In Proceedings of 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'20)

Evaluation of an InfiniBand Switch: Choose Latency or Bandwidth, but Not Both

M. R. Siavash Katebzadeh
University of Edinburgh
Edinburgh, United Kingdom
m.r.katebzadeh@ed.ac.uk

Paolo Costa
Microsoft Research
Cambridge, United Kingdom
paolo.costa@microsoft.com

Boris Grot
University of Edinburgh
Edinburgh, United Kingdom
boris.grot@ed.ac.uk

***Abstract*—Today's cloud datacenters feature a large number of concurrently executing applications with diverse intra-datacenter latency and bandwidth requirements. To remove the network as a potential performance bottleneck, datacenter operators have begun deploying high-end HPC-grade networks, such as InfiniBand (IB), which offer fully offloaded network stacks, remote direct memory access (RDMA) capability, and non-discarding links. While known to provide both low latency and high bandwidth for a single application, it is not clear how well such networks accommodate a mix of latency- and bandwidth-sensitive traffic that is likely in a real-world deployment.**

As a step toward answering this question, we develop a performance measurement tool for RDMA-based networks, RPerf, that is capable of precisely measuring the IB switch performance without hardware support. Using RPerf, we benchmark a rack-scale IB cluster in isolated and mixed-traffic scenarios. Our key finding is that the evaluated switch can provide either low latency *or* high bandwidth, but not both simultaneously in a mixed-traffic scenario. We evaluate several options to improve the latency-bandwidth trade-off and demonstrate that none are ideal.

***Keywords*-InfiniBand, Datacenter Networks, Quality-of-Service**

I. Introduction

Cloud datacenters feature an ever-growing mix of traditional and emerging applications that place high-performance demands on the datacenter network. Some applications, including those relying on disaggregated memory [1]–[6] and distributed in-memory storage [7]–[15], mandate ultra-low network latency to provide the illusion of a scale-up system. These applications produce short flows with small message sizes and are responsible for a minority of bytes sent/received inside a datacenter. In many instances, such as memory disaggregation, achieving the lowest possible per-packet latency (on the order of a few microseconds) is critical to the success of a service [1]. In such scenarios, increased network latency directly translates to diminished service quality. Moreover, if the processing is distributed across multiple nodes, it is not enough to achieve low average latency as the slowest node determines the actual latency of task completion. For that reason, tail latency (e.g., 99th or 99.9th percentile) is the metric of interest [16]. Meanwhile, other applications, such as big-data analytics using Hadoop or Spark [17]–[19], distributed machine-learning training [20]–[25], data backup and VM migrations, employ a bulk communication model that requires exchanging large amounts of data among the nodes, necessitating high bandwidth.

To meet the network latency and bandwidth needs, datacenter operators [26] have begun deploying high-end networking solutions in the form of InfiniBand (IB) [27]. Initially developed for the HPC domain, these networks tend to combine custom fully offloaded network stacks, RDMA capability and lossless links to provide high end-to-end performance. A number of recent works have demonstrated that, indeed, IB-based deployments can offer low latency (order of microseconds) and high bandwidth for a given cloud application, such as a distributed in-memory KVS [12], [28]–[30]. In most of these cases, network parameters are tuned for an individual application to harness the maximum potential of an IB fabric [31]. In practice, however, in public and private datacenters, multiple applications with different latency and bandwidth demands might coexist in a cluster and share the network [32].

In this work, we aim to answer the following question: *How well do the existing IB switches support the resulting mix of latency- and bandwidth-intensive traffic?*

To answer this question, we study a rack-scale IB deployment with a single top-of-rack (ToR) switch, which represents the simplest fieldable cluster setup. Evaluating a switch in such a rack-scale setup requires an accurate measurement methodology, which is able to assess the latency of the switch under stress in isolation (i.e., without end-point processing overheads). We observe that existing performance measurement tools for RDMA-based networks suffer from end-point (local and remote) processing overheads that impact precise latency measurement.

To mitigate these overheads, we develop RPerf, a high-precision RDMA performance measurement tool. RPerf overcomes deficiencies of existing tools to precisely measure latency and avoids the need for expensive hardware-based solutions or support for hardware timestamping on the NICs.

Using RPerf, we observe that our IB setup achieves very low latency in an unloaded network, corroborating prior work. We are further able to achieve consistently high bandwidth utilization while varying the number of bandwidth-intensive flows. However, we find that our IB switch is unable to provide low latency to a latency-sensitive flow in the presence of bandwidth-intensive flows. To enable low latency without compromising throughput, we consider several strategies, including using different packet sizes and priority levels but find all evaluated techniques deficient. We also use an IB switch simulator to explore different packet scheduling policies in the IB switch and observe that readily-available packet scheduling policies, such as First Come First Serve and Round-Robin, are unable to guarantee performance isolation for both latency- and bandwidth-intensive flows.

Based on our findings, we conclude that the contemporary IB gear (NICs and switch) used in our evaluation may be effective for applications with homogeneous traffic, but is unable to accommodate heterogeneous demands of modern datacenters.

The rest of the paper is structured as follows: Section II provides essential background information about IB and RDMA. Section III first explores the challenges in performance measurement of RDMA-based networks, then reviews existing measurement tools and their pitfalls. Section IV introduces an accurate micro-benchmarking tool, which provides sub-microsecond precision measurement. Section V describes our experimental setup. Section VI evaluates the latency and bandwidth of one-to-one setup, with and without our IB switch. Section VII evaluates the IB switch in the presence of mixed-type flows. Section VIII shows different approaches we take to satisfy both latency and bandwidth demands and describes the weaknesses of each approach.

II. Basics of InfiniBand and RDMA

To answer the need for both high-performance and low-latency, datacenter operators have started deploying high-performance network gear such as IB. In an IB fabric, servers are equipped with RDMA-enabled NICs (RNICs), which connect servers through IB switches. By providing support for RDMA capability, IB removes network software stack overheads, eliminates context switches and avoids the need for software execution on the remote CPU. High bandwidth, data integrity and reliability are other important aspects that make IB well suited for high-end datacenter networks.

Several factors impact the performance of an IB network, including the type of RDMA primitives, the choice of transport and the quality-of-service (QoS) configuration. In this section, we provide some background on these factors and explain how they interact with each other.

A. RDMA verbs

In RDMA terminology, a *verb* defines the type of a communication operation. There are two types of verbs: *two-sided* (SEND, RECV) and *one-sided* (READ, WRITE). Two-sided verbs involve both communication end-points. In such a communication, the remote host needs to pre-post RECVs and the local host *posts* SENDs. In contrast, one-sided verbs involve only one communication end-point (the source). Thus, using a WRITE one-sided primitive, the local host can write the data directly to the remote host's memory region. The local host can also use READ to fetch data from the remote host memory region without notifying and involving the remote host.

RDMA verbs follow the asynchronous I/O model, in which data transfers are non-blocking, hence allowing the application to continue execution before a posted request has finished. With both one-sided and two-sided verbs, when a request has finished, the RNIC (optionally) issues a completion signal (CQE) to an application-visible *completion queue* to notify the host. The application can receive the CQE signal by *polling* the completion queue.

B. RDMA transport

RDMA provides both unreliable (UD) and reliable (RC) transport types. UD transport does not guarantee delivery of requests. Moreover, UD provides only two-sided verbs. With RC transport, the RNIC uses acknowledgments to guarantee delivery of requests. In addition, RC transport supports both one-sided and two-sided verbs.

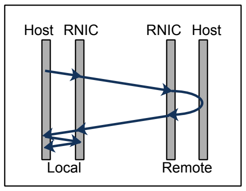
C. RDMA execution path

Depending on the choice of RDMA verb and transport type, an RDMA transaction follows a particular sequence of interactions between the communicating hosts. Figure 1 illustrates a complete sequence of interactions between hosts and RNICs for several RDMA verb and transport pairs.

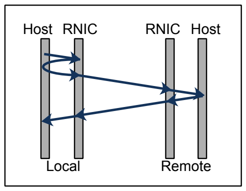
At the start of each transaction, regardless of the type of verb and transport used, the local host posts a request to the local RNIC via an MMIO transaction over PCIe. Based on verb type specified in the request, the local RNIC processes the request as follows:

READ: The local RNIC sends the request over the network fabric. The remote RNIC serves the request through a DMA read from the host's memory hierarchy and sends the data back to the local RNIC. Upon receipt of the data, the local RNIC issues a DMA write to store the data in local memory. Following that, the local RNIC performs another DMA write to issue a CQE (Figure 1a).

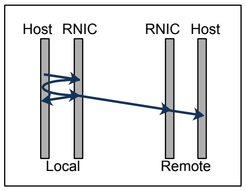
WRITE: First, the local RNIC fetches the payload through a DMA read. Next, the request is sent over the network fabric. The RNIC on the remote side performs a DMA write



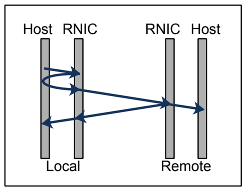
(a) READ operation using RC



(b) WRITE operation using RC



(c) SEND operation using UD



(d) SEND operation using RC

Figure 1: RDMA operations execution sequence.

to store the data in its host's memory and sends back an ACK. Upon the receipt of the ACK, the local RNIC uses a DMA write to issue a CQE (Figure 1b).

SEND: First, the local RNIC fetches the payload through a DMA read. Then, the request is sent over the network fabric. Once the remote RNIC receives the request, it sends back an ACK (in case of using RC transport) and writes the payload into its host's memory through a DMA write. Depending on the RDMA transport used by the SEND request, the local RNIC issues a CQE either as soon as the request is sent over the fabric (UD, Figure 1c) or once it receives the ACK from the remote RNIC (RC, Figure 1d).

D. InfiniBand QoS support

To provide per-flow performance differentiation, IB provides a set of priority levels, called *Service Levels (SLs)* that can be assigned to flows. For each SL, IB governs buffer allocation, flow control, queuing and scheduling. SLs are exposed to the application developer and are carried through the network in the header of IB packets.

IB uses the abstraction of SLs to hide two of its architectural components that help in achieving QoS:

1) Virtual Lane (VL): The concept of VLs allows a physical link to be divided into different logical communication links, each with its own buffering, flow-control, and congestion management resources. A VL arbiter controls the bandwidth usage by selecting flows according to the VL arbitration table. IB specification specifies that each port must have a minimum of two and a maximum of 16 VLs [27].

2) Virtual Lane Arbitration (VLArb): Every network component in the subnet of an IB fabric has a Service Level to Virtual Lane (SL2VL) mapping table which specifies the VL and priority for each packet. The SL to VL mapping and the priorities set for each VL are configurable in every IB switch.

Further, because IB implements a hop-by-hop credit-based flow control, whereby a sender does not send packets beyond the credit amount that has been advertised by the receive buffer on the opposite side of a link, the transport layer guarantees lossless communication. The combination of differentiated services and credit-based flow control helps IB achieve performance isolation.

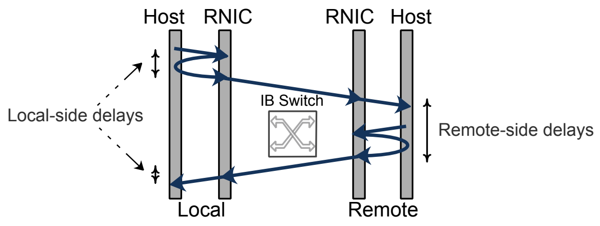


Figure 2: Ping-pong style RTT calculation.

III. IB SWITCH LATENCY MEASUREMENT

Thanks to fully offloaded network stack processing and other high performance features summarized in Section II, IB achieves sub-10 microsecond latency in an unloaded network. Such low latency presents several challenges for accurate NIC-to-NIC latency measurement.

The main challenge is isolating the latency of the switch from other components, particularly the software and PCIe. An ideal solution is to directly measure one-way port-to-port latency through the switch; doing so, however, requires the use of expensive data acquisition devices [33]. Another option is to use precise sub-microsecond clock synchronization at the end-points [34]; however, this approach relies on the assumption that one-way latencies in both directions are the same, which is not the case under congestion, particularly with a converged traffic pattern.

An alternative approach for latency measurement is a ping-pong style test to find the round-trip time (RTT) in software (Figure 2). Problematically, RTT calculation can be biased by *remote-side processing*, which is required to generate and transmit the response packet at the remote end. Such remote-side processing includes the software overhead for generating the response and the PCIe transactions necessary for transferring data to/from the RNIC. This remote-side processing delay does not reflect true network latency and, as such, should be excluded from the measurement. In addition, RTT calculation suffers from *local-side processing* delays. Local-side processing includes multiple PCIe transactions that the local RNIC performs to fetch data from the host memory, putting the data into packets and enqueuing packets. As software captures the posting time of a request not transmiting time, the calculated RTT includes PCIe transactions, RNIC processing and queuing delays, and results in biased measurement.

Related work

While several methodologies and tools are available for measuring the performance of datacenter networks [34]–[37], only a few RDMA-based tools are available for measuring the latency over an IB fabric. Unfortunately, none of the existing tools can precisely measure the latency of an IB switch, particularly under load.

RDMA Bench [38] benchmarks an RDMA-based fabric from the application layer and does not isolate the NIC-to-NIC latency. Perftest [39] consists of a collection of microbenchmarks that use a ping-pong latency measurement approach. In Perftest, the remote-side responds to a ping with a pong generated in software; consequently, it suffers from the remote-side processing problem. QPerf [40], another microbenchmarking tool, calculates the RTT at a high load using a *post-poll* measurement approach. In the post-poll approach, a QPerf client posts a WRITE request and then polls for the completion of the request; therefore, the QPerf server does not respond to the request in software. Such an approach removes the remote-side software overhead; however, it still includes the PCIe delays for DMA-ing the data into remote memory, which is required for a WRITE request (Figure 1b). QPerf also fails to perform precise tail latency measurement (it does not track per packet latency) and only reports the average latency. The accuracy of both Perftest and QPerf is further diminished because both tools include the local-side processing delays in their measurements. To conclude, existing measurement tools fail to factor out local-side and/or remote-side processing overheads, which impedes their ability to accurately measure the latency through the switch.

IV. RPerf

As existing methodologies fail to provide accurate latency estimates through an IB switch, we propose *RPerf*, a microbenchmarking tool capable of precisely measuring latency of an IB switch across a range of loads.

RPerf design details: RPerf measures the RTT between local and remote hosts, and leverages RDMA verbs in order to accurately measure the latency without including endpoint delays. We next describe key aspects of RPerf's design.

1. Excluding remote-side processing: In order to exclude all software overheads at the remote end, RPerf adopts the post-poll approach, in which only the local host posts a request and polls for the completion. By leveraging the RC transport, in which the remote RNIC generates a response without involving the destination host, RPerf avoids software processing overheads at the remote end. To exclude the PCIe latency on the remote end, RPerf uses the SEND verb. SENDs cause the remote RNIC to generate a response to the source RNIC immediate upon the receipt of the request and without waiting for the PCIe transaction to complete at the remote end (see Figure 1d).

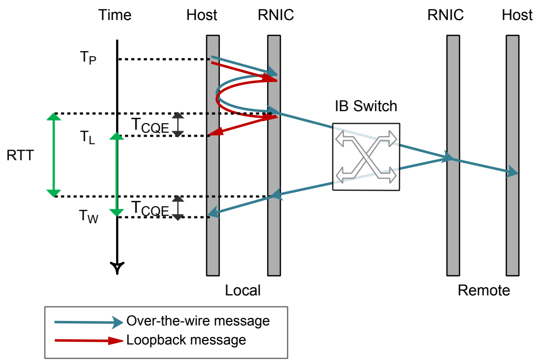


Figure 3: RTT calculation by RPerf.

The combination of using post-poll and RDMA SENDs avoids biasing latency measurements with remote-side software overheads and PCIe delays.

2. Excluding local-side processing: When the application posts a request using one of the IB verbs, the RNIC handles the request asynchronously and the control returns to the application. Meanwhile, the host sends the request to the local RNIC through PCIe and the local RNIC performs a DMA read to fetch the data for the SEND operation (Figure 1d). After fetching data, the local RNIC processes, enqueues and eventually transmits the request. The sum of latencies incurred by these actions at the local host and the RNIC make up the *local-side processing overhead.*

In order to avoid including local-side processing delays into RTT, RPerf calculates local-side processing overhead for every SEND request so that it can be excluded from the measurement of the switch latency. To do so, RPerf leverages *loopback* messages, which are messages that are sent from a host to itself via the local RNIC. Specifically, after sending a SEND to the destination host (which we call an over-the-wire SEND), RPerf immediately generates a loopback SEND request at the local host and times it. The latency of the loopback request is the local-side processing overhead, which can then be subtracted from the latency of the over-the-wire SEND.

3. RTT calculation: Using the ideas introduced above, we now describe how RPerf precisely measures the RTT through an IB switch. The process for RTT measurement and calculation is shown in Figure 3. At the outset, the local host posts a pair of SEND requests: an over-the-wire request and a loopback, storing the posting time (T_P). While the over-the-wire request is being sent out, the loopback work request is processed by the local RNIC, which generates a CQE when it is finished; RPerf captures the completion time for the loopback request (T_L). When the ACK for the over-the-wire request arrives at the local RNIC, the RNIC issues a CQE. RPerf records this completion time as (T_W) and calculates the RTT as follows:

$$RTT = (T_W - T_P) - (T_L - T_P) = T_W - T_L \quad (1)$$

By subtracting the time a loopback message takes to be completed, RPerf effectively identifies and removes the time taken by the over-the-wire request to be processed at the local RNIC as well as the local PCIe latency.

Additional details: In order to minimize software-induced performance variability, each RPerf thread is pinned to a CPU core and Huge Pages are allocated for all required buffers. For capturing the timestamps of events accurately, RPerf uses Time Stamp Counter through *rdtsc* x86 assembly instruction, which offers high-accuracy timestamping measurement within user-space. RPerf follows Intel recommendations for TSC calibration and access [41]. Multiple instances of RPerf can be run on different servers, and a user can specify a traffic pattern (e.g., one-to-one or many-to-one) to measure specific aspects of the system, such as zero-load latency, peak bandwidth or latency at load.

V. Experimental Setup

In this section, we present the details of our hardware testbed, simulator and traffic pattern.

Hardware testbed: For our tests, we use seven identical hosts with dual-socket Intel Xeon E5-2630 v4 (Broadwell) CPUs at 2.20GHz and 64GB RAM. All hosts run Ubuntu server 18.04 LTS with kernel version 4.15.0-50. Each host is equipped with an IB Mellanox MT27700 ConnectX-4 RNIC [42]. The RNICs are connected via a Mellanox SX6012 IB switch with 12 QSFP ports, 16 MBs of buffer capacity per port, and 9 VLs [43]. The switch and the RNICs have a peak bandwidth of 56Gbps. Mellanox reports up to 200ns port-to-port latency through the switch.

Simulator: We use a modified version of IB OMNeT++, originally developed by Mellanox. Similar to the hardware testbed, the simulator models seven nodes connected to an IB switch. The modeled switch provides two different packet scheduling policies: First Come, First Served (FCFS) and Round-Robin (RR). All other parameters such as peak bandwidth, port-to-port latency and number of VLs, are set according to our real IB switch.

Traffic pattern: One of the seven nodes serves as a destination; the remaining six are sources that send messages to the destination. Not all sources are active in all tests.

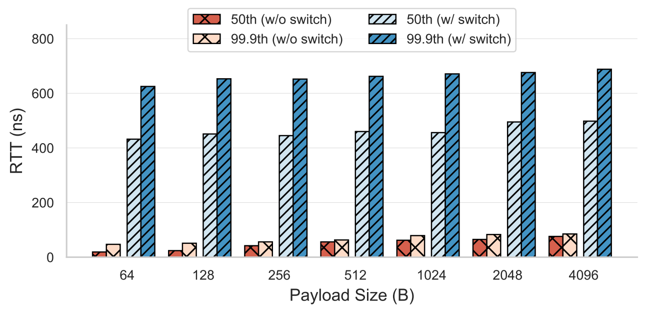


Figure 4: RTT calculated by RPerf for different packet sizes with and without the switch.

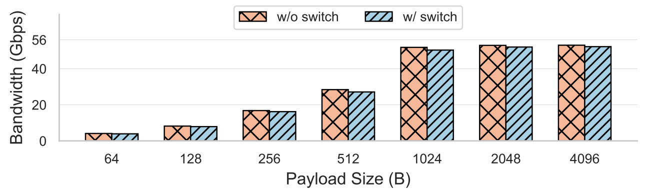


Figure 5: Bandwidth for different packet sizes with and without the switch.

Sources can be configured to use one of two types of traffic generators:

1. Bandwidth-Sensitive Generator (BSG): sends RC flows asynchronously (open-loop) and calculates the bandwidth during the tests. The message size varies in different experiments.

2. Latency-Sensitive Generator (LSG): sends RC packets synchronously (closed-loop). In the hardware testbed, an RPerf instance calculates the RTT using the methodology described in Section IV. The size of each message is 64B.

Metrics: We consider the 99.9$^{\text{th}}$ latency percentile as the tail. In all experiments, we run the test three times and the duration of each test is 15 minutes. All graphs plot the average values of the three runs; we do not plot standard errors, as they are negligible (below 0.001).

VI. Performance under one-to-one traffic

In this section, we study the performance of the switch in a one-to-one setup, whereby a single generator sends traffic to the destination server. First, we evaluate the isolated latency of the switch using RPerf by measuring the RTT with and without the switch, then we measure the end-to-end RTT using Perftest and Qperf.

A. Latency and bandwidth without the switch

We first measure the RTT without the switch by directly connecting the RNICs of a generator and the destination server.

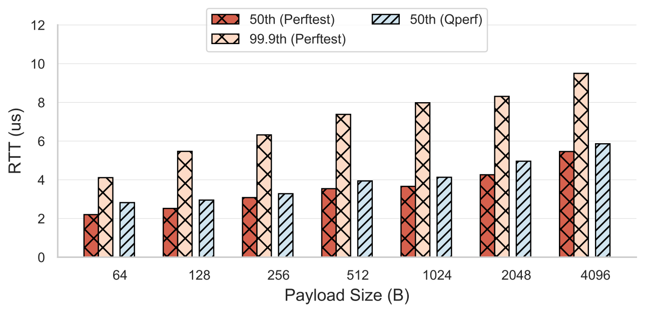


Figure 6: End-to-end RTT calculated by Perftest and Qperf for different packet sizes with the switch.

In the first test, we study the RTT at zero-load by having a server sending messages to the destination server. Figure 4 shows the RTT distributions of two hosts directly connected for different payload sizes. As Figure 4 illustrates, the median RTT for 64B is 20ns, and the tail RTT is 47ns. Furthermore, by increasing the message size to 4096B, the median and tail RTT grow to 76ns and 85ns, respectively. Additionally, Figure 4 shows that the difference between the median and tail RTT of a setup without the switch is at most 30ns. We observe that the RTT is very low and payload size has a small effect on RTT.

The second test evaluates the maximum achievable bandwidth with different payload sizes. In this test, the traffic generator is a BSG. As Figure 5 shows, the attained bandwidth varies with payload size. Using a payload size in the range of 1024B to 4096B, BSGs can achieve 51.8 to 53Gbps at the destination port, showing that with large payload sizes, our setup attains over 90% of bandwidth of a 56Gbps link. However, achieved bandwidth is very poor with small payloads; e.g., with 64B messages, the bandwidth is 4.1Gbps, meaning that less than 10% of link capacity is utilized. This problem is largely due to two reasons:
1) Header size of an IB packet can be up to 52B [27]; hence, less than 56% of the frame is the payload for a 64B message.
2) To achieve line rate bandwidth, here 56Gbps, the RNIC must be capable of processing ≈ 110 million 64B packet per second, which is beyond the RNIC's capability; this problem is well known and [38] discusses the reasons.

Take-aways:

1. Without the switch, the latency between a pair of RNICs connected back-to-back is extremely low, well under 100ns for all evaluated payload sizes.

2. While over 90% of link capacity can be achieved with large packet sizes, bandwidth utilization is poor with small packets.

3. RPerf is able to exclude almost all of end-points delays, which existing IB latency measurement tools fail to do.

B. Latency and bandwidth with the switch

Next, we study the performance of the IB switch. We connect two servers (one generator, one destination) through the switch and run the same one-to-one traffic pattern as above.

In the first test, we examine the RTT at zero-load for messages sent by the generator. As Figure 4 shows, the median RTT for 64B messages is 432ns and the tail is 625ns. Moreover, increasing the payload size to 4096B results in the median and tail RTT of 498ns and 688ns, respectively. We can observe that the measured RTT through the switch by RPerf is ≈ 432-498ns for 64-4096B payload sizes, which is close to the latency claimed by Mellanox (200ns one-way port-to-port latency, i.e. 400ns RTT [43]). Additionally, Figure 4 shows that regardless of the size of payload, the difference between the median and tail RTT through the switch is ≈ 200ns. By comparing to the no-switch setup, in which the difference between the median and tail RTT is at most 30ns, we can deduce that the switch introduces at least a 170ns delay to the tail RTT, which is about 45% of the median RTT; therefore, the switch suffers from tail latency, even at zero-load.

In the second test, we evaluate the maximum achievable bandwidth for a BSG over the switch with different payload sizes. Figure 5 illustrates the bandwidth achieved for different payload sizes over the switch. With payload size of 64B and 4096B, our setup attains 3.9 and 52.2Gbps, respectively. We can observe that the bandwidth achieved through the switch is slightly lower (by up to 0.8Gbps) than without the switch.

Take-aways:

1. Using RPerf, the median RTT through the switch is 432-498ns, depending on payload size. This latency is close to the expected 400ns round-trip latency per the switch spec.

2. The switch increases the tail latency to about 45% above the median latency.

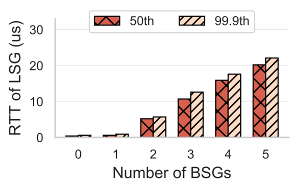
3. The switch has a negligible effect on the bandwidth of BSG in the one-to-one setup.

C. Latency calculation by existing tools

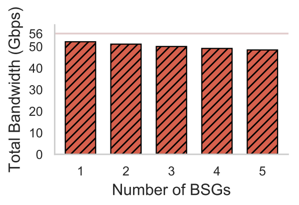
Finally, we measure the end-to-end RTT with the switch using Perftest and Qperf.

Figure 6 shows the RTT distributions of two hosts connected through the switch calculated by Perftest and Qperf for different payload sizes. In Figure 6, Perftest reports 2.20μs median RTT, and 4.11μs tail RTT for 64B. Furthermore, by increasing the message size to 4096B, the median and tail RTT grow to 5.46μs and 9.51μs, respectively. Figure 6 also shows that the median RTT reported by Qperf is 2.82μs for 64B. By increasing the message size to 4096B, the median grows to 5.85μs. Unfortunately, Qperf does not report tail RTT.

We observe that while Perftest and Qperf are useful for measuring end-to-end latency, their calculated latency is an order of magnitude higher than the reported latency in the switch specification. We conclude that these tools are unable



(a) RTT of LSG



(b) Total bandwidth of all BSGs

Figure 7: Converged traffic

to isolate the switch latency due to end-point overheads, especially when we increase the message size.

Take-away: Existing tools are unable to isolate the switch latency due to end-point overheads.

VII. Performance under converged traffic

In this section, we evaluate the network performance of the IB switch in a many-to-one setup. In this setup, varied number of BSGs (from one to five) send bandwidth-intensive flows with 4096B payload size to one destination server, forming a converged traffic pattern. Meanwhile, a LSG sends latency-sensitive flows to the same destination server.

Latency of LSG: Figure 7a shows the median and tail RTT of the LSG as we vary the number of active BSGs. With one active BSG, the LSG's median and tail RTT through the switch is 0.6μs and 0.9μs, respectively. Adding a second BSG increases the median and tail RTT of LSG to 5.2μs and 5.7μs, respectively. The third BSG further worsens the LSG's latency by increasing the median and tail RTT to 10.7μs and 12.6μs, respectively. As the figure shows, adding yet more BSGs further degrades the LSG's median and tail latency. Indeed, with each added BSG, the median RTT of the LSG increases by 4.8μs to 6.1μs, leading us to conclude that the switch fails to provide latency isolation in the presence of bandwidth-intensive flows, and that latency-sensitive flows are *unprotected*.

Bandwidth of BSGs: Figure 7b illustrates the total bandwidth achieved by BSGs as we vary the number of active BSGs. As Figure 7b shows, the bandwidth attained by one active BSG is 52.2Gbps. With two BSGs, each BSG achieves 25.5 to 25.6Gbps, resulting in the overall bandwidth of 51.1Gbps. When five BSGs are active, the bandwidth per BSG ranges from 8.9 to 9.9Gbps, with overall attained bandwidth of 48.4Gbps. While one would not expect the total bandwidth through the switch to vary as a function of the number of active BSGs, we observe that increasing the number of BSGs from one to five deteriorates the total achieved bandwidth of all BSGs by 7% (from 52.2 to 48.4Gbps).

Take-aways:

1. The latency observed by the latency-sensitive source is proportional to the number of active bandwidth-intensive

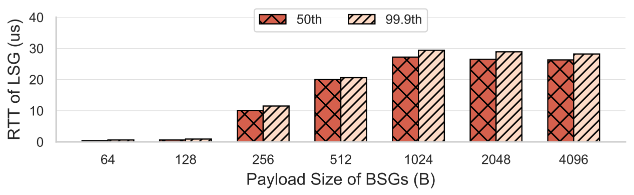


Figure 8: RTT of the LSG. Note that BSGs send different message sizes in each test.

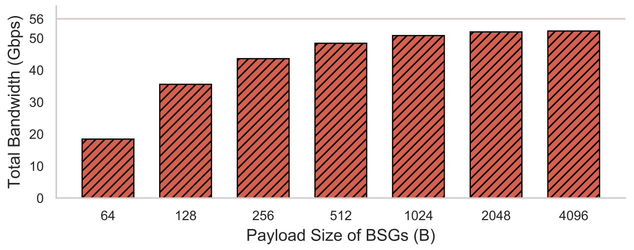


Figure 9: Total bandwidth achieved by BSGs as a function of the message size.

flows, indicating that the switch fails to provide latency isolation.

2. Increasing the number of convergent bandwidth-hungry flows diminishes the total achieved bandwidth through the switch.

VIII. Attempts to protect latency-sensitive flows

In Section VII, we observed that the IB switch fails to isolate latency-sensitive flows from bandwidth-intensive ones. In this section, we explore different approaches to help the switch in providing protection for latency-sensitive flows.

A. BSGs with different message sizes

Observing high latency for latency-sensitive flows in the presence of bandwidth-intensive ones, we hypothesize that the large message sizes used by our BSGs force packets from the LSG to wait for a long time while the large messages are transmitted. Thus, we set up an experiment to see whether using a smaller message size for the BSGs can improve the LSG's latency without sacrificing BSGs' bandwidth.

We direct five BSGs sending flows to one destination server. The payload size of BSGs varies in different tests. We also use *batching* with small payload sizes to improve the bandwidth utilization. At the same time, the LSG sends 64B messages to the same destination server.

Figure 8 shows the RTT of the LSG in the presence of flows from BSGs (with different payload sizes in different tests), and Figure 9 shows the overall bandwidth that the BSGs can achieve with different payload size.

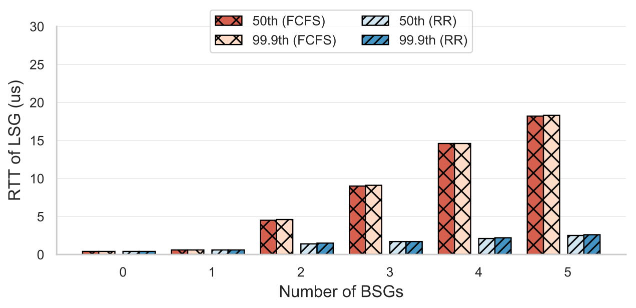


Figure 10: The impact of the number of BSGs on RTT of LSG in the simulator.

According to Figures 8 and 9, small payload sizes for BSGs lead to low LSG latency. For instance, when BSGs use a payload size of 64B, the median and tail RTT of LSG are 0.4μs and 0.6μs, and with 128B payload size of BSGs, the median and tail RTT of LSG are 0.6μs and 0.9μs. However, using small payloads for the BSGs sacrifices their ability to achieve high throughput. With 64B or 128B payload sizes, BSGs can barely utilize 35% or 70%, respectively, of link capacity at the destination port.

Meanwhile, if BSGs generate flows with large payload sizes, they can achieve high bandwidth utilization. Using a payload size in the range of 512B to 4096B, BSGs can achieve from 88% to 93% of link capacity at the destination port. However, choosing a large payload size for BSGs hurts the latency of the LSG. With a 512B payload size, the median and tail RTT of LSG are 20.0μs and 20.6μs. Larger payloads further worsen the median and tail RTT of LSG (26.3μs and 28.2μs for 4096B).

Take-away: By changing the payload size of BSG flows, we can achieve *either* low-latency for LSG flows *or* high-bandwidth for BSG flows, but not both at the same time.

B. *Packet scheduling policy at the switch*

In Section VIII-A, we observed that reducing the payload size of bandwidth-intensive flows does not resolve the latency-bandwidth trade-off. In this section, we take another step in an attempt to prevent latency-sensitive flows from being stalled by bandwidth-intensive ones. To this end, we study the impact of different in-switch packet scheduling policies on per-flow latency and bandwidth.

We consider a policy to be fair if the time each flow spends in the switch is proportional to the size of the flow. Thus, if the switch uses an unfair policy for packet scheduling, latency-sensitive flows might be stalled by bandwidth-intensive ones. Such a policy is unfair because it does not take flow size into account and fails to perform proportionally fair scheduling in each turn.

As the scheduling policy of our switch is not configurable, we use the IB simulator (Section V) to assess the effect of scheduling policies on fairness. We use the same setup as in the Section VII, with five BSGs sending 4096B flows and one LSG sending 64B messages to the same destination server. The IB simulator provides two different packet scheduling policies: First Come, First Served (FCFS) and Round-Robin (RR). We calculate the RTT of LSG flows in the converged setup using different packet scheduling policies and compare them with the real switch. Note that Mellanox documentation does not specify the scheduling policy implemented in our switch, so one of our goals is to understand the implemented policy.

FCFS policy: Figure 10 shows the median and tail RTT of the LSG as we vary the number of BSGs in the simulator under the FCFS scheduling policy. In the absence of a BSG, both the median and tail RTT of the LSG are 0.4μs. With one active BSG, both the median and tail RTT of the LSG are 0.6μs. Adding the second BSG increases the median and tail RTT of LSG to 4.5μs and 4.6μs, respectively. With five active BSGs, the median and tail RTT is 18.2μs and 18.3μs, respectively. We can observe that the median and tail RTT in the simulator are almost identical (0.1μs difference). Thus, unlike the real switch, simulator does not introduce significant tail RTT. The reason is that the switch uArch is not modeled in detail in the simulator; therefore the median and tail RTT of simulator are much closer, compared to the median and tail RTT of real switch.

In the simulator, each additional BSG adds a delay of 3.9μs to 4.6μs to the median RTT of the LSG. This trend closely matches the behavior observed with the real switch, where each additional BSG adds 4.6μs to 5.2μs to the LSG's latency.

To investigate the additional delay added by each BSG, we look into the architecture of the simulated switch. In the modeled switch, each input port has dedicated buffering used for absorbing bursts. With the FCFS policy, in each turn, the arbiter examines the packet at the head of each input buffer and chooses the oldest packet. In our converged traffic experiment, each BSG fills to capacity its respective input buffer. Once an LSG packet enters the switch, it too is enqueued at its port's input buffer. Following the FCFS policy, the arbiter selects the LSG packet only after *all* other packets present at the switch when the LSG's packet arrived have been scheduled. Therefore, in our setup, the minimum amount of time a LSG packet needs to wait can be computed as follows:

$$W_t = \frac{N \times BufferSize}{LinkBandwidth} \qquad (2)$$

where N is the number of BSG ports (i.e., whose input buffers are full), BufferSize is the size of each input buffer, and LinkBandwidth is the bandwidth of a link.

In the modeled switch, the size of each input buffer is 32KB, and the link bandwidth is 56Gpbs. In this case, each additional BSG adds 3.6μs waiting time to each LSG packet, which is close to the latency observed in both the simulator and the real switch.

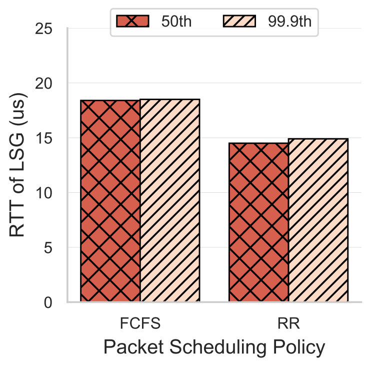


Figure 11: RTT of LSG in a multi-hop setup.

Take-away: With the FCFS policy, the simulator suffers from latency unfairness and behaves similar to the real switch.

Round-Robin policy: With RR policy, the arbiter in each turn selects a port and chooses the packet at the head of the port. In this case, whenever a LSG packets arrives, it waits for at most the number of active ports.

Figure 10 shows the median and tail RTT of LSG flows, when different numbers of BSGs are active in the simulator with the RR scheduling policy. Without any active BSGs, both the median and tail RTT of LSG are 0.4μs. With one active BSG, both the median and tail RTT of LSG are 0.6μs. By increasing the number of active BSGs to five, the median and tail RTT grow to 2.5μs and 2.6μs.

Unlike the experiment with FCFS policy, with RR policy, increasing the number of BSGs does not change the RTT for LSG dramatically.

Take-aways:

1. The measurements attained on the simulator with the RR policy are vastly different to those on the real switch. This further indicates that the real switch uses the FCFS scheduling policy.

2. Unlike the FCFS policy, the RR scheduling policy is more effective at protecting the latency-sensitive flow.

Packet scheduling policies in a multi-hop topology: At first glance it seems that the RR policy on the switch resolves the dilemma of isolation and protection of a latency-sensitive flow. However, can the RR policy continue to be effective in a multi-hop topology? To answer this question, we extend our simulated setup to a two-hop topology, where a pair of switches are connected together. Two BSGs and one LSG are connected to the upstream switch, and three BSGs are attached to downstream switch. The destination server is also attached to the downstream switch. All BSGs send 4096B messages to the destination server.

We calculate the RTT of LSG messages in the multi-hop setup using different packet scheduling policies and compare them with each other. In each test, the packet policy of *both* switches is either FCFS or RR.

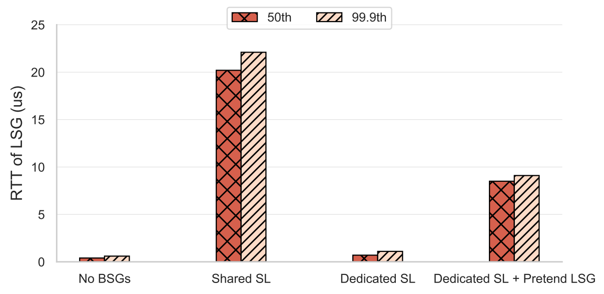


Figure 12: RTT of the real LSG in different setups.

Figure 11 shows the median and tail RTT of LSG, when different packet policy is used in the switches in the simulator. Using FCFS policy in both switches, the median and tail RTT of LSG are 18.4μs and 18.5μs. Using RR policy, the median and tail RTT of LSG are 14.5μs and 14.9μs.

We can observe that if a latency-sensitive flow shares a link (in this setup the link that connects two switches) with bandwidth-intensive flows, the RR policy is unable to protect the latency-sensitive flows. The reason is that the latency-sensitive flow will be queued at the same input buffer as the bandwidth-intensive flow in the downstream switch, and will hence suffer from head-of-line blocking.

Take-away: The RR policy fails to isolate latency-sensitive flows in a multi-hop setup.

C. *InfiniBand QoS*

The previous experiments motivate the need for separate buffer resources and scheduling priorities for latency-sensitive and bandwidth-intensive flows. Such a strategy allows the latency-sensitive flow(s) to avoid queuing alongside the bandwidth-intensive flows; therefore the bandwidth-intensive flows will not block latency-sensitive flows. IB QoS configuration provides such a mechanism through a combination of SLs and VLs.

To protect the latency-sensitive flows from the LSG, we want to assign a dedicated SL to latency-sensitive flows at the local host and map this SL to a high-priority VL in the switch. Due to the fact that the notion of latency-sensitivity is not defined in IB terminology, we consider small messages (up to 256B) as latency-sensitive flows.

The following experiment evaluates the effectiveness of using dedicated SLs for latency-sensitive and bandwidth-intensive traffic, by assigning SL0 to BSGs and SL1 to LSG flows. In the switch, SL0 is mapped to low-priority VL0, and SL1 is assigned to high-priority VL1.

Figure 12 shows the median and tail RTT for LSG traffic using a dedicated SL. As the figure shows, using a dedicated SL protects the latency-sensitive flows. While with the shared SL/VL (the same as Section VII) the median and tail RTT of LSG is 20.2μs and 22.1μs, with dedicated

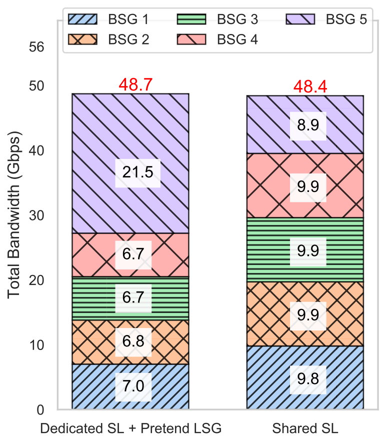


Figure 13: Total bandwidth achieved by BSGs under converged traffic.

SL/VL, LSG packets have 0.7μs median and 1.1μs tail RTT. The figure shows that using dedicated SL/VL improves the latency of LSG by $\approx$ 29X for the median and $\approx$ 20X for the tail RTT. Compared to the shared SL/VL setup, the RTT of LSG with dedicated SL/VL is closer to the RTT of LSG in the absence of BSGs (0.4μs and 0.6μs). Moreover, the total bandwidth achieved by five BSGs is the same as the bandwidth achieved without using a dedicated SL (Section VII), which indicates that this SL configuration does not introduce a throughput penalty.

Take-away: Differentiating flow type and assigning priority to each flow type can effectively protect latency-sensitive flows.

Gaming the dedicated SL/VL setup: While assigning a dedicated SL to small messages seems promising, a BSG may abuse (intentionally or not) this approach by pretending to be a LSG so as to achieve bandwidth higher than its fair share. To game the QoS, BSG sends large amounts of data segmented into small packets. To show how this BSG harms the bandwidth of other BSGs, we devise a test with a dedicated SL for latency-sensitive flows, in which the BSG pretending to be an LSG (referred to as a *pretend LSG*) sends 256B messages asynchronously. It further optimizes for throughput by using *batching*.

Figure 12 shows the median and tail RTT of LSG traffic, and Figure 13 illustrates the total bandwidth achieved by all BSGs and the pretend LSG, along with their share of the bandwidth. Figure 12 shows that pretend LSG hurts the latency of the real LSG (8.5μs median and 9.1μs tail RTT), as both pretend and real LSGs have the same SL/VL. As Figure 13 illustrates, the pretend LSG achieves 21.5Gbps bandwidth, while each of the other BSGs achieves 6.7 to 7Gbps. We can observe that a bandwidth-intensive source can pretend to be latency-sensitive and take three times higher bandwidth share compared to other bandwidth-intensive sources, leading to bandwidth unfairness. One might think that limiting the bandwidth for each SL/VL mapping will prevent gaming the SL/VL setup; however, imposing such a limit will hurt the latency of the LSG, specially when a burst of latency-sensitive packets arrives at a switch.

Take-away: By using IB QoS in the switch, latency-sensitive flows can be protected in an environment with different types of flows. The risk, however, is that it opens up the possibility of gaming to achieve a higher bandwidth share by bandwidth-intensive flows.

EVALUATION SUMMARY: Our evaluation reveals that the tested IB switch can either provide low latency to a latency-sensitive flow or high bandwidth for bandwidth-intensive flow(s), but not both simultaneously. We showed that our switch uses the FCFS scheduling policy that is particularly damaging for latency-sensitive flows in the presence of bandwidth-intensive flows. An alternative policy using Round-Robin improves fairness over FCFS, but only in a single-hop topology; with just two network hops, latency-sensitive packets can be blocked by other packets, inevitably hurting latency.

Motivated by these observations, we examined IB's QoS mechanism that assigns different service levels and virtual lanes to flows. While the results are encouraging in that a latency-sensitive flow can achieve low latency without compromising throughput of bandwidth-intensive ones, such an approach is prone to gaming. Specifically, we observed that a bandwidth-intensive flow can pretend to be a latency-sensitive one and burst small messages, which allows it to achieve a larger share of network bandwidth than other bandwidth-intensive flows that do not try to game the system.

IX. Conclusion

In this paper, we identify shortcomings in existing RDMA-based performance measurement tools and show why they are unable to accurately assess the latency of an IB switch. We introduce the RPerf performance measurement tool that leverages RDMA verbs to exclude end-point overheads and provide a highly accurate latency measurement for RDMA-based switches. Using the precise measurements enabled by RPerf, we analyze the latency and bandwidth of an IB switch in one-to-one and many-to-one traffic scenarios. We show that the switch fails to protect the latency-sensitive flows from bandwidth-intensive ones, and that the latency is proportional to the number of active bandwidth-hungry flows. We consider several strategies for improving latency fairness, including using small packet sizes for bandwidth-intensive flows and the use of IB's QoS mechanism, but find all evaluated approaches deficient in some respect. We thus conclude that better mechanisms are needed to provide performance isolation in a mixed traffic environment.

References

[1] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, pp. 249–264, 2016.

[2] V. Shrivastav *et al.*, "Shoal : A Network Architecture for Disaggregated Racks This paper is included in the Proceedings of the of Jerusalem," *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019*, pp. 255–270, 2019.

[3] K. Katrinis *et al.*, "Rack-scale disaggregated cloud data centers: The dReDBox project vision," in *Proceedings of the 2016 Design, Automation and Test in Europe Conference and Exhibition, DATE 2016*, ser. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, pp. 690–695.

[4] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with Infiniswap," *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*, pp. 649–667, 2017.

[5] M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei, "Remote memory in the age of fast networks," *SoCC 2017 - Proceedings of the 2017 Symposium on Cloud Computing*, pp. 121–127, 2017.

[6] L. Shuang, R. Noronha, and D. K. Panda, "Swapping to remote memory over InfiniBand: An approach using a high performance network block device," *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, p. nil, 2005.

[7] B. Fitzpatrick, "Distributed caching with memcached," *Linux Journal*, vol. 2004, no. 124, p. 5, aug 2004.

[8] 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 2014*. Seattle, WA: {USENIX} Association, 2014, pp. 429–444.

[9] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky, "SILT: A memory-efficient, high-performance key-value store," in *SOSP'11 - Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 1–13.

[10] Y. Mao, E. Kohler, and R. Morris, "Cache craftiness for fast multicore key-value storage," in *EuroSys'12 - Proceedings of the EuroSys 2012 Conference*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 183–196.

[11] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian, "The End of slow networks: It's time for a redesign," *Proceedings of the VLDB Endowment*, vol. 9, no. 7, pp. 528–539, 2016.

[12] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro, "FaRM: Fast remote memory," *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014*, pp. 401–414, 2014.

[13] 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 2016*. Savannah, GA: {USENIX} Association, 2016, pp. 185–201.

[14] C. Mitchell, K. Montgomery, L. Nelson, S. Sen, and J. Li, "Balancing CPU and network in the cell distributed b-tree store," in *Proceedings of the 2016 USENIX Annual Technical Conference, USENIX ATC 2016*. Denver, CO: {USENIX} Association, 2016, pp. 451–464.

[15] V. Gavrielatos, A. Katsarakis, V. Nagarajan, B. Grot, and A. Joshi, "Kite: Efficient and Available Release Consistency for the Datacenter," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–16.

[16] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[18] M. Zaharia *et al.*, "Apache spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, oct 2016.

[19] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *OSDI 2004 - 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, 2004, pp. 137–149.

[20] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image Classification at Supercomputer Scale," in *NeurIPS*, 2018.

[21] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," *arXiv preprint arXiv:1706.02677*, 2017.

[22] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik, "Scaling Distributed Machine Learning with In-Network Aggregation," *arXiv preprint arXiv:1903.06701*, 2019.

[23] S. Li, T. Ben-Nun, S. Di Girolamo, D. Alistarh, and T. Hoefler, "Taming Unbalanced Training Workloads in Deep Learning with Partial Collective Operations," *arXiv preprint arXiv:1908.04207*, 2019.

[24] H. Zhu, D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and M. Erez, "Kelp: QoS for accelerated machine learning systems," in *Proceedings - 25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019*. IEEE, 2019, pp. 172–184.

[25] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton, "Large scale distributed neural network training through online distillation," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[26] M. Azure, "Introducing the new HB and HC Azure VM sizes for HPC," 2018.

[27] I. T. ASSOCIATION, "Infiniband Architecture Specification Volume 1," 2015.

[28] V. Gavrielatos, N. Oswald, A. Katsarakis, B. Grot, A. Joshi, and V. Nagarajan, "Scale-Out ccNUMA: Exploiting Skew with Strongly Consistent Caching," *Proceedings of the 13th EuroSys Conference, EuroSys 2018*, vol. 2018-Janua, pp. 21:1—-21:15, 2018.

[29] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," *Computer Communication Review*, vol. 44, no. 4, pp. 295–306, 2015.

[30] A. Katsarakis, V. Gavrielatos, M. Katebzadeh, A. Joshi, A. Dragojevic, B. Grot, and V. Nagarajan, "Hermes: a fast, fault-tolerant and linearizable replication protocol," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 12 2019.

[31] Y. Zhang, J. Gu, Y. Lee, M. Chowdhury, and K. G. Shin, "Performance isolation anomalies in RDMA," in *KBNets 2017 - Proceedings of the 2017 Workshop on Kernel-Bypass Networks, Part of SIGCOMM 2017*, ser. KBNets '17. New York, NY, USA: ACM, 2017, pp. 43–48.

[32] Y. Zhu *et al.*, "Congestion control for large-scale RDMA deployments," *SIGCOMM 2015 - Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, vol. 45, no. 5, pp. 523–536, aug 2015.

[33] N. Zilberman, M. Grosvenor, D. A. Popescu, N. Manihatty-Bojan, G. Antichi, M. Wójcik, and A. W. Moore, "Where has my time gone?" in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10176 LNCS, 2017, pp. 201–214.

[34] D. A. Popescu and A. W. Moore, "PTPmesh: Data Center Network Latency Measurements Using PTP," in *Proceedings - 25th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2017*. IEEE, sep 2017, pp. 73–79.

[35] Y. Zhang, D. Meisner, J. Mars, and L. Tang, "Treadmill: Attributing the Source of Tail Latency through Precise Load Testing and Statistical Inference," *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, pp. 456–468, 2016.

[36] C. Guo *et al.*, "Pingmesh: A Large-scale system for data center network latency measurement and analysis," in *SIGCOMM 2015 - Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, vol. 45, no. 4. ACM, 2015, pp. 139–152.

[37] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, "Simon: A simple and scalable method for sensing, inference and measurement in data center networks," in *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019*, 2019, pp. 549–564.

[38] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," *Proceedings of the 2016 USENIX Annual Technical Conference, USENIX ATC 2016*, pp. 437–450, 2016.

[39] OFED, "perftest," [Online]. Available: https://github.com/linux-rdma/perftest.

[40] OFED, "qperf," [Online]. Available: https://github.com/linux-rdma/qperf.

[41] G. Paoloni, "How to Benchmark Code Execution Times on Intel ® IA-32 and IA-64 Instruction Set Architectures," *Intel Manual*, vol. 123, no. September, pp. 1–37, 2010.

[42] Mellanox, "ConnectX®-4 VPI IC PRODUCT BRIEF," [Online]. Available: http://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX-4_VPI_IC.pdf.

[43] Mellanox, "Mellanox SwitchX and SwitchX®-2 1U Switch and Gateway Systems Hardware User Manual."