STORAGE INDUSTRY SUMMIT

Convergence of Storage and Memory Developing the Needed Ecosystem

JANUARY 20, 2016, SAN JOSE, CA

Tom Talpey

Microsoft

Storage Architect

Going Remote at Low Latency: a Future Networked NVM Ecosystem

SNIA SSSI | SOLID STATE STORAGE

Problem Statement

- Provide applications with remote access to Non-Volatile/Persistent Memory storage at ultra-low latency
- Examine storage protocol and RDMA protocol extensions in support of applications' workload

RDMA-Aware Storage Protocols

- Ecosystem – Enterprise / Private Cloud-capable storage protocols
  - Scalable, manageable, broadly deployed
  - Provide authentication, security (integrity AND privacy)
  - Natively support parallel and highly available operation
- SMB3 with SMB Direct
- NFS/RDMA
- iSER
- Others exist

Storage Latencies Decreasing

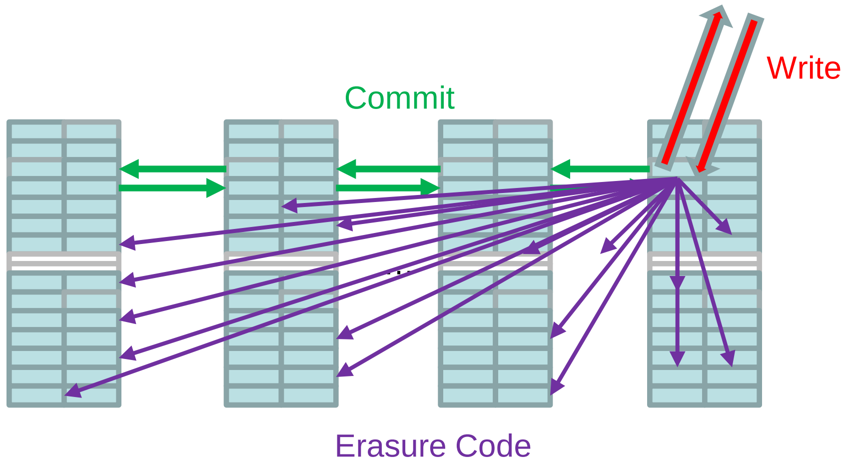
- Write latencies of storage protocols (e.g. SMB3) today down to 30-50us on RDMA
  - Good match to HDD/SSD
  - Stretch match to NVMe
  - PM, not so much ☺
- Storage workloads are traditionally highly parallel
  - Latencies are mitigated
- But workloads are changing:
  - Write replication adds a latency hop
  - Write latency critical to reduce

Technology	Latency (high)	Latency (low)	IOPS
HDD	10 msec	1 msec	100
SSD	1 msec	100 µsec	100K
NVMe	100 µsec	10 µsec (or better)	500K+
PM	< 1 µsec	(~ memory speed)	BW/size (>>1M/DIMM)

Orders of magnitude decreasing

Writes, Replication, Network

- Writes (with possible erasure coding) greatly multiplies network I/O demand
  - Small, random
    - Virtualization, Enterprise applications
  - MUST be replicated and durable
    - A single write creates multiple network writes
  - The “2-hop” issue
- All such copies must be made durable before responding
  - Therefore, latency of writes is critical!
- Reads
  - Small, random are latency sensitive
  - Large, more forgiving
    - But recovery/rebuild are interesting/important



APIs and Latency

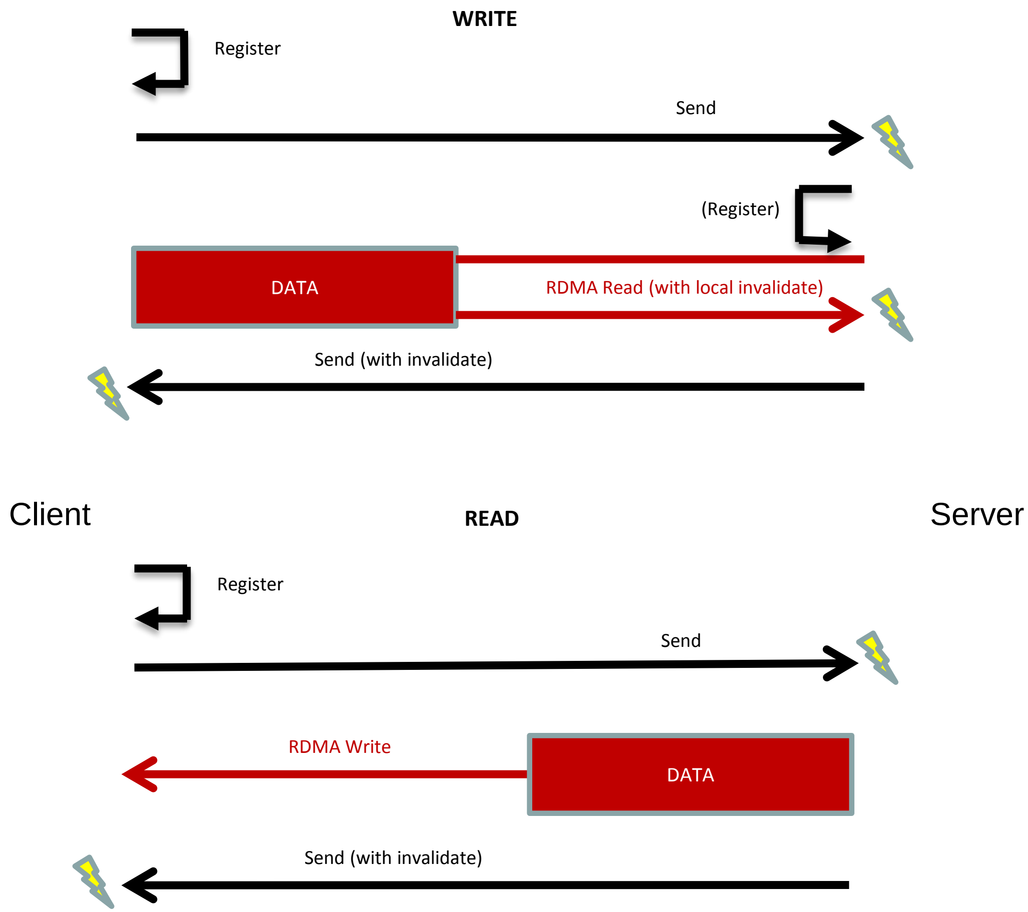
- APIs also shift the latency requirement
- Traditional Block and File are often parallel
- Memory Mapped and PM-Aware APIs not so parallel
  - Effectively a Load/Store expectation
  - Memory latency, with possibly expensive Commit
  - Local caches can improve Read (load) but not Write (store/remotely durable)

Many Layers Are Involved

- Storage layers
  - SMB3 and SMB Direct
  - NFS, pNFS and NFS/RDMA
  - iSCSI and iSER
- RDMA Layers
  - iWARP
  - RoCE, RoCEv2
  - InfiniBand
- I/O Busses
  - Storage (Filesystem, Block e.g. SCSI, SATA, SAS, …)
  - PCIe
  - Memory

RDMA Transfers – Storage Protocols Today

- Direct placement model (simplified and optimized)
  - Client advertises RDMA region in scatter/gather list
  - Server performs all RDMA
    - More secure: client does not access server's memory
    - More scalable: server does not preallocate to client
    - Faster: for parallel (typical) storage workloads
  - SMB3 uses for READ and WRITE
    - Server ensures durability
    - NFS/RDMA, iSER similar
- Interrupts and CPU on both sides



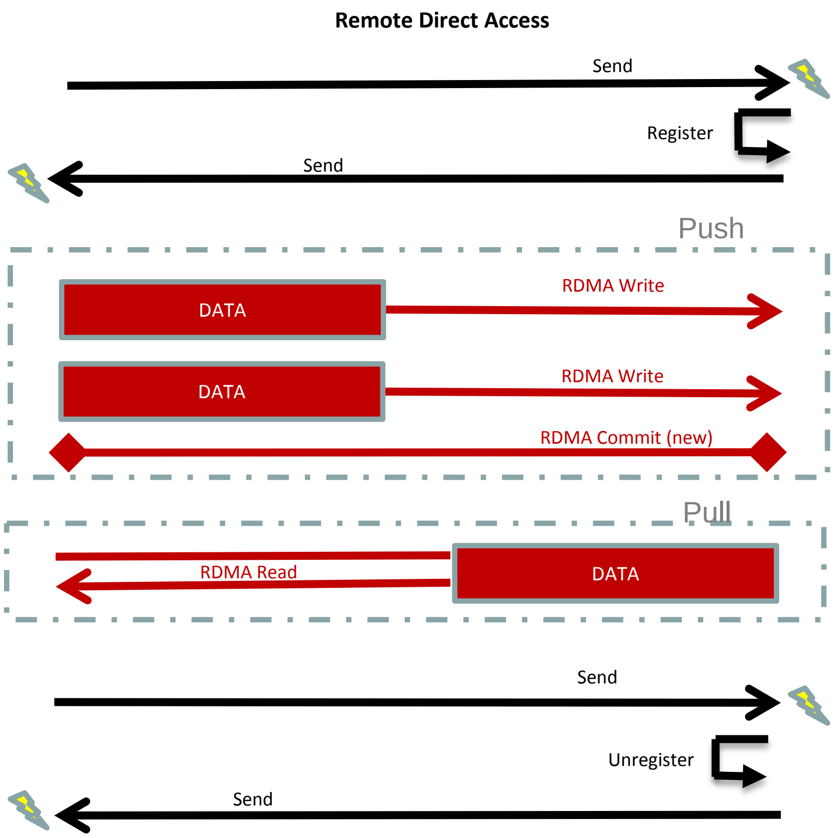
Latencies

- Undesirable latency contributions
  - Interrupts , work requests
    - Server request processing
    - Server-side RDMA handling
  - CPU processing time
    - Request processing
  - I/O stack processing and buffer management
    - To “traditional” storage subsystems
  - Data copies
- Can we reduce or remove all of the above to PM?

RDMA Push Mode (Schematic)

SNIA Solid State Storage Initiative

- Enhanced direct placement model
  - Client requests server resource of file, memory region, etc
    - MAP_REMOTE_REGION(offset, length, mode r/w)
  - Server pins/registers/advertises RDMA handle for region
  - Client performs all RDMA
    - RDMA Write to region
    - RDMA Read from region ("Pull mode")
    - No requests of server (no server CPU/interrupt)
      - Achieves near-wire latencies
  - **Client remotely commits to PM (new RDMA operation!)**
    - **Ideally, no server CPU interaction**
    - **RDMA NIC optionally signals server CPU**
    - **Operation completes at client only when remote durability is guaranteed**
- Client periodically updates server via master protocol
  - E.g. file change, timestamps, other metadata
- Server can call back to client
  - To recall, revoke, manage resources, etc
- Client signals server (closes) when done



Storage Layers Push Mode (hypothetical)

- SMB3 (hypothetical)
  - Setup – a new create context or FSCTL, registers and takes a lease
  - Write, Read – direct RDMA access by client
  - Commit – Client requests durability, SMB2_FLUSH-like processing
  - Callback – Server manages client access, similar to oplock/lease break
  - Finish – Client access complete, close or lease return
- NFSv4/RDMA (hypothetical)
  - Setup – new NFSv4.x Operation, registers and offers delegation (or pNFS layout)
  - Write, Read – direct RDMA access by client
  - Commit – Client requests durability, NFS4_COMMIT-like processing
  - Callback – via backchannel, Similar to current delegation or layout recall
  - Finish – close or delegreturn/layoutreturn
- iSER (very hypothetical)
  - Setup – a new iSER operation – registers and advertises buffers
  - Write – a new Unsolicited SCSI-In operation
    - Implement RDMA Write within initiator to target buffer
      - No Target R2T processing
  - Read – a new Unsolicited SCSI-Out operation
    - Implement RDMA Read within initiator from target buffer
      - No Target R2T processing
  - Commit – a new iSER / modified iSCSI operation
    - Perform Commit, via RDMA with optional Target processing
    - Leverage FUA processing for signaling if needed/desired
  - Callback – a new SCSI Unit Attention
    - ???
  - Finish – a new iSER operation

RDMA protocols

- Need a remote guarantee of Durability
- RDMA Write alone is not sufficient for this semantic
  - Completion at sender does not mean data was placed
    - NOT that it was even sent on the wire, much less received
    - Some RNICs give stronger guarantees, but never that data was stored remotely
  - Processing at receiver means only that data was accepted
    - NOT that it was sent on the bus
    - Segments can be reordered, by the wire or the bus
    - Only an RDMA completion at receiver guarantees placement
      - And placement != commit/durable
  - No Commit operation
- Certain platform-specific guarantees can be made
  - But the remote client cannot know them
  - E.g. RDMA Read-after-RDMA Write (which won’t generally work)

RDMA protocol extension

- Two "obvious" possibilities
  - RDMA Write with placement acknowledgement
    - Advantage: simple API – set a "push bit"
    - Disadvantage: significantly changes RDMA Write semantic, data path (flow control, buffering, completion). Requires creating a "Write Ack".
    - Requires significant changes to RDMA Write hardware design
      - And also to initiator work request model (flow controlled RDMA Writes would block the send work queue)
    - Undesirable
  - RDMA "Commit"
    - New operation, flow controlled/acknowledged like RDMA Read or Atomic
    - Disadvantage: new operation
    - Advantage: simple API – "flush", operates on one or more regions (allows batching), preserves existing RDMA Write semantic (minimizing RNIC implementation change)
    - Desirable

RDMA Commit (concept)

- RDMA Commit
  - New wire operation
  - Implementable in iWARP and IB/RoCE
- Initiating RNIC provides region list, other commit parameters
  - Under control of local API at client/initiator
- Receiving RNIC queues operation to proceed in-order
  - Like RDMA Read or Atomic processing currently
  - Subject to flow control and ordering
- RNIC pushes pending writes to targeted regions
  - Alternatively, NIC may simply opt to push all writes
- RNIC performs PM commit
  - Possibly interrupting CPU in current architectures
  - Future (highly desirable to avoid latency) perform via PCIe
- RNIC responds when durability is assured

Other RDMA Commit Semantics

- Desirable to include other semantics with Commit:
  - Atomically-placed data-after-commit
    - E.g. “log pointer update”
  - Immediate data
    - E.g. to signal upper layer
  - Entire message
    - For more complex signaling
    - Can be ordinary send/receive, only with new specific ordering requirements
- Decisions will be workload-dependent
  - Small log-write scenario will always commit
  - Bulk data movement will permit batching

Local RDMA API extensions (concept)

- New platform-specific attributes to RDMA registration
  - To allow them to be processed at the server *only*
  - No client knowledge – ensures future interop
- New local PM memory registration
  - Register(region[], **PMType**, **mode**) -> Handle
    - **PMType** includes type of PM
      - i.e. plain RAM, or “commit required”, or PCIe-resident, or any other local platform-specific processing
    - **Mode** includes disposition of data
      - Read and/or write
      - Cacheable after operation (needed by CPU on data sink)
    - Handle is processed in receiving NIC under control of original Mode

Local RDMA API Extensions

- Transparency is possible when upper layer provides Completions (e.g. messages or immediate data)
  - Commit to durability can be piggybacked by data sink upon signaling
  - Upper layer may not need to explicitly Commit
  - Dependent on upper layer and workload
- Can apply to RDMA Write with Immediate
- Or … ordinary receives
  - Ordering of operations is critical:
    - Such RDMA Writes cannot be allowed to "pass" durability
  - Therefore, protocol implications exist

Platform-specific Extensions

- PCI extension to support Commit
  - Allow NIC to provide durability directly and efficiently
  - To Memory, CPU, PCI Root, PM device, PCIe device, …
  - Avoids CPU interaction
  - Supports strong data consistency model
- Performs equivalent of:
  - CLFLUSHOPT (region list)
  - PCOMMIT
- Or if NIC is on memory bus or within CPU complex…
  - Other possibilities exist

Latencies (expectations)

- Single-digit microsecond remote Write+Commit
  - Push mode minimal write latencies (2-3us + data wire time)
  - Commit time NIC-managed and platform+payload dependent
- Remote Read also possible
  - Roughly same latency as write, but without commit
- No server interrupt
  - Once RDMA and PCIe extensions in place
- Single client interrupt
  - Moderation and batching can reduce further when pipelining
- Deep parallelism with Multichannel and flow control management

Open questions

- Getting to the right semantic?
  - Discussion in multiple standards groups (PCI, RDMA, Storage, …)
  - How to coordinate these discussions?
  - Implementation in hardware ecosystem
  - Drive consensus from upper layers down to lower layers!
- What about new API semantics?
  - Does NVML add new requirements?
  - What about PM-aware filesystems (DAX/DAS)?
- Other semantics – or are they Upper Layer issues?
  - Authentication?
  - Integrity/Encryption?
  - Virtualization?