Palantir: Hierarchical Similarity Detection for Post-Deduplication Delta Compression

Hongming Huang
honhuang7-c@my.cityu.edu.hk
City University of Hong Kong
Huawei Technologies
Hong Kong SAR, China

Peng Wang
wangpeng423@huawei.com
Theory Lab, Central Research Institute, 2012 Labs, Huawei Technologies
Hong Kong SAR, China

Qiang Su
qiangsu3-c@my.cityu.edu.hk
City University of Hong Kong
Hong Kong SAR, China

Hong Xu
hongxu@cuhk.edu.hk
The Chinese University of Hong Kong
Hong Kong SAR, China

Chun Jason Xue
jason.xue@mbzuai.ac.ae
Mohamed bin Zayed University of Artificial Intelligence
City University of Hong Kong
United Arab Emirates, Hong Kong SAR, China

André Brinkmann
brinkman@uni-mainz.de
Johannes Gutenberg University Mainz
Germany

Abstract

Deduplication compresses backup data by identifying and removing duplicate blocks. However, deduplication cannot detect when two blocks are very similar, which opens up opportunities for further data reduction using delta compression. Most existing works find similar blocks by characterizing each block by a set of features and matching similar blocks using coarse-grained super-features. If two blocks share a super-feature, delta compression only needs to store their delta for the new block.

Existing delta compression techniques constrain their super-features to find only matching blocks that are likely to be very similar. Palantir introduces hierarchical super-features with different sensitivities to block similarities to find more candidates of similar blocks and increase overall backup compression including deduplication by 7.3% over N-Transform and Odess and 26.5% over Finesse. The overhead of Palantir for storing additional super-features is overcome by exploiting temporal localities of backup streams, and the throughput penalty is within 7.7%. Palantir also introduces a false positive filter to discard matching blocks that are counterproductive for the overall data reduction.

ACM Reference Format:
Hongming Huang, Peng Wang, Qiang Su, Hong Xu, Chun Jason Xue, and André Brinkmann. 2024. Palantir: Hierarchical Similarity Detection for Post-Deduplication Delta Compression. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3620665.3640353

1 Introduction

Data centers protect critical data against system crashes, cyberattacks, or accidental deletion by regularly backing up primary data. The amount of data stored on backup storage systems can be orders of magnitude larger than the size of the primary data because the same primary data can be part of many generations of backup data.The resulting redundant data takes up a lot of unnecessary storage space, making data reduction techniques such as deduplication and delta compression essential for modern backup systems.

Data deduplication reduces the physical size of backup data by storing only unique data blocks. Most deduplication solutions characterize blocks by computing their cryptographic fingerprints. Two blocks are considered identical if their fingerprints are the same [22, 28]. Deduplication is often coupled with lossless compression of the remaining unique data to further improve storage efficiency [8, 19]. Deduplication leaves similar blocks with minor differences untouched, because blocks that differ by a few bytes produce completely different fingerprints. To take advantage of similar blocks, delta compression is introduced to find a potential base block for a new block and then to store only the delta between them [26, 31].

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0385-0/24/04
https://doi.org/10.1145/3620665.3640353

The critical steps in delta compression are estimating the similarity of two blocks and finding the most similar candidate. The most commonly used approach for delta compression of backup data is “N-transform super-feature (SF)”, which is based on Broder’s theorem that the number of independently computed features shared by two blocks is proportional to their similarity [3, 4]. N-transform SF computes N independent linear transformations for each output of a rolling hash function on the block; the minimum or maximum hash value of each linear transformation is then an independent feature. These features are combined into k super-features by partitioning them into k subsets of s elements each and computing one super-feature per subset. Two blocks are considered similar if they have at least one super-feature in common, and N-transform SF simply takes the first matching block as the base block.

However, existing implementations of the N-transform SF approach, such as the original N-transform SF [4], Finesse [31], and Odess [34], are unable to overcome the accuracy-versus-coverage dilemma. They use fixed parameters (k, s) to compute super-features. In particular, the choice of s determines the probability threshold that two blocks with a given resemblance will be matched as similar. A high value for s ensures that the detected blocks are very similar to the new block, whereas many blocks that are still good candidates for delta compression will be discarded, resulting in a low coverage of matching blocks. On the other hand, choosing a lower value for s may not ensure high accuracy in the candidate selection step, in which case the first fit when selecting a matching block may not have enough similarity to the new block, which is detrimental to the compression ratio.

As a main contribution, our system Palantir introduces a tiered structure to exploit multiple thresholds to detect blocks with different degrees of similarity to improve both compression and coverage compared to using a single threshold. Tier i builds its super-features for a fixed threshold s_i, and s_i is decreasing for higher tiers. A decreasing number of features s per super-feature implies an increasing probability for the same blocks to be classified as similar. Palantir considers two blocks to be similar and candidates for delta compression, if they share at least one common super-feature from any tier. Similarity detection is then performed hierarchically, starting from the tier with the highest threshold, to first detect base blocks with very high similarity.

Base blocks detected for lower thresholds may be false positives for which delta compression does not increase or even decrease the overall compression ratio. Therefore, Palantir’s second contribution is an adaptive filter to detect false positives found using low-threshold super-features. The false positive filter considers both the reduction due to delta compression and the estimated remaining compressibility of the delta block and flags the base block as a false positive if its compressibility is below the lossless compression ratio averaged over a sliding window of previous chunks. We will show that the results of this false-positive filter are very close to the results of an optimal greedy solution in our scenario.

Palantir stores more super-features to build its multiple tiers compared to fixed (k, s) approaches. However, similar blocks detected by super-features with lower thresholds have lower expected similarity and therefore contribute less to the overall compression ratio. The third contribution of Palantir is a metadata lifecycle manager that keeps super-features from higher tiers for shorter periods of time and successfully reduces the additional metadata overhead compared to fixed (k, s) approaches from a linear scale to a constant while maintaining a high compression ratio.

We have built a prototypical backup environment that pipelines deduplication, delta compression, and lossless compression and includes implementations of Palantir, N-Transform SF, Odess, and Finesse [4, 31, 34]. Our evaluation is based on five backup datasets generated from real-world cloud datasets with synthetically generated backup versions and a Linux source code repository. The evaluation shows that the tiered super-features and the false positive filter of Palantir can improve the overall compression ratio (including deduplication) by an average of 7.3% over N-Transform SF and Odess and 26.5% over Finesse. Similarity detection coverage is increased by 27.4% over N-Transform SF and Odess and by 95.8% over Finesse. The false-positive filter improves the compression ratio up to 6.4% for datasets where tiered super-features produce many false positives, demonstrating the robustness of Palantir. Its throughput is 7.7% lower than that of Odess, which is acceptable for backup scenarios. The metadata lifecycle manager is able to reduce the additional metadata for tiered super-features from 133% to 32%. We also see variability between workloads. Some datasets benefit more from the tiered approach, while others only benefit from the false positive filter.

Outline. The remainder of this paper is structured as follows. Section 2 introduces the background and discusses our motivation. Section 3 discusses the challenge and describes the design of Palantir. Section 4 introduces the implementation of the prototype system. Section 5 compares the performance and compression ratio of Palantir with the N-transform SF, Odess, and Finesse. The related work is presented in Section 6, and we will summarize our work and give an outlook on future work in the conclusion.

2 Background and Motivation

We start by introducing the background and related work on data reduction for backup storage systems. Then we focus on delta compression, discuss the limitations of the current super-feature approaches, and motivate the need for hierarchical super-features to further improve compression ratios.

2.1 Background on data reduction for backup storage

Modern storage systems typically employ three data reduction techniques [25, 31]: deduplication, delta compression, and lossless compression (see Figure 1). First, deduplication ensures that data blocks with identical content are stored only once. Then, delta compression further reduces storage consumption by storing only the differences between pairs of similar but not identical blocks. Finally, conventional lossless compression (such as Zip or ZSTD) is used to compress the remaining data by exploiting repeated bit patterns. Since lossless compression is generic and well studied [11, 24, 33], we will focus here on deduplication and delta compression.

Deduplication reduces storage consumption by storing identical blocks only once [7, 14, 18, 19, 27, 32]. In a first step, data is chunked into either fixed-sized or content-defined data blocks. Content-defined chunking (CDC) uses a rolling hash function like Rabin fingerprints to calculate chunking positions based on the data content [23, 30]. Small data insertions or deletions therefore do not affect other chunking positions, increasing the overall deduplication ratio for backup data [17]. Deduplication then generates a strong cryptographic signature (called a fingerprint or FP) for each block of data using a hash algorithm such as SHA-2 [2], and maintains FPs of unique blocks in the FP database. An incoming block can be marked as being a duplicate if its FP is already stored in the database.

Delta compression. In backup systems, many highly similar data blocks are generated by small modifications. Delta compression, also called delta encoding, has been designed to eliminate such data redundancies between non-duplicate but highly similar blocks [12, 15, 25]. For each incoming block, it first searches for a similar block (the base block) and then stores only the differences (the delta block) between the incoming block and the base block. Integrating delta compression with deduplication and lossless compression in a large-capacity storage system is helpful for achieving a high data reduction ratio: 1) compared to deduplication, it can further compress similar but not identical blocks, and 2) compared to lossless compression, it can search for and eliminate data redundancy over a global area rather than just within the local compression window. Because of these benefits, delta compression is gaining popularity in backup storage systems [21, 31, 34, 35].

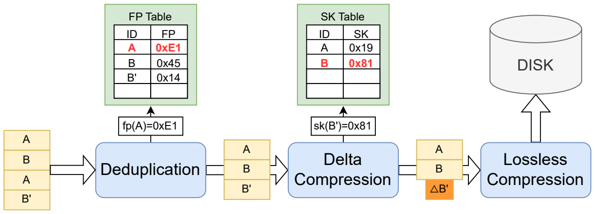


Figure 1. Backup storage system including deduplication, delta compression, and lossless compression.

The critical challenge for delta compression is how to find the base block that matches an incoming block. Most existing methods rely on well-designed hash functions [4, 31, 34] to calculate a *sketch* with the property that if two blocks generate the same sketch, they are likely to be very similar. Sketches are maintained in a sketch database. The system compares the sketches of a new block with the database to find a similar block and to perform delta compression. In Figure 1, block B′ has the same sketch as B and the system treats them as similar blocks and performs delta compression between them. Alternative machine learning approaches to find matching base blocks are still in their infancies and have to overcome scalability and performance challenges [21].

2.2 Similarity matching based on super-features

A straightforward way to find the most similar block is to compare an incoming block with all n blocks in the database, introducing O(n^2) algorithm complexity for inserting n elements, which is very inefficient. Therefore, most recent research chooses to exploit features [1, 6] or super-features [3, 4, 12, 25, 31, 34] for delta compression, which are more effective and efficient. Next, we discuss how to derive features and super-features by Broder's theorem [3].

Let $S(A, w)$ denote the set of shingles. A chunk A is partitioned into $m = length(A) - w + 1$ shingles of length w each, and each shingle $i \in \{0, ..., m-1\}$ contains the bytes i to $(i + w - 1)$ of A. The resemblance of two chunks A and B is then defined as

$$Resemblance_w (A, B) = \frac{|S(A, w) \cap S(B, w)|}{|S(A, w) \cup S(B, w)|}. \tag{1}$$

Broder's theorem now states that a set of N minimum values $min_N(S(A, w))$ and $min_N(S(B, w))$ can be used to estimate the $Resemblance_w (A, B)$. In the following, we call these minimum values *features* and we can, for instance, compute the N minimum values for each block by selecting the minimum hash values of N independent hash functions.

The resemblance of two blocks increases as they share more joint features, so that the quality of resemblance detection increases in N. For a new data block C, feature-based approaches therefore choose the block with the maximum number of common features with C as its base block. However, iteratively comparing N features between all blocks with at least one matching feature is extremely expensive when scaling N, as the underlying computational complexity grows in $O(N^2)$ and the memory consumption in $O(N)$ (see, e.g., [12]). Most feature-based approaches therefore limit the number of features to a small N of, e.g., 4 [1].

To speed up base block search and limit memory consumption, the *N-Transform super-feature* (SF) approach has been proposed. N-Transform SF computes the first hash function using a rolling hash function [23] and then applies $N - 1$

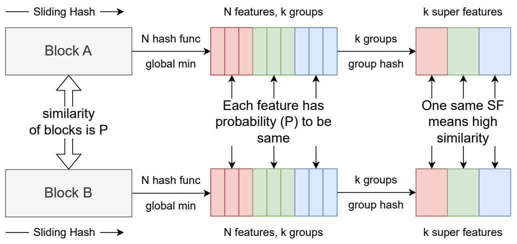
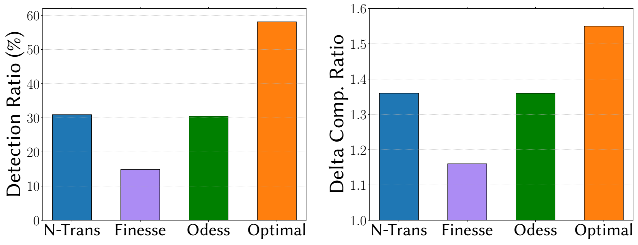


Figure 2. Similarity detection with super-features.



(a) Similarity detection **(b)** Delta-compression ratio

Figure 3. Detection and compression ratio for existing approaches and an optimal delta compression algorithm for DB_Stock dataset.

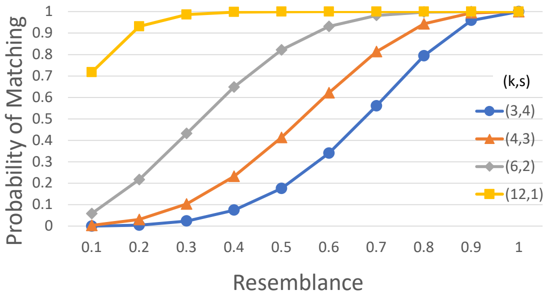


Figure 4. The impact of different groupings with 12 independent features. k means the number of super-features and s means the number of features grouped in each super-feature.

linear transformations to it [3, 4]. For each hash function, it takes its global minimum hash value as an additional feature. N-Transform SF then divides the N features into k groups of s features each (N can be divided by k) and calculates a joint hash value for each group to get k super-features. Two blocks are potential matches if they have one or more SFs in common. All implementations of *N-Transform SF* use a first fit approach to select a matching base block, because loading many potential matches for a new block from storage and comparing them with the new block is too costly. *N-Transform SF* is effective in detecting similar blocks for two reasons: 1) If two blocks have a SF in common, all features of that SF are identical between them, meaning that the blocks are most probably highly similar, and 2) multiple SFs are used to increase the probability to find highly similar blocks. Figure 2 shows the general workflow of using SFs to find similar blocks.

2.3 Motivation

Most delta-compression approaches for backup data rely on super-features generated for a single threshold value s. We therefore compared the similarity detection and compression ratio of the delta encoding stage of three such super-feature-based approaches with an optimal brute-force approach to understand how many similar blocks remain undetected by them. The brute-force algorithm delta-encodes each new block with all existing blocks to find the most similar base block. To decide whether the detected base block is “similar enough”, the optimal solution additionally uses Palantir’s false-positive filter, which will be discussed in Section 3.3.

Figure 3 shows the corresponding results for the DB_Stock dataset (see Table 2). The similarity detection ratio is defined as the ratio between the number of blocks, for which a similar base block can be found, and the total number of blocks. The compression ratio only depicts the compression achieved by the delta compression after the deduplication step.

The reason for the huge gap between the state-of-the-art and the brute-force approach is that a single threshold cannot reduce both the number of false positives and false negatives. A high threshold will miss many potentially similar blocks (false negatives), while a low threshold will select potentially dissimilar blocks as base blocks (false positives). In particular, the first-fit approach of current super-feature-based systems does not allow them to quantify and discriminate the degree of similarity between similar base block candidates for a new block, leaving them stuck in this dilemma.

This motivated us to develop a flexible similarity detection method with multiple thresholds. As indicated in previous work based on Broder’s theorem [4], we can control the similarity detection threshold by adjusting the number of features s grouped into each of the k super-features. Assuming that two blocks have a resemblance of p, the probability that two blocks share at least one common super-feature is

$$P(k,s) = \sum_{i=1}^{k} \binom{k}{i} p^{s*i}(1-p^s)^{k-i} \qquad (2)$$

Figure 4 shows the impact of different super-feature groupings for 12 features. The setting $(k, s) = (3, 4)$ used in previous work [25, 26, 34] can filter out most blocks with a resemblance of less than 30%, but only finds 35% of those blocks with a resemblance of 65%. An important observation from the figure is that we can greatly increase the detection capability while still maintaining a good accuracy if we combine and prioritize multiple thresholds. A high accuracy is achieved if we first try to match base blocks for, e.g., $(k, s) = (3, 4)$. If we do not find a matching base block for this threshold, we can increase the number of potential base blocks by increasing the threshold to $(k, s) = (4, 3)$ in a second step, or even to $(k, s) = (6, 2)$ in a third step. For example, for 65% similar blocks, this increases the detection ratio from 35% to over 90%.

This observation inspired the design of Palantir, which uses hierarchical super-features to match similar blocks. Palantir achieves a higher data reduction ratio than traditional super-feature approaches for two reasons: 1) it extends similarity detection to find more similar blocks by using multiple thresholds, and 2) it searches from the highest similarity tier to the following tiers to always find the expected best base block that maximizes the data reduction ratio.

3 Palantir Design

In this section, we first discuss the challenges of designing a delta-encoding framework that can achieve a very high compression ratio, and present our main ideas for overcoming them. In the following subsections, we describe how these ideas influenced the design of Palantir.

3.1 Challenges and Ideas

In order to optimize the post-deduplication compression ratio, Palantir must identify more potential base blocks than existing approaches, while still selecting the (expectedly) best possible one from among them. More base blocks can lead to more false positives, which can reduce the overall compression ratio and need to be filtered out. The hierarchical super-features also potentially increase the amount of metadata, and we present ideas for storing only relevant metadata using a metadata lifecycle manager.

How to find the best base block? Existing approaches use fixed (k, s) values and detect similar blocks using this single threshold. Figure 4 shows that the high threshold required by existing approaches misses the opportunity to detect many base blocks that are still similar enough to a new data block to benefit from delta compression.

Decreasing the similarity threshold increases the number of similar blocks detected, while the similarity of these base blocks to the new data block may vary significantly. Simply selecting a random base block from this set may select a base block with low similarity and may even decrease the overall compression ratio. Palantir overcomes this trade-off by introducing a set of tiered super-features with different detection thresholds and by matching base blocks tier by tier (see Section 3.2).

How to define and identify false-positives? Palantir's tiered super-features increase the similarity range of base blocks and the risk of selecting false positives, for which the compression ratio including delta encoding is even lower than when applying lossless compression alone.

Dataset	ZSTD Ratio	False-Positive Ratio
DB_Stock	2.75	8.68%
DB_Forex	2.68	10.44%
VSI_StackOverflow	3.41	7.83%
VDI_Server	1.80	4.78%
VDI_Ipod	1.12	6.41%
Linux	4.97	0.68%

Table 1. Lossless compression ratio when only deduplication and ZSTD compression are enabled, and the ratio of false positives to all base blocks found with tiered SFs.

Unfortunately, false-positive filtering cannot be defined solely on the basis of the similarity of blocks, because whether two blocks are "similar enough" to benefit from delta encoding also depends on the lossless compression ratio. For a dataset with a low compression ratio, delta encoding two blocks with low similarity might be beneficial. However, for a dataset with a higher compression ratio, the metadata overhead and reduced compressibility after delta encoding may decrease the overall compression.

Table 1 shows that the compressibility (using only deduplication and lossless compression) can vary significantly between different datasets. In general, we can see a correlation between compressibility and the number of false positives for the five backup datasets. An exception is the Linux code repository, which has a huge compressibility and most similar blocks are found by the first tier, so that it does not generate a significant amount of false positives.

In Section 3.3, we therefore discuss and define false positives based on the interaction between delta encoding and lossless compression and present a self-adaptive mechanism to detect false positives using the history of the lossless compression ratio.

How to control metadata space consumption? Existing delta compression approaches already add time and space to the backup system. The tiered super-features increase the amount of metadata even more. For example, three tiers of super-features increase the amount of metadata by more than three times because lower tiers of Palantir contain more super-features because each super-feature contains fewer raw features.

The main insight to reduce this metadata overhead is that older super-features for lower similarity detection thresholds are less likely to contribute positively to the compression ratio. Therefore, we introduce a generational lifecycle control mechanism within Palantir to identify and remove such obsolete super-features (see Section 3.4).

3.2 Similarity Matching with Hierarchical Feature Grouping

In this section, we introduce the similarity detection mechanism of Palantir, which is designed to detect more similar blocks and match the (expected) best base block among all candidates. The main idea of Palantir is inspired by insights from Broder's theorem: For a fixed number of features N, the similarity detection threshold of a super-feature depends on the parameters k and s, and different grouping parameters lead to different detection thresholds. Palantir exploits this insight by generating a hierarchy of multiple tiers of super-features.

Figure 5 illustrates how Palantir groups features hierarchically and then matches similar blocks. In each tier, Palantir uses a different (k, s) setting to generate super-features. Therefore, different tiers will naturally have different similarity detection thresholds. Palantir sorts these tiers in descending order of their thresholds, which means that Tier-1 super-features have the highest threshold.

The number of features for each tier i must be divisible by k_i to ensure that all super-features in the same tier contain the same number of raw features. Previous studies [4, 31, 34] show that generating 12 features is sufficient to build effective super-features. Therefore, by default, Palantir generates three tiers of super-features from $N = 12$ raw features. The highest tier sets the parameters (k, s) to $(3, 4)$, the second tier uses the setting $(k, s) = (4, 3)$, and the lowest tier sets $(k, s) = (6, 2)$. Thus, all features can be used in all layers.

With tiered super-features, Palantir can now prioritize base block matching. Palantir queries Tier-1 super-features in the database first. If no base block can be matched for Tier-1, it queries Tier-2 super-features and then proceeds to the following lower tiers. Once a super-feature is successfully matched, Palantir will use that result as a base block. Super-features in the same tier are matched according to the first-fit principle [12] to select the first record that has at least one super-feature in common with the new data block, since all super-features in a tier have the same detection threshold.

This ordered matching has two advantages: 1) Tiers with lower thresholds allow Palantir to find more base blocks. 2) Blocks with higher similarity are more likely to be matched by tiers with high thresholds, ensuring a high accuracy of the matching result.

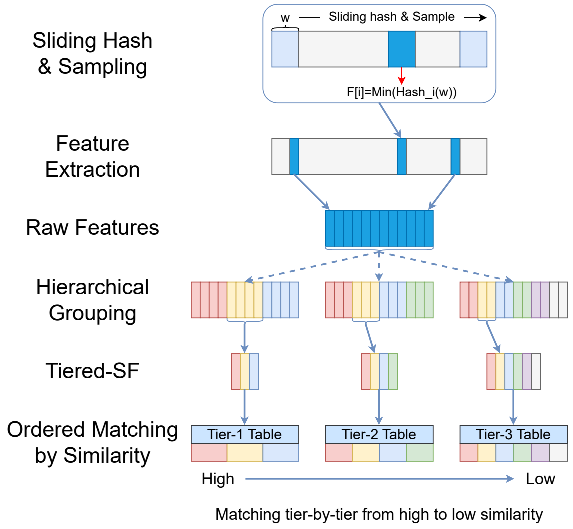


Figure 5. Hierarchical feature grouping and block matching.

Palantir does not store super-features of delta blocks in its SF tables, nor does it select such delta blocks as base blocks. The reason is that the resulting chained references would cause severe data fragmentation and require more I/O operations when writing/restoring delta blocks.

However, two challenges remain before Palantir can be successfully applied: First, there can be a lot of false positives introduced by the super-features of the lower tiers. Second, more super-features consume more space to store metadata. By default, Palantir generates 3/4/6 SFs for three tiers and 13 hash tables to store them, while the traditional solution needs only 3 hash tables. The following two sections show how Palantir solves these challenges.

3.3 Adaptive False Positive Filter

The objective of applying delta compression is to improve the overall compression ratio. Thus, it is natural to define a true positive if delta encoding a new block using a base block improves the overall compression ratio and to define a false positive case if it cannot.

This criterion is extremely costly to apply directly. Modern lossless compression algorithms typically require an input buffer to achieve a good compression ratio. This input buffer is much larger than a 4 or 8 KB data block, so each input buffer contains tens of data blocks. Directly testing whether delta compressing a block improves the overall compression ratio is therefore very inefficient, because it generally requires computing the lossless compression ratio for every combination of delta-compressed and non-delta-compressed blocks. This overhead does not even take into account that overflowing blocks between buffers can change the lossless compression ratio for subsequent buffers.

However, the large input buffers for lossless compression also have a dampening effect on the compression ratio, and the compressibility of a dataset typically does not change much in the case of small input changes. Additionally, commonly used delta encoding libraries such as xDelta [16] include a local lossless compression step to compress the delta block. As a result, it is difficult to compress the encoded delta block further in the subsequent lossless compression step.

The key idea of Palantir's false positive filter is to replace the exact comparison with an estimate by comparing the delta compression ratio of the current block (including the lossless compression of the delta encoding library) with that of the previous L lossless compression records. If this delta compression ratio is higher than the average value of those L records, we mark it as a true positive; otherwise as a false positive.

An advantage of the false positive filter is that we only need to compress the delta block, not the entire compression window, to identify false positives. Furthermore, we

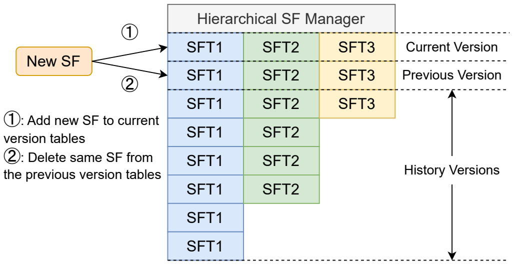


Figure 6. Palantir's metadata control mechanism. Each colored rectangle indicates a set of SF tables that stores all SFs of one tier from one version.

will show in Section 5.5 that this approach achieves a compression ratio which is very close to an optimal greedy false positive filter.

3.4 Version-based Lifecycle of Metadata

Tiered super-features create nearly four times more metadata per data block than a single tier (3,4) solution. Palantir therefore includes a metadata lifecycle mechanism that is based on two insights to optimize the metadata space consumption:

- Lower-tier SFs contribute less to the compression ratio because they have a lower detection threshold and therefore base blocks detected by them are expected to have less similarity to the original block. Conversely, they take up much more space than higher-tier super-features.
- The majority of similar blocks appear in newer versions because they are usually generated by minor modifications to existing data [35].

We want to reduce the metadata size with as little compression ratio loss as possible, so we focus on removing metadata from older versions of lower tiers. Palantir manages the metadata lifecycle with the same priorities as it manages the base block matching: Tier-1 SFs are most important and have the longest lifespan, while SFs in lower tiers are less important and have shorter lifespans.

The default setting of Palantir keeps Tier-1 SFs for all versions, Tier-2 SFs for 5 versions, and Tier-3 SFs for 2 versions. Compared to existing solutions with only one SF tier, the additional metadata of Palantir comes from the Tier-2 and Tier-3 SFs of the last five and two versions, respectively. This metadata overhead is a constant and does not increase as more backup versions are stored.

One remaining challenge is that duplicate blocks can create duplicate records in the SF tables. Traditional storage systems do not compute or store super-features for duplicated blocks because their super-features have surely been stored before. However, Palantir removes SFs of blocks after the end of the lifetime of the corresponding tier. If there is a more recent duplicate version of a block that belongs to removed metadata, Palantir must take measures to ensure that this information is not lost. Otherwise, it would be impossible to find the corresponding super-features and delta-encode against that block. Simply storing super-features for duplicates in every backup generation is not efficient, as this would significantly bloat the metadata tables.

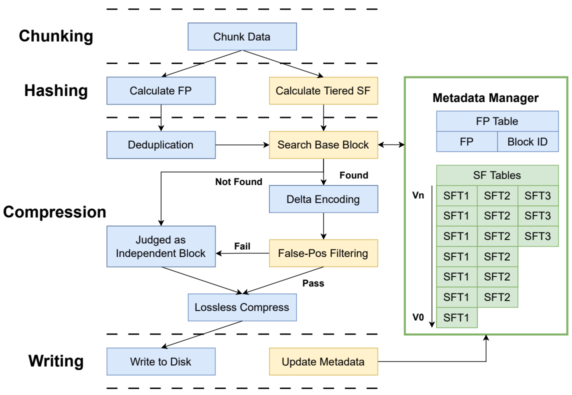


Figure 7. Workflow of Palantir.

Palantir therefore additionally computes super-features for duplicates and adds them to the most recent version T of the SF tables. Then it deletes this entry from version $T - 1$ of the SF tables. After a backup run finishes, Palantir directly removes outdated tables of super-features and creates a new table at the tail of the queue for the next version (see Figure 6).

Our profiling of the compression stage in the evaluation section will show that the overheads for computing more super-features than traditional solutions is small and can be hidden by the pipeline architecture.

4 System Implementation

In this section, we present the implementation of the Palantir prototype (see Figure 7). Our framework works as a four-stage pipeline. The blue rectangles in the Figure 7 represent operations that work identically in Palantir compared to traditional storage systems that provide deduplication, delta compression, and lossless compression. The yellow rectangles represent operations that are added or modified by Palantir. The metadata manager in Palantir differs from traditional solutions by having more hash tables to store tiered SFs. In the following, we introduce these stages:

Chunking: We use the FastCDC algorithm to chunk the input data stream into data blocks [30]. FastCDC is a content-defined chunking (CDC) algorithm, similar to Rabin fingerprinting, that computes chunking positions according to the data content. FastCDC requires fewer operations per byte than Rabin fingerprinting and can normalize the chunk size distribution, which simplifies later delta encoding.

Hashing: The hashing stage computes a strong 160-bit SHA1 FP for each data block to support data deduplication and 64-bit SFs to enable delta encoding. We compute SFs using the N-Transform algorithm [4] and apply an additional sampling approach introduced by Odess [34]. Setting the sampling ratio to our default value of 1/128 means that, on average, we only need to compute one linear transformation for every 128th byte to generate SFs. Previous studies have shown that this sampling ratio has almost no effect on the following delta compression ratio [34].

Compression: The third stage compresses data blocks by performing deduplication, delta compression, and lossless compression. Palantir first checks the FP of an incoming block in the fingerprint table. If it is already stored there, the data block is a duplicate and the system will only store a reference for it. Otherwise, the system tries to match a similar base block according to the block's SFs. If there is a similar block, it conducts delta encoding and then checks whether it is a false positive. False positives will be marked as independent blocks. Finally, the encoded delta block or the independent block will be passed to the ZSTD compression, which combines data blocks into 128 kByte segments.

Writing: The fourth stage writes the compressed data to the storage backend and updates the metadata tables. It will first add new FPs to the FP table and SFs into the SF table of the current backup version. It then removes the same SF records from the previous versions' SF table. When a complete backup version finishes, it additionally removes outdated SF tables.

5 Evaluation

In this section, we present the evaluation of Palantir using five backup datasets and a Linux source code repository. The evaluation includes a comparison with three state-of-the-art (SOTA) delta encoding approaches, and sensitivity studies on the impact of Palantir's components and the change rate between backups on compression ratio. We also compare Palantir's false positive filter with an optimal false positive filter and a heuristic, and profile the performance of Palantir. We further analyze the ability of our metadata lifecycle manager to reduce metadata overhead using an additional dataset of 100 backup versions. Our main findings are:

- Palantir increases the end-to-end compression ratio (including deduplication and lossless compression) on average by 7.3% over the best alternative approach, ranging from 1.1% to 12.0% for individual datasets.
- Palantir is able to find 27.4% more similar blocks than the next best approach and to increase the compression ratio of the delta encoding stage by 9.3%.
- The throughput loss compared to the fastest delta compression algorithm can be bounded by 7.7%.
- The detection rate and end-to-end compression ratio of our false-positive filter is close to that of an optimal greedy filter, successfully filtering out base blocks that degrade the overall compression ratio.
- The metadata lifecycle manager reduces the amount of additional metadata by 49% on average for all six datasets, and from 133% to 32% for the 100-version dataset. As the number of backup generations increases, the additional space converges to a constant overhead compared to SOTA.

Dataset	Size (GB)	Description
DB_Stock	5.211	Stock price database
DB_Forex	55.006	Forex database
VSI_StackOverflow (VSI_SOF)	200.375	Web pages of StackOverflow
VDI_server	318.193	Server images
VDI_ipod	1256.018	Ipod images
Linux	16.888	Linux kernel code 5.0.1 - 5.0.21

Table 2. Test datasets

5.1 Experimental setup

Testbed: All experiments were performed on a single socket Intel Xeon 8260 CPU server with 24 cores running at 2.4 GHz. The server had 512 GB of RAM and a 14 TB SSD disk, and was running Ubuntu 16.04.7 LTS. We used the FastCDC[30] algorithm to chunk the data, xDelta-3 [16] for delta encoding with the local compression level set to 1, and ZSTD-1.3.1 with level 10 for lossless compression.

Datasets: The evaluation contains results for five backup datasets and a Linux kernel source code repository (Table 2).

The initial versions of the backup datasets are based on data collected from a global cloud provider, including database images, websites, server images, and iPod images. We synthetically created 20 backup versions for each of these datasets by first dividing the datasets into 8 kB blocks. Then, for each block, we randomly selected whether to add a new block, modify the block, or delete the block. By default, we added new 8 kB blocks after 1% of the existing blocks. For each byte in a new block, we randomly selected one byte from the original block to maintain the same byte distribution between blocks. We modified 3.5% of the blocks by randomly choosing a modification length between (0, 50%) and then randomly choosing a starting position that ensured that all modifications could be made within the block. Again, all bytes in the section were replaced with a randomly selected byte from the original block. We also randomly deleted 0.5% of the blocks from the new record. Note that the changes to the 8 kB blocks regularly exceeded the boundaries of the chunks created by the content-defined FastCDC algorithm, which created blocks with an average size of 9.7 kB. The Linux kernel dataset contains 21 kernel versions from 5.0.1 to 5.0.21, and each version has been packed into an uncompressed .tar file.

Baseline setup: We have compared Palantir with the following three state-of-the-art delta-compression algorithms:

- The original **N-Transform SF** approach uses $N = 11$ linear transformations to the output of its rolling hash function to generate N independent hash functions and features [4]. Then it combines 12 features into 3 super-features. It has a high compression ratio but a low performance, as the linear transformations are computationally expensive [31].
- **Finesse** splits each data block into 12 sub-blocks and uses a hash function to generate 12 independent features from them [31]. Finess does not perform linear transformations and greatly improves the performance of the super-feature generation at the cost of a decreased compression rate [34].
- **Odess** first generates a small content-defined sample of the outputs of its rolling hash function and then computes the N-Transform algorithm on this sample [34]. It has the highest speed with a very minor compression ratio loss compared to the N-Transform SF approach.

To fairly compare the performance of these baseline algorithms with the performance of Palantir, we use our pipeline for all implementations and simply replace the similarity detection algorithm with the baseline algorithms. The parameters for all baseline algorithms have been set according to the settings in the corresponding papers, e.g., all of them generate 12 raw features and 3 super-features.

Evaluation metrics:

- The **end-to-end compression ratio (E2E ratio)** of the entire system includes deduplication, delta encoding, and lossless compression, and maximizing the E2E ratio is a primary goal of backup compression frameworks. When calculating the average E2E ratio, we normalize the E2E ratio to the Finesse ratio to provide a fair comparison and remove the effect of dataset variances.
- The **Delta Compression Coverage (DCC)** divides the number of similar blocks by the total number of blocks after the deduplication step and characterizes the capability of the similarity detection approach to detect base blocks.
- The **Delta Compression Ratio (DCR)** divides the data size before delta compression by the data size after delta compression. It measures the data saved in the delta compression stage and higher values therefore are better.
- The **Delta Compression Efficiency (DCE)** computes the average delta compression ratio over all similar blocks.
- **Throughput**: We measure the end-to-end throughput of the entire system to directly compare the efficiency of all algorithms when being integrated into a backup framework.

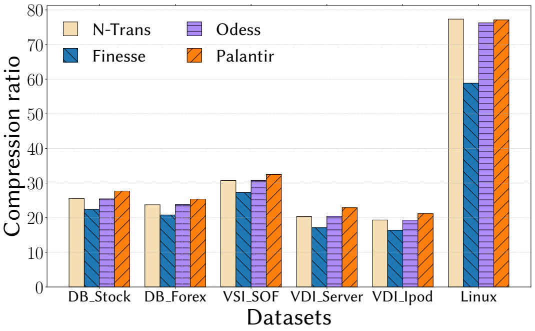


Figure 8. End-to-end compression ratios

DCC, DCR, and DCE have been used in previous works (e.g., [31] and [34]), whereas previous work typically have not investigated the E2E ratio and the throughput based on the complete backup pipeline.

5.2 Comparison with baseline approaches

This section uses the different datasets and our five metrics to compare Palantir with the three baseline algorithms. Table 3 shows the average results for all metrics for each algorithm, while the following figures also show the individual results over the different datasets. Palantir improves the average end-to-end compression ratio over all datasets by 7.3% compared to Odess and N-Transform and by 26.5% compared to Finesse. We also run the optimal brute-force algorithm from Section 2 on the DB_Stock dataset. Its E2E ratio is 17.8% higher than Palantir and 27.7% higher than Odess.

Figure 8 shows that Palantir improves the compression ratio compared to all baseline approaches for all cloud backup datasets. For the Linux dataset, which is extremely highly compressible, Palantir still slightly improves the compression ratio by 1.1% compared to Odess. The contribution of the tiered super-features and the false-positive filter varies between datasets and is further analyzed in Section 5.3.

The improved compression rate comes at the cost of a slightly reduced throughput. On average, Palantir is 7.7% slower than Odess, which seems acceptable in backup environments. The performance ranking varies depending on the dataset (see Figure 9), which can be explained by our pipeline structure. The baseline approaches compute features in the compression stage only for non-duplicate blocks, while

Algorithm	E2E ratio	DCC	DCR	DCE	Thrpt.
N-Transform SF	1.183	0.442	2.559	20.821	123.969
Finesse	1.000	0.290	1.973	28.691	165.998
Odess	1.180	0.446	2.544	20.596	177.947
Palantir	1.265	0.568	2.780	17.868	165.249

Table 3. Average performance over all datasets. The E2E ratio is normalized to Finesse and the throughput is in MB/s.

Palantir already computes features in the hashing stage for all blocks. Computing features in the hashing stage makes the pipeline more balanced at the cost of computing more features.

The computational overhead dominates performance for backup datasets with a high deduplication ratio, while for the Linux dataset, whose deduplication ratio is only 2.19, the more balanced pipeline of Palantir can increase throughput. Since data reduction is its primary goal, Palantir introduces more computational overhead to save space. In scenarios that focus more on performance, using the traditional pipeline structure may be a better choice.

We further analyze the impact of the pipeline on performance in Section 5.6 and show that the performance loss of Palantir can be overcome in future work by a better pipeline design and implementation, as the impact of the super-feature sampling itself on the overall runtime is low.

Figure 10 shows that Palantir finds many more base blocks than the baselines, on average 27.4% more than the next best approach (see also Table 3), demonstrating the effectiveness of Palantir's hierarchical feature grouping. The only slight exception is the Linux dataset, a code repository with an extremely high degree of redundancy for Tier-1 features (see also Figure 15). In this case, Palantir finds only 0.3% more similar blocks than Odess and N-Transform SF. The DCC ratios of Odess and N-Transform SF are nearly identical, while Finess misses many opportunities to find base blocks.

Finding more base blocks also leads to an improved delta compression ratio of Palantir of 9.3% compared to Odess and N-Transform SF (see Table 3). Figure 11 shows that the improvements are again consistent across almost all backup datasets. The results of this metric show that the base blocks additionally detected by Palantir are true positives and can improve the compression ratio.

Figure 12 depicts the average similarity of all similar blocks found by the different algorithms. Since Palantir introduces tiered super-features to increase the number of base blocks, it is natural that it finds more base blocks with comparatively low similarities. This result also indicates the importance of the false positive filter.

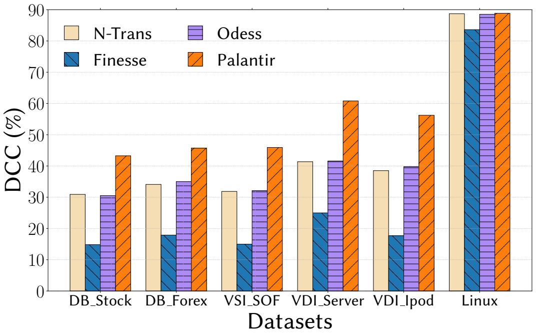


Figure 10. Delta Compression Coverage (DCC)

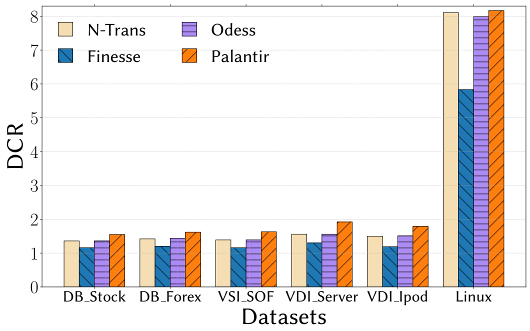


Figure 11. Delta Compression Ratio (DCR)

This section has shown that the improvements of Palantir depend on the dataset characteristics. These datasets are all realistic but represents different scenarios. For backup datasets, Palantir detects many similar base blocks missed by traditional methods, significantly improving the E2E ratio. For source code repository datasets such as the Linux dataset, which has a very high degree of redundancy for Tier-1 features, there are few such blocks, so they can also

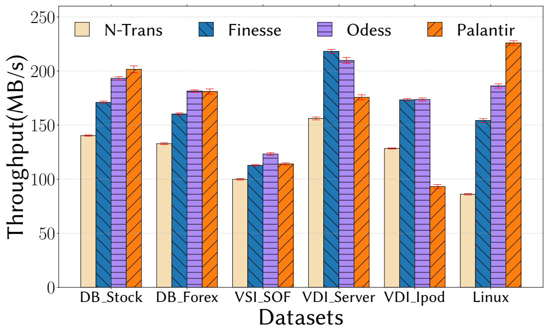


Figure 9. System throughput (MB/s)

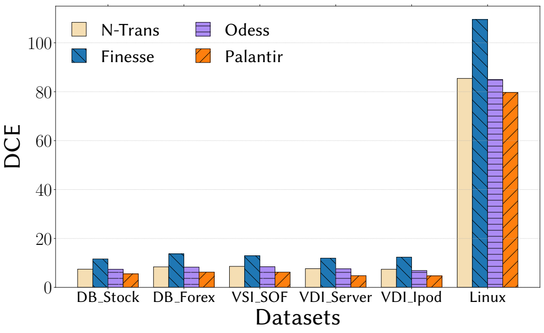


Figure 12. Delta Compression Efficiency (DCE)

be handled well by traditional methods such as Odess with higher performance.

5.3 Compression ratio contribution breakdown

This experiment is designed to understand the contribution of each technique: deduplication, lossless compression (ZSTD), Tier-1 delta compression, tiered-SF, and false positive filter in Palantir on the compression ratio. To quantify the individual contributions, we enable these techniques one at a time and use the difference between the settings as the contribution of a technique. Since the metadata lifecycle manager slightly reduces the compression ratio, we disable it to better understand the contribution of each technique.

Figure 13 shows the contribution of Tier-2 and Tier-3 super-features (summarized in Tiered-SF) and of the false positive filter in relation to deduplication, ZSTD compression, and standard delta compression. The compression ratios shown for each technique assume that all previous techniques are enabled. For datasets with a high degree of redundancy, such as *VSI_StackOverflow* and *Linux*, almost all of Palantir's improvement comes from the false positive filter, while the contribution of the tiered super-features is close to zero for them. The reason is that the final ZSTD compression achieves a very high compression ratio for these datasets, and performing delta-encoding between two blocks with low resemblance then contributes very little or even negatively to the end-to-end compression ratio. These false positives diminish the benefit of the true positives, leading to a zero contribution from tiered super-features alone.

Figures 14 and 15 show the proportion of similar blocks detected and data reduced by the three tiers after the false positive filter. Tier-2 and Tier-3 contribute about 50% of the similar blocks detected and reduce about 43% of the data in the delta compression stage on five backup datasets. The higher contribution of Tier-3 compared to Tier-2 can be explained by the integral under the curves in Figure 4. The figures also show that the true positives from Tier-2 and Tier-3 feature detection contribute significantly for the VSI_StackOverflow dataset, while for the Linux dataset most similar blocks are already detected after Tier-1 and the task of the false positive filter is to filter out base blocks with too little similarity.

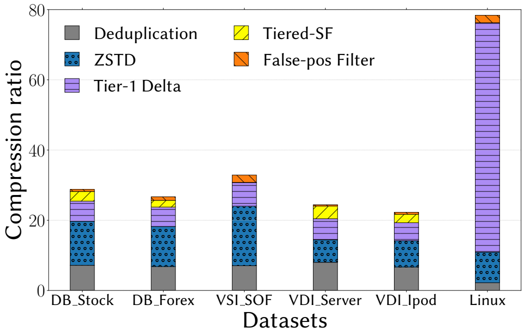


Figure 13. Impact of the techniques on compression ratio

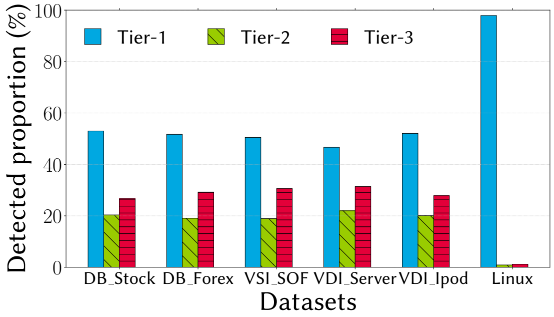


Figure 14. Fraction of base blocks detected by the three tiers

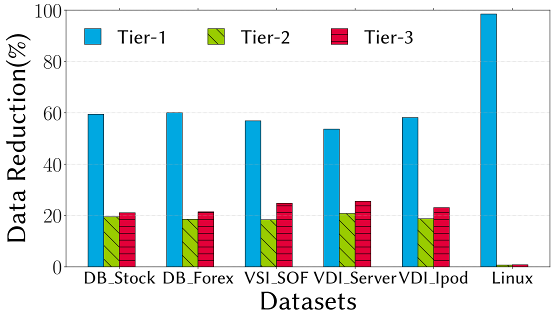


Figure 15. Contribution to data reduction by the three tiers

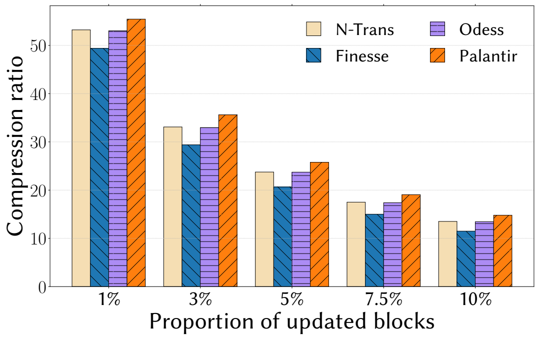


Figure 16. Compression ratio for different fractions of updated blocks

5.4 Different backup generation settings

In this section, we evaluate the sensitivity of Palantir to the modification ratio between different backup versions (see Section 5.1 for the default settings). We chose DB_Stock as the base version and generated 20 backup versions with different modification settings.

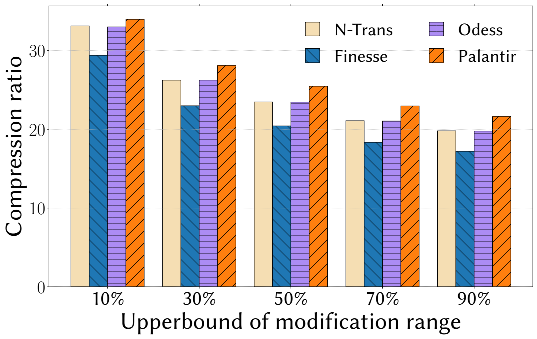


Figure 17. Compression ratio with different upperbound of modification section's length.

In the first experiment, we varied the proportion of blocks for which we trigger changes between 1% and 10%. For each backup version, we kept the relative proportions of additions, modifications and deletions the same as in the default setup. Figure 16 shows that the E2E compression ratio of Palantir is between 4.57 and 10.04% (average 8.22%) higher than that of Odess, and between 12.27 and 28.72% (average 22.78%) higher than that of Finesse. The advantage of Palantir increases as more blocks are updated between backup versions, since this naturally leads to more opportunities to find similar blocks.

In the second experiment we changed the parameters of the modification operation. The default modification operation randomly selected a length for each modification in [0, 50%] of the block length. In this experiment we changed the upper limit for the length of the modified segment to be between 10% and 90% of the block length. Figure 17 shows the E2E compression ratio for different modification ranges. Palantir gains 2.92 − 9.28% (average 7.39%) over Odess and 15.73 − 25.57% (average 22.70%) over Finesse.

The modified block is very similar to the original block when the modified part of a block is small, and these small modifications can often be detected by both Palantir and Odess. As the range increases, the modified blocks have less similarity to the original blocks and often cannot be detected by traditional methods, so the advantage of Palantir increases.

In conclusion, these two experiments outline the advantages of Palantir for different backup behaviors. Palantir outperforms traditional algorithms in all cases, while this gap depends on the similarity between backup versions.

5.5 Quality of the false positive filter

In this section, we compare our false positive filter with an optimal greedy approach that has no information about blocks written in the future, and with an algorithm that has limited knowledge about such upcoming blocks.

We start with two constraints on the capabilities of an optimal algorithm A_{opt}. First, we want to compare it to an optimal streaming algorithm, so we require that A_{opt} must be greedy in deciding whether a new block is a false positive or not, without having any information about subsequent blocks. Second, the assignment of data blocks to ZSTD compression segments does not differ between algorithms. The reason for the second restriction is that we are focusing on the local properties of the false positive filter, and moving a single data block between two segments can theoretically completely change the compression ratio of all subsequent segments.

We now assign m data blocks to a ZSTD compression segment. The uncompressed content of a block $i \in \{0, ..., m-1\}$ is called b_i and the delta encoded content of the same block is called δ_i. If there is no similar base block for block i, then $b_i = \delta_i$. The task of a false positive filter A is now to add $a_i \in \{b_i, \delta_i\}$ to the ZSTD segment. The input for an algorithm is the new block b_i with $0 \leq i \leq (m-1)$ and the representations of the previous blocks $\{a_0, \ldots, a_{i-1}\}$.

Algorithm A is now defined to be optimal under these constraints if for all other algorithms $\tilde{A}$ under the same constraints it holds that if the encoding for $\{a_0, ..., a_{j-1}\}$ is the same for A and $\tilde{A}$, but the decision for b_j is different for $0 \leq j \leq (m-1)$, then there exists a set of blocks $\{b_{j+1}, ..., ..., b_{m-1}\}$ for which the compression ratio $ZSTD(a_0, ..., a_{m-1})$ of A_{opt} is higher than $ZSTD(\tilde{a}_0, ..., \tilde{a}_{m-1})$ of $\tilde{A}$[1]. An optimal greedy algorithm does not have to produce better results than competing algorithms for all sequences of input blocks, but if its decision differs from a competing algorithm, then there must be at least one sequence of subsequent blocks for which A achieves the same or a better compression ratio.

Optimal greedy algorithm A^*: We will show that the following algorithm A^* is optimal: For each new input block b_i, A^* computes the compression ratio $ZSTD(a^*_0, ..., a^*_{i-1}, b_i)$ and $ZSTD(a^*_0, ..., a^*_{i-1}, \delta_i)$ and then simply selects the representation which produces the higher compression ratio.

For the proof, we select the smallest j for which the representation of A^* and another algorithm $\tilde{A}$ differ. In this case, the compression ratio of A^* must be better than or equal to (by definition) the compression ratio of $\tilde{A}$ for the set of blocks $\{b_0, \ldots, \ldots, b_j\}$. We then simply select a set of blocks $\{b_{j+1}, \ldots, \ldots, b_{m-1}\}$ as subsequent blocks, which are completely random and which cannot be delta encoded or compressed. In this case, the compression ratio of A^* remains better than the compression ratio of $\tilde{A}$.

Segment Future: For comparison, we also include the Segment Future Algorithm S that knows the complete ZSTD segment before making its decisions. This algorithm first fills the ZSTD segment with the m delta blocks by setting $s_i = \delta_i$ for all $i \in \{0, m-1\}$. Then the algorithm iterates once through all the delta blocks and checks if replacing δ_i with

[1] For $i = 0$ and $j = m - 1$, $\{a_0, a_{-1}\}$ and $\{a_m, a_{m-1}\}$ both represent the empty set.

Strategy	False positive count	End-to-end comp. ratio	Throughput (MB/s)
Palantir	165,946	32.490	131.519
Optimal Greedy	189,648	32.819	20.468
Segment Future	197,088	32.921	9.720

Table 4. Comparison between the Palantir heuristic and an optimal greedy false positive filter on VSI_StackOverflow dataset.

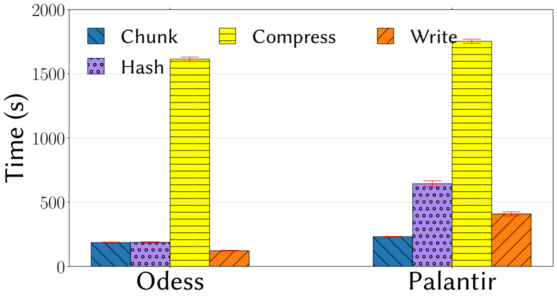


Figure 18. Runtime of the pipeline stages for VSI_StackOverflow

b_i results in a higher compression ratio. If this is true, δ_i is considered a false positive and is greedily replaced by b_i.

Table 4 compares the false positive filter in Palantir with the optimal strategy and the segment future approach. We chose the VSI_StackOverflow dataset for this comparison because the false positive filter has the most impact on it. The results show that the optimal false positive filter detects slightly more false positives and also slightly increases the end-to-end compression ratio by 1.0%. However, this comes at the cost of an 84.4% decrease in throughput, showing that the assumptions behind our false positive filter in Palantir are valid. Segment Future can further increase the E2E compression ratio slightly at the cost of even lower performance.

5.6 Performance profiling

Figure 18 shows the time taken by different pipeline stages in Odess and Palantir for the VSI_StackOverflow dataset. Palantir introduces overhead when calculating super-features, and as a result the hashing stage takes 3.5× as long as in Odess. In addition, the writing stage takes 3.5× as long as in Odess due to the additional metadata update mechanism. However, this overhead can be hidden by the pipeline structure and does not affect throughput.

Figure 19 breaks down the time spent in the compression phase for Odess and Palantir, which is the bottleneck of the system. Palantir spends more time on delta encoding because it finds more similar blocks. However, the time spent on ZSTD compression is 13.9% shorter than in Odess, because Palantir eliminates more data in the delta compression stage and submits less data to ZSTD compression, which is much slower than delta encoding. This result shows that putting more effort into delta compression can have two benefits: First, it can increase the compression ratio. Second, it can reduce the amount of data subjected to the subsequent, less efficient lossless compression, thus saving time.

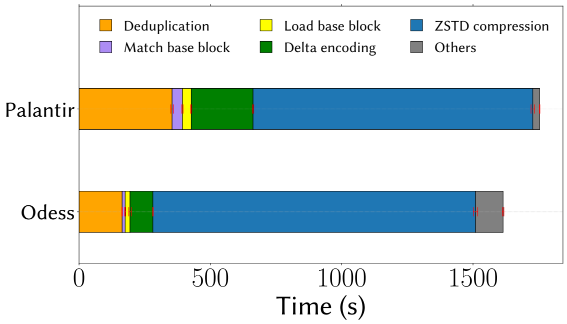


Figure 19. Breakdown of the compression stage for VSI_StackOverflow. For Odess, the Others part also includes feature calculation after Deduplication.

Another finding is that Palantir spends much more time on deduplication than Odess. This is because the performance of the Palantir deduplication module is sensitive to resource contention (both compute and memory). Unfortunately, Palantir introduces additional compute overhead and increases memory pressure to compute the features of all blocks and update metadata. It degrades the performance of the deduplication module and slows down the entire system. This negative impact increases with larger datasets.

In summary, the pipeline structure already helps to hide the overhead of additional feature computation in Palantir, and the time spent on additional delta encoding can be partially saved in the lossless compression procedure. The profiling results also point to two future optimizations. The first is to make the pipeline more balanced by moving the ZSTD compression to a new stage. The second is to optimize the deduplication module to make it less sensitive to resource contention.

Dataset	Palantir w/o metadata mana.	Palantir w/ metadata mana.	Overhead reduction in %
DB_Stock	3.002	1.946	52.75%
DB_Forex	2.971	1.988	49.87%
VSI_SOF	2.890	1.922	51.21%
VDI_Server	2.476	1.815	44.81%
VDI_Ipod	2.541	1.730	52.60%
Linux	3.736	2.562	42.90%

Table 5. Size of SF tables (normalized to Odess) of Palantir without and with metadata lifecycle management

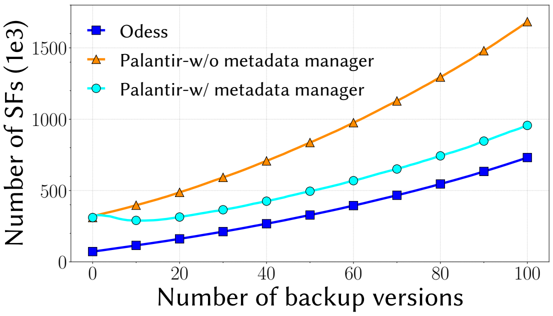


Figure 20. Number of SFs for increasing number of backups.

5.7 Metadata size measurement

This experiment is designed to measure metadata space consumption and evaluate the effectiveness of Palantir's metadata lifecycle control mechanism. In addition to the six datasets used in previous experiments, we also created a backup dataset of 100 versions based on the DB_Stock dataset using the method from Section 5.1.

Table 5 shows the size of SF tables for the six datasets compared to the size required by Odess. The optimized metadata manager successfully reduced 42.90-52.75% (average 49.02%) of extra metadata introduced by tiered super-features. The metadata overhead on the Linux dataset is relatively high because this dataset has an extremely high degree of redundancy, and Palantir only finds 0.4% more similar blocks than Odess. In other datasets, Palantir finds much more similar blocks, and those similar blocks will not generate SF records, which then reduces the size of SF tables.

Figure 20 shows the number of SF records for an increasing number of backup versions for the DB_Stock dataset. Palantir without metadata lifecycle manager generates 133.3% extra super-features. Palantir with metadata manager can effectively eliminate valueless metadata and only produces 32.4% extra metadata. This extra space consumption converges to a near-constant level within 10 backup versions. This result proves that the metadata lifecycle control mechanism successfully solves the metadata explosion problem.

The metadata space of Palantir with and without optimization are both smaller than we expected in Section 3.1. The reason is that Palantir finds 40.8% more similar blocks than Odess on backup datasets (see Figure 10). These similar blocks generate SF records in the Odess system, but not in Palantir. Therefore, improving delta coverage also alleviates the metadata explosion problem.

6 Related Works

Palantir builds on previous work on deduplication and delta compression. Deduplication is a widely used technique that uses fingerprints to eliminate duplicate blocks [7, 10, 14, 19, 27, 32]. Since a small change can change the data chunking point and affect all following blocks, content-defined chunking (CDC) was developed to realign data streams [20, 30].

Delta compression is able to compress similar but not identical blocks. The main approach to find similar blocks is the N-Transform SF approach [4] based on Broder's theorem [3]. Finesse [31] uses the min hash of different regions to replace the global min hash of different hash functions and speed up computation. Odess [34] optimizes SF generation by content-defined-sampling to generate a small sample from the original block and then perform the N-Transform SF algorithm only on the small sample.

Aronovich et al. optimized the selection of features to overcome skewed distributions. They first find the top four hash values, then move a few positions in the block and use the corresponding hashes as features. From the possible set of candidate blocks, they then select the best matching block [1]. MeGA and Dare use data locality to optimize the system. MeGA [35] assumes that most similar block pairs occur in adjacent versions. Therefore, it limits the SF search range to blocks that are related to the current and previous versions. Dare [29] also takes advantage of the fact that modified blocks in a backup system may be very similar to their previous versions. It then quickly identifies all blocks adjacent to duplicated blocks as possible similar blocks.

Several approaches have focused on applying delta compression to distributed scenarios. Shilane et al. proposed to apply delta compression to wide area networks (WAN) [25]. AA-Dedup [9] performs source deduplication by clustering incoming data by application type. EF-Deduplication [13] proposes a cluster grouping mechanism to facilitate the deployment of delta compression on edge devices.

DeepSketch [21] uses machine learning for delta compression. It trains a CNN model to generate a sketch from a data block and uses an approximate nearest neighbor (ANN) engine to find the block with the closest sketch as the base block.

Migratory Compression (MC) groups similar data together and then uses lossless compression to compress the entire group. MC [15] therefore uses super-features to identify similar blocks on a GB-level range and migrates them together before applying lossless compression. MC was inspired by the Burrows-Wheeler transformation [5], which detects and groups similar data on a small KB-level range.

7 Conclusion

In this paper, we proposed Palantir to improve similarity detection and compression ratio for delta compression. Palantir groups raw features into multiple tiers of SFs with different matching thresholds, which allows Palantir to find more similar blocks while matching highly similar blocks first. We proposed a self-adaptive false positive filter using the average lossless compression ratio as a threshold, which improves the robustness of the system against false positives.

We also designed a metadata lifecycle control mechanism to control the metadata overhead to a constant level.

Using six different datasets, we evaluated Palantir and compared it to three other baseline works. The results show that Palantir finds 27.4% more similar blocks and improves the end-to-end compression ratio by 7.3% at the cost of a 7.7% reduction in system throughput. Palantir also achieves a stable compression ratio gain for different change rates among backup versions. The metadata lifecycle manger reduces the metadata overhead by 49% and this overhead converges to a constant value for more backup versions. In the future, we will further optimize the pipeline design to improve throughput and explore new caching mechanisms for large datasets. We will also explore an automatic approach to decide whether tiered super-feature should be enabled based on the contribution to data reduction by each tier.

8 Acknowledgements

This work is supported in part by the Research Grants Council of Hong Kong (GRF 11209520, CRF C7004-22G). We would like to thank our anonymous reviewers and our shepherd Dany Harnik for their valuable feedback and guidance.

References

[1] Lior Aronovich, Ron Asher, Eitan Bachmat, Haim Bitner, Michael Hirsch, and Shmuel T. Klein. The design of a similarity based deduplication system. In *Proceedings of the Israeli Experimental Systems Conference (SYSTOR), Haifa, Israel, May 4-6,*, page 6, 2009.

[2] John Black. Compare-by-hash: A reasoned analysis. In *USENIX Annual Technical Conference (ATC), Boston, MA, USA, May 30 - June 3*, pages 85–90, 2006.

[3] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)*, pages 21–29, 1997.

[4] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *11th Annual Symposium on Combinatorial Pattern Matching (CPM), Montreal, Canada, June 21-23*, pages 1–10, 2000.

[5] Michael Burrows. A block-sorting lossless data compression algorithm. *SRC Research Report, 124*, 1994.

[6] Fred Douglis and Arun Iyengar. Application-specific delta-encoding via resemblance detection. In *Proceedings of the USENIX Annual Technical Conference (ATC), June 9-14, San Antonio, Texas, USA*, pages 113–126. USENIX, 2003.

[7] Cezary Dubnicki, Leszek Gryz, Lukasz Heldt, Michal Kaczmarczyk, Wojciech Kilian, Przemyslaw Strzelczak, Jerzy Szczepkowski, Cristian Ungureanu, and Michal Welnicki. Hydrastor: A scalable secondary storage. In *7th USENIX Conference on File and Storage Technologies (FAST), February 24-27, 2009, San Francisco, CA, USA*, pages 197–210, 2009.

[8] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Ottean, Jin Li, and Sudipta Sengupta. Primary data deduplication - large scale study and system design. In *USENIX Annual Technical Conference (ATC), Boston, MA, USA, June 13-15*, pages 285–296, 2012.

[9] Yinjin Fu, Hong Jiang, Nong Xiao, Lei Tian, and Fang Liu. Aa-dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment. In *2011 IEEE International Conference on Cluster Computing (CLUSTER), Austin, TX, USA, September 26-30*, pages 112–120, 2011.

[10] Jürgen Kaiser, André Brinkmann, Tim Süß, and Dirk Meister. Deriving and comparing deduplication techniques using a model-based classification. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys), Bordeaux, France, April 21-24*, pages 11:1–11:13, 2015.

[11] Byron Knoll and Nando de Freitas. A machine learning perspective on predictive coding with PAQ8. In *22nd Data Compression Conference (DCC), Snowbird, UT, USA, April 10-12*, pages 377–386, 2012.

[12] Purushottam Kulkarni, Fred Douglis, Jason D. LaVoie, and John M. Tracey. Redundancy elimination within large collections of files. In *USENIX Annual Technical Conference (ATC), June 27 - July 2, 2004, Boston Marriott Copley Place, Boston, MA, USA*, pages 59–72, 2004.

[13] Shijing Li, Tian Lan, Bharath Balasubramanian, Moo-Ryong Ra, Hee Won Lee, and Rajesh K. Panta. Ef-dedup: Enabling collaborative data deduplication at the network edge. In *39th IEEE International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, July 7-10*, pages 986–996, 2019.

[14] Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezis, and Peter Camble. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *7th USENIX Conference on File and Storage Technologies (FAST), February 24-27, 2009, San Francisco, CA, USA*, pages 111–123, 2009.

[15] Xing Lin, Guanlin Lu, Fred Douglis, Philip Shilane, and Grant Wallace. Migratory compression: coarse-grained data reordering to improve compressibility. In *12th USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, February 17-20*, pages 257–271, 2014.

[16] Joshua P. MacDonald. File system support for delta compression. http://www.xmailserver.com/xdfs.pdf, 2000.

[17] Dirk Meister and André Brinkmann. Multi-level comparison of data deduplication in a backup scenario. In *Israeli Experimental Systems Conference (SYSTOR), Haifa, Israel, May 4-6*, 2009.

[18] Dirk Meister, André Brinkmann, and Tim Süß. File recipe compression in data deduplication systems. In *11th USENIX conference on File and Storage Technologies (FAST), San Jose, CA, USA, February 12-15*, pages 175–182, 2013.

[19] Dutch T. Meyer and William J. Bolosky. A study of practical deduplication. *ACM Transactions on Storage (ToS)*, 7(4):14:1–14:20, 2012.

[20] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A low-bandwidth network file system. In *18th ACM Symposium on Operating System Principles (SOSP), Chateau Lake Louise, Banff, Alberta, Canada, October 21-24*, pages 174–187, 2001.

[21] Jisung Park, Jeonggyun Kim, Yeseong Kim, Sungjin Lee, and Onur Mutlu. Deepsketch: A new machine learning-based reference search technique for post-deduplication delta compression. In *20th USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, February 22-24*, pages 247–264, 2022.

[22] Calicrates Policroniades and Ian Pratt. Alternatives for detecting redundancy in storage systems data. In *USENIX Annual Technical Conference (ATC), June 27 - July 2, 2004, Boston Marriott Copley Place, Boston, MA, USA*, pages 73–86, 2004.

[23] Michael O Rabin. Fingerprinting by random polynomials. Technical Report Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[24] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423, 1948.

[25] Philip Shilane, Mark Huang, Grant Wallace, and Windsor Hsu. Wan-optimized replication of backup datasets using stream-informed delta compression. *ACM Transactions on Storage (ToS)*, 8(4):13:1–13:26, 2012.

[26] Philip Shilane, Grant Wallace, Mark Huang, and Windsor Hsu. Delta compressed and deduplicated storage using stream-informed locality. In *4th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage), Boston, MA, USA, June 13-14*, 2012.

[27] Kiran Srinivasan, Timothy Bisson, Garth R. Goodson, and Kaladhar Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *10th USENIX Conference on File and Storage Technologies (FAST), San Jose, CA, USA, February 14-17*, page 24, 2012.

[28] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. Characteristics of backup workloads in production systems. In *10th USENIX Conference on File and Storage Technologies (FAST), San Jose, CA, USA, February 14-17*, 2012.

[29] Wen Xia, Hong Jiang, Dan Feng, and Lei Tian. DARE: A deduplication-aware resemblance detection and elimination scheme for data reduction with low overheads. *IEEE Transactions on Computers*, 65(6):1692–1705, 2016.

[30] Wen Xia, Yukun Zhou, Hong Jiang, Dan Feng, Yu Hua, Yuchong Hu, Liu, and Yucheng Zhang. Fastcdc: a fast and efficient content-defined chunking approach for data deduplication. In *USENIX Annual Technical Conference (ATC), Denver, CO, USA, June 22-24*, pages 101–114, 2016.

[31] Yucheng Zhang, Wen Xia, Dan Feng, Hong Jiang, Yu Hua, and Qiang Wang. Finesse: Fine-grained feature locality based fast resemblance detection for post-deduplication delta compression. In *17th USENIX Conference on File and Storage Technologies (FAST), Boston, MA, February 25-28*, pages 121–128, 2019.

[32] Benjamin Zhu, Kai Li, and R. Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *6th USENIX Conference on File and Storage Technologies (FAST), February 26-29, San Jose, CA, USA*, pages 269–282, 2008.

[33] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[34] Xiangyu Zou, Cai Deng, Wen Xia, Philip Shilane, Haoliang Tan, Haijun Zhang, and Xuan Wang. Odess: Speeding up resemblance detection for redundancy elimination by fast content-defined sampling. In *37th IEEE International Conference on Data Engineering (ICDE), Chania, Greece, April 19-22*, pages 480–491, 2021.

[35] Xiangyu Zou, Wen Xia, Philip Shilane, Haijun Zhang, and Xuan Wang. Building a high-performance fine-grained deduplication framework for backup storage with high deduplication ratio. In *USENIX Annual Technical Conference (ATC), Carlsbad, CA, USA, July 11-13*, pages 19–36, 2022.