Edelta: A Word-Enlarging Based Fast Delta Compression Approach

Wen Xia, Chunguang Li, Hong Jiang[†], Dan Feng[*], Yu Hua, Leihua Qin, Yucheng Zhang

School of Computer, Huazhong University of Science and Technology, Wuhan, China
Wuhan National Laboratory for Optoelectronics, Wuhan, China
[†]Dept. of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, USA

Abstract

Delta compression, a promising data reduction approach capable of finding the small differences (i.e., delta) among very similar files and chunks, is widely used for optimizing replicate synchronization, backup/archival storage, cache compression, etc. However, delta compression is costly because of its time-consuming word-matching operations for delta calculation. Our in-depth examination suggests that there exists strong word-content locality for delta compression, which means that contiguous duplicate words appear in approximately the same order in their similar versions. This observation motivates us to propose Edelta, a fast delta compression approach based on a word-enlarging process that exploits word-content locality. Specifically, Edelta will first tentatively find a matched (duplicate) word, and then greedily stretch the matched word boundary to find a likely much longer (enlarged) duplicate word. Hence, Edelta effectively reduces a potentially large number of the traditional time-consuming word-matching operations to a single word-enlarging operation, which significantly accelerates the delta compression process. Our evaluation based on two case studies shows that Edelta achieves an encoding speedup of 3X∼10X over the state-of-the-art Ddelta, Xdelta, and Zdelta approaches without noticeably sacrificing the compression ratio.

1 Introduction

Delta compression is gaining increasing attention as a promising technology that effectively eliminates redundancy among the non-duplicate but very similar data chunks and files in storage systems. Most recently, Difference Engine [2] combines delta compression, deduplication, and LZ compression to reduce memory usage in VM environments, where delta compression delivers about 2X more memory savings than VMware ESX server's deduplication-only approach. Shilane et al. [4] implement delta compression on top of deduplication to further eliminate redundancy among similar data to accelerate the WAN replication of backup datasets, which obtains an additional compression factor of 2X-3X. Dropbox [1] implements delta compression to reduce the bandwidth requirement of uploading the updated files by calculating the small differences between two revisions and sending only the delta updates.

[*]Corresponding author: dfeng@hust.edu.cn.

Although delta compression has been applied in many areas for space saving, challenges facing high-performance delta compression remain. One of the main challenges is its time-consuming word-matching process for delta calculation, which tries to first find the possible duplicate words and then the delta between two similar chunks or files. As suggested by the state-of-the-art approaches [5, 9], delta compression only offers speeds of about 25 MB/s (Zdelta), 60 MB/s (Xdelta), 150 MB/s (Ddelta), a worsening problem in face of the steadily increasing storage bandwidth and speed, for example, an IOPS of about 100,000 and sequential I/O speed of about 500MB/s offered by Samsung SSD 850 PRO100,000.

Our examination of delta compression suggests that contiguous duplicate words appear in approximately the same order among the similar chunks and files. We call this phenomenon the word-content locality, which is similar to the chunk data locality observed in many deduplication based storage systems [4]. This observation motivates us to propose Edelta, a fast delta compression approach based on a work-enlarging process that exploits the word-content locality to reduce the conventional time-consuming word-matching operations. Specifically, if Edelta finds a matched word between two similar chunks (or files) A and B, it directly uses a byte-wise comparison in the remaining regions immediately after the matched word in chunks A and B to find the potentially much longer (i.e., enlarged) duplicate words. This word-enlarging method helps avoid most of the traditional duplicate-checking operations, such as hashing, indexing, etc., and thus significantly speeding up the delta compression process.

The Edelta research makes the following three key contributions: (1) The observation of the word-content locality existing in delta compression of the similar chunks and files, which suggests that the size of an actual duplicate segment is usually much larger than that of a word conventionally used for duplicate checking in delta compression. (2) A novel word-enlarging based delta compression approach, Edelta, to accelerate the duplicate-word checking process by directly enlarging each matched word into a much longer one and thus avoiding the word-matching operations in the enlarged regions. (3) Experimental results on two case studies demonstrating Edelta's very high encoding speed that is 3X-10X faster than the state-of-the-art approaches with-

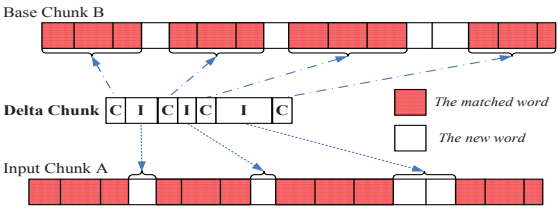


Figure 1: Illustration of delta encoding.

out significantly sacrificing the compression ratio.

2 Background

Delta compression is a special dictionary compression technique that targets files or chunks considered very similar. It is widely used in the areas of replicate synchronization, backup/archival storage, cache compression, etc., to save storage space or network bandwidth.

Figure 1 shows a detailed example of the Ddelta compression [9] on two similar chunks, i.e., input (new) chunk A and its similar (base) chunk B. Ddelta first divides the two chunks into several words (e.g., with an average size of 32 Bytes) by content-defined chunking, and then finds the duplicate words by hashing (fingerprinting) and indexing. Specifically, Ddelta uses the words in the base chunk B as "dictionary" and indexes the words of chunk A in the "dictionary" for duplicate checking.

For the detected duplicate words, which are copied from base chunk B, Ddelta will encode it into a "Copy" message with the offset and size information. For the new words, which are inserted from input chunk A, Ddelta will directly store it as an "Insert" message. Note that Ddelta will merge the contiguous duplicate or new words into a bigger word for better decoding performance. After that, Ddelta obtains the encoded delta chunk $\Delta_{(A,B)}$ that is much smaller than the original input chunk A for storage space saving. With delta chunk $\Delta_{(A,B)}$ and base chunk B, Ddelta can easily restore chunk A according to the "Copy" and "Insert" messages.

Shilane et al. [4, 5] and Xia et al. [8, 9] conclude that implementing delta compression on top of deduplication to further eliminate redundancy among non-duplicate but very similar data in backup systems can obtain an additional compression factor of 2X-3X. Delta compression increases space savings but comes at the expense of significant computation and indexing overheads, as revealed by many recent studies [4, 5, 9]. There are two types of candidates for delta compression, namely, the already known similar data (e.g., updated files [3]) and the resemblance-detected data (e.g., post-deduplication data reduction [4, 9]). The delta-compression overhead for the former type is mainly from delta encoding, while that for the latter is from both resemblance detection and delta encoding. Here resemblance detection refers to the two-stage process of hashing and indexing for features and super-features that measure similarity, which has been well explored in many studies [5, 8]. In this paper, we focus on designing a very fast delta encoding approach, i.e., accelerating the duplicate-checking process of known delta-compression candidates.

Table 1: Average lengths (in Bytes) of the grouped "Copy"/"Insert" messages after Ddelta compression.

Dataset	LX	SC	GC	EC	GD	GL	PH	PT
Copy	10K	5K	3k	18K	10K	6K	4K	15K
Insert	123	340	133	124	173	136	101	149

3 Observation and Motivation

In this section, we examine the duplicate-word checking process in the state-of-the-art Ddelta compression approach [9] with several updated tarred files, a common case for delta compression. As shown in previous studies [3, 9], the larger the size of the word used for duplicate-checking is, the lower the delta compression ratio but the faster the compressing speed are. Here we use the 64-Byte word for studying delta calculation.

Table 1 shows the average lengths of the "Copy" and "Insert" messages obtained by Ddelta compression for eight updated tarred files. The updated files listed in the first row are the neighboring versions selected from eight known open-source projects, i.e., Linux, Scilab, GCC, Emacs, GDB, Glib, PHP, and Python, respectively. Most of these datasets are also used for evaluating the Ddelta and Zdelta compression approaches [9, 7].

The results in Table 1 suggest that the average length of "Insert" is about 128 Bytes while that of "Copy" is about several and even tens of Kilobytes, which provides two important findings. **Finding 1**: Modification (i.e., updating) always occurs at relatively small and far apart regions. **Finding 2**: A significant portion of all regions remain common (i.e., duplicate) after updating files, meaning that many contiguous duplicate words appear in approximately the same order among the similar chunks and files. We call this phenomenon word-content locality, which is similar to the observed chunk-data locality in deduplication-based backup systems [4].

Figure 2 plots the CDFs of the "Copy" and "Insert" lengths in Ddelta to help better understand their length distribution. The results suggest that more than 50% and 80% of "Copy" messages of the updated GCC and Linux files are larger than 1KB, and about 99% of "Insert" messages of both GCC and Linux are smaller than 1KB, which are consistent with the above two **Findings**.

Meanwhile, we measure the breakdown of Ddelta compression's time overhead and find more than 96% of the time overhead can be attributed to content-defined chunking (about 45%), hashing (about 16%), and indexing (about 35%). These observations lead us to firmly believe that the full exploitation of the above-observed word-content locality can help avoid the time-consuming chunking, hashing, and indexing operations by enlarging

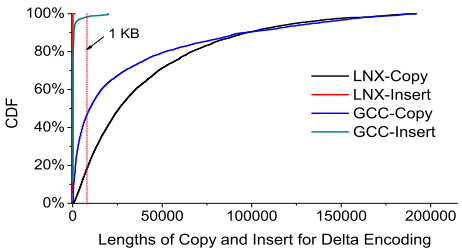


Figure 2: CDFs of the "Copy" and "Insert" lengths of Ddelta compressing updated tarred files.

the duplicate word to one much longer than the conventional word used for matching. The design and implementation of this word-enlarging based delta compression approach, called Edelta, will be detailed in the next section and its effectiveness and efficiency will be further demonstrated in Section 5.

It is worth noting that other studies such as network traffic redundancy elimination [6] and Ddelta compression [9], also observe and exploit the fine grained redundancy locality. But their approaches aim to detect more redundancy while Edelta exploits word-content locality to reduce the time overhead for redundancy detection.

4 Edelta Design and Implementation

Edelta is a general approach to delta compression. Currently we implement Edelta on top of our previous work Ddelta [9]. The idea of Edelta is simple and easy to follow as illustrated in Figure 3. More concretely, Edelta works as follows. It **(1)** tentatively detects a duplicate word between the input chunk A and its base chunk B by Ddelta's scheme, and then **(2)** uses a byte-wise comparison in the remaining regions of the two chunks immediately neighboring the duplicate words to enlarge the duplicate words until the comparison fails. Thus the enlarged duplicate regions (i.e., the shaded areas in chunks A and B) can be quickly regarded as "Copy" *without needing the conventional time-consuming duplicate-checking operations involving chunking, hashing, indexing, etc.,* thereby significantly speeding up the delta compression process. This process repeats when Edelta finds another pair of duplicate words between the two chunks, until the ends of the chunks are reached.

Note that we maximize the byte-wise comparing speed by performing one 64-bit XOR operation for each pair of 8-byte strings at a time as suggested in Ddelta [9], which, at about several GB/s in our observation, is extremely fast and comparable to the main memory bandwidth. Therefore, comparing with the hashing and indexing overheads for duplicate-word checking in Ddelta and Xdelta, the time overhead for the in-memory byte-wise comparison is negligible.

Depending on the trade-off between encoding speed and compression ratio, there are two schemes for Edelta

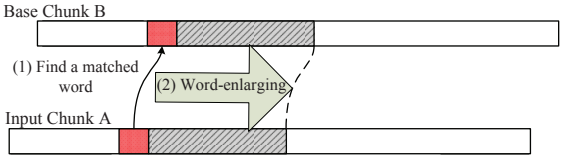


Figure 3: Accelerating delta compression via the word-enlarging process.

to accelerate delta compression.

- *Scheme I*: Skipping the conventional duplicate-checking operations only on the enlarged regions of the input chunk A and still calculating the words in the enlarged regions of the base chunk B to put them into the "dictionary" for delta calculation.
- *Scheme II*: Skipping the duplicate-checking operations on the enlarged regions of both the input chunk A and base chunk B, meaning that no chunking, hashing, and indexing will be performed on these regions and thus the delta calculation for duplicate-word checking is minimized.

Compression ratio and speed are two key metrics for assessing delta compression performance. For the compression ratio, *Scheme I* should be nearly the same as the traditional approach while *Scheme II* can be very different since the dictionary is much smaller due to the word-enlarging scheme on the base chunk B, which may result in failure to detect some duplicate words. For the compression speed, *Scheme II* has the highest speed since it maximally reduces the duplicate-checking operations while *Scheme I* only reduces these operations on the input chunk A. In general, the two schemes tradeoff between the compression ratio and speed by varying word-enlarging regions, which will be evaluated in Section 5.

5 Performance Evaluation

5.1 Experimental Setup

To evaluate Edelta, we implement an Edelta prototype on the Ubuntu 12.04.2 operating system running on a quad-core Intel i7 processor at 2.8GHz, with a 16GB RAM, two 1TB 7200rpm hard disks, and a 120GB SSD of Kingston SVP200S37A120G. Three state-of-the-art approaches, Ddelta [9], Xdelta [3], and Zdelta [7], are used as the baselines for evaluating Edelta.

Two important metrics are used for this evaluation, namely, *Compression ratio (CR)* measured in terms of percentage of data reduced by a given delta compression scheme and *encoding speed* recorded by the response time of delta calculation in memory by a given evaluated delta compression scheme.

5.2 Case Study I: Delta Compressing the Updated Tarred Files

This subsection evaluates the Edelta performance on the eight datasets of updated similar tarred files as introduced in Section 3. And we implement both Edelta's *Scheme I*, which word-enlarges only the input data file/chunk

Table 2: Compression ratio of the four delta compression approaches on the eight datasets of updated tarred files.

Dataset	Xdelta	Ddelta	Edelta-I	Edelta-II
Linux	99.81%	97.35%	98.14%	98.72%
SciLab	97.08%	93.04%	94.71%	95.05%
GCC	99.69%	96.10%	96.76%	97.04%
Emacs	99.89%	98.51%	99.22%	99.32%
GDB	99.87%	96.14%	98.61%	98.91%
GLib	99.74%	96.85%	97.68%	98.08%
PHD	99.62%	95.41%	96.82%	97.75%
Python	99.85%	97.69%	98.82%	99.03%

and is denoted **Edelta-I**, and *Scheme II*, which word-enlarges both the input and base files/chunks and is denoted **Edelta-II**, for evaluation (see Section 4). Note that we configure both of the Ddelta, Edelta-I, and Edelta-II, with an average word (string) size of 64B for Gear based content-defined chunking [9].

Table 2 shows the compression ratio results among the four approaches. Comparing with Xdelta, Edelta has a slightly lower compression ratio. This is due to Edelta's use of Ddelta's content-defined chunking schemes [9] and thus has a smaller dictionary for duplicate-word checking. But Edelta-II achieves a higher compression ratio than Ddelta and Edelta-I, a counterintuitive outcome. Our in-depth study finds that this is because Edelta's word-enlarging scheme helps find the right duplicate-word candidates. For example, assuming two similar files F_b and F_i with contents of "ABEFCDABHI" (base) and "ABXYCDABHZ" (input) respectively, there are two occurrences for the word "AB" in base file F_b. Ddelta will match the two "AB" occurrences in input file F_i with the first occurrence of "AB" in file F_b, which we call a "dirty match", thus missing the opportunity to detect more possible redundant contents (e.g., the 'H' neighboring the second occurrence of "AB" in F_i). On the other hand, Edelta can avoid this dirty match by simply enlarging the word "CD" into a longer one "CDABH", leading to the detection of more duplicates.

Figure 3 shows that Edelta significantly outperforms the Xdelta and Ddelta approaches in encoding speed on the eight workloads. Note that Zdelta is not included in this comparison because of its poor support for delta compression of large files. First, Edelta-I nearly doubles the delta encoding speed of Ddelta by reducing the duplicate-checking operations on the input files via word-enlarging. Second, Edelta-II accelerates Edelta-I by a factor of 2X-6X by further reducing the duplicate-word checking operations on the base files. The speedup of Edelta-II over Edelta-I generally reflects the distribution of the "Copy" length as studied in Table 1. Note that, while the encoding speeds of Edelta-I, Ddelta, and Xdelta remain relatively stable across all eight datasets, Edelta-II's encoding speed varies noticeably across these datasets. This is because, as discussed earlier, Edelta-II's performance is sensitive to how the length of "Copy"/"Insert" is distributed and such distribution tends to vary noticeably across different datasets as revealed in Table 1 and Figure 2.

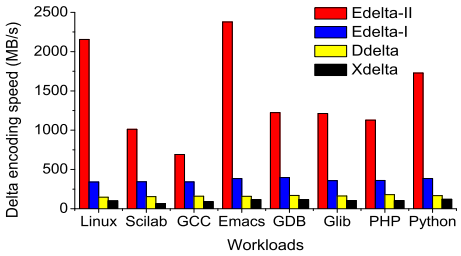


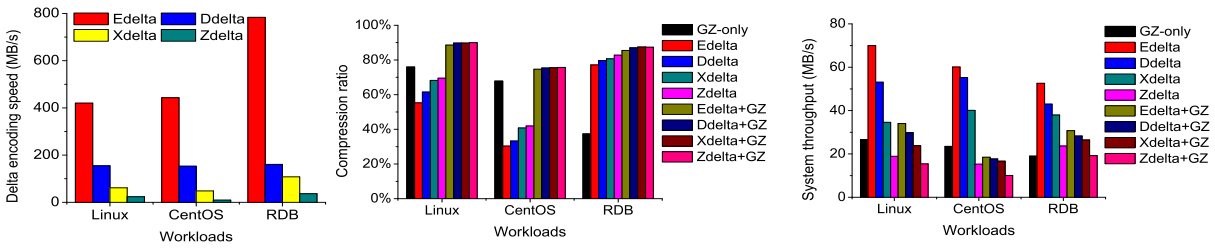
Figure 4: Delta encoding speeds of the four approaches.

Generally, the results shown in Table 2 and Figure 4 demonstrate the effectiveness and efficiency of Edelta's word-enlarging idea, that is, enlarging the duplicate-word into a longer one can significantly reduce the time overhead for duplicate-word checking while maintaining a comparable level of compression ratio.

5.3 Case Study II: Delta Compressing the Resemblance-Detected Chunks

Three typical backup/archival datasets, Linux, CentOS, and RDB, are used for evaluating delta compression in this subsection. The Linux dataset is collected from 258 Linux versions with a total size of about 104 GB; CentOS is collected from the archive of VM images of 23 CentOS release versions with a total size of about 43 GB; RDB is collected from 200 backups of the Redis database with a total size of about 1080 GB. We first deduplicate the three datasets with content-defined chunking assuming an average chunk size of 8KB and then implement delta compression (including resemblance detection, reading base chunks, delta encoding [5, 9]), GZIP compression (short for GZ), or a combination of them for post-deduplication data reduction. The deduplication factors of the three datasets are 44.7, 2.0, and 22.4 respectively. Note that in this case study, we configure Edelta and Ddelta with 32B-word for chunking to balance the compression ratio and speed, and Edelta denotes the *Edelta-II Scheme*.

Figure 5(a) shows the delta encoding speeds of for delta compression approaches on the three deduplicated datasets. As in Case Study I, Edelta achieves the highest delta encoding speed among the four approaches. The speedup of Edelta over Ddelta in this case study is not as high as that shown in Case Study I. This is because these datasets have much lower compression ratio since they have already been deduplicated before delta compression, which means that some word-content locality exploited by Edelta may be missing or weakened in this case. However, Edelta still offers a speed of 400+ MB/s for delta encoding, which is about 2.5X-5X faster than the Ddelta and Xdelta approaches.



(a) Encoding speed (b) Compression ratio (c) System throughput

Figure 5: Post-deduplication data reduction performance on the Linux, CentOS, and RDB datasets.

Figure 5(b) shows the compression ratio among GZIP-only, delta-compression-only approaches, and combinations of GZIP compression and delta compression approaches. In this case study, Edelta has the lowest compression ratio, because the dictionary for duplicate-word checking is reduced by its word-enlarging process while many duplicate words reside in the enlarged regions. For the GZIP-combined Edelta compression approach, it achieves similar compression ratio to the other three GZIP-combined approaches. Thus it is reasonable to argue that the GZIP compression, when combined with Edelta, can well compensate for the latter's loss in the compression ratio, if higher compression ratio is required for the data reduction systems.

Figure 5(c) further examines the data reduction system throughput under the nine solutions. Generally speaking, Edelta achieves the highest throughput of compressing the three deduplicated datasets with or without combining the GZIP compression. Although we compute fewer features to reduce computation overhead for resemblance detection and use SSD to store compressed data to reduce I/O latency due to frequent reading of base chunks (for delta encoding), the system throughput of Edelta is still much lower than its delta encoding speed. These results suggest that the system bottleneck has been shifted from delta encoding to another part of the critical path, e.g., resemblance detection, which will be considered as our future work in this project.

In this case study, Edelta still achieves superior performance in both delta encoding and data reduction throughput to the baseline schemes, but at a cost of slightly lower compression ratio, which can be compensated for by combining with the GZIP compression as shown in Figure 5. On the other hand, Edelta can effectively improve the GZIP compression performance in both compression ratio and throughput if there are sufficient resembling delta compression candidates in the datasets (e.g., the datasets Linux and RDB).

6 Conclusion and Future Work

In this paper, we present the motivation and design of Edelta, a fast delta compression scheme based on the observation of word-content locality during the delta calculation. Edelta effectively leverages the idea of word-enlarging to reduce the traditional time-consuming duplicate-word checking operations, which significantly speeds up the delta compression process.

In its present form, Edelta is an unoptimized prototype for just demonstrating the word-enlarging scheme. In our future study, we plan to conduct more sensitivity studies on Edelta with more workloads, improve its compressibility, and increase data reduction system throughput if resemblance detection and I/Os for reading base chunks/files are required. On the other hand, we would like to study Edelta in other application scenarios, such as in-cache compression, WAN optimization, etc.

Acknowledgments

We are grateful to our shepherd Garth Goodson and the anonymous reviewers for their insightful feedback. The work was partly supported by Chinse 973 Program No. 2011CB302301; NSFC No. 61025008, 61173043, 61232004, and 61402061; 863 Project 2013AA013203; US NSF under Grants CNS-111660, and CNS-1016609; Fundamental Research Funds for the Central Universities, HUST, under Grant No. 2014QNRC019. This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China.

References

[1] DRAGO, I., MELLIA, M., MUNAFÒ, M. M., SPEROTTO, A., SADRE, R., AND PRAS, A. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proc. ACM IMC'12*.

[2] GUPTA, D., LEE, S., VRABLE, M., SAVAGE, S., SNOEREN, A. C., VARGHESE, G., VOELKER, G. M., AND VAHDAT, A. Difference Engine: Harnessing memory redundancy in virtual machines. In *Proc. OSDI* (2010).

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

[4] SHILANE, P., HUANG, M., WALLACE, G., AND HSU, W. Wan optimized replication of backup datasets using stream-informed delta compression. In *Proc. USENIX FAST* (2012).

[5] SHILANE, P., WALLACE, G., HUANG, M., AND HSU, W. Delta compressed and deduplicated storage using stream-informed locality. In *Proc. HotStorage* (2012).

[6] SPRING, N. T., AND WETHERALL, D. A protocol-independent technique for eliminating redundant network traffic. *ACM SIGCOMM Computer Communication Review 30*, 4 (2000), 87–95.

[7] TRENDAFILOV, D., MEMON, N., AND SUEL, T. Zdelta: An efficient delta compression tool. *Tech. Report, Department of Computer and Information Science at Polytechnic University* (2002).

[8] XIA, W., JIANG, H., FENG, D., AND TIAN, L. Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets. In *Proc. IEEE DCC* (2014).

[9] 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.