Geographically distributed Swift clusters

Alistair Coles
Swift core developer

alistairncoles@gmail.com
irc: acoles
Overview

• What is Swift?
• Geographically distributed clusters
  • What?
  • Why?
  • How?
• Erasure coded geographically distributed cluster
  • Swift now supports these!
  • ...enabled by fragment duplication and composite rings
• Summary
What is Swift?

• *openstack.* object storage service

• REST API – Create, Read, Update, Delete

• Simple naming hierarchy
  • objects belong to containers
  • containers belong to accounts
Swift is durable

- Multiple replicas of every object (or erasure coding)
- The Ring always tries to disperse replicas across different devices and servers
Swift is scalable

- The Ring always tries to balance load across all devices
- No centralized services
Swift is highly available

- Write succeeds on quorum of replicas
- Missing replicas are updated asynchronously
Swift is eventually consistent

- Possibility to read stale data
- Async process makes data eventually consistent
Overview

• What is Swift?

• Geographically distributed clusters
  • What?
  • Why?
  • How?

• Erasure coded geographically distributed cluster
  • Swift now supports these!
  • … but there’s some new stuff to know about

• Deep dive: erasure coding, fragment duplication, composite rings

• Summary
Geographically distributed clusters

• What?
  • Multiple physical locations
    • Typically connected by high latency/low bandwidth WAN
  • Copies of data in each location
  • Single namespace

• Why?
  • Disaster recovery
  • Data locality

• Also known as “Global clusters”, “Multi-region Swift”
The Ring always tries to disperse replicas across different devices and servers ...and regions
Geographically distributed clusters - *disaster recovery*

- A 4-replica policy makes each region independently robust to a single device failure
Geographically distributed Swift clusters - data locality

- By default the ring will try to balance read load by choosing random replicas
Read affinity – trade off load balancing for read performance

GET a/c/o @ t1
GET a/c/o @ t2
GET a/c/o @ t3
GET a/c/o @ t4

Proxy server

Ring

read_affinity -> region1

Proxy server

Ring

read_affinity -> region2

always read replica in local region

WAN

Region1

Region2
What if remote writes fail?

- Remote replicas are written to temporary locations.
- Async process moves them when remote region is available.
Overwrite - replicas are *eventually* consistent

PUT a/c/o @ t1
  ‘my old data’

PUT a/c/o @ t3
  ‘my new data’

GET a/c/o @ t4
  ‘my old data’

WAN fails at t2

async move to remote region

temporarily stale replicas
What if remote writes are slow?

"But isn’t this a terrible idea? All my writes will be slowed down by requests to the remote region!"
Effect of remote region write time on PUTs
(remote region servers artificially slowed)

post_quorum_timeout = 0.5

post_quorum_timeout puts an upper bound on the extra latency
Write affinity – temporarily trade off *dispersion* for write performance

- Remote replicas are initially written to the local region.
- Async process moves them to remote region.
Write affinity performance improvement
(remote region servers artificially slowed)
Write affinity – data is always available

PUT a/c/o @ t1 ‘my data’

Proxy server

write_affinity = region1

Proxy server

GET a/c/o @ t1 + ∆ ‘my data’

reads fall back to remote region

WAN

SwiftStack
Write affinity – trade off consistency for write performance

PUT a/c/o @ t1
‘my data’

PUT a/c/o @ t2
‘my new data’

GET a/c/o @ t2 + ∂
‘my data’

write_affinity = region1

async move to remote region

temporarily stale replicas
Workload considerations

• Use write affinity with care!
  • “No free lunch” - replicas have to be copied across WAN eventually

• Suitable workloads:
  • Moderate or bursty write rates
  • Non-immediate remote reads
  • E.g. replicating archive data across sites

• Unsuitable workloads:
  • Continuous high write rate
    • misplaced replicas will back up in local region
  • Immediate remote reads
    • reads will fetch data over WAN before async move has happened
Overview

• What is Swift?

• Geographically distributed clusters
  • What?
  • Why?
  • How?

• Erasure coded geographically distributed cluster
  • Swift now supports these!
  • ... but there’s some new stuff to know about

• Deep dive: erasure coding, fragment duplication, composite rings

• Summary
Erasure coding – same durability, less storage

Example: 4 data fragments + 2 parity fragments

Any subset of 4 unique fragments is sufficient to reconstruct data:

Requires only \textbf{1.5 x size of data} to store all fragments
Erasure coding in Swift

https://github.com/openstack/pyeclib
Python interface to liberasurecode

https://github.com/openstack/liberasurecode
C library with pluggable Erasure Code backends

4 + 2 erasure coding requires approx. 50% storage vs 3 replicas with similar durability
Erasure coding across regions

- The Ring always tries to disperse **fragments** across different devices and servers ...and **regions**
Erasure coding across regions

With EC 4+2 we don’t have enough fragments in one region to reconstruct the data!
Erasure coding across regions requires more fragments

How about EC 4+6? requires **2.5 x size of data**

vs

Replication requires **4 x size of data** for similar durability

SwiftStack
Erasure coding time increases with number of parity fragments
EC duplication: more fragments, less compute

Each region has EC 4+1 fragments requires **2.5 x size of data**

vs

Replication requires **4 x size of data** for similar durability
But duplicates must be correctly dispersed...

We don’t have enough **unique** fragments in this region to reconstruct the data!
Solution: EC duplication + composite rings

Each region has its own ‘component’ ring - this guarantees a set of unique fragments in each region.
Summary

• Swift enables you to build geographically distributed clusters
• Good for disaster recovery and data locality
• Tuning via options in the Swift proxy server
  • `write_affinity`, `read_affinity`
  • Understand your workloads!
• Erasure coded storage policies can also be geographically distributed\(^1\)
  • Uses new features: EC fragment duplication and composite rings

\(^1\) new in Swift 2.15.0
Swift welcomes new users and contributors

- You can find us in freenode #openstack-swift
- Project: https://launchpad.net/swift
- Docs: https://docs.openstack.org/swift
- Code: https://github.com/openstack/swift