Idea of Metadata Writeback Cache for Lustre

Oleg Drokin

Apr 23, 2018

Current Lustre caching

- Data:
  - Fully cached on reads and writes in face of no contention
  - Really fast as the result (grant is another consideration)
- Metadata:
  - Only reads are cached
  - All modifications are sluggish as the result
  - Even non modifications like opens are also sluggish
- As the result – multiple proposals for extra caching were made
  - Amongst them subtree locks
  - PCC is another project aiming at this problem from another angle
    - Fujitsu had a similar one in the past

So how hard metadata caching could really be

- I set out to find limits of easy with a prototype
- If we create a dir, we know 100% all the names inside
  - Just get the exclusive lock and nobody else would interfere
  - We could accumulate normal names
  - Serve readdir out of dcache
  - Even store file data totally in pagecache without talking to mds
  - Ramfs of sorts
- Overall the idea sounds pretty simple, right?

Implementation notes

- Mkdir is a reint create RPC, no locks.
- Server actually has reint create handler, but it's not used
  - And sending a create intent results in LBUG
- Making client to send mkdir as intent create is pretty easy
- Making server return EXclusive lock if the create succeeded is easy as well.
- Flag such directories on the client as "fully locally owned"

Magic begins

- For “fully locally owned” directories we can override everything
  - All lookups are either in local cache or are safely negative
  - All creates go straight to dcache and stay there
    - Client side FID allocation allows for consistent FIDs even if we want to flush to server later
  - All unlinks just remove dcache entries
  - Same dir or subdir renames are dcache-only ops
  - Hardlinks in this subtree is really easy too
  - Stat just reads data from inode
  - Attaching file data to locally owned files is pretty easy.

But what if the lock is cancelled

- Iterate over the directory entries in the cache
- For every entry do intent-create RPC with “I got the parent lock”
  - We get EX lock back, for subdirs that means the subtree is preserved
  - For files that means we get to keep our file data safely until we establish layout and grab proper data locks
  - Other entries don’t care
  - Hardlink is a major complication since we cannot do create
- Once all entries are done with – drop the lock and the directory is magically visible to all clients.
- This is a real easy conversion path back to shared access unlike other approaches.

EXclusive metadata lock – like a data lock

- Allows the client to operate on locked directories without deadlocks
  - A hard requirement for the whole scheme
- Just like with data locks – we can send/execute metadata ops under metadata EX locks
- Every RPC that furnishes “parent EX lock” prolongs the lock so it does not time out prematurely

Data writeback handling

- We already have the data in the pagecache, but CLIO knows nothing about it.
- To assimilate data first we need the layout and data locks.
- We must enqueue the locks while still holding the exclusive layout lock so nothing can peek in the file
  - Very similar to HSM restore
- Once we got the locks – simply add CLIO data structures to existing pages (convenient cl_page_find()-> cl_io_commit_async())
  - Would be better to be able to just do cl_lpage_alloc
  - Thanks to Jinshan for guidance
- Once file reverts to normal Lustre file, with regular writeback

The result

- As expected, uncontended operations just fly at unbelievable speeds
  - 10x-20x improvement in createmany performance on local VMs
  - FPP mdtest with 16 clients – ~6M/sec cumulative ops
  - Unpacking linux kernel tarball – 10 seconds (vs 210s)
- Actual workloads improve too
  - Building Lustre in VM – 25%+ improvement on idle servers
    - Overloaded servers are not affecting WBC operations
  - Building rhel7.4 kernel on real HW 4.5m (vs
- Would really shine in interactive kind of workloads with congested servers

Limitations – "benchmark cache mode"

- Great "benchmark" workload handler
  - Create X files, stat, remove -> 0 RPCs need to be sent
- No accounting (changelog)
- Bursty flushes on lock cancels instead of smoothed trickling out
- Operating on preexisting directories is complicated.

Another mode – write behind cache

- Every operation creates suitable RPC that is sent asynchronously
- Userspace gets control right away so they are not impacted
- Smoothes server load – useful for real workloads
  - Untar archive and it starts to trickle out right away
  - We know that data we write WILL be used by other nodes
- No ‘cancelling of operations’, but changelogs become possible
- Easier to work with preexisting directories
  - Read in the data into cache and get an EX lock, done.
  - Readdir/readdir+ alike combining would help
  - Decided by the server

Other possible improvement ways

- Compounding multiple operations into a single network RPC
  - Now that we actually have string of operations cached
- DoM can get create+data sort of RPCs for small file writes
- Hooks for more permanent storage of cached data on clients
  - Log-based fs of some sort? Just cachefs?

Prototype limitations

- No hardlinks
- Root only file ownership on flush
- No error handling
- DNE status unknown
- Based on 2.11 release for rhel7.4 only
- No xattrs/posix ACLs
- No grants/limits/accounting
- Sync is noop
- No memory use limits
- Only “benchmark mode” implemented

Conclusion

- Many aspects are not as hard as they seemed at first
- Some parts are useful on their own
- Even limited implementations would have successful niches

- You can see my prototype patches linked from LU-10938

Questions?

Questions?