SDC 20

BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

Decoupling SDS Architectures for Agility

Arun Raghunath (Research Scientist)
Yi Zou (Research Scientist)

Intel Corp.

The Case for Decoupling SDS Architectures

Previously, we have illustrated,

- SDS decoupling via data and control plane separation
  - ✓ Ceph, SPDK, and NVMe-oF ecosystem
- Benefits of decoupling when storage is disaggregated
  - ✓ Removes extra hop ("datacenter tax") -> reduce latency & bw cost [SDC2018 talk]
- Decoupled SDS architecture PoC [SDC2019 talk]
  - ✓ Scale-out of multiple heterogenous storage services
  - ✓ Separating of failure domains and benefits to recovery
  - ✓ Integration of next-gen storage and mixed media types [SDC2020: Mortimer talk]

Focus of this talk,

→ ***Agility must be considered in SDS architecture design!***

SDS = Software Defined Storage

The Need for Agility in SDS

Emerging elastic workloads

- Applications based on serverless computing paradigm
  - Enable massive scaling based on event-based triggers
  - Demand high throughput from storage services in short duration bursts
- Video analytics pipelines, AI/ML workloads
  - Require low latency access to storage
  - Need performance boosts for subsets of data only
- Applications built using Microservices pipelines or graphs
  - Desire efficient data movement and storage capabilities
- Edge and mobile workloads are inherently dynamic in nature
  - Low latency services needed close to client

Agility Bottlenecks in Current SDS Architectures

- Offer capacity scale-out by adding new nodes
  - But this is orders of magnitude slower than application needs
- No scale-out knobs for short duration bursts
- Over-provisioning is too costly for short term performance demands
- No means to scale down cluster resources with load
- No mechanisms to dynamically boost performance for subsets of data

- Deeply coupled and closely integrated monolithic stack
- Adding or removing storage nodes rebalances data in cluster
  - This is by design! But increases background cluster traffic
- Background tasks like scrubbing, rebalancing

  → global impact on foreground performance

Can Decoupling Enable SDS Agility?

To provide agility, we need SDS mechanisms to,

- Dynamically add and remove resources based on load
- Offer performance boosts for subsets of data, regardless of its location in the cluster
- Not trigger data rebalance when resources are added or removed in response to load

In the rest of this talk,

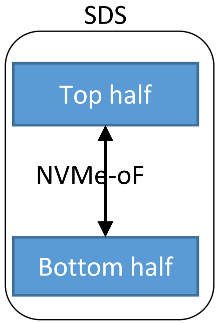
We examine if Decoupling the SDS architecture can help enable such mechanisms

Decoupling Revisited

Decouple SDS architecture into control ("top-half") and data-plane ("bottom-half") components

Top half: performs cluster-wide operations

- Chooses data placement destination
- Manages replicas and erasure coded chunks
- Monitors data integrity
- Performs failure recovery

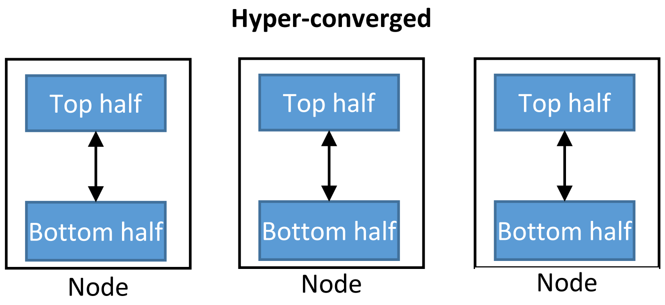


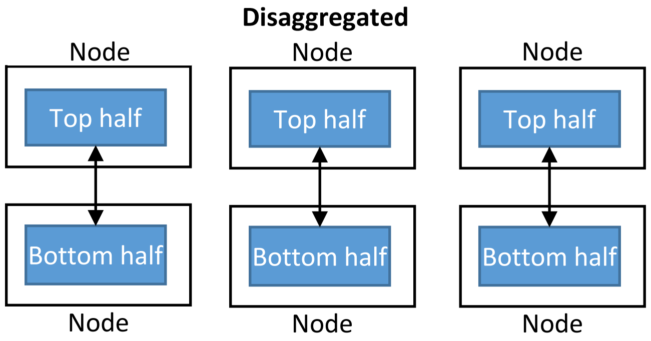
Bottom half: actually stores data and metadata

- responsible for durability
- provides persistence
- manages block layout
- Provides object semantics
- Supports transactions

Standards-based NVMe-oF to communicate between top-half and bottom-half components

Applies to hyper-converged as well as disaggregated deployments

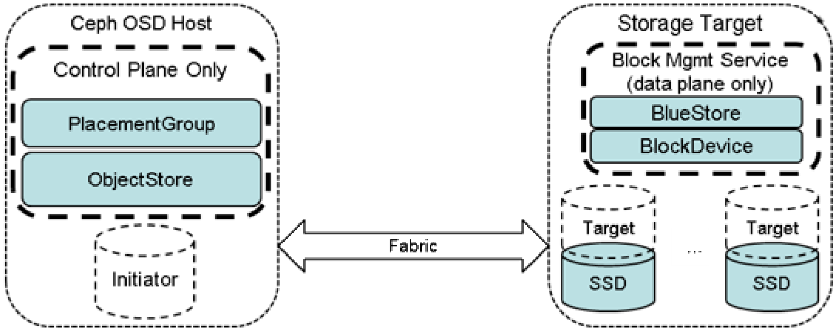




Containers for Decoupled SDS Components

- Deploy “top-half” and “bottom-half” in containers
  - Enable quick bring-up/bring-down of desired SDS components
- Independently scale-out top and bottom-halves
  - Clone top/bottom halves. Can have multiple tops for one bottom or vice-versa!
  - Add or remove clones where needed based on load
- Leverage NVMe-oF between top and bottom halves
  - Enable communication even when SDS components are on different nodes

Decoupled Ceph PoC [SDC'18]



- Ceph Luminous + SPDK v19.07
- New Ceph ObjectStore backend
  - SPDK NVMe-oF Initiator
  - ObjectStore APIs over NVMe-oF
- SPDK RDMA transport for NVMe-oF
- New SPDK bdev
  - SPDK NVMe-oF target
  - instantiates standalone Ceph BlueStore
  - maps requests to Ceph BlueStore

Challenges in managing agile SDS containers

- Rebalance challenge:

  How to avoid data rebalance as clones dynamically come and go?

- Location challenge:

  How to find object location in the presence of dynamic clones?

- Cache consistency challenge:

  How to keep cached data current, in the face of updates?

A Novel Approach for SDS Agility

Clone OSD top-half

- Need: Place data close to where needed, scale-out OSD read throughput
- Does not contribute to object placement → not part of CRUSH
- Functions as a cache node → does not require durable backend
- Fundamentally different from tiering → not a cache tier pool

Introduce *clonemap* as part of the cluster metadata

- Clones are tracked as part a *clone set* for a given OSD in *clonemap*
- Separate from placement SLA related metadata (e.g., CRUSH)

Clone OSD bottom-half

- Need: I/O bottlenecks from write bursts
- Completely transparent to every entity in cluster except fronting OSD top half
- OSD top-half independently creates/removes clones based on load

Decoupling enables adding SDS clones without data rebalance
Does not cause any object migration across failure domains

Scale-out to meet client's I/O demand

How to teach clients to find a clone OSD?

Ceph uses CRUSH

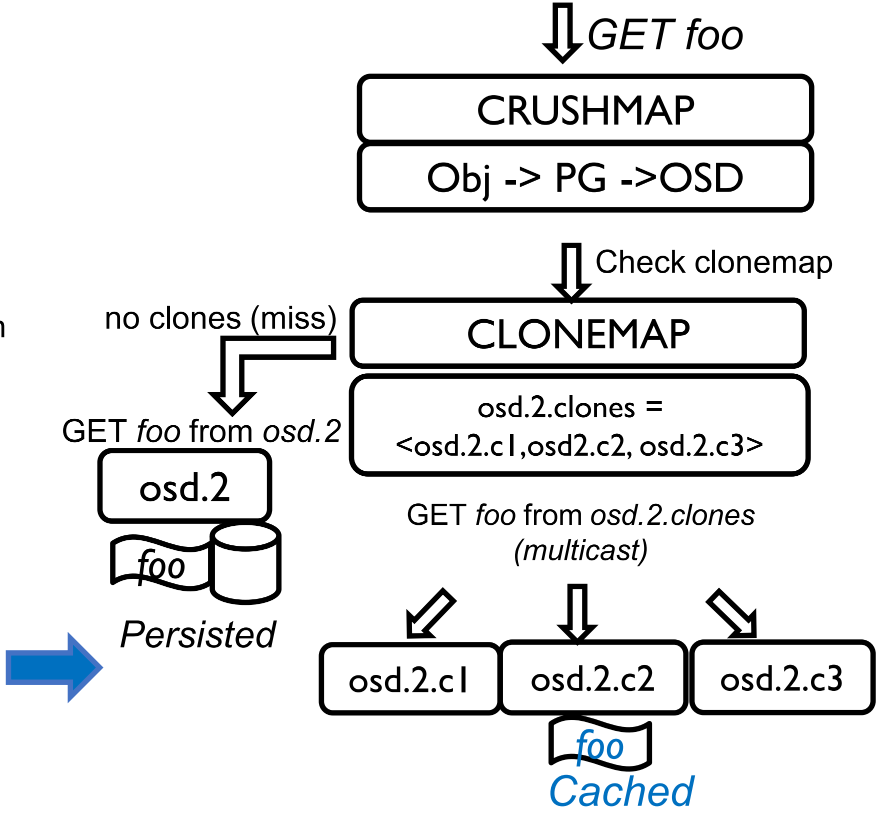
- Object => PG => OSD
- Updated when adding/removing OSD

An Agility Driven Mechanism? Multicast-based Approach

- Per OSD per multicast group
- Tracked by the metadata ***clonemap***
- Add/remove clone => add/remove to multicast group

Works Seamlessly with Existing Cluster, e.g., in Ceph

- *foo => PG => OSD => osd.2*
- Check if *osd.2.clones* is empty from *clonemap*
    - |***osd.2.clones***| != 0 ***=> Cache Hit => osd.2.c2***
    - |***osd.2.clones***| == 0 ***=> Cache Miss => osd.2***



Enables scaling out of client read throughput from individual OSD

Scale-out to address storage I/O bottlenecks

How does top-half find the correct bottom-half clone?

Assumptions

- Clones can come and go as needed.
- Clones expected to exist for short duration
- Majority of objects are located on original bottom half

Handling Writes

- Add entry into bloom filter to track objects stored on clones
- In round-robin fashion, pick one of the clones and write

Handling Reads

- Query Bloom Filter:

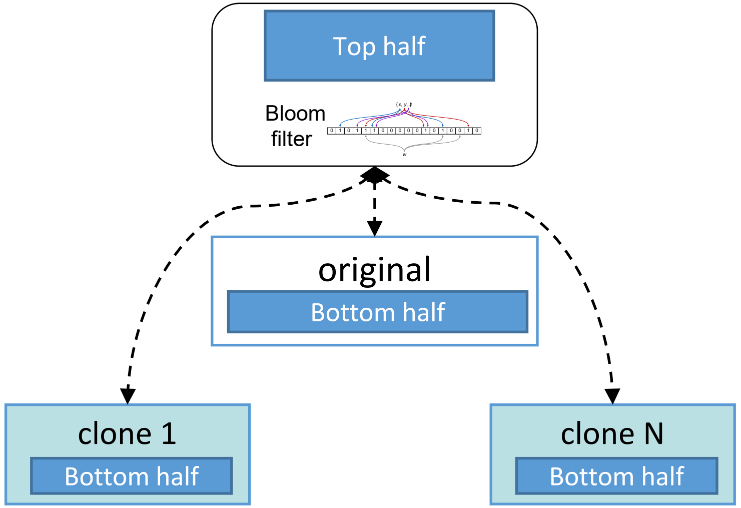
  if not found on clone(likely), send read request to original target

  if found on clone(unlikely), multicast read request to all clones of this target

  false positive, send read request to original target

Handling load reduction

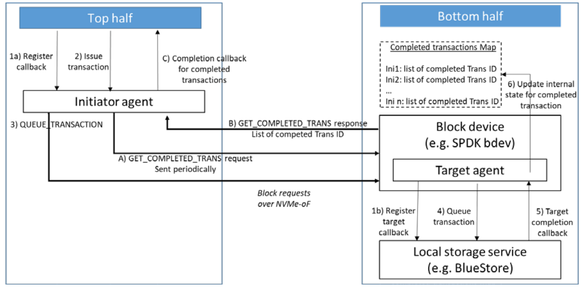
- Copy back data to original target as load reduces
- Remove clones after data copied off, reset bloom filter



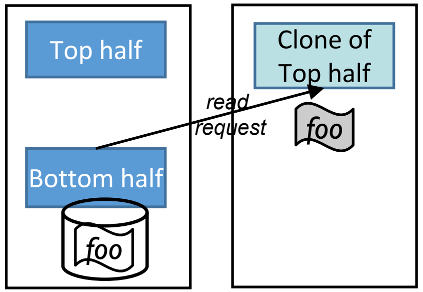
Enables scaling out of write bandwidth for individual OSD

Remote Transaction Support

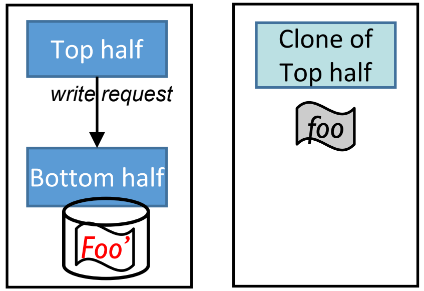
- Decoupled Ceph PoC implements remote asynchronous transactions
  - top half can issue transaction requests to remote bottom half
  - asynchronous transaction completion notification from bottom half
  - crucial for high throughput
- **Cohort notification** concept
  - Transaction can be submitted “on behalf” of other OSDs
  - Request message includes list of OSDs to notify
  - Original goal: Eliminate data hop in a disaggregated scenario


Top half
1a) Register callback
2) Issue transaction
C) Completion callback for completed transactions
Initiator agent
3) QUEUE_TRANSACTION
B) GET_COMPLETED_TRANS response List of competed Trans ID
A) GET_COMPLETED_TRANS request Sent periodically
Block requests over NVMe-oF
Bottom half
Completed transactions Map
Ini1: list of completed Trans ID
Ini2: list of completed Trans ID
...
Ini n: list of completed Trans ID
6) Update internal state for completed transaction
Block device (e.g. SPDK bdev)
Target agent
1b) Register target callback
4) Queue transaction
5) Target completion callback
Local storage service (e.g. BlueStore)

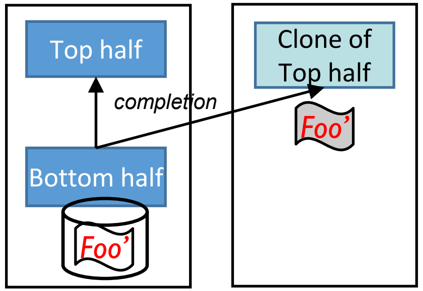
Maintaining consistency



Object cached on clone



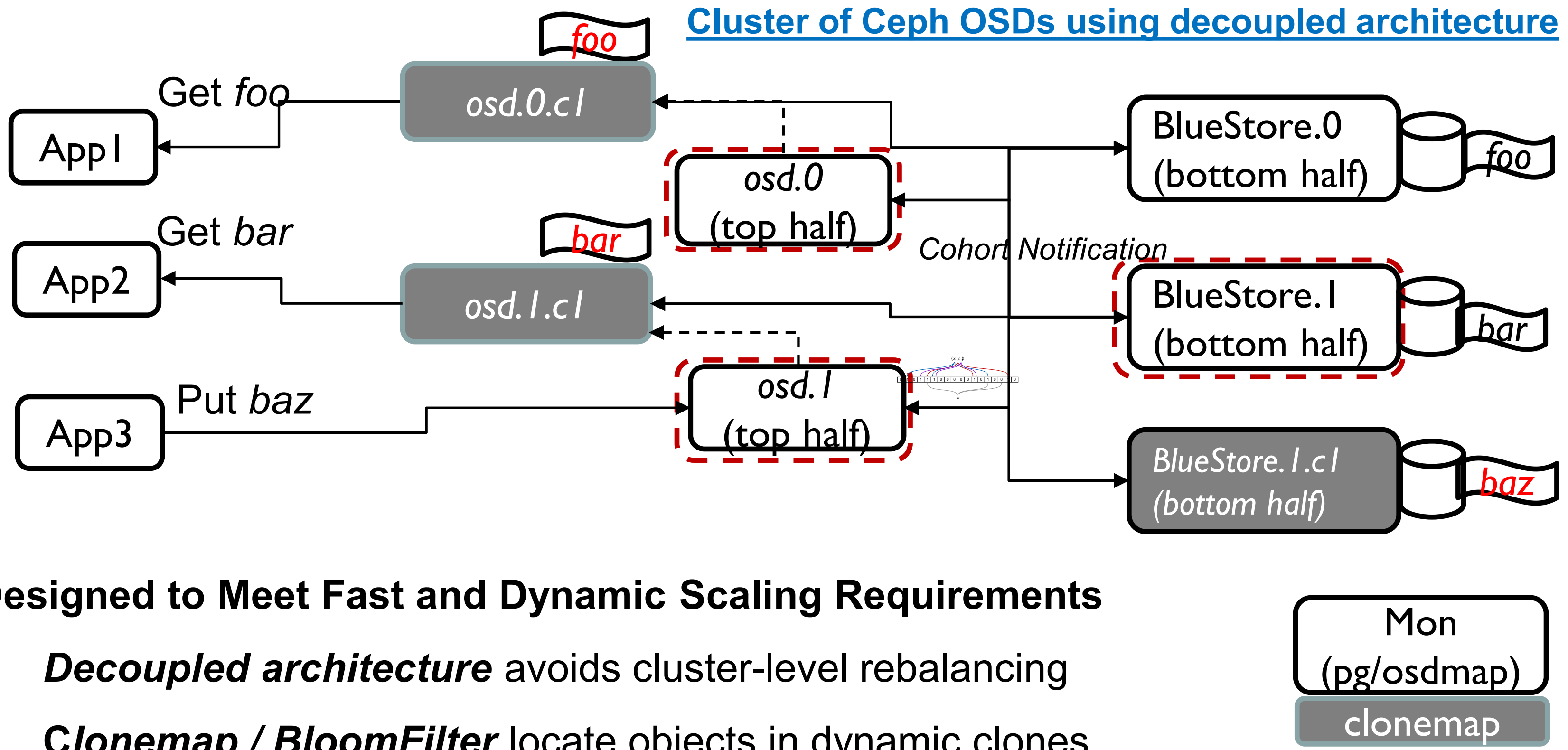
Object write transaction request



Completion sent to top + clone

Cohort notification enables clones with cached data to be informed of updates

Putting Everything Together: An Illustrative Example Using Ceph



Designed to Meet Fast and Dynamic Scaling Requirements

- ***Decoupled architecture*** avoids cluster-level rebalancing
- ***Clonemap / BloomFilter*** locate objects in dynamic clones
- ***Cohort notification*** updates cached data

Advantages of the Proposed Approach

Unique Advantages

- Scale up and down with minimal overhead
- Clone OSD can be started with an intelligently warmed-up cache
- Clone OSD can directly pull (or batch) objects from targets or other clones
- Scale write throughput to handle bursts by cloning target

Also Note

- Updating *clonemap* is likely to be more frequent than others, e.g. osdmap
- Target clones are only visible to corresponding top-half
- There are no side effects
  - No rebalancing or impact to on-going data placement
  - Behaves as a cache-miss even if clonemap is not recent
  - Clone is not removed till clonemap is updated to remove the clone

Discussion: Existing Containerized SDS solutions

- **Prior Art for SDS and Containers**
  - Minio[1], rook.io[2], Ceph OSD in container [3]
- **Different Goals for Containerized SDS**
  - Entire monolithic SDS stack is containerized

    => cannot scale bottleneck components independently
  - Focus is on ease of management

    => cannot help dynamic I/O demands of clients

[1] https://min.io/
[2] https://rook.io/
[3] L. Baumann, S. B. Abraxas, L. Militano, and T. M. Bohnert, “Monitoring Resilience in a Rook-managed Containerized Cloud Storage System,” IEEE European Conf. on Networks and Comm. (EuCNC), 2019.

Discussion: Existing Caching Solutions

Client-Side Cache: web cache (e.g., Nginx) or Ceph RBD/RGW caching

- Ineffective for clients with high mobility or pipelined microservices
- Additional support required on client platforms (e.g. caching software)
- Unable to help at a per overloaded OSD level

Storage Node Cache: typically block cache e.g. bcache, flashcache

- Per block device, thus not client oriented
- Additional support required on all storage nodes

Memory-based Caching Frameworks: e.g. memcacheD, Redis etc.

- Requires managing separate cluster

Tiering: data landed to tier first, then moved to backend later

- Complicated, and requires additional Hardware/Software
  - Cache tiering in Ceph is particularly inefficient
- Does not meet agility needs

Summary

Emerging workloads demand agility from storage

Decouple SDS architecture to make them agile!

- Independently clone control and data plane via containers
- Scale based on load with no cluster-level rebalancing
  - ✓ Transparently clone bottom-half under write pressure
  - ✓ Clone top-half to reduce latency

Next Steps

- Complete PoC to support *clone osd and clonemap* using Ceph
- Evaluate scaling capability with dynamic clients and pipelined microservices

<u>We make a case for a Decoupled SDS architecture</u>

Reduce datacenter tax from disaggregation [SDC2018]

- BW consumption reduction: cluster network (99%) / total network (35%)
- Up to 35% end-end latency reduction (25% average)

Enable scale-out of multiple heterogenous storage services [SDC 2019]

<u>Make SDS agile</u> [this talk]

Ease integration of next-gen storage and mixed media types [<u>SDC2020: Mortimer talk</u>]

Please take a moment to rate this session.

Your feedback matters to us.