LNet Transaction Timeouts

Overview

After reducing the transaction timeout, which drives the LND timeout, to 5 seconds, lnet_selftest has been failing regularly mainly on ARM systems. ~~LU-11389~~ - lnet-setltest test smoke fails with 'lst Error found' RESOLVED has been opened to track these failures. A patch, https://review.whamcloud.com/#/c/33231/, has been landed on 2.12 to increase the transaction timeout to 50 seconds, the original LND timeout value before LNet Health.

The problem is scene only in lnet_selftest scripts. This is most likely due to the heavy traffic lnet_selftest generates. lnet_selftest has no failure handling code. For example, if an ACK for a PUT expires, there is no handling of that in the lnet_selftest code. It simply counts this as a failure of the test. With LNet Health's lower timeouts lnet_selftest should be modified to handle timeouts on ACK and REPLY.

Furthermore, this patch is problematic when you turn on health. Such a large timeout value basically renders Health functionality irrelevant. When the LND times out the Health code attempts to resend the message. But if the timeout is larger than the ptlrpc timeout, then the resend becomes useless at this point. The goal is to try and do multiple retries before the ULP timeout, in this case ptlrpc.

The current configuration burden on the admin lies in attempting to align the FS, PTLRPC and LNet timeouts in a way that makes sense. The goal of the lnet_transaction_timeout is to simplify this configuration burden somewhat, by having only one timeout for all LNet. LND timeouts are derived from that timeout.

The LND timeouts should not be long. The LND timeouts track the timeout for each LND message, not RPC message. These timeouts ensure that an LND message makes it on the wire within the timeout provided. If the LND message requires a response, for example in the o2iblnd case as described in the Health HLD, then this timeout ensures that a response message is received in a timely manner. These LND messages/responses do not traverse the entire software stack. IE they do not need to go up to ptlrpc before they are processed. They are processed in the LND. It follows then that if there is a 50 second delay before the LND messages complete, then the connection is in a problematic state. In fact we see the connection closes and RPC messages fail. The LNet Health retry mechanism, will preempt this process, by closing the connection and re-establishing it, potentially on a different interface to attempt and send the message. If this is done multiple times and the message hasn't gone through yet, then it is safe to say that the peer is in problematic state and error handling should now be delegated to the ULP.

As mentioned the benefits of Health will be nullified with this patch.

Solution

It is understood that with LNet Health disabled having a lower timeout can result in similar issues as we're seeing. Therefore, a proper solution is to have different default timeouts. One when health is disabled. And another when it is enabled. The solution outlined below:

1. Make Health enabled by default
2. Health defaults are:
    a. lnet_transaction_timeout = 10
    b. lnet_retry_count = 3
    c. lnet_health_sensitivity = 1
3. If lnet_health_sensitivity is set to 0 then set the following defaults
    a. lnet_transaction_timeout = 50
    b. lnet_retry_count = 0

Solution Justification

First Lustre uses an adaptive timeout. It wouldn't make sense to keep changing the LNet timeouts along with the adaptive timeout. It will make debugging LNet extremely confusing and will not add any benefit. Second, I had already implemented a way to pass an override timeout by changing the LNetGet() and LNetPut() APIs, but decided against including it in the final feature, because it wasn't being used anywhere, and as I mentioned above, having a moving timeout is not good design in case of LNet.

The reason adaptive timeout makes sense for Lustre is because the adaptive timeout is driven by a request/response mechanism. So Lustre sends a message to the server, the server responds saying, I'm too busy, so that triggers the timeout to change. There is no such mechanism for LNet, and I would argue against adding that, since the lower protocols do that for us, IE IB/TCP. So there is no need to rehash that implementation.

In summary, having a moving timeout is not a good idea for debugability reasons and because this makes LNet rely on an RPC concept, which is completely different from LNet messages. It makes more sense to build LNet health (which the LNet timeouts are part of) around the needs of LNet.

It is then up to the admin to adjust these timeouts for their purposes. The admin will need to tune the different timeouts over the system. This covers how to tune the timeouts across a Lustre FS.