THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
KOREA

서울

THE BUTTERFLY EFFECT OF A BROKEN DISK

TOP-DOWN CEPH TROUBLESHOOTING, ALL THE WAY TO AN UPSTREAM CONTRIBUTION

SANGYUN LEE
AI Platform Engineer · CJ OliveNetworks

서울

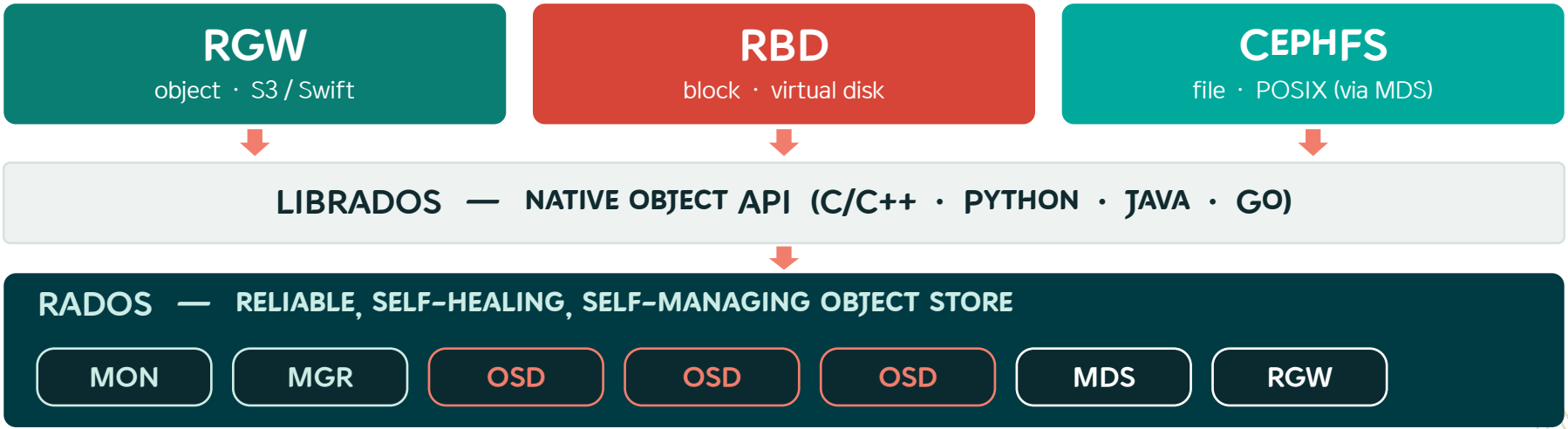
PART 1

CEPH 101 & OUR SMALL REALITY

WHAT CEPH IS — AND WHY "SMALL" MATTERS LATER.

CEPH 101 — ONE CLUSTER, ALL STORAGE

Open-source distributed storage (Red Hat). Many access methods — one self-healing object store.



Data path: Object → PG → OSD · CRUSH spreads data · 3× replicated · no single point of failure

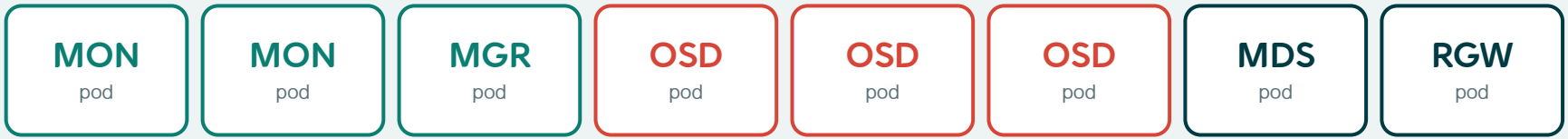
ROOK — CEPH, MADE KUBERNETES-NATIVE

ROOK

Rook is the operator that runs & manages Ceph as Kubernetes pods. Developers just use StorageClass + PVC.

↑ THIS STORAGECLASS CHOICE BECOMES OUR FIRST SUSPECT.

KUBERNETES CLUSTER — ROOK-MANAGED CEPH DAEMONS, EACH RUNNING AS A POD



CEPH IS USUALLY... HUGE

Most Ceph you read about is hyperscale. For example, the LINE / Yahoo group runs it like this:

30	2,500+	70,000+	700 PB
Ceph clusters	nodes	disks	capacity

Source: LY Corporation Engineering Blog — "Hyperscale Ceph project"

OUR REALITY — A SMALL "RESEARCH" CLUSTER

Key for the whole talk: an internal AI research cluster — not a production fleet. Small, shared, busy.

2	~15	~20	60 TB
Ceph clusters	nodes	NVMe OSDs	capacity

70,000+ DISKS VS ~20 OSDS — HOLD THE GAP. "SMALL" IS THE PLOT TWIST.

SO... WHY CEPH, AT THIS SCALE?

Fair question — isn't Ceph overkill for ~20 disks? Three reasons it earns its place:

1

ONE CLUSTER, EVERY STORAGE TYPE

Block, file, and object (RBD · CephFS · RGW) for K8s — instead of running a separate NFS, MinIO, and SAN.

2

SURVIVES NODE FAILURES

3× replication across nodes: lose a node, lose nothing. Disaster recovery becomes simple.

3

SELF-SERVICE FOR RESEARCHERS

Rook + CSI: a PVC is all they need. Volumes in seconds — no storage tickets, no waiting.

Small cluster, but not simple needs — one Ceph beats three ad-hoc storage systems.

서울

PART 2

THE SUSPECT HUNT

TOP-DOWN HYPOTHESIS TESTING: CLEAR ONE LAYER AT A TIME.

ONE ORDINARY DAY — THE FOOTSTEPS

"Infra team — Valkey is super slow."

"apt install just hangs inside my pod..."

"pods keep restarting, I can't develop."

TWO PROBLEMS AT ONCE

Disk I/O in pods painfully slow;
Valkey WAL writes so slow that pods crash-loop.

Restart pods to stop the bleeding → then hunt the real cause. Principle: Top-Down.

SUSPECT #1 — IS CEPHFS SLOW BY DESIGN?

Every pod used CephFS. In CephFS, every I/O goes through the metadata server (MDS) — an anti-pattern for a DB. RBD skips it.

TESTED STEP BY STEP:

1. CephFS PVC mounts & works — OK
2. Move Valkey PVC: CephFS → RBD
3. RBD PVC mounts & works — OK

CEPHFS PATH

RBD PATH

RESULT: LATENCY UNCHANGED → not the StorageClass. Alibi.

SUSPECT #2 — IS IT THE NETWORK?

App is cleared, OS not yet — so maybe the network between services?

We run Calico (CNI) + Istio (service mesh), so metrics are easy to read.

I OPENED GRAFANA FOR THE WHOLE CLUSTER →

NO SPIKES. FLAT. → Suspect #2 alibi. Only storage + OS / hardware remain.

서울

PART 3

THE TRUTH ROOM

HOW ONE DYING DISK FROZE AN ENTIRE CLUSTER.

TRUTH ROOM — OPENING UP CEPH ITSELF

```
$ ceph osd perf
osd  commit_latency  apply_latency
  0              1              1
  5              2              2
  2            118            121
  8            104            109
  3              1              2
  1            133            140
  7              2              2
```

* recreated for clarity (values representative)

THE CLUE

A few OSDs on ONE node show 100ms+ latency.

FOR NVME SSD, THAT IS ABSURD.
THE VILLAIN LIVES ON THAT NODE.

WORKER-DEV-3 — CULPRIT ① : THE OS DISK

```
nbd15

airnd@worker-dev-3:~$ sudo tune2fs -l /dev/sda2 | grep -iE "state|error|time"
Filesystem state:         clean with errors
Errors behavior:          Continue
Last mount time:          Mon Dec 22 09:52:46 2025
Last write time:          Thu Feb  5 06:13:02 2026
Lifetime writes:          27 TB
FS Error count:           79716
First error time:         Wed Jan 21 13:58:37 2026
First error function:     htree_dirblock_to_tree
First error line #:       1083
First error inode #:      2490833
First error err:          EFSBADCRC
Last error time:          Thu Feb  5 06:13:02 2026
Last error function:      __ext4_find_entry
Last error line #:        1694
Last error inode #:       21509613
Last error err:           EFSBADCRC
```

tune2fs -l /dev/sda2 (real capture)

SDA2 — THE NODE'S OS DISK (EXT4)

"clean with errors", 79,716 errors, EFSBADCRC.

Important: sda is NOT a Ceph OSD — but the kernel keeps trying to recover it. (Key for the next slide.)

WORKER-DEV-3 — CULPRIT ② : THE DATA NVME

JOURNALCTL -K — NVME0N1 RESET LOOP (~1/HOUR)

```
airnd@worker-dev-3:~$ sudo journalctl -k -f
Mar 09 10:24:33 worker-dev-3.h200-140g-m1 kernel:  nvme0n1:
Mar 09 10:59:10 worker-dev-3.h200-140g-m1 kernel:  nvme0n1:
Mar 09 11:59:18 worker-dev-3.h200-140g-m1 kernel:  nvme0n1:
Mar 09 12:59:27 worker-dev-3.h200-140g-m1 kernel:  nvme0n1:
Mar 09 13:58:05 worker-dev-3.h200-140g-m1 kernel: EXT4-fs (sda2): error count since last fsck: 79716
Mar 09 13:58:05 worker-dev-3.h200-140g-m1 kernel: EXT4-fs (sda2): initial error at time 1768971517: htree_dirblock_to_tree:1083: ino
de 2490833
Mar 09 13:58:05 worker-dev-3.h200-140g-m1 kernel: EXT4-fs (sda2): last error at time 1770239582: __ext4_find_entry:1694: inode 21509
613
Mar 09 13:59:37 worker-dev-3.h200-140g-m1 kernel:  nvme0n1:
Mar 09 15:00:17 worker-dev-3.h200-140g-m1 kernel:  nvme0n1:
Mar 09 16:01:24 worker-dev-3.h200-140g-m1 kernel:  nvme0n1:
Mar 09 17:01:50 worker-dev-3.h200-140g-m1 kernel:  nvme0n1:
```

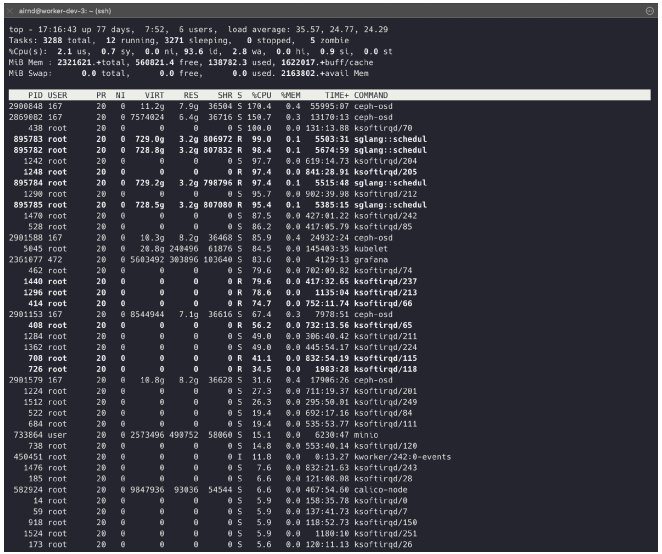
IOSTAT — NVME0N1 GONE

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.45    0.00    1.27    1.72    0.00   95.56

Device            r/s     rkB/s   rrqm/s  %rrqm r_await rareq-sz     w/s     wkB/s   wrqm/s  %wrqm w_await wareq-sz
    d/s     dkB/s   drqm/s  %drqm d_await dareq-sz     f/s f_await  aqu-sz  %util
nvme1c1n1       0.00      0.00     0.00   0.00    0.00     0.00    0.00     4.00     40.00     6.00  60.00    0.00    10.00
   0.00      0.00     0.00   0.00    0.00     0.00    0.00    0.00    0.00   0.80
nvme2c2n1      20.00    388.00     2.00   9.09    0.05    19.40  123.00 18416.00    286.00  69.93    0.73   149.72
   0.00      0.00     0.00   0.00    0.00     0.00    0.00    0.00    0.09   8.40
nvme3c3n1       0.00      0.00     0.00   0.00    0.00     0.00    0.00    29.00   8240.00   111.00  79.29    0.38   284.14
   0.00      0.00     0.00   0.00    0.00     0.00    0.00    0.00    0.01   2.00
nvme4c4n1       0.00      0.00     0.00   0.00    0.00     0.00    0.00    12.00   4112.00    56.00  82.35    0.50   342.67
   0.00      0.00     0.00   0.00    0.00     0.00    0.00    0.00    0.01   0.80
nvme5c5n1       3.00     16.00     0.00   0.00    0.00     5.33   84.00 16612.00    228.00  73.08   15.13   197.76
   0.00      0.00     0.00   0.00    0.00     0.00    0.00    0.00    1.27  16.40
sda             0.00      0.00     0.00   0.00    0.00     0.00    1.00     4.00      0.00   0.00    3.00     4.00
   0.00      0.00     0.00   0.00    0.00     0.00    0.00    0.00    0.00   0.40

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
```

nvme0n1 = 1 of 5 NVMe OSDs on this node — physically dying. Cause = aging hardware, not Ceph.

But only ~2 disks were dying — so why did the healthy NVMe OSDs slow down too, and freeze the whole cluster?

DEEP DIVE 1 (KERNEL) — THE KERNEL ATE THE CPU



```
top - 17:16:43 up 77 days,  7:52,  6 users,  load average: 35.57, 24.77, 24.29
Tasks: 3288 total,  12 running, 3271 sleeping,   0 stopped,   5 zombie
%Cpu(s):  2.1 us,  0.7 sy,  0.0 ni, 93.6 id,  2.8 wa,  0.0 hi,  0.9 si,  0.0 st
MiB Mem : 2321621.+total, 560821.4 free, 138782.3 used, 1622017.+buff/cache
MiB Swap:       0.0 total,      0.0 free,      0.0 used. 2163802.+avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
2900848 167       20   0   11.2g   7.9g  36504 S 170.4   0.4  55995:07 ceph-osd
2869682 167       20   0 7574024   6.4g  36716 S 150.7   0.3  13170:13 ceph-osd
    438 root      20   0       0      0      0 S 100.0   0.0 131:13.88 ksoftirqd/70
 895783 root      20   0  729.0g   3.2g 806972 R  99.0   0.1   5503:31 sglang::schedul
 895782 root      20   0  728.8g   3.2g 807832 R  98.4   0.1   5674:59 sglang::schedul
   1242 root      20   0       0      0      0 S  97.7   0.0 619:14.73 ksoftirqd/204
   1248 root      20   0       0      0      0 R  97.4   0.0 841:28.91 ksoftirqd/205
 895784 root      20   0  729.2g   3.2g 798796 R  97.4   0.1   5515:48 sglang::schedul
   1290 root      20   0       0      0      0 S  95.7   0.0 502:39.98 ksoftirqd/212
 895785 root      20   0  728.5g   3.2g 807080 R  95.4   0.1   5385:15 sglang::schedul
   1470 root      20   0       0      0      0 S  87.5   0.0 427:01.22 ksoftirqd/242
    528 root      20   0       0      0      0 S  86.2   0.0 417:05.79 ksoftirqd/85
2901588 167       20   0   10.3g   8.2g  36468 S  85.9   0.4  24932:24 ceph-osd
   5045 root      20   0   20.8g 248496  61876 S  84.5   0.0 145403:35 kubelet
2361677 472       20   0 5603492 303896 103640 S  83.6   0.0   4129:13 grafana
    462 root      20   0       0      0      0 S  70.6   0.0 702:09.82 ksoftirqd/74
   1440 root      20   0       0      0      0 R  79.6   0.0 417:32.65 ksoftirqd/237
   1296 root      20   0       0      0      0 R  78.6   0.0   1135:04 ksoftirqd/213
    414 root      20   0       0      0      0 R  74.7   0.0 752:11.74 ksoftirqd/66
2901153 167       20   0 8544944   7.1g  36616 S  67.4   0.3   7978:51 ceph-osd
    408 root      20   0       0      0      0 R  56.2   0.0 732:13.56 ksoftirqd/65
   1284 root      20   0       0      0      0 S  49.0   0.0 386:40.42 ksoftirqd/211
   1362 root      20   0       0      0      0 S  49.0   0.0 445:54.17 ksoftirqd/224
    708 root      20   0       0      0      0 R  41.1   0.0 832:54.19 ksoftirqd/115
    726 root      20   0       0      0      0 R  34.5   0.0   1983:28 ksoftirqd/118
2901579 167       20   0   10.8g   8.2g  36628 S  31.6   0.4  17906:26 ceph-osd
   1224 root      20   0       0      0      0 S  27.3   0.0 711:19.37 ksoftirqd/201
   1512 root      20   0       0      0      0 S  26.3   0.0 295:50.81 ksoftirqd/249
    522 root      20   0       0      0      0 S  19.4   0.0 692:17.16 ksoftirqd/84
    684 root      20   0       0      0      0 S  19.4   0.0 535:53.77 ksoftirqd/111
 733864 user      20   0 2573496 498752  58060 S  15.1   0.0   6230:47 minio
    738 root      20   0       0      0      0 S  14.8   0.0 553:40.14 ksoftirqd/120
 450451 root      20   0       0      0      0 I  11.8   0.0   0:13.27 kworker/242:0-events
   1476 root      20   0       0      0      0 S   7.6   0.0 832:21.63 ksoftirqd/243
    185 root      20   0       0      0      0 S   6.6   0.0 121:08.08 ksoftirqd/28
 582924 root      20   0 9847936  93036  54544 S   6.6   0.0 467:54.60 calico-node
     14 root      20   0       0      0      0 S   5.9   0.0 158:35.78 ksoftirqd/0
     59 root      20   0       0      0      0 S   5.9   0.0 137:41.73 ksoftirqd/7
    918 root      20   0       0      0      0 S   5.9   0.0 118:52.73 ksoftirqd/150
   1524 root      20   0       0      0      0 S   5.9   0.0   1180:10 ksoftirqd/251
    173 root      20   0       0      0      0 S   5.6   0.0 128:11.13 ksoftirqd/26
```

top — worker-dev-3 (real)

1. KERNEL CLINGS TO DYING DISKS

ext4 errors + hourly nvme resets → block-I/O errors run in softirq.

2. SOFTIRQ STORM → KSOFTIRQD

Per-CPU helper floods the cores. load average = 35.57.

3. CEPH-OSD STARVES

Real work gets no CPU. OSD can't process I/O → the 100ms+ we saw.

Honest: softirq starvation + disk corruption are measured; D-state/iowait are inference.

DEEP DIVE 2 (DISTRIBUTED) — SLOWEST COPY WINS

ALL 3 MUST COMMIT BEFORE THE CLIENT GETS AN ACK.

TAIL LATENCY

One slow copy stalls the 2 healthy ones. Slowest sets the speed.

GRAY FAILURE

Not dead, just late — Ceph waits for timeouts; PGs pile up slow ops.

SO VALKEY DIED

Its WAL mapped to a PG with that slow OSD → ack late → crash loop.

DEEP DIVE 3 (SCALE) — WHY "SMALL" MADE IT DEADLY

Only 1 OSD physically died — but CPU starvation made all 5 OSDs on the node slow.
Blast unit = one node = 5 OSDs.

Hyperscale (LINE)

5 slow OSDs / ~70,000 OSDs

≈ 0.01%

a local, almost invisible event

Our research cluster

5 slow OSDs / ~20 OSDs

≈ 25%

≈ a quarter of everything

Smaller cluster = bigger blast radius. One node's health decides the cluster's SLA. (simplified model)

THE FIX — AND WHY CEPH SAVED US

```
$ ceph osd down osd.2
$ ceph osd out  osd.2
marked out osd.2
# Ceph auto-rebalances to
#   healthy disks ...
```

* commands recreated (no capture)

LATENCY 130 MS → 2 MS

Mark bad OSD down → out. Ceph rebalances; latency back to normal immediately.

Shared pool → reschedule pods to a healthy node. No data loss, instant recovery.

Small scale made it hurt — but Ceph is also what saved us.

서울

PART 4

FROM USER TO CONTRIBUTOR

THE INCIDENT ENDED. BUT I TOOK ONE MORE STEP.

FROM USER TO CONTRIBUTOR

REMEMBER THIS SCREEN?

```
$ ceph osd perf
osd  commit_latency ...
  0         1
  5         2
  2       118
  8       104
  3         1
  ^ IDs unsorted!
```

during the outage, scanning these shuffled IDs drove me crazy

To operate a tool deeply, go from user to contributor.

The first contribution doesn't need to be big — start from the smallest thing that annoyed you.

That shuffled OSD-ID list became my first upstream PR.

SO I FIXED IT — PR #67915 (MERGED)

ceph/ceph · Pull Request #67915

✓ MERGED

MON/PGMAP: SORT 'OSD DF' AND 'OSD PERF' OUTPUTS BY OSD ID

falconlee236 merged into ceph:main · commit 93d40c1 · 13 checks passed ✓

```
// monitor printed OSDs straight
// from a hash map → no order
- unordered_map<int,..> m;
+ vector<int> ids(m...);
+ std::sort(ids.begin(),ids.end());
```

osd perf & osd df (plain) now sorted by OSD ID

CONTRIBUTOR — REVIEW "not sure osd df should sort by ID"

MEMBER — APPROVED ✓ "good to go — validate via teuthology (CI)"

RADOS TEAM — APPROVED ✓ merged into ceph:main

NOT AN ENDING — A BEGINNING

PR #67915
my fix · MERGED ✓

→

PR #68492
per-OSD latency KPIs · OPEN

My merged patch became someone else's starting point. **OPEN SOURCE KEEPS ROLLING.**

A small annoyance one bad night → code the whole world now runs.

WRAP-UP — THREE THINGS TO TAKE HOME

1. **GO TOP-DOWN** — clear each layer — app → network → storage → kernel → hardware.

2. **SMALL SCALE = BIG BLAST RADIUS** — one node can be a quarter of the whole cluster.

3. **USER → CONTRIBUTOR** — your smallest annoyance can be your first upstream PR.

What bugs you most in the tool you use every day? Maybe that's your first PR.

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT KOREA

서울

THANK YOU!

THE NEXT CHAPTER — YOUR FIRST PR — IS YOURS.

SANGYUN LEE

AI Platform Engineer · CJ OliveNetworks

GitHub @falconlee236 · Ceph PR #67915 (merged)

Come find me after the talk!

CONNECT ON LINKEDIN