Swift Object Storage: Adding Erasure Codes

Paul Luse, Sr. Staff Engineer – Intel Corporation
Kevin Greenan, Staff Software Engineer – Box
Sep 2014
The material contained in this tutorial is copyrighted by the SNIA unless otherwise noted.

Member companies and individual members may use this material in presentations and literature under the following conditions:
- Any slide or slides used must be reproduced in their entirety without modification
- The SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.

This presentation is a project of the SNIA Education Committee.

Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.

The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.
Abstract

Swift Object Storage: Adding Erasure Codes

This session will provide insight into this extremely successful community effort of adding an Erasure Code capability to the OpenStack Swift Object Storage System by walking the audience through the design and development experience through the eyes of the developers from key contributors. An overview of Swift Architecture and basic Erasure Codes will be followed by design/implementation details.
Agenda

• Swift
  – A Community Project
  – Swift Overview
  – Storage Policies

• Erasure Codes
  – History
  – Variations
  – Matrix encode/decode
  – PyECLib & liberasurecode

• Erasure Code Implementation for Swift
  – Design considerations
  – Architecture overview
Swift: A Community Project

• Core OpenStack* Service
  • One of the original 2 projects
  • 100% Python
  • ~ 35K LOC
  • > 2x that in unit, functional, error injection code

• Vibrant community,
  • top contributing companies for Juno include: SwiftStack*, Intel, Redhat*, IBM*, HP*, Rackspace*, Box*

• The path to EC…

<table>
<thead>
<tr>
<th>Early EC Discussion</th>
<th>Policies Development</th>
<th>EC Library Development</th>
<th>Policies Released</th>
<th>Swift EC Development</th>
<th>EC Ready!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Havana</td>
<td>IceHouse</td>
<td>Juno</td>
<td>Kilo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fall '13</td>
<td>Spring '14</td>
<td>Fall '14</td>
<td>Spring '15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Swift Overview

- Uses container model for grouping objects with like characteristics
  - Objects are identified by their paths and have user-defined metadata associated with them

- Access via RESTful interface
  - GET, PUT, DELETE

- Built upon standard hardware and highly scalable
  - Cost effective, efficient

Objects are organized with containers
What Swift is Not

• Distributed File System
  – Does not provide POSIX file system API support

• Relational Database
  – Does not support ACID semantics

• NoSQL Data Store
  – Not built on the Key-Value/Document/Column-Family model

• Block Storage System
  – Does not provide block-level storage service

Not a “One Size Fits All” Storage Solution
Swift Software Architecture

Proxy Nodes
- wsgi server
- middleware
- swift proxy
  - wsgi application
  - account controller
  - object controller
  - container controller
  - helper functions

Storage Nodes
- wsgi server
- middleware
- swift object
  - wsgi application
  - replicator
  - expirer
  - auditor
  - updater
- swift account
  - wsgi application
  - replicator
  - reaper
  - auditor
  - helper functions
- swift container
  - wsgi application
  - replicator
  - sync
  - auditor
  - updater

© 2014 Storage Networking Industry Association. All Rights Reserved.
Swift 2.0: Why Storage Policies?

- Prior to, durability scheme applies to entire cluster
  - Can do replication of 2x, 3x, etc., however the entire cluster must use that setting

- There were no core capabilities to expose or make use of differentiated hardware within the cluster
  - If several nodes of a cluster have newer/faster characteristics, they can't be fully realized (the administrator/users are at the mercy of the dispersion algorithm alone for data placement).

- There's was no extensibility for additional durability schemes
  - Use of erasure codes (EC)
  - Mixed use of schemes (some nodes do 2x, some do 3x, some do EC)
The Swift Ring

- The ring is a static data structure maintained external to the cluster (tools provided)
- An object name maps to a partition via MD5 hash
- Each partition maps to a list of devices via two array elements within the ring structure
- Devices are assigned to partitions with several policies (regions, zones, etc.) and constraints to assure fault tolerance and load balancing

<table>
<thead>
<tr>
<th>Idx</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Node 3, device 1</td>
</tr>
<tr>
<td>1</td>
<td>Node 12, device 2</td>
</tr>
<tr>
<td>34</td>
<td>Node 1, device 4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

MD5*(URL) = index

Note: Swift uses a modified MD5 consistent hashing ring
What are Policies?

• Introduction of multiple object rings
  - Swift supports multiple rings already, but only one for object – the others are for account and container DB.

• Introduction of container tag: X-Storage-Policy
  - New immutable container metadata
  - Policy change accomplished via data movement
  - Each container is associated with a potentially different ring

- Triple Replication
  - 3 locations, same object

- Reduced Replication
  - 2 locations, same object

- Erasure Codes
  - n locations, object fragments

Swift Object Storage: Adding Erasure Codes
© 2014 Storage Networking Industry Association. All Rights Reserved.
Swift Object Storage: Adding Erasure Codes

Putting it All Together

Upload

Clients

Download

RESTful API, Similar to S3

Access Tier

- Handle incoming requests
- Handle failures, ganged responses
- Scalable shared nothing architecture
- Consistent hashing ring distribution

Load Balancer

Auth Service

Proxy

Copy 1

Copy 2

Copy 3

Zone 1

Zone 2

Zone 3

Zone 4

Zone 5

Capacity Tier

- Actual object storage
- Variable replication count
- Data integrity services
- Scale-out capacity

Scalable for concurrency and/or capacity independently

© 2014 Storage Networking Industry Association. All Rights Reserved.
Agenda

• Swift
  – Swift Overview
  – Storage Policies

• Erasure Codes
  – Background
  – Example encode/decode using Reed-Solomon
  – Minimizing reconstruction cost
  – PyECLib & liberasurecode

• Erasure Code Implementation for Swift
  – Design considerations
  – Architecture overview
History of Erasure Coding

1960's
- Coding Theory
  - Reed Solomon, Berlekamp–Massey algorithm

1990's
- Storage
  - RAID-6: EVENODD, RDP, X-Code
- Graph Theory
  - LDPC Codes (Tornado, Raptor, LT)

2000's
- Coding Theory
  - Network / Regenerating Codes

2010's
- Storage
  - Non-MDS codes for cloud and recovery

Terminology

• Split a file into $k$ chunks and encode into $n$ chunks, where $n-k=m$
• Systematic vs. Non-systematic
  – Systematic: encoded output contains input symbols
  – Non-systematic: encoded output does not contain input symbols
• Code word
  – A set of data and parity related via a set of parity equations
  – Systematic: $f(\text{data}) = \text{code word} = (\text{data, parity})$
• Layout
  – Flat horizontal: each coded symbol is mapped to one device
  – Array codes have multiple symbols per device: horizontal and vertical
• MDS vs. non-MDS
  – MDS: any $k$ chunks can be used to recover the original file
  – Non-MDS: $k$ chunks may not be sufficient to recover the file

Traditionally, storage systems use systematic, MDS codes
Variations

- Reed-Solomon Codes
- Fountain Codes
- RAID-6 EVENODD
- RAID-6 X-Code
- Generalized XOR
- Pyramid Codes
- Local Repairable Codes (LRC)
- Partial MDS Codes (PMDS)
- Simple Regenerating Codes

and the list goes on…
Reed Solomon Systematic Horizontal Code Layout

Example RS(8,5)

- Total disks = \( n = 8 \)
- Data disks = \( k = 5 \)
- Parity disks = \( m = 3 \)

- Can Tolerate \((n-k)\) Failures
- Overhead of just \((n/k)\)

Code word

\[
\begin{align*}
&d_{0,0} &d_{1,0} &d_{2,0} &d_{3,0} &d_{4,0} \\
&d_{0,1} &d_{1,1} &d_{2,1} &d_{3,1} &d_{4,1} \\
&d_{0,2} &d_{1,2} &d_{2,2} &d_{3,2} &d_{4,2} \\
&d_{0,3} &d_{1,3} &d_{2,3} &d_{3,3} &d_{4,3} \\
&P_{0,0} &P_{1,0} &P_{2,0} \\
&P_{0,1} &P_{1,1} &P_{2,1} \\
&P_{0,2} &P_{1,2} &P_{2,2} \\
&P_{0,3} &P_{1,3} &P_{2,3}
\end{align*}
\]
Reed Solomon Systematic Generator Matrix

\[ f(\alpha_0) = y_0 \]
\[ f(\alpha_1) = y_1 \]
\[ \vdots \]
\[ f(\alpha_{n-1}) = y_{n-1} \]

Reed-Solomon is *encoded* by oversampling a polynomial

\[ f(x) = c_0 + c_1 x^1 + c_2 x^2 + \ldots + c_{k-1} x^{k-1} \]

Coefficients are the data

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 \\
g_0 & g_1 & g_2 & g_3 & g_4 \\
g_5 & g_6 & g_7 & g_8 & g_9
\end{bmatrix}
\]

Elementary Ops

Result has same rank

\( f(\alpha_0) \)
# Reed Solomon Systematic Matrix Encoding Process

All operations are done in a Galois field

Any (k x k) sub-matrix is invertible

Code word is the vector-matrix product of the generator matrix and source data

\[ p_i = d_0 + g_i d_1 + g_i^2 d_2 + g_i^3 d_3 + \ldots + g_i^{k-1} d_{k-1} \]
**Reed Solomon Systematic Matrix Decoding Process**

<table>
<thead>
<tr>
<th></th>
<th>g₀</th>
<th>g₁</th>
<th>g₂</th>
<th>g₃</th>
<th>g₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>g₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 1: Eliminate all but k available rows in the generator matrix.
Reed Solomon Systematic Matrix Decoding Process

$$\begin{bmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 \\
g_0 & g_1 & g_2 & g_3 & g_4
\end{bmatrix} \rightarrow \begin{bmatrix}
G' 
\end{bmatrix}$$

Step 2: Invert the resulting matrix
Reed Solomon Systematic Matrix Decoding Process

Step 3: “Solve” by multiplying k element vector of available symbols by corresponding rows of $G'$

\[
\begin{align*}
    d_0 &= g'_{00}d_1 + g'_{01}d_2 + g'_{02}d_4 + g'_{03}p_1 + g'_{04}p_2 \\
    d_3 &= g'_{30}d_1 + g'_{31}d_2 + g'_{32}d_4 + g'_{33}p_1 + g'_{34}p_2
\end{align*}
\]
Minimizing Reconstruction Cost

- Reed-Solomon requires \( k \) available elements to reconstruct any missing element
- This has given rise to many codes that minimize repair costs
  - Regenerating codes, locally repairable codes, flat-XOR codes, etc.
  - Trade space efficiency for more efficient reconstruction
- Replication repair-optimal, RS is space-optimal, these are somewhere in the middle
- Simple XOR-only example with \( k = 6, m = 4 \):

\[
\begin{align*}
P_0 &= \times D_0 + D_1 + D_3 \\
P_1 &= D_1 + D_2 + D_5 \\
P_2 &= \times D_0 + D_2 + D_4 \\
P_3 &= D_3 + D_4 + D_5 \\
D_0 &= P_0 + D_1 + D_3 \\
D_0 &= P_2 + D_2 + D_4
\end{align*}
\]

Only requires 3 devices to reconstruct one failed device
• **Goal**: provide a pluggable, easy-to-use EC library for Python
• Swift is the main use-case
• Originally had *all logic* in PyECLib, but have offloaded “smarts” to liberasurecode
  – Separation of concerns: one converts between Python and C, and the other does erasure coding
  – API of PyECLib is same as liberasurecode

PyECLib

<table>
<thead>
<tr>
<th>API</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>encode</td>
<td>Python-C Interface</td>
</tr>
<tr>
<td>decode</td>
<td>liberasurecode</td>
</tr>
<tr>
<td>reconstruct</td>
<td></td>
</tr>
<tr>
<td>required.fragments</td>
<td></td>
</tr>
<tr>
<td>check.metadata</td>
<td></td>
</tr>
<tr>
<td>segment_info</td>
<td></td>
</tr>
</tbody>
</table>
**liberasurecode**

- **Goal**: Separate EC-specific logic from language-specific translation
- **Embedded metadata**: original file size, checksum, version info, etc.
- **Provides ability to plug-in and use new erasure code schemes/libraries**
  - In addition to XOR codes, we currently provide Jerasure and ISA-L

![Diagram of liberasurecode](https://example.com/diagram.png)

- **API**
  - encode
  - decode
  - reconstruct
  - required_fragments
  - check_metadata
  - segment_info

- **Core**
  - Backend plug-in layer
  - built-in
  - utils
  - Backend API
  - XOR Codes
  - Checksums
  - Helpers
  - Pre/Post-Processing

- **External Libraries**
  - libA
  - libB
  - ...
Agenda

- Swift
  - Swift Overview
  - Storage Policies
- Erasure Codes
  - History
  - Variations
  - Matrix encode/decode
  - PyECLib & liberasurecode
- Erasure Code Implementation for Swift
  - Design considerations
  - Architecture overview
**Design Considerations**

- **GET/PUT Erasure Code encode/decode done at proxy server**
  - Aligned with current Swift architecture to focus hardware demanding services in the access tier
  - Enable in-line Erasure Code directed by client as well as off-line Erasure Code directed by sideband application / management tier

- **Build Upon Storage Policies**
  - New container metadata will identify whether objects within it are erasure coded

- **Keep it simple and leverage current architecture**
  - Multiple new storage node services required to assure Erasure Code chunk integrity as well as Erasure Code stripe integrity; modeled after replica services
  - Storage nodes participate in Erasure Code encode/decode for reconstruction analogous to replication services synchronizing objects
Swift With EC Architecture High Level

Proxy Nodes
- wsgi server
- middleware
- swift proxy
  - wsgi application
  - existing modules
  - controller modifications
    - PyECLib
      - Plug in 1
      - Plug in 2

Storage Nodes
- wsgi server
- middleware
- swift object
  - wsgi application
  - existing modules
  - EC Auditor
  - PyECLib
    - Plug in 1
    - Plug in 2
  - EC Reconstructor
- swift container
  - wsgi application
  - existing modules
  - metadata changes
- swift account
  - wsgi application
  - existing modules

© 2014 Storage Networking Industry Association. All Rights Reserved.
Swift With Erasure Code

- Applications control policy
- EC can be inline or offline

- Supports multiple policies
- EC flexibility via plug-in

Erasure Code Technology Lowering TCO for Swift
For More Information...

- Trello discussion board:
  https://trello.com/b/LlvIFIQs/swift-erasure-codes
- Launchpad blueprints:
  https://blueprints.launchpad.net/swift
- Swift Code (see feature/EC branch):
  https://code.launchpad.net/swift
- PyECLib:
  https://bitbucket.org/kmgreen2/pyeclib
- Liberasurecode:
  https://bitbucket.org/tsg-/liberasurecode
The SNIA Education Committee thanks the following individuals for their contributions to this Tutorial.

Authorship History
Name/Date of Original Author here:
Paul Luse, Kevin Greenan. 8/2014
Updates:
None

Additional Contributors
None

Please send any questions or comments regarding this SNIA Tutorial to tracktutorials@snia.org
Backup
Block, File & Object

Block

Specific location on disks / memory

Tracks
Sectors

File

Specific folder in fixed logical order

File path
File name
Date

Object

Flexible container size

Data and Metadata
Unique ID
Object Store

- **Scalability**
  - Flat namespace
  - No volume semantics
  - No Locking/Attributes
  - Contains metadata

- **Durability**
  - Replication or Erasure code

- **Manageability**
  - REST API
  - Low overhead

- **Consistency**
  - Eventually consistent
Swift Object Storage: Adding Erasure Codes

© 2014 Storage Networking Industry Association. All Rights Reserved.

**Web/Mobile**
- Images/Audio/Video
- Middleware for dynamic resizing

**VM Images**
- OpenStack* Integration
- Managed by Glance

**Cloud Backup**
- Large unstructured data

**IAAS**
- Public or private
- Multi-tenant storage
- Multiple available gateways

---

**Swift Cluster**

**Access Tier**
- Load Balancer
- Proxy Nodes
- Auth

**Capacity Tier**
- Storage Node
- Storage Node

---

Swift Object Storage: Adding Erasure Codes
© 2014 Storage Networking Industry Association. All Rights Reserved.
Swift Object Storage: Adding Erasure Codes

© 2014 Storage Networking Industry Association. All Rights Reserved.

The CAP Theorem: Pick 2

- **Consistency** (same view)
  - Enforced Consistency

- **Availability** (data access)
  - Eventual Consistency

- **Partition Tolerance** (node access)

Swift chooses **Availability** and **Partition Tolerance** over **Consistency**