Making Ceph Fast in the Face of Failure

Neha Ojha
Senior Software Engineer, Red Hat
The Dark Ages

The Triumph of Death - Pieter Bruegel the Elder
Recovery vs Baseline IOPS

The graph shows the IOPS (Input/Output Operations Per Second) comparison between Baseline and Hammer. The y-axis represents the IOPS ratio compared to Baseline, while the x-axis shows the system labels: Baseline and Hammer. The Baseline system has a significantly higher IOPS compared to the Hammer system.
Hammer

- Favored maximum recovery speed
- Default client impact was huge
- Tuning could help

- osd max backfills = 10 → 1
- osd recovery max active = 15 → 3
- osd recovery op priority = 10 → 3
- osd recovery max single start = 5 → 1
Recovery vs Baseline IOPS

- Baseline
- Hammer
- Infernalis
Luminous – Throttling Recovery

- osd_recovery_sleep
- changing this will shift the balance between recovery and client I/O
- Different configurations based on underlying hardware
Recovery Sleep - HDDs

Bluestore on HDDs with Fio 4k random writes

➢ osd_recovery_sleep_hdd chosen as 0.1
Recovery Sleep - SSDs

➢ osd_recovery_sleep_ssd chosen as 0

Bluestore on SSDs with Fio 4k random writes
Recovery Sleep - Hybrid

Bluestore on HDDs+SSDs with Fio 4k random writes

➢ osd_recovery_sleep_hybrid chosen as 0.025
Luminous Defaults are much better

Recovery vs Baseline IOPS

IOPS / Baseline

Baseline | Hammer | Infernalis | Luminous
Recovery Priority

Manual way to recover higher-level constructs, e.g. rbd images

pg force-recovery command

```bash
ceph pg force-recovery {pg-id} [{pg-id #2}] [{pg-id #3} ...]
ceph pg force-backfill {pg-id} [{pg-id #2}] [{pg-id #3} ...]
```

If you change your mind or prioritize wrong groups, use:

```bash
ceph pg cancel-force-recovery {pg-id} [{pg-id #2}] [{pg-id #3} ...]
ceph pg cancel-force-backfill {pg-id} [{pg-id #2}] [{pg-id #3} ...]
```
Improving Latency during Recovery

Recovery in Ceph has been a synchronous process - it blocked writes to an object until it was recovered.

Problem: This increases write latencies and affects availability.

Solution in Mimic: Asynchronous Recovery

Do not block writes on objects, which are only missing on non-actingset OSDs.

Perform recovery in the background on an OSD, out of the acting set, similar to backfill, and use the PG log to determine what needs recovery.
Async recovery targets - OSDs that are not part of the acting set and are chosen based on the following:

- approximate magnitude of the difference in length of logs is used as the cost of recovery, async recovery targets have higher cost to recover
- threshold `osd_async_recovery_min_pg_log_entries` (default value=100) is used to determine when asynchronous recovery is appropriate
- `min_size` replicas should be available
How does it work?

- Use choose_async_recovery_ec() or choose_async_recovery_replicated() to identify async_recovery_targets.
- If an object is missing on an async_recovery_target, don’t send the complete write to it.
- Only send log entries of writes, so that they can completely catch up during log based recovery.
- When recovery completes, allow the recovered async_recovery_targets to get back to the acting set.
Recovery Experiments

RGW Workload generated using Cosbench.

Operations - read, list, write, delete

Cluster prefilled ~ 30%, 40000 objects

Kill one OSD to induce recovery
Impact on Throughput during Recovery

Throughput (op/s) - No failure v/s OSD failure

- Throughput (op/s) - no failure
- Throughput (op/s) - osd failure
Impact on Latency during Recovery

**Avg-ProcTime(ms) - No failure v/s OSD failure**

- **Avg-ProcTime(ms) - no failure**
- **Avg-ProcTime(ms) - osd failure**

### Actions
- **read**
- **list**
- **write**
- **delete**

### Time in ms
- **0**
- **50**
- **100**
- **150**
- **200**
Throughput Comparison - COSBench

No Failure

throughput Graph

s1-no_failure

0 op/s  25 op/s  50 op/s  75 op/s
0 28 52 78 104 130 156 182 208 234 260 286 312 338 364 390 416 442 468 494 520 546 572 598 624 650 676 702 728 754 780 806 832 858 884 910 936

No Failure

throughput Graph

s2-osd_failure

0 op/s  10 op/s  20 op/s  30 op/s  40 op/s  50 op/s  60 op/s
0 21 42 63 84 105 126 147 168 189 210 231 252 273 294 315 336 357 378 399 420 441 462 483 504 525 546 567 588 609 630 651 672 693 714 735 756 777

OSD Failure
Nautilus – PG Log Hard Limit

- Recovery/backfill can consume unbounded amounts of memory to store PG logs
- Hard limit for the PG log length
Nautilus – Pool-level Forced Recovery

Manual way to prioritize - forced recovery/backfill at a pool-level

`osd force-recovery command`

`ceph osd pool force-recovery $pool_name`
`ceph osd pool force-backfill $pool_name`

If you change your mind or prioritize wrong pools, use:

`ceph osd pool cancel-force-recovery $pool_name`
`ceph osd pool cancel-force-backfill $pool_name`
Nautilus – Improved Async Recovery

- Original version in mimic - uses the difference in length of PG logs as the cost of recovery
- Nautilus - also takes into account missing objects to select OSDs for asynchronous recovery
- Now called `osd_async_recovery_min_cost`
- More realistic way of choosing OSDs to postpone recovery on
Nautilus – Backfill Improvement

- Calculate the tentative amount of space required for backfill
- Deny reservation if the expected amount is not available on the target
- Avoids going over mon_osd_backfillfull_ratio
Nautilus – Throttling Deletes

- `osd_delete_sleep`
- Helps to shift the balance between PG deletion and client I/O
- Different configurations based on underlying hardware
Nautilus – “Clay” erasure code plugin

- Coupled-Layer erasure code – experimental
- Savings in terms of network bandwidth and disk IO during recovery
- Developed as a part of academic research

Thanks to Myna V for the contribution!
Octopus - What’s next?

- Partial Recovery - PR#21722 merged!
- EC recovery below min_size - PR#17619
- Default high recovery priority for index and metadata pools
Future Work

- Adaptive recovery throttling - set the value of osd_recovery_sleep based on client load

- QoS
Summary: Upgrade!

- Much better defaults and finer tuning in Luminous, recovery sleep is all you need to change client i/o vs recovery balance
- Better performance in Mimic with recovery being asynchronous
- Nautilus - smarter + robust + better performance
- Watch out for new features in Octopus
THANK YOU

Neha Ojha : nojha@redhat.com
GitHub : neha-ojha
IRC : neha