Immediate Mirroring Writes

See Immediate Write Mirroring Design for current design

Introduction

Currently the only way to get a redundancy for a file (or have it mirrored) is to run lfs utility. The file is supposed to be closed at that time or the utility interrupts and must be restarted from the beginning. If any application opens a file being replicated then the replication interrupts as well.
Another problem with such offline mirroring is resource consumption as the replication needs to read data from OSTs (disk IOs, then network transfers), then send it back to another OSTs (network transfers, then disk IOs). For any 1MB of data to be replicated we need: 1MB disk/network transfer to write original data, 1MB disk/network transfer to read original data and 1MB disk/network transfer to write a replica - 3 x disk/network bandwidth.
In contrast immediate mirroring should let clients maintain replica(s) while file is being used by regular apps and save 1 disk/network transfer - 2x disk /network bandwidth, not 3x.

Functional Specifications

The basic idea of the design is to let clients send modifications to original objects and its replicas at same time roughly. Up on file close MDS collects results of all modifications and makes a final decision whether the file is fully replicated.
When a replicated file is opened for write, from in-sync replicas we choose primary one and 1+ secondary ones. Depending on results of actual operations, primary and all secondary replicas may become up-to-date replicas again. Primary replica is used for changes and reads. Secondary replicas are used to apply all changes, but not for reads.

Changes to the disk layout

Layout component (representing mirrors) can be additionally flagged with:
LCME_FL_PRIMARY: component will be used as primary and extent locking will go to this component initially
LCME_FL_PARTIAL: component is involved in IMW and clients are supposed to send updates to this component; so far all clients reported clients reported all changes succeed on this component
LCME_FL_FAILED: at least one client reported failed operation on this component

Examples of layout transitions

Per-component state: U – up-to-date (not a real flag, no Stale flag); S – Stale P - partially mirrored; F - mirroring failed.

LOVEA	Componen 1	Component 2	Component 3	Component 4
before open	Uptodate	Uptodate	Uptodate	Stale
after open	/ primary	Stale / secondary	Stale/ secondary	Failed
client 1	Partial	Partial	Partial	
client 2	Partial	Failed	Partial	
client 3	Partial	Partial	Partial	
client 4	Partial	Partial	Failed	
After resync	Uptodate	Stale	Stale	Stale

Flags (in form of component layout) are accumulated in LOVEA during mirroring epoch. Initially the file had 3 uptodate mirrors (components 1-3) and 1 stale mirror (component 4). After initial open for write MDS updates the layout so now Component 1 is primary (up-to-date by definition), Component 2 and Component 3 are stale (will be used for immediate replication) and Component 4 is failed in this epoch because it was already stale. 4 clients write to the file and do replication. At some point they report their own result in a form of layout with flagged components. Client1 and client3 successfully wrote Components 1-3, client2 failed to write Component 2, client4 failed to write Component 3. Thus the new accumulated state is a non-replicated file. Given we differentiate the flags we can accumulate updates in LOVEA directly.

More lucky case:

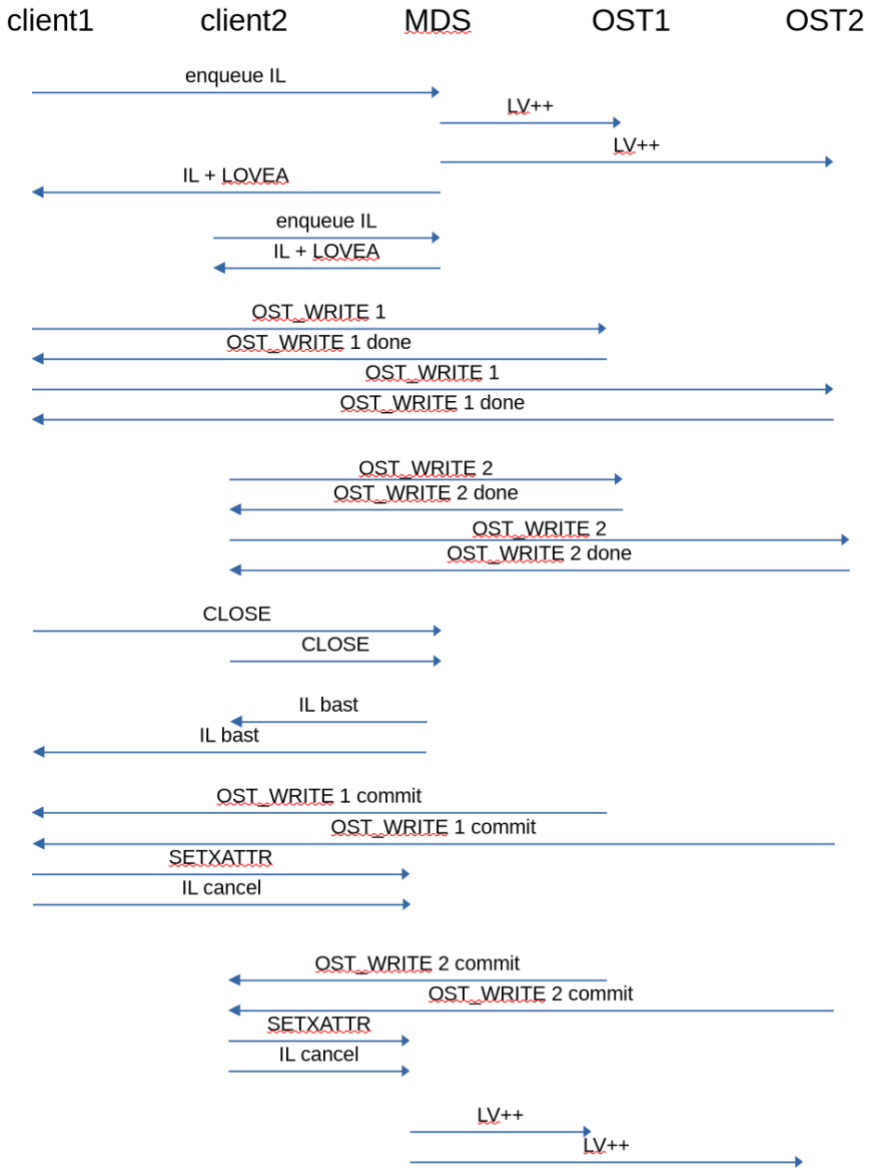
LOVEA	Componen 1	Component 2	Component 3	Component 4
before open	Uptodate	Uptodate	Uptodate	Stale
after open	/ primary	Stale / secondary	Stale/ secondary	Failed
client 1	Partial	Partial	Partial	
client 2	Partial	Partial	Partial	
client 3	Partial	Partial	Partial	
client 4	Partial	Partial	Failed	

After resync	Uptodate	Uptodate	Stale	Stale

This time Component 2 was successfully written on all the clients, so the file becomes a replicated one.

Changes to the network protocol

Inodebit locks get an additional bit - MDS_INODELOCK_IO. This is to let resync agent to notify clients involved in immediate mirroring to stop corresponding activity and report their status back.
IL – IO lock, LV – layout version



Client-MDS interaction

When a first client is going to make changes to a mirrored file, it follows the replication protocol and enqueues LAYOUT lock with LAYOUT_INTENT_WRITE. This initiates a new mirror epoch for file:

- layout's state changes from RO (read-only) to WP (write-pending)
- MDS chooses primary/secondary replicas updating the layout's flags
- layout's version increments
- MDS propagates this version to all involved OST to block IO under old layout
- MDS grants the client a lock with LAYOUT + IO bits

Another client opening this file finds the layout with primary replica marked, thus requests LAYOUT+IO lock to notify MDS about a new source of changes.
To save an enqueue RPC MDS can grant IO lock to client at open when such a file is opened by other client(s) already (like we do for Data-on-MDT).
Client doesn't cancel IO lock on its own, only upon MDS's request. MDS cancels client's IO locks in few cases:

- when all clients have a file closed - at this point MDS initiates epoch close to turn a file fully-replicated and make all qualified replicas available for reads
- when client can't update some replica, it can report this to MDS right away, MDS can force clients to close mirroring epoch so that clients don't need to keep updating already failed replica in a new epoch
- when a client is evicted, so MDS can't trust changes made by this client potentially

If client can't complete/commit IO before cancel expiration, then it reports current status of replication as is (incomplete means failed) and corresponding replicas are marked stale in the end.
When the last client closes a file MDS initiates a resync (not really resync, a better name is needed). Resync doesn't need to be synchronous in the context of close processing.
Like with FLR, the resync procedure involves the following steps:

- MDS changes layout's state from WP to SP (sync pending) to catch new opens
- MDS enqueues IO+LAYOUT lock to notify clients that the epoch is closing
- the clients send their cached layouts to MDS with LCME_FL_PARTIAL and LCME_FL_FAILED set properly to reflect which replicas are good
- MDS accumulates these flags in actual LOV EA
- the clients cancel their IO locks [probably we can pack LOVEA into cancel RPC?]
- MDS marks replicas with LCME_FL_FAILED as stale, drops LCME_FL_PRIMARY and increments layout's version - this is epoch close event
- MDS does not block new opens during resync procedure, but new changes can't start due to IO lock held during resync procedure
- MDS can interrupt resync procedure when IO lock has been acquired and new open is found

Any change to layout (outside of IMW flags) should cause existing epoch to close and a new epoch to open: IL locks are cancelled, current replication status is reported by all involved clients.
[think which replica we choose up-to-date if all replicas including primary one met a failure]
[think what component resync functionality belongs to: 1) works with LOVEA details – LOD 2) works with inodebit/extent locks]

Changes to CLIO

The basic idea is to push same page (vmpage) into few OSCs and then just let each OSC to operate on those pages as usual (mostly).
When CLIO is processing system calls (e.g. write) it creates so-called subios - originally to break the original IO into smaller IOs corresponding to stripes. We can re-use this mechanism for IMW, but this time LOV will be duplicating IOs for each OSC involved.
The kernel provides CLIO with a set of pages (from pagecache or userspace). Currently LOV pushes pages into corresponding OSCs which are calculated using striping information (number of stripes, stripe size, objects). To support IMW it will be pushing pages into few OSCs and we need to change few structures (e.g. struct cl_page, osc_page) to enable that.
LOV will need to recognize IMW-enabled layouts:

- use primary replica for reads
- use primary replica to order extent locking
- use primary and secondary replica(s) for all changes
- Find correct components for given offsets

The results of all changes (OST_WRITE, OST_SETATTR, OST_PUNCH, OST_FALLOCATE) are tracked in LOV on per-component basis. Llite is also aware of all changes (mmap?). Later, upon IO lock cancellation, llite will fetch LOVEA from LOV and send it to MDS to report replication status - for this purpose MDS_REINT_SETXATTR is used.
[in order to speed up development add a support for non-striped LOV_PATTERN_RAID1 pattern - basically just a set of object, no support for replica flags, etc.]

Compatibility

If an old client opens a mirrored file, then MDS just doesn't start a mirroring epoch. If any client opens a mirrored file with no primary selected (as found in LOVEA), then MDS doesn't start a mirroring epoch. If an old client opens a mirrored file with mirroring epoch started (as found in LOVEA), then MDS should create mfd and initiate mirroring epoch abort immediately: cancel IO bitlock (makes sense to notify clients it's abort so they don't flush data?).

Tasks

MDS

Client-MDS interaction:

- MDS should be able to create an empty file with few in-sync replicas with default striping (thus no lfs mirror resync is needed if IMW activity goes smoothly)
- MDS to reflect new mirroring epoch in layout upon first INTENT_WRITE, choose primary/secondary, update LOVEA
- MDS to recognize IMW case and apply for IO bitlock
  - e.g. mdt_object_open_lock() and mdt_lmm_dom_entry_check()
- MDS to update layout in case of eviction

- MDS to update layout in case of failover
- MDS to initiate resync agent upon last close
- Client to get and hold MIRROR bitlock until it's requested back
- Client to send up-to-date layout to MDS (MDS_REINT_SETXATTR) before releasing IO bitlock

Resync functionality

- Resync agent queue (few threads)
- Resync agent to request extent lock (file is supposed to be closed finally)
- Resync agent to enqueue IO bitlock and check no new opens have happened
- Resync agent to process intermediate layout to reflect accumulated change: mark successful replicas non-stale

llite

- llite to track all changes to a file in form of dirty-clean state on per-inode basis
- when a IO lock cancellation is requested by MDS, llite flushes all dirty pages (if found) and schedules IO lock cancellation
- Lock cancellation happens when all changes to used replicas have been committed
- OR lock cancellation happens if lock is about to expire this is needed to avoid import eviction leading application-visible errors – at cost of failed replicationDon't quite understand this?if MDS enqueues a lock and can't get it in time - whole import is invalidated which in turn means app-visible EIO. this is not good. so instead the client can try to flush/commit in limited time to get replica done, but cancel in time always: if it wasn't lucky and commit took too long, then we just lose replica, but not whole import.
  another option is to use glimpse to "poll" for flush/commit.

- Before lock cancellation llite requests LOVEA from LOV and sends it to MDS in form of MDS_REINT_SETXATTR. This LOVEA contains per-component flags reflecting replication status as described above.

[probably we can send a shrinked version of LOVEA containing updated components only?]

CLIO

- support for LOV_PATTERN_RAID1
  - Mostly to speed up development
  - Should be replaced with a composite layout supporting striped mirroring
- Duplicate PG_dirty and PG_writeback in a form of refcounter in cl_page
- lov_io_iter_init() to duplicate IO for each object
- lov_io_commit_async() to save list of pages passed in and repeat the call to cl_io_commit_async() with that list
- Lov_io_commit_async() to use a special callback and use that callback to maintain per-component flags if IO fails
- lov_lock_sub_init() to generate locks
- lov_lock_enqueue() to order locks (primary first)
- lov_init_raid1() to initialize subobjects and initialize co_slice_off properly to support multi-object IO in cl_page_alloc() and osc_page_init()
- lov_attr_get_raid1()
- lov_page_init_composite() to initialize same page in few OSCs
- osc_extent_make_ready() to recognize a case when page has been added into an RPC in another OSC and increment in-writeback refcounter
- osc_completion() to respect in-writeback refcounter
- osc_io_commit_async() to maintain cl_page's dirty state
- new set of methods similar to composit layout methods with IO/lock duplication
- brw_commit() to propagate result to LOV/llite and maintain per-component flagsThis looks good - one comment, remember that direct IO does not use OSC dlmlocks on the client. No problem I think but need to be kept in mind

Utils

- lfs mirror extent/resync to recognize IMW-enabled files
- lfs setstripe to support RAID1
- lfs getstripe to support new component flags

Implementation steps

1. Initial mirroring: single client, no errors, no concurrency
2. Errors at write, errors at commit
3. Concurrency: few clients, overlapping changes, open at resync, changes to layout during IO

Use Cases

- Normal cases:
  - A new mirrored file
  - An existing mirrored file
  - A mirrored file written from few clients with overlapped partial writes:
    - Read from correct replica
    - Write to all replicas
  - A mirrored file read from another client in parallel
  - Sync in any form, layout change causing epoch to close/re-open on clients
  - Pre-IMW client opens file for write – try to close epoch
  - Pre-IMW client opens file for read – send a special layout backwould this just be a layout from the primary mirror? and revoked if the primary changes?
  - A mirrored file opened for write gets a change to layout (e.g. append or removal of stale replica)
- Recovery cases
  - Client failed a write to primary
  - Client failed a write to secondary
  - MDS restarts
    - If MDS can't be sure about all involved clients, then abort epochs reported in-use by clientsshould have an upcall or call to MDS coordinator to trigger async replication as soon as file is closed?probable lamigo could notice this via a changelog record?
    - Clients should stop mirroring activity
  - OST restarts
    - OST_WRITE replays recover state
  - Client evicted from MDS
    - Epoch aborts, another clients are notified and stop mirroring activity
    - What about window when client is not aware and keep writing to OSTs?
  - Client evicted from OST
  - Primary OST is offline
  - Secondary OST is offline

Problems:

- Can use layout lock for epochs?When first looking at this, I decided we couldn't make this work and we needed a different lock bit, which I called ACTIVE_WRITER, but that's ~the same as the apoch thing. I am trying to find my notes...
  The code (an augment generated not-even-prototype) is here: https://review.whamcloud.com/c/ex/lustre-release/+/59791
  The doc is here, but it's pretty tentative:
  https://wiki.whamcloud.com/display/EXA/Immediate+Mirror+Design#ImmediateMirrorDesign-WriteOperationFlow
  Still trying to remember why I thought we shouldn't use the layout lock - It may have been the mode?because layout lock can be invalidated due to unrelated layout change?I think that may have been the issue, yeah.I think that may have been the issue, yeah.Maybe that, yeah

- Space management:
  - Object creation is not well coordinatedThe object creation shouldn't matter, since the layout will be initialized before any writes start
  - Writes to a replica can hit ENOSPCwrites should grab grant from all mirror OSCs before starting primary write, which should avoid any ENOSPC issues
- Llite
  - Need to get aware of that to some extentWhy does llite need to care about mirroring? Couldn't llite return the LOV EA (or at least a part) to the MDS without understanding any details.well, to join/leave epoch at least? we won't do this for every file?
  - All actual IO iterations (including locking and memcpy) are handled in kernel + LOV/OSC, thus we can't just repeat write for another replica – all need to be done under original extent lockI think I may disagree about the extent lock here - how do we do writes to a secondary mirror if the primary mirror is unavailable? There is at least some complexity herethen we should change primary. I think primary is also a way to order ldlm locking.Changing primary makes sense to me, I'm persuaded
- Pages, OSCs and RPCs
  - Page/RPC sharing is a problem@Patrick Farrell had discussed this a bit with @Bobijam Xu previously. The page itself is from the inode mapping, so it could be shared. It is the cl_page sharing that is the real issue.yes, this is where my thinking about cl_page stopped :)
    Agreed; Yingjin had some decent ideas in this vein he started on, details escape me but we should look at thatHere's @Qian Yingjin 's prototype here:
    https://review.whamcloud.com/c/ex/lustre-release/+/59958
    Yingjin, could you briefly explain the approach?The approach is as follows: the client only acquires the extent locks for the primary mirror, and buffer I/O only applied on the primary I/O (this means that cache pages are only maintaining by the primary mirror (the rbtree of OSC objects for the primary), not other slave mirrors). But when an OSC build I/O RPC (for buffer I/O or direct I/O) in osc_build_rpc(), it will also upcall to LOV layer, and build I/Os for other slave mirrors by using TRANSIENT pages. Here we ignore the space and quota for slave I/Os. When the primary failed, then the client needs to report to MDS. And then MDS will coordinate the data consistency recovery and mark the failed OSTs and objects and then select the new primary.I don't think we can ignore space problem (e.g. grants) because this would affect other grants users - they would hit ENOSPC having space granted before.
    I think we should try to introduce osc_page for every OSC page is involved in and go full path reserving grants, etc.
    in general trying to duplicate RPCs doesn't look good to me - pipes can be in very different states: import can be disconnected, no empty slot, etc.
    Agreed about duplicating RPCs. For space handling…. Hmm.
- mmap
  - RPCs (referring same page) aren't sent at same time