Finesse: Fine-Grained Feature Locality based Fast Resemblance Detection for Post-Deduplication Delta Compression

Yucheng Zhang[†], *Wen Xia*[‡,⋆], *Dan Feng*[§,⋆], *Hong Jiang*[¶], *Yu Hua*[§], *Qiang Wang*[§]

[†]*Hubei University of Technology* [‡]*Harbin Institute of Technology, Shenzhen & Peng Cheng Laboratory*

[§]*WNLO, School of Computer, Huazhong University of Science and Technology* [¶]*University of Texas at Arlington*

[⋆]*Corresponding authors: xiawen@hit.edu.cn and dfeng@hust.edu.cn*

Abstract

In storage systems, delta compression is often used as a complementary data reduction technique for data deduplication because it is able to eliminate redundancy among the non-duplicate but highly similar chunks. Currently, what we call *'N-transform Super-Feature' (N-transform SF)* is the most popular and widely used approach to computing data similarity for detecting delta compression candidates. But our observations suggest that the *N-transform SF* is compute-intensive: it needs to *linearly transform each Rabin fingerprint of the data chunks N times to obtain N features*, and can be simplified by exploiting the fine-grained feature locality existing among highly similar chunks to *eliminate time-consuming linear transformations*. Therefore, we propose *Finesse*, a fine-grained feature-locality-based fast resemblance detection approach that divides each chunk into several fixed-sized subchunks, computes features from these subchunks individually, and then groups the features into super-features. Experimental results show that, compared with the state-of-the-art *N-transform SF* approach, *Finesse* accelerates the similarity computation for resemblance detection by 3.2×∼3.5× and increases the final throughput of a deduplicated and delta compressed prototype system by 41%∼85%, while achieving comparable compression ratios.

1 Introduction

Data deduplication, a popular data reduction technique, usually identifies duplicate data at the chunk level (e.g., 8KB size) by using secure fingerprints (e.g., SHA1) to uniquely and globally represent data chunks in storage systems [34, 44]. Hence, deduplication-based storage systems only store one physical instance referred to by any other duplicates, which helps improve storage space efficiency [18, 25, 44] or network bandwidth efficiency [26, 29].

Recently, delta compression has also gained increasing attention due to its ability to eliminate data redundancy among non-duplicate but highly similar chunks, which can be used

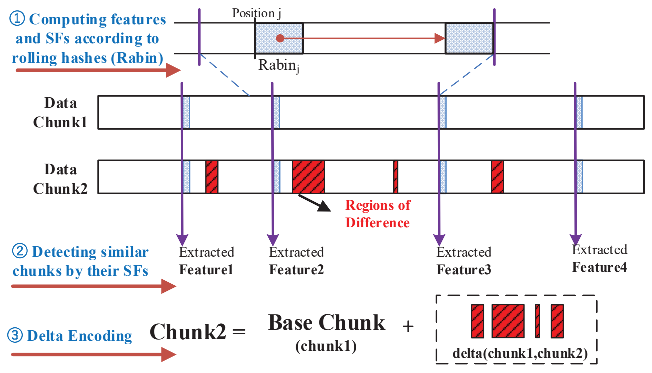


Figure 1: An example of delta compression on two similar chunks with the three typical steps: ① computing similarity, ② indexing, and ③ delta encoding.

post-deduplication as a complementary technique to further eliminate redundancy. For example, if chunk A_2 is similar to chunk A_1 (the base chunk), the delta compression approach only stores or transfers the differences (*delta*) and the mapping relation between A_2 and A_1, removing the redundant data to improve storage space efficiency [21, 30, 36, 37, 41] or network bandwidth efficiency [8, 29, 42, 43]. Several studies [29, 30, 37, 38] suggest that delta compression is able to achieve about 2× additional compression ratio beyond deduplication and local compression in backup storage workloads.

For delta compression in deduplication-based storage systems, resemblance detection is the first key step in its workflow, which identifies delta compression candidates. This is because a higher similarity degree in the detected chunks implies more space savings from delta compression. Currently, the most commonly used chunk-level resemblance detection approach computes the 'super-features' (***SF*** for short) [5, 17, 29] based on the Rabin fingerprints [28] of data contents, to detect highly similar chunks. Figure 1 gives an example of the general workflow for this *SF*-based delta compression approach: ① computing the similarity of chunks, namely, computing features and grouping features into *SFs* (detailed in Section 3), ② detecting similar chunks according to their *SFs* (any two chunks having a *SF* in common are considered highly similar [6]), ③ delta encoding the two similar chunks, i.e., calculating their differences, also called 'delta'. For decompression, the input chunk is recovered by decoding the 'delta' with the base chunk.

To achieve high delta compression efficiency, some recent works on delta compression [17, 19, 29] recommend grouping four or more features into one *SF* to reduce false positives in resemblance detection, and using three or more *SFs* to detect more highly similar chunks for delta compression. But according to observations in our delta compressed prototype system, computing the similarity of data chunks, namely, generating their *SFs*, is quite time-consuming. Specifically, to ensure high similarity detection efficiency, Rabin fingerprints are calculated byte-by-byte on data chunks (similar to Content-Defined Chunking [26, 40, 43]), and are each then linearly transformed N times to calculate N-dimensional hash value sets. Finally the N maximal values, one from each of the N dimensions, are selected as features. Thus, the traditional *SF* approach needs to linearly transform each Rabin fingerprint of data chunks N times, which we refer to as ***'N-transform SF'*** to distinguish it from our approach in the remainder of the paper.

Consistent with the backup stream locality observed by many studies on deduplication [13, 16, 18, 22, 33, 35, 44], we observe that there also exists fine-grained locality among similar chunks. This locality refers to the fact that the corresponding subregions (subchunks) of chunks and their features also appear in the same order among the similar chunks with a very high probability, which is referred to as *feature locality* in this paper. Based on this key observation, we argue that a collection of features, exactly one extracted from each subchunk of a chunk, can also be used for representing the similarity of a chunk for generating *SFs*, which is much less compute-intensive than the *N-transform SF* since it eliminates the time-consuming linear transformations.

In this paper, we propose *Finesse*, a fast resemblance-detection approach that exploits the fine-grained feature locality of similar chunks. Specifically, *Finesse* simplifies computing the similarity by first dividing each chunk into several subchunks and then quickly computing features from each subchunk, finally grouping these features into *SFs*. Experimental results based on six datasets show that, compared with the baseline *N-transform SF* approach, *Finesse* accelerates the similarity computation by $3.2\times \sim 3.5\times$ and increases the throughput of a delta compression prototype system by $41\% \sim 85\%$, while achieving comparable and even higher compression ratios.

2 Background and Related Work

Data reduction has gained increasing attention and popularity in storage and file-transfer systems due to the explosive growth of digital data. Compared with local compression (e.g., LZ [34]), data deduplication is able to identify and eliminate redundancy globally at at a much larger granularity (i.e., chunk- or file-level) in large-scale storage systems. Thus it is widely studied and used in large-scale backup storage [18, 29, 33, 44], primary storage [10, 24, 31], and HPC storage [23].

Meanwhile, delta compression, another data reduction technique that removes redundancy among non-duplicate but highly similar chunks, is able to help maximize the compression ratio when combined with deduplication and local compression in backup storage [30], storage replication [29], database storage [41], etc. Shilane et al. [29, 30] suggest that delta compression can achieve an additional $2\times$ compression ratio beyond data deduplication in their production backup storage systems. Similar results are also observed in other scenarios, such as, database storage [41, 42] and migratory compression [19].

While greatly improving storage efficiency, delta compression also introduces extra compute and I/O overheads. SIDC [29] suggests that the issue of on-disk large-sized similarity indexing faced by delta compression can be addressed by exploiting (caching) backup stream locality, in a way similar to data deduplication systems [44]. Ddelta [38] and Edelta [39] have been proposed to accelerate the delta encoding process by using the idea of CDC-based deduplication and exploiting fine-grained locality of the backup data streams.

3 Super-Feature based Approach

Resemblance detection is the first step needed for delta compression to compute the similarity of data chunks and find compression candidates. As mentioned earlier, the ***'N-transform SF'*** approach is currently the most popular method for chunk-level resemblance detection. It was first proposed by Broder [6] and is based on "Broder's theorem" [5], which evaluates the resemblance between two sets, as detailed below:

Theorem 1 *Consider two sets A and B, with H(A) and H(B) being the corresponding sets of the hashes of the elements of A and B respectively, where H is chosen uniformly and randomly from a min-wise independent family of permutations [2, 7]. An element in the set is mapped to an integer. Let min(S) denote the smallest element of the set of integers S. Then:*

$$\mathrm{Pr}[\min(\mathrm{H(A)})) = \min(\mathrm{H(B)})] = \frac{|\mathrm{A} \cap \mathrm{B}|}{|\mathrm{A} \cup \mathrm{B}|}.$$

Broder's theorem states that the probability of the two sets A and B having the same minimum hash element is the same as their Jaccard similarity coefficient [14]. Based on this theorem, Broder proposed a resemblance detection approach called super-features [6, 29] that extracts a fixed number of features from a chunk. Specifically, this *SF*-based

Algorithm 1 Extracting features in *N-transform SF*.

```
Require: chunk content, Str; length of the chunk, L; randomly
    value pair for linear transformation, $m_i$ and $a_i$;
Ensure: N features, Feature[N];
function FEATURE-EXTRACT-N-TRANSFORM_SF(Str, L)
    $\text{Feature}[0, \cdots, N-1] \leftarrow 0$;
    for $m = 0$ to $L-1$ do
        $\text{FP} \leftarrow \text{RabinFunction}(\text{Str}, m)$;
        for $i = 0$ to $N-1$ do
            $\text{Transform}[i] \leftarrow (m_i * \text{FP} + a_i) \bmod 2^{32}$;
            if $\text{Feature}[i] \leq \text{Transform}[i]$ then
                $\text{Feature}[i] \leftarrow \text{Transform}[i]$;
            end if
        end for
    end for
end function
```

approach [29] (referred to as *N-transform SF* in this paper) computes data similarity by extracting features from Rabin fingerprints (a rolling hash algorithm [28]) and then grouping these features into *SFs* to detect resemblance for data reduction. For example, Feature_i of a chunk (length = L), is uniquely generated with a randomly pre-defined value pair m_i & a_i (i.e., linear transformation) and L Rabin fingerprints (as used in Content-Defined Chunking [26, 40, 43] with a sliding window size of 48 bytes as follows:

$$\text{Feature}_i = \text{Max}_{j=1}^{L}\{(m_i \cdot \text{Rabin}_j + a_i) \bmod 2^{32}\} \qquad (1)$$

Where Rabin_j is the Rabin fingerprint of the sliding window located at position j) Thus chunks that have one or more such features (maximal values) in common are likely to be very similar, but small changes to the data are unlikely to perturb the maximal values [5, 29]. Algorithm 1 provides a detailed pseudo-code implementation of extracting features by *N-transform SF*. Then a super-feature of this chunk, SF_x, can be calculated by several such features as follows:

$$\text{SF}_x = \text{Rabin}(\text{Feature}_{x \cdot k}, ..., \text{Feature}_{x \cdot k+k-1}) \qquad (2)$$

For example, to generate three *SFs* with k=4 features each, we must first generate N=12 features, namely, features 0...3 for SF_0, features 4...7 for SF_1, etc. For similar chunks that differ only in a tiny fraction of bytes, most of their features will be identical and thus so are their *SFs* [6]. More specifically, this *N-transform SF* approach is able to maximally detect the highly similar chunks for two reasons. ① The matching of one *SF* means that almost all the features grouped in this *SF* are identical and thus grouping features into *SFs* reduces false positives for resemblance detection. ② Multiple SFs are computed to increase the probability of detecting highly similar chunks. Meanwhile, this *N-transform SF* approach needs to linearly transform Rabin fingerprints of the data chunks *N* times, which is time-consuming and slows the whole post-deduplication delta compression process.

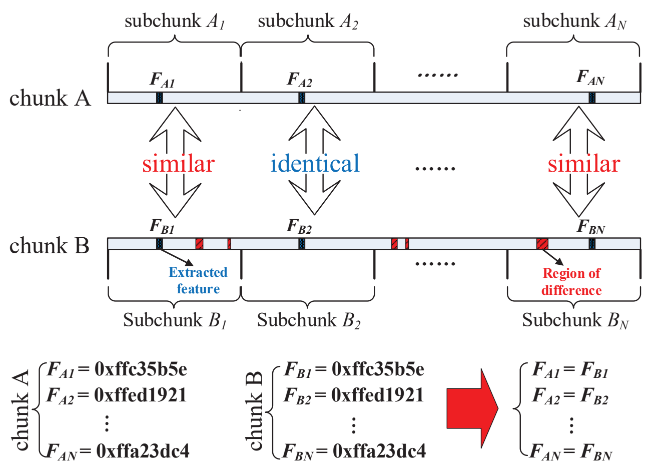


Figure 2: An example of the existence of fine-grained locality among two similar chunks. Here each chunk is divided into *N* fixed-sized subchunks. The corresponding subchunks in chunk B are largely similar (one-by-one) to subchunks in chunk A, and thus their features are largely identical.

It is worth noting that there are also some other *coarse-grained resemblance detection* approaches [4, 9, 11, 15, 27, 41] for matching similar files or large data blocks (e.g., size of 16MB), which extract features from non-overlapped strings (or chunks) and thus may suffer from high false positives. In this paper, we focus on improving the most popular *N-transform SF* approach for the chunk-level resemblance detection in post-deduplication delta compression scenario [30].

4 *Finesse* Design and Implementation

4.1 Observations

As analyzed above, the root cause of the relatively high computation overhead of the *N-transform SF* approach is its linearly transforming the whole chunk's Rabin fingerprints *N* times (i.e., compute multiple rounds of transformations on each fingerprint) to extract *N* features. According to our observation of delta compression on backup workloads, the features extracted from the subchunks inside individual chunks can also be used for resemblance detection, which means that we eliminate the linear transformations and thus simplify the feature computation.

Computing features from subchunks is motivated by our observation that the fine-grained stream locality widely exists in the detected similar chunks. Figure 2 provides an example of this locality: the subregions (subchunks) of individual chunks also appear in the same order among their highly similar chunks with a very high probability, meaning that these subchunks are also very similar to each other.

Table 1 studies this locality on six deduplicated backup datasets (the detailed experimental environment and work-

Table 1: A study of the repeatability of subchunks and their features (i.e., the fine-grained locality) in the identified similar chunks in six deduplicated backup datasets. Here the identified chunks are all divided into 12 equal-sized subchunks and then we verify the locality shown in Figure 2.

Datasets	WEB	TAR	RDB	SYN	VMA	VMB
Avg. # of subchunks (identical)	8.27	9.19	6.86	5.78	5.99	6.34
Avg. # of subchunks (own the same features)	10.82	10.97	10.23	10.10	10.04	10.64

Here *identical* is judged by checking SHA-1 fingerprints of subchunks.

Algorithm 2 Extracting features in *Finesse*.

```
Require: chunk content, Str; length of the chunk, L;
Ensure: N features, Feature[N];
 1: function FEATURE-EXTRACT-FINESSE(Str, L)
 2:     subChunkSize ← $\frac{L}{N}$;
 3:     Feature[$0, \cdots, N-1$] ← 0;
 4:     for m = 0 to N − 1 do
 5:         for i = 0 to subChunkSize − 1 do
 6:             FP ← RabinFunction(Str, m*subChunkSize + i);
 7:             if Feature[m] ≤ FP then
 8:                 Feature[m] ← FP;
 9:             end if
10:         end for
11:     end for
12: end function
```

load characteristics can be found in Section 5), which demonstrates that most of the corresponding subchunk pairs in the detected similar chunks have the same features, accounting for 87.22% on average, although many of them are non-duplicate, accounting for 41.07% on average. Therefore, grouping some of these features into *SFs* by exploiting this fine-grained locality of similar chunks may also potentially enable maximal detection of highly similar chunks.

More importantly, this fine-grained locality-based resemblance detection approach has the potential to greatly reduce the execution time of computing features while achieving comparable resemblance detection efficiency relative to the *N-transform SF* approach, which is comprehensively evaluated and demonstrated in in Section 5.

4.2 Implementation

In this subsection, we discuss some implementation issues of *Finesse*, including the feature extraction and grouping strategies, overhead analysis, and other design considerations.

Feature Extraction. To exploit the fine-grained feature locality of similar chunks for extracting features, *Finesse* first divides a chunk into several fixed-sized subchunks, and then computes features on each subchunk based on the Rabin fingerprints of the data contents, in the same way as the traditional *N-transform SF* approach, which is detailed in Algorithm 2. Note that a chunk can be divided

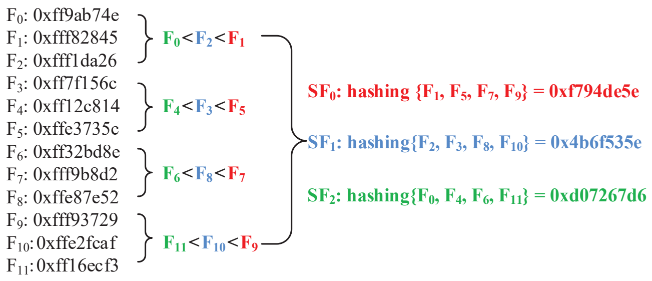


Figure 3: A concrete example of the grouping strategy in *Finesse* with actual values for the features and *SFs*.

into variable-sized subchunks, similar to Content-Defined Chunking [26, 40, 43], but the feature grouping process will become very complicated since the subchunks' sizes and the number of features will become unknown. In addition, our preliminary results suggest that extracting features on fixed-sized subchunks is able to achieve nearly the same compression ratio as *N-transform SF*, and thus we use the fixed-sized subchunks for feature extraction.

Feature Grouping. The grouping strategy in *Finesse* is different from traditional *N-transform SF* since the way features are extracted is changed in *Finesse*. Specifically, *Finesse* first divides the subchunks and their corresponding features into several contiguous sets of the same size. Then the biggest features (with the largest hash values from each of the sets) are grouped to constitute the first *SF*, the second-biggest features of the sets are grouped to form the second *SF*, and so on and so forth. This grouping strategy in *Finesse* ensures that the grouped features in each *SF* are selected uniformly and consistently all over the chunks, which achieves grouping efficiency similar to *N-transform SF*.

To better illustrate this grouping strategy, we provide a detailed example with three *SFs* and four features per *SF* in *Finesse* as shown in Figure 3. We first divide the chunk into twelve subchunks to extract 12 features $F_0 ... F_{11}$. These features are further divided into four sets and sorted by their values, $\{F_0<F_2<F_1\} ... \{F_{11}<F_{10}<F_9\}$. Finally, SF_0 is composed of the maximal values from the above four sets, namely, $\{F_1, F_5, F_7, F_9\}$, SF_2 of the 2nd biggest values $\{F_2, F_3, F_8, F_{10}\}$, and SF_2 of the 3rd biggest values $\{F_0, F_4, F_6, F_{11}\}$. Therefore, compared with feature extraction, feature grouping is fast since it only processes a small amount of features instead of the whole data chunk.

Note that we tried other grouping strategies for *Finesse* but the performance differences were small or ever worse. And our final evaluation result suggests *Finesse* using this grouping strategy achieves nearly the same delta-compression ratio as the classic *N-transform SF*.

Computational Overhead. As discussed above, the computational overhead of grouping features in *Finesse* is insignificant compared with computing features. Thus, we only analyze the computational overhead on computing features. Specifically, to generate *N* features from one chunk,

Table 2: Workload characteristics of the tested datasets.

Name	Size	DR	Workload descriptions
WEB	367 GB	4.21	135 days' snapshots of the website: *news.sina.com*.
TAR	112 GB	1.70	258 versions of Linux kernel source code [1]. Each version is packaged as a tar file.
RDB	540 GB	12.25	100 backups of the redis key-value store database.
SYN	330 GB	13.07	176 synthetic backups by simulating file create/delete/modify operations [32].
VMA	117 GB	1.61	78 virtual machine images of different OS release versions, including Fedora, CentOS, Debian, etc [3].
VMB	321 GB	10.45	20 backups of an Ubuntu 12.04 VM image in use by a research group.

Deduplication Ratio (DR) is measured by $\frac{\text{total data size before deduplication}}{\text{total data size after deduplication}}$.

for each Rabin fingerprint on the data chunk contents:

- *N-transform SF* needs at least 3×N operations, including N multiply, N add, and N conditional branch operations, to select N maximal values (i.e., features) after linear transformation as discussed in Section 3.
- *Finesse* only needs one operation, i.e., one 'conditional branch', to select one maximal value (one feature) in each subchunk.

Therefore, *Finesse* greatly reduces the computation overhead for feature extraction and thus accelerates the whole resemblance-detection process.

Limitations. Note that *Finesse* has one limitation in that it does not detect "similar" chunks with very different sizes. This is because *Finesse* divides a chunk into several equal-sized subchunks and the features will be totally different if the two "similar" chunks are of very different sizes. But in the delta-compression scenario, detecting chunks with similar sizes is reasonable since "similar" chunks with very different sizes (detected by the *N-transform SF approach*) may result in a low delta-compression ratio [17]. For example, two non-similar chunks that only have a small region in common may have many features and *SFs* in common and thus be considered to be similar chunks by the *N-transform SF approach*.

5 Performance Evaluation

5.1 Evaluation Setup

Experimental Platform. We implement delta compression in an open-source deduplication prototype system called Destor [12, 13] on the Ubuntu 12.04.2 operating system running on a quad-core Intel i7-4770 processor at 3.4 GHz and two 1TB 7200RPM hard disks. Another Intel E5-2620 processor at 2.4 GHz is also used for performance comparison.

Data Reduction Configurations. In our prototype system, deduplication is configured with Rabin-based chunking with the expected chunk size of 8KB as used in LBFS [26] and an in-memory SHA1 fingerprint table for duplicate detection.

For the post-deduplication delta compression, the non-duplicate chunks are processed in three steps: *resemblance detection, base chunk reading, and delta encoding*. The *resemblance detection* step for both *Finesse* and *N-transform SF* is configured to compute 3 *SFs* and 4 features per *SF* for matching highly similar chunks as suggested by SIDC [29] and MC [19] (to trade off the space savings and the computation & indexing overheads). In addition, a chunk may have multiple similar chunks, and our prototype system selects the first matched chunk as its base, which is also known as "FirstFit" [17]. For the *base chunk reading* step, delta compression needs to read for each matched similar chunk its base chunk from the disk for delta encoding. Here we use a base chunk cache with LRU and a size of 400MB to reduce base chunk I/Os. For *delta encoding*, we employ the classic Xdelta [20] to calculate the delta of the similar chunks for space saving.

Performance Metrics. We evaluate resemblance detection performance of *Finesse* using two metrics, *Delta Compression Ratio* (DCR) and *Delta Compression Efficiency* (DCE). DCR is measured by $\frac{\text{total size before delta compression}}{\text{total size after delta compression}}$, reflecting the total space saved by resemblance detection and then delta compression. DCE is used to estimate the similarity degree between the similar chunks detected by *Finesse*, i.e., $\frac{\text{the chunk data size after delta compression}}{\text{the chunk data size before delta compression}}$. It is worth noting that DCR focuses on the overall space savings while DCE emphasizes the detected resembling chunks themselves. Thus higher DCE means lower probability of false positives for detecting similar chunks.

In addition, *Similarity Computing Speed* is measured by the processing speed at which the input data are calculated to obtain *SFs* for resemblance detection. *System Throughput* is measured by the throughput with which the input data are deduplicated and then delta compressed. We run each experiment five times to get the stable and the average results of the deduplication throughput.

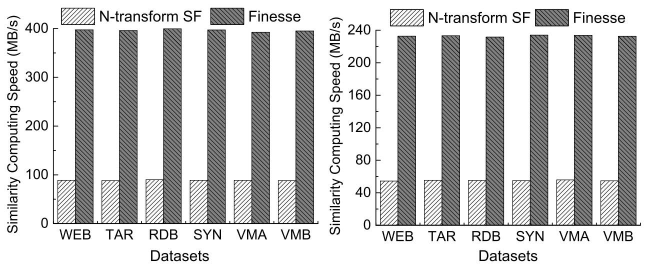
Evaluated Datasets. Six datasets are used for evaluation as shown in Table 2. These datasets represent various typical workloads, including website snapshots, tarred source code files, database snapshots, and virtual machine images.

5.2 Evaluation of *Finesse* vs. *N-transform SF*

Resemblance Detection Efficiency. Table 3 provides the delta compression results of all the similar chunks detected (i.e., they have a super-feature in common) by *Finesse* and *N-transform SF* respectively. Generally, evaluation results in Table 3 suggest that *Finesse* achieves comparable compression ratio (with difference of -3.21%∼+7.36%) to the N-transform SF approach in the metrics of DCR and DCE. In addition, the resemblance detection performance of *Finesse* is sensitive to the datasets due to the different ways in which the files of each workload are evolved (i.e., modified) during backups. Thus the six workloads have different levels of the

Table 3: Comparison of resemblance detection efficiency of *N-transform SF* and *Finesse* on the six datasets.

Dataset	Approaches	DCR	DCE
WEB	*N-transform SF*	7.60	0.8749
	Finesse	7.52 (–1.05%)	0.8795 (+0.53%)
TAR	*N-transform SF*	15.00	0.9516
	Finesse	15.34 (+2.27%)	0.9846 (+3.47%)
RDB	*N-transform SF*	3.67	0.9129
	Finesse	3.94 (+7.36%)	0.9448 (+3.49%)
SYN	*N-transform SF*	1.75	0.9326
	Finesse	1.70 (–2.86%)	0.9640 (+3.37%)
VMA	*N-transform SF*	1.56	0.9088
	Finesse	1.51 (–3.21%)	0.9161 (+0.80%)
VMB	*N-transform SF*	1.30	0.9093
	Finesse	1.28 (–1.54%)	0.9193 (+1.10%)



(a) Intel i7-4770 (b) Intel E5-2620

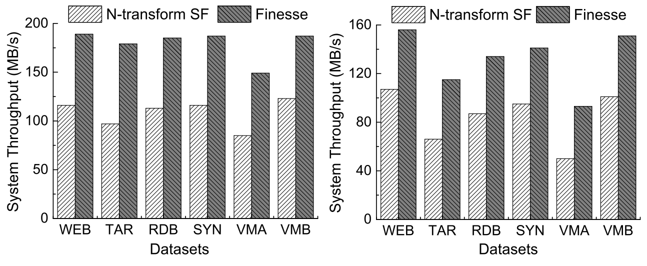
Figure 4: Similarity computing speed.

fine-grained locality as studied in Table 1 (see Section 4.1). For example, *Finesse* achieves higher DCR on datasets TAR and RDB, and lower DCR on datasets SYN and VMB.

Meanwhile, *Finesse* achieves higher DCE than *N-transform SF* on all the six datasets. There are two reasons. ① The *N-transform SF* approach may obtain all the features from one subregion of the chunk, which can lead to possible false positive resemblance detection and thus lower DCE. In contrast, *Finesse*'s *SF* grouping strategy ensures that the features grouped for each *SF* are coming from multiple subchunks of a chunk. ② *N-transform SF* may detect "similar" chunks with the very different sizes, which can result in poor delta compression efficiency as discussed in Section 4.2.

Speed of Computing SFs. While *Finesse* achieves comparable compression ratios to that of *N-transform SF*, it greatly accelerates the similarity computation as shown in Figure 4. *Finesse* improves this speed by an average of 3.5$\times$ and 3.2$\times$ respectively on the i7-4770 and E5-2620 CPUs. This is because it requires much fewer operations on computing features as discussed in Section 4. Note that the SF computing speed is not sensitive to the datasets because the time on computing features is decided by the size of the data chunks (i.e., scan all the bytes to calculate features and SFs).

System Throughput. To understand the impact of the resemblance detection approaches on the total throughput



(a) Intel i7-4770 (b) Intel E5-2620

Figure 5: Throughputs of the *Finesse* based and *N-transform SF* based delta compression prototype systems.

of the composite data reduction system combining deduplication and delta compression, we construct and evaluate the throughputs of two such systems with *Finesse* and *N-transform SF* as their delta compression components respectively. In our prototype system of both *Finesse* and *N-transform SF*, we pipeline the deduplication subtasks (i.e., chunking, fingerprinting, and indexing) and delta compression subtasks (i.e., resemblance detection, reading base chunk, and delta encoding) for high system throughput.

Figure 5 shows the evaluation results comparing these two systems. The system based on the *Finesse* approach outperforms the one based on *N-transform SF* by 41%-85% in total system throughput. This is because in the delta compression phase after deduplication, *Finesse* is running 3$\times$ faster than *N-transform SF* for resemblance detection.

6 Conclusion

In this paper, we propose *Finesse*, a much faster resemblance detection approach than the state-of-the-art *N-transform SF* approach. The key idea behind *Finesse* is to exploit the fine-grained feature locality of highly similar chunks by dividing data chunk into multiple subchunks and extract features from each subchunk, thus reducing the computation overhead of resemblance detection. Our experimental results based on six datasets demonstrate the superior performance of *Finesse* in terms of delta compression ratio, delta compression efficiency, speed of computing SFs, and throughput of the composite data reduction prototype system combining deduplication and delta compression.

Acknowledgments

We are grateful to our shepherd Geoff Kuenning and the anonymous reviewers for their insightful comments and feedback. The work was partly supported by NSFC No.61502190, No.61821003, No.U1705261, No.61832007, No.61772222, No.61772212 and No.61672120; National Key Research and Development Program of China under Grant 2016YFC0702002; The Scientific Research Fund of Hubei Provincial Department of Education B2017042; US NSF under Grants CCF-1704504 and CCF-1629625.

References

[1] Linux archives. `https://www.kernel.org`.

[2] Minhash. `https://en.wikipedia.org/wiki/MinHash`.

[3] VMs archives. `http://www.thoughtpolice.co.uk`.

[4] ARONOVICH, L., ASHER, R., BACHMAT, E., BITNER, H., HIRSCH, M., AND KLEIN, S. T. The design of a similarity based deduplication system. In *the 2th Annual International Systems and Storage Conference (SYSTOR'09)* (Haifa, Israel, 2009), ACM Association, pp. 1–14.

[5] BRODER, A. Z. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES'97)* (Washington, DC, USA, 1997), IEEE, pp. 21–29.

[6] BRODER, A. Z. Identifying and filtering near-duplicate documents. In *Combinatorial Pattern Matching* (Montreal, Canada, 2000), Springer, pp. 1–10.

[7] BRODER, A. Z., CHARIKAR, M., FRIEZE, A. M., AND MITZENMACHER, M. Min-wise independent permutations. *Journal of Computer and System Sciences 60*, 3 (2000), 630–659.

[8] CUI, Y., LAI, Z., WANG, X., DAI, N., AND MIAO, C. Quicksync: Improving synchronization efficiency for mobile cloud storage services. In *International Conference on Mobile Computing and NETWORKING (MobiCom'15)* (Paris, France, 2015), ACM Association, pp. 592–603.

[9] DOUGLIS, F., AND IYENGAR, A. Application-specific delta-encoding via resemblance detection. In *USENIX Annual Technical Conference, General Track* (San Antonio, TX, USA, 2003), USENIX Association, pp. 113–126.

[10] EL-SHIMI, A., KALACH, R., KUMAR, A., AND ET AL. Primary data deduplication-large scale study and system design. In *the 2012 conference on USENIX Annual Technical Conference* (Boston, MA, USA, 2012), USENIX Association, pp. 1–12.

[11] FORMAN, G., ESHGHI, K., AND CHIOCCHETTI, S. Finding similar files in large document repositories. In *the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05)* (Chicago, Illinois, USA, 2005), ACM Association, pp. 394–400.

[12] FU, M. Destor: An experimental platform for chunk-level data deduplication. https://github.com/fomy/destor, 2014.

[13] FU, M., FENG, D., HUA, Y., HE, X., CHEN, Z., XIA, W., ZHANG, Y., AND TAN, Y. Design tradeoffs for data deduplication performance in backup workloads. In *the 13th USENIX Conference on File and Storage Technologies (FAST'15)* (Santa Clara, CA, USA, 2015), vol. 9, USENIX Association, pp. 331–345.

[14] JACCARD, P. Etude de la distribution florale dans une portion des alpes et du jura. *Bulletin De La Societe Vaudoise Des Sciences Naturelles 37*, 142 (1901), 547–579.

[15] JAIN, N., DAHLIN, M., AND TEWARI, R. TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization. In *the USENIX Conference on File and Storage Technologies (FAST'05)* (San Francisco, CA, USA, 2005), USENIX Association, pp. 281–294.

[16] KAISER, J., MEISTER, D., AND BRINKMANN, A. Deriving and comparing deduplication techniques using a model-based classification. In *the 10th European Conference on Computer Systems (EuroSys'15)* (Bordeaux, France, 2015), ACM Association, pp. 1–13.

[17] KULKARNI, P., DOUGLIS, F., LAVOIE, J. D., AND TRACEY, J. M. Redundancy elimination within large collections of files. In *the 2004 USENIX Annual Technical Conference (ATC'04)* (Boston, MA, USA, 2004), USENIX Association, pp. 1–14.

[18] LILLIBRIDGE, M., ESHGHI, K., BHAGWAT, D., AND ET AL. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *the 7th USENIX Conference on File and Storage Technologies (FAST'09)* (San Jose, CA, 2009), vol. 9, USENIX Association, pp. 111–123.

[19] LIN, X., LU, G., DOUGLIS, F., SHILANE, P., AND WALLACE, G. Migratory compression: Coarse-grained data reordering to improve compressibility. In *the 12th USENIX Conference on File and Storage Technologies (FAST'14)* (Santa Clara, CA, USA, 2014), USENIX Association, pp. 257–271.

[20] MACDONALD, J. *File system support for delta compression*. PhD thesis, Masters thesis. Department of Electrical Engineering and Computer Science, University of California at Berkeley, 2000.

[21] MEISTER, D., JÜRGEN, AND BRINKMANN. Multi-level comparison of data deduplication in a backup scenario. In *the 2th Annual International Systems and Storage Conference (SYSTOR'09)* (Haifa, Israel, 2009), ACM Association, pp. 1–12.

[22] MEISTER, D., KAISER, J., AND BRINKMANN, A. Block locality caching for data deduplication. In *the 6th International Systems and Storage Conference (Systor'13)* (Haifa, Israel, 2013), ACM Association, pp. 1–12.

[23] MEISTER, D., KAISER, J., BRINKMANN, A., AND ET AL. A Study on Data Deduplication in HPC Storage Systems. In *the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)* (Salt Lake City, Utah, USA, 2012), IEEE Computer Society Press, pp. 1–11.

[24] MEYER, D., AND BOLOSKY, W. A study of practical deduplication. In *the 9th USENIX Conference on File and Storage Technologies (FAST'11)* (San Jose, CA, USA, 2011), USENIX Association, pp. 229–241.

[25] MIN, J., YOON, D., AND WON, Y. Efficient deduplication techniques for modern backup operation. *IEEE Transactions on Computers 60*, 6 (2011), 824–840.

[26] MUTHITACHAROEN, A., CHEN, B., AND MAZIERES, D. A Low-Bandwidth Network File System. In *the ACM Symposium on Operating Systems Principles (SOSP'01)* (Banff, Canada, 2001), ACM Association, pp. 1–14.

[27] PUCHA, H., ANDERSEN, D. G., AND KAMINSKY, M. Exploiting similarity for multi-source downloads using file handprints. In *the 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI'07)* (Cambridge, MA, 2007), USENIX Association, pp. 15–28.

[28] RABIN, M. O. *Fingerprinting by Random Polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[29] SHILANE, P., HUANG, M., WALLACE, G., AND ET AL. WAN optimized replication of backup datasets using stream-informed delta compression. In *the 10th USENIX Conference on File and Storage Technologies (FAST'12)* (San Jose, CA, USA, 2012), USENIX Association, pp. 49–63.

[30] SHILANE, P., WALLACE, G., HUANG, M., AND HSU, W. Delta Compressed and Deduplicated Storage Using Stream-Informed Locality. In *the 4th USENIX conference on Hot Topics in Storage and File Systems (HotStorage'12)* (Boston, MA, USA, 2012), USENIX Association, pp. 201–214.

[31] TARASOV, V., JAIN, D., KUENNING, G., MANDAL, S., PALANISAMI, K., SHILANE, P., TREHAN, S., AND ZADOK, E. Dmdedup: Device mapper target for data deduplication. In *Ottawa Linux Symposium (OLS'14)* (2014), pp. 1–10.

[32] TARASOV, V., MUDRANKIT, A., BUIK, W., SHILANE, P., KUENNING, G., AND ZADOK, E. Generating realistic datasets for deduplication analysis. In *Proceedings of the 2012 conference on USENIX Annual technical conference* (2012), USENIX Association, p. 24C34.

[33] WALLACE, G., DOUGLIS, F., QIAN, H., AND ET AL. Characteristics of backup workloads in production systems. In *the 10th USENIX Conference on File and Storage Technologies (FAST'12)* (San Jose, CA, 2012), USENIX Association, pp. 33–48.

[34] Xia, W., Jiang, H., Feng, D., Douglis, F., Shilane, P., Hua, Y., Fu, M., Zhang, Y., and Zhou, Y. A comprehensive study of the past, present, and future of data deduplication. *Proceedings of the IEEE 104*, 9 (2016), 1681–1710.

[35] Xia, W., Jiang, H., Feng, D., and Hua, Y. SiLo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput. In *the 2011 conference on USENIX Annual Technical Conference (ATC'11)* (Portland, OR, 2011), USENIX Association, pp. 285–298.

[36] Xia, W., Jiang, H., Feng, D., and Tian, L. Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets. In *Data Compression Conference (DCC), 2014* (2014), IEEE, pp. 203–212.

[37] Xia, W., Jiang, H., Feng, D., and Tian, L. DARE: A deduplication-aware resemblance detection and elimination scheme for data reduction with low overheads. *IEEE Transactions on Computers 65*, 6 (2016), 1692–1705.

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

[39] Xia, W., Li, C., Jiang, H., Feng, D., Hua, Y., Qin, L., and Zhang, Y. Edelta: A word-enlarging based fast delta compression approach. In *the 7th USENIX conference on Hot Topics in Storage and File Systems* (Santa Clara, CA, 2015), USENIX Association, pp. 1–5.

[40] Xia, W., Zhou, Y., Jiang, H., Feng, D., Hua, Y., Hu, Y., Liu, Q., and Zhang, Y. FastCDC: A fast and efficient content-defined chunking approach for data deduplication. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)* (Denver, CO, 2016), USENIX Association, pp. 101–114.

[41] Xu, L., Pavlo, A., Sengupta, S., and Ganger, G. R. Online deduplication for databases. In *ACM International Conference on Management of Data (SIGMOD'17)* (Chicago, IL, USA, 2017), ACM Association, pp. 1355–1368.

[42] Xu, L., Pavlo, A., Sengupta, S., Li, J., and Ganger, G. R. Reducing replication bandwidth for distributed document databases. In *the 6th ACM Symposium on Cloud Computing (SoCC'15)* (Big Island, Hawaii, USA, 2015), ACM Association, pp. 222–235.

[43] Zhang, Y., Jiang, H., Feng, D., Xia, W., Fu, M., Huang, F., and Zhou, Y. AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication. In *Proceedings of the IEEE INFOCOM* (2015), IEEE, pp. 1337–1345.

[44] Zhu, B., Li, K., and Patterson, R. H. Avoiding the disk bottleneck in the data domain deduplication file system. In *the 6th USENIX Conference on File and Storage Technologies (FAST'08)* (San Jose, CA, USA, 2008), vol. 8, USENIX Association, pp. 269–282.