A Randomized Caching Algorithm for Distributed Data Access

Tianyu Zuo
School of Comp. Sci. & Eng.
Nanyang Technological University
Singapore
zuot0001@e.ntu.edu.sg

Xueyan Tang
School of Comp. Sci. & Eng.
Nanyang Technological University
Singapore
asxytang@ntu.edu.sg

Bu Sung Lee
School of Comp. Sci. & Eng.
Nanyang Technological University
Singapore
ebslee@ntu.edu.sg

***Abstract*—In this paper, we study an online cost optimization problem for distributed data access. The goal of this problem is to dynamically create and delete data copies in a multi-server distributed system as time goes, in order to minimize the total storage and network cost of serving access requests. We propose an online algorithm with randomized storage periods of data copies in the servers, and derive an optimal probability density function of storage periods, which makes the algorithm achieve a competitive ratio of $1+\frac{\sqrt{2}}{2}$. An example is presented to show that the competitive analysis of our algorithm is tight. Experimental evaluations using real data access traces demonstrate that our algorithm outperforms the best known deterministic algorithm.**

I. Introduction

Over the past decade, the world has witnessed rapid growth in the amount of data created, captured, and consumed globally [22]. Network-based applications often deploy geo-distributed storage to facilitate data access and improve quality of service. In distributed storage services such as clouds, a natural problem arises as to minimize the overall cost of storing data and serving their access requests. This service cost normally includes 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 [15], [16]. Storage cost is typically proportional to the storage period of the data copy in each server, and network cost is proportional to the amount of network traffic when transferring data [6], [12].

In this paper, we study a cost optimization problem for distributed data access. Our goal is to manage data copies in a distributed system, so that the requests for data access can be satisfied with minimal service costs. A variety of works have been done for different cost optimization targets in distributed storage. Boukhelef *et al.* [3] proposed heuristic algorithms to periodically optimize the placement of data objects in two storage device classes, i.e., SSD and HDD, to balance a trade-off between monetary cost and performance. Mansouri *et al.* [14] developed data placement algorithms in cloud storage services that offer two storage tiers with different quality of service and prices, according to historical user accesses. Veeravalli [24] studied a general problem of migrating and caching shared data in a network of servers. He applied dynamic programming to derive the optimal cost with full knowledge of user requests in advance. Similar studies of cost optimization by data placement and migration can also be found in [8], [13], [20], with different models and considerations of service level agreements.

However, the works mentioned above either assume prior knowledge of user requests, or are only based on historical data accesses which lack performance guarantees if the access pattern changes. Thus, these methods may not effectively tackle a practical scenario where future data access requests are not known beforehand. Competitive analysis, referring to an idea of guaranteeing a bound on the performance of an online algorithm over all possible instances, can be applied to deal with such uncertainty. In competitive analysis, a metric called *competitive ratio* is used to measure the worst-case performance of an online algorithm (without knowledge of future) against the optimal offline solution (with full knowledge of future) [2], [23]. There are some works in this line of research for optimizing the total storage and network cost. Assuming identical storage cost rates for all servers, Bar-Yehuda *et al.* [1] designed an $O(\log \delta)$-competitive online algorithm, where δ is the normalized diameter of the underlying network. Wang *et al.* [25] proposed a 3-competitive online algorithm for a multi-server system where all servers have the same storage cost rate and transfer cost rate. Recently, they also considered a system where servers may have distinct storage cost rates, and developed a 2-competitive online algorithm [26].

Inspired by the above studies, we aim at improving the performance of online algorithms, by incorporating *randomization* into the algorithm design. Randomization can make considerable enhancement in the competitive ratio for many online problems [9]–[11].

Our contributions are summarized as follows.

1) We propose an online algorithm with randomized storage periods of data copies in the servers (Section IV).
2) To derive the optimal probability density function $f(t)$ of storage periods, we model and solve an optimization problem. With the optimal $f(t)$, our algorithm achieves a competitive ratio of $1+\frac{\sqrt{2}}{2} \approx 1.7$ (Section V).
3) We present an example to show the tightness of competitive analysis. Using real data access traces, we also experimentally demonstrate that our randomized algorithm outperforms the state-of-the-art deterministic algorithm.

II. Problem Definition

We consider a system with n geo-distributed servers (or sites) $s_1, s_2, \ldots, s_n$. A data object is hosted in the system

and copies of this object can be created, stored and deleted in any servers.[1] By normalization of related cost values, we assume that storing a data copy in each server incurs a cost of 1 per time unit. Whenever needed, the data object can also be transferred among servers. A transfer of the object between any two servers costs λ ($\lambda > 0$).

Requests to access the data object arise at different servers as time passes, owing to computational tasks or instant user requirements at local servers. When a request arises at a server s_j, if s_j holds a data copy at that time, the request is served locally. Otherwise, a transfer of the data object is needed from other server whichever holds a copy, to serve the request at s_j. After serving the request, s_j can keep (cache) the imported copy for some time, in case another local request arises soon afterwards. We denote the sequence of requests arising in the system as $\langle r_1, r_2, \ldots, r_m \rangle$. For each request r_i, we use t_i to denote the time when it arises. For simplicity, we assume that all the requests in the sequence are ordered chronologically, i.e., $0 < t_1 < t_2 < \cdots < t_m$. Moreover, we use $s[r_i]$ to represent the server where r_i arises. We assume that initially there is only one data copy in the system placed in server s_1. 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 aim to develop a *randomized caching algorithm* that minimizes the expected total cost of storage and transfer in the system, under the conditions that (1) all requests in the sequence are served, and (2) there is at least one copy in the system at any time. We focus on an *online* setting, i.e., the arising time and location of each request remains unknown until it arises. Our main metric for algorithm analysis is the *competitive ratio* [2], meaning the worst-case ratio between the (expected) cost generated by the (randomized) online algorithm and the cost of the optimal offline strategy.

III. Optimal Offline Cost

First, we analyze an optimal offline strategy, and propose a method to allocate the total optimal offline cost to each individual request, which forms a building block for the competitive analysis of our proposed algorithm later.

A. Characteristics of An Optimal Offline Strategy

The exact form of an optimal offline strategy for a request sequence is not straightforward to derive. Nevertheless, some characteristics of it can be derived to facilitate our analysis.

Proposition 1. *There exists an optimal offline 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 Proposition 1 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. Since the idea is straightforward, we omit a formal proof due to space limitations.

To facilitate presentation, for each request r_i, we define $r_{p(i)}$ as the preceding request of r_i arising at the same server $s[r_i]$. If r_i is the first request at its local server, both $p(i)$ and $t_{p(i)}$ are defined as $-\infty$.

Proposition 2. *There exists an optimal offline strategy with the characteristic in Proposition 1 and that for each request r_i, if r_i is served by a local copy, the copy is created no later than $r_{p(i)}$.*

Proof. In an optimal offline strategy satisfying Proposition 1, 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 1(a)). By Proposition 1, 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 (see Figure 1(b)). 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 1 is retained. □

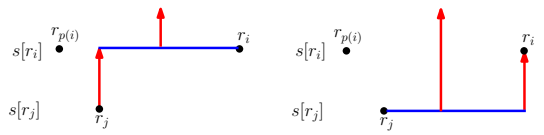


Fig. 1. Illustration for the proof of Proposition 2

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.

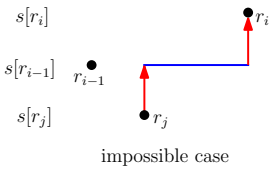
Proposition 3. *There exists an optimal offline strategy with the characteristics in Propositions 1, 2 and that for each request r_i, if $t_i - t_{p(i)} \leq \lambda$, server $s[r_i]$ holds a copy throughout the period $(t_{p(i)}, t_i)$, so that r_i is served by a local copy.*

Proof. In an optimal offline strategy satisfying Propositions 1 and 2, 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, while retaining the features in Propositions 1 and 2. □

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.

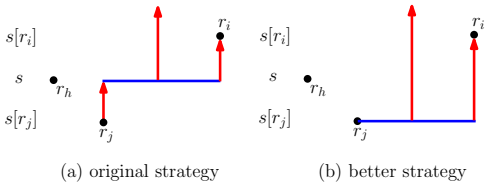
Proposition 4. *There exists an optimal offline strategy with the characteristics in Propositions 1, 2, 3 and that for each request r_i, if r_i is served by a transfer and no server holds a copy crossing the time t_i of r_i, then (i) r_{i-1} and r_i arise at different servers; and (ii) the source server of the transfer is $s[r_{i-1}]$ which keeps a copy since t_{i-1}.*

[1] We do not consider any capacity limit 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.



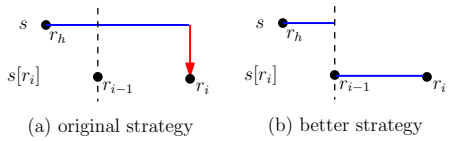
impossible case

Fig. 2. $s[r_{i-1}]$ must keep a copy since t_{i-1}



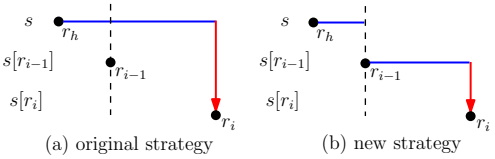
(a) original strategy (b) better strategy

Fig. 3. The copy in s must serve a local request



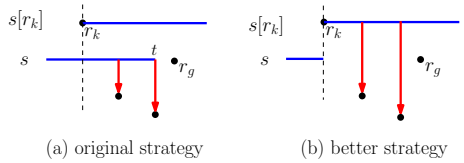
(a) original strategy (b) better strategy

Fig. 4. r_{i-1} and r_i arise at the same server



(a) original strategy (b) new strategy

Fig. 5. r_{i-1} and r_i arise at different servers



(a) original strategy (b) better strategy

Fig. 6. Illustration of a data copy at s crossing time t_k

Proof. In an optimal offline strategy satisfying Propositions 1, 2 and 3, if the source server of the transfer to r_i 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 hold its copy since t_{i-1}, the copy must be created by a transfer after t_{i-1} (see Figure 2 for an illustration). By Proposition 1, 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 hold its copy since t_{i-1}, so (ii) also holds.

Now suppose that the source server of the transfer to r_i 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 3(a) for an illustration). By Proposition 1, 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 (see Figure 3(b)). 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 $h < i - 1$.

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 4). 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 5 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 1, 2 and 3 are retained in the new strategy. □

Hereafter, an optimal offline strategy shall always refer to one with the characteristics in Propositions 1, 2, 3 and 4.

B. Allocation of Optimal Offline Cost

Starting from the last request r_m, we pick a set of storage periods of data copies in an optimal offline strategy via backtracking to cover the time span of the request sequence.

We define a variable r_k as the current request reached in backtracking, and it is initialized as r_m. During backtracking, the storage periods of data copies picked always cover the time span from r_k onward.

If r_k is served by a local copy, by Proposition 2, this copy must be stored in server $s[r_k]$ from $r_{p(k)}$ to r_k. We pick the storage period $(t_{p(k)}, t_k)$ of the copy at $s[r_k]$, and update r_k as $r_{p(k)}$.

If r_k is served by a transfer, we check whether there exists a copy at some other server $s \neq s[r_k]$ crossing t_k. If so, we can show that the copy at s must be stored till at least the first local request r_g at s after t_k. Suppose this copy does not serve any local request at s after t_k (see Figure 6), then it can be deleted earlier at t_k, because all the outgoing transfers from the copy at s after t_k can originate from some other copies (since the storage periods of data copies picked so far cover the time span from r_k onward). This reduces the total cost, which contradicts the optimality of the strategy. Now since r_g is served locally, by Proposition 2, the copy at $s[r_g]$ is kept from $r_{p(g)}$ to r_g. We pick the storage period $(t_{p(g)}, t_g)$ of the copy at $s[r_g]$, and update r_k as $r_{p(g)}$.

If there is no data copy crossing t_k, by Proposition 4, a copy is kept in a server $s[r_{k-1}] \neq s[r_k]$ from t_{k-1} to t_k, and r_k is served by a transfer from $s[r_{k-1}]$. We pick the storage period (t_{k-1}, t_k) of the copy at $s[r_{k-1}]$, and update r_k as r_{k-1}.

When r_k goes to the dummy request r_0, backtracking is completed. Eventually, we collect a set of storage periods of data copies covering the time span of request sequence. Each storage period picked either (1) starts and ends with two successive requests $r_{p(i)}$ and r_i at the same server, or

(2) starts with a request r_{i-1}, and ends with a transfer to the next request r_i. We refer to the request r_i at the end of the storage period as the *end sentinel request*, and use Q_1 and Q_2 to denote the sets of end sentinel requests in cases (1) and (2) respectively. Since all the storage periods picked cover the time span of the request sequence, we have $\sum_{r_i \in Q_1}(t_i - t_{p(i)}) + \sum_{r_i \in Q_2}(t_i - t_{i-1}) \geq t_m - t_0$. Note that for each $r_i \in Q_2$, since r_i is served by a transfer in the optimal offline strategy, by Proposition 3, it must hold that $t_i - t_{p(i)} > \lambda$.

We allocate the total optimal offline cost to each request in the sequence. For each end sentinel request r_i, if $r_i \in Q_1$, we allocate to r_i the storage cost of the copy in server $s[r_i]$ during $(t_{p(i)}, t_i)$. If $r_i \in Q_2$, we allocate to r_i the storage cost of the copy in server $s[r_{i-1}]$ during (t_{i-1}, t_i), and the transfer cost of λ to serve r_i.

For other requests that are not end sentinel requests, we use R_L to denote those requests r_i with $t_i - t_{p(i)} \leq \lambda$, and use R_T to denote those requests r_i with $t_i - t_{p(i)} > \lambda$. By Proposition 3, each request $r_i \in R_L$ is served locally by the copy kept in $s[r_i]$ from $t_{p(i)}$ to t_i. Hence, we allocate the cost of the storage period $(t_{p(i)}, t_i)$ to each $r_i \in R_L$. Recall that we have picked a set of storage periods that covers the whole time span. Thus, to optimize the total cost, each request $r_i \in R_T$ should be served by a transfer from the copy associated with one of the storage periods picked (which incurs a transfer cost of λ), since serving r_i by a local copy kept from $r_{p(i)}$ to r_i incurs a storage cost larger than λ. Hence, we allocate the transfer cost of λ to each $r_i \in R_T$.

We use $\mathbf{OPT}(r_i)$ to denote the offline cost allocated to a request r_i and summarize the cost allocation as follows.

Proposition 5. *The optimal offline cost allocated to a request r_i is given by*

- $\forall r_i \in Q_1$: $\boldsymbol{OPT}(r_i) = t_i - t_{p(i)}$;
- $\forall r_i \in Q_2$: $\boldsymbol{OPT}(r_i) = (t_i - t_{i-1}) + \lambda$, *and it holds that* $t_i - t_{p(i)} > \lambda$;
- $\forall r_i \in R_L$: $\boldsymbol{OPT}(r_i) = t_i - t_{p(i)}$, *and it holds that* $t_i - t_{p(i)} \leq \lambda$;
- $\forall r_i \in R_T$: $\boldsymbol{OPT}(r_i) = \lambda$, *and it holds that* $t_i - t_{p(i)} > \lambda$.

Since all the storage costs and transfer costs in the optimal offline strategy have been allocated, the total optimal offline cost is equal to $\sum_{1 \leq i \leq m} \mathbf{OPT}(r_i)$. Note that the dummy request r_0 is not allocated any cost.

IV. Randomized Online Algorithm

A. Algorithm Design

The intuition of our algorithm design is that after serving each request, the local server can hold its data copy for some period of length t. We randomize this length t according to some predefined distribution, and refer to such a period of t as an *intended storage period*. If a request arises at a server during the intended storage period, the local server renews the copy for a new period according to the predefined distribution. Otherwise, if no request arises during the intended storage period, the copy should be deleted when it expires. When the intended storage periods of all servers expire, we let the server with the last expiring copy continue to keep its copy so as to maintain at least one copy in the system.[2]

Algorithm 1 Randomized Caching

```
1:  randomly generate a storage period t based on f(t);
2:  initialize: c ← 1; E_1 ← t; E_j ← −∞ for all 2 ≤ j ≤ n;
    K_j ← 0 for all 1 ≤ j ≤ n;          ▷ 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 ≤ E_j or K_j = 1 then    ▷ 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 ← c + 1;
10:     randomly generate a storage period t based on f(t);
11:     E_j ← t_i + t;
12:     K_j ← 0;
13: upon (s_j transfers the object to another server s_k) do
14:     if K_j = 1 then                 ▷ s_j holds the only copy
15:         drop the copy in s_j;
16:         K_j ← 0;
17:         c ← c − 1;
18: upon (a copy expires in server s_j at time E_j) do
19:     if c = 1 then                   ▷ s_j holds the only copy
20:         K_j ← 1;
21:     else
22:         drop the copy in s_j;
23:         c ← c − 1;
```

Algorithm 1 shows the details of our randomized algorithm. E_j denotes the expiration time of the intended storage period in server s_j. K_j is a binary tag to indicate whether server s_j keeps the local copy beyond its expiration time E_j. c records the total number of servers holding data copies. Initially, only server s_1 has a data copy, so $c = 1$ (line 2).

When a request r_i arises at a server s_j, if s_j holds a data copy, r_i is served locally (lines 4-5). Otherwise, r_i is served by a transfer from another server holding a copy (lines 6-9). In both cases, after r_i is served, we randomly generate an intended storage period t according to a *probability density function* $f(t)$ (line 10). We let s_j keep its data copy for a period of length t, and clear its tag K_j (lines 11-12). We will elaborate how to set $f(t)$ in Section V.

During the intended storage period, if another request arises at s_j, s_j renews the local copy for a new intended storage period following $f(t)$. When the intended storage period of s_j expires, if s_j holds the only copy in the system, it continues to keep the copy and activates its tag K_j (lines 19-20), meaning

[2]Note that while our strategy appears similar to TTL caching, our model is different. Existing work on TTL caching often assumes a backend maintaining a data copy permanently [4], [5], [7], [18], [19], [21]. In contrast, we do not require any server to always keep a data copy. We consider a self-organized system in which at least one copy is maintained somewhere at any time.

that the copy in s_j is now beyond its intended storage period. Otherwise, s_j drops its copy (lines 21-23).

When server s_j needs to transfer the object to another server s_k to serve a request at s_k, if the copy in s_j is beyond the intended storage period (i.e., $K_j = 1$), s_j drops its copy after the transfer and clears its tag K_j (lines 14-17). Meanwhile, a new copy will be created in s_k so that the requirement of maintaining at least one copy is met (lines 8-9).

To facilitate competitive analysis, we shall refer to a data copy within the intended storage period as a *regular copy*, and a copy beyond that period as a *special copy*. Apparently, each regular copy immediately follows a request.

B. Allocation of Online Cost

We present a method to allocate the total cost produced by Algorithm 1 (referred to as the *online cost*) to each individual request. As illustrated in Figure 7, we categorize all requests into four types based on how they are served in Algorithm 1. For each request r_i served by a transfer, the regular copy after $r_{p(i)}$ must have expired before r_i arises. At the time of the transfer, if the copy in the source server is a regular copy, r_i is called a **Type-1** request; if the copy in the source server is a special copy, r_i is called a **Type-2** request. For each request r_i served by a local copy, when serving r_i, if the copy is a regular copy, r_i is called a **Type-3** request; if the copy is a special copy, r_i is called a **Type-4** request.

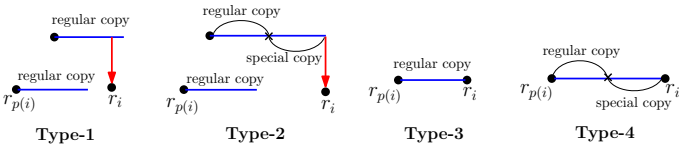


Fig. 7. Illustration of different request types in our online algorithm

In the example of Figure 8, r_1, r_2, r_3, r_5 and r_7 are **Type-1** requests, r_4 and r_6 are **Type-2** requests, r_8 is a **Type-3** request, and r_9 is a **Type-4** request.

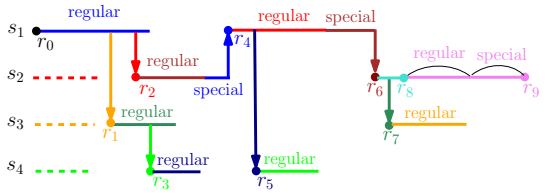


Fig. 8. An example of our online algorithm (each request and its allocated online cost are shown in the same color)

Note that since the intended storage periods of regular copies are randomly generated in Algorithm 1, given a request sequence, the type of each request r_i is not definite. Thus, we focus on studying the expected online cost.

The total online cost of Algorithm 1 consists of three parts: (1) the storage cost of regular copies; (2) the storage cost of special copies (if any); and (3) the cost of transfers (if any).

Storage cost of regular copies. By the algorithm definition, there is a regular copy after each request. We allocate the storage cost of the regular copy after a request to the succeeding request arising at the same server. That is, for each request r_i, we allocate to r_i the storage cost of the regular copy after $r_{p(i)}$ (see Figure 8). Since this regular copy cannot be longer than the period $(t_{p(i)}, t_i)$, its expected storage cost is

$$\int_0^{t_i - t_{p(i)}} t \cdot f(t)\,\mathrm{d}t + \int_{t_i - t_{p(i)}}^{\infty} (t_i - t_{p(i)}) \cdot f(t)\,\mathrm{d}t. \quad (1)$$

The regular copy after the last request at each server is treated specially. Given a request sequence $\langle r_1, r_2, \ldots, r_m \rangle$, 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 that exist after r_m, the copy expiring the latest would switch to a special copy and stay infinitely. We shall not account for the cost of the regular copy created after r_m and the special copy that stays infinitely. The rationale is that the storage periods of these two copies do not overlap and they are both entirely beyond r_m. They are considered to be in existence for maintaining at least one copy in the system beyond r_m. In an optimal offline strategy, no copy needs to be stored beyond r_m. Thus, we do not account for the cost of the aforesaid two copies by Algorithm 1.[3]

The storage costs of the remaining regular copies after the last requests at the servers are allocated to the first requests at the servers. Specifically, if there are n servers receiving requests, there will be $n-1$ remaining regular copies after the last requests at the servers (other than that after r_m). 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 aforesaid $n-1$ regular copies to these $n-1$ first requests (one regular copy for each request). That is, each first request is conceptually seen as infinitely away from its preceding request at the same server and thus must be served by a transfer. As illustrated in Figure 8, the first requests r_1, r_2 and r_3 are allocated with the storage costs of the regular copies after r_7, r_4 and r_5 respectively.

The expected storage cost of the regular copy after the last request at each server is $\int_0^{\infty} t \cdot f(t)\,\mathrm{d}t$, since there is no succeeding request at the same server. Recall that if r_i is the first request at a server, we define $t_{p(i)} = -\infty$. Thus, the storage cost of the regular copy allocated to each first request can also be represented by (1).

Storage cost of special copies. By the definition of Algorithm 1, a special copy occurs when some server s holds the only copy in the system at the expiration of its regular copy. In this case, the regular copy switches to become the special copy and is held until the next request arises. This implies that a special copy must be the only copy in the system. Thus, we have the following propositions.

[3]If 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.

Proposition 6. *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.*

Proposition 7. *A special copy does not cross the time of any request and can exist only between two consecutive requests.*

By Propositions 6 and 7, each special copy must be contained within one of the storage periods picked via backtracking in Section III-B. Thus, we allocate the storage cost of each special copy to the end sentinel request of the storage period that contains it.

According to this principle, for each end sentinel request $r_i \in Q_1$, we allocate to it the storage costs of all special copies that appear within $(t_{p(i)}, t_i)$ (see Figure 8, assuming $r_4, r_6, r_9 \in Q_1$). By Algorithm 1, there is a regular copy of randomized storage period up to $t_i - t_{p(i)}$ after $r_{p(i)}$. By Proposition 6, all special copies appearing within $(t_{p(i)}, t_i)$ do no overlap with this regular copy, and they are also mutually non-overlapping. Hence, the expected total storage cost of all special copies within $(t_{p(i)}, t_i)$ is bounded by

$$\int_0^{t_i - t_{p(i)}} (t_i - t_{p(i)} - t) \cdot f(t)\,\mathrm{d}t. \tag{2}$$

For each end sentinel request $r_i \in Q_2$, we allocate to it the storage costs of all special copies that appear within (t_{i-1}, t_i), whose total must be bounded by $t_i - t_{i-1}$. Note that if r_i is served by the regular copy after $r_{p(i)}$ (i.e., r_i is a **Type-3** request, which happens with probability $\int_{t_i - t_{p(i)}}^{\infty} f(t)\,\mathrm{d}t$), no special copy exists during $(t_{i-1}, t_i) \subset (t_{p(i)}, t_i)$ by Proposition 6. Hence, the expected total storage cost of all special copies within (t_{i-1}, t_i) is bounded by

$$(t_i - t_{i-1}) \cdot \int_0^{t_i - t_{p(i)}} f(t)\,\mathrm{d}t. \tag{3}$$

Transfer cost. If a request r_i is served by a transfer (i.e., r_i is a **Type-1/2** request), we allocate to r_i the cost λ of the transfer (see Figure 8). Since r_i is possibly served by a transfer only when the regular copy after $r_{p(i)}$ expires before r_i arises (which happens with probability $\int_0^{t_i - t_{p(i)}} f(t)\,\mathrm{d}t$), the expected transfer cost allocated to r_i is bounded by

$$\lambda \cdot \int_0^{t_i - t_{p(i)}} f(t)\,\mathrm{d}t. \tag{4}$$

Note that this is an upper bound because r_i can also be a **Type-4** request (served by a local special copy) if the regular copy after $r_{p(i)}$ expires before r_i arises.

We use $\textbf{Online}(r_i)$ to denote the online cost allocated to a request r_i and summarize the cost allocation by adding up (1) and (4) as well as (2) or (3).

Proposition 8. *The online cost allocated to a request r_i is bounded by*

- $\forall r_i \in Q_1$: $\textbf{\textit{Online}}(r_i) \le \int_0^{t_i - t_{p(i)}} (t_i - t_{p(i)} + \lambda) \cdot f(t)\,\mathrm{d}t + \int_{t_i - t_{p(i)}}^{\infty} (t_i - t_{p(i)}) \cdot f(t)\,\mathrm{d}t$;
- $\forall r_i \in Q_2$: $\textbf{\textit{Online}}(r_i) \le \int_0^{t_i - t_{p(i)}} (t_i - t_{i-1} + t + \lambda) \cdot f(t)\,\mathrm{d}t + \int_{t_i - t_{p(i)}}^{\infty} (t_i - t_{p(i)}) \cdot f(t)\,\mathrm{d}t$;
- $\forall r_i \in R_L \cup R_T$: $\textbf{\textit{Online}}(r_i) \le \int_0^{t_i - t_{p(i)}} (t + \lambda) \cdot f(t)\,\mathrm{d}t + \int_{t_i - t_{p(i)}}^{\infty} (t_i - t_{p(i)}) \cdot f(t)\,\mathrm{d}t$.

Since all the storage costs and transfer costs by Algorithm 1 have been allocated, the total online cost is bounded by $\sum_{1 \le i \le m} \textbf{Online}(r_i)$. Note that the dummy request r_0 is not allocated any cost.

V. Optimal $f(t)$ and Competitive Analysis

A. Formulation and Solution of An Optimization Problem

We aim to derive an optimal probability density function $f(t)$ to minimize the competitive ratio of our randomized online algorithm. Recall that the total optimal offline cost is $\sum_{1 \le i \le m} \textbf{OPT}(r_i)$ and the total online cost is at most $\sum_{1 \le i \le m} \textbf{Online}(r_i)$. Hence, the competitive ratio is bounded by $\sum_{1 \le i \le m} \textbf{Online}(r_i) / \sum_{1 \le i \le m} \textbf{OPT}(r_i)$. It is not easy to optimize this ratio directly. Our approach is to examine the ratio $\textbf{Online}(r_i)/\textbf{OPT}(r_i)$ for each individual request r_i separately and minimize the maximum such ratio among all requests. We shall present an example to demonstrate the tightness of our analysis and derivation in Section V-C.

To simplify presentation, we use $x := t_{p(i)} - t_i$ to denote the inter-request time between $r_{p(i)}$ and r_i. By Propositions 5 and 8, we have:

- $\forall r_i \in Q_1$: $\frac{\textbf{Online}(r_i)}{\textbf{OPT}(r_i)} \le \frac{\int_0^x (x+\lambda) \cdot f(t)\,\mathrm{d}t + \int_x^{\infty} x \cdot f(t)\,\mathrm{d}t}{x} = 1 + \frac{\lambda}{x} \int_0^x f(t)\,\mathrm{d}t$;
- $\forall r_i \in Q_2$: $\frac{\textbf{Online}(r_i)}{\textbf{OPT}(r_i)} \le \frac{\int_0^x (t_i - t_{i-1} + t + \lambda) \cdot f(t)\,\mathrm{d}t + \int_x^{\infty} x \cdot f(t)\,\mathrm{d}t}{(t_i - t_{i-1}) + \lambda}$, where $x > \lambda$;
- $\forall r_i \in R_L$: $\frac{\textbf{Online}(r_i)}{\textbf{OPT}(r_i)} \le \frac{\int_0^x (t+\lambda) \cdot f(t)\,\mathrm{d}t + \int_x^{\infty} x \cdot f(t)\,\mathrm{d}t}{x}$, where $0 < x \le \lambda$;
- $\forall r_i \in R_T$: $\frac{\textbf{Online}(r_i)}{\textbf{OPT}(r_i)} \le \frac{\int_0^x (t+\lambda) \cdot f(t)\,\mathrm{d}t + \int_x^{\infty} x \cdot f(t)\,\mathrm{d}t}{\lambda}$, where $x > \lambda$.

For each fixed $x > \lambda$, since $\frac{(t_i - t_{i-1}) \cdot \int_0^x f(t)\,\mathrm{d}t}{(t_i - t_{i-1})} \le 1 \le \frac{1}{\lambda} \cdot \left(\int_0^x (t+\lambda) \cdot f(t)\,\mathrm{d}t + \int_x^{\infty} x \cdot f(t)\,\mathrm{d}t \right)$, it follows that $\frac{\int_0^x (t_i - t_{i-1} + t + \lambda) \cdot f(t)\,\mathrm{d}t + \int_x^{\infty} x \cdot f(t)\,\mathrm{d}t}{(t_i - t_{i-1}) + \lambda} \le \frac{1}{\lambda} \cdot \left(\int_0^x (t+\lambda) \cdot f(t)\,\mathrm{d}t + \int_x^{\infty} x \cdot f(t)\,\mathrm{d}t \right)$. Thus, the above bound on the ratio for a request in Q_2 is no higher than the bound on the ratio for a request in R_T. Likewise, for each fixed $x \le \lambda$, since $\int_0^x (t+\lambda) \cdot f(t)\,\mathrm{d}t \le \int_0^x (x+\lambda) \cdot f(t)\,\mathrm{d}t$, the above bound on the ratio for a request in R_L is no higher than the bound on the ratio for a request in Q_1. Thus, we shall focus on minimizing the bounds on the ratios for requests in Q_1 and R_T only. Since the competitive ratio is defined as the worst-case ratio over all problem instances, we shall consider all possible values of inter-request time x. Hence, we formulate the following optimization problem for deciding the probability density function $f(t)$.

$$\underset{f}{\text{minimize}}\ C \text{ subject to}$$

$$\forall x > 0,\ 1 + \frac{\lambda}{x} \int_0^x f(t)\,\mathrm{d}t \le C, \tag{5}$$

$$\forall x > \lambda,\ \frac{1}{\lambda} \left(\int_0^x (t+\lambda) \cdot f(t)\,\mathrm{d}t + x \cdot \int_x^{\infty} f(t)\,\mathrm{d}t \right) \le C, \tag{6}$$

$$\int_0^{\infty} f(t)\,\mathrm{d}t = 1, \tag{7}$$

$$\forall t \geq 0, f(t) \geq 0. \tag{8}$$

In practice, it is not sensible to set an intended storage period with infinite length. Thus, we shall restrict $f(t)$ to a finite range of $[0, B]$, i.e., $\int_0^B f(t)\,\mathrm{d}t = 1$, where $B > 0$ is a variable to be decided. In addition, the left hand side of (6) is non-decreasing with respect to x because its first-order derivative is $f(x) + \frac{1}{\lambda}\int_x^B f(t)\,\mathrm{d}t \geq 0$. Hence, by taking $x = B$, (6) can be reduced to $\frac{\int_0^B (t+\lambda)\cdot f(t)\,\mathrm{d}t}{\lambda} = 1 + \frac{\int_0^B t\cdot f(t)\,\mathrm{d}t}{\lambda} \leq C$. Then, the optimization problem can be rewritten as

$$\begin{aligned}
&\underset{B,f}{\text{minimize}}\ C \\
&\text{subject to } \forall 0 < x \leq B,\ 1 + \frac{\lambda}{x}\int_0^x f(t)\,\mathrm{d}t \leq C, && (9)\\
&\qquad 1 + \frac{\int_0^B t\cdot f(t)\,\mathrm{d}t}{\lambda} \leq C, && (10)\\
&\qquad \int_0^B f(t)\,\mathrm{d}t = 1, && (11)\\
&\qquad \forall t \geq 0,\ f(t) \geq 0. && (12)
\end{aligned}$$

We solve this optimization problem in two steps. First, for each fixed B, we derive an optimal $f(t)$ to minimize C. Then, we solve the problem by deriving the optimal value of B.

To derive an optimal $f(t)$ given B, we assume intuitively that the left hand side of constraint (9) is a constant: $1 + \frac{\lambda}{x}\int_0^x f(t)\,\mathrm{d}t = C'$ for all $0 < x \leq B$, where C' is a constant.[4] Taking the second-order derivative of both sides with respect to x, we have $f'(x) = 0$, which implies that $f(x)$ is a constant. By constraint (11), we have $f(t) = \frac{1}{B}$ for all $0 < t \leq B$.

Substituting $f(t) = \frac{1}{B}$ into constraint (9), we have $C \geq 1 + \frac{\lambda}{B}$. We also substitute this $f(t)$ into constraint (10) and get $C \geq 1 + \frac{B}{2\lambda}$. Hence, finding the optimal B to minimize C is equivalent to minimizing $\max\left\{1 + \frac{\lambda}{B}, 1 + \frac{B}{2\lambda}\right\}$.

It is easy to see that the optimal $B = \sqrt{2}\lambda$ such that $1 + \frac{\lambda}{B} = 1 + \frac{B}{2\lambda}$. Hence, the optimal probability density function $f(t)$ is given by $f(t) = \frac{1}{\sqrt{2}\lambda}$ for all $0 \leq t \leq \sqrt{2}\lambda$. When taking this $f(t)$ and B, the corresponding objective value $C = 1 + \frac{\lambda}{B} = 1 + \frac{B}{2\lambda} = 1 + \frac{\sqrt{2}}{2}$, i.e., our randomized online algorithm has a competitive ratio of $1 + \frac{\sqrt{2}}{2}$.

B. Proof of Optimality

Now we prove that the probability density function $f(t)$ derived above is an optimal solution, i.e., the minimum possible value of C is $1 + \frac{\sqrt{2}}{2}$.

We prove it by contradiction. Assume that there exists another probability density function $g(x)$ defined over a finite range of $[0, B']$, which makes the objective value C_g strictly lower than $1 + \frac{\sqrt{2}}{2}$ in the optimization problem, i.e., $C_g = \max\limits_{x\in[0,B']}\left\{1 + \frac{\lambda}{x}\int_0^x g(t)\,\mathrm{d}t, 1 + \frac{\int_0^{B'} t\cdot g(t)\,\mathrm{d}t}{\lambda}\right\} < 1 + \frac{\sqrt{2}}{2}$.

[4]To verify the correctness of this approach, we shall formally prove that the $f(t)$ derived is optimal in Section V-B.

If $B' \leq \sqrt{2}\lambda$, when taking $x = B'$, we have $1 + \frac{\lambda}{B'}\int_0^{B'} f(t)\,\mathrm{d}t = 1 + \frac{\lambda}{B'} \geq 1 + \frac{\sqrt{2}}{2}$, which implies that $C_g \geq 1 + \frac{\sqrt{2}}{2}$. Hence, it must hold that $B' > \sqrt{2}\lambda$.

Since $C_g < 1 + \frac{\sqrt{2}}{2}$, we must have $1 + \frac{\lambda}{x}\cdot\int_0^x g(t)\,\mathrm{d}t < 1 + \frac{\lambda}{x}\cdot\int_0^x f(t)\,\mathrm{d}t = 1 + \frac{\sqrt{2}}{2}$ for all $0 < x \leq \sqrt{2}\lambda$, which is equivalent to $\int_0^x g(t)\,\mathrm{d}t < \int_0^x f(t)\,\mathrm{d}t$ for all $0 < x \leq \sqrt{2}\lambda$.

Let $I = \{(p_1, q_1), (p_2, q_2), \ldots, (p_z, q_z)\}$ denote the set of all maximal intervals within $[0, \sqrt{2}\lambda]$ such that $f(t) > g(t)$ or $f(t) < g(t)$ holds consistently within each $(p_i, q_i) \in I$, where $0 \leq p_1 < q_1 \leq p_2 < q_2 \leq \cdots \leq p_z < q_z \leq \sqrt{2}\lambda$. We further divide I into two subsets I^+ and I^-, where $f(t) > g(t)$ within each $(p_i, q_i) \in I^+$ and $f(t) < g(t)$ within each $(p_i, q_i) \in I^-$.

By the definition of I, we have:

Observation 1. *In the period of* $[0, \sqrt{2}\lambda]\backslash I$, $f(t) = g(t)$.

Observation 2. $\sum\limits_{i=1}^{z}\int_{p_i}^{q_i}(f(t) - g(t))\,\mathrm{d}t = \int_{\sqrt{2}\lambda}^{B'} g(t)\,\mathrm{d}t$.

Proof. Since $\int_0^{\sqrt{2}\lambda} f(t)\,\mathrm{d}t = \int_0^{B'} g(t)\,\mathrm{d}t = 1$, by Observation 1, we have $\int_0^{\sqrt{2}\lambda} f(t)\,\mathrm{d}t - \int_0^{B'} g(t)\,\mathrm{d}t = \sum_{i=1}^{z}\int_{p_i}^{q_i}(f(t) - g(t))\,\mathrm{d}t + \int_{\sqrt{2}\lambda}^{B'}(0 - g(t))\,\mathrm{d}t = 0$. □

Observation 3. *For each* $k \in \{1, 2, \ldots, z\}$, *it holds that* $\sum_{i=1}^{k}\int_{p_i}^{q_i}(f(t) - g(t))\,\mathrm{d}t > 0$.

Proof. This can be proved by contradiction. If $\sum_{i=1}^{k}\int_{p_i}^{q_i}(f(t) - g(t))\,\mathrm{d}t \leq 0$, it implies that $\int_0^{q_k} f(t)\,\mathrm{d}t \leq \int_0^{q_k} g(t)\,\mathrm{d}t$ by Observation 1, which contradicts that $\int_0^x g(t)\,\mathrm{d}t < \int_0^x f(t)\,\mathrm{d}t$ for all $0 < x \leq \sqrt{2}\lambda$. □

We would like to show that $\int_0^{B'} t\cdot g(t)\,\mathrm{d}t > \int_0^{\sqrt{2}\lambda} t\cdot f(t)\,\mathrm{d}t$ to derive a contradiction, that is to prove that $\int_0^{\sqrt{2}\lambda} t\cdot(f(t) - g(t))\,\mathrm{d}t < \int_{\sqrt{2}\lambda}^{B'} t\cdot g(t)\,\mathrm{d}t$, or equivalently $\sum_{j=1}^{z}\int_{p_j}^{q_j} t\cdot(f(t) - g(t))\,\mathrm{d}t < \int_{\sqrt{2}\lambda}^{B'} t\cdot g(t)\,\mathrm{d}t$. For each interval $(p_i, q_i) \in I^+$, we define $u_i := q_i$; for each $(p_i, q_i) \in I^-$, we define $u_i := p_i$. Note that $0 \leq u_1 \leq u_2 \leq \cdots \leq u_z \leq \sqrt{2}\lambda$, as all the intervals in I are indexed in ascending order. With these definitions, we have

$$\sum_{i=1}^{z}\int_{p_i}^{q_i} t\cdot(f(t) - g(t))\,\mathrm{d}t < \sum_{i=1}^{z} u_i\cdot\int_{p_i}^{q_i}(f(t) - g(t))\,\mathrm{d}t. \tag{13}$$

We can further show by induction that the right hand side of (13) is bounded by $u_z\cdot\sum_{i=1}^{z}\int_{p_i}^{q_i}(f(t) - g(t))\,\mathrm{d}t$.

- The base case is obvious: $u_1\cdot\int_{p_1}^{q_1}(f(t) - g(t))\,\mathrm{d}t \leq u_1\cdot\int_{p_1}^{q_1}(f(t) - g(t))\,\mathrm{d}t$.
- Suppose for some $1 < k < z$, we have

$$\sum_{i=1}^{k} u_i\cdot\int_{p_i}^{q_i}(f(t) - g(t))\,\mathrm{d}t \leq u_k\cdot\sum_{i=1}^{k}\int_{p_i}^{q_i}(f(t) - g(t))\,\mathrm{d}t.$$

- Then, for $k + 1$, we have

$$\sum_{i=1}^{k+1} u_i\cdot\int_{p_i}^{q_i}(f(t) - g(t))\,\mathrm{d}t$$

$$\le u_k \cdot \sum_{i=1}^{k} \int_{p_i}^{q_i} (f(t)-g(t))\,\mathrm{d}t + u_{k+1} \cdot \int_{p_{k+1}}^{q_{k+1}} (f(t)-g(t))\,\mathrm{d}t$$

(by induction hypothesis)

$$\le u_{k+1} \cdot \sum_{i=1}^{k+1} \int_{p_i}^{q_i} (f(t)-g(t))\ \mathrm{d}t$$

(by Observation 3 and $u_k \le u_{k+1}$).

- Therefore, we have

$$\sum_{i=1}^{z} u_i \cdot \int_{p_i}^{q_i} (f(t)-g(t))\,\mathrm{d}t \le u_z \cdot \sum_{i=1}^{z} \int_{p_i}^{q_i} (f(t)-g(t))\,\mathrm{d}t. \quad (14)$$

By (13) and (14), we have

$$\sum_{i=1}^{z} \int_{p_i}^{q_i} t \cdot (f(t)-g(t))\,\mathrm{d}t \le u_z \cdot \sum_{i=1}^{z} \int_{p_i}^{q_i} (f(t)-g(t))\,\mathrm{d}t$$

$$= u_z \cdot \int_{\sqrt{2}\lambda}^{B'} g(t)\,\mathrm{d}t \quad \text{(by Observation 2)}$$

$$\le \sqrt{2}\lambda \cdot \int_{\sqrt{2}\lambda}^{B'} g(t)\,\mathrm{d}t \quad (\text{since } u_z \le \sqrt{2}\lambda)$$

$$\le \int_{\sqrt{2}\lambda}^{B'} t \cdot g(t)\,\mathrm{d}t \quad (\text{since } \sqrt{2}\lambda < B').$$

By the above inequality and Observation 1, we have $\int_0^{B'} t \cdot g(t)\,\mathrm{d}t \ge \int_0^{\sqrt{2}\lambda} t \cdot f(t)\,\mathrm{d}t$. This implies that $1 + \frac{\int_0^{B'} t \cdot g(t)\,\mathrm{d}t}{\lambda} \ge 1 + \frac{\int_0^{\sqrt{2}\lambda} t \cdot f(t)\,\mathrm{d}t}{\lambda} = 1 + \frac{\sqrt{2}}{2}$ and hence, $C_g \ge 1 + \frac{\sqrt{2}}{2}$, which leads to a contradiction.

To conclude, the probability density function $f(t) = \frac{1}{\sqrt{2}\lambda}$ for all $0 \le t \le \sqrt{2}\lambda$ is an optimal solution, which gives the lowest possible value of $C = 1 + \frac{\sqrt{2}}{2}$ in the optimization problem of Section V-A.

C. A Tight Example

We construct an example to show that the competitive ratio $1+\frac{\sqrt{2}}{2}$ of Algorithm 1 under the $f(t)$ derived is tight. Consider n servers $s_1, s_2, \ldots, s_n$ and a sequence of $n+1$ requests $r_1, r_2, \ldots, r_{n+1}$. Requests r_1 and r_{n+1} arise in server s_1 at times 0 and $2\sqrt{2}\lambda + \epsilon$ respectively, where $\epsilon > 0$ is a small value. Requests $r_2, r_3, \ldots, r_n$ arise in servers $s_2, s_3, \ldots, s_n$ respectively within a short period of ϵ after time $\sqrt{2}\lambda$.

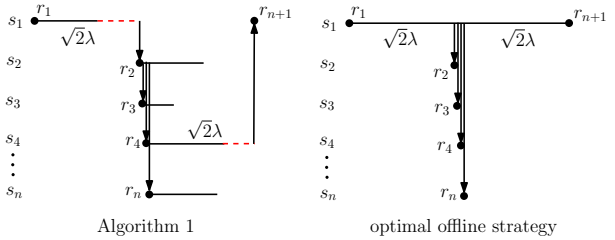


Fig. 9. A tight example

As shown in Figure 9, by Algorithm 1, s_1 keeps a regular copy after r_1 for a randomized period following $f(t)$. This regular copy must expire before r_2 arises, because the longest possible storage period of the regular copy is $\sqrt{2}\lambda$. Since s_1 keeps the only copy when the regular copy expires, it continues to keep the copy until r_2 arises, and serves r_2 by a transfer to s_2. Then, s_1 drops the copy, and s_2 keeps a regular copy for a randomized period after r_2.

Right after a copy is created in s_2, a bunch of requests $r_3, r_4, \ldots, r_n$ arise at $s_3, s_4, \ldots, s_n$ within a short period of ϵ, so they are all served by transfers from s_2.[5] Then, all these servers keep regular copies for randomized periods. Hence, the online cost produced till r_n is at least $\sqrt{2}\lambda + (n-1)\cdot\lambda$.

All $n-1$ regular copies in $s_2, s_3, \ldots, s_n$ will expire before r_{n+1} arises. The copy that expires the latest (e.g., the regular copy in s_4 in Figure 9) will switch to a special copy and be kept until r_{n+1} arises to serve r_{n+1} by a transfer. Thus, the storage cost in the server with the latest expiring regular copy is at least $\sqrt{2}\lambda$. The total storage cost of all the other regular copies is $\sum_{i=2}^{n} X_i - \max\{X_2, X_3, \ldots, X_n\}$, where X_i denotes the intended storage period of the regular copy in s_i. Thus, the total storage cost in $s_2, s_3, \ldots, s_n$ is $\sum_{i=2}^{n} X_i - \max\{X_2, X_3, \ldots, X_n\} + \sqrt{2}\lambda$.

Following the $f(t)$ derived, the expectation of each X_i is $E(X_i) = \int_0^{\sqrt{2}\lambda} \frac{t}{\sqrt{2}\lambda}\,\mathrm{d}t = \frac{\sqrt{2}}{2}\lambda$.

Since the random variables $X_2, X_3, \ldots, X_n$ are independent and identically distributed, the cumulative distribution function of $Y := \max\{X_2, X_3, \ldots, X_n\}$ is

$$\begin{aligned} F_Y(y) &= P(\max\{X_2, X_3, \ldots, X_n\} \le y) \\ &= P(X_2 \le y) \cdot P(X_3 \le y) \cdot \cdots \cdot P(X_n \le y) \\ &= \left(\int_0^y \frac{1}{\sqrt{2}\lambda}\,\mathrm{d}t\right)^{n-1} = \left(\frac{y}{\sqrt{2}\lambda}\right)^{n-1}. \end{aligned}$$

Therefore, the probability density function of Y is $f_Y(y) = F_Y'(y) = \frac{(n-1)\cdot y^{n-2}}{(\sqrt{2}\lambda)^{n-1}}$. Hence, the expectation of Y is

$$E(Y) = \int_0^{\sqrt{2}\lambda} y \cdot f_Y(y)\,\mathrm{d}y = \frac{n-1}{n} \cdot \sqrt{2}\lambda.$$

Thus, the expected total storage cost in $s_2, s_3, \ldots, s_n$ is

$$E\left(\sum_{i=2}^{n} X_i - \max\{X_2, X_3, \ldots, X_n\} + \sqrt{2}\lambda\right)$$

$$= (n-1)\cdot\frac{\sqrt{2}}{2}\lambda - \frac{n-1}{n}\cdot\sqrt{2}\lambda + \sqrt{2}\lambda.$$

Finally, the transfer cost to serve r_{n+1} is λ. So, the expected total online cost is $2\sqrt{2}\lambda + n\cdot\lambda + (n-1)\cdot\frac{\sqrt{2}}{2}\lambda - \frac{n-1}{n}\cdot\sqrt{2}\lambda$.

In the optimal offline strategy, s_1 keeps its copy until serving r_{n+1} locally, and all the requests $r_2, r_3, \ldots, r_n$ are served by transfers. Thus, the total optimal offline cost is at most $2\sqrt{2}\lambda + \epsilon + (n-1)\cdot\lambda$. Hence, the online-to-optimal cost ratio of this example is at least

$$\frac{2\sqrt{2}\lambda + n\cdot\lambda + (n-1)\cdot\frac{\sqrt{2}}{2}\lambda - \frac{n-1}{n}\cdot\sqrt{2}\lambda}{2\sqrt{2}\lambda + \epsilon + (n-1)\cdot\lambda}$$

[5] These requests arise in close time proximity to r_2, so we can assume that they all arise before the regular copy in s_2 expires.

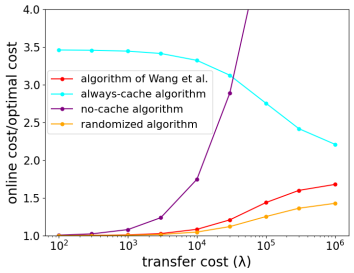


Fig. 10. Uniform request distribution

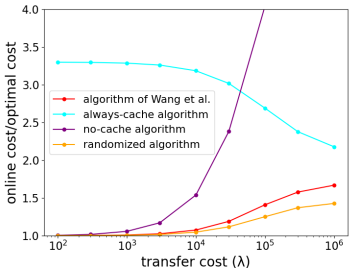


Fig. 11. Zipf request distribution with $\beta = 1$

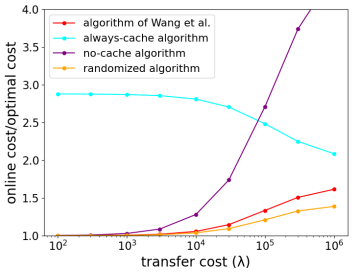


Fig. 12. Zipf request distribution with $\beta = 2$

$$= \frac{(1+\frac{1}{n})\cdot\sqrt{2}\lambda + n\cdot\lambda + (n-1)\cdot\frac{\sqrt{2}}{2}\lambda}{2\sqrt{2}\lambda + \epsilon + (n-1)\cdot\lambda}$$
$$\to 1 + \frac{\sqrt{2}}{2} \quad (\text{as } \epsilon \to 0 \text{ and } n \to \infty).$$

Note that this example can be extended to a request sequence of arbitrary length by repeating $r_1, r_2, \ldots, r_n$ (i.e., treating r_{n+1} as r_1 of a new cycle). Hence, the competitive ratio $1 + \frac{\sqrt{2}}{2}$ of our algorithm is also asymptotic.

VI. Experimental Evaluation

We conduct simulation experiments to evaluate our proposed algorithm, using the object storage traces provided by IBM [17]. Similar performance trends are observed for the results of different traces. In this section, we present the results of the trace named "IBM Object Store Trace Number 066". The trace records over 700,000 read requests made to 5031 different objects in a cloud-based storage service over 7 days in the year of 2019. The most popular object has 1548 requests, while the least popular object has 1 request only. We treat one second as one time unit. To include the storage costs of all regular copies by our algorithm, we simulate the time span of 7 days plus $\sqrt{2}\lambda$ time units, since the longest possible storage period of a regular copy is $\sqrt{2}\lambda$. The requests of each object are distributed over 10 different servers at random following a uniform distribution or a Zipf distribution (where each request is assigned to server s_i with probability $i^{-\beta} / \sum_{j=1}^{10} j^{-\beta}$ and β is a parameter). Initially, one copy of each object is placed in server s_1. We compare our randomized online algorithm with the following algorithms.

- Algorithm of Wang *et al.*: It is a state-of-the-art deterministic online algorithm from [26], which is 2-competitive. The main idea is for each server to hold a copy after every local request for some period over which the storage cost equals the transfer cost. If there is only one copy in the system and there is no request for a sufficiently long period, the copy is moved to the server with the lowest storage cost rate to save cost.
- Always-cache algorithm: For every object, a copy is created in each server upon its first local request. The copy is never deleted afterwards, so that all subsequent requests arising at the same server are served locally.
- No-cache algorithm: A copy of each object is always kept in server s_1. Other servers never keep any copy of objects, so their local requests are all served by transfers from s_1.

We experiment with different settings of the transfer cost λ from 100 to 1,000,000, and record the total online cost of each algorithm over the aforesaid time span. For our randomized algorithm, we run it for 20 times and compute the average online cost. The 95% confidence interval of the experimental results was calculated to be within 1 percent of the mean. We normalize the online costs of different algorithms by the optimal offline cost derived from dynamic programming [25].

Figures 10 to 12 show the results. The online-to-optimal cost ratio of the no-cache algorithm rises rapidly with increasing transfer cost, because all the requests arising at servers other than s_1 are served by transfers. In contrast, by creating an object copy permanently whenever the object is accessed, the always-cache algorithm has its cost ratio dropping with increasing transfer cost. These two baselines have much higher cost ratios than Wang *et al.*'s algorithm and our algorithm. Comparing the latter two, Wang *et al.*'s deterministic algorithm has a higher online-to-optimal cost ratio than our randomized algorithm for all settings of transfer cost. The main reason is that in Wang *et al.*'s algorithm, each server keeps an object copy for at least λ time after serving a local request. The associated storage cost of λ is greater than the expected cost $\frac{\sqrt{2}}{2}\lambda$ of a regular copy in our algorithm. The relative performance of the algorithms is similar for uniform and Zipf request distributions. In summary, our proposed algorithm outperforms all the other algorithms compared.

VII. Conclusion

In this paper, we have proposed a randomized online algorithm for caching data to optimize the service cost of distributed data access. Our algorithm can achieve a competitive ratio of $1 + \frac{\sqrt{2}}{2}$, which is better than the state-of-the-art deterministic algorithm. We also show that this ratio is tight and asymptotic for our algorithm by presenting an example. Experimental evaluations using real data access traces confirm that our algorithm outperforms deterministic algorithms.

Acknowledgment

This work is supported by the Ministry of Education, Singapore, under its AcRF Tier 2 (Awards MOE-T2EP20121-0005 and MOE-T2EP20122-0007) and Tier 1 (Award RG23/23).

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] Djillali Boukhelef, Jalil Boukhobza, Kamel Boukhalfa, Hamza Ouarnoughi, and Laurent Lemarchand. Optimizing the cost of DBaaS object placement in hybrid storage systems. *Future Generation Computer Systems*, 93:176–187, 2019.

[4] Edith Cohen and Haim Kaplan. Aging through cascaded caches: Performance issues in the distribution of web content. In *Proceedings of the 2001 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, pages 41–53, 2001.

[5] Nicaise Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. Performance evaluation of hierarchical TTL-based cache networks. *Computer Networks*, 65:212–231, 2014.

[6] Jiehui Ju, Jiyi Wu, Jianqing Fu, Zhijie Lin, and Jianlin Zhang. A survey on cloud storage. *Journal of Computers*, 6(8):1764–1771, 2011.

[7] Jaeyeon Jung, Arthur W. Berger, and Hari Balakrishnan. Modeling TTL-based Internet caches. In *Proceedings of the 22nd IEEE Conference on Computer Communications (INFOCOM)*, pages 417–426, 2003.

[8] Konstantinos Kalpakis, Koustuv Dasgupta, and Ouri Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):628–637, 2001.

[9] Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgement and other stories about e/(e-1). In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 502–509, 2001.

[10] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive Randomized Algorithms for Nonuniform Problems. *Algorithmica*, 11(6):542–571, 1994.

[11] Ali Khanafer, Murali Kodialam, and Krishna P.N. Puttaswamy. The constrained ski-rental problem and its application to online cloud cost optimization. In *Proceedings of the 32nd IEEE Conference on Computer Communications (INFOCOM)*, pages 1492–1500, 2013.

[12] Mingyu Liu, Li Pan, and Shijun Liu. Cost optimization for cloud storage from user perspectives: Recent advances, taxonomy, and survey. *ACM Computing Surveys*, 55(13s):1–37, 2023.

[13] Ying Liu, Qiang He, Dequan Zheng, Xiaoyu Xia, Feifei Chen, and Bin Zhang. Data caching optimization in the edge computing environment. *IEEE Transactions on Services Computing*, 15(4):2074–2085, 2020.

[14] Yaser Mansouri and Abdelkarim Erradi. Cost optimization algorithms for hot and cool tiers cloud storage services. In *Proceedings of IEEE International Conference on Cloud Computing*, pages 622–629, 2018.

[15] 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.

[16] 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.

[17] Effi Ofer, Danny Harnik, and Ronen Kat. Object storage traces: A treasure trove of information for optimizing cloud workloads, 2021. https://www.ibm.com/cloud/blog/object-storage-traces, last accessed on 2023-07-20.

[18] Xueyan Tang and Samuel T. Chanson. Analysis of replica placement under expiration-based consistency management. *IEEE Transactions on Parallel and Distributed Systems*, 17(11):1253–1263, 2006.

[19] Xueyan Tang, Huicheng Chi, and Samuel T. Chanson. Optimal replica placement under TTL-based consistency. *IEEE Transactions on Parallel and Distributed Systems*, 18(3):351–363, 2007.

[20] Xueyan Tang and Jianliang Xu. QoS-aware replica placement for content distribution. *IEEE Transactions on Parallel and Distributed Systems*, 16(10):921–932, 2005.

[21] Xueyan Tang, Jianliang Xu, and Wang-Chien Lee. Analysis of TTL-based consistency in unstructured peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(12):1683–1694, 2008.

[22] Petroc Taylor. Amount of data created, consumed, and stored 2010-2020, with forecasts to 2025, 2022. https://www.statista.com/statistics/871513/worldwide-data-created/#statisticContainer, last accessed on 2023-06-16.

[23] Vijay V. Vazirani. *Approximation Algorithms*. Springer Science & Business Media, 2013.

[24] 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.

[25] Yang Wang, Shuibing He, Xiaopeng Fan, Chengzhong Xu, and Xian-He Sun. On cost-driven collaborative data caching: A new model approach. *IEEE Transactions on Parallel and Distributed Systems*, 30(3):662–676, 2018.

[26] 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.