Using UID Mapping in Lustre 2.7 and GSS Shared Key Update

Stephen Simms, Jeremy Filizetti, Chris Hanna,
Nathan Lavender, Kit Westneat

High Performance File Systems
Indiana University

Lustre User Group
Denver
April 13, 2015

RESEARCH TECHNOLOGIES
INDIANA UNIVERSITY
University Information Technology Services

PERVASIVE TECHNOLOGY INSTITUTE
INDIANA UNIVERSITY

UID Mapping

- IU's Software Development Contract with OpenSFS
  - IU developed UID Mapping solution now called "nodemap"
    - Coordinates client access across administrative domains
    - Allows workflows involving geographically distributed resources
    - Enables easy data sharing to facilitate collaboration
    - Misnomer to think of this as "WAN" code
  - This portion of the talk will detail the basics of using the nodemap feature in its current state

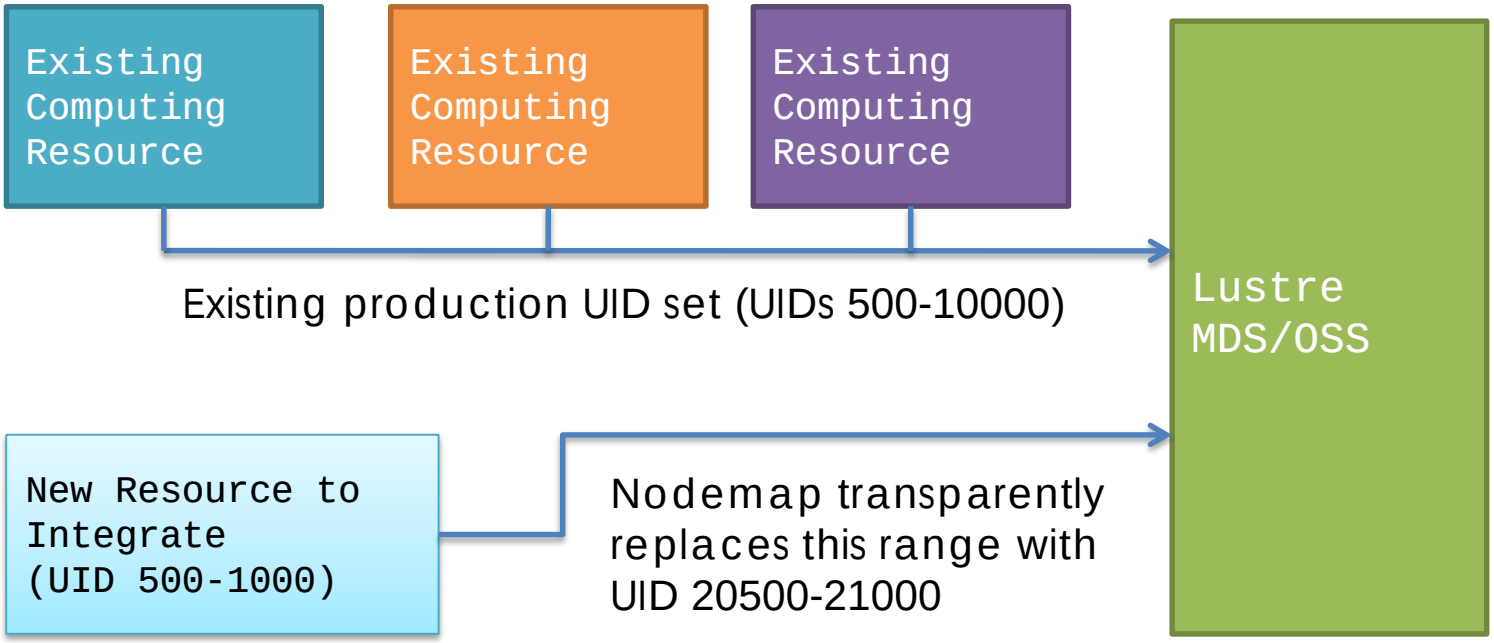
What does Preview Mean?

- The core nodemap features are in 2.7
  - Administrative tools are coming soon

- Currently
  - Nodemap commands need to be run on both MDS and OSS nodes and configurations do not persist between module reloads
    - Scripts can be used to redeploy nodemap configuration
    - Map synchronization will be in 2.8
  - Maps can only be listed through /proc
    - an lctl command will be in 2.8

- We would like your help to test this feature in the wild and share your thoughts and experiences

Why Map?



A mature Lustre system may have UID and GID sets that conflict with another system being integrated. Nodemap enables that system to be seamlessly mounted to the existing Lustre installation.

Nodemap Elements

Nodemap is deployed on the MDS and OSS nodes and is invisible to clients. Key elements include:

- NIDs, to which a unique *mapping* is defined
- *Policy groups*, which consist of one or more sets of NIDs
- Two *properties*, "trust" and "admin", which can optionally be applied to a policy group
- A collection of identity maps or *idmaps* which determine the translation table for a policy group

Configuring Nodemap

- Before activating the nodemap feature, create a policy group for your Lustre servers

```
#lctl nodemap_add LustreServers
```

- Add the NID numbers (or range) of your servers
  - 1 MDS, 2OSS, 1 Management Client

```
#lctl nodemap_add_range --name LustreServers -
-range 192.168.4.[1-4]@tcp
```

Configuring Nodemap (cont'd)

- Set properties for Lustre servers
  - admin = exempt from root squash (on by default)

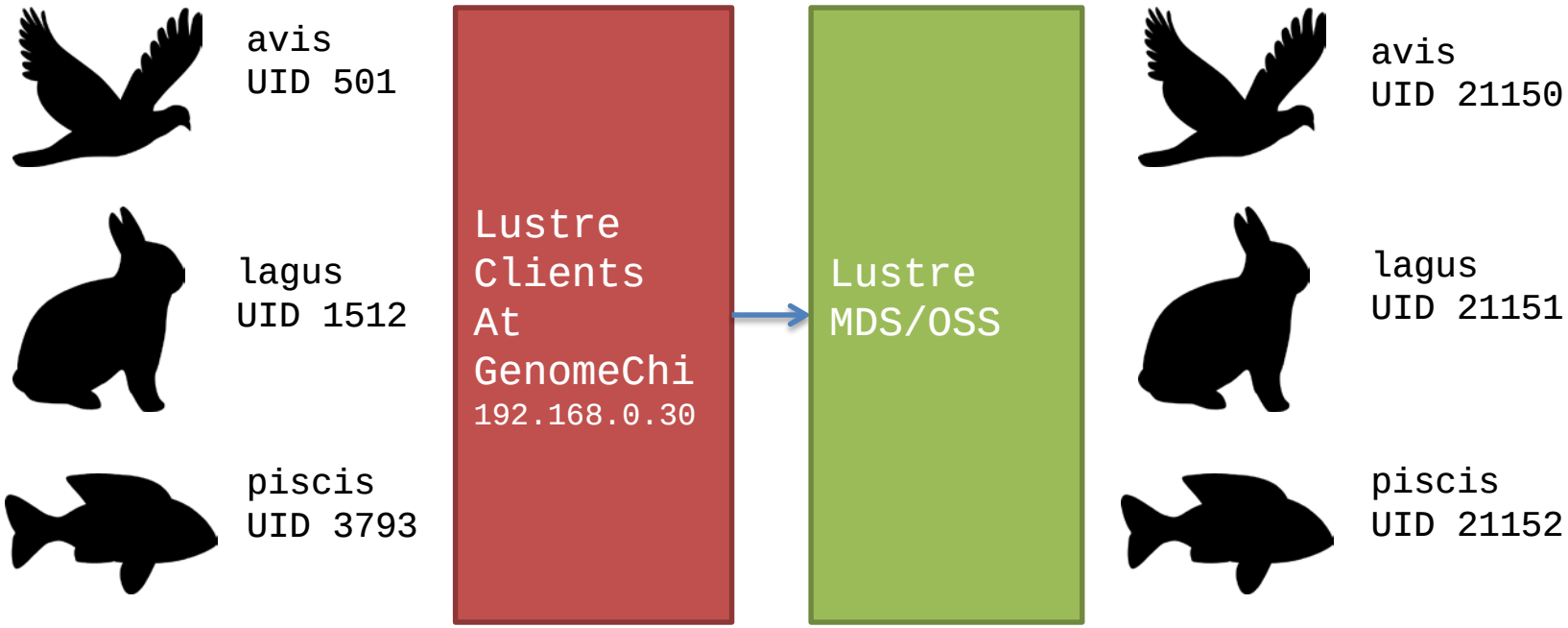
```
#lctl nodemap_modify --name LustreServers --property
admin --value 1
```

  - trusted = used for a policy group that is unmapped

```
#lctl nodemap_modify --name LustreServers --property
trusted --value 1
```

Basic Mapping

To span administrative domains, a nodemap idmap translates client UID / GIDs to canonical UID / GIDs to prevent namespace collision.



Basic Mapping Commands

- Create a policy group for machine GenomeChi

```
# lctl nodemap_add GenomeChi
```

- Add GenomeChi's NID to the policy group

```
# lctl nodemap_add_range --name GenomeChi --range 192.168.0.30@tcp
```

- Populate GenomeChi's idmap with the three users to be mapped nothing that not all users of GenomeChi need to be mapped, only those who will be using the Lustre file system.

```
# lctl nodemap_add_idmap --name GenomeChi --idtype uid --idmap 501:21150
# lctl nodemap_add_idmap --name GenomeChi --idtype uid --idmap 1512:21151
# lctl nodemap_add_idmap --name GenomeChi --idtype uid --idmap 3793:21152
```

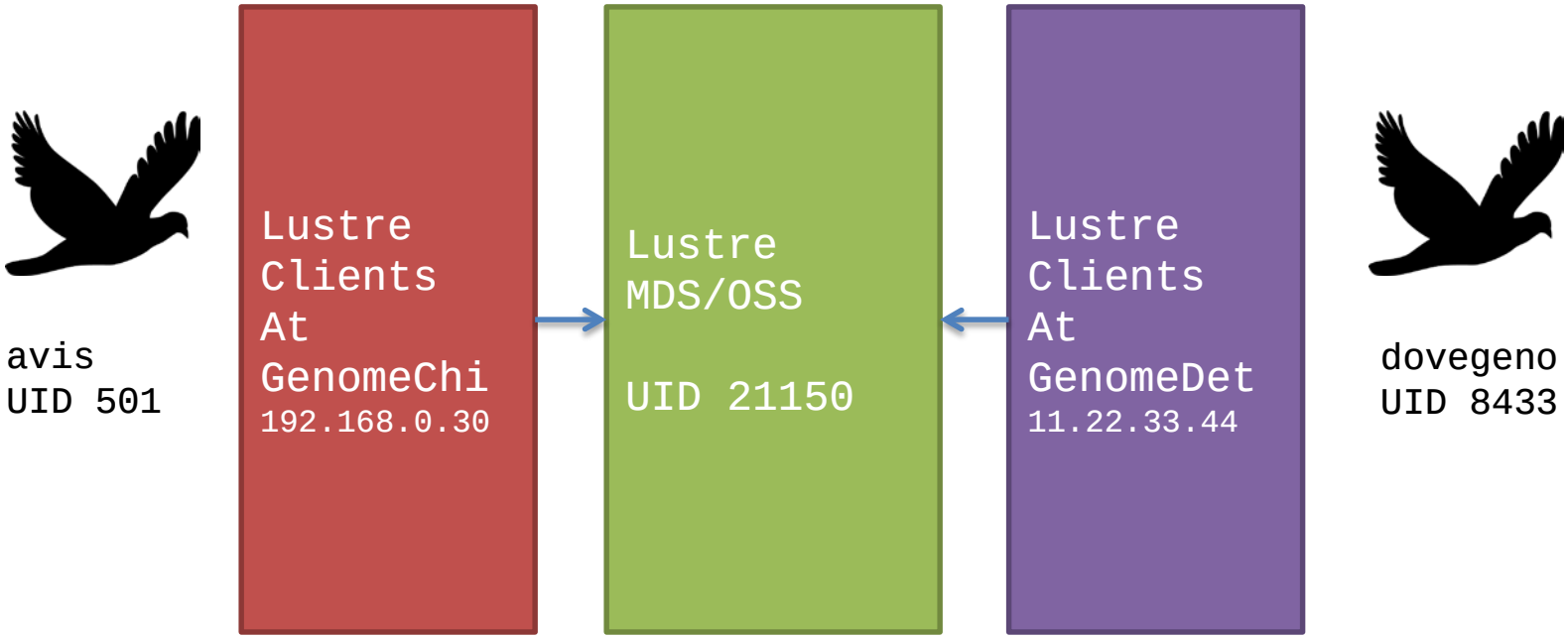
Listing Maps in 2.7

- To examine the contents of the GenomeChi idmap

```
# cat /proc/fs/lustre/nodemap/GenomeChi/idmap
[
 { idtype: uid, client_id: 501, fs_id: 21150 },
 { idtype: uid, client_id: 1512, fs_id: 21151 },
 { idtype: uid, client_id: 3793, fs_id: 21152 }
 ]
# cat /proc/fs/lustre/nodemap/GenomeChi/ranges
[
 { id: 1, start_nid: 192.168.0.30@tcp, end_nid:
192.168.0.30@tcp }
]
```

Example: Mapping Two Identities to One

Two user identities can be mapped to a single identity. The below example shows Mr. Bird using his accounts on two different systems to access a single Lustre filesystem.



Two to One Mapping Commands

- Create a second policy group called GenomeDet

```
# lctl nodemap_add GenomeDet
```

- Add a NID for GenomeDet

```
# lctl nodemap_add_range --name GenomeDet --range
11.22.33.44@tcp
```

- Add an idmap for the user

```
# lctl nodemap_add_idmap --name GenomeDet --idtype uid --
idmap 8433:21150
```

This map is kept independently of the map for GenomeChi

GSS Shared Key Security

- IU developing Shared Key Security solution using GSS
  - Think of GSS as a vacuum cleaner
    - Kerberos is an attachment
    - Shared key would be another GSS attachment
  - Shared key as an alternative to kerberos
    - File System admins shouldn't have to run a KDC
    - Kerberos shops adding new service can be difficult
    - Politics can affect kerberos cross-realm

Shared Key Mechanism and Flavors

- Two modes of operations supported
  - Shared Key Integrity (ski)
    - Shared key for HMACs for assurance of message integrity
  - Shared Key Privacy (skpi)
    - Uses Two keys
    - Shared key for HMACs (integrity)
    - Generated session key using Diffie-Hellman (privacy)
    - Provides Perfect Forward Secrecy

GSS

- Security context initialization through userspace upcalls

- Client Side
  - Uses /usr/sbin/lgss_keyring
  - Called from the kernel key ring request_key binary
  - Requires setting up a file in /etc/request.d named after the key_type's name (lgssc)

```
# cat /etc/request-key.d/lgssc.conf
create  lgssc  *     *     /usr/sbin/lgss_keyring %o %k %t %d %c %u %g %T %P %S
```

GSS (cont'd)

- Server Side
  - Uses /usr/sbin/lsvcgssd
  - Reads and writes to a proc file
  - Must be running or SEC_CTX_INIT RPCs will be missed

- Basic Flow
  - Requests received
  - Unpacked in sptlrpc_svc_unwrap_request
  - Instantiated through upcalls in sunrpc caching layer
  - Upcall handles mechanism specific initialization

Current GSS Work

lgss_keyring and lsvcgssd code
- Restructuring some existing code
  - Upcall passes the mechanism type
  - Determines which service handler to call

lgss_keyring work complete
lsvcgssd work nearing completion and untested

Loading Keys
- Uses a mount.lustre command option
  - File on Client
  - Directory on servers (multiple keys/clusters)
  - Parses and adds keys to kernel keyring

mount command is next on the agenda

Thank you!

Questions?