Whamcloud

Lustre Nodemap Update

LAD 2025

Marc-André Vef & Sébastien Buisson

ddn

Introduction

Whamcloud

- Multiple (remote) sites may use a single Lustre file system
- Their *User ID* (UID), *Group ID* (GID), and *Project ID** (PROJID) spaces may conflict
- Nodemap is a long-established feature
  - Initially developed to solve ID conflicts by mapping client IDs to a global ID space
  - Introduced in Lustre 2.7 as a technology preview
  - Supported from Lustre 2.9 (2015)
- Today, Nodemap supports many advanced features, including isolation:
  - ID map ranges and ID offsets
  - Role-based Admin Controls (RBAC)
  - Multiple filesets and dynamic nodemaps
  - Banning clients and more …
- This talk will present the latest and upcoming developments of Nodemap

**PROJID mapping support since Lustre 2.15*

The core of Nodemap

Whamcloud

- Lustre distinguishes clients based on their Network Identifier (NID)
- On client mount-time:
  - Lustre checks which nodemap the client NID is part of
  - Strong authentication (Kerberos or SSK) can verify client NIDs to prevent address spoofing
- On FS access:
  - Nodemap's ID mapping policy engine filters access for each client and ID
  - Map a client's ID to their respective canonical FS ID based on the Nodemap's configuration
  - All unknown client IDs are squashed
  - Apply further access conditions, e.g., RBAC
- "Trusted" & "Admin" nodemap used for servers to operate on the canonical FS IDs

Nodemap mapping properties overview

Whamcloud

- Privileged access:
  - `admin`: root remains root
  - `trusted`: no explicit ID mapping, client IDs are kept as-is
  - ➢ `admin` & `trusted`: for servers and administrative clients
- Mapping options
  - `idmap`: client IDs are mapped to FS IDs
  - `map_mode`: enables ID mapping for UID, GID, and PROJID
  - `offset`: client IDs are automatically mapped to ID+OFFSET
- Handling unmapped IDs:
  - `squash_uid`, `squash_gid`, `squash_projid`: IDs are squashed to set value if not mapped
  - `deny_unknown`: denies all access to unmapped IDs

The basics

Whamcloud

- Create ”nm1” nodemap
- NID ranges can’t overlap with any other range across **all** nodemaps

```
mgs $ lctl nodemap_add nm1
mgs $ lctl nodemap_add_range --name nm1 --range 192.168.1.[100-200]@tcp
mgs $ lctl nodemap_add_range --name nm1 --range 192.168.2.[0-50]@tcp
```

- Define ID mappings*

```
mgs $ lctl nodemap_add_idmap --name nm1 --idtype uid --idmap 530:11000
mgs $ lctl nodemap_add_idmap --name nm1 --idtype gid --idmap 530:11000
mgs $ lctl nodemap_add_idmap --name nm1 --idtype projid --idmap 101:1001
```

**PROJID mapping support since Lustre 2.15*

The basics

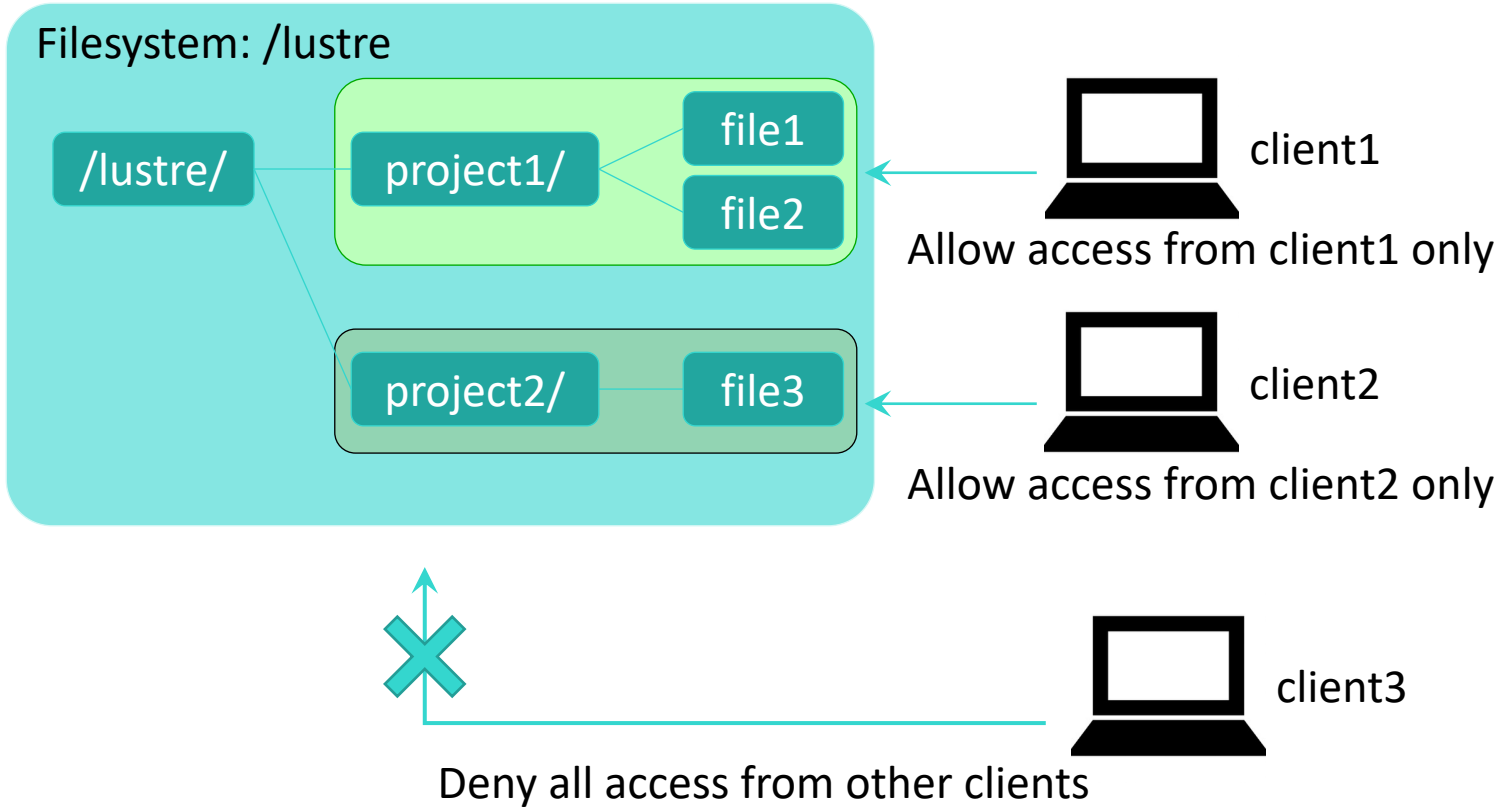
Whamcloud

- The `default` nodemap is used for NIDs that can't be assigned to any nodemap
  - Implementing special behavior compared to other nodemaps
  - It cannot be removed
  - No ID mapping can be defined
  - Be mindful of altering admin and trusted properties
- Enable Nodemap
  - Nodemap activation is done globally and affects all nodemaps
  - Allow time for nodemap definitions to propagate
  - Change events are queued and distributed across the cluster in tens of seconds

```
mgs # lctl nodemap_activate 1
```

Nodemap isolation

- The isolation design includes filesets (subdirectory mount) and ID mapping



ID range mapping

Whamcloud

- LU-17922 (Lustre 2.16) allows declaring idmap ranges
- Extended syntax for `lctl nodemap_add_idmap` command:
  - `<clientid_start> – <clientid_end> : <fsid_start> [– <fsid_end>]`
  - `fsid_end` is optional

```
# Create ID mapping range
mgs $ lctl nodemap_add_idmap --name nm1 --idtype uid --idmap 500-510:10000

# Delete ID mapping range
mgs $ lctl nodemap_del_idmap --name nm1 --idtype uid --idmap 500-510:10000
```

ID mapping offsets

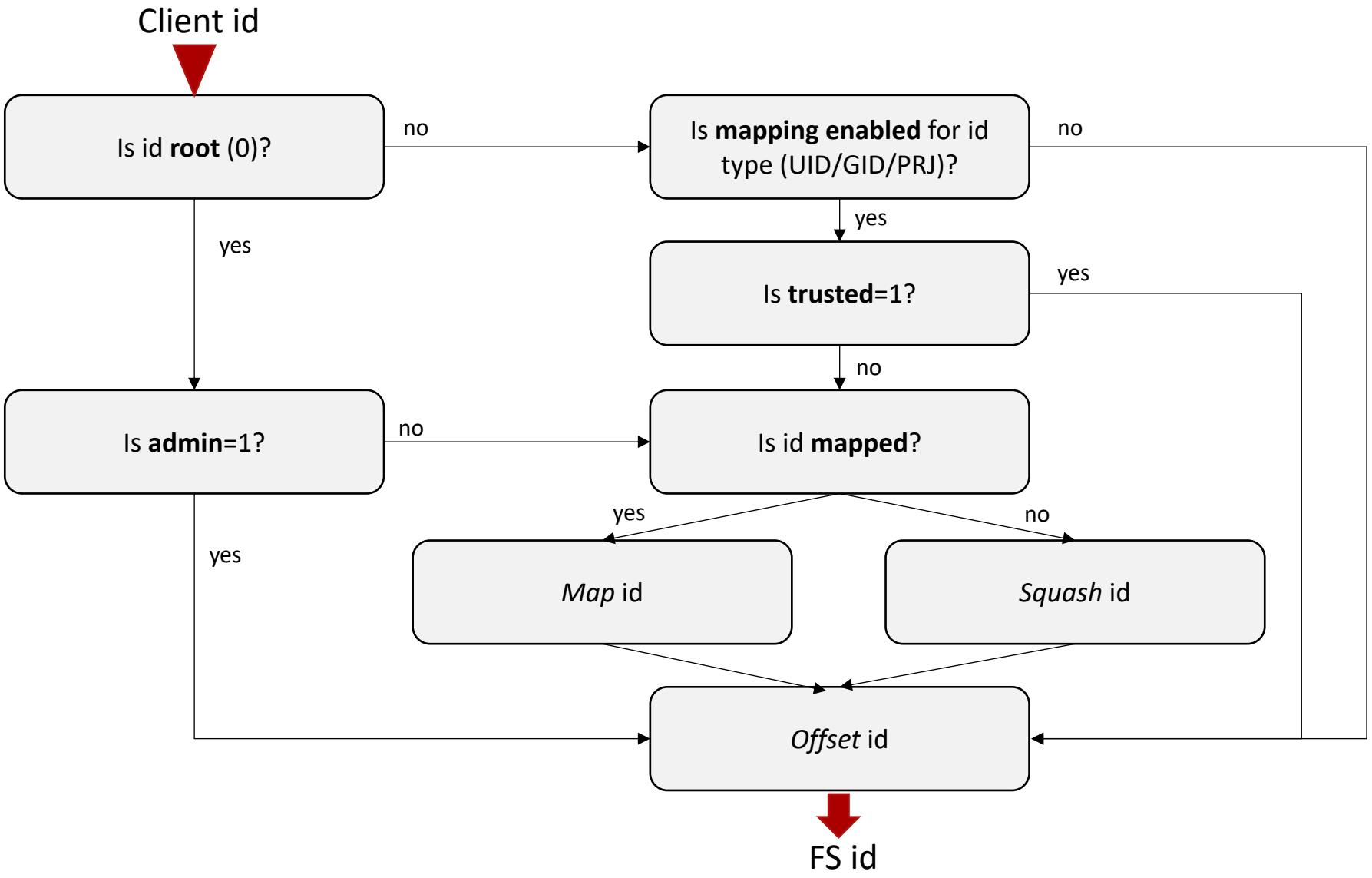
Whamcloud

- LU-18109 (Lustre 2.17) allows ID mapping offsets
- Challenges with individual ID mapping
  - Defining and maintaining mappings for each ID and nodemap can be time-consuming
  - Access or knowledge of nodemap ID may be unknown (Cloud Service Provider)
  - `FS_ID` isolation must be maintained manually
- ID offsets automatically maps each local `client_id` to its global `fs_id` (ID+OFFSET)
  - No need for individual mapping rules
  - Offset ranges cannot overlap with another nodemap
  - ➢ Guarantee for disjoint ID spaces

```
# Map client_ids from range 0-49999 to fs_ids in range 100000-149999
mgs $ lctl nodemap_add_offset --name nm1 --offset 100000 --limit 50000

# Deactivate offsets (does not affect existing mappings on files)
mgs $ lctl nodemap_del_offset --name nm1
```

ID mapping flow chart



Role-based admin controls (rbac)

Whamcloud

- LU-16524 (Lustre 2.16) allows to specify admin capabilities per nodemap
- rbac roles are continuously added (some only for Lustre 2.17)
- Available roles (multiple choices are possible):
  - `byfid_ops`: to allow operations by FID (e.g. `lfs rmfid`)
  - `chlg_ops`: to allow access to Lustre Changelogs
  - `dne_ops`: to allow operations related to DNE (e.g. `lfs mkdir`)
  - `file_perms`: to allow modifications of file permissions and owners
  - `quota_ops`: to allow quota modifications
  - `fscrypt_admin`: to allow fscrypt admin actions
  - `ignore_root_prjquota`: to control if project quota is enforced for root
  - `pool_quota_ops`: to allow pool specific quotas (e.g., `lfs setquota --pool`)
  - `hsm_ops`: to allow HSM actions (archive, release, restore, …)
  - `local_admin`: to keep root capabilities even if mapped
  - `server_upcall`: whether clients should use the server-side defined identity upcall

Deactivate nodemaps

Whamcloud

- LU-18469 (Lustre 2.17) allows to deactivate individual nodemaps
- New mounts from clients associated with a nodemap are rejected
- Mounted clients are unaffected
- The `deny_mount` property controls this feature

```
mgs $ lctl nodemap_modify --name nm1 --property deny_mount=1
```

Preventing client access

- LU-19157 (Lustre 2.17) allows banning clients based on their NID
- Especially useful in environments where client nodes can't be shut down
- Client ban is done on the RPC-layer, but clients are not evicted (force un-mounted)
  - New mount requests and RPCs from mounted clients that access the file system are denied
  - Re-allowing client access is instantaneous, and no re-mounting required
- Preventing client access uses the banlist feature
  - On `default` nodemap: banned NID range cannot overlap with NID range of any nodemap
  - On normal nodemaps: banned NID range must be within NID range of the same nodemap
  - Name is not required: nodemap is derived from banned NID range – if not found, `default` is used

```
# Range must be included in NID range for nm1 nodemap
mgs $ lctl nodemap_banlist_{add,del} --name nm1 --range 192.168.1.[1-254]@tcp

# Nodemap is derived from banned NID range
mgs $ lctl nodemap_banlist_{add,del} --range 192.168.1.[1-254]@tcp
```

Sub-directory mounts and multiple filesets

Whamcloud

- Nodemap fileset restrict clients to a sub-directory introduced in Lustre 2.9
- One fileset may be too limiting, e.g., shared resources required by many nodemaps
- LU-18357 allows multiple filesets on a single nodemap (Lustre 2.17)
  - The “old” fileset is now called `primary` and is still the default fileset (only one primary fileset allowed)
  - Additional `alternate` filesets allow access to additional subdirectories (up to 255 alternate filesets)
  - Each fileset can be defined as read-only (forces `mode=ro` on mount)
- Backward-compatible:
  - The existing fileset is transferred to be a primary fileset
  - Note: Defining filesets through `lctl [–P] set_param` is deprecated and should no longer be done!
- New commands for adding and deleting filesets
- Additional command for in-place fileset modification
  - Rename (change sub-directory path) and change type (primary or alternate)
  - Switch between read-only or read-write mode per fileset

Multiple filesets examples


```
# Create primary and alternate filesets
mgs $ lctl nodemap_fileset_add --name nm1 --fileset /dir1
mgs $ lctl nodemap_fileset_add --name nm1 --fileset /dir2 --alt
mgs $ lctl nodemap_fileset_add --name nm1 --fileset /data_src --alt --ro

# Modify filesets
mgs $ lctl nodemap_fileset_modify --name nm1 --fileset '/dir1' --rename '/dir3'
mgs $ lctl nodemap_fileset_modify --name nm1 --fileset '/dir3' --ro --alt
mgs $ lctl nodemap_fileset_modify --name nm1 --fileset /dir2 --prim

# Delete filesets
mgs $ lctl nodemap_fileset_del --name nm1 --fileset /dir3
mgs $ lctl nodemap_fileset_del --name nm1 --all
```

Multiple filesets

Whamcloud

- The following rules apply when clients mount a subdirectory:
  1. If the nodemap is inactive or no filesets are defined, no subdirectory restrictions are applied.
  2. If the primary fileset is set and no subdirectory is presented when mounting the Lustre client, the primary fileset's subdirectory is used as the file system root directory.
  3. If any defined fileset (primary or alternate) matches the presented mounted subdirectory exactly or as a prefix, the subdirectory mount is used as the file system root directory.
  4. If the fileset matches in 3, the presented mounting subdirectory is appended to the fileset's subdirectory.
- Only alternate filesets allowed: If no primary fileset is set, only 3 applies

```
nodemap.nm1.fileset=
[
    { primary: /dir1 },
    { alternate: /dir2 },
    { alternate: /data_src, mode: ro }
]
```

Dynamic nodemap

- LU-17431 allows dynamic nodemaps (Lustre 2.17)
- For ephemeral use cases that require frequent nodemap changes, e.g., in compute jobs
- The current global mechanism requires synchronization time and is persistent
- Dynamic nodemaps are…
  - in-memory only and must be created directly on the servers,
  - hierarchical with child-parent relationship including permission control, and
  - are removed on server restart or global nodemap changes.
- Dynamic nodemaps refine the generic behavior of the parent nodemap
- External orchestration recommended to set dynamic nodemaps on all MDSs and OSSs
  - `clush –a lctl nodemap_add –d`

Dynamic nodemap and permissions

Whamcloud

- Only dynamic nodemaps can have parents
- Parent-child relationship for dynamic nodemaps:
  - A child must have a at least one persistent nodemap parent, possibly `default`
  - A child's NID ranges must be within the parent's NID ranges
  - A child inherits all properties and ID mappings from the parent on creation
- A child can only lower privileges from the parent unless it grants permission to raise
- New nodemap property child_raise_privileges defines properties a child can raise:
  - admin, trusted, deny_unknown, readonly_mount, forbid_encryption, caps, child_raise_privs
  - Also accepts all RBAC roles in additional
  - Defaults to `none` and can be set to `all`

Dynamic nodemap commands

Whamcloud

```
# -d for dynamic; -p for parent nodemap
mgs $ lctl nodemap_add -d -p nm1 subnm1
# NID range must be within parent's NID range
mgs $ lctl nodemap_add_range --name subnm1 --range 192.168.1.[100-150]@tcp
# Other properties are inherited from parent nm1, e.g.,
nodemap.subnm1.admin_nodemap=0
nodemap.subnm1.fileset=/remote_dir
nodemap.subnm1.idmap=
[
{ idtype: uid, client_id: 530, fs_id: 11000 },
{ idtype: gid, client_id: 530, fs_id: 11000 },
{ idtype: projid, client_id: 101, fs_id: 1001 }
]
nodemap.subnm1.rbac=file_perms,dne_ops,quota_ops,byfid_ops,chlg_ops,fscrypt_admin
nodemap.subnm1.squash_gid=65534
nodemap.subnm1.squash_projid=65534
nodemap.subnm1.squash_uid=65534
nodemap.subnm1.trusted_nodemap=0
```

Dynamic nodemap and multiple filesets

Whamcloud

- Parent nodemap namespace restrictions through filesets are passed on to children
- With multiple filesets there are several considerations:
  - If the parent has filesets defined, a child must keep at least one
  - Additional filesets on the child can only further restrict the namespace
  - Read-only filesets cannot be set read-write (vice-versa allowed)
  - There are no restrictions which fileset must be primary or alternate
- Note, a nodemap's `readonly_mount` property overwrites fileset read-only settings
- If the parent has no filesets, the child has no restrictions

Further improvements

Whamcloud

- Quality-of-life improvements (Lustre 2.17)
  - `lctl nodemap [cmd]` format is now possible
  - `lctl nodemap_modify --name NODEMAP_NAME --property deny_mount=1` is now possible
  - `lctl nodemap_info` improvements:
    - The arguments `--name` and `--property` were added
    - Prints all nodemaps by default
    - Now fully supports YAML-format
- OST object tagging with parent FID and UID/GID/PROJID (Lustre 2.17+)
  - Allows scanning the OST and getting the MDT FID from objects, e.g., when migrating files off an OST
  - Check OST objects against nodemap ID/offset mapping
- Per-nodemap capabilities mask (Lustre 2.17)
  - New command: `lctl nodemap_set_cap` (instead of the global `enable_cap_mask` parameter)

```
# Define a capability mask as "cap_chown" on nodemap ‘nm1':
mgs $ lctl nodemap_set_cap --name nm1 --caps cap_chown --type mask
```

Closing

Whamcloud

- Nodemaps were initially developed to solve conflicts on the ID space
- Today, Nodemap is the core mechanism to support isolation
  - ID mapping with automatic offsets
  - Dynamic nodemaps and multiple filesets
  - A rich interface to control client permissions and capabilities through nodemaps
- Several new features are introduced in Lustre 2.17. The highlights:
  - ID mapping offsets
  - Additional RBAC controls
  - Deactivation of nodemaps and banning clients
  - Ephemeral dynamic nodemaps including privileges control
  - Multiple filesets
- Nodemap is still evolving with further features planned

Whamcloud

Thank you!

Marc-André Vef - mvef@whamcloud.com

Sébastien Buisson – sbuisson@whamcloud.com

ddn