Edelta: A Word-Enlarging Based Fast Delta Compression Approach

Wen Xia,Chunguang Li, Hong Jiang, Dan Feng, Yu Hua, Leihua Qin, Yucheng Zhang
Outline

• Background and Problems
• Observation and Motivation
• Edelta Design and Implementation
• Performance Evaluation
• Conclusion and Future Work
Dedup vs. Delta Compression

• In recent years, deduplication and delta compression are gaining increasing attentions.

<table>
<thead>
<tr>
<th></th>
<th>Delta Compression</th>
<th>Data Deduplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>Similar data</td>
<td>Duplicate data</td>
</tr>
<tr>
<td>Processing Granularity</td>
<td>String</td>
<td>Chunk/File</td>
</tr>
<tr>
<td>Representative Methods</td>
<td>KMP based Copy/Insert</td>
<td>CDC &amp; Secure Fingerprint</td>
</tr>
<tr>
<td>Scalability</td>
<td>Weak</td>
<td>Strong</td>
</tr>
<tr>
<td>Representative Prototypes</td>
<td>Xdelta, Zdelta</td>
<td>LBFS, DDFS</td>
</tr>
</tbody>
</table>

- Data deduplication runs much faster than delta compression.
- Delta compression is able to eliminate more redundancy among non-duplicate but similar chunks (about 2-3X more).

Can delta compression run faster than deduplication?
State of the Art on Delta Compression

\[ A_i + A_b \xrightarrow{\text{Delta}} \Delta_{b,i} \quad \Delta_{b,i} + A_b \xrightarrow{\text{Reverse delta}} A_i \]

• Delta encoding
  – Xdelta, Zdelta, Ddelta (Performance’14)

• Cache compression
  – Difference Engine (OSDI’08)
  – I-CASH (HPCA’12)

• WAN optimization/backup storage
  – Dropbox…
  – SIDC (FAST’12, HotStorage’12)
Delta Encoding

• Our Previous Work: Ddelta
  - Use Gear-based CDC to fast partition strings (words)
  - Encode the Matched /New words into Copy/Insert messages

About 3X faster than Xdelta, Cloud it be more faster??
Observation and Motivation

- **Observation 1:** In Ddelta, 96% of the time overhead is from Chunking (~45%), hashing (~16%), and indexing (~35%)
- **Observation 2:** “Copy” is very long while “Insert” is short,

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LX</th>
<th>SC</th>
<th>GC</th>
<th>EC</th>
<th>GD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>10K</td>
<td>5K</td>
<td>3k</td>
<td>18K</td>
<td>10K</td>
</tr>
<tr>
<td>Insert</td>
<td>123</td>
<td>340</td>
<td>133</td>
<td>124</td>
<td>173</td>
</tr>
</tbody>
</table>

**Motivation:** Can we exploit word-content locality to reduce some unnecessary computation operations.
An Example

• For those contiguous duplicate words \{b8, 5f, a9, c4\}, the chunking, hashing, and indexing for the words \{5f, a9, c4\} would be unnecessary by directly enlarging the detected word \{b8\}, which is just a fast byte-wise comparison.

3e b8 5f a9 c4 7d

8d b8 5f a9 c4 1b
Implementation of Edelta

- We implement Edelta on top of our previous work Ddelta
- For the two known or detected similar chunks, Edelta consists of two key steps: Find a matched word and then enlarge the word
Continue..

- Step (1): Tentatively detects a duplicate word by Ddelta’s scheme.
- Step (2): Directly enlarge the detected word into a much longer one and thus avoid the word-matching operations in the enlarged regions.

Therefore, Edelta is able to quickly identify the modified areas for delta compression by word-enlarging.

- Scheme I only word-enlarges the input data file/chunk
- Scheme II word-enlarges both the input and base files/chunks
Evaluation

• Metrics: Compression ratio and encoding speed

• Experimental Setup
  – Intel i7 processor, 16GB RAM, two 1TB 7200rpm hard disks, and a 120GB SSD of Kingston VP200S37A120G.

• Two case studies
  – 1. Delta compressing the updated tarred files
    • Datasets: linux, GDB, GCC, etc. tarred files
  – 2. Delta compressing the non-duplicate but similar Chunks
    • Datasets: RDB, VM images, Linux
    • First deduplication, and then resemblance detection, delta encoding the detected chunks.
Case Study I

- 5-20X Improvement

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Xdelta</th>
<th>Edelta-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>99.81%</td>
<td>98.72%</td>
</tr>
<tr>
<td>SciLab</td>
<td>97.08%</td>
<td>95.05%</td>
</tr>
<tr>
<td>GCC</td>
<td>99.69%</td>
<td>97.04%</td>
</tr>
<tr>
<td>Emacs</td>
<td>99.89%</td>
<td>99.32%</td>
</tr>
<tr>
<td>GDB</td>
<td>99.87%</td>
<td>98.91%</td>
</tr>
<tr>
<td>GLib</td>
<td>99.74%</td>
<td>98.08%</td>
</tr>
<tr>
<td>PHD</td>
<td>99.62%</td>
<td>97.75%</td>
</tr>
<tr>
<td>Python</td>
<td>99.85%</td>
<td>99.03%</td>
</tr>
</tbody>
</table>

- Only 1-2% decrease

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LX</th>
<th>SC</th>
<th>GC</th>
<th>EC</th>
<th>GD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>10K</td>
<td>5K</td>
<td>3k</td>
<td>18K</td>
<td>10K</td>
</tr>
<tr>
<td>Insert</td>
<td>123</td>
<td>340</td>
<td>133</td>
<td>124</td>
<td>173</td>
</tr>
</tbody>
</table>
Case Study II

- Post-deduplication delta compression
  - Dedup factors of the three datasets are 44.7, 2.0, and 22.4 respectively

- More than 400MB/s
- 2.5-5X Improv. over Delta
- Not as high as Case Study I
  - Locality missing
The hybrid data reduction system performance

- Post-dedupe Delta+GZ data reduction

Delta+GZIP have the similar compression ratio (Edelta, Xdelta)

Edelta based solutions have the highest system throughputs.
Conclusion and Future Work

• Edelta is able to delta encode a 4KB-chunk within 2-10 µs
• Edelta achieves an encoding speedup of 3-10X over the state-of-the-art DDelta, Xdelta, and Zdelta without noticeably decreasing the compression ratio

• Future Work
  – Find more promising application scenarios for Edelta
  – There are still other bottlenecks for delta compression, such as resemblance detection and reading base chunks/file

Try to make delta compression “faster” than deduplication
Thanks!

Q & A