Imperative Recovery

Nicolas.Williams@sun.com

Background

- Lustre is a distributed filesystem with POSIX semantics
- POSIX doesn't talk about crashes, much less recovery
  - POSIX only says when different processes see others' transactions, and when transactions must be on-disk
- Clients can crash, servers can crash, networks can partition, and you can get multiple failures

Background

- POSIX semantics + distributed + crash recovery → hard
  - Recovery can take a long time, or so I'm told
  - I've just started on this project, so I've no real data yet

What Makes Lustre Recovery Slow

- Transactions must be recovered in order
- Servers have to be [back] online
- Active clients have to be around when their transactions' time to recover comes up
  - In generic deployments clients → heterogeneous
  - Re-connect waits
- Network partitions, slow server reboots/takeovers, slow clients → wait for long times

Solutions

- Fundamental protocol improvements
  - > VBR?
- Management improvements
  - > Health Network
  - > Imperative Recovery

Lustre Core Protocol Changes

Not this presentation!

Health Network

- Clients (including OSTs) will ping next hop to MGS
  - Can be an LNet router, or some server assigned for this
  - Or the MGS itself
  - Pings include reboot generation ID
- Servers will ping too, w/ status
- Routers will keep track of live clients and servers
  - Simple dead client detection heuristics
  - Compressed updates sent upstream towards MGS
- Ultimately MGS will keep track of live clients
  - On reboot must recover cluster status info

Health Network

- Admins will be able to see status of cluster
  - Network partitions
  - Dead clients
  - Would be nice to add recovery progress / ETA info
    - Probably a follow-on project
    - Should export status info via MGS PTLRPC service

Health Network Protocols

- Client ⟶ router/MGS will be simple RPC based on LNet selftest code
  - Productionize LNet self-test RPC
    - Make it a first-class, re-entrant, reusable facility
    - Add ASTs? (see imperative recovery)
- Router ⟶ router/MGS will use the same simple RPC protocol
  - Don't need no stinking re-transmissions – DCD
  - No transactions ⟶ no replays, no need for PTLRPC

Health Network Protocols

- Client joins cluster → starts pinging
- Clients drive the pings – clients are never pinged
  - For firewall traversal...
- Clients identified by client ID?
  - Need to detect multi-pathing to distinguish client death from network partitions
  - Router death → network partition?

Imperative Recovery

- From bz18767

  “There are 2 issues to address....

  1. Timely reconnection of all clients to the rebooted/failover server.

  2. Timely end to the recovery window when all clients that can possibly participate in recovery have reconnected.”

Imperative Recovery

- Server fails →
  - Server reboots / HA backup takes over →
    - Server sends status update via Health Network
- Recovery must start when the server comes back
  - > Waiting here → bad
  - > Today clients may not notice right away – too passive
  - > Today there's a wait for all clients to reconnect
- Solution:
  - > actively tell [live] clients to begin recovery
  - > close re-connect window to dead clients

Imperative Recovery & Health Net

- If you know the list of all live clients...
- ...and they've all reconnected...
- ...reconnect window can be closed, recovery can start

Imperative Recovery Protocol

- Also simple RPC?
  - > As with health network, a hierarchical system to reduce the number of messages seems useful
  - > Unicast over LNet – for firewall traversal
    - – Multicasting might be an option in some cases? Would require new LNet features
- "begin recovery" order is an AST-like operation
  - > SRPC needs ASTs, or maybe ping replies say "recover!"
- No retransmissions needed?
  - – ACKs not really needed – DCD → implicit ACKs
  - – Send order AST many times in case of loss?
  - – Order ID/# to distinguish re-transmits from new orders

Putting it All Together

- Imperative recovery has *soft* dependency on health network
  - No srv status@MGS → recov order is manual
  - But must know how/where to unicast!
- Given both we can shorten the time it takes for clients to begin recovery
  - Faster client recovery start → less time waiting for clients to re-join
  - Knowing what clients are alive → reconnect window can close sooner

Status

- Early, ***early*** stages of design and planning
- First task: design simple RPC protocol and adapt LNet self-test code
  - Wait, first task is to design/impl hierarchical update net
- Second task: clients/servers ping routers, routers ping next hop
- Third task: implement ASTs?, orders
- Fourth task: implement remainder of DCD
  - Aggregation of DCD data
  - Interfaces for displaying / protocol for retrieving status

Questions

- From me, for you:
  - Is this the right track?
  - Is this too ambitious? bz18767 talks of using pdsh to send imperatives – easy to script, but not fw friendly
  - Should we prototype based on PTLRPC first?
  - Should we add redundancy for status information?
    - Not just MGSes but also MDTs should track cluster status?