Analyzing In-Memory NoSQL Landscape

Analyzing In-Memory NoSQL Landscape

Masoud Hemmatpour†, Bartolomeo Montrucchio†,
Maurizio Rebaudengo†, and Mohammad Sadoghi ‡
†Dipartimento di Automatica e Informatica, Politecnico di Torino
‡Computer Science Department, University of California Davis

Abstract—In-memory key-value stores have quickly become a key enabling technology to build high-performance applications that must cope with massively distributed workloads. In-memory key-value stores (also referred to as NoSQL) primarly aim to offer low-latency and high-throughput data access which motivates the rapid adoption of modern network cards such as Remote Direct Memory Access (RDMA). In this paper, we present the fundamental design principles for exploiting RDMAs in modern NoSQL systems. Moreover, we describe a break-down analysis of the state-of-the-art of the RDMA-based in-memory NoSQL systems regarding the indexing, data consistency, and the communication protocol. In addition, we compare traditional in-memory NoSQL with their RDMA-enabled counterparts. Finally, we present a comprehensive analysis and evaluation of the existing systems based on a wide range of configurations such as the number of clients, real-world request distributions, and workload read-write ratios.

Index Terms—RDMA, memory, key-value store, big data, high performance, cluster, parallel programming.

1 Introduction

As demand for big data analytics grows every day, companies have become aware of the critical role of real-time data-driven decision making to gain a competitive edge. This creates a challenge for companies needing to accelerate (fine-grained) the access to massively distributed data, in particular, those dealing with online services [1]. A *distributed key-value* store offers a flexible data model that is more performant at the cost of weaker consistency models for partitioning data across many compute nodes. Starting in mid-2000, numerous commercial key-value stores have emerged [2], each with its own unique characteristics, such as Google Bigtable [3], Amazon Dynamo [4], and Facebook Cassandra [5] to enable managing massively distributed data at unseen scale, which simply was not feasible with traditional relational database systems running on commodity hardware. These systems have become critical for large-scale applications, such as social networks [4], [6], realtime processing [7], and recommendation engines [8], [9], [10] to achieve higher performance.

Given the rise of key-value store—broadly classified as NoSQL—over the last decade there have been two major efforts to accelerate NoSQL platform using modern hardware [2]. The first approach was to employ storage-class memory, e.g., such as solid-state disks (SSDs), that focused on exploiting SSDs as a cache between main memory and disks [11], such as CaSSanDra [12], cassandraSSD [13], Flashstore [14], Flashcache [15], BufferHash [16], [17]. The second approach has been to capitalize on the ever-increasing size of the main memory in each machine. These machines can now be connected through fast optical interfaces from a massive virtual shared memory space at an affordable cost that continues to decline, such as RAMCloud [18], Memcached [19], MICA [20], SILT [21] and Redis [22].

Much research has been carried out in order to improve the communication performance either by optimizing the existing protocol [23] or inventing new communication standards. A great deal of work on high-performance communication, such as Arsenic Gigabit Ethernet [24], U-Net [25], VIA [26], (Myricom/CSPi)'s Myrinet [27], Quadrics's QSNET [28] has led to modern high-speed networks including InfiniBand [29], RoCE [29], iWARP [30], and Intel's Omni-Path [31], which support Remote Direct Memory Access (RDMA) [32]. RDMA blurs the boundary of each machine by creating a virtual distributed, shared memory among connected nodes, i.e., substantially reducing communication and processing on the host machine. Through RDMAs, clients can now directly access remote memory without the need to invoke the NoSQL's traditional client-server model. This motivates the NoSQL community to invest in developing purely in-memory key-value stores with RDMA capability, such as HydraDB [33], Herd [34], Pilaf [35], DrTM [36], FaRM [37]. RDMA capable protocol (i.e., InfiniBand) supports legacy socket applications through *IP over InfiniBand* (IPOIB); however, running existing in-memory systems on top of it can not efficiently exploit the benefits in the infrastructure [33], [35]. So existing in-memory key-value stores strive to reduce latency and achieve higher performance by exploiting RDMA operations [38], [39].

Recently, we have witnessed the emergence of many NoSQL systems, such as mongoDB [40], HBase [41], VoltDB [42], Cassandra [5], Voldemort [42], Redis [22], Memcached [19], Pilaf [35], HERD [34], HydraDB [33], FaRM [37], DrTM [36], and Nessie [43]. Therefore, there have been several efforts to evaluate NoSQL systems [44], [45], [46], [47], [48]. Some of these studies performed experimental analysis and some of them adopt case studies in their evaluation.

†*Dipartimento di Automatica e Informatica, Politecnico di Torino E-mail: firstname.lastname@polito.it*
‡*Computer Science Department, University of California Davis Email:msadoghi@ucdavis.edu.*

A. Gandini et al. evaluates commonly used NoSQL databases: Cassandra [5], MongoDB [40] and HBase [41] [44]. T. Rabl et al. presents a comprehensive performance evaluation of six key-value stores: Cassandra [5], Voldemort [42], HBase [41], VoltDB [42], Redis [22], and MySQL [49] focusing on maximum throughput [50]. J. Klein et al. evaluates the performance of Riak [51], Cassandra [5] and MongoDB [40] in a case study for an Electronic Health Record (EHR) system for delivering healthcare support to the patients [45]. P. cudr-Mauroux et al. compares NoSQL systems for the sake of data exchange on the web according to Resource Description Framework (RDF) standard [47]. J. R. Lourenc et al. evaluates quality attribute requirements of NoSQL systems regarding to their suitability on the design of enterprise systems [48]. H. Zhang focuses on the design principles and implementing of efficient and high-performance in-memory data management and processing systems [46]. W. Cao evaluates Redis and Memcached [52]. Their evaluation focused on performance challenges in internal data structures and memory allocators. B. atikoglu et al. analyzes the workloads of Facebooks Memcached according to the size, rate, patterns, and use cases [53].

To the best of our knowledge there is no comprehensive study of widely deployed in-memory key-value stores with RDMA acceleration. We outline a set of principles for exploiting RDMA in key-value stores along with in-depth analysis and lessons learned. Our work corroborates previous findings on RDMA performance issues and highlights the importance of RDMA operations choices on the overall performance [34], [37], [54]. Moreover, we compared the RDMA systems with legacy ones, namely, those that do not rely on any hardware acceleration in their design such as storage class memory or advanced network cards. Moreover, the following parameters are considered when selecting legacy systems: language (e.g., C), general purpose systems, relying on standard communication protocol (i.e., TCP), single-node setup to analyze the throughput, and support of set/get operations. In our study, we describe Memcached [19], a system that can be deployed for caching heavily used objects, as well as Redis, a system that can be used as a database cache and message broker [22] as legacy systems. We chose Memcached and Redis since these two systems have been extensively studied in the literature [52], [55] and commonplace in industry. For example, Facebook uses Memcached [6] and Cisco uses Redis Enterprise as its primary database for IoT solutions [56].

In this paper, we present one-of-a-kind comprehensive study of modern NoSQL systems including HydraDB [33], Pilaf [35], HERD [34], FaRM [37], DrTM [36], Memcached [19], and Redis [22]. We review the key performance challenges of how to exploit RDMA in NoSQL systems in Section 2. We provide an in-depth review of modern key-value stores in an unified representation that reveals architectural differences along with strength and weakness of each system in Section 3. We present a comprehensive evaluation methodology and extensive analysis of state-of-the-art approaches in Section 4. [1]. Finally, we draw conclusions in Section 5.

1. Our implementation is open-sourced and publicly available at (https://github.com/mashemat/local-key-value)

2 RDMA Performance Challenge Analysis

InfiniBand architecture specification describes the functions, called *verbs*, to configure, manage, and operate with InfiniBand adapter [57]. This paper focuses on InfiniBand, which is a commonly used protocol in commodity servers [58]. InfiniBand is an advanced network protocol with low latency and high bandwidth, as well as with advanced features such as RDMA, atomic operations, multicast, QoS, and congestion control. InfiniBand Network Interface Cards (NICs) follow two approaches in packet processing: Onload or Offload (e.g., Mellanox offloading and Qlogic onloading) [59]. In the Onload approach, packets are processed in the host CPU while in the Offload approach packets are processed in NIC processor.

RDMA supports two memory semantics consisting of `SEND/RECV` (two-sided verb) and `READ/WRITE` (one-sided verb). `READ` and `WRITE` semantic reads/writes a data from/to a remote node exploiting the DMA engine of the remote machine (i.e., bypassing the CPU and kernel). `SEND` and `RECV` semantic sends/receives a message through the CPU of the remote node. In order to adopt RDMA semantics as efficiently as possible, multiple programming models have emerged, such as Remote Memory Access (RMA) and Partitioned Global Address Space (PGAS).

The asynchronous nature of InfiniBand architecture allows an application to queue up a series of requests to be executed by the adapter. These queues are created in pairs, called Queue Pair (QP), for send and receive operations. The verb application submits a Work Queue Element (WQE) on the appropriate queue. The channel adapter executes WQEs in the FIFO order on the queue. When the channel adapter completes a WQE, a Completion Queue Element (CQE) is enqueued on a Completion Queue (CQ).

Many studies demonstrate that adopting one-sided verb outperforms two-sided verb by eliminating the overhead of the CPU access in the remote node [33], [35], [37]; however, there are different opinions on the best strategy [34], [60]. Since one-sided verb (i.e., `READ`) requires two round-trip times (RTTs) for remote memory access, [34] advocates using two-sided verb combining with a local memory access in the remote machine. Moreover, it should be considered that the latency of two verbs (i.e., `READ/WRITE` and `SEND/RECV`) are changing through evolving adapters [61]. So measuring semantic costs is necessary before designing a system.

In high-performance applications not only board-to-board communication is critical, but also core-to-core, CPU-to-CPU, and I/O communications require careful investigation to explore the performance tradeoffs. For example, hardware message passing among cores [62], [63], [64], CPU-to-CPU Intel QuickPath Interconnect (QPI) and AMD HyperTransport (HT) [65], [66], and I/O PCI express, Intel Data Direct I/O [67], [68] all strive to enhance the communication performance.

Fig. 1 shows the schematic view of a node equipped with InfiniBand card [57], [69], [70]. It shows the architecture of request queues in an InfiniBand channel adapter and the components of the system. Although the network architecture in the cluster affects the contention for resources, its impacts are outside the scope of this paper.

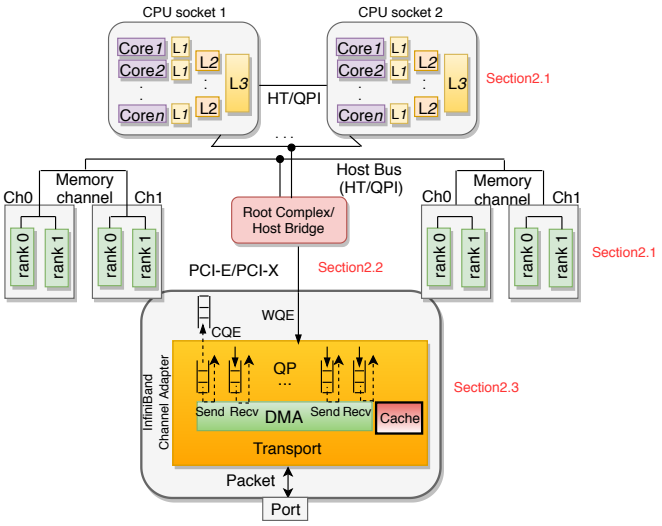

Fig. 1. System view of a compute node with InfiniBand adapter.

2.1 Memory

Although remote memory access through RDMA operation is quite fast compared to traditional network operations, they are still substantially slower than a local memory access [37]. Thus, a better management of the local memory can influence the performance. The main RDMA challenges are described in the following.

Cache miss The experiment on cache miss rate of the requester once the RDMA message uses different memory addresses (i.e., the cache is never hit), and once the message uses the same memory address (i.e., cache is always hit) shows that high cache-miss rate of the requester can reduce the performance [54].

Data Alignment Since NICs work more efficiently on aligned data [71], using aligned message size can improve the performance of RDMA systems [54].

NUMA Affinity The distance between processor and data has a critical role on the performance. Generally speaking, better latency can be achieved through confining the memory access to local NUMA node. However, the appropriate deployment of processes and data can exploit memory bandwidth of other NUMA nodes [72], [73], [74]. Since the operating systems delegate the burden of NUMA-related issues on the application, designer must be aware of the data distribution on the main memory in order to reduce the latency and to increase bandwidth in memory access. The impact of the NUMA affinity on the RDMA applications has been studied in the literature [54].

Memory Prefetching Software prefetching is a classical technique for overcoming the speed gap between the processor and the memory [75], [76]. Software prefetching in RDMA-based applications can improve the performance [34]. Modern CPUs equipped with automated prefetching (e.g., Smart Prefetch) predict and preload the potentially needed data [77].

2.2 Host Bus communication

PCI Express (PCIe) technology [78] is an ubiquitous scalable, high-speed and serialized protocol standard for device communication and mainly a replacement for the PCI-X bus. Both PCIe and PCI-X allowed the device to initiate an independent communication, called first-party DMA [79]. InfiniBand vendors nowadays adopt PCIe bus family for host communication due to the high-speed serial and dedicated link [80], [81], [82]. PCIe generations are evolved based on the speed of the link (i.e., lane speed and number of lanes), encoding, traffic, and packet overheads [83]. PCIe provides a root-tree based network topology, where all I/Os are connected, through switches and bridges, to a root complex. The root complex connects one or more processors and their associated memory subsystems.

Any movement through PCIe has an overhead on the performance, so it is important to understand the CPU and InfiniBand NIC interaction for high-performance applications. Aside from protocol and traffic overhead, *maximum payload size* and *maximum read request size* [83] may impact the performance in a PCIe system. These parameters might cause a limitation on transaction rate over PCIe. Though tuning these parameters have an impact on the performance of InfiniBand devices [84]. Moreover, interrupt request affinity on PCIe can improve application scalability and latency [85].

Profiling PCIe transactions are important to have a comprehensive view of the CPU-NIC interaction [86], [87]. Modern CPUs provide a list of events in Performance Monitoring Unit (PMU) to measure micro-architectural events of PCIe. For example, the events *PCIeRdCur* and *PCIeItoM* monitor DMA reads and writes from PCIe, respectively.

CPU can submit a work request to the NIC out of writing to the memory mapped I/O (MMIO) register (i.e., BlueFlame in Mellanox) or sending a list of works (i.e., Doorbell). It is recommended to use BlueFlame in the light load and Doorbell in high-bandwidth scenarios [88]. Each PCIe device equipped with a DMA engine can access to main memory independently. Firstly, the device sends a memory Read Request to the root complex. Then, it returns the desired memory by completion with a data packet. Comparing the CPU MMIO overhead with NIC DMA memory access reveals that the best trade-off is obtained by reducing the number of MMIOs [86].

The cache coherency between the NIC and CPU is a critical issue which is not written in the specification of InfiniBand protocol and is fully vendor specific. [37] reveals that there is a single cache line coherency for Mellanox adapters on *x86* processors. In addition, memory order on `READ/WRITE` verbs over PCIe are important concerns which are vendor specific.

2.3 NIC Memory

First InfiniBand products (developed by Mellanox) provided memory on the NIC board. However, recently essential resources have been moved to the host memory and only a cache memory remains on the board since the memory access time of the host does not have a significant impact on the performance [89].

The NIC cache memory, particularly in Mellanox adapters, are served for several purposes such as maintaining page tables of registered memory (to translate virtual to physical address) or queue pair (QP) data (i.e., state

and elements) [86]. Since adapter has limited resources, the optimization of this scope can improve the performance. Adopting larger memory pages will reduce the number of entries of page tables, reducing fetching page table entries from system memory to NIC (i.e., page faults) [37], [90]. Reducing the number of queue pairs can reduce the memory usage [37]. In addition, the work request submission rate is quite important for avoiding cache miss in the NIC [86].

2.4 RDMA Features

Choosing the right RDMA features is critical to the scalability and the reliability of the application. In this section, several RDMA features and their impact on the performance are described in more detail.

Transport Type RDMA supports unreliable and reliable types of connections. Reliable connection guarantees the delivery and the order of the packet by the acknowledgment from the responder. An unreliable connection does not guarantee the delivery and the order of the message.

RDMA provides two types of transports: unconnected and connected. Each QP is connected to exactly one QP of a remote node in connected mode unlike unconnected (datagram). Since for each connected connection between two nodes, two QPs are required one for the requester and one for the responder, the number of QPs increases $2\times$ with the number of connections. There are different approaches to reduce the number of QPs. In a reliable connection, threads can share QPs to reduce the QP memory footprint [37]. Sharing QP reduces CPU efficiency because threads contend for the cache lines for QP buffers between CPU cores [91], [92]. In the Annex A14 of the InfiniBand specification 1.2, eXtended Reliable Connection (XRC) was introduced to connect nodes [93]. Multiple connections of a process in a node can be reduced to one.

Shared receive queue (SRQ) shares receive queue on multiple connections and reduce the number of QPs. SRQ solves the two-sided communication synchronization problem between the requester and the responder which previously was solved by using backoff in the requester and over provisioning of receive WQEs in responder [57]. SRQ can solve this problem since an incoming receive message on any QP associated with an SRQ can use the next available WQE to receive the incoming data.

Inline Data Inlining the data to the work queue element (WQE) eliminates the overhead of memory access through DMA for payload after submitting a WQE and expecting the performance raising. However, an inline message has limitation according to the size of the payload.

Message Size The size of the message could be the bottlenecked in two places: host bus communication (i.e., PCIe) or the Path Maximum Transfer Unit (PMTU). *Maximum payload size* and *maximum read request size* in the PCIe communication affects the performance of memory access from the InfiniBand adapter [83], [84]. It basically specifies the number of essential completion with data packets. The higher read-request size increases the efficiency of packet transfers. When a QP (reliable/unreliable connected) is created, the PMTU is determined in the queue and if the desired message to be sent is larger than the PMTU of the queue, the message is divided into multiple messages. However, if InfiniBand receives a message larger than its port Maximum Transfer Unit (MTU) it silently drops the message [94].

Reducing the number of cache lines used by a WQE can improve throughput drastically [86]. Roughly speaking, increasing the size of the message increases the communication latency. [86] demonstrates that increasing the size of the message will decrease the performance.

Completion Detection While InfiniBand adapter completes a work request, it enqueues a CQE in the completion queue. Mainly, two approaches can be adopted to detect completion of a work request: busy polling and event handling. In the former mechanism, the application polls the completion queue to receive a CQE. This approach has high CPU utilization; however, the cost of polling is quite low since the operating system is bypassed. In the second approach, a notification is received when a CQE arrives to the completion queue. This approach is much better based on CPU utilization. However, it requires the operating system intervention. Busy polling outperforms event handling in all possible RDMA operations [54], [95]. However, in large message size, both methods converge.

Completion Signaling A work request can be sent with signaled or unsignaled opcode. If the opcode of the work request is set to signaled, once the work request is completed a work completion element is generated. Unsignaled opcode generates no element to the completion queue and consequently, there isn't extra overhead. However, the latter approach cannot be adopted due to the resource depletion, and a signaled work request must be sent periodically to release the taken resources by unsignaled work requests. Finding the best period to send a signaled work request is a challenging task. The send queue depth and the message size can be considered as parameters in finding the best period [54].

Batching Once the CPU sends a list of requests to NIC instead of sending one request per each is called *batching*. The advantage of batching is reducing the number of CPU-NIC and network communications due to coalescing the requests. However, hardware limitation does not allow to batch requests of different QPs [86]. The batching scheme is more appropriate for the datagram transport due to its intuitive multicast support. So this scheme can be used to batch requests over datagram connections to multiple remote QPs. In addition, sending multiple requests in a message is another approach that allows the requester to send several requests to a specific responder and amortize communication overheads [6].

Atomic operations RDMA intrinsically provides a shared memory region in a distributed environment. Cross-access to the same memory region must be handled in order to avoid the race condition. RDMA supports two types of primitives to avoid concurrent access from other RDMA operations (not only atomic) on the same NIC: *fetch-and-add* and *compare-and-swap*. These operations adopt an internal lock mechanism of the NIC. The performance of these primitives depends on both the NIC atomic implementation mechanisms and the level of parallelism [86]. The atomicity of RDMA CAS is hardware-specific with different granularity (i.e., IBV_ATOMIC_HCA, IBV_ATOMIC_GLOB) [96]. However, there is no concurrency control between the

operation from local and the remote RDMA operations. To solve the concurrency problem, concurrent data structures [35], [37] have been proposed or special hardware instructions providing strong atomicity (i.e., hardware transactional memory in Intel processor) are exploited [36].

RDMA support Protocols InfiniBand is different from Ethernet in different aspects [97]. There are several InfiniBand alternative protocols supporting the RDMA technologies including RoCE and iWARP. Adopting an appropriate protocol requires the awareness of their advantages and drawbacks. The iWARP [30] is a complex protocol published by Internet Engineering Task Force (IETF). It only supports reliable connected transport (i.e., it does not support multicast) over a generic non-lossless network [98]. It was designed to convert TCP to RDMA semantics. On the contrary, RoCE is an Ethernet-based RDMA solution published by InfiniBand Trade Association supporting reliable and unreliable transports over lossless network [99]. There are several studies comparing these protocols over time [100], [101], [102], [103]. The consensus is that iWARP is unable to achieve the same performance of InfiniBand and RoCE.

There are protocols which are using technology like Intel DPDK to implelemt RDMA verbs over them [104]. However, they are not as performant as the native RDMA implementation in the NIC. While Data Plane Development Kit (DPDK) also provides kernel bypass that reduce the reliance on the CPU it does not go far enough and Mellanox makes the claim [105].

Wire Speed Mellanox InfiniBand has been made in 5 speeds: Single-Data Rate (SDR), Dual-Data Rate (DDR), Quad-Data Rate (QDR), Fourteen-Data Rate (FDR), and Enhanced Data Rate (EDR) offering 2.5, 5, 10, 14, 25 Gbps respectively, so enhancing the link speed increases the performance and decreases the latency [106].

Adaptor Connection InfiniBand adapters are based on PCI-X, PCIe. Different studies show that NIC based on PCIe outperforms PCI-X [81], [107], [108]. InfiniBand host communication has been started to be integrated on the chip of the processor [86], [109], [110].

2.5 Application level issues

Aside from all presented performance issues, an application must be suited for RDMA design, otherwise performance gain is negligible. RDMA is appropriate for applications that require shared memory access with low latency, long connection duration [111], and consistent alike message size [112]. The application level parameters that impact on the performance are presented in the following.

Memory registration Memory registration allows the RDMA device to read and write data from/to this memory. The cost of memory registration can be divided into three parts: (1) mapping virtual to physical memory, (2) pinning the memory region (3) registering the memory region to NIC driver. Memory registration is a costly operation because of the kernel call and the write operation to the NIC driver. Pre-registering the memory can eliminate this cost in runtime. However, if the applications can not store their data in a pre-register memory, then data need to be copied within a register memory region.

The cost of copying memory versus registering the new memory depends on the memory size and the power of the host [90]. The cost of the memory registration can be even more than RDMA operation itself (i.e., `WRITE`) [90]. So different techniques are proposed to solve this problem. When a new memory region is registered and the corresponding page is not resident in the memory, then the cost of page fault is added to the cost of memory registration. So ensuring the residency of the memory page before registering a memory region can decrease the cost of the memory registration. Memory allocation from kernel space (i.e., *_get_free_pages*) and registering in the kernel (e.g., *ib_reg_phys_mr*) instead of resorting to user space can decrease the memory registration latency [113]. Consequently, the first two steps of memory registration are eliminated since kernel memory space are physically contiguous and never swapped out. Since submitting a work request is not a blocking function, the overlapping memory registration with communication can hide the cost of registration. Yet comparing the cost of RTT with the cost of memory registration reveals that it fully depends on the size of memory registration [90]. Parallel memory registration can also hide the cost. This technique is particularly effective when pages are resident in memory.

Data Structures Hashtable is a popular data structure in a multi-client and multi-threaded server environment due to its fast direct lookups; however, hash collision is inevitable, which leads the increased number of probes. The higher number of probes in a hash table naturally increases the cost of lookup and pollutes the CPU cache by keys which are irrelevant. In many modern RDMA-based NoSQL systems [35], [37], Cuckoo and Hopscotch hashing are often employed [114], [115]. These hash tables strive to have the constant lookup cost by keeping the key in a bounded neighborhood to its original hash position.

Pipelining Pipeline allows simultaneous (sub)tasks at different stages. For example, pipeline can be used for memory registration and communication in order to hide the cost of registration [116]. This approach can improve the performance depending on the size of memory. However, [33] compares multi-threaded request handling pipeline versus single threaded request handling which multi-threaded request handling harms the performance.

Flow Isolation Latency-sensitive and throughput-sensitive applications may need to share the network resources (i.e., NIC) in large-scale environment. The application deployment is critical in such scenario to avoid the performance isolation of either types of applications. In the presence of both type of applications, a latency-sensitive flow will suffer [117]. So throughput-sensitive and latency-sensitive flows are better to be isolated.

In-bound vs. Out-bound Requests can be categorized to *inbound* and *outbound* according to the requester [34]. Sending the request from multiple clients to one server is called inbound, and sending requests from one server to multiple clients is called outbound. Before designing an RDMA-based application, measuring inbound and outbound throughput is important. Outbound is bottlenecked by PCIe and NIC processing power while inbound is bottlenecked by NIC processing power and InfiniBand bandwidth [86].

2.6 RDMA suitability

There are plenty of studies showing RDMA benefits over legacy protocols (i.e., TCP/UDP). However, there is a trade-off in using RDMA. We aimed to compare RDMA with legacy protocols to identify the cost of using RDMA in the pursuit of higher performance.

RDMA can perform one operation in each Round Trip Time (RTT), however traditional protocols provide more flexibility by fulfilling multiple operations. Thus, it makes RDMA more suitable for environments with single and quick operations. Furthermore, RDMA is more suitable for small and alike message sizes. Additionally, exploiting RDMA with dynamic connections cannot be beneficial due to the heavy initial cost of RDMA. Moreover, although RDMA provides high-speed communication, it limits scaling in both distance and size, meaning that the distance among nodes and the number of nodes cannot be arbitrarily large [118]. Generally speaking, programmability in RDMA is more involved in comparison with legacy protocols since the application layer requires to manage the concurrency control over the distributed shared memory. RoCE and InfiniBand require a lossless network. This limitation brings the management, scalability and deployment issues which are the main challenges while using these protocols. iWARP does not require such assumption, while this is great for generalizability, it results more complex NICs which leads to performance degradation in comparison with RoCE and InfiniBand [98]. In addition, there is a software type of RDMA protocols like SoftRDMA which is using Data Plane Development Kit (DPDK) technology to implement RDMA verbs over it [104]. DPDK is the Intel technology which provides kernel bypass that reduces the reliance on the CPU, however it is not as performant as the native RDMA implementation in the NIC [105].

Programmability in legacy protocols (i.e., TCP/IP) is simpler than RDMA. Moreover, TCP/IP is resilient to packet loss, while RoCE and InfiniBand are not. Generally speaking, legacy protocols will be replaced by RDMA if the problem is addressing volume and scaling up, and not ultra-high-performance problems.

3 NoSQL Systems

NoSQL systems, in particular in-memory key-value stores [33], [34], [35], [36], [37], [43], [119], [120] are vital in accelerating memory-centric and disk-centric distributed systems [1]. Modern in-memory key-value stores adopt RDMA to alleviate the communication and remote processing overheads. However, [121] presents the possible missing features of RDMA to meet the needs of future data processing applications such as in-memory key-value stores. It introduces the remote memory copy and conditional read to enable the client to remotely copy the data and to remotely filter the data before delivering it to the client. Moreover, introducing the dereferencing in RDMA can accelerate the existing systems by exploiting an indirection mechanism and omitting the cost of extra operation [122], [123], [124], [125]. Next, a comprehensive review of state-of-the-art RDMA-based key-value stores, examining indexing, consistency models, and communication protocols is provided.

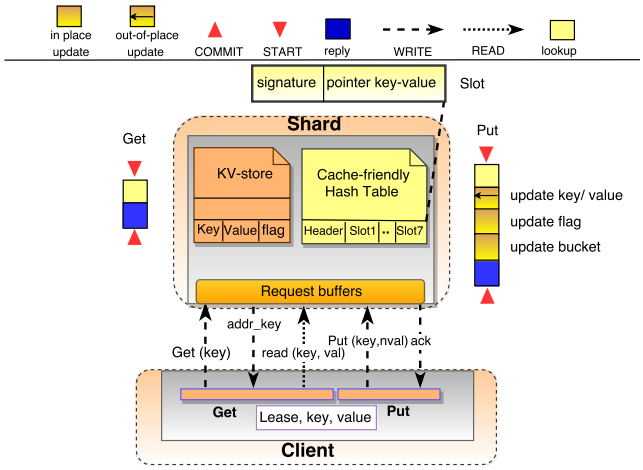

Fig. 2. Schematic view of HydraDB.

3.1 HydraDB

HydraDB is a general-purpose in-memory key-value store designed for low latency and high availability environment [33]. The HydraDB partitions data into different instances coupled with a single-threaded, partitionable execution model, called *shard*. Each shard is exclusively assigned to a core allows to fully exploiting the computational power and the cache of each core.

Each shard maintains a cache-friendly hash table with the memory location of the key-value stores. This hash table is not visible to the clients and is a compact hash table that has each bucket aligned to 64 bytes with 7 slots and a header to an extended slot, in order to avoid link list traversal which has an excessive pointer dereference. Each slot contains a signature of the key-value and a pointer. The server processes a request if its signature (i.e., a short hash key) matches to the requested key. Values are stored with a word-size flag in order to show the validity of the content and to avoid reading the stale data by client. Fig. 2 shows the schematic view of HydraDB. The scheme shows the indexing, communication protocol, and the required operations for *Get* and *Put* transactions.

HydraDB supports single-statement *Get* or *Put* transactions. Each client locates key-values according to the consistent hashing algorithm [126]. Clients and servers use `WRITE` to send and receive requests/responses because `WRITE` outperforms the other RDMA communications. In HydraDB, the shard process keeps polling a memory area to detect a new arrival message (i.e., sustained-polling). Each shard uses a single thread to poll the buffer requests. In the case of *Get*, shard first finds the corresponding key-value address in the compact hash table. Then, it replies to the client the address of the key-value pairs. So for the next request of the same key from the client, it exploits the `READ` based on the cached address.

Put operation fully relies on the server; the server finds the key in the compact hash table, then it flips the word-size flag of the key-value atomically to notify the readers about the update. Since remote read through RDMA may conflict with local write, shard exploits out-of-place update with lease-based time mechanism to guarantee the data consistency and memory reclamation, respectively [127].

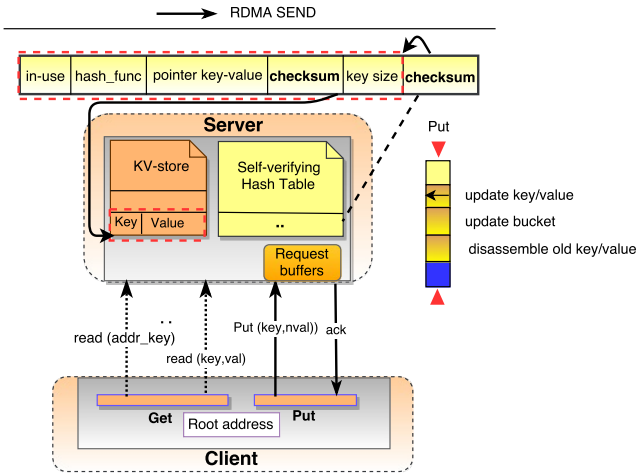

Fig. 3. Schematic view of Pilaf system.

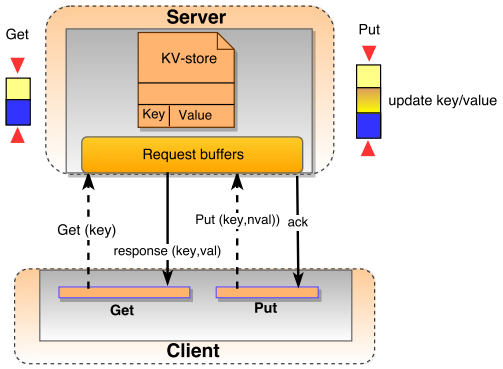

Fig. 4. Schematic view of HERD system.

3.2 Pilaf

Pilaf [35] is a distributed in-memory key-value store exploiting RDMA operation with the goal of reducing latency and improving the performance of traditional in-memory key-value stores (i.e., Redis [22] and Memcached [19]). As Fig. 3 demonstrates, Pilaf exploits two distinct memory regions visible to the client: a variable extent area and a fixed-size self-verifying hash table. The extent area is the region to store the actual value and each bucket in the fixed-size self-verifying hash table keeps the address of the key-value, its checksum, its value size, and the checksum of the bucket itself. Pilaf supports single-statement transactions *Get* and *Put*. Unlike the *Get*, the *Put* operation is fully server-driven. The *Get* operation probes fixed-size self-verifying hash table through RDMA `READ` to find the appropriate bucket for the key with the valid content (i.e., *in_use*). Since the address of the value is stored in the bucket, client can read the value in the extent area. The *Put* operation fully relies on the server to avoid the write-write race condition.

The Pilaf client and server uses `SEND` message to exchange request and response in *Put* operation. Once the Pilaf server receives a *Put* operation, first, it allocates a new memory location and updates it with the new value. Then it updates the corresponding bucket in the self-verifying hash table and disassembles the previous key-value content. Each bucket equipped with a checksum over the value to guarantee the data consistency. Disassembling the previous content notifies the clients about the recent update by the inconsistency between the read value and its checksum in the corresponding bucket. Moreover, each bucket is equipped with a checksum over the bucket itself to solve the race condition between the client's read and server's update on the same bucket. Once a client detects this inconsistency, it initiates the lookup in the hash table to retrieve the updated address of the key and its content.

3.3 HERD

HERD is an in-memory key-value store designed for efficient use of RDMA operations [34]. HERD adopts a simple lossy associative index, and a circular log for storing values (i.e., MICA back-end data structures [20]) , illustrated in Fig. 4. The clients write their requests (i.e., *Get*, *Put*) to the server using `WRITE` on an unreliable connection and, the server replies using `SEND` over an unreliable datagram. Adopting these operations are due to the scaling of `WRITE` and `SEND` in inbound and outbound communications. HERD's designers claim that single RTT communications (i.e., `WRITE`, `SEND`) combining with a memory lookup can outperform multiple RTT communications (i.e., `READ`).

Although it is shown that zero packet loss is detected in 100 trillion packets over unreliable datagram [91], there is the belief that unreliable transports may bring the unreliability to the enterprise applications [37]. HERD designers propose *FaSST* which is a transactional in-memory store with a loss detection algorithm over unreliable connection [91]. HERD applied several optimizations, such as window request, prefetching, selective signaling, huge pages, batching, multi-port request in order to enhance the performance [86].

3.4 FaRM

FaRM is a distributed in-memory transaction processing system designed to improve the latency and throughput of the TCP/IP communication [37]. FaRM exploits symmetric model in which each machine uses its local memory to store data. This symmetric model helps to exploit the local memory and the CPU which is mostly idle. The FaRM adopts two memory areas for storing and handling transactions: a chained associative hopscotch hash table and a key-value store area. FaRM employs a modified version of hopscotch hashing [115] that uses a chain in the bucket. A chain is used to keep the new data in the bucket instead of resizing the table in the overflow situation. However, it attempts to remove this chain and to move the last element of the chain to the available slot. Each bucket in the hopscotch hash table consists of an incarnation, the address of the value, and its size. FaRM stores small value sizes into the bucket and the bigger sizes in the key-value store area and keeps its address in the bucket. Values in FaRM are stored in a structure called *object*. Each object consists of a header version (V_{obj}), a lock (L), an incarnation (I), and cache line versions (V_c). The incarnation is used to determine the validity in case of the removed object. Lock, header and cache line versions are used to guarantee the data consistency. The FaRM supports multi-statement *Put* and *Get* transactions. Fig. 5 shows the FaRM data model and the

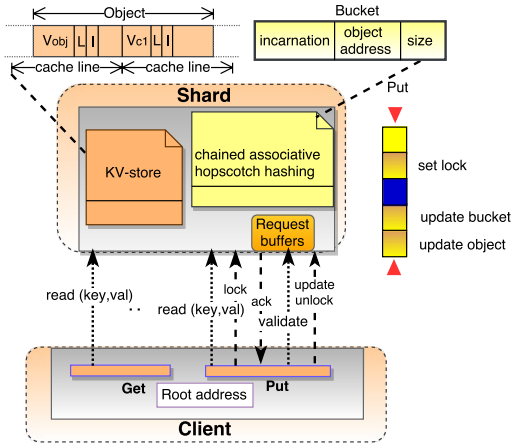

Fig. 5. Schematic view of FaRM system.

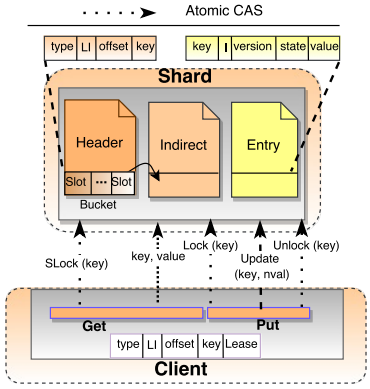

Fig. 6. Schematic view of DrTM.

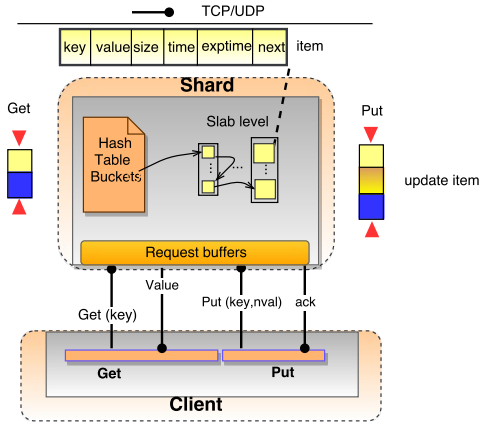

Fig. 7. Schematic view of Memcached.

interaction of the two transactions. *Get* operation uses `READ` for lookup process. It is performed by reading consecutive buckets according to the size of the neighbourhood (H) in hopscotch, H=6 in FaRM. In addition, the client checks the lock and the header version to be matched with all the cache line versions to guarantee the data consistency. In case of failure, the client retries to read after a backoff time. In case of *Put* transaction, client fetches the desired key from the shard, then it locks the key by sending a request to the shard. Shard sets the lock atomically (i.e., compare and swap) and sends the acknowledgment to the client. Afterwards, client validates the key by reading the key and sends the update (i.e., key and updated value) to the shard. Shard updates the value and finally, the cache line and header versions are incremented and the object is unlocked. FaRM uses fences after each memory write to guarantee the memory ordering in case of concurrent read.

FaRM uses `WRITE` and sustained-polling mechanism to exchange requests and responses as HydraDB. However, this approach is incompatible with out-of-order packet delivery, and a retransmitted packet from an old message might cause a memory to be overwritten and causes inconsistency in the execution [99].

3.5 DrTM

DrTM [36] and its successor DrTM+R [128] are in-memory key-value systems, which exploit concurrency instruction provided by modern CPUs. DrTM adopts traditional hash table with collocated memory regions for keys and values, called *cluster hashing*. Memory regions are managed in three different areas, called *main header*, *indirect header*, and *entry*, as shown in Fig. 6. The main header includes the incarnation, key, and its offset. The indirect header has the same structure as the main header and it is used in the overflow situation of the slots of the bucket in the main header. In this case the last slot of the bucket points to an available indirect header.

The value in DrTM is stored in a structure called *entry* containing the incarnation, value, version and status. Status represents the state of the key to perform *Get* and *Put*. DrTM supports multi-statement transaction *Get* and *Put*. To guarantee the data consistency, it uses a lease-based in combination with lock and Hardware Transactional Memory (HTM). At the beginning of a transaction, the executor locks the remote key through the one-sided atomic RDMA verb (i.e., compare-and-swap) and fetches the keys, then a local HTM is started. DrTM uses the strong consistency and atomicity of RDMA and HTM. Concurrent RDMA and HTM operations on the same memory will abort the HTM transaction. Once the transaction is committed, all remote keys are updated and locks are released.

3.6 Memcached

Memcached is a legacy in-memory key-value store based on TCP/IP protocol. It stores keys with their values into an internal hash table as shown in Fig. 7. It uses slab allocator to efficiently manage the memory according to the size of the key and value. Since Memcached supports multithreaded access to the hash table, the server orchestrates access through the locks. Memcached supports single-statement *Get* and *Put* transactions. Once the server receives a *Get* request, it finds the appropriate bucket and acquires the lock. Then, it replies the value or a miss, in case of missing the key, to the client. Once the server receives the *Put* request, it acquires the corresponding bucket lock, then updates the item. Memcached keeps an expiration (*exptime*) and the recent access time (*time*) to the key to replace the new items with the old ones in case of memory shortage according to the Least Recently Used (LRU) algorithm.

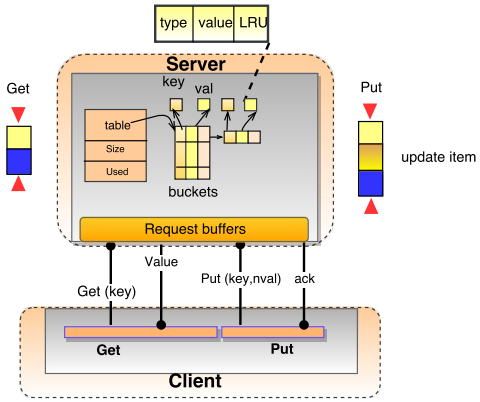

Fig. 8. Schematic view of Redis.

3.7 Redis

Redis is an in-memory key-value store with the ability to asynchronously store data on the disk. Flushing data to the disk can release memory space. Redis uses the TCP/IP communication scheme, and supports single-statement *Get* and *Put* transactions considering that it is single threaded and does not use locks for accessing the data. One of the main advantages of Redis is supporting the various value data types, such as lists and sets. Fig. 8 shows the communication and data model of the Redis. It uses *LRU* algorithm to flush the data to the disk; however, there is a high probability of data loss in case of system power loss events.

3.8 Systems Comparison

Table 1 differentiates aforementioned systems based on the usage of one-side or two-sided verbs, connection type in sending/receiving request/response, client-driven/server-driven operations, and architectural model of the systems. Client-driven/server-driven are operations that fully managed by the client/server. Architectural model captures the symmetric/asymmetric memory model by usage of client local memory in storing key-values.

Table 2 classifies the systems based on amplification in size and computation. Amplification highlights the extra amount of bytes or computation a client must exchange or compute in *Get* and *Put* operations. The word-size flag is the overhead of *Get* operation in HydraDB system. HydraDB client must check the flag and lease time which has computation overhead. In Pilaf, the client performs a lookup in the self-verifying hash table to find the address of the value, then it reads the value. To guarantee the consistency, the client must compute the checksum of the bucket and value. Client on average performs 1.6 probes in the self-verifying hash table plus one more `READ` to read the value. On average checksum computation is amplified by a factor of 2.6. In FaRM, the header and cache line versions are amplification to the value. The DrTM requires to read and set the *State* in *Get* and *Put* operations. Moreover, *Get* operation requires to read an Incarnation (*I*) and version.

Table 3 shows the data consistency, indexing, and transaction type of the systems. In-memory key-value stores can be categorized to single-statement (also called *caching systems*) and multi-statement transactions. In the single-statement systems, such as HydraDB [33], Pilaf [35], Herd [34], Memcached [34], and Redis [22] there is one operation in the transaction. However, multi-statement transaction, such as FaRM [37], and DrTM [36] have multiple operations in the transaction.

Each of the aforementioned systems has its own characteristics in its design which can carries its own drawbacks. These shortcomings for each system are discussed next.

HydraDB exploits lease mechanism to invalidate the remote cache. This method introduces overhead needed to synchronize outdated data items. It increases the number of control messages in the network. Furthermore, determining the lease period is a non-trivial task that is highly dynamic, can change overtime, and may vary significantly across different records. Technically speaking, HydraDB uses sustained-polling which violates the RDMA specification [99].

Pilaf uses cuckoo hashing which requires multiple reads to probe a key. So, the network communication overhead is amplified in Pilaf, hindering its maximum sustained throughput. Moreover, the use of checksum computation in Pilaf brings extra overhead not only for server but also for client [127].

HERD has simple design which uses unreliable communication transport to respond to clients which may lead to uncertainty or unreliability. Another disadvantage of this system is mixing the control and data planes. So, data are not able to move between the nodes.

FaRM uses hopscotch hashing which requires larger reads and consequently increases the read amplification. Moreover, it suffers from the same issue as HydraDB in violating the RDMA specification.

DrTM relies on specific hardware features (i.e., hardware transactional memory) that are not available in all CPUs, and may change across vendors or CPU generations. So, it cannot be seamlessly ported and executed on all hardware.

Memcached and Redis perform approximately the same for set and get operations. However, the impact of swapping pages in Memcached is rather high while the memory fragmentation in Redis is high [52].

4 Experimental Evaluation

Since our study focused on performance challenges of RDMA operations, it allowed us to capture bottlenecks of RDMA-based in-memory key-value stores. In order to study the impact of the RDMA design choices on the performance of the in-memory key-value stores, we performed a comprehensive set of experiments. We employed the Redis 3.2.9 [22], Memcached 1.4.37 [19], and HERD [129]. However, HydraDB, Pilaf, and FaRM are implemented from scratch since they are not publically available. All messages in HydraDB and FaRM are exchanged using the inline RDMA messages. Since DrTM uses a special limited CPU feature that is not widely accessible, we did not include it in the analysis. FaRM is the only system in our list that supports multi-statement transaction. Redis and Memcached are executed over IPOIB and the other systems have native RDMA over InfiniBand support.

4.1 Experimental Setting

To provide the reproducibility and interpretability of the experiments, the deterministic and non-deterministic parame-

KV Store	One-sided / (Two-sided)	Request / (Response) connection type	Client-driven / (Server-driven) operations	Architectural model
HydraDB	✓/ (✗)	RC / (RC)	✗/ (Get,Put)	Asymmetric
Pilaf	✓/ (✓)	RC / (RC)	Get / (Put)	Asymmetric
HERD	✓/ (✓)	UC / (UD)	✗/ (Get,Put)	Asymmetric
FaRM	✓/ (✗)	RC / (RC)	Get / (Put)	Symmetric
DrTM	✓/ (✗)	RC / (RC)	Get,Put / (✗)	Symmetric
Memcached	✗/ (✗)	RC / (RC)	✗/ (Get,Put)	Asymmetric
Redis	✗/ (✗)	RC / (RC)	✗/ (Get,Put)	Asymmetric

TABLE 1
Decoupling the communication of the existing systems.

KV Store	Size Get / (Put) Amplification	Computation Get / (Put) Amplification
HydraDB	Word / (0)	lease validity + check flag / (0)
Pilaf	(0) / (0)	(2.6) × CRC64 / (0)
HERD	0 / (0)	0 / (0)
FaRM	V_{obj} + I + L + (#cache lines - 1) * V_{c1} / (0)	check lock / (0)
DrTM	2 × State + I + version / (2 × State)	0 / (0)
Memcached	0 / (0)	0 / (0)
Redis	0 / (0)	0 / (0)

TABLE 2
Decoupling the client amplification of the existing systems.

KV Store	Data Consistency	Indexing	Transaction
HydraDB	flag & lease	compact hash table	Single
Pilaf	self-verifying	cuckoo hashing	Single
HERD	✗	lossy associative index	Single
FaRM	versioning	chained hopscotch hashing	Multiple
DrTM	lock and HTM	cluster chaining hashing	Multiple
Memcached	lock	chaining hash table	Single
Redis	✗	chaining hash table	Single

TABLE 3
Data consistency and indexing of existing systems.

ters which can impact on the performance are described in the following.

Deterministic Setting All benchmarks are compiled with the gcc version 4.4.7 with 50 seconds warmup and 25 seconds measuring time. In addition, to achieve the certainty on the result the experiments are repeated three times. Benchmarks are executed on a machine with 2 sockets AMD Opteron 6276 (Bulldozer) 2.3 GHz equipped with 20 Gbps DDR ConnectX Mellanox on PCI-E 2.0 with offload processing. The network topology is a direct connection with a Infiniscale-III Mellanox switch. Each machine has 4 NUMA nodes connecting to two sockets.

Non-deterministic Setting Query distribution (i.e., object popularity) is one of the main parameters in our experiments. Facebook analysis reported that web requests follow Zipfian-like distribution [130] [131], and the majority of in-memory systems present experiments based on the Zipfian distribution with high α ratio ($\alpha = 0.99$) indicating skew curve [33], [34], [35], [37]. In this paper, for comprehensiveness, two distributions are considered that closely model the real-world traffic: *Uniform* and *Zipfian*.

The number of keys is an important parameter in the experiment. For example, the server needs to register corresponding memory size to the number of keys which can increase the cache misses in the NIC to fetch the page table entries and influence on the performance. In this paper, we use 4 million keys, which can be seen as the active set. We experiment with the key and value size of 16 and 32 bytes, respectively, close to real-world workloads [6], [132]. Real-world workload ratio of read and write vary from *68%* to *99%* [133]. Thus, we consider the various read-write ratio workload ranging from read-only to write-only.

4.2 Process Mapping

Each process is mapped to a single core to avoid context switching and uses its local NUMA node. Since NIC (i.e., Mellanox adapters) installed over PCIe is closer to one of the CPUs on the board [84], we conduct an experiment to examine the impact when both sender and receiver are mapped on the closer CPU and once on the far CPU to NIC on two machines. Fig. 9 shows the bandwidth difference on `SEND` operation when the processes are bind to a close or far CPU from the NIC. Therefore, we pined the shard to the closer CPU to the NIC in all experiments. In addition, in order to stress each system, we dedicate one machine as the server with a single shard and one machine with multiple clients.

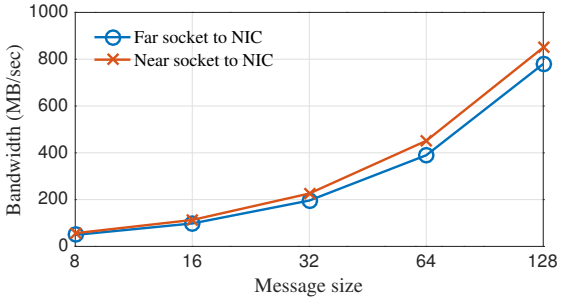

Fig. 9. RDMA Send bandwidth difference on far and close socket to NIC.

4.3 Memory issues

Memory alignment and cache misses are two important memory issues on the RDMA performance. In the following, we perform experiments to investigate their impacts.

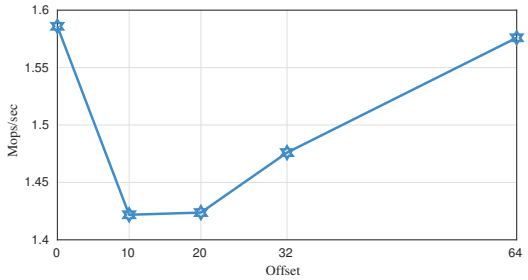

Fig. 10. Memory alignment performance evaluation.

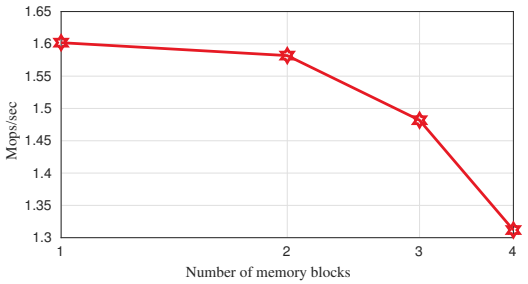

Fig. 11. Cache misses impact on the performance.

Memory alignment We allocated a memory block that is aligned with the page boundary (4096 bytes). Then, the client sends write requests that may reside on the different offset of the page. We conducted an experiment to show the throughput when we vary the offset alignment, namely, dividing the page into a set of slots such that a record may span multiple slots depending on its size. Fig. 10 illustrates that the throughput is always improved when the slot offset is aligned on a multiple of 64-bytes, which is aligned with the processors cache line.

Cache miss We performed an experiment when each message uses separate memory areas up to 4 so that the cache is rarely hit, and when the cache is always hit, which means no cached item is ever replaced. As can be seen in Fig. 11, when memory is assigned from the same memory block it has higher performance.

4.4 Throughput

Impact of varying the number of clients and workload read-write ratio on throughput Fig. 12 shows the results on different workload read-write ratios on Uniform distribution. As expected, the throughput of Redis and Memcached is order of magnitude slower than RDMA-based system in particular in a read-intensive workload. HydraDB scales well by increasing the number of clients and almost outperforms the other systems because of the smaller amplification in the HydraDB and the use of `READ` for *Get* statement in read-intensive workload. FaRM uses hopscotch hashing, and for each read, it requires reading a neighborhood consisting of six buckets; thus, FaRM suffers from read amplification. Although Pilaf uses `READ` in *Get* statement, it can not outperform HydraDB and FaRM due to the higher number of `READs` and cost of CRC64. In

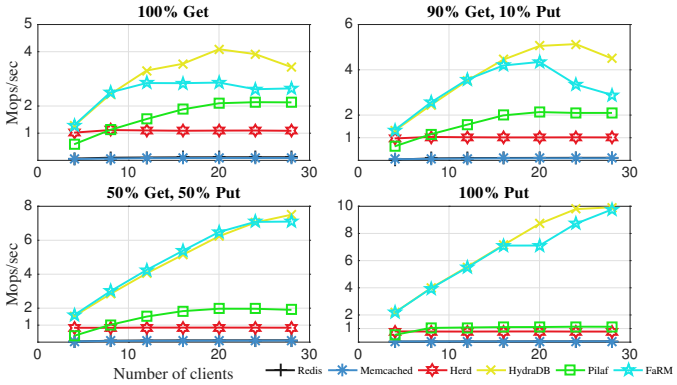

Fig. 12. Uniform throughput with single shard.

particular, the latency of CRC64 computation can be even higher than `READ` latency [127].

HERD uses `WRITE` over unreliable connection type for sending request, and `SEND` for responding to the request. Each client creates one QP to send its requests while the server creates one QP for each client to receive the requests. The server uses one QP for all clients to respond their requests and each client creates one QP for each server to receive the response. This mechanism puts more overhead on the clients due to the higher number of QPs. In our experiment, HERD is bottlenecked with the `SEND` operation [34]. Moreover, we run HERD without the hugepages which have an impact on the performance.

Throughput impact varying the workload read-write ratio To highlight the impact of decreasing the workload read-write ratio for both Zipfian and Uniform distributions, we devise an experiment with 24 clients while varying workload read-write ratio. As can be seen in Fig. 13, HydraDB and FaRM scale gracefully when decreasing the read-write ratio due to the higher performance of `WRITE` exploited in *Put* transaction comparing to the `READ` in *Get* transaction. The performance difference of `READ` and `WRITE` is due to the limited number of outstanding `READ` requests for each QP (i.e., our case 24) and the overhead of maintaining their state in the NIC. Moreover, `READ` uses PCIe non-posted transaction comparing to cheaper posted transactions for `WRITE` [34]. A non-posted transaction is a type of PCIe request, in which requester needs a response from the destination device unlike the posted transaction. Pilaf does not scale with decreasing the read-write ratio due to increased overhead of two-sided verb in *Put* operations. In the case of HERD, decreasing the read-write ratio does not have a noticeable impact on the performance since HERD is bottlenecked by the two-sided verb.

Throughput impact of the memory access distribution To isolate the impact of the memory access distribution on the performance, we performed an experiment with 24 processes reading remote memory either using Zipfian and Uniform distributions. We deploy data in the clustered (i.e., in order) and unclustered (i.e., random) way. We observed that both distributions perform identically with the small value size due to the locality of the data in the smaller portion of the memory. However, increasing the value size

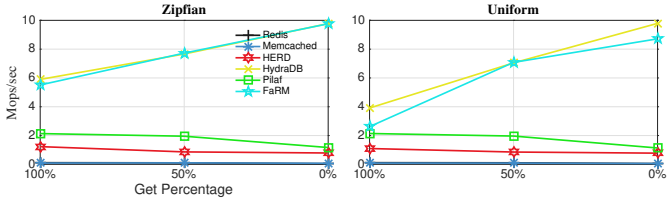

Fig. 13. Throughput comparison when varying read-write ratio for 24 clients.

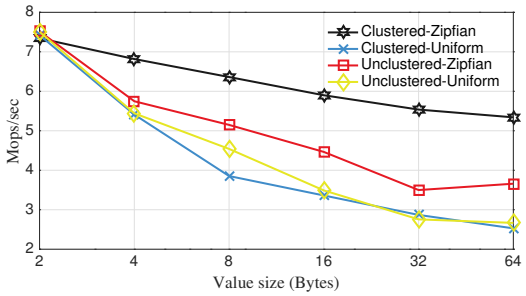

Fig. 14. Uniform and zipfian distributions throughput varying value size for clustered and unclustered insertions.

results reducing the locality, and Zipfian memory access outperforms (i.e., 2.1×) the Uniform due to higher data locality, as can be seen in Fig. 14. Furthermore, Zipfian distribution with clustered deployment has higher (i.e., 1.4×) performance comparing to the unclustered setting due to the higher data locality. Generally speaking, Zipfian distribution increases the chances of reusing cached data for hot keys. However, it will increase the race condition when updating hot keys. Fig. 15 shows the throughput results for Zipfian distribution.

4.5 Average latency

Fig. 16 and 17 show the latency of Uniform and Zipfian distributions. Increasing the number of clients will increase the number of requests and consequently will increase the latency.

HydraDB and FaRM on both distributions are two order of magnitude slower than legacy systems. By decreasing the read-write ratio of HydraDB and FaRM the latency decreases 2.5 and 2.3 times respectively. The drop is due to the lower latency of `WRITE` operation compared to `READ` used

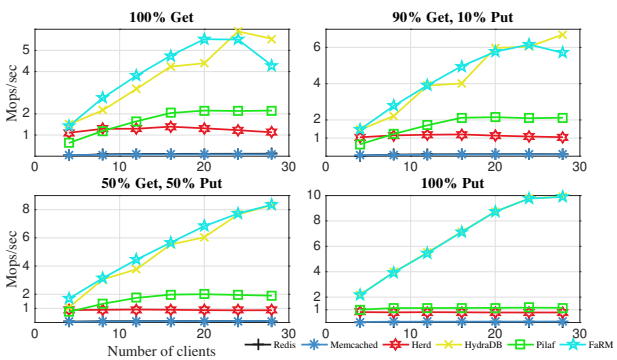

Fig. 15. Zipfian throughput with single shard.

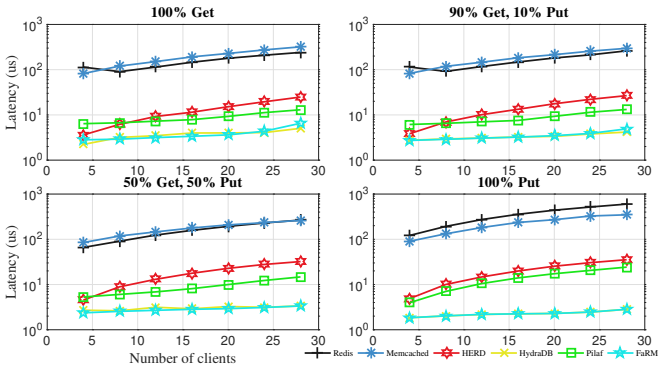

Fig. 16. Latency Zipfian.

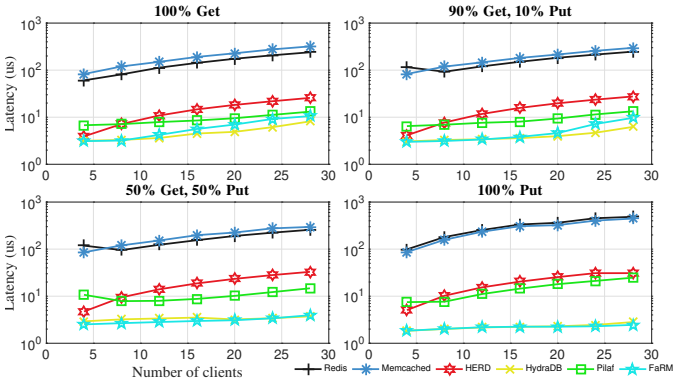

Fig. 17. Latency Uniform.

in *Put* and *Get* operations. However, decreasing the read-write ratio will increase the latency up to 2 times on Pilaf due to the higher latency of the verb messages comparing to `READ`. Since HERD uses the same operations in both *Get* and *Put* operations, there is no noticeable impact on the latency by decreasing the read-write ratio. Moreover, the latency of the Redis and Memcached are higher than RDMA-based systems because the use of the IP over InfiniBand instead of RDMA operations.

4.6 Value size

The payload size of RDMA message influences the throughput of `READ` and `WRITE` operations as shown in Fig. 18. Increasing the payload size up to 64 bytes does not have impact on the throughout. The throughput degradation after 64 bytes (i.e., cache line size) is due to the occupying the multiple cache lines in the Work Queue Element (WQEs). Increasing the WQE size will increase the MMIO and DMA operations in the inline and non-inline messages. The key difference between the inline and non-inline RDMA operations depends on the payload in the message, in which unlike in-line message, a non-inline message requires to read the payload by a DMA read operation [86]. The throughput of inline `WRITE` is 4.5 times higher than non-inline `WRITE` in small payload size in our experiment, and the reason is the elimination of the DMA reads in an inline message. Although inline message has higher throughput but it poses a limit on the message payload size. For example, it must be less than 1K bytes in the NIC employed in our evaluation. Moreover, increasing the payload size

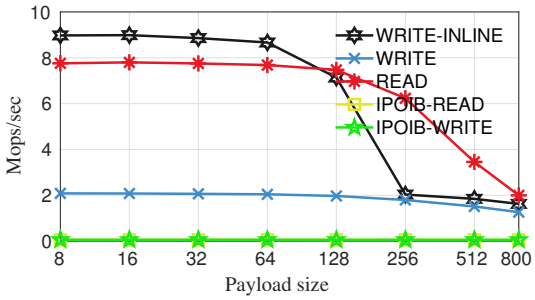

Fig. 18. Payload size impact on the performance with 24 processes on one machine and one remote process.

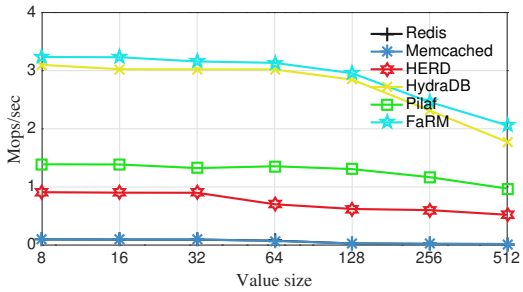

Fig. 19. Value size impact on the performance of the systems with 8 clients and 50% Get and Put.

causes higher throughput degradation in inline message comparing to non-inline messages.

We performed the same experiment using IPOIB. In the experiment clients read and write from/to a server. As can be seen in Fig. 18, these operations are less sensitive to the payload size. The performance degradation ratio in our experiment is 1.02%.

Since the bigger value in in-memory key-value systems implies the bigger payload size of RDMA message, we performed an experiment with 8 clients and 50% read-write ratio to measure the impact of varying the value size on the throughput. As Fig. 19 shows, the throughput of the RDMA systems reduce up to 1.7× by increasing the value size. Moreover, the HydraDB and FaRM are more sensitive to the size of the value due to inline `WRITE` in the experiment. The transition point when performance begins to drop depends on the NIC and the size of the value [86]. Since the performance degradation of IPOIB operations is much less than RDMA systems, the legacy systems are less sensitive to message size.

4.7 Uniformity Ratio

Since the server copes with a large number of simultaneous access, satisfying the uniformity of the client requests is essential. The uniformity ratio indicates the maximum number of completed requests over the minimum number of completed requests among the clients in a specific period of time. Closer ratio to 1 indicates the better uniformity and further ratio indicates the worse. We employ the uniformity ratio as it is an indicator of fairness [134]. As shown in Fig. 20, all RDMA-based systems have the uniformity ratio close

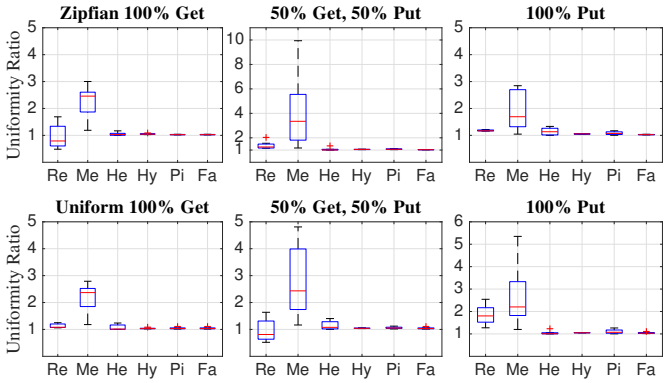

Fig. 20. Uniformity Ratio on 2 machines. Names are summarized to the first two letters.

to 1. However, requests in Redis and Memcached are not as uniform as RDMA based systems.

5 Conclusion and Future Direction

In this paper, we describes RDMA performance challenges and reviews the state-of-the-art in-memory key-value stores. We present a unified representation of a wide spectrum of NoSQL systems with respect to their indexing, consistency model, and communication protocols. We conduct extensive evaluation comparing HydraDB [33], FaRM [37], Pilaf [35], HERD [34], Memcached [19], and Redis [22]. Our work shows that operation using one-sided RDMA combining with hopscotch hashing in FaRM can outperform cuckoo hashing in Pilaf because of the higher number of probes (i.e., `READs`) in cuckoo hash table. However, hopscotch hashing requires to read a fixed size neighborhood, to have a constant lookup time, which has a substantial impact on the throughput comparing to the compact hash table in HydraDB. We have observed that exploiting one-sided verb (i.e., `WRITE`) in exchanging the request/response can outperform the two-sided verb. However, increasing the number of machines can influence the performance using connected QP [34]. We have further observed that the performance of memory access distribution is greatly influenced when the data is clustered or unclustered. The clustered access achieves higher performance by factor of 1.4. In addition, we have demonstrated that the latency of the legacy systems are up to 2 order of magnitude higher than the RDMA-based systems. We have shown that increasing the size of the value will decrease the performance, and systems using inline RDMA are more sensitive to the value. Finally, we have observed that RDMA-based systems can uniformly serve the requests comparing to the legacy systems.

Achieving high performance in key-value stores is not trivial and it requires solving several problems. Future directions in performance enhancement of key-value stores are placed in improving the network and storage drivers. For example, sophisticated operations such as dereferencing, traversing a list, conditional read, on-the-fly operations (e.g., compression and decompression), histogram computations, results consolidation [121] are possible features in future

RDMA protocols, which can be used to improve key-value stores. In addition, new storage class memory like Intel Optane can be used in key-value stores [135] as well as Non-Volatile Memory Express (NVMe) which is a state-of-the-art protocol for high-performance storage [136]. The advantage of NVMe is that it supports several transport protocols such as RDMA (i.e., RoCE and iWARP), Fibre Channel, and TCP [137].

ACKNOWLEDGMENT

Authors would like to thank to HPC project within the Department of Control and Computer Engineering at the Politecnico di Torino (http://www.hpc.polito.it).

REFERENCES

[1] X. Lu, D. Shankar, and D. K. Panda, "Scalable and Distributed Key-Value Store-based Data Management Using RDMA-Memcached." *IEEE Data Eng. Bull.*, vol. 40, no. 1, pp. 50–61, 2017.

[2] M. Sadoghi and S. Blanas, "Transaction processing on modern hardware," *Synthesis Lectures on Data Management*, vol. 14, no. 2, pp. 1–138, 2019.

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, 2008.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *Special interest Group in Operating Systems*, vol. 41, no. 6, pp. 205–220, 2007.

[5] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *Special interest Group in Operating Systems*, vol. 44, no. 2, pp. 35–40, 2010.

[6] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab *et al.*, "Scaling Memcache at Facebook." in *Networked Systems Design and Implementation*, vol. 13, 2013, pp. 385–398.

[7] D. Dai, X. Li, C. Wang, M. Sun, and X. Zhou, "Sedna: A memory based key-value storage system for realtime processing in cloud," in *Cluster Computing Workshops*. IEEE, 2012, pp. 48–56.

[8] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers *et al.*, "Practical lessons from predicting clicks on ads at facebook," in *International Workshop on Data Mining for Online Advertising*. ACM, 2014, pp. 1–9.

[9] C. Li, Y. Lu, Q. Mei, D. Wang, and S. Pandey, "Click-through prediction for advertising in twitter timeline," in *International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1959–1968.

[10] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu, "Tencentrec: Real-time stream recommendation in practice," in *International Conference on Management of Data*. ACM, 2015, pp. 227–238.

[11] G. Graefe, "The five-minute rule twenty years later, and how flash memory changes the rules," in *International workshop on Data management on new hardware*. ACM, 2007.

[12] P. Menon, T. Rabl, M. Sadoghi, and H.-A. Jacobsen, "CaSSanDra: An SSD boosted key-value store," in *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 2014, pp. 1162–1167.

[13] ——, "CaSSanDra: An SSD boosted key-value store," in *International Conference on Data Engineering*. IEEE, 2014, pp. 1162–1167.

[14] B. Debnath, S. Sengupta, and J. Li, "FlashStore: high throughput persistent key-value store," *Proceedings of the Very Large Database Endowment*, vol. 3, no. 2, pp. 1414–1425, 2010.

[15] "Flashcache," accessed: 12-October-2017. [Online]. Available: https://www.facebook.com/note.php?note_id=388112370932

[16] A. Anand, S. Kappes, A. Akella, and S. Nath, "Building cheap and large cams using bufferhash," *University of Wisconsin Madison Technical Report TR1651*, 2009.

[17] T. Kgil, D. Roberts, and T. Mudge, "Improving NAND flash based disk caches," in *International Symposium on Computer Architecture*. IEEE, 2008, pp. 327–338.

[18] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in RAMCloud," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 29–41.

[19] "Memcached: High-Performance, Distributed Memory Object Caching System," accessed: 12-October-2017. [Online]. Available: http://memcached.org/

[20] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A Holistic Approach to Fast In-Memory Key-Value Storage," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 429–444.

[21] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky, "SILT: A Memory-efficient, High-performance Key-value Store," in *Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2011, pp. 1–13.

[22] S. Sanfilippo, "Redis," accessed: 12-October-2017. [Online]. Available: http://redis.io/

[23] D. Sidler, Z. István, and G. Alonso, "Low-latency TCP/IP stack for data center applications," in *Field Programmable Logic and Applications*. IEEE, 2016, pp. 1–4.

[24] I. Pratt and K. Fraser, "Arsenic: A user-accessible gigabit ethernet interface," in *Joint Conference of the IEEE Computer and Communications Societies*, vol. 1. IEEE, 2001, pp. 67–76.

[25] T. Von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A user-level network interface for parallel and distributed computing," in *Special interest Group in Operating Systems*, vol. 29, no. 5. ACM, 1995, pp. 40–53.

[26] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, and C. Dodd, "The virtual interface architecture," *IEEE micro*, vol. 18, no. 2, pp. 66–76, 1998.

[27] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, "Myrinet: A gigabit-per-second local area network," *IEEE micro*, vol. 15, no. 1, pp. 29–36, 1995.

[28] J. Beecroft, D. Addison, D. Hewson, M. McLaren, D. Roweth, F. Petrini, and J. Nieplocha, "QSNET/sup II: defining high-performance network design," *IEEE micro*, vol. 25, no. 4, pp. 34–47, 2005.

[29] "InfiniBand," accessed: 12-October-2017. [Online]. Available: http://infinibandta.org/index.php

[30] "iWARP," accessed: 12-October-2017. [Online]. Available: https://tools.ietf.org/html/rfc5040

[31] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak, "Intel® omni-path architecture: Enabling scalable, high performance fabrics," in *High-Performance Interconnects*. IEEE, 2015, pp. 1–9.

[32] K. Gildea, R. Govindaraju, D. Grice, P. Hochschild, and F. C. Chang, "Remote direct memory access system and method," Aug. 30 2004, uS Patent App. 10/929,943.

[33] Y. Wang, L. Zhang, J. Tan, M. Li, Y. Gao, X. Guerin, X. Meng, and S. Meng, "HydraDB: A Resilient RDMA-driven Key-value Middleware for In-memory Cluster Computing," in *SC'15*.

[34] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *Special Interest Group on Data Communication*, vol. 44, no. 4. ACM, 2014, pp. 295–306.

[35] Y. G. Christopher Mitchell and J. Li, "Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*. San Jose, CA: USENIX, 2013, pp. 103–114.

[36] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen, "Fast In-memory Transaction Processing Using RDMA and HTM," in *Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2015, pp. 87–104.

[37] 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*, 2014, pp. 401–414.

[38] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur *et al.*, "Memcached design on high performance rdma capable interconnects," in *International Conference on Parallel Processing*. IEEE, 2011, pp. 743–752.

[39] W. Tang, Y. Lu, N. Xiao, F. Liu, and Z. Chen, "Accelerating Redis with RDMA Over InfiniBand," in *International Conference on Data Mining and Big Data*. Springer, 2017, pp. 472–483.

[40] "MongoDB," accessed: 12-October-2017. [Online]. Available: https://www.mongodb.com

[41] "HBase," accessed: 12-October-2017. [Online]. Available: http://hbase.apache.org/

[42] "VoltDB," accessed: 12-October-2017. [Online]. Available: https://www.voltdb.com

[43] B. Cassell, T. Szepesi, B. Wong, T. Brecht, J. Ma, and X. Liu, "Nessie: A Decoupled, Client-Driven Key-Value Store Using RDMA," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3537–3552, 2017.

[44] A. Gandini, M. Gribaudo, W. J. Knottenbelt, R. Osman, and P. Piazzolla, "Performance evaluation of NoSQL databases," in *European Workshop on Performance Engineering*. Springer, 2014, pp. 16–29.

[45] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser, "Performance evaluation of NoSQL databases: A case study," in *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*. ACM, 2015, pp. 5–10.

[46] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang, "In-memory big data management and processing: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920–1948, 2015.

[47] P. Cudré-Mauroux, I. Enchev, S. Fundatureanu, P. Groth, A. Haque, A. Harth, F. L. Keppmann, D. Miranker, J. F. Sequeda, and M. Wylot, "NoSQL databases for RDF: an empirical evaluation," in *International Semantic Web Conference*. Springer, 2013, pp. 310–325.

[48] J. R. Lourenço, B. Cabral, P. Carreiro, M. Vieira, and J. Bernardino, "Choosing the right NoSQL database for the job: a quality attribute evaluation," *Journal of Big Data*, vol. 2, no. 1, p. 18, 2015.

[49] "MySQL," accessed: 12-October-2017. [Online]. Available: https://www.mysql.com

[50] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, "Solving big data challenges for enterprise application performance management," *Proceedings of the Very Large Database Endowment*, vol. 5, no. 12, pp. 1724–1735, 2012.

[51] "Riak," accessed: 12-October-2017. [Online]. Available: http://docs.basho.com/riak/1.4.10/

[52] W. Cao, S. Sahin, L. Liu, and X. Bao, "Evaluation and analysis of in-memory key-value systems." IEEE, 2016, pp. 26–33.

[53] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *Special Interest Group for the computer systems performance evaluation community*, vol. 40, no. 1. ACM, 2012, pp. 53–64.

[54] Q. Liu and R. D. Russell, "A performance study of InfiniBand fourteen data rate (FDR)," in *Proceedings of the High Performance Computing Symposium*. Society for Computer Simulation International, 2014.

[55] B. M. T. G. C. Zhang, Hao and B. C. Ooi, "Efficient In-memory Data Management: An Analysis," *VLDB Endowment*, vol. 7, no. 10, pp. 833–836, Jun. 2014.

[56] "Redis Labs," accessed: 12-October-2017. [Online]. Available: https://redislabs.com/why-redis/

[57] "InfiniBandTM Architecture Specification Volume 2," accessed: 12-October-2017. [Online]. Available: https://cw.infinibandta.org/document/dl/7859

[58] "Top500," accessed: 12-October-2017. [Online]. Available: https://www.top500.org/statistics/list/

[59] G. Shainer and A. Ancel, "Network-based processing versus host-based processing: Lessons learned," in *Cluster Computing Workshops and Posters*. IEEE, 2010, pp. 1–4.

[60] A. Dragojevic, D. Narayanan, and M. Castro, "RDMA Reads: To Use or Not to Use?" *IEEE Data Eng. Bull.*, vol. 40, no. 1, pp. 3–14, 2017.

[61] S. Sur, M. J. Koop, D. K. Panda *et al.*, "Performance analysis and evaluation of Mellanox ConnectX InfiniBand architecture with multi-core platforms," in *High-Performance Interconnects*. IEEE, 2007, pp. 125–134.

[62] O. Shahmirzadi, "High-Performance Communication Primitives and Data Structures on Message-Passing Manycores:Broadcast and Map," 2014.

[63] "TILE-G," accessed: 12-October-2017. [Online]. Available: http://www.mellanox.com/page/multi_core_overview?mtag=multi_core_overview

[64] "kalray," accessed: 12-October-2017. [Online]. Available: http://www.kalrayinc.com/

[65] D. Slogsnat, A. Giese, M. Nüssle, and U. Brüning, "An open-source HyperTransport core," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 1, no. 3, 2008.

[66] D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek, "Intel® quickpath interconnect architectural features supporting scalable system architectures," in *High Performance Interconnects*. IEEE, 2010, pp. 1–6.

[67] "PCI Express," accessed: 12-October-2017. [Online]. Available: http://pcisig.com/specifications/pciexpress/

[68] "Intel Data Direct I/O Technology," accessed: 12-October-2017. [Online]. Available: https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html

[69] F. Mietke, R. Rex, R. Baumgartl, T. Mehlan, T. Hoefler, and W. Rehm, "Analysis of the Memory Registration Process in the Mellanox InfiniBand Software Stack," *International European Conference on Parallel and Distributed Computing*, pp. 124–133, 2006.

[70] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu, "Decoupled direct memory access: Isolating CPU and IO traffic by leveraging a dual-data-port DRAM," in *Parallel Architecture and Compilation*. IEEE, 2015, pp. 174–187.

[71] R. E. Bryant, O. David Richard, and O. David Richard, *Computer Systems A Programmers Perspective*. Prentice Hall Upper Saddle River, 2003, vol. 2.

[72] B. Lepers, V. Quéma, and A. Fedorova, "Thread and Memory Placement on NUMA Systems: Asymmetry Matters." in *USENIX Annual Technical Conference*, 2015, pp. 277–289.

[73] T. Li, Y. Ren, D. Yu, and S. Jin, "Analysis of NUMA effects in modern multicore systems for the design of high-performance data transfer applications," *Future Generation Computer Systems*, vol. 74, pp. 41–50, 2017.

[74] Y. Ren, T. Li, D. Yu, S. Jin, and T. Robertazzi, "Design and performance evaluation of NUMA-aware RDMA-based end-to-end data transfer systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013.

[75] A.-H. A. Badawy, A. Aggarwal, D. Yeung, and C.-W. Tseng, "Evaluating the impact of memory system performance on software prefetching and locality optimizations," in *Proceedings of the 15th international conference on Supercomputing*. ACM, 2001, pp. 486–500.

[76] D. Callahan, K. Kennedy, and A. Porterfield, "Software Prefetching," in *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS IV. New York, NY, USA: ACM, 1991, pp. 40–52.

[77] "AMD SenseMI," accessed: 12-October-2017. [Online]. Available: http://www.amd.com/en/technologies/sense-mi

[78] "3GIOPCIE," accessed: 12-October-2017. [Online]. Available: https://newsroom.intel.com/news-releases/pci-sig-introduces-pci-express-formerly-3gio-high-speed-serial-interconnect-specification/

[79] R. B. Thompson and B. F. Thompson, *PC Hardware in a Nutshell, 3rd Edition*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2003.

[80] R. Noronha and D. K. Panda, "Can high performance software DSM systems designed with InfiniBand features benefit from PCI-Express?" in *Cluster, Cloud and Grid computing*, vol. 2. IEEE, 2005, pp. 945–952.

[81] J. Liu, A. Mamidala, A. Vishnu, and D. K. Panda, "Performance evaluation of infiniband with pci express," in *High Performance Interconnects, 2004. Proceedings. 12th Annual IEEE Symposium on*. IEEE, 2004, pp. 13–19.

[82] "Understanding PCI Bus, PCI-Express and In finiBand Architecture," accessed: 12-October-2017. [Online]. Available: https://community.mellanox.com/docs/DOC-2491

[83] "Understanding Performance of PCI Express Systems," accessed: 12-October-2017. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp350.pdf

[84] "Understanding PCIe Configuration for Maximum Performance," accessed: 12-October-2017. [Online]. Available: https://community.mellanox.com/docs/DOC-2496

[85] "IRQ Affinity," accessed: 12-October-2017. [Online]. Available: https://community.mellanox.com/docs/DOC-2123

[86] A. K. M. Kaminsky and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *2016 USENIX Annual Technical Conference*, 2016.

[87] M. Flajslik and M. Rosenblum, "Network Interface Design for Low Latency Request-Response Protocols." in *USENIX Annual Technical Conference*, 2013, pp. 333–346.

[88] "Mellanox Adapters Programmers Reference Manual," accessed: 12-October-2017. [Online]. Available: http://www.mellanox.com/related-docs/user_manuals/Ethernet_Adapters_Programming_Manual.pdf

[89] S. Sur, A. Vishnu, H.-W. Jin, D. Panda, and W. Huang, "Can Memory-Less Network Adapters Benefit Next-Generation InfiniBand Systems?" in *High Performance Interconnects, 2005. Proceedings. 13th Symposium on*. IEEE, 2005, pp. 45–50.

[90] P. W. Frey and G. Alonso, "Minimizing the hidden cost of RDMA," in *International Conference on Distributed Computing Systems*. IEEE, 2009, pp. 553–560.

[91] A. Kalia, M. Kaminsky, and D. G. Andersen, "FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs." in *Operating Systems Design and Implementation*, 2016, pp. 185–201.

[92] 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 *Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2015, pp. 54–70.

[93] "Annex A14: Extended Reliable Connected (XRC) Transport Service," accessed: 12-October-2017. [Online]. Available: https://cw.infinibandta.org/document/dl/7146

[94] T. Shanley, *Infiniband Network Architecture*. Addison-Wesley Professional, 2003.

[95] P. MacArthur and R. D. Russell, "A Performance Study to Guide RDMA Programming Decisions," in *International Conference on High Performance Computing and Communication & International Conference on Embedded Software and Systems*. IEEE, 2012, pp. 778–785.

[96] "RDMA Aware Networks Programming User Manual," accessed: 12-October-2017. [Online]. Available: http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf

[97] P. MacArthur, Q. Liu, R. D. Russell, F. Mizero, M. Veeraraghavan, and J. M. Dennis, "An integrated tutorial on InfiniBand, Verbs and MPI," *IEEE Communications Surveys & Tutorials*, 2017.

[98] "RoCE vs. iWARP Competitive Analysis," accessed: 12-October-2017. [Online]. Available: https://www.mellanox.com/pdf/whitepapers/WP_RoCE_vs_iWARP.pdf

[99] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting Network Support for RDMA."

[100] J. Vienne, J. Chen, M. Wasi-Ur-Rahman, N. S. Islam, H. Subramoni, and D. K. Panda, "Performance Analysis and Evaluation of InfiniBand FDR and 40GigE RoCE on HPC and Cloud Computing Systems," in *High-Performance Interconnects*. IEEE, 2012, pp. 48–55.

[101] M. Beck and M. Kagan, "Performance Evaluation of the RDMA over Ethernet (RoCE) Standard in Enterprise Data Centers Infrastructure," in *Proceedings of the 3rd Workshop on Data Center - Converged and Virtual Ethernet Switching*, ser. DC-CaVES '11. International Teletraffic Congress, 2011, pp. 9–15.

[102] H. Subramoni, P. Lai, M. Luo, and D. K. Panda, "RDMA over EthernetA preliminary study," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–9.

[103] L. A. Kachelmeier, F. V. Van Wig, and K. N. Erickson, "Comparison of High Performance Network Options: EDR InfiniBand vs. 100Gb RDMA Capable Ethernet," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2016.

[104] "RDMA over DPDK," accessed: 12-October-2017. [Online]. Available: https://www.openfabrics.org/images/eventpresos/2017presentations /114_Urdma_PMacArthur.pdf

[105] "RDMA vs DPDK," accessed: 12-October-2017. [Online]. Available: http://www.mellanox.com/related-docs/whitepapers/WP_heavyreadingNFVperformance.pdf

[106] M. J. Koop, W. Huang, K. Gopalakrishnan, and D. K. Panda, "Performance Analysis and Evaluation of PCIe 2.0 and Quad-Data Rate InfiniBand," in *High Performance Interconnects*. IEEE, 2008, pp. 85–92.

[107] J. Liu, A. Mamidala, A. Vishnu, and D. K. Panda, "Evaluating InfiniBand performance with PCI express," *IEEE Micro*, vol. 25, no. 1, pp. 20–29, 2005.

[108] "InfiniBand PCI PCIE," accessed: 12-October-2017. [Online]. Available: http://www.mellanox.com/pdf/ whitepapers/PCI_3GIO_IB_WP_120.pdf

[109] "Intel Xeon Processor D-1500 Product Family," accessed: 12-October-2017. [Online]. Available: https://www.intel.in/content/dam/www/public/us/en/documents/product-briefs/xeon-processor-d-brief.pdf

[110] "Intel Xeon Phi Processor Knights Landing Architectural Overview," accessed: 12-October-2017. [Online]. Available: https://www.nersc.gov/assets/Uploads/KNL-ISC-2015-Workshop-Keynote.pdf

[111] D. Dalessandro and P. Wyckoff, "Accelerating Web Protocols Using RDMA," in *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*. IEEE, 2007, pp. 205–212.

[112] K. Magoutis, S. Addetia, A. Fedorova, and M. I. Seltzer, "Making the Most Out of Direct-Access Network Attached Storage." in *USENIX Conference on File and Storage Technologies*, 2003.

[113] L. Ou, X. He, and J. Han, "An efficient design for fast memory registration in RDMA," *Journal of Network and Computer Applications*, vol. 32, no. 3, pp. 642–651, 2009.

[114] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *European Symposium on Algorithms*. Springer, 2001, pp. 121–133.

[115] M. Herlihy, N. Shavit, and M. Tzafrir, "Hopscotch hashing," in *International Symposium on Distributed Computing*. Springer, 2008, pp. 350–364.

[116] D. Dalessandro and P. Wyckoff, "Memory management strategies for data serving with RDMA," in *High-Performance Interconnects*. IEEE, 2007, pp. 135–142.

[117] Y. Zhang, J. Gu, Y. Lee, M. Chowdhury, and K. G. Shin, "Performance Isolation Anomalies in RDMA," in *Workshop on Kernel-Bypass Networks*, 2017, pp. 43–48.

[118] A. Romanow and S. Bailey, "An Overview of RDMA over IP," in *Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks*, 2003.

[119] X. Wu, L. Zhang, Y. Wang, Y. Ren, M. Hack, and S. Jiang, "zExpander: a key-value cache with both high performance and fewer misses," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 14.

[120] K. Zhang, K. Wang, Y. Yuan, L. Guo, R. Lee, and X. Zhang, "Mega-KV: a case for GPUs to maximize the throughput of in-memory key-value stores," *Proceedings of the Very Large Database Endowment*, vol. 8, no. 11, pp. 1226–1237, 2015.

[121] C. Barthels, G. Alonso, and T. Hoefler, "Designing Databases for Future High-Performance Networks." *IEEE Data Eng. Bull.*, vol. 40, no. 1, pp. 15–26, 2017.

[122] M. Sadoghi, M. Canim, B. Bhattacharjee, F. Nagel, and K. A. Ross, "Reducing database locking contention through multi-version concurrency," *Proceedings of the Very Large Database Endowment*, vol. 7, no. 13, pp. 1331–1342, 2014.

[123] ——, "Reducing Database Locking Contention Through Multi-version Concurrency," *Very Large Database*.

[124] M. Sadoghi, K. A. Ross, M. Canim, and B. Bhattacharjee, "Exploiting SSDs in operational multiversion databases," *Very Large Database*, vol. 25, no. 5, pp. 651–672, 2016.

[125] M. Hemmatpour, B. Montrucchio, M. Rebaudengo, and M. Sadoghi, "Kanzi: A distributed, in-memory key-value store," in *International Middleware Conference*. ACM, 2016, pp. 3–4.

[126] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM, 1997, pp. 654–663.

[127] Y. Wang, X. Meng, L. Zhang, and J. Tan, "C-Hint: An Effective and Reliable Cache Management for RDMA-Accelerated Key-Value Stores," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SOCC '14. New York, NY, USA: ACM, 2014, pp. 23:1–23:13.

[128] 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*. ACM, 2016.

[129] "HERD Source code," accessed: 12-October-2017. [Online]. Available: https://github.com/efficient/rdma_bench

[130] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications,"

in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 1999, pp. 126–134.

[131] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An Analysis of Facebook Photo Caching," in *Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2013, pp. 167–181.

[132] B. Fan, D. G. Andersen, and M. Kaminsky, "MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing." in *Symposium on Networked Systems Design and Implementation*, vol. 13, 2013, pp. 371–384.

[133] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1. ACM, 2012, pp. 53–64.

[134] D. Petrović, T. Ropars, and A. Schiper, "Leveraging Hardware Message Passing for Efficient Thread Synchronization," in *Symposium on Principles and Practice of Parallel Programming*. ACM, 2014, pp. 143–154.

[135] "Intel Optane," accessed: 12-October-2017. [Online]. Available: https://arxiv.org/pdf/1903.05714.pdf

[136] "NVMe Key-value store," accessed: 12-October-2017. [Online]. Available: http://mvapich.cse.ohio-state.edu/static/media/talks/slide/shankar-sc18_osu_booth_2.pdf

[137] D. Minturn, "NVM Express Over Fabrics," in *11th Annual OpenFabrics International OFS Developers Workshop*, 2015.

Masoud Hemmatpour received the M.S. degree in Computer and Communication Network Engineering and the Ph.D. degree in Control and Computer Engineering both from Politecnico di Torino, Italy, in 2015, and 2019, respectively. His research interest includes High Performance Computing (HPC) in the context of concurrent programming and dynamic memory management in parallel systems utilizing modern hardware, such as hardware message passing and Remote Direct Memory Access (RDMA). He is currently a postdoctoral fellow in Cisco System. He focuses on control and management plane in cloud-native environment.

Bartolomeo Montrucchio received the M.S. degree in Electronic Engineering and the Ph.D. degree in Computer Engineering both from Politecnico di Torino, Italy, in 1998, and 2002, respectively. He is currently an Associate Professor of Computer Engineering at the Dipartimento di Automatica e Informatica of Politecnico di Torino, Italy. His current research interests include image analysis and synthesis techniques, scientific visualization, sensor networks and RFIDs.

Maurizio Rebaudengo (M'95)(S'14) received the M.S. degree in Electronics (1991), and the Ph.D. degree in Computer Engineering (1995), both from Politecnico di Torino, Italy. He is an IEEE Senior Member since 2014. Currently, he is a Full Professor at the Dipartimento di Automatica e Informatica of the same institution. His research interests include ubiquitous computing and testing and dependability analysis of computer-based systems.

Mohammad Sadoghi is an Assistant Professor in the Computer Science Department at the University of California, Davis. Formerly, he was an Assistant Professor at Purdue University and Research Staff Member at IBM T.J. Watson Research Center. He received his Ph.D. from the University of Toronto in 2013. He leads the Expo-Lab research group with the aim to pioneer a distributed ledger that unifies secure transactional and real-time analytical processing (L-Store), all centered around a democratic and decentralized computational model (ResilientDB). He has cofounded a blockchain company called Moka Blox LLC, the ResilientDB spinoff. He has over 80 publications in leading database conferences/journals and 34 filed U.S. patents. He served as the Area Editor for Transaction Processing in Encyclopedia of Big Data Technologies by Springer. He has co-authored the book "Transaction Processing on Modern Hardware", Morgan & Claypool Synthesis Lectures on Data Management, and currently co-authoring a book entitled "Fault-tolerant Distributed Transactions on Blockchain" also as part of Morgan & Claypool series.