45Drives ENTERPRISE

Closing the Gap: Does Ceph Tentacle Make EC Viable for VM Storage and more?

A Deep Dive into Ceph fast_ec Performance on NVMe — Squid vs Tentacle

Mitch Hall | Chief Architect, 45Drives

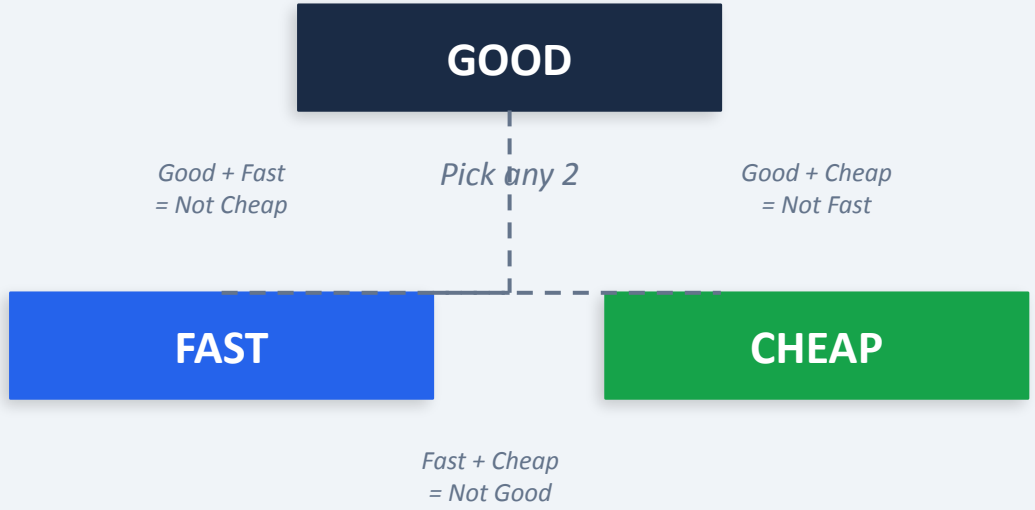
Ceph Squid (v19.2.3) / Ceph Tentacle (v20.2.0) · 6-Node Stornado F16 NVMe Cluster · 2026

The Architect's Dilemma

Balancing performance, redundancy, and cost in 2026

45Drives ENTERPRISE

- Every architect lives in this triangle — pick any two
- In 2026, that balance is harder than it has ever been
- Witnessed AI compute bubble Double RAM/Flash prices in a single week
- 3-replica NVMe storage is becoming genuinely prohibitive at scale



Erasure Coding offers a potential path out.

- Already proven for RGW object storage & CephFS sequential workloads
- Block storage — VM boot drives, databases — always out of reach due to write latency
- fast_ec in Ceph Tentacle changes the equation. But does it change it enough?

fast_ec: Can we finally have all three for block workloads?

Why Erasure Coding for Block Storage?

The capacity efficiency argument

45Drives ENTERPRISE

Usable capacity comparison — 92 TB raw across 6 nodes

3-Replica

~30 TB

usable capacity

3× overhead

33% storage efficiency

Standard HA — every byte written 3 times

4+2 EC

~61 TB

usable capacity

1.5× overhead

67% storage efficiency

4 data + 2 parity shards — survives 2 OSD failures

2+2 EC

~46 TB

usable capacity

2× overhead

50% storage efficiency

2 data + 2 parity shards — survives 2 OSD failures

4+2 EC delivers 2× more usable capacity than 3-Replica from the same hardware — potentially halving $/TB for customers

The Problem: EC in Ceph

Why erasure coding has historically been off-limits for block workloads

45Drives ENTERPRISE

01 Full-Stripe Read-Modify-Write

In previous versions of Ceph, any write smaller than a full stripe triggers a read of the entire stripe, merges in the new data, recalculates all parity shards, then writes the full stripe back. For 4K random writes — the dominant VM workload — this is catastrophically wasteful. A single 4K client write forces reads and writes across all K+M OSDs.

02 Whole-Stripe Reads for Small IOs

Reads in previous version of Ceph always fetch the entire stripe regardless of how little data the client actually needs. For a 4K read in a 4+2 pool with a 4K stripe unit, previous version of Ceph reads 24K of data across 6 OSDs to satisfy a 4K request — 6× IO amplification. The higher the K value, the worse the waste.

03 Small Objects Padded to Full Stripe

Objects smaller than the full stripe width are padded out to fill the entire stripe before writing. This wastes both storage capacity and IOPS, since data is written to more OSDs than the object size requires — compounding the write penalty for typical virtualization workloads.

04 IO Amplification Scales with K

In previous versions of Ceph, the partial-write path the IO cost scales linearly with K. A 4+2 pool requires reading 3 unchanged data strips per partial write. A 6+2 pool requires reading 5. This made larger K values (which offer better capacity efficiency) actively harmful for latency.

Bottom line in previous version of Ceph: EC pools are optimised for large sequential workloads. For 4K random IO — the heartbeat of any virtualisation environment — the overhead is too severe for production block storage.

How fast_ec Fixes It: Tentacle Internals

Three fundamental optimisations that change the economics of EC block storage

45Drives ENTERPRISE

I Partial Reads

Read only what the client needs

In previous versions of Ceph, reads always fetch the entire stripe. In Tentacle, partial reads honour the exact client request — a 4K read touches only the single OSD holding that strip. IO amplification from large K values is eliminated. This also enables larger stripe units to be used without the read penalty that made them impractical before.

II Partial Writes

Write only the strips that changed

Instead of reading the whole stripe to merge a small write, fast_ec reads only the data strips not being overwritten, recalculates parity, and writes back only the modified strips. For a 4+2 pool, a sub-stripe write goes from touching all 6 OSDs to touching only the affected data strip plus 2 parity OSDs.

III Parity Delta Writes (PDW)

XOR-based parity update without full re-encode

PDW reads the old data, XORs it with the new data to produce a delta, then applies that delta to each parity shard directly — without needing to re-read all unchanged data strips. For a 4+2 pool this costs only 3 reads + 3 writes regardless of K. Fast_ec dynamically selects PDW vs partial write per IO for optimal performance.

Enabling fast_ec: Configuration & stripe_unit

The one tuning decision that matters most

Why increase stripe_unit to 16K?

Squid's 4K default was a workaround

Small stripe units existed to limit the damage from whole-stripe reads and write amplification. With partial reads and no object padding, that constraint is gone.

16K dramatically reduces IO splits

A 4K stripe unit means any IO larger than 4K gets split across multiple OSDs unnecessarily. At 16K, small IOs (4K–16K) hit a single OSD per strip — no splitting, no extra round trips.

Capacity neutral at 16K

Padding waste at 16K is minimal and comparable to the old 4K behaviour. Going above 16K (e.g. 256K) improves performance further but wastes more capacity for small objects — 16K is the recommended sweet spot.

Cannot be changed on existing pools

stripe_unit is baked into the erasure code profile at pool creation. Existing Squid pools cannot be migrated — fast_ec can still be enabled on them, but with a smaller performance gain.

Enabling fast_ec (per pool)

```
ceph osd pool set <pool> allow_ec_optimizations 1
```

Creating a new 4+2 pool with 16K stripe unit:

```
ceph osd erasure-code-profile set fast42 \
  k=4 m=2 stripe_unit=16384
ceph osd pool create rbd_ec_42 erasure fast42
ceph osd pool set rbd_ec_42 allow_ec_optimizations 1
```

⚠️ allow_ec_optimizations cannot be disabled once set. Test on non-production pools first.

This study's configuration

Phase 1 (Squid):	stripe_unit=4K (default), fast_ec disabled
Phase 2 (Tentacle):	stripe_unit=16K, allow_ec_optimizations=1
2+2 EC pool:	Included after promising early fast_ec results at 2+2

Test Cluster: 45Drives Stornado F16

6-node all-NVMe configuration

45Drives ENTERPRISE

COMPUTE & STORAGE

Server Model	45Drives Stornado F16
Nodes	6
CPU (per node)	AMD EPYC 8534P — 64c / 128t
RAM (per node)	256 GB DDR5 ECC
NVMe Drives	16× Micron 7450 Pro 960 GB U.3 (Gen 4)
HBA	LSI 9600-16i
Raw / Node	15.35 TB
Total Raw	~92 TB

384
CPU Cores

1.5 TB
Total RAM

92 TB
Raw NVMe

200 Gb/s
Client Net

NETWORK

2× 100 GbE LACP | Mellanox ConnectX-5 100G | MTU 9000 | L3+4 bond | mlnx_tune low_latency_vma

Software Stack

Cluster OS, Ceph version, and client environment

45Drives ENTERPRISE

CEPH CLUSTER

OS	Rocky Linux 9.7
Kernel	5.14.0-611.36.1.el9_7
Ceph Release	Squid(v19.2.3)/Tentacle(20.2.0)
Orchestrator	cephadm
Network	LACP 2x100GbE
Compression	None

CLIENT / PROXMOX

Proxmox VE	9.1.2
VM OS	Rocky Linux 9.7
Kernel	5.14.0-611.36.1.el9_7
RBD Access	Kernel rbd map
VM vCPUs	8 vCPU
VM RAM	8 GB

BENCHMARK TOOLING

fio	3.35
Ansible	Automated playbooks
Results	JSON — 3 passes averaged
Plotter	Python / matplotlib
Output	PNG graphs
RBD Image Size	500 GiB each

Ceph Pool Configuration

Three pools benchmarked across both phases

45Drives ENTERPRISE

3-Replica

rbd

PG Count:	1024
CRUSH Rule:	replicated_rule
Overhead:	3× (33% efficiency)
Usable:	~30 TB
Failure Tolerance:	2 OSD failures

Standard Ceph HA — baseline for all comparisons

4+2 Erasure Code

rbd_ec_42

PG Count:	1024
EC Plugin:	jerasure
Overhead:	1.5× (67% efficiency)
Usable:	~61 TB
Failure Tolerance:	2 OSD failures

Maximum capacity efficiency — the primary EC target

2+2 Erasure Code

rbd_ec_22

PG Count:	1024
EC Plugin:	jerasure
Overhead:	2× (50% efficiency)
Usable:	~46 TB
Failure Tolerance:	2 OSD failures

Balanced EC — included after promising early fast_ec results

Phase 2 adds: fast_ec enabled (allow_ec_optimizations=1) · EC stripe unit increased 4K ⟶ 16K · Tentacle topology

Test Methodology

Automated benchmarking with Ansible + fio

45Drives ENTERPRISE

01 Block Size Latency Sweep

```
Single VM, QD=1, 1 job
4K / 16K / 64K / 128K
Rand read + rand write
3 passes averaged
```

02 Response Curve (Latency)

```
1→16 VMs, +1 per step
4K, QD=1, 1 job per VM
Rand read + rand write
3 passes averaged
```

03 IOPS vs Latency Curve

```
1→16 VMs, +1 per step
4K, QD=32, numjobs=4
Plots total IOPS vs P50 latency
3 passes averaged
```

04 Sequential Throughput

```
1→16 VMs, +1 per step
1MB, QD=16, numjobs=2
Seq read + seq write
MB/s or GB/s per pool
3 passes averaged
```

Fully automated: Custom Ansible playbooks orchestrate all fio jobs across 16 VMs simultaneously. Results collected as JSON, averaged across 3 passes, and plotted with a custom Python/matplotlib pipeline.

How We Tested: Automation Pipeline

Ansible + Python — zero manual data entry

Ansible Playbooks

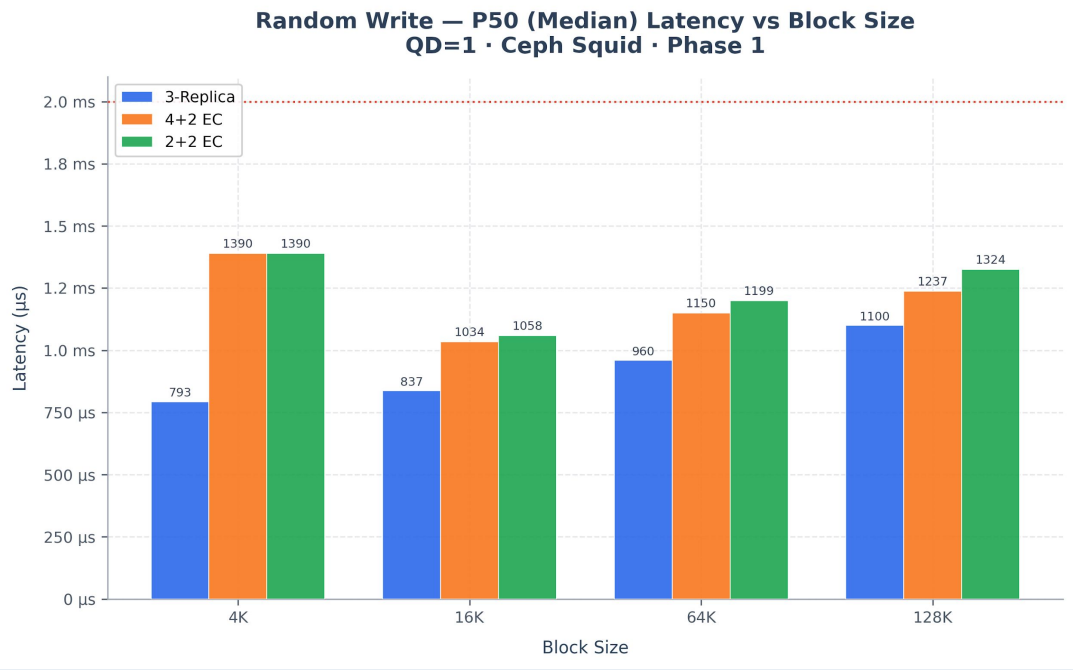
- benchmark_blocksize_sweep.yml — runs 8 fio jobs (4 BS × 2 rw), 3 runs each, on a single VM
- benchmark_response_curve.yml — sweeps 1→16 VMs simultaneously, 3 passes, collecting per-step JSON
- benchmark_iops_latency.yml — high QD/numjobs sweep to find IOPS saturation knee sweeps 1→16 VMs simultaneously, 3 passes, collecting per-step JSON
- benchmark_throughput.yml — 1MB sequential sweep across all pools 1→16 VMs simultaneously
- Auto pass detection — playbook re-runs increment pass1 → pass2 → pass3 automatically, never overwriting results
- Results fetched to control node via Ansible fetch module — organised by phase/pool/pass/step

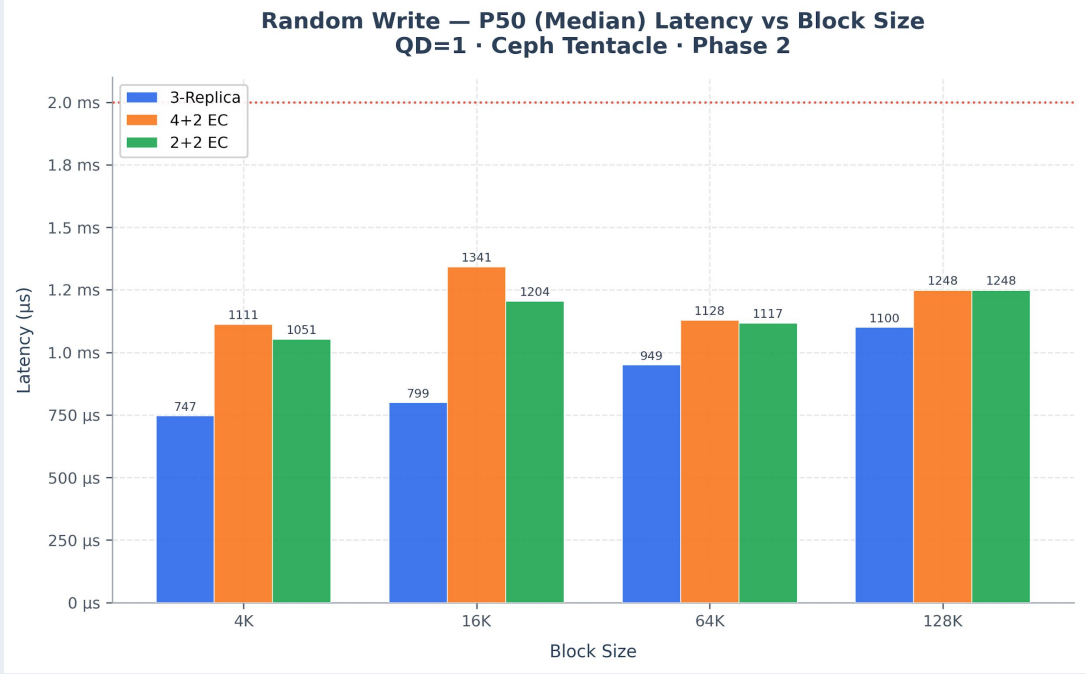
Python Plotting Pipeline

- plot_results.py — single CLI tool, 4 subcommands: blocksize, responsecurve, iopslatency, throughput
- Parses fio JSON — reads clat_ns percentile arrays directly, averages P50/P99 across all passes – identifies any abnormal variance for closer look
- blocksize — grouped bar charts, all 3 pools side by side per block size
- responsecurve — latency vs VM count, with 2ms threshold line and shaded danger zone
- iopslatency — total IOPS (X) vs P50 latency (Y) — shows saturation knee per pool

Block Size Latency — Random Write

Phase 1/2 — Ceph Squid/Tentacle · P50 · QD=1 · Single VM · 3-Replica vs 4+2 EC vs 2+2 EC

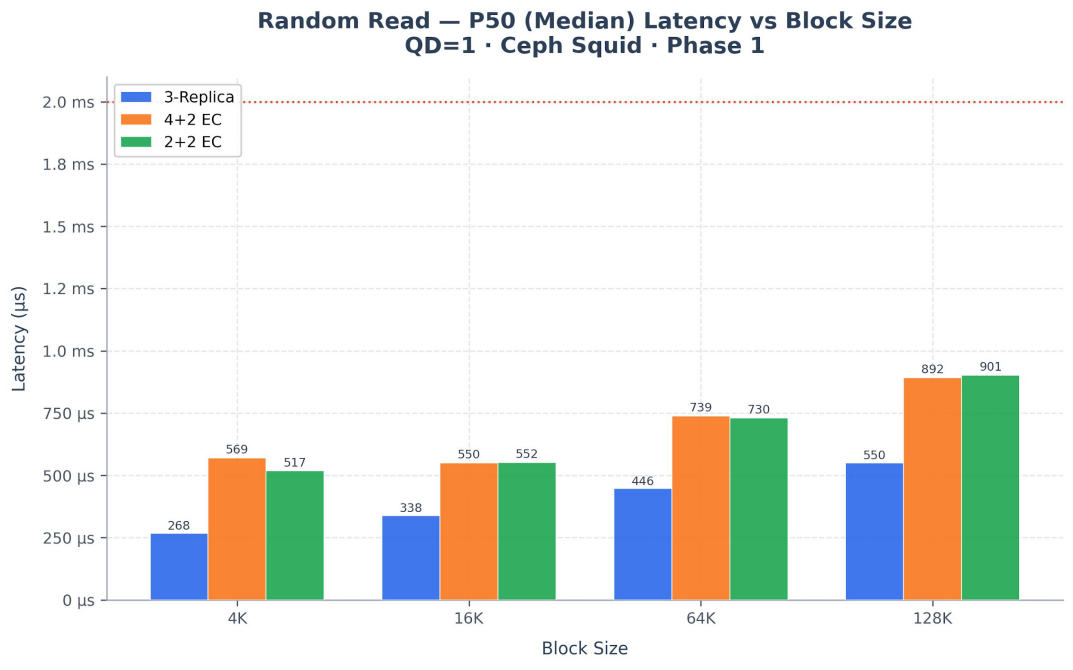


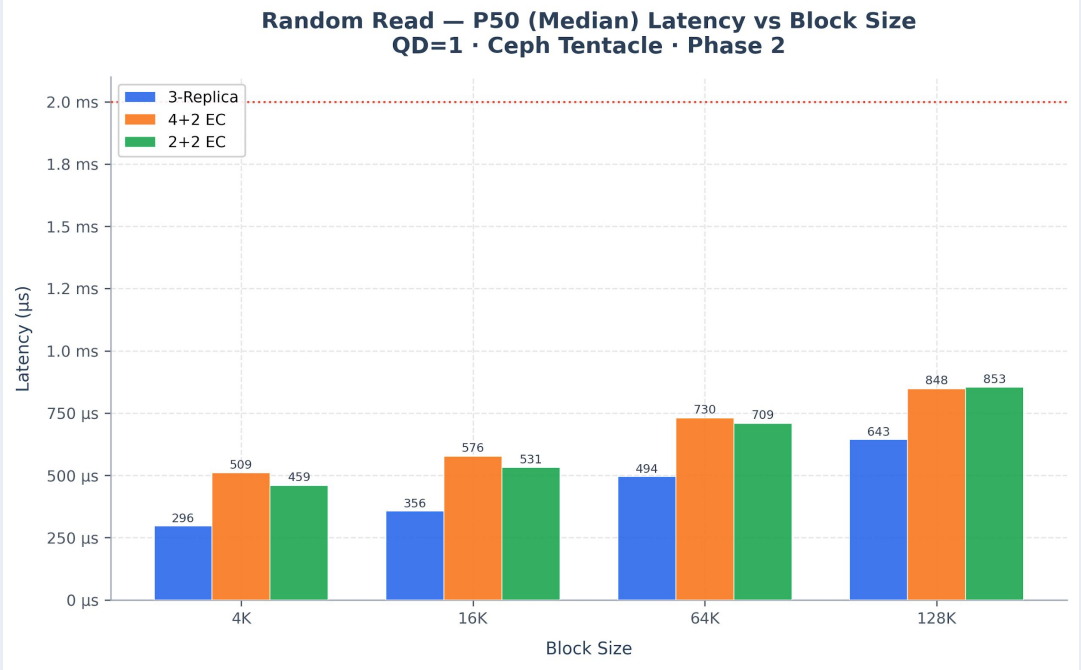


Finding: Key finding: EC write penalty visible at all block sizes. Small improvements in Tentacle – 16k is interesting.

Block Size Latency — Random Read

Phase 1/2 Ceph Squid/Tentacle · P50 & P99 · QD=1 · Single VM · 3-Replica vs 4+2 EC vs 2+2 EC

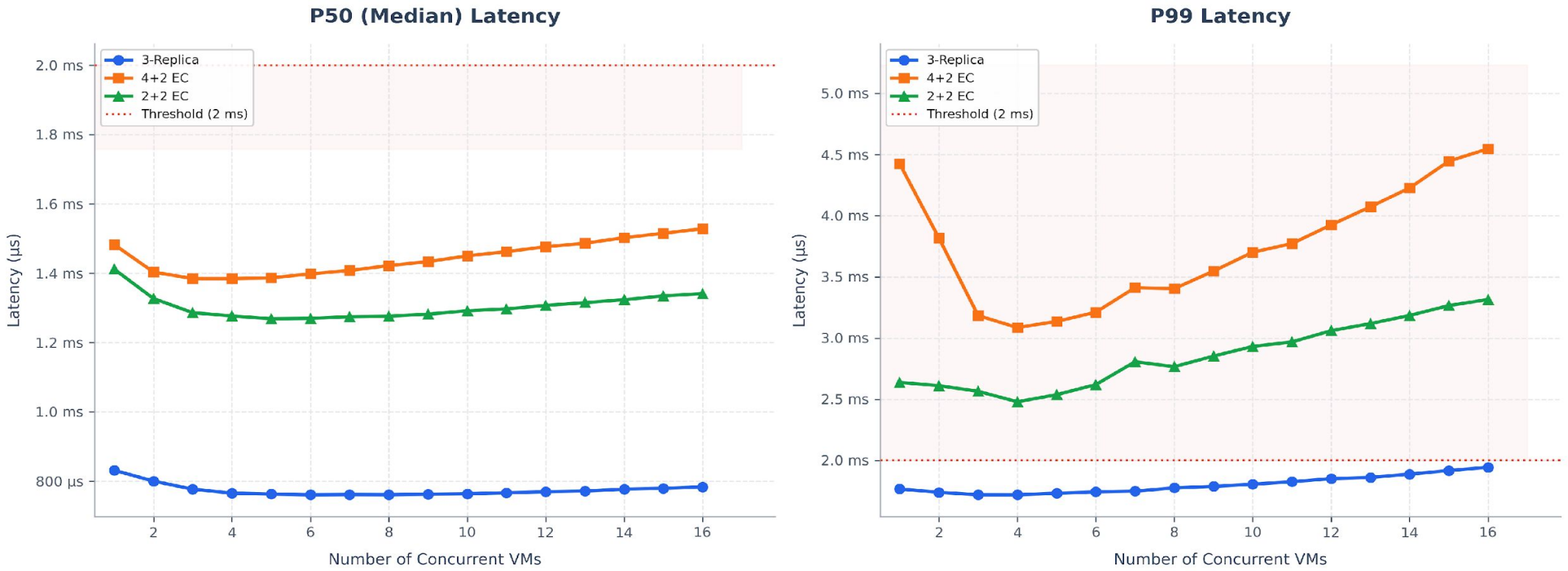




Finding: Key finding: Read latency gap between pools is smaller — EC recovers better on reads.

Response Curve - Random Write

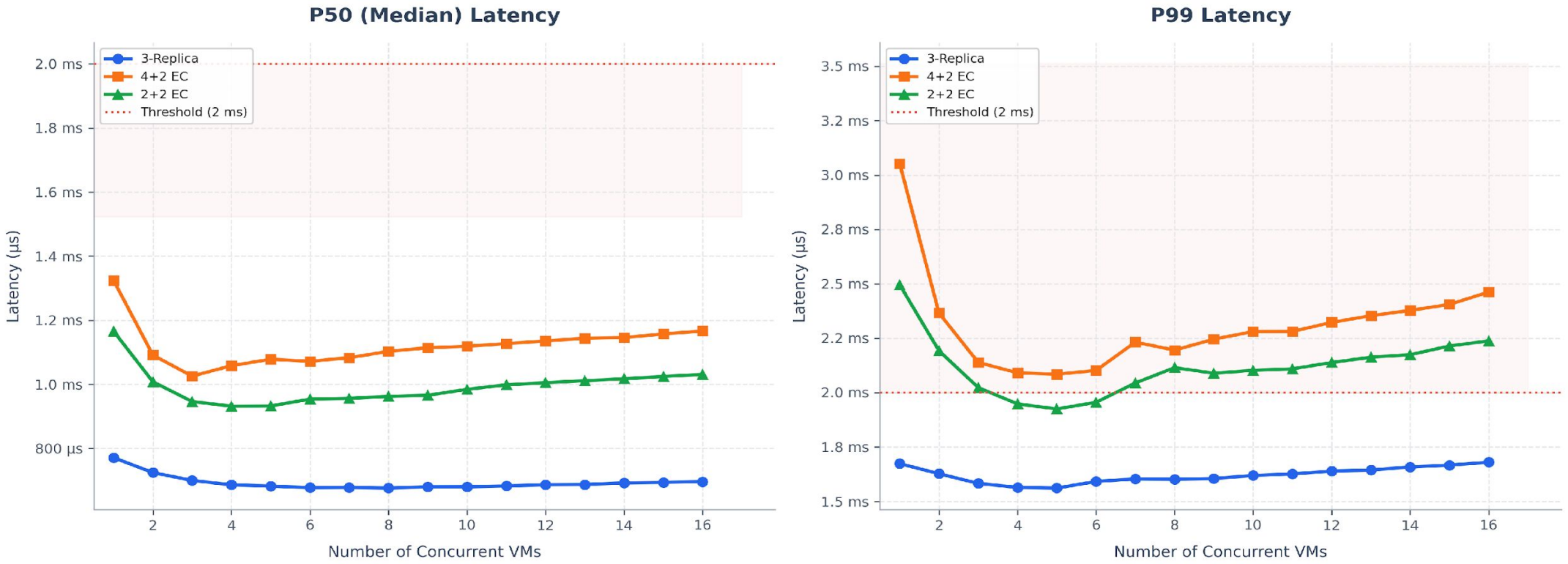
Phase 1 — Ceph Squid · P50 & P99 Latency · 4K QD=1 · 1→16 Concurrent VMs · 3 passes averaged



Finding: 3-Replica stays under 2ms P99 at 16 VMs. EC pools remain above threshold at P99.

Response Curve - Random Write

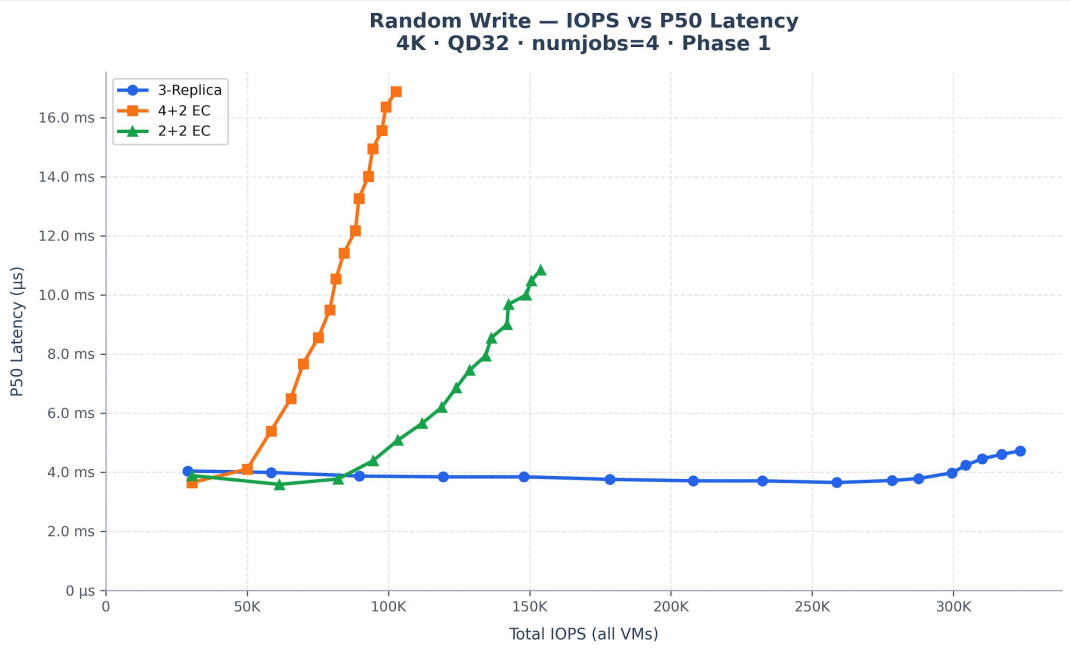
Phase 2 — Ceph Tentacle + fast_ec ·P50 & P99 Latency · 4K QD=1 · 1⟶16 Concurrent VMs · 3 passes averaged

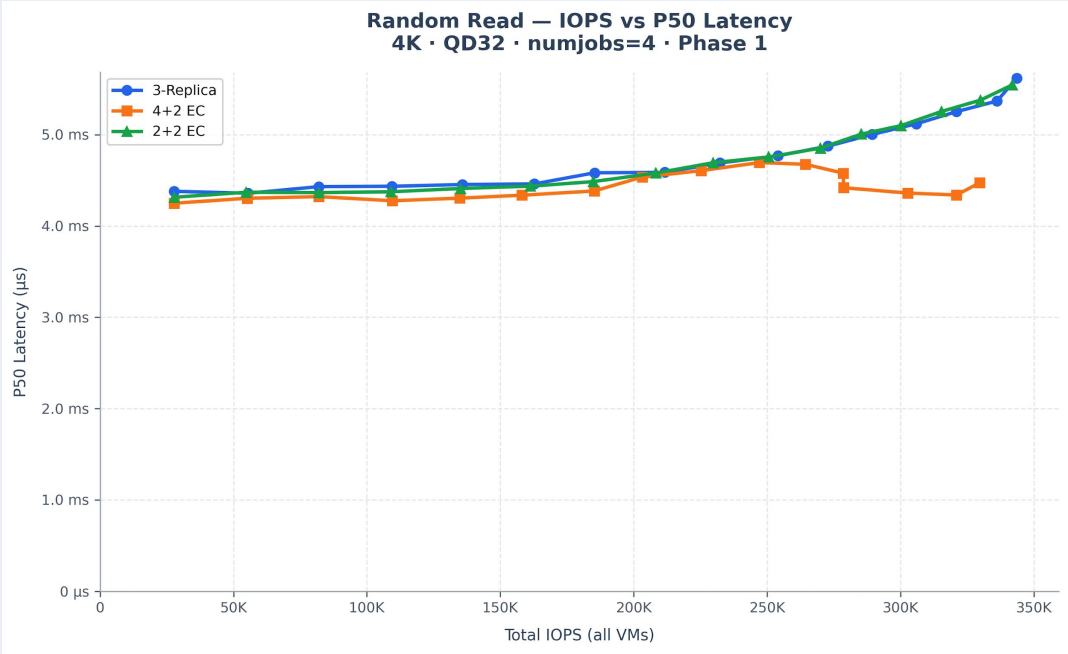


Finding: THE key comparison: does fast_ec bring EC latency curves close enough to 3-replica to be viable for block workloads?

IOPS vs Latency Curve

Phase 1 — Ceph Squid · 4K · QD=32 · numjobs=4 · Total IOPS vs P50 Latency · All 3 Pools

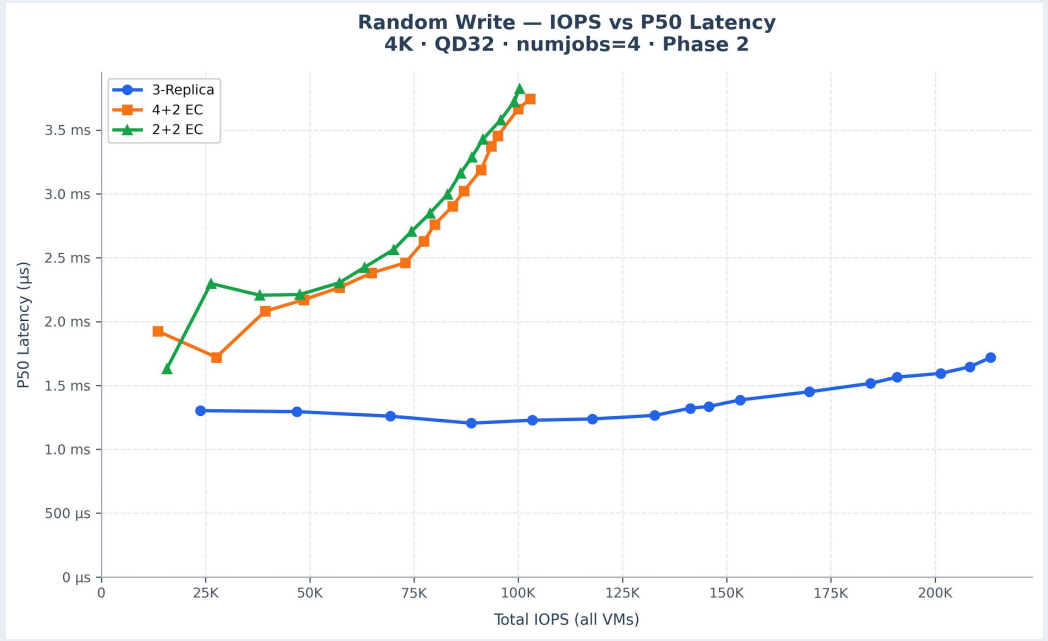


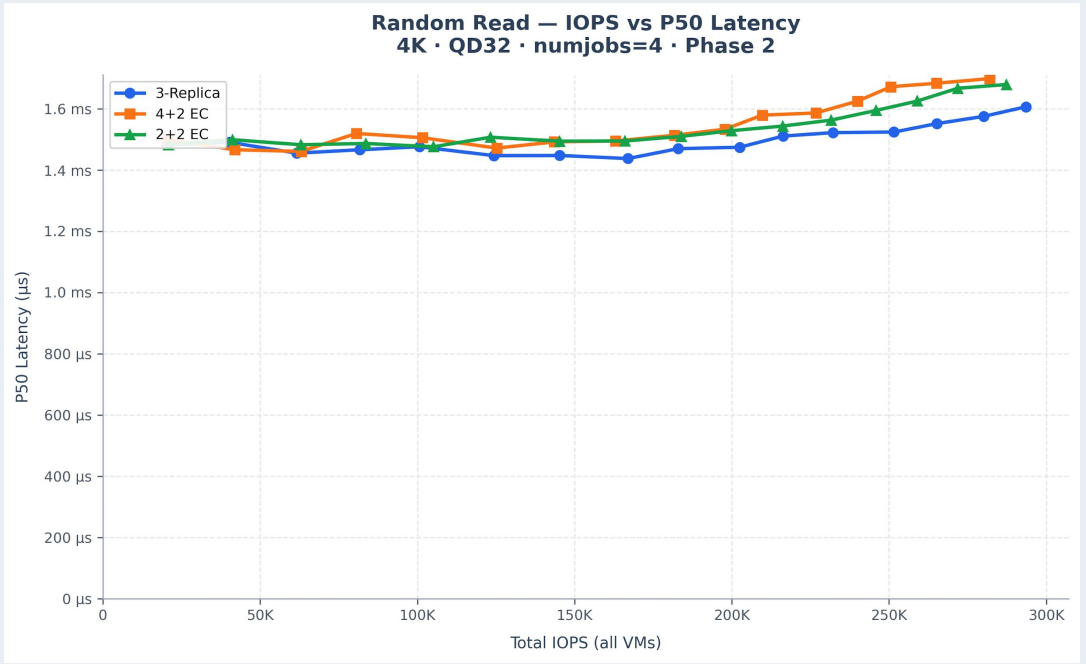


Finding: Shows IOPS saturation knee — where pushing more IOPS causes latency to spike. 3-rep reaches ~320K IOPS before degrading.

IOPS vs Latency Curve

Phase 2 — Ceph Tentacle · 4K · QD=32 · numjobs=4 · Total IOPS vs P50 Latency · All 3 Pools

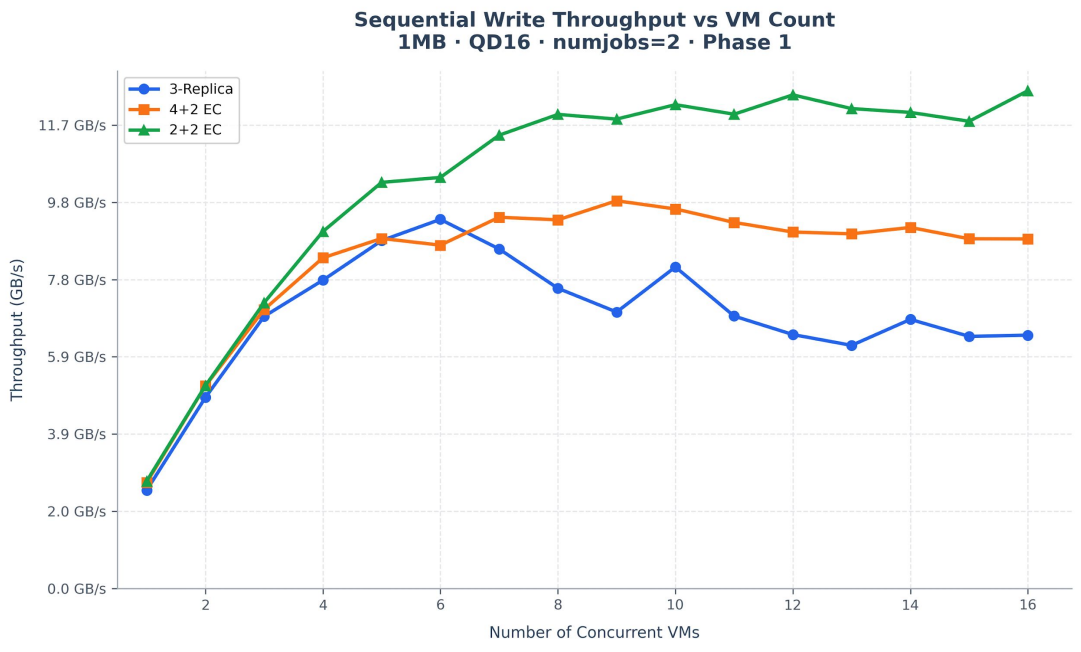


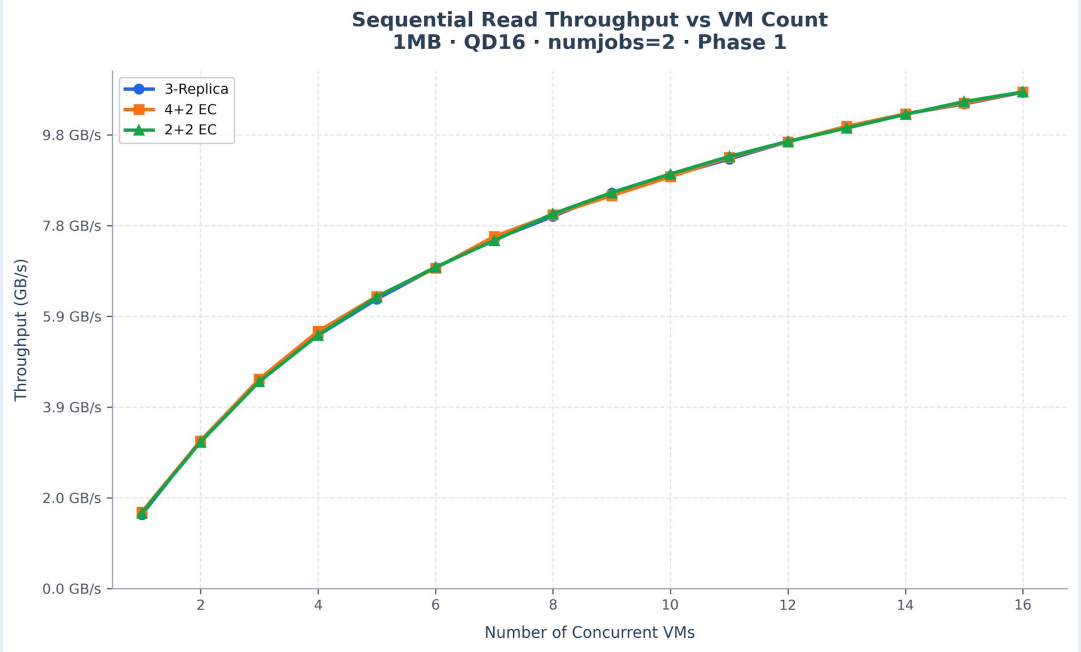


Finding: Shows IOPS saturation knee — 4X reduction of latency for 4+2 at same iops – read iops latency highly improved.

Sequential Throughput

Phase 1 — Ceph Squid · 1MB · QD=16 · numjobs=2 · 1→16 VMs · MB/s or GB/s

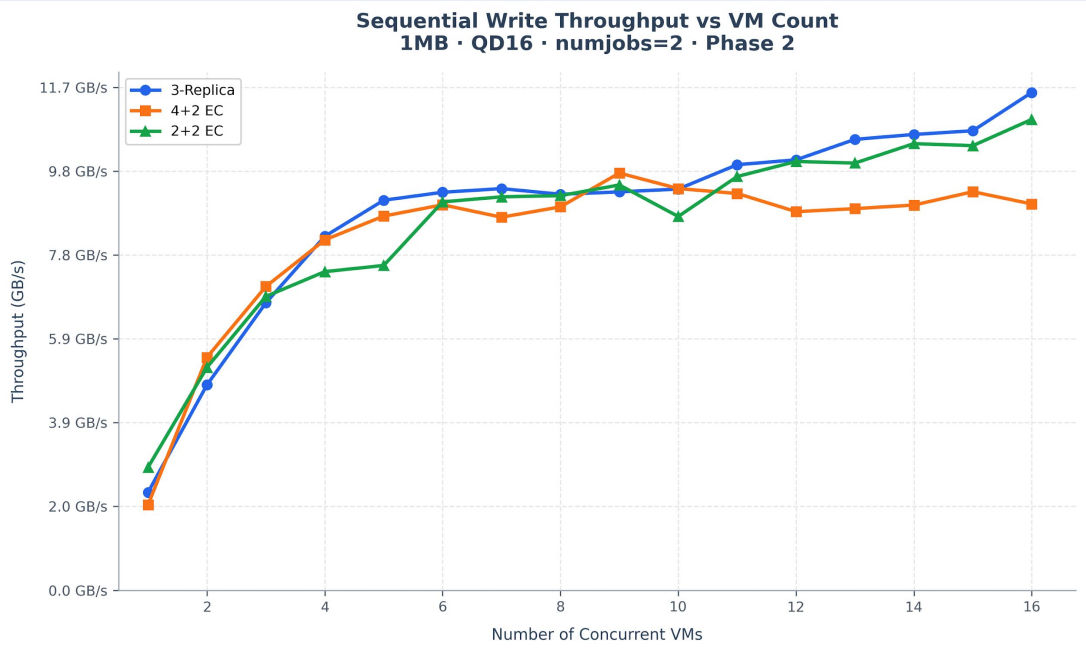


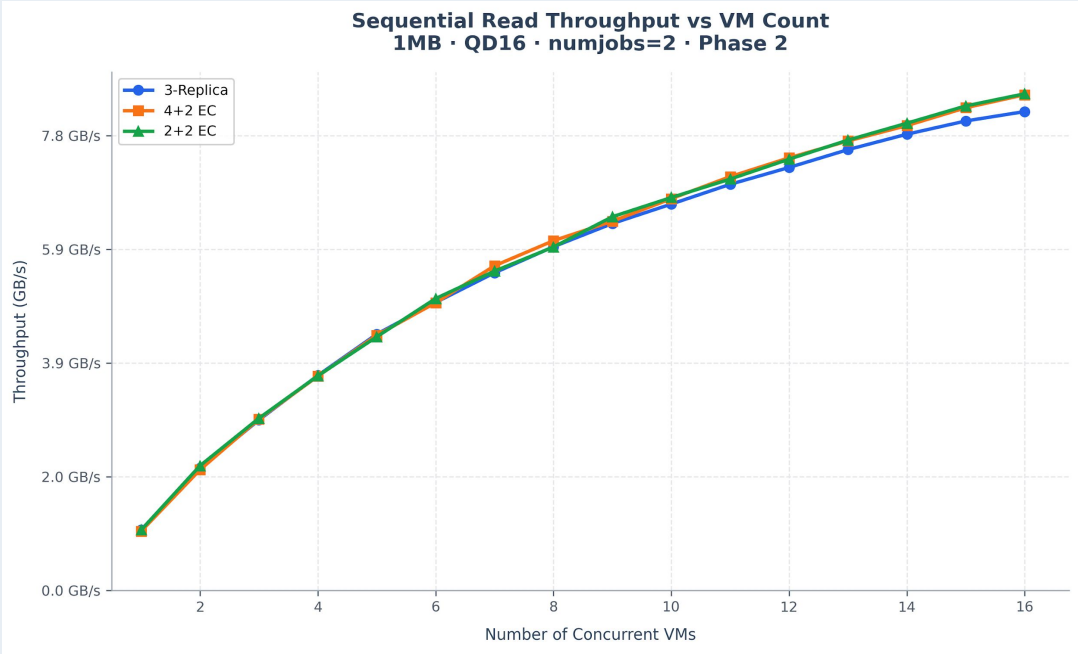


Finding: EC outperforms 3-replica here — large sequential IO is where EC shines.

Sequential Throughput

Phase 2 — Ceph Tentacle · 1MB · QD=16 · numjobs=2 · 1→16 VMs · MB/s or GB/s





Finding: replication improved greatly to be on par or better than EC – reads suffered across the board

What fast_ec Unlocks: Block & Virtualization

Workloads where the latency tradeoff now makes economic sense

45Drives ENTERPRISE

01 Dev / Test VM Fleets

Latency-tolerant by nature — no SLA depends on a dev VM booting 50ms faster. Organisations running dozens of dev/test VMs on 3-replica are paying 3× storage overhead needlessly. Switch to EC and cut NVMe footprint nearly in half with no user-visible impact.

02 CI/CD Build Infrastructure

Jenkins, GitLab runners, self-hosted GitHub Actions — IO-intensive but not microsecond-latency-critical. A build job that takes 4 minutes doesn't notice if individual IOs are 1ms vs 2ms. A massively over-provisioned segment of enterprise Ceph deployments.

03 Virtual Desktop Infrastructure (VDI)

Users don't perceive latency differences below ~5-10ms at the application level. With EC P99 now approaching that threshold under realistic concurrent load, hundreds of virtual desktops at half the storage cost becomes a genuine architectural option.

04 Database Read Replicas

Primary databases still want 3-replica for write predictability. But read replicas — MySQL secondaries, PostgreSQL standbys, Elasticsearch data nodes — are predominantly reads. Your phase 2 read data shows EC excelling here. Split-tier: primaries on 3-rep, replicas on EC.

05 Kubernetes Persistent Volumes

Container PVCs for message queues, log aggregation, Prometheus, InfluxDB — write-heavy but not latency-critical. Previously provisioned on 3-replica just to be safe. fast_ec may now be viable for stateless and observability workloads.

06 Warm Backup & Rapid Restore

EC has always been fine for cold backup. fast_ec enables warm backup storage where restores are practical. The read latency improvement means recovery time objectives are now achievable from an EC pool — combine the capacity savings with genuine restore speed.

What fast_ec Unlocks: Beyond Block Storage

The improvements ripple across every Ceph workload type

45Drives ENTERPRISE

CephFS Home Directories & Shared Storage

EC on CephFS data pools has been production-viable for a while, but small file workload performance anxiety kept some shops on 3-replica. For sequential throughput data — showing EC matching 3 replica combined with the latency improvements at small block — gives administrators the hard evidence to finally make the switch. For universities or enterprises with petabytes of home directory data, research data and more - the capacity savings are transformational.

S3 / RGW with Mixed Workload Patterns

The fast_ec improvements specifically benefit small objects and random-access reads on RGW. Combined with dramatically improved read latency, RGW deployments with active small-object workloads — application assets, thumbnails, ML training data with random access patterns — are now far more viable on EC now. More capacity, better random read performance.

Higher K Values Now Practical

In previous versions of Ceph, larger K values (6+2, 8+2) were actively harmful — IO amplification scaled linearly with K, making high-efficiency profiles reserved for very cold data. With PDW and partial reads, IO cost is now fixed at 3R+3W regardless of K. This unlocks 6+2 and 8+2 profiles — storage efficiencies of 75-80% — for workloads that previously could only justify 4+2.

The honest framing: fast_ec doesn't eliminate the write latency gap — it narrows it significantly. 3-replica will likely always win a pure latency contest. The right question is no longer 'can we use EC?' but 'for which workloads is this tradeoff worth the capacity savings?'

TCO Impact for 45Drives Customers

What viable EC means in dollars per usable TB

If fast_ec enables EC for block workloads, the capacity efficiency gains translate directly to customer savings:

Same Hardware, More Storage

2×

usable capacity
with 4+2 EC vs 3-Replica

Fewer NVMe and less RAM Needed

−50%

:ess NVMes needed to achieve
same usable capacity

Power & Rack Savings

−50%

power, cooling, and
rack space costs

NVMe Cost at Scale

½ $/TB

effective cost per
usable TB (approx.)

Real-world context: In today's flash market, a customer who can replace a 3-replica NVMe deployment with an EC-backed one saves not just on drives — as the cluster scales larger, they save on servers, rack units, power, and cooling. For a 500 TB usable deployment, this could represent hundreds of thousands of dollars in capital expenditure. fast_ec makes this conversation possible for the first time for latency-sensitive workloads.

45Drives ENTERPRISE

Conclusion & Next Steps

1 EC has a quantifiable latency penalty in Ceph Squid — this study puts real numbers on what was previously understood qualitatively.

---

2 fast_ec (Tentacle) represents the most significant advance in Ceph EC performance for block workloads to date.

---

3 Phase 2 data shows *some* virtual/latency-based workloads are now viable to run on EC. This means that in the future, 45Drives can offer EC-backed block storage to virtualization customers — fundamentally improving TCO.

---

4 The benchmarking methodology (Ansible + Python pipeline) is fully reproducible and will be published for community validation.

---

5 This is just the beginning. Follow our journey in validating erasure coding in Tentacle for new workloads over on our YouTube channel at youtube.com/45drives

Questions?

mhall@45drives.com · 45drives.com · github.com/45drives