Multipathing PCI-Express Storage

Keith Busch
Linux Vault
12 March 2015
Agenda
(in no particular order)

• Why we care
• PCI-e Storage Standardization
• Storage stacking details
• Results and future work
PCI-e Storage Standard:

Non-Volatile Memory Express
NVMe: Who
NVMe: What

Storage standard defining:
• Host controller interface
• Queueing model
• Command set

Designed for performance and scalability in mind
NVMe: Why

CPU vs. Storage Performance Gap

RELATIVE PERFORMANCE
Log scale (not necessarily 100% accurate)

1990 2000 2010 2020

CPU Performance
HDD Performance
NVMe: Why

... but PCI-e storage predates NVMe, right?
NVMe: Why

How standards proliferate:
(See: A/C chargers, character encodings, instant messaging, etc)

Situation: There are 14 competing standards.

14?! Ridiculous! We need to develop one universal standard that covers everyone's use cases. Yeah!

[Two stick figures pointing at each other]

Soon:

Situation: There are 15 competing standards.

http://xkcd.com/927/
NVMe: When

- NVMe Technical Work Begins
- NVMe 1.0 Released March 1, 2011
  - Queueing Interface
  - NVM Command Set
  - Admin Command Set
  - End-to-end Protection (DIF/DIX)
  - Security
  - Physical Region Pages (PRPs)
- NVMe 1.1 Released October 11, 2012
  - General Scatter Gather Lists (SGLs)
  - Multi-Path I/O & Namespace Sharing
  - Reservations
  - Autonomous Power Transitions During Idle
- NVMe 1.2 Released November 12, 2014
  - Enhanced Power Management
  - Host Memory Buffer
  - Controller Memory Buffer
  - Namespace Management
  - Live F/W Update

Timeline:
- 2009
- 2010
- 2011
- 2012
- 2013
- 2014
## NVMe: How

<table>
<thead>
<tr>
<th>Offset</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>CAP</td>
<td>Controller Capabilities</td>
</tr>
<tr>
<td>08h</td>
<td>VS</td>
<td>Version</td>
</tr>
<tr>
<td>0Ch</td>
<td>INTMS</td>
<td>Interrupt Mask Set</td>
</tr>
<tr>
<td>10h</td>
<td>INTMC</td>
<td>Interrupt Mask Clear</td>
</tr>
<tr>
<td>14h</td>
<td>CC</td>
<td>Controller Configuration</td>
</tr>
<tr>
<td>18h</td>
<td>Rsvd</td>
<td>Reserved</td>
</tr>
<tr>
<td>1Ch</td>
<td>CSTS</td>
<td>Controller Status</td>
</tr>
<tr>
<td>20h</td>
<td>NSSR</td>
<td>NVM Subsystem Reset</td>
</tr>
<tr>
<td>24h</td>
<td>AQA</td>
<td>Admin Queue Attributes</td>
</tr>
<tr>
<td>28h</td>
<td>ASQ</td>
<td>Admin Submission Queue Base Address</td>
</tr>
<tr>
<td>30h</td>
<td>ACQ</td>
<td>Admin Completion Queue Base Address</td>
</tr>
<tr>
<td>38h</td>
<td>Rsvd</td>
<td>Reserved</td>
</tr>
<tr>
<td>F00h</td>
<td>Rsvd</td>
<td>Command set Specific</td>
</tr>
<tr>
<td>1000h</td>
<td>SQ 0 TDBL</td>
<td>Admin Submission Queue Tail Doorbell</td>
</tr>
<tr>
<td>1000h + (1 * 4 &lt;&lt; CAP.DSTRD)</td>
<td>CQ 0 HDBL</td>
<td>Admin Completion Queue Head Doorbell</td>
</tr>
<tr>
<td>1000h + (2 * 4 &lt;&lt; CAP.DSTRD)</td>
<td>SQ 1 TDBL</td>
<td>IO Submission Queue 1 Tail Doorbell</td>
</tr>
<tr>
<td>1000h + (3 * 4 &lt;&lt; CAP.DSTRD)</td>
<td>CQ 1 TDBL</td>
<td>IO Completion Queue 1 Head Doorbell</td>
</tr>
</tbody>
</table>
NVMe: How

Logical View

High Memory
Tail
Head

Low Memory
# NVMe: How

<table>
<thead>
<tr>
<th>Byte/Word</th>
<th></th>
<th></th>
<th>Command ID</th>
<th>Flags</th>
<th>Opcode</th>
<th>Metadata Pointer</th>
<th>PRP 1</th>
<th>PRP 2</th>
<th>Command DWORDS 10 - 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Metadata Pointer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PRP 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PRP 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Command DWORDS 10 - 15</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NVMe: How
NVMe: How
NVMe: Where
NVMe: Where
NVMe: Where
NVMe: per-cpu h/w queues
When CPUs exceed available h/w queues

Share with your neighbors
NVMe: CPU Efficient

Submission latency and CPU cycles reduced >50%*:
• NVMe: 2.8us, 9,100 cycles
• SAS: 6.0us, 19,500 cycles

* Measurement taken on Intel® Core™ i5-2500K 3.3GHz 6MB L3 Cache Quad-Core Desktop Processor using Linux kernel 3.12
The importance of reducing software latency

![App to SSD IO Read Latency (QD=1, 4KB)](chart.png)
NVMe: When we know we succeeded
NVMe: When we know we succeeded

root@pc# grep "SCSI\|NVM" .config
CONFIG_BLK_DEV_NVM=y
# CONFIG_SCSI is not set
NVMe: Original Driver Implementation

- bio-based for performance: lockless block layer
- Driver burdened to manage:
  - timeouts, io statistics, h/w access, tagging, SGL mapping, trace points, queue-to-cpu binding, splitting
Converting NVMe to Request Based

- High IOPS devices cannot reach their potential under single lock
Converting NVMe to Request Based

- High IOPS devices cannot reach their potential under single lock
- But blk-mq can be multithreaded all the way to the h/w
Supporting blk-mq from dm-mpath

• Problems:
  – Required clone before path chosen
  – Submission occurs in atomic context
Supporting blk-mq from dm-mpath

- Request handling deferred to worker thread
- Clone allocated from path’s request_queue
PCI-e Storage Multipathing
NVM-e: Subsystems
NVMe: Identifying Paths

- IEEE EUI-64 and NGUID globally unique identifiers in a subsystem
- Linux tooling relies on SG_IO to inquire device identification and access restriction
Know your PCI-e Topologies: Which is invalid?

A

PCIe Switch

PCI Function 0 NVMe Controller

PCI Function 0 NVMe Controller

NSID 1

NS

A

B

PCIe Switch

PCI Function 0 NVMe Controller

PCI Function 0 NVMe Controller

NSID 1

NS

A

C

PCIe Switch

PCI Function 0 NVMe Controller

PCI Function 0 NVMe Controller

NSID 1

NS

A
blk-mq + dm-mpath in kernel 4.0-rc1

Credits:
• Matias Bjørling: nvme conversion
• Mike Snitzer: multipath device-mapper
• Jens Axboe: block multiqueue
• Bart Van Assche: regression debugging
• Christoph Hellwig: moral support

Still more work to do!
Multipathing for performance

- Submitting I/O to device on remote NUMA node incurs additional latency
- Worsens as node count increases
Case Study: 32 Sockets, 960 CPUs

- NUMA penalty: >30% performance lost
- NUMA “trickery” recovered:
  - irqbalance, numactl, libnuma, custom cpu-queue mapping
  - 30 Million IOPS (SC’14)
The cost of NUMA

Observed Performance Loss on Randomly Scheduled Workloads

Number NVMe: 2x node count, 4k random read to all drives
No CPU pinning
Locality based path selection proposal

• NUMA aware path selection: choose path closest to dispatching CPU
• Ineffective in 4.0: single threaded dispatch
NVMe mpath performance in 4.0

Request based DM performance comparison

Percent of raw device access performance

bs=128k, iodepth=32, numjobs=1
bs=4k, iodepth=16, numjobs=8
device-mapper blk-mq conversion

• Parallelizes entire stacking layer
  – Make blk-mq entry capable in preempt disabled context

• queue setup: how to determine number of “h/w” queues and tags per queue to allocate
  – Want enough to satisfy h/w, but not wasteful

• Developed by Mike Snitzer during LSFMM
  – performs at 99% raw speed in fio benchmarks on tested NVMe h/w

• Staged for Linux kernel 4.1 integration
Alternative multipath proposal

• Make blk-mq multipath aware
  – Removes stacking/re-entry requirement
  – More efficient use of resources
  – Tighter integration to h/w
• Ideas initiated by Hannes Reinecke and Christoph Hellwig
• Implementation and collaboration details ongoing
  – PoC expected within weeks
Thank you

keith.busch@intel.com
Backup
Path selection code snippet

```c
static struct dm_path *numa_select_path(struct path_selector *ps,
                                         unsigned *repeat_count, size_t nr_bytes)
{
    struct selector *s = (struct selector *) ps->context;
    struct path_info *best = NULL, *pi;
    int cur = INT_MAX, node = cpu_to_node(smp_processor_id());

    list_for_each_entry(pi, &s->valid_paths, list) {
        int pnode = pi->path->dev->bdev->bd_queue->node;
        int val = node_distance(node, pnode);
        if (val < cur) {
            best = pi;
            cur = val;
        }
    }

    *repeat_count = 1;
    return best ? best->path : NULL;
}
```