Unified Virtual RDMA for Hybrid Virtual Environments

ABSTRACT

Supercomputing is more and more popular in many fields for its extreme high-performance, such as artificial intelligence and big data processing. For efficiency and elasticity, supercomputing is going cloud recently, especially hybrid virtual environments (virtual machines and containers). RDMA is basic high-performance network in supercomputing. In hybrid virtual environments, unified RDMA virtualization is necessary because of integrated management and high resource utilization. However, existing RDMA virtualization solutions is not unified. To solve this problem, we present uniRDMA, a unified RDMA software virtualization framework. In uniRDMA, vRNICs (virtual RDMA network interface cards) are virtualized with basic RDMA attributes (e.g. QP) in host user-space. In VMs or containers, the driver of vRNICs is unified and vRNICs' RDMA resources are mapped to applications for performance. The virtual layer manages vRNICs' instantiation (including mapping vRNIC to physical cards) and constructs a virtual RDMA network. In our evaluation, uniRDMA can be deployed in large-scale hybrid virtual environments within 5% overhead to native RDMA.

KEYWORDS

RDMA, Virtulization, Hybrid Virtual Environment

ACM Reference Format:

. 2021. Unified Virtual RDMA for Hybrid Virtual Environments. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

RDMA (remote direct memory access) is a high-performance network, with high throughput, low latency and low CPU load. Thus, RDMA is widely adopted in supercomputing for various applications, such as artificial intelligence, and big data processing.

Recently, migrating to cloud is a trend for supercomputing users. Different from traditional HPC environment, resource provisioning are done in the form of virtual machines or containers in clouds [4]. Thus, to use RDMA for applications in clouds, RDMA devices need to be virtualzed. Existing solutions for RDMA virtualization include software-based and hardware-based solutions. The software-based solutions are mainly about HyV, FreeFlow or MasQ [12] [10] [9]. FreeFlow is designed for containers and HyV/MasQ are proposed for virtual machines. The hardware-based solution is based on PCI pass-through for less overhead, like SR-IOV [7].

However, in practice, hybrid virtual environments are common for clouds. In specific, VMs, containers orother form of virtualization (e.g. containers in VMs) can be found in the same datacenter or even on the same host machine. The existing solutions are not suitable for hybrid virtual environments for the following reasons:

For software-based solutions, they are designed for specific virtual environments. Thus, in hybrid virtual environments, we should deploy multiple frameworks or extend one framework for another environment. If multiple frameworks co-exist in the same cluster, RDMA resources should be divided statically to avoid management conflict. This may cause the low resource utilization and higher management complexity. If single framework is extended to hybrid virtual environments: for FreeFlow, the communication between applications and the virtual layer does not suit VMs and it has apperant overhead in RDMA network without bypassing the virtual layer; for HyV or MasQ, compared to FreeFlow, their virtual layers are in kernel-space that brings new problems: the kernel's attack surface is larger due to lots of inserted code in kernel, managment development is inflexible in kernel programming, and the inserted modules are depenedent on hardware-specific RDMA kernel drivers.

For hardware-based solutions (e.g. SR-IOV), RDMA device resources are directly allocated to virtual machines or containers. Thus, the device utilization is static and inefficient. Moreover, RDMA networks in clouds are still managed by physical switches or routers. Thus, virtual network management is not scalable and portable in large-scale clouds.

To address these problems, we propose a unified RDMA virtualization framework for hybrid virtual environment in clouds, namely uniRDMA, also with the design goals of flexible management and high-performance.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

As the basic abstract unit of uniRDMA, each vRNIC (virtual RDMA network interface card) is virtualized in user-space with basic RDMA attributes, such as QPs (Queue Pairs). Thus, vRNICs are flexible and hardware-awareness. To support VMs or containers unifiedly, vRNICs are the device of VMs and containers and a specific kernel driver is in guest OS. VMs and containers use the same communication protocal to vRNICs. Moreover, to optimize the performance, RDMA resources in vRNICs are mapped to applications in VMs or containers.

Virtual layer tacks charge of vRNICs in each host server, including vRNICs instantiation and vRNICs mapping to physical RNIC. Besides, the virtual layer is also a software virtual switch with configuring vRNIC address and routing rules.

Finally, we implement the prototype and evaluate it in different aspectsbenchmarks, such as throughput, latency, scalability, with and real-word applications. From the result, uniRDMA's performance is close to native RDMA in hybrid virtual environment and the overhead is less than 5% to hardware-based virtualization. uniRDMA also has high scalability and adapts to real-world RDMA applications in hybrid cloud environments. The main contributions in this paper are as follows:

- We proposed vRNIC, which is a complete virtual RDMA device in host user-space for high flexibility and hardware independency. vRNIC is unified for both VMs and containers with specific driver. Meanwhile, the performance is optimized by mapping vRNIC to physical RNIC same as native RDMA.
- We proposed a virtual layer for vRNICs, which is responsible for vRNICs' instantition and mapping to physical RNIC. Also, virtual RDMA network are configured in the virtual layer.
- The whole unified framework, namely uniRDMA, is evaluated and the results proved that uniRDMA maintains high performance close to native RDMA.

The paper is organized as follows. Section 2 describes the backgroud of uniRDMA designs. Section 4 describes the vRNIC design in uniRDMA, and Section 5 introduces the design of virtual layer. Then, Section 6 reports the experimental results. Section 7 introduces related work. Finally, Section 8 concludes our work.

2 BACKGROUND

Virtualization of VMs and containers are in different spaces because of their characteristics. VMs need emulate whole virtual hardware environments for guest operation system with hypervisor (kernel-space). So, applications in VMs is isolated by guest OS with more security but higher overhead. Containers are shared with the host OS but with runtime isolation. So, containers have low overhead and fast boot-up time.

For software virtualization of I/O devices, there are two main technologies: full virtualization and para-virtualization. Full virtualization is couple with hypervisor and emulate all device interface for VMs. Thus, application in VMs can directly use the devices without custom driver. Para-virtualization is flexible with the help of custom dirver for the backend device. In specific, the backend device can be in kernel-space or user-space. In kernel-space, the virtual devices are couple with the kernel (hypervisor), e.g. vhost-net. In user-space, the virtual devices can be in another processes and independent of kernel, e.g. vhost-user-net.

RDMA (Remote Direct Memory Access) has hardware protocol stack and zero copy technology, so applications can bypass the kernel to read and write remote memory data, without the participation of remote CPU. As a result, RDMA has high throughput, low latency and low CPU load. Applications need to use Verbs interface when using RDMA. In Verbs, as shown in Figure 1, RDMA is separate control path and data path. The former is the management about RDMA context, mainly including lots of RDMA resources, such as Queue Pairs (QPs), and Memory Regions (MRs). the operations like ibv_create_qp, reg_mr; The latter is the usage of RDMA context and resources, which is the data commands like ibv_post_send and ibv_post_recv. In a RDMA workflow, the communication is based on Queue Pair (QP). The application writes the RDMA work request to the QP, and then "press" the RNIC's doorbell register, which is mapped to application when context init, and the RNIC's hardware processor will execute the work request in the QP to forward data. For applications, the entire operator is in the user space.

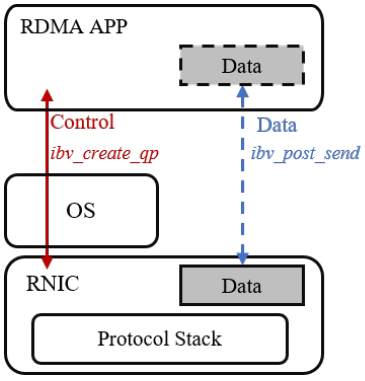

Figure 1: Native RDMA Feature

3 OVERVIEW

For hybrid virtual environments in clouds, RDMA virtualization not only needs to be unified, but also maintains high

performance and manageability. Therefore, our design goals are as follows:

- `Unification`: Only single RDMA virtualization framework needs to be deployed in hybrid virtual environments and it provides centralized management. To form unified RDMA virtualization, single centralized virtual layer should be set up, which is provided to virtual machines and containers with general interfaces.
- `High performance`: Performance of virtual RDMA should be close to native RDMA in terms of throughput, latency, or CPU load. Meanwhile it should suit for large-scale virtual cluster.
- `High manageablility`: Basic managment shoud be meet for the clouds, such as, performance isolation, virtual network management or portability.

To achieve above goals, we propose a software RDMA virtualization framework, namely uniRDMA. As Figure 2 shows, uniRDMA consists of vRNICs(including its' driver/library in VMs or containers) and the virtual layer:

vRNICs is a simple software emulation of physical RNIC. Each vRNIC is instanced in the virtual layer when VM or container's RDMA applications start. The RDMA commands of applications can be transported to vRNIC through guest driver or container libraries. Besides, RDMA resources (e.g. QPs and DoorBells) in vRNICs are mapped to physical RNIC and upper applications. Thus, the data commands in application can be executed locally for high-performance.

The virtual layer is also in host user-space. We design it for centralized management of vRNICs. It controls all RDMA devices through RDMA verbs library in host user space. Meanwhile, vRNICs are configured in the virtual layer to construct virtual RDMA network, e.g. vRNIC addresses and routing rules.

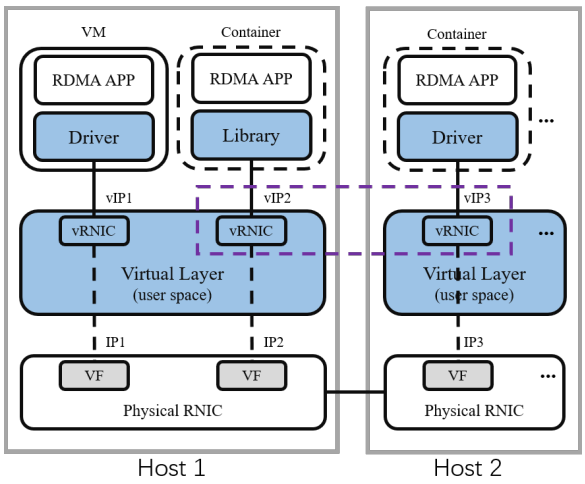

Figure 2: uniRDMA Framework Overview

4 VRNIC DESIGN

With software complete virtualiztion, vRNIC is the core unit for unified RDMA virtualization. In this section, we introduce the detail design for vRNIC, including vRNIC virtualization, vRNIC driver and optimizations for performance.

4.1 vRNIC Virtualization

The first problem is which space vRNICs are located: host's kernel-space or host's user-space. We found that both kernel-space and user-space are feasible to construct vRNICs. However, compared to kernel-space, there are multiple advantages for vRNICs in user-space, such as minimal attack surface, flexible management and independent on RDMA kernel drivers. Thus, we choose the user-space for vRNIC virtualization.

We find that RDMA resources (e.g. QP, MR or DoorBells) are the key roles in whole RDMA communication. In control path, the application creates queue instances such as QP, and registers memory regions (MRs) in host memory; In data path, the application writes DoorBell to notify RNIC to deal with WQE in QP and transform data in MRs. Thus, vRNIC virtualization is mainly about how to construct the virtual RDMA resources to provide complete RDMA communication.

Based on above analysis, we summarize that RNIC has two kinds of hardware properties about theses RDMA resources, namely static property and dynamic property:““

For static properties, since RDMA sends and receives messages based on QPs, MRs and other RDMA resources, it can be abstracted that RNIC has the following buffers:

- `Queue Buffer`: storing information of queue instances, such as QP number, QP state and CQ number. The network card uses the information to read and write work requests, establish connections with remote QPs, etc.
- `Data Buffer`: storing the information of registered memory regions, such as page table, memory key, etc. The network card uses the information to access local or remote memory for data transform.
- `Doorbell Buffer`: including multiple doorbell registers. The network card uses it to accept user commands and notify the internal hardware processor to perform DMA, processing and forwarding.

For dynamic attributes, Therefore, the state of RDMA resources are dynamic in control path and data path. Control path includes the creation and destruction of RDMA resources. RDMA resources or information in buffer are always changed. For example, RNIC records the QP number when QP is created, changes QP state for RDMA connection and clear QP information in the destroy. Notify that this process has less latency due to the kernel, Data path is mainly

about the usage of RDMA resources. RDMA resources or information in buffer are always maintained in RNIC. When RDMA applications post a send or receive operation, only write the DoorBell and the hardware processor performs DMA, encapsulates and forwards data.

To emulate the static attributes, virtual queue, data and doorbell buffers are respectively set up in vRNIC. For example, the QP buffer stores virtual QP information and the virtual doorbell buffer is including virtual doorbells. Virtual buffer is flexible and unlimited for the numbs of RDMA resources instances.

To emulate the dynamic attributes, we need an overlay network to realize the RDMA's control path and data path. For example, when application calls post_send, the overlay network in vRNICs can transport the corresponding data to remote. Thus, in our design, the overlay network is flexible for multiple choices, such as, physical RDMA, TCP/IP network or even share memory in the same server. Obviously, the performance of overlay network is critical for our vRNICs. In this paper, uniRDMA is based on physical RDMA.

Fortunately, we found all RDMA resources information in RNIC are only changed in control path and maintained in data path. So, we can map the virtual RDMA resources to RNIC only in control path, such as QP and Doorbell, and do not need introduce any operations in data path. Mapped RDMA resources are directly used and RNIC is notified by mapped virtual DoorBell in vRNIC. As a result, vRNIC are still with DMA zero-copy, hardware protocol stack processing and other high-performance capability in data path. Therefore, we put each vRNIC with a map unit. As Figure 3 shows, it maps or unmaps virtual RDMA resources from vRNIC to RNIC in control path:

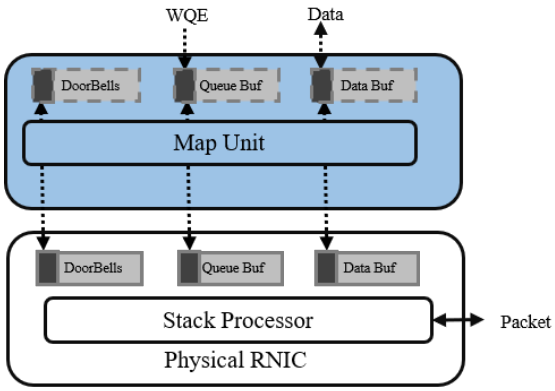

Figure 3: Map Unit in vRNIC

For RDMA resources (e.g. QPs, CQs): Taking QP as an example, regularly, vRNIC record virtual RDMA resources information when the virtual QP instance are created. However, virtual QP are still not generated-associated with the RNIC. To make the mapping, the map unit will create corresponding real QP instance in RNIC based on the information of virtual QP instances, such as the same memory address information and the same device id. Equivalently, the virtual QP information are recognized in RNIC, such as QP number and QP state, and can be one-to-one synchronous with RNIC's physical instance by lots of similar map operations in control path. All operations can be completed by calling the Verbs interface of RNIC in user space. After the mapping is completed, the work requests in the vRNIC virtual QP can be zero-copied into the RNIC. Also, data in registered memory of vRNIC can also be zero-copied to RNIC in the same way.

For DoorBells: It needs to be mapped to the hardware doorbell in the physical NIC device space, so that vRNIC can notify the RNIC hardware processors. In vRNIC, the mapping unit will map the virtual address of the virtual doorbell to the hardware doorbell address of the corresponding physical NIC device space through a system call. As shown in Figure 4-2, after the mapping is completed, the write operation to the vRNIC virtual doorbell is equivalent to performing the doorbell notification to the RNIC.

Map unit is the key for vRNICs' performance. Note that all mapping relationships are all one-to-one, therefore, the correctness and isolation of resources in different RDMA context are guaranteed. Meanwhile, because the mapping operation is only executed in control path, the overhead is one-off compared to data commands. For the data path, vRNICs can directly utilize the hardware processing capability of RNIC, such as DMA zero-copy and hardware protocol stack processing.

4.2 Unified vRNIC Driver

vRNIC is virtual device with complete RDMA attributes. However, vRNICs are still independent software in host user space. Thus, for VMs and containers, the driver (or library in containers) for vRNICs should be designed.

For containers, vRNIC can be directly provided to RDMA applications in containers with some modification in containers' verbs library. However, vRNICs and VMs' application are isolated with the hypervisor and guest OS. Thus, vRNICs needs to be recongnized by hypervisor and then provided by guest OS. We use I/O virtualization technology to extend each vRNIC as VM's I/O device. Then, driver for this device is installed in guest OS to support the I/O process. And the detail works are as following:

The existing I/O virtualization technologies are mainly divided into full virtualization and paravirtualization. The full virtualization completely simulates all the functions of the device, there are frequent context switching and data copy overhead.

In contrast, paravirtualization does not emulate the hardware complementally to reduce the times of data copy and switch. Therefore, we exploit paravirtualization to expand vRNIC as an I/O device. In our design, the I/O channel between vRNIC and the virtual machine is a shared memory queue, which is created by file; the signal and interrupt mechanism can be realized through the event descriptor between virtual layer process and each virtual machine process, then the event notification is converted into an internal interrupt signal by virtual machine monitor.

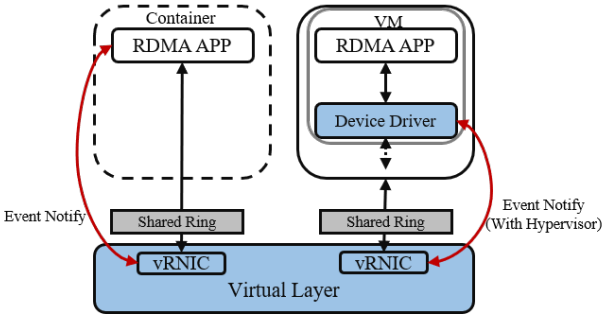

Figure 4: Unified vRNIC Driver

The goal of the device driver is to support I/O process inside each guest. As shown in in Figure 4, the commands of RDMA application be forwarded into the memory-shared queue, and trigger events to notify the vRNIC to process them; similarly, the device driver receives interrupt notifications and reads the result from vRNIC. In short, the device driver can be implemented by a lightweight kernel module.

For generality, As shown in in Figure 4, the same design as virtual machines is adopted for containers: in I/O channel, the file-based shared queue is also used; in the synchronization mechanism, the same event notification mechanism is used. But remind the container does not fall into the monitor or inject interrupts during the synchronization.

For multiple containers and virtual machines, if shared memory files for vRNIC driver are not isolated, they can be discovered by every container through scanning files. In order to solve this problem, we run the virtual machine in a isolated container environment. As shown in in Figure 5, based on the container's mount namespace [2], we respectively place the shared files of each vRNIC in the dedicated directories and mounts each directory to the corresponding container(including containers running virtual machines). As a result, due to the isolation of the mount namespace, the shared files of each vRNIC in the virtualization layer are only visible to the used container or virtual machine.

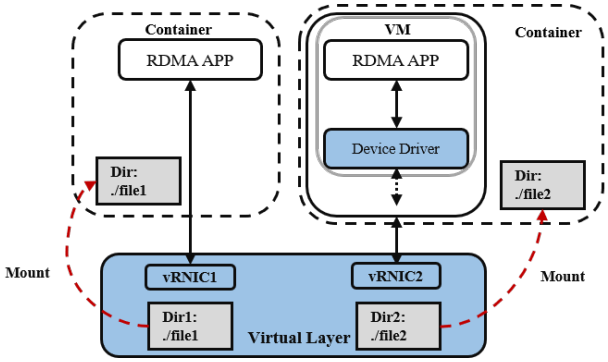

Figure 5: Isolation for vRNIC driver

4.3 Performance Optimization

Trough vRNIC driver, all commands of RDMA applications can be excuted in vRNICs. However, in data path of native RDMA, there are zero-copy and bypassing for high-performance. Thus, to make vRNICs' performance close to native RDMA, vRNICs and applications should realize the same optimizations.

(1) zero-copy optimization: The fact of zero-copy is that both processes have common available physical memory pages. Same as native RDMA, the zero-copy contents are including the RDMA work request in QPs and data in MRs. Similar as the above I/O channels, we use shared memory to map applications' memroy to vRNICs' RDMA resources (e.g. QPs and MRs). And this mapping is feasible for both VMs and containers.

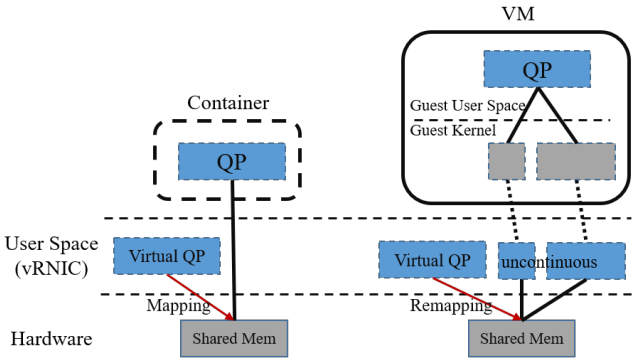

Figure 6: Mapping QP to vRNIC

However, in the virtual machine, due to the memory management mechanism of guest operating system, the virtual machine's physical memory of the RDMA resource may be not continuous, and the mapped memory area in vRNIC is not continuous like Figure 6. So, vRNIC cannot map the virtual memory area as a virtual RDMA resource to RNIC. To

solve this problem, the virtual memory remapping mechanism in user space is used. It remaps the discontinuous RDMA resource virtual memory area in vRNIC to the a block of continuous virtual memory, and the sequence of mapped physical memory page must be unchanged.

(2) Bypassing optimization: Pressing the doorbell is necessary in RDMA data path to drive RNIC. In vRNIC, the doorbell that is mapped to physical RNIC, still need to be mapped to RDMA application to meet bypassing. Otherwise, the pressing command needs be forward to vRNIC and that's imports apparent latency in data path.

However, the RDMA application and the vRNIC belong to two different processes on the host, and they have isolated virtual address spaces. At the same time, the doorbell register is in the device address space and cannot be mapped by shared memory. The key to solving this problem is that the process of RDMA application needs to know the physical address of the doorbell register.

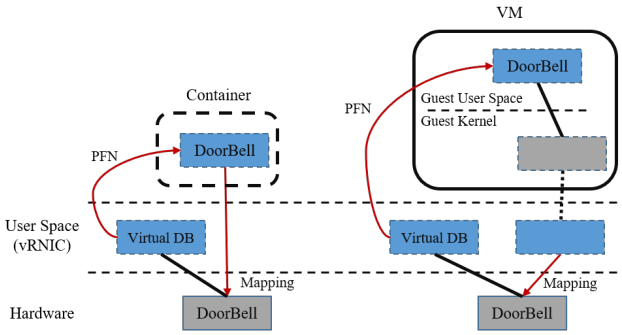

Figure 7: Mapping DB from vRNIC

Therefore, when an RDMA application creates a RDMA context, as shown in Figure 7, it sends a request to the vRNIC at first. Under the supervision of virtual layer, vRNIC forwarded to application with the corresponding physical address of the doorbell, commonly the physical page number. After that, the application maps its doorbell virtual address to the physical page in its own process, that needs the host kernel and hypervisor if the application in virtual machines.

5 VIRTUAL LAYER DESIGN

Each vRNIC is still independent unit in host server. Thus, we design a virtual layer to manages vRNICs centrally, including vRNIC instantiation, vRNIC mapping to RNIC. Besides, virtual layer manages the virtual network for VMs or containers, such as configuring virtual vRNIC address, routing rules or more management in clouds.

5.1 vRNIC Device Management

Each vRNIC is instantiated in the virtual layer. We found that another differences for VMs and containers: VM is a unstopped process for host onece it boosts, even though RDMA applications in VM is not running; container's RDMA application is a regular process for host. Thus, we make static instantiation for VM's vRNIC before VM boosts, and make dynamic instantiation when the verbs library are loaded in container's application. In contrast, we destroy the vRNIC instance when VM stoping or container's application exiting. Besides saving system resources, another advantage of this is to avoid contend and assure corrects for multipe applications in one container. Note the contend for mutiple applications in one VM are solved with guest OS and our kernel vRNIC driver.

In this paper, we use the physical RDMA as the overlay network of vRNIC. Thus, the virtual layer should manage the mapping from vRNIC to RNIC. To achive performance isolation, we utilize the SR-IOV, which virtualize multiple isolated VFs (different hardware interfaces) in one physical RNIC. The virtual layer maps vRNICs to VFs respectively.

However, we note that VF resources of SR-IOV are limited, for example, only 126 VFs are supported in Mellanox ConnectX-3 at most [1]. Therefore, the existing VFs need to be managed and coordinated in a unified to meet many vRNICs.

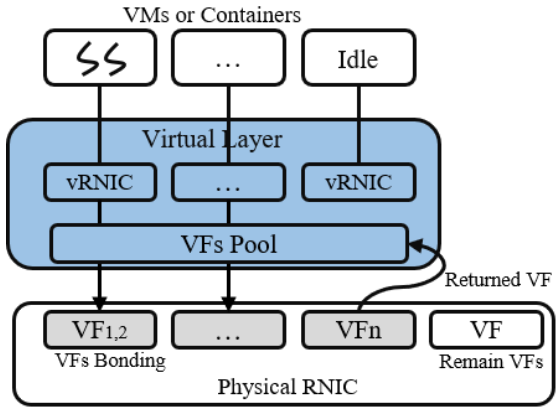

Figure 8: Management of vRNIC Mapping

As Figure 8 shows: First, the virtual layer constructs a dynamic VF pool. The initial number of VFs in the pool is usually the number of pre-determined virtual instances. If lack of free VFs in the pool, the device pool can dynamically expand the number of VFs. Second, the virtual layer supports dynamic mapping between vRNIC and VF. When all virtual RDMA resources have been destroyed, the virtual layer marks the VF as idle and puts it back into the pool.

So that the virtual layer can support the number of vRNICs that exceed the VF limit. Finally, the virtual layer supports various mapping relationships between vRNICs and VFs. For example, load balancing can be meted by map a vRNIC with multiple VFs. VF resources are saved by mapping multiple vRNICs of the same virtual instance to the single VF.

5.2 Virtual RDMA Management

To maintain portability and realize RDMA network management, RDMA connections between vRNICs cannot be established by the physical address of VFs. But this problem has a solution that uniRDMA virtual layer acts as a software RDMA switch or router for virtual RDMA network configuration and routing management, etc.

RNICs are usually managed by the subnet manager in the cluster. For the same purpose, a control center is set up in uniRDMA to assign virtual RDMA addresses vGIDs to each vRNIC and configure routing rules between vRNICs. As shown in Figure 9 , the vRNICs are divided into two groups: group 1 and group 2; vRNICs in the same group are allowed to establish RDMA connections, and cross-group RDMA connections cannot succeed due to the isolated routing rules between group 1 and group 2.

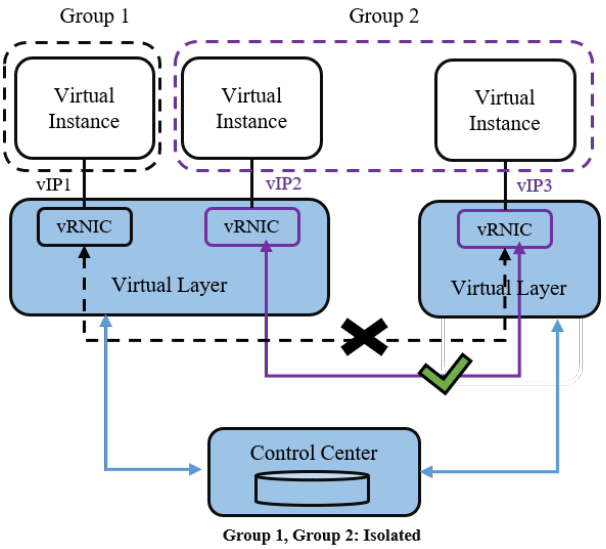

Figure 9: Virtual RDMA Network Routing

Consistent with native RDMA, vRNICs in each virtual layer need to exchange each other's virtual RDMA addresses, virtual QP queue information, registered memory keys and other information to establish virtual RDMA connections. However, the vRNIC RDMA address is virtual, and does not recognized in RNIC. Therefore, the mapping relations between the virtual addresses of vRNICs and the physical addresses of VF needs to be exchanged between virtual layers. When establishing the virtual RDMA connection, the virtual RDMA address is converted to the physical address of the mapped VFs. Note that RDMA resources information, such as virtual QP number and memory keys, have been mapped to the VF interface by the map unit in vRNIC, they can all be recognized VF and directly used to create RDMA connection.

5.3 Discussion

In virtual layer, besides above managements, there are more ongoing features for clouds:

Control police for clouds: To manage the resources precisely, there are lots of important polices in clouds, e.g. QoS, metering and so on. In uniRDMA, even though all RDMA commands in data path are executed in applications' user space, we can still meet the control features in virtual layer for these resons: First, the virtual layer can monitoring the QPs or CQs for RDMA traffic information, such as, size and status of one RDMA work request, because all RDMA resources (including QPs and CQs) are mapped at host's virtual layer; Second, the virtual layer can control the virtual RDMA traffic by controling the RDMA resources, with the help of host RDMA libraries. For an example, we can destroy the QP in the virtual layer when a RDMA connections' traffic is excessive. These will cause more CPU overhead in the host virtual layer, but would not impact the performance of RDMA applications in clouds.

Virtual Instances Migration: Migration is important for both containers and VMs in clouds with many benifits, e.g. resource utilization and fail-over. With the virtual RDMA network , uniRDMA can support offline migration without reconfiguring the physical RDMA network for applications. In specific,after rebooting the migrated virtual instance, the application can rebuild the RDMA connection only by modifing the address mapping in software virtual layer. Currently, for live migrations, it is still hard because memory regions in RDMA application may be uncertain under bypassing or one-side communication. Thus, uniRDMA does not support live migrations for both VMs and containers.

Other network extensions: RDMA can also be exploited to optimize the performance of other network applications, such as TCP/IP. Existing works include vSocket [13], SockDirect [11] so on. In uniRDMA, we can extend the simiar design for socket applications in containers and VMs: unified vNICs and the driver of vNICs in containers or VMs. The difference in the virtual layer is mainly how to emulate the vNICs based on host RDMA's libraries.

6 EVALUATION

In this section, we evaluate the uniRDMA prototype on the physical RDMA platform. In addition to the basic network

performance, such as throughput and latency, it also includes real-world RDMA applications. We expect to answer the following questions: (1)Can uniRDMA's network performance in container and virtual machine environments be close to native RDMA? (2)Does uniRDMA have high scalability in both large-scale container and virtual machine cluster environments? (3)Can uniRDMA be adapted to the real RDMA application environment in containers and virtual machines?

6.1 Experiment Methodology

All experiments are carried out on two servers. The settings mainly include three parts: host server, container and virtual machine.

The RNIC used by the server is Mellanox ConnectX-3 56 Gb/sec, which performs RDMA communication under Infiniband. For host servers, the operating system is all CentOS 7.4.1708 (Linux 3.10.0-693.el7.x86_64). The RDMA driver installed on the host server is Mellanox OFED 4.4-2.0.7.0 [6]. To get consistence, virtual machines and containers are built based on the same OS images as host. All virtual machines are based on QEMU(5.1.50) [8] enabled with KVM [5] and the container engines are Docker(18.06.1-ce) [3]. In addition, the entire uniRDMA framework is compiled with GCC/G++ 4.8.5, and the O3 compilation optimization level is selected.

6.2 Basic benchmark

Throughput and latency are the key target of network performance. RDMA supports two different data transmission modes: unilateral and bilateral. Due to the difference performance between them, we evaluate them respectively.

Based on the RDMA benchmark test tool Mellanox perftest, we evaluated the throughput and latency of uniRDMA, native RDMA, hardware virtualization SR-IOV, and software virtualization FreeFlow in virtual machines or containers. For bilateral operations (Send and Recv), we use the "ib_send_bw" and "ib_send_lat" commands; for two unilateral operations (Write and Read), with Write as the representative, we use the "ib_write_bw" and "ib_write_lat" commands. The specific process is: after the RDMA connection is established between the client and the server, the bytes of transmitted message each time will be increased from 4B to 1MB, the data will be iteratively transmitted 1000 times with each message size, and finally the average throughput and latency are calculated.

(1) Throughput: The results of bilateral operation are shown in Figure 10, and the one of unilateral operation are shown in Figure 11. Whether uniRDMA is in a virtual machine or in a container scenario, the throughput of its bilateral and unilateral operations is similar as SR-IOV and close to native RDMA.

Compared with FreeFlow, when the message is small, the throughput of uniRDMA has reached 4-6 times that of FreeFlow. Because FreeFlow forwards all data commands to the software virtualization layer for processing. Therefore, the forward latency gradually accumulates and decrease the throughput significantly. However, uniRDMA maps all RDMA resources to execute data commands in the user space of the container or virtual machine. Therefore, there is no latency for commands forwarding in data path.

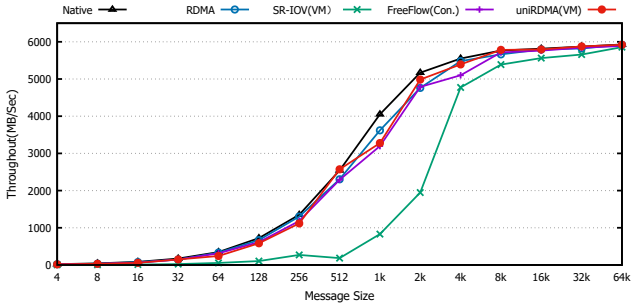

Figure 10: The Throughout of RDMA Send and Recv

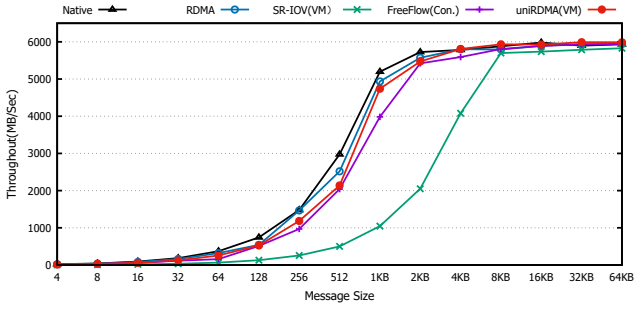

Figure 11: The Throughout of RDMA Write

When the message gradually increases, such as reaching 64KB, the throughput of each framework tends to be consistent. The reason is that the bandwidth is saturated, and the delay overhead of FreeFlow has been covered by waiting delay in RNIC.

(2) Latency: The results of bilateral operation are shown in Figure 12, and the one of unilateral operation are shown in Figure 13. Whether uniRDMA is in a virtual machine or in a container scenario, the latency of its bilateral and unilateral operations is similar as SR-IOV and close to native RDMA.

Compared with FreeFlow, when the message is small, the latency of uniRDMA has reached 40% 60% of FreeFlow because of FreeFlow's forwarding latency. Also, when the message gradually increases, such as reaching 64KB, the latency of each framework tends to be consistent. Because the main latency has been caused by RNIC data processing.

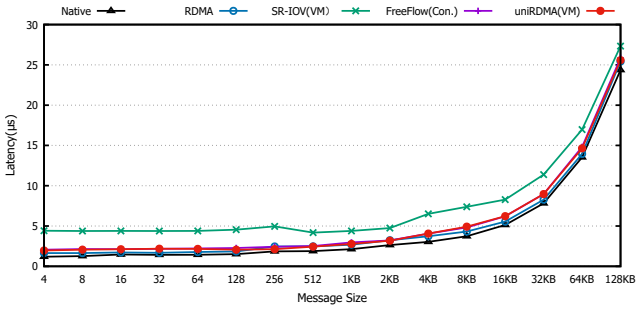

Figure 12: The Latency of RDMA Send and Recv

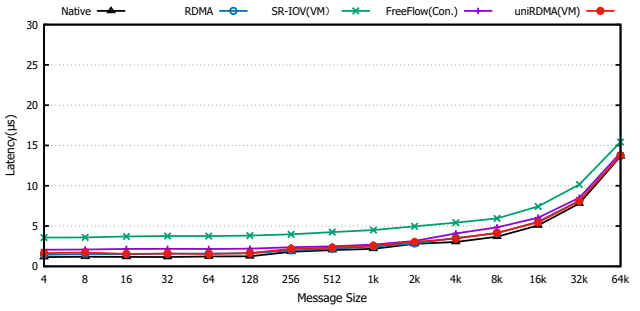

Figure 13: The Latenct of RDMA Write

(3) Scalability: Scalability is a challenge when RDMA virtualization in large-scale virtual clusters. To evaluate uniRDMA's scalability, for 2, 4, 8, 16, 32, 64, and 128 pairs of virtual instances, a random number of virtual instances are selected to execute "ib_write_bw" command with 128KB message. Moreover, there are full virtual machines, full containers and hybrid virtual instances(50% virtual machine and 50% containers) when evaluating. Finally, the average throughput between virtual instance pairs is shown in Figure 14.

From Figure 14, for all virtual environments, uniRDMA has good scalability and still maintains high performance. In contrast, the throughput of FreeFlow is down to only 10% throughput compared to the peak. Because uniRDMA data

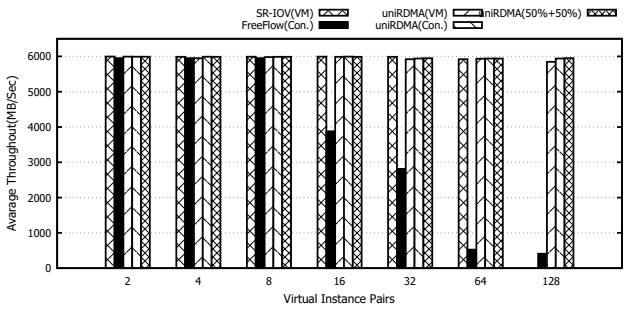

Figure 14: The Scabality

commands do not need to be forwarded to the software virtualization layer for processing, the virtualization overhead does not increase due to the expansion of virtual instances. Constantly, when FreeFlow runs RDMA commands at the same time in a large-scale container, it is forwarded to the same RDMA context in virtual layer. When calling the QP queue or registered memory region in virtual layer, there is the overhead of lock mutual exclusion. Therefore, FreeFlow suffers from a drastic drop in performance in large-scale container scenarios.

In addition, as shown in Figure 14, both uniRDMA and FreeFlow can support communication between 128 pairs of virtual instances. However, the maximum number of VFs of the Mellanox ConnectX-3 network card is only 126, so 128 pairs of virtual instances are not supported. This shows that uniRDMA has higher scalability than SR-IOV. Because VFs are statically allocated in SR-IOV and each VF is exclusively occupied by the virtual machine; while uniRDMA improves the utilization of VF through dynamic device pool and flexible mapping mechanism in virtual layer.

6.3 Real-world Applications

The worth of RDMA is mainly about its optimized performance in real-world applications. RDMA virtualization needs to maintain the performance close to native RDMA. Therefore, we evaluate uniRDMA and other frameworks in different RDMA applications, such as high-performance computing benchmark Graph-500 and data processing framework Spark-RDMA.

(1) Graph-500: For high-performance computing, Graph-500 is a benchmark framework used to test the performance of the Message Passing Interface (MPI). Based on the constructed graph structure, users test the performance of breadth-first search (BFS) and single source shortest path (SSSP). The

performance index is the number of edges traversed per second (traversed edges). per second, TEPS), the larger the value, the better the performance.

In this paper, the node scale of the computational graph in Graph-500 is set to 26, and the ratio of edges to points is set to the default parameter of 16. The constructed graph has a total of 2$\hat{2}$5 vertices, with 2$\hat{2}$9 edges, the entire graph occupies approximately around 16GB. When testing BFS and SSP, 16 MPI processes are scattered and executed on two nodes in turn, and the average value is taken according to the results of 12 tests. The data obtained is shown in Table Figure 15 (because there are core dump problems when using FreeFlow for Graph-500, the corresponding data is lacking).

As shown in Figure 15, the performance of uniRDMA is close to native RDMA. Because uniRDMA bypasses the kernel and virtualization layer in the data path, and there is no forwarding latency.

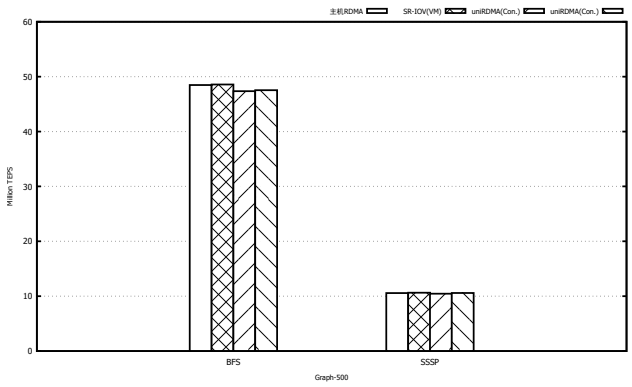

Figure 15: The Performance of Graph-500

(2) Spark-RDMA: It is a classical big data processing framework for distributed computing. Based on Spark-RDMA (v0.9.5), we run two tasks: GroupBy and SortBy. For each task, two Spark processes are run on two servers. At the same time, the number of cores used is limited to 8 and the memory is 32GB. There are 8 mappers and 8 reducers for 262144 key-value pairs of 2KB. At last, we run 12 times for each task to get the average execution time.

7 RELATED WORK

RDMA Virtualization: The solutions of RDMA virtualization including hardware-based and software-based.

The basic solution is SR-IOV in hardware-based virtualization of RDMA. Its virtual layer is in the hardware and are limited by hardware resources. In uniRDMA, the isolation of SR-IOV are utilized and the unscalable problems are solved by dynamic vRNIC mapping.

In software virtualization, existing solutions don't suit for hybrid virtual environments. FreeFlow's forward mechanism is different with uniRDMA. Its data path needs extra CPU to reduce the forward latency. HyV and MasQ are not used in containers, but are similar with us because all have achieved zero-copy and by-pass by mapping all resources. Even though, the implementation of mapping are different at essential for two reasons: First, HyV and MasQ's mapping is in the same process and uniRDMA is inter-process; Second, HyV and MasQ's mapping management in kernel-space and our in user-space for more manageability and lightweight.

8 CONCLUSION

In this paper, we design a unified RDMA virtualization framework, namely uniRDMA, which consists of single user-space virtual layer and general uniVerbs interfaces. In the user space virtual layer: the isolated and high-performance vRNICs are virtualized based on the VFs from SR-IOV; In the uniVerbs interfaces: uniRDMA uses shared files to realize general I/O channel for VMs and containers; the isolation of interface is ensured with mount namespace in containers; uniVebrs realizes zero-copy and bypassing virtual layer in RDMA data path, by mapping all RDMA resource between vRNIC and RDMA applications. The experimental results show that uniRDMA has high performance close to native RDMA while meeting generality and management.

REFERENCES

[1] 2018. Mellanox OFED for Linux User Manual. Retrieved March 13, 2021 from https://www.mellanox.com/related-docs/prod_software/Mellanox_OFED_Linux_User_Manual_v4_3.pdf
[2] 2020. Linux Mount Namespace. Retrieved March 13, 2021 from https://man7.org/linux/man-pages/man7/mount_namespaces.7.html
[3] 2021. Docker. Retrieved March 13, 2021 from https://www.docker.com/
[4] 2021. HPC in the Cloud? Yes, No and In Between. Retrieved March 13, 2021 from https://www.hpcwire.com/2020/12/14/hpc-in-the-cloud-yes-no-and-in-between/
[5] 2021. Linux KVM. Retrieved March 13, 2021 from https://www.linux-kvm.org/page/Main_Page
[6] 2021. Mellanox OpenFabrics Enterprise Distribution for Linux. Retrieved March 13, 2021 from https://www.mellanox.com/products/infiniband-drivers/linux/mlnx_ofed
[7] 2021. PCI-SIG SR-IOV. Retrieved March 13, 2021 from https://pcisig.com/specifications/iov/
[8] 2021. QEMU. Retrieved March 13, 2021 from https://www.qemu.org/
[9] Zhiqiang He, Dongyang Wang, Binzhang Fu, Kun Tan, Bei Hua, Zhi-Li Zhang, and Kai Zheng. 2020. MasQ: RDMA for Virtual Private Cloud. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. 1–14.
[10] Daehyeok Kim, Tianlong Yu, Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Shachar Raindel, Chuanxiong Guo, Vyas Sekar, and Srinivasan Seshan. 2019. FreeFlow: Software-based Virtual {RDMA} Networking for Containerized Clouds. In 16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19). 113–126.

[11] Bojie Li, Tianyi Cui, Zibo Wang, Wei Bai, and Lintao Zhang. 2019. SocksDirect: Datacenter sockets can be fast and compatible. In Proceedings of the ACM Special Interest Group on Data Communication. 90–103.

[12] Jonas Pfefferle, Patrick Stuedi, Animesh Trivedi, Bernard Metzler, Ionnis Koltsidas, and Thomas R Gross. 2015. A hybrid I/O virtualization framework for RDMA-capable network interfaces. ACM SIGPLAN Notices 50, 7 (2015), 17–30.

[13] Dongyang Wang, Binzhang Fu, Gang Lu, Kun Tan, and Bei Hua. 2019. vSocket: virtual socket interface for RDMA in public clouds. In Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. 179–192.