Sketching Volume Capacities in Deduplicated Storage

Danny Harnik and Moshik Hershcovitch, *IBM Research;* Yosef Shatsky, *IBM Systems;*
Amir Epstein, *Citi Innovation Lab TLV;* Ronen Kat, *IBM Research*

https://www.usenix.org/conference/fast19/presentation/harnik

This paper is included in the Proceedings of the
17th USENIX Conference on File and Storage Technologies (FAST '19).

February 25–28, 2019 • Boston, MA, USA

978-1-939133-09-0

Open access to the Proceedings of the
17th USENIX Conference on File and
Storage Technologies (FAST '19)
is sponsored by

Sketching Volume Capacities in Deduplicated Storage

Danny Harnik
IBM Research

Moshik Hershcovitch
IBM Research

Yosef Shatsky
IBM Systems

Amir Epstein*
Citi Innovation Lab TLV

Ronen Kat
IBM Research

Abstract

The adoption of deduplication in storage systems has introduced significant new challenges for storage management. Specifically, the physical capacities associated with volumes are no longer readily available. In this work we introduce a new approach to analyzing capacities in deduplicated storage environments. We provide sketch-based estimations of fundamental capacity measures required for managing a storage system: How much physical space would be reclaimed if a volume or group of volumes were to be removed from a system (the *reclaimable* capacity) and how much of the physical space should be attributed to each of the volumes in the system (the *attributed* capacity). Our methods also support capacity queries for volume groups across multiple storage systems, e.g., how much capacity would a volume group consume after being migrated to another storage system? We provide analytical accuracy guarantees for our estimations as well as empirical evaluations. Our technology is integrated into a prominent all-flash storage array and exhibits high performance even for very large systems. We also demonstrate how this method opens the door for performing placement decisions at the data center level and obtaining insights on deduplication in the field.

1 Introduction

The rise of all-flash storage arrays has also brought deduplication technology to the forefront and many prominent all-flash systems now support deduplication across entire systems or data pools (e.g., [2–5, 7, 9]). While this shift helped reduce storage costs, it also created new storage management challenges for storage administrators. This work focuses on technologies and tools for managing storage capacities in storage systems with deduplication.

Storage Management and Deduplication: Volume placement and capacity management were challenging yet well understood management tasks before deduplication was introduced. A storage volume needs to be allocated appropriate resources and connectivity. In large data centers, spanning multiple storage arrays, managing where to place volumes optimally is a tricky task. It involves satisfying two main measures that characterize a volume: its capacity and workload (IOPS/throughput). The main problem tackled in this paper is that once deduplication is brought in to the equation, the capacity of a volume is no longer a known quantity. Hence a storage administrator is left without clarity about one of the main resources that he needs to manage.

Our solution serves a number of appealing applications that are otherwise hard to accomplish in a deduplicated setting. In a recent paper titled “99 deduplication problems" [31], Shilane et al. present some burning deduplication related problems that need to be addressed. Our methods turn out to be helpful in solving three of the five problem classes that are discussed in this paper (it was actually 5 rather than 99 problems...). In particular, our solution is relevant to the issues of: 1) understanding capacities, 2) storage management and 3) tenant chargeback.

Why is managing volumes with deduplication hard? Volume level capacity statistics are the primary tools for managing the system capacity. However, in a system with deduplication those statistics are no longer naturally available. There are two different aspects that are the main reasons for this:

1. The first is that once cross-volume deduplication is enabled, it is no longer clear which volume owns what data. This brings up a conceptual question of what should actually be reported to the storage administrator? In Section 3.2 we discuss in detail and define what capacity can be attributed to a volume and why this information is useful. More importantly, we point out that a critical and well-defined question about a volume is how much space will be freed in case this volume was removed from the system (termed the *reclaimable* space of a volume in this paper). Note that with deduplication enabled, one could possibly remove the largest volume in the system, yet

*Work was conducted while at IBM Research.

not free up a single byte of user data from the storage.

2. The second aspect is a pure computational challenge: once we decide what we want to report, how can this number be calculated in a typical architecture of a storage system with deduplication. There is a fundamental difference between capacity statistics in the presence of deduplication and traditional capacity statistics (in traditional statistics we also include those of storage that supports compression only without deduplication).
   Traditional statistics are all additive and can be aggregated per each volume - i.e., hold a counter of how much space was held by a volume and update this on every write/overwrite/delete operation.[1] On the other hand, in the case of storage with deduplication, the capacity statistics of a single volume do not depend solely on what happens in this specific volume, but rather could be affected by any write operation to any volume in the system (as long as they are part of the same deduplication domain). Moreover, the reclaimable statistic of a group of volumes is not additive - i.e. the reclaimable space of removing two different volumes does not equal the sum of the reclaimable quantities of the two volumes separately. As a result, the methods for calculating such statistics are much harder than traditional stats, and near impossible to do if considering any arbitrary combination of volumes.
   That being said, it is clear that a storage array holds all of the information required to actually compute these numbers. It is just that the sheer amount of metadata that needs to be analyzed to produce these statistics is too large to actually analyze with acceptable resources (CPU and memory).

Our Work - Sketching Capacities: We present a novel approach to produce capacity statistics in a deduplicated storage system for any single volume or any combination of volumes. Moreover, our approach can answer complex placement questions, e.g., not only do we answer how much space would be reclaimed when moving an arbitrary set of volumes out of a system, we can answer how much space this set would take up at a different deduplicated storage system (which holds different content than the original system).

At the core of our solution is the decision to forgo the attempt to produce accurate statistics. Rather, we settle for estimations of the capacity statistics as long as we can gauge the accuracy of these estimations. We borrow techniques from the realm of streaming algorithms in which the metadata of each volume is sampled using a content-based sampling technique to produce a so-called *sketch* (or *capacity sketch*) of the volume. The key is that the sketch is much smaller than the actual metadata, yet contains enough information to evaluate the volumes capacity properties in conjunction with any other set of volume sketches. To illustrate this, consider a storage system holding 1 PB of data. In order to manage such a huge amount of data, a system has to hold a very large amount of metadata which could be on the order of 10 TB (depending on the specific design). In contrast, our sketch data for such a system takes under 300 MB, which makes our statistics calculations easily manageable. Part of this technology is integrated into the IBM FlashSystem A9000/A9000R.

Main Contributions: In the paper we provide details of the technique, its accuracy statement and a description of our implementation. In our design, sketch data is collected by the storage system and pulled out of the system to an adjacent management server where it is analyzed and presented to the administrator. Our implementation includes the following:

- Provides reclaimable capacities and attributed capacities for any volume in the system.
- Supports queries of these capacities on any arbitrary group of volumes within a deduplication domain (an option that to the best of our knowledge is not available in any system to date).
- In a multi-system environment, we answer how much physical space such a volume/group would consume if it were to be migrated to another deduplicated storage system (with different content).

The implementation is optimized for high performance, providing a real-time user-experience. After initial extraction of the sketch and ingestion into the sketch analyzer, we can answer queries in well under a second.

The high performance is also key for providing next level optimization of management functions. We present an example of a greedy solution for multi-system space reclamation optimization. The algorithm creates a migration plan from a full source system onto the other systems in a way that optimizes overall capacity utilization.

2 Background and Related Work

Deduplication is a form of compression in which duplicate chunks of data are replaced by pointers to the original repeating data chunk. This practice can greatly reduce the amount of physical space required to store a large data repository, depending on the amount of repetitiveness of the data chunks. Deduplication is typically used in concert with traditional “local" compression. Unlike deduplication, which looks for repeating data across an entire repository of data, in compression a single data chunk or block is compressed on its own (typically using a techniques such as Zip [13, 25, 35]). To measure data reduction, we use the convention by which the data reduction ratio is the size of the data after reduction divided by the size of data before reduction (so 1 means no compression and close to 0 is highly compressible). Deduplication can consider fixed size data chunks or variable size

[1] It should be noted that while this is conceptually easy, many times it collecting these statistics requires complex engineering, especially in highly distributed storage systems.

chunks. In our work we refer to systems that use fixed size chunks (we use chunks of size 8KB), but our techniques can generalize nicely to variable sized chunking as well.

The common technique for performing deduplication is via chunk fingerprinting. Namely, for each data chunk a hash of its content is calculated, creating an identifier of the data. If a strong cryptographic hash is used, then for all practical purposes this hash is considered a unique identifier of the content. Duplications are found by holding a database of chunk fingerprints and finding repeating fingerprints across the entire data set.

Related Work: Variations of content-based sampling have been deployed in the context of deduplication for various tasks. Mainly for identifying deduplication potential in data sets that have not yet been deduplicated (e.g. [17, 23, 24, 34]), for finding repetitive streams/files as part of the actual deduplication process (e.g. [10,26]) or for automatic routing of data in a distributed setting [12, 14, 18, 19]. Accuracy guarantees for data reduction estimations have been explored in [22–24, 34]. These works focused on analyzing data that has not been deduplicated for assessing their potential data reduction and sizing of the storage required to store them.

In contrast, our aim is to address gaps for reporting and management of data that *has already been deduplicated*, which prior works do not address. We use similar techniques, but the application presents different requirements and hence we deploy slightly different practices. The idea of content based sampling dates back to the classical algorithm of Flajolet and Martin [16] and was thoroughly studied in the context of streaming algorithms for estimating the number of distinct elements in a data stream, a problem which is similar to estimating the amount of distinct data chunks in a data set. We use a variation of a method introduced by Gibbons and Tirthapura [20] and by Bar Josef et al. [11]. In the deduplication context, a similar technique was used by Xie et al. [34] who use filtering according to hash values. The main technical difference between the sketch that they use and ours is that Xie et al. always keep a bounded sample of hashes, a practice that cannot provide adequate accuracy in our context. There is also a significant difference in what the sketches are eventually used for. We also extend the methods to handle the combination of compression with deduplication and provide a different approach to the accuracy analysis (See Section 4). In the storage realm, variants of sketches on the space of LBA addresses have also been used by Wires at al. [33] to estimate amount of exclusive blocks in a snapshot and by Waldspurger et al. [32] for simulating cache behaviors.

There have also been studies tackling the problem of freeing space from a deduplicated system [30] or balancing of content between several deduplicated systems [15, 27]. These suggest various heuristics to perform such optimizations, but largely avoid the question of how to actually learn the interplay between various volumes and assume that this is a given quantity and is computed as a preprocessing step. As such, our work is very suitable to work together with any of these methods.[2]

3 Sketches and their Use

Capacity Sketches: The main idea of capacity sketches is to choose samples of the metadata according to the respective data content. At its core, the sampling technique is very simple: for each data chunk, examine its fingerprint (the hash of its content) and include it in the sketch only if it contains k leading zeros for a parameter k (namely, the k most-bits of the hash are all zeros). So if, for example, $k = 13$ and the fingerprints are random, then on average a $\frac{1}{2^{13}} = \frac{1}{8192}$ fraction of the data chunks participate in the sketch and the rest are ignored. We refer to 2^k as the *sketch factor* and in our implementation we typically set the sketch factor to be 8192. This choice was made by balancing the tradeoffs between the required resources to handle the sketches vs. the accuracy which they provide (See Section 4).

Denote by $\mathcal{S}$ a data set that corresponds to a volume/group/system, denote by $Sketch_{\mathcal{S}}$ the set of hashes which were included in the sketch of the data set and denote by $Space_{\mathcal{S}}$ the physical space required for storing $\mathcal{S}$ ($Space_{\mathcal{S}}$ is the value that we aim to estimate). Denote by $Written_{\mathcal{S}}$ the amount of logical data written by the user (prior to compression and deduplication). For each hash $h \in Sketch_{\mathcal{S}}$ in the sketch we also hold the chunk's compression ratio – denoted by $CompRatio(h)$. We estimate the number of unique chunks in $\mathcal{S}$ by the amount of chunks that participate in the sketch times 2^k, namely by $2^k \cdot |Sketch_{\mathcal{S}}|$. The estimated amount of space required for storing this data set in a clean system is:

$$\widehat{Space_{\mathcal{S}}} = 2^k \cdot \sum_{h \in Sketch} CompRatio(h) \cdot ChunkSize$$

In such a case the estimated data reduction ratio (combining both deduplication and compression) is :

$$ReductionRatio_{\mathcal{S}} = \frac{\widehat{Space_{\mathcal{S}}}}{Written_{\mathcal{S}}}$$

This is the basis for a sketch-based estimation of a single set.[3] In the following sections we describe how to use sketches for estimating volume level statistics in an existing deduplicated storage system. In Section 4 we discuss the accuracy of these estimations as a function of the size of the sketch and the sketch factor.

[2] Note that these heuristics are typically based solely on the knowledge of the pair-wise deduplication relations between volumes but ignore the numbers about a combination of a larger number of volumes. This is mainly because computing such quantities is extremely taxing. Our techniques open the door to utilizing much more information than just pair-wise information.

[3] Note that the same estimation method for $\widehat{Space_{\mathcal{S}}}$ holds for variable sized chunking, only that then $ChunkSize$ is also a function off the hash h.

Using Sketches for Data Inside a Deduplicated System: When discussing the statistics of volumes or groups with respect to a storage system, additional challenges arise. Unlike the stand-alone case, the statistics of a volume or group do not depend solely on the contents (or the sketch) of this single data set. Rather, they depend on the contents of all of the data in the system and may change even though the volume itself observed no changes at all. To facilitate efficient computation of the statistics for a live existing system, we maintain at all time a full system sketch (denoted $Sketch_{FULL}$) representing all of the data in the system. We also collect further parameters in the sketch. Specifically, for each fingerprint $h \in Sketch_{\mathcal{S}}$ the sketch holds:

- Reference count - Denoted $Ref(h,\mathcal{S})$ is the number of times the data chunk with fingerprint h was written in the data set by the user.
- Physical count - Denoted $Phys(h,\mathcal{S})$ is the number of physical copies stored during writes to the data set $\mathcal{S}$. In contrast to reference count which refers to the virtual space, this counter refers to how many virtual copies of this chunk where eventually written in the physical space. Note that there are a number of reasons for a chunk to have more than a single physical copy in a system. Most obvious is deduplication opportunities that were not identified. But sometimes this is done out of choice, e.g., as a means to limit the size of the reference counters, or the choice to forgo a deduplication opportunity for avoiding extensive data fragmentation.

With this additional information we can calculate the statistics of a volume or group as part of a larger system. It should be noted that a real deduplicated system may also hold quite a bit of *unhashed* data (data written at IO's smaller than a single chunk size, or misaligned with the deduplication chunk alignment). We use various techniques to account for such data, but this is out of the scope of this paper. We now describe the main estimations that we calculate as well as their motivation.

3.1 Reclaimable Capacity

As mentioned in the introduction, a key product of our method is the ability to accurately predict how much physical space would be freed from a system if a volume or a group of volumes were to be removed from it. Note that the reclaimable capacity is an inherently non-linear quantity. For example, if a system contains just two identical volumes, then the reclaimable capacity of each of the volumes separately is essentially 0, yet the reclaimable capacity of their combination amounts to the system's entire space which is very different than the sum of their respective reclaimable numbers. As such, some deduplicated storage vendors do not produce such a number at all. Others (e.g. [8]) resort to reporting how much *unique* data a volume holds.[4] This number is additive and is easier to maintain, but can be very misleading when a volume holds internal deduplication, a situation that is magnified when trying to estimate the reclaimable of a group of volumes.

Our strategy for estimating the reclaimable capacity consists of "subtracting" the sketch of the data set being examined from the full system sketch as follows:

Calculate Reclaimable

Input: $Sketch_{\mathcal{S}}$, $Sketch_{FULL}$
$Reclaimable = 0$
for $h \in sketch_{\mathcal{S}}$ do:
  if $(ref(h,\mathcal{S}) == ref(h,FULL))$ then
    $Reclaimable \mathrel{+}= CompRatio(h) \cdot Phys(h,FULL)$
$Reclaimable = Reclaimable \cdot SketchFactor \cdot ChunkSize$

While the algorithm above gives the general idea, there are some additional subtleties that need to be addressed. For instance, each chunk held in the system also holds some metadata associated with it. While this is typically much smaller than the data itself, it can amount to a significant portion of the space, especially for highly compressed or deduplicated data. So reclaimable space should account for metadata space that is released when a chunk is removed (whether it was a physical chunk or a reference). Another subtlety is the fact that it is hard to gauge if a physical chunk would be released when its physical count is two or more. Handling this requires additional information from the system, but for the most part tends to account for a very small portion of the physical space.

3.2 Attributed Capacity and Data Reduction Ratios

Unlike the reclaimable statistic which is very clearly defined, it is not straightforward to define the data reduction ratio of a volume, or what capacity is owned by a volume. This is because data is shared across volumes and has no clear owner. Still, there are a number of motivating reasons to define and support such numbers. The first reason is the possibility to do fair chargeback of tenant capacities for service providers (see discussion in [31], Section 6). Another reason is to allow the storage administrator to understand the data reduction properties of volumes – how much is a volume involved in deduplication? how much does it gain from compression? Such knowledge can allow better planning of storage capacities (e.g., an administrator can learn what data reduction to expect from her Databases), and better placement decisions (e.g., a volume that has no data reduction saving can be placed in a system that does not support data reduction).

To that end, we define a measure that we call the *attributed* capacity of a volume and a breakdown of its space savings to deduplication and compression. Our definition follows a fair sharing principle: Data which is shared among a number of volumes will receive its proportionate share in attributed capacity. For example, if a data chunk is only referenced twice, once in each of two volumes, then the space to hold

[4] Unique data counts only data chunks that have reference count = 1.

this chunk is split evenly in the attributed capacity of the corresponding volumes. If it has 3 references, 2 originating from volume A and one from volume B, then its space is split in a $\frac{2}{3}$ and $\frac{1}{3}$ fashion between volumes A and B respectively. Note that there is no single correct definition of attributed capacity, but rather a choice of what the vendor deems as fair sharing. Our sketches approach can accommodate more or less any definition.[5]

For the breakdown to deduplication and compression we define the following: Deduplication savings are an estimate of what the savings would have been if compression was not deployed. For compression we give a different estimate, basically answering how much additional space was saved after deduplication was performed. This does not answer the question of how much space savings we would gain if only compression was performed (without deduplication). In order to answer the latter question, one needs to sample the virtual space of data (as described in [22]), rather than sample the fingerprint space which is what our sketch does.

The following method is used for attributed space and deduplication savings:

Calculate Attributed

Input: $Sketch_{\mathcal{S}}$, $Sketch_{FULL}$
$Attributed = 0$
$DedupeOnly = 0$
for $h \in sketch_{\mathcal{S}}$ do:
  $Attributed \mathrel{+}= \frac{ref(h,\mathcal{S})}{ref(h,FULL)} \cdot CompRatio(h) \cdot Phys(h,FULL)$
  $DedupeOnly \mathrel{+}= \frac{ref(h,\mathcal{S})}{ref(h,FULL)} \cdot Phys(h,FULL)$
$Attributed = Attributed \cdot SketchFactor \cdot ChunkSize$
$DedupeOnly = DedupeOnly \cdot SketchFactor \cdot ChunkSize$

3.3 Insights on the Achieved Deduplication

An additional benefit for our methodology is the ability to collect drill down statistics regarding deduplication and compression at a very low price. For example, we collect statistics regarding the effectiveness of the deduplication in the storage system. Another set of interesting statistics is the correlation between deduplication and compression, this can be done at a volume granularity, as well as at the single chunk granularity (e.g. is there a correlation between the reference count of a chunk and its compression ratio?). A summary of such insights is sent back to us via a call-home mechanism, and will serve as a mechanism for collecting information from the field about the deduplication properties of real data in the field in order to improve the deduplication process and design.

Explaining deduplication behavior: It is not uncommon for gaps between the customer expectation of deduplication and compression effectiveness versus the reality. In many cases the gap arises from the data written to the system. The ability to correlate hashes between volumes very quickly, and identify correlations and anti-correlations can provide the explanation of why certain deduplication and compression ratios are achieved.

[5] Having said that, it makes sense to use a definition that allows for correct accumulation of attributed capacity between any set of volumes.

3.4 Cross System Capacity Estimations

In a data center environment spanning a number of storage systems, the question of space reclamation from one system is accompanied by the question of where to move the volume to? The goal here is to provide insight into the overall capacity management of the data center rather than just managing a single system; namely, can I gain capacity by moving data between systems? The use of sketches allows to answer the capacity aspect of such complex "what if" questions. Specifically, how much capacity would be freed when moving a volume from system A to another system, and how much capacity would this volume potentially consume in each of the target migration systems. This question can be answered given a sketch for a data set $\mathcal{S}$ and a full sketch of a target system using the following method:

Calculate Space in Target System

Input: $Sketch_{\mathcal{S}}$, $Sketch_{TARGET}$
$TargetSpace = 0$
for $h \in sketch_{\mathcal{S}}$ do:
  if $h \notin Sketch_{TARGET}$ then
    $TargetSpace \mathrel{+}= CompRatio(h)$
$TargetSpace = TargetSpace \cdot SketchFactor \cdot ChunkSize$

Data Center Space Optimizations: Such a cross system estimation presents a strong tool for performing optimizations across multiple systems. Without clarity of the capacity required to store a volume on various systems, such decisions are made in the blind. Our technique provides clarity and allows us to explore optimizations and advanced placement, rebalancing and space reclamation decisions. In Section 6.4 we present an example of such an optimization for data center level space reclamation.

4 Accuracy Guarantees

A crucial property of sketches is the ability to make concrete statements regarding its accuracy, hence allowing decision makers to make educated decisions with confidence while taking into account known error margins. To this end, we provide a mathematically proven statistical accuracy theorem. We also evaluate this guarantee empirically and see that the actual estimations indeed behave according to the mathematical statement (see Section 6.3).

As is common in statistical statements, we have two parameters: the error (or skew) denoted by ε and the confidence denoted by δ. The formal statement says that the estimation will be off by error greater than ε with probability no larger than δ. It turns out that the key parameter relating between

ε and δ is the size of the physical space being estimated – whether it is reclaimable, attributed, or the space required for storing $\mathcal{S}$ in a deduplicated and compressed system. In the following statement we simply use $Space_{\mathcal{S}}$ to denote this size.

Theorem 1. *Let $\mathcal{S}$ be a data set whose physical space (after deduplication and compression) is $Space_{\mathcal{S}}$ and let $\widehat{Space_{\mathcal{S}}}$ be a sketch-based estimation of this space using a random hash function. Then the probability of over-estimation:*

$$Pr[\widehat{Space_{\mathcal{S}}} > (1+\varepsilon)Space_{\mathcal{S}}] < \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{(1+\varepsilon)}}\right)^{\frac{Space_{\mathcal{S}}}{ChunkSize \cdot SketchFactor}}$$

and the probability of under-estimation:

$$Pr[\widehat{Space_{\mathcal{S}}} < (1-\varepsilon)Space_{\mathcal{S}}] < \left(\frac{e^{-\varepsilon}}{(1-\varepsilon)^{(1-\varepsilon)}}\right)^{\frac{Space_{\mathcal{S}}}{ChunkSize \cdot SketchFactor}}$$

The theorem follows from the classical multiplicative variant of the Chernoff Bound (e.g., in [29]). However, we needed to reprove a more generalized form of this bound in order to capture variations including compression ratios, variable sized chunking, reclaimable and attributed. Note that [34] use a different method to achieve their accuracy guarantee. Their guarantee relies on the estimation of Bernoulli variables by a normal distribution, which for smaller numbers may add some noise. Our estimation avoids this and turns out to be slightly more conservative in the guarantees it provides.

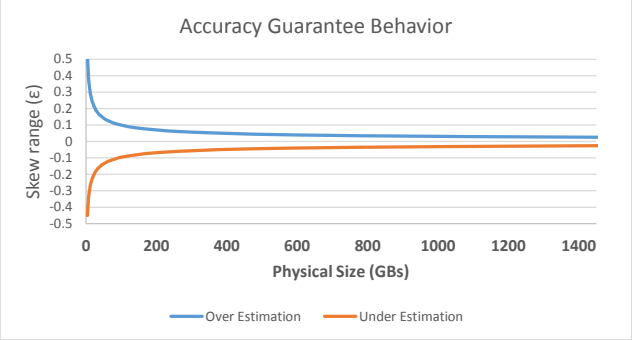

Figure 1: The behavior of the accuracy guarantee ε as a function of the physical size for a fixed choice of the confidence $\delta = \frac{1}{2000}$. The smallest values in the graph are at 4 GBs.

Figure 1 depicts the behavior of our bound for a fixed choice of confidence δ. Note that we need to consider a δ which is small enough to account for a large number of volumes and sets being tested. For example, if we evaluate 750 volumes, using $\delta = \frac{1}{100}$ is not sufficient, as we expect on average 7.5 volumes to exceed the error that corresponds to $\delta = \frac{1}{100}$. In our evaluations we typically use $\delta = \frac{1}{2000}$, but this should be adapted depending on the circumstances.

Using Theorem 1: The goal of the mathematical guarantee is to produce an estimation together with a range for which we can say with confidence that the actual value resides in. For example, we estimate that the reclaimable of a volume is 200GB +/- 14 GB. To this end, for a given fixed confidence parameter δ we create an inverse lookup table that on input $Space_{\mathcal{S}}$ returns the corresponding $\varepsilon(Space_{\mathcal{S}})$. We can then return a +/- value of $Space_{\mathcal{S}} \cdot \varepsilon(Space_{\mathcal{S}})$. One subtlety, however, is that the bound above is dictated by the actual physical space $Space_{\mathcal{S}}$ whereas we only have the estimation of this value $\widehat{Space_{\mathcal{S}}}$. Therefore, in order to get an accurate ε one has to find what is the smallest $Space_{\mathcal{S}}$ such that $\widehat{Space_{\mathcal{S}}} > Space_{\mathcal{S}} - Space_{\mathcal{S}} \cdot \varepsilon(Space_{\mathcal{S}})$. Note that this is important only for evaluating over-estimations, since it turns out that the function $Space_{\mathcal{S}} \cdot \varepsilon(Space_{\mathcal{S}})$ is monotonically increasing with $Space_{\mathcal{S}}$. We also note that the difference between this method and simply returning $\widehat{Space_{\mathcal{S}}} \cdot \varepsilon(\widehat{Space_{\mathcal{S}}})$ is only noticeable for small volumes.

The Effect of the Sketch Factor: The sketch factor appears in Theorem 1 as a divisor of the actual space. This means that, for example, moving from sketch factor 2^k to sketch factor 2^{k+1} will shift the same accuracy guarantees to volumes that are double the size. On the other hand, the amount of sketch data to handle will be cut in half. We arrived at our choice of $k = 8192$ by taking a sketch factor that is high enough to ensure good performance and low overheads, yet still give acceptable accuracy guarantees.

Handling Small Volumes: The accuracy we can achieve when estimating small volumes is limited (and more precisely, volumes of small physical space). For example, the guarantee for a volume of physical size 50GB is only $\varepsilon = 0.14$. There are a number of points to consider here:

- The virtual capacity of the volume is important in understanding if the estimation is worthwhile. For example, for an estimation in the range of 2GB we can only say with confidence that the value is in the range between 0.5GB and 4.2GB. This is not saying much if the volume's virtual size is 4GB but contains very valuable information if the volume's virtual size is 100GB. In the latter case it means that the reclaimable space of the volume is just a small fraction of the original volume (and is a bad candidate for space reclamation).
- Small logical volumes gain very little from sketches (except in very extreme cases, e.g. if the volume is very compressible). It could be argued that small volumes are not very interesting from a capacity management perspective since they have very little impact. On the other hand, grouping several small volumes together to form a larger group is highly recommended. The sketch merge functionally accommodates this and accuracy improves as the size of the merged data set increases.

5 Architecture and Implementation

We turn to describe our actual implementation and integration of the sketch-based capacity analytics for a production storage system. Our overall strategy is to pull the sketch data out of the storage system onto an adjacent management server where the sketch data is analyzed and the outcome is displayed to the storage client (See Figure 2). The choice to do the analysis outside of the storage box has a number of reasons. For one, this avoids using CPU and memory resources in the storage that could otherwise be spent on serving IO requests. But more importantly, it is the optimal location for managing cross system placement options (such as the ones discussed in Section 3.4). As such, our design has two separate components: the *sketch collection* embedded in the storage system and the *sketch analysis* running on an external server. We next describe our design and implementation of these two components.

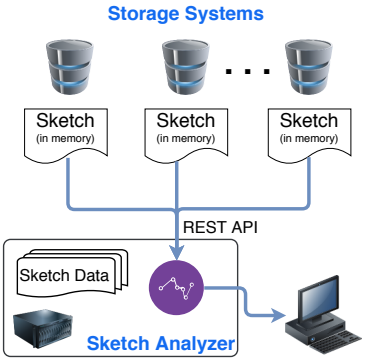

Figure 2: The general sketches support architecture.

5.1 Sketch Collection

There are multiple approaches that can be used for sketch collection. One can use a bump in the wire approach that directs all sketch information as data is written to a sketch collection mechanism. However, in such a design, support for updating the sketch should also be added for deletion or data overwrites which makes this harder to maintain. In addition, in a highly distributed storage system such as ours, it is unclear where the sketch collector should run and if it should be likewise distributed. Another approach is to do an offline metadata scan to extract the actual sketches. In this approach all metadata in the system must be read, and since the metadata is typically paged in and out of memory, such a scan can be relatively slow and may have a negative performance impact on the storage system by introducing additional reads from disk. Instead, we use a third variant which is somewhat of a medium of the two aforementioned approaches.

Our design has the following key principles:

- All sketch data is held in memory at all times - This allows to retrieve this data swiftly, but more crucially avoids adding IOs to the disk backend for sketch update and retrieval.
- Each process is in charge of the sketch data for its jurisdiction - Our storage system is highly distributed, with hundreds of processes working in parallel to serve IOs. Each process is in charge of serving IOs for slices of the entire virtual space. In our design each slice has its own sketch which is maintained by the owning process.
- The sketch portrays the state of a slice at a point in time - The sketch data is held and managed as an integral part of the metadata of a slice, and therefore there is no history of writes and deletes as part of the sketch.

These principles are achieved using the following methodology: During sketch retrieval, if the metadata of a slice happens to be in memory, then the sketch data for this slice is extracted directly from the slice metadata. Whenever the metadata of a slice is paged out of memory, its sketch data is kept “alive" in a designated memory area and retrieved from there.

For the act of sketch extraction, a central process contacts all processes and retrieves their respective sketch data. These are streamed out of the system to the adjacent server. Note that the sketch data in the storage is always in distributed form, and the aggregation of this data only takes place outside the system once the sketch data has been extracted. It should also be noted that as in many cases for distributed systems, the extracted sketch does not actually reflect a single point in time. In our case, the sketch provides a fuzzy state, e.g., when we actually obtain the sketch for the last slice, the sketch in the system for the early slices might have changed. This is an inaccuracy that we are willing to tolerate since storage systems are dynamic and we cannot expect to freeze them at a specific state. That being said, the fact that we can serve the sketches quickly from memory is a considerable advantage as it can reduce the time window in which the sketches are extracted.

Extensive performance tests were run to ensure that our sketch collection and retrieval mechanisms do not interfere with the performance of the storage system and the effects are unnoticeable even at peak performance.[6] In order to minimize the memory footprint, we hold the sketches in packed format and the size of each element in the sketch is limited to 19 bytes. This includes volume information, compression ratio, reference and physical counters and 8 Bytes for the actual hash value, truncated down from 20 bytes of a full SHA1 hash (we take 8 bytes that do not include the leading zero bits). We point out that while an 8 byte hash is not enough for avoiding collisions in a deduplicated system, it is well

[6] The retrieval process is throttled to ensure it does not interfere with the systems IO chores.

suited for achieving high accuracy estimations.[7] Overall this means that for 1PB of user data the sketch data will amount to approximately 300MB on average.

5.2 The Sketch Analyzer

The statistics provided by our sketch analysis do not reflect a real time state of the system. Rather, they reflect a fuzzy state of the storage over the sketch retrieval time duration. Additionally, we can provide the resulting statistics only after the sketch has been fully transferred and processed by the analyzer. That being said, we invest quite a bit of effort to make our sketch analysis as fast as possible, for a couple of reasons: 1) In order to be interactive and support online statistics queries on arbitrary volumes groups by the storage administrator. Our aim is to provide a real-time user experience for this, and indeed we manage to answer all queries well within one second; and 2) Using our tools for performing optimizations typically entails performing a very large number of queries, and therefore the fast processing of queries on sketches allows such optimizations to be feasible.

Recall that the sketch extracted from the storage arrives as a stream in its distributed form. It contains hardly any aggregation at all, and therefore the first phase that we need to do is to ingest it (including sorting to volumes and aggregation). The next phase is the actual analysis using the methods described in Section 3.

The Ingest Phase: The first phase of the process is therefore an ingestion phase. In a nutshell, the sketch contains a stream of hash values along with their respective compression ratios, a local reference count (within the slice), an indication if this was written as a physical copy or just a reference, and finally the name of the owning volume. For each volume we need to collect all of its relevant hashes while merging and aggregating multiple appearances of hashes. The same applies toward creating the full system sketch.

In order to accommodate this, we create two types of data structures at ingest time:

1. **The full system table:** An open addressing hash table holding all of the hashes seen in the full system (including the compression ratio, reference counts and physical copies count). We use statistics from the system to estimate the size of this table and allocate the memory for this table accordingly.
2. **Volume level structures:** We hold a temporary B-Tree for each volume in the system which aggregates the hashes that appeared in the respective volume (along with the reference count for each volume). At the end of the ingest phase we replace each B-Tree with an immutable sorted array, which is a more space efficient data structure which will support faster volume merges.

[7] Under the randomness of SHA1, a false collision of 8 bytes in the sketch data would occur on average once on every 256PB of logical written data.

Our sketch analyzer is designed as a micro-service and uses a REST API to receive the raw sketch data and to then answer statistics queries.

The implementation of the REST interface is in Python, but the core data structures are implemented in C++, using the C-types interface. The implementation in C++ is critical for achieving the performance that we require and for minimizing the memory utilization of the sketch analyzer. For space and time optimization we leveraged the following implementation details:

Space optimization: instead of holding the full hash in the B-Trees and volume arrays, they are held only once in the full-system table, and in the volume level structures we only store a pointer to the entry in the full table (the pointer takes just 5 bytes rather than 8 bytes for the sketch hash).

Sketch distribution and concurrency: Each of our data structures is divided in to 16 partitions, each handling on average $\frac{1}{16}$ of the hashes in the sketch, depending on the first four bits of the hash value. This allows for analysis concurrency as each (hash range) partition can be independently analyzed. Keep in mind that due to the randomness of the hash function, it is expected that each such partition will receive a fair share of the load.[8] In addition, the partitioning provides easy sparsification of the sketch. Depending on the query, a higher sketch factor than 8192 may be sufficient for allowing faster computations (by handling smaller sketches). Simply working with j out of the 16 partitions can easily allow us to work with a sketch factor of $\frac{16}{j} \cdot 8192$.

The Analysis Phase: The basic analysis phase consists of computing the reclaimable and attributed statistics for all volumes in a system. In addition, we implemented support for running these queries on any arbitrary group of volumes and the ability to query cross system migration costs for any group of volumes. We emphasize that we consider this phase as the most performance critical phase, since it should support interactive administrator queries that should be satisfied online to deliver a favorable user experience. In addition, the performance of grouping and merging is critical for the next level optimization algorithms as discussed in Section 3.4.

The basic functionality is straightforward and tightly follows the methods described in Section 3. For the group queries, however, an additional step is required to generate the sketch of a newly defined group. This process receives a list of volumes that form the group and merges them into a single sketch (reference counts and physical counts are summed in the merge, whereas compression ratio is averaged). To this end we implemented a classical heap-based k-way merge of sorted arrays (e.g., see [21]).[9]

[8] A different approach would be to run the analysis for many volumes/groups in parallel. However, the volumes can be very different in size which could create a strong imbalance between the processes and require more complex load sharing between the processes.

[9] Recall that at this point the volume sketches are held in sorted arrays.

Test Name	Data Written (TB)	Number of volumes	Ingest time (sec)	Analysis time (sec)
UBC-Dedup	63	768	22	0.21
Synthetic	1500	5	89	0.93
Customer System 1	980	3400	104	4.80
Customer System 2	505	540	65	2.70

Table 1: Performance of the ingest and analysis phase

6 Evaluation

6.1 Methods and Workloads

We used a number of methodologies to evaluate our sketches implementation, each with its own workloads. We describe these below:

1. **Synthetic data** - These are end-to-end tests performed in our lab in which various sets of synthetic data were written to the storage system, the sketches were extracted by the adjacent manager server and analyzed with the sketch analyzer. The tests evaluate both the performance of the mechanism as well as the accuracy of the results. The data was crafted in a way that allows us to predict the expected outcome and evaluate it. In addition we also ran tests that delete volumes from the system and compare the space that was released to the reclaimable that was predicted by the sketches.
   The data was generated using the following methodology: a number of equally sized data units were created, each with a different chosen compression ratio and with *no deduplication*. These were generated using the VDBench benchmarking suite [1]. We then wrote a number of data volumes to the storage, each consisting of a chosen set of units. Deduplication was created by reusing the same data units in different volumes, or repeating in inside the same volume. Our tests were of various sizes, ranging from small tests with data units of size 100GB each, to a large scale test in which 1.5 PB of data was written to the storage system, using data units of size 100TB each.
2. **UBC data traces** We leveraged the data trace called UBC-Dedup from the SNIA's IOTTA repository [6] (The Input/Output Traces, Tools, and Analysis repository). This are traces that were collected for the study of Meyer and Boloski [28], spanning 852 file systems from Microsoft in the form of hashes of the data and some related metadata (compression ratios are not available in this trace). After cleaning some small file systems, we ended up with hashes for 768 file systems representing 63 TBs of data.[10] The traces offers several chunking options and we used the fixed 8KB chunks. We use these file systems to simulate volumes in a storage system and evaluate our sketch analyzer both from a performance standpoint and from an accuracy standpoint. Note that these tests are not an end-to-end evaluation as real data was not involved. Rather, from the full hash traces we generated a much smaller sketch and ingested it into our sketch analyzer.
3. **Production data in the field:** The product implementation allows us to gain some insights via call-home data. While this data is very succinct and contains only general statistics, we learn from it about the performance of the sketch analyzer and can gather some insights about the data reduction properties of the written data.

[10] The traces also contain a number of versions of each file system, but we use only a single snapshot from each file system.

6.2 Performance Evaluation

Ingest & Analyze Performance: We first evaluate the performance of the ingestion and volume level statistics calculation. These provide an idea on the time it takes to acquire sketch statistics at a volume level on large production systems. The timing of the ingestion phase is less critical and it is also hampered by the fact that the sketch data is read from disk in the manager system and passed to the sketch analyzer via a REST API. The performance of the analysis phase is more crucial, as it gives us an indication on our ability to process real-time queries and answer mass queries for optimization purposes.

In Table 1 we present timing results for four large scale tests. The times are affected both by the actual sketch size as well as by the number of volumes. In addition the hardware available for the sketch analyzer also has an effect. We ran the local tests on a virtual machine running on an Intel® Xeon® Gold 6130 CPU @ 2.10GHz (cpu cores: 4) with 4×16 GB DDR3 RAM and do not know the configuration of the external runs. But in general, the take home is that we can easily perform a cycle of sketch statistics once in a couple of minutes even for very large systems (small systems can run in seconds). In our field deployment the cycle is longer since the sketch retrieval process is throttled to reduce network overheads.

Group Query Performance: We turn to evaluate our ability to support online queries on reclaimable capacities of arbitrary groups. The latency of answering such queries for large groups is dominated by the merge operation which creates the sketch of the queried group (described in Section 5.2). Figure 3 plots the performance of the merge operation and the entire query response time (with reclaimable computation) for

random groups from the UBC-Dedup workload, on various group sizes. The graph depicts the average of 50 runs, with the largest skew being under 10 ms. We are able to satisfy such queries in less than 0.2 seconds even for a very large group which contains half of the volumes in the system. For a small group, e.g. of 12 volumes, the average test managed to run in under 4 ms.

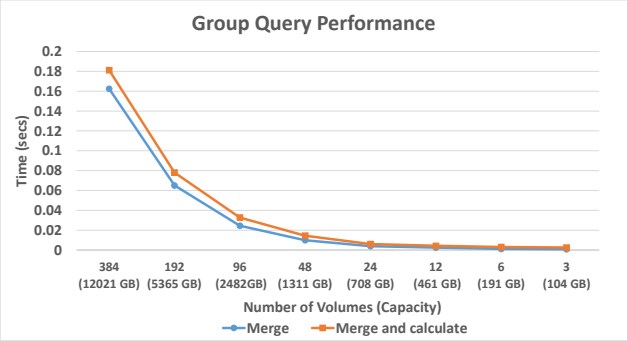

Figure 3: Performance of group query operations as a function of the group size ranging from a group of half the volumes in the system (368) through a small group of just 3 volumes.

Note that for further speedup we can run the merge operation using multiple threads, where each thread runs on separates partitions (according to hash values). For example, we tested a merge over all the 768 volumes on a virtual machine with four cores. The merge operation took 0.396 seconds using a single process, 0.186 seconds when using two processes and using 4 processes brought us down to 0.121 seconds.

6.3 Accuracy Evaluation

As mentioned in Section 6.1 we evaluated the accuracy of our work using two methods: using synthetic data and by studying the UBC-Dedup traces. The first is by writing synthetic data with expected behavior to the storage system and evaluating the expected reclaimable and attributed numbers. We complemented this by deleting volumes and measuring the amount of physical space reclaimed from the system. A crucial aspect of these tests was to evaluate the combination of compression and deduplication, since the UBC traces do not contain compression ratios. The synthetic tests therefore included writing data with a variety of deduplication and compression ratios. The skew observed in these test (not presented here) was always well within the accuracy guarantee.

The second method that we used to evaluate accuracy was using the UBC-Dedup traces. In order to evaluate the accuracy behavior of the sketch estimation method we first computed the exact physical capacities required to store each of the 768 volumes from the UBC-Dedup traces. This was done once by a lengthy offline process and recorded. We then evaluated the same physical capacities using our sketch mechanism with sketch factor 8192. In our evaluation we compare the observed error for each of the sketch estimations to the accuracy guarantee obtained in Theorem 1. For example, if the estimation is off by a factor of -3%, and the accuracy guarantee is $\varepsilon = 0.05$, the relative measured skew is $\frac{-0.03}{0.05} = -0.6$. In Figure 4 we show a histogram detailing the number of volumes in each range of relative measured skew of reclaimable estimations. The behavior comes out as a very nice bell curved distribution. Note that this behavior is not symmetric like a normal distribution, but rather is shifted to the negative skew, a behavior expected in a Binomial distribution $B(n,p)$ in which p is very small ($\frac{1}{8192}$ in our case).[11]

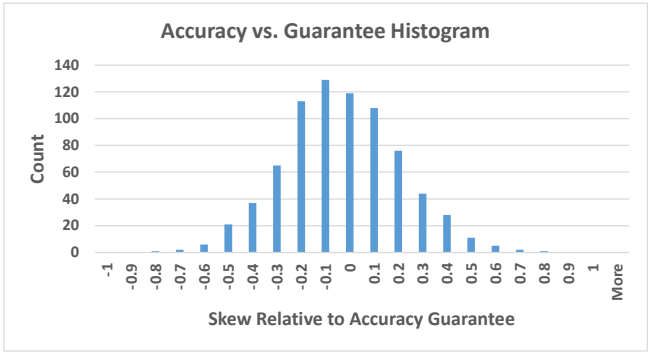

Figure 4: The errors observed over 768 volumes relative to the accuracy guarantee. The observed estimation skew was always smaller than the accuracy bound. In fact, in over 95% of the volumes the skew was less than half of the calculated accuracy bound.

To give a further indication on the behavior of sketch estimations, we picked six large volumes from the UBC-Dedup trace and examined the sketch estimation for with a growing sketch factor (starting with $2^{13} = 8192$ through 2^{17}). Figure 5 depicts the skew observed for each volume as the sketch factor grows. We observe that indeed the error tends to grows significantly as the sketch becomes sparser.

6.4 Data Center Level Optimizations

As an example of the potential of our methods for cross system optimizations, we implemented a greedy algorithm for space reclamation in a data center consisting of a number of deduplicated storage systems. The input to the algorithm is the name of the source system that is filling up and a minimal amount of space that needs to be reclaimed from it. The output is a migration plan of volumes from the source system to the other available system. The goal is to minimize the overall space usage of the entire data center by finding data similarities and exploiting them.

[11] This serves as justification for our choice not to use a binomial to normal estimation in the accuracy proof (as was done in [34]).

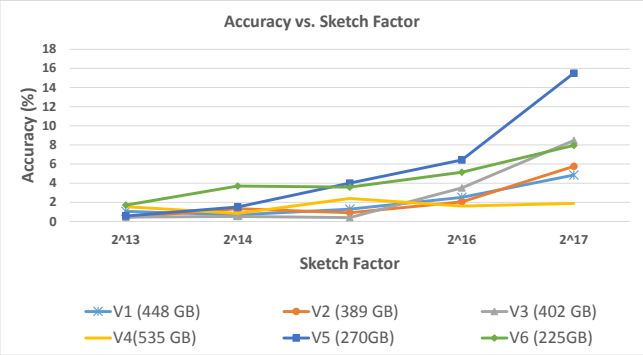

Figure 5: The errors observed for six volumes (and their physical sizes) as the sketch factor grows.

In a nutshell, at each round we enumerate over all of the volumes in the source system and evaluate what is their reclaimable space from the source system, and how much space they would take up in each of the other systems. At each round we pick the single volume for which the migration would yield the best space saving ratio[12] and update the sketches of the systems as though the migration has already happened. We then move to the next round in which the same process is repeated until the amount of reclaimed space from the source system is reached. Note that if data reduction mechanisms exist as part of the networking used for migrations (such as compression or WAN deduplication) then these consideration can easily be taken into account as part of the decisions in such a greedy algorithm.

The above algorithm does not attempt to find an overall optimum for such a process, and would generally not work well in situations in which the optimal solution involves moving several highly correlated volumes together. That being said, it exemplifies the insight and capabilities that the storage administrator has with clarity about expected volume capacities across multiple systems in the datacenter.

Evaluation: We evaluated the above algorithm using a simulated environment of four storage systems. The UBC-Dedup workload was partitioned among the four storage systems in a random fashion (each system received 192 volumes). On average, the physical space in each system amounted to 7TB.

We then ran the algorithm four times, each time with a different system serving as the source. In each test we asked to release at least 1TB of data from the source system. The tests ran between 30 to 55 seconds (depending on the system) and produced a migration plan that frees over 1TB of data from the source while taking up significantly less physical space in the other systems. The space savings achieved were between 257GB to 296GB, depending on the source system.

Figure 6 plots the progression of space reclamation and the capacity consumed at the targets for one of the experiments. We point out that as the rounds progress the ratio of space savings achieved by the migration process predictably declined. For example, at the beginning some volumes were found for which the space saving ratio from migration was as high as 10:1 and 3.7:1. As the algorithm progresses the extremely beneficial volumes have already been migrated and the saving ratio went down to around 1.15:1.

[12] We slightly penalize small volumes since we have a preference to migrate fewer volumes rather than many.

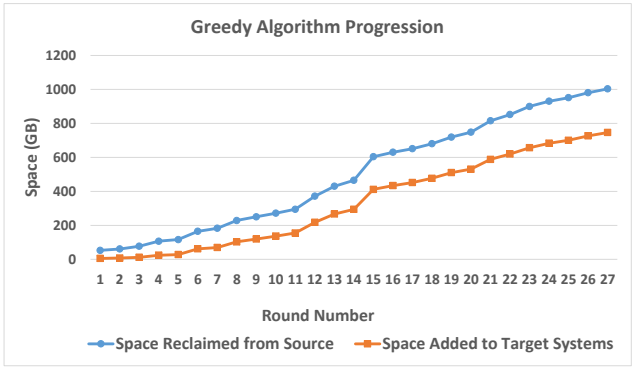

Figure 6: The progression of the reclaimed space from the source and the space the suggested volumes would take up in other systems.

6.5 Results from Early Adopters

As mentioned, our implementation is running as a beta offering for early adopters. This gives us initial statistics acquired in the field, on real customer data. We show here a glimpse of some insights that we have learnt (other than the performance numbers presented in Section 6.2): We evaluated how different the reclaimable numbers of a volume would be if they rely solely on unique data accounting rather than on sketches. The early numbers show that on average there is a 42% difference between the numbers and this can be attributed to the relative high internal volume deduplication encountered in the data sets that have been analyzed. This result strongly motivates the use of sketches for reclaimable capacity estimation.

Another example of insight that can be learnt is regarding the correlation between a data chunks' deduplicability and compression ratio. In the data sets that were scanned by the sketches mechanism we found no evidence of such correlation. Specifically, 99.9% of the data chunks had reference count between 1 and 4. For these four reference count values, we observed the exact same compression ratio of the data chunks.We are confident that as this feature is integrated and widely adopted we would gain some important insights on deduplication.

7 Conclusions and Discussion

We described a novel and efficient approach to analyzing volume capacities in storage systems with deduplication. Our mechanism provides accurate estimations for capacity measures that are not available in deduplicated storage systems to date. We have shown the accuracy of the capacity statistics computed from the sketch and demonstrated how it can be seamlessly collected from a system. From a performance standpoint our algorithms scale well and exhibit high performance even with high capacities. The small scale of the sketch and the ability to pull it out of the storage systems allows for further analytics and automation. To date, placement decision algorithm were mostly focused only on performance optimization and just making sure we don't overrun the system overall capacity. The sketch mechanism enables a new dimension of data center capacity optimization. This opens the door for performing insight analytics on storage capacities and making placement decisions at the pool or system level as well as across multiple deduplication domains and systems. Among the potential uses of our technology is the ability to reduce the overall space usage in a number of circumstances: Upon space reclamation when a system fills up (as described in Section 6.4); as part of data rebalancing between systems upon introducing of new systems; or by actively relocating volumes to reside in the same deduplication domain together with their optimal cluster of related volumes.

Acknowledgements: We are grateful to our many colleagues at the IBM Systems Israel Development Center who contributed to our efforts to bring this technology to the field.

References

[1] VDBench users guide. https://www.oracle.com/technetwork/server-storage/vdbench-1901683.pdf, 2012.

[2] HPE storeonce data protection backup appliances. https://www.hpe.com/us/en/storage/storeonce.html, 2018.

[3] IBM FlashSystem 9100. https://www.ibm.com/us-en/marketplace/flashsystem-9100, 2018.

[4] IBM FlashSystem A9000. https://www.ibm.com/il-en/marketplace/small-cloud-storage/specifications, 2018.

[5] Pure storage: purity-reduce. https://www.purestorage.com/products/purity/purity-reduce.html, 2018. (Retrieved Sept. 2018).

[6] SNIA: Iotta repository home. http://iotta.snia.org/, 2018.

[7] VMware vsan: Using deduplication and compression. https://docs.vmware.com/en/VMware-vSphere/, 2018.

[8] XIOS 6.1 data reduction (drr) reporting per a volume. https://xtremio.me/, 2018.

[9] XtremIO integrated data reduction. https://www.emc.com/collateral/solution-overview/h12453-xtremio-integrated-data-reduction-so.pdf, 2018. (Retrieved Sept. 2018).

[10] ARONOVICH, L., ASHER, R., BACHMAT, E., BITNER, H., HIRSCH, M., AND KLEIN, S. T. The design of a similarity based deduplication system. In *SYSTOR* (2009), ACM.

[11] BAR-YOSSEF, Z., JAYRAM, T. S., KUMAR, R., SIVAKUMAR, D., AND TREVISAN, L. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002* (2002), pp. 1–10.

[12] BHAGWAT, D., ESHGHI, K., LONG, D. D. E., AND LILLIBRIDGE, M. Extreme binning: Scalable, parallel deduplication for chunk-based file backup. In *MASCOTS* (2009), pp. 1–9.

[13] DEUTSCH, P., AND GAILLY, J. L. Zlib compressed data format specification version 3.3. Tech. Rep. RFC 1950, Network Working Group, May 1996.

[14] DONG, W., DOUGLIS, F., LI, K., PATTERSON, R. H., REDDY, S., AND SHILANE, P. Tradeoffs in scalable data routing for deduplication clusters. In *FAST* (2011), pp. 15–29.

[15] DOUGLIS, F., BHARDWAJ, D., QIAN, H., AND SHILANE, P. Content-aware load balancing for distributed backup. In *Proceedings of the 25th Large Installation System Administration Conference, LISA* (2011).

[16] FLAJOLET, P., AND MARTIN, G. N. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci. 31*, 2 (1985), 182–209.

[17] FORMAN, G., ESHGHI, K., AND SUERMONDT, J. Efficient detection of large-scale redundancy in enterprise file systems. *Operating Systems Review 43*, 1 (2009), 84–91.

[18] FREY, D., KERMARREC, A., AND KLOUDAS, K. Probabilistic deduplication for cluster-based storage systems. In *ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012* (2012), p. 17.

[19] FU, Y., JIANG, H., AND XIAO, N. A scalable inline cluster deduplication framework for big data protection.

In *Middleware 2012 - ACM/IFIP/USENIX 13th International Middleware Conference, Montreal, QC, Canada, December 3-7, 2012. Proceedings* (2012), pp. 354–373.

[20] GIBBONS, P. B., AND TIRTHAPURA, S. Estimating simple functions on the union of data streams. In *SPAA* (2001), pp. 281–291.

[21] GREENE, W. k-way merging and k-ary sorts. In *Proceedings of the 31-st Annual ACM Southeast Conference* (1993), pp. 127–135.

[22] HARNIK, D., KAT, R., SOTNIKOV, D., TRAEGER, A., AND MARGALIT, O. To zip or not to zip: Effective resource usage for real-time compression. In *USENIX FAST'13* (2013).

[23] HARNIK, D., KHAITZIN, E., AND SOTNIKOV, D. Estimating Unseen Deduplication—from Theory to Practice. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (2016), pp. 277–290.

[24] HARNIK, D., MARGALIT, O., NAOR, D., SOTNIKOV, D., AND VERNIK, G. Estimation of deduplication ratios in large data sets. In *IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST 2012* (2012), pp. 1–11.

[25] HUFFMAN, D. A. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers 40*, 9 (September 1952), 1098–1101.

[26] LILLIBRIDGE, M., ESHGHI, K., BHAGWAT, D., DEOLALIKAR, V., TREZISE, G., AND CAMBLE, P. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST'09)* (2009).

[27] LU, M., CONSTANTINESCU, C., AND SARKAR, P. Content sharing graphs for deduplication-enabled storage systems. *Algorithms 5*, 2 (2012).

[28] MEYER, D. T., AND BOLOSKY, W. J. A study of practical deduplication. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '11)* (2011), pp. 1–13.

[29] MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[30] NAGESH, P. C., AND KATHPAL, A. Rangoli: Space management in deduplication environments. In *Proceedings of the 6th International Systems and Storage Conference* (2013), SYSTOR '13, pp. 14:1–14:6.

[31] SHILANE, P., CHITLOOR, R., AND JONNALA, U. K. 99 deduplication problems. In *8th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2016, Denver, CO, USA, June 20-21, 2016.* (2016).

[32] WALDSPURGER, C. A., PARK, N., GARTHWAITE, A., AND AHMAD, I. Efficient MRC construction with SHARDS. In *13th USENIX Conference on File and Storage Technologies (FAST 15)* (Santa Clara, CA, 2015), USENIX Association, pp. 95–110.

[33] WIRES, J., GANESAN, P., AND WARFIELD, A. Sketches of space: ownership accounting for shared storage. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24 - 27, 2017* (2017), pp. 535–547.

[34] XIE, F., CONDICT, M., AND SHETE, S. Estimating duplication by content-based sampling. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (2013), USENIX ATC'13, pp. 181–186.

[35] ZIV, J., AND LEMPEL, A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory 23*, 3 (1977), 337–343.