Deconstructing RDMA-enabled Distributed Transactions: Hybrid is Better!

Xingda Wei, Zhiyuan Dong, Rong Chen, Haibo Chen
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Contacts: rongchen, haibochen@sjtu.edu.cn

Abstract

There is currently an active debate on which RDMA primitive (i.e., one-sided or two-sided) is optimal for distributed transactions. Such a debate has led to a number of optimizations based on one RDMA primitive, which was shown with better performance than the other.

In this paper, we perform a systematic comparison between different RDMA primitives with a combination of various optimizations using representative OLTP workloads. More specifically, we first implement and compare different RDMA primitives with existing and our new optimizations upon a single well-tuned execution framework. This gives us insights into the performance characteristics of different RDMA primitives. Then we investigate the implementation of optimistic concurrency control (OCC) by comparing different RDMA primitives using a phase-by-phase approach with various transactions from TPC-C, SmallBank, and TPC-E. Our results show that no single primitive (one-sided or two-sided) wins over the other on all phases. We further conduct an end-to-end comparison of prior designs on the same codebase and find none of them is optimal.

Based on the above studies, we build DrTM+H, a new hybrid distributed transaction system that always embraces the optimal RDMA primitives at each phase of transactional execution. Evaluations using popular OLTP workloads including TPC-C and SmallBank show that DrTM+H achieves over 7.3 and 90.4 million transactions per second on a 16-node RDMA-capable cluster (ConnectX-4) respectively, without locality assumption. This number outperforms the pure one-sided and two-sided systems by up to 1.89X and 2.96X for TPC-C with over 49% and 65% latency reduction. Further, DrTM+H scales well with a large number of connections on modern RDMA network.

1 Introduction

Distributed transactions with serializability and high availability provide a powerful abstraction to programmers with the illusion of a single machine that executes transactions with strong consistency and never fails. Although distributed transaction used to seem slow [19], the prevalence of fast networking features such as RDMA has boosted the performance of distributed transactions by orders of magnitudes [51, 5, 11, 18]. RDMA NIC (RNIC) provides high bandwidth, ultra-low latency datagram communication (two-sided primitive), together with offloading technology (one-sided primitive): the network card can directly access the memory of remote machines while bypassing kernel and remote CPUs.

Recently, there is an active debate over which RDMA primitive, namely one-sided or two-sided, is better suited for distributed transactions. One-sided primitive (e.g., `READ`, `WRITE`, and `ATOMIC`) provides higher performance and lower CPU utilization [10, 11, 51, 5]. On the other hand, two-sided primitive simplifies application programming and is less affected by hardware restrictions such as the limitation of RNIC's cache capacity [16, 18].

It is often challenging for system designers to choose the right primitive for transactions based on previous studies. Most work on RDMA-enabled transactions presents a new system built from scratch and compares its performance with previous ones using other codebases. Some only compare the performance of different primitives or designs using micro-benchmarks. This makes their results hard to interpret: differences in hardware configurations and software stacks affect the observable performance. Further, different RDMA primitives may significantly affect the overall performance [16, 17].

There have been several valuable studies in the database community in comparing different transactional systems [55, 13]. Harding et al. [13] conduct a comprehensive study on how different transaction protocols behave under different workloads in a distributed setting using a single framework. However, for a particular protocol, there may be many different implementations which have very different performance, especially when embracing new hardware features like RDMA.

In this paper, we conduct the first systematic study on how different choices of RDMA primitives and designs affect the performance of distributed transactions.[1] Unlike most previous research efforts which compare different overall systems, we compare different designs within

[1]Note that optimizing distributed transaction protocol is not the focus of this work.

a single execution framework. The goal is to provide a guideline on optimizing distributed transactions with RDMA, and potentially, for other RDMA-enabled systems (e.g., distributed file systems [26, 38] and graph processing systems [52, 36, 58]). In summary, this paper makes the following contributions:

A primitive-level comparison using a well-tuned RDMA execution framework (§4). We implement and tune an execution framework with all RDMA implementation techniques we know so far. We then systematically compare the performance of different primitives with existing and our newly proposed optimizations using micro-benchmarks that simulate common transactional workloads. The main results are the following (§4.2):

- One-sided primitive has better performance than two-sided with the same round trips.
- Two-sided primitive has better scalability with small payloads in large clusters.
- Two-sided primitive can be faster than one-sided when receiving ACK is done off the critical path.

A phase-by-phase evaluation of transactional execution (§5). We carefully study different primitives at different phases of transactional execution, including all optimizations proposed on both primitives, and then present their performance. More specifically, we focus on transactions with *optimistic concurrency control* (OCC)[2] for strong consistency and *primary-backup replication* for high availability. Nowadays OCC is widely used for transactions, from centralized databases [47, 49, 21] to distributed databases [11, 57, 24, 5, 18]. OCC is efficient and scalable on common workloads which stimulates many OCC-based RDMA-enabled transactions [11, 5, 18]. The protocol contains four steps: *Execution*, *Validation*, *Logging* and *Commit phase*. We show that *no single primitive always wins over the other*. To gain optimal performance for such phases, the main findings include:

- Using hybrid primitives for the execution (§5.1) and validation phases (§5.2).
- Using two-sided primitives for the commit phase (§5.3)
- Using one-sided primitives for the logging phase (§5.4).
- Using hybrid primitives and one-sided primitives for the read and validation phases of read-only transactions, respectively (§5.5).

[2]We use the shorter but more general term transactions to refer to distributed transactions executed using OCC in the rest of this paper.

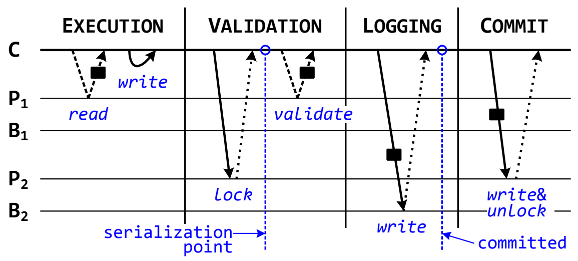

Fig. 1: *A phase-by-phase overview of transaction processing with OCC.* **C***,* **P***, and* **B** *stand for the coordinator, the primary and the backup of replicas, respectively.* $\boldsymbol{P}_1$ *is read and* $\boldsymbol{P}_2$ *is written. The dashed, solid, and dotted lines stand for read, write, and hardware ack operations, and rectangles stand for record data.*

An end-to-end study of existing and our new system on a single platform (§6). By further leveraging results from our phase-by-phase evaluations, we built `DrTM+H`, a hybrid design that optimizes every phase executed with appropriate primitives (§6.1). Evaluations using two popular OLTP workloads on a 16-node RDMA-capable cluster show that `DrTM+H` can perform over 7.3 and 90.4 million transactions per second for simplified `TPC-C` and `SmallBank` respectively. Further, our hybrid design does not suffer from scalability issues on an emulated 80-node connection setting (§6.2). Note that we do not make locality assumptions like previous work [18].

We finally make a comparable study on how previous systems leverage RDMA by evaluating three representative designs upon a single execution framework. To emulate previous systems, we choose the primitives and optimizations at each phase as the original design and implement them using the same codebase and transaction protocol. The experimental results show that none of them has the optimal performance (§6.3). Our hybrid design can outperform the pure two-sided design (FaSST) and the pure one-sided design (DrTM+R) by up to 2.96X and 1.89X for simplified `TPC-C`, respectively.

The source code of our execution framework and `DrTM+H`, including all benchmarks, are available at `https://github.com/SJTU-IPADS/drtmh`.

2 Background

2.1 RDMA and Its Primitives

RDMA (Remote Direct Memory Access) is a network feature with high speed, low latency, and low CPU overhead [10, 17]. It has generated considerable interests in applying it in modern datacenters [11, 46, 12]. RDMA is well known for its *one-sided primitive* including `READ`, `WRITE` and `ATOMIC` operations, which can directly access the memory of a remote machine without involving kernel and remote CPUs. Because RDMA bypasses the kernel and traditional network stack, RPC implementations over RDMA (*two-sided primitive*) can also have

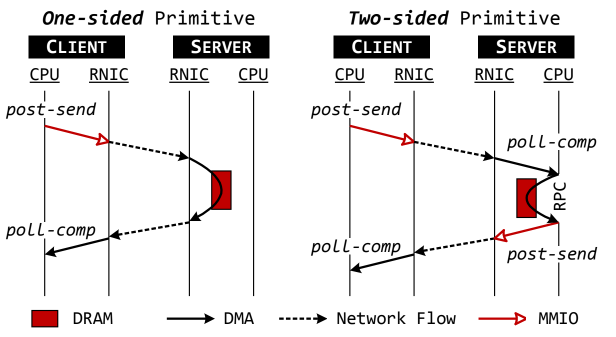

Fig. 2: *An overview of different RDMA primitives.*

orders of magnitude higher throughput than those using TCP/IP [10, 18].

Fig. 2 presents the workflow of these two primitives. No matter which primitive an application uses, the client (sender) uses a similar interface to post requests to (and poll results from) the server (receiver) via the RDMA-capable NIC (RNIC). The interface, called queue pairs (QPs), is used to communicate between the client (sender) and server (receiver). The client starts an RDMA request by posting the requests (called `Verbs`) to the sender queue, which can either be one-sided or two-sided verbs. The client can get the completion events of requests by polling a completion queue (CQ) associated with the QP. For two-sided primitives, the server polls requests from a receiver queue, calls a local RPC routine and posts results back to the sender queue.

Moreover, QPs have different transport modes which support different sets of primitives, as summarized in Table 1. The Reliable Connected (RC) mode supports all RDMA primitives, while the Unreliable Datagram (UD) mode only supports two-sided primitive (`SEND/RECV`). On the other hand, UD is connectionless so the application can use fewer UD QPs than RC QPs [18].

2.2 RDMA-enabled Distributed Transactions

There is an active line of research in using RDMA for serializable distributed transactions [11, 5, 18]. Most of such systems use variants of optimistic concurrency control (OCC) for consistency [22] and variants of primary-backup replication (PBR) for availability [23]. PBR uses fewer round trips and messages to commit one transaction than Paxos [11], which fits distributed transactions in a well-connected cluster.

Although these systems have different design choices and leverage different RDMA primitives, they use a similar transaction protocol (OCC)[3] to execute and commit serializable transactions. The operations performed in the protocol can be briefly summarized as four consecutive phases, as shown in Fig. 1. A transaction first executes by reading the records in its read set (**<u>E</u>xecution**). Then it executes a commit protocol, which locks the records in the write set and validates the records in the read set is unchanged (**<u>V</u>alidation**). If there is no conflicting transaction, the coordinator sends transaction's updates to each backup and waits for the accomplishment (**<u>L</u>ogging**). Upon successful, the transaction will be committed by writing and unlocking the records at the primary node (**<u>C</u>ommit**). Note that the execution order of the protocol is very important. For example, the transaction is considered to be committed if and only if the log replies have been received [11, 18]. Thus the commit phase must be executed after the completion of logging.

OCC can be directly used to execute read-only transactions, which is an important building block for modern applications [25]. A read-only transaction may use a two-phase protocol: the first phase reads all records (**<u>R</u>ead**), and then the second phase validates all of them have not been changed (**<u>V</u>alidation**).

[3]While DrTM [51] implements a two phase locking (2PL) scheme using HTM and RDMA, it provides no high availability support and a later version [5] uses a variant of OCC to provide high availability. We are not aware of other RDMA-enabled distributed transaction systems using 2PL. Hence, we focus on OCC in this paper.

Table 1: *Different transport modes of QP and supported operations.* ***RC****,* ***UC****, and* ***UD*** *stand for Reliable Connection, Unreliable Connection, and Unreliable Datagram, respectively.*

	SEND/RECV	WRITE	READ/ATOMIC
RC	✓	✓	✓
UC	✓	✓	✗
UD	✓	✗	✗

3 Execution Framework

To provide an apple-to-apple comparison on different primitives and transactions, we implement an execution framework for RDMA, which contains both one-sided and two-sided RDMA primitives, various prior optimizations and our newly proposed optimization.

3.1 Primitives

Symmetric model. We use a *symmetric model* in our experiments as prior work [51, 5, 11, 18]. In a symmetric model, each machine acts both a client and a server. On each machine, we register the memory with huge pages for RNIC to reduce RNIC's page translation cache misses [10].[4]

QP creation. We use a dedicated context to create QPs for each thread; otherwise, there will be false synchronizations within the driver even each thread uses its own QP. The performance impact is shown in Fig. 3(a)[5]. The root cause is that each QP uses a pre-mapped buffer to send MMIOs to post requests while the buffer may be shared. The buffer is allocated from a context according to Mellanox's driver implementation, where each context

[4]Currently, we use kernel's native huge page support (i.e., 2MB), which is sufficient for our current workloads.

[5]The details of experimental setup can be found in §4.1.

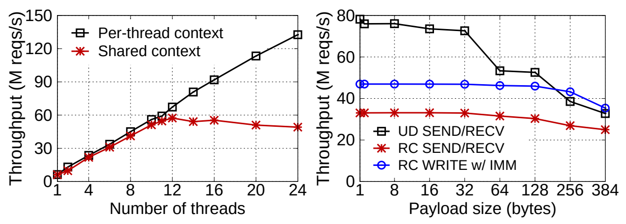

Fig. 3: *(a) The performance of RDMA `WRITE` using different QP creation strategies. (b) A comparison of different RDMA-enabled RPC implementations*

has limited buffers. For example, the mlx4 driver [41] uses 7 dedicated buffers and 1 shared buffer. This means that if the context is used to create more than 8 QPs, then extra QPs have to share the same buffer. Even if each thread uses one exclusive QP, the throughput of a shared context drops by up to 63% with the increase of threads. The overhead comes from synchronizations on the shared MMIO buffer.

One-sided primitive. Each thread manages *n* RC QPs to connect to *n* machines. We use standard Verbs API to post a one-sided request to the QP corresponding to the machine. RDMA `WRITE` requests with payloads less than 64 bytes are inlined to improve throughput [16]. Note that we do not simply wait until the completion of the operation (§3.2): we execute other application requests or RPC functions for better utilizing CPU and network bandwidth.

Two-sided primitive. Unlike one-sided primitive which has a simple and straightforward implementation, there are many proposed RPC implementations (two-sided primitive) atop of RDMA [10, 16, 18, 26, 46, 39]. They can be categorized into `SEND/RECV` verb based [18], RDMA `WRITE` based [10, 39, 46, 26] and hybrid one [16].

We use `SEND/RECV` verbs over UD QP as our two-sided implementation in this paper for three reasons. First, in a symmetric setting, `SEND/RECV` verbs over UD has better performance than other implementations over RDMA, especially for transaction systems [18]. This is also confirmed in our experiment (see Fig. 3(b)). Second, based on our studies of one-sided RDMA performance, *one-sided RDMA based RPC is unlikely to outperform UD based RPC* especially for small messages. The peak throughput of one-sided `WRITE` reaches 130M reqs/s when the payload size is smaller than or equal to 64 bytes (Fig. 5). For an RPC communication, two RDMA `WRITE`s are required (one for send and one for reply). Thus, the peak throughput of RPC implemented by one-sided RDMA operations is about 65M reqs/s, lower than that of the implementation based on `SEND/RECV` over UD (79M reqs/s).

Discussions. `SEND/RECV` over UD does not provide a reliable connection channel. Therefore, it may be unfair to compare it to RC based two-sided implementations which have reliability guarantees. However, since RDMA network assumes a lossless link layer, UD has much higher reliability than expected [18]. Further, packet losses can be handled by transaction's protocol [18].

3.2 Optimizations Review and Passive ACK

Many optimizations have been proposed in prior work to better leverage RDMA [10, 16, 17]. We first briefly review them here and show that when using RDMA properly, *one-sided primitive yields better performance than two-sided primitive with the same round trips.* We further propose a new optimization, *Passive ACK*, which improves RDMA primitives when the completion acknowledgement (ACK) of the request is not on the critical path of the application.

Coroutine (CO). Even the latency of RDMA operations reaches several microseconds, it is still higher than the execution time of many applications [18]. Thus, it is worth to use coroutines to further hide the network latency by sending multiple requests from different transactions in a pipelined fashion. FaSST [18] uses coroutine to improve the throughput of its RPC. FaRM [10, 11] optimizes both one-sided operations and RPCs using an event loop to schedule transactions with RDMA operations. We use a set of coroutines to execute application logic at each thread. Each coroutine yields after issuing some network requests (including both one-sided and two-sided ones), and they resume the execution until they receive the completions of one-sided requests (or the replies of two-sided RPCs). Typically, a small number of coroutines is sufficient for RDMA latency hiding (e.g., 8) [18].

Outstanding requests (OR). Even coroutine overlaps computation with I/O from different transactions, it is still important to send requests from one transaction in parallel. This further increases the utilization of RNICs and reduces the end-to-end latency of transactions, i.e., there is no need to wait for the completion of one request before issuing another one. For example, the read/write set of many OLTP transactions can be known in advance [37]. Therefore, it is possible to issue these reads and writes in parallel.

Doorbell batching (DB). There are several ways to issue multiple outstanding requests to RNIC. A common approach is to post several MMIOs corresponding to different requests. On the other hand, doorbell batching rings a doorbell to notify RNIC to fetch multiple requests by itself using DMA [17]. MMIO is costly which usually requires hundreds of cycles. Therefore, doorbell batching can reduce CPU overhead on the sender side and make

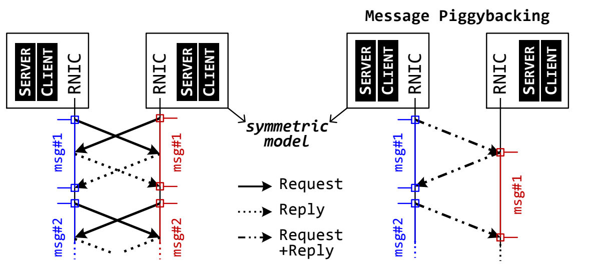

Fig. 4: *A sample of passive ACK for two-sided primitive.*

a better usage of PCIe bandwidth, since it only requires one MMIO per batch to ring the doorbell.

One restriction of doorbell batching is that only requests from one QP can be fetched by the RNIC in a batched way. This means that different one-sided requests cannot be batched together if they are not sent to the same machine. Due to this limitation, doorbell batching is usually applied to two-sided implementation based on UD QP [18].

Passive ACK (PA). The performance can be further improved if the completion of requests (ACK) is done off the critical path of transactional execution. We achieve this by acknowledging the request passively.

For one-sided primitive, the request is marked as unsignaled, and then the completion of the request is confirmed passively after a successful polling of one subsequent signaled request. This avoids consuming RNIC's bandwidth.[6] For two-sided primitive, the optimization has the potential to double the throughput in a symmetric model by piggybacking the reply messages with the request messages. As shown in Fig. 4, passive ACK can save half of the messages (replies).

It should be noted that not all of the completions can be acknowledged passively. For example, one-sided `READ` requires a completion event; otherwise, the application does not know whether the read is successful or not. Fortunately, in transactional execution, a transaction is considered to be committed when the log has been successfully written to all backups (see Fig. 1). Hence the write-back request at the commit phase can be acknowledged passively.

4 A Primitive-level Performance Analysis

In this section, we first present our execution framework with both one-sided and two-sided primitive of RDMA. We then present the basic performance of different RDMA primitives, including raw RDMA performance and the performance of micro-benchmarks. The micro-benchmarks simulate common transactional workloads. These experimental results serve as the guideline for using the appropriate primitives for transactions.

[6]Verbs from the same send queue are processed in a FIFO manner [2].

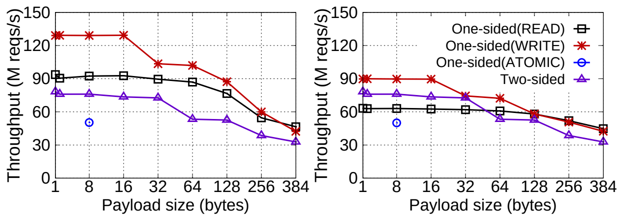

Fig. 5: *A comparison of one-sided and two-sided primitives using (a) a 16-node cluster and (b) an emulated 80-node connection setting. RDMA* `ATOMIC` *only supports the 8-byte payload.*

4.1 Setup

Testbed. Unless otherwise specified, we use a local rack-scale RDMA-capable cluster with 16 machines for all experiments. Each machine is equipped with two 12-core Intel Xeon E5-2650 v4 processors, 128GB of RAM, and two ConnectX-4 MCX455A 100Gbps Infiniband NIC via PCIe 3.0 x16 connected to a Mellanox SB7890 100Gbps InfiniBand Switch.

Execution. We run 24 worker threads (same as the number of available cores per machine) on each machine in our experiments. Each worker thread runs an event loop to execute transactions, handles RPC requests, and polls RDMA events. The events of RDMA including the completion of one-sided RDMA requests and the reception of RPC requests/replies. We follow FaSST [18] by using coroutine from Boost C++ library to manage context switches between clients when issuing network requests. Boost coroutine is efficient in our experiments, which has very low overhead for context switch (about 20 ns).

4.2 Primitive-level Performance Analysis

RDMA raw performance. Prior work has shown that two-sided primitives have better performance and scalability than one-sided ones [18]. This conclusion is drawn from an old generation of RNIC (ConnectX-3). Further, they only show the poor scalability of one-sided primitive using small payloads (less than 32 bytes). We extend their evaluation [18] on raw RDMA performance to show that: one-sided primitives have better performance than two-sided ones using 16 nodes, as shown in Fig. 5(a). More importantly, the scale of the cluster only affects *one-sided primitives with small payloads*. For example, with our emulated 80-node connection setting, one-sided primitives still outperform two-sided ones when data payloads are larger than 64 bytes.

Emulating massive RDMA connections. On our 16-node cluster, we create 5 RC QPs to connect to each machine at each worker. The number of QPs (5x16 QPs per thread) is sufficient to run in an 80-node cluster. We choose the QPs randomly to post upon issuing a request. Note that the total number of QPs (960 per NIC) has exceeded the total number of QPs that can be cached at RNIC.

<u>*Primitive evaluation*</u>. Fig. 5(a) presents the evaluation results of the primitive analysis. For read operations, one-sided primitives (`READ`) outperform two-sided ones by up to 1.6X when payload size is below 64 bytes, and by up to 1.37X for larger payloads. For write operations, one-sided primitives (`WRITE`) outperform reads on small payloads but get a similar trend on large payloads (from 1.03X to 1.35X). Note that we do not incur memory copy overhead for two-sided primitives, as done in prior work [18], since adding such overhead will affect the performance of two-sided ones, especially for large messages.

Fig. 5(b) further presents the results on an emulated 80-node connection setting. The performance of one-sided `READ` becomes slow-growing with the decrease of payloads from 128 bytes. This is because RNIC experiences QP cache misses at this time.[7] However, one-sided `READ`s can still outperform two-sided primitives when payloads are larger than 64 bytes. Because the cost of data transfer instead of QP cache misses dominates the performance for larger payloads.

A final takeaway is that, although one-sided `ATOMIC` is relatively slow [17], it can still achieve 48M reqs/s on each machine, which is much higher than the requirements of many workloads (e.g., `TPC-C`). Therefore, the performance will not be the main obstacle to leverage one-sided atomic primitives in transactional execution (e.g., distributed spinlock). We evaluate this approach in the transactional workload (§5.2).

Micro-benchmarks. Better performance in raw throughput does not always mean better performance in real applications. We use two micro-benchmarks to compare how different primitive performs under common transactional workloads, and how previous optimizations affect the performance (Fig. 6).

<u>*Workloads*</u>. We use a workload with multi-object reads and writes to compare the performance of different RDMA primitives. This workload simulates common operations in transactional workloads: at the execution phase, transaction reads multiple records from remote servers; at the commit phase, transaction writes multiple updates back to remote servers. Note that the workloads use fixed-length 64-byte payloads, and issue 10 operations.

<u>*Effects of optimizations*</u>. We first show how existing optimizations improve the performance of each primitive from Fig. 6. Coroutine, outstanding requests and doorbell can be applied to both workloads.

Coroutine hides the latency and improves the performance of one-sided and two-sided by 7X and 6.46X, respectively. Adding outstanding requests by posting more requests per batch further improve the throughput due to better uses of RNIC's processing capability.

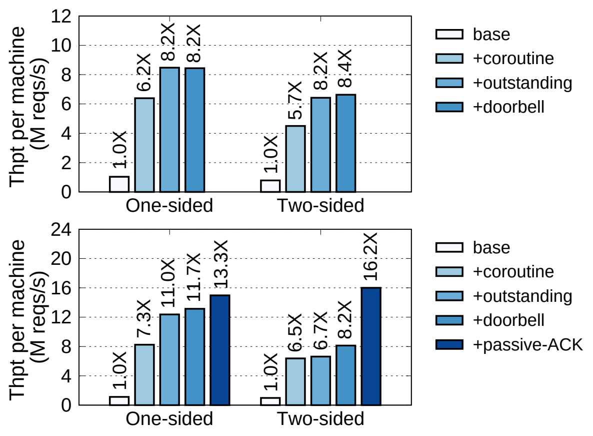

Fig. 6: *A comparison of one-sided and two-sided primitives for multiple-object (a) reads and (b) writes with 64-byte payloads.*

Doorbell batching does not always improve the performance of one-sided primitive, but it constantly improves the throughput of two-sided ones. This is because doorbell batching can only apply to a single QP, which is suitable for UD-based two-sided implementation. On the contrary, one-sided requests are sent through multiple RC QPs, which reduces the chances of using doorbell batching. Further using doorbell batching requires bookkeeping the status of posted requests, which adds additional overhead.

<u>*Offloading when completion is required*</u>. By enabling passive acknowledgement (PA), the performance of one-sided `WRITE` is further improved by 1.13X, while that of two-sided primitive is nearly doubled (1.96X) due to the reduction of half of the messages (for reply). This makes the only case where two-sided outperforms one-sided. Otherwise, one-sided primitive always has better performance than two-sided ones. This is consistent with the results in Fig. 5. For example, multiple `READ`s can achieve peak throughput about 8.43M, which is close to the raw performance of one-sided `READ` (about 86.9M per machine).

5 A Phase-by-phase Performance Analysis

In this section, we study the performance of transactions phase-by-phase with different RDMA primitives. Table 2 summaries whether we apply the optimization discussed in §3.2 at different phases of transactional execution. Below is some highlights of our phase-by-phase analysis:

- One-sided primitive is faster when the number of round trips is the same and the completion acknowledgement of requests are required (§5.1,5.2,5.4,5.5).
- It is always worth checking and filling the lookup

[7] We use PCIe counters to measure QP cache misses, similar to pmu-tools (`https://github.com/andikleen/pmu-tools`).

cache for one-sided primitive, even using two-sided primitive (§5.1).

- One-sided primitive is faster, even using more network round trips, for CPU-intensive workloads (§5.1).
- Two-sided primitive with passive ACK has comparable or better performance than one-sided (§5.3).

Benchmarks. We use two popular OLTP benchmarks, `TPC-C` [44] and `SmallBank` [42], to measure the performance[8] of every phase with different primitives, since they represent *CPU-intensive* and *network-intensive* workloads respectively. We use a partitioned data store where data is sharded by rows and then distributed to all machines. We enable 3-way logging and replication to achieve high availability, namely each primary partition has two backup replicas.

TPC-C simulates an order processing application. We scale the database by deploying 384 warehouses to 16 machines. We use this benchmark as a CPU-intensive workload. `TPC-C` is known for good locality: only around 10% of transactions access remote records. To avoid the impact of local transactions, which our work does not focus on, we only run `new-order` transaction of `TPC-C` and make transactions always distributed, which is a major type of transaction (45%) and representative in `TPC-C`.[9]

SmallBank simulates a simple banking application. Each transaction performs simple reads and writes operations on account data, such as transferring money between different users. We use this benchmark as a network-intensive workload because transaction only contains simple arithmetic operations on few records. We do not assume locality as previous work [18], which means that all transactions use network operations to execute and commit transactions. To scale the benchmark, we deploy 100,000 accounts per thread, while 4% of records are accessed by 90% of transactions.

5.1 Execution (E)

Overview. The transaction coordinator fetches the records a transaction reads in the *execution* phase. This requires traversing the index structure and fetching the record. We can simply send an RPC to remote server to fetch the record, which only requires one round-trip communication. On the other hand, we can also leverage one-sided RDMA `READ`s to traverse the data structure and read the record. This typically requires multiple round trips but saves remote CPUs. Prior work has proposed two types of optimizations to reduce the number of round trips required by one-sided primitives [29, 10, 51, 30].

[8]We scale up the concurrent requests handled by the server to achieve the peak throughput.

[9]For brevity, we refer to our simplified `TPC-C` benchmark as `TPC-C/no`.

Table 2: *A summary of optimizations on RDMA primitives at different phases (§3.2). **`OR`**, **`DB`**, **`CO`** and **`PA`** stand for outstanding request, doorbell batching, coroutine, and passive ACK. **`RW`** and **`RO`** stand for read-write and read-only transactions. `I` and `II` stand for one-sided and two-sided primitives.*

		OR		DB		CO		PA	
		I	II	I	II	I	II	I	II
RW	E	✗	✗	✗	✓	✓	✓	✗	✗
	V	✓	✓	✓	✓	✓	✓	✗	✗
	L	✓	✓	✗	✓	✓	✓	✗	✗
	C	✓	✓	✓	✓	✓	✓	✓	✓
RO	R	✗	✗	✗	✓	✓	✓	✗	✗
	V	✓	✓	✗	✓	✓	✓	✗	✗

RDMA-friendly key-value store. Many hash-based data structures can be optimized to reduce the number of RDMA operations for traversing the remote server to find the given key, these include cuckoo hashing [29], hopscotch hashing [10], and cluster hashing [51]. We adopt DrTM-KV [51], a state-of-the-art RDMA-friendly key-value store in all experiments.

RDMA-friendly index cache. The ideal case for one-sided primitive is to use one one-sided `READ` to get the record back. DrTM [51] introduces a location-based cache to eliminate the lookup cost (one RDMA `READ`) in the common case. FaRM [11] and Cell [30] use a similar design for caching the internal nodes of B-tree. In our experiment, we maintain a 300MB index cache on each machine, which will be used and filled in the execution phase. Note that the index cache is quite effective since a relatively small cache is usually enough for skewed OLTP workloads [15, 8, 3, 33, 20], such as `SmallBank` [42], `TATP` [32], and `YCSB` [6].

Evaluation. Fig. 7 compares the performance of using one-sided and two-sided primitives for the execution phase on `TPC-C/no` and `SmallBank`, respectively. Two-sided uses one RPC to fetch the record. One-sided fetches records with at least two one-sided `READ`s (one for index and one for payload). One-sided/Cache always fetches the indexes from the local index cache and then get the record from a remote server using a single one-sided `READ`. This presents an ideal case for the performance of the execution phase using one-sided primitives.

TPC-C/no: One-sided/Cache outperforms Two-sided by up to 1.45X in throughput (from 1.26X), and the median latency is only around 69% of Two-sided (from 89%). The benefits mainly come from the better performance of one-sided `READ`s. Two-sided outperforms One-sided (no cache) by up to 1.28X in throughput. Without caching, the coordinator requires an average of double round trips (one for lookup and another for read) to fetch one record.

Interestingly, when increasing the number of corou-

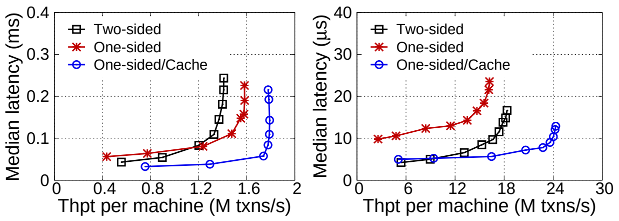

Fig. 7: *The performance of (a)* `TPC-C/no` *and (b)* `SmallBank` *with different implementations of* E*xecution phase.*

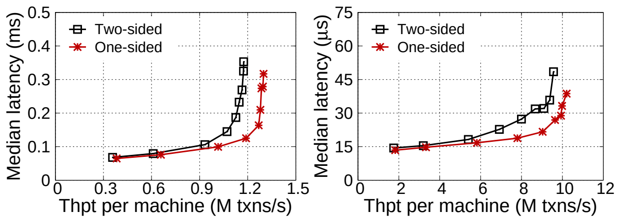

Fig. 8: *The performance of (a)* `TPC-C/no` *and (b)* `SmallBank` *with different implementations of locking in* V*alidation phase.*

tines, the peak throughput of One-sided (no cache) outperforms that of Two-sided (about 13%). The median latency is also slightly better when using more than 10 coroutines. The performance gain comes from lower CPU utilization on each machine. This confirms the benefits of using one-sided primitives when remote servers are busy [29, 30]. The adaptive caching scheme in prior work [29, 30] can be used to get better performance by balancing CPU and network.

SmallBank: Not surprisingly, One-sided/Cache still outperforms Two-sided by up to 1.36X in throughput due to the better performance and CPU utilization of one-sided `READ`. However, compared to One-sided (no cache), the speedup of peak throughput for Two-sided reaches up to 2.01X (from 1.13X). This is due to two reasons. First, without location-based cache, one-sided uses more round trips to finish the execution phase. Further, the performance of `Smallbank` is bottlenecked by network bandwidth since it is a network-intensive workload.

Summary. If one round-trip RDMA `READ` can retrieve one record using the index cache, one-sided primitive is always a better choice than two-sided one. Otherwise, two-sided primitive should be used when servers are not overloaded. Hence, a *hybrid* scheme should be used in the execution phase. Specifically, we should *always enable the index cache and look from it* before choosing either one-sided primitive (on cache hit) or two-sided primitive (on cache miss). We should also *always refill the cache even if two-sided primitive is chosen upon a miss.*

5.2 Validation (V)

Overview. To ensure serializability, OCC atomically checks the read/write sets of the transaction in the validation phase. The coordinator first *locks* all records in the transaction's write set and then *validates* all records in the read/write set to ensure that they have not been changed after the execution phase.

Lock. RDMA provides one-sided *atomic compare and swap* operations (`ATOMIC`), which can be used to implement distributed spinlock [51, 5]. Although `ATOMIC` is slower than other two-sided primitives, on recent generation of RNIC (e.g., ConnectX-4), `ATOMIC` can achieve 48M reqs/s, which is enough for many OLTP workloads (e.g., `TPC-C`). More importantly, the throughput of two-sided primitive (76M) was evaluated with an empty RPC workload. When locking the record in the RPC routine, the impact of CPU efficiency may change the relative performance of one-sided and two-side primitives. This is especially the case for the symmetric model adopted by transaction systems [51, 5, 11, 18, 56], when the servers are busy processing transactions.

Validate. Different from the execution phase, a single RDMA `READ` is enough to retrieve the current version of the record for validation, thanks to caching the index in the execution phase of the transaction. Therefore, one-sided primitive is always a better choice for read-only records compared to two-sided one, according to the results in Fig. 5 and Fig. 7.

Optimization. OCC demands the validation should start exactly after locking all records [47, 11]. This takes two round trips for every read-write record in the validation phase. Fortunately, the locked record can be validated immediately since it can not be changed again. Therefore, each read-write record can be handled by both one-sided and two-sided primitives in one round trip. For one-sided, the RDMA `READ` request will be posted just after the RDMA `CAS` request in a doorbelled way to the same send queue of target QP, since they are processed in a FIFO manner. Further, with passive ACK, the `CAS` request can be made unsignaled (§3.2). For two-sided, the RPC routine will first lock the record and then read its version. On commodity x86 processors, compiler fences are sufficient to ensure the required ordering.

Restriction of RDMA atomicity. Currently, the key challenge for using one-sided primitive (RDMA `ATOMIC`) for distributed locking is that `ATOMIC` cannot correctly work with CPU's atomic operations (e.g., CAS). To remedy this, local atomic operations must also use RNIC's atomic operations [51], which will slow down the validation phase of local transactions. Leveraging advanced hardware features, like hardware transactional memory (HTM), can overcome this issue [51].

Evaluation. Fig. 8 compares the performance of using one-sided and two-sided primitives for the validation phase on `TPC-C/no` and `SmallBank`, respectively.

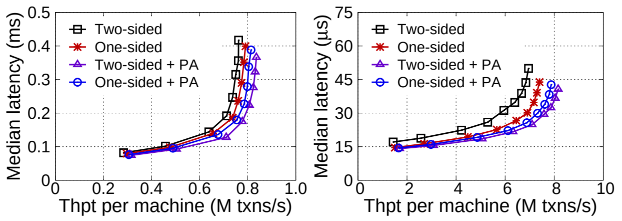

Fig. 9: *The performance of (a)* `TPC-C/no` *and (b)* `SmallBank` *with different implementations of Commit phase.*

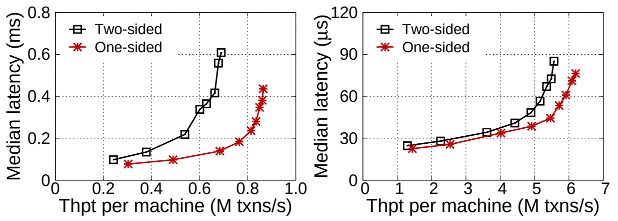

Fig. 10: *The performance of (a)* `TPC-C/no` *and (b)* `SmallBank` *with different implementations of Logging phase.*

Since the read/write sets are the same in `TPC-C/no` and `SmallBank`, one-sided will send one `ATOMIC` and one `READ` sequentially to lock the record and retrieve the current version in one round trip. We can see in Fig 8 that for both workloads, one-sided primitive (`ATOMIC`) is faster, even it has lower peak throughput.

Summary. Although RDMA `ATOMIC` is slower than other RDMA network primitives, it may not be the bottleneck for many applications and can further improve the performance of many workloads. If the atomicity between RNIC and CPU will not cause a performance issue, One-sided RDMA `ATOMIC` is a better choice to implement distributed locking due to high CPU efficiency. Otherwise, two-sided primitive is preferred in this phase since local CASs are much faster than RNIC's CASs.

5.3 Commit (C)

Overview. In the commit phase, the coordinator first writes the updates of the transaction back and then releases the locks. One-sided `WRITE` can be used to implement the commit operation with two requests, one to write updates back and one to release the locks (i.e., zeroing the lock state of the record).

Similar to the validation phase, two one-sided `WRITE`s (one to write the update back and one to release the lock) will be posted sequentially to the same QP in a doorbelled way, which preserves the required ordering (release after write-back). Therefore, the commit phase can be handled by both one-sided and two-sided primitives in one round trip.

Optimization with passive ACK. Since the transaction is considered to be committed after the completion of logging, the completion of the commit message can be acknowledged passively by piggybacking with other messages. Thus we enable passive ACK optimization to both one-sided and two-sided primitives in the commit phase.

Evaluation. Fig. 9 presents the performance of `TPC-C/no` and `SmallBank` using different commit approaches. Note that we use two-sided as the validation implementation in this experiment. This is because one-sided ATOMICs cannot work correctly with the commit phase with two-sided primitive due to the atomicity issue with our current RNIC.

For both workloads, without passive ACK, one-sided WRITEs are faster due to better CPU utilization at the receiver's side. With passive ACK, two-sided is faster. This is because, although two-sided primitive costs more CPU at the receiver side, it can save CPU at sender side due to doorbell batching [17] (see Table 2). One-sided primitive requires multiple MMIOs to commit multiple records, while two-sided primitive can chain these requests by using one doorbell. Passive ACK can further save the cost of two-sided primitives when sending the replies back. These results match up with the results observed in our primitive-level performance analysis (§4).

Summary. To commit transactions, two-sided primitive with passive ACK is the better choice.

5.4 Logging (L)

Overview. In the logging phase, the coordinator writes transaction logs with all updates to all backups. After receiving the completion acknowledgements from all backups, the transaction commits. The coordinator will notify backups to reclaim the space of logs lazily by updating records in-place.

One-sided primitive. To enable logging with one-sided RDMA `WRITE`, each machine maintains a set of ring-buffers for remote servers to log. The integrity of the log is enforced by setting the payload size at the begin and end of the message, similar to previous work [10]. Note that since we use RC (Reliable Connection) QP to post one-sided `WRITE`s, the logging is considered success after polling the ACK from the RNIC. We use two-sided primitive to reclaim the log since it must involve remote CPUs [11]. Since log reclaiming is not on the critical path of transactional execution, this request can be marked as unsignaled and the claiming can be done in the background.

Two-sided primitive. Logging with two-sided primitive is relatively simple. The RPC routine copies the log content to a local buffer after receiving the log request, and then it sends a reply to the sender. The log reclaiming can also be executed in the background.

Evaluation. Fig. 10 presents the performance of `TPC-C/no` and `SmallBank` using different logging approaches. For both of them, one-sided logging always

Table 3: *A summary of execution time (cycles) and payload size (bytes) in different phases for* `TPC-C` *and* `SmallBank`*.*

	TPC-C		SmallBank	
	Time	Payload	Time	Payload
Execution	342	68	678	71
Validation	454	157	185	105
Logging	363	1006	134	149
Commit	108	34	87	20

has higher throughput and lower latency than its two-sided counterpart, thanks to offloading write operations to one-sided primitives. Using one-sided logging increases the throughput of `TPC-C/no` and `SmallBank` by up to 1.29X (from 1.24X) and 1.12X (from 1.10X), respectively. One-sided logging has more improvements in peak throughput in `TPC-C/no` since the payload size of logs in `TPC-C` is much larger than that of `SmallBank` (1,006B vs. 149B), as shown in Table 3.

Summary. Since the logging phase can be offloaded using one-sided RDMA `WRITE`s with one round trip, one-sided primitive is always preferred to write logs.

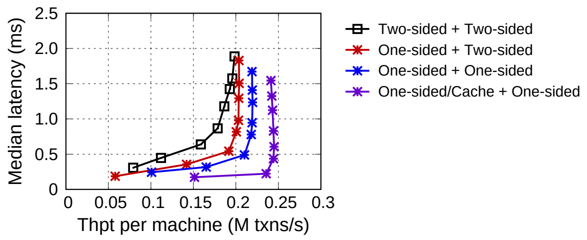

Fig. 11: *The performance of* `customer-position` *in* `TPC-E` *with different implementations of the read-only transaction (Read and Validation phases).*

5.5 Read-only Transaction (R+V)

Overview. We use a simplified two-phase protocol to run read-only transactions as prior work [25]. The first phase reads all records like the execution phase, and the second phase validates that the versions of all records have not been changed, which is similar to the operations in the validation phase for the records in read set. For single-key read-only transactions, the validation phase can be ignored. These transactions are popular in many OLTP workloads (e.g., `TATP` [32]), as reported by prior work [11, 18].

Evaluation. With a proper sharding, there is no distributed read-only transaction in `TPC-C`, which needs remote data accesses. Further, there is only one single-key read-only transaction in `Smallbank` (i.e., `Balance`), which does not require the second phase (validation) [11, 18]. Therefore, we use the `customer-position` transaction in `TPC-E` [43] to evaluate the performance of distributed read-only transactions.

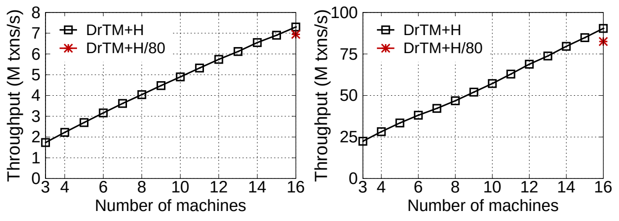

Fig. 12: *The performance of* `DrTM+H` *with the increase of machines for (a)* `TPC-C/no` *and (b)* `SmallBank`*.*

TPC-E. is designed to be a more realistic OLTP benchmark, which simulates the workload of a brokerage firm. One of well-known characteristics is the high proportion of read-only transactions, reaching more than 79%. The `customer-position` transaction is read-only and has the highest execution ratio. It simulates the process of retrieving the customer's profile and summarizing their overall standing based on current market values for all assets. The assets prices are fetched in a distributed way.

Fig. 11 compares different choices of primitives for distributed read-only transactions. As expected, by offloading read operations to RNICs and bypassing remote CPUs, using one-sided primitives for both the read and validation phases can gain the best performance in both throughput and latency. One-sided outperforms Two-sided by about 10% in peak throughput (0.19 vs. 0.21), and the median latency is around 80% of Two-sided. Enabling the index cache (One-sided/Cache) in the read phase will further improve the peak throughput by close to 20% (0.25 vs. 0.21) and reduce the median latency more than 20%.

Summary. The hybrid scheme used in the execution phase (see §5.1) is also suitable to the first phase, and one-sided `READ` is always a better choice for the second phase (see §5.2). For single-key read-only transactions, a single one-sided `READ` is usually efficient.

6 Fast Transactions using Hybrid Schemes

In this section, we conclude our studies of using RDMA primitives for transactions by showing how to improve the performance of prior designs by choosing appropriate primitives and techniques at different phases of transactional execution. This leads to `DrTM+H`, an efficient distributed transaction system using hybrid schemes.

6.1 Design of `DrTM+H`

`DrTM+H` optimizes different phases of the transaction by choosing the right primitives guided by our previous studies (§4 and §5). `DrTM+H` supports serializable transaction with log replication for high availability. Currently, we have not implemented the reconfiguration and recovery, which is necessary to achieve high availability. Yet, since our replication protocol is exactly the same as the one

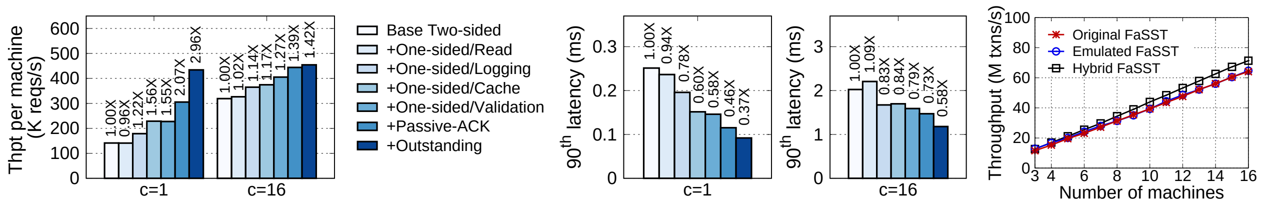

Fig. 13: *The contribution of optimizations to (a) throughput and (b,c) latency to* `TPC-C/no` *for* `DrTM+H` *using 1 and 16 coroutines, respectively. Optimizations are cumulative from left to right. (d) A performance comparison between original and emulated FaSST.*

used in FaRM [11], `DrTM+H` can use its method to recover from failure.

Execution. `DrTM+H` uses a hybrid design of one-sided `READ`s with caching and two-sided RPC. If the record's address has been cached locally, one RDMA `READ` is sufficient to fetch the record. Otherwise, `DrTM+H` uses RPC to fetch the record and its address.

Validation. `DrTM+H` uses one-sided `ATOMIC` for validation if there is no atomic issue (e.g., Network accesses do not conflict with local ones). Otherwise two-sided is preferred since using RDMA atomic operations will slow down local operations [51].

Logging. `DrTM+H` always uses one-sided `WRITE`s to replicate transaction logs to all backups and uses two-sided primitive to lazily reclaim logs on backups.

Commit. `DrTM+H` uses one-sided `WRITE`s to commit if one-sided `ATOMIC` is used in the validation phase. Otherwise `DrTM+H` uses two-sided RPC. `DrTM+H` always uses passive ACK optimization since the completion of commit message is not on the critical path of transactional execution.

Using outstanding request with speculative execution. In §5.1, we disable the outstanding request optimization at the execution phase to avoid requiring advance knowledge of read/write set. However, this usually means that transaction must fetch records one-by-one, which increases the latency of a single transaction.[10] We found that even the record has not been fetched to local, the transaction can still speculatively execute until the involved value is really used. This can greatly reduce the lifespan of a transaction. For example, the remote records required by `new-order` transaction in `TPC-C` are independent. Thus `DrTM+H` uses speculative execution to fetch these records in parallel.

6.2 Performance Evaluation

Fig. 12 presents the throughput and scalability of `DrTM+H` using `TPC-C/no` and `SmallBank`. To show that `DrTM+H`'s usage of one-sided primitive has good scalability on a larger-scale cluster, we use the QP setting which is enough to run on an 80-node cluster (`DrTM+H`-80). Each

[10]We still send multiple requests in parallel for different transactions using coroutines.

Table 4: *A review of the existing RDMA-enabled transaction systems.* `I` *and* `II` *stand for one-sided and two-sided primitives.*

	RW-TX				RO-TX	
	E	V	L	C	R	V
FaRM	I	II+I	I	II	I	I
DrTM+R	I	I+I	I	I+I	I	I
FaSST	II	II	II	II	II	II
DrTM+H	I/II	I/II	I	I/II	I/II	I

thread uses 80 QPs (16x5) to connect to 16 nodes and chooses the usage of QP in a round-robin way.

Performance and scalability. `DrTM+H` scales linearly with the increasing of machines. The throughput of `TPC-C/no` and `SmallBank` decrease 5% and 9% on the emulated 80-node connection setting, respectively. `SmallBank` is more sensitive to the number of QPs since its payload size is much smaller than that of `TPC-C/no`. However, `SmallBank` is still 1.3X higher than a pure two-sided solution in throughput, with a significant decrease in the tail latency. The 50^{th} (median), 90^{th}, and 99^{th} latency are reduced by 22%, 39%, and 49%, respectively.

Factor analysis. To investigate the contribution of the primitive choices in `DrTM+H`, we conduct a factor analysis in Fig. 13. Due to space limits, we only report the experimental results of `TPC-C/no`; `SmallBank` is similar. First, we observe that using one-sided primitives can significantly improve the throughput and latency when servers are underloaded (1 coroutine). This is because one-sided primitive has lower CPU utilization and lower latency compared to two-sided one. Second, by increasing coroutines, the two-sided implementation has close throughput with one-sided one. However, a hybrid scheme in `DrTM+H` improves both median and tail latency. Finally, when leveraging RDMA, the number of round trips has more impacts on latency but not throughput, especially for CPU-intensive workloads (e.g., `TPC-C`). When using 16 coroutines, the throughput increases even using more network round trips (adding one-sided `READ`s). This is because coroutines hide most of waiting for request's completion while one-sided primitive has lower CPU utilization.

6.3 Comparison Against Prior Designs

There have been several designs to optimize transactional execution using RDMA. To understand the effects of RDMA primitive decisions, we implemented and evaluated emulated versions of FaRM [11], DrTM+R [5] and FaSST [18].[11] We adopted the same codebase and transaction protocol (OCC) of `DrTM+H`, but choosing the RDMA primitives and techniques at different phases of transactional execution as the originals. Table 4 summarizes the primitives used in the three systems and compares the performance of emulated versions of them with `DrTM+H`. Note that all existing optimizations on RDMA primitives are enabled, including coroutine, outstanding requests, and doorbell batching.

Emulating FaRM. FaRM [11] is designed to run transactions atop of a global memory space over RDMA networking. FaRM uses one-sided `READ` at the execution/logging phase and one-sided `WRITE` at the logging phase, as well as a hybrid choice at the validation phase. Moreover, FaRM adopts an RDMA-friendly memory store (FaRM-KV) proposed in their prior work [10]. Our emulated store (DrTM-KV) has been shown to have a comparable performance even without the location cache [51]. Further, our two-sided RPC implementation has also better performance than the implementation in FaRM [18] (see `RC WRITE` w/ `IMM` in Fig 3(b)). Hence, we believe our emulated version has similar or even better performance compared to the vanilla FaRM.

Emulating DrTM+R. DrTM+R [5] offloads all network operations to one-sided RDMA primitives for CPU efficiency, including using one-sided `ATOMIC` for locking remote records in the validation phase. Further, DrTM+R exploits hardware transactional memory (HTM) [14] to handle local transactions, but does not leverage coroutines to obtain higher throughput. To focus on comparing different choices of RDMA primitives, our emulated version disables HTM (similar to the implementation of DrTM-OCC [4]) but enables coroutine optimization.

Emulating FaSST. FaSST [18] proposes a well-optimized RPC implementation (see `UD SEND/RECV` in Fig 3(b)) based on two-sided primitives for running transactions. Since our framework provides a similar UD-based RPC implementation, it is straightforward to emulate FaSST by using two-sided primitives at all phases of transactional execution. Since FaSST uses a simplified OCC protocol [18] by moving lock operations from the validation phase to the execution phase, we use FaSST-OCC to name the pure two-sided implementation on our platform with OCC protocol, to avoid confusion.

[11]FaRM is not open-sourced, `DrTM+R` depends on hardware transactional memory, and FaSST uses a simplified OCC and protocol.

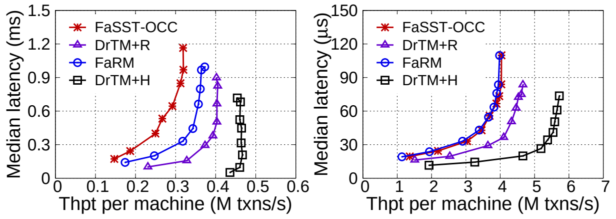

Fig. 14: *An end-to-end comparison of different designs for (a)* `TPC-C/no` *and (b)* `SmallBank`*.*

Calibrating performance with FaSST. Since the source code of FaSST is available and does not depend on specialized hardwares, we compare the performance of the original version and our emulated version using the `SmallBank` benchmark with 24 threads and 1 NIC per machine.[12] Note that we also implement their specific OCC protocol which can reduce one network round trip with advance knowledge of transaction's read/write set.As shown in Fig. 13(d), emulating FaSST on our platform can achieve comparable performance. Further, under our guideline for using the appropriate primitives, a hybrid design can also improve the implementation of FaSST's protocol by using one-sided primitives in the validation phase and the logging phase. As shown in Fig. 13(d), the hybrid choice (Hybrid FaSST) outperforms the original FaSST by up to 11% in throughput.

Evaluation. Compared to other prior designs, `DrTM+H` always embraces the best performance in terms of latency and throughput. Fig. 14 presents our results. `DrTM+H` has the best throughput than previous designs with the right choice of RDMA primitives and a set of optimizations to better leverage the chosen primitive. On `TPC-C/no`, `DrTM+H`'s throughput is up to 2.96X of FaSST (from 1.41X), up to 1.89X of DrTM+R (from 1.12X) and up to 2.50X of FaRM (from 1.21X). When using 16 coroutines, the median latency is reduced by 33%, 23% and 34%, respectively. We broke down the performance improvements in §6.2. FaRM optimizes baseline two-sided (FaSST) by using one-sided operation for logging and execution. DrTM+R further adds location cache and use one-sided for validation and commit. In `TPC-C/no`, FaRM and DrTM+R outperforms FaSST due to better leveraging one-sided primitives for CPU-intensive workloads. DrTM+R outperforms FaRM due to the usage of location cache at the execution phase and the usage of atomics at the validation phase. FaSST has a comparable performance to FaRM for `SmallBank` since two-sided primitive is faster at the execution phase.

Latency breakdown. To study the performance influence of choosing RDMA primitives, we further show the latency breakdown in each phase for different designs

[12]The original FaSST does not support the `TPC-C` benchmark, and we utilized the configuration as suggested by the authors of FaSST.

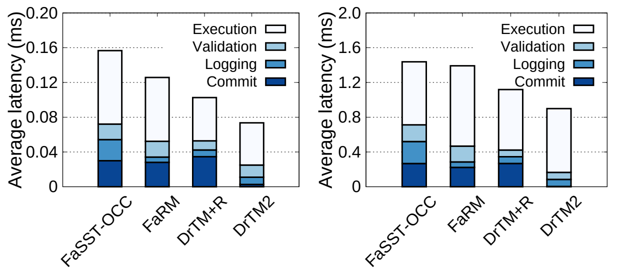

Fig. 15: *The latency breakdown of* `TPC-C/no` *using (a) 1 coroutine and (b) 16 coroutines.*

in Fig. 15. By leveraging one-sided `READ`s, the latency of the execution phase is reduced by 13% and 41% in FaRM and DrTM+R respectively, when using one coroutine. However, the increase of coroutines can narrow the performance gap by hiding the latency of network operations. FaSST can outperform FaRM by 22% when using 16 coroutines, since FaRM requires more network round trips to read remote data. To remedy this, DrTM+R enables the location-based cache [51] for one-sided operations and achieves the lowest latency (less than 0.7ms). In the validation phase, DrTM+R has the lower latency by offloading lock operations to RDMA NICs. Using one-sided `WRITE`s, the latency of the logging phase in DrTM+R and FaRM is reduced by about 69% and 75% respectively, compared to using two-sided primitives (FaSST). Finally, `DrTM+H` can always choose appropriate RDMA primitives to embrace the latency reduction at each phase. Note that `DrTM+H` has the lowest latency at the commit phase due to enabling Passive ACK optimization (§3.2), such that receiving the acknowledgement of commit messages is done off the critical path.

Table 5: *Different RDMA NICs used in the experiments. N denotes the number of nodes. C/N denotes the number of RNICs per node. P/C denotes the number of ports per RNIC. Each machine is the same as in §4.1.*

Name	N	C/N	P/C	RNIC
CX3	5	2	1	40Gbps ConnectX-3 InfiniBand
RoCE	2	2	1	100Gbps ConnectX-4 RoCE
CX5	2	1	1	100Gbps ConnectX-5 InfiniBand

7 Other RDMA NICs

The design choice of using specific RDMA primitive is guided by our primitive analysis in §4. The analysis itself depends on the performance of RNIC for different primitives. So our results depend on specific RNIC hardware characteristics.

In this section, we provide experiments using different RDMA platforms to show how our results applied to other settings. The experiment settings are summarized in Table 5. In summary, if RNIC can provide better performance for one-sided primitive using the same round trip, our results generally hold. We have observed faster one-sided primitive in recent generations of RNICs, like ConnectX-4 (CX4) and ConnectX-5 (CX5). On the other hand, if one-sided primitive cannot provide better performance, which is the case in old generations of RNIC, like ConnectX-3 (CX3), two-sided primitive shall be used.

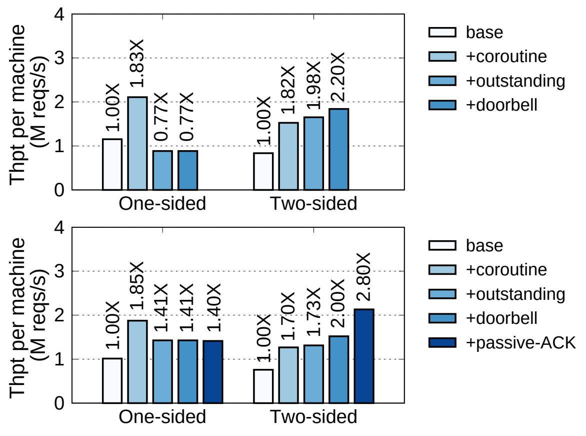

Fig. 16: *A comparison of one-sided and two-sided primitives using ConnectX-3 RNICs for multiple-object (a) reads and (b) writes with 256-byte payloads.*

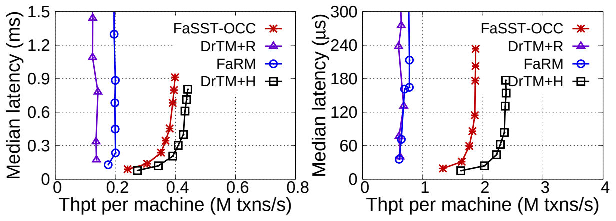

Fig. 17: *An end-to-end comparison of different designs for (a)* `TPC-C/no` *and (b)* `SmallBank` *using ConnectX-3 RNICs.*

ConnectX-3 (CX3). CX3 is an old generation of RNIC released in 2011. It is well-known for its poor one-sided performance [17, 18]. Using our micro-benchmarks, as shown in Fig. 16, one-sided primitive cannot provide better performance than two-sided primitive even using the same number of network round trips. Further, one-sided `ATOMICS` blocks RNIC operations [17] in CX3, resulting in relatively poor performance.

The poor performance of one-sided primitive in CX3 makes two-sided primitive a better design choice. This is because UD based two-sided primitive is less affected by hardware restrictions [16, 17]. Fig. 17 shows the end-to-end comparison of prior designs we reviewed in §6 using CX3. We can see that FaSST achieves the best performance due to better performance of two-sided primitives at each phase. FaRM uses slower one-sided primitive with more round-trip at the execution phase (for traversing the index). DrTM+R achieves the slowest performance because its performance is bottlenecked by the one-sided `ATOMICS`. Yet, `DrTM+H` can still choose the right primitive based on the evaluation results and achieve the best results.

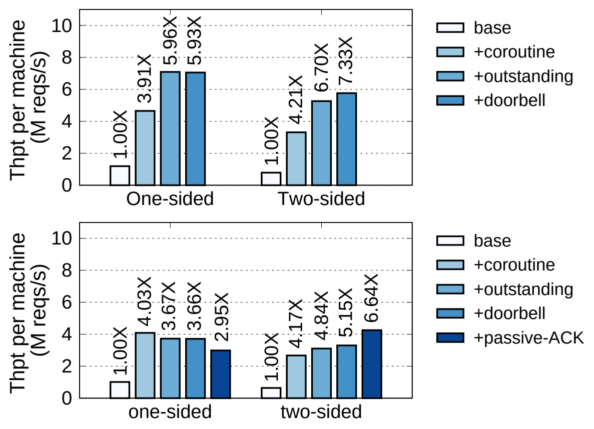

Fig. 18: *A comparison of one-sided and two-sided primitives using RoCEv2 NICs for multiple-object (a) reads and (b) writes with 256-byte payloads.*

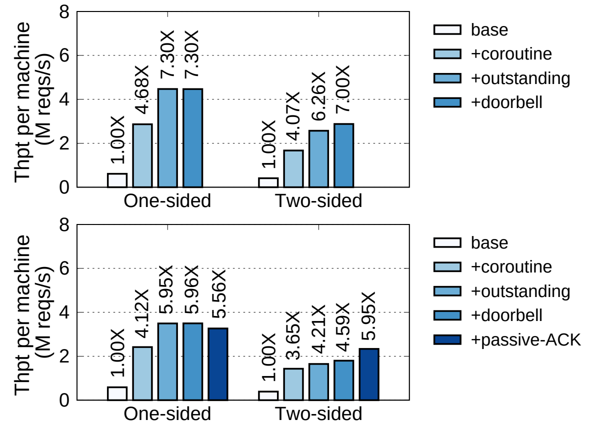

Fig. 20: *A comparison of one-sided and two-sided primitives using ConnectX-5 RNICs for multiple-object (a) reads and (b) writes with 256-byte payloads.*

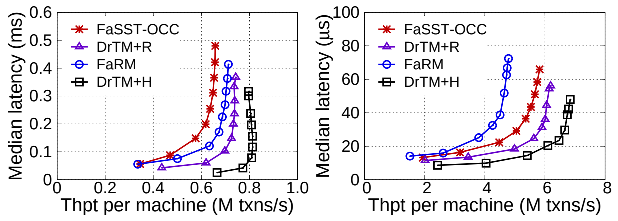

Fig. 19: *An end-to-end comparison of different designs for (a)* `TPC-C/no` *and (b)* `SmallBank` *using RoCEv2 NICs.*

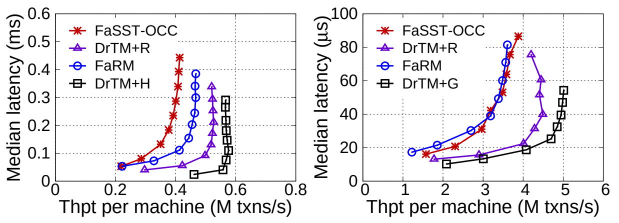

Fig. 21: *An end-to-end comparison of different designs for (a)* `TPC-C/no` *and (b)* `SmallBank` *using ConnectX-5 NICs.*

RDMA over Converged Ethernet (RoCE). RoCE is a network protocol which allows RDMA to run atop of an Ethernet network. It uses Ethernet as the link layer compared to the Infiniband network. Since RoCE only uses a different link layer, it usually has little effects on the comparison between one-sided and two-sided primitives. Fig. 18 presents the results of micro-benchmarks using ConnectX-4 RNICs in an RoCE cluster. One-sided primitive still has better performance, except when two-sided one uses the passive-ACK optimization.

Fig. 19 further presents the end-to-end comparisons of prior designs using the RoCE cluster. We can achieve similar results as in §6. Note that both workloads achieve a better performance in this experiment. This is because both workloads use 2-way replication due to the restrictions of cluster size.

ConnectX-5 (CX5). Finally, we evaluate different systems using CX5, the later product of CX4. Due to the restrictions of RNIC (1 per machine), we utilize threads on the same socket for the experiments using CX5.

Fig. 20 presents the performance of primitive level analysis. One-sided primitives can achieve better performance, even when applying passive-ack optimization for two-sided primitive. This result is as we expected since Mellanox marks CX5 and CX4 as the same generation RNIC in its document (while CX3 is the previous generation RNIC). Fig. 21 further presents the end-to-end comparisons on `TPC-C/no` and `SmallBank` using CX5. The results are similar to results when using CX4.

8 Discussion

Trends, features, and extensions. Our studies focus on Mellanox ConnectX-4 RNIC. Previous generations of RNICs like ConnectX-3 yields slower performance of one-sided `READ`s. However, we have seen a trend that one-sided primitives become faster and more scalable in recent RNICs, from Connect-IB to ConnectX-4 to ConnectX-5. Further, new generation RNIC may introduce more features for one-sided primitives. For example, ConnectX-5 integrates one-sided `WRITE` with NVM [27]. This suggests an optimistic opinion about providing offloading features in modern data centers.

On the other hand, one-sided primitive still has many limitations due to the lack of expressiveness [51]. For example, it is not competent for complicated operations, like searching in a sorted store. Furthermore, one-sided primitive is unlikely to have orders of magnitude higher performance than messaging, because we have also seen a trend on providing fast messaging rate in later generation RNICs [28]. Hence, how to properly choose the right primitive is very important given a specific workload. This paper gives an example of how to optimize transactional processing with a combination of different primitives in a phase-by-phase way. The resulting system and insights may be reused for further studies.

Some proposed RDMA extensions, including the coherence of atomic operations, atomic object reads [9], and multi-address atomics [35], may provide further exploration spaces once being commercialized. We believe that there will be a continued line of research in this field with more new features, implementations and application domains.

Emulating a large-scale RDMA cluster. Currently, we mainly focus on emulating massive RDMA connections in a rack-scale cluster, because QP cache misses will dominate the impact on the performance of various primitives. Consequently, we do not consider other scalability issues in a real large-scale RDMA cluster. For example, a large cluster has to use multiple layers of RDMA networking such as multiple switches or congestion control mechanism [60]. However, such aspects affect all network primitives instead of affecting only primitives. We plan to further validate our conclusion on a real, large RDMA-capable cluster in future.

9 Related Work

Existing RDMA optimizations. A set of optimizations on how to better leverage RDMA have been proposed. FaRM [10] proposes a set of techniques to mitigate cache pressure of RNIC, including using huge page to reduce page entries stored in RNIC and sharing QPs between threads to reduce the connections. HERD [16] first discovers the benefits of using UD QPs for messaging to improve performance and scalability. A recent guideline paper of RDMA [17] describes several optimizations on better leveraging RDMA features, including using doorbell mechanism to post a batch of requests. It also studies how low-level factors (e.g., payload inlining) impact the overall performance. FaSST [18] argues that UD, though unreliable as its name, has high reliability in modern datacenters because RDMA assumes a lossless link layer. Hence, UD QP is well suited for two-sided primitives. Finally, LITE [46] proposes a kernel indirection layer for RDMA which improves the scalability and programmability of RDMA. Many of such optimizations can be used cumulatively to improve performance. We apply most of them in our execution framework to make a fair comparison between one-sided and two-sided primitives, except for LITE. Because the optimization requires modifying the kernel and is not designed for our scenario.

Comparisons on RDMA primitives. Prior work has done valuable comparisons on different RDMA primitives [16, 10, 11, 18]. Our work continues such comparisons with a comprehensive study of both RDMA primitives and state-of-the-art optimizations. Further, we show that, even if one primitive performs better in microbenchmarks, applications still need careful choices of RDMA primitives and optimizations to achieve the optimal performance.

Fast distributed transaction systems. We continue the line of research of providing fast distributed transactions [45, 7, 59, 31, 53, 51, 1, 24, 11, 57, 1, 54, 5, 18, 56]. Some systems leverage variants of OCC for consistency [24, 11, 5, 18], while others use algorithms such as 2PL [51] and SI [56] to handle workloads with more contentions. This paper uses OCC as an example to illustrate the effectiveness of a novel combination of RDMA primitives. We believe our insights on RDMA primitives may be applied to other concurrency control algorithms.

Other RDMA-enabled systems. A large number of systems have used RDMA features to improve performance. These include transaction processing systems [51, 11, 5, 18, 56, 50], key-value stores [30, 29, 16, 10, 40], distributed file systems [26, 38], consensus algorithms [34, 48] and graph processing systems [52, 36, 58]. Such systems also have different RDMA primitive choices according to their own demands. Our study may also inspire an optimal use of RDMA primitives on such systems.

10 Conclusion

We have presented a detailed analysis of how different choices of RDMA primitive affect the performance of transactional execution. Unlike previous studies, we compare different primitives and techniques using one well-optimized RDMA framework. This makes the comparison of techniques and primitives comparable and comprehensive. The main observations made by our study is that no single primitive is the winner all the time, even at different phases of transactional execution. We then propose a hybrid solution which uses the most appropriate primitive at each phase of transactions. This not only improves the throughput but also reduces the latency of transactions. Finally, our study gives hints about whether it is cost-effective to offload RDMA one-sided, or just use two-sided for easy porting. We hope this can stimulate and provide a guideline for future co-design with RDMA.

11 Acknowledgment

We sincerely thank our shepherd Marcos Aguilera and the anonymous reviewers in OSDI for their insightful suggestions, Hong Yang for sharing his experience on RDMA. This work is supported in part by the National Natural Science Foundation of China (No. 61772335, 61572314), the National Key Research & Development Program (No. 2016YFB1000500), the National Youth Top-notch Talent Support Program of China, and Singapore NRF (CREATE E2S2).

References

[1] M. K. Aguilera, J. B. Leners, and M. Walfish. Yesquel: Scalable sql storage for web applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP'15, pages 245–262, New York, NY, USA, 2015. ACM.

[2] I. T. Association. Infiniband architecture specification. https://cw.infinibandta.org/document/dl/7859, 2015.

[3] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'12, pages 53–64, New York, NY, USA, 2012. ACM.

[4] H. Chen, R. Chen, X. Wei, J. Shi, Y. Chen, Z. Wang, B. Zang, and H. Guan. Fast in-memory transaction processing using rdma and htm. *ACM Trans. Comput. Syst.*, 35(1):3:1–3:37, July 2017.

[5] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen. Fast and general distributed transactions using rdma and htm. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys'16, pages 26:1–26:17, New York, NY, USA, 2016. ACM.

[6] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC'10, pages 143–154. ACM, 2010.

[7] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 251–264. USENIX Association, 2012.

[8] C. Curino, E. Jones, Y. Zhang, and S. Madden. Schism: A Workload-driven Approach to Database Replication and Partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57, Sept. 2010.

[9] A. Daglis, D. Ustiugov, S. Novaković, E. Bugnion, B. Falsafi, and B. Grot. Sabres: Atomic object reads for in-memory rack-scale computing. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-49, pages 6:1–6:13, Piscataway, NJ, USA, 2016. IEEE Press.

[10] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. FaRM: Fast remote memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 401–414. USENIX Association, 2014.

[11] 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 Symposium on Operating Systems Principles*, SOSP'15, pages 54–70, New York, NY, USA, 2015. ACM.

[12] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM'16, pages 202–215, New York, NY, USA, 2016. ACM.

[13] R. Harding, D. Van Aken, A. Pavlo, and M. Stonebraker. An evaluation of distributed concurrency control. *Proc. VLDB Endow.*, 10(5):553–564, Jan. 2017.

[14] M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, ISCA'93, pages 289–300, New York, NY, USA, 1993. ACM.

[15] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 121–136, New York, NY, USA, 2017. ACM.

[16] A. Kalia, M. Kaminsky, and D. G. Andersen. Using rdma efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM'14, pages 295–306. ACM, 2014.

[17] A. Kalia, M. Kaminsky, and D. G. Andersen. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference*, USENIX ATC '15, pages 437–450, Denver, CO, 2016. USENIX Association.

[18] A. Kalia, M. Kaminsky, and D. G. Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided (rdma) datagram rpcs. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 185–201, Berkeley, CA, USA, 2016. USENIX Association.

[19] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-Store: A High-performance, Distributed Main Memory Transaction Processing System. *Proc. VLDB Endow.*, 1(2):1496–1499, Aug. 2008.

[20] A. Khandelwal, R. Agarwal, and I. Stoica. BlowFish: Dynamic Storage-Performance Tradeoff in Data Stores. In *13th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'16, pages 485–500, Santa Clara, CA, Mar. 2016. USENIX Association.

[21] H. Kimura. Foedus: Oltp engine for a thousand cores and nvram. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 691–706, New York, NY, USA, 2015. ACM.

[22] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, June 1981.

[23] L. Lamport, D. Malkhi, and L. Zhou. Vertical Paxos and Primary-backup Replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC'09, pages 312–313, New York, NY, USA, 2009. ACM.

[24] C. Lee, S. J. Park, A. Kejriwal, S. Matsushita, and J. Ousterhout. Implementing linearizability at large scale and low latency. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP'15, pages 71–86, New York, NY, USA, 2015. ACM.

[25] H. Lu, C. Hodsdon, K. Ngo, S. Mu, and W. Lloyd. The snow theorem and latency-optimal read-only transactions. In *Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'16, page 135, 2016.

[26] Y. Lu, J. Shu, Y. Chen, and T. Li. Octopus: an rdma-enabled distributed persistent memory file system. In *Proceedings of the 2017 USENIX Annual Technical Conference*, USENIX ATC'17, pages 773–785, Santa Clara, CA, 2017. USENIX Association.

[27] Mellanox Technologies. Nvme over fabrics standard is released, 2018. http://www.mellanox.com/blog/2016/06/nvme-over-fabrics-standard-is-released/.

[28] Mellanox Technologies. Products overview, 2018. http://www.mellanox.com/page/products_overview.

[29] 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 Conference on Annual Technical Conference*, USENIX ATC'13, pages 103–114. USENIX Association, 2013.

[30] C. Mitchell, K. Montgomery, L. Nelson, S. Sen, and J. Li. Balancing CPU and network in the cell distributed b-tree store. In *Proceedings of the 2016 USENIX Annual Technical Conference*, USENIX ATC'16, pages 451–464, Denver, CO, 2016. USENIX Association.

[31] S. Mu, Y. Cui, Y. Zhang, W. Lloyd, and J. Li. Extracting more concurrency from distributed transactions. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 479–494, Berkeley, CA, USA, 2014. USENIX Association.

[32] S. Neuvonen, A. Wolski, M. Manner, and V. Raatikka. Telecom Application Transaction Processing (TATP) Benchmark. http://tatpbenchmark.sourceforge.net/, 2011.

[33] A. Pavlo, C. Curino, and S. Zdonik. Skew-aware Automatic Database Partitioning in Shared-nothing, Parallel OLTP Systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD'12, pages 61–72, New York, NY, USA, 2012. ACM.

[34] M. Poke and T. Hoefler. Dare: High-performance state machine replication on rdma networks. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 107–118, New York, NY, USA, 2015. ACM.

[35] S. Raikin, L. Liss, A. Shachar, N. Bloch, and M. Kagan. Remote transactional memory, 2015. US Patent App.

[36] J. Shi, Y. Yao, R. Chen, H. Chen, and F. Li. Fast and concurrent rdf queries with rdma-based distributed graph exploration. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 317–332, Berkeley, CA, USA, 2016. USENIX Association.

[37] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1150–1160. VLDB Endowment, 2007.

[38] P. Stuedi, A. Trivedi, J. Pfefferle, R. Stoica, B. Metzler, N. Ioannou, and I. Koltsidas. Crail: A high-performance i/o architecture for distributed data processing. *IEEE Data Eng. Bull.*, 40(1):38–49, 2017.

[39] M. Su, M. Zhang, K. Chen, Z. Guo, and Y. Wu. Rfp: When rpc is faster than server-bypass with rdma. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 1–15, New York, NY, USA, 2017. ACM.

[40] T. Szepesi, B. Wong, B. Cassell, and T. Brecht. Designing a low-latency cuckoo hash table for write-intensive workloads using rdma. In *First International Workshop on Rack-scale Computing*, 2014.

[41] M. Technologies. libmlx4 driver. http://www.mellanox.com/downloads/ofed/MLNX_OFED-4.0-1.0.1.0/MLNX_OFED_LINUX-4.0-1.0.1.0-ubuntu16.04-x86_64.tgz, 2017.

[42] The H-Store Team. SmallBank Benchmark. http://hstore.cs.brown.edu/documentation/deployment/benchmarks/smallbank/, 2018.

[43] The Transaction Processing Council. TPC-E Benchmark V1.14. http://www.tpc.org/tpce/.

[44] The Transaction Processing Council. TPC-C Benchmark V5.11. http://www.tpc.org/tpcc/, 2018.

[45] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Calvin: Fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD'12, pages 1–12. ACM, 2012.

[46] S.-Y. Tsai and Y. Zhang. Lite kernel rdma support for datacenter applications. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 306–324, New York, NY, USA, 2017. ACM.

[47] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy Transactions in Multicore In-memory Databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP'13, pages 18–32. ACM, 2013.

[48] C. Wang, J. Jiang, X. Chen, N. Yi, and H. Cui. Apus: Fast and scalable paxos on rdma. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 94–107, New York, NY, USA, 2017. ACM.

[49] Z. Wang, H. Qian, J. Li, and H. Chen. Using restricted transactional memory to build a scalable in-memory database. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys'14, pages 26:1–26:15. ACM, 2014.

[50] X. Wei, S. Shen, R. Chen, and H. Chen. Replication-driven live reconfiguration for fast distributed transaction processing. In *Proceedings of the 2017 USENIX Annual Technical Conference*, USENIX ATC'17, pages 335–347, Santa Clara, CA, 2017. USENIX Association.

[51] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen. Fast in-memory transaction processing using rdma and htm. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 87–104. ACM, 2015.

[52] M. Wu, F. Yang, J. Xue, W. Xiao, Y. Miao, L. Wei, H. Lin, Y. Dai, and L. Zhou. Gram: Scaling graph computation to the trillions. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 408–421, New York, NY, USA, 2015. ACM.

[53] C. Xie, C. Su, M. Kapritsos, Y. Wang, N. Yaghmazadeh, L. Alvisi, and P. Mahajan. Salt: Combining ACID and BASE in a distributed database. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 495–509. USENIX Association, 2014.

[54] C. Xie, C. Su, C. Littley, L. Alvisi, M. Kapritsos, and Y. Wang. High-performance acid via modular concurrency control. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP'15, pages 279–294, New York, NY, USA, 2015. ACM.

[55] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. Staring into the abyss: An evaluation of concurrency control with one thousand cores. *Proc. VLDB Endow.*, 8(3):209–220, Nov. 2014.

[56] E. Zamanian, C. Binnig, T. Harris, and T. Kraska. The end of a myth: Distributed transactions can scale. *Proc. VLDB Endow.*, 10(6):685–696, Feb. 2017.

[57] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. K. Ports. Building consistent transactions with inconsistent replication. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 263–278, New York, NY, USA, 2015. ACM.

[58] Y. Zhang, R. Chen, and H. Chen. Sub-millisecond stateful stream querying over fast-evolving linked data. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP'17, pages 614–630, New York, NY, USA, 2017. ACM.

[59] Y. Zhang, R. Power, S. Zhou, Y. Sovran, M. K. Aguilera, and J. Li. Transaction chains: Achieving serializability with low latency in geo-distributed storage systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP'13, pages 276–291. ACM, 2013.

[60] 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 *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM'15, pages 523–536, New York, NY, USA, 2015. ACM.