Remote Memory Calls

Emmanuel Amaro[⋆] Zhihong Luo[⋆] Amy Ousterhout[⋆] Arvind Krishnamurthy[∘]
Aurojit Panda[†] Sylvia Ratnasamy[⋆] Scott Shenker[⋆‡]
[⋆] UC Berkeley [∘] University of Washington [†] NYU [‡] ICSI

Abstract

In this paper we propose an extension to RDMA, called *Remote Memory Calls* (RMCs), that allows applications to install a customized set of 1-sided RDMA operations. We then explain how RMCs can be implemented on the forthcoming generation of SmartNICs and discuss the resulting tradeoffs between RMCs, 1-sided RDMA operations, and 2-sided RDMA operations.

ACM Reference Format:
Emmanuel Amaro, Zhihong Luo, Amy Ousterhout, Arvind Krishnamurthy, Aurojit Panda, Sylvia Ratnasamy, Scott Shenker. 2020. Remote Memory Calls. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20), November 4–6, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3422604.3425923

1 Introduction

RDMA bypasses CPUs and the kernel by offloading much of the network stack to the NIC, thereby lowering latencies, achieving higher throughput, and using fewer CPU cycles on both client and server. As a result, in recent years RDMA has seen significant adoption in datacenters [11, 27, 48], and is currently used to improve the performance of a variety of distributed applications including key-value stores [10, 23, 24, 26], machine learning [47], and graph processing [34]. RDMA is also a key enabler for disaggregated datacenters [2, 12, 15, 33] and GPU clusters [16, 22, 45].

RDMA supports two kinds of operations: 1-sided, where an RDMA operation issued by a client is handled by the server's RDMA NIC; and 2-sided, where a client's RDMA operation is processed by a core on the server. The main advantage of 1-sided over 2-sided is the reduced load on the server CPU. However, since 1-sided operations are implemented in NIC hardware, they are less flexible than 2-sided; 1-sided operations only include read, write, atomic fetch and add, and atomic compare and swap [41]. As others [1, 21, 23, 36] have observed, real applications exhibit more complex access patterns such as indirect memory accesses and memory scans that would require multiple 1-sided RDMA operations, which increase latency, network traffic, and client CPU utilization. The gap between the breadth of application requirements and the narrowness of 1-sided RDMA operations has caused many [20, 21] to turn to 2-sided RDMA operations, where the greater flexibility at the server allows more complicated operations to be performed with a single RDMA invocation, but imposes a higher load on server CPUs.

Previous efforts [1] have called for a broader set of 1-sided operations to be implemented in RDMA NIC hardware, but this approach faces two challenges: **(1)** it will take years to agree on and then deploy a new generation of hardware-enabled RDMA operations, and **(2)** it is not clear that any fixed set of additional operations will suffice for all datacenter applications, particularly since RDMA does not provide mechanisms through which operations can be chained together. To address these challenges we propose *Remote Memory Calls* (RMCs), which are a specialization of software RPCs to memory-oriented (rather than compute-oriented) remote operations. We find that currently available RDMA-enabled SmartNICs have the features required to implement RMCs but lack the performance advantage over 1-sided RDMA. However, our analysis indicates that forthcoming SmartNICs will have both the features and higher performance to allow RMCs to prevail over 1-sided operations. As we discuss in §3, host memory access latency, especially for small memory reads, is the main impediment to currently achieving this vision. Forthcoming improvements to SmartNICs–including faster CPUs, new IO busses, and improvements to the software libraries used to access host memory–will reduce host memory access latency. As a result, we think an RMC-enabled future, where applications can define custom 1-sided operations, is within reach.

In the rest of this paper we describe RMCs (§2), provide a feasibility roadmap to how they can be implemented on SmartNICs (§3), and demonstrate their benefits (§4). Then, in §5, we compare the tradeoffs between RMCs, 1-sided, and 2-sided operations, and discuss how RMCs as an abstraction can allow users to flexibly navigate these tradeoffs.

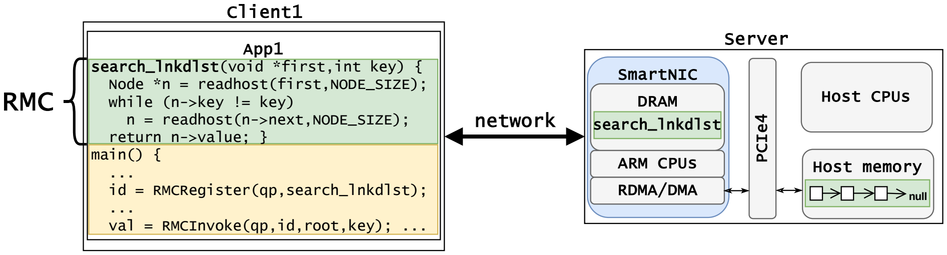
2 RMC design

RMCs are memory-oriented software procedures that are registered and invoked by a client, and executed by a remote server's RDMA NIC CPUs (see Figure 1). RMCs are useful for any distributed application that needs to make multiple round trips to the same server to achieve desired functionality; e.g., key value stores, lock managers, replication and consensus protocols, etc. An RMC can perform a variety of actions including: **(a)** reading and writing from host memory (e.g., `readhost()` in Figure 1); **(b)** performing atomic operations on host memory; **(c)** conditional branching and looping; **(d)** monitoring writes to host memory addresses (similar to `mwait`); and **(e)** basic computation, including calls to functions that hash or compress data. Despite the apparent generality of these operations, RMCs are not meant to be used for arbitrary application logic, only for memory access logic. In practice, we expect the limited NIC resources will restrict the size (i.e., number of instructions) and runtime of RMCs, but this will depend on the hardware on which they are eventually deployed.

Client applications register RMCs with the server NIC at runtime; they do this by first creating a queue pair (QP) and then using a new `RMCRegister` operation that associates the RMC with the QP. RMCs can allocate and store state in the NIC's local memory. Distinct QPs

HotNets '20, November 4–6, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8145-1/20/11.
https://doi.org/10.1145/3422604.3425923



```
search_lnkdlst(void *first,int key) {
  Node *n = readhost(first,NODE_SIZE);
  while (n->key != key)
    n = readhost(n->next,NODE_SIZE);
  return n->value; }
main() {
  ...
  id = RMCRegister(qp,search_lnkdlst);
  ...
  val = RMCInvoke(qp,id,root,key); ...
```

Figure 1: *RMC architecture. We show a client application* `App1` *registering and invoking the RMC* `search_lnkdlst()`, *which searches for a Node with a given key in a (remote) host memory linked list. The RMC uses the* `readhost()` *function to access host memory from the SmartNIC.*

(from the same application on different clients) might register and use the same RMC, in which case the NIC keeps a single copy of the RMC code in local memory, and the RMC dictates whether to use per-QP state, or share state across QPs. An RMC is unregistered (i.e., all state erased and resources freed) when all QPs that registered it have disconnected. In order to call RMCs we add another `RMCInvoke` operation to the RDMA verbs library, which takes an `id` returned by the registration operation, and any RMC parameters. The same RMC can be invoked many times after a single registration.

Our goal is to allow applications to avail themselves of whatever RMCs would best suit their needs. To that end, we require that: **(1)** application developers should be able to easily develop new RMCs suited to an application's memory access patterns; and **(2)** client applications should be able to register RMCs quickly at runtime. Although we leave a complete RMC API specification for future work, at a minimum it should support most of the features the SmartNIC's RDMA or DMA engine provides to access host memory; e.g., reading, writing, scatter/gather, and atomics. In addition, the API should allow RMCs to declare variables and small buffers that outlive an RMC execution. This would be crucial to support counters, cached values, etc., and importantly, synchronization primitives that use NIC-memory; e.g., in `lock(x)`, the address of `x` should be NIC-memory accessible to all RMCs. Further, notification support could be directly embedded in an RMC; e.g., at the end of an RMC a condition can be added, and if the condition is true, the RMC enqueues a message to be sent to a set of subscribed clients.

RMCs' approach is similar to previous efforts to enable near-memory compute [37], and programmable SSD controllers [9] except RMCs target remote memory access over the network fabric.

3 Implementing RMCs on SmartNICs

3.1 SmartNIC hardware

We argue that the next generation of SmartNICs will have the appropriate hardware capabilities to efficiently support RMCs. As such, in the following three paragraphs we review SmartNICs in detail and note how specific hardware components enable desirable RMC capabilities.

Currently, most SmartNICs are built using either FPGAs (e.g., Xilinx Alveo [46], Mellanox Innova-2 [38]), or embedded CPUs for computation. While prior work such as StRoM [36] has looked at how FPGAs can enable custom RDMA extensions, in our work we focus on CPU NICs. This is for four reasons: first, FPGA synthesis and reconfiguration (including partial-reconfiguration) is a slow process that in many cases requires manual intervention [44], thus adding unacceptably high latency when registering new RMCs. Second, FPGAs on NICs impose static resource limitations which limit the number of RMCs that can be registered on the NIC at one time. Third, many programmers find it challenging to write FPGA designs and this could be an impediment for them to use RMCs for their applications. Finally, RMCs designed to be executed on SmartNIC CPUs can be trivially ported to run on the server or client CPU. As we discuss later in §5, the flexibility to run RMCs on both SmartNIC and host CPUs allows a scheduler to optimize the RDMA primitives used by an application; this cannot be done when FPGAs are used. Thus, while RMCs and StRoM both strive to enable RDMA extensibility, the projects have different goals which result in significantly different designs and implementations.

In this paper we focus on *CPU-based RDMA-enabled SmartNICs* which usually have the following components: **(1)** a dedicated ASIC for handling network and RDMA operations, **(2)** a hardware embedded switch which can be programmed to forward packets to the NIC or host, **(3)** a multi-core CPU, **(4)** some amount of on-board memory, and **(5)** a mechanism that allows the NIC CPUs to efficiently access host memory; depending on the specific design this could be achieved by reusing the RDMA engine, or with an independent DMA unit.

To make our feasibility analysis more concrete we focus on Mellanox Bluefield-1 NICs [39] (BF1); the measurements in [25] previously showed that specifications for BF1 are comparable to other CPU-based SmartNICs. Bluefield-1 has a 16-core 64-bit ARM processor, where each core runs at a frequency of 1.35GHz.[1] CPU frequency is important as it influences overall RMC runtime. BF1 includes 16GB of DRAM and is cache coherent, paving the way for RMCs to support locks and transactions without accessing host memory.

3.2 Feasibility

We now consider whether RMCs can be implemented in a platform similar to BF1. There are three main concerns to address: **(1)** *is there enough bandwidth on the bus between the NIC and host memory*; **(2)** *are SmartNIC to host memory access latencies smaller than RTTs in a datacenter* (if not, then multiple 1-sided operations would be faster); and **(3)** *can SmartNICs execute RMCs at a sufficiently high rate.*

The bandwidth between the NIC and host memory is a concern because RMCs are memory-oriented procedures, and we expect each RMC to issue more than one access to host memory. Thus for RMCs to be viable, a SmartNIC must be able to drive at least as much bandwidth and operations per second to its host memory as a 1-sided client across the network. CPUs in BF1 access host memory through

[1] BF1 CPUs run by default at 800MHz, however a control register allows the clock frequency to be raised.

PCIe4 by reusing its RDMA engine, and there are two device versions: an 8 PCIe lane version that supports network links of 25Gb/s, and a 16 lane version that supports 100Gb/s. In PCIe4 each lane can transfer data at 16Gb/s,[2] therefore, the x8 device can transfer data to its host at 128Gb/s, ≈ 5× the network bandwidth the interface supports. The ratio between available PCIe and network bandwidth is reduced to ≈ 2.5× with the x16 version (i.e., 256Gb/s for PCIe vs 100Gb/s for network), but in both cases PCIe has more capacity than the network bandwidth. In terms of operations per second, FaSST [21] previously reported that ConnectX-3 over RoCE could achieve ≈ 11M 1-sided RDMA reads/s and an Infiniband device could achieve ≈ 50M 1-sided RDMA reads/s. In contrast, a recent PCIe study [29] reported that PCIe3 with 8 lanes can support 200M reads/s.[3] Therefore, PCIe3 supports ≈ 18× more reads/s compared to ConnectX-3 RoCE, and ≈ 4× higher reads/s compared to the Infiniband device. Thus BF1 has sufficient bandwidth to support RMCs that issue multiple memory operations and that transfer more data between SmartNIC and host during RMC execution than they do over the network.

Regarding the second concern–whether access latencies from SmartNIC to host memory are lower than network RTTs–note that any traditional network communication between host CPUs must traverse the PCIe bus. In particular, the core-to-core delays (one way) across a datacenter are composed of two PCIe traversals in addition to the network traversal, while communication from the NIC to host memory only requires one PCIe traversal. Prior work [27] has observed network RTTs of approximately 25µs in a highly-tuned production datacenter. In multi-tenant datacenters, three recent measurement studies [4, 18, 32] have found that the round-trip times between two AWS instances in the same region and availability zone vary between 90 and 500µs.[4] Popescu et al [32] observed similar network latencies on GCE and Azure. By contrast, [29] reports NIC PCIe traversal latencies ranging between 0.6 and 1.2µs. Thus, in the pessimal case (i.e., when taking the lowest network latency and the highest PCIe latency) two PCIe delays are more than 10× smaller (2.4µs vs. 25µs) than the full round-trip delays, while in multi-tenant datacenters they are more than 37× smaller (2.4µs vs. 90µs). Furthermore, Ranganathan and Vahdat recently proposed a future datacenter architecture in a keynote [43], where datacenters are built as collections of cliques with network latency of ≈ 10µs. Additionally, they reported current PCIe latencies of ≈ 1µs. These are indications that future datacenter architectures are likely to maintain a 10× latency gap between the network and PCIe. Finally, we speculate that even if the interconnect between NIC and its host changes from PCIe4 to a new bus (e.g., PCIe5 [31], Gen-Z [13], CXL [8], CCIX [7]), server platforms are likely to use the new bus to support multiple I/O devices, so the performance of the internal bus is likely to remain higher than the network fabric.

Now we consider the last concern–whether SmartNICs can execute RMCs at a sufficiently high rate. To answer this question we used a microbenchmark that compares RDMA read throughput between: **(1)** one BF1 core and its host memory (i.e., **NIC-Host**); and **(2)** one server-grade core at 2.6Ghz[5] with a 100Gb/s RDMA NIC connected back-to-back to another host equipped with a BF1 NIC (i.e., **Host-Host**). We found that for 1KB reads,[6] Host-Host achieves 10.2M reads/s while NIC-Host achieves 9.6M reads/s (94% of Host-Host). However, with smaller reads of 128B, Host-Host achieves 19.4M reads/s while NIC-Host achieves 9.5M reads/s (49% of Host-Host). This lower rate of operations supported by NIC-Host is due partially to the 1.35Ghz frequency of the SmartNIC CPUs–much lower than many server CPUs including our comparison. In our testing we found that invoking RDMA operations from BF1 CPUs uses a significant number of cycles and frequency becomes a bottleneck especially at small access sizes. Fortunately, RMCs are memory dominated and will not include significant amounts of additional compute. Furthermore, adoption of RMCs and other techniques that rely on host memory access from SmartNIC CPUs is likely to catalyze efforts to optimize the host memory access stack, thus reducing its cost in cycles. Additionally, future SmartNICs (including the already announced Bluefield-2 [40]) include CPUs with higher clock frequencies (i.e., 2.5Ghz), which should further address this issue. While hardware specifications alone do not allow us to project the rate at which SmartNICs can execute and schedule RMCs (which depends on system design, host memory access pattern, CPU microarchitecture, etc.), we believe our analysis shows that RMCs could achieve sufficient throughput (compared to 1-sided RDMA) using multiple cores on future SmartNICs.

This analysis leads us to conclude that the answer to the three questions is either already yes, or will soon be with newer SmartNICs, so we believe that SmartNICs represent a viable approach to deploying RMCs. However, many implementation challenges remain, which we turn to next.

3.3 Implementation challenges

In the previous section we showed that one BF1 core could support up to 9M reads/s of 128B. On average, this would require executing RMCs that issue host memory accesses every 111ns. These small time budgets pose several challenges including:

Low-cost memory isolation: In our model, a single SmartNIC executes RMCs registered by multiple client applications. Enforcing memory isolation between these RMCs is crucial to ensure that bugs in one application do not impact the correctness of another, and is also a key building block for network security. However, on most systems, switching between memory-isolated processes can take a microsecond or more [5]. Containers and VMs further exacerbate this problem [30]. Instead, prior work including SPIN [6], Singularity [17], and Netbricks [30] have shown that compiler-assisted isolation, which relies on static type checking and runtime bounds checking, can provide lower isolation overheads and allow switching between isolated contexts in nanoseconds. We believe that these techniques, appropriately extended to the RMC context, provide a promising approach to address this challenge.

Low-cost preemption and scheduling: We also need to ensure performance isolation between RMCs, in particular, to guarantee that a given QP cannot monopolize processing resources to the detriment of others. Current systems rely on preemption to interrupt long-running tasks and provide performance isolation. However, on

[2] We ignore encoding overheads in this discussion.

[3] And 180M writes/s.

[4] The wide range is due to configuration differences; the use of Virtual Private Cloud usually enables lower network latencies.

[5] Intel Xeon Sandy Bridge E5-2640 v3.

[6] Using one BF1 core, 4 queue pairs, posting lists of 16 work requests, and using signaled sends only once every 16 posts.

most CPUs interrupts require 1000s of cycles [19] which translates to overheads on the order of microseconds. As such, interrupt-based preemption is too slow for an RMC system. Instead, an RMC system could switch between RMCs when they call helper functions, such as those for asynchronous host memory accesses (e.g., `readhost()`).

In order to safely install RMCs on a SmartNIC we propose to borrow techniques that enable safe extensions of highly sensitive code. Several projects, including the Linux kernel, have recently started using eBPF programs to provide such extensions. By restricting the class of programs that can be expressed by using eBPF and a verifier that proves termination, these projects can guarantee that code functions will finish in a bounded number of CPU cycles. While the need to prove termination often complicates the task of writing these programs (e.g., imposing limits on how loops are written), recent efforts [14] have looked at extending eBPF verification using abstract interpretation and other techniques in order to support a wider range of programs, thereby reducing the programming effort. Therefore, semantically-constrained programming languages provide a mechanism for RMCs to avoid runtime protection overheads.

An RMC system would also need to decide the order in which RMCs execute. Fair scheduling algorithms often require significant state updates so their execution can take microseconds. For example, Torrey et al. [42] found that the O(1) completely fair scheduler included in Linux 2.6 required about 4µs to schedule jobs on a uniprocessor system (a best case scenario). Simpler schedulers might be faster but may not provide fairness between QPs. A system for RMCs therefore needs to adopt a scheduling discipline that provides an adequate compromise between fairness and timeliness.

Programming language: Our proposed solutions for memory isolation and scheduling concerns require programmers to use a domain-specific language to write RMCs. Such a language should be portable across CPU architectures; e.g., based on C or C++, and as we mentioned before, should be verifiable like eBPF. Furthermore, the programming language should allow our API to be used (see §2). Fully identifying a concrete programming model for RMCs is a major open question that we leave for future work.

4 How far away are RMCs in practice?

Now that we have argued for the feasibility of RMCs on SmartNICs, we turn to the question of *what are the benefits of using RMCs on today's, and future SmartNICs.* We do so by comparing the RMC approach to 1-sided operations in two relevant workloads: **(1)** scanning a remote data structure; and **(2)** hashing a remote buffer. Since we do not have an RMC system yet, we evaluate the RMC approach by modeling various delays. In particular, for both workloads, we assume the RMC runtime adds 0.5µs of scheduling delay between invocation and execution. We estimate currently achievable latencies using Mellanox Bluefield-1 NICs (described in §3.1); and project achievable performance in the near future using data from recent works that measure PCIe performance [29] and SmartNIC performance [25, 28].

The results that follow indicate that, as expected, the use of RMCs *significantly* reduces the load on the network, and therefore, reduces congestion; for contexts where network congestion is a problem, this is critical. However, the results on latency are mixed. The delays incurred using BF1 are typically worse than those achieved with 1-sided host-to-host operations. On the other hand, the delays estimated using idealized NICs fare better (for the scan workload) or comparably (for the hashing workload) than 1-sided delays. These results indicate that while RMCs implemented on today's SmartNICs would decrease network load, they would not decrease delays. However, the results also indicate that RMCs implemented on *forthcoming* SmartNICs would yield decreased delays for workloads that issue repeated host memory accesses.

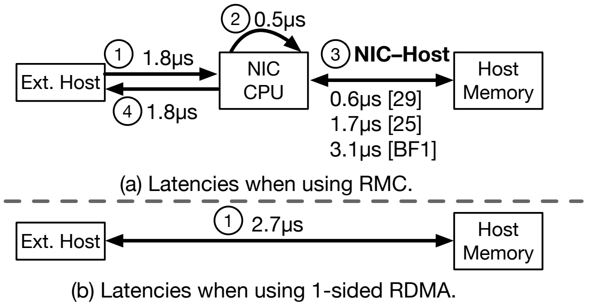


Figure 2: *Latencies used when modeling an n-element linked list scan. Arrows indicate direction of data movement, double arrows indicate round-trip times (RTTs). We use three possible values of NIC-Host RTTs: (a)* 0.6µs*, the DMA RTT reported for NFP6000 [29]; (b)* 1.7µs*, the DMA RTT reported for LiquidIOII [25]; and (c)* 3.1µs*, the measured RTT for Bluefield-1.*

4.1 Scanning data structures

Many distributed RDMA applications need to scan lists of data in order to find a specific value. For example, key-value stores might need to search an index to find a key, and hashmaps use linked lists to store values whose keys hash to the same bucket. When scanning *n* elements, an application using 1-sided operations must make *n* 1-sided requests, whereas with RMCs a single `RMCInvoke` call suffices.

We model the scan of a remote linked list in a SmartNIC by using the `search_lnkdlist` RMC from Figure 1, and we break down the model latencies as Figure 2a illustrates. In Step 1, the client invokes the RMC by sending 24-bytes: 8-bytes for the RMC id, 8-bytes for the head of the list, and 8-bytes for the scan key. We use a latency of 1.8µs for this step, as this was the value we measured between a server[7] and a 100Gb/s BF1 NIC connected back-to-back. For Step 2, we assume a 0.5µs scheduling delay as previously stated. Next, since each linked list node size is 24-bytes, the RMC performs *n* 24-byte reads from host memory through PCIe (i.e., Step 3). In this step we consider three different round-trip times (RTTs) from the SmartNIC CPU to host memory (**NIC-Host** in Figure 2a): **(1)** 0.6µs, which is the DMA RTT reported for small accesses when using NFP6000 NICs [28, 29]; **(2)** 1.7µs, another DMA RTT reported by iPipe when using LiquidIOII NICs [25]; and **(3)** 3.1µs, the RTT we measured from the BF1 CPU to host memory using RDMA. Once the node with the key is found, Step 4 returns the value to the client.

As Figure 2 shows, in Bluefield-1 the round-trip time of 1-sided RDMA reads from another host is lower (2.7µs) than the RTT from the BF1 CPU (3.1µs) to its host memory: this matches previously reported results from iPipe [25] for small memory accesses. We hypothesize this is due to the use of ARM cores with slow frequencies, and a non-optimized RDMA library, but we expect this will be resolved in future SmartNICs (see §3.2).

[7] Equipped with Intel Xeon Sandy Bridge E5-2640 v3 CPUs running at 2.6Ghz.

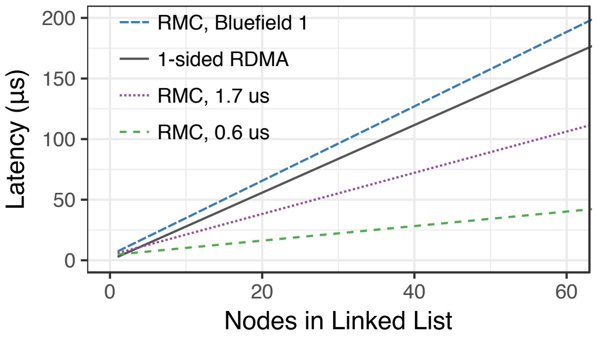


Figure 3: *Latency comparison in a model that scans a remote linked list. 1-sided issues as many RDMA reads as there are nodes, while RMC reads the nodes through PCIe. When PCIe round-trip time is lower than a network RTT, it is feasible to improve latencies compared to 1-sided.*

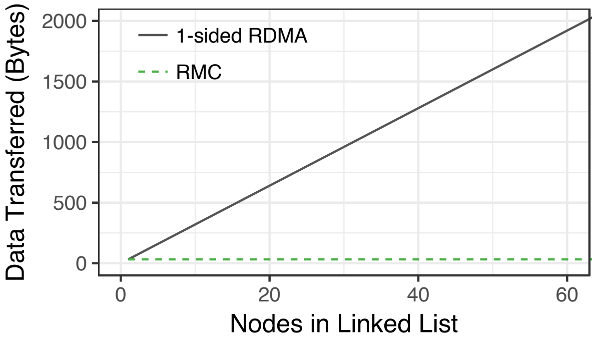


Figure 4: *Data transferred through the network in a model that scans a remote linked list. RMC moves a constant number of bytes, while 1-sided needs to move data proportional to the number of scanned nodes.*

Figure 3 shows the results of using our scan model with the three **NIC-Host** round-trip times previously mentioned. As discussed before, 1-sided RDMA outperforms RMCs when using BF1. In contrast, both DMA-based RMC results show better latency relative to 1-sided when scanning lists with 4 or more nodes, and the latency gap between them and 1-sided reads increases as a function of the list length. As such, these idealized SmartNICs perform better than 1-sided RDMA, showing the promise of RMCs. For the same model, Figure 4 shows the amount of data transferred over the network. RMCs allow the system to transfer a constant amount of data (24 bytes for the request, 8 bytes for the result), while 1-sided operations require transferring an increasing amount of data, dependent on the linked list length.

4.2 Hashing a remote buffer

Next we consider a workload where an application needs to read a large amount of data, compute a smaller value from it (e.g., a hash), and then make use of the computed digest. This pattern is common in applications that verify data integrity or deduplicate data.

Implementing this workload using 1-sided RDMA requires transferring the entire buffer to the client in order to compute the hash. Using RMCs, the hash can be computed directly on the server NIC; many SmartNIC CPUs, including BF1, have specialized instructions to compute hashes (ARM's Neon extensions [3]). In our model we omit hash computation time at both the client and the SmartNIC because transmission time dominates for the large buffers we consider.

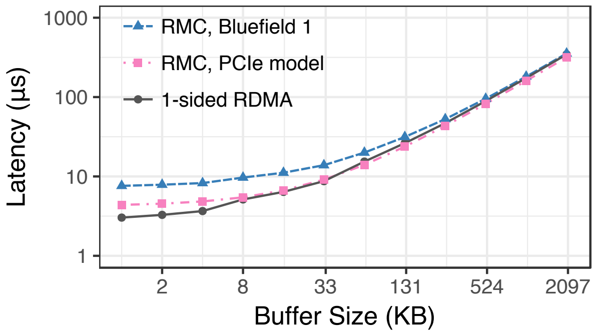


Figure 5: *Modeled latency when reading and hashing a remote buffer from a host (1-sided RDMA) vs. using RMCs.*

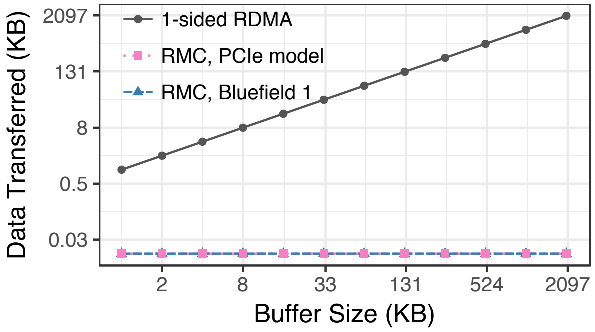


Figure 6: *Data transferred through the network when hashing a remote buffer. With regular 1-sided RDMA reads, the complete buffer needs to be moved to the client host, whereas with RMCs the buffer is moved through PCIe, so only the resulting hash is moved through the network.*

Figure 5 plots the latency for this scenario as a function of data size. The BF1 results (i.e., "RMC, Bluefield 1") report the latency we measured when issuing a host memory read of the given buffer size, and we use PCIe bandwidth (from [29]) to model the idealized SmartNIC (i.e., "RMC, PCIe Model"). As before, the 1-sided request does not encounter the 0.5µs scheduling delay, while the others do. We observe that 1-sided RDMA has comparable latency for the entire range of buffer sizes we consider. We also observe that, despite its limitations with small memory accesses, once access size grows sufficiently, RMCs on BF1 can provide comparable latency to 1-sided operations. As expected, Figure 6 shows only a constant amount of data traverses the network when using RMCs to compute hashes at the server. Prior work [27, 35, 48] has discussed many of the challenges of using RDMA in congested networks. In this case, the use of RMCs reduces the amount of bulk data transferred over the network fabric, thereby reducing congestion.

5 Implications of RMCs

5.1 Application design

So far we have presented RMCs as a way to make 1-sided RDMA operations more flexible, and therefore, more widely applicable. However, there is still an important role for 2-sided operations. The tradeoffs between the various ways of implementing an application (i.e., using

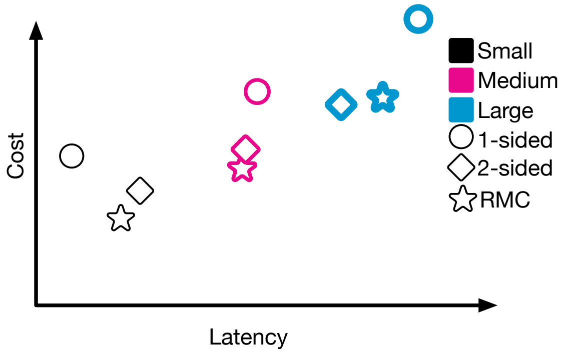


Figure 7: *Total latency vs. total cost (CPU resources and network utilization) when accessing remote memory N times within a computation. Shapes show RDMA operation type, colors indicate size of N.*

traditional 1-sided operations, RMCs, or 2-sided operations) involve weighing two main factors. The first is performance, which of course will depend on the details of the application and the nature of the underlying hardware. The second is the "cost" of the resources used; these resources include the client core, the server core, the server's NIC core, and network usage. The costs assigned to these depend on both the long-term design strategies (e.g., some hyperscale cloud operators view host cores as being far more precious than NIC computing resources, while others do not), and the short-term conditions where some resources may be scarce, raising their cost.

To illustrate these tradeoffs, Figure 7 compares different implementation options for an application whose access pattern requires N remote memory accesses. In this illustration we make three assumptions: **(1)** SmartNIC cores are less powerful (i.e., have slower clock speeds, shallower pipelines, etc.), and are therefore "cheaper" than host cores; **(2)** network costs are proportional to data transferred over the network; and **(3)** CPU costs are proportional to execution time.

Given these assumptions, when N is small, 1-sided operations provide lower latency than both RMCs and 2-sided operations because the client CPU, which initiates the requests and processes the resulting values, is the only processor involved. In contrast, RMCs require processing at the SmartNIC CPU, and 2-sided operations require processing at the host CPU; participation of these additional CPUs adds latency for small accesses. However, as N grows, time spent transferring data over the network dominates any time saved by avoiding remote CPUs, so 1-sided RDMA operations show increased latency when compared to the alternatives. FaSST [21] and others have observed the same.

When N is small, RMCs may begin executing sooner compared to 2-sided operations, as we expect the SmartNIC CPUs to be dedicated for the RMC runtime system. In contrast, host CPUs are likely to be shared with other processes; i.e., host CPUs are more expensive, so dedicating cores to RMC processing has a higher cost. As a result, scheduling an RMC on a SmartNIC CPU could take less time compared to the host CPU. The fact that RMCs may start running sooner potentially improves completion time when N is small. However, as N grows, the performance gap between SmartNIC CPUs and host CPUs becomes more important, and for large N, processing time and memory access bandwidth are likely to dominate, which would result in RMCs producing higher latencies compared to 2-sided operations.

In terms of costs: 1-sided operations are more expensive compared to both RMCs and 2-sided operations because they require use of client host CPUs and network fabric. Additionally, since the amount of data transferred over the network and processing time increase with larger N, 1-sided operations show the fastest growth in cost. Network utilization for RMCs and 2-sided operations is independent of N, and CPU time is their main cost source. In this case RMCs are cheaper than 2-sided operations because of our assumption that SmartNIC cores are cheaper than host cores. However, since runtime when using RMCs grows faster than when using 2-sided RDMA, RMC costs grow faster as well.

5.2 Navigating RDMA tradeoffs

RDMA applications have always had to navigate the trade-off between 1-sided and 2-sided RDMA, and the decision about which to use has so far been made by application developers. For example, implementing applications that use 1-sided RDMA requires developers to choose data layouts and algorithms that are amenable to such operations (e.g., trading network round trips for read size [10]). On the other hand, applications that use 2-sided operations must include a server component [21]. While the choice between 1-sided and 2-sided operations is made when the application is designed, the relative merits of each vary based on where an application is deployed–which dictates the relative cost of host CPU cycles, SmartNIC CPU cycles, and network transfers–and can even vary over time due to changing demands for processing power and network capacity. Therefore, choosing between these modes dynamically at runtime is more desirable than choosing at development time.

RMCs–as an abstraction–presents a third option for applications, and can enable runtime decisions about how to access remote memory. This is because RMCs could be executed not only at the server's SmartNIC (our focus so far), but also at the server's CPU (resembling 2-sided operations with local memory access), at the client's SmartNIC, or at the client's CPU (resembling 1-sided operations). Thus, the location of an RMC could be controlled by its API, and it would dictate the type of memory access required (i.e., the underlying access mechanism for `readhost()` could be adaptive), so moving from 1-sided, to RMCs, to 2-sided, would require no application changes and could therefore be automated. The ability to switch between these modes could enable the use of schedulers that could account for instantaneous resource demands and deployment costs when choosing RMC locations, enabling greater efficiency and application performance. We believe this dynamic choice is a significant benefit provided by RMCs, and we plan to explore this direction in future work.

Acknowledgements

We thank our shepherd Radhika Niranjan Mysore, and the anonymous reviewers for their helpful comments. This work was funded in part by NSF Grants 2029037, 2028832, 2028771, 2006349, 1817115, 1817116, and 1704941, and by grants from Intel, VMware, Ericsson, Futurewei, and Cisco.

References

[1] M. K. Aguilera, K. Keeton, S. Novakovic, and S. Singhal. Designing far memory data structures: Think outside the box. In *Workshop on Hot Topics in Operating Systems*, HotOS'19, pages 120–126, 2019.

[2] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker. Can far memory improve job throughput? In *European Conference on Computer Systems*, EUROSYS'17, pages 1–16, 2020.

[3] ARM. Neon programmer guides for armv8-a, Accessed 2020/06/10. https://developer.arm.com/architectures/instruction-sets/simd-isas/neon.

[4] P. Bailis. Communication Costs in Real World Networks, Accessed 2020/06/10. http://www.bailis.org/blog/communication-costs-in-real-world-networks/.

[5] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan. Attack of the killer microseconds. *Communications of the ACM*, 60(4):48–54, 2017.

[6] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility safety and performance in the spin operating system. In *ACM Symposium on Operating Systems Principles*, SOSP'95, pages 267–283, 1995.

[7] The ccix consortium, Accessed 2020/09/24. https://www.ccixconsortium.com/.

[8] Compute express link, Accessed 2020/09/24. https://www.computeexpresslink.org/.

[9] J. Do, S. Sengupta, and S. Swanson. Programmable solid-state storage in future cloud datacenters. *Communications of the ACM*, 62(6):54–62, 2019.

[10] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. Farm: Fast remote memory. In *Symposium on Networked Systems Design and Implementation*, NSDI'14, pages 401–414, 2014.

[11] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *Symposium on Networked Systems Design and Implementation*, NSDI'18, pages 51–66, 2018.

[12] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker. Network requirements for resource disaggregation. In *Symposium on Operating Systems Design and Implementation*, OSDI'16, pages 249–264, 2016.

[13] The gen-z consortium, Accessed 2020/09/24. https://genzconsortium.org/.

[14] E. Gershuni, N. Amit, A. Gurfinkel, N. Narodytska, J. A. Navas, N. Rinetzky, L. Ryzhyk, and S. Sagiv. Simple and precise static analysis of untrusted linux kernel extensions. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI'19, 2019.

[15] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin. Efficient memory disaggregation with infiniswap. In *Symposium on Networked Systems Design and Implementation*, NSDI'17, pages 649–667, 2017.

[16] K. Hamidouche, A. Venkatesh, A. A. Awan, H. Subramoni, C.-H. Chu, and D. K. Panda. Exploiting gpudirect rdma in designing high performance openshmem for nvidia gpu clusters. In *IEEE Transactions on Parallel and Distributed Systems*, TPDS'15, pages 78–87, 2015.

[17] G. C. Hunt and J. R. Larus. Singularity: rethinking the software stack. *ACM SIGOPS Operating Systems Review*, 41(2):37–49, 2007.

[18] R. Imaoka. Using ping to test AWS VPC network latency within a single region, Accessed 2020/06/10. https://richardimaoka.github.io/blog/network-latency-analysis-with-ping-aws/.

[19] K. Kaffes, T. Chong, J. T. Humphries, A. Belay, D. Mazières, and C. Kozyrakis. Shinjuku: Preemptive scheduling for μsecond-scale tail latency. In *Symposium on Networked Systems Design and Implementation*, NSDI'19, pages 345–360, 2019.

[20] A. Kalia, M. Kaminsky, and D. G. Andersen. Using rdma efficiently for key-value services. In *ACM Special Interest Group on Data Communications*, SIGCOMM'14, pages 295–306, 2014.

[21] A. Kalia, M. Kaminsky, and D. G. Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided (rdma) datagram rpcs. In *Symposium on Operating Systems Design and Implementation*, OSDI'16, pages 185–201, 2016.

[22] A. Li, S. L. Song, J. Chen, X. Liu, N. Tallent, and K. Barker. Tartan: evaluating modern gpu interconnect via a multi-gpu benchmark suite. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 191–202. IEEE, 2018.

[23] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang. Kv-direct: High-performance in-memory key-value store with programmable nic. In *ACM Symposium on Operating Systems Principles*, SOSP'17, pages 137–152, 2017.

[24] S. Li, H. Lim, V. W. Lee, J. H. Ahn, A. Kalia, M. Kaminsky, D. G. Andersen, O. Seongil, S. Lee, and P. Dubey. Architecting to achieve a billion requests per second throughput on a single key-value store server platform. In *International Symposium on Computer Architecture*, ISCA'15, pages 476–488, 2015.

[25] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta. Offloading distributed applications onto smartnics using ipipe. In *ACM Special Interest Group on Data Communications*, SIGCOMM'19, 2019.

[26] C. Mitchell, Y. Geng, and J. Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *USENIX Annual Technical Conference*, ATC'13, 2013.

[27] R. Mittal, V. T. Lam, N. Dukkipati, E. R. Blem, H. M. G. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. Timely: Rtt-based congestion control for the datacenter. In *ACM Special Interest Group on Data Communications*, SIGCOMM'15, 2015.

[28] Netronome. Nfp-6000 intelligent ethernet controller family, Accessed 2020/06/10. https://www.netronome.com/static/app/img/products/silicon-solutions/PB_NFP6000.pdf.

[29] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore. Understanding pcie performance for end host networking. In *ACM Special Interest Group on Data Communications*, SIGCOMM'18, pages 327–341, 2018.

[30] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker. Netbricks: Taking the v out of nfv. In *Symposium on Operating Systems Design and Implementation*, OSDI'16, pages 203–216, 2016.

[31] Pci-sig specifications library, Accessed 2020/09/24. https://pcisig.com/specifications.

[32] D. A. Popescu. *Latency-driven performance in data center.* PhD thesis, University of Cambridge, 2019.

[33] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. Legoos: A disseminated, distributed os for hardware resource disaggregation. In *Symposium on Operating Systems Design and Implementation*, OSDI'18, pages 69–87, 2018.

[34] J. Shi, Y. Yao, R. Chen, H. Chen, and F. Li. Fast and concurrent rdf queries with rdma-based distributed graph exploration. In *Symposium on Operating Systems Design and Implementation*, OSDI'16, 2016.

[35] A. Shpiner, E. Zahavi, V. Zdornov, T. Anker, and M. Kadosh. Unlocking credit loop deadlocks. 2016.

[36] D. Sidler, Z. Wang, M. Chiosa, A. Kulkarni, and G. Alonso. Strom: smart remote memory. In *European Conference on Computer Systems*, EUROSYS'20, pages 1–16, 2020.

[37] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra. Near-memory computing: Past, present, and future. *Microprocess. Microsystems*, 71, 2019.

[38] M. Technologies. Mellanox innova-2 flex open programmable smartnic, Accessed 2020/06/10. https://www.mellanox.com/sites/default/files/doc-2020/pb-innova-2-flex.pdf.

[39] M. Technologies. Nvidia mellanox bluefield-1 smartnic, Accessed 2020/06/10. https://www.mellanox.com/files/doc-2020/pb-bluefield-smart-nic.pdf.

[40] M. Technologies. Nvidia mellanox bluefield-2 smartnic, Accessed 2020/06/10. https://www.mellanox.com/files/doc-2020/pb-bluefield-2-smart-nic-eth.pdf.

[41] M. Technologies. Rdma aware networks programming user manual, Accessed 2020/06/10. https://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf.

[42] L. A. Torrey, J. Coleman, and B. P. Miller. A comparison of interactivity in the linux 2.6 scheduler and an mlfq scheduler. *Software - Practice and Experience*, 37:347–364, 2007.

[43] P. R. . A. Vahdat. Plotting a Course to a Continued Moore's Law - Keynote, Accessed 2020/06/10. https://youtu.be/6wq6g_vi6yw.

[44] K. Vipin and S. A. Fahmy. Fpga dynamic and partial reconfiguration: a survey of architectures, methods, and applications. *ACM Computing Surveys (CSUR)*, 51(4):1–39, 2018.

[45] H. Wang, S. Potluri, D. Bureddy, C. Rosales, and D. K. Panda. Gpu-aware mpi on rdma-enabled clusters: Design, implementation and evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 25:2595–2605, 2014.

[46] Xilinx. Xilinx alveo u280, Accessed 2020/06/10. https://www.xilinx.com/publications/product-briefs/alveo-u280-product-brief.pdf.

[47] J. Xue, Y. Miao, C. Chen, M. Wu, L. Zhang, and L. Zhou. Fast distributed deep learning over rdma. In *European Conference on Computer Systems*, EUROSYS'19, 2019.

[48] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang. Congestion control for large-scale rdma deployments. In *ACM Special Interest Group on Data Communications*, SIGCOMM'15, 2015.