LUSTRE DESIGN PROPOSAL

Combined MDT and OST Pools

Author Di Wang **Status** Draft **Date** 2026-10-16 **Scope** Pool definition, recovery failure isolation

1 Summary

Today, a Lustre pool contains OSTs. This proposal adds MDT membership so one pool can control both metadata and file-data placement. New objects inherit the pool throughout a bound subtree. Existing contents do not move when the binding changes; administrators use lfs migrate to move them.

There are two kinds of pool:

- Plain pool (lctl pool_new). A placement policy. Targets may be shared with other pools and used outside the pool. Cross-pool renames are allowed and do not move existing objects. Plain pools do not change recovery. Existing OST pools retain this model.
- Isolated pool (lctl pool_new -I). A proposed failure domain with exclusive targets and server-enforced containment. Rename and hard link cannot cross its boundary; controlled migration can move whole subtrees in or out. After an administrator fences an unavailable pool, healthy MDTs should recover without waiting for it. Access inside the pool fails promptly. A permanently lost pool can be retired through a separate drop operation.

MDT0 remains outside all pools in this proposal because it provides services that the filesystem cannot skip. Isolation also requires a boundary recovery protocol; exclusive placement alone does not remove cross-MDT transaction dependencies.



Figure 1. Intended dependency model outside migration: each isolated pool depends on the shared core, while the core can recover without an offline pool. The boundary protocol must establish this guarantee.

2 Current state

2.1 OST pools

- `lctl pool_new` and `pool_add` go to `mgs_pool_cmd()`, which writes `LCFG_POOL_*` records to the client log (for the `clilov` device) and to every MDT log (for the LOD device).
- `mgs_pool_check_ostname()` accepts only `fsname-OSTxxxx_UUID`.
- `lov_pool.c` on the client and `lod_pool.c` on the server keep near-identical pool tables built on `lu_tgt_pool`. `lod_pool_add()` searches only `lod_ost_descs`.
- `class_process_config()` ignores errors from `LCFG_POOL_*` commands (`err = 0`). A node that can't apply a pool record logs it and keeps processing the log.

2.2 Directory placement

- A default LMV normally selects one MDT (-i N) or balances new subdirectories across available MDTs (-i -1). An explicit MDT list selects stripes for a directory; it does not provide a named, inherited allocation set for balanced single-stripe subdirectories.
- The pool name is already reserved in both LMV formats: `lmv_user_md_v1.lum_pool_name` and `lmv_mds_md_v1.lmv_pool_name`, both covered by wiretest.
- `lfs setdirstripe -p` already fills `lum_pool_name`, and LMV unpacks `lmv_pool_name` into `lsm_md_pool_name`. No allocator reads it.
- llite clears `lum_pool_name` when it sends an inherited default LMV (`MDS_CREATE_DEFAULT_LMV`). LOD's cached default (`lds_dir_def_*` in `lod_default_striping`) has no pool field.
- `lmv_locate_tgt_create()` falls back to `lmv_locate_tgt_by_space()` when the chosen MDT is marked no-create. That fallback chooses among all MDTs today.

2.3 DNE recovery

- At startup each MDT runs one `lod_sub_recovery_thread()` per MDT to fetch its update log. `lod_sub_init_llog()` starts these threads for every entry in `lod_mdt_descs`.
- `tdtd_replay_ready` is set only after every MDT has `ltd_got_update_log`. Until then `check_for_recovery_ready()` keeps extending the recovery timer.
- Transient update-log fetch failures (-ETIMEDOUT, -EAGAIN, or -EIO) are retried. The existing abort_recov_mdt escape path abandons update recovery broadly rather than declaring one isolated pool unavailable.
- Permanent deactivation (`conf_param <target>.mdc.active=0`) marks the target's config records `CM_EXCLUDE`, so the target is added inactive. Update-log retrieval still runs for it. The MGS already refuses to deactivate MDC0.

A persistently unavailable MDT can therefore delay recovery elsewhere. The proposal aims to limit that dependency to the affected pool without discarding required recovery information.

3 Goals and exclusions

3.1 Goals

- One pool can contain MDTs and OSTs.
- A directory bound to a pool keeps everything created below it on the pool's targets.
- After an isolated pool is fenced offline, other MDTs recover without waiting for its logs, subject to a valid boundary recovery protocol.
- Clients fail fast, without RPC timeouts, inside an offline pool.
- A lost isolated pool can be dropped permanently.
- Move directories and files between pools through an explicit, recoverable migration operation.

3.2 Excluded from this proposal

- New security mechanisms or performance-tier policies. Existing nodemap and fileset policies may be used for tenant access.
- Inode quotas per pool.
- Detecting bad pools automatically. The administrator declares a pool offline.
- Automatic, policy-driven movement of data between pools.

4 Pool definition

4.1 Membership

```
lctl pool_new  testfs.proj
lctl pool_add  testfs.proj MDT[1-2] OST[8-f]
lctl pool_list testfs.proj
```

- The MGS would accept fsname-MDTxxxx_UUID, validate it against fsdb_mdt_index_map, and reject MDT0. This extends the current OST-name validation path (I1).
- Each pool keeps separate MDT and OST lists, with allocation state for each target type. Both lists can use lu_tgt_pool.
- All pools share one namespace, limited to 15 characters (`LOV_MAXPOOLNAME`).

4.2 What a pool limits

A pool has no fixed target type: it can list MDTs, OSTs, or both. Its binding controls future allocation for the target types it contains.

```
lctl pool_new testfs.proj
lctl pool_add testfs.proj MDT[1-2] OST[8-f]
lfs setdirstripe -D -p proj /mnt/testfs/proj   # tree limited to the pool
lfs setstripe -p proj /mnt/testfs/file         # objects limited to OST8-OST15
```

- New subdirectories use the pool's MDTs, and new file objects use its OSTs. File inodes and DoM data remain with the MDT holding the file's directory entry.

- A pool constrains only the target types it lists. No MDT members means directory placement is unchanged; no OST members means file-data placement is unchanged. Existing OST-only pools therefore retain their behavior.
- For plain pools, membership changes affect future allocations without moving existing objects. Adding the first OST starts constraining new file objects; removing the last MDT ends the directory constraint. Isolated pools need the additional membership rules in section 5.9.
- Separate metadata and data pool defaults can provide similar placement for a plain subtree. For isolation, use a single combined pool so both target lists belong to the same failure domain.

4.3 Configuration records

Reuse LCFG_POOL_NEW/ADD/REM/DEL membership records and the existing configuration logs where possible. A separate pool-state record is proposed for isolation and offline state.

Device	Today	Proposed
LOD (MDT logs)	OST members only	`lod_pool_add()` looks the name up in `lod_ost_descs` or `lod_mdt_descs` and adds it to the matching list.
LOV (client log)	OST members only	Skips MDT members for its own allocator and stores the full membership in a per-filesystem pool table shared with LMV.
LMV (client)	No pool support	Reads MDT members from the shared table. No new client log records are needed.
Old clients and servers	n/a	An MDT member fails `lov_pool_add()` or `lod_pool_add()`, and `class_process_config()` ignores the error. Old nodes see only the pool's OSTs.

4.4 Binding a directory

```
lfs setdirstripe -D -p proj -c 1 -i -1 /mnt/testfs/proj
lfs getdirstripe -D /mnt/testfs/proj
```

- Pool storage. Record the binding in the directory's default LMV (lum_pool_name). Striped directories also carry the pool in lmv_pool_name.
- Single-stripe directories. Restrict lmv_locate_tgt_create() and its selection helpers to the pool's MDT members, including the fallback when a selected MDT is marked no-create.
- Striped directories. Restrict lod_mdt_alloc_qos() and lod_mdt_alloc_rr() to the pool. Reject an explicitly selected MDT outside the pool with -EINVAL. The proposed default behavior reduces a stripe count that exceeds the available pool members; explicit overstriping semantics need validation.

- Inheritance. Keep the pool binding at every depth even when other default LMV settings reach max_inherit. The proposed fallback is a pool-only default: stripe count 1, offset -1, unlimited max_inherit, and no max_inherit_rr. MDT selection continues to use space balancing within the pool.
- File data. When the effective default LOV names no pool, use the directory pool's OST list. Cache that directory binding in a proposed lds_dir_def_pool field. An explicit LOV pool can override a plain pool; an isolated pool rejects layouts that violate containment. A pool without OST members leaves file-data placement unchanged.
- Client inheritance. Preserve lum_pool_name when llite sends an inherited default LMV.

4.5 Tools

- `lctl pool_add`, `pool_remove` and `pool_list` accept and show MDTs.
- `lfs df -p proj` shows the pool's MDTs and OSTs.
- `lfs getdirstripe` shows the pool. `lfs find --pool` also matches directories by their LMV pool.
- `lfs migrate` moves directories and files between pools (section 6).

4.6 Rules for plain pools

Plain pools control placement without creating failure domains. Members may overlap and may also be used outside the pool. A rename changes the namespace but does not relocate objects. Clients without MDT-pool support retain existing directory allocation behavior, so their creates do not honor the MDT binding.

5 Isolated pools and failure isolation

5.1 Isolated pools as failure domains

The intended model is that the shared core can recover without an offline isolated pool, and each pool depends only on the core and its own members. Ordinary operations do not span isolated pools. A transit link temporarily changes this model during migration (section 6.3). Boundary recovery must make the intended dependency direction safe.

MDT0 provides the filesystem root and shared metadata services, including FID location, sequence allocation, and quota coordination. This proposal keeps MDT0 outside both plain and isolated pools. The core remains a shared availability dependency.

An isolated pool must contain at least one MDT. Pools without OST members provide metadata isolation only (section 5.8). Membership removal is covered in section 5.9.

ID	Invariant	Enforced by
I1	MDT0 is never in a pool.	The MGS rejects MDT0 membership under this proposal.

ID	Invariant	Enforced by
I2	Exclusive membership. A target belongs to at most one isolated pool, and no allocation outside the pool uses it.	The MGS checks at `pool_add`. LOD and LMV leave isolated members out of pool-less allocation and out of other pools.
I3	The pool root sits on the boundary. The name entry of the pool's top-level directory is on an MDT outside every isolated pool, such as MDT0. Its inode and everything below it are in the pool.	Create the root through a boundary protocol that makes the core side recoverable without the pool. Remote mkdir alone does not establish this guarantee.
I4	No rename or hard link crosses the boundary. Migration crosses it only under a transit link (section 6.3).	The server returns -EXDEV for boundary-crossing rename and hard link. Tools may offer copy-and-delete behavior; its application semantics must remain explicit.
I5	No striped directory spans the boundary.	Allocation enforces it. A root directory striped over pool MDTs blocks creating the pool.
I6	The server enforces placement.	Validate pool placement on the server, including explicit layouts and migration. Reject unsupported clients for operations that require pool-aware enforcement.

```
# proposed syntax
lctl pool_new -I testfs.proj              # isolated pool
lctl pool_add testfs.proj MDT[2-3] OST[8-f]
lfs mkdir -p proj /mnt/testfs/proj        # pool root: entry outside, inode inside

lctl pool_new -I testfs.meta              # isolated pool, no OSTs listed
lctl pool_add testfs.meta MDT[4-5]
```

5.2 Marking a pool offline

```
# proposed syntax
lctl pool_offline testfs.proj
lctl pool_online  testfs.proj
```

- Only isolated pools can be taken offline. MDT0 remains outside them under I1.
- Persist the pool state and generation in the client and MDT configuration logs so restarting nodes can apply the state before recovery. Generations distinguish the current state from stale connections.
  - The existing CM_EXCLUDE mechanism is a candidate for representing inactive members. Inactivity alone must not be treated as proof that a target is safe to omit from recovery.
  - A pool-level state record distinguishes an intentionally offline pool from an ordinary inactive target.
- Notify running nodes through configuration-lock updates. Before declaring the pool offline, fence stale servers and clients and resolve or contain outstanding requests and locks. Notification alone is not fencing.

5.3 What the other MDTs do

- Update-log retrieval. The proposed change to lod_sub_init_llog() omits members of a fenced offline pool from the readiness wait. Keep their update logs for rejoin. Exclusion must be explicit; setting ltd_got_update_log must not imply that missing records were read.
- Boundary transactions. Retain records involving offline members. A boundary protocol must define which local updates can safely replay and when records can be cancelled. Replaying only the local portion of an arbitrary distributed transaction is not sufficient to establish consistency.
- OSP activity. Deactivate connections to offline members and stop precreate and update-log synchronization to them under the fencing protocol.
- Filesystem-wide operations. Capacity reports distinguish offline capacity, quota accounting retains outstanding grants, and checks report skipped targets. Barriers and snapshots must report incomplete scope or operate on an explicitly selected healthy scope.
- **LFSCK.**
  - Namespace LFSCK recognizes an offline pool root and preserves its entry rather than treating it as dangling.
  - Layout LFSCK skips inaccessible pool targets and reports incomplete coverage.

5.4 What clients do

- Deactivate MDC and OSC imports for pool members using persisted state at mount and configuration updates at runtime.
- Fast failure. Resolve the target through FLD or the file layout, then return -EIO for a known offline target. In-flight requests and stale cached state are handled by fencing; fail-fast behavior applies after the offline state is effective.
- Keep healthy namespace operations available. Listing a parent can return the pool root name, while operations requiring the offline root inode or its descendants return -EIO.
- Define replay and lock cleanup per pool. Do not discard healthy-pool state because the same client also accesses an offline pool. Aborting recovery for offline targets does not by itself reconcile their durable transactions.

5.5 Full restart with a pool offline

1. Start the MGS and load the persisted pool state and generation.
2. Start core and healthy MDTs. Exclude fenced offline members from update-log retrieval only under the boundary protocol's rules.
3. Complete client replay and distributed update recovery for the healthy targets, including durable boundary decisions.
4. Allow mounts whose roots and required services are available. A tenant mount rooted in the offline pool remains unavailable.
5. Keep the offline pool's targets down or fenced until an administrator initiates rejoin.

5.6 Rejoin

1. Start the repaired targets in recovery mode. Validate identity, generation, and membership before permitting client access. Recover pool-local transactions and retrieve any required core boundary records without depending on unrelated offline pools.
2. Resolve boundary transactions before restoring access.
   - Outside migration, expected boundary changes include creating, renaming, and removing pool roots. The protocol must retain enough information to determine their committed outcomes.
   - Possible inconsistencies include a root entry without its inode or an inode without its entry; transaction records may carry additional obligations that also need resolution.
   - Use a targeted LFSCK pass for residual consistency checks. LFSCK is not a substitute for resolving transaction outcomes, and delayed update replay cannot be assumed optional without a recovery proof.
3. After replay, boundary checks, lock cleanup, and quota reconciliation complete, pool_online publishes the new state and permits client imports to reactivate.

5.7 Dropping a lost pool

```
# proposed syntax; the pool must be offline
lctl pool_drop testfs.proj
```

- Retire the pool root entries and finalize retained boundary records according to the drop protocol. Cancelling records is safe only after their surviving-side obligations are resolved.
- Remove the lost targets from active configuration and retain sufficient identity history to reject returning targets, stale FIDs, and obsolete replay requests.
- Finalize the quota accounting for the retired targets after fencing makes it impossible for them to return and issue updates.
- With exclusive OST membership, no surviving file outside the pool should reference its OSTs. This containment must be verified or established by the protocol before claiming the remaining filesystem is consistent. Shared-OST cleanup is covered in section 5.8.

Open question: surviving OST objects may support recovery using parent FIDs, but formatting a replacement MDT at the same index is not sufficient to establish safe identity reuse or file reconstruction. Treat salvage as a separate recovery design.

5.8 Isolated pools without OSTs

An isolated pool may contain MDTs without dedicated OSTs. Its metadata is contained in the pool, while file objects remain on shared OSTs.

- Recovery. Apply the same metadata boundary protocol. Shared OST availability remains a separate dependency; the pool does not gain data failure isolation.
- **Scope.** The failure domain covers metadata only. An OST failure still affects files both inside and outside the pool.
- **Membership.** I2 applies to the MDT members. The OSTs stay shared.

- Drop. Files from the lost MDTs can leave orphan objects on shared OSTs. Cleanup must identify objects belonging to retired metadata identities, avoid live objects, and reconcile block quota.

Open question: layout LFSCK needs an explicit policy for retired MDT identities. Destruction cannot rely only on an MDT index or an assumed current sequence mapping.

5.9 Removing members and pools

Removing a member does not move its objects. Plain-pool changes affect future placement; isolated-pool changes must also preserve containment and recovery information.

- Whole pool. Destroying a plain pool removes its allocation policy. Converting or destroying an isolated pool also ends its failure isolation and requires a controlled transition. An offline lost pool uses pool_drop by default.
- Some members. Existing objects remain on removed targets. That is allowed for plain pools, but breaks isolated containment unless the objects are drained or the pool loses its isolation guarantee.

The following administrative controls are proposed. An unverified pool cannot claim safe failure isolation; the safety of any force override remains a design question.

```
# proposed syntax
lctl pool_set testfs.proj drain_before_remove=on
lctl pool_set testfs.proj online_destroy_only=off
```

Option	When on	When off	Default
`drain_before_remove`	`pool_remove` is refused until the member holds none of the pool's directories or objects. Drain it first with `lfs migrate` (section 6).	A forced removal leaves data in place and marks the pool unverified. Safe pool_offline must remain unavailable until containment and recovery dependencies are re-established; a force bypass needs separate justification.	On for isolated pools, off for plain pools
`online_destroy_only`	Destroying an isolated pool, or removing all its members, is refused while the pool is offline. Use `pool_drop` instead (5.7).	A proposed override ends isolation and restores ordinary target dependencies. It must not be used to bypass lost-target retirement or fencing requirements.	On

6 Migration between pools

Migration moves existing objects between pools or between a pool and ordinary targets. It builds on existing MDT migration and file-data migration through layout swaps. Pool-aware subtree migration and its recovery rules are proposed extensions.

6.1 Operations

```
# proposed syntax
# file data onto the OSTs of pool fast (exists today for OST pools)
lfs migrate -p fast /mnt/testfs/proj/file
# subtree metadata onto the MDTs of pool meta
lfs migrate -m -p meta /mnt/testfs/dir
# whole subtree: binding, metadata and file data
lfs migrate -m -p proj2 --with-data /mnt/testfs/dir
```

Operation	What moves	Rules
File data: `lfs migrate -p P file`	The file's OST objects, by layout swap.	Exists today for OST pools. Into an isolated pool, only for files whose directory is already in that pool (I2).
Subtree metadata: `lfs migrate -m -p P dir`	Directory and file inodes, with any Data-on-MDT data.	Target MDTs are chosen by QoS within the pool. Striped directories are restriped onto pool MDTs. Each migrated directory gets P in its sticky default.
Whole subtree: `lfs migrate -m -p P --with-data dir`	The binding, the metadata and the file data.	Move the binding and both object types selected by the destination. A missing target-type list normally leaves that placement unchanged, except when source isolation requires evacuation from exclusive targets.

6.2 Moving a whole subtree

1. Rebind the subtree to the target pool under the migration protocol. Update its default LMV and any default LOV that still names the source pool. Coordinate concurrent creates so they cannot escape the tracked migration.
2. Move directory and file metadata using MDT migration, with the required directory traversal and locking.
3. Move file data through layout swaps. Busy-file skipping and retry can be supported, but completion must account for every skipped file.
4. Verify actual MDT and OST placement, layouts, and references before declaring completion. A pool-name search alone cannot prove that all objects have left the source targets.

- Resume. Record enough progress to continue after interruption. Re-running the operation should skip completed moves and revisit outstanding objects; existing MDT migration behavior needs validation for this combined workflow.
- During a plain-pool migration, the subtree may temporarily occupy both pools.
- File-data migration uses the client path. Large moves can be partitioned among clients, provided progress tracking and concurrent operations remain coordinated.

- When leaving an isolated pool with dedicated OSTs, evacuate its file objects as well as metadata. If the destination has no OST list, select an explicit permitted data layout outside the source pool. This evacuation requirement takes precedence over the usual unchanged-data default.
- To retire hardware, migrate each affected subtree to other permitted targets, verify the drain and recovery state, then remove the membership or destroy the pool.

Validation required: DoM migration for every supported layout, hard-link handling, open-file behavior, concurrent creates, and restartable progress tracking.

6.3 Isolated pools

Controlled migration is the proposed exception to isolated containment. Its temporary dependencies must be recorded durably and reflected in offline and recovery decisions.

- Migration unit. Move a complete pool root, or a complete directory tree outside isolated pools. Keep the external name entry in place while moving the subtree. Entering an isolated pool creates a pool root; leaving it retires that boundary role.
- Partial subtree moves. A subdirectory cannot retain cross-pool namespace references. Moving only part of an isolated tree requires copy-and-delete semantics or a separately defined detach protocol.
- Transit link. Before migration crosses an isolated boundary, record a durable link between the source and destination. Both must be online. Servers reject migration without a matching link, and the link survives coordinator or MGS restart.
- Recovery during transit. Linked isolated pools temporarily form one recovery domain; offline handling must cover the entire linked group. Migration between a pool and ordinary targets can introduce core dependencies. The original guarantee that the core never waits needs a separate protocol and cannot be inferred from the link alone.
- Close the link only after checking placement, references, pending transactions, and recovery state. Interrupted moves need a defined resume or rollback path; migrating back is not automatically a complete rollback.

```
# proposed syntax
lctl pool_transit testfs.projA testfs.projB      # open the link
lfs migrate -m -p projB --with-data /mnt/testfs/projA
lctl pool_transit -d testfs.projA testfs.projB   # close it after the check
```

7 Compatibility

- Wire and disk. Reuse the reserved LMV pool-name fields where possible. The full persistent and wire-format impact remains to be validated, especially for pool generations, boundary state, and transit links. Proposed additions include:
  - a durable pool-state record, including generation and transition state;
  - an in-memory lds_dir_def_pool field and shared client membership table;
  - a capability flag for pool-aware clients, plus durable transit-link state.

- Old clients. Unsupported MDT members are expected to be ignored through existing pool configuration error handling. Verify this behavior for each supported mixed-version combination.
  - In a plain pool, their mkdir ignores the pool.
  - In an isolated pool, the MDT refuses their creates in the pool's subtree.
  - A pool that lists only MDTs looks to them like an empty OST pool.
  - Their MDT migration does not enforce pool-aware rules; servers must reject operations that would cross an isolated boundary.
- Old servers. Isolated pools require support on every MDT that can participate in their boundary, allocation, or recovery operations. Plain-pool behavior on mixed versions needs explicit validation.
- Downgrade. End isolation through a safe online conversion or permanent-drop procedure before running older code. Resolve transit links and outstanding recovery state first; clearing the pool name alone is insufficient.