Erasure Coding in Windows Azure Storage (WAS)

CHENG HUANG, HUSEYIN SIMITCI, YIKANG XU, AARON OGUS, BRAD CALDER, PARIKSHIT GOPALAN, JIN LI, AND SERGEY YEKHANIN

MICROSOFT CORPORATION

PRESENTED BY VITA KOPILOV & DANIELLE MOVSOWITZ
Outline

• Introduction to Windows Azure Storage (WAS)
• Conventional Erasure Coding (CEC) in WAS - Motivation
• Local Reconstruction Coding (LRC)
• Storage Cost and Performance Trade-offs
• WAS High Level Architecture
• Erasure Coding Implementation In WAS
• Performance
• Summary
Windows Azure Storage

Forms of Storage-

- **Blobs** - User Files
- **Drives** - virtual hard disks for VMs
- **Tables** - Structured storage - DataBase
- **Queues** - message delivery for application running in the cloud

**Extent** - definition:

- Unit of replication
- Size target (3GB), unsealed/sealed
- Erasure-coded in the background and then the copies are deleted.
Conventional Erasure Coding - RS

overhead

(6+3)/6 = 1.5x

(12+4)/12 = 1.33x

sealed extent (3 GB)

\[
\begin{array}{cccccc}
    d_0 & d_1 & d_2 & d_3 & d_4 & d_5 \\
\end{array}
\]

\[
\begin{array}{cccc}
    p_0 & p_1 & p_2 \\
\end{array}
\]

\[
\begin{array}{cccc}
    d_0 & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 & d_9 & d_{10} & d_{11} \\
\end{array}
\]

\[
\begin{array}{cccc}
    p_0 & p_1 & p_2 & p_3 \\
\end{array}
\]
So why not?

reconstruction read is expensive – reading $d_0$ during unavailability
→ requiring 12 fragments (12 disk I/Os, 12 net transfers)
Some key observations

- In convention EC- all failures are equal-> same reconstruction cost.
- In Cloud storage- $P(1\ \text{failure}) >> P(2\ \text{or more failures})$

The Goal

Optimizing Erasure Coding in cloud storage  making single failure reconstruction cost (# of fragments to read) most efficient.
**Local Reconstruction Code - LRC**

**A(6,2,2) LRC Example.** $K=6$ data fragments, $l=2$ local parities, $r=2$ global parities.

Overhead $= (6+2+2)/6 = 1.67x$

Reconstruction cost for 1 failure $= 3$
Coding Equations

\[ q_{x,0} = \alpha_0 x_0 + \alpha_1 x_1 + \alpha_2 x_2 \]  
\[ q_{x,1} = \alpha_0^2 x_0 + \alpha_1^2 x_1 + \alpha_2^2 x_2 \]  
\[ q_{x,2} = x_0 + x_1 + x_2 \]  

and

\[ q_{y,0} = \beta_0 y_0 + \beta_1 y_1 + \beta_2 y_2 \]  
\[ q_{y,1} = \beta_0^2 y_0 + \beta_1^2 y_1 + \beta_2^2 y_2 \]  
\[ q_{y,2} = y_0 + y_1 + y_2. \]  

Then, the LRC coding equations are as follows:

\[ p_0 = q_{x,0} + q_{y,0}, \quad p_1 = q_{x,1} + q_{y,1}, \]  
\[ p_x = q_{x,2}, \quad p_y = q_{y,2}. \]
**Fault Tolerance**

**Definition**-

*Information-theoretically decodable*- failure patterns that are possible to reconstruct.

3 failures- decodable- using $p_0$, $p_x$ and reconstruction of $p_1$.

4 failures- decodable- using all the parities.
Non-decodable- we need 3 parities to reconstruct 3 data fragments.
How can we check if a given failure pattern is information-theoretically decodable?

Algorithm:

- For each local group:
  - If local parity is available and at least 1 data fragment is erased - swap between them. (marks the data fragment as available and the parity as erased).
  - If total # of erased fragments (data + global parities) \( \leq \) global parity, declare decodable. else declare non-decodable.
Fault Tolerance

LRC can tolerate arbitrary 3 failures and 86% of 4 failure cases.

Examples of non trivial cases of decodable 4 failures:

1. None of the four parities fails.
Fault Tolerance

LRC can tolerate arbitrary 3 failures and 86% of 4 failure cases.

2. Only One of local parities fail.
Fault Tolerance

LRC can tolerate arbitrary 3 failures and 86% of 4 failure cases.

3. Both local parities fail
Fault Tolerance

LRC can tolerate arbitrary 3 failures and 86% of 4 failure cases.

We need to ensure:

\[ \alpha_i, \alpha_j, \beta_s, \beta_t \neq 0 \]  \hspace{1cm} (9)
\[ \alpha_i, \alpha_j \neq \beta_s, \beta_t \]  \hspace{1cm} (10)
\[ \alpha_i + \alpha_j \neq \beta_s + \beta_t \]  \hspace{1cm} (11)

By choosing GF(2^4) and \( \alpha' \)'s from the #s divided by 4 and \( \beta' \)'s from #s less then 4 we ensure those conditions are fulfilled.
Theorem 1. for any \((n,k)\) linear code (with \(k\) data symbols and \(n-k\) parity symbols) to have the property:

1. Arbitrary \(r+1\) symbols failures can be decoded.
2. Single data symbol failure can be recovered from \([k/l]\) symbols

The following conditions is necessary:

\[
\begin{align*}
    n - k & \geq l + r
\end{align*}
\]

LRC meets the lower bound exactly achieves its properties with the minimal # of parities.
Code Parameters Selection

• Each fragment has to place on a different fault domain.
• In WAS we use 20 fault domains.
• We vary the parameter $k$ and $r$ (as long as $k + r \leq 20$) and choose the lower bound of all the trade-off points.
Comparison to RS

![Comparison to RS Diagram](image)

- **STORAGE COST AND PERFORMANCE TRADE-OFFS**
  - Comparison to RS
    - **RS(10,4)**
      - Same read cost: 1.5x to 1.33x
    - **RS(6,3)**
      - Same overhead: half read cost
    - **LRC(12,2,2)**
    - **LRC(12,4,2)**
Comparison to Modern Storage Codes
WAS High Level Architecture
Access blob storage via the URL: http://<account>.blob.core.windows.net/
WAS Stream Layer Architecture

- Append – only distributed file system
- Streams are very large files

Stream Operations:
- Open, Close, Delete streams
- Rename streams
- Concatenate streams together
- Appends for writing
- Random reads
**WAS Stream Layer Concepts**

**Block**
- Min unit of write/read
- Checksum
- Up to N bytes (e.g. 4MB)

**Extent**
- Unit of replication
- Sequence of blocks
- Size limit (1GB – 3GB)
- Sealed/unsealed

**Stream**
- Hierarchical namespace
- Ordered list of pointers to extents
- Append/Concatenate

**Diagram**
- Stream //foo/myfile.data
- Pointer E1 to Extent E1 sealed
- Pointer E2 to Extent E2 unsealed
- Pointer E3 to Extent E3 unsealed
- Pointer E4 to Extent E4 unsealed
Creating an Extent

Partition Layer

Create Stream/Extent

EN1 Primary
EN2, EN3 Secondary

Primary
Secondary A
Secondary B

Allocate Extent replica set

Stream Master

Paxos

EN
Replication Flow

Partition Layer

EN1 Primary
EN2, EN3 Secondary

Paxos

WAS HIGH LEVEL ARCHITECTURE
Step 1:
The SM determines the set of ENs that will store the fragments of the extent that is being erasure coded.

Example: 16 fragments for LRC(12,2,2)
Step 2:

The SM:

- Assigns a Coordinator EN from the Extents Replica Set
- Sends it all the relevant metadata needed for erasure coding
Step 2:
The coordinator:
- Divides the Extent into fragments at append blocks
Step 2:
The Coordinator:
- Communicates the fragment boundaries to the target ENs
- Starts the encoding process and keeps sending the encoded fragments to their designated Ens
- All of the ENs keep track of the progress made and persist that information into each new fragment.

If a failure occurs at any moment during the process, the rest of the work can be picked up by another EN.
Step 3:

The Coordinator:

- notifies the SM the process has ended.

The SM:

- Updates the metadata of the extent with fragment boundaries and completion flags.
- Schedules full replicas of the extent for deletion.
Reading a fragment
Reading a fragment:

The fragments of an extent can be read directly by contacting the EN that has the fragment.
Target EN is unavailable or a Hot Spot

The Client:
- Contacts one of the other ENs holding the extents fragments

The Contacted EN:
- Reads the other needed fragments
- Reconstructs the fragment
- Caches the reconstructed fragment locally in case there are other reads
- Returns the results to the client

If the EN or the disk drive that hosts the extent fragment is unavailable for an extended period of time, the SM initiates the reconstruction of the fragment on a different EN
Using LRC reconstruction codes in WAS

- Each extent is divided into $k$ equal-size data fragments
- $l$ local and $r$ global parity fragments are created
- During an upgrade period, when one upgrade domain is taken offline, every single data fragment can still be accessed efficiently
LRC(12,2,2) example:

Each of the 16 data fragments is placed in a different rack.
Designing for Erasure Coding

Scheduling of Various I/O Types

- To make the system fair and responsive, operations are subject to throttling and scheduling at all levels of the storage system:
  - Every EN keeps track of its load at the network ports and on individual disks
  - The SM keeps track of data replication load on individual ENs and the system as a whole
Designing for Erasure Coding

Reconstruction Read-ahead and Caching

- Reconstruction of unavailable fragments is done in unit sizes greater than the individual append blocks (up to 5MB) to reduce the number of disk and network I/Os.

- The read-ahead data is cached in memory (up to 256MB) of the EN that has done the reconstruction.

- Further sequential reads are satisfied directly from memory.
Designing for Erasure Coding

Consistency of Coded Data

- WAS, use various CRC (Cyclic Redundancy Check) fields to detect data and metadata corruptions.

- After each erasure encoding operation, several decoding combinations are tried.

- When an operation fails due to CRC checks:
  - The operation is retried using other combinations.
  - The fragment with the corrupted block is scheduled for regeneration.
Designing for Erasure Coding

Consistency of Coded Data – LRC(12,2,2) example:

- Randomly choose one data fragment in each local group and reconstruct it using its local group.
- Randomly choose one data fragment and reconstruct it using one global parity and the remaining data fragments.
- Randomly choose one data fragment and reconstruct it using the other global parity and the remaining data fragments.
- Randomly choose two data fragments and reconstruct them.
- Randomly choose three data fragments and reconstruct them.
- Randomly choose four data fragments (at least one in each group) and reconstruct them.
**Performance**

Compare LRC (12, 2, 2) to Reed-Solomon (12, 4):

- Both yield storage overhead cost at 1.33x.
- (12, 3) Reed-Solomon is not an option - its reliability is lower than 3-replication.
Small I/Os:

Trade-off:
Number of I/Os vs Latency
- RS - read 12-15 fragments out of 15.
- LRC - only 6 fragments need to be read.

Figure 9: Small (4KB) I/O Reconstruction - (12, 4) Reed-Solomon vs. (12, 2, 2) LRC.
Large I/Os:

Trade-off:

Latency vs Bandwidth Consumption

Figure 10: Large (4MB) I/O Reconstruction - (12, 4) Reed-Solomon vs. (12, 2, 2) LRC.
Decoding Latency – LRC vs RS

- Average latency of decoding 4KB fragments:
  - 13.2us for Reed-Solomon
  - 7.12us for LRC.

- Decoding is faster in LRC because only half the number of fragments are involved.

- However, The gain of faster decoding, **would not** matter in WAS. (since the decoding time is orders of magnitude smaller than the transfer time)
Summary

- LRC is not MDS - requires higher storage overhead for the same fault tolerance. The additional storage overhead are exploited for efficient reconstruction.

- Erasure coding is critical to reduce the cost of cloud storage - our target storage overhead is 1.33x of the original data.

- LRC (12, 2, 2), which has a storage overhead of 1.33x, saves significant I/Os and bandwidth during reconstruction when compared to Reed-Solomon (12, 4).

- WAS uses LRC (12, 2, 2) because:
  - It achieves the 1.33x storage overhead target
  - Has the above latency, I/O and bandwidth advantages over Reed-Solomon.
  - Provides better durability than the traditional approach of keeping 3 copies.

- Showed how to efficiently lay out LRC (12, 2, 2) across the 20 fault domains and 10 upgrade domains used in Windows Azure Storage.