A Survey of Storage Systems in the RDMA Era

Shaonan Ma, Teng Ma, Kang Chen, and Yongwei Wu, *Senior Member, IEEE*

Abstract—Remote Direct Memory Access (RDMA) based network devices are increasingly being deployed in modern data centers. RDMA brings significant performance improvements over traditional network devices such as Ethernet due to its unique features: *protocol offloading* and *memory semantics*. In particular, it can achieve microsecond level latency, which is about 2∼3 orders of magnitude improvement. With such improvement in hardware, the software stack, including device drivers and programming libraries, is becoming a new performance bottleneck. Developers need to use new programming libraries to take full advantage of the performance of the underlying hardware. Storage systems are very important in modern data centers. This article surveys the current efforts to use RDMA for optimizing storage systems. We first present five classes of RDMA-based storage systems, including key-value stores, file systems, distributed memory systems, database systems, and systems using smart NICs, to demonstrate different design choices. Then, we examine the core modules of storage systems from different perspectives: communication mode, concurrency control, fault tolerance, caching, and resource management. Finally, we provide some design guidelines for new RDMA-based storage systems, as well as a discussion of opportunities and challenges.

Index Terms—Network, storage, RDMA, RPC, key-value store, file system, distributed memory, smart NIC

1 Introduction

Current distributed data center storage systems can be equipped with high-throughput, low-latency storage media and networking devices. The latency of NVMe SSDs can reach tens of microseconds. A storage node with multiple NVMe SSDs can easily provide 10GB/s of throughput [1], which is far beyond what the traditional network can match. Current persistent memory (PM) devices can even persist data in around 100 nanoseconds [2], [3]. On the network side, RDMA networks also offer low latency and high bandwidth with unique features including *protocol offloading* and *memory semantics*. Protocol offloading refers to offloading the network protocol onto the RDMA NICs (RNICs), which greatly saves CPU cycles. Memory semantics provides the capabilities to access remote memory directly bypassing kernels of both sides, without remote CPU involvement. Memory semantics can reduce context switching and data copying. Both features help RDMA networks to reach high bandwidth and low latency. The latest RDMA devices can achieve $600ns$ access latency and 200Gbps bandwidth [4]. As network protocol overhead is greatly reduced and network performance improves, the bottleneck in storage systems shifts from hardware to software [5]. The use of RDMA networks and the combination of recent fast storage media to optimize storage systems is a popular research topic, which has attracted a lot of attention in both industry and academia.

RDMA provides a different programming model compared to the traditional socket programming model. In order to be compatible with traditional communication libraries, one can go through a protocol conversion approach. But the conversion approach does not fully leverage the performance of the underlying hardware for fast networks. Since many storage systems in data centers were originally designed for slow network devices, they cannot fully exploit the performance of RDMA. As a result, there have been many efforts in recent years to design and optimize RDMA-based storage systems. Researchers have proposed different approaches to redesign the various modules of storage systems to match the high speed and unique characteristics of RDMA networks [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25]. In the industry, many Internet-scale companies (e.g., Microsoft [26], [27], Google [28], Alibaba [29], [30], [31]) also have shared their experiences in deploying and using RDMA in large data centers.

To understand the ways and principles of RDMA usage in storage systems, this paper investigates various storage systems and their associated enabling technologies. The survey is conducted through the following three steps.

1) We study five classes of storage systems, namely key-value stores (Section 3.1), file systems (Section 3.2), distributed memory systems (Section 3.3), database systems (Section 3.4), and systems using smart NICs (Section 3.5). We discuss their special considerations for RDMA at the software level. Some systems use high-speed storage media, such as DRAM or PM, for

- *Shaonan Ma is with the Department of Computer Science and Technology, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100190, China, and also with Beijing HaiZhi XingTu Technology Co., Ltd, Beijing 100083, China. E-mail: msn18@mails.tsinghua.edu.cn.*
- *Teng Ma is with Alibaba Group, Hangzhou 311121, China. E-mail: sima.mt@alibaba-inc.com.*
- *Kang Chen and Yongwei Wu are with the Department of Computer Science and Technology, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100190, China. E-mail: {chenkang, wuyw}@tsinghua.edu.cn.*

Manuscript received 29 June 2021; revised 21 June 2022; accepted 30 June 2022. Date of publication 12 July 2022; date of current version 23 August 2022.
This work was supported in part by the National Key Research & Development Program of China under Grant 2020YFC1522702, in part by the Natural Science Foundation of China under Grants 62141216 and 61877035, and in part by the Tsinghua University - Meituan Joint Institute for Digital Life.
(Corresponding author: Kang Chen.)
Recommended for acceptance by K. Mohror.
Digital Object Identifier no. 10.1109/TPDS.2022.3188656

TABLE 1
Categories of RDMA-Based Storage Systems and Software Techniques

System Types	Related Works
Key-value Store	HERD [6] ccKVS [7] FaSST [8] Pilaf [9] RFP [10] HydraDB [11] C-Hint [12] DrTM [13] FaRM [14] Nessie [15] RStore [16] ScaleTX [17] Cell [18] Catfish [19] NAM-Tree [20] NVDS [21] FlatStore [22] RDMP-KV [23] RACE [24] RAMCloud [25] Sherman [32]
File System	CephFS [33] GlusterFS [34] Crail [35] NVFS [36] Octopus [5] Orion [37] FileMR [38] Assise [39] DeltaFS [40] GekkoFS [41] DAOS [42] PolarFS [43] Lustre [44] GPFS [45] BeeGFS [46] PVFS2 [47]
Distributed Memory	FaRM [14] RackOut [48] Grappa [49] InfiniSwap [50] Hotpot [51] Clover [52] AsymNVM [53] Kona [54] CoRM [55]
Databases	NAM-DB [56], [57] Chiller [58] PolarDB Serverless [59] D-RDMA [60] Zamanian *et al.* [61] Li *et al.* [62] HyPer [63] Barthels *et al.* [64] I-Store [65] L5 [66] Liu *et al.* [67]
Smart NICs	FlexNIC [68] KV-Direct [69] Lynx [70] StRoM [71] LineFS [72] Xenic [73] 1RMA [28] D-RDMA [60] HyperLoop [74]
Core Modules	**Related Works**
Communication Mode	DrTM-H [75] Cell [18] Catfish [19] Storm [76] DaRPC [77] HERD [6] FaSST [8] RF-RPC [78] ScaleRPC [17] Storm [76] Octopus [5] FlatStore [22] LITE [79] eRPC [80] Accelio [81] Mercury [82] X-RDMA [29] FLOCK [83] DFI [84] HatRPC [85]
Concurrency Control	DrTM [13] FaRM [14] Cell [18] NAM-Tree [20] Pilaf [9] RACE [24]
Fault Tolerance	HydraDB [11] Mojim [86] Orion [37] Tailwind [87] HyperLoop [74] DARE [88] APUS [89] Derecho [90] Odyssey [91] INEC [92] Aguilera *et al.* [93] Zamanian *et al.* [61]
Caching Resource Management	GAM [94] Aguilera *et al.* [95] DrTM [13] HydraDB [11] C-Hint [12] XStore [96] RACE [24] Kumar *et al.* [97] HERD [6] FaSST [8] FaRM [14] LITE [79] ScaleRPC [17] X-RDMA [29] FLOCK [83]

back-end data storage, which can fully exploit the performance of RDMA [5], [98], [99].

2) To understand the impact of RDMA on different modules in the storage system, we identify five important modules, namely communication mode (Section 4.1), concurrency control (Section 4.2), fault tolerance (Section 4.3), caching (Section 4.4), and resource management (Section 4.5). The communication mode describes how the nodes talk to each other, such as one-sided operations, two-sided operations, or RPC mode. Concurrency control and fault tolerance are indispensable modules to ensure correctness and availability. Caching is used to improve performance based on locality. The resource management module tries to manage RDMA resources in an efficient way.
3) Based on the analysis and evaluation, we provide practical guidelines on how to use RDMA effectively (Section 5.1). In addition, we compare the differences in RDMA research between industry and academia (Section 5.2) and discuss future work (Section 5.3).

To the best of our knowledge, this is the first comprehensive survey on RDMA-based storage systems. Other surveys on RDMA focus on the network layer [100], which can be considered as prior and complementary material. We consider papers presented at major conferences in the field of computer systems. Table 1 shows the papers surveyed.

The rest of this survey is organized as follows. Section 2 presents a preliminary overview of RDMA. Section 3 describes five classes of RDMA-based storage systems and their research roadmaps. Section 4 describes the five core modules of storage systems in detail. Section 5 summarizes some practical guidelines from existing works and possible future work on RDMA-based storage systems. Section 6 concludes the entire paper.

2 RDMA Preliminaries

Currently, RDMA is a broad concept used to describe a range of hardware and software techniques for remote memory reads and writes. This survey focuses more on the interface level, such as RDMA Verbs [101] and associated programming models, from a systems research perspective. We are concerned with system implementation using RDMA rather than the network layer, i.e., for unique features we are more concerned with *memory semantics* rather than *protocol offloading*. Most of the systems we study in this survey use RDMA Verbs, or similar user-space libraries such as Libfabric [102]. The differences between user-space and kernel-space interfaces are discussed in Section 5.1.

2.1 Hardware Architecture

There are currently four RDMA implementations: InfiniBand, RoCE, RoCEv2, and iWARP [101]. InfiniBand uses special RNICs and switches. The other three are Ethernet-enabled RDMA network protocols. With specially designed RNICs and switches, InfiniBand can now offer the best performance. The latest generation of RNIC ConnectX-6 [4] can reach 215MPPS (million packets per second), 600ns latency and 200Gbps throughput.

Fig. 1 shows a typical RNIC hardware connection under the NUMA architecture. Each RNIC is connected to a CPU's PCIe controller. There is on-chip memory (SRAM) on the

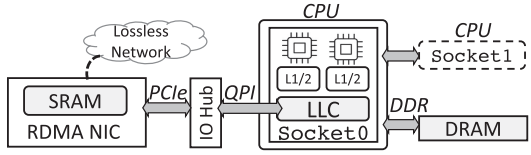


Fig. 1. Hardware connections with RNIC (QPI: QuickPath Interconnect, LLC: Last-Level Cache).

TABLE 2
Transport Modes and Supported RDMA Operations

Mode	Send	Write	Read	Atomic	MMS	Connection
RC*	✓	✓	✓	✓	2GB	1-to-1
UC	✓	✓	✗	✗	2GB	1-to-1
UD	✓	✗	✗	✗	MTU	1-to-n
SRD	✓	✗	✗	✗	MTU	1-to-n
DCT*	✓	✓	✓	✗	2GB	1-to-n

(: Reliable Transmission, MMS: Maximum Message Size).*

RNIC for caching memory address translation table (MTT), memory protection table (MPT), QP (queue pair) data, and WQE (work queue entry, see Section 2.3) [6], [17], [79]. RNIC requires MTT and MPT to locate pages and check permissions for each RDMA operation. However, the capacity of SRAM is limited [76], [79]. If the size of the metadata exceeds the capacity of SRAM, performance degrades because RDMA operations need to wait for the metadata to move from main memory to SRAM via PCIe.

2.2 Communication

RDMA uses queue pairs (QPs) for communication. Before communication, a connection between two sides needs to be established in three steps [101].

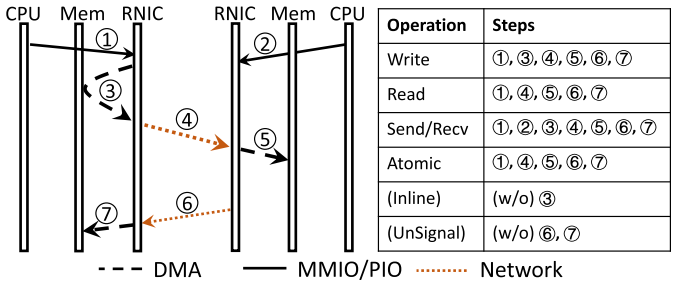
1) Each side creates one queue pair (QPs). Each QP is a memory-mapped structure containing two FIFO work queues (WQ), a sending queue (SQ) and a receiving queue (RQ). Each work queue is associated with a completion queue (CQ). Notice that a CQ can be shared by multiple WQs.
2) A memory region (MR) is registered for remote access, and a key is generated to access this MR.
3) Both sides exchange keys and addresses of MRs.

After the connection is established, both sides can send and receive work requests (WR) via WQ. When a WR is in the WQ, it can also be called a work queue entry (WQE).

2.3 Programming Paradigms

RDMA Semantics. The communication paradigms of RDMA can be roughly divided into two categories: *Memory Semantics* (one-sided operations) and *Channel Semantics* (two-sided operations). Memory semantics provide rich memory operations similar to their local memory counterparts: remote memory read/write/atomic operations (RDMA Write/Read/Atomic). RDMA atomic operations support remote memory synchronization including Compared And Swap (CAS) and Fetch And Add (FAA). While the initiator CPU starts the RDMA operation, memory semantics does not need any involvement of the target CPU, i.e., the target RNIC is responsible for accessing the remote memory. Thus, the processes running in the initiator can directly access the remote memory in the target through one-sided operations. Channel semantics is similar to traditional communication paradigms such as socket programming or RPC, where target CPUs are needed to process the requests.

Transport Modes. RDMA Verbs can support three transport modes: Reliable Connection (RC), Unreliable Connection (UC), and Unreliable Datagram (UD). Two-sided operations are supported by all transport modes. As shown in Table 2, RC and UC support RDMA Write, while only RC supports RDMA Read/Atomic. Recent works also propose new transport modes. Amazon AWS uses SRD (scalable reliable datagram) [103] to enjoy both the reliability of RC and the scalability of UD. Some new generation RNICs (e.g., ConnectX-6 [4]) support a new experimental feature called DCT (Dynamically-connected Transport) [104], which can achieve good scalability through QP sharing and reestablishment.



Operation	Steps
Write	①, ③, ④, ⑤, ⑥, ⑦
Read	①, ④, ⑤, ⑥, ⑦
Send/Recv	①, ②, ③, ④, ⑤, ⑥, ⑦
Atomic	①, ④, ⑤, ⑥, ⑦
(Inline)	(w/o) ③
(UnSignal)	(w/o) ⑥, ⑦

Fig. 2. The transmission between CPU and RNIC (MMIO: Memory-mapped I/O, PIO: Programmed I/O).

Fig. 2 shows the life cycles for different kinds of RDMA operations. The table in Fig. 2 shows the data flows required for each operation. If using a two-sided operation, when the initiator CPU sends a request, a WR will be posted with two steps [99]: 1) MMIO/PIO, CPU initiates network operations by sending a message to the NIC (①); and 2) DMA, RNICs access data from DRAM through PCIe without involving the CPU (③). The target side should additionally post a receiving WR to its RQ (②). It also transfers request data from the RNIC to DRAM by DMA and produces a completion queue entry (CQE) to the corresponding CQ to notify the end of the WR (⑤). When the RDMA Send is finished, the initiator CPU will also produce a CQE (⑦). If using a one-sided operation, the behavior on the initiator side is similar to a two-sided operation. ⑤ is a DMA read for RDMA Read or a DMA write for RDMA Write . Meanwhile, ② is not required because of kernel bypass, and the CQE on the target side has no need to be generated in ⑤. There are also some optimizations (e.g., *inline*, *unsignal*) to reduce network traffic and data transfer between local CPU and RNIC [99], [105].

3 RDMA-Based Storage Systems

This section studies five classes of RDMA-based storage systems: key-value stores (Section 3.1), file systems (Section 3.2), distributed memory systems (Section 3.3), database systems (Section 3.4), and systems using smart NICs (Section 3.5). We use system implementations in each class to show the typical usage of RDMA in the corresponding class, together with the discussions on design considerations, challenges and drawbacks.

3.1 Key-Value Stores

There are two typical key-value stores (KVS): hash-based KVS (Section 3.1.1) and tree-based KVS (Section 3.1.2). Hash-based KVS has constant time for point query while tree-based KVS can support range query. We also discuss a special kind of KVS implementation combining RDMA and persistent memory (PM) (Section 3.1.3). Table 3 summarizes several research efforts based on different indexes (hash or tree data structures) and their implementation details.

TABLE 3
RDMA-Based Key-Value Stores (*Index: Hash or Tree Data Structures)

Name	Index	PUT	GET	Replication	Cache	Transaction	Race Detection
Pilaf [9]	Cuckoo	Send	Read	✗	✗	✗	checksum
HERD [6]	MICA	Write(UC)+Send(UD)	Write(UC)+Send(UD)	✗	✗	✗	server-side
FaSST [8]	MICA	Send(UD)	Send(UD)	✓	✗	✓	server-side
RFP [10]	Bucket Hash	Write+Read	Write+Read	✗	✗	✗	server-side
FaRM [14]	Hopscotch	Write	Read	✓	✗	✓	cacheline version
ccKVS [7]	MICA	Send(UD)	Send(UD)	✓	✓	✗	partition
HydraDB [11]	Compact Hash	Send	Read	✓	✗	✗	key-value version
C-Hint [12]	Compact Hash	Send	Read	✗	✓	✗	key-value version
Nessie [15]	Cuckoo	Write+CAS	Read	✗	✗	✗	timestamp
DrTM [13]	Cluster	Write	Read	✓	✓	✓	lease-based lock
RStore [16]	Cluster	Write	Read	✗	✓	✓	COW
ScaleTX [17]	MICA	Read+Write	Read+Write	✓	✗	✓	server-side
RACE [24]	Extendible Hash	Write+CAS	Read	✗	✓	✗	RDMA Atomic
Cell [18]	B-Tree	Send	Send/Read	✗	✓	✗	key-value version
Catfish [19]	R-Tree	Read/Write	Read/Write	✗	✗	✗	server-side
NAM-Tree [20]	B-Tree	Send/Write+CAS	Send/Read	✗	✗	✗	OLC

3.1.1 Hash-Based KVS

Hash-based KVS using RDMA can be classified into three categories: 1) one-sided design uses only one-sided operations, 2) two-sided design uses only two-sided operations, and 3) hybrid design uses both one-sided and two-sided operations. We start our discussion with the hybrid design since it was proposed earlier than the other two.

Hybrid Design. Pilaf [9] is the first remote in-memory KVS to optimize GET operation using RDMA Read. Its PUT is based on two-sided design to simplify the processing logic. Pilaf has two memory areas, a hash table area to store data addresses and a data area to store real data. The one-sided GET reduces the CPU consumption of the server, but introduces two network round trips (one to look up the hash table to get the data address, and the other to fetch the data). For two-sided PUT, only one network round trip is needed. PUT can cause hash collisions and data movement in the hash table. Handling these tasks using one-sided operations would be complicated, so Pilaf chooses two-sided operations to enforce target CPU to complete these tasks. Pilaf uses checksum to guarantee the consistency of GET, i.e., partially updated data will be detected and discarded in GET. HydraDB [11] and C-Hint [12] use similar storage layouts and communication modes. HydraDB detects data races by using version numbers (PUT increases the version twice, thus GET can detect concurrent PUT if the version number is odd). C-Hint further implements an RDMA-aware cache. Because the server is not aware of the one-sided GET, clients explicitly use RDMA Write to tell the server about hot data and access history. The server can then run the cache replacement policies. HERD [6] is a little different in that it implements RPC with Write under UC mode and Send under UD mode (details in Section 4.1.2).

One-Sided Design. This category becomes prevalent because of the disaggregation architecture separating computation and storage, with no computational power on the storage side [52]. RACE [24] is a hash-based KVS for disaggregated memory. It chooses extendable hashing to reduce the overhead of resizing. It caches directories on the client-side to reduce one RDMA Read. When the hash table is resizing, the client cache may become stale. RACE adds a header to each bucket to detect stale caches. For PUT, RACE fixes each entry field to 8 bytes so that remote atomic writes can be performed using RDMA CAS, eliminating the overhead of locks (remote locks can result in multiple round trips). Previous one-sided systems are not as optimized as RACE. FaRM [14] uses RDMA Write to implement the RPC primitives, achieving the same functionality as the two-sided operations. DrTM [13] uses RDMA CAS to implement remote locks, resulting in higher overhead than RACE. DrTM is also the first system to combine RDMA and HTM (Hardware Transactional Memory) to support distributed transactions.

Two-Sided Design. Despite the attractive features of one-sided operations, Kalia *et al.* [8] demonstrate that systems based on two-sided operations can also achieve comparable performance and better scalability. One-sided operations need to create connections before communication, which consumes on-RNIC SRAM space. A large number of connections can overflow SRAM, leading to performance degradation (Section 2.1). FaSST [8] only uses RDMA Send/Recv under UD mode to improve scalability. It is a viable choice because with the help of InfiniBand hardware, packet loss under UD mode is extremely low. RAMCloud [25] is also a fast in-memory KVS using two-sided design. It presents that one-sided operation is less suitable for KVS due to multiple network round trips.

Discussion. Using one-sided operations in hash-based KVS to bypass the server's CPU seems to be a huge improvement, but it still comes with several challenges such as data races and extra network round trips. To address these challenges, data validation and client-side caching mechanisms are necessary. Also, keeping the cached data fresh is a new challenge. Due to these challenges, some systems [8], [25] have abandoned this primitive and adopted only two-sided design for programmability and scalability. However, in some scenarios, such as disaggregated memory, one-sided operations are still essential and become more important for their rich RDMA memory semantics.

3.1.2 Tree-Based KVS

Tree-based KVS are more complex than hash-based KVS. It is of great challenge to have efficient implementations only using one-sided or two-sided operations. Most existing

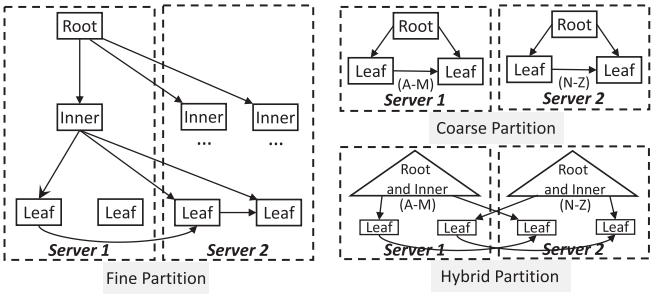


Fig. 3. Different partition strategies of distributed B-Tree.

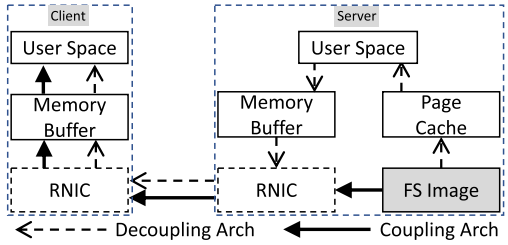


Fig. 4. Data copying in distributed file systems.

works [18], [19], [20] use the hybrid design and may even switch between one-sided and two-sided operations based on the system load.

Tree partition is important in the tree-based KVS. There are three common ways to partition a tree (Fig. 3). (1) Coarse partition divides the tree based on the key ranges (e.g., A-M and N-Z in Fig. 3). Each server stores a subtree. The drawback of such design is that it is hard to handle skew workloads. (2) Fine partition assigns nodes to servers randomly. It can reach load balance but tree traversal needs excessive network communication. (3) Hybrid partition takes the advantages of above approaches. The leaf nodes are partitioned randomly. Other nodes are partitioned by range. Since coarse partition has almost same implementation as a single node implementation, we only discuss the fine and hybrid partitions here.

Fine Partition. Cell [18] partitions B-Tree into multiple "fat" nodes (64MB) across servers. Each fat node is a small local B-Tree. The server-side serves read and write requests, and the client-side handles read-only requests via RDMA Read. Depending on the workload and resource usage, Cell can dynamically switch between one-sided client-side read and two-sided server-side read. However, even with dynamic switching, its partition strategy leads to a high number of network round trips when the data size is large. The client cache of the nodes near the root can alleviate this problem.

Hybrid Partition. NAM-Tree [20] adopts the hybrid partition. Internal nodes of NAM-Tree use coarse partition based on key ranges. Leaf nodes use fine partition. In addition, it proposes a hybrid access approach. For GET, it traverses the internal nodes by using two-sided operations and fetches the data in the leaf node using RDMA Read. PUT is similar. RDMA Write is used to update the leaf data.

Discussion. Tree-based KVS and hash-based KVS face different challenges using RDMA. A good partition will generate fewer network round trips and prevent load imbalance. Obviously, the hybrid partition strategy requires only two round trips per operation, which keeps network overhead well under control compared to fine partition.

3.1.3 KVS With Persistent Memory

KVS with persistent memory is special because PM provides both byte-addressability and persistency. There are new challenges to implement KVS combining RDMA and PM, such as the granularity mismatch and persistence overhead. Several works [98], [106] have evaluated the performance of remote PM and have proposed some optimizations. FlatStore [22] is a representative work. Since the size of PCIe data words does not match the size of PM blocks, it uses a log-based structure in PM and proposes horizontal batch processing to persist requests in the log with the aim of eliminating unnecessary writes in PM. It uses volatile indexes to quickly locate data in the logs, mitigating the performance loss of searching in PM. Pure one-sided accesses from the client require multiple round trips, so FlatStore designs FlatRPC to alleviate this problem.

Discussion. RDMA and PM provide a great opportunity to build fast and persistent distributed KVS. RDMA and PM have different hardware parameters such as access granularity and persistence mechanisms. The current RDMA does not yet consider persistence, which may change in the future. We would like to see more hardware and software efforts to build fast and crash-consistent systems based on RDMA and PM.

3.2 File Systems

RDMA provides new mechanisms to boost the performance of distributed file systems. We divide existing works into two categories (Fig. 4): *decoupling architecture* and *coupling architecture*. The *decoupling architecture* decouples the storage and network devices, similar to the traditional distributed file systems using block devices. The *coupling architecture* couples the storage and network devices as the storage devices are persistent memory. File systems can directly access the remote persistent memory through RDMA network.

Decoupling Architecture. Current production RDMA-based file systems usually use the decoupling architecture because they have to use block devices (e.g., NVMe SSD) that do not support byte-addressable access. As a result, existing works mainly focus on using RDMA-optimized communication modules to improve network performance. CephFS [33] uses Accelio [81], an asynchronous RPC library, to support RDMA. Similar approaches are used by GlusterFS [34] and PolarFS [43]. Crail [35] uses DaRPC [77] which adopts asynchronous I/O for accessing remote storage and reduces data copying on the server-side. Lustre [44] uses RDMA in its network communication module LNet [107] to accelerate bulk data movement. LNet is a Linux kernel module supporting a variety of network driver plug-ins. GPFS [45] uses multiple QPs between each pair of nodes to pursue higher throughput. In addition, GPFS automatically switches to TCP/IP protocol to increase system availability in the case of RDMA failure. PVFS2 [47] proposes two protocols to serve different scenarios, eager and rendezvous. The former is mainly used for short messages with send and receive ring buffers, and the latter is designed for long messages with RDMA Read/Write. Similarly, PVFS2 and BeeGFS [46] can also switch between

multiple network protocols. There are other communication building blocks available. DeltaFS [40] and GekkoFS [41] use an RPC framework called Mercury [82]. DAOS [42] implements an RDMA-based RPC CaRT [108]. DAOS uses user-mode drivers wherever possible (e.g., DPDK, SPDK) to eliminate the high latency of context switch. In addition, DAOS has a tiered storage design, PM for small-sized data and NVMe SSD for large-sized data.

Coupling Architecture. The coupling architecture is special to combine RDMA and PM as we can use one-sided operation to access the remote PM. Octopus [5] is among the first ones. It directly accesses remote and local PM through RNIC and DAX [109] respectively, reducing the number of memory copying in traditional file systems (Fig. 4). It allows clients to retrieve file metadata via RDMA-based RPC and access the data directly through a one-sided operation. This design reduces the involvement of server CPUs and can effectively improve the throughput. However, Octopus implements only a simplified file system with limited metadata management. For example, all files are indexed by their path names in a static distributed hash table, and operations such as `rename` would lead to high overhead. As a result, Octopus cannot adapt to complex workloads and large-scale deployment.

Orion [37] follows the same coupling architecture and addresses the remaining issues, such as fault tolerance and metadata management, in Octopus. Orion creates inode logs for each file in PM in the metadata server. Clients need to reconstruct the metadata using logs while opening a file. First, clients initiate the log synchronization using RDMA Send to notify MDS. Then, MDS uses RDMA Write to transfer the first log page. Later, clients use RDMA Read to fetch additional log pages. Finally, clients use one-sided operations to directly access the data stored in data servers. RDMA Send is used to send the corresponding metadata changes (e.g., file size) to MDS.

Discussion. Traditional file systems sitting in the kernel suffer from the context switch and multiple data copying. RDMA drastically reduces the latency and CPU consumption, making the software a new bottleneck. Thus, the coupling architecture which completely refactors the software layer can fully utilize the performance of the underlying hardware. However, for compatibility and stability reasons, we will continue to see the improvement of the decoupling architecture in current distributed file systems.

3.3 Distributed Memory Systems

Distributed memory systems consolidate memory modules from multiple machines to form a unified address space. This architecture can benefit in-memory data processing applications.

FaRM [14] implements a shared address space and supports transaction processing, using RDMA-based communication primitives (Section 3.1.1). Its shared address space contains multiple 2GB-sized memory regions (MRs). Furthermore, FaRM guarantees fault-tolerant transaction processing using a distributed concurrent control protocol based on the optimistic concurrent control. FaRM tries to use one-sided operations as much as possible. The protocol commits to backup before primary to ensure recoverability.

Unlike FaRM in user-space, Hotpot [51] provides a kernel-level distributed shared memory based on the page fault handler. When a page fault occurs, it fetches the corresponding page from local or remote and then makes a local copy as a cache. Hotpot's network layer uses two-sided operations. To reduce CPU overhead, it shares a ring buffer for all connections and uses a thread for polling.

Kona [54] cooperates with cache coherence protocol to work at the cache-line granularity. The above efforts (FaRM and Hotpot) work at the coarser granularity (object or page). The cache-line granularity can effectively mitigate the write amplification issue. Kona proposes an FPGA-based memory manager unit (by simulation) to pull data from remote memory in the case of cache misses. The memory management unit also needs to keep track of dirty data and write back the dirty data to remote memory via RDMA Write for cache eviction.

Discussion. RDMA-based distributed memory systems still share the same performance problem as traditional distributed memory systems because the network performance is still much lower than the memory system in a single node. However, with performance improvement of the RDMA network, such systems can now boost in-memory processing applications. We can see the practical usage of such systems [110]. With the prevalence of PM, we can see more improvements in distributed shared persistent memory systems.

3.4 Database Systems

Undoubtedly, RDMA can improve the performance of various modules in a database system, including storage, transaction processing and query execution.

Storage. Li *et al.* [62] explore how to use RDMA to extend the local memory to accelerate SQL Server [111]. Using memory as a buffer is a common way to speed up data access in database systems. However, due to the limited buffer size, cache misses can lead to expensive storage accesses. Thus, this work exposes the remote memory as an extended buffer through RDMA. Inside the implementation, each MR is represented as a file and can be accessed by using one-sided operations.

Transaction Processing. NAM-DB [56], [57] uses RDMA-based snapshot isolation (RSI) to deal with the poor scalability problem of the traditional two-phase commit protocol (2PC) based snapshot isolation (SI). As a result, there will be six round trips in 2PC+SI: (1) a client sends the request, (2) the transaction manager gets a timestamp from the timestamp service, (3) prepare phase, (4) commit phase, (5) the transaction manager updates the timestamp service, and (6) the client receives a response. Such a long latency increases the transaction contention and the abort rate. NAM-DB uses new network-attached memory (NAM), which supports RDMA memory semantics. The transaction processing can run on the compute node (also as the transaction manager). The number of network round trips is reduced to four by using one-sided operations. Based on a similar architecture, PolarDB Serverless [59] further optimizes remote latching (RDMA CAS) and global timestamp (RDMA FAA).

Query Execution. D-RDMA [60] points out that RDMA has a high overhead when transferring small data. This phenomenon hurts the database query performance because

query results are often small data scattered in tables. D-RDMA uses non-contiguous regions (NCRs) to replace scatter-gather elements (SGEs) in RDMA Verbs. This structure consists of multiple contiguous blocks. Each block contains data and gaps, and gaps appear at the same locations in all blocks. D-RDMA also extends the regular RNIC by adding an optimizer. After receiving the information of NCRs, the NIC uses the optimizer to first generate and then run the fastest DMA execution plan to send data. The optimizer knows the locations of all data and gaps for NCRs. It will merge the data areas, choosing whether to fetch them along with the gaps. Fetching gaps increases the amount of data access but reduces the number of small DMAs. The optimizer will calculate all possible execution plans based on tunable parameters (e.g., gap size).

*Discussion.*Databases may be the most widely used storage system today. Existing database systems have a similar layered design, including a storage engine (also with buffer pool management), concurrent control method, query analysis, and execution engine. All the modules need to consider the RDMA to harness the improved network performance. Most of the current works focus on one specific module. It is a big challenge to integrate the high-performance RDMA network into database systems fully.

3.5 Systems Using Smart NICs

One-sided and two-sided operations have their own advantages and disadvantages. Despite the effective use of current hardware, network architecture changes are also attractive for further improvements. Smart NICs try to extend existing interfaces. Smart NICs have more computing power and are more flexible than normal RNICs.

StRoM [71] uses smart NICs in near-data processing by offloading some data processing operations to the FPGA-based RoCEv2 NICs. StRoM introduces four programmable kernels: (1) accessing remote data structures, (2) remote data checksum, (3) data shuffling, and (4) data access statistics. A single network round trip is enough to complete complex operations using these kernels, which is equivalent to a two-sided operation. At the same time, no remote CPU is needed, which is the same as a one-sided operation. StRoM places these kernels on the data path between the RoCE stack and the DMA engine to increase the bandwidth and reduce the latency. KV-Direct [69] observes a similar problem and implements the key-value operations (GET, PUT, DELETE) inside smart NICs.

LineFS [72] is a high-performance distributed file system using smart NICs with client-side data cache in PM. It is natural to offload some common file system operations, such as data replications, compression, data publishing, and indexing, to smart NICs. However, the computation power of smart NICs is much lower than CPU. LineFS proposes the *persist-and-publish* model to improve the parallelism of the system, i.e., host cores and smart NIC cores to work cooperatively. For example, the data persistence and replication can work in a pipeline fashion, i.e., host cores make sure the data are persisted in PM, and then smart NIC cores replicate persisted data.

Other than RDMA Verbs extension above, some works customize application-specific network protocols in smart NICs. 1RMA [28] deviates from the standard RDMA protocol and tries to design a software and hardware cooperation protocol to meet Google's multi-tenant scenario. For example, 1RMA abandons the concept of connection because connection in RDMA is not scalable. Thus, 1RMA has to deal with congestion control on its own. In addition, 1RMA ensures data security for multi-tenant.

*Discussion.*The programmability of smart NICs provides great flexibility in system designs. The key to effectively using smart NICs is finding the appropriate offloading operations. Considering the limitations of current RDMA Verbs, it is a great opportunity and a challenge to apply smart NICs in various storage systems.

4 Core Modules of Storage Systems

This section discusses five core modules that play important roles in the design of RDMA-based storage systems.

4.1 Communication Mode

The communication mode presents how applications use the RDMA primitives to talk to each other. RPC (i.e., two-sided) is the classical mode in traditional networks. RDMA introduces one-sided operation, which brings a richer set of primitives. These primitives offer new ways to implement communications between nodes in the system. We roughly divide the communication methods into two modes: customized mode and RPC mode. The customized mode attempts to improve the performance without traditional restrictions, while the RPC mode is compatible with the traditional mode to support existing applications.

4.1.1 Customized Mode

One-sided and two-sided operations have their advantages and disadvantages. In many cases, it is not enough to use just one type. As discussed in Section 3.1, communications in the customized mode try to use both operations (e.g., Pilaf [9], C-Hint [12]).

Cell [18] balances the client-side and server-side processing to maximize the search throughput. Server-side processing performs better by reducing the network traffic but may overload a server. Therefore, clients can switch to client-side search (using RDMA Read) when a server is considered overloaded. Catfish [19] implements two types of communications: fast messaging (RPC) and RDMA offloading (one-sided operations). It monitors CPU utilization and the number of network round trips to predict the workload and determines the appropriate type. Storm [76] proposes a dynamic access strategy called *one-two-sided*. Pointer chasing usually needs multiple network round trips if using one-sided operations, in which case two-sided operations are more suitable. Storm will first fetch a block of data (maybe more than needed size) via RDMA Read and then switch to two-sided operations for pointer chasing if necessary. DrTM-H [75] evaluates the effects of one-sided and two-sided operations for different phases of distributed transaction processing (execution, validation, logging, commit). The conclusion is that using only one mode alone cannot achieve the best performance, and a hybrid strategy is better.

DFI [84] is special that it tries to propose new communication interfaces. It proposes a new set of data flow interfaces on InfiniBand for various storage systems. The flow

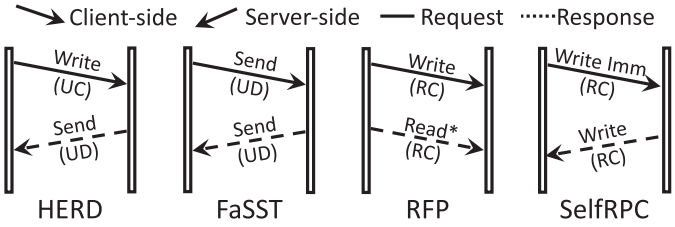


Fig. 5. Different RPC paradigms (* denotes repeat more than once).

abstraction represents the data movement between nodes. There are three flow types (Shuffle, Replicate, and Combiner). These are building blocks for more complex communication topologies that meet application requirements. DFI also makes optimizations for different flows.

Discussion. Customized mode tries to achieve higher performance but loses generality. Experienced programmers familiar with RDMA prefer the customized mode to get higher performance. However, RDMA's rich set of primitives and parameters makes it complex to use in practice. The complexity of data-center applications makes such an approach even harder. Thus, many practical systems prefer the RPC (two-sided) mode.

4.1.2 RPC Mode

RPC is a very important communication mode because it has better compatibility. Many efforts have been made to improve the RPC performance under RDMA.

Industrial Frameworks. X-RDMA [29] is a communication library widely used in Alibaba data centers to reduce the buffer allocation overhead for small messages. X-RDMA provides two modes to deal with small messages and large messages separately. Small messages directly use RDMA Send. For large messages, the sender will first call RDMA Send to wake up the receiver, who will prepare the buffer as needed. Later, large messages are transferred through one-sided operations. Large message communication requires at least two network round trips, while small message only needs one. This is because small messages are more latency-sensitive in the production environment than large ones. Large messages can tolerate some latency increase from buffer allocation.

Mercury [82] is an RDMA-based asynchronous RPC framework based on Libfabric [102] for high-performance computing applications (including burst buffer file systems like DeltaFS, GekkoFS). Libfabric supports various kinds of network protocols, including TCP/UDP networks, Omni-Path, InfiniBand, iWARP, and RoCE. Whenever possible, Libfabric tries to use the RDMA capability of the underlying network. Similar to X-RDMA, Mercury uses two-sided operations to send small data (e.g., metadata) and uses RDMA Write to transfer bulk data.

Research Prototypes. There are also plenty of research prototypes (Fig. 5) to improve RPC via RDMA as RPC is important and compatible with existing applications. These efforts mainly use the rich RDMA primitives to optimize scalability and performance.

HERD [6] and FaSST [8] are designed for scalability, (partially) using UD mode and two-sided operation in the RPC design. HERD-RPC demonstrates that RDMA Write always has lower latency and higher throughput than RDMA Send. From the server-side, RDMA Write is an in-bound operation with better scalability than out-bound operations [6], [10]. Thus, in the HERD-RPC, clients directly write requests to the per-client buffer in the server. The server polls the buffers to get new requests. After processing the requests, HERD-RPC uses RDMA Send/Recv (UD) to send the responses. UD mode has better scalability because it does not need to manage connections, which is also observed by FaSST. FaSST-RPC uses RDMA Send/Recv under UD mode for requests and responses to reach better scalability. There are mainly three reasons. (1) RC mode has to manage massive connections under all-to-all communication. (2) The probability of packet loss of InfiniBand is extremely low. (3) With several optimizations (e.g., doorbell batching, cheap Recv post), two-sided operation can achieve comparable performance to one-sided operation.

Some works investigate the ways to improve the communication performance under RC mode. RFP [10] proposes a client-centric solution. For the RPC response, the server writes the response in its own local per-client buffer. The RFP client uses RDMA Read to fetch the response. To make the trade-off between high throughput and low latency, RF-RPC [78] uses an online tuning strategy to sample latency and chooses the appropriate size to batch RPC requests.

Additional CPU cycles (polling) are needed to discover new incoming messages under one-sided operations. Some works try to mitigate this problem. Octopus [5] proposes a self-identified RPC (selfRPC) for metadata management by using RDMA Write With Imm to carry the sender's identifier (*node id* of the client and *offset* of the response buffer). This mechanism allows immediate notification to the server since a CQE (completion queue entry) on the server-side will be generated after the completion of RDMA Write With Imm. With this identifier, the server can locate the new request directly without polling and then write back the response based on the sender's identifier. Similar to selfRPC, LITE [79] implements a general RPC with RDMA Write With Imm to provide control information.

*Discussion.*The main drawback of UD-based RPC is that the response packet size is limited by MTU (4KB) (Table 2), and another problem is the connection reliability. According to the RDMA deployment in Alibaba [29], packet loss brings serious problems and can damage the entire communication system. Thus, packet loss should not be ignored even if it rarely happens under UD or UC mode. In contrast, RC-based RPC is reliable but requires more efforts in resource management (e.g., QP). There is plenty of design room for RDMA-based RPC frameworks. With new features introduced in future hardwares, especially with smart NICs, we can hope for more works on RPC.

4.2 Concurrency Control

Many storage systems, even file systems, now require transaction support. Thus, concurrent control in storage systems is unavoidable.

Remote Locks. 2PL (two-phase locking) is a fundamental concurrent control method. DrTM [13] uses one-sided operations to implement a lease-based lock for supporting exclusive writes and shared reads. Exclusive write lock can be supported by using RDMA CAS and shared read lock uses leases. The local transaction is supported by HTM (Hardware Transactional Memory). However, remote RDMA access will

cause the HTM to abort. So in the case of simultaneous local and remote reads, false conflicts may happen, which reduces the overall performance.

Optimistic Concurrency Control. FaRM [14] supports distributed transactions based on its communication primitives using RDMA Write and ring buffers. It also supports remote lock-free read by using RDMA Read. Upon commit, an object is written using local memory access with locking. Servers are not aware of concurrent lock-free reads from clients, so a version number is placed in the cacheline to detect data races. After fetching the result, the client needs to check that the header version is unlocked and matches the versions of all cachelines. Otherwise, retries are needed.

Cell [18] supports both client-side and server-side processing. To synchronize RDMA Read with the server's local memory access, it stores two version numbers in each B-Tree node (head and tail). The head version number gets incremented before updating and the tail version number gets incremented after updating. If RDMA Read finds two versions mismatch, the read request needs to retry. NAM-Tree [20] uses two-sided operations to traverse the index and one-sided operations to read/update leaf nodes. It implements OLC (optimistic lock coupling) [112] to deal with data races. The lock is denoted as <version, lock-bit> in an 8-byte field, which can be modified atomically by both RDMA operations and local instructions.

Discussion. In general, existing approaches to remote concurrent control are close to those in single-machine systems. The difference is that even with RDMA, the network latency is still at least an order of magnitude higher than the latency for accessing local memory. Therefore, lock contention can be more severe in the RDMA network. Thus, one should either reduce the influence of remote locks or resort to other mechanisms without using locks. As a result, many works choose lock-free designs with additional checking mechanisms [9], [24].

4.3 Fault Tolerance

For fault tolerance, while replication is more common, some systems use erasure codes [14], [86], [92].

Replication. As shown in Fig. 6, Mojim [86] uses a single mirror node as a replica and several backups. It combines PM and RDMA to provide low-latency replication. Each request only updates the primary. The primary node will transfer the update log to the mirror node via RDMA Send. Later, the mirror node sends the logs to backups. Replication can be synchronous or asynchronous, satisfying strong or weak consistency.

Tailwind [87] is an RDMA-based log replication protocol that fully uses one-sided operations. It demonstrates that replication can consume about 80% of CPU cycles under write-intensive workloads. Therefore, it uses RDMA Write to replicate logs from the primary to backups, bypassing backup server CPUs. The core challenge is how the backups detect partially applied writes in case of failure. To this end, Tailwind uses metadata, including checksum, offset, and open/close RPC. HyperLoop [74] enables one RNIC to enqueue RDMA operations on other RNICs in the network without CPU involvement. This hardware-enabled chain replication without CPU involvement is more efficient than pure software implementation.

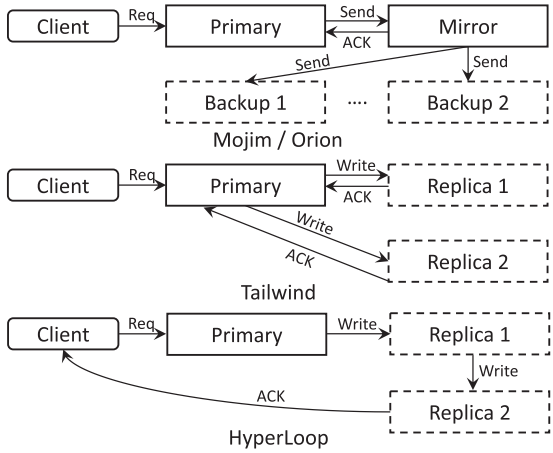


Fig. 6. Comparisons of different RDMA-based replication protocols.

*Erasure Code.*Erasure code (EC) can greatly reduce the amount of disk space required for fault tolerance but with huge computation. Some new RDMA devices can offload the erasure code [113] computation to RNIC. A set of APIs is available to support both synchronous and asynchronous encoding/decoding. INEC [92] proposes a set of in-network EC primitives and implements them on RNIC. It reports that EC offloading can effectively optimize latency, end-to-end throughput, and degraded read performance.

Discussion. Fault tolerance is vital for any storage system. Integrating RDMA can improve the fault-tolerant protocols. Implementing fault tolerance also requires consideration of other requirements (e.g., concurrency control), which are often complicated, but necessary. EC offloading seems a promising approach to guarantee fault tolerance while reducing the storage overhead. Therefore, we should see more EC-offloading systems that also implement other requirements in the future.

4.4 Caching

Caching is an effective way to boost performance and reduce network round trips. The caching strategies used in RDMA-based storage systems can be classified into three types according to cache invalidation strategies.

*Notification-Based Cache.*The simplest way to invalidate cache is through server notification. Cell [18] caches several tree nodes near the root to accelerate tree traversals. DrTM [13] caches a hash table index on the client-side with a similar structure in the remote data store to locate data for reducing the number of RDMA Read. Both systems use server notifications to invalidate the client-side cache.

Validation-Based Cache. Clients can also detect the validity of the cache. RACE [24] designs a client-side cache for directories in the extendible hash to reduce network round trips. Since hash resizing makes the cache stale, an extra checking mechanism is necessary. RACE proposes a scheme where the client can still access the hash table based on the stale cache. A header with some metadata (*local depth* and *suffix bits*) is attached to each bucket to help verify if the correct bucket is accessed. XStore [96] proposes a learning-based cache algorithm to reduce both the number of network round trips and client memory footprints. It can directly obtain the addresses of data using a trained model.

TABLE 4
Comparison of Different Systems

Name	QP Number
Naive RC-based approach	$2\times N \times T$
FaRM [14]	$2\times N \times T/Q$
HERD [6]	$(N+1)\times T$
RFP [10]	$N \times T$
FaSST [8]	T
LITE [79]	$K \times N$
FlatStore [22]	C

(N: node number, K: configurable factor, T: thread number, Q: sharing factor, C: the server-side cores number).

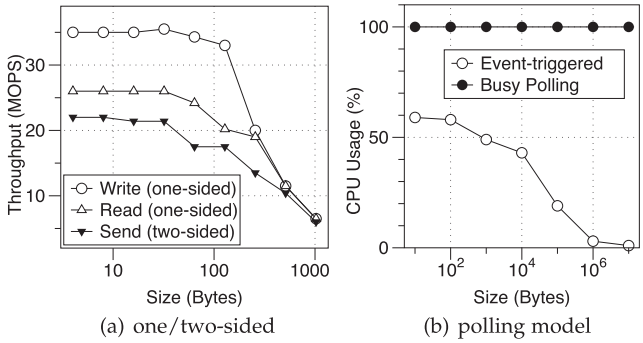


Fig. 7. Performance of different RDMA design choices.

*Lease-Based Cache.*The lease-based cache ensures that the cached data are valid to clients within a certain period of time. C-Hint [12] caches item locations on the client-side. It leverages the lease mechanism to efficiently manage memory reclamation without affecting performance. Due to the unawareness of RDMA Read, the client maintains a global view of the access history and transfers it to servers periodically by using RDMA Write. C-Hint collects the frequencies of recent access and memory utilization on the server for each item and proposes an algorithm to determine the expiration time of each item.

Discussion. Caching the critical parts of indexes can effectively reduce network round trips. It is a challenge to design an efficient cache invalidation scheme according to different storage systems. Using some prediction mechanisms, like machine learning, to optimize caching is helpful.

4.5 Resource Management

Resource management is of great importance while using RDMA devices. As mentioned in Section 2.1, the SRAM size of current commercial RNICs is limited. SRAM needs to cache the MTT and MPT of MRs and the QP data of connections. SRAM overflow will slow down the overall performance.

Memory Region (MR) Management. Some works try to reduce the size of MTT and MPT to save the SRAM space. FaRM [14] proposes a kernel driver, called PhyCo, to allocate contiguous 2GB MRs. Since the size of each MR is 2GB, the number of MRs is effectively reduced, thus reducing the size of MTT and MPT. Similarly, Storm [76] manages RDMA-enabled memory within the same allocator of FaRM.

Other works are related to region permissions. Native RDMA memory registration is inflexible. Each MR can only have one permission for all applications. LITE [79] proposes LMR (LITE Memory Region), an arbitrary size virtual memory indirect region, and it can map to more than one different ranges of physical memory. Thus, it can provide different permissions for different applications.

QP Management. Since QP creation is expensive, many works try to reuse created QPs or reduce the number of QPs. ScaleRPC [17] explores how to mitigate the poor performance problem caused by massive connections under RC mode. It divides the server-side QPs into multiple groups, and only one group can communicate with the clients at a time. X-RDMA [29] manages QPs in the same context (per-thread) as the QP cache. When a QP is reclaimed, X-RDMA resets the QP and pushes it into the QP cache instead of deallocating directly. The QPs in the QP cache can be reused to accelerate connection establishment. LITE [79] uses shared receive CQ, where one thread per node is responsible for busy polling. It chooses an appropriate K to make a trade-off between the total system bandwidth and the total number of QPs. Table 4 shows the number of QPs required in the representative systems on a single server. A non-sharing RC-based implementation uses a large number of QPs. FaRM [14] shares QPs in an application, and RFP [10] shares the same QP for both request and response to reduce the number of QPs. HERD [6] and FaSST [8] need fewer QPs because they are not RC-based. By delegating RDMA operations to agent cores, FlatStore [22] reduces the number of QPs to the number of cores on the server-side. Its delegation phase gathers all RDMA operations to a socket near the RNIC to reduce the MMIO overhead. In order to prevent a CQ from becoming a bottleneck when shared by multiple QPs, DaRPC [77] monitors the load of each CQ and assigns new connections to a CQ with the lowest load.

Discussion. Resource management becomes vital when the system scales up. It brings severe performance degradation once the metadata of MR and QP exceed the SRAM size of RNIC. Current works on MR metadata management require driver modifications, making it difficult for wide adoption. As QP management can be implemented in the user level, it has been adopted by most systems and deployed to datacenters [14], [29].

5 Analysis and Future Work

This section first summarizes a few practical guidelines for RDMA from existing works (Section 5.1). Then we give some high-level discussions between industry and academia (Section 5.2). Finally, we also share our views on the future research directions of RDMA for storage systems (Section 5.3).

5.1 Practical Guidelines

One-Sided versus Two-Sided. Generally, one-sided operation has better performance than two-sided operation [75] (Fig. 7a). With the data size smaller than 128 bytes,Write is 59% faster thanSend and 34% faster thanRead. The reason isWrite maintains fewer states, whileRead has to maintain the states for receiving the response.However, one-sided operation only supports simple operations (e.g., Read, Write, CAS, FAA), resulting in additional network round trips for complex operations. The choice of which mode to use depends on the particular scenario. For read-intensive workloads,

developers can use one-sided operations for read and two-sided for write [9], [12]. For write-intensive workloads, one can either use two-sided operations to reduce the conflicts or one-sided RDMA Write if a server is overloaded [18], [19], [75]. The server processing (two-sided operations) should always be considered for complex operations other than simple read and write. For the disaggregated applications, the storage node can only be accessed through one-sided operations [24], [52]. In addition, smart NICs (Section 3.5) can be used to extend the one-sided semantics to achieve the advantages of both one-sided and two-sided operations.

Connection versus Datagram. Connection-based communication has to maintain QPs. As mentioned in Section 2.1, QPs are cached in on-chip SRAM, and SRAM overflow results in poor scalability and 50%∼70% performance degradation [17], [78], [80].Grouping and reusing QPs can mitigate the scalability issue [14], [29]. However, these optimizations are not sufficient for fully connected scenarios (e.g., distributed transaction processing). In this case, the datagram mode (UD) would be a better choice because it directly eliminates the connection overhead [8].

*Memory Region (MR) Registration (pre- versus on-Demand Registration).*There are two strategies for registering RDMA-enabled memory [14], [29]: pre-registration and on-demand registration. Pre-registration registers MRs before issuing any RDMA request, while on-demand registration registers an MR before each RDMA request.

Existing evaluation [114] shows that the MR registration cost is relatively fixed when the MR size is smaller than 4KB, while it increases linearly when the MR size is larger than 4KB. It is more efficient to reuse a MR with a size smaller than 256KB because memory copy overhead is less than registering a MR.Therefore, we suggest pre-registration for small MRs (e.g., less than 4KB) and on-demand registration for large MRs.

*Polling Mode (Busy Polling versus Event-Triggered).*Busy polling provides low latency with high CPU consumption, while event-triggered polling [115] consumes less CPU but with relatively longer latency. As in Fig. 7b, event-triggered polling has comparable CPU usage (40%∼60%) to busy polling when the message size is small but incurs higher latency. On the contrary, for large-size messages, event-triggered polling leads to significant CPU savings — nearly $50\times$ when the message size is 1M.

Programming Interface (User-Space versus Kernel-Space). Developers can use user-space interface (e.g., Verbs [101], UCX [116], Libfabric [102]) or kernel-space interface (e.g., Verbs, LITE [79]). The kernel-space interface can access physical memory directly, while the user-space interface can only use virtual memory. These two interfaces' performance and behavior are comparable for read and write operations. For memory registration, the user-space interface must prevent the pages from being swapped out, so it has to pin memory, which is expensive. On the contrary, the kernel-space interface can register physical memory directly. LITE [79] provides a more flexible abstraction than kernel-space Verbs. It allows users to transparently access memory on multiple nodes while providing different permissions for different users.

*Optimizations.*RDMA programming has some standard optimizations. (1) Batching can improve the performance. The batching mechanism in RDMA includes scatter/gather list (SGL) and doorbell [99] notification mechanism. SGL packs several scatter/gather elements (SGE) (each SGE represents one memory operation), into a list and issues only one RDMA operation to complete all operations. The doorbell mechanism can use only one notification for a batch of SGLs. (2) NUMA effect should be considered. If CPU, RNIC, and RDMA-enabled memory are not on the same socket, latency/throughput will increase/reduce by up to 55%/49% [105]. (3) RDMA atomic operations can reduce the network round trips. Though atomic operations have to lock memory buses and perform slightly slower than RDMA Read/Write, they are more powerful than simple read and write and scale well. Therefore, they can be used as needed.

*Experiences.*We summarize some key points from existing works. (1) One-sided operations are difficult to use, but they are useful for performance in some scenarios. (2) Hybrid strategies perform better. One-sided and two-sided operations alone are not perfect. (3) Client-side caching requires validation mechanisms to detect data staleness. (4) Lock-free concurrent control is more efficient due to network latency. (5) QP management is critical to scalability. (6) Coupling RDMA with other modules is necessary to get extremely high performance. (7) The interfaces provided by RDMA are low-level. Hence high-level abstractions need to be designed to make RDMA programming easier. (8) Smart NICs can extend the current RDMA interfaces and semantics.

5.2 Academia-Research versus Industry-Research

Memory semantics in RDMA provides more choices instead of just transferring data. Thus, a lot of academic works try to use memory semantics effectively and efficiently. However, we can still see that in industry, as well as the high-performance computing community, most of the current deployments consider RDMA as a faster network (e.g., Microsoft [26], [27], Google [28], Alibaba [29], [30], [31]).

Academia-Research. Researchers want to take full advantage of the memory semantics of RDMA to optimize the system design. Academic RDMA-based storage systems use DRAM and PM as storage media, which are more compatible with RDMA in terms of byte-addressable feature. In order to exploit the potential of high-speed hardware, many efforts use coupling architectures and propose many specialized optimizations [5], [13], [24], [37], [38], [39]. Due to the hardware cost and the need to modify existing source codes, such architectures are not widely used in data centers. But some works are applied in practice that requires high-speed data processing. For example, FaRM [14] as an RDMA-based distributed memory can support Microsoft's graph database A1 [110]. Academic efforts also share some solutions and experiences on various storage system modules such as caching and fault-tolerant protocols [12], [24], [74], [86], [87], [96]. They fully exploit the features of new hardware and even new algorithms to improve the performance of different storage modules.

Industry-Research. The industry works usually use RDMA as a fast network protocol in many cases. Developers have designed several RDMA communication libraries, such as X-RDMA [29], Mercury [82], UCX [116], Libfabric [102]. Such libraries try to provide simple interfaces, support multiple networks (high-speed networks or traditional Ethernet), and

can adapt to many different scenarios and applications. These libraries are usually designed to be compatible with existing communication libraries as much as possible.

However, some works from the industry try to use one-sided operations. For example, PolarDB [59] uses one-sided operations to optimize remote latching and global timestamp. For resource resilience, industry starts to design disaggregated systems [30], [59], [117], which inspires many academia works [24], [52], [53]. Disaggregation is an interesting new architecture combining different kinds of resources using RDMA devices.

5.3 Future Work

With the rapid development of hardware and changes in distributed architectures, there are many opportunities to improve RDMA-based storage systems. Below we have listed some future research directions. Rather than attempting a comprehensive list of research directions, here are some that we think are valuable.

Resource Disaggregation. The disaggregation architecture [118] is proposed to provide flexibility and resilience in organizing hardware resources. Applying the RDMA network to disaggregation architecture allows different types of computing resources can be decoupled to different blades. This is a significant shift in architecture for distributed systems. Thus, many system software, as well as applications, need to be redesigned to satisfy this architecture.

Virtual RDMA Network. One of the core requirements in the cloud environment is to support multi-tenant. Therefore, we can make isolations between tenants and achieve high performance by using the RDMA network. The virtual RDMA network is one way to satisfy this requirement. FreeFlow [119] is the current RDMA virtualization solution for containers. vSocket [120] explores how to provide a software-based RDMA virtualization framework in public clouds, while MasQ [121] proposes a low-cost solution in private clouds.

New RDMA Interfaces. Another essential future work is about the new functionalities for RDMA devices. The current one-sided operation is simple but results in multiple network round trips. On the other hand, two-sided operations do not take full advantage of RDMA features. PRISM [122] argues that an extension to the RDMA interface can resolve this dilemma and proposes four new primitives to increase the expressivity of RDMA. The smart NIC is also a development in this direction.

6 Conclusion

This survey studies the current works on RDMA-based storage systems. We have identified five different kinds of systems related to storage. Next, we point out the core and generic modules that make up these storage systems and analyze their considerations and tradeoffs when using RDMA. The guidelines are summarized in Section 5.1. Finally, we think that although many efforts have been made, RDMA-based storage systems still need more research to adapt to new applications and architectures.

Acknowledgments

Shaonan Ma and Teng Ma equally contributed to this work.

References

[1] Q. Xu *et al.*, "Performance analysis of NVMe SSDs and their implication on real world databases," in *Proc. 8th ACM Int. Syst. Storage Conf.*, 2015, Art. no. 6.

[2] Intel Optane DC Persistent Memory Module, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

[3] S. Ma *et al.*, "ROART: Range-query optimized persistent art," in *Proc. 19th USENIX Conf. File Storage Technol.*, 2021, pp. 1–16.

[4] Connectx-6, 2020. [Online]. Available: https://network.nvidia.com/files/doc-2020/pb-connectx-6-en-card.pdf

[5] Y. Lu, J. Shu, Y. Chen, and T. Li, "Octopus: An RDMA-enabled distributed persistent memory file system," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2017, pp. 773–785.

[6] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 295–306.

[7] V. Gavrielatos, A. Katsarakis, A. Joshi, N. Oswald, B. Grot, and V. Nagarajan, "Scale-out ccNUMA: Exploiting skew with strongly consistent caching," in *Proc. 13th EuroSys Conf.*, 2018, Art. no. 21.

[8] A. Kalia, M. Kaminsky, and D. G. Andersen, "FaSST: Fast, scalable and simple distributed transactions with two-sided RDMA datagram RPCs," in *Proc. 12th USENIX Conf. Oper. Syst. Des. Implementation*, 2016, pp. 185–201.

[9] C. Mitchell, Y. Geng, and J. Li, "Using one-sided RDMA reads to build a fast, CPU-efficient key-value store," in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2013, pp. 103–114.

[10] M. Su, M. Zhang, K. Chen, Z. Guo, and Y. Wu, "RFP: When RPC is faster than server-bypass with RDMA," in *Proc. 12th Eur. Conf. Comput. Syst.*, 2017, pp. 1–15.

[11] Y. Wang *et al.*, "HydraDB: A resilient RDMA-driven key-value middleware for in-memory cluster computing," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2015, pp. 1–11.

[12] Y. Wang, X. Meng, L. Zhang, and J. Tan, "C-Hint: An effective and reliable cache management for RDMA-accelerated key-value stores," in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–13.

[13] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen, "Fast in-memory transaction processing using RDMA and HTM," in *Proc. 25th Symp. Oper. Syst. Princ.*, 2015, pp. 87–104.

[14] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *Proc. 11th USENIX Conf. Netw. Syst. Des. Implementation*, 2014, pp. 401–414.

[15] B. Cassell, T. Szepesi, B. Wong, T. Brecht, J. Ma, and X. Liu, "Nessie: A decoupled, client-driven key-value store using RDMA," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3537–3552, Dec. 2017.

[16] A. Trivedi, P. Stuedi, B. Metzler, C. Lutz, M. Schmatz, and T. R. Gross, "RStore: A direct-access DRAM-based data store," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, 2015, pp. 674–685.

[17] Y. Chen, Y. Lu, and J. Shu, "Scalable RDMA RPC on reliable connection with efficient resource sharing," in *Proc. 14th EuroSys Conf.*, 2019, Art. no. 19.

[18] C. Mitchell, K. Montgomery, L. Nelson, S. Sen, and J. Li, "Balancing CPU and network in the cell distributed B-tree store," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2016, pp. 451–464.

[19] M. Xiao, H. Wang, L. Geng, R. Lee, and X. Zhang, "Catfish: Adaptive RDMA-enabled r-tree for low latency and high throughput," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 164–175.

[20] T. Ziegler, S. T. Vani, C. Binnig, R. Fonseca, and T. Kraska, "Designing distributed tree-based index structures for fast RDMA-capable networks," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 741–758.

[21] K. Dong, L. Huang, and Y. Zhu, "Exploiting RDMA for distributed low-latency key/value store on non-volatile main memory," in *Proc. IEEE 23rd Int. Conf. Parallel Distrib. Syst.*, 2017, pp. 225–231.

[22] Y. Chen, Y. Lu, F. Yang, Q. Wang, Y. Wang, and J. Shu, "FlatStore: An efficient log-structured key-value storage engine for persistent memory," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2020, pp. 1077–1091.

[23] T. Li, D. Shankar, S. Gugnani, and X. Lu, "RDMP-KV: Designing remote direct memory persistence based key-value stores with PMEM," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2020, pp. 1–14.

[24] P. Zuo, J. Sun, L. Yang, S. Zhang, and Y. Hua, "One-sided RDMA-conscious extendible hashing for disaggregated memory," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2021, pp. 15–29.
[25] J. Ousterhout *et al.*, "The RAMCloud storage system," *ACM Trans. Comput. Syst.*, vol. 33, no. 3, pp. 1–55, 2015.
[26] C. Guo *et al.*, "RDMA over commodity ethernet at scale," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 202–215.
[27] Y. Zhu *et al.*, "Congestion control for large-scale RDMA deployments," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 523–536.
[28] A. Singhvi *et al.*, "1RMA: Re-envisioning remote memory access for multi-tenant datacenters," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl. Techno. Archit. Protoc. Comput. Commun.*, 2020, pp. 708–721.
[29] T. Ma *et al.*, "X-RDMA: Effective RDMA middleware in large-scale production environments," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2019, pp. 1–12.
[30] Y. Gao *et al.*, "When cloud storage meets RDMA," in *Proc. 18th USENIX Symp. Netw. Syst. Des. Implementation*, 2021, pp. 519–533.
[31] Y. Li *et al.*, "HPCC: High precision congestion control," in *Proc. ACM Special Int. Group Data Commun.*, 2019, pp. 44–58.
[32] Q. Wang, Y. Lu, and J. Shu, "Sherman: A write-optimized distributed B+ tree index on disaggregated memory," in *Proc. Int. Conf. Manage. Data*, 2022, pp. 1033–1048.
[33] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Oper. Syst. Des. Implementation*, 2006, pp. 307–320.
[34] Glusterfs on RDMA, 2022. [Online]. Available: https://gluster.readthedocs.io/en/latest/AdministratorGuide/RDMATransport/
[35] P. Stuedi *et al.*, "Crail: A high-performance I/O architecture for distributed data processing," *IEEE Data Eng. Bull.*, vol. 40, no. 1, pp. 38–49, Jun. 2017.
[36] N. S. Islam, M. Wasi-ur Rahman, X. Lu, and D. K. Panda, "High performance design for HDFS with byte-addressability of NVM and RDMA," in *Proc. ACM Int. Conf. Supercomput.*, 2016, Art. no. 8.
[37] J. Yang, J. Izraelevitz, and S. Swanson, "Orion: A distributed file system for non-volatile main memory and RDMA-capable networks," in *Proc. 17th USENIX Conf. File Storage Technol.*, 2019, pp. 221–234.
[38] J. Yang, J. Izraelevitz, and S. Swanson, "FileMR: Rethinking RDMA networking for scalable persistent memory," in *Proc. 17th USENIX Conf. Netw. Syst. Des. Implementation*, 2020, pp. 111–126.
[39] T. E. Anderson *et al.*, "Assise: Performance and availability via client-local NVM in a distributed file system," in *Proc. 14th USENIX Conf. Oper. Syst. Des. Implementation*, 2020, Art. no. 57.
[40] Q. Zheng, K. Ren, G. Gibson, B. W. Settlemyer, and G. Grider, "DeltaFS: Exascale file systems scale better without dedicated servers," in *Proc. 10th Parallel Data Storage Workshop*, 2015, pp. 1–6.
[41] M.-A. Vef *et al.*, "GekkoFS-A temporary distributed file system for HPC applications," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2018, pp. 319–324.
[42] M. S. Breitenfeld *et al.*, "DAOS for extreme-scale systems in scientific applications," 2017, *arXiv:1712.00423*.
[43] W. Cao *et al.*, "PolarFS: An ultra-low latency and failure resilient distributed file system for shared storage cloud database," *Proc. VLDB Endowment*, vol. 11, no. 12, pp. 1849–1862, 2018.
[44] P. Braam, "The lustre storage architecture," 2019, *arXiv:1903.01955*.
[45] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proc. 1st USENIX Conf. File Storage Technol.*, 2002, pp. 19–es.
[46] F. Herold, S. Breuner, and J. Heichler, "An introduction to BeeGFS," ThinkParQ, Kaiserslautern, Germany, Tech. Rep., 2014. [Online]. Available: https://www. beegfs
[47] W. Yu, S. Liang, and D. K. Panda, "High performance support of parallel virtual file system (PVFS2) over quadrics," in *Proc. 19th Annu. Int. Conf. Supercomput.*, 2005, pp. 323–331.
[48] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot, "The case for RackOut: Scalable data serving using rack-scale systems," in *Proc. 7th ACM Symp. Cloud Comput.*, 2016, pp. 182–195.
[49] J. Nelson *et al.*, "Latency-tolerant software distributed shared memory," in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2015, pp. 291–305.
[50] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with INFINISWAP," in *Proc. 14th USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 649–667.
[51] Y. Shan, S.-Y. Tsai, and Y. Zhang, "Distributed shared persistent memory," in *Proc. ACM Symp. Cloud Comput.*, 2017, pp. 323–337.
[52] S.-Y. Tsai, Y. Shan, and Y. Zhang, "Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated key-value stores," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2020.
[53] T. Ma, M. Zhang, K. Chen, Z. Song, Y. Wu, and X. Qian, "AsymNVM: An efficient framework for implementing persistent data structures on asymmetric NVM architecture," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, pp. 757–773.
[54] I. Calciu *et al.*, "Rethinking software runtimes for disaggregated memory," in *Proc. 26th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2021, pp. 79–92.
[55] K. Taranov, S. Di Girolamo, and T. Hoefler, "CoRM: Compactable remote memory over RDMA," in *Proc. Int. Conf. Manage. Data*, 2021, pp. 1811–1824.
[56] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian, "The end of slow networks: It's time for a redesign," *Proc. VLDB Endowment*, vol. 9, no. 7, pp. 528–539, 2016.
[57] E. Zamanian, C. Binnig, T. Harris, and T. Kraska, "The end of a myth: Distributed transactions can scale," *Proc. VLDB Endowment*, vol. 10, no. 6, pp. 685–696, 2017.
[58] E. Zamanian, J. Shun, C. Binnig, and T. Kraska, "Chiller: Contention-centric transaction execution and data partitioning for modern networks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 511–526.
[59] W. Cao *et al.*, "PolarDB serverless: A cloud native database for disaggregated data centers," in *Proc. Int. Conf. Manage. Data*, 2021, pp. 2477–2489.
[60] A. Ryser, A. Lerner, A. Forencich, and P. Cudré-Mauroux, "D-RDMA: Bringing zero-copy RDMA to database systems,".
[61] E. Zamanian, X. Yu, M. Stonebraker, and T. Kraska, "Rethinking database high availability with RDMA networks," *Proc. VLDB Endowment*, vol. 12, pp. 1637–1650, 2019.
[62] F. Li, S. Das, M. Syamala, and V. R. Narasayya, "Accelerating relational databases by leveraging remote memory and RDMA," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 355–370.
[63] W. Rödiger, T. Mühlbauer, A. Kemper, and T. Neumann, "High-speed query processing over high-speed networks," 2015, *arXiv:1502.07169*.
[64] C. Barthels, I. Müller, T. Schneider, G. Alonso, and T. Hoefler, "Distributed join algorithms on thousands of cores," *Proc. VLDB Endowment*, vol. 10, pp. 517–528, 2017.
[65] A. Salama, C. Binnig, T. Kraska, A. Scherp, and T. Ziegler, "Rethinking distributed query execution on high-speed networks," *IEEE Data Eng. Bull.*, vol. 40, no. 1, pp. 27–37, 2017.
[66] P. Fent, A. van Renen, A. Kipf, V. Leis, T. Neumann, and A. Kemper, "Low-latency communication for fast DBMS using RDMA and shared memory," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1477–1488.
[67] F. Liu, L. Yin, and S. Blanas, "Design and evaluation of an RDMA-aware data shuffling operator for parallel database systems," *ACM Trans. Database Syst.*, vol. 44, 2019, Art. no. 1.
[68] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy, "High performance packet processing with FlexNIC," in *Proc. 21st Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2016, pp. 67–81.
[69] B. Li *et al.*, "KV-direct: High-performance in-memory key-value store with programmable NIC," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 137–152.
[70] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A SmartNIC-driven accelerator-centric architecture for network servers," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, pp. 117–131.
[71] D. Sidler, Z. Wang, M. Chiosa, A. Kulkarni, and G. Alonso, "StRoM: Smart remote memory," in *Proc. 15th Eur. Conf. Comput. Syst.*, 2020, Art. no. 29.
[72] J. Kim *et al.*, "LineFS: Efficient SmartNIC offload of a distributed file system with pipeline parallelism," in *Proc. 28th Symp. Oper. Syst. Princ.*, 2021, pp. 756–771.
[73] H. N. Schuh, W. Liang, M. Liu, J. Nelson, and A. Krishnamurthy, "Xenic: SmartNIC-accelerated distributed transactions," in *Proc. 28th Symp. Oper. Syst. Princ.*, 2021, pp. 740–755.

[74] D. Kim *et al.*, "Hyperloop: Group-based NIC-offloading to accelerate replicated transactions in multi-tenant storage systems," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2018, pp. 297–312.
[75] X. Wei, Z. Dong, R. Chen, and H. Chen, "Deconstructing RDMA-enabled distributed transactions: Hybrid is better!," in *Proc. 13th USENIX Conf. Oper. Syst. Des. Implementation*, 2018, pp. 233–251.
[76] S. Novakovic *et al.*, "Storm: A fast transactional dataplane for remote data structures," in *Proc. 12th ACM Int. Conf. Syst. Storage*, 2019, pp. 97–108.
[77] P. Stuedi, A. Trivedi, B. Metzler, and J. Pfefferle, "DaRPC: Data center RPC," in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–13.
[78] Y. Wu, T. Ma, M. Su, M. Zhang, C. Kang, and Z. Guo, "RF-RPC: Remote fetching RPC paradigm for RDMA-enabled network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1657–1671, Jul. 2019.
[79] S.-Y. Tsai and Y. Zhang, "LITE kernel RDMA support for datacenter applications," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 306–324.
[80] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *Proc. 16th USENIX Conf. Netw. Syst. Des. Implementation*, 2019, pp. 1–16.
[81] Accelio, 2013. [Online]. Available: http://www.accelio.org
[82] J. Soumagne *et al.*, "Mercury: Enabling remote procedure call for high-performance computing," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2013, pp. 1–8.
[83] S. K. Monga, S. Kashyap, and C. Min, "Birds of a feather flock together: Scaling RDMA RPCs with flock," in *Proc. 28th Symp. Oper. Syst. Princ.*, 2021, pp. 212–227.
[84] L. Thostrup, J. Skrzypczak, M. Jasny, T. Ziegler, and C. Binnig, "DFI: The data flow interface for high-speed networks," in *Proc. Int. Conf. Manage. Data*, 2021, pp. 1825–1837.
[85] T. Li, H. Shi, and X. Lu, "HatRPC: Hint-accelerated thrift RPC over RDMA," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2021, Art. no. 36.
[86] Y. Zhang, J. Yang, A. Memaripour, and S. Swanson, "Mojim: A reliable and highly-available non-volatile memory system," in *Proc. 20th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2015, pp. 3–18.
[87] Y. Taleb, R. Stutsman, G. Antoniu, and T. Cortes, "Tailwind: Fast and atomic RDMA-based replication," in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2018, pp. 851–863.
[88] M. Poke and T. Hoefler, "DARE: High-performance state machine replication on RDMA networks," in *Proc. 24th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2015, pp. 107–118.
[89] C. Wang, J. Jiang, X. Chen, N. Yi, and H. Cui, "APUS: Fast and scalable paxos on RDMA," in *Proc. ACM Symp. Cloud Comput.*, 2017, pp. 94–107.
[90] S. Jha *et al.*, "Derecho: Fast state machine replication for cloud services," *ACM Trans. Comput. Syst.*, vol. 36, pp. 1–49, 2019.
[91] V. Gavrielatos, A. Katsarakis, and V. Nagarajan, "Odyssey: The impact of modern hardware on strongly-consistent replication protocols," in *Proc. 16th Eur. Conf. Comput. Syst.*, 2021, pp. 245–260.
[92] H. Shi and X. Lu, "INEC: Fast and coherent in-network erasure coding," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2020, pp. 1–17.
[93] M. K. Aguilera, N. Ben-David, R. Guerraoui, V. Marathe, and I. Zablotchi, "The impact of RDMA on agreement," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2019, pp. 409–418.
[94] Q. Cai *et al.*, "Efficient distributed memory management with RDMA and caching," *Proc. VLDB Endowment*, vol. 11, pp. 1604–1617, 2018.
[95] M. K. Aguilera, K. Keeton, S. Novakovic, and S. Singhal, "Designing far memory data structures: Think outside the box," in *Proc. Workshop Hot Topics Oper. Syst.*, 2019, pp. 120–126.
[96] X. Wei, R. Chen, and H. Chen, "Fast RDMA-based ordered key-value store using remote learned cache," in *Proc. 14th USENIX Conf. Oper. Syst. Des. Implementation*, 2020, Art. no. 7.
[97] S. Kumar *et al.*, "PAMI: A parallel active message interface for the blue gene/Q supercomputer," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp.*, 2012, pp. 763–773.
[98] X. Wei, X. Xie, R. Chen, H. Chen, and B. Zang, "Characterizing and optimizing remote persistent memory with RDMA and NVM," in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2021, pp. 523–536.
[99] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2016, pp. 437–450.
[100] Z. Guo, S. Liu, and Z.-L. Zhang, "Traffic control for RDMA-enabled data center networks: A survey," *IEEE Syst. J.*, vol. 14, no. 1, pp. 677–688, Mar. 2020.
[101] D. Barak, "RDMA aware networks programming user manual," 2015.
[102] OpenFabrics, 2019. [Online]. Available: https://ofiwg.github.io/libfabric/
[103] L. Shalev, H. Ayoub, N. Bshara, and E. Sabbag, "A cloud-optimized transport protocol for elastic and scalable HPC," *IEEE Micro*, vol. 40, no. 6, pp. 67–73, Nov./Dec. 2020.
[104] Mellanox, "Dynamically-connected transport," 2018. [Online]. Available: https://openfabrics.org/images/2018workshop/presentations/303_ARosenbaum_DynamicallyConnectedTransport.pdf
[105] T. Ma, K. Chen, S. Ma, Z. Song, and Y. Wu, "Thinking more about RDMA memory semantics," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2021, pp. 456–467.
[106] A. Kalia, D. Andersen, and M. Kaminsky, "Challenges and solutions for fast remote persistent memory access," in *Proc. ACM Symp. Cloud Comput.*, 2020, pp. 105–119.
[107] Lustre, "Lnet," 2022 [Online]. Available: https://github.com/DDNStorage/lustre_manual_markdown
[108] CaRT, 2022. [Online]. Available: https://github.com/daos-stack/cart
[109] A. Rudoff, "Persistent memory programming," *Login: Usenix Mag.*, vol. 42, pp. 34–40, 2017.
[110] C. Buragohain *et al.*, "A1: A distributed in-memory graph database," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 329–344.
[111] R. Mistry and S. Misner, *Introducing Microsoft SQL Server 2014*. Redmond, WA, USA: Microsoft Press, 2014.
[112] V. Leis, M. Haubenschild, and T. Neumann, "Optimistic lock coupling: A scalable and efficient general-purpose synchronization method," *IEEE Data Eng. Bull.*, vol. 42, no. 1, pp. 73–84, 2019.
[113] OpenFabric, "OFVWG: Erasure coding RDMA offload," 2017. [Online]. Available: https://downloads.openfabrics.org/ofv/erasurecoding.pdf
[114] P. W. Frey and G. Alonso, "Minimizing the hidden cost of RDMA," in *Proc. IEEE 29th Int. Conf. Distrib. Comput. Syst.*, 2009, pp. 553–560.
[115] Y. Zhang, J. Gu, Y. Lee, M. Chowdhury, and K. G. Shin, "Performance isolation anomalies in RDMA," in *Proc. Workshop Kernel-Bypass Netw.*, 2017, pp. 43–48.
[116] N. Papadopoulou, L. Oden, and P. Balaji, "A performance study of UCX over InfiniBand," in *Proc. IEEE/ACM 17th Int. Symp. Cluster Cloud Grid Comput.*, 2017, pp. 345–354.
[117] Q. Zhang *et al.*, "Understanding the effect of data center resource disaggregation on production DBMSs," *Proc. VLDB Endowment*, vol. 13, pp. 1568–1581, 2020.
[118] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 267–278.
[119] D. Kim *et al.*, "FreeFlow: Software-based virtual RDMA networking for containerized clouds," in *Proc. 16th USENIX Symp. Netw. Syst. Des. Implementation*, 2019, pp. 113–126.
[120] D. Wang, B. Fu, G. Lu, K. Tan, and B. Hua, "vSocket: Virtual socket interface for RDMA in public clouds," in *Proc. 15th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, 2019, pp. 179–192.
[121] Z. He *et al.*, "MasQ: RDMA for virtual private cloud," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl. Technol. Archit. Protoc. Comput. Commun.*, 2020, pp. 1–14.
[122] M. Burke, S. Joyner, A. Szekeres, J. Nelson, I. Zhang, and D. R. Ports, "PRISM: Rethinking the RDMA interface for distributed systems," in *Proc. 28th Symp. Oper. Syst. Princ.*, 2021, pp. 228–242.

Shaonan Ma received the BE degree from the Dalian University of Technology, China, in 2018, and the PhD degree from the Department of Computer Science and Technology, Tsinghua University, China. His research interests include persistent memory, distributed storage system, and RDMA.

Teng Ma received the BE degree from the University of Science and Technology Beijing, China, in 2016, and the PhD degree in computer science and technology from Tsinghua University, China, in 2021. Now, he is a posdoctoral with Alibaba Group. His research interests include distributed database, remote direct memory access (RDMA), and key-value store systems.

Kang Chen received the PhD degree in computer science and technology from Tsinghua University, Beijing, China, in 2004. Currently, he is a professor of computer science and technology with Tsinghua University. His research interests include parallel computing, distributed processing, and cloud computing.

Yongwei Wu (Senior Member, IEEE) received the PhD degree in applied mathematics from the Chinese Academy of Sciences, in 2002. He is currently a professor in computer science and technology with the Tsinghua University of China. His research interests include parallel and distributed processing, and cloud storage. He has published more than 80 research publications and has received two best paper awards.