



kDMU: code walkthrough

Alex Zhuravlev
2009-11-10



kDMU: intention

- support for ZFS/DMU backend
- make lustre core portable:
 - > currently depends on linux VFS/ldiskfs
 - > even HEAD's MDS

ZFS layering

- contains two major components:
 - > ZPL
 - implements POSIX operations
 - depends on Solaris VFS
 - don't export transactions
 - > DMU
 - implements basic operations on objects
 - portable (still needs some support from OS)
 - export transactions
- we use DMU, not ZPL

DMU: transaction API

- Different API from Idiskfs/jbd:
 - > have explicit *declaration* stage
 - > changes are declared in terms of operations, not number of blocks
 - > Can also simplify Idiskfs usage using same mechanism
- How “unlink” looks like:

```
tx = dm_u_tx_create();  
dm_u_tx_hold_free(tx, oid, 0, DMU_OBJECT_END);  
dm_u_tx_hold_zap(tx, pid, 0, name);  
dm_u_tx_assign(tx);  
dm_u_object_free(oid, tx);  
zap_remove(pid, name, tx);  
dm_u_tx_commit(tx);
```

OSD API: transactions

- currently uses credits and 2-phase transactions
- changed in b_hd_kdmu to support DMU-like transactions:
- ->do_trans_create() is added
- each *modify* method got complimentary *declare* method
- like:
 - > ->do_create()
 - > ->do_declare_create()

DMU: indices

- provides user with library to manipulate key=value pairs - ZAP
- good to export via OSD API:
 - > zap_add() to insert key=value
 - > zap_remove() to remove key=value
 - > zap_lookup() to lookup value by key
 - > zap_cursor_init() to initialize iterator
 - > zap_cursor_retrieve() to get current key
 - > zap_cursor_advance() to move to next
- and methods to declare insert/delete

DMU indices: limitation

- DMU support for ASCIIZ keys only
 - > DMU to support binary keys soon
 - > Binary keys need to be well distributed
- there is no way to iterate the keys in other than key-value order
 - > internal hash is used, like in htree
- FLDB currently uses binary keys to support ranges of sequence, though this is being rethought
- so, CMD is disabled in `b_hd_kdmu` yet

OSD API: data

- Existing HEAD OSD API doesn't handle 0-copy IO
- we need it on OST
- actual implementation of caching, details of IO should be part of specific OSD
 - > too many diffs between Idiskfs and DMU

OSD API: data is simple

- get buffers with `->dbo_get_bufs()`
 - > takes object, offset and length
 - > returns struct `niobuf_local`
 - > `niobuf_local` is host-specific
 - 4K pages on x86, bigger bufs on other architectures
- get data from disk with `->dbo_read_prep()`
- put data to filesystem with `->dbo_write_commit()`
- release with `->dbo_put_bufs()`

OFD is new obdfilter

- implements what obdfilter does
- almost:
 - > most of IO handling moved to OSD layer
 - > doesn't care about io requests, pagecache, etc
 - > lacks nice way to provide all the stats
- left with essential features of obdfilter
 - > locking
 - > grants
 - > recovery

OFD: object ids

- OSD API identify object by fids only
- so, OFD has to convert on-wire id (objid) to fid
- for compatibility purposes range of sequences (ID in FID, IDIFs) are reserved for old OST object IDs
- OFD converts objid to FID in that range
- `lu_idif_build()` is supposed to do so

OFD: namespace?

- OST is object storage
- it doesn't need ANY names
- so we can build it on top of OSD
- but it still need some internal storages for things like *last_rcvd*
- well-known FIDs are used to identify such files
 - > Idiskfs on-disk compatibility is done internal to OSD
- MDS uses different approach
 - > to be fixed, I believe

OST module is same

- it still uses OBD API to access OFD
- currently it's not a problem
- but raises interesting questions about future of this approach
- especially with regard to MDS stack
 - > where no OBD API is used
- nice thing about OST separated from OFD by OBD API
 - > we can use OFD w/o RPC overhead
 - > for example, in benchmarks

MDS on DMU

- current HEAD is tricky as many recovery related things still depend on ldiskfs/jbd (old code)
 - > llog, lovobjids file
- we can't use them with DMU
- the biggest implementation issue is nested transactions
- b_hd_kdmu code is changed in ugly way to pass tx around
- less ugly will be proposed in LOV/OSC presentation

llog in b_hd_kdmu

- the simplest approach is taken
 - > to get something working ASAP
 - > ... and speedup development of kDMU
- llog api got 2nd set of methods
 - > ->llop_open_2()
 - > ->llop_declare_create_2()
 - > ->llop_create_2()
 - > ->llop_declare_write_2()
 - > ->llop_write_2()
 - > they accept transaction handle
- old methods still create transaction

Ilog with DMU

- raised very interesting problem
- DMU requires write declaration to specify offset and length
- actual write can NOT be outside of declared window
- by time we are appending new Ilog records, Ilog can get many records making declared window useless

Ilog on DMU: solutions

- use existing DMU API
 - > use file per thread so that offset never change from declaration to execution
 - simple
 - doesn't scale: $\langle \#OSTs * \#threads \rangle$ Ilog files
 - > declare big window to fit this any number of “awaiting” appends
 - should scale, but complicated, reserves lots of ARC cache
- change DMU API to allow writes at any offset
 - > the best one, pending approval from ZFS team

Code organization

- dmufsd/
 - > osd_handler.c
 - implements most of OSD API
 - > udmu.h
 - a wrapper to avoid conflicts between lustre and zfs definitions in osd_handler.c
 - implements extended attributes (ZFS implements them in ZPL)
 - > ofd/
 - > obdclass/llog_osd.c

kDMU: server stack

