Design and Evaluation of a Scalable RDMA Network Platform

Authors
RDMAvisor 12 pages

ABSTRACT

Due to the low CPU overhead, low latency and high throughput, RDMA technique has attracted the attention of cloud service providers and was increasingly integrated into the main memory computing systems. Though the performance gain has been observed by using the advanced networking technique, e.g., RDMA over Ethernet (RoCE), to overcome the traditional TCP/IP communication bottleneck, it is still a challenging problem for developers to fully utilize the performance of RDMA hardware. Because the application performance is highly related to the low-level details of RDMA operations and requires high scalability for the current large scale distributed computing systems with thousands of connections. To address this problem, we present a simple and scalable RDMA data transfer platform to mitigate the impact of RDMA operational details. The system provides careful connection management through adaptive RDMA performance tuning to improve the utilization of CPU, memory and network bandwidth. These optimized designs lead to simple and flexible programming model for both common users and knowledgeable developers to make the best usage of RDMA networks and support rapid product development. We have implemented a prototype, named `RDMAvisor`, and evaluated its performance on a cluster with a large number of connections. Our experiment results demonstrate that `RDMAvisor` achieves high throughput for thousand of connections and maintains low CPU and memory overhead through adaptive RDMA transport selection.

CCS CONCEPTS

• **Networks** → *Programming interfaces*; • **Software and its engineering** → *Message passing*;

KEYWORDS

RDMA, Cloud Computing, Interconnect Technology

ACM Reference format:
Authors. 2017. Design and Evaluation of a Scalable RDMA Network Platform. In *Proceedings of CoNEXT, Korea, Nov. 2017 (CoNext'17),* 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Remote Direct Memory Access (RDMA) technique provides the messaging service that directly access the virtual memory on remote machines. Since data can be copied by the RDMA network interface cards (RNICs), RDMA bypasses the complex protocol stack and minimizes the operating system involvement. This design achieves low data transportation latency, high throughput, and removes the overhead of remove CPU. RDMA has been widely used by the high performance computing (HPC) community which is supported by the InfiniBand (IB) network.

CoNext'17, Korea
2017. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

Recently, due to the decreasing price of RDMA hardware and the compatible design of RDMA over Ethernet (RoCE) standard, cloud platforms have been substantially deploying RDMA in order to alleviate communication bottleneck for distributed services [10, 25, 26]. To make RDMA network scalable to support hundreds of thousands of nodes in modern datacenter, the IP routable RoCE (RoCEv2) [6] protocol was defined and quickly deployed in the Microsoft datacenter [12, 26]. Furthermore, by leveraging the advanced techniques of priority-based flow control (PFC) [5] and congestion notification (QCN) [26] it has indicated the potential RDMA by replacing TCP for intra-datacenter communications [12].

RDMA defines asynchronous network programming interfaces, RDMA verbs, for submitting work requests to the channel adapter and returning completion status. QPs. Three transport types are provided: Reliable Connection (RC), Unreliable Connection (UC), and Unreliable Datagram (UD). Reliable transport guarantees lossless transfer and ensures in-order delivery of messages by using acknowledge. The unreliable transport providing no sequence guarantee consumes less bandwidth and delay due to no ACK/NACK packet. In the RDMA verbs, both channel semantics and memory semantics are provided to users. The channel semantics are two-sided operations, sometimes called *Send/Receive*, which is the communication style used in a classic I/O channel. The Memory verbs are one-sided operations, embedded in the *Read and Write* operations, which allows the initiator of the transfer to specify the source or destination location without the involvement of the other endpoint CPUs.

Despite rich transport modes and operations have been provided, it is still a challenging task to achieve advance capabilities for distributed applications deploying in the RDMA-capable datacenters. With regard to the specific demands of different applications and the shared environment of datacenter, applying the naive RDMA may not be a wise move, which can not fully utilize the advanced hardware performance [7, 16**?** , 17]. The best-fit RDMA-aware system design for a specific application requires careful parameters turning which highly relies on the low-level knowledge associated with the RNIC architecture, operation details, etc. [8, 10, 16, 17, 23].

Scalability is a practical concern for modern datacenters where a large number of connections are generated in one machine and also requires for incremental expansion. However, due to the limited cache space, currently ConnectX-3 RNIC can only supports around 400 Queue Pairs (QPs). Thus, the connected transport which provides exclusive access to QPs is not scalable. QPs sharing is a common solution to this problem [3]. For small messages, key-value store [15] and RPC [16] leveraged the two-side verbs, which allow each CPU core to create one datagram QP that can communicate with all remote cores. However, Send/Recv operations require the involvement of CPUs at both the sender and receiver. For large items, QP sharing for one-side verbs was implemented in [10]. But it can reduce CPU efficiency because threads contend for locks [16].

To address this challenge, a more efficient lock-free design is provided in our design for one-side verbs to achieve high scalability but low CPU overhead. The shared receive queue (SRQ) model partially solve the problem of one-side Recv operation which posts receive WRs to a queue that is shared by a set of connections. However, under the naive RDMA consumer queuing model, each application maintains and manages its own RDMA queues. Our key observation is that the resource such as SRQs can be shared among multiple applications that are running on the same physical machine. To this end, we provide careful memory management among transport connections of consumers that achieves an high utilization of memory usage, low CPUs overhead for the low latency, high throughput RDMA network service.

Currently, the available design of RDMA network functionality is tightly coupled with the targeting system. The basic units of RDMA network functionality, e.g., RDMA enabled buffers management, data structures and polling thread, usually work as modules embedded in the system. The coupled design makes it hard to reuse the code in other applications. In addition, since the low-level details are important for RDMA system design [17], the application performance is highly determined by the choice of RDMA operations. Inappropriate configurations may cause even worse performance than the TCP/IP stack. Unfortunately there is no one-size-fits-all approach. The optimal setting of a given system can be the poison to other ones. It is not easy to train the system designers to master low-level details of RDMA technologies and understand the guidelines [9, 17] for various conditions in a short time, which hind the fast development of RDMA-based system in datacenters. Thus, it is necessary to ensure that the application is not sensitive to the RDMA network operations by introducing a shim layer to mitigate the influence of low-level details. To address this challenge, the flexible RDMA functions are abstracted by Socket-like network interfaces. All the conditional sections in the program are described with parameters. For common users, the low-level details are not exposed and the performance is guaranteed by adaptive RDMA primitive selection in the shim layer. For specific requirements their demands can be realized through customized setting through macro arguments.

The contribution of this paper is two fold. First, we present a simple and scalable RDMA network platform for fast deployment of RDMA-capable services to serve all applications running on a physical machine. The interface semantics are fairly straightforward and friendly to developers. The user only needs to decide to use connection or datagram transport service, because the platform has mitigated the impact of low-level details by improving the scalability of one-sided operations, the resource utilization of two-sided operations and providing adaptive RDMA transport based on the usage of CPUs and memory at both end-hosts. Second, we have implemented a prototype `RDMAvisor`, and evaluated the performance in a cluster running a large number of connections. `RDMAvisor` shows more efficient usage of memory and CPUs, and achieves high throughput for thousand of connections. the macro-benchmark result on real systems.

	SEND/RECV	WRITE	READ	Max Message
RC	✓	✓	✓	1GB
UC	✓	✓	✗	1GB
UD	✓	✗	✗	MTU

Table 1: Operations and maximum message size supported by each transport type. RC and UC support message size up to 1GB. The message is divided into MTU-sized frames during transmission on Ethernet. UD supports the maximum size of a message up to MTU size.

2 BACKGROUND

RDMA transfers data in and out of pre-registered buffers at end hosts. The Data transfer requests are delivered to RNIC over send and receive queues (queue pairs, QPs). RNIC provides data transportation according to the work requests in QPs and thus releases the precious CPUs resource for the computation services. The hardware operation simplifies the data copy without the involvement of kernel and thus significantly reduces the overall latency. RDMA technique has been supported by InfiniBand (IB) which are widely used for the high performance computing (HPC). The IB protocol assumes a lossless networking fabric and uses credit-based flow control to prevent packet drops due to buffer overflow, which simplifies the design and implementation of protocol stack.

To compatible with modern datacenter the RDMA over Converged Ethernet (RoCE) [13] standard has been defined. RoCE replaces IB Layer 2 with Ethernet. To avoid buffer overflow, Priority-based Flow Control (PFC) [5] ensures reliable data transfer by forcing the immediate upstream entity to pause data transmission. Scalability is an important issue for modern datacenter. To support thousands of nodes there, the IP routable RoCE (RoCEv2) [6] protocol was later defined through layer 3 extension, i.e., RoCEv2 protocol replaces IB networking layer with IP and UDP encapsulation. The IP header allows routing in traditional network and UDP header enables load balancing through ECMP [12].

2.1 RDMA Verbs

RDMA defines the asynchronous network programming interface, RDMA verbs. Verbs describe operations that take place between a host channel adapter and its operating system based on a particular queuing model for submitting work requests to the channel adapter and returning completion status. Verbs describe the parameters necessary for configuring and managing the channel adapter, allocating (creating and destroying) queue pairs, configuring QP operation, posting work requests to the QP, getting completion status from the completion queue. Table 1 shows the operations available in each transport mode.

RDMA Operations: In the RDMA verbs, both channel semantics and memory semantics are provided to the user. The *channel semantics* are two-sided operations, Send/Recv. RDMA defines Send operation which is posted at the source to initiate the transfer, and the Recv operation which is posted at the destination to specify a target buffer for the transfer. The *Memory sematics* are one-sided operations. Memory semantic embedded in the Read and Write operations, which allows the initiator of the transfer to specify the source or destination location without the involvement of the other

endpoint. The initiating party directly reads or writes the virtual address space of a remote node, does not involve the remote CPU.

RDMA transports: Reliable transport uses acknowledge to guarantee in-order delivery of messages. Unreliable transport providing no sequence guarantee. Connected transport requires one-to-one connections between QPs. Unconnected transport (datagram) allows each CPU core to create one datagram QP that can communicate with all remote cores. The Unreliable Datagram service greatly improves the scalability. Since the service is connectionless, an end node with a fixed number of QPs can communicate with multiple consumers compared to the Reliable Connection and Unreliable Connection service. Connected transport provides exclusive access to QPs, which is not scalable due to limited RNIC cache space. Threads can share QPs to reduce the QP memory footprint [10]. Sharing QPs reduces CPU efficiency because threads contend for locks and the cache lines for QP buffers bounce between their CPU cores.

Traditional applications have the requirement of migrating the services to the high-speed RDMA network environment. However, it requires careful performance tuning when using the native verbs interface. An easy-to-use software tool is necessary to support effective data transfer over current RDMA network.

3 DESIGN

This section introduces the design of `RDMAvisor`, which includes the characteristic of communication primitives, programing models and the optimization for the connection management.

3.1 Communication Primitives

The foundation of RDMA operation is the ability of a consumer to queue up a set of instructions that hardware executes. This facility is referred to as a Work Queue (WQ). Work queues are always created in pairs, called a Queue Pair (QP), one for send operations and one for receive operations. The consumer submits a work request (WR), which causes an instruction called Work Queue Element (WQE) to be placed in the work queue. RDMA NIC executes WQEs in the order without involving the kernel. When finishing a request, a Completion Queue Element (CQE) is placed in the completion queue (CQ).

Unreliable Datagram (UD) has good scalability, since the connectionless service allows use one QP to communicate with multiple other QPs. UD operates with less state maintained at each end-host is suitable for latency-sensitive applications with small messages, such as key-value store [15] or RPC service [16]. UD Send/Recv operations require involvement of CPU at both sender and receiver. Besides, since the maximum message length is constrained to fit in a single packet (MTU), UD messages are conveyed by multiple packets, which is not suitable for throughput-sensitive applications.

One-side verbs (Read and Write) bypass the remote CPU to operate directly on the remote memory. This advanced feature reduces the CPU overhead for communication. We compared the performance of RC Write, UC Write and RC Read, as shown in Figure 1. The throughput of RC Write is close to that of UC Write, while RC Read is comparable to RC Write for large message. It is notable, however, that UC QPs do not support Shared Receive Queue (SRQ) [2], which is important for building the scaling out

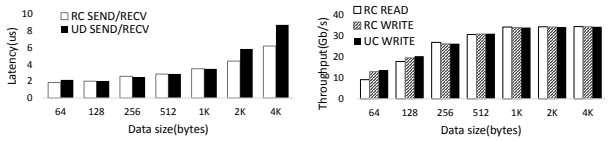

Figure 1: Comparision of RDMA operations

```
int connect(Target* t, int FLAGS);
int listen(Target* t, int FLAGS);
int accept(int fd, int FLAGS);
uint32_t send(int fd, void* buf, uint32_t len, int FLAGS);
uint32_t recv(int fd, void* buf, uint32_t len, int FLAGS);
uint32_t recv_zero_copy(int fd, void** buf_addr, uint32_t len, int FLAGS);
uint32_t sendto(int fd, void* buf, uint32_t len, Target* t, int FLAGS);
uint32_t recvfrom(int fd, void* buf, uint32_t len, Target* t, int FLAGS);
uint32_t recvfrom_zero_copy(int fd, void** buf_addr, uint32_t len, int FLAGS);
```

Figure 2: Network API

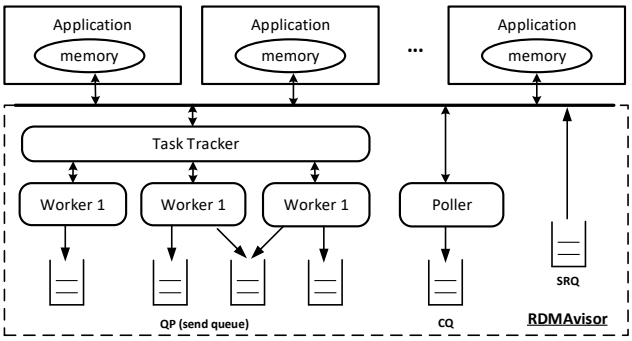

Figure 3: Overview of RDMAvisor

RDMA platform. Therefore in our design we use RC as the default RDMA transport instead of UC when using the connected transport. The selection of RC Read and Write is adaptively adjusted based on the current CPU and memory consumption of servers at both sides (see Sec. 3.2). To improve the scalability of one-side verbs, we introduce a lock-free approach for effective QPs sharing to support thousands of connections in one host.

3.2 Architecture and Programming Model

```
int connect(Target* t)
int listen(Target* t)
int accept(int fd)

uint32_t send(int fd, void* buf, unin32_t len, int FLAGS)
uint32_t recv(int fd, void* buf, unin32_t len, int FLAGS)
uint32_t recv_zero_copy(int fd, void** buf_addr, unin32_t len, int FLAGS)
uint32_t sendto(int fd, void* buf, unin32_t len, Target* t, int FLAGS)
uint32_t recvfrom(int fd, void* buf, unin32_t len, Target* t, int FLAGS)
uint32_t recvfrom_zero_copy(int fd, void** buf_addr, unin32_t len, int FLAGS

uint32_t sendmsg(int fd, struct msghdr* msg, int FLAGS);
uint32_t recvmsg(int fd, struct msghdr* msg, int FLAGS);
```

Figure 3 shows the components of RDMAvisor. Applications store data in main memory. RDMAvisor works as a daemon process to serve all applications through well designed user-level interfaces. In RDMAvisor, the working thread, named *Worker*, is responsible for translating application's data transfer request into appropriate WRs and then pushing WQEs to the corresponding QPs. Another working thread, called *Poller*, is responsible for polling CQ and delivering message to the corresponding application in asynchronous mode.

RDMAvisor supports simple interfaces to invoke RDMA functions by hiding the complicate details. Figure 2 shows the defined operations. They are socket-like operations which are friendly to users. The first three interfaces `connect, listen, accept` are used to set up connections. Here `Target` is an unified encapsulation of host address, including IPv4, IPv6 ,and it holds a socket fd to transfer necessary IB information for activating queue pairs and opt proper visual queuepair number for visual connection. The default value of FLAGS means RC operations in our design. Users can use `connect` to initiate active connections and `accept` to passively wait for new incoming connections. The field of `fd` is used to specify on which logical connection the interface needs to be invoked.

Normal users can simply use `send`, `recv` operations for connection-oriented data transport, and `sendto`, `recvfrom` operations for datagram transport[1]. RDMAvisor will adaptively select RDMA Send/Recv for data block of small size and RDMA Read/Write operations for large data according to the explicitly indicated data size in the parameter of 'len'. Specifically, to reduce latency, RDMAvisor chooses one-side verbs based on the current CPU consumption and work load, which is easily measured by the RDMAvisor daemon process. It is notable that a parameter called FLAGS is introduced in the RaaS's interface. FLAGS is used to specific RDMA transport for one user with any special requirement on RDMA operations, e.g., *RC|WRITE* or *Atomic*. This design provides the flexibility to help knowledgeable users to achieve customized setting. RDMAvisor also provides `sendmsg` and `recvmsg` operations for sending and receiving multi-buffer data. This two operations pack these buffers into one WR. Actually, users can use `send` and `recv` to send and receive multiple buffers one by one. However, this causes RDMAvisor using multiple WRs to finish transferring data, which is conflicted to `sendmsg` and `recvmsg` original meanings and leads to resources waste. Furthermore, multiple sends may causes logical errors in applications. `sendmsg` and `recvmsg` guarantee that users can send and recvive data in one msghdr struct and one WR.

We use `send` for outbound data transmission. Both `recv` and `recv_zero_copy` are used for receiving incoming data. `send` and `recv` work in the non-blocking model. The interface of `recv_zero_copy` is provided for application which works in the blocking mode. `recv_zero_copy` interface is popular in high performance system. Since in the blocking mode, it may take longer time to pick the data from the receive buffer of RDMA, `recv_zero_copy` will deliver the incoming data directly from receiving buffers to the pre-registered private memory of the application, instead of occupying the shared memory of RDMAvisor.

[1]Here we only explain the operations for connection-oriented transport due to the limited space. The operations for datagram transport are similar.

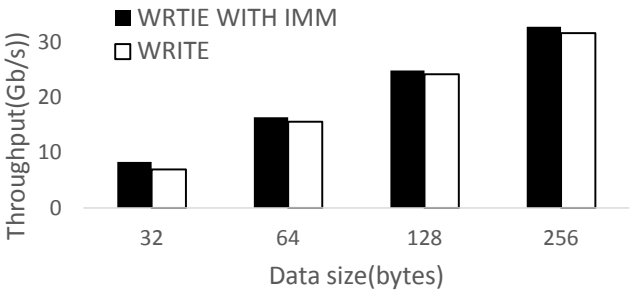

Figure 4: WRITE vs. WRITE_WITH_IMM

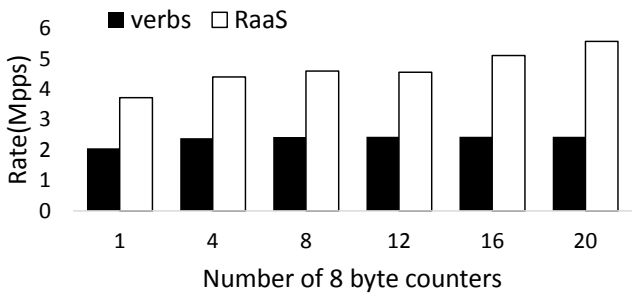

Figure 5: atomic operations

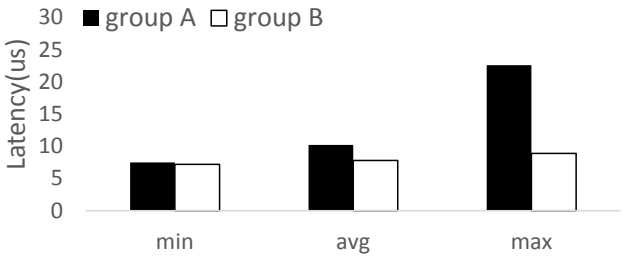

Figure 6: WRITE vs. WRITE_WITH_IMM

We do not provide `send_zero_copy` due to the revelation of related work [11]. The reason is as follows. For sending tasks, RNICs must get the physical address of outbound data to initiate a DMA operation. The memory region in which outbound data lying must be registered to RNICs beforehand. One way is to copy applications' outbound data to pre-registered memory region, called *memcpy*. Another approach is to register the memory region of applications' outbound data to RNICs every time, called *memreg*. However, in the case of relative small size of outbound data, *memcpy* achieves better latency, but *memreg* outperforms *memcpy* when the size of outbound data increases. Therefore, `send_zero_copy` may be inefficient in sending the relatively big size messages and RDMAvisor is going to handle this according to the previous work [11].

3.3 Adaptive Operations

introduce a function for adaptive operation, explain the relationship between num. of requests and the CPU/memory/traffic

3.4 Scalability and Multiplexing of Queues

need proof! We focus on solving the scalability problem of RDMA RC by introducing a more effective lock-free design of QPs sharing. The scaling problem incurs due to the limited RNIC cache. If the accessing information is not cached in the RNIC, it needs to fetch the desired data in memory. This process slows down the packets processing of RDMA transport. Dynamically Connected Transport (DCT) has been proposed as a hardware solution to the scalability problem [4]. DCT reduces RNICs cache miss by dynamically creating and destroying one-to-one connections. However it requires three additional network messages when the target machine of a DCT queue pair changes: a disconnect packet to the current machine, and a two-way handshake with the next machine to establish a connection [1]. DCT increases the number of packets associated with each RDMA request [16]. Other approaches have also been proposed to reduce RNIC cache miss, like QP sharing (contention for locks), using huge page to reduce mapping entries of physical address to virtual address [10], or adopting UD transport to reduce QPs required by server applications (small message but CPU overhead on both sides) [15, 16]. Our implementation also leverages the huge page and shared QPs. While, we introduce the design of a lock-free shared QPs approach.

As shown in Figure 3, for sending task, applications push requests to RDMAvisor, and get responses from RDMAvisor. Both requests and responses are delivered by the shared memory. This process is similar to the producer-consumer problem [18]. Applications produce send requests and RDMAvisor consumes WQEs. Since we run RDMA operations in user space, RDMAvisor applies shared memory with event descriptor for notification. Such operation reduces the involvement of kernel. Therefore, we can significantly reduce the intra-process communication cost.

Here, we explain how to identify the connection in the "producer-consumer" model. In RDMAvisor, all connections targeting at the same node are sharing with one QP. We use a 4 byte data field in WQE to identify connections, called virtual QP Number (vQPN). First, a vQPN is assigned to a new connection when it is generated. Then, RDMAvisor places connections' vQPN in some fields of WQE when pushing WRs to queues.

RDMAvisor uses a data field called imm_data of WQEs and CQEs to transfer connections' vQPN, multiplexing and demultiplexing shared QPs respectively. For two side verbs, as shown in Figure 7, the initiator places connections' vQPN in WQEs, then pushes them to the shared QP. As the RNIC of the initiator finished processing WQEs, the RNIC of the destination node generates CQEs, which copies the imm_data of WQEs to the imm_data field of CQEs. The destination node can use imm_data of CQEs to dispatch payloads for the corresponding connections.

For RDMAvisor, Poller in the destination node is able to identify the source by accessing the vQPN of each connection in the field of CQEs, and then deliver data to the corresponding application. As mentioned above, we place logical connection ID in WR to recognize the right connection in shared QPs. This operation is lock free and easy to implement. Moreover, sharing QP can not only reduce NICs cache misses, but also improve the probability of batching WRs among applications, which in turn improves the throughput performance.

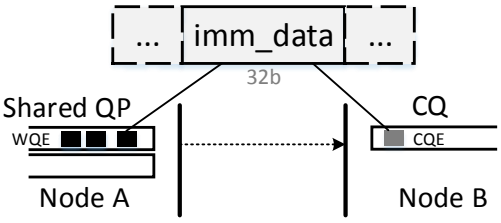

Figure 7: vQPN in the imm_data field of WQE/CQE

It is notable, however, that for one side verbs like RDMA Read and Write, the destination node is bypassed so that the RNIC at the destination is unable to generate CQEs to indicate the ending of transmission. FaRM [10] used a zeroed memory region as indicator of incoming RDMA Write, and applied a thread to continuously checking the status of the indicator. After processing the incoming data, the indicator will be reset. Since both sides of the communication should carefully synchronize several pointers, this design is complicated and hard to maintain.

To this end, RDMAvisor generates an additional small control message to pass the synchronization information from the initiator to the target node. Since RDMA Read or Write are used to deliver large message (usually over 1MB), the overhead of such additional control message is negligible. Specifically, this process can be further simplified for RDMA Write operation by applying the "RDMA Write with immediate data" which allows transfer of another piece of status information simultaneously with the message and returns as a special field of the RECEIVE CQE status. Thus, applying the RDMA Write with immediate data operation can avoid issuing additional control messages.

3.5 Remote Atomic Operations

RDMA also supports remote atomic operations like CMP_AND_SWAP and FETCH_AND_ADD, which is widely used for distributed transaction processing [7, 24]. However, in our practical and studies in [9] revealed that RDMA verbs "remote atomic operations" are not efficient. The poor performance of atomic operations occurs due to internal lock in RNICs. RNICs have multiple Processing Units (PUs) and the atomic operations always harm the parallelism, which produces a bad influence on the overall performance.

To solve this problem, we provide more efficient remote atomic operations by moving the atomic operations from RNICs to RDMAvisor. RDMAvisor takes advantage of C++11 library, which supports atomic_uint_least64_t type of data. Objects of atomic types are the C++ objects that are free from data races. Therefore, RDMAvisor atomic operation only needs to create a map with unsigned 64 bits address as key and objects of atomic_uint_least64_t as value. By so doing, client can initiate SEND operations with 64 bits address in payload and the server will get the specific objects of atomic_uint_least64_t according to the address given in the payload. To this end, RDMAvisor not only removes lock contention in RNICs but also ensure lock free in the whole data path of the remote atomic operations.

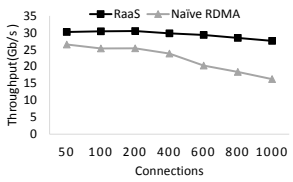

Figure 8: Scalability

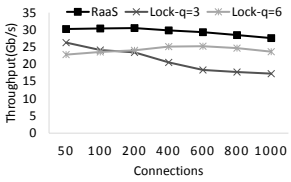

Figure 9: Throughput vs. QPs sharing

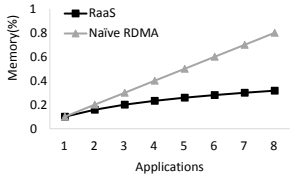

Figure 10: Memory usage

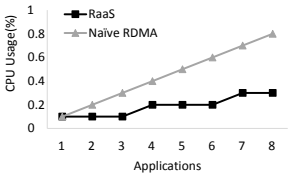

Figure 11: CPU consumption

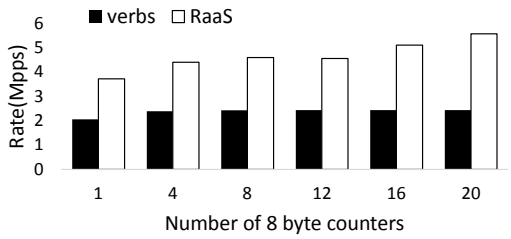

Figure 12: Atomic operations

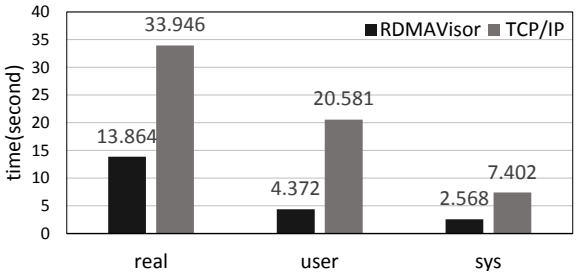

Figure 13: Latency of migrating database

3.6 Resources sharing

Here, we explain the shared SRQs and CQs approach to improve the resource utilization. Resources are referred to memory and CPU here mostly. As we mentioned before, without RDMAvisor, deploying RDMA enabled applications faces severe waste of resources because each application possess their own RDMA module, which can be sharing like receiving buffers, polling thread and so on. RDMAvisor manages and controls resource sharing by combining basic assignment and demand assignment. Basic assignment means initial resources assignment if a new application requesting for RDMA service. Demand assignment is referred to the case that RDMAvisor appends more resources when detecting there is performance dropping due to lack of some kinds of resources.

4 EVALUATION

4.1 Setting

We evaluate the performance of RDMAvisor on a cluster with four nodes. Each node ran CentOS 7.1 on four 2.1 GHz Intel Xeon processors with a total of 24 cores. Each machine has 64 GB of memory and a 40 Gb ConnectX-3 RoCE network interface card.

4.2 Micro-Benchmark

To evaluate the scalability of RDMAvisor design, up to 1000 connections are generated on one machine to randomly read 64KB data from other machines. We compare the RaaS Read operations with the naive RDMA Read verbs where the QPs are not shared by those connections. As shown in Figure 8, the throughput of naive RDMA starts to drop when the size of connections, i.e., QPs, exceeds 400. RDMAvisor shows stable performance since RDMAvisor allows the reuse of QPs to communication with multiple QPs in remote nodes. It is notable that RDMAvisor outperforms the naive RDMA when the number of connections is even less than 400. This is because the reuse of QPs has higher opportunity of batching WRs than the case of one QP per thread. With the increase of connections, RaaS has a higher probability to realize QPs sharing.

We compare RaaS with the design of sharing QPs with locks [10], where each QP is shard by q threads. Figure 9 shows results when q is 3 and 6 by running random read. We can see that sharing QPs can reduce throughput when threads contend for locks. RaaS can mitigate the impact of lock contention. Specifically, our design is not sensitive to the number of q and cluster size since RaaS performance scales well with the connection number.

We compare the RaaS resource (memory and CPU) consumption with naive RDMA. The resource consumption is normalized according to the resource required by one application to set up the connection. Figure 10 shows the memory consumption and Figure 11 shows the CPUs overhead respectively. We can see that the resource required by the naive RDMA operations increases linearly with the increase of applications. However, the design of RaaS proposes more effective usage of resource which results in a slow growth of memory/CPU consumption.

Figure 12 shows the result of remote atomic operations implemented by RDMA verbs and RDMAvisor. We create 1 to 20 8-byte counters at server side and generate parallel atomic operations at client side. When using the naive verbs atomic operations, it achieves 2.4 million operations per second. While RDMAvisor can achieve up to 3.7-5.5 million operations per second with regard to different counters, which verified the effectiveness of our remote atomic operations.

To evaluate the performance of RaaS on applications. We used RDMAvisor to speedup a database migration procedure by using Xtra-Backup[**?**], which is a complete online backup solution for MySQL. The size of our migrating database is about 1.3GB. RDMAvisor enables fast development of service by simply replacing the traditional TCP Socket interface. Therefore, we only show baseline (TCP) results. As shown in Figure 13 and Figure 14, RDMAvisor reduces the migration period by 60% and involves less CPU cycles by around 40% due to the introduction of one-side RDMA write operation. The

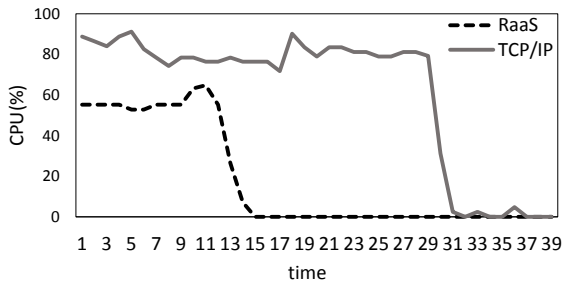

Figure 14: CPU utilization of migrating database

long delay is mainly caused by snapshot generation of XtraBackup instead of network transmission. It is notable that a short period of high CPU cost at time slot [10, 11] happens because RDMAvisor needs to deregister the memory region allocated for RNIC when the database migration accomplished.

4.3 Macro-Benchmark

4.3.1 RPC. We evaluate the performance of RDMAvisor on the application with short flows, e.g., RPC.

(1) a brief system review, the main characteristics of the data flows (2) the problem we met and the solution, like block and unblock mode.

4.3.2 Key-value Storage. We evaluate the performance of RDMAvisor on the application with short flows, e.g., RPC.

(1) a brief system review, the main characteristics of the data flows (2) the problem we met and the solution, like block and unblock mode.

4.3.3 Distributed Storage System. We evaluate the performance of RDMAvisor on the distributed storage system, Ceph []

(1) a brief system review, the main characteristics of the data flows (2) the problem we met and the solution, like block and unblock mode.

5 RELATED WORK

RDMA satisfies the demand of modern datacenter applications by providing high throughput (40-100Gbps), ultra-low latency (10 μs per hop) message transfers and low CPU overhead for transmission. Therefore, RDMA technology has attracted a lot of attentions in the cloud computing research community. For example, Microsoft's datacenter has been deploying RDMA over commodity Ethernet (RoCE) to support highly reliable, latency-sensitive services [12, 26], which has indicated the potential deployment of RDMA by replacing TCP for intra datacenter communications and achieve low latency, low CPU overhead, and high throughput. Specifically, priority based flow control (PFC) and QCN-based congestion control guarantee "lossless channel" and thus eliminate the complicated protocol stack to ensure reliable communication.

In-memory computing has started using RDMA to provide ultra-low latency access services. Over InfiniBand or RoCE network, significant improvement in performance for Hbase, Hadoop, Spark, RPC have been achieved by replacing the communication layer with RDMA [14, 19, 20]. The two-side verbs were usually applied to quickly replace the socket-based interface. However, for large size message communication in such write-once-read-many model, not fit

For small message communication, like key-value store, the key-value store is a typical application. Pilaf [21] used one-side RDMA READ for GET and SEND/RECV operation for PUT. They make this design decision igit s to have the server handle all the write operations and have the clients implement read-only operations.Since real-world workloads are skewed towards reads,this design captures most of the performance benefits of RDMA while drastically simplifying the problem of synchronization.

C-Hint [23] aims at efficient cache management for high hit ratio.They leverage one-side RDMA READ for Get and RDMA WRITE for accelerating item access history report.They favor WRITE for two reasons.One is that C-Hint bears much computing tasks and history report message can be fairly large (eg.8KB) ,excessively involving the servers can degrade the quality-of-service but WRITE can bypass remote cpu and save cpu cycles for computing tasks.The other is that for each client,the hydra server allocates a small memory container to allow the client to direct put a history report message via RDMA WRITE,so the address exchange only occurs once for each client and from then message can be write directly to remote memory.In contrast,leveraging SEND to implement this is not a good choice.

HERD [15] used UC WRITE operations for request and UD SEND for responses to eliminating the overhead fo posting RECV operations at the receiver side.UC is an ignored connection type.They choose UC/UD for that Infiniband and RoCE employ lossless link-level flow control,namely,credit-based flow control and Priority Flow Control.Reasons for packet loss include bit errors on the wire and hardware failures,which are extremely rare.So they sacrifices transport-level retransmission for fast common case performance at the cost of rare application-level retries,and FaSST(ijÓÒýÓÃ)do their work for the same reason.They favor WRITE over READ for diverse reasons from application level to hardware level.Basically,They aim at supporting key-value servers that achieve the highest possible throughput,which means that it's asymmetric.Only servers will bear heavy communication load from clients.In terms of hardware,it involves in latency and RNIC processing.one UC WRITE only takes half round trip but one READ need a whole round trip,the latency of an unsignaled WRITE is about half that of a READ.As UC does not generate ACK/NAK packets,it causes less network traffic than RC(here it indicates READ) and thus can support higher inbound throughput than with READs.because each queue pair can only service a few outstanding READ requests(16 in their RNICs).Similarly,at the PCIe level,reads are performed using non-posted transactions,wheras writes use cheaper,posted transactions.In outbound throughput,for small sizes,inlined WRITEs and SENDs have significantly higher throughput than READs.In terms of application level,key-value storage usually need to access hash table or tree structure and it takes multiple READs on average to retrive the value,which increases the latency and influences the throughput.Leveraging UD for response is for consideration of scalability and throughput.Although UD throughput is not so good as READ but better than multiple READ and UD is good at one-to-all efficient communication.For small size workload(48byte) a single

WRITE plus a single SEND outperforms than multiple READs in latency,scalability and throughput.

FaRM [10] demonstrated its effectiveness on key-value store and graph store [25] that uses RDMA-Write for request and RDMA-Read operation for message delivery, and leverage data locality to improve memory access latency. It is important to note that FaRM [10] is a general-purpose distributed computing platform. Our work can actually work as the RDMA network function for FaRM to access remote memory but let the designer focus on the data and memory management.

Studies on the graph computing have applied RDMA to speed up the graph query [22], speed up the in-memory distributed transaction processing with RDMA datagram RPCs [16] or advanced hardware features, like HTM [8].

A design guideline has been proposed on low-level RDMA design such as PCIe transactions and NIC architecture [17]. Generally, RaaS works as a middleware to support the communication for upper-layer in-memory computation (e.g., RPC) or storage (e.g., distributed share memory) while provide adaptive and sufficient performance for most applications. RaaS also allows flexible configuration by leveraging the lower-level RDMA design for specific services [17].

6 CONCLUSION

This paper presented a simple and scalable RDMA as Service (RaaS) to mitigate the impact of RDMA operational details. RaaS provides careful message buffer management to improve CPU/memory utilization and improve the scalability of RDMA operations. These optimized designs lead to simple and flexible programming model for common and knowledgeable users. We have implemented a prototype of RaaS, named `RDMAvisor`, and evaluated its performance on a cluster with a large number of connections. Our experiment results demonstrate that `RDMAvisor` achieves high throughput for thousand of connections and maintains low CPU and memory overhead through adaptive RDMA transport selection.

REFERENCES

[1] 2011. Dynamically-connected transport service. https://www.google.com/patents/US20110116512. (2011).

[2] 2015. *InfiniBand Architecture Volume 1 and Volume 2, released specification.* https://cw.infinibandta.org/document/dl/7859.

[3] 2015. *Mellanox Connect-IB produce brief.* http://www.mellanox.com/related-docs/prod_adapter_cards/PB-connect-IB.pdf.

[4] 2015. Mellanox Connect-IB product brief. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_Connect-IB.pdf. (2015).

[5] IEEE. 802.11Qau. 2011. *Priority based flow control.*

[6] InfiniBand Trade Association. 2014. *Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2 (IP routable RoCE).*

[7] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. 2016. The end of slow networks: It's time for a redesign. *Proceedings of the VLDB Endowment* 9, 7 (2016), 528–539.

[8] Yanzhe Chen, Xingda Wei, Jiaxin Shi, Rong Chen, and Haibo Chen. 2016. Fast and general distributed transactions using RDMA and HTM. In *European Conference on Computer Systems (EuroSys)*. ACM.

[9] Mellanox Documentation. *Performance Tuning Guide for Mellanox Network Adapters.* http://www.mellanox.com/related-docs/prod-software/.

[10] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. 2014. FaRM: Fast remote memory. In *USENIX NSDI*. USENIX.

[11] Philip W Frey and Gustavo Alonso. 2009. Minimizing the hidden cost of RDMA. In *ICDCS*.

[12] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over commodity ethernet at scale. In *ACM SIGCOMM*.

[13] InfiniBand Trade Association 2010. *Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A16: RDMA over converged Ethernet (RoCE).* InfiniBand Trade Association.

[14] Nusrat S Islam, MW Rahman, Jithin Jose, Raghunath Rajachandrasekar, Hao Wang, Hari Subramoni, Chet Murthy, and Dhabaleswar K Panda. 2012. High performance RDMA-based design of HDFS over InfiniBand. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society Press, 35.

[15] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2014. Using RDMA Efficiently for Key-value Services. In *SIGCOMM*.

[16] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. FaSST: fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In *USENIX OSDI*.

[17] Anuj Kalia Michael Kaminsky and David G Andersen. 2016. Design guidelines for high performance RDMA systems. In *USENIX ATC*. 437.

[18] S Li, E Clarke, T Suzuki, and E Gupta. 2002. Analysis of the producer-consumer problem. *Journal of Large-Scale Archetypes, vol. 0* (2002), 72–92.

[19] Xiaoyi Lu, Nusrat S Islam, Md Wasi-Ur-Rahman, Jithin Jose, Hari Subramoni, Hao Wang, and Dhabaleswar K Panda. 2013. High-performance design of Hadoop RPC with RDMA over InfiniBand. In *International Conference on Parallel Processing (ICPP)*. IEEE, 641–650.

[20] Xiaoyi Lu, Md Wasi Ur Rahman, Nahina Islam, Dipti Shankar, and Dhabaleswar K Panda. 2014. Accelerating spark with RDMA for big data processing: Early experiences. In *HOTI*.

[21] 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*.

[22] Jiaxin Shi, Youyang Yao, Rong Chen, Haibo Chen, and Feifei Li. 2016. Fast and concurrent rdf queries with rdma-based distributed graph exploration. In *USENIX OSDI*. 317–332.

[23] Yandong Wang, Xiaoqiao Meng, Li Zhang, and Jian Tan. 2014. C-Hint: An Effective and Reliable Cache Management for RDMA-Accelerated Key-Value Stores. In *ACM Symposium on Cloud Computing (SoCC)*.

[24] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. 2015. Fast In-memory Transaction Processing Using RDMA and HTM. In *SOSP*.

[25] Ming Wu, Fan Yang, Jilong Xue, Wencong Xiao, Youshan Miao, Lan Wei, Haoxiang Lin, Yafei Dai, and Lidong Zhou. 2015. GRAM: scaling graph computation to the trillions. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC)*. ACM, 408–421.

[26] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. In *ACM SIGCOMM*. ACM, 523–536.