

Competitive Caching for Distributed Data Access

Tianyu Zuo
Nanyang Technological University
Singapore
zuot0001@e.ntu.edu.sg

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

Bu Sung Lee
Nanyang Technological University
Singapore
ebslee@ntu.edu.sg

Abstract—We study an online cost optimization problem for distributed storage and access. The objective of this problem is to dynamically transfer, create, and delete copies of data objects over time at distributed servers to serve access requests, while minimizing the total cost of storage and network transfers. We develop an online algorithm and prove that it achieves a competitive ratio of $\max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$, where $\mu_{\max}$ is the max/min ratio of storage cost rates among all servers. This improves on the best known algorithm which has a competitive ratio of up to 3. Examples are given to confirm the tightness of competitive analysis. We also conduct simulation experiments to evaluate our proposed algorithm, and compare it with a state-of-the-art algorithm.

I. INTRODUCTION

The exponential growth of data in today's digital age has transformed data storage into a critical infrastructure that supports various industries and services. With growing demands for seamless data access, companies are increasingly relying on geo-distributed systems such as clouds to store, manage, and serve vast amounts of data efficiently. These systems enable rapid access to data from diverse geographical locations, providing scalability and flexibility that traditional centralized storage models struggle to match.

Managing distributed storage systems, however, presents significant challenges, especially in minimizing operational costs. With data spread across multiple servers and regions, there is an intrinsic trade-off between the cost of storing data redundantly to ensure availability and the cost of transferring data across networks to satisfy access requests [1]. It is generally believed that in these systems, storage costs are typically proportional to the time duration of storing data objects in each server, while network costs are related to the amount of network traffic for transferring data among servers [2], [3]. This makes cost management particularly complicated, as both factors are influenced by dynamic and often unpredictable access patterns.

Several studies have been carried out to minimize the overall operational costs of distributed storage systems, while tackling the uncertainties of user access requests for data objects in the future. Boukhelef *et al.* [4] introduced heuristic algorithms aimed at periodically optimizing the allocation of data objects between two types of storage devices, i.e., SSDs and HDDs, and striking a balance between performance and monetary cost. Mansouri *et al.* [5] developed data placement algorithms for cloud storage services that provide two storage tiers, each offering distinct quality of service and pricing, based on historical user access patterns. Veeravalli [6] addressed a

broader issue involving the migration and caching of shared data across server networks. He utilized dynamic programming to derive the minimum cost, assuming complete knowledge of future user requests. Similar studies were also conducted in [7]–[9], focusing on varying aspects of user request patterns or service pricing models.

All the above works either utilize historical user access traces for decision-making, or are based on an assumption that the future access pattern is fully known beforehand. Therefore, these methods lack performance guarantees if the access pattern changes in the future. Competitive analysis is a widely used method to characterize the impact of uncertainty. In competitive analysis, a metric called *competitive ratio* is used to evaluate an online algorithm (without knowledge of the future), which compares it against the optimal offline strategy (with full knowledge of the future) in the worst case scenario [10], [11]. There were some works that used competitive analysis for the cost optimization of distributed storage and access. Bar-Yehuda *et al.* [12] developed an online algorithm with a competitive ratio of $O(\log \delta)$ under uniform storage cost rates of servers, where δ represents the normalized diameter of the underlying network. Wang *et al.* [13] introduced an optimal offline strategy and proposed a 3-competitive online algorithm for data storage and transfer. Recently, we proposed an online algorithm with randomized periods of storing data copies among all servers [14]. This algorithm achieves an expected competitive ratio of $1 + \frac{\sqrt{2}}{2}$. In addition, we also considered incorporating machine-learned predictions to facilitate online decision-making. We developed a learning-augmented algorithm [15], which was shown to be $(\frac{5+\alpha}{3})$ -consistent, representing its competitive ratio under perfect predictions, and $(1 + \frac{1}{\alpha})$ -robust, indicating its competitive ratio under poor predictions, where $\alpha \in [0, 1]$ is a tunable hyper-parameter.

The above works, however, are based on an assumption that all the servers in the system have the same storage cost rates, which largely restricts their application scope. In light of this, Wang *et al.* [16], [17] designed an optimal offline strategy and brought up an online algorithm for a system of servers with potentially different storage cost rates. They claimed that their online algorithm has a competitive ratio of 2. However, we found that its competitive ratio is actually at least 3, by providing a counterexample in our earlier work [18], [19]. We also presented an online algorithm that has a competitive ratio of no more than 3.

In this paper, we aim at developing a better online algorithm for the cost optimization of distributed storage and access, with

servers having distinct storage cost rates. Our contributions are summarized as follows.

- 1) We derive a lower bound of the optimal cost based on some key properties of an optimal offline strategy (Section III).
- 2) We propose an online algorithm for data storage and transfer and prove that it has a competitive ratio of $\max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$ (Section IV), where $\mu_{\max}$ is the max/min ratio of storage cost rates among all servers. This improves on the competitive ratio of $\max\{2, \min\{\mu_{\max}, 3\}\}$ achieved in our earlier work [18], [19]. While our earlier work performed competitive analysis by an induction method, this paper carries out competitive analysis using a very different approach by directly comparing the total cost of the online algorithm against the aforesaid lower bound of the optimal cost.
- 3) We conduct simulation experiments to evaluate the proposed online algorithm (Section V), and empirically compare it with the online algorithm by Wang *et al.* [16], [17].

II. PROBLEM DEFINITION

Consider a data object hosted by a distributed system with a set of geo-distributed servers $\{s_1, s_2, \dots, s_n\}$. Copies of this object can be stored in any servers at any time. We refer to the cost of storing a data copy at each server s_j per unit of time as the *storage cost rate*, and denote it by $\mu(s_j)$. We assume that s_1 is a server with the lowest storage cost rate. Without loss of generality, we assume that $\mu(s_1) = 1$ for server s_1 , and $\mu(s_j) \geq 1$ for other servers s_j 's. Then, the maximum storage cost rate $\mu_{\max} = \max_j \mu(s_j)$ represents the max/min ratio of storage cost rates among all servers. The data object can be transferred among servers whenever necessary to serve access requests or create additional data copies. The cost of transfer between any pair of servers is denoted by λ .

We use $\langle r_1, r_2, \dots, r_m \rangle$ to denote the sequence of access requests arising in the system, where m is the total number of requests. For each request r_j , we use t_j to denote its arising time, and $s[r_j]$ to denote the server where it arises. For simplicity, we assume that the arising times of requests are distinct and the requests are arranged in the chronological order in the sequence, i.e., $t_1 < t_2 < \dots < t_m$. Each request r_i can be served locally by a data copy at server $s[r_i]$ if available, or be served by a transfer from another server which has a data copy. After a transfer in response to a request, the transferred data may be kept or discarded by both the sending and receiving servers.

Initially, at time 0, there is only one copy of the object placed in server s_1 . We aim to develop a *caching strategy* that determines the data copies to create and hold over time in different servers and the transfers to carry out between servers in an *online manner* to serve all requests. To preserve the object in the system, there must be at least one copy kept in some server at any time. Our objective is to minimize the total storage and transfer cost of serving the whole request sequence.

To facilitate explaining our proposed algorithm, we define a dummy request r_0 arising at server s_1 (the server where the initial copy is located) at time 0. Note that r_0 does not incur any additional cost for serving the request sequence. Moreover, to simplify the representation of cost incurred by an algorithm, we further define $\mu[r_j]$ as the storage cost rate of server $s[r_j]$, i.e., $\mu[r_j] = \mu(s[r_j])$.

III. OPTIMAL OFFLINE STRATEGY

A. Properties of an Optimal Offline Strategy

We start by presenting some important properties of an optimal offline strategy, which partly motivates our algorithm design and lays a foundation of our competitive analysis. To facilitate presentation, we define $r_{p(i)}$ as the preceding request of r_i arising at the same server as r_i does. For example, in the request sequence of Figure 3, $r_{p(7)} = r_2$, $r_{p(3)} = r_1$, and $r_{p(10)} = r_6$. If r_i is the first non-dummy request at server s_1 , its preceding request is the dummy request r_0 , so we define $r_{p(i)} := r_0$ and $t_{p(i)} := 0$. If r_i is r_0 or the first request at a server $s \neq s_1$, we define $r_{p(i)} := \emptyset$ and $t_{p(i)} := -\infty$. For example, in Figure 3, $r_{p(12)} = r_0$, and $r_{p(2)} = \emptyset$.

The following two propositions were presented and proved in our earlier work [18], [19]. We include brief proof sketches here for completeness.

Proposition 1. *There exists an optimal offline strategy, such that for each request r_i , if $\mu[r_i] \cdot (t_i - t_{p(i)}) \leq \lambda$, server $s[r_i]$ holds a copy throughout the period $(t_{p(i)}, t_i)$ to serve r_i .*

Proof Sketch. In any optimal offline strategy, if $s[r_i]$ does not hold a copy throughout the period $(t_{p(i)}, t_i)$, $s[r_i]$ must receive a transfer during $(t_{p(i)}, t_i)$ in order to serve r_i . We can replace the transfer with a copy at $s[r_i]$ during $(t_{p(i)}, t_i)$ without increasing the total cost. $\square$

Proposition 2. *There exists an optimal offline strategy with the characteristic in Proposition 1 and that there is a request at either the source or destination server of each transfer.*

Proof Sketch. If there is no request at the source and destination servers of a transfer, the transfer can be advanced or delayed to save cost or keep the same cost based on the relative storage cost rates of the source and destination servers. $\square$

In what follows, an optimal offline strategy shall always refer to one with the characteristics stated in both Proposition 1 and Proposition 2.

B. A Lower Bound of Optimal Cost

The cost incurred by an optimal offline algorithm [16], [17] lacks a simple closed-form representation. To overcome this barrier to competitive analysis, we derive a lower bound of the optimal cost for serving a request sequence. For any sequence $\langle r_1, r_2, \dots, r_m \rangle$, we divide all its requests into the following five categories, and express the total cost of the optimal offline strategy in the time span $(0, t_m)$ based on the categorization, where t_m is the arising time of the final request r_m .

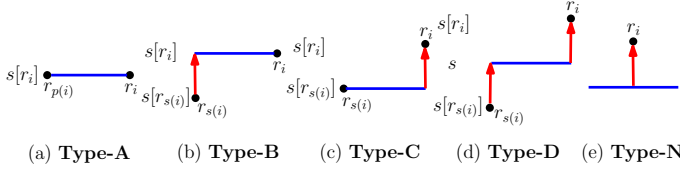


Fig. 1. Illustration of five types of requests

If a request r_i ($1 \leq i \leq m$) is served locally (by an available data copy at server $s[r_j]$), the local copy can be either (1) kept since the preceding request $r_{p(i)}$ at the same server (see Figure 1(a)), or (2) created by an inward transfer to server $s[r_i]$ after $r_{p(i)}$. In the latter case, by Proposition 2, there must be a request at the source server of this transfer. We use $r_{s(i)}$ to denote this request at the source server, where $t_{p(i)} < t_{s(i)} < t_i$ (see Figure 1(b)). In case (1), we call r_i a **Type-A** request. In case (2), we call r_i a **Type-B** request.

If the request r_i is served by a transfer (from another server which has a data copy), we examine the data copy at the source server s of this transfer. If the copy is not deleted right after the transfer, we call r_i a **Type-N** request (see Figure 1(e)). Otherwise, if the copy is deleted right after the transfer, this copy can be either (1) kept since the last local request at s before r_i (we also use $r_{s(i)}$ to denote this last local request, see Figure 1(c)), or (2) created by an inward transfer to s , and this transfer is after the last local request at s before r_i . In the latter case, by Proposition 2, there must be a request at the source server of this transfer to s . Again, we use $r_{s(i)}$ to denote this request (see Figure 1(d)). In case (1), we call r_i a **Type-C** request. In case (2), we call r_i a **Type-D** request.

Apparently, the above categorization of requests is complete. Note that if r_i is a **Type-B/C/D** request, $r_{s(i)}$ consistently denotes the request that provides the data copy to serve r_i . Thereafter, we use $A/B/C/D/N$ to denote the set of **Type-A/B/C/D/N** requests respectively. We further divide A into two subsets A_1 and A_2 : for each $r_i \in A$, if $\mu[r_i] \cdot (t_i - t_{p(i)}) > \lambda$, we define $r_i \in A_1$, otherwise we define $r_i \in A_2$, so that we have $A = A_1 \cup A_2$.

Proposition 3. *If $r_i \in A_1 \cup B \cup C \cup D \cup N$, we have $\mu[r_i] \cdot (t_i - t_{p(i)}) > \lambda$.*

Proof. If $r_i \in C \cup D \cup N$, since r_i is served by a transfer in the optimal offline strategy, by Proposition 1, we must have $\mu[r_i] \cdot (t_i - t_{p(i)}) > \lambda$. If $r_i \in B$, by definition, the transfer from $s[r_{s(i)}]$ to $s[r_i]$ is between $r_{p(i)}$ and r_i . Suppose $\mu[r_i] \cdot (t_i - t_{p(i)}) \leq \lambda$, by Proposition 1, $s[r_i]$ keeps a data copy throughout $(t_{p(i)}, t_i)$, which implies that $s[r_i]$ already holds a copy when $s[r_{s(i)}]$ transfers its copy to $s[r_i]$ at time $t_{s(i)}$. Hence, we can eliminate the transfer from $s[r_{s(i)}]$ to $s[r_i]$, to further reduce cost, which contradicts the optimality of the offline strategy. So, $\mu[r_i] \cdot (t_i - t_{p(i)}) > \lambda$ holds for each $r_i \in B$. If $r_i \in A_1$, $\mu[r_i] \cdot (t_i - t_{p(i)}) > \lambda$ holds by the definition of A_1 . $\square$

Each **Type-A/B/C/D** request r_i is associated with a data copy before r_i arises. If $r_i \in A$, the storage duration of the

copy is given by $t_i - t_{p(i)}$. If $r_i \in B \cup C \cup D$, the storage duration of the copy is given by $t_i - t_{s(i)}$. By definitions, all these copies are distinct for different **Type-A/B/C/D** requests. Therefore, the total storage cost of the optimal offline strategy is at least

$$\sum_{r_i \in A} \mu[r_i] \cdot (t_i - t_{p(i)}) + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}),$$

where we simply take the storage cost rate of the server holding the copy as 1 (the lowest storage cost rate among all servers) for **Type-B/C/D** requests (Figures 1(b)–(d)). We remark that the aforesaid storage durations associated with **Type-A/B/C/D** requests actually cover all data copies in the optimal offline strategy in the time span $(0, t_m)$. Whenever a server s holds a copy, the copy must either (i) serve the next request arising at s , or (ii) be deleted right after an outward transfer from s to serve a request at some other server. Otherwise, if the copy is deleted at a time instant t with neither a local request nor an outward transfer, it must not be the only copy in the system due to the at-least-one-copy requirement. Then, we can advance the deletion time from t to $t - \epsilon$, which reduces the storage cost and contradicts the optimality of the strategy. Hence, either (i) or (ii) must hold. In case (i), the data copy is associated with a **Type-A/B** request, while in case (ii), the data copy is associated with a **Type-C/D** request.

Each **Type-C/D/N** request is served by a transfer, and each **Type-B** request r_i is associated with a transfer originating from server $s[r_{s(i)}]$. It is obvious that all these transfers are distinct. Thus, the total transfer cost of the optimal offline strategy is at least $\lambda \cdot (|B| + |C| + |D| + |N|)$.

To conclude, the total cost **OPT** of the optimal offline strategy is at least

$$\begin{aligned} & \sum_{r_i \in A} \mu[r_i] \cdot (t_i - t_{p(i)}) + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}) \\ & + \lambda \cdot (|B| + |C| + |D| + |N|) \\ & = \sum_{r_i \in A_2} \mu[r_i] \cdot (t_i - t_{p(i)}) + \sum_{r_i \in A_1} \mu[r_i] \cdot \left((t_i - t_{p(i)}) - \frac{\lambda}{\mu[r_i]} \right) \\ & + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}) + \lambda \cdot (|A_1| + |B| + |C| + |D| + |N|) \\ & \geq \sum_{r_i \in A_2} \mu[r_i] \cdot (t_i - t_{p(i)}) + \sum_{r_i \in A_1} \left((t_i - t_{p(i)}) - \frac{\lambda}{\mu[r_i]} \right) \\ & + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}) + \lambda \cdot (|A_1| + |B| + |C| + |D| + |N|). \end{aligned} \quad (1)$$

IV. A PROPOSED ONLINE ALGORITHM

At first glance, there is a trade-off between local storage and inward transfer at every server. An intuitive algorithm design is to find the break-even point: let each server s_j keep a data copy for $\frac{\lambda}{\mu(s_j)}$ time after serving each local request and then delete the copy if no request arises at s_j during this period. This strategy appears to achieve a competitive ratio of 2, but if we simply apply this strategy at every server, the requirement of always maintaining at least one data copy may not be satisfied. The reason is that there may exist a sufficiently long silent

period without any request, so that all copies will be deleted before the next request arises.

A naive approach to address the at-least-one-copy requirement is to always keep a data copy at server s_1 which has the lowest storage cost rate. All other servers serve their local requests by transferring from s_1 when needed and adopt the aforesaid strategy to maintain a data copy for $\frac{\lambda}{\mu(s_j)}$ time after each local request. Following the above arguments, aside from the cost of storing at s_1 , the cost is at most $2 \cdot \mathbf{OPT}$. Storing a copy at s_1 all the time costs at most $\mathbf{OPT}$. Thus, the algorithm is 3-competitive. We would like to design an algorithm better than 3-competitive.

One alternative approach to address the at-least-one-copy requirement is to extend the storage of a data copy, if it is the only copy in the system. Once the next request arises and a new copy is created at some other server, this extended copy can then be dropped. Nevertheless, this strategy is not cost-efficient when the server holding the only copy has a high storage cost rate, as keeping the copy in such a server during the long silent period incurs unnecessarily high storage cost. Therefore, the aforesaid extension strategy is suitable only for servers with low storage cost rates. To improve this strategy, we can let a server, with a high storage cost rate, transfer the copy to s_1 , if it holds the only copy in the system. Thereafter, s_1 keeps the *relocated copy* during the long silent period. In this way, the storage cost is saved while the at-least-one-copy requirement is met. This was the main idea of our earlier work [18], [19]. Suppose that when a new request arises at a high-cost server afterwards, the relocated copy is dropped and a new copy is created at the high-cost server. Then, if this new copy becomes the only copy, it will be transferred to s_1 again to create the relocated copy, resulting in the copy transferred back and forth repeatedly between s_1 and high-cost servers. As illustrated by the example in Figure 2, the competitive ratio would be at least $\frac{\mu(s_1) \cdot \epsilon + \mu(s_2) \cdot \frac{\lambda}{\mu(s_2)} + 2\lambda}{\mu(s_1) \cdot (\frac{\lambda}{\mu(s_2)} + \epsilon) + \lambda} = \frac{3\lambda + \epsilon}{\frac{\lambda}{\mu(s_2)} + \lambda + \epsilon}$, which approaches 3 as $\mu(s_2)$ goes towards infinity. Since s_1 's storage cost rate is much lower than high-cost servers, continuously keeping the relocated copy at s_1 would be more cost-efficient than such transfers. Thus, in this paper, we propose to keep the relocated copy at s_1 after serving requests at high-cost servers until a request arises at some low-cost server.

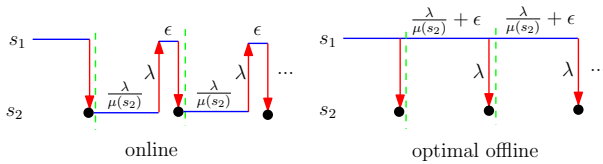


Fig. 2. An example of creating and deleting relocated copies

In our proposed online algorithm, we use a threshold of d to determine whether a server s_j has a high or low storage cost rate. i.e., if $\mu(s_j) \leq d$, its storage cost rate is considered low; otherwise, its storage cost rate is considered high. The best setting of d will be made clear in the competitive analysis. Algorithm 1 shows the details of our proposed online

algorithm. We define E_j as the *intended expiration time* of the data copy in server s_j , and define a binary tag K_j to indicate whether server s_j keeps a copy beyond its intended expiration time. We use another binary tag R to indicate whether a relocated copy is created and kept in server s_1 by a transfer from a server with a storage cost rate higher than d . We use c to denote the total number of data copies in the system. Initially, only server s_1 holds a data copy, so $c = 1$ (line 1).

When a request arises at a server s_j , if s_j does not hold a data copy, a transfer is conducted from another server with a copy to serve the request (line 8). After serving the request, s_j updates its intended expiration time to keep the copy for an *intended duration* of $\frac{\lambda}{\mu(s_j)}$ (line 12).¹ On the other hand, if s_j holds a data copy, the request is served locally (line 4). Then, s_j 's intended expiration time is also updated unless $s_j = s_1$ and s_1 holds a relocated copy (lines 11-12). Note that right before serving this request, s_j may have already kept its copy beyond the intended expiration time (line 5). In this case, we clear the binary tag K_j , indicating that the storage of this copy at s_j is no longer beyond the intended duration (lines 5-6).

In the intended duration, if a new request arises at s_j , it is served by the local copy and s_j renews the copy for another intended duration of $\frac{\lambda}{\mu(s_j)}$. When the intended duration of the copy in s_j expires (i.e., no request arises at s_j), if there are other copies kept in the system, s_j deletes its copy (lines 31-33). Otherwise, if s_j holds the only copy, it acts according to its storage cost rate $\mu(s_j)$:

- If $\mu(s_j) \leq d$, s_j keeps on storing its copy and sets its tag K_j as 1 to indicate the extension of storage (lines 29-30). In this case, s_j holds its copy until the next request arises. If the next request arises at s_j , s_j sets a new intended duration of $\frac{\lambda}{\mu(s_j)}$ and clears its tag K_j (lines 5-6 and 12). If the next request arises at another server s_k ($s_k \neq s_j$), s_j performs a transfer to s_k to serve the request. Then, s_j drops its copy and clears its tag K_j (lines 19-22). Meanwhile, a copy will be created at s_k upon receiving the transfer (lines 9-10), so the requirement of maintaining at least one copy is met.
- If $\mu(s_j) > d$, s_j performs a transfer to server s_1 to create a relocated copy at s_1 , and then the binary tag R is set to 1, implying that s_1 is holding a relocated copy thereafter. After the transfer, s_j drops its copy (lines 25-28). When s_1 holds the relocated copy, if a request arises at a server s_k ($s_k \neq s_1$) without a local copy, we further check its storage cost rate $\mu(s_k)$. If $\mu(s_k) > d$, s_1 continues to keep its relocated copy after performing a transfer to s_k to serve that request. If $\mu(s_k) \leq d$, s_1 drops its relocated copy after the transfer to s_k , and clears the R tag (lines 15-18). In both cases, a copy is created in s_k after serving its local request, and s_k keeps the copy for an intended duration of $\frac{\lambda}{\mu(s_k)}$ (lines 9-12). If a request arises at s_1 when it holds the relocated copy, the request is served locally (lines 3-4).

¹With the dummy request r_0 arising at server s_1 at time 0, s_1 sets an intended duration λ for the initial copy (line 1).

Algorithm 1 A Proposed Online Algorithm

```

1: initialize:  $c \leftarrow 1$ ;  $E_1 \leftarrow \lambda$ ;  $E_j \leftarrow -\infty$  for all  $j \geq 2$ ;
    $K_j \leftarrow 0$  for all  $j$ ;  $R \leftarrow 0$ ;  $\triangleright s_1$  holds an initial copy
2: upon (a request  $r_i$  arises at server  $s_j$  at time  $t_i$ ) do
3:   if ( $t_i \leq E_j$ ) or ( $K_j = 1$ ) or ( $j = 1$  and  $R = 1$ ) then  $\triangleright s_j$  has a copy
4:     serve  $r_i$  by the local copy in  $s_j$ ;
5:     if  $K_j = 1$  then
6:        $K_j \leftarrow 0$ ;
7:   else
8:     serve  $r_i$  by a transfer from any server with a copy;
9:     create a copy in  $s_j$ ;
10:     $c \leftarrow c + 1$ ;
11:    if not ( $j = 1$  and  $R = 1$ ) then
12:       $E_j \leftarrow t_i + \frac{\lambda}{\mu(s_j)}$ ;
13:  upon ( $s_j$  transfers the object to another server  $s_k$ ) do
14:    if  $R = 1$  then  $\triangleright s_1$  holds a relocated special copy
15:      if  $\mu(s_k) \leq d$  then
16:        drop the copy in  $s_1$ ;
17:         $R \leftarrow 0$ ;
18:         $c \leftarrow c - 1$ ;
19:      else if  $K_j = 1$  then  $\triangleright s_j$  holds a resident special copy
20:        drop the copy in  $s_j$ ;
21:         $K_j \leftarrow 0$ ;
22:         $c \leftarrow c - 1$ ;
23:  upon (a copy expires in server  $s_j$  at time  $E_j$ ) do
24:    if  $c = 1$  then  $\triangleright s_j$  holds the only copy
25:      if  $\mu(s_j) > d$  then
26:        transfer the copy to  $s_1$ ;  $\triangleright$  create a relocated
           special copy at  $s_1$ 
27:         $R \leftarrow 1$ ;
28:        drop the copy in  $s_j$  after transfer;
29:      else
30:         $K_j \leftarrow 1$ ;  $\triangleright s_j$  keeps a resident special copy
31:    else
32:      drop the copy in  $s_j$ ;
33:       $c \leftarrow c - 1$ ;

```

To facilitate presentation, we refer to the data copy kept in each server s_j during the intended duration of $\frac{\lambda}{\mu(s_j)}$ as the *regular copy*. Upon the expiration of the regular copy, when s_j holds the only copy in the system, (1) if s_j keeps on holding its copy (which implies $\mu(s_j) \leq d$), we refer to the data copy beyond the $\frac{\lambda}{\mu(s_j)}$ period as the *resident special copy*; (2) if s_j transfers its copy to s_1 and let s_1 keep the relocated copy thereafter (which implies $\mu(s_j) > d$), we refer to such a data copy created at s_1 as the *relocated special copy*. We collectively call these two types of copies stored beyond the intended duration of $\frac{\lambda}{\mu(s_j)}$ the *special copy*. Figure 3 shows a running example of Algorithm 1 for 4 servers with $\mu(s_1) = 1$, $\mu(s_2) = 2$, $\mu(s_3) = 3$, $\mu(s_4) = 4$, where regular copies are colored darkcyan, resident special copies are colored orange, relocated special copies are colored purple or green, and transfers are colored red.

According to Algorithm 1, any storage periods of special copies cannot overlap. Moreover, since a resident special copy is the only copy in the system, it cannot overlap with any regular copy. Furthermore, as a relocated special copy is deleted after performing a transfer to a low-cost server s_k (with $\mu(s_k) \leq d$), it cannot overlap with any regular copy in any low-cost server.

Observation 1. *There is no overlap between the storage periods of:*

- any two special copies;
- any relocated special copy and any regular copy in any low-cost server s_k (with $\mu(s_k) \leq d$);
- any resident special copy and any regular copy.

In what follows, we show that Algorithm 1 has a competitive ratio of $\max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$ when setting $d = \frac{5}{2}$. The main intuitions behind the analysis are as follows. For any two successive requests $r_{p(i)}$ and r_i at the same server s , if $t_i - t_{p(i)} \leq \frac{\lambda}{\mu(s)}$, Algorithm 1 maintains a regular copy in s during the period $(t_{p(i)}, t_i)$ to serve r_i and the optimal offline strategy does the same by Proposition 1, so the online-to-optimal cost ratio is 1. If $t_i - t_{p(i)} > \frac{\lambda}{\mu(s)}$, Algorithm 1 maintains a regular copy in s during the period $(t_{p(i)}, t_{p(i)} + \frac{\lambda}{\mu(s)})$ (where the storage cost incurred is λ) and r_i will typically be served by a transfer (where the cost incurred is also λ). On the other hand, the optimal offline strategy either keeps a copy in s throughout the period $(t_{p(i)}, t_i)$ to serve r_i or performs a transfer to serve r_i , thereby incurring a cost of at least λ . Thus, the online-to-optimal cost ratio is bounded by 2. Note that by Algorithm 1, a relocated special copy may be concurrently kept in server s_1 during the period $(t_{p(i)}, t_{p(i)} + \frac{\lambda}{\mu(s)})$ if $\mu(s) > d$. In this case, the total cost incurred by Algorithm 1 is up to $\frac{\lambda}{\mu(s)} \cdot \mu(s) + \lambda + \frac{\lambda}{\mu(s)} \cdot 1 < (2 + \frac{1}{d})\lambda < \frac{5}{2}\lambda$. Consequently, the online-to-optimal cost ratio is bounded by $\frac{5}{2}$. If a server s maintains a resident special copy, by Algorithm 1, the storage cost rate of s is not larger than $\min\{\mu_{\max}, d\}$ and the resident special copy must be the only copy in the system. The optimal offline strategy also needs to maintain a copy to meet the at-least-one-copy requirement during that time, where the storage cost rate of the copy is at least 1. Hence, the online-to-optimal cost ratio is bounded by $\min\{\mu_{\max}, d\} = \min\{\mu_{\max}, \frac{5}{2}\}$. In summary, the competitive ratio is $\max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$.

A. Online Cost of Algorithm 1

To analyze the competitive ratio of Algorithm 1, we can assume without loss of generality that no request arises at server s_1 when s_1 is holding a relocated special copy by Algorithm 1. This is because by the algorithm definition, the copy at s_1 remains a relocated special copy after serving a local request arising at s_1 . Hence, if we remove the local request at s_1 , it does not affect any data copy or transfer to serve any other request. Let R be any request sequence and R' be the subsequence of R with all requests arising at s_1 served by the relocated special copies removed. The only exception is the final request r_m in R : if r_m arises at s_1 and is served by

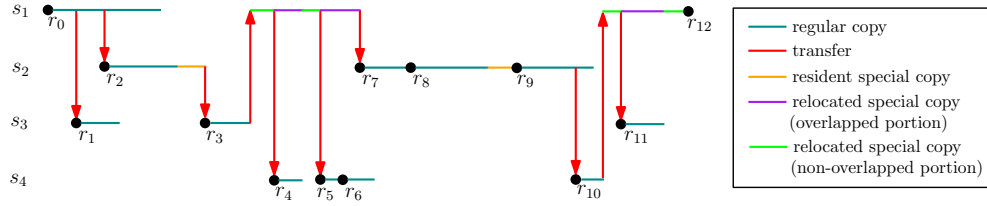


Fig. 3. A running example of Algorithm 1

a relocated special copy at s_1 , we keep r_m in R' so that the time span $(0, t_m)$ of the sequence remains unchanged. Then, the online cost **Online** for serving R equals the online cost **Online'** for serving R' . Meanwhile, the optimal cost **OPT** for serving R must be no less than the optimal cost **OPT'** for serving R' . Thus, it holds that $\frac{\text{Online}}{\text{OPT}} \leq \frac{\text{Online}'}{\text{OPT}'}$. Therefore, we just need to focus on studying the online-to-optimal cost ratios of request sequences where there is no request arising at server s_1 when s_1 is holding a relocated special copy by Algorithm 1, except the final request r_m . This implies that each request arising at s_1 is followed by a regular copy, except possibly the final request r_m . This helps to simplify the analysis.

We now derive the total online cost of serving the request sequence $\langle r_1, r_2, \dots, r_m \rangle$ by Algorithm 1. To do so, we classify all the requests into the following six categories as illustrated in Figure 4, based on how they are served in Algorithm 1.

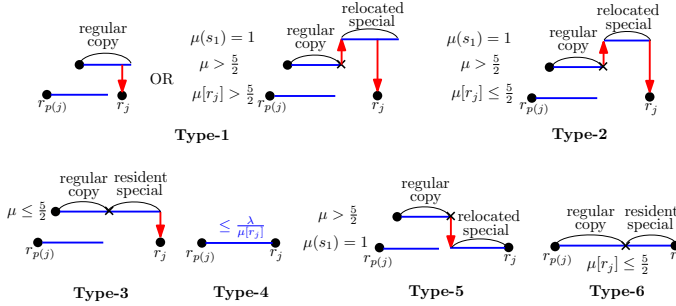


Fig. 4. Different types of requests in Algorithm 1

For each request r_j served by a transfer, at the time of the transfer, if the source server holds a regular copy, we call r_j a **Type-1** request. If the source server holds a relocated special copy (it implies that the source server is s_1), and this copy is not deleted after the transfer (in other words, $\mu[r_j] > d$ so that s_1 keeps its relocated special copy after r_j), we also call r_j a **Type-1** request. Meanwhile, if this relocated special copy is deleted after the transfer (it implies that $\mu[r_j] \leq d$), we call r_j a **Type-2** request. If the source server holds a resident special copy at the time of the transfer, this copy must be dropped after r_j , and we call r_j a **Type-3** request.

For each request r_j served locally, if the local copy is a regular copy, we call r_j a **Type-4** request. If the local copy is a relocated special copy (it implies that r_j is at server s_1), we call r_j a **Type-5** request. Note that by the aforesaid assumption, there can be at most one **Type-5** request, which

is the final request r_m . If the local copy is a resident special copy (it implies that $\mu[r_j] \leq d$), we call r_j a **Type-6** request.

Obviously, the above request categorization is complete since there are only three types of copies by Algorithm 1.

In Figure 3, r_5 and r_{10} are **Type-1** requests, r_7 is a **Type-2** request, r_3 is a **Type-3** request, r_6 and r_8 are **Type-4** requests, r_{12} is a **Type-5** request, and r_9 is a **Type-6** request.

Proposition 4. If r_j is not a **Type-4** request, then $t_j - t_{p(j)} > \frac{\lambda}{\mu[r_j]}$. If r_j is a **Type-4** request, then $t_j - t_{p(j)} \leq \frac{\lambda}{\mu[r_j]}$.

Proof. If r_j is a **Type-1/2/3** request, since it is served by a transfer from some other server, the regular copy in $s[r_j]$ after $r_{p(j)}$ must be deleted before r_j arises. Thus, $t_j - t_{p(j)} > \frac{\lambda}{\mu[r_j]}$. If r_j is a **Type-5/6** request, by definition, $t_j - t_{p(j)} > \frac{\lambda}{\mu[r_j]}$. If r_j is a **Type-4** request, as it is served locally by the regular copy in $s[r_j]$ after $r_{p(j)}$, we have $t_j - t_{p(j)} \leq \frac{\lambda}{\mu[r_j]}$. $\square$

If the final request r_m arises at s_1 and is served by a relocated special copy at s_1 , the relocated special copy will stay infinitely after r_m . Otherwise, after serving the final request r_m , a regular copy is created in server $s[r_m]$. Among this regular copy and all other regular copies that exist after r_m , the copy expiring the latest would either switch to a resident special copy, or be transferred to s_1 to create a relocated special copy. In either case, the special copy will stay infinitely. In our analysis, we shall not account for the cost of the regular copy (if any) created in server $s[r_m]$ after r_m and the special copy which stays infinitely, as these two copies are considered to be in existence for meeting the at-least-one-copy requirement beyond r_m . In an optimal offline strategy, no copy needs to be stored beyond r_m . Thus, to focus on the cost in a finite time horizon, we do not account for the cost of the aforementioned two copies by Algorithm 1.

The total cost incurred by running Algorithm 1 consists of four portions.

Cost of Regular Copies: By Algorithm 1, a regular copy is created after serving each request (colored darkcyan in Figure 3). Thus, the total number of regular copies ever created is $m = \text{num}(\text{Type-1/2/3/4/5/6})$.² If a regular copy is followed by a **Type-4** request r_i in its local server, the regular copy serves r_i locally, and hence the storage duration of this regular copy is $t_i - t_{p(i)}$. Therefore, the total storage cost of regular copies followed by **Type-4** requests is $\sum_{r_i \text{ is Type-4}} \mu[r_i] \cdot (t_i - t_{p(i)})$. If

²As we consider the regular copy after the dummy request r_0 and ignore the regular copy after the final request r_m , the total number of regular copies is still m .

a regular copy is followed by a **Type-1/2/3/5/6** request r_i , the regular copy expires before r_i arises, and hence its storage cost is λ . Since there are $\text{num}(\text{Type-1/2/3/5/6})$ such regular copies altogether,³ their total storage cost is $\lambda \cdot \text{num}(\text{Type-1/2/3/5/6})$.

In summary, the total storage cost of all the regular copies can be expressed as

$$\lambda \cdot \text{num}(\text{Type-1/2/3/5/6}) + \sum_{r_i \text{ is Type-4}} \mu[r_i] \cdot (t_i - t_{p(i)}). \quad (2)$$

Cost of Transfers: The transfers performed by Algorithm 1 (colored red in Figure 3) include (i) the transfers to serve **Type-1/2/3** requests; and (ii) the transfers to s_1 to create relocated special copies. Therefore, if we use $\text{num}(\text{relocated})$ to denote the total number of relocated special copies ever created, the total cost of transfers can be expressed as

$$\lambda \cdot \text{num}(\text{Type-1/2/3}) + \lambda \cdot \text{num}(\text{relocated}). \quad (3)$$

By Algorithm 1, each relocated special copy is deleted after serving a **Type-2** request. The only exception is that a relocated special copy may *cross* the final request r_m , i.e., a relocated special copy is kept in server s_1 before and after r_m . Thus, if no relocated special copy crosses the final request r_m , we must have $\text{num}(\text{relocated}) = \text{num}(\text{Type-2})$. In this case, r_m cannot be a **Type-5** request, so $\text{num}(\text{Type-5}) = 0$. If a relocated special copy crosses the final request r_m , r_m must be either a **Type-1** request or a **Type-5** request. If r_m is a **Type-5** request (i.e., it arises at s_1 and is served locally by the relocated special copy), we have $\text{num}(\text{Type-5}) = 1$ and hence, $\text{num}(\text{relocated}) = \text{num}(\text{Type-2}) + 1 = \text{num}(\text{Type-2/5})$. If r_m is a **Type-1** request (i.e., it is served by a transfer from the relocated special copy), we have $\text{num}(\text{relocated}) = \text{num}(\text{Type-2}) + 1$. In this case, $\text{num}(\text{Type-5}) = 0$. In summary, we have the following proposition.

Proposition 5. *If a relocated special copy crosses the final request r_m and r_m is a **Type-1** request, we have*

$$\text{num}(\text{relocated}) = \text{num}(\text{Type-2/5}) + 1.$$

Otherwise, we have

$$\text{num}(\text{relocated}) = \text{num}(\text{Type-2/5}).$$

Cost of Resident Special Copies: We use t_{resident} to denote the total storage duration of all resident special copies in the time span $(0, t_m)$. By Algorithm 1, servers holding resident special copies (colored orange in Figure 3) must have storage cost rates no larger than $\min\{\mu_{\max}, d\}$. Therefore, the total storage cost of all resident special copies is at most

$$\min\{\mu_{\max}, d\} \cdot t_{\text{resident}}. \quad (4)$$

Cost of Relocated Special Copies: We use $t_{\text{relocated}}$ to denote the total storage duration of all relocated special copies in the

³If there are n servers ever having requests, there are $n - 1$ regular copies created after the last local request at each of these servers (except $s[r_m]$), which are not followed by any request. Meanwhile, the first local request at each of these servers (except s_1) does not have any preceding regular copy and therefore must be a **Type-1/2/3** request. So, the number of regular copies with cost λ still matches $\text{num}(\text{Type-1/2/3/5/6})$.

time span $(0, t_m)$. As relocated special copies can only be held by server s_1 with storage cost rate $\mu(s_1) = 1$, their total storage cost is $t_{\text{relocated}}$.

By Observation 1, relocated special copies may overlap with regular copies which are stored in servers with storage cost rates higher than d (the overlapped portion is colored purple in Figure 3). Let us consider all the regular copies in these servers that have a storage cost of λ (that is, the regular copy expires before the next local request arises). Among these regular copies, we use X to denote the set of regular copies that do not overlap with relocated special copies, and use Y to denote the set of regular copies that overlap with relocated special copies. In the example of Figure 3, X includes the regular copies after requests r_1 and r_3 , while Y includes the regular copies after requests r_4 , r_6 and r_{11} . With sets X and Y defined, we have the following proposition (proof given in the appendix).

Proposition 6. *It holds that*

$$|X| + |Y| \leq \text{num}(\text{Type-1/3}).$$

*If a relocated special copy crosses the final request r_m and r_m is a **Type-1** request, we further have*

$$|X| + |Y| \leq \text{num}(\text{Type-1/3}) - 1.$$

By Proposition 5 and Proposition 6, it always holds that

$$|X| + |Y| + \text{num}(\text{relocated}) \leq \text{num}(\text{Type-1/2/3/5}). \quad (5)$$

For each regular copy of Y , since its storage duration is $\frac{\lambda}{\mu(s)}$ where $\mu(s) > d$, the duration of its overlap with relocated special copies is bounded by $\frac{\lambda}{d}$. Thus, for all the regular copies of Y , the total duration of their overlaps with relocated special copies is at most $\frac{\lambda}{d} \cdot |Y|$.

Apart from the regular copies of Y , relocated special copies may also overlap with the regular copies that serve **Type-4** requests locally (that is, the regular copy has not expired when the next local request arises). In the example of Figure 3, a relocated special copy overlaps with the regular copy after request r_5 . The duration of such overlaps is bounded by

$\sum_{r_i \text{ is Type-4}} (t_i - t_{p(i)})$. Therefore, the total duration of overlaps between regular copies and relocate special copies is at most $\frac{\lambda}{d} \cdot |Y| + \sum_{r_i \text{ is Type-4}} (t_i - t_{p(i)})$. We use $t'_{\text{relocated}}$ to denote the storage duration (or storage cost) of relocated special copies (colored green in Figure 3) after removing all their overlaps with regular copies. Then, the total storage cost of relocated special copies is bounded by

$$t_{\text{relocated}} \leq t'_{\text{relocated}} + \frac{\lambda}{d} \cdot |Y| + \sum_{r_i \text{ is Type-4}} (t_i - t_{p(i)}). \quad (6)$$

Adding up the four portions of cost in (2)(3)(4)(6), the total online cost can be expressed as

$$\begin{aligned} \text{Online} \leq & 2\lambda \cdot \text{num}(\text{Type-1/2/3}) + \lambda \cdot \text{num}(\text{Type-5/6}) \\ & + \lambda \cdot \text{num}(\text{relocated}) \end{aligned}$$

$$\begin{aligned}
& + \frac{\lambda}{d} \cdot |Y| + \sum_{r_i \text{ is Type-4}} (1 + \mu[r_i]) \cdot (t_i - t_{p(i)}) \\
& + \min\{\mu_{\max}, d\} \cdot t_{\text{resident}} + t'_{\text{relocated}}. \tag{7}
\end{aligned}$$

Proposition 7 (proof given in the appendix).

$$\text{num}(\text{relocated}) \leq |X|. \tag{8}$$

By Proposition 7, we have

$$\begin{aligned}
& \lambda \cdot \text{num}(\text{relocated}) + \frac{\lambda}{d} \cdot |Y| \\
& \leq \frac{1}{2} \lambda \cdot (|X| + \text{num}(\text{relocated})) + \frac{\lambda}{d} \cdot |Y| \\
& \leq \max\left\{\frac{1}{2}, \frac{1}{d}\right\} \cdot \lambda \cdot (|X| + |Y| + \text{num}(\text{relocated})) \\
& \leq \max\left\{\frac{1}{2}, \frac{1}{d}\right\} \cdot \lambda \cdot \text{num}(\text{Type-1/2/3/5}),
\end{aligned}$$

where the third inequality is due to (5).

As a result, the total online cost in (7) can be rewritten as

$$\begin{aligned}
\text{Online} & \leq \max\left\{\frac{5}{2}, 2 + \frac{1}{d}\right\} \cdot \lambda \cdot \text{num}(\text{Type-1/2/3/5/6}) \\
& + \sum_{r_i \text{ is Type-4}} (1 + \mu[r_i]) \cdot (t_i - t_{p(i)}) + d \cdot (t_{\text{resident}} + t'_{\text{relocated}}). \tag{9}
\end{aligned}$$

B. Competitive Analysis of Algorithm 1

We analyze the competitive ratio of Algorithm 1 based on the bounds of the optimal and online costs in (1) and (9). We start with some important observations to link up the optimal cost and the online cost. Recall that all requests are classified into **Type-A/B/C/D/N** in the optimal offline strategy and into **Type-1/2/3/4/5/6** in Algorithm 1.

Proposition 8. $r_i \in A_2$ if and only if r_i is a **Type-4** request.

Proof. By definition, each request $r_i \in A_2$ satisfies $\mu[r_i] \cdot (t_i - t_{p(i)}) \leq \lambda$. By Proposition 3, if $r_i \in A_1 \cup B \cup C \cup D \cup N$, then $\mu[r_i] \cdot (t_i - t_{p(i)}) > \lambda$. Thus, $r_i \in A_2$ if and only if $\mu[r_i] \cdot (t_i - t_{p(i)}) \leq \lambda$. By Proposition 4, $\mu[r_i] \cdot (t_i - t_{p(i)}) \leq \lambda$ if and only if r_i is a **Type-4** request. $\square$

By Proposition 8, we have $|A_2| = \text{num}(\text{Type-4})$ and $|A_1| + |B| + |C| + |D| + |N| = \text{num}(\text{Type-1/2/3/5/6})$. Hence, the optimal cost in (1) can be rewritten as

$$\begin{aligned}
\text{OPT} & \geq \sum_{r_i \text{ is Type-4}} \mu[r_i] \cdot (t_i - t_{p(i)}) + \sum_{r_i \in A_1} \left((t_i - t_{p(i)}) - \frac{\lambda}{\mu[r_i]}\right) \\
& + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}) + \lambda \cdot \text{num}(\text{Type-1/2/3/5/6}). \tag{10}
\end{aligned}$$

Proposition 9 (proof given in the appendix).

$$t_{\text{resident}} + t'_{\text{relocated}} \leq \sum_{r_i \in A_1} \left((t_i - t_{p(i)}) - \frac{\lambda}{\mu[r_i]}\right) + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}).$$

To analyze the competitive ratio of Algorithm 1, consider two cases of the maximal storage cost rate $\mu_{\max}$ separately.

Case 1: If $\mu_{\max} \leq d$, by Algorithm 1, there is no relocated special copy (so $t'_{\text{relocated}} = 0$, $\text{num}(\text{relocated}) = 0$ and $Y =$

$\emptyset$), and therefore no **Type-2/5** requests. Hence, the optimal cost in (10) can be rewritten as

$$\begin{aligned}
\text{OPT} & \geq \sum_{r_i \text{ is Type-4}} \mu[r_i] \cdot (t_i - t_{p(i)}) + \sum_{r_i \in A_1} \left((t_i - t_{p(i)}) - \frac{\lambda}{\mu[r_i]}\right) \\
& + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}) + \lambda \cdot \text{num}(\text{Type-1/3/6}).
\end{aligned}$$

Meanwhile, the online cost in (7) can be rewritten as

$$\begin{aligned}
\text{Online} & \leq 2\lambda \cdot \text{num}(\text{Type-1/3}) + \lambda \cdot \text{num}(\text{Type-6}) \\
& + \sum_{r_i \text{ is Type-4}} (1 + \mu[r_i]) \cdot (t_i - t_{p(i)}) \\
& + \mu_{\max} \cdot \left(\sum_{r_i \in A_1} \left((t_i - t_{p(i)}) - \frac{\lambda}{\mu[r_i]}\right) + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}) \right) \\
& \leq \max\{2, \mu_{\max}\} \cdot \text{OPT},
\end{aligned}$$

where the first inequality is due to Proposition 9.

Case 2: If $\mu_{\max} > d$, it is possible for relocated storage copies to occur. By (9), (10) and Proposition 9, we have

$$\begin{aligned}
\text{Online} & \leq \max\left\{\frac{5}{2}, 2 + \frac{1}{d}\right\} \cdot \lambda \cdot \text{num}(\text{Type-1/2/3/5/6}) \\
& + \sum_{r_i \text{ is Type-4}} (1 + \mu[r_i]) \cdot (t_i - t_{p(i)}) \\
& + d \cdot \left(\sum_{r_i \in A_1} \left((t_i - t_{p(i)}) - \frac{\lambda}{\mu[r_i]}\right) + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}) \right) \\
& \leq \max\left\{\frac{5}{2}, 2 + \frac{1}{d}, d\right\} \cdot \text{OPT}.
\end{aligned}$$

In summary, if $d = \frac{5}{2}$, the competitive ratio of case 1 is $\max\{2, \mu_{\max}\} = \max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$, while that of case 2 is $\frac{5}{2} = \min\{\mu_{\max}, \frac{5}{2}\}$. Hence, the overall competitive ratio can be written as $\max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$, which is actually the lowest achievable across all values of d . In fact, if $d < 2$ or $d > \frac{5}{2}$, the competitive ratio of case 2 is $\max\{2 + \frac{1}{d}, d\} > \frac{5}{2}$. If $2 \leq d < \frac{5}{2}$, the competitive ratio of case 2 is $\frac{5}{2}$, which implies that when $d < \mu_{\max} < \frac{5}{2}$, the competitive ratio is $\frac{5}{2} > \min\{\mu_{\max}, \frac{5}{2}\} = \max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$.

To conclude, Algorithm 1 has a competitive ratio of $\max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$ when setting $d = \frac{5}{2}$. Our competitive analysis is tight. The tightness for the cases of $\mu_{\max} \leq 2$ and $2 < \mu_{\max} \leq \frac{5}{2}$ can be shown by the same examples presented in our earlier work [18], [19]. In what follows, we construct an example to show the tightness for the case of $\mu_{\max} > \frac{5}{2}$. As shown in Figure 5, there are three servers where $\mu(s_1) = 1$, $\mu(s_2) > \frac{5}{2}$ and $\mu(s_3) = 1$ so that $\mu_{\max} > \frac{5}{2}$. Three requests r_1, r_2 and r_3 arise at s_3, s_2 and s_3 respectively at times $t_1 = \lambda$, $t_2 = (2 - \frac{1}{\mu(s_2)}) \cdot \lambda + \epsilon$ and $t_3 = 2\lambda + 2\epsilon$, where $\epsilon > 0$ is a small value. In both Algorithm 1 and the optimal offline strategy, s_1 keeps a copy until r_1 arises and serves it by a transfer. After that, by Algorithm 1, s_3 keeps its regular copy until time 2λ and serves r_2 by a transfer. As the regular copy in s_2 expires later than that of s_3 and $\mu(s_2) > \frac{5}{2}$, s_2 transfers its copy to s_1 to create a relocated special copy at time $2\lambda + \epsilon$. Right after this special copy is created, r_3 arises so that it is served by a transfer from s_1 . Then, the relocated special copy

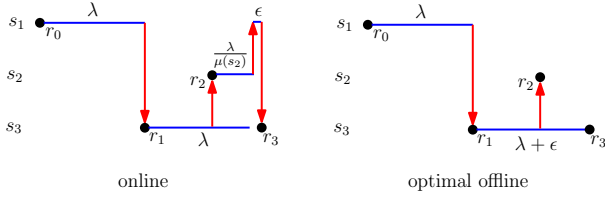


Fig. 5. An example when $\mu_{\max} > \frac{5}{2}$

is deleted. In the optimal offline strategy, s_3 keeps a copy till r_3 and serves r_2 by a transfer. Therefore, the online-to-optimal cost ratio over the period $(t_1, t_3]$ (i.e., excluding the transfer at t_1 and including the transfer at t_3) is $\frac{5\lambda+\epsilon}{2\lambda+\epsilon} \rightarrow \frac{5}{2}$ as $\epsilon \rightarrow 0$. Note that the request pattern during $(t_1, t_3]$ can be repeated, by treating r_3 as r_1 of a new cycle. As we create infinitely many cycles (i.e., let $m \rightarrow \infty$), the online-to-optimal cost ratio of the whole request sequence from time 0 to r_m approaches $\frac{5}{2}$.

V. EXPERIMENTAL EVALUATION

We conduct simulation experiments to evaluate Algorithm 1 using the object access traces of a cloud-based storage service provided by IBM [20]. As similar performance trends are observed for the results of different traces, we present here the representative results of the access trace of a particular object “cdb1824b71efe7d4” in a dataset named “IBM Object Store Trace Number 003”. The access trace of this object consists of 11,683 read requests made over 7 days in the year of 2019. In the experiment, we treat one second as one time unit during this 7-day time period. The requests of the object are uniformly distributed over 10 servers at random. It can be estimated that the average inter-request time of the whole request sequence is about 50 time units, so the inter-request time at each server is about 500 time units on average. Initially, there is only one copy of the object placed at server s_1 .

We compare Algorithm 1 with a state-of-the-art online algorithm by Wang *et al.* [16], [17], which works as follows:

- After a local request at each server s_i is served (either by a transfer or by the local copy), s_i keeps its data copy for $\frac{\lambda}{\mu(s_i)}$ time. We also call the copy over this period a *regular copy*.
- If a new request arises at s_i in this period, the request is served by the local copy. Then s_i keeps the copy for another $\frac{\lambda}{\mu(s_i)}$ time.
- When the data copy in s_1 expires (i.e., the server with the lowest storage cost rate), if it is the only copy in the system, s_1 continues to keep its copy for another $\frac{\lambda}{\mu(s_1)}$ time. Otherwise, s_1 deletes its copy.
- When the data copy in s_i ($s_i \neq s_1$) expires, if it is not the only copy in the system, s_i drops its copy. If s_i holds the only copy, the algorithm checks whether s_i has held the copy for only $\frac{\lambda}{\mu(s_i)}$ time since its most recent local request (i.e., no request arises at s_i for $\frac{\lambda}{\mu(s_i)}$ time). If so, s_i keeps on storing the copy for another $\frac{\lambda}{\mu(s_i)}$ time. Otherwise, it indicates that s_i has kept the copy for $\frac{2\lambda}{\mu(s_i)}$

time without seeing any local request. In this case, s_i transfers the object to s_1 and then drops its local copy.

Wang *et al.* [16], [17] claimed that their algorithm is 2-competitive, but this is not true. A counterexample can be found in our earlier work [18], [19].

We also implement the 3-competitive algorithm discussed at the beginning of Section IV, which always keeps a data copy at server s_1 and maintains a copy for $\frac{\lambda}{\mu(s_j)}$ time (we also call it a *regular copy*) after each local request at any server $s_j \neq s_1$. This algorithm shall be called the *simple algorithm*.

To compare the algorithms, we calculate their total costs of serving the request sequence, under different settings of storage cost rates of the 10 servers:

- **Set 1:** $\{1, 1, 1, 1, 1, 1.1, 1.1, 1.1, 1.1, 1.1\}$ (all rates are close to 1),
- **Set 2:** $\{1, 5, 6, 7, 7, 8, 9, 10, 11, 13\}$ (all rates except that of s_1 are way above $\frac{5}{2}$),
- **Set 3:** $\{1, 1.1, 1.3, 1.6, 1.7, 2.6, 8, 15, 21, 25\}$ (a mix of rates in different ranges),

and different transfer costs $\{1, 25, 50, 75, \dots, 1175, 1200\}$ (which include a wide range of values below and above the average inter-request time at each server). We normalize the calculated online cost with the cost incurred by an optimal offline strategy [17] as a benchmark.

Figures 6-8 show the calculated ratio between the online cost of an algorithm and the optimal cost (referred to as the *online-to-optimal ratio* hereafter) under different settings of storage cost rates and transfer costs. It can be seen that the online-to-optimal ratio of Algorithm 1 is bounded by $\max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$ under all tested settings, which confirms our previous competitive analysis.

One common feature of all the figures is that the online-to-optimal ratios of all three algorithms gradually converge as the transfer cost grows larger. The reason behind this phenomenon is that when the transfer cost λ is large (above 500), the inter-request time at server s_1 tends to be shorter than λ (i.e., the intended duration of a regular copy at s_1). As a result, in both Algorithm 1 and Wang *et al.*’s algorithm, most requests at s_1 are served locally by its regular copies. In other words, the regular copies of s_1 tend to cover the whole time span, emulating the simple algorithm which always keeps a copy at s_1 . Hence, it rarely happens that a server (except s_1) holds the only copy and has to transfer it to s_1 in Algorithm 1 and Wang *et al.*’s algorithm. For the requests at servers other than s_1 , by all three algorithms, they are served locally by regular copies if the inter-request time is short, and served by transfers if the inter-request time is long. Thus, the online costs of all three algorithms are almost equal.

In addition, as the transfer cost λ continues to grow beyond 500, the online-to-optimal ratios of all algorithms gradually drop. The reason is that when the transfer cost λ is very large, the inter-request time at each server s becomes smaller than $\frac{\lambda}{\mu(s)}$ (i.e., the intended duration of a regular copy at s), so most of the requests at s are served by local regular copies in all algorithms. Thus, there is less cost of transfers incurred for serving the requests.

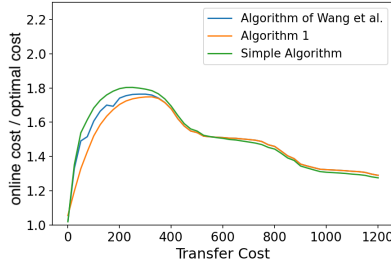


Fig. 6. Results of Set 1

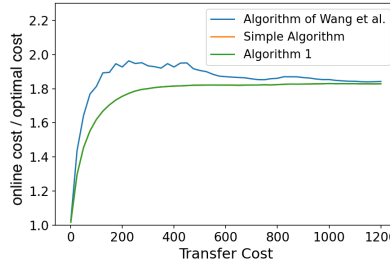


Fig. 7. Results of Set 2

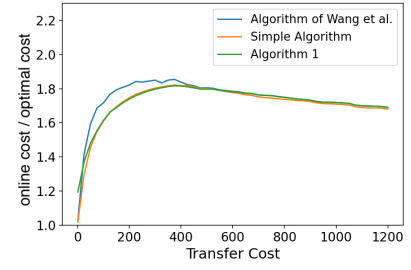


Fig. 8. Results of Set 3

When the transfer cost λ is between 25 and 500, the online-to-optimal ratio of Algorithm 1 is smaller than that of Wang *et al.*'s algorithm. The reason is that in this case, the average inter-request time of the whole sequence is longer than the intended duration of a regular copy at most servers but shorter than twice of that duration (i.e., $\frac{\lambda}{\mu(s)} < 50 < 2 \cdot \frac{\lambda}{\mu(s)}$). Consider the scenario where a server s holds the only copy in the system when its regular copy expires. In Algorithm 1, if $\mu(s) \leq \frac{5}{2}$, s continues to keep its copy until the next request arises, and drops it right after serving the next request. In contrast, in Wang *et al.*'s algorithm, s keeps its copy for another fixed period of $\frac{\lambda}{\mu(s)}$ after its regular copy expires (that is, s does not drop its copy during this new $\frac{\lambda}{\mu(s)}$ period, even after the next request arises and is served). This incurs redundant storage cost because a new regular copy would be created at some other server after the next request, resulting in higher online cost than Algorithm 1. Since the simple algorithm always keeps a copy at s_1 , it also incurs redundant storage cost and thus has higher online cost than Algorithm 1 (see Figure 6 for the results of Set 1). If $\mu(s) > \frac{5}{2}$, in Algorithm 1, s performs a transfer to server s_1 to create a relocated special copy at s_1 . Although it incurs an additional transfer cost of λ , this relocated special copy would still be kept at s_1 after performing transfers to serve subsequent requests arising at any server s' with $\mu(s') > \frac{5}{2}$, which saves transfers from s' to s_1 if s' holds the only copy when its regular copy expires. This mechanism makes Algorithm 1 gain an advantage over Wang *et al.*'s algorithm (which needs to perform transfers from s' to s_1 to migrate the copy to s_1 again), especially when the majority of servers have storage cost rates higher than $\frac{5}{2}$ (see Figures 7-8 for the results of Sets 2-3). On the other hand, the simple algorithm works almost the same as Algorithm 1 when all servers except s_1 have storage cost rates higher than $\frac{5}{2}$. Thus, their online costs are almost equal (see Figures 7-8).

VI. CONCLUSION

In this paper, we have developed an online algorithm for optimizing the total storage and network cost of distributed data access, and shown that it has a competitive ratio of $\max\{2, \min\{\mu_{\max}, \frac{5}{2}\}\}$, where $\mu_{\max}$ is the max/min ratio of storage cost rates among all servers. The competitiveness of our algorithm is quite close to the lower bound of 2 on the competitive ratio of any deterministic algorithm [17]. Direc-

tions of future research include studying whether it is possible to further reduce the competitive ratio when $\mu_{\max} > 2$.

ACKNOWLEDGMENT

This research is supported by the Ministry of Education, Singapore, under its AcRF Tier 2 (Award MOE-T2EP20122-0007) and Tier 1 (Award RG23/23).

REFERENCES

- [1] A. Q. Khan, M. Matskin, R. Prodan, C. Bussler, D. Roman, and A. Soyul, "Cloud storage cost: a taxonomy and survey," *World Wide Web*, vol. 27, no. 4, p. 36, 2024.
- [2] J. Ju, J. Wu, J. Fu, Z. Lin, and J. Zhang, "A survey on cloud storage," *Journal of Computers*, vol. 6, no. 8, pp. 1764–1771, 2011.
- [3] M. Liu, L. Pan, and S. Liu, "Cost optimization for cloud storage from user perspectives: Recent advances, taxonomy, and survey," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–37, 2023.
- [4] D. Boukhlef, J. Boukhobza, K. Boukhalfa, H. Ouarnoughi, and L. Lemarchand, "Optimizing the cost of DBaaS object placement in hybrid storage systems," *Future Generation Computer Systems*, vol. 93, pp. 176–187, 2019.
- [5] Y. Mansouri and A. Erradi, "Cost optimization algorithms for hot and cool tiers cloud storage services," in *Proceedings of the 11th IEEE International Conference on Cloud Computing*, 2018, pp. 622–629.
- [6] B. Veeravalli, "Network caching strategies for a shared data distribution for a predefined service demand sequence," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1487–1497, 2003.
- [7] N. Le Scouarnec, C. Neumann, and G. Straub, "Cache policies for cloud-based systems: To keep or not to keep," in *Proceedings of the 7th IEEE International Conference on Cloud Computing*, 2014, pp. 1–8.
- [8] Y. Mansouri and R. Buyya, "To move or not to move: Cost optimization in a dual cloud-based storage architecture," *Journal of Network and Computer Applications*, vol. 75, pp. 223–235, 2016.
- [9] K. P. Puttaswamy, T. Nandagopal, and M. Kodialam, "Frugal storage for cloud file systems," in *Proceedings of the 7th ACM European Conference on Computer Systems*, 2012, pp. 71–84.
- [10] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press Cambridge, 1998, vol. 53.
- [11] V. V. Vazirani, *Approximation Algorithms*. Springer Science & Business Media, 2013.
- [12] R. Bar-Yehuda, E. Kantor, S. Kuten, and D. Rawitz, "Growing half-balls: Minimizing storage and communication costs in CDNs," in *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming*, 2012, pp. 416–427.
- [13] Y. Wang, S. He, X. Fan, C. Xu, and X.-H. Sun, "On cost-driven collaborative data caching: A new model approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 662–676, 2018.
- [14] T. Zuo, X. Tang, and B. S. Lee, "A randomized caching algorithm for distributed data access," in *Proceedings of the 43rd IEEE Conference on Computer Communications (INFOCOM)*, 2024, pp. 1–10.
- [15] T. Zuo, X. Tang, and B. S. Lee, "Cost-driven data replication with predictions," in *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, 2024, pp. 309–320.
- [16] Y. Wang, H. Dai, X. Han, P. Wang, Y. Zhang, and C. Xu, "Cost-driven data caching in edge-based content delivery networks," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1384–1400, 2023.

- [17] Y. Wang, Y. Zhang, X. Han, P. Wang, C. Xu, J. Horton, and J. Culberson, "Cost-driven data caching in the cloud: An algorithmic approach," in *Proceedings of the 40th IEEE Conference on Computer Communications (INFOCOM)*, 2021, pp. 1–10.
- [18] T. Zuo, X. Tang, B. S. Lee, and J. Cai, "On competitiveness of dynamic replication for distributed data access," in *Proceedings of the 27th Int'l Conf. on Distributed Computing and Networking*, 2026, pp. 11–20.
- [19] T. Zuo, X. Tang, B. S. Lee, and J. Cai, "On competitiveness of dynamic replication for distributed data access," *Theoretical Computer Science*, vol. 1077, p. 115992, 2026.
- [20] O. Eytan, D. Harnik, E. Ofer, R. Friedman, and R. Kat, "IBM object store," 2021, <http://iotta.snia.org/traces/key-value/36305>.

APPENDIX

Proposition 6 *It holds that*

$$|X| + |Y| \leq \text{num}(\mathbf{Type-1/3}). \quad (11)$$

*If a relocated special copy crosses the final request r_m and r_m is a **Type-1** request, we further have*

$$|X| + |Y| \leq \text{num}(\mathbf{Type-1/3}) - 1. \quad (12)$$

Proof. For each regular copy of $X \cup Y$ which is stored in a server s , we would like to find a distinct **Type-1/3** request at s that corresponds to it, in order to prove (11). If the regular copy is followed by a local request r at s , since $\mu(s) > d$, by Algorithm 1 (lines 27-36), the regular copy must be either dropped or transferred to s_1 before r arises. Thus, r cannot be a **Type-4/6** request. It also follows from $\mu(s) > d$ that r cannot be a **Type-2/5** request. Therefore, r must be a **Type-1/3** request. If the regular copy is not followed by any local request (i.e., it is the last regular copy of s), since $\mu(s) > d$ so that $s \neq s_1$, the first local request of s must be served by a transfer and it has no preceding regular copy. So, this first local request must be a **Type-1/3** request. Hence, we have found a distinct **Type-1/3** request for each regular copy in $X \cup Y$, so (11) is proven.

If a relocated special copy crosses the final request r_m and r_m is a **Type-1** request, we can find an additional **Type-1/3** request to prove (12). In this case, since the relocated special copy is not deleted after the final request r_m , it implies that $\mu[r_m] > d$ and hence $s[r_m] \neq s_1$. Thus, the first local request at $s[r_m]$ must be served by a transfer and it has no preceding regular copy. Therefore, it must be a **Type-1/3** request. Since we do not account for the regular copy created after r_m , no regular copy of $X \cup Y$ was mapped to this request before. As a result, we have found $|X| + |Y| + 1$ different **Type-1/3** requests in total, so (12) is proven. $\square$

Proposition 7

$$\text{num}(\text{relocated}) \leq |X|. \quad (13)$$

Proof. For each relocated special copy, we would like to find a distinct regular copy in X that corresponds to it, in order to prove (13). Recall that each relocated special copy is created by a transfer from a source server s with $\mu(s) > d$ to server s_1 , after the expiration of a regular copy that follows a request r_i at s at time $t_i + \frac{\lambda}{\mu(s)}$ (see Figure 9 for an illustration). We would like to show that this regular copy after r_i is in X , i.e., it does not overlap with any relocated special copy.

Assume on the contrary that the regular copy after r_i overlaps with a relocated special copy (call it *special copy A* thereafter). Since another relocated regular copy is created at time $t_i + \frac{\lambda}{\mu(s)}$ (call it *special copy B* thereafter), special copy A must be deleted before time $t_i + \frac{\lambda}{\mu(s)}$, as by Observation 1 two relocated special copies cannot overlap.

By Algorithm 1, special copy A can be deleted only due to an outward transfer for serving another request r_j at some server s' with $\mu(s') \leq d$ (as shown in Figure 9). Since $t_i < t_j$ and $\mu(s') \leq d < \mu(s)$, the regular copy after r_j must expire later than the regular copy after r_i , i.e. expiring later than time $t_i + \frac{\lambda}{\mu(s)}$. This leads to an overlap between the regular copy after r_i and special copy B, which is impossible by Observation 1.

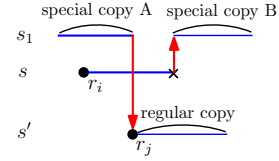


Fig. 9. Illustration of an impossible case

For each relocated special copy, we have found a distinct regular copy in X , so (13) is proven. $\square$

Proposition 9

$$t_{\text{resident}} + t'_{\text{relocated}} \leq \sum_{r_i \in A_1} ((t_i - t_{p(i)}) - \frac{\lambda}{\mu[r_i]}) + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)}). \quad (14)$$

Proof. Recall from Section III-B that the storage durations associated with **Type-A/B/C/D** requests cover all data copies in the optimal offline strategy in the time span $(0, t_m)$. So, the storage periods of resident special copies and relocated special copies are entirely contained in the storage durations associated with **Type-A/B/C/D** requests. By Observation 1, the resident and relocated special copies do not overlap. Thus, we have $t_{\text{resident}} + t'_{\text{relocated}} \leq \sum_{r_i \in A} (t_i - t_{p(i)}) + \sum_{r_i \in B \cup C \cup D} (t_i - t_{s(i)})$.

Moreover, recall that $t'_{\text{relocated}}$ is defined as the storage duration of relocated special copies after removing their overlaps with all regular copies, and by Observation 1, resident special copies do not overlap with any regular copies. Thus, the special copies of $t_{\text{resident}} + t'_{\text{relocated}}$ cannot overlap with any regular copies. By definition, each request $r_i \in A_2$ has $\mu[r_i] \cdot (t_i - t_{p(i)}) \leq \lambda$, so it is served locally by the regular copy stored over the period $(t_{p(i)}, t_i)$ in Algorithm 1. Therefore, the special copies of $t_{\text{resident}} + t'_{\text{relocated}}$ cannot appear during $(t_{p(i)}, t_i)$ for any $r_i \in A_2$. Moreover, for each request $r_i \in A_1$, by Algorithm 1, there is a regular copy over the period $(t_{p(i)}, t_{p(i)} + \frac{\lambda}{\mu[r_i]})$, so the special copies of $t_{\text{resident}} + t'_{\text{relocated}}$ cannot appear during this period.

Excluding the aforesaid storage periods of regular copies after requests of A_1 and A_2 , we have the upper bound of $t_{\text{resident}} + t'_{\text{relocated}}$ as given in (14). $\square$