Telepathic Datacenters: Efficient and High-Performance RPCs using Shared CXL Memory

Suyash Mahar, Ehsan Hajyasini, Seungjin Lee, Zifeng Zhang, Mingyao Shen, Steven Swanson
UC San Diego

1 Introduction

Communication within the datacenter needs to be fast, efficient, and secure against unauthorized access and rogue actors. Remote procedure calls (RPCs) are one of the popular ways of communicating between independent applications and make up a significant portion of datacenter communication, particularly among microservices. However, RPCs require substantial resources to service today's datacenter communication needs. For instance, requests spend over 25% of their time in the RPC stack for the 95th percentile [1] at Google. One of the significant sources of latency for RPC frameworks is their need to serialize and compress data before transmission, which is especially resource-intensive for dynamic data structures like trees and graphs.

The upcoming Compute Express Link (CXL) [5] based multi-host shared memory offers an exciting alternative by providing hardware-supported cache coherency among multiple compute nodes. Instead of serializing data structures and transmitting them as a bitstream over the network, applications can now just share a pointer to data, significantly lowering their CPU usage and energy needs.

However, shared access to pointer-rich data over shared memory raises several safety concerns. First, it eliminates the traditional isolation of the sender from the receiver in TCP/IP-based networking. Second, applications now need to coordinate shared memory management to avoid memory leaks. Finally, hardware-based cache coherency networks have limited scalability, often limited to 32 compute nodes.

To address these issues, we present RPCool, an RPC framework designed for low-latency, high-throughput communication in untrusted, datacenter-scale environments. RPCool supports native pointers, avoids the security risks of communicating between distrusting entities, scales beyond a single rack, and enables applications to manage shared resources efficiently.

However, to implement RPCool, we must solve major challenges associated with shared memory communication. For example, enabling concurrent access to data structures by the sender and the receiver poses several security risks. If raw memory is shared, the sender could modify data structures while the receiver processes them, resulting in data races and leaving the receiver vulnerable. Moreover, a malicious actor could share pointers to the receiver's private memory, potentially leaking sensitive data, corrupting the receiver's memory state, or crashing it entirely.

Another major challenge with CXL-based shared memory RPC is that they are limited to rack scales [4], also shown in Fig. 1(a). Thus, in a traditional datacenter environment

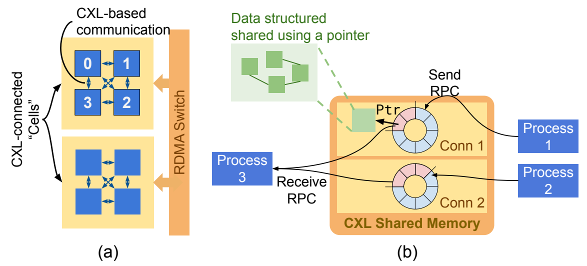


Figure 1. (a) Expected scale of CXL 3.0. (b) Working of RPCool's shared memory communication.

where two communicating applications might be situated on different compute nodes—for instance, to meet specific hardware needs, the application must dynamically switch RPC protocol, adding significant development overhead.

Finally, using shared memory for communication results in challenges with availability and memory management. Take, for instance, situations in which an application sharing a region of shared memory crashes but does not relinquish the memory for others to use. In this case, a dilemma emerges: should the orchestrator release the region on the application's behalf? Or should it be saved for recovery?

To solve these issues, we propose RPCool, a secure and efficient RPC framework over CXL-based shared memory that seamlessly falls back to RDMA for intra-datacenter communication. Using RPCool, applications make RPCs with pointer-rich data structures as their arguments directly over the CXL-based shared memory. The receiver can optionally ask the sender to give up access rights to the corresponding memory region to avoid data races. Once the receiver is notified of an incoming RPC, it can process RPC arguments in a lightweight sandbox to prevent pointers from escaping the shared memory if the sender is distrusted. Finally, to prevent memory leaks when applications crash and to restrict servers and clients from retaining access to the heap indefinitely, potentially starving other processes, RPCool implements quotas for shared memory accessible to an application and leases for each heap.

2 Overview

RPCool provides a fast, efficient, and secure RPC mechanism for applications to communicate while sharing pointer-rich data structures. When available (Fig. 1a), RPCool uses CXL-based shared memory to communicate, relying on the hardware cache coherence. However, if one of the RPC participants is accessible only over RDMA, RPCool automatically and seamlessly switches to software-based coherence.

Applications in RPCool communicate over *channels* that an RPC server creates and clients *connect* to (Fig. 1b). Each channel has a unique name and can support multiple client connections. Clients can allocate their data structures in a connection-only heap that is not shared among clients or a channel-wide shared heap.

Next, we will discuss different features supported by RPCool.

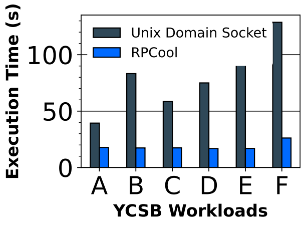
Support for Native Pointers As RPCool supports native pointers, these pointers need to work across different processes. To facilitate this, RPCool maps each connection to the same address across all processes that access it. Furthermore, RPCool reserves a 32 TiB region when an application starts and assigns a globally unique address to each connection to prevent conflicts.

Preventing Data Races across Server and Client To prevent the RPC sender from modifying shared data structures while processing an RPC request, the receiver can require the sender to relinquish write access to the shared heap. RPCool facilitates this by supporting a `seal()` system call that revokes write access for the sender until the RPC returns and also generates an identifier for the receiver to verify the region is sealed. Upon completion of the RPC call, the receiver can call a corresponding `release()` system call to lift the seal on the heap.

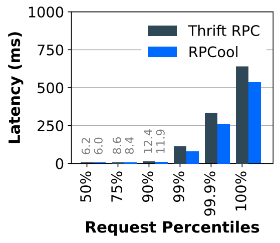
Securely Processing Shared Data Structures To mitigate the risk where the sender maliciously or otherwise crafts data structures with wild or invalid pointers, RPCool implements a lightweight sandbox based on Intel Memory Protection Keys (MPK) [2]. When the receiver activates the sandbox to process the received data, e.g., traversing a tree, the sandbox disables read/write access to all the private memory and automatically creates readable copies of variables explicitly allowed by the programmer. Thus effectively preventing any unauthorized access to private memory using malformed pointers.

Managing Shared Memory RPCool must ensure that applications release shared memory heaps to the orchestrator when no longer in use and avoid memory leaks if one or more participants of an RPC channel crash. To support this, RPCool assigns a user-defined *quota* to each application, specifying the maximum number of heaps they can access concurrently. If the application exceeds its quota (e.g., because of unreleased heaps with no clients), it will be unable to connect to or serve on new channels until it releases sufficient number of existing heaps. Moreover, the orchestrator maintains leases for each heap on the shared memory to notify participants of an RPC channel's failed nodes. The server and the client periodically renew these leases; if an application crashes, the orchestrator can notify other participants about the failure.

Seamless Fallback to RDMA To enable datacenter-scale communication despite the limited scalability of CXL-based shared memory, RPCool provides an automatic fallback to the RDMA network if a client connects to a server that is available only over RDMA. RPCool achieves this by automatically deep-copying the arguments of an RPC call when



(a) Memcached using Unix Domain Sockets and RPCool.



(b) Latency distribution for the SocialNetwork benchmark.

Figure 2. RPCool performance comparison.

communicating over RDMA. This, however, results in a mismatched programming interface by being pass-by-reference RPC over CXL and pass-by-value RPC over RDMA. Thus, to provide a unified interface and reduce programming effort, RPCool assumes that only one writer will modify the RPC argument at a time and requires the applications to follow the single-writer rule.

3 Results

To evaluate RPCool, we modified Memcached and DeathStarBench's [3] SocialNetwork benchmark and measured their performance (Fig. 2). Across the workloads, we see that RPCool significantly outperforms unix domain sockets and ThriftRPC, thus showing the usefulness of RPCool.

4 Conclusion

In this extended abstract, we presented RPCool, a fast, efficient, scalable, and secure RPC framework for the age of rack-scale coherent shared memory. RPCool solves the significant problems associated with using shared memory for making remote procedure calls by providing support for native pointer-rich data structures, shared memory heap sealing, processing shared data in a low-overhead sandbox, and automatic RDMA fallback. Overall, RPCool outperforms traditional RPC and shared memory techniques.

References

[1] Korakit Seemakhupt et al. 2023. A Cloud-Scale Characterization of Remote Procedure Calls. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 498–514.

[2] Soyeon Park et al. 2019. libmpk: Software abstraction for intel memory protection keys (Intel MPK). In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 241–254.

[3] Yu Gan et al. 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 3–18.

[4] James Morra. 2023. CXL Switch SoC Unlocks More Memory for AI. Retrieved from https://www.electronicdesign.com/technologies/embedded/article/21272132/electronic-design-cxl-switch-soc-unlocks-more-memory-for-ai.

[5] Debendra Das Sharma. 2022. Compute Express Link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 5–12.