RAMBDA: RDMA-driven Acceleration Framework for Memory-intensive μs-scale Datacenter Applications

Yifan Yuan[†*§], Jinghan Huang[†],Yan Sun[†], Tianchen Wang[†], Jacob Nelson[‡], Dan R. K. Ports[‡], Yipeng Wang[*], Ren Wang[*], Charlie Tai[*], Nam Sung Kim[†]
[†]*University of Illinois at Urbana-Champaign*, [*]*Intel Labs*, [‡]*Microsoft Research*
{yifan.yuan, yipeng1.wang, ren.wang, charlie.tai}@intel.com
{jinghan4, yans3, tw12, nskim}@illinois.edu {jacob.nelson, dan.ports}.microsoft.com

***Abstract*—Responding to the "datacenter tax" and "killer microseconds" problems for memory-intensive datacenter applications, diverse solutions including Smart NIC-based ones have been proposed. Nonetheless, they often suffer from high overhead of communications over network and/or PCIe links. To tackle the limitations of the current solutions, this paper proposes RAMBDA, a holistic network and architecture co-design solution that leverages current RDMA and emerging cache-coherent off-chip interconnect technologies. Specifically, RAMBDA consists of four hardware and software components: (1) unified abstraction of inter- and intra-machine communications synergistically managed by one-sided RDMA write and cache-coherent memory write; (2) efficient notification of requests to accelerators assisted by cache coherence; (3) cache-coherent accelerator architecture directly interacting with NIC; and (4) adaptive device-to-host data transfer for modern server memory systems comprising both DRAM and NVM exploiting state-of-the-art features in CPUs and PCIe. We prototype RAMBDA with a commercial system and evaluate three popular datacenter applications: (1) in-memory key-value store, (2) chain replication-based distributed transaction system, and (3) deep learning recommendation model inference. The evaluation shows that RAMBDA provides 30.1∼69.1% lower latency, 0.2∼2.5× throughput, and ∼ 3× higher energy efficiency than the current state-of-the-art solutions, including Smart NIC. For those cases where RAMBDA performs poorly, we also envision future architecture to improve it.**

***Index Terms*—cache-coherent interconnects and accelerators, RDMA, heterogeneous and disaggregated memory, datacenters**

I. INTRODUCTION

Datacenter networks are evolving rapidly. 100 Gbps Ethernet is widely deployed today, and so will be 400 Gbps Ethernet soon [121]. To keep pace, a server may have to process hundreds of millions of packets per second. However, single-thread performance of CPUs has remained comparatively stagnant, requiring the CPUs to spend more cores and their cycles for network processing alone – a major component of the "datacenter tax" [79]. For application processing, accelerators can be used. Yet, conventional accelerators are inefficient for processing many small tasks [147], and thus unable to address "killer microsecond" problem [13, 110].

Current approaches use one of three strategies. (Tab. I). First, kernel-bypass networking, using user-space network stacks [15, 63, 71] or two-sided Remote Direct Memory Access (RDMA) [75–78, 98], reduce packet processing overhead by delivering data directly to user space, avoiding kernel crossings. Nonetheless, both user-space network stacks and application processing tax server CPU cores [75–78, 98]. Second, one-sided RDMA [27, 39, 111, 112, 118, 160, 161] allows clients to bypass the server CPU and directly read or write server memory. However, the limited semantics of one-sided RDMA operations require multiple network round trips to serve a single request from a client [19]. Finally, Smart NICs can perform more sophisticated remote operations in a single network round trip [7, 83, 94, 130, 154], offering higher execution and energy efficiency than host CPUs [84, 94, 172]. Nevertheless, this solution sometimes yields lower performance than the first and second solutions [7, 116] for two reasons. First, *cost, power, and form factor* constraints limit Smart NICs memory capacity to $O(10GB)$ [100], and applications that do not fit in on-NIC memory must access main memory over slow PCIe links [116]. This leads to significant performance degradation [19, 84, 94, 116, 141]. Note that many modern datacenter applications have large working sets, often requiring not only DRAM but also higher-density byte-addressable NVM (e.g., Intel Optane Persistent DIMM [67]) to cost-effectively provide capacity [11, 18, 84]. Second, Smart NICs use specialized accelerators and/or energy-efficient wimpy CPUs that are often unable to run applications with high performance. As such, processing must be partitioned and coordinated between Smart NIC and the faster server CPU, but frequent communications over slow PCIe links become a performance bottleneck.

As a result, system architects face a dilemma: *whether to use (many) expensive CPU cores for application processing, or to suffer from the performance overhead incurred by multiple round trips over network or PCIe links*. This paper proposes an alternative, RAMBDA that can efficiently serve memory-intensive μs-scale datacenter applications. Specifically, RAMBDA envisions a server with a standard RDMA NIC (RNIC) and a *cache-coherent accelerator* (cc-accelerator) [30, 69] connected to an *cache-coherent off-chip interconnect* (cc-interconnect) such as CXL [33]. The cc-accelerator *directly communicates with* the RNIC for efficient network and application processing with little involvement of the server CPU. RAMBDA takes a modularized approach – the cc-accelerator is a separate device, not integrated with the RNIC – to allow the use of standard RNICs and support application-specific customization of the cc-accelerator. As such, RAMBDA is a cost-effective and efficient

[§]This research was partially done when the first author was at UIUC.

TABLE I: Taxonomy of hardware-based μs-scale datacenter applications offloading/acceleration.

Optimization	Net. Overhead	PCIe Overhead	CPU Overhead	Flexibility	Perf. Stability
Two-sided RDMA/kernel-bypass with multi-core [75–78, 98]	Low	Low	High	High	Low
One-sided/mixed RDMA [27, 39, 111, 112, 118, 160, 161]	High	High	Low	Low	Low
(Smart)NIC offloading [7, 19, 24, 83, 84, 94, 130, 136, 141, 145, 154]	Low	High	Low	High	Low
RAMBDA	**Low**	**Low**	**Low**	**High**	**High**

approach for accelerating a broader range of applications. To our knowledge, RAMBDA is the first work to explore the role of a cc-accelerator for end-to-end datacenter applications.

RAMBDA consists of four logically-coupled software and hardware components. Specifically, we propose:

- A unified abstraction for inter- and intra-machine communications, using lockless ring buffers to facilitate inter-machine communications with one-sided RDMA write and CPU-accelerator communications with `load/store`;
- A fast and efficient mechanism to notify the cc-accelerator of requests, exploiting its access to coherence information;
- A cc-accelerator architecture for processing the requests and handling RNIC-CPU-accelerator interactions; and
- An adaptive device-to-host data transfer mechanism for a server with a DRAM-NVM heterogeneous memory system.

We prototype RAMBDA using a commercial system based on an Intel Xeon 6138P CPU, which integrates an on-package FPGA connected to the CPU coherence bus. We evaluate three popular μs-scale datacenter applications: (1) an in-memory key-value store, where requests fully bypass the CPU; (2) a chain replication-based distributed transaction processing system – an example of a latency-sensitive system with NVM; and (3) a deep learning recommendation model (DLRM) inference, which requires collaboration between a server CPU and a cc-accelerator to process requests. We show that RAMBDA provides 30.1–69.1% lower latency, up to 0.2∼2.5× throughput, and 3× higher power efficiency than current state-of-the-art solutions. However, RAMBDA is not a panacea – for the cases where it is significantly slower than the current CPU-based solutions, we find that RAMBDA's memory bandwidth can be a critical bottleneck compared to other solutions, limiting the speed data is retrieved in bandwidth-bound workloads. Hence, we also envision and demonstrate that a future cc-accelerator with accelerator-local memory can further improve latency and throughput by 11.2% and 62.1×, respectively.

II. BACKGROUND AND MOTIVATION

A. RDMA Primer

RDMA allows machines to access the memory of remote machines at high bandwidth and low latency. RDMA has now been widely deployed by datacenters [49, 55, 166, 180] and used to build various research and production systems. RDMA offloads the network transport stack to RNIC hardware and supports operations completely in user space, bypassing the kernel. One-sided RDMA operations (*e.g.*, read/write/atomics) completely bypass the remote server's CPU for remote memory accesses. Meanwhile, two-sided RDMA operations (*e.g.*, send/receive) are similar to conventional network communications (*e.g.*, TCP/UDP) as they involve the CPUs of both clients and servers for data transmission.

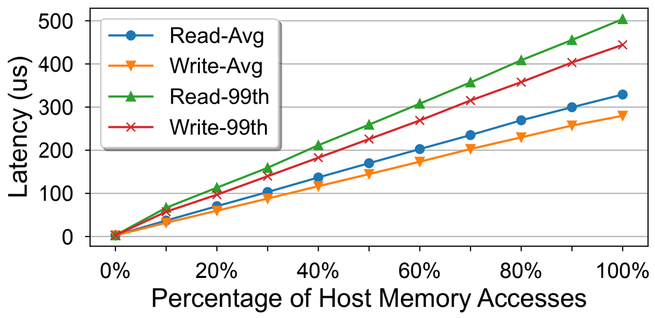

Fig. 1: Random memory access request latency from Smart NIC for different percentages of host memory accesses. Each data point contains 100 back-to-back 64B memory accesses.

The key data structures in RDMA programming are queue pair (QP) and completion queue (CQ), shared between the host (user space) and the RNIC. A QP consists of two work queues (WQs): a send queue (SQ) and a receive queue (RQ), both ring buffers in the host memory. To post an RDMA operation, the user writes to a work queue entry (WQE) at the tail of the WQ with a pre-defined device-specific format and rings the RNIC's doorbell register using an MMIO write. Upon completion of the RDMA operation, the RNIC (optionally) writes to a completion queue entry (CQE) at the tail of the CQ (also a ring buffer in the host memory) associated with the QP. Hosts learn about operation completion by polling the CQ.

B. Dilemma of Using Smart NIC

Recent Smart NICs integrate either FPGA or customized low-profile CPU with NICs. Prior work has shown that Smart NICs running datacenter applications can offer higher performance and energy efficiency than host CPUs [84, 94]. However, Smart NICs have limited memory capacity (*O(10 GB)* [100]) under *cost, power, and form factor* constraints. As such, they often need to access the host memory when running applications with large working sets. Unfortunately, such host memory accesses are not cheap, primarily because they must go through PCIe links. This has been identified by multiple system designs [19, 84, 94, 116, 141]. We also quantify this effect using an experiment on a NVIDIA BlueField-2 Smart NIC (see Sec. V for the detail of our testbed). We run a simple benchmark on the NIC's ARM cores, which randomly access data from one of 1 GB pre-allocated buffers, one in the NIC's on-board DRAM and the other in the host DRAM. The benchmark accesses the on-board DRAM with `load/store`

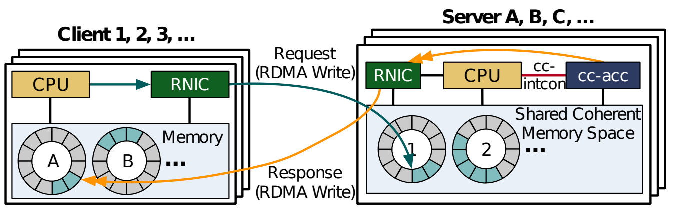

Fig. 2: RAMBDA's high-level system architecture; only the snapshot of client-1 and server-A are demonstrated.

instructions and accesses the host DRAM with one-sided RDMA read/write (we use direct verbs [137] to minimize the RDMA software stack overhead). In each request from the benchmark, we consecutively access the memory 100 times (64 bytes for each access). We report the request latency for different percentages of local/host access in Fig. 1 (each data point is based on 1M requests). For example, "80%" means 80% of the 100 memory accesses are to the host DRAM and the remaining 20% are to the on-board DRAM. Both the average and 99th-percentile tail latency values increase linearly with more memory accesses to the host DRAM. This is attributed to long latency of going through the physical PCIe link, memory management unit (MMU), DMA engine, and I/O controller. Such a high overhead can also affect throughput, although host memory accesses can be batched/pipelined to hide latency.

Consequently, the Smart NIC is only well suited for applications with working sets that are either small enough to fit in the Smart NIC's local on-board memory or effective for caching (*i.e.*, sufficient spatial locality or skewed distributions of memory accesses). Otherwise, the frequent communications between the NIC and the host will adversely affect the end-to-end performance of applications.

C. Cache-coherent Interconnects and Accelerators

Originally, cc-interconnects were developed for NUMA systems where CPUs share the memory space in a cache-coherent manner; recent examples include UPI and Infinity Fabric. More recent cc-interconnect designs such as CXL [33], CAPI [148], and CCIX [25] support fine-grained data sharing between CPUs and accelerators (accessed via standard chip-to-chip physical links such as PCIe). Accelerators built on such cc-interconnects are referred to as cc-accelerators. Because host-accelerator communication is cheaper, especially for small data sizes, cc-accelerators can be more efficient than conventional accelerators.This makes cc-accelerators suitable for μs-scale acceleration/offloading. Currently, some cc-accelerators are commercially available [29, 30, 69], and more are likely to emerge as CXL support becomes more widespread [125].

III. RAMBDA SYSTEM ARCHITECTURE

We depict a high-level system architecture of RAMBDA in Fig. 2. RAMBDA envisions a system comprising a cc-accelerator, a standard RNIC, and a CPU, all cooperating for efficient network and application processing. Specifically, (1) the cc-accelerator not only offloads parts of application processing from the server CPU but also *directly interacts* with the RNIC for client communication without involving the server CPU; (2) the standard RNIC handles network stack processing; and (3) the server CPU tackles accelerator-unfriendly (irregular and branch-rich) part of application processing ("fast path") in addition to initialization, control, and management of hardware resources, applications, and network connections ("slow path"). The synergistic orchestrations among (1) – (3) are facilitated by RAMBDA's four software and hardware components described in this section. Also, RAMBDA envisions a unified memory subsystem with both CPU-attached and accelerator-attached physical memory, which should be in the same address space and coherence domain. Depending on different workloads, the latency/bandwidth requirements of the accelerator-attached memory may vary, but we do envision that the capacity is at the same level as the CPU-attached memory [30, 32, 68] so that more application data can be mapped to the accelerator-attached memory.

A. Inter- and Intra-machine Communication

RAMBDA provides a fast and efficient communication abstraction for both inter-machine communication (between a server and clients) and intra-machine communication (between a server's CPU and cc-accelerator). Although the underlying data structure (lock-free ring buffer) is not new and has been employed before [39, 154], our abstraction unifies inter- and intra-machine communication using the same programming model with unique features of cache-coherence and RDMA. For each client-server connection, we establish a pair of a request ring buffer (in the server memory) and a response ring buffer (in the client memory) for inter-machine RDMA communications. For example, buffer-1 on server-A and buffer-A on client-1 form a request-response buffer pair for a connection between client-1 and server-A in Fig. 2. Further, for each accelerator in a server, we establish one request-response ring buffer pair in the server memory for intra-machine communications. One-sided RDMA write is used by both servers and clients for high-performance inter-machine communications through message passing where all the underlying network transport processing is offloaded to the RNIC [39, 154]. For intra-machine communications, leveraging the shared coherence domain, the server CPU or cc-accelerator directly writes and read the ring buffers. This abstraction not only guarantees high performance by avoiding extra CPU cycles on the communication stack, but also facilitates the accelerator operations, as it can fetch application data directly (instead of accessing intermediate data first, like any form of descriptor).

Note that we do not share the ring buffers (and the underlying RDMA QPs for the inter-machine communications) across different client-server connections, to avoid performance overheads with atomic updates or consistency issues at the head/tail of the buffer without atomic updates. However, we do allow sharing the ring buffers (and the RDMA QPs) across threads on the same machine for better scalability, as a software layer/library can manage cross-thread contentions with slight performance overheads [28, 113, 155]. Specifically, we employ the Flock's method [113], *i.e.*, one dedicated

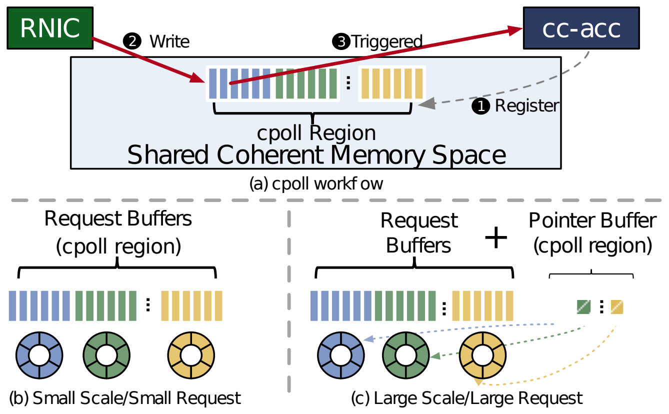

Fig. 3: `cpoll` region for cc-accelerator notification.

thread on the client for request synchronization and dispatch, so that there is only one request-response buffer pair (and QP) per client-server pair per application and observe no performance loss compared to native RDMA primitives.

The client is responsible for tracking the tail of the request buffer in the server memory and the head of the response buffer in its local memory, similar to the credit-based flow control [86]. Whenever it writes a message to the request buffer, it will update its local record of the request buffer's tail; whenever it receives a message in the response buffer (by polling), it will update its local record of the response buffer's head and reset the buffer entry to "0". Only if the request buffer's tail is behind the response buffer's head can the client issue a request. Otherwise, it knows that the buffer is full of on-the-fly requests and should not send more requests. A similar mechanism is applied to the server for request buffer's head and response buffer's tail. This guarantees that any message can be passed by only one network trip without any conflict.

B. Coherence-assisted Accelerator Notification

Since messages are directly written to request buffers in server memory, the RAMBDA cc-accelerator needs to detect their arrival. Typically, this would be done with spin-polling. However, frequent polling wastes the limited bandwidth of the cc-interconnect, leaving little available for accessing memory during application processing. Polling also leads to high power consumption [14, 45, 52], fast transistor aging [122], and poor scalability with many queues [52]. Hence, we propose a coherence-assisted notification mechanism, called `cpoll`.

Conceptually and semantically, `cpoll` is similar to `MWAIT` in the x86 architecture [65], HyperPlane's `QWAIT` [109], and PCIe's lightweight notification (LN) proposal [51, 132]. `cpoll` differs from them because it is designed to be portable and platform-agnostic (*i.e.*, requires no CPU modification and no specific protocol) for off-chip devices. Specifically, we insert a `cpoll checker` in the datapath of the coherence controller's port connected to the cc-interconnect. During initialization, we first allocate the inter- and intra-machine communication request buffers (Sec. III-A) in a contiguous region of server memory (*i.e.*, `cpoll` region), and register this region to the cc-accelerator's `cpoll checker` for snooping, as depicted in Fig. 3(a). Then, when the cc-accelerator's coherence controller receives a coherence signal from the registered address region (*e.g.*, `Modified` → `Invalid`), it will notify the cc-accelerator of arrival of a request. The `cpoll checker` will only need to monitor a single address region illustrated in Fig. 3(a). If the address of a coherence signal falls into this region, the `cpoll checker` can identify which request buffer (associated with a specific client or the server CPU) received a new request based on its address. By monitoring only one region, and using consecutive, fixed-size buffers, this dispatch is trivially scalable. (Even if buffers are not allocated consecutively in memory, HyperPlane [109] has demonstrated reasonable address lookup overhead to thousands of buffers.)

To implement the `cpoll` mechanism, we propose two approaches. First, we allocate the `cpoll` region in CPU-attached memory, and then pin the region on the cc-accelerator's local cache. Note that the cc-accelerator resets the request buffer entry associated with a `cpoll` signal after it processes the request. This makes the cc-accelerator's local cache always own the `cpoll` region from the cache coherence viewpoint, so any change to the `cpoll` region by clients or the server CPU triggers a coherence signal. Alternatively – though our prototype platform does not support this – the `cpoll` region could be allocated from memory attached to the cc-accelerator. As such, any request from the RNIC to the request buffers in server memory will go through the cc-interconnect. Subsequently, they will be delivered to the cc-accelerator's coherence controller which is responsible for monitoring any change to the `cpoll` region.

The first approach is feasible with our prototype platform (Sec. V), but the size of request buffers is constrained by the cc-accelerator's local cache size, limiting its scalability at the moment. When the scale of the system is large (*i.e.*, many request buffers) or each request itself is large (*i.e.*, large buffers), we cannot pin the entire `cpoll` region on the cc-accelerator's cache. To tackle this scalability issue in our setup, we introduce a data structure called pointer buffer where each 4-byte entry corresponds to each inter- or intra-machine request buffer and stores a pointer (or index) to an entry in the request buffer, as depicted in Fig. 3(c). Subsequently, we register the pointer buffer allocated to a contiguous address space as the `cpoll` region. When writing a new request to a request buffer in the server memory, a client or the server CPU will also increment the value of the pointer buffer entry corresponding to the request buffer such that the pointer buffer entry points to the request buffer tail. For a remote client, this can be efficiently done by posting two contiguous WQEs (only the second one is signaled) with a batched doorbell to the RNIC [77] or remapping/interleaving the two buffers with user-mode memory registration (UMR) [120] and only posting one WQE.

Note that one additional small PCIe write to the server side is inevitable in both ways. However, since RAMBDA has already reduced the PCIe traffic and mainly leverages the coherence traffic, such overhead will not notably hurt the overall performance, which is confirmed by our experiments in Sec. VI. In addition, as a 4-byte pointer buffer entry covers an entire request buffer, which can be as large as several MBs

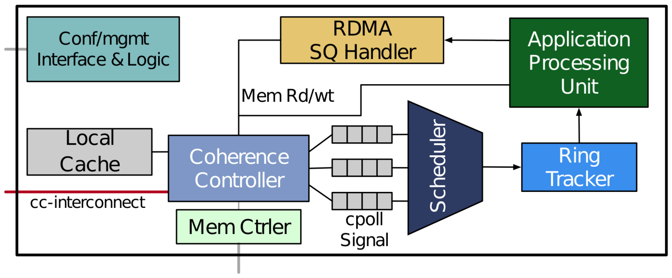

Fig. 4: RAMBDA cc-accelerator architecture.

for some applications such as the one described in Sec. IV-B, it can substantially reduce the memory space requirement for the `cpoll` region. Finally, coherence signals are not guaranteed to come in the order of actual data writes. However, this does not affect the correctness of `cpoll` because it is designed to be used with a ring buffer, and leverages the semantics of the ring buffer, *i.e.*, request buffer entries are written in order.

C. RAMBDA *cc-accelerator Architecture*

The RAMBDA coherence controller (Fig. 4) handles all the coherence traffic (both regular read/write and `cpoll`) to or from the cc-accelerator, as well as the virtual-physical address translation (*i.e.*, TLB). The local cache is also in the coherence domain and handled by the coherence controller. The RAMBDA cc-accelerator may have its own local memory controller and memory [30, 32] that constitutes the unified memory space with the CPU memory. In such an architecture, the CPU may allocate application data to the cc-accelerator's local memory, as the NUMA-aware memory management does in the modern Linux kernel.

The scheduler fetches `cpoll` signals associated with different request buffers based on a given scheduling algorithm. The nature of the coherence bus may cause `cpoll` signals to be coalesced. For example, two updates to the same entry in the pointer buffer in a short period might generate only one `cpoll` signal. However, we leverage the semantics of the ring buffer, by tracking the previous tail of the request buffer in the cc-accelerator. This allows it to determine how many new requests were received since the last notification and inform the application processing unit (APU) accordingly.

The RDMA SQ handler is responsible for direct RNIC control. It assembles the response information from the APU into WQE format, then writes it to the corresponding RDMA connection's WQ, and rings the RNIC's doorbell register. Since the cc-accelerator is in the coherence domain with unified address space, we do not need to store the entire WQ into the SQ handler. Instead, during the initialization, only the starting address and number of entries of each WQ is registered to the SQ handler from the software. Later, upon the response, it will write information into the WQ one by one, causing little scalability overhead on the cc-accelerator.

Since polling the CQ is not on the critical datapath, we do not process it with the cc-accelerator. Instead, we use a single CPU core to handle all the CQs polling and bookkeeping. Unsignaled WQE [77] is applied here so that only the selected operations will notify the CQ of their completion. This can alleviate the overhead of RNIC-CPU communication when the CPU is polling multiple CQs. Besides, this helps reduce unnecessary traffic on the cc-interconnect.

The APU is *the only application-specific part* in the entire RAMBDA architecture, yielding a fine balance between RAMBDA programmability and user implementation complexity. It provides the user with *standard interfaces* for (1) `cpoll` signal reception, (2) coherent data read/write, and (3) RDMA WQE output. First, a (de)serializer can be optionally used, if the application uses an RPC protocol for inter-machine communications [89]. Then, to process requests, we typically need a data structure walker [53, 85, 103, 168, 171] to find the location of the target data of the request. To maximize the memory-level parallelism and hide the memory access latency, multiple outstanding requests and out-of-order execution should be supported. Inspired by the stateful network function accelerator [133], we employ a table-based finite state machine for this purpose, where the outstanding request status is stored in a TCAM or cuckoo hash table [177] for fast lookup. Upon the arrival of a new request or intermediate result, the corresponding TCAM or hash table entry is updated and then the next-step action is issued to a corresponding functional unit (*e.g.*, ALU or coherence controller).

The APU should invoke the CPU in two scenarios. The first scenario is when a library call or OS `syscall` is needed. For example, if the user space memory pool has been pre-allocated by the CPU (`malloc/mmap`), the APU itself can allocate objects for new data in the memory pool [94]; if not, `malloc` is called each time when a new object is needed. The second scenario is when CPU is more suitable than the APU for a certain part of application processing. For example, in a recommender inference system (see Sec. IV), while the APU can handle the embedding reduction and fully-connected layers, the request preprocessing (*e.g.*, transforming a human-readable request to a model input) should still run on the CPU due to its irregularity and complexity. In these scenarios, the cc-accelerator and CPU interact with low latency in a fine-grained manner described in Sec. III-A.

D. Optimizing Device-host Data Transfer: Adaptive DDIO

Having the RAMBDA system design, we finally consider the optimization of device-memory-cache interaction inside a single machine, or specifically, how to choose between the cache and memory as the data destination for optimal device-host data transfer. Given the data-intensive nature of RAMBDA's usage scenarios, this optimization is notably important for the entire system. As the device I/O speeds increase, Intel introduced data-direct I/O (DDIO) [70], a CPU-wide technology, to allow the device to directly inject data to the CPU's last level cache (LLC) instead of main memory. This reduces memory bandwidth consumption and latency required by I/O.

DDIO has been proven to be effective [4, 20, 46, 47, 74, 87, 105, 131, 153, 162, 170], improving the performance of DRAM-based systems [4, 20, 87]. However, it does not always improve performance of NVM-based systems [74, 162]. which

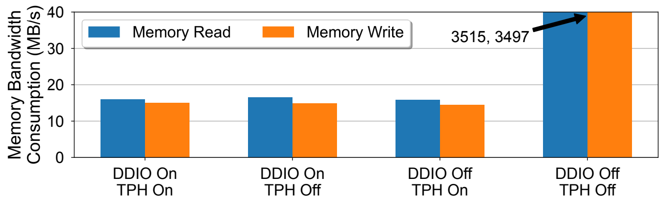

Fig. 5: Memory bandwidth consumption by PCIe-bench's DMA write with different DDIO/TPH settings.

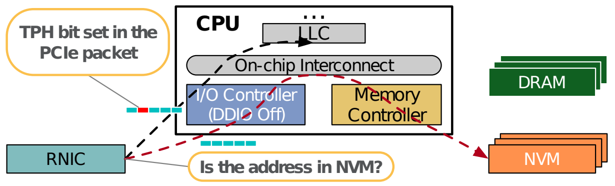

Fig. 6: DDIO/TPH configurations in the system with NVM.

is increasingly deployed by datacenters to cost-effectively provide large memory capacity for applications such as in-memory database [11, 18]. This is mainly because of two reasons. **(1)** NVM usually has a larger access granularity than DRAM and cache. For example, the access granularity of the Intel® Optane DIMM is 256 bytes while that of DRAM and cache is 64 bytes in Intel-based system [167]. When the DDIO-ed data is evicted from LLC to NVM, the write-back to the NVM will be randomized because of the cache replacement policies. As a result, write amplification wastes the bandwidth of NVM [74], which is already lower than that of DRAM. **(2)** CPU caches are typically not persistent; Intel® eADR [64] makes cache as part of the persistency domain but it requires a large battery and has high power consumption [6, 16]. As such, applications often have to flush data in cache to NVM to remain correct in case of a crash, with performance cost [151].

To tackle the aforementioned limitation, we propose to exploit a rarely-discussed field in the PCIe packet header, TLP processing hints (TPH). It is the 16th bit in the PCIe header and a performance feature that allows the CPU to prefetch or keep certain PCIe writeback data in LLC for quick consumption by CPU cores [123]. To our best knowledge, no current commercial I/O device (including SSD and NIC) uses the TPH bit; it is always set to 0 as a placeholder in both hardware and device drivers.

Our experiment confirms that changing the TPH bit allows us to control the destination of data to either LLC or memory per PCIe packet. *Being the first to clarify the DDIO-TPH relationship on modern and general server platforms*, we perform an experiment running PCIe-bench [116] on a VC709 FPGA board [165] in which we implement a module that allows an API to set the TPH bit on-the-fly for each PCIe packet. The FPGA DMAs (random write) data to the (DRAM-based) host at a constant speed of 3.5GB/s. We measure the memory bandwidth consumption on the host side in four configurations (DDIO on/off + TPH on/off) in Fig. 5. Only when both DDIO and TPH are off, do we observe large memory bandwidth consumption (*i.e.*, ∼3.5GB/s for both read and write), which is aligned with the DMA throughput reported by the FPGA. This indicates that all DMA data is sent to the main memory. Otherwise, if *either* DDIO or TPH is on, there will be little memory bandwidth consumption, meaning data is sent to the LLC directly.

Since TPH is applicable to each PCIe packet, we propose two guidelines for future systems with heterogeneous memory, as depicted in Fig. 6. (1) DDIO should be disabled globally on the CPU by default. (2) The device should expose the knob of changing the TPH bit for the programmer. Taking RNIC as an example, one way to do this would be to make it a configuration parameter set when registering a memory region to the RNIC, specifying whether the registered address range belongs to DRAM or NVM (or based on more flexible conditions). Later, when executing an RDMA operation (*e.g.*, write), the RNIC hardware will set the TPH bit only for operations in DRAM regions, avoiding DDIO-induced write amplification on NVM regions. While this requires hardware modifications to the RNIC, our experiment that adds this functionality to the original PCIe-bench shows that it adds almost no cost to the NIC hardware design. Following these two guidelines, we can make DDIO NVM-aware independently for each I/O device.

E. RAMBDA *Programming Model*

We provide a software user-space framework/interface for upper-level applications. An application needs to register itself to the RAMBDA framework with initialization information, such as connection establishment, the memory region of the application data, and the targeting cc-accelerator. The RAMBDA framework then is responsible for allocating the request/response buffers by `mmap/malloc` calls and registering the buffers to the RNIC by verbs APIs. Also, it makes the application's and the request/response buffers' virtual memory space visible to the cc-accelerator. To pin the `cpoll` region to cc-accelerator's local cache, the framework writes the cc-accelerator's configuration registers, which are accessible in user-space as well through MMIO. The coherence controller then will never evict the corresponding cachelines from the local cache.

We do not restrict any programming model for the APU, as there could be a huge diversity in different applications, and it is not the focus of this work. Programmers may refer to other related work [104, 171, 174] to explore how a single model/instance can support many applications.

F. RAMBDA *Scalability*

Scalability with faster network. As the speed of network is growing fast, a critical question is whether RAMBDA will keep up with the speed of future network (*e.g.*, 400Gbps). First, in our implementation (Sec. V) and experiments (Sec. VI), the cc-interconnect's bandwidth is not saturated in RAMBDA KV and RAMBDA TX. Furthermore, accelerator-attached memory with comparable capacity to the CPU [30, 32, 68] will further liberate the bandwidth of the cc-interconnect from application-related memory requests. Hence, the cc-interconnect can better serve the RNIC/CPU-cc-accelerator interaction. This means RAMBDA will be bottlenecked by the network bandwidth and can achieve higher performance with newer network technologies (also note that the cc-interconnect performance will evolve as well).

Scalability with larger cluster. With the advance in SoC technologies [118, 119] (*e.g.*, larger on-NIC cache for connection information), the previous scalability issues and on-NIC resource constraints of reliable RDMA connections [75, 77, 78, 90] have been alleviated. Building on top of RDMA, RAMBDA offers the same scalability as RDMA does. The dedicated buffer pair does not limit scalability as each buffer is small, *e.g.*, 1MB for 1K entries that are enough for most request sizes and saturate network bandwidth in modern datacenters [80]. To support connections from 1K clients, each running a single application, a server only needs to allocate 1GB of its main memory, a small fraction of the total memory capacity of modern servers. Also, sharing the buffers across threads does not limit scalability either [113].

In fact, RAMBDA is not strictly bound to the existing commodity RDMA technology, and more recent remote memory access (RMA) variants [10, 143, 146, 159] (as long as they share the same semantics) can be adopted to RAMBDA for better scalability.

IV. RAMBDA USE CASES

A. In-Memory Key-Value Store

In-memory key-value store (KVS) is a basic building block of many datacenter services. In addition to software optimizations, researchers have leveraged all the three directions mentioned in Sec. I for further KVS acceleration. The major requirement of KVS is *high memory access parallelism across requests*.

An RAMBDA design for KVS, dubbed RAMBDA KV aims to *fully offload* request processing to the cc-accelerator. At the algorithm/data structure level, RAMBDA KV is similar to MICA [98], but RAMBDA KV follows the architectural description of Sec. III-C at the hardware level, including a pipelined hash unit for hash value/index calculation. RAMBDA KV performs a GET/UPDATE request by calculating the hashed key value and finding the corresponding entry in the set-associated hash table's bucket. The entry contains a pointer to the actual key-value data. For PUT requests, after finding the address where a new key-value pair should be allocated (*i.e.*, an empty entry in the bucket), the slab allocator will simply put it in the pre-defined memory pool. If the bucket indexed by the hashed key is full (*i.e.*, hash collision), another bucket with the same format will be allocated and linked to the existing bucket by a pointer. Similar to KV-Direct [94] and MICA [98]'s study, on average, each GET request requires three memory accesses and each PUT request requires four.

B. Distributed Transaction with NVM-based Chain Replication

Distributed transactional systems are widely used by datacenters to provide the ACID feature for distributed storage systems. To this end, cross-machine protocols for data replication are usually needed, and chain replication [5, 8, 12, 12, 22, 43, 43, 50, 50, 108, 114, 124, 124, 129, 138, 150, 152, 156, 178] is a popular primary-backup replication protocol. In chain replication, machines are virtually organized into a linear chain. Any change to the data will begin at the head of the chain and pass through the chain. When the last machine in the chain makes the change in its log, it will back-propagate the ACK signal through the chain so that each machine can locally commit the transaction. When the head of the chain commits the transaction, it sends the ACK signal back to the client, marking the end of the transaction.

The state-of-the-art work, HyperLoop [83], leverages the RNIC and NVM to achieve low-latency chain replication with little CPU involvement. Specifically, it proposes and implements group-based RDMA primitives, which can be triggered automatically by the RNIC. One key-value pair (addressed by the offset in the NVM space) is modified in the entire chain once the client initiates a group-based RDMA operation. However, due to the restricted (group-based) RDMA semantics, to process multi-value transactions, the client needs to sequentially issue RDMA operations for each key-value pair, which often leads to long latency in the network and PCIe link.

An RAMBDA design for such a distributed transaction system, dubbed RAMBDA TX is similar to RAMBDA KV with respect to request processing, but it additionally implements a concurrency control unit in the APU. That is, any single key-value pair can only be accessed by one outstanding transaction, and the other related transactions will be buffered in the queue in the order of arrival. The concurrency control unit is a small hash table, and its entries are indexed by the key of the key-value pair. Key-value pairs are stored in the NVM and accessed by the address offset relative to the starting address, which is the same as HyperLoop. Also, it adds the functionality of chain-based communication across replica machines. The inter-machine communications still rely on ring buffers described in Sec. III-A, but the ring buffers are allocated in the NVM as the redo-log for failure recovery. One log entry (transaction) can contain multiple `(data, len, offset)` tuples, and the first byte of the log entry indicates the number of tuples. One exception is pure read transactions. Similar to HyperLoop, since the chain replication protocol already provides data consistency, a client can conduct a pure read transaction by directly accessing the chain's head/tail machine with one-sided RDMA read.

C. DLRM Inference

DLRMs have received much attention by Internet giants [2, 26, 38, 38, 58, 59, 81, 115] as they can offer more revenue and better user experience. In an end-to-end recommender system, the most expensive part of serving an inference request is the embedding reduction step, consuming huge memory capacity [115, 175, 176] and $\frac{1}{2} \sim \frac{3}{4}$ of the inference time [40, 59, 61, 81, 92]. The embedding reduction operation processes queries on a set of features. It finds a (sparse) embedding vector in the embedding table (a high-dimension matrix) and aggregates a value. The values of all features are assembled as the result. Also, the embedding reduction is bounded by memory bandwidth and exhibits poor data locality [81, 92, 172]. Last but not least, it also incorporates routines like request parsing and transforming (pre-processing),

TABLE II: Rambda testbed configurations.

2× **Intel® Xeon 6138P CPUs@2.0GHz [69]**	
20 Skylake cores, hyperthreading enabled, running Ubuntu 18	
27.5MB shared LLC	
Six DDR4-2666 channels, 192GB DRAM in total	
In-package Intel® Arria 10GX FPGA@400MHz [66]	
One UPI link to the CPU, 10.4GT/s (20.8GB/s)	
64KB local cache	
LUT usage	11K (26%)
Registers usage	130K (8%)
BRAM blocks usage	387 (14%)
NVIDIA BlueField-2 Smart NIC SoC [119]	
2x25Gbps Ethernet ports, backed by ConnectX-6 Dx controller	
RDMA over converged Ethernet V2 (RoCEv2)	
Eight ARM A72 cores@2.5GHz, running Ubuntu 20	
6MB shared LLC	
16GB on-board DDR4-1600 DRAM	
One-sided RDMA for ARM to access the host memory	

which are irregular and branch-rich. As such it is not suitable for hardware accelerators. These characteristics make it unsuitable, if not impossible, to be fully offloaded to any Smart NIC or cc-accelerator. In addition to acceleration with specialized hardware like in-memory processing [9, 81, 88, 128], MERCI [92] takes an algorithmic way to memoize sub-query grouped results to reduce memory pressure on the commodity server platform.

Different from Rambda KV and Rambda TX, we design Rambda DLRM as an example of *CPU-accelerator collaboration* for request processing. Upon receiving a request from a client, the cc-accelerator first goes through the RPC stack, and then passes the request to the CPU through the ring buffer, where the request is parsed and transformed to model-ready input. Now, the input (request) is passed again to the cc-accelerator's APU, where the full inference, especially embedding reduction, is done. Finally, the cc-accelerator sends the result (response) back to the client through the RNIC. Empirically, we observe that one CPU core with 60% usage can already keep up with the network and the cc-accelerator processing rate. In DLRM, not all memory accesses in a single query need to be serialized. Hence, in the APU, we issue 64 memory requests for each query's iteration so that the memory bandwidth can be fully utilized and the memory access latency can be hidden. Lastly, the ALU is enhanced to support various aggregation operators (*e.g.*, max/min/inner product).

V. Implementation and Methodology

We prototype Rambda with a system containing two Intel® Xeon Gold 6138P CPUs with an in-package FPGA. The configurations of the system are listed in Tab. II.

We implement a round-robin scheduler. The APU can support 256 outstanding requests. Each request buffer has 1024 entries. We adopt HERD's RPC protocol [76, 77] for its simplicity, but any advanced RPC stack could be applied [89]. The resource utilization numbers in Tab. II reflect our Rambda key-value store accelerator (Sec. IV). The utilization results for the other applications we have built are similar, because ∼80% of used resources go to the common components of the coherence controller and the local cache.

The current implementation has two major limitations. The first one is the performance of the coherence controller. As a soft design in the programmable fabric of the FPGA, it suffers from synthesis constraints and can perform at at most 400 MHz, incurring limited data access performance, which has also been observed by prior work [89]. However, its counterparts on a regular server CPU can operate at ∼2 GHz [1]. We expect such infrastructural parts can be fixed by hard IP in future FPGAs, offering comparable performance to that of CPUs [68]. The FPGA also lacks local memory in the same coherence domain or with comparable capacity to CPU-attached memory. Consequently, most application memory requests must go through the cc-interconnect (due to their large working sets), similar to cross-NUMA memory access. Additionally, the `cpoll` region must be pinned in the cc-accelerator's local cache. We expect these limitations would be lessened when Rambda is implemented in CXL-based devices [32, 54] or Enzian [30].

To explore the potential of Rambda's performance on future platforms we also use a stand-alone U280 FPGA card [164] with 32 GB DDR4 memory and 8 GB HBM2 to emulate a cc-accelerator with local coherent memory [30, 32, 54]. Prior work [158] has shown that these two types of memory can achieve ∼36 GB/s and ∼425 GB/s throughput, respectively. Specifically, we adapt the APU to the U280 card using either the DDR4 or HBM2 controller. The application data is mapped and initialized in the FPGA's local memory. Rather than interacting with a real RNIC, we emulate arrival of RDMA requests by generating requests within the FPGA matching the RDMA write rate measured on the testbed. We believe this emulation methodology offers correct and convincing results since coherence (of the application data) makes no big difference here after the data has been allocated and initialized in the FPGA-attached memory; during request processing, most memory traffic does not need to go across the cc-interconnect. For throughput experiments, we measure requests processed on the FPGA per second. For latency experiments, we compute the emulated end-to-end latency by combining application processing time measured on the U280 with the average latency of the rest of the stack. Specifically, we measure the average latency from a request's generation to its completion on the U280, then add the average full-system end-to-end latency without an APU, measured on the client machine. Note that this approach emulates average latency, so does not apply to tail latency measurements. In the following sections, we notate the U280 DDR4-based results as "Rambda-LD (local DDR4)" and HBM2-based results as "Rambda-LH (local HBM2)".

Lastly, we use the NVIDIA ConnectX-6-based BlueField-2 Smart NIC [119] as the RNIC. It also provides eight ARM cores, which we use to compare against a Smart NIC approach.

VI. Evaluation

A. Microbenchmark

To demonstrate the benefits of each Rambda component and avoid the network bandwidth as the bottleneck (as we will see

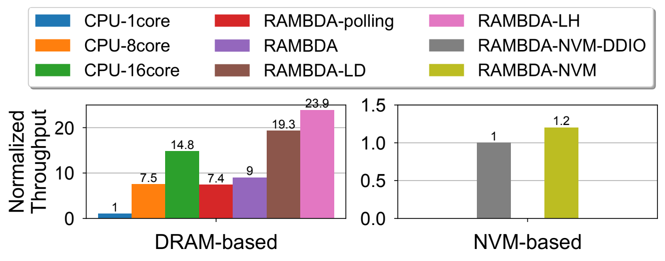

Fig. 7: Normalized throughput performance of different approaches on the microbenchmark.

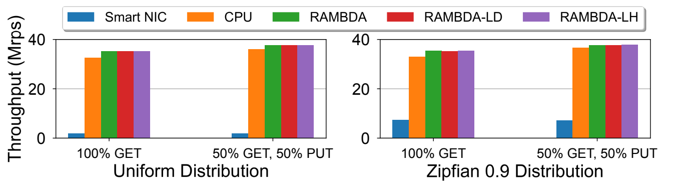

Fig. 8: Peak throughput performance of different KVS designs. The batch size of 32 is applied.

in the following sections), we first run a single-machine microbenchmark and analyze RAMBDA's performance against the highly-optimized software solutions on multiple cores. Specifically, we use the CPU cores on the other NUMA node in the server to feed requests to the CPU cores or RAMBDA accelerator on the local NUMA node via shared memory buffer (to emulate the one-sided RDMA behavior). For each request, the core or RAMBDA accelerator will randomly pick a node from a predefined and permuted 10M-node in-memory linked list, traverse the two succeeding nodes, and return the value in the second node. In CPU solutions, we use one, eight, and 16 cores to perform the microbenchmark, each with a batch size of 16 requests (optimized for throughput). In RAMBDA, we have a "RAMBDA-polling" variant that replaces `cpoll` with regular spin-polling to demonstrate `cpoll`'s benefit. Empirically, we choose 30 FPGA clock cycles as the spin-polling interval. Also, we have a "RAMBDA-DDIO" variant on the NVM-based experiment, which always turns DDIO on, to show the benefit of the proposed adaptive DDIO mechanism. All RAMBDA variants have 16 connections to the cores on the other NUMA node. Since the (Skylake) CPU with in-package FPGA does not support Intel Optane DIMMs, we emulate NVM's behavior by adding latency and throttling memory bandwidth in the FPGA and the ARM emulation program. We follow NVM's characteristics from recent Optane-based studies [74, 167] to calibrate our emulation.

Fig. 7 plots the throughput of different approaches. All DRAM-based results are normalized to single-core's throughout; and the NVM-based results are normalized to RAMBDA-DDIO. In the DRAM-based experiment, since the microbenchmark has little cross-thread contention, the CPU-based solutions scale almost linearly. On the other hand, RAMBDA-polling, limited by bandwidth of the cc-interconnect, is equivalent to ∼8 cores' performance. By reducing the latency and traffic of polling, RAMBDA with `cpoll` further improves the throughput by ∼21.6%. Having more local memory for better bandwidth and latency characteristics, RAMBDA-LD and RAMBDA-LH further remove the cc-interconnect in the original RAMBDA, bringing 114.4% ∼ 165.6% more improvement. Also, for the NVM case, reducing the write amplification and thus bandwidth utilization, adaptive DDIO mechanism can improve the throughput by ∼20%, aligned with the number reported in the prior work [74, 162].

B. In-Memory Key-Value Store

Due to the limited availability of devices, we run RAMBDA KV on one server and one client. We compare RAMBDA with two state-of-the-art baselines: highly-optimized open-source two-sided RDMA-RPC (MICA-backed) [76, 77] (denoted "*CPU*") and *Smart NIC* [94, 145]. For *CPU*, we use ten threads (cores) on the testbed to maximize KVS throughput. Each thread is fed with requests by one client instance (each with two dedicated Skylake cores) on the client machine (also equipped with the BlueField-2 Smart NIC). For RAMBDA, we also use 10 client instances to feed requests that are processed on the same RAMBDA accelerator. For *Smart NIC*, we use the Smart NIC's eight ARM cores to emulate the behavior of the specialized hardware in KV-Direct [94] and StRoM [145]. The ARM cores process the request, which is sent from the client Intel CPU by RDMA. The ARM cores retrieve necessary data from their server host through RDMA. ARM cores are not as efficient as specialized FPGA designs [94, 145] when processing KVS and accessing host memory, but the ARM cores' frequency is ∼ 10× higher, alleviating the efficiency gap. When running KVS entirely on the Smart NIC's on-board DRAM, we measure the eight ARM cores' peak throughput to match that of six Intel CPU cores.

We pre-load 100M key-value pairs (64 B size, ∼7 GB memory in total) and then access them using uniform and Zipfian 0.9 distributions. We test two types of workloads: read-intensive (100% GET), and write-intensive (50% GET, 50% PUT). The MICA-based mechanism [76, 77] eliminates concurrency issues (*e.g.*, only allowing the "owner core" to read/write the data partition). As such, the performance of a heavy PUT workload differs little from the GET-only workload, matching results of prior work [77, 94].

In *CPU* and RAMBDA, both the hash table and key-value pairs are stored in the host memory; in *Smart NIC*, we allocate 512 MB of the Smart NIC's on-board DRAM as a cache of the most recently accessed hash entries and key-value pairs. The cache-total ratio (512 MB : 7 GB) is roughly the memory capacity ratio (16 GB : 192 GB). We also test the impact of batching: in *CPU* and *Smart NIC*, we process requests in a batch to improve memory access efficiency [98]. In RAMBDA, since the APU can already exploit the memory-level parallelism across requests [85, 103, 168, 171], request batching is not needed; instead, we batch the doorbell signals to the RNIC [77] when posting RDMA operations for response. These configurations resemble prior work [77, 89, 94, 98].

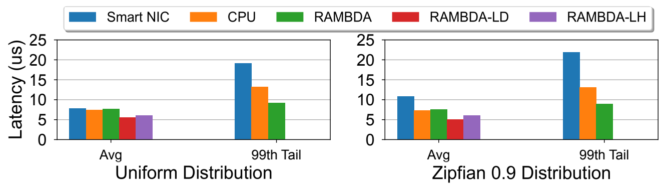

Fig. 9: Latency performance of different KVS designs on the 100% GET workload. The batch size of 32 is applied. RAMBDA-LD/LH's tail latency is inapplicable.

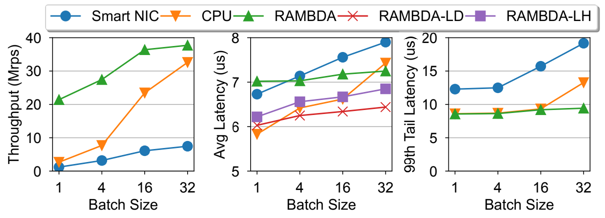

Fig. 10: The impact of batch size on throughput and latency (100% GET workload, Zipfian 0.9 distribution). RAMBDA-LD/LH's tail latency is inapplicable.

We first show each design's peak performance (with batch size 32). Fig. 8 shows throughput; we find that the request distribution significantly affects *Smart NIC*'s performance. With a uniform distribution (*i.e.*, more than 90% memory accesses are to the host via PCIe), throughput is 27.2%–28.6% of that with a skewed Zipfian distribution (*i.e.*, most memory accesses are local). And even the throughput with Zipfian distribution is only ∼60% of that with pure on-board memory accesses. Conversely, the distribution does not affect *CPU* and RAMBDA's performance, since even with the Zipfian distribution, the KVS's memory footprint is larger than the CPU or FPGA's cache. Second, we observe that RAMBDA's peak throughput is 2.3%∼8.3% higher than *CPU*. This is because the peak KVS throughput is ***bounded by the network bandwidth*** now, and RAMBDA's one-sided RDMA performs a little better than *CPU*'s two-sided RDMA, which is aligned with prior studies [75, 118]. The throughput of RAMBDA-LD and RAMBDA-LH demonstrate this as well – extra memory bandwidth does not help improve performance (in fact, the UPI link is not saturated), since the network has reached its limit. Note that, having higher network bandwidth, using more CPU cores can still saturate it, as long as there is little dependency/contention across requests. And *CPU*'s throughput may still be similar to RAMBDA's, as RAMBDA accelerator is not saturated now. So we believe the current data under 25GbE can faithfully reflect *CPU* and RAMBDA's difference.

Regarding latency, taking the 100% GET workload as an example (Fig. 9), *Smart NIC*'s performance is again affected by the request distribution. The PCIe link adds significant latency even if the accesses are batched. Meanwhile, we observe that RAMBDA's average latency is a bit higher than *CPU*. This is mainly because, unlike *CPU*, RAMBDA needs to access data through the UPI link, adding more time on the request processing critical path. This deficiency is overcome with RAMBDA-LD/LH's accelerator-attached memory – it only goes through the UPI to interact with the RNIC. Note that due to HBM's nature, RAMBDA-LH has a higher average latency than RAMBDA-LD since the workload is no longer bounded by memory bandwidth. For tail latency, RAMBDA is 52.0% lower than *Smart NIC* and 30.1% lower than *CPU*, because it not only mitigates PCIe overhead, but also has more stable behavior than the CPU core, whose performance is affected by factors like OS scheduling and CPU resource contention.

TABLE III: Overall power efficiency of different KVS approaches with GET operations in uniform distribution.

	CPU	*Smart NIC*	RAMBDA
Kop/W	130.4	25.2	188.7

We also investigate the impact of the batch size on each design and demonstrate the results in Fig. 10 (since the KVS throughput is now network-bound, we do not include RAMBDA-LD/LH's throughput as they are the same as RAMBDA). For *CPU* and *Smart NIC*, batching can significantly improve throughput (*e.g.*, $\sim 12\times$) – by batching the data accesses across requests, the memory bandwidth is efficiently utilized and the memory/PCIe latency is hidden. RAMBDA's throughput also benefits from batching (*i.e.*, $\sim 2\times$) by reducing MMIO-based doorbell access [48, 77] and surrounding `sfence` signals from the RAMBDA cc-accelerator, which is relatively expensive. On the other hand, unlike *CPU* and *Smart NIC*, RAMBDA's latency sub-linearly increases with batch size. This is because RAMBDA does not need to wait for the batch size of arrived requests to start processing, and the RNIC may execute the WQE promptly before the doorbell is rung [106].

Finally, we use the Intel RAPL interface [127] (for CPU and DIMMs), IPMI tool (for the entire server box), and the FPGA's firmware (for the FPGA chip) to measure power consumption. Take the case in Fig. 8 as an example. We find that the *CPU* and *Smart NIC*'s Intel/ARM CPUs consume ∼90 Watts and ∼15 Watts respectively when fully loaded, while RAMBDA's FPGA power is in the range of 24–27W to achieve peak throughput. This demonstrates RAMBDA cc-accelerator's $\sim 3\times$ power efficiency over the beefy Intel CPU to achieve comparable performance; although the absolute power value of RAMBDA is $\sim 2\times$ higher than the Smart NIC, it still has a significant op per watt advantage, as demonstrated in Tab. III. All these lead to ∼38% power consumption reduction of the entire server box.

C. Distributed Transaction with NVM-based Chain Replication

We compare RAMBDA with HyperLoop [83] since their results show that their mechanism outperforms CPU-based chain replication, especially in multi-tenant cloud environments. Following HyperLoop's example, we adopt RocksDB [44], a persistent key-value database, to use the emulated NVM (see Sec. VI-A) as a persistent storage medium for RAMBDA and HyperLoop. Since HyperLoop modifies the RNIC's firmware, which is not open-source, we use the ARM cores on the Smart NIC to emulate its behavior. We disable DDIO on the server.

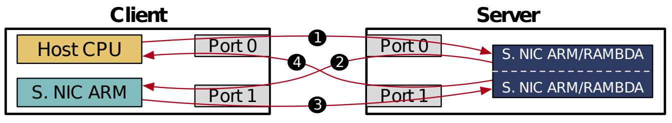

Fig. 11: Emulated 2-node replication; the Smart NIC (S. NIC in the figure) ARM in the client simply forwards data from port 0 to port 1, and the Smart NIC ARM/RAMBDA accelerator is separated to handle traffic from port 0 and 1 independently.

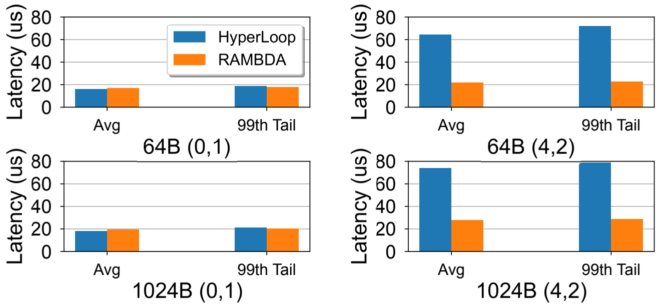

Fig. 12: Latency comparison with different key-value pair size and transactions with different numbers of *(read,write)*.

As with previous experiments, we run the experiments on one client and one server. We use the two ports on the Smart NIC to have two replica machines (instances) in the same physical server; transactions are forwarded between the ports, as shown in Fig. 11. The client's host CPU initiates a transaction and sends it to the server's port 0 (❶). The corresponding processing unit (either a Smart NIC ARM core or RAMBDA) forwards the transaction to the client's Smart NIC ARM via port 0, which is attached to a RocksDB instance (❷). The client's Smart NIC ARM routes the transaction to the server's port 1, where another RocksDB instance (and the corresponding processing unit) serves as the second replica (❸). Finally, the transaction is sent back to the client's host CPU (❹). Our measurements show that the ARM-based routing adds 2∼3μs overhead, which resembles datacenter network latencies.

We initiate RocksDB with 100K key-value pairs and issue 100K transactions from the client to measure end-to-end latency. We test two key-value pair sizes (64B and 1024B) and two types of transactions with different *(read, write)* counts (*(0,1) and (4,2)*, representative of real-world transactional systems [78]). Since the RAMBDA Tx and HyperLoop have identical mechanisms for pure-read transactions, we exclude such transactions from the evaluation.

We show latency results in Fig. 12 (note that since the transactions are issued by the client one by one, the latency improvement also reflects throughput improvements). For the *(0,1)* transaction, RAMBDA's performance does not differ from HyperLoop, since they experience the same overhead – one PCIe round-trip per replica machine and one round-trip over the 2-machine replication chain; RAMBDA may even be a bit (less than 3%) slower than HyperLoop since it also has the overhead of UPI link. However, when the transaction contains multiple operations, RAMBDA begins to show its advantage. Unlike HyperLoop, RAMBDA's client only needs to issue one combined transaction request to the replication chain, and the RAMBDA accelerator will handle the transaction execution and chain replication protocol itself in a near-data manner – still one PCIe round-trip per machine and one round-trip over the chain. This saves network and PCIe latency, offering a 63.2%∼66.8% reduction for average end-to-end latency and 64.5%∼69.1% for 99th tail latency. Note that RAMBDA's latency can be further reduced with accelerators (FPGA) directly attached to NVM [68].

D. DLRM Inference

We base our experiments on MERCI [92] and facebook's DLRM model [115]. Since their open-source ones include single-machine versions only, we extend them to an RDMA-based end-to-end application (the networking part is similar to the optimized HERD [77]) to reflect a real deployment.

We follow the configurations in the MERCI paper [92] for our evaluation. We compare RAMBDA with the CPU-based software version on the popular Amazon Review dataset [60] (electronics, clothing-shoe-jewelry, home-kitchen, books, sports-outdoors, and office-products). Both the native embedding reduction and MERCI reduction are evaluated. The data is clustered and loaded into embedding and memoization tables by the CPU in MERCI's way. The embedding dimension is set to the default value of 64; for MERCI reduction, we build memoization tables with 0.25× the size of the original embedding tables.

Since query lengths (number of features) in the dataset are diverse, it is unfair to measure the per-query latency, so we only measure the throughput. For brevity, we only show the results of inference with MERCI reduction in Fig. 13 because the ones with native reduction show the same trend. The CPU-based version of MERCI scales linearly until eight cores (threads), which is bounded by the host memory bandwidth. For RAMBDA, however, we find poor throughput performance over all the datasets – only 19.7%∼31.3% throughput of a single CPU core. After further analysis, we find this phenomenon is because (1) unlike KVS, the nature of the embedding reduction in DLRM (*i.e.*, pure and dense memory accesses within nested "for" loops, few branches, thousands of memory accesses per query) makes it relatively efficient on the CPU core – the instruction window and the load-store queue can be fully utilized; (2) the CPU core can leverage the entire bandwidth of the memory channels (*i.e.*, ∼120GB/s on our testbed) with good parallelism, while the RAMBDA cc-accelerator can only leverage the cc-interconnect's bandwidth, and the memory requests have to be issued serially from the FPGA's wimpy coherence controller (also observed in [42]); (3) the compute-intensive fully connected layer is relatively lightweight in the model, making RAMBDA's hardware acceleration only a small portion in the end-to-end inference. Hence, with higher frequency and memory bandwidth, CPU outperforms the RAMBDA cc-accelerator. This deficiency can be solved by RAMBDA-LD and RAMBDA-LH, the future cc-accelerator we envision. Fully utilizing the two DDR4 channels ($\sim \frac{1}{3}$ of the CPU memory channels' bandwidth), RAMBDA-LD is able to achieve 52.8%∼95.3% throughput of the eight CPU cores. Furthermore, the 32-channel HBM2 eliminates

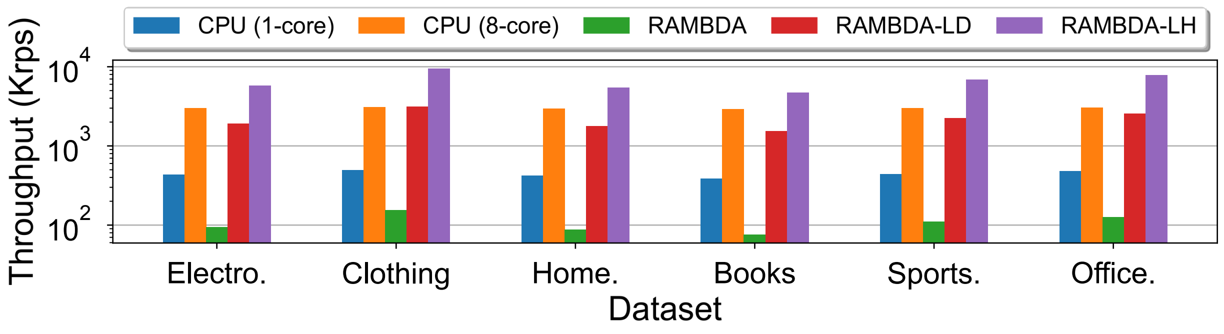

Fig. 13: MERCI-based DLRM inference throughput on the Amazon Review dataset.

the memory bandwidth bottleneck in the reduction, leading to RAMBDA-LH's 1.6×∼3.1× throughput improvement over the CPU [172], and the RDMA network becomes the limiting factor for higher end-to-end throughput. Note that, CPUs with integrated HBM2 [31] in the near future may also achieve similar throughput compared to RAMBDA-LH; but similar to KVS (see Sec. VI-B), even with the same memory bandwidth (and thus the inference throughput), RAMBDA cc-accelerator shows better power efficiency over the CPU-based reduction.

VII. Related Work

Accelerating μs-scale datacenter applications with emerging devices. In modern datacenters, commodity networking devices, especially programmable switches and Smart NICs, have been leveraged to accelerate datacenter applications. Such approaches include caching [73, 94, 101, 142], compute offloading [7, 17, 57, 93, 94, 107, 139, 140, 145], protocol offloading [36, 37, 72, 83, 91, 95–97, 134, 141, 157, 169, 179], *etc*. However, due to the limited memory capacity of such devices (*e.g.*, *O(10MB)* of on-chip SRAM for programmable switches [1, 82] and *O(10GB)* of on-board DRAM for Smart NICs [100, 119]), they may fall short of efficiently handling datacenter applications requiring large memory footprints. Also, extending the memory space to the host side [82, 173] can only handle simple/customized data structures.

To alleviate interconnect (PCIe) overhead, some research integrates the entire NIC or accelerator to/near the CPU package [3, 23, 34, 35, 56, 62, 80, 99, 117, 135, 149]. While this provides fast NIC-core interaction, it also has (1) high design and manufacturing cost and (2) low flexibility. For example, a NIC ASIC's die area can be more than $60mm^2$ [1], which is roughly the area of four server CPU cores [163]. But network upgrades would require replacing server CPUs too, leading to high total cost of ownership. Dagger [89] also leverages cc-accelerator for NIC design, but still involves the server CPU for request processing, and only uses the cc-interconnect for lower latency over PCIe.

Compared to these works, RAMBDA takes a modularized design with low TCO towards a framework for general memory-intensive distributed application acceleration, while still leveraging cache coherence to more efficiently handle applications with irregular/uniform memory access patterns. Also, RAMBDA is not mutual-exclusive with these works, as our goal is to make the most out of each component in the system.

Some prior work accelerates applications in datacenters with cc-accelerators [21, 42, 61, 89, 104, 126, 144], but they either accelerate single-machine applications/operations or a specific routine/layer in the system. RAMBDA takes a holistic cross-stack approach to achieve end-to-end datacenter application offloading at μs-scale.

NIC-host co-design framework. With the growing popularity of Smart NICs, researchers have developed frameworks to schedule/offload datacenter applications to their processors [100, 102, 130]. However, they are still constrained to separate the Smart NIC and host's memory to two domains and memory-intensive code suffers from high PCIe latency when offloaded. RAMBDA tackles these challenges with its unique near-data processing capability, while keeping the networking part offloaded on the RNIC for low cost and flexibility. Lynx [154] proposes Smart NIC-based communication offloading for accelerator-rich systems, and FlexDriver [41] proposes PCIe-based NIC control by accelerators. RAMBDA takes one step further to let the client directly communicate with the accelerators.

VIII. Conclusion

We present RAMBDA, a holistic system design to offload modern μs-scale datacenter applications. It leverages the RDMA and cc-accelerator technologies to achieve high throughput and low latency performance with minimal CPU involvement. We apply RAMBDA to three representative datacenter applications. Our evaluation on a real system shows that, compared with CPU- and NIC-based offloading solutions, RAMBDA achieves better and more stable performance with higher power efficiency.

Acknowledgment

We would like to thank Narayan Ranganathan, Nikhil Rao, Rajesh Sankaran, as well as the anonymous reviewers for their insightful and helpful information and feedback. We also thank Yixiao Gao, Nikita Lazarev, Jiacheng Ma, Shaojie Xiang, and Zeran Zhu for their technical support and discussion. This research is supported by National Science Foundation (No. CNS-1705047), Intel Corporation's Academic research funding, and NVIDIA device donation. Nam Sung Kim has a financial interest in Samsung.

References

[1] "Private communication with Intel," 2021.

[2] B. Acun, M. Murphy, X. Wang, J. Nie, C.-J. Wu, and K. Hazelwood, "Understanding training efficiency of deep learning recommendation models at scale," in *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*, 2021.

[3] M. Alian and N. S. Kim, "NetDIMM: Low-latency near-memory network interface architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'19)*, 2019.

[4] M. Alian, Y. Yuan, J. Zhang, R. Wang, M. Jung, and N. S. Kim, "Data direct I/O characterization for future I/O system exploration," in *Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'20)*, 2020.

[5] S. Almeida, J. a. Leitão, and L. Rodrigues, "Chainreaction: A causal+ consistent datastore based on chain replication," in *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys'13)*, 2013.

[6] M. Alshboul, P. Ramrakhyani, W. Wang, J. Tuck, and Y. Solihin, "BBB: Simplifying persistent programming using battery-backed buffers," in *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*, 2021.

[7] E. Amaro, Z. Luo, A. Ousterhout, A. Krishnamurthy, A. Panda, S. Ratnasamy, and S. Shenker, "Remote memory calls," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets'20)*, 2020.

[8] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: A fast array of wimpy nodes," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*, 2009.

[9] B. Asgari, R. Hadidi, J. Cao, D. E. Shim, S.-K. Lim, and H. Kim, "FAFNIR: Accelerating sparse gathering by using efficient near-memory intelligent reduction," in *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*, 2021.

[10] AWS, "Elastic Fabric Adapter – Run HPC and ML applications at scale," https://aws.amazon.com/hpc/efa/.

[11] G. Bablani, "Introducing new product innovations for SAP HANA, expanded AI collaboration with SAP and more," https://azure.microsoft.com/en-us/blog/introducing-new-product-innovations-for-sap-hana-expanded-ai-collaboration-with-sap-and-more/, 2019.

[12] M. Balakrishnan, D. Malkhi, V. Prabhakaran, T. Wobbler, M. Wei, and J. D. Davis, "CORFU: A shared log design for flash clusters," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, 2012.

[13] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Communications of the ACM*, vol. 60, no. 4, 2017.

[14] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, 2007.

[15] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion, "IX: A protected dataplane operating system for high throughput and low latency," in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*, 2014.

[16] S. Blanas, "From FLOPS to IOPS: The new bottlenecks of scientific computing," https://www.sigarch.org/from-flops-to-iops-the-new-bottlenecks-of-scientific-computing/, 2020.

[17] M. Blott, K. Karras, L. Liu, K. Vissers, J. Bär, and Z. István, "Achieving 10Gbps line-rate key-value stores with FPGAs," in *Proceedings of the 5th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'13)*, 2013.

[18] N. Boden, "Available first on Google Cloud: Intel Optane DC persistent memory," https://cloud.google.com/blog/topics/partners/available-first-on-google-cloud-intel-optane-dc-persistent-memory, 2018.

[19] M. Burke, S. Dharanipragada, S. Joyner, A. Szekeres, J. Nelson, I. Zhang, and D. R. K. Ports, "PRISM: Rethinking the RDMA interface for distributed systems," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*, 2021.

[20] Q. Cai, S. Chaudhary, M. Vuppalapati, J. Hwang, and R. Agarwal, "Understanding host network stack overheads," in *Proceedings of the 2021 ACM SIGCOMM conference (SIGCOMM'21)*, 2021.

[21] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*, 2021.

[22] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas, "Windows Azure storage: A highly available cloud storage service with strong consistency," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*, 2011.

[23] H. Caminal, Y. Chronis, T. Wu, J. M. Patel, and J. F. Martínez, "Accelerating database analytic query workloads using an associative processor," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA'22)*, 2022.

[24] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *Proceedings of the 49th IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*, 2016.

[25] CCIX Consortium, "CCIX," https://www.ccixconsortium.com.

[26] S. J. Chen, Z. Qin, Z. Wilson, B. Calaci, M. Rose, R. Evans, S. Abraham, D. Metzler, S. Tata, and M. Colagrosso, "Improving recommendation quality in Google Drive," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'20)*, 2020.

[27] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen, "Fast and general distributed transactions using RDMA and HTM," in *Proceedings of the 11th European Conference on Computer Systems (EuroSys'16)*, 2016.

[28] Y. Chen, Y. Lu, and J. Shu, "Scalable RDMA RPC on reliable connection with efficient resource sharing," in *Proceedings of the 14th EuroSys Conference (EuroSys'19)*, 2019.

[29] Y.-K. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "In-depth analysis on microarchitectures of modern heterogeneous CPU-FPGA platforms," *ACM Transactions on Reconfigurable Technology Systems*, vol. 12, no. 1, 2019.

[30] D. Cock, A. Ramdas, D. Schwyn, M. Giardino, A. Turowski, Z. He, N. Hossle, D. Korolija, M. Licciardello, K. Martsenko, R. Achermann, G. Alonso, and T. Roscoe, "Enzian: An open, general, CPU/FPGA platform for systems software research," in *Proceedings of the 27th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, 2022.

[31] I. Cutress, "Intel to Launch Next-Gen Sapphire Rapids Xeon with High Bandwidth Memory," https://www.anandtech.com/show/16795/intel-to-launch-next-gen-sapphire-rapids-xeon-with-high-bandwidth-memory, 2021.

[32] I. Cutress, "Using a PCIe slot to install DRAM: New Samsung CXL.mem expansion module," https://www.anandtech.com/show/16670/using-a-pcie-slot-to-install-dram-new-samsung-cxlmem-expansion-module, 2021.

[33] CXL Consortium, "Compute express link (CXL)," https://www.computeexpresslink.org.

[34] A. Daglis, S. Novaković, E. Bugnion, B. Falsafi, and B. Grot, "Manycore network interfaces for in-memory rack-scale computing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*, 2015.

[35] A. Daglis, M. Sutherland, and B. Falsafi, "RPCValet: NI-driven tail-aware balancing of µs-scale RPCs," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*, 2019.

[36] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, H. Weatherspoon, M. Canini, F. Pedone, and R. Soulé, "Network hardware-accelerated consensus," Università della Svizzera italiana, Tech. Rep. USI-INF-TR-2016-03, 2016.

[37] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, H. Weatherspoon, M. Canini, N. Zilberman, F. Pedone, and R. Soulé, "P4xos: Consensus as a network service," Università della Svizzera italiana, Tech. Rep. USI-INF-TR-2018-01, 2018.

[38] J. Dean, D. Patterson, and C. Young, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE*

Micro, vol. 38, 2018.

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

[40] A. Eisenman, M. Naumov, D. Gardner, M. Smelyanskiy, S. Pupyrev, K. Hazelwood, A. Cidon, and S. Katti, "Bandana: Using non-volatile memory for storing deep learning models," in *Proceedings of the 2nd SysML Conference (SysML'19)*, 2019.

[41] H. Eran, M. Fudim, G. Malka, G. Shalom, N. Cohen, A. Hermony, D. Levi, L. Liss, and M. Silberstein, "FlexDriver: A network driver for your accelerator," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, 2022.

[42] Z. F. Eryilmaz, A. Kakaraparthy, J. M. Patel, R. Sen, and K. Park, "FPGA for aggregate processing: The good, the bad, and the ugly," in *Proceedings of the IEEE 37th International Conference on Data Engineering (ICDE'21)*, 2021.

[43] R. Escriva, B. Wong, and E. G. Sirer, "HyperDex: A distributed, searchable key-value store," in *Proceedings of the ACM SIGCOMM 2012 Conference (SIGCOMM'12)*, 2012.

[44] Facebook, "RocksDB: A persistent key-value store for fast storage environments," https://rocksdb.org.

[45] B. Falsafi, R. Guerraoui, J. Picorel, and V. Trigonakis, "Unlocking energy," in *Proceedings of the 2016 USENIX Annual Technical Conference (ATC'16)*, 2016.

[46] A. Farshin, A. Roozbeh, G. Q. Maguire Jr., and D. Kostić, "Make the most out of last level cache in Intel processors," in *Proceedings of the 14th European Conference on Computer Systems (EuroSys'19)*, 2019.

[47] A. Farshin, A. Roozbeh, G. Q. Maguire Jr., and D. Kostić, "Reexamining direct cache access to optimize I/O intensive applications for multi-hundred-gigabit networks," in *Proceedings of 2020 USENIX Annual Technical Conference (ATC'20)*, 2020.

[48] M. Flajslik and M. Rosenblum, "Network interface design for low latency request-response protocols," in *Proceedings of the 2013 USENIX Annual Technical Conference (ATC'13)*, 2013.

[49] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan, F. Feng, Y. Zhuang, F. Liu, P. Liu, X. Liu, Z. Wu, J. Wu, Z. Cao, C. Tian, J. Wu, J. Zhu, H. Wang, D. Cai, and J. Wu, "When cloud storage meets RDMA," in *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI'21)*, 2021.

[50] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[51] S. D. Glasser and M. D. Hummel, "Methods and apparatus for implementing PCI express lightweight notification protocols in a CPU/memory complex," 2013, US Patent 20130173837A1.

[52] H. Golestani, A. Mirhosseini, and T. F. Wenisch, "Software data planes: You can't always spin to win," in *Proceedings of the 2019 ACM Symposium on Cloud Computing (SoCC'19)*, 2019.

[53] D. Gope, D. J. Schlais, and M. H. Lipasti, "Architectural support for server-side PHP processing," in *Proceedings of the 44th IEEE/ACM International Symposium on Computer Architecture (ISCA'17)*, 2017.

[54] D. Gouk, S. Lee, M. Kwon, and M. Jung, "Direct access, High-Performance memory disaggregation with DirectCXL," in *Proceedings of the 2022 USENIX Annual Technical Conference (ATC'22)*, 2022.

[55] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "RDMA over commodity Ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM'16)*, 2016.

[56] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: A hardware-software co-designed disaggregated memory system," in *Proceedings of the 27th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, 2022.

[57] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proceedings of the 2018 ACM SIGCOMM Conference (SIGCOMM'18)*, 2018.

[58] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*, 2020.

[59] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia, H.-H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, "The architectural implications of Facebook's DNN-based personalized recommendation," in *Proceedings of the 2020 IEEE International Symposium on High-Performance Computer Architecture (HPCA'20)*, 2020.

[60] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *Proceedings of the 25th International Conference on World Wide Web (WWW'16)*, 2016.

[61] R. Hwang, T. Kim, Y. Kwon, and M. Rhu, "Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*, 2020.

[62] S. Ibanez, A. Mallery, S. Arslan, T. Jepsen, M. Shahbaz, C. Kim, and N. McKeown, "The nanoPU: A nanosecond network stack for datacenters," in *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*, 2021.

[63] Intel Corporation, "Data plane development kit (DPDK)," https://www.dpdk.org.

[64] Intel Corporation, "eADR: New opportunities for persistent memory applications," https://software.intel.com/content/www/us/en/develop/articles/eadr-new-opportunities-for-persistent-memory-applications.html.

[65] Intel Corporation, "Intel 64 and IA-32 architectures software developer's manual volume 2: Instruction set reference," https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-2a-2b-2c-and-2d-instruction-set-reference-a-z.html.

[66] Intel Corporation, "Intel Arria 10 GX 1150 FPGA," https://ark.intel.com/content/www/us/en/ark/products/210381/intel-arria-10-gx-1150-fpga.html.

[67] Intel Corporation, "Intel Optane Persistent Memory," https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.

[68] Intel Corporation, "Intel Stratix 10 DX FPGAs," https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10/dx.html.

[69] Intel Corporation, "Intel Xeon Gold 6138P Processor," https://ark.intel.com/content/www/us/en/ark/products/139940/intel-xeon-gold-6138p-processor-27-5m-cache-2-00-ghz.html.

[70] Intel Corporation, "Intel® data direct I/O (DDIO)," https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html.

[71] E. Jeong, S. Woo, M. A. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park, "mTCP: A highly scalable user-level TCP stack for multicore systems," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*, 2014.

[72] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica, "NetChain: Scale-free sub-RTT coordination," in *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, 2018.

[73] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "NetCache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP'17)*, 2017.

[74] A. Kalia, D. Andersen, and M. Kaminsky, "Challenges and solutions for fast remote persistent memory access," in *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC'20)*, 2020.

[75] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*, 2019.

[76] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *Proceedings of the 2014 ACM SIGCOMM Conference (SIGCOMM'14)*, 2014.

[77] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *Proceedings of the 2016 USENIX Annual Technical Conference (ATC'16)*, 2016.

[78] A. Kalia, M. Kaminsky, and D. G. Andersen, "FaSST: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, 2016.

[79] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proceedings of the 42nd IEEE/ACM International Symposium on Computer Architecture (ISCA'15)*, 2015.

[80] S. Karandikar, C. Leary, C. Kennelly, J. Zhao, D. Parimi, B. Nikolic,

K. Asanovic, and P. Ranganathan, "A hardware accelerator for protocol buffers," in *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'21)*, 2021.

[81] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C.-J. Wu, M. Hempstead, and X. Zhang, "RecNMP: Accelerating personalized recommendation with near-memory processing," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*, 2020.

[82] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "TEA: Enabling state-intensive network functions on programmable switches," in *Proceedings of the 2020 ACM SIGCOMM Conference (SIGCOMM'20)*, 2020.

[83] D. Kim, A. Memaripour, A. Badam, Y. Zhu, H. H. Liu, J. Padhye, S. Raindel, S. Swanson, V. Sekar, and S. Seshan, "HyperLoop: Group-based NIC-offloading to accelerate replicated transactions in multi-tenant storage systems," in *Proceedings of the 2018 ACM SIGCOMM conference (SIGCOMM'18)*, 2018.

[84] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostić, Y. Kwon, S. Peter, and E. Witchel, "LineFS: Efficient SmartNIC offload of a distributed file system with pipeline parallelism," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*, 2021.

[85] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, "Meet the walkers: Accelerating index traversals for in-memory databases," in *Proceedings of the 46th IEEE/ACM International Symposium on Microarchitecture (MICRO'13)*, 2013.

[86] H. T. Kung, T. Blackwell, and A. Chapman, "Credit-based flow control for ATM networks: Credit update protocol, adaptive credit allocation and statistical multiplexing," in *Proceedings of the 1994 ACM SIGCOMM Conference (SIGCOMM'94)*, 1994.

[87] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi, "NetCAT: Practical cache attacks from the network," in *Proceedings of the 41st IEEE Symposium on Security and Privacy (Oakland'20)*, 2020.

[88] Y. Kwon, Y. Lee, and M. Rhu, "TensorDIMM: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'19)*, 2019.

[89] N. Lazarev, S. Xiang, N. Adit, Z. Zhang, and C. Delimitrou, "Dagger: Efficient and fast RPCs in cloud microservices with near-memory reconfigurable NICs," in *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*, 2021.

[90] Y. Le, M. Malekpourshahraki, B. Stephens, A. Akella, and M. M. Swift, "On the impact of cluster configuration on RoCE application design," in *Proceedings of the 3rd Asia-Pacific Workshop on Networking (APNet'19)*, 2019.

[91] S.-s. Lee, Y. Yu, Y. Tang, A. Khandelwal, L. Zhong, and A. Bhattacharjee, "MIND: In-network memory management for disaggregated data centers," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*, 2021.

[92] Y. Lee, S. H. Seo, H. Choi, H. U. Sul, S. Kim, J. W. Lee, and T. J. Ham, "MERCI: Efficient embedding reduction on commodity hardware via sub-query memoization," in *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*, 2021.

[93] A. Lerner, R. Hussein, and P. Cudre-Mauroux, "The case for network accelerated query processing," in *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research (CIDR'19)*, 2019.

[94] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang, "KV-Direct: High-performance in-memory key-value store with programmable NIC," in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17)*, 2017.

[95] J. Li, E. Michael, and D. R. K. Ports, "Eris: Coordination-free consistent transactions using in-network concurrency control," in *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP'17)*, 2017.

[96] J. Li, E. Michael, A. Szekeres, N. K. Sharma, and D. R. K. Ports, "Just say NO to Paxos overhead: Replacing consensus with network ordering," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, 2016.

[97] J. Li, J. Nelson, E. Michael, X. Jin, and D. R. K. Ports, "Pegasus: Tolerating skewed workloads in distributed storage with in-network coherence directories," in *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, 2020.

[98] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A holistic approach to fast in-memory key-value storage," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*, 2014.

[99] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, "Thin servers with smart pipes: Designing SoC accelerators for Memcached," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*, 2013.

[100] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto SmartNICs using iPipe," in *Proceedings of the 2019 ACM SIGCOMM conference (SIGCOMM'19)*, 2019.

[101] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya, "IncBricks: Toward in-network computation with an in-network cache," in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*, 2017.

[102] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3: Energy-efficient microservices on SmartNIC-accelerated servers," in *Proceedings of the 2019 USENIX Annual Technical Conference (ATC'19)*, 2019.

[103] E. Lockerman, A. Feldmann, M. Bakhshalipour, A. Stanescu, S. Gupta, D. Sanchez, and N. Beckmann, "Livia: Data-centric computing throughout the memory hierarchy," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020.

[104] J. Ma, G. Zuo, K. Loughlin, X. Cheng, Y. Liu, A. M. Eneyew, Z. Qi, and B. Kasikci, "A hypervisor for shared-memory FPGA platforms," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020.

[105] A. Manousis, R. A. Sharma, V. Sekar, and J. Sherry, "Contention-aware performance prediction for virtualized network functions," in *Proceedings of the 2020 ACM SIGCOMM Conference (SIGCOMM'20)*, 2020.

[106] Mellanox, "Mellanox adapters programmer's reference manual (PRM)," https://www.mellanox.com/related-docs/user_manuals/Ethernet_Adapters_Programming_Manual.pdf.

[107] Mellanox, "Mellanox scalable hierarchical aggregation and reduction protocol (SHARP)," http://www.mellanox.com/page/products_dyn?product_family=261&mtag=sharp.

[108] A. Memaripour, A. Badam, A. Phanishayee, Y. Zhou, R. Alagappan, K. Strauss, and S. Swanson, "Atomic in-place updates for non-volatile main memories with Kamino-Tx," in *Proceedings of the 12th European Conference on Computer Systems (EuroSys'17)*, 2017.

[109] A. Mirhosseini, H. Golestani, and T. F. Wenisch, "HyperPlane: A scalable low-latency notification accelerator for software data planes," in *Proceedings of the 53rd IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*, 2020.

[110] A. Mirhosseini, A. Sriraman, and T. F. Wenisch, "Enhancing server efficiency in the face of killer microseconds," in *Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*, 2019.

[111] C. Mitchell, Y. Geng, and J. Li, "Using one-sided RDMA reads to build a fast, CPU-efficient key-value store," in *Proceedings of the 2013 USENIX Annual Technical Conference (ATC'13)*, 2013.

[112] C. Mitchell, K. Montgomery, L. Nelson, S. Sen, and J. Li, "Balancing CPU and network in the Cell distributed B-Tree store," in *Proceedings of the 2016 USENIX Annual Technical Conference (ATC'16)*, 2016.

[113] S. K. Monga, S. Kashyap, and C. Min, "Birds of a feather flock together: Scaling RDMA RPCs with Flock," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*, 2021.

[114] mongoDB, "mongoDB manual: Manage chained replication," https://docs.mongodb.com/manual/tutorial/manage-chained-replication/.

[115] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[116] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore, "Understanding PCIe performance for end host

networking," in *Proceedings of the 2018 ACM SIGCOMM Conference (SIGCOMM'18)*, 2018.

[117] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot, "Scale-out NUMA," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*, 2014.

[118] S. Novakovic, Y. Shan, A. Kolli, M. Cui, Y. Zhang, H. Eran, B. Pismenny, L. Liss, M. Wei, D. Tsafrir, and M. Aguilera, "Storm: A fast transactional dataplane for remote data structures," in *Proceedings of the 12th ACM International Conference on Systems and Storage (SYSTOR'19)*, 2019.

[119] NVIDIA Corporation, "NVIDIA BlueField-2 DPU: Data center infrastructure on a chip," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf.

[120] NVIDIA Corporation, "RDMA aware networks programming user manual," https://docs.mellanox.com/display/RDMAAwareProgrammingv17.

[121] NVIDIA Corporation, "NVIDIA extends data center infrastructure processing roadmap with BlueField-3," https://nvidianews.nvidia.com/news/nvidia-extends-data-center-infrastructure-processing-roadmap-with-bluefield-3, 2021.

[122] F. Oboril and M. B. Tahoori, "ExtraTime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level," in *Proceedings of the 2012 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'2012)*, 2012.

[123] H. Ohara, "Revisit DCA, PCIe TPH and DDIO," https://www.slideshare.net/hisaki/direct-cacheaccess, 2014.

[124] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in RAMCloud," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*, 2011.

[125] Optocrypto, "Intel Sapphire Rapids with HBM2E, CXL 1.1, and PCIe 5.0 by end of 2022," https://optocrypto.com/intel-sapphire-rapids-with-hbm2e-cxl-1-1-and-pcie-5-0-by-end-of-2022/.

[126] M. Owaida, D. Sidler, K. Kara, and G. Alonso, "Centaur: A framework for hybrid CPU-FPGA databases," in *Proceedings of the 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*, 2017.

[127] S. Pandruvada, "Running average power limit – RAPL," https://01.org/blogs/2014/running-average-power-limit-âĂŞ-rapl.

[128] J. Park, B. Kim, S. Yun, E. Lee, M. Rhu, and J. H. Ahn, "TRiM: Enhancing processor-memory interfaces with scalable tensor reduction in memory," in *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'21)*, 2021.

[129] A. Phanishayee, D. G. Andersen, H. Pucha, A. Povzner, and W. Belluomini, "Flex-KV: Enabling high-performance and flexible KV systems," in *Proceedings of the 2012 Workshop on Management of Big Data Systems (MBDS'12)*, 2012.

[130] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson, "Floem: A programming system for NIC-accelerated network applications," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*, 2018.

[131] B. Pismenny, L. Liss, A. Morrison, and D. Tsafrir, "The benefits of general-purpose on-NIC memory," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, 2022.

[132] PLDA, "Lightweight Notification," https://www.plda.com/pcie-glossary/lightweight-notification.

[133] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, F. Huici, and G. Siracusano, "FlowBlaze: Stateful packet processing in hardware," in *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*, 2019.

[134] D. R. K. Ports, J. Li, V. Liu, N. K. Sharma, and A. Krishnamurthy, "Designing distributed systems using approximate synchrony in datacenter networks," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*, 2015.

[135] A. Pourhabibi, M. Sutherland, A. Daglis, and B. Falsafi, "Cerebros: Evading the RPC tax in datacenters," in *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'21)*, 2021.

[136] W. Reda, M. Canini, D. Kostic, and S. Peter, "RDMA is Turing complete, we just did not know it yet!" in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*, 2022.

[137] L. Romanovsky, "mlx5dv – Linux manual page," https://man7.org/linux/man-pages/man7/mlx5dv.7.html.

[138] SAP, "How to perform system replication for SAP HANA," https://www.sap.com/documents/2013/10/26c02b58-5a7c-0010-82c7-eda71af511fa.html, 2017.

[139] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proceedings of the 16th Workshop on Hot Topics in Networks (HotNets'17)*, 2017.

[140] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," in *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI'21)*, 2021.

[141] H. N. Schuh, W. Liang, M. Liu, J. Nelson, and A. Krishnamurthy, "Xenic: SmartNIC-accelerated distributed transactions," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*, 2021.

[142] K. Seemakhupt, S. Liu, Y. Senevirathne, M. Shahbaz, and S. Khan, "PM-Net: In-network data persistence," in *Proceedings of the 48th IEEE/ACM International Symposium on Computer Architecture (ISCA'21)*, 2021.

[143] L. Shalev, H. Ayoub, N. Bshara, and E. Sabbag, "A cloud-optimized transport protocol for elastic and scalable HPC," *IEEE Micro*, vol. 40, no. 6, 2020.

[144] D. Sidler, Z. István, M. Owaida, and G. Alonso, "Accelerating pattern matching queries in hybrid CPU-FPGA architectures," in *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD'17)*, 2017.

[145] D. Sidler, Z. Wang, M. Chiosa, A. Kulkarni, and G. Alonso, "StRoM: Smart remote memory," in *Proceedings of the 15th European Conference on Computer Systems (EuroSys'20)*, 2020.

[146] A. Singhvi, A. Akella, D. Gibson, T. F. Wenisch, M. Wong-Chan, S. Clark, M. M. K. Martin, M. McLaren, P. Chandra, R. Cauble, H. M. G. Wassel, B. Montazeri, S. L. Sabato, J. Scherpelz, and A. Vahdat, "1RMA: Re-envisioning remote memory access for multi-tenant datacenters," in *Proceedings of the 2020 ACM SIGCOMM Conference (SIGCOMM'20)*, 2020.

[147] A. Sriraman and A. Dhanotia, "Accelerometer: Understanding acceleration opportunities for data center overheads at hyperscale," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020.

[148] J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel, "CAPI: A coherent accelerator processor interface," *IBM Journal of Research and Development*, vol. 59, no. 1, 2015.

[149] M. Sutherland, S. Gupta, B. Falsafi, V. Marathe, D. Pnevmatikatos, and A. Daglis, "The NeBuLa RPC-optimized architecture," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*, 2020.

[150] A. Tai, M. Wei, M. J. Freedman, I. Abraham, and D. Malkhi, "Replex: A scalable, highly available multi-index data store," in *Proceedings of the 2016 USENIX Annual Technical Conference (ATC'16)*, 2016.

[151] T. Talpey, "RDMA persistent meory extensions," https://www.openfabrics.org/wp-content/uploads/209_TTalpey.pdf, 2019.

[152] J. Terrace and M. J. Freedman, "Object storage on CRAQ: High-throughput chain replication for read-mostly workloads," in *Proceedings of the 2009 USENIX Annual Technical Conference (ATC'09)*, 2009.

[153] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, and S. Shenker, "ResQ: Enabling SLOs in network function virtualization," in *Proceedings of 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, 2018.

[154] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A SmartNIC-driven accelerator-centric architecture for network servers," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020.

[155] S.-Y. Tsai and Y. Zhang, "LITE kernel RDMA support for datacenter applications," in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17)*, 2017.

[156] R. van Renesse and F. B. Schneider, "Chain replication for supporting high throughput and availability," in *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI'04)*, 2004.

[157] Q. Wang, Y. Lu, E. Xu, J. Li, Y. Chen, and J. Shu, "Concordia:

Distributed shared memory with in-network cache coherence," in *Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST'21)*, 2021.

[158] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking high bandwidth memory on FPGAs," in *Proceedings of the 2020 IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'20)*, 2020.

[159] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng, T. Wang, W. Ling, K. Huo, P. An, K. Ji, S. Zhang, B. Xu, R. Feng, T. Ding, K. Chen, and C. Guo, "SRNIC: A scalable architecture for RDMA NICs," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)*, 2023.

[160] X. Wei, Z. Dong, R. Chen, and H. Chen, "Deconstructing RDMA-enabled distributed transactions: Hybrid is better!" in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*, 2018.

[161] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen, "Fast in-memory transaction processing using RDMA and HTM," in *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP'15)*, 2015.

[162] X. Wei, X. Xie, R. Chen, H. Chen, and B. Zang, "Characterizing and optimizing remote persistent memory with RDMA and NVM," in *Proceedings of the 2021 USENIX Annual Technical Conference (ATC'21)*, 2021.

[163] WikiChip, "Skylake (server – microarchitectures – intel," https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server).

[164] Xilinx, "Alveo U280 data center accelerator card," https://www.xilinx.com/products/boards-and-kits/alveo/u280.html.

[165] Xilinx, "Xilinx Virtex-7 FPGA VC709 connectivity kit," https://www.xilinx.com/products/boards-and-kits/dk-v7-vc709-g.html.

[166] J. Xue, M. U. Chaudhry, B. Vamanan, T. N. Vijaykumar, and M. Thottethodi, "Dart: Divide and specialize for fast response to congestion in RDMA-based datacenter networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, 2020.

[167] J. Yang, J. Kim, M. Hoseinzadeh, J. Izraelevitz, and S. Swanson, "An empirical guide to the behavior and use of scalable persistent memory," in *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*, 2020.

[168] C. Ye, Y. Xu, X. Shen, X. Liao, H. Jin, and Y. Solihin, "Hardware-based address-centric acceleration of key-value store," in *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*, 2021.

[169] Z. Yu, Y. Zhang, V. Braverman, M. Chowdhury, and X. Jin, "NetLock: Fast, centralized lock management using programmable switches," in *Proceedings of the 2020 ACM SIGCOMM Conference (SIGCOMM'20)*, 2020.

[170] Y. Yuan, M. Alian, Y. Wang, R. Wang, I. Kurakin, C. Tai, and N. S. Kim, "Don't forget the I/O when allocating your LLC," in *Proceedings of the 48th IEEE/ACM International Symposium on Computer Architecture (ISCA'21)*, 2021.

[171] Y. Yuan, Y. Wang, R. Wang, R. B. R. Chowhury, C. Tai, and N. S. Kim, "QEI: Query acceleration can be generic and efficient in the cloud," in *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*, 2021.

[172] C. Zeng, L. Luo, Q. Ning, Y. Han, Y. Jiang, D. Tang, Z. Wang, K. Chen, and C. Guo, "FAERY: An FPGA-accelerated embedding-based retrieval system," in *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)*, 2022.

[173] C. Zeng, L. Luo, T. Zhang, Z. Wang, L. Li, W. Han, N. Chen, L. Wan, L. Liu, Z. Ding, X. Geng, T. Feng, F. Ning, K. Chen, and C. Guo, "Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing," in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*, 2022.

[174] Y. Zha and J. Li, "Hetero-ViTAL: A virtualization stack for heterogeneous FPGA clusters," in *Proceedings of the 48th IEEE/ACM International Symposium on Computer Architecture (ISCA'21)*, 2021.

[175] W. Zhao, D. Xie, R. Jia, Y. Qian, R. Ding, M. Sun, and P. Li, "Distributed hierarchical GPU parameter server for massive scale deep learning ads systems," in *Proceedings of the 3rd Conference on Machine Learning ans Systems (MLSys'20)*, 2020.

[176] W. Zhao, J. Zhang, D. Xie, Y. Qian, R. Jia, and P. Li, "AIBox: CTR prediction model training on a single node," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM'19)*, 2019.

[177] D. Zhou, B. Fan, H. Lim, M. Kaminsky, and D. G. Andersen, "Scalable, high performance Ethernet forwarding with CuckooSwitch," in *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT'13)*, 2013.

[178] S. Zhou and S. Mu, "Fault-tolerant replication with pull-based consensus in MongoDB," in *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI'21)*, 2021.

[179] H. Zhu, Z. Bai, J. Li, E. Michael, D. R. K. Ports, I. Stoica, and X. Jin, "Harmonia: Near-linear scalability for replicated storage with in-network conflict detection," in *Proceedings of the 2019 International Conference on Very Large Data Bases (VLDB'19)*, 2019.

[180] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale RDMA deployments," in *Proceedings of the 2015 ACM SIGCOMM Conference (SIGCOMM'15)*, 2015.