Boosting Persistence Parallelism in Memory Bus and RDMA Network

***Abstract*—Emerging non-volatile memories (NVMs) incorporate the features of fast byte-addressability and data persistence, which are beneficial for data services, such as file systems and databases. To support data persistence, a persistent memory system requires ordering for write requests. We observe that the memory bus and the Remote Direct Memory Access (RDMA) network are severely under-utilized during the process of ensuring data persistence, because the persistence parallelism in the memory bus and the RDMA network is not fully leveraged during ordering.**

To address this problem, we propose an architecture design to exploit the persistence parallelism in the memory bus and the RDMA network. First, we utilize *inter-thread persistence parallelism* for barrier epoch management with better bank level parallelism (BLP). Second, we enable *intra-thread persistence parallelism* with multiple barrier epochs inside one network packet, and propose a fine-grained memory persistence scheme through the network. With these features, our architecture is capable of efficiently supporting transaction-level persistence through RDMA network for file systems or NVM libraries. The experimental results show that our work can achieve 1.3× performance improvement for local applications and about 1.93× for remote applications serviced through the RDMA network.

I. Introduction

Emerging non-volatile memory (NVM) technologies show great promise as replacements for DRAM main memory. Intel has announced 3D XPoint production for both storage and memory in 2016. Compared with DRAM technology, NVMs incorporate the features of fast byte-addressability and disk-like data persistence in addition to a superior storage density. However, ensuring recoverability of persistent data in data services, regardless of power failure or program crash, is non-trivial. Previous work adopts both software versioning mechanisms and hardware persistence ordering capability to achieve this goal [4]. The correctness of versioning mechanisms relies on the hardware's capability to maintain the persistence ordering exactly in the way that software defines it. Specifically, the requests must be persisted before after-barrier requests [4].

Persistent requests go through the memory hierarchy with ordering control, following a datapath (as shown in Figure 1) that can be divided into three segments: (1) *through the cache hierarchy to the memory controller*; (2) *through the bus from the memory controller to memory devices*; (3) *through the Remote Direct Memory Access (RDMA) memory network from a remote node to a local node*.

For the first segment of the datapath, previous work propose buffered epoch persistence methods to alleviate the persistence ordering overhead in the cache hierarchy [4], [5]. The latter two datapath segments, however, are observed to be severely underutilized during data persistence.

Inefficiencies in the Memory Bus: The relaxed memory model provides the **persistence parallelism** in the cache hierarchy which enables multiple request epochs (a request sequence divided by barriers) flowing through the cache hierarchy to the memory controller. However, prior work aims to alleviate persistent barrier constraints with a larger epoch size without considering bank locations of requests [8], [4], [5]. Therefore, the memory request epochs sent to the memory controller has low bank-level parallelism (BLP) in prior work, while low BLP may lead to bad memory scheduling efficiency [11].

Inefficiencies in the RDMA Network: With the emergence of NVM technologies, remote NVM systems based on RDMA protocol have been deployed by many storage systems [3], [6], [10]. These systems rely on memory persistence through the network, which is referred to as **network persistence**. The Storage Networking Industry Association (SNIA) has proposed an ordering solution for network persistence that requires an RDMA read after RDMA writes (RDMA-RAW) for every barrier epoch [2]. In this solution, transactions with many epochs need multiple RDMA round trips for data persistence. Hence, it is extremely inefficient for transactions with multiple epoches using such a coarse-grained network persistence, without intra-thread persistence parallelism.

To maximize the persistence parallelism for better memory bus and network utilization, we distinguish the differences among these three segments of datapaths. In Path ① and Path ③, it is more efficient to record rather than implement the ordering constraints. In Path ②, we should maintain the ordering constraints for the system correctness. Therefore, we leverage *intra-thread persistence parallelism* in the Path ① and Path ③ by coalescing more persistent requests. Then we leverage *BLP-aware inter-thread parallelism* to ensure persistent ordering with better memory throughput, in the Path ②.

- We propose an architectural design to improve persistent parallelism in memory bus and RDMA network without violating ordering constraints.
- We propose a BLP-aware barrier epoch management technique to offer higher bank-level parallelism for memory scheduling.
- For the first time, we propose a fine-grained memory persistence scheme through the network, enabled by intra-thread persistence parallelism with multiple barrier epochs inside one network packet. Fine-grained memory persistence efficiently supports transaction-level persistence in the memory network.

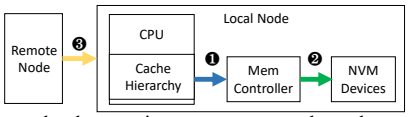
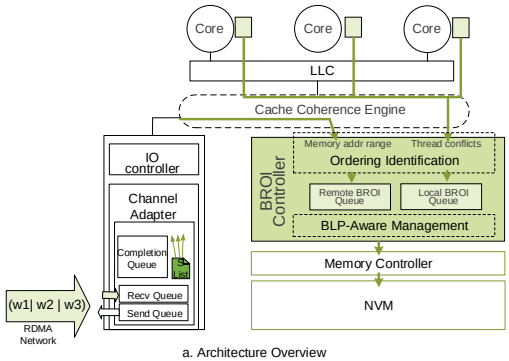


Fig. 1. The datapaths that persistent requests go through.

II. Architecture Overview

We propose an architecture design which is enhanced with a **barrier region of interest (BROI) controller** to optimize barrier epoch management, aiming to address the inefficiencies in persistent datapaths. The architecture overview is shown in Figure 2. The BROI controller has following functions: 1) barrier epoch identification for ordering detection; 2) barrier epoch buffering; 3) barrier epoch management for exploiting persistence parallelism under ordering constraints.

Barrier Epoch Identification: Our work is based on buffered epoch memory model rather than synchronous model. The coherence engine maintains the inter-thread persistent ordering and the BROI controller maintains the intra-thread persistent ordering [5]. For local requests, prior work proposed the persistent dependency tracking method based on barrier recording in persistent buffers and cache hierarchy [5][8][9]. We use the same design for local request ordering. For remote requests coming through the network, however, only data, not instructions, can come through RDMA network after handshake, which raises the difficulty to indicate and identify the barrier epoch in remote requests. We propose a method to indicate ordering constraints



a. Architecture Overview

Fig. 2. Architecture overview.

through network based on scatter/gather features of RDMA [1], which allocates barrier epochs in the scatter/gather elements in sender node. Then the BROI controller at the receiver node detects the intra-thread ordering constraints of requests in terms of the memory address range, according to the data length of every element in the scatter/gather list. If the request is out of the range of the previous epoch, the BROI controller will mark in the remote BROI queues to note the barrier location. In this way, multiple barrier epoches can be transferred inside one network packet to achieve fine-grained network persistence.

Barrier Epoch Buffering: The BROI controller allocates the barrier epochs in BROI queues which store the persistent requests from local and remote processors.

Barrier Epoch Management: The BROI controller maintains the request sequence going into BROI queues from cache hierarchy and out of the BROI queues to the memory controller. This process is referred to as barrier epoch management. The strategies for barrier epoch management are as follows: 1) Persistent ordering of requests in one BROI entry must be obeyed, which can be achieved by forcing the requests after a barrier to stay in the BROI queues until all the requests before the barrier have been executed; 2) for the requests in different BROI entries with inter-thread parallelism, management should provide more BLP for the memory controller.

III. System Overview for Network Persistence

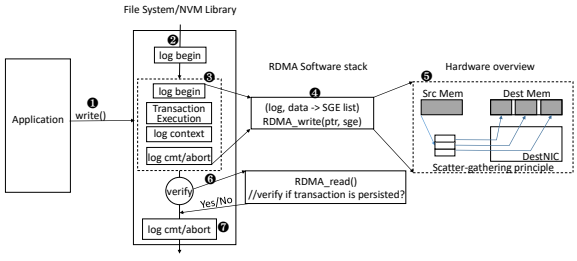


Fig. 3. An example for a transaction through network with fine-grained network persistence.

Exploiting the fine-grained network persistence needs the cooperation across the system stack. We show an example to explain the programming model and how a transaction persists in remote memory system with fine-grained network persistence, as shown in Figure 3. In these programming model, the software library benefits from the capability of transferring one transaction through the network with much less RDMA RAW operations if they organize logs and data properly in the memory system.

IV. Experiment

In this section, we show the experimental result of BROI architecture, including both the local and remote application operational throughput evaluations.

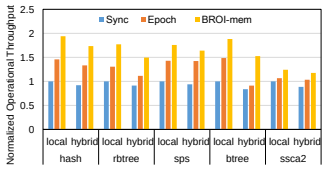


Fig. 4. OPS normalized to *Local-Sync*

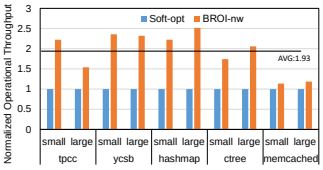


Fig. 5. OPS normalized to *Soft-opt*

Local Application Throughput. The evaluation of the application operational throughput in local datapaths is as shown in Figure 4. The y-axis shows the operational throughput (OPS, Operational transactions per second) normalized to *Sync-local* scenarios. Under scenarios with *local* and *hybrid* persistent operations, *BROI-mem* (epoch memory model with optimized barrier epoch management) can improve performance by 72% and 67% compared to *Sync* method (using synchronous model). *BROI-mem* improves performance by 28% and 30% compared to the *Epoch* model method (using buffered epoch memory model). The results show the effectiveness of BLP-aware barrier epoch management that improves application performance significantly.

Remote Application Throughput We evaluate the effectiveness of fine-grained network persistence which improves network persistence performance for remote applications. We analyze the performance of ycsb, tpcc, memcached, hashmap, ctree in whisper benchmarks[7]. Operational throughput under two scenarios are being compared: *Soft-opt* (soft optimization scheme which uses much less epoch number as the adaption for execution on remote node) and *BROI-nw* (our scheme which enables the fine-grained network persistence). The operational throughput results are normalized to *Soft-opt*, as shown in Figure 5. On average, *BROI-nw* improves performance by 1.93x. Such improvement is introduced because of that the RDMA round trips are eliminated significantly thus reduce the network persistence latency, which is important for application performance.

References

[1] Rdma aware networks programming user manual. http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf.

[2] C. Douglas. Rdma with pmem, software mechanisms for enabling access to remote persistent memory. http://www.snia.org/sites/default/files/SDC15_presentations/persistant_mem/ChetDouglas_RDMA_with_PM.pdf, 2015.

[3] N. S. Islam, M. Wasi-ur Rahman, X. Lu, and D. K. Panda. High performance design for hdfs with byte-addressability of nvm and rdma. In *ICS' 16*, pages 8:1–8:14. ACM, 2016.

[4] A. Joshi, V. Nagarajan, M. Cintra, and S. Viglas. Efficient persist barriers for multicores. In *MICRO 2015*, pages 660–671, 2015.

[5] A. Kolli, J. Rosen, S. Diestelhorst, A. Saidi, S. Pelley, S. Liu, P. M. Chen, and T. F. Wenisch. Delegated persist ordering. In *MICRO' 2016*, pages 1–13, 2016.

[6] Y. Lu, J. Shu, Y. Chen, and T. Li. Octopus: an rdma-enabled distributed persistent memory file system. In *USENIX ATC 17*, pages 773–785, 2017.

[7] S. Nalli, S. Haria, M. D. Hill, M. M. Swift, H. Volos, and K. Keeton. An analysis of persistent memory use with whisper. In *ASPLOS '17*, pages 135–148. ACM, 2017.

[8] S. Pelley, P. M. Chen, and T. F. Wenisch. Memory persistency. In *ISCA' 14*, pages 265–276, 2014.

[9] S. Pelley, P. M. Chen, and T. F. Wenisch. Memory persistency: Semantics for byte-addressable nonvolatile memory technologies. *IEEE Micro*, 35(3):125–131, May 2015.

[10] Y. Shan, S.-Y. Tsai, and Y. Zhang. Distributed shared persistent memory. In *Proceedings of the ACM Symposium on Cloud Computing*, 2017.

[11] J. Zhao, O. Mutlu, and Y. Xie. Firm: Fair and high-performance memory control for persistent memory systems. In *MICRO' 2014*, pages 153–165, 2014.