The Design and Implementation of uniRDMA Virtualization

Anonymous Author(s)*

ABSTRACT

Supercomputing is important in most data-driven fileds for its extrem high-performance, such as such as artificial intelligence and large-scale data processing. As the developement of cloud, supercomputing is also in virtual machines or containers for effiecncy and elasticity. Today, hybrid virtual enviroments is a trend in most datacenters because of combined advatanges and historic reasons. RDMA is neccesay in supercomputing for its high network performance. However, existing RDMA virtualization solutions lack centralized virtualization layer and general interfaces for hybrid virtual environments. To solve this problem, we present uniRDMA, a generic RDMA software virtualization framework that consists of single virtual layer and general uniVerbs interfaces. The centralized user space virtual layer constructs multiple isolated vRNICs(virtual RDMA NICs) with the help of hardware virtualization technology and realizes a unified virtual RDMA network. To realize both generality and high-performance, uniVerbs interfaces use shared memory as I/O approach for RDMA applications and map RDMA resources to vRNIC. In our evaluation based on uniRDMA prototype, the performance and scalability of uniRDMA is close to baremetal RDMA in virtual machines, containers and hybrid cloud environments.

KEYWORDS

RDMA, Virtulization, Hybrid Virtual Environment

ACM Reference Format:
Anonymous Author(s). 2021. The Design and Implementation of uniRDMA Virtualization. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Compared to tranditional computing, supercomputing has been widely adopted in different fields for high-performance, such as artificial intelligence, and large-scale data processing and engineering calculations. As the data dirves and the hardware develops, cloud computing is expanding everywhere for its high effiecncy and elasticity. Moreover, hybird virtual environments, which include both virtual machines and containers, is a trend in cloud due to combined advantages or historical reasons. As a result, lots of supercomputing has been in cloud, especially hybrid virtual environments, such as AWS HPC Cloud [3].

As a high-performance network, RDMA is basic in supercomputing for each data-intensive application. For examples, MPI [19], TensorFlow [14], Spark [11], and Hadoop [10] all have supported RDMA. With hardware protocol stack and zero copy technologies, RNICs(physical RDMA network cards) can bypass the kernel to read/write remote memory regions in applications, without the aware of remote CPU. Therefore, RDMA has high throughput, low latency and low CPU load.

To expolit RDMA's high-performance, RDMA virtualization is necassery. Like network or other virtulization, adding a virtual layer and providing virtual RDMA interface to upper VMs or containers is important works in RDMA virtualization. Howerver, RDMA's hardware protocal stack and direct-access user memory make it hard to use traditional network virtual layer, such as virtual switch. Even though this problem has been atempped to solve in lots of different solutions, neither of them suit for hybrid virtual environments due to lack of network performance or unified management.

Exsiting solutions mainly about hardware-based and software-based. The representative hardware virtualization is SR-IOV [8]. Its virtual layer is located in the hardware. Although the isolation and high performance are maintained, SR-IOV lacks portability and other manageability without software virtual layer. In software virtualization, existing solutions treat virtual machines and containers differently. For containers, FreeFlowl [17] forwards all RDMA commands to the virtual layer, and that is ineffective because of losing RDMA's kernel by-pass. For virtual machines, HyV [21] [20] avoids forwarding overhead by mapping RDMA resources, but lacks the management of virtual RDMA networks; although MasQ [16] makes up for this problem, its virtual layer is located in kernel space. Extending MasQ to the container environment will lose lightweight management in user space for containers.

We proposes an unified RDMA virtualization framework for both containers and virtual machines, namely uniRDMA, which achieves high performance and high manageability in hybrid virtual environment. UniRDMA is mainly composed of single centralized uniRDMA virtual layer and general uniVerbs interfaces. All managements are concentrated in the user space virtual layer; the UniVerbs interface is general to the RDMA applications in virtual machines or containers.

There are mainly two challenges in the design of uniRDMA: First, virtual machines and containers are essentially different virtualization technologies. Virtual machines are under the management of hypervisor in kernel space, containers are isolted runtime mainly about user space. It is a challenge to build a centralized virtual layer for both containers and virtual machines; second, mapping all RDMA resources is the key idea to achive high performance. However, the existing solutions only implement the mapping operation in the same process for virtual machine. In this paper, the virtual layer belongs to another host process. So, it is also another challenge to map all RDMA resources.

To address the first challenge, uniRDMA separates the control and use of RDMA resources in virtulization, builds multiple vRNICs in single unified user space virtualization layer. Each vRNIC encapsulates virtual RDMA resources, and isolates with others by hardware virtualization. Moreover, through a common file-based shared memory queue, each vRNIC has a general interface for virtual machines and containers. For the second challenge, at first, virtual RDMA resources in vRNICs are mapped to physical RNICs, and then shared memory are used to ensure RDMA resources in applications are mapped to vRNICs in the virtual layer.

We implements the prototype of uniRDMA and evaluate uniRDMA against with hardware virtualization, software virtualization FreeFlow, and native RDMA in different benchmarks, such as throughput, latency, scalability, and real-word applications. From the result, uniRDMA's performance is close to native RDMA in both virtual machine and container environments, and is significantly better than FreeFlow. The throughput can reach up to 6 times that of FreeFlow, and the latency can reach down to 40% of FreeFlow.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:

- Unified RDMA virtulization in hybrid virtual environments is firstly proposed in this paper and uniRDMA is general RDMA virtualization framework, while maintaining high performance and high manageability.
- uniRDMA are evaluated and the results proved that uniRDMA maintains high performance close to native RDMA.

2 BACKGROUND

This section introduces the background about RDMA virtualization, including hybrid virtual environment and the principles and working mechanisms of RDMA networks.

2.1 Hybrid Virtual Environment

Both containers and virtual machines have been a widley used in datacenters. Virtual machines have strong isolation and containers are lightweight and less-performance-loss. Before containers, virtual machines are the base of cloud computing. Todays, applications or microservices is both complicated and dynamic, not all applications are suit to deployed in containers instead of virtual machines. Therefore, hybrid virtual machine and container are common in datacenters. As a result, unified platforms for hybrid virtual environments is requirement for easier management. Nowadays, there are lots of platform catering to this trend. For exmaples, KubeVirt [5], VMWare vSphere [13] and RedHat OpenShift [12].

2.2 RDMA network

As a high-performance network, RDMA is currently widely used in supercomputing, artificial intelligence, and large-scale data processing. For examples, TensorFlow, Spark, and Hadoop all have supported RDMA. With the help of hardware protocol stack and zero copy technology, RDMA network card can bypass the kernel to read and write remote memory data, without the participation of remote CPU. Therefore, RDMA has high throughput, low latency and low CPU load.

Applications need to use Verbs interface when using RDMA. The RDMA communication is based on Queue Pair (QP). The application writes the RDMA work request to the QP, and then "press" the doorbell register in RNIC, and the RNIC's hardware processor will execute the work request in the QP to forwared data. For application, the entire operator is in the user space without the kernel.

The QP queue consists of a pair of Send Queue (SQ) and Receive Queue (RQ), which are respectively responsible for the sending and receiving work requests during RDMA communication. After the RDMA connection is established, RDMA support two communication modes:

- `Bilateral operation`: Two endpoints of RDMA connection execute send or recv respectively. For details, each writes the sending or receiving work request into the corresponding SQ or RQ queue. Similar to the traditional TCP network, the sender's RNIC will transmit data to the receiver.

- `Unilateral operation`: Only one endpoints of the RDMA connection needs to perform "send". The RDMA application writes the sending work requests to the SQ queue of QP, and RNIC read or write the remote memory according to the request without the participation of the remote CPU.

However, RDMA virtualization in hybrid virtual environments is still a problem. Beacuause uncenterlized RDMA virtual layer and respective interfaces for containers and virtual machines in exsiting solutions.

3 RELATED WORK

This section introduces related works about network virtualizations, including traditional network and RDMA.

3.1 Traditional Network Virtualization

NIC(Virtual network cards) and virtual network bridges is important to traditional network virtualization. When two virtual instances communicate across servers, the traffic of vNIC will be forwarded to the physical network card through the virtual network bridge, and then through the remote physical network card and virtual network bridge to reach the destination virtual instance.

vNICs are commonly implemented by software emulation. The solutions about that can be divided into full virtualization, paravirtualization and container virtual network card. They can all be configured with virtual IP addresses and send/receive data packets as physical network cards. Virtual network bridges connect virtual network cards and physical network cards by routing forwarding, tunnel networking, and other methods. In a word, virtual network is constructed through the unified management of vNICs and virtual network bridges.

3.2 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 located in the hardware, so it is high-performance close to native RDMA. However, SR-IOV utilzie the limited hardware resouces to virtualize multiple VFs for different VMs or containers and that makes SR-IOV unscabale. Finnally, SR-IOV' virtual interfaces are located in hardware, so it lacks portability and other manageability without software virtual layer.

In software virtualization, existing solutions treat virtual machines and containers differently. For containers, FreeFlowl [17] forwards all RDMA commands to the virtual layer, and that is ineffective because of losing RDMA's kernel by-pass. For virtual machines, HyV [21] [20] avoids forwarding overhead by mapping RDMA resources, but lacks the management of virtual RDMA networks; although MasQ [16] makes up for this problem, its virtual layer is located in kernel space. Extending MasQ to the container environment will lose lightweight management in user space for containers.

4 UNIFIED VIRTUAL LAYER

To realize unified RDMA virtualization for containers and virtual machines, we propose a software RDMA virtualization framework, namely uniRDMA. The vRNIC constructed by uniRDMA based on the user space virtualization layer provides unified virtual RDMA for virtual machines or containers.

As Figure 1 shows, uniRDMA is consist of two parts: uniRDMA user space virtual layer and uniVerbs interface. The former is responsible for the establishment and management of virtual RDMA. The specific work includes the virtualization of vRNIC, the mapping management of vRNIC to VF, and the virtual RDMA network management, such as virtual RDMA address configuration and routing; the latter is mainly aout how RDMA applications use vRNICs, including the construction of general interfaces and the optimization.

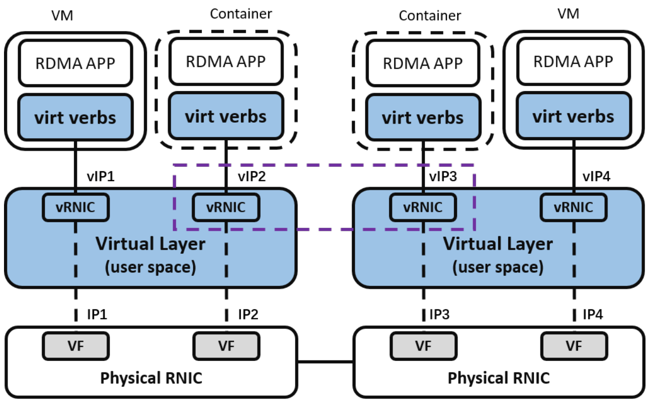

Figure 1: uniRDMA Framwork Overview

5 UNIFIED VIRTUAL LAYER

The first problem is which space the virtual layer is implemented: kernel space or user space. Because container applications and virtual machines are both host processes, they can interact with the host kernel or other host processes. In addition, there are both user-level and kernel-level RNIC interface. Therefore, both kernel space and user space can realize the unified RDMA virtual layer. However, user space layer is friendly to containers because of lightweight and in favour of secure and flexible management. Moreover, software development in user space is less difficult, more portable and compatible. Therefore, uniRDMA chooses a user space virtual layer.

In traditional network virtualization, vNICs are virtulized and bridged to physical NIC, all vNICs are isolated but configed and manageed uniformly as a virtual network. Similarly, the main work of uniRDMA virtual layer is including vRNIC virtualization, vRNIC mapping and virtual RDMA network management.The primary challenges are to make both isoaleted and high-performance for different vRNICs in the same virtual layer.

5.1 vRNIC Virtualization

The RDMA resources is the key in both control path and data path at 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.So, vRNIC virtulization is mainly about how to construct the virtual RDMA resoucrs to provide complete RDMA communication. 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 the RDMA NIC has the following buffers inside:

- `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 use the informaation to access local or remote memory for data transform.
- `Doorbell Buffer`: including multiple doorbell registers. The network card use 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 daynamic in control path and data path:

- `Control Path`: The creation and destruction of RDMA resources. RDMA resources or informations 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 informations in the destory. Nofity that this porcess has losts latency due to the kernel .
- `Data Path`: The usage of RDMA resources.RDMA resources or informations 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 include virtual doorbells. Virtual buffer is flexible and unlimited for the nums of RDMA resoucrs instances.

To emulate the dynamic attributes, apparently, if only using pure software, it may cause low performance because of the software overhead in whole RDMA communication. Fortunately, we found all RDMA reources informations in RNIC are only changed in control path and maintained in data path. So, we can map the virtual RDMA reources 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.

To realize this idea, we put each vRNIC with with a map unit. As Figure 2 shows, it maps or unmaps virtual RDMA resources from vRNIC to RNIC in control path:

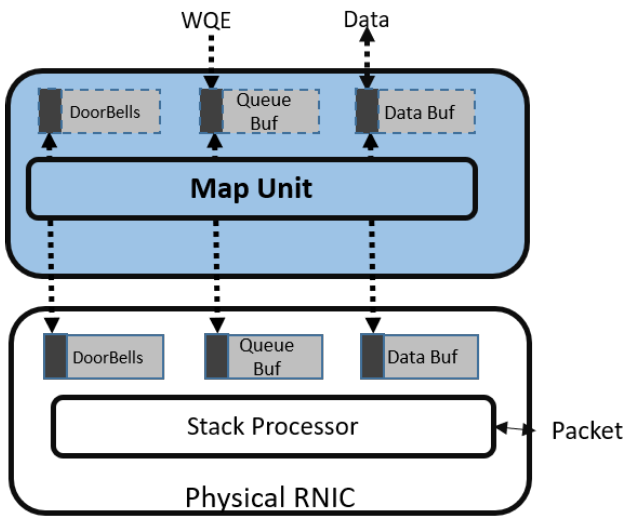

Figure 2: Map Unit in vRNIC

- `For QPs, CQs and MRs`: Taking QP as an example, regularly, vRNIC record virtual RDMA resources informations when the virtual QP instance are created. However, virtual QP are still not generated-associated with the RNIC. To make the maping, 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 informations are recongnized 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 memroy 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 has the ability to 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 are 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.

5.2 vRNIC Mapping

The same virtual layer needs to create multiple vRNICs and provide them to different containers or virtual machines respectively. If each vRNIC is directly mapped to the same RNIC, it will compete for the same PCIe bus and share the configuration space of RNIC. The vRNICs are still not isolated or limited.

SR-IOV is a popular hardware-based virtualization technology.RNIC can virtualize multiple different hardware interface, called VFs. Each VF has an unique PCIe bus and configuration space. At the same time, when configuring VFs, users can limit network rate and other hardware resources, and implement management policies such as QoS. The uniRDMA virtual layer maps each vRNIC to the VF interface of RNIC separately, so hardware-level isolation are guaranteed for vRNICs.

However, the 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.

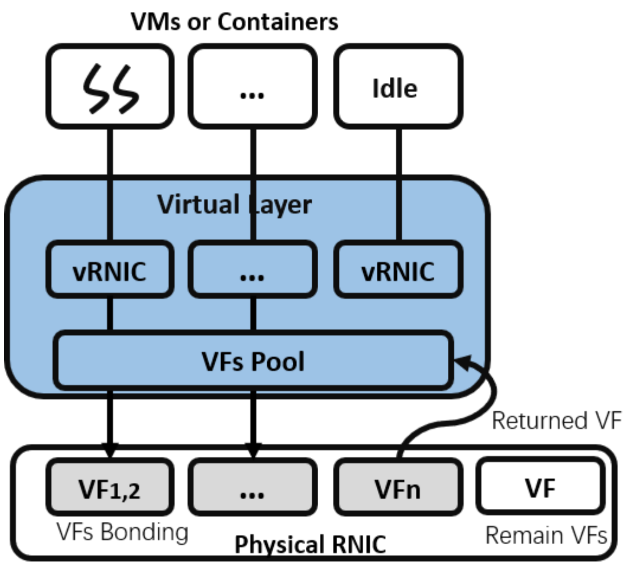

Figure 3: Management of vRNIC Mapping

As Figure 3 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 thatt 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 meeted by map a vRNIC with multipe VFs. VF resources are saved by mapping multiple vRNICs of the same virtual instance to the single VF.

5.3 Virtual RDMA Management

To maintain portability and realize RDMA network management, RDMA connections between vRNICs can not 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 4 , 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 can not succeed due to the isolated routing rules between group 1 and group 2.

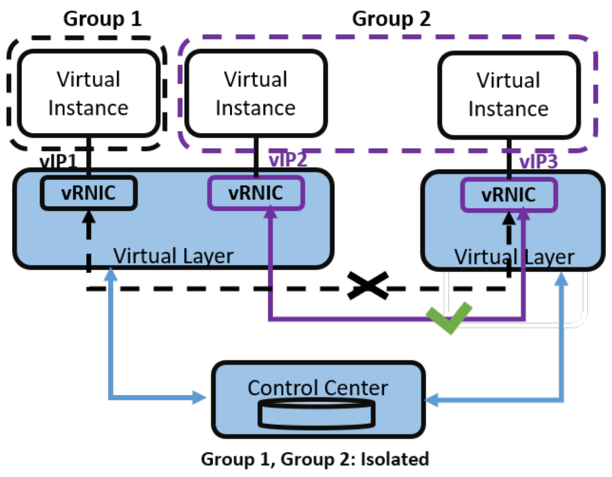

Figure 4: Virtual RDMA Network Routing

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 informations, 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.

6 GENERAL VRNIC INTERFACE

In native RDMA, the application accesses RNIC by the Verbs interface. Correspondingly, we presents uniVerbs interface, which is general to containers and virtual machines. Moreover, the interfaces have strong isolation and are transparent to all RDMA applications. At the same time, uniVerbs interface maps RDMA resources bewteen RDMA applications and vRNICs for zero copy and bypassing the virtual layer.

6.1 Basic uniVerbs Interface Construction

The RDMA applications in containers and virtual machines are both the processes of host. Therefore, the general interface are avialable for both containers and virtual machines.

For containers, vRNIC can be directly provided to RDMA applications in containers by IPC(inter-process communication). However, for virtual machines, the vRNIC in the host user space and the RDMA application in the virtual machine user space are separated from the emualted hardware environment and the guest operating system. Therefore, it is necessary to use I/O virtualization technology to extend each vRNIC as an I/O device of the virtual machine. Then, driver for this device must be installed in virtual machine to support the I/O process. Therefore, the uniVerbs interface for the virtual machine includes two following works:

- `Extend vRNIC as I/O device`: 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 complemently 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.

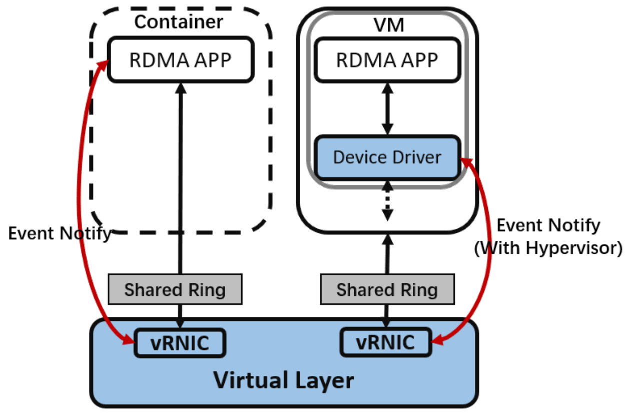

Figure 5: General uniVerbs Interface

- `Design the I/O device driver`: The goal of the device driver is to support I/O process inside each guest. As shown in in Figure 5, 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 5, 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 the file in each vRNIC interface are not isolated, they can be discovered by every container through scanning files. In order to solve this problem, we runs the virtual machine in a isolated container environment. As shown in in Figure 6, 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.

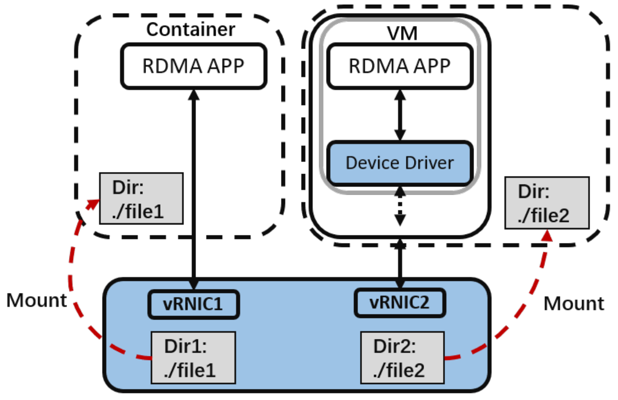

Figure 6: General uniVerbs Interface

6.2 uniVerbs Interface Optimization

After completing the interface construction, the requests of RDMA applications can be transmitted to vRNICs, and the results can also be returned to the applications. In uniVerbs, zero copy of data and bypassing the virtual layer are realized by mapping the RDMA resources to vRNIC. As a result, data commands do not need to be forward to the virtual layer and this mode is consistent with native RDMA.

(1) zero copy optimization The zero copy contents are including the RDMA work request in the QP queue and the transfer data in the registered memory, and the process is from RDMA application to pyhiscal RNIC. Remind that vRNIC RDMA resources are mapped to VFs, so we only need to make sure the zero-copy from RDMA application to vRNIC.

The fact of zero-copy is that both processes have common availiable physical memory pages. Similar as the above I/O channels, file-based shared memory are also used when mapping QP and other RDMA resouces and this is general for both virtual machines and containers.

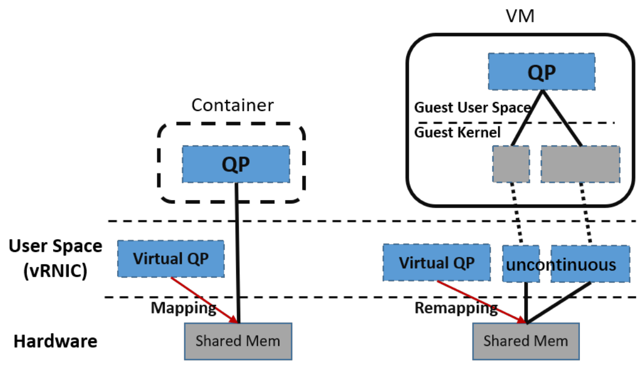

Figure 7: Mapping QP to vRNIC

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 7. So, vRNIC can not 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 virtual layer optimization Pressing the doorbell is necessary in RDMA data path to drive RNIC. In vRNIC, the doorbell that is mapped to VF, still need to be mapped to RDMA application tp meet bypassing. Otherwise, the pressing command needs be forward to vRNIC and thats imports apperant 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 located 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.

Therefore, when an RDMA application creates a RDMA context, as shown in Figure 8, it sends a request to the vRNIC at first. Under the supervision of virtual layer, vRNIC

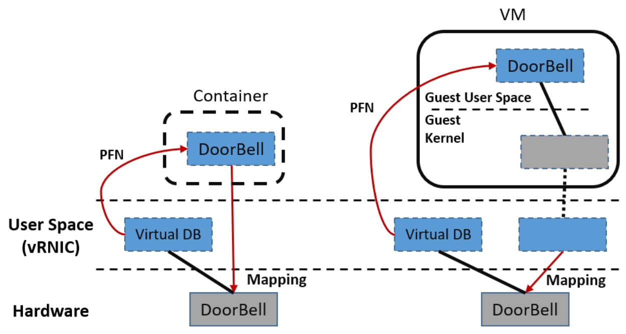

Figure 8: Mapping DB from vRNIC

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.

7 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-wprd 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?

7.1 Experiment Methodology

All experiments are carried out on two servers. The settings mainly include three parts: host server, container and virtual machine. The detailed settings are shown in Table 1:

The RNIC used by the server is Mellanox ConnectX-3 56 Gb/sec, which performs RDMA communication under Infiniband. The operating system of servers is CentOS 7.4.1708, and the Linux kernel version is 3.10.0-693.el7.x86_64. The RDMA driver installed on the host server is Mellanox OFED 4.4-2.0.7.0 [7], which adapts to the RNIC and host operating system. 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) [9] enabled with KVM [6] and the container engines are Docker(18.06.1-ce) [4]. In addition, the entire uniRDMA framework is compiled with GCC/G++ 4.8.5, and the O3 compilation optimization level is selected.

Table 1: Experiment Environments

Parameters		Settings
Server	CPU	Four Intel Xeon E7-4850 v4 16-core CPUs
	Memory	1 TiB DDR4/DDR3
	Kernel	CentOS 7.4.1708
	Physical RNIC	Mellanox ConnectX-3 56 Gb/s
	RDMA Driver	Mellanox OFED 4.4-2.0.7.0
	Hypervisor	QEMU 5.1.50
	Container Engine	Docker 18.06.1-ce
VM	CPU	16 cores
	Memory	64 GB
	Kernel	CentOS 7.4.1708
Container	CPU	16 cores
	Memory	64 GB
	Image	CentOS 7.4.1708

7.2 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 thems, 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 transmited 1000 times with each message size, and finally the average throughput and latency are calucated.

(1)Throughput: The results of bilateral operation are shown in Figure 9, and the one of unilateral operation are shown in Figure 10. Whether uniRDMA is in a virtual machine or in a container scenario, the throughput of its bilateral and unilateral operations is slimilar 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.

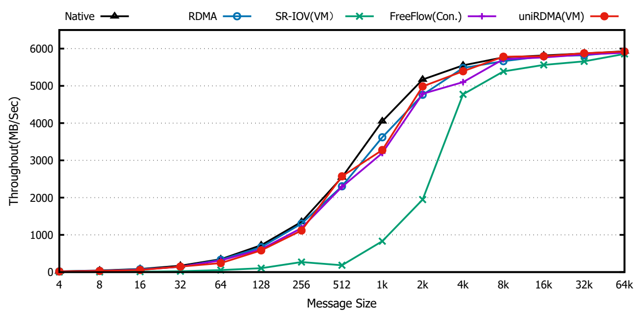

Figure 9: The Throughout of RDMA Send and Recv

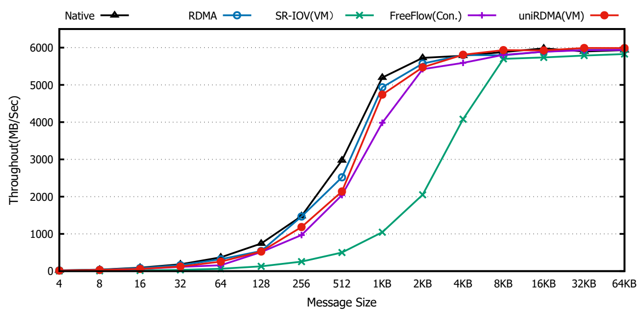

Figure 10: The Throughout of RDMA Write

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 11, and the one of unilateral operation are shown in Figure 12. Whether uniRDMA is in a virtual machine or in a container scenario, the latency of its bilateral and unilateral operations is slimilar 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 beacause of FreeFlow's forwarding latency. Also, when the message gradually increases, such as reaching 64KB, the latency of each framework tends to be consistent. Beacause the main latency has been caused by RNIC data processing.

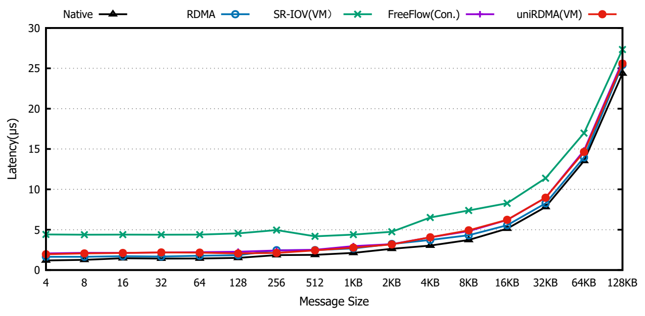

Figure 11: The Latency of RDMA Send and Recv

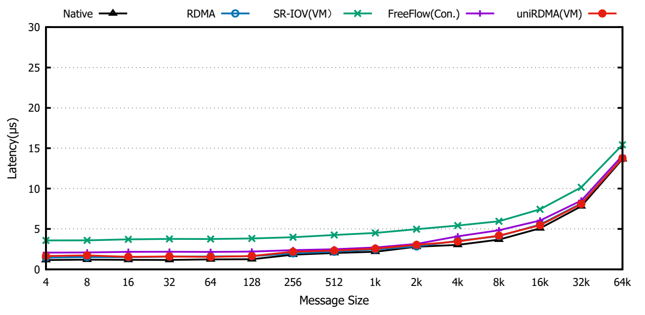

Figure 12: The Latenct of RDMA Write

(3)Scabality: Scalability is a challenge when RDMA virtualization in large-scale virtual clusters. To evaluate uniRDMA's scalabilitu, 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 13.

From Figure 13, 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 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. Constractly, whhen 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

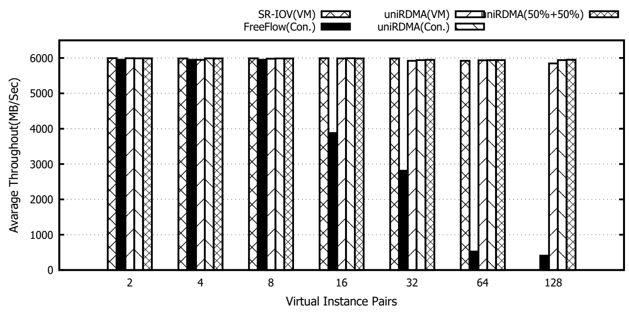

Figure 13: The Scabality

suffers from a drastic drop in performance in large-scale container scenarios.

In addition, as shown in Figure 13, 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.

7.3 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) [42]. 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 14 (because there are core dump problems when using FreeFlow for Graph-500, the corresponding data is lacking).

As shown in Figure 14, the performance of uniRDMA is close to native RDMA. Beacause uniRDMA bypasses the kernel and virtualization layer in the data path, and there is no forwarding latency.

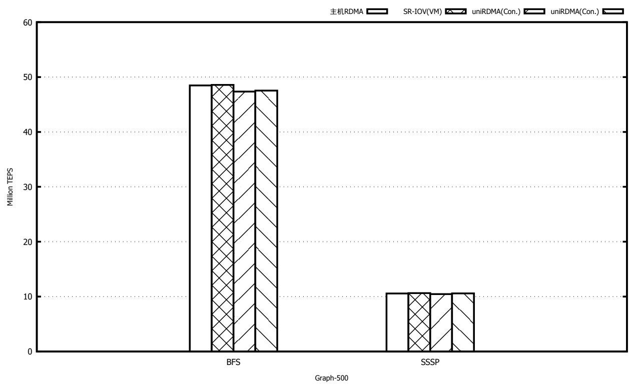

Figure 14: The Performance of Graph-500

8 DISCUSSION

In this section, there are some concerns about uniRDMA in cloud environments:

- `Security`: uniRDMA's virtual layer is in the user space. Unlike HyV and MasQ, RDMA resources, including data, do not need to be mapped to the kernel space. Therefore, it avoids buffer overflow and other attacks against the kernel. Through the isolated vRNIC and interface, there are no potential threats between different virtual instances.
- `Other network extensions`: The high performance of RDMA can integrate other network protocol stacks, such as TCP/IP networks, to optimize the performance of network applications. Existing work includes vSocket and so on [22] [18]. However, there is currently no unified RDMA-based optimized socket for container and virtual machine applications, and uniRDMA can be also easily extended to meet it.
- `Virtual Instances Migration`: For virtual instances containing RDMA applications, uniRDMA can directly migrate statically without reconfiguring the RDMA address, etc., only for dynamic migration in the virtual, because RDMA bypasses the transmission mechanism of the kernel and remote read and write, its memory capacity It is difficult to perceive management and monitoring, and uniRDMA needs to cooperate with other research work to achieve this goal [15].

9 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. AWS HPC Cloud. Retrieved March 13, 2021 from https://aws.amazon.com/cn/hpc/
[4] 2021. Docker. Retrieved March 13, 2021 from https://www.docker.com/
[5] 2021. KubeVirt. Retrieved March 13, 2021 from https://kubevirt.io/
[6] 2021. Linux KVM. Retrieved March 13, 2021 from https://www.linux-kvm.org/page/Main_Page
[7] 2021. Mellanox OpenFabrics Enterprise Distribution for Linux. Retrieved March 13, 2021 from https://www.mellanox.com/products/infiniband-drivers/linux/mlnx_ofed
[8] 2021. PCI-SIG SR-IOV. Retrieved March 13, 2021 from https://pcisig.com/specifications/iov/
[9] 2021. QEMU. Retrieved March 13, 2021 from https://www.qemu.org/
[10] 2021. RDMA-based Apache Hadoop. Retrieved March 13, 2021 from http://hibd.cse.ohio-state.edu/#hadoop
[11] 2021. RDMA-based Apache Spark. Retrieved March 13, 2021 from http://hibd.cse.ohio-state.edu/#spark
[12] 2021. RedHat OpenShift. Retrieved March 13, 2021 from https://www.openshift.com/
[13] 2021. VMWare vSphere. Retrieved March 13, 2021 from https://www.vmware.com/products/vsphere.html
[14] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16). 265–283.
[15] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure accelerated networking: Smartnics in the public cloud. In 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). 51–66.
[16] 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.
[17] 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.
[18] 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.
[19] Jiuxing Liu, Jiesheng Wu, and Dhabaleswar K Panda. 2004. High performance RDMA-based MPI implementation over InfiniBand. International Journal of Parallel Programming 32, 3 (2004), 167–198.
[20] Jonas Pfefferle. 2014. vVerbs: a paravirtual subsystem for RDMA-capable network inter
Master's thesis. ETH-Zürich.
[21] 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.
[22] 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.