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

Awais Khan, Chang-Gyu Lee, Prince Hamandawana, Sungyong Park and Youngjae Kim

Sogang University, Seoul
Republic of Korea
Why is Deduplication Important?

- Massive data explosion in recent years and expected to grow

<table>
<thead>
<tr>
<th>Year</th>
<th>Storage Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>281 EB</td>
</tr>
<tr>
<td>2010</td>
<td>1.2 ZB</td>
</tr>
<tr>
<td>2013</td>
<td>4.4 ZB</td>
</tr>
<tr>
<td>2020</td>
<td>~44 ZB</td>
</tr>
</tbody>
</table>

- Opportunity
  - Data Redundancy widely exists in primary datasets
  - Deduplication can reduce storage cost

Growing Capacity Demands
Deduplication101: Object Write Flow

1. Chunking

2. Fingerprinting

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

4. Update Dedup Metadata

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

1. Chunking

2. Fingerprinting

3. Duplicate Lookup I/O

4. Update Dedup Metadata

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

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O
4. Update Dedup Metadata
5. Redirect Chunks to Storage
Deduplication101: Duplicate Object Write Flow

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O
4. Update Dedup Metadata
5. Redirect Chunks to Storage
Deduplication101: Duplicate Object Write Flow

1. Chunking
   - Foo Object
   - Bar Object

2. Fingerprinting
   - A B C D
   - A E F D

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

4. Update Dedup Metadata
   - Store Unique Chunks

5. Redirect Chunks to Storage
   - Duplicate pointers refer to original copies
Agenda

- Introduction
- Background
- Global Deduplication
  - Centralized and Decentralized
- Challenges: Shared-Nothing Architecture
- Proposed Ideas
- Evaluation
- Summary
Global Deduplication with Centralized Servers

Clients

Object Storage Servers

OSD 1

...
Global Deduplication with Centralized Servers

Clients

Object Storage Servers

Dedicated Deduplication Servers

OSD 1

...
Global Deduplication with Centralized Servers

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

- **Foo Object**
  - Chunking: A B C D
  - Fingerprinting: A B C D
  - Deduplication Server
  - I/O from Dedup Server to Storage Server

- **Bar Object**
  - Chunking: A E F D
  - Fingerprinting: A B C D
  - Deduplication Server
  - I/O from Dedup Server to Storage Server

Store Unique Chunks

Add/Update Dedup Metadata

Chunk Fingerprint Matching
Global Deduplication with Centralized Servers

Clients

All dedup I/O traffic

All data storage traffic

Object Storage Servers

Dedicated Deduplication

OSD 1

...
Global Deduplication with Centralized Servers

Problems
- Data un-availability
- Metadata inconsistencies
- High I/O traffic
- Client congestion
- Low deduplication efficiency

Clients

Object Storage Servers

All dedup I/O traffic

All data storage traffic
Global Deduplication with Decentralized Servers

Clients

Object Storage Servers

Dedicated Deduplication Servers

Hash-based Data Placement
  e.g., Ceph’s CRUSH, Gluster’s DHT
Decentralization solve all the problems ....
Challenge I: I/O Broadcasting Overhead

1. Chunking

Bar Object

I/O from Client to Storage Server
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

Dedup Metadata
Dedup Metadata
Dedup Metadata
Dedup Metadata

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

How to minimize I/O Broadcasting Overhead?

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

Challenge I

Bar Object

Storage Servers
Bar Object

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O
   - Chunk Fingerprint Matching

Dedup Metadata

Add/Update Dedup Metadata

4. Update Dedup Metadata

Storage Servers

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

Bar Object

Deduplication Metadata

<table>
<thead>
<tr>
<th>Object</th>
<th>Chk_Offset</th>
<th>FP/Hash</th>
<th>Loc</th>
<th>RFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar</td>
<td>0-4</td>
<td>A</td>
<td>SS1</td>
<td>1</td>
</tr>
<tr>
<td>Bar</td>
<td>4-8</td>
<td>E</td>
<td>SS2</td>
<td>1</td>
</tr>
<tr>
<td>Bar</td>
<td>8-12</td>
<td>F</td>
<td>SS3</td>
<td>1</td>
</tr>
<tr>
<td>Bar</td>
<td>12-16</td>
<td>D</td>
<td>SS4</td>
<td>1</td>
</tr>
</tbody>
</table>

1. Chunking

2. Fingerprinting

3. Duplicate Lookup I/O

4. Update Dedup Metadata

Storage Servers
Challenge II: Partial Transaction Failure

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

Challenge II
How to ensure dedup transaction & metadata consistency?
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

How to address such dynamic relocation?
Proposed Idea I: Content Fingerprint-based Redirection

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

No Random or Blind Lookup I/O
Proposed Idea I: Content Fingerprint-based Redirection

1. Chunking

2. Fingerprinting

3. Duplicate Lookup I/O

Content Fingerprint-based Lookup I/O

Redirect (A) = SS 1
Proposed Idea I: Content Fingerprint-based Redirection

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

Content Fingerprint-based Lookup I/O

Dedup Service

Storage Servers

CRUSH (A) = SS 1
Proposed Idea I: Content Fingerprint-based Redirection

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

DHT (A) = SS 1

Content Fingerprint-based Lookup I/O

Storage Servers
Proposed Idea II: Distributed DM-Shard

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

Bar Object

Object Map (OMAP)

<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>

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>

Dedup Service

Dedup Service

Dedup Service

Dedup Service

Storage Servers

Dedup Metadata Shard (DM-Shard)

OMAP

CIT

Chunk Fingerprint Matching

Add/Update Dedup Metadata
Proposed Idea II: Distributed DM-Shard

1. Chunking
2. Fingerprinting
3. Duplicate Lookup I/O
   - Chunk Fingerprint Matching
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

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>

1. Match Chunk FP
2. Check CIT Flag
3. Reference Count Update

Duplicate Chunk with Valid Flag
Proposed Idea III: Async-Tagged Consistency

1. Match Chunk FP
2. Check CIT Flag
3. Reconciliation Check
4. Verify Chunk Data
5. Switch Flag & RFC

Duplicate Chunk with Invalid Flag

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

1. Match Chunk FP
2. Add entry with Invalid Flag
3. Register FP to Async-Thread
4. I/O ends, Async-thread switches Flag
5. Ack to Storage Server

Unique Chunk
Proposed Idea III: Async-Tagged Consistency

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

Bar Object

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>

Dedup Service

Dedup Metadata Shard (DM-Shard)

Chunk Fingerprint Matching

Add/Update Dedup Metadata

Ack to Storage Server

Storage Servers
Proposed Idea III: Async-Tagged Consistency

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

Object Map (OMAP)

<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>

Ack to Storage Server

Storage Servers

Dedup Service
Dedup Service
Dedup Service
Dedup Service
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 KB</td>
<td>58%</td>
<td>65%</td>
<td>61%</td>
</tr>
<tr>
<td>8 KB</td>
<td>27%</td>
<td>27%</td>
<td>18%</td>
</tr>
<tr>
<td>64 KB</td>
<td>61%</td>
<td>16%</td>
<td>18%</td>
</tr>
<tr>
<td>128 KB</td>
<td>14%</td>
<td>16%</td>
<td>18%</td>
</tr>
<tr>
<td>256 KB</td>
<td>53%</td>
<td>12%</td>
<td>16%</td>
</tr>
<tr>
<td>512 KB</td>
<td>50%</td>
<td>10%</td>
<td>14%</td>
</tr>
<tr>
<td>1024 KB</td>
<td>44%</td>
<td>10%</td>
<td>12%</td>
</tr>
<tr>
<td>4096 KB</td>
<td>7%</td>
<td>7%</td>
<td>10%</td>
</tr>
</tbody>
</table>
Result 2: Varying Deduplication Ratio

Performance Analysis

Bandwidth (MB/s)

Deduplication Ratio

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

Result 2: Varying Deduplication Ratio

- 50% Deduplication Ratio: 12%
- 70% Deduplication Ratio: 7%
- 80% Deduplication Ratio: 42%

42
Result 3: Varying Client Threads

Performance Analysis

Number of Client Threads

Bandwidth (MB/s)

Baseline-Ceph  DB-Shard Dedup  Proposed Dedup

Result 3: Varying Client Threads

- Baseline-Ceph
  - 2 threads: 610MB/s
  - 4 threads: 1070MB/s
  - 8 threads: 1104MB/s

- DB-Shard Dedup
  - 2 threads: 313MB/s
  - 4 threads: 610MB/s

- Proposed Dedup
  - 2 threads: 313MB/s
  - 4 threads: 610MB/s
  - 8 threads: 1070MB/s
  - 16 threads: 1104MB/s
  - 32 threads: 1104MB/s
Result 4: Async-Tagged Consistency

Tagged Consistency Analysis

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

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

Bandwidth values are shown for different chunk sizes, with percentage improvements noted for each size.
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?

Awais Khan
awais@sogang.ac.kr
Laboratory for Advanced System Software
Sogang University, Seoul,
Republic of Korea
Backup Slides ...
Result 4: Varying DB-Shards

Fingerprint-based Lookup I/O Scalability

- Normalized Lookup I/Os
- Number of DB-Shards
- DB-Shard Dedup
- Proposed Dedup