
Beyond RAID-6

md-kmec

Many-Parity Software RAID for Lustre OSTs

In the md ecosystem: high-speed encoding/decoding, redundancy, integrity and quick rebuild

Hiroshi Nishida, PhD The Lustre Collective

Presenter: Artem Blagodarenko, PhD The Lustre Collective

LAD'26 — Lustre Administrator and Developer Workshop



Where Idiskfs OSTs sit today: md, and RAID-6

- Many Idiskfs-backed OSTs run on Linux software RAID (**md**).
- md tops out at **RAID-6**: two parity disks.
- At OST width (tens of disks), a **third concurrent failure means data loss**.
- And a wide disk rebuilds **slowly** — the long degraded window is the dangerous part.

The two things that hurt at scale: *shallow redundancy* and *slow rebuild*.



What a wide OST wants beyond RAID-6

- A wide OST wants **three things past RAID-6**:
 - **Deeper redundancy** — tolerate more than two failures,
 - **End-to-end integrity** — detect and self-heal silent corruption,
 - **Declustered rebuild** — no single-spare funnel.
- ZFS is a mature, capable option for these — the trade is leaving `ldiskfs+md` and the **in-place, zero-copy, stripe-aligned O_DIRECT** write path a busy OST is tuned around.

In-place = overwrite-in-place: the same physical block is rewritten on every overwrite, so an object stays where `mballoc` aligned it. ZFS is copy-on-write — every write lands on fresh blocks, so stripe alignment has to be re-won each time.

The question: can we get all three *inside* the md ecosystem?

md-kmec: the answer, in the md ecosystem

- **md-kmec** — the “raidkm” md personality (**level 71** > raid6).
 - “*km*” = $k + m$: **k** data disks + **m** parity disks ($m \leq 16$).
 - “*kmec*” = $km + ec$: **ec** Erasure Coding.
- Stays in the **md** / **mdadm** / **dm** / **LVM** ecosystem: same tools, no new filesystem.
- **Compatible with stock md** RAID-5 and RAID-6.
- Adds three capabilities past RAID-6 — **redundancy**, **integrity**, **fast rebuild**:

Redundancy

Reed–Solomon,
up to 16 parities

Integrity

per-4 KiB checksums,
self-heal from EC

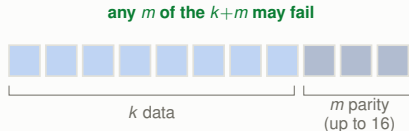
Fast rebuild

declustered RAID,
distributed spare

... plus md/raid5 performance work that helps stock RAID too.

Redundancy: many-parity erasure coding

- Generalizes RAID-6 to **Reed–Solomon: up to 16 parities** (more are possible) — survives **16 simultaneous** disk failures.
- **Erasure-coding engine:**
 - Same encoding/decoding approach as stock mdraid for $m \leq 3$,
 - For $m > 3$, a more generalized technique with **our fork of Intel ISA-L** (open source EC library):
 - Patent-clean,
 - Faster encoding/decoding on non-accelerated hardware.
- **Read-Modify-Write (RMW) base** — different from ZFS's Copy-on-Write (CoW).



m chosen per array; swept $m = 2 \dots 8$
incl. degraded I/O, wired to $k+m \leq 255$.

Integrity: native checksums that self-heal

- Native **per-4 KiB CRC-32C**, verified **inline on read**.
- On a mismatch, the block is **reconstructed from the erasure code and healed in place** — not merely flagged.
- End-to-end protection against **silent corruption**, now **composing** with the many-parity redundancy.
- **Faster** than dm-integrity.

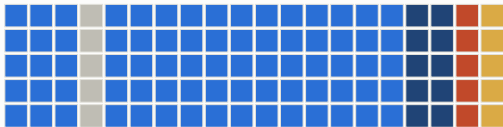
read block → verify CRC → **mismatch** → decode from parity → **heal in place, return good data**

Declustered RAID: many narrow arrays, scattered per row

Split the wide array into **many narrow $k+m$ arrays**, placed at **permuted positions each row** — not fixed sub-arrays. A failed disk is then rebuilt from a *different* set of survivors in every row; a **distributed spare** spreads the rebuild *writes* too.

Non-declustered

one wide 16+2 array • 80% usable



rebuild funnels to **one** spare

Declustered

two 7+2 groups • 70% usable



rebuild **spread** over the pool

■ data
 ■ parity
 ■ 2nd group
 ■ spare
 ■ failed (D3)
 ■ rebuild write

Declassified rebuild: the win scales with the pool

Rebuild-write funnelling

busiest disk: classic $\div$ declassified

$N=14, g=6$  **14.2 $\times$**

$N=42, g=10$  **42.5 $\times$**

$N=80, g=13$  **85.0 $\times$**

Wall-clock: rebuild a failed 16 GB member on an 80-disk pool — **44.9 s** vs **785.1 s** for a classic 78+2 = **17.5 $\times$** faster (real NVMe).

Write funnelling $\approx$ pool width N — reconstruction fans across the pool instead of one replacement.

Rebuild cost is really one number: stripe width

A classic array is just declustered with one stripe across every non-spare disk ($G=N-s$): a decode reads the **other members of the stripe**, width -1 survivors — so a rebuild's read cost is stripe width minus one, pool-wide or narrow group.

Survivor reads per rebuilt chunk

$N=80, s=2$ • classic $G=78$, dcl $g=13$ • lower is better

classic decode  $77 = G-1$

dcl decode  $12 = g-1$

dcl copy  1

Two independent cuts, stacked: **narrow stripes** take classic's $G-1$ to a group's $g-1$ — a factor $(G-1)/(g-1) \approx N/g (\approx 6.4\times)$; **copy-from-spare** then takes $g-1$ to the ~ 1 -disk spare copy — a further $\approx g-1$.

Three widths: rebuild **write** spread $\approx N$ (the $85\times$ headline) • decode **read** amplification $\approx N/g$ • copy vs decode $\approx g-1$. And N/g is also the rebuild's parallelism ceiling — one geometry knob, three consequences.

Performance work — and it helps stock RAID too

- Three fixes to the shared md write path — they speed up **ordinary RAID-5/6 too**, not just md-kmec:
 - **Use every CPU core**, not one — a global lock was serialising writes,
 - **Stop copying** data that is already a full stripe,
 - **Faster parity math** (AVX-512).
- Result: **several times** the small-write throughput of today's default — and the gap *grows* with more cores and wider arrays.
- We also found a **correctness** bug stock md shares: while an array is being reshaped, some reads could be served from the wrong place. Fixed here; headed upstream.

All three target exactly what a busy OST does: large, aligned, direct writes.

Upstream: the fixes are headed into Linux

- **Not** md-kmec-only — they land in stock md/raid5, so every RAID-5/6 user gets them. On the linux-raid list, in review with the md maintainers.
- Three focused series, after review of the first posting:
 - **Correctness** — NULL deref on CPU-hotplug registration failure; degraded-count arithmetic,
 - **Rebuild** — submit a *window* of stripes during resync; reserve stripe cache for user I/O,
 - **Scalability** — per-NUMA worker groups; raise the stripe-cache, hash-lock and batch limits that cap multi-core scaling.
- Covered before it merged — Phoronix, June 2026: *“Linux MD RAID5 Seeing Scalability Improvements Up To 17%”*

<https://www.phoronix.com/news/Linux-MD-RAID5-Scalability-Work>

<https://pbxscience.com/linux-md-raid5-patches-deliver-up-to-17-scalability-boost-on-multi-core-systems/>

Stock md, tuned stock md, md-kmec

8+2, NVMe	stock	tuned stock	md-kmec	vs tuned
Random 4 KiB write, IOPS	47,678	102,856	107,186	1.04×
OLTP 16 KiB, IOPS	39,742	76,124	76,103	1.00×
Degraded seq read, MiB/s	1,223	4,412	4,906	1.11×
Degraded random read, IOPS	88,946	154,699	240,591	1.56×
Rebuild, idle array, MiB/s	158	261	296	1.13×
Rebuilding: foreground read, MiB/s	666	2,396	3,912	1.63×

10 GCP local NVMe, 128 KiB chunk, 32 vCPU; 4 ABBA rounds. **Tuned stock** = stock md with md-kmec's defaults set by hand (`group_thread_cnt`, `stripe_cache_size`, `skip_copy`). On `null_blk`, where the drives are not the limit: degraded random read 2.30× tuned stock, idle rebuild 1.63×, and rebuilding under a read 2.85× faster while serving 1.89× the read.

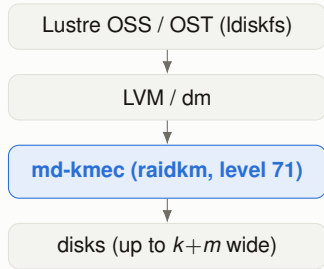
Healthy: the gain is tuning **md-kmec ships on by default** — upstream's defaults suit small, memory-thrifty boxes (embedded, home NAS), not a wide OST.

Degraded or rebuilding: md-kmec reads a lost disk's data as whole rows.

Why staying in the md ecosystem matters

- **Add or remove disks and parities while the OST stays mounted** — stock md can only add.
- **dm-raid** and **LVM** support (`lvcreate -type raidkm`).
- Same operational model as md: `mdadm`, `dm`, `lvm` — no new filesystem to learn or trust.
- Builds against **RHEL 9** and **RHEL 10** kernels — i.e. today's Lustre servers.

Easy migration from existing RAID-6 to md-kmec
— in place, no rebuild, no data copied.



Free win 1: tell the filesystem how wide a stripe is

- ext4 already lines its writes up with the RAID stripe — but nobody ever told it the width, so writes landed *across* stripe boundaries and the array had to read-modify-write.
- `mkfs.lustre` now reads the width from the array and sets it for you; `mount.lustre` warns if it is missing or stale.
- **Not one line of `ldiskfs` changed** — plain ext4 on md gets this too.

8 data disks, 128 KiB chunk, 4 KiB blocks: **stride** = chunk/block = 32, **stripe_width** = 32 × 8 = 256

```
mkfs.lustre --ost --mkfsoptions="-E stride=32,stripe_width=256" /dev/md0
```

```
mke2fs -t ext4 -E stride=32,stripe_width=256 /dev/md0
```

```
tune2fs -E stride=32,stripe_width=256 /dev/md0 (existing filesystem)
```

Our `mkfs.lustre` probes the array and seeds these itself — an explicit `--mkfsoptions` still wins.

Free win 2: the journal — its own disk, or async

- Once writes were aligned, the leftover cost was not the data at all — it was the filesystem's *journal* sharing the array.
- **Give the journal its own disk: +56–58 %** write bandwidth on real NVMe when the workload commits often.
 - `mke2fs -O journal_dev /dev/nvme0n1`
 - `mkfs.lustre --mkfsoptions="-J device=/dev/nvme0n1"`
- **No spare device? Stop waiting for the commit instead:**
`obdfilter.*.sync_journal=0` replies before the journal lands — clients keep the data cached until it does, and replay it after a crash.

Both wins are *configuration*. Any `ldiskfs` OST on md can do this today.

Takeaways

- Four things a wide OST gets, **without leaving md**:
 - **Choose how many failures to survive** — 2, 3, 4 ... up to 16, paying capacity for each,
 - **Silent corruption** found *and* repaired on the fly,
 - **Rebuild** 17.5× **faster** on an 80-disk pool,
 - **Change the array's shape while it runs** — add or remove disks and parities.
- And it keeps the fast, direct write path a busy OST depends on.
- Plus md RAID-5/6 speed-ups for **everyone**, headed upstream.

Many-parity, self-healing, fast-rebuilding storage — with the md tools you already run.

Open source: github.com/scopedog/mdraid-super
umbrella repo — md-kmec, mdraid, mdadm

Thank you

Questions?

Hiroshi Nishida, PhD The Lustre Collective

Presenter: Artem Blagodarenko, PhD The Lustre Collective

Open source: github.com/scopedog/mdraid-super

