LUG 2026 · April 29, 2026 · Indianapolis

High-Performance Cloud-Native Lustre on Object Storage with Active-Active HA

Lustre 2.17 on ZFS · GCS / S3 / Azure Blob · objbacker.io

Supramani (Sam) Sammandam

Principal, ZettaLane Systems

suprasam@zettalane.com · zettalane.com

ABOUT SAM

Supramani (Sam) Sammandam — Principal, ZettaLane Systems

- 25+ years building storage systems — Dell, IBM, HP, Lucent Technologies
- Sr. Director of Engineering at Arkologic, the first company to ship all-flash NAS array
- Creator of MAYASTOR (2005) — early Linux software-defined iSCSI/FC SAN
- Developed native ZFS vdev layer objbacker.io presented at openZFS Developer Summit 2025

ZettaLane Mission — Self-Managed easy to use cloud-native Storage Solutions

Leverage object storage to present POSIX file storage — NFS, Lustre, SMB, — with 11 nines durability and no application changes.

- No data copy-in / copy-out · Training data stays in object storage
- Open-source Linux data plane · No vendor lock-in · Terraform-deployed · Multi-cloud · Data sovereignty

THE IRONY

Lustre's 20-Year Detour Back to Object Storage

OST = Object Storage Target

- Name from 2002 T10 OSD standard — intelligent object-storage hardware
- T10 OSD never shipped commercially
- Lustre fell back to ldiskfs / ZFS on block devices
- For 20 years
- 2026: finally do what was intended — ZFS vdev → objbacker.io → actual S3 / GCS

AWS went the other way

- 2006 — S3: everything as objects
- 2008 — EBS: customers demanded blocks
- 2018 — FSx Lustre: on local SSDs, S3 only as HSM tier
- Invented object storage, retreated to block for two decades
- We skipped that detour

Lustre OST was always supposed to run on object storage? We just needed a ZFS vdev to close the loop.

OBJBACKER.IO

Object Storage as a Native ZFS Vdev

Presented at OpenZFS Developer Summit 2025

What it does

Kernel module · cloud bucket as native ZFS vdev

- vdev_read(off,sz) → HTTP GET on bucket
- vdev_write(off,buf) → HTTP PUT / multipart
- At the DMU layer — below CoW, ARC, checksums

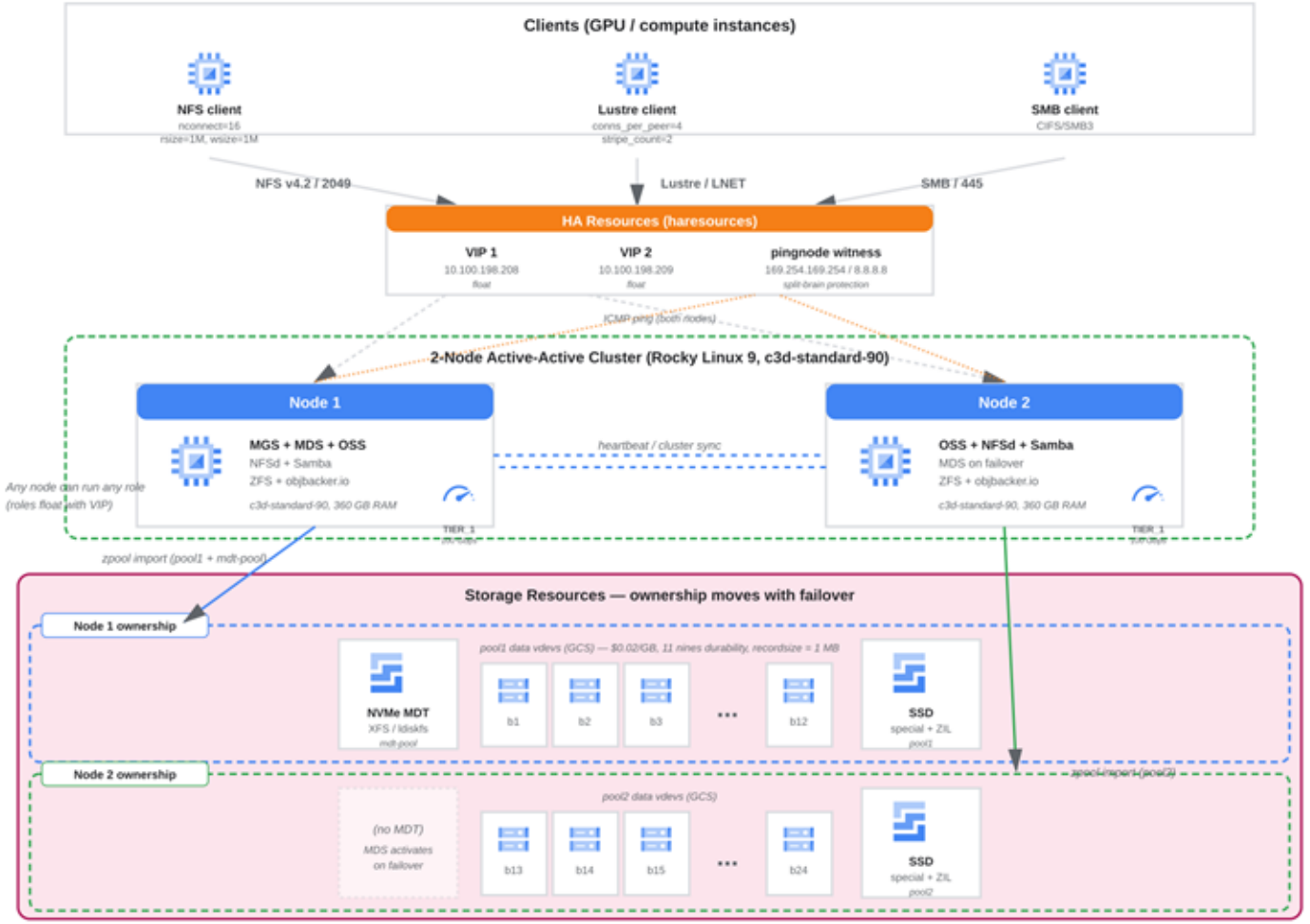
ZFS VDEV IO	/dev/objbacker	objbacker.io
ZIO_TYPE_WRITE	WRITE	PUT
ZIO_TYPE_READ	READ	GET
ZIO_TYPE_TRIM	UTRIM	DELETE
ZIO_TYPE_IOCTL	USYNC	PUT SYNC ALL

Why native vdev, not FUSE?

- FUSE splits large I/O into 4 KB chunks
- Kernel vdev — page cache → NIC zero-copy — 13+ GB/s
- All ZFS features apply: ARC, prefetch, scrub, snapshots, send/receive
- Vendor Go SDKs for S3, GCS, Azure Blob — extreme concurrency

ARCHITECTURE

2 Nodes · 2 OST zpools · 1 MDT (ldiskfs) · 1 MGS · LNET tcp0



One Terraform apply, enable_lustre = true — deploys on GCP, AWS, Azure (incl. Gov)

Each node runs any role — MGS, MDS, OSS, NFSd, Samba — roles float with VIP. Lustre + NFS + SMB from the same ZFS pool.

RUNNING SYSTEM

Lustre 2.17 on ZFS / objbacker.io — live capture

Node 1 — MGS + MDT + OSS0 (VIP 10.100.198.208)

```
$ lctl list_nids
10.100.198.208@tcp

$ lctl dl | awk '$2=="UP"'
 5 UP osd-ldiskfs  zettafs-MDT0000-osd
 6 UP mgs          MGS
 8 UP mds          MDS
10 UP mdt          zettafs-MDT0000
16 UP osd-zfs      zettafs-OST0000-osd
17 UP obdfilter    zettafs-OST0000
13 UP osp          zettafs-OST0000-osc-MDT0000
14 UP osp          zettafs-OST0001-osc-MDT0000
 2 UP ost          OSS
 1 UP mgc          MGC10.100.198.208@tcp
```

Node 2 — OSS1 (VIP 10.100.198.70)

```
$ lctl list_nids
10.100.198.70@tcp

$ lctl dl | awk '$2=="UP"'
 0 UP osd-zfs     zettafs-OST0001-osd
 3 UP obdfilter   zettafs-OST0001
 2 UP ost         OSS
 1 UP mgc         MGC10.100.198.208@tcp
 4 UP lwp         zettafs-MDT0000-lwp-OST0001

$ cat /proc/fs/lustre/mgs/MGS/live/zettafs
fsname: zettafs
   zettafs-MDT0000
   zettafs-OST0000
   zettafs-OST0001
```

One command proves the whole architecture — MDT on ldiskfs NVMe, OSTs on ZFS / objbacker

```
$ lctl get_param osd-*.*.mntdev
osd-ldiskfs.zettafs-MDT0000.mntdev  =  /dev/nvme0n2                    # MDT on NVMe
osd-zfs.zettafs-OST0000.mntdev      =  lperf-pool-node1/lustre-ost     # OST0 on ZFS
osd-zfs.zettafs-OST0001.mntdev      =  lperf-pool-node2/lustre-ost     # OST1 on ZFS
```

Each ZFS pool = 12 objbacker file vdevs (GCS-backed) + 1 NVMe special vdev · Lustre 2.17.0 · brw_size = 16 MB · active-active, VIPs float

POOL STRIPING

12 Buckets = 12 OSTs (Without lfs_migrate)

Push parallelism into ZFS — one OST per node over N bucket vdevs

Traditional Lustre

- Many OSTs, each a separate pool
- Striping at the Lustre layer
- lfs_migrate to rebalance when adding OSTs
- Fixed capacity per OST
- Add capacity = add server + OST + migrate

MayaNAS Cloud-Native

- 1 OST per node
- ZFS pool striped across 10–12 bucket vdevs
- Lustre sees 1 OST; every I/O hits all buckets in parallel
- ZFS allocation classes auto-tier metadata → NVMe
- Add capacity = add a bucket. No downtime, no lfs_migrate.

Same parallelism as 10–12 OSTs — with ONE layer of striping instead of two.

ZFS BASELINE

13 GB/s From Object Storage — Before Lustre Even Enters the Picture

c4-highcpu-144 · 150 Gbps Tier_1 · 12 GCS bucket vdevs · Feb 2026

13.1 GB/s

Cold single-pass FIO
17 sec, 210 GB dataset

10.7 GB/s

5-minute sustained
3 TB read, ARC churn

95.3%

Load balance across buckets
~10-11K ReadObject/sec

Why this works

- ZFS pooled storage — parallel requests saturate the NIC
- recordsize = 1 MB — 1 GCS GET per block, no read amplification
- zfetch prefetch — speculative reads hide object-storage latency
- objbacker.io extreme concurrency per vdev · Google Cloud Monitoring confirms

LUSTRE ON ZFS

Two Tunables Unlock 400× Write Throughput

osd-zfs on high-latency object-storage vdevs

osd_txg_sync_delay_us

- Default: -1 · forces txg_wait_synced() on every write RPC
- Round-trips GCS before ACK
- Measured throughput: 12 MB/s
- Set to 0 — ZFS batches writes into TXGs normally
- Measured throughput: 5.2 GB/s

→ 400× improvement from one tunable

zfetch (ZFS prefetch)

- Works when Lustre bypasses ZPL via osd-zfs
- Verified with DTrace · speculative DMU reads
- Tune for high-latency vdevs:
  - zfetch_max_distance = 512 MB
  - zfetch_min_distance = 128 MB
  - zfetch_max_reorder = 16 MB
  - zfs_vdev_async_read_max_active = 96

MLPerf shuffle: max_reorder=0, max_streams=8

MLPERF 48 H100

ResNet-50 @ 48 H100: 90.72% AU — PASS

MLCommons threshold: 90% · stddev 1.76 across 5 epochs · 2026-03-13

90.72%

Accelerator Utilization

48 simulated H100 GPUs

8.3 GB/s sustained I/O · 77,485 samples/sec

Epochs

Epoch	AU %
1	93.3
2	90.4
3	91.4
4	91.4
5	87.9

Cluster

- 2× c3d-std-90 servers
- Lustre 2.17 on ZFS
- c4-highcpu-144 client
- 150 Gbps Tier_1
- 10,629 files · 210 GB
- 5 shuffled epochs

What makes this unique

- First MLPerf Storage submission with object-storage-backed Lustre — no pre-staging, no cache tier
- 2 server VMs feeding 48 H100-equivalent clients · 90.7% AU is publishable, not marketing

GCS TELEMETRY

8.53 GiB/s peak · 51.6 TiB in 3 hours · 2 VMs

Google Cloud Monitoring — GCS bucket egress during MLPerf runs

Metrics

- Peak: 8.53 GiB/s
- Avg: 5.12 GiB/s (incl. idle)
- Total: 51.6 TiB pulled
- Window: 3 hrs
- 5 MLPerf runs
- Protocol: Lustre → ZFS → objbacker → GCS
- No NVMe cache
- Data never left GCS

UNCONVENTIONAL

Fan-In Lustre — 1 Fat Client · Many OSTs

AI inverts the Lustre workload shape — fan-out becomes fan-in

Traditional Lustre (fan-out)

```
1000+ tiny clients  →  few big OSTs
Each client: a slice of aggregate
HPC: simulation, weather, genomics
Striping = balance server load
```

MayaNAS Lustre (fan-in)

```
1 fat GPU client @ 150–400 Gbps  →  few OSTs
Client gets SUM of OST bandwidth
AI: MLPerf, H100 / B200 training boxes
Striping = aggregate into 1 mount
```

Why NFS can't serve this shape · Why Lustre does

- NFS client mounts ONE server VIP
- Capped at that one server's NIC
- c3d-std-90 NIC: ~12 GB/s · dead-end
- No stripe-across-servers concept

- lfs setstripe -c 2 splits file across OSTs
- Single mount sees SUM of OST bandwidth
- Client NIC becomes the gate, not servers
- Exactly what MLPerf 48 H100 demands

Client 150 Gbps (~18 GB/s) · OST1 ~12 GB/s · OST2 ~12 GB/s · stripe=2 → client-NIC-capped, not server-capped

CLIENT KNOBS

Lustre vs NFS — Same Game, Different Names

Both clients tune the same three things: parallel TCP, RPC payload, direct I/O

What it does	NFS client	Lustre client
Parallel TCP sockets (saturate 100 Gbps+ NIC)	nconnect=16	conns_per_peer=4 (ksocklnd)
RPC payload size = ZFS recordsize	rsize=1M, wsize=1M	max_pages_per_rpc=256 (× 4K = 1 MB)
Bypass page cache for large I/O	O_DIRECT	hybrid_io=1 (auto DIO ≥ 2 MB, Lustre 2.17)
In-flight RPCs per target	(kernel-driven)	max_rpcs_in_flight=64
Checksums (off for perf on trusted net)	— (not configurable)	osc.*.checksums=0

Measured 400 GB cold — same pool, 2 nodes, GCS

- Write: NFS 5.9 GB/s · Lustre 7.0 GB/s (+19%, stripe fan-out)
- Read buffered: NFS 10.3 GB/s · Lustre 9.7 GB/s
- Read O_DIRECT+libaio: Lustre 9.8 GB/s (matches buffered)

The point isn't Lustre ≈ NFS

- Same client architecture, different syntax
- Same ZFS pool serves both
- Lustre wins writes — stripe fan-out for AI checkpoints
- Workload picks the protocol — no duplication

HA ARCHITECTURE

Active-Active · 2 Nodes · VIPs Float · NIDs Don't

Resource ordering (heartbeat)

`mayastor → IPaliases → volmaps`

- Storage ready → VIP up → Lustre bind registers LNET NID
- Stop is reverse: unmount → remove VIP → teardown storage
- Every target uses its VIP as LNET NID — NIDs never change

Why not --servicenode?

- --servicenode announces alternate NIDs if primary goes away
- With floating VIPs the NID never goes away — it lands on another node
- Client reconnects to the same address
- Same concept as NFS mounting a VIP

Data is always in object storage

- Failover = ZFS import from the same buckets
- No RAID rebuild, no degraded mode, no data movement
- Compute role moves — data doesn't
- Traditional deployments use 4+ servers to avoid co-location. We do it with 2 (see next slide).

CHALLENGES

LNET Loopback NID Mismatch (and fail_fast)

1. Half-open TCP — fail_fast for Lustre

- Node dies — client has half-open TCP to VIP
- VIP lands on new node — next packet gets RST
- Default keepalive detects in 60+ sec
- Tune ksocklnd: keepalive_idle=10, intvl=3, count=3
- Detected in ~19 sec instead of 60+

2. LNET loopback NID mismatch

- MDT fails over to node already running an OST
- MDT → co-located OST via 0@lo (hardwired)
- OST has old MDT under VIP@tcp; same UUID, new NID
- OST treats it as replacement — orphan cleanup ~65s
- Bulk RDMA timeouts — OST INACTIVE — client stuck

The Fix

- Pre-evict stale MDT exports from co-located OSTs BEFORE mounting MDT
- 7 lines of shell in the bind path
- MDT connects via 0@lo = fresh registration — no replacement, no disruption
- localrecov kept as defense-in-depth — insufficient alone
- Unique to 2-node active-active — traditional deploys avoid it with separate MDS/OSS servers
- options lnet _peer_discovery_disabled=1

3. LNet Multi-Rail Discovery Breaks Floating-VIP HA

- During failover the survivor briefly carries BOTH VIPs on its NIC
- MR-DD sees two NIDs reply from the same MAC and merges them into one Multi-Rail peer
- Merged-peer cache survives failback. RPCs misroute. Clients EIO / ESHUTDOWN

FAILOVER

30 Seconds · Zero Evictions · Zero Data Movement

Measured with active fio I/O during failover

Step	Duration	Notes
Heartbeat detects dead node	10 s	configurable
Surviving node imports ZFS + acquires VIP	10 s	ZFS import from same buckets
MDT bind (evict stale + mount)	1 s	pre-eviction fix
OST0 bind (mount)	5 s	—
MDT recovery	1 s	0 evictions
OST0 recovery	1 s	0 evictions
OST1 on surviving node	—	never went down
Client detects dead TCP	~19 s	ksocklnd keepalive
Client reconnects MDT + OST0	5 s	—
TOTAL	~30 sec	fio resumes, no EIO

Zero evictions · no RAID rebuild · no data movement · compute role moved, data stayed

COMPARISON

Cloud-Native vs Traditional · vs Managed Lustre

Cloud-Native vs Traditional Lustre

	Cloud-Native (MayaNAS)	Traditional Lustre
Storage backend	GCS / S3 / Blob ($0.02/GB)	Local NVMe ($$$)
Server VMs	2 (MDS + OSS roles float)	2 MDS + multiple OSS
Capacity	Unlimited (object storage)	Fixed per disk
Data durability	11 nines (object storage)	RAID on local disk
Deploy	Terraform, minutes	Days, expertise

Managed Lustre offerings

Service	HA Model	Failover	Data Location	Client Recovery
AWS FSx Lustre	Replace server, reattach EBS	Minutes	Local SSDs / S3 HSM	Automatic
Azure Managed Lustre	Single-AZ HA; Blob import/export	Component-level	Local SSDs	Automatic within AZ
DDN EXAScaler	Active-active pairs, 4+ servers	Sub-minute	Local NVMe / HDD	Automatic
MayaNAS	2min to deploy, Active-Active, 2 nodes	~30–60 sec	Object storage	Automatic, 0 evictions

$0.02/GB object storage vs $0.14/GB FSx Lustre — 7× cheaper at rest, no data movement on failover

"Lustre on object storage — bring your own GCP project, 2 min to mount"

Try It Yourself

MayaNAS Community Edition

Lustre + NFS + SMB + S3 over object storage

BYOC — your GCP project, your buckets

Public Marketplace listing

GCP · AWS · Azure

```
$ git clone github.com/zettalane-
systems/zettalane-terraform
$ cd terraform
$ ./deploy-lustre.sh \
    --cloud gcp \
    -p $GCP_PROJECT \
    -z us-central1-f \
    -k ~/.ssh/google_compute_engine \
     --machine-type n2-highcpu-32 -b 6 --deploy-
     client n2-highcpu-48 \
        --spot
→ 2-node Lustre + client
→ active-active VIP HA
→ buckets in your project
```

github.com/zettalane-systems/zettalane-terraform

Supramani (Sam) Sammandam · suprasam@zettalane.com · zettalane.com

"Failover is boring when your data is in object storage"

What's Next

Lustre Unified Data Platform

Lustre · NFS · SMB · S3

same ZFS pool, same backend

Available on

GCP · AWS · Azure Marketplaces

Follow ZettaLane on LinkedIn

linkedin.com/company/zettalane-systems

Thank You — Questions?

Supramani (Sam) Sammandam · suprasam@zettalane.com · zettalane.com