OPENZFS DEVELOPER SUMMIT 2026

The Best of Both Worlds

Ceph clusters with a ZFS substrate

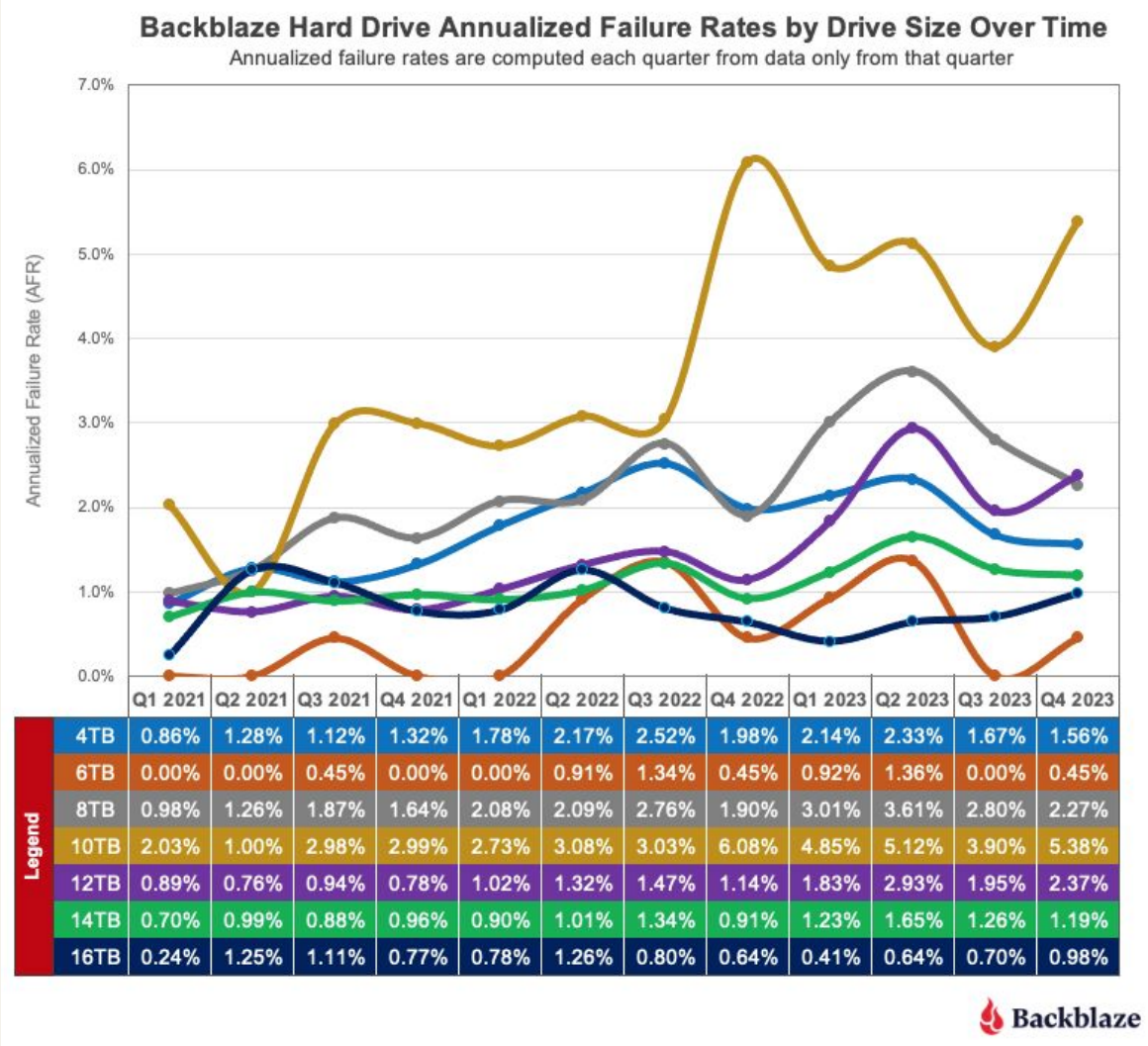
Designing scale-out clusters for scale, durability, and maintainability

Steve Umbehocker

CEO and Founder, OSNexus | Ceph Foundation Board Member

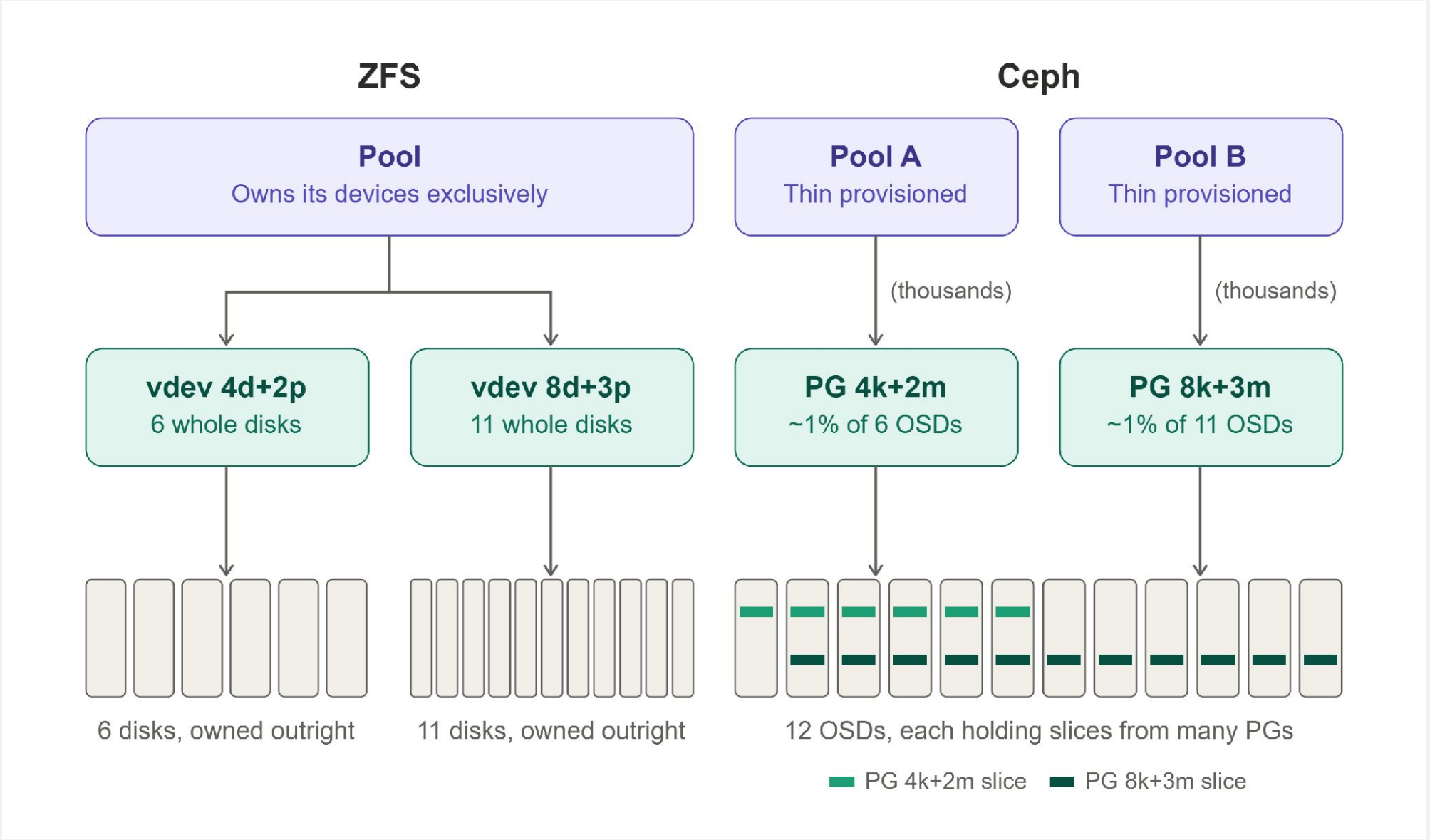
Problems at Scale

- Designing storage solutions at scale (over 1000 drives per cluster) requires additional durability.
- Cloud providers like AWS, Azure, Wasabi, Backblaze and others use a double-layer (or triple) to isolate and heal from drive failures locally rather than impacting the network and production workloads.
- Markov chain analysis shows isolated layers of fault tolerance deliver more 9s of uptime than a single layer with more parity bits and same usable capacity.



https://www.backblaze.com/blog/backblaze-drive-stats-for-2023/

Translating ZFS concepts to Ceph concepts



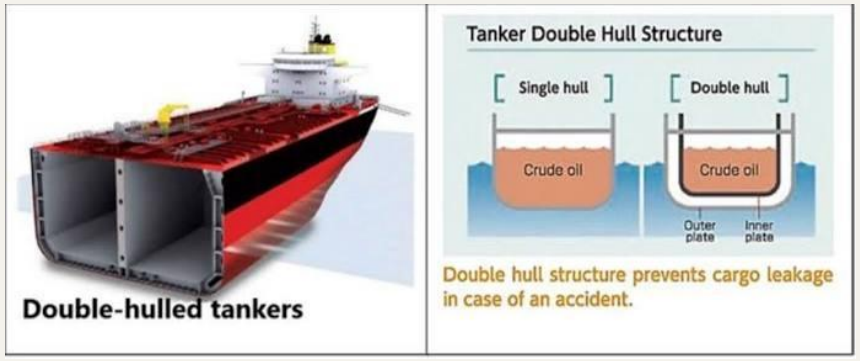
ZFS to Ceph Decoder Ring

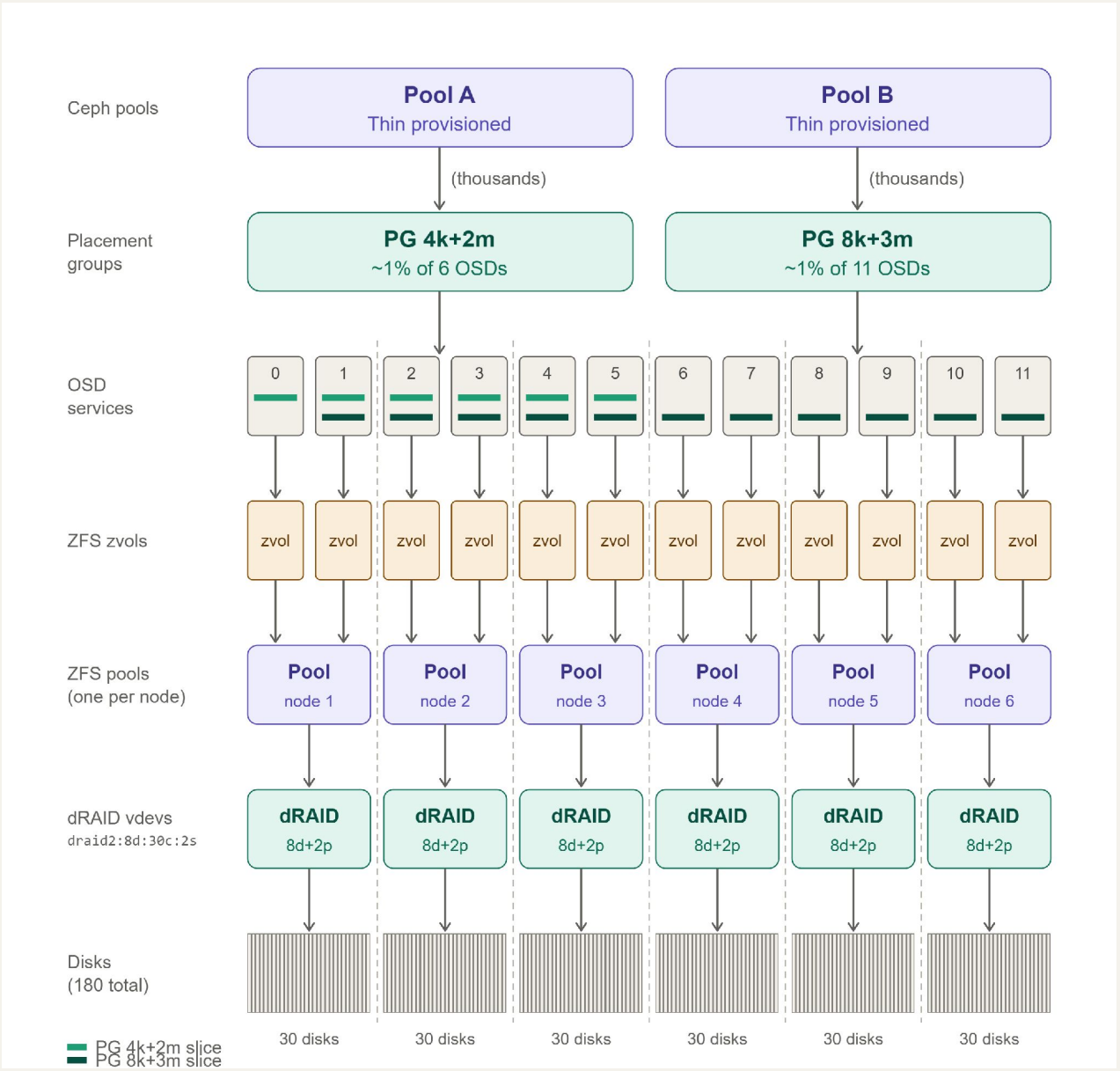
Layer	ZFS	Ceph
Pool	Pool Owns its block devices exclusively. Capacity is fixed by the devices in it.	Pool Thin provisioned. Many pools share the same set of devices.
Redundancy group	vdev Consumes entire devices. One vdev per set of disks.	Placement group (PG) Stripes use ~1% of each device. Thousands of PGs per device.
Layout notation	`4d+2p, 8d+3p` RAIDZ / dRAID data + parity	`4k+2m, 8k+3m` Erasure coding data chunks + coding chunks
Device	Block device Local to the host, addressed directly by the kernel.	OSD A service that puts a block device on the network and owns its block map and write log.
Mapping	`pool → vdev → block device`	`pool → PG → OSD`



Overview

- Double-layered architectures
- ZFS strengths vs Ceph strengths
- Using SANs/RAID under Ceph
- Using ZFS under Ceph
- Problems with Ceph at scale
- How ZFS solves those problems
- ZFS tuning params / optimizations needed when stacking Ceph on top





FRAMING

Double-layered architectures already exist in both worlds

Scale-up ZFS, layered

- ZFS over hardware RAID appliances (Seagate CORVAULT and others)
- ZFS over RAID-protected SAN LUNs and external arrays

Reason: the lower layer handles the physical drive; ZFS handles integrity and features.

Scale-out Ceph, layered

- Ceph over hardware RAID appliances (Seagate CORVAULT and others)
- Ceph over RAID-protected SAN LUNs and external arrays

Reason: the same. Keep the OSD up when a drive dies; let Ceph handle distribution.

STRENGTHS AND GAPS

Where each layer is strong

Capability	ZFS	Ceph
Horizontal scale	Scale-up single or dual-node HA	Scale-out across nodes (100+)
Drive failure	Local resilver, pool stays online, nothing above notices, no network load	Network backfill, PGs degraded for days
Integrity	Checksum on every block, self-healing on read and scrub	BlueStore CRC, repair only via scrub and replica
Snapshots	Instant, per dataset or zvol, pool-wide in one txg	Per RBD image or CephFS subtree; no cluster-wide point in time
Compression and encryption	Inline, native, per dataset	BlueStore compression; dm-crypt per OSD
Metadata offload	Special vdev: metadata and the ZIL	WAL and DB device per OSD

STRENGTHS AND GAPS

Our old way: Ceph over hardware RAID appliances

What it buys

- OSD survives a drive failure; rebuild is local to the shelf
- Fewer, larger OSDs per node, less OSD memory, lower CPU requirements, and fewer PGs
- Dense JBOD economics with enterprise-array failure handling
- Familiar to the storage team that already owns the arrays

What it costs

- No end-to-end checksum below BlueStore; parity is not a checksum
- Rebuild logic, scheduling and caching are opaque firmware
- No snapshots
- No concept of a special device for WAL and DB; NVMe sits beside, not inside
- Controller write-back cache becomes a hidden sync path
- Vendor-specific tooling for every alert, every firmware update



STRENGTHS AND GAPS

The new way: Ceph over ZFS on commodity JBOD

Clients	File / Block / Object
Ceph	MON, MGR, CRUSH · 20 - 40 OSDs per node, host failure domain · replication or EC across nodes
zvols	20-40 data zvols, volblocksize 128K, one OSD each · RocksDB WAL/DB colocated on the data zvol
ZFS pools	One pool per node · lz4 · logbias=latency · autotrim

dRAID2 HDD vdevs
Data zvols. Distributed spares. Resilver in hours.

Special vdev, two 3-way NVMe mirrors
All pool metadata, plus every commit as the ZIL (OpenZFS 2.4).

Hardware	97% HDD in 4U60 to 4U100, 10 NVMe with PLP, 1.5GHz per zvol/OSD, 6GB RAM per zvol/OSD



The arithmetic of scale: a bad drive every few weeks

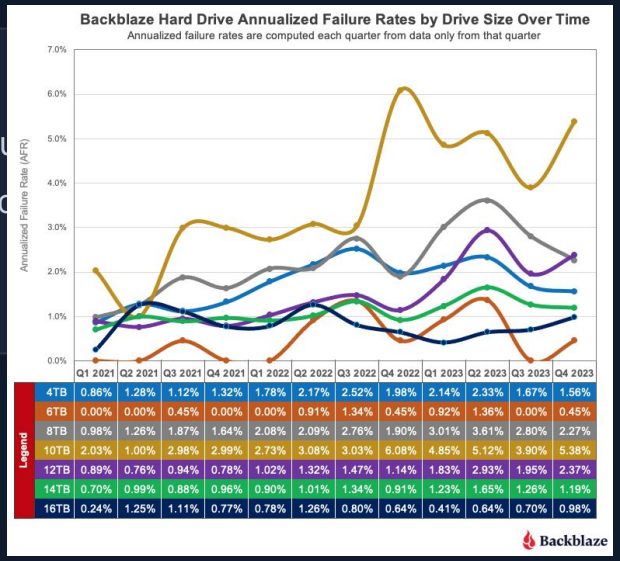
Drives	Failures per year at 1.5%	Days non-clean per year (Good year at 1.5%)	Days non-clean per year (Bad year at 2.5%)
500	8, one every 7 weeks	60 to 90	100 to 150
1,000	15, one every 24 days	120 to 180	200 to 300
2,500	38, one every 10 days	Continuous	Continuous
5,000	75, one every 5 days	Continuous, overlapping	Continuous, overlapping

Assumptions

24 TB HDD at ~80% fill. Backfill out of the dead OSD 4 to 6 days, backfill into the replacement another 4 to 6, under a client-priority recovery profile. AFR 1.5% is the Backblaze fleet average [cite current quarterly report].

Greenfield makes it worse

Same-age fleets follow the bathtu
mortality and wear-out arrive as c
Year one is also when drive, HBA



Legend	Q1 2021	Q2 2021	Q3 2021	Q4 2021	Q1 2022	Q2 2022	Q3 2022	Q4 2022	Q1 2023	Q2 2023	Q3 2023	Q4 2023
4TB	0.86%	1.28%	1.12%	1.32%	1.78%	2.17%	2.52%	1.98%	2.14%	2.33%	1.67%	1.56%
6TB	0.00%	0.00%	0.45%	0.00%	0.00%	0.91%	1.34%	0.45%	0.92%	1.36%	0.00%	0.45%
8TB	0.98%	1.26%	1.87%	1.64%	2.08%	2.09%	2.76%	1.90%	3.01%	3.61%	2.80%	2.27%
10TB	2.03%	1.00%	2.98%	2.99%	2.73%	3.08%	3.03%	6.08%	4.85%	5.12%	3.90%	5.38%
12TB	0.89%	0.76%	0.94%	0.78%	1.02%	1.32%	1.47%	1.14%	1.83%	2.93%	1.95%	2.37%
14TB	0.70%	0.99%	0.88%	0.96%	0.90%	1.01%	1.34%	0.91%	1.23%	1.65%	1.26%	1.19%
16TB	0.24%	1.25%	1.11%	0.77%	0.78%	1.26%	0.80%	0.64%	0.41%	0.64%	0.70%	0.98%

CEPH'S DRIVE-FAILURE PROBLEM

Anatomy of one drive failure in Ceph

T+0	Drive dies. OSD process fails its heartbeats and is marked **down**. Every PG with a copy on it re-peers.
T+10 min	mon_osd_down_out_interval (600 s) expires. OSD marked **out**, CRUSH remaps its PGs, second round of peering.
T+10 min	PGs queue for backfill reservations: osd_max_backfills per OSD, osd_recovery_max_active, mclock profile throttle the rate.
Days 1 to 5	Backfill off the dead OSD. ~20 TB read from surviving copies, reconstructed for EC, written to new targets. Cluster is **degraded** until backfill completes.
Day 5	Replacement drive installed, new OSD created and added to CRUSH. PGs remap toward it.
Days 5 to 10	Second backfill, into the replacement. Same throttles, same client contention. Cluster is **misplaced**, not degraded, but still not clean.
Day 10	HEALTH_OK. Scrub backlog and balancer catch-up begin.

CEPH'S DRIVE-FAILURE PROBLEM

No reboots: the min_size arithmetic



Healthy: 3 copies on nodes A, B, C

size 3, min_size 2. active+clean.



Drive dies on node B: 2 copies

Degraded, backfilling. I/O continues. Every PG that lived on that drive is in this state, spread across the whole cluster.



Reboot node C for firmware: 1 copy, below min_size

PG goes inactive. Client I/O to it blocks until node C returns. No data loss, but an outage for every volume or file touching an affected PG.

EC is the same arithmetic: 8+3 with min_size 9 and one shard rebuilding leaves one failure of headroom.

What that rules out

- Drive, HBA, backplane, BMC and NIC firmware updates
- Kernel patches
- Unsafe to restart some OSD(s) that are misbehaving for unrelated reasons
- Any reboot without noout and norebalance set first, or a second backfill wave starts at 600 s
- Peering storms: every PG on the node peers twice; reservations drop and re-queue
- The tempting fix, min_size 1, is how clusters lose data

The operational lockdown, until HEALTH_OK

Cannot do, or should not

- Reboot a node (use ok-to-shutdown check)
- Update drive, HBA, BMC or NIC firmware
- Patch the kernel
- Upgrade Ceph
- Restart a misbehaving OSD (depends)
- Add hosts or OSDs
- Change CRUSH rules or weights
- Change pool size or pg_num

Stops happening on its own

- Deep scrub of affected PGs
- Balancing
- PG autoscaling
- Snap trimming (by convention)
- Mirroring and multisite keeping up
- Full fault tolerance
- Predictable client latency
- Meaningful slow-op alerts

Backfill was designed for an occasional event. Past a thousand drives it is a background process, and every restriction tied to it becomes a permanent condition.

WHAT ZFS UNDERNEATH CHANGES

Same failure on ZFS: the drive dies, the vdev resilvers, Ceph never notices

Restrictions that no longer apply

- ✓ ~~Reboot a node~~ ordinary noout maintenance
- ✓ ~~Firmware, kernel patches~~ redundancy intact, go ahead
- ✓ ~~Upgrade Ceph~~ HEALTH_OK, as scheduled
- ✓ ~~Add OSDs, change CRUSH, pg_num~~ cluster is clean
- ✓ ~~Second rebuild on replacement~~ local, OSD stays up

Things that keep running

- ✓ Deep scrub on every PG, plus ZFS scrub underneath
- ✓ Balancer and PG autoscaler
- ✓ Snap trimming, mirroring, multisite
- ✓ Full Ceph-level fault tolerance
- ✓ No east-west traffic, no source-OSD read load
- ✓ Fullness ratios unchanged: no capacity left the pool

Ceph-level days non-clean per year from drive loss, at every row of the arithmetic table: **zero**. Resilver cost is local IOPS on one node, tunable there.

WHAT ZFS UNDERNEATH CHANGES

dRAID: the resilver that makes 24+ TB drives workable

RAIDZ2

- Healing resilver walks the block tree; random I/O on HDD
- Bounded by one replacement drive's write speed
- 24 TB: 1 to 2 days, longer under OSD load
- Hot spare sits idle until needed, then becomes the bottleneck

dRAID2 with distributed spares

- Sequential resilver onto spare capacity spread across every drive in the vdev
- Write bandwidth of the whole vdev, not one drive
- 24 TB rebuild measured in hours
- Healing scrub afterward verifies checksums without blocking redundancy

Reference geometry

`Example [draid2:8d:58c:2s] on the 4U60`

Why it matters here

The exposure window on the node drops from days to hours, the rebuild never leaves the chassis, and zfs_resilver_* tunables set the client tradeoff per node instead of per cluster.

WHAT ZFS UNDERNEATH CHANGES

Where BlueStore's pieces land in the pool

BLUESTORE, PER OSD	ZFS, ONE POOL PER NODE
RocksDB WAL and DB Small sync writes. Corruption here takes the OSD down.	**Colocated on the data zvol, volblocksize=128K** Checksummed, self-healing from dRAID parity. Small writes become ZIL records; the txg merges them into whole blocks.
Object data Large writes; BlueStore min_alloc_size matched to the zvol block.	**Data zvol, volblocksize=128K, on dRAID2** Whole-block writes, no read-modify-write. L1 metadata near 0.1% of pool, on the special vdev.
Flush at transaction commit One per commit on the data device. This is OSD commit latency.	**ZIL commit to the special vdev (OpenZFS 2.4), logbias=latency** Every zvol write commits on NVMe; the txg writes full dRAID stripes later.
BlueStore cache osd_memory_target (6 GB) caches onodes and RocksDB blocks.	**96 GB ARC, no L2ARC** Holds ZFS metadata; with primarycache=all it also catches RocksDB blocks BlueStore evicts.

Memory: OSD caches first, then a large ARC

120 to 160 GB

OSDs

20 × osd_memory_target at 6 to 8 GB. 4 GB is thin for a 55 TB HDD OSD, and the target is not a cap during recovery.

96 GB

ARC, zfs_arc_max set explicitly

All L2 and hot L1 indirect blocks, spacemaps, plus RocksDB blocks with primarycache=all. Never let ARC and OSDs fight; the OOM killer takes an OSD.

32 GB

Headroom

Kernel, co-located mon and mgr, peering spikes.

384 GB

Node total, 4U60

256 GB only with 6 GB OSD targets and a smaller ARC. No L2ARC.

Why metadata, and why volblocksize decides the size

1.1 PB usable at 128K is ~8.6 billion blocks. L1 indirect blocks are 128 B per pointer: ~1.1 TB of L1 on disk, ~1 GB of L2. The ARC holds only the hot part of L1; the special vdev makes a cold L1 miss a ~100 µs NVMe read instead of a seek. At 16K the same pool is ~9 TB of L1, which blows the special vdev before it blows the ARC.

WHAT ZFS UNDERNEATH CHANGES

NVMe log: yes, because the flush is the commit path

No NVMe log, logbias=throughput

Data goes indirect to the main pool and the ZIL record goes there too. Every flush is a small random write to HDD. Commit latency is HDD-bound.

This is the setup people benchmark, then blame ZFS.

Special vdev as the log, logbias=latency

On OpenZFS 2.4, every zvol write bound for HDD is copied to the special vdev at the flush and commits in microseconds; the txg writes it to HDD as full stripes afterward.

Cost: the special vdev carries the node's full write volume. Two 3-way mirrors split it.

sync=disabled

BlueStore's DB references data it believes was flushed. After a power loss the OSD will not start, and because DB and data are on different devices the corruption is silent until replay.

Never.

Sizing

The log needs little space: a few seconds of dirty data, 16 to 32 GB, beside 1 to 2 TB of metadata. Endurance is the real budget.

Device

PLP is mandatory. Size endurance by turnover: 7.68 TB at 1 DWPD covers ~20 full-node turnovers in 5 years.

Why 3-way

The special vdev holds all pool metadata, so losing it loses the pool. The log inherits that protection, with no SLOG mirrors to manage.

The reference node: where the NVMe goes, and what it trades

10 NVMe per node



Metadata OSDs — Special: two 3-way mirrors, metadata + ZIL — Spares

Ceph: 20 HDD OSDs, RocksDB colocated, 6 GB each
bucket index and metadata pools on the NVMe OSDs, replica-3

20 data zvols · volblocksize 128K = min_alloc 128K
logbias=latency · lz4 · sync=standard

dRAID2 8d+2p+2s
~50 HDD · resilver in hours

Special, two 3-way mirrors
all metadata + the ZIL

384 GB RAM: 120 GB OSD targets + 96 GB ARC + headroom

Needs OpenZFS 2.4: with no SLOG, the special vdev takes the ZIL (zil_special_is_slog). With spare NVMe, RocksDB can go there too.

Advantages

- One pool, one toolset, no SLOG devices to manage
- A dead drive stays inside the node: dRAID resilvers in hours and the OSDs stay up
- Every commit lands on NVMe in microseconds; HDDs only see full-stripe txg writes
- Pool metadata on NVMe: a cold lookup is ~100 µs, not a seek
- No WAL/DB devices to track through drive or OSD replacement

Trade-offs

- All object data passes through the special vdev: ~20 node turnovers of 1 DWPD endurance
- RocksDB cache misses read from HDD; the ARC helps only with primarycache=all
- Log and metadata share the special vdev: size it for both
- Special vdev loss is pool loss, hence 3-way
- Not yet measured; the NVMe test bed ran 2.3 with no HDDs

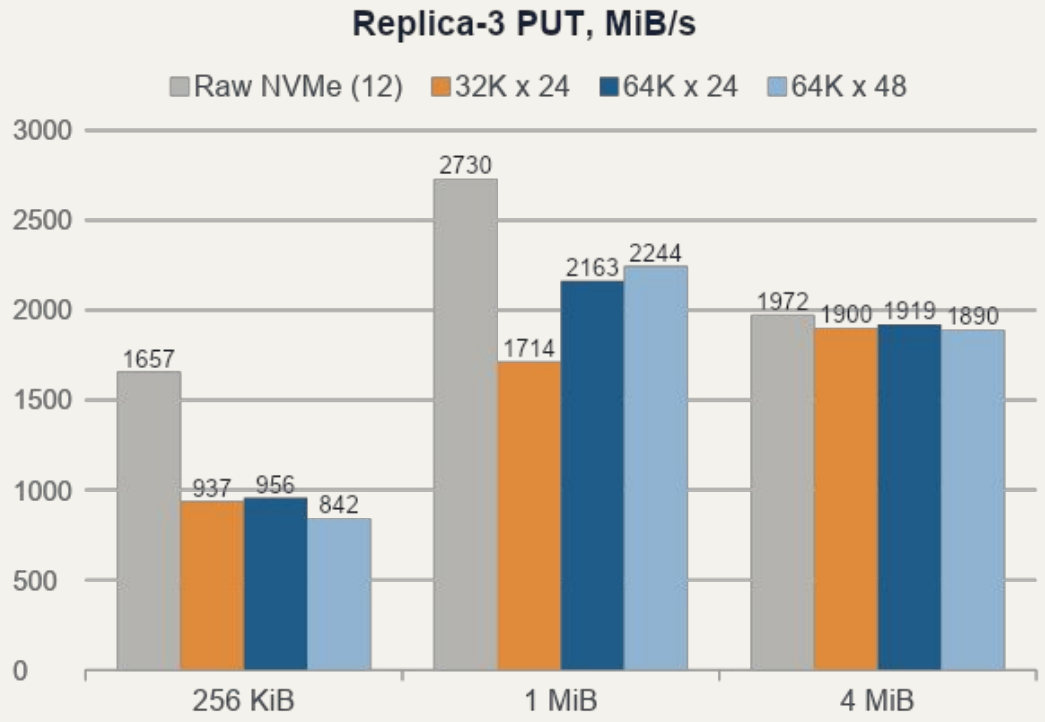
APPENDIX: RGW ON A 3-NODE NVMe BED

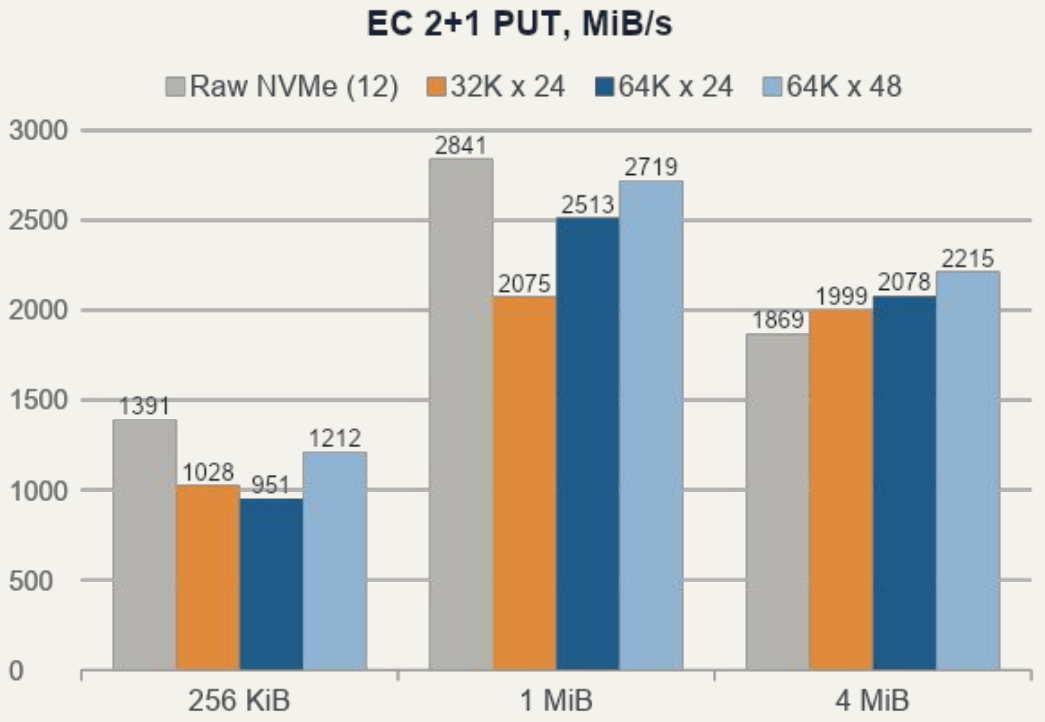
Test bed: 3 all-NVMe nodes, raw OSDs against zvol OSDs

Nodes	3 x Xeon Gold 6140 (72 threads), 251 GB RAM, 40GbE, 4 x Intel P4510 2 TB NVMe each
Software	Ceph 20.2.4, OpenZFS 2.3.9, QuantaStor 7.0 development build
Control	12 BlueStore OSDs, one per raw NVMe
ZFS layouts	One raidz1 pool per node (4 NVMe, ashift 12, lz4); 4, 8 or 16 zvols per node at 32K or 64K; min_alloc_size = volblocksize; ARC 25% of RAM, prefetch off
Workload	MinIO warp on the Ceph nodes against RGW; PUT then GET at 256 KiB, 1 MiB, 4 MiB; replica 3 and EC 2+1; 120 s per cell
Steady state	Every NVMe discarded, then 40% of raw capacity written and deleted with incompressible data, before every layout
Cold reads	OSDs restarted before each GET, BlueStore read cache off, GET reads the whole bucket
Measured	Client MiB/s and latency; node, OSD and ZFS CPU; NVMe bytes; ZIL bytes copied vs logged by reference

Read the ratios between layouts, not the MB/s. All-NVMe on OpenZFS 2.3.9: the reference node (HDD dRAID, special vdev as the log on 2.4) still needs its own run.

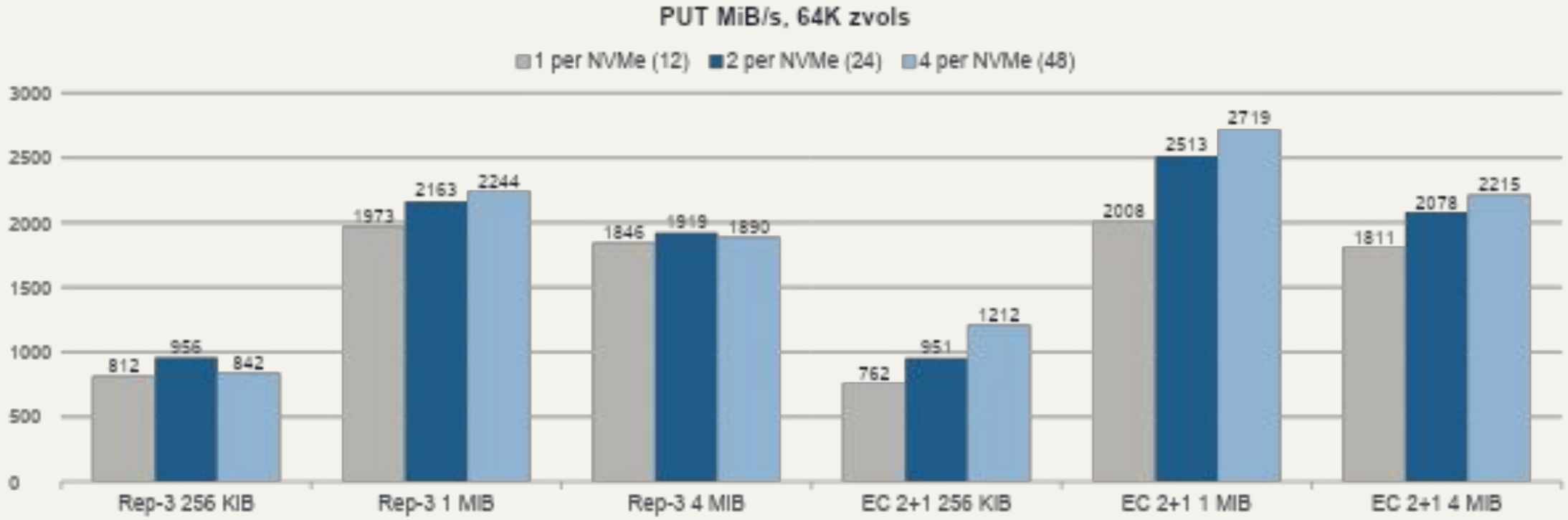
Large-object PUTs on 64K zvols: 82-119% of raw NVMe





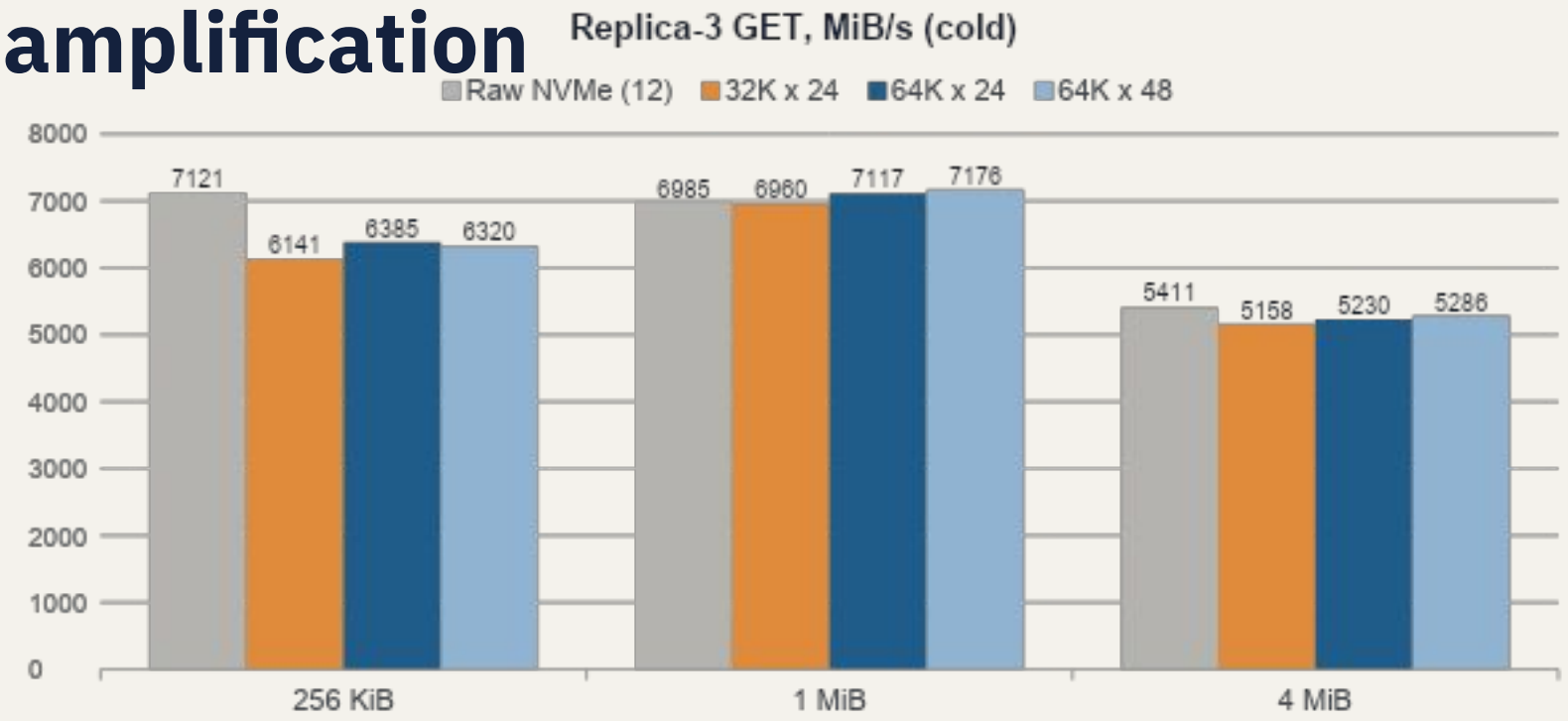
All zvol layouts logbias=latency on zpool 3d+1p. Raw NVMe: 12 OSDs, one per drive.

OSDs per NVMe: two for replica 3, up to four for EC



Same pools and drives; zvols (64K blk sz) per node 4, 8, 16 on raidz1 3d+1p.

Reads on zvols: 90-103% of raw NVMe, same read amplification



90-103%

of raw NVMe read throughput

1.03-1.06

NVMe bytes read per client byte, raw and zvol alike

~1.4x

raw's node CPU per GiB/s read

OSDs restarted before every GET; BlueStore read cache off; whole bucket read.

APPENDIX: RGW ON A 3-NODE NVMe BED

What to set for zvol OSDs, measured on NVMe

Setting	Set to	Evidence (all-NVMe bed)
volblocksize + logbias	64K with latency (QuantaStor default)	2.3: 32K latency 9.9x writes; 32K throughput 5.5x but EC 256 KiB -39%
min_alloc_size	= volblocksize	Whole-block BlueStore writes, no read-modify-write
OSDs per NVMe	2 for replica-3, up to 4 for EC	1 to 2: +18-25% at 256 KiB; EC 2 to 4: +7-27%
zvol_request_sync	0 (OpenZFS default)	QuantaStor previously shipped 1, found a oops with blk-mq fixing upstream
ARC / prefetch	25% of RAM / off	Reads 1.03-1.06 bytes per byte, same as raw NVMe
compression	lz4 on the zvol; BlueStore compression only if you need the space	Zvol savings are invisible to Ceph: don't overcommit zvols

Result 64K zvols: 82-119% of raw NVMe on large-object writes, 98-103% on reads, at 1.4-1.8x the CPU per byte.

ASKS

What would make ZFS a better substrate

zvol blk-mq at OSD density

20 OSDs on 20 zvols is not one zvol under a hypervisor. With non-default settings, blk-mq zvols crashed under batched O_DIRECT; a fix is headed upstream.

Direct I/O on zvols

BlueStore opens its block device O_DIRECT. A zvol path that honors it, as 2.3 does for datasets, would let BlueStore's cache and the ARC stop overlapping by default.

dRAID resilver under sustained OSD load

A resilver priority that can be set and changed live per pool, with a client-latency target rather than a delay knob.

TRIM and discard through zvols

BlueStore discard on the zvol, mapped to freed space in the pool, so that dRAID's sequential resilver and autotrim both benefit.

SUMMARY

ZFS fills the gaps needed to deploy Ceph at scale

1. Past a thousand 24 TB drives, backfill is a continuous background process, and every restriction tied to it is permanent: no reboots, no upgrades, no scrubs on the PGs that need them most.
2. dRAID under the OSDs turns single-drive failure into a local event measured in hours that Ceph never sees. The Ceph layer stays clean, and the lockdown never starts.
3. Ceph operational complexity is massively reduced and scalability increases to over 10K OSDs per cluster (300PB+) without a bunch of specialized configuration settings.
4. It costs CPU, write amplification and some latency, but the durability goes from 6x 9s to over 12x 9s for about 12% overhead.

Questions