ACM Transactions on Storage 2018

Cluster and Single-Node Analysis of Long-Term Deduplication Patterns

ZHEN "JASON" SUN, National University of Defense Technology
GEOFF KUENNING, Harvey Mudd College
SONAM MANDAL, Stony Brook University
PHILIP SHILANE, Dell EMC
VASILY TARASOV, IBM Research
NONG XIAO, National University of Defense Technology
EREZ ZADOK, Stony Brook University

Deduplication has become essential in disk-based backup systems, but there have been few long-term studies of backup workloads. Most past studies either were of a small static snapshot or covered only a short period that was not representative of how a backup system evolves over time. For this article, we first collected 21 months of data from a shared user file system; 33 users and over 4,000 snapshots are covered. We then analyzed the data set, examining a variety of essential characteristics across two dimensions: single-node deduplication and cluster deduplication. For single-node deduplication analysis, our primary focus was individual-user data. Despite apparently similar roles and behavior among all of our users, we found significant differences in their deduplication ratios. Moreover, the data that some users share with others had a much higher deduplication ratio than average. For cluster deduplication analysis, we implemented seven published data-routing algorithms and created a detailed comparison of their performance with respect to deduplication ratio, load distribution, and communication overhead. We found that per-file routing achieves a higher deduplication ratio than routing by super-chunk (multiple consecutive chunks), but it also leads to high data skew (imbalance of space usage across nodes). We also found that large chunking sizes are better for cluster deduplication, as they significantly reduce data-routing overhead, while their negative impact on deduplication ratios is small and acceptable. We draw interesting conclusions from both single-node and cluster deduplication analysis, and make recommendations for future deduplication systems design.

ACM Reference format:
Zhen "Jason" Sun, Geoff Kuenning, Sonam Mandal, Philip Shilane, Vasily Tarasov, Nong Xiao, and Erez Zadok. 2018. Cluster and Single-Node Analysis of Long-Term Deduplication Patterns. *ACM Trans. Storage* 14, 2, Article 13 (May 2018), 28 pages.
https://doi.org/10.1145/3183890

1 INTRODUCTION

The explosive growth of data in recent years [38] has made deduplication an important research area. Deduplication systems split data into identifiable pieces ("chunks") and then use hashes to identify and eliminate redundant chunks. This approach has proven highly effective in saving space, especially in backup storage [44].

1553-3077/2018/5-ART13 $$15.00
https://doi.org/10.1145/3183890

Many researchers have analyzed data sets from various environments, such as disk and tape backups [14, 39], primary storage [6, 18, 25, 30], archival systems [17], and HPC centers [29]. By understanding such data sets' characteristics, we can design more efficient storage systems [10, 11, 35]. However, data sets may vary significantly across different environments (e.g., whole-file chunking efficiencies range between 20% and 87% compared to sub-file chunking [6, 29, 39]). As a result, conclusions drawn from only a few data sets cannot be used to guide the design of a practical and efficient deduplication system. Thus, new, large-scale studies using different types of data sets and investigating new metrics are desirable.

High data redundancies in backups [15] make deduplication an integral part of backup systems. The space savings of deduplication made the transition from tape-based to disk-based backup systems possible [44]. However, our understanding of these systems' real-life performance is still poor, because there are few long-term studies of large backup data sets: most prior studies draw conclusions based on the entire data set, whereas we show that studying per-user trends is valuable and produces surprising results.

For many backup storage systems, data sizes are growing rapidly, and although deduplication can reduce the space consumption by a factor of 10× or more, single-node storage systems still gradually fail to meet capacity and throughput needs. Cluster deduplication is an appealing approach to address these challenges by distributing the load and aggregating throughput from multiple nodes. Data-routing algorithms are a key component of cluster deduplication; selecting the best destination node for data assignment has a direct impact on the overall (and per-node) deduplication ratio, load balance, and routing overheads. Many data-routing algorithms have been proposed with various goals, such as design simplicity or maximizing the deduplication ratio.

Because there are significant differences in the characteristics of data sets from various environments, the performance of a particular algorithm may vary significantly depending on the data set. We have classified data-routing algorithms and implemented seven published algorithms that adopt different strategies. We provide a detailed comparative analysis of the performance of each algorithm using key metrics and also draw interesting conclusions.

In this article, we first introduce the data set that we have been collecting and have released publicly, and the tools we developed. Our data set, *Homes*, has a longer duration than previously published studies, which is important for investigating realistic, long-term trends: 21 months of daily snapshots taken over a 2.5-year period. One of our goals was to find out whether previously published findings still hold with our much longer data set.

We then present an analysis of single-node deduplication based on this data set, with sometimes expected and sometimes unexpected results. For example, we found that because of the size of the chunk index itself, smaller chunk sizes are not always better at saving space, as others discovered as well [30, 39]. We also found that whole-file chunking identifies significantly fewer duplicates than sub-file chunking, because larger files tend to dominate space usage and have a small deduplication ratio (defined as the logical storage space divided by the physical storage space after deduplication); this was also reported before [30].

In addition, we studied the data set from the users' point of view. Given that our users were largely similar in their background, behavior, and job function, we expected their storage footprints and deduplication patterns to be similar too. Yet that was not the case. We found, investigated, and explained three surprising new results:

(1) The deduplication ratios of each user's own data set varied significantly, and their sensitivity to chunking size was also different. This suggests that even similar users behave quite differently; this should be taken into account in future deduplication systems.

(2) Deduplication ratios across users ranged widely. Yet, in combination with other information, the deduplication ratio can help us group users together to improve the effectiveness of clustered deduplication systems.
(3) The data that users share with each other had a higher deduplication ratio than average, and the duplicate data shared by different user pairs tended to be fairly similar. This knowledge can benefit the caching and prefetching components of deduplication systems.

Next, we studied the *Homes* dataset by using cluster deduplication techniques. We introduce five important parameters that allow us to classify many cluster deduplication algorithms. We implemented seven published algorithms that cover several combinations of these parameters, and simulated their behavior accurately using our extensive data set. We plan to release the implementations of these algorithms in the future, so that others can run or modify the parameters and thresholds of each algorithm easily. We analyzed the behavior of these seven algorithms in terms of cluster deduplication ratios, load distributions, and communication overheads. We have come to three conclusions:

(1) Using whole file as the routing unit leads to a better deduplication ratio at the cost of poor load balance. In addition, communication overheads increase, because most files in *Homes* are small.
(2) Physical and logical load distribution are not always consistent. Here, the *physical load* means the data that is stored in each node after deduplication; and the *logical load* means the data that is assigned to each node before deduplication. The physical load shows the real space consumption, while the logical load includes the meta-data size (mainly the file recipe) and the number of I/O requests. We show in our results that algorithms that can achieve a good physical load balance may lead to a huge skew in their logical distribution.
(3) For our data set, larger chunking sizes are a preferred choice, considering that they reduce both deduplication and routing overheads, while the drop in effective deduplication ratio remains acceptable.

The rest of this article is organized as follows: Section 2 provides background and related work. In Section 3 we give a brief introduction of the *Homes* data set and the tools used to collect the snapshots. We discuss our single-node-based analysis results in Section 4. Cluster deduplication analysis results are shown in Section 5. Section 6 concludes our contributions.

2 BACKGROUND AND RELATED WORK

Now that deduplication has become essential in backup and at times even in primary storage, researchers have tried to improve its performance, for example by enhancing storage efficiency [16, 41], enabling scaling [1, 4, 5, 8, 12, 22, 36], and resolving bottlenecks [3, 10, 19, 21, 27, 44]. Among these studies, those that analyzed real-world data played an important role because they helped improve design decisions based on data characteristics [6, 18, 29, 31, 39].

2.1 Single-Node-Based Data-set Analysis

Most prior deduplication-based data-set analyses have focused on single node deduplication. A study of Microsoft primary storage [30] collected data from 857 users over four weeks. They found that whole-file chunking works well in their specific environment, and fragmentation is not a serious problem when a defragmenter runs weekly. Follow-on work by Microsoft [6] implemented a scalable deduplication system in Windows Server 2012, based on the findings from the primary deduplication analysis. Wallace *et al.* [39] investigated EMC's Data Domain backup systems, showing that the backup characteristics vary significantly with primary workloads. Backup workloads tend to have a high churn rate, a lot of stream locality, high demand for writing, and high redundancy.

Other studies focused on file analysis [25]; conclusions about file type and size distributions, and about deduplication and compression ratios, have helped improve the design of deduplication systems. Data sets from different storage environments, such as high-performance computing [29] and virtual disk images [18], have also been collected and analyzed, and many of their conclusions are also applicable to other storage systems.

Most of the aforementioned data sets are static snapshots of storage systems or only cover a short time period. Our work extends these studies by using a long-term workload that spans March 2012 to November 2014. Because backup systems are designed for long-term storage, our results—based on a long history—offer new valuable insights.

2.2 Cluster Deduplication

When data sets or I/O requirements grow beyond the capabilities of an individual deduplicated storage server, creating a clustered deduplication solution is a promising and even necessary approach. Due to the large differences in the characteristics of the data sets and application environments, many cluster deduplication schemes have been designed for different situations and purposes. Extreme Binning [1] exploited file similarity to improve the throughput for data sets that lack modification locality. It has proven effective for workloads that contain many duplicated files. DEBAR [42] introduced an effective hash index for parallel fingerprint searching; it also used a memory cache to optimize disk operations, which improved the throughput and scalability of the system. MAD2 [40] used file-level and chunk-level deduplication, and employed a hash-bucket matrix to maintain high data locality, improving throughput and scalability. Both DEBAR and MAD2 adopted a fine-grained routing unit, which might lead to larger communication overhead. HYDRAstor [36] used larger chunking sizes for both deduplication and data routing, which sacrifices some of the deduplication ratio but achieves a lower data-routing overhead compared with using a finer-grained routing unit.

Dong et al. proposed two strategies [4] for different purposes. The *stateless* strategy is designed for implementation simplicity; it uses a *super-chunk* as the routing unit and assigns super-chunks to cluster nodes based on a hash table. It can reduce communication overhead significantly and keep the system load-balanced. The *stateful* strategy also routes data by super-chunks, but it checks the similarity of the super-chunk in all nodes, an approach that achieves a higher deduplication ratio than the stateless one. Sigma-Dedup [12] introduced a hand-print technique, which uses a small group of representative hashes from a super-chunk to make a routing decision, reducing communication overhead. To leverage locality in the data stream, it uses a similarity hash index that contains mappings from hashes in hand-prints to storage containers. By prefetching the whole container into the cache, the cache hit ratio is improved and chunk index lookup is accelerated. Produck [8] used Probabilistic Counting with Stochastic Averaging (PCSA), a technique that can quickly estimate the cardinality of a collection of hashes using bitmaps. Produck uses PCSA to quickly select the best destination node for a super-chunk. Its memory and CPU overheads are small, but to ensure a low estimation error, its super-chunk size is much larger than in other algorithms, which can hurt the deduplication ratio.

Although many cluster deduplication algorithms have been proposed, their performance using different data sets can vary significantly due to differing data-set characteristics and to the differences in the algorithms themselves. In addition, there is little published work that compares several algorithms together using a single data set (let alone a large data set such as ours). In our work, we first classified cluster deduplication algorithms based on their characteristics. We then implemented seven representative algorithms and evaluated them carefully. By showing their performance on key metrics, we draw interesting conclusions for future cluster deduplication system design.

Finally, in a technical report, Eshgi *et al.* propose a modification of Auction-based routing called *Sticky-Auction Routing* [7]. The main problem they address for clustered deduplicated storage is that databases tend to interleave data, which is inconsistent from backup to backup. This happens because a database may choose to read from multiple files in parallel, fill an internal buffer, and then transfer the buffer to backup storage in a streaming fashion. Our standard routing approaches in a cluster assume that large transfer units (called super-chunks) will tend to be consistent from backup to backup. The interleaving pattern from a database, however, can vary from day to day based on changing disk performance. Sticky-Auction routing modifies a stateful Auction-routing algorithm to handle this situation. The authors noted that although interleaving patterns change, each page of the database tends not to move around much in practice because the parallel read streams appear to vary only within a reasonably small range. Their approach is that when Auction fails to find a good node—based on a threshold number of chunk matches—this is likely to be the first full backup; if so, they direct a much larger transfer unit from each node to a lightly loaded node (tens of GBs instead of a typical 8MB transfer size). The authors call their approach Sticky-Auction: it outperforms Auction routing for interleaved data-sets and has similar performance as Auction for non-interleaved data-sets. Load balancing is also similar between Sticky-Auction and Auction when there are multiple streams to process. We did not investigate the Sticky-Auction algorithm because our data set includes users' home directories, which tend to exhibit serial I/Os, as opposed to database-like random ones. Still, the idea of detecting an initial full backup might be appealing, and we hope to investigate this option in future work.

3 METHODOLOGY

We now describe the tools we developed, the data set itself, and the limitations of this study. Both the tools and the data set have been publicly available online [9]; the data set is updated periodically and released as we continue to collect snapshots. *Homes* has already been used in a number of research papers [20, 23, 24, 33, 35, 37, 43].

3.1 Tools

To conduct our analysis, we developed tools that collect and parse file-system snapshots. *Fs-hasher* scans a file system and collects snapshots that contain both file system data and rich meta data. It does not collect actual file content; instead, it chunks each file and collects the hashes, similar to real-world deduplication systems. *Fs-hasher* supports both fixed and variable chunking. Users can specify various parameters (e.g., expected average chunk size for variable chunking) and hash functions (e.g., MURMUR, MD5, and SHA1) when collecting snapshots. Hash files collected by *Fs-hasher* can be parsed by a purpose-written program (linked against the provided *libhash* library) or by *Hf-stat*, which prints hash files in a human-readable and post-processable format. *Hf-stat* provides options to control and filter its output. *Fs-hasher* does not anonymize the snapshots; however, for privacy protection, we anonymized the released data set during post-processing. Our tools are similar to `fs-c` [26, 28] and were developed independently around the same time. Whereas `fs-c` was written in Java, our tools were written C with scalability and efficiency in mind. We have released and continue to maintain our tools on our Web site (http://tracer.filesystems.org).

3.2 Data set

The *Homes* data set contains daily snapshots of our users' home directories on a shared file system; we collected one daily snapshot per user for a total of over 4,000 snapshots. The users are all Linux systems software developers, mainly graduate students and faculty working in a Unix/Linux environment, who work on several joint projects; we believe this data set is reasonably representative of similar software development environments. The snapshots have been collected from March 2012 to April 2015 with periodic releases of updated data sets. The data presented in this article includes snapshots up to November 2014 because we had to start analyzing our data at some points, even though we continued to collect snapshots. (We sampled the newer snapshots and do not believe that they alter this article's conclusions substantially.) We used variable chunking with 7 average chunk sizes (powers of two ranging from 2KB to 128KB) and whole-file chunking (WFC); this allows us to conduct a detailed comparison of performance and overhead among common chunking methods. To speed the snapshot collection procedure and reduce the data-set size, we chose a 48-bit MD5-based hash function. Although we used a short hash, our collision rate is still acceptable for research purposes [39], because even with 2KB chunking, the number of unique hashes is about 10^8, so the expected number of collisions in *Homes* is only about 4,000 (or 0.004%). Although this number would be unacceptable in commercial deduplication systems, it has a negligible impact on our findings. For larger chunk sizes, our collision rate was even lower.

Previous data sets have used static snapshots or have covered a short time period. *Homes* contains over 21 months of daily snapshots taken from a nearly 3-year period. It includes data on 33 active users, allowing us to analyze deduplication on a per-user basis. Although we did not collect the full file content, the rich meta-data and extensive hashes make it possible to conduct a wide range of studies. For example, by examining modification and access times, we can simulate various backup strategies, including full and incremental backups at different periods. Table 1 shows the details of the *Homes* data set.

Table 1. Features of the Homes data set.

Data set	Homes
Total size	456TB
Start and end time	03/09/2012–11/23/2014
Number of users	33
Number of snapshots	4,181 dailies (about 21 months)
Chunking method	Content-defined chunking (CDC), Whole-file chunking (WFC)
Average chunking size	2, 4, 8, 16, 32, 64, and 128KB
Hash function	48-bit MD5
Number of files	1.3×10^9
Number of logical chunks	1.9×10^{11} (2KB chunk size) 4.0×10^9 (128KB chunk size)
Number of unique chunks	9.8×10^8 (2KB chunk size) 3.3×10^7 (128KB chunk size)
Meta-data included	File pathname, size, atime, ctime, mtime, UID, GID, permission bits, device ID, inode number

3.3 Limitations

Since storing full file contents would consume too much space, we recorded only hashes and meta-data. Thus, we are unable to analyze content-based properties, such as compression performance. For other properties like backup throughput, we can calculate simulated results from other metrics, such as the cache miss ratio and average disk I/O latency.

As we discussed in Section 3.2, a 48-bit MD5 hash is long enough to generate an acceptable collision ratio for research purposes. However, it may become a limitation in some special circumstances when longer hash values are needed (e.g., the PCSA technique in Section 5.1 needs a longer hash to insure its accuracy).

Although we attempted to collect daily snapshots for 3 years, some periods were missed, mainly due to major power outages (e.g., severe weather events), hardware failures, and long breaks when most data remain unchanged due to user absence. Still, we believe that our data set is sufficiently large, long-term, and continuous to serve as the basis for a valuable study.

4 SINGLE-NODE-BASED ANALYSIS

We begin by describing results from analyzing the entire data set as a whole, without assuming it would be divided among nodes in a storage cluster. Section 4.1 describes our deduplication-ratio analysis and Section 4.2 describes our file-based analysis. Then, in Section 4.3, we present the results of the user-centric analysis.

4.1 Deduplication Ratios

One of the key measures of a deduplication system is the *deduplication ratio*, defined as the size of original data set divided by the size of what is physically stored on the media. In *Homes*, the variety of meta-data that we collected makes it possible to simulate a number of realistic backup strategies. For this article we chose four typical strategies: (1) *Full*, (2) *Incremental*, (3) *Weekly-Full* (a full backup each Saturday and incrementals for the rest of the week) and (4) *Fixed Retention Period* (snapshots will be deleted after a fixed number of days). Since our full snapshots were collected daily, they inherently represent full backups. For incremental and weekly-full backups we needed to detect newly added and modified files. By comparing two consecutive snapshots, we could identify whether a file was newly added. By checking the *mtime*, we determined which files were modified since the previous snapshot. In the first three methods listed above, backups are accumulated daily and never deleted. But in many deployed systems, backups are deleted periodically to cap storage costs [13]. This led us to simulate the fixed-retention policy, in which expired daily-full backups are deleted after a fixed period of time (e.g., 7 days). Table 2 shows the deduplication ratios we observed when using different backup strategies and chunk sizes.

Table 2. Average raw deduplication ratios for various chunking methods and backup strategies. WFC stands for Whole-File chunking.

Chunk size	Full backup	Incremental backup	Weekly-full backup	20-Day Retention
2KB	**218.5**	**13.6**	**42.8**	**41.5**
4KB	197.0	12.6	39.4	39.6
8KB	181.9	11.7	36.5	38.0
16KB	167.4	10.7	33.6	36.3
32KB	153.3	9.8	30.8	34.8
64KB	139.1	8.9	27.9	33.2
128KB	128.0	8.2	25.7	31.6
WFC	16.4	1.1	2.3	2.7

In Table 2, the results for the retention backup policy are averages, since deduplication ratios change dynamically as we store new snapshots and remove expired ones. Figure 1 shows how deduplication ratios change with time, given different retention periods and 128KB chunking; we found similar results for all other chunking sizes. In most cases, longer retention periods achieve higher deduplication ratios, but in rare circumstances storing more snapshots can produce *less* deduplication. We can also see that the deduplication ratio varies over time inside a given data set. Since the deduplication ratio does not directly reflect the space usage in each storage node. Figures 2 and 3 show the logical and the physical space consumption, respectively. The results are similar: both logical and physical consumption change over time. The main reason for this phenomenon is the large difference in the characteristics among each user's data, which we discuss in Section 4.3. The characteristics of *Homes* vary significantly as users join and leave.

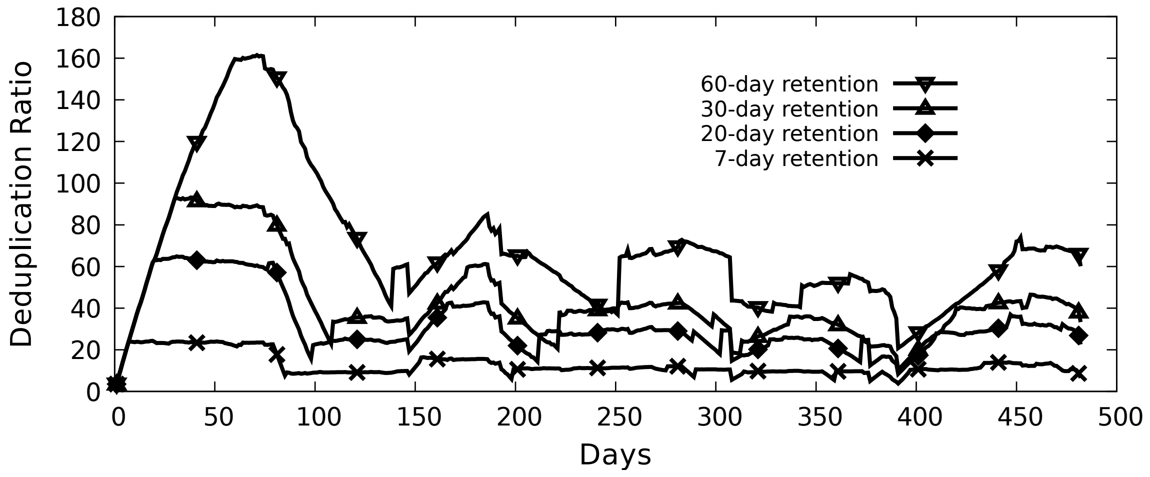


Fig. 1. Deduplication ratio using fixed retention backup policy.

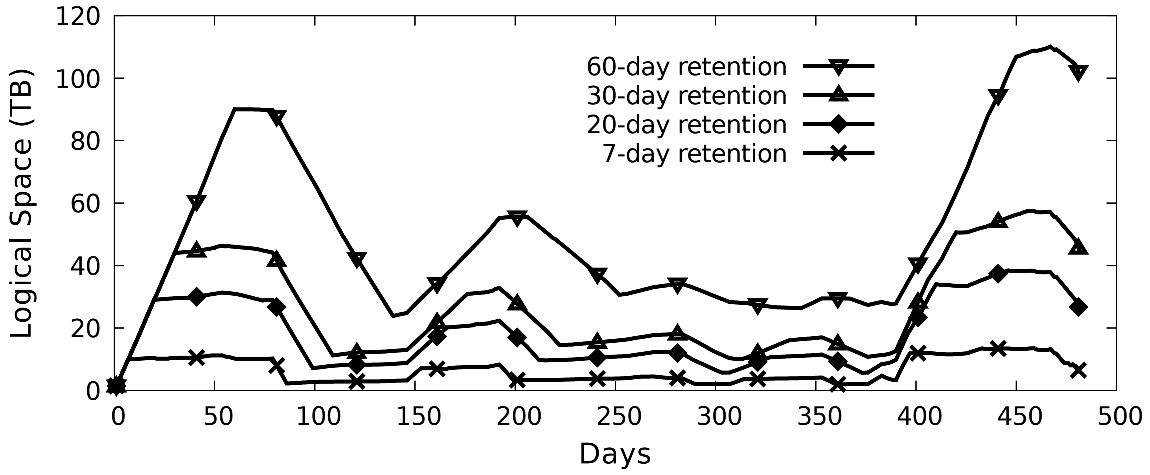


Fig. 2. Logical space consumption using fixed retention backup policy.

The numbers shown in Table 2 are raw deduplication ratios. However, a smaller chunk size implies higher meta-data overhead, since more hashes must be stored. To calculate the efficiency

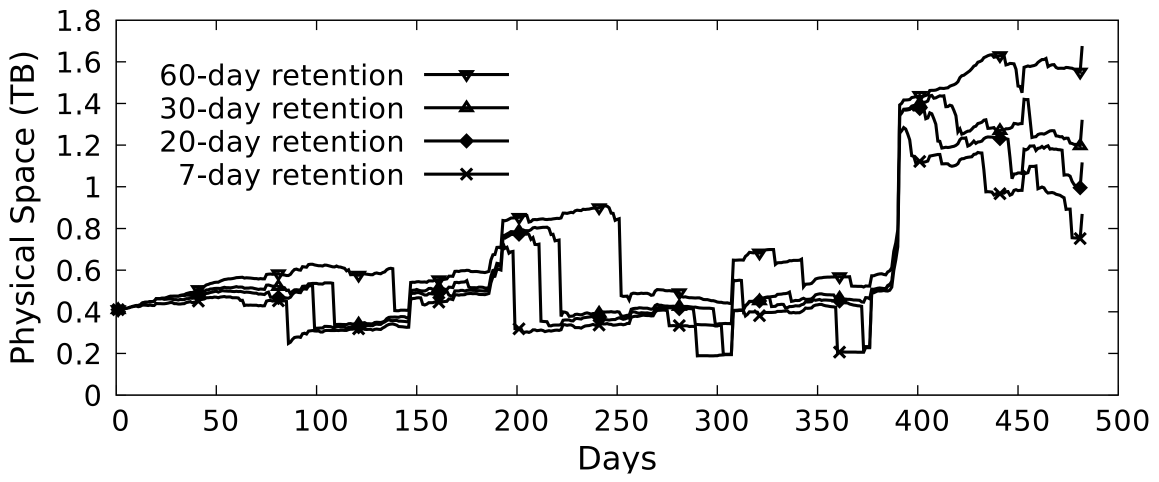


Fig. 3. Physical space consumption using fixed retention backup policy.

Table 3. Effective deduplication ratios after accounting for meta-data overheads.

Chunk size	Full backup	Incremental backup	Weekly-full backup	20-day Retention
2KB	50.9	11.1	25.8	25.6
4KB	79.3	**11.4**	30.2	30.5
8KB	107.9	11.1	**32.0**	33.3
16KB	127.2	10.5	31.6	**34.0**
32KB	**133.9**	9.7	29.9	33.7
64KB	130.5	8.9	27.6	32.7
128KB	124.3	8.2	25.7	31.4

of various chunking sizes, we adopted the approach proposed by Wallace *et al.* [39]. Suppose L is the size before deduplication, P is the raw data size afterwards, and f is the size of each chunk's meta-data divided by the chunk size (i.e., the fraction of meta-data overhead). Then the raw deduplication ratio is $D = \frac{L}{P}$. The meta-data size is $f \times (L + P)$; $f \times L$ is the size of a file's *recipe* (needed to reconstruct its original contents) and $f \times P$ is the size of the hash index. Thus the overall stored size is $P + f \times (L + P)$. Based on this formula, the effective deduplication ratio including all costs, D', is:

$$D' = \frac{L}{P + f \times (L + P)} = \frac{D}{1 + f \times (D + 1)} \tag{1}$$

Although our snapshots used a 6-byte (48-bit) MD5-based hash function, in this analysis we assume 30 bytes per chunk to show what would happen in a real deduplication system that stores longer hashes and other information such as chunk pointers and file recipes. This value is chosen from the middle of a range of common meta-data sizes [28, 39].

Table 3 shows the effective deduplication ratio for each strategy. We can see that 32KB chunking performed best for full backups; 4KB for incrementals; 8KB for weekly-full ones; and 16KB for the 20-day retention strategy. This suggests that the best chunk size may depend on the backup strategy and frequency. Table 3 also shows that as chunk sizes increase, a decrease in deduplication ratio is not guaranteed. With higher deduplication ratios, meta-data can become a large fraction of post-deduplication space, so larger chunk sizes reduce the number of meta-data entries significantly enough to compensate for missing some duplicates. Moreover, reducing the amount of meta-data also reduces the number of I/Os to the chunk index, so larger chunk sizes can offer benefits beyond mere storage savings.

4.2 File-Based Analysis

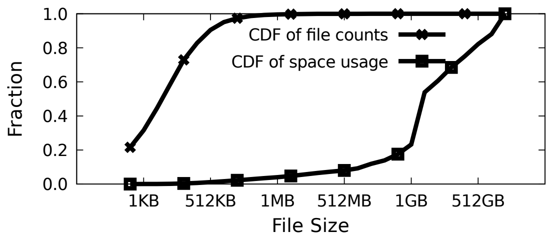


Fig. 4. Distribution of file counts and their space usage.

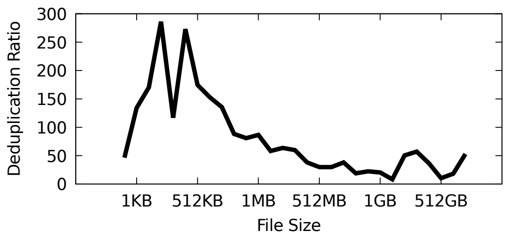


Fig. 5. Whole-file chunking deduplication ratio of files of different sizes.

The performance of whole-file chunking varies widely based on the characteristics of different data sets [30, 39], but it is clearly not a good choice in *Homes*, as shown in Table 2. The reason can be found in Figures 4 and 5. Figure 4 shows the distribution of file sizes as a fraction of total file count and their space usage. The upper line of Figure 4 shows that more than 99% of the files are smaller than 1MB, but the lower line demonstrates that in aggregate, these small files consume less than 4% of total space. We observed large variations when we grouped files by size and calculated a separate whole-file deduplication ratio for each group (see Figure 5). Files between 2KB and 256KB deduplicate well—all are better than 100× and the best is about 290×. In contrast, the average deduplication ratio for files larger than 1MB is less than 50×. Thus we can conclude that the total size of *Homes* is dominated by large files that have a low WFC deduplication ratio. As a result, whole-file chunking is ineffective at saving space in this situation.

Figure 6 shows the percentage of total space occupied by various common file types; we can see that virtual-machine images (vmdk files) account for nearly 60% of all data. Figure 7 shows the raw

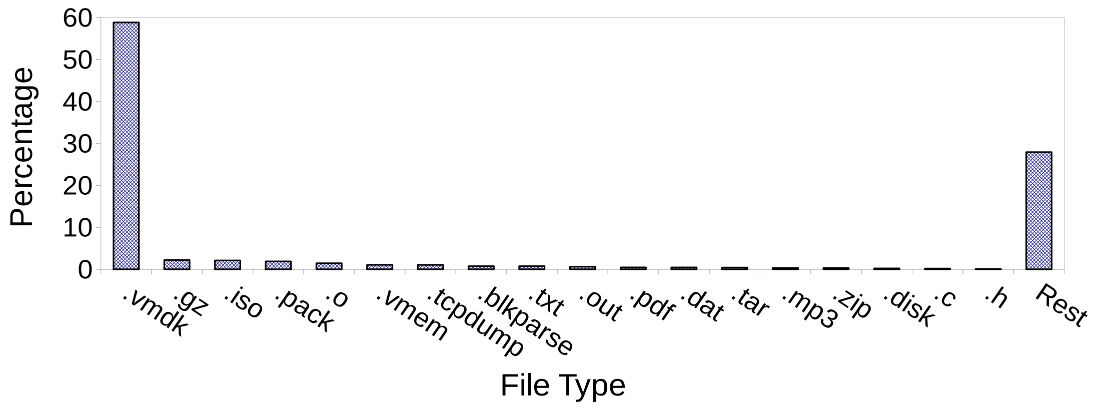


Fig. 6. Distribution of total storage by file type before deduplication. The *Rest* bar represents the file types that independently contributed less than 0.4% of total storage each.

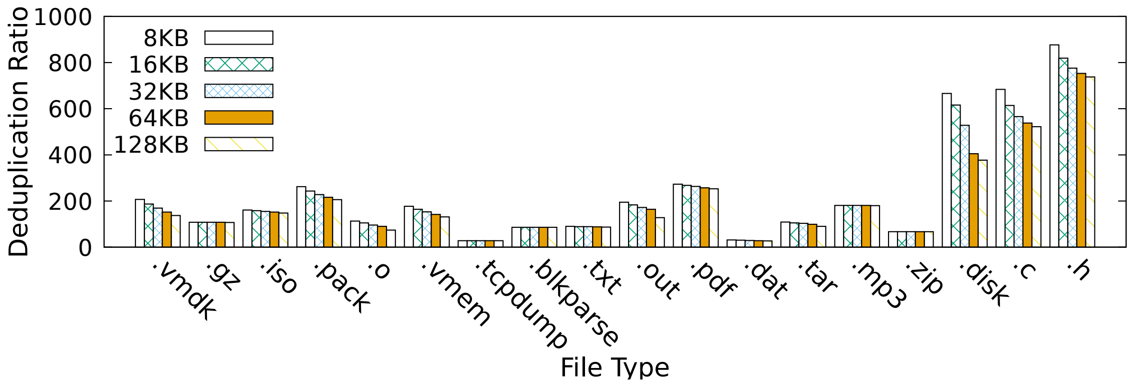


Fig. 7. Deduplication ratio of different file types at different chunking sizes.

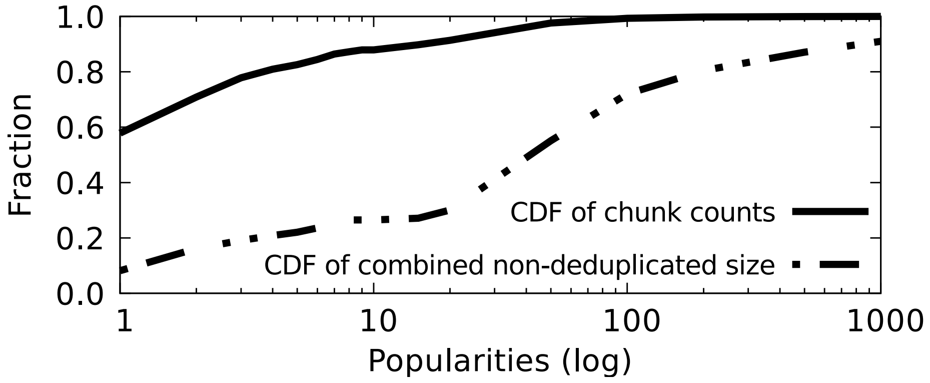


Fig. 8. Chunk popularities and their sizes. Because of a few extremely popular chunks, the lower CDF doesn't reach 100% until more than 0.3 billion occurences.

deduplication ratio of different file types at various chunking sizes; here we selected the file types that occupied the most disk space. We can see that the deduplication ratio of different file types varies significantly: using 8KB chunking, the ratio ranges from 50 (`.vdi`) to 877 (`.h`). In addition, file types also have different sensitivities to the chunking size. For some types, increasing the chunks from 8KB to 128KB leads to a large drop in deduplication ratio: e.g., a 43% loss (`.disk`), 34% (`.vmdk`), and 26% (`.vmem`), respectively. However, for most file types this decrease is not significant, with some types showing no reduction at all (e.g., `.gz`, `.mp3`, and `.zip`). The main reason for this phenomenon is that most of these file types store compressed data. In a software-development environment, these files are rarely changed. Furthermore, changing even a single byte in an uncompressed file causes the whole compressed version to contain completely different data. As a result, the whole compressed file is completely different from its original version for any chunk size. Therefore, the deduplication ratio is independent of the chunk size for these file types.

We define *chunk popularity* as the number of duplicate occurrences of a chunk. In Figure 8 we present the cumulative chunk popularity distribution for incremental backup with a 4KB chunk size. (We also evaluated all other chunk sizes and did not find any significant difference from what we present here.) The upper curve shows that about 60% of all chunk hashes appear only once in the data set, and according to the lower curve these chunks consume less than 10% of the entire

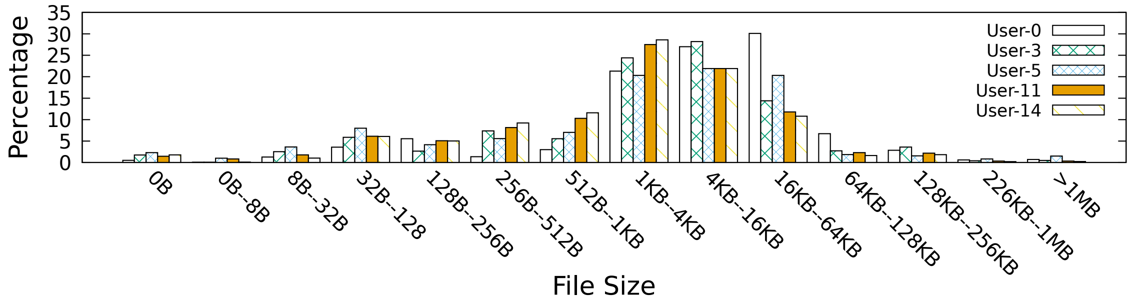


Fig. 9. Distribution of counts of different file types for a few users.

non-deduplicated data size. In contrast, chunks that appear at least 20 times take more than 70% of the total non-deduplicated space (lower curve, center to right-hand side) but account for fewer than 9% of all chunks (upper curve, right-hand portion). Chunks that appear at least 100 times are fairly important: they take less than 0.6% of the unique chunks but occupy nearly 30% of the whole space.

This skew in chunk popularity has also been found in primary storage [6, 25] and HPC systems [29]; those researchers found that chunks that appear 2–32 times contribute the most to the deduplication ratio in primary storage, while in *Homes*, the popularity is much higher. Identifying such popular chunks would be useful in optimizing performance. For example, keeping hot chunks in memory could accelerate chunk indexing and improve cache hit ratios.

We also found that most of the shared data among different users belongs to these popular chunks, a phenomenon that we discuss in Section 4.3.2.

To study the distribution of file sizes, we chose several representative users (Figure 9). This figure shows that for those users, most files are between 1KB and 64KB. However, the average file size in our data set is 366KB due to large VMDK files. Without VMDK files, the average file size is 151KB. Our average file size is smaller than has been reported in backup workloads [39] because backup software often combines many smaller files into larger `tar` files.

4.3 User-Based Analysis

Past studies have often focused on whole data sets and did not study the data from the users' perspective. Although each user's data forms a basic unit that is part of the data set, per-user data can have its own special characteristics. In this section, we show some interesting results from our 33 users with non-empty data. We have studied our data carefully, both individually and in various groupings. We present a representative sample of these individuals and groups, carefully chosen to highlight key findings seen repeatedly. For example, when studying each user's deduplication ratio, we selected accounts that covered different characteristics, such as the total size or lifetime of the user's data. To show how deduplication ratios change over time, we selected users who have matching start times.

4.3.1 Per-User Analysis. Due to users joining and leaving our system, their snapshots have varying start and end times. When users leave, we keep all their previous snapshots but stop collecting new ones. The duration of each user's observations varied from 1–20 months. The data-set sizes are also different; the largest user's data (62TB) is about three orders of magnitude larger than the smallest (11GB). The deduplication ratio for each user at different chunk sizes is shown in Figure 10. In the *Homes* data-set, we anonymized user names to protect their privacy; each user was given a number rather than a name. We have 39 users (0–38) in our data-set. However,

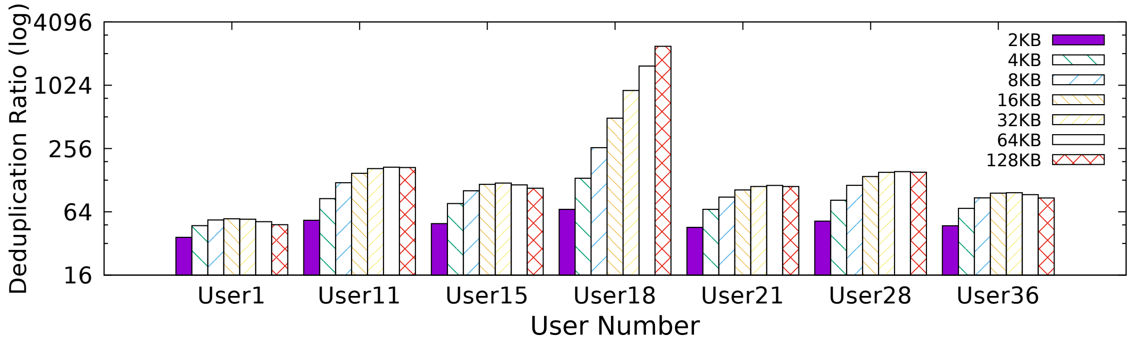


Fig. 10. Effective deduplication ratios of seven users at different chunking sizes, considering meta-data overheads and using the Full backup method.

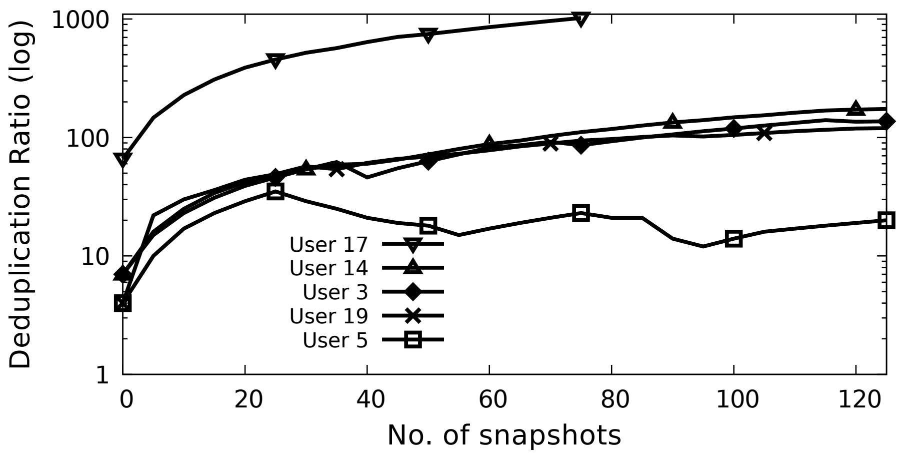


Fig. 11. Users' deduplication ratios (log) vs. number of snapshots in a backup.

the data for 6 users was empty (i.e., they received accounts but either never created any files or had their accounts deleted), so for this article we analyzed a total of 33 users.

For Figure 10 we chose 7 representative users based on characteristics such as size and duration; user 15 has both the longest duration (20 months) and the largest data-set size (62TB). The figure shows large differences among users' effective deduplication ratios (considering meta-data overhead). Using 128KB chunking, the highest deduplication ratio is over 2,400 (uncommon for most users) while the lowest is less than 40. Moreover, each user's sensitivity to chunk size also varies significantly. For Users 18 and 21, the raw deduplication ratio is so large that meta-data takes a large fraction of the space after deduplication; as a result, their effective deduplication ratio at 2KB chunking is only about 33% of that achieved by 128KB chunks. But for User 1, the 2KB-chunking deduplication ratio is about 76% of the 128KB one. The best chunking size for each user is also different: User 1 does well at 16KB, User 18 at 128KB, and several others are in between. User 18 is special, as we can see a steady increase in deduplication ratio as the chunking size increases. This is because several large and highly redundant benchmarking data files represent over 98% of this user's total disk usage. This user's raw deduplication ratio is over 5,000; thus the meta-data takes a large fraction of the total post-deduplication space.

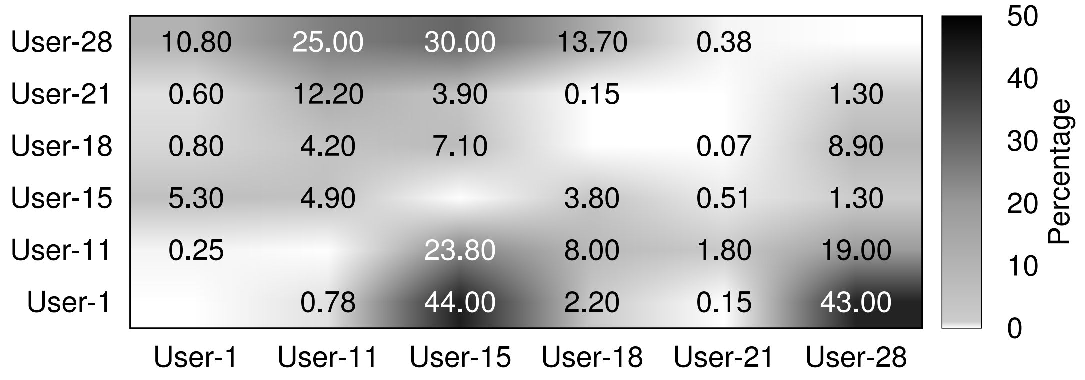


Fig. 12. Data redundancies among users.

One reasonable explanation for the difference in deduplication ratios is the varying user lifetimes in the system. For full backups, a user's deduplication ratio will probably increase over time as more snapshots are added. However, we found that the large differences in deduplication ratios among users in *Homes* were independent of their lifetimes. Figure 11 shows how ratios changed as snapshots accumulated. The snapshots of these five users had the same start times. But as we can see, User 17's deduplication ratio increased rapidly from the start. Although User 17 had fewer snapshots, the final deduplication ratio was over 1,000, far greater than the others, primarily due to high internal redundancy within a single snapshot. Conversely, User 5's deduplication ratio *dropped* over time, mainly because a high file churn rate led to the addition of many new chunks. From Figure 11, we can see that the number of snapshots is not the sole factor affecting a user's overall deduplication ratio. We also found that the file type is not the main reason for differences in deduplication ratios. Even users with similar file type distributions (e.g., heavy users of VMDK files) also varied significantly in deduplication ratio, while some users with similar deduplication ratios had different file types. The important factors seems to be the characteristics of the users' own data, such as internal redundancy, and the user's activity level. Overall, we conclude that "not all users are created equal," even when they are performing similar jobs. Therefore, future deduplication systems should account for such behavior to improve efficiency.

4.3.2 Analysis of Groups of Users. We now turn to cross-user redundancies. We used a representative sample of users, shown in Figure 12. Each number in the heat map is the percentage of a user's data that is also found in another's. For example, for User 1, more than 40% of that user's data can be found in User 15's data (bottom row, third column), and the same is true for User 28's (though precisely what is shared might differ between the two). This figure is not symmetric because each user's own data size is different. For example, the third row, first column shows that only 5.3% of User 15's data is shared with User 1 because User 15 has much more data overall. The figure shows that for each user, redundancies with others varied significantly. We confirmed these results for all users and found no obvious pattern.

While studying inter-user duplication, we found that users can be divided into groups in which the shared data of any two members occupies a large fraction of their total space. To analyze the characteristics of the shared data, we selected four representative users (13, 15, 19, and 28) as a group. Here we define a user's unique data set as S; $|S|$ means the number of chunks in S; and the data shared between users X and Y is $S_X \wedge S_Y$. Our results showed that 98% of the chunks shared

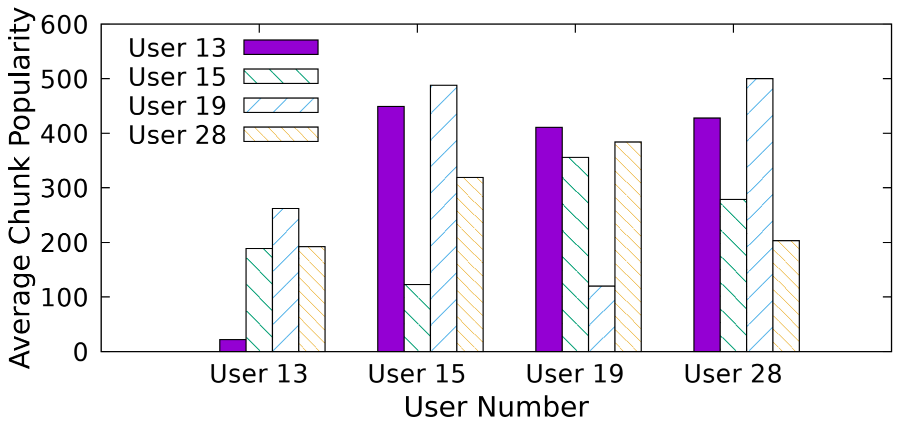


Fig. 13. The popularity of users' shared data. From this figure we can see that the shared chunks are much more popular than the average inner chunk popularity for each user.

between Users 13 and 19 could also be found in the shared data between Users 13 and 15. Stated mathematically:

$$\frac{|(S_{13} \wedge S_{19}) \wedge (S_{13} \wedge S_{15})|}{min(|S_{13} \wedge S_{19}|, |S_{13} \wedge S_{15}|)} = 0.98 \tag{2}$$

We checked this number for the other combinations in this 4-user virtual group, and the results were always between 91–98%. Thus, the data shared by the users in this group is fairly similar, so in a cluster deduplication system, grouping them into one storage node would improve the overall deduplication ratio.

Lastly, we found that chunks shared among users tended to have much higher popularity than average. Suppose B is the number of chunks before deduplication, and A is the number of chunks after deduplication. Then the average chunk popularity of user X is $P_x = \frac{B_x}{A_x}$. Based on this formula, the average popularity of shared chunks between users X and Y is:

$$P_{X \wedge Y} = \frac{B_{X \wedge Y}}{A_{X \wedge Y}} \tag{3}$$

Note here that $A_{X \wedge Y} = A_{Y \wedge X}$, but $B_{X \wedge Y} \neq B_{Y \wedge X}$. This is because although the unique hash values they share are the same, each user has their own inner deduplication ratio. The popularity of shared chunks is shown in Figure 13. For User 13, for example, the average chunk popularity (P_{13} or $P_{13 \wedge 13}$), was 22 (shown as the leftmost bar), while the average popularities of chunks shared with Users 15 ($P_{13 \wedge 15}$), 19 ($P_{13 \wedge 19}$), and 28 ($P_{13 \wedge 28}$) were 189, 262, and 192, respectively. This means there was a 8.6–11.9× increase in these shared chunks' popularity compared with the average popularity of all chunks in User 13's data. Our results for other users and groups supported this conclusion. Another way to interpret this result is that when a given chunk is shared between two or more users, it is also more likely to appear several times in any single user's data, and thus the deduplication ratio for shared data is higher even for a single user.

5 ANALYSIS OF CLUSTER DEDUPLICATION

As data size continues to grow, especially in backup systems, a single-node storage system will gradually fail to meet performance requirements such as capacity and throughput. One effective solution is to use a clustered system. There are two main challenges when applying deduplication techniques in a cluster storage system.

(1) **Maintaining high deduplication ratio**. In most cluster systems, deduplication is performed individually at each storage node and no information about further duplicates is exchanged among them. As a result, duplicate chunks may exist in different nodes and the deduplication ratio of the entire system drops.
(2) **Balancing the load across cluster nodes**. There is a tradeoff between load balance and deduplication ratio in cluster deduplication system. At one extreme, placing all data on one node will maximize deduplication while having the worst load balance. At the other extreme, placing data with a round-robin approach will lead to balanced usage but duplicates will be spread across nodes, reducing potential deduplication. As a result, maximizing deduplication while achieving a balanced load is a a major challenge when designing a cluster storage system.

In this section, we first describe seven classic cluster-deduplication algorithms that we reimplemented and studied. In Section 5.1 we classify them based on their characteristics. Their performance using a number of key metrics is discussed in Section 5.2. In Section 5.3, we summarize the analysis results.

5.1 Classification of Cluster Deduplication Algorithms

In cluster deduplication systems, the algorithm for routing data to a particular storage node plays a key role. It has a direct influence on performance metrics such as the deduplication ratio and load balance. Because there is no "silver bullet" that can meet both of the challenges mentioned above, many data-routing algorithms have been proposed to meet different requirements. Based on the characteristics of a particular data set and storage environment, each algorithm uses different design principles to trade off among the system's deduplication ratio, load distribution, and throughput. We begin with a detailed description of these design choices and their impact.

- *Stateless vs. Stateful:* By recording the assignment of previous chunks, *stateful* algorithms can route similar chunks to the same node. Thus they can get a higher deduplication ratio, but maintaining all previous routing information and searching for the best destination node leads to higher RAM and CPU cost. *Stateless* algorithms assign chunks by considering only the content of each chunk, such as using a hash of the content to determine assignment with a simple function. They normally can route chunks faster while achieving a reasonably balanced load distribution due to their randomized nature.
- *Centralized vs. Distributed:* In a *Centralized* system a *master node* is in charge of data-routing; information on new chunks is sent to the master node to be assigned. Because the master node maintains the system's meta-data (such as file recipes), centralized algorithms can make full use of this information to optimize its data routing algorithm. However, the master node itself may become the system bottleneck. In *distributed* algorithms, clients can make routing decisions on their own. This can reduce or eliminate the burden on the master node and reduce network overheads by not having to send chunk information to a central server all the time.
- *Routing Unit:* Many cluster deduplication systems assign data using a *super-chunk*, which is composed of a number of regular chunks. Comparing with routing using fine-grained units such as individual (small) files or chunks, routing by super-chunks has several advantages. It can significantly reduce routing overhead while preserving chunk locality, which is useful for cache optimization and file restore performance. However, using a coarse-grained

routing unit may hurt the deduplication ratio since even the best assignment of a super-chunk to a node may result in duplicate chunks across nodes.

- *Deduplication Unit:* In most cluster deduplication systems, the *deduplication unit* is made much smaller than *routing unit* so that the system can achieve a higher deduplication ratio. Although in Section 4.1 we found that smaller chunks do not always improve deduplication ratios, most cluster deduplication systems use small chunks as the deduplication unit. This approach is necessary because super-chunks are normally larger than 1MB, which would result in poor deduplication ratios. The deduplication unit is necessarily smaller than or equal to the routing unit; a larger routing unit will reduce the data routing overhead and improve data locality (by grouping related chunks together). Both the deduplication unit and the routing unit impact the deduplication ratio in cluster system.
- *Exact vs. Approximate Deduplication: Exact* deduplication systems can detect and remove all redundant chunks in a data set, and thus their deduplication ratio is as high as that of a single-node system. Chunk-level routing is required to achieve exact deduplication, resulting in more CPU and memory consumption. Conversely, approximate deduplication systems focus on improving other metrics, such as the throughput, at the cost of a decrease in deduplication ratios.

Unlike previously studied data sets, *Homes* covers a long usage history and comprises many user-based snapshots. Therefore, we investigated how different cluster deduplication strategies perform on this unique data set. We implemented seven representative cluster algorithms, which cover a wide range of design principles and configurations. For uniformity of comparisons, in all the implementations, we used a 4KB chunking size, except for HYDRAstor [36], which used 64KB chunking as described in the HYDRAstor paper. Here we describe the implementation of each algorithm.

(1) *Stateless [4]:* The main advantage of *Stateless* algorithms is simplicity. The routing unit is a super-chunk; a *featured* chunk is selected as a representative by choosing the smallest hash value. The featured hash value is looked up in a hash table to assign the super-chunk to a node. In our implementation, a super-chunk contains 256 chunks: we chose this value so that the average super-chunk size (about 1MB) matches one that was published previously in the original paper [4]. Because we are using Content-Defined Chunking (CDC), the size of the super-chunk is variable; this also applies to all other algorithms that route data by super-chunk.

(2) *Extreme Binning [1]:* For workloads that lack locality, *Extreme Binning* is designed to improve throughput by exploiting file similarity; the method is similar to *stateless* when using an entire-file as the super-chunk. The file is the routing unit. For each file, *Extreme Binning* selects the minimum chunk hash as the file's representative chunk ID, and assigns files to different bins based on those IDs. By comparing whole-file hashes, duplicate files can be removed directly. Since similar files are routed to the same bins, *Extreme Binning* has proven effective in deduplicating files.

(3) *Based on File Type:* Since duplicates mainly occur among the files of the same type, routing files based on their type is a straightforward way to improve deduplication ratios. In our implementation, we used an index to record the destination nodes for all files of the same type. When a new file type arrives, it will be assigned to the least-loaded node to achieve balance.

(4) *HYDRAstor [36]:* HYDRAstor is a variant of a stateless routing algorithm. It uses a larger chunk size of 64KB for both routing and deduplication to trade routing overhead off against the deduplication ratio. Because a 64KB chunk size was used for the HYDRAstor paper, we

used the same value for our implementation. This is in contrast to the 4KB size we chose for all other algorithms.

(5) *Stateful* [4]: *Stateful* uses one Bloom filter per storage node, to record all chunk hashes assigned to that node. These Bloom filters can be kept either in the storage node or together in the master; in our implementation they reside in the storage node. When assigning new super-chunks, *Stateful* searches each Bloom filter to find how many chunks from this super-chunk are currently on the node, and then selects a destination considering both the load and the amount of similarity. The filters consume 500MB of memory in each node, except for our simulation of a 128-node cluster, where we used 128MB Bloom filters to limit the total size of the simulation. In all cases, the Bloom filters produced a false-positive rate of 0.2% or less.

(6) *Sigma-dedup* [12]: *Sigma-dedup* uses a *hand-print* technique to reduce the lookup and network overhead in other stateful routing algorithms. A hand-print is a group of representative chunk hashes selected from the super-chunk. Thus we can send only the hand-print to a limited number of candidate nodes to select the best destination. In our implementation, we tried 8, 16, and 32 hashes for the size of the hand-print; the results showed that using 16 hashes resulted in better deduplication ratios than 8, while the difference between 16 and 32 was negligible. Thus we selected 16 hashes as the hand-print chunk super-unit.

(7) *Produck* [8]: *Produck* uses Probabilistic Counting with Stochastic Averaging (PCSA), a technique to improve the throughput of the system while maintaining a balanced load distribution. PCSA keeps a set of bitmaps that are used to quickly estimate the cardinality of the overlap between the incoming super-chunk and super-chunks on each node. In our implementation, a super-chunk contains $15 \times 1,024 = 15,360$ chunks, as suggested in the original Produck paper. To reduce estimation error, the paper suggests using 8,192 bitmaps for each super-chunk and storage node. Our data set contains 8×10^{10} chunks using 4KB chunking, which would require 37 bits to uniquely identify each chunk. Since we used a 48-bit hash function, we chose 1,024 bitmaps, indexed from 10 bits of the hash, to reduce estimation error. That left 38 bits for calculating the cardinality.

Table 4 summarizes the algorithms and their corresponding design principles and properties.

5.2 Experimental Results on Key Metrics

We used the following three metrics to evaluate the performance of the seven cluster deduplication algorithms: (1) Cluster-Deduplication Ratio, (2) Load-Distribution, and (3) Communication Overhead. We now describe each metric as well as the performance of each algorithm using these metrics.

5.2.1 Cluster Deduplication Ratio. We define the *cluster-deduplication ratio* as the total logical data size divided by the combined size of physical data on all storage nodes after deduplication. This metric directly shows the space savings when using a cluster deduplication system. Since redundant data may exist among different storage nodes, the deduplication ratio will usually be smaller than in a single-node system. Thus, one of the main tasks of the data routing algorithm is to assign duplicated data to the same node so as to increase the deduplication ratio. Figure 14 shows the cluster-deduplication ratio of different data-routing algorithms using *Homes*.

To check how the deduplication ratio changes as the number of storage nodes in the cluster increases, we implemented clusters that have one node (corresponding to single-node deduplication), 8, 32, and 128 nodes. The results are shown in Figure 14. We can see that routing data based on the file type can achieve the best deduplication ratio among all algorithms. In addition, even when using 128 storage nodes, routing by file type can reach a deduplication ratio as high as 192, compared

Table 4. Cluster deduplication algorithms used in this study

Cluster Algorithm	**Stateless vs. Stateful**	**Centralized vs. Distributed**	**Routing Unit**	**Deduplication Unit**	**Exact vs. Approximate Deduplication**
Stateless	Stateless	Distributed	Super-Chunk	Chunk	Approximate
Extreme Binning	Stateless	Distributed	File	Chunk	Approximate
Based on File Type	Stateless	Distributed	File	Chunk	Approximate
HYDRAstor	Stateless	Distributed	Large Chunk	Large Chunk	Exact
Stateful	Stateful	Distributed	Super-Chunk	Chunk	Approximate
Sigma-Dedup	Stateful	Distributed	Super-Chunk	Chunk	Approximate
Produck	Stateful	Centralized	Large Super-Chunk	Chunk	Approximate

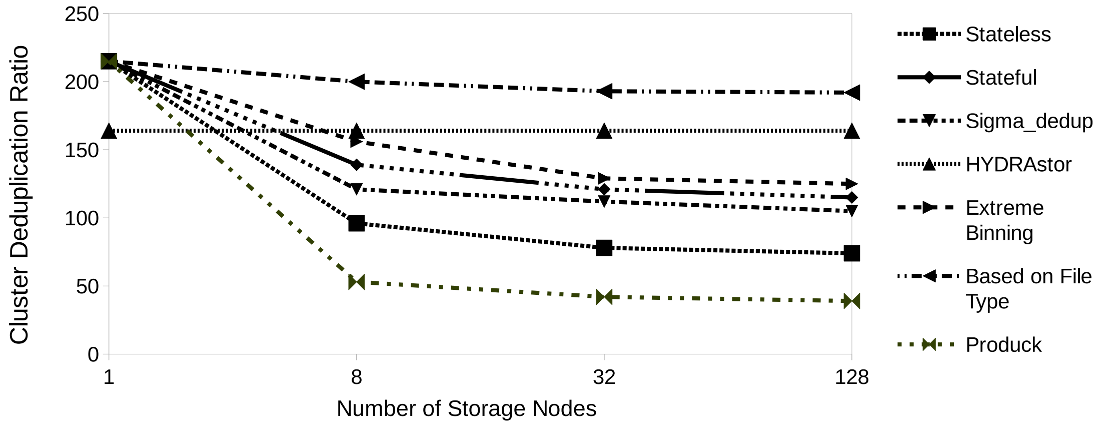


Fig. 14. Cluster-Deduplication Ratios for each algorithm

with 215 on single-node system. This result confirms the conclusion that redundancy mainly exists in a given file type, and that duplicate data across different file types is negligible.

HYDRAstor is the only exact cluster deduplication algorithm[1] we implemented, which is why its deduplication ratio was not influenced as we increased the cluster size. To trade off routing and deduplication overhead, HYDRAstor uses 64KB chunking for both data routing and deduplication. Thus its deduplication ratio is lower than that of all other algorithms in single-node mode, but its exact-deduplication character makes it the second-best algorithm in terms of deduplication ratio in clusters that have more than 8 nodes.

[1]Here, "exact" means that all duplicate chunks are routed to the same node, so that precisely one copy of each unique chunk is stored.

Stateful, Sigma-dedup, and Produck are three similar data routing algorithms. They are all stateful algorithms, using a super-chunk for routing and single chunks for deduplication. Stateful gets the best deduplication ratio among the three, because it will consider all chunks that belong to a super-chunk and search for them in each storage node.

Conversely, Sigma-dedup uses a hand-print technique, which only searches representative hashes in a super-chunk and in fewer nodes. Therefore, its communication (shown in Section 5.2.3) and CPU overhead are lower but at the cost of a drop in deduplication ratios.

Produck is a special data-routing scheme: it routes data using the PCSA technique. But its deduplication ratio is not as high as other stateful algorithms, mainly because we are using a short 48-bit hash for each chunk. PCSA uses bitmaps to check the cardinality of the overlap between chunks in a super-chunk and chunks in a node. In the Produck paper, the authors used 8,192 bitmaps to reduce the error rate. But in *Homes*, we have 8×10^{10} chunks using 4KB chunking. Since we use a shorter hash, we have limited bits left to reduce the error rate, which influences the deduplication ratio. In addition, the strict load-balancing strategy adapted by Produck also affects the deduplication ratio, which we discuss in Section 5.2.2.

The deduplication ratio of the Stateless routing scheme is not as good as stateful ones, because Stateless assigns data based on a hash table without considering previous assignment information.

Extreme Binning routes data by files. Although it is also a stateless routing algorithm, it gets a higher deduplication ratio than Stateful and Sigma-dedup. This result shows that in *Homes*, routing by file (both in Extreme Binning and in routing based on file type) can get a better deduplication ratio than routing by super-chunk. This is mainly because we are collecting data every day, and most files are not modified daily. As a result, we have many redundant files in our data-set.

Figure 14 shows an interesting phenomenon. Although deduplication ratios for each cluster algorithm drop as the number of storage nodes grows (except for HYDRAstor, whose deduplication ratio is not influenced by the number of storage nodes), the figure shows no single point of inversion for any two algorithms. This means that all these cluster algorithms have good scalability in their deduplication ratio, and no algorithm's deduplication ratio will drop more severely than others when we increase the number of storage nodes.

5.2.2 Load Distribution. Both physical and logical load distributions are important metrics to a cluster deduplication algorithm. Physical load distribution shows the capacity usage at the nodes. If the space of each storage node is consumed at a balanced rate, then we can fully use the system's capacity without having to redistribute data among nodes. In addition, when a node becomes full, similar data will be routed to a less loaded node, hurting the overall deduplication ratio. Logical load distribution has a direct impact on the I/O performance, because a logically overloaded node tends to deal with more I/O requests. Therefore, a logically load-balanced system can avoid having one node become I/O-bottlenecked and support more I/O demand from clients. The metric we used to evaluate the performance of load balance is the Coefficient of Variation (the ratio of the standard deviation to the mean) of the load across all cluster nodes, measured in bytes stored (for physical load) or transferred (for logical load).

Figures 15 and 16 show the physical and logical load distribution of each algorithm, respectively. From Figure 15 we can see that routing based on file type, which achieves the highest deduplication ratio, performs much worse than other algorithms in terms of physical-data load balance. Extreme Binning, which also uses files as its routing unit, performs much better than routing by file type, but is nevertheless worse than algorithms that use a super-chunk as the routing unit. This result shows that although routing by file can produce a better deduplication ratio, it will also lead to higher data skew (imbalance), mainly because of the large difference in the sizes among different file types. For other algorithms, stateless routing algorithms perform better than stateful ones, mainly due

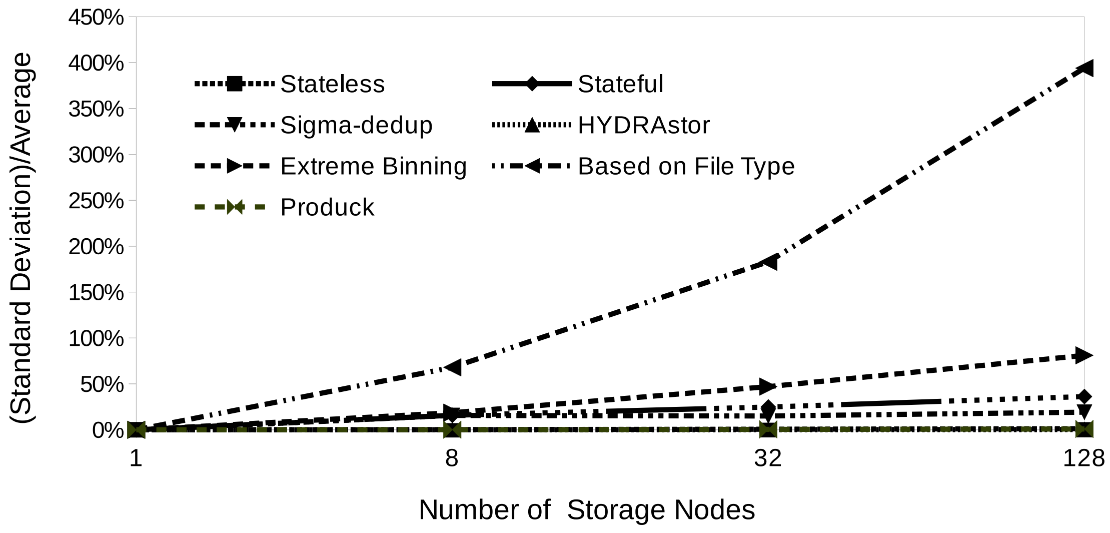


Fig. 15. Physical Load Distribution for each algorithm

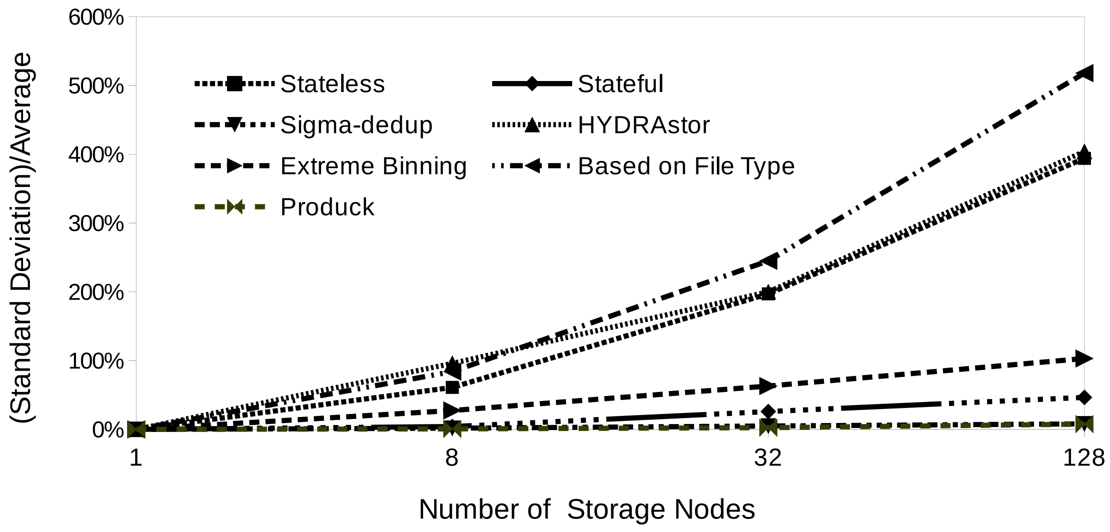


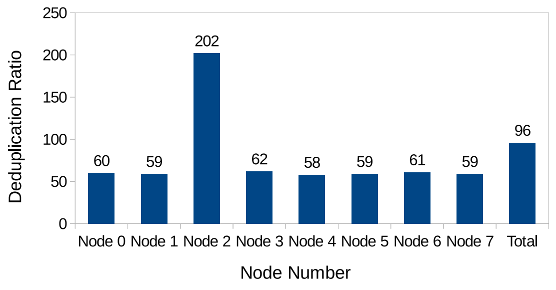
Fig. 16. Logical Load Distribution for each algorithm

to the random nature of the hash table. Produck has the best physical load balance, because of its strict load-balancing strategy. HYDRAstor, which is a stateless and exact deduplication system, performs well both in terms of deduplication ratios and physical load balance.

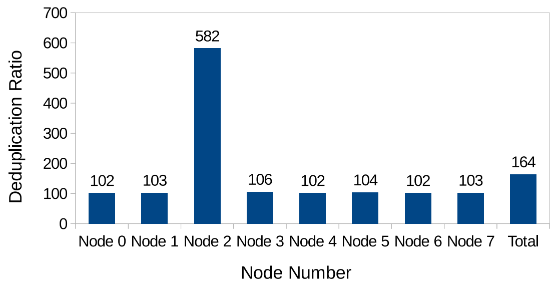
Overall, we can see that when adopting an approximate deduplication strategy, algorithms that get better deduplication ratios tend to perform poorly in terms of physical load balance, which means that the deduplication ratio and physical load balance are design trade-offs in cluster deduplication.

Figure 16 shows an interesting result in logical load distribution. Stateless and HYDRAstor, which are both stateless routing algorithms, lead to high data skew in terms of the logical load distribution.

This is opposite to their performance in terms of physical load distribution. To investigate the reason for this phenomenon, we checked the deduplication ratio of each node using these two algorithms. The results are shown in Figure 17. We can see that the per-node deduplication ratio of one node is much higher than all other nodes; this is caused by the presence of all-zero chunks, which are frequent in many workloads and can reach the maximal size when using the content-defined chunking method. In *Homes*, all-zero chunks occupy 23% of the total space; thus, although these two algorithms get a good physical load distribution, their logical load balance is poor. We confirmed that clusters with 32 and 128 nodes show the same results.



(a) Deduplication Ratio of each node in Stateless



(b) Deduplication Ratio of each node in HYDRAstor

Fig. 17. Deduplication Ratio of each node using Stateless and HYDRAstor algorithms.

5.2.3 Communication Overhead. To ensure a fair comparison of communication overheads among all algorithms, we made two pragmatic assumptions. First, in all routing algorithms, after a destination node is selected, only chunk *hashes* are sent to storage nodes to detect duplicate chunks; afterwards, only non-duplicate chunks are sent to storage nodes in large (1MB) packets. This reduces the total amount of data transferred in the cluster significantly, considering our high deduplication ratio. Second, variations in meta-data transfer overheads, such as sending the file recipes, are not included in our comparison. This is because the papers that presented the algorithms [1, 4, 8, 12, 36] did not detail how file recipes are transferred and stored in the master node. For example, recipes can be sent to master node by super-chunk or by file, or they can be sent after the whole snapshot is stored. For simplicity, we chose the second strategy in our implementation: because each snapshot has its own recipe, sending per-snapshot recipes is more straightforward. As a result, the meta-data communication overheads for all algorithms are the same. We plan to evaluate more comprehensive file-recipe transfer and storage policies in the future.

Figure 18 shows the communication overheads of each algorithm, which are represented by the number of messages[2] needed to route all the data. As we can see, Stateful incurs the most communication overhead, because our implementation is a distributed Stateful algorithm. To route a super-chunk, its information will be sent to all storage nodes to request its similarity index (i.e., how many chunks in a super-chunk can be found in that node). An alternative approach would be to store all the Bloom filters in the master node, which would significantly decrease the communication

[2]Because the data itself is so much larger than its hash, the overhead in terms of total bytes transferred is similar in character to the deduplication ratio. For that reason, we measure overhead in terms of message count, which captures the latency and networking costs of communicating hashes (which currently are not batched) to nodes without being distorted by the data transfers—which must take place in any case.

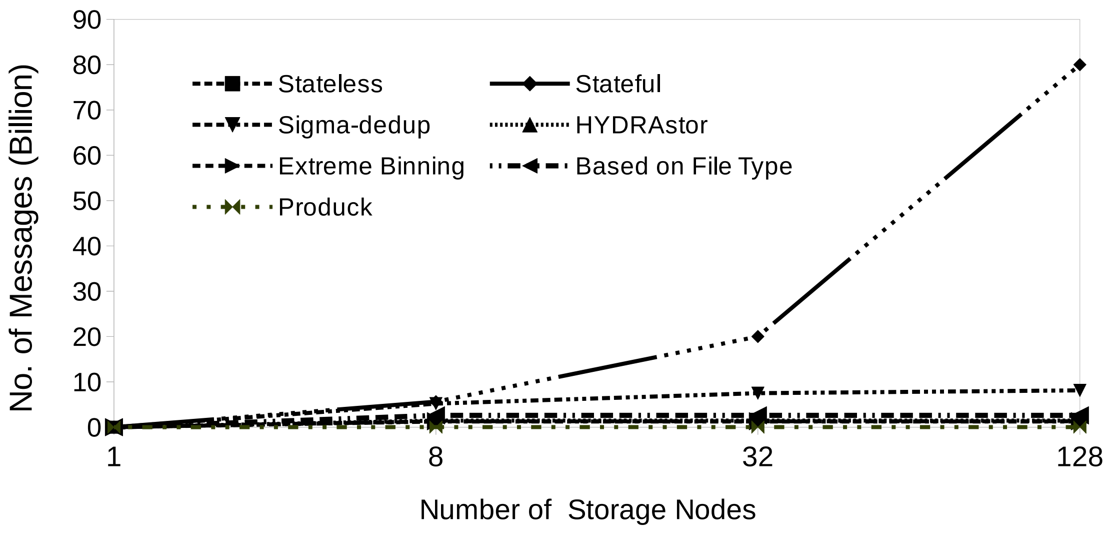


Fig. 18. Communication overheads in each cluster deduplication algorithm.

overhead. Doing so would not influence the deduplication ratio and load distribution, but the master node might become a bottleneck as the system scales up.

Another stateful algorithm, Sigma-dedup, is less expensive than Stateful because the hand-print technique reduces the number of candidate nodes, but its stateful nature causes more communication overhead than stateless approaches.

Produck incurs the least communication overhead, because (1) its average routing size (15,360 chunks) is much larger than other algorithms, and (2) routing is performed by the master node, which means that it does not need to broadcast super-chunk information to all storage nodes.

Routing by files (Extreme Binning and Routing by File Type) rank between super-chunk-based stateful algorithms (Stateful and Sigma-dedup) and super-chunk-based stateless ones (Stateless and HYDRAstor). Extreme Binning has much less communication overhead than Routing by File Type, because Extreme Binning removes duplicate files directly based on whole-file hashes. As we discussed in Section 4.2, many small duplicate files exist in our data set, so the communication overhead of Extreme Binning is reduced significantly.

Stateless and HYDRAstor cause lower communication overhead than other algorithms (except for Produck), because each of their clients can choose the destination node without sending any messages. In our HYDRAstor implementation, chunks are not assigned directly to storage nodes, because that would lead to excessive communication. Our strategy is to put all chunks that are sent to the same node into a buffer and to wait until the buffer is full before sending its contents together. Each of our buffers can store 16 chunks; we chose 16 so that the buffer size would be close to the super-chunk size adopted by other algorithms. The network cost of HYDRAstor is slightly higher than Stateless, since our use of content-defined chunking causes many chunks to be much smaller than the chunking size. This results in HYDRAstor needing more messages to transfer the same amount of data.

5.3 Summary of Cluster Deduplication

Based on our analysis of cluster deduplication ratios, logical and physical load distribution, and communication overhead, we draw the following four conclusions:

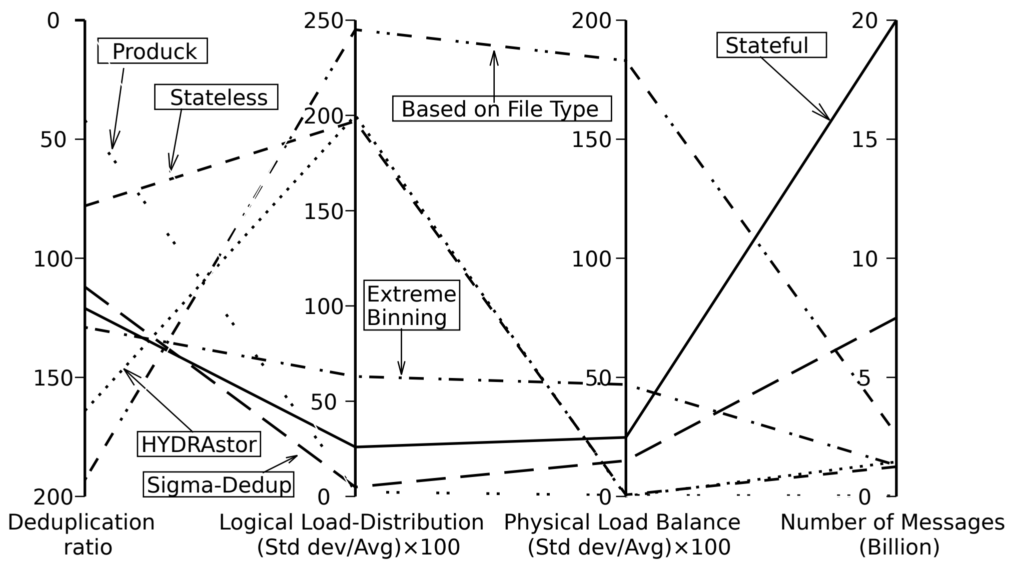


Fig. 19. Summary of the performance of each algorithm as evaluated on key metrics. For all metrics, values at the lower end of the vertical scale are better.

(1) Routing by files can generally achieve a better deduplication ratio than other schemes using *Homes*, because most of the duplicate data exists in matching file types. In addition, since *Homes* is a daily backup system, many duplicated files exist in our data set. However, routing by files leads to poor load balance, since there are large differences in file sizes among different file types.
(2) Using a larger chunking size does not lead to a large difference in deduplication ratio, especially when considering meta-data overhead as we discussed in Section 4.1. Therefore, HYDRAstor, which adopts 64KB chunking, actually performs well in terms of deduplication ratios, especially when the system scales up.
(3) Super-chunk and large-chunk based stateless routing strategies (Stateless and HYDRAstor) can achieve good physical load distributions; however, unexpectedly, their logical load distributions are much worse due to the fact that in *Homes*, a small number of chunks have a fairly high deduplication ratio: all these chunks will be assigned to the same node using these routing algorithms, making the node overloaded in terms of its logical size.
(4) Distributed, stateful routing algorithms lead to much higher communication overhead than others. Using a stateless strategy, routing by files leads to higher communication overhead than routing by super-chunk, because most files in *Homes* are small.

Figure 19 shows the overall performance of each algorithm using a parallel coordinates graph, where every axis (coordinate) represents a metric (deduplication ratio, physical and logical load distribution, and communication overhead) and each algorithm is depicted as a line crossing the axes at the corresponding coordinates.

6 CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

We studied a locally collected data set that spans a period of 2.5 years (over 4,000 daily user snapshots accounting for over 21 months). Our data is publicly available, with continuing release updates.

The long time span and rich meta-data make the data valuable for studies of realistic long-term trends. Our findings can help guide future deduplication storage designs.

We analyzed our data set, *Homes*, as it would be seen by both a single-node deduplication system and a cluster deduplication system. In our single-node analysis, we found that a smaller chunk size does not always save more space, given the cost of additional meta-data. In *Homes*, a 32KB or even larger chunk size made the system the most space-efficient. Whole-file chunking produced the worst results in our study, because large files tended to dominate the data and had much lower deduplication ratios than smaller files. Surprisingly, our user-centric studies showed that data belonging to each user varies significantly in deduplication ratio and sensitivity to chunk size. We also found a large difference in deduplication ratios across users, an observation that could help in grouping users for future cluster storage. Our detailed study of users who shared data showed that they have a higher deduplication ratio than average, which suggests that data shared among users tends to be more popular in general.

In our cluster deduplication analysis, we first classified several well known data-routing algorithms based on their characteristics; we then studied their performance in terms of deduplication ratios, load balance, and communication overhead. The results showed that routing data to nodes by whole files can get the best deduplication ratio, but it also leads toward large data skews because of significant differences in file sizes. Using a larger chunking size, such as 64KB in HYDRAstor, is a good choice for cluster deduplication using *Homes*. Both data-routing and deduplication overheads (including chunking, indexing, and meta-data overheads) are reduced, while the drop in deduplication ratio is small when using a large chunk size. The logical and physical load distribution using a given cluster algorithm can be fairly different, because some chunks have a much higher deduplication ratio than others, making the nodes that store highly repetitive chunks experience much higher logical load.

Limitations and Future Work. Although we believe this work is the first of its kind from the perspective of both dataset size and clustering analysis, there are a few limitations to this study. Several of these limitations are part of our future work, some of which we have begun to investigate using this data set and additional data sets we have been collecting.

First, this study does not consider aging and fragmentation effects, which are complex topics that are orthogonal to this work. These effects are very sensitive to file system types, device types, and specific layouts on media [2]. Since *Homes* has a long duration, a future study can investigate how fragmentation accumulates and how it affects backup and restore speeds. While collecting the snapshots, we also found that fragmentation decreases backup throughput because a traditional depth-first scan of the file system causes newly added or modified chunks to be stored into new containers, leading to cache misses in future scans. We are currently investigating alternative scan orders to improve cache hit ratios and restore speeds.

Second, this study does not evaluate restore performance, which is affected by various factors and techniques that either write duplicates to maintain restore locality or analyze the file being restored to reduce random I/O [10, 21, 32]. To our knowledge such techniques have not been investigated in depth for deduplication clusters [34].

Third, due to serious privacy concerns, we were able to collect data only within our own local environment, which includes a few dozen (mostly graduate) students and faculty. This home-directories environment approximates engineers and software developers working on Unix-based sites. We hope that our study and the massive data set we released will encourage others to collect, release, and investigate multi-year data sets for other environments (e.g., Windows users, MS Exchange, databases, Web servers, etc.).

ACKNOWLEDGMENTS

We thank the ACM TOS anonymous reviewers for their helpful comments. This work was made possible in part thanks to Dell-EMC, NetApp, and IBM support; NSF awards CNS-1251137, CNS-1302246, CNS-1305360, and CNS-1622832; ONR award N00014-16-1-2264; and the National Natural Science Foundation of China under Grants No. 61433019 and U1435217, and China 863 program grant 2015AA015305.

REFERENCES

[1] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge. Extreme binning: Scalable, parallel deduplication for chunk-based file backup. In *Proceedings of the MASCOTS Conference*, pages 1–9, London, UK, 2009. IEEE Computer Society.

[2] Zhen Cao, Vasily Tarasov, Hari Raman, Dean Hildebrand, and Erez Zadok. On the performance variation in modern storage stacks. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST)*, pages 329–343, Santa Clara, CA, February/March 2017. USENIX Association.

[3] B. Debnath, S. Sengupta, and J. Li. ChunkStash: Speeding up inline storage deduplication using flash memory. In *Proceedings of the USENIX Annual Technical Conference*, page 16, Boston, MA, USA, 2010. USENIX.

[4] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *Proceedings of the Ninth USENIX Conference on File and Storage Technologies (FAST '11)*, pages 15–29, SAN JOSE, CA, USA, 2011. USENIX.

[5] F. Douglis, D. Bhardwaj, H. Qian, and P. Shilane. Content-aware load balancing for distributed backup. In *Proceedings of USENIX Large Installation System Administration Conference*, pages 13–13, SAN Diego, CA, USA, 2011. USENIX.

[6] A. El-Shimi, R. Kalach, A. Kumar, A. Oltean, J. Li, and S. Sengupta. Primary data deduplication—large scale study and system design. In *Proceedings of the USENIX Annual Technical Conference*, pages 285–296, BOSTON, MA, USA, 2012. USENIX.

[7] Kave Eshghi, Mark Lillibridge, Deepavali Bhagwat, and Mark Watkins. Improving multi-node deduplication performance for interleaved data via sticky-auction routing. Technical Report HPL-2015-77, HP Laboratories, 2015. https://www.labs.hpe.com/techreports/2015/HPL-2015-77.pdf.

[8] D. Frey, A. Kermarrec, and K. Kloudas. Probabilistic deduplication for cluster-based storage systems. In *Proceedings of the Symposium on Cloud Computing (SOCC)*, page 17, SAN Jose, CA, USA, 2012. ACM.

[9] Fslhomes data set and tools, 2016. tracer.filesystems.org.

[10] Min Fu, Dan Feng, Yu Hua, Xubin He, and Zuoning Chen. Accelerating restore and garbage collection in deduplication-based backup systems via exploiting history information. In *Proceedings of Annual Technical Conference*, pages 181–192, Philadelphia, PA, USA, 2014. USENIX.

[11] Y. Fu, N. Xiao, X. Liao, and F. Liu. Application-aware client-side data reduction and encryption of personal data in cloud backup services. *Journal of Computer Science and Technology*, 28(6):1012–1024, November 2013.

[12] Yinjin Fu, Hong Jiang, and Nong Xiao. A Scalable Inline Cluster Deduplication Framework for Big Data Protection. In *Proceedings of International Conference on Middleware*, pages 354–373, Montreal, Quebec, Canada, 2012. ACM.

[13] A. George and B. Medha. Identifying trends in enterprise data protection systems. In *USENIX Annual Technical Conference*, pages 151–164, SANTA CLARA, CA, USA, 2015. USENIX.

[14] A. Gharaibeh, C. Constantinescu, M. Lu, A. Sharma, R. Routray, P. Sarkar, D. Pease, and M. Ripeanu. DedupT: Deduplication for tape systems. In *Proceedings of 30th Symposium on Mass Storage Systems and Technologies(MSST)*, pages 1–11, SANTA CLARA, CA, USA, 2014. IEEE Computer Society.

[15] Jhon Gratz and David Reinsel. The digital universe decade - are you ready? IDC White Paper, www.idc.com, 2010.

[16] F. Guo and P. Efstathopoulos. Building a high-performance deduplication system. In *Proceedings of the USENIX Annual Technical Conference*, pages 25–25, Portland, OR, USA, 2011. USENIX.

[17] M. Jianting. A deduplication-based data archiving system. In *Proceedings of the International Conference on Image, Vision and Computing (ICIVC)*, pages 1–12, Shanghai, China, 2012. ACM.

[18] K. Jin and E. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of the Israeli Experimental Systems Conference (SYSTOR)*, page 7, Haifa, Israel, 2009. ACM.

[19] R. Koller and R. Rangaswami. I/O deduplication: Utilizing content similarity to improve I/O performance. *ACM Transaction on Storage (TOS)*, 6(3):13, 2010.

[20] M. Li, C. Qin, and P. Lee. Cdstore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. In *USENIX Annual Technical Conference*, pages 111–124, SANTA CLARA, CA, USA, 2015. USENIX.

[21] M. Lillibridge and K. Eshghi. Improving restore speed for backup systems that use inline chunk-based deduplication. In *Proceedings of the Eleventh USENIX Conference on File and Storage Technologies (FAST '13)*, pages 183–197, SAN JOSE, CA, USA, 2013. USENIX.

[22] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proceedings of the Seventh USENIX Conference on File and Storage Technologies (FAST '09)*, pages 111–123, SAN JOSE, CA, USA, 2009. USENIX.

[23] X. Lin, F. Douglis, J. Li, X. Li, R. Ricci, S. Smaldone, and G. Wallace. Metadata considered harmful ... to deduplication. In *Proceedings of the 7th USENIX Conference on Hot Topics in Storage and File Systems*, page 11, SANTA CLARA, CA, USA, 2015. USENIX.

[24] X. Lin, M. Hibler, E. Eide, and R. Ricci. Using deduplicating storage for efficient disk image deployment. In *Proceedings of IEEE International Conference on Software Testing, Verification and Validation*, pages 1–14, Graz, Austria, 2015. IEEE Computer Society.

[25] M. Lu, D. Chambliss, J. Glider, and C. Constantinescu. Insights for data reduction in primary storage: A practical analysis. In *Proceedings of the Israeli Experimental Systems Conference (SYSTOR)*, page 14, Haifa, Israel, 2012. ACM.

[26] D. Meister and A. Brinkmann. Multi-level comparison of data deduplication in a backup scenario. In *Proceedings of the Israeli Experimental Systems Conference (SYSTOR)*, 2009.

[27] D. Meister and A. Brinkmann. dedupv1: Improving deduplication throughput using solid state drives (SSD). In *Proceedings of the MSST Conference*, pages 1–6, Inline Village, NV, USA, 2010. IEEE Computer Society.

[28] D. Meister, A. Brinkmann, and T. Suss. File recipe compression in data deduplication systems. In *Proceedings of the Eleventh USENIX Conference on File and Storage Technologies (FAST '13)*, pages 175–182, SAN JOSE, CA, USA, 2013. USENIX.

[29] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel. A study on data deduplication in HPC storage systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, page 7, Salt lake City, Utah, USA, 2012. IEEE Computer Society.

[30] D. Meyer and W. Bolosky. A study of practical deduplication. *ACM Transaction on Storage (TOS)*, 7(4):14, 2011.

[31] N. Park and D. Lilja. Characterizing datasets for data deduplication in backup applications. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–10, Atlanta, GA, USA, 2010. IEEE Computer Society.

[32] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. iDedup: Latency-aware, inline data deduplication for primary storage. In *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, 2012.

[33] Zhen Sun, Geoff Kuenning, Sonam Mandal, Philip Shilane, Vasily Tarasov, Nong Xiao, and Erez Zadok. A long-term user-centric analysis of deduplication patterns. In *Proceedings of the 32nd International IEEE Symposium on Mass Storage Systems and Technologies (MSST '16)*, pages 1–7, Santa Clara, California, May 2016. IEEE.

[34] Yujuan Tan, Dan Feng, Fangting Huang, and Zhichao Yan. Sort: A similarity-ownership based routing scheme to improve data read performance for deduplication clusters. *IJACT*, 3(9):270–277, 2011.

[35] V. Tarasov, A. Mudrankitony, W. Buik, P. Shilane, G. Kuenning, and E. Zadok. Generating realistic datasets for deduplication analysis. In *Proceedings of the USENIX Annual Technical Conference*, pages 261–272, BOSTON, MA, USA, 2012. USENIX.

[36] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Calkowski, C. Dubnicki, and A. Bohra. HydraFS: a high-throughput file system for the HYDRAstor content-addressable storage system. In *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST '10)*, pages 225–239, SAN JOSE, CA, USA, 2010. USENIX.

[37] C. Vaughn, C. Miller, O. Ekenta, H. Sun, M. Bhadkamkar, P. Efstathopoulos, and E. Kardes. Soothsayer: Predicting capacity usage in backup storage systems. In *Proceedings of MASCOTS conference*, pages 208–217, Atlanta, GA, USA, 2015. IEEE.

[38] R. Villars, C. Olofson, and M. Eastwood. Big data: What it is and why you should care. A White Paper from www.idc.com, June 2011.

[39] 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 (FAST '12)*, pages 33–48, SAN JOSE, CA, USA, 2012. USENIX.

[40] J. Wei, H. Jiang, K. Zhou, and D. Feng. MAD2: A scalable high-throughput exact deduplication approach for network backup services. In *Proceedings of the MSST Conference*, pages 1–14, Inline Village, NV, USA, 2010. IEEE Computer Society.

[41] W. Xia, H. Jiang, D. Feng, and Y. Hua. SiLo: A similarity-locality based near-exact deduplication scheme with low RAM overhead and high throughput. In *Proceedings of the USENIX Annual Technical Conference*, pages 26–28, Portland, OR, USA, 2011. USENIX.

[42] T. Yang, H. Jiang, D. Feng, Z. Niu, K. Zhou, , and Y. Wan. DEBAR: A scalable high-performance de-duplication storage system for backup and archiving. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pages 1–12, Atlanta, GA, USA, 2010. IEEE Computer Society.

[43] Y. Zhou, D. Feng, W. Xia, M. Fu, F. Huang, Y. Zhang, and C. Li. Secdep: A user-aware efficient fine-grained secure dedupication scheme with multi-level key management. In *Proceedings of 31th Symposium on Mass Storage Systems and Technologies(MSST)*, pages 1–14, SANTA CLARA, CA, USA, 2015. IEEE Computer Society.

[44] 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*, pages 1–14, SAN JOSE, CA, USA, 2008. USENIX.