PG REMOVAL

See OSD::_remove_pg, OSD::RemoveWQ

There are two ways for a pg to be removed from an OSD:

1. MOSDPGRemove from the primary
2. OSD::advance_map finds that the pool has been removed

In either case, our general strategy for removing the pg is to atomically set the metadata objects (pg->log_oid, pg->biginfo_oid) to backfill and asynronously remove the pg collections. We do not do this inline because scanning the collections to remove the objects is an expensive operation.

OSDService::deleting_pgs tracks all pgs in the process of being deleted. Each DeletingState object in deleting_pgs lives while at least one reference to it remains. Each item in RemoveWQ carries a reference to the DeletingState for the relevant pg such that deleting_pgs.lookup(pgid) will return a null ref only if there are no collections currently being deleted for that pg.

The DeletingState for a pg also carries information about the status of the current deletion and allows the deletion to be cancelled. The possible states are:

1. QUEUED: the PG is in the RemoveWQ
2. CLEARING_DIR: the PG’s contents are being removed synchronously
3. DELETING_DIR: the PG’s directories and metadata being queued for removal
4. DELETED_DIR: the final removal transaction has been queued
5. CANCELED: the deletion has been canceled

In 1 and 2, the deletion can be canceled. Each state transition method (and check_canceled) returns false if deletion has been canceled and true if the state transition was successful. Similarly, try_stop_deletion() returns true if it succeeds in canceling the deletion. Additionally, try_stop_deletion() in the event that it fails to stop the deletion will not return until the final removal transaction is queued. This ensures that any operations queued after that point will be ordered after the pg deletion.

OSD::_create_lock_pg must handle two cases:

1. Either there is no DeletingStateRef for the pg, or it failed to cancel
2. We succeeded in canceling the deletion.

In case 1., we proceed as if there were no deletion occuring, except that we avoid writing to the PG until the deletion finishes. In case 2., we proceed as in case 1., except that we first mark the PG as backfilling.

Similarly, OSD::osr_registry ensures that the OpSequencers for those pgs can be reused for a new pg if created before the old one is fully removed, ensuring that operations on the new pg are sequenced properly with respect to operations on the old one.