Adaptive Data Placement in Multi-Cloud Storage: A Non-Stationary Combinatorial Bandit Approach

Li Li, Jiajie Shen, Bochun Wu, *Senior Member, IEEE*, Yangfan Zhou, *Member, IEEE*, Xin Wang, and Keqin Li, *Fellow, IEEE*

***Abstract*—Multi-cloud storage is recently a viable approach to solve the vendor lock-in, reliability, and security issues in cloud storage systems. As a key concern, data placement influences the cost and performance of storage services. Yet, in practice it remains challenging to address the huge solution space. Previous studies typically focus on constructing efficient data placement schemes based on the predicted pattern of workloads or assuming fully a-priori known network conditions. They cannot be easily applied in multi-cloud storage scenarios, which typically involve dynamic network conditions and time-varying workloads. To this end, we formulate the data placement optimization in a combinatorial multi-arm bandit (CMAB) perspective and solve it by learning placement strategy online. In contrast to a stationary setting where reward distributions are unknown but identical over time, we consider a realistic multi-cloud environment with non-stationary conditions, i.e., reward distributions change over time. To swiftly accommodate this, we propose an adaptive window combinatorial upper confidence bound based data placement (AW-CUCB-DP) scheme to reduce latency and cost. In AW-CUCB-DP, a simple and efficient change detector, i.e., *Page-Hinkley test* with forgetting mechanism (FM-PHT), is employed to enable variable-size sliding windows to handle both gradual and abrupt variations in network conditions or workloads. We establish that AW-CUCB-DP is asymptotically optimal in the non-stationary multi-cloud environment. Trace-driven experiments further verify that our scheme outperforms alternatives, especially in highly dynamic environments.**

***Index Terms*—Combinatorial multi-armed bandit, data placement, erasure codes, multi-cloud storage, non-stationary.**

Manuscript received 19 June 2022; revised 21 July 2023; accepted 13 August 2023. Date of publication 17 August 2023; date of current version 12 September 2023. This work was supported in part by Shanghai Municipal Science and Technology Commission under Grant 22dz1204900, in part by the National Natural Science Foundation of China under Grant 61971145, and in part by the Natural Science Foundation of Shanghai under Grant 22ZR1407900. Recommended for acceptance by R. Prodan. *(Corresponding authors: Xin Wang; Keqin Li.)*

Li Li is with the School of Computer Science, Fudan University, Shanghai 200433, China, and also with Informatization Office, Yunnan University, Kunming, Yunnan 650504, China (e-mail: lil18@fudan.edu.cn).

Jiajie Shen and Bochun Wu are with Informatization Office, Fudan University, Shanghai 200433, China (e-mail: jiajieshen@fudan.edu.cn; wubochun@fudan.edu.cn).

Yangfan Zhou and Xin Wang are with the School of Computer Science, Fudan University, Shanghai 200433, China, and also with the Shanghai Key Laboratory of Intelligent Information Processing, Shanghai 200433, China (e-mail: zyf@fudan.edu.cn; xinw@fudan.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TPDS.2023.3306150, provided by the authors.

Digital Object Identifier 10.1109/TPDS.2023.3306150

I. Introduction

A. Motivation and Challenges

CLOUD storage systems are widely adopted in current online applications to provide reliable, scalable, and low-cost services. However, storing data in a single cloud is susceptible to vendor lock-in, reliability, and security issues [1], [2]. In this regard, as a recent, promising paradigm to address these issues, multi-cloud storage [3] splits user data with erasure codes [4] and distributes data chunks to multiple clouds provided by different vendors. It can reduce user-perceived latency and storage cost. Moreover, it can provide higher data availability and security. In recent years, more application providers have tried to build their own multi-cloud storage by exploiting cloud resource options both within and across vendors, yielding a better cost-performance tradeoff [5].

The critical step in building efficient multi-cloud storage is to determine the strategy for placing data across multiple clouds [6], which can affect the system performance in both system efficiency and user-experienced latency. In practice, we should consider the variety of storage services among different cloud storage providers in terms of performance, data-center locations, and prices. Unfortunately, such variety brings huge complexity to determining the best data placement strategy.

Recent studies generally focus on different aspects of geo-distributed data placement. Some studies propose to predict future user access dynamics with historical workloads, and accordingly optimize the placement strategy [7], [8], [9]. However, such approaches rely heavily on the stability of workloads in general. They cannot work effectively for time-varying workloads due to impacts of peak demand or major events, which is common in practice [10].

Although, some efforts (e.g., [11], [12]) optimized data placement by considering the trade-off between access latency, storage cost, and fault tolerance. They either consider stable network conditions, or make a priori assumptions about network dynamics. For example, the latency between storage nodes is assumed to be deterministic or constant [8], [9], [11]; The bandwidth between nodes is assumed to be known and its real-time fluctuations can be ignored [13]. However, in practice, the network conditions fluctuate significantly due to various reasons (e.g., network congestion and load dynamics) [14].

Considering the above situation and the importance of latency variance for geographically distributed data placement, we conduct a measurement study on cloud storage services of

Amazon AWS and Microsoft Azure. The results show that the latency variance exists and is significantly-large over time. It is not merely a problem in specific cloud, but a general issue in accessing cloud storage providers. In addition, the issue is also noticed by extensive research work that focuses on optimizing request mechanism to reduce tail latency or latency variance of cloud services by issuing redundant requests [15], [16], [17] and requests splitting [18].

We argue that data placement in multi-cloud environments should consider not only dynamic network conditions and time-varying workloads, which are hard to attain a priori in practical scenarios, but consider the efficiency or convergence speed of its solution. This poses a major challenge to the solution of this problem.

B. Our Contributions

To overcome the above challenge, we propose to formulate the problem from a combinatorial multi-armed bandit (CMAB) [19] perspective. CMAB is a widely-adopted optimization approach to deal with uncertainty, which in nature is a lightweight online learning model with low complexity and fewer assumptions of the environment [20]. CMAB can provide fast convergences and online adaptation due to high sampling efficiency and has the advantages of simple implementation and theoretical guarantees [21], [22], [23]. We formulate our CMAB problem with non-stationary reward so that the unknown dynamics can be learned online while effectively incorporating such learned information into the placement decision process. The bandit learning method is utilized to solve our placement problem online to adapt to the dynamic network conditions and workloads, which ensures the solving efficiency while not introducing high overhead to I/O operations.

We further tailor an adaptive window combinatorial upper confidence bound based data placement algorithm (namely, AW-CUCB-DP) specifically to deal with variations in network conditions and workloads as well as non-stationarity, i.e., network latency distribution changes over time. AW-CUCB-DP adopts the passive sliding-window method to handle gradual changes in workloads and network conditions. It can also actively detect their abrupt changes and adjust the window size accordingly. Finally, to guarantee performance and low overhead, we customize our upper confidence bound-based (UCB) policy [24] to solve the CMAB problem with guaranteed performance and low complexity.

Our contributions are summarized as follows.

- We are the first to formulate the data placement optimization in multi-cloud storage as a CMAB problem. This learning-based framework can swiftly respond to dynamic network conditions or workloads typically a-priori unknown in practical scenarios.
- We propose a novel AW-CUCB-DP algorithm, which can enable adaptive data placement with low latency and cost under uncertain network and workload dynamics. An efficient change detector (FM-PHT) is adopted for variable-size sliding windows to handle both gradual and abrupt variations in network and workloads.
- We establish that AW-CUCB-DP can asymptotically approach the optimal performance by proving an upper-bounded number of sub-optimal decisions.
- We conduct extensive data trace-driven experiments on real-world clouds. The evaluation results indicate that AW-CUCB-DP outperforms the existing methods in terms of I/O performance and cost reduction, which can reduce access latency by 40.22–62.92% under different workloads compared to existing methods. We also evaluate the robustness and scalability of AW-CUCB-DP. The results show that AW-CUCB-DP can adapt to highly dynamic environments and data placements on a large storage scale.

The rest of this paper is organized as follows. In Section II, we survey related work, which contrasts with ours. We present a measurement study in Section III. We model the data placement problem in Section IV. In Section V, we solve this problem considering the realistic multi-cloud environment. We provide experimental results in Section VI. Finally, the paper is concluded in Section VII.

II. Related Work

A. Data Placement in Cloud Storage System

Data placement in cloud storage has been widely studied to improve the quality of services. Existing literature shows that diversity and geo-distributed features can be utilized to provide better performance and cost efficiency.

By analyzing the workload features, Agarwal et al. [9] solved the data placement problem over globally distributed data centers, where data objects are iteratively migrated to storage nodes close to user devices or storage locations of the associated objects. Jiao et al. [7] proposed an iterative graph-cut based multi-objective data placement policy to minimize inter-cloud traffic, access cost, and replacement cost. Assuming that data request traffic is stable over a period of time, Yu et al. [8] designed a hypergraph-based scheme to place associated data in geo-distributed clouds, which can reduce the routing latency and traffic, and storage cost. Atrey et al. [25] used a randomized solution for the low-rank approximation of the hypergraph matrix, which improves the efficiency of data placement. Cui et al. [13] formulated the data placement problem by constructing a tripartite graph and adopted a genetic algorithm (GA) based strategy to reduce cloud traffic and access latency. Based on analysis of historical workloads, Ren et al. [26] modeled the data purchasing and placement as an integer linear programming problem and proposed a near-optimal algorithm to jointly reduce cost and latency. These offline schemes can iteratively approach the optimal solutions, but they are rather computationally-complex and thereby time-consuming. Moreover, these solutions cannot respond swiftly to workload variations [27].

Some studies have optimized multi-objective data placement under complex requirements. Su et al. [11] presented a systematic data-placement model (namely, *Triones*) in multi-cloud storage, which was solved by calculating euclidean distance. However, the network dynamics are not considered and handled. Oh et al. [5] proposed an integrated solution, *Wiera*, to optimize

TABLE I
COMPARISON OF DATA PLACEMENT SCHEMES

Scheme	Performance Metrics	Solution	Efficiency	Adaptivity
Volley [9]	Read(write) traffic, latency, load balance	Iterative optimization	Low	Offline
MODP [7]	Read(write) traffic, operation cost, re-placement cost	Graph cuts	Low	Offline
ADP [8]	Read traffic, read latency, storage cost for associated data	Hypergraph partitioning	Low	Offline
SpeCH [25]	Inter-datacenter traffic, latency and storage cost	Spectral clustering	Medium	Offline
Genetic Strategy [13]	Data traffic, latency in clouds	Genetic algorithm	Low	Offline
Datum [26]	Service cost and latency	Integer linear programming	High	Offline
Our scheme	Access latency, storage cost, fault tolerance	Combinatorial MAB	High	Online

data placement across multiple storage tiers, data centers, and providers, which can effectively reduce placement cost. However, it does not fully consider migration cost while handling dynamics and managing data movement.

B. Combinatorial and Non-Stationary MABs

The multi-armed bandit (MAB) is a simple but powerful framework to make decisions over time under information uncertainty. Auer et al. [24] proposed the classical UCB1 algorithm, which has been widely adopted for its effective trade-off between exploration and exploitation and fast convergence. Combinatorial MAB (CMAB) is an extension of MAB that allows multiple arms to be selected simultaneously. Chen et al. [19] first presented the framework of CMAB with a more general class of reward functions, in which combinatorial upper confidence bound (CUCB) is proposed as a general algorithm for CMAB. Unlike most studies on CMAB frameworks focusing on linear reward functions [28], our joint reward is a non-linear function of individual rewards.

A class of CMAB problems with non-stationary settings has received wide attention [29], wherein the reward distributions change over time. This setting often occurs in practical scenarios. Efficient strategies to balance *remembering* and *forgetting*, i.e., using more but maybe obsolete historical data or less but fresh one, are more desirable to handle such changing environments. To this end, Garivier and Moulines [30] presented discounted UCB (D-UCB) and sliding-window UCB (SW-UCB), in which they compute the UCB bound using discounted sampling history and recent sampling history within a time window, respectively. Different from the above passive methods ignoring the change instants, active policies [31], [32], [33] reset the bandit algorithm when detecting changes. In contrast, we combine passive SW-CUCB algorithm and active Page-Hinkley test (PHT) [34] to approach the optimal placement policies, in which the window size can be adapted to changes in latency distributions.

In order to clearly position our investigation and highlight our unique features, we compare the proposed schemes with existing research work in Table I.

- The impacts of geo-distributed locations on access latency have gained widespread attention in wide-area data storage research [35], [36], [37], [38]. Previous studies focusing on geo-distributed data placement generally assume the network conditions stable or rely on a-priori known conditions. For example, they consider the latency between storage nodes deterministic or constant [8], [9], [11], or assume that the bandwidth between nodes known and ignore real-time fluctuations [13]. In [5], [25], real-time measurements on bandwidth are conducted by a separate component. However, such active measurements involve significant overhead and monetary cost on clouds, and the sampling rate affects the accuracy. Our study considers the *uncertainty* of network latency, which originates from the dynamically-evolving network conditions in the realistic cloud environment. We handle such uncertainties with CMAB and learn their dynamics in a fine-grained time scale, enabling adaptive data placement.
- Considering the cases with abrupt change instants in access latency distribution, we do not directly adopt the conventional SW-UCB algorithm with a fixed window size. The design intuition behind our proposed AW-CUCB-DP is that under the non-stationary multi-cloud environment, the obsolete latency observations are not as informative for estimation, which crucially hinges on the size of time window. Intuitively, the faster the access latency means vary, the smaller the window size should be as the past observations lapse faster. This motivates us to combine the active change detection mechanism and the sliding window-based bandit algorithm.

III. MEASUREMENT STUDY

To investigate the impact of network dynamics on storage performance, we conduct a field measurement study. Specifically, we aim to study i) the latency variances of data requests to the same data center over different time of day; and ii) the latency variances on different data centers for the same data object size.

We conduct our measurements on object storage (i.e., S3 and Azure Blob Storage) of two popular cloud providers Amazon AWs and Microsoft Azure. We deploy cloud storage services in the Asia-Pacific DCs (regions), including Amazon-Hongkong, Amazon-Singapore, Azure-East Asia, and Azure-Korea. Each region creates one bucket to store data. Quantitatively, we define the user-perceived access latency as the time for a client to completely upload (download) a certain data object to (from) the cloud. We measure the write (read) latency of data objects in the size of 1 K, 10 K, 1 M, and 10 M. Deployed on a VM instance in Google Cloud's Asia-East-2 Region (Hongkong), the client periodically sends requests (every 10 seconds) to write (read) random data to (from) the storage service (via SDK from S3 and Blob Storage), respectively, then measures the request latency for uploading (downloading) data to (from) the cloud. We conducted the measurements in Sept. 2022.

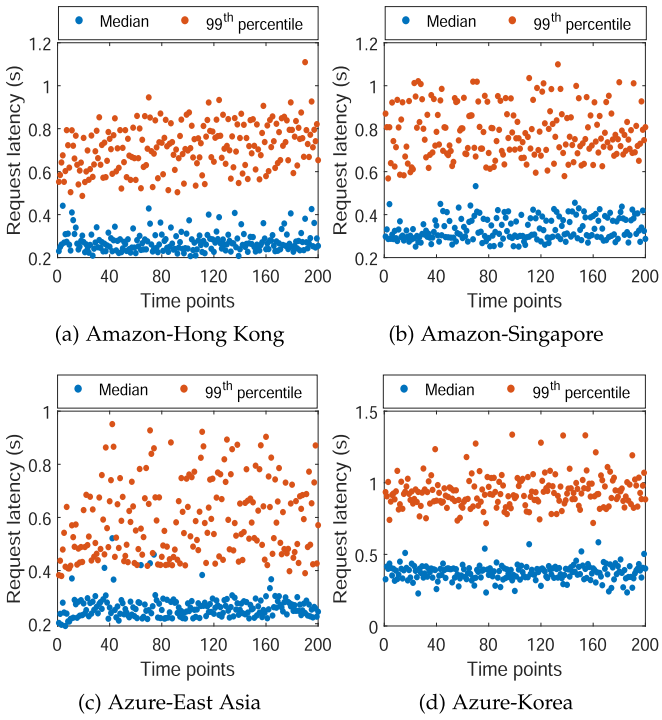


Fig. 1. Latency of requesting 10 KB data chunk.

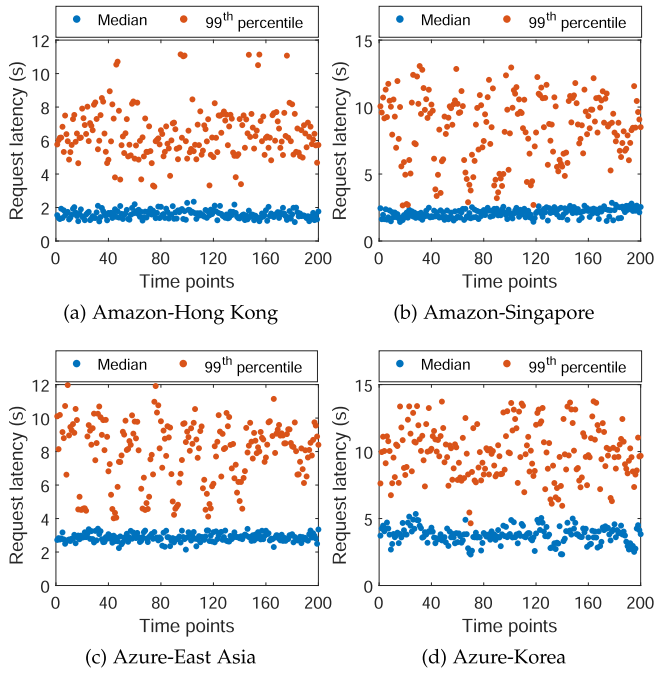


Fig. 2. Latency of requesting 10 MB data chunk.

Figs. 1 and 2 plot the measured request latency for 10 K and 10 M groups of data objects as requested from the client to the cloud data center, respectively. Each time point in the X-axis represents 1 h, and we plot the median (orange points) and the 99th percentile (blue points) of the request latency within each hour. It can be observed that the 99th percentile latency is 2–5 times the median latency. Such high variability and unpredictability network conditions have a great impact on optimizing

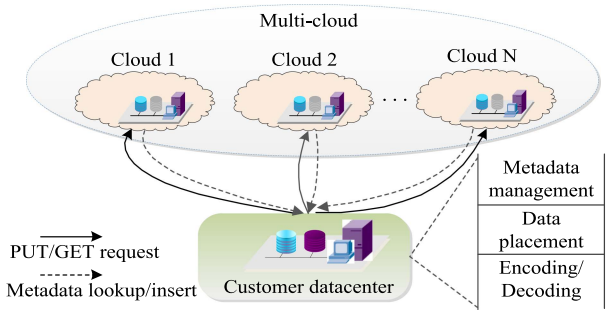


Fig. 3. Data placement in a multi-cloud environment.

the data placement strategy. The high latency variance is due to the network instability, as usually caused by network congestion, load dynamics, disk I/O interference, update/maintenance activities, and unpredictable failures. Therefore, cloud service providers and clients can only take the network as a black box when predicting latency. Then we have

Observation 1: The variance of the request latency for the data objects with the same size level from the same data center over time is very high.

Observation 2: The request latency variance exists in all data centers and is greater for requests for data objects with larger sizes. The request latency is not proportional to the client-data center distance.

Discussions: Multi-cloud storage service, e.g., Pure Cloud Block Store, utilizes the clouds located in different geographical locations as the underlying building blocks for storage. But such service typically provides only a virtual storage interface that does not allow us to designate which individual cloud when we want to save data. Hence, we can only measure such storage service as a black box. We cannot measure the performance differences of its underlying clouds. Therefore, to understand QoS characteristics of the building blocks of multi-cloud storage service, we measure the data I/O performance when accessing different clouds. Our aim is to better understand real-world multi-cloud storage services via measuring the performance of their underlying clouds.

IV. System Modelling and Problem Formulation

A. Multi-Cloud System Overview

We consider a multi-cloud scenario with N clouds, denoted by a set $[N] = \{1, 2, \ldots, N\}$, which are built by enterprises or application providers through purchasing services (e.g., VM instance or storage) on multiple clouds across vendors. Application providers build services and save metadata in the local data center (DC), and transfer erasure-coded data to cloud nodes of different vendors, as shown in Fig. 3.

Specifically, when writing data with a (n, k) Reed-Solomon (RS) code ($n > k$), the encoding/decoding module divides each object into k data chunks of equal sizes, which are then encoded into n chunks and uploaded to n different cloud nodes to guarantee reliability [39]. As all chunks are successfully written, metadata files are accordingly updated. While performing data retrieval, the data placement module first looks up its storage

TABLE II
PARAMETERS OF SYSTEM MODEL

Notation	Definition
$\mathcal{D}_t$	Data placement decision at time slot t
$x_{i,t}$	Binary variable to indicate whether chunks are uploaded (downloaded) to (from) cloud node i at time slot t
(n,k)	The erasure coding parameter ($1<k<n$)
$l^{\mathrm{W}}_{d,i}/l^{\mathrm{R}}_{d,i}$	The write/read latency from DC d to provider i
v_{obj}	The size of data object obj
$q_{\mathrm{w}}(t)/q_{\mathrm{r}}(t)$	The number of write/read requests at time slot t
$c_{\mathrm{s},i}/c_{\mathrm{o},i}$	The unit price of storage and outbound traffic for provider i
$c_{\mathrm{w},i}/c_{\mathrm{r},i}$	Price for a write/read request
a_i	The probability of availability of cloud i
$L_{\mathcal{D}_t}(t)/C_{\mathcal{D}_t}(t)$	Access latency/storage cost under placement policy $\mathcal{D}_t$ at time slot t
$A_{\mathcal{D}_t}$	Availability under placement policy $\mathcal{D}_t$ at time slot t

location and then selects k nodes to access and reconstruct the original object.

B. Data Placement Modelling

We consider a multi-cloud storage system designed and implemented based on object-based storage,[1] where data placement decisions are made in a time horizon with T slots. At each time slot, when a write or read request is issued, chunks (i.e., data chunks and parity chunks) are uploaded to different n cloud nodes or downloaded from k ones. We define a binary vector $\mathcal{D}_t=[x_{1,t},\ldots,x_{N,t}]$ to collect dynamic data placement decisions at time slot $t\in T$. Here, each $x_{i,t}$ ($i\in[N]$) equals 1 if chunks are uploaded to, or downloaded from, cloud node i at time slot t and 0 otherwise. To ensure the data reliability and access efficiency of multi-cloud storage, we have the following decision-making constraint

$$\sum_{i=1}^{N} x_{i,t}=\begin{cases} n, & \text{data write}\\ k, & \text{data read}\end{cases} \tag{1}$$

Based on the above definition, we model access latency, storage cost, fault-tolerance level, and data availability for storage service, which are associated with data placement decisions. The key notations are summarized in Table II.

Access Latency: The round-trip time to complete the write or read operation based on the current placement policy $\mathcal{D}_t$ is defined as the access latency,[2]

$$L_{\mathcal{D}_t}(t)=\begin{cases}\max\limits_{i\in[N]}\{x_{i,t}l^{\mathrm{W}}_{d,i}(t)\}, & \sum\limits_{i\in[N]}x_{i,t}=n\\ \max\limits_{i\in[N]}\{x_{i,t}l^{\mathrm{R}}_{d,i}(t)\}, & \sum\limits_{i\in[N]}x_{i,t}=k\end{cases} \tag{2}$$

where $l^{\mathrm{W}}_{d,i}(t)$ and $l^{\mathrm{R}}_{d,i}(t)$ denote the write and read latency between DC d that issues requests and cloud node i, respectively. Note that the definition of $L_{\mathcal{D}_t}$ is based on the assumption that accessing operations are executed in parallel [3], [11], [15].

Storage Cost: The monetary cost for storing (retrieving) data chunks of objects to (from) cloud node i under placement policy $\mathcal{D}_t$ is defined as storage cost,

$$C_{\mathcal{D}_t}(t)=\begin{cases}\sum\limits_{i\in[N]}x_{i,t}\left(\frac{v_{obj}}{k}c_{\mathrm{s},i}+c_{\mathrm{w},i}q_{\mathrm{w}}(t)\right), & \sum\limits_{i\in[N]}x_{i,t}=n\\ \sum\limits_{i\in[N]}x_{i,t}\left(\frac{v_{obj}}{k}c_{\mathrm{o},i}r(t)+c_{\mathrm{r},i}q_{\mathrm{r}}(t)\right), & \sum\limits_{i\in[N]}x_{i,t}=k\end{cases} \tag{3}$$

where $c_{\mathrm{s},i}$, $c_{\mathrm{o},i}$, and $c_{\mathrm{w},i}$ ($c_{\mathrm{r},i}$) denote the unit price of storage space, outbound traffic, and write/read operations for cloud i, respectively; $q_{\mathrm{w}}(t)$ ($q_{\mathrm{r}}(t)$) represents the number of times to write (read) objects at time slot t and v_{obj} denotes the size of objects.

Fault-Tolerance Level: The ability of a data placement policy to tolerate failures is defined as fault-tolerance level. We adopt Reed Solomon (RS) code to ensure data reliability. Each data object is divided into $k\geq 2$ chunks in the equal size, and a linear combination of these chunks is picked to generate $n-k$ parity chunks in the same size. Then these chunks are stored on different cloud nodes. The original data object can reconstruct by accessing any k cloud nodes. Therefore, in case of n storage nodes, at most $n-k$ cloud failures can be tolerated. That is, the data placement policy provides an $n-k$ fault-tolerance level.

Data Availability: We define $\mathcal{F}$ as all subsets of $m\in[k,n]$ unavailable clouds, and there are $|\mathcal{F}|=\binom{n}{m}$ cases of m unavailable clouds at the same time. Data availability under placement policy $\mathcal{D}_t$ can be defined as the probability that no more than m cloud nodes fail at the same time [11]

$$A_{\mathcal{D}_t}(t)=1-\left(\sum_{m=1}^{n-k}\sum_{j=1}^{|\mathcal{F}|}\prod_{i\in\mathcal{F}_{j,t}}(1-a_i)\prod_{i\in S_t\setminus\mathcal{F}_{j,t}}a_i\right), \tag{4}$$

where $\mathcal{F}_{j,t}$ is the jth element of the set $\mathcal{F}$ at time slot t; $S_t\setminus\mathcal{F}_{j,t}$ is a difference set that denotes $n-m$ nodes still working; $S_t=\arg\max\ \mathcal{D}_t$, a_i is the availability that cloud i guarantees through its service-level agreement (SLA).

C. Problem Formulation

Our objective is to determine the best data placement policy $\mathcal{D}_t$ at each time slot while achieving the optimal latency-cost

[1] We refer to each logical data as a data object, which can be a unit of data allocation and access in storage systems. It can refer to a traditional file fragment, a data block group, or an object in object-based storage.

[2] The access latency contains the network latency from remote storage nodes to the front-end server, the encoding/decoding latency, and the network latency from the front-end server to end users. Compared to high network latency over WAN (in hundreds of milliseconds), the encoding/decoding latency (in hundreds of nanoseconds) and the network latency from the front-end server to end users are negligible.

tradeoff. To this end, by allocating two weight parameters ψ_1 and ψ_2, we define a utility function under $\mathcal{D}_t$ as

$$u_{\mathcal{D}_t}(t) = \psi_1 L_{\mathcal{D}_t}(t) + \psi_2 C_{\mathcal{D}_t}(t). \tag{5}$$

Here, symbol (t) in $u_{\mathcal{D}_t}(t)$ identifies the different observation of the utility variable u. Since the observed utility values vary over time slots even under the same placement policy. Given a time horizon T, the data placement problem can be formulated as

$$\mathbf{P1}: \min_{\{\mathcal{D}_t\}_T} \frac{1}{T}\sum_{t=1}^{T} u_{\mathcal{D}_t}(t). \tag{6}$$

$$\text{s.t.} \quad (1)$$

$$n - k \geq F^{\text{req}} \tag{7}$$

$$A_{\mathcal{D}_t} \geq A^{\text{req}}. \tag{8}$$

Here, constraint (1) ensures the data reliability and access efficiency of erasure coding based multi-cloud storage, constraint (7) ensures that the fault-tolerance level of a placement policy has a lower bound F^{req}, and constraint (8) is to guarantee the promised availability. For different application scenarios, e.g., read-intensive workloads or write-intensive workloads, rich solutions of performance-cost tradeoffs can be achieved by adjusting the factor ψ_1/ψ_2.

There are two difficulties in solving problem $\mathbf{P1}$. First, it is a stochastic programming problem [29]. The exact information about the utility $u_{\mathcal{D}_t}(t)$ is not available before the tth data placement is completed. Additionally, even if $u_{\mathcal{D}_t}(t)$ is known a priori, this problem is still a combinatorial optimization problem. Then we consider converting this challenging problem into one low-complexity sequential decision problem in each time slot. To avoid a high computing burden, we select MAB rather than other reinforcement learning algorithms (e.g., Q-learning) to solve this sequential decision problem. The consideration is in that i) MAB is a lightweight online learning model with lower computation complexity and performance guarantee (as shown in Theorems 1 and 2); ii) MAB is efficient in sampling and converges quickly through strategic exploration; iii) MAB has more robust adaptive capabilities for faster online decision-making.

D. CMAB Reformulation

In problem $\mathbf{P1}$, it is difficult to obtain a-priori known information about dynamic network conditions and time-varying workloads. To deal with such uncertainty, we reformulate the intended data placement problem as a CMAB model with non-stationary rewards.

The combinatorial bandit setting involves repeatedly selecting multiple actions from N possible actions, aiming to find the optimal decision to achieve the maximum cumulative reward over time. At each time slot $t \in \mathcal{T}$, the CMAB i) selects a set of arms (referred to as super arm S), where $\mathcal{A} = \{S \in [N]^n \mid S(i) \neq S(j) \text{ for } \forall i \neq j\}$ is the set of all such possible super arms, and ii) observes the random reward of super arm played at time slot t (denoted by S_t).

To recast our problem, we regard each placement option (i.e., a combination of n out of N cloud nodes) as a super arm S, $|S| = n$, with $\binom{N}{n}$ super arms in total. We let S_t denote the super arm played at time slot t. Then the the utility (i.e., observed reward) under S_t can be expressed accordingly as

$$u_{S_t}(t) = \psi_1 L_{S_t}(t) + \psi_2 C_{S_t}(t). \tag{9}$$

Based on historical reward observations, S_t is selected to minimize utility $u_{S_t}(t)$ at each time slot.

We define the sequential data placement decisions as $\mathcal{D}$ (i.e., a series of played super arms over T time slots). Let $u_{S_t^*}(t)$ denote the minimal expected utility achieved by the optimal policy at time slot t. Then we can define the expected cumulative difference between $u_{S_t}(t)$ and $u_{S_t^*}(t)$ over time T as *regret*

$$R_{\mathcal{D}}(T) = \mathbb{E}\left[\sum_{t=1}^{T}(u_{S_t}(t) - u_{S_t^*}(t))\right]. \tag{10}$$

Here, the expectation $\mathbb{E}[\cdot]$ is taken with respect to the selection of S_t depending on obtained rewards through $\mathcal{D}$ and the randomness of reward given by the environment.

The objective of problem $\mathbf{P1}$ is to minimize the accumulative utility over time T by optimizing the data placement policy at each time slot t. By the definition of regret, minimizing the accumulated utility is equivalent to minimizing the regret in (10). Therefore, solving $\mathbf{P1}$ is equivalent to finding an optimal solution to the problem

$$\mathbf{P2}: \quad \arg\min_{\mathcal{D}} R_{\mathcal{D}}(T).$$

$$\text{s.t.} \quad (1), (7), (8). \tag{11}$$

The goal of $\mathbf{P2}$ is to minimize the accumulative regret over time based on the cumulative knowledge about unknown dynamics in network and workloads. There exists a trade-off between exploration (i.e., placing chunks to all nodes enough times to estimate rewards more accurately) and exploitation (i.e., placing chunks to empirically best nodes for instantaneous rewards) during data placement.

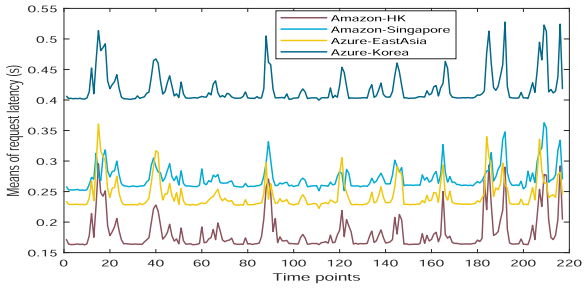
Therefore, we aim to find an asymptotically optimal strategy that achieves a sublinear regret, which implies that the per-round average cumulative regret approaches zero after sufficient time slots, i.e.,

$$\lim_{T\to\infty} \frac{R_{\mathcal{D}}(T)}{T} = 0. \tag{12}$$

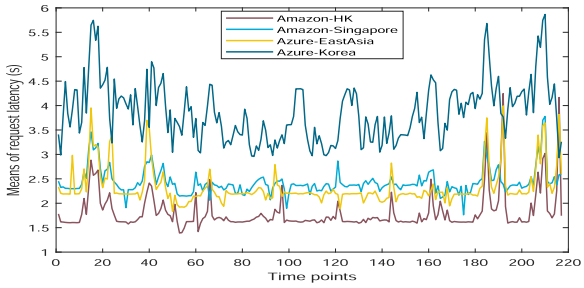
V. Placement Strategy Under Non-Stationary Multi-Cloud Environment

A. Non-Stationary Multi-Cloud Environment

In contrast to a stationary setting where reward distributions are unknown and stay the same through time, we consider a realistic multi-cloud environment with non-stationary reward [40], where the access latency is dynamically-evolving due to the arbitrarily time-varying network conditions, as shown in Fig. 4. That is, the mean of access latency ($\mathbb{E}[l_i^{\text{W/R}}(t)]$) will change at unknown time slots, referred as change-points (break-points).



(a) 10K objects



(b) 10M objects

Fig. 4. History of means of request latency.

We use this quantity to measure the speed of distribution changing, and assume the total number of change-points over T time slots is $\gamma_T = \mathcal{O}(T^\nu), \nu \in [0, 1)$, between consecutive breakpoints, network latency distributions remain constant. γ_T can be defined as

$$\gamma_T = \sum_{t=1}^{T-1} \mathbb{1}\{\exists i \in [N] : \mu_i(t) \neq \mu_i(t+1)\}, \tag{13}$$

where $\mu_i(t)$ denotes the expectation of access delay $l_i^{\mathrm{W/R}}(t)$ at time slot t. Note that such changes may occur asynchronously on cloud nodes. To limit the complexity of environment dynamics, we make the following mild assumptions for tractability.

Assumption 1 (Detectability [31]): Let $\mu_i(t)$ and $\mu_i(t+1)$ denote the pre- and post-change expectations of access latency of cloud node i at the change-point $t+1$. There exists a known parameter $\epsilon > 0$, such that for $\forall i \in [N]$ and $\forall t \leq T-1$, if $\mu_i(t) \neq \mu_i(t+1)$, then $|\mu_i(t) - \mu_i(t+1)| \geq 2\epsilon$.

We are only concerned with abrupt changes above some threshold. Assumption 1 excludes infinitesimal mean shift, which is reasonable in practice when detecting abrupt changes bounded by a certain threshold.

B. Change Detection

We adopt an active detection method to detect changes in access latency distribution and remove old observations to make more accurate estimates. Specially, we choose Page-Hinkley test (PHT) [34] to make detection because it is simple, computationally efficient, and requires the fewest distribution parameters compared with other detection methods.

In particular, we design a tailored PHT algorithm with forgetting mechanism (FM-PHT) to work in the bandit setting, which can detect both abrupt and gradual changes early, as described in Algorithm 1.

Algorithm 1: FM-PHT.

Input: Observations $y_1, \ldots, y_t$, threshold $b \geq 0$.
Output: Change-point t_{ch}.
Initialize: $U_0 = 0, L_0 = 0$;
$U_t = \frac{t-1}{t} U_{t-1} + (y_t - \overline{y}_t - \delta)$;
$U_t^{\min} = \min\{U_s | s \in \{1, \ldots, t\}\}$;
if $U_t - U_t^{\min} \geq b$ **then** ▷ Increase cases
  $t_{\mathrm{ch}} = \arg\min\{U_s | s \in \{1, \ldots, t\}\}$;
end if
$L_t = \frac{t-1}{t} L_{t-1} + (y_t - \overline{y}_t + \delta)$;
$L_t^{\max} = \max\{L_s | s \in \{1, \ldots, t\}\}$;
if $L_t^{\max} - L_t \geq b$ **then** ▷ Decrease cases
  $t_{\mathrm{ch}} = \arg\max\{L_s | s \in \{1, \ldots, t\}\}$;
end if

To be specific, we run two tests in parallel (i.e., two-sided) to monitor the possible positive/negative mean shift. Test variables U_t and L_t are defined as the cumulative difference between the observed access latency and the corresponding current average. That is

$$\begin{cases} U_t = \frac{t-1}{t} U_{t-1} + (y_t - \overline{y}_t - \delta) \\ L_t = \frac{t-1}{t} L_{t-1} + (y_t - \overline{y}_t + \delta) \end{cases}, \tag{14}$$

where $\overline{y}_t = 1/t \sum_{\tau=1}^{t} y_\tau$ and δ specifies the tolerable magnitude of changes. The ratio $\frac{t-1}{t}$ makes the most recent samples get more weight during the update process. The test also updates the minimum U_t (maximum L_t), denoted as $U_t^{\min}$ ($L_t^{\max}$), using

$$\begin{cases} U_t^{\min} = \min\{U_\tau | \tau \in 1, \ldots, t\} \\ L_t^{\max} = \max\{L_\tau | \tau \in 1, \ldots, t\} \end{cases}. \tag{15}$$

The two statistics ($U_t - U_t^{\min}$ and $L_t^{\max} - L_t$) are monitored and a change is reported when either of them is above the given threshold b. The parameter b can be tuned to balance false alarm and miss detection. When a change is detected, all variables are reset and the test is reinitialized.

C. Online Data Placement

Under a non-stationary environment, handling two main trade-offs, *exploration-exploitation* and *remembering-forgetting*, are key to decision-making. The former can be handled by utilizing the idea of UCB. For the latter, SW-CUCB [30] policy uses recent observations during a fixed size time window to compute the upper confidence bound. However, SW-CUCB may ignore sudden changes within the window, while observations after changes are more effective for accurate estimation. This motivates us to leverage FM-PHT to enhance the robustness of sliding-window policy against both gradually and abruptly changing environments.

1) AW-CUCB-DP Learning Algorithm: We first outline the main ideas of proposed AW-CUCB-DP.

Algorithm 2: AW-CUCB Based Data Placement.

Input: w_0, δ, b, λ.
Output: Placement policy S_t.
1: Initialize $\tau_i = 1$ and $w_i = w_0$ for each $i \in [N]$;
2: **for** $t = 1 : T$ **do**
3: **if** $t \leq N$ **then**
4: $I_t = t$;
5: Randomly choose a super arm S_t with $I_t \in S_t$;
6: **else**
7: Estimate the utility bound for each $i \in [N]$ $\hat{u}_i(t) = \psi_1 \bar{l}_i^{\mathrm{W/R}}(t) + \psi_2 C_i(t) - e_i(t)$;
8: Rank $\hat{u}_i(t)$ in ascending order;
9: Select the top n arms to added into S_t;
10: **end if**
11: Play super arm S_t and observe $\{l_i^{\mathrm{W/R}}(t)\}_{i \in S_t}$;
12: Update statistics in time-window $W_i(t)$ according to (18);
13: **if** FM-PHT$(i, l_i^{\mathrm{W/R}}(t)) == 1$ **then**
14: $\tau_i = t_{\mathrm{ch}}$;
15: Reset FM-PHT $(i, \cdot)$;
16: **if** $\lambda == 1$ **then**
17: LDM (S_t', S_t);
18: **end if**
19: **end if**
20: $w_i = \min(w_0, (t - \tau_i + 1))$;
21: **end for**

1) Unlike the general CUCB algorithm, AW-CUCB-DP probes possible base arms instead of super arms, i.e., each cloud node is learned and stored separately. Additionally, each node's access latency and storage cost are learned separately, which significantly reduces uncertainty and speeds up learning. Therefore, we decomposed the total reward function in (9), into a non-linear combination of individual reward functions

$$u_{S_t}(t) = \psi_1 \max_{i \in S_t} l_i^{\mathrm{W/R}}(t) + \psi_2 \sum_{i \in S_t} C_i(t), \tag{16}$$

where the joint access latency is the maximum of individual ones, i is a base arm (i.e., a cloud node).

2) Adopting a simple and efficient change detector FM-PHT to monitor changes in access latency distribution and trigger the adjustment of window size.

We present our AW-CUCB-DP algorithm in Algorithm 2. The inputs of AW-CUCB-DP include the initial window size $w_0 = L^B \sqrt{T \ln(T)/\gamma_T}$ and two tuning factors, δ, b for FM-PHT detector (line 13), and λ. Here, L^B is the utmost amplitudes of access latency, λ indicates whether this is a read request. Let τ_i denote the last detection time of node i, initialized to 1. w_i is the adaptive window size on node i.

We note that AW-CUCB-DP selects the best data placement options in five steps as follows.

i) Make Placement Decisions (Lines 3-9): In the exploration phase (Lines 3-5), AW-CUCB-DP makes decisions according to the exploration policy,[3] i.e., selecting a placement option containing at least one newly explored node. In the exploitation phase (Lines 6-9), based on the returned latency observations, AW-CUCB-DP first estimates the optimistic utility bound $\hat{u}_i(t)$ for each cloud node $i \in [N]$ by

$$\hat{u}_i(t) = \psi_1 \bar{l}_i^{\mathrm{W/R}}(t) + \psi_2 C_i(t) - e_i(t). \tag{17}$$

Here, $e_i(t)$ is a bonus, also called confidence radius, to further encourage exploration, and the corresponding exploration is controlled by T_{i,w_i}. ξ is a tuning parameter to adjust the preference towards exploration, i.e., a larger ξ inclines more exploration. Then AW-CUCB-DP sorts $\hat{u}_i(t)$ for $i \in [N]$ in ascending order, selects the top n/k nodes to make data placement.

ii) Perform Data Write (Read) According to Placement Decision S_t and Observe Access Latency for Node $i \in S_t$: Specially, the data placement module uploads (downloads) data chunks on (from) the selected cloud nodes and collects access latency $l_i^{\mathrm{W/R}}(t)$ for each node $i \in S_t$. Note that this step belongs to storage system operations, as performed by a separate thread in implementation.

iii) Update Statistics Based on the Current Window Size w_i for Each Cloud Node $i \in [N]$: To estimate the utility bound for each cloud node by (17), AW-CUCB-DP calculates associated statistics within time-window $W_i(t) = \{\max(1, t - w_i), \ldots, t - 1\}$ based on the returned latency observations. That is

$$\begin{cases} \bar{l}_{i,w_i}^{\mathrm{W/R}} = \frac{1}{T_{i,w_i}} \sum_{t \in W_i(t)} l_i^{\mathrm{W/R}}(t) \mathbb{1}\{i \in S_t\} \\ T_{i,w_i} = \sum_{t \in W_i(t)} \mathbb{1}\{i \in S_t\} \\ e_i(t) = L^B \sqrt{\xi \ln(w_i)/T_{i,w_i}} \end{cases}, \tag{18}$$

where $\bar{l}_{i,w_i}^{\mathrm{W/R}}$ and T_{i,w_i} are the empirical average of access latency within window w_i and the number of times cloud node i has been selected, respectively. L^B is the utmost amplitude of access latency $l_i^{\mathrm{W/R}}(t)$, i.e., for $\forall i \in S_t$ [41],

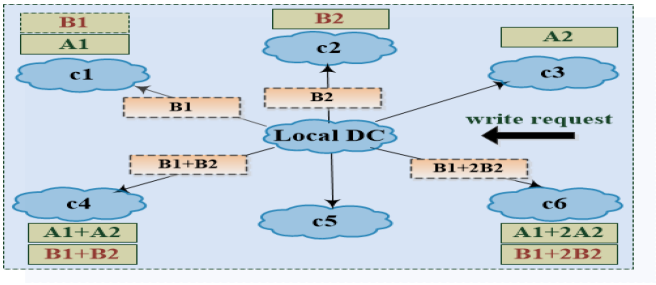
$$L^B = \sup_{\forall t,i} l_i^{\mathrm{W/R}}(t) - \inf_{\forall t,i} l_i^{\mathrm{W/R}}(t). \tag{19}$$

iv) Detect Changes by FM-PHT: FM-PHT (Algorithm 1) runs on each node with the updated statistics. When changes in distribution are detected for any node (Line 13), AW-CUCB-DP sets τ_i to be the change time slot t_{ch} (Line 14) and reset the detector (Line 15). For a read request (i.e., $\lambda = 1$), re-evaluations of storage locations for requested objects are triggered at this moment (Line 17), i.e., calling Algorithm 3.

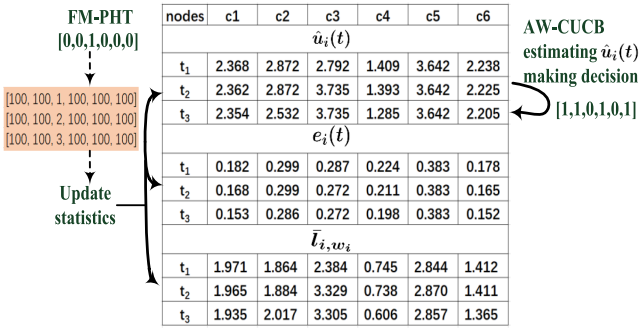
v) Update the Window Size (Line 20): The window size $w_i = \min(w_0, (t - \tau_i + 1))$ is adjusted accordingly to adapt to changes on node i, enabling AW-CUCB-DP to estimate the utility bound $\hat{u}_i(t)$ based on valid observations (i.e., those obtained since the last detection time τ_i by the current time).

Remark 1: The combinatorial bandit model requires exploring the optimal set of base arms (i.e., super arm) among $\binom{N}{n}$ options at each time step, making the action space large. Due to the independence among the base arms, we can judiciously learn

[3]The initial exploration policy is not unique and can be any method that selects each cloud node at least once.



(a)



nodes	c1	c2	c3	c4	c5	c6
$\hat{u}_i(t)$						
t_1	2.368	2.872	2.792	1.409	3.642	2.238
t_2	2.362	2.872	3.735	1.393	3.642	2.225
t_3	2.354	2.532	3.735	1.285	3.642	2.205
$e_i(t)$						
t_1	0.182	0.299	0.287	0.224	0.383	0.178
t_2	0.168	0.299	0.272	0.211	0.383	0.165
t_3	0.153	0.286	0.272	0.198	0.383	0.152
$\bar{l}_{i,w_i}$						
t_1	1.971	1.864	2.384	0.745	2.844	1.412
t_2	1.965	1.884	3.329	0.738	2.870	1.411
t_3	1.935	2.017	3.305	0.606	2.857	1.365

(b)

Fig. 5. An example of placement decision by AW-CUCB-DP.

and store the individual reward of each cloud node separately to avoid combinatorial explosion. This allows AW-CUCB-DP to select the top n out of N nodes greedily according to the current estimate $\{\hat{u}_i(t)\}_{i=1}^{N}$ to form the super arm. Finally, the action space of AW-CUCB-DP is reduced from $\binom{N}{n}$ to N, i.e., only $O(N)$ storage and $O(N)$ computation are required for the sampling per time step, reducing the storage and computational complexity.

Remark 2: By actively detecting changes in the access latency distribution to adjust window sizes, AW-CUCB-DP can handle both abrupt and gradual changes. Specifically, after completing the write (read) operation, the observed access delay is used to learn the policy by AW-CUCB-DP and fed to the change detector (FM-PHT) to make detection simultaneously. Each time a change in distribution is detected on a cloud node i, its window size w_i will shrink to $w_i = \min(w_0, (t - \tau_i + 1))$, thereby disregarding stale observations. Here, τ_i is the latest time slot of change-point detection. After that, window size w_i gradually increases to w_0 to obtain higher estimating accuracy of access latency. w_i is upper-bounded by the initial window size w_0, a best-tuned parameter, ensuring the performance of SW-UCB policies in the absence of abrupt changes.

Notably, a separate adaptive window (in size w_i) is maintained for each node to achieve better empirical performance. This is reasonable since τ_i are usually not equal for all nodes in practice.

2) An Illustrative Example: Considering a multi-cloud storage scenario with six nodes, adopting RS (4, 2) codes to provide fault-tolerance, the initial window size $w_0 = 100$, as shown in Fig. 5(a).

We start with the example in a middle decision time slot t_2. For a write request for object A at t_2, the chunks of object A, i.e., $A1$, $A2$, $A1 + A2$, and $A1 + 2A2$, are uploaded to cloud node $c1$, $c3$, $c4$, and $c6$ based on the placement decision [1, 0, 1, 1, 0, 1] made at t_1. The returned latency observations, i.e., $l_1(t_2)$, $l_3(t_2)$, $l_4(t_2)$, and $l_6(t_2)$, are used to update node associate statistics according to (18) and are also sent to FM-PHT for change detection. The updated results for $e_i(t_2)$, $\bar{l}_{i,w_i}$ are stored in the t_2 row of matrices $e_i(t)$ and $\bar{l}_{i,w_i}$, respectively, in Fig. 5(b). Then AW-CUCB-DP updates estimations for the utility bound $\hat{u}_i(t_2)$ of node $c1$, $c3$, $c4$, and $c6$ based on (17), the results are stored in the t_2 row of matrix $\hat{u}_i(t)$, in Fig. 5(b). AW-CUCB-DP sorts $\hat{u}_i(t_2)$ of among 6 nodes in ascending order, selecting the top 4 nodes to obtain the placement decision ([1, 1, 0, 1, 0, 1]) for time slot t_3. At this moment, FM-PHT detects a distribution change in node $c3$ at time slot t_2. Then AW-CUCB-DP adjusts the window size of node $c3$ (w_3) to 1, which means only one validated historical observation is used to update statistics of node c3 at time slot t_3. Note that the resized window size at t_2 will be used when updating statistics at t_3. It will gradually increase until $w_0 = 100$ in the following decision time slots. Next, for another write request arrived at t_3, chunks of object B, $B1$, $B2$, $B1 + B2$, and $B1 + 2B2$, are uploaded to the cloud node $c1$, $c2$, $c4$, and $c6$ according to the placement decision ([1, 1, 0, 1, 0, 1]), as shown in Fig. 5(a).

Algorithm 3: Proactive Local Data Migration.

Input:: S'_t, S_t
Output:: Migrated placement of requested objects
1: Compare the current placement with the previous one: $M_s = S'_t \setminus S_t$, $M_d = S_t \setminus S'_t$;
2: **while** $|M_s| > 0$ **do**
3: Calculate migration gains according (20);
4: **end while**
5: **while** $G_M(t) \geq 0$ **do**
6: Move chunks from $M_s(j)$ to $M_d(j)$ sequentially for each $j \in |M_s|$;
7: **end while**

D. Data Migration

Applications may re-evaluate data placement when sustained changes in workloads or network conditions occur and compromise application goals. However, it is impractical to globally shuffle all the previous placement (i.e., migrating all chunks stored based on the previous placement decision) for real systems, which typically incurs huge migration overhead [12]. To avoid huge move for infrequently accessed data, we consider local chunk migrations to reduce migration cost, i.e., the re-evaluation of placement is taken only for the frequently accessed data. Specifically, whether to migrate is evaluated when the access latency distribution changes and the previous placement is not the best for the requested object. Migrations occur only when the migration condition is met, i.e., migration gains are non-negative.

Algorithm 3 describes our proactive local data migration (PLDM) algorithm, which improves the existing placement for requested objects. At each time slot t for data read, PLDM first

compares the current placement (S_t) with the placement of the requested object (S'_t) to obtain $M_s = S'_t \setminus S_t$ ($M_d = S_t \setminus S'_t$), which is a difference set containing cloud nodes that need to move out (move into) chunks of the requested object (Line 1). If S_t and $S_{t'}$ are different (i.e., $|M_s| > 0$), PLDM calculates migration gains $G_M(t)$ for chunks stored in node $i' \in M_s$ by

$$G_M(t) = \sum_{i' \in M_s} \left(\frac{v_{obj}}{k} (c_{\mathrm{s},i'} - c_{\mathrm{o},i'}) - c_{\mathrm{r},i'} \right) - \sum_{i \in M_d} \left(\frac{v_{obj}}{k} c_{\mathrm{s},i} + c_{\mathrm{w},i} \right), \quad (20)$$

where $S_t = \arg\max\ \mathcal{D}_t$, $S'_t = \arg\max\ \mathcal{D}'_t$ can be obtained by looking up metadata files. When the migration gains $G_M(t) \geq 0$, PLDM sequentially moves chunks of requested object to target nodes due to the same chunk size, i.e., moving chunks stored in node $M_s(j)$ to $M_d(j)$ for each $j \in |M_s|$, and then updates metadata files accordingly.

Remark 3: PLDM can locally improve the existing placement of requested objects, thereby avoiding huge migration overhead for infrequently accessed data. Moreover, migrations are performed in the background after read operations are completed, not affecting data availability.

E. Regret Analysis

We adopt the learning regret of utility as the performance criteria widely used in the MAB work. We analyze the regret bound of AW-CUCB-DP and thereby guarantee that AW-CUCB-DP achieves an asymptotically optimal regret bound.

We define $\tilde{N}_i(t)$ as a counter of sub-optimal decisions for each base arm $i \in [N]$ at time slot t and update the counters as follows: i) After the initial exploration (i.e., $t \leq N$), we set $\tilde{N}_i(t) = 1, i \in [N]$. ii) For a time slot $t > N$, when super arm S_t is chosen and S_t is a sub-optimal decision (i.e., $S_t \neq S_t^*$), we update the counter $\tilde{N}_i(t)$ in each time slot by

$$i = \arg\min_{j \in S_t} \tilde{N}_j(t-1),\ \tilde{N}_i(t) = \tilde{N}_i(t) + 1. \quad (21)$$

That is, we find the base arm i with the smallest counter in S_t and increase its counter $\tilde{N}_i(t)$ by one. If multiple base arms in S_t meet the condition, we arbitrarily pick one. On the other hand, when the played super arm S_t is the optimal one, $\tilde{N}_i(t)$ will not be updated.

By definition, the total number of sub-optimal decisions at time slot T is no more than $\sum_{i \in [N]} \tilde{N}_i(T)$. Note that when a sub-optimal super arm is played, it incurs loss at most $\Delta_{S_t}^{\max}(t)$, Thus,

$$R_{\mathcal{D}}(T) \leq \Delta_{S_t}^{\max}(T) \mathbb{E}\left[\sum\nolimits_{i \in [N]} \tilde{N}_i(T) \right], \quad (22)$$

where

$$\Delta_{S_t}^{\max}(T) = \max_{t \in \{1,\ldots,T\}, S_t \neq S_t^*} (u_{S_t} - u_{S_t^*}). \quad (23)$$

Therefore, it suffices to establish the regret bound by accumulating the upper bound of $\mathbb{E}[\sum_{i \in [N]} \tilde{N}_i(T)]$.

Theorem 1: Assuming $\xi > \frac{1}{2}$ for each arm $i \in [N]$, we have the following upper bound on sub-optimal decisions counter $\tilde{N}_i(t)$.

$$\mathbb{E}\left[\sum_{i \in [N]} \tilde{N}_i(T) \right] \leq \varphi_1 n T \frac{\ln(w_0)}{w_0} + \varphi_2 \gamma_T \ln(w_0), \quad (24)$$

$$\text{where} \begin{cases} \varphi_1 = \varphi_2 \frac{\lceil T/w_0 \rceil}{T/w_0} + \frac{2}{\ln(w_0)} \left\lceil \frac{\ln(w_0)}{\ln(1+4\sqrt{1-(2\xi)^{-1}})} \right\rceil \\ \varphi_2 = 4(L^B)^2 \xi / (\Delta_{S_t}^{\min}(T))^2 \\ \Delta_{S_t}^{\min}(T) = \min_{t \in \{1,\ldots,T\}, S_t \neq S_t^*} (u_{S_t} - u_{S_t^*}). \end{cases}$$

Clearly, the upper bound depends on the total number of placement tasks, change-points number γ_T, and the initial window size w_0. From (24), we see that the first term decreases as the w_0 increases, while the last term is the opposite. However, continuously larger windows may result in slow reaction to abrupt changes, while smaller windows may miss changes. Therefore, it is beneficial to adjust the window size according to detected changes dynamically.

Theorem 2: Under Assumption 1, if the number of change-points γ_T is known in advance, then we can choose

$$w_0 = L^B \sqrt{T \ln(T)/\gamma_T}, \quad (25)$$

so that

$$\mathbb{E}[R_{\mathcal{D}}(T)] \leq \mathcal{O}\left(n \sqrt{T \gamma_T \ln T} \right). \quad (26)$$

Furthermore, when $\gamma_T = \mathcal{O}(T^\nu), \nu \in [0, 1)$, and $T \to \infty$, we achieve a vanishing average regret

$$\lim_{T \to \infty} \frac{1}{T} \mathbb{E}[R_{\mathcal{D}}(T)] \leq \lim_{T \to \infty} \mathcal{O}\left(n \sqrt{T^{\nu-1} \ln T} \right) = 0. \quad (27)$$

Remark 4: This regret bound quantifies the impacts of change extent in latency distribution on the best achievable performance of data placement. The sublinear dependence on γ_T indicates AW-CUCB-DP is suitable for non-stationary environments with $\gamma_T \in (0, T)$. However, learning effects cannot be guaranteed in adversarial environments ($\gamma_T = T$) where latency distributions change all the time. The regret-bound polynomial depends on the number of cloud nodes n (upper-bounded by order $\mathcal{O}(n)$), indicating that AW-CUCB-DP can adapt to large-scale placement scenarios. The vanishing average regret implies AW-CUCB-DP is asymptotically optimal as T increases. Moreover, if γ_T is independent of T (i.e., $\nu = 0$), the regret bound in (26) is sublinear with T, which theoretically guarantees a significantly improved performance over any learning-free policy.

VI. Performance Evaluation

We evaluate the performance of our proposed algorithms on commercial clouds and conduct trace-driven experiments to evaluate their effectiveness under time-varying workloads. In addition, we also evaluate their robustness and performance at scale.

TABLE III
PRICING POLICIES OF CLOUD PROVIDERS

Cloud	Google	Rackspace	Amazon	Azure	Aliyun	Tencent
	Asia	China Hong Kong	Asia	Asia	China	China
Storage (per GB)	0.02	0.105	0.023	0.15	0.023	0.018
Network (per GB)	0.21	0.199	0.07	0.02	0.077	0.077
Put (per 1,000)	0.05	0	0.05	0.0005	0.0017	0.0015
Get (per 1,000)	0.004	0	0.004	0.0005	0.0017	0.0015
Availability	99.9%	99.9%	99.99%	99.9%	99.9%	99.99%

* The above prices are measured in dollars.

TABLE IV
AVERAGE UPLOAD (DOWNLOAD) TIME OF DIFFERENT CLOUD PROVIDERS

Cloud	Google	Rackspace	Amazon	Azure	Aliyun	Tencent
10 K	4.6	0.52	0.23	0.29	0.052	0.04
	3.11	3.97	0.23	0.29	0.056	0.04
100 K	4.8	0.79	0.86	0.295	0.12	0.063
	3.22	3.68	0.78	0.295	0.227	0.722
1 M	5.30	1.38	1.21	0.36	0.65	0.288
	3.35	4.31	2.77	0.42	2.72	8.079
10 M	7.53	6.08	2.65	1.28	0.779	2.41
	4.30	6.52	4.42	1.87	5.51	0.122
100 M	40.29	60.16	20.53	10.5	8.7	17.66
	16.69	39.68	27.55	13.7	0.236	2.21

* As to each object size (*e.g.*, 10K), the data in the first (second) row is uploading (downloading) latency (in the unit of s).

A. Experimental Settings

1) Real-World Clouds: We conducted trace-driven experiments on real-world cloud providers including Amazon S3, Windows Azure Blob Storage, Google Cloud Storage, Alibaba Cloud OSS, and Tencent Cloud COS. We assumed customer's local datacenter is located in Google Cloud's Asia-east2 (Hongkong) Region. For each provider, storage services are deployed over two regions, each region creates one bucket to store data.

2) Simulated Clouds: Cloud Parameters

i) The storage cost depends on the pricing policies of cloud providers (i.e., the unit prices of Storage, Put, Get, and outbound traffic), as shown in Table III.

ii) The dynamic link state in our simulations is set according to actual measurements.

We develop a measurement tool that contains a client and a server counterpart running on the cloud data center, to measure read/write latency of data objects in the size of 10 K, 100 K, 1 M, 10 M, and 100 M, which should cover the most common data sizes [4]. The client periodically sends requests to the server, every two minutes, to write random data to the storage service, and then reads the data. We measure the write (read) latency as the time of completely uploading (downloading) the data from the storage service. Particularly, we deploy the client on a node of Plantlab (Asia-East), and run the server on VM instances in 6 commercial cloud data centers, i.e., Google (Asia: East), Rackspace (HongKong, China), Amazon (Asia Pacific: East), Azure (Asia: East), Aliyun (East China), and Tencent (East China). This procedure is conducted in different weekdays in July 2021, and averaged to achieve the results in Table IV.

3) Traces: We employ different I/O traces from the MSRC benchmark suite [42] collected from 13 real enterprise server workloads to better simulate the real storage environment. We select different traces of eight storage servers to ensure that workloads with different access characteristics are covered. In addition, these traces are also representative and widely used in previous studies for experiments [43], [44]. Table V summarizes the characteristics of selected traces, including the types of traces and the statistics of write/read operations. There are three types of workloads, write-intensive, read-intensive, and balanced. A trace is considered write-intensive (read-intensive) if its write (read) ratio exceeds 0.75. Otherwise, a trace is considered balanced. Each trace consists of records of I/O requests, including timestamp, file name, file size, request type, etc. We replay the traces to schedule write/read requests according to each record's timestamp, request type and block sizes. The arrival rates of write/read requests for data objects are shown in Fig. 6.

TABLE V
DESCRIPTION OF TRACES [42]

Trace name	Description	Write ratio	Read ratio	Classify
rsrch0	Research projects	0.91	0.09	write-intensive
ts0	Terminal server	0.82	0.18	write-intensive
proj0	Project directories	0.88	0.12	write-intensive
usr1	User home directories	0.09	0.91	read-intensive
mds1	Media server	0.25	0.75	read-intensive
hm0	Hardware monitoring	0.65	0.36	balanced
prxy1	Project directories	0.35	0.65	balanced
web0	Web/SQL server	0.70	0.30	balanced

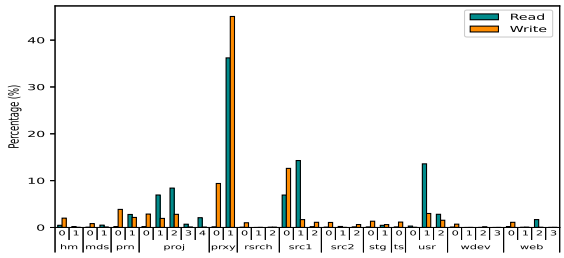


Fig. 6. Request arrival rates of traces.

To balance the storage overhead and access cost, we choose the (4, 2)-RS codes (i.e., $n = 4, k = 2$) encoding scheme for data dispersal. Here $k = 2$ is a popular option for applications to reduce access costs [45]. In general, larger values of k reduce the storage overhead while potentially incurring higher access costs due to accessing more nodes in parallel. This is often caused by a slow chunk access delaying the completion of an entire write or read request. To simplify the utility function to a dimensionless quantity while better weighing the impact of access latency and storage cost on the optimization objective, the latency- and cost-weights are set as $\psi_1 = 1\ \mathrm{s}^{-1}$ and $\psi_2 = 200\ \mathrm{dollar}^{-1}$. The required availability of applications $A^{\mathrm{req}} = 99.9\%$.

TABLE VI
BEST-TUNED PARAMETERS FOR DIFFERENT ENVIRONMENTS

Algorithm	slowly changing $\gamma_T = 10$	moderately changing $\gamma_T = 60$	frequently changing $\gamma_T = 150$
AW-CUCB	$w_0 = 550$	$w_0 = 250$	$w_0 = 100$
SW-CUCB	$w_s = 550$	$w_s = 250$	$w_s = 100$
D-CUCB	$\gamma = 0.9958$	$\gamma = 0.9898$	$\gamma = 0.9839$

B. State-of-the-Art Alternatives

1) Real Clouds Alternatives: We compared our proposed AW-CUCB-DP algorithm with the following alternative methods.

i) *A simple scheme:* It updates the utility of each cloud node according to the last write/read operation without considering the time-varying latency features.
ii) *Wiera's scheme:* It measures network latency periodically as inputs, and makes data placement policy by a mixed integer programming solver (Gurobi) [5].
iii) *Random scheme:* It places data chunks randomly to the cloud nodes regardless of the network conditions and storage cost of each cloud node.

2) Simulated Clouds Alternatives: We compare our proposed AW-CUCB-DP algorithm against the following state-of-the-art non-stationary bandit algorithms. The offline optimal policy with a-priori known information is also presented as a benchmark.

i) *Discounted CUCB (D-CUCB) [30]:* It averages the past observations with a discount factor γ, giving more weight to recent observations.
ii) *Sliding window CUCB (SW-CUCB) [30]:* It relies on a local empirical average of the observations within a fixed size time window.
iii) *Reset-CUCB (RS-CUCB) [33]:* It employs the same change detector (FM-PHT) to detect changes and restarts the bandit algorithm when there is an alarm.

C. Parameters Tuning

The performance of bandit algorithms also depends on tuned parameters. To this end, we performed parameter tuning in different changing environments (i.e., the respective numbers of change-points $\gamma_T = 10, 60$ and 150) to better adapt to the non-stationarity. Specifically, γ in D-CUCB is numerically searched over (0, 1) to find the target value that achieves the minimum average utility in each changing setting. The window size w_s of SW-CUCB and the initial window size w_0 of AW-CUCB-DP are tuned similarly. Specifically, we first calculated w_s and w_0 according to our theoretical analysis (Theorem 2, (26)) to obtain guide values, then tuned w_s and w_0 by data-driven approaches to obtain the target values.Two tuning parameters for exploration bonus are set as $\xi = 1$ and $L^B = 6$. For FM-PHT detector, we mainly care about large distribution changes since small changes do not incur much regret. We set $b = 1$ and $\delta = 0.5$. The RS-CUCB policy uses the same change detector (FM-PHT) and detection parameters as AW-CUCB-DP. The detailed parameter values are shown in Table VI.

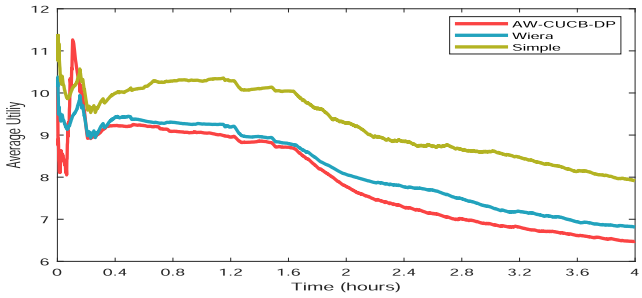


Fig. 7. Average utility under different schemes.

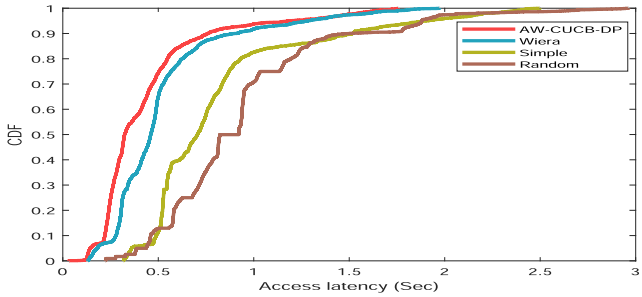


Fig. 8. CDF of access latency under different schemes.

D. Experimental Results

1) Results in Real-World Clouds.

Utility Compare: We compare our AW-CUCB-DP with three alternatives using the same data trace (i.e., Prxy1). The expected average utility is shown in Fig. 7. Our AW-CUCB-DP policy and Wiera's policy have achieved a much lower average utility, whereas the simple policy performs poorly. This is because the simple policy selects cloud nodes only based on the last observed access latency and does not estimate and learn latency performance of each node. Wiera's policy performs slightly worse than AW-CUCB-DP due to the need to use the measured bandwidth knowledge as input, which may miss changes in latency distribution. Interestingly, the expected average utility of the AW-CUCB-DP at the beginning is not necessarily better than Wiera's policy. This is because at the beginning of learning, AW-CUCB-DP is exploration-based, and all possible placement options will be explored at least once. The expected average utility peaks in the short term due to the smaller number of explorations per placement option. However, as exploration and exploitation continue, the expected average utility decreases as the number of data placement tasks increases.

Ability to Reduce Access Latency: We first evaluate AW-CUCB-DP's ability to reduce access latency based on real-world I/O traces, and Fig. 8 plots the access latency under different placement schemes using the same data trace (i.e., Prxy1). AW-CUCB-DP can effectively decrease access latency and reduce the latency variance compared to other alternatives. AW-CUCP-DP policy and Wiera's policy can significantly reduce the 100th percentile access latency compared to simple policy and random policy. Simple policy gets the worst latency performance except for the random scheme. This is because the simple scheme does

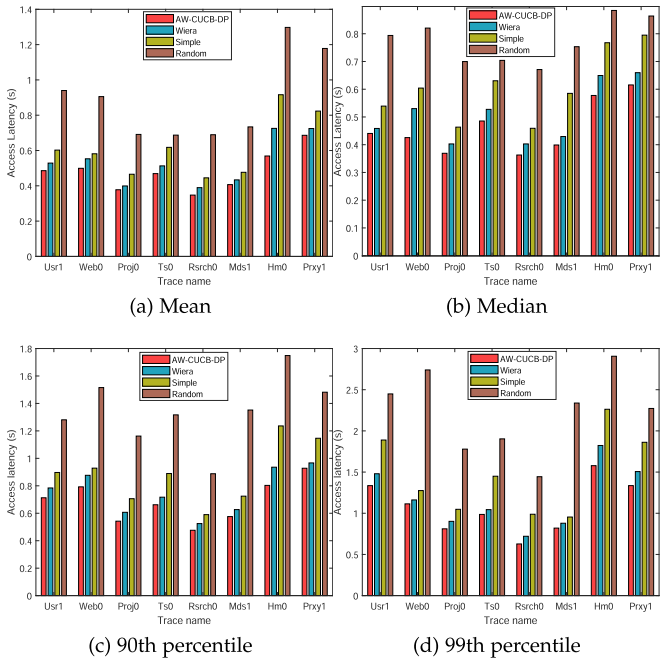


Fig. 9. Access latency under different policies.

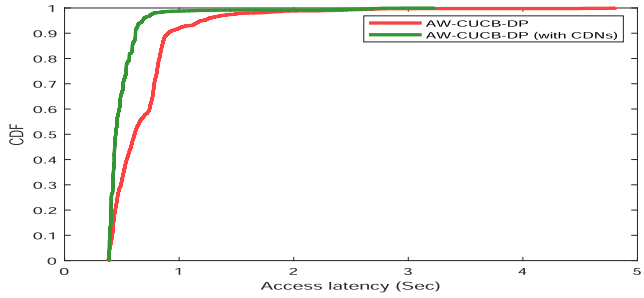


Fig. 10. Impact of CDNs on access latency.

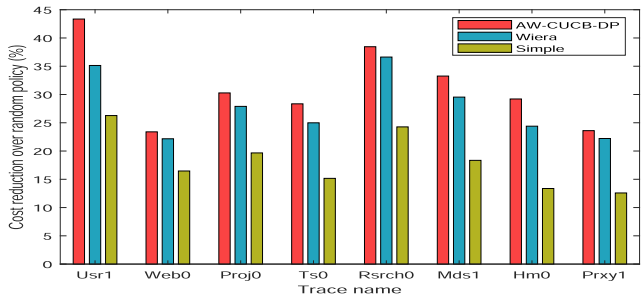


Fig. 11. Cost reduction against random policy.

not learn the latency performance of cloud nodes, thus being affected significantly by network latency fluctuations.

We further validate the performance of each placement scheme by evaluating the access latency under different dynamic workloads. Fig. 9 shows the results of average and tail access latency. Specifically, under different workloads, our AW-CUCB-DP reduces average latency by 18.88%–37.90% and 41.76%–56.15% compared to simple schemes and random policy, respectively. For the 90th (99th) percentile latency, our AW-CUCB-DP outperforms simple schemes by 20.55%–35.05% (14.08%–36.52%) among different workloads and outperforms random policy by 37.38%–57.42% (40.22%–62.92%). Our scheme reduces the inflation in the 99th percentile access latency to 1.6×–3× the median latency among different workloads. The latency reduction is mainly attributed to the AW-CUCB-DP's well capabilities for learning and handling changes.

Impact of Adopting CDNs: We also validate the performance variation achieved by integrating CDN services. We integrated storage services with the respective CDN service of each provider, e.g., combing Amazon CloudFront with S3 standard, Microsoft Blob Storage with Azure CDN, Aliyun OSS storage with their CDN, and Tencent COS storage with their CDN. We performed the comparison by replaying the "Mds1" trace, which is a read-intensive trace.

Fig. 10 shows the distributions of access latency obtained by AW-CUCB-DP and AW-CUCB-DP with CDNs. The former accesses objects from storage buckets, and the latter accesses objects from the CDN nodes. When the edge node does not cache the requested objects, a back-to-origin request to the storage bucket will be issued. As shown in Fig. 10, the beneficial impact of CDNs in terms of access performance is not particularly obvious, as it deliver better performance on average 23%. However, adopting CDNs makes significant improvement in tail access latency, AW-CUCB-DP (with CDNs) reduces the 90th (99th) percentile access latencies by 28% (43%). This may be related to some complex factors, for example, the locations of edge nodes (level-1, level-2) of each cloud provider. In addition, CDN caching cannot reduce access latency of PUT requests.

Notably, CDNs are associated to roughly the same storage and data transfer costs as object storage, but to markedly higher cost for download requests (e.g., Amazon CloudFront is around 2×).

Cost Comparison: Besides the latency performance, which significantly affects user experience, the cost is also a critical metric for application providers. We compare the cost overhead of different placement schemes in different workloads. The storage cost reduced by different policies over random policy is shown in Fig. 11. Clearly, AW-CUCB-DP performs best and exhibits well adaptability to dynamic workloads. The cost savings are different for each trace with different load characteristics, ranging from 23.39%–43.35%.

Data Migration Impacts: To verify the impact of data migration on the expected average utility and future access performance, we compare the proposed AW-CUCB-DP with the version that does not contain the proactive local data migration policy (PLDM), referred to as AW-CUCB-DP (no migration). We performed the comparison by replaying the read-intensive "Usr1" trace.

As the number of placement tasks increases, the average utility of AW-CUCB-DP shows a decreasing trend (Fig. 12). The average utility gap between AW-CUCB-DP and AW-CUCB-DP (no migration) is gradually increasing. This is because as the network latency distribution changes, some chunks of objects are migrated, benefiting subsequent access requests. The comparison verifies that active local chunk migration assists well in achieving our optimization goals.

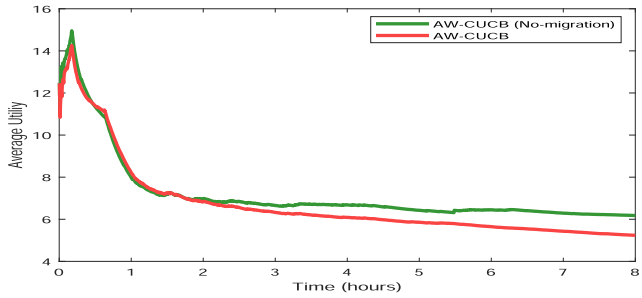


Fig. 12. Impact of migration on average utility.

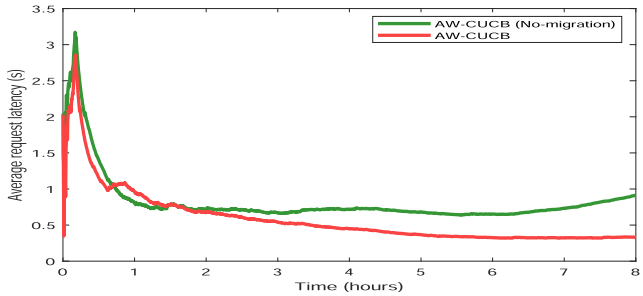


Fig. 13. Impact of migration on access latency.

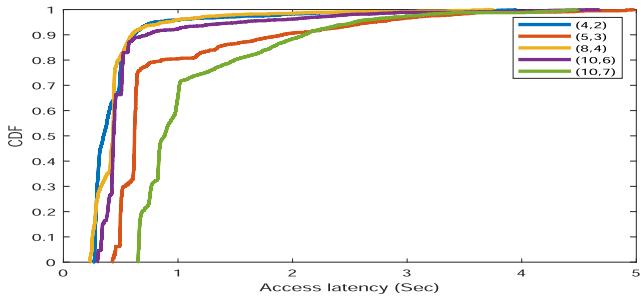
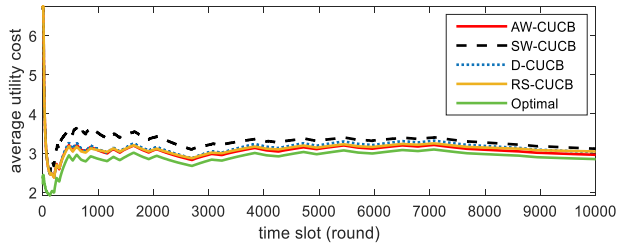


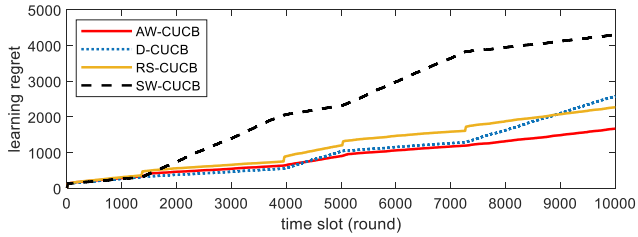
Fig. 14. CDF of access latency under different erasure codes.

From Fig. 13, with the increase in the number of placement tasks, the average access latency of AW-CUCB-DP tends to be more stable, whereas AW-CUCB-DP (no migration) shows a gradual upward trend. With proactive local chunk migrations, the average access latency is effectively reduced and the access performance is further improved.

Erasure Coding Parameters Impacts: To evaluate the impact of different erasure coding parameters on access performance, we respectively replay the "Mds1" trace to run AW-CUCB-DP utilizing widely used erasure coding parameters, i.e., $(n, k) = (4, 2), (5, 3), (8, 4), (10, 6), (10, 7)$. The access latency distributions are shown in Fig. 14. We note that AW-CUCB-DP under erasure codes $(4, 2)$, $(8, 4)$, and (10, 6) outperforms (5, 3) and (10, 7) codes in access latency due to a higher level of redundancy, where 90% of requests with erasure code $(4, 2)$ complete within 0.62 s, while $(10, 7)$ code complete within 2.15 s. In addition, erasure codes $(4, 2)$ and (8,4) outperforms $(10, 6)$ slightly in access latency due to accessing fewer-but-larger chunks is much faster than multiple smaller one in general.



(a) Average utility



(b) Learning regret

Fig. 15. Comparison of average utility and learning regret.

2) Results in Simulated Clouds: We run the simulation experiments 100 times for all algorithms and show the average results.

Utility and Regret: We compare our AW-CUCB-DP with three alternatives when the environment does not change frequently ($\gamma_T = 10$). The expected average utility and cumulative regrets are shown in Fig. 15. When the access latency distribution changes slowly, each policy achieves a much lower average utility, as shown in Fig. 15(a).

In Fig. 15(b), we compare the regrets under different schemes. In particular, we tune the D-CUCB policy under an appropriate discount factor γ to better handle changes in the environment. It can be observed that except SW-CUCB algorithm, other algorithms perform well in slowly changing environment. Due to the variable sliding window can better adapt to changes of the environment, the regret growth of our AW-CUCB-DP is smaller and smoother. The saw-tooth behavior of RS-CUCB is attributed to frequent restarts, and regrets accumulate quickly around these restart points. The SW-CUCB policy with fixed window size does not consider distribution changes, resulting in a larger regret.

Adaptation to Highly Dynamic Environments: To compare the robustness, we simulate two non-stationary environments. Notably, the two settings, with change points $\gamma_T = 60$ and 150, represent cases where the environment changes moderately and frequently. Change points under two settings are introduced at time slots where the next element of the sequence $\{\lfloor t^{\nu} \rfloor\}_{t \in \{1,\ldots,T\}}$ is different from the current element, and $\nu = 0.45, 0.55$, respectively.

From the cumulative regrets of different policies, shown in Figs. 16 and 17, we can see the effectiveness and robustness of our AW-CUCB-DP algorithm, which can handle frequent changes and exhibits better learning capability. In addition, it

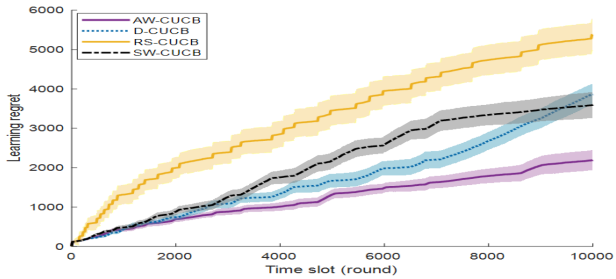


Fig. 16. Regret comparison under $\gamma_T = 60$.

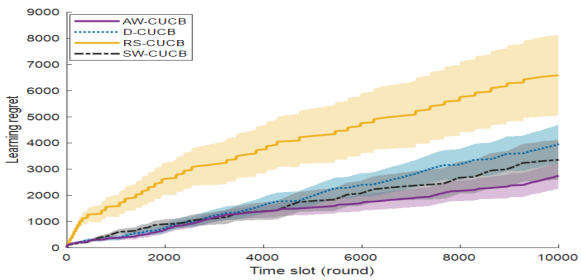


Fig. 17. Regret comparison under $\gamma_T = 150$.

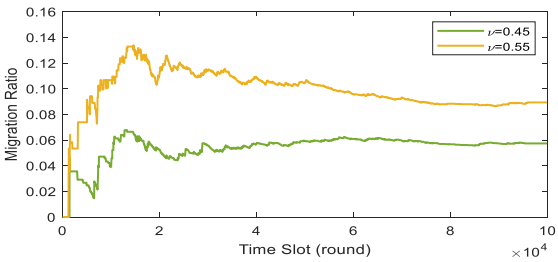


Fig. 18. Migration ratio of PLDM algorithm under highly dynamic environments.

can be observed that D-CUCB and SW-CUCB policies outperform RS-CUCB under a frequently-varying environment. The frequent restart behavior makes the cumulative regret of RS-CUCB frequently fluctuate (sawtooth behavior) and leads to a larger regret growth. A much wider variance on the RS-CUCB plot demonstrates how the performance of RS-CUCB can vary significantly compared to the narrow one on the AW-CUCB-DP, demonstrating the stability and adaptability to a highly dynamic environment.

Impacts of Nonstationarity on Data Migration: To evaluate the impacts of environment's nonstationarity on data migration, we run AW-CUCB-DP under different changing environment settings of $\nu = 0.45$ and 0.55 by replaying the "Web0" trace, respectively, then observe the number of object moves triggered by changes in latency distribution. We plot the percentage of total moves to object retrievals, referred to as migration ratio. As shown in Fig. 18, our PLDM algorithm can improve performance-cost balance with only small movements in highly dynamic environments. Specifically, migration ratios are quite low, typically 5%–6% and 9%–11% under $\nu = 0.45$ and 0.55, respectively.

Performance-Cost Balance: By tuning ψ_1/ψ_2, the proposed AW-CUCB-DP algorithm can achieve rich trade-offs between

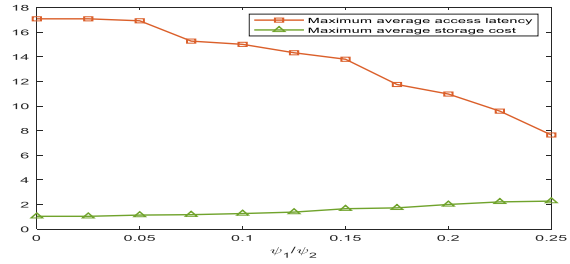


Fig. 19. Trade-off between access latency and storage cost.

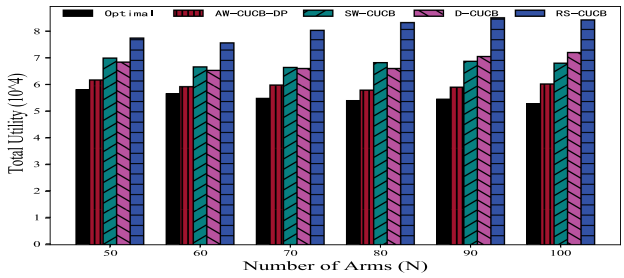


Fig. 20. Impact of N on total utility.

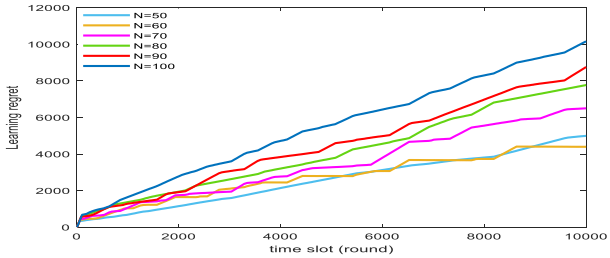


Fig. 21. Impact of N on learning regret of AW-CUCB-DP.

access latency and storage cost for different application scenarios, as shown in Fig. 19. It can be observed that as the ratio ψ_1/ψ_2 increases, our scheme focuses on optimizing performance, and vice versa on optimizing storage cost.

Parameter Impacts: To evaluate how AW-CUCB-DP scales with the size of arms, we use the bootstrap method [46] to resample the available measurement data of $N = 6$ commercial clouds and generate cloud parameter datasets with larger N. Specifically, set the number of base arms N (i.e., the number of cloud nodes) among {50, 60, 70, 80, 90, 100}, and let $n = 20$ by default.

We analyze the impact of N on the performance under a moderately changing environment (i.e., $\gamma_T = 60$), as shown in Figs. 20 and 21. In any setting, AW-CUCB-DP converges quickly, and achieves the smallest total utility, i.e., 12.26%, 12.34%, and 28.11% smaller than SW-CUCB, D-CUCB, and RS-CUCB, respectively. Moreover, even as N increases, the gap between AW-CUCB-DP and optimal (i.e., "regret") almost remains constant.

VII. Conclusion

In this paper, we have studied the data placement optimization problem of multi-cloud storage. To handle dynamics in workloads and network conditions, we have leveraged a CMAB

framework to formulate and solve this placement problem by learning data placement strategy online. To adapt to both gradual and abrupt variations in the environment, we have proposed AW-CUCB-DP by combining the active change detection and the passive sliding-window method, enabling adaptive data placement with low latency and cost. We have provided a strong performance guarantee by proving that its regret is bounded by $\mathcal{O}(n\sqrt{T\gamma_T \ln T})$. Trace-driven experimental results show that AW-CUCB-DP can improve the access performance of multi-cloud storage and reduce storage cost.

In future work, we consider optimizing the unknown parameters (e.g., the sliding window size) with the idea of bandit over bandit. Specifically, formulating the selection of unknown parameters as another multi-armed bandit problem. Under this formulation, a separate algorithm (e.g., adversarial bandit algorithms) is maintained to learn unknown parameters, and the returned parameter is provided to run SW-CUCB to better adapt to non-stationary environments.

REFERENCES

[1] M. Li, C. Qin, and P. P. Lee, "CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," in *Proc. USENIX Annu. Tech. Conf.*, Santa Clara, CA, Jul. 6–7, 2015, pp. 111–124.

[2] Q. Zhang, S. Li, Z. Li, Y. Xing, Z. Yang, and Y. Dai, "CHARM: A cost-efficient multi-cloud data hosting scheme with high availability," *IEEE Trans. Cloud Comput.*, vol. 3, no. 3, pp. 372–386, Third Quarter 2015.

[3] A. Bessani, M. Correia, B. Quaresma, F. Andre, and P. Sousa, "DepSky: Dependable and secure storage in a cloud-of-clouds," *ACM Trans. Storage*, vol. 9, no. 4, pp. 1–33, 2013.

[4] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "SPANStore: Cost-effective georeplicated storage spanning multiple cloud services," in *Proc. ACM Symp. Operating Syst. Princ.*, Farmington, PA, Nov. 3–6, 2013, pp. 292–308.

[5] K. Oh, N. Qin, A. Chandra, and J. Weissman, "Wiera: Policy-driven multi-tiered geo-distributed cloud storage system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 294–305, Feb. 2020.

[6] A. Ruiz-Alvarez and M. Humphrey, "A model and decision procedure for data storage in cloud computing," in *Proc. IEEE/ACM 12th Int. Symp. Cluster Cloud Grid Comput.*, Washington, DC, May 13–16, 2012, pp. 572–579.

[7] L. Jiao, J. Lit, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *Proc. IEEE Conf. Comput. Commun.*, Toronto, Canada, Apr. 27-May, 2014, pp. 28–36.

[8] B. Yu and J. Pan, "A framework of hypergraph-based data placement among geo-distributed datacenters," *IEEE Trans. Serv. Comput.*, vol. 13, no. 3, pp. 395–409, May/Jun. 2020.

[9] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, and H. Bhogan, "Volley: Automated data placement for geodistributed cloud services," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, San Jose, CA, Apr. 28–30, 2010, Art. no. 2.

[10] M. Xia, M. Saxena, M. Blaum, and D. A. Pease, "A tale of two erasure codes in HDFS," in *Proc. USENIX Conf. File Storage Technol.*, Santa Clara, CA, Feb. 22–25, 2015, pp. 213–226.

[11] M. Su, L. Zhang, Y. Wu, K. Chen, and K. Li, "Systematic data placement optimization in multi-cloud storage for complex requirements," *IEEE Trans. Comput.*, vol. 65, no. 6, pp. 1964–1977, Jun. 2016.

[12] Y. Hu and D. Niu, "Reducing access latency in erasure coded cloud storage with local block migration," in *Proc. IEEE 35th Annu. Int. Conf. Comput. Commun.*, San Francisco, Apr. 10–15, 2016, pp. 1–9.

[13] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, and D. Yuan, "A genetic algorithm based data replica placement strategy for scientific applications in clouds," *IEEE Trans. Serv. Comput.*, vol. 11, no. 4, pp. 727–739, Jul./Aug. 2018.

[14] G. Liang and U. C. Kozat, "FAST CLOUD: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 2012–2025, Dec. 2014.

[15] Y. Cui et al., "TailCutter: Wisely cutting tail latency in cloud CDNs under cost constraints," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1612–1628, Aug. 2019.

[16] Z. Wu, C. Yu, and H. V. Madhyastha, "CosTLO: Cost-effective redundancy for lower latency variance on cloud storage services," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, Oakland, CA, USA, May 4–6, 2015, pp. 543–557.

[17] L. Suresh, M. Canini, S. Schmid, and A. Feldmann, "C3: Cutting tail latency in cloud data stores via adaptive replica selection," in *Proc. Symp. Netw. Syst. Des. Implementation*, 2015, pp. 513–527.

[18] H. Wang, H. Shen, Z. Li, and S. Tian, "GeoCol: A geo-distributed cloud storage system with low cost and latency using reinforcement learning," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst.*, Jul. 7–10, 2021, pp. 149–159.

[19] W. Chen, Y. Wang, Y. Yuan, and Q. Wang, "Combinatorial multi-armed bandit and its extension to probabilistically triggered arms," *J. Mach. Learn. Res.*, vol. 17, no. 50, pp. 1746–1778, Jan. 2016.

[20] A. Slivkins, "Introduction to multi-armed bandits," *Found. Trends Mach. Learn.*, vol. 12, pp. 1–286, 2019.

[21] S. Barrachina-Muñoz, A. Chiumento, and B. Bellalta, "Stateless reinforcement learning for multi-agent systems: The case of spectrum allocation in dynamic channel bonding WLANs," in *Proc. IEEE Wireless Days*, Paris, France, Jun. 30-Jul. 2, 2021, pp. 1–5.

[22] C. Jin, Z. Allen-Zhu, S. Bubeck, and M. I. Jordan, "Is Q-learning provably efficient?," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, Montréal, Canada, Dec. 3–8, 2018, pp. 4868–4878.

[23] R. M. Perera, B. Oetomo, B. I. P. Rubinstein, and R. Borovica-Gajic, "No DBA? No regret! Multi-armed bandits for index tuning of analytical and HTAP workloads with provable guarantees," 2021, *arXiv:2108.10130*.

[24] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, pp. 235–256, 2002.

[25] A. Atrey, G. V. Seghbroeck, H. Mora, F. D. Turcka, and B. Volckaert, "SpeCH: A scalable framework for data placement of data-intensive services in geo-distributed clouds," *J. Netw. Comput. Appl.*, vol. 142, pp. 1–14, 2019.

[26] X. Ren, P. London, J. Ziani, and A. Wierman, "Datum: Managing data purchasing and data placement in a geodistributed data market," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 893–905, Apr. 2018.

[27] K. Liu, J. Peng, J. Wang, W. Liu, Z. Huang, and J. Pan, "Scalable and adaptive data replica placement for geo-distributed cloud storages," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1575–1587, Jul. 2020.

[28] Y. Gai, B. Krishnamachari, and R. Jain, "Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations," *IEEE/ACM Trans. Netw.*, vol. 20, no. 5, pp. 1466–1478, Oct. 2012.

[29] Z. Zhu, T. Liu, Y. Yang, and X. Luo, "BLOT: Bandit learning-based offloading of tasks in fog-enabled networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2636–2649, Dec. 2019.

[30] A. Garivier and E. Moulines, "On upper-confidence bound policies for switching bandit problems," in *Proc. Int. Conf. Algorithmic Learn. Theory*, Espoo, Finland, Oct. 5–7, 2011, pp. 174–188.

[31] F. Liu, J. Lee, and N. Shroff, "A change-detection based framework for piecewise-stationary multi-armed bandit problem," in *Proc. Conf. Assoc. Advance. Artif. Intell.*, New Orleans, LA, Feb. 2–7, 2018, pp. 3651–3658.

[32] Y. Cao, Z. Wen, B. Kveton, and Y. Xie, "Nearly optimal adaptive procedure with change detection for piecewise-stationary bandit," in *Proc. Int. Conf. Artif. Intell. Statist.*, Okinawa, Japan, Apr. 16–18, 2019, pp. 418–427.

[33] H. Zhou, L. Wang, L. Varshney, and E. Lim, "A near-optimal change-detection based algorithm for piecewise-stationary combinatorial semi-bandits," in *Proc. Conf. Assoc. Advance. Artif. Intell.*, New York, NY, Feb. 7–12, 2020, pp. 6933–6940.

[34] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, pp. 100–115, 1954.

[35] M. Uluyol et al., "Near-optimal latency versus cost tradeoffs in geo-distributed storage," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, Santa Clara, CA, USA, Feb. 25–27, 2020, pp. 157–180.

[36] K. Hsieh et al., "Gaia: Geo-Distributed machine learning approaching LAN speeds," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, Boston, MA, 2017, pp. 629–647.

[37] Q. Pu et al., "Low latency geo-distributed data analytics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 421–434, 2015.

[38] I. Narayanan et al., "Towards a leaner geo-distributed cloud infrastructure," in *Proc. USENIX Workshop Hot Top. Cloud Comput.*, Philadelphia, PA, 2014, Art. no. 3.

[39] J. Shen, K. Zhang, J. Gu, Y. Zhou, and X. Wang, "Efficient scheduling for multi-block updates in erasure coding based storage systems," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 573–581, Apr. 2018.

[40] M. Min, L. Xiao, C. Xie, M. Hajimirsadeghi, and N. B. Mandayam, "Defense against adcanced persistent threats in dynamic cloud storage: A colonel blotto game approach," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4250–4261, Dec. 2018.
[41] B. Wu, T. Chen, W. Ni, and X. Wang, "Multi-agent multi-armed bandit learning for online management of edge-assisted computing," *IEEE Trans. Commun.*, vol. 69, no. 12, pp. 8188–8199, Dec. 2021.
[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] Z. Shen, P. P. Lee, J. Shu, and W. Guo, "Encoding-aware data placement for efficient degraded reads in XOR-coded storage systems: Algorithms and evaluation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 12, pp. 2757–2770, Dec. 2018.
[44] E. Tomes, E. N. Rush, and N. Altiparmak, "Towards adaptive parallel storage systems," *IEEE Trans. Comput.*, vol. 67, no. 12, pp. 1840–1848, Dec. 2018.
[45] 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 Symp. Operating Syst. Des. Implementation*, Savannah, GA, USA, Nov. 2–4, 2016, pp. 401–417.
[46] B. Kveton, C. Szepesvari, S. Vaswani, Z. Wen, M. Ghavamzadeh, and T. Lattimore, "Garbage in, reward out: Bootstrapping exploration in multi-armed bandits," in *Proc. ACM Int. Conf. Mach. Learn.*, Long Beach, CA, Jun. 9–15, 2019, pp. 3601–3610.

Li Li received the MS degree in computer software and theory from Yunnan University, China, in 2008. She is currently working toward the PhD degree with the School of Computer Science, Fudan University, China. She is currently an engineer with Informatization Office, Yunnan University, China. Her research interests lie in cloud storage, erasure-coded storage, and cloud computing.

Jiajie Shen received the MS degree from East China Normal University, China, in 2014, and the PhD degree from Fudan University, China, in 2019. He is currently working as an engineer with Informatization Office, Fudan University. His research interests include cloud storage and erasure-coded storage systems.

Bochun Wu (Senior Member, IEEE) received BSc, MEng, and PhD degrees in electronic engineering from Fudan University, Shanghai, China, in 2007, 2011, and 2021, respectively. He is currently an associate professor with Fudan University. His research interests include Wi-Fi/5 G/6G HetNets, MEC, IoT/IoV, resource allocation, machine learning, WSN and CPS. He served as a publication co-chair for CCF CHINANET, a Symposium TPC member for IEEE GLOBECOM and VTC, and a guest editor for *Sensors*.

Yangfan Zhou (Member, IEEE) received the BSc degree from Peking University, in 2000, and the MPhil and PhD degrees from the Chinese University of Hong Kong, in 2006 and 2009, respectively. He is currently a professor with Fudan University. Before joining Fudan, he was a research staff member with The Chinese University of Hong Kong from 2009 to 2014. His research interests include software engineering and computer-supported cooperative work.

Xin Wang (Member, IEEE) received the BS degree in information theory and the MS degree in communication and electronic systems from Xidian University, China, in 1994 and 1997, respectively, and the PhD degree in computer science from Shizuoka University, Japan, in 2002. He is a professor with Fudan University, Shanghai, China. His research interests include quality of network service, next-generation networks, mobile Internet, and network coding. He is a distinguished member of CCF.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of computer science with the State University of New York. He is also a National distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, Big Data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 940 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds more than 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top 5 most influential scientists in parallel and distributed computing based on a composite indicator of Scopus citation database. He is currently an associate editor of the *ACM Computing Surveys* and the *CCF Transactions on High Performance Computing*. He has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*.