TiDedup: A New Distributed Deduplication Architecture for Ceph

Myoungwon Oh, Sungmin Lee, Samuel Just, Young Jin Yu, Duck-Ho Bae
Sage Weil, Sangyeun Cho, Heon Y. Yeom
Samsung Electronics  IBM  Ceph Foundation  Seoul National University
Conventional (local) deduplication system

1. Divide an object into chunks using chunk algorithm
2. Generate fingerprint from the chunk
3. Find out the location of chunks

Hash(Cx) = fingerprint
We had a discussion on how deduplication can be implemented on Ceph. The primary concern is how Ceph manage fingerprint index at scale

- How to look up a pair < fingerprint : address >
- How to distribute fingerprint index entry evenly

Fingerprint index ( fingerprint : address)

<table>
<thead>
<tr>
<th>Hash(C0)</th>
<th>Hash(C2)</th>
<th>Hash(C3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

How can we ensure the scalability?

Our prior solution is double hashing !!
What is double hashing?

- Use existing data placement algorithm once again
- Completely remove fingerprint index itself
Background (prior work)

What is double hashing?

- Use existing data placement algorithm once again
- Completely remove fingerprint index itself
What is double hashing?

- Use existing data placement algorithm once again
- Completely remove fingerprint index itself
What is double hashing?

- Use existing data placement algorithm once again
- Completely remove fingerprint index itself

Metadata and chunk objects are equal to existing object
More details in the paper!
Motivation

We made a prototype version of cluster-level deduplication, but the following challenges arose:

1. *Is deduplication really helpful?*

2. *Structure limitations: performance degradation and inefficient chunking*

3. *Snapshot and scrub (reference GC) overhead*
Is deduplication always helpful?

- Ceph is a general-purpose storage system

![Diagram of Block service, File service, and Object service connected by Ceph RADOS and Deduplication Logic]

- Deduplication penalty: If objects have unique content, the storage space is wasted, if anything.

![Graph showing increased metadata size (MB) and metadata size written data size (%) vs. chunk size (4KB, 8KB, 16KB, 32KB, 64KB, 128 KB)]
Is deduplication always helpful?

- Ceph is a general-purpose storage system

  - Block service
  - File service
  - Object service

  Ceph.RADOS

  Deduplication Logic

  Workload differs in the services: Dedup could or could not be helpful

- Deduplication penalty: If objects have unique content, the storage space is wasted, if anything.

  - Increased Metadata Size (MB)  
  - Metadata Size / Written Data Size (%)

  Metadata size added for deduplication increases significantly
Structure limitations

Performance degradation

User I/Os  Replication  Recovery  Scrub

Enumeration

< Object Service Daemon >

Chunking

Move chunks

< Object Service Daemon >
1. The background thread for deduplication job disturbs high-priority I/Os as the number of objects grows.

2. Limited chunking algorithm (fixed)
Snapshot support

• An naive approach can not work out because snapshot creation generates messages as much as the chunks the object contains (e.g., if SNAP 10 contains three chunks, the three messages are generated)

< 1. Send increment message for three chunks for snapshot creation, 2. Acks, 3. Snapshot creation is done >
Snapshot and scrub time

**Snapshot support**

- An naive approach can not work out because snapshot creation generates messages as much as the chunks the object contains (e.g., if SNAP 10 contains three chunks, the three messages are generated):

  1. Send increment message for three chunks for snapshot creation,
  2. Acks,
  3. Snapshot creation is done

  Must wait for all increment messages to complete snapshot creation (1, 2, and 3)

```
0  2  4  6  8  10
New HEAD
chunk A  chunk D  chunk C
HEAD
(chunk will become SNAP 10)
chunk A  chunk B  chunk C
SNAP 8
chunk A  chunk C

<Snapshot object>  <Chunk pool>
```

< 1. Send increment message for three chunks for snapshot creation, 2. Acks, 3. Snapshot creation is done >
### Scrub overhead

- **Scrub** is something like reference garbage collection that check the reference is valid.

- Scrub is needed because reference leak (rarely) occur from false-positive design, as follows:

  - **Possible case**
    - Metadata object
      - Fingerprints {a, b, c..}
      - Chunk object A
        - Ref. count : 2
      -Metadata object
        - Fingerprints {a, d, e..}
      - Metadata object
        - Fingerprints {a, c, t..}

  - **Impossible case**
    - Metadata object
      - Fingerprints {a, d, e..}
      - Chunk object A
        - Ref. count : 1
      - Metadata object
        - Fingerprints {a, c, t..}
**Snapshot and scrub time**

**Scrub overhead**

- Scrub requires a full search on the base tier. So, it takes a significant time to complete as the number of objects grows.

![Diagram]

Need to check all metadata objects to see if ref. count is valid $O (#\text{chunk objects} \times #\text{metadata objects})$. 
- Selective cluster-level crawling

- Event driven architecture

- OID shared reference scheme
System Overview

- Selective dedup processing using crawler
  - Perform deduplication if data is dedup-able

- Event-driven architecture with content defined chunking
  - Only act in the event of predefined APIs with CDC (content defined chunking)

- OID-shared reference scheme
  - Reduce the number of message for snapshot and scrub based on false-positive ref. management
System Overview

- Selective dedup processing using crawler
  - Perform deduplication if data is dedup-able

- Event-driven architecture with content defined chunking
  - Only act in the event of predefined APIs with CDC (content defined chunking)

- OID-shared reference scheme
  - Reduce the number of message for snapshot and scrub based on false-positive ref. management
**System Overview**

- **Selective dedup processing using crawler**
  - Perform deduplication if data is dedup-able

- **Event-driven architecture with content defined chunking**
  - Only act in the event of predefined APIs with CDC (content defined chunking)

- **OID-shared reference scheme**
  - Reduce the number of message for snapshot and scrub based on false-positive ref. management

---

**Diagram:**

- **Base Tier**
  - OSD 1
    - Chunk Map
      - Start: 0
        - Size: 4096
        - Fp A
      - Start: 4096
        - Size: 4096
        - Fp B
    - Cached Data

- **Chunk Tier**
  - OSD 2
    - Reference List
      - Ref. MOID 1
      - Chunk Data
    - Chunk Object A
      - Reference List
        - Ref. MOID 1
        - Chunk Data
    - Chunk Object B
      - Reference List
        - Ref. MOID 3
        - Chunk Data
    - Chunk Object C
      - Data of MOID 2

- **Client and Crawler Connections**
  - Client 1 (Write)
  - Client 2 (Read)
  - Crawler (Incremental / Full)

---

**Re-architecting entire tiering mechanism on the base tier**

- **Flush Cold State Object (FC)**
System Overview

- Selective dedup processing using crawler
  - Perform deduplication if data is dedup-able

- Event-driven architecture with content defined chunking
  - Only act in the event of predefined APIs with CDC (content defined chunking)

- OID-shared reference scheme
  - Reduce the number of message for snapshot and scrub based on false-positive ref. management
Decoupled dedup controller scheme from server daemon

- Unlike prior work, TiDedup employ a separate process, called crawler.
- The crawler process searches objects, then makes a decision on deduplication if the chunk is dedup-able (> threshold value).

Perform deduplication if the chunk is found more than threshold times with two modes

- Incremental mode
  - At daytime, scan a small set of metadata object gradually to minimize resource usage.
- Full mode
  - At nighttime, scan all metadata object, then perform deduplication without consider resource utilization.
Object management

- Object is one of three states: hot or cold or deduped

Stateless

- Re-execution to overcome failures
- Repeat the loop in the reverse direction
- In-memory fingerprint store
  - Store fingerprints to check which fingerprint is dedup-able
  - Map < fingerprint : duplicate count >
The goal is to react an action in the event of external APIs with CDC

- I/O path and APIs are designed (e.g., set_chunk, tier_flush, tier_evict, tier_promote)
- Relevant metadata (e.g., chunk_info_t) is embedded into existing object metadata, called “object_info_t”
- chunk_state in chunk_info_t can be one of \{MISSING, CLEAN\}
  - MISSING: no cached content, CLEAN: cached content

Read

- Find chunk_info_t which is associated with the requested offset
- Forward read request to a chunk object (OID is the destination OID)
  In chunk_info_t if the chunk state is MISSING

Write

- Write contents, then clear the corresponding chunk_info_t---content define chunking calculates the different chunk boundary depending on the contents, so modification requires a recalculation

```c
object_info_t { // object’s metadata
  manifest_info_t {
    chunk_map <offset, chunk_info_t>;
    version;
    object state;
  }
}

chunk_info_t { // fingerprint value from chunk’s object
  chunk state;
  destination offset;
  length;
  destination OID;
}
### Set_chunk()

- Set a part of the object to deduped
- Synchronous call (wait for increment message’s ack, then reply the result to caller and update metadata)
- Transactional processing
**Scrub worker**

- Each chunk object has **OIDs (like a back pointer)** instead of ref. count
  - Ceph's OID format includes location information such as tier and object name
- To check if a reference is valid, TiDedup needs only two reads (chunk object’s extended attributed and metadata object), unlike the prior work
Scrub worker

- Each chunk object has **OIDs (like a back pointer)** instead of ref. count
  - Ceph’s OID format includes location information such as tier and object name
- To check if a reference is valid, TiDedup needs only two reads (chunk object’s extended attributed and metadata object), unlike the prior work

<table>
<thead>
<tr>
<th>Metadata object A</th>
<th>Metadata object B</th>
<th>Metadata object C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fingerprints {a, b, c..}</td>
<td>Fingerprints {a, d, e..}</td>
<td>Fingerprints {a, c, t..}</td>
</tr>
</tbody>
</table>

No need to check all metadata objects to see if chunk object’s ref. count is valid

<table>
<thead>
<tr>
<th>Chunk object A</th>
</tr>
</thead>
<tbody>
<tr>
<td>OIDs {A, B, C}</td>
</tr>
</tbody>
</table>
**Objective**
- Based on false-positive design, minimize the number of generated message

**Snapshot creation**
- **Check adjacent snapshot’s chunk first**, then **do not add chunk’s reference if adjacent snapshot’s chunk is identical** when creating a new snapshot.
- The number of chunk’s reference:
  - AAA: 1, BBB: 1, DDD: 1, CCC: 1

<table>
<thead>
<tr>
<th>Offset</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>New HEAD</td>
<td>AAA</td>
<td>BBB</td>
<td>DDD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNAP 7</td>
<td>AAA</td>
<td>BBB</td>
<td>DDD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEAD</td>
<td>AAA</td>
<td>BBB</td>
<td>DDD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNAP 5</td>
<td>AAA</td>
<td>BBB</td>
<td>CCC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Snapshot deletion**
- If either adjacent snapshots has the same chunk, no messages needs to be generated.
- TiDedup would send a delete reference message only if adjacent chunks and itself are unreferenced.
**Objective**

- Based on false-positive design, minimize the number of generated message

**Snapshot creation**

- Check adjacent snapshot’s chunk first, then do not add chunk’s reference if adjacent snapshot’s chunk is identical when creating a new snapshot.
- The number of chunk’s reference:
  - AAA: 1, BBB: 1, DDD: 1, CCC: 1

**Snapshot deletion**

- If either adjacent snapshots has the same chunk, no messages needs to be generated.
- TiDedup would send a delete reference message only if adjacent chunks and itself are unreferenced.

More details are in the paper.
Test Environment

- 10 machines in total (a 2-way AMD EPYC 7543 32-cores)
- 512 GB DRAM for each machine
- QLC SSDs (4 TB) for each machine
- The latest version (Reef) of the Ceph
- Two replicas
- FastCDC [Wen Xia et al., USENIX ATC 2016]
- SHA1 is used to generate fingerprint value among the available fingerprint algorithm options (e.g., SHA1, SHA128, and SHA256).
The effect of deduplication in real world workload

<table>
<thead>
<tr>
<th>Workload</th>
<th>Type</th>
<th>Total Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>equipment</td>
<td>44 TiB</td>
</tr>
<tr>
<td>2</td>
<td>chip</td>
<td>70 TiB</td>
</tr>
<tr>
<td>3</td>
<td>photo</td>
<td>37 TiB</td>
</tr>
<tr>
<td>4</td>
<td>metrology</td>
<td>20 TiB</td>
</tr>
</tbody>
</table>

- **Workload 1**: equipment status
- **Workload 2**: chip information during manufacturing
- **Workload 3**: logs for photo lithography
- **Workload 4**: metrology and inspection image files

< Space saving on factory data >
The effect of deduplication in real world workload

<table>
<thead>
<tr>
<th>Type</th>
<th>Workload 1</th>
<th>Workload 2</th>
<th>Workload 3</th>
<th>Workload 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Size</td>
<td>44 TiB</td>
<td>70 TiB</td>
<td>37 TiB</td>
<td>20 TiB</td>
</tr>
</tbody>
</table>

- **Workload 1**: equipment status
- **Workload 2**: chip information during manufacturing
- **Workload 3**: logs for photo lithography
- **Workload 4**: metrology and inspection image files

< Space saving on factory data >

The effect of deduplication differs in the workloads
Evaluation

- Throughput test

(A) Throughput

(B) Read average latency

(C) Write average latency

< YCSB throughput (Workload a) >
Throughput test

(A) Throughput
(B) Read average latency
(C) Write average latency

Performance fluctuation disappear in TiDedup
Evaluation

Throughput test

More experiments are in the paper

< YCSB throughput (Workload a) >
Conclusion

- Prior work has a few limitations to be used in Ceph
  - Storage saving, structure limitations, scrub and snapshot support

- TiDedup proposed three key design to resolve observed problems
  - Selective crawler, event-driven architecture, OID-shared reference

- The source code is available at Ceph github
  - [https://github.com/ssdohammer-sl/ceph/tree/tidedup](https://github.com/ssdohammer-sl/ceph/tree/tidedup)

- Welcome feedback
  - GitHub, private email, mailing list, whatever
Thank You
The effect of deduplication in real world workload

<table>
<thead>
<tr>
<th>Chunk size</th>
<th>Virtual disks</th>
<th>Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8K</td>
<td>16K</td>
</tr>
<tr>
<td>Fixed</td>
<td>21%</td>
<td>12%</td>
</tr>
<tr>
<td>TiDedup</td>
<td>45%</td>
<td>36%</td>
</tr>
<tr>
<td>CDC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Space saving on real-world datasets depends on the chunking algorithm and average chunk size. Virtual disks represents VMware vSphere images (10.1 TB) from a developer cloud service (67 users). Logs represents service logs (560 GB) for cloud Infrastructure including monitoring and device state.
The effect of deduplication in real world workload

<table>
<thead>
<tr>
<th>Chunk size</th>
<th>Virtual disks</th>
<th>Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8K</td>
<td>16K</td>
</tr>
<tr>
<td>Fixed</td>
<td>21%</td>
<td>12%</td>
</tr>
<tr>
<td>TiDedup (CDC)</td>
<td>45%</td>
<td>36%</td>
</tr>
</tbody>
</table>

< Space saving on real-world datasets depends on the chunking algorithm and average chunk size. Virtual disks represents VMware vSphere images (10.1 TB) from a developer cloud service (67 users). Logs represents service logs (560 GB) for cloud infrastructure including monitoring and device state.>

CDC shows considerable improvement in terms of space saving in some workload. The both algorithms can be a compliment approach.