Prototyping
Byte-Addressable NVM Access

Bernard Metzler & Animesh Trivedi
IBM Zurich Research Laboratory
Agenda

• NVM host integration
  – Status quo of DSA development
  – The Storage Abstraction Layer
  – OFA DSA user interfaces
  – Other user interfaces

• NVM/RDMA networking integration
  – Example prototypes

• Findings: New OFA interface requirements

• Outlook
Recap from 2014

• Idea: Use OFA stack for local NVM access
  – Integrate with OFA as just another verbs provider

• Benefits
  ✓ Application private device channel (virtually unlimited number)
  ✓ Deep request queues / async. operations
  ✓ Byte-level I/O
  ✓ OFA Verbs API: well established interface

• Issues
  o Inflexible memory registration/re-registration
  o RDMA network access
Host Integration: DSA

- **Integrates with OpenFabrics industry standard environment**
- **Direct Storage Access Driver (DSA)**
  - ‘DSA’ OFA module and ‘libdsa’ library
  - Provides RDMA API for access to all integrated flash resources at byte granularity
- **Storage Abstraction Layer**
  - Abstracts from device specifics
  - Exports flash partitions
  - Device I/O attached (local or network)
- **Block Layer** OFA kernel verbs client
  - Supports legacy block I/O to NVM devices
SAL: Storage Abstraction Layer

- Storage device interface
  - Upcalls:
    - register/unregister device
    - register/unregister/change partition
    - I/O and command completion(rv, *ctx)
    - publish_region(*part, *attrs)
  - Downcalls:
    - sal_write(*part, off, len, *sl, *ctx)
    - sal_read(*part, off, len, *sl, *ctx)
    - sal_trim(*part, off, len, flags, *ctx)
    - sal_reg_region(*part, *attrs)
    - sal_modify_region(*part, *attrs)
    - sal_dereg_region(*part)
  - All fast path operations share context with provider
    - Context with provider and caller private regions
    - Aim at cache and multi-core efficiency

March 15 – 18, 2015

#OFADevWorkshop
**SAL: Maintaining NVM Resources**

- All storage providers register with Storage Abstraction Layer
  - Provider resource representation: Partition [ID, length, rights]
  - SAL exposes resources at /sys fs
- Storage resource reserved with OFA subsystem: `ibv_reg_mr()` call
  - Same as local DRAM registration
  - Flash and local DRAM can be source or sink of any RDMA operation
  - Will allow for flash-to-flash read/write operations
- Kernel DSA clients have function call/RPC interface (not shown here)
- DSA/SAL maps between RDMA key, len, off and partition ID, off, len
  - During registration
  - On any data operation, enforces protection

---

**Diagram:**

- **Storage Abstraction Layer**
  - **OpenFabrics/DSA**
    - **DRAM Memory Region**
    - **IO Memory Region**
    - ** RDMA operation**
  - **Storage Device A**
    - **Partition: ID, length, rights**
    - **free space**
    - **Example local IO attached storage**
  - **Storage Device B**
    - **Partition: ID, length, rights**
    - **free space**
    - **Example network attached storage backend**
    - **NW Link**
    - **Storage Box**
DSA Application Interface: Referencing NVM resources

- Nothing different from DRAM access
  - NVM resource described by [key, off, len]
- Single key space shared with DRAM reservations
  - Both DRAM and (IO) NVM are registered with DSA
  - On the fast path, DSA detects memory type by key
  - SGL support: WR’s with mixed SGL’s possible (not yet supported by DSA)

Example mixed SGL usage
Register Memory with mmap() / VA

• DSA/SAL : Storage resources in /sys file system
• User
  – fd = open(/sys/....../partitions/f1/memory, O_RDWR)
  – val= mmap(NULL, 40960, PROT_NONE, fd, 0)
  – Takes va1 to DSA OFA device for registration:
    mr1 = ibv_reg_mr(dsa_pd, va1, ...)
  – Registers source/target va2 in DRAM:
    mr2 = ibv_reg_mr(dsa_pd, va2, ...)
  – Makes and connects Queue Pair within DSA
  – Posts READ/WRITE RDMA operations:
    src=mr1, trgt=mr2
  – Reaps work completions
  – Persistent reservations can be replayed at system boot

• Extensible for
  – storage <-> storage transfers
  – Direct load/store into IO mem (work in progress)
Register Memory w/o VA

Not maintaining a VA may have its merits

1. RPC protocol between application and SAL
   - post_send()/post_receive() between SAL and application
   - RPC’s to discover NVM resources and make reservations
   - SAL translates into/from SAL device down calls/upcalls
   - Reservations visible in /sys file system as well
   - Reservation RPC returns key to be used with DSA
   - No VA: zero based addressing for given key
   - Used by kernel clients, supported also at user level
     o RPC mechanism shadows send/receive application usage, tagged messages would help

2. Alternative: File handle (not VA) for registration
   - Needs extended memory registration semantics
   - `ibv_reg_mr(struct ibv_pd *pd, void *addr, size_t length, int access);`
   - `fi_mr_reg(...);`
   - Not supported yet
Strictly ordered I/O execution/completion: *not supported by DSA/SAL: application or device duty*

- Lazy Ordered completion: *default*
- Explicit unordered completion: *work in progress*
Legacy File I/O Integration: Block IO

- Block Driver: Kernel Verbs client
- Integrates DSA with Linux file system
- Multiple DSA QP’s for efficient multi-core support
  - Similar to Multi-Q-BIO
- Memory reservation via RPC protocol with SAL
  - Send/Receive work requests with SAL peer
  - Resource discovery (devices and partitions)
  - Resource reservation (whole partition only)
  - Reservations visible in /sys file system
- I/O throughput similar to user level verbs
- TRIM command supported
  - dsa_rpc_trim(key, flags, length, offset) : currently send WR
  - Asynchronous completion: currently RPC interface: receive WR
DSA: Supporting Load/Store to NVM

✓ File I/O
  ✓ Supported via DSA block device

• Load/store to mmap()’ed NVM
  – mmap(PROT_READ|PROT_WRITE)
  – Handling page faults
  – Own page pool
  – OFA kernel client
  – Work in progress
Prototype NVM - RDMA Network Integration

Some ways to integrate NVM with RDMA network

1. Bridging application
   (Breaks end-to-end RDMA semantic)
   1. User- or kernel-level verbs client
      • DRAM buffer registered with DSA and RNIC
      • Tolerable latency (user level app: some I/O 65us + RDMA Read 3us + appl. 7us)
   2. read/write mmap’ed file, register with RNIC
      o Would bring in all pages

2. Fusing with RDMA NW stack
   1. RDMA/NVM Appliance
   2. In-kernel fusing with software RDMA stack (see next page)
NVM - RDMA Network Integration Prototype

- Splice SoftiWarp with NVM access
- Preserves RDMA end-to-end semantics
  - Application reserves IO memory for RDMA
  - Peer directly accesses via reservation key
  - Direct remote READ/WRITE execution by siw
- Needs extensions
  - RDMA provider (siw)
    - IO memory registration similar to DSA
    - \( \text{rx} + \text{tx} \) path: resolve IO memory, bail/resume
  - SAL interface additions
    - **Downcalls**
      - inbound READ/WRITE: \( \text{get_iomem}(\text{part}, \text{off}, \text{size}, \text{op}, *\text{ctx}) \)
      - after read/write finished: \( \text{sync}() \)
      - if not longer referenced: \( \text{put_iomem}_\text{page}() \)
    - **Upcall**
      - complete get_iomem(): \( \text{get_iomem}_\text{callback}(\text{err}, \text{off}, *\text{page}, *\text{ctx}) \)
Example Operation (WRITE)

1. Resource registration with SAL
2. Application mmap() of resource
3. ibv_reg_mr()
4. Resource key passed to peer
5. Peer WRITE access
6. SIW resolves IO mem
7. SAL request get_iomem(), siw bail-out
8. SAL upcalls with IO pages
9. SIW resumes placing data into IO pages
10. sync() with storage provider & put_iomem_page()

- SIW requests/maintains current IO pages
  - Pre-fetching if signaled by DDP
  - Direct placement if page available, stall/resume otherwise
- Local Completion semantics
  - Data ‘visible’ in provider, or data placed into persistence domain
  - Currently completion if ‘visible’ since no completion semantics selectable
- Head of line blocking if some I/O pages are ‘cold’
  - RDMA UC Service: SIW/UDP version ready to be tested
Findings

• OFA infrastructure good fit for NVM access
• Incomplete wish list of API extensions
  – Re-registration of persistent memory objects
  – Selectable NVM access completion semantics
  – Selectable NVM access completion ordering
  – Registration of NVM w/o VA
  – Zero based addressing from user space
  – Larger key space (currently just 24 bit) preferred
  – Command interface (e.g. explicit Trim support etc.)
Outlook

- NVM integration part of Zurich IBM Research effort for cloud stack optimization (jVerbs, DaRPC, siw, HyV, Peregrine, ...)
- DSA open sourcing
  - Will come with example storage provider (fakes NVM device/partitions in DRAM)
  - We will add NVMe/SAL integration
  - Working on load/store interface
- Further work towards NVM/network integration
  - Consider open sourcing siw extensions
  - Experiments with UC RDMA Flash access (UDP based siw/NVM integration as already prototyped for radio-astronomic SKA project)

March 15 – 18, 2015 #OFADevWorkshop
Thank You