Bug #14115: EAs in dnode

Girish Shilamkar

2007-12-29

1 Introduction

ZFS has feature of file fork, which is additional data associated with a file system object (like EAs - extended attributes).

There are many consumers who would like a mechanism for storing EAs much nearer to the metadata of file. Mac OS/X requires finder info in EAs, pNFS metadata server requires EAs for storing layout information and Lustre requires LOV extended attribute data, to name some. So EAs in dnode is solution which caters to this requirement

The document discusses how libnvpair can be used to fulfill the need for storing EAs in dnode.

2 Architecture

2.1 Requirements

1. Fast access to Lustre EA values.
2. The overhead to to store the EAs should not be more than approximately 128 bytes + 48 bytes/EA.
3. EAs should be accessible via ZPL as this will allow the DMU to be mounted as ZFS for debugging purpose.

2.2 Architectural description

2.2.1 Current Scenario:

|<————————————————”bonus” buffer(320) ————————————————>|

|<————- znode (264)—————————>|<————————————- data(56)————————>|

In the dnode, the last few bytes after storing znode are vacant. Antivirus scanstamp might be stored in this areas, sometimes in case of the object being symlink the target will be stored in this space if the space available is enough. This space is increased by increasing the size of the dnode (https://bugzilla.lustre.org/show_bug.cgi?id=14113). The increase in the size will enable storing some more information in it i.e extended attributes (EAs).

2.2.2 libnvpair for in-dnode EAs:

libnvpair is a library for manipulating <name, value> pairs. libnvpair is used for packing nv pairs to memory buffers and saving it onto the disk .

The typical usage of nvpairs is as follows:

1. Get the data i.e packed nvlist into memory.
2. Unpack this list.
3. Lookup the desired name in this list.
4. Add new nvpair.
5. Pack this nvlist into memory.
6. Write this memory to persistent storage.

A typical reading of EAs would involve reading the packed nvpairs from the dnode_phys_t and then decoding into nvlist_t. The consumer can then read the EAs from this nvlist. An abstraction layer i.e. set of APIs will be provided which will encapsulate all the nvlist operations so that the implementation of this feature is not consumer specific.

dmu_[set/get]_xattr() are the APIs which are used for handling the extended attributes. These APIs will use the libnvpair methods to store the extended attributes in the dnode in XDR format.

The dnode will require a new flag to indicate presence of nvlist in dnode which will be stored in znode_phys_t.

```
#define ZFS_INDNODE_XATTR      0x100           /* Dnode has EAs */
```

One of the pad fields will be modified to be used for store the size of packed nvlist.

```
struct znode_phys {
            ...
            uint64_t zp_nvsize;             /* 152 - NVlist size for EAs in dnode */
            uint64_t zp_pad[2];             /* 160 - future */
            ...
}
```

As the lookups to this list will be frequent, when libnvpair is used for EAs, the pointer to unpacked nvlist is stored in dnode_t (dn_xattr), to allow EAs from the in-memory unpacked nvlist.

Changes to struct dnode_t :

```
struct dnode_t {
                krwlock_t dn_struct_rwlock;
                ...
                ...
                /* nvlist for unpacked nvlist. */
                nvlist_t *dn_xattr;
               }
```

When for the first time an EA is to be accessed from dnode the packed nvlist is read from the disk and the unpacked nvlist pointer is added to struct dnode_t. The packing of nvlist can be done lazily in dnode_sync() but as there can be multiple transaction groups (open, quiescing and syncing) there can be different groups of EAs for same dnode. In order to avoid the problem of multiple transactions been processed the nvlist is packed and stored to bonus buffer whenever the nvlist is updated in dmu_setxattr.

2.2.3 Handling existing data in bonus buffer:

Symlinks:

The existing space in bonus buffers is sometimes used for storing symlinks if the symlink is small enough to fit into it. If the EAs are stored after following the symlink then it is assured that EAs will be stored in dnode which doesn't contain symlink.

Antivirus Scanstamp:

The antivirus scanstamp can be handled by DMU code. The DMU code if finds a scanstamp on existing dnode it will add it to the nvlist thus becoming the part of EAs. All the new additions of the antivirus scanstamp will then be completely handled by dmu_[set/get]_xattr functions. This will necessarily imply moving antivirus scanstamp from realm of ZPL to DMU.

2.2.4 Overflow of EAs:

In case the bonus buffer space is not enough for EAs then zp_xattr which stores the object id of ZAP object (containing mapping between xattr name and xattr objid) can be used. This mechanism can be made more efficient by storing all the xattrs in single object instead of xattr per object thus reducing the overhead to some extent.

2.2.5 Lustre and in-dnode EAs:

Lustre will interface with DMU through fsfilt. udmu_object_[get/set]_xattr() functions will be implemented to interfacing Lustre with uDMU. This set of uDMU functions will call the underlying dmu_[get/set]_xattr()

2.2.6 ZPL Compatibility:

The current operations on vnodes (struct vnodeops) doesn't contain an API which can be easily modified for ZPL compatibility for in-dnode EAs. A subsequent entry for new vnode functions for extended attributes needs to be added to file vnode operations template.

3 External Functional specifications

3.1 Prototypes

3.1.1 Common libnvpair APIs:

Function to allocate a nvlist, this nvlist stores the EAs in memory.

```
nvlist_alloc(nvlist_t **nvlp, uint_t nvflag, int kmflag)
```

Function to lookup the nvpair :

```
nvlist_lookup_common(nvlist_t nvl, const char *name, data_type_t type, unit_t *nelem, v
```

Unpack the buf read from the disk to nvlist:

```
int nvlist_unpack(char *buf, size_t buflen, nvlist_t **nvlp, int kmflag)
```

Pack nvlist to memory

```
int nvlist_pack(nvlist_t *nvl, char **bufp, size_t *buflen, int encoding, int kmflag)
```

3.1.2 Lustre:

New fsfilt Functions:

```
int fsfilt_udmu_set_md(struct inode *inode, void *handle, void *lmm, int lmm_size, cons
int fsfilt_udmu_get_md(struct inode *inode, void *lmm, int lmm_size, const char *name)
```

New uDMU Functions:

```
int udmu_object_set_xattr(udmu_objset_t *uos, dmu_buf_t *db, void *buf, int buf_size, c
int udmu_object_get_xattr(udmu_objset_t *uos, dmu_buf_t *db, void *buf, int buf_size, c
```

Functions to be added to ZFS/DMU code:

```
int zfs_set_xattr(objset_t *os, dnode_t * dn, void *buf, int buf_size, const char *name
int zfs_get_xattr(objset_t *os, dnode_t * dn, void *buf, int buf_size, const char *name
```

3.2 Layering of APIs

3.2.1 Lustre:

```
fsfilt_dmu_set_md() -->
            udmu_object_set_xattr -->
                  dmu_set_xattr()
fsfilt_dmu_get_md() -->
            udmu_object_get_xattr() -->
                  dmu_get_xattr()
```

3.2.2 ZPL:

```
fop_set_xattr() -->
           zfs_set_xattr() -->
               dmu_set_xattr()
fop_get_xattr() -->
           zfs_get_xattr() -->
               dmu_get_xattr
```

4 High Level Logic

These two functions are wrappers for interfacing Lustre fsfilt method with the dmu methods for extended attributes handling.

```
udmu_object_set_attr()
udmu_object_get_attr()
```

Set/Get functions for EAs.

```
dmu_set_xattr() {

      IF dn_xattr exists
            Read the packed list from the buffer.
      ELSE
           Allocate nvlist dn_xattr.
           nvlist_unpack[dn_xattr, ...];
      IF space available in bonus buffer
            nvlist_add_nvpair(dn_xattr, nvpair);
      ELSE
            Return ENOSPC
      Store size of nvlist in znode_phys
      nvlist_pack[dn_xattr, ...];
      Change the bonuslen.
      Write the packed nvlist to dnode;
}
dmu_get_xattr() {
      IF dn_xattr exists
            Read the packed list from the buffer.
      ELSE
           Allocate nvlist dn_xattr.
           nvlist_unpack[dn_xattr, ...];

      nvlist_lookup_common(dn_xattr, name, ...);
      Return.
}
```

5 Use-Case Scenarios

5.1 Normal use cases

1. Lustre reads the LOV extended attribute data from the object.
2. Lustre OSD (MDT/OST) stores LOV EA data to the object.

5.2 Scalability use cases

Fast access to LOV EAs is critical hence implementation of EAs in dnode should scale for reading LOV EAs.

6 State management

6.1 Locking

The race condition between threads trying to update EAs in same dnode is handled by a new lock added to dnode_t.

```
struct dnode {
        ...
        krwlock_t dn_xattr_rwlock;
        ...
}
```

The reader/writer lock will be used for better concurrency than mutex.

6.2 Cache Usage

N/A

6.3 Recovery

N/A

6.4 Disk state changes

The vacant space after bonus buffer and the space increased by bug #14113 Large dnodes will be used for EAs.

|<————————————————”bonus” buffer ————————————————————————>|

|<————- znode ————————————>|<-AV_SCANSTAMPS-><————— EAs ——————->|

7 Test plan

- Performance test for measuring the performance of IO.
- Sanity Test for ensuring the data written and read from dnode is the same.