Llite Dcache Cleanup

# Dcache Code Analysis

1. dentry instantiation

1.1

Llite open codes d\_instantiate and call d\_rehash\_cond in ll\_d\_add. However, the logic of ll\_d\_add is quite similar with VFS d\_add. And it has two callers, one is ll\_lookup\_it\_finish, which takes dcache\_lock, call ll\_d\_add and then release dcache\_lock. It is identical to just calling d\_add therefore can be replaced with d\_add.

[This is already done in latest RCU walk patch. See <http://review.whamcloud.com/#change,1865> for details.]

Another caller is ll\_find\_alias, which is called by ll\_lookup\_it\_finish to find proper local alias to put the dentry. It looks very much like d\_splice\_alias and the only difference is ll\_find\_alias flags the dentry with DCACHE\_LUSTRE\_INVALID which is cleared if client has LOOKUP lock. However we cannot simply replace it with d\_splice\_alias as lustre lookup(IT\_OPEN) doesn’t acquire child LOOKUP lock. So we need to instantiate the dentry even if client doesn’t have LOOKUP lock in order to make lookup success and meanwhile mark dentry as DCACHE\_LUSTRE\_INVALID to hide it from other lookup processes. The same is true for do\_statahead\_enter , the other caller of ll\_find\_alias.

1.2

When ll\_lookup\_it() finds a negative dentry but loses parent UPDATE lock, it flags dentry with DCACHE\_LUSTRE\_INVALID and calls d\_rehash and d\_drop just to initialize d\_hash list. However, it is to work around an old bug in d\_move (bug 11179) that tries to access hash list without checking whether the dentry is hashed. After checking kernel code, the d\_move bug no longer exists neither in oldest supported 2.6.18-latest kernel nor upstream kernel. So the negative dentry DCACHE\_LUSTRE\_INVALID logic can be safely dropped from ll\_lookup\_it(), and llite can just set d\_inode to NULL and return in such case.

Also in ll\_find\_alias(), becase d\_move() on longer has that bug, it is unnecessary to call d\_rehash() before d\_move().

2. dentry invalidation.

2.1

Lustre uses MDS\_INODELOCK\_UPDATE bitlock to protect file size, links, timestamps. When a directory’s MDS\_INODELOCK\_UPDATE is canceled, llite drops all negative child dentries if not in use, otherwise unhashes them and flags with DCACHE\_LUSTRE\_INVALID.

Lustre has to unhash all negative child dentries upon MDS\_INODELOCK\_UPDATE cancelation because otherwise if client regains UPDATE lock, it will think the negative child dentries still valid. However, DCACHE\_LUSTRE\_INVALID is unnecessary in the case because the dentry is already unhashed so anyone holding the dentry count can detect it by checking d\_unhashed() if it really matters (by replacing DCACHE\_LUSTRE\_INVALID checks with d\_unhashed). In fact, the only user may be interested is d\_revalidate, while d\_compare results will be passed to d\_revalidate and d\_delete already checks d\_unhashed by its callers in VFS layer.

OCFS2 deals with this by saving parent inode dir\_lock generation in each child negative dentry’s d\_fsdata field during lookup and popping parent inode dir\_lock generation upon lock acquiring and canceling, so that lock-cancel-relock sequence can be detected in d\_revalidate. The d\_fsdata field is overloaded with two usages, parent inode lock generation for negative dentry and dentry lock data for positive dentry, while d\_inode pointer is used to distinguish between them.

2.2

Also Lustre uses MDS\_INODELOCK\_LOOKUP bitlock to protect dentry, mode, owner and group attributes. When MDS\_INODELOCK\_LOOKUP is canceled, llite frees all unused aliases of the inode. If an alias directory dentry is still in use, llite flags it with DCACHE\_LUSTRE\_INVALID instead of unhashing it to preserve sys\_getcwd functionality (NB: Why can’t sys\_getcwd() fail –ENOENT when the cwd directory is removed? Because it is not always removed. Other client setattr on the directory can cause LOOKUP lock revoked as well.). The flagged dentry will be unhashed and freed when last user calls dput() where ll\_ddelete() that tell dput() to kill the dentry when it is flagged with DCACHE\_LUSTRE\_INVALID.

For similar situation, OCFS2 uses a PR lock to protect each lookuped positive dentry and once it is canceled, OCFS2 knows that it is either unlinked or renamed. So it just calls d\_delete to mark the dentry negative.

Llite cannot call d\_delete in this case because losing LOOKUP lock doesn’t necessarily mean file is unlinked or renamed. In fact, none of the inode bitlocks in Lustre uniquely represents the validity of a dentry. So when these locks are canceled, llite doesn’t know if it is because the file is deleted or just someone wants to update the file metadata. The downside is that llite has to query MDS again for these possible negative dentries flagged with DCACHE\_LUSTRE\_INVALID, instead of finding out true negativity from local dcache like OCFS2 does.

3. DCACHE\_LUSTRE\_INVALID flag consideration

DCACHE\_LUSTRE\_INVALID flag lives in dentry->d\_flag field but it is a Lustre private state flag. So in order to make the code more acceptable to mainline kernel, it is best to save the private flag in a Lustre private data section, namingly dentry->d\_fsdata field.

To do it, we can add int lld\_flags field to ll\_dentry\_data structure and add three functions, ll\_mark\_dentry\_invalid() to set the bit, ll\_clean\_dentry\_invalid() to clear it, and ll\_dentry\_invalid() to test it. Because during lookup, whether dentry is positive or negative, in ll\_lookup\_it\_finish ll\_dentry\_data is attached to dentry->d\_fsdata before hashing, therefore d\_fsdata exists for all dentries in hash and the three functions can be called in non-blocking context (e.g., under spinlock) for dentries after that.

4. other cleanups on dcache code

4.1

line 454 if ((it->it\_op == IT\_OPEN) && de->d\_inode) { can be cleaned up as dentry must be positive when it comes there.

# Cleanup Proposals

1. Perform trivial cleanup listed in 4.
2. Remove code setting negative dentry DCACHE\_LUSTRE\_INVALID then initializing d\_hash from ll\_lookup\_it(). Let llite just set d\_inode to NULL and return if client finds negative dentry but loses parent’s UPDATE lock.
3. Remove d\_rehash before d\_move in ll\_find\_alias.
4. When UPDATE lock is canceled, don’t flag in-use negative dentry DCACHE\_LUSTRE\_INVALID, but just unhash it and change negative dentry DCACHE\_LUSTRE\_INVALID checks to d\_unhashed in ll\_revalidate\_it.
5. Add lld\_flags field to struct ll\_dentry\_data, and define LLD\_LUSTRE\_INVALID to replace DCACHE\_LUSTRE\_INVALID flag. Add three functions (ll\_mark\_dentry\_invalid(), ll\_clear\_dentry\_invalid() and ll\_dentry\_invalid()) to set, clear, and test LLD\_LUSTRE\_INVALID flag in ll\_dentry\_data->lld\_flags. Replace all DCACHE\_LUSTRE\_INVALID reference with above three functions.