Reliability-Aware Deduplication Storage: Assuring Chunk Reliability and Chunk Loss Severity

Youngjin Nam, Guanlin Lu, David H. C. Du

Dept. of Computer Science & Engineering
University of Minnesota
http://cris.cs.umn.edu
Talk Outline

• Background & Motivation
  • Our goal & solution
  • Performance analysis
  • Summary & Future work
Background

Deduplication Overview

• Data stream, dedupe appliance, underlying storage
• Dedupe appliance: chunking + deduplication
Background

How Deduplication Works?

• Dividing data(object) into (variable/fixed-sized) small chunks & computing hash (SHA-1) for each chunk
How Deduplication Works?

- Unless a chunk has a copy (already-existing chunk) by looking up hash index, store the new (unique) chunks into storage.
Background

Beauty of Deduplication

- Reuse of the already-existing chunks as many as possible, in order to
  
  ⇒ reduce storage capacity usage
  ⇒ increase write I/O performance by reducing actual write I/O requests to underlying physical storage
Motivation

Deduplication & Reliability

• Some data(file) requires a level of reliability, mostly creating data duplication or using more storage spaces for redundant information

• This is a **conflicting direction** that the data deduplication is pursuing!
Motivation

Typical Steps to Assure Reliability

1. Perform data **deduplication** process (s chunks)
2. **Aggregate** chunks into a large fixed-sized container
3. Reliably store the container over a fixed # of disks through erasure-coding (1-out-of-3)
Motivation

Limitation of Current Approach (1)

- What if Data B demands higher reliability?

⇒ Chunks a2 & a3 can’t be shared by Data B

⇒ Chunk reliability: each chunk should be stored in storage to meet the given reliability
Motivation

Limitation of Current Approach (2)

• What if Chunk “a2” or “a3” is lost?

⇒ Chunk a2 & a3 should be stored more reliably

⇒ Chunk loss severity: severity of a chunk loss without dedupe should be also maintained
Talk Outline

- Background & Motivation
- **Our problem & solution**
- Performance analysis
- Summary & Future work
Chunk Reliability, $C_i^j$ of DSR_i

- Demanded reliability for data stream (DSR_i)

Data stream should tolerate any ONE fragment(disk) loss

Apply erasure-coding to create an redundant fragment

Each chunk also can tolerate any ONE disk loss ($C_i^j$)
Chunk Loss Severity, $\text{CLS}_{ij}$

- **Expected damage level with a chunk loss**
  - actual damage impact (file losses) $\times$ probability of having the damage (a chunk loss)

- With deduplication, the expected damage level will be maintained as low as no-deduplication case
Problem Description

• Given a set of data streams, associative reliability demands & a predefined set of reliability levels $\mathcal{R} = \{r_i|0 \leq i \leq m-1\}$ by storage.

• Devise a reliable deduplication solution to effectively meet the following requirements:

  1. guarantee the demanded reliability levels of the data streams, while performing deduplication (allowing chunk sharing), and

  2. keep CLS of each chunk not to be higher than CLS of no-deduplication case (no chunk sharing).
Proposed Solution (1-1)

• To assure a demanded chunk reliability,

• **Original**: if a chunk has its already-existing copy, then the copy can be used

• **Changed**: if a chunk has its already-existing copy & the demanded reliability of the chunk is not higher than that of the already-existing copy, then the copy can be used
Proposed Solution (1-2)

• What if Data B demands higher reliability?

Data A : 1-loss tolerant

Data B : 2-loss tolerant
Proposed Solution (2-1)

• To assure a chunk loss severity,

• **Chunk sharing level**: keep track of a sharing level of each chunk on real time

• **Increasing chunk reliability**: as a chunk’s sharing level becomes high, increase the chunk reliability accordingly to meet the chunk loss severity of no-deduplication case (1 data loss)
Proposed Solution (2-2)

- The adjusted chunk reliability can be computed as follows:

\[
CR_k^l \leftarrow \min \{r_i \mid r_i \geq 1 - \frac{DSF_k}{S_k^l}, r_i \in \mathcal{R}\}
\]

\[
CLS_i^j = (1 - DSR_i) = DSF_i
\]

Note: Adjusted chunk reliability should be one of the predefined set of reliability levels!
Reliability-aware Deduplication

Algorithm 1 Reliability-aware deduplication

Require: a newly incoming chunk $c_i^j$ in $DS_i$
Ensure: $CR_i^j \geq DSR_i$ and $CLS_i^j \leq DSF_i$ for all $i, j$
1: if ($c_i^j$ has (one or more) duplicate) then
2: // pick $c_k^j$ whose $CR_k^j$ is higher than the other instances of the same chunk
3: if ($CR_i^j > CR_k^j$) then
4: store $c_i^j$ into $DS_i$ container // and remove $c_k^j$
5: $CTR_i$ ← max{$CTR_i, CR_k^j$, $S_k^i = 1$
6: else
7: $S_k^i ← S_k^i + 1$
8: if ($CLS_k^j > DSF_k$) then
9: adjust $CR_k^j ← min\{r_i | r_i ≥ 1 - \frac{DSF_k}{S_k^i}, r_i ∈ ℜ\}$
10: store $c_i^j$ into $DS_k$ container // and remove older $c_k^j$
11: $CTR_k$ ← max{$CTR_k, CR_k^j$}
12: else
13: store the pointer information of $c_k^j$
14: end if
15: end if
16: else
17: store $c_i^j$ into open container of $DS_i$
18: end if
19: if (container of any $DS_m$ is full) then
20: store the container with $CTR_m$
21: prepare another container, $CTR_m ← DSF_m$
22: end if

Assuring demanded chunk reliability is met with already existing copy

Increase sharing level & Assuring chunk loss severity by adjusting chunk reliability
Duplicate or Relocate?

• How to handle the already-existing copy of a chunk whose reliability is lower than the demanded one?

1. Duplicating the copy (prop_dup)
   – should maintain multiple duplicated chunks having different reliability levels
   – extra write I/Os & increased storage capacity usage

2. Relocating the copy (prop_reloc)
   – remove the already-existing copy & write a new copy with a higher reliability level
   – read performance of a data stream using the already-existing copy could be decreased
Performance Analysis

• Experimental setup
  – erasure-coding\((k,m)\): any chunk can be recovered if \(k\)-out-of-\((k+m)\) fragment are available
  – predefined reliability levels: \(r_0=(0,6), r_1=(1,6), r_2=(2,6)\)
    \[ R = \{0.86088(r_0), 0.98719(r_1), 0.99910(r_2)\}, \text{MTTF}=10^6 \text{ hrs} \]
  – workloads: synthetic, realistic (backup traces)

• Five(5) deduplication schemes under analysis
  – min\_rel : always use \(r_0\)
  – max\_rel: always use \(r_2\)
  – prev: typical dedupe (reliability-unaware)
  – prop\_dup & prop\_reloc
Performance Analysis

Storage Capacity Consumption

• Three successive versions of a data stream having different reliability levels (duplication ratio=30%)

prev & prop_reloc are the best (except for min_rel)
Performance Analysis

Guarantee of Chunk Reliability

- With variations of reliability levels,
- Typical deduplication can assure the demanded chunk reliability only 80-90%(60-80%)

\[
\text{dup\_ratio} = 30\%
\]

\[
\text{dup\_ratio} = 60\%
\]

University of Minnesota
Performance Analysis

Guarantee of Chunk Loss Severity

- Increased sharing level (with different dup-ratio=30 or 60%, fixed reliability level=$r_0$),
- **Typical deduplication assure only 30% of the required chunk loss severity** (as low as no-deduplication case)
Concluding Remarks

• Defined a reliable deduplication problem
• Proposed a reliability-aware deduplication scheme (prop_dup, prop_reloc)

• The prop_reloc could be a good solution (with lazy erasure-coding)
  – Idea: always apply the most reliable erasure-coding, say (2,6), while gradually generating only a required number of redundant information on demand
Thank you!
Question & Answering

Youngjin Nam, Guanlin Lu, David H. C. Du
Dept. of Computer Science & Engineering
University of Minnesota
http://cris.cs.umn.edu