Design Guidelines for High Performance RDMA Systems

ANUJ KALIA, MICHAEL KAMINSKY, AND DAVID G. ANDERSEN

Anuj Kalia is a PhD student in the Computer Science Department at Carnegie Mellon University. He is interested in networked systems, especially using high-speed networks to build distributed systems. akalia@cs.cmu.edu

Michael Kaminsky is a Senior Research Scientist at Intel Labs and an adjunct faculty member of the Computer Science Department at Carnegie Mellon University. He is part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), based in Pittsburgh, PA. His research interests include distributed systems, operating systems, and networking. michael.e.kaminsky@intel.com

David G. Andersen is an Associate Professor of Computer Science at Carnegie Mellon University. He completed his SM and PhD degrees at MIT, and holds BS degrees in biology and computer science from the University of Utah. In 1995, he co-founded ArosNet, an ISP in Salt Lake City, Utah. dga@cs.cmu.edu

Modern RDMA hardware offers the potential for exceptional performance, but achieving this performance is challenging. Directly mapping an application's low-level reads and writes to RDMA primitives is often suboptimal, and design choices, including which RDMA operations to use and how to use them, significantly affect observed performance. We lay out guidelines that can be used by system designers to navigate the RDMA design space. Our guidelines emphasize paying attention to low-level details such as individual RDMA packets, PCIe transactions, and NIC architecture. We present two case studies—a key-value store and a networked sequencer—demonstrating the effectiveness of these guidelines.

In recent years, new entrants into the datacenter and cluster networking space have started to provide hardware capabilities formerly available only in expensive High Performance Computing (HPC) interconnects. The NICs (network interface cards) from manufacturers such as Mellanox now support RDMA (remote direct memory access) features out of the box, at a price comparable to non-RDMA-capable NICs.

The "RDMA Background" section describes RDMA in more detail, but at a high level RDMA is a networking approach consisting of two basic concepts:

1. Operating system "stack bypass": In many applications, the overhead of going through the kernel networking layers is the bottleneck to processing speed. This is particularly the case with applications that send and receive relatively small amounts of data per packet exchange, but do so at high rates.
2. Full CPU bypass: For certain, more-specialized applications, RDMA hardware can allow one computer to read and write directly to/from the memory of another node in the cluster, without the remote node's CPU or OS being involved at all.

RDMA has been a key ingredient of HPC and supercomputing environments for years, but it is also intriguing to datacenter application developers. RDMA hardware presents programmers with numerous choices, so using it efficiently requires care. For example, should applications provide reliability, or should the NIC's reliability protocol be used? In the rest of this article, we help readers navigate this space to understand what RDMA capabilities might be the best match for their application. Our open-source rdma_bench toolkit (https://github.com/efficient/rdma_bench) can be used for evaluating and optimizing the most important system factors that affect end-to-end throughput when using RDMA.

RDMA Background

RDMA is a general approach to networking for which several different models exist. The most popular model is the Virtual Interface Architecture (VIA) [3]. Other models include open-source specifications such as Portals from Sandia Labs, and proprietary HPC architectures such as Fujitsu's Tofu interconnect. For a given model, there can be more than one implementation. For example, VIA has three well-known implementations: InfiniBand, RoCE (RDMA over Converged Ethernet), and iWARP (Internet Wider Area RDMA Proto-

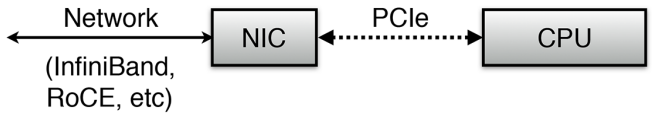


Figure 1: Hardware components of a node in an RDMA cluster

Verb	Abstract API function call
WRITE READ	write(qp, local_buf, size, remote_addr) read(qp, local_buf, size, remote_addr)
SEND RECV	send(qp, local_buf, size) recv(qp, local_buf, size)

Table 1: Abstract RDMA API showing one-sided (WRITE, READ) and two-sided (SEND, RECV) verbs

col). Our work focuses on VIA-based NICs, which are the only commodity NICs currently available. Several observations are applicable to other architectures as well. Table 1 shows an abstract RDMA API for VIA NICs.

Compared to conventional Ethernet-based TCP/IP networking, RDMA networks remove several sources of CPU overhead. They support user-level NIC access, removing the overhead of the kernel's heavyweight networking stack. At the remote host, RDMA reads and writes bypass the CPU. RDMA NICs typically implement a reliable transport layer, freeing up host CPU cycles used for implementing a reliable protocol such as TCP/IP. RDMA-capable networks with 100 Gbps of per-port bandwidth, and 2 µs round-trip latency are commercially available.

Figure 1 shows the relevant hardware components of a machine in a modern RDMA cluster. A NIC with one or more ports connects to the PCIe controller of a multicore CPU, and provides direct access to the memory of remote nodes.

Verb types: *One-sided* verbs (RDMA operations) operate directly on a remote node's memory, bypassing its CPU, and include RDMA reads, writes, and atomic operations. *Two-sided* verbs include the send and receive verbs; their functionality resembles `send()` and `recv()` functions in traditional sockets programming. These verbs involve the responder's CPU: the send's payload is written to an address specified by a receive that was posted previously by the responder's CPU.

The choice of verbs is a key determinant of application performance, but it is not the only factor. The choice of transport and the verb initiation method (discussed below) require equal attention.

Queue pairs: RDMA hosts communicate by posting verbs to interfaces called queue pairs (QPs). On completing a verb, the requester's NIC optionally signals completion by writing a completion queue entry to host memory via direct memory access (DMA).

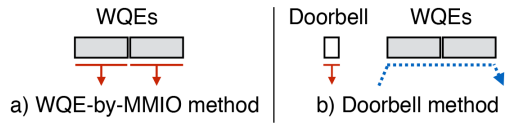


Figure 2: The WQE-by-MMIO and Doorbell methods for transferring two WQEs. Arrows represent PCIe transactions. Solid arrows are PCIe MMIO writes; the dashed arrow is a PCIe DMA read. Arrow width represents transaction size.

RDMA transports are either reliable or unreliable and are either connected or unconnected (also called datagram). With reliable transports, the NIC uses acknowledgments to guarantee in-order delivery of messages. Unreliable transports do not provide this guarantee. However, modern high-speed networks such as InfiniBand and RoCE use a reliable link layer, so unreliable transports do not lose packets due to congestion or bit errors. Connected transports require one-to-one connections between QPs, whereas a datagram QP can communicate with multiple QPs. Datagram transport is more scalable, but it only supports two-sided verbs.

Current RDMA transports include Reliable Connected (RC), Unreliable Connected (UC), and Unreliable Datagram (UD). Note that although these transports resemble non-RDMA transport layers to some extent (e.g., RC and UD are analogous to TCP and UDP, respectively), the underlying protocol and message formats are different.

Verb initiation: To initiate RDMA operations, the user-mode NIC driver at the requester creates work queue elements (WQEs) in host memory. These WQEs are transferred to the NIC over the PCIe bus in one of two ways. In the "WQE-by-MMIO" method, the CPU directly writes the WQEs to device memory using memory-mapped I/O (MMIO). In the "Doorbell" method, the CPU writes a short Doorbell message to the NIC, indicating the new WQEs. This action is called "ringing the Doorbell." On receiving the Doorbell, the NIC reads the WQEs from the CPU via a DMA read. Both methods bypass the host's OS kernel. Figure 2 summarizes the two methods. The Doorbell method reduces CPU use: it requires one MMIO for a batch of WQEs, whereas WQE-by-MMIO requires separate MMIOs for each WQE.

Factors Affecting RDMA System Performance

In datacenters, RDMA is being proposed for use in key-value stores, graph processing systems, and online transaction processing systems [1, 2]. These applications access irregular data structures (e.g., hash tables and trees) and use small packet sizes on the order of tens of bytes. Three main factors are important for high performance with these workloads: the extent to which remote CPU bypass is used, low-level optimizations for verbs, and the NIC architecture.

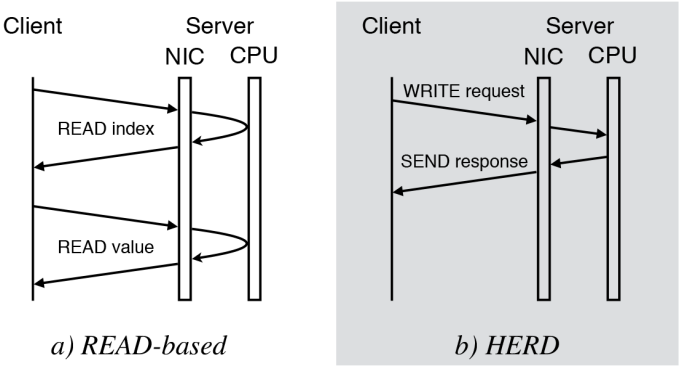


Figure 3: Messages for a GET operation in (a) a READ-based key-value store and (b) HERD

Remote CPU Bypass

In an RDMA-based datastore, data is stored in the memory of a server machine and is accessed by client machines. To take advantage of RDMA's ability to bypass the remote CPU, several projects use one-sided READs and WRITEs to accomplish this. For datastore GET operations, they do so by traversing the remote data structure using READs. This typically requires multiple round trips. For example, Pilaf [6] is an RDMA-based key-value store that handles key-value GET operations in 2.6 READs on average. It uses 1.6 READs to access its hash table–based index to locate the value's address, and one READ to fetch the value. FaRM's key-value store [1] reduces the number of READs required for the index from 1.6 to 1.

In contrast, the design of our HERD system [4] is focused on reducing the number of round trips to one. To accomplish this, HERD does not entirely bypass the remote CPU. Instead of traversing the remote data structure, HERD clients send their requests to the server using an RPC request. The HERD server traverses the data structure for the client, but it does so in local memory.

Local memory accesses are 10x–100x faster than remote accesses in latency, bandwidth, and the amount of host CPU they consume. Then the server sends a response to the client. Several combinations of verbs and transports can be used to implement fast RPCs; HERD uses a combination of one-sided and two-sided verbs over unreliable transport, which is optimized for high performance and number-of-clients scalability. Figure 3 shows the difference in HERD and READ-based key-value stores.

HERD's RPC mechanism is very fast: its throughput and latency is similar to RDMA reads. As a result, HERD delivers higher throughput and lower latency than Pilaf or FaRM's key-value store. Key to achieving high RPC throughput is the use of the low-level optimizations discussed below. An important lesson from HERD is that one-sided RDMA is not always the best solution; using an RDMA network simply for fast, OS kernel–bypass RPCs is an equally important design to consider.

Low-Level Verb Optimizations

RDMA verbs allow for a variety of low-level optimizations. Effectively using these optimizations requires a good understanding of how different verbs use the CPU, the PCIe bus, and the RDMA network, and how this varies when different optimizations are enabled. Our USENIX ATC paper [5] addresses this topic thoroughly. We discuss the most important optimizations briefly here.

Unreliable transports reduce NIC overhead by not requiring RDMA acknowledgment packets, and provide higher performance than reliable transports. Unreliable transports do not provide reliable packet delivery. However, modern RDMA implementations such as InfiniBand use a reliable link layer, so even unreliable transports drop packets extremely rarely.

Payload inlining reduces NIC processing and PCIe bandwidth use by eliminating the DMA read for the payload. By default, WRITEs and SEND work queue elements contain a pointer to the payload; the NIC reads it via a DMA read. Small payloads up to a few hundred bytes can be encapsulated inside the WQE and written to the NIC using MMIO.

Selective signaling also reduces NIC processing and PCIe bandwidth use by eliminating the completion DMA. By default, the NIC writes a completion queue entry to host memory on completing a verb. If an application can detect completion using alternate methods (e.g., using a future response from a remote node), it can mark the verb as "unsignaled," instructing the NIC to not issue the completion DMA.

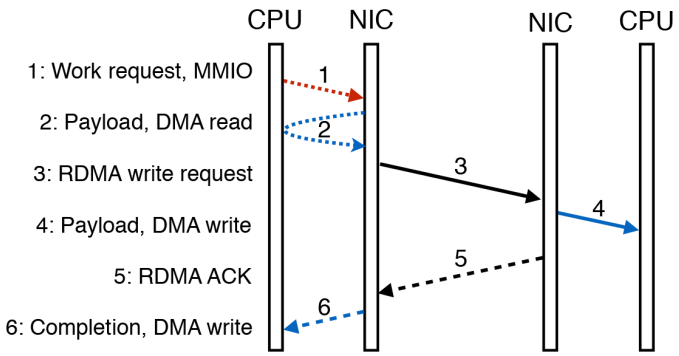
Doorbell batching reduces CPU use and PCIe bandwidth use by allowing applications to issue a batch of RDMA operations using one Doorbell. This reduces CPU use by requiring only one MMIO per batch. The default approach of transferring WQEs one by one using WQE-by-MMIO requires separate MMIOs for each WQE.

Figure 4a and Figure 4b show the PCIe and RDMA network messages for one WRITE without optimizations, and two WRITEs with the above optimizations, respectively.

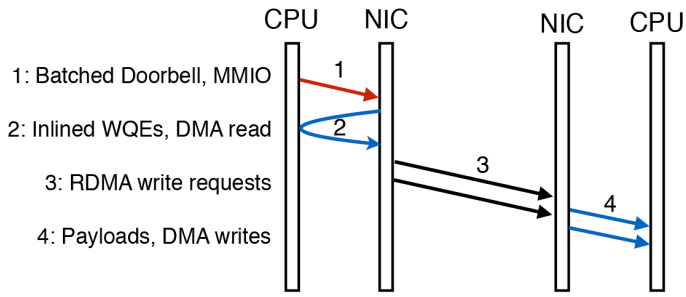
NIC Architecture

Modern high-speed NICs are composed of multiple processing units (PUs), such as packet processing engines and DMA engines. Exploiting parallelism among the NIC's PUs is necessary for high performance but requires explicit attention. Further, RDMA verbs and workloads that introduce contention between the PUs should be avoided.

Engage multiple NIC PUs: A common RDMA programming decision is to use as few queue pairs as possible, but doing so limits NIC parallelism to the number of QPs. This is because operations on the same QP have ordering dependencies and are ideally handled by the same NIC processing unit to avoid cross-



(a) RDMA and PCIe messages for one RDMA write



(b) RDMA and PCIe messages for two WRITEs with optimizations

Figure 4: Effect of optimizations on RDMA and PCIe messages. The optimized WRITEs are unreliable, inlined, unsignaled, and are issued using a batched Doorbell. The dashed messages are removed in the optimized version; the dotted messages are repurposed.

PU synchronization. For example, in datagram-based RDMA communication, one QP per CPU core is sufficient for communication with all remote cores. However, it "binds" a CPU core to a PU and may limit core throughput to PU throughput. This is likely to happen when per-message application processing is small and a high-speed CPU core overwhelms a less powerful PU. In such cases, using multiple QPs per core increases CPU efficiency; we call this the *multi-queue* optimization.

Avoid contention among NIC PUs: RDMA operations that require cross-QP synchronization introduce contention among PUs, and can perform over an order of magnitude worse than uncontended operations. For example, RDMA provides atomic operations such as compare-and-swap and fetch-and-add on remote memory. To our knowledge, all commodity NICs available at the time of writing use internal concurrency control for atomics: PUs acquire an internal lock for the target address and issue read-modify-write over PCIe. Therefore, the NIC's internal locking mechanism, such as the number of locks and the mapping of atomic addresses to these locks, is important. Note that due to the limited SRAM in NICs, the number of available locks is small, which amplifies contention in the workload.

Case Studies

We now describe the design of two high-performance RDMA-based systems: the HERD key-value store and the X-Seq sequencer. X-Seq is named Spec-S0 in our USENIX ATC paper [5].

RPC Overview

For both systems, we use an RPC protocol for communication between clients and the key-value/sequencer server. In HERD, clients use unreliable WRITEs to write requests to a request memory region at the server. A server thread (a *worker*) checks for new requests from every client by polling on the request memory region, and collects a batch of requests. It computes a batch of responses, and sends them to clients using the SEND verb over datagram transport. The worker uses a batched Doorbell for the batch of response SENDs. To use the multi-queue optimization, each worker alternates among a configurable number (1–3) of datagram QPs across batches of response SENDs. In addition to unreliable transport, Doorbell batching, and multi-queue, the server also uses payload inlining and selective signaling.

Key-Value Stores

Figure 5 shows the throughput of a HERD key-value store server with an increasing number of server CPU cores. We use a cluster with Mellanox Connect-IB InfiniBand NICs, and 14-core Intel CPUs. The key-value store maps 16-byte keys to 32-byte values; the workload consists of keys chosen uniformly at random and 5% PUT operations. HERD's single-core throughput is 12.3 million operations/s (Mops), and its peak throughput is 98.3 Mops. At its peak, HERD is bottlenecked by PCIe bandwidth.

Figure 5 also compares HERD to a READ-based key-value store that requires two RDMA reads per GET operation. The Connect-IB NIC supports up to 120 million READs/s, so such a READ-based key-value store is limited to 60 Mops. HERD delivers up to 64%

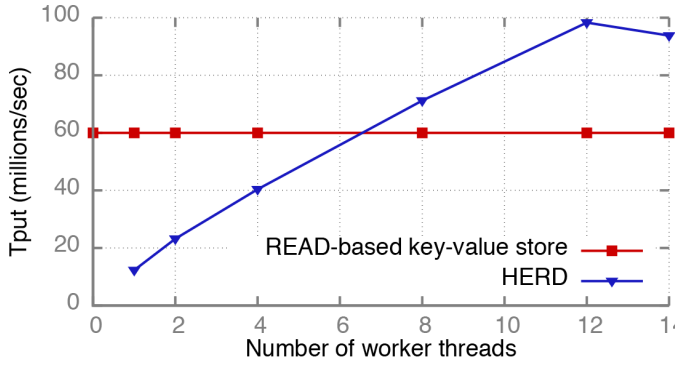


Figure 5: Throughput of HERD and READ-based key-value stores

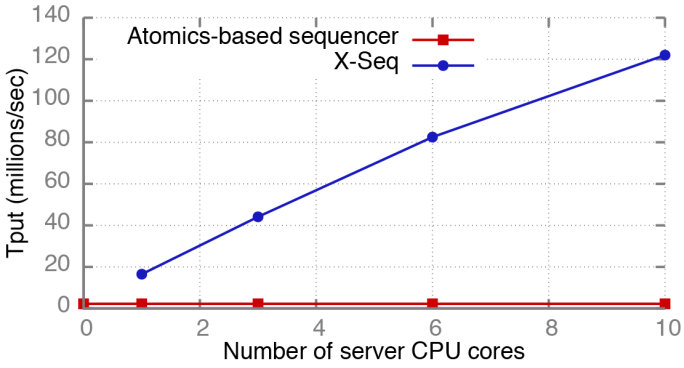


Figure 6: Throughput of X-Seq and atomics-based sequencers

higher throughput, while using a single round trip per operation. HERD uses significant server CPU resources: it requires at least seven CPU cores to outperform a READ-based design. However, HERD uses less client CPU than a READ-based store because it requires fewer client-initiated data transmissions.

Networked Sequencers

Centralized sequencers are useful building blocks for a variety of network applications, such as ordering operations in distributed systems via logical or real timestamps. A centralized sequencer can be the bottleneck in high-performance distributed systems, so building a fast sequencer is an important step to improving whole-system performance.

Our X-Seq sequence server runs on a single machine and provides an increasing eight-byte integer to client processes running on remote machines. The worker threads at the server share an eight-byte counter. After collecting a batch of *N* client requests, a worker thread atomically increments the shared counter by *N*, thereby claiming ownership of a sequence of *N* consecutive integers. It then sends these *N* integers to the clients using a batched Doorbell (one integer per client).

Directly adapting HERD's RPC protocol to a sequencer provides good performance. However, even higher throughput and scalability can be achieved by optimizing the RPCs specifically for the sequencer. The key insight is that the request and response packets in the sequencer are small, with up to eight bytes of data. This allows RPC optimizations that reduce the number of cache lines used by WQEs, and DMAs issued, by 50%. We also use a speculation technique where the clients speculate the most significant bytes of the current sequencer number.

Figure 6 shows the throughput of X-Seq with increasing server CPU cores. Its single-core throughput is 16.5 Mops, and its peak throughput is 122 Mops. Figure 6 also shows the throughput of a sequencer where clients use the atomic fetch-and-add verb to increment an eight-byte counter in the server's memory. This sequencer achieves only 2.24 Mops.

The poor performance of the atomics-based sequencer is because of lock contention among the NIC's processing units. The effects of contention are exacerbated by the *duration* for which locks are held—several hundred nanoseconds for PCIe round trips. Our RPC-based sequencers have lower contention and shorter lock duration: the programmability of general-purpose CPUs allows us to batch updates to the counter, which reduces cache line contention, and proximity to the counter's storage (i.e., core caches) makes these updates fast.

Code release: The code for our low-level RDMA benchmarks, HERD, and X-Seq is available at https://github.com/efficient/rdma_bench.

References

[1] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro, "FaRM: Fast Remote Memory," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)*, 2014.

[2] A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro, "No Compromises: Distributed Transactions with Consistency, Availability, and Performance," in *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, 2015.

[3] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, and C. Dodd, "The Virtual Interface Architecture," *IEEE Micro*, 1998, pp. 66–76.

[4] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA Efficiently for Key-Value Services," in *Proceedings of ACM SIGCOMM*, 2014, pp. 295–306.

[5] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design Guidelines for High-Performance RDMA Systems," in *Proceedings of the 2016 USENIX Annual Technical Conference (ATC '16)*.

[6] C. Mitchell, Y. Geng, and J. Li, "Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store," in *Proceedings of the 2013 USENIX Annual Technical Conference (ATC '13)*.

ENIGMA

MORE TO DECIPHER

It's time for the security community to take a step back and get a fresh perspective on threat assessment and attacks. This is why in 2016 the USENIX Association launched Enigma, a new security conference geared towards those working in both industry and research. Enigma will return in 2017 to keep pushing the community forward.

Expect three full days of high-quality speakers, content, and engagement for which USENIX events are known.

The full program and registration will be available in October.

enigma.usenix.org

JAN 30–FEB 1 2017

OAKLAND, CALIFORNIA, USA