Brief Announcement: Asymmetric Mutual Exclusion for RDMA

Jacob Nelson-Slivon
Lehigh University, Betlehem, PA, USA

Lewis Tseng
Boston College, MA, USA

Roberto Palmieri
Lehigh University, Betlehem, PA, USA

Abstract

In this brief announcement, we define operation asymmetry, which captures how processes may interact with an object differently, and discuss its implications in the context of a popular network communication technology, remote direct memory access (RDMA). Then, we present a novel approach to mutual exclusion for RDMA-based distributed synchronization under operation asymmetry. Our approach avoids RDMA loopback for local processes and guarantees starvation-freedom and fairness.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Mutual exclusion, Synchronization, Remote direct memory access (RDMA)

Digital Object Identifier 10.4230/LIPIcs.DISC.2022.50

Related Version *Technical Report*: `https://arxiv.org/abs/2208.09540`

Funding This material is based upon work supported by the National Science Foundation under Grant No. CNS-2045976.

1 Introduction

In contrast to traditional message-passing, remote direct memory access (RDMA) is a popular network communication technology that directly implements the shared-memory abstraction in the distributed setting by allowing a process to access memory on a remote machine *without* interacting with another process. These operations are known as *one-sided*, since they only involve one process. In addition to reads and writes, RDMA also provides atomic read-modify-write (RMW) operations on remote memory, like compare-and-swap (CAS) and fetch-and-add (FAA). Hence, the API closely resembles that of modern shared-memory.

Also similar to modern architectures, the memory semantics of RDMA is *not* sequentially consistent. Since remote operations complete asynchronously, local and remote access to a given memory location may be reordered. Furthermore, while remote reads and writes are atomic with their local counterparts due to cache coherent I/O (e.g., Intel's DDIO), atomicity between local and remote RMW operations is not guaranteed. Without *global atomicity* support (i.e., atomicity among all local and remote operations), all processes must rely on the RDMA-capable network interface controller (RNIC) for consistency. More precisely, local processes should use the *loopback* mechanism, which allows a process to access memory on its own machine by passing through the RNIC.

In practice, both one-sided RDMA RMW and message-passing (e.g., RPCs) have their drawbacks. For the one-sided approach, RDMA loopback is still an order of magnitude slower than local accesses and introduces internal congestion [2]. While RPCs are prevalent in RDMA-based systems, in part due to the many challenges associated with synchronizing local

and remote processes, message-passing can nullify the performance benefits that one-sided RDMA provides. Thus, a primary motivation for our work is how to balance the needs of both local and remote processes in the context of one-sided RDMA.

To that end, this brief announcement introduces the concept of *operation asymmetry*, a property that captures how processes interact with memory differently, and describes a new mutual exclusion algorithm designed to capture the nuanced requirements of synchronizing local and remote processes in RDMA-enabled systems. To the best of our knowledge, we are the first to solve mutual exclusion specifically for RDMA in a manner that does *not* require RDMA loopback or message-passing. Our solution is *starvation-free* (i.e., a calling process eventually executes its critical section) and *fair* (i.e., first-come-first-served).

2 Mutual Exclusion Under Operation Asymmetry

In our system model, processes communicate by accessing *local* or *remote* shared memory, consisting of *registers*. For each class of access (local/remote), the registers in our system support three operations: read (`Read`/`rRead`), write (`Write`/`rWrite`) and compare-and-swap (`CAS`/`rCAS`). Local operations access memory natively while remote operations pass through the RNIC. An operation on a register is *enabled* for a process if the process is able to access the register using the given operation. Intuitively, local accesses are only enabled for local processes (i.e., on the same machine as the register).

We define an object as *operation asymmetric* if, given two processes, the intersection of their respective enabled operations on the object is *not equal* to their union. Under one-sided RDMA, registers are operation asymmetric since remote processes cannot perform local accesses. To demonstrate the consequences of operation asymmetry, recall that the atomicity of local and remote operations is not guaranteed. Due to this behavior, an RDMA RMW operation (e.g., `rCAS`) appears to a local process as if it were a `Read` then `Write`. Hence, local processes must utilize RDMA loopback to ensure atomicity of RMW operations with remote processes.

When designing a mutual exclusion primitive for RDMA-based systems (without global atomicity), remote RMW operations provide the necessary atomicity but RDMA loopback introduces overhead for local processes and network congestion. Therefore in our model, to avoid local processes using RDMA loopback, we restrict the set of enabled operations for local processes to *only include local operations*. However, due to operation asymmetry, any solution satisfying these constraints can only be built from the greatest common denominator: atomic read-write registers. Thus, approaches like Peterson's lock [5] are appropriate.

2.1 Algorithm Description

To implement multi-process synchronization using operation asymmetric registers, we modify the original (two-process) Peterson's lock algorithm to embed an orthogonal mutual exclusion primitive whereby local and remote processes compete amongst themselves for the right to participate in the Peterson's lock protocol. To limit the number of remote operations required for remote processes, we embed the widely used MCS queue lock [3], allowing processes to spin locally while waiting to acquire the lock. Our combination of locks is an extension of lock cohorting [1] to an RDMA-enabled distributed system, and we adopt the naming conventions by calling Peterson's lock the *global* lock and the MCS queue locks *cohort* locks. In our approach, processes with the same set of enabled operations (local or remote) compete amongst themselves using their cohort lock to determine a leader that then participates in the global protocol, relying only on process-wide atomic operations (i.e., read and write).

The original Peterson's lock algorithm has two global variables: `flag[2]`, which is a two element array of boolean values indicating interest in the critical section, and `victim`, which is an integer deciding which process yields execution. We modify the algorithm by replacing `flag` with our MCS queue cohort locks.

A process first announces interest in executing its critical section by locking the corresponding (local or remote) cohort lock, effectively raising its `flag`. If the calling process acquires the cohort lock from another member of its cohort, it may enter the critical section without additional steps. Otherwise, the process must engage in the (global) Peterson's lock protocol, by setting `victim` to its own process identifier then waiting while the other cohort lock is held and `victim` is not changed. Since Peterson's lock is constructed from atomic read-write registers, and local and remote reads and writes are atomic, local operations need only use local accesses, remote operations use one-sided RDMA, and no RDMA loopback is necessary. To unlock, a process simply releases its cohort lock, effectively lowering the `flag` variable of the original algorithm.

Each cohort lock is specifically designed for the class of processes in the cohort. That is, there is one for local processes and another for remote processes. Note that MCS queue locks are particularly well-suited for RDMA since they perform *local-spinning*, meaning that a process need not repeatedly access remote memory while waiting for the lock. To implement fairness, we alter our MCS queue algorithms to support a budget, similar to the technique used by Dice et al. [1]. Once the budget is exhausted, the detecting process is required to reacquire the global lock. If there is a waiting process of the opposite cohort, it will be allowed to proceed. Otherwise, the calling process reacquires the global lock then resets the budget. Since the global lock is released after a bounded number of cohort lock acquisitions, and the global lock is itself fair (i.e., a waiting process cannot be overtaken), our approach is fair [1]. Our technical report [4] includes more details, the pseudo-code, and a model-checked TLA+ specification of our mutual exclusion primitive.

3 Conclusion

Motivated by our definition of *operation asymmetry*, we propose a starvation-free and fair mutual exclusion mechanism for RDMA, enabling local and remote processes to synchronize while optimizing for their individual behavioral constraints. To the best of our knowledge, we present the first mutual exclusion solution that synchronizes local and remote processes while avoiding both RDMA loopback and message-passing.

References

1 David Dice, Virendra J. Marathe, and Nir Shavit. Lock Cohorting: A General Technique for Designing NUMA Locks. In *PPoPP '12*, pages 247–256, 2012. doi:10.1145/2145816.2145848.

2 Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. Collie: Finding Performance Anomalies in RDMA Subsystems. In *NSDI '22*, pages 287–305, Renton, WA, 2022.

3 John M. Mellor-Crummey and Michael L. Scott. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. *ACM Trans. Comput. Syst.*, 9(1):21–65, February 1991. doi:10.1145/103727.103729.

4 Jacob Nelson-Slivon, Lewis Tseng, and Roberto Palmieri. Technical Report: Asymmetric Mutual Exclusion for RDMA, 2022. arXiv:2208.09540.

5 Gary L. Peterson. Myths About the Mutual Exclusion Problem. *Information Processing Letters*, 12:115–116, 1981.