Using Elasticity to Improve Inline Data Deduplication Storage Systems

Yufeng Wang
Temple University
Philadelphia, PA, USA
Y.F.Wang@temple.edu

Chiu C Tan
Temple University
Philadelphia, PA, USA
cctan@temple.edu

Ningfang Mi
Northeastern University
Boston, Massachusetts, USA
ningfang@ece.neu.edu

***Abstract*—Elasticity is the ability to scale computing resources such as memory on-demand, and is one of the main advantages of utilizing cloud computing services. With the increasing popularity of cloud based storage, it is natural that more deduplication based storage systems will be migrated to the cloud. Existing deduplication systems however, do not adequately take advantage of elasticity. In this paper, we illustrate how to use elasticity to improve deduplication based systems, and propose EAD (elasticity aware deduplication), an indexing algorithm that uses the ability to dynamically increase memory resources to improve overall deduplication performance. Our experimental results indicate that EAD is able to detect more than 98% of all duplicate data, however only consumes less than 5% of expected memory space. Meanwhile, it claims four times of deduplication efficiency than the state-of-art sampling technique while costs less than half of the amount of memory.**

I. Introduction

Data deduplication is a technique used to reduce storage and transmission overhead by identifying and eliminating redundant data segments. Data deduplication plays an important role in existing storage systems [1], and its importance will continue to grow as the amount of data increases (the growth of data is estimated to reach 35 zettabytes in the year 2020).

The flexibility and cost advantages of cloud computing providers such as Azure [2], Amazon [3], etc. make it deploying storage services in the cloud as an attractive option. A key property of cloud computing is *elasticity*, the ability to dynamically adjust the amount of computing resources quickly. Elasticity can improve deduplication systems by allowing deduplication storage systems to dynamically adjust the amount of memory resources as needed to detect sufficient amount of duplicate data. This is especially useful for *inline* deduplication systems [4], [5] where the index used for deduplication is often kept within the memory to avoid the performance bottleneck from disk I/O operations.

Intuitively, there is a basic tradeoff between amount of duplicate data been detected and the amount of memory space required. Smaller memory resources lead to small indexes, which in turn leads to worse deduplication performance due to missed deduplication opportunities. Selecting too much memory, on other hand, leads to wasted memory, since RAM allocated to the index cannot be used for other purposes. Elasticity provides the ability to scale memory resources as needed to improve deduplication performance without incurring wasted resources.

In this paper, we proposed an elasticity-aware deduplication (EAD) algorithm that takes advantage of the elasticity of cloud computing. The key features of our solution is that our deduplication algorithm is compatible with current deduplication techniques such as sampling to take advantage of locality [6], [7], and content-based chunking [8]–[10]. This means our solution can take advantage of state-of-the-art algorithms to improve performance. Furthermore, we also present a detailed analysis of our algorithm, as well evaluation using extensive experiments on real world dataset.

The rest of the paper is organized as follows: Section 2 contains the related work. Section 3 explores limitations of existing approaches. The EAD is presented in Section 4. Section 5 evaluates our solution, and Section 6 concludes.

II. Related Work

Cloud based backup systems typically use inline data deduplication, where redundant data chunks are identified at run time to avoid transmitting the redundant data from the source to the cloud. This is opposed to offline deduplication where the source transmits *all* the data to the cloud, which then runs deduplication process to conserve storage space. For the remaining paper, deduplication will refer to inline deduplication.

Numerous research has been done to improve the performance of finding duplicate data. Work by [11] focused techniques to speed up the deduplication process. Researchers have also proposed different chunking algorithms to improve the accuracy of detecting duplicates [12]–[16]. Other research considers the problem of deduplication of multiple datatypes [17], [18]. This line of research is complimentary to our work, can be easily incoporated into our solution.

The ever increasing amounts of data coupled with the performance gap between in-memory searching and disk look ups, mean that increasingly, disk I/O has become the performance bottleneck. Recent deduplication research have focused on addressing the problem of limited memory. Work by [19] proposed integrated solutions which can avoid disk I/Os on close to 99% of the index lookups. However, [19] still puts index data on the disk, instead of memory. Estimation algorithms

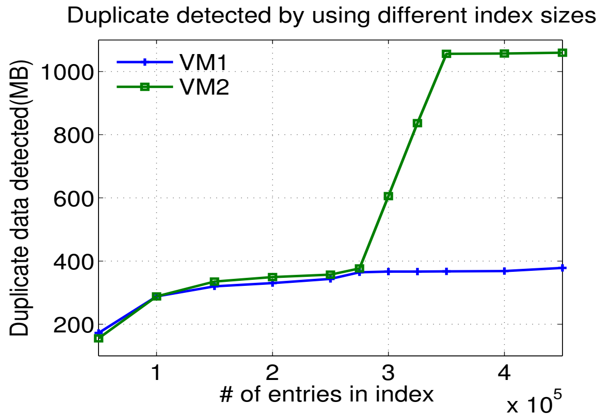


Fig. 1: Intuitive test on amount of duplicate detected on two equal-sized(4.7GB) VMs by using equal-size indexes

like [20] can be used to improve the performance by reducing total number of chunks, but the fundamental problem remains as the amount of data increases. Other existing research in this area have proposed different sampling algorithms to index more data using less memory: [6] introduces a solution by only keeping part of chunks' information in the index; [7] proposes a more advanced method based on the work in [6] by deleting chunks' fingerprints (FPs) from the index when it's approaching fullness.

The fundamental limitation with sampling based approach is that it is impossible to maintain deduplication performance by reducing the sampling rate while the input data size increases but RAM resources remains fixed. Our solution builds upon earlier work on sampling by taking advantage of the elasticity property of cloud computing to carefully combine sampling with increasing memory resources.

III. The Case for Using Elasticity

This section will explore some alternatives to improve deduplication performance, and their limitations.

A. Why not pick the best memory size?

One alternative is to try and estimate the appropriate amount of memory that is needed prior to deploying the deduplication system. A straightforward approach is to perform simple profiling on a sample of data and compute the expected memory requirements based on the results.

To illustrate why this is difficult to choose the right amount of RAM in practice, we conducted a simple experiment that represents a storage system used to archive virtual machine (VM) images (this is a common workload used in deduplication evaluations [7], [21]). We want to maximize deduplication ratio to conserve bandwidth and storage costs. For simplicity, we assume that all VMs are running the same OS, and are the same size. A simple way to estimate memory requirements is to first estimating the index size for a single VM, and then use that to estimate the total RAM necessary for all users. Thus, given n users, and since each user stores the same size VM, we estimate m amounts of RAM to index one user, our backup system will need $n \cdot m$ amounts of RAM. We can derive m via experiments.

Fig. 1 shows the results for two VMs. As far as we know, VM2 contains more text files while VM1 has more video files. Number of index entry slots indicates how much information of already stored data the system can provide for duplicate detection. We set fixed number of index entries for duplicate detection and gradually increase it. We see that when index entry slots number increases to 270 thousand, both VMs exhibit the same amount of duplicate data. As we increase the index size, VM1 shows limited improvement, while VM2 shows much better performance. If we had used VM1 to estimate m would have led to much less bandwidth savings, especially if a significant number of VMs resemble VM2. Buying too much memory is wasteful if most of the data resemble VM1.

B. Using Locality and Downsampling

Storage systems that make use of data deduplication generally operate on chunk-level, and in order to quickly determine potential duplicate chunks, an index for existing chunks needs to be maintained in memory. For example, a 100TB data will need about 800GB amounts of RAM for the index under standard deduplication parameters [22]. This makes keeping the entire index in memory challenging.

The principle of locality is used to design sampling algorithms that utilize smaller index size while providing good performance [6]. The locality principle suggests that if chunk X is observed to be surrounded by chunks Y, Z, W in the past; the next time chunk X appears, there's a high probability that chunks Y, Z, W will also appear. In sampling-based deduplication, the data will be first divided into larger segments, each of which contains thousands of chunks. Deduplication is executed based on these segments by identifying existence of their sampled chunks' fingerprints in the index. If a chunk's fingerprint is found in the index, the correspondent segment which contains that chunk will be located and fingerprints information of all the other chunks in this segment will be pre-fetched from disk to the chunk cache in memory.

Downsampling algorithm [7] works as an optimized sampling approach, by taking advantage of the locality principle. The difference is that the sampling rate is initialized as 1, which indicates it picks all the chunks in a segment as its sampled chunks. As the amount of incoming data increases, this value gradually decreases by dropping half of index entries. Thus the indexing capacity doubles by only accepting a part of chunks' fingerprints as samples to represent each segment. In other words, instead of indexing chunks X, Y, Z, and W in RAM, the downsampling algorithm will only index chunk X (or another one among four of them) in RAM after two times of adjustments, and the rest on disk.

The above sampling-based approaches have two main drawbacks. The first (obvious) drawback is that not all data will exhibit locality [17], and thus sampling algorithms do not work well with these datasets. The second drawback is that even for data that exhibits locality, it is difficult to select the correct

sampling rate or how to adjust it, due to the large variance in possible deduplication ratio [23] [24].

IV. EAD: Elasticity-Aware Deduplication

Storage deduplication services in the cloud often run in virtual machines (VM). Unlike a conventional OS which runs directly on physical hardware, the OS in a VM is running on top of a *hypervisor* or *virtual machine monitor*, which in turn, communicates with the underlying physical hardware. The hypervisor is responsible for increasing RAM resources to the virtual machine (VM) dynamically. This can be done in two generic ways.

The first is to use a ballooning algorithm to reclaim memory from other VMs running on the same physical machine (PM) [25]. This is a relatively lightweight process that relies on the OS's memory management algorithm, but can only increase relatively small amounts of memory. Deduplication systems that require increasingly larger amounts of memory need to run a VM migration algorithm [26], [27]. In VM migration, the hypervisor migrates the RAM contents from one PM to another with sufficient memory resources [26]. Regardless of the migration algorithm used, some downtime can inevitably occur when switching over to a new VM [27].

A naive approach towards incorporating elasticity is to increase the memory size once the index is close to being full. This naive approach does not perform well since frequent migrations induce a high overhead. Furthermore, the naive approach always retains the entire old index during each migration, even those index entries do not finger print many chunks. Such poor performing index entries take up valuable index space without providing much benefits.

Our approach combines the benefits of downsampling [7] and VM migration to allow users to maintain a satisfactory level of performance by adjusting sampling rate and memory size accordingly. Our system design consists of two components, an *EAD client* that is responsible for file chunking, fingerprint computation and sampling, and an *EAD server* which controls the index management and other memory management operations. The *EAD client* is run on the client side, for instance, at the gateway server for a large company. The *EAD server* can be executed by the cloud provider. The entire system design is shown in Fig. 2. Only unique data is supposed to be store in Physical Storage. The File Manager is responsible for data retrieval and maintenance, how it works is out of this paper's scope.

A. EAD Algorithm

Different types of users have different deduplication requirements. Some users will be willing to tolerate worse deduplication performance in exchange for lower costs, while others are not. To accomodate different requirements, EAD is designed to allow a user to specify a migration trigger, Γ ($\in (0,1)$), which specifies the level of deduplication performance the user is willing to accept.

Deduplication performance is usually measured by *reduction ratio* [18], [28], which is the size of the original dataset

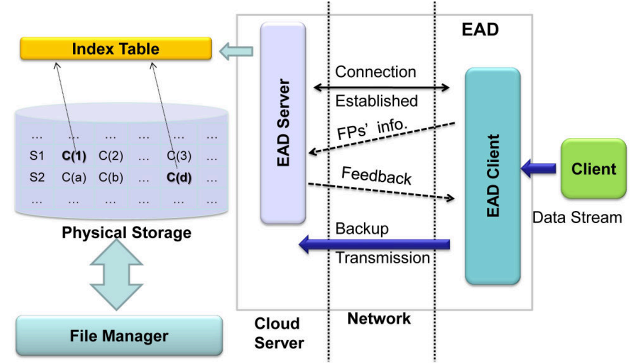


Fig. 2: EAD infrastructure.

divided by the size of the dataset after deduplication. To help the user select the migration trigger, we define Deduplication Ratio (DR),

$$\text{Deduplication Ratio} = 1 - \frac{\text{Size after deduplication}}{\text{Size of original data}}.$$

Intuitively, we would like to first apply downsampling algorithms until the deduplication performance becomes unsatisfactory, and then migrate the index to larger memory in order to obtain better performance. EAD will migrate to larger RAM **only** when migration will result in deduplication performance better than Γ. This has an important but subtle implication. **EAD will *not* always migrate when deduplication performance falls under Γ, but only when migration will improve performance.** This is important because given a dataset that inherently exhibits poor deduplication characteristics [29], adding more RAM will incur the migration overehad without improving deduplication performance.

This means that EAD cannot simply compare the measured DR against Γ because the measured DR may not necessarily reflect the amount of duplication that exists. To illustrate, let us assume that the deduplication system measures its DR and it is less than Γ. There are two possibilities. The first is that the system has performed overly aggressive downsampling, and can benefit from increasing RAM. The second possibility is that the dataset itself has poor deduplication performance, e.g. data in multimedia or encrypted files. In this case, increasing RAM does not result in better performance.

How our EAD algorithm determines when to migrate to more RAM resources can be found in Alg. 1. It executes in two phases as generic in-line deduplication systems do. We use S_{in} and x to denote the incoming segment and chunks inside it. FP_x represents the fingerprint of chunk x. In **Phase I** the *EAD Client* sends all chunks' fingerprints information (FP_{x_i}) ($\forall x_i \in S_{in}$) , including labeling $FP_{x_i}^{est}$ and $FP_{x_i}^{dedup}$ of sampled chunks used for estimation and duplication detection, in each segment S_{in} to *EAD Server*. The latter will search index table $\mathbb{T}$ and $chunk\ cache$ for duplication identification, as well as updating *estimation base* $\mathbb{B}$. Based on results generated, in which each chunk x_i is marked as dup or $uniq$, indicating it is a duplicate or unique chunk. *EAD Client* only transmits unique data chunks along with metadata of duplicate ones to *EAD Server* in **Phase II**, saving bandwidth and storage

space. At the meantime, current sampling rate R_0 is subject to change to R based on deduplication performance. Details on features of the EAD algorithm will be presented next.

B. Estimating Possible Deduplication Performance

One of the key features of EAD is that the algorithm is able to determine whether migration will be beneficial. In order to distinguish whether poor deduplication performance is due to overly aggressive downsampling or inherent within the dataset, we first need to be able to estimate the potential DR of the dataset. Obtaining the actual DR is impractical since it requires performing the entire deduplication process.

Prior work from [30] provided an estimation algorithm to estimate the deduplication performance for static, fixed-size data sets. Their algorithm requires the actual data to be available in order to perform random sampling and comparisons. However, in our problem, the dataset can be viewed as a stream of data. There is no prior knowledge of the size or characteristics of the data to be stored in advance. We also cannot perform back and forth scanning of the complete dataset for estimation.

In our EAD algorithm, we let the *EAD Server* maintain an *estimation base* $\mathbb{B}$. The *EAD Client* randomly selects κ fingerprints from each segment and sends them to *EAD Server* to be stored in $\mathbb{B}$. Suppose there are n_s segments come in, there will be $\kappa \cdot n_s$ samples, which will increase along with the increasing amount of incoming data. Each entry slot in $\mathbb{B}$ includes a fingerprint as well as two counters, x_{c1} and x_{c2}, where counter x_{c1} records the number of occurrences of fingerprint FP_x appears in the $\mathbb{B}$, and x_{c2} records the number of occurrences of fingerprint FP_x appears among that of all the chunks uploaded.

We integrate our estimation process into the regular deduplication operations so as to avoid the separate sampling and scanning phases by [30]. While the client sends the samples for duplication searching to the storage server, these samples for estimation are transmitted at the same time for updating $\mathbb{B}$. During the fingerprint comparison of incoming chunks against that in chunk cache, we update $\mathbb{B}$ again, incrementing the counter x_{c2} by one every time its correspondent fingerprint appears. Thus, there is no extra overhead for our estimation purpose.

Using $\mathbb{B}$, we can compute the estimated deduplication ratio, EDR, as

$$EDR = 1 - \frac{1}{\kappa \cdot n_s} \sum_{x \in \mathbb{B}} \frac{x_{c1}}{x_{c2}}.$$

The computation of EDR happens while the index size is approaching the memory limit. Only in the case that DR is smaller than $\Gamma \cdot EDR$, there will be a potential performance improvement by migration, and EAD will migrate the index to larger RAM. Otherwise, EAD will apply downsampling on the index as the exchange for larger indexing capacity.

C. EAD Refinements

The performance of the EAD algorithm can be further improved by observing additional information obtained during the run time and then adjusting the parameters of the algorithm.

Algorithm 1 Elastic deduplication strategy

1: The incoming segment S_{in}:
Deduplication Phase I: Identify duplicate chunks
2: $\forall x_i \in S_{in}$: EAD Client sends FP_{x_i} to EAD Server
3: **for all** $FP_{x_i}^{dedup}$ **do**
4:   **if** $FP_{x_i}^{dedup} \in \mathbb{T}$ **then**
5:     Locate its correspondent segments S_{dup}
    $\forall x_j \in S_{dup}$: Fetch information of x_j (FP_{x_j})
    Set $x_j \in chunk\ cache$
6:   **else**
7:     Add $FP_{x_i}^{dedup}$ to $\mathbb{T}$
8: **for all** $FP_{x_i}^{est}$ **do**
9:   **if** $FP_{x_i}^{est} \in \mathbb{B}$ **then**
10:     $x_{i_{c1}} = x_{i_{c1}} + 1$
11:   **else**
12:     Add $FP_{x_i}^{est}$ to $\mathbb{B}$
    Set $x_{i_{c1}} = x_{i_{c2}} = 0$
13: **for all** $x_i \in S_{in}$ **do**
14:   $\forall x_k \in chunk\ cache$: Compare FP_{x_i} with FP_{x_k}
15:   **if** $FP_{x_i} = FP_{x_k}$ **then**
16:     Set $x_i \in dup$ (x_i^{dup})
17:   **else**
18:     Set $x_i \in uniq$ (x_i^{uniq})
19:   $\forall x_l \in \mathbb{B}$: Compare FP_{x_i} with FP_{x_l}
20:   **if** $FP_{x_i} = FP_{x_l}$ **then**
21:     $x_{i_{c2}} = x_{i_{c2}} + 1$
Deduplication Phase II: Data transimission
22: **for all** $x_i \in S_{in}$ **do**
23:   Transmits x_i^{uniq} along with only metadata of x_i^{dup}
24: EAD finishes processing S_{in}
25: **if** Index is approaching the RAM limit **then**
26:   **if** DR $< \Gamma \cdot EDR$ **then**
27:     **if** $R_0 = 1$ **then**
28:       EAD sets $\Gamma = \frac{DR}{EDR}$
29:     **else**
30:       EAD triggers migration, setting rate $R = \Delta \cdot R_0$
31:   **else**
32:     EAD sets $R = \frac{R_0}{\Delta}$

Adjusting Γ**.** The parameter Γ is specified by the user, and indicates the user's desired level of deduplication performance. However, the user may sometimes be unaware of the underlying potential deduplication performance of the data, and set an excessively high Γ value, resulting in unnecessary migration over time. We adjust the user's Γ value to $\frac{DR}{EDR}$ after each migration, and also in the case that DR has not reached accepted performance even the sampling rate is one. So that it represents the current system's maximum deduplication ability. In this way, EAD is able to elastically adapt variations on incoming data.

Amount of RAM and Sampling Rate post migration. A simple way to compute the amount of RAM is allocating after

migration by using a fixed sized Δ, e.g. doubling the RAM each time ($\Delta = 2$). We then reset the sampling rate back to 1, and start all over again.

We can improve over this process by observing the next to last sampling rate used prior to migration. This rate is the last known sampling rate that produced acceptable deduplication performance. This is valid because if it did *not* produce an acceptable performance, EAD would have already triggered migration.

Once we have this new sampling rate, we can compute the amount of RAM by introducing a new counter d (initialized as zero) to record occurrences of downsampling. We can then compute the new RAM, RAM_{new} as

$$RAM_{new} = \begin{cases} \Delta \cdot RAM_{org} & d = 1 \\ \Delta \cdot [1 - \sum_{i=1}^{d-1} \frac{1}{\Delta^{i-1}}] \cdot RAM_{org} & d \geq 2 \end{cases}$$

As the times of downsample operation increase, EAD requires less amount of RAM for index table after migration. Compared with always requiring Δ times of original RAM, such optimized approach is able to claim higher memory utilization efficiency.

Managing Size of $\mathbb{B}$. One concern with our estimation scheme is that the size of $\mathbb{B}$ may become too large. If we need a large amount of RAM to store $\mathbb{B}$, we will be wasting RAM resources that could be used in the index. In practice, the size of $\mathbb{B}$ is relatively modest. Each entry in $\mathbb{B}$ consists of a fingerprint and two counters. Using SHA-1 to compute the fingerprint results in a 20 byte fingerprint. An additional four bytes are used for each counter. Thus, each $\mathbb{B}$ entry is 28 bytes, indicating that the total size of $\mathbb{B}$ would be at most approximately 33.38 MB to support 1 TB of data. In our experiment, it only requires 4.32 MB for estimating 163.2 GB dataset.

V. IMPLEMENTATION

A. Experimental setup

For our experiments, we collected a dataset consisting of VMs that all run the Ubuntu OS, but each VM has different types of software and utilities installed and contains different types of application data, which majority comes from Wikimedia Archives [31] and OpenfMRI [32]. The total size of our dataset is approximately 163.2 GB. While the dataset size is relatively modest compared to some prior work [5], [10], [18], we believe that it still adequately reflects real-world usages of backup systems, such as backing up employee laptops. To ensure a fair comparison, we have scaled down our index size to correspond to our dataset size, in order to better represent a large scale environment.

We have implemented our EAD algorithm in Java. For all experiments, we use variable block deduplication parameters respectively with a minimum and maximum size of 4 KB and 16 KB, and a corresponding average chunk size is 8 KB. We set the segment size to be 16 MB. These are common parameters used in previous research [33] [19]. The experiments are carried out on a 4-core Intel i3-2120T at 2.60GHz with total 8GB RAM, running on linux.

Strategy	# of index entries	size of index(MB)
Full index	10×10^6	640
With down-sample	5×10^5	32
EAD	1×10^5	6.4

TABLE I: RAM deployment for index under different deduplication strategies. We set the down-sampling trigger as 0.85, which means while the storage is approaching 85% of its current limit, the index will be down-sampled(Half of its entries will be removed. e.g. delete index FPs with $FP \ mod \ 2 = 0$).

We evaluate our solution, denoted as `Elastic` in the figures, against two alternatives approaches. The first alternative, denoted as `FullIndex`, represents an ideal situation where there is unlimited RAM available. This will serve as an upper bound on the total amount of space savings. The other alternative is denoted as `DownSample`, which is based on [7], a recent approach that dynamically adjusts the sampling rate to deal with insufficient RAM.

B. Deduplication ratio

We hereby compare our algorithm with a generic deduplication mechanism without sampling and a state-of-art high performance deduplication strategy with down-sampling mechanism [7]. Before deploying the deduplication process, we allocate a specific amount of RAM for index in different strategies. Table I shows the amount of RAM allocated for different deduplication strategies.

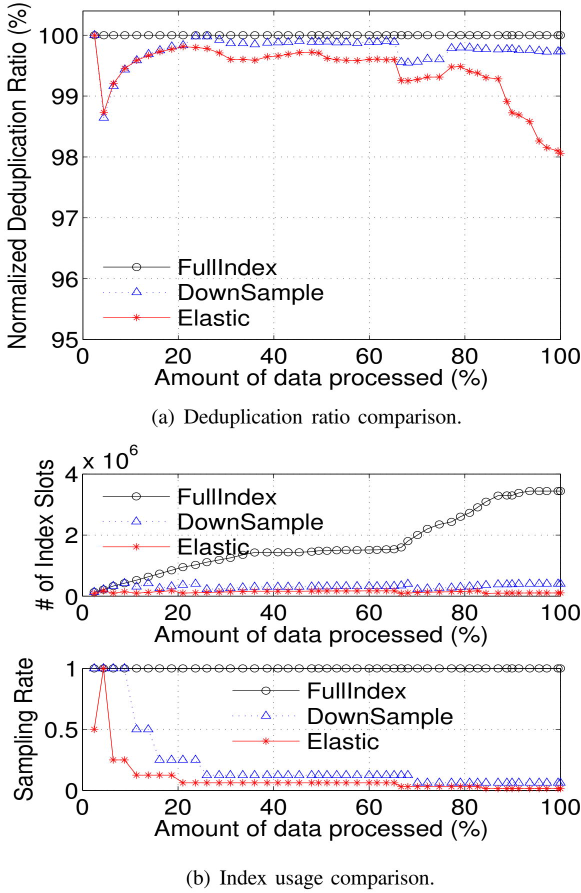
We set the size of each entry slot in the index as 64 bytes, which consists of three parts: FP, chunk metadata (storage address, chunk length,etc) and counter, which is 20 bytes (SHA-1 hash signature [34]), 40 bytes and 4 bytes, respectively. These sizes may vary under different hash functions or addressing policies, however it will not differ too much.

We assume that the capacity is 75 GB. Nearly 10 million index entries are needed to index all the unique data if we do not use any sampling strategies. While under the down-sample strategy with the minimum sampling rate of 0.05, we need 500K index entries for 75 GB of unique data. EAD always picks a much more conservative size of index, specifically only 100K entry slots in this case.

We here use *Normalized Deduplication Ratio* as the metric for deduplication ratio comparison. It is defined as the ratio of measured Deduplication Ratio to Deduplication Ratio of `FullIndex` deduplication. Note that `FullIndex` detects all the duplicate data chunks and can claim highest deduplication ratio. Thus, such a metric is meaningful because it indicates how close the measured deduplication ratio is to the ideal deduplication ratio achievable in the system.

Fig. 3(a) shows the *Normalized Deduplication Ratio* of the above deduplication strategies. Downsampling and computation of EDR happen when the usage of index approaches 85% of its capacity. For the down-sample strategy, it has the ratio higher than 99.5 %, showing the benefits of taking advantage of locality. The EAD does not claim equally high ratio, however the gap is less than 2 %. Also consider that the performance requirement for EAD is defined by Γ, which is 0.95 in this case, the performance of `Elastic` is always higher than 98%, performing better than what is required.

However, purely comparing the deduplication ratio is not fair for evaluating their performance. Since that these three strategies spend different amount of RAM for index from the start. Fig. 3(b) shows how sampling rate and number of index slots used vary above cases. Obviously, it brings too much memory cost without sampling. We notice that both `DownSample` and `Elastic` have comparatively very low memory cost (small number of index entry slots). Also we can observe that when about 5% of data has been processed, the sampling rate in `Elastic` increases, reflecting its feature of elasticity. The above results show that EAD is able to use less RAM space to achieve a satisfying deduplication ratio, which is only slightly lower than the other two. Next we derive a more meaningful metric *Deduplication Efficiency*, as a single utility measure that encompasses both deduplication ratio and RAM cost, to make a more fair comparison among these three strategies.



(a) Deduplication ratio comparison.

(b) Index usage comparison.

Fig. 3: The sampling rate is 1 for all of them at the start of back up. The index migration in EAD is triggered when the normalized deduplication ratio drops below 95 % (Γ=0.95), after that, sampling rate doubles(Δ=2).

C. Deduplication efficiency

As discussed in Section V-B, neither deduplication ratio nor memory cost alone can fully represent the system performance. Therefore we define:

$$Deduplication\ Efficiency = \frac{Duplicate\ Data\ Detected}{Index\ Entry\ Slots}$$

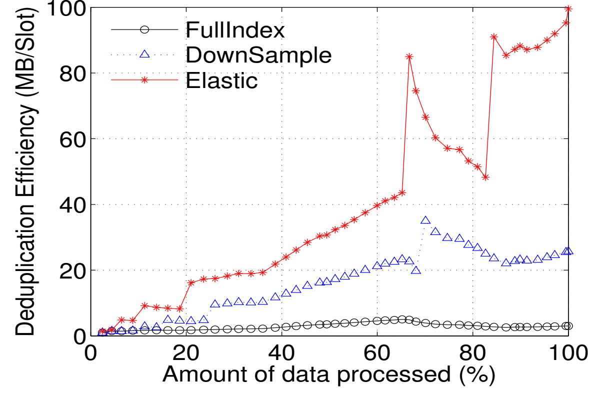


Fig. 4: Deduplication efficiency performance.

as a more advanced performance evaluation criteria. By using this criteria, we make more fairly comparisons among EAD and the other two solutions, as shown in Fig. 4.

It shows that `Elastic` outperforms both `Downsample` and `FullIndex` on efficiency. Notice that `Elastic` always yields a higher efficiency, almost 4 times of that from `Downsample` and 30 times of that from `FullIndex`. This is because that its elastic feature enable it utilize as little memory space as possible to detect enough duplicate data as required, avoiding memory waste as the other two do.

D. Elasticity Optimization

In this section, we explore the variations of EAD from different aspects.

Monitoring accuracy. EAD can work properly only when it is able to accurately monitor the real time deduplication efficiency. As the criteria of judging deduplication performance, estimated duplication rate is supposed to be as accurate as possible. Otherwise, elasticity might bring unexpected effect on the performance if it makes an inappropriate decision for index migration.

Fig. 5 shows the accuracy of monitored deduplication ratios during the backup process. 500 times of independent test were conducted on the dataset. We here consider the ratio of estimated deduplication ratio in EAD to that in `FullIndex` as error deviation, which indicates the real time accuracy of monitoring. From the figure we can see that initially the error deviation is at most 10% , but as more data comes in, the deviation reduces to 2%, which offers a reliable criteria for evaluation on system performance.

According to [30], the reduction ratio of a dataset of up to 7 TB can be estimated with accuracy less than 1%. Notice that we here dynamically estimate the ratio which represents a very different situation (as elaborated in Section IV-A). It can been seen from Fig. 5 that the deviation is higher than expected when only part of the dataset has been estimated. So that we reserve more RAM for estimation, even though it only costs approximately 4.32 MB of RAM for 162078 samples.

Impact on initial index size. There is no standardized criteria of selection on amount of memory size for index table in deduplication systems. We only give examples of memory

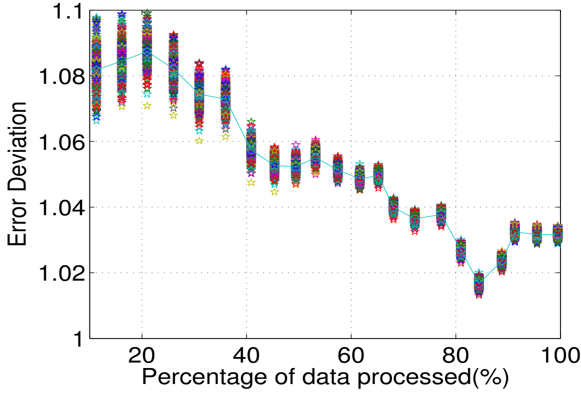


Fig. 5: Estimation Accuracy verification. 20 samples per segment are randomly picked out from incoming data.

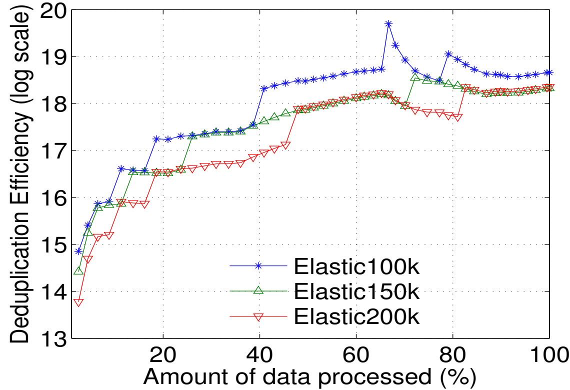
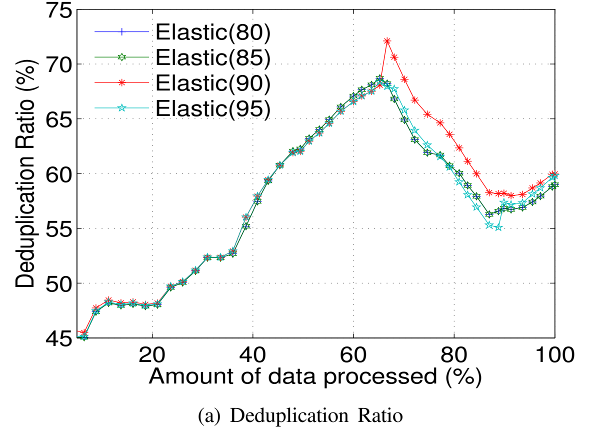
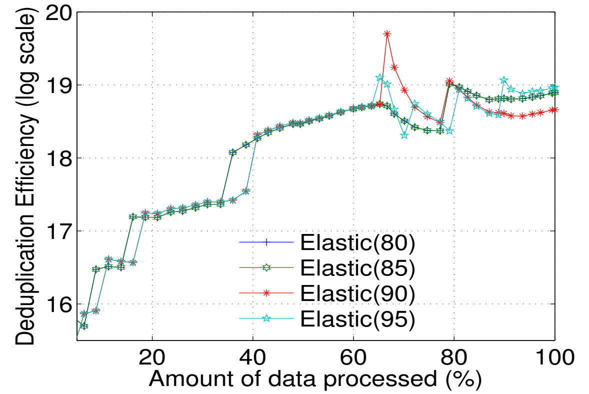


Fig. 6: EAD Performance under different initial index sizes



(a) Deduplication Ratio



(b) Deduplication Ratio

Fig. 7: The performance under different migration triggering values. Shown are results of EAD performance when the measured deduplication ratio fall below $80\%, 85\%, 90\%$ and 95% of estimated one, while $\Delta = 2$.

size of one fifth to existing solutions in Section V-B. Because the elasticity feature of EAD is supposed to be low memory tolerant that allows us to give a very low RAM space for index at the beginning, since it is able to migrate index table if there's no enough space.

We explore and verify its performance under different initial memory sizes for index. Table II shows the normalized deduplication ratio as data comes in when initial index entry slots are 100K, 150K and 200K, respectively.

It is not a surprise that a smaller index table helps us detect less duplicate data, but the gap is only at most approximately 4%. Another more interesting observation is that by applying our algorithm, initially smaller index table case sometimes is able to claim even higher deduplication ratio. This is because it has a higher probability to be migrated so that there will be more index adjustments which brings performance improvement.

Fig. 6 shows the *Deduplication Efficiency* of EAD with different initial memory sizes. It shows that the most conservative RAM initialization case claims the highest deduplication efficiency, which also proves that EAD provides well balance between memory and storage savings.

Impact of Γ. We then step further to verify the effectiveness of EAD under different policies. Fig. 7 shows the performance when we apply different values of Γ, while initial index entry slots are 100K. As we analyzed, Γ represents the system's tolerance to duplicate data detection missing. The higher the trigger is, it'll be more sensitive and easily to trigger migration, and vice versa. A higher Γ guarantees a higher deduplication ratio as shown in Fig. 7(a), although not too much in this case. However, we notice that $\Gamma = 0.95$ case also yields the highest overall efficiency, which implies that EAD is able to achieve double-win on both deduplication ratio and efficiency.

Memory usage comparison. Since our goal is to introduce a comparable elasticity-aware deduplication solution to existing approaches. Based on above results, we are able to estimate the integrated memory overhead in EAD, and we here compare it to state-of-art. Aside of memory space needed for index, EAD requires extra space for estimation.

As shown in Table III, the extra memory overhead of EAD mainly comes from the estimation part, compared with the other two. Even though, the total RAM cost by EAD is less than 50% and 5% of that by `DownSample` and `FullIndex`, respectively. Also note that there is 0.1 MB of index incrementation at the end of deduplication because of its conservative migration mechanism.

VI. Conclusion and Future work

As a significant technique for eliminating duplicate data, deduplication largely reduces storage usage and bandwidth

	2000	3000	4000	5000	6000	7000	8000	9000	9600
1×10^5(6.4 MB)	99.73%	99.08%	96.23%	94.76%	99.66%	99.11%	97.13%	94.41%	93.93%
1.5×10^5(9.6 MB)	99.79%	98.97%	99.14%	98.74%	98.25%	97.63%	97.31%	96.70%	96.61%
2×10^5(12.8 MB)	99.79%	99.62%	99.72%	99.17%	98.60%	98.88%	98.69%	97.89%	97.72%

TABLE II: The normalized deduplication ratios while we deploy different amount of memory for index. The ratio is measured as every 200 incoming data segments have been processed, $\Gamma = 0.9$.

	Initial Index(MB)	Final Index(MB)	Est.(MB)	Total(MB)
EAD	6.40(1×10^5 slots)	6.50(106581 slots)	4.32	10.82
Down-sample	32 (5×10^5 slots)	25.91(404818 slots)	0	25.91
Full Dedup	640(10×10^5 slots)	220.23(3441107 slots)	0	220.23

TABLE III: The RAM cost is broken into index and estimation(Est.) parts for analyzing under different deduplication strategies. $\Gamma = 0.95$ and $\Delta = 2$, respectively in this case.

occupation in the enterprise backup systems; Implementation of Sampling further solves both chunk-lookup disk bottleneck problem and limited memory. However setting sampling rate only with the consideration of memory size cannot guarantee the performance of the whole system. We hereby proposed the elasticity-aware deduplication solution, in which deduplication performance and memory size are both considered. We detailedly showed EAD's efficient adjustment on sampling rate by case analysis which shows that EAD claims much better performance than existing algorithms, offering a complete guideline for its large scale deployment.

Directions for future research mainly focus on the large scale implementation of our proposed solution. We are aiming to build such a deduplication infrastructure, verifying its property of elasticity and explicitly demonstrating the space saving on both storage and memory. Another long-term goal is to explore its application on distributed environment in a even larger scalability.

REFERENCES

[1] D. Geer, "Reducing the storage burden via data deduplication," *Computer*, 2008.

[2] B. Calder, J. d. Wang *et al.*, "Windows Azure Storage: a highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011.

[3] "Amazon S3, Cloud Computing Storage for Files, Images, Videos," Accessed in 03/2013, http://aws.amazon.com/s3/.

[4] T. T. Thwel and N. L. Thein, "An efficient indexing mechanism for data deduplication," in *Current Trends in Information Technology (CTIT)*, 2009.

[5] K. Srinivasan, T. d. Bisson *et al.*, "iDedup: Latency-aware, inline data deduplication for primary storage," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*, 2012.

[6] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: large scale, inline deduplication using sampling and locality," in *Proccedings of the 7th conference on File and storage technologies*, 2009.

[7] F. Guo and P. Efstathopoulos, "Building a high performance deduplication system," in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, 2011.

[8] A. Adya, B. di *et al.*, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment," *ACM SIGOPS Operating Systems Review*, 2002.

[9] G. Forman, K. Eshghi, and S. Chiocchetti, "Finding similar files in large document repositories," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005.

[10] U. Manber *et al.*, "Finding similar files in a large file system," in *Proceedings of the USENIX winter 1994 technical conference*, 1994.

[11] A. Sabaa, P. d. Kumar *et al.*, "Inline Wire Speed Deduplication System," 2010, US Patent App. 12/797,032.

[12] L. L. You and C. Karamanolis, "Evaluation of efficient archival storage techniques," in *Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2004.

[13] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams," in *Proceedings of the 8th USENIX conference on File and storage technologies*, 2010.

[14] J. Min, D. Yoon, and Y. Won, "Efficient deduplication techniques for modern backup operation," *IEEE Transactions on Computers*, 2011.

[15] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *ACM SIGOPS Operating Systems Review*, 2001.

[16] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," *Hewlett-Packard Labs Technical Report TR*, 2005.

[17] W. Xia, H. d. Jiang *et al.*, "Silo: a similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput," in *Proceedings of USENIX annual technical conference*, 2011.

[18] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, 2009.

[19] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, 2008.

[20] G. Lu, Y. Jin, and D. H. Du, "Frequency based chunking for data de-duplication," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010.

[21] C. Kim, Park *et al.*, "Rethinking deduplication in cloud: From data profiling to blueprint," in *Networked Computing and Advanced Information Management (NCM)*, 2011.

[22] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems," in *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST12)*, 2012.

[23] P. Kulkarni, F. Douglis, J. LaVoie, and J. M. Tracey, "Redundancy elimination within large collections of files," in *Proceedings of the USENIX Annual Technical Conference*, 2004.

[24] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (TOS)*, 2012.

[25] C. A. Waldspurger, "Memory resource management in VMware ESX server," *ACM SIGOPS Operating Systems Review*, 2002.

[26] F. Travostino, P. d. Daspit *et al.*, "Seamless live migration of virtual machines over the MAN/WAN," *Future Generation Computer Systems*, 2006.

[27] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, 2005.

[28] M. Dutch, "Understanding data deduplication ratios," in *SNIA Data Management Forum*, 2008.

[29] M. Hibler, L. d. Stoller *et al.*, "Fast, Scalable Disk Imaging with Frisbee," in *USENIX Annual Technical Conference, General Track*, 2003.

[30] D. Harnik, O. Margalit, D. Naor, D. Sotnikov, and G. Vernik, "Estimation of deduplication ratios in large data sets," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, 2012.

[31] "Wikimedia Downloads Historical Archives," Accessed in 04/2013, http://dumps.wikimedia.org/archive/.

[32] "OpenfMRI Datasets," Accessed in 05/2013, https://openfmri.org/data-sets.

[33] B. Debnath, S. Sengupta, and J. Li, "ChunkStash: speeding up inline storage deduplication using flash memory," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, 2010.

[34] J. H. Burrows, "Secure hash standard," DTIC Document, Tech. Rep., 1995.