SDC 19

September 23-26, 2019
Santa Clara, CA

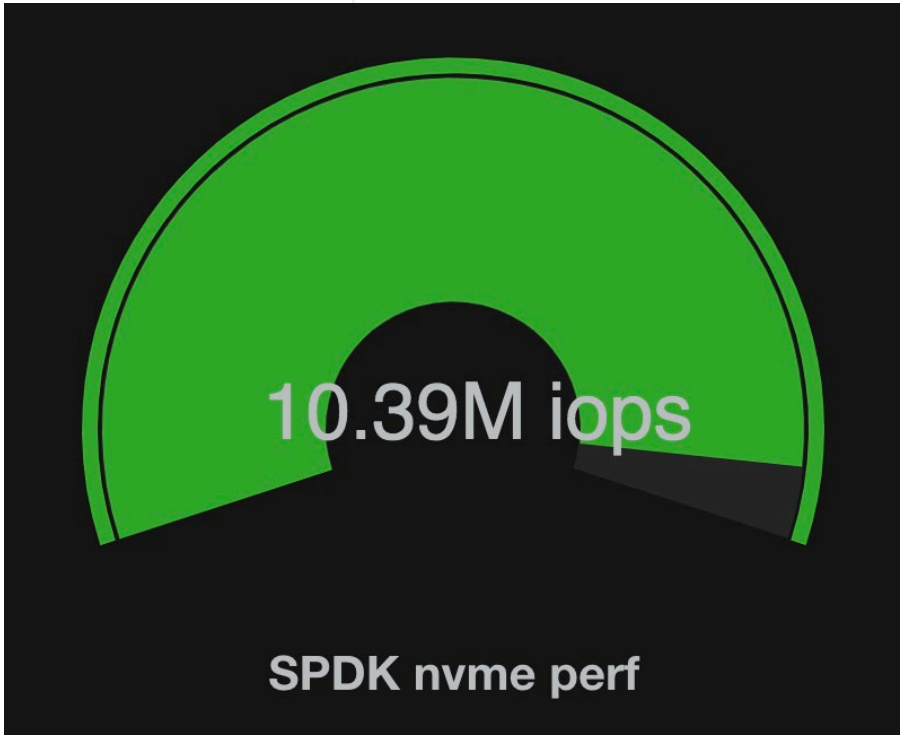
10 Million I/Ops From a Single Thread

Ben Walker
Intel

Storage Performance Development Kit

- User-space block storage stack with similar features to an OS
  - Includes an NVMe driver
- Open Source, BSD 3-Clause License (https://spdk.io)
- Very active community
  - 1200 commits from 56 committers in last 3 months

BIG NUMBERS



4KiB random reads at queue depth 128 to each device

3

System Configuration

Configuration	
CPU	Intel® Xeon® Platinum 8280L CPU @ 2.70GHz
Memory	12x 16GB 26667
Storage	21x Intel® SSD DC P4600 1.6TB

4

Setting the Stage

- NVMe drives are getting really, really fast
  - No, like really.
- PCIe Gen 4 and eventually Gen 5 are huge bandwidth boosts.
- We should expect storage servers doing tens of millions of I/Ops to be the *norm* going forward.

5

Overview

Talk based on this blog post:
https://spdk.io/news/2019/05/06/nvme/

Covering only 3 techniques from that post today for time reasons.

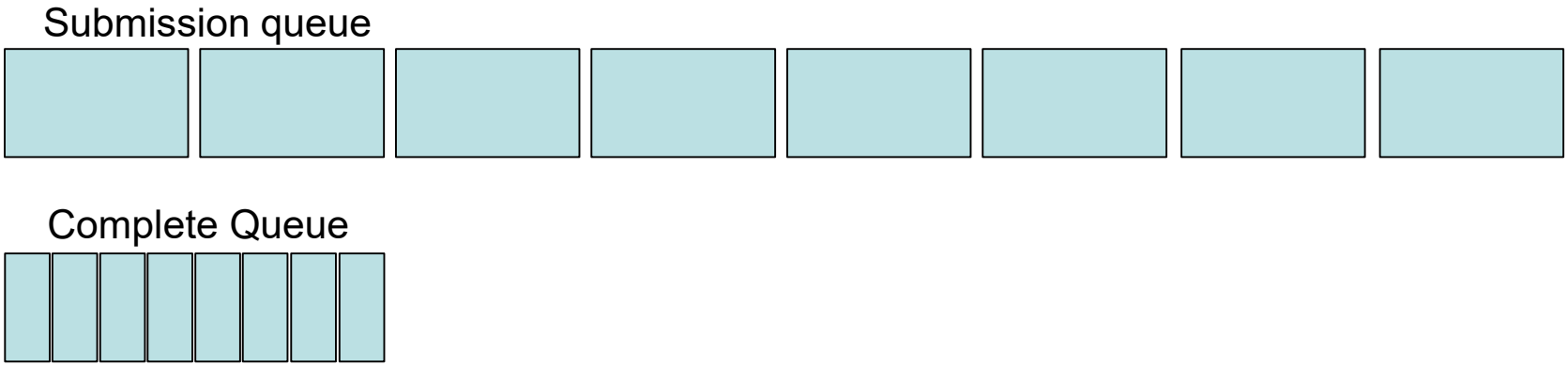
Not covering any active areas of research, but there are several!

6

Background: NVMe I/O Queues

Mechanics of Submitting an I/O

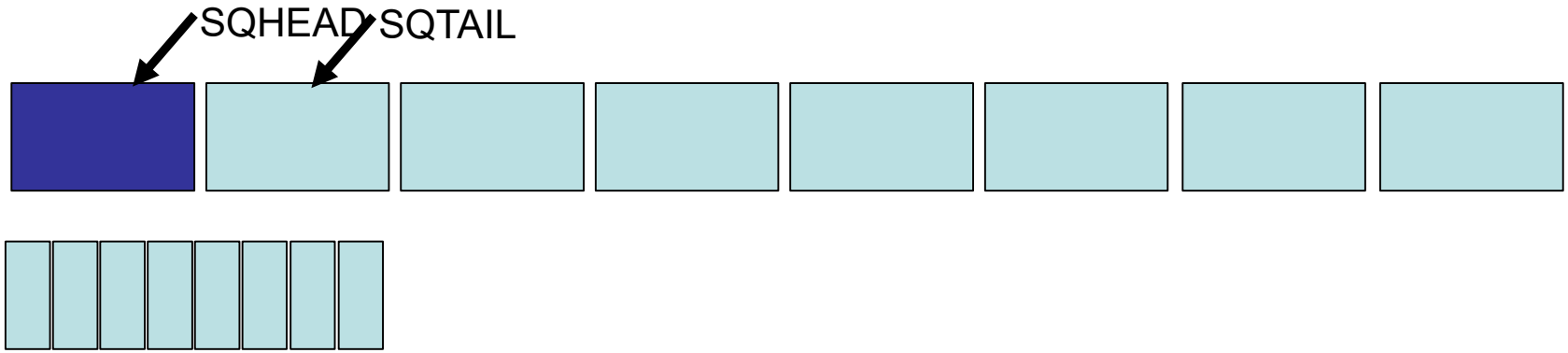
NVMe queues consist of two arrays in host memory (submission queue and completion queue) plus two doorbells (SQTAIL, CQHEAD) in the BAR



8

Mechanics of Submitting an I/O

To submit: Copy command into next slot. Write SQTAIL.



To complete: Compare phase bit. If flipped, contains valid completion. When done, write CQHEAD.
Completions hold updates to SQHEAD.

9

SPDK NVMe Design Basics

- **Assign NVMe queue pairs to threads.**
  - No locks!
- **Disable interrupts**
  - Completions are handled when the application is ready to handle them. No context switch.
- Code is compiled with –O2, LTO enabled, PGO disabled.

10

Keys To Performance

Don’t let the CPU stall!

- No cross-thread coordination (locks, etc.)
- Poll instead of interrupt
- Minimize MMIO
- Get the right things into the CPU cache at the right time
- Pack structures. Separate hot data from cold.

11

Strategy 1: Minimize MMIO

Tricks For Minimizing MMIO

```
while (true) {
    ... work ...
        spdk_nvme_ns_cmd_read(..., cb_fn);
        spdk_nvme_ns_cmd_read(..., cb_fn2);
    ... work ...

    spdk_nvme_qpair_process_completions(...); /* calls cb_fns if the
                                                 read is done */
}
```

Program Flow: Submit several commands. Check for completions. Repeat.

Naïve implementation: For each command, 1 MMIO on submit, 1 MMIO on complete

13

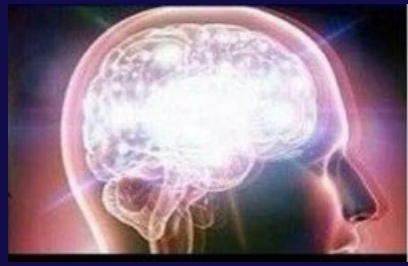
Trick 1

- When checking for completions by reading the phase bit, don’t write the completion queue doorbell until all currently outstanding completions have been discovered.

- Everyone knows this trick.

Trick 2

- When a completion is posted, don't write the completion queue doorbell unless the device actually needs more slots free.
- Devices often have large queues – say 1024. We can write the completion queue doorbell every ~512 commands.
- I've only seen SPDK do this. (Doesn't help this benchmark because trick 1 is already finding 30 to 50 completions at a time)



Trick 3

- When a command is submitted, copy the command into the SQE slot but don't ring the submission queue doorbell. Instead, ring the doorbell only when the user polls (which is happening very frequently).

- This is a tricky way to batch command submissions transparently to the user in a polling system.

2.89M I/Ops with this disabled. 10.39M with this enabled.

Strategy 2: Optimize Completion Handling

Completing I/O

```
for each cqe {
    if (cqe->phase != phase_flipped) {
        break;
    }

    struct tracker *tr = tracker_array[cqe->cid];

    struct request *req = tr->req;

    req->cb_fn(req->cb_arg);
}
```

- “Tracker” is the context for a command. Array of trackers pre-allocated.
- CQE points (via CID) at a tracker (1:1)
- Tracker points at a request (1 request to N trackers)

18

Eliminate Data Dependent Loads

```
struct nvme_request {
        spdk_nvme_cmd_cb cb_fn; /* Callback function */
        void *cb_arg;
};

struct nvme_tracker {
        struct nvme_request *req;

        spdk_nvme_cmd_cb cb_fn; /* Copied callback function */
        void *cb_arg;

        /* Other stuff */
};
```

On submission, copy cb_fn and cb_arg into tracker

19

Eliminating Data Dependent Loads

```
for each cqe {
    if (cqe->phase != phase_flipped) {
        break;
    }

    struct tracker *tr = tracker_array[cqe->cid];

    tr->cb_fn(tr->cb_arg); /* Can call cb_fn immediately! */
}
```

500K I/O per second improvement

20

Strategy 3: Pre-fetch

Clever Pre-fetching

```
for each cqe {
    if (cqe->phase != phase_flipped) {
        break;
    }

    next_cqe = cqe + 1;
    if (next_cqe->phase == phase_flipped) {
        __builtin_prefetch(tracker_array[next_cqe->cid]);
    }

    __builtin_prefetch(next_cqe + 1);

    struct tracker *tr = tracker_array[cqe->cid];

    struct request *req = tr->req;

    req->cb_fn(req->cb_arg);
}
```


22

Questions?

- https://spdk.io
- https://spdk.io/news/2019/05/06/nvme/
- https://spdk.io/doc/nvme_spec.html
- https://spdk.io/doc/ssd_internals.html

23

Thank You