The cascading timeouts problem and the solution

Peter Braam, Alex Tomas

February 9, 2008

1 Introduction

Creation of OST objects from MDS and clustered metadata introduced new problem - cascading timeouts. The core of the problem is nesting requests. As in the current model we use timeouts to maintain connection's state we're starting to have a problem: dependencies between requests arise dependencies between connections.

The situation we see very often:

- client C1 is given lock L1
- client C1 sends mkdir request to MDS1
- MDS1 sends object creation request to MDS2
- MDS2 fails
- client C1 gets timeout on the mkdir request and makes the import disconnected
- client C2 asks for lock L1
- MDS1 finds the lock is given to client C1 and sends BL callback to client C1
- client C1 can't answer because of disconnectivity and MDS1 evicts him

2 Functional Specification

We don't know real dependencies between requests (we can, but that would makes code much more complex, so we don't consider a fine-grained solution for a while). Therefore, we have to treat all connections are dependent each to other. Failure of request to node N can cause failure of any other request in our model. To prevent related requests (and their connections) from get failed, we have to know failed node before any dependent timeout. This is job for pinger. Then we suspend all regular timeouts on all imports until we're sure all nodes are ready to continue.

That could make all recovery activity serialized: node by node and not >1 node being recovered at a time. This doesn't look acceptable from scalability point of view. To work this limitation around, pinger should check state of all suspended connection and starts recovery if the node being pinged hasn't respond on time.

There is special case when timeout comes without request: lock cancel timeout. Server gets them if a client doesn't cancel lock on time. A client can wait for recovery of another MDS holding a lock from first one. To handle this case, server node should use the same logic as clients do: ping all servers and suspend timeouts (including lock cancel timeout) until all servers are recovered.

3 Use Cases

3.1 Single MDS failure

The pinger sends ping requests to all server nodes. Once node doesn't reply on time, ptlrpc raises global flag, disconnects failed one and starts recovery on it. Every ordinary timeouts recharge while the global flag isn't zero. As failed import is recovered, ptlrpc drops the global flag.

3.2 Single MDS failure with dependent requests

The pingers just observed one MDS node doesn't respond, it marks all the imports "suspended" and starts recovery for failed import. An request that depends on failed MDS indirectly gets timeout, finds own import marked "suspended" and waits until recovery of failed node is completed. As failed import goes to FULL state, ptlrpc drops suspend counter and wakes that request up.

4 Logic Specification

All the changes should stay at ptlrpc layer. Imports get suspend counter that's used to force import to ignore timeouts. The code uses two primitives to wait for a reply: `ptlrpc_queue_wait()` and `ptlrpc_set_wait()`. Their expiration callbacks check whether the global flag is greater than 0. If it is then wait on dedicated waitqueue until all the nodes are ready again. After that the callbacks should recharge timeout and repeat awaiting. The suspend counter must not be taken into account in all the code, but expiration callbacks of `ptlrpc_queue_wait()` and `ptlrpc_set_wait()`. This way all the recovery paths keep working.

The routine `waiting_locks_callback()` also checks whether timeouts are suspended and doesn't force lock cancel and doesn't evict clients.

If the pinger observes suspended import didn't get a reply, then it disconnects the import and starts recovery for it.

5 State Management

As first failure happens, a node moves to recovery state that can be characterized by the following:

- all imports suspend regular timeouts
- global suspend flag is incremented
- a pinger is used to maintain import's state

Any subsequent failure (can be discovered by pinger only) is reflected by incrementing suspend counters. A node leaves this state as soon as suspended counters get back to zero. This means all the failed imports are recovered now and regular timeout mechanism can be used to maintain import's state.

6 Alternatives

The proposal focuses on a client side (including client's code on MDS), but we also can give MDS an active role. For example, server node could send intermediate "wait more" reply on per-request basis. This way we could have finer grained mechanism to detect dependencies. This alternative solution looks more complex, though.

7 Focus for inspection

- is there any race between few imports getting failed at the same time?
- how to differ imports belong to different filesystems?
- can the pinger observe the server has changed? should it fail import in the case?
- if we lose ping request or reply, then we make a decision the import is failed and will try to reconnect. we'll be refused because the server is still processing our request. I'd allow server to accept connections with no requests to be replayed.