OSD THROTTLES

There are three significant throttles in the filestore: wbthrottle, op_queue_throttle, and a throttle based on journal usage.

WBTHROTTLE

The WBThrottle is defined in src/os/filestore/WBThrottle.[h,cc] and included in FileStore as FileStore::wbthrottle. The intention is to bound the amount of outstanding IO we need to do to flush the journal. At the same time, we don't want to necessarily do it inline in case we might be able to combine several IOs on the same object close together in time. Thus, in FileStore::_write, we queue the fd for asyncronous flushing and block in FileStore::_do_op if we have exceeded any hard limits until the background flusher catches up.

The relevant config options are filestore_wbthrottle*. There are different defaults for xfs and btrfs. Each set has hard and soft limits on bytes (total dirty bytes), ios (total dirty ios), and inodes (total dirty fds). The WBThrottle will begin flushing when any of these hits the soft limit and will block in throttle() while any has exceeded the hard limit.

Tighter soft limits will cause writeback to happen more quickly, but may cause the OSD to miss oportunities for write coalescing. Tighter hard limits may cause a reduction in latency variance by reducing time spent flushing the journal, but may reduce writeback parallelism.

OP_QUEUE_THROTTLE

The op queue throttle is intended to bound the amount of queued but uncompleted work in the filestore by delaying threads calling queue_transactions more and more based on how many ops and bytes are currently queued. The throttle is taken in queue_transactions and released when the op is applied to the filesystem. This period includes time spent in the journal queue, time spent writing to the journal, time spent in the actual op queue, time spent waiting for the wbthrottle to open up (thus, the wbthrottle can push back indirectly on the queue_transactions caller), and time spent actually applying the op to the filesystem. A BackoffThrottle is used to gradually delay the queueing thread after each throttle becomes more than filestore_queue_low_threshhold full (a ratio of filestore_queue_max_(bytes|ops)). The throttles will block once the max value is reached (filestore_queue_max_(bytes|ops)).

The significant config options are: filestore_queue_low_threshhold filestore_queue_high_threshhold filestore_expected_throughput_ops filestore_expected_throughput_bytes filestore_queue_high_delay_multiple filestore_queue_max_delay_multiple

While each throttle is at less than low_threshhold of the max, no delay happens. Between low and high, the throttle will inject a per-op delay (per op or byte) ramping from 0 at low to high_delay_multiple/expected_throughput at high. From high to 1, the delay will ramp from high_delay_multiple/expected_throughput to max_delay_multiple/expected_throughput.

filestore_queue_high_delay_multiple and filestore_queue_max_delay_multiple probably do not need to be changed.

Setting these properly should help to smooth out op latencies by mostly avoiding the hard limit.

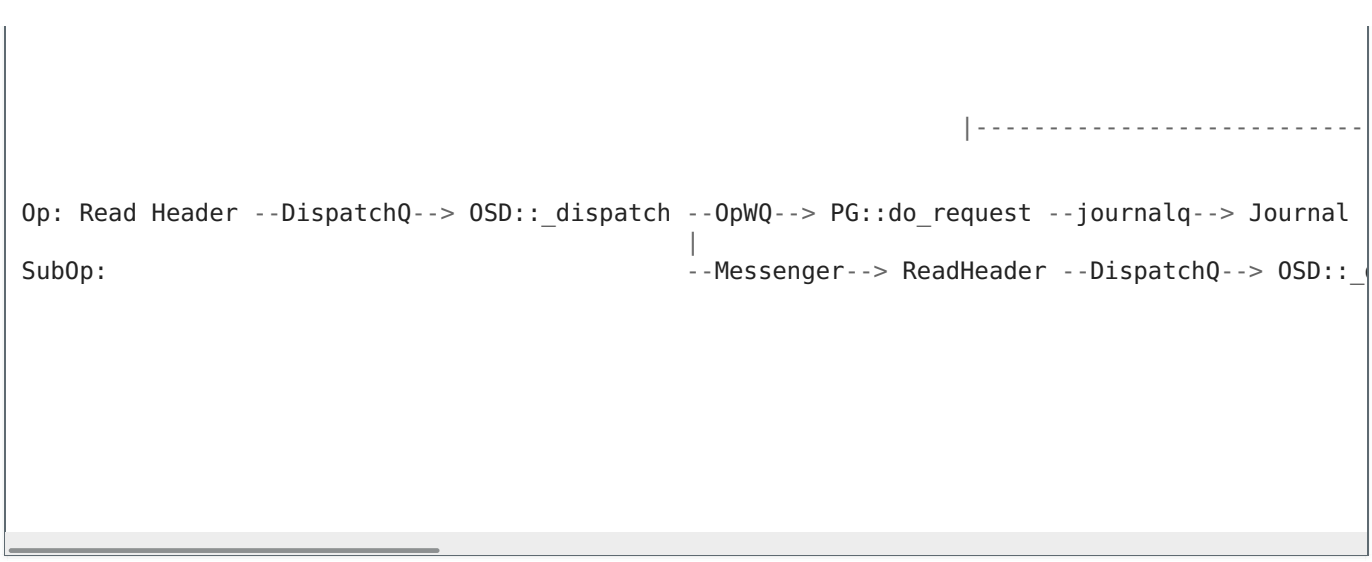
See FileStore::throttle_ops and FileSTore::thottle_bytes.

JOURNAL USAGE THROTTLE

See src/os/filestore/JournalThrottle.h/cc

The intention of the journal usage throttle is to gradually slow down queue_transactions callers as the journal fills up in order to smooth out hiccup during filestore syncs. JournalThrottle wraps a BackoffThrottle and tracks journaled but not flushed journal entries so that the throttle can be released when the journal is flushed. The configs work very similarly to the op_queue_throttle.

The significant config options are: journal_throttle_low_threshhold journal_throttle_high_threshhold filestore_expected_throughput_ops filestore_expected_throughput_bytes journal_throttle_high_multiple journal_throttle_max_multiple


|----------
Op: Read Header --DispatchQ--> OSD::_dispatch --OpWQ--> PG::do_request --journalq--> Journal
SubOp:
--Messenger--> ReadHeader --DispatchQ--> OSD::_