High Level Design of v2 for Remote ACL

Fan Yong

2007-08-08

1 Introduction

In fact, there is <**High Level Design of Remote ACL**> already (called “Remote_ACL_HLD_v1”), and we have implemented remote_acl in branch “b_new_cmd” based on Remote_ACL_HLD_v1. But such design has some defects as discussed following:

1. Confusing uid/gid policy.

   The key issue is which uid/gid should be used for remote operations (uid/gid related ones, such as stat/chown/chgrp/{s,g}etfacl, and so on): client-side or server-side? In branch “b_new_cmd”, client_side uid/gid are used for remote_stat/remote_chown/remote_chgrp, but server-side uid/gid for remote_acl, this is somewhat confusing to the user.

2. Invalid username/groupname information.

   Generally speaking, local client shares the same user database with the server, but the remote client does not. "lfs getfacl /mnt/lustre/file" on remote client returns the result of "getfacl /mnt/lustre/file" on local client directly. But the username/groupname in the result belong to the server-side user database. There maybe no such names or indicate different ones on the remote client.

3. Security hole: remote user can scan out all the server-side "username/groupname <=> uid/gid" mapping.

   Remote user can scan the mapping by the following operations:

   ```
   <1> touch /mnt/lustre/file
   <2> chmod 0444 /mnt/lustre/file
   <3> lfs setfacl -m u:UID:w /mnt/lustre/file
   <4> lfs getfacl /mnt/lustre/file
   ```

   The output of <4> contains either (if the "UID" exist on server):

   ```
   user:USERNAME:-w-
   ```

   or (if the "UID" does not exist on server):

```
user:UID:-w-
```

Repeat <2> to <4> with different "UID", the remote user can scan out all the server-side "username <=> uid" mapping. The similar scan for "groupname <=> gid" mapping also can be done.

Since remote user is not trusted, such information should be kept unknown from him. It is a serious security hole.

4. Local client on each MDS.

   MDS uses "upcall" for redirecting the "{s,g}etfacl" command from remote client to local client on MDS. For that we have to mount a local client on each MDS, and make all the possible users login kerberos before remote_acl. It makes the Lustre configure more complex and increases the MDS loading.

5. Limited command format.

   Only one target file can be specified in each “lfs {s,g}etfacl” command. For example, for local user, such command "setfacl -m u:UID:w /mnt/lustre/file1 /mnt/lustre/file2" can work well, but for remote user, it has to be divided into two commands. Other limitations: it can not support “lfs setfacl -M{X}” (such cases can read ACL entries from the stdin, or read from a specified file which maybe exist in the local FS on remote client, but local client on MDS can not access such file), and so on.

On the other hand, the following issues are arguable, not sure whether reasonable or not.

- Should the remote users know “client-side uid <=> server-side uid” mapping or not?

  On a local system, an user can use “id -u username” to detect users uid. If such case can be extended, a distributed system user also can know his uid (not only the client-side uid, but also the server-side one) by some command or means. But for security, maybe we only permit him to know his own server-side uid or the ones belong to the same remote client users, but not all the system users.

  In fact, even if we forbid remote user to know his server-side uid, he also can scan out such mapping by the following operations:

  ```
  <1> touch /mnt/lustre/file
  <2> lfs getfacl /mnt/lustre/file
  ```

  The output of <2> contains the following:

  ```
  # file: file
  # owner: username
  # group: groupname
  ```

The “username” and “groupname” are both server-side, since he can scan out server-side “username/groupname <=> uid/gid” mapping in 1.3, he also knows the LUID/LGID of the “file”, and then knows the mapping is “RUID <=> LUID”.

- “client-side uid/gid <=> server-side uid/gid” mapping is required before setfacl for user/group on remote client.

  If remote user wants to setfacl for user/group on local client, he can use “server-side” uid/gid directly. But if he wants to setfacl for another remote user, he has to know which id the target remote user is mapped to on server. Such requirement seems not reasonable, maybe cause security information divulged. Or another possibility is that: if remote user1 wants to obtain some permission on “/mnt/lustre/file” which is owned by remote user2, he can submit his request to system administrator who knows remote user1 is mapped to “idXXXXX”, then the system administrator can ask remote user2 to setfacl for “idXXXXX”, and the remote user2 does not know remote user1 at all.

2 Requirements

- Support ACL-based permission check for remote user.
- Support setfacl/getfacl for remote user.

In Remote_ACL_HLD_v2, we will design a new remote_acl framework to try to resolve the issues discussed above. On one hand, we should preserve the system security, especially the server’s security (as issue 1.3); on the other hand, we will improve the system usability to make the remote_acl more like local_acl in both “command format” (as issue 1.5) and “command result” (as issue 1.2). But we have not found out a complete solution for all the known issues and without new issues introduced yet. Some of the argumentative issues maybe exist in the new design also. It will be a compromise design anyway.

3 Functional specification

1. Use kernel level {s,g}etxattr to implement remote_acl. (Try to resolve issues 1.4/1.5)

   No need server-side “upcall” for remote_acl anymore, neither the related local client on each MDS.

2. Provide “lfs l{s,g}etfacl” utils to support remote user to {s,g}etfacl for user/group on the same remote client. It uses the same uid/gid police with other remote operations – the client-side uid/gid. (Try to resolve issues 1.1/1.2/1.3)

   For “lfs lsetfacl”, if the uid/gid in argument are not mapped dynamically on server, setfacl fails, just like remote_chgrp does.

For ”lfs lgetfacl”, if the uid/gid in result are not mapped dynamically on server, they will be remapped back to “nobody”, just like remote_stat does.

3. Provide “lfs rsetfacl” utils to support remote user to setfacl for users/groups on other clients (both local clients and remote ones).

   It is not the same as old “lfs setfacl” utils. The old one packs the user command to server and executes it as local user by server-side upcall. But the new one runs the user command just on the remote client with {s,g}etxattr.

   Username/groupname which exist on other clients maybe not exist on the remote client, or different clients have the same username/groupname but with different uid/gid, so in “lfs rsetfacl” case, only server-side uid/gid can be used (it is remote user’s duty to guarantee that, otherwise the result is undefined, but how to? something similar to old “lfs setfacl”: either the remote user knows the uid/gid or the system administrator asks him to do that).

   Such utils can be used to setfacl for other remote user whose uid/gid are not mapped dynamically on server or for local user.

4. Provide “lfs rgetfacl” utils to support remote user to obtain the ACL entries which contain non-mapped user/group.

   It is not the same as old “lfs getfacl” utils. The old one packs the user command to server and executes it as local user by server-side upcall, and returns all the ACL entries with server-side uid/gid. But the new one runs the user command just on the remote client with getxattr, and returns the ACL entries with “nobody” if uid/gid in such entries have been mapped dynamically on server (for “lfs lgetfacl” case, client-side uid/gid are used); otherwise returns the ACL entries non-mapped dynamically on server with the server-side uid/gid directly (for “lfs lgetfacl” case, “nobody” are used). For security, only the file owner can use “lfs lgetfacl”.

   It is a complementary utils for “lfs lgetfacl”, and very important and useful to check and maintain the ACL entries for the file owner.

5. Server can config which remote user has the permission to use “lfs r{s,g}etfacl”.

   Remote user can use “lfs r{s,g}etfacl” to scan the “client-side uid/gid <=> server-side uid/gid” mapping for his remote client (if we do not want him to know such information), so the system administrator should control such permission in the server configure file (/etc/lustre/setxid.conf) with flags “rmtacl/normtacl” just like “setuid/nosetuid” flags do.

4 Use cases

Remote user1 (501/501) on remote client1 is mapped to local user1 (1001/1001) on server.

Remote user2 (502/502) on remote client1 is mapped to local user2 (1002/1002) on server.

Remote user3 (503/503) on remote client2 is mapped to local user3 (1003/1003) on server.

Remote user4 (504/504) on remote client2 is mapped to local user4 (1004/1004) on server.

Local client shares the same user database with server, and local user operations are performed on local client.

1. Local user setfacl for remote user.
   (a) local user1: echo “foo” > /mnt/lustre/file1 && chmod 0222 /mnt/lustre/file1
   (b) remote user2: “cat /mnt/lustre/file1” fails.
   (c) local user1: setfacl -m u:user2:r /mnt/lustre/file1
   (d) remote user2: “cat /mnt/lustre/file1” succeeds.
   (e) local user1: setfacl -x u:user2 /mnt/lustre/file1
   (f) remote user2: “cat /mnt/lustre/file1” fails.
2. Remote user setfacl for user on the same remote client.
   (a) remote user1: echo “foo” > /mnt/lustre/file2 && chmod 0222 /mnt/lustre/file2
   (b) remote user2: “cat /mnt/lustre/file2” fails.
   (c) remote user1: lfs lsetfacl -m u:user2:r /mnt/lustre/file2
   (d) remote user2: “cat /mnt/lustre/file2” succeeds.
   (e) remote user1: lfs lsetfacl -x u:user2 /mnt/lustre/file2
   (f) remote user2: “cat /mnt/lustre/file2” fails.
3. Remote user setfacl for user on other clients.
   (a) remote user1: echo “foo” > /mnt/lustre/file3 && chmod 0222 /mnt/lustre/file3
   (b) local user3: “cat /mnt/lustre/file3” fails.
   (c) remote user1: lfs rsetfacl -m u:1003:r /mnt/lustre/file3
   (d) local user3: “cat /mnt/lustre/file3” succeeds.
   (e) remote user1: lfs rsetfacl -x u:1003 /mnt/lustre/file3
   (f) local user3: “cat /mnt/lustre/file3” fails.
   (g) remote user4: “cat /mnt/lustre/file3” fails.
   (h) remote user1: lfs rsetfacl -m u:1004:r /mnt/lustre/file3
   (i) remote user4: “cat /mnt/lustre/file3” succeeds.
   (j) remote user1: lfs rsetfacl -x u:1004 /mnt/lustre/file3

(k) remote user4: “cat /mnt/lustre/file3” fails.

4. Remote user setfacl for user on the same remote client who is not mapped dynamically on server.

   (a) remote user2 logout kerberos.
   (b) remote user1: echo “foo” > /mnt/lustre/file4 && chmod 0222 /mnt/lustre/file4
   (c) remote user1: “lfs lsetfacl -m u:user2:r /mnt/lustre/file4” fails.

5. Remote user “lfs lgetfacl”.

   (a) remote user1: echo “foo” > /mnt/lustre/file5 && chmod 0222 /mnt/lustre/file5
   (b) remote user2 login kerberos. && df
   (c) remote user1: lfs lsetfacl -m u:user2:r /mnt/lustre/file5
   (d) remote user1: lfs lgetfacl /mnt/lustre/file5

```
# file: /mnt/lustre/file5
# owner: user1
# group: user1
user::-w-
user:user2:r--
group::-w-
mask::-w-
other::-w-
```

   (e) remote user2 logout kerberos.
   (f) remote user1: lfs lgetfacl /mnt/lustre/file5

```
# file: /mnt/lustre/file5
# owner: user1
# group: user1
user::-w-
user:nobody:r--
group::-w-
mask::-w-
other::-w-
```

   (g) remote user1: lfs rsetfacl -m u:1003:r /mnt/lustre/file5
   (h) remote user1: lfs lgetfacl /mnt/lustre/file5

```
# file: /mnt/lustre/file5
# owner: user1
# group: user1
user::-w-
user:nobody:r--
user:nobody:r--
group::-w-
mask::-w-
other::-w-
```

6. Remote user “lfs rgetfacl”.

   (a) remote user1: echo “foo” > /mnt/lustre/file6 && chmod 0222 /mnt/lustre/file6

   (b) remote user2 login kerberos. && df

   (c) remote user1: lfs lsetfacl -m u:user2:r /mnt/lustre/file6

   (d) remote user1: lfs rgetfacl /mnt/lustre/file6

```
# file: /mnt/lustre/file6
# owner: user1
# group: user1
user::-w-
user:nobody:r--
group::-w-
mask::-w-
other::-w-
```

   (e) remote user2 logout kerberos.

   (f) remote user1: lfs rgetfacl /mnt/lustre/file6

```
# file: /mnt/lustre/file6
# owner: user1
# group: user1
user::-w-
user:1002:r--
group::-w-
mask::-w-
other::-w-
```

   (g) remote user1: lfs rsetfacl -m u:1003:r /mnt/lustre/file6

   (h) remote user1: lfs rgetfacl /mnt/lustre/file6

```
# file: /mnt/lustre/file6
# owner: user1
# group: user1
user::-w-
user:1002:r--
user:1003:r--
group::-w-
mask::-w-
other::-w-
```

   (i) remote user2 login kerberos. && df

   (j) remote user1: lfs rgetfacl /mnt/lustre/file6

```
# file: /mnt/lustre/file6
# owner: user1
# group: user1
user::-w-
```

```
user:nobody:r--
user:1003:r--
group::-w-
mask::-w-
other::-w-
```

7. local user use “lfs {l,r}{s,g}etfacl”.
   (a) local user1: echo “foo” /mnt/lustre/file7 && chmod 0222 /mnt/lustre/file7
   (b) local user1: “lfs lsetfacl -m u:1002:r /mnt/lustre/file7” succeeds, just like local_acl.
   (c) local user1: “lfs rsetfacl -m u:1003:r /mnt/lustre/file7” succeeds, just like local_acl.
   (d) local user1: “lfs lgetfacl /mnt/lustre/file7” succeeds, just like local_acl.
   (e) local user1: “lfs rgetfacl /mnt/lustre/file7” succeeds, just like local_acl.
8. Non-owner use “lfs rgetfacl”.
   (a) remote user1: echo “foo” /mnt/lustre/file8 && chmod 0222 /mnt/lustre/file8
   (b) remote user2: “lfs rgetfacl /mnt/lustre/file8” fails.
9. Server config “rmtacl/normtacl” permission.
   (a) system administrator config remote user1 with permission “normtacl”.
   (b) remote user1: echo “foo” /mnt/lustre/file9 && chmod 0222 /mnt/lustre/file9
   (c) remote user1: “lfs rsetfacl -m u:1002:r /mnt/lustre/file9” fails.
   (d) remote user1: “lfs rgetfacl /mnt/lustre/file9” fails.
   (e) system administrator config remote user1 with permission “rmtacl” and flush cache on server.
   (f) remote user1: “lfs rsetfacl -m u:1002:r /mnt/lustre/file9” succeeds.
   (g) remote user1: “lfs rgetfacl /mnt/lustre/file9” succeeds.

The similar cases for group also can be done.

5 Logic specification

5.1 User level utils

- The “lfs {l,r}{s,g}etfacl” utils use the system {s,g}etfacl command to do the real {s,g}etfacl work. Before that, the utils will ioctl lustre mountpoint with some flags to notify llite kernel which acl operations will be done.

- For “lfs rgetfacl” case, the system getfacl command utils converts the server-side uid/gid in the result ACL entries to username/groupname based on the client-side user database before outputting to the user. But it will misguide the user. So the “lfs rgetfacl” utils forks out a child process firstly, then the child process executes system getfacl command, and redirects its output to the parent process, and the parent process converts the username/groupname back to the uid/gid based on the same client-side user database, and then does the real output. The final output of ACL entries only contain server-side uid/gid (except “nobody”).

5.2 Client-side kernel level logic specification

“lfs {l,r}{s,g}etfacl” utils ioctl lustre mountpoint to notify llite kernel which acl operations will be done. llite kernel records such information as “rmtacl_ctl_entry” which are linked into “rmtacl_ctl_table”.

For remote client, setxattr and getxattr search the “rmtacl_ctl_table” with “pid” to check whether it is “lfs {l,r}{s,g}etfacl” cases or not, and pack some new flags to notify MDS these cases. Beyond that, the “lfs lsetfacl” need more additional process.

- For “lfs lsetfacl” case.

  It is some complex for such case. Before setxattr, “lfs lsetfacl” fetches the old ACL entries back which contain client-side uid/gid or “nobody” by getxattr. When setxattr sends the new ACL entries to MDS, the MDS will map such client-side uid/gid to server-side ones. There maybe some ACL entries with “nobody” which can not be processed by client, so the MDS has to process such ACL entries based on the old ACL entries on server. Now the issue occurring:

  If user wants to delete an ACL entry, such ACL entry are fetched back (if it exists and the uid/gid in which is mapped dynamically on server) first, and then such entry do not exist in the ACL entries list sent by the succedent setxattr. If the MDS found that some old ACL entries on server with mapped uid/gid do not exist in the new ACL entries list sent by setxattr from remote client, how does the MDS process such entries: are they deleted ACL entries or the related “client-side uid/gid <=> server-side uid/gid” mapping were established just between getxattr and setxattr? How to distinguish the two cases?

  In fact, it is difficult for MDS to distinguish the two cases without other hints. For resolving such issus, we introduce the following mechanism:

  The getxattr fetches back ACL entries from MDS for “lfs lsetfacl”, and backups it with “ext_acl_xattr_entry” format in “eacl_table“ on llite. The succedent setxattr compares the new ACL entries list from user-space with the extend ACL entries list backuped in the “eacl_table”. Then setxattr knows which ACL entries are deleted (added or modified) and records such information in extend ACL entries. Finally, the fixed “ext_acl_xattr_entry” format ACL entries list is sent to MDS.

- Garbage collection.

  When "lfs {l,r}{s,g}etfacl" utils ioctl lustre mountpoint, it opens the lustre mountpoint file handle, and such file handle is held until "lfs {l,r}{s,g}etfacl" utils thread exit. When the thread exit, all the unclosed file handle related with this thread are closed by kernel, and then the llite kernel processes the left entries in the "rmtacl_ctl_table" and "eacl_table" related with this thread. So it is unnecessary to do any additional garbage collection in case of "lfs {l,r}{s,g}etfacl" utils exit whether abnormally or not.

5.3 Server-side kernel level logic specification

1. setxattr: map client-side uid/gid in ACL entries to server-side ones and process "nobody".
   - For "lfs lsetfacl" case.
     - Map uid/gid in the "ext_acl_xattr_entry" format ACL entries (except "nobody" ones) to server-side uid/gid. If some related mapping do not exist, "lfs lsetfacl" failed.
     - Merge the old ACL entries on server and the fixed "ext_acl_xattr_entry" format ACL entries into new normal "posix_acl_xattr_entry" format ACL entries.
     - Invoke low layer function to setxattr.
   - For "lfs rsetfacl" case.

     Very simple for such case, the uid/gid in ACL entries are server-side already since sent from the remote client. The only work is checking each entry to make sure no "nobody" ones in the ACL entries list.
2. getxattr: remap the server-side uid/gid back to client-side ones.
   - For "lfs l{s,g}etfacl" case.

     Remap the server-side uid/gid in ACL entries back to client-side ones if mapped; for the non-mapped uid/gid, set as "nobody".
   - For "lfs rsetfacl" case.

     No any other additional process needed.
   - For "lfs rgetfacl" case.

     Set the mapped uid/gid in ACL entries as "nobody".

6 State management

6.1 Scalability & performance

In branch "b_new_cmd", "client-side uid/gid <=> server-side uid/gid" mapping process has been done in MDT layer, and server-side ACL entries related process has been

done in MDD layer. But for the new remote_acl design, in which layer to process the ACL entries (including uid/gid mapping, ACL entries merging, ACL-based permission checking, and so on) – MDT or MDD? It seems not good in any layer for either breaking the original MD stack layout or maintaining some similar processes in different MD stack layers. In the new design, we will move all the ACL entries process code and idmap related code to obdclass module as public functions, and can be used in both MDT and MDD (in fact, llite will use some ACL entries process for "lfs lsetfacl" also), thus we keep the original MD stack layout and the code unity.

For remote_acl, many additional processes are introduced, and make it bad performance than local_acl, but {s,g}etfacl are not frequently-used operations, such performance drop is acceptable.

6.2 Locking changes

In Remote_ACL_HLD_v1, remote_setfacl need not hold rpc lock. Because it just sends command to MDS, and the local client on MDS runs such command, then it will invoke "`mdc_get_rpc_lock`()" to hold rpc lock in "`mdc_xattr_common`()". In the new design, remote_setfacl ("lfs {l,r}setfacl") command is run on the remote client directly, so need hold rpc lock in "`mdc_xattr_common`()" first, and then release it after using.

6.3 Other changes

- Rename server configure file "/etc/lustre/setxid.conf" to "/etc/lustre/perm.conf". Add permission flags "rmtacl/normtacl", they are similar to "setuid/nosetuid", for enable/disable permission of "lfs r{s,g}etfacl", and processed by "l_getidentity".
- Drop the old remote_acl related code, mainly for MDS rmtacl upcall process.
- The ACL entries on remote client are incompletable, and can not be used for doing ACL-based permission check on remote client, so keep the "remote_perm" related code.
- "client-side uid/gid <=> server-side uid/gid" mapping related fix.

  In branch "b_new_cmd", we assume that: "client-side uid <=> server-side uid" is one-to-one mapping, and the same for "client-side gid <=> server-side gid" mapping.

  In fact, it seems more convenient for the system administrator that allowing the "N:1" mapping of "client-side uid <=> server-side uid". E.g.: all the remote users from remote client1 will be mapped as the same local user1.

  For gid mapping, "N:1" mapping is not enough even sometimes, the "N:M" cases exist. E.g.: luid1 and luid2 belong to the same group (lgid1) on server-side, and ruid1 and ruid3 belong to the same group (rgid1) on client-side.

```
ruid1/rgid1 <=> luid1/lgid1
ruid2/rgid2 <=> luid2/lgid1
ruid3/rgid1 <=> luid3/lgid3
```

But “N:M” mapping rule is too complex to implement. So here we assume that: “client-side uid/gid <=> server-side uid/gid” mapping is “N:1” case. It is improvement for “remote_uid” against last cycle design, and ACL entries mapping related, will be done in the same task this cycle.

7 Alternatives

- Maybe we can use “lfs setfacl ... nid” to avoid using the server-side uid/gid in “lfs rsetfacl”. The idea is that: the remote user can use the uid/gid with the nid to setfacl for the user/group on the specified client (both local and remote cases), the “uid/gid” belong to the specified client-side. But such idea cause other issues: the remote user can not setfacl for the non-mapped remote user/group, it is very inconvenient for canceling some granted permission; on the other hand, the local client nid and server nid are always similar, so the baleful remote user can guess the local client nid, and scan out all the system “client-side uid <=> server-side uid” mapping. So seems not better than the current new design.

8 Focus for inspections

1. The remote users can scan out “client-side uid/gid <=> server-side uid/gid” mapping on its remote client by “lfs r{s,g}etfacl” utils, is it acceptable? which exists in both the Remote_ACL_HLD_v1 and the new design.
2. Whether the issue of “client-side uid/gid <=> server-side uid/gid” mapping is required before setfacl for user/group on remote client” should be resolved? which exists in both the Remote_ACL_HLD_v1 and the new design.
3. Only the server-side uid/gid can be used for “lfs rsetfacl”, and the remote user should guarantee that, is it acceptable?
4. The new design does not provide an utils to list all the ACL entries only with either client-side uid/gid or server-side uid/gid for remote_getfacl (means without “nobody”), and cause some local {s,g}etfacl command formats can not be supported fully, e.g.:

```
Copying the access ACL into the Default ACL
local_acl:
    getfacl --access dir | setfacl -d -M- dir
remote_acl:
    lfs lgetfacl --access dir | lfs lsetfacl -d -M- dir && lfs rgetfacl --access di
```

5. Remote user setfacl for user/group on the same remote client can use “lfs lsetfacl”, but such user/group must be mapped dynamically on server, otherwise maybe get unexpected result. e.g.:

   remote user1 and remote user2 on the same remote client.

   (a) remote user1: echo “foo” > /mnt/lustre/file && chmod 0222 /mnt/lustre/file
   (b) remote user2 logout kerberos.
   (c) remote user1: “lfs lsetfacl -m u:user2:r /mnt/lustre/file” fails.
   (d) remote user2 login kerberos. && df
   (e) remote user1: “lfs lsetfacl -m u:user2:r /mnt/lustre/file” succeeds.
   (f) remote user2 logout kerberos.
   (g) remote user1: “lfs lsetfacl -x u:user2 /mnt/lustre/file” succeeds, but such ACL entry has not been removed really.
   (h) remote user1: lfs lgetfacl /mnt/lustre/file

   ```
   # file: /mnt/lustre/file
   # owner: user1
   # group: user1
   user::-w-
   user:nobody:r--
   group::-w-
   mask::-w-
   other::-w-
   ```

   (a) remote user2 login kerberos. && df
   (b) remote user1: “lfs lsetfacl -x u:user2 /mnt/lustre/file” succeeds really.
   (c) remote user1: lfs lgetfacl /mnt/lustre/file

   ```
   # file: /mnt/lustre/file
   # owner: user1
   # group: user1
   user::-w-
   group::-w-
   mask::-w-
   other::-w-
   ```

6. The most important recovery related work is that: how to reestablish the “client-side uid/gid <=> server-side uid/gid” mapping which existed before MDS crashed in recovery. Such issue is not new addressed by the new remote_acl design, it is related with both remote_acl and remote_stat/remote_chgrp (without such reestablishment, remote_stat/remote_getfacl maybe get unexpected user/group: “nobody”, and remote_chgrp/remote_setfacl maybe failed for unexpected error: “EPERM, unmapped user/group”). Here, we arise such issue just for making sure the related recovery can work as expected.