Performance Evaluation of the Impact of NUMA on One-sided RDMA Interactions

Jacob Nelson
CSE, Lehigh University
jjn217@lehigh.edu

Roberto Palmieri
CSE, Lehigh University
palmieri@lehigh.edu

***Abstract*—Remote direct memory access (RDMA) and non-uniform memory access (NUMA) are critical technologies of modern high-performance computing platforms. RDMA allows nodes to directly access memory on remote machines. Multi-processor architectures implement NUMA to scale up memory access performance. When paired together, these technologies exhibit performance penalties under certain configurations. This paper is the first study to explore these configurations to provide quantitative findings on the impact of NUMA for RDMA-based systems. One of the consequences of ultra-fast networks is that known implications of NUMA locality now constitute a higher relative impact on the performance of RDMA-enabled distributed systems. Our study quantifies its role and uncovers unexpected behavior. In summary, poor NUMA locality of remotely accessible memory can lead to an automatic 20% performance degradation. Additionally, local workloads operating on remotely accessible memory can lead to 300% performance gap depending on memory locality. Surprisingly, configurations demonstrating this result contradict the presumed impact of NUMA locality. Our findings are validated using two generations of RDMA cards, a synthetic benchmark, and the popular application Memcached ported for RDMA.**

***Index Terms*—RDMA, NUMA, Performance, Locality**

I. Introduction

Remote Direct Memory Access (RDMA) is a technology that boosts network interactions by providing not only faster communication speed, but new features that allow a node to access memory physically stored on another node without needing to invoke software routines on the target machine (called *one-sided* communication). RDMA network interface cards (RNICs) are more expensive than the traditional Ethernet counterparts [1] but still affordable for high-performance computing infrastructures. In fact, RDMA-enabled computation is widely deployed by many computing infrastructures, spanning industry and academia, such as Microsoft Azure [27], CloudLab [36] and Chameleon [25].

As recently corroborated by several studies [11], [13], [31], one-sided verbs represent the true innovation of RDMA communication, opening up endless opportunities to redesign existing algorithms and systems in order to fully exploit such high-performance interaction. This paper presents the first empirical study on the performance implication of RDMA one-sided communication when deployed along side Non-Uniform Memory Access (NUMA)[1]. *While NUMA is a recognized challenge for RDMA-based systems [13], [41], no comprehensive study exists characterizing one-sided verbs.*

A NUMA server divides main memory into groups, where each of these groups is directly connected with a set of CPU-cores (typically a physical processor or CPU). Such a organization allows a thread executing on a core to access memory fast, if the requested address is located on the memory module attached to its executing core. Otherwise, the access latency is higher. NUMA represents the de-facto standard for medium/large multicore servers. All current servers equipping more than one physical processor are NUMA. Additionally, single processor infrastructure with high number of cores can be NUMA as well [35]. Many solutions have been proposed to make existing technology NUMA-aware [5], [8], [9], [12].

Three factors motivate our investigation.

- First, RDMA and NUMA are two technologies that belong to all high-performance computing infrastructures, and they are meant to stay in the foreseen future since NUMA is widely adopted and RDMA's capabilities are being rapidly explored in a multitude of settings [3], [14], [25], [28], [36], [39], [41].
- Second, one-sided communication is the real innovation underlying RDMA, and has the unique characteristic of being agnostic to local server computation. This is because memory transfers over RDMA are not scheduled by the operating system or by any software component in the server receiving the memory operation; the RNIC operates autonomously. The implication of such a design is that, although a remote operation shares hardware resources with the local server, thus hardware contention can affect its performance, the local server has no control or visibility of the operation, therefore runtime optimizations are disallaowed (e.g., NUMA balancing [2]).
- Third, a growing number of distributed data repositories (e.g., [22]) prefer storing data and metadata in main memory, while using replication to ensure fault-tolerance and availability. Because of this deployment, understanding the impact of NUMA on RDMA becomes essential for subsequent improvements.

An example that summarizes the above conclusions is the following. Let us assume two nodes in a cluster N_1 and N_2 and one application thread T running on node N_2. When T performs a read or write operation on a memory location that is stored by N_1, it performs a one-sided interaction, which

[1]A poster version of this paper is available at [29].

is directly handled by the RNIC of N_1 without involvement of N_1's operating system. The performance of T's memory operation depends upon factors such as:

- the NUMA locality of the requested memory;
- the workload executing on N_1, which might be memory intensive and (over)loading the memory hierarchy.

None of these factors are under the control of T or N_1, which means that T's operations might be slowed down because of reasons that neither T nor N_1 can control.

To conduct our study, we develop and test a microbenchmark and the well-known Memcached application [16], ported to use RDMA [20]. We develop the former to isolate behaviors stemming from RDMA in the presence of NUMA. We use the latter to demonstrate the aggregate effects analyzed in our microbenchmark. We configure our runtime to reflect a real-world configuration as detailed by Facebook's survey in [4], which demonstrates that a large percentage of key-value operation target data smaller than 64 bytes.

A quick summary of our finding is the following:

- access to memory on the same NUMA zone as the RNIC can lead to 10-20% better performance, for an unladen system;
- under local workloads, performance of NUMA-local accesses can be up to 300% worse than NUMA-remote;
- memory intensive independent workloads can reduce performance as much as 50% when operating in the same NUMA zone as remotely accessible memory.

Looking ahead, as the popularity of RDMA expands, the performance of the RNICs themselves will improve. The newest generation of hardware achieves an impressive sub-$600ns$ operation latency [1]. While the hardware used in this evaluation study is not the most recent, we postulate that our results will only become more pronounced in the newer devices. This is because the penalty of NUMA is fixed. As the access times for RDMA trend toward the upper bound of local accesses, the penalty incurred for the extra hop across the processor interconnect represents a higher relative cost. What is currently 10%-20% degradation in performance imply even higher losses with a faster RNIC.

To the best of our knowledge this is the first empirical study on the performance implication of RDMA and NUMA architectures in a distributed system. As stated earlier, what makes our work unique is the focus on RDMA one-sided interactions, which currently cannot be optimized by either the receiving nor the sending server since these operations are entirely handled by the hardware. We believe our findings open up exciting new directions aiming at the exploitation of RDMA one-sided interactions along with NUMA servers in a less agnostic way so that remote accesses can be optimized and not penalized.

The source-code of our implementations is available at https://github.com/sss-lehigh/nurdma along with the raw data collected for the experiments included in this paper, instructions and scripts for execution and for plotting data, as well as additional experiments.

II. Background

Non-Uniform Memory Access (NUMA) and Remote Direct Memory Access (RDMA) are two technologies widely used by multicore servers and high-performance computing environments; each solving a unique problem as systems scale up and out. NUMA enables efficient memory access on multiprocessor machines [23], while RDMA lowers the network overhead by avoiding unnecessary operating system's context switches (and change of mode) and memory copies [32]. The InfiniBand network protocol [17] supports RDMA at the transport layer over a high throughput and low latency interconnect. Together, clusters with tens/hundreds of cores per node and sub-microsecond communication between nodes can be realized. In the following two subsections we provide implementation details of NUMA and RDMA to support the terminology used in our evaluation study.

A. Non-uniform Memory Access

NUMA addresses the issue of memory bus contention for multiprocessor machines, which can scale up to hundreds of cores [23]. Concurrent memory access by a large number of cores causes the memory bus to become a bottleneck for system performance. To alleviate conflicts, NUMA partitions both computing and memory resources into *zones*, with a special interconnect between them. All system memory is still shared by all cores, and all caches remains coherent, but now contention for each zone's memory bus is reduced if threads only access memory in the same zone as the core they are running on (called *NUMA-local*). However, accesses to remote zones (called *NUMA-remote*) must pass over the interconnect and thus incur additional overhead.

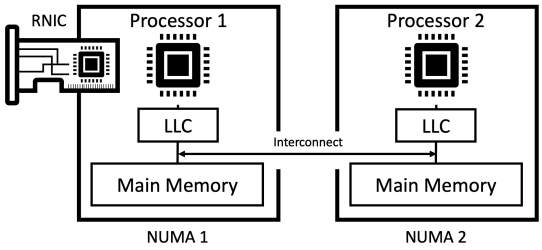


Fig. 1. High-level organization of a multi-processor machine with two NUMA zones and the RNIC attached to NUMA 1.

Figure 1 illustrates the high level structure of NUMA architecture machines. Each processor is associated with a physical module of memory, but all system memory is accessible over the memory interconnect. Since NUMA-remote accesses are slower, it is the applications best interest to maintain data locality by binding pages to NUMA-local memory. Some operating systems use NUMA balancing [2] to transparently co-locate threads and memory in the same zone.

This diagram only shows two NUMA zones but modern processors like the Intel Xeon E7 8800 series can support up to an eight socket configuration [18]. As the number of sockets grows, an inter-socket message may need to make multiple hops to reach its destination, exacerbating the problem of

NUMA-remote memory access. The table in Figure 2 shows memory access latency results in nanoseconds for a four-socket machine of Intel Xeon Platinum 8160 processors with a total of 192 cores.

Zone	1	2	3	4
1	**99.4**	131.4	142.4	140.3
2	136.5	**99.5**	139.6	145.4
3	145.6	140.4	**93.6**	131.2
4	140.2	145.3	134.4	**99.3**

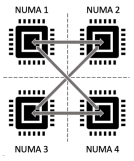


Fig. 2. Memory access latency in nanoseconds in a server with 4 NUMA zones. The diagonal values in bold correspond to NUMA-local accesses.

One consequence of the NUMA organization is that I/O is also bound to a given zone, thus playing a role in networked computation. External devices communicate with processors through adapters that are physically connected to one NUMA zone. Section II-C provides a more detailed explanation of this aspect. The result is that memory accesses originating from a device targeting an address whose page resides in NUMA-remote memory must traverse the interconnect in order to be served. As we will see in our evaluation, this can impact the performance of RDMA despite an order of magnitude difference between the network and interconnect latency.

B. Remote Direct Memory Access

RDMA gives a node the ability to remotely access the physical memory of another node directly, without involving the CPU of the remote node. Bypassing the CPU avoids two significant overheads, which are present in standard TCP/IP stacks. First, everything is performed in user-space so there is no CPU overhead due to mode or context switches. Second, direct memory access (DMA) is leveraged by the RDMA-capable network interface controller (RNIC) to avoid unnecessary memory copies, as detailed below.

All RDMA interaction shares certain procedures to set up communication and register remotely accessible memory. Each endpoint maintains a queue pair (QP), which consists of a send queue and receive queue. Work requests are added to these queues during operation depending on the communication protocol used. During creation, this structure is associated with a remotely accessible memory region.

Endpoints exchange data via *one-sided* or *two-sided* verbs. Two-sided verbs are somewhat analogous to the socket model presented by TCP/IP. The name of this type of RDMA communication is based on the fact that both sides are needed. We avoid a detailed discussion of this form of communication as it does not pertain to our study.

To issue a one-sided RDMA, the requester posts a work request containing local and remote addresses, size, and remote key to their send queue to initiate the operation on remote memory. These verbs only require action from the requester and, in contrast to two-sided communication, the requester has knowledge of the virtual address on the remote node. In the case of read operations, remote data is copied to the local address and a work completion is added to the completion queue. For writes, the remote RNIC ACKs the request and a work completion is added to the requester's completion queue. One-sided communications requires a reliable connection and QPs cannot serve multiple connections [17]. Communication is faster than for two-sided verbs with the caveat that scalability can suffer when the number of connections increases significantly.

As mentioned before, since one-sided interactions are agnostic to the machine's operating system and, at the same time, they share the same physical hardware resources of the local operating system and applications, their performance can be significantly affected by factors that cannot be optimized at runtime by either the requesting applications nor the software on the receiving machine. One consequence of this transparency is that NUMA balancing optimization [2] cannot be employed to move memory pages between NUMA zones because pages must be locked upon memory registration with the RNIC.

C. NUMA and RDMA I/O

Modern architectures offer a mechanism for I/O to directly access the last-level cache. For example, the Intel machines used in this study use Data Direct I/O (DDIO) to achieve direct cache access. Given modern I/O speeds and cache sizes, it is practical to allow I/O to access cache to avoid overhead. Previously, incoming data would be written to main memory and local accesses would then read it into cache. With technology like DDIO, I/O latency improves for accesses to cached memory and local computation benefits from I/O placing memory directly into the cache.

The intent of this technology is to transparently improve latency and throughput for I/O operations. However, it is important to note that this is currently only applicable to the cache in the same NUMA zone as the I/O controller and is enabled by default. Physical memory in the remote NUMA zone is accessed by a normal DMA [18]. As we will address in our evaluation, this behavior can act against RDMA and negatively impact performance.

III. Related Work

Extensive investigation of the role of NUMA locality in system performance solidifies it as an important consideration when designing high-performance applications to run on modern multicore machines. Integrating NUMA-awareness into algorithms and data structures improves performance [8], [9], [12], [24], [42].

Recent literature has been flooded with systems that exploit RDMA for different types of computation. Because of its reduced communication latency, RDMA is an ideal technology for, but not limited to, distributed transactional systems [10], [14], [20], [22], [39], [40], distributed shared-memory [3], [7], [13], data transfer and storage [6], [33], and group communication [38]. Most systems using one-sided communication implement a similar pattern that resembles the traditional client-server model using one-sided writes for message passing and one-sided reads for direct data access. Their designs

leverage the low latency and remote CPU bypass of one-sided operations to achieve very high performance. Some of these works address NUMA locality and attempt to mitigate its impacts by pinning memory and threads to the NUMA zone local to the RNIC. However, none of them offer an in-depth study of the relationship between the two technologies and the combined performance impact.

Kalia et. al [21] propose a set of guidelines for designing high-performance RDMA systems. They explore the intricacies of RDMA in terms of the low level details. We do not directly address many of the optimizations described because they are specific to two-sided communications, such as message in-lining and WQE shrinking. Their work highlights the implications of the PCIe bus and RNIC architecture, but their analysis fails to include a discussion on NUMA locality of remote memory. One aim of our work is to provide an additional resource for designing high-performance RDMA-based systems.

Recent distributed systems use the combination of RDMA interconnections and large multicore servers to test their performance. Examples include FaRM [14], HydraDB [39] and DrTM [40]. FaRM uses two RNICs and requires threads to interact only with the RNIC in the same NUMA zone. To address scalability FaRM allows QP sharing between threads to pass messages, but only threads within the same NUMA zone share a QP. Similarly, HydraDB ensures NUMA awareness by co-locating memory and server processing threads and deploying multiple instances on multi-socket machines. DrTM explicitly partitions their TPC-C benchmark between NUMA zones, such that all work is isolated to a single zone. In all cases, NUMA locality is acknowledged but a conservative approach is taken; its impacts are not quantified.

FaSST is a distributed in-memory transactional system implemented using RPCs over two-sided communication [22]. It represents an important orthogonal discussion on the cost-benefit analysis of one- and two-sided RDMA operations. Additionally, from what we can surmise from the source code, FaSST takes a similar approach to FaRM and DrTM by pinning resources to the same NUMA zone as the RNIC.

The most explicit discussion of the impact of NUMA on RDMA, that we are aware of, is presented by Wu et. al [41]. They propose a distributed graph engine, GRAM, which is designed using FaRM's message-passing mechanism. Their work demonstrates that maintaining message buffers in the same NUMA zone as the worker threads receiving the messages can improve performance by 40%. Here the measured cost of NUMA stems from locality with respect to the local threads that access the memory, not the memory access performed by the RNIC. In our evaluation study, we explore the latter and demonstrate the impact of NUMA locality on the performance of one-sided communication itself.

Similar to FaRM, Ren et. al [34] side step the effects of NUMA in their end-to-end data transfer service by partitioning data into each NUMA zone then using multiple RNICs, each attached to a NUMA zone, to route accesses to the appropriate zone. This induces a multiplicative effect on the number of RNICs that are needed per machine, which in some cases might not be supported by hardware and in all cases is more expensive.

Another solution is Mellanox's Socket Direct technology, which shares a single RNIC between NUMA zones by splitting 16x PCIe connection into a form factor configured for two 8x slots [26]. This allows two processor sockets to share the RNIC while avoiding inter-socket communication. This approach has two limitations. First, the bandwidth for a given socket is cut in half due to the decreased width of the PCIe connection. Second, if the system has more than two NUMA zones, then the issues we discuss in this paper remain.

Other research looked into the similarity between NUMA architectures and distributed systems leveraging RDMA, proposing an implementation of an RDMA-based Distributed Shared Memory [3]. The main idea is to treat remote memory similarly to a NUMA node when designing algorithms. In contrast to our investigation, this work does not address the performance penalties incurred by memory locality and the interplay between NUMA and RDMA.

The inevitability of NUMA means that completely eliminating its penalty entails rethinking fundamental architectural decisions. IOctopus [37] is a recent solution that attempts to address this by re-imagining the PCIe device architecture to mitigate the more expected results demonstrated in our evaluation. Even so, the unexpected behaviors we encounter relating to RDMA and DDIO on NUMA-local memory may still be unavoidable under the proposed PCIe framework, as none is commercially available yet.

To the best of our knowledge, the common approach is to handle the NUMA penalty by conservatively allocating remotely accessible memory on the same NUMA zone where the RNIC is physically connected. Such a design limits the available computing capability to the cores/threads scheduled for execution on that NUMA zone and it disallows further optimizations. Our work illuminates the coupling between these two technologies so that future systems make take full advantage of each of their potentials with confidence.

IV. Experimental setup

Our analysis starts with a client-server microbenchmark designed to capture subtle interactions between the two hardware components of interest. After understanding the physical implications of NUMA on RDMA performance of one-sided verbs, we then demonstrate that NUMA locality indeed plays a role in performance for more complex scenarios. We choose an RDMA-based implementation of Memcached [20] to explore the effects of NUMA in a real-world setting. The microbenchmark isolates phenomenon to understand primitive interactions; RDMA-memcached represents behavior for a more complex application.

A. Locality

Before delving into our evaluation study, we first define some terminology that will clarify our notions of locality. We assume a RNIC-centric view of NUMA for each machine. In

other words, *RDMA-local* refers to the NUMA zone where the RNIC is connected via the PCIe bus. Memory can also be *NUMA-local*, which means it is physically mapped to the same NUMA zone as accessing threads. Additionally, resources can also be local or remote with respect to a given node.

To illustrate this, consider a two NUMA system like the one in Figure 1, consisting of NUMA zones N_1 and N_2. Let us assume the RNIC is attached to N_1, thus any resource in this zone is considered RDMA-local. Resources in N_2 are considered RDMA-remote. Furthermore, if thread T_1 is running on N_2, then T_1 may access NUMA-local memory residing in N_2. Finally, if a client reads (writes) memory on the server's N_2 then we refer to this as a RDMA read (write) on RDMA-remote memory.

B. Testbed configuration

For the following experiments, we use nodes consisting of two Intel Xeon E5-2670 v3 processors with a total of 48 cores on each machine. For each physical core, the L1 data cache is 32 KB, the L2 cache is 256 KB and there is a 30 MB L3 cache shared between all cores. All nodes in the system run Ubuntu 16.04 and are equipped with one Mellanox ConnectX-3 single port RDMA adapter connected over a 56 Gbps InfiniBand network. All of experiments are implemented in C++ using version 4.5 of the Mellanox driver for Linux.

In addition to testing our results on the ConnectX-3, we also validate the more surprising findings on the subsequent generation of RNIC for consistency. In the modern RNIC testbed, each machine has two 10-core Intel Xeon E5-2650 v3 processors with the same higher-level cache sizes as the previous setup, a 25 MB L3 cache and a single Mellanox ConnectX-4.

We do not explore the configuration where multiple RNICs are connected to each machine because we believe it would obfuscate our findings. In fact, it is possible to connect multiple RNICs, where each is physically connected with a different NUMA zone. As mentioned in Section III, the common approach is to use a static assignments of software resources with multiple RNICs to address some of the issues highlighted by our evaluation study. However, even with this design, large multiprocessor systems can easily deploy 8 NUMA zones. Equipping each single machine with 8 RNICs would be significantly more expensive and would still not change the outcome of our findings.

C. Workload characteristics

For the initial set of experiments we assume that the client runs in a favorable configuration. That is, we bind all resources to the RDMA-local NUMA zone such that it does not incur any overhead due to NUMA locality. Under this assumption, we restrict any behavioral outcome to stem from changes in remotely accessible memory location or the server workload. We address poor NUMA-awareness on the part of the client when evaluating RDMA-Memcached in Section VI.

Based on our preliminary results, we fix the size of accesses to be between 8 and 64 bytes. Atikoglu et. al report that in Facebook's key-value store, for the most general workload, the majority of key and value sizes are less than 64 bytes [4]. Another class of workload consists of only requests with keys of 16 or 21 bytes and value size of 2 bytes. Thus, our configuration is representative of a real workload. Similar statistics are reported by Nishtala et. al [30] and small key and value sizes are used to test numerous state-of-the-art key-value store designs [13], [15], [22].

As mentioned in Section II-B, one-sided communication requires dedicated QPs for every connection. Avoiding RNIC cache misses on QP states is a known optimization for high-performance RDMA [21]. We limit the number of QPs to be fewer than 32 in the following experiments to probe behavior without perturbing results with RNIC cache misses or contention on RNIC resources. Similarly, we allocate remotely accessible buffers smaller than a page to avoid RNIC address translation overhead. Larger pages can be used to accomplish the same result if large memory regions are required [13].

V. Microbenchmark

The primary goal of our microbenchmark is to highlight the primitive interactions between RDMA and NUMA for one-sided verbs.

During initialization, the server creates a shared memory region for remote operations and listens on a socket for client connection requests. Once a connection is received, the server sends the virtual address and access key to the client so that the client can begin RDMA operations. The client and server synchronize over sockets again to ensure initialization is complete, then the client starts issuing remote accesses to the buffer located on the server.

In a real world setting, other processes may run on a server, either interacting with data stored in a remotely accessible buffer or as an independent application. We mimic both behaviors by concurrently running a synthetic workload or by introducing load threads on the server. In both cases, memory accesses and inter-socket communication impacts performance of remote one-sided operations. As we will show in Section V-D, architectural design can have a surprising effect on performance when a local workload is introduced on the server node.

In each experiment, every client connection issues 1,000,000 operations to the server, during which we record individual client latency or throughput. Throughput is measured as the average of 10ms-long instantaneous throughput taken during execution. When measuring latency, we immediately poll for completions after issuing the RDMA operation then calculate the average and standard deviation per client, reporting the overall average across all clients. For throughput, we average the results of each connection over the course of the run, which typically consists of 200 samples, then report the total system throughput as the sum of each client's average throughput.

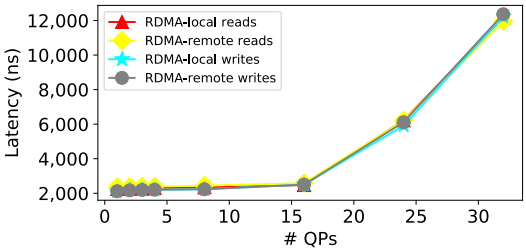
Although the entire memory region is shared between QPs, writes are non-conflicting. This is explicitly done to ensure that clients do not interfere directly by accessing the same memory addresses. Thus, memory allocated for remote access on the

server is equal to the number of connections multiplied by the size of each access. For smaller access sizes false sharing is possible. Each connection writes to disjoint but adjacent addresses and the starting address of the memory region is cache aligned.

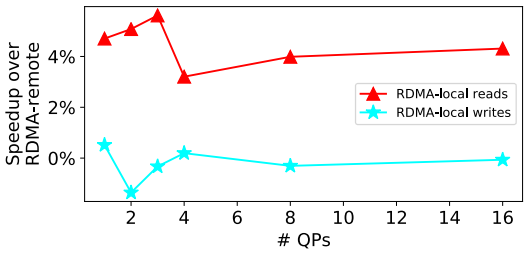
A. Individual operation latency

As a preliminary motivation for subsequent experiments, we first measure the latency of client one-sided reads and writes as a function of the NUMA locality. The aim of this simple experiment is to understand the baseline impact of NUMA, if any, while controlling for additional factors, such as server local workload. In this experiment, latency is measured over one-million iterations of a closed loop for clients issuing requests of 64 bytes, which corresponds to the maximum individual write by a cache-coherent I/O operation and avoids false sharing.

Hypothesis. We hypothesize that latency will degrade for reads when the remotely accessed buffer is kept in the RDMA-remote NUMA zone, primarily due to traversing the inter-socket connection. Writes on the other hand are acknowledged as they arrive at the server RNIC, so we expect to see similar latency for both NUMA zones.



(a) Latency measured for varying numbers of QPs.



(b) Speedup of RDMA-local operations over their RDMA-remote counterparts.

Fig. 3.

Findings. Figure 3(a) shows latency measurements for both RDMA-local and RDMA-remote operations. To illustrate how remote operations exercise the memory hierarchy of the remote machine, no workload executes on the server. We can see from this plot that when using more than 16 QPs, latency spikes.

We attribute this behavior to inherent limitations of the RNIC, which involve the number of parallel execution units as well as a small cache [13]. When limited to 16 or fewer QPs, we observe that the additional load on the network adapter does not significantly change latency. A round trip for each operation ranges from $2.3\mu s$ to $2.5\mu s$ ($2.4\mu s$ to $2.6\mu s$) for RDMA-local (RDMA-remote) reads and $2.1\mu s$ to $2.5\mu s$ for both RDMA-local and RDMA-remote writes. Reads are slower than writes because a data return is required. In contrast, write acknowledgements are produced as soon as the write is received by the server's RNIC.

The speedup of RDMA-local latency over RDMA-remote latency is shown in Figure 3(b) for up to 16 QPs. Our simple test reveals a consistent 3-5% latency degradation for RDMA-reads when the accessed memory region resides in the RDMA-remote NUMA zone. We estimate the contribution of DDIO and the socket interconnect to latency by measuring local access times using Intel's Memory Latency Checker (MLC) tool [19]. Similar to Table 2 we measure local access latency for our server node but also record L3 hit latency. As previously discussed, DDIO does not utilize cache-coherent operations when memory does not reside on the same NUMA zone as the I/O. Local L3 access within a NUMA zone is $38ns$ on our machines, estimating the time spent in the memory hierarchy during an RDMA-read. Access to NUMA-remote memory is $122ns$. With respect to RDMA, our observed difference of 3.6% over the $2.3\mu s$ baseline RDMA-read, to RDMA-local memory, is due to both traversing the socket interconnect and going to memory instead of leveraging cache. Figure 3(b) also shows that, as expected, there is no significant difference between write operations.

Although relatively small, the impact of NUMA is consistent. For all runs with 16 or fewer QPs the standard deviation was less than .01% of the average latency. In subsequent experiments we will demonstrate that a 3.5% increase in latency can lead to 10-20% worse throughput in the presence of an independent workload. One peculiar result we observed is a bi-modal distribution in access latency for more than 8 QPs. The first seven QPs created have lower average latency than the remainder. When only 16 QPs are used, the difference is small. However, a $5000ns$ differential is observed for 24 QPs and $12000ns$ for 32 QPs. Without fully understanding the internals of the RNIC, we avoid speculating about the cause of this behavior, although we suspect it relates to caching QP contexts and how the RNIC schedules QPs to its execution units. Regardless, for all subsequent experiments, we turn our attention to throughput. This configuration serves to demonstrate performance characteristics without introducing additional bottlenecks, such as the RNIC itself.

B. Throughput of different RDMA access sizes

Demonstrating the impact of NUMA locality as a function of access size informs the design of RMDA-based systems, especially those with fine-grained memory accesses. In all of our experiments the maximum transfer unit (MTU) is 256 bytes, which is the smallest configurable MTU for the ConnectX-3 and also corresponds to the configured maximum PCIe payload size of 256 bytes for the RNIC.

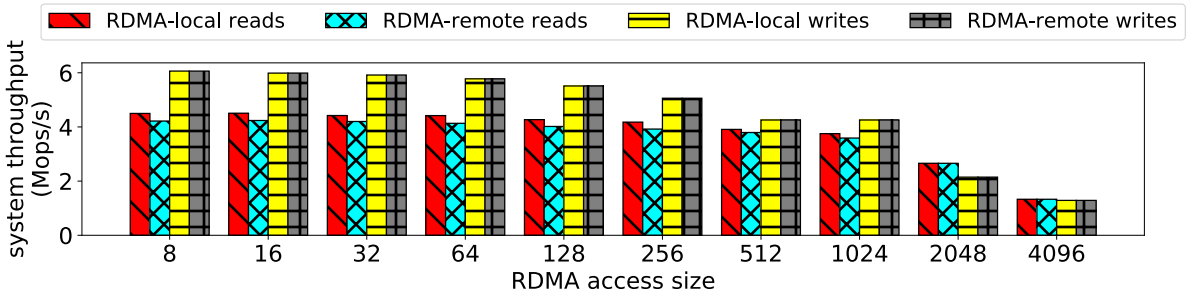


Fig. 4. Total throughput for 8 client connections with different memory access sizes at an MTU of size 256B.

Hypothesis. For RDMA-reads, we expect to see degradation for RDMA-remote accesses, similar to our latency experiment. However, the load induced by concurrent clients is now a factor because of the open-loop execution model. The additional I/O is expected to heighten the impact of NUMA locality because the remote memory hierarchy will be loaded. Again, writes are acknowledged when received by the RNIC of the remote machine and therefore we do not expect NUMA to play a role.

Findings. Figure 4 shows the system throughput as RDMA access size increases for 8 clients. Similarly to our experiments on latency, the throughput of RDMA-writes are not affected by NUMA locality. On the other hand, RDMA-reads consistently see 5-7% worse throughput when RDMA-remote. Note that NUMA locality does not influence throughput when messages are segmented (i.e., size is greater than 256). This is because more time is spent on network communication overshadows the impact of RDMA-remote accesses.

Another observation is that the disparity between RDMA-reads and RDMA-writes diminishes as access sizes increase. In other words, reads and writes are impacted differently by incrementally larger memory access sizes. RDMA-reads retain over 80% of the throughput at 8 bytes through 1024-byte accesses, while RDMA-writes are at 70% of the baseline. Writes suffer in comparison to reads beyond 1024-byte accesses, which we attribute to the additional communication needed between the RNIC and memory controller across the PCIe bus. At 4096-byte accesses, both operation types have similar throughput.

We also test larger MTU sizes to determine the impact, if any, on system performance and NUMA locality assumptions. A designer might elect to use a larger MTU if there is batching at the application level. At larger packet sizes (i.e., greater than 256 bytes), the impact of NUMA is still present through 1024-byte accesses. Regardless of changes in overall throughput, a larger MTU does not eliminate the effects of NUMA locality for smaller access sizes, which persists for individual RDMA operations. Independent of MTU, throughput for 2048- and 4096-byte accesses was consistently lower and the impact of NUMA is negligible, both resulting from other bottlenecks like the need of undergoing multiple PCIe trips.

In summary, this experiment demonstrates that in configurations where throughput is not restricted by other aspects of the complex interaction of RDMAs, the effects of NUMA locality are visible across varied RDMA access sizes.

C. Independent application load

In this experiment we run an independent process which allocates a buffer in memory and spawns threads that access the buffer randomly. The memory footprint, the number of threads and the read-write ratio are all configurable. Additionally, we specify the NUMA locality of the buffer and threads to examine their impacts on RDMA performance.

Hypothesis. When an independent workload executes concurrently with remote memory accesses, we expect to see performance decrease for RDMA-reads. The root of the expected change stems from the contention on the server machines memory subsystem. For the same reason as before, we anticipate RDMA-writes to remain stable.

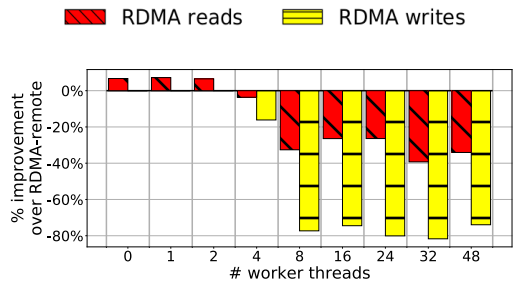
Findings. We first measure the throughput of one-sided RDMA operations as a function of the size s of the buffer while 128 threads randomly read and write with 50% probability each. This experiment is NUMA-agnostic, meaning the workload threads and memory are not bound to a specific NUMA zone on the server. As in the previous section, tests consist of 8 QPs between the client and the server and read or write 64-byte chunks of memory. With this configuration we were surprised to see no impact on performance. However, reducing the access size to 8 bytes demonstrated that for RDMA-remote reads, the independent workload indeed plays a role; restricting performance by as much as 20% of the no load baseline. As the buffer increases in size, the effects of NUMA locality also become more substantial.

Next, we study the impact of NUMA locality of the independent workload and find that regardless of NUMA placement, there is no substantial difference in throughput, thus pointing toward contention on the QPI as the primary culprit for the performance drop measured for a NUMA-agnostic workload. This is because RDMA operations are transparent to the remote machine, and therefore are susceptible to similar performance characteristics as local accesses. In this case, the local workload is enough to saturate the QPI and ultimately impact performance. NUMA bound workloads, on the other hand, do not saturate intra-processor network and therefore do not impact RDMA performance.

In contrast to the previous experiments the calculated performance drop is slightly more in this setting, and as we will see in Section VI, the existence of an independent workload has a much higher impact on overall performance when there is server-side computation. In which case, the combined effects seen in our microbenchmark impact performance to a much larger degree.

D. Load on remotely accessible memory

As we mention, serving processes themselves are typically not idle, but may perform computation on local data. Symmetric distributed transactional systems, graph engines and distributed shared memory are all examples of this paradigm [7], [10], [13], [40], [41]. To demonstrate the impacts of this class of workload by introducing server load threads to our microbenchmark, which operate on data that is also accessible by remote clients.



(a) ConnectX-3.

(b) ConnectX-4.

Fig. 5. Percentage improvement in throughput for RDMA-local operations over RDMA-remote on remote reads and writes by 8 client connections, with increasing local load. Local load is 10% reads and 90% writes.

Worker threads are launched at server startup and begin randomly accessing remotely accessible memory. Initially, use a workload consisting of 90% writes and 10% reads. Client connections then begin issuing operations to the server. In line with the results seen in the previous experiment, we reduce the size of the remote memory accesses to 8 bytes; again testing with 8 connections. Note that all remotely accessible memory can be stored in cache.

Hypothesis. Because the accessed memory is common to both the local and remote workload, we expect to see higher degradation than before for RDMA-reads in the presence of a workload. RDMA-writes are expected to maintain performance since they are ACK'ed immediately upon receipt by the remote RNIC.

Findings. Our results reveal something entirely unexpected. Under a local workload, one-sided RDMA writes to RDMA-remote memory outperform their RDMA-local counterparts by 70-80% when more than 8 threads also accesses the same memory region. That is, both memory and worker threads are in the RDMA-remote NUMA zone. Under the same circumstances, RDMA-remote reads have 35% better throughput than the RDMA-local operations.

This behavior contradicts the expectation that RDMA-local operations should be preferred over RDMA-remote. Additionally, we see the largest performance impact for RDMA-writes, which base on our previous experiments was not anticipated to change. We save a full discussion of this for Section VIII, but we note here that this is likely due to RDMA-local operations directly accessing cache. Although DDIO is designed to enhance I/O performance, in the presence of a write-intensive local workload on remotely accessible memory, DDIO actually hinders overall performance. On the other hand, client operations to RDMA-remote memory use DMA and do not interact with the local workload at the cache level. We suspect cache pressure limits throughput for the integrated I/O memory management unit, ultimately bottlenecking the RNIC.

To validate our empirical findings we rerun the 8-connection and 8-byte configuration on a cluster equipped with Mellanox ConnectX-4 RNICs and the same Intel servers, albeit with slightly fewer cores. Figure 5(b) demonstrates that even on the newer generation network adapter, the trend still holds. Remarkably, on the new hardware the behavior is intensified, with RDMA-remote writes outperforming RDMA-local writes by 3x. This suggests that the underlying interaction is not RNIC specific, but is directly correlated to machine architecture. We believe that DDIO plays an important role in the existence of this trend.

We also tested various other configurations, including lower write ratios, more client connections, larger access sizes and the next generation RDMA card. Importantly, this unexpected result remains when the workload is 1:1 reads and writes. In this setting, we observe nearly 50% better performance for RDMA-remote writes. When we increased the number of connections to 32, we also observed similar behavior as for the 8 connection results reported, but a muted response. For higher than 8 worker threads there was 5%-20% better performance for RDMA-remote writes. Finally, increasing the size of each remote access to 64 bytes eliminated the behavior, with RDMA-local returned to being slightly better performance than RDMA-remote. This unanticipated performance bug leads us to conclude that system designers requiring small RDMA accesses should account for the potential impact of local workloads on the behavior of their particular system. Our results also show that traditional beliefs about NUMA locality are not steadfast when it comes to I/O, and may be contradicted with significant consequences.

It should be noted that we do not implement mutual exclusion in this case, because it is not feasible between local load and RDMA operations on the available RNIC. Reads and writes are coherent, however, because of the underlying cache

coherence policy implemented by the machine hardware.

VI. RDMA-Memcached

We also deploy an RDMA-based Memcached implementation [20] and measure performance against NUMA locality. We follow the same principles as before to show behavior for a real application. The setup is similar to our microbenchmark with a Memcached instance serving clients emulated using the provided `memslap` workload generation tool.

To the best of our knowledge, for less than 512 connections, RDMA-Memcached uses one-sided communication. At larger numbers of connections a hybrid approach is used to avoid the memory cost associated with one-sided connections. Our experiments target less than this threshold and therefore are purely one-sided.

Memcached uses active messaging with one-sided RDMA reads for all data transfers [20]. For puts, the client passes the location of data to write then server reads data via RDMA from the client to the key's corresponding slot. To get a value, the client first exchanges information about the location and size of the value desired key, it then allocates a destination buffer and RDMA reads the data from the server. Note that both Put and Get leverage RDMA reads for data transfer.

We configure Memcached to execute as a single worker thread responding to requests from a single client to pinpoint the impact of resource locality. During execution, each key-value pair is a total of 67 bytes, with the values being 64 bytes long. Keys and values are randomly filled with alphanumeric characters. Client issue 90% reads and 10% writes to Memcached. We report the average throughput of 10 trials of 100,000 operations per client connection.

Figure 6(a) shows throughput for varied number of client connections when no load is running on the server. Memcached becomes fully loaded after 4 connections and the RNIC becomes saturated after 16 connections. We test three scenarios: (1) both server and client are pinned to the RDMA-local or (2) RDMA-remote zone in their respective nodes; or (3) we bind the client to the RDMA-local zone and the server to the RDMA-remote zone, similar to the microbenchmark.

It is clear from the plot that when no load is present performance is best when both the client and the server are RDMA-local. Moving each to the RDMA-remote zone has an incrementally worse impact on overall throughput.

Figure 6(b) shows system performance for the same configurations as before, except that a NUMA-agnostic load is introduced. This load consists of the remainder of the server threads, not being used by Memcached, randomly accessing a large memory buffer, with a mix of 50% reads and 50% writes. The load running on the server causes a significant impact in overall performance for all configurations, but still we see a 10% decrease in performance due to being RDMA-remote.

Perhaps a more interesting case is when the workload is bound to a specific NUMA zone. Figure 6(c) shows this case where the load runs on the RDMA-local NUMA zone. When the load is RDMA-local, RDMA-remote throughput is not impacted by the load. We do not include the plot due to space constraints, but when the load runs on the RDMA-remote zone, RDMA-local memory accesses are not affected. More importantly, for both cases the workload reduces RDMA operations to the same NUMA zone by 40-50% for fewer than 24 connections. Therefore we expect 2x speedup for operations to an unburdened NUMA zone over the loaded zone.

VII. Additional Findings

Over the course of our experimentation, we recorded interesting trends that involve RDMA and the memory architecture, and further contribute to the performance penalties discussed.

First, we observed that RDMA reads are not cache allocating. That is, if the memory targeted by an RDMA read is not already in the cache, then the cache will not be filled, but rather the value will be retrieved from main memory and performance will be about 3% worse. This behavior was confirmed by performing to cache the data and measuring the change in performance, then flushing the cache and measuring the delta again.

Second, we noticed during our investigation of message size that larger operations tend to have an inverse relationship between locality and performance when they are not segmented. In other words, if the MTU is larger than 1024 bytes and operations are larger than 256 bytes, then RDMA-remote operations tend to outperform RDMA-local by 5-10%.

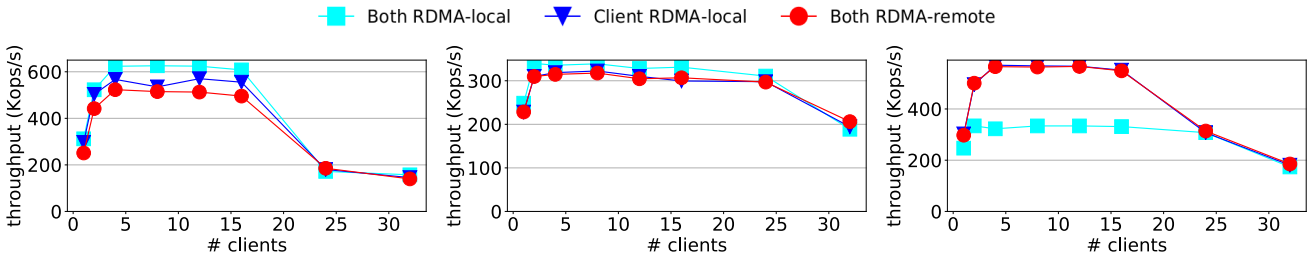
Finally, we attempted to measure the impact of DDIO by disabling write allocating flows in the PCIe configuration space of the root port to which our RNIC is connected. Unfortunately, this disables optimizations that are necessary for internal operations and leads to significantly worse performance across the board. Further investigation in this direction is necessary to fully dissect behavior when there is local work on remotely accessible buffers.

VIII. Discussion and Future Directions

The relationship between NUMA and RDMA hinges on a number of complex hardware interactions and depends on system workload. Here we summarize our findings and give the intuitions behind the observed behaviors. Then, we provide general expectations for performance impacts of NUMA on one-sided RDMA interaction. First, we give an itemized list of our key findings, then provide more explanation below:

- When execution is not subjected to additional workloads, remote reads to an RDMA-remote buffer can lead to 5-10% worse performance.
- In the presence of an independent workload, RDMA-remote operations can experience 2x slowdown for complex applications.
- Workloads operating on remotely accessible memory that is RDMA-remote are shown to have 3x *better* performance than RDMA-local. Contradicting widely accepted assumptions about NUMA locality.

In the case of an unloaded RDMA-Memcached server, NUMA locality on average reduces throughput of our one-sided operations by 10% for each side of the connection. In the worst-case scenario, which is when both client and server are



(a) No workload. (b) NUMA-agnostic workload. (c) RDMA-local workload.

Fig. 6. Memcached throughput for different independent workload configurations.

RDMA-remote, we expect performance to be 20% worse than the best-case (i.e., both RDMA-local). The fundamental reason for this difference is twofold. First, RDMA-local interaction automatically leverages direct cache access when handling requests, which improves latency and throughput. Second, data retrieval must pass over the interconnect between sockets, incurring additional overhead. The combined influence of these two factors leads to 20% worse performance when the RDMA memory regions on both the client and server are RDMA-remote.

Although a 10% per-side performance degradation might not seem substantial, it is consistent across our microbenchmark and real application. This performance penalty might hide the performance advantages of new designs and implementations. It is an open question how to combat this behavior since physical pages containing RDMA accessible memory cannot be moved at runtime, and one-sided RDMA operations are transparent to the remote machine.

When the server is loaded, RDMA operations are no longer executing in an isolated setting. In our evaluation we test both an independent workload and a workload that access remotely accessible memory. When the workload is independent and the application involves computation on the server, the NUMA locality of the workload has a significant negative effect (i.e., 2x slowdown) on one-sided operations to the same NUMA zone as the workload. This is because both RDMA operations and server computation contend for resources with the workload.

If the workload is not read-only and operates directly on memory that is accessible via RDMA, and the size of RDMA operations are small, then RDMA-local operations perform worse than RDMA-remote. This result was unexpected and attributed to DDIO. RDMA operations see additional overhead from load on the cache interconnects, such as cache-line invalidation, when the memory they access also resides in other core's higher-level caches. Thus, if applications require this type of workload it may benefit to use RDMA-remote memory to avoid cache contention.

As a final observation, we are motivated by our comparison with the newer RNIC. Future hardware improvements will continue to yield lower latency and higher throughput. One consequence of these advancements is that the impact of NUMA will become more visible; it will constitute a larger portion of the overall hardware path performance.

Future Directions. Leveraging our findings, we identify a number of research opportunities to provide components optimizing distributed systems using multiprocessors servers. These ideas share the aim of making RDMA operations more intelligent with regard to NUMA locality, allowing for solutions that can avoid falling into the performance bugs highlighted in this paper. The following is an overview of the particularly interesting directions we envision, with additional details in the subsequent paragraphs:

- RDMA-aware NUMA balancing to automatically move remotely accessible memory to the optimal NUMA zone.
- Workload analysis to automatically pin memory regions to NUMA zones.
- NUMA-aware synchronization mechanism for local and remote operations.

NUMA balancing is a scheduling policy implemented by many Operating Systems that attempts to move pages and computation in order to minimize traffic over the NUMA interconnection. NUMA balancing cannot be of help in the case of RDMA one-sided interactions since these operations are invisible to the operating system. Making the local operating system aware of such remote access pattern would allow reducing some of the overhead discussed in this paper due to RDMA-remote computation.

A more immediate solution to the issues discussed in this paper is to adopt workload analyzer and use the outcome to preemptively pin memory to NUMA zones depending upon access patterns. We believe our findings will drive system designers to make decisions that avoid the listed overheads.

An orthogonal problem is that operations cannot be atomic across both RDMA and local accesses, unless supported by the device. Current RNICs only support atomicity between remote operations. To provide global atomicity, loopback or advanced hardware technologies (i.e., Intel's TSX) must be leveraged [10]. A unified synchronization API that is NUMA aware and does not force local operations to be routed through the local RNIC, will provide programmers with a powerful tool to build high performance distributed shared memory systems.

IX. Conclusion

In this paper we evaluated the performance of one-side RDMA communication in the presence of NUMA servers. We found that locality of the accessed memory affects

performance of remote operations, and in addition to that, runtime optimizations are hard to provide since one-sided interactions occur unnoticed by the receiving operating system. Performance degradation spans from a "simple" 10-20%, in the case the accessed memory is not RDMA-local and the receiving server is under-loaded, to 300% better performance for RDMA-remote operations in the case of a write-intensive local workload on remotely accessible memory. Moreover, we demonstrate that performance characteristics do not necessarily adhere to the traditional wisdom that NUMA-local memory access is better than NUMA-remote.

Acknowledgment

Authors thank our shepherd and all anonymous reviewers for their important comments. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0367 and by the National Science Foundation under Grant No. CNS-1814974.

References

[1] Mellanox adapters. https://store.mellanox.com/categories/adapters/infiniband-and-vpi-adapter-cards.html, 2018.

[2] NUMA Balancing. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_tuning_and_optimization_guide/sect-virtualization_tuning_optimization_guide-numa-auto_numa_balancing, 2018.

[3] Scaling out NUMA-Aware Applications with RDMA-Based Distributed Shared Memory. *Journal of Computer Science and Technology*, 34(1):94–112, 2019.

[4] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-scale Key-value Store. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 53–64. ACM, 2012.

[5] A. Banerjee, R. Mehta, and Z. Shen. NUMA Aware I/O in Virtualized Systems. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 10–17. IEEE, 2015.

[6] E. Burns and R. Russell. Implementation and Evaluation of iSCSI over RDMA. In *2008 Fifth IEEE International Workshop on Storage Network Architecture and Parallel I/Os*, pages 3–10. IEEE, 2008.

[7] Q. Cai, W. Guo, H. Zhang, D. Agrawal, G. Chen, B. C. Ooi, K.-L. Tan, Y. M. Teo, and S. Wang. Efficient Distributed Memory Management with RDMA and Caching. *Proceedings of the VLDB Endowment*, 11(11):1604–1617, 2018.

[8] I. Calciu, D. Dice, Y. Lev, V. Luchangco, V. J. Marathe, and N. Shavit. Numa-aware reader-writer locks. PPoPP, pages 157–166. ACM, 2013.

[9] I. Calciu, S. Sen, M. Balakrishnan, and M. K. Aguilera. Black-box Concurrent Data Structures for NUMA Architectures. *ACM SIGOPS Operating Systems Review*, 51(2):207–221, 2017.

[10] H. Chen, R. Chen, X. Wei, J. Shi, Y. Chen, Z. Wang, B. Zang, and H. Guan. Fast In-memory Transaction Processing Using RDMA and HTM. *ACM Transactions on Computer Systems (TOCS)*, 35(1):3, 2017.

[11] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen. Fast and general distributed transactions using RDMA and HTM. In *EuroSys*, pages 26:1–26:17. ACM, 2016.

[12] H. Daly, A. Hassan, M. F. Spear, and R. Palmieri. NUMASK: high performance scalable skip list for NUMA. In *DISC*, pages 18:1–18:19, 2018.

[13] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. FaRM: Fast Remote Memory. In *USENIX NSDI*, pages 401–414, 2014.

[14] A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro. No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In *SOSP*, pages 54–70. ACM, 2015.

[15] B. Fan, D. G. Andersen, and M. Kaminsky. Memc3: Compact and concurrent memcache with dumber caching and smarter hashing. In *USENIX NSDI*, pages 371–384, 2013.

[16] B. Fitzpatrick. Distributed Caching with Memcached. *Linux journal*, 2004(124):5, 2004.

[17] InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 1 and 2*, 11 2007. Release 1.2.1.

[18] Intel. Intel xeon processor e7-8800/4800 v4 product families, 2016.

[19] Intel. Intel® memory latency checker v3.6, 2018.

[20] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur, et al. Memcached Design on High Performance RDMA Capable Interconnects. In *ICPP*, pages 743–752. IEEE, 2011.

[21] A. Kalia, M. Kaminsky, and D. G. Andersen. Design Guidelines for High Performance RDMA Systems. In *2016 USENIX ATC*, page 437, 2016.

[22] A. Kalia, M. Kaminsky, and D. G. Andersen. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided RDMA Datagram RPCs. In *USENIX OSDI*, pages 185–201, 2016.

[23] C. Lameter. Numa (non-uniform memory access): An overview. *Queue*, 11(7):40:40–40:51, July 2013.

[24] Y. Li. NUMA-aware Algorithms: the Case of Data Shuffling. In *CIDR*, Jan 2013.

[25] J. Mambretti, J. Chen, and F. Yeh. Next generation clouds, the chameleon cloud testbed, and software defined networking (sdn). ICCCRI, pages 73–79. IEEE, 2015.

[26] Mellanox. Maximizing server performance with mellanox socket direct adapter, 2018.

[27] Microsoft. Availability of h-series vms in microsoft azure, 2016.

[28] C. Mitchell, Y. Geng, and J. Li. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In *USENIX ATC*, pages 103–114, 2013.

[29] J. Nelson and R. Palmieri. On the performance impact of numa on one-sided rdma interactions. In *ICDCS '20 Poster paper*, 2020.

[30] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, et al. Scaling Memcache at Facebook. In *USENIX NSDI*, pages 385–398, 2013.

[31] S. Novakovic, Y. Shan, A. Rajaraman, and J. D. Ullman. Storm: a fast transactional dataplane for remote data structures. ACM, 2019.

[32] G. F. Pfister. An introduction to the infiniband architecture. In *High Performance Mass Storage and Parallel I/O*, chapter 42, pages 617–632. John Wiley & Sons, Inc., 2001.

[33] Y. Ren, T. Li, D. Yu, S. Jin, and T. Robertazzi. Middleware Support for RDMA-based Data Transfer in Cloud Computing. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 1095–1103. IEEE, 2012.

[34] Y. Ren, T. Li, D. Yu, S. Jin, and T. Robertazzi. Design and Performance Evaluation of NUMA-aware RDMA-based End-to-end Data Transfer Systems. In *SC*, page 48. ACM, 2013.

[35] T. Research. AMD Optimizes EPYC Memory with NUMA, 2018.

[36] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX*, 39(6), Dec. 2014.

[37] I. Smolyar, A. Markuze, B. Pismenny, H. Eran, G. Zellweger, A. Bolen, L. Liss, A. Morrison, and D. Tsafrir. Ioctopus: Outsmarting nonuniform DMA. In J. R. Larus, L. Ceze, and K. Strauss, editors, *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, pages 101–115. ACM, 2020.

[38] C. Wang, J. Jiang, X. Chen, N. Yi, and H. Cui. APUS: Fast and Scalable Paxos on RDMA. In *SoCC*, pages 94–107. ACM, 2017.

[39] Y. Wang, L. Zhang, J. Tan, M. Li, Y. Gao, X. Guerin, X. Meng, and S. Meng. HydraDB: a Resilient RDMA-driven Key-value Middleware for In-memory Cluster Computing. In *SC*, pages 1–11. IEEE, 2015.

[40] X. Wei, Z. Dong, R. Chen, and H. Chen. Deconstructing rdma-enabled distributed transactions: Hybrid is better! In *USENIX OSDI*, pages 233–251, 2018.

[41] M. Wu, F. Yang, J. Xue, W. Xiao, Y. Miao, L. Wei, H. Lin, Y. Dai, and L. Zhou. Gram: Scaling graph computation to the trillions. In *SoCC*, pages 408–421. ACM, 2015.

[42] K. Zhang, R. Chen, and H. Chen. NUMA-aware Graph-structured Analytics. *ACM SIGPLAN Notices*, 50(8):183–193, 2015.