45Drives ENTERPRISE

ceph

45Drives High Performance Ceph Reference Architecture

Table Of Contents

Chapter 1: Executive Summary

Chapter 2: **Network Configurations**
- 2.0 Network Configuration
- 2.1 Traffic Separation: Public and Cluster Networks
- 2.2 Capacity Planning
- 2.3 Latency Objectives
- 2.4 Physical Topology and Redundancy
- 2.5 Quality of Service and Flow Control
- 2.6 Host-Level Optimisation
- 2.7 Reference Implementation Summary

Chapter 3: Reference Architecture Design
- 3.0 Reference Architecture Design
- 3.1 Ceph Cluster Nodes – 45Drives Stornado F16 NVMe
- 3.2 Access & Metadata Layer – 45Drives Ceph Gateway Servers
- 3.3 Ceph Software Architecture & QoS Policy
  - 3.3.2 mClock Scheduler Policy
  - 3.3.3 OSD Memory & BlueStore
  - 3.3.4 Pools, CRUSH, and PG Sizing
  - 3.3.5 Client Path (RBD)
  - 3.3.6 Why These Choices
- 3.4 Performance & Resiliency Highlights
- 3.5 Why This Matters

Chapter 4: The Test Environment
- 4.0 Test Environment
- 4.1 Physical Client Hosts
- 4.2 Benchmark VM Topology
- 4.3 Block & File Layout
- 4.4 Network & System Validation
- 4.5 Benchmark Orchestration & Methodology

Chapter 5: Performance Results
- 5.0 Performance Results
- 5.1 Headline Numbers (Fully Tuned, 6-Node Cluster)
- 5.2 Single-Client Baselines
- 5.3 Multi-Client Scale & Saturation (36 VMs)
- 5.4 Observations & Interpretation

Chapter 6: The Real World
- 6.0 Viewing these numbers in real-world terms

Appendix B – Ceph configuration commands completed
Appendix B – Benchmark Command Suite (60 s ramp, 300 s run)
- B.1 Pre-fill (required for thin-provisioned RBD)
- B.2 Single-client, single-thread (60 s ramp, 300 s run)
- B.3 Single-client, multi-thread (numjobs=4; 60 s ramp, 300 s run)
- B.4 Notes

Executive Summary

As real world business workloads continue to scale the demand for ever higher tiers of performance while maintaining low latency all while ideally having near perfect scaling as more workloads and threads come online, businesses need to look to storage architectures that can meet these demands and continue to scale in performance alongside capacity. Ceph is highly unique in this regard, as it is able to continue to scale with great linearity until saturating the entire IOPS/throughput capability of the cluster, and once this happens, we can simply add additional nodes/OSDs to begin the scaling journey once more.

This paper aims to dive into the installation and best practices of a high-performance NVMe Ceph cluster, from network design and tuning, to hardware choices to get the most out of the server, to system OS level tunings to maintain the lowest latency possible. Ceph will live and die by its latency. The better you can tune for lower system and network latency, the more performant and responsive your cluster will be. Ceph is an infinitely scalable open-source software-defined-storage solution that provides native block, object and file system within a single unified Ceph cluster.

This paper will emphasize real world design principles and practices alongside real-world workloads that are best suited for this architecture. Benchmarks will show how much performance a single client can generate in a single threaded and multithreaded workload, and we will focus in on how well the solution will scale as more and more workloads come online in this topology.

In aggregate, the baseline cluster offers **96 NVMe OSDs, 384 CPU cores, 1.5 TiB RAM, and 1.2 Tb s^{-1} of east–west network bandwidth**, all under a single Cephadm-managed control plane. Because Ceph's CRUSH algorithm distributes data placement intelligently and eliminates central bottlenecks, performance scales near-linearly: when the cluster hits its IOPS or throughput ceiling, operators simply add more NVMe OSDs or drop in additional nodes and once the new OSDs are detected, Ceph begins it rebalancing its operations to migrate existing data onto the new nodes to keep the data as evenly distributed as possible. Our reference architecture is a fully open platform, with no licencing fees or proprietary software.

This white paper walks through the critical factors that let the hardware shine:

- **Network architecture** – queue-depth-aware LACP tuning, NIC buffer tuning, Mellanox affinity tuning for low latency, flow-hash selection, DCTCP/ECN, and host-side NUMA pinning to keep 100 GbE links saturated without retransmits.
- **OS/firmware optimization** – BIOS settings (SR-IOV off, c-states locked, SMT on), hugepages, tuned profiles, and disabling CPU sleep states that shave latencies under mixed R/W IOPS workloads.
- **Disk level tuning** – Using UBM backplane with NVMe, we will change the default IO scheduler to let Ceph make best decisions on IO scheduling.
- **Ceph-design choices** – Deploying Ceph via cephadm we will discuss all ceph configuration choices such as Ceph version, OSD memory target, PG count, replicated vs. erasure-coded pools, compression and more.

Finally, we benchmark single-client and multi-client scenarios—both single-threaded and multi-threaded—to illustrate how the reference architecture maintains relatively low latency while driving millions of IOPS and saturating 100 GbE links. Real-world traces (MySQL, HPC, ML, VDI, and 4K UHD production) validate that performance, resilience, and cost transparency can coexist in an open-source SDS platform.

Chapter 2: Network Configurations

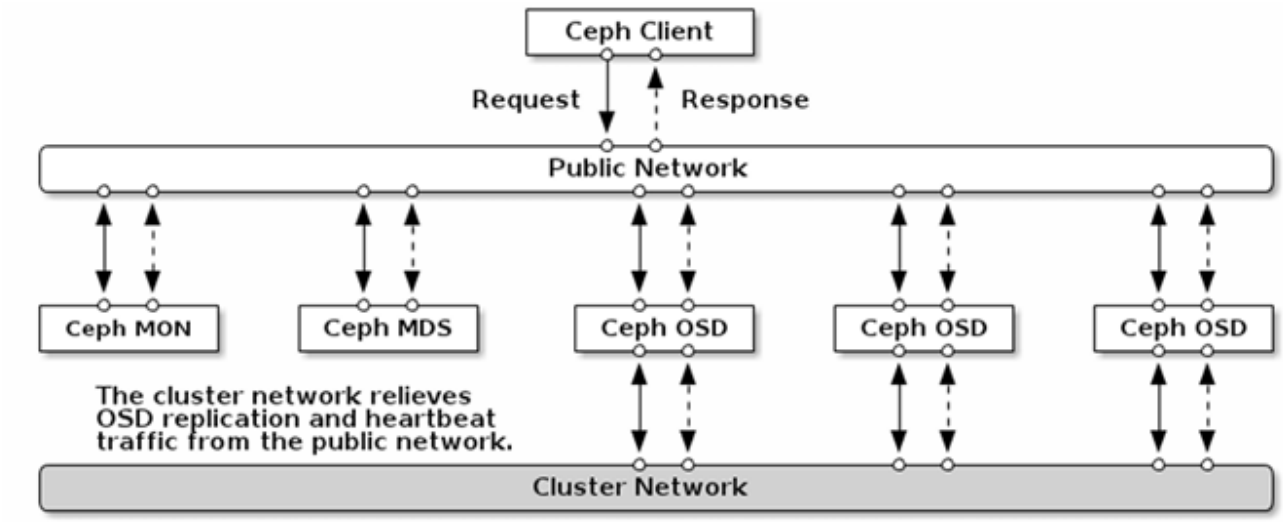
2.0 Network Configuration

A Ceph cluster built on NVMe media will expose the network as its principal bottleneck: every write is acknowledged only after all replicas are durably written, so round-trip latency between OSDs sets the floor on application response time. The purpose of this section is to outline the architectural principles and concrete implementation targets required to ensure that the network sustains the performance envelope of an all-NVMe Ceph deployment.

2.1 Traffic Separation: Public and Cluster Networks

Ceph distinguishes two logical fabrics:

Network	Primary Function	Participants
Public	Client I/O, monitor traffic	Clients, MONs, MGRs, RGWs, MDSs, OSDs
Cluster	OSD replication, heartbeats, recovery	OSDs only



2.2 Capacity Planning

Because each client write is replicated (replication factor X3) times inside the cluster, east–west traffic inherently exceeds north–south traffic. In steady state a 3× replicated pool produces approximately twice the bandwidth on the cluster network as on the public network; during recovery or back-fill the multiplier rises further. The recommended provisioning rule is therefore:

Cluster network aggregate bandwidth = 2 – 4 × public network aggregate bandwidth per node.

Example
Front-end target: 200 Gb/s
Replication: 3×
Provisioned cluster fabric: 200 – 400 Gb/s per storage node

This envelope accommodates peak replication, rebalancing, and recovery loads without throttling client throughput.

2.3 Latency Objectives

Network-only latency target

In modern NVMe media the median device write latency is ≈ 40 µs. This means, in a network replicated storage solution with ultra fast storage media, the goal to keep performance at peak is to ensure the network latency is tuned to be as low as possible:

Host↔host round-trip latency across the storage fabric—NIC-to-NIC, including switch hops but excluding the storage stack—should land in the 20–40 µs (0.02–0.04 ms) range. In order to achieve this, we will tune the systems and network heavily.

Non-blocking leaf–spine topology with a ≤ 2-hop data path:

- Strong CPU affinity mapping to network interfaces
- Consistent 9000-byte MTU across both fabrics to minimise interrupt overhead
- PTP-derived time-stamping to validate latency budgets during commissioning
- System and NIC level tunings that will be discussed in depth later

2.4 Physical Topology and Redundancy

- **Leaf–Spine Fabric** Each storage node connects to two independent leaf switches per fabric, and each leaf maintains at least dual-100 Gb/s uplinks to **redundant spines.**
- **Link Aggregation** Use LACP bonds of homogenous speed to smooth traffic distribution and provide fast fail-over.
- **Failure Domains** Place the public and cluster links on separate line cards (or separate switches where budget allows) so that a single NIC failure doesn't take down an entire LACP bond.

2.5 Quality of Service and Flow Control

- **Priority Flow Control / ECN** Enable explicit congestion notification (ECN) or DC-QCN on RoCE fabrics; enable RED/ECN on pure TCP fabrics to reduce queuing latency under burst conditions.
- **Jumbo Frames** Adopt 9000-byte MTU uniformly; mixed MTU environments introduce fragmentation penalties that erase latency gains.
- **Monitoring** Capture RTT at both the transport layer (TCP RTT) and fabric layer (telemetry on buffering, PFC pause events) to detect emerging hotspots before they affect client latency.

2.6 Host-Level Optimisation

- **IRQ and NUMA Affinity** Pin NIC interrupts and Ceph-OSD processes to CPUs closest to their attached NVMe devices, preventing costly cross-socket memory traffic.
- **Queue Depths** Scale RX/TX descriptors to at least 4096 to absorb burst replication traffic.

2.7 Reference Implementation Summary

Component	Public Fabric	Cluster Fabric
NICs per node	2 × 100 GbE (bonded)	2 × 200 GbE (bonded)
Aggregate node BW	200 Gb/s	400 Gb/s
Target RTT	≤ 20 µs	≤ 20 µs
MTU	9 000 B	9 000 B
Oversubscription	≤ 1 : 1	1 : 1

This reference profile has been validated to sustain > 90 % of raw NVMe throughput while keeping 99th-percentile network latency under 80 µs in a six-node, 3× replicated cluster even during highly active network traffic.

2.8 Key Takeaways

1. **Network and CPU/system latency now define storage latency;** design the fabric as an integral component of the storage stack.
2. **Over-provision the cluster network by a factor of 2–4** relative to the public network to prevent replication from throttling client I/O and for self-healing as well as rebalancing
3. **Maintain symmetry and isolatio**n between fabrics to preserve availability during component failures.
4. **Set and enforce latency budgets** from transceiver to daemon to guarantee network and Bluestore CPU time is the limiting factor is tuned to its maximum level.

By adhering to these principles, a Ceph cluster can exploit the full performance potential of NVMe while retaining the durability and scalability properties that underpin distributed storage.

Chapter 3: Reference Architecture Design

3.0 Reference Architecture Design

This section details every component selected for the six-node reference cluster and explains how each one contributes to the performance-per-watt, resiliency, and linear scale-out goals laid out in Sections 1 and 2.

3.1 Ceph Cluster Nodes – 45Drives Stornado F16 NVMe

This section details every component selected for the six-node reference cluster and explains how each one contributes to the performance-per-watt, resiliency, and linear scale-out goals laid out in Sections 1 and 2.

Item	Configuration	Rationale
Chassis	2U 45Drives Stornado F16, direct-wire backplane, tool-less trays	16 U.3 bays give us one NVMe per OSD with zero PCIe bottlenecks and <1 µs backplane latency. 45drives.com
Drives	16 × Micron 9400 PRO NVMe (PCIe 4.0 ×4 each)	100 µs class latency and 1.6 M IOPS/drive ensure the network (not storage) remains the ceiling.
CPU	2AMD EPYC 8534P (64 C / 128 T, Zen 4c)	High core count keeps BlueStore, RocksDB/BlueFS, and user I/O threads isolated without pinning.
Memory	256 GiB DDR5-4800, split evenly across the two NUMA nodes	Sized for ~10 GiB/OSD for BlueStore cache + 96 GiB OS/metadata head-room; the NUMA split keeps foreign-page hits <1 %.
Networking	2 × dual-port Mellanox ConnectX-5 (4 × 100 GbE per node)	Sized for ~10 GiB/OSD for BlueStore cache + 96 GiB OS/metadata head-room; the NUMA split keeps foreign-page hits <1 %.
Software	Rocky Linux 9.6 + Ceph Squid 19.2.3 (containerised via cephadm)	Latest Squid point-release is the current “active” branch and contains critical BlueStore fixes. docs.ceph.com

3.1 Ceph Cluster Nodes – 45Drives Stornado F16 NVMe Continued

Daemon layout per node:

- 16 × OSD daemons (one per NVMe, block.db on-device)
- Either 1 × MON or 1 × MGR on three of the six nodes (3 MON, 3 MGR total) to guarantee quorum even during a full chassis outage
- Crash, Prometheus exporter, and node-exporter containers pinned to non-NUMA0 cores



Stornado F16 NVMe

3.2 Access & Metadata Layer – 45Drives Ceph Gateway Servers

Item	Configuration	Role
Chassis	2 × 45Drives Ceph Gateway Servers	Provides protocol translation and metadata offload.
CPU	AMD Ryzen 9 7950X (16 C, boost 5.7 GHz)	MDS and RGW requests are latency-sensitive and largely single-threaded; high clock wins. 45drives.com
Memory	64 GiB DDR5-5600	Plenty for MDS in-memory inode cache and RGW front-end buffers.
Networking	1 × dual-port Mellanox ConnectX-5 (2 × 100 GbE)	Gives excessive bandwidth and low latency to interface with cluster nodes.
Services	*Active/Standby* Ceph-MDS pair, HA RGW and NFS-Ganesha containers	Clean separation lets storage nodes focus solely on data path.



45Drives Ceph Gateway Server

3.3 Ceph Software Architecture & QoS Policy

This section documents the Ceph-side choices that shape performance, resiliency, and test repeatability for the reference cluster.

3.3.1 Control Plane & Runtime

Area	Choice	Notes
Release	Ceph Squid 19.2.3	Current LTS line used for all results.
Orchestration	cephadm	All daemons are containerized and lifecycle-managed via the built-in orchestrator.
Container Engine	Podman	Rootless where applicable; images pinned by digest for reproducibility.
Transport / msgr	msgr2 (default)	Secure, modern messenger(async+posix); no custom tweaks required.
Daemon Layout	3× MON, 3× MGR, 96× OSD	MON/MGR spread across failure domains; one OSD per NVMe device.

Operational test toggles (only during benchmarks):

- ceph osd set noscrub; ceph osd set nodeep-scrub (disabled scrubbing)
- No backfill/self-healing activity during tests; cluster was in HEALTH_OK prior to each run.

3.3.2 mClock Scheduler Policy

mClock is used to ensure predictable arbitration between client IO and background work. For this study, background work was quiesced; nonetheless we set an explicit profile and device ceilings to reflect the NVMe media capability.

Setting	Value	Rationale
osd_op_queue	mclock_scheduler	Default in Squid; required for the mClock params below.
osd_mclock_profile	high_client_ops	Favors client IO; with scrubbing disabled, profile choice does not alter results but reflects production intent.
osd_mclock_max_capacity_iops_ssd	800000	Caps scheduling model at 800k IOPS per OSD to match Micron 9400 class media and avoid unrealistic queue inflation.
osd_mclock_max_sequential_bandwidth_ssd	3.5 GiB/s (3 758 096 384 B/s)	Aligns with per-device seq ceilings, preventing bandwidth over-commit in the model.

Note: For live clusters under recovery, these caps bound contention so client SLOs remain stable; for our tests, recovery/scrub were off.

3.3.3 OSD Memory & BlueStore

Setting	Value	Rationale
osd_memory_target	10 GiB per OSD	With 96 OSDs, this budgets ~960 GiB for BlueStore caches; on the 1.5 TiB cluster, this leaves ~576 GiB for OS/page-cache/daemons.
osd_memory_target_autotune	false	Fixed target improves cross-run repeatability.
BlueStore layout	On-device DB/WAL	One OSD per NVMe; no shared DB devices to avoid cross-device amplification.
CPU/NUMA	OSDs pinned near their NVMe	Minimizes cross-socket memory traffic and tail latency.

3.3.4 Pools, CRUSH, and PG Sizing

Item	Value	Rationale
Test Pool	RBD pool (replicated, **size=3, min_size=2**)	Matches common enterprise durability expectations.
Placement Groups	4096 PGs	With 96 OSDs and size=3 → 128 PGs/OSD (4096×3/96). Sits in the 100–200 PG/OSD target band for SSD/NVMe.
CRUSH Rule	replicated on device-class ssd, failure-domain host	Spans nodes; aligns with hardware topology and availability goals.
Balancer	upmap mode, enabled	Ensures even PG distribution prior to testing.

3.3.5 Client Path (RBD)

Area	Choice	Notes
Client Type	Kernel RBD (krbd)	All benchmark VMs used kernel mapping for production-parity IO path.
Filesystem	XFS on each mapped RBD	Pre-filled to 100% of the test file to remove thin-provisioning variance.
Queue Depth	Per-VM tuned via fio	VM NIC and block queue depths sized to saturate a 100 Gb link.
Compatbility	require_min_compat_client = reef (recommended)	Kernel clients often advertise Reef features; raising to Squid is possible if all clients assert Squid feature bits.

3.3.6 Why These Choices

- Predictability: Fixed memory targets and explicit mClock ceilings remove hidden variability between runs.
- Fairness to hardware: Per-device IOPS and bandwidth caps reflect the NVMe media's realistic envelope, avoiding scheduler oversubscription that can inflate latencies.
- Balance: 4096 PGs yields ~128 PGs/OSD—enough parallelism to keep OSDs busy without excessive metadata overhead.
- Production parity: Kernel RBD + XFS mirrors how most real workloads consume block from Ceph today.

3.4 Performance & Resiliency Highlights

- **64 PCIe lanes** – 16 drives × 4 lanes plus root-complex reserve keep the PCIe fabric congestion-free.
- **400 Gb/s aggregate east-west bandwidth** – with cluster-network bandwidth = 2 × front-end, an OSD commit never queues behind client I/O.
- **NUMA-aware CPU pinning** – OSDs local to each memory domain; BlueStore wal/db remains on-device, avoiding cross-socket write-amplification.
- **Three-zone failure domain** – MONs and MGRs are placed 1-per-rack; a whole rack loss still leaves quorum and dashboard access.
- **Fast fail-in/fail-out** – 9400 PRO's endurance (1 DWPD) paired with a feature introduced in Reef - improved osd_fast_fail_on logic minimises rebuild time and wear.

3.5 Why This Matters

By combining an all-NVMe data path, NUMA-balanced Zen 4 compute, and 200 Gb/s fabric, this design eliminates the usual bottlenecks that appear first in high-IOPS Ceph environments: context-switch thrash, PCIe oversubscription, and replication congestion.

Chapter 4: The Test Environment

4.0 Test Environment

Accurate performance characterization requires that load-generators live outside the storage fabric—otherwise CPU-scheduler contention and shared PCIe links skew the results. For this study we therefore stood up three dedicated client hosts and populated them with identically configured benchmark VMs.

4.1 Physical Client Hosts

Component	Specification
Platform	3 × Proxmox VE 8.2 bare-metal servers
CPU	AMD EPYC 9334 (32 C / 64 T, Zen 4)
Memory	128 GiB DDR5-4800, interleaved across NUMA nodes
Networking	2 × Mellanox ConnectX-5 dual-port 100 GbE • Ports bonded with LACP (802.3ad) • Jumbo MTU 9000
Bandwidth to Ceph	200 Gb/s full-duplex per host

Each Proxmox node is used only for benchmarking traffic—no storage daemons, no auxiliary services.

4.2 Benchmark VM Topology

- **Count** – 36 VMs (12 per host)
- **Guest OS** – Rocky Linux 9.6 (minimal)
- **vCPUs** – 4 (pinned, host-passthrough)
- **RAM** – 8 GiB
- **Disk bus** – virtIO-SCSI single queue, 1 dedicated iothread
- **NIC** – virtIO-net 100 Gb, connected to the same bond/MTU 9000 bridge as the host

4.3 Block & File Layout

- **RBD tests**
  - Each VM receives a freshly-created RBD image.
  - Images are formatted XFS and **100 % pre-filled** to eliminate thin-provisioning variability.
  - Kernel RBD module is used for maximum IO path parity with production.
- **CephFS tests** (where applicable)
  - Single POSIX namespace mounted via kernel client with noatime, inode64.

4.4 Network & System Validation

Before any workload is fired:

1. **45Drives tuning playbook** is applied cluster-wide (IRQ affinity, NUMA pinning, sleep states, disk scheduler, network buffers, tuned, and more) – In-depth tuning guide included in next section.

2. **iperf3, 60** s between every client ↔ OSD-host pair.
    - Acceptance criteria: zero retransmits, ≥ 95 Gb/s per link, congestion window stable.

3. Host dmesg, ethtool -S, netstat -i | column -t and Ceph health are checked for NIC or link anomalies.

4.5 Benchmark Orchestration & Methodology

- **Tool** – fio 3.35 using libaio for both RBD+XFS and CephFS testing.

- **Coordinator** – Ansible control node triggers fio on all VMs simultaneously; forks raised to 50 to avoid serial start-up lag.

- **Workloads**
    - **Single-client, single-thread** (4 kB random writes & reads) to chart latency and IOPS.
    - **Multi-client scale-out** – incrementing active VMs until throughput plateaus or 99th-percentile latency exceeds 2× baseline. (throughput and IOPS)
    - **Large-block sweeps** – 1 MiB sequential read/write for bandwidth ceiling.

- **Ramp/Runtime** – 60 s ramp-up + 300 s steady-state; samples discarded if std-dev > 5 %.

- **Metrics capture** – fio JSON output fed to Prometheus; Grafana dashboards record cluster OSD, MDS, and network utilisation for correlation.

This isolated, repeatable setup ensures that the performance numbers presented in Section 6 reflect the cluster's true headroom rather than artefacts of shared-node benchmarking.

Chapter 5: Performance Results

5.0 Performance Results

This section presents measured throughput and IOPS from the environment defined above. Results are organized as (a) single-client single-thread (b) single-client multi-thread, and (c) cluster-level saturation. Unless noted, bandwidth is shown in **GiB/s** (binary) with MiB/s equivalents preserved where useful.

5.1 Headline Numbers (Fully Tuned, 6-Node Cluster)

- **Single client, single-thread (many workloads rely on single threads)**
  - Seq Read **7.4 GiB/s** (~63.6 Gb/s) · Seq Write **4.26 GiB/s** (~36.6 Gb/s)
  - 4k Rand Read **100 k IOPS** · 4k Rand Write **63.9 k IOPS**

- **Single client, multi-thread (max per-VM):**
  - Seq Read **10.1 GiB/s** (~86.8 Gb/s) · Seq Write **7.36 GiB/s** (~63.2 Gb/s)
  - 4k Rand Read **104 k IOPS** · 4k Rand Write **132 k IOPS**

- **36-client saturation (aggregate):**
  - Seq Read **64.7 GiB/s** (~556 Gb/s) · Seq Write **16.9 GiB/s** (~145 Gb/s)
  - 4k Rand Read **2.57 M IOPS** (~26.8 k/OSD) · 4k Rand Write 8**81.5 k IOPS (~8.04 k/OSD)**
  - 70/30 Seq Mix: **27.7 GiB/s** read / **11.9 GiB/s write**
  - 70/30 Rand Mix: **1.46 M read IOPS** / **627.8 k write IOPS**

Client-side link efficiency (36 clients): peak seq read ≈ **67%** of the 600 Gb/s client NIC aggregate; write ≈ **16%**. (Room to climb with additional senders and/or write-path tuning; see §6.4.)

5.2 Single-Client Baselines

Workload	Single-Thread	Multi-Thread	Scale-Up
Sequential Read	**7.40 GiB/s** (7579 MiB/s, ~63.6 Gb/s)	**10.1 GiB/s** (≈ 86.8 Gb/s)	**1.36×**
Sequential Write	**4.2 GiB/s** (4506 MiB/s, ~36.6 Gb/s)	**7.36 GiB/s** (7539 MiB/s, ~63.2 Gb/s)	**1.67×**
4k Random Read	**100 k IOPS** (392 MiB/s)	**104 k IOPS** (406 MiB/s)	**1.04×**
4k Random Write	**63.9 k IOPS** (250 MiB/s)	**132 k IOPS** (515 MiB/s)	**2.07×**
70/30 Seq Mix – Read	**4.40 GiB/s** (4506 MiB/s)	**6.81 GiB/s** (6969 MiB/s)	**1.55×**
70/30 Seq Mix – Write	**1.89 GiB/s** (1935 MiB/s)	**2.92 GiB/s** (2994 MiB/s)	**1.55×**
70/30 Rand Mix – Read	**58.4 k IOPS** (228 MiB/s)	**81.2 k IOPS** (317 MiB/s)	**1.39×**
70/30 Rand Mix – Write	**25.1 k IOPS** (97.9 MiB/s)	**34.8 k IOPS** (136 MiB/s)	**1.39×**

Notes:

- Multi-thread read pushes a single 100 Gb VM NIC to **~87%** utilization; write to **~63%**.
- Write slopes > read slopes (2.26× vs 1.36×) reflect Ceph's parallel commit path benefitting from deeper queues; read is already close to per-VM NIC limits.

5.3 Multi-Client Scale & Saturation (36 VMs)

Aggregate Metric Result

- Sequential Read: 66 263 MiB/s = 64.7 GiB/s ≈ 556 Gb/s
- Sequential Write: 17 268 MiB/s = 16.9 GiB/s ≈ 145 Gb/s
- 4k Random Read: 2 575 900 IOPS (~26.8 k/OSD)

Aggregate Metric Result

- 4k Random Write: 771 485 IOPS (~8.04 k/OSD)
- 70/30 Seq Mix: 27 698 MiB/s read / 11 938 MiB/s write
- 70/30 Rand Mix: 1 464 400 read IOPS / 627 876 write IOPS

Per-node view (6 storage nodes):

- Seq Read ≈ **11 044 MiB/s (~10.78 GiB/s**, ~92.6 Gb/s) per node
- Seq Write ≈ **2 878 MiB/s (~2.81 GiB/s**, ~24.1 Gb/s) per node

5.4 Observations & Interpretation

- **Ceph's scalability is unmatched:** During testing, all tests were run on a 5-node configuration before running a 6-node configuration, where we saw near-linear scaling with all multi-client saturation testing.

- **Read vs Write gap (expected):** Reads scale higher because they require one successful replica; writes incur 3× replication and durable commit semantics. Your write results are therefore **network/CPU + commit-path bound**, not media-bound.

- **Client-aggregate ceiling:** With **3× 200 Gb/s** client hosts (600 Gb/s total), the **556 Gb/s** peak read indicates some small remaining headroom; additional client senders or distributing VMs across more client hosts should raise the ceiling.

- **Proxmox VE client host CPU saturation:** Testing was regularly showing the CPU of the client host hitting 95-100% utilization, indicating a clear bottleneck – we are confident the Ceph cluster is capable of substantially more throughput and IOPS for cluster-level saturation testing.

- **Queue-depth sensitivity:** Write workloads benefited substantially from multi-threading (2.06×–2.26× gains), consistent with OSD pipelines preferring deeper submission queues.

- **Per-OSD efficiency: ~26.8 k** 4k read IOPS/OSD and **~8.04 k** 4k write IOPS/OSD align with highly tuned NVMe+Squid behavior at 3× replication and conservative fsync semantics. Keep in mind, the 8.04K IOPS for the write workload does not represent the real-world IOPS each OSD was performing/second. Due to a replication factor of 3X happening, the IOPS per disk is actually 3X this at minimum. Beyond this, the cluster was not at peak utilization during these tests. More physical client nodes would have continued to push saturation of cluster further.

5.4 Observations & Interpretation

- **Single-client** = what one big application VM can expect.
- **Multi-thread** = single client is able to do IO in parallel with multiple threads
- **Saturation** = the point where more clients/threads no longer increase throughput without violating latency SLOs.

Chapter 6: The Real World

6.0 Viewing these numbers in real-world terms

Pushing a cluster through a set of rigorous benchmarks has many benefits such as it allows us understand where bottlenecks lie, but most importantly – it can give us a very good idea on the performance of real world workloads will look like. This is because a tool like fio, which is the benchmarking tool used for this whitepaper is able to simulate just about any real-world workload possible.

With the raw numbers out of the way, getting a handle on some of 45Drives' customer base workloads and how this cluster will perform with them will help bring the usefulness of this whitepaper into focus.

Use case	Dominant metric(s)	Your capacity (headline)
Video editing (4K H.265)	Aggregate sequential read/write	556 Gb/s read, 145 Gb/s write → hundreds–thousands of 4K streams
VDI / general VMs	70/30 random mix IOPS	2,087,800 IOPS → ~208,780 steady @10 IOPS; ~20,878 boots @100 IOPS
OLTP(Online Transaction Processing) DBs (Postgres/MySQL)	4k random IOPS (70/30 mix)	2,087,800 IOPS → ~417 @5k, 208 @10k, 104 @20k
HPC checkpoints	Aggregate sequential write	16.9 GiB/s → 1 TiB ≈ 60 s; 2.5 TiB ≈ 151 s; reads 1 TiB ≈ 15 s
AI/ML (checkpoints & input)	Write for checkpoints; read for loaders	16.9 GiB/s → 14 GiB checkpoint ≈ ~0.8 s; 64.7 GiB/s read feeds many trainings
High-res imaging (PACS/pathology)	Large-object sequential read	64.7 GiB/s → ~120 CT studies/s (550 MB), ~132 WSI/s (500 MB)
General file server (NFS/SMB)	Sequential read throughput	64.7 GiB/s → ~64 concurrent 1 GB/s copies

Appendix B – Ceph configuration commands completed

mClock configuration

```
ceph config set osd osd_op_queue mclock_scheduler
ceph config set osd osd_mclock_profile high_client_ops
ceph config set osd osd_mclock_max_capacity_iops_ssd 800000
ceph config set osd osd_mclock_max_sequential_bandwidth_ssd 3758096384  # 3.5 GiB/s
```

OSD configuration

```
ceph config set global osd_memory_target 10737418240                # 10 GiB
ceph config set global osd_memory_target_autotune false
```

Pool configuration

```
ceph osd pool create rbd 4096
ceph osd pool set rbd size 3
ceph osd pool application enable rbd rbd
```

Balancer

```
ceph balancer mode upmap
ceph balancer on

# Test-only toggles (unset after)
ceph osd set noscrub
ceph osd set nodeep-scrub
```

Prior to benchmarking

```
ceph osd unset noscrub
ceph osd unset nodeep-scrub
```

Appendix B – Benchmark Command Suite (60 s ramp, 300 s run)

This appendix lists the exact fio invocations used to generate the results in §6. Replace --filename=<path> with a file on the mounted XFS filesystem inside each benchmark VM (e.g., /mnt/bench/seq.fio and /mnt/bench/rand.fio). All tests use direct I/O and report per-job and aggregate stats.

B.1 Pre-fill (required for thin-provisioned RBD)

Run once per VM per test file before benchmarking to eliminate allocation variability.

```
# Pre-fill sequentially to 100 GiB test file
fio --name=prefill_seq \
  --filename=seq.fio \
  --size=100G \
  --rw=write --bs=1M \
  --ioengine=libaio --iodepth=32 \
  --numjobs=1 --direct=1 --group_reporting \
  --end_fsync=1
(Optional for random tests if using a separate file:)
fio --name=prefill_rand \
--filename=rand.fio \
  --size=10G \
  --rw=write --bs=1M \
  --ioengine=libaio --iodepth=32 \
  --numjobs=1 --direct=1 --group_reporting \
  --end_fsync=1
```

B.2 Single-client, single-thread (60 s ramp, 300 s run)

Sequential write

```
fio --bs=1M --iodepth=32 --size=100G --rw=write \
  --ioengine=libaio --name=test --numjobs=1 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
  --filename=seq.fio --direct=1
```

Sequential read

```
fio --bs=1M --iodepth=32 --size=100G --rw=read \
  --ioengine=libaio --name=test --numjobs=1 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
  --filename=seq.fio --direct=1
```

Random write (4k)

```
fio --bs=4k --iodepth=64 --size=10G --rw=randwrite \
  --ioengine=libaio --name=test --numjobs=1 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
--filename=rand.fio --direct=1
```

Random read (4k)

```
fio --bs=4k --iodepth=64 --size=10G --rw=randread \
  --ioengine=libaio --name=test --numjobs=1 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
--filename=rand.fio --direct=1
```

Sequential mixed 70% read / 30% write (1M)

```
fio --bs=1M --iodepth=32 --size=100G --rw=rw --rwmixread=70 \
  --ioengine=libaio --name=test --numjobs=1 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
  --filename=seq.fio --direct=1
```

Random mixed 70% read / 30% write (4k)

```
fio --bs=4k --iodepth=64 --size=10G --rw=randrw --rwmixread=70 \
  --ioengine=libaio --name=test --numjobs=1 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
--filename=rand.fio --direct=1
```

B.3 Single-client, multi-thread (numjobs=4; 60 s ramp, 300 s run)

Sequential write

```
fio --bs=1M --iodepth=32 --size=100G --rw=write \
  --ioengine=libaio --name=test --numjobs=4 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
  --filename=seq.fio --direct=1
```

Sequential read

```
fio --bs=1M --iodepth=32 --size=100G --rw=read \
  --ioengine=libaio --name=test --numjobs=4 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
  --filename=seq.fio --direct=1
```

Random write (4k)

```
fio --bs=4k --iodepth=64 --size=10G --rw=randwrite \
  --ioengine=libaio --name=test --numjobs=4 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
--filename=rand.fio --direct=1
```

Random read (4k)

```
fio --bs=4k --iodepth=64 --size=10G --rw=randread \
  --ioengine=libaio --name=test --numjobs=4 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
--filename=rand.fio --direct=1
```

Sequential mixed 70% read / 30% write (1M)

```
fio --bs=1M --iodepth=32 --size=100G --rw=rw --rwmixread=70 \
  --ioengine=libaio --name=test --numjobs=4 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
  --filename=seq.fio --direct=1
```

Random mixed 70% read / 30% write (4k)

```
fio --bs=4k --iodepth=64 --size=10G --rw=randrw --rwmixread=70 \
  --ioengine=libaio --name=test --numjobs=4 \
  --group_reporting --time_based --ramp_time=60 --runtime=300 \
--filename=rand.fio --direct=1
```

B.4 Notes

- Tests now use **60 s ramp-up** (--ramp_time=60) and **300 s steady-state** (--runtime=300) as requested.
- --direct=1 bypasses page cache; XFS is mounted with defaults unless otherwise noted.
- Using separate files (seq.fio and rand.fio) avoids cache interaction between sequential and random test sets.
- For multi-client runs, the above commands are executed concurrently across VMs via Ansible with increased forks as described in §4.5.

45Drives ENTERPRISE



Contact Us Today

www.45Drives.com/contact

info@45Drives.com

1-866-594-7199

Authored by:
Mitchell Hall
45Drives Chief Solutions Architect