BACKFILL RESERVATION

When a new osd joins a cluster, all pgs containing it must eventually backfill to it. If all of these backfills happen simultaneously, it would put excessive load on the osd. osd_num_concurrent_backfills limits the number of outgoing or incoming backfills on a single node.

Each OSDService now has two AsyncReserver instances: one for backfills going from the osd (local_reserver) and one for backfills going to the osd (remote_reserver). An AsyncReserver (common/AsyncReserver.h) manages a queue by priority of waiting items and a set of current reservation holders. When a slot frees up, the AsyncReserver queues the Context* associated with the next item on the highest priority queue in the finisher provided to the constructor.

For a primary to initiate a backfill, it must first obtain a reservation from its own local_reserver. Then, it must obtain a reservation from the backfill target's remote_reserver via a MBackfillReserve message. This process is managed by substates of Active and ReplicaActive (see the substates of Active in PG.h). The reservations are dropped either on the Backfilled event, which is sent on the primary before calling recovery_complete and on the replica on receipt of the BackfillComplete progress message), or upon leaving Active or ReplicaActive.

It's important that we always grab the local reservation before the remote reservation in order to prevent a circular dependency.

We want to minimize the risk of data loss by prioritizing the order in which PGs are recovered. We use 3 AsyncReserver priorities to hand out reservations. The highest priority is log based recovery (RECOVERY) since this must always complete before backfill can start. The next priority is backfill of degraded PGs (BACKFILL_HIGH). The lowest priority is backfill of non-degraded PGs (BACKFILL_LOW).