13th ANNUAL WORKSHOP 2017

REMOTE PERSISTENT MEMORY ACCESS – WORKLOAD SCENARIOS AND RDMA SEMANTICS

Tom Talpey

Microsoft

[March 31, 2017]

OUTLINE

- **Windows Persistent Memory Support**
  - A brief summary, for better awareness
- **RDMA Persistent Memory Extensions**
  - And their motivation/use by Storage Protocols
- **Example Application Scenarios for Persistent Memory Operations**
  - RDMA Operation Behavior

WINDOWS PERSISTENT MEMORY SUPPORT

WINDOWS PMEM SUPPORT

- **Persistent Memory is supported in Windows 10 and Windows Server 2016**
  - PM support is foundational in Windows and is SKU-independent
- **Support for JEDEC-defined NVDIMM-N devices available in**
  - Windows Server 2016
  - Windows 10 (Anniversary Update – Fall 2016)
- **Access methods:**

✓**Direct Access (DAX) Filesystem**
  - Mapped files with load/store/flush paradigm
  - Cached and noncached with read/write paradigm

✓**Block-mode ("persistent ramdisk")**
  - Raw disk paradigm

✓**Application interfaces**
  - Mapped and traditional file
  - NVM Programming Library
  - "PMEM-aware" open coded

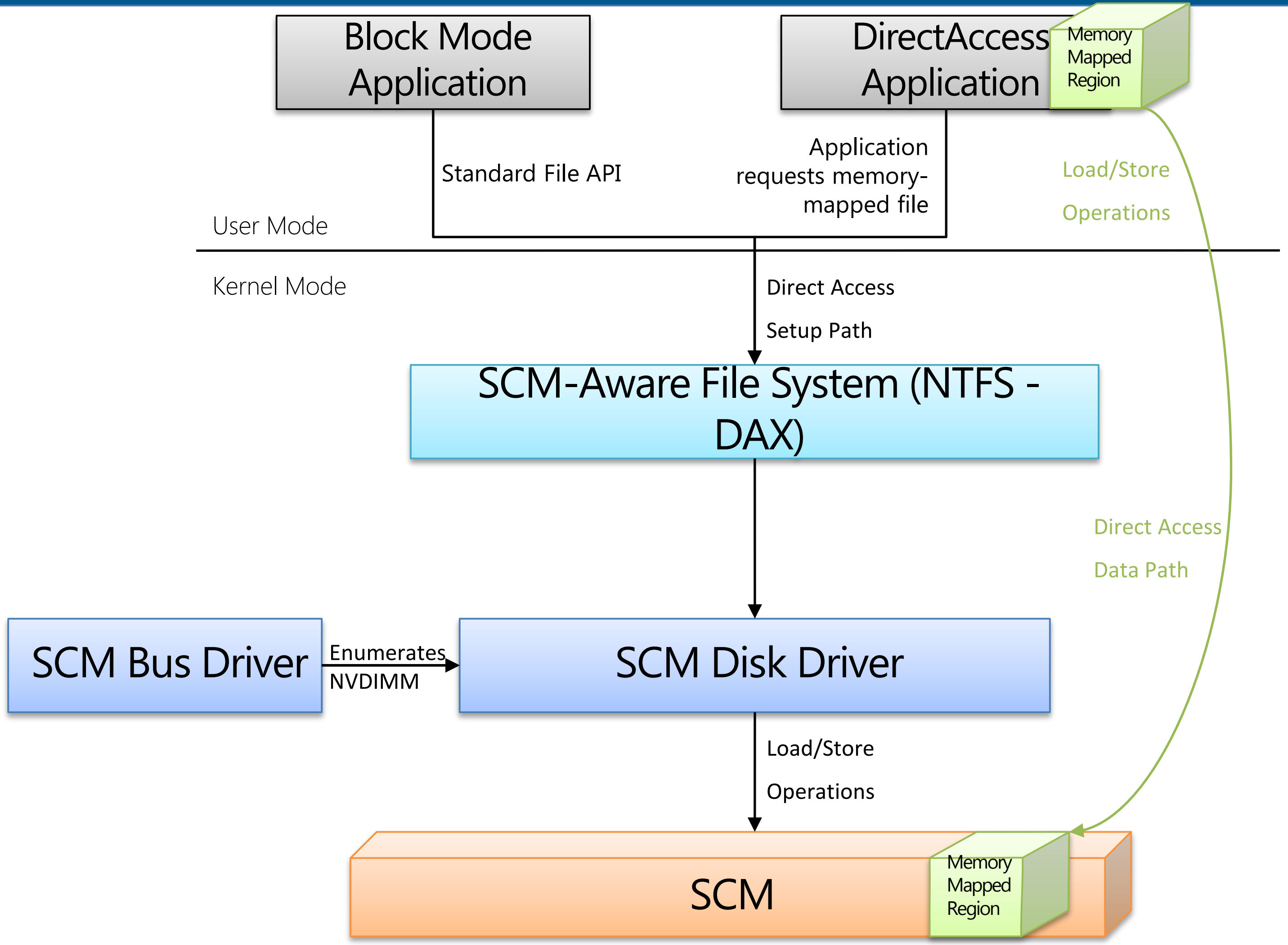
DIRECT ACCESS ARCHITECTURE

Overview

- Support in Windows Server 2016 and Windows 10 Anniversary Update (Fall 2016)
- App has direct access to Storage Class Memory (SCM/Pmem) via existing memory-mapping semantics
- Updates directly modify SCM, Storage Stack not involved
- DAX volumes identified through new flag

Characteristics

- True device performance (no software overhead)
- Byte-Addressable
- Filter Drivers relying on I/O may not work or attach – no I/O, new volume flag
- AV Filters can still operate (Windows Defender already updated)



IO IN DAX MODE

- **Memory Mapped Access**
  - This is true zero-copy access to storage
    - An application has direct access to persistent memory
  - Important → No paging reads or paging writes will be generated
- **Cached IO Access**
  - The cache manager creates a cache map that maps directly to PM hardware
  - The cache manager copies directly between user's buffer and persistent memory
    - Cached IO has one-copy access to persistent storage
  - Cached IO is coherent with memory mapped IO
  - As in memory mapped IO, no paging reads or paging writes are generated
    - No Cache Manager Lazy Writer thread
- **Non-Cached IO Access**
  - Is simply converted to cached IO by the file system
    - Cache manager copies directly between user's buffer and persistent memory
  - Is coherent with cached and memory mapped IO

BACKWARD APP COMPATIBILITY ON PM HARDWARE

- **Block Mode Volumes**
  - Maintains existing storage semantics
    - All IO operations traverse the storage stack to the PM disk driver
    - Sector atomicity guaranteed by the PM disk driver
    - Has shortened path length through the storage stack to reduce latency
      - No storport or miniport drivers
      - No SCSI translations
  - Fully compatible with existing applications
  - Supported by all Windows file systems
  - Works with existing file system filters
  - Block mode vs. DAX mode is chosen at format time

PERFORMANCE COMPARISON

4K random writes

1 Thread, single core

	IOPS	Avg Latency (ns)	MB / Sec
NVMe SSD	14,553	66,632	56.85
Block Mode NVDIMM	148,567	6,418	580.34
DAX Mode NVDIMM	1,112,007	828	4,343.78

USING DAX IN WINDOWS

DAX Volume Creation

→`Format n: /dax /q`

→`Format-Volume –DriveLetter n –IsDAX $true`

DAX Volume Identification

Is it a DAX volume?

→**call** `GetVolumeInformation(“C:\”, ...)`

→**check** `lpFileSystemFlags` **for** `FILE_DAX_VOLUME (0x20000000)`

Is the file on a DAX volume?

→**call** `GetVolumeInformationByHandleW(hFile, ...)`

→**check** `lpFileSystemFlags` **for** `FILE_DAX_VOLUME (0x20000000)`

Memory Mapping

```
1. HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, 0, NULL);
2. LPVOID baseAddress = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, size);
3. memcpy(baseAddress + writeOffset, dataBuffer, ioSize);
4. FlushViewOfFile(baseAddress, 0);
```

OR … use non-temporal instructions for NVDIMM-N devices for better performance

```
1. HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, 0, NULL);
2. LPVOID baseAddress = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, size);
3. RtlCopyMemoryNonTemporal(baseAddress + writeOffset, dataBuffer, ioSize );
```

Future … use open source NVM Programming Library

REMOTE ACCESS TO PERSISTENT MEMORY

GOING REMOTE

- **One local copy of storage isn't storage at all**
  - Basically, temp data
- **Enterprise-grade storage requires replication**
  - Multi-device quorum
  - In addition to integrity, privacy, manageability, … (requirements vary)
- **Remote access is required**

- **Pmem value is all about LATENCY**
  - Single digit microsecond remote latency goal
  - Which btw is 2-3 orders of magnitude better than today's block storage
    - We can take steps to get there, with great benefit at each
- ➢ **Use RDMA**
  - Requires an RDMA protocol extension

RDMA PROTOCOLS

- **Need a remote guarantee of <u>Durability</u>**
- **RDMA Write alone is not sufficient for this semantic**
  - This is an RDMA conference, you know that ☺
- **An extension is required**
  - Proposed “RDMA Commit”, a.k.a. “RDMA Flush”
- **Executes like RDMA Read**
  - Ordered, Flow controlled, acknowledged
  - Initiator requests specific byte ranges to be made durable
  - Responder acknowledges only when durability complete
  - Strong consensus on these basics
- **Being discussed in IBTA, SNIA and other venues**
  - Details being worked out
  - Scope of durability: region-based, region-list-based, connection, all under discussion
    - Connection scope seems most efficient for implementations
  - Additional semantics possible (later in this deck)

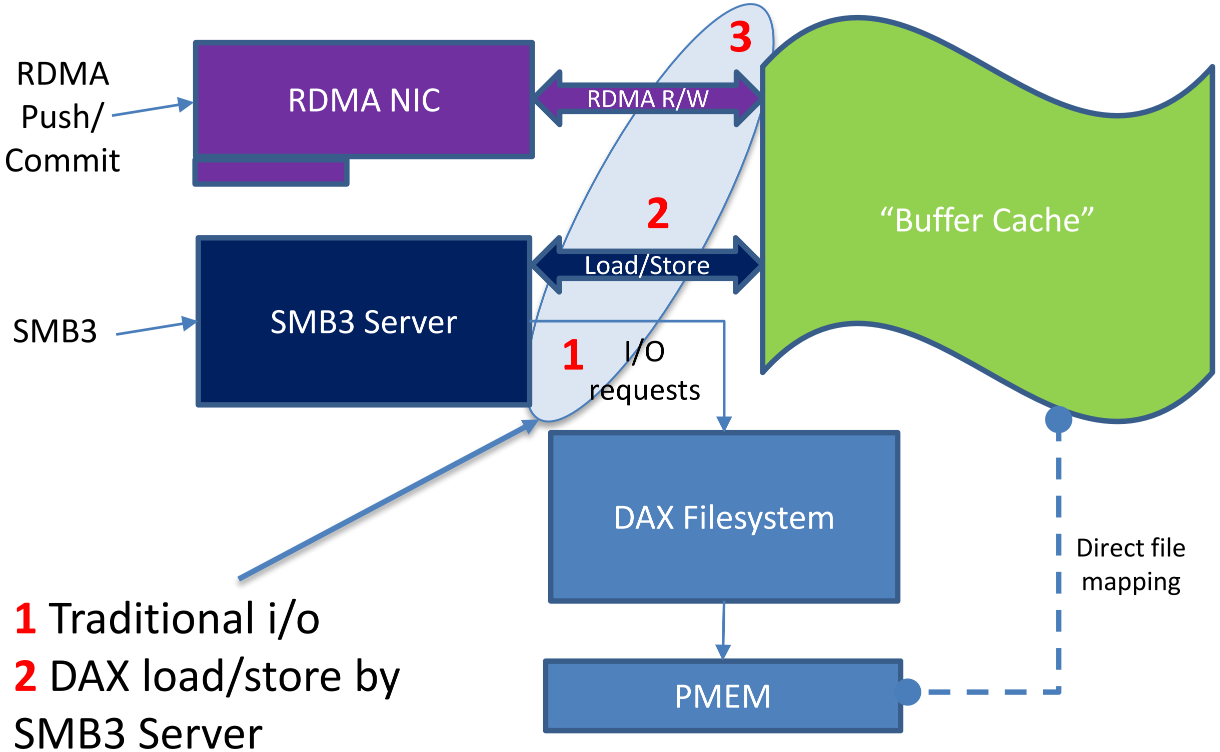
RDMA-AWARE STORAGE PROTOCOL USE

- **SMB3/SMB Direct**
  - “Push Mode”
- **NFS/RDMA**
  - See Chuck Lever’s Tuesday presentation
- **Other**
  - Commit can work to any remotely-mappable device, e.g. NVMe with a PCIe BAR
  - Anything that can be memory-registered and accessed via RDMA

- **Note to OFA: there will be Verbs.**

EXAMPLE: GOING REMOTE – SMB3

- **SMB3 RDMA and "Push Mode" discussed at previous SNIA Storage Developers Conferences**
- **Enables zero-copy remote read/write to DAX file**
  - Ultra-low latency and overhead
- **2, 3 can enable even *before* RDMA Commit extensions become available, with slight extra cost**



1 Traditional i/o
2 DAX load/store by SMB3 Server
3 Push Mode direct from RDMA NIC

REMOTE PMEM WORKLOADS

BASIC REPLICATION

- **Write, optionally more Writes, Commit**
  - No overwrite
  - No ordering dependency (but see logwriter and non-posted write)
  - No completions at data sink
  - Can be pipelined
- **Other semantics:**
  - Asynchronous mode
    - Discussion in SNIA NVMP TWG
      - Where it’s affectionately named “Giddy-Up”
    - Local API behavior at initiator to perform Commit asynchronously
    - Enables remote write-behind for load/store access, among other scenarios
    - Complicates error recovery, but in well-defined way
  - Reads are interesting too
    - But easily interleaved with writes/commits
  - ✓ No protocol implications (envisioned)

LOG WRITER (FILESYSTEM)

- **For (ever)**

  { Write log record, Commit }, { Write log pointer, Commit }

  - Latency is critical
  - Log pointer cannot be placed until log record is successfully made durable
    - Log pointer is the validity indicator for the log record
    - Transaction model
  - Log records are eventually retired, buffer is circular
- **Protocol implications:**
  - Must wait for first commit (and possibly the second)
  - Introduces a pipeline bubble – very bad for throughput and overall latency
  - Desire an ordering between Commit and second Write
- **Possible solution: "Non-posted write"**
  - Special Write which executes "like a Read" – ordered with other non-posted operations
    - For example, Commits, Reads
  - Being discussed in IBTA

LOG WRITER (DATABASE)

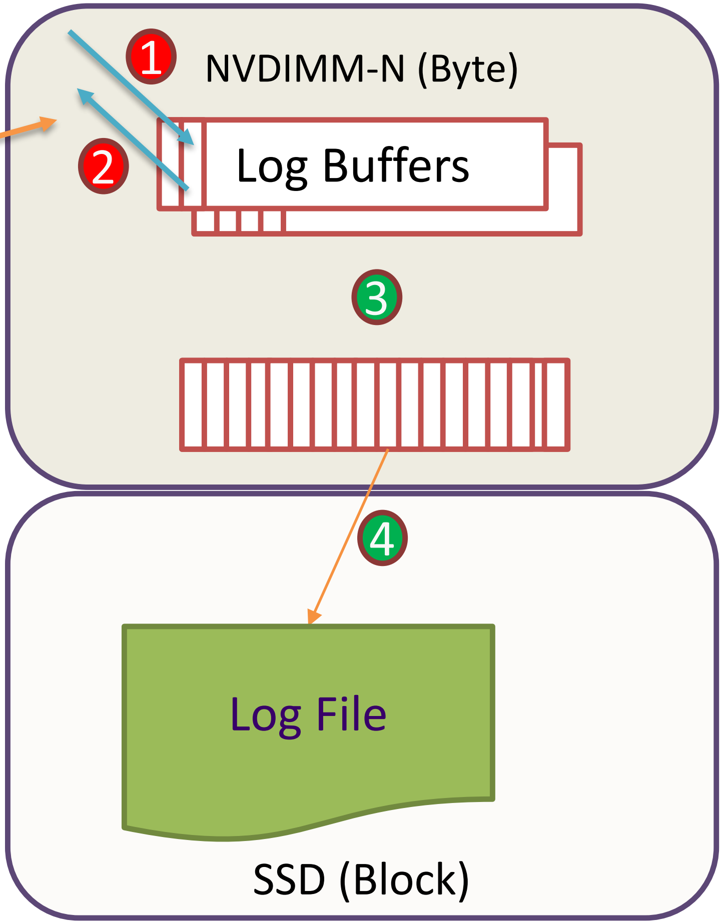
- **For (ever)**

  While (!full) { Write log record, Commit }

  Commit log segment

  Persist segment to disk (asynchronously)

  - Log record write/commit latency (red 1/2) **critical**
  - Log segment persist to disk latency (green ¾) **not** critical
  - Large improvement to database transaction rate
    - Approximately **2x** for SQL Hekaton*
- **Very similar to log-based filesystem scenario**
  - Similar RDMA protocol implication, but see next



Configuration	HK on NVMe (block)	HK on NVDIMM-N (DAX)
Row Updates / Second	63,246	124,917
Avg. Time / Txn (ms)	0.379	0.192

Configuration: Row Size: 32B, Table Size: 5GB, Threads:24, Batch Size: 1

LOG WRITE WITH ACTIVE-ACTIVE SIGNALING

- **After log record replication, how to make peer aware of it?**
  - Non-posted operations do not generate peer completions
    - E.g. Commit, RDMA Read, Atomic
    - … and are not ordered with Posted operations (e.g. Send, Write with Immediate)
- **Desire to generate a peer completion, only after durability achieved**
- **Simple way: initiator waits for Commit completion**
  - Using an initiator Fence, or explicitly waiting
  - Pipeline bubble (bad for latency)
- **Better way: ordered operation rule <u>at target</u>**
  - Under discussion as part of the extension

REMOTE DATA INTEGRITY

- **Assuming we have an RDMA Write + RDMA Commit**
- **And the Writes + Commit all complete (with success or failure)**
- **How does the initiator know the data is intact?**
  - Or in case of failure, which data is **not** intact?
- **Possibilities:**
  - Reading back
    - extremely undesirable (and possibly not actually reading media!)
  - Signaling upper layer
    - high overhead
    - Upper layer possibly unavailable (the “Memory-Only Appliance”!)
  - Other?
- **Same question applies also to:**
  - Array “scrub”
  - Storage management and recovery
  - etc

RDMA "VERIFY"

- **Concept: add integrity hashes to a new operation**
  - Or, possibly, piggybacked on Commit
  - Note, not unlike SCSI T10 DIF
- **Hash algorithms to be negotiated by upper layers**
- **Hashing implemented in RNIC or Library "implementation"**
  - Which could be in
    - Platform, e.g. storage device itself
    - RNIC hardware/firmware, e.g. RNIC performs readback/integrity computation
    - Other hardware on target platform, e.g. chipset, memory controller
    - Software, e.g. target CPU
  - Ideally, as efficiently as possible
- **Options:**
  - A. Source requests hash computation, receives hash as result, performs own comparison
  - B. Source sends hash to target, target computes and compares, returns success/failure
  - C. ???
- **Under discussion in SNIA NVMP TWG OptimizedFlushAndVerify()**

THE (NEAR?) FUTURE

- **Hope to see all the above remote scenarios supported**
- **Operating system support well established (Windows, Linux)**
- **Protocol standards process well under way**
- **High hopes for 2017!**

OPENFABRICS ALLIANCE

13th ANNUAL WORKSHOP 2017

THANK YOU