SPDK NVMe
In-depth Look at its Architecture and Design

Jim Harris
Intel
What is SPDK?

- Storage Performance Development Kit
- Open Source, BSD Licensed
- [http://spdk.io](http://spdk.io)
- User-space Drivers and Libraries for Storage, Storage Networking and Storage Virtualization
  - This talk focused specifically on the SPDK NVMe userspace driver
Focus for today’s talk
SPDK and Kernel

- Better performance and efficiency compared to traditional interrupt-driven approaches
- BUT...
- SPDK is not a general-purpose solution
  - covers some use cases very well – others not at all (or at least not well)
- Polled mode design and userspace implementation drove much of the SPDK design
SPDK Paradigm

- SSD dedicated to single process
  - NVMe SR-IOV can alleviate this restriction
- Smallish, known number of threads
- Pre-allocated pinned memory
Environment Abstraction

- Provides services for:
  - enumerating PCIe devices
  - polled mode thread creation
  - allocating/pinning memory
    - including VA to IOVA translation
- Default implementation is based on DPDK
- Enables alternative implementations

```
NVMe Driver
    ↓
Environment API (include/spdk/env.h)
    ↓
DPDK Environment (lib/env_dpdk)
    ↓
Alternate Environment
```
Transport Abstraction

- Enables separate implementations for different transports
  - construct/destruct controller
  - set/get register value
  - create/delete I/O queue pair
  - submit request
  - process completions

```
NVMe API
↓
NVMe Transport API
↓
NVMe PCIe Transport
↓
NVMe TCP Transport (Future)

↓
NVMe RDMA Transport
↓
NVMe FC Transport (Future?)
```
NVMe Probe and Attach

```c
int spdk_nvme_probe(const struct spdk_nvme_transport_id *trid,
                     void *cb_ctx,
                     spdk_nvme_probe_cb probe_cb,
                     spdk_nvme_attach_cb attach_cb,
                     spdk_nvme_remove_cb remove_cb);
```

- Controller identification by transport ID
  - Transport ID used to identify local (PCIe) and remote controllers
  - NULL means “all PCIe”
NVMe Probe and Attach (continued)

- `probe_fn`
  - decide whether to attach
    - based on PCIe ID when `trid == NULL`
    - based on NQN when `trid = discovery controller identification by transport ID`
  - specify options (struct `spdk_nvme_ctrlr_opts`)
    - includes: number of I/O queues, use CMB (if available), arbitration mechanism, KATO, host ID
NVMe Probe and Attach (continued)

- `attach_fn`
  - notification controller is ready for use
  - "negotiated" controller options
- Synchronous but will attach multiple controllers in parallel (asynchronous version in next release)
- `spdk_nvme_connect()`
  - `spdk_nvme_probe()` but for specific controller
Queue Pair Creation

- `struct spdk_nvme_io_qpair_opts`
  - Priority (for WRR)
  - I/O queue size, # I/O requests (discussed later)

- Submit admin commands
  - CREATE_CQ, CREATE_SQ

- Allocate memory for requests, trackers

- Synchronous API
Command Submission

```c
int spdk_nvme_ns_cmd_read(struct spdk_nvme_ns *ns, struct spdk_nvme_qpair *qpair,
    void *payload, uint64_t lba, uint32_t lba_count,
    spdk_nvme_cmd_cb cb_fn, void *cb_arg, uint32_t io_flags);
```

- *All* command submission APIs are async
  - `cb_fn` and `cb_arg` parameters
  - completion only by polling `qpair`
- Payload buffers are virtual addresses
- Not necessarily 1:1 with NVMe commands
  - i.e. I/O splitting
Command Completions

```c
int32_t spdk_nvme_qpair_process_completions(struct spdk_nvme_qpair *qpair,
                                           uint32_t max_completions);
```

- Application responsible for:
  - polling for completions
  - submit and poll on same thread

- Completion functions called by SPDK NVMe driver from `spdk_nvme_qpair_process_completions()` context

- Use `max_completions` to limit number of completions
struct nvme_request

- Private, internal data structure
- Dedicated set of objects per qpair
  - Memory for objects allocated along with qpair
  - Count dictated by controller or qpair options
- Transport agnostic
- Can be queued
  - Depending on transport specifics
I/O Splitting

- I/O may need to be split before sent to controller
  - Max Data Transfer Size (MDTS)
  - Namespace Optimal IO Boundary (NOIOB)
- Two possible approaches
  - 1) User must split I/O before calling SPDK API
  - 2) SPDK split I/O internally
- SPDK has implemented approach #2
I/O Splitting in SPDK

- Simplify SPDK usage and avoid code duplication
- Ignoring NOIOB is functionally OK, but will result in unexpected performance issues
- Minimal (if any) impact on applications that require no I/O splitting
- struct nvme_request supports parent/child relationships
  - one allocated per child I/O
Vectored I/O

int spdk_nvme_ns_cmd_readv(struct spdk_nvme_ns *ns, struct spdk_nvme_qpair *qpair,
uint64_t lba, uint32_t lba_count,
spdk_nvme_cmd_cb cb_fn, void *cb_arg, uint32_t io_flags,
spdk_nvme_req_reset_sgl_cb reset_sgl_fn,
spdk_nvme_req_next_sge_cb next_sge_fn);

- SGEs fetched by SPDK NVMe driver using callback functions
  - avoids application translating to struct iovec
  - avoids struct iovec => NVMe PRP translation
VA-to-IOVA Translation

- SPDK maintains a userspace “page table”
  - describes the pre-allocated pinned memory
- Two-level page table
  - First level: 1GB granularity (0x0 => 256TB)
  - Second level: 2MB granularity
- Similar registration/translation scheme used for RDMA
  - Translates to MR instead of IOVA
Timeout Callbacks

```c
void spdk_nvme_ctrlr_register_timeout_callback(struct spdk_nvme_ctrlr *ctrlr,
                                          uint64_t timeout_us,
                                          spdk_nvme_timeout_cb cb_fn, void *cb_arg)
```

- Optional – can be set at controller level
- Timeouts without interrupts?
  - Expirations checked during completion polling
- Driver takes no action (i.e. abort or reset)
  - Application is notified and can take action
- Pending requests linked in submission order
Benchmarking

- Two options
  - fio plugin
  - nvme/perf
- fio per I/O overhead significant at 3-4M IOPs/core
- nvme/perf – lower overhead but fewer features
## nvme/perf histograms

- Very granular (<1us) buckets
- Enabled via –LL option to nvme/perf

**Latency histogram for INTEL SSDPE2KE020T7 from core 0:**

<table>
<thead>
<tr>
<th>Range in us</th>
<th>Cumulative</th>
<th>IO count</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.429 - 4.460</td>
<td>0.0001%</td>
<td>1</td>
</tr>
<tr>
<td>4.460 - 4.491</td>
<td>0.0008%</td>
<td>6</td>
</tr>
<tr>
<td>4.491 - 4.521</td>
<td>0.0018%</td>
<td>8</td>
</tr>
<tr>
<td>4.521 - 4.552</td>
<td>0.0052%</td>
<td>28</td>
</tr>
<tr>
<td>4.552 - 4.582</td>
<td>0.0125%</td>
<td>60</td>
</tr>
<tr>
<td>4.582 - 4.613</td>
<td>0.0247%</td>
<td>101</td>
</tr>
<tr>
<td>4.613 - 4.643</td>
<td>0.0560%</td>
<td>258</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Error Injection
Raw/Passthrough Commands
Extended LBA Formats