Latency Mitigation: Archive Storage Systems With SLAs At Minimal Cost

BENJAMIN DISCHINGER and DAVID H.C. DU, University of Minnesota

We present *latency mitigation*, an allocation and transfer strategy for archive storage systems that mitigates the access latency of low-cost devices in order to meet individual object latency and transfer rate SLAs for sequential read workloads. It achieves this balance between performance and cost by spreading individual objects across multiple device pools and using a pipelined data transfer strategy. The goal is to help address the high latency barrier so more organizations can adopt cost saving storage. We construct the latency mitigation algorithm for three device pools by analyzing a generalized storage system and demonstrate its feasibility using trace data from a real-world large scale archive.

CCS Concepts: • **Information systems → Digital libraries and archives**; *Hierarchical storage management*; • **Social and professional topics → Pricing and resource allocation**.

Additional Key Words and Phrases: Object Storage, Allocation, Tiered Storage, Archive

ACM Reference Format:
Benjamin Dischinger and David H.C. Du. 2020. Latency Mitigation: Archive Storage Systems With SLAs At Minimal Cost. *ACM Trans. Storage* 1, 1, Article 1 (January 2020), 23 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Modern storage systems must continuously grow to contain the ever expanding mountain of data. Individual science experiments can now generate more data in a day than what was imagined to store world-wide only a few years ago [23]. Sources of data constantly emerge as consumers acquire more devices and organizations deploy newer data driven technologies [9], and the deluge is only accelerating [20].

Adding to the mountain, because of the big data phenomenon, data can't be deleted. Data has unknown future value because unforeseen methods to extract value may exist [16]. Not only does the data need to be saved, but it is typically stored in online storage where it can be available for real time analytic queries providing potential value and profit. This requires a large number of computing systems and storage devices consuming power and space, costing tremendous sums of money.

As data ages without being used, the likelihood of extracting future value decreases. We still want to preserve the data for its possible value, but eventually it makes sense to migrate infrequently used data to lower cost storage [21]. Fortunately there's a strong ecosystem of storage device types varying in performance and price [11] that can be used to place data on the most appropriate device to maximize performance or minimize cost.

The goal of archive storage systems is to provide massive capacity for long term data storage, while maintaining accessibility at the lowest possible cost. To help meet this goal, low cost tape storage is still used by large multi-petabyte archive storage systems. The European Centre for

Authors' address: Benjamin Dischinger; David H.C. Du, University of Minnesota.

1553-3077/2020/1-ART1 $15.00
https://doi.org/0000001.0000001

Medium-Range Weather Modeling (ECMWF), who manage an archive with a capacity of over 100 PB, found that 30% of ECMWF requests to tape have a latency to first byte of more than three minutes, and that only 27% of all files were ever requested from tape during the study [13].

There are two main factors why we believe it is possible to lower the adoption barrier to tape storage by bridging the latency gap between low and high latency storage devices. The first is that as sensor and network technology have improved, individual data objects have increased in size more than device bandwidth has increased, therefore resulting in longer transfer times. The second factor is the recent emergence of cold storage devices (CSDs) [**?**] that provide a latency and cost middle ground between disk and tape devices. Since objects in archive storage are typically read sequentially, the combination of the larger object sizes and CSDs can be used to hide tape latency to store a significant percentage of data on tape, therefore reducing costs.

We ask the following question: *How should data objects be allocated in an archive storage system containing multiple storage pools with differing performance characteristics, in order to meet individual latency and rate service requirements at the least possible cost?*

To answer this question we analyze a storage system containing three device pools, each pool having an expected access delay, bandwidth and cost. Based on this analysis we propose a novel data placement algorithm called *latency mitigation* that spreads individual object data across multiple device pools in order to achieve simultaneous goals of acceptable latency and transfer rate at minimal cost.

Illustrative Example. A user wishes to store a 1 GB object and be able to access it sometime in the future. They will think the service is poor quality if no data is received within 1 second or if the transfer rate is lower than 10 MB/s. Let's assume a system contains three storage pools: disk, cold storage, and tape, with storage costs of $.04, $.02, and $.01 per GB respectively. Also assume that the average latency for disk is negligible, for cold storage it is 10 seconds, and for tape 60 seconds. The transfer bandwidth of each device is greater than 10 MB/s. Figure 1 shows the different allocation choices graphically.

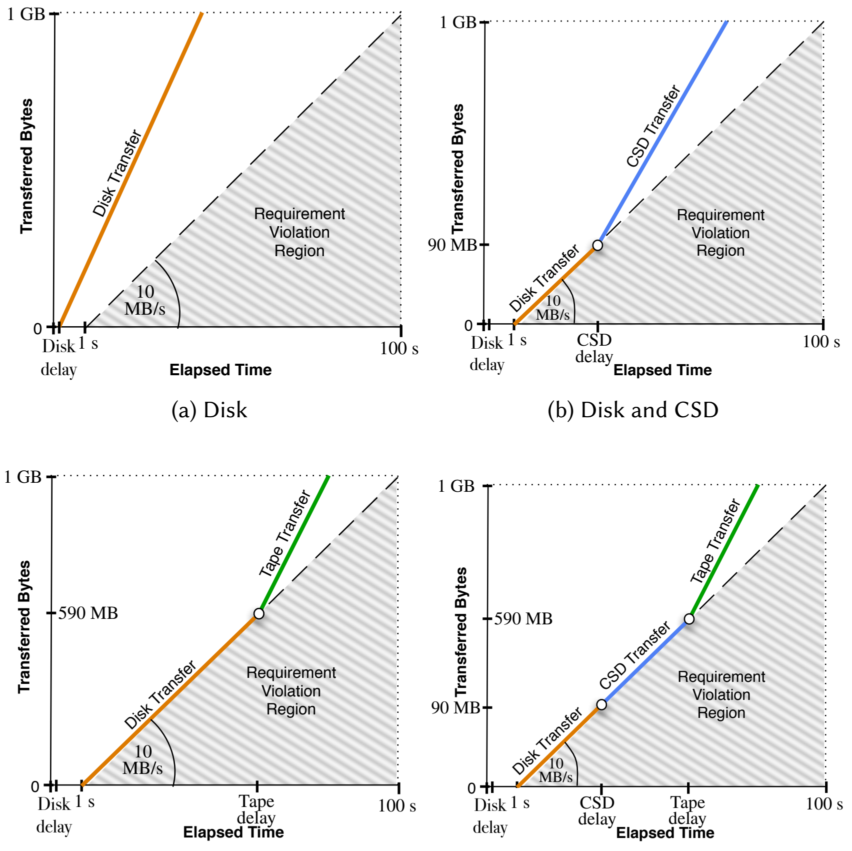
First we notice it is not possible put the object entirely on either cold storage or tape without violating the user's latency expectations. We could place the entire object on disk (Figure 1a) and this would meet the user requirements, but it would be the most expensive solution at $0.04.

Another possibility is to use the disk pool to mitigate the latency of the second pool by transferring the first portion of the file while waiting for data from the second pool (Figure 1b). Doing this for cold storage would require to mitigate a 10 second delay, but since the user accepts a 1 second latency, we need to transfer data from disk at 10 MB/s for at least 9 seconds starting after 1 second. Therefore 90 MB of storage is required on disk and 910 MB on cold storage for a total cost of $0.0218. Similarly doing this for tape (Figure 1c) would require to mitigate 59 seconds, or 590 MB of disk storage and 410 MB on tape, a cost of $0.0277.

The last possibility is to use all three pools; the disk transfer to mitigate cold storage delay, and the cold storage transfer to mitigate tape delay (Figure 1d). Again disk would need to mitigate 9 seconds of cold storage delay, and cold storage would have to mitigate an additional 50 seconds of tape delay. So in this case 90 MB would be on disk, 500 MB on cold storage and 410 MB on tape for a total cost of $0.0177.

For this example we see that using all three pools is the least expensive option resulting in a 19% cost savings over using just disk and cold storage. By using the lower latency pools to mitigate the high latency of tape we are able to still meet the user expectations while reducing cost.

Paper Organization. In the next section we discuss storage devices used today and the motivation for latency mitigation (§2). Next we describe the latency mitigation concept (§3), the system model (§3.2), the notation used throughout the paper (§3.3), and present the latency mitigation algorithm



(a) Disk

(b) Disk and CSD

(c) Disk and Tape

(d) Disk, CSD, and Tape

Fig. 1. Allocations across disk, cold storage device (CSD) and tape with latency and rate requirements of 1 sec and 10 MB/s.

for a generalized storage system with three device pools (§4). Finally we evaluate the feasibility of latency mitigation (§5) using real-world system trace data from the European Centre for Medium-Range Weather Modeling (ECMWF). We conclude with a discussion of related work (§6), possible applications, and future work (§7).

2 BACKGROUND

In this section we provide background for the endless growing need for storage and the storage devices in use today.

2.1 Data Explosion

The total amount of electronic data generated worldwide continues to grow exponentially as organizations seek ways to reduce possible cost while still meeting performance requirements. By 2025, it is estimated there will be more than 165 zettabytes of data generated per year, created by a proliferation of smart devices, high resolution data formats, embedded devices, and big data analytics [20]. In order to store just 5% of the generated data, more storage capacity across all types

	Device Properties	Expected Latency	Rate (MB/s)	Cost ($/GB)	Capacity (TB)
LTO-8 Tape [14]	Removable, sequential, mechanical	60 s	360	.011	12
Blu-ray BD-R Disc [24]	Removable, random, mechanical	180 ms	54	.054	.1
Archive SMR HDD [26]	Fixed, sequential writes, random reads, mechanical	4 ms	255	.021	15
Enterprise HDD [27]	Fixed, random, mechanical	4 ms	267	.026	14
SATA SSD [22]	Fixed, random, solid state	100 µs	550	.140	4

Table 1. Example individual storage devices with costs from 1/4/2020. Tape and Blu-ray cost is for media only, excluding hardware. Shingled Magnetic Recording (SMR) drives will experience latency variation due to internal operations, 4 ms is an average seek time. Due to unavailable public pricing, SMR drive cost is based on 25% capacity increase over conventional drive mentioned in data sheet [26].

of media will have to be manufactured in the year 2025 alone than in previous ten years prior to 2018 combined [20]. Given the enormous demand expected in the future, storage systems will need to be designed to use multiple types of devices with varying cost and performance profiles to have a chance at being able to store this data economically.

2.2 Storage Devices

The main differentiation between storage devices are latency (response time), rate (transfer speed), cost, and capacity. Some high-level properties that determine these metrics are whether a device is removable or fixed, sequential or random, and mechanical or solid state [1]. Table 1 shows a selection of individual storage devices available at the time of writing this paper giving a general idea of the performance characteristics and comparative costs for each type of storage device.

Tape. Magnetic tape has been in use over 60 years [5] and is still used today by multi-petabyte data repositories due to its durability and economic advantages [19]. Unfortunately the reduced cost of tape technology comes with a trade-off: high access latency. A tape must first be loaded into a drive and positioned hundreds of meters which results in an latency of seconds to minutes for a random access to tape. Despite its drawbacks, we believe that tape will continue to be used as a storage technology in order to meet long term storage demand at minimal cost [9]. Tape uses no power when idle and has a strong storage density road map for future capacity growth [11]. A major barrier for using tape is its large access latency, which latency mitigation could help address.

Optical. Optical media, such as Blu-ray, uses disc material variations in order to record information that can be detected by a laser sensor. Similar to tape, optical disc is a removable media that requires no power when not in use. The main benefit of optical media over tape is its ability to support random reads with lower latency. In recent years optical storage has fallen out of use in mass storage systems due to the lack of capacity growth, although research continues to improve optical device capacity and durability [28].

Disk. Disk has previously served both performance and capacity roles in storage systems, but today it is mainly used to deliver cost-effective high capacity storage. This is especially true for use cases such as big data analytics and content delivery where a large amount of data needs to saved but only a subset of the data be retrieved randomly on demand. Compared to tape, a spinning disk can quickly locate to a random location with access times in milliseconds. The main challenge for disk is continuing to increase its capacity. The physical limits of storing bits of data in shrinking magnetized regions on the surface of a spinning disk is being reached [25].

Flash. Solid state disk (SSD) devices store data electronically and have no moving mechanical parts. This allows random access on flash to be more than forty times faster than disk and enables a large number of operations to be performed per second. Flash storage is still more expensive than disk and will likely continue to be so for the foreseeable future, although great progress has been made reducing cost and increasing the capacity of flash devices [17, 21].

Storage Arrays. In order to store a large amount of data, more than can fit on any single storage device, multiple devices must be used. Storage arrays contain multiple storage devices and spread data across them. RAID storage devices are the most common example [18]. Utilizing multiple devices in novel ways can produce new system properties that provide more benefits than the individual devices alone. For example a low latency SSD device can be used as a write-back cache for an array of disks to form a hybrid device that has both high capacity and low write latency for many workloads. The bandwidth of the composite device can be improved by by striping data across individual devices and the reliability can be improved through techniques such as erasure coding.

Cold Storage Devices. Another emerging type of storage device providing a middle ground between the latency of hard disk and tape are *cold storage devices* (CSDs) [**?**]. In order to reduce power cost, the concept of spinning up and down a disk dynamically in response to its usage is used by CSDs to pack as many disk devices as possible closely together in trays and group disks together in power and heating domains so that only a small subset of disks, perhaps only 5%, are powered concurrently [2, 8]. To further increase storage efficiency, Shingled Magnetic Recording (SMR) drives are now being used [3].

3 APPROACH

The goal of latency mitigation is to store as much data as possible in higher latency, low cost storage devices but still meet user specified latency and rate service level agreements (SLAs). Data from an object is allocated to each storage pool as needed to meet the user requirements given device performance and cost characteristics.

3.1 Archive Storage Assumptions

We make the following assumptions about archive storage systems:

- Workloads are sequential and start from offset zero.
- Pool capacity can be expanded as needed.
- Devices can be added to pools to avoid excessive queuing delays for concurrent requests.
- An object can be divided into segments on arbitrary byte boundaries.
- A latency and transfer rate *service level objective* (SLO) is provided for each object.
- SLOs are static over time.
- A large transfer buffer is available to use during get operations.

3.2 Storage System Model

Figure 2 shows an archive storage system model with three storage pools ordered by their expected delay, so that lower numbered pools have lower expected delay than higher numbered pools. Specific device types are displayed for illustrative purposes.

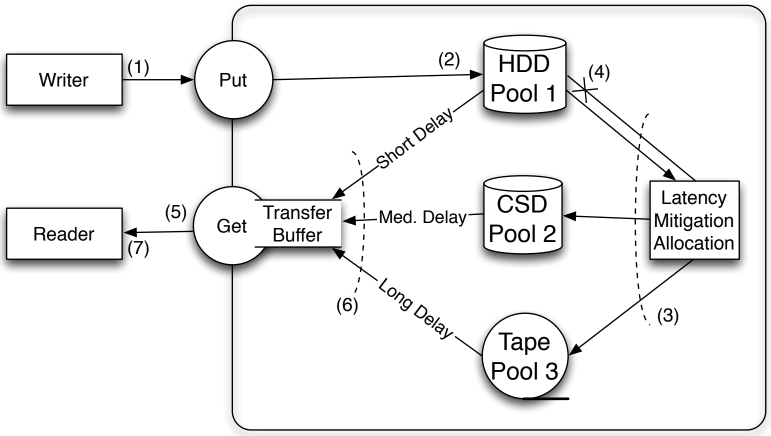


Fig. 2. Archive storage system using latency mitigation.

A writer (1) puts objects into the system where they are initially placed into to a landing pool (2). The latency and transfer rate SLO for each object are specified as metadata by the writer, or determined by system policy. The latency mitigation allocation algorithm is periodically executed (3). This algorithm determines how much data from each object should be allocated to each storage pool in order to meet transfer rate and latency requirements. The object data is transferred across the pools and any object data not required in the landing zone is deleted (4). The current objects are now allocated across the storage pools to successfully meet their transfer rate and latency requirements for future read requests. The reader requests an object through the Get operation (5), which determines the sequence of operations that must occur across the storage pools to meet object requirements (6). A transfer buffer may be used during the get operation in order to optimize the data retrieval across multiple devices simultaneously. The object is transferred sequentially to the reader meeting both latency and rate requirements (7).

3.3 Storage System Parameters

This section describes the parameters used in the construction of the latency mitigation algorithm for a general three pool storage system. Figure 3 illustrates the data transfer graph and regions formed by a single device and object requirements. The device bandwidth b_1 and delay d_1 define the infeasible transfer region in the upper left where any requirements would be impossible to meet. The latency requirement l_i and the rate requirement r_i define the requirement violation region, where a transfer would break the SLA if crossing the boundary. The region formed between these two regions is the feasible transfer region where transfers meet the SLA. Our goal is to identify the least cost transfer and allocation that meets the specified SLA. In the next subsections, we will explain in more detail each of the device, object, allocation and transfer parameters.

Device Pools. Consider a storage system consisting of three device pools with characteristics shown in Table 2. Pool bandwidth (b_i) is the expected read bandwidth of a sequential data stream accounting for expected concurrent requests. The pool cost (c_i) is an estimate of the total cost of ownership per TB of storage per month including acquisition, power, cooling, management, and floor space costs. Finally the expected delay (d_i) is the average expected latency of a sequential read request to the pool due to operating constraints. For example, a storage device may need to

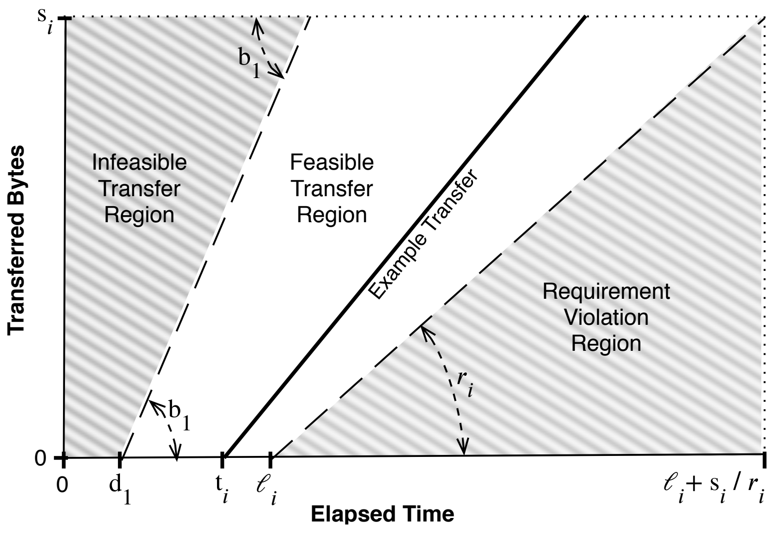


Fig. 3. Object transfer graph with regions formed by transfer requirements and parameters of a single pool. The x-axis shows the elapsed time since the system received a read request at $t = 0$, and the y-axis shows the total transferred bytes. The slopes of lines drawn on this graph are interpreted as transfer rates (bytes/sec) and the area between the long-dashed lines is the feasible transfer region where the provided system parameters can be met. If a line ventures outside of this region it is not feasible. Note that device bandwidths can be shown as alternate interior angles at the top of diagrams to reduce clutter.

$\mathbf{b} =< b_1, b_2, b_3 >$	bandwidth (bytes/sec)
$\mathbf{c} =< c_1, c_2, c_3 >$	cost ($/TB/month)
$\mathbf{d} =< d_1, d_2, d_3 >$	expected delay (secs)

Table 2. Device Pool Model

power-up disks, or mount a tape. For this paper we ignore delay variation due to physical data placement within a storage pool.

Device pools are ordered by increasing expected delay ($d_1 < d_2 < d_3$), and decreasing cost ($c_1 > c_2 > c_3$). If a device pool can not meet this assumption then it is not suitable for inclusion in the pooled storage system since it would be more cost effective to instead acquire more of a lower cost, lower latency device.

Object Requirements. Objects stored in the pooled storage system have individual object size and transfer requirements shown in Table 3. Each object has a size (s_i), a maximum latency SLO (l_i), which is the maximum acceptable time to first byte, and a minimum rate SLO (r_i), which is the minimum acceptable average transfer rate for the object. A transfer violates these requirements if it has not transferred the first byte by time l_i and at least $(t - l_i)r_i$ bytes at time $t > l_i$. In other words, if the transfer enters the requirement violation region discussed in Figure 3.

Allocation. The allocation of the M objects across three pools is recorded in a *sequential allocation matrix*: $\mathbb{A} = (a_{ij}) \in [0, 1]^{M \times 3}$. The entry a_{ij} is the proportion of object i stored in pool j. The object segment stored in pool j begins at byte offset $(\sum_{k=1}^{j-1} a_{ik}) * s_i$, and is $a_{ij} * s_i$ bytes long. Each object i must be completely allocated in the system so that $a_{i1} + a_{i2} + a_{i3} = 1$.

$\mathbf{s} =< s_1, \ldots, s_M >$	object size (bytes)
$\mathbf{l} =< \ell_1, \ldots, \ell_M >$	maximum latency (secs)
$\mathbf{r} =< r_1, \ldots, r_M >$	minimum rate (bytes/sec)

Table 3. Object Requirements Model

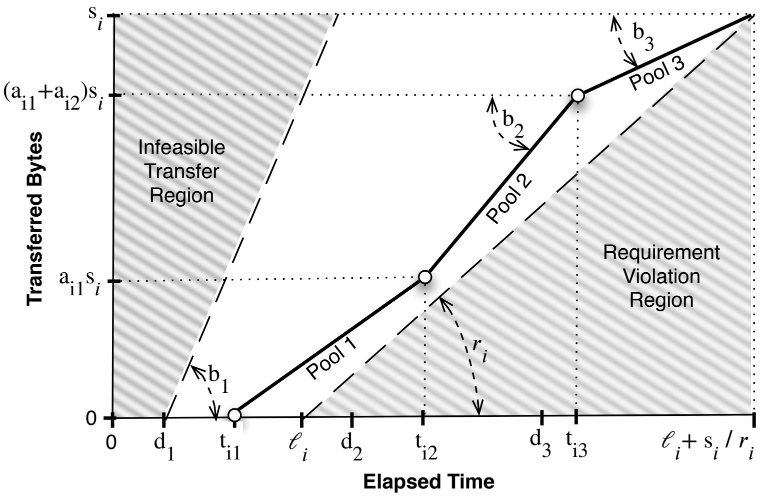
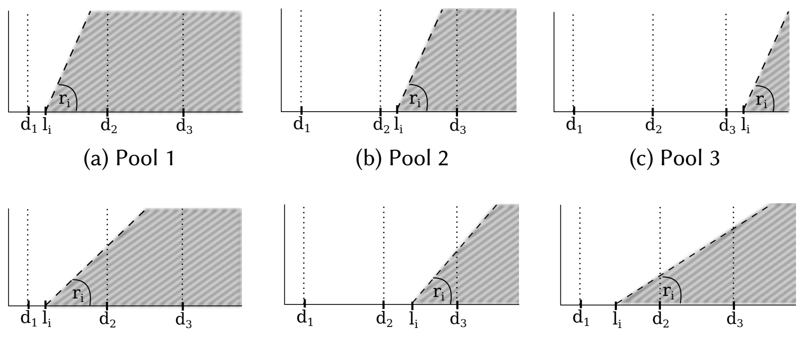


Fig. 4. Object transfer graph in a three-pool storage system. The pool parameters, object requirements, and example allocation is shown. The points defined by the allocation and transfer plan, (a_{ij}, t_{ij}), determine the transfer rate to the user from each pool. The goal of latency mitigation is to find the lowest cost feasible allocation.

Transfer. After an object is requested its data transfer must be managed so that the object requirements are met. The specific time to initiate a request of an object segment from a pool and when to begin data transfer in order to meet the transfer requirements will be a function of the pool characteristics. The transfer plan of the M objects across three pools is recorded in a *sequential transfer matrix:* $\mathbb{T} = (t_{ij}) \in [0, \infty)^{M \times 3}$. An entry of 0 means that that no data is allocated for that pool. The entry t_{ij} is defined as the time since the initial request of object i, that data transfer should start for the object segment beginning at byte offset $(\sum_{k=1}^{j-1} a_{ik}) * s_i$ of size $a_{ij} * s_i$, stored in pool j. The deadline to initiate the request for the segment of object i in pool j is $t_{ij} - d_j$ and the deadline to complete transfer for object i is $D_i = l_i + s_i/r_i$. The transfer rate must be managed by the storage system so that one pool is finishing transfer as the next pool is becoming ready. If the a pool can begin transfer before the previous pool finishes then a buffer will be used to ensure sequential transfer to the user.

N-pool Discussion. The general problem does not have optimal subproblem structure, so dynamic programming can not be directly applied. The object assignment problem across a generalized multi-pooled storage system is NP-complete [10]. In the general case with unconstrained pool bandwidths and rate requirements, the greedy algorithm of assigning the maximum to the least expensive tier and solving the remaining subproblem does not always yield feasible or optimal solutions. For instance, we may be able to lower costs by using a less expensive storage with



(a) Pool 1 (b) Pool 2 (c) Pool 3

(d) Pool 1 and 2 (e) Pool 2 and 3 (f) Pool 1, 2 and 3

Fig. 5. The feasible requirement regions in relation to device pool delays.

bandwidth lower than the required rate to store some data, but this may increase the amount of data required in higher cost tiers.

If we restrict rate requirements to be less than all of the pool bandwidths, i.e. $\forall_{i,j} r_i \leq b_j$, then the greedy algorithm does produce an optimal solution. Constraining the rate requirements to be less than any device pool bandwidth may be acceptable in most real-world situations.

4 THREE POOL ALGORITHM

Problem Statement. Given a three pool storage system under the assumptions from §3.1, with bandwidth, cost, and delay parameters $(\mathbf{b}, \mathbf{c}, \mathbf{d})$*, and with M objects having size, latency and rate requirements* $(\mathbf{s}, \mathbf{l}, \mathbf{r})$*, determine the sequential allocation matrix* $\mathbb{A}$ *and transfer plan* $\mathbb{T}$ *such that total cost is minimized.*

Figure 4 shows a non-optimal allocation and data transfer of an object stored across three pools in a storage system. The circle connectors on the transfer line show when data transfer begins to the user from each pool respectively.

Our assumptions allow us to avoid exponential complexity and determine the minimal cost allocation in a three pool system by considering each object individually, in an arbitrary order without global constraints. First we avoid discrete bin packing problems by assuming byte-boundary object segments and no capacity constraint. Next, by only considering future sequential read workloads, we have a predictable order that data must be provided to the user. This allows us to know that the beginning of the object will be accessed before later portions, allowing us to hide the latency of later portions in order to meet the SLO requirements. Finally the strict ordering of pools based on delay and cost allows us to find the minimal cost allocation by analyzing the relationships between pool bandwidths, relative pool costs, and the provided object SLO requirements.

We have constructed the latency mitigation algorithm by analyzing the problem on a case-by-case basis. Figure 5 shows the six feasible cases created by the relationship between the device delays and object SLA requirements. The cases show when the object must begin and finish transfer and determine which devices can participate in allocation. We will analyze each case in more detail in the next sections.

Figure 6 shows the high-level logic of the algorithm as a flow chart. The algorithm takes as input the object transfer requirements and the device pool model, and given our simplifying assumptions calculates the minimal cost allocation and transfer plan. Intuitively the minimal cost allocation

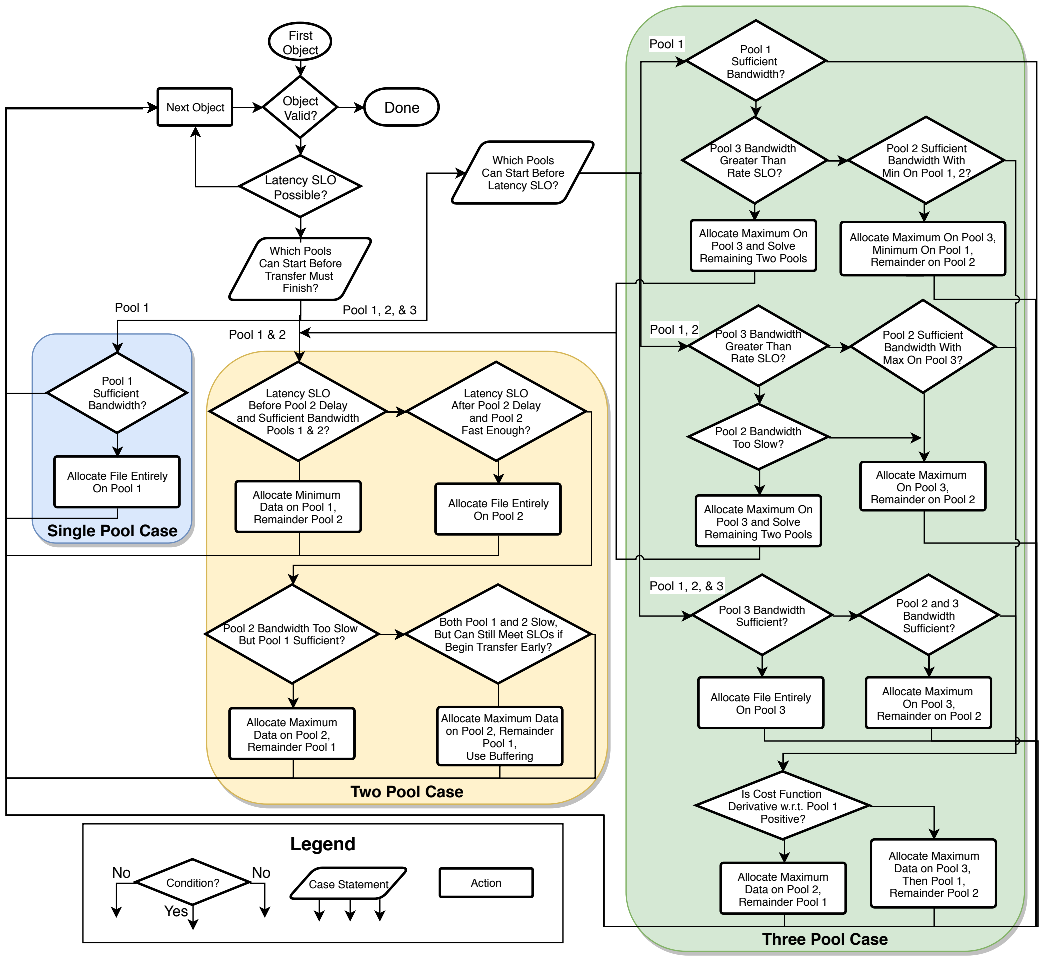

First Object
Object Valid?
Done
Next Object
Latency SLO Possible?
Which Pools Can Start Before Transfer Must Finish?
Pool 1
Pool 1 & 2
Pool 1, 2, & 3
Which Pools Can Start Before Latency SLO?
Pool 1 Sufficient Bandwidth?
Allocate File Entirely On Pool 1
Single Pool Case
Latency SLO Before Pool 2 Delay and Sufficient Bandwidth Pools 1 & 2?
Latency SLO After Pool 2 Delay and Pool 2 Fast Enough?
Allocate Minimum Data on Pool 1, Remainder Pool 2
Allocate File Entirely On Pool 2
Pool 2 Bandwidth Too Slow But Pool 1 Sufficient?
Both Pool 1 and 2 Slow, But Can Still Meet SLOs if Begin Transfer Early?
Allocate Maximum Data on Pool 2, Remainder Pool 1
Allocate Maximum Data on Pool 2, Remainder Pool 1, Use Buffering
Two Pool Case
Pool 1
Pool 1 Sufficient Bandwidth?
Pool 3 Bandwidth Greater Than Rate SLO?
Pool 2 Sufficient Bandwidth With Min On Pool 1, 2?
Allocate Maximum On Pool 3 and Solve Remaining Two Pools
Allocate Maximum On Pool 3, Minimum On Pool 1, Remainder on Pool 2
Pool 1, 2
Pool 3 Bandwidth Greater Than Rate SLO?
Pool 2 Sufficient Bandwidth With Max On Pool 3?
Pool 2 Bandwidth Too Slow?
Allocate Maximum On Pool 3, Remainder on Pool 2
Allocate Maximum On Pool 3 and Solve Remaining Two Pools
Pool 1, 2, & 3
Pool 3 Bandwidth Sufficient?
Pool 2 and 3 Bandwidth Sufficient?
Allocate File Entirely On Pool 3
Allocate Maximum On Pool 3, Remainder on Pool 2
Is Cost Function Derivative w.r.t. Pool 1 Positive?
Allocate Maximum Data on Pool 2, Remainder Pool 1
Allocate Maximum Data on Pool 3, Then Pool 1, Remainder Pool 2
Three Pool Case
Legend
No
Condition?
No
Yes
Case Statement
Action

Fig. 6. Latency mitigation flow chart showing the different possible cases of system parameters.

maximizes the amount of data in the third pool, only allocating data to lower pools when required to meet SLO requirements or device bandwidth constraints.

As LatencyMitigation executes, the matrix $\mathbb{A}$ and $\mathbb{T}$ are constructed by looping over the M objects to determine their minimal cost allocation across pools. If no such allocation is possible the object allocation remains zero and the algorithm continues to the next object. First, at line 5, the algorithm checks whether the latency SLO can be met by the first pool. If not then no pool can respond in time because higher numbered devices have higher delays, and we continue to the next object.

Next the transfer deadline is checked at lines 8-22 to see what type of problem this represents: either a one, two or three pool case. If the transfer deadline is before when pool 2 can begin, then only pool 1 needs to be considered. Likewise if the deadline is before pool 3 can begin data transfer, only the first two pools need to be considered. Finally if the deadline is after the pool 3 delay, all three pools can be potentially used. We break the logic down into sub-functions for the two and three pool cases.

Algorithm 1 Three Pool Latency Mitigation Allocation

```
 1: Global $\mathbb{A} \leftarrow \{0\}^{M\times 3}$
 2: Global $\mathbb{T} \leftarrow \{0\}^{M\times 3}$
Require: Cost $\mathbf{c}$ sorted decreasing
Require: Delay $\mathbf{d}$ sorted increasing
 3: function LatencyMitigation($\mathbf{s}, \mathbf{l}, \mathbf{r}, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
 4:     for $i$ from 1 to $M$ do
 5:         if $l_i < d_1$ then
 6:             continue
 7:         end if
 8:         $D \leftarrow l_i + s_i/r_i$                          ▷ Transfer Deadline
 9:         if $D \leq d_2$ then                                  ▷ Single Pool, Figure 5a
10:             if $b_1 \geq \frac{s_i}{D-d_1}$ then
11:                 $a_{i1} \leftarrow 1$
12:                 $t_{i1} \leftarrow \min(l_i, D - s_i/b_1)$
13:             end if
14:         else if $D \leq d_3$ then                             ▷ Figures 5b and 5d
15:             TwoPoolCase($i, s_i, l_i, r_i, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
16:         else if $d_1 \leq l_i < d_2$ then                     ▷ Figure 5f
17:             ThreePoolCase1($i, s_i, l_i, r_i, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
18:         else if $d_2 \leq l_i < d_3$ then                     ▷ Figure 5e
19:             ThreePoolCase2($i, s_i, l_i, r_i, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
20:         else                                                  ▷ Figure 5c
21:             ThreePoolCase3($i, s_i, l_i, r_i, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
22:         end if
23:     end for
24: end function
```

4.1 Single Pool Case

At line 10 only the first device pool can be used because the object must finish data transfer before any of the other pools can begin. If pool 1 has sufficient bandwidth to meet the transfer requirement we allocate the entire object.

4.2 Two Pool Cases

Figures 5b and 5d show the two cases solved by the function TwoPoolCase. There are four possible outcomes depending on the relationship between the latency SLO and pool delays, and between the pool bandwidth and rate SLO. These relationships are described by the five boolean variables $x_0, \ldots, x_5$. Variable x_0 checks if data transfer must start before pool 2 can begin. Variable x_1 is true if pool 1 fast enough to transfer the minimum required data until pool 2 can begin transfer. Variable x_2 checks if pool 2 can transfer at the rate SLO. Variable x_3 holds if pool 1 is fast enough to transfer the remainder of the data if pool 2 transfers at its maximum bandwidth for the full duration it is able. Variable x_4 checks whether pool 2 is fast enough to transfer the entire object in the duration available to it. Finally variable x_5 checks whether a buffer could be used to transfer from both pools simultaneously to meet the SLO.

Algorithm 2 Two Pool Case ($D \leq d_3$)

```
25: function TwoPoolCase($i, s, l, r, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
26:     $D \leftarrow l + s/r$
        ▷ Must use pool 1? (Figure 5d)
27:     $x_0 \leftarrow l < d_2$
        ▷ Pool 1 able to transfer minimum required data?
28:     $x_1 \leftarrow b_1 \geq \frac{(d_2-l)r}{d_2-d_1}$
        ▷ Pool 2 faster than rate SLO?
29:     $x_2 \leftarrow b_2 \geq r$
        ▷ Pool 1 able to transfer remainder of data?
30:     $x_3 \leftarrow b_1 \geq \frac{s-b_2(D-d_2)}{d_2-d_1}$
        ▷ Pool 2 able to transfer maximum amount of data?
31:     $x_4 \leftarrow b_2 \geq \frac{s}{D-d_2}$
        ▷ Can buffering avoid requirement violation?
32:     $x_5 \leftarrow b_2 \geq \frac{s-b_1 r(l-d_1)/(r-b_1)}{D-d_2}$
33:     if $x_0 \wedge x_1 \wedge x_2$ then
34:         $a_{i1} \leftarrow \frac{(d_2-l)r}{s}$
35:         $a_{i2} \leftarrow 1 - a_{i1}$
36:         $t_{i1} \leftarrow \min(l, d_2 - \frac{a_{i1}s}{b_1})$, $t_{i2} \leftarrow d_2$
37:     else if $\neg x_0 \wedge x_4$ then
38:         $a_{i2} \leftarrow 1$
39:         $t_{i2} \leftarrow l$
40:     else if $(x_0 \wedge \neg x_2 \wedge x_3) \vee (\neg x_0 \wedge x_3 \wedge \neg x_4)$ then
41:         $a_{i2} \leftarrow \frac{b_2(D-d_2)}{s}$
42:         $a_{i1} \leftarrow 1 - a_{i2}$
43:         $t_{i1} \leftarrow \min(l, d_2 - \frac{a_{i1}s}{b_1})$, $t_{i2} \leftarrow d_2$
44:     else if $(x_0 \wedge \neg x_2 \wedge \neg x_3 \wedge x_5) \vee$
45:             $(\neg x_0 \wedge \neg x_3 \wedge \neg x_4 \wedge x_5)$ then
46:         $t_{i1} \leftarrow d_1$
47:         $t_{i2} \leftarrow d_2$
48:         $a_{i2} \leftarrow \frac{b_2(D-d_2)}{s}$
49:         $a_{i1} \leftarrow 1 - a_{i2}$
50:     end if
51: end function
```

At line 34 some data must be allocated to pool 1 in order to meet the latency requirement since x_0 is true. If both pool 1 and 2 have sufficient bandwidth, that is $x_1 \wedge x_2$ holds, then the minimum feasible amount of data is allocated to pool 1 and the remainder to pool 2.

Next at line 38, pool 2 can by itself meet both the latency ($\neg x_0$) and rate requirement (x_4) , therefore the entire object is allocated to pool 2.

In the third possibility at line 41, data must be allocated to pool 1 to compensate for limited pool 2 bandwidth, which can occur for two reasons. The first reason is if the latency SLO is before pool 2 can begin transfer and pool 2 is too slow to meet the rate SLO ($x_0 \wedge \neg x_2$), but pool 1 is fast enough to compensate by transferring more data before pool 2 (x_3). The second reason is if the latency SLO is after the pool 2 delay and pool 2 is too slow to transfer the entire object ($\neg x_0 \wedge \neg x_4$), but pool 1 is fast enough to transfer the remaining data (x_3). In either case, the maximum amount of data that

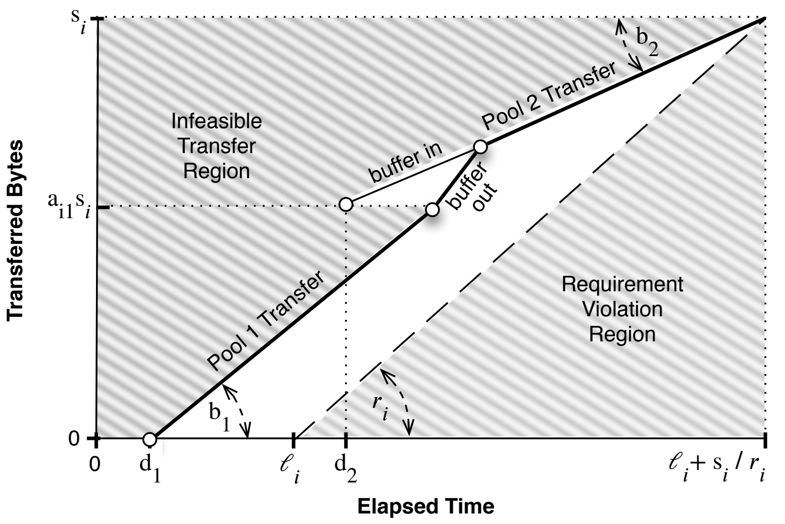


Fig. 7. Pool transfer buffering when both pool 1 and 2 have limited bandwidth.

pool 2 can possibly transfer is allocated to pool 2, with the remainder of the object allocated to pool 1.

Finally at line 46 both pool 1 and 2 are bandwidth limited, so we must check whether buffering can be used as shown in Figure 7. Again there are two conditions depending on whether the latency SLO is before or after pool 2 delay (x_0). If before, then we check if the pool 2 bandwidth is slower than the required rate SLO ($\neg x_2$) and pool 1 bandwidth is too slow to let pool 2 to start transfer immediately after its delay ($\neg x_3$). If after, then we check that pool 2 cannot transfer the entire object itself ($\neg x_4$) and again that pool 1 bandwidth is too slow ($\neg x_3$). In both cases we also check that pool 1 is fast enough to transfer enough data before violating the transfer requirement (x_5). If either condition is true then the pool 1 transfer will complete before crossing into the violation region and we can buffer the pool 2 transfer.

If none of the previous conditions are true, then the object does not have a feasible allocation and we continue to the next object.

4.3 Three Pool Cases

If the transfer deadline is after the pool 3 delay as in Figures 5c, 5e, and 5f, then we can consider allocating data across all three pools. The minimal cost allocation will depend on the relative delays, bandwidth and cost differences of the pools. The latency SLO is used to further divide the problem into cases depending on when the data transfer must begin relative to the device delays.

ThreePoolCase1 is called when the latency requirement is between the first two pool delay times (Figure 5f), which means that some amount of pool 1 storage is required in order to meet the latency requirement. At line 54 we check if it possible for pool 1 to transfer the minimum amount of data required using the maximum amount of available time. If not then there is no feasible allocation and we continue on to the next object, otherwise we continue by checking the other pool bandwidths. If pool 3 bandwidth is greater than or equal to the required transfer rate (line 55), then we can allocate the maximum possible to pool 3. This is equal to size of the object minus the amount of data required to transfer at the rate SLO before pool 3 can begin transfer. The problem is now reduced to a two pool case with object size $(d_3 - l_i)r_i$.

Algorithm 3 Three Pool Case ($D > d_3 \wedge d_1 \leq l_i < d_2$)

```
52: function ThreePoolCase1($i, s, l, r, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
53:     $D \leftarrow l + s/r$
54:     if $b_1 < \frac{(d_2-l)r}{d_2-d_1}$ then return
55:     else if $b_3 \geq r$ then
56:         $a_{i3} \leftarrow \frac{(D-d_3)r}{s}$
57:         $t_{i3} \leftarrow d_3$
            ▷ Solve two pool case of size $(d_3 - l)r$
58:         TwoPoolCase($i, (d_3 - l)r, l, r, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
59:     else if $b_2 \geq \frac{s-(D-d_3)b_3-(d_2-l)r}{d_3-d_2}$ then
60:         $a_{i1} \leftarrow \frac{(d_2-d_1)r}{s}$
61:         $a_{i3} \leftarrow \frac{(D-d_3)b_3}{s}$
62:         $a_{i2} \leftarrow 1 - a_{i1} - a_{i3}$
63:         $t_{i1} \leftarrow l,\ t_{i2} \leftarrow d_2,\ t_{i3} \leftarrow d_3$
64:     else
65:         ThreePoolMinCost($i, s, l, r, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
66:     end if
67: end function
```

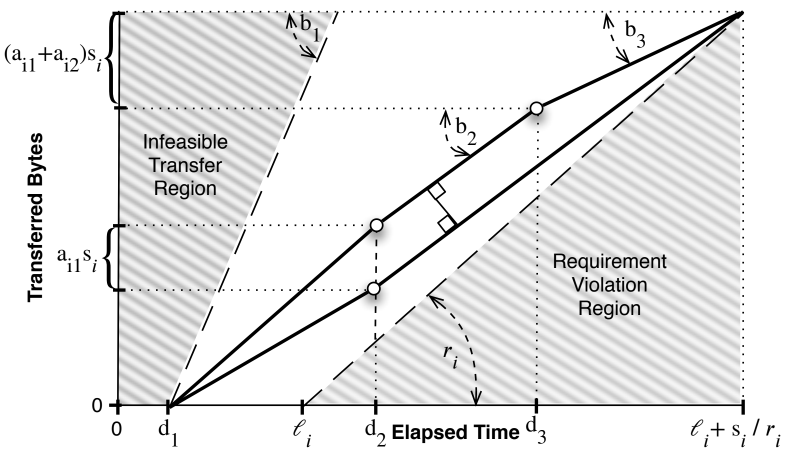


Fig. 8. Two possible allocations when both pool 2 and 3 have limited bandwidth. The top transfer uses all pools, while the bottom transfer only uses pool 1 and 2. The lowest cost option between the two must be found. Note that the lower transfer starting at d_2 also has rate b_2 (it is parallel to the line above it).

If pool 3 bandwidth isn't greater than the required transfer rate, then we check if pool 2 has sufficient bandwidth to transfer the remaining data with the maximum allocation to pool 3 and the minimum allocation to pool 1 (line 59).

If none of the preceding conditions matched then pool 2 and 3 have limited bandwidth and we must determine if some combination of pool allocation is feasible and minimal cost. Figure 8 shows two candidate transfer lines for this limited bandwidth case. The upper boundary uses all three pools allocating the maximum possible to pool 3, while the bottom boundary uses pool 1 and 2

Algorithm 4 Three Pool Case With Limited Bandwidth

```
68: function ThreePoolMinCost($i, s, l, r, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
69:     $D \leftarrow l + s/r$
        ▷ Derivative of cost function wrt. $a_{i1}$
70:     if $c_1 - c_2 + \frac{(c_3-c_2)b_3}{b_2-b_3} \geq 0$ then
        ▷ Use lower transfer line in Figure 8
71:         $t \leftarrow d_1$
72:         if $b_1 \neq b_2$ then
73:             $t \leftarrow \frac{b_1 d_1 - b_2 D + s}{b_1 - b_2}$
74:         end if
75:         $t_{i2} \leftarrow \max(t, d_2, l)$
76:         $a_{i2} \leftarrow \frac{(D-t_{i2})\min(b_2, r)}{s}$
77:         $a_{i1} \leftarrow 1 - a_{i2}$
78:         if $a_{i1} \neq 0$ then
79:             $t_{i1} \leftarrow \min(l, t_{i2} - \frac{a_{i1}s}{b_1})$
80:         end if
81:     else
        ▷ Use upper transfer in Figure 8
82:         $a_{i1} \leftarrow \frac{(d_2-d_1)b_1}{s}$
83:         $t \leftarrow \frac{b_1 d_1 - b_2 d_3 - (D - d_3 + s_i)b_3}{b_1 - b_2}$
84:         if $t > d_2$ then
85:             $a_{i1} \leftarrow 1 - \frac{(D-d_3)b_3 + (d_3-d_2)b_2}{s}$
86:         end if
87:         $a_{i3} \leftarrow \frac{b_3(D-d_3)}{s}$
88:         if $t > d_3$ then
89:             $a_{i3} \leftarrow 1 - a_{i1}$
90:         end if
91:         $a_{i2} \leftarrow 1 - a_{i1} - a_{i3}$
92:         $t_{i1} \leftarrow \min(l, t - \frac{a_{i1}s}{b_1})$
93:         $t_{i2} \leftarrow t,\ t_{i3} \leftarrow d_3$
94:     end if
95: end function
```

alone allocating the maximum possible to pool 2 with no pool 3 transfer. We must choose between the two possibilities based on the relative costs of the pools.

The function ThreePoolMinCost determines the optimal allocation for this limited bandwidth case depending on the relative costs of the three pools. This is implemented as a separate function because the situation arises in all three three-pool cases.

The formula at line 70 is the derivative of the storage cost function for this case with respect to a_{i1}. If this value is positive then the cost increases as a_{i1} increases, so to minimize cost we choose the minimum value of a_{i1}. This corresponds to the lower boundary. Likewise, if the derivative is negative then the cost decreases as a_{i1} increases, so we choose the maximum value of a_{i1} which corresponds to the upper boundary. This seems counter-intuitive, but the reason is the upper boundary includes pool 3 storage which is the least expensive, and can offset the increased cost of using more pool 1 storage. This would be the case if pool 2 cost was closer in cost to pool 1 than pool 3. If by chance the derivative is zero, then both boundaries would have the same cost. By using

	Pool 1 Disk	Pool 2 CSD	Pool 3 Tape
b	1 GB/s	1 GB/s	250 MB/s
c	\$2.75 TB/mo	\$1.32 TB/mo	\$0.45 TB/mo
d	.01 secs	30 secs	300 secs

s	Sizes from ECMWF dataset
l_i	20 secs
r_i	10 MB/s

Table 4. Baseline parameters used for evaluation. We vary each parameter independently from the baseline to view its affect on the total cost of the system.

the derivative we can avoid calculating the cost each iteration, and notice that the decision is static entirely based on the pool properties independent of the individual object SLOs.

We have omitted explanations of the functions ThreePoolCase2 and ThreePoolCase3 because they are similar to ThreePoolCase1. A full derivation and proof of the algorithm is provided as a supplement to this paper.

Complexity. The main function LatencyMitigation loops over all of the M objects once, and each of the called sub-functions is a conditional lookup of $O(1)$ complexity, therefore the total complexity is $O(M)$.

5 EVALUATION

This section examines the feasibility of latency mitigation allocation for current storage technology and datasets. We have used a snapshot of the ECMWF MARS filesystem containing metadata for 12.8 million files representing 59.6 PB worth of data created over about seven years from 2008 to 2014 [13]. Table 4 shows the chosen baseline device parameters and object requirements in our study.

The monthly storage costs (**c**) are an amortized estimate over 6 years of the archive system taking into account hardware acquisition, maintenance, power, cooling, and floor space costs. The disk and tape costs are based on results found in [19] and the CSD costs are based on an assumption that maintenance and floor space costs can be reduced by 50% and energy costs reduced by 75% as shown in [2]. Pool bandwidth (**b**) is the expected transfer rate for each object and not total bandwidth of the pool. Finally the delay (**d**) for each pool is an expected 99th percentile time-to-first-byte for a object stored in the storage pool.

Our goal is to show that latency mitigation could be used for real world data sets in order to meet specific latency and rate requirements at lower costs. We use the object size distribution from the ECMWF dataset and for the baseline object parameters assume simply that r_i = 10 MB/s and l_i = 20 secs for all objects. This means that the system will allocate objects so that we will begin returning data to the user no later than 20 secs at a rate of at least 10 MB/s. In an actual implementation transfer parameters could be chosen by the system designer to meet specific requirements, provided by applications on a per object basis, or calculated using some form of policy.

To determine total cost, we process the ECMWF objects in trace order. As objects are created in the system they contribute to the total cost on a daily basis. At the end of the 7 year period all of the objects have been created in the system. Figure 9 shows the total cost over the 7 years for the four possible pool configurations using latency mitigation with the baseline parameters from Table

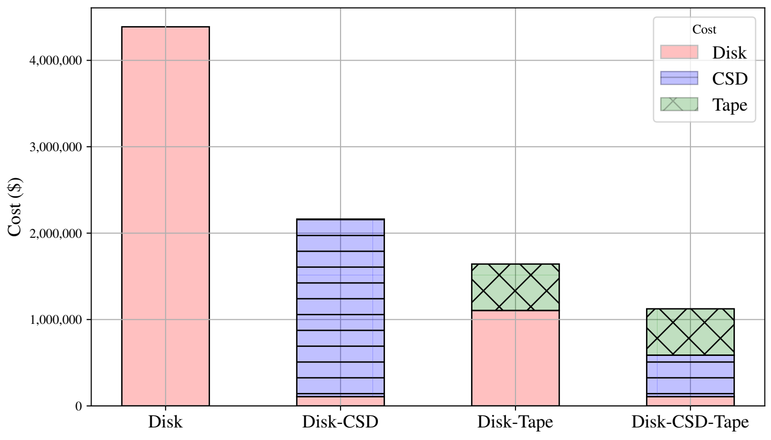


Fig. 9. Total storage costs over 7 years for different system configurations using latency mitigation with baseline parameters from Table 4.

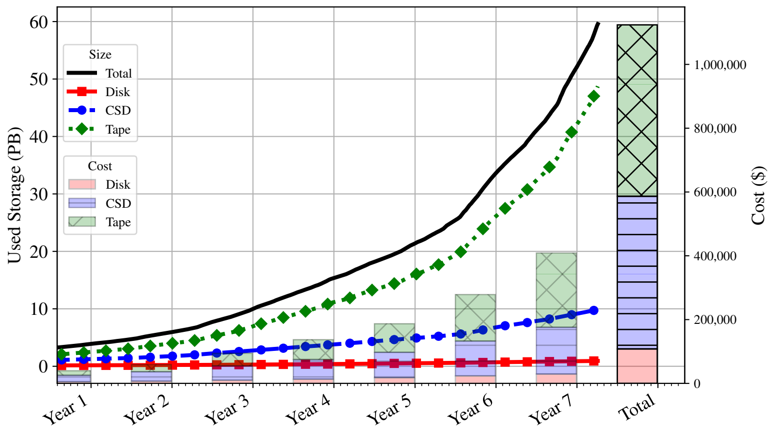


Fig. 10. Breakdown of yearly storage costs and usage over time for the Disk-CSD-Tape system configuration from Figure 9.

4. All four configurations meet the baseline SLA, and the graph shows that using all three pools results in a substantially lowest cost configuration.

Figure 10 shows a more detailed breakdown of the pool allocation and total cost breakdown over time for each device pool in the three pool configuration. The used storage increases exponentially over the seven years, with a majority of the data able to be stored on tape. The costs for each individual year is shown, and also the total cost across all seven years. We will focus on the total cost after the 7 years for the remainder of the graphs as we explore the parameter space.

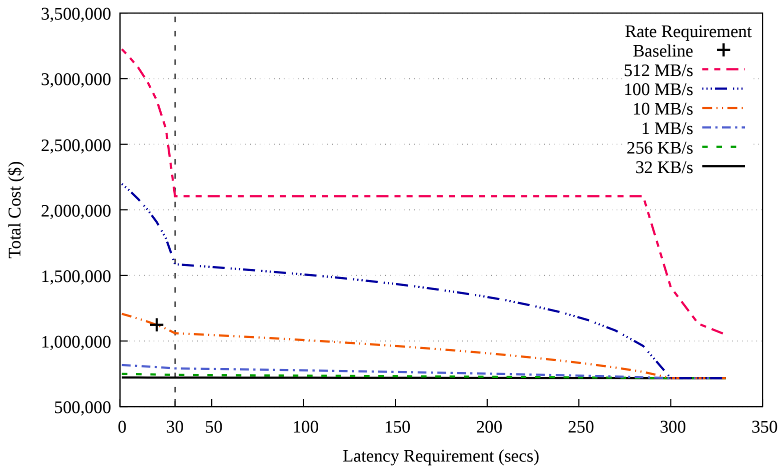


Fig. 11. Effect of Latency and Rate SLOs

5.1 Parameter Sensitivity

We will explore the latency mitigation parameter space by varying each variable individually from the baseline to see how different design decisions effect the total cost of the system. In each graph the baseline parameter case is shown by a plus symbol for reference.

Latency and Rate SLO. Figure 11 shows the effect of the latency and rate SLO on total cost. Each line represents a particular rate requirement and varying latency requirement between .01 and 330 seconds. As the latency requirement increases the amount of storage required in pools 1 and 2 decreases which reduces the cost except in the 512 MB/s high rate case. In this case a very high latency requirement is needed before tape can be used. The low rate cases less than 10 MB/s are not sensitive to the latency requirement since a majority of the object data can already be served by the tape pool. As the rate requirement increases the proportion of data stored in disk and CSD pools grows, raising the cost. This graph illustrates an opportunity for a system designer to use latency mitigation to balance the tradeoff of cost and performance, or to provide multiple classes of service at different pricing.

Average Object Size. The larger the average object is, the more data can be stored in high latency pools. This is because the transfer of the beginning of the object from lower pools overlaps the delay of the higher pools. Figure 12 was created by multiplying the object sizes found in the ECMWF dataset by a scaling factor. The baseline average object size is just above 4GB and was incrementally scaled down, rerunning the latency mitigation allocation, until the average was just above 4MB (1/1000). The solid line shows the total cost for the average size, and the stacked background shows the percentage allocated to each tier. At an average size of 4GB about 80% of the data can be stored on tape. As the average file size decreases the proportion of disk and cold storage increase until at 4MB, about 80% of the data must be stored on disk.

For the ECMWF dataset the total cost approximately cuts in half for each 16x reduction in average object size, approaching the cost of storing all data in the tape pool. This means that large object sizes are needed to make latency mitigation a feasible option to reduce storage costs.

Relative Device Costs. We vary the cost of pool 2 between the cost of pool 1 and 3 in Figure 13. The total cost increases linearly for each case as pool 2 cost increases. The low rate cases are not

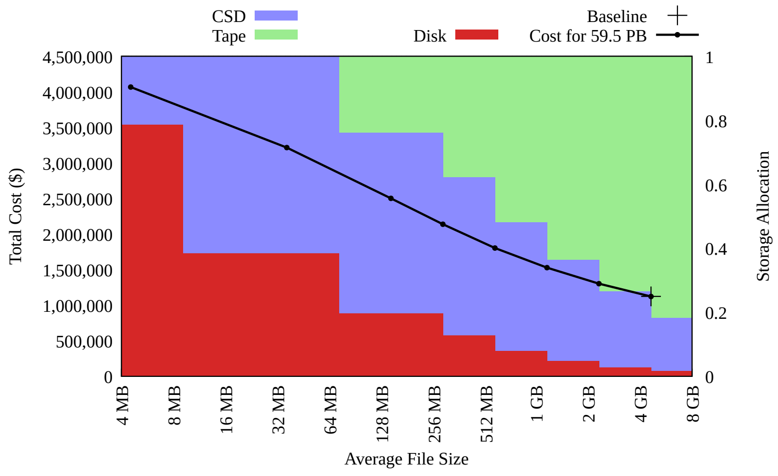


Fig. 12. Effect of Average Object Size

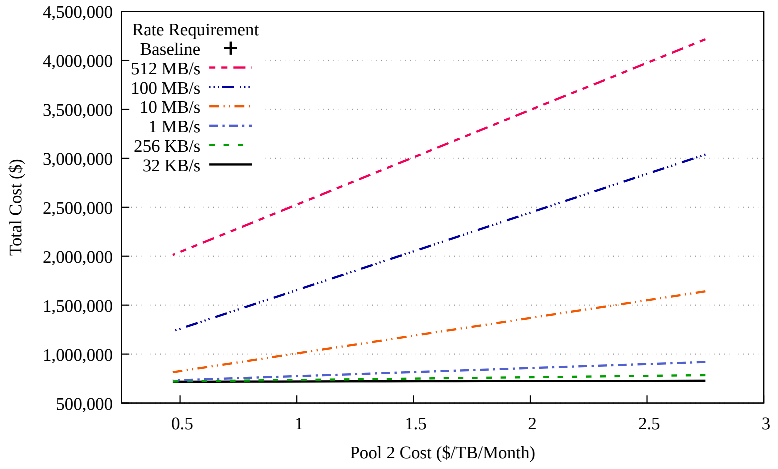


Fig. 13. Effect of Relative Device Costs

sensitive to the cost increases, but as the rate requirement increases more data must be stored in pool 2, making it more sensitive to pool 2 cost. For allocation decisions, the relative pool cost is only considered for bandwidth limited cases. As we saw from Figure 11, the decision boundary occurs at closer to pool 3 latencies but this graph only considers the baseline latency ($l_i = 20$) so we do not see any non-linear effects.

Relative Device Delay. Figure 14 shows the effect on total cost of varying pool 2 device delay. For the low rate cases, the increased pool 2 delay does not have much effect on total cost since most of the data is stored on tape. For the 10 MB/s and greater rate cases, the effect is more significant. The cost increases quickly at first for each of these cases but then tapers off to a limit as pool 2 delay approaches pool 3 delay. This is because as the pool 2 delay increases, the amount of data

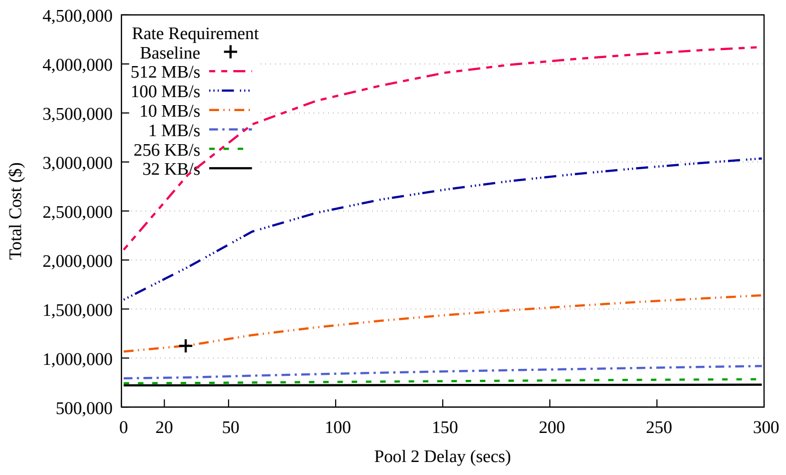


Fig. 14. Effect of Relative Device Delay

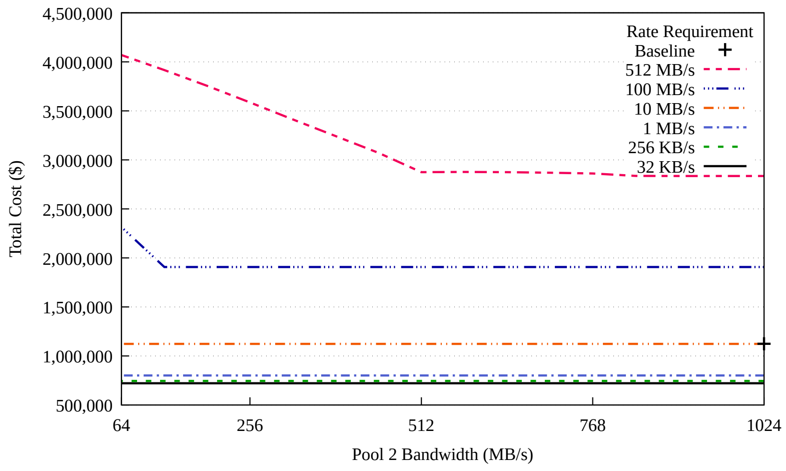


Fig. 15. Effect of Relative Device Bandwidth

stored in pool 1 increases. Some files that could previously be partially stored in pool 2, must now be completely stored in pool 1.

Relative Device Bandwidth. Finally we inspect the effect of relative pool bandwidth on total cost in Figure 15 by varying pool 2 bandwidth from 64 to 1024 MB/s. For rate requirements less than 100 MB/s, there is no effect on the total cost as there is no bandwidth restriction that would affect the allocation decision. If pool 2 bandwidth is less than the rate SLO, then a larger portion of the object must be stored on pool 1. This is seen for the 100 and 512 MB/s rates in the figure where the cost decreases until the rate requirement matches the pool 2 bandwidth. For larger bandwidths the cost is no longer affected.

6 RELATED WORK

Early solutions to multi-device file allocation focus on solving integer linear programs in order to assign complete files to storage devices. The earliest papers we found mentioning optimal file assignment across multiple storage systems were written in 1969 [7] and 1973 [6]. These models consider allocating complete objects to storage devices and also consider storage space as a constraint. Chen provides a problem statement for "minimizing the storage cost with individual response time requirements" that closely parallels our model but does not consider the addition of a rate requirement. A survey of early solutions to the file assignment problem can be found in [10].

Hierarchical Storage Management (HSM) systems typically store entire files in a particular layer, although some solutions allow an initial fragment of the file to to exist on the disk cache [4]. Latency mitigation could be used in HSM systems to determine the size and rate of these initial fragments. The HSM system could use a calculated expected delay in order to figure out the required transfer rate of the initial fragment.

CSDs provide intermediate characteristics between disk and tape that provides new design possibilities. Devices such as shingled write disks (SWDs) or Blu-ray drives can act as a latency bridge between disk and tape. One example of a CSD is Microsoft's Pelican which utilizes a high-density array of SWDs arranged in power and cooling groups in order to increase storage capacity and lower cost [2, 3]. The higher latency of CSDs still poses a challenge to integration with traditional software applications where design assumptions may no longer be valid. Borovica et. al. investigated how to modify database query executions for CSDs [**?**]. The authors modified applications to utilize the data locality on the disk groups to reorder query execution to avoid unnecessary power-down cycles. In contrast, latency mitigation is a transparent way to aid applications to integrate with large latency low cost devices.

Another related research topic is content delivery. Video is a continuous media that is typically sequentially accessed, and therefore an ideal candidate for latency mitigation. The idea of pipelining transfer from tape in order to reduce access latency was introduced in [12]. Latency mitigation employs pipelined data transfer to begin returning data to the user as soon as possible. Ma investigated allocation of video frames to proxy servers in order to reduce bandwidth requirements [15]. The graphical representation of feasible transfer schedules was very helpful in working through the latency mitigation cases.

7 CONCLUSION

The main contribution of this paper is the latency mitigation algorithm that uses individual object SLOs and device performance characteristics to allocate individual objects across three pools to minimize cost. We have shown this approach feasible with a real-world data set.

Latency mitigation could be applied in a variety of ways. One possible application is to reduce storage costs by using high-latency storage devices in an object storage system such as OpenStack Swift to provide predictable access latencies and transfer rates to avoid timeout issues. Another possibility is to provide SLA guarantees in active archives such as at ECMWF. Latency mitigation would provide a means for determining the cost in order to meet performance SLAs and the sizing of storage pool capacities.

Our current assumptions (§3.1) would likely cause limitations when used in actual implementation. In storage systems the bandwidth and delay will depend on how busy the system is. In the case of tape systems, if the number of drives in use becomes saturated or if multiple requests are made for objects stored on the same physical tape, delays will be many times more than the average. The result would be that object SLOs would be violated and either the actual latency would exceed the requirement or the transfer rate would need to be reduced in order to mitigate the increased latency. We plan to extend our model to incorporate predictive performance and device optimizations in order to handle these challenges.

In the future we plan to implement a generalized N-pool system, and analyze a variety of pooled storage architectures utilizing new storage devices in order to meet a wide variety of object requirements.

ACKNOWLEDGEMENTS

This work was partially supported by NSF I/UCRC Center Research in Intelligent Storage and the following NSF awards 1439662, 1525617, 1536447, 1708886, 1763008, and 1812537. We would like to thank ECMWF for providing the MARS filesystem trace data and Mr. Grawinkel for making his Perl scripts to parse the trace data publicly available.

REFERENCES

[1] Raja Appuswamy, Renata Borovica, Goetz Graefe, and Anastasia Ailamaki. 2017. The Five minute Rule Thirty Years Later and its Impact on the Storage Hierarchy. *Proceedings of the 7th International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures* (2017). https://infoscience.epfl.ch/record/230398

[2] Shobana Balakrishnan, Richard Black, Austin Donnelly, Paul England, Adam Glass, Dave Harper, Sergey Legtchenko, Aaron Ogus, Eric Peterson, and Antony Rowstron. 2014. Pelican: A Building Block for Exascale Cold Data Storage. *Proceedings of the Operating Systems Design and Implementation* (2014).

[3] Richard Black, Austin Donnelly, Dave Harper, Aaron Ogus, and Antony Rowstron. 2016. Feeding the Pelican: Using Archival Hard Drives for Cold Storage Racks. In *USENIX Workshop on Hot Topics in Storage and File Systems*.

[4] P. L. Bradshaw, K. W. Brannon, T. Clark, K. Dahman, S. Doraiswamy, L. Duyanovich, B. L. Hillsberg, W. Hineman, M. Kaczmarski, B. J. Klingenberg, X. Ma, and R. Rees. 2008. Archive storage system design for long-term storage of massive amounts of data. *IBM Journal of Research and Development* 52, 4.5 (jul 2008), 379–388. https://doi.org/10.1147/rd.524.0379

[5] R. Bradshaw and C. Schroeder. 2003. Fifty years of IBM innovation with information storage on magnetic tape. *IBM Journal of Research and Development* 47, 4 (jul 2003), 373–383. https://doi.org/10.1147/rd.474.0373

[6] Peter P S Chen. 1973. Optimal File Allocation In Multi-level Storage Systems. In *National Computer Conference*. 277–282.

[7] Wesley W Chu. 1969. Optimal File Allocation in Multiple Computer System. *IEEE Trans. Comput.* 100, 10 (1969), 885–889.

[8] Dennis Colarelli and Drik Grunwald. 2002. Massive Arrays of Idle Disks For Storage Archives. *ACM/IEEE SC 2002 Conference (SC'02)* 00, c (2002), 1–11. https://doi.org/10.1109/SC.2002.10058

[9] Tom Coughlin. 2017. How Big Are Your Dreams? Gauging the Size of Future Content.

[10] Lawrence W. Dowdy and Derrell V. Foster. 1982. Comparative Models of the File Assignment Problem. *Comput. Surveys* 14, 2 (1982), 287–313. https://doi.org/10.1145/356876.356883

[11] Robert E. Fontana and Gary M. Decad. 2018. Moore's law realities for recording systems and memory storage components: HDD, tape, NAND, and optical. *AIP Advances 8(5)* 8, 5 (2018).

[12] S Ghandeharizadeh, Ali Dashti, and Cyrus Shahabi. 1995. Pipelining mechanism to minimize the latency time in hierarchical multimedia storage managers. *Computer Communications* 16, 3 (1995), 170–184. http://www.sciencedirect.com/science/article/pii/014036649598540L

[13] Matthias Grawinkel, Lars Nagel, Markus Mäsker, Federico Padua, André Brinkmann, and Lennart Sorth. 2015. Analysis of the ECMWF Storage Landscape. In *FAST'15*.

[14] IBM. 2017. TS4500 Drive Performance. https://www.ibm.com/support/knowledgecenter/en/STQRQ9/com.ibm.storage.ts4500.doc/ts4500{_}ipg{_}3584{_}ircc4.html

[15] Weihsiu Ma and David Du. 2002. Reducing Bandwidth Requirement for Delivering Video Over Wide Area Networks With Proxy Server. *IEEE Transactions on Multimedia* 4, 4 (2002), 539–550. http://vc.cs.nthu.edu.tw/home/paper/codfiles/clchan/200303101323/Reducing{_}Bandwidth{_}Requirement{_}for{_}Delivering{_}Video{_}Over{_}Wide.pdf

[16] Viktor Mayer-Schonberger and Kenneth Cukier. 2014. *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. Houghton Mifflin Harcourt, New York. https://doi.org/10.2501/IJA-33-1-181-183 arXiv:arXiv:1011.1669v3

[17] Chris Mellor. 2018. Got some fancy new flash in the works, huh Micron? Join the QLC. *The Register* (2018). https://www.theregister.co.uk/2018/02/09/micron{_}qlc/

[18] David A Patterson, Garth Gibson, Randy H Katz, and Evans Hall. 1988. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *ACM*, Vol. 17. 109–116.

[19] David Reine and Mike Kahn. 2015. Continuing the Search for the Right Mix of Long-Term Storage Infrastructure - A TCO Analysis of Disk and Tape Solutions. *The Clipper Group Calculator* (2015), 1–18.

[20] David Reinsel, John Gantz, and John Rydning. 2017. *Data Age 2025: The Evolution of Data to Life-Critical.* Technical Report April. International Data Corporation. 1–25 pages. http://www.seagate.com/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf

[21] David S. H. Rosenthal. 2017. The Medium-Term Prospects for Long-Term Storage. *Library Hi Tech* 35, 1 (2017).

[22] Samsung. 2018. *860 EVO Datasheet.* Technical Report. 1–6 pages.

[23] R. Spencer. 2013. The Square Kilometre Array: the ultimate challenge for processing big data. *IET Seminar on Data Analytics 2013: Deriving Intelligence and Value from Big Data* (2013), 5–5. https://doi.org/10.1049/ic.2013.0235

[24] Verbatum. 2018. Bluray 100GB BD-R XL Triple Layer. http://a.co/d/6fon0JP

[25] D. Weller and a. Moser. 1999. Thermal effect limits in ultrahigh-density magnetic recording. *IEEE Transactions on Magnetics* 35, 6 (1999), 4423–4439. https://doi.org/10.1109/20.809134

[26] Western Digital. 2018. Ultrastar DC HC620 SMR HDD Datasheet. (2018).

[27] Western Digital. 2019. *Ultrastar DC HC530 HDD Datasheet.* Technical Report.

[28] Qiming Zhang, Zhilin Xia, Yi-bing Cheng, and Min Gu. 2018. High-capacity optical long data memory based on enhanced Young's modulus in nanoplasmonic hybrid glass composites. *Nature Communications* 9, 2018 (2018), 1–6. https://doi.org/10.1038/s41467-018-03589-y

Received -; revised January 2020; accepted -