Proxy-centric EC design
Target Use Case

- Clients understand their own data lifecycle
- Clients understand their storage/latency tradeoffs and benefits
- Very large data sets
- Migration and upgrade path for existing Swift clusters
High-level

• All EC encoding and decoding is done in the proxy server
• Keeps CPU-heavy SKUs in access tier (proxy servers)
• Trends for storage are to lower CPU per TB
• Objects are explicitly marked as EC
• Allows for client-driven inline and background movement
• Clients have control of lifecycle management
Granularity Option: Per Container

- **Pros:**
  - Same as existing granularity in Swift (ACLs, CORS, versioning, etc)
  - Cacheable EC flag and parameters (per-container EC policy)
  - Can still use existing manifest files as indirection objects

- **Cons:**
  - Expensive reconciliation in rare cases (split-brain cluster + non-coordinated clients)
  - Set at container creation, can’t toggle
Granularity Option: Per Object

(Included for completeness, but not recommended)

• Pros:
  • Can mix replicas and EC in a single container

• Cons:
  • Double object count (very expensive at scale)
  • Extra lookup latency (likely minor for larger EC objects)
  • Can create orphans (very expensive to clean up)
Overall Pieces

- EC Object Controller in proxy
- EC Auditor daemon
- EC Reconstructor daemon
- EC Stripe Auditor daemon
- EC Ring
Proxy Server

• New ECOObjectController to handle the encoding and decoding of EC data

• Controller chosen today by URL parts; new scheme will use request header or container metadata

• Allows for future storage policies to be implemented here
  • eg reduced redundancy, fast vs slow, global vs local

• Allows existing clusters to scale EC capacity separately
Replicated to EC data process

This process can be initiated by either an end-user or a process external to the cluster. Additionally, a separate proxy to handle this migration could be set up.

1. client uploads object as replicated data

2. later, a client (or the same one) issues a COPY request to copy the data from replicated to an EC container

3. on success, overwrite the replicated object with a manifest object pointing at the EC object
Proxy-side Implications

- Client chooses to EC data by putting it in a specific container or setting a particular header
- Migration could still be done by a helper process
- Proxy servers are already optimized for CPU and network throughput
- Keeps request load independently scalable from storage capacity
Which EC scheme?

- Support multiple EC policies
- Use an external library for the EC bits
  - Requires a common interface between libraries
  - Intel (ISA-L), Box (Kevin)
- Allows for differently-licensed libraries
On-Disk Impact

- EC data stored in a new high-level directory
- Each EC scheme (storage policy) has a new top-level directory
- EC chunks stored as files on disk
  - eg 10 files total across the cluster in an 8:2 scheme
EC Quorum Concept

- One plus minimum chunks
- eg 9 in an 8 + 4 scheme
- Keep same durability model as replicas
  - ie “at least 2 drive failures” between client and data loss
PUT

• Proxy gets request and determines that it’s EC data

• Looks up locations in EC ring

• Read some data off the wire from client, EC it, stream to storage nodes

• Return success if quorum chunks written successfully
GET

- Proxy gets request and determines that it’s EC data
- Looks up locations in EC ring
  - Configurable, default to “read all”
- Read chunks from storage nodes, decoding EC data
  - Errors allow for early failure alerting
DELETE

- Request comes to proxy and is determined to be for EC data
- Tombstone sent to storage nodes
- Success on quorum written
EC Auditor

• Exactly like existing auditor

• Could even be the exact same code, just looking at a different top-level directory

• Existing auditors could be updated to audit both replicas and EC data

• Check individual chunks with their own checksum
EC Reconstructor

- On each storage node for its local data:
  - Check siblings
  - If sibling missing or out of date, reconstruct and push that chunk
  - Push to handoff if necessary (same as in replica storage)
  - If newer data is found, reconstruct the local data
  - If a newer tombstone is found, add it locally
EC Reconstructor

- For all partitions stored locally in ec-objects
- Check siblings
- Reconstruct missing data if sibling isn’t found (push to handoff)
- Reconstruct local data if newer data is on the sibling
EC Stripe Auditor

- Local process to check full object health
- Compute and store chunk fragment signature (Algebraic Signatures)
- Query remote signatures for whole-object validity
- If corruption found, delete offending chunk (rebuilder repairs)
- Protects against parity poisoning
EC Ring

• Stores EC parameters

• Placement and rebalancing can take parameters into account
  • eg Keep locations in a single region
  • eg How many locations to reassign during one rebalance

• Allows for a separate set of storage media distinct from replicated data
EC Ring Example

- Given 10+4 RS coding parameters:
  - EC Quorum is 11 volumes
  - 14 locations stored for each partition
  - A rebalance only moves 3 locations at one time
Failure Handling: Drive

- When detected, push chunk data to a handoff drive
- Same strategy as existing replica storage
Failure Handling: Server

• Don’t repair automatically
• Node failure is a temporary availability, not durability, issue
• Same strategy as existing replica storage
Failure Handling: During Write

- Use handoff if error detected in connect phase
- Require quorum successful for client success
- Same strategy as existing replica storage
Failure Handling: During Read

- If chunk checksum error, locally quarantine chunk
- If chunk fails, reconstruct during read
  - Alert but don’t repair
  - EC reconstructor will repair
Global Clusters

- EC Ring specific to a particular region
- Future work may be for EC chunks to be replicated to another region
Metadata

• “Whole-object” metadata stored on each EC chunk
  
  • ie Content-Length, Content-Type, X-Object-Meta-*, etc
  
• EC-specific metadata stored with each chunk but not served to users
  
  • Used to check per-chunk checksums
  
  • eg per-chunk checksums, algebraic signature of stripes, etc