RapidCDC: Leveraging Duplicate Locality to Accelerate Chunking in CDC-based Deduplication

Fan Ni*  
fan@netapp.com  
ATG, NetApp  

Song Jiang  
song.jiang@uta.edu  
UT Arlington  

* The work was done when he was a Ph.D. student at UT Arlington
Data is Growing Rapidly

Many of the data needs to be stored for preservation and processing.

Efficient data storage and management has become a big challenge.

From storagenewsletter.com
The Opportunity: Data Duplication is Common

- Sources of duplicate data:
  - The same files are stored by multiple users into the cloud.
  - Continuously updating of files to generate multiple versions.
  - Use of checkpointing and repeated data archiving.

- Significant data duplication has been observed for both backup and primary storage workloads.
The Deduplication Technique can Help

When duplication is detected (using fingerprinting):

File1

File2

SHA1( ) = SHA1( )

Only one copy is stored:

<table>
<thead>
<tr>
<th>Logical</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>File1</td>
<td></td>
</tr>
<tr>
<td>File2</td>
<td></td>
</tr>
</tbody>
</table>

- **Benefits**
  - Storage space
  - I/O bandwidth
  - Network traffic

- **An important feature in commercial storage systems.**
  - NetApp ONTAP system
  - Dell-EMC Data Domain system

- **Two critical issues:**
  - How to deduplicate more data?
  - How to deduplicate faster?
Deduplicate at Smaller Chunks …

… for higher deduplication ratio

- Two potentially major sources of cost in the deduplication:
  - Chunking
  - Fingerprinting
- Can chunking be very fast?
Fixed-Size Chunking (FSC)

- FSC: partition files (or data streams) into equal- and fixed-sized chunks.
  - Very fast!

- But the deduplication ratio can be significantly compromised.
  - The boundary-shift problem.

File A

<table>
<thead>
<tr>
<th>HOW ARE YOU? OK? REALLY? YES? NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

File B

<table>
<thead>
<tr>
<th>HOW ARE YOU? OK? REALLY? YES? NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
Fixed-Size Chunking (FSC)

- FSC: partition files (or data streams) into equal- and fixed-size chunks.
  - Very fast!

- But the deduplication ratio can be significantly compromised.
  - The boundary-shift problem.

```
File A
HOWAREYOU? OK? REALLY? YES? NO
\[ X \times X \times X \times X \times X \times X \times X \times X \]

File B
HHOWAREYOU? OK? REALLY? YES? NO
```
Content-Defined Chunking (CDC)

- CDC: determines chunk boundaries according to contents (a predefined special marker).
  - Variable chunk size.
  - Addresses boundary-shift problem

- Assume the special marker is ‘?’

File A

HOWAREYOU?OK?REALLY?YES?NO

File B

HHOWAREYOU?OK?REALLY?YES?NO

File A

X

File B

✓ ✓ ✓ ✓
The Advantage of CDC

- Real-world datasets include two-week’s google news, Linux kernels, and various Docker images.
- CDC’s deduplication ratio is much higher than FSC.
- However, **CDC can be very expensive.**
CDC can be Too Expensive!

Assume the special marker is ‘?’

- The marker for identifying chunk boundaries must
  - be evenly spaced out with a controllable distance in between.

- Actually the marker is determined by applying a hash function on a window of bytes.
  - E.g., \( \text{hash(“YOU?”)} == \text{pre-defined-value} \)

- The window rolls forward byte-by-byte and the hashing is applied continuously.
CDC Chunking Becomes a Bottleneck

Breakdown of CPU time

- Chunking time > 60% of the CPU time.
- I/O bandwidth is not fully utilized.
- The bottleneck shifts from the disk to CPU.
CDC Chunking Becomes a Bottleneck

- Chunking time > 60% of the CPU time.
- I/O bandwidth is not fully utilized.
- The bottleneck shifts from the disk to CPU.
CDC Chunking Becomes a Bottleneck

- Chunking time > 60% of the CPU time.
- I/O bandwidth is not fully utilized.
- The bottleneck shifts from the disk to CPU.
Efforts on Acceleration of CDC Chunking

- **Make hashing faster**
  - Example functions: SimpleByte, gear, and AE
  - More likely to generate small chunks
    - increasing size of metadata cached in memory for performance

- **Use GPU/multi-core to parallelize the chunking process**
  - Extra hardware cost
  - Substantial efforts to deploy
  - The speedup is bounded by hardware parallelism.

- **Significant software/hardware efforts, but limited performance return**
We proposed RapidCDC that …

- is still **sequential** and doesn’t require additional cores/threads.

- makes the hashing speed **almost irrelevant**.

- accelerates the CDC chunking often by **10-30 times**.

- has a deduplication ratio **the same** as regular CDC methods.

- can be adopted in an existing CDC deduplication system by adding **100~200 LOC** in a few functions.
The Path to the Breakthrough

Unique Chunks in the Disk

- [ ]
- [ ]
- [ ]
- [ ]
- [ ]
The Path to the Breakthrough

Fingerprint Matched!
The Path to the Breakthrough

Fingerprint Matched!

Confirms it!
The Path to the Breakthrough

Fingerprint Matched!
The Path to the Breakthrough

Fingerprint Matched!
The Path to the Breakthrough

Fingerprint Matched!

Fingerprint Matched!
The Path to the Breakthrough

Fingerprint Matched!
Fingerprint Matched!
Fingerprint Matched!
The Path to the Breakthrough

Fingerprint Matched!
Fingerprint Matched!
Fingerprint Matched!
Fingerprint Matched!
Fingerprint almost always happens!
Duplicate Locality

- Duplicate locality: if two of chunks are duplicates, their next chunks (in their respective files or data stream) are likely duplicates of each other.

- Duplicate chunks tend to stay together.

![Graph showing duplicate locality](Debian)
Duplicate Locality

- Duplicate locality: if two of chunks are duplicates, their next chunks (in their respective files or data stream) are likely duplicates of each other.

- Duplicate chunks tend to stay together.

---

![Graphs showing duplicate locality across different datasets](image)

(a) Linux-tar  (b) Debian  (c) Neo4j  (d) Nodejs

(e) Wordpress  (f) Cassandra  (g) Redis  (h) Google-news
RapidCDC: Using Next Chunk in History as a Hint

- History recording: whenever a chunk is detected, its size is attached to its previous chunk (fingerprint);

- Hint-assisted chunking: whenever a duplication is detected, use the history chunk size as a hint for the next chunk boundary.

When \( \text{FP}(B_1) == \text{FP}(A_1) \):

- Regular CDC is used for chunking until a duplicate chunk (e.g., \( B_1 \)) is found
More Design Considerations …

- A chunk may have been followed with chunks of different sizes
  - Maintain a size list

- Validation of Hinted Next Chunk Boundaries
  - Four alternative criterions with different efficiency and confidences
    - FF (fast-forwarding only)
    - FF+RWT (Rolling window Test)
    - FF+MT (Marker Test)
    - FF+RWT+FPT (Fingerprint Test)

- Please refer to the paper for detail.
Evaluation of RapidCDC

- Prototype: based on a rolling-window-based CDC system.
  - Using Rabin/Gear as rolling function for rolling window computation.
  - Using SHA1 to calculate fingerprints.

- Three disks with different speed are tested.
  - SATA Hard disk: 138 MB/s and 150MB/s for sequential read/write.
  - SATA SSD: 520 MB/s and 550MB/s for sequential read/write.
  - NVMe SSD: 1.2 GB/s and 2.4G/s for sequential read/write.
- Chunking speedup correlates to the deduplication ratio.
- Deduplication ratio is little affected (except for one very aggressive validation criterion).
- Chunking speedup approaches deduplication ratio.
- Negligible deduplication ratio reductions (if any).
Conclusions

- RapidCDC represents a disruptively new approach to improve CDC chunking speed.
- It increases chunking speed by up to 33X without loss of deduplication ratio.
- Its adoption in an existing CDC deduplication system does not require any major change of its current operation flow.
- Its implementation in any existing CDC deduplication systems requires minimal code changes (100-200 lines of C code in our prototype)
- A prototype implementation is available at https://github.com/moking/rapidcdc