INDIANA UNIVERSITY

High Level Design For UID/GID Mapping

Revision History

Date	Revision	Author
12/18/2012	1	jjw
01/21/2013	2	jjw
02/04/2013	3	jjw

I. Introduction

UID/GID Mapping allows the Lustre file system to be used across clusters' heterogeneous user populations while still respecting POSIX file ownership, permissions, ACLs and quotas in a manner without credential timeouts or methods to transfer credentials across a client cluster.

II. Definitions

Cluster

A distinct set of Lustre client NIDs that can be treated as a single unit with regards to UID/GID namespace. No NID can belong to two separate clusters on the same file system.

File system UID/GID

The UID/GID that represents the owner/group of the file or directory as it is stored on the Lustre file system.

Client UID/GID

The UID/GID that represents the owner/group of the file or directory as it is represented on a Lustre client.

Canonical UID/GID Map

The UID/GID map held on the MGS server that serves as the master.

Null Cluster

The default cluster acts as a container for all clients that are not included in an explicitly defined cluster definition. The null cluster is configurable in all parameters except for NID ranges, as all undefined ranges are encapsulated within the null cluster.

Working UID/GID Map

The UID/GID map held in memory on the MDS/OSS nodes.

III. Requirements

Toggle and Configuration [idmap.toggle]

UID/GID Mapping must be available as an option. When toggled off, the behavior of the file system will be as it is currently.

UID/GID Mapping must provide configuration parameters to specify whether a particular NID is to have its UID/GID space mapped to the canonical set or the UID/GIDs in the request (and subsequent response) are to be trusted.

A configuration parameter must be provided to specify for a particular NID what UID/GID to map to a UID/GID contained in a request and subsequent response if it does not exist in the map.

A configuration parameter must be provided to specify whether a particular NID will allow (if its UID/GID space is trusted) UID 0 to perform operations on the file system. This will supersede the existing root squash mechanism.

ID Mapping [idmap.idmap]

UID/GID Mapping must provide the ability for MDS and OSS nodes to translate between client UID/GIDs and canonical UID/GIDs to permit clients with a heterogeneous UID/GID space to mount the file system while maintaining and respecting POSIX file ownerships, permissions, and ACLs. Canonical UIDs and GIDs must be maintained on the OSS nodes to respect and maintain quotas.

Management Interface [idmap.management]

UID/GID Mapping must provide a single master copy of the map on the Management Server, where each Metadata Server and Object Storage Server operates from a copy of the map that is kept in sync with the master. Administrative operations must take place only on the master copy of the map. The management interface must be able to perform the following operations while the file system is still mounted by clients.

- Add NIDs or groups of NIDs to the map
- Remove NIDs or groups of NIDs from the map
- Add a UID or GID to the map
- Remove a UID or GID from the map
- Force an update of the MDS and OSS with a new version of the map
- Dump the map on the MGS
- Force a complete reload of the map on the MDS or OSS

Performance [idmap.performance]

The UID/GID Map must be structured to ensure that lookups to the map do not adversely affect the file system operations within which they are taking place.

Memory Usage [idmap.memory]

Both the canonical UID/GID Map and the working copies must be structured to minimize memory consumption

IV. Design Highlights

An additional kernel module (idmap.ko) will be written to provide exported functions that can categorize connected clients into clusters and provide forward and reverse mapping given the cluster and UID/GID. The idmap.ko module also provides an end point for PTLRPCs necessary to communication with the MGS to keep the map updated.

V. Functional Specification

The general idea behind UID/GID mapping is an extension of root squash. Root squash, is simply a specific case of UID/GID mapping such that UID 0 is, for all but a specified set of administrative clients, translated to a UID that has limited access to the file system. UID/GID mapping is an extension of that idea to allow any client UID or GID to be translated to a canonical UID or GID for file system operations, and translated back to the client's view of UIDs and GIDs upon completion of the operation so that they client's state is consistent in it's view of the permissions of file system objects.

Clusters

Clients are partitioned by NID ranges into groups called clusters. Clusters are defined by these NID ranges, and include other configuration information that specifies the behavior of members of the cluster with regards to file system access. Clients that cannot be categorized into a defined cluster are added to a default null cluster. No client can be contained in more than one cluster.

Included in this definition are forward and reverse maps for UIDs and GIDs, a default UID and GID for requested that have UIDs and GIDs for which there is no explicit mapping, flags for controlling whether the file system allows client UIDs and GIDs to be passed without transformation, whether UID 0 file system requests from the cluster, a list of actively mounted client NIDs, and a hash containing shared keys for use with GSSAPI and security flavors.

The null cluster has a set of sane default values that minimize file system access. File system administrators can change the configuration of the null cluster with the exception of adding or removing NID ranges. Since the null cluster contains all clients that are not otherwise partitioned, no explicit ranges can be permitted and still maintain its function.

To maintain a consistent state of the file system with regards to permissions and quotas, each MDS and OSS node must have the ability to properly map UIDs and GIDs.

File system administrators will maintain cluster definitions on the MGS, which will update the MDS and OSS servers via LNET RPCs. When clusters are enabled before configurations, there is a default null cluster that encapsulates all clients that are not explicitly included in other defined clusters. When no other clusters are defined, all clients are included in the null cluster.

Client Mounting

When the client makes its initial connection to an MDS or OSS node, the NID of the incoming client is looked up in the cluster definitions, and a pointer to the correct cluster definition is attached to the client export structure, to prevent lookups on every operation. Additionally the client NID is added to a list of active NIDs kept in the cluster, to keep from needing to walk the entire export structure when the cluster definitions are updated.

UID/GID Mapping

When a request is made of the MDT, the transformation from client UID/GID to canonical UID/GID will be handled at the unpacking of the request at the beginning of its lifetime, and transformation from canonical UID/GID to client UID/GID when the response is packed.

When a request is made of the OST, the UID/GID transformation needs to occur prior to a write and any quota operations, as the only reason for UID/GID information on the OST is to maintain quotas.

VI. Logical Specification

Cluster Kernel Module

An additional kernel module will be written to manage an in memory representation of the client clusters and their UID/GID maps. This module will be present on MGS, MDS and OSS nodes, and provide the functions by which other modules that handle incoming requests can use the NID of the remote client to properly categorize it into a pre-defined client cluster.

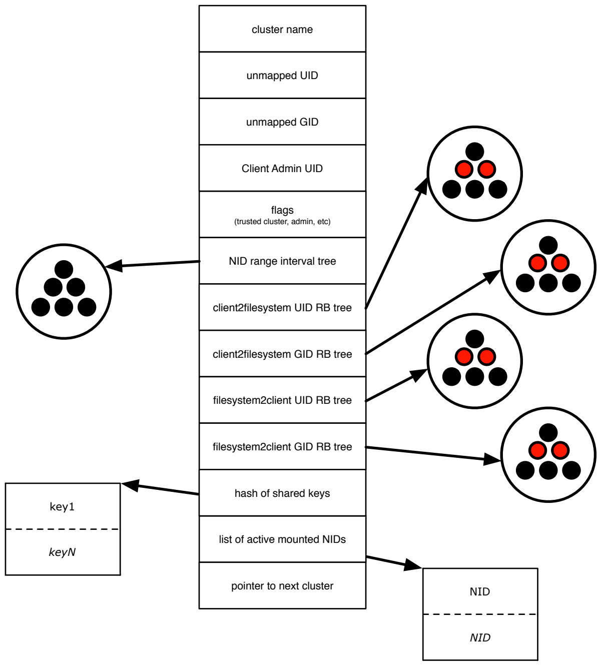
Each cluster will include a textual name for administrator identification, a unique identifier, a file system UID and GID to assign to unmapped client UIDs and GIDs, a client admin UID (for future work), flags indicating whether the cluster is trusted and the client UIDs and GIDs that should be used by the file system.

Each cluster definition contains a pointer to an interval tree representing one or more NID inclusive ranges that comprise the cluster. Any NID can be contained in only one range for only one cluster definition in the latest version of the cluster lists, enforced by the cluster kernel module. The cluster definition also contains a list of currently mounted NIDs, for use in re-evaluating cluster membership upon update of the cluster definitions.

Each cluster definition will also contain an hash of shared keys to be used by the Shared Key GSSAPI mechanism.

Each cluster definition contains four pointers to red-black trees that provide fast forward and reverse mapping between client and file systems UIDs and GIDs.

Additional space in the cluster definition may be defined later for shared key derived session keys, etc.


cluster name
unmapped UID
unmapped GID
Client Admin UID
flags
(trusted cluster, admin, etc)
NID range interval tree
client2filesystem UID RB tree
client2filesystem GID RB tree
filesystem2client UID RB tree
filesystem2client GID RB tree
hash of shared keys
list of active mounted NIDs
pointer to next cluster
key1
keyN
NID
NID

The following defined constants to determine whether a function is mapping UIDs to GIDs from client to the canonical or from the canonical to the client (direction).

```
#define CLUSTER_FILESYSTEM 0
#define CLUSTER_CLIENT 1
```

The cluster kernel module will export the following functions:

```
struct cluster *cluster_get_cluster(struct lnet_nid_t nid)
```

Accepts an incoming NID, presumably of a mounted client, and returns a pointer to the cluster definition. If the NID cannot be found within the defined ranges of a cluster, a pointer to the null client is returned.

uid_t cluster_lookup_uid(struct cluster *c, uid_t uid, int direction)
gid_t cluster_lookup_gid(struct cluster *c, gid_t gid, int direction)

Accepts a pointer to a cluster definition, a uid_t or gid_t, and an integer indicating the direction of mapping (client to canonical: 0 or canonical to client: 1) and returns the mapped uid_t or gid_t.
If there is no map for the uid_t or gid_t that is passed, the unmapped uid_t or gid_t is returned. If the uid_t or gid_t passed is 0, and the flags are not set defined the cluster as having admin access, the unmapped uid_t or gid_t is returned.

If the flag is set such that the cluster is trusted, the incoming uid_t or gid_t is returned, unless the passed value is 0, and then policy regarding the cluster having the admin flag set are used to determine the return value.

The cluster module will also provide a procfs interface on all server nodes (MGS, MDS, and OSS nodes) to provide information about the cluster definitions and UID/GID mappings.

Maintenance of Cluster Definitions on the MGS

The MGS will maintain an in memory image of current cluster definitions and their associated UID/GID maps, including a unique version identifier. The MGS will also maintain a representation of previous iterations of the cluster definitions and associated UID/GID maps such that a MDS or OSS node with an outdated version of the map will be able to request an update, along with their current version, and be provided with a delta of the MGS and [MDS | OSS] version.

Alterations to the cluster definition and UID/GID map on the MGS will be made to a private in memory version, such that the changes can be validated within the constraints of the data structures and policies. Changes to this private memory version can be committed after validation. Committing will apply the changes to the current memory version (for use in updating clients) and made to the on disk version. Only the latest valid version of the cluster definitions and the associated UID/GID mappings will be written to disk. If a configuration in private memory does not validate, the changes will be discarded and the cluster definitions and UID/GID maps restored to the current valid version.

Rather than keep prior versions of cluster definitions and UID/GID maps, each version will be kept as a list of changes from the previous version, as the use is for updating older version of the cluster definitions and UID/GID maps and not rolling back to previous versions. Changes to the private memory version are kept in a private memory log of change commands, which is truncated when either validation fails or the cluster definitions and UID/GID maps are committed and the changes copies to the version log.

Communication between userspace configuration commands via lctl and the MGS will be done via an additional ioctl, that will be accessed from userspace via llapi interface functions. Additional logs will be written to CONFIG on the MGT. The extra CONFIG logs will have the naming convention of cluster-<CLUSTER_ID>.

```
#define OBD_IOC_CLUSTER      IOWR('f', <VALUE>, OBD_IOC_DATA_TYPE)
```

The OBD_IOC_CLUSTER ioctl will honor the following commands:

LCFG_CLUSTER_ADD

The takes a configuration representation from the obd_ioctl_data structure, parses it, and adds the definition to the in private memory version of the cluster definitions. The addition and its passes values are added to the private memory cluster change log.

LCFG_CLUSTER_DEL

Deletes the cluster from the private memory cluster definitions. The deletion is noted in the private memory cluster change log.

LCFG_CLUSTER_ENABLE

Enables cluster definitions and UID/GID mapping.

LCFG_CLUSTER_MODIFY

Alters the configuration of the private memory cluster definitions. The update is noted in the private memory cluster change log.

LCFG_CLUSTER_COMMIT

Commits the private memory cluster definition and its UID/GID maps and its cluster version change log. The changes will be applied to the current version in memory and its version identifier incremented. The changes will be written to disk, the cluster definition change log updated, and the private memory cluster change log truncated.

Transfer of Cluster Definitions from MGS to MDS/OSS Nodes

From the point of view of the MGS, the cluster definitions are an additional log file to be managed in the same manner as other configuration logs. The same end points should be used on the MGS to manage the transfer of the definitions.

In the mgc.ko kernel module, in mgc_process_log(), when the LCFG_LOG_START gets the configuration logs from the MGS, it will additionally grab the cluster definition log, add it to the logs that are watched and have a lock enqueued on the MGS. The configuration files will then have a local copy on each MDT and OST.

The idmap.ko will process these files to create the in memory representations of the cluster definition for use in processing requests for the policy defined in the cluster definition.

Because the mapping of UIDs and GIDs impacts the security of the file system, if clusters are enabled for a file system, no file system requests will be processed by the MDS or OSS unless the MGS is available and the latest cluster definitions and UID/GID maps are have transferred.

Once the MDS/OSS node has a copy of the current version of the cluster definitions, the MGC on the MDS/OSS makes a read-only request for the cluster definition resource. This necessitates adding an additional lock resource to the MGS. The MGC will provide a blocking callback to the lock manager. When the cluster definitions are updated on the MGS, the locks are revoked, triggering the blocking callback, which will contact the MGS and request the changes between the version on the MDS/OSS node and the current version on the MGS. If the changes between the versions are not available (because of MGS restart, or perhaps an older version that has been purged from the cluster definition change log), the entire cluster definition and UID/GID map will be transferred.

A new operation will be added to mgs_handle() called MGS_CONFIG_DELTA to handle the new request from the MDS/OSS node to produce changes between a version passed as a parameter and the current version on the MGS. Efforts will be made to make this a generic interface for passing updated configuration between MGS and MGCs on the MDS/OSS nodes. Similarly, an additional function will be added to the mgc.ko module to allow it to call the additional operation on the MGS.

Client Connections to the MDS and OSS Nodes

Upon client connection to the MDT, in mdt_connect(), and after a successful call to target_handle_connect() and mdt_init_sec_level(), the incoming NID can be used to lookup the client cluster in the cluster definitions. The mdt_init_sec_level() should complete here so that if there is a relevant security context, via GSSAPI, a client cannot falsely identify itself as a cluster NID to gain access to files and directories for which it would not otherwise have access.

This should be done via a call to idmap_get_cluster() and have a pointer to a cluster definition returned.

The pointer to the cluster definition should be held in the export structure for lookup upon file system request.

Similarly in the ost_handle() function, after the target_handle_connect() and ost_init_sec_level(). The pointer to the cluster definition will be held in the export structure for lookup upon file system request.

Recovery of MGS, MDS and OSS Nodes

If idmaps are enabled, recovery of clients on an MDS or OSS node cannot begin before MGS is available and the latest cluster definition map is available on all nodes. Until these conditions are met, recovery will block.

If the cluster definitions are available on an OST or MDT, the mdc.ko can request an update of the definitions rather than an entre copy. If the definitions do not exist, an entire copy will be necessary. In both operations read-only locks should be requested to ensure that the recovered server receives updates.

Mapping of a Request on the MDS

To ensure that UID/GID mapping occurs prior to any other file system operation for a request, the mapping will occur during the unpack function for each operation defined within the MDT. It should take place as an alternative to the idmap functionality available with Kerberos GSSAPI. Toggling the idmap.ko module to active will allow cluster UID/GID mapping to supersede the idmap linked list that can be populated by the mdt_identity upcall. All functions in mdt_lib.c suffixed with "_unpack" will be altered. Additionally POSIX ACL handing routines in obdclass/acl.c will be altered to handle UID/GID mapping routines.

The quotactl functions in the MDT will also be altered to handle UID/GID mapping.

Similarly, upon packing attributes for a reply, the reverse mapping is applied to all file system objects returned.

Mapping of a Request on the OSS

Handling write requests on the OSS is less complex, since the UID/GID only need be changed prior to a write and any quota handling. Since UIDs are only written to the OSTs to manage quotas, and do not provide any additional file permission protection, UID mapping does not need to occur at all for outgoing replies or from any file operation other than writes.

Additionally, changes will need to be made in osd-ldiskfs and osd-zfs to ensure that inodes contain the proper canonical file system UIDs and GIDs.

VII. State

The cluster module and UID/GID mapping functions can be in any of the following states:

ACTIVE, denoting that each request to the MDS and OSS nodes is unpacked, and the UID and GID are transformed according to the policy defined by the cluster for which the client NID is a member. Additionally, outgoing information contains UIDs and GIDs will be transformed according to the policy defined by the cluster definition.

DISABLED, denoting that regardless of the cluster definitions, no transformation will take place on the UID and GID of the incoming request.

VIII. Use Cases

Inter-Organization Collaboration

Researchers from multiple sites are analyzing large images taken from a telescope using local compute resources. The compute resources are managed separately by each researcher's organization, and each site's policies differ in terms of account management. Copying the images back and forth would be time consuming and inefficient. Storing the images on a Lustre file system with UID/GID mapping and mounting the file system on resources local compute resources allows researcher to analyze the data in near real time, with file ownership and group permissions respected.

Inter-Department Collaboration

Researchers in the Chemistry and Computer Science departments of a large university have received funding for small cluster to perform interdisciplinary research. Both researchers want to make the data that they've collected available on resources local to the Chemistry and CS departments, which are managed by each department staff.

By mounting a centrally managed file system on each of the resources, and mapping the UIDs and GIDs to the appropriate values, allows each researcher to analyze and share data without exposing data owned by any other user or group.

Personal Workstation

A graduate student with a Linux workstation wants to use her advisor's data for visualization. Her hard drive would not be large enough to contain the data necessary for the visualization. By mounting the university's central Lustre file systems and using UID/GID mapping, her workstation allows her to easily perform the necessary work on the data set to produce visualization results.

Batch Job Processing

A compute resource has a Lustre file system mounted from another site across the wide area network. The workload on the compute resource is substantial and jobs need to be submitted to the queue several weeks in advance. A researcher can submit his or her job with data stored on the remote Lustre file system without needing a work-around to manage stored Kerberos credentials if the file system is mounted and UID/GUID mapping enabled and configured.

IX. Analysis

Scalability

Data structures for the data lookups were chosen to reduce the cost of lookups in the UID/GID maps while maintaining a low memory profile. In preliminary tests lookups upwards of 50000 mappings per second were measured with no appreciable CPU load.

Rationale

Other designs were considered, especially ones that used the idmap upcall facility that is in current versions of Lustre to provide Kerberos principle to UID mapping. Because this is done on an individual NID basis rather than a partitioned client cluster, this avenue was not pursued. Additionally, any method that does not use static UID/GID mapping does not solve the problem inherent with long running batch processing systems.

X. Deployment

Compatibility

As all changes to Lustre to support client clusters and UID/GID mapping are transformations on the server side, client compatibility with clusters enabled will be the same as compatibility of the version would be with no idmap.ko module included.

Installation

Component will be installed with all other Lustre components via make install or encapsulation of the make install via a package manager.

XI. Configurable parameters

Configurable parameters are all defined by file system administrators via the lctl command line interface. New switches to lctl will be documented in the man page as well as changes to Lustre Operations Manual.

XII. API and Protocol Changes

Additional Lustre RPCs will be added on the MGS and MGC to facilitate updating the cluster definitions with changes rather than entire configuration file copies. This was explained in Section VI outlining work done to keep the cluster definitions updated.

Also, an additional lock resource on the MGS, as the current namespace covers all configuration logs.

XIII. Open issues

None

XIV. Risks and Unknowns

None