Lustre* Network Failure

Liang Zhen
High Performance Data Division, Intel ® Corporation

Background

- More and more nodes in cluster
  - Tens of thousands of client nodes, hundreds of server nodes
- Timeout is not scalable
  - Current timeout relies on service time
  - Disk seek time is unpredictable.
  - Latency = (service time) * N
- Lustre* router
  - Many large sites have routers
  - Routers can fail, packets can be dropped
- Lustre* is not robust enough to handle packets loss
  - Debug & test on direct connected system

How we inject network failures

- OBD_FAIL_LOC
  - Change code for each single failure case
  - not random, always the same RPC state machine
- Power cycle or unplugging cables?
  - Too expensive, very slow
  - Can’t afford to repeat failed cases for thousands of times.
- Low level network stack failure injection
  - Different control commands for different networks
  - can’t filter messages

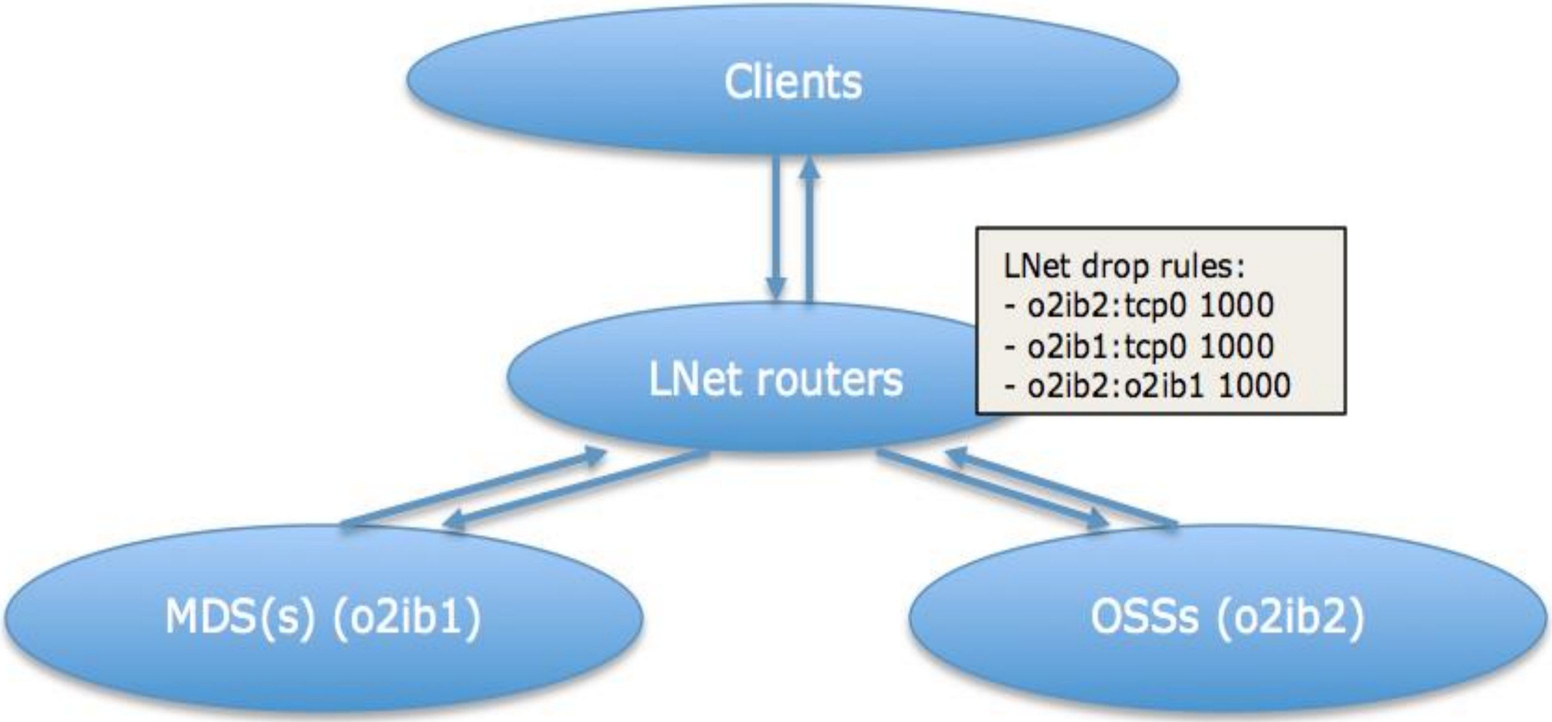
Lustre* network failure simulation (1/2)

- In core LNet
  - Independent to network type
  - Filters can understand Lustre* network protocol
- Control via “lctl” command
  - Drop Rule
    - Drop messages at specific rate or duration
  - Delay Rule
    - Delay messages for a few seconds at specific rate
- Filters
  - Portal (service ID)
  - Message types
  - Source/destination network addresses

Lustre* network failure simulation (2/2)

- Run simulator on routers
  - No backport, no version compatibility issue.
- Sample commands:
  - Lctl net_drop_add –source *@tcp –dest 192.168.1.102@o2ib --rate 10000 –portal 15 –portal 16 –message PUT
  - Lctl net_drop_list
  - Lctl net_drop_del –source *@tcp

Lustre* Network Bermuda Triangle



*Other names and brands may be claimed as the property of others.

Exposed problems

- Eviction and eviction…
  - Lock AST loss
  - Lock enqueue reply loss
- Unreasonable timeout
  - Service time and network latency calculation have defects
  - Adaptive Timeout (AT) mixed service timeout and network timeout
- Mis-matched replies
  - Sometimes service can't drop resent request when early reply is lost
  - Multiple replies fit in the same reply buffer

Exposed problems (1/3) Evictions

- Blocking AST loss
  - client does not even know
  - Solution: resend blocking AST
- Completion AST loss
  - Client cannot cancel a lock which is not granted yet
  - Solution: resend completion AST
- Lock enqueue reply loss
  - Both above situations
  - Lock timeout should be longer than client RPC timeout?
    - Mixed two different timeout systems, it is bad
    - What if resent request lost again?

Exposed problems (2/3) Timeout

- Adaptive timeout is a “best guess”
  - 125% of estimate service time + 5s
  - Early reply if server found it may take longer than “best guess”
- What if unexpected situation happened
  - Early reply loss
    - Overhead of reconnect and resend
  - Extremely large service time
    - service time may include may phases of an operation, for example, revoke lock + data flush + lock cancel,
    - What if any of these messages lost
  - Client eviction, even with resent AST

Exposed problems (3/3) Router

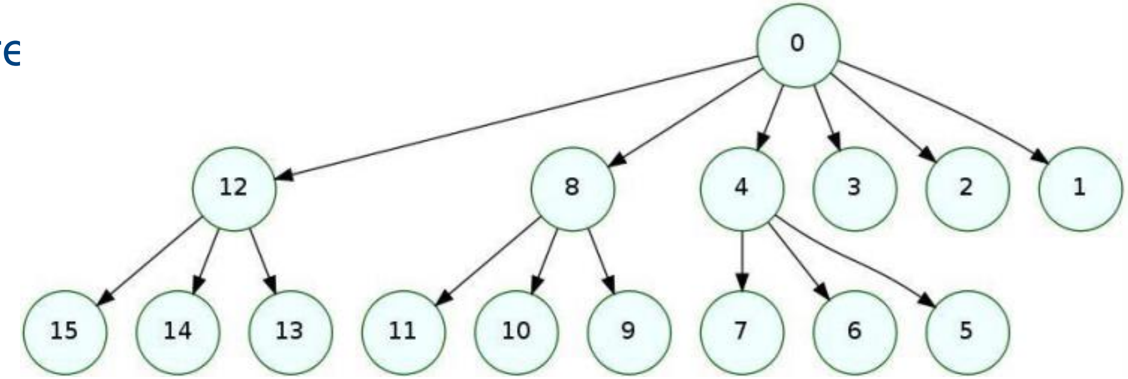
- Router pinger
  - Take long time to find out a dead router
  - Take long time to detect dead->alive NI on routers
- Avoid to use potentially dead/congested router
  - Last alive of routers
- Regular message to update NI status on router
  - Check source network of messages from routers.

RAS improvements

- Primary fault diagnosis based on resilient collective health protocol
  - Independent of storage service latency
  - More scalable
- Separate network & node fault handling
  - Simple retry on network failure
  - Full recovery on peer failure

Fast Forward components for RAS

- Gossip
  - Peer health monitoring
  - Fault tolerant O(log n) state distribution
  - Query & notification APIs
- Collective RPC
  - Arbitrary membership
  - Fast fail on member failure
  - Idempotent



Separation of network & peer failure handling

- Network fault handing
  - Make all RPC steps a round-trip
  - Make all RPC steps idempotent
  - Retry active RPC steps

- Peer failure handing
  - Assume peer healthy until notified otherwise
    - Robust lock callbacks
    - Large fixed timeouts catch complete deadlock or bugs
  - Global client eviction

Summary

- Better testing framework
  - Found more corner cases and issues
  - Short term fixes

- RAS improvements
  - Real solution
  - Take longer time

intel®

Legal Disclaimer