Ceph中国社区 IT大咖说 知识共享平台

CEPHALOCON APAC 2018

THE FUTURE OF STORAGE

22-23 March 2018 | BEIJING

# CEPH QoS

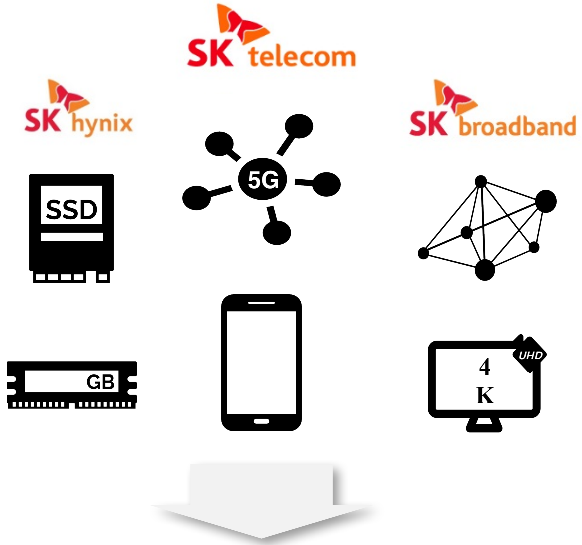
-

## How to support QoS in distributed storage system

-

Taewoong Kim
SW-Defined Storage Lab.
SK Telecom

# SK Telecom and Ceph



**Flash device**

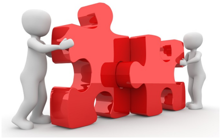
*High Performance, Low Latency, SLA*

*Scalable, Available, Reliable, Unified Interface, Open Platform*

*High Performance, Low Latency*

**All-flash Ceph !**

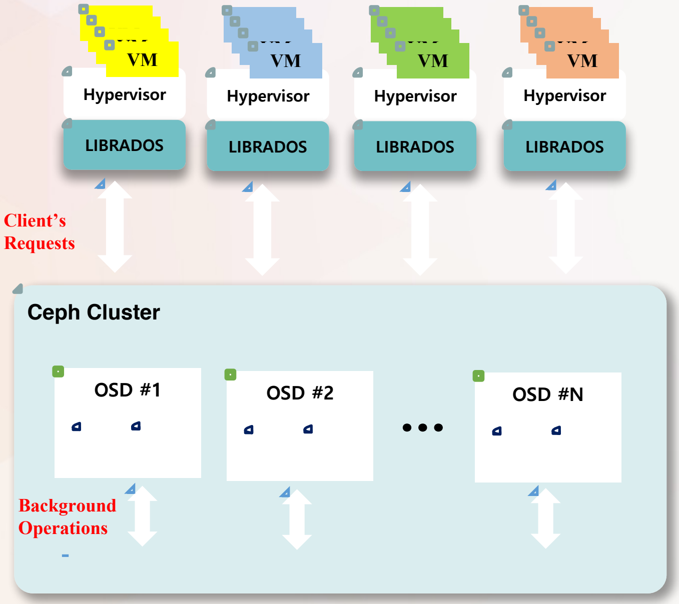
Contribution : QoS, Deduplication, etc.



**Storage Solution for**

- **Private Cloud for Developers**
- **Virtual Desktop Infra**

# Why QoS?



□ **Shared Storage : Virtualization, Consolidation**

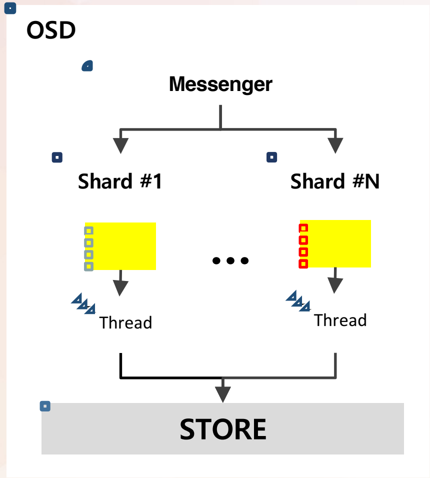
- Many tenants, even more VMs
  - ✓ Competition among clients for shared resource
  - ✓ Various workload & requirements
- Difficulty for Storage SLA
  - ✓ Not deterministic performance
    - Situation is changing every time rely on neighbors

□ **SW-defined storages like CEPH provide many features but need more background operations**

- Background Operations
  - ✓ Replication
  - ✓ Recovering
  - ✓ Scrubbing
- Competition with foreground(client's) operations

□ **QoS schedules requests along administrator's pre-configured policies**

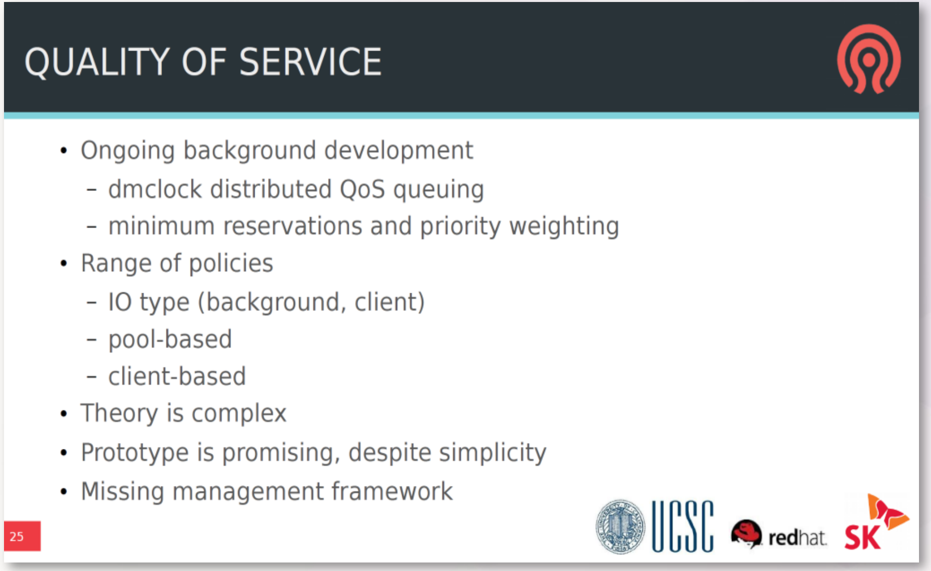
# QoS Support on Ceph



PriorityQueue :

- Weighted
- Prioritized

+ mClockOpClass
+ mClockClient
+ mClockPool(WIP by SKT)



## QUALITY OF SERVICE

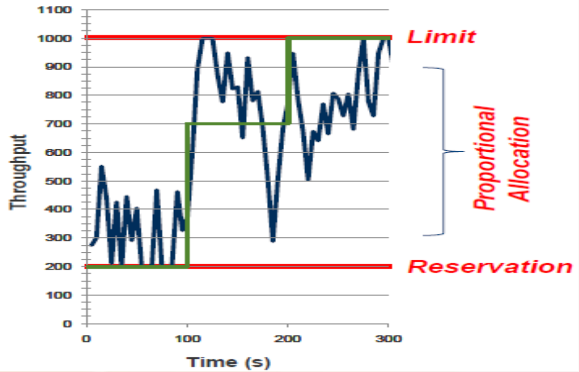
- Ongoing background development
  - dmclock distributed QoS queuing
  - minimum reservations and priority weighting
- Range of policies
  - IO type (background, client)
  - pool-based
  - client-based
- Theory is complex
- Prototype is promising, despite simplicity
- Missing management framework

25

UCSC
redhat
SK

- Source: Sage Weil's 'Ceph Project Update' in OpenStack Summit 2017 Boston

# What is mClock?



A. Gulati, A. Merchant, and P. J. Varman. mClock: Handling throughput variability for hypervisor IO scheduling. In OSDI, 2010

□ **Motivation**

- Lack of existing research to support QoS (reservation + proportional share) for storage
  - ✓ Support only simple proportional share
  - ✓ Support for other hardware devices (CPU, memory)

□ **Key Idea**

- Controls the number of I/O requests to be serviced using time tags
- Uses multiple real-time clocks & time tags for reservation, limit and weight(proportion) based I/O scheduling
- Dynamic clock selection depends on clocks' progress status

```
if ( smallest reservation tag < current time) // constraint-based
    Schedule smallest eligible reservation tag
else // weight-based, reservations are met
    Schedule smallest eligible shares tag
    Subtract 1/$r_k$ from reservation tags of VM k.
    A VM is eligible if (limit tag < current time)
```

- Can be extended for supporting distributed storage : dmClock
  - ✓ Clients track progress of each storage server & send feedback to each server with I/O requests

# QoS on SKT: Contributions

□ **Development and stabilization of QoS algorithm (https://github.com/ceph/dmclock)**

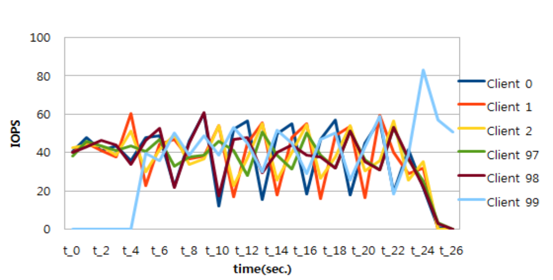
- Improved QoS instability in high load situations
- Fix QoS error due to heap management bug
- Fix tag adjustment algorithm calibrating proportional share tags against real time
- Enable changing QoS parameters in run time
- Improved Client QoS service status monitoring and reporting algorithm
- Add anticipation mechanism to solve deceptive idleness problem

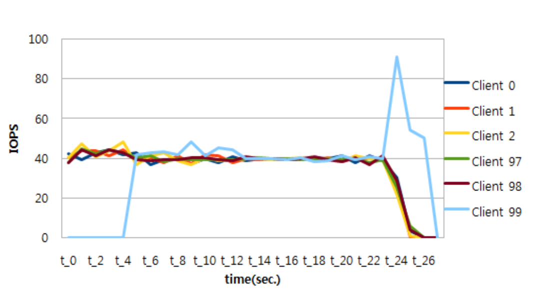
□ **QoS simulator stabilization and convenience improvement (https://github.com/ceph/dmclock)**

- File-based simulator settings
- High performance setup and fixed simulation error reporting error
- Fixed server node selection error

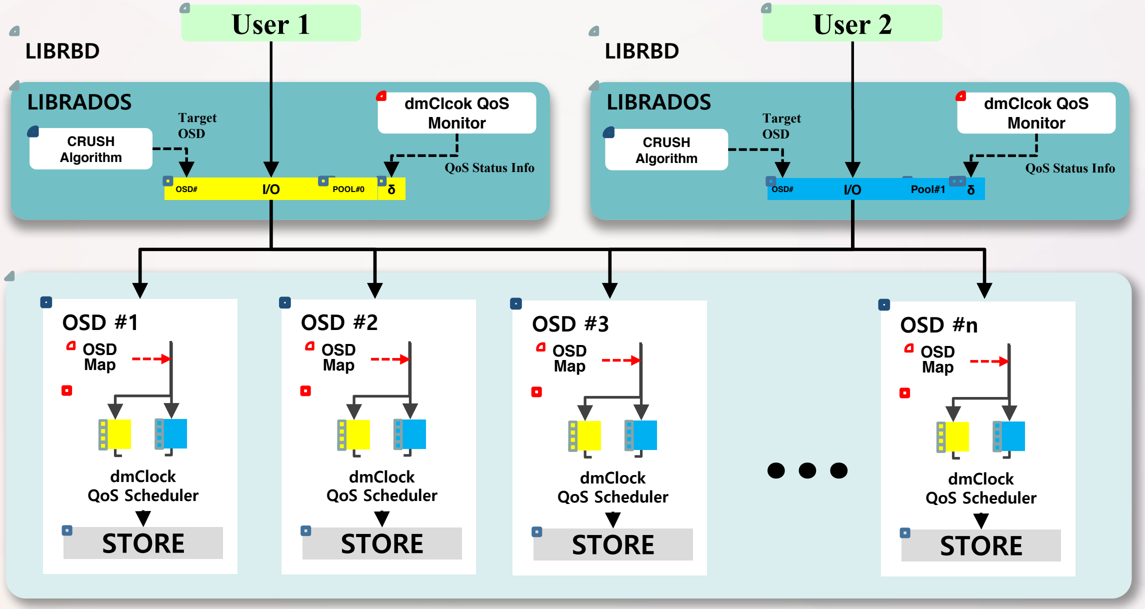
□ **Ceph Integration Work**

- Delivery of mClock distribution parameters (delta, rho and phase) + Enabling client QoS tracker (https://github.com/ceph/ceph/pull/16369)
- osd: use dmclock library client_info_f function dynamically (https://github.com/ceph/ceph/pull/17063)
- Pool based dmClock Queue (WIP) (https://github.com/ceph/ceph/pull/19340)
- Anticipation timeout Configuration (https://github.com/ceph/ceph/pull/18827)





# Current Implemented Ceph QoS Pool Units



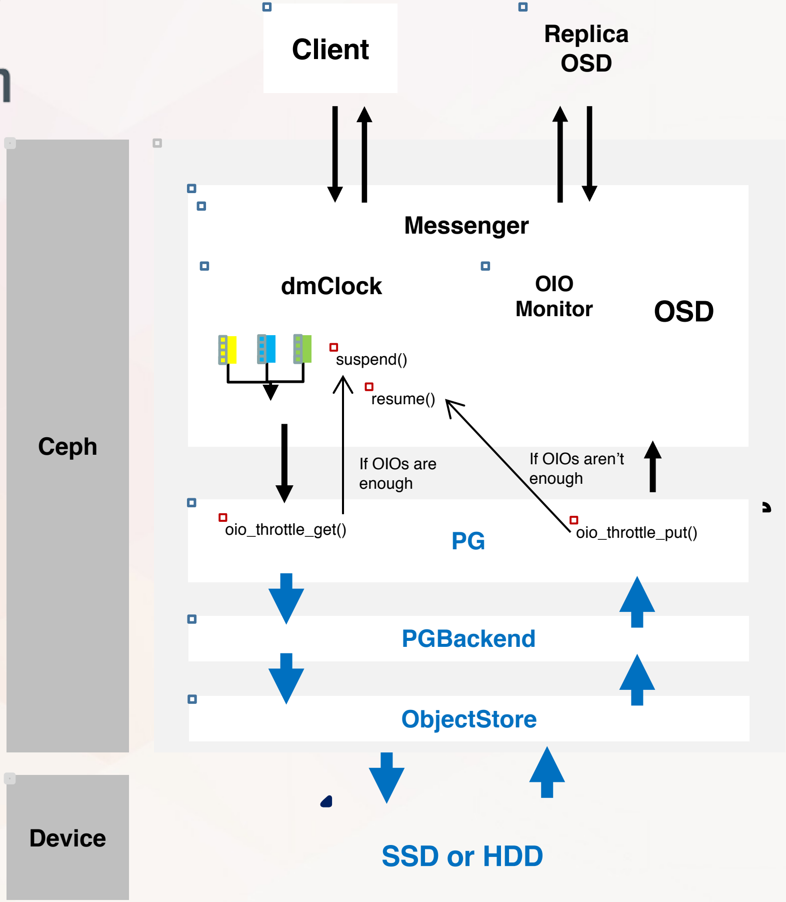
Pool's Properties {MIN, MAX,Weight]

1. Each pool's properties(MAX, MIN, Weight) are stored in the OSD Map

   CLI Example: *ceph osd pool set [pool ID] resv 4000.0 (IOPs)*

2. Distribution status is monitored by "dmClock QoS Monitor" and status info is embedded in RADOS requests
3. Since each OSD has the latest OSD Map, it can know the QoS control of the pool corresponding to the received request, and proceed QoS with **Pool's QoS properties & QoS status info**

# Outstanding IO Based Throttling



- **Throttler for QoS**
  - ✓ Dispatch the number of I/O requests that is just enough for exploiting the system.
  - ✓ The I/Os staying in the dmClock queue will be used for more accurate scheduling later.

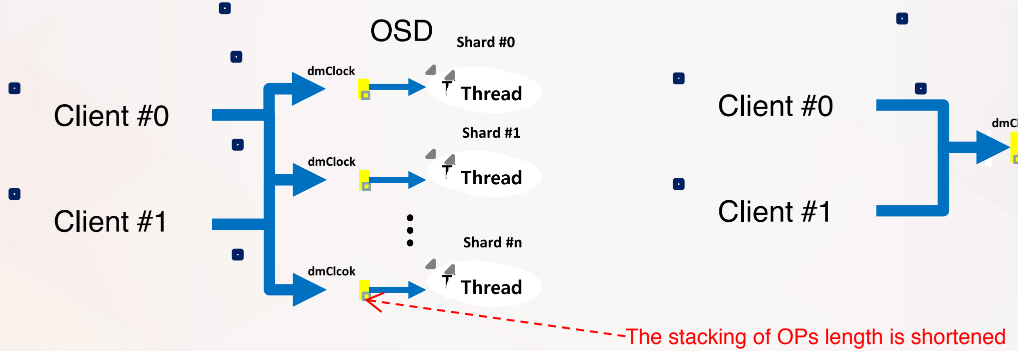
- **OIO Monitor**
  - ✓ Measure the average throughput.
  - ✓ Measure the average outstanding I/Os. (OIOs from PG to disk)
  - ✓ Tracking the maximum throughput.

- **Suspending dmClock scheduling**
  - ✓ Just proportional scheduling will be throttled
    - – Null operation will be returned
  - ✓ However, reservation scheduling will be continue because it's the minimum requirement.

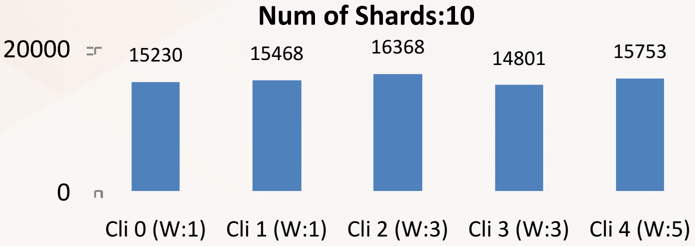
Measure current load on this range by counting outstanding IOs

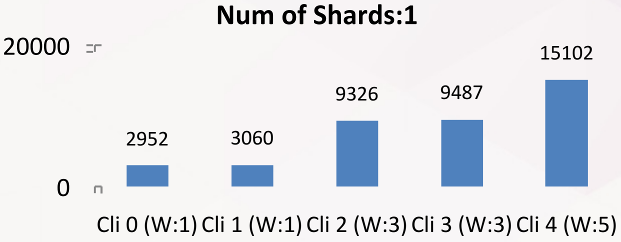
# Queue Depth Problem

- ✓ In one OSD, there will be dmClock queues depending on the number of shards (1-on-1)
- ✓ As operations are distributed by the increased number of dmClock queues, the average queue depth in one queue will be shorten
- ✓ Not enough requests in the queue result in No rearrangement, No competition
- ✓ Recommendation : Set the shard count to small number when using dmClock queue



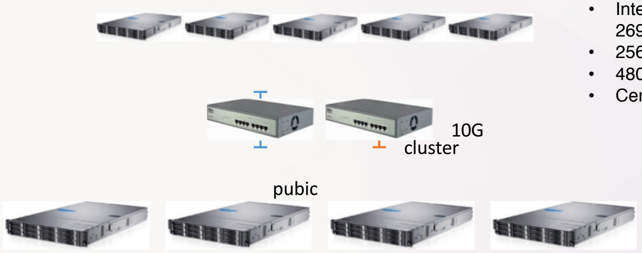
The stacking of OPs length is shortened



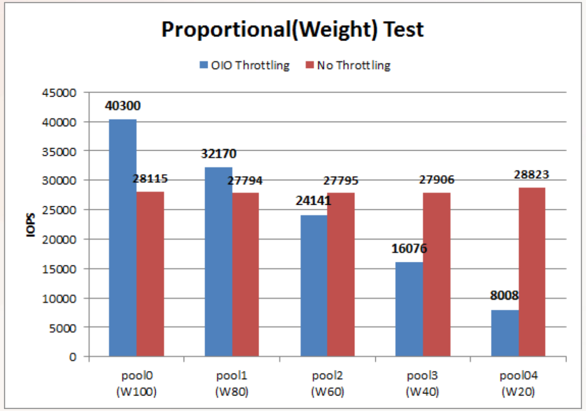


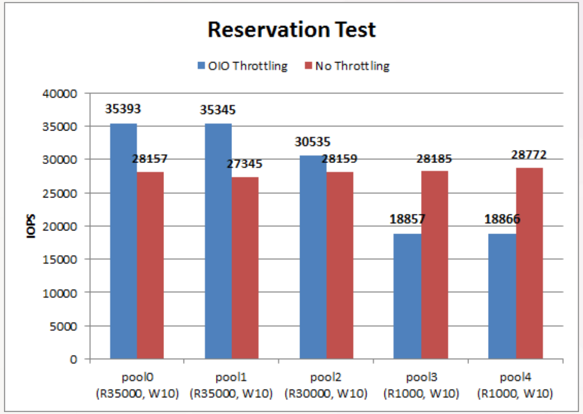
# OIO Based Throttler Test

- **Test env.**
  - ✓ 5 Client Nodes
    - Each client node uses single pool exclusively. so, 5 pools total are used for this test.
  - ✓ 4 OSD Nodes
    - 8 OSD daemons for single node.
    - BlueStore.
    - Single shard and 10 shard threads
  - ✓ Test Code
    - https://github.com/ceph/ceph/pull/17450
  - ✓ fio options
    - ioengine=rbd, 64 QD, 4 Jobs, 4KB rand write



- Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz
- 256GB Memory
- 480GB SATA SSD x 8
- CentOS 7





# Plan and so on...

- **Plan**
  - ✓ **Weighting on request size or its type.**
  - ✓ **Improve the dmClock algorithm**
  - ✓ **Extend to serve QoS to an individual RBD**
  - ✓ **Add more metric. (Throughput, Latency, etc...)**
  - ✓ **Test dmClock QoS & OIO throttler in various environments**

# Q & A