2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

One Size Does Not Fit All: The Case for Chunking Configuration in Backup Deduplication

Huijun Wu*†, Chen Wang*, Kai Lu§, Yinjin Fu§, Liming Zhu*†
*Data61, CSIRO
†University of New South Wales, Australia
‡Army University of Engineering, China
§ College of Computer, National University of Defense Technology, China

***Abstract*—Data backup is regularly required by both enterprise and individual users to protect their data from unexpected loss. There are also various commercial data deduplication systems or software that help users to eliminate duplicates in their backup data to save storage space. In data deduplication systems, the data chunking process splits data into small chunks. Duplicate data is identified by comparing the fingerprints of the chunks. The chunk size setting has significant impact on deduplication performance. A variety of chunking algorithms have been proposed in recent studies. In practice, existing systems often set the chunking configuration in an empirical manner. A chunk size of 4KB or 8KB is regarded as the sweet spot for good deduplication performance. However, the data storage and access patterns of users vary and change along time, as a result, the empirical chunk size setting may not lead to a good deduplication ratio and sometimes results in difficulties of storage capacity planning. Moreover, it is difficult to make changes to the chunking settings once they are put into use as duplicates in data with different chunk size settings cannot be eliminated directly. In this paper, we propose a sampling-based chunking method and develop a tool named SmartChunker to estimate the optimal chunking configuration for deduplication systems. Our evaluations on real-world datasets demonstrate the efficacy and efficiency of SmartChunker.**

I. Introduction

The exponential growth of data volumes makes it necessary to explore techniques such as data deduplication to make data manageable and reduce the archive or backup cost. Although large-scale systems like Data Domain [1] are provided for enterprise-level data backup, a large amount of critical business data sits solely on users' PCs or on the edge of the Internet. It is therefore important to have a backup solution specifically designed around business endpoint data protection. There are several commercial products (e.g., Acronis [2], NovaBackup [3] and AOMEI Backupper [4]) providing software or service for personal data backups. Data deduplication is a mechanism used in this kind of systems to reduce the volume of backup data, thus saving storage capacity. Compared to the large-scale systems which often do inline or hybrid deduplication on the data, personal data backup is often performed offline. In the process of data deduplication, one critical step is data chunking. Specifically, data are split into chunks and the hash values of these chunks are used as fingerprints. Duplicate chunks are detected if their fingerprints match with existing ones.

Existing chunking algorithms can be classified into fixed-sized chunking methods (FSC) and content-defined chunking (CDC) algorithms. Fixed-sized chunking splits the data into chunks with the same size. Its limitation is that it cannot deal with the boundary-shift problem [5]. The boundary-shift problem can be addressed by content-defined chunking (CDC) algorithms. CDC algorithms generate variable-sized data chunks by cutting the data based on the local content. Besides the classical rabin fingerprint algorithm [6], there are many alternatives for CDC chunking [7], [8], [9]. For different workloads, different chunking algorithms or different data chunking settings may result in drastically different deduplication efficiency.

In a typical content-defined chunking algorithm, one often needs to set the max, min and average chunk size to avoid the high variation in chunk sizes [10]. According to existing study on backup workloads [11], smaller chunk sizes (e.g.,4KB and 8KB) tend to produce higher deduplication ratio and an average chunk size of 8KB seems to be a sweet spot in most cases. Many existing systems empirically set the average chunk size to 4KB or 8KB [12], [13], [14]. Unfortunately, due to the overhead incurred for storing deduplication metadata, smaller chunk sizes may not achieve good deduplication ratios as one may think [15]. In some cases, a deduplication system may achieve a better deduplication ratio and I/O performance (i.e., fragmentation [16]) by setting to a larger chunk size. It is difficult to change the chunk size once set because the data are already chunked based on the previous setting and changing the chunk size requires re-chunking data using the new chunk size. The process is costly. Therefore, setting the chunk size properly is important in a deduplication system.

Some previous research proposes to set different chunk sizes for data of different file types [17]. Although the empirical optimal chunk size values for the data of different applications (thus different types) are reasonable approximations for the optimal chunk sizes, this method faces the following obstacles in practice. First, the number of applications can be huge therefore it may not be feasible to derive a reasonable chunk size for each application type. Second, we observe a large number of binary files without explicit file types in a real-world dataset (see Section II). A brute-force approach may apply different chunk sizes on the data in a storage system and calculate their deduplication ratios to select the best one.

0-7695-6410-0/18/$31.00 ©2018 IEEE
DOI 10.1109/CCGRID.2018.00036

However, such an approach incurs high computing and data access overhead.

To address the above-mentioned problems, we propose a method and develop a tool named SmartChunker to assist with the chunking configuration selection for deduplication systems. Without scanning the whole data, SmartChunker samples data on the disk and uses different chunk size settings to estimate the potential capacity savings under different chunk sizes. The chunk size corresponding to the most capacity savings is used in the deduplication system.

Essentially, predicting the storage capacity savings of chunking configurations (we call it capacity saving estimation in the following part of the paper) is equivalent to estimating the deduplication ratio of a large dataset. Simply using the deduplication ratio of samples does not work because the deduplication ratio of samples does not reflect the distribution of duplicate chunks and it mostly underestimates the deduplication ratio. Statistically, for two datasets with different deduplication ratios, it is highly possible that the samples from one dataset have a higher deduplication ratio than those from the other dataset even when the real deduplication ratio of the latter is higher than that of the former. Accurate estimation methods have been investigated by existing work [18], [19], [20]. In the chunk size setting problem, the trade-off between the estimated deduplication ratio and the overhead introduced by maintaining deduplication metadata also needs to be taken into account. The deduplication metadata include the fingerprint table, recipe table, and auxiliary data structures.

There are two main challenges for chunk size estimation. First, it requires a high estimation accuracy. Deciding the chunk size directly affects the storage space saving, which is different from obtaining a rough approximation for the deduplication ratio for evaluating the cost-effectiveness [19]. Quantifying the impact of different chunk sizes is a challenging problem. Second, it requires the estimator to be able to deal with variable chunk sizes. The content-defined chunking algorithms generate chunks with different sizes and variable sizes add to the complexity of estimation. Moreover, the efficiency of an estimation algorithm is also important as the amount of data to process is often non-trivial.

Most of the existing deduplication ratio estimation methods do not fulfill the above-mentioned requirements well. Specifically, some of them often require to read every block from the disk volume and compute their fingerprints to perform the estimation [19], [20]. This is infeasible for chunking size estimation as it would be time-consuming to compute the hash values for all the data with different chunk size choices. On the other hand, the estimation error is not bounded by these methods.

The latest research on deduplication ratio estimation is the unseen method [18] derived from [21], [22]. Compared to previous methods, this estimation algorithm can avoid reading all the blocks from disks while providing theoretically guaranteed accuracy. However, using the unseen method to estimate the capacity savings of different chunk sizes is not as direct as one may think. In this paper, we extend unseen estimation to take into account of the characteristics of content-defined chunking. Specifically, rather than directly sampling the data blocks according to the chunk sizes, we propose a file-package based sampling method. This is mainly for avoiding too many small random reads which take much longer time. Moreover, we perform the unseen method for each popular chunk size for accurate estimation. Meanwhile, we group the chunks from unpopular chunk sizes to avoid the estimation error caused by insufficient samples. We evaluated SmartChunker on real-world datasets. We found out that we are able to choose the best chunking configuration among various choices by the estimation. SmartChunker uses significantly shorter time than the full scan baseline and yields more accurate estimation than the application-aware heuristics [17].

The rest of the paper is organized as follows. In Section II, we present the background and motivation for this research. We describe the details of the design and implementation of SmartChunker in Section III. We give the experimental results in Section IV and conclude the paper in Section VI.

II. Background and Motivation

Properly choosing parameters for data chunking is critical for achieving high deduplication ratios with low overhead. In existing deduplication systems, the chunking configurations are often chosen empirically, which may result in uncertainty in the system performance in some cases. If we do not consider the storage overhead of deduplication metadata like fingerprint table and recipe tables, small chunk sizes are always better as duplicates can be identified in a finer granularity. However, the number of metadata entries increases as the chunk size decreases. When the storage overhead of the metadata is taken into account, choosing the optimal chunk size is, essentially, to find the best balance point between the storage space of the eliminated duplicate chunks and the overhead introduced by storing the deduplication metadata.

For example, if the chunk size for a deduplication system is set to 8KB,a common configuration in deduplication systems, the metadata overhead of the system is high when a user mainly uses this system to backup data whose redundancies exist as large duplicate files. In this case, a larger chunk size is better since the system is still able to identify the duplicates but the overhead for storing the deduplication metadata shrinks. Moreover, a larger chunk size also leads to a faster data restore process.

Some previous study proposes application-aware chunk settings [17] for files from different applications. Their rationale is that different files often have some common optimal chunk sizes. Unfortunately, by looking into the real-world dataset [23], we observe that binary files or user-defined files without any explicit file types commonly exist in the backup workloads. For example, Figure 1 shows the data size of the top 10 file types for user14 in *FSL Home dataset*. The files with unknown type occupy much more storage space than other known file types. Nowadays, since there are a massive number of applications and developer-defined types, it is common to see new types unknown by the chunking method. In addition,

some file types may not be informative for the chunk size setting. Simple examples are "tar"/"iso" files that contain a variety of files with different file types. In this case, the application-aware chunk size setting methods [17] do not work well.

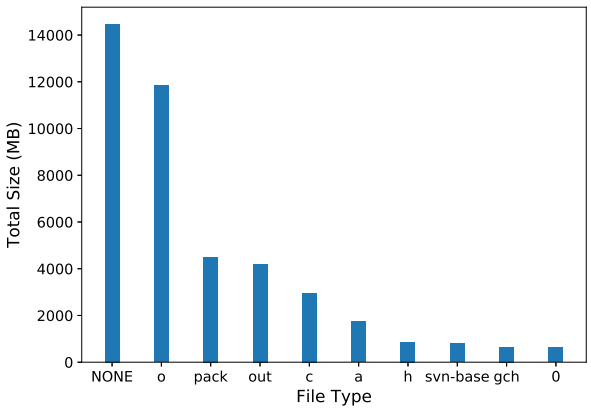


Fig. 1: The total sizes of top10 file types for user014 in FSL Home dataset. NONE means the file type is unknown.

All these motivate the question we attempt to answer in this paper: *Is it possible to accurately and efficiently predict the storage space saving under different chunking configurations?*

III. Design and Implementation of SmartChunker

In this section, we describe the detailed design and implementation of SmartChunker.

SmartChunker is designed as a tool to estimate the best chunking configuration for a given dataset under a specific chunking algorithm. Specifically, we focus on the offline deduplication procedure where data is already stored on disks.

A. Unseen Estimation for CDC Chunking Methods

SmartChunker extends the unseen estimation algorithm to accurately estimate the storage capacity saving for different chunk sizes. In the following, we first give a brief introduction of the basic unseen estimation algorithm [18], and then we discuss how to extend the algorithm to estimate storage space saving under different chunk size settings. Before describing the algorithm, we define two concepts as below:

- A **fingerprint frequency histogram (FFH)** for a dataset is a histogram $f = f_1, f_2, ...$ where the value of f_i is the number of fingerprints that appear exactly i times in the dataset.
- **Chunk fingerprint samples** are a portion of fingerprints for data chunks uniformly sampled from a large dataset.

The target of the unseen estimation algorithm is to use the observed FFH of the sampled chunks to compute the number of unique chunks in the whole dataset. According to the entropy estimation theory [22], the theoretical value of H_s (denoted by $H_s^{'}$) can be computed by $H_s^{'} = T(p) \cdot H$ where matrix $T(p)$ is computed by a series of binomial probabilities with the sampling rate p as the parameter. As the real value of

Algorithm 1: Basic Unseen Estimation Algorithm

Input: H_s^o: The obeseved FFH for chunk fingerprint samples s, H: FFH for the fingerprints of the whole dataset. p:sampling rate, N:total chunk numbers of the whole dataset.

Output: Estimated number of unique chunks.

Subroutine `Unseen Estimation Algorithm()`

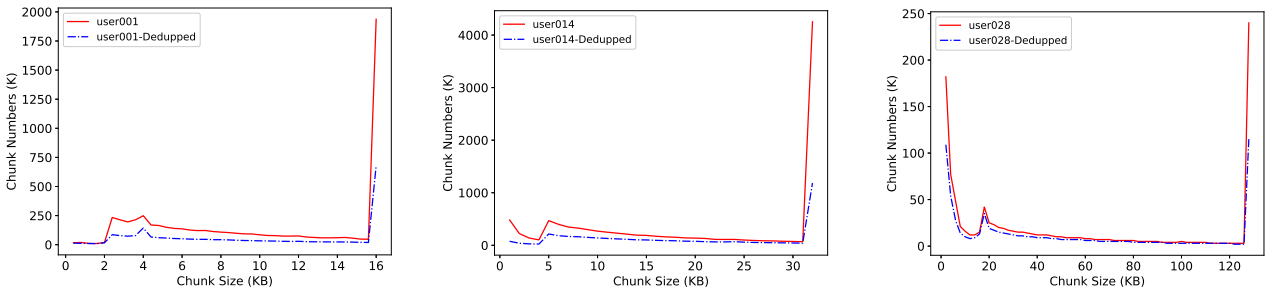
- Compute the transformation matrix $T(p)$ by binomial probabilities;
- $H_s^{'} = T(p) \cdot H$;
- Solve the linear programming:
  - Objective function: $min(\Delta(H_s^o, H_s^{'}))$, in which $\Delta(H_s^o, H_s^{'}) = \Sigma_i \frac{1}{\sqrt{H_s^o[i]+1}} |H_s^o[i] - (T \cdot H)[i]|$.
  - under constraints:
    - $\Sigma_i H[i] = N$
    - $\forall i \quad H[i] > 0$
  - return $N_{unique} = \Sigma_i H[i]$

H_s^o has been observed, we minimize the distance between the observed H_s (H_s^o) and its theoretical value $H_s^{'}$ to estimate H. This then becomes an optimization problem of linear programming. From H, we can then obtain the number of unique data chunks which is just the sum of the data chunks that appear for different times.

The vanilla unseen estimation algorithm cannot be applied directly to serve the purpose of SmartChunker due to the following two main reasons. First, the original unseen algorithm considers that each chunk has the same size. Although it can be used for fixed-sized chunking where all chunks have the same size, SmartChunker focuses more on content-defined chunking which is more practical for deduplication systems. Second, the original unseen algorithm does not consider multiple rounds of sampling, each with a different chunk size setting.

To enable variable-sized chunk support for the unseen estimation algorithm, SmartChunker performs the estimation for the chunk sizes separately. The computational overhead is acceptable because the execution time for the unseen algorithm is short compared with the sampling time which involves many disk I/O operations. Moreover, since the estimation of the chunks with different sizes is independent of each other, the computation can be fully paralleled in the implementation.

However, the individual estimations on different chunk sizes do not work well. The key reason is that not all chunk sizes are equal in the estimation. We observed that the chunk size distribution for the data chunks after content-defined chunking are often highly skewed. Figure 2 shows the distribution of chunks with different chunk sizes before and after the deduplication, respectively. The gap between the two lines in each figure indicates the duplicates of certain-sized chunks. An interesting phenomenon is that for a certain range of chunk sizes, the more chunks exist before deduplication, the more duplicates would be identified during the deduplication process. This is consistent with the findings of some previous study [15] that



(a) User001 (b) User014 (c) User028

Fig. 2: The chunk size distribution for the traces of three users in FSL Home dataset. Red lines show the distribution before deduplication while the blue lines show the distribution after deduplication. For user1, user14 and user28, the average chunk sizes are 8KB, 16KB and 64KB, respectively.

redundant data chunks are often highly referenced.

This skewness makes it improper to perform the unseen estimation directly to estimate the unique chunks for each chunk size. Furthermore, the major part of the capacity saving can be estimated from the popular chunk sizes since capacity saving mainly come from popular chunk sizes. On the other hand, the estimation may work poorly on unpopular chunks. The key reason is that few samples can be drawn from the unpopular chunks so that the unseen estimation algorithm becomes error-prone due to the insufficient samples. According to the theoretical analysis [21], an accurate estimation requires $O(\frac{n}{logn})$ samples. For some unpopular chunk sizes, since they may only have several chunks, we may not even be able to get any samples by uniform sampling.

To address this, we propose a grouping based capacity saving estimation algorithm (see Algorithm 2). For the popular chunk sizes, we do the estimation directly and use each chunk size to compute the capacity saving. For the unpopular portion of chunk sizes, we group the chunks of a range of chunk sizes together to conduct the estimation. The potential inaccuracy that might be introduced by this method is that we do not have the information about the size of the duplicate chunks. What we get from unseen estimation is just the number of duplicate chunks. To estimate the capacity savings, we use the mean of the chunk size in the unpopular chunk size group $\overline{C}_{G_u} = \sum_{i=1}^{k} p_i \cdot C_i$ as the expected chunk size where k is the number of different chunk sizes in a group. This inevitably introduces error in the unseen estimation. However, it can be proved that the error is trivial as follows.

The unpopular chunk sizes are mostly continuous due to the skewness of the chunk size distribution. For an unpopular chunk group p, suppose the number of duplicate chunks in the group is D, the total number of chunks in a group is N and the chunk sizes in the group are $C_i (i = 1, 2, ..., k)$. N_i denotes the number of chunks with size C_i. The estimated capacity saving S is therefore as below:

$$S = \sum_{i=1}^{k} \frac{N_i \cdot C_i \cdot D}{N} \tag{1}$$

In the worst case, the capacity saving may all come from the smallest chunks. In the best case, the capacity saving all comes from the largest chunks. As a result, the range of real capacity saving is $[S_{low}, S_{high}]$.

$$S_{low} = D \cdot min_{i \in [1,k]}(C_i) \tag{2}$$

$$S_{high} = D \cdot max_{i \in [1,k]}(C_i) \tag{3}$$

Therefore, for a group, the largest error introduced by the group chunk size approximation is as below:

$$E = max(|S - S_{low}|, |S - S_{high}|) \tag{4}$$

To compute E, we have the following:

$$E = S_{high} - S \leq (S_{high} - S_{low}) \cdot D \tag{5}$$

Here we assume $|S - S_{high}| \geq |S - S_{low}|$ since the other way is symmetric.

As each group has a sequence of contiguous chunk sizes and each sequence has a size of k, for all unpopular chunk groups, the overall error is therefore computed as shown in Equation 6.

$$\begin{aligned} \sum E &\leq \sum_{p=1}^{N_g} (max^g_{i \in [1,k]}(C_i) - min^g_{i \in [1,k]}(C_i)) \cdot D^p \\ &= (k-1) \sum_{p=1}^{N_g} D^p \end{aligned} \tag{6}$$

where N_g is the total number of unpopular chunk size groups and D_g is the number of duplicate chunks for the group g.

Algorithm 2: Capacity Saving Estimation

Input: C_{min}: min chunk size, C_{max}: max chunk size, $s = s_{C_1}, s_{C_2}, ..., s_{C_M}$ are samples drawn from each chunk size. $n = n_{C_1}, n_{C_2}, ..., n_{C_M}$ are the numbers of chunks in for samples with different chunk sizes. t is the frequency threshold for popular/unpopular chunks.

Output: Estimated capacity savings by grouping.

Subroutine `capacitySavingEstimation()`

- $S = 0$ //the capacity saving without considering metadata
- G_p is the chunk size group for popular chunk sizes
- G_u is the chunk size group for unpopular chunk sizes
- **foreach** n_{C_i} *in* n **do**
  - **if** $n_{C_i} > t$ **then**
    - $G_p.add(C_i)$
  - **else**
    - // group every k chunk sizes to be an unpopular group. G_{u_c} is the current unpopular group.
    - $G_{u_c}.add(C_i)$
    - **if** $G_{u_c}.full()$ **then**
      - $G_u.add(G_{u_c})$
    - **end**
  - **end**
  - **foreach** C_j *in range*(C_{min}, C_{max}) **do**
    - **if** C_j *in* G_p **then**
      - N_u = unseen($FFH_{s_{C_j}}$, $\frac{n_{C_j}}{p}$)
      - $S+ = C_j \cdot (\frac{n_{C_j}}{p} - N_u)$
    - **end**
  - **end**
  - **foreach** G_{u_k} *in* G_u **do**
    - N_u = unseen($FFH_{s_{C_p}}$ for C_p in G_{u_k}, $\frac{n_{G_{u_k}}}{p}$)
    - $S+ = \overline{C}_{G_{u_k}} \cdot (\frac{n_{C_j}}{p} - N_u)$
  - **end**
- **end**
- return S

Hence, the overall error $r_E = \frac{\sum E}{S_{popular} + \sum_E}$ is bounded by the group range k and the total number of duplicates in the unpopular chunk size groups. The parameter N depends on the number of samples that unseen estimation can work well while k depends on how unpopular chunk sizes are defined. More intuitively, in our evaluation on *FSL Home dataset*, if we take the sampling rate of 15% as an example, the unseen estimation works well with 50 samples in each group. For this setting, $k = \frac{50/0.15}{4} \approx 83$ is a reasonable value as each unpopular chunk size is observed to have 4 chunks in average. Note that the threshold parameter t in Algorithm 2 is also set according to k. As the duplicate chunk numbers in unpopular chunk groups are very small (i.e., less than 10% of the total duplicate chunks for the traces we evaluated). Therefore, the error rate will be less than $\frac{83 \times 0.1}{C_e \times 0.9 + 83 \times 0.1}$ where we use the expected chunk size C_e to approximate the capacity savings from the popular chunks. Obviously, the error rate will become larger for smaller chunk size settings. Even though, the extra error introduced beyond the error of unseen estimation will be less than 0.45% for the expected chunk size of $2KB$.

B. The Sampling Method

The accurate estimation of unseen algorithm relies on proper sampling. Harnik et al. [18] propose to sample at the granularity of large segments (1MB) and then further chunk the segment into 4KB blocks. However, direct sampling on disk volumes is not suitable for content-defined chunking. The reason is that the sampled segment may cross file boundaries while in practice chunks are cut from individual files. To address this problem, we force the segment points to be aligned with file boundaries while roughly maintaining the large segment sizes.

C. The Metadata Overhead

So far, Algorithm 2 does not consider the metadata overhead. The main reason that smaller chunk sizes may perform worse is the overhead introduced to store the deduplication metadata including fingerprint table, recipe table, etc. For example, when the duplicates often show strong locality and tend to appear as long sequences, chunking the files into small chunks (e.g., the average chunk size is 4KB) does not improve the deduplication ratio but increases the metadata overhead.

For a given dataset, we define the total number of chunks with size c_i before deduplication as N_{c_i}. The data size before deduplication S_L will then be $S_L = \sum N_{c_i} \cdot c_i$ and the data size after deduplication is $S_P = \sum N^u_{c_i} \cdot c_i$. The raw deduplication ratio without considering metadata overhead is $D = \frac{S_L}{S_P}$.

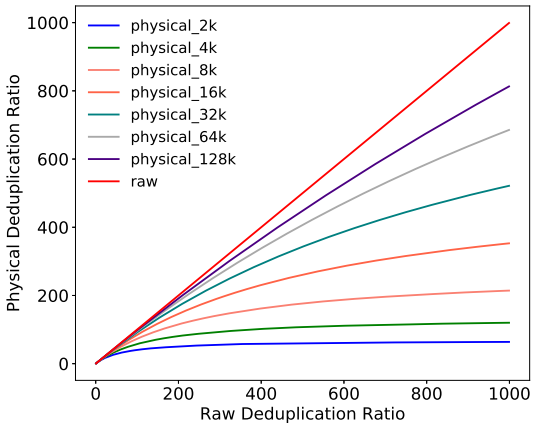


Fig. 3: The relationship between logical deduplication ratio and physical deduplication ratio.

Let m denote the relative size of the metadata for each chunk compared to the chunk size. The deduplication ratio taken into account of the metadata overhead (physical deduplication ratio) is $D' = \frac{S_L}{S_P + m \cdot (S_L + S_P)} = \frac{D}{1 + m \cdot (D+1)}$. This

equation characterizes the relationship between the raw deduplication ratio and the physical deduplication ratio. Figure 3 illustrates the relationship, in which m is set to the same value in [15]. Note that for a certain deduplication system, m needs to be set according to specific implementations of metadata data structures. Compared to the raw deduplication ratio, the physical deduplication ratio considers the overhead of metadata. Their relationships give very interesting hints for setting the chunk size. Although a smaller chunk size gives a higher raw deduplication ratio than a larger chunk size, it does not ensure a higher physical deduplication ratio when the metadata overhead is taken into account.

IV. Evaluation

We implement the prototype of SmartChunker in Python and evaluated the implementation on FSL datasets [23], one of the most widely used datasets for backup deduplication research. The dataset consists of the traces for 38 users from 2011 to 2014. The experiments are conducted on a server with Intel Xeon E5 CPU, 64GB memory and 2TB 7200RPM HDD. Traces are downloaded to the file server due to its large size. Each experiment is repeated three times to measure the variance. The results are described as follows.

1) Estimate the optimal chunk size for different users: We conduct experiments to show whether SmartChunker can accurately estimate the optimal chunk size for the data of each user. We choose 9 active users in 2012 in the following experiments. We use the trace to simulate a daily incremental backup and weekly full backup scenario. For each user, three months' traces are used.

In order to evaluate how the estimated capacity saving under different chunk sizes matches the situation of the whole data, we compare the normalized physical deduplication ratio between the estimation and the ground truth obtained from the traces. The use of normalization is due to the fact that we focus on the relative capacity savings for different chunk sizes rather than the specific values. Figure 4 shows the estimated raw deduplication ratio (Raw Estimation), the estimated physical deduplication ratio with metadata storage overhead taken into account (Corrected Estimation) and the real physical deduplication ratio (Real). The sampling rate is set to 10% since the results are stable when the sampling rate is above 10%. After correcting the raw deduplication ratio with the metadata overhead, the estimated physical deduplication ratio is able to accurately approximate the trend of the real deduplication ratio under different chunk sizes. For different users, the capacity savings under different chunk sizes vary significantly. Some users are not sensitive to the change of chunk sizes. For example, the amount of capacity saving variation is less than 5% for User 3 and User 8. For User 12, when the chunk size increases from 2KB to 128KB, the capacity saving drops more than 40%. Large chunk sizes improve the deduplication ratios for User 17 and User 18. In particular, there is more than 70% improvement in storage capacity saving for user 18 under the chunk size of 128KB than under the chunk size of 2KB. Based on the estimation given by SmartChunker, a user can choose a proper chunk size that fits their performance requirements. For example, if the performance requirement for data restore is high for user 8, choosing a large chunk size and sacrifice 2% of capacity saving is appropriate. As shown in the dataset, the optimal chunk sizes for different users vary from 2KB to 128KB. It demonstrates the diversity of user data access patterns so that estimating the impact of chunk size choices is important before applying deduplication.

We also compare the deduplication performance of SmartChunker and ALG-Dedup [17] (see Figure 5). ALG-Dedup sets the chunk size based on file types. There are files without explicit file type information. We set a default chunk size of 8KB in ALG-Dedup for these files as 8KB is considered to be optimal in most workloads by previous study[11].

For user U5, U8, U16, and U28, we observe that virtual machine images or vmdk files dominate their storage usage. For these cases, ALG-Dedup and SmartChunker produce nearly identical chunk sizes. However, for user U12, U17, and U18, a large number of files do not have file type information. ALG-Dedup fails to produce optimal chunk sizes to data of these users. Moreover, there are also some uncommon file types such as svn-base ALG-Dedup fails to infer proper chunk sizes. Note that the y-axis is log scaled so that the differences between ALG-Dedup and SmartChunker are significant for some users. Essentially, the idea of application-aware chunk size setting is to differentiate the duplicate patterns of different files belonging to different types. In some extreme cases, duplicates may exist as copies of the same file rather than different versions. Our estimation method is able to infer a proper chunk size for the better storage space saving. It also gives users better understanding of the performance implication of different chunk size choices so that a balance between storage space saving and data restoring speed can be achieved for different system requirements.

2) Comparing with the full scan method: One objective of using estimation to infer the optimal chunk size is to reduce the cost of scanning the whole data on disks to obtain the optimal chunk size value (called full scan method). We have already shown that the accuracy of our estimation method. In the following, we compare SmartChunker with the full scan method on their execution time.

The FSL dataset is provided as the fingerprints of the data chunks together with file metadata so that we cannot evaluate the time of sampling, chunking, and hash computations directly. Instead, we evaluate the execution time of SmartChunker on a dataset that consists of 167GB data in a desktop computer. To evaluate the execution time, we use SHA-1 as the hash function to compute the fingerprints. Note that the potential collision rate has little impact on the whole deduplication ratio. We use Rabin fingerprint as the chunking algorithm. Note, there are other chunking algorithms (e.g., [7]) that are faster.

The procedures of the baseline method are the followings. First, a full scan is required to read all the data from the disk. Then files are chunked to different sizes and the fingerprints

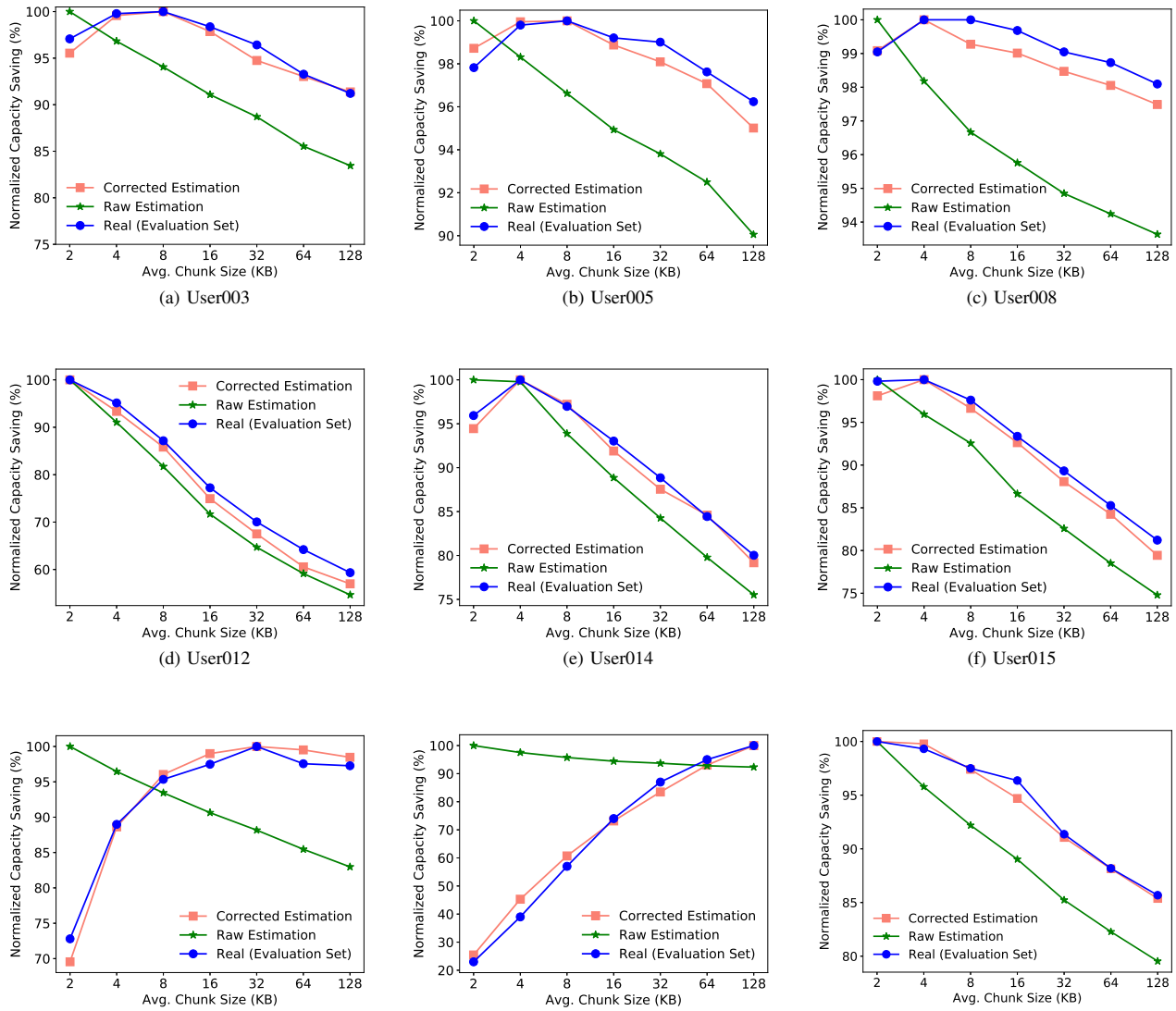


Fig. 4: The storage capacity savings for different users in FSL dataset. For each line on the plots, the y-axis is normalized by dividing the maximum value. The green lines are the estimation without considering the deduplication metadata overhead while the red lines are the physical deduplication ratio. The blue lines demonstrate the ground truth.

need to be computed for each chunk. For each chunk size setting, the deduplication ratio is computed by the number of duplicate chunks and the sizes of these chunks.

Table I shows the comparison of execution time for the full scan method and SmartChunker. The execution time of chunking and hash computing is evaluated by the implementation of destor [24]. It is clear that SmartChunker is much faster than the baseline method. This is mainly due to that our sampling method avoids the full disk scan. Sampling also avoids a large amount of computing on chunking and hashing. The estimation time is longer as one may expect. The main reason is that for each chunk size setting (i.e., a tuple of (max, min, avg)), SmartChunker computes the estimated duplicates for multiple groups of data with variable-sized chunks.

To give a more intuitive feeling about the execution time, we use destor to perform deduplication under the chunk size of 8KB on the data and the total deduplication time is 5,092s. This indicates that the baseline method uses around twice the deduplication time to get the optimal chunk size while the overhead introduced by SmartChunker is only around 20% of the total deduplication time. Further, we also found that even if we used some weaker hash functions like MD5, the estimation results barely changed. This indicates that further performance improvements are also achievable.

TABLE I: Execution Time for baseline method and SmartChunker. The sampling rate is set to 10%.

	Sampling	Chunking	Hash Computing	Estimation	Total
Baseline	946s	4091s	4379s		9416s
SmartChunker	167s	397s	413s	138s	1115s

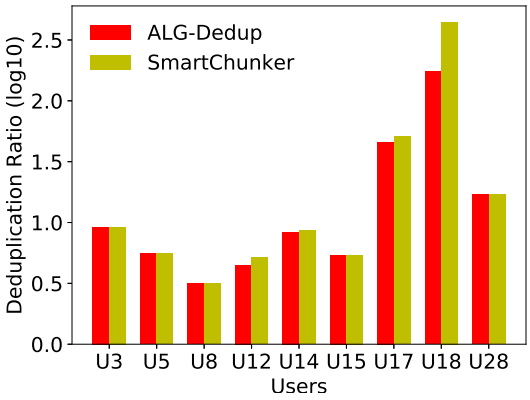


Fig. 5: The deduplication under the chunk size given by ALG-Dedup and SmartChunker for users.

3) Sensitivity to Sampling Rate: We also studied the sensitivity to the sampling rate for SmartChunker. On the workloads we evaluated, we found that the sampling rate between 10% and 15% work well. Unlike in [18] where unseen estimation is used for getting an accurate estimated value of the deduplication ratio, SmartChunker can tolerant a little bit higher inaccuracy since our focus is the relative comparisons of capacity savings between different chunk sizes. Hence, what we are interested in is how does the sampling rate affect the correct selection of the optimal chunk size. We present the result of three representative users to demonstrate this. Since each run of the estimation is based on independent samples, they may result in slightly different results. Hence, each experiment was repeated for three times and the average numbers are presented.

As shown in Figure 6c, p is the sampling rate in the estimation. In general, larger sampling rate delivers more accurate estimation. As we can see, even with the sampling rate of 5%, SmartChunker can still relatively capture the trend of the differences between chunk sizes very accurately (especially for User18). Nevertheless, it is noteworthy that if the capacity savings corresponding two chunk sizes are quite close (e.g., 4KB, 8KB for User005), it is hard to differentiate them due to the estimation error.

V. Related Work

CDC algorithms normally have large chunk size variance. That means there exist many chunks with very large or small chunk sizes. Some recent studies address this by imposing minimum and maximum chunk size restrictions. Moreover, Rabin fingerprint algorithm also has a high computational overhead which is alleviated by some recent explorations like Gear [25] and AE [7]. Some other works [26], [27] further improve the deduplication ratio by re-chunking non-duplicate chunks or reduce the metadata overhead by merging consecutive duplicate chunks [28].

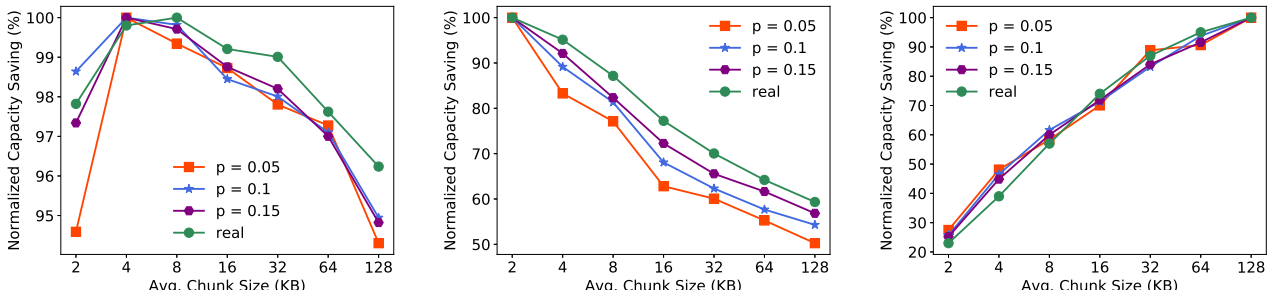
As a critical parameter for deduplicated storage systems, different chunk size settings have been recommended by practitioners. For example, Symantec recommends 16KB or larger chunk sizes [29]. Commuvault suggests to use large chunk size like 128KB [30]. Some deduplication systems for dedicated purposes (e.g., virtual machine images) even use 4096KB as the default chunk size [31]. However, it is obvious that there is no one chunk size configuration which fits all scenarios. Hence, SmartChunker aims to provide a practical tool for users to analyze the potential capacity savings for different chunk sizes in a given storage system. Compared to the empirical methods, SmartChunker can provide the system managers with more insights about the system workload to assist them to make proper settings. More broadly, the tradeoff between the deduplication efficiency and metadata overhead has also been explored by the hybrid file/chunk level deduplication in cloud deduplication [32] or using coarse/fine-grained deduplication for inline/offline deduplication [14].

An alternative way to alleviate the space occupied in deduplicated storage systems is to use specific data structures. For example, Data Domain uses Merkle tree [33] to represent files in deduplicated storage. By using the hierarchical reference for files, they can reduce the size of deduplication metadata for duplicate files. SmartChunker solves the problem for offline deduplication in a more general way. For instance, SmartChunker can also deal with the scenarios that unique data widely exist even between different backup versions. Moreover, the chunk size configuration for the leaf nodes in the Merkle tree also needs to be set carefully for Data Domain like systems so that the estimation method given by SmartChunker is also applicable.

Finally, since it achieves accurate estimation for deduplication properties, unseen estimation has also been explored to assist estimating the temporal locality in the primary deduplication scenario [34].

VI. Conclusion and Future Work

We proposed a chunk size estimation method and implemented a tool called SmartChunker in this paper to obtain the optimal chunk size configuration for backup deduplication systems. SmartChunker only needed to use a small number of samples to learn the potential storage capacity saving under different chunk configurations. Our experiments on real-world datasets demonstrated that SmartChunker clearly outperforms existing methods that set the chunk size empirically or give



(a) User005 (b) User012 (c) User018

Fig. 6: Normalized capacity savings for users under different sampling rate.

application-specific chunk configurations. We envision that learning based methods can be utilized to assist the configuration of storage systems when they need to meet the requirements of dynamic workloads.

Currently, SmartChunker mainly focuses on offline deduplication. As a future work, we will explore using learning methods to detect the drift of deduplication patterns to further improve the storage efficiency of inline deduplication systems.

Acknowledgment

The authors would like to thank Jiantao Jiao from Stanford University for insightful discussions. This work is partially supported by the The National Key Research and Development Program of China (2016YFB0200401), by the program for New Century Excellent Talents in University, by National Science Foundation (NSF) China 61402492, 61402486, 61402518, 61379146, by the laboratory pre-research fund (9140C810106150C81001), by the HUNAN Province Science Foundation 2017RS3045.

References

[1] Y. Allu, F. Douglis, M. Kamat, P. Shilane, H. Patterson, and B. Zhu, "Backup to the future: How workload and hardware changes continually redefine data domain file systems," *Computer*, vol. 50, no. 7, pp. 64–72, 2017.

[2] "Acronis storage," https://www.acronis.com/en-us/, accessed: 2017-09-30.

[3] "Novastor," http://www.novastor.com/, accessed: 2017-09-30.

[4] "Aomei backupper," https://www.acronis.com/en-us/, accessed: 2017-09-30.

[5] C. Policroniades and I. Pratt, "Alternatives for detecting redundancy in storage systems data." in *USENIX Annual Technical Conference, General Track*, 2004, pp. 73–86.

[6] M. O. Rabin *et al.*, *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[7] Y. Zhang, H. Jiang, D. Feng, W. Xia, M. Fu, F. Huang, and Y. Zhou, "Ae: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 1337–1345.

[8] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Q. Liu, and Y. Zhang, "Fastcdc: a fast and efficient content-defined chunking approach for data deduplication." in *USENIX Annual Technical Conference*, 2016, pp. 101–114.

[9] B. Agarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "Endre: An end-system redundancy elimination service for enterprises." in *NSDI*, 2010, pp. 419–432.

[10] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5. ACM, 2001, pp. 174–187.

[11] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems." in *10th USENIX Conference on File and Storage Technologies (FAST 10)*, 2012.

[12] B. K. Debnath, S. Sengupta, and J. Li, "Chunkstash: Speeding up inline storage deduplication using flash memory." in *USENIX Annual Technical Conference*, 2010, pp. 1–16.

[13] Y. Fu, H. Jiang, N. Xiao, L. Tian, and F. Liu, "Aa-dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*. IEEE, 2011, pp. 112–120.

[14] Y.-K. Li, M. Xu, C.-H. Ng, and P. P. Lee, "Efficient hybrid inline and out-of-line deduplication for backup storage," *ACM Transactions on Storage (TOS)*, vol. 11, no. 1, p. 2, 2015.

[15] Z. Sun, G. Kuenning, S. Mandal, P. Shilane, V. Tarasov, N. Xiao, and E. Zadok, "A long-term user-centric analysis of deduplication patterns," in *International Conference on Massive Storage Systems and Technology (MSST)*, 2016.

[16] Y. Nam, G. Lu, N. Park, W. Xiao, and D. H. Du, "Chunk fragmentation level: An effective indicator for read performance degradation in deduplication storage," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*. IEEE, 2011, pp. 581–586.

[17] Y. Fu, H. Jiang, N. Xiao, L. Tian, F. Liu, and L. Xu, "Application-aware local-global source deduplication for cloud backup services of personal storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 25, pp. 1155–1165, 2014.

[18] D. Harnik, E. Khaitzin, and D. Sotnikov, "Estimating unseen deduplication-from theory to practice." in *FAST*, 2016, pp. 277–290.

[19] F. Xie, M. Condict, and S. Shete, "Estimating duplication by content-based sampling." in *USENIX Annual Technical Conference*, 2013, pp. 181–186.

[20] 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*. IEEE, 2012, pp. 1–11.

[21] G. Valiant and P. Valiant, "Estimating the unseen: an n/log (n)-sample estimator for entropy and support size, shown optimal via new clts," in *The Forty-Third Annual ACM Symposium on Theory of computing*, 2011.

[22] P. Valiant and G. Valiant, "Estimating the unseen: improved estimators for entropy and other properties," in *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[23] "Trace and snapshot archive," http://tracer.filesystems.org/, accessed: 2017-09-30.

[24] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, Y. Zhang, and Y. Tan, "Design tradeoffs for data deduplication performance in backup workloads," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*. Santa Clara, CA: USENIX Association, 2015, pp. 331–344. [Online]. Available: https://www.usenix.org/conference/fast15/technical-sessions/presentation/fu

[25] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, and Y. Zhou, "Ddelta: A deduplication-inspired fast delta compression approach," *Performance Evaluation*, vol. 79, pp. 258–272, 2014.

[26] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams." in *Fast*, 2010, pp. 239–252.

[27] B. Zhou and J. Wen, "Hysteresis re-chunking based metadata harnessing deduplication of disk images," in *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 2013, pp. 389–398.

[28] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving duplicate elimination in storage systems," *ACM Transactions on Storage (TOS)*, vol. 2, no. 4, pp. 424–448, 2006.

[29] "About deduplication chunk size," http://sort.symantec.com/public/documents/vif/7.0/aix/productguides/html/sf_admin/ch29s01s01.htm, accessed: 2017-09-30.

[30] "Deduplication building block guide," http://documentation.commvault.com/commvault/v10/article?p=features/deduplication/deduplication_building_block.htm, accessed: 2017-09-30.

[31] "Veeam backup & replication 9.5," https://helpcenter.veeam.com/docs/backup/hyperv/compression_deduplication.html?ver=95, accessed: 2017-09-30.

[32] Y. Tan, H. Jiang, D. Feng, L. Tian, Z. Yan, and G. Zhou, "SAM: A semantic-aware multi-tiered source de-duplication framework for cloud backup," in *The 39th International Conference on Parallel Processing (ICPP)*, 2010.

[33] F. Douglis, A. Duggal, P. Shilane, T. Wong, S. Yan, and F. C. Botelho, "The logic of physical garbage collection in deduplicating storage." in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017, pp. 29–44.

[34] H. Wu, C. Wang, Y. Fu, S. Sakr, K. Lu, and L. Zhu, "A differentiated caching mechanism to enable primary storage deduplication in clouds," *IEEE Transactions on Parallel and Distributed Systems*, 2018.