A Robust Fault-Tolerant and Scalable Cluster-wide Data Deduplication for Shared-Nothing Storage Systems

Youngjae Kim (PhD)

Sogang University, Seoul
Republic of Korea

Laboratory for Advanced System Software
Why is Deduplication Important?

- Massive data explosion in recent years and expected to grow
  - 2007, 281 EB
  - 2010, 1.2 ZB
  - 2013, 4.4 ZB
  - 2020, ~44 ZB

- Opportunity
  - Data Redundancy widely exists in primary datasets
  - Deduplication can reduce storage cost
Deduplication in Cloud and Distributed Storages

- A simple way to accomplish storage efficiency is;
  - centralized deduplication servers
Deduplication101: Write Flow

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O
4. Update Dedup Metadata
5. Redirect Chunks to Storage
Deduplication 101: Duplicate Write Flow

1. Chunking

2. Fingerprinting

3. Duplicate Lookup I/O

4. Update Dedup Metadata

5. Redirect Chunks to Storage
Deduplication101: Duplicate Write Flow

1. Chunking

2. Fingerprinting

3. Duplicate Lookup I/O

4. Update Dedup Metadata

5. Redirect Chunks to Storage
Deduplication 101: Duplicate Write Flow

1. Chunking

2. Fingerprinting

3. Duplicate Lookup I/O

4. Update Dedup Metadata

5. Redirect Chunks to Storage

Duplicate pointers refer to original copies
Agenda

- Introduction
- Background
- Global Deduplication
  - Centralized and Decentralized
- Challenges
- Proposed Ideas
- Evaluation
- Summary
Global Deduplication with Centralized Servers

Clients

Storage Servers

OSD 1

OSD N

...
Global Deduplication with Centralized Servers

Clients → Storage Servers

Dedicated Deduplication Servers

Responsibility:
- Chunking
- Fingerprinting
- Fingerprint Lookup IOs
- Dedup Metadata Update IOs
- IO Redirection
Global Deduplication with Centralized Servers

Clients

All dedup I/O traffic (Lookup IOs, Reference updates, New fingerprint entries)

All data storage traffic

Dedicated Deduplication

Object Storage Servers

OSD 1

OSD N

...
Global Deduplication with Centralized Servers

Clients

Object Storage Servers

Problems/Limitations

• violates the shared nothing property of storage systems
  • Ceph, Glustre
• Data un-availability
• Metadata inconsistencies
• High I/O traffic
• Client congestion
• Deduplication efficiency drops due to lost fingerprints

All dedup I/O traffic

All data storage traffic
Global Deduplication with Decentralized Servers
Decentralization solve all the problems ....

Let’s investigate with Ceph
Ceph: Distributed Object Storage System

- Scalable
  - No centralized metadata bottleneck
- Autonomous
  - Sustains single-node failures
- Cluster expansion
  - Dynamic node addition and removal
- Evenly balanced I/O and Storage
  - Load-aware I/O redirection and data storage
Challenge I: I/O Broadcasting Overhead

Bar Object

I/O from Client to Storage Server

1. Chunking
Challenge I: I/O Broadcasting Overhead

1. Chunking

2. Fingerprinting

Bar Object
Challenge I: I/O Broadcasting Overhead

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O

Bar Object

Chunk Fingerprint Matching
Challenge I: I/O Broadcasting Overhead

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O

Broadcasting FPs to check duplicates

Storage Servers
Challenge I: I/O Broadcasting Overhead

1. Chunking

2. Fingerprinting

3. Duplicate Lookup I/O

Broadcasting FPs to check duplicates

Storage Servers

Dedup Metadata

Dedup Metadata

Dedup Metadata

Dedup Metadata
Challenge I: I/O Broadcasting Overhead

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O

Challenge I: How to minimize I/O Broadcasting Overhead?
Challenge II: Partial Transaction Failure

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O
4. Update Dedup Metadata

Storage Servers

Dedup Metadata
Dedup Metadata
Dedup Metadata
Dedup Metadata
Challenge II: Partial Transaction Failure

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O
4. Update Dedup Metadata

Location bound dedup approaches fail to validate chunk location in case of content shuffling
Challenge II: Partial Transaction Failure

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O

Causes several problems

1. Data lost in case of duplicate chunk arrival
2. Incorrect reference count value (RFC)

Challenge II

How to repair partial transaction failures?

Deduplication Metadata

<table>
<thead>
<tr>
<th>Object</th>
<th>Chk_Offset</th>
<th>FP/Hash</th>
<th>Loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar</td>
<td>0</td>
<td>A</td>
<td>SS1</td>
</tr>
<tr>
<td>Bar</td>
<td>E</td>
<td>E</td>
<td>SS2</td>
</tr>
<tr>
<td>Bar</td>
<td>F</td>
<td>F</td>
<td>SS3</td>
</tr>
<tr>
<td>Bar</td>
<td>D</td>
<td>D</td>
<td>SS4</td>
</tr>
<tr>
<td>Bar</td>
<td>3</td>
<td>D</td>
<td>SS4</td>
</tr>
</tbody>
</table>

fingerprint entries with no associated data chunks

Storage Servers (SS)
SS1
SS2
SS3
SS4
Challenge III: Dynamic Content Relocation

Scaling-out Storage

Addition of Storage Server
Challenge III: Dynamic Content Relocation

Triggers Storage Rebalancing or Content Relocation

Utilization Ratio of all Servers: 3:3:0
Challenge III: Dynamic Content Relocation

Previous and new Location Update I/Os for each relocated victim chunk

Challenge III

How to address such dynamic relocation?
Our approaches

1. Content Fingerprint-based Redirection
2. Distributed DM-Shard
3. Async-Tagged Consistency
Proposed Idea I: Content Fingerprint-based Redirection

1. Chunking

2. Fingerprinting

3. Duplicate Lookup I/O

- Content Fingerprint-based Lookup I/O

- CRUSH (A) = SS 1

- e.g., Ceph Object Storage

- Dedup Service

- Storage Servers
Proposed Idea II: Distributed DM-Shard

1. Chunking
   - Bar Object
   - Object Map (OMAP)
     - Object | Object FP | Chunk List
     - Bar     | 0xAB2     | A,E,F,D

2. Fingerprinting
   - Chunk Fingerprint
   - Matching

3. Duplicate Lookup I/O
   - Add/Update Dedup Metadata

4. Update Dedup Metadata

Deduplication Metadata Shard (DM-Shard)

OMAP

Maintains only chunk hashes & not location bound

Storage Servers
Proposed Idea II: Distributed DM-Shard

1. Chunking

2. Fingerprinting

3. Duplicate Lookup I/O

4. Update Dedup Metadata

Chunk Information Table

<table>
<thead>
<tr>
<th>Chunk FP</th>
<th>RFC</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>Valid</td>
</tr>
</tbody>
</table>
Proposed Idea III: Async-Tagged Consistency
Proposed Idea III: Async-Tagged Consistency

- Use case I: Unique Data Chunk

1. Match Chunk FP

2. Add entry with Invalid Flag

- No

Chunk Information Table

- Chunk FP
- RFC
- Flag

- Async-Tagged Consistency
- Async-Thread
- I/O ends, Async-thread switches Flag

- Store Chunk Data
- Ack to Storage Server

- Register FP to Async-Thread

Use case I: Unique Data Chunk
Proposed Idea III: Async-Tagged Consistency

- Use case II: Duplicate Chunk with Valid Flag

<table>
<thead>
<tr>
<th>Chunk FP</th>
<th>RFC</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>Valid</td>
</tr>
</tbody>
</table>

2. Check CIT Flag

- Valid

3. Reference Count Update

Duplicate exists!
Proposed Idea III: Async-Tagged Consistency

Use case III: Duplicate Chunk with an Invalid Flag

Chunk Information Table

<table>
<thead>
<tr>
<th>Chunk FP</th>
<th>RFC</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>Valid</td>
</tr>
</tbody>
</table>

2. Check CIT Flag

3. Reconciliation Check

5. Switch Flag & RFC

Duplicate exists!
Proposed Idea III: Async-Tagged Consistency

Chunk Information Table

<table>
<thead>
<tr>
<th>Chunk FP</th>
<th>RFC</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>Valid</td>
</tr>
</tbody>
</table>

Storage Servers
**Proposed Idea III: Async-Tagged Consistency**

1. **Chunking**
   - Bar Object

2. **Fingerprinting**
   - Chunk Fingerprint Matching
   - OM - CIT

3. **Duplicate Lookup I/O**
   - Add/Update Dedup Metadata

4. **Update Dedup Metadata**

5. **Ack to Storage Server**

<table>
<thead>
<tr>
<th>Object</th>
<th>Object FP</th>
<th>Chunk List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar</td>
<td>0xAB2</td>
<td>A,E,F,D</td>
</tr>
</tbody>
</table>
Experimental Setup

• Ceph v10.2.3
• Cluster setup
   • 7 Ceph OSD Servers, 3 Ceph Monitors, 4 Ceph Clients
   • 2 x 256GB Samsung SSD/OSD Server, 32GB DRAM, 10Gbps network
• FIO benchmark
   • 500GB of synthetic write I/O workload, object size 4MB
• Comparison
  • Baseline Ceph: Ceph with no Deduplication
  • DB-Shard Dedup: Ceph with DB-shard deduplication and no fingerprint based Redirection
Result 1: Varying Chunk Size

Performance Analysis

<table>
<thead>
<tr>
<th>Chunk Size (KB)</th>
<th>Baseline-Ceph</th>
<th>DB-Shard Dedup</th>
<th>Proposed Dedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>58%</td>
<td>27%</td>
<td>27%</td>
</tr>
<tr>
<td>8</td>
<td>65%</td>
<td>27%</td>
<td>27%</td>
</tr>
<tr>
<td>64</td>
<td>61%</td>
<td>18%</td>
<td>18%</td>
</tr>
<tr>
<td>128</td>
<td>57%</td>
<td>16%</td>
<td>16%</td>
</tr>
<tr>
<td>256</td>
<td>53%</td>
<td>14%</td>
<td>14%</td>
</tr>
<tr>
<td>512</td>
<td>50%</td>
<td>12%</td>
<td>12%</td>
</tr>
<tr>
<td>1024</td>
<td>44%</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>4096</td>
<td>32%</td>
<td>7%</td>
<td>7%</td>
</tr>
</tbody>
</table>
Result 2: Varying Deduplication Ratio

Performance Analysis

- Baseline-Ceph
- DB-Shard Dedup
- Proposed Dedup

Bandwidth (MB/s) vs. Deduplication Ratio

- 0% Deduplication Ratio: 12%
- 20% Deduplication Ratio: 50%
- 40% Deduplication Ratio: 12%
- 60% Deduplication Ratio: 7%
- 80% Deduplication Ratio: 42%
- 100% Deduplication Ratio: 42%
Result 3: Varying Client Threads

Performance Analysis

<table>
<thead>
<tr>
<th>Number of Client Threads</th>
<th>Baseline-Ceph</th>
<th>DB-Shard Dedup</th>
<th>Proposed Dedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>610MB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1070MB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>610MB/s</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1104MB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>313MB/s</td>
<td></td>
</tr>
</tbody>
</table>
Result 4: Async-Tagged Consistency

Tagged Consistency Analysis

Bandwidth (MB/s) vs. Chunk Size (KB)

- Proposed Dedup
- Object-based Sync
- Chunk-based Sync
- Proposed Dedup-Async

- 63%
- 20%
- 2%
- 11%
- 31%
- 1.7%
Summary

- Cluster-wide Deduplication framework with SN-SS design constraints
- Content Fingerprint-based I/O Redirection
- Distributed Dedup Metadata via DB-Sharding
- Asynchronous tagged consistency
Questions?

Youngjae Kim
youkim@sogang.ac.kr
Laboratory for Advanced System Software
Sogang University, Seoul,
Republic of Korea