CANONICAL

Virtio-blk Multi-queue Conversion and QEMU Optimization

Ming Lei, Canonical Ltd.
KVM Forum 2014

Virtio-blk Linux driver evolution

- Traditional request based(V2.6.24 ~ V3.12)

  - support I/O scheduler for merging I/O

  - single coarse lock for protecting request_queue

- BIO based(V3.7 ~ V3.12)[1]

  - no coarse request queue lock in I/O path

  - don't support I/O scheduler/merge, and not help slow device

  - generate more requests to host, and more Vmexit

- Block multi-queue : single dispatch queue(V3.13 ~ V3.16)

  - request based, but without coarse lock for protecting request queue

  - still support I/O merge in software staging queue

  - single virtqueue means single vq lock is needed in both submit and complete path

- **Block multi-queue : multi dispatch queue(V3.17 ~)**

  - with all advantage of block multi-queue

  - use multi virtqueue as dispatch queues for improving scalability and throughput

Linux Block multi-queue[2]

- Introduced for support high IOPS SSD storage
- Remove coarse request queue lock
- Two level queues

  - per-cpu software queue

    staging queue

    schedule / IO account / merge / tagging

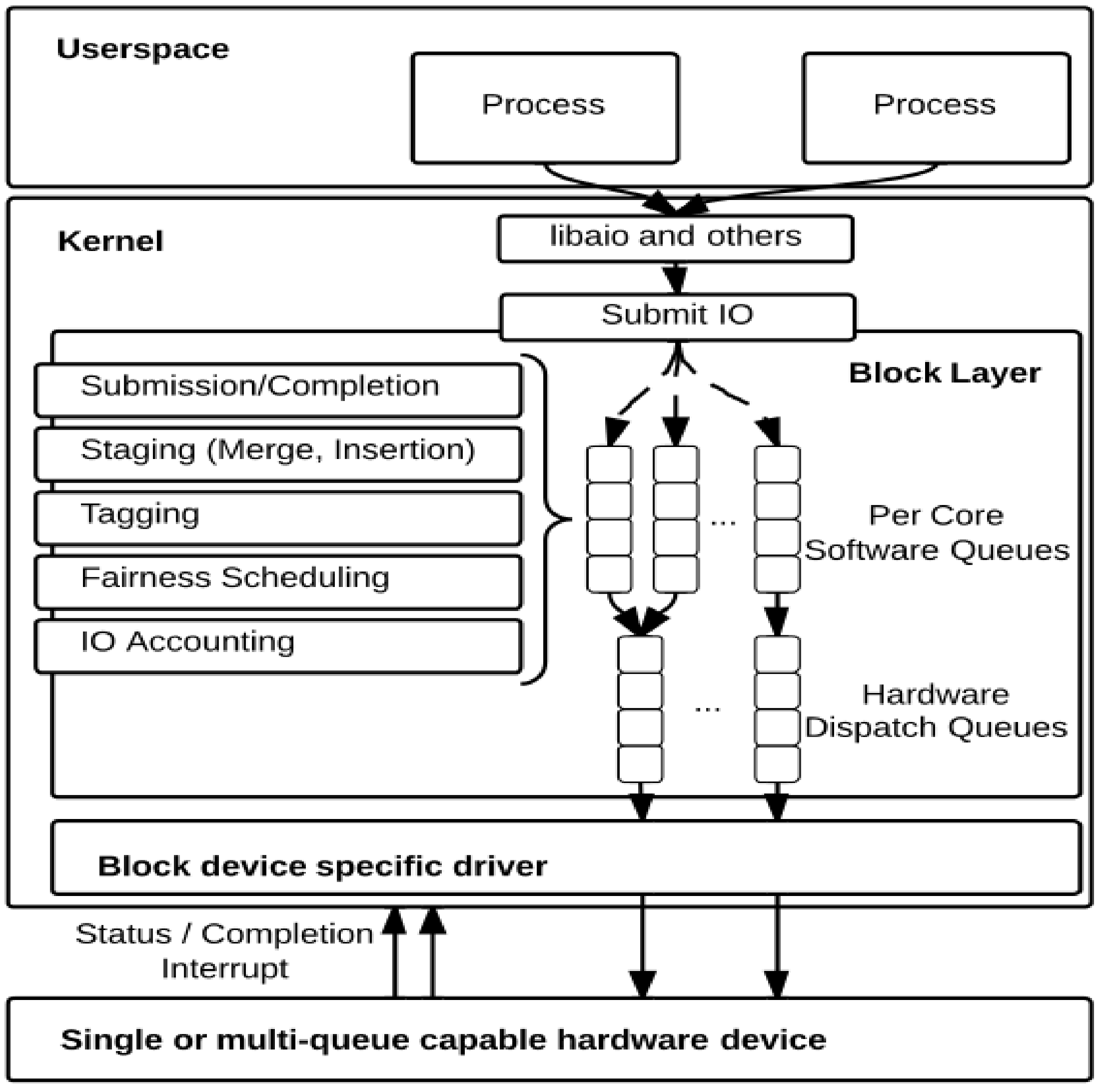
  - dispatch queue(hardware queue)

    submit request to hardware

    need hardware support(each queue has its irq, hw queue,...)

  - N:M mapping between SW queue and HW queue
- Pre-allocated request and driver specific data
- Merged to v3.13

Linux Block multi-queue[2]



Ming Lei

CANONICAL

Linux Block multi-queue

- merged blk-mq drivers up to v3.17-rc5

  Null_blk

  Virtio_blk

  SCSI middle layer(need to enable mq via module parameter)

  mtip32xx

- In review

  Loop

  NVME

  Xen, blkfront

Virtio-blk: Linux multi-queue support

- Linux v3.13

  - single hardware / dispatch queue

  - improve throughput on quick devices

  - don't need hypervisor(QEMU) change

  - single virtqueue lock is required in both I/O submit and complete path

    scalability can't be good

    single virtqueue may not be enough for very quick devices

    (such as, QEMU's I/O thread may be in starvation state)

- Linux v3.17-rc1

  - multi dispatch/hardware queue patches merged

  (it is natural to map blk-mq's dispatch queue to virtqueue)

  - no bottleneck from VM any more

  - need hypervisor(QEMU)'s support for the feature:

  ```
  +#define VIRTIO_BLK_F_MQ		12	/* support more than one vq */
  ```

QEMU: optimization

- QEMU 2.0
  - simple multi virtqueue conversion gets very good throughput
    https://lkml.org/lkml/2014/6/25/909
    https://lkml.org/lkml/2014/6/26/125
- After commit 580b6b2aa2(dataplane: use the QEMU block layer for I/O)
  - throughput becomes not good even with multi virtqueue conversion
  - then I started the investigation
- investigation from QEMU virtio-blk dataplane I/O model
  - single I/O thread, compute bound type for high IOPS device
  - CPU usage per request
  - linux AIO

QEMU: optimization

- Benching environment

  - VM: Linux kernel 3.17-rc5 / quad core / 8GB RAM

  - virtio-blk(dataplane) device backend: null_blk

    – null_blk feature is very similar with SSD

    – not depend on specific block device

  - QEMU 2.2-dev master: patches against commit 30eaca3acdf17

  - host:

    Ubuntu trusty

    Intel Xeon 2.13GHz(2 sockets, 8 physical cores(16 threads)) / 24G RAM

  - FIO:

    numjobs = 2 for single virtqueue, and numjobs = 4 for dual virt queues

    ioengine=libaio

    direct=1

    iodepth=64

    group_reporting=1

QEMU: I/O batch submission

- What is I/O batch submission

  - handle more requests in one single system call(io_submit), so calling number of the syscall of io_submit can be decrease a lot

  - be helpful for kernel to merge / batch process since per-task IO plug is held during handling all I/O from 'iocbs'

- Linux-AIO interface[3]

  int io_submit(io_context_t ctx, long nr, struct iocb *iocbs[]);

  The io_submit function can be used to enqueue an arbitrary number of read and write requests at one time. The requests can all be meant for the same file, all for different files or every solution in between.

- Used in virtio-blk dataplane from beginning

  - looks no one noticed its power, and just thought it is a natural way to do that

  - LKVM doesn't take it, and big difference was observed when I compared these two

- removed in commit 580b6b2aa2(dataplane: use the QEMU block layer for I/O)

QEMU: I/O batch submission

- abstracting with generic interfaces:

  - bdrv_io_plug() / bdrv_io_unplug()

  - merged in dd67c1d7e75151(dataplane: submit I/O as a batch)

- Wider usage

  - can be used for other block devices in case of 'aio=native', either dataplane or not

  - support to submit I/O from more than one files(typical use case: multi-lun SCSI)

- Performance improvement

  - single queue, with bypass coroutine optimization enabled

bench	Without I/O batch	With I/O batch	improvement
randread	111K	171K	54%
randwrite	115K	162K	40%
read	109K	172K	57%
write	152K	157K	3%

Ming Lei

QEMU: I/O batch submission

- Randread benchmark(use simple trace)

	Without I/O batch	With I/O batch	improvement
Average request number per I/O submission	1	26.58	
Average request number handled in one time(process_notify)	50.92	27.21	
Average time for submitting one request by QEMU	6.539us	4.986us	23%

- Write benchmark(use simple trace)

  - multi-write makes the difference compared with other three bench

	Without I/O batch	With I/O batch	improvement
Average request number Per io submission	1	3.52	
Average merged request number by QEMU block multi-write	14.19	3.50	
Average request number Handled in one time(process_notify)	52.84	14.79	
Average time for submitting one request by QEMU	4.338us	3.97us	-8%

QEMU: bypass coroutine

- Coroutine isn't cheap for high IOPS device

  - coroutine direct cost(not mention dcache miss caused by switching stack)

  $./tests/test-coroutine -m perf --debug-log

  /perf/cost: {*LOG(message):{Run operation 40000000 iterations 12.965847 s, 3085K operations/s, **324ns per coroutine**}:LOG*}

  - example

  100K IOPS: throughput may decrease 3.2%

  300K IOPS: throughput may decrease 9.6%

  500K IOPS: throughput may decrease 16%

- bypass coroutine for linux-aio

  - it is reasonable since QEMU 2.0 didn't use that virtio-blk dataplane

  - linux-aio with O_DIRECT won't block most of times, so it is OK for dataplane

QEMU: bypass coroutine

- Throughput improvement with bypassing coroutine

  - single queue, with I/O batch submission enabled

bench	Without bypass coroutine	With bypass coroutine	improvement
randread	114K	171K	50%
randwrite	111K	162K	45%
read	108K	172K	59%
write	175K	157K	-10%

- Why does sequential write become slower?

  - QEMU block's multi write: more requests merged if it becomes a bit slower?

	Without bypass coroutine	With bypass coroutine	improvement
Average merged request number	4.039	3.50	
Average request number handled in one time(process_notify)	20.17	14.79	
Average time for submitting one request by QEMU	3.759us	3.97us	-5%

QEMU: perf stat on bypass coroutine

- **without bypass cocoutine**

  - single-vq, IOPS 96K, 155.019584628 seconds time elapsed

  65,449,606,848 L1-dcache-loads [39.77%]

  2,087,494,534 L1-dcache-load-misses # 3.19% of all L1-dcache hits [39.72%]

  231,736,388,638 cpu-cycles [39.95%]

  222,828,102,544 instructions # 0.96 insns per cycle [49.80%]

  44,117,817,799 branch-instructions [49.93%]

  716,777,979 branch-misses # 1.62% of all branches [49.99%]

- **with bypass coroutine**

  - single-vq, IOPS 147K, 153.717004314 seconds time elapsed

  80,608,083,902 L1-dcache-loads [40.12%]

  1,955,370,293 L1-dcache-load-misses # 2.43% of all L1-dcache hits [39.96%]

  292,247,715,774 cpu-cycles [40.01%]

  276,707,625,913 instructions # 0.95 insns per cycle [50.06%]

  53,657,048,721 branch-instructions [49.92%]

  681,296,161 branch-misses # 1.27% of all branches [49.92%]

QEMU: 'perf stat' on bypass coroutine

- **without bypass coroutine**
  - **- quad-vqs, IOPS 130K, 152.461739616 seconds time elapsed**

```
  84,530,958,503   L1-dcache-loads                                        [40.26%]
   2,654,266,200   L1-dcache-load-misses   #  3.14% of all L1-dcache hits [40.29%]
 290,572,301,418   cpu-cycles          [40.19%]
 290,424,820,982   instructions          #  1.00  insns per cycle     [50.13%]
  58,099,370,492   branch-instructions                                    [50.05%]
     924,204,540   branch-misses          #  1.59% of all branches     [49.94%]
```

- **with bypass coroutine**
  - **- quad-vqs, IOPS 173K, 152.004454306 seconds time elapsed**

```
  87,074,630,884   L1-dcache-loads                                        [40.08%]
   2,034,388,767   L1-dcache-load-misses   #  2.34% of all L1-dcache hits [40.13%]
 280,337,907,649   cpu-cycles          [39.98%]
 301,037,129,202   instructions          #  1.07  insns per cycle     [49.91%]
  58,909,482,717   branch-instructions                                    [49.93%]
     682,183,716   branch-misses          #  1.16% of all branches     [49.98%]
```

QEMU: multi virtqueue support

- Which cases are suitable for enabling multi virtqueue
  - lots of concurrent I/O requirement from application
  - can't get satisfied throughput with single virtqueue, for high IOPS devices
- How to use multi virtqueue
  - 'num_queues' parameter
  - support both dataplane and non-dataplane
- Handle all requests from multi virtqueues in one I/O thread
  - may increase request count per system call
  - try to make I/O thread at full loading
- In the future, more I/O threads may be introduced for handling requests from multi virtqueues

QEMU: multi virtqueue support

- scalability improvement

  - single virtqueue: num_queues = 1

bench	numjobs=2	numjobs=4	improvement
randread	171K	118K	-30%
randwrite	162K	114K	-29%
read	172K	120K	-30%
write	157K	158K	+0.6%

  - dual virtqueue: num_queues = 2

bench	numjobs=2	numjobs=4	improvement
randread	168K	174K	+3%
randwrite	157K	163K	+3%
read	162K	174K	+7%
write	161K	260K	+61%

QEMU: multi virtqueue support

- Throughput improvement

  - no obvious improvement for non-write bench

  - null blk throughput(randread, single job) on host is ~250K

  - CPU utilization of IO thread is close to 100% for non-write bench

  - write throughput increased a lot because IO thread takes fewer CPU to submit I/O when multi-write merged lots of sequential writes

- Multi virtqueue can help two situations

  - I/O handling is too slow, so requests can be exhausted easily from single queue, then I/O thread may become blocked, for example, multi queue can improve throughput ~30% if bypass coroutine is disabled

  - I/O handling is too quick, requests from VM(single vq) can't catch up with QEMU's I/O processing, and multi queue can help this case too, for example, QEMU 2.0 can get much improvement with multi virtqueue

QEMU: virtio-blk multi queue status

- Git tree:

  - git://kernel.ubuntu.com/ming/qemu.git v2.1.0-mq.4

- Linux aio fix patches

  - pending

- Fast path patch for bypassing coroutine from Paolo Bonzini

  - not ready for merge

- Multi virtqueue conversion

  - pending

  - depends on linux-aio fix patches

QEMU: related block optimization

- Next step work
  - Push these patches
  - Apply bdrv_io_plug() / bdrv_io_unplug() to other block devices
  - Support to submit I/O in batch from multiple files for SCSI devices
  - Virtio-SCSI block mq support
    - virtio-scsi dataplane
    - linux virtio-scsi support multi dispatch queue

References:

[1] Asias He, Virtio-blk Performance Improvement, KVM forum 2012

[2] Matias Bjorling, Jens Axboe, David Nellans, Philippe Bonnet, Linux Block I/O: Introducing Multi-queue SSD Access on Multi-core Systems

[3] man 3 io_submit

Questions please

Thank you