_1\}$. On the other hand, if d_1 has been evicted, there are still $|\mathcal{U}|$ evictions among the data items in the set $C(s_i, t_l) \setminus \{d_1\}$, as evicting d_1 leads to a cache miss occurring when d_1 is requested, which in turn triggers an additional eviction. In both cases, the probability that d_2 has been evicted from s_i over $\mathbb{S}_g[R^T(s_i)]$ is $|\mathcal{U}|/(K-1)$.

Generalizing this analysis to all data items in $\mathcal{V}$, we can infer that the probability that the i-th requested data item in $\mathcal{V}$ has been evicted from s_i's cache over $\mathbb{S}_g[R^T(s_i)]$ is $|\mathcal{U}|/(K-i+1)$. Hence, the expected number of cache misses incurred by $\mathrm{DLRU_R}$ on $\mathcal{V}$ is $|\mathcal{U}|(\frac{1}{K} + \frac{1}{K-1} + \cdots + \frac{1}{K-|\mathcal{V}|+1})$. As all items in $\mathcal{U}$ are not stored in $C(s_i, t_l)$, $\mathrm{DLRU_R}$ will inevitably incur a cache miss for each item in $\mathcal{U}$. Since $|\mathcal{U}| = K - |\mathcal{V}|$, the total expected number of cache misses incurred by $\mathrm{DLRU_R}$ over $\mathbb{S}_g[R^T(s_i)]$ is $|\mathcal{U}|(\frac{1}{K} + \frac{1}{K-1} + \cdots + \frac{1}{K-|\mathcal{V}|+1}) + |\mathcal{U}| = |\mathcal{U}|(\mathcal{H}_K - \mathcal{H}_{|\mathcal{U}|} + 1)$. By Lemma 3, the aggregate latency of $\mathrm{DLRU_R}$ over $\mathbb{S}_g[R^T(s_i)]$ is at most $|\mathcal{U}|(\mathcal{H}_K - \mathcal{H}_{|\mathcal{U}|} + 1)\mathcal{L}^* \leq |\mathcal{U}|\mathcal{L}^*\mathcal{H}_K$. Lemma 5 is thus established.

Moreover, the results in [21] demonstrate that the optimal offline algorithm OPT incurs an amortized number of $|\mathcal{U}|/2$ cache misses over $\mathbb{S}_g[R^T(s_i)]$. Building on a proof framework analogous to that of Theorem 3, Lemma 5 introduces three key modifications to Equation (5): (1) the numerator is multiplied by the parameter $|\mathcal{U}|$, (2) the total number of cache misses incurred by OPT is adjusted from QM to $QM|\mathcal{U}|/2$, and (3) the parameter K is replaced by $\mathcal{H}_K$. Since $|\mathcal{U}| \leq K$, these adjustments together complete the proof of Theorem 5.

VI. Experimental Evaluation

We conduct experiments to compare the performance of $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ against state-of-the-art algorithms.

A. Evaluation Methodology

Datasets. We run our experiments on the Wikimedia dataset [22], which comprises 21 upload data files and 21 text data files. From this dataset, we sample $10,000$ consecutive text requests (i.e., $T = 10,000$) for each edge server.

Baselines. We compare the performance of $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ with four baselines: LRU [21], CaLa [16], ARC [23], and Landlord [24]. LRU prioritizes items by their most recent use. CaLa is the latest online algorithm under the single-cache model with delayed hits, which schedules cached items via a ranking function. ARC is a randomized caching algorithm that balances recency and frequency to decide item eviction probabilistically. Landlord is designed for online general file caching, which maintains a credit value for each item and evicts items with zero credit. For a fair comparison, we equip these four baselines with two data acquisition strategies: (a) each edge server first retrieves items from peers, and if it fails, it requests from the central server s_c; (b) each edge server fetches items directly from s_c. For clarity, we use subscripts WZ and Z to denote strategies (a) and (b), respectively.

Experimental setup. Given that the feasibility of computing optimal offline algorithms in polynomial time remains an open question, we employ the near-optimal offline solution Belady-AD [1] to assess the competitive ratio of each simulated algorithm. The default parameter settings are as follows: $K = 5$, $M = 3$, $Z = 50$, and $W = 5$. All results presented are the average of 100 experimental runs.

B. Experiment Result

Aggregate latency. The aggregate latency comparison is presented in Figure 3. It is evident that as K increases, the aggregate latency of each algorithm decreases linearly, while it surges with growing M and Z. This is because a larger cache capacity for each edge server enhances the hit ratio. In contrast, increasing Z directly escalates the cumulative latency of accessing s_c incurred by each algorithm. Increasing M boosts the number of edge caching servers and thus the aggregate latency. Additionally, from Figure 3(c), we observe that the aggregate latency of LRU_Z, ARC_Z, $\mathrm{Landlord}_Z$, and CaLa_Z remains stable as W grows. This is because these algorithms fetch data items exclusively from s_c. In comparison, our algorithms $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ dramatically outperform the baselines. Statistically, as shown in Figure 3(a), $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ achieve an average reduction of at least 18% in aggregate latency compared to all other algorithms. In Figures 3(b), 3(c), and 3(d), the performance improvement of our algorithms reaches roughly 30%. Furthermore, $\mathrm{DLRU_R}$ is more sensitive to parameter changes than $\mathrm{DLRU_D}$, but it is slightly superior to $\mathrm{DLRU_D}$.

Competitive ratio. Figure 4 presents the comparison of the empirical competitive ratio, which is defined as the ratio between the aggregate latencies of an online algorithm and the

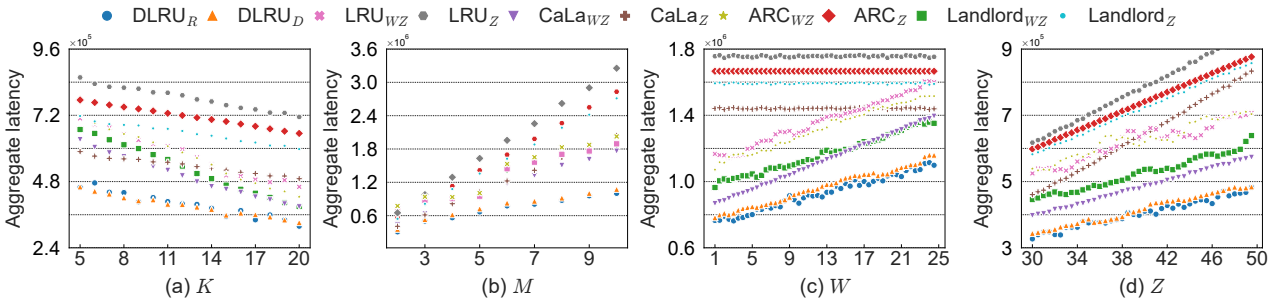

Fig. 3: Aggregate latency of simulated algorithms under various parameter configurations.

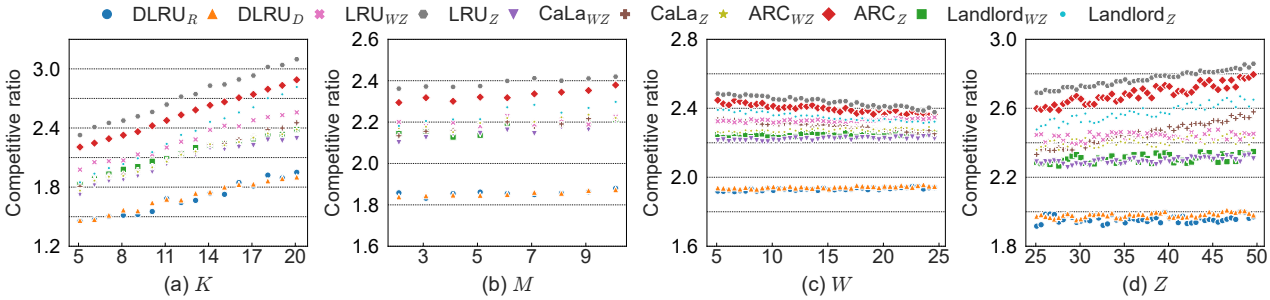

Fig. 4: Empirical competitive ratio of simulated algorithms under various parameter configurations.

near-optimal offline algorithm Belady-AD. Similar to Figure 3, Figure 4 shows a substantial enhancement in competitive ratio for $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ over the baselines. Specifically, across various parameter settings, $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ reduce the competitive ratio by at least 18% on average compared to the other algorithms, while also exhibiting less performance variation. From Figures 4(a) and 4(b), we observe that the competitive ratio of each algorithm exhibits a linear increase with K, while experiencing a slight growth with M. In Figure 4(c), as W goes up, the competitive ratios of LRU_Z, ARC_Z, $\mathrm{Landlord}_Z$, and CaLa_Z decrease rapidly, whereas those of the others increase moderately. In Figure 4(d), the competitive ratios of all algorithms grow with Z, but LRU_Z, ARC_Z, $\mathrm{Landlord}_Z$, and CaLa_Z exhibit a steeper increase than the others.

VII. Related Work

Caches [25], [26] have been investigated extensively for decades in both offline [10], [11], [13], [14], [21], [27] and online settings [12], [28]–[30]. Recently, a phenomenon, known as delayed hits, was unveiled by [1], which demonstrates latencies beyond the scope of traditional caching models. For optimization, [1] designed a near-optimal offline algorithm tailored for classical caching with delayed hits. From a theoretical perspective, [15] established a tight lower bound of $\Omega(KZ)$ on the competitive ratio of any deterministic online algorithm for delayed hits. Building on this foundation, [3] demonstrated that the LRU algorithm achieves $\mathcal{O}(KZ)$-competitiveness under the single-cache model with delayed hits, which is asymptotically optimal by the result derived in [15]. Moreover, [16] extended the study of delayed hits to the realm of online general file caching, which proved a lower bound of $\Omega(KZ)$ on the competitive ratio of any deterministic online algorithm and a lower bound of $\Omega(Z \log K)$ for any randomized online algorithm. Armed with these findings, [16] developed an algorithm with both deterministic and randomized variants. The deterministic version achieves a competitive ratio of $\mathcal{O}(Z^{3/2}K)$, whereas the randomized version achieves a competitive ratio of $\mathcal{O}(Z^{3/2} \log K)$.

VIII. Conclusion

In this paper, we have pioneered a generic algorithm analysis for distributed caching with delayed hits. We have established competitive ratio lower bounds for both deterministic and randomized online caching algorithms. We have developed a novel online caching algorithm, DLRU, encompassing a deterministic variant $\mathrm{DLRU_D}$ and a randomized variant $\mathrm{DLRU_R}$. The key technique of these variants is a timer-based data fetching scheme that leverages query history to limit aggregate data retrieval latency. We have analytically proven that both $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ achieve competitive ratios asymptotically matching the lower bounds derived. We have conducted extensive experiments using a real-world dataset, demonstrating that $\mathrm{DLRU_D}$ and $\mathrm{DLRU_R}$ outperform state-of-the-art solutions by at least 18% on average both in aggregate latency and in empirical competitive ratio. Looking ahead, our vision is to explore several crucial yet unexplored problems regarding delayed hits, including the complexity of optimal offline algorithms.

Acknowledgment

This research is supported by the Ministry of Education, Singapore, under its AcRF Tier 1 (Award RG23/23).

References

[1] N. Atre, J. Sherry, W. Wang, and D. S. Berger, "Caching with delayed hits," SIGCOMM '20, (New York, NY, USA), p. 495–513, Association for Computing Machinery, 2020.

[2] G. Quan, A. Eryilmaz, and N. B. Shroff, "Optimal edge caching for individualized demand dynamics," *IEEE/ACM Transactions on Networking*, vol. 32, no. 4, pp. 2826–2841, 2024.

[3] 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.

[4] S. Fan, I.-H. Hou, and V. S. Mai, "Dynamic regret of randomized online service caching in edge computing," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pp. 1–10, 2023.

[5] R. Zhou, X. Wu, H. Tan, and R. Zhang, "Two time-scale joint service caching and task offloading for UAV-assisted mobile edge computing," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pp. 1189–1198, 2022.

[6] E. Ghabashneh and S. Rao, "Exploring the interplay between CDN caching and video streaming performance," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 516–525, 2020.

[7] J. Chen, N. Sharma, T. Khan, S. Liu, B. Chang, A. Akella, S. Shakkottai, and R. K. Sitaraman, "Darwin: Flexible learning-based CDN caching," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, (New York, NY, USA), p. 981–999, Association for Computing Machinery, 2023.

[8] K. Zhu, Y. Zhao, Y. Gao, P. Braun, T. A. Khan, H. Litz, B. Kasikci, and S. Deng, "From optimal to practical: Efficient micro-op cache replacement policies for data center applications," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 716–731, 2025.

[9] Y. Yuan, P. Jin, *et al.*, "twcache: Thread-wise cache management with high concurrency performance," in *The 41st International Conference on Data Engineering (ICDE)*, (Beijing, China), pp. 123–134, IEEE, 2025.

[10] 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.

[11] 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.

[12] T. Lykouris and S. Vassilvitskii, "Competitive caching with machine learned advice," *J. ACM*, vol. 68, July 2021.

[13] 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.

[14] 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.

[15] P. Manohar and J. Williams, "Lower bounds for caching with delayed hits," in *arXiv*, 2020.

[16] 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.

[17] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, "Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 2499–2508, 2020.

[18] Z. Xu, L. Zhou, S. Chi-Kin Chau, W. Liang, Q. Xia, and P. Zhou, "Collaborate or separate? distributed service caching in mobile edge clouds," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 2066–2075, 2020.

[19] T. X. Tran and D. Pompili, "Octopus: A cooperative hierarchical caching strategy for cloud radio access networks," in *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 154–162, 2016.

[20] J. Bartelt, P. Rost, D. Wubben, J. Lessmann, B. Melis, and G. Fettweis, "Fronthaul and backhaul requirements of flexibly centralized radio access networks," *IEEE Wireless Communications*, vol. 22, no. 5, pp. 105–111, 2015.

[21] 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.

[22] Wikimedia Foundation, "Wikimedia caching data." Web page, 2019. Accessed on May 1st, 2025.

[23] 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.

[24] L. Epstein, C. Imreh, A. Levin, *et al.*, "Online file caching with rejection penalties," *Algorithmica*, vol. 71, pp. 279–306, 2015.

[25] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, p. 202–208, 1985.

[26] Y. Liu, Y. Mao, X. Shang, Z. Liu, and Y. Yang, "Distributed cooperative caching in unreliable edge environments," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pp. 1049–1058, 2022.

[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] G. Quan, J. Tan, and A. Eryilmaz, "Counterintuitive characteristics of optimal distributed LRU caching over unreliable channels," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 694–702, 2019.