# lfs drain throughput

How fast `lfs drain` (LU-20908) empties an OST, before and after the copy and scan changes this report led to, compared with `lfs migrate`. Measured as the drop in the drained OST's used space per second. Test VM, 12 November 2026.

**3.9×**
faster drain of one 1 GiB object: 314 → 1,209 MB/s ( `lfs migrate` : 846)

**2.2–3.6×**
drain against `lfs migrate` on 3-stripe files from 16 MiB objects up (was 1.1–2.4×)

**2.1–2.6×**
drain against `lfs migrate` on 1 MiB files, unchanged: the cost per file decides there

**1.5×**
faster OST scan with 4 threads than 1, on 400k objects (0.37 → 0.24 s)

VM numbers: OST disks are files on the host, so these compare tools and versions against each other and are not real bandwidth. One run per case, 1 GiB on the drained OST.

---

## What changed

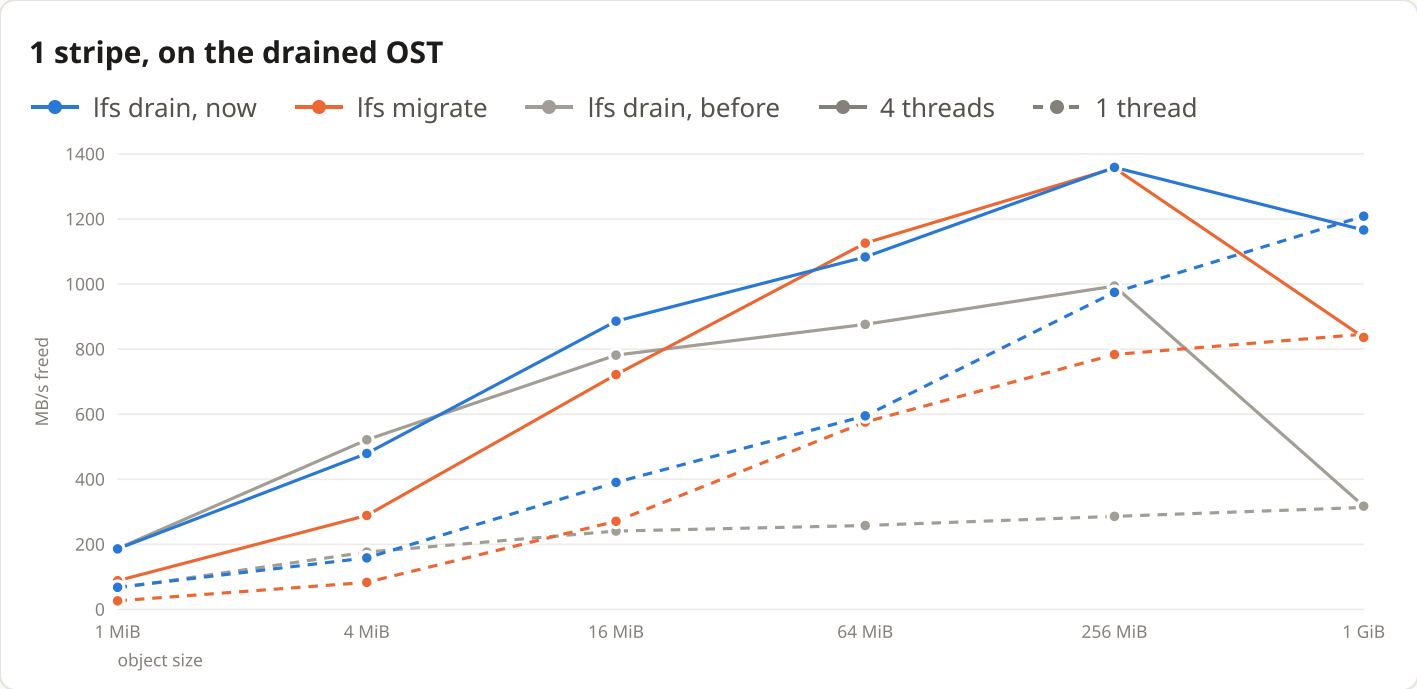
The first version of this report found the drain copying in 4 MiB synchronous pieces and reading the OST with one thread. These changes are now in Gerrit 70470 and 70471, not pushed yet:

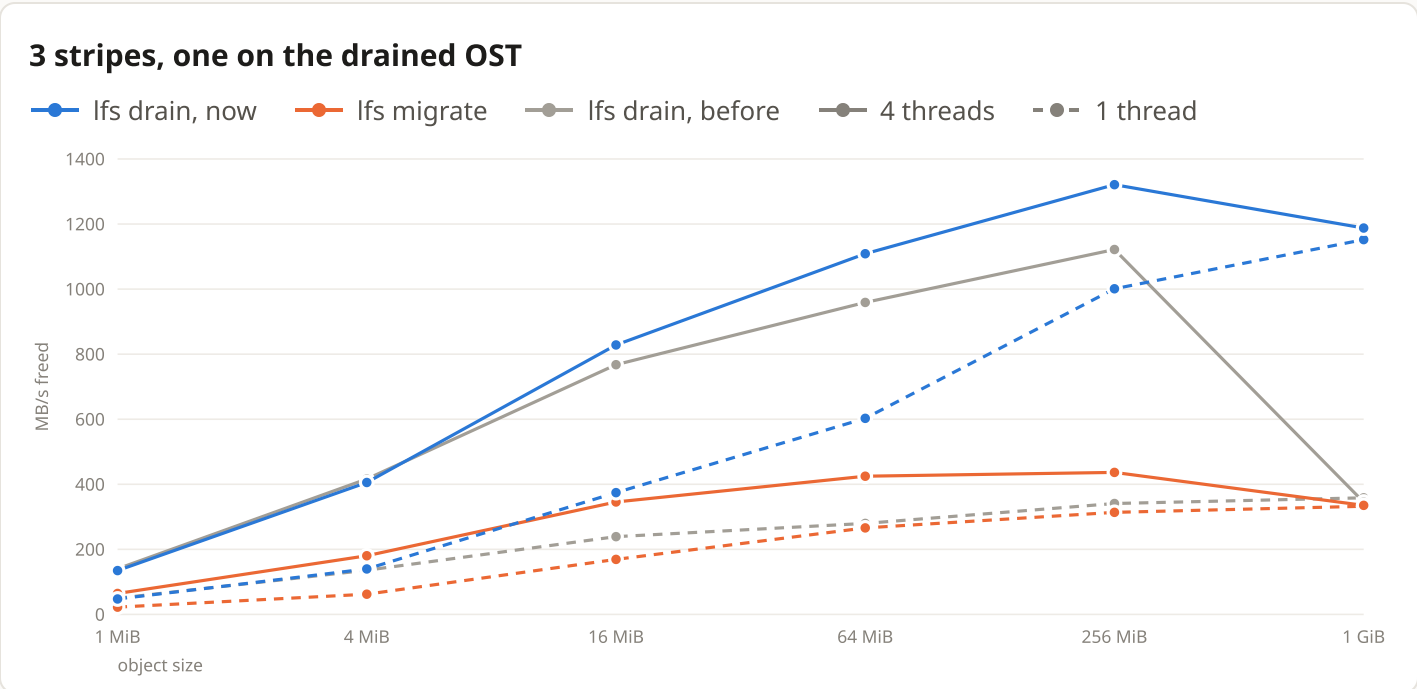
1. **64 MiB pieces**, as `lfs migrate` uses, no bigger than the object; `--io-size` sets it (4K to 1G). done
2. **Reads and writes overlap**: `--io-threads` threads (2 by default) take the next piece of an object from a shared cursor, so one writes while another reads. done
3. **A large object is copied in several places at once** with `--io-threads` above 2. done
4. **The OST is read by as many threads as `--threads`**, up to 32 ( `lfsp_thread_count` ). done
5. **`--stats` reports `freed` and `fmbps`**, the drop in the drained OSTs' used space and its rate, next to the MB copied. done

sanity-pfl 29a–30l pass on the new code (17 of 17), including a new test, 30l, that copies a sparse object in 1 MiB pieces with 4 threads and checks its data and holes.

## Results

MB/s freed on the drained OST against object size (log scale). Solid lines: 4 file threads; dashed: 1. The new drain runs with its defaults (64 MiB pieces, 2 copy threads per object). `lfs migrate` was run again on the same build. The drain finding its files by scanning the OST ran within a few percent of `--files-from`, so only the `--files-from` runs are drawn; both are in the data table.




### What the results show

- **Objects of 16 MiB and up: 1.1–3.9× faster than before**, most for few, large objects. One file thread on a 1 GiB object went from 314 to 1,209 MB/s.
- **The drain is now level with `lfs migrate` or faster at every size.** On 1-stripe files it is level at 64 MiB and 1.2–1.4× faster at 16 MiB and at 1 GiB. On 3-stripe files it is 2.2–3.6× faster from 16 MiB, because `lfs migrate` still copies three times what it frees.
- **Small objects are unchanged.** At 1 and 4 MiB an object is one piece, and the time goes to the work per file: lease, volatile file, layout swap. The drain is still about 2× faster than `lfs migrate` there. Moving 400k empty files took 3.3 ms per file with 4 threads.
- **`--io-threads 4` helps when there are fewer large objects than file threads.** With one 1 GiB object it gave 1,312 to 1,378 MB/s against 1,152 to 1,209 with the default 2, and with 256 MiB objects at 1 file thread 1,161 against 975 (1-stripe). With 4 files in flight it is within noise, so 2 stays the default.

| layout | object | threads | drain before | drain now | now / before | --io-threads 4 | lfs migrate | drain / migrate |
|---|---|---|---|---|---|---|---|---|
| 1 stripe | 1 MiB | 1 | 65 | 68 | 1.0× | 67 | 26 | 2.6× |
| 1 stripe | 1 MiB | 4 | 187 | 186 | 1.0× | 188 | 88 | 2.1× |
| 1 stripe | 4 MiB | 1 | 176 | 158 | 0.9× | 158 | 83 | 1.9× |
| 1 stripe | 4 MiB | 4 | 521 | 479 | 0.9× | 499 | 289 | 1.7× |
| 1 stripe | 16 MiB | 1 | 241 | 390 | **1.6×** | 389 | 271 | 1.4× |
| 1 stripe | 16 MiB | 4 | 782 | 886 | 1.1× | 907 | 722 | 1.2× |
| 1 stripe | 64 MiB | 1 | 258 | 595 | **2.3×** | 599 | 576 | 1.0× |
| 1 stripe | 64 MiB | 4 | 876 | 1083 | **1.2×** | 1175 | 1126 | 1.0× |
| 1 stripe | 256 MiB | 1 | 286 | 975 | **3.4×** | 1161 | 784 | 1.2× |
| 1 stripe | 256 MiB | 4 | 994 | 1359 | **1.4×** | 1276 | 1357 | 1.0× |
| 1 stripe | 1 GiB | 1 | 314 | 1209 | **3.9×** | 1312 | 846 | 1.4× |
| 1 stripe | 1 GiB | 4 | 317 | 1166 | **3.7×** | 1350 | 836 | 1.4× |
| 3 stripes | 1 MiB | 1 | 50 | 47 | 1.0× | 51 | 22 | 2.1× |
| 3 stripes | 1 MiB | 4 | 140* | 135* | 1.0× | 136* | 64 | 2.1× |
| 3 stripes | 4 MiB | 1 | 136 | 140 | 1.0× | 146 | 62 | 2.2× |
| 3 stripes | 4 MiB | 4 | 416* | 406* | 1.0× | 399* | 180 | 2.2× |
| 3 stripes | 16 MiB | 1 | 239 | 374 | **1.6×** | 391 | 169 | 2.2× |

| layout | object | threads | drain before | drain now | now / before | --io-threads 4 | lfs migrate | drain / migrate |
|---|---|---|---|---|---|---|---|---|
| 3 stripes | 16 MiB | 4 | 768 | 828 | 1.1× | 797 | 346 | 2.4× |
| 3 stripes | 64 MiB | 1 | 280 | 603 | **2.2×** | 592 | 266 | 2.3× |
| 3 stripes | 64 MiB | 4 | 959 | 1109 | 1.2× | 1076 | 425 | 2.6× |
| 3 stripes | 256 MiB | 1 | 341 | 1001 | **2.9×** | 1061 | 314 | 3.2× |
| 3 stripes | 256 MiB | 4 | 1122 | 1321 | 1.2× | 1291 | 437 | 3.0× |
| 3 stripes | 1 GiB | 1 | 358 | 1152 | **3.2×** | 1378 | 332 | 3.5× |
| 3 stripes | 1 GiB | 4 | 345 | 1188 | **3.4×** | 1325 | 335 | 3.5× |

MB/s freed. “before”: 4 MiB synchronous pieces, one OST scan thread. * the run failed and left some data on the OST (see the open issue).

---

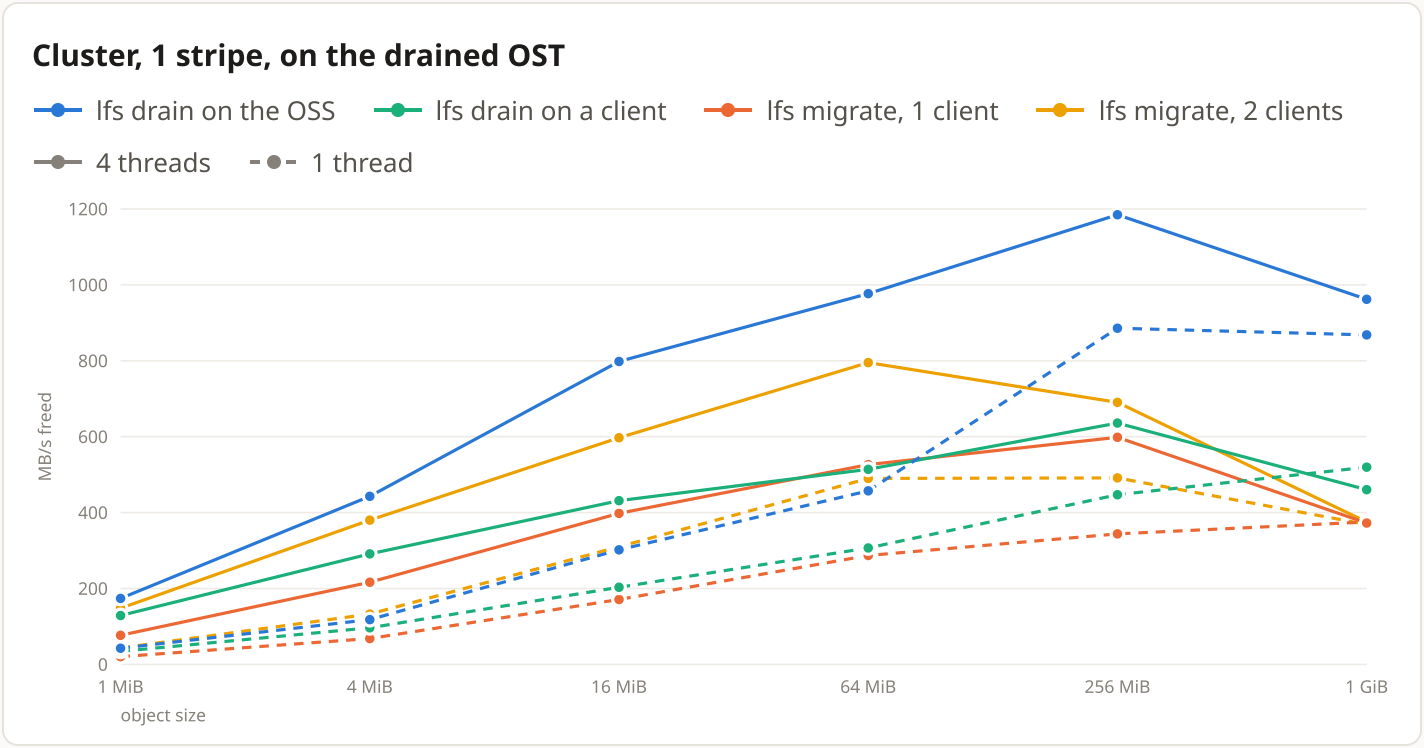
## On a cluster: 2 MDS, 2 OSS, 2 clients

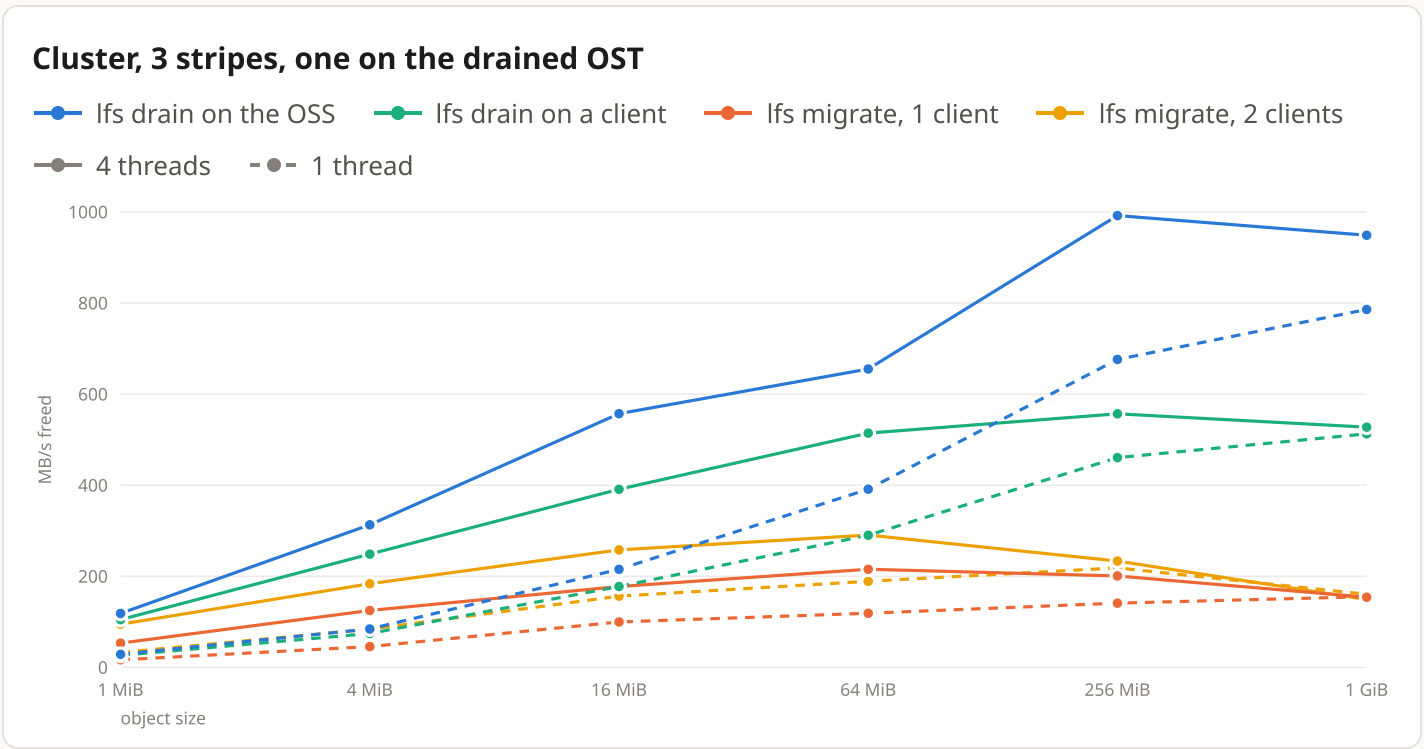
The same matrix on six VMs, so that the copy crosses the network and `lfs migrate` can run where an admin would run it, on a client. All six are Rocky Linux 10.2 VMs with 4 vCPUs and 6 GiB on the same host, built from the same Lustre as “now” above.

| | |
|---|---|
| mds1 | MGS and MDT0000 |
| mds2 | MDT0001 |
| oss1 | OST0000 (drained) and OST0001; also mounts the filesystem, to run `lfs drain` |
| oss2 | OST0002 and OST0003 |
| client1, client2 | clients |
| Targets | ldiskfs on 6 GB virtio disks whose image files are in host RAM (tmpfs), so no run waits on the SSD |
| Network | `tcp` over a host bridge between the VMs; on oss1, OST0000 and OST0001 are reached over `0@lo` |
| Test files | in a directory striped over both MDTs, so both MDSs create and destroy objects; the 3-stripe files use OST0000 and two other OSTs |

Tools, all on the same files: **lfs drain on the OSS** (oss1, with `--files-from`; the scan variant is in the data table), **lfs drain on a client** (client1, `--files-from`), **lfs migrate on one client**, and **lfs migrate**

**on two clients**, each taking half the list with the same thread count, so twice the threads in all. Solid lines: 4 threads; dashed: 1.




### What the cluster shows

- **Run on the OSS, the drain is about 2× faster than `lfs migrate` on 1-stripe files** (1.9–2.6× at 4 threads, 1.6–2.6× at 1), and **1.7–6.2× faster on 3-stripe files**, where migrate copies three times what it frees. With 4 threads it went from 174 MB/s at 1 MiB objects to 1,185 MB/s at 256 MiB.

- **Running `lfs migrate` on two clients does not close the gap.** With twice the threads it got 1.5–2.1× more than one client at 1–4 MiB. At 4 threads per tool the OSS drain is still faster at every size: 1.2–2.6× on 1-stripe files and 1.3–6.4× on 3-stripe files. (At 16 and 64 MiB with 1 thread, two clients running 2 threads in all were level with the 1-thread drain. At 1 GiB there is one file, so the second client has nothing to do.)
- **Where the drain runs matters more than which tool copies.** On 1-stripe files a drain run from a client was level with `lfs migrate` from 16 MiB up (0.98–1.24×), and 1.35–2.1× slower than the same drain on the OSS, which reads the drained OST locally and sends only the writes over the network. On 3-stripe files the client drain still beats migrate by 1.6–3.4×, as it copies a third of the data.
- **`--io-threads 4` gave no consistent gain here.** Against the default 2 it was within ±10% or lower at every size (for example 828 against 962 MB/s with one 1 GiB object at 4 threads), so 2 stays the default.
- **The failure in the open issue is much more frequent across two OSSs**: 24 of 48 drain runs on 3-stripe files, at 1 and 4 threads and every object size up to 256 MiB, against none of 48 on 1-stripe files. See the open issue.

| layout | object | threads | drain on OSS | drain on client | migrate, 1 client | migrate, 2 clients | OSS drain / migrate 1 client | drain --io-threads 4 on OSS |
|---|---|---|---|---|---|---|---|---|
| 1 stripe | 1 MiB | 1 | 43 | 34 | 20 | 43 | 2.1× | 43 |
| 1 stripe | 1 MiB | 4 | 174 | 129 | 77 | 148 | 2.3× | 170 |
| 1 stripe | 4 MiB | 1 | 118 | 97 | 68 | 133 | 1.7× | 121 |
| 1 stripe | 4 MiB | 4 | 443 | 291 | 217 | 380 | 2.0× | 441 |
| 1 stripe | 16 MiB | 1 | 302 | 203 | 171 | 310 | 1.8× | 291 |
| 1 stripe | 16 MiB | 4 | 798 | 431 | 398 | 597 | 2.0× | 683 |
| 1 stripe | 64 MiB | 1 | 457 | 307 | 287 | 490 | 1.6× | 478 |
| 1 stripe | 64 MiB | 4 | 977 | 514 | 526 | 795 | 1.9× | 1001 |
| 1 stripe | 256 MiB | 1 | 886 | 447 | 344 | 491 | 2.6× | 966 |
| 1 stripe | 256 MiB | 4 | 1185 | 636 | 598 | 690 | 2.0× | 1059 |

| layout | object | threads | drain on OSS | drain on client | migrate, 1 client | migrate, 2 clients | OSS drain / migrate 1 client | drain --io-threads 4 on OSS |
|---|---|---|---|---|---|---|---|---|
| 1 stripe | 1 GiB | 1 | 868 | 520 | 375 | 369 | 2.3× | 796 |
| 1 stripe | 1 GiB | 4 | 962 | 460 | 373 | 375 | 2.6× | 828 |
| 3 stripes | 1 MiB | 1 | 29* | 26* | 17 | 33 | 1.7× | 28* |
| 3 stripes | 1 MiB | 4 | 119* | 105* | 53 | 95 | 2.2× | 117* |
| 3 stripes | 4 MiB | 1 | 84* | 74* | 46 | 84 | 1.8× | 83* |
| 3 stripes | 4 MiB | 4 | 313* | 249* | 125 | 184 | 2.5× | 309* |
| 3 stripes | 16 MiB | 1 | 215* | 178* | 100 | 156 | 2.2× | 213 |
| 3 stripes | 16 MiB | 4 | 557* | 391 | 178 | 258 | 3.1× | 531 |
| 3 stripes | 64 MiB | 1 | 391 | 290 | 119 | 189 | 3.3× | 367 |
| 3 stripes | 64 MiB | 4 | 655 | 514 | 216 | 291 | 3.0× | 682* |
| 3 stripes | 256 MiB | 1 | 676* | 461 | 141 | 219 | 4.8× | 586* |
| 3 stripes | 256 MiB | 4 | 992 | 557 | 201 | 234 | 4.9× | 801 |
| 3 stripes | 1 GiB | 1 | 786 | 513 | 155 | 161 | 5.1× | 829 |
| 3 stripes | 1 GiB | 4 | 949 | 528 | 154 | 149 | 6.2× | 831 |

MB/s freed on OST0000. * the run failed and left some data on the OST (see the open issue).

---

## OST scan: records against the LFU Object Stream

Gerrit 70423–70426 (LU-20730) read a device scan as LFU chunks, which a consumer can read in parallel as shards. The drain does not use them; for this test a test-only switch made it read the OST through `llapi_lfu_open()`, as N shards in N threads, against `llapi_scan_device()` with N

workers. Only the scan was timed, with cold caches in the VM, on an OST holding 400k objects, 3 runs each.

| reader | threads | seconds (median) | runs | objects/s | against records ×1 |
|---|---|---|---|---|---|
| LFU stream, shards | 1 | 0.58 | 0.53, 0.60, 0.58 | 690,817 | 0.63× |
| LFU stream, shards | 2 | 0.37 | 0.36, 0.37, 0.50 | 1,098,174 | 1.00× |
| LFU stream, shards | 4 | 0.24 | 0.24, 0.24, 0.24 | 1,649,502 | 1.51× |
| llapi_scan_device(), workers | 1 | 0.37 | 0.37, 0.41, 0.37 | 1,095,198 | 1.00× |
| llapi_scan_device(), workers | 2 | 0.29 | 0.34, 0.26, 0.29 | 1,398,367 | 1.28× |
| llapi_scan_device(), workers | 4 | 0.24 | 0.25, 0.24, 0.24 | 1,663,078 | 1.52× |

With 4 threads, `llapi_scan_device()` read the 400k objects in 0.24 s and the LFU stream in 0.24 s, against 0.37 s for one thread. The stream is slower than records with one thread, from decoding the chunks, and level with them at 4, so it gives the drain no speedup here. Either way the scan is a small part of a drain: reading 400k objects took a quarter of a second, and moving those 400k files took 22 minutes. The VM reads its disk from the host page cache; on a real OST, where the scan waits on the disk, more threads may help more, and that needs measuring there.

**To use the stream, 70423–70426 need changes:**

- The stream does not send `trusted.fid`, so it cannot say which file an OST object belongs to. For this test, `LLAPI_SCAN_OWNER` was made to send `LFU_EA_FID` (and the LMA), which `scan_owner()` reads.
- `scan_device_lfu_open()` in 70424 does not build on the LU-20613/LU-20637 scanner: backend selection is now per kind (`scan_backend[kind]`, `sb_open(device, search, …)`). It was ported for this test.
- 70426 conflicts with LU-20637 in `llapi_find_device()` (`--fid2path`, pre-pass); the drain does not need it and it was left out.

---

## Open issue: new objects placed on the closed OST

> In every build tested, some drains failed with *"the MDS put a new object on an OST being drained"*, although `no_create` had been set 15 seconds before: 4 of 72 runs before the changes, 2 of 20 with 64 MiB pieces only, 4 of 72 after, and 2 of 24 with `--io-threads 4`. All were 3-stripe files of 1 and 4 MiB objects at 4 threads, so it does not depend on the copy; 3–7% of the data stayed on the OST. On a 4-OST filesystem a 3-stripe file leaves only one OST that holds none of its objects, so this may be the MDS's fallback when that OST runs short of precreated objects. That is not confirmed. A second drain run picks up what is left.

> **On the 6-VM cluster it failed in 24 of 48 drain runs on 3-stripe files** (OSS, scan, client and `--io-threads 4` alike), with 1 and 4 threads and object sizes from 1 to 256 MiB, leaving 1–4% of the files on the OST (1 of 4 files at 256 MiB). None of the 48 runs on 1-stripe files failed, and `lfs migrate`, which replaces all three stripes and may use any open OST, never did. Two things differ from the single VM and may explain the higher rate: two MDSs allocate objects, and each learns that the OST is closed only from its own statfs of it; and the one free OST for a 3-stripe file is on the other OSS. A run with more OSTs, or with a longer wait after closing, would tell these apart.

---

## Answers to Andreas's questions

**How should bandwidth be measured?**
As Andreas suggested: the drop in used space on the drained OST, divided by the tool's wall time, read from the OST itself (`osd-*.kbytestotal − kbytesfree`) after the destroys of the old objects have reached it. `lfs drain --stats` now prints this as `freed` and `fmbps`, from statfs through the mount; it lags the copy by the time the MDS takes to destroy the old objects.

**What I/O size does the drain use?**
Now 64 MiB by default, set with `--io-size` up to 1 GiB, with 2 pieces of an object in flight at once (`--io-threads`). It was 4 MiB, one synchronous read and then write at a time.

**Are the scan and the copies multithreaded?**
Both are now. `--threads N` (default 4) moves N files at once and reads the OST with N threads, up to 32; `--io-threads` copies pieces of each object in parallel.

**Sergey's 17 / 28 / 37 MB/s**
Comparing these needs the object sizes and stripe counts on that OST, and how each figure was measured. Below 16 MiB per object every tool here is bound by per-file overhead, not by bandwidth. The script below measures all three tools the same way and can run unchanged on that cluster.

---

## Method

For each layout and object size the script:

1. writes files with 1 GiB in total on the drained OST (1 stripe placed on it, or 3 stripes with one on it);
2. waits until the OST's used space holds steady with no pending OSP changes, and records it;
3. closes the OST (`no_create=1`), waits 15 s, drops caches;
4. runs one tool over exactly those files and times it;
5. waits for the old objects' destroys to land on the OST and the used space to hold steady again, and records the drop.

It records `seconds`, `freed_mb`, `drain_mbps` (freed / seconds, the figure to compare), `moved_mb` (what the tool had to copy) and `amplif` (moved / freed).

**Test system**

| | |
|---|---|
| **Host** | |
| CPU | 2 × Intel Xeon E5-2650 v4 @ 2.20 GHz, 12 cores / 24 threads each (48 CPUs), 2 NUMA nodes |
| Memory | 377 GiB |
| OS | Rocky Linux 10.1, kernel 6.12.0-124.56.1.el10_1 |
| Hypervisor | QEMU/KVM 10.1.0 (qemu-kvm-10.1.0-16.el10_2.5), ltvm 0.5.226 |
| VM disk storage | Raw image files on XFS on a Kingston SV300S37A 120 GB SATA SSD; virtio-blk, default cache mode (writes go through the host page cache) |
| **VM: servers and client on one node** | |
| vCPUs / memory | 4 vCPUs, 6 GiB |
| OS | Rocky Linux 10.2, kernel 6.12.0-55.41.1.el10_0_lustre |
| Lustre, before | 2.17.60 master with Gerrit 70746, 70747, 70470, 70471 PS7, 69511 and a test-only change allowing component 0; the 64 MiB-piece build differs only in the piece size |
| Lustre, now | The same with 70470 and 70471 as changed above, plus 70423–70425 and the test-only scan switch |
| Targets | 1 MDT and 4 OSTs, ldiskfs, 8 GB virtio disks each (7.3 GB usable per OST, 524,288 inodes, 4 KiB blocks, 512-byte inodes), on dm-flakey as the test framework sets up |
| Network | Client and servers over `tcp` on the same node |
| Drained OST | OST0003; the 3-stripe files use OST0003, OST0000 and OST0001 |

**Limits.** The host caches the OST disk files in its 377 GiB of memory, so the drain reads data that is mostly in host RAM rather than on the SSD, and the large-object runs last only 1–3 s, so the absolute numbers mean little and small differences are noise. The ratios between tools and versions, and the trends with object size, are what this shows.

---

## Running it on a real cluster

Run as root on the OSS of the OST to drain, with the filesystem mounted there. Test on an empty or spare OST: the script writes and deletes data on it and toggles its `no_create`.

```
# 10 GiB per case, object sizes 4 MiB to 1 GiB, 1 to 16 threads, all tools, 3 runs
drain-bench.sh -m /mnt/fs -o 7 -d 10240 -s "4 16 64 256 1024" \
    -l "plain wide" -w 4 -t "1 4 8 16" -T "drain drainscan migrate lmf" -r 3 \
    > results.csv 2> progress.log
```

```
# the drain with 4 copy threads per object and 256 MiB pieces
drain-bench.sh ... -T drain -D "--io-threads 4 --io-size 256M"
```

For the scan cost, which only shows at scale, run `drainscan` against `drain` on an OST that already holds millions of objects, with a small `-d`.

▶ **drain-bench.sh**

---

## Data

| build / tool | layout | object MiB | files | threads | seconds | freed MB | MB/s freed | copied MB | copied / freed |
|---|---|---|---|---|---|---|---|---|---|
| before: lfs drain --files-from | 1 stripe | 1 | 1024 | 1 | 15.69 | 1024 | 65.3 | 1024 | 1.00 |
| before: lfs drain --files-from | 1 stripe | 1 | 1024 | 4 | 5.47 | 1024 | 187.1 | 1024 | 1.00 |
| before: lfs drain (scan) | 1 stripe | 1 | 1024 | 1 | 14.99 | 1024 | 68.3 | 1024 | 1.00 |
| before: lfs drain (scan) | 1 stripe | 1 | 1024 | 4 | 5.25 | 1024 | 194.9 | 1024 | 1.00 |
| before: lfs migrate | 1 stripe | 1 | 1024 | 1 | 38.19 | 1024 | 26.8 | 1024 | 1.00 |
| before: lfs migrate | 1 stripe | 1 | 1024 | 4 | 11.20 | 1024 | 91.4 | 1024 | 1.00 |
| before: lfs drain --files-from | 1 stripe | 4 | 256 | 1 | 5.81 | 1024 | 176.2 | 1024 | 1.00 |
| before: lfs drain - | 1 | 4 | 256 | 4 | 1.96 | 1024 | 521.4 | 1024 | 1.00 |

Rows with less than 1,024 MB freed are the failed runs above.