NVME OVER FABRICS OFFLOAD

Tzahi Oved

Mellanox Technologies

[ March, 2019 ]
NVME, NVME-OF INTRO
NVME INTRODUCTION

- **Standard PCIe host controller interface for solid-state storage**
  - Driven by industry consortium of 80+ members
  - Standardize feature, command, and register sets
  - Enhance PCIe capabilities: low latency, scalable BW, power efficiency etc.

- **Focus on efficiency, scalability and performance**
  - All parameters for 4KB command in single 64B DMA fetch
  - Simple command set (13 required commands)
  - Supports MSI-X and interrupt steering
THE NEED FOR NVME OVER FABRICS

- Regular NVMe Devices “Captive” In the Server
  - Only supports drives in or near server/storage box
  - Limits number of PCIe-attached devices

- PCIe, SAS and SATA Don’t Support Scale-Out
  - Distance limitations; Difficult to share
  - Limited robustness and error handling

NVM Express, Inc. solution

- NVMe Over Fabrics Announced September 2014
  - Preserves NVMe command set
  - Simplifies storage virtualization, migration, & failover
  - Allows scale-out without SCSI protocol translation
  - Goal: <10µs added latency compared to local NVMe

- Standard V1.0 done, published June 2016
# NVME MAPPING TO FABRICS

<table>
<thead>
<tr>
<th>NVMe</th>
<th>NVMe-oF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit Queue (SQ)</td>
<td>QP SQ</td>
</tr>
<tr>
<td>Completion Queue (CQ)</td>
<td>QP RQ (+CQ)</td>
</tr>
<tr>
<td>Host write SQE, ring doorbell</td>
<td>Host SEND SQE capsule, ring doorbell</td>
</tr>
<tr>
<td>Device write CQE, interrupt or poll</td>
<td>Target SEND CQE capsule, RX CQE int/poll</td>
</tr>
<tr>
<td>PCIe data exchange</td>
<td>RDMA RD/WR, immediate up to 8K</td>
</tr>
</tbody>
</table>
COMMUNITY NVMF TARGET KERNEL

TARGET SYSTEM

NVMF Target
Convert NVMe -> Block Request

RDMA Rx
NVMe Request

Get properties
- IO queues
- Comp queues
- Doorbell

Block Layer

NVMe Driver
SCSI Layer
File Loop

Clients

ConnectX/BlueField

Offload

NVMe Device
SCSI Device
Local File

OpenFabrics Alliance Workshop 2019
NVME-OF TARGET OFFLOAD FLOW
NVME-OF TARGET OFFLOAD WITH CMB

Zero CPU cycles
No traffic through system memory

Initiator (client)
Fabric
ConnectX-5 RNIC
CPU
Memory
IOQ
E CQ
Data Buffer
Doorbell

OpenFabrics Alliance Workshop 2019
STATUS

- **Submitted RFC**
  - [https://www.spinics.net/lists/linux-rdma/msg58512.html](https://www.spinics.net/lists/linux-rdma/msg58512.html)

- **Added/Updated files**
  - `Documentation/nvmf_offload.md` | 172
  - `libibverbs/man/ibv_create_srq_ex.3` | 48
  - `libibverbs/man/ibv_get_async_event.3` | 15
  - `libibverbs/man/ibv_map_nvmf_nsid.3` | 89
  - `libibverbs/man/ibv_qp_set_nvmf.3` | 53
  - `libibverbs/man/ibv_query_device_ex.3` | 26
  - `libibverbs/man/ibv_srq_create_nvme_ctrl.3` | 89
  - `libibverbs/verbs.h` | 107
Identify
- NVMe-oF offload capabilities of the device

Create SRQ
- with NVMe-oF offload attributes
- represents a single NVMe-oF subsystem

Create NVMe backend
- Represents locally attached NVMe subsystems

Map namespaces
- From: front-end facing namespace id
- To: backend NVMe + namespace id

Create QP
- connected with clients
- bound to an SRQ with NVMe-oF offload
- enable NVMe-oF offload
- After CONNECT seen

Modify QP
IDENTIFY - IBV_QUERY_DEVICE_EX()

- New NVMf caps
- Offload types
- Supported min/max values

```c
struct ibv_device_attr_ex {
    ...
    struct ibv_nvmf_caps  nvmf_caps;
};

struct ibv_nvmf_caps {
    enum nvmf_offload_type  offload_type;
    uint32_t  max_backendCtrls_total;
    uint32_t  max_backendCtrls;
    uint32_t  max_namespaces;
    uint32_t  max_staging_buf_pages;
    uint32_t  min_staging_buf_pages;
    uint32_t  max_ioccsz;
    uint16_t  max_nvme_queue_sz;
    uint16_t  min_nvme_queue_sz;
    uint32_t  max_ioccsz;
    uint32_t  min_ioccsz;
    uint16_t  max_icdoff;
};
```
CREATE SRQ - IBV_CREATE_SRQ_EX()

- Represents a fabric-facing NVMf target
- Set params according to caps
- Add staging buffer
  - Use MR!

```c
struct ibv_srq_init_attr_ex {
  ...
  struct ibv_nvme_attr nsf_attr;
};

struct ibv_nvme_attr {
  enum nvme_offload_type offload_type;
  uint32_t max_namespaces;
  uint8_t nvme_log_page_sz;
  uint32_t ioccsz;
  uint16_t icdoff;
  uint32_t max_io_sz;
  uint16_t nvme_queue_sz;
  struct ibv_mr *staging_buf_mr;
  uint64_t staging_buf_addr;
  uint64_t staging_buf_len;
};
```
CREATE NVME BACKEND - IBV_SRQ_CREATE_NVME_CTRL()

- **ibv_nvme_ctrl** belongs to specific SRQ

- **Attributes are**
  - NVMe SQ
  - NVMe CQ
  - NVMe SQ-DB
  - NVMe CQ-DB
  - NVMe SQ-DB init val
  - NVMe CQ-DB init val

- **SQ, CQ, and DBs are described by**
  - {MR, VA, LEN}
  - Need to ibv_reg_mr()
New map within the subsystem between
  • \{ fe_nsid \} -> \{ nvme_ctrl, nvme_nsid, params \}

SRQ is available from the nvme_ctrl
  • It was created for a specific SRQ

To map the same NVMe to different SRQ
  • Meaning to different NVMe-oF subsystems
  • Create different nvme_ctrl with each SRQ
  • Each nvme_ctrl represents exclusive NVMe queues on the same NVMe device

```c
int ibv_map_nvmf_nsid(
    struct ibv_nvme_ctrl *nvme_ctrl,
    uint32_t fe_nsid,
    uint16_t lba_data_size,
    uint32_t nvme_nsid);

int ibv_unmap_nvmf_nsid(
    struct ibv_nvme_ctrl *nvme_ctrl,
    uint32_t fe_nsid);
```
ENABLE QP - IBV_QP_SET_NVMF()

- Create QP – as normal
  - RDMA-CM in case of NVMf

- Enable QP for NVMf offload
  - New verb specifically for NVMf attrs

- First message is CONNECT
  - No offload should be done before it
  - Software will enable offload after seeing CONNECT

```c
int ibv_modify_qp_nvmf(
    struct ibv_qp *qp,
    int flags);

enum {
    IBV_QP_NVMF_ATTR_FLAG_ENABLE = 1 << 0,
};
```
EXCEPTIONS

- **Non-offloaded operations**
  - Go as normal to SRQ and CQ
  - Software can post responses on QP

- **QP goes to error**
  - Async event IBV_EVENT_QP_FATAL
  - Software may not see CQE with errors...

- **NVMe errors**
  - New async event, nvme_ctrl scope
  - PCI error (when reading CQ)
  - Command timeout

- **Must listen to async events!**

```c
struct ibv_async_event {
    union {
        ...
        struct ibv_nvme_ctrl *nvme_ctrl
    } element;
    ...
};
```

NVMe controller events:

- IBV_EVENT_NVME_PCI_ERR NVMe backend controller PCI error
- IBV_EVENT_NVME_TIMEOUT NVMe backend controller completion timeout
**Implemented Out of band**
- Connect to Cloud provider management/orchestration
- Implement any proprietary protocol
- Direct from network, bypassing host
- Utilizing NVMe Emulation SDK

**Vendor-specific Admin commands**
- Using the NVMe driver
- Limited / controlled capabilities
- Example: allow user to choose QoS policy
NVME EMULATION DATAPATH

- **SDK**
  - Handle NVMe registers and admin queue
  - Efficient memory management based on SPDK
  - Zero-copy all the way
  - Full polling
  - Multi queues, multi threads, lockless
  - Well defined APIs: vBdev, Bdev drivers…
THANK YOU
Tzahi Oved
Mellanox Technologies