Windows Persistent Memory Support

Neal Christiansen

Microsoft

Principal Development Lead

What is “Persistent Memory”?

- Non-volatile storage with RAM-like performance
  - Low latency/high bandwidth.
- Resides on the memory bus
- Terms used to describe the hardware:
  - Storage Class Memory (SCM)
  - Byte Addressable Storage (BAS)
  - Non-Volatile Memory (NVM)
  - Persistent Memory (**PM**) ⇦ Industry converging on this term

File Systems and Persistent Memory

- **PM is a disruptive technology**

- Customers want the fastest performance
  - System software is in the way!
- Customers want application compatibility
- Conflicting goals

Windows Goals for Persistent Memory

- Support zero-copy access to persistent memory
- Most existing user-mode applications will run without modification
- Provide an option to support 100% backward compatibility
  - Does introduce new types of failure modes
- Provide sector granular failure modes for application compatibility

Windows PM Support

- PM support is foundational and Windows SKU independent
- Support for JEDEC-defined NVDIMM-N devices available in Windows 10 Anniversary Update and Windows Server 2016
  - Available for preview in Windows 10 Insider Builds and Windows Server 2016 TP5

Introducing a New Class of Volume

- Direct Access Storage (DAX) Volume
  - On DAX formatted volumes memory mapped files map directly to PM hardware
    - No change to existing memory mapping APIs
  - Maximizes application performance
  - DAX Volumes are currently supported by NTFS
    - Part of Windows 10 Anniversary Update / Server 2016 releases

Memory Mapped IO in DAX mode

- Supports 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 in DAX mode

- 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
- No paging reads or paging writes
- No Cache Manager Lazy Writer thread

Non-cached IO in DAX Mode

- Is converted to cached IO by the file system
  - Cache manager copies directly between user's buffer and persistent memory

File System Metadata in DAX Mode

- NTFS file system metadata does not use DAX mode sections
  - Meaning paging reads/writes will be generated for all file system metadata operations
  - Needed to maintain existing ordered write guarantees for write-ahead logging

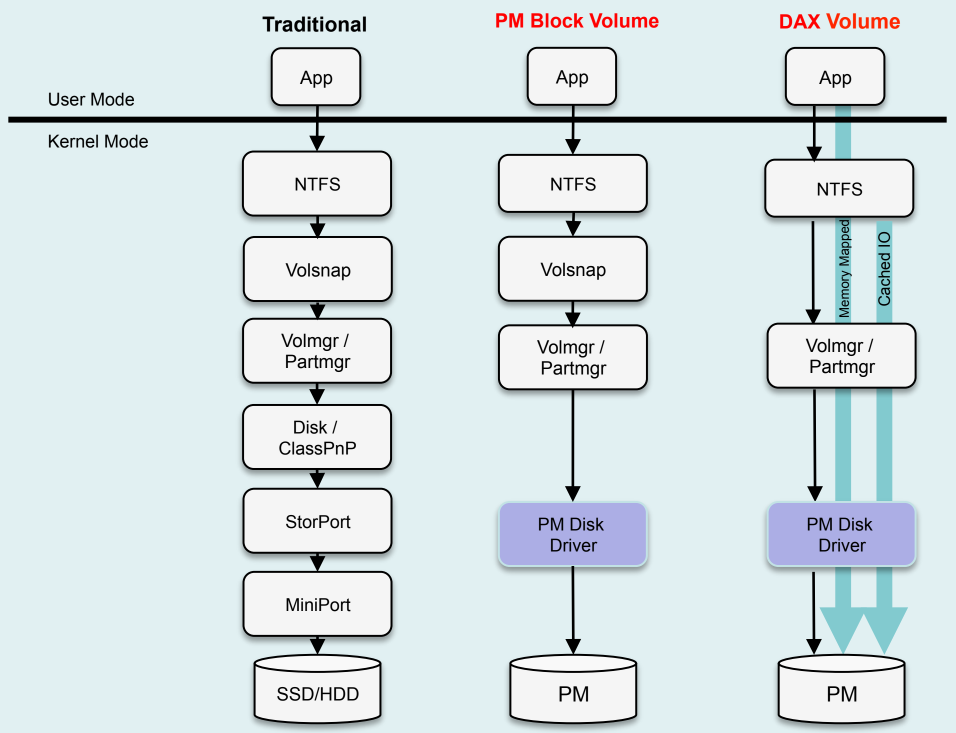
Impacts to File System Functionality in DAX Mode

- Direct access to persistent memory eliminates the traditional hook points that file systems use to implement various features
- File system functionality not supportable on DAX volumes:
  - No NTFS software encryption support (EFS)
  - No NTFS software compression support
- File system no longer knows when a writeable memory mapped section is modified:
  - These like modification and access times are updated when a writeable mapped section is created

Backward Compatibility on PM Hardware

- Block Mode Volumes
  - Maintains existing storage semantics
    - All IO operations traverse the storage stack to the SCM disk driver
    - Sector atomicity guaranteed by the SCM disk driver
    - Has shortened path length through the storage stack to reduce latency
  - 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

IO Stack Comparisons



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-N	148,567	6,418	580.34
DAX Mode NVDIMM-N	1,112,007	828	4,343.78

Accelerating SQL 16 with PM

	Row Updates / Second	Avg. Time / Txn (ms)
NVMe SSD	63,246	0.379
Dax Mode NVDIMM-N	124,917	0.192

Sector Atomicity

- BTT – Block Translation Table
  - Algorithm created by Intel
  - Provides efficient sector level atomicity of writes
    - Eliminates sub-sector torn writes
    - On power loss either see contents of old sector or new sector
    - Provides compatibility for existing applications that have built-in assumptions around storage failure patterns
    - Minimal performance impact
  - Implemented by remapping the physical address of a given LBA (volume relative logical block address)

Application use of PM

- Intel NVML Library
  - Open source library implemented by Intel
    - Available for Linux via GitHub
    - https://github.com/pmem/nvml/
  - Defines a set of application API's for efficient use of PM hardware
    - Abstracts out OS specific dependencies
    - Underlying implementation uses memory mapped files
      - All access via API calls
    - Has its own per-file BTT implementation for atomicity guarantees
    - Works in both PM and non-PM hardware environments
  - Microsoft is working with Intel, HPE and HP Labs on a Windows port
    - Most functionality is up and running
    - We welcome anyone else that would like to contribute

Overview of NVML Libraries

- libpmemobj – transactional object store
- libpmemblk – provides arrays of atomically updated fixed size blocks
- libpmemlog – atomic append to log
- libpmem – low level support for rest of libraries
- libvmem – a volatile memory pool from a DAX mapped file

- http://pmem.io/nvml

Call to Action

- PM is an exciting new technology
- PM is a disruptive technology
- Performance tradeoffs
  - Significant storage performance improvement without application modification
  - Even better performance improvements possible with application modification
- **Windows supports PM today**
  - What are you doing to be ready?
  - Engage with your preferred OEM about their PM platform support