SDC 19

September 23-26, 2019
Santa Clara, CA

intel

A Crash-consistent Client-side Cache for Ceph

Lisa Li, Tushar Gohad
Intel

Agenda

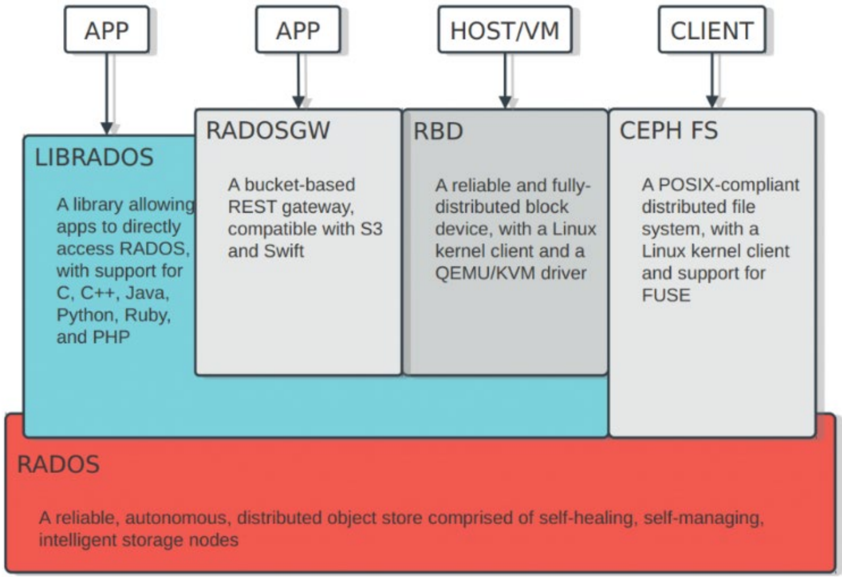
- Motivation and Background
- Design and Implementation
- Performance Evaluation
- Upstream Status and Future Work

Background and Motivation

Ceph Introduction

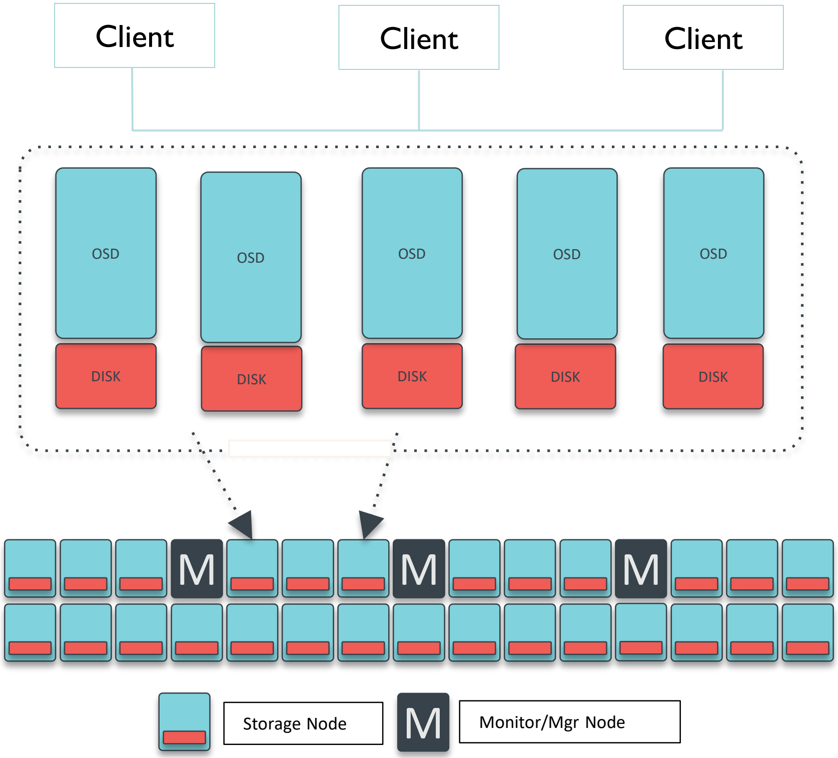
- Ceph is a unified, distributed storage system designed for excellent performance, reliability and scalability

- Object Store (RADOSGW)
  - A bucket based REST gateway
  - Compatible with S3 and swift
- Block device service (RBD)
  - Block device
  - Kernel client and FUSE
- File System (CEPH FS)
  - POSIX-compliant distributed file system
  - Kernel client and FUSE

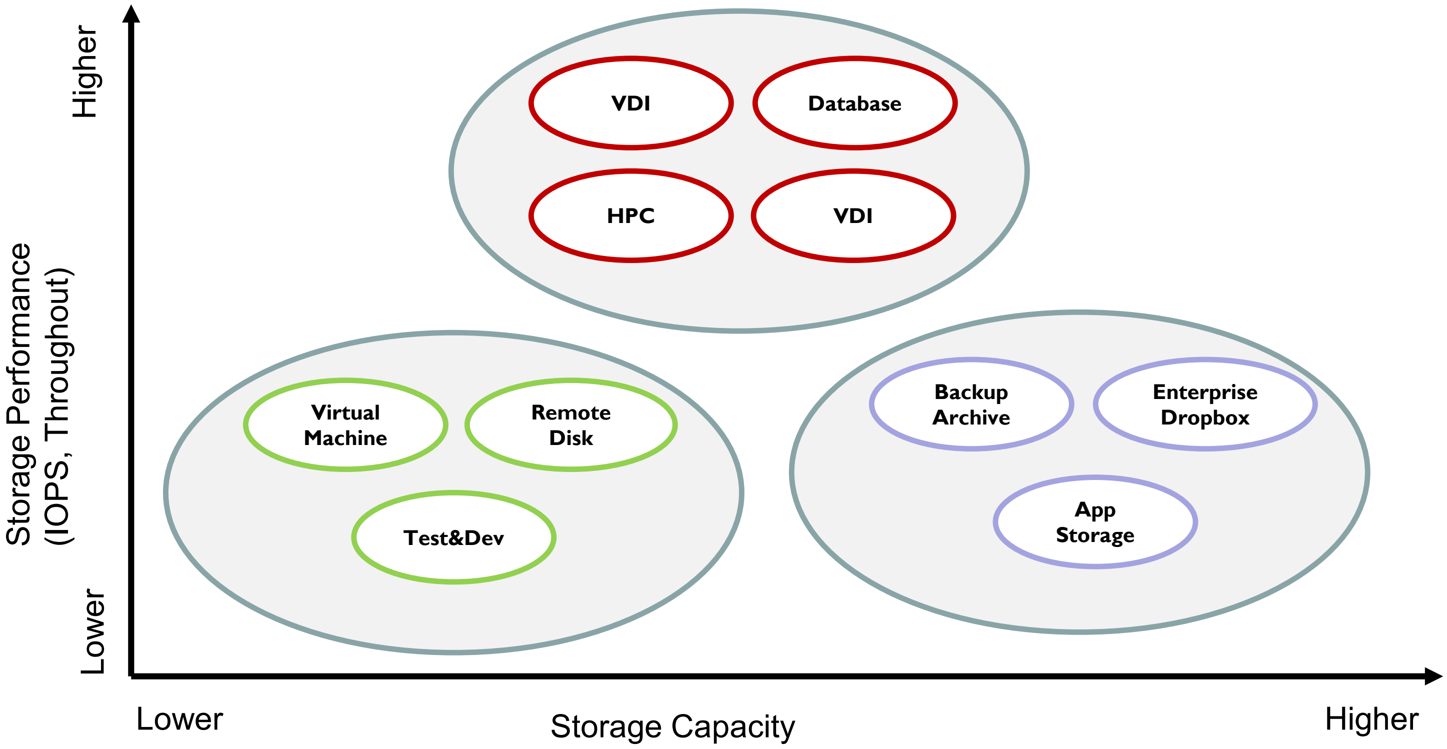


Ceph Cluster

- Ceph clients
  - User space/Kernel driver
- Peer to Peer via Network
  - Direct access to storage
  - No centralized metadata node
- Storage Nodes
  - Data distributed and replicated across nodes

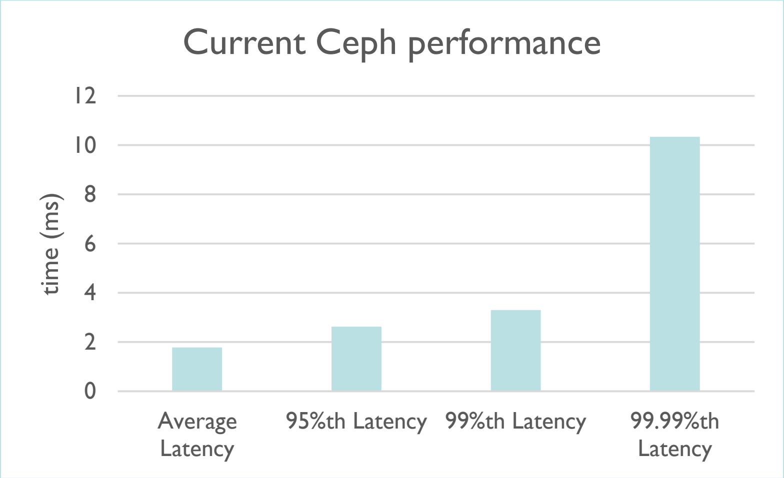


Ceph Workloads



Latency-sensitive IO loads

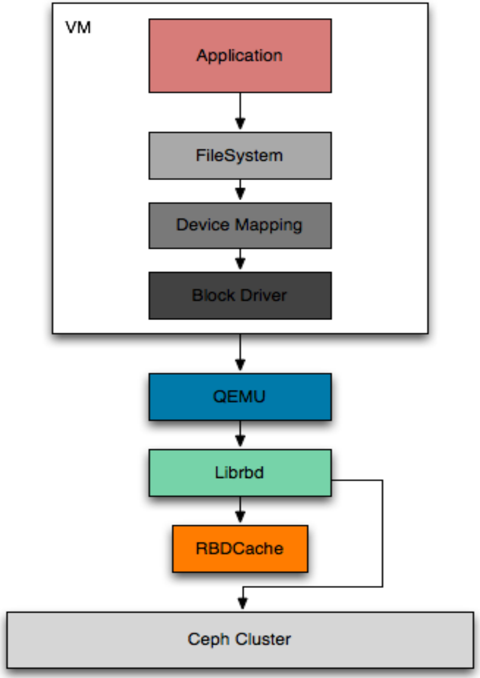
Test	Average Latency	95%th Latency	99%th Latency	99.99%th Latency
4k randwrite	1.775	2.625	3.294	10.342



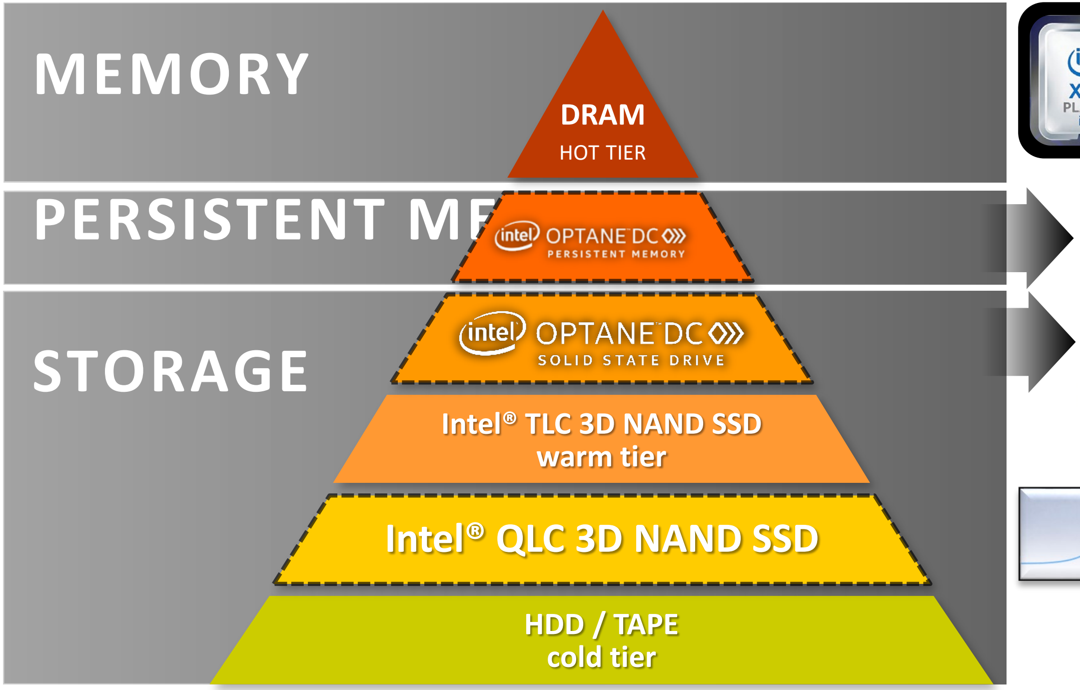
- IOPS requirement may be not high, Tail latency is key
- Database etc
- High performance target for Ceph: caching one way to improve tail latency while Crimson OSD project takes shape

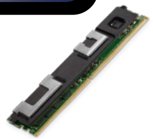
Block Caching in Ceph

- Ceph supports RAM-based block cache today
- No persistent block cache
- Small capacity
- Ongoing effort to support persistent block cache in read and writeback mode



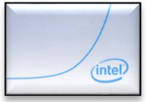
Latest Generation Hardware for Ceph





Intel® CPUs with up to 56 cores
Intel® Optane™ DC PM
DDR4 DIMM form factor
Cost-effective memory expansion
Lowest latency
Uses: flexible memory for AI, CDN, DBs, fast restart

Intel® Optane™ DC SSD (3D XPoint)
High write IOPs
High endurance
Low & consistent latency
Uses: Metadata, Tier, Cache

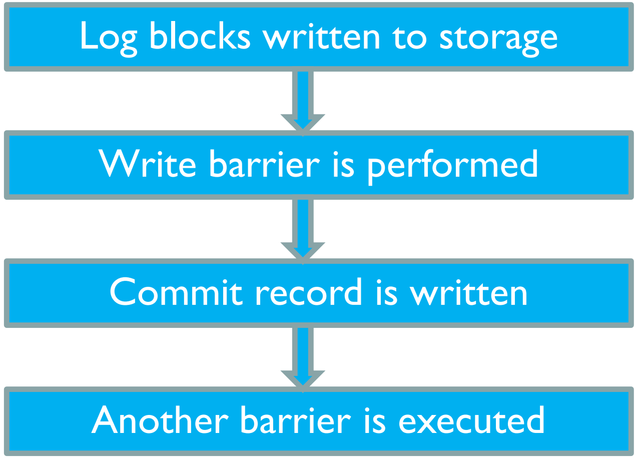


Intel® QLC SSD (3D-NAND)
Cost & capacity optimized
Scale up to 32TB per SSD
High density, small footprint servers
Read-oriented workloads

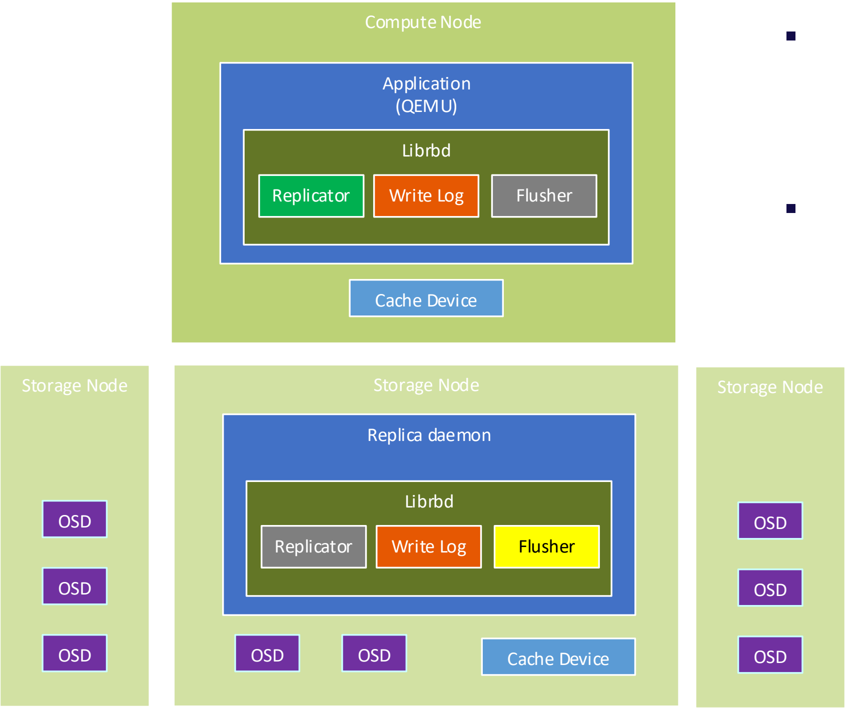
Design and Implementation

Ordered, Crash-consistent WB

- Crash Consistent Caching
  - Journaled
  - Snapshot
- Ordered Writeback
  - Write-barriers to enforce write ordering
- RBD image consistent despite compute node crash

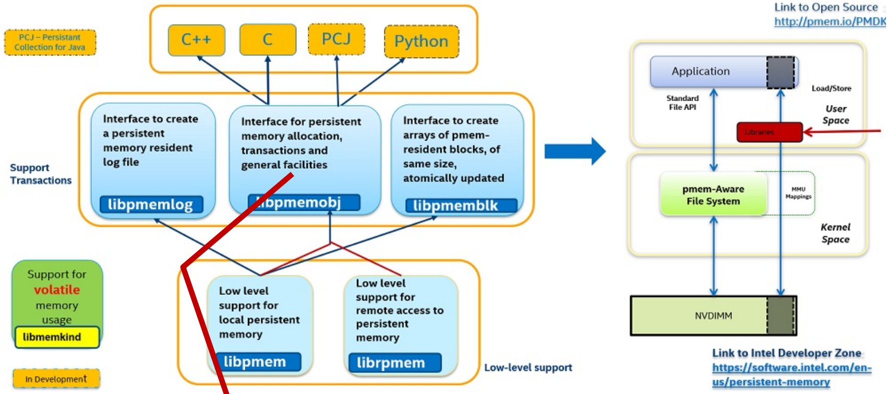


Architecture



- Local write log on cache device is replicated
  - If cache device fails, so does client node
  - Avoids remote log read & local resilver
- Replica write log is in a Ceph storage node
  - Fully monitored, unlike client
  - Can be replaced on the fly, and resilvered from the local log
  - We'll flush from here
    - Flushing must complete if client dies
    - Closer to OSDs
  - Multiple replicas possible

Persistent Memory* as Cache Device



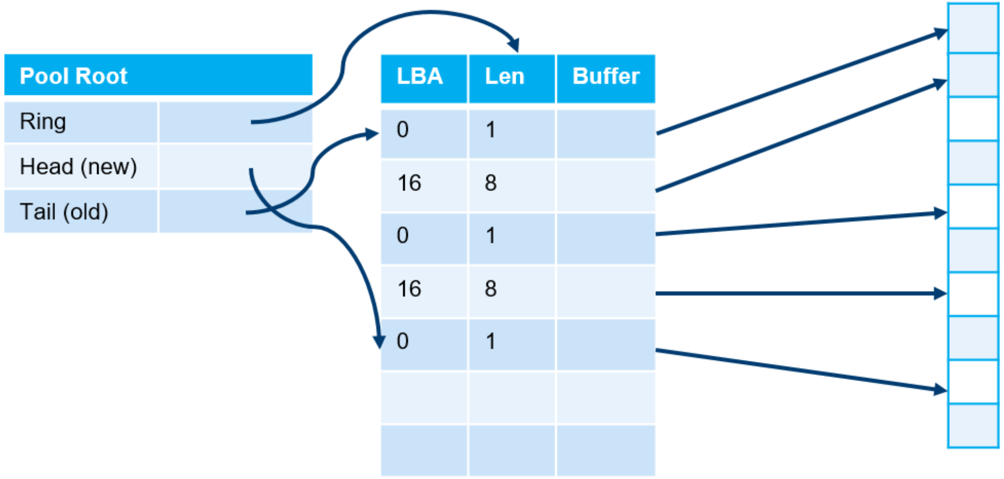
Write back cache uses this

- **PMDK** is a growing collection of libraries which build on DAX feature and allows application direct load/store access to persistent memory
- **libpmemobj**: provides a transactional object store for cache metadata

* Our work uses the Intel® Optane^{TM} DC Persistent Memory Module

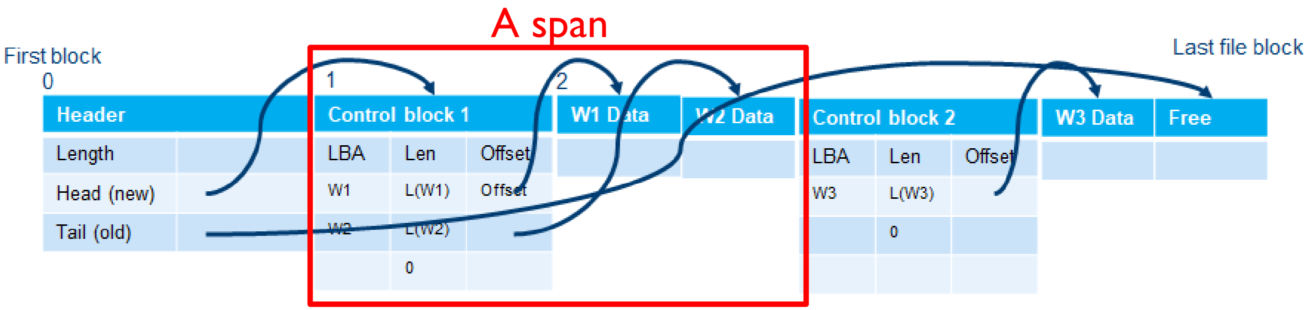
Persistent Memory as Cache Device

- Operated as memory
- Superblock + journal array + data buffer
- Both superblock and journal array are fixed during initialization, and data buffer managed by libpmemobj
- Libpmemobj provides transactional metadata store



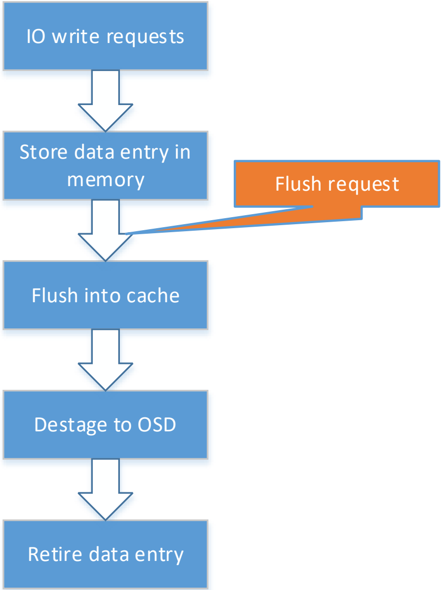
NVMe SSD as Cache Device

- Operations are block-based
- Superblock + ring buffer of spans
- Each span includes a control block and a number of data blocks
- Flush, Reclaim on a span unit



Generic Workflow

- The overall process:
  - Receive requests from librbd
  - Save data in memory
  - Flush data to cache when a flush request is triggered
  - De-stage data to storage cluster in background
  - Retire data in cache

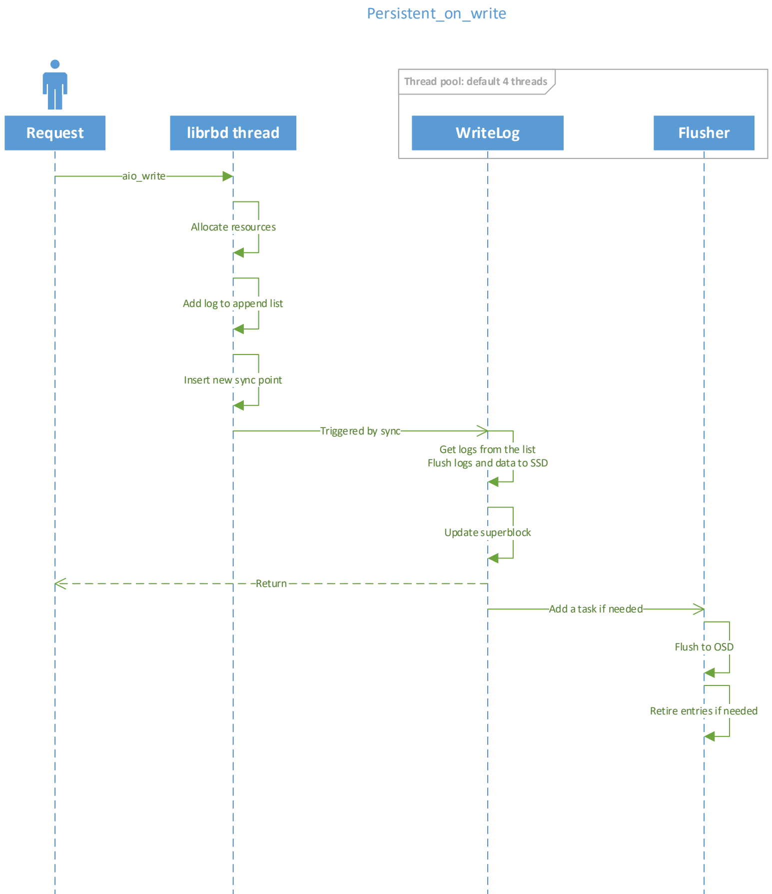


Persistence control

- **Persist on write**
  - Write ACKed after it has been persisted to cache device(s)
- **Persist on flush**
  - Write ACKed after data is in RAM buffer
  - When Flush request is issued (by client or cache), data is persistent to cache device(s)
    - All prior completed writes are guaranteed persistent before ACKing
    - Linux write barrier (client), Internal flush requests (cache)
    - Every flush is mapped to a sync point in the journal array
  - We'll defer reads of LBAs with in flight writes until those writes completed
- Policy is selectable
  - auto-detect possible, set writethrough_until_flush as true

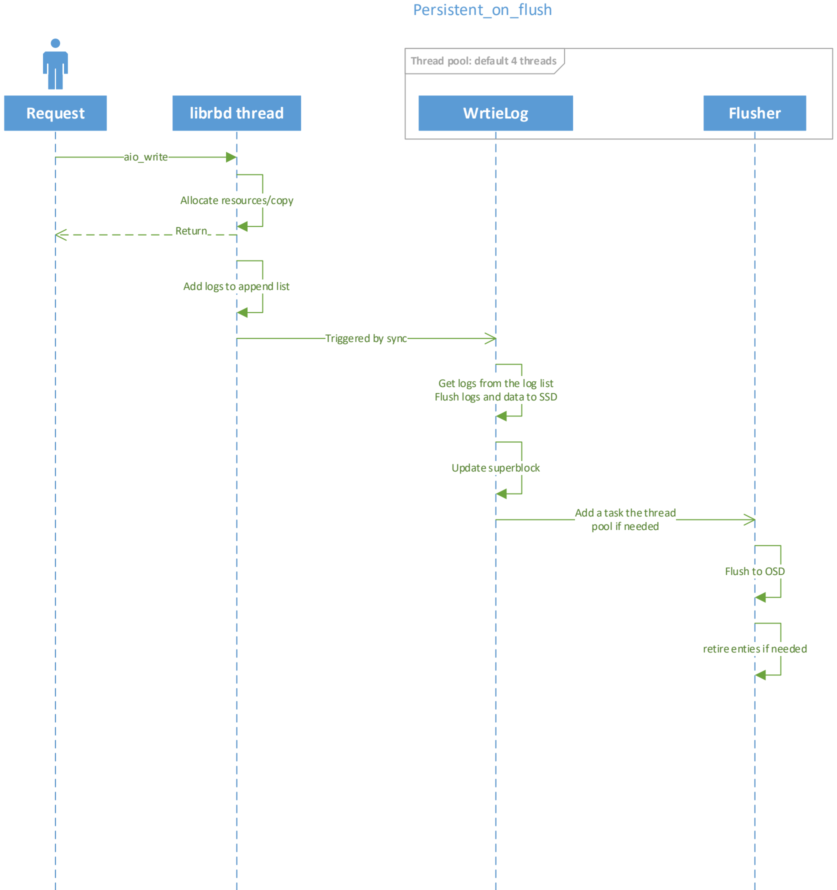
Persist-on-Write

- When a write request is handled, it creates a log entry in RAM and its user buffers are copied in RAM.
- The log entry is added to a log append queue.
- The logs and corresponding data are sent to cache device.
- Update tail in the superblock.
- Once write completes on cache device, it returns to users that a write request is completed.



Persist-on-Flush

- When a write request is handled, it creates a log entry in RAM and its data buffers are copied in RAM.
- The log entry is added to a log append queue.
- It returns to the user that the request is completed.
- When users send out a flush request or the dirty logs/data exceeds limitation, the logs and corresponding data in the append queue will be written to cache device.
- Update tail in the superblock.



Flush and Reclaim

- Flush log entries on sync point boundary
  - Flusher can flush in parrallel
  - Once flushed into OSD, the log entry is marked as completed
  - WIP: Write-coalescing (merge write entries between sync points)
- Reclaim on a span unit
  - All the log entries in a span are completed, the span can be reclaimed.
  - Update superblock and the span is reclaimed.

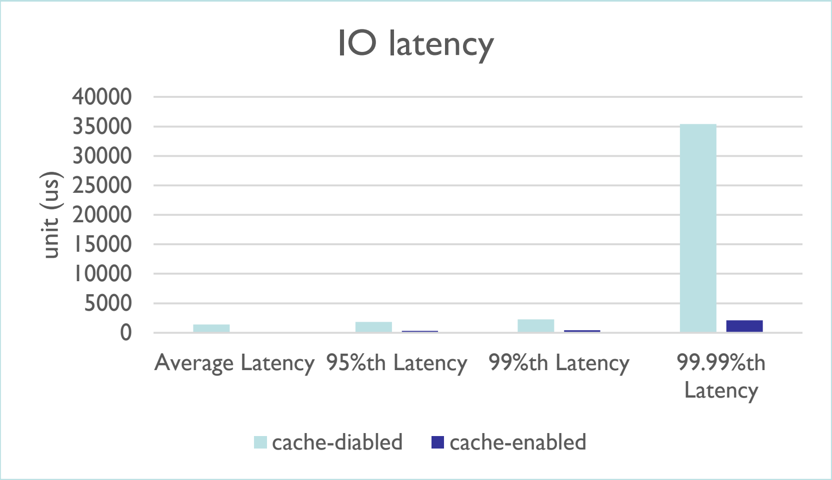
Performance

FIO, Single RBD Image

94% better tail wr latency

- **Client**
  - Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
  - Intel® Optane™ SSD DC P4800X 375G
- **Storage Node**
  - Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
  - One Intel® SSD DC P3700 Series 400G
- **IO loads**
  - 1x 10G rbd image
  - Cache_size = 1G
  - FIO + librbd
  - 4K random writes
  - RBD IOPS Limit = 4000
  - Send sync request every 32 IOs.
  - ramptime = 10 min, runtime=10 min

	Average Latency	95%th Latency	99%th Latency	**99.99%th Latency**
Cache-disabled	1382	1811	2245	**35390**
Cache-enabled	89	326	424	**2089**

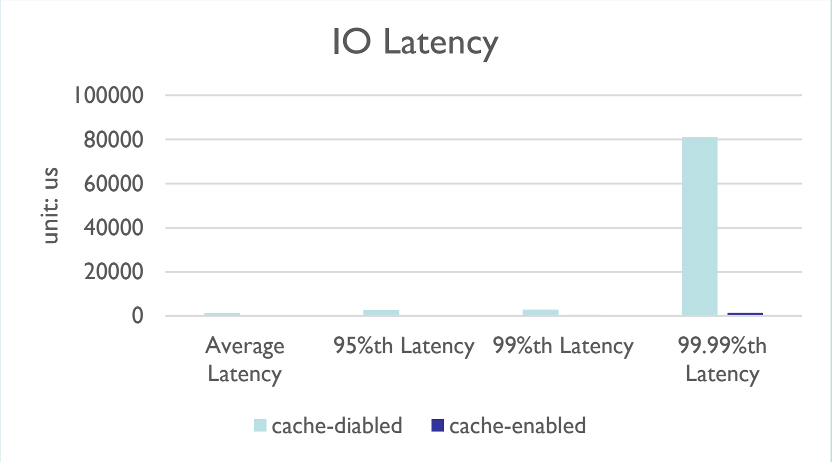


FIO, Multiple RBD Images

98% better tail wr latency

- **Client**
  - Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
  - Intel® Optane™ SSD DC P4800X 375G
- **Storage Node**
  - Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
  - One Intel® SSD DC P3700 Series 400G
- **IO loads**
  - 5x 10G rbd image
  - Cache_size = 1G
  - FIO + librbd
  - 4K random writes
  - RBD IOPS Limit = 4000
  - Send sync request every 32 IOs.
  - ramptime = 10 min, runtime=10 min

cache	Average Latency	95%th Latency	99%th Latency	**99.99%th Latency**
cache-diabled	1320	2540	2868	**81265**
cache-enabled	83	285	429	**1450**



Future Work

Upstream Status + Future Work

- Upstream Status
  - Under review https://github.com/ceph/ceph/pull/29078
  - SSD part is under development. Merget target 2020
- Replication
  - PM Replication over RDMA support in PMDK (WIP)
- CAS/OCF (Cache Acceleration Software)
  - Open-sourced: https://open-cas.github.io
  - OCF plugin for librbd in POC stage

Thank you!