Augmenting Corfu Shared Log System with RDMA

Shikhar Solanki
shikhar4

Ashwin Saxena
ashwins2

Abstract

Corfu is the first complete implementation of a fault-tolerant distributed system that utilizes flash devices attached directly to the network to organize a shared log, providing concurrent accessibility by multiple clients over the network. While Corfu has highly competitive performance statistics, opportunities do exist for improvement. The initial approach performs memory writes to replica sets of flash pages using the traditional TCP/IP networking protocol. In this paper, we extend the Corfu shared log by utilizing RDMA for log replication. This modified implementation results in up to 2.2x increase in throughput and 1.9x decrease in latency when compared to the base Corfu implementation.

1 Introduction

At a high level, the Corfu [2] distributed shared log abstraction provides an efficient means for clients to issue writes to a globally ordered log, which are replicated across multiple flash servers to ensure data durability and availability. Shared logs in general are powerful for ensuring high consistency in real-world faulty environments, and flash storage is ideal for supporting fast reads and sequential appends.

To provide this high performance, Corfu is implemented with three separate servers: a layout server, which maps each shared log position to a set of replica flash pages across the flash drive cluster, a sequencer server to provide the clients with the next available positions for writes, and a log server that stores all log updates across the replica flash units. The clients utilize the layout server to read from flash pages, and calls the sequencer to write to the replica set over the network. By maintaining a client-centric design, Corfu earns its title as the first distributed, shared log design that ensures write throughput is not limited by the bandwidth of a single flash unit. Instead, it becomes bottlenecked by the sequencer server, which can provide tens or even hundreds of times more bandwidth than a single flash drive. Corfu's reconfiguration mechanism, based on Vertical Paxos, provides tolerance for flash unit failures, fully restoring availability within tens of milliseconds.

In this paper, we delve into the idea of using RDMA to augment the Corfu shared log system. This modified implementation involves replacing the traditional networking protocol used to replicate writes with an RDMA-based solution, allowing clients to directly write to replicas using RDMA write operations, rather than sending messages to replicas over the network. By leveraging the low overhead of RDMA, we showcase that this approach can considerably reduce the latency and increase the throughput of the Corfu shared log.

The organization of this paper is outlined as follows. We firstly provide an overview of how RDMA can benefit the design of consensus protocol (§2). We then discuss the design of the Corfu shared log abstraction, and propose the modified design for Corfu with RDMA integration (§3). Followed by that, we describe our RDMA implementation on top of the open-source Corfu implementation developed in Java known as CorfuDB [13] (§4). Base on the implementation, we evaluate the throughput and latency of Corfu with RDMA on a cluster of server-attached SSDs (§5). Finally, we discuss a few related works (§6) and conclude the paper (§7).

2 Background and Motivation

The shared log is a fundamental abstraction that has been widely used in distributed systems for various purposes, such as replication, coordination, and metadata management. Corfu is a distributed shared log system that provides high-performance, fault-tolerant, and scalable storage services through a simple append-only interface [2]. Corfu's design is based on the idea of using a log as a shared memory abstraction, which allows it to efficiently support a wide range of distributed data structures, including distributed databases, file systems, and key-value stores [3].

However, as the demand for higher throughput and lower latency in distributed systems continues to grow, there is a need to further improve the performance of shared log systems like Corfu. The process of writing to replicas using traditional

networking protocols proves to be effective, given Corfu's high read/write performance. However, as mentioned previously, using TCP/IP protocols does introduce extra latency and overhead that may be unnecessary for applications implementing the Corfu system. TCP/IP protocols use the kernel to send messages from a client to server node, which incurs high data replication overheads before passing the data to the server application. Remote Direct Memory Access (RDMA) is a technology that allows direct memory access between two networked computer applications, bypassing the network stack overhead associated with OS kernels on both ends and offloading transfer functionality to the network adapter hardware [11]. This direct memory connection, supported by modern hardware, can potentially reduce the overhead of writing to replicas in Corfu. This approach could result in lower latency and higher throughput, providing better performance and scalability.

In the field of consensus protocol optimization, several approaches have been proposed that employ RDMA. One such approach is presented in the work of Aguilera et al. [1], which proposes a leader-follower based SMR (State Machine Replication) system that leverages RDMA for leader replication. However, the proposed system suffers from a bottleneck issue, where all replication requests pass through the leader node, and RDMA write requests are issued from the leader to followers. In the event of increased write throughput, the leader node may become a performance bottleneck and fail to scale efficiently. In comparison, the Corfu-based shared log, which utilizes client-side replication, achieves better performance due to its scalability [2]. In this approach, clients directly replicate writes to the servers, leading to load balancing of RDMA write requests across different clients, rather than congesting on a single leader. Other works, for example, Herd [9] and Farm [5] are RDMA-based key-value stores that achieve high performance by exploiting the low-latency and high-throughput capabilities of RDMA networks. Similarly, eRPC [8] is a general-purpose remote procedure call (RPC) library that uses RDMA to provide efficient communication between distributed nodes. These works suggest that augmenting Corfu with RDMA can potentially lead to significant performance improvements.

In this paper, we propose augmenting the Corfu Shared Log System with RDMA to overcome the limitations of traditional networking protocols and further enhance its performance and scalability. By leveraging the advantages of RDMA, we aim to provide a more efficient and scalable shared log system, while maintaining the desirable properties of the original Corfu design.

3 Design

We begin by outlining the original design of the Corfu shared log, followed by a comparison with our RDMA RPC design. Additionally, we will discuss how RDMA RPC is achieved using basic RDMA verbs.

The initial design of the Corfu shared log is shown in Figure 1. We consider a client consisting of an application linked with the Corfu runtime. A client with a Corfu runtime can manipulate a Corfu object, which could be a built-in container such as an array or hash-map, or a custom class that implements serialization functions. All updates are automatically translated into appends to the global shared log. Furthermore, the client can utilize the log interface provided by the Corfu runtime to append directly to the global log.

Upon issuing a write from the application through the Corfu runtime, the runtime requests an index in the global log using RPC to communicate with the sequencer service. The sequencer service assigns the client a token, representing a reserved location in the global log for a future write. This reservation helps avoid contention when multiple clients attempt to append to the log and allows the client to write data directly to a position mapped to a physical log server.

Meanwhile, the client maintains a local projection table that maps a global log position to disjoint ranges representing the address space of individual log servers. A single log position can be mapped to multiple address spaces for replication purposes. Once a position is reserved through the sequencer server, the client can perform a local lookup from the projection table and identify all log servers and address spaces where the update will be replicated.

The final step in the Corfu design involves writing the log to the log servers. The client connects to each log server using a chain replication strategy and issues parallel RPC requests, waiting for responses to ensure durability.

After discussing the basic version of the Corfu design, we present our design for Corfu with RDMA integration, as highlighted in Figure 2. The process of reserving positions and looking up address spaces remains the same as in the basic Corfu design. However, before the client starts replicating logs, it performs different steps when RDMA is available. First, the log server is configured with specific RDMA information, such as the available RDMA device ID, the IP for the RDMA NIC, and the port at which the RDMA service listens.

By setting up the log server in this manner, we design a handshake protocol for clients to upgrade their communication channels with the log server. Specifically, before a client replicates a log for the first time, it sends an RPC request called `getRDMAServer` using the default TCP channel to probe RDMA availability with the log server. Upon receiving the request, the server checks if the RDMA endpoint is available. If so, the server initializes the RDMA connection with the client by setting up WQ, CQ, and registering memory regions that allow future client write requests. If all steps are successful, the server acknowledges the client within the same RPC response. When the client receives the server's confirmation to upgrade the channel to RDMA, the client sets up its queues and registers memory for RPC buffers. At this point, the connection upgrade is considered complete, and all

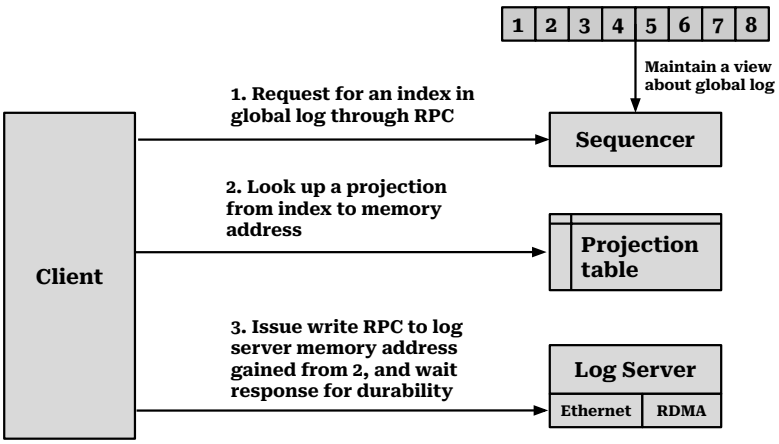


Figure 1: Initial design of Corfu

future log replications will use the RDMA channel. If any failure occurs during the handshake process, the client reverts to using the TCP channel for log replication.

Finally, for each append to the global log, the client sends it to the server via RDMA RPC. The RPC request contains the log data as bytes and an address where the log will be written. Upon receiving the request, the server copies the log data to its persistent storage and sends a response to the client with an OK message. Once the majority of servers acknowledge the request, the log data is considered committed with durability, and the client can return to the upper-layer application.

We then discuss how RDMA RPC is realized in our design through two-sided RDMA and the trade-offs with one-sided RDMA. To send an RDMA RPC request, the server must pre-register RPC buffers for potential client requests. The address space for these memory regions is shared with the client during the handshake stage. When the client needs to send log data, it uses `post_send()` to write data to the server's memory at a specified address. On the server side, the request is directly DMAed to the pre-registered RPC buffer by an RDMA-capable NIC. Due to the use of two-sided RDMA, the server continuously polls the CQ and gets notified by the receiving event. Then, corresponding handlers are triggered to process the request and persist any log data. Finally, the server sends a response to the client, and the process is reversed when the client sends the request.

Evidently, the two-sided RDMA RPC design requires server interaction to persist log data, such as polling the CQ and copying log data from the request buffer to persistent storage. To reduce server interaction and achieve zero-copy, it would be beneficial if the client could issue a one-sided RDMA write directly to the final memory address for data persistence without notifying the server. However, due to current limitations in the CorfuDB implementation we use, the persistent storage for log data is maintained through a non-linear hash map structure, making it difficult for the client to determine the exact memory address beforehand. As a result, we have not chosen the one-sided RDMA design. Future work could propose a more linear storage design for CorfuDB, allowing each log entry's exact memory address to be calculated in advance. This would enable clients to issue one-sided RDMA writes directly to the address, potentially reducing latency and improving server throughput.

4 Implementation

We have implemented RDMA on top of the open-source Corfu implementation, known as CorfuDB [13]. The RDMA replication feature is developed entirely in Java. The client functionality is primarily integrated within the Corfu runtime of CorfuDB, which serves as a foundational library for log abstraction that applications can be linked with. The shared data structure is also placed within the Corfu runtime, since the runtime library is linked to the server component as well.

For upgrading RDMA connections, we employ Corfu's existing protobuf [6] based protocol. The request and response structures are defined as part of CorfuDB's established schema.

To access RDMA from user space, we utilize DiSNI (Direct Storage and Networking Interface) [7], which is a Java wrapper for libverbs that employs JNI [10] to invoke native C functions. Since Java features built-in garbage collection and an object's lifecycle is not determined by the application, DiSNI makes use of off-heap memory for RDMA queues and buffers, managing their lifecycles independently.

For RDMA RPC in Java, we use DaRPC [12], which is built on top of DiSNI. DaRPC is capable of delivering 2-3 million operations per second with a latency of 10 microseconds (iWARP).

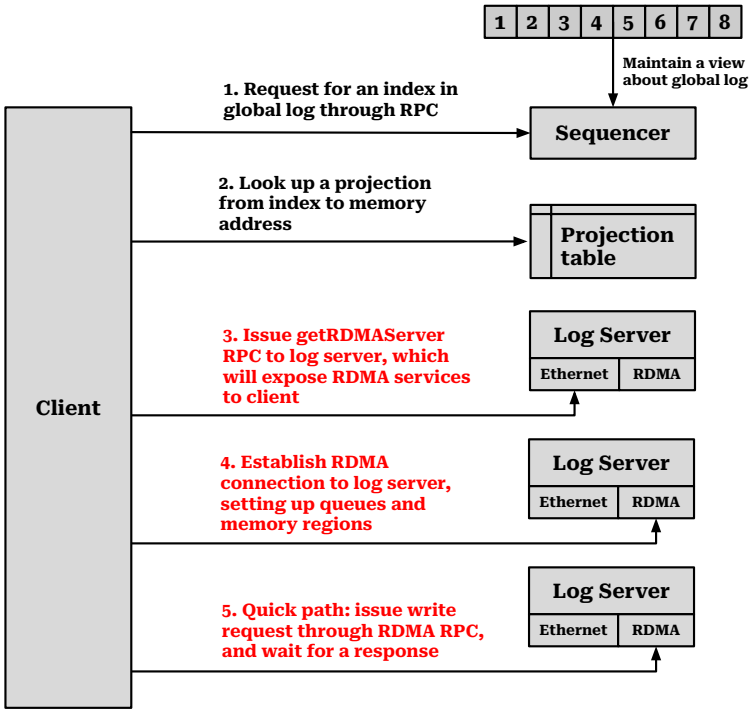


Figure 2: Corfu design with RDMA

5 Evaluation

To test the base Corfu implementation, we simulate a Corfu deployment on Cloudlab nodes. We have set up RDMA capabilities on the VMs and have verfied that these operations are working effectively, utilizing the multiple guides available online for this purpose.

The hardware utilized for our experiments are the Mellanox ConnectX-4 NICs provided by Cloudlab, which support RDMA operations. Our experiments focus on evaluating the performance of our RDMA-augmented CorfuDB in terms of latency and throughput and comparing these measurements with the base CorfuDB implementation under the same workload. We will also record the CPU usage of the system and compare its efficiency with the original CorfuDB implementation.

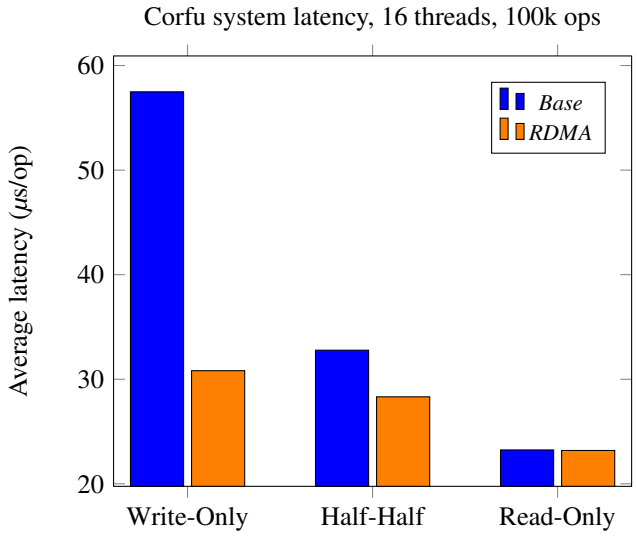
In our project, we aim to evaluate the performance of our system by comparing it with the basic Corfu implementation and Raft/Paxos-based consensus implementation optimized by RDMA. The focus of our study is on measuring the latency of our system on a given throughput and comparing the upper bound of the throughput with the increase in latency, which serves as a reliable indicator of the overall throughput capability of the system.

To carry out this evaluation, we conduct a series of experiments in a controlled environment that simulates a real-world scenario. We measure the latency of the system using a variety of workloads: Write-Only, Read/Write, Read-Only.

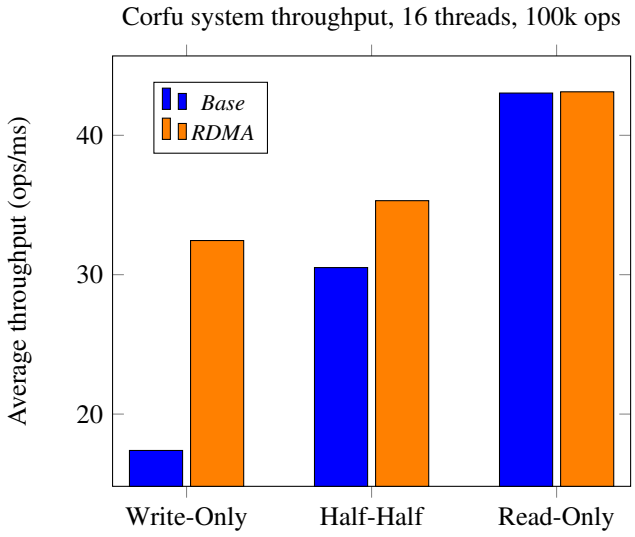
Our study is significant as it will provide insights into the performance of our system compared to existing consensus implementations on top of RDMA. By measuring the latency and throughput, we will be able to identify the system's strengths and limitations, which will guide us in further optimizing its design.

Figure 3 shows our evaluation result. Figure 3a compares the latency of the base CorfuDB writes to the RDMA-enabled CorfuDB writes. For evaluation setup, we set up a single Corfu server on 2 different Cloudlab nodes, both RDMA enabled. Each server spawned 16 threads and issued 100K operations each on 4KB objects. In this table, the x-axis represents the various workloads and percentage of writes and reads being performed. The y-axis indicates the average time it took for each server's process to perform 100K operations of the desired workload. We can see that writes issued via RDMA take nearly half the time when compared with the base implementation. These results make sense as the RDMA issued writes from a client bypasses the OS network stack on the other nodes making the overall time to issue writes shorter.

Likewise in Figure 3b, the x-axis represents the type of workload performed and the y-axis represents the operations / millisecond the setup performs. We can see that the throughput of the RDMA write workload is a little over 2x compared



(a) Latency for different workloads



(b) Throughput for different workloads

Figure 3: Throughput and Latency for base Corfu implementation vs RDMA implementation

to the normal write workload. Evidently, RDMA greatly increases the efficiency of the system.

6 Related Work

6.1 Microsecond Consensus for Microsecond Applications [1]

In this paper, the authors implement Mu, a state machine replication and consensus architecture that leverages RDMA to allow replicas to execute requests in less than 1.6 microseconds. Their architecture achieves consensus and replicates a request in one round of parallel RDMA write operations on a majority of replicas in contrast to prior approaches which achieve consensus through multiple rounds. Mu guarantees that a replica log can only be written by one leader via RDMA writes.

One key difference between MU and Corfu is that MU utilizes a leader and follower architecture where only the leader is capturing client requests and issuing RDMA writes to the local logs of all replicas. In Corfu, hundreds of concurrent client machines are able to append to the tail of a shared log and read from it over a network. There is no single I/O bottleneck to append or read and the cluster throughput can be utilized by the clients. Our proposal differs from Mu by utilizing this parallelism in operations and we replace writes over the network with RDMA writes.

6.2 Fast In-memory Transaction Processing using RDMA and HTM [14]

In this paper, the authors implement a fast in-memory transaction processing system, DrTM, that leverages Hardware transactional memory and RDMA to greatly improve latency and throughput of a state of the art distributed transaction system. Their system leverages HTM to perform concurrency control like tracking read/write operations and detecting conflicting operations on a local machine. To ensure serializability amongst concurrent transactions on distributed machines, their system utilizes RDMA's strong consistency to glue HTM transactions together which an RDMA operation will abort if an HTM transaction is accessing the same memory location. Their system preserves serializability amongst distributed transactions due to the strong consistency of RDMA and strong atomicity of HTM as any conflicting transactions on a remote machine will be aborted.

6.3 PRISM: Rethinking the RDMA Interface for Distributed Systems [4]

In this paper, the authors augment the basic RDMA interface by creating extensions that allow distributed systems to better utilize the low latency and CPU offload capabilities of RDMA hardware. Their proposal is the PRISM interface which extends the RDMA read/write interface with four additional primitives: indirection, allocation, enhanced compare-and-swap, and operation chaining. These support common remote access patterns such as data structure navigation and concurrency control and are simple enough to implement in a RDMA NIC as it utilizes existing micro architectural mechanisms.

7 Conclusion

In conclusion, we demonstrated the immediate benefits of utilizing RDMA to lower the latency of writes in CorfuDB. We showed the decrease in latency and increase in operation throughput. Despite this, our implementation has limits with regards to the overhead incurred by our two-sides RDMA RPC design. Future work in CorfuDB could propose a linear storage design and allow each log entry's memory address to be calculated ahead of time. With this, one-sided RDMA writes could be performed directly to the address which would reduce latency and increase throughput. Another line of future work would be developing RDMA reads. Our currently implementation only utilizes RDMA writes and RPC reads. Since read operations are much more frequent that write operations, increasing the efficiency of RDMA reads would massively benefit the system.

References

[1] Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra J Marathe, Athanasios Xygkis, and Igor Zablotchi. Microsecond consensus for microsecond applications. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 599–616, 2020.

[2] Mahesh Balakrishnan, Dahlia Malkhi, John D. Davis, Vijayan Prabhakaran, Michael Wei, and Ted Wobber. Corfu: A distributed shared log. *ACM Trans. Comput. Syst.*, 31(4), dec 2013.

[3] Mahesh Balakrishnan, Dahlia Malkhi, Ted Wobber, Ming Wu, Vijayan Prabhakaran, Michael Wei, John D. Davis, Sriram Rao, Tao Zou, and Aviad Zuck. Tango: Distributed data structures over a shared log. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, page 325–340, New York, NY, USA, 2013. Association for Computing Machinery.

[4] Matthew Burke, Sowmya Dharanipragada, Shannon Joyner, Adriana Szekeres, Jacob Nelson, Irene Zhang, and Dan RK Ports. PRISM: Rethinking the RDMA interface for distributed systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 228–242, 2021.

[5] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. No compromises: distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th symposium on operating systems principles*, pages 54–70, 2015.

[6] Google. Protocol buffers., 2023.

[7] IBM. Disni: Direct storage and networking interface., 2017.

[8] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter RPCs can be general and fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 1–16, 2019.

[9] Anuj Kalia, Michael Kaminsky, and David G Andersen. Using RDMA efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, pages 295–306, 2014.

[10] Oracle. Java native interface., 2023.

[11] Renato Recio, Bernard Metzler, Paul Culley, Jeff Hilland, and Dave Garcia. A remote direct memory access protocol specification. Technical report, RFC 5040, October, 2007.

[12] Patrick Stuedi, Animesh Trivedi, Bernard Metzler, and Jonas Pfefferle. Darpc: Data center rpc. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–13, 2014.

[13] VMWare. Corfudb — a cluster consistency platform., 2017.

[14] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. Fast in-memory transaction processing using RDMA and HTM. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 87–104, 2015.