InftyDedup: Scalable and Cost-Effective Cloud Tiering with Deduplication

Iwona Kotlarska, Andrzej Jackowski, Krzysztof Lichota, Michal Welnicki, and Cezary Dubnicki, *9LivesData, LLC;* Konrad Iwanicki, *University of Warsaw*

https://www.usenix.org/conference/fast23/presentation/kotlarska

This paper is included in the Proceedings of the 21st USENIX Conference on File and Storage Technologies.

February 21–23, 2023 • Santa Clara, CA, USA

978-1-939133-32-8

InftyDedup: Scalable and Cost-Effective Cloud Tiering with Deduplication

Iwona Kotlarska
9LivesData, LLC

Andrzej Jackowski
9LivesData, LLC

Krzysztof Lichota
9LivesData, LLC

Michal Welnicki
9LivesData, LLC

Cezary Dubnicki
9LivesData, LLC

Konrad Iwanicki
University of Warsaw

Abstract

Cloud tiering is the process of moving selected data from on-premise storage to the cloud, which has recently become important for backup solutions. As subsequent backups usually contain repeating data, deduplication in cloud tiering can significantly reduce cloud storage utilization, and hence costs.

In this paper, we introduce InftyDedup, a novel system for cloud tiering with deduplication. Unlike existing solutions, it maximizes scalability by utilizing cloud services not only for storage but also for computation. Following a distributed batch approach with dynamically assigned cloud computation resources, InftyDedup can deduplicate multi-petabyte backups from multiple sources at costs on the order of a couple of dollars. Moreover, by selecting between hot and cold cloud storage based on the characteristics of each data chunk, our solution further reduces the overall costs by up to 26%–44%. InftyDedup is implemented in a state-of-the-art commercial backup system and evaluated in the cloud of a hyperscaler.

1 Introduction

Managing the surging volumes of data that require protection or long-term retention increasingly necessitates novel backup strategies [18]. A popular approach is employing cloud-based solutions. For instance, according to Veeam, the number of organizations adopting cloud-powered data protection is expected to rise from 60% in 2020 to 79% in 2024 [66]. Similarly, in a survey by ESG, 72% of the participants confirmed using *tiering* techniques to move colder data (e.g., older backups and archives) from on-premise storage to the cloud [22].

In this context, data *deduplication* can become effective. Since consecutive backups often contain repeating data [47, 50], this technique reduces storage utilization tens of times [70]. As a result, deduplication is a core feature of several storage systems for on-premise backup applications [33, 57, 84]. In this light, for backup use cases, it is sensible to consider *cloud tiering with deduplication*, that is, moving data from a local tier (e.g., on-premise backup appliances) to a cloud tier (e.g., a cloud object store), so that ultimately the data kept in the cloud tier are deduplicated.

However, implementing cloud tiering with deduplication poses two major problems. First, state-of-the-art cloud storage systems provided by hyperscalers (e.g., Amazon, Google, and Microsoft) do not offer deduplication as a core functionality for their clients. Consequently, deduplication algorithms tailored for cloud tiering have to be developed. In the process, the extra tier should be treated not only as a challenge but also a potential opportunity for exploring novel deduplication paradigms dedicated for the cloud. Second, there is a large variety of available cloud storage service types, notably differing in pricing models. Initially, a lower storage cost implied a longer retrieval time (e.g., AWS Glacier) but nowadays, systems like AWS Glacier Instant Retrieval [77] offer the same performance as other cloud storage services. The trade-off is that with a decreased per-byte monthly storage fee, the costs of data retrieval and the minimal data storage period are increased. Therefore, algorithms have to be devised to decide what type of service to use for which data, specifically considering the peculiarities due to deduplication.

As we discuss in more detail further in the paper, despite some research progress, these two problems are largely open. In short, regarding the first problem, although a few backup applications [54, 59] and backend appliances [34, 68] with deduplication offer mechanisms for cloud tiering, they heavily rely on and are implemented mainly in the local tier. In effect, deduplication between different local tier systems is not supported for data stored in the cloud. Moreover, the entire process is fundamentally limited by the resources of the local tier. In other words, despite the possibilities offered by the hyperscalers, the actual scalability of the cloud tier in such solutions is severely limited, proportionally to what is offered by the local tier. When it comes to the second problem, although the diversity of the service models offered by the hyperscalers can also be exploited in some solutions [52], this has to be configured manually or, at best, through policies depending on the ages of data collections. However, deduplication typically entails chunking data collections into smaller pieces that may be referenced multiple times, thereby possibly having different access patterns. This calls for finer-grained and more automated approaches to storage type selection.

In this paper, we address both these problems, introducing solutions for scalable and cost-effective cloud tiering with deduplication. Accordingly, our contribution is twofold.

First, we present InftyDedup, a novel system for cloud tiering with deduplication. Like the existing tiering-to-cloud backup solutions, InftyDedup moves selected data from a local-tier system to the cloud, based on customer-specific backup policies. However, its operation aims to maximize scalability by exploiting cloud services — not only for storage but also for computation. Therefore, rather than relying on deduplication methods of on-premise solutions, InftyDedup deduplicates data using the cloud infrastructure. This is done periodically in batches before actually transferring data to the cloud, which, among others, enables the dynamic allocation of cloud resources. Other functionalities, such as garbage collection of deleted data, are supported in the same way. We integrate InftyDedup with HydraStor [33], a commercial backup system with deduplication, and evaluate its performance in AWS, demonstrating that multiple petabytes can be deduplicated for a couple of dollars. Being highly independent of the local tier, InftyDedup overcomes the limitations of similar state-of-the-art technologies and offers unprecedented scalability. To the best of our knowledge, this is the first application of such solutions to backup systems.

The second contribution is an algorithm for decreasing the financial cost of storing deduplicated data in the cloud tier. It extends InftyDedup by allowing it to move deduplicated data chunks between cloud services dedicated to hot and cold storage. Whereas existing solutions do not address the problem at all or enable some optimizations at the level of data collections (e.g., backups or files), the fact that chunks are deduplicated between backups/files makes them a better unit for optimizations. In InftyDedup, the chunks are moved based on their metadata, notably deduplication reference counts and terse information provided by system administrators on their data collections. Our empirical evaluation of the algorithm shows that mixing storage types can reduce the total financial cost of cloud tiering with deduplication by up to 26–44%.

The rest of the paper is organized as follows. Section 2 gives the background. Section 3 describes the overall architecture and specific algorithms comprising InftyDedup. Section 4 discusses the algorithm for exploiting cold cloud storage for cost minimization. Section 5 presents the experimental results. Section 6 surveys related work. Finally, Section 7 concludes.

2 Background

This section reviews the characteristics of deduplication storage, backups, and cloud services, which are essential to InftyDedup architecture.

2.1 Deduplication Storage

Deduplication is a data reduction technique that avoids writing the same data twice. For data with many duplicates, deduplication reduces the storage capacity requirements of the system [70], increases throughput, and decreases network traffic [3]. Typically, deduplication is implemented in the following steps [79]. Firstly, the data stream is chunked into small immutable blocks of size from 2 KB to 128 KB [71]. Secondly, each block receives a fingerprint, for instance, by computing the SHA-256 hash of the block's data. Finally, the fingerprint is compared with other fingerprints in the system, and if the fingerprint is unique, the block's data is written.

The deduplicated blocks are typically organized in a directed acyclic graph. Each file has its *root block* corresponding to a vertex that references other blocks. The blocks with actual data are leaves of the DAG and keep no references. Blocks with data corresponding to a particular file form a subgraph reachable from the root block representing that file. Therefore, the movement of a deduplicated file to a different tier is effectively the movement of a subset of leaves that are reachable from the root block of the file.

A block can be removed after it is migrated to another system. However, reclaiming storage capacity in the presence of deduplication is nontrivial, as the system must ensure there are no other references to the removed block. Therefore, complex garbage-collecting algorithms that can process blocks' metadata for hours are implemented [36, 69].

The most natural use case for deduplication is backup storage, as most data do not change in consecutive backups. In our research, we leverage the characteristics and lifecycle of backups to decrease the total storage cost.

2.2 Lifecycle of Backups

Typically, backups are created and managed based on assigned retention policies [61]. From the perspective of our research, there are two essential constraints regarding the timing and life cycle of protected data.

On the one hand, the data should be up-to-date and available quickly in case of a disaster. For instance, Zerto reports [82] that their customers achieve Recovery Point Objectives of seconds and Recovery Time Objectives of minutes. To achieve such ambitious objectives, recent data is kept as closely as possible to the infrastructure being recovered.

On the other hand, older versions of backups need to be stored for weeks, months, or even years [65]. As the objective points for older data differ, backups are often moved to cheaper storage after a specific time [55, 83]. Cloud is often chosen to keep the older backups for many reasons, including storing data in a different physical location. The pricing model of cloud storage is also appealing, but as described in the next section, many factors influence the total costs.

2.3 Cloud Storage

The market of cloud storage is mostly shared between three hyperscalers (Amazon Web Services, Microsoft Azure, and Google Cloud) [74], so in our considerations, we assume services offered by the three as a market standard.[1] The portfolio of hyperscalers comprises numerous storage and computing products: from databases, queues, and distributed filesystems to simple storage primitives, such as objects or blocks. Our goal is to minimize the storage cost of backups, so our research focuses on the most affordable products. The lowest price per stored gigabyte is offered by cold archival object stores, which are orders of magnitude cheaper than block devices, as shown in Tab. 1. However, many factors determine the total cost, including fees per request or IO, charges for removing data before meeting the minimal storage duration, and data transfer costs. Accessing data in some types of the coldest storage takes additional time (e.g., 12 hours), but every hyperscaler offers cold storage with instant access [23, 28, 77].

	Amazon Web Services	Microsoft Azure	Google Cloud
Block Storage [$/GB]	0.08	0.15	0.04
Object Storage [$/GB]	0.021	0.0166	0.02
Archival Object Storage [$/GB]	0.004	0.01	0.004
Coldest Archival Object Storage [$/GB]	0.00099	0.00099	0.0012

Table 1: Sample[2] monthly costs of storing blocks and objects in public clouds [5, 7, 26, 27, 49].

Uploading data to the cloud is usually free, whereas the cost of downloading data once a month can outweigh the cost of monthly data storage. In either case, network throughput to the cloud is a major concern. Hyperscalers offer connecting data centers to the cloud directly (e.g., with 100 GbE) [9, 15], but the availability of such networks is limited to specific regions. Alternatively, physical devices can be used for the movement of data [11], but it is rather for niche applications. Therefore, moving terabytes to the cloud can take up days.

2.4 Cloud Computing

The product portfolio of cloud computing services is also versatile. There are virtual machines (e.g., AWS EC2), containers (e.g., AWS ECS), and other services, such as event-driven function execution (e.g., AWS Lambda). Some products are prepared for specific use cases, including machine learning [29] and databases [4, 25].

The pricing model of computation services is typically based on the cost of the lower-level resources. For instance, ECS allows running containers on EC2 instances, so the cost of container execution depends on the amount and size of virtual machines which host the containers [6]. This billing model enables using numerous nodes (e.g., hundreds of servers) for short periods at a very low cost.

What is important for cost reduction, hyperscalers offer so-called *spot instances*, that are virtual machines with a discounted price of up to 90%. Spot instances can be interrupted by a cloud provider at any moment, but the computations interrupted within the first hour are free [13]. The exact price of a spot instance depends on multiple factors (e.g., the momentary demand), but historical data shows that achieving both a very low risk of termination and a significant cost reduction is possible [35]. Virtual machines (including spot instances) can have their local storage (e.g., SSD drives), which is cheaper than network-attached drives but has limited durability as the data are lost if the machine is destroyed or fails.

To minimize the costs of computations, we considered these cloud attributes in InftyDedup architecture, which we describe in the next section.

3 InftyDedup Architecture

InftyDedup moves selected data from local tier systems (i.e., on-premise backup appliances implemented as described in Section 2.1) to the cloud tier. The local tier is expected to have its own deduplication and to be hardware-failure resistant (e.g., by implementing erasure codes or RAID), as it persistently stores local data (e.g., data not selected for tiering). As shown in Fig. 1, the cloud tier stores deduplicated data with necessary persistent metadata, and occasionally executes highly optimized *batch algorithms*.

Before we describe the details of the structures and algorithms, we discuss our study of cloud characteristics (Section 3.1) and the assumptions we made based on them (Section 3.2). After that, we describe the structure of in-cloud data and metadata (Section 3.3), the model of communication between tiers (Section 3.4), and algorithms of deduplication (Section 3.5), garbage collection (Section 3.6), and file restore (Section 3.7).

3.1 Cloud Cost Considerations

We studied the pricing of public clouds to design InftyDedup in line with the current trends. First, we chose product types common for all vendors and compared the pricing models and capabilities of each product with other products of the same vendor. We did not compare pricing between vendors, as our goal was to design cost-efficient architecture for any regular cloud, not choosing a particular vendor.

[1] However, there are numerous innovative services offered by other providers. For instance, the latest trend to decentralize the cloud [62, 63] can help to implement InftyDedup efficiently.

[2] The price of storage products depends on many factors, including region. Each cloud provides many products (e.g., each provider offers more than one cold object store). The prices between providers cannot be compared directly because the products differ. However, there are several categories of cloud storage products similar to the order of magnitude of the price. The table contains list prices as of 2023-01-01.

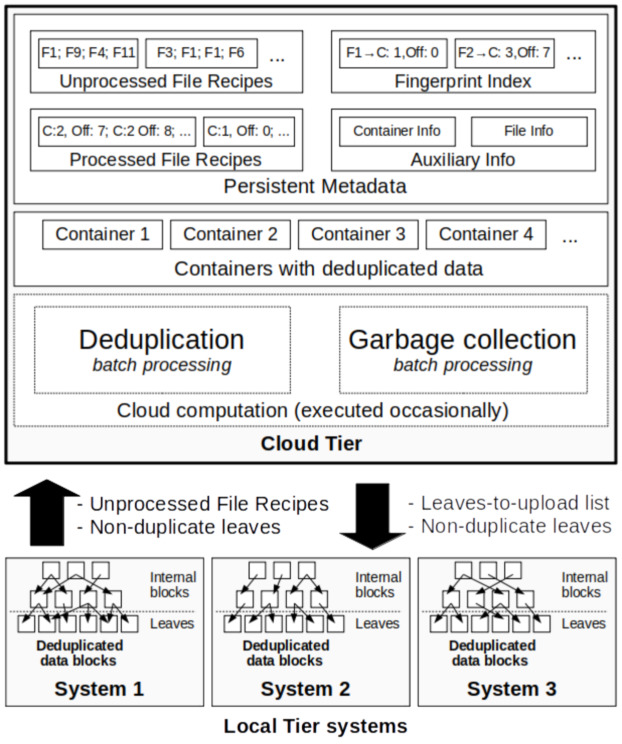

Figure 1: InftyDedup architecture.

Keeping 1 PB of non-deduplicated data in standard cloud object storage costs between $16,600 and $21,000 per month, and between $4,000 and $10,000 for archival object storage with instant access. Therefore, the overall cost of storing data with deduplication, including additional storage for deduplication metadata and costs of computations, must be lower than that to bring any financial benefit.

Assuming a deduplication block size of 8 KB, 10:1 deduplication, and 20 bytes per fingerprint, 1 PB of data requires 262 GB of fingerprints. If new backups of similar size are written each week, over 496 billion fingerprint existence queries to the cloud are needed each month.

Modern architectures of inline deduplication often keep the fingerprint index (or its parts) on SSDs [3,30,48]. Considering a naive approach in which each deduplication query requires a read IO from an SSD drive, at least 190k IOPS are required to perform the necessary queries each month. To estimate the cost, let us consider AWS as an example. The monthly cost of EBS gp3 block storage which provides such an amount of IOPS is $978, and EC2 instances (m5.large) capable of utilizing the IOPS cost $3827. With a total cost of nearly $5000 monthly for just handling deduplication queries, there is still room for a cost benefit from deduplication (depending on the deduplication ratio). Still, the price is significant compared to the cost of storage without deduplication.

These calculations led us to our conclusion that, despite the fact that SSDs provide a high number of random-read IOPS, relying on a random-read-intensive fingerprint index *is not negligibly cheap* in the cloud environment. Although there are techniques that reduce the number of read IOs for traditional sequential workloads [84], their efficiency is decreased for modern non-sequential workloads, which need to be handled in addition to classic sequential workloads, as explained by Y.Allu et al. [2]. Similarly, the efficiency of methods that rely on data locality (like SISL [84]) decreases when data is highly fragmented.[3] Finally, these methods are often not prepared to update block information during deduplication, which is a necessary part of our algorithms for cold storage.

On the other hand, transferring data within the cloud is free of charge, and even the cheapest instance can transfer hundreds of gigabytes per hour [14]. Having the possibility of dynamically scaling resources between zero and hundreds of servers, processing the fingerprint index sequentially with a batch job can be more cost-effective than keeping the fingerprint index online 24/7 or relying on short-lived lambdas [10]. This is particularly true considering up to 10 times cheaper computation using the aforementioned spot instances. This key observation was used when designing the InftyDedup architecture based on assumptions explained in the next section.

3.2 Assumptions and Design Decisions

Our principal assumption is that *our cloud tiering deduplication must be processed outside the local tier* to prevent resource restrictions and enable functionalities like deduplication between many local tier systems. Therefore, all metadata required for deduplication must be stored and processed outside the local tier.

As network throughput between the tiers is limited, *data movement between the tiers should be minimal*. Therefore, only non-duplicate data must be uploaded to the cloud tier. When restoring data, it must be possible to download only data absent from the local tier. However, for efficient disaster recovery, *quick and granular backup restores must be possible*, even when the local tier is unavailable.

The next central assumption is that *batch processing is preferred over streaming processing*. Therefore, the algorithms are executed occasionally (e.g., once a day or week for deduplication and even less frequently for garbage collection). There are multiple reasons for that. Firstly, as our cost analysis of public clouds shows, being prepared for data deduplication 24/7 is not negligibly cheap. Secondly, as explained in Section 2.2, backups are typically moved to the cloud after a specified period, so batch processing can be done without disrupting the data lifecycle. Finally, tiering to cloud with deduplication requires steps that take a significant amount of time: uploading data to the cloud, and running garbage collection in the local tier to reclaim data there. All in all, per-

[3]Fragmentation also concerns restore throughput [40,44]. However, in the case of cloud storage, the read performance scales, and even with random 8 KB reads, the egress traffic cost is equal to the per-request fee.

forming a costly deduplication query with each write brings few benefits in practice, and we decided to use the cheaper option of infrequent batch processing.

Garbage collection in the cloud tier must be cost-aware to ensure that data removal costs are not higher than keeping data for a longer period. Similarly, storing frequently accessed data in cold cloud storage actually increases the costs, so the deduplication and garbage collection algorithms must be extendable with intelligent storage type selection.

Finally, our solution is meant to be suitable for a variety of cloud platforms and providers. Although in our description and evaluation we focus on the most popular hyperscalers, our architecture can be easily adapted to others. In particular, private clouds ensure privacy and compliance, so we verified our solution in our private cloud environment.

3.3 Data and Metadata in Cloud

Based on the assumptions, we designed persistent structures of InftyDedup to be kept in cloud object storage as follows.

The largest structure contains blocks with deduplicated data grouped into *containers*. Selecting the size of containers depends on the cloud pricing, as writing and reading larger containers requires fewer requests but can increase rewriting costs when reclaiming space after garbage collection.

The largest metadata structure contains *file recipes*, which are effectively a list of per-block metadata as they appear in each file. If one block exists in a file multiple times, its metadata also occurs multiple times in its file recipe. There are two types of file recipes. Firstly, there are *unprocessed file recipes* (UFR in short), which are provided by the local tier. UFRs contain the fingerprint of each block, as the local tier does not know the block's cloud location. Later, during deduplication processing, each entry of UFR receives a cloud address of the block it references, so the file recipes are converted to *processed file recipes* (PFR in short). PFRs can be a simple list of cloud addresses or have a tree structure to enable the deduplication of PFR's parts. In the latter case, fingerprints of PFR chunks are added to the fingerprint index, which is described shortly.

The second largest metadata structure is Fingerprint Index (FingIdx in short) which contains a mapping from the deduplication fingerprint of each block to the block's cloud location. FingIdx is expected to be smaller than PFRs, as it contains only one entry per unique fingerprint. FingIdx is bucketed [72] rather than sorted, meaning the fingerprints are divided into thousands of buckets based on a hash function. Such data representation enables optimization of distributed FingIdx processing, as each bucket is small enough to fit into server memory.

There are also a few orders of magnitude smaller structures that keep information per file or container. The metadata structures are compressed to reduce space and network usage.

3.4 Communication between Tiers

The data exchange between the tiers is bidirectional but kept to a minimum, as the network connection between the tiers can easily become a bottleneck. Two types of information are sent from the local tier to the cloud. For each file selected for cloud tiering, the local tier system generates a UFR (a list of fingerprints of all blocks in the file). The UFR is later used as an input to batch deduplication, which generates in return a *blocks-to-upload list* that is, in fact, a list of containers. Each container comprises unique blocks that still need to be uploaded to the cloud tier. Based on the list, the local tier uploads the blocks to the cloud. During a file restore operation, blocks can be later downloaded from the cloud tier.

Therefore, the cloud tier has minimal requirements on the interface of the local tier. It is sufficient that the local tier can generate a UFR and later upload blocks based on the list of fingerprints. The local tier can be composed of multiple systems if each system uses consistent chunking and fingerprinting.

3.5 Batch Deduplication

Batch deduplication (BatchDedup in short) is our distributed method of block deduplication in the cloud. It is expected to be run periodically, in harmony with the schedule of backups and garbage collection in local tier systems. Each execution of BatchDedup is a distributed, fault-tolerant computation that ultimately changes persistent structures kept in the cloud object storage. The computations are divided into steps, and each of the steps comprises smaller jobs that are parallelized and repeated in the event of failure. In our implementation, we used YARN [76] to schedule jobs, and HDFS [64] for reliable storage of temporary data, so the jobs can be run on spot instances as proposed in the AWS guide [12]. The state of computation is maintained by the YARN master node, which can be hosted on a non-spot instance to increase reliability, but even if the entire computation fails, the valid version of metadata always remains in the cloud object storage.

In short, BatchDedup takes UFRs as input, specifies new containers with blocks to be uploaded, waits until the local tier uploads the blocks, and updates persistent metadata. The UFRs are expected to be uploaded to the cloud before BatchDedup is started (partially uploaded UFRs do not take part in the process). The steps are as follows:

Step #1: UFR processing selects blocks that need to be uploaded to the cloud by comparing fingerprints from both UFRs and FingIdx. FingIdx and UFRs are bucketed based on fingerprints, and the buckets are distributed across multiple servers. After that, the fingerprints are compared in batches that are small enough to fit in memory.

Step #2: Container generation splits blocks selected in *Step #1* into containers to generate descriptions for the local tier. Each server processes a subset of blocks, and the blocks are distributed based on their original file (so blocks from the

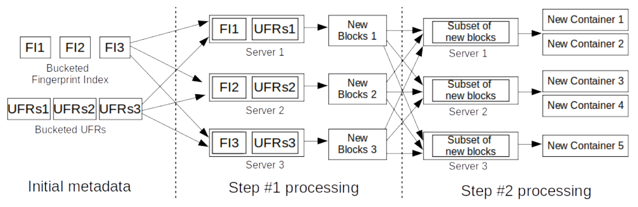

Figure 2: The first two steps of BatchDedup processed in a distributed manner.

same file can be placed in the same container). The blocks are sorted by the order (offsets) in their original files,[4] as preserving the original order makes the latter step of uploading the container easier, and reduces the number of requests for garbage collection and data restores for non-fragmented data.

Step #3: PFR update is conducted after the first two steps, when the block location (its container and offset) is finally known for both new and old blocks. Based on that information, each newly written file receives its PFR.

Step #4: Blocks upload is initiated by the local tier systems. The local tier systems first download the descriptions of new containers (i.e., which blocks should be uploaded to which container). After that, each of the local tier systems uploads the actual data. When the uploads are successfully completed, the in-cloud metadata structures are updated to mark the new files as ready in the cloud.

The first two steps of BatchDedup are depicted in Fig. 2. Similar techniques are used to perform the remaining steps of BatchDedup and garbage collection at scale.

BatchDedup processes FingIdx and all recently uploaded UFRs but does not touch any previously generated PFRs. As UFRs likely contain duplicates, in practice, the size of UFRs is expected to be at least comparable to the size of the whole FingIdx, and with such assumption, processing FingIdx does not dominate the asymptotic cost. Overall, the process is expected to take time: BatchDedup is executed periodically, the computations in Steps #1-#3 take from minutes to hours, and the block upload in Step #4 can even take days, depending on the data volume and network bandwidth. As Step #4 is inevitable in any cloud-tiering solution, the cloud tier alone is not suitable for providing very short RPO. However, as backups are moved to the cloud typically after a specific time, the steps can be scheduled in periods that will not violate the timing constraints of the backup policy.

3.6 Batch Garbage Collection

Batch garbage collection (BatchGC in short) identifies blocks no longer referenced by any PFR and reclaims free space in the containers. BatchGC is expected to be executed periodically but less frequently than BatchDedup. Both algorithms modify the same metadata structures, so they cannot be executed simultaneously. However, file restores are possible at any moment.

PFRs keep the addresses of containers, so rewriting a container requires modifications of PFRs. The cost of processing PFRs is discouraging, as PFRs can be many times larger than FingIdx. However, garbage collection is done only occasionally, so even if it is a few times more expensive than BatchDedup, the overall cost of InftyDedup is not affected that much. Therefore, our primary goal is ensuring scalability, which enables meeting the time constraints of other garbage collection algorithms for deduplication storage [31, 69].

BatchGC comprises the following steps:

Step #1: File removal processes non-removed PFRs to find blocks that are still referenced by at least one file.

Step #2: Container verification checks how many blocks in each container are live. Based on one of the strategies (which we introduce shortly), a set of containers that will be removed or rewritten is selected.

Step #3: Metadata are updated based on the results of *Step #2*. More specifically, new metadata for modified containers are calculated. Some blocks may receive a new address, so new versions of FingIdx and PFRs are also needed.

Step #4: Containers are rewritten to actually reduce space usage. When all newly generated containers are written, the metadata computed in *Step #3* take effect, and old containers are deleted.

Immediate removal of unreferenced data is not always optimal, as rewriting a container in the cloud has a significant cost. Therefore, we investigated three strategies to decide whether a container should be rewritten:

GC-Strategy #1: Reclaim only empty containers. In most cloud services sending a request to remove an entire container is free, so the strategy brings cost reduction (as less capacity needs to be stored) with no additional cost. However, the strategy does not remove containers in which only a fraction of data has been deleted.

GC-Strategy #2: Reclaim containers if the rewrite pays for itself after T days. To determine whether rewriting a container will bring a cost-benefit, the following ratio can be calculated for each container:

$$x = \frac{COST_{rewrite}}{T_{days} * CAPACITY_{to\_be\_reclaimed} * COST_{byte\_per\_day}} \quad (1)$$

Only if $x < 1.0$, rewriting a container is cheaper than storing deleted data from the container for T_{days}. However, picking the proper value of T_{days} is nontrivial. For instance, if T_{days} is the time left until the next BatchGC, the containers are rewritten only if it brings financial benefit before the next chance to remove any data. In many cases, such T_{days} value is too small and will prevent rewriting a container, although rewriting the container would bring a financial benefit in the long run. On the other hand, a large T_{days} value implies frequent rewriting, which can lead to exceeding *Strategy #1* costs.

[4]A block is expected to exist in multiple files or to be repeated within one file. In such a case, only the first appearance is stored in a container.

GC-Strategy #3: Reclaim containers based on file expiration dates. *GC-Strategy #2* can be improved if files contain information about their expiration date (denoted as EXP_{time}). Such information can be provided by the local tier systems in UFRs, if the EXP_{time} results from the backup configuration. Therefore, for each container, T_{days} can be calculated as the maximal EXP_{time} of its blocks (aligned up to the BatchGC schedule). EXP_{time} is expected to increase in time,[5] as new files with later EXP_{time} are stored. However, even with rising EXP_{time}, the cost never exceeds *GC-Strategy #1*, as a non-empty container is rewritten only when it is beneficial.

3.7 File Restore

The cloud metadata format supports straightforward file restores. Each file has its own object, with the key based on the local tier system identifier and file path. Therefore, object storage interface features such as ACLs and per-prefix listings can be used for convenient file management. Based on the PFR, which stores the container address and data offset, the file can be read without accessing the local tier systems. As PFRs are updated during BatchGC, the movement of data between containers during GC does not spoil the reads.

However, egress traffic is a major cost, so restores can be additionally integrated with the local tier for cost reduction. For blocks available locally, the download from the cloud can be omitted. Blocks absent locally can be optionally stored in the local tier system after downloading, as some workloads require reading data again in the near future (e.g., restoring multiple similar VMs). Implementing such local-tier assisted reads requires storing fingerprints in PFRs, which increases the metadata size, but the fingerprints can be easily added and removed from PFRs on-demand in batch algorithms.

4 Cold Storage Utilization

To reduce the cost of storing data in the cloud, InftyDedup can be extended with an algorithm that selects whether a block should be stored in hot or cold cloud storage. We aimed to use cold storage services offering different pricing models than other cloud storage products but comparable durability and latency [28, 77] (otherwise, the movement of data to cold storage negatively affects the recovery time).[6] Therefore, we focused on colder storage which offers a reduced price of storing data but increases the price of restores, and demands a minimal storage period (e.g., 90 days). To utilize the storage effectively, we rely on two additional pieces of information provided with each file (in UFRs):

[5] Theoretically, EXP_{time} can decrease if someone deletes a file before the expiration date. We find such a case rather marginal. In particular, enabling WORM protection [53] prevents such removals.

[6] Our algorithms can also work with the coldest storage services, which lengthens the retrieval process. However, in such case additional information are needed to specify the allowed retrieval time of each file.

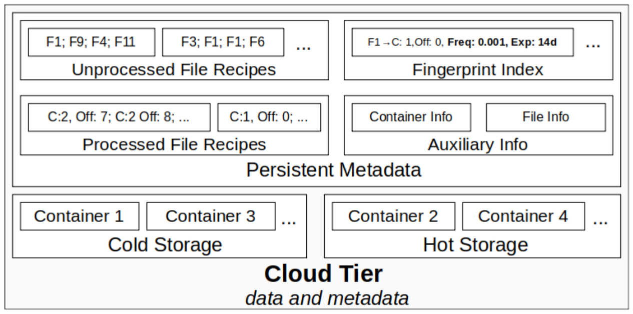

Figure 3: The architecture of data and metadata with two types of data storage (hot and cold). Fingerprint Index is extended.

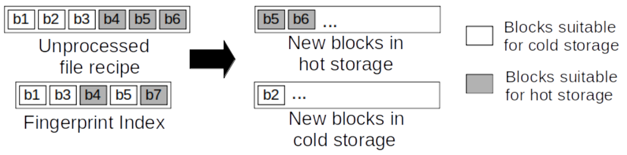
b1 b2 b3 b4 b5 b6
Unprocessed file recipe
b1 b3 b4 b5 b7
Fingerprint Index
b5 b6 ...
New blocks in hot storage
b2 ...
New blocks in cold storage
Blocks suitable for cold storage
Blocks suitable for hot storage

Figure 4: Writing blocks to more than one storage type. Block *b5* is written to hotter storage, although it is already available in colder storage if it brings a cost benefit (e.g., due to expected frequent restores of *b5*).

1. Current expiration date, as in *GC-Strategy #3*.
2. Rough, expected frequency of file restore.

As explained earlier, the expiration time is typically known. The restore frequency is unknown in advance, but assessing the read frequency of a file is common practice for data kept in the cloud. For instance, Amazon explicitly recommends different storage classes for data accessed "once per quarter" and "1-2 times per year" [8]. In the specific case of backups, assessing restore frequency should be possible, as a study of numerous backup jobs [16] suggests that backup domains fall into three categories: those with very frequent restores, sporadic restores, and virtually no restores. Moreover, particular backup policies influence the restore frequency [58], and an upper bound on the restores can be calculated based on restore SLAs. Finally, modern backup software already implements tools that allow viewing historical data on the restore frequency of selected resources [67].

The persistent data and metadata structures are organized as shown in Fig. 3. The process of container writing during BatchDedup and BatchGC is extended to store each block in an appropriate cloud storage type, as shown in Fig. 4. Each block is stored in a storage type for which the following formula has lower value:

$$t = COST_{insert} + (COST_{B/day} + COST_{restore} * FREQ_{restore}) * EXP_{time} \quad (2)$$

In the formula, $COST_{insert}$ depends on cloud pricing, as well as the sizes of the block and its container, as the amortized cost of data insertion is included. $COST_{B/day}$ describes the storage cost of the block. $COST_{restore}$ depends on the data locality, as many blocks can be read with one request, so the upper bound for the $COST_{restore}$ can be calculated as *one request per block* or assessed with a heuristic. $FREQ_{restore}$ and EXP_{time} are inherited from files referencing block, and stored with each block in FingIdx.

However, further adjustments to $FREQ_{restore}$ and EXP_{time} are required. That is because the first decision about storage type must be taken when the block is stored for the first time and block's $FREQ_{restore}$ and EXP_{time} are understated, as more references will come in the future. For instance, a block can be initially stored in cold storage but later it receives more references (and its $FREQ_{restore}$ increases). Vice versa, data with a short EXP_{time} can be kept in hot storage, although a reference with a more distant EXP_{time} will come soon.

Therefore, both $FREQ_{restore}$ and EXP_{time} should be heuristically modified. A heuristic that worked very well in our experiments relies on block reference counts. First, we select a number R of expected references for each block (e.g., a hard-coded value 5 or a value calculated from the system state). Then, we modify $FREQ_{restore}$ and EXP_{time} for blocks that have not reached the expected number based on the formula (e.g., we multiply it by $R - r$, where r is the actual number of references). In the end, $FREQ_{restore}$ and EXP_{time} for newly written blocks are more similar to their future values.

In justified cases, a block can be stored in multiple storage types (e.g., when a block stored in cold storage receives a reference with high $FREQ_{restore}$), but BatchGC will eventually remove the unnecessary copies. Similarly, BatchGC can move a block from one type of storage to another (e.g., when a reference with high restore frequency has been deleted). Generally, during BatchGC, a formula for calculating whether a container should be rewritten considers the potential cost reduction caused by a change of the storage type. A decision on whether rewriting a particular container is profitable must be made for the whole container because rewriting the container also introduces costs. Nevertheless, blocks from one container can be moved to containers in various storage types (Fig. 5).

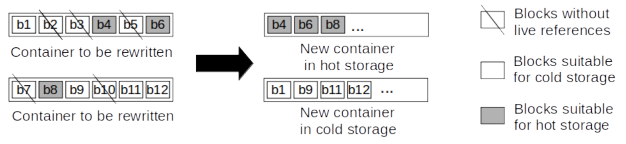
b1 b2 b3 b4 b5 b6
Container to be rewritten
b7 b8 b9 b10 b11 b12
Container to be rewritten
b4 b6 b8 ...
New container in hot storage
b1 b9 b11 b12 ...
New container in cold storage
Blocks without live references
Blocks suitable for cold storage
Blocks suitable for hot storage

Figure 5: Rewriting containers to multiple types of storage.

5 Evaluation

We present our experimental evaluation of InftyDedup in two parts. Firstly, we evaluate the performance and cost of our implementation executed in a public cloud. Secondly, we evaluate our strategies for garbage collection and storage type selection under various workloads.

5.1 Performance Evaluation

To evaluate the performance, we implemented InftyDedup using Apache Hive [24], which we selected as a possible approach to provide portability between different public and private clouds. We present results of the evaluation of our batch algorithms, as uploading containers and restoring data are straightforward object storage operations in which the bottleneck is expected mostly on the network to the cloud (even a naive implementation can saturate 1 GbE network with uploads and restores using a single core).

Our batch algorithms differ greatly from the state-of-the-art tiering to cloud with deduplication techniques, therefore a fair comparison with existing solutions was virtually impossible. Instead, we present the results using publicly available hardware. The evaluation was conducted in AWS using m5d.xlarge instances with 4x vCPU, 16 GiB of RAM, and 1x 150 GB NVMe (which costs less than network-attached EBS). We aimed to use the smallest possible instances (to maximize the horizontal scaling), but in our workloads, the technological stack of Apache Hive was inefficient in utilizing the limited memory of the smallest instances.

The presented experiments used synthetic data with the following characteristics. Each file contained approximately 51 GB (as backup files typically have tens of gigabytes or more [78]) chunked into blocks of approximately 64 KB (the target block size of the deduplication system for which we prepared InftyDedup). The contents of the files are described in each experiment. We present results with synthetically generated data, as our algorithms mostly distribute the data (e.g., based on fingerprints) and later sort the data in small portions, so the exact characteristic of the data (e.g., the initial order of blocks) does not affect the performance much.

5.1.1 Batch Deduplication Processing

We evaluated BatchDedup in configurations varying in size. Each experiment comprised two steps. In the first (initial) step, a large number of files without duplicates is processed to resemble a situation in which new backups are uploaded to the cloud. In the second (incremental) step, a dataset 3x smaller than the initial backup is uploaded (as typically incremental backups are smaller than their corresponding full backups [16]), where 90% of the blocks are duplicates (which matches the expected average daily deduplication ratio [16]). The smallest configuration (8 instances) uploads 3072 files in the first step and 1024 in the second step. In larger configurations, the amount of data to be processed is scaled linearly with the system size. Therefore, the smallest experiment processed metadata of 208 TB data and the largest one of 1.66 PB.

In all configurations, the first step takes between 1h53m and 2h10m (Fig. 6), and the second step takes up to 30m. Overall, the performance scales close to linearly. We analyzed the resource utilization, and the main bottleneck is the CPU, as most of the time its usage is above 95%. The network and

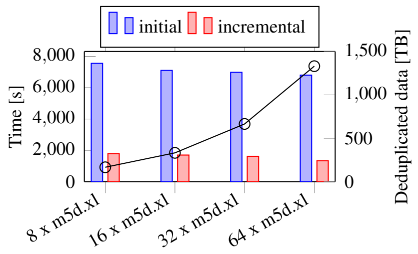

Figure 6: BatchDedup performance. The line and right y-axis show size of deduplicated data.

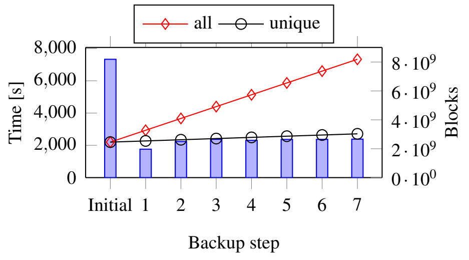

Figure 7: BatchDedup with growing data. The lines and right y-axis present number of blocks pre-deduplication (all) and post-deduplication (unique).

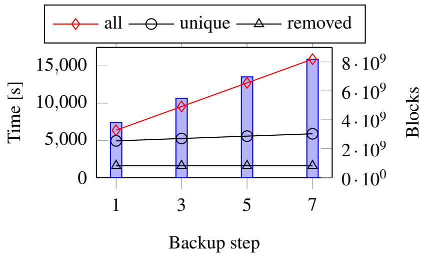

Figure 8: BatchGC performance. After 1-7 incremental steps, data from one incremental step was deleted (the removed blocks on right y-axis).

the local NVMe drive are underutilized, with peak per-node usage of respectively 350 MB/s of network bandwidth and 6% of disk utilization. We expect the computations can be further optimized, but the computation costs are already marginal compared to the cost of data storage. For instance, in the experiment with 32 instances, the second stage eliminates 191 TB of duplicates and costs below $1, which is less than 0.1% of monthly savings on storage. Similarly, the costs of accessing in-cloud metadata during processing are marginal, as both steps require roughly 250K GETs ($0.1) and 20K PUTs ($0.1), and transfer fees within one availability zone are free.

We also conducted a different experiment with multiple steps of incremental uploads in one configuration (8 instances). As shown in Fig. 7, the computation time increases close to linearly with the amount of non-duplicate data which is added to FingIdx in each experiment.

5.1.2 Batch Garbage Collection Processing

First, we evaluated BatchGC by removing a fraction of data uploaded in the first experiment from Section 5.1.1. Specifically, we removed the data uploaded in the first step to resemble removing the oldest backup. The processing took between 61 and 65 minutes in each verified configuration.

BatchGC, unlike BatchDedup, reads all PFRs, so we also verify that the processing time increases close to linearly with the size of both FingIdx and PFRs. The experiment shown in Fig. 8 had multiple incremental steps, and in BatchGC we removed data from one incremental step. The results confirm that, for data with many duplicates, BatchGC is more expensive than BatchDedup. However, BatchGC is expected to be executed less frequently, so both algorithms will have comparable total execution costs.

5.2 Strategies Evaluation

We evaluated how our garbage collection and storage type selection strategies behave in numerous workload simulations. The strategies optimize the costs of storing data for months and years, so we could not conduct these experiments in the public cloud, as it would take too long. Instead, we ran some initial experiments to confirm our understanding of the pricing model and features of the cloud, and based on the results, we implemented a simulator. The simulator calculates costs based on cloud pricing of storage, requests, transfer, and other factors, like the minimal storage duration.

Each experiment was conducted in many configurations of workload characteristics and system parameters. We present aggregated (minimal, maximal, and average) results, with values normalized to the result with the minimal cost.

5.2.1 Workload Characteristics

Our simulator allowed specifying the following factors to evaluate various backup workloads:

Data source was selected from the following two sets. Firstly, we generated synthetic workloads in which a given fraction of data was *modified* and *deleted* each day. Both types of modifications were applied in variable length *stream-contexts* (of size from 1 to 1024 blocks), so a given number of consecutive blocks was modified at once. The introduction of the stream-contexts was necessary, as data modified in small contexts are more fragmented, so reading data requires more requests. Secondly, FSL traces [73] were used, as they are real-world datasets that contain information on how the data of multiple users change over the years.

Retention policy specifies how long each file (backup) is stored. We analyzed guidelines related to retention policies [1,32,75] to generate realistic policies. Typically, each type of backup is stored for a longer time than its backup period (e.g., weekly backups are kept for four weeks). In our experiments, daily backups are kept for one week, weekly backups are kept for a month, monthly backups are kept for a year, and yearly backups are kept for five years. Based on that, we came up with three different policies: *keepAll* policy in which all types of backups are stored in the cloud, *dailyExcluded* in which daily backups are excluded (so only backups stored for at least a month are kept in the cloud), and *dailyOnly* in which only daily backups are kept in the cloud. The garbage collection was, in turn, executed every 7, 30, or 90 days. In all experiments, the simulation covered a period of 5 years.

Read patterns remarkably affect the total cost of ownership of data in the cloud. Unlike for writing data, we found no collected read traces for backup data. Similarly, there are no precise guidelines that describe typical backup read patterns. Therefore, we adapted a model in which each file is read with a given probability and verified the full spectrum of potential values.

5.2.2 Garbage Collection Strategies Evaluation

To evaluate how the proposed garbage collection strategies perform in different workloads, we conducted experiments with the pricing model of *AWS S3 Standard* as *hot* storage and *Glacier Instant Retrieval* as *cold* storage.[7] Experiments in which the storage types are mixed based on our strategy are denoted as *mixed*. Garbage collection strategies are denoted as follows: Strategy #1 is denoted as *onlyEmpty*, *less*{*25; 50; 75; 99*} denotes Strategy #2 with the T_{days} parameter such that the behavior is equivalent to reclaiming space when less than *25 / 50 / 75 / 99* percent of container capacity is used by live data, and Strategy #3 is denoted as *costBased*.

First, we evaluated a case in which there were no reads. As shown in Fig. 9, *onlyEmpty* strategy achieved the worst results. For *cold* and *mixed* storage, *costBased* strategy gave significantly better results than others (on average 1.4%-23%), whereas for hot storage (where the rewrite cost is marginal) it gave similar results to *less99*.

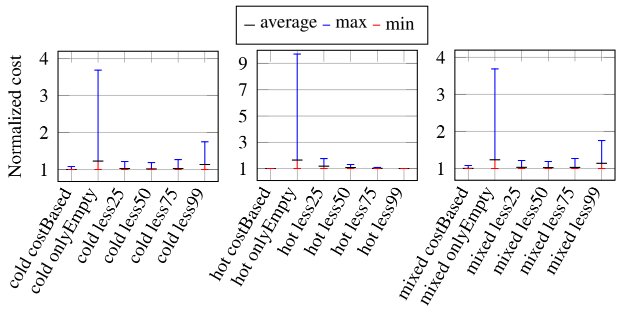

Figure 9: Garbage collection with different strategies.

In the next set of experiments, which included reads (with patterns explained in Section 5.2.3) and *mixed* storage, there are more differences between the strategies (Fig. 10). On average, *costBased* strategy is only 2.2% better, but comparing the worst cases, the difference is 24%. The analysis of the number of containers that are rewritten, deleted empty, or remain live at the end of the experiment, confirms that *onlyEmpty* has the largest number of containers that are live (Fig. 11).

[7]At the moment of writing, cold storage had 4x/25x more expensive PUT/GET requests, 5.25x times cheaper storage costs, the minimum storage duration was 90 days, and an additional per-gigabyte retrieval cost for cold storage was equal to the fee for 3000 GET requests.

The analysis of garbage collection strategies led to the question of how container sizes affect the costs, as smaller containers increase the probability of removing the entire container but also increase the number of PUT requests needed to store data initially or during container rewriting. As shown in Fig. 12, for *costBased* strategy, the lowest average cost is with 16 MB containers (4 MB and 64 MB are respectively 4.5% and 2% more expensive). The smallest, 1 MB containers were the most expensive, even with the *onlyEmpty* strategy, because of the cost of initial container generation (Fig. 13). Especially in *cold* storage, the cost of PUT requests is high (up to 40% of all costs with 1 MB containers).

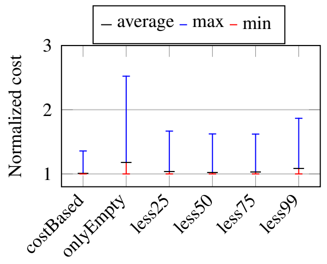

Figure 10: Garbage collection strategies with reads.

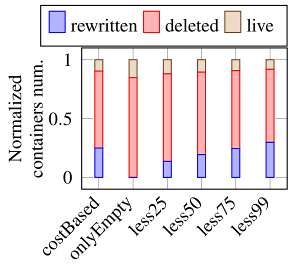

Figure 11: Breakdown of containers number.

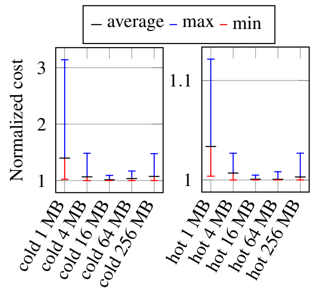

Figure 12: Garbage collection with varying container sizes.

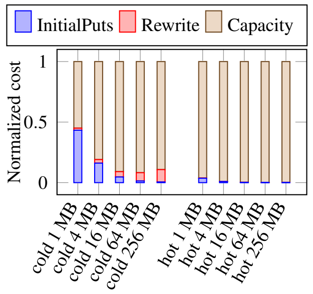

Figure 13: Cost breakdown with varying container sizes.

5.2.3 Storage Type Selection Evaluation

We evaluated our storage type selection strategies in workloads with varying read frequencies. For each experiment, there are 4 synthetically generated sets of files, and each set has a different read frequency: once a month, once a year, once a year with 1% probability, and once a year with 0.1% probability. All 4 sets were written together, just as in a storage system that keeps files with varying read frequencies. The experiments were conducted in series, and in each series, the read frequency was scaled by a factor from 0.001 to 10. Therefore, cases in which reads are virtually nonexistent, cases in which reads dominate the total cost, and cases in-between were evaluated. A real-world ratio between backup and recovery jobs is typically 100 : 1 [17] but varies depending on the system [16]. In our experiments, the ratio of backups to recoveries for scale factor 0.01 is $70-700:1$ (mean $=216:1$) depending on the retention policy. Therefore, we expect results with scale factors 0.01 and 0.1 to reflect a typical use-case.

As shown in Fig. 14, on average *mixed* strategy gives 55% cost savings compared to *cold* if there are many reads and 70% compared to *hot* if there are hardly any reads. The breakdown of newly created containers (Fig. 15) confirms that data ends up in cold storage when there are hardly any reads, and in hot storage reads are frequent. The cost breakdown (Fig. 16) confirms that *mixed* strategy balances the high storage cost in *hot* and the expensive reads in *cold*.

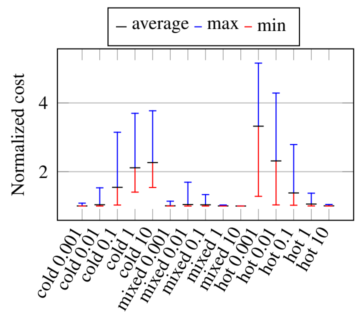

Figure 14: Storage type selection depending on the read frequency.

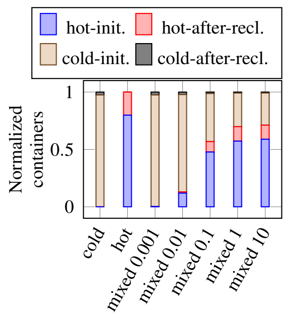

Figure 15: Containers created initially and after reclamation.

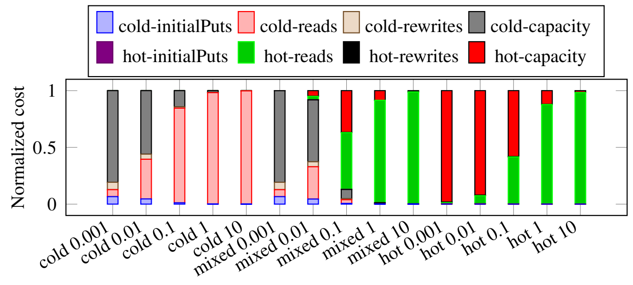

Figure 16: Cost breakdown with varying read frequencies.

We also evaluated how the changes in the predicted reference number affect the cost. Fig. 17 presents the normalized cost, depending on the selection of the expected reference number. Without predicting that more references will come in the future, the cost is higher on average by 11% (worst case 289%) compared to predicting 5-10 references, so we confirmed that predicting the number of references brings a significant cost reduction. The results with 3-10 references are very similar, so the slight inaccuracies in the expected number of references do not change the results much.

The mixed strategy depends on the expected frequency of reads, which may be incorrectly assessed. We conducted experiments with a significant prediction error (the value was underestimated and overestimated ten times). Even with such a large estimation error, the results are close to perfect (Fig. 18). Therefore, in all other experiments, we assumed perfect estimation to facilitate studying other experimental parameters.

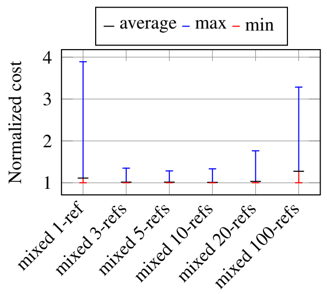

Figure 17: Cost of storing data depending on the expected number of references.

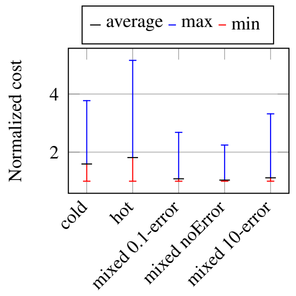

Figure 18: Cost of storing data depending on the error of frequency prediction.

5.2.4 Different Public Clouds

To confirm that our strategies are generally applicable to public clouds, we repeated most of the experiments with the pricing model of Google Cloud and Microsoft Azure. As our evaluation shows, mixing cold and hot storage reduces the costs for all three major providers (Fig. 19). The noticeable differences in gain between the cloud providers follow from the different ratios of costs, especially the cost of storing data and egress traffic. On average, keeping data only in hot storage is 61% more expensive, and keeping data only in cold storage is 30% more expensive than using the mixed strategy.

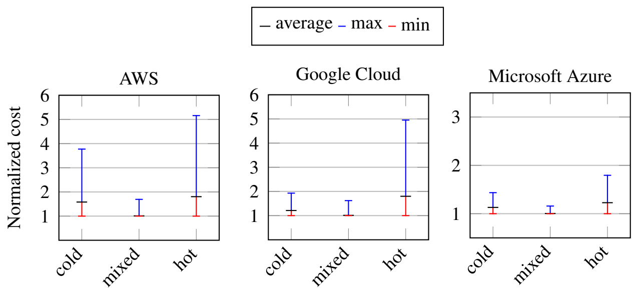

Figure 19: Storage type selection in different public clouds.

5.2.5 FSL Traces

Finally, we verified our strategies using the FSL traces [73]. Specifically, we used all data available with 64 KB chunking in *homes snapshots* dataset. The traces contain metadata of files chunked during writing, but they have no information about the read pattern. Therefore, for each user, we verified how our storage type selection works with a varying number of reads (restoring each backup with a frequency from 0.0001 to 1 time a month). As shown in Fig. 20, at the extreme read frequencies the mixed strategy keeps almost all the data in the cheapest of the two storage types. However, if the number of reads is in between, the mixed strategy works better than keeping data in a single type of storage, as depending on the data characterization a different decision should be made for each block. In particular, the characterization of the reference number of each block is important, as frequently referenced blocks are accessed more often. Therefore, mixing storage types can outperform keeping the data in one storage type, decreasing the cost by 26%-44%. This result shows that even when the restore frequency of each file is known in advance, relying on selecting one storage type can be significantly more expensive than using our mixed strategy.

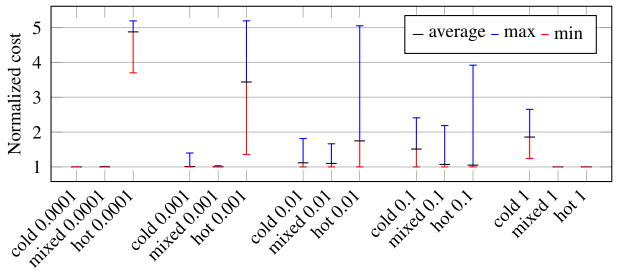

Figure 20: Total costs in experiments with FSL traces.

6 Related Work

Hierarchical storage is widely adapted, as storage devices offer a trade-off between cost, capacity, and performance [56]. Systems with storage tiers are actively researched, and in recent years, many publications have referred to tiering in the cloud [39, 46, 60]. Hsu et al. [38] propose an AI-based prediction model for classifying whether data is cold or hot. Liu et al. [45] describe an online algorithm for two-tier cloud storage, which works with no prior knowledge of future access frequencies. However, less attention is given to tiering techniques in the cloud in the context of deduplication.

MUSE [80] is a framework focused on providing SLA for deduplicated data focused on the primary storage use case, which is a different use case than storing backups. DD Tier [34] is tiering with deduplication that performs its computation in the local tier, hence imposing fundamental restrictions and limitations. First, deduplicating data from more than one local tier system is impossible, as each system performs deduplication on its own. Furthermore, all or at least a large fraction of metadata is needed locally to operate. Therefore, metadata are stored in both tiers, which not only increases storage capacity usage but also forces downloading a large amount of metadata to recover even a single file. Moreover, the resources for metadata storage and processing of the local tier are limited. As locally stored metadata can consume hundreds of terabytes of local storage, the size of the cloud tier is limited (to 2x the size of the local tier). Alike, deduplication and garbage collection algorithms cannot overuse scarce local resources, especially RAM, CPUs, and disk I/Os. Therefore, perfect hashing is used to decrease memory requirements below 3 bits per fingerprint, so extending it with techniques similar to our storage type selection is very difficult.

DD Tier introduces a technique for estimating how much space will be freed from the local tier after moving data to the cloud, and in recent years, significant research attention has been paid to the problem of selecting files for efficient data removal and migration in systems with deduplication [37, 42, 51]. As long as such methods do not require storing additional metadata locally, they can be used with InftyDedup.

A large number of publications explore security threats of deduplication in the cloud. Therefore, several methods of preventing particular attack types were proposed [21, 41, 43, 81]. Alike, side channels leaking information from deduplication storage have been studied [19, 20]. Most threats arise from the situation in which a public cloud provider implements deduplication between users. InftyDedup is meant to be used by a single organization, and writing to InftyDedup requires accessing the local tier, so the situation is much different. Still, some organizations might find the deduplication side-channels as a threat within the organization, and adding security mechanisms to InftyDedup can be required. Moreover, users of InftyDedup may not trust the cloud provider, so the local tier can encrypt data before storing them in the cloud. The structure of the data (information on block sizes and which blocks are referenced by which files) is still exposed to allow the computations, but the situation is similar in other tiering with deduplication solutions, as restoring blocks reveals the structure of files.

7 Conclusions

We presented InftyDedup, a novel, cloud-native approach to tiering to cloud for a storage system with deduplication. Compared to the state of the art, our architecture does not impose any limit on the size of the cloud tier and supports deduplication from multiple local tier systems. We implemented InftyDedup for a commercial storage system (HydraStor) and evaluated it in a public cloud (AWS). The evaluation confirmed the desired scalability of deduplication handling: our batch algorithms, designed to reduce cloud costs and harness dynamic resource allocation, were able to process metadata of multi-petabyte data collections for a couple of dollars.

To further decrease the cost of cloud storage, we proposed an extension to InftyDedup which moves chunks between hot and cold cloud stores based on their anticipated access patterns. Its evaluation with real-world traces showed that our deduplication-specific heuristic for adjusting the expected read frequency, which takes into account block reference counts, decreased the costs on average by 11%, and the overall solution achieved 26%–44% reductions. The algorithm requires minimal input from a system administrator and was demonstrated to retain its cost benefits even when the administrator's estimations were inexact.

Acknowledgments

We sincerely thank our shepherd Philip Shilane and the anonymous reviewers for helping us improve our paper significantly.

References

[1] Acronis. Retention rules: how and when they work. 2022. https://kb.acronis.com/content/68304.

[2] Yamini Allu, Fred Douglis, Mahesh Kamat, Ramya Prabhakar, Philip Shilane, and Rahul Ugale. Can't We All Get Along? Redesigning Protection Storage for Modern Workloads. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018.

[3] Yamini Allu, Fred Douglis, Mahesh Kamat, Philip Shilane, Hugo Patterson, and Ben Zhu. Backup to the future: How workload and hardware changes continually redefine data domain file systems. *Computer*, 50(7):64–72, 2017.

[4] Amazon Web Services, Inc. Amazon aurora - fully mysql and postgresql compatibile managed database service. 2023. https://aws.amazon.com/rds/aurora/?did=ap_card&trk=ap_card.

[5] Amazon Web Services, Inc. Amazon ebs pricing. 2023. https://aws.amazon.com/ebs/pricing/.

[6] Amazon Web Services, Inc. Amazon elastic container service pricing. 2023. https://aws.amazon.com/ecs/pricing/.

[7] Amazon Web Services, Inc. Amazon s3 pricing. 2023. https://aws.amazon.com/s3/pricing/.

[8] Amazon Web Services, Inc. Amazon s3 storage classes. 2023. https://aws.amazon.com/s3/storage-classes/.

[9] Amazon Web Services, Inc. Aws direct connect locations. 2023. https://aws.amazon.com/directconnect/locations/.

[10] Amazon Web Services, Inc. Aws lambda - faqs. 2023. https://aws.amazon.com/lambda/faqs/.

[11] Amazon Web Services, Inc. Aws snow family faqs. 2023. https://aws.amazon.com/snow/faqs/.

[12] Amazon Web Services, Inc. Best practices for cluster configuration. 2023. https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-plan-instances-guidelines.html.

[13] Amazon Web Services, Inc. Billing for interrupted spot instances. 2023. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/billing-for-interrupted-spot-instances.html.

[14] Amazon Web Services, Inc. General purpose instances - network performance. 2023. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/general-purpose-instances.html#general-purpose-network-performance.

[15] Amazon Web Services, Inc. Link aggregation groups. 2023. https://docs.aws.amazon.com/directconnect/latest/UserGuide/lags.html.

[16] George Amvrosiadis and Medha Bhadkamkar. Identifying trends in enterprise data protection systems. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, 2015.

[17] George Amvrosiadis and Medha Bhadkamkar. Getting back up: Understanding how enterprise data backups fail. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016.

[18] Associate Research Director Andrew Smith, Research Manager; Archana Venkatraman. Enterprise data growth and adoption of cloud applications challenge traditional data protection strategies. 2021. https://afi.ai/r/US48310921.pdf.

[19] Frederik Armknecht, Colin Boyd, Gareth T Davies, Kristian Gjøsteen, and Mohsen Toorani. Side channels in deduplication: Trade-offs between leakage and efficiency. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017.

[20] Andrei Bacs, Saidgani Musaev, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Dupefs: Leaking data over the network with filesystem deduplication side channels. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, 2022.

[21] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Annual international conference on the theory and applications of cryptographic techniques*, 2013.

[22] Christophe Bertrand. Esg research report: The evolution of data protection cloud strategies. 2021. https://www.esg-global.com/research/esg-research-report-the-evolution-of-data-protection-cloud-strategies.

[23] Sriprasad Bhata. Introducing azure cool blob storage. Microsoft, 2016. https://azure.microsoft.com/en-us/blog/introducing-azure-cool-storage/.

[24] Jesús Camacho-Rodríguez, Ashutosh Chauhan, Alan Gates, Eugene Koifman, Owen O'Malley, Vineet Garg, Zoltan Haindrich, Sergey Shelukhin, Prasanth Jayachandran, Siddharth Seth, et al. Apache hive: From mapreduce to enterprise-grade big data warehousing. In *Proceedings of the 2019 International Conference on Management of Data*, 2019.

[25] Google Cloud. Cloud sql. 2023. https://cloud.google.com/sql.

[26] Google Cloud. Cloud storage pricing. 2023. https://cloud.google.com/storage/pricing#north-america.

[27] Google Cloud. Disk pricing. 2023. https://cloud.google.com/compute/disks-image-pricing#disk.

[28] Google Cloud. Storage classes. 2023. https://cloud.google.com/storage/docs/storage-classes#descriptions.

[29] Google Cloud. Vertex ai - google cloud's unified ml platform. 2023. https://cloud.google.com/vertex-ai.

[30] Biplob Debnath, Sudipta Sengupta, and Jin Li. Chunkstash: Speeding up inline storage deduplication using flash memory. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[31] Fred Douglis, Abhinav Duggal, Philip Shilane, Tony Wong, Shiqin Yan, and Fabiano Botelho. The logic of physical garbage collection in deduplicating storage. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017.

[32] Druva. What is backup retention policy? how is it implemented? 2023. https://docs.druva.com/Knowledge_Base/inSync/Client/010_FAQ/What_is_Backup_Retention_Policy%3F_How_is_it_implemented%3F.

[33] Cezary Dubnicki, Leszek Gryz, Lukasz Heldt, Michal Kaczmarczyk, Wojciech Kilian, Przemyslaw Strzelczak, Jerzy Szczepkowski, Cristian Ungureanu, and Michal Welnicki. Hydrastor: A scalable secondary storage. In *Proccedings of the 7th Conference on File and Storage Technologies (FAST 09)*, 2009.

[34] Abhinav Duggal, Fani Jenkins, Philip Shilane, Ramprasad Chinthekindi, Ritesh Shah, and Mahesh Kamat. Data domain cloud tier: Backup here, backup there, deduplicated everywhere! In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019.

[35] Nnamdi Ekwe-Ekwe and Adam Barker. Location, location, location: exploring amazon ec2 spot instance pricing across geographical regions. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2018.

[36] Fanglu Guo and Petros Efstathopoulos. Building a high-performance deduplication system. In *2011 USENIX Annual Technical Conference (USENIX ATC 11)*, 2011.

[37] Danny Harnik, Moshik Hershcovitch, Yosef Shatsky, Amir Epstein, and Ronen Kat. Sketching volume capacities in deduplicated storage. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, 2019.

[38] Ying-Feng Hsu, Ryo Irie, Shuuichirou Murata, and Morito Matsuoka. A novel automated cloud storage tiering system through hot-cold data classification. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018.

[39] Ryo Irie, Shuuichirou Murata, Ying-Feng Hsu, and Morito Matsuoka. A novel automated tiered storage architecture for achieving both cost saving and qoe. In *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*, 2018.

[40] Michal Kaczmarczyk, Marcin Barczynski, Wojciech Kilian, and Cezary Dubnicki. Reducing impact of data fragmentation caused by in-line deduplication. In *Proceedings of the 5th Annual International Systems and Storage Conference*, 2012.

[41] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. Dupless: Server-aided encryption for deduplicated storage. In *22nd USENIX Security Symposium (USENIX Security 13)*, 2013.

[42] Roei Kisous, Ariel Kolikant, Abhinav Duggal, Sarai Sheinvald, and Gala Yadgar. The what, the from, and the to: The migration games in deduplicated systems. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, 2022.

[43] Jingwei Li, Chuan Qin, Patrick P. C. Lee, and Jin Li. Rekeying for encrypted deduplication storage. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016.

[44] Mark Lillibridge, Kave Eshghi, and Deepavali Bhagwat. Improving restore speed for backup systems that use inline chunk-based deduplication. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013.

[45] Mingyu Liu, Li Pan, and Shijun Liu. To transfer or not: An online cost optimization algorithm for using two-tier storage-as-a-service clouds. *IEEE Access*, 7:94263–94275, 2019.

[46] Yaser Mansouri and Abdelkarim Erradi. Cost optimization algorithms for hot and cool tiers cloud storage services. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018.

[47] Dirk Meister and André Brinkmann. Multi-level comparison of data deduplication in a backup scenario. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009.

[48] Dirk Meister and André Brinkmann. dedupv1: Improving deduplication throughput using solid state drives (ssd). In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[49] Microsoft. Azure storage pricing. 2023. https://azure.microsoft.com/en-us/pricing/details/storage/blobs/#pricing.

[50] Jaehong Min, Daeyoung Yoon, and Youjip Won. Efficient deduplication techniques for modern backup operation. *IEEE Transactions on Computers*, 60(6):824–840, 2010.

[51] Aviv Nachman, Gala Yadgar, and Sarai Sheinvald. Goseed: Generating an optimal seeding plan for deduplicated storage. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020.

[52] Netapp. Cloud tiering documentation. 2023. https://docs.netapp.com/us-en/cloud-manager-tiering/pdfs/fullsite-sidebar/Cloud_Tiering_documentation.pdf.

[53] Veritas NetBackup. About netbackup worm storage support for immutable and indelible data. 2020. https://www.veritas.com/support/en_US/doc/25074086-143197427-0/v143250065-143197427.

[54] Veritas NetBackup. Veritas netbackup™ deduplication guide. 2021. https://www.veritas.com/support/en_US/doc/25074086-146020141-0/v145698641-146020141.

[55] Veritas NetBackup. Aws cloud storage with veritas netbackup. 2022. https://www.veritas.com/content/dam/www/en_us/documents/white-papers/WP_aws_cloud_storage_with_netbackup_long_term_retention_solution_V1259.pdf.

[56] Junpeng Niu, Jun Xu, and Lihua Xie. Hybrid storage systems: A survey of architectures and algorithms. *IEEE Access*, 6:13385–13406, 2018.

[57] Myoungwon Oh et al. Design of global data deduplication for a scale-out distributed storage system. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018.

[58] Oracle. Backing up file-system data. 2023. https://docs.oracle.com/cd/E91325_01/OBADM/osb_filesystem_backup.htm.

[59] Michael Paul. Cloud object storage deep dive – part two, implementation. Veeam Software, 2021. https://www.veeam.com/blog/cloud-object-storage-implementation.html.

[60] Ajaykrishna Raghavan, Abhishek Chandra, and Jon B Weissman. Tiera: Towards flexible multi-tiered cloud storage instances. In *Proceedings of the 15th International Middleware Conference*, 2014.

[61] Santhosh Rao, Nik Simpson, Michael Hoeck, and Jerry Rozeman. Magic quadrant for enterprise backup and recovery software solutions. 2021. https://www.gartner.com/en/documents/4003661.

[62] Meet Shah, Mohammedhasan Shaikh, Vishwajeet Mishra, and Grinal Tuscano. Decentralized cloud storage using blockchain. In *2020 4th International conference on trends in electronics and informatics (ICOEI)(48184)*, 2020.

[63] Pratima Sharma, Rajni Jindal, and Malaya Dutta Borah. Blockchain-based decentralized architecture for cloud storage system. *Journal of Information Security and Applications*, 62:102970, 2021.

[64] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[65] Veeam Software. Step 7. configure long-term retention. 2021. https://helpcenter.veeam.com/docs/backup/vsphere/backup_job_gfs_vm.html?ver=110.

[66] Veeam Software. 2022 data protection trends. 2022. https://go.veeam.com/wp-data-protection-trends-2022.

[67] Veeam Software. Restore operator activity. 2023. https://helpcenter.veeam.com/docs/one/reporter/restore_operator_activity.html?ver=110.

[68] Hewlett Packard Enterprise storage experts. Hpe cloud bank storage: A data protection solution you can bank on. 2017. https://community.hpe.com/t5/Around-the-Storage-Block/HPE-Cloud-Bank-

Storage-A-Data-Protection-Solution-You-Can-Bank/ba-p/6965903.

[69] Przemyslaw Strzelczak, Elzbieta Adamczyk, Urszula Herman-Izycka, Jakub Sakowicz, Lukasz Slusarczyk, Jaroslaw Wrona, and Cezary Dubnicki. Concurrent deletion in a distributed content-addressable storage system with global deduplication. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013.

[70] Zhen Sun, Geoff Kuenning, Sonam Mandal, Philip Shilane, Vasily Tarasov, Nong Xiao, et al. A long-term user-centric analysis of deduplication patterns. In *2016 32nd Symposium on Mass Storage Systems and Technologies (MSST)*, 2016.

[71] Zhen "Jason" Sun, Geoff Kuenning, Sonam Mandal, Philip Shilane, Vasily Tarasov, Nong Xiao, and Erez Zadok. Cluster and single-node analysis of long-term deduplication patterns. *ACM Transactions on Storage (TOS)*, 14(2):1–27, 2018.

[72] Liyin Tang and Namit Jain. Join strategies in hive. *Hive Summit*, 2011.

[73] Vasily Tarasov, Amar Mudrankit, Will Buik, Philip Shilane, Geoff Kuenning, and Erez Zadok. Generating realistic datasets for deduplication analysis. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012.

[74] Lionel Sujay Vailshery. Cloud infrastructure services vendor market share worldwide from 4th quarter 2017 to 4th quarter 2021. 2022. https://www.statista.com/statistics/967365/worldwide-cloud-infrastructure-services-market-share-vendor/.

[75] Global Data Vault. Data backup: Developing an effective data retention. 2023. https://www.globaldatavault.com/blog/data-retention-policy-and-scheduled-backups/.

[76] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013.

[77] Marcia Villalba. Amazon s3 glacier is the best place to archive your data – introducing the s3 glacier instant retrieval storage class. Amazon Web Services, Inc., 2021. https://aws.amazon.com/blogs/aws/amazon-s3-glacier-is-the-best-place-to-archive-your-data-introducing-the-s3-glacier-instant-retrieval-storage-class/.

[78] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. Characteristics of backup workloads in production systems. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST 12)*, 2012.

[79] Wen Xia, Hong Jiang, Dan Feng, Fred Douglis, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou. A comprehensive study of the past, present, and future of data deduplication. *Proceedings of the IEEE*, 104(9):1681–1710, 2016.

[80] Jianwei Yin, Yan Tang, Shuiguang Deng, Bangpeng Zheng, and Albert Y. Zomaya. Muse: A multi-tierd and sla-driven deduplication framework for cloud storage systems. *IEEE Transactions on Computers*, 70(5):759–774, 2021.

[81] Haoran Yuan, Xiaofeng Chen, Jin Li, Tao Jiang, Jianfeng Wang, and Robert H Deng. Secure cloud data deduplication with efficient re-encryption. *IEEE Transactions on Services Computing*, 15(1):442–456, 2019.

[82] Zerto. Maximize recovery achieve your best rtos and rpos. 2020. https://www.zerto.com/wp-content/uploads/2020/08/Fastest-RTO-and-RPO-in-the-Industry_Guide.pdf.

[83] Zerto. Deploy & configure zerto long-term retention amazon s3. 2022. https://www.zerto.com/page/deploy-configure-zerto-long-term-retention-amazon-s3/.

[84] Benjamin Zhu, Kai Li, and Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST 08)*, 2008.