

Sampling-Based Caching for Low Latency in Distributed Coded Storage Systems

Kaiyang Liu , *Member, IEEE*, Jingrong Wang , *Graduate Student Member, IEEE*, Heng Li , *Member, IEEE*, Jun Peng , *Senior Member, IEEE*, and Jianping Pan , *Fellow, IEEE*

Abstract—Caching has been considered as a promising solution to achieve low latency in distributed erasure coded storage systems. The previous research work categorizes all feasible caching decisions into a set of cache partitions, and then obtains the optimal solution by applying the market clearing price on each cache partition. While enjoying the ultimate performance of low data access latency, the optimal scheme suffers from high computation overheads when applied to large-scale storage systems. This paper presents SampleX, which constructs the sparsification of cache partitions through sampling to approximate the optimal caching scheme with substantially reduced computation complexity. Theoretical analysis guarantees the performance of SampleX. Furthermore, SampleX is implemented in a streaming fashion, capturing the characteristics of recent traffic for online cache content replacement. Trace-driven experimental results show that online SampleX is up to $95\times$ faster than the state-of-the-art online scheme while only incurring a performance loss of 0.81%.

Index Terms—Caching, cloud-edge storage systems, erasure codes, sampling.

I. INTRODUCTION

THE flourish of data-intensive applications, e.g., online video streaming, Big Data analytics, and social networking, has put significant burdens on the underlying storage systems. According to the prediction from International Data Corporation, the global data will grow to 175 Zettabytes by the coming year 2025 [1]. Modern distributed storage systems, e.g., Amazon Simple Storage Service (S3) [2], Google Cloud Storage [3], and Microsoft Azure [4], often use erasure codes since they provide space-optimal data redundancy for high reliability.

As one of the most widely used erasure codes, the (K, R) Reed-Solomon (RS) code generates K data chunks and R parity

chunks for each data item. Since RS codes are able to detect and correct data errors of up to R chunks, the R -fault tolerance can be achieved by placing chunks at different storage locations. Compared with other data protection methods, e.g., data replication with a minimum of $2\times$ redundancy, erasure codes significantly reduce storage costs while keeping the same or higher level of reliability [5], [6]. In the considered geo-distributed storage systems, erasure codes incur high data access latency as users access multiple remote storage nodes for data reconstruction.

As supplements to the geo-distributed storage systems, major content providers and distributors, e.g., Google and Akamai, deploy edge servers between users and Wide Area Networks (WANs) to achieve low latency [8], [9]. The edge servers have cached a pool of frequently accessed data items to serve requests from users. Although typically with limited cache capacity [13], the edge servers can be flexibly deployed according to user needs. Many existing works investigated caching policies at the data item level to increase cache hit ratio [11], [14], [15], [16], minimize service latency [12], or achieve load balancing [17], [18]. Due to the limited cache capacity, caching entire data items may not yield the lowest data access latency in distributed coded storage systems [6], [7]. The network conditions from some storage nodes may be better than others. Caching certain data chunks of some popular data items can not only alleviate the bottleneck of high data access latency but also tackle the limitations of cache capacity [19], [20].

To the best of our knowledge, we were the first to investigate the optimal scheme to determine which chunks should be cached at the edge servers for low data access latency [21]. Through iterative search, all feasible caching decisions are categorized into a set of cache partitions χ based on the number of cached chunks for each data item. The optimal caching decision can be obtained by applying the market clearing price [22] on all cache partitions, which, however, faces the challenges of high computation overheads. Guided by the offline optimal scheme, an online near-optimal scheme [21] is also designed to update the caching decision upon the arrival of each request. As the computation complexity (which is determined by the number of cache partitions) grows exponentially with the increase of K , the online near-optimal scheme works well when K is relatively small. However, in real-world storage systems, e.g., Microsoft Pelican with $K = 15$ [23], the computation overheads are considerably high. Experimental results in Section V-B show that the edge server needs tens of milliseconds computation delay to update the caching decision of each data

Manuscript received 9 October 2022; accepted 18 September 2023. Date of publication 22 September 2023; date of current version 13 December 2023. This work was supported in part by NSERC, CFI, and BCKDF. Recommended for acceptance by E. Damiani. (Corresponding author: Jingrong Wang.)

Kaiyang Liu is with the Department of Computer Science, Memorial University of Newfoundland, St. John's, NL A1B 3X5, Canada, also with the School of Computer Science and Engineering, Central South University, Changsha 410075, China, and also with the Department of Computer Science, University of Victoria, Victoria, BC V8P 5C2, Canada (e-mail: kaiyang.liu@mun.ca).

Jingrong Wang is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 1A1, Canada (e-mail: jr.wang@mail.utoronto.ca).

Heng Li and Jun Peng are with the School of Computer Science and Engineering, Central South University, Changsha 410075, China (e-mail: liheng@csu.edu.cn; pengj@csu.edu.cn).

Jianping Pan is with the Department of Computer Science, University of Victoria, Victoria, BC V8P 5C2, Canada (e-mail: pan@uvic.ca).

Digital Object Identifier 10.1109/TSC.2023.3318315

request, which is the direct consequence of high computation overheads.

To address the challenges, we propose to construct the sparsification of cache partitions through sampling on χ , reducing the computation overheads while still preserving the ultimate performance of caching on latency reduction. The main contributions are summarized as follows:

- Instead of using exhaustive search over all cache partitions for the optimal decisions, we investigate the performance of different cache partitions and identify the “dense” structure and rough periodicity of the incurred data access latencies, which motivates the design of SampleX.
- We propose a simple but efficient SampleX scheme. Preliminary experimental results demonstrate the efficacy of SampleX. The optimal scheme can be well approximated by just considering a small number of samples even with millions of cache partitions in χ .
- Theoretical analysis provides the worst-case performance guarantee of the proposed scheme SampleX.
- A prototype of the distributed coded storage system is built based on Amazon S3. Driven by real-world data request traces, experimental results show that SampleX is up to $95\times$ faster than the online near-optimal scheme while maintaining a similar low data access latency.

The rest of this paper is organized as follows. Section II reviews the related work. Section III presents the research background. Section IV presents the design of SampleX. Section V evaluates the performance of SampleX based on experiments. Section VI draws the conclusion and discusses future work.

II. RELATED WORK

Caching at the data item level: As a promising solution to achieve low latency, caching design has received a significant amount of research attention. Belady’s MIN [11] evicts the data item with the furthest next request, which has been considered as the standard offline optimal caching algorithm. However, Belady’s MIN assumes perfect knowledge of future requests, which is impractical. By considering the request recency and frequency features, a group of research efforts focus on maximizing cache hit ratios, with significant advances in recent work [14], [15], [24], [25]. These schemes rely on heuristics to maintain an ordering of the cached data items for eviction decisions. Unlike previous studies that try to optimize heuristics-based algorithms, Song et al. [16] proposed a learning-based scheme to approximate Belady’s MIN with high efficiency. Atre et al. [12] found that the hit-rate optimal Belady’s MIN may not be latency optimal under the influence of high data access latency. The minimum latency caching was constructed as a Minimum-Cost Multi-Commodity Flow problem with an efficient heuristic ranking algorithm as a solution. These previous studies focus on caching at the data item level. If the data items are encoded into multiple chunks with erasure codes, keeping a full copy of data items may not enjoy the lowest data access latency.

Caching in coded storage systems: Ma et al. [26] proposed an ensemble scheme of data replication and erasure codes to reduce storage costs and data access latencies. Previous research

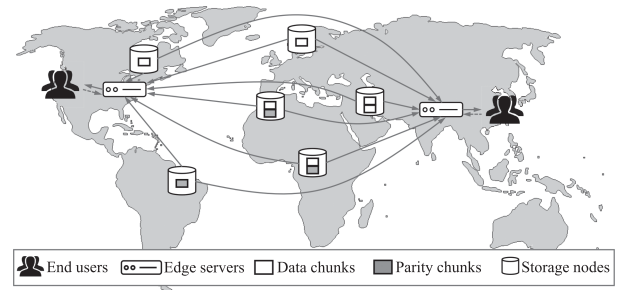


Fig. 1. Geo-distributed storage system with caching at the edge servers is shown. End users send data requests to their nearest edge servers, which have cached a pool of popular data chunks from remote storage nodes.

work [27] indicated that caching a partial number of chunks had more scheduling flexibility. Assuming an oracle about the data request rates, an analytical framework was proposed to optimize the cache contents for low latency. In coded storage systems, Al-Abbasi et al. [28] proposed a time-to-live cache scheme to jointly reduce the mean and tail service latencies. EC-Cache [29] uses online erasure coding to achieve load balanced and low latency cluster caching. Agar [7] extends the application scenario of erasure codes to geo-distributed storage systems. Agar is a dynamic programming-based scheme which pre-computes a cache configuration for a certain time with no worst-case performance guarantees. Unlike Agar, [20] proposed an adaptive content replacement scheme, which could achieve at least 50% of the maximum reducible latency in theory. Our paper [21] is the first work that explores the optimal caching solution to achieve low latency in distributed coded storage systems. The proposed SampleX is a novel extension to the optimal caching scheme [21] based on sampling, reducing the computation overheads by orders of magnitude while maintaining the performance of low data access latency.

Random sampling: Sampling is a very versatile and powerful tool for estimating vital properties of large-scale data sets by maintaining random samples [30]. Dobra et al. [31] designed a sharing scheme of query samples to improve the efficiency of stream data processing. In geo-distributed storage systems, Yu et al. [32] proposed a sampling-based scheme to simplify the formulated hypergraph of data replica placement, reducing the computation overheads. This paper applies sampling to achieve near-optimal caching in distributed coded storage systems for the first time. Unlike the previous random sampling schemes which typically have no worst-case performance guarantees, we analyze the structures of caching decisions and propose SampleX with a theoretical guarantee.

III. BACKGROUND

A. Geo-Distributed Coded Storage System

We consider a geo-distributed storage system as in Fig. 1, which contains a set of storage nodes $\mathcal{N}$ distributed at various locations (with size $N = |\mathcal{N}|$).¹ Each storage node may represent a data center in the real world. The set of stored data

¹Table I shows the major notations adopted in this paper.

TABLE I
NOTATIONS

Symbol	Definition
$\mathcal{N}$	Set of storage nodes, $N = \mathcal{N} $
$\mathcal{M}$	Set of stored data items, $M = \mathcal{M} $
K, R	Number of data and parity chunks per data item
m_k, m_r	Data chunks and parity chunks
C	Cache capacity
r_m	Request popularity of data item m
λ_m	Number of cached data chunks for m
$\tau_{m,k}$	Reduced latency when k data chunks are cached for m
l_{m_k}	Average data access latency for data chunk m_k from remote storage nodes
χ	Set of cache partitions, $\{x_1, \dots, x_K\} \in \chi$
Θ	Total amount of reduced latency
S	Number of samples
$\mathcal{T}$	Data service period
Γ	Set of data requests in $\mathcal{T}$
γ_m^t	Request to data item m at time t , $\gamma_m^t \in \Gamma$

items is denoted by $\mathcal{M}$ (with size $M = |\mathcal{M}|$). Using the same settings as in Hadoop [34], all data items (e.g., file blocks) are of the same block size. The (K, R) RS codes are adopted as the storage scheme to achieve the targeted reliability with maximum storage efficiency. With a generator matrix, RS codes divide each data item into equal sized K data chunks and generate R parity chunks. As shown in Fig. 1, the coded chunks are distributed among storage nodes.² We choose not to place the coded chunks at a single storage node because it will increase the data access latency of end users far away from the node [8].

The flow of data read is generalized as follows: when a data item is requested, the read request is fulfilled by accessing K data chunks from remote storage nodes, reconstructing the needed data item with low overheads. If the data chunks are provisionally unavailable, the needed data item can be recovered via the decoding from any K out of $K + R$ data and parity chunks. The parity chunk retrieval for data recovery is defined as degraded read, which inherently incurs a non-negligible computational [5]. Therefore, the degraded read will only be triggered when the data chunks are provisionally unavailable from the storage nodes. Furthermore, data write is not considered in this paper.³

To serve geographically dispersed end users, caching at edge servers is adopted for low data access latency. Each edge server creates a Dynamic Random Access Memory (DRAM) caching layer (with capacity C) to cache popular data chunks near end users. At each edge server, a thread pool is adopted to access data chunks in parallel. Without server failure, only data chunks will be cached to avoid degraded read. As shown in Fig. 1, instead of interacting with remote storage nodes directly, end users send data requests to the nearest edge server. In geo-distributed coded storage systems, previous studies demonstrate that caching all

²By default, data replication at the remote storage nodes is not considered to ensure low storage overheads. Our design can be extended to the scenario of data replication. If more copies of coded data chunks exist, the data request is served by fetching K data chunks from the closest storage nodes. The performance of replication is evaluated in Sec. V-C.

³The reason is that many storage systems, e.g., Content Delivery Networks (CDN), are append-only where data items are immutable. Data items with any updates are treated as disparate data items [5].

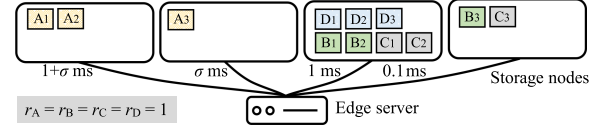


Fig. 2. Example of the distributed coded storage system with caching at the edge server is shown. Each data item is coded into $K = 3$ data chunks. The coded parity chunks are not shown as only data chunks are accessed unless data chunks are temporarily unavailable due to storage node or network failure.

chunks for each data item may not yield the lowest data access latency [7], [21]. This is mainly because in the caching services, geographically distributed data chunks contribute differently to the latency reduction. Due to the scarcity of DRAM resources at the edge servers, not all $M \cdot K$ data chunks can be cached in the caching layer. To minimize the overall data access latency, the caching scheme should determine which data chunks to cache for each data item.

B. Problem Statement

Let r_m denote the request popularity of data item m . Let L_i denote the average data access latency from storage node i to end users,⁴ $i \in \mathcal{N}$. Based on the storage location information of data chunks, the latency of chunk retrieval $\{l_{m_1}, \dots, l_{m_K}\}$ for data item m can be obtained. Data chunks are re-labeled according to the latency in descending order, i.e., $l_{m_1} \geq \dots \geq l_{m_K}$. To progressively reduce the latency, the data chunk with higher access latency will always be cached first. Let us use data item A in Fig. 2 as an example. Without caching, the access latency is determined by the chunks placed at the farthest storage node, i.e., A_1 and A_2 . When A_1 and A_2 are cached, the access latency is determined by the farthest uncached chunk, i.e., A_3 . For each data item, a $(K + 1)$ -dimensional array $\tau_m = \{\tau_{m,0}, \tau_{m,1}, \dots, \tau_{m,K}\}$ is maintained

$$\tau_{m,k} = \begin{cases} 0, & \text{if } k = 0, \\ r_m \cdot (l_{m_1} - l_{m_{k+1}}), & \text{if } k \in \{1, \dots, K-1\}, \\ r_m \cdot l_{m_1}, & \text{if } k = K, \end{cases} \quad (1)$$

where $\tau_{m,k}$ represents the total reduced latency when k data chunks are cached. As caching more data chunks for a data item is always beneficial to latency reduction for that data item, we have

$$\tau_{m,k-1} \leq \tau_{m,k}, k \in \{1, \dots, K\}. \quad (2)$$

To maximize the total amount of reduced latency Θ , we need to determine the number of cached data chunks for each data item λ_m , $\lambda_m \in \{0, \dots, K\}$, $m \in \mathcal{M}$, i.e.,

$$\begin{aligned} \max_{\lambda_m \in \mathbb{N}, m \in \mathcal{M}} \quad & \Theta(\lambda_m) = \sum_{m \in \mathcal{M}} \tau_{m,\lambda_m} \\ \text{s.t.} \quad & 0 \leq \lambda_m \leq K, \\ & \sum_{m \in \mathcal{M}} \lambda_m = C, \end{aligned} \quad (3)$$

where $\sum_{m \in \mathcal{M}} \lambda_m = C$ ensures that the cache capacity is fully utilized for latency reduction.

⁴The edge servers are deployed near end users. Compared with the high access latency over WAN (in hundreds of milliseconds), the latency from the edge server to end users is negligible.

C. Overview of the Optimal Caching Scheme

Unlike traditional studies which typically use heuristics to solve the integer caching problem for suboptimal solutions, [21] designed a novel scheme based on cache partitions and market clearing price to find the optimal caching decision for low latency. The optimal scheme is an offline solution by assuming the future network condition and data popularity information are available. The fundamentals of the method are introduced as follows:

1) *Cache partitions*: Let $\{x_1, \dots, x_K\} \in \chi$ denote a partition of caching decisions, where x_k represents the number of data items with k data chunks cached. According to the constraints in (3), $\{x_1, \dots, x_K\}$ satisfies the Diophantine equations

$$\begin{cases} x_1 + 2x_2 + \dots + K \cdot x_K = C, \\ x_1 + x_2 + \dots + x_K \leq M. \end{cases} \quad (4)$$

The complete set of cache partitions χ can be derived via iterative search. Initially, $\{x_1, x_2, \dots, x_K\} = \{C, 0, \dots, 0\}$ is a feasible solution if $C \leq M$. Then, the values of $\{x_2, \dots, x_K\}$ are iteratively increased and x_1 is set to $C - \sum_{k=2}^K k \cdot x_k$. For example, let $K = 3$ and $C = 4$. Four cache partitions, i.e., $\{4, 0, 0\}$, $\{2, 1, 0\}$, $\{0, 2, 0\}$, and $\{1, 0, 1\}$, are sequentially appended to χ .⁵ Theoretical analysis shows that the set size $|\chi| < \prod_{k=2}^K (\lfloor \frac{C}{k} \rfloor + 1)$ [21]. 2) *Market clearing price for optimal caching decision*: For a cache partition $\{x_1, \dots, x_K\}$, we should determine which data item $m \in \mathcal{M}$ should be assigned to x_k . For example, if m is assigned to x_k , the caching decision is $\lambda_m \leftarrow k$. The data items and cache partitions can be considered as sellers and buyers, respectively. Based on the valuation array in (1), each buyer has a valuation $\tau_{m,k}$ for each data item. To maximize the reduced latency, cache partitions compete for data items with higher valuations. A perfect matching between buyers and sellers can be obtained by gradually increasing the price of competed data item p_m from 0. Then, the valuation is updated with $\tau_{m,k} - p_m$. The introduction of price p_m is known as the market clearing price method, which yields the optimal decision for each cache partition [22]. Theoretical analysis shows that the computation complexity of market clearing price for a cache partition is $O(MPK)$, where P is the “potential energy” driving the pricing process.

$$P = \sum_{k=1}^K \sum_{m \in \mathcal{M}} \text{sum_top}\{\tau(m, k), x_k\}, \quad (5)$$

and function $\text{sum_top}\{\tau(m, k), x_k\}$ represents the sum of largest x_k elements. The global optimal caching decision is obtained by considering the perfect matching for all cache partitions in χ . The computation complexity of the optimal scheme is no more than $O(MPK \cdot \prod_{k=2}^K (\lfloor \frac{C}{k} \rfloor + 1))$.

⁵The iterative search can also be applied to the scenario of $C > M$ by removing the derived cache partitions with $x_1 + x_2 + \dots + x_K > M$ in χ . For example, if $M = 3$, $\{4, 0, 0\}$ will be removed.

TABLE II
NUMBER OF CACHE PARTITIONS $|\chi|$ AND THE AVERAGE RUNNING TIME (ART) OF THE OPTIMAL SCHEME WITH THE VARIATION OF CACHE CAPACITY C AND NUMBER OF CODED DATA CHUNKS K

C	40	60	80	100	120
$ \chi $	3,692	19,858	69,624	189,509	436,140
Iterative search for χ (ms)	78.2	328.8	1,254.2	3,126.2	7,020.3
Market clearing price (s)	53.0	391.6	2,239.7	7,864.0	23,746.9
K	2	4	6	8	10
$ \chi $	51	8,037	189,509	1,527,675	6,292,069
Iterative search for χ (ms)	0.01	109.8	3,126.2	28,712.5	145,534.1
Market clearing price (s)	2.0	666.3	7,864.0	49,662.9	341,572.8

D. Drawback of the Optimal Caching Scheme

In theory, the optimal scheme achieves the lowest data access latency. On the downside, however, it causes unacceptable computation overheads when applied to a large-scale storage system. According to the analysis in Section III-C, the computation complexity of the optimal scheme is mainly determined by the total number of cache partitions $|\chi|$. Table II shows $|\chi|$ with the variation of cache capacity C and number of data chunks per data item K . Fixing $K = 6$, when C increases from 40 to 120 data chunks, $|\chi|$ increases rapidly from 3,692 to 436,140. Fixing $C = 100$ data chunks, with the increase of K from 2 to 10, $|\chi|$ increases exponentially from 51 to 6,292,069. With the increase of C and K , the Average Running Time (ART) of the iterative search method for χ reaches up to hundreds of seconds. Furthermore, with millions of required iterations $|\chi|$, the ART of the market clearing price method for a round of optimization reaches up to tens of hours.⁶ The situation gets even worse when the optimal scheme is applied to real-world storage systems, e.g., Microsoft Pelican with $K = 15$ with a larger cache capacity [23]. The long ART implies that the optimal scheme reacts slowly to real-time network changes.

IV. DESIGN OF SAMPLEX

In this section, we theoretically analyze the structure of cache partitions and compare the performance of various sampling schemes, which motivates the design of SampleX, an extension to the optimal caching scheme through sampling on set χ to address the problem of high computation overheads. The streaming-based implementation of SampleX captures the statistical characteristics of the recent request traffic, ensuring that the caching problem can be solved in an online manner.

A. Theoretical Analysis of Cache Partitions

Intuitively, to obtain near-optimal caching decisions, we do not need to consider all cache partitions in χ . In this way, the computation complexity can be reduced with a decreased set size. The key challenges are 1) which cache partitions should be selected, and 2) how far is the latency reduction performance of this selection from the optimal value? To solve these challenges,

⁶Please see Section V-A for details of the experimental setup.

we explore the structure of the solution space χ with the following theoretical analysis, which guides us to construct a subset $\tilde{\chi}$ via sampling.

Definition 1: For any two cache partitions $\{x_1, \dots, x_K\}$ and $\{x'_1, \dots, x'_K\}$ in χ , if $\exists k \in \{2, \dots, K\}$, $x_{k-1} \geq x'_{k-1}$, $x_k < x'_k$, $x_{k+1} \leq x'_{k+1}, \dots, x_K \leq x'_K$ hold, it can be defined that $\{x_1, \dots, x_K\}$ is prior to $\{x'_1, \dots, x'_K\}$ in the sequence of χ .

Considering the example in Section III-C with $K = 3$ and $C = 4$, cache partition $\{2, 1, 0\}$ is prior to $\{1, 0, 1\}$. Then, we have the following Lemma.

Lemma 1: Let $\{x_1^*, \dots, x_K^*\}$ denote the optimal cache partition with the reduced latency in maximum Θ^* . For an arbitrary cache partition $\{x_1, \dots, x_K\}$ with reduced latency Θ , if $\{x_1, \dots, x_K\}$ is prior to $\{x_1^*, \dots, x_K^*\}$, the approximation ratio $\frac{\Theta}{\Theta^*}$ could be positive infinity.

Proof: We prove the statement is correct by construction. Let $K = 3$, $C = 4$, and $M = 4$. Fig. 2 shows an example of the distributed coded storage system. The data storage locations, the average data access latencies and the request popularities are provided. The valuation arrays are

$$\begin{cases} \tau_A = \{0, 0, 1, 1 + \sigma\}, \\ \tau_B = \{0, 0, 0.9, 1\}, \\ \tau_C = \{0, 0, 0.9, 1\}, \\ \tau_D = \{0, 0, 0, 1\}. \end{cases} \quad (6)$$

We assume σ is an arbitrarily large positive number. The optimal caching partition $\{x_1^*, \dots, x_K^*\}$ is $\{1, 0, 1\}$ with $\Theta^* = 1 + \sigma$. For all cache partitions prior to $\{x_1^*, \dots, x_K^*\}$, i.e., $\{4, 0, 0\}$, $\{2, 1, 0\}$, and $\{0, 2, 0\}$, we have Θ equals 0, 1, and 1.9, respectively. In this case, the approximation ratio is $\frac{\Theta}{\Theta^*} \rightarrow \infty$. $\square$

Let $\chi^{[-1]} = \{x_1^{[-1]}, \dots, x_K^{[-1]}\}$ denote the last cache partition in the sequence of χ with reduced latency $\Theta^{[-1]}$.

$$x_k^{[-1]} = \begin{cases} \lfloor \frac{C}{K} \rfloor, & \text{if } k = K, \\ 1, & \text{if } k = C \bmod K, \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

The intuition of $\chi^{[-1]}$ is to cache data items in their entirety as many as possible.⁷ Lemma 1 shows that if $\chi^{[-1]}$ is not considered, the performance could be arbitrarily bad. Theorem 1 provides the worst-case performance bound of $\chi^{[-1]}$.

Theorem 1: When only considering the last cache partition $\chi^{[-1]}$, the worst-case approximation ratio is $\frac{\Theta^{[-1]}}{\Theta^*} > \frac{1}{K+1}$.

Proof: For caching partition $\chi^{[-1]}$, let $\mathcal{M}_1^{[-1]}$ denote the set of data items where the number of cached chunks lies between 1 and $K - 1$. Let $\mathcal{M}_2^{[-1]}$ denote the set of data items with K chunks cached, which are defined as

$$\begin{cases} \mathcal{M}_1^{[-1]} = \{m \in \mathcal{M} \mid \varepsilon_m \in \{1, \dots, K - 1\}\}, \\ \mathcal{M}_2^{[-1]} = \{m \in \mathcal{M} \mid \varepsilon_m = K\}, \end{cases} \quad (8)$$

where ε_m is the caching decision derived by the market clearing price in [21]. Let $\Theta_1^{[-1]}$ and $\Theta_2^{[-1]}$ denote the amount

⁷In large-scale storage systems, the cache capacity C is larger than the number of coded data chunks K , i.e., $C \geq K$.

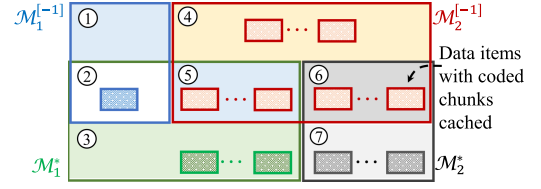


Fig. 3. Illustration of caching decision sets.

of reduced latency for set $\mathcal{M}_1^{[-1]}$ and $\mathcal{M}_2^{[-1]}$, respectively. Let $\{x_1^*, \dots, x_K^*\}$ denote the optimal cache partition with the reduced latency in maximum Θ^* . For $\{x_1^*, \dots, x_K^*\}$, $\mathcal{M}_1^*$, $\mathcal{M}_2^*$, Θ_1^* , and Θ_2^* can be defined in the same way. Fig. 3 shows an illustration of caching decision sets.

1) We consider sets $\mathcal{M}_2^*$ and $\mathcal{M}_2^{[-1]}$ first.

- For $m^*, m \in \mathcal{M}_2^{[-1]} \cap \mathcal{M}_2^*$ (denoted by set ⑥ in Fig. 3), $\tau_{m^*, K} = \tau_{m, K}$ always holds with $m^* = m$.
- If $\mathcal{M}_2^* \setminus \mathcal{M}_2^{[-1]} \neq \emptyset$ (denoted by set ⑦ in Fig. 3), for $\forall m^* \in \text{⑦}, \forall m \in \mathcal{M}_2^{[-1]} \setminus \mathcal{M}_2^*$ (denoted by set ④ $\cup$ ⑤ in Fig. 3), $\tau_{m^*, K} = \tau_{m, K}$ also holds. This is because the market clearing price outputs the optimal decision for caching partition $\chi^{[-1]}$. If $\tau_{m^*, K} > \tau_{m, K}$, $m \in \text{④} \cup \text{⑤}$ will be replaced by $m^* \in \text{⑦}$ for further latency reduction. Similarly, the market clearing price also outputs the optimal decision for $\{x_1^*, \dots, x_K^*\}$. If $\tau_{m^*, K} < \tau_{m, K}$, $m^* \in \text{⑦}$ will also be replaced by $m \in \text{④} \cup \text{⑤}$. This indicates that for $\forall m, m' \in \text{④} \cup \text{⑤}$, if $\tau_{m, K} \neq \tau_{m', K}$, we have set ⑦ = $\emptyset$.

The definition in (7) means that $x_K^* \leq x_K^{[-1]}$. With $|\mathcal{M}_2^*| \leq |\mathcal{M}_2^{[-1]}|$, we have

$$\Theta_2^* \leq \Theta_2^{[-1]}. \quad (9)$$

2) Then, we consider sets $\mathcal{M}_1^*$ and $\mathcal{M}_2^{[-1]}$.

- For $\forall m^* \in \mathcal{M}_1^* \setminus \mathcal{M}_2^{[-1]}$ (denoted by set ② $\cup$ ③ in Fig. 3), $\forall m \in \mathcal{M}_2^{[-1]} \setminus \mathcal{M}_1^*$ (denoted by set ④ $\cup$ ⑥ in Fig. 3), we have $\tau_{m^*, \varepsilon_{m^*}} \leq \tau_{m, K}$. This is because if $\tau_{m^*, \varepsilon_{m^*}} > \tau_{m, K}$, as $1 \leq \varepsilon_{m^*} \leq K - 1$, m will be replaced by m^* for further latency reduction with the market clearing price.
- According to (2), for $\forall m^*, m \in \mathcal{M}_1^* \cap \mathcal{M}_2^{[-1]}$ (denoted by set ⑤ in Fig. 3), $\tau_{m^*, \varepsilon_{m^*}} \leq \tau_{m, K}$ also holds with $m = m^*$.

Therefore, the value of $\Theta_1^*/\Theta_2^{[-1]}$ is less than the ratio of the numbers of cached data items

$$\frac{\Theta_1^*}{\Theta_2^{[-1]}} \leq \frac{\sum_{k=1}^{K-1} x_k^*}{x_K^{[-1]}}. \quad (10)$$

Then, we have the following two different cases:

1) $x_K^* \geq 1$: According to (4), we have $\sum_{k=1}^{K-1} x_k^* \leq C - K$. According to (9) and (10), the approximation ratio is

$$\frac{\Theta^*}{\Theta^{[-1]}} \leq \frac{\Theta_2^*}{\Theta_1^{[-1]} + \Theta_2^{[-1]}} + \frac{\Theta_1^*}{\Theta_2^{[-1]}} \leq 1 + \frac{C - K}{\lfloor \frac{C}{K} \rfloor} < K + 1. \quad (11)$$

2) $x_K^* = 0$: We have $\Theta_2^* = 0$. Furthermore, according to (7), the caching decision $\varepsilon_m = C \bmod K$, $m \in \mathcal{M}_1^{[-1]}$.

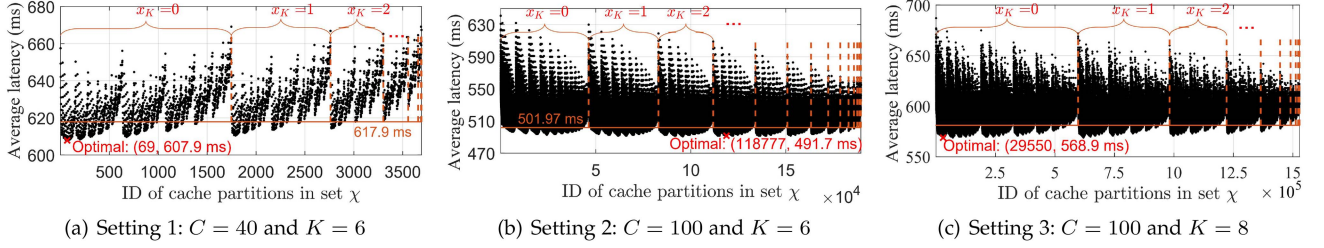


Fig. 4. Experimental results show the average data access latencies of all caching partitions in χ under various settings.

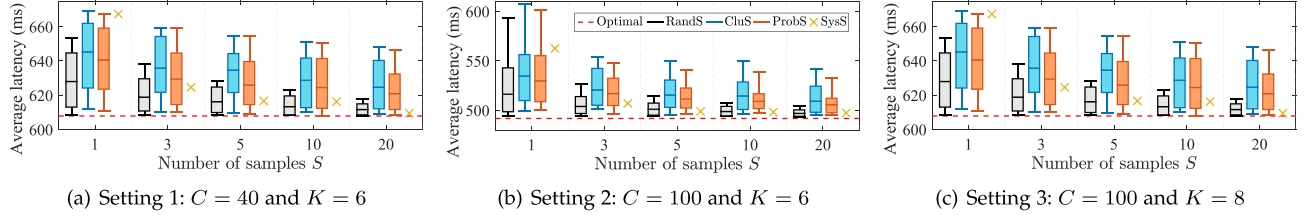


Fig. 5. Experimental results show the approximation performance of sampling schemes under various settings.

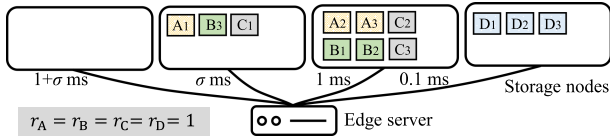


Fig. 6. Another example of the distributed coded storage system with caching at the edge server is shown. Compared with Fig. 2, the storage locations of data items are changed but still $K = 3$.

- If $\exists m^* \in \mathcal{M}_1^*$ with the caching decision $\varepsilon_{m^*} > C \bmod K$, we have $\sum_{k=1}^{K-1} x_k^* < C - C \bmod K + 1$. The approximation ratio is

$$\frac{\Theta^*}{\Theta^{[-1]}} = \frac{\Theta_1^*}{\Theta_1^{[-1]} + \Theta_2^{[-1]}} < \frac{C - C \bmod K + 1}{\lfloor \frac{C}{K} \rfloor} < K + 1. \quad (12)$$

- If $\forall m^* \in \mathcal{M}_1^*$ with the caching decision $1 \leq \varepsilon_{m^*} \leq C \bmod K$, we have $\tau_{m^*, \varepsilon_{m^*}} \leq \tau_{m, \varepsilon_m}, \forall m \in \mathcal{M}_1^{[-1]} \cup \mathcal{M}_2^{[-1]}$. The approximation ratio is less than the ratio of the numbers of cached data items

$$\frac{\Theta^*}{\Theta^{[-1]}} = \frac{\Theta_1^*}{\Theta_1^{[-1]} + \Theta_2^{[-1]}} \leq \frac{C}{\lfloor \frac{C}{K} \rfloor + 1} < K. \quad (13)$$

According to (11), (12), and (13), we can conclude that $\frac{\Theta^{[-1]}}{\Theta^*} > \frac{1}{K+1}$. $\square$

We use a simple example in Fig. 6 to demonstrate that the approximation ratio is a fairly tight bound. Compared with Fig. 2, we change the storage locations of data items. Furthermore, we set the cache capacity $C = 3$. The valuation arrays are

$$\begin{cases} \tau_A = \{0, \sigma - 1, \sigma - 1, \sigma\}, \\ \tau_B = \{0, \sigma - 1, \sigma - 1, \sigma\}, \\ \tau_C = \{0, \sigma - 1, \sigma - 1, \sigma\}, \\ \tau_D = \{0, 0, 0, 0.1\}. \end{cases} \quad (14)$$

As σ is an arbitrarily large positive number, the optimal caching partition $\{x_1^*, \dots, x_K^*\}$ is $\{3, 0, 0\}$ with $\Theta^* = 3\sigma - 3$.

If only the last cache partition $\chi^{[-1]} = \{0, 0, 1\}$ is considered, we have $\Theta^{[-1]} = \sigma$. This means $\frac{\Theta^{[-1]}}{\Theta^*} = \frac{\sigma}{3\sigma - 3} > \frac{1}{3} > \frac{1}{K+1}$ as $K = 3$. The approximation performance can be further improved if more than one sample can be considered.

B. Performance Comparison of Sampling Schemes

Beyond the theoretical analysis above, we investigate the structure of cache partitions and then compare the performance of four sampling schemes through preliminary experiments.⁸ Fig. 4 illustrates the average data access latencies for all caching partitions in χ under three different settings selected from Table II. Based on experimental results, we have the following two observations: 1) The average data access latencies of all cache partitions show periodicity, although the patterns are unstable and change under different settings. Due to the periodicity, the variation interval of data access latencies incurred by all cache partitions is bounded. A “dense” structure of cache partitions can be observed in the real-world storage system. 2) The optimal caching decision is influenced by the settings, i.e., cache capacity C , number of coded data chunks K , and valuation array τ_m . Although the periodicity can be observed, it is hard to use this information to directly obtain the optimal caching decisions as the periodicity changes under different settings. Therefore, we choose to achieve near-optimal performance via sampling, with no need to consider all cache partitions. The following four sampling schemes are used to construct the cache partition subset $\tilde{\chi}$.

Random sampling (RandS): Let $S = |\tilde{\chi}|$ denote the predefined number of samples. We randomly select S samples from χ .

Cluster sampling (CluS): In this scheme, we utilize the periodicity of caching partitions. First, we select a cluster by

⁸Please see Section V-A for details of the experimental setup. The storage nodes are deployed over $N = 6$ Amazon Web Services (AWS) regions. The edge server is deployed in Victoria, BC, Canada.

randomly choose $x_K \in \{0, \dots, \lfloor \frac{C}{K} \rfloor\}$. Then, S samples are randomly chosen from the selected cluster.

Probabilistic sampling (ProbS) [32]: Similar to CluS, we utilize the periodicity by randomly choosing S samples from a cluster. Fig. 5 shows that the cache partitions with lower data access latencies are more likely to be in the front part of χ . Different from CluS, the cluster $x_K \in \{0, \dots, \lfloor \frac{C}{K} \rfloor\}$ is chosen according to the probability

$$p_{x_K} = \frac{(\lfloor \frac{C}{K} \rfloor + 1) - x_K}{1 + 2 + \dots + (\lfloor \frac{C}{K} \rfloor + 1)}, \quad (15)$$

which means the cluster in the front part is assigned with a higher selection probability.

Systematic sampling (SysS) [33]: It is a deterministic scheme. The sequence of χ is divided into S equal-sized intervals. The last cache partition in each interval will be selected. For example, let $|\chi| = 100$ and $S = 4$, the 25-th, 50-th, 75-th, and 100-th cache partitions are added to set $\tilde{\chi}$.

After the construction of cache partition subset $\tilde{\chi}$, the market clearing price method is adopted to calculate the amount of reduced latency Θ for all samples in $\tilde{\chi}$. The caching decision λ_m is obtained by selecting the caching partition with the highest latency reduction, $m \in \mathcal{M}$.

Experimental results in Fig. 5 show the approximation performance of data access latencies incurred by the four sampling schemes. Surprisingly, RandS performs better than CluS and ProbS. The main reason is that the periodicity is unstable and may change under different settings. It is easy to select a cluster with high data access latency. In contrast, RandS takes better advantage of the “dense” structure for performance approximation. Fig. 5 illustrates the average data access latency of RandS over 100 rounds of execution and its 90% confidence interval. As expected, more samples improve the approximation performance but have diminishing returns. With $S = 5$ samples, RandS yields the average data access latencies of 616.23 ms, 501.03 ms, and 580.41 ms for the three different settings, respectively. When compared with the offline optimal scheme, RandS only incurs a performance loss of 1.37%, 1.89%, and 2.02%, respectively. Fig. 5 also shows that RandS performs better than SysS.

C. Offline SampleX

Theoretical analysis and experimental results motivate the design of SampleX.

- The last cache partition $\chi^{[-1]}$ is added into set $\tilde{\chi}$, ensuring that the offline SampleX scheme has a worst-case performance guarantee under any circumstances, e.g., the example in Lemma 1. If $S = 1$, only $\chi^{[-1]}$ will be considered.
- For the remaining cache partitions in set $\chi \setminus \{\chi^{[-1]}\}$, we rely on RandS to construct $\tilde{\chi}$ as it only incurs a performance loss of about 2%.

Although with good approximation performance, the main drawback of RandS is also the long computation delay because the iterative search for χ introduces non-negligible computation overheads. As the market clearing price method needs $O(MPK)$ steps to finish, the computation complexity of the

TABLE III
ART OF OFFLINE SAMPLING SCHEMES (MS) UNDER VARIOUS SETTINGS

S		1	2	3	4	5	10	20
RandS	Set. 1	82.1	99.4	116.3	133.2	154.2	232.8	397.5
	Set. 2	3,133.8	3,210.3	3,287.6	3,333.9	3,421.3	3,885.0	4,612.2
	Set. 3	28,701.5	28,881.4	28,981.9	28,986.5	29,139.3	29,617.8	30,648.2
SampleX	Set. 1	3.2	21.4	36.7	53.6	74.2	153.6	317.5
	Set. 2	4.4	77.4	145.9	206.4	297.4	754.3	1,571.1
	Set. 3	6.9	84.7	234.3	273.9	396.3	824.6	1,763.2

Algorithm 1: Offline SampleX.

Input: Valuation array τ , number of samples S .

Output: Subset $\tilde{\chi}$ and caching decision λ_m .

Initialization: $\forall \lambda_m \leftarrow 0, \tilde{\chi} \leftarrow \emptyset$.

- 1: Calculate the last cache partition $\chi^{[-1]}$ based on (7) and add it to set $\tilde{\chi}$;
- 2: **while** $|\tilde{\chi}| < S$ **do**
- 3: $C' \leftarrow C$;
- 4: Randomly shuffle $\mathcal{K} = \{2, \dots, K\}$;
- 5: **for** $k \in \mathcal{K}$ **do**
- 6: Randomly set x_k from $\{0, \dots, \lfloor \frac{C}{k} \rfloor\}$;
- 7: $C' \leftarrow C' - x_k \cdot k$;
- 8: **end for**
- 9: $x_1 = C'$;
- 10: Add $\{x_1, \dots, x_K\}$ to $\tilde{\chi}$ if $\{x_1, \dots, x_K\} \notin \tilde{\chi}$;
- 11: **end while**
- 12: **for** cache partition $\{x_1, \dots, x_K\} \in \tilde{\chi}$ **do**
- 13: Invoke the market clearing price method in [21] to obtain temporary caching decision $\tilde{\lambda}_m, m \in \mathcal{M}$;
- 14: $\forall \lambda_m \leftarrow \tilde{\lambda}_m$ if $\sum_{m \in \mathcal{M}} \tau_{m, \lambda_m} < \sum_{m \in \mathcal{M}} \tau_{m, \tilde{\lambda}_m}$;
- 15: **end for**

market clearing price method is $O(SMPK)$. With the set size $|\chi| < \prod_{k=2}^K (\lfloor \frac{C}{k} \rfloor + 1)$, the computation complexity of RandS is $O(\prod_{k=2}^K (\lfloor \frac{C}{k} \rfloor + 1) + SMPK)$. So in the design of SampleX, we go one step further by implementing RandS with no need to acquire the complete set of χ , which can further reduce the computation overheads.

The pseudo code of SampleX is listed in Algorithm 1. The random shuffling and value selection (Line 2–11) ensure that each caching partition has the same probability to be selected in $\tilde{\chi}$ with the computation complexity of $O(SK)$. The computation complexity of SampleX is reduced to $O(SMPK)$. As shown in Table III, compared with RandS, SampleX can reduce the computation overhead significantly without sacrificing the approximation performance. With $S = 5$ samples, SampleX yields the average data access latencies of 617.90 ms, 501.97 ms, and 581.13 ms for the three different settings, respectively. Compared with the offline optimal scheme, SampleX only incurs a performance loss of 1.65%, 2.08%, and 2.11%, respectively. As shown in Tables II and III for the three selected settings, the ART is reduced by $715\times$, $26,453\times$, and $125,389\times$, respectively. Results on other settings in Table II are not shown but are similar.

As shown in Table II, the offline optimal schemes may contain millions of caching partitions. Then, we explain why SampleX can approximate the optimal scheme well with only several

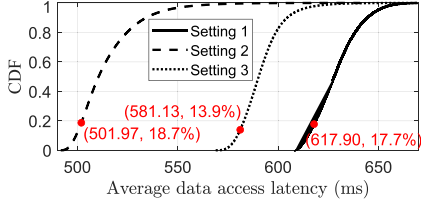


Fig. 7. CDF of the average data access latencies for all cache partitions with SampleX.

samples. Fig. 7 shows the Cumulative Distribution Function (CDF) of the average data access latencies for all cache partitions in set χ . If we randomly choose a sample from set $\chi \setminus \{\chi^{[-1]}\}$, the probabilities p that the incurred data access latencies below 617.90 ms, 501.97 ms, and 581.13 ms (i.e., the average performance of SampleX with $S = 5$) are 17.7%, 18.7% and 13.9%, respectively. As offline SampleX selects the caching partition with the highest latency reduction, the probabilities that the incurred data access latencies below 617.90 ms, 501.97 ms, and 581.13 ms can be calculated by $1 - (1 - p)^{S-1}$, i.e., 54.2%, 56.3%, and 45.1%, respectively.

So far, our discussion is based on the assumption that future data popularity and network condition information are known in advance. The significance of offline SampleX is that it inspires the design of an online scheme via sampling.

D. Online SampleX

Finally, we turn to the online setting, where we do not have the knowledge of future data popularity and network condition information. Fortunately, we can predict the future from the past. Let $\mathcal{T}$ denote the data service period. With the streaming-based implementation, the online SampleX captures the characteristics of the recent traffic, updating the caching decisions upon the arrival of each data request.

Data popularity: Exponentially Decayed Counter (EDC) [16] approximately counts the number of requests for each data item, which has been adopted to estimate future data popularity r_m^t in real time, $t \in \mathcal{T}$. Initially, r_m^t is set to 0. Then, Δ_m is updated upon the arrival of each request for data item m , which denotes the amount of time since m was last requested. Please note that many successful heuristics apply Δ to update caching decisions, such as least recently used (LRU) and its variants. Then, r_m^t is updated as follows

$$r_m^t = 1 + r_m^t \cdot 2^{-\frac{\Delta_m}{2\alpha_r}}, \quad (16)$$

where α_r is a predefined discount factor to reduce the impact of previous requests. In contrast with LRU which only cares about the latest data request, EDC can accurately predict the decay rate of data popularity over a longer period, which has been widely applied in data block caching [16], [20], [35] and video popularity estimation [36]. Another advantage is that EDC only requires $O(1)$ space to maintain the prediction information for each data item.

Network condition: To identify the future trend of network latency, Exponentially Weighted Moving Average (EWMA) [39] is employed for analyzing the time series of data requests [37].

Then, L_i^t is updated after a data read operation from the edge server to node i

$$L_i^t = \alpha_l \cdot L_i^t + (1 - \alpha_l) \cdot \iota_i, \quad (17)$$

where ι_i is the measured end-to-end data access latency, and α_l is a predefined discount factor. Similar to EDC, EWMA also needs $O(1)$ space to maintain the prediction for each storage node. To achieve low implementation overheads, we apply the long-tested EDC and EWMA schemes to predict future data popularity and network latency information. Recent research advances in future information prediction, e.g., Least Hit Density (LHD) [15] and Hyperbolic [24], are also applicable to our solution.

The online caching decision λ_m^t is updated in real time according to the maintained data popularity r_m^t and network latencies L_i^t , $t \in \mathcal{T}$. The set of all data requests in period $\mathcal{T}$ is denoted by Γ . When a data item m is requested, the valuation array $\tau_m = \{\tau_{m,0}, \tau_{m,1}, \dots, \tau_{m,K}\}$ is updated according to (16) and (17) first. Then, the online scheme needs to determine 1) whether m should be cached, and 2) which data items in the caching layer are replaced. To ensure adaptivity, the challenge is how to reduce the computation complexity of online caching schemes while maintaining the performance of low data access latency. An intuitive idea is to select a subset of replacement candidates in the caching layer, without completely overriding the existing decisions. For instance, Hyperbolic [24] and Learning Relaxed Belady (LRB) [16] obtain replacement candidates through random selection, which may lead to arbitrarily bad performance. In contrast, the online near-optimal scheme [21] proposed a greedy scheme to generate the replacement candidates, with which the worst-case performance guarantee can be derived.

Online near-optimal scheme [21]: Let $\hat{\mathcal{M}}'$ denote the set of cache replacement candidates. The data items with the lowest valuations per unit (i.e., $\arg \min \frac{\tau_{n,t}}{\lambda_n^t}$) in the caching layer are successively added into $\hat{\mathcal{M}}'$. This is because the data items in $\hat{\mathcal{M}}'$ are expected to be replaced first by the requested m to reduce the data access latency. Data item m is also added into $\hat{\mathcal{M}}'$. The expansion of $\hat{\mathcal{M}}'$ needs $K + 1$ iterations at most (with $|\hat{\mathcal{M}}'| \leq K + 1$) to ensure all K data chunks of m have a chance to be cached. The iterative search algorithm calculates the cache partition set $\hat{\chi}$ for $\hat{\mathcal{M}}'$. Based on subset $\hat{\mathcal{M}}'$ and $\hat{\chi}$, the market clearing price method is invoked to update the caching decisions λ_n^t , $n \in \hat{\mathcal{M}}'$. Theoretical analyses show that the approximation ratio is $1 - \frac{2K-1}{C}$. With $|\hat{\chi}| < K!$, the computation complexity is less than $O(K^2 PK!)$ in handling a data request.

Experimental results in [21] show that the online near-optimal scheme performs well when K is set to a relatively small number, e.g., $K = 6$. When K is set to a larger number, e.g., $K = 15$ in Microsoft Pelican, considerable computation overheads may be encountered when updating caching decisions. Table IV shows that the maximum number of required iterations $|\hat{\chi}|$ in theory increases quickly with the increase of K . Experimental results in Section V-B also confirm our conclusion. With the increase of K from 6 to 15, the ART of the online near-optimal scheme increases rapidly from 2.21 ms to 27.41 ms to update the caching

TABLE IV
MAXIMUM NUMBER OF REQUIRED ITERATIONS $|\hat{\chi}|$ IN THEORY WITH
THE VARIATION OF K

K	2	4	6	8	10	15	18
$\max \hat{\chi} $	2	9	37	127	378	3,920	13,284

Algorithm 2: Online SampleX.

Input: Valuation array τ and number of samples S .

Output: Online caching decision $\lambda_m^t, m \in \mathcal{M}$.

Initialization: $\forall \lambda_m^t \leftarrow 0$.

```

1: for data request  $\gamma_m^t \in \Gamma, t \in \mathcal{T}$  do
2:   Update  $\{\tau_{m,0}, \tau_{m,1}, \dots, \tau_{m,K}\}$  based on (16)
   and (17);
3:   if  $\sum_{n \in \mathcal{M}} \lambda_n^t \leq C - K$  and  $\lambda_m^t < K$  then
4:      $\lambda_m^t \leftarrow K$ , add  $m$  to  $\hat{\mathcal{M}}$ ;
5:   else if  $\sum_{n \in \mathcal{M}} \lambda_n^t > C - K$  and  $\lambda_m^t < K$  then
6:     Invoke the online near-optimal scheme in [21]
     to obtain data item subset  $\hat{\mathcal{M}}'$ ;
7:     Invoke Algorithm 1 to obtain samples  $\tilde{\chi}$  (let
        $\tilde{\chi} = \hat{\chi}$  if  $|\hat{\chi}| < S$ ) and online caching decision
        $\lambda_m^t$ ;
8:   end if
9: end for

```

decision for each data request. The long computation delay indicates that high computation overhead is indeed encountered with the online near-optimal scheme, especially when data items are intensively requested. For example, the edge server needs to handle tens of thousands of requests per second in CDN [16]. The high computation overheads may be unacceptable for the resource limited edge servers.

Our observation in Section IV-C is that several samples are enough to well approximate the optimal decision. Therefore, we can also reduce the computation complexity of the online near-optimal scheme through the proposed sampling scheme in Algorithm 1. The pseudo code of the online SampleX scheme is listed in Algorithm 2. With S samples, the computation complexity of the online SampleX scheme is only $O(K^2PS)$ in handling a data request. Please note that online SampleX extends the online near-optimal scheme [21] (with approximation ratio $1 - \frac{2K-1}{C}$) through the proposed sampling scheme in Algorithm 1. If only the last cache partition $\chi^{[-1]}$ is considered, the approximation ratio of is $(1 - \frac{2K-1}{C}) \cdot \frac{1}{K+1}$.

V. PERFORMANCE EVALUATION

In this section, a prototype of the distributed coded storage system is built based on Amazon S3. Extensive experiments are conducted by using Python to evaluate the performance of SampleX.

A. Experimental Setup

The geo-distributed coded storage system contains $N = 6$ storage nodes, which are deployed over six Amazon Web Services (AWS) regions, i.e., Tokyo, Ohio, Ireland, São Paulo,

Oregon, and Northern California. Each storage node creates fifteen buckets, each of which represents a server for remote data storage. The storage system is populated with $M = 1,000$ data items [28], [40]. The `zfec` [41] library is used for the RS codes implementation. The coded chunks of each data item are of the same block size 1 MB [7], which are uniformly distributed among six storage nodes to achieve load balancing.

To show the geographically dispersed feature of requests, three edge servers are deployed on personal computers at various locations, i.e., Victoria (V), Canada, San Francisco (SF), United States, and Toronto (T), Canada. Each personal computer is equipped with an Intel(R) Core(TM) i7-7700 HQ processor and 16 GB of memory. Memcached [38] module is used adopted for data caching in DRAM. To request data chunks in parallel, a thread pool is created on each edge server. The data services last for an hour ($\mathcal{T} = 1$).

MSR Cambridge Traces [42]: These are production traces gathered from Microsoft Research, which have been extensively adopted for the performance evaluation of caching schemes [12], [15]. In this paper, these traces are used as the workloads of data requests, but not the data sizes as they are not collected from coded storage systems. All data items are assumed to be the same size.

Besides the offline optimal scheme and the online near-optimal scheme in [21], Agar [7] is also introduced as the baseline for a fair performance comparison. Agar is a dynamic programming-based caching scheme which iteratively caches data chunks with larger request rates and higher data access latencies. Agar can be considered as an offline scheme as it optimizes the caching configuration for all data items in the storage system.

B. Prototype Evaluation on Production Traces

Does the online SampleX approximate the offline optimal scheme well? To answer this question, the average request latencies provided by the five caching schemes are compared by replaying the given traces. For simplicity, the average latency represents that of all data requests from three edge servers in the following. We set the cache capacity $C = 100$ data chunks on three edge servers. The numbers of coded data and parity chunks per data item are set to $K = 6$ and $R = 3$, respectively. This case is considered as the offline optimal scheme is only computationally feasible at a small scale (see Table II). With the offline optimal scheme, the average request latency is 501.71 ms.

Through sampling on cache partitions, offline SampleX achieves close-to-optimal performance. With the increase of S from 1 to 20, the average request latency decreases from 562.52 ms to 507.12 ms. Furthermore, as shown in Fig. 8 and Table V, offline SampleX (with $S = 5$) is about $2.6\times$ faster than Agar with smaller request latency. The main drawback of offline schemes is that they require future request popularity of all data items and network condition information, which may not be practical in real-world storage systems. Therefore, we mainly focus on online caching schemes in the following parts of the paper. As shown in Fig. 8, compared with the offline optimal scheme, the online near-optimal scheme only incurs a

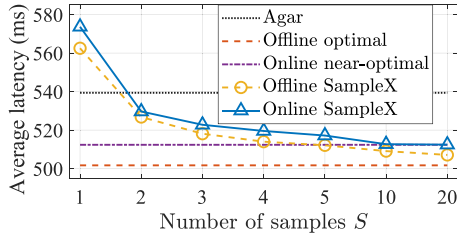


Fig. 8. Average data access latencies with $C = 100$ and $K = 6$.

TABLE V
ART OF OFFLINE SCHEMES AND THE ART OF ONLINE SCHEMES UPON THE
ARRIVAL OF EACH DATA REQUEST IN THE SCENARIO WITH $C = 100$,
 $K = 6$ AND $C = 1,000$, $K = 15$

$C = 100, K = 6$									
Offline (ms)	Agar	773.20							
	Optimal	7,864,032.12							
	SampleX	1	2	3	4	5	10	20	
Online (ms)	Near-optimal	2.21							
	SampleX	0.66	0.99	1.19	1.30	1.48	1.87	1.97	
$C = 1,000, K = 15$									
Offline (s)	Agar	10.19							
	SampleX	0.60	1.13	2.02	2.59	3.23	6.52	13.23	
	Near-optimal	27.41							
Online (ms)	SampleX	0.31	0.43	0.52	0.60	0.68	1.07	1.77	

performance loss of 2.13%. By applying sampling to the online near-optimal scheme, the online SampleX scheme also achieves close-to-optimal performance. With the increase of S from 1 to 20, the average request latency decreases from 573.67 ms to 512.47 ms.

With no need to completely override the existing caching decisions, the average and the maximum number of required iterations of the online near-optimal scheme $|\hat{\chi}|$ are 5.62 and 32, respectively. The ART is 2.21 ms upon the arrival of each data request. Few extra delays will be introduced to handle intensive data requests. As shown in Table V, the online SampleX scheme can only slightly decrease the ART in the small-scale scenario as online near-optimal is already very fast.

What are the overheads of using the online near-optimal and online SampleX schemes? We then consider a more realistic setting with $C = 1,000$ data chunks and $K = 15$.⁹ As expected, online SampleX achieves a similar performance to the online near-optimal scheme. As can be seen in Fig. 9(a), the performance loss is 1.72% when $S = 5$. Please note that due to the nature of random sampling, online SampleX can have better performance than the offline one in some scenarios.

In the current setting, the average and the maximum number of required iterations of the online near-optimal scheme $|\hat{\chi}|$ are 52.10 and 1,140, respectively. The ART increases to 27.41 ms. Fig. 9(b) shows that as the number of required iterations is bounded by S , considerable computation overheads can be reduced by the proposed online SampleX scheme. As can be

seen in Table V, the ART of online SampleX can be reduced to 0.68 ms when $S = 5$. Please note that the ART of online SampleX is 1.48 ms when $C = 100$ and $K = 6$. This is because, with the increase of cache capacity from 100 to 1,000, the cache hit ratio increases from 20.88% to 67.27%. When a data request arrives, the caching decision will not be updated if the data item is already cached, i.e., nearly no overhead is introduced. This means online SampleX is much more scalable for a large-scale storage system than the online near-optimal.

More importantly, Fig. 9(c) shows that online SampleX can reduce the tail latencies of scheme running time. The 99-th and 99.99-th percentile tail latencies of the online near-optimal scheme reach 549.88 ms and 1,119.56 ms, respectively. In contrast, when $S = 5$, the 99-th and 99.99-th percentile tail latencies of the online SampleX are reduced to 6.08 ms and 12.67 ms, respectively. The online SampleX is much more efficient to handle intensive requests without incurring much performance loss even in the worst cases.

Beyond the data access latency and ART, Fig. 10 compares the throughput with various caching schemes under two different settings. Experimental results show that compared with Agar, Online SampleX ($S = 20$) increases the throughput by 5.25% with $C = 100$ and $K = 6$. Under the setting of $C = 1,000$ and $K = 15$, the throughput of all caching schemes increases as more requests can be served locally. Compared with Agar, Online SampleX ($S = 20$) increases the throughput by 7.45% in this case.

C. Impact of Factors

To fully evaluate the performance of various caching schemes, the impacts of factors, i.e., cache capacity, number of coded data chunks, number of data replicas, number of data items, and server failure, are considered to fully evaluate the performances of various caching schemes. By default, we set the cache capacity $C = 1,000$ data chunks. The number of data items is set to $M = 1,000$. Without considering data replication, the numbers of coded data and parity chunks per data item are set to $K = 15$ and $R = 3$, respectively. With an overall consideration of data access latencies and computation overheads, the number of samples is set to $S = 5$.

Cache Capacity C : In this experiment, the cache capacity of each edge server is increased from 800 to 4,000 chunks (i.e., with cache capacity ranging from 800 MB to 4 GB at the resource-limited edge server). With more data requests enjoying the benefits of caching, the average data access latency decreases. Fig. 11 illustrates that the two online caching schemes achieve similar performances under the variation of C . Compared with the online near-optimal scheme, the performance loss incurred by online SampleX ranges from 0.70% to 1.48%. With the increase of cache capacity, the cache hit ratio of online SampleX increases from 63.02% to 89.41%. With more requested data items already in the caching layer, the caching decision will be less frequently updated upon the arrival of requests. As shown in Table VI, the ART of online SampleX decreases from 0.75 ms to 0.36 ms. For the same reason, the ART of the online near-optimal scheme

⁹In the larger scale scenario, the offline optimal scheme is not included in the comparison due to its prohibitively high computation overheads.

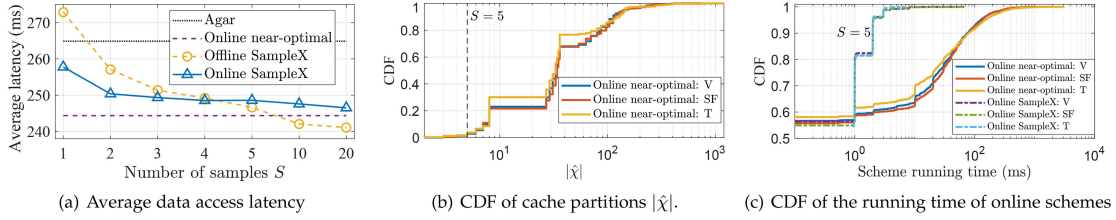
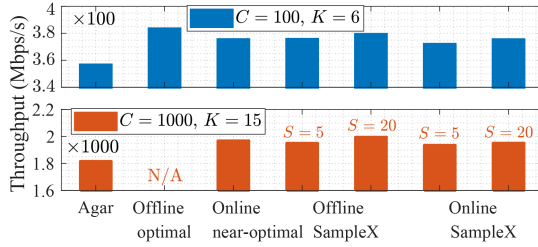
Fig. 9. Experimental evaluation with $C = 1,000$ and $K = 15$.

Fig. 10. Throughput with various schemes.

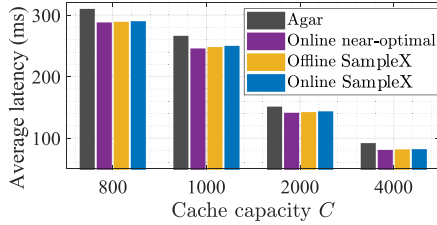


Fig. 11. Impact of cache capacity.

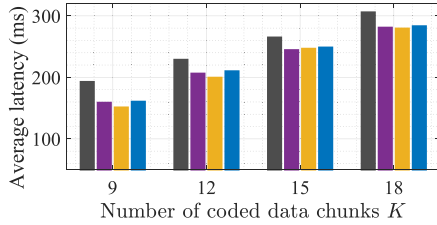


Fig. 12. Impact of coded data chunks.

decreases from 31.78 ms to 8.31 ms. The online SampleX is at least $23\times$ faster than the online near-optimal scheme.

Number of Coded Data Chunks K : The size of data items is increased from 9 MB to 18 MB. As the size of coded chunks remains unchanged (1 MB), the number of coded data chunks K changes from 9 to 18. The total number of data chunks increases from 9,000 to 18,000. Fixing the cache capacity $C = 1,000$ data chunks, the cache hit ratio of online SampleX decreases from 78.24% to 63.85%. As shown in Fig. 12, the average data access latency increases from 160.56 ms to 283.18 ms. Compared with the online near-optimal scheme, the incurred performance loss ranges from 0.81% to 1.84%. With the increase of K , the average number of required iterations of the online near-optimal scheme $|\hat{X}|$ increases from 15.58 to 125.17. The ART increases rapidly from 2.61 ms to 68.24 ms. In contrast, with the number of considered samples fixed at $S = 5$, the ART of online SampleX

TABLE VI
ART OF OFFLINE SCHEMES AND THE ART OF ONLINE SCHEMES UPON THE ARRIVAL OF EACH DATA REQUEST UNDER VARIOUS SETTINGS

Cache Capacity C	800	1,000	2,000	4,000
Agar (s)	7.06	10.19	19.16	33.25
Offline SampleX (s)	2.52	3.23	5.04	9.63
Online near-optimal (ms)	31.78	27.41	17.42	8.31
Online SampleX (ms)	0.75	0.68	0.45	0.36

Number of Coded Data Chunks K	9	12	15	18
Agar (s)	5.99	9.13	10.19	11.83
Offline SampleX (s)	2.87	3.01	3.23	4.13
Online near-optimal (ms)	2.61	8.38	27.41	68.24
Online SampleX (ms)	0.48	0.60	0.68	0.72

Number of Data Replicas	0	1	2	3
Agar (s)	10.19	10.08	10.26	10.21
Offline SampleX (s)	3.23	3.41	3.28	3.37
Online near-optimal (ms)	27.41	32.47	31.56	28.70
Online SampleX (ms)	0.68	0.72	0.67	0.65

Number of Data Items M	800	1,000	2,000	5,000
Agar (s)	8.08	10.19	21.42	45.52
Offline SampleX (s)	2.77	3.23	6.98	13.67
Online near-optimal (ms)	19.73	27.41	42.80	59.21
Online SampleX (ms)	0.46	0.68	1.05	1.87

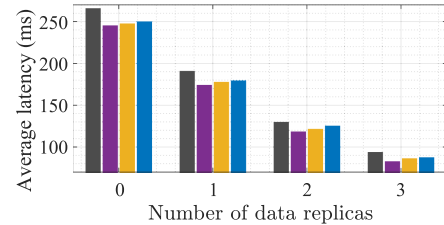


Fig. 13. Impact of data replicas.

increases slightly from 0.48 ms to 0.72 ms. The online SampleX is up to $95\times$ faster than the online near-optimal scheme.

Number of Data Replicas: With the consideration of replication, more copies of coded chunks will be placed at remote buckets. Since the coded chunks are uniformly distributed, more data chunks will be placed at the nodes close to end users as the number of data replicas gets larger. Then, the faraway storage nodes with high data access latency are no longer the bottleneck. Fig. 13 shows that the average data access latency of online SampleX is considerably reduced from 248.58 ms to 86.56 ms. Compared with the online near-optimal scheme, online SampleX is about $40\times$ faster with a performance loss ranging from 1.72% to 5.90%.

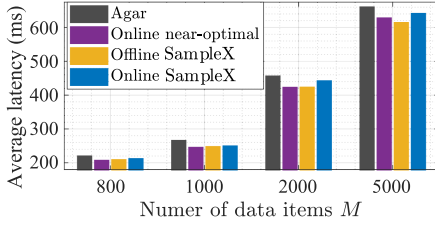


Fig. 14. Impact of data items.

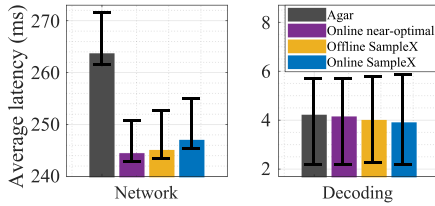


Fig. 15. Impact of server failure.

Number of Data Items M : As shown in Fig. 14, the number of deployed data items M is increased from 800 to 5,000. With the number of coded data chunks per data item $K = 15$, the total number of data chunks increases from 12,000 to 75,000. Due to the limited cache capacity $C = 1,000$, more and more data requests are served by fetching data chunks from remote buckets. The cache hit ratio of online SampleX decreases from 72.36% to 16.51%. Therefore, the average data access latency increases from 211.01 ms to 640.34 ms. With the increase of M , the caching decisions are more frequently updated upon the arrival of data requests. The ART of online SampleX increases from 0.46 ms to 1.87 ms. For the same reason, the ART of the online near-optimal scheme increases from 19.73 ms to 59.21 ms. Compared with the online near-optimal scheme, online SampleX is over $30\times$ faster with a performance loss ranging from 1.72% to 4.57%.

Server Failure: The performance of caching schemes is evaluated when server failure happens. If the requested data chunk m_k is unavailable, the parity chunk m_r with the lowest network latency will be fetched from the remote bucket for data reconstruction. Similar to [21], when the caching schemes suggest that m_r should be cached, the recovered data chunk m_k (instead of m_r) is added into the caching layer to avoid the decoding overheads of the subsequent data requests. Otherwise, if m_k is not cached, the degraded read is always triggered to serve the data requests. In this case, the data access latency contains two parts, i.e., the network latency and the decoding latency. Furthermore, previous research work [43] indicated that single server failure accounts for 99.75% of all kinds of server failures. Although erasure codes can tolerate up to R simultaneous server failures, this paper considers single server failure by terminating each bucket in turn. Fig. 15 illustrates the maximal, minimal, and average data access latencies with various caching schemes. By caching the recovered data chunks, the decoding latency is an order of magnitude lower than the network latency. Compared with the online near-optimal scheme (with the ART of 28.19 ms), online SampleX (with the ART of 0.73 ms) is about $39\times$ faster with a performance loss of 0.93% under server failure.

VI. CONCLUSION AND FUTURE WORK

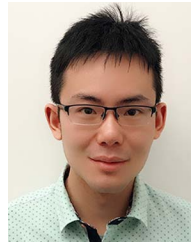
Caching in distributed coded storage systems has drawn increasing attention in recent years. The state-of-the-art caching schemes achieve the lowest data access latency, but suffer from high computation overheads when applied to a large-scale storage system. This paper presents SampleX, a novel extension to the optimal schemes based on sampling, reducing the computation overheads while maintaining the ultimate performance of caching on latency reduction. The streaming-based implementation of SampleX captures the characteristics of the recent traffic, updating the caching decisions in a scalable and online manner. Trace-driven experimental results demonstrate the superior performance of SampleX in reducing data access latencies and computation overheads.

In this paper, we consider the scenario where each edge server separately provides caching services to its affiliated end users. In future work, we plan to investigate the cooperative caching problem among edge servers to further reduce the data access latency.

REFERENCES

- [1] D. Reinsel, J. Gantz, and J. Rydning, "The digitization of the world from edge to core," *IDC White Paper*, vol. 16, pp. 1–28, Nov. 2018.
- [2] Amazon simple storage service, 2020. [Online]. Available: <https://aws.amazon.com/s3/>
- [3] Google Cloud Storage, 2020. [Online]. Available: <https://cloud.google.com/storage/>
- [4] Microsoft Azure, 2020. [Online]. Available: <https://azure.microsoft.com/en-us/>
- [5] Y. Hu, Y. Wang, B. Liu, D. Niu, and C. Huang, "Latency reduction and load balancing in coded storage systems," in *Proc. Symp. Cloud Comput.*, 2017, pp. 365–377.
- [6] M. Abebe, K. Daudjee, B. Glasbergen, and Y. Tian, "EC-Store: Bridging the gap between storage and latency in distributed erasure coded systems," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 255–266.
- [7] R. Halalai, P. Felber, A. Kermarrec, and F. Taïani, "Agar: A caching system for erasure-coded data," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 23–33.
- [8] M. Uluoyol, A. Huang, A. Goel, M. Chowdhury, and H. V. Madhyastha, "Near-optimal latency versus cost tradeoffs in geo-distributed storage," in *Proc. 17th Usenix Conf. Netw. Syst. Des. Implementation*, 2020, pp. 157–180.
- [9] Q. Ma, E. Yeh, and J. Huang, "How bad is selfish caching?," in *Proc. 20th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2019, pp. 11–20.
- [10] A. Singla, B. Chandrasekaran, P. B. Godfrey, and B. Maggs, "The Internet at the speed of light," in *Proc. 13th ACM Workshop Hot Top. Netw.*, 2014, pp. 1–7.
- [11] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, 1966.
- [12] N. Atre, J. Sherry, W. Wang, and D. S. Berger, "Caching with delayed hits," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl., Technol., Architectures, Protoc. Comput. Commun.*, 2020, pp. 495–513.
- [13] S. ElAzzouni, F. Wu, N. Shroff, and E. Ekici, "Predictive caching at the wireless edge using near-zero caches," in *Proc. ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2020, pp. 121–130.
- [14] D. S. Berger, R. Sitaraman, and M. Harchol-Balter, "AdaptSize: Orchestrating the hot object memory cache in a CDN," in *Proc. 14th USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 483–498.
- [15] N. Beckmann, H. Chen, and A. Cidon, "LHD: Improving cache hit rate by maximizing hit density," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2018, pp. 389–403.
- [16] Z. Song, D. S. Berger, K. Li, and W. Lloyd, "Learning relaxed belady for content distribution network caching," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2020, pp. 529–544.
- [17] X. Jin et al., "NetCache: Balancing key-value stores with fast in-network caching," in *Proc. ACM Symp. Operating Syst. Princ.*, 2017, pp. 121–136.

- [18] Z. Liu et al., "DistCache: Provable load balancing for large-scale storage systems with distributed caching," in *Proc. 17th USENIX Conf. File Storage Technol.*, 2019, pp. 143–157.
- [19] Y. Li, J. Zhou, W. Wang, and Y. Chen, "RE-store: Reliable and efficient KV-store with erasure coding and replication," *IEEE Int. Conf. Cluster Comput.*, 2019, pp. 1–12.
- [20] K. Liu, J. Peng, J. Wang, Z. Huang, and J. Pan, "Adaptive and scalable caching with erasure codes in distributed cloud-edge storage systems," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1840–1853, Apr./Jun. 2023.
- [21] K. Liu, J. Peng, J. Wang, and J. Pan, "Optimal caching for low latency in distributed coded storage systems," *IEEE/ACM Trans. Netw.*, vol. 30, no. 3, pp. 1132–1145, Jun. 2022.
- [22] D. Easley and J. Kleinberg, *Networks, Crowds, and Markets*. Cambridge, U.K.: Cambridge Univ. Press, 2010.
- [23] S. Balakrishnan et al., "Pelican: A building block for exascale cold data storage," in *Proc. 11th USENIX Conf. Operating Syst. Des. Implementation*, 2014, pp. 351–365.
- [24] A. Blankstein, S. Sen, and M. J. Freedman, "Hyperbolic caching: Flexible caching for web applications," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2017, pp. 499–511.
- [25] A. Jain and C. Lin, "Back to the future: Leveraging Belady's algorithm for improved cache replacement," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Architecture*, 2016, pp. 78–89.
- [26] Y. Ma, T. Nandagopal, K. P. Puttaswamy, and S. Banerjee, "An ensemble of replication and erasure codes for cloud file systems," in *Proc. IEEE INFOCOM*, 2013, pp. 1276–1284.
- [27] V. Aggarwal, Y. F. Chen, T. Lan, and Y. Xiang, "Sprout: A functional caching approach to minimize service latency in erasure-coded storage," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3683–3694, Dec. 2017.
- [28] A. O. Al-Abbasi and V. Aggarwal, "TTLCache: Taming latency in erasure-coded storage through TTL caching," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 3, pp. 1582–1596, Sep. 2020.
- [29] K. V. Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran, "EC-Cache: Load-balanced, low-latency cluster caching with online erasure coding," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 401–417.
- [30] A. Birlir, B. Radke, and T. Neumann, "Concurrent online sampling for all, for free," in *Proc. 16th Int. Workshop Data Manag. New Hardware*, 2020, pp. 1–8.
- [31] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi, "Sketch-based multiquery processing over data streams," in *Proc. Data Stream Manage.*, 2016, pp. 241–261.
- [32] B. Yu and J. Pan, "Sketch-based data placement among geo-distributed datacenters for cloud storages," *IEEE 35th Annu. Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
- [33] F. Yates, "Systematic sampling," *Philos. Trans. R. Soc. A*, vol. 241, no. 834, pp. 345–377, 1948.
- [34] HDFS Architecture Guide, 2019. [Online]. Available: <https://hadoop.apache.org/>
- [35] D. Lee et al., "LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Trans. Comput.*, vol. 50, no. 12, pp. 1352–1361, Dec. 2001.
- [36] L. Tang, Q. Huang, A. Puntambekar, Y. Vigfusson, W. Lloyd, and K. Li, "Popularity prediction of facebook videos for higher quality streaming," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2017, pp. 111–123.
- [37] K. Liu et al., "A learning-based data placement framework for low latency in data center networks," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 146–157, First Quarter 2022.
- [38] Memcached, 2019. [Online]. Available: <http://memcached.org/>
- [39] J. S. Hunter, "The exponentially weighted moving average," *J. Qual. Technol.*, vol. 18, no. 4, pp. 203–210, 1986.
- [40] S. Shukla, O. Bhardwaj, A. A. Abouzeid, T. Salonidis, and T. He, "Hold'em caching: Proactive retention-aware caching with multi-path routing for wireless edge networks," in *Proc. ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2017, pp. 24–33.
- [41] zfec, 2013. [Online]. Available: <https://github.com/richardkiss/py-zfec>
- [42] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Trans. Storage*, vol. 4, no. 3, pp. 1–23, 2008.
- [43] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing i/o for recovery and degraded reads," in *Proc. USENIX Conf. File Storage Technol.*, 2012, pp. 20–33.



Kaiyang Liu (Member, IEEE) received the PhD degree from the School of Information Science and Engineering, Central South University, Changsha, China, and did his postdoctoral research with the University of Victoria, Victoria, British Columbia, Canada. He is currently an assistant professor with the Department of Computer Science, Memorial University of Newfoundland, St John's, Newfoundland and Labrador, Canada. His current research areas include distributed cloud/edge computing and storage networks, data center networks, and distributed machine learning. In 2020, he was awarded the IEEE Technical Committee on Cloud Computing (TCCLD) Outstanding PhD Thesis Award.



Jingrong Wang (Graduate Student Member, IEEE) received the bachelor's degree from the School of Electronic and Information Engineering, Beijing Jiaotong University, in 2017, and the MSc degree in computer science from the University of Victoria, in 2019. She is currently working toward the PhD degree with the University of Toronto. Her research interests cover wireless communications, mobile edge computing, and distributed machine learning.

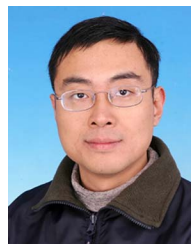


Heng Li (Member, IEEE) received the bachelor's and PhD degrees from Central South University, Changsha, China, in 2011 and 2017, respectively. He is currently an associate professor with the School of Computer Science and Engineering, Central South University. From 2015 to 2017, he was a research assistant with the Department of Computer Science, University of Victoria, Victoria, BC, Canada. His research interests include digital twin and cyber-physical systems.



Jun Peng (Senior Member, IEEE) received the BS degree from Xiangtan University, Xiangtan, China, in 1987, the MSc degree from the National University of Defense Technology, Changsha, China, in 1990, and the PhD degree from Central South University, Changsha, China, in 2005. In 1990, she joined Central South University. From 2006 to 2007, she was with the School of Electrical and Computer Science, University of Central Florida, USA, as a visiting scholar. She is a professor with the School of Computer Science and Engineering, Central South University,

China. Her research interests include cooperative control, cloud computing, and wireless communications.



Jianping Pan (Fellow, IEEE) received the bachelor's and PhD degrees in computer science from Southeast University, Nanjing, Jiangsu, China. He did his postdoctoral research with the University of Waterloo, Waterloo, Ontario, Canada. He is currently a professor of computer science with the University of Victoria, Victoria, British Columbia, Canada. He also worked with Fujitsu Labs and NTT Labs. His area of specialization is computer networks and distributed systems, and his current research interests include protocols for advanced networking, performance analysis of networked systems, and applied network security. He received the IEICE Best Paper Award in 2009, the Telecommunications Advancement Foundation's Telesys Award in 2010, the WCSP 2011 Best Paper Award, the IEEE Globecom 2011 Best Paper Award, the JSPS Invitation Fellowship in 2012, the IEEE ICC 2013 Best Paper Award, and the NSERC DAS Award in 2016, and has been serving on the technical program committees of major computer communications and networking conferences including IEEE INFOCOM, ICC, Globecom, WCNC and CCNC. He was the Ad Hoc and Sensor Networking Symposium Co-Chair of IEEE Globecom 2012 and an associate editor of *IEEE Transactions on Vehicular Technology*. He is a senior member of ACM.