High Level Design for CIFS LOV EA

Matt Wu

2005/09/05

1 Introduction

The "LOV EA support" is a sub-task of windows CIFS parallel i/o filter project. The parallel filter client access lustre via Samba servers over CIFS protocol. It accesses the meta-data from the MDS Samba server, and does the I/O requests directly to the OST Samba servers to achieve parallel I/O and high data throughput. So in the window client side, we need get the data distribution information for the inode and also the Samba share points names for the OSTs where the inode is stripped over. Then parse the stripe distribution information and redirect the I/Oes to differnt OST servers. So we need create special EAs to store our own information inside on lustre side. And also implement the support routines to help the client to query these EAs when needed.

2 Requirements

1. export the lov_mds_md / lov_stripe_md information as an EA for every inode
2. build the mapping relationship bewteen the OST object name and the OST index number in the lov_mds_md / lov_stripe_md structure. The relationship is unique per cluster. We just need build one EA for the root inode.

3 Functional Specification

3.1 "lov_dist" EA for every inode

Currently the data distribution information is stored in the MDS in the format of lov_mds_md structure. The EA name is "trusted.lov". It contains the stripe size / count and the OST object device index number. see the followings structures:

```
#define lov_ost_data lov_ost_data_v1
struct lov_ost_data_v1 {  /* per-stripe data structure (little-endian)*/
```

```
    __u64 l_object_id;    /* OST object ID */
    __u64 l_object_gr;    /* OST object group (creating MDS number) */
    __u32 l_ost_gen;      /* generation of this l_ost_idx */
    __u32 l_ost_idx;      /* OST index in LOV (lov_tgt_desc->tgts) */
};
#define lov_mds_md lov_mds_md_v1
struct lov_mds_md_v1 {    /* LOV EA mds/wire data (little-endian) */
    __u32 lmm_magic;      /* magic number = LOV_MAGIC_V1 */
    __u32 lmm_pattern;    /* LOV_PATTERN_RAID0, LOV_PATTERN_RAID1 */
    __u64 lmm_object_id;  /* LOV object ID */
    __u64 lmm_object_gr;  /* LOV object group */
    __u32 lmm_stripe_size;/* size of stripe in bytes */
    __u32 lmm_stripe_count;/* num stripes in use for this object */
    struct lov_ost_data_v1 lmm_objects[0]; /* per-stripe data */
};
```

But llite could only export the user EAs. That means we could not access the "lov" EA via Samba, becuase it's typed of "trusted" EA. We could get are two ways to achieve our target:

- 1), ldiskfs: add the "lov" EA to user EA list

functions related:

```
a) ldiskfs_xattr_ibody_get
b) ldiskfs_xattr_block_get
```

implementation:

```
when handling LDISKFS_XATTR_INDEX_USER (user EA list), we need treat "trusted.lov" as u
```

- 2), llite: create a fake / read-only EA on the fly

for every inode, the unpacked stripe information are stored in the inode structure (ll_i2info(inode)->lli_smd, in lov_stripe_md format).

```
struct lov_stripe_md { /* Public members. */
    __u64 lsm_object_id; /* lov object id */
    __u64 lsm_object_gr; /* lov object id */
    __u64 lsm_maxbytes; /* maximum possible file size */
    ulong_ptr lsm_xfersize; /* optimal transfer size */
    /* LOV-private members start here -- only for use in lov/. */
    __u32 lsm_magic;
    __u32 lsm_stripe_size; /* size of the stripe */
    __u32 lsm_pattern; /* striping pattern (RAID0, RAID1) */
    unsigned lsm_stripe_count; /* number of objs being striped over */
    struct lov_oinfo lsm_oinfo[0];
};
```

```
struct lov_oinfo { /* per-stripe data structure */
    __u64 loi_id; /* object ID on the target OST */
    __u64 loi_gr; /* object group on the target OST */
    int loi_ost_idx; /* OST stripe index in lov_tgt_desc->tgts */
    int loi_ost_gen; /* generation of this loi_ost_idx */
    ......
};
```

These structures contain plenty of information. We can travel back from the ost index to the ost UUID (lov_tgt_desc.obd_uuid). The UUID is unique among all the cluster.

During rutime, we could get the lov object device (struct lov_obd *lov) from the super bock (ll_sb_info). The lov- >tgts contains the array of the ost devices.

```
struct lov_tgt_desc {
    struct obd_uuid uuid;
    __u32 ltd_gen;
    struct obd_export *ltd_exp;
    int active; /* is this target up for requests */
};
struct lov_obd {
    spinlock_t lov_lock;
    struct lov_desc desc;
    int bufsize;
    int refcount;
    int lo_catalog_loaded:1, async:1;
    struct semaphore lov_llog_sem;
    ulong_ptr lov_connect_flags;
    struct lov_tgt_desc *tgts;
};
```

the member lov_tgt_desc.obd_export represents the OSC client object device. Every server has it's own name and UUID, though they may repsents the same OST. But from the lov_tgt_desc.obd_uuid we can get to the unique ost device.

Summary:

method 1 is better to go down to next step: directly sharing the OST volume via Samba. That time we could have direct access to all trusted EAs. But with method 2, with the OST naming information being stored in the inode "lov" EA, we can pack two EAs into a single one to reduce one extra EA querying request .

Generally these two methods are not very complex and difficult. We could change to another in case of performance issues or other problems.

3.2 "ost_map" EA for root inode on MDS server

The content of this EA should be created dynamically becuase there might be possibe OST migration / deletion / newly addition. So this EA is to be designed as read-only and created on the runtime in llite EA handler routnines (file.c ll_getxattr / ll_setxattr).

The Samba sharing names are to be defined in this rule (1 MDS + N OST):

Samba Server	Mount Point (Ex)	Sharing Name
MDS server	/mnt/lustre	mds\lustre
OST1 server	/mnt/lustre	ost1\lustre
OST2 server	/mnt/lustre	ost2\lustre
...	...	...

- 1), structres / format of the EA value

```
struct ost_info {    /* information per ost*/
    __u32 ost_index; /* ost index number */
    __u16 uuid_len;  /* uuid string length, including \0 in the tail */
    __u16 name_len;  /* length of samba sharing name, with \0 in tail */
    const char * uuid[]; /* ost uuid string */
    const char * name[]; /* samba sharing name. EX: ost1, ost2 ... */
};
struct ost_map {       /* the format of the EA content */
    __u32 osts_count; /* number of ost devices */
    struct ost_info osts_info[]; /* flexible size array */
};
```

- 2), ll_getxattr / ll_getxattr_internal

It will gather the ost information from the super_block (ll_sb_info) and the obdclass, then fill the ost_map structure and return to Samba. All the information are already set up in the ll_sb_info, lov_obd and the obdclass obd_dev array. On MDS server, the ost devices names are managedt by obdclass. We can get to them via their UUID stored in lov->tgts. And use the ost device name as the samba sharing name. A script could be used to configure all the samba server settings (setup the hostname and Samba settings) instead of manually.

- 3), ll_setxattr / ll_setxattr_internal

Here it just forbids any writing requests to this EA.

4 Use cases

For a CIFS parallel I/O system that is already set up, all the samba servers are ready sharing lustre as the corresponding mds / ost names. And windows

should be already connecting to the mds Samba server.

When client starts an I/O request, the whole system behaves like this:

1. windows filter: request EA “ost_map” of the root inode from MDS Samba
2. MDS Samba server: ll_getxattr will dynamically create the "ost_map" EA and return to Samba.
3. windows filter: request EA “lov_dist” of the root inode from MDS Samba
4. MDS Samba server: ll_getxattr will finally get to ldiskfs. Then ldiskfs will return the content of "trusted.lov" as the result to Samba.
5. Windows filter: parse the lov strip information and issue requests to the corresponding ost Samba servers.

5 Logic Specifications

5.1 "lov_dist" EA for every inode

when user is querying the "lov_dist" EA, we just let ldiskfs_xattr_ibody_get or ldiskfs_xattr_block_get find the EA of "trusted.lov" and return the content as there exists such a "lov_dist" EA.

5.2 "ost_map" EA for root inode on MDS server

- 1), ll_getxattr / ll_getxattr_internal

```
static int ll_getxattr_internal(struct inode *inode, const char *name, int namelen, voi
    ......
    /* to query the value of "ost_map" */
    if (name is "ost_map") {
        /* yes. it's just our EA querying request */
        1) find the ll_sb_info and the lov object (lov->tgts)
        2) get the ost names from obdclass (obdclass: obd_dev
           or ioctl request)
        3) fill the structure of ost_map and ost_info array
           with the gatherred information
        4) now it's ready return to Samba
    } else {
       /* normal handling operations */
       ...
    }
    RETURN (rc);
}
```

- 2), ll_setxattr / ll_setxattr_internal

```
static int ll_setxattr_internal(struct inode *inode, const char *name,  void *value, si
    ......
    /* to set the value of "ost_map" */
    if (name is "ost_map") {
        /* we forbid this !!! */
        1) just return -ENOENT or -EACCES
    } else {
       /* normal handling operations */
       ...
    }
    RETURN (rc);
}
```

6 State Management

- Lock protection:

We are only reading, no any changes are made. But the core structure access should be under protection to avoid possible races. Existing routines re-usage should be considerred prior to direct access.

- OST Changes:

This part may cause possible race. I 'm not very clear of this part.

7 Focus of Inspection

1. The design is reasonable ? Could be better ?
2. How could OST changes (migration / deletion / addition) affect ?
3. Could there be possible DLM deadlocks ?

8 Inspection Summary

Summary for the inspection results:

8.1 "lov_dist" EA

to be handled in llite: ll_getxattr / ll_getxattr_internal. So the server codes won't be touched.

8.2 "ost_map" EA

- a) fixed-size ost_info to avoid unexpected complexity.
- b) adding ip address of the ost server.

since we have the ip address already, the Samba netbios share name won't care much for us. We still keep it here for verificaiton purpose.

- c) using the uuid as netbios share name

```
struct ost_info { /* information per ost*/
    __u16  magic;  /* Magic and flags */
    __u16  flags;
    __u32  ost_index; /* ost index number */
    const  char uuid[40]; /* ost uuid string, trailing with NUL */
    __u32 ipaddress; /* ip address of the ost server */
};
struct ost_map { /* the format of the EA content */
    __u16  magic; /* magic and flags */
    __u32 osts_count; /* number of ost devices */
    struct ost_info osts_info[]; /* array of ost information */
};
```

8.3 Performance Improvements

To reduce frequent querying of ost_map EA, we'd better use a file to store the ost_map instead of using an EA of root inode.

- a) using a file (LOV_TGTS) under root or a hidden virtual directory

to store the OST server informations (ip addresses, UUID ...)

- b) using directory change notify to act as OST failover callback

8.4 Issues

a) Oleg: patch for lock on root inode