

Cost-Driven Data Replication with Predictions

Tianyu Zuo, Xueyan Tang, and Bu Sung Lee
Nanyang Technological University
Singapore

zuot0001@e.ntu.edu.sg, {asxytang, ebslee}@ntu.edu.sg

Abstract

This paper studies an online replication problem for distributed data access. The goal is to dynamically create and delete data copies in a multi-server system as time passes to minimize the total storage and network cost of serving access requests. We study the problem in the emergent learning-augmented setting, assuming simple binary predictions about inter-request times at individual servers. We develop an online algorithm and prove that it is $(\frac{5+\alpha}{3})$ -consistent (competitiveness under perfect predictions) and $(1 + \frac{1}{\alpha})$ -robust (competitiveness under terrible predictions), where $\alpha \in (0, 1]$ is a hyper-parameter representing the level of distrust in the predictions. We also study the impact of mispredictions on the competitive ratio of the proposed algorithm and adapt it to achieve a bounded robustness while retaining its consistency. We further establish a lower bound of $\frac{3}{2}$ on the consistency of any deterministic learning-augmented algorithm. Experimental evaluations are carried out to evaluate our algorithms using real data access traces.

1 Introduction

Online decision-making problems are an important field of algorithmic research. These problems seek to make a chain of judicious decisions as time moves on, without the knowledge of what will happen in the future. The data replication (or caching) problem for distributed data access at various locations is one such example with many application scenarios, such as cloud storage services, content distribution networks, edge and fog computing. The goal of this problem is to store data copies in a distributed system, so that the requests for data access can be satisfied with minimal service costs. The service costs normally include the storage and network costs, because storing data copies in servers and transferring data among servers both consume resources and incur expenses for service providers [10, 13]. In this paper, we study dynamic replication in a distributed system to minimize the total storage and network cost for serving a sequence of data access requests, where the storage cost is modelled proportional to the time duration of storage and the network cost is modelled proportional to the amount of network traffic. Unaware of the access times and locations of future requests, an online algorithm of managing data copies is needed to minimize the cumulative service cost over the time span.

One paradigm of tackling uncertainty in future inputs is competitive analysis, which refers to the idea of guaranteeing a bound on the performance of an online algorithm over all instances of the problem. To this end, a metric called *competitive ratio* is used to measure the ratio between the costs produced by an online algorithm (without knowledge of future) and the optimal offline solution (with full knowledge of future) in the worst case [2, 14]. Some researches have been carried out to study the data replication problem with competitive analysis. Mansouri and Erradi [9] developed online algorithms for a two-tier storage system which consists of a hot tier with high storage cost but low access cost, and a cold tier

with low storage cost but high access cost. Mansouri *et al.* [11] developed online algorithms for data replication and migration in cloud storage services with competitive ratios dependent on the max/min cost ratio among data centers. Veeravalli [15] presented a general model for migrating and caching shared data in a network of servers. Assuming the storage costs of all servers are identical, Bar-Yehuda *et al.* [1] developed an $O(\log \delta)$ -competitive online algorithm where δ is the normalized diameter of the underlying network. Under the same assumption of identical storage costs of all servers, Wang *et al.* [16] proposed an optimal offline solution by dynamic programming and a 3-competitive online algorithm in the case of uniform transfer costs among servers. Recently, they also extended the solutions to servers with distinct storage costs and presented an online algorithm claimed to be 2-competitive [17] (the claim however is not true, as shall be explained in Section 11).

Traditional competitive analysis, however, is often too pessimistic in that it assumes no knowledge about future inputs at all. With the rapidly growing power of machine learning technologies, the emergence of predictive models in recent years provides another possible way of beating uncertainty in online decision-making problems. By forecasting the future based on historical data, predictive models offer additional information for decision-making. We can use the predictions about future inputs to design more effective online algorithms both in theory and practice. Nevertheless, predictive models are usually imperfect and prone to errors. Unconditional reliance on the forecasts of predictive models may on the contrary deteriorate online algorithms [12]. Incorrect or inaccurate predictions may mislead online algorithms to make wrong decisions and result in unbounded competitive ratios. Hence, machine-learned predictions should be wisely incorporated into the algorithm design, so that both prediction use and worst-case guarantee can be achieved.

Some recent studies have explored augmenting online algorithms with predictions to address the classical ski-rental problem [6]. Kumar *et al.* [8] proposed a deterministic algorithm and a randomized algorithm with the use of predictions. By introducing a hyper-parameter to manipulate the reliance on the predictions, their algorithms achieve a tight trade-off between the best case of perfect predictions and the worst case of terrible predictions [18]. Gollapudi and Panigrahi [3] further considered a more general case where multiple predictions are provided by a collection of machine learning predictors. Based on the distribution of the predictions, their algorithm carefully determines when to rent or buy skis. Kodialam [7] studied another scenario of using machine-learned predictions, where the predictions are in the probabilistic form rather than the deterministic form. He developed a randomized algorithm with the aid of such soft machine-learned predictions.

Our work is inspired by the above studies. As shall be elaborated later, due to the trade-off between storage and network costs, making a replication decision at a single server in our context of distributed data access resembles the ski-rental problem. But our problem is much more complicated than ski-rental in that it involves multiple servers and multiple requests with inter-dependencies. To the best of our knowledge, there has been no study on making sensible use of predictions for cost-driven data replication in a distributed system. We aim to develop a learning-augmented online algorithm to optimize the cumulative cost of serving a request sequence.

Our contributions are summarized as follows.

1. We propose an online replication algorithm which uses the forecasts provided by predictors, in order to minimize the total cost of serving a request sequence.
2. We conduct theoretical analysis to show that the proposed algorithm is $(\frac{5+\alpha}{3})$ -consistent (competitive ratio under perfect predictions), and $(1 + \frac{1}{\alpha})$ -robust (competitive ratio under terrible predictions), where $\alpha \in (0, 1]$ is a hyper-parameter representing the level of distrust in the predictions. We also study the impact of mispredictions on the competitive ratio and adapt our algorithm to

achieve a bounded robustness while retaining its consistency.

3. We establish a lower bound of $\frac{3}{2}$ on the consistency of any deterministic learning-augmented algorithm. This implies that it is not possible to achieve a consistency approaching 1 in our problem.
4. We empirically evaluate our algorithm using real data access traces, and show that it can make effective use of predictions to improve performance with increasing prediction accuracy.

2 Problem Definition

We consider a system including n geo-distributed servers (or sites) s_1, s_2, \dots, s_n . A data object is hosted in the system and copies of this object can be created and stored in any servers.¹ Storing a data copy in a server incurs a cost of 1 per time unit. The data object can also be transferred between servers when needed. The transfer cost of the object between any two servers is λ ($\lambda > 0$).

Requests to access the data arise at the servers over time (e.g., due to the computational jobs running at the servers or user requests). When a request arises at a server s_j , if s_j holds a data copy, the request is served locally. Otherwise, the object has to be transferred from a server holding a copy to s_j in order to serve the request. s_j may, as a result of the transfer, create a data copy and hold it for some period of time. We denote the sequence of requests arising at the servers as $\langle r_1, r_2, r_3, \dots \rangle$. For each request r_i , let $s[r_i]$ denote the server at which r_i arises and t_i denote the time when r_i arises. For simplicity, we assume that all requests arise at distinct time instants, i.e., $t_1 < t_2 < t_3 < \dots$. In addition, we assume only one data copy placed in server s_1 initially. To facilitate algorithm design and analysis, we add a dummy request r_0 arising at server s_1 at time 0. Note that r_0 does not incur any additional cost for serving the request sequence.

We seek to develop a *replication strategy* that determines the data copies to create and hold as well as the transfers to carry out in an *online* manner to serve all requests. There must be at least one data copy stored at all times to preserve the data. Our objective is to minimize the total storage and transfer cost of serving a request sequence. We focus on the problem in the learning-augmented setting and assume there are predictions of inter-request times at individual servers (e.g., based on the request history or other features). Specifically, we assume that after a request arises at a server, a simple binary prediction is available about whether the next request at the same server will arise within or beyond a period of λ time units from the current request. Since there are n servers, we have n predictions at any time, each for the next request at one server. We would like to make sensible use of the predictions to reduce the cost for serving requests.

For an online algorithm with predictions, the competitive ratio is often expressed as a function of prediction errors. As two special cases, when all predictions are perfect (zero error), the competitive ratio is known as *consistency*, whereas when the predictions can be arbitrarily bad (unbounded errors), the competitive ratio is known as *robustness* [8].

3 Algorithm Design

We start with investigating the problem structure. Observe that there is a trade-off between holding and not holding a data copy at each individual server. If no copy is held by a server, we have to pay the

¹We do not consider any capacity limit on creating data copies in servers, since storage is usually of large and sufficient capacity nowadays. Hence, we focus on the management of one data object, as different objects can be handled separately.

transfer cost for serving each local request. If a copy is held by a server, we have to pay the storage cost which is proportional to the duration of storage. Such a trade-off appears to resemble that between rent and buy in the classical ski-rental problem [6]. Without predictions of inter-request times, to achieve decent competitiveness, an intuitive idea is to let a server s_j hold a copy for a period of λ time units after serving every local request. If the next request arises at s_j before this period ends, the request will be served by the local copy and we pay the storage cost which is optimal. After that, s_j will renew the copy and hold it for another period of λ long. If no request arises at s_j during the period of λ , s_j will delete the copy and an incoming transfer will then be required to serve the next request at s_j . In this case, the total cost paid is $2 \cdot \lambda$, while the optimal cost is a transfer cost λ (by not holding a copy in s_j). On the other hand, with perfect predictions of inter-request times, we can always achieve the optimal cost. That is, if the inter-request time between two consecutive requests at a server s_j is no more than λ , we keep a data copy in s_j between the requests and pay the storage cost which is optimal. Otherwise, we do not keep a copy in s_j and pay the transfer cost for serving the latter request which is optimal. Nevertheless, this solution does not have bounded robustness, because (1) if the inter-request time is predicted to be longer than λ but is actually shorter than λ , the transfer cost to pay is fixed while the optimal cost can be close to 0; and (2) if the inter-request time is predicted to be shorter than λ but is actually longer than λ , the storage cost can go towards infinity in the worst case. Furthermore, if we simply apply the above ideas (with or without predictions) to every server, it may not meet the requirement of maintaining at least one data copy at any time, because all copies will be deleted after a sufficiently long silent period without any request.

Our approach to algorithm design is to integrate and balance between following predictions and not using predictions. We introduce a hyper-parameter $\alpha \in (0, 1]$ to indicate the level of distrust in the predictions. $\alpha \rightarrow 0$ indicates nearly full trust and reliance on the predictions, while $\alpha \rightarrow 1$ indicates almost no trust and not using the predictions. If the next request at a server is predicted to arise beyond a period of λ from the current request, we let the server hold a copy for a period of $\alpha \cdot \lambda$ instead of deleting the copy immediately. Then, even if the prediction is not correct, the next request still has some chance to be served by the local copy, avoiding the transfer. This will save the cost when the next request actually arises quite soon and thus improve robustness while keeping the loss in consistency under control. On the other hand, if the next request at a server is predicted to arise within a period of λ from the current request, we let the server hold a copy for a period of λ rather than up to the next request. Then, even if the prediction is not correct, the storage cost will not increase infinitely, thereby enhancing robustness while still ensuring consistency. In addition, to meet the at-least-one-copy requirement, the copy whose period ends the latest according to the above strategies will continue to be kept beyond its period.

Algorithm 1 shows the details of our proposed algorithm. We use E_j to denote the intended expiration time of the data copy in server s_j , use K_j to denote a binary tag indicating whether server s_j keeps a copy beyond the intended expiration time, and use c to denote the number of servers holding data copies. Initially, there is only one data copy in server s_1 , so $c = 1$ (line 2).

When a request arises at a server s_j , if s_j holds a data copy, the request is served locally (lines 4-5). Otherwise, the request is served by a transfer from another server with a copy (lines 6-9). After serving the request, s_j keeps the data copy for an *intended duration* based on the prediction of the following inter-request time. If the prediction forecasts the next request at s_j to arise within a period of λ , s_j keeps the copy for an intended duration of λ . Otherwise, s_j keeps the copy for an intended duration of $\alpha \cdot \lambda$ (lines 10-13). In either case, the tag K_j of s_j is cleared (line 14).²

In the intended duration, if a new request arises at s_j , the request is served by the local copy and

²With the dummy request r_0 arising at server s_1 at time 0, s_1 sets an intended duration λ or $\alpha \cdot \lambda$ for the initial copy according to the prediction of the first request arrival (line 2).

Algorithm 1 Dynamic Replication with Predictions

```
1: /* a data copy is initially stored in server  $s_1$  */
2: initialize:  $c \leftarrow 1$ ;  $E_1 \leftarrow \lambda$  or  $\alpha \cdot \lambda$ ;  $E_j \leftarrow -\infty$  for all  $2 \leq j \leq n$ ;  $K_j \leftarrow 0$  for all  $1 \leq j \leq n$ ;  $\triangleright s_1$ 
   holds a data copy
3: upon (a request  $r_i$  arises at server  $s_j$  at time  $t_i$ ) do
4:   if  $t_i \leq E_j$  or  $K_j = 1$  then  $\triangleright s_j$  holds a data copy
5:     serve  $r_i$  by the local copy in  $s_j$ ;
6:   else
7:     serve  $r_i$  by a transfer from any other server with a copy;
8:     create a copy in  $s_j$ ;
9:      $c \leftarrow c + 1$ ;
10:  if the prediction forecasts the next request at  $s_j$  will arise no later than time  $t_i + \lambda$  then
11:     $E_j \leftarrow t_i + \lambda$ ;
12:  else
13:     $E_j \leftarrow t_i + \alpha \cdot \lambda$ ;
14:   $K_j \leftarrow 0$ ;
15: upon ( $s_j$  transfers the data object to another server  $s_k$  at time  $t$ ) do
16:   if  $K_j = 1$  then  $\triangleright s_j$  holds the only copy
17:     drop the copy in  $s_j$ ;
18:      $K_j \leftarrow 0$ ;
19:    $c \leftarrow c - 1$ ;
20: upon (a copy expires in server  $s_j$  at time  $E_j$ ) do
21:   if  $c = 1$  then  $\triangleright s_j$  holds the only copy
22:      $K_j \leftarrow 1$ ;
23:   else
24:     drop the copy in  $s_j$ ;
25:      $c \leftarrow c - 1$ ;
```

s_j renews the copy for another intended duration based on the new prediction of the subsequent inter-request time. When the intended duration of the copy in s_j expires (i.e., no request arises at s_j), if s_j does not hold the only copy in the system, it drops the copy (lines 23-25). Otherwise, if s_j holds the only copy, it continues to keep the copy by setting its tag K_j (lines 21-22) until the next request in the system. If the next request arises at s_j , s_j sets a new intended duration of the copy according to the prediction and clears its tag (lines 10-14). If the next request arises at another server s_k ($s_k \neq s_j$), a transfer has to be performed to s_k to serve the request. In this case, s_j drops its copy right after the transfer and clears its tag (lines 15-19). Meanwhile, a copy will be created at s_k upon receiving the transfer (lines 8-9), so the requirement of maintaining at least one copy is met.

To facilitate algorithm analysis, we refer to a data copy held within the intended duration derived from the prediction as a *regular copy*, and a data copy held beyond the intended duration as a *special copy*. Figure 1 shows an example produced by our algorithm, where horizontal edges represent data copies and vertical edges represent transfers. By the algorithm definition, since a *special copy* is the only copy in the system, it is easy to infer the following property.

Proposition 1. *The storage periods of any two special copies do not overlap. Moreover, the storage period of any special copy does not overlap with that of any regular copy.*

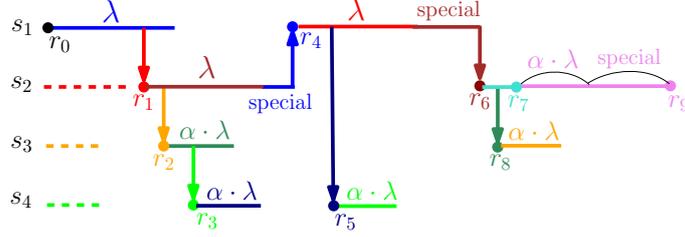


Figure 1: An example of our online algorithm

4 Preliminaries for Analysis

Our general approach to competitive analysis is to first divide a request sequence into partitions based on the characteristics of an optimal offline strategy, and then study the costs of Algorithm 1 and the optimal strategy for each partition separately. To this end, we need to allocate the cost produced by Algorithm 1 (referred to as the *online cost* for short) to individual requests and characterize an optimal offline strategy to prepare for the analysis.

4.1 Allocation of Online Cost

For each request r_i , we define $r_{p(i)}$ as the preceding request of r_i arising at the same server,³ and l_i as the intended duration of the regular copy in server $s[r_i]$ after $r_{p(i)}$. By the algorithm definition, l_i equals either $\alpha \cdot \lambda$ or λ . As illustrated in Figure 2, we categorize all the requests in the sequence into four types based on how they are served by our online algorithm.

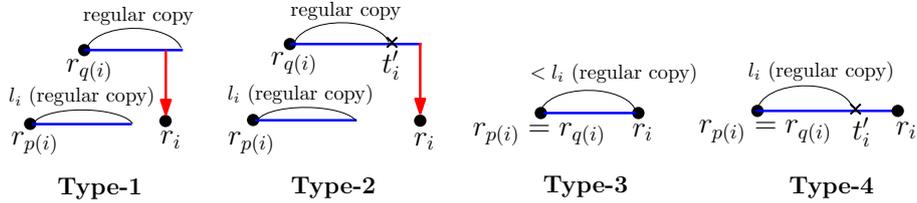


Figure 2: Different request types in our online algorithm

If a request r_i is served by a transfer from another server, by the algorithm definition, that server must keep a data copy since its most recent request before r_i . We denote this most recent request by $r_{q(i)}$ and its server by $s[r_{q(i)}]$. At the time of r_i , if the copy held by $s[r_{q(i)}]$ is a regular copy, r_i is called a **Type-1** request. If the copy held by $s[r_{q(i)}]$ is a special copy, r_i is called a **Type-2** request. For each **Type-2** request r_i , we define t'_i as the time instant when the copy in $s[r_{q(i)}]$ switches from a regular copy to a special copy.

If a request r_i is served by a local copy, by the algorithm definition, the copy must be kept in $s[r_i]$ since the preceding request $r_{p(i)}$ at $s[r_i]$. If the copy is a regular copy at r_i 's arrival, r_i is called a **Type-3** request. If the copy is a special copy at r_i 's arrival, r_i is called a **Type-4** request.

For notational convenience, for each **Type-3/4** request r_i , we define $r_{q(i)} = r_{p(i)}$. Then, $r_{q(i)}$ uniformly denotes the earlier request that provides the data copy to serve r_i for all request types. For each

³For example, in Figure 1, $p(6) = 1$ since r_1 and r_6 arise in succession at server s_2 .

Type-4 request r_i , we also define t'_i as the time instant when the copy in $s[r_i]$ switches from a regular copy to a special copy.

For each **Type-1/2** request r_i , the cost allocated to r_i includes: (1) the transfer cost of λ for serving r_i ; (2) the storage cost of the special copy in server $s[r_{q(i)}]$ over the period (t'_i, t_i) if r_i is a **Type-2** request; (3) the storage cost of the regular copy in server $s[r_i]$ after $r_{p(i)}$, which is given by l_i .⁴

For each **Type-3/4** request r_i , we allocate the storage cost of the copy in server $s[r_i]$ during the period $(t_{p(i)}, t_i)$ to r_i , i.e., r_i is allocated a cost of $t_i - t_{p(i)}$.

The cost allocation can be summarized as follows.

Proposition 2. *The online cost allocated to a request r_i , based on its type, is given by*

- **Type-1:** $l_i + \lambda$;
- **Type-2:** $(t_i - t'_i) + l_i + \lambda$;
- **Type-3:** $t_i - t_{p(i)}$;
- **Type-4:** $t_i - t_{p(i)} = (t_i - t'_i) + l_i$.

In the example of Figure 1, each request and its allocated cost are shown in the same color. r_5 and r_8 are **Type-1** requests, r_4 and r_6 are **Type-2** requests, r_7 is a **Type-3** request, and r_9 is a **Type-4** request.

There are some special considerations to note in the cost allocation. Given a request sequence $\langle r_1, r_2, \dots, r_m \rangle$, 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 in the system after r_m , the copy expiring the latest would switch to a special copy and stay infinitely. In our analysis, we shall not account for the cost of the regular copy created in server $s[r_m]$ after r_m and the special copy to stay infinitely. The rationale is that the storage periods of these two copies do not overlap and they are both beyond r_m . These two copies are considered to be in existence for ensuring that there is at least one copy in the system beyond r_m . In an optimal offline strategy, no copy will need to be stored beyond r_m . To focus on the cost in a finite time horizon, we do not account for the cost of the aforesaid two copies by the online algorithm.⁵

By the above assumption, if there are n servers ever holding copies in serving a request sequence, there will be $n - 1$ regular copies after the last request at each server (except the server where r_m arises), whose costs have not been allocated. Note that the first request at each server (except s_1 with the initial copy) must be served by a transfer because no copy was stored in the server. Since there are $n - 1$ such first requests in total, we allocate the storage costs of the aforementioned $n - 1$ regular copies to these $n - 1$ first requests (one regular copy for each request). In this way, the first request r_j of each server (except s_1) is allocated a total cost of $l + \lambda$ if it is served by a transfer from a regular copy (**Type-1** request), or a total cost of $(t_j - t'_j) + l + \lambda$ if it is served by a transfer from a special copy (**Type-2** request), where l is the intended duration of the regular copy after the last request at some server). This is then consistent with the cost allocations of other **Type-1/2** requests as given in Proposition 2.

In the example of Figure 1, r_1 , r_2 and r_3 are the first requests arising at servers s_2 , s_3 and s_4 respectively, and they are all **Type-1** requests. We do not account for the regular copy after the final request r_9 which arises at s_2 . There are three regular copies after the last requests at the other servers

⁴Note that it is possible for the copy in server $s[r_i]$ to switch from a regular copy to a special copy (i.e., extend beyond the l_i period). In this case, the storage cost of the special copy is allocated to a request at another server served by a transfer from $s[r_i]$ according to the aforementioned (2).

⁵If we insist in considering these two copies, their storage costs can be bounded by the storage cost of a data copy in any server beyond r_m in the optimal offline strategy. Hence, it would not affect the correctness of our competitive analysis.

(i.e., after r_4 , r_5 and r_8). Their storage cost of $\lambda + 2 \cdot \lambda \cdot \alpha$ is allocated to r_1 , r_2 and r_3 , with the cost of one regular copy for each of them (dashed lines).

It is easy to verify that the sum of the costs allocated to all requests is equal to the total online cost and the dummy request r_0 is not allocated any cost.

4.2 Characteristics of An Optimal Strategy

Deriving the exact form of an optimal offline strategy for a request sequence is not straightforward. In this section, we present some characteristics of an optimal replication strategy to facilitate our competitive analysis. The proofs of the following propositions are given in the appendix.

Proposition 3. *There exists an optimal replication strategy in which for each transfer, there is a request at either the source server or the destination server of the transfer.*

The main idea to prove the above proposition is that if there is no request at the source and destination servers, we can always advance or delay the transfer to save or maintain the total cost.

Proposition 4. *There exists an optimal replication strategy with the characteristic in Proposition 3 and that for each request r_i , if r_i is served by a local copy, the copy is created no later than $r_{p(i)}$.*

The following feature says that if two successive requests at the same server are sufficiently close in time, the server should hold a copy between them.

Proposition 5. *There exists an optimal replication strategy with the characteristics in Propositions 3, 4 and that for each request r_i , if $t_i - t_{p(i)} \leq \lambda$, server $s[r_i]$ holds a copy throughout the period $(t_{p(i)}, t_i)$, so that r_i is served by a local copy.*

If a data copy is consistently stored in a server before and after a time instant t , we say that this copy crosses time t .

Proposition 6. *There exists an optimal replication strategy with the characteristics in Propositions 3, 4, 5 and that for each request r_i , if r_i is served by a transfer and no server holds a copy crossing the time t_i of r_i , then (i) r_{i-1} and r_i arise at different servers; and (ii) the source server of the transfer is $s[r_{i-1}]$ which keeps a copy since t_{i-1} .*

Hereafter, an optimal offline strategy shall always refer to one with the characteristics stated in Propositions 3, 4, 5 and 6.

5 Division Approach for Analysis

We adopt a division approach to analyze the robustness and consistency of our algorithm. Consider an optimal offline strategy of a request sequence $\langle r_1, r_2, \dots, r_m \rangle$. Remember that each request r_i arises at server $s[r_i]$. For each request r_i , we examine if any server other than $s[r_i]$ holds a copy crossing the time t_i of r_i . We identify all such requests r_i 's where no other server holds a copy crossing t_i and divide the request sequence into partitions at these requests. Note that the dummy request r_0 and the final request r_m naturally meet the criterion.

Let $\langle r_d, r_{d+1}, \dots, r_e \rangle$ denote a partition of requests, where $0 \leq d < e \leq m$. That is, no copy at any other server crosses t_d , no copy at any other server crosses t_e , and for each request r_i where $d < i < e$, there exists a copy at some other server crossing t_i . We use **Online**(d, e) to denote the total online cost

allocated to the requests $r_{d+1}, r_{d+2}, \dots, r_e$ by the allocation method in Section 4.1, and use $\mathbf{OPT}(d, e)$ to denote the total cost of storage and transfers incurred over the period $(t_d, t_e]$ in the optimal offline strategy. In the analysis of robustness (resp. consistency), we shall bound the ratio $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)}$ by $1 + \frac{1}{\alpha}$ (resp. $\frac{5+\alpha}{3}$) for each partition. Then, aggregating over all the partitions, the ratio between the online cost and the optimal offline cost of the entire request sequence is bounded by the same constant.

To analyze $\mathbf{OPT}(d, e)$, in the optimal offline strategy, we shall pick a set of storage periods of data copies to cover the time span from t_d to t_e .

Case A: Request r_e is served by a transfer. By Proposition 6, in this case, r_{i-1} and r_i arise at different servers, and server $s[r_{e-1}]$ holds a data copy during the period (t_{e-1}, t_e) and performs a transfer to serve r_e at time t_e . It can further be proved that no server other than $s[r_{e-1}]$ can hold a copy crossing t_{e-1} . Assume on the contrary that there exists a server $s \neq s[r_{e-1}]$ that keeps a copy crossing t_{e-1} . This copy must be deleted at some time instant $t \leq t_e$, since there is no copy crossing t_e . Hence, this copy is not used for serving any local request at s after t_{e-1} , since there is no request during (t_{e-1}, t_e) . Thus, the copy in s can be removed during (t_{e-1}, t) . As a result, the total cost is reduced, contradicting the optimality of the offline strategy. Therefore, no copy at any other server crosses t_{e-1} . It follows from the partition definition that $d = e - 1$. So, we just pick the storage period (t_{e-1}, t_e) of the copy at $s[r_{e-1}]$ to cover the time span of the partition (see Figure 4).

Case B: Request r_e is served by a local copy. In this case, we pick a set of data copies iteratively in reverse order of time. By Proposition 4, the copy serving r_e must be created no later than the time $t_{p(e)}$ of the preceding request $r_{p(e)}$. First, we pick the storage period $(t_{p(e)}, t_e)$ of the copy at $s[r_e]$ and move backward to time $t_{p(e)}$. By the partition definition, we must have $p(e) \geq d$ (otherwise, the copy crosses t_d). If $p(e) = d$, we stop. If $p(e) > d$, by definition, there exists a copy at some other server $s \neq s[r_{p(e)}]$ crossing $t_{p(e)}$. It can be proved that the copy at s must be kept till at least the first local request r_g at s after $t_{p(e)}$ and hence will serve that request. If it does not serve any local request at s (see Figure 3 for an illustration), the copy can be deleted earlier at $t_{p(e)}$, because all transfers originating from the copy at s after $t_{p(e)}$ can originate from the copy at $s[r_e]$ instead. This reduces the total cost and contradicts the optimality of the offline strategy.

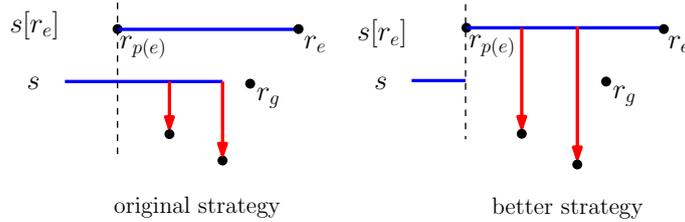


Figure 3: Illustration of a data copy at s crossing time $t_{p(e)}$

Since the copy at s is kept to serve the first request r_g at s after $t_{p(e)}$, by Proposition 4, the copy at s must be created no later than the last request $r_{p(g)}$ at s before $t_{p(e)}$. So, we pick the storage period $(t_{p(g)}, t_g)$ of the copy and move backward to time $t_{p(g)}$. If $p(g) = d$, we stop. If $p(g) > d$, again there exists a copy at some other server $s' \neq s[r_{p(g)}]$ crossing $t_{p(g)}$. Similarly, the copy must be kept till at least the first request r_h at s' after $t_{p(g)}$ and created no later than the last request $r_{p(h)}$ at s' before $t_{p(g)}$. Then, we pick the storage period $(t_{p(h)}, t_h)$ of the copy and move backward to time $t_{p(h)}$. This backtracking process is repeated until time t_d is reached. Eventually, as illustrated in Figure 4, we collect a set of storage periods of copies covering the time span of the partition. Each storage period has the form of $(t_{p(i)}, t_i)$, i.e., between two consecutive requests at the same server. We shall

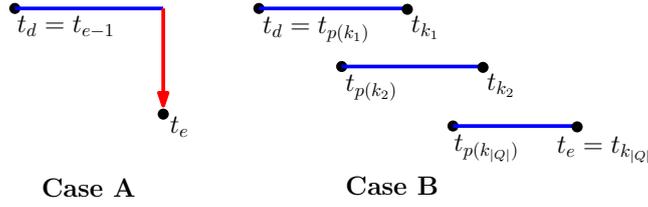


Figure 4: Illustration of **Case A** and **Case B**

denote the set of storage periods picked as $Q = \{(t_{p(k_1)}, t_{k_1}), (t_{p(k_2)}, t_{k_2}), \dots, (t_{p(k_{|Q|})}, t_{k_{|Q|}})\}$, where $d = p(k_1) < k_1 < k_2 < \dots < k_{|Q|} = e$, each $(t_{p(k_j)}, t_{k_j})$ crosses the start time $t_{p(k_{j+1})}$ of the next period, and $|Q|$ is the number of storage periods picked.

6 Robustness Analysis

In this section, we show that our proposed algorithm is $(1 + \frac{1}{\alpha})$ -robust, i.e., the competitive ratio of the algorithm is always bounded by $1 + \frac{1}{\alpha}$, irrespective of the prediction correctness. We analyze partitions of cases A and B separately.

Case A: In this case, $d = e - 1$, i.e., the partition has the form $\langle r_{e-1}, r_e \rangle$. So, $\mathbf{Online}(d, e)$ refers to the online cost allocated to request r_e , and $\mathbf{OPT}(d, e)$ refers to the cost incurred over the period $(t_{e-1}, t_e]$ in the optimal offline strategy. By the analysis in Section 5, in the optimal offline strategy, server $s[r_{e-1}]$ holds the only copy during (t_{e-1}, t_e) and performs an outgoing transfer to serve r_e . Hence, $\mathbf{OPT}(d, e) = (t_e - t_{e-1}) + \lambda$.

According to Proposition 2, if r_e is a **Type-1** request by our online algorithm, the online cost allocated to r_e is $l_e + \lambda$ (where l_e is the duration of the regular copy after $r_{p(e)}$). Recall that l_e equals either $\alpha \cdot \lambda$ or λ . Since $l_e \leq \lambda$, we have $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} \leq \frac{\lambda + \lambda}{\lambda} = 2 \leq 1 + \frac{1}{\alpha}$.

If r_e is a **Type-2/4** request, the online cost allocated to r_e is at most $(t_e - t'_e) + l_e + \lambda$. By Proposition 1, there is no overlap between the storage period of the regular copy after r_{e-1} and the storage period (t'_e, t_e) of the special copy for serving r_e . This implies that (t'_e, t_e) is completely contained in (t_{e-1}, t_e) , so $t_e - t'_e < t_e - t_{e-1}$. Thus, $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} < \frac{(t_e - t_{e-1}) + l_e + \lambda}{(t_e - t_{e-1}) + \lambda} < 1 + \frac{\lambda}{\lambda} = 2 \leq 1 + \frac{1}{\alpha}$.

If r_e is a **Type-3** request, the online cost allocated to r_e is $t_e - t_{p(e)}$. By definition, $t_e - t_{p(e)} \leq l_e \leq \lambda$. Hence, $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} \leq \frac{\lambda}{\lambda} = 1 < 1 + \frac{1}{\alpha}$.

Case B: In this case, we have a set of storage periods Q covering the time span of the partition. For each storage period $(t_{p(k_j)}, t_{k_j}) \in Q$, we refer to the request r_{k_j} at time t_{k_j} as the *end sentinel request*. To calculate the optimal offline cost, we divide all requests $r_{d+1}, r_{d+2}, \dots, r_e$ into: $R_Q = \{r_{k_1}, r_{k_2}, \dots, r_{k_{|Q|}}\}$ (the end sentinel requests), R_L (r_i 's that are not end sentinel requests and $t_i - t_{p(i)} \leq \lambda$), and R_T (r_i 's that are not end sentinel requests and $t_i - t_{p(i)} > \lambda$). By Proposition 5, each request $r_i \in R_L$ is served by a local copy over the period $(t_{p(i)}, t_i)$. Since the storage periods Q cover the time span, to minimize cost, each request $r_i \in R_T$ must be served by a transfer from a copy of some storage period in Q . Thus, the optimal offline cost can be written as

$$\mathbf{OPT}(d, e) = \sum_{r_i \in R_Q} (t_i - t_{p(i)}) + \sum_{r_i \in R_L} (t_i - t_{p(i)}) + \sum_{r_i \in R_T} \lambda. \quad (1)$$

On the other hand, by our online algorithm, we divide all requests $r_{d+1}, r_{d+2}, \dots, r_e$ into: R_1 (**Type-1** requests), R_2 (**Type-2** requests), R_3 (**Type-3** requests), and R_4 (**Type-4** requests). By Proposition 2,

the total online cost allocated to all requests can be written as

$$\mathbf{Online}(d, e) = \sum_{r_i \in R_1 \cup R_2} \lambda + \sum_{r_i \in R_1 \cup R_2 \cup R_4} l_i + \sum_{r_i \in R_2 \cup R_4} (t_i - t'_i) + \sum_{r_i \in R_3} (t_i - t_{p(i)}). \quad (2)$$

By the algorithm definition, special copies cannot cross the time of any request, because each special copy is deleted after serving the next request. Recall that $Q = \{(t_{p(k_1)}, t_{k_1}), (t_{p(k_2)}, t_{k_2}), \dots, (t_{p(k_{|Q|}}), t_{k_{|Q|}})\}$, where $p(k_1) = d$, $k_{|Q|} = e$, and each $(t_{p(k_j)}, t_{k_j})$ crosses time $t_{p(k_{j+1})}$. Thus, we can divide the time span of the partition into the following disjoint intervals: $(t_{p(k_1)}, t_{p(k_2)})$, $(t_{p(k_2)}, t_{p(k_3)})$, \dots , $(t_{p(k_{|Q|-1}}), t_{p(k_{|Q|})})$ and $(t_{p(k_{|Q|}}), t_{k_{|Q|}})$. The storage period (t'_i, t_i) of the special copy for serving each request $r_i \in R_2 \cup R_4$ must be completely contained in one of these intervals, because there are requests at both endpoints of these intervals. For those intervals containing special copies, we select their original periods in Q to form a subset $X \subset Q$, i.e., for each $j < |Q|$, $(t_{p(k_j)}, t_{k_j}) \in X$ if there exist special copies during $(t_{p(k_j)}, t_{p(k_{j+1})})$; and $(t_{p(k_{|Q|}}), t_{k_{|Q|}}) \in X$ if there exist special copies during $(t_{p(k_{|Q|}}), t_{k_{|Q|}})$. Let R_X denote the end sentinel requests of all storage periods in X . Each period $(t_{p(i)}, t_i) \in X$ covers all special copies therein and l_i . By Proposition 1, all the special copies do not overlap. Thus, we have

$$\sum_{(t_{p(i)}, t_i) \in X} (t_i - t_{p(i)}) = \sum_{r_i \in R_X} (t_i - t_{p(i)}) \geq \sum_{r_i \in R_2 \cup R_4} (t_i - t'_i) + \sum_{r_i \in R_X} l_i \quad (3)$$

In addition, by our online algorithm, a special copy occurs only when all the regular copies expire. Thus, for each end sentinel request $r_i \in R_X$, the regular copy after the preceding request $r_{p(i)}$ must expire before r_i 's arrival. This implies that r_i cannot be a **Type-3** request.

Proposition 7. *All requests in R_X are **Type-1/2/4** requests, i.e., $R_X \subset R_1 \cup R_2 \cup R_4$.*

Hence, applying (3) to (2), we can bound the total online cost by

$$\mathbf{Online}(d, e) \leq \sum_{r_i \in R_1 \cup R_2} \lambda + \sum_{r_i \in R_1 \cup R_2 \cup R_4 \setminus R_X} l_i + \sum_{r_i \in R_3 \cup R_X} (t_i - t_{p(i)}). \quad (4)$$

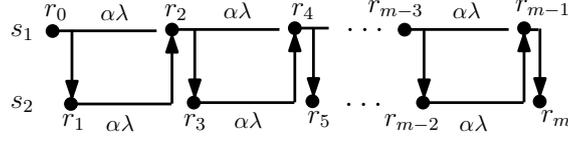
We show $(1 + \frac{1}{\alpha})$ -robustness by comparing the corresponding terms in (1) and (4) for individual requests. Observe that for each request $r_i \notin R_3$, we have $t_i - t_{p(i)} \geq l_i \geq \alpha \cdot \lambda$ since the regular copy after $r_{p(i)}$ must expire before r_i 's arrival and a regular copy is at least $\alpha \cdot \lambda$ long. On the other hand, for each request $r_i \in R_3$, we have $t_i - t_{p(i)} \leq l_i \leq \lambda$ since a regular copy is at most λ long.

By definition, $R_X \subset R_Q$. For each request $r_i \in R_X$, since $r_i \notin R_3$ by Proposition 7, we have $t_i - t_{p(i)} \geq \alpha \cdot \lambda$. The online cost for r_i in (4) is at most $\lambda + t_i - t_{p(i)}$, while the offline cost for r_i in (1) is $t_i - t_{p(i)}$. Hence, the online-to-offline ratio for r_i is bounded by $\frac{\lambda + t_i - t_{p(i)}}{t_i - t_{p(i)}} \leq 1 + \frac{\lambda}{t_i - t_{p(i)}} \leq 1 + \frac{1}{\alpha}$.

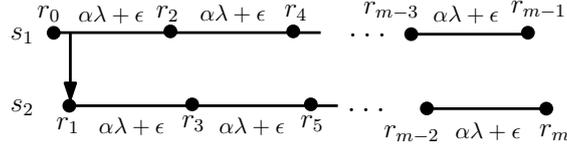
For each request $r_i \in R_Q \setminus R_X$ or $r_i \in R_L$, the offline cost for r_i in (1) is $t_i - t_{p(i)}$. If $r_i \in R_3$, the online cost for r_i in (4) is the same. If $r_i \notin R_3$, the online cost for r_i in (4) is at most $\lambda + l_i$. Since $t_i - t_{p(i)} \geq l_i \geq \alpha \cdot \lambda$, the online-to-offline ratio for r_i is bounded by $\frac{\lambda + l_i}{t_i - t_{p(i)}} \leq \frac{\lambda + l_i}{l_i} \leq 1 + \frac{1}{\alpha}$.

For each request $r_i \in R_T$, the offline cost for r_i in (1) is λ . If $r_i \in R_3$, the online cost for r_i in (4) is $t_i - t_{p(i)} \leq \lambda$. Thus, the online-to-offline ratio for r_i is bounded by 1. If $r_i \notin R_3$, the online cost for r_i in (4) is at most $\lambda + l_i$. Since $l_i \leq \lambda$, the online-to-offline ratio for r_i is bounded by $\frac{\lambda + l_i}{\lambda} \leq 2 \leq 1 + \frac{1}{\alpha}$.

Since the online-to-offline ratio for each individual request in the partition is bounded by $1 + \frac{1}{\alpha}$, it can be concluded that $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} \leq 1 + \frac{1}{\alpha}$.



Algorithm 1



optimal offline strategy

Figure 5: A tight example for robustness analysis

Figure 5 gives an example to show that the robustness analysis above is tight. A total of $m + 1$ requests arise at two servers s_1 and s_2 . The dummy request r_0 arises at s_1 at time $t_0 = 0$. The first request r_1 arises at s_2 at time $t_1 = \epsilon$. The inter-request time between two consecutive requests at s_1 or s_2 is $\alpha\lambda + \epsilon$, where $\epsilon > 0$ is a small value. Suppose that the inter-request times at s_1 and s_2 are always predicted to be greater than λ . Then, the intended duration of the regular copy after each request is $\alpha\lambda$. By our online algorithm, r_1 is served by a transfer. r_2 arises right after the expiration of the regular copy in s_1 , so it is served by a transfer from s_2 . After the outgoing transfer, the regular copy in s_2 expires, and then r_3 arises at s_2 shortly afterwards. Hence, r_3 is served by a transfer from s_1 . This pattern is repeated continuously, so that all the requests r_2, r_3, \dots, r_m are served by transfers, as they all arise after the expiration of the preceding regular copies. Thus, the total online cost is $(m - 1) \cdot (\alpha\lambda + \lambda) + \lambda$. In the optimal offline strategy, after r_1 , both s_1 and s_2 keep copies to serve all their local requests. So, the optimal cost is $(m - 1) \cdot (\alpha\lambda + \epsilon) + \lambda$. Therefore, the online-to-optimal cost ratio is $\frac{(m-1) \cdot (\alpha\lambda + \lambda) + \lambda}{(m-1) \cdot (\alpha\lambda + \epsilon) + \lambda}$, which approaches $1 + \frac{1}{\alpha}$ as $m \rightarrow \infty$ and $\epsilon \rightarrow 0$.

7 Consistency Analysis

In this section, we show that our proposed algorithm is $\frac{5+\alpha}{3}$ -consistent, i.e., the competitive ratio of the algorithm is $\frac{5+\alpha}{3}$ when all predictions are correct. We start with an important observation for correct predictions.

Proposition 8. *Given that the prediction of the inter-request time between $r_{p(i)}$ and r_i is correct, if r_i is a **Type-1/2/4** request, then $t_i - t_{p(i)} > \lambda$ and $l_i = \alpha \cdot \lambda$; if r_i is a **Type-3** request, then $t_i - t_{p(i)} \leq \lambda$ and $l_i = \lambda$.*

Proof. If r_i is a **Type-1/2/4** request, by definition, the inter-request time between $r_{p(i)}$ and r_i (i.e., $t_i - t_{p(i)}$) must be longer than the duration l_i of the regular copy after $r_{p(i)}$. Since the prediction is correct, if $t_i - t_{p(i)} \leq \lambda$, then $l_i = \lambda \geq t_i - t_{p(i)}$, leading to a contradiction. Thus, $t_i - t_{p(i)} > \lambda$ and hence, $l_i = \alpha \cdot \lambda$. If r_i is a **Type-3** request, by definition, the inter-request time $t_i - t_{p(i)}$ must be no longer than l_i . Since the prediction is correct, if $t_i - t_{p(i)} > \lambda$, then $l_i = \alpha \cdot \lambda \leq \lambda < t_i - t_{p(i)}$, leading to a contradiction. Thus, $t_i - t_{p(i)} \leq \lambda$ and hence, $l_i = \lambda$. \square

Again, we analyze partitions of cases A and B (as presented in Section 5) separately.

Case A: In this case, the partition has the form of $\langle r_{e-1}, r_e \rangle$, i.e., $d = e - 1$. Same as in the robustness analysis, $\mathbf{OPT}(d, e) = (t_e - t_{e-1}) + \lambda$, and $\mathbf{Online}(d, e)$ is the online cost allocated to r_e . Since r_e is served by a transfer in the optimal offline strategy, by Proposition 5, $t_e - t_{p(e)} > \lambda$. It follows from Proposition 8 that r_e is a **Type-1/2/4** request by our online algorithm and $l_e = \alpha\lambda$.

Based on Proposition 2, if r_e is a **Type-1** request, the online cost allocated to r_e is $l_e + \lambda = \alpha\lambda + \lambda$. Thus, $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} = \frac{\alpha\lambda + \lambda}{(t_e - t_{e-1}) + \lambda} < 1 + \alpha \leq \frac{5 + \alpha}{3}$.

If r_e is a **Type-2/4** request, the online cost allocated to r_e is at most $(t_e - t'_e) + l_e + \lambda = (t_e - t'_e) + \alpha\lambda + \lambda$. By Proposition 1, there is no overlap between the storage period of the regular copy after r_{e-1} and the storage period (t'_e, t_e) of the special copy for serving r_e . Hence, (t'_e, t_e) is fully contained in (t_{e-1}, t_e) so that $t_e - t'_e < t_e - t_{e-1}$. Thus, $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} < \frac{(t_e - t_{e-1}) + \alpha\lambda + \lambda}{(t_e - t_{e-1}) + \lambda} < 1 + \alpha \leq \frac{5 + \alpha}{3}$.

Case B: In this case, we have picked a set of storage periods Q covering the time span of the partition. Recall that $R_1/R_2/R_3/R_4$ denotes all **Type-1/2/3/4** requests among $r_{d+1}, r_{d+2}, \dots, r_e$ by our online algorithm. By Proposition 8, for each request $r_i \in R_1 \cup R_2 \cup R_4$, $l_i = \alpha\lambda$. Hence, the online cost (2) can be rewritten as

$$\mathbf{Online}(d, e) = (|R_1| + |R_2|) \cdot (1 + \alpha)\lambda + |R_4| \cdot \alpha\lambda + \sum_{r_i \in R_2 \cup R_4} (t_i - t'_i) + \sum_{r_i \in R_3} (t_i - t_{p(i)}). \quad (5)$$

Recall that X is the set of storage periods in Q that contain special copies. Using the definition of X , we can bound the total cost of special copies in (5). The proofs of the lemmas hereafter are given in the appendix.

Lemma 1. *When all predictions are correct, we have*

$$\sum_{r_i \in R_2 \cup R_4} (t_i - t'_i) \leq \sum_{r_i \in R_X} (t_i - t_{p(i)}) - (|R_2| + |R_4|) \cdot \alpha\lambda.$$

By (5) and Lemma 1, we have

$$\mathbf{Online}(d, e) \leq |R_1| \cdot (1 + \alpha)\lambda + |R_2| \cdot \lambda + \sum_{r_i \in R_3 \cup R_X} (t_i - t_{p(i)}). \quad (6)$$

Recall that in the optimal offline strategy, we divide all requests $r_{d+1}, r_{d+2}, \dots, r_e$ into: R_Q (the end sentinel requests), R_L (r_i 's that are not end sentinel requests and $t_i - t_{p(i)} \leq \lambda$), and R_T (r_i 's that are not end sentinel requests and $t_i - t_{p(i)} > \lambda$). According to the request types by our online algorithm, we further divide the end sentinel requests R_Q into: R'_1 (**Type-1** requests), R'_2 (**Type-2** requests), R'_3 (**Type-3** requests), and R'_4 (**Type-4** requests). Obviously, it holds that

$$|Q| = |R_Q| = |R'_1| + |R'_2| + |R'_3| + |R'_4|. \quad (7)$$

Moreover, it follows from $R_X \subset R_Q$ and Proposition 7 that $R_X \subset R'_1 \cup R'_2 \cup R'_4$. In addition, by Proposition 8, all requests in R_L are **Type-3** requests and all requests in R_T are **Type-1/2/4** requests. Thus, $R_L = R_3 \setminus R'_3$ and $R_T = (R_1 \cup R_2 \cup R_4) \setminus (R'_1 \cup R'_2 \cup R'_4)$. Hence, the optimal offline cost (1) can be rewritten as

$$\mathbf{OPT}(d, e) = \sum_{r_i \in R'_3 \cup R_X} (t_i - t_{p(i)}) + \sum_{r_i \in R'_1 \cup R'_2 \cup R'_4 \setminus R_X} (t_i - t_{p(i)})$$

$$\begin{aligned}
& + \sum_{r_i \in R_3 \setminus R'_3} (t_i - t_{p(i)}) + \sum_{r_i \in (R_1 \cup R_2 \cup R_4) \setminus (R'_1 \cup R'_2 \cup R'_4)} \lambda \\
& \geq \sum_{r_i \in R_3 \cup R_X} (t_i - t_{p(i)}) + \sum_{r_i \in R'_1 \cup R'_2 \cup R'_4 \setminus R_X} \lambda \\
& + \sum_{r_i \in (R_1 \cup R_2 \cup R_4) \setminus (R'_1 \cup R'_2 \cup R'_4)} \lambda \quad (\text{by Proposition 8}) \\
& = (|R_1| + |R_2| + |R_4| - |X|) \cdot \lambda + \sum_{r_i \in R_3 \cup R_X} (t_i - t_{p(i)}). \tag{8}
\end{aligned}$$

Combining (6) and (8), we have

$$\begin{aligned}
\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} & \leq 1 + \frac{|R_1| \cdot \alpha \lambda - |R_4| \cdot \lambda + |X| \cdot \lambda}{(|R_1| + |R_2| + |R_4| - |X|) \cdot \lambda + \sum_{r_i \in R_3 \cup R_X} (t_i - t_{p(i)})} \\
& \leq 1 + \frac{|R_1| \cdot \alpha \lambda - |R_4| \cdot \lambda + |X| \cdot \lambda}{(|R_1| + |R_2| + |R_4|) \cdot \lambda} \quad (\text{by Propositions 7, 8}) \\
& = 1 + \alpha + \frac{|X| - (|R_2| + |R_4|) \cdot \alpha - |R_4|}{|R_1| + |R_2| + |R_4|}. \tag{9}
\end{aligned}$$

By definition, for each $(t_{p(k_j)}, t_{k_j}) \in X$, if $j < |Q|$, the period $(t_{p(k_j)}, t_{p(k_{j+1})})$ contains at least one special copy and hence one **Type-2/4** request; if $j = |Q|$, the period $(t_{p(k_{|Q|})}, t_{k_{|Q|}})$ contains at least one special copy and hence one **Type-2/4** request. Since these periods are disjoint, all these **Type-2/4** requests are distinct. Thus, $|X| \leq |R_2| + |R_4|$. As a result,

$$\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} \leq 1 + \alpha + \frac{|X| \cdot (1 - \alpha) - |R_4|}{|R_1| + |R_2| + |R_4|}. \tag{10}$$

In order to bound the above ratio, we discuss the cases of $|Q| \geq 2$ and $|Q| = 1$ separately.

Case 1: $|Q| \geq 2$. In this case, we have the following result about the total number of **Type-1/2/4** requests in the partition.

Lemma 2. *When $|Q| \geq 2$ and $|X| \geq 1$, it holds that $|R_1| + |R_2| + |R_4| \geq \max\{2|X| - 1, 2\}$.*

By (10) and Lemma 2, we have:

- When $|X| \geq 2$, $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} \leq 1 + \alpha + \frac{|X| \cdot (1 - \alpha)}{2|X| - 1} \leq 1 + \alpha + \frac{2 \cdot (1 - \alpha)}{3} = \frac{5 + \alpha}{3}$.
- When $|X| = 1$, $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} \leq 1 + \alpha + \frac{(1 - \alpha)}{2} \leq \frac{5 + \alpha}{3}$.
- When $|X| = 0$, $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} = 1 + \alpha \leq \frac{5 + \alpha}{3}$.

Case 2: $|Q| = 1$. In this case, there is only one storage period $(t_{p(k_1)}, t_{k_1}) \in Q$ and $|X| \leq |Q| = 1$.

If the end sentinel request r_{k_1} is a **Type-3** request by our online algorithm, the copy during $(t_{p(k_1)}, t_{k_1})$ is a regular copy. By Proposition 1, there is no special copy during $(t_{p(k_1)}, t_{k_1})$ and hence $|X| = 0$. It follows from (10) that $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} \leq 1 + \alpha \leq \frac{5 + \alpha}{3}$. If r_{k_1} is a **Type-4** request, we have $|R_4| \geq 1$ and thus $\frac{\mathbf{Online}(d, e)}{\mathbf{OPT}(d, e)} \leq 1 + \alpha + \frac{(1 - \alpha) - 1}{1} = 1 < \frac{5 + \alpha}{3}$. If r_{k_1} is a **Type-1/2** request, we have the following lemma.

Lemma 3. *When $|Q| = 1$, if the only end sentinel request r_{k_1} is a **Type-1/2** request, there are at least two **Type-1/2/4** requests in the partition, i.e., $|R_1| + |R_2| + |R_4| \geq 2$.*

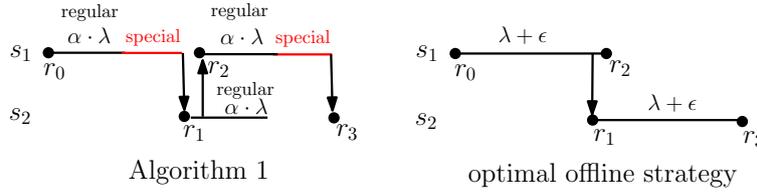


Figure 6: A tight example for consistency analysis

By (10) and Lemma 3, we have $\frac{\mathbf{Online}(d,e)}{\mathbf{OPT}(d,e)} \leq 1 + \alpha + \frac{1-\alpha}{2} \leq \frac{5+\alpha}{3}$ if r_{k_1} is a **Type-1/2** request.

In conclusion, for both cases of $|Q| \geq 2$ and $|Q| = 1$, the ratio $\frac{\mathbf{Online}(d,e)}{\mathbf{OPT}(d,e)}$ is bounded by $\frac{5+\alpha}{3}$.

Figure 6 gives an example to show that the consistency analysis above is tight. Request r_2 arises at server s_1 , and requests r_1 and r_3 arise at server s_2 . The arising times of the requests are $t_1 = \lambda$, $t_2 = \lambda + \epsilon$ and $t_3 = 2\lambda + \epsilon$, where $\epsilon > 0$ is a small value. In the optimal offline strategy, both r_2 and r_3 are served locally, and r_1 is served by a transfer from s_1 . So, the optimal cost is $3\lambda + 2\epsilon$. By our online algorithm, since the inter-request time at each server is longer than λ , the duration of each regular copy is $\alpha\lambda$ if all predictions are correct. When the regular copy after the dummy request r_0 expires, it switches to a special copy. Upon an outgoing transfer to serve r_1 , the special copy is deleted. Then, r_2 has to be served by a transfer from the regular copy after r_1 . Upon expiration, the regular copy after r_2 switches to a special copy since it is the only copy in the system. This special copy is later used to serve r_3 by a transfer. The total online cost is $5\lambda + \alpha\lambda$. Hence, the online-to-optimal cost ratio is $\frac{5\lambda + \alpha\lambda}{3\lambda + 2\epsilon}$, which approaches $\frac{5+\alpha}{3}$ as $\epsilon \rightarrow 0$. We note that this example can also be extended to a request sequence of arbitrary length by repeating r_1, r_2, r_3 (i.e., treating r_3 as r_0 of a new cycle with the roles of servers s_1 and s_2 swapped).

8 Impact of Mispredictions

In this section, we study the impact of mispredictions on the competitive ratio of the proposed algorithm. For each request r_i , if the inter-request time between $r_{p(i)}$ and r_i is mispredicted, we call r_i a *mispredicted request*. Otherwise, we call r_i a *correct request*. For each mispredicted request r_i , we investigate how r_i may be affected and how correct requests may be affected. Recall from Proposition 2 that the online cost allocated to a request consists of three possible parts: (1) the cost of a regular copy; (2) the cost of a special copy (if any); (3) the cost of a transfer (if any). We study the impact on these three parts.

We divide all the mispredicted requests into three sets M_1 , M_2 and M_3 based on the real inter-request time between the mispredicted request and its preceding request.

For each $r_i \in M_1$, the inter-request time satisfies $t_i - t_{p(i)} \leq \alpha\lambda$. In the case of correct prediction, the duration l_i of the regular copy after $r_{p(i)}$ is set to λ , so r_i is a **Type-3** request since $t_i - t_{p(i)} \leq \lambda$. In the case of misprediction, l_i reduces to $\alpha\lambda$, and r_i remains a **Type-3** request. Therefore, misprediction of the requests in M_1 does not affect the online cost.

For each $r_i \in M_2$, the inter-request time satisfies $\alpha\lambda < t_i - t_{p(i)} \leq \lambda$. In the case of correct prediction, l_i is set to λ , so r_i is a **Type-3** request since $t_i - t_{p(i)} \leq \lambda$. In the case of misprediction, l_i reduces to $\alpha\lambda$, and r_i becomes a **Type-1/2/4** request.

- By Proposition 2, the transfer cost allocated to r_i can increase by at most λ . The regular copy after $r_{p(i)}$ is shortened from $t_i - t_{p(i)}$ to $\alpha\lambda$, so the regular copy cost allocated to r_i is reduced by

$t_i - t_{p(i)} - \alpha\lambda$. Meanwhile, the cutback of the regular copy after $r_{p(i)}$ can create special copies at server $s[r_i]$ and other servers. By Proposition 1, the total duration of all special copies created must be bounded by the duration cut of the regular copy after $r_{p(i)}$, i.e., $t_i - t_{p(i)} - \alpha\lambda$. The costs of the special copies created may be allocated to r_i or correct requests, which would be offset by the aforesaid reduction in the regular copy cost allocated to r_i .

- For correct requests, a **Type-1** correct request may change to **Type-2/4**. In this case, the transfer cost allocated to the request can never increase. A **Type-2** (resp. **Type-4**) correct request remains **Type-2** (resp. **Type-4**), since the special copy for serving the request is not affected by the cutback of the regular copy after $r_{p(i)}$. A **Type-3** correct request remains **Type-3**. In these cases, the transfer cost allocated to the request does not change. In addition, the regular copy costs allocated to all correct requests do not change.
- In summary, the total increase in the online cost due to each $r_i \in M_2$ is bounded by λ .

For each $r_i \in M_3$, the inter-request time $t_i - t_{p(i)} > \lambda$. In the case of correct prediction, l_i is set to $\alpha\lambda$, so r_i is a **Type-1/2/4** request since $t_i - t_{p(i)} > l_i$. In the case of misprediction, l_i increases to λ , and r_i remains a **Type-1/2/4** request. However, the specific type of r_i may change.

- If r_i is **Type-4** under correct prediction, it implies that there is no request at other servers between $r_{p(i)}$ and r_i , because $l_i = \alpha\lambda$ is the shortest intended duration of a regular copy. As a result, r_i remains **Type-4** under misprediction. Thus, the possible type changes of r_i are from **Type-1/2** to **Type-1/2/4** only. By Proposition 2, the transfer cost allocated to r_i can never increase. The regular copy cost allocated to r_i increases by $\lambda - \alpha\lambda$. The extension of the regular copy after $r_{p(i)}$ at $s[r_i]$ can only shorten the special copy at $s[r_i]$ or any other server. Hence, the special copy cost allocated to r_i can never increase. Overall, the online cost allocated to r_i can increase by at most $\lambda - \alpha\lambda$.
- For correct requests, a **Type-1** (resp. **Type-3**) correct request remains **Type-1** (resp. **Type-3**). Thus, its allocated online cost does not change. A **Type-2** correct request may change to **Type-1**, where its allocated online cost can only decrease. A **Type-4** correct request may change to **Type-1/2**, where its allocated online cost can increase by at most the cost λ of a transfer. Note that if a request r_j is a **Type-4** correct request, we must have $t_j - t_{p(j)} > \lambda$ (otherwise, the regular copy after $r_{p(j)}$ would be λ long and r_j must be a **Type-3** request) and no request arises elsewhere between $r_{p(j)}$ and r_j . Thus, at most one **Type-4** request can change to **Type-1/2**.
- In summary, the total increase in the online cost due to each $r_i \in M_3$ is bounded by $(2 - \alpha) \cdot \lambda$.

By the above analysis, the total increase in the online cost due to all mispredicted requests is bounded by $\lambda \cdot |M_2| + (2 - \alpha) \cdot \lambda \cdot |M_3|$.

Consider an optimal offline strategy of a request sequence $\langle r_1, r_2, \dots, r_m \rangle$. For each request r_i , if $t_i - t_{p(i)} \leq \lambda$, by Proposition 5, server $s[r_i]$ holds a copy throughout the period $(t_{p(i)}, t_i)$ to serve r_i , which incurs a storage cost of $t_i - t_{p(i)}$. If $t_i - t_{p(i)} > \lambda$, either server $s[r_i]$ holds a copy throughout the period $(t_{p(i)}, t_i)$ to serve r_i or an inward transfer to server $s[r_i]$ is performed to serve r_i . In both cases, the cost incurred is at least λ . Moreover, due to the at-least-one-copy requirement, the storage cost incurred between two successive requests r_{i-1} and r_i , i.e., over the period (t_{i-1}, t_i) , must be at least $t_i - t_{i-1}$. If $t_i - t_{i-1} > \lambda$, the portion of the storage cost beyond λ is not counted in the aforesaid lower bound. Hence, the optimal offline cost is at least $\sum_{i: t_i - t_{p(i)} > \lambda} \lambda + \sum_{i: t_i - t_{p(i)} \leq \lambda} (t_i - t_{p(i)}) +$

$\sum_{i:t_i-t_{i-1}>\lambda}(t_i - t_{i-1} - \lambda)$. Thus, with mispredictions, the increase in the online-to-optimal cost ratio is bounded by

$$\frac{\lambda \cdot |M_2| + (2 - \alpha) \cdot \lambda \cdot |M_3|}{\sum_{i:t_i-t_{p(i)}>\lambda} \lambda + \sum_{i:t_i-t_{p(i)}\leq\lambda} (t_i - t_{p(i)}) + \sum_{i:t_i-t_{i-1}>\lambda} (t_i - t_{i-1} - \lambda)}. \quad (11)$$

In Sections 6 and 7, we have shown that Algorithm 1 has a tight robustness of $1 + \frac{1}{\alpha}$ and a tight consistency of $\frac{5+\alpha}{3}$. Such a consistency-robustness tradeoff however is not attractive. In order to achieve a better consistency, we have to reduce the hyper-parameter α , but the robustness goes towards infinity as α decreases. On the other hand, when $\alpha = 1$ (setting the intended duration of the regular copy after each request to λ , irrespective of the prediction), the consistency and robustness become the same which is 2. This is in fact the best achievable competitive ratio of a conventional online algorithm [17]. Note that in our data replication problem, when a request arises, we will find out whether a previous prediction is correct or not. This makes it possible to optimize the robustness-consistency trade-off by changing the α setting when needed. An intuitive idea is to start with setting a small $\alpha < 1$ and reset α to 1 once a misprediction is found. Nevertheless, this strategy is not practically useful because it cannot make use of most predictions unless all predictions are correct. Im *et al.* [4, 5] addressed online problems with such temporal aspects by defining prediction error measures based on the difference in the optimal objective value between the predictions and ground truth. In a similar spirit, we propose to leverage the above analysis of mispredictions to monitor (an upper bound of) the online-to-optimal cost ratio as time goes on and adjust α dynamically to achieve a good consistency while maintaining a bounded robustness.

Specifically, we maintain a lower bound of the optimal cost as $\mathbf{OPTL} = \sum_{i:t_i-t_{p(i)}>\lambda} \lambda + \sum_{i:t_i-t_{p(i)}\leq\lambda} (t_i - t_{p(i)}) + \sum_{i:t_i-t_{i-1}>\lambda} (t_i - t_{i-1} - \lambda)$, i.e., the denominator of (11). When a new request r_i arises, we update \mathbf{OPTL} based on $t_i - t_{p(i)}$ and $t_i - t_{i-1}$. We also maintain an upper bound of the online cost $\mathbf{OnlineU}$ as the sum of two parts. The first part is the online costs allocated to all requests that have arisen (as given in Proposition 2). The second part is a conservative estimate of the costs beyond the last seen request at each server by treating the prediction after the last seen request as a misprediction. According to the above analysis, for each $r_i \in M_2$, the correct duration of the regular copy after $r_{p(i)}$ is λ and the penalty due to misprediction is bounded by λ , so the total cost is at most 2λ ; and for each $r_i \in M_3$, the correct duration of the regular copy after $r_{p(i)}$ is $\alpha\lambda$ and the penalty due to misprediction is bounded by $(2 - \alpha)\lambda$, so the total cost is also at most 2λ . Hence, the second part of the online cost is estimated as $2\lambda n'$, where n' is the number of servers that have received requests. When a new request r_i arises, we update $\mathbf{OnlineU}$. Suppose we would like to achieve a robustness of $2 + \beta$ where $\beta \geq 0$. If $\frac{\mathbf{OnlineU}}{\mathbf{OPTL}} > 2 + \beta$, we apply the conventional online algorithm by setting the intended duration of the regular copy after r_i to λ , irrespective of the prediction. If $\frac{\mathbf{OnlineU}}{\mathbf{OPTL}} \leq 2 + \beta$, we apply Algorithm 1 by setting the intended duration of the regular copy after r_i to $\alpha\lambda$ or λ according to the prediction. In this way, we can maintain a bounded robustness.

9 Lower Bound of Consistency

In this section, we establish a lower bound of $\frac{3}{2}$ on the consistency of any deterministic learning-augmented algorithm for our problem. This implies that it is not possible to achieve a consistency approaching 1 in our problem.

Consider two servers. The inter-request times at each server are always longer than λ . Correct predictions (i.e., forecasting the next request at each server to arise beyond a period of λ from the previous request) are input to the learning-augmented algorithm all the time. Initially there is only one data copy placed at one server, and a dummy request r_0 arises at this server at time 0. The request r_1

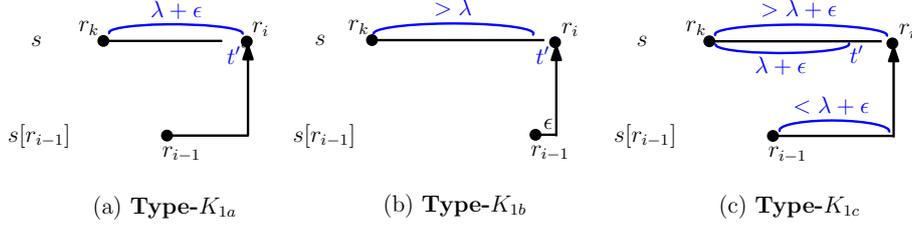


Figure 7: r_i is a **Type- K_1** request

arrives at the other server right after time 0 (i.e., $t_1 = \epsilon$ is a small value), so that in any online or offline strategy, r_1 is served by a transfer, and both servers hold data copies immediately thereafter.

The adversary generates subsequent requests according to the behaviour of the online (learning-augmented) algorithm in the following way. After a request r_{i-1} ($i \geq 2$) (to facilitate presentation, we use s to denote the server other than $s[r_{i-1}]$, and assume the last request at s before r_{i-1} is r_k where $k < i - 1$),

- If s does not hold a data copy at time $t' := \max\{t_{i-1} + \epsilon, t_k + \lambda + \epsilon\}$ (ϵ is a small value), the adversary generates the next request r_i in server s at time t' , so that r_i must be served by a transfer from $s[r_{i-1}]$. If $t' = t_k + \lambda + \epsilon$, we call r_i a **Type- K_{1a}** request (see Figure 7(a)).⁶ If $t' = t_{i-1} + \epsilon > t_k + \lambda + \epsilon$, we call r_i a **Type- K_{1b}** request (see Figure 7(b)).
- If s holds a data copy at time t' , the adversary monitors whether s drops its copy during the period $(t', t_{i-1} + \lambda)$. Once s drops its copy at some time instant $t^* \in (t', t_{i-1} + \lambda)$, the adversary generates the next request r_i in server s at time $t^* + \epsilon$ (i.e., right after s drops its copy), so that r_i must be served by a transfer from $s[r_{i-1}]$ (see Figure 7(c)). In this case, we also call r_i a **Type- K_{1c}** request.

In all the above cases, r_i and r_{i-1} are at different servers, so $p(i) = k$. We collectively call r_i a **Type- K_1** request in these cases. It is easy to make the following observation.

Observation 1. *If r_i is a **Type- K_1** request, $t_i - t_{i-1} < \lambda + \epsilon$. If r_i is a **Type- K_{1a}** request, $t_i - t_{p(i)} = \lambda + \epsilon$. If r_i is a **Type- K_{1b}** or **Type- K_{1c}** request, $t_i - t_{p(i)} > \lambda + \epsilon$.*

- If s persistently keeps its copy during the period $(t', t_{i-1} + \lambda)$, the adversary generates the next request r_i in server $s[r_{i-1}]$ at time $t_{i-1} + \lambda + \epsilon$. In this case, r_i and r_{i-1} are at the same server. We call such r_i a **Type- K_2** request. If $s[r_{i-1}]$ holds a data copy when r_i arises, r_i is served locally (see Figure 8(a)). Otherwise, r_i is served by a transfer from s (see Figure 8(b)).

Observation 2. *If r_i is a **Type- K_2** request, $t_i - t_{i-1} = \lambda + \epsilon$.*

We use $\mathbf{online}(r_i, r_j)$ to denote the total storage and transfer cost produced by the online algorithm over the period $(t_i, t_j]$. The proof of the following observation is given in Appendix H.

Observation 3.

- *If r_i is a **Type- K_{1a}** request, $\mathbf{online}(r_{i-1}, r_i) \geq (t_i - t_{i-1}) + \lambda$.*

⁶There might be other transfers between s and $s[r_{i-1}]$ during the period (t_{i-1}, t') by the online algorithm. To simplify illustration, they are not marked in Figures 7 and 8.

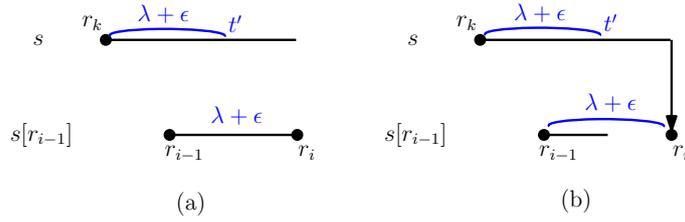


Figure 8: r_i is a **Type- K_2** request

- If r_i is a **Type- K_{1b}** or **Type- K_{1c}** request, $\mathbf{online}(r_{i-1}, r_i) \geq 2(t_i - t_{i-1}) + \lambda - \epsilon$.
- If r_i is a **Type- K_2** request, $\mathbf{online}(r_{i-1}, r_i) \geq 2\lambda + \epsilon$.

To analyze the online-to-optimal cost ratio, we partition the whole request sequence into subsequences. Each subsequence consists of one or two requests.

Case 1: A subsequence $\langle r_{i-1}, r_i \rangle$, where r_{i-1} is **Type- K_2** and r_i is **Type- K_1** .

Case 2: A subsequence $\langle r_{i-1}, r_i \rangle$ of two successive **Type- K_1** requests.

Case 3: A subsequence $\langle r_i \rangle$ of a single **Type- K_2** request.

For each subsequence, we construct an offline strategy and show that the cost ratio between the online algorithm and the offline strategy is at least $\frac{3}{2}$ or can be made arbitrarily close to $\frac{3}{2}$ as $\epsilon \rightarrow 0$. This then gives rise to a lower bound $\frac{3}{2}$ on the online-to-optimal cost ratio for the whole request sequence. Please refer to Appendix I for details. Therefore, the consistency of any deterministic learning-augmented algorithm is at least $\frac{3}{2}$.

10 Experimental Evaluation

We conduct simulation experiments to evaluate our algorithms using real data access traces. The results (1) verify our robustness and consistency analysis; (2) show that our algorithms can make effective use of predictions to improve performance; and (3) demonstrate that the adapted Algorithm 1 (presented in Section 8) can achieve a specified robustness. Due to space limitations, we defer the detailed experimental setup and results to Appendix J.

11 Concluding Remarks

Note that when $\alpha = 1$ (i.e., not using predictions), our algorithm achieves a competitive ratio of 2, which is better than the competitive ratio of 3 by the algorithm proposed in Wang *et al.* [16].

Recently, Wang *et al.* [17] presented an online algorithm for a multi-server system where the servers can have distinct storage costs, and claimed that the algorithm is 2-competitive. We find that the claim is not true. The competitive ratio of this algorithm is at least $\frac{5}{2}$, even if all servers have the same storage cost.

Let $\mu(s_i)$ denote the cost for storing a data copy in server s_i per time unit. Assume that the servers are indexed in ascending order of storage cost rate, i.e., $\mu(s_1) \leq \mu(s_2) \leq \dots \leq \mu(s_n)$. The main idea of Wang *et al.*'s algorithm is as follows.

- For each server s_i , after serving a local request (either by a transfer or by a local copy), s_i keeps the data copy for $\frac{\lambda}{\mu(s_i)}$ time units. Note that the cost of storing the copy in s_i over this period matches the cost of transferring the object.

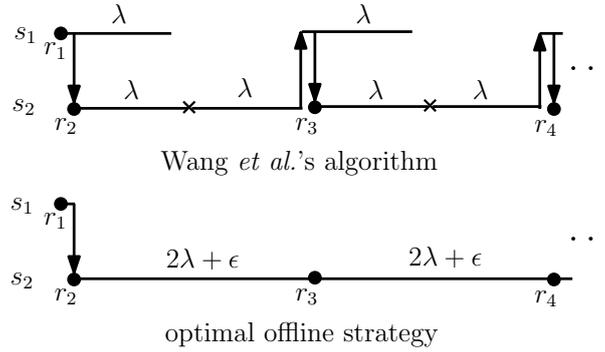


Figure 9: A counterexample of Wang *et al.*'s algorithm

- If a new request arises at s_i in this period, the request is served by the local copy and s_i keeps the copy for another $\frac{\lambda}{\mu(s_i)}$ time units (starting from the new request).
- When the data copy in s_1 expires (i.e., the server with the lowest storage cost rate), the algorithm checks whether s_1 holds the only copy in the system. If so, s_1 continues to keep the copy for another $\frac{\lambda}{\mu(s_1)}$ time units. Otherwise, s_1 drops its copy.
- When the data copy in s_i ($i \neq 1$) expires, the algorithm also checks whether s_i holds the only copy. If not, s_i drops its copy. If s_i holds the only copy, the algorithm further checks whether s_i has kept the copy for $\frac{\lambda}{\mu(s_i)}$ time units since its most recent local request (i.e., there is no request at s_i for $\frac{\lambda}{\mu(s_i)}$ time). If so, s_i continues to keep the copy for another $\frac{\lambda}{\mu(s_i)}$ time units. Otherwise, it implies that s_i has kept the copy for $\frac{2\lambda}{\mu(s_i)}$ time units without serving any local request. In that case, s_i transfers the object to s_1 and drops the local copy.

Figure 9 gives a counterexample to the claim of the competitive ratio 2, where two servers s_1 and s_2 both have storage cost rates 1. There are m requests. Requests r_1 arises at s_1 and all subsequent requests arise at s_2 . The arising times of the requests are $t_1 = 0$, $t_2 = \epsilon$, $t_3 = 2\lambda + 2\epsilon$, $t_4 = 4\lambda + 3\epsilon$, $t_5 = 6\lambda + 4\epsilon$ and so on,⁷ where $\epsilon > 0$ is a small value and every two consecutive requests at s_2 are slightly longer than 2λ apart.

By Wang *et al.*'s algorithm, after request r_1 , the copy in s_1 would expire at time $t_1 + \lambda = \lambda$, and after request r_2 , the copy in s_2 would expire at time $t_2 + \lambda = \lambda + \epsilon > \lambda$ which is after s_1 's copy expires. Hence, s_1 drops its copy when it expires. When s_2 's copy expires, since it is the only copy in the system, it is renewed for another λ time units. When the renewal expires, s_2 transfers the object to s_1 and drops the local copy. Then, the copy in s_1 would expire at time $3\lambda + \epsilon$. When request r_3 arises at s_2 , s_1 transfers the object to s_2 to serve r_3 , after which the copy in s_2 would expire at time $t_3 + \lambda = 3\lambda + 2\epsilon > 3\lambda + \epsilon$ which is again after s_1 's copy expires. Hence, s_1 drops its copy when it expires. When s_2 's copy expires, it is renewed for another λ time units. When the renewal expires, s_2 transfers the object to s_1 and drops the local copy. Then, the copy in s_1 would expire at time $5\lambda + 2\epsilon$. When request r_4 arises at s_2 , s_1 transfers the object to s_2 to serve r_4 , after which the copy in s_2 would expire at time $t_4 + \lambda = 5\lambda + 3\epsilon > 5\lambda + 2\epsilon$ which is again after s_1 's copy expires. This pattern is repeated continuously. As a result, the total cost of serving all requests is at least $(m - 2) \cdot 5\lambda$ (where we include the cost incurred up to the final request r_m only for fair comparison with the optimal offline solution).

⁷To simplify boundary conditions, Wang *et al.* [17] assumed that (1) the object is initially stored in s_1 (the server with the lowest storage cost rate) and (2) the first request arises at s_1 at time 0. Our example follows these assumptions.

In the optimal offline solution, server s_2 should always keep a data copy after request r_2 is served by a transfer. So, the total cost is $(m - 2) \cdot (2\lambda + \epsilon) + \lambda + \epsilon$.

It is easy to see that as $m \rightarrow \infty$ and $\epsilon \rightarrow 0$, the ratio between the online cost and the optimal offline cost approaches $\frac{5}{2}$.

References

- [1] Reuven Bar-Yehuda, Erez Kantor, Shay Kutten, and Dror Rawitz. Growing half-balls: Minimizing storage and communication costs in CDNs. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming*, pages 416–427, 2012.
- [2] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*, volume 53. Cambridge University Press Cambridge, 1998.
- [3] Sreenivas Gollapudi and Debmalya Panigrahi. Online Algorithms for Rent-or-Buy with Expert Advice. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2319–2327, 2019.
- [4] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 285–294, 2021.
- [5] Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Online dynamic acknowledgement with learned predictions. In *Proceedings of the 42nd IEEE Conference on Computer Communications (INFOCOM)*, pages 1–10, 2023.
- [6] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1-4):79–119, 1988.
- [7] Rohan Kodialam. Optimal Algorithms for Ski Rental with Soft Machine-Learned Predictions. *arXiv preprint arXiv:1903.00092*, 2019.
- [8] Ravi Kumar, Manish Purohit, and Zoya Svitkina. Improving Online Algorithms via ML Predictions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 9684–9693, 2018.
- [9] Yaser Mansouri and Abdelkarim Erradi. Cost optimization algorithms for hot and cool tiers cloud storage services. In *IEEE International Conference on Cloud Computing*, pages 622–629, 2018.
- [10] Yaser Mansouri, Adel Nadjaran Toosi, and Rajkumar Buyya. Data storage management in cloud environments: Taxonomy, survey, and future directions. *ACM Computing Surveys*, 50(6):1–51, 2017.
- [11] Yaser Mansouri, Adel Nadjaran Toosi, and Rajkumar Buyya. Cost optimization for dynamic replication and migration of data in cloud data centers. *IEEE Transactions on Cloud Computing*, 7(3):705–718, 2019.
- [12] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Communications of the ACM*, 65(7):33–35, 2022.

- [13] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2018.
- [14] Vijay V. Vazirani. *Approximation Algorithms*. Springer Science & Business Media, 2013.
- [15] Bharadwaj Veeravalli. Network caching strategies for a shared data distribution for a predefined service demand sequence. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1487–1497, 2003.
- [16] Yang Wang, Shuibing He, Xiaopeng Fan, Chengzhong Xu, and Xianhe Sun. On cost-driven collaborative data caching: A new model approach. *IEEE Transactions on Parallel and Distributed Systems*, 30(3):662–676, 2018.
- [17] Yang Wang, Yong Zhang, Xinxin Han, Pengfei Wang, Chengzhong Xu, Joseph Horton, and Joseph Culberson. Cost-driven data caching in the cloud: An algorithmic approach. In *Proceedings of the 40th IEEE Conference on Computer Communications (INFOCOM)*, pages 1–10, 2021.
- [18] Alexander Wei and Fred Zhang. Optimal Robustness-Consistency Trade-offs for Learning-Augmented Online Algorithms. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 8042–8053, 2020.

A Proof of Proposition 3

Proof. Since one data copy is initially placed in server s_1 where the dummy request r_0 arises at time 0, no transfer is needed to serve r_0 . Thus, we consider only transfers done after time 0.

Suppose that a transfer from a server s_x to a server s_y is carried out at time t in an optimal replication strategy, and there is no request at servers s_x and s_y at time t . It can be inferred that either (1) a data copy is held by s_x before the transfer, or (2) s_x just receives the data object from another server s_z at time t . In the latter case, since there is no request at s_x at time t , the transfer from s_x to s_y can be replaced by a transfer from s_z to s_y without affecting the cost of the replication strategy (see Figure 10 for an illustration). If s_z does not have a local request at time t and does not hold a copy before time t , we can find another server transferring to s_z at time t and repeat the replacement until a source server of the transfer having a local request at time t or holding a copy before time t is found. So without loss of generality, we assume that s_x holds a copy before the transfer.

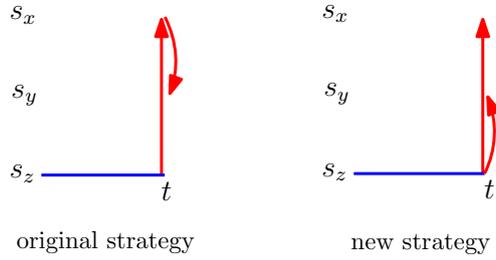


Figure 10: Source server holds a copy before the transfer

Similarly, since there is no request at s_y at time t , if s_y does not hold a copy after the transfer, s_y must be sending the data object to another server s_z at time t (otherwise, the transfer from s_x to s_y can be removed, which contradicts the optimality of the replication strategy). We can then replace the transfers from s_x to s_y and from s_y to s_z by a single transfer from s_x to s_z without affecting the ability of the replication strategy to serve all the requests, which again contradicts the optimality of the replication strategy (see Figure 11 for an illustration). Thus, s_y must hold a copy after the transfer.

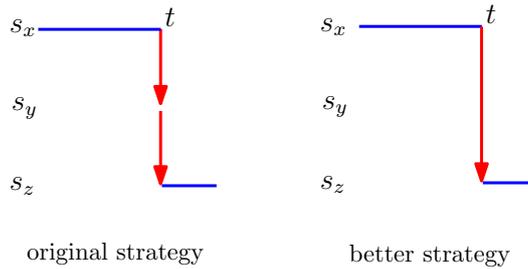


Figure 11: Destination server holds a copy after the transfer

If s_x continues to hold a copy after the transfer, we can delay the transfer from s_x to s_y by some small $\epsilon > 0$ and create a copy at s_y at time $t + \epsilon$ to save the storage cost (see Figure 12 for an illustration). This would not affect the feasibility of the replication strategy since all the transfers originating from s_y during the period $(t, t + \epsilon)$ can originate from s_x instead. As a result, it contradicts the optimality of the replication strategy.

Now suppose that s_x does not hold a copy after the transfer. Then it can be shown that the copy

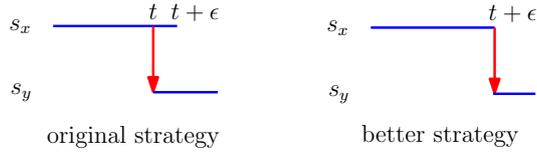


Figure 12: Transfer can be delayed if s_x keeps its copy after t

held by s_x before the transfer serves at least one local request at s_x . Otherwise, if the copy held by s_x before the transfer does not serve any local request at s_x , let t' denote the creation time of the copy. At time t' , there must be a transfer from another server s_z to s_x . We can replace the two transfers from s_z to s_x at time t' and from s_x to s_y at time t by one transfer from s_z to s_y at time t' , remove the copy at s_x and add a copy at s_y during (t', t) , which reduces the total cost, contradicting the optimality of the replication strategy (see Figure 13 for an illustration). Thus, the copy held by s_x before the transfer serves at least one local request at s_x . We look for the last such local request of s_x . Let t^* denote the time of this request. Then we can bring the transfer from s_x to s_y earlier to time t^* , remove the copy at s_x and add a copy at s_y during the period (t^*, t) (see Figure 14 for an illustration). This would not increase the cost of the replication strategy, and would not affect the service of other requests because all transfers originating from s_x during (t^*, t) can originate from s_y instead. In the new strategy, there is a request at s_x at the time of the transfer from s_x to s_y . \square

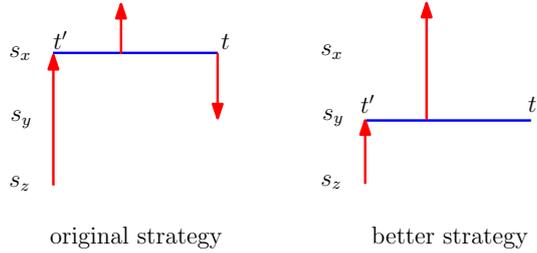


Figure 13: The copy at s_x must serve at least one local request

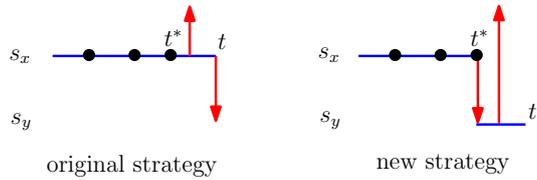


Figure 14: Change the strategy to have a request at the source server of the transfer

B Proof of Proposition 4

Proof. In an optimal replication strategy satisfying Proposition 3, suppose r_i is served by a local copy, but the copy is created later than $r_{p(i)}$. This implies that the copy must be created by a transfer (see Figure 15 for an illustration). By Proposition 3, there must be a request r_j at the source server of this transfer. Then, we can replace the copy at $s[r_i]$ during (t_j, t_i) with a copy at $s[r_j]$ during (t_j, t_i) , and

delay the transfer to the time t_i of r_i . This would not affect the service of other requests, because all transfers originating from $s[r_i]$ during this period can originate from $s[r_j]$ instead. As a result, the total cost does not change. In the new strategy, r_i is served by a transfer, and the characteristic in Proposition 3 is retained. \square

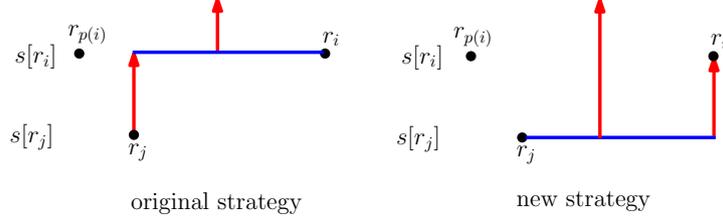


Figure 15: Illustration for the proof of Proposition 4

C Proof of Proposition 5

Proof. In an optimal offline strategy satisfying Propositions 3 and 4, if server $s[r_i]$ 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 request r_i , where the transfer cost incurred is λ . Since $t_i - t_{p(i)} \leq \lambda$, we can replace the transfer with a copy at $s[r_i]$ during $(t_{p(i)}, t_i)$ without increasing the total cost. Such replacement retains the characteristics in Propositions 3 and 4. \square

D Proof of Proposition 6

Proof. In an optimal replication strategy satisfying Propositions 3, 4 and 5, if the source server of the transfer is $s[r_{i-1}]$, (i) holds naturally. By assumption, since no server holds a copy crossing t_i , $s[r_{i-1}]$ must drop its copy after the transfer. If $s[r_{i-1}]$ does not keep its copy since t_{i-1} , the copy must be created by a transfer after t_{i-1} (see Figure 16 for an illustration). By Proposition 3, there must be a request r_j at the source server of this transfer. This leads to a contradiction, because r_{i-1} and r_i are two consecutive requests in the sequence. Hence, $s[r_{i-1}]$ must keep its copy since t_{i-1} , so (ii) also holds.

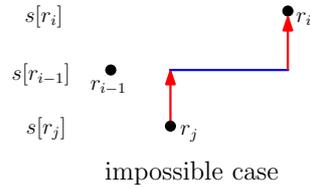


Figure 16: Illustration that $s[r_{i-1}]$ must keep a copy since t_{i-1}

Now suppose that the source server of the transfer is a server s other than $s[r_{i-1}]$. By assumption, since no server holds a copy crossing t_i , s must drop its copy after the transfer. It can be proved that the copy in s must be created no later than the last request r_h at s before r_i . Otherwise, the copy in s does not serve any local request at s , so the copy in s must be created by a transfer (see Figure 17 for an illustration). By Proposition 3, there must be a request r_j at the source server of this transfer. Then, we can replace the copy at s during (t_j, t_i) with a copy at $s[r_j]$ during (t_j, t_i) , replace the transfer from s to

$s[r_i]$ with a transfer from $s[r_j]$ to $s[r_i]$, and remove the transfer from $s[r_j]$ to s . This would not affect the service of other requests, because all transfers originating from s during this period can originate from $s[r_j]$ instead. As a result, the total cost is reduced, contradicting the optimality of the strategy. Thus, the copy in s must be created no later than the last request r_h at s before r_i . Note that since $s \neq s[r_{i-1}]$, we have $h < i - 1$.

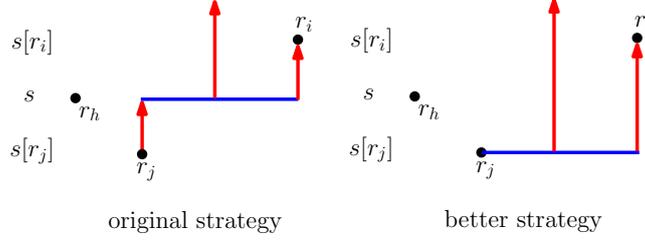


Figure 17: The copy in s must serve a local request

If r_{i-1} and r_i arise at the same server, we can replace the copy at s during (t_{i-1}, t_i) with a copy at $s[r_i]$ during (t_{i-1}, t_i) (see Figure 18 for an illustration). This replacement can save a transfer cost of λ , which contradicts the optimality of the strategy. Hence, r_{i-1} and r_i must arise at different servers. Then, we can replace the copy at s during (t_{i-1}, t_i) with a copy at $s[r_{i-1}]$ during (t_{i-1}, t_i) , and change the transfer for serving r_i to originate from $s[r_{i-1}]$ (see Figure 19 for an illustration). This does not affect the total cost of the strategy, and both (i) and (ii) hold in the new strategy. Meanwhile, the characteristics in Propositions 3, 4 and 5 are retained in the new strategy. \square

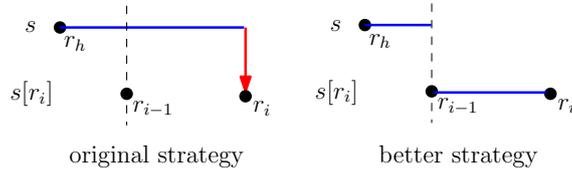


Figure 18: r_{i-1} and r_i arise at the same server

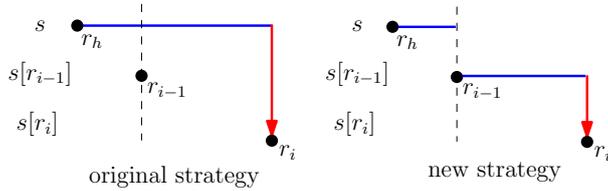


Figure 19: r_{i-1} and r_i arise at different servers

E Proof of Lemma 1

Proof. By our online algorithm, each special copy occurs only after the expiration of a regular copy which has a duration of $\alpha\lambda$ according to Proposition 8. It follows from Proposition 1 that all special copies must be at least $\alpha\lambda$ apart from each other. In addition, each period $(t_{p(i)}, t_i) \in X$ covers all

special copies therein and the regular copy after $r_{p(i)}$ (which has a duration $l_i = \alpha\lambda$ by Propositions 7 and 8). Thus, if a period $(t_{p(i)}, t_i) \in X$ covers y special copies, the total duration of these special copies is bounded by $(t_i - t_{p(i)}) - y \cdot \alpha\lambda$. Since each special copy is used to serve one request in $R_2 \cup R_4$ so that there are $|R_2| + |R_4|$ special copies in total, the lemma follows by aggregating the bounds for all the periods $(t_{p(i)}, t_i) \in X$:

$$\begin{aligned} \sum_{r_i \in R_2 \cup R_4} (t_i - t'_i) &\leq \sum_{(t_{p(i)}, t_i) \in X} (t_i - t_{p(i)}) - (|R_2| + |R_4|) \cdot \alpha\lambda \\ &= \sum_{r_i \in R_X} (t_i - t_{p(i)}) - (|R_2| + |R_4|) \cdot \alpha\lambda. \end{aligned}$$

□

F Proof of Lemma 2

Proof. In the case of $|X| \geq 2$, by Proposition 7, the end sentinel requests R_X of all storage periods in X are **Type-1/2/4** requests.

In the optimal offline strategy, following the discussion in Section 5, for each storage period $(t_{p(k_j)}, t_{k_j}) \in X$ where $j < |Q|$, $(t_{p(k_j)}, t_{k_j})$ crosses the start time $t_{p(k_{j+1})}$ of the next period $(t_{p(k_{j+1})}, t_{k_{j+1}}) \in Q$. In server $s[r_{k_{j+1}}]$, there must be a **Type-1/2/4** request in the period $(t_{p(k_j)}, t_{p(k_{j+1})})$. This is because if all requests in $s[r_{k_{j+1}}]$ during $(t_{p(k_j)}, t_{p(k_{j+1})})$ are **Type-3** requests, by definition, a regular copy is stored in $s[r_{k_{j+1}}]$ throughout $(t_{p(k_j)}, t_{p(k_{j+1})})$ in our online algorithm. By Proposition 1, it implies that there is no special copy in the period $(t_{p(k_j)}, t_{p(k_{j+1})})$, which contradicts the assumption $(t_{p(k_j)}, t_{k_j}) \in X$.

Let r_g denote the last **Type-1/2/4** request in server $s[r_{k_{j+1}}]$ in the period $(t_{p(k_j)}, t_{p(k_{j+1})})$. Then, there is a copy in server $s[r_{k_{j+1}}]$ throughout $(t_g, t_{p(k_{j+1})})$ (since all requests therein are **Type-3**) and hence $(t_g, t_{k_{j+1}})$. We can show that $r_g \notin R_X$ by contradiction. Assume on the contrary that $r_g \in R_X$, i.e., $(t_{p(g)}, t_g) \in X$. Then, there is a copy in server $s[r_{k_{j+1}}]$ throughout $(t_{p(g)}, t_{k_{j+1}})$ in the optimal offline strategy (see Figure 20 for an illustration). In addition, there is also a copy in server $s[r_{k_j}]$ during $(t_{p(k_j)}, t_{k_j})$. By Propositions 7 and 8, both periods $(t_{p(k_j)}, t_{k_j})$ and $(t_{p(g)}, t_g)$ are longer than λ . If $t_{p(g)} < t_{p(k_j)}$, the storage period $(t_{p(k_j)}, t_{k_j})$ can be replaced by a transfer from $s[r_{k_{j+1}}]$ to $s[r_{k_j}]$ to serve r_{k_j} which saves cost. If $t_{p(g)} > t_{p(k_j)}$, the storage period $(t_{p(g)}, t_g)$ can be replaced by a transfer from $s[r_{k_j}]$ to $s[r_{k_{j+1}}]$ to serve r_g which saves cost. In either case, it contradicts the optimality of the offline strategy. Therefore, we must have $r_g \notin R_X$. Moreover, note that the request r_g found falls in the period $(t_{p(k_j)}, t_{p(k_{j+1})})$. Thus, all the requests r_g 's for different storage periods in X are distinct.

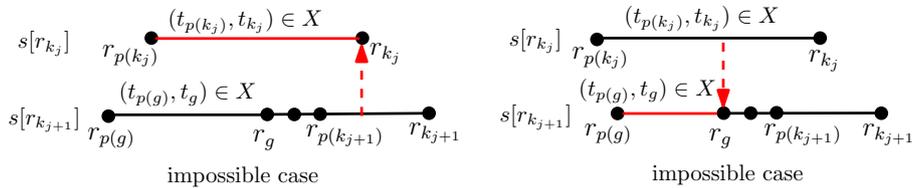


Figure 20: Illustration for the case of $|X| \geq 2$

In summary, for each storage period $(t_{p(k_j)}, t_{k_j}) \in X$ where $j < |Q|$, we can find two **Type-1/2/4** requests r_g and r_{k_j} . If the storage period $(t_{p(k_{|Q|}}), t_{k_{|Q|}}) \in X$, we can find one **Type-1/2/4** request $r_{k_{|Q|}}$. All the above requests are distinct. Thus, in total, there are at least $(2|X| - 1)$ **Type-1/2/4** requests.

In the case of $|X| = 1$, suppose $(t_{p(k_j)}, t_{k_j})$ is the only storage period in X . If $j < |Q|$, similar to the above arguments, we can find two **Type-1/2/4** requests. If $j = |Q|$, by Proposition 7, the end sentinel request $r_{k_{|Q|}} \in R_X$ is a **Type-1/2/4** request. To find another **Type-1/2/4** request, note that we have two different servers $s[r_{k_{|Q|}}]$ and $s[r_{k_{|Q|-1}}]$ since $|Q| \geq 2$. At least one of them is not the server $s[r_d]$ holding the copy over the first storage period $(t_{p(k_1)}, t_{k_1})$ in Q (where $p(k_1) = d$) in the optimal offline strategy. Let s_g denote that server. Let r_g denote the first request at s_g after time $t_{p(k_1)} = t_d$. By the partition definition, there is no copy at s_g crossing time t_d , so r_g must be served by a transfer in the optimal offline strategy. By Proposition 5, $t_g - t_{p(g)} > \lambda$. It follows from Proposition 8 that r_g is a **Type-1/2/4** request by our online algorithm. Together with $r_{k_{|Q|}}$, we have two **Type-1/2/4** requests. \square

G Proof of Lemma 3

Proof. First, we prove the existence of at least one request during $(t_{p(k_1)}, t_{k_1})$ at some server other than $s[r_{k_1}]$. Since r_{k_1} is a **Type-1/2** request, by Proposition 8, $t_{k_1} - t_{p(k_1)} > \lambda$ and the duration of the regular copy after $r_{p(k_1)}$ is $\alpha\lambda$. Suppose there is no request during $(t_{p(k_1)}, t_{k_1})$ at any server other than $s[r_{k_1}]$. For each server $s \neq s[r_{k_1}]$, we use r_h to denote its first local request after t_{k_1} , then $r_{p(h)}$ is its last local request before $t_{p(k_1)}$, i.e., $t_{p(h)} < t_{p(k_1)} < t_{k_1} < t_h$ (see Figure 21 for an illustration). Since $t_{k_1} - t_{p(k_1)} > \lambda$, we have $t_h - t_{p(h)} > \lambda$ and hence by Proposition 8, the duration of the regular copy after $r_{p(h)}$ is $\alpha\lambda$. This regular copy must expire before the regular copy after $r_{p(k_1)}$ since $t_{p(h)} < t_{p(k_1)}$. Thus, $s[r_{k_1}]$ must hold the only copy when its regular copy expires, so its regular copy will switch to a special copy and r_{k_1} will be served by a local copy. This implies that r_{k_1} is a **Type-4** request, which contradicts the assumption that r_{k_1} is a **Type-1/2** request.

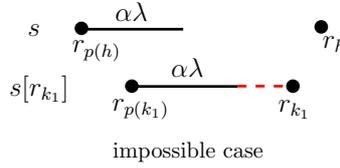


Figure 21: Illustration for the proof of Lemma 3

Next, consider a server $s \neq s[r_{k_1}]$ with requests arising during $(t_{p(k_1)}, t_{k_1})$. Let r_g denote the first request at s_g after time $t_{p(k_1)} = t_d$ (the beginning time of the partition). By the partition definition, there is no copy at s crossing time t_d , so r_g must be served by a transfer in the optimal offline strategy. By Proposition 5, $t_g - t_{p(g)} > \lambda$. It follows from Proposition 8 that r_g is a **Type-1/2/4** request by our online algorithm. Together with r_{k_1} , we have at least two **Type-1/2/4** requests. \square

H Proof of Observation 3

Proof. If r_i is a **Type- K_{1a}** request, $\mathbf{online}(r_{i-1}, r_i)$ includes a transfer cost of λ (for serving r_i) and a storage cost at least $t_i - t_{i-1}$ (since there is at least one data copy at any time).

If r_i is a **Type- K_{1b}** request, by the same arguments, $\mathbf{online}(r_{i-1}, r_i)$ is at least $(t_i - t_{i-1}) + \lambda = \epsilon + \lambda = 2(t_i - t_{i-1}) + \lambda - \epsilon$.

If r_i is a **Type- K_{1c}** request, $\mathbf{online}(r_{i-1}, r_i)$ can be divided into three parts: (1) the storage cost at s and the cost of transfers from $s[r_{i-1}]$ to s (if any) during (t_{i-1}, t_i) , (2) the storage cost at $s[r_{i-1}]$ and the cost of transfers from s to $s[r_{i-1}]$ (if any) during (t_{i-1}, t_i) , (3) the cost λ of the transfer to serve r_i at time

t_i . For part (1), by definition, s keeps a copy during $(t', t_i - \epsilon)$, where $t' = \max\{t_{i-1} + \epsilon, t_{p(i)} + \lambda + \epsilon\}$. If s persistently keeps its copy throughout the period $(t_{i-1}, t_i - \epsilon)$, its storage cost is $t_i - t_{i-1} - \epsilon$. Otherwise, there must be a transfer from $s[r_{i-1}]$ to s for s to hold a copy from t' onward, which incurs a transfer cost $\lambda > t_i - t_{i-1} - \epsilon$ by Observation 1. Hence, the cost of part (1) is at least $t_i - t_{i-1} - \epsilon$. For part (2), since r_i is served by a transfer from $s[r_{i-1}]$, it implies that $s[r_{i-1}]$ holds a copy at time t_i . If $s[r_{i-1}]$ persistently keeps its copy throughout the period (t_{i-1}, t_i) , its storage cost is $t_i - t_{i-1}$. Otherwise, there must be a transfer from s to $s[r_{i-1}]$ for $s[r_{i-1}]$ to hold a copy at t_i , which incurs a transfer cost $\lambda > t_i - t_{i-1} - \epsilon$ by Observation 1. Moreover, since s does not hold a copy during $(t_i - \epsilon, t_i)$, $s[r_{i-1}]$ must hold a copy during this period, which implies that the storage cost at $s[r_{i-1}]$ is at least ϵ . Hence, the cost of part (2) is at least $t_i - t_{i-1}$. Adding up parts (1), (2) and (3), we have $\mathbf{online}(r_{i-1}, r_i) \geq 2(t_i - t_{i-1}) + \lambda - \epsilon$.

In the case that r_i is a **Type- K_2** request, if there is at least one transfer between $s[r_{i-1}]$ and s during (t_{i-1}, t_i) , $\mathbf{online}(r_{i-1}, r_i)$ includes a transfer cost of λ and a storage cost at least $t_i - t_{i-1}$ (since there is at least one data copy at any time). Otherwise, if there is no transfer, $s[r_{i-1}]$ must persistently keep its copy during (t_{i-1}, t_i) to serve r_i locally, which incurs a storage cost $t_i - t_{i-1}$. Meanwhile, by definition, s persistently keeps its copy during $(t', t_{i-1} + \lambda + \epsilon)$. Since there is no transfer, then it must persistently keep its copy during $(t_{i-1}, t_{i-1} + \lambda + \epsilon)$, which incurs a storage cost $\lambda + \epsilon$. Hence, $\mathbf{online}(r_{i-1}, r_i)$ includes a total storage cost at least $t_i - t_{i-1} + \lambda = 2\lambda + \epsilon$ (Observation 2). \square

I Analysis of Subsequences

Similar to $\mathbf{online}(r_i, r_j)$, we use $\mathbf{offline}(r_i, r_j)$ to denote the total storage and transfer cost incurred over the period $(t_i, t_j]$ in the offline strategy constructed.

Case 1: A subsequence $\langle r_{i-1}, r_i \rangle$, where r_{i-1} is **Type- K_2** and r_i is **Type- K_1** . By definition, r_{i-2} and r_{i-1} are at the same server. By Observation 2, $t_{i-1} - t_{i-2} = \lambda + \epsilon$. In an offline strategy, $s[r_{i-1}]$ can keep its copy throughout the period (t_{i-2}, t_i) and serve r_i by a transfer (see Figure 22(a)). Hence, $\mathbf{offline}(r_{i-2}, r_i) \leq (t_i - t_{i-2}) + \lambda = (t_i - t_{i-1}) + 2\lambda + \epsilon$.

Since r_{i-1} is a **Type- K_2** request, $\mathbf{online}(r_{i-2}, r_{i-1}) \geq 2\lambda + \epsilon$ by Observation 3. Since r_i is at a different server from r_{i-2} and r_{i-1} , we must have $p(i) < i - 2$. It follows from $t_{i-1} - t_{i-2} = \lambda + \epsilon$ that $t_i - t_{p(i)} > t_{i-1} - t_{i-2} = \lambda + \epsilon$. By Observation 1, r_i must be a **Type- K_{1b}** or **Type- K_{1c}** request. Then by Observation 3, $\mathbf{online}(r_{i-1}, r_i) \geq 2(t_i - t_{i-1}) + \lambda - \epsilon$. Hence, $\mathbf{online}(r_{i-2}, r_i) = \mathbf{online}(r_{i-2}, r_{i-1}) + \mathbf{online}(r_{i-1}, r_i) \geq 2(t_i - t_{i-1}) + 3\lambda$. Thus, $\frac{\mathbf{online}(r_{i-2}, r_i)}{\mathbf{offline}(r_{i-2}, r_i)} \geq \frac{2(t_i - t_{i-1}) + 3\lambda}{(t_i - t_{i-1}) + 2\lambda + \epsilon}$, which approaches at least $\frac{3}{2}$ as $\epsilon \rightarrow 0$.

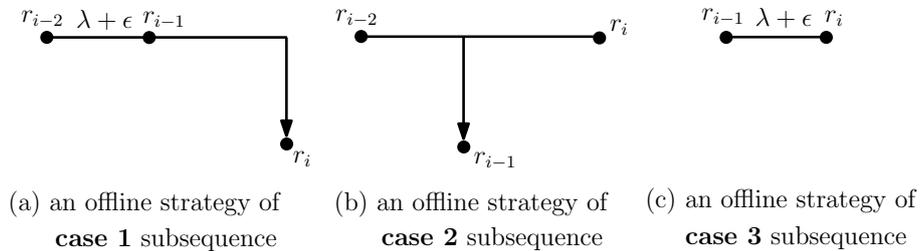


Figure 22: Offline strategies of **case 1/2/3** subsequences

Case 2: A subsequence $\langle r_{i-1}, r_i \rangle$ of two successive **Type- K_1** requests. By definition, r_{i-2} and r_i are at the same server, while r_{i-1} is at a different server. In an offline strategy, $s[r_i]$ can keep its copy

throughout the period (t_{i-2}, t_i) and serve r_{i-1} by a transfer (see Figure 22(b)). Hence, $\mathbf{offline}(r_{i-2}, r_i) \leq (t_i - t_{i-2}) + \lambda$.

(1) If r_i is a **Type- K_{1a}** request, we have $\mathbf{online}(r_{i-1}, r_i) \geq (t_i - t_{i-1}) + \lambda$ by Observation 3 and $t_i - t_{i-2} = t_i - t_{p(i)} = \lambda + \epsilon$ by Observation 1. Since r_{i-1} is a **Type- K_1** request, by Observation 3, we always have $\mathbf{online}(r_{i-2}, r_{i-1}) \geq (t_{i-1} - t_{i-2}) + \lambda$. Hence, $\mathbf{online}(r_{i-2}, r_i) = \mathbf{online}(r_{i-2}, r_{i-1}) + \mathbf{online}(r_{i-1}, r_i) \geq (t_i - t_{i-2}) + 2\lambda$. Thus, $\frac{\mathbf{online}(r_{i-2}, r_i)}{\mathbf{offline}(r_{i-2}, r_i)} \geq \frac{(t_i - t_{i-2}) + 2\lambda}{(t_i - t_{i-2}) + \lambda} = \frac{3\lambda + \epsilon}{2\lambda + \epsilon}$, which approaches $\frac{3}{2}$ as $\epsilon \rightarrow 0$.

(2) If r_i is a **Type- K_{1b}** or **Type- K_{1c}** request, we have $\mathbf{online}(r_{i-1}, r_i) \geq 2(t_i - t_{i-1}) + \lambda - \epsilon$ by Observation 3. Since r_{i-1} is a **Type- K_1** request, by Observation 3, we always have $\mathbf{online}(r_{i-2}, r_{i-1}) \geq (t_{i-1} - t_{i-2}) + \lambda$, and by Observation 1, we have $t_{i-1} - t_{i-2} < \lambda + \epsilon$. Hence, $\mathbf{online}(r_{i-2}, r_i) = \mathbf{online}(r_{i-2}, r_{i-1}) + \mathbf{online}(r_{i-1}, r_i) \geq (t_i - t_{i-2}) + (t_i - t_{i-1}) + 2\lambda - \epsilon$. Thus, $\frac{\mathbf{online}(r_{i-2}, r_i)}{\mathbf{offline}(r_{i-2}, r_i)} \geq \frac{(t_i - t_{i-2}) + (t_i - t_{i-1}) + 2\lambda - \epsilon}{(t_i - t_{i-2}) + \lambda} = 1 + \frac{(t_i - t_{i-1}) + \lambda - \epsilon}{(t_i - t_{i-1}) + (t_{i-1} - t_{i-2}) + \lambda} > 1 + \frac{(t_i - t_{i-1}) + \lambda - \epsilon}{(t_i - t_{i-1}) + 2\lambda + \epsilon}$, which approaches at least $\frac{3}{2}$ as $\epsilon \rightarrow 0$.

Case 3: A subsequence $\langle r_i \rangle$ of a single **Type- K_2** request. By definition, r_{i-1} and r_i are at the same server. By Observation 2, $t_i - t_{i-1} = \lambda + \epsilon$. In an offline strategy, $s[r_i]$ can keep its copy throughout the period (t_{i-1}, t_i) and serve r_i locally (see Figure 22(c)). Hence, $\mathbf{offline}(r_{i-1}, r_i) \leq t_i - t_{i-1} = \lambda + \epsilon$. By Observation 3, $\mathbf{online}(r_{i-1}, r_i) \geq 2\lambda + \epsilon$. Thus, $\frac{\mathbf{online}(r_{i-1}, r_i)}{\mathbf{offline}(r_{i-1}, r_i)} \geq \frac{2\lambda + \epsilon}{\lambda + \epsilon}$, which approaches 2 as $\epsilon \rightarrow 0$.

In summary, the cost ratio between the online algorithm and the offline strategy for any **case 1/2/3** subsequence is at least $\frac{3}{2}$ or can be made arbitrarily close to $\frac{3}{2}$ as $\epsilon \rightarrow 0$.

Any request sequence $\langle r_0, r_1, r_2, \dots, r_m \rangle$ generated by the adversary can be partitioned into **case 1/2/3** subsequences starting from r_1 or r_2 . Specifically, we can identify maximal segments of successive **Type- K_1** requests (called maximal **Type- K_1** segments for short). A maximal **Type- K_1** segment means that the requests immediately preceding and succeeding the segment are both **Type- K_2** requests. If a maximal **Type- K_1** segment has an even number of **Type- K_1** requests, we can further break it into **case 2** subsequences. If a maximal **Type- K_1** segment has an odd number of **Type- K_1** requests, we can pair the first **Type- K_1** request with the preceding **Type- K_2** request to form a **case 1** subsequence and break the remaining segment into **case 2** subsequences. For the rest **Type- K_2** requests, each of them constitutes a **case 3** subsequence on its own. With the above partitioning, we are left with only $\langle r_0, r_1 \rangle$ or $\langle r_0, r_1, r_2 \rangle$ at the beginning of the request sequence, where $\langle r_0, r_1, r_2 \rangle$ happens only when r_2 is at the head of a maximal **Type- K_1** segment of odd length.

The aforesaid offline strategies constructed for individual **case 1/2/3** subsequences can be concatenated to form an offline strategy for all the requests after r_1 or r_2 . Thus, by the above analysis, $\frac{\mathbf{online}(r_1, r_m)}{\mathbf{offline}(r_1, r_m)}$ or $\frac{\mathbf{online}(r_2, r_m)}{\mathbf{offline}(r_2, r_m)}$ approaches at least $\frac{3}{2}$ as $\epsilon \rightarrow 0$, where $\mathbf{offline}(r_i, r_j)$ denotes the total storage and transfer cost incurred over the period $(t_i, t_j]$ in the offline strategy constructed. Next, let us consider the requests left at the beginning. In the case of $\langle r_0, r_1 \rangle$, an offline strategy can perform a transfer at time t_1 to serve r_1 and then both servers hold data copies right after r_1 for whatever replication strategy to carry on (see Figure 23). As a result, $\mathbf{offline}(r_0, r_1) = \lambda + \epsilon$. Consequently, $\frac{\mathbf{online}(r_0, r_m)}{\mathbf{offline}(r_0, r_m)} \geq \frac{\mathbf{online}(r_1, r_m)}{\mathbf{offline}(r_0, r_m)} = \frac{\mathbf{online}(r_1, r_m)}{\mathbf{offline}(r_1, r_m) + \lambda + \epsilon}$, which approaches at least $\frac{3}{2}$ as $\epsilon \rightarrow 0$ and $m \rightarrow \infty$. Similarly, in the case of $\langle r_0, r_1, r_2 \rangle$, an offline strategy can hold data copies at both servers after r_1 until r_2 and then they both have copies right after r_2 for whatever replication strategy to carry on (see Figure 24). As a result, $\mathbf{offline}(r_0, r_2) < 3\lambda + 3\epsilon$.⁸ Hence, $\frac{\mathbf{online}(r_0, r_m)}{\mathbf{offline}(r_0, r_m)} \geq \frac{\mathbf{online}(r_2, r_m)}{\mathbf{offline}(r_0, r_m)} > \frac{\mathbf{online}(r_2, r_m)}{\mathbf{offline}(r_2, r_m) + 3\lambda + 3\epsilon}$, which also approaches at least $\frac{3}{2}$ as $\epsilon \rightarrow 0$ and $m \rightarrow \infty$. Therefore, the consistency of any deterministic learning-augmented algorithm must be at least $\frac{3}{2}$.

⁸Since r_2 is a **Type- K_1** request, by Observation 1, the time duration between r_1 and r_2 is shorter than $\lambda + \epsilon$.

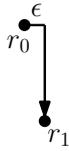


Figure 23: An offline strategy for $\langle r_0, r_1 \rangle$

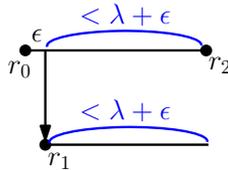


Figure 24: An offline strategy for $\langle r_0, r_1, r_2 \rangle$

J Experimental Evaluation

J.1 Experimental Setup

We conduct simulation experiments to evaluate our proposed algorithms, using the object storage traces provided by IBM (available at <https://www.ibm.com/cloud/blog/object-storage-traces>). The traces include read and write requests made to objects in a cloud-based object storage over 7 days in the year of 2019. We select several objects and filter out their write operations. For each object, we randomly distribute its requests over 10 different servers following a Zipf distribution, where each request is assigned to a server indexed by i with probability $i^{-1} / \sum_{j=1}^{10} j^{-1}$ ($i = 1, 2, \dots, 10$). The storage cost rate of each server is assumed to be 1 per second. The predictions of inter-request times are randomly generated according to the ground truth and a specified prediction accuracy. We obtain the online costs of serving the request sequence by running our proposed algorithm with different combinations of hyper-parameter $\alpha \in \{0, 0.1, 0.2, \dots, 1\}$, transfer cost $\lambda \in \{10, 100, 1000, 10000\}$ and prediction accuracy of $\{0\%, 10\%, 20\%, \dots, 100\%\}$. We normalize these online costs by the optimal offline costs derived from dynamic programming [16], and produce 3-dimensional plots to illustrate the algorithm performance under different settings. Similar performance trends are observed for the objects that we experiment with. Thus, we present the representative results of one particular object (with identifier “652aaef228286e0a”) which has 11688 read requests over 7 days.

J.2 Experimental Results

First, we study the original Algorithm 1. Figures 25 to 28 show the online-to-optimal cost ratios for different settings (note that the vertical axis has different scales in different figures). All the ratios are bounded by $1 + \frac{1}{\alpha}$ (robustness derived in Section 6), and the ratios of those 100% prediction accuracy settings are all bounded by $\frac{5+\alpha}{3}$ (consistency derived in Section 7). These results verify our robustness and consistency analysis.

One common feature of all the figures is that when the hyper-parameter $\alpha = 1$, the online-to-optimal cost ratio is a constant, no matter what the prediction accuracy is. The reason is that when $\alpha = 1$, the proposed algorithm does not make use of predictions to adjust the durations of regular copies and thus its performance is independent of the prediction accuracy. Another common feature is that the online-to-optimal cost ratio attains its minimum when the hyper-parameter α gets close to 0 and the prediction

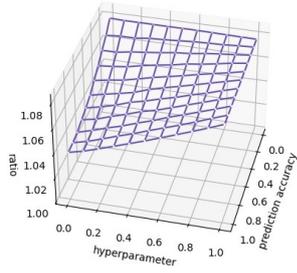


Figure 25: $\lambda = 10$

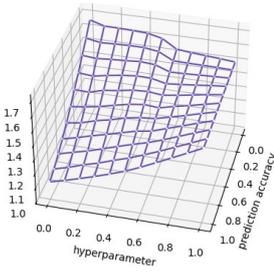


Figure 26: $\lambda = 100$

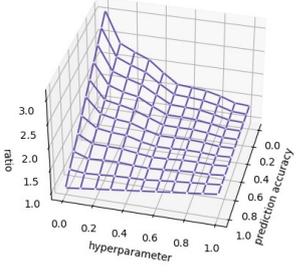


Figure 27: $\lambda = 1000$

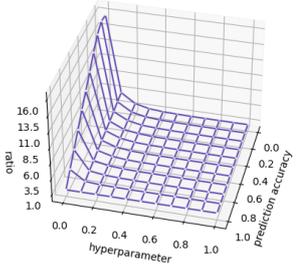


Figure 28: $\lambda = 10000$

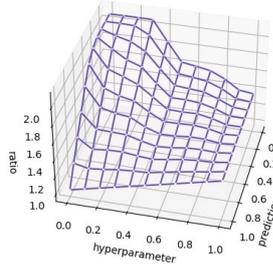


Figure 29: $\lambda = 1000, \beta = 0.1$

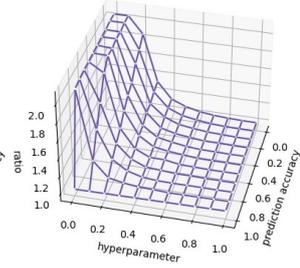


Figure 30: $\lambda = 10000, \beta = 0.1$

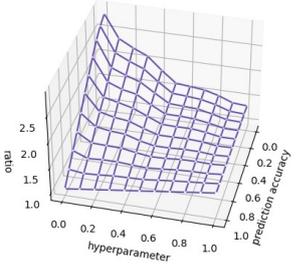


Figure 31: $\lambda = 1000, \beta = 1$

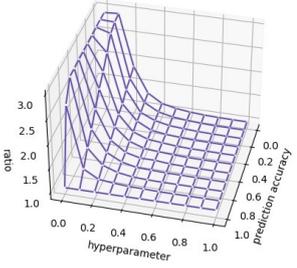


Figure 32: $\lambda = 10000, \beta = 1$

accuracy goes towards 100%. This is because when the predictions forecast future request arrivals correctly and the proposed algorithm makes almost full use of the predictions, the online cost would be reduced. This demonstrates the effectiveness of our proposed algorithm when incorporating accurate predictions.

Since there are 11688 requests over 7 days distributed among 10 servers, the average inter-request time at a server is about 500 seconds. When the transfer cost λ is very low (Figure 25), inter-request times are generally much greater than λ . Hence, in the optimal offline strategy, usually only one data copy is maintained in the system, and most requests are served by transfers. By our online algorithm, the regular copies have intended durations bounded by λ irrespective of the prediction accuracy, and thus most requests are also served by transfers. Since inter-request times are much greater than λ , one special copy is typically maintained in the system. Therefore, our online algorithm behaves similarly to the optimal offline strategy. As a result, the online-to-optimal cost ratio is close to 1.

When the transfer cost λ is higher (Figures 26, 27 and 28), the online-to-optimal cost ratio reaches its peak when both the hyper-parameter α and prediction accuracy approach 0. The reason is that as λ grows, more inter-request times become smaller than λ , and hence more requests are served by local copies in the optimal offline strategy. For those requests r_i 's where $\alpha\lambda < t_i - t_{p(i)} \leq \lambda$, mispredictions would lead to them served by transfers by our online algorithm, which incurs unnecessary transfer cost. Therefore, the performance of our online algorithm deteriorates as the prediction accuracy and α decrease. When the transfer cost λ is very high (Figure 28), there is almost no difference among the online-to-optimal cost ratios for different settings, unless both α and prediction accuracy approach 0. In most settings, the ratios are close to 1. This is because most inter-request times are less than $\alpha\lambda$ in this case. By our online algorithm, the regular copies have intended durations at least $\alpha\lambda$ irrespective of the prediction accuracy, and thus most requests are served by local copies, consistent with the optimal offline strategy.

Next, we study the adapted Algorithm 1 to achieve a robustness of $2 + \beta$. We run the original Algorithm 1 for the first 100 requests of the trace as a warm-up to get the initial **OnlineU** and **OPTL**. After that, we run the adapted Algorithm 1 to decide whether to set the intended duration of the regular copy after each request always to λ or according to the prediction. Figures 29 to 32 show the online-to-optimal cost ratios for $\lambda = 1000, 10000$ and $\beta = 0.1, 1$. As can be seen, the adapted Algorithm 1 successfully prevents the cost ratio from growing beyond the target robustness. This demonstrates the effectiveness of our adaptation to deal with terrible predictions. The results for $\lambda = 10, 100$ are the same as those for the original Algorithm 1 (hence not repeated here) because the online-to-optimal cost ratios are well below the target robustness $2 + \beta$.