WHAT ARE “CAPS”?
(And why won’t my client drop them?)

Greg Farnum
May 20 2019
I’m Greg

- Working on Ceph since 2009 (10 years!), all over
- Currently: RADOS Team and Testing
- Once upon a time: CephFS Tech Lead
- gfarnum@redhat.com
- @gregsfortytwo
WHAT ARE “CAPS”?  

- CephFS and filesystem structures  
- Coherency means lots of locking  
- What caps are and mean  
- Caps are governed by the MDS  
  - How caps change  
- Consequences  
  - Interpreting cap-derived health warnings  
- Questions?
Ceph Cluster Daemons
CephFS Data Flow
<table>
<thead>
<tr>
<th>Path 1</th>
<th>Path 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;ino 0,v2&gt;/home&lt;ino 1,v5&gt;/greg&lt;ino 5,v9&gt;/</code></td>
<td><code>&lt;ino 0,v2&gt;/home&lt;ino 1,v5&gt;/greg&lt;ino 5,v9&gt;/</code></td>
</tr>
<tr>
<td><strong>Mydir[01], total size 7MB</strong></td>
<td><strong>Mydir[10], total size 8MB</strong></td>
</tr>
<tr>
<td>foo -&gt; ino 1342, 4 MB</td>
<td>hi -&gt; ino 1000, 6 MB</td>
</tr>
<tr>
<td>bar -&gt; ino 1001, 1024 KBytes</td>
<td>hello -&gt; ino 6743, 1024 KB</td>
</tr>
<tr>
<td>baz -&gt; ino 1242, 2 MB</td>
<td>whaddup -&gt; ino 9872, 1 MB</td>
</tr>
</tbody>
</table>

CephFS Metadata storage
<table>
<thead>
<tr>
<th>Directory</th>
<th>Inode</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>mydir</td>
<td>ino 1</td>
<td>2074 bytes</td>
</tr>
<tr>
<td>foo</td>
<td>ino 1</td>
<td>6 MB</td>
</tr>
<tr>
<td>bar</td>
<td>ino 1</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>baz</td>
<td>ino 1</td>
<td>2 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CephFS Data Storage
CephFS is Coherent
FILES LOOK THE SAME ON DIFFERENT CLIENTS

- Looking at a file on different clients presents the same metadata and data
- This means we need a single “authority” that owns info and mediates access/changes
  - Hurray, we need distributed locking!
OUR SOLUTION

- MDS is generally authoritative for metadata it holds
- Distributed locks describe allowed state accesses/changes
  - Can other MDSes (read-only) replicate metadata? (hot directories, subtree boundaries, etc)
- Issue “capabilities” (“caps”) to clients
  - Allows caching of data, local timestamp updates, etc
SIMILAR SOLUTIONS

- NFS Delegations
  - Coarser but useful, give clients ownership of file metadata/data
- Samba oplocks
CephX Caps Are Completely Different
CEPHX CAPS ARE COMPLETELY DIFFERENT

- ceph auth get-or-create client.foo mds "allow rw path=/foodir" osd "allow rw pool=foopool" mon "allow r"
- These are security permissions on the Ceph daemons
- Nothing to do with the CephFS distributed file locking caps we talk about in this presentation

- Sorry our name selection was terrible! :(
The Meaning of Caps
CEPHFS CAP BREAKDOWN – COMPONENTS

- **Pin – p**
  - the inode exists

- **Auth – A**
  - authentication metadata: mode, uid, gid

- **Link – L**
  - the inode’s link count (number of dentries linked to an inode)

- **Xattr – X**
  - the inode’s xattrs; their presence and values

- **File – F**
  - file data, file size, mtime et al
CEPHFS CAP BREAKDOWN — PERMISSIONS

- **Shared — s**
  - The client has shared access to this state; one of many

- **Exclusive — x**
  - The client is the only one with access to this state

- **Read — r**
  - The client can read state

- **Write — w**
  - The client can write the state

- **Cache — c**
  - The client can cache the state locally

- **Buffer — b**
  - The client can buffer changes to the state locally
CEPHFS CAP BREAKDOWN – PERMISSIONS

- LazyIO – HPC-inspired extensions for the client to handle consistency itself.

We’re not going to talk about this more today; you won’t see it unless you write custom software!
Not every cap uses every permission

pin: binary; the client can remember an inode’s existence

Auth, Link, Xattr: Shared or eXclusive

[ALX]s – can save the state locally, reference it
  ○ Hurray, we can do permission checking on the client

[ALX]x – Nobody else can look at this state; we “own” it
  ○ We can change the metadata locally and tell the MDS later on!
Not every cap uses every permission.

Fs: can cache and read mtime, size locally
Fx: can write mtime, size locally
Fr: can read the file data (...synchronously from OSD)
Fc: can cache file data for local reads
Fw: can write the file data (synchronously to OSD)
Fb: can buffer data writes; flush in the background
Caps come from the MDS
WHERE DO CAPS COME FROM?

● An MDS is the ultimate authority for metadata
● Each piece of metadata is controlled by a lock ("SimpleLock", "ScatterLock", "FileLock") that decides how that state can be shared
  ○ Replica MDSes get certain permissions clients don’t
  ○ Which capabilities can be issued to which clients
● Clients send in metadata updates and cap requests
● A big pile of heuristics changes the lock state based on these inputs to try and optimize
const struct state_t filelock[LOCK_MAX] = {
    [LOCK_SYNC] = { false, LockLock, ANY, 0, ANY, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD) },
    [LOCK_EXCL_SYNC] = { LockLock, AUTH, 0, 0, 0, 0, ANY, 0, XCL, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_MIX_SYNC] = { false, LockMix, 0, 0, 0, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_LSN_SYNC] = { LockMix, AUTH, 0, 0, 0, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_LOCK] = { LockLock, TRUE, 0, XCL, 0, 0, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_SNAPSHOT] = { LockMix, AUTH, 0, 0, 0, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_MIX] = { LockMix, TRUE, 0, 0, 0, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_MIX2] = { LockMix, TRUE, 0, 0, 0, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_EXCL] = { LockExcl, TRUE, 0, 0, XCL, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_EXCL2] = { LockExcl, TRUE, 0, 0, XCL, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_XSYN] = { LockXsyn, TRUE, 0, 0, 0, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_PRE_SCAN] = { LockScan, FALSE, 0, 0, 0, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),
    [LOCK_SCAN] = { LockLock, FALSE, 0, 0, 0, 0, 0, 0, 0, 0, CEPH_CAP_GS NARED(CEPH_CAP_GCACHE|CEPH_CAP_GREAD),

};
HOW DO CAPS CHANGE?

- The MDS can *grant* or *revoke* caps to individual clients
  - Usually based on other client activity
- On revoke, the client has to stop using the cap, then sends the MDS an MClientCaps message acknowledging the change
  - “Stop using” can be easy or hard: if it’s a cache or shared-read, just drop the data and change state
  - For something like buffered writes, the client has to flush out a bunch of dirty data first — can take some time!
Example: change mode
Example: change mode

Mode change:
- rwxr-xr-x

pLsXsFrw

pAsLsXsFrw

LINUX HOST
KERNEL MODULE
Example: change mode
Example: change mode
Example: change mode

Grant As, mode -rwxr-xr-x
Example: change mode
CONSEQUENCES OF CAP ARCHITECTURE

- The MDS has to remember everything the client has pinned
  - Plus even more, since it’s tracking all the clients!
- So your MDS cache needs to be larger than your client caches
- Since caps are cooperative, clients need to work right or access can be blocked to others
There are some health warnings
Client <name> failing to respond to capability release

- The MDS has sent a revoke message but the client isn’t returning the cap
- Possible causes:
  - network or OSDs are so busy it’s just backed up
  - The client got so many revokes it literally couldn’t flush them all in time
  - Software bug?
  - Malicious client?
  - Client is slow for some other reason
- Defaults to printing this after 60 seconds
- mds_cap_revoke_eviction_timeout (default: 0, off) can be set to forcibly disconnect clients which aren’t responding
Client <name> failing to respond to cache pressure

- The MDS has asked the client to drop some pinned inodes to reduce memory use, but the client isn’t dropping enough
- Possible causes:
  - Maybe your client is actively using all its pinned metadata and just can’t drop more
  - network or OSDs are so busy it’s just backed up
  - Software bug?
  - Malicious client?
  - Client is slow for some other reason
- This warning is a complicated decay counter based on how quickly the client is releasing caps compared to how many it has to release
Programming With Caps
```c
int want, have;
want = CEPH_CAP_FILE_BUFFER;
int r = get_caps(in, CEPH_CAP_FILE_WRITE|CEPH_CAP_AUTH_SHARED, want, &have, endoff);
if (r < 0)
    return r;

// permission checks requiring As go here
put_cap_ref(in, CEPH_CAP_AUTH_SHARED)
if (cct->conf->client_cap & have & CEPH_CAP_FILE_BUFFER) {
    // do buffered write
    if (in->oset.dirty_or_tx)
        get_cap_ref(in, CEPH_CAP_FILE_CACHE | CEPH_CAP_FILE_BUFFER);

    get_cap_ref(in, CEPH_CAP_FILE_BUFFER);

    // async, caching, non-blocking.
    r = objectcacher->file_write(&in->oset, &in->layout,
                                  in->snaprealm->get_snap_context(),
                                  offset, size, bl, ceph::real_clock::now(), 0);

    put_cap_ref(in, CEPH_CAP_FILE_BUFFER);
    if (r < 0)
        goto done;
} else {
    if (f->flags & O_DIRECT)
        _Flush_range(in, offset, size);

    // simple, non-atomic sync write
    C_SaferCond onfinish("Client::write flock");
    unsafe_sync_writer++;
    get_cap_ref(in, CEPH_CAP_FILE_BUFFER); // released by unsafe callback

    filer->write_trunc(in->ino, &in->layout, in->snaprealm->get_snap_context(),
                       offset, size, bl, ceph::real_clock::now(), 0,
                       in->truncate_size, in->truncate_seq, OnFinish);

    client_lock.Unlock();
onfinish.wait();
    client_lock.Lock();
    _sync_write_commit(in);
}

// if we get here, write was successful, update client metadata
```
Interfaces for re-exporting file servers

```c
#define CEPH_DELEGATION_NONE 0
#define CEPH_DELEGATION_RD 1
#define CEPH_DELEGATION_WR 2

typedef void (*ceph_deleg_cb_t)(struct Fh *fh, void *priv);
uint32_t ceph_get_cap_return_timeout(struct ceph_mount_info *cmount);
int ceph_set_deleg_timeout(struct ceph_mount_info *cmount, uint32_t timeout);
int ceph_ll_delegation(struct ceph_mount_info *cmount, Fh *fh,
                        unsigned int cmd, ceph_deleg_cb_t cb, void *priv);
mode_t ceph_umask(struct ceph_mount_info *cmount, mode_t mode);
```
Thanks!

Community Resources:

ceph-users@ceph.com,
ceph-devel@vger.kernel.org

#ceph, #ceph-devel on irc.oftc.net

http://docs.ceph.com/docs/master/cephfs/capabilities/

Greg Farnum

gfarnum@redhat.com

@gregsfortytwo