EFA: A Viable Alternative to RDMA over InfiniBand for DBMSs?

Tobias Ziegler
Technical University Darmstadt

Dwarakanandan B. M.
Technical University Darmstadt

Viktor Leis
Friedrich-Alexander-Universität Erlangen-Nürnberg

Carsten Binnig
Technical University Darmstadt

ABSTRACT

RDMA over InfiniBand offers high bandwidth and low latency which provides many benefits for distributed DBMSs. However, in the cloud RDMA over InfiniBand is still not widely available. Instead, cloud providers often invest in their own high-speed networking technology and start to expose their own native networking interfaces. For example, the largest cloud provider, Amazon Web Services (AWS), introduced instances with Elastic Fabric Adapter (EFA) in 2018. In this paper, we aim to analyze EFA as an alternative to RDMA in the cloud by performing an in-depth and systematic evaluation. As a first contribution, we describe the EFA stack and summarize the main differences to RDMA over InfiniBand. Second, we evaluate the performance of EFA and compare it with RDMA over InfiniBand in a set of reproducible benchmarks. Third, based on our evaluation we derive lessons learned for DBMS designers.

ACM Reference Format:
Tobias Ziegler, Dwarakanandan B. M., Viktor Leis, and Carsten Binnig. 2022. EFA: A Viable Alternative to RDMA over InfiniBand for DBMSs?. In *Proceedings of (DaMoN '22)*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

InfiniBand offers low network latency. Remote Direct Memory Access (RDMA) over InfiniBand offers single-digit microsecond network latencies at extremely high bandwidth. As such RDMA over InfiniBand had a tremendous impact on distributed machine learning [16, 26, 36, 48], high performance computing [24, 32, 34], and distributed database systems [3, 7, 17, 18, 21, 23, 30, 31, 35, 44–46, 49–51, 53]. In fact, distributed system designs that were previously thought to be inefficient had to be re-evaluated [7, 44, 49, 53]. For instance, distributed shared memory architectures are becoming increasingly common due to the low network latencies. This is evidenced by several distributed database systems that expose their aggregated memory via RDMA such as FaRM [5, 6, 42], NAM-DB [49], or Tell [25]. Unfortunately, to get the benefits of RDMA, specialized and expensive hardware is required.

InfiniBand is not widely available in the cloud. At the same time, many companies transition from on-premise to the cloud and are dependent on currently-available cloud offerings. Unfortunately, even though cloud providers offer hundreds of heterogeneous instance types, InfiniBand is *not widely available* in public clouds. In fact, of the three major cloud providers, only Microsoft Azure offers InfiniBand for a very limited set of instance families [29] (i.e., instance types of H, HB, HC, and some of the N series). This limited availability takes away one of the main benefits of the cloud; a broad hardware landscape to satisfy every application requirement. And although several vendors offer networking that achieves the same bandwidth as RDMA over InfiniBand (100 Gbit), the low network latencies of InfiniBand are still unmatched.

AWS's low latency network EFA. As an alternative to RDMA over InfiniBand, in 2018 the largest cloud provider, Amazon Web Services (AWS), introduced instances with *Elastic Fabric Adapter (EFA)*. EFA is a network interface for Amazon EC2 instances that is specifically designed to achieve low latencies in the cloud. Today, EFA is widely available in AWS, and Amazon presently offers 15 instance families with EFA support [39] to satisfy different application requirements, e.g., memory optimized (r5dn, r5n) or storage optimized instances (i3en).

No systematic evaluation of EFA. Surprisingly, even though EFA has become broadly available in AWS, there has not yet been a systematic evaluation of this technology from the data management community. The limited research that addresses EFA was driven by the HPC community and specifically geared towards MPI in distributed HPC applications [4, 41, 47]. However, MPI is a higher-level library on top of the EFA networking stack and was built with different assumptions that do not necessarily match those of data-intensive systems [1, 22, 44]. Additionally, there are no official hardware specifications published by AWS which makes it difficult to optimize a system for utilizing the hardware optimally. Therefore, the following question remains: *Can EFA be a viable alternative to RDMA over InfiniBand for data-intensive systems?*

Contributions. In this paper, we answer this question by performing an in-depth systematic evaluation. First, we describe the EFA stack and highlight qualitative differences to RDMA over InfiniBand which we think is an already highly valuable contribution for system design since today the information about EFA is scattered across many resources, Second, we evaluate the performance of EFA and compare it with RDMA over InfiniBand in a set of reproducible microbenchmarks. Third, based on our evaluation we derived lessons learned that can be used by system designers to utilize EFA in data-intensive systems.

2 ELASTIC FABRIC ADAPTER

In this section, we describe the EFA stack as shown in Figure 1. We do this from the bottom up, starting with the hardware and Amazon's *Scalable Reliable Datagram* (SRD) protocol. Afterwards, we move on to the software layers *ibverbs* and *libfabric*.

2.1 SRD: A Reliable and Unordered Protocol

SRD is reliable. The foundation of EFA is SRD, a proprietary network protocol that provides reliable but unordered communication on top of commodity Ethernet switches [41]. Of course, reliability is not unique to SRD as many common protocols, including TCP/IP and reliable connected RDMA, guarantee reliable packet delivery. However, unlike existing Ethernet protocols such as TCP/IP, SRD is

DaMoN '22, June 12–17, 2022, Philadelphia, USA
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

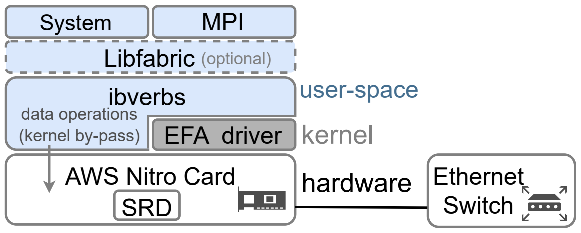

Figure 1: EFA Stack: Software and Hardware.

purpose-built for Amazon's data centers. Therefore, Amazon controls the hardware SRD operates on, which in turn enables them to implement SRD's reliability efficiently in hardware (in the AWS Nitro network cards [38]) rather than in software.

SRD is out-of-order. Moreover, unlike other reliable protocols that often guarantee in-order delivery, SRD has out-of-order delivery. There are two main reasons for this decision: (1) Only a few applications require in-order delivery and thus the application layer should handle in-order delivery if needed. (2) Out-of-order delivery reduces tail latencies. This is because in-order delivery may cause head-of-line blocking [41]. Additionally, to avoid hot paths in the network and thus reduce the chance of packet drops, SRD packets are sent across multiple network paths. Thus, out-of-order delivery is a direct consequence of sending packets across multiple paths, and enforcing the order would require large intermediate buffers or dropping out-of-order messages.

2.2 Ibverbs: Low Level Interface

SRD is exposed to the application via the *ibverbs* library. *Ibverbs* is the same library that allows user-space processes to use RDMA primitives to perform high-throughput, low-latency network operations on InfiniBand. The fact that EFA and RDMA over InfiniBand share the same low-level library is no coincidence because EFA closely resembles the InfiniBand verbs specification [41]. The library itself is split into a control path and a data path.

Control path. The control path is implemented through system calls to the kernel, which further calls the low-level EFA driver depicted gray in Figure 1. The control path creates, modifies, queries, and destroys resources that are needed for connection setup and communication. Because the control path may interact with the kernel, those operations are commonly avoided in the hot path.

Data path. The data path avoids the kernel and network stack completely and instead interacts directly with the hardware. The reason for kernel and network stack bypass is that the traditional kernel network stack was shown to be prohibitively expensive and thus ill-suited for high-performance networks [3, 8]. This is evident by the rise of available frameworks such as DPDK [33], mTCP [15], or eRPC [20]. Furthermore, *ibverbs* is designed to provide zero-copy and asynchronous networking.

Asynchronous networking. To implement asynchronous networking two designs are possible: (1) Copy data from the application buffer into an intermediate buffer and return to the application. Thus, the application buffer can be reused instantly. However, this approach comes with the overhead of copying data and is often prohibitively expensive [8]. (2) To avoid copies, modern network interfaces get the data buffer from the application and notify the application when data is actually sent, via traditional synchronization mechanisms such as completion queues. Unlike in the first option, the buffer cannot be immediately reused but only after the completion event is received.

Send/Receive queues. EFA follows the second approach: send/receive queues with the respective completion queues. (1) The *send queue* is used by the sender to issue a send operation that returns immediately to achieve asynchronous networking. (2) The *receive queue* is used by the receiving side to issue receive requests. A receive request instructs the network interface card (NIC) in which memory buffer to copy the incoming data via direct memory access (DMA). Therefore, to handle incoming messages the application needs to post such receive requests beforehand.

No one-sided. As mentioned earlier, because EFA and RDMA over InfiniBand share the same low-level interface they have a common intersection of primitives. However, there are important distinctions between EFA and reliable RDMA over InfiniBand. Reliable connected RDMA provides two types of operations: One-sided *read/write/atomic* primitives and two-sided primitives *send/recv*. EFA does not support one-sided *read/write/atomic* primitives [41], but is limited to two-sided primitives. This means when applications heavily rely on one-sided operations, they have to be redesigned when porting them to EFA.

Connection scalability. On the other hand, SRD offers better scalability because a well-known limitation of reliable connected (RC) RDMA is lifted. That is, in RC RDMA, one connection endpoint can exclusively communicate only with one other end-point This means to achieve an all-to-all communication many connections need to be established. Having a large number of connections can lead to performance degradation because the NIC cannot cache all connections [19]. In contrast, in EFA one connection end-point can communicate with all other end-points without creating an endpoint for every connection. This reduces the number of required connections in EFA drastically [41].

2.3 Libfabric: EFA's Programming Interface

Even though *ibverbs* provides all available primitives, the recommended interface for application developers is *libfabric* [37], which provides a higher level interface on top of *ibverbs* as shown in Figure 1. *Libfabric* is a framework that provides a unified abstraction for high-performance network devices [10]. In contrast to *ibverbs*, *libfabric* offers higher level functionality and is somewhat easier to use. It provides a standard set of APIs that are agnostic to the underlying network protocol and hardware device.

Provider. In *libfabric*, hardware devices are termed as *provider* which hook into the framework and implement optimized device-specific functionality. EFA is essentially one such provider under *libfabric* [13] (called *fi_efa*). This EFA provider is layered above *ibverbs* and thus also provides control and data transfer operations. Because the data operations eventually use *ibverbs*, they bypass the kernel completely (see Figure 1) and interact with the send and receive queues of the NIC. Those data operations include the actual data transfer between communication end-points.

Endpoint types. Internally, endpoints use the primitives provided by *ibverbs* and may expose some higher- level functionality to the application developer. There are two types of communication end-points [12] exposed for EFA: An unreliable datagram (DGRAM) endpoint, with a maximum datagram size equaling the maximum

Table 1: SRD compared to reliable connected RDMA and traditional TCP/IP Sockets.

SRD (EFA)	RC RDMA (IB)	Sockets (TCP/IP)
Ethernet	InfiniBand	Ethernet
reliable	reliable	reliable
Messages	Messages	Stream
unordered	ordered	ordered
user-space	user-space	kernel-space
asynchronous	asynchronous	synchronous
no one-sided	one-sided	no one-sided

transmission unit (MTU) of the underlying layer. A reliable datagram (RDM) that exposes the proprietary SRD protocol and is thus reliable, but unordered as we discussed in Section 2.1. In the RDM endpoint, *libfabric* implements higher-level features such as message segmentation in the software layer. As such RDM supports data transfers that are larger than the MTU.

Endpoint capabilities. Both endpoint types support the native send/recv operations. If an endpoint supports those send/recv operations it is called to support the FI_MSG capabilities in *libfabric*. However, when looking at the documentation [12] of the EFA provider one can see that the RDM endpoint supports various other capability types. One such capability is FI_RMA, which is an umbrella term for all remote memory operations including one-sided read/write. One-sided operations can directly access the remote memory without any involvement of the remote CPU (OS by-pass). As discussed before, this does not mean that EFA natively supports one-sided operation. Instead, *libfabric* emulates the behavior of one-sided operations [9] in the software layer. This simplifies porting applications from one network device that supports one-sided operations to another device that does not support them. However, since *libfabric* emulates the behavior of one-sided operations for EFA, the remote CPU is involved in data transfers; i.e., a thread on the remote side emulates one-sided behavior. Moreover, as most high-performance applications including DBMSs manage their resources themselves, the emulated FI_RMA capability may interfere. There is a possibility to change the default behavior [11] (FI_PROGRESS_AUTO) and let the application threads handle the progress (FI_PROGRESS_MANUAL), but doing so would mostly require the application to mimic *libfabric*'s functionality.

2.4 Comparison to Reliable RDMA and Sockets

Table 1 summarizes the key differences between reliable EFA, reliable connected RDMA over InfiniBand, and traditional TCP/IP sockets. As a consequence of their similar design objectives (low latency and high bandwidth), we can observe that EFA and RDMA share many common characteristics. Both are designed to use message-based semantics. These message-based semantics help SRD to handle out-of-order packets at the application level if needed. That would be infeasible with a byte streaming protocol as used by TCP/IP because message boundaries are opaque to the application. RDMA's unique feature is the native one-sided support. It is important to note that there are other combinations that we have not discussed or shown in the table. For instance, RDMA over Converged Ethernet (RoCE) uses Ethernet as its fabric instead of InfiniBand. However, those variations are outside of the scope of this paper.

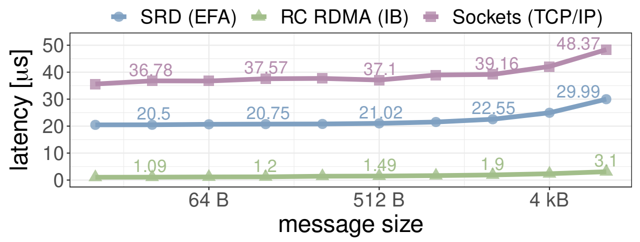

Figure 2: Impact of message size on the average latency (half round-trip-time) of the network fabrics SRD (EFA), RDMA (IB), and Sockets (TCP/IP).

3 EVALUATION

In the following, we discuss the results of our extensive evaluation which compares the performance of EFA with RDMA over InfiniBand in a set of reproducible microbenchmarks.

Methodology. To isolate the fundamental properties of EFA and RDMA over InfiniBand we used the well-known performance microbenchmark library *perftest* (available at [14]). Perftest uses *ibverbs* directly and supports EFA (SRD and UD) as well as reliable connected RDMA over InfiniBand. This makes the experiments directly comparable as both implementations use the same benchmarking code. Since perftest only supports single-threaded experiments, we additionally implemented a multi-threaded benchmark with *libfabric*.

Setup. Because EFA is only available on AWS which does not offer RDMA over InfiniBand we used two different hardware platforms – both running Linux. We conducted the EFA experiments using two EC2 *c5n.18xlarge* instances with 192 GB main memory and 72 vCPUs connected via 100 Gigabit EFA. Both *c5n.18xlarge* instances were deployed in the recommended *cluster* placement group to achieve the best low-latency network performance [40]. We also replicated our experiments on *c5n.metal* and obtained similar results. The RDMA over InfiniBand experiments were conducted on two bare-metal machines with 1 TB main memory and 56 CPUs connected with an InfiniBand network using Mellanox ConnectX-5 MT27800 NICs (InfiniBand EDR 4x, 100 Gbps).

3.1 Latency Comparison

As latency is critical for many applications, we start our evaluation by comparing the latency of SRD (EFA) with RC RDMA (IB). To classify the latency improvement of SRD (EFA) over EC2's traditional 100 Gigabit networking solution we included sockets (TCP/IP) as a reference. As mentioned, we use perftest [14] for EFA as well as RDMA and sockperf [27] for sockets.

Figure 2 compares the effect of varying message sizes on the average latency, i.e., half-round-trip latency. Both EFA and RDMA provide lower latencies than sockets. However, when comparing RDMA with SRD, we can observe that RDMA's latency is around 20× lower for messages below 512 bytes. For 8 kB messages RDMA's latency is still 10× lower than SRD's latency. Thus, although the latency results for EFA are considerably better than for the TCP/IP sockets, there remains a substantial gap to RDMA (similar latencies for RDMA are achievable on Azure VMs [28]).

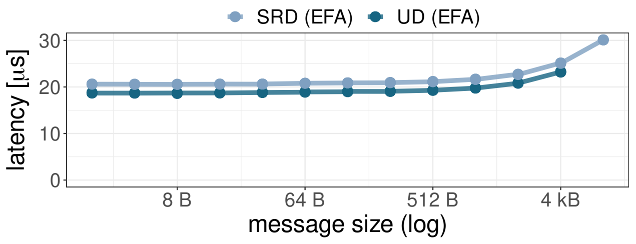

Figure 3: Average latency comparison between EFA's transport modes: SRD (EFA) and UD (EFA).

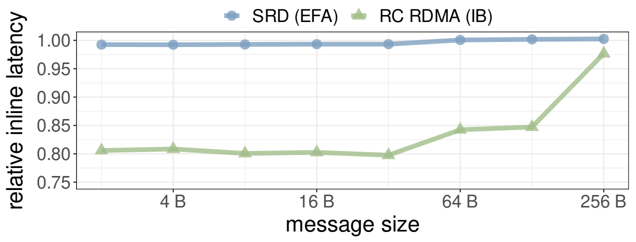

Figure 4: Relative impact of the inline optimization for small messages on latency. Inline optimization is applicable for messages <=32 B in SRD and messages <=220 B in RDMA.

3.2 EFA: SRD vs. UD

As the latency of SRD (EFA) is one order of magnitude higher compared to RDMA, we next evaluate if Unreliable Datagram (UD) (EFA) offers lower latencies. Figure 3 compares the effect of varying message sizes on latency on both connection types. Important to note is that SRD has an MTU of 8 kB while UD supports only 4 kB. From Figure 3 we can see that SRD (EFA) consistently has a slightly higher latency of around 1 μs. These are minor differences – especially when considering that SRD provides reliability in contrast to UD. One argument made in the past to favor UD RDMA over RC RDMA was its superior scalability [18, 19]. However, this does not translate to EFA SRD as described in Section 2.2. Therefore, we think the trade-off to sacrifice reliability to get a slight latency improvement is often not worthwhile for EFA. Since many applications require reliable delivery, and if not provided from the hardware as in SRD, it needs to be handled by the software stack, which increases code complexity and may equalize the latency improvement.

3.3 Inline Optimization

To improve the latencies of small messages a well-known optimization in RDMA is to *inline* the payload in the message request. Without inlining, the NIC performs an additional step to fetch the payload of a message request using DMA over PCIe. When using inlining, the CPU directly copies the payload inside the message request and thus the NIC can avoid the additional step to fetch the payload. Consequently, the message can be transmitted faster, and thus latency decreases, as shown by Kalia et al. [18]. Essentially, inlining shifts work from the network card back to the CPU. Because inlining can be applied to SRD for messages up to 32 bytes, we investigate if this optimization is as beneficial as for RDMA (which supports inlining for messages up to 220 bytes). Figure 4 shows the relative latency improvement of inlining compared to a no inlining baseline. Although the absolute latency improvement of inlining is around 500 nanoseconds for both (EFA and RDMA), in relative

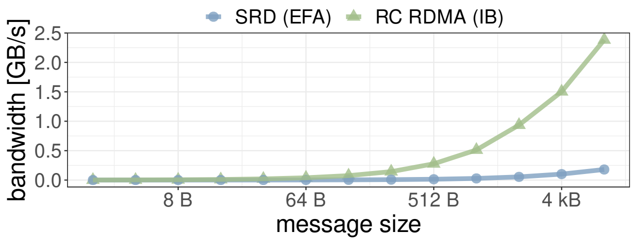

Figure 5: Impact of message size on the synchronous bandwidth, i.e, before sending the next message we wait for the completion of the previous message (single-threaded).

terms the effect is only substantial for RDMA due to EFA's higher base latencies.

3.4 Synchronous Bandwidth

We now turn our attention from latency to bandwidth. In Figure 5, we examine the synchronous bandwidth of SRD (EFA) and RC RDMA (IB) with varying message sizes using a single thread. In this context, *synchronous* means that we wait for the completion of the previous message before sending the next message. SRD and RC RDMA both adhere to a delivery-complete semantics which means that the completion is generated when the remote network card receives the message. For this setup, we can see in Figure 5 that RDMA achieves a much higher bandwidth. The reason is that RDMA's latency is much smaller and thus when cross-referencing the bandwidth results with the latency results from Figure 2 the achieved bandwidth is not surprising. Therefore, the achievable bandwidth of both (EFA and RDMA) is limited by their respective latencies. For instance, with 8 kB RDMA achieves roughly 2.5 GB/s which is 10 times more than SRD achieves in this setup (similar to the latency gap).

3.5 Asynchronous Bandwidth and Message Rate

As shown above, latency becomes the limiting factor in synchronous networking. Subsequently, we investigate how asynchronous networking avoids this limitation. Therefore, we vary the number of outstanding messages (i.e, the transmission depth) and show in Figure 6 how the bandwidth and message rate are affected.

First, let us focus on larger messages. With 4 kB messages and a transmission depth of about 64, RDMA achieves the maximum bandwidth (i.e., around 12GB/s). In contrast, EFA does not fully saturate the bandwidth and instead seems to be message bound at around 2 M messages, i.e., the same message rate as with smaller message sizes. When messages are sufficiently large, i.e., 8 kB and larger, both EFA and RDMA achieve the maximum bandwidth. However, Figure 6 shows that only RDMA is able to reach the full bandwidth when the transmission depth is small. The reason for that lies in RDMA's lower latency since it enables RDMA to process outstanding messages more quickly. For instance, a transmission depth of 8 means that we always try to have 8 messages outstanding. As RDMA's latency is lower, the completion for these 8 outstanding messages is generated faster and new messages can be transmitted. Conversely, because EFA's latency is higher the completion takes longer and thus requires a higher transmission depth to achieve the maximum bandwidth.

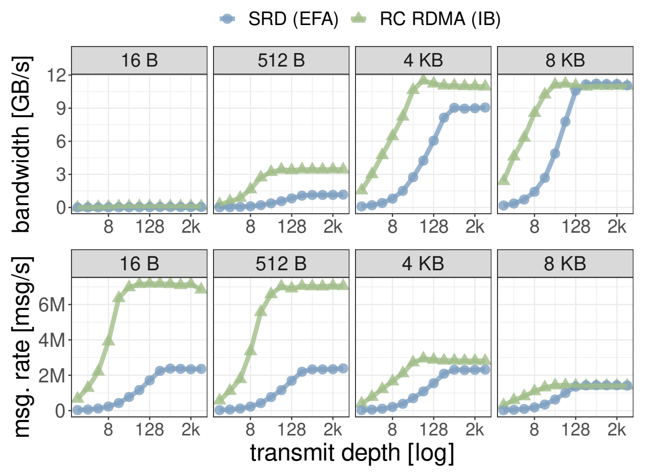

Figure 6: Effect of transmission depth (number outstanding messages) on asynchronous bandwidth (upper) and message rate (lower) for selected message sizes (single-threaded).

We now move on to smaller messages, i.e., 16 and 512 byte. When comparing the respective message rates for both 16 and 512 byte, we can observe that the message size does not affect the maximum message rate. RDMA achieves around 7 M messages per second and EFA around 2 M messages per second for both message sizes. One may now wonder what the limiting factor for EFA's message rate is. What we can already rule out is being latency-bound because we send multiple messages asynchronously. Additionally, we are not bandwidth bound either for 16 and 512 byte messages as we can clearly observe from Figure 6. Hence, we argue that the message rate is limited by the network card, as discussed in the next experiment.

3.6 NIC Parallelism

In this experiment, we examine if the processing unit (PU) of the network card became the limiting factor in the previous experiment. This can happen if a powerful CPU core overwhelms a less powerful PU on the NIC. Often a single connection is handled by a single NIC PU [18]. Consequently, increasing the number of connections may lead to the utilization of multiple NIC PUs, which ultimately improves the message rates.

To evaluate if NIC parallelism can be exploited we use a single thread and increase the number of connections. Figure 7 shows that a single CPU core utilizes multiple connections with SRD and thus achieves a peak message rate of around 4 M with 4 connections (16 and 512 byte messages). With 4 kB message size, EFA eventually reaches the full bandwidth with 2 connections (same message rate as RDMA which achieves full bandwidth). In contrast, RDMA does not profit from having multiple connections, instead, performance degrades slightly for smaller messages. We can conclude that RDMA is CPU-bound with small messages. We confirmed this statement by using a single connection and posting a linked list of multiple messages to the NIC, i.e., in perftest this is done by setting the *post list* parameter to 16. This allows the NIC to process the requests at full speed without being dependent on the CPU. For RDMA the message rate with 64 byte messages peaked at around 17 M messages per second whereas EFA's messages per second remained at 2 M due to the processing limit of a single NIC PU.

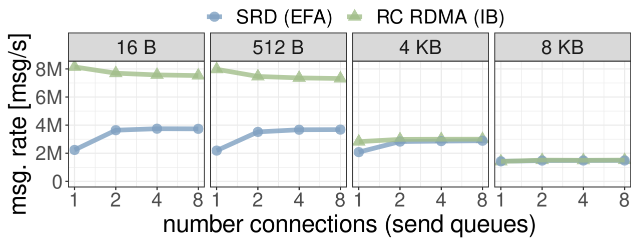

Figure 7: Effect of increasing number of connections (send queues) on message rate to exploit NIC parallelism (single-threaded, transmission depth is 256).

Based on these results, we can reasonably speculate that a single Nitro Card PU can achieve around 2 M messages per second. To achieve the maximum of 4 M messages per second multiple Nitro Card PUs need to be exploited. However, it is not clear if a single thread can saturate all available Nitro Card PUs thus in the next experiment we will investigate if multiple threads achieve a higher peak message rate.

3.7 Multi-Threading

After the single-threaded experiments, we move on to showing the multi-threaded performance. Multi-threading helps us to resolve the previously discovered limitations in a single-threaded setup.

For this experiment, we do not use the perftest benchmark suite since it does not support multi-threading. Therefore, we implemented our own benchmark with *libfabric* which is also the reason that the single-threaded numbers in Figure 8 do not match the previous experiments. *Libfabric* has substantial CPU overhead as we will show in the next experiment. However, this overhead is compensated for in this experiment by using multiple CPU resources.

Figure 8 shows how increasing the number of CPU threads affects both bandwidth (upper) and message rate (lower). For 4 kB message size, EFA and RDMA reach the full bandwidth. Whereas RDMA reaches the bandwidth limit already with 2 threads, EFA requires 4 threads. The message rate with 64 byte messages for EFA is capped at around 8 M. When considering the results from Figure 7 where we deduced that a single NIC PU reaches 2 M messages per second we can speculate that the NIC may have 4 PUs in total. On the other hand, the RDMA network card achieves more than 30 M messages per second.

3.8 EFA Interface Evaluation

We now compare the performance of *libfabric* with the lower-level *ibverbs* interface. We examine the bandwidth as well as the message rate in Figure 9. To verify that our *libfabric* implementation performs as expected, we compared it to the OSU MPI performance test, which uses *libfabric* as well. Figure 9 shows that using the lower-level library *ibverbs* yields the best performance, i.e., around 50% more messages per second. Conversely, one may argue that when using multiple threads, we may hit the limits of the EFA NIC anyways and *libfabric*'s better usability may be worth it. However, we found that the library had many performance knobs and thus was not easier to use than *ibverbs* at all.

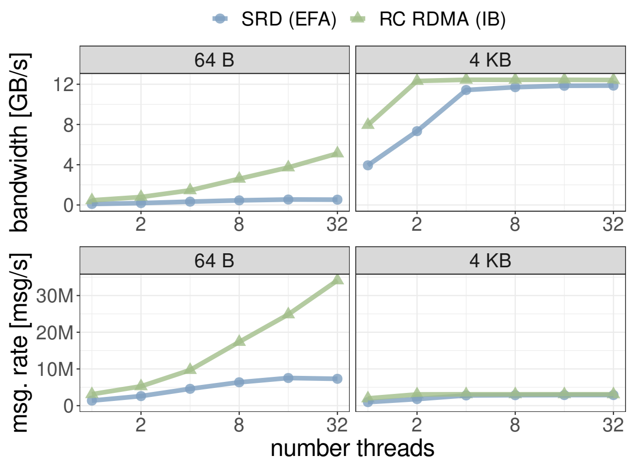

Figure 8: Impact of increasing number of threads on bandwidth (upper) and message rate (lower) for 64 B and 4 kB messages (transmission depth is 256).

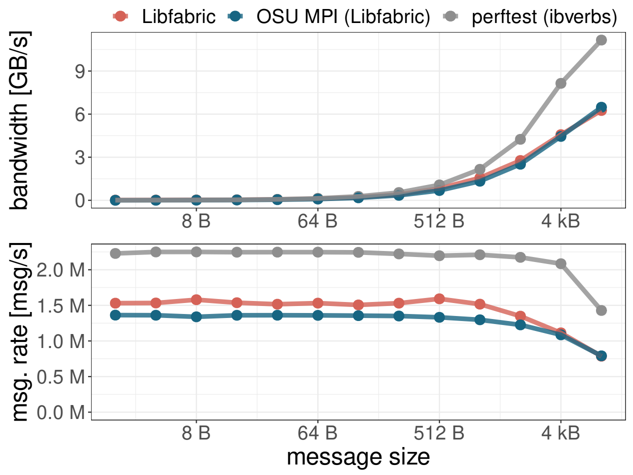

Figure 9: Performance implication of the selected EFA interface (*ibverbs* or *libfabric*) on bandwidth (upper) and message rate (lower) for message size up to 8 kB (single-threaded, transmission depth is 256).

4 RELATED WORK

We think that the broad availability of EFA has the potential to trigger a redesign of distributed databases. Unfortunately, there has only been very limited work on EFA in general.

Most research was driven by the HPC community and specifically geared towards MPI as it is the de-facto standard in distributed HPC applications [4, 41, 47]. Most notably is the paper from Shalev et al. (Amazon) [41] which discusses some of the design decisions behind the proprietary SRD protocol. The other two papers [4, 41, 47] specifically focus on MPI primitives and on how different MPI implementations perform with EFA.

There are some papers in the machine learning community which use EFA for distributed training [43, 52]. However, they focus on the machine learning part and use EFA transparently. The only database paper which mentions EFA is from Barthels et al. [2] in which they show in one experiment that their findings are transferable to the off-the-shelf network infrastructure of AWS.

5 LESSONS LEARNED AND SUMMARY

Although both EFA and RDMA over InfiniBand are advertised for a similar audience, their performance characteristics are quite different. This has major implications on how distributed data processing systems for the cloud should be designed and optimized. In the following, we summarize our main findings and their implications on system design:

Latency. As we have seen in Figure 2, EFA's latency decreased twofold compared to traditional TCP/IP sockets. Nonetheless, the latencies of EFA are still an order of magnitude higher than those of RDMA. ⇒ Due to EFA's higher latency it is not as beneficial to send small messages as in RDMA. Therefore, RDMA techniques which aim at decreasing the message size, such as message shrinking [18] and inline optimization, are not as effective in EFA. Rather, to amortize the high latency techniques such as batching or piggybacking should be considered for system design.

Bandwidth. As shown in Figure 6, the bandwidth of EFA is strongly dependent on the transmission depth and the message size. ⇒ To saturate the bandwidth, a large transmission depth (e.g., 256) and message sizes are important. When the message size is below 8 kB, NIC-parallelism should be exploited by either using multiple connections for a single-threaded application or multiple threads. With even larger messages (e.g., 8 kB) those sophisticated optimizations are not necessary to achieve the full bandwidth.

Message Rate. Figure 6 shows that the achievable message rate for small messages in EFA is considerably smaller than in RDMA. ⇒ NIC parallelism is crucial to utilize the maximum message rate, preferable with multiple threads to achieve 8 M messages/second.

No one-sided operations. Besides performance characteristics, other factors such as the functionality of the interface play an important role. For example, one-sided operations, the key primitives for many state-of-the-art distributed RDMA systems, are not natively available in EFA. ⇒ System designs based on one-sided primitives might need to be revisited.

Out-of-order-delivery. In contrast to reliable connected RDMA, EFA does not guarantee in-order delivery. ⇒ Systems which rely on ordering need to handle re-ordering in the software stack which may induce some overhead.

Ibverbs vs. libfabric. The available EFA interfaces offer different characteristics. ⇒ We argue that for database systems the low-level *ibverbs* is likely better suited. *Ibverbs* yields better performance and some higher-level functions of *libfabric* may impact the database system. For instance, the emulation of one-sided verbs might interfere with the thread scheduler of the database system.

EFA is proprietary. Unlike InfiniBand, it is not yet possible to equip on-premise machines with EFA. ⇒ Therefore, EFA can exclusively be used in AWS ec2 instances (vendor lock-in).

Conclusion. Overall, even though EFA does not yet come close to the latency of RDMA , we believe that it comes with potential for distributed database systems. This is because EFA is already widely available in AWS today [39] which comes with a wide offering of available compute instances. Compared to TCP/IP, which was the only available networking option in AWS before, EFA considerably reduces the latency. Finally, we believe that some of the current limitations of EFA, such as its low message rate, may be resolved by future Nitro Cards generations.

REFERENCES

[1] Gustavo Alonso, Carsten Binnig, Ippokratis Pandis, Kenneth Salem, Jan Skrzypczak, Ryan Stutsman, Lasse Thostrup, Tianzheng Wang, Zeke Wang, and Tobias Ziegler. 2019. DPI: The Data Processing Interface for Modern Networks. In *CIDR*.

[2] Claude Barthels, Ingo Müller, Konstantin Taranov, Gustavo Alonso, and Torsten Hoefler. 2019. Strong consistency is not hard to get: Two-Phase Locking and Two-Phase Commit on Thousands of Cores. *PVLDB* 12, 13 (2019).

[3] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. 2016. The End of Slow Networks: It's Time for a Redesign. *PVLDB* 9, 7 (2016).

[4] Sourav Chakraborty, Shulei Xu, Hari Subramoni, and Dhabaleswar K. Panda. 2019. Designing Scalable and High-Performance MPI Libraries on Amazon Elastic Fabric Adapter. In *2019 IEEE Symposium on High-Performance Interconnects, HOTI 2019,*. IEEE.

[5] Aleksandar Dragojevic, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *NSDI*.

[6] Aleksandar Dragojevic, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. 2015. No compromises: distributed transactions with consistency, availability, and performance. In *SOSP*.

[7] Philipp Fent, Alexander van Renen, Andreas Kipf, Viktor Leis, Thomas Neumann, and Alfons Kemper. 2020. Low-Latency Communication for Fast DBMS Using RDMA and Shared Memory. In *ICDE*.

[8] Philip Werner Frey and Gustavo Alonso. 2009. Minimizing the Hidden Cost of RDMA. In *ICDCS*.

[9] OpenFabrics Interfaces Working Group. 2021. EFA RDM Communication Protocol version 4. https://github.com/ofiwg/libfabric/blob/main/prov/efa/docs/efa_rdm_protocol_v4.md

[10] OpenFabrics Interfaces Working Group. 2022. Libfabric. https://ofiwg.github.io/libfabric/

[11] OpenFabrics Interfaces Working Group. 2022. Libfabric Programmer Manual: fi_domain. https://ofiwg.github.io/libfabric/v1.1.1/man/fi_domain.3.html

[12] OpenFabrics Interfaces Working Group. 2022. Libfabric Programmer Manual: fi_efa. https://ofiwg.github.io/libfabric/v1.14.0/man/fi_efa.7.html

[13] OpenFabrics Interfaces Working Group. 2022. Libfabric Provider. https://ofiwg.github.io/libfabric/master/man/fi_provider.7.html

[14] OpenFabrics Interfaces Working Group. 2022. Perftest. https://github.com/linux-rdma/perftest

[15] Eunyoung Jeong, Shinae Woo, Muhammad Asim Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems. In *NSDI*.

[16] Chengfan Jia, Junnan Liu, Xu Jin, Han Lin, Hong An, Wenting Han, Zheng Wu, and Mengxian Chi. 2018. Improving the Performance of Distributed TensorFlow with RDMA. *Int. J. Parallel Program.* 46, 4 (2018).

[17] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2014. Using RDMA efficiently for key-value services. In *SIGCOMM*.

[18] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. Design Guidelines for High Performance RDMA Systems. *login Usenix Mag.* 41, 3 (2016).

[19] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *OSDI*.

[20] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2019. Datacenter RPCs can be General and Fast. In *NSDI*.

[21] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. 2014. MICA: A Holistic Approach to Fast In-Memory Key-Value Storage. In *NSDI*.

[22] Feilong Liu, Claude Barthels, Spyros Blanas, Hideaki Kimura, and Garret Swart. 2020. Beyond MPI: New Communication Interfaces for Database Systems and Data-Intensive Applications. *SIGMOD Rec.* (2020).

[23] Feilong Liu, Lingyan Yin, and Spyros Blanas. 2017. Design and Evaluation of an RDMA-aware Data Shuffling Operator for Parallel Database Systems. In *EuroSys*.

[24] Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Pete Wyckoff, and Dhabaleswar K. Panda. 2003. High performance RDMA-based MPI implementation over InfiniBand. In *ICS*.

[25] Simon Loesing, Markus Pilman, Thomas Etter, and Donald Kossmann. 2015. On the Design and Scalability of Distributed Shared-Data Databases. In *SIGMOD*.

[26] Xiaoyi Lu, Dipti Shankar, Shashank Gugnani, and Dhabaleswar K. Panda. 2016. High-performance design of apache spark with RDMA and its benefits on various workloads. In *Big Data*.

[27] Mellanox. 2022. Sockperf. https://github.com/Mellanox/sockperf

[28] Microsoft. 2021. HC-series virtual machine sizes. https://docs.microsoft.com/en-us/azure/virtual-machines/workloads/hpc/hc-series-performance

[29] Microsoft. 2022. High performance computing VM sizes. https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-hpc

[30] Christopher Mitchell, Yifeng Geng, and Jinyang Li. 2013. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In *USENIX ATC*.

[31] Christopher Mitchell, Kate Montgomery, Lamont Nelson, Siddhartha Sen, and Jinyang Li. 2016. Balancing CPU and Network in the Cell Distributed B-Tree Store. In *USENIX ATC*.

[32] Xiangyong Ouyang, Sonya Marcarelli, Raghunath Rajachandrasekar, and Dhabaleswar K. Panda. 2010. RDMA-Based Job Migration Framework for MPI over InfiniBand. In *CLUSTER*.

[33] DPDK Project. 2022. DPDK. https://www.dpdk.org/

[34] Mohammad J. Rashti and Ahmad Afsahi. 2008. Improving Communication Progress and Overlap in MPI Rendezvous Protocol over RDMA-enabled Interconnects. In *HPCS*.

[35] Wolf Rödiger, Tobias Mühlbauer, Alfons Kemper, and Thomas Neumann. 2015. High-Speed Query Processing over High-Speed Networks. *PVLDB* 9, 4 (2015).

[36] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *CoRR* (2018).

[37] Amazon Web Services. 2019. Elastic Fabric Adapter is officially integrated into Libfabric Library. https://aws.amazon.com/about-aws/whats-new/2019/07/elastic-fabric-adapter-officially-integrated-into-libfabric-library/

[38] Amazon Web Services. 2021. AWS Nitro System. https://aws.amazon.com/ec2/nitro/

[39] Amazon Web Services. 2021. EFA is now mainstream, and that's a Good Thing. https://aws.amazon.com/blogs/hpc/efa-is-now-mainstream/

[40] Amazon Web Services. 2022. Placement Groups. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html

[41] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. 2020. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro* 40, 6 (2020).

[42] Alex Shamis, Matthew Renzelmann, Stanko Novakovic, Georgios Chatzopoulos, Aleksandar Dragojevic, Dushyanth Narayanan, and Miguel Castro. 2019. Fast General Distributed Transactions with Opacity. In *SIGMOD*.

[43] Indu Thangakrishnan, Derya Cavdar, Can Karakus, Piyush Ghai, Yauheni Selivonchyk, and Cory Pruce. 2020. Herring: rethinking the parameter server at scale for the cloud. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*.

[44] Lasse Thostrup, Jan Skrzypczak, Matthias Jasny, Tobias Ziegler, and Carsten Binnig. 2021. DFI: The Data Flow Interface for High-Speed Networks. In *SIGMOD*.

[45] Tianzheng Wang, Ryan Johnson, and Ippokratis Pandis. 2017. Query Fresh: Log Shipping on Steroids. *PVLDB* 11 (2017).

[46] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. 2015. Fast in-memory transaction processing using RDMA and HTM. In *SOSP*.

[47] Shulei Xu, Seyedeh Mahdieh Ghazimirsaeed, Jahanzeb Maqbool Hashmi, Hari Subramoni, and Dhabaleswar K. Panda. 2020. MPI Meets Cloud: Case Study with Amazon EC2 and Microsoft Azure. In *Fourth IEEE/ACM Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware, IPDRM@SC 2020*. IEEE.

[48] Jilong Xue, Youshan Miao, Cheng Chen, Ming Wu, Lintao Zhang, and Lidong Zhou. 2019. Fast Distributed Deep Learning over RDMA. In *EuroSys*.

[49] Erfan Zamanian, Carsten Binnig, Tim Kraska, and Tim Harris. 2016. The End of a Myth: Distributed Transactions Can Scale. *CoRR* abs/1607.00655 (2016).

[50] Erfan Zamanian, Julian Shun, Carsten Binnig, and Tim Kraska. 2021. Chiller: Contention-centric Transaction Execution and Data Partitioning for Modern Networks. *SIGMOD Rec.* 50, 1 (2021).

[51] Steffen Zeuch, Sebastian Breß, Tilmann Rabl, Bonaventura Del Monte, Jeyhun Karimov, Clemens Lutz, Manuel Renz, Jonas Traub, and Volker Markl. 2019. Analyzing Efficient Stream Processing on Modern Hardware. *PVLDB* 12 (2019).

[52] Shuai Zheng, Haibin Lin, Sheng Zha, and Mu Li. 2020. Accelerated Large Batch Optimization of BERBT Pretraining in 54 minutes. *CoRR* abs/2006.13484 (2020). https://arxiv.org/abs/2006.13484

[53] Tobias Ziegler, Sumukha Tumkur Vani, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. 2019. Designing Distributed Tree-based Index Structures for Fast RDMA-capable Networks. In *SIGMOD*.