Progress Report: Page Cache Consistency Model

Zhengyi Chen

December 12, 2023

Literature Review: (Shan, Tsai, & Zhang. 2017[1])

- Concerns with the sharing of persistent memory –
  - More or less similar to sharing regular memory, but…
  - Data replication is key ⇒ Multiple data provider.
- Supports both Multi-Writer Multi-Reader and Multi-Writer Single-Writer Protocols
  - MRMW "support(s) great parallelism"
  - MRSW enables "stronger consistency"
- Makes distinction between 3 variants of nodes:
  - Commit Node – Node who wishes to commit changes wrt. the system.
  - Owner Node – Node(s) who act as data provider for latest page content.
  - Manager Node – Node who provide (serialized) write access control to page.

[1] Shan, Tsai, and Zhang, "Distributed shared persistent memory".

Literature Review: (Shan, Tsai, & Zhang. 2017[2])

- For data replication and fault tolerance, necessitates:
  1. Commit status logging (akin to journaled file system)
  2. Persistent Commit ID
  3. **Required** deg. of replication – each ON shares to N nodes.
- Fault tolerance is out of this thesis's scope. However...
  - Prob. no need for requiring any degree of data replication.
  - Dropping data replication req. $\Rightarrow$ no need for replication comms.
  - Commit status logging & persistent CID can be helpful & should not introduce additional comms.
- MRSW provides "simpler and more efficient" commits than MRMW – no concurrent commits to same shared memory object exists.
  - Also makes more sense from a CPU-accelerator dichotomy outlook (ofc. wrt. this thesis's system).

[2] Shan, Tsai, and Zhang, "Distributed shared persistent memory".

MRSW: (Shan, Tsai, & Zhang. 2017[3])

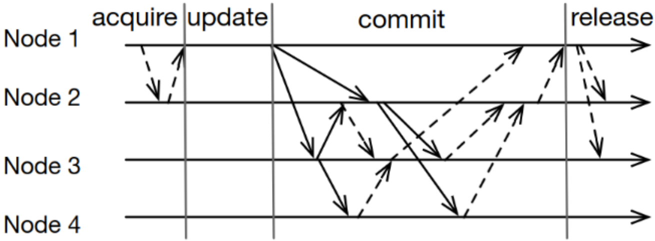


Figure 8: MRSW Example. *Node 1 (CN) first acquires write permission from Node 2 (MN) before writing data. It then commits the new data to ONs at Node 2 and 3 with replication degree four and finally releases the write permission to MN.*

Note: CN: Node 1, MN: Node 2, ON: Node 2 & 3. Node 4 may or may not already share the committed page prior to acquire.

[3] Shan, Tsai, and Zhang, "Distributed shared persistent memory".

Literature Review: (Ramesh. 2023)

- Popcorn-derived.
- Sequential consistency, MRSW protocol offloaded onto sNIC:
  - DSM protocol processor implemented on sNIC FPGA core.
  - sNIC **keeps track of memory ownership, status, R/W permissions** at page level granularity.
  - Removes the need for distinct memory management nodes.
  - (i.e., the sNIC IS the memory management node – except of course allocation).
- Similar idea occurred in *Concordia*[4]:
  - Concurrency control and multicast offloaded to network switch.
  - Authors claim this is more scalable (?)

5

---

[4] Wang et al., "Concordia: Distributed shared memory with {In-Network} cache coherence".

[5] Ramesh., "SNIC-DSM: SmartNIC based DSM Infrastructure for Heterogeneous-ISA Machines"

Literature Review: (Endo, Sato, & Taura. 2020)[6]

- Eager Release Consistency.
- Prob. using MSI coherence protocol? Authors did not mention it.
- MRMW: use timestamps to store reader "intervals".
- Introduces the home-migration concept:
  - At commit, make the CN the home node instead of invalidating the home node.
  - This removes communications needed for diff-merging at home node – this can be done locally.
  - No support for multiple home nodes.
- No performance improvement over PGAS programming framework (OpenMPI).

---

[6] Endo, Sato, and Taura, "MENPS: A Decentralized Distributed Shared Memory Exploiting RDMA".

Literature Review: (Endo, Sato, & Taura. 2020)[7]

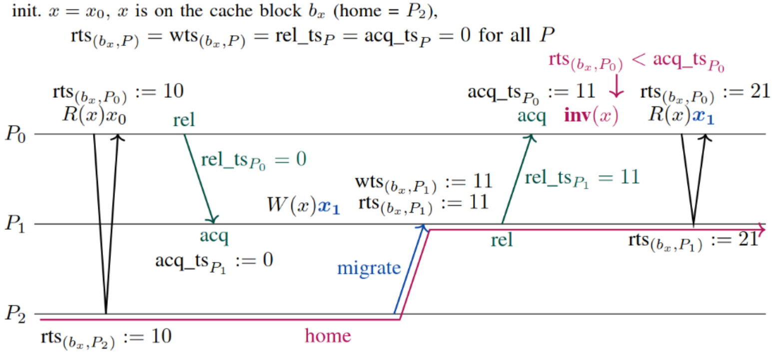


Fig. 5: Example of logical lease-based invalidation. The lease value is set to 10. P_1 only sends the logical timestamp value (rel_ts = 11) in the synchronization with P_0, and then P_0 invalidates x because the read timestamp (= 10) is smaller than acq_ts.

[7] Endo, Sato, and Taura, "MENPS: A Decentralized Distributed Shared Memory Exploiting RDMA".

The System

- Remote node(s) abstracted as shared memory device "`/dev/rshm`"
- Heterogeneous Memory Management (HMM) ensures unified address space between local and device memory.
- Migration of pages between CPU and "device" is transparent to userspace – no need for copying/mapping.
- In reality, "`/dev/rshm`" a handler for RDMA access between nodes.
  - This involves remote read/write and moving page content between nodes.
  - Local node serves as *home node & address space host* at share time.
  - Remote nodes attached on `/dev/rshm` as accelerator.

The Problem: Consistency Protocol

- Single-Writer, Multiple-Reader Protocol
- Need to be performant. . . with some ergonomics
- Two Hypothetical Protocols:
  - "RwLock" Consistency Protocol
  - Acq-Rel Consistency Protocol
- Former ensures *strong* single-writer consistency
  - – Also easier to program with!
- Latter allows concurrent in-memory *non-committal* computation

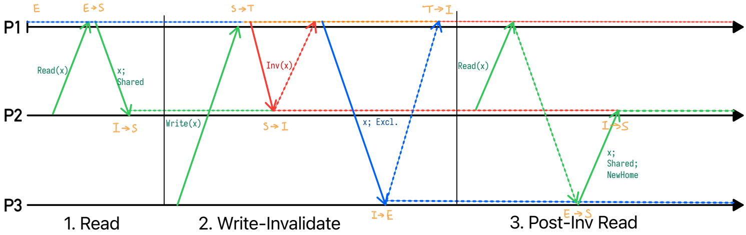
“RwLock” Consistency Protocol

Similar to a read-write lock where:

- 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 readers need to have their page cache invalidated.
- While the sole writer node has not yet committed, no other reader or writer nodes are allowed to be served this page.
- When the sole writer commits, it becomes the new home node which serves the updated page content.
- Invalidates reader must fetch from MN for read access, which maintains RAW ordering.

“RwLock” Consistency Protocol

P1: Allocated X — PT Home; Access Ctrl.



Acq-Rel Consistency Protocol

In RwLock's case, read requests result in installation of read-only pages at remote nodes.
Alternatively, this protocol allows read/write pages to be installed at remote nodes at read time. Such writes are *non-committal* and cannot be synced with the entire system.
To summarize:

- "Readers" can write to its locally installed page without any means to synchronize the change.
- "Writers" need to acquire global write access from the *PT node*, which invalidates all shared pages.
- i.e., Instead of write-invalidate, perform acquire-invalidate.

This may require pages to be marked as CoW if the sharer wants also to act as a home node.

Consistency Protocol: Knobs and Mods

We can modify these two protocols further as follows:

- Multi-home Protocol: instead of having one home at a time, have multiple homes (e.g., when writer commits) to prevent network bottleneck.
  - Extra metadata can limit scalability (e.g., granularity of directories)
- Auto-share: Automatically share pages at commit time using 1-way communications.
  - Potential for communication reduction – debatable.
- Request aggregation: Aggregate RDMA requests for optimal RDMA transfer performance.
  - Need to be coherent with program sequence!

Why this design?

- Largely inspired by DSPM[8].
- Removed arrows for enforced data duplication – duplication is solely on-demand.
- Introduces transitional state "T":
  - Used to flag a page as unserviceable – visible only at MN.
  - All read/write access to T-page is kept on hold until MN receives commit msg.
  - After commit, MN forwards queued R/W access to moved home.
  - This (at least) maintains RAW, WAW data dependency for whichever interleaving issues.
  - Removing T allows stale data to be served – violates RAW for better throughput.
- Extensible (as mentioned in prior page).

[8] Shan, Tsai, and Zhang, "Distributed shared persistent memory".

Why not this design?

At the very least...

- De-coupled home and access-management nodes require:
  - Each home node need to be MN-aware (easy).
  - MN need to be home-aware (also easy with single-writer, but spatial complexity is a concern):
    - Naive directory scheme is not scalable.
    - Coarse directory scheme (e.g., SGI Origin 2000) is wasteful (but may be the fastest in practice).
    - Distributed directory scheme may provide terrible latency.
    - More sophisticated schemes are possible but needs work & experimentation.
- Strict consistency limits throughput.

What about Consistency **Model**?

- The weaker a consistency model is, the more difficult it is to program with.
  - Weak ordering architectures (e.g., ARMv8) more or less depends on compiler/interpreter to emit barriers as see fit[9].
  - Bad for usability/portability – programs may need to be compiled using a modified toolchain, else need to add these synchronization instructions/function calls everywhere.
- [10] uses Partial Store Order.
  - Preserves RAR, WAR – "synchronous read...asynchronous write"
  - Easier to use than relaxed ordering.
- [11] uses strong consistency, but warns about its scalability.

[9] Haynes, *Sequential consistency in armv8*.

[10] Cai et al., "Efficient distributed memory management with RDMA and caching".

[11] Wang et al., "Concordia: Distributed shared memory with {In-Network} cache coherence".

Consistency Model: Cont.

- Similar to Concordia[12], the proposed protocols also assume strong consistency.
- Further work needed to see how to adapt these protocols for weaker consistency models.
  - Low-hanging fruit: TSO
  - Allowing read requests to be served for T-pages @ MN: W$\rightarrow$R violation.
  - Allowing read requests to be served via non-MN homes: also W$\rightarrow$R violation (exploits a race condition between write msg and invalidation msg).
  - Request workers work on one request at a time: no R$\rightarrow$W violation.
  - W$\rightarrow$W violation simply cannot happen – they always serialize @ MN.

[12] Wang et al., "Concordia: Distributed shared memory with {In-Network} cache coherence".

Summary

- Based on MSI coherence protocol, with possible T-state extension.
  - T-state can be instead implemented as an additional flag parallel to MSI FSM.
  - T-pages cannot be serviced by MN – all read/write requests blocked.
- One consistency model (for now): sequential consistency.
  - Maintains RAW via T-state @ MN – removing blocking on T-pages results in TSO.
  - Maintains WAR via sequentially worked RDMA RQ.
  - Maintains WAW via single-writer.
- Two consistency protocols:
  - RwLock consistency protocol only allows read-only sharing.
  - Acq-Rel consistency protocol differentiates non-committal writes, allows proc-local writable sharing.