UniRDMA Design and Implementation

## 摘要

RDMA是一种高吞吐量、低延迟和低CPU负载的高性能网络，广泛应用于人工智能、大数据处理和高性能计算等领域。近年来，虚拟机和容器的统一管理成为云计算的趋势。在混合的虚拟场景下，已有RDMA虚拟化工作仍不具备统一管理能力，因为缺乏集中的虚拟层和通用的接口设计。为了解决这一问题，本文设计了一个通用的RDMA软件虚拟化框架uniRDMA。该框架的统一的用户空间虚拟层借助硬件虚拟化技术构建多个隔离的虚拟RDMA网卡vRNIC，实现了统一的虚拟RDMA网络；通用的uniVerbs接口负责虚拟机或容器内RDMA应用与vRNIC的I/O过程，其中基于共享内存的I/O方式同时适应于虚拟机和容器，并通过映射RDMA资源维持了高性能。根据uniRDMA原型的测试结果，uniRDMA在虚拟机和容器及其混合场景下，性能表现与硬件虚拟化技术接近，且具备高可扩展性，并适应于真实的RDMA应用场景。

## 1，引言

RDMA，即远程直接访问内存，是一种新型的高性能网络技术。RDMA目前广泛应用于人工智能、大数据处理和高性能计算等领域，如TensorFlow、Spark、和Hadoop[21]等框架均具有对应的RDMA版本。通过硬件协议栈和零拷贝等技术，RDMA网卡可以根据应用的工作请求，绕过内核读写远端的内存数据，而无需远端CPU的参与，因此，RDMA具备了高吞吐、低延迟和低CPU负载的网络性能。

云计算的发展离不开底层虚拟化技术的进步。主流的虚拟化技术有虚拟机或容器[2]，容器技术隔离轻量级的运行时环境，无需设备模拟，性能损耗小。虚拟机具有强隔离性，更加安全，但性能损耗大。虚拟机和容器各自的优势使得它们在如今的云环境中均得到了广泛应用，它们统一部署管理、混合使用的趋势也越来越明显。例如，VMware的虚拟化平台vSphere[22]和ReadHat的容器云平台 OpenShift[23]，在最新的版本迭代中，均明确支持对虚拟机和容器的统一部署与管理。

云端应用需要借助RDMA虚拟化技术，才能够利用RDMA网络的高性能。RDMA在云环境中的虚拟化，不仅需要维持高性能和可管理性，还需要具备通用性，以适应虚拟机和容器混合部署统一管理的趋势。因此，本文的RDMA虚拟化目标如下：

* 通用性：RDMA虚拟层中具有集中的虚拟层，并借助通用的接口提供给虚拟机和容器，形成统一的RDMA虚拟化。
* 高性能：虚拟RDMA在吞吐量、延迟以及CPU负载上应接近原生特点，同时满足大规模的虚拟实例场景。
* 高可管理性：RDMA虚拟化中，应当维持已有的容器和虚拟机特点，满足云环境中的可移植、网络隔离和管理等需求。

RDMA具有不同的硬件特性和工作机制，因此，RDMA虚拟化与传统网络虚拟化存在差异。如表，目前的RDMA虚拟化工作主要有硬件虚拟化和软件虚拟化。但是，已有的RDMA虚拟化方案均无法满足上述要求。

硬件虚拟化的代表工作是SR-IOV，其虚拟层位于硬件，虽然保持了网络的隔离性，具备与原生一致的高性能。但是SR-IOV在软件层面能定制的管理逻辑极为有限，缺乏可管理性，例如可移植性。软件虚拟化中，已有工作均是将虚拟机和容器区别对待，例如，容器场景下的FreeFlow，将RDMA所有命令转发给软件虚拟层处理，丧失了原生RDMA绕过内核的特点，在性能上不足；虚拟机场景下的HyV，通过映射RDMA资源避免了FreeFlow的转发操作，实现了高性能，但是缺乏对RDMA网络的管理；MasQ虽然弥补了这一不足，但是，其虚拟层位于内核空间，移植到容器场景下会导致容器丧失用户空间管理编排的轻量级特点。

本文提出了一种面向容器和虚拟机的通用RDMA虚拟化框架uniRDMA，在不损失容器轻量级和虚拟机隔离性的前提下，实现了高性能和高可管理性性。UniRDMA主要由集中的uniRDMA虚拟层和通用的uniVerbs接口两部分组成。虚拟层的管理功能均集中在用户空间实现，能灵活和安全地实现容器和虚拟机RDMA的统一管理；UniVerbs接口对虚拟机和容器的RDMA应用具有通用性，同时通过资源映射实现了RDMA的高性能。

在uniRDMA的设计过程中，主要面临以下两方面的挑战：其一，通用性挑战；虚拟机和容器本质上是不同的虚拟化技术，但是虚拟机不是普通的进程，其与主机其他进程的交互需要虚拟机监视器参与，用户空间虚拟层如何与虚拟机和容器交互；其二，高性能挑战；在已经具备通用性和高可管理性的前提下，保持RDMA的高性能。高性能的关键在于RDMA资源的映射，但是目前，已有的RDMA虚拟化工作中仅在虚拟机场景下实现了RDMA映射操作。本文虚拟层在用户空间，与虚拟机和容器区别属于另一主机进程，如何完成映射资源操作以实现高性能RDMA目标是一大挑战。

为了解决通用性方面的挑战，uniRDMA将RDMA虚拟化中的构建和使用分开，在统一的用户空间虚拟层构建虚拟的vRNIC，封装RDMA服务，并借助硬件虚拟化进行隔离。vRNIC对于虚拟机和容器来说并无区别，进一步，通过通用的基于文件的内存共享队列，充当虚拟机和容器与vRNIC之间的通用接口；为了解决性能问题，本文利用共享内存等机制，将RDMA资源进行映射，首先，在vRNIC的设计中，将虚拟的RDMA资源映射到了物理网卡中，然后通过共享内存等机制确保了RDMA应用中的RDMA资源映射到虚拟层的vRNIC，从而实现了数据的零拷贝和绕过虚拟层。

本文按设计实现了uniRDMA框架的原型，并从吞吐量、延迟、可扩展性以及真实应用效果等多个维度测试了uniRDMA、硬件虚拟化、软件虚拟化FreeFlow以及原生RDMA的运行效果。从测试数据来看，uniRDMA在虚拟机和容器场景下均实现了与原生RDMA接近的高性能，整体上性能表现与硬件虚拟化相当且明显优于FreeFlow，其中吞吐量最高可以达到FreeFlow的6倍，同时延迟最低仅为FreeFlow的40%；uniRDMA在各种场景下均具有高可扩展性，仍然保持了与原生RDMA一致的性能；uniRDMA在各种场景下均适应真实的RDMA应用，其中，uniRDMA的性能和硬件虚拟化技术之间的性能差异低于10%。

本文的主要贡献有：

* 基于RDMA虚拟化目标，设计了通用的RDMA虚拟化框架uniRDMA，同时维持了高性能与高可管理性，并实现了框架的原型。
* 测试了uniRDMA框架在网络基准性能、可扩展性和真实RDMA应用场景中的性能效果，表明uniRDMA在满足通用和高可管理性的同时，维持了与原生RDMA接近的网络性能。

## 2， 背景

本节介绍了与RDMA虚拟化相关的背景知识，包括了传统网络虚拟化以及RDMA网络的原理和工作机制。

### 2.1 传统网络虚拟化

在传统网络虚拟化过程中，需要借助虚拟网卡和虚拟网桥，才能实现各虚拟实例之间的网络连接和数据传输。在两个虚拟实例跨节点通信时，虚拟网卡的流量会经虚拟网桥转发到物理网卡，再经过远端物理网卡、虚拟网桥，进入目的虚拟实例。

在虚拟网卡的主要实现方式是软件模拟。根据虚拟网卡的实现方式以及适应的场景，可以分为全虚拟化、半虚拟化和容器虚拟网卡。它们均可以配置对应的虚拟IP地址，能模拟网卡收发数据包的基本功能。传统网络虚拟化，还需要借助虚拟网桥，虚拟网桥通过路由转发、隧道网络等方式来连接虚拟网卡和物理网卡。通过对虚拟网卡和虚拟网桥的统一管理，构建了完整的虚拟网络。

### 2.2 RDMA 网络

RDMA目前广泛应用于人工智能、大数据处理和高性能计算等领域，如TensorFlow、Spark、和Hadoop[21]等框架均具有对应的RDMA版本。通过硬件协议栈和零拷贝等技术，RDMA网卡可以根据应用的工作请求，绕过内核读写远端的内存数据，而无需远端CPU的参与，因此，RDMA具备了高吞吐、低延迟和低CPU负载的网络性能。

应用程序为了利用RDMA网络的高性能，需要借助Verbs接口访问和控制RDMA网卡。RDMA网卡和Verbs接口向应用提供了基于队列对（Queue Pair，QP）的通信方式。应用往QP中写入RDMA工作请求，然后对网卡的门铃寄存器执行写操作，通知网卡处理器执行QP中的工作请求，进行数据的传输。整个过程均可以在用户空间调用Verbs接口实现，无需经过内核。

QP队列由一对发送队列（Send Queue，SQ）和接收队列（Receive Queue，RQ），分别服务于RDMA数据传输时的发送工作请求和接收工作请求。在RDMA连接建立后，RDMA网卡和Verbs接口支持双边操作和单边操作的数据传输：

* **双边操作：**RDMA连接两端分别执行发送工作请求和接收工作请求，即各自将发送或接收工作请求写入对应的SQ或RQ队列中。和传统的TCP网络类似，发送方的RDMA网卡会将数据传输到接收方，此时，需要两端CPU的共同参与。
* **单边操作：**只需RDMA连接中的一端执行发送工作请求。RDMA应用往QP中的SQ队列中写入发送工作请求，RDMA网卡将在远端CPU无需参与的情况下，按请求对远端内存进行读写。

## 3, 整体设计

为了实现针对容器和虚拟机的通用的RDMA虚拟化，同时兼顾RDMA虚拟化性能和管理方面的目标，本文设计了uniRDMA的软件虚拟化框架。uniRDMA基于用户空间虚拟层构建的vRNIC为不同虚拟机或容器实例提供统一的虚拟RDMA服务。

如图4-1所示，uniRDMA总体上分为两个部分：uniRDMA用户空间虚拟层和uniVerbs接口。前者负责虚拟RDMA服务的建立和管理，具体工作包括vRNIC的构建、vRNIC到硬件虚拟化接口VF的映射管理以及vRNIC的虚拟RDMA网络管理，例如分组和路由建立；后者负责虚拟层vRNIC与容器或虚拟机RDMA应用的交互，具体工作包括通用性接口的构建和接口的性能优化工作。

## 4. 统一的虚拟层

构建RDMA虚拟层的首要问题是虚拟层在哪个层次实现。在软件虚拟化中，虚拟层主要有两个候选的实现层次：内核空间和用户空间。从可行性上分析，容器应用以及虚拟机都是主机操作系统管理的进程，它们均可与主机的内核或其他进程进行交互。此外，在内核和用户空间均有RDMA物理网卡的管理和使用接口，因此，内核空间和用户空间均可实现对虚拟机和容器统一的RDMA虚拟层。

从虚拟层实现效果来看，在用户空间实现维持了容器实例的用户空间轻量级特点，提供了更加安全和灵活的虚拟层管理；此外，用户空间实现的软件开发难度更低、软件可移植性和兼容性更强。因此，uniRDMA选择在用户空间构建统一的虚拟层。

和传统网络虚拟化一致，uniRDMA虚拟层的目的是构建vRNIC并连接物理RDMA网卡，并基于vRNIC构建虚拟的RDMA网络。因此，uniRDMA虚拟层的主要工作是vRNIC虚拟化，将vRNIC映射到物理网卡并管理虚拟RDMA网络。稍后，容器和虚拟机可以直接使用vRNIC获取RDMA服务，彼此之间互不影响。

### 4.1 vRNIC虚拟化

RDMA通信的控制路径和数据路径是分离的。在控制路径中，应用在主机内存中创建QP等队列实例，以及注册数据块等；在数据路径中，应用直接通过写门铃的方式，通知RDMA网卡按QP队列中的工作请求完成数据的收发。从设备的视角出发，RDMA物理网卡具备了静态属性和动态属性这两类硬件属性。

在静态属性上，由于RDMA基于QP等队列收发消息，因此，可以抽象地认为RDMA网卡内部具有下述缓冲区：

* **队列缓冲区：**存放QP，CQ等队列实例的信息，如QPN，CQN。网卡使用该信息读写工作请求，与远端QP建立连接等。
* **数据缓冲区：**存放注册内存块的信息，如页表，key等。网卡根据该信息读取内存数据块，授权远端访问等。
* **门铃缓冲区：**包含多个门铃寄存器组成。网卡用以接受用户命令并通知内部硬件处理器执行DMA、处理和转发等工作。

在动态属性上，RDMA网络以各种RDMA资源为核心，因此，动态属性主要和控制路径和数据路径中的RDMA资源状态相关：

* **控制路径：** 缓冲区中RDMA资源的建立和销毁。例如，在资源创建阶段，物理网卡记录创建的QP，CQ等队列实例和注册数据块信息，将物理网卡门铃实例映射给用户空间的RDMA应用。
* **数据路径：**RDMA资源的使用过程。当RDMA应用直接使用RDMA资源，由RNIC硬件处理器进行DMA、封装和转发数据等，以减少CPU负载。

针对上述静态属性的虚拟化，本文设置了一一对应的虚拟缓冲区，即虚拟的队列、数据和门铃缓冲区，分别存放虚拟的QP或注册内存块信息以及对应的虚拟门铃实例。

针对上述动态属性的虚拟化，本文在vRNIC中设计了一个映射单元功能模块，以确保vRNIC能够与RDMA物理网卡相关联，利用物理网卡硬件的高速处理能力，如DMA零拷贝和硬件协议栈处理等硬件能力。

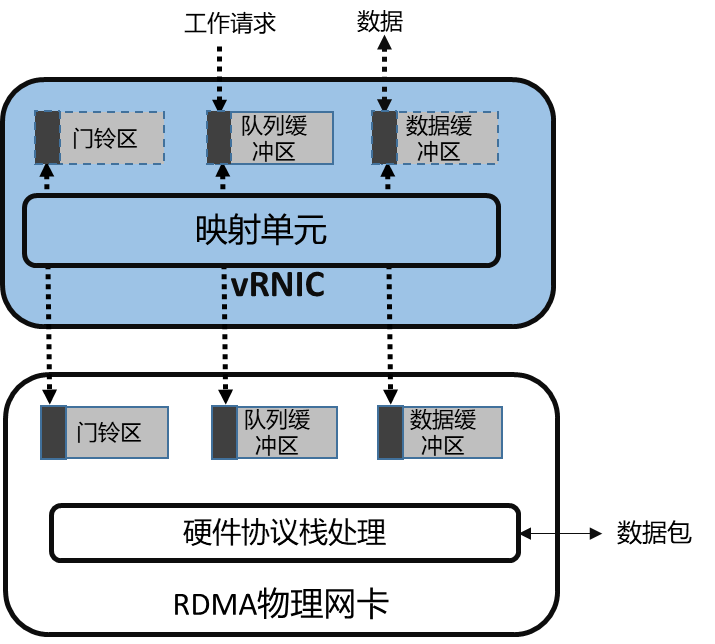


图 4‑2 虚拟层vRNIC的映射单元设计

对于虚拟的QP或注册内存块等虚拟资源实例，所有虚拟资源实例仍未与物理网卡产生-关联。映射单元会根据虚拟资源实例的信息，在vRNIC中基于物理网卡创建对应的QP等队列实例或注册对应的数据块，其中QP等队列实例和注册内存块使用与虚拟资源实例一致的内存地址信息，具体实现可以通过调用主机的Verbs库接口完成。因此，此时虚拟资源实例与物理资源实例建立了一对一的映射关系。如图4-2所示，完成映射后，vRNIC虚拟QP中的工作请求和注册内存块中的内容均能零拷贝到RDMA网卡中。

对于虚拟的门铃实例，需要映射到物理网卡设备空间的硬件门铃，才能让vRNIC具备通知物理RDMA网卡的能力。在vRNIC中，映射单元会通过系统调用将虚拟门铃的虚拟地址映射到对应物理网卡设备空间的硬件门铃地址。如图4-2所示，完成映射后，对vRNIC虚拟门铃的写操作就等价于执行物理网卡的门铃通知。

上述虚拟资源实例的映射过程，均由vRNIC中的映射单元完成。其映射关系均是一对一的，因此，保证了虚拟资源实例功能的正确性和资源的隔离性。同时，因为映射操作只在虚拟资源创建时执行，开销仅存在于控制路径。对于数据路径，由于虚拟资源实例已经和RDMA物理资源实例完成了内存层面的映射，因此，vRNIC可以直接利用RDMA物理网卡的硬件处理能力，如DMA零拷贝和硬件协议栈处理等，具备了RDMA物理网卡的高性能特点。

### vRNIC映射管理

同一虚拟层需要面向多个容器或虚拟机，因此，虚拟层需要创建多个vRNIC设备，分别提供给不同容器或虚拟机。如果各个vRNIC直接映射到同一物理RDMA网卡，将竞争同一PCIe总线，共同使用RDMA物理网卡的配置空间，此时，vRNIC无法得到隔离或限制。

SR-IOV是一种流行的基于硬件的虚拟化技术。RDMA物理网卡可以虚拟出多个不同的硬件接口VF，各自具备不同的PCIe总线和配置空间。同时，在设置VF时，用户可以对其占用的硬件资源、网络速率等进行限制，实现资源分配和QoS等管理策略。因此，uniRDMA虚拟层在将各vRNIC分别映射到物理网卡的VF接口，从而保证了各个vRNIC在硬件资源层面的隔离性。

然而，SR-IOV的VF接口资源是有限的，例如，在Mellanox ConnectX-3中，最多仅支持126个VF接口。因此，将vRNIC与对应的VF接口关联时，需要对已有的VF资源进行统一管理和协调，以满足多个vRNIC对VF接口的需求。

首先，虚拟层会构建一个动态的VF设备池。设备池中VF的初始数量通常为预判的虚拟实例的数量。如果有新的虚拟实例启动，此时设备池中缺少空闲VF，设备池可以动态扩充VF的数量。其次，虚拟层支持vRNIC与VF接口之间的动态映射。当虚拟实例的RDMA应用中各虚拟资源实例已经销毁后，虚拟层便将该vRNIC映射的VF接口标记为空闲状态，重新放入到VF设备池中。通过动态映射机制，虚拟层可以支持超过VF限制的vRNIC数量。最后，虚拟层支持vRNIC与VF接口的多种映射关系。例如，通过与多个VF接口关联，可以有效提高网络带宽负载，实现冗余备份和负载均衡等。将同一虚拟实例的多个vRNIC映射到同一个VF接口，节省了VF接口资源，也维持了与外部虚拟实例之间的隔离性。

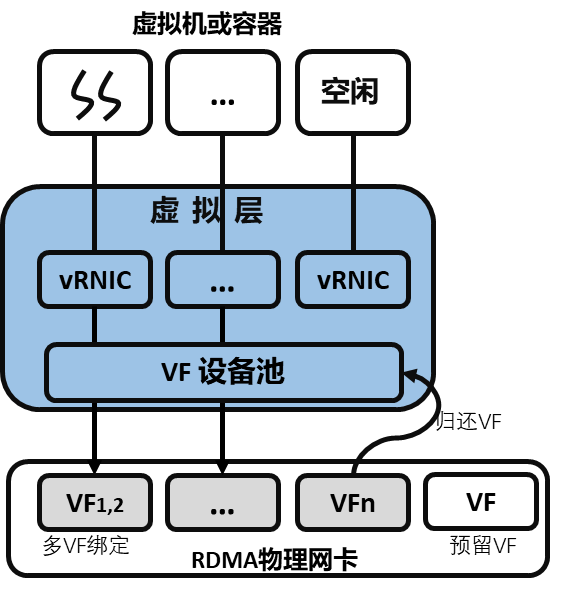


图 4‑3 vRNIC到VF接口的映射管理

### 虚拟RDMA管理

在完成vRNIC设备的虚拟化和映射后，此时，各vRNIC之间如果直接使用VF的物理地址建立RDMA连接，损失了云环境中的可移植性，也浪费了用户空间软件虚拟层的管理潜力。uniRDMA虚拟层充当软件的RDMA交换机或路由器，对基于vRNIC的虚拟RDMA网络进行配置和路由管理等。

RDMA物理网卡通常由集群中的子网管理器进行统一的网络管理，为此，在uniRDMA中设置了一个控制中心，对各个vRNIC统一分配虚拟的RDMA地址vGID，同时支持分组和多种路由规则的建立。如图4-4所示，对vRNIC进行分组划分，得到组1和组2；同组的vRNIC允许建立连接，组2的两个虚拟实例允许使用vRNIC建立连接，但是，组1和组2之间由于设置了隔离的路由规则，各自的vRNIC之间不能建立RDMA连接。

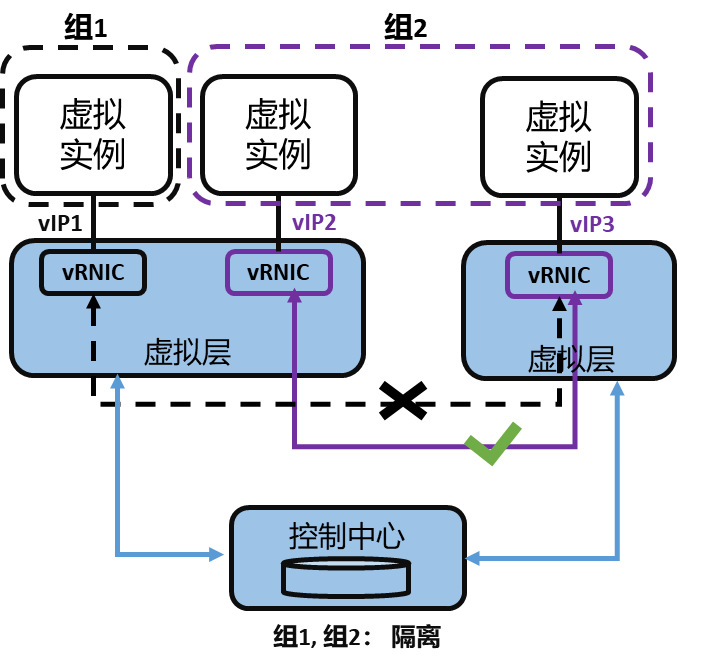


图 4‑4 虚拟RDMA网络分组和路由管理

各vRNIC可以按虚拟RDMA地址和路由规则建立连接。与原生RDMA一致，各虚拟层的vRNIC之间在建立虚拟RDMA连接前，需要交换彼此的虚拟RDMA地址、虚拟QP队列信息、注册的数据块密钥等信息。但是， vRNIC配置的RDMA地址是虚拟的，物理RDMA网卡并不能识别该地址。因此，各虚拟层之间仍需要交换vRNIC虚拟地址与VF接口物理地址的对应关系。在建立vRNIC之间的虚拟RDMA连接时，将虚拟RDMA地址转换为映射VF接口的物理地址。QP队列等RDMA资源信息等，已通过vRNIC中的映射单元与VF接口完成了映射，均能被物理网卡VF接口所能识别，因此建立RDMA连接时直接使用即可。

## 5. 通用的uniVerbs接口

在原生RDMA中，应用通过Verbs接口访问物理网卡，执行RDMA的控制命令和数据命令。对应地，本文设计了基于vRNIC的uniVerbs接口，其中，接口对容器和虚拟机具有通用性，不同虚拟实例之间接口具有强隔离性，各接口对RDMA应用具有透明性。同时，为了实现高性能，本文通过映射机制对接口进行了优化，容器和虚拟机RDMA应用在传输数据时可以绕过虚拟层。

### uniVerbs接口构建

容器的RDMA应用是主机的进程，虚拟机RDMA应用也可视为主机虚拟机进程内的一部分，因此，虚拟机和容器的RDMA应用均可以与用户空间的vRNIC之间进行交互，通用的uniVerbs接口是可行的。

对于容器，vRNIC能以进程间通信的方式直接提供给容器RDMA应用。但对于虚拟机，主机用户空间的vRNIC和虚拟机用户空间的RDMA应用之间隔着模拟硬件环境与虚拟机操作系统。因此，需要借助I/O虚拟化技术，将vRNIC进一步模拟为虚拟机的I/O设备。虚拟机RDMA应用对vRNIC的访问控制将通过该设备I/O过程完成，此时虚拟机操作系统还需安装对应的设备驱动。因此，虚拟场景下uniVerbs的接口设计包含了两部分工作：

* **扩展vRNIC为虚拟机的I/O设备：**

现有的I/O虚拟化技术主要分为全虚拟化和半虚拟化。其中，全虚拟化技术完全模拟设备所有功能，每次I/O操作都将陷入到虚拟机监视器，存在频繁上下文切换和数据拷贝开销，性能较差；半虚拟化技术通过简化设计，并不完全模拟设备，通过优化减少了数据拷贝和切换次数，性能上一般优于全虚拟化。因此，本文选择了半虚拟化技术来扩展vRNIC为I/O设备。在半虚拟化技术中，将基于文件的共享队列作为虚拟机和vRNIC的命令通道，RDMA应用的命令和参数传输到vRNIC，然后由vRNIC执行命令的处理逻辑。同样的，信号和中断机制，可以通过进程间的事件描述符实现，再借助虚拟机监视器将事件通知转化为内部的中断信号。

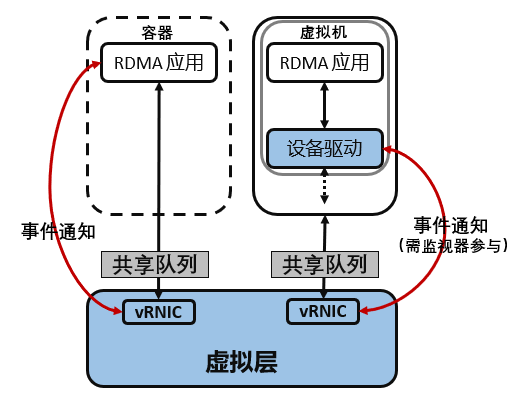


图 4‑5 uniVerbs接口通用性设计

* **设计虚拟机中的设备驱动：**

设备驱动的目标是以I/O方式完成RDMA应用对vRNIC的访问。其具体工作内容是：如图4-5所示，将RDMA应用数据和命令转发到内存共享队列，并触发事件通知上述I/O设备；同样的，设备驱动接收设备的中断通知，读取从vRNIC返回的RDMA命令处理结果。总之，该设备驱动可以通过一个轻量的内核模块实现。

通过上述设计，虚拟机内的RDMA应用可以访问vRNIC。为了保证接口的一致性，如图4-5所示，在容器RDMA应用访问vRNIC时也采用了同等设计，具体为：在数据传输上，同样使用基于共享文件的内存共享队列；在同步机制上，同样采用事件通知机制，只不过容器在同步过程中不存在陷入监视器或注入中断这些虚拟化环节。

当主机上存在多个虚拟机或容器时，如果各vRNIC的共享文件不采取隔离措施，可以被其他容器通过扫描发现，。为了解决这个问题，本文将虚拟机运行在容器环境中，并基于容器的挂载命名空间，将每个vRNIC中供uniVerbs接口使用的共享文件分别放置在对应的专有目录，挂载到相应容器中，包括运行虚拟机的容器。此时，由于挂载命名空间隔离，虚拟层中各vRNIC的共享文件仅对使用的容器或虚拟机是可见的。

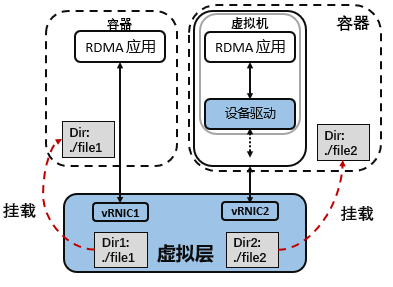


图 4‑6 uniVerbs接口隔离性设计

### uniVerbs接口优化

完成接口构建后，虚拟机和容器的RDMA应用请求均可以传输到虚拟层的vRNIC设备进行处理，同时，命令处理的结果也可以返回给RDMA应用。在uniVerbs中，通过对vRNIC中的RDMA资源进行映射，实现了数据零拷贝和绕过虚拟层，数据命令不用在vRNIC中进行处理，实现了与原生一致的工作模式，提高虚拟RDMA的网络性能。

**（1）完全的零拷贝**

QP队列中RDMA工作请求以及注册内存中的传输数据，最终需要传输到物理RDMA网卡中。vRNIC通过映射单元，其内部虚拟QP和数据块等已经与物理RDMA网卡之间完成了映射，因此，RDMA应用只需与vRNIC的QP或数据块等资源完成映射，即可实现QP中工作请求和注册内存块中数据到RDMA物理网卡的零拷贝。

数据的零拷贝技术是基于数据的使用双方拥有可见的共同的内存物理页面。在前面一致性接口的设计过程中，恰好通过基于文件的共享内存来传输RDMA应用的命令和参数。因此，同样可以通过共享文件将RDMA应用的QP和数据块映射到vRNIC中，实现RDMA工作请求和数据从虚拟实例到vRNIC的完全的零拷贝。

但是，在虚拟机中，由于虚拟机操作系统的内存管理机制，RDMA资源的虚拟机物理内存不一定连续，对应vRNIC中的映射内存区域也不是连续的。此时，vRNIC无法再将该虚拟内存区域作为虚拟RDMA资源映射到物理网卡中。为了解决这一问题，利用了用户空间的虚拟内存重映射机制，将vRNIC中不连续的RDMA资源虚拟内存区域，重映射为对应的连续虚拟内存，并且映射后的物理内存页面顺序与原来一致。

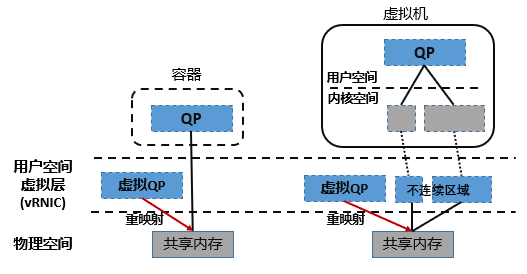


图 4‑7 QP资源从容器和虚拟机到vRNIC的映射

**（2）绕过虚拟层优化**

在RDMA传输过程中，除了将工作请求和数据送达网卡设备，还需要执行按门铃的操作驱动物理网卡工作。在前面的设计中，RDMA物理网卡的门铃仅被映射到了虚拟层vRNIC中，仍需要映射到虚拟机或容器的RDMA应用中，才能实现传输数据时绕过虚拟层以及内核，与原生RDMA一致。否则，按门铃的动作需要转发到虚拟层执行，这一操作在RDMA数据路径中，是高频操作，因此，转发会带来较大的性能损耗。

为了实现绕过虚拟层的数据传输效果，本文将门铃寄存器在虚拟层的监管下，映射到了虚拟机或容器的RDMA应用空间。但是，RDMA应用和vRNIC在主机上属于两个不同的进程，彼此具有隔离的虚拟地址空间；同时门铃寄存器位于设备地址空间，无法通过共享内存进行映射。要解决这一问题的关键在于，RDMA应用的进程需要知晓门铃寄存器的物理地址。

因此，虚拟机或容器的RDMA应用在创建RDMA上下文时，会向虚拟层的vRNIC发送请求，在虚拟层的监管下，获取对应的门铃物理地址，例如门铃对应的物理页码。RDMA应用获取物理地址后，在自身进程内借助主机操作系统将RDMA应用中的门铃虚拟地址映射到对应的门铃物理页面。在这个过程中，虚拟机的映射操作需要经过地址翻译。

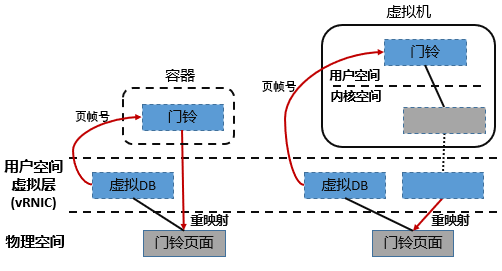


图 4‑8 门铃寄存器从vRNIC到容器和虚拟机的映射

## 6, 测试

为了验证uniRDMA是否能够实现前文提出的RDMA虚拟化目标，本章在物理RDMA平台上对uniRDMA原型进行了多方面的测试，除了网络性能的基准测试，如吞吐量、延迟等，还包括真实的应用程序测试。通过对uniRDMA虚拟化框架的相关测试，本文期望得到以下几个问题的答案：

* uniRDMA在容器和虚拟机场景下的网络性能能否与硬件虚拟化接近？
* uniRDMA是否在大规模容器和虚拟机集群环境下均具有高可扩展性？
* uniRDMA是否能适应于容器和虚拟机的真实RDMA应用环境？

### 6.1 测试环境配置

uniRDMA的所有测试工作均在两台服务器上面进行，测试环境主要包括三个部分： 主机服务器、容器和虚拟机。详细的环境配置参数如表6-1所示：

表 6‑1 测试环境主要参数

|  |  |  |
| --- | --- | --- |
| **服务器** | CPU型号 | Intel Xeon E7-4850 v4 |
| CPU 核数 | 64 |
| 内存 | 1 TiB DDR4/DDR3 |
| 内核 | CentOS 7.4.1708 (Linux 3.10.0) |
| RDMA物理网卡 | Mellanox ConnectX-3 56 Gb/s |
| RDMA驱动 | Mellanox OFED 4.4-2.0.7.0 |
| 虚拟机监视器 | QEMU 5.1.50 |
| 容器引擎 | Docker 18.06.1-ce |
| **虚拟机** | 虚拟CPU核数 | 16 |
| 内存 | 64 GB |
| 内核 | CentOS 7.4.1708 (Linux 3.10.0) |
| **容器** | 虚拟CPU核数 | 64 |
| 内存 | 64 GB |
| 内核 | CentOS 7.4.1708 (Linux 3.10.0) |

如表6-1所示，服务器采用的RDMA物理网卡是Mellanox ConnectX-3 56 Gb/sec，其按照Infiniband协议进行RDMA通信。服务器、容器和虚拟机的操作系统均为CentOS 7.4.1708，对应的Linux内核版本是3.10.0-693.el7.x86\_64。主机服务器安装的RDMA驱动是Mellanox OFED 4.4-2.0.7.0，适配RDMA物理网卡设备和操作系统。此外，整个uniRDMA框架采用GCC/G++ 4.8.5进行编译，并选择O3编译优化级别。

### 6.2 Basic benchmark

RDMA作为一种高性能的新型网络，大量应用于深度学习训练、大数据应用等数据密集型场景，吞吐量和延迟是衡量RDMA网络性能的核心指标。在QP队列建立连接之后，RDMA支持单边和双边这两种不同的数据传输操作。由于单边操作和双边操作在延迟和吞吐量的性能表现上存在差异，因此，本文对它们分别进行了测试和分析。

本文基于RDMA基准测试工具Mellanox perftest[41]，分别测试了各场景下uniRDMA、原生RDMA、硬件虚拟化SR-IOV与软件虚拟化FreeFlow的吞吐量和延迟，以检测uniRDMA是否能够达到与原生RDMA性能接近的目标。其中，对于双边操作（Send/Recv），使用ib\_send\_bw和ib\_send\_lat命令进行测试；对于两种单边操作（Write和Read），以Write为代表，使用ib\_write\_bw和ib\_write\_lat命令进行测试。具体测试过程是：在测试程序的客户端和服务端建立RDMA连接后，每次传输的消息字节数将从4B依次递增到1MB，在每种特定的消息大小下，分别迭代测试1000次数据传输的吞吐量和延迟，最后计算对应的平均吞吐量和平均延迟。

**吞吐量**：uniRDMA等在perftest基准测试中，双边操作的吞吐量测试结果如图6-1和表6-2所示，单边操作的吞吐量测试结果如图6-2和表6-3所示。从已有的测试数据来看，uniRDMA无论在虚拟机还是在容器场景下，其双边操作和单边操作的吞吐量始终与硬件虚拟化SR-IOV接近，与原生RDMA差距较小。

对比FreeFlow，在小消息场景下，uniRDMA的吞吐量达到了是对方的4~6倍。这是因为，在小消息场景下，在发送同等的数据量时，测试程序需要更加频繁的执行RDMA数据命令。FreeFlow将所有的数据命令都转发给软件虚拟层处理，因此，命令转发的延迟会逐渐积累导致其吞吐量下降明显。而uniRDMA通过映射所有的RDMA资源，在容器或虚拟机的用户空间即可执行数据命令，因此，不存在数据命令转发到软件虚拟层的延迟积累。此时，uniRDMA性能与硬件虚拟化技术SR-IOV以及原生RDMA接近。

当消息逐渐增大时，如达到64KB时，各框架的吞吐量都趋于一致。这是因为RDMA物理网卡对于消息的处理已经达到了物理带宽的极限，FreeFlow的延迟开销已经被RDMA物理网卡的硬件等待延迟所覆盖。

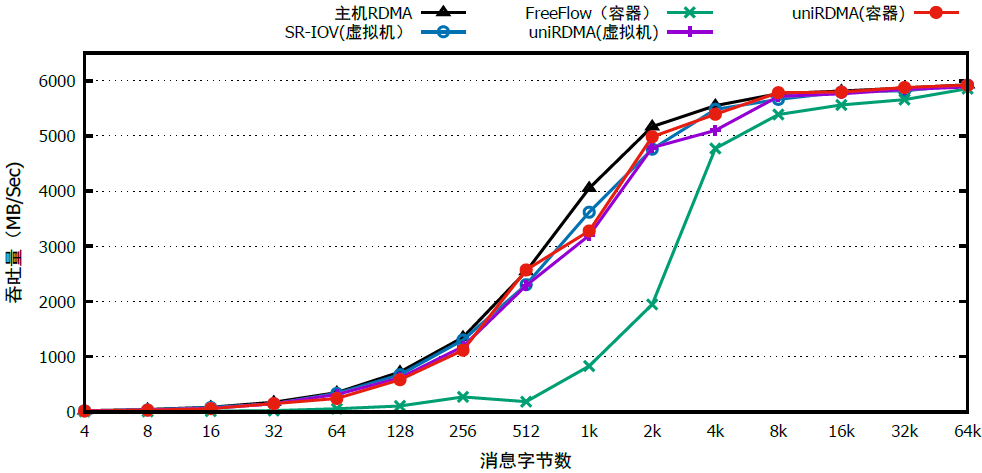


图 6‑1 双边操作Send/Recv吞吐量 (MB/Sec)

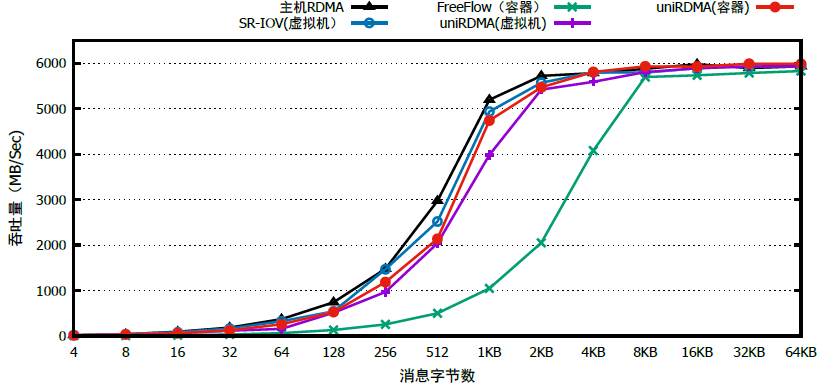


图 6‑2 单边操作Write吞吐量 (MB/Sec)

**延迟**：uniRDMA等在perftest基准下，双边操作的延迟测试结果如图6-3和表6-4所示，单边操作的吞吐量测试结果如图6-4和表6-5所示。从已有的测试数据来看，uniRDMA无论在虚拟机还是在容器场景下，其双边或单边操作的延迟始终与硬件虚拟化SR-IOV相接近，与原生RDMA差距较小。对比FreeFlow，在小消息场景下， uniRDMA双边或单边操作的延迟仅是对方的40%~60%。当消息逐渐增大时，如达到64KB时，各框架的延迟逐渐趋于一致。

从测试数据看，uniRDMA始终对FreeFlow保持延迟上的优势。这是因为，在小消息场景下，RDMA物理网卡硬件处理的延迟很小，此时，FreeFlow数据命令转发到虚拟层的开销成为RDMA通信的主要延迟。而uniRDMA由于没有数据命令转发开销，因此，其延迟与硬件虚拟化SR-IOV较为接近。

当消息逐渐增大时，RDMA处理消息的延迟也逐渐增加。而FreeFlow和uniRDMA软件虚拟化框架在数据上均实现了零拷贝，因此，其延迟开销仍保持不变，此时，网卡硬件处理延迟随着转发消息变大而逐渐增加，开始成为影响网络延迟的最主要因素。最终，FreeFlow的转发开销相比网卡硬件处理延迟来说可以忽略不计，此时各框架的延迟逐渐趋于一致。

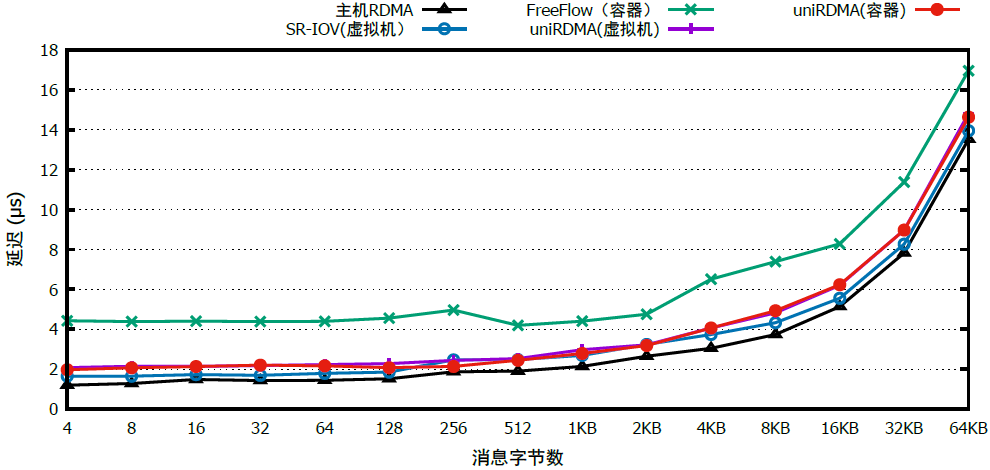


图 6‑3 双边操作Send/Recv延迟 ()

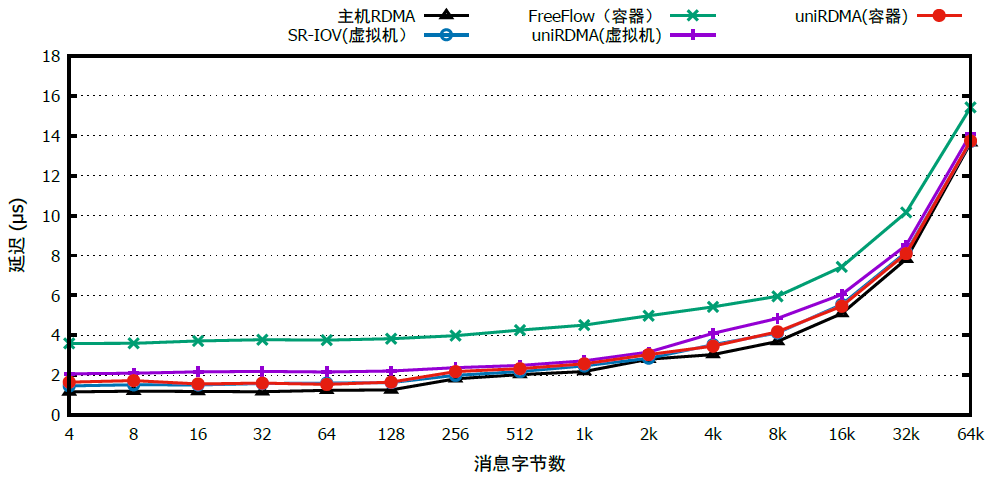


图 6‑4 单边操作Write延迟 ()

**可扩展性：**

可扩展性是RDMA虚拟化在大规模虚拟机或容器集群下需要面临的挑战。为了分析uniRDMA的可扩展性， 本文使用Mellanox perftest中的ib\_write\_bw命令，分别测试了uniRDMA在两台服务器之间，针对2、4、8、16、32、64、128对虚拟实例，选择随机数量的虚拟实例对，执行ib\_write\_bw命令测试发送128KB数据的吞吐量结果。其中，对于服务器的虚拟实例，既有全是虚拟机、全是容器的场景，也有虚拟机和容器混合（各占50%比例）的场景。最终得到虚拟实例对之间的平均吞吐量结果如表6-6所示。

表 6‑6 不同场景和规模下的平均吞吐量（MB/Sec）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **SR-IOV** | **FreeFlow** | **uniRDMA** | | |
| **虚拟机占比** | **100%** | **0%** | **100%** | **0%** | **50%** |
| **容器占比** | **0%** | **100%** | **0%** | **100%** | **50%** |
| **2** | 5994.87 | 5951.3 | 5993.43 | 5990.92 | 5990.19 |
| **4** | 5988.53 | 5949.54 | 5949.02 | 5990.01 | 5987.13 |
| **8** | 5991.81 | 5950.77 | 5981.04 | 5987.29 | 5988.62 |
| **16** | 5992.07 | 3877.21 | 5987.65 | 5994.23 | 5987.98 |
| **32** | 5988.74 | 2810.19 | 5919.44 | 5940.64 | 5948.33 |
| **64** | 5920.59 | 523.32 | 5935.81 | 5940.4 | 5939.74 |
| **128** | - | 407.44 | 5847.03 | 5940.56 | 5951.06 |

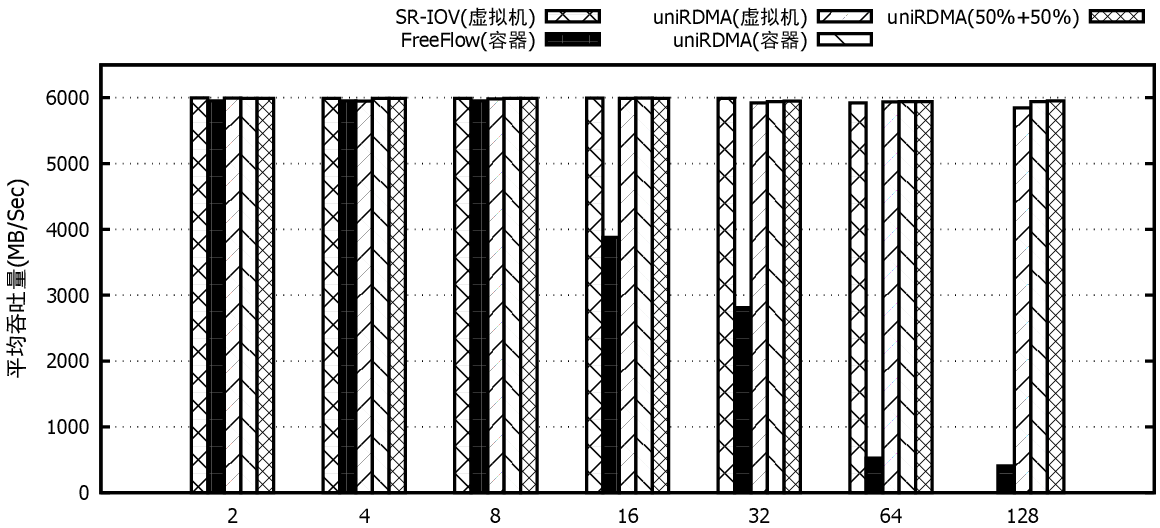


图 6‑5 不同场景和规模下的平均吞吐量（MB/Sec）

从图6-5来看，在全虚拟机场景、全容器场景以及虚拟机和容器混合的场景，uniRDMA均具有良好的可扩展性，仍然保持了稳定的RDMA吞吐量表现。与之相比，FreeFlow此时的吞吐量不到uniRDMA吞吐量的10%。因为uniRDMA的数据命令不需要转发到软件虚拟层进行处理，其虚拟化开销并不因为虚拟实例扩展而增加。FreeFlow在大规模容器同时运行RDMA命令时，其转发到路由虚拟层的同一RDMA上下文进行处理，在调用虚拟实例数据命令对应的QP队列或数据块时，存在锁互斥的开销。因此，FreeFlow在大规模容器场景下出现性能剧烈下降的现象。

**此外，如图6-5所示，uniRDMA和FreeFlow均可以支持128对虚拟实例之间进行通信。然而，Mellanox Conne**ctX-3网卡的最大VF数量仅126个，故不支持128对虚拟实例。这说明了，uniRDMA比SR-IOV具有更高的可扩展性。因为：SR-IOV技术将VF接口静态地分配给某一虚拟机，由该虚拟机独占，无法实现动态共享；而uniRDMA通过虚拟层动态设备池和灵活的映射机制，提高了VF的利用率，能满足超VF数量的虚拟实例规模。

### RDMA应用程序

RDMA网络的高性能，需要为各大数据应用提供有效的性能提升，才能体现其价值。好的RDMA虚拟化框架，需要对各种场景的RDMA应用都有与原生相接近的性能效果。因此，为了测试uniRDMA在真实RDMA应用场景的性能表现，本文测试了uniRDMA等框架在高性能计算应用Graph-500、大数据框架Spark等RDMA应用下的性能结果。

**Graph-500**：在高性能计算领域，Graph-500是一个用来测试信息传递接口（Message Passing Interface，MPI）性能的基准框架[42]。其基于构造的图结构，用户对广度优先搜索（breadth-first search，BFS）和单源最短路径（single source shortest path，SSSP）的性能进行测试，性能指标是每秒遍历的边数（traversed edges per second，TEPS），该值越大说明性能越好。

本文将Graph-500中计算图的节点规模设置为26，边与点的比例按默认参数16，构造完成后的图一共有2^25个顶点，以2^29条边，整个图占用空间大约16GB左右。在测试BFS和SSP时，16个MPI进程轮流分散在两个节点上执行，根据12次测试的结果取平均值，得到的数据如表6-7和图6-6所示（由于FreeFlow在测试过程中存在程序崩溃问题，故缺乏对应数据）。

表 6‑7 Graph-500中各任务的性能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Graph-500** | **uniRDMA**  **(虚拟机)** | **uniRDMA**  **(容器)** | **SR-IOV**  **(虚拟机)** | **主机**  **RDMA** |
| **BFS** | 47.4616 | 47.5301 | 48.4667 | 48.5778 |
| **SSSP** | 10.4557 | 10.463 | 10.588 | 10.6402 |

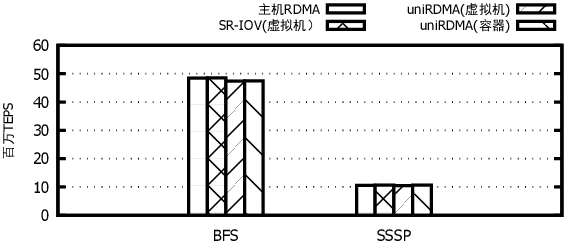


图 6‑6 Graph-500中各任务的性能

从表6-7和图6-6可以看出，uniRDMA的性能表现与主机RDMA和硬件虚拟化SR-IOV技术相当，并不存在明显的性能损耗。这是因为，uniRDMA在数据路径中绕过内核和虚拟层的，不存在因为虚拟化导致的软件转发延迟。

## 7. 总结

本文设计了一种基于用户空间vRNIC的通用RDMA虚拟化框架uniRDMA，由用户空间虚拟层和通用的uniVerbs接口组成。在用户空间虚拟层：基于硬件虚拟化SR-IOV的VF接口，构建了隔离而且高效的vRNIC设备；通过。在uniVerbs接口部分：uniRDMA使用了基于共享文件的I/O通道，实现了虚拟机和容器的统一使用，同时易于虚拟层对vRNIC的管理；通过将共享文件放在独立的专有目录并挂载到各个容器或虚拟机环境，确保了接口的隔离性；通过解决RDMA应用的QP等资源映射以及门铃寄存器映射问题，实现了传输数据时的零拷贝和绕过内核（包括虚拟层）。实验结果表明，uniRDMA在满足通用性和网络管理等功能的前提下，仍具有与原生RDMA接近的高性能。