Online Deduplication for Databases

Lianghong Xu¹, Andy Pavlo¹
Sudipta Sengupta², Greg Ganger¹
Carnegie Mellon University¹, Microsoft Research²
CMU Database Student Stabbed By Rival DBAs

PITTSBURGH (CBSDFW.COM) – Pittsburgh Police say an 26-year-old database PhD student at Carnegie Mellon University was stabbed during a fight with a gang of database administrators (DBAs). This gang has had a long-standing feud with the CMU Database Research group.
Goal: Data Reduction in DBMSs

New/updated records

Primary Database

Operation logs (Oplogs)

WAN

Secondary

Secondary

Operation logs (Oplogs)
Goal: Data Reduction in DBMSs

New/updated records

Primary Database

Operation logs (Oplogs)

Secondary

Operation logs (Oplogs)

Secondary
Goal: Data Reduction in DBMSs

- **Reduce cost**
  - Storage & network hardware
  - Maintenance

- **Improve Performance**
  - Reduce disk I/Os
  - Lower replication bandwidth
Current Reduction Approaches

• Block-level compression
  – Simple, effective
  – Fails to address redundancy across blocks

• Chunk-based Deduplication
  – Identify global redundancy across data corpus
  – Not suitable for database workloads
    • Records are relatively small
    • Modifications are small and dispersed
Similarity-based Dedup

• sDedup [Xu et al., SoCC’15]
  – Use byte-level delta compression against similar records to achieve high compression ratio
  – Focused on reduction in network bandwidth for replication

• dbDedup (this work)
  – Achieve superior compression on both network and storage
  – Address challenges with accessing delta-encoded storage
  – Allow use of dedup in online DBMSs with low perf. overhead
Outline

• Goal and Motivation
• **Need For Similarity-based Dedup**
• Our Approach: dbDedup
• Evaluation
Duplication Beyond A Block

• Application-level versioning
  – Few apps take advantage of sys-level versioning
  – Revisions of data items seen as unrelated records
  – E.g., WordPress, Wikipedia

• Partial record copying
  – Inclusion between records not exposed to the DBMS
  – E.g., email reply/forward, quotes in message boards
Traditional Dedup: Ideal Case

- Chunk Boundary
- Modified Region
- Duplicate Chunk

Incoming Data

{BYTE STREAM}
Traditional Dedup: Ideal Case

| Chunk Boundary | Modified Region | Duplicate Chunk |

Incoming Data

Database
Traditional Dedup: Ideal Case

Chunk Boundary  Modified Region  Duplicate Chunk

Incoming Data

1 2 3 4 5
Traditional Dedup: Ideal Case

Chunk Boundary  Modified Region  Duplicate Chunk

Incoming Data

| 1 | 2 | 3 | 4 | 5 |

Deduped Data

| 1 | 2 | 3 | 4 | 5 |

Store and send dedup’ed data
Dedup: Common Case in DBs

- Chunk Boundary
- Modified Region
- Duplicate Chunk

Incoming Data
Dedup: Common Case in DBs

Chunk Boundary  | Modified Region  | Duplicate Chunk

Incoming Data

Deduped Data
Dedup: Common Case in DBs

- Chunk Boundary
- Modified Region
- Duplicate Chunk

Incoming Data

 Deduped Data

Store and send almost the entire record
Similarity Dedup (dbDedup)

- Chunk Boundary
- Modified Region
- Duplicate Chunk

Incoming Data

Delta!
Similarity Dedup (dbDedup)

Chunk Boundary  Modified Region  Duplicate Chunk

Incoming Data

Dedup’ed Data

Only store and send delta
Compress vs. Dedup

Wikipedia Database (20GB sample)
MongoDB v3.1 w/ Snappy Compression
Outline

• Goal and Motivation
• Need for Similarity-based Dedup
• **Our Approach: dbDedup**
• Evaluation
dbDedup Encoding Steps

• For each new record:
  – Identify Similar Records
  – Select the “Best” Match
  – Delta Compression
Identify Similar Records

New Record

Rabin Chunking

Consistent Sampling

Similarity Sketch

Feature Index Table

Candidate Records

Similarity Score

Rec #1

Rec #2

Rec #3

Rec #1

Rec #2

Rec #3
Select the Best Match

Initial Ranking

<table>
<thead>
<tr>
<th>Rank</th>
<th>Candidates</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rec #2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>Rec #3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Rec #1</td>
<td>1</td>
</tr>
</tbody>
</table>
Select the Best Match

Initial Ranking

<table>
<thead>
<tr>
<th>Rank</th>
<th>Candidates</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rec #2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>Rec #3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Rec #1</td>
<td>1</td>
</tr>
</tbody>
</table>

Is record cached?

Source Record Cache
## Select the Best Match

**Initial Ranking**

<table>
<thead>
<tr>
<th>Rank</th>
<th>Candidates</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rec #2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>Rec #3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Rec #1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Final Ranking**

<table>
<thead>
<tr>
<th>Rank</th>
<th>Candidates</th>
<th>Cached?</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rec #3</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>Rec #1</td>
<td>Yes</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Rec #2</td>
<td>No</td>
<td>2</td>
</tr>
</tbody>
</table>

*Is record cached?*

*Cache-aware selection*

*If yes, reward +2*
Delta Compression

• Based on the “xDelta” algorithm
  – Byte-level diff between source and target records

• Challenge:
  – Reading encoded records incurs decoding overhead

• Solution: new encoding techniques
  – Reduce both freq. and cost of decode
New Encoding Techniques

- Two-way encoding
  - *Forward encoding* for network-level dedup
    - Sending encoding of new records against source
    - Replicas need to use existing record as source
  - *Backward encoding* for storage-level dedup
    - Encode/rewrite source; write unencoded new record
    - Minimize read overhead for the most recent records

- Hop encoding
  - *Reduce worst-case re-construction time*
  - *Maintain compression ratio*
Two-way Encoding

Write order

Forward encoding

\[ R_1 \]

\[ R_1 + \Delta_{1,2} \]

\[ R_2 + \Delta_{2,3} \]

\[ R_2 \]

\[ R_3 \]
Two-way Encoding

Write order

Forward encoding

\[ R_1 \rightarrow R_1 + \Delta_{1,2} \rightarrow R_1 + \Delta_{1,2} + \Delta_{2,3} \]

Backward encoding

\[ R_2 \rightarrow R_2 + \Delta_{2,1} \rightarrow R_2 + \Delta_{2,1} + \Delta_{2,3} \]
Two-way Encoding

Forward encoding

Write order

$R_1$  

$R_1$  

$Δ_{1,2}$  

$R_1 + Δ_{1,2}$  

$Δ_{2,3}$  

$R_2 + Δ_{2,3}$  

Backward encoding

$R_2$  

$Δ_{2,1}$  

$R_2 + Δ_{2,1}$  

$Δ_{1,2}$  

$R_1$  

$R_1$  

$R_2$  

$R_3$
Two-way Encoding

Write order

Forward encoding

Backward encoding

$R_1$ $Δ_{1,2}$ $R_1 + Δ_{1,2}$ $Δ_{2,3}$ $R_2 + Δ_{2,3}$

$R_2$ $Δ_{2,1}$ $R_2 + Δ_{2,1}$

$R_3$ $Δ_{3,2}$ $R_3 + Δ_{3,2}$
Two-way Encoding

Write order

Forward encoding

$R_1$

$\Delta_{1,2}$

$R_1 + \Delta_{1,2}$

$\Delta_{2,3}$

$R_2 + \Delta_{2,3}$

Backward encoding

$R_1$

$\Delta_{2,1}$

$R_2 + \Delta_{2,1}$

$\Delta_{3,2}$

$R_3 + \Delta_{3,2}$

$R_3$
Two-way Encoding

Write order

Forward encoding

Backward encoding

Network-level dedup

Storage-level dedup
Hop Encoding

Write order

Backward encoding

Version jumping
Hop Encoding

Write order: $R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_4 \rightarrow R_5$

Backward encoding:
- $\Delta_{2,1} \rightarrow \Delta_{3,2} \rightarrow \Delta_{4,3} \rightarrow \Delta_{5,4} \rightarrow R_5$

Version jumping:
- $R_1 \rightarrow \Delta_{3,2} \rightarrow R_3 \rightarrow \Delta_{5,4} \rightarrow R_5$

Hop encoding:
- $\Delta_{3,1} \rightarrow \Delta_{3,2} \rightarrow \Delta_{5,3} \rightarrow \Delta_{5,4} \rightarrow R_5$

Hop distance:
- $R_1 \rightarrow \Delta_{3,2} \rightarrow R_3 \rightarrow \Delta_{5,4} \rightarrow R_5$

Caching Mechanisms

• Unique challenge for delta compression
  – Extra disk I/Os to read/update records

• Solution: specialized caches
  – Source Record cache
    • Reduce disk reads to source records
    • Keep only the latest version of a record
  – Lossy write-back cache
Lossy Write-back Cache

• Reduce overhead of writing backward-encoded sources
  – *Delay updates until system is idle*
  – *Prioritize updates based on space savings*

• “Lossy” property
  – *Unapplied updates can be safely discarded*
  – *Records remain un-encoded in such case*
Integration into DBMS

Client

Inserts & Updates

Database

Primary Node

Oplog

Secondary Node
Integration into DBMS

Client

Inserts & Updates

Oplog

Primary Node

Database

Secondary Node

Oplog

Oplog syncer
Integration into DBMS

Client

Inserts & Updates

Oplog

Source cache

Scanner

dbDedup Encoder

Writeback cache

Database

Backward-encoded data

Primary Node

Oplog

Oplog syncer

Database

Secondary Node
Integration into DBMS

Primary Node

Client

Reads

Inserts & Updates

dbDedup Decoder

Source cache

dbDedup Encoder

Database

Writeback cache

Backward-encoded data

Secondary Node

Reads

Oplog syncer

Olog

Forward-Encoded data

dbDedup Decoder

Source cache

dbDedup Re-Encoder

Database

Writeback cache
Outline

• Goal and Motivation
• Need for Similarity-based Dedup
• Our Approach: dbDedup
• Evaluation
Evaluation

• MongoDB (v3.1)
  – 1 primary, 1 secondary node, 1 client
  – Node Config: 4 cores, 8GB RAM, 100GB HDD storage

• Datasets:
  – Wikipedia dump (20GB out of ~12TB)
  – Additional datasets evaluated in the dissertation
Compression Ratio

- **Compression from dedup alone**
- **Additional compression from Snappy**

The diagram shows the compression ratio for different data sizes and methods:
- **1KB dbDedup**
- **64B**
- **4KB trad-dedup**
- **64B**
- **Snappy**

The compression ratios vary across these methods, with significant differences observed between them.
Comp. Ratio & Index Memory

![Bar chart showing compression ratio and index memory usage for different data sizes and methods.](image)

- **Compression from dedup alone**
- **Additional compression from Snappy**
- **Index memory usage**

**Data Sizes and Methods**
- **1KB**: dbDedup
- **64B**: trad-dedup
- **64B**: Snappy

**Compression Ratio**
- Range: 0 to 90

**Index Memory Usage (MB)**
- Range: 0 to 1200
Effect of Source Cache

Normalized compression ratio
Cache miss ratio

Normalized comp. ratio

No cache

Reward score

0
2
4
8

0
0.2
0.4
0.6
0.8
1.0

0
0.2
0.4
0.6
0.8
1.0

Cache miss ratio
Effect of Lossy Write-back

Throughput (ops)

Time (seconds)

- dbDedup (w/o write-back cache)
- dbDedup (w/ write-back cache)
Conclusion

• **dbDedup**: Similarity-based dedup for DBMSs
  – First DB dedup system for both storage and network layers
  – Much greater data reduction than traditional dedup
  – Up to 38x compression ratio for Wikipedia
  – Resource-efficient design with negligible overhead