Reducing Replication Bandwidth for Distributed Document Databases

Lianghong Xu¹, Andy Pavlo¹, Sudipta Sengupta²
Jin Li², Greg Ganger¹
Carnegie Mellon University¹, Microsoft Research²
Today I am visiting @eliothorowitz at @mongodbinc to try to convince them to ditch MMAP & switch to anti-caching.
#1 – You can sleep with grad students but not undergrads.
#2 - Keep a bottle of water in your office in case a student breaks down crying.
#3 – Kids love MongoDB, but they want to go work for Google.
## System Votes

<table>
<thead>
<tr>
<th>Database</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Spanner</td>
<td>24</td>
</tr>
<tr>
<td>MongoDB</td>
<td>23</td>
</tr>
<tr>
<td>Redis</td>
<td>10</td>
</tr>
<tr>
<td>Amazon DynamoDB</td>
<td>5</td>
</tr>
<tr>
<td>MySQL</td>
<td>2</td>
</tr>
<tr>
<td>Apache Hbase</td>
<td>1</td>
</tr>
<tr>
<td>DB Shards</td>
<td>1</td>
</tr>
</tbody>
</table>
Reducing Replication Bandwidth for Distributed Document Databases

In ACM Symposium on Cloud Computing, pg. 1-12, August 2015.

More Info:
http://cmudb.io/doc-dbs
Replication Bandwidth

Primary Database

Operation logs

Secondary

WAN

Operation logs

MMS
Replication Bandwidth

Goal: Reduce bandwidth for WAN geo-replication.
Why Deduplication?

• Why not just **compress**?
  – *Oplog batches are small and not enough overlap.*

• Why not just use **diff**?
  – *Need application guidance to identify source.*

• **Deduplication** finds and removes redundancies.
Traditional Dedup

Chunk Boundary  Modified Region  Duplicate Region

Incoming Data

Deduped Data

Send deduped data to replicas.
Traditional Dedup

Incoming Data

Deduped Data

Chunk Boundary

Modified Region

Duplicate Region

Must send the entire document.
Only send delta encoding.
Compress vs. Dedup

20GB sampled Wikipedia dataset.
MongoDB v2.7 // 4MB Oplog batches
sDedup: Similarity Dedup

Client

Insertion & Updates

Oplog

Source documents

Database

Oplog

Source documents

Database

Unsynchronized oplog entries

Delta Compressor

Deduplicated oplog entries

Secondary Node

Delta Decompressor

Re-constructed oplog entries

Replay

Oplog syncer

Source documents

Primary Node

Insertion & Updates

Oplog

Source documents

Database

Source Document Cache

Oplog batch

Deduplicated oplog entries

Re-constructed oplog entries
Encoding Steps

• Identify Similar Documents
• Select the Best Match
• Delta Compression
Identify Similar Documents

Target Document

Rabin Chunking

Consistent Sampling

Similarity Sketch

Feature Index Table

Candidate Documents

Similarity Score

Doc #1

Doc #2

Doc #3

Doc #2

Doc #3

Doc #3
## Selecting 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>Doc #2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>Doc #3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Doc #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>Doc #3</td>
<td>Yes</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>Doc #1</td>
<td>Yes</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Doc #2</td>
<td>No</td>
<td>2</td>
</tr>
</tbody>
</table>

*Is doc cached? If yes, reward 3x*
Delta Compression

- Byte-level diff between source and target docs:
  - Based on the xDelta algorithm
  - Improved speed with minimal loss of compression

- **Encoding:**
  - Descriptors about duplicate/unique regions + unique bytes

- **Decoding:**
  - Use source doc + encoded output
  - Concatenate byte regions in order
Evaluation

• MongoDB setup (v2.7)
  – 1 primary, 1 secondary node, 1 client
  – Node Config: 4 cores, 8GB RAM, 100GB HDD storage

• Datasets:
  – Wikipedia dump (20GB out of ~12TB)
  – Stack Exchange data dump (10GB out of ~100GB)

<table>
<thead>
<tr>
<th>Chunk Size</th>
<th>sDedup</th>
<th>trad-dedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB</td>
<td>9.9</td>
<td>2.3</td>
</tr>
<tr>
<td>1KB</td>
<td>26.3</td>
<td>4.6</td>
</tr>
<tr>
<td>256B</td>
<td>38.4</td>
<td>9.1</td>
</tr>
<tr>
<td>64B</td>
<td>38.9</td>
<td>15.2</td>
</tr>
</tbody>
</table>

*20GB sampled Wikipedia dataset*

- **sDedup**
- **trad-dedup**

<table>
<thead>
<tr>
<th>Chunk Size</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB</td>
<td>34.1</td>
</tr>
<tr>
<td>1KB</td>
<td>47.9</td>
</tr>
<tr>
<td>256B</td>
<td>57.3</td>
</tr>
<tr>
<td>64B</td>
<td>61.0</td>
</tr>
<tr>
<td>20GB sampled Wikipedia dataset</td>
<td><strong>780.5</strong></td>
</tr>
</tbody>
</table>
Compression: StackExchange

10GB sampled StackExchange dataset

<table>
<thead>
<tr>
<th>Chunk Size</th>
<th>sDedup</th>
<th>trad-dedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1KB</td>
<td>1.2</td>
<td>1.0</td>
</tr>
<tr>
<td>256B</td>
<td>1.3</td>
<td>1.1</td>
</tr>
<tr>
<td>64B</td>
<td>1.8</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Dedup + Sharding

Compression Ratio

# of Shards

20GB sampled Wikipedia dataset
Failure Recovery

20GB sampled Wikipedia dataset.

Failure Point
Conclusion

• Similarity-based deduplication for replicated document databases.

• sDedup for MongoDB (v2.7)
  – Much greater data reduction than traditional dedup
  – Up to 38x compression ratio for Wikipedia
  – Resource-efficient design for inline deduplication with negligible performance overhead
What’s Next?

• Port code to MongoDB v3.1
• Integrating sDedup into WiredTiger storage manager.
• Need to test with more workloads.
• Try not to get anyone pregnant.
## WiredTiger vs. sDedup

<table>
<thead>
<tr>
<th></th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snappy</td>
<td>1.6x</td>
</tr>
<tr>
<td>zLib</td>
<td>3.0x</td>
</tr>
<tr>
<td>sDedup (no compress)</td>
<td>38.4x</td>
</tr>
<tr>
<td>sDedup + Snappy</td>
<td>60.8x</td>
</tr>
<tr>
<td>sDedup + zLib</td>
<td>114.5x</td>
</tr>
</tbody>
</table>

*20GB sampled Wikipedia dataset.*