Sliding Look-Back Window Assisted Data Chunk Rewriting for Improving Deduplication Restore Performance

Zhichao Cao\textsuperscript{1}, Shiyong Liu\textsuperscript{2}, Fenggang Wu\textsuperscript{1}, Guohua Wang\textsuperscript{3}, Bingzhe Li\textsuperscript{1}, and David H.C. Du\textsuperscript{1}

\textsuperscript{1}University of Minnesota, Twin Cities
\textsuperscript{2}Ocean University of China \textsuperscript{3}South China University of Technology

02/26/2019
Agenda

• Deduplication and Restore
• Data Chunk Rewrite Preliminary
• Container Capping Introduction and Limitations
• Proposed Solutions
  – Flexible Container Referenced Count based Design (FCRC)
  – Sliding Look-Back Window (LBW)
• Evaluations
• Conclusions and Future Work
Agenda

• Deduplication and Restore
• Data Chunk Rewrite Preliminary
• Container Capping Introduction and Limitations
• Proposed Solutions
  – Flexible Container Referenced Count based Design (FCRC)
  – Sliding Look-Back Window (LBW)
• Evaluations
• Conclusions and Future Work
Deduplication Process [1]

Byte Stream

Recipe

22 23 8 22 18 2 5 10

Sliding Window

Active container

22 23 5

Container Storage

Indexing Table
Deduplication Process [1]

Byte Stream

Recipe

Active container

Indexing Table

Container Storage

Center for Research in Intelligent Storage
Deduplication Process \[1\]

 Byte Stream

 22 23 8 22 18 2 5 10

 Sliding Window

 Recipe
 22 23 8 22 18 2 5 10

 Active container
 22 23 5

 Container Storage

 18 19 20 21
 2 3 10 13

 8 9 17 25

 Indexing Table
Deduplication Process [1]
Deduplication Process [1]

Recipe

| 22 | 23 | 8 | 22 | 18 | 2 | 5 | 10 |

Container Storage

| 18 | 19 | 20 | 21 | 2 | 3 | 10 | 13 |
| 8 | 9 | 17 | 25 | ...... |

Active container

| 22 | 23 | 5 |

Sliding Window

Indexing Table

Byte Stream
Deduplication Process \[1\]
Deduplication Process [1]

Byte Stream

Recipe

| 22 | 23 | 8 | 22 | 18 | 2 | 5 | 10 |

Active container

| 22 | 23 | 5 | 14 |

Container Storage

| 18 | 19 | 20 | 21 | 2 | 3 | 10 | 13 |

| 8 | 9 | 17 | 25 |

Indexing Table
Not Found
Deduplication Process [1]

Byte Stream

Container Storage

Active container

Indexing Table
   Not Found

Recipe

Sliding Window
Deduplication Process [1]

Recipe

Active container

Container Storage

Indexing Table

Deduplication Process [1]

Recipe

22 23 8 22 18 2 5 10 14

Container Storage

18 19 20 21 2 3 10 13 22 23 5 14

Indexing Table

Deduplication Process \[1\]

Recipe

Container Storage

Indexing Table

Deduplication Process [1]

Recipe

22 23 8 22 18 2 5 10 14

Active container

18 19 20 21 2 3 10 13 22 23 5 14

……

Container Storage

Indexing Table

Deduplication Process \[1\]

Recipe

\begin{center}
\begin{tabular}{cccccccc}
22 & 23 & 8 & 22 & 18 & 2 & 5 & 10 & 14 & 5
\end{tabular}
\end{center}

Active container

\begin{center}
\begin{tabular}{cccc}
22 & 23 & 5 & 14
\end{tabular}
\end{center}

Container Storage

\begin{center}
\begin{tabular}{ccccccc}
18 & 19 & 20 & 21 & 2 & 3 & 10 & 13 & 22 & 23 & 5 & 14
\end{tabular}
\end{center}

Indexing Table

Deduplication Process [1]

Deduplication Process [1]

Deduplication Process [1]

Recipe

Active container

Container Storage

Indexing Table

Found

Restore Process with Chunk-based Caching\cite{2}

Recipe

```
22 23 8 22 18 2 5 10 14 18 13 22 3 28 23 12 13 32 23 28 6 .......
```

![Diagram showing the restore process with chunk-based caching](Image)

- **Restored Data Storage**: Contains restored data chunks.
- **Assembling Buffer**: Used to assemble chunks from the restored data storage.
- **Container Read Buffer**: Holds chunks that are read from the container storage.
- **Chunk Cache**: Stores chunks for efficient access.
- **Container Storage**: Holds the chunks for the container read buffer.

The process involves reading chunks from the container storage and assembling them in the assembling buffer, then restoring the data in the restored data storage.
Restore Process with Chunk-based Caching[2]

Recipe

22 23 8 22 18 2 5 10 14 18 13 22 3 28 23 12 13 32 23 28 6

Restored Data Storage

22 23 8 22

2 3 5 13 12 14 5 13

Assembling Buffer

18 2 5

Container Read Buffer

2 3 10 13

Chunk Cache

2 3 18 22

14 13 5 10

Container Storage

18 19 20 21

2 3 10 13

8 9 17 25

22 23 5 14
Restore Process with Chunk-based Caching[2]

Recipe

22 23 8 22 18 2 5 10 14 18 13 22 3 28 23 12 13 32 23 28 6

Restored Data Storage

22 23 8 22
2 3 5 13 12 14 5 13

Assembling Buffer

18 2 5

Container Read Buffer

2 3 10 13

Chunk Cache

2 3 18 22
14 13 5 10

Container Storage

18 19 20 21
2 3 10 13
8 9 17 25
22 23 5 14

………
Restore Process with Chunk-based Caching[2]

Recipe

22 23 8 22 18 2 5 10 14 18 13 22 3 28 23 12 13 32 23 28 6 ……

Restored Data Storage

22 23 8 22

2 3 5 13 12 14 5 13

……

Assembling Buffer

18 2 5 10

Container Read Buffer

2 3 10 13

Chunk Cache

2 3 18 22

14 13 5 10

Container Storage

18 19 20 21

2 3 10 13

8 9 17 25

22 23 5 14

……
Restore Process with Chunk-based Caching \cite{2}

Recipe

Restored Data Storage

Assembling Buffer

Container Read Buffer

Chunk Cache

Container Storage
Restore Process with Chunk-based Caching

Recipe

Restored Data Storage

Assembling Buffer

Container Read Buffer

Chunk Cache

Container Storage
Restore Process with Chunk-based Caching[2]

Recipe

22 23 8 22 18 2 5 10 14 18 13 22 3 28 23 12 13 32 23 28 6 ……

Restored Data Storage

22 23 8 22 18 2 5 10
2 3 5 13 12 14 5 13
……

Assembling Buffer

Container Read Buffer

2 3 10 13
14 13 5 10

Chunk Cache

2 3 18 22
……

Container Storage

18 19 20 21
2 3 10 13
8 9 17 25
22 23 5 14
……
Restore Process with Chunk-based Caching[2]

Recipe

22 23 8 22 18 2 5 10 14 18 13 22 3 28 23 12 13 32 23 28 6

Restored Data Storage

22 23 8 22 18 2 5 10

2 3 5 13 12 14 5 13

Assembling Buffer

Container Read Buffer

2 3 10 13

14 13 5 10

Container Storage

18 19 20 21

2 3 10 13

8 9 17 25

22 23 5 14

Chunk Cache

2 3 18 22

…….
Restore Process with Chunk-based Caching

Recipe

22 23 8 22 18 2 5 10 14 18 13 22 3 28 23 12 13 32 23 28 6 .......

- Restored Data Storage
  - 22 23 8 22 18 2 5 10
  - 2 3 5 13 12 14 5 13

- Assembling Buffer
  - 14

- Container Read Buffer
  - 2 3 10 13

- Chunk Cache
  - 2 3 18 22
  - 14 13 5 10

- Container Storage
  - 18 19 20 21
  - 8 9 17 25
  - 22 23 5 14
 Restore Process with Chunk-based Caching [2]
Restore Process with Chunk-based Caching[2]

Recipe

Restored Data Storage

Container Read Buffer

Assembling Buffer

Container Storage

Chunk Cache

Center for Research in Intelligent Storage
Restore Process with Chunk-based Caching

Recipe

22 23 8 22 18 2 5 10 14 18 13 22 3 28 23 12 13 32 23 28 6

Assembling Buffer

14 18

Container Read Buffer

2 3 10 13

Chunk Cache

2 3 18 22
14 13 5 10

Restored Data Storage

22 23 8 22 18 2 5 10
2 3 5 13 12 14 5 13

……

Container Storage

18 19 20 21
2 3 10 13

……

8 9 17 25
22 23 5 14

……
Agenda

• Deduplication and Restore
• Data Chunk Rewrite Preliminary
• Container Capping Introduction and Limitations
• Proposed Solutions
  – Flexible Container Referenced Count based Design (FCRC)
  – Sliding Look-Back Window (LBW)
• Evaluations
• Conclusions and Future Work
Limitations of Caching Scheme

restore direction

Recipe

restored data storage

22 23 8 22

2 3 5 13 12 14 5 13

assembling buffer

container read buffer

container storage

18 35 33 66

41 51 10 52

56 79 17 81

61 63 42 14

49x406

Limitations of Caching Scheme
Limitations of Caching Scheme

Restore Direction

Recipe

Restored Data Storage

Assembling Buffer

Container Read Buffer

Container Storage

Limitations of Caching Scheme
Limitations of Caching Scheme

- 4 container reads to restore 4 chunks

Recipe

Restore Direction

- 5 13 22 23 8 22 2 2 5 10 14 18 17 22 3 28 23 12 13 32 23 28 6
Limitations of Caching Scheme

- 4 container reads to restore 4 chunks
- Other chunks cannot benefit restore

Recipe

| 5 | 13 | 22 | 23 | 8 | 22 | 2 | 2 | 5 | 10 | 14 | 18 | 17 | 22 | 3 | 28 | 23 | 12 | 13 | 32 | 23 | 28 | 6 |

Restore Direction

- 4 container reads to restore 4 chunks
- Other chunks cannot benefit restore

Restored Data Storage

22 23 8 22
2 3 5 13 12 14 5 13

Assembling Buffer

Container Read Buffer

Container Storage

18 35 33 66 41 51 10 52
56 79 17 81 61 63 42 14

Center for Research in Intelligent Storage
Limitations of Caching Scheme

4 container reads to restore 4 chunks, other chunks cannot benefit from restore

Caching loses its power

- 4 container reads to restore 4 chunks
- Other chunks cannot benefit from restore

Limitations of Caching Scheme

- Caching loses its power
Data Chunk Rewrite Schemes

Deduplication Direction

Container Storage

Active container
Data Chunk Rewrite Schemes

Deduplication Direction

Container Storage

Active container
Data Chunk Rewrite Schemes

Deduplication Direction

Container Storage

Active container
Data Chunk Rewrite Schemes

Deduplication Direction

Container Storage

Active container
Data Chunk Rewrite Schemes

Deduplication Direction

Container Storage

Active container
Data Chunk Rewrite Schemes

Deduplication Direction

• 1 container reads can restore 4 chunks

Center for Research in Intelligent Storage
Data Chunk Rewrite Schemes

Deduplication Direction

- 1 container reads can restore 4 chunks
- Tradeoff between space saving and restore performance

Center for Research in Intelligent Storage
Related Work

• Nam et al. introduced the **Chunk Fragmentation Level** based data chunk rewrite [4-5]

• The **mismatch level** between byte stream context and data chunk disk context is used to decide the data chunks to be rewrite, which is presented by Kaczmarczyk et al. [6]

• Fu et al. proposed a **History-Aware Rewriting algorithm (HAR)** which identifies and rewrites sparse containers [7-8]

• Tan et al. proposed a **Fine-Grained defragmentation** approach (FGDefrag) to identify and rewrite the fragmental chunks [9]

• **Container capping** was proposed by Lillibridge et al. [3]

• ……
Challenges of Rewrite

- Rewrite sacrifices deduplication ratio (space saving), how to make **better tradeoffs** between **deduplication ratio** and **restore performance** is challenging;

- Rewrite is done during the deduplication process, information is limited. Most related studies are based on the **past statistic information** to make the decision. How to decide the data chunks to be rewritten with limited information is challenging;

- The **restore caching effectiveness** should be considered during the dedup to reduce unnecessary rewrites. However, how to integrate caching with rewrite is not clearly investigated
Agenda

• Deduplication and Restore
• Data Chunk Rewrite Preliminary
• Container Capping Introduction and Limitations
• Proposed Solutions
  – Flexible Container Referenced Count based Design (FCRC)
  – Sliding Look-Back Window (LBW)
• Evaluations
• Conclusions and Future Work
Introduction of Container Capping\textsuperscript{[3]}

Deduplication Direction

\begin{itemize}
\item Active container
\end{itemize}
Introduction of Container Capping\textsuperscript{[3]}

Deduplication Direction

Active container
Introduction of Container Capping\textsuperscript{[3]}

Deduplication Direction

Segment

Active container
Introduction of Container Capping\textsuperscript{[3]}

Deduplication Direction

Segment

Active container
Introduction of Container Capping\textsuperscript{[3]}

Deduplication Direction

Unique chunk

Segment

Active container
Introduction of Container Capping

Unique chunk

Deduplication Direction

Segment

Active container
Introduction of Container Capping [3]

Deduplication Direction

Unique chunk

Segment

Active container
Introduction of Container Capping\cite{3}

Deduplication Direction

Segment

Unique chunk

Active container

- 8 chunks
- 4 chunks
- 3 chunks
- 1 chunk
- 1 chunk
Introduction of Container Capping\textsuperscript{[3]}

Deduplication Direction

Segment

Unique chunk

Active container

Capping level = 3

8 chunks
4 chunks
3 chunks
1 chunk
1 chunk
Introduction of Container Capping \textsuperscript{[3]}

Deduplication Direction

Segment

Unique chunk

Active container

8 chunks
4 chunks
3 chunks
1 chunk
1 chunk

Keep the original referencing

Capping level = 3
Introduction of Container Capping\textsuperscript{[3]}

Deduplication Direction

Unique chunk

Segment

Active container

- 8 chunks
- 4 chunks
- 3 chunks
- 1 chunk
- 1 chunk

Capping level = 3

Keep the original referencing

Rewrite and reference to the new container
Introduction of Container Capping

Segment

Deduplication Direction

Unique chunk

Active container

Capping level = 3

Keep the original referencing

Rewrite and reference to the new container

8 chunks

4 chunks

3 chunks

1 chunk

1 chunk
Introduction of Container Capping \[3\]

Deduplication Direction

Unique chunk

Segment

Active container

- 8 chunks
- 4 chunks
- 3 chunks
- 1 chunk
- 1 chunk

Capping level = 3

Keep the original referencing

Rewrite and reference to the new container
Introduction of Container Capping [3]

Deduplication Direction

Unique chunk

Segment

Active container

8 chunks
4 chunks
3 chunks
1 chunk
1 chunk

Capping level = 3

Keep the original referencing

Pros:
1) simple and efficient
2) guarantee the higher bound of container reads

Rewrite and reference to the new container
Introduction of Container Capping[3]

Deduplication Direction

Unique chunk

Segment

Active container

8 chunks

4 chunks

3 chunks

Capping level = 3

1 chunk

1 chunk

Keep the original referencing

Pros:
1) simple and efficient
2) guarantee the higher bound of container reads

Cons:
1) a fixed capping level cannot adapt to the workload
2) might make wrong rewrite decision due to the segment cut
3) deduplication ratio is not guaranteed
Capping Limitation 1: Fixed Capping Level

- The number of data chunks referenced by one container in a segment is called *container referenced count (CNRC)*.
- If we sort the CNRC of containers in a segment, we can get the distributions above, the distributions of different segment can be very different.
- *Capping level* is a fixed threshold, containers with CNRC ranked lower than the capping level are rewritten.
Capping Limitation 1: Fixed Capping Level
Capping Limitation 1: Fixed Capping Level

- Use the same capping level, we will have 20 old container reads, and need rewrite 232 chunks
Use the same capping level, we will have 20 old container reads, and need rewrite 232 chunks.
Capping Limitation 1: Fixed Capping Level

- Use the same capping level, we will have 20 old container reads, and need rewrite 232 chunks
Capping Limitation 1: Fixed Capping Level

- Use the same capping level, we will have 20 old container reads, and need rewrite 232 chunks.
- Use different actual capping level for different segments, the total old container reads are still 20, but we **rewrite fewer data chunks** (187 chunks).
Capping Limitation 2: Fixed Segment Cutting Issue

Data chunks close to the cutting boundary have higher probability to be rewritten.
Capping Limitation 3: Restore Caching

When caching is applied, data chunks from one container must be cached at least a small range of restore. The data chunks that are **covered by the cache effective range should not be rewritten**.
Agenda

• Deduplication and Restore
• Data Chunk Rewrite Preliminary
• Container Capping Introduction and Limitations
• Proposed Solutions
  – Flexible Container Referenced Count based Design (FCRC)
  – Sliding Look-Back Window (LBW)
• Evaluations
• Conclusions and Future Work
Flexible Container Referenced Count based Design (FCRC)

- To be adaptive to the ContaiNer Referenced Count (CNRC) distributions, we can use a CNRC value as the threshold

- **Higher threshold** → rewrite more chunks (lower deduplication ratio), but fewer container reads (e.g., threshold = 4)

- **Lower threshold** → rewrite fewer chunks (higher deduplication ratio), but more container reads (e.g., threshold = 1)

According to the target container reads and target deduplication ratio to calculate (estimate) the lower bound and higher bound of the threshold in each segment.
# Set Two Bounds

<table>
<thead>
<tr>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Segment" /></td>
</tr>
<tr>
<td><img src="image" alt="Segment" /></td>
</tr>
<tr>
<td><img src="image" alt="Segment" /></td>
</tr>
<tr>
<td><img src="image" alt="Segment" /></td>
</tr>
<tr>
<td><img src="image" alt="Segment" /></td>
</tr>
<tr>
<td><img src="image" alt="Segment" /></td>
</tr>
<tr>
<td><img src="image" alt="Segment" /></td>
</tr>
</tbody>
</table>
Set Two Bounds

- According to the deduplication ratio reduction limit, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
Set Two Bounds

Segment

- 4 chunks
- 4 chunks
- 4 chunks
- 2 chunk
- 2 chunk
- 1 chunk
- 1 chunk

• According to the deduplication ratio reduction limit, we can estimate, at most, how many data chunks can be rewritten in this segment → higher bound
Set Two Bounds

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment → **lower bound**
Set Two Bounds

According to the deduplication ratio reduction limit, we can estimate, at most, how many data chunks can be rewritten in this segment → higher bound.

According to the target container reads, we can estimate, at most, how many old containers can be referenced in this segment → lower bound.
Set Two Bounds

- According to the deduplication ratio reduction limit, we can estimate, at most, how many data chunks can be rewritten in this segment \(\rightarrow\) **higher bound**
- According to the target container reads, we can estimate, at most, how many old containers can be referenced in this segment \(\rightarrow\) **lower bound**
- The actual CNRC threshold is in between (if cannot be set, satisfy deduplication ratio first)
Set Two Bounds

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**

- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment → **lower bound**

- The **actual CNRC threshold is in between** (if cannot be set, satisfy deduplication ratio first)
Set Two Bounds

Segment

Higher bound

- 4 chunks
- 4 chunks
- 4 chunks
- 2 chunk
- 2 chunk
- 1 chunk
- 1 chunk

Threshold is 3

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment → **lower bound**
- The **actual CNRC threshold is in between** (if cannot be set, satisfy deduplication ratio first)
- If we rewrite fewer chunks or referenced fewer containers in this segment, we can accumulate the “**credits**” of **container reads or rewrites savings** for the future segments to extend the two bounds
**Set Two Bounds**

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment → **lower bound**
- The **actual CNRC threshold is in between** (if cannot be set, satisfy deduplication ratio first)
- If we rewrite fewer chunks or referenced fewer containers in this segment, we can accumulate the “**credits**” of **container reads or rewrites savings** for the future segments to extend the two bounds
- e.g., for next segment, we can move the lower bound **3 containers “down”** and higher bound **4 chunks “up”**.
Agenda

• Deduplication and Restore
• Data Chunk Rewrite Preliminary
• Container Capping Introduction and Limitations
• Proposed Solutions
  – Flexible Container Referenced Count based Design (FCRC)
  – Sliding Look-Back Window (LBW)
• Evaluations
• Conclusions and Future Work
Sliding Look-back Window Assisted Rewrite

- Basic idea: using a sliding window to cover a range of data chunks, ensure that each chunk is evaluated with the **same amount of past and future information**
- To be efficient, the window is moved in container size (4 chunks in this example)
- Data chunks, whose rewrite decision cannot be made, will be temporally cached until the window moves to cover its subsequence. In this way, we can finally make the rewrite decision.
Architecture

Recipe Persistent store

Recipe cache

Look back window

Candidate chunk
Non-rewrite chunk
Immutable recipe entry
Mutable recipe entry
Container

Rewrite candidate cache

Active container

Persistent Storage
Address limitation 3: consider cache effective range when making rewrite decisions

Considering Restore Caching During Rewrite

• According to the caching algorithm and cache space size, we can estimate the cache effective range (all data chunks from read-in container are cached at least # data chunks restore)

For example:
• FAA: the cache effective range is the FAA size (guaranteed)
• Chunk-based cache: if the cache space is $S$ chunks, $x\%$ data chunks in a container are used in average, container size is $C$ chunks, the cache effective range is: (not guaranteed, just the estimate in average)
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always $\leq 2$

![Diagram]

- Recipe cache
- Rewrite candidate cache
- Unique chunk
- Look-back window
- Active container
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always $\leq 2$

![Diagram showing recipe cache, rewrite candidate cache, unique chunk, look-back window, and active container.]

Center for Research in Intelligent Storage
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

• LBW size: 2 containers (8 chunks)
• Cache effective range: 3 containers (12 chunks)
• Rewrite condition:
  • The container has not been referenced for the LBW size chunks
  • The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always ≤ 2

![Diagram showing Recipe cache, Rewrite candidate cache, Look-back window, and Active container]
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always $\leq 2$
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2

![Diagram showing recipe cache, rewrite candidate cache, look-back window, and active container]
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2

![Diagram](image_url)

- Recipe cache
- Rewrite candidate cache
- Look-back window
- Active container
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2
Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always $\leq 2$
Rewrite Policy

- **Combine FCRC algorithm** with the **sliding look-back window** design to decide the CNRC threshold for the containers that to be rewritten.

- Make the **rewrite or non-rewrite decision as early as possible** in each LBW movement to reduce the caching overhead.

- Considering the **cache effect of restore engine** to configure the LBW size and cache sizes.

- Considering the **workload data locality** to slightly adjust the threshold after it is decided, such that the restore engine cache space utilization can be potentially improved.
Agenda

• Deduplication and Restore
• Data Chunk Rewrite Preliminary
• Container Capping Introduction and Limitations
• Proposed Solutions
  – Flexible Container Referenced Count based Design (FCRC)
  – Sliding Look-Back Window (LBW)
• Evaluations
• Conclusions and Future Work
Experiment Setup

• Evaluation metrics
  – **Speed factor:** mean size data being restored (MB) per container read
  – **Deduplication ratio:** total amount of data in the original byte stream divided by the total size of stored unique data chunks

• **Eight deduplication-restore combinations:**
  – Normal – FAA, Normal – ALACC
  – Capping – FAA, Capping – ALACC
  – FCRC – FAA, FCRC – ALACC
  – LBW – FAA, LBW – ALACC

• **Six Traces** with 10 backup versions each from FSL:
  – 3 MacOS server snapshots traces (5, 20, and 60 days backup intervals)
  – 3 students’ home directory snapshots traces (5, 20, and 60 days backup intervals)
Capping vs. LBW

Speed Factor = 3.7

Deduplication Ratio

1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16

0  0.5  1  1.5  2  2.5  3  3.5  4  4.5

LBW-FAA  Capping-FAA  Normal
Capping vs. LBW

- Speed Factor = 3.7
- Deduplication ratio = 14

Graph showing the comparison between Capping vs. LBW with speed factors and deduplication ratios.
Speed Factor Comparison

![Speed Factor Comparison Diagram]

- Normal-FAA
- Capping-FAA
- FCRC-FAA
- LBW-FAA
- Normal-ALACC
- Capping-ALACC
- FCRC-ALACC
- LBW-ALACC

Trace Name:
- MAC1
- MAC2
- MAC3
- FSL1
- FSL2
- FSL3
Detailed Comparison

![Graphs showing comparison of speed factors for different versions.](image-url)
Agenda

• Deduplication and Restore
• Data Chunk Rewrite Preliminary
• Container Capping Introduction and Limitations
• Proposed Solutions
  – Flexible Container Referenced Count based Design (FCRC)
  – Sliding Look-Back Window (LBW)
• Evaluations
• Conclusions and Future Work
Conclusions and Future Work

- Investigated the advantages and limitations of capping

- Proposed an improved scheme based on capping (FCRC), which reduces container reads and rewrites fewer data chunks

- Proposed a new scheme called sliding look-back window based rewrite scheme, which solves the cutting boundary issue of capping and integrated the rewrite algorithm of FCRC

- LBW achieves the best restore performance (highest speed factor) in our evaluations

- In the future, we will investigate the restore friendly GC mechanism.
Thanks!

Q&A
Reference