Testing Lustre for Robustness and Scalability

Cory Spitz and Chris Horn
Cray Inc.

LAD ‘14
Background

- Started with a great customer that motivated and pushed us to fix some long standing architectural issues

- Looking for comprehensive RAS

- But there is a fundamental single point of failure in the Lustre protocol

- Namely, ASTs can’t be resent
  - bugzilla.lustre.org BZ 3622, opened June 2004
  - Aka LU-7 and LU-5520
Goals

● **Cover as many RAS cases as possible**
  ● Nominal operation
  ● Failure cases
  ● Secondary failures

● **Survive a network flap (and lost traffic) (for some finite time) without suffering any client evictions**

● Don’t destabilize the codebase

● **Start regular testing to ensure that there are no future regressions**
Testing proved problematic

- Made best efforts to reproduce using in-house systems
  - However, these systems lack true scale

- But we really had to rely on the customer to give a thumbs-up/down on any changes

- A call to action; We can’t rely on customers to find all problems and validate all fixes

- Especially with RAS testing, which is too demanding on the datacenter and admins
What are the issues?

- Lots of issues; bugs started coming out of the woodwork
- What are all of the possible scenarios?
- What type of traffic could be lost?
- What behaviors exist between client & server?
- What is the proper test response?
Where It All Goes Wrong

Adventures in Avoiding Client Evictions
The Goal

- **Survive loss of traffic without evictions**
  - Finite time (we won’t wait forever)
  - Minimize impact on application performance
- **When a packet is dropped:**
  - Client disconnect/reconnect
  - Packet needs to be resent
  - Avoid repeating the cycle
  - Bad router? → Modify routing table
Lost Connections

**Router Issues**
- Using bad routes wastes time and resources
- Need to wait for router ping
- Remote interface death is potentially worse
  - need to wait for interface marked down then another router ping
    (asymmetric route failure detection)
- Cray able to leverage node health to help
  - You can too! Use lctl to mark peers up/down if you know the route is bad
- `router_ping_timeout` and ping interval tuning

**The connect RPC**
- Timeliness is important
  - Often dependent on proper router health detection
- Clients couldn’t connect if they had outstanding RPCs (LU-1239)
- Want quick-ish reconnect intervals
Let’s talk

● **Bulk I/O**
  ● Already resent (yay)
  ● Handling different between nominal and failure/recovery
  ● Want timeouts to happen quickly
    ● at_min, at_max tuned so we wait long enough, but not too long
  ● Found bug with early reply
    ● Fix introduced a regression (sorry about that)

● **AST**
  ● Blocking, Cancel, Completion, Glimpse (and replies)
  ● LU-5520 landed, ASTs are now resent (yay)
    ● LU-2827, LU-5266, LU-5496, LU-5579, LU-5530
  ● Broke POSIX compliance (oops)
    ● LU-5569, LU-5581
  ● Client reconnect and route health detection is very important
    ● (lost replies can lead to eviction)
Let’s talk some more

● **AST (cont.)**
  ● ldIm_enqueue_min tuned to allow resend logic to work its magic
    ● ldIm_enqueue_min = max(2*net_latency, net_latency + quiescent time) + 2*at_min
  ● Best effort
  ● Will open LUDOC to share what we’ve learned
Our test response

● Unit tests can’t cover it all, we need lots of manual testing

● Continue the typical tests, but ratchet up what constitutes passing. Look at data verification and client evictions.
  ● Failover/failback
  ● Router death
  ● Remote interface death (cable pulls)
  ● Total network flap
  ● Blade failure (Cray HSN resiliency)
  ● Warmswap (Cray HSN resiliency)

● Create secondary failures
  ● e.g. fail a router during recovery
Next steps, increase the level of difficulty

- Drop a certain % of traffic (via FAILLOC failure injection)
  - Incorporate this into regular workload testing
  - SWL testing for releases

- Implement an NRS policy to simulate high server load
  - Stress ptlrpc state machine, recovery, and adaptive timeouts
  - Ditto for test usage, but need to be careful about valid evictions

- Use imperative recovery to trick clients into reconnecting

- Begin combinations of the above
Reference

- LU-5520 Idlm: resend AST
  - Fallout:
    - LU-2827 mdt: Also handle resend for layout-lock
    - LU-5266 Idlm: granting the same lock twice on recovery
    - LU-5496 Idlm: granting the same lock twice on recovery
    - LU-5496 Idlm: reconstruct proper flags on enqueue resend
    - LU-5579 Idlm: re-sent enqueue vs lock destroy race
    - LU-5530 mdt: Properly match open lock and unlock
  - Fixes tangentially related to resending AST callbacks:
    - LU-5569 recreating a reverse import produce a various fails.
    - LU-5581 Idlm: evict clients returning errors on ASTs
  - Enhancements related to resending AST callbacks:
    - LU-4942 at: per-export lock callback timeout

- LU-4578 ptrprpc: Early replies need to honor at_max
  - Fallout: LU-5079 ptrprpc: fix early reply timeout for recovery

- LU-5073 ptrprpc: unlink request buffer correctly
  - LU-5073 ptrprpc: prevent req completion until LNet drops ref
  - Fallout:
    - LU-5259 ptrprpc: request gets stuck in UNREGISTERING phase
    - LU-5341 ptrprpc: rpc times out in unregistering phase

- LU-5528 ptrprpc: fix race between connect vs resend
  - LU-5528 ptrprpc: race at req processing

- Client connect related:
  - LU-793 ptrprpc: allow client to reconnect with RPC in progress
  - LU-1239 Idlm: cascading client reconnects
Merci!

Also, very special thanks to Xyratex for solving LU-7/LU-5520 and for assistance with the ensuing fallout