leil OS

BENCHMARK REPORT / IO500 IOEASY BANDWIDTH

IO500 LeilOS Essential Benchmarks

A single 60-drive SMR node under the IO500 IOEasy workloads, measured against a FIO reference on identical hardware.

AUTHOR
Sergej Paršikov
Senior Performance and DevOps Engineer

DATE
August 2026

SOFTWARE UNDER TEST
LeilOS 5.11 · EC(6,2)
IO500 ISC26 v2 build

MEASURED RESULT — SINGLE NODE, 60 SMR DRIVES, 10 CONTAINERIZED CLIENTS

IO500 WRITE	IO500 READ	IOEASY BANDWIDTH (GEOMETRIC MEAN*)
9.22 GiB/s	**10.25** GiB/s	**9.72** GiB/s
89% of FIO reference · ~3.4 TiB in 374 s	98% of FIO reference · ~3.3 TiB in 333 s	Drive-limited, not software-limited

*Geometric mean of the `ior-easy-write` and `ior-easy-read` results. This is not an IO500 list score — the official IO500 bandwidth score is the geometric mean of all four bandwidth phases including `ior-hard`, and the overall score also incorporates the metadata phases.

CONTENTS

01 What we measured

The storage side is a single storage node with 60 SMR hard drives, running LeilOS 5.11 with EC(6,2) erasure coding and connected over dual 100GbE. The client side is a separate physical server running 10 Docker containers, each with its own LeilFS client mount and 6 I/O jobs, so the node was serving 60 concurrent streams from 10 independent clients throughout the test.

The test had two steps. First we ran our standard FIO benchmark, scaled from the usual single client to 10 parallel containerized clients, to establish the bandwidth reference of the setup (1 MiB blocks, sequential, direct I/O). Then we ran the IO500 IOEasy workloads on the same configuration: `ior-easy-write`, `ior-easy-read`, plus the small random-read probe `ior-rnd4K-easy-read`.

STORAGE NODE	DATA PROTECTION	CLIENTS	NETWORK
60 × SMR HDD 1.4 PiB raw / 1.06 PiB usable	**EC(6,2)** 1.33× write amplification	**10 containers** 6 I/O jobs each — 60 streams	**2 × 100GbE** LACP bond, MTU 9000

02 Headline result

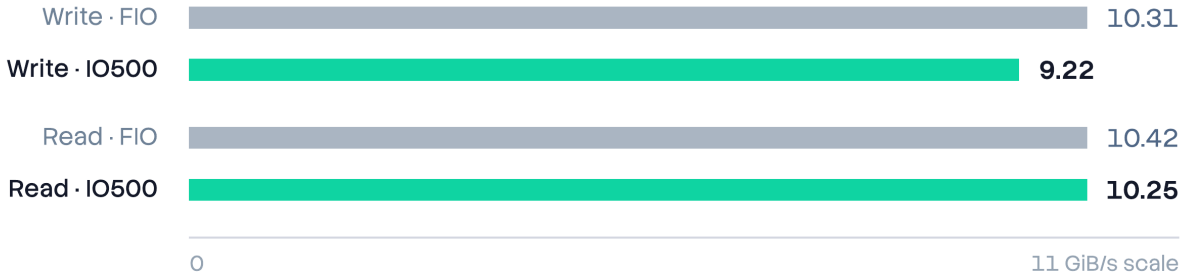
Through 10 concurrent containerized clients, a single 60-drive node sustained **9.22 GiB/s** of IO500 write and **10.25 GiB/s** of IO500 read, for an IOEasy bandwidth score of **9.72 GiB/s**. Each phase moved roughly 3.4 TiB of data. That is 89% (write) and 98% (read) of the FIO reference measured on the same setup, meaning the IO500 harness, the POSIX layer and the containerization add almost no overhead on top of what the drives themselves deliver.

03 Single-node results

TABLE 1 · 60 SMR DRIVES, 10 CONTAINERIZED CLIENTS

WORKLOAD	FIO REFERENCE	IO500 IOEASY	VOLUME / DURATION
Sequential 1M write	10.31 GiB/s ~176 MiB/s per drive	**9.22 GiB/s** ~157 MiB/s per drive	~3.4 TiB in 374 s
Sequential 1M read	10.42 GiB/s ~178 MiB/s per drive	**10.25 GiB/s** ~175 MiB/s per drive	~3.3 TiB in 333 s
Random 4K read	—	**4.44 kIOPS** ~74 IOPS per drive	305 s probe

FIGURE 1 · SEQUENTIAL BANDWIDTH, IO500 IOEASY AGAINST FIO REFERENCE (GIB/S)



The sequential rates correspond to roughly 157–178 MiB/s per drive across the 60 drives. The FIO reference was measured with the same 10 containers and the recommended client cache settings. Individual clients landed within a few percent of one another (1037–1072 MiB/s on writes), so the 10 containers share the node evenly. The random 4K read row is an informational IO500 probe rather than a bandwidth workload, and has no FIO counterpart in this run. See the note on hard drives in section 04.

04 Reading the numbers

Containerized clients are free	10 Docker-packaged LeilFS clients on one physical machine deliver the same aggregate bandwidth as a single host mount, with an even split between clients.
The harness is not the bottleneck	The IO500 numbers sit within 2–11% of the raw FIO reference, so the benchmark is measuring the hardware, not software bottlenecks.
The drives are the limit, in the best sense	With EC(6,2), every 6 units of user data carry 2 units of parity, so at 9.22 GiB/s of user writes the drives are absorbing roughly 12.3 GiB/s of raw writes — about 210 MiB/s per drive, close to the sequential capability of the media.

The random 4K read probe behaves exactly as rotating media dictates: a hard drive serves small scattered reads at seek speed, roughly 74 IOPS per drive here. This is a property of HDD mechanics, not of LeilOS, and it is why LeilOS targets large-file, bandwidth-oriented workloads where SMR capacity economics shine.

05 Scaling to 8 nodes

LeilOS scales out by adding nodes, and aggregate bandwidth grows in proportion to the number of drives. Every node serves its own share of the load in parallel, so per-drive throughput stays flat as the cluster grows; on the client side, the 10-containers-per-machine pattern simply repeats on additional client nodes. An 8-node cluster of 480 SMR drives therefore projects to 8 times the measured single-node figures, with each node's dual 100GbE links keeping the network comfortably ahead of the drives.

TABLE 2 · 8-NODE PROJECTION — 480 SMR DRIVES

WORKLOAD	FIO REFERENCE	IO500 IOEASY	BASIS
Sequential 1M write	~82 GiB/s	**~74 GiB/s**	measured × 8
Sequential 1M read	~83 GiB/s	**~82 GiB/s**	measured × 8
Random 4K read	—	**~36 kIOPS**	measured × 8

Projection from measured single-node results, assuming the client fleet is scaled with the cluster (8 client machines with 10 containers each). Per-drive throughput and the balance between clients stay unchanged as nodes are added.

06 Bottom line

On the IO500 IOEasy bandwidth workloads, a single LeilOS node with 60 SMR drives delivers **9.2 GiB/s writes** and **10.2 GiB/s reads** to 10 concurrent containerized clients — within a few percent of the raw FIO capability of the same hardware, and drive-limited rather than software-limited. Each node serves its own drives independently, so aggregate bandwidth is expected to track drive count; an 8-node system projects to the ~75–80 GiB/s class for IOEasy bandwidth.

The next steps are a multi-node measurement and the metadata phases of IO500, and we will share those results as they come in.

A Appendix A — Hardware and OS setup

A.1 STORAGE SERVER (LEILOS NODE)

COMPONENT	CONFIGURATION
Platform	AIC Vega, baseboard rev. C06, BIOS VEGA_0.13.0 (2025-09-25)
CPU	2 × Intel Xeon Gold 5515+: 16 cores / 32 threads in total, 2 NUMA nodes
Memory	512 GB: 16 × 32 GB DDR5 ECC RDIMM
Storage HBA	1 × ATTO Technology SAS/SATA controller (PCIe, 117c:00a5)
Data drives	60 × host-managed SMR HDD: 52 × WDC WSH722626AL 26 TB (firmware W930) and 8 × Seagate ST30000NM011F-3S 30 TB (firmware SR04): about 1.4 PiB raw, about 1.06 PiB usable at EC(6,2)
Metadata device	1 × 3.84 TB NVMe SSD (WDC WUS4BB038D7P3E1)
System drives	2 × 960 GB NVMe SSD (Samsung MZQLB960HAJR), Linux md mirror
Data network	2 × 100GbE Broadcom BCM57508 in an 802.3ad LACP bond: 200 Gb/s aggregate, MTU 9000

A.2 CLIENT HOST

COMPONENT	CONFIGURATION
Platform	TYAN B8056G68AE12HR, baseboard S8056GME-B, BIOS V1.17 (2024-06-14)
CPU	1 × AMD EPYC 9254: 24 cores / 48 threads, 4 NUMA nodes (NPS4)
Memory	384 GB: 12 × 32 GB DDR5 ECC RDIMM
Data network	2 × 100GbE Broadcom BCM57508 in an 802.3ad LACP bond: 200 Gb/s aggregate, MTU 9000

All test traffic between the client host and the storage node runs over the bonded 2 × 100GbE links. At peak the measured 10.25 GiB/s of reads corresponds to roughly 88 Gb/s on the wire. On writes the client emits 6+2 blocks, so 9.22 GiB/s of user data becomes 12.3 GiB/s on the wire, roughly 106 Gb/s. It is therefore the bonded pair, rather than a single 100GbE port, that carries the load. Even at that peak the bond runs at just over half of its 200 Gb/s capacity, so the network was never the limiting factor in these runs.

A.3 SOFTWARE SETUP

ITEM	STORAGE SERVER	CLIENT HOST
Distribution	Ubuntu 24.04.4 LTS	Ubuntu 24.04.4 LTS
Kernel	`6.17.0-35-generic`	`6.17.0-35-generic`
Leil Software	LeilOS package; 10 chunkserver instances serving the 60 drives	LeilFS client 5.11.0 inside each container
Zoned storage stack	zonefs disk plugin 5.11.0-dev-36, zonefs-tools 1.6.0-2	—
Container runtime	—	Docker 29.6.1, containerd 2.2.5, runc 1.3.6, overlay2 storage driver, cgroup v2
Benchmark tooling	—	fio 3.36, Open MPI 4.1.6, IO500 (ISC26 v2 build)

A.4 BLOCK LAYER, STORAGE SERVER

Each of the 60 SMR drives is exposed through the kernel's zonefs filesystem and mounted with `rw,nosuid,nodev,noexec,noatime,errors=zone-ro`; the zone size is 256 MiB. Per-drive queue settings during the test were the mq-deadline scheduler, `read_ahead_kb=128`, `nr_requests=256` and `max_sectors_kb=4096`, the last of which is applied explicitly before the run:

```
echo 4096 | sudo tee /sys/block/sd*/queue/max_sectors_kb
```

Metadata is stored on XFS on the NVMe device, mounted with `noatime,nodiratime,allocsize=4k,logbufs=8,logbsize=256k`.

A.5 NETWORK AND KERNEL TUNING

Jumbo frames (MTU 9000) are used end to end on the data path. Otherwise, both hosts run stock Ubuntu network and virtual memory settings, with the single exception of `net.ipv4.tcp_rmem`, whose maximum is raised to 32 MiB on the client.

B Appendix B — LeilOS 5.11 and IO500 test setup

B.1 KEY LEILOS CHUNKSERVER PARAMETERS

HDD_TEST_FREQ	10000
GARBAGE_COLLECTION_FREQ_MS	0
NR_OF_NETWORK_WORKERS	6
NR_OF_HDD_WORKERS_PER_NETWORK_WORKER	4
MAX_BLOCKS_PER_HDD_WRITE_JOB	32
MAX_BLOCKS_PER_HDD_READ_JOB	1024

The target filesystem was freshly created, so no reclaim work was pending. Garbage collection was disabled to remove a no-op background task from the measurement.

B.2 CONTAINER DEPLOYMENT

The client containers were created from the official `leil-client` image. First, the image was pulled on the client host and 10 identical containers were started:

```
docker pull leilfs/leil-client:5.11.0-rc1-1-staging-ubuntu-24.04

for i in {0..9}; do
    docker run -dit \
        --name leil-client${i} \
        --hostname leil-client${i} \
        --network host \
        --privileged \
        --entrypoint /bin/bash \
        leilfs/leil-client:5.11.0-rc1-1-staging-ubuntu-24.04 \
        -c "sleep infinity"
done
```

Then, containers were connected to a common Docker network with stable container-name DNS resolution:

```
docker network create leil-net

for i in {0..9}; do
    docker network connect leil-net leil-client${i}
done
```

B.3 REQUIRED PACKAGES

Each container was prepared with the packages required for SSH, MPI, and IO500:

```
apt-get update
apt-get install -y \
    openssh-server \
    openssh-client \
    openmpi-bin \
    libopenmpi-dev \
    bash \
    coreutils
```

SSH was enabled in each container and passwordless root SSH keys were installed on all containers, allowing MPI to launch remote processes.

B.4 IO500 INSTALLATION

The `io500` executable was built on host according to the instructions from https://github.com/io500/io500 and copied to `/usr/local/bin/io500`, then made executable:

```
chmod +x /usr/local/bin/io500
```

B.5 FILESYSTEM MOUNT

The LeilOS filesystem was mounted in every container:

```
sfsmount /mnt/leil -H 192.168.50.211 \
  -o sfswritecachesize=8192,\
sfschunkserverwriteto=10000,\
sfschunkserverwavereadto=5000,\
sfschunkservertotalreadto=10000,\
maxreadaheadrequests=2,\
cacheexpirationtime=5000,\
sfscacheperinodepercentage=100,\
readbuffersexpirationtime=3000,\
readworkers=32,\
sfswriteworkers=32,\
readcachemaxsizepercentage=80,\
readaheadmaxwindowsize=65536,\
sfswritewindowsize=256,\
sfscachemode=NEVER,\
sfsusewriteflushpacket=1
```

B.6 IO500 CONFIGURATION

The following configuration was deployed identically to all containers as `/root/config-ioeasy.ini`:

```
[global]
datadir = /mnt/leil/tests/ec62smr/
resultdir = /root/results/
api = POSIX
drop-caches = FALSE
drop-caches-cmd = sudo -n bash -c "echo 3 > /proc/sys/vm/drop_caches"

[debug]
stonewall-time = 300

[ior-easy]
API = POSIX
transferSize = 1m
blockSize = 9920000m
filePerProc = TRUE
uniqueDir = FALSE

[ior-easy]
run = TRUE
[ior-easy-write]
run = TRUE
[ior-easy-read]
run = TRUE
[ior-rnd4K-easy-read]
run = TRUE
[mdtest-easy]
run = FALSE
[mdtest-easy-write]
run = FALSE
[ior-hard]
run = FALSE
[ior-hard-write]
run = FALSE
[ior-hard-read]
run = FALSE
[mdtest-hard]
run = FALSE
[mdtest-hard-write]
run = FALSE
[find]
run = FALSE
[find-hard]
run = FALSE
[mdtest-easy-stat]
run = FALSE
[mdtest-hard-stat]
run = FALSE
[mdtest-easy-delete]
run = FALSE
[mdtest-hard-read]
run = FALSE
[mdtest-hard-delete]
run = FALSE
```

Page cache was not dropped between phases. The IO500 drop-caches step writes to `/proc/sys/vm/drop_caches`, which is not container-namespaced: from inside a client container it would flush the host's page cache globally, once per MPI rank, and would still leave the storage node's cache untouched — clearing both sides would require a remote `drop-caches-cmd` hook. Cache residency cannot account for the results in any case: each phase moved about 3.4 TiB against 384 GB of client RAM and 512 GB on the storage node, roughly 10× and 7× respectively, and the client mount ran with `sfscachemode=NEVER`.

B.7 DEPLOYMENT AND EXECUTION

```
for i in {0..9}; do
    docker cp ~/io500-io500-isc26_v2/io500 \
        leil-client${i}:/usr/local/bin/io500
    docker cp ~/io500-io500-isc26_v2/config-ioeasy.ini \
        leil-client${i}:/root/config-ioeasy.ini
    docker exec leil-client${i} chmod +x /usr/local/bin/io500
done
```

MPI was executed with four ranks per container:

```
docker exec leil-client0 mpirun \
  --allow-run-as-root \
  --host leil-client0:4,leil-client1:4,leil-client2:4,leil-client3:4,\
         leil-client4:4,leil-client5:4,leil-client6:4,leil-client7:4,\
         leil-client8:4,leil-client9:4 \
  -np 40 \
  --map-by ppr:4:node \
  io500 /root/config-ioeasy.ini
```

This setup provides 10 containers × 4 MPI ranks = 40 MPI processes, with all I/O directed to the shared LeilOS filesystem.