Concurrent Deletion in a Distributed Content-Addressable Storage System with Global Deduplication

Przemysław Strzelczak, Elżbieta Adamczyk, Urszula Herman-Iżycka, Jakub Sakowicz, Łukasz Ślusarczyk, Jarosław Wrona, Cezary Dubnicki

9LivesData, LLC
Who we are

- **9LivesData**
  - R&D company based in Warsaw, Poland
  - 50+ scientists and software engineers
  - designers/coders of HYDRAstor backend for NEC

- **HYDRAstor**
  - scalable, content-addressable backup storage
  - global dedup, self-healing
  - owned by NEC, on sale in the USA and Japan
  - started by 9LivesData founder in Princeton, NJ
  - **fastest and largest** dedup backup system*

* Curtis W.Preston performance comparison – 10/28/2010
Core features of secondary storage systems

- Large capacity (deduplication)
- High performance
- High reliability
- Data deletion
Agenda

- Requirements and challenges
- Deletion algorithm for CAS systems
  - read-only period
  - centralized system
- Extension to support writes with deduplication
- Extension to distributed systems
- Implementation in HYDRAstor
Key requirements

- Continuous system availability
  - no read-only period
  - failure tolerance

- Deduplication enabled during deletion

- Low impact on user operations
  - backup
  - restore
  - replication

- Deletion impact independent from system size
Summary of data model

Files chunked into variable sized blocks
Summary of data model

Regular blocks
- immutable
- content-addressable

New blocks written **sequentially**

Files chunked into variable sized **blocks**
Summary of data model

Regular blocks can contain **pointers** to other blocks

Files chunked into variable sized **blocks**

New blocks written **sequentially**

Regular blocks

Immutable

Content-addressable

/mnt/fs1

..

file1

AB

cD

file2

A

B

C

D
Summary of data model

Filesystem backups as **trees** of blocks built **bottom-up**

**Named blocks**:  
- read using user provided **key**  
- **retention roots** – tree preservation  
- **deletion roots** – tree deletion

Regular blocks can contain **pointers** to other blocks

New blocks written **sequentially**  
Files chunked into variable sized **blocks**
Data model in CAS storage system

regular data block
Data model in CAS storage system
Data model in CAS storage system

regular block with pointers
Data model in CAS storage system

/mnt/fs1
  .
  ..
file1  A B
file2  C D

A
B
C
Data model in CAS storage system

/mnt/fs1
..
file1  A B
file2  C D

[Diagram of file structure with boxes labeled A, B, C, D]
Data model in CAS storage system

/mnt/fs1
 ..
..
file1  A B
file2  C D
Data model in CAS storage system

/fs1_#1

retention root

/mnt/fs1
.
..
file1  AB
file2  CD

A B C D
Data model in CAS storage system

```
/mnt/fs1
  ..
file1
file2
```

Duplicate block

```
A
B
C
D
A
```
Data model in CAS storage system

/mnt/fs1
  ...
file1  A E
file2  C D

fs1_#1
A
B
C
D
A
E
Data model in CAS storage system
Data model in CAS storage system

The diagram illustrates the data model in a CAS storage system. The root directory is `/mnt/fs1`, and it contains two directories: `..` and `file1` with files `A` and `E`, and `file2` with files `C` and `D`. The file system structure is depicted with a tree-like diagram, showing the hierarchical relationship between directories and files.
Data model in CAS storage system
Data model in CAS storage system

/mnt/fs1
  .
  ..
file1  A E
file2  C D

fs1_#1

A  B  C  D  E
C  D

HYDRAstor
Data model in CAS storage system

Duplicate block with pointers

/mnt/fs1
  ..
file1
file2
Data model in CAS storage system
Data model in CAS storage system
Data model in CAS storage system

Delete previous version of a filesystem
Data model in CAS storage system

Orphaned block F after e.g. client failure during 3rd backup
Deletion semantics

Deletion root

Alive retention root

Dead retention root

Blocks reachable from not deleted roots are alive. Remaining blocks are dead.
Deletion semantics

Deletion algorithm deletes dead blocks and preserves alive blocks.
Challenges for deletion in CAS systems
Challenges for deletion in CAS systems

3rd backup of fs1 is started

/mnt/fs1
.
..
file1

fs1_#1

fs1_#2

A
B
C
D
E
F

time
Challenges for deletion in CAS systems

```
file1
```

```
/mnt/fs1
.
..  
file1  FB
```
Challenges for deletion in CAS systems

```
/mnt/fs1
.
.. file1
```

```
fs1_#1
```

```
fs1_#2
```

```
time
```

```
A B C D E F
```
Challenges for deletion in CAS systems
Challenges for deletion in CAS systems
Challenges for deletion in CAS systems

Dead block made alive by deduplication during deletion

Dead block made alive by new writes during deletion
Base deletion algorithm in centralized CAS system
Base deletion algorithm

- Design decisions
  - deferred reference counting
  - incremental
- Limitations
  - no writes during deletion
- Two phases:
  - garbage identification
  - space reclamation
- Deletion runs started on demand
- Two versions of counters
  - effective
  - temporary (become effective when deletion run ends)
Base algorithm in steps

New blocks always get special value called **initial zero** (i0)
Base algorithm in steps

Deletion run starts

Time line:
- fs1_#1
- i0
- i0
- i0
- i0
- A
- B
- C
- D
Base algorithm in steps

Incrementation subphase

- fs1_#1
- A
- B
- C
- D

Time
Base algorithm in steps

Deletion run ends
Base algorithm in steps
Base algorithm in steps
Base algorithm in steps
Base algorithm in steps

![Diagram of algorithm steps]

- **fs1_#1**
  - A
  - B
  - C
  - D

- **fs1_#2**
  - E
  - F

Time progression:
- i0
- i0
- i0

**Diagram notes:**
- Green boxes: Progression stages
- Blue box: Final state
Base algorithm in steps

Incrementation subphase
Base algorithm in steps

The diagram illustrates the base algorithm in steps, showing the progression of data elements (A, B, C, D, E, F) through different stages (fs1_#1, fs1_#2) at specific time points (1, 2).

- **fs1_#1**:
  - Time 1: Data elements A, B, C, D
  - Time 2: Data elements A, B, C, D

- **fs1_#2**:
  - Time 1: Data elements E
  - Time 2: Data elements E

The diagram also includes a trash can symbol indicating data removal or deletion.
Base algorithm in steps
Base algorithm in steps

Recursive decrementation
Problems with base algorithm
Supporting new writes with deduplication enabled
Supporting new writes during deletion

Block written after deletion run starts **NOT** removed during this deletion run
Supporting new writes during deletion

**Goal:** preserve blocks that can be pointed by block written after deletion starts

**Problem:** client can potentially write a pointer to any block written earlier
Supporting new writes during deletion

Every deletion run advances epoch twice
Supporting new writes during deletion

Block addresses stamped with epochs

Epoch (T)  Epoch (T + 1)  Epoch (T + 2)
Supporting new writes during deletion

Block addresses stamped with epochs

In epoch (T + 2) client can only keep addresses with epoch (T + 1) and (T + 2)
Supporting new writes during deletion

Block addresses stamped with epochs

System can advance to epoch (T + 2) if client confirms disposal of addresses from epoch (T)

In epoch (T + 2) client can only keep addresses with epoch (T + 1) and (T + 2)
Supporting new writes during deletion

- Non-duplicate block written in epoch $T$ has epoch $T$ assigned
- System can advance to epoch $(T + 2)$ if client confirms disposal of addresses from epoch $(T)$
- In epoch $(T + 2)$ client can only keep addresses with epoch $(T + 1)$ and $(T + 2)$
Supporting new writes during deletion

Minimum rule: Address of block with pointers is assigned the lowest epoch of addresses within T time.

In epoch (T + 2), client can only keep addresses with epoch (T + 1) and (T + 2).

System can advance to epoch (T + 2) if client confirms disposal of addresses from epoch (T).
Supporting new writes during deletion

In epoch (T + 2) client can only keep addresses with epoch (T + 1) and (T + 2)

System can advance to epoch (T + 2) if client confirms disposal of addresses from epoch (T)
Supporting new writes during deletion

- Epoch (T)
- Epoch (T + 1)
- Epoch (T + 2)

- advance
- tmpRR

- DEC
- INC

- time
Supporting new writes during deletion

- System can advance to epoch (T + 2) if client confirms disposal of addresses from epoch (T).
- In epoch (T + 2), client can only keep addresses with epoch (T + 1) and (T + 2).
- System rejects writes of blocks with addresses that have illegal epoch.
Supporting deduplication during deletion
Supporting deduplication during deletion
Supporting deduplication during deletion
Supporting deduplication during deletion

Undelete marker is set during deduplication

Undelete markers kept as bitmaps in RAM not to impact dedup much
Supporting deduplication during deletion
Supporting deduplication during deletion
Supporting deduplication during deletion

\[ \text{fs1}_{\#9} \rightarrow \text{K} \rightarrow \text{L} \rightarrow \text{fs2}_{\#1} \]

\[ \text{fs1}_{\#9} \rightarrow \text{U} \rightarrow \text{fs2}_{\#1} \]

\[ \text{time} \]

\[ 0 \rightarrow 0 \rightarrow 0 \]
Supporting deduplication during deletion

- fs1_#9
- fs2_#1

Setting undelete markers

Undeletion

Time: 0 0

K L U
Supporting deduplication during deletion
Supporting deduplication during deletion

Diagram showing the process of setting undelete markers and undeletion over time.
Deletion in a distributed architecture
New challenges introduced by distribution

- Assumptions:
  - block data distributed

- Dynamic configuration
  - nodes can be added/deleted

- Consistency of blocks preservation/removal

- Handling failures
  - detached nodes with obsolete counters
  - continue on failure of some nodes
Key techniques to support distribution

- Continue on failure of some nodes
  - redundancy of counter computation
  - redundancy of undelete markers

- Recognize obsolete counters
  - stamping counters with epochs
  - counter epochs refreshed when deletion ends
Implementation in HYDRAstor
Performance evaluation in HYDRAstor

- Configurable resource division
  - default: 30% deletion, 70% user operations
  - minimal: less than 1% for deletion
- Experiments used 4 to 16 machines
  - 2\textsuperscript{nd} or 3\textsuperscript{rd} generation HYDRAstor servers
    - 2 quad core 2.4Ghz CPU
    - 12x1TB SATA disks
    - 24GB RAM
    - 4 GigE cards
- Measurements
  - bandwidth of reads and writes during deletion
  - deletion duration
Impact of garbage identification on writes

- No deletion
- 30% deletion
- 1% deletion

<table>
<thead>
<tr>
<th>Dedup Ratio</th>
<th>Write Bandwidth (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>-26%</td>
</tr>
<tr>
<td>25%</td>
<td>-28%</td>
</tr>
<tr>
<td>50%</td>
<td>-28%</td>
</tr>
<tr>
<td>75%</td>
<td>-17%</td>
</tr>
<tr>
<td>99%</td>
<td>-20%</td>
</tr>
</tbody>
</table>

Legend:
- Cyan: -26%
- Green: -4%
- Turquoise: -28%
- Yellow: -5%
- White: -20%
- Gray: 0%
Scalability

![Graph showing scalability with respect to the number of nodes. The x-axis represents the number of nodes, and the y-axis represents the deletion duration in seconds. Two lines are shown: initial and incremental. The initial line shows a slight increase in deletion duration as the number of nodes increases. The incremental line shows a more significant decrease in deletion duration as the number of nodes increases.]
Conclusions
Conclusions

• Problem
  • Deletion in distributed CAS systems

• Solution
  • concurrent deletion algorithm
  • writes with dedup enabled during deletion
  • failure tolerance
  • scalability

• Key techniques
  • epochs
  • undelete markers

• Implementation in HYDRAstor
  • fairly low performance impact
Questions?

strzelczak@9livesdata.com
Appendix
Deduplication support during undeletion
Supporting deduplication during deletion

Deduplication to blocks to be deleted is disabled during base block filtering.
Supporting deduplication during deletion

Deduplication to blocks to be deleted is disabled during base block filtering.
Supporting deduplication during deletion

Deduplication to blocks to be preserved is still works during base block filtering.
Deduplication of blocks with pointers during deletion
Deduplication of blocks with pointers during deletion

Minimum rule prevents references from Epoch (T + 2)

Epoch (T)

Epoch (T + 1)

Epoch (T + 2)
Deduplication of blocks with pointers during deletion

Epoch (T) \[\rightarrow\] Epoch (T + 1) \[\rightarrow\] Epoch (T + 2)

Undelete markers can lead to unreachable blocks with dangling pointers deleted during next deletion run.

Minimum rule prevents references from Epoch (T + 2)

Undelete markers can lead to unreachable blocks with dangling pointers deleted during next deletion run.

Epoch (T) \[\rightarrow\] Epoch (T + 1) \[\rightarrow\] Epoch (T + 2)
Distributed architecture
DHT, supernodes, peers

Empty Prefix

0
00
Node1
01
Peer
1
10
Node2
11
Node3
Supernode
Node4
Node5
Node6
Container based organization

Supernode 01

Erasure coding

Peer1

Peer2

Peer3

Peer4

A1

A2

A3

A4
Container based data organization
Container based organization

Supernode 01

Peer1

Peer2

Peer3

Peer4

Erasure coding
Redundancy of computation

Empty Prefix

0

00

Node1

01

Node2

10

Node3

11

Node4

Node5

Node6

Redundancy

3

3

3

3

Good peer
Redundancy of computation

![Diagram of redundancy with empty prefix and nodes](image-url)
Redundancy of computation

Empty Prefix

0

00

Node1

01

Node2

1

10

Node3

11

Node4

Node5

Node6

Redundancy

3

1

2

2

Not enough good peers. Deletion run is aborted.