Best Practices for Increasing Ceph Performance with SSD

Jian Zhang Jian.zhang@intel.com

Jiangang Duan Jiangang.duan@intel.com

Agenda

- Introduction
- Filestore performance on All Flash Array
- KeyValueStore performance on All Flash Array
- Ceph performance with Flash cache and Cache tiering on SSD
- Summary & next steps

Agenda

- Introduction
- Filestore performance on All Flash Array
- KeyValueStore performance on All Flash Array
- Ceph performance with Flash cache and Cache tiering on SSD
- Summary & next steps

Introduction

- Intel Cloud and BigData Engineering Team
  - Working with the community to optimize Ceph on Intel platforms
  - Enhance Ceph for enterprise readiness – path finding Ceph optimization on SSD
  - Deliver better tools for management, benchmarking, tuning - VSM, COSBench, CeTune
  - Working with China partners to build Ceph based solution

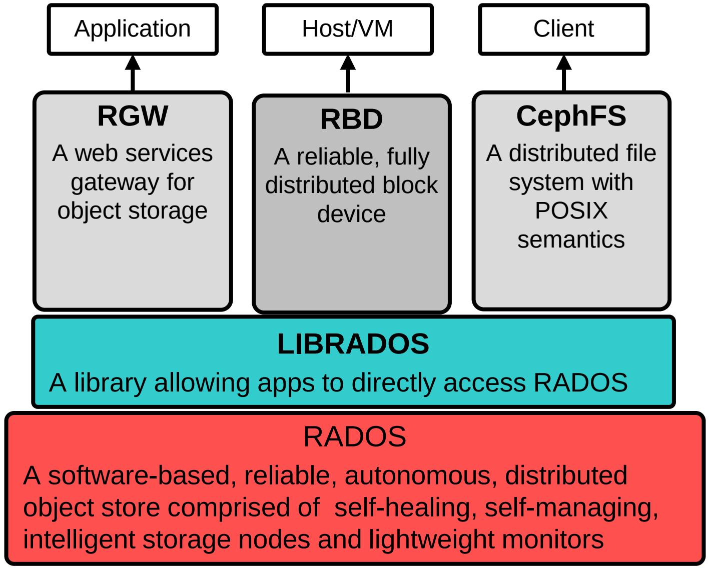
- Acknowledgement
  - This is a team work: Credits to Chendi Xue, Xiaoxi Chen, Xinxin Shu, Zhiqiang Wang etc.

Executive summary

- Providing high performance AWS EBS like service is common demands in public and private clouds
- Ceph is one of the most popular block storage backends for OpenStack clouds
- Ceph has good performance on traditional hard drives, however there is still a big gap on all flash setups
- Ceph needs more tunings and optimizations on all flash array

Ceph Introduction

- Ceph is an open-source, massively scalable, software-defined storage system which provides object, block and file system storage in a single platform. It runs on commodity hardware—saving you costs, giving you flexibility
- Object Store (RADOSGW)
  - A bucket based REST gateway
  - Compatible with S3 and swift
- File System (CEPH FS)
  - A POSIX-compliant distributed file system
  - Kernel client and FUSE
- Block device service (RBD)
  - OpenStack* native support
  - Kernel client and QEMU/KVM driver

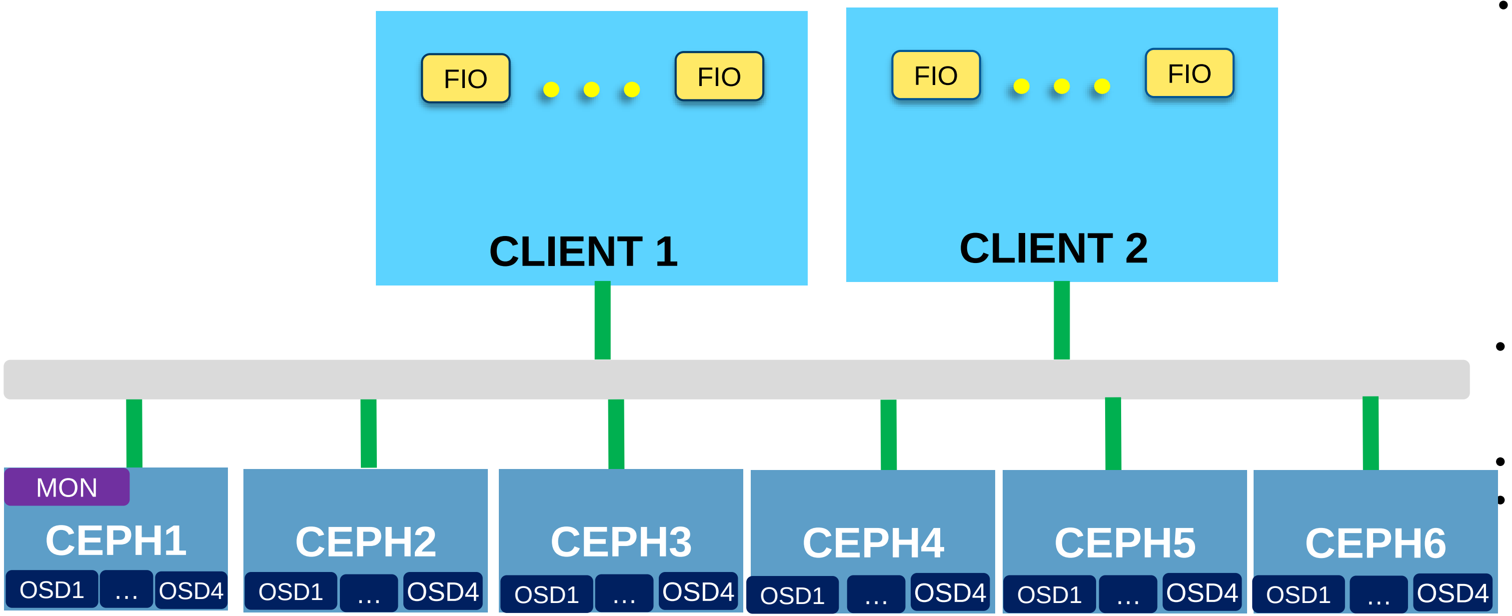


Agenda

- Introduction
- Filestore performance on All Flash Array
- KeyValueStore performance on All Flash Array
- Ceph performance with Flash cache and Cache tiering on SSD
- Summary & next steps

Filestore System Configuration

Test Environment



Client Node

- 2 nodes: Intel® Xeon ® CPU E5-2680 v2 @ 2.80GHz, 64GB mem

Storage Node

- 6 node : Intel® Xeon ® CPU E5-2680 v2 @ 2.80GHz
- 64GB memory each node
- Each node has 4x Intel® DC3700 200GB SSD, journal and osd on the same SSD

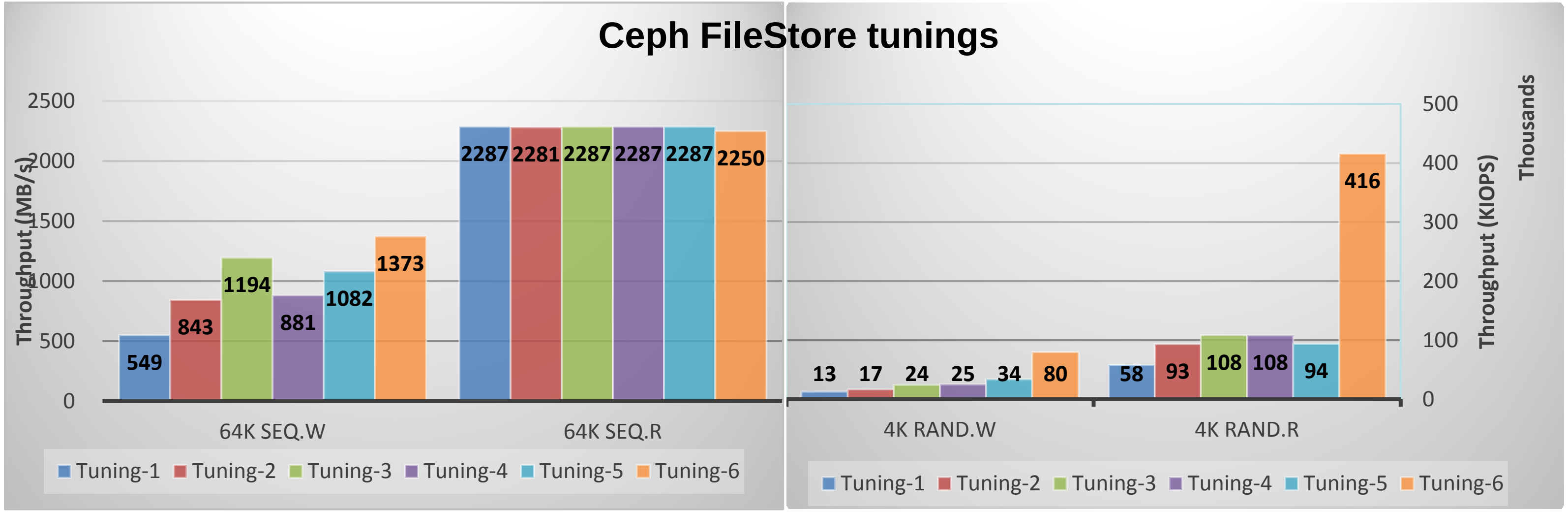
Note: Refer to backup for detailed software tunings

Filestore Testing Methodology

- Storage interface
  - Use ***FIORBD*** as storage interface
- Data preparation
  - Volume is ***pre-allocated*** before testing with seq write
- Benchmark tool
  - Use **fio** (ioengine=libaio, direct=1) to generate 4 IO patterns: ***64K sequential write/read*** for bandwidth, ***4K random write/read*** for IOPS
  - No capping
- Run rules
  - ***Drop OSDs page caches*** (echo "1“ > /proc/sys/vm/drop_caches)
  - ***Duration***: 100 secs for warm up, 300 secs for data collection

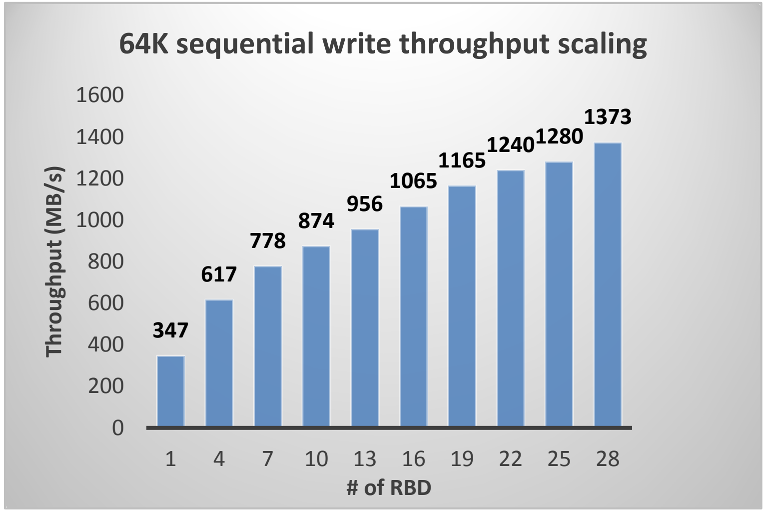
Performance tunings

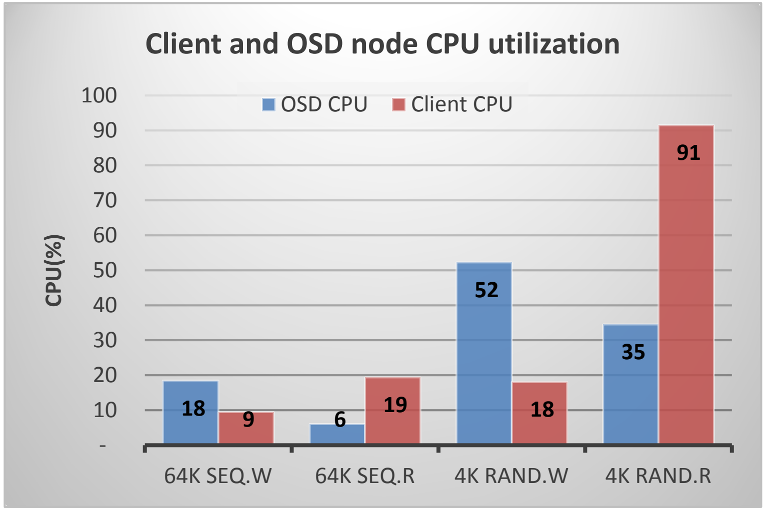
Tunings	Tuning Description
Tuning-1	One OSD on Single SSD
Tuning-2	2 OSDs on single ssd
Tuning-3	T2 + debug = 0
Tuning-4	T3 + 10x throttle
Tuning-5	T4 + disable rbd cache, optracker, tuning fd cache
Tuning-6	T5 +jemalloc



- 2250WB/s for seq_R, 1373MB/s for seq_W, 416K IOPS for random_R and 80K IOPS for random_w
- Ceph tunings improved Filestore performance dramatically
  - **2.5x** for Sequential write **6.3x** for 4K random write, **7.1x** for 4K Random Read

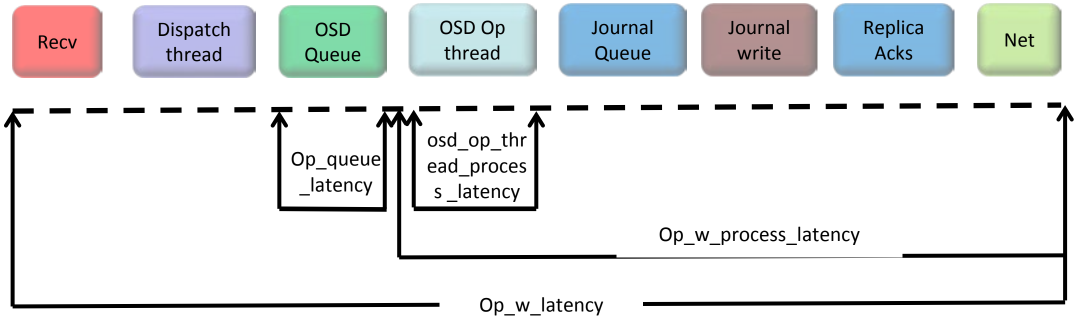
Bottleneck Analysis





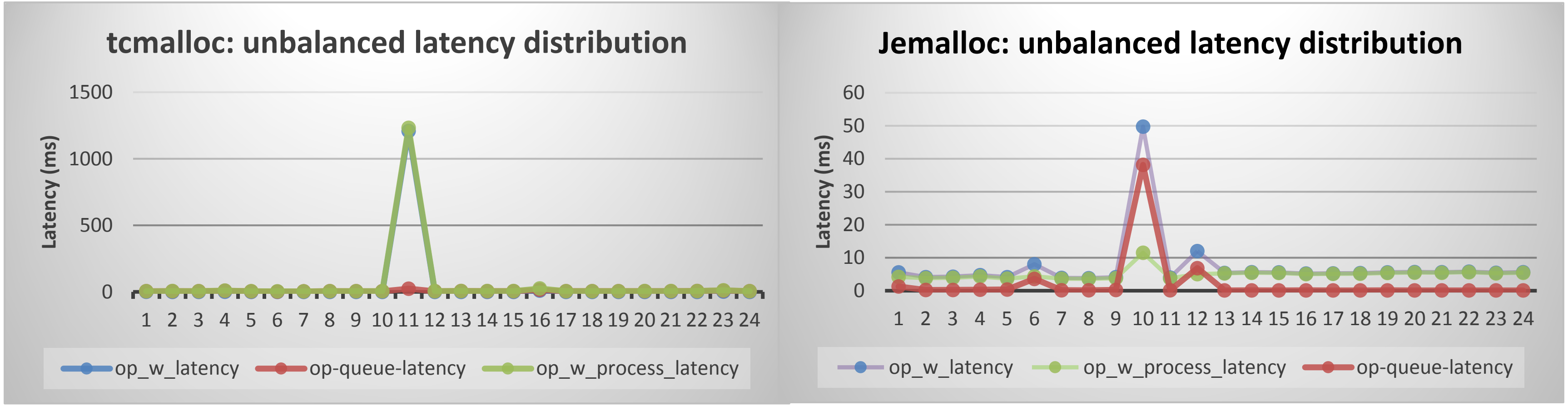
- 64K Sequential read/Write throughput still increase if we increase more # of clients – need more testing
- 4K random read performance is throttled by client CPU
- No clear Hardware bottleneck for random write - Suspected software issue - further analysis in following pages

Latency breakdown methodology



- We use latency breakdown to understand the major overhead
  - Op_w_latency: process latency by the OSD
  - Op_w_process_latency: from dequeue of OSD queue to sending reply to client
  - Op_queue_latency: latency in OSD queue
  - Osd op thread process latency: latency of OSD op thread.

Latency breakdown



- Unbalanced latency across OSDs
  - Jemalloc brings most improvement for 4K random write
  - The op_w_latency unbalance issues was alleviated, but not solved

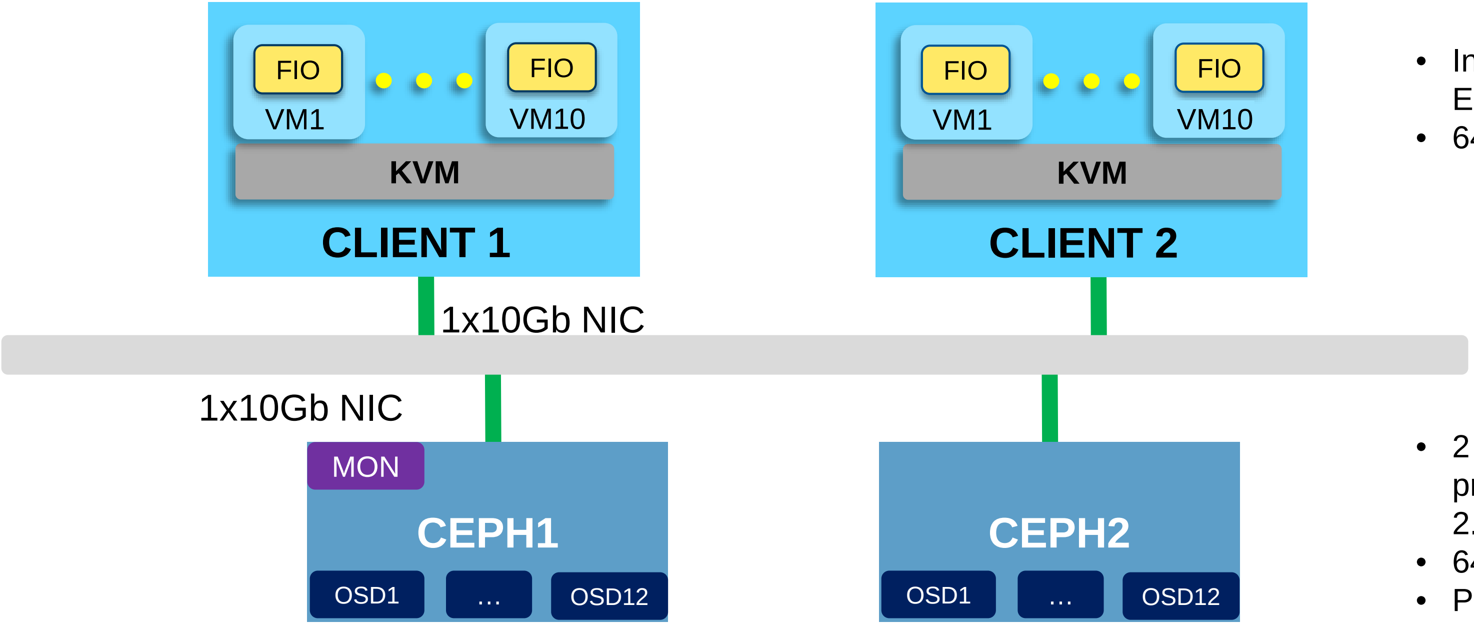
Note: This only showed the OSD latency breakdown - there are other issues on the rest part

Agenda

- Introduction
- Filestore performance on All Flash Array
- KeyValueStore performance on All Flash Array
- Ceph performance with Flash cache and Cache tiering on SSD
- Summary & next steps

KeyValueStore System Configuration

Test Environment



Client Node

- Intel® Xeon ® processor E5-2680 v2 2.80GHz
- 64 GB Memory

Ceph OSD Node

- 2 nodes with Intel® Xeon ® processor E5-2695 2.30GHz,
- 64GB mem each node
- Per Node: 6 x 200GB Intel® SSD DC S3700
- 2 OSD instances on each SSD, with journal on the same SSD

Note: Refer to backup for detailed software Tunings

Software Configuration

Ceph cluster	
OS	Ubuntu 14.04
Kernel	3.16.0
Ceph	0.94

Client host	
OS	Ubuntu 14.04
Kernel	3.13.0

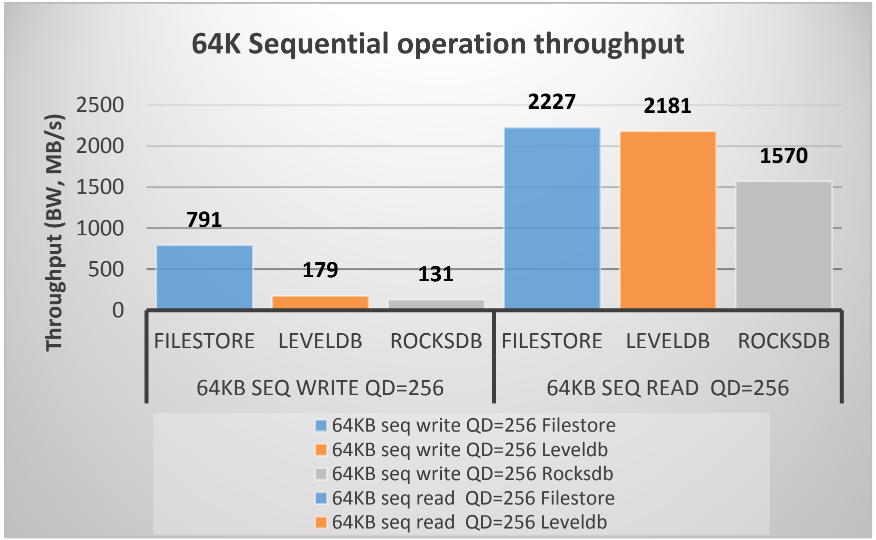
- ***Ceph version is 0.94***
- **XFS** as file system for Data Disk
- replication setting (2 replicas), **1536** pgs

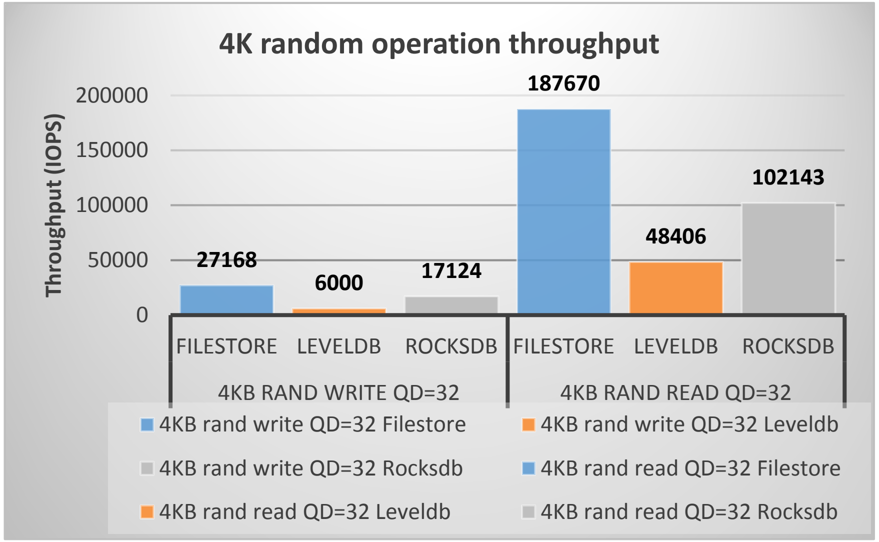
Note: Software Configuration is all the same for the following cases

KeyValueStore Testing Methodology

- Storage interface
  - Use ***FIORBD*** as storage interface
- Data preparation
  - Volume is ***pre-allocated*** before testing with seq write
- Benchmark tool
  - Use **fio** (ioengine=libaio, direct=1) to generate 4 IO patterns: ***64K sequential write/read*** for bandwidth, ***4K random write/read*** for IOPS
  - No capping
- Run rules
  - ***Drop OSDs page caches*** (echo "1“ > /proc/sys/vm/drop_caches)
  - ***Duration***: 100 secs for warm up, 300 secs for data collection

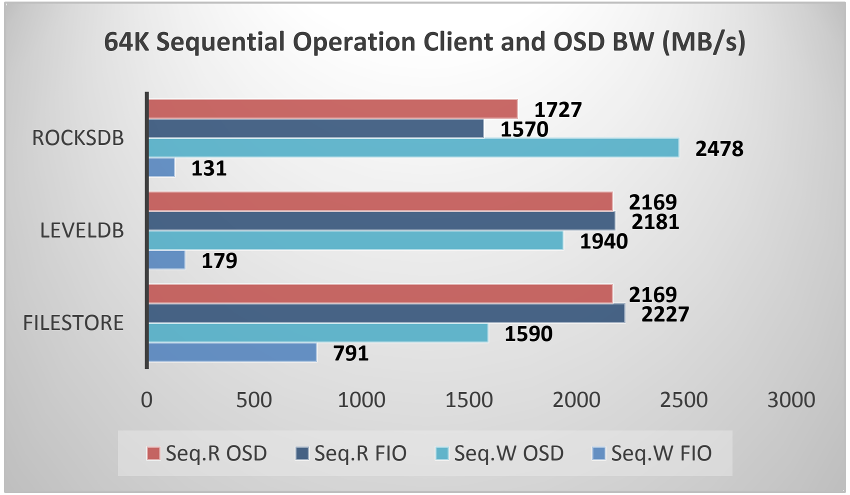
Ceph KeyValueStore performance

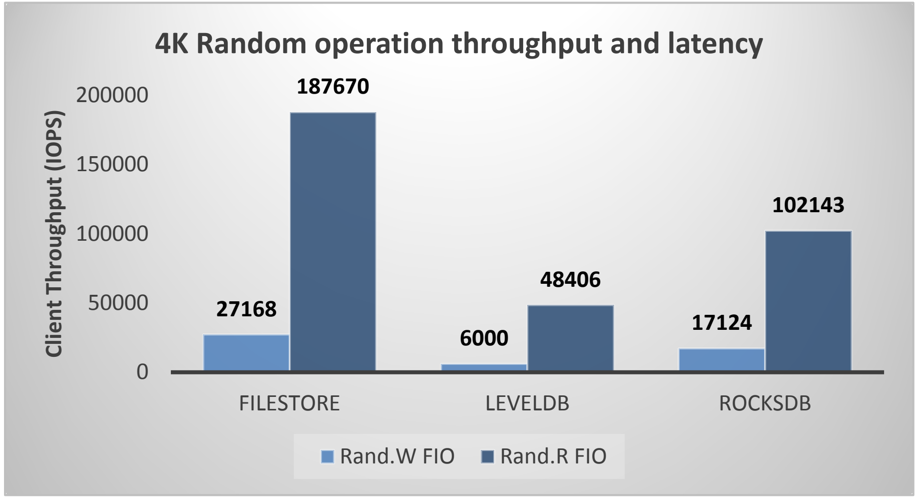




Key/value store performance is far below expectation and much worse compared with Filestore

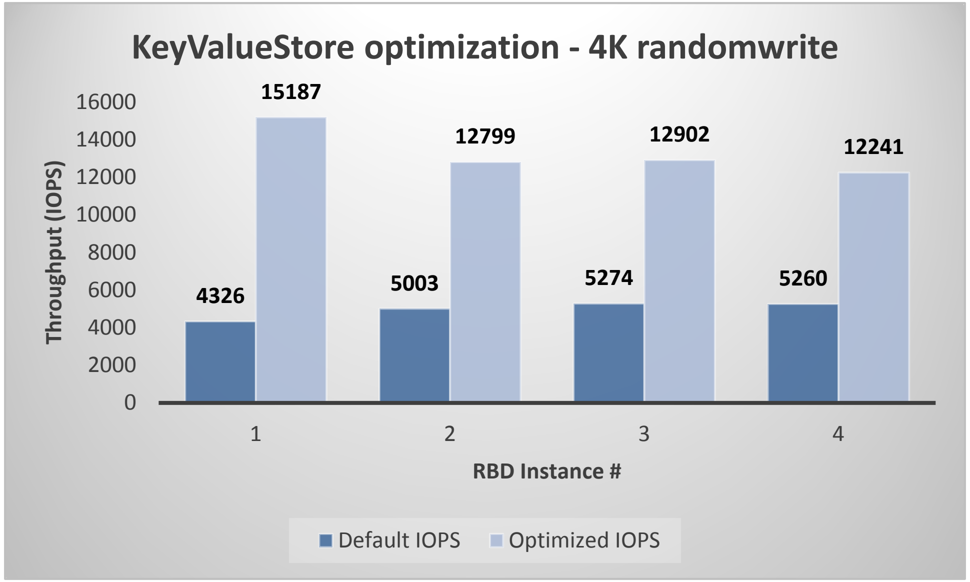
Ceph KeyValueStore analysis





- Sequential write is throttled by OSD Journal BW – up to 11x and 19x write amplification of LevelDB and Rocksdb due to compaction
- Sequential Read performance is acceptable – throttled by client NIC BW
- Random performance bottleneck is software stack – up to 5x higher latency compared with Filestore

Ceph KeyValueStore optimizations



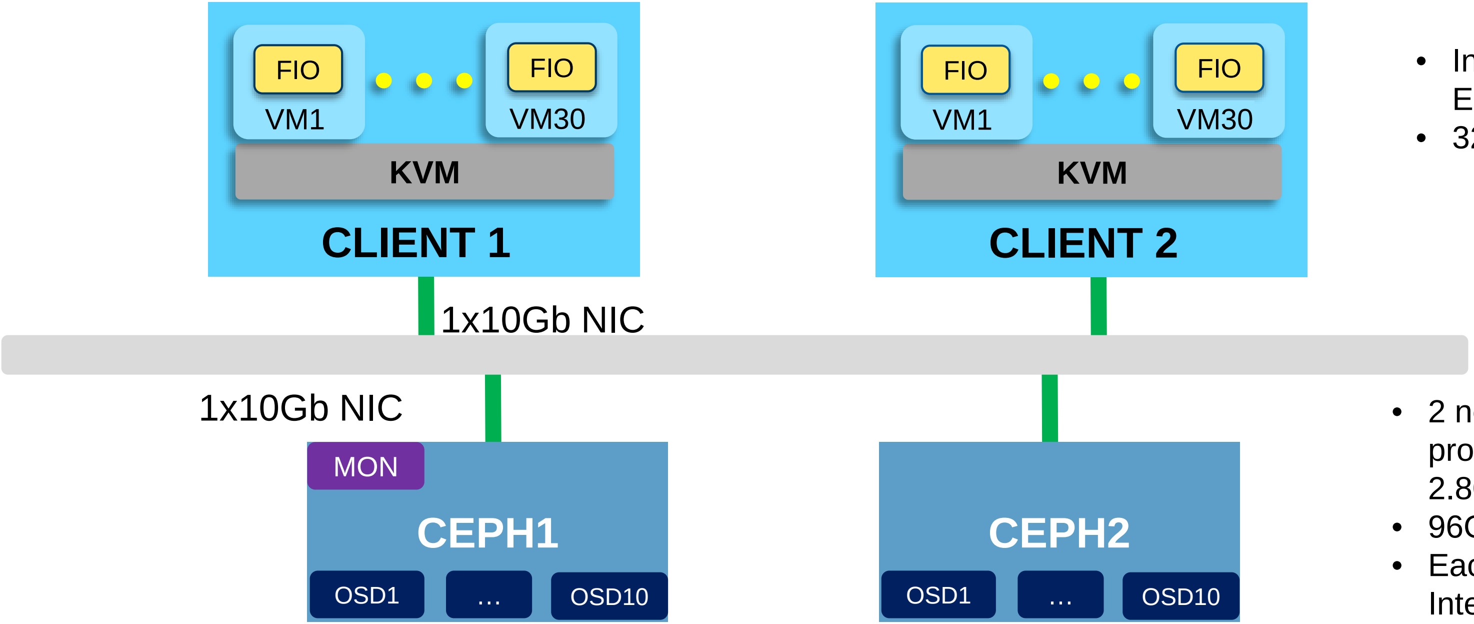
- Shorten Write path for KeyValueStore by removing KeyValueStore queue and thread (PR #5095)
- ~**3x** IOPS for 4K random write

Agenda

- Introduction
- Filestore performance on All Flash Array
- KeyValueStore performance on All Flash Array
- Ceph performance with Flash cache and Cache tiering on SSD
- Summary & next steps

Flashcache and Cache tiering System Configuration

Test Environment



Client Node

- Intel® Xeon ® processor E5-2680 v2 2.80GHz
- 32 GB Memory

Ceph OSD Node

- 2 nodes with Intel® Xeon ® processor E5-2680 v2 2.80GHz ,
- 96GB mem each node
- Each node has 4 x 200GB Intel® SSD DC S3700(2 as cache, 2 as journal), 10 x 1T HDD

Note: Refer to backup for detailed software tunings

Testing Methodology

- Storage interface
  - Use ***FIORBD*** as storage interface
- Data preparation
  - Volume is ***pre-allocated*** before testing with randwrite/randread
  - Volume size 30G
  - Rbd_num 60
  - Fio with Zipf:
    - Zipf 0.8 and 1.2: Modeling hot data access with different ratio
- Benchmark tool
  - Use **fio** (ioengine=libaio, iodepth=8) to generate 2 IO patterns: ***4K random write/read*** for IOPS
  - Empty & runtime drop_cache

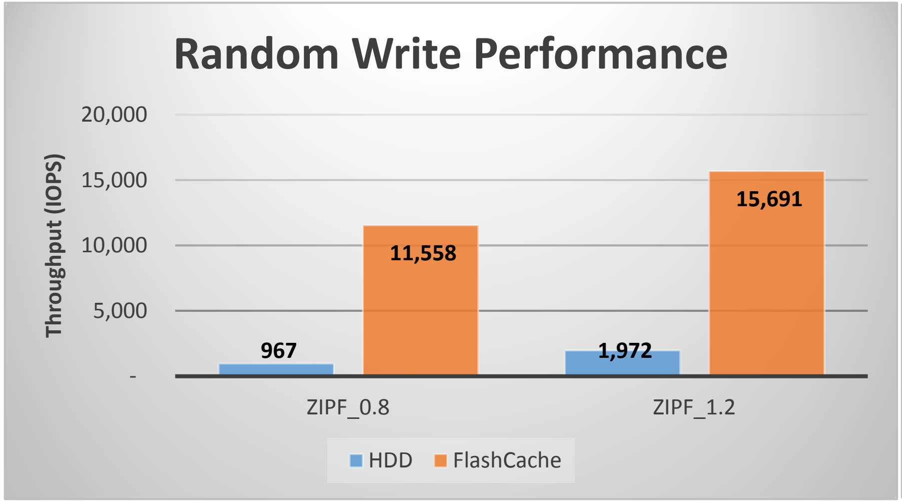
FlashCache and Cache Tiering Configuration

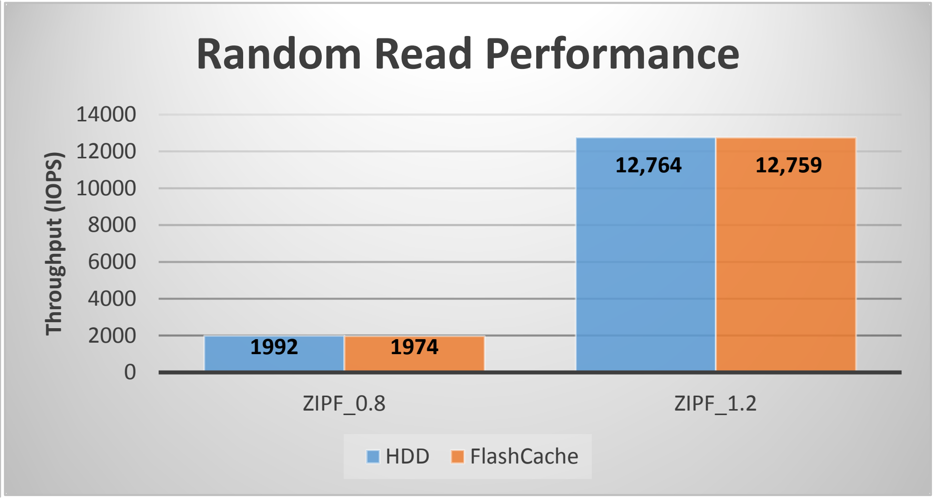
Flashcache:
- 4 Intel® DC 3700 SSD in Total
- 5 partitions on each SSD as flashcache
- Total capacity 400GB

Cache Tiering:
- 4 Intel® DC3700 SSD in Total
- Each as one OSD
- Total capacity: 800GB
- Cache Tier target max bytes: 600GB
- Cache tier full Ratio: 0.8

Note: Refer to backup for detailed zipf distribution

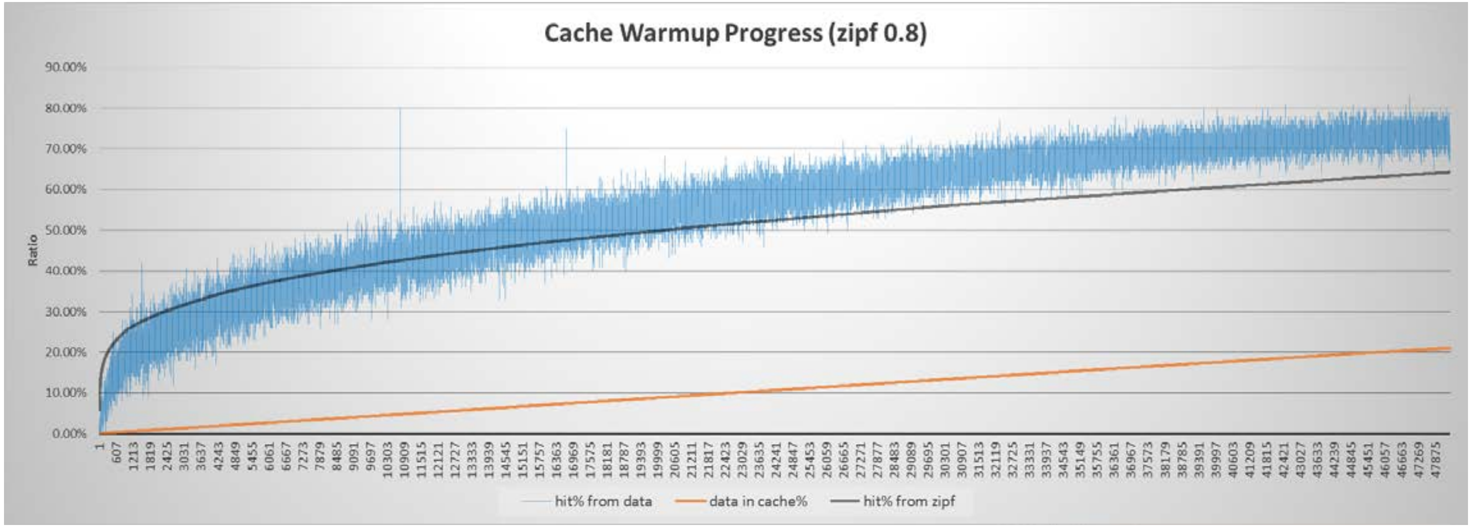
Flashcache Performance Overview





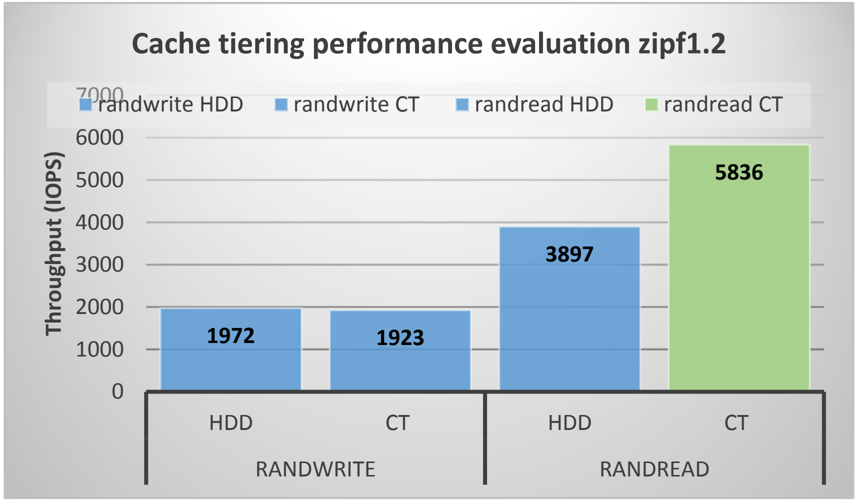
- For random write case, FlashCache can significantly benefit performance
  - IOPS increased by ~**12X** for zipf=0.8 and ~**8X** for zipf=1.2.
- For random read case, FlashCache performance is on par with that of HDD
  - This is because of FlashCache is not fully warmed up. Detail analysis can be found in following section.

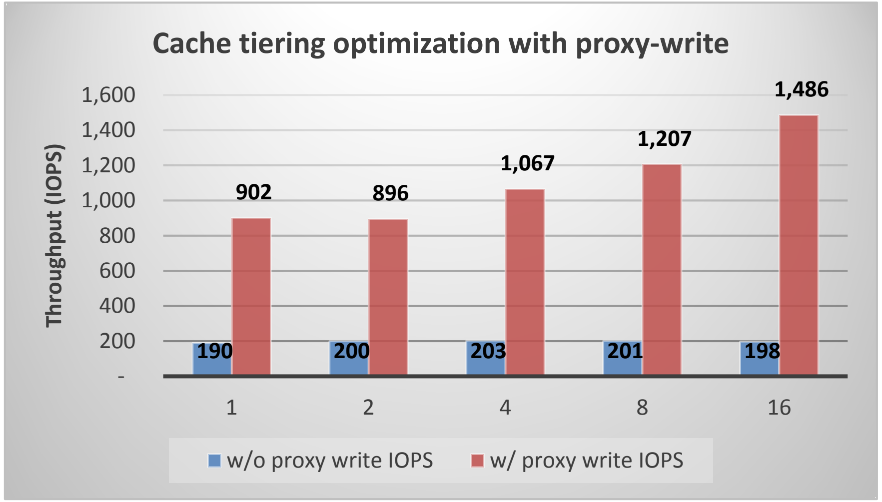
Cache Warmup Progress



- Cache warmup
  - ~ 20000s to read 5% of the total data into Cache(no matter pagecache or FlashCache), which is significant longer than our test runtime.
  - The random read throughput is throttled by HDD random read IOPS, which is 200 IOPS * 20= 4000 IOPS
- Flashcache benefits on random read is expected to be higher if cache is fully warmed

Cache Tiering





- Cache tiering demonstrated **1.46x** performance improvement for random read
  - Cache tier is warmed before testing through fio warmup
- For random write the performance is almost the same as without cache tiering
  - Proxy-write separates the promotion logic with replication logic in cache tiering
  - Proxy-write brought up to **6.5x** performance improvement for cache tiering

Agenda

- Background
- Filestore performance on All Flash Array
- KeyValueStore performance on All Flash Array
- Ceph performance with Flash cache and Cache tiering
- Summary & next steps

Summary & Next Step

- Cloud service providers are interested in using Ceph to deploy high performance EBS services with all flash array
- Ceph has performance issues on all flash setup
  - Filestore has performance issues due to messenger, lock and unbalance issues
    - Performance tunings can leads to **7x** performance improvement
  - KeyValueStore depends on KV implementation
  - Flashcache would be helpful in some scenario – **12x** performance improvement
  - Cache tiering need more optimization on the promotion and evict algorithm
- Next Step
  - Path finding the right KV backend
  - Smart way to use HDD and SDD together
  - NVMe optimization

QA

Backup

Software Configuration – ceph.conf

```
[global]
    debug_lockdep = 0/0
    debug_context = 0/0
    debug_crush = 0/0
    debug_buffer = 0/0
    debug_timer = 0/0
    debug_filer = 0/0
    debug_objecter = 0/0
    debug_rados = 0/0
    debug_rbd = 0/0
    debug_ms = 0/0
[osd]
    osd_enable_op_tracker: false
    osd_op_num_shards: 10
    filestore_wbthrottle_enable: false
    filestore_max_sync_interval: 10
    filestore_max_inline_xattr_size: 254
    filestore_max_inline_xattrs: 6
    filestore_queue_committing_max_bytes: 1048576000
    filestore_queue_committing_max_ops: 5000
    filestore_queue_max_bytes: 1048576000
    filestore_queue_max_ops: 500
    journal_max_write_bytes: 1048576000
    journal_max_write_entries: 1000
    journal_queue_max_bytes: 1048576000
    journal_queue_max_ops: 3000
    debug_monc = 0/0
    debug_tp = 0/0
    debug_auth = 0/0
    debug_finisher = 0/0
    debug_heartbeatmap = 0/0
    debug_perfcounter = 0/0
    debug_asok = 0/0
    debug_throttle = 0/0
    debug_mon = 0/0
    debug_paxos = 0/0
    debug_rgw = 0/0
    mon_pg_warn_max_per_osd=1000
    ms_nocrc=true
    throttle_perf_counter=false
```

Zipf distribution

Zipf:0.8

Rows	Hits %	Sum %	# Hits	Size
Top 5.00%	41.94%	41.94%	3298659	12.58G
|-> 10.00%	8.60%	50.54%	676304	2.58G
|-> 15.00%	6.31%	56.86%	496429	1.89G
|-> 20.00%	5.28%	62.14%	415236	1.58G
|-> 25.00%	4.47%	66.61%	351593	1.34G
|-> 30.00%	3.96%	70.57%	311427	1.19G
|-> 35.00%	3.96%	74.53%	311427	1.19G
|-> 40.00%	3.25%	77.78%	255723	998.92M
|-> 45.00%	2.64%	80.42%	207618	811.01M
|-> 50.00%	2.64%	83.06%	207618	811.01M
|-> 55.00%	2.64%	85.70%	207618	811.01M
|-> 60.00%	2.64%	88.34%	207618	811.01M
|-> 65.00%	2.42%	90.76%	190396	743.73M
|-> 70.00%	1.32%	92.08%	103809	405.50M
|-> 75.00%	1.32%	93.40%	103809	405.50M
|-> 80.00%	1.32%	94.72%	103809	405.50M
|-> 85.00%	1.32%	96.04%	103809	405.50M
|-> 90.00%	1.32%	97.36%	103809	405.50M
|-> 95.00%	1.32%	98.68%	103809	405.50M
|-> 100.00%	1.32%	100.00%	103800	405.47M

Zipf:1.2

Rows	Hits %	Sum %	# Hits	Size
Top 5.00%	91.79%	91.79%	7218549	27.54G
|-> 10.00%	1.69%	93.47%	132582	517.90M
|-> 15.00%	0.94%	94.41%	73753	288.10M
|-> 20.00%	0.66%	95.07%	51601	201.57M
|-> 25.00%	0.55%	95.62%	42990	167.93M
|-> 30.00%	0.55%	96.16%	42990	167.93M
|-> 35.00%	0.29%	96.45%	22437	87.64M
|-> 40.00%	0.27%	96.72%	21495	83.96M
|-> 45.00%	0.27%	96.99%	21495	83.96M
|-> 50.00%	0.27%	97.27%	21495	83.96M
|-> 55.00%	0.27%	97.54%	21495	83.96M
|-> 60.00%	0.27%	97.81%	21495	83.96M
|-> 65.00%	0.27%	98.09%	21495	83.96M
|-> 70.00%	0.27%	98.36%	21495	83.96M
|-> 75.00%	0.27%	98.63%	21495	83.96M
|-> 80.00%	0.27%	98.91%	21495	83.96M
|-> 85.00%	0.27%	99.18%	21495	83.96M
|-> 90.00%	0.27%	99.45%	21495	83.96M
|-> 95.00%	0.27%	99.73%	21495	83.96M
|-> 100.00%	0.27%	100.00%	21478	83.90M

Legal Notices and Disclaimers

Legal Information: Benchmark and Performance Claims Disclaimers

Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase.

Test and System Configurations: See Back up for details.

For more complete information about performance and benchmark results, visit http://www.intel.com/performance.