Cache Coherency & Memory Model in RDMA-Backed Software-Coherent DSM

Zhengyi Chen

January 31, 2024

Table of Contents

1. Overview
2. Design
3. Progress
4. Future Work

1. Overview

- DSM used to be constrained by NIC bandwidth & transfer rate (e.g., during the 1990s).
- The advent of high(er) transfer rate NICs allows the DSM idea to be revived.
- Orthogonally, hardware acceleration resources are scarce and highly valuable.
  - Traditional Scheduling Mechanisms within a Cluster cannot dynamically allocate hardware accelerators without high overhead.
- Ideally, via high-speed NICs, hardware accelerator could be statically allocated such that:
  - Every node have access to the hardware accelerator node in a time-shared fashion.
  - Accelerator-attached node can access remote memory much like attaching accelerator over, say, PCIe.

Heterogeneous Memory Management

- **HMM** facilitates shared address space and transparent data migration between CPU and peripherals. Specifically:
  - HMM provides interface for duplicating the CPU page table with that of the device's, which are transparently synchronized.
  - It also provides corresponding `struct page` representation of device memory pages, which are faulted between the CPU and device.
- Theoretically, this should allow for devices in remote nodes to perform HMM using the DMA-capable NIC as a "proxy HMM device".
- Details of implementation of DSM-over-HMM is beyond this thesis's scope.
  - This thesis focuses on studying and implementing cache coherency and later, memory model for the DSM subsystem of this wider, ongoing project.

Cache Coherency, and Why It Matters Here

- Cache-incoherent RDMA (e.g., mlx) performs DMA without synchronization with CPU cache.
- We cannot assume MMU to magically maintain coherence.
  - This seems the case for x86_64 (cache-coherent DMA), but not ARM64.
- At transportation time:
  - Send to remote: flushes cache into memory before posting send message.
  - Receive from remote: invalidate cache entry after worked recv message.
- Example: Linux kernel tree, *smbdirect* implementation.
  - *smbdirect* opportunistically establish SMB over RDMA-capable network.
  - `smbd_post_send` cleans cache entry prior to posting send request.
  - `recv_done` invalidates cache entry after exiting softirq for recv request (as callback from RDMA driver).

Consistency Model and Protocol

- Majority of DSM literatures apply **release consistency** as the system's memory model.
- With **single-writer** protocol, however, the memory model can be strengthened with little increase in code complexity.
  - *DSPM*[1], for example, achieves a *de-facto* TSO consistency from its multi-writer release consistency counterpart – assuming correct memory barriers within each node's CPU, distributed writes are never reordered, and distributed reads can overtake writes.
  - Consequently, one can easily achieve sequential consistency by designating the entire write-access duration as a critical section.
- HMM's "CPU-or-device" data migration model also strongly implies a single-writer consistency protocol.

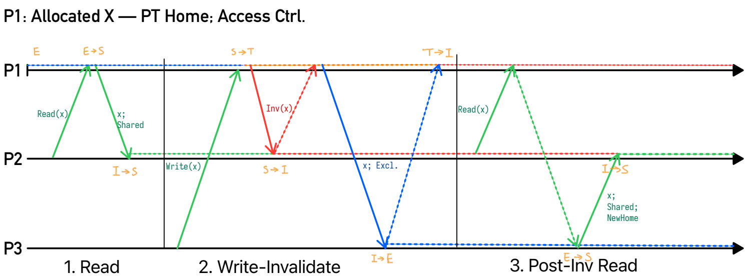
2. Design

- Designing a DSM necessitates designing:
  - Consistency Model.
  - Coherence Protocol and State Machine.
  - Access Control.
- Care needs to be taken to ensure that the in-kernel implementation is:
  - Correct,
  - Performant,
  - Exploits RDMA's traits.

Protocol Overview

- Multiple readers can exist for a clean page – the page is **shared**.
- Only one writer is allowed for a clean page – the page becomes **exclusive**.
- For one writer node be allowed sole write access to some page, all other sharers need to have their page cache invalidated prior to making the change global (commit-invalidate).
- While the sole writer node has not yet committed, either:
  - no other reader or writer nodes are allowed to be served this page (stronger consistency model).
  - no writers are allowed to be served this page. Readers can be served stale data (provided data providers do not receive invalidation message prior to service).
- When the sole writer commits, it becomes the sole home node (data provider) which serves the updated page content.
  - Optionally, some nodes can register to have commits written back instead.

Protocol Excerpt: Write-Invalidate



The T-state indicates a transitionary state for some shared page.

Consistency Model: TSO

- Total Store Ordering allows Reads to overtake Stores.
- Assuming correct use of node-local synchronization on all nodes, applying TSO in a home-based DSM allows for:
    - Another node tries to read T-page from access-control node, served stale data: W→R violation.
    - Another node tries to read S-page from data-provider nodes, served stale data: W→R violation (if e.g., the invalidation message from access-control node was received afterwards).
    - Data-provider and access-control nodes work on one request at a time: no R→W violation.
    - Write-accesses serialized at access-control node: no W→W violation.

Consistency Model: Strengthen to Sequential

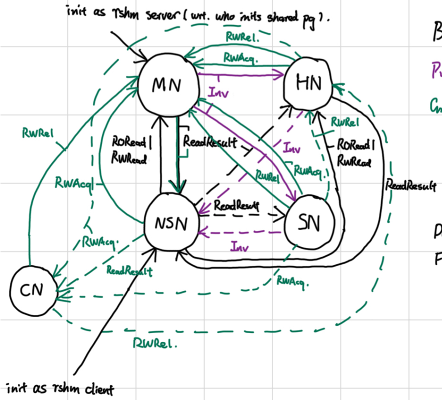
- By corollary, can reverse the previous page's statements to strengthen to sequential consistency:
  - Disallow T-pages from being serviced until new page content is installed: lengthens critical section.
  - Abolish data-provider nodes: access-control nodes become bottleneck.

Coherence Protocol: Possible Features

- Multi-data-provider Protocol: Instead of having one data-provider, have multiple data-provider nodes that are automatically write-back to prevent network bottleneck.
  - Data provider nodes may be dynamically assigned.
  - Extra metadata can limit scalability.
- Auto-share: likewise, write-back pages to non-data-provider nodes, which takes advantage of 1-sided communications provided by RDMA.
- Request aggregation: aggregate RDMA transfers for optimal transfer performance.
  - Need to be coherent with program sequence!
  - Enables write-request merging.

Stateful Nodes & Transitions (Provisional)

- Nodes (e.g., within the cluster) become tightly bound with the properties of each shared page(s).



Black — Non-committal sharing path.

Purple — Invalidation path.

Green — Committal sharing path.

Dashed transitions are internal, happens @ reception.

Filled transitions are messages.

*. Cache entries can be Valid / Invalid.

Stateful Nodes & Transitions (Provisional) (Cont.)

- MN (Manager Nodes): Provide access-control and (fallback) data-provision.
- HN (Home Nodes): Provide data-provision. Can be write-back or write-invalidate.
- SN (Sharer Nodes): Share data within a reader-only "epoch". Can be write-back or write-invalidate.
- NSN (Non-sharer Nodes): Nodes in network without sharing the particular page(s).
- CN (Commit Node): Node that acquired the single-writer access to the shared page.
- Problem: Message variants are not finalized:
  - Goal: Composable message chains that allow for "piggy-backing" of multiple procedures.

Stateful Nodes: Transition Paths

- Filled line transitions indicate local requests remote to perform state transition.
- Dashed line transitions indicate local implicitly transitions prior to sending request to remote.
- *Non-committal* path concerns about read-only and copy-on-write sharing. Sharers cannot make global modification to cached local data.
- *Invalidation* path is duo with commit operations (due to write-invalidation).
- *Committal* path concerns about global write sharing. Only one writer is allowed to write and commit at one time.
- Problem: How exactly to integrate RDMA remote read/write into this?

3. Progress

- Goal: in-kernel implementation of software cache-coherency via non-coherent RDMA hardware.
- Optimistic Goal: in-kernel implementation of memory model in DSM.
- Progress: studied and isolated mechanism for data cache invalidation/flushing in ARM64, which allows the DSM to run in heterogeneous ISA clusters.
- Integration with kernel & main DSM kernel module remains at hand: is it absolutely necessary to export new symbols for such an important operation?
- Repository: `https://github.com/rubberhead/unnamed_ba_thesis.git`.

On-demand Coherency in ARM64

- ARMv8 defines two levels of cache coherence:
    - *Point-of-Unification*: Within a core, instruction cache, data cache, and TLB all agree in the copy seen for a particular address.
        - Notably, changing PTE requires PoU.
    - *Point-of-Coherence*: Between all DMA-capable peripherals (CPU or otherwise), they all agree in the copy seen for a particular address.

  For this thesis's purposes, strive for PoC.
- Operations to achieve the latter are encapsulated in the Linux kernel as `(d|i)cache_(clean|inval)_poc`.
    - Declared under `arch/arm64/include/asm/cacheflush.h`.
    - Defined in `arch/arm64/mm/cache.S`.
    - Takes virtual address wrt. *current* address space to writeback/invalidate cache entries.
    - Problem: Can only be called in process context (for user virtual addresses) or in all contexts (for kernel virtual addresses)?

Kernel Patch for On-demand Coherency

- Problem: These symbols are not exported – not intended for driver use.
- Temporary solution: re-export them via patching the kernel.
    - Note: Kernel version v6.7.0
    - Longish-term solution: arrange kernel module code in a way that takes advantage of existing driver API (e.g., via DMA API, which for example *smbdirect* uses).
- Implements wrapper function `__dcache_clean_poc` to re-export `dcache_clean_poc` into driver namespace.
- Exports symbol into separate header file.
    - Declared in `arch/arm64/include/asm/cacheflush_extra.h`.
    - Defined in `arch/arm64/mm/flush.c`.

Proof-of-Concept Kernel Module

- Dynamically allocates GFP_USER pages and remaps to userspace on mmap.
  - GFP_USER so (for convenience) pages can be directly addressable in kernelspace (via kernel page table).
  - Pages are lazily allocated and shared between multiple processes (i.e., user address spaces).
  - Exposed as character device /dev/my_shmem.
- Around 300+ LoC.
- Problem: flawed premise for testing cache writeback!
  - Summary: CPU datapath differs from DMA datapath, common cache coherency maintenance operations are already performed in common file/virtual memory area operation code.
  - Idea: perform cache write-back on vm_ops->close.
  - Reality: virtual memory area already cleaned from cache and removed from address space prior to calling vm_ops->close.
  - Fix: Implement custom ioctl?

4. Future Work

- TBD:
  1. Incorporate cache coherence mechanism into the larger project.
  2. Implement memory model within the larger project. This involves:
     - Making adjustment to message type and structure specifications for better inter-operation with RDMA.
     - Implement memory model programmatically.
- Further Studies:
  1. Swappable RDMA memory region.
     - As of now, all DMA pages are non-swappable – they must be allocated using the SLAB/SLUB allocator for kernel memory, or via GFP page allocators.
  2. Automatic frequent sharer detection for MUX-ing between commit-invalidation and commit-writeback.

References

[1] Yizhou Shan, Shin-Yeh Tsai, and Yiying Zhang. "Distributed shared persistent memory". In: *Proceedings of the 2017 Symposium on Cloud Computing*. 2017, pp. 323–337.