What HSM brings us

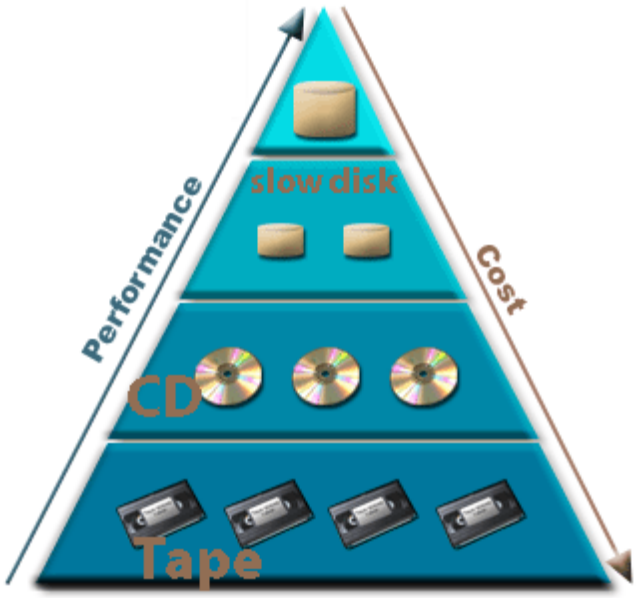
New API bits

And future possibilities

October 27, 2010
Oleg Drokin

A bit of a background

- HSM is being worked on by CEA
  - Under our supervision and guidance
- What is HSM?
  - A way to save money on storage
  - By moving unused data
  - To less expensive media



How do they do it

- Two aspects to that:
  - Changelogs for tracking file activity
    - We are not going to dive into that
  - Ways to change layouts “on the fly”
    - Includes ability to lock layouts
  - Empty “placeholder” files
    - We still need to remember and show filenames in place
  - Locking file content while HSM works on it
    - Good old group lock

Layout lock

- Another bit in inodebits lock

LOOKUP	UPDATE	OPEN	LAYOUT

Layout lock

- Another bit in inodebits lock
  - Layout is stored on MDS, so most logical place
  - Other benefits include:
    - ability to be granted with other bits to decrease lock traffic
    - Less memory usage
    - Less cpu usage for accesses

Usage

- Get lock around lsm accesses
  - ll_get_layout_lock(parent, inode);
  - LSM accessing code
  - ll_get_layout_put(parent, inode);
    - NULL parent is fine if unknown.
- Calling into LOV currently is lsm access
- IO as a whole is lsm access too
  - Of course we should be smarter about it

Usage example

```
static int ll_lov_getstripe(struct inode *inode, unsigned long arg)
{
        struct lov_stripe_md *lsm;
        int rc;
        ENTRY;

       ll_get_layout_lock(NULL, inode);
       lsm = ll_i2info(inode)->lli_smd;
       if (!lsm)
              GOTO(out, rc = -ENODATA);

        rc = obd_iocontrol(LL_IOC_LOV_GETSTRIPE, ll_i2dtexp(inode),
0, lsm,
                           (void *)arg);
out:
        ll_put_layout_lock(NULL, inode);
        RETURN(rc);
 }
```

About layout lock getting

- We fetch the layout lock on first lsm access
  - Introduces a performance regression
  - Originally planned to always fetch it unless blocked
    - Did not pan out due to time
  - We can fight the regression by always getting the lock, at the cost of access time.
    - Easy tradeoff for those who don't run any HSM now.

Opportunistic lock get

- Asking for two sets of lock properties:
  - Must have (LOOKUP or UPDATE)
  - Nice to have (LAYOUT)
- Will get us layout lock every time unless file is being migrated.
- Lock conversion to request and release just individual bits in a lock
  - Some day in the future

Layout lock on a client

- Lock is referenced from struct ll_inode_info
  - Refcount, lock handle and lock mode
- In ll_get_layout:
  - Check struct ll_inode_info for active handle
    - If present, just increase the refcounts
  - Otherwise, execute an intent to get the lock
    - IT_LAYOUT intent type
    - Put received lock into the struct ll_inode_info
- In ll_put_layout
  - Drop refcounts and release the handle if needed

Layout lock on client – cont'd

- During cancel
  - We set LLIF_LAYOUT_INVALID flag in inode flags
    - But nobody looks at the flag yet.
- What else do we need:
  - Lsm refcounting
    - Less lock holding
    - Layouts stay while needed

HSM migration scenario

- Migration thread exclusively locks file layout
  - This does not invalidate the cached data on files
- A Group lock is taken on all stripes of the file
  - Flushes and invalidates client caches
  - Allows copy tool to “join” the lock and actually move data
    - Other IO is still locked out during this
- A migration thread starts to move the data out

HSM migration scenario – cont'd

- Once the data copy is done, stripes are deleted
  - Group lock unlocked too
- Lsm is replaced with empty one
- Inode is populated with size/blocks (som-alike)
  - Would be great to retain a bit of file data too. e.g. so that file(1) works without restore.
- Inode is marked “data offline”
- Layout lock unlocked.

HSM – restore scenario

- When you try to open a file
- A layout is locked
- We create the new layout
- Lock all stripes with a group lock
  - Great opportunity to release layout lock here, but...
- “copy tool” starts to move data in
- Once done, release group lock, then layout lock.

Changes to LSM format

- LSM format v3 is changed somewhat:
  - Without actually making it v4
  - Stripe count is now 16 bit instead of 32 bits
  - Freed 16 bits are now for “lsm version”
- It is only considered valid for LSM to change if id/gid of LSM remains the same and just version increases
  - Prevents files from having multiple layouts due to errors.

Other applications of layout lock

- Allows to restripe files on the fly
  - Long overdue ability to grow file stripes as it grows
  - Handle OST-out of space gracefully
  - Very basic version already implemented
  - Rebalancing of space usage on OSTs

Thank you