OSD

CONCEPTS

Messenger

See src/msg/Messenger.h

Handles sending and receipt of messages on behalf of the OSD. The OSD uses two messengers:

1. cluster_messenger - handles traffic to other OSDs, monitors
2. client_messenger - handles client traffic

   This division allows the OSD to be configured with different interfaces for client and cluster traffic.

Dispatcher

See src/msg/Dispatcher.h

OSD implements the Dispatcher interface. Of particular note is ms_dispatch, which serves as the entry point for messages received via either the client or cluster messenger. Because there are two messengers, ms_dispatch may be called from at least two threads. The osd_lock is always held during ms_dispatch.

WorkQueue

See src/common/WorkQueue.h

The WorkQueue class abstracts the process of queueing independent tasks for asynchronous execution. Each OSD process contains workqueues for distinct tasks:

1. OpWQ: handles ops (from clients) and subops (from other OSDs). Runs in the op_tp threadpool.
2. PeeringWQ: handles peering tasks and pg map advancement Runs in the op_tp threadpool. See Peering
3. CommandWQ: handles commands (pg query, etc) Runs in the command_tp threadpool.
4. RecoveryWQ: handles recovery tasks. Runs in the recovery_tp threadpool.
5. SnapTrimWQ: handles snap trimming Runs in the disk_tp threadpool. See SnapTrimmer
6. ScrubWQ: handles primary scrub path Runs in the disk_tp threadpool. See Scrub
7. ScrubFinalizeWQ: handles primary scrub finalize Runs in the disk_tp threadpool. See Scrub
8. RepScrubWQ: handles replica scrub path Runs in the disk_tp threadpool See Scrub
9. RemoveWQ: Asynchronously removes old pg directories Runs in the disk_tp threadpool See PGRemoval

ThreadPool

See src/common/WorkQueue.h See also above.

There are 4 OSD threadpools:

1. op_tp: handles ops and subops
2. recovery_tp: handles recovery tasks
3. disk_tp: handles disk intensive tasks
4. command_tp: handles commands

OSDMap

See src/osd/OSDMap.h

The crush algorithm takes two inputs: a picture of the cluster with status information about which nodes are up/down and in/out, and the pgid to place. The former is encapsulated by the OSDMap. Maps are numbered by *epoch* (epoch_t). These maps are passed around within the OSD as std::tr1::shared_ptr<const OSDMap>.

See MapHandling

PG

See src/osd/PG.* src/osd/PrimaryLogPG.*

Objects in rados are hashed into *PGs* and *PGs* are placed via crush onto OSDs. The PG structure is responsible for handling requests pertaining to a particular *PG* as well as for maintaining relevant metadata and controlling recovery.

OSDService

See src/osd/OSD.cc OSDService

The OSDService acts as a broker between PG threads and OSD state which allows PGs to perform actions using OSD services such as workqueues and messengers. This is still a work in progress. Future cleanups will focus on moving such state entirely from the OSD into the OSDService.

OVERVIEW

See src/ceph_osd.cc

The OSD process represents one leaf device in the crush hierarchy. There might be one OSD process per physical machine, or more than one if, for example, the user configures one OSD instance per disk.