Brief Announcement: ROMe: Wait-free Objects for RDMA

Jacob Nelson-Slivon
jjn217@lehigh.edu
Lehigh University
Bethlehem, PA, USA

Reilly Yankovich
rdy221@lehigh.edu
Lehigh University
Bethlehem, PA, USA

Ahmed Hassan
ahh319@lehigh.edu
Lehigh University
Bethlehem, PA, USA

Roberto Palmieri
palmieri@lehigh.edu
Lehigh University
Bethlehem, PA, USA

ABSTRACT

Ensuring data consistency under remote direct memory access (RDMA) is challenging due to the combined effects of various hardware components. This brief announcement introduces remote object memory (ROMe), the first technique to guarantee wait-free consistent reads of arbitrarily sized objects over RDMA without the use of specialized hardware, and while allowing the concurrent execution of conflicting local updates. We integrated ROMe into ROMe-KV, an RDMA-enabled key-value store whose underlying B-link tree nodes are ROMe objects that enable supporting wait-free linearizable range queries.

CCS CONCEPTS

• **Information systems** → **Parallel and distributed DBMSs**; • **Software and its engineering** → **Distributed programming languages**.

KEYWORDS

Consistency, Distributed Data Structures, RDMA

ACM Reference Format:
Jacob Nelson-Slivon, Reilly Yankovich, Ahmed Hassan, and Roberto Palmieri. 2024. Brief Announcement: ROMe: Wait-free Objects for RDMA. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '24), June 17–21, 2024, Nantes, France.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3626183.3660262

1 INTRODUCTION

Remote Direct Memory Access (RDMA) is a networking technology being deployed in a growing number of computing clusters [3, 4, 6, 16, 17, 22]. It innovates over traditional network infrastructures (e.g., Ethernet) by introducing the capability for a process to read and write remote memory directly over a network (i.e., *one-sided* operations) while also bypassing the host kernel [10]. RDMA improves performance over traditional communication [6]; however, data consistency in RDMA-enabled systems involves unique challenges not present in other networking technologies [10, 13, 18].

SPAA '24, June 17–21, 2024, Nantes, France

ACM ISBN 979-8-4007-0416-1/24/06
https://doi.org/10.1145/3626183.3660262

Consistency for small RDMA objects is provided directly by hardware [1, 10, 11]. However, three drawbacks arise when considering arbitrarily large RDMA-accessible objects (ie., spanning two or more cache lines). First, RDMA primitives cannot be used for arbitrarily large objects. Second, most RDMA implementations do not support atomicity between remote and local operations, which means local processes must use *RDMA loopback* [10] to access RDMA-accessible objects safely, leading to increased network card congestion [13]. Finally, cache lines may be retrieved in parallel [18], meaning there is no guaranteed ordering between writes to memory and RDMA reads that exceed a single cache line.

Method	Max. Obj. Size	Progress	Spec. Hardware
Locking [2, 4]	∞	blocking	no
Versioning [6, 17]	∞	blocking	no
Checksums [16, 21]	∞	blocking	no
Cache coherence	cache line (e.g., 64B)	wait-free	no
Honeycomb [15]	∞	wait-free	yes
ROMe	∞	wait-free	no

Table 1: Summary of RDMA data consistency methods.

Existing solutions to the above problems are summarized in Table 1. It is important to note that all these approaches are *blocking* except for *Cache Coherence* and *Honeycomb* [15]. The former is limited to objects whose size does not exceed a cache line, and the latter requires specialized hardware.

In this brief announcement, we introduce ROMe, an abstraction to develop RDMA-aware data management systems in which reads are guaranteed to terminate after a finite number of steps (i.e., wait-freedom) regardless of concurrent writes and without limitations on the size of the accessed memory. It also guarantees that remote reads do not interfere with ongoing updates. Finally, it eliminates the need for RDMA loopback.

2 SYSTEM MODEL

Consider a system consisting of a set of RDMA-enabled nodes in a data center. A host exposes a region in main memory accessible to processes on one or more client nodes via RDMA. We define a *local* process as one that is resident on the host and interacts with its RDMA-accessible memory through the underlying memory subsystem (i.e., shared-memory APIs). Hence, in order to avoid RDMA loopback, a local process *does not* use RDMA. Otherwise, a *remote* process can communicate with a local process by sending messages or by direct access to the remotely accessible memory via RDMA.

Our design adopts a common approach that restricts remote processes to using RDMA read operations when directly accessing host memory [16]. If a remote process on node n_j wishes to update the memory on the host n_i, it sends a message to n_i to perform the operation on its behalf. This can be done using either two-sided RDMA operations [12] or a message channel built using one-sided writes [6, 7].

3 REMOTE OBJECT MEMORY

In ROMe, a local process accumulates changes to memory in an append-only log and atomically updates a reference to the head of the log, making newly written log records visible to remote processes. Remote operations can then use the log to reconstruct a consistent view of the memory. To avoid an unbounded log, the object is occasionally *compacted*, meaning that a snapshot of the current state replaces the original object using a copy-on-write mechanism [20].

An object encoded using ROMe consists of the following metadata that, when combined, amounts to the object's current state:

- The *base region* is an immutable memory region that represents the initial abstract state of memory. This portion must be written before any remote accesses are made.
- The *active region* consists of the *current log offset* (`CLO`) and other user-defined metadata that can be updated in place. To ensure consistent access with concurrent writes, this region must never span multiple cache lines, but a level of indirection can be used if needed.
- The *log region* records updates to the memory in the base region. Once written, log records are immutable.
- The *supplemental region* is a region for any additional metadata that may be required to synchronize updates but is not used for remote reads.

Our design assumes that the above regions all reside in contiguous memory.

Next, we describe how operations produce a consistent view of the above memory regions.

3.1 Writes

Updates to the object first check whether there is room in the *log region*. If there is, then a log record corresponding to the modification is written and the `CLO` is updated to point to it, with the appropriate memory fences between them. Since the write to the `CLO` is cache coherent, readers (local and remote) are guaranteed to observe the log record.

3.2 Reads

Remote operations utilize RDMA while local reads avoid RDMA entirely, which is a unique characteristic of our design compared to state-of-the-art solutions. Both reads follow the same steps. First, the operation reads the *active region*. Using the `CLO` obtained from this initial read, the operation then reads the remaining regions and reconstructs the current state of the memory corresponding to the observed `CLO`. Since the `CLO` is only updated after a log record is written, both local and remote reads of the memory are linearized at the moment they read the `CLO`. All previous logs are guaranteed to be written and immutable, and future log records will be ignored. Regardless of size, remote reads require exactly two RDMA read operations, which is effectively the overhead to implement wait-freedom.

3.3 Compaction

To avoid an unbounded log, we assign a log capacity and clean up the memory whenever the log becomes full. Similar to LSM-tree databases (e.g., [8]), we call this step *compaction*. During compaction, the most recent log records are reconciled with the base region to generate the base region of a new object, applying the copy-on-write [20] strategy. Once the newly allocated memory is initialized, the system then coordinates, making it visible to remote readers. Utilizing copy-on-write is an important design choice that enables wait-free read operations.

4 EVALUATION

To demonstrate the effectiveness of ROMe, we develop ROMe-KV, a range-partitioned, sharded B-link tree [9, 14] that supports *wait free linearizable remote range queries* by encoding its data structure nodes as ROMe objects. For ROMe-KV, our ROMe nodes are structured as:

(1) The *base region* of each node contains the initial state of the node, consisting of possibly many key-value pairs or child pointers, along with metadata used during traversals: its version at initialization, the minimum and fence keys, and a pointer to its right sibling.
(2) The *log region* holds an array of log records containing updates to the *base region*.
(3) The *active region* contains only the `CLO`, as it is the only metadata updated in place that is required by remote operations.
(4) The *supplemental region* includes a lock, a pointer to the node's left sibling, the version of the most recent update, and a logical deletion flag.

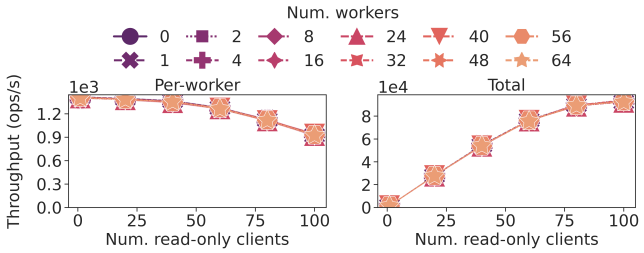
Our evaluation utilizes the CloudLab testbed [19] and we run our experiments on an 11-node cluster of `r6525` nodes, which consist of two 32-core AMD 7543, running at 2.8GHz, 256GB of memory and are equipped with a dual-port Mellanox ConnectX-6 100 Gbps RNIC. Our implementations are written in C++ and compiled using Clang 12 with `-O3` optimization and `-std=c++20`. We pin cores to the same NUMA zone as the RNIC to control for NUMA effects. Our workload uses 8B keys and 8B values. In practice, the 8B values can be pointers to larger values.

In our experiment, we analyze the scalability of remote read-only clients in the presence of a local workload on a single server node using the YCSB B workload [5], representing a workload consisting of 95% reads and 5% writes. We use a Zipfian distribution, with a skew of $\theta = 0.99$. In addition to regular YCSB, we also run concurrent *read-only* clients, which execute a workload consisting of scans whose start keys are sampled from the same Zipfian distribution as the YCSB workload, and whose length is uniformly generated between 5000-10000 elements. This workload represents an analytical background task that should not interfere with request handling. Read-only requests are performed as one-sided remote traversals. We use remote read-only clients to focus on the performance of the request handling workload without network overheads. The

aim is to determine the mutual effects of a local workload and remote readers. This represents one of the important use cases for our design in which long-running analytical workloads execute concurrently with updates.

Throughput is measured as the average instantaneous throughput during a 10-second period, recorded at 50ms intervals. For per-client (and per-worker) results, the shaded region represents one standard deviation from the mean. This experiment is initialized with 5M entries in the data repositories.

(a) Worker throughput



(b) Read-only client throughput

Figure 1: ROMe-KV scalability for YCSB B local workload and read-only clients.

Figure 1 confirms the effectiveness of our design goals. Figure 1a shows the scalability of the local workload at different numbers of clients, while Figure 1b shows the scalability of clients at different local workload configurations. Our single ROMe-KV server can simultaneously support a local workload achieving over 2.4 Mops/s and a remote read-only workload of over 90 Kops/s. In the figure, all configurations provide similar performance (overlapping lines). That means, with increasing numbers of clients, and different amounts of workers, the performance of both the server and the clients are not negatively affected.

5 CONCLUSION

This brief announcement proposed ROMe, a novel solution to providing consistent access to objects with arbitrary sizes via RDMA. We then developed a distributed key-value store built from a B-link tree whose nodes are encoded using ROMe to suppport wait-free linearizable range queries.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-2045976. This research was also funded by a CORE grant from Lehigh University and a gift grant from the Stellar Dev. Foundation.

REFERENCES

[1] ARM. 2018. Arm CoreLink CCI-550 Cache Coherent Interconnect Technical Reference Manua. https://developer.arm.com/documentation/100282/0100/?lang=en. https://developer.arm.com/documentation/100282/0100/?lang=en

[2] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. 2016. The End of Slow Networks: It's Time for a Redesign. *PVLDB* 9, 7 (2016), 528–539. https://doi.org/10.14778/2904483.2904485

[3] Qingchao Cai, Wentian Guo, Hao Zhang, Divyakant Agrawal, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, and Sheng Wang. 2018. Efficient Distributed Memory Management with RDMA and Caching. *Proceedings of the VLDB Endowment* 11, 11 (jul 2018), 1604–1617.

[4] Haibo Chen, Rong Chen, Xingda Wei, Jiaxin Shi, Yanzhe Chen, Zhaoguo Wang, Binyu Zang, and Haibing Guan. 2017. Fast In-Memory Transaction Processing Using RDMA and HTM. *ACM Transactions on Computer Systems* 35, 1 (jul 2017), 1–37.

[5] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*. ACM Press.

[6] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. 2014. FaRM: Fast Remote Memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. 401–414.

[7] Philipp Fent, Alexander van Renen, Andreas Kipf, Viktor Leis, Thomas Neumann, and Alfons Kemper. 2020. Low-Latency Communication for Fast DBMS Using RDMA and Shared Memory. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE. https://doi.org/10.1109/icde48307.2020.00131

[8] Google. 2019. LevelDB:. https://github.com/google/leveldb.

[9] Goetz Graefe. 2010. Modern B-Tree Techniques. *Foundations and Trends in Databases* 3, 4 (2010), 203–402. https://doi.org/10.1561/1900000028

[10] InfiniBand Trade Association 2007. *InfiniBand Architecture Specification Volume 1*. InfiniBand Trade Association. Release 1.2.1.

[11] Intel. 2012. Intel Data Direct I/O Technology (Intel DDIO): A Primer. https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/data-direct-i-o-technology-brief.pdf

[12] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be General and Fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 1–16.

[13] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. 2022. Collie: Finding Performance Anomalies in RDMA Subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 287–305.

[14] Philip L. Lehman and s. Bing Yao. 1981. Efficient Locking for Concurrent Operations on B-trees. *ACM Transactions on Database Systems* 6, 4 (dec 1981), 650–670. https://doi.org/10.1145/319628.319663

[15] Junyi Liu, Aleksander Drakojević, Shane Flemming, Antonios Katsarakis, Dario Korolija, Igor Zablotchi, Ho cheung Ng, Anuj Kalia, and Miguel Castro. 2024. Honeycomb: ordered key-value store acceleration on an FPGA-based SmartNIC. *IEEE Trans. Comput.* 73, 857–871. https://doi.org/10.1109/TC.2023.3345173

[16] Christopher Mitchell, Yifeng Geng, and Jinyang Li. 2013. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*. USENIX Association, San Jose, CA, 103–114. https://www.usenix.org/conference/atc13/technical-sessions/presentation/mitchell

[17] Christopher Mitchell, Kate Montgomery, Lamont Nelson, Siddhartha Sen, and Jinyang Li. 2016. Balancing CPU and Network in the Cell Distributed B-Tree Store. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 451–464. https://www.usenix.org/conference/atc16/technical-sessions/presentation/mitchell

[18] PCI-SIG. 2014. PCI Express Base Specification Revision 4.0. (2014).

[19] Robert Ricci, Eric Eide, and The CloudLab Team. 2014. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications. *USENIX* 39, 6 (Dec. 2014).

[20] Dennis M. Ritchie and Ken Thompson. 1974. The UNIX time-sharing system. *Commun. ACM* 17, 7 (jul 1974), 365–375. https://doi.org/10.1145/361011.361061

[21] Arjun Singhvi, Aditya Akella, Maggie Anderson, Rob Cauble, Harshad Deshmukh, Dan Gibson, Milo M. K. Martin, Amanda Strominger, Thomas F. Wenisch, and Amin Vahdat. 2021. CliqueMap: Productionizing an RMA-Based Distributed Caching System. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. ACM. https://doi.org/10.1145/3452296.3472934

[22] Tobias Ziegler, Carsten Binnig, and Viktor Leis. 2022. ScaleStore: A Fast and Cost-Efficient Storage Engine using DRAM, NVMe, and RDMA. In *Proceedings of the 2022 International Conference on Management of Data*. ACM. https://doi.org/10.1145/3514221.3526187