Understanding Lustre Timeouts

Rick Mohr  
mohrrf@ornl.gov

James Simmons  
simmonsja@ornl.gov
Overview

• Lustre uses timeouts in several places as a way of detecting problems and ensuring forward progress
  – Packet loss on the network
  – Prevent a crashed client from blocking IO from other clients

• Too many timeouts to discuss all of them in detail

• Our goals:
  – Discuss some of the most relevant timeouts used in Lustre
  – Describe the purpose of the timeouts and any relationships between them
Types of Timeouts

• Timeouts can be (roughly) split into two groups

Lustre

• Ensure RPCs complete in a finite time
  – Bulk data transfers
  – Granting/revoking locks
  – Client evictions
• Other uses
  – Imperative recovery
• Printed to console

LNet

• Ensure point-to-point communications across the network complete in a finite time
  – LND timeouts (driver-specific)
  – General LNet transactions
  – Router health
• Not printed to console
  – Check Lustre log or enable printing for “neterror”

```
lctl set_param printk+=neterror
```
Lustre Timeouts
Adaptive Timeouts

• By default, Lustre enables adaptive timeouts
  – Servers track completion times for RPCs and report that info to clients
  – Clients use info to estimate timeouts for future requests
  – Estimates can dynamically change based on performance of system
  – Servers can also send an early reply to client asking for more time

• Since timeouts are dynamically determined, there aren’t many parameters to tune
  – Three most important are at_min, at_max, and ldlm_enqueue_min
  – A few others control things like time increment for early replies, time window for tracking timeout history, etc.
  – Default values probably work fine for most people
Adaptive Timeouts (cont.)

- **at_min** (default = 0 s)
  - Minimum time server will report for processing RPC
  - Not the actual time taken to handle an RPC
  - Can be increased to avoid timeouts for transient issues

- **at_max** (default = 600 s)
  - Upper limit of any service time estimate
  - If reached, the RPC will time out
  - Lowering this value could detect problems faster, but setting it too low will just result in spurious timeouts for minor slowdowns
  - Setting `at_max = 0` will disable adaptive timeouts
Adaptive Timeouts (cont.)

- `ldlm_enqueue_min` *(default = 100 s)*
  - Minimum timeout to enqueue a lock request
  - Lock requests can be more complicated than other RPCs (ex – may need to revoke lock on another client)
  - Using same minimum as other RPC requests *(at_min)* doesn’t necessarily make sense

- Commands to set these values and query historical info about adaptive timeouts are given in the Lustre manual
Static Timeouts

• Static timeouts are controlled by two parameters
  
  • **timeout** (default = 100 s)
    – Time that client waits for server to complete an RPC
    – Sometimes referred to as the ”master timeout”
    – In Lustre code, it is identified as `obd_timeout`
    – Most other timeout values are calculated from this one
  
  • **ldlm_timeout** (default = 20 s/6 s for OST/MDS)
    – Time that server waits for client to reply to a lock cancellation request
Derived Static Timeouts

• Other timeouts based on `obd_timeout`
  - Imperative recovery timeout = $4 \times \text{obd\_timeout}$
  - LDLM completion AST timeout = `obd_timeout`
  - LDLM blocking AST timeout = $\text{obd\_timeout} / 2$
  - Time to wait for OSC connection to become active = `obd_timeout`
  - OBD ping interval = $\text{obd\_timeout} / 4$
  - Time server waits for client reply to AST callback = $\text{obd\_timeout} / 2$
  - PTLRPC health check = $\text{obd\_timeout} \times \frac{3}{2}$ (instead of `at\_max`)
  - Watchdog timeout = $10 \times \text{obd\_timeout}$

• Harder to fine-tune timeouts when they are all related
Static ldlm_timeout

- PTLRPC requests set a static time based on ldlm_timeout and obd_timeout
  \[ \text{min}(ldlm\_timeout, obd\_timeout / 3) \]

- But there are restrictions on setting ldlm_timeout:
  - If ldlm_timeout > obd_timeout,
    then ldlm_timeout = obd_timeout / 3

- There doesn’t appear to be any reason to set ldlm_timeout more than a third of obd_timeout
Bulk IO Timeout

- There is a timeout for bulk IO that is not documented in the Lustre manual
  - `lctl get_param bulk_timeout` (default = 100 s)

- If the deadline set in the RPC request is greater than `bulk_timeout`, then `bulk_timeout` is used for the deadline instead
  - Sets a hard limit on time for bulk IO regardless of other timeout values
  - Lustre code doesn’t seem to alter the deadline in the RPC request itself
LNet
General LNet Timeouts

• Some LNet timeouts are not tied to a specific LND

• `lnet_transaction_timeout` (default = 150 s)
  – Message dropped if not sent when timeout expires and retry count not reached
  – If response expected, message dropped if response not received within the timeout

• `lnet_lnd_timeout`
  – Not independently set; derived from `lnet_transaction_timeout`

\[
lnet_lnd_timeout = \frac{(lnet_transaction_timeout - 1)}{(lnet_retry_count + 1)}
\]
LNet Router Timeouts

• In a routed environment, nodes will ping LNet routers periodically to keep track of which routers are alive or dead
  – Frequency of pings is controlled by `live_router_check_interval` and `dead_router_check_interval`
  – If a reply is not received before `router_ping_timeout` expires, the router is considered dead

• `router_ping_timeout` (default = 50 s)
  – Should be consistent with LND timeout
  – If LND timeout is increased, may need to also increase `router_ping_timeout`
ko2ibInd Timeouts

- **timeout**
  - How long to wait on transmissions before considering them failed
  - If not specified, value will be set to `lnet_lnd_timeout` discussed earlier
  - Setting timeout lower can cause problems to be detected sooner, but setting too low could lead to spurious timeouts
  - Should be based on properties of local fabric (size, congestion, etc.)
  - Recommendations can vary from 10 s to 100 s
ko2iblnd Timeouts (cont.)

• peer_timeout (default = 180 s)
  – Used to determine when a peer is down
  – Default values comes from generic LNet layer, but value set for ko2iblnd module will propagate up to LNet layer
  – In a routed environment, peer_timeout should be enabled only on the routers (set peer_timeout = 0 on clients and servers)
  – In general, peer_timeout should not be less than LND timeout
  – For ko2iblnd, peer_timeout should be at least twice the value of the keepalive option
Summary

• Timeouts are key to ensuring Lustre resiliency
• Optimal values for timeouts are often system-specific
  – Adaptive timeouts can help avoid the need for tuning
  – If changes are needed, knowing the relationships between timeout values can be helpful
  – Sometimes trial and error is still needed
• There are more timeouts in Lustre than what are covered here
  – Check Lustre manual and source code to learn more
Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.
Questions?