redhat.

INTRODUCTION BLOCK MQ IO SCHEDULER

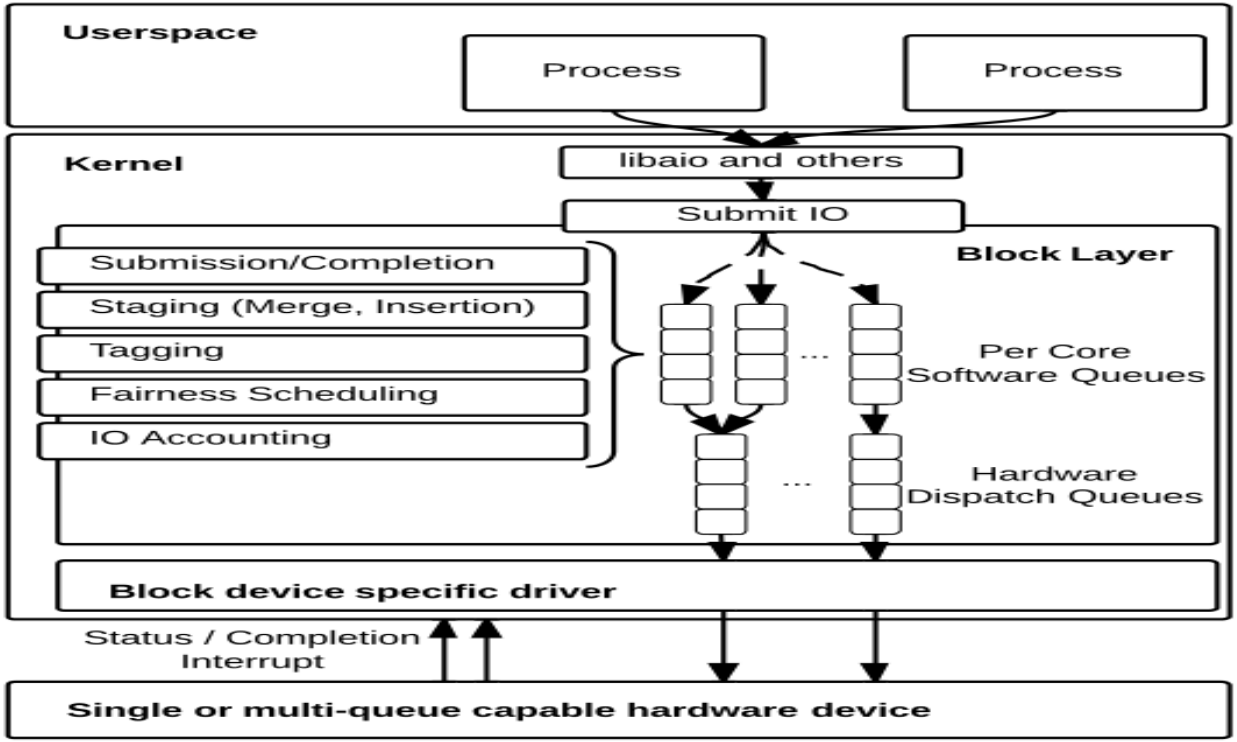
Ming Lei <ming.lei@redhat.com>, Red Hat

Oct. 22, 2017, Beijing, CLF2017

Overview

- BLOCK MQ Background
  - Introduced in V3.13
  - for better supporting new storage of NVMe
  - address scalability issue of q→queue_lock
  - initially without any IO scheduler

BLK MQ Framework

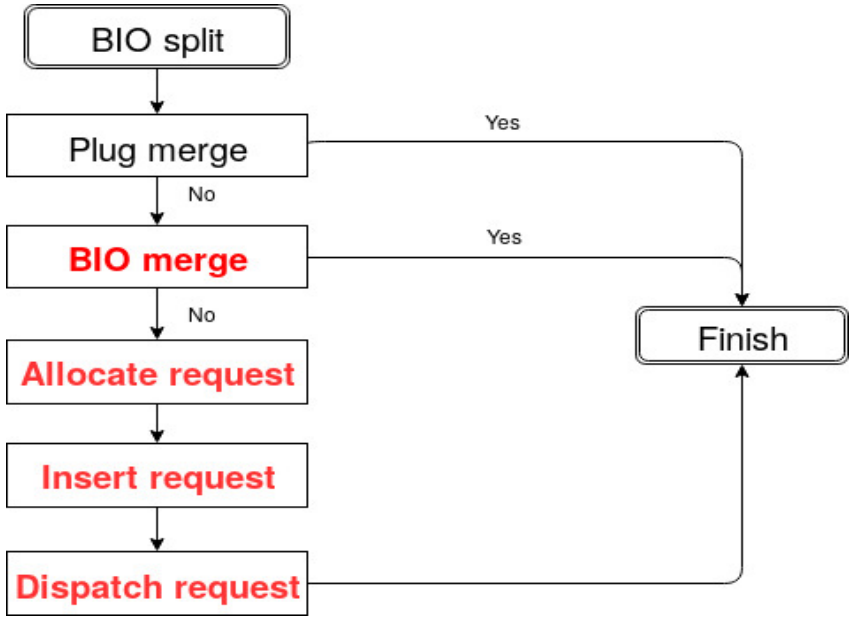


BLOCK MQ IO SCHEDULER

- BLOCK MQ IO Scheduler background
- Better support traditional Storage device
- Replace blk_queue_bio() path totally in future
- Available in V4.11, initially with mq-deadline only
- BFQ and Kyber is merged in V4.12

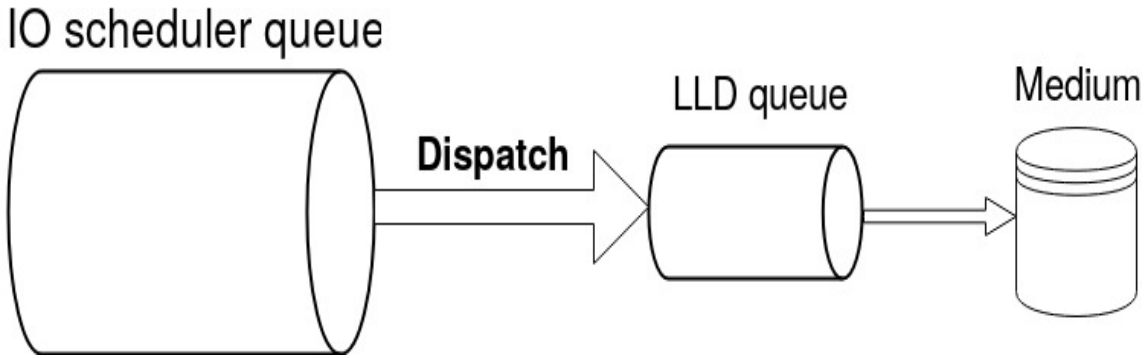
MQ IO Scheduler Framework

- BLOCK MQ IO Path(blk_mq_make_request)



MQ IO Merge Model

- IO scheduler queue Vs. LLD queue



IO Merge is possible IFF IO scheduler queue depth > LLD queue’s

MQ IO Merge Model

- IO scheduler queue depth
  - controlled via /sys/block/XXX/queue/nr_requests
  - respected via allocating request
- LLD queue depth
  - driver/device specific way to control, or not controllable
  - .queue_rq() returns BLK_STS_RESOURCE when LLD queue is full
  - LLD queue depth is highly related with storage device performance

MQ IO Merge Model

- None Scheduler

  - introduced for NVMe at the beginning

  - no scheduler queue, so IO merge is possible IFF driver has specific queue depth, such as q->queue_depth on SCSI, not possible on NVMe actually

  - IO merge is on percpu SW queue, and use simple policy, merge isn’t efficient

MQ IO Merge Model

- MQ-DEADLINE / BFQ

  - basically similar with old block

  - introduced for making MQ working well on traditional disks

  (such as, SCSI)

  - IO merge is good because of per-request-queue scheduler queue

  - may not scale well for high performance MQ devices, such as NVMe, SCSI FC/SRP, because of per-request-queue lock

MQ IO Merge Model

- Kyber

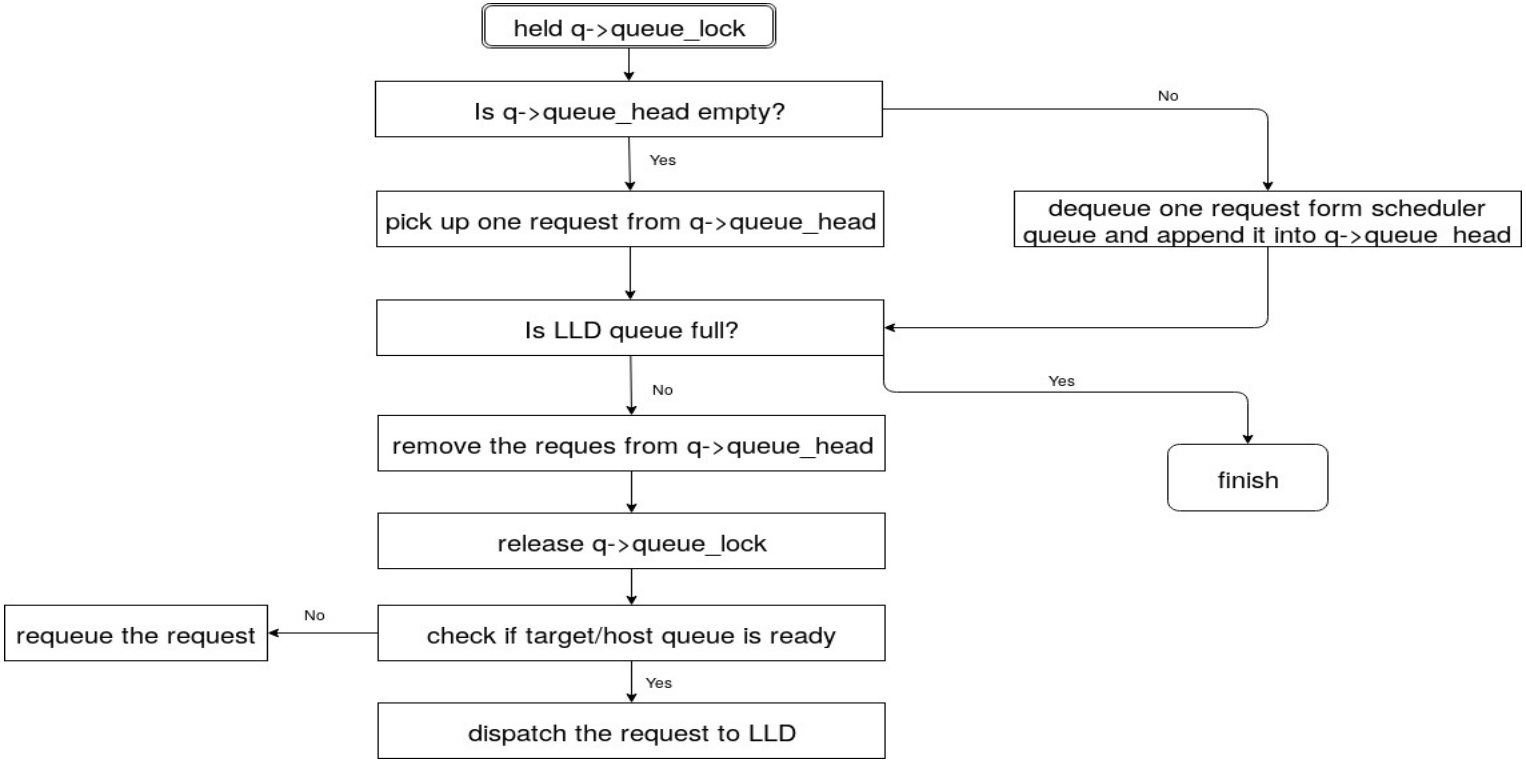
  - introduced for high performance devices, such as NVMe, NVMe OF

  - introduce READ, SYNC_WRITE, OTHER domains, and each domain has its queue depth for simulating LLD queue depth,

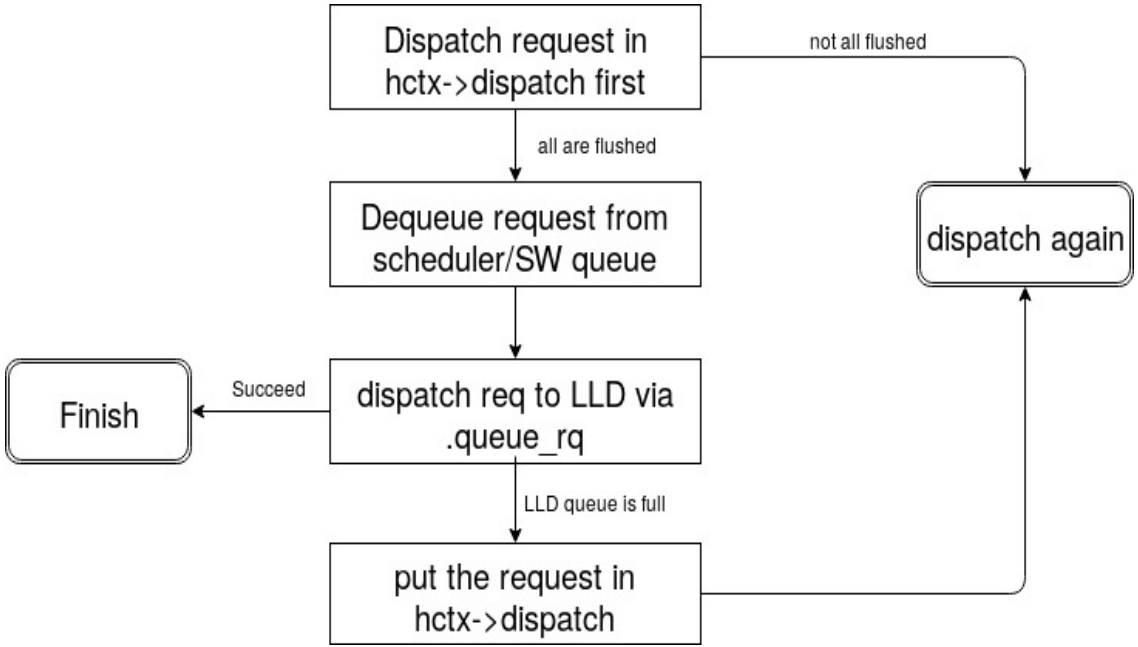
  - IO merge is possible because of domain queue

  - IO merge is on percpu SW queue

Block legacy IO Dispatch Model



MQ IO Dispatch Model



MQ IO Dispatch Model

- Issues

  - hctx->dispatch can’t be dispatched one by one without holding hctx→lock; between moving hctx→disptach moved to one temp list and being flushed out, scheduler can’t be dequeued

  - q->queue_depth is often among the whole request queue, all hctx should respect this limit

MQ IO Dispatch Model

- Solutions for these issues

  - bypass hctx->dispatch totally

  - reserving budget before before dequeuing from IO scheduler queue by introducing .get_budget and .put_budget in blk_mq_ops

  - will be merged to V4.15 if everything is fine

  - better than legacy path in theory without holding per-queue lock

Performance data

- mq-deadline(fio, libaio, direct, bs=4k, queue_depth=64, jobs=64, disk=SRP/IB, V4.14-rc4)

IOPS(K)	V4.14-rc4 DEADLINE	V4.14-rc4 MQ-DEADLINE	patched V4.14-rc4 MQ-DEADLINE
read	450.0	154.12	474.0
write	419.65	135.88	481.89

Next Step of MQ IO scheduler

- Improving on Kyber
  - pre-defined/hard coded domain depth
  - hard coded latency
  - domain queue depth adjust approach
  - very young
- SSD friendly IO schedule

Next Step of MQ IO scheduler

- One big challenge

  - need to provide excellent support on modern high performance storage, such as NVMe, NVMe OF

  - meantime not cause performance regression on traditional storage, such as SCSI

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos