计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(10):1−8 [doi: 10.15888/j.cnki.csa.007637]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

RDMA 虚拟化相关技术研究①

代　超[1], 刘　强[1], 蒋金虎[2], 张为华[2]

[1](复旦大学 软件学院, 上海 201203)
[2](复旦大学 上海市数据科学重点实验室, 上海 201203)
通讯作者: 张为华, E-mail: zhangweihua@fudan.edu.cn

摘　要: RDMA 网络具有高带宽, 低延时, 低 CPU 负载的特点, 广泛应用于数据密集型任务中, 例如深度学习, 高性能计算, 数据分析等. RDMA 的实现需要软硬件支持, 在云环境下, RDMA 虚拟化方案有助于多用户共享 RDMA 网络传输的高性能, 同时实现对 RDMA 网络的统一管理和控制. 本文调研了近年来的 RDMA 虚拟化解决方案, 覆盖了虚拟机和容器环境; 然后将这些解决方案进行分类和比较; 最后, 对 RDMA 虚拟化中存在的问题和未来的发展做出了总结和展望.

关键词: RDMA; 虚拟化; 云计算

引用格式: 代超,刘强,蒋金虎,张为华.RDMA 虚拟化相关技术研究.计算机系统应用,2020,29(10):1–8. http://www.c-s-a.org.cn/1003-3254/7637.html

Survey on RDMA Virtualization Technology

DAI Chao[1], LIU Qiang[1], JIANG Jin-Hu[2], ZHANG Wei-Hua[2]

[1](Software School, Fudan University, Shanghai 201203, China)
[2](Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)

Abstract: RDMA networks have the characteristics of high bandwidth, low latency, and low CPU load, and are used in a wide range of data-intensive tasks, such as deep learning, high-performance computing, and data analysis. The implementation of RDMA requires software and hardware support. In a cloud environment, the RDMA virtualization solutions help multiple users to share the high performance of RDMA network, while achieving unified management of RDMA networks. In this study, solutions of RDMA virtualization in recent years are collected, which cover virtual ma-chines and container environments. Then, these solutions are classified and compared. Finally, the existing problems and future development in RDMA virtualization are analyzed.

Key words: RDMA; virtualization; cloud computing

1　引言

随着互联网技术的快速发展, 应用的用户规模越来越大, 需要的服务器资源越来越多. 传统服务器部署和维护成本高, 资源利用率低, 难以跟上互联网应用的发展需求. 为了实现对服务器资源的高效利用, 满足大规模用户的需求, 亚马逊最早于 2006 年推出了弹性云计算服务. 云计算将服务器硬件资源, 例如 CPU, 内存, 网络等进行虚拟化, 以按需使用的方式提供给多用户使用. 目前, 云计算已经引起了学界和业界的高度重视, 例如, 谷歌、微软、阿里巴巴和腾讯等企业也竞相发布了各种云计算平台.

虚拟化技术广泛应用于云计算, 模拟器, 安全, 分析等领域[1–6]. 虚拟机相关的虚拟化技术包括 CPU 虚拟化, 内存虚拟化, I/O 虚拟化等. 虚拟化技术可以分为硬

① 基金项目: 国家自然科学基金 (61672160)
Foundation item: National Natural Science Foundation of China (61672160)
收稿时间: 2020-03-10; 修改时间: 2020-04-10; 采用时间: 2020-04-21; csa 在线出版时间: 2020-09-30

件辅助的虚拟化和软件辅助的虚拟化等. 容器也是目前流行的虚拟化方案. 区别于虚拟机, 容器通过 Linux 的命名空间和控制组机制, 让应用程序在独立的运行时环境执行, 具有轻量级隔离和可移植的特点. 目前, 流行的容器引擎有 Docker[7], RKT[8] 等.

RDMA (Remote Direct Memory Access) 是一种高速网络传输技术, 传输数据时可以绕过操作系统直接对远端内存进行读写. 相较于传统的 TCP/IP 网络, RDMA 具有低延迟, 高吞吐, 低 CPU 负载的特点, 更适合数据中心的网络传输需求. 目前, RDMA 广泛应用于云计算系统和大数据平台的网络密集型任务, 例如 TensorFlow[9]、Spark[10]、Hadoop[11] 等.

RDMA 技术提供了高性能的网络传输, 虚拟机和容器技术提供了云计算用户隔离, 共享物理资源以及灵活管理的特点. RDMA 虚拟化技术的目标就是将云环境的特点与 RDMA 网络的高性能特点相结合, 以满足云计算用户对可靠的高性能 RDMA 网络传输服务的需求. 本文调研了近年来 RDMA 虚拟化的主要解决方案, 覆盖了虚拟机和容器的云计算环境, 对相关技术进行了系统全面的分析和讨论. 在此基础上, 对现有 RDMA 虚拟化技术进行了总结, 对存在的问题进行了探讨和展望.

2 RDMA 介绍

RDMA 是一种远程直接访问内存的网络技术. RDMA 的出现是为了解决数据中心网络传输的延迟问题. 在使用 RDMA 连接时, 主机可以直接绕过双方的操作系统, 对远端主机的内存进行读写, 从而避免经过内核时数据拷贝的开销, 同时不占用 CPU 资源. 因此, 相较于传统的 TCP/IP 网络, RDMA 具有低延迟, 高吞吐, 低 CPU 负载的特点, 更适合数据中心的网络传输需求.

目前, RDMA 广泛应用于人工智能、高性能计算、大数据分析和分布式存储等领域. TensorFlow、PaddlePaddle[12] 等流行深度学习框架均支持 RDMA; 根据 2019 年 11 月份的 HPC top500 统计[13], 排名前 10 的超算系统有 6 个使用了 200Gb/s InfiniBand[14](一种 RDMA 网络), 500 强名单中使用该网络的超算系统达到了 140 个; 大数据分析框架 Spark、Hadoop 等均有用到 RDMA 的成熟方案; 在分布式存储系统领域, 键值对系统、文件系统、分布式事务等也都利用了 RDMA 网络特性[15], 例如键值对系统 Pilaf[16]、文件系统 Octopus[17] 以及分布式事务系统 DrTM[18]、FaRM[19] 等.

如图 1 所示, 应用通过调用统一的 Verbs [20] 接口来进行 RDMA 网络传输. 完整的 RDMA 传输过程具有控制路径和数据路径. 首先, 在控制路径上建立连接, 两端的 RDMA 应用在各自的网卡上创建 QP (Queue Pairs) 和 CQ (Complete Queue), 并完成内存注册. QP 和 CQ 均映射到应用的地址空间, 其中, QP 由接收队列和发送队列组成, 应用通过它们控制 RDMA 传输, 而 CQ 负责提供任务完成的通知. 然后, 两端应用的 QP 通过交换信息来完成配对. 当连接建立后, 在数据路径上, 应用可以绕过内核, 直接通过 QP 请求网卡将内存数据以 DMA (Direct Memory Access) 方式读取到网卡缓冲并发送给远端, 或者将网卡缓冲中接收的远端数据以 DMA 方式写入本地内存.

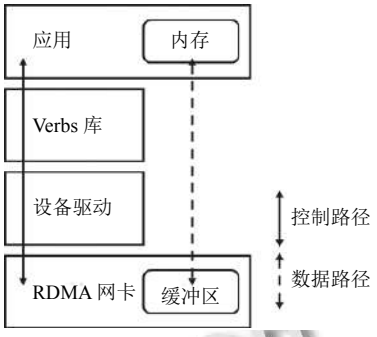

图 1 RDMA 通信

RDMA 有 Write/Read 和 Send/Recv 两种通信方式. Write/Read 是单边操作, 读 (写) 方不需要事先通知对方, 在建立连接后直接读 (写) 远端服务中 RDMA 应用内存地址中的数据. 例如, 在建立 RDMA 连接后, 应用调用 Write 命令, 根据远端提供的虚拟机地址, 直接通过 RDMA 网卡进行数据传输. Send/Recv 是双边操作, 发送方调用 Send 命令时, 接收方需要提前调用 Recv 命令. 发送方和接收方都需要进行两步操作: 用 QP 来进行发送和接收数据, 用 CQ 来通知发送或接收任务的完成. 对于 CQ 中产生的新消息, RDMA 应用可以采用轮询的方式, 定时检查 CQ, 也可以采用基于事件的通知方式.

RDMA 的技术实现需要硬件和软件的支持. 目前, 主要有 3 种 RDMA 技术, 分别是 InfiniBand、RoCE[21] (RDMA over Converged Ethernet)、iWARP[22] (internet Wide Area RDMA Protocol). 其中, InfiniBand 是一种专

为 RDMA 设计的网络, 而 RoCE 和 iWARP 都是基于以太网的 RDMA 技术, 支持相应的 Verbs 接口. RoCE 具有和 InfiniBand 相同的上层协议栈, 在数据链路层完全兼容以太网, 而 iWARP 则保留了 TCP/IP 的完整协议栈.

3 网络 I/O 虚拟化

在虚拟化技术中, 除了 CPU 虚拟化, 内存虚拟化等, I/O 虚拟化也是必不可少的组成部分. I/O 的虚拟化方法主要有: 设备模拟, 全虚拟化, 半虚拟化和设备直连[23].

(1) 设备模拟: 用纯软件来模拟 I/O 设备, 实现硬件设备所具备的接口. 软件模拟可以应用于缺乏物理设备的场景下, 性能较差.

(2) 全虚拟化: 客户机中的 I/O 请求会直接陷入虚拟机监视器, 并由其将请求映射到物理设备, 由设备驱动完成 I/O 请求. 全虚拟化需要虚拟机监视器具有比主机操作系统更高的优先级, 可以进行与操作系统无关的存储管理和虚拟化环境的切换.

(3) 半虚拟化: 使用前后端分离的驱动来处理 I/O 请求, 前端驱动程序位于客户机操作系统, 转发客户机应用的 I/O 请求, 后端驱动程序安装在虚拟机监视器中, 接收前端转发的请求并映射到对应的物理设备, 由设备驱动处理请求. 半虚拟化在实现上比全虚拟化简单, 在性能上比设备模拟好, 但需要修改客户机操作系统.

(4) 旁路直连: 由于客户机和设备之间存在监视器, 上述三种方法都会给 I/O 性能带来额外的开销. 旁路直连方法绕过了虚拟机监视器, 由客户机直接使用物理设备, 因此, 能提高 I/O 性能并能开放硬件设备的所有功能. 最初, 旁路直连只允许简单的设备直连, 即物理设备被一台客户机独占. 之后, 通过 SR-IOV (Single-Root Input/Output Virtualization)[24] 等方式, 设备可以提供多个接口供多台客户机直接使用.

网卡是常见的 I/O 设备, 网络传输是典型的 I/O 密集型任务. 网络 I/O 虚拟化需要考虑虚拟化方案的性能. 常用的网络 I/O 虚拟化方案有硬件辅助的虚拟化方案, 如 SR-IOV, 和软件辅助的虚拟化方案, 如半虚拟化等.

4 RDMA 虚拟化

RDMA 作为一种高性能的网络传输技术, 在数据中心和云计算系统中应用十分广泛. 为了维持云环境中的隔离性, 可移植性等, 不能像原生环境一样使用 RDMA, 需要进行 RDMA 虚拟化, 这就不可避免地在 RDMA 的使用引入额外的开销. 如何在虚拟化 RDMA 的同时, 尽可能地维持原生的性能, 这是一大挑战. RDMA 网络传输过程与 TCP/IP 网络不同, 传统的网络 I/O 虚拟化方案并不一定适用于 RDMA 网络虚拟化. 此外, RDMA 网络的应用需求广泛, 应用场景众多, 各种场景下 RDMA 硬件软件栈也不尽相同, 这也给 RDMA 的虚拟化带来了一定的挑战.

随着云计算的发展, RDMA 的虚拟化技术也在不断发展, 覆盖更多的应用场景. 针对不同的应用场景, RDMA 的虚拟化方法也不同. 从虚拟化方法的实现上来讲, 总体可以分为硬件辅助的虚拟化方法, 如设备直连和 SR-IOV[25,26]; 软件辅助的虚拟化方法, 有半虚拟化方案 vRDMA[27] 和 vSocket[28,29] 等, 混合虚拟化方案 HyV[30,31] 和 virtio-RDMA 等[32,33], 针对容器的虚拟化方案 FreeFlow 等[34–36]; 纯软件模拟的方法, 如 SoftRoCE[37] 和 SoftiWARP[38] 等.

4.1 硬件辅助虚拟化

硬件辅助的虚拟化方法让 RDMA 应用直接使用物理网卡设备, 而不用经过虚拟机监视器或其他虚拟机层. 主要有设备直连和 SR-IOV 两种方案:

(1) 设备直连: 由虚拟机监视器将整个 RDMA 网卡设备透传给一个虚拟机. 如图 2 左侧所示, 该方案只能让 RDMA 网卡设备被一个虚拟机 (灰色部分) 独占, 此时虚拟机的 RDMA 性能和原生 RDMA 一致. 但是, RDMA 网卡设备没法与其他虚拟机或虚拟机监视器共享, 虚拟机迁移, 快照等管理功能受到限制.

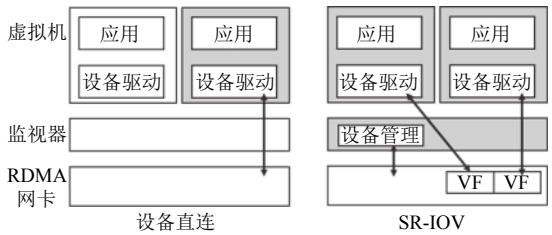

图 2 设备直连和 SR-IOV

(2) SR-IOV: 在支持 SR-IOV 的 RDMA 网卡上, 通过 RDMA 物理网卡 PF (Physical Function) 拥有的多个 VF (Virtual Function), 将一个 RDMA 物理网卡注册为多个独立的虚拟设备. 如图 2 右侧所示, 每个 VF 都有自己的配置空间, 可以分配给一个客户机, 同时虚拟

机监视器仍可访问物理设备并管理注册的虚拟设备. 因此, SR-IOV 能让多个客户机以及虚拟机监视器共享物理 RDMA 网卡. SR-IOV 的性能非常接近于原生 RDMA 性能, 但是, 实现上需要硬件的支持, 而且虚拟机的迁移和子网管理等功能受到限制.

Jithin 等人分析了 SR-IOV 技术对于 InfiniBand 网络的影响, 重点评估了 MPI (Message Passing Interface), PGAS (Partitioned Global Address Space) 的性能表现, 其结果发现: 在点对点通信中, 对于大多数长度的消息 SR-IOV 的性能都能与原生网络相当, 但在集合通信中, SR-IOV 性能比原生网络要差[25]. 同样针对 InfiniBand 网络, Musleh 等人通过对网卡中断参数进行调优, 提高了 SR-IOV 系统的响应能力, 减少了 15–30% 的虚拟化开销[26].

4.2 软件辅助虚拟化

硬件辅助的虚拟化方案, 不仅需要硬件支持, 也不利于云计算环境下虚拟机的统一管理. 软件辅助的虚拟化方法更加灵活, 鉴于不同的应用场景, 可以进一步细分为虚拟机环境下的半虚拟化方法 vRDMA、混合虚拟化方法 HyV 以及容器环境下的 FreeFlow 框架等.

4.2.1 虚拟机环境

从针对应用来看, 虚拟机环境下的 RDMA 虚拟化方案主要有面向 RDMA 应用的 vRDMA[27], HyV[30,31] 和 virtio-RDMA 等[32,33], 以及面向 Socket 应用的 vSocket[28,29] 等. 从虚拟化方式来看, vRDMA 和 vSocket 等和传统半虚拟化方式相似, HyV 和 virtio-RDMA 则采用了混合虚拟化方式, 两者主要区别在于在 RDMA 数据路径上, 混合虚拟化方式实现了零拷贝, 绕过了监视器或主机内核.

VMware 提出了适应于 vSphere[39] 的 RDMA 半虚拟化框架 vRDMA, 由分离的前后端驱动组成. 前端驱动位于客户机, 向客户机应用提供虚拟的 Verbs 接口, 并转发应用的 RDMA 请求给虚拟机监视器, 后端位于虚拟机监视器, 负责转发来自虚拟机的 RDMA 请求给 RDMA 驱动, 由 RDMA 驱动执行请求. 在跨主机的数据传输时, 数据会经过后端驱动到客户机应用. 因此, vRDMA 的数据传输过程并不能完全绕过监视器, 虽然向客户机隐藏了 RDMA 设备细节, 便于虚拟机迁移和管理, 但也带来了一定的延迟. 此外, 当使用基于事件的完成通知时, 主机和客户机都进行轮询操作, 增加了 CPU 负载.

HyV 的整体框架如图 3 所示, 将虚拟机中 RDMA 的控制路径和数据路径进行了分离. 针对控制路径, HyV 采用了半虚拟化技术, 其前端驱动负责截获应用的 RDMA 请求并转发给监视器的后端驱动, 后端驱动则会继续转发 RDMA 请求给主机的 RDMA 网卡驱动. 此外, 后端驱动还会将主机的 RDMA 资源 (例如 QP, CQ 等) 映射到虚拟机中 RDMA 应用的地址空间中. 同样的, RDMA 应用的内存缓冲区也可以通过前后端驱动映射到主机的内存空间. 与传统半虚拟化不同的是, 在完成内存映射后, 数据传输可以直接在虚拟机中 RDMA 应用的数据路径上进行, 从而绕过监视器, 减少数据传输的延迟.

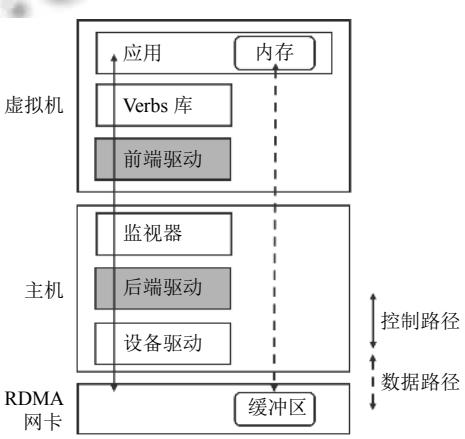

图 3 HyV 框架

在具体实现上, HyV 基于 KVM[40] 虚拟机. 在控制路径上, RDMA 利用了半虚拟化 I/O 框架 virtio[41], 虚拟机使用定制的内核驱动转发控制命令, 主机端使用 vhost[42] 后端驱动. 在数据路径上, 当 RDMA 应用采用基于事件的通知机制时, 采用加锁共享环形队列的方式, 以减少通知的延迟. 由于 HyV 在具体实现上均需要定制的虚拟机和主机内核驱动, 因此对 RDMA 驱动依赖较强, 容器受限于具体的内核版本和网卡硬件.

vRDMA 只适应于 vSphere, 而 HyV 则受限于具体版本的 Linux 系统. 为了支持 HPC 环境轻量级系统 OSv 的 RDMA 虚拟化, Fan 等人提出了与 HyV 相似的轻量级框架 virtio-RDMA. 客户机应用的 RDMA 请求直接从用户空间转发到主机, 避免客户机中用户态到内核态的上下文切换, 同时减少了对 RDMA 内核驱动模块的依赖. 此外, 主机后端驱动还可以使用 vhost-user[43]. 该驱动位于用户空间, 减少了用户和内核之间的切换次数. 对于同一主机的客户机通信, 该框架采用共

享内存的方式进行. 类似 virtio-RDMA 的还有 Mouzakitis 等人提出的轻量级的 RDMA 虚拟化框架[33], 同样利用用户级别的 API, 避免对内核驱动的依赖. 不同的是, 在控制路径上, 主机和客户机之间采用共享内存方式来交互.

除了针对 RDMA 应用的虚拟化方案, 王冬洋等人提出了针对 socket 应用的 RDMA 虚拟化框架 vSocket, 利用 RDMA 网络的高性能加速虚拟机之间的 socket 通信. vSocket 同样由前后端驱动组成, 前端驱动中的用户库负责拦截 Socket 应用的 API 请求. vSokcet 针对 socket 通信中连接建立和数据传输做不同处理, 为了保障安全性, 连接建立过程通过重用原有的内核协议栈进行. 与 HyV 不同的是, 当连接建立后, vSocket 通过前后端驱动的半虚拟化框架进行数据传输, 因为云环境中的网络往往需要主机对来自客户机的数据包进行封装或者对送往客户机的数据包进行解封, 而 RDMA 网卡又不支持该功能, 例如 VXLAN 等. 此外, vSokcet 后端驱动直接面向主机 Verbs 接口, 即数据再通过主机 Verbs 库进行传输, 因此, vSocket 具有一定的通用性, 不像 HyV 依赖特定内核版本, 维护困难等.

4.2.2 容器环境

随着容器技术的发展和流行, 许多 RDMA 应用部署在轻量级的容器中. 针对虚拟机环境的 RDMA 虚拟化方案, 如 vRDMA 和 HyV 等, 并不适应于容器环境. 针对容器环境的 RDMA 虚拟化框架有 FreeFlow 等[34–36].

FreeFlow[34] 在设计上和 HyV 十分类似, 对 RDMA 的控制路径和数据路径分开处理. FreeFlow 在 RDMA 应用容器和底层 RDMA 驱动之间设计了一个路由层, 该层由容器实现, 负责接收来自 RDMA 应用容器的控制命令, 并进行 RDMA 网络传输, 实现对 RDMA 传输的管理和控制. 路由容器和 RDMA 应用容器之间通过共享内存实现 RDMA 传输数据的零拷贝.

如图 4 所示, FreeFlow 的架构由 3 部分组成 (图中灰色部分), 分别是位于 RDMA 应用容器的网络库, 路由容器, 和网络编排器组成. 网络库负责为容器中的 RDMA 应用提供虚拟的与原生一致的 Verbs 接口. 在控制路径上, 应用容器中的 FreeFlow 网络库通过套接字或共享内存的进程间通信方式转发命令和参数给路由容器. 每个主机上都有一个路由容器, 和 RDMA 应用容器位于同一虚拟网络, 路由容器可以直接访问 RDMA 网卡设备, 并执行接收到的 RDMA 应用请求. 路由容器通过共享内存的方式获取 RDMA 应用数据并将网卡接收数据返回给 RDMA 应用, 在数据路径上做到了零拷贝. 网络编排器负责容器 IP 的统一管理以及对 RDMA 网络传输的数据策略, 例如 QoS (Quality of Service) 或计费服务等.

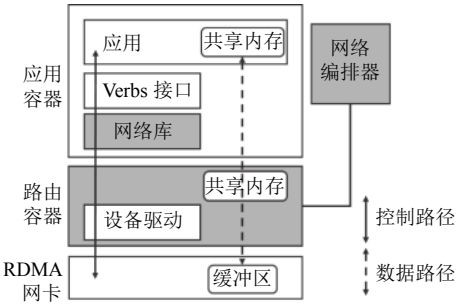

图 4 FreeFlow 框架

FreeFlow 适应于容器云计算环境, 强调了对 RDMA 网络传输数据路径的控制, 提供了数据的 QoS 和流量计费等策略. 但是, FreeFlow 在虚拟化过程中, 在 RDMA 的控制路径上带来了额外的开销, 采用自旋轮询共享内存的方式需要占用 CPU 资源, 而采用 Socket 通信会造成更高的延迟; 在数据路径上, 应用容器和路由容器通过共享内存实现零拷贝, 也带来了一定的安全问题.

除了 FreeFlow, Mellanox 针对 Linux 命名空间和控制组机制, 让多个容器可以共享同一个 RDMA 物理网卡[35,36]. 该方案将物理 RDMA 网卡通过 MACVLAN 技术扩展为多个虚拟接口, 每个容器可以使用一个或多个虚拟接口. 同时, 对于 QP, CQ 等网卡资源采用更细粒度的控制组机制, 容器可以直接使用硬件资源进行数据传输. 该方案虽然保障了隔离性, 但是使用 VLAN 网络时, 在容器迁移过程中需要更新路由信息, 因此可移植性比较差. 另外, 由于容器直接使用 RDMA 网卡进行数据传输, 其控制性比 FreeFlow 要差.

4.3 纯软件模拟

RDMA 网络的普及程度比不上普通以太网. 在缺乏 RDMA 硬件的环境中, 可以通过软件模拟的方法来实现 RDMA 通信. 在该方法中, RDMA 设备的功能完全通过软件的方式得以实现, 并向用户提供与原生一致的 Verbs 编程接口. 因此, 无需依赖 RDMA 网卡、交换机等硬件设备.

RDMA 软件模拟方案主要有 SoftRoCE[37] 和 SoftiWARP[38]. SoftRoCE 是一个软件辅助实现的 RoCE-

Linux 驱动程序, 用于在普通以太网上模拟 RoCE 网络. 而 SoftiWARP 是用于在以太网络上模拟 iWARP 网络的驱动程序. 在具体实现上, SoftRocE 和 SoftiWARP 都可以分为用户层和内核层两个部分. 用户层用以支持上层 RDMA 应用, 用户级库提供了与 Verbs 一致的接口. 内核层包含了可加载的内核模块, 用以支持内核层次的应用. SoftRoCE 的内核模块利用 UDP 栈进行数据传输, 而 SoftiWARP 则在 TCP 内核套接字之上运行.

SoftRoCE 和 SoftiWARP 可以让缺乏 RDMA 硬件条件的大数据中心或云计算平台构建起可用的 RDMA 网络. 但是, 由于软件模拟的方法基于 TCP/UDP 进行网络通信, 它们的性能上受到限制, 在吞吐量和延迟上和传统的以太网 TCP/UDP 通信较为接近, 和原生 RDMA 网络通信的性能差距较大. 此外, SoftRoCE 或 SoftiWARP 由于包含内核模块, 也受限于特定版本的 Linux 系统. 因此, SoftRoCE 和 SoftiWARP 主要用于测试、应用开发以及缺乏 RDMA 硬件却需要部署 RDMA 应用的环境.

4.4 评价

目前, RDMA 的虚拟化方案主要可以分为硬件辅助和软件辅助的方案. 虚拟机环境下, 软件辅助的虚拟化方案可以细分为半虚拟化 (例如 vRDMA 和 vSocket) 和混合虚拟化 (例如 HyV 和 virtio-rdma 等), 两者主要区别在于 RDMA 数据路径是否绕过主机监视器或内核. 容器环境下, 主要有 FreeFlow 等虚拟化框架. 此外, 在缺少 RDMA 硬件条件的场景下, 可以用 SoftiWARP 或 SoftRoCE 来模拟 RDMA 网络.

RDMA 虚拟化的目标是既保证网络的高性能, 同时也不损失云环境的特点. 因此, 对于上述方案的虚拟化效果, 我们从 RDMA 网络性能与云环境特点这两个方面进行讨论. 针对 RDMA 网络性能, 主要考虑延迟, 吞吐量和 CPU 负载这 3 个指标. 针对云环境, 则从隔离性, 可移植性, 控制性 3 个维度来讨论. 其中:

(1) 隔离性: 每个虚拟机或容器实例都有其专属的 RDMA 网络资源, 不与其他实例冲突.

(2) 可移植性: 虚拟机或容器实例迁移管理的便捷程度. 例如, 虚拟机的快照和迁移, 容器的迁移等.

(3) 控制性: RDMA 虚拟化方案对于 RDMA 网络传输的控制, 例如流量限制, 流量计费和 QoS 等.

针对 RDMA 网络性能, 硬件辅助的虚拟化方案由于没有软件虚拟层的额外开销, 所以延迟和吞吐量可以与原生 RDMA 相当. 但是 SR-IOV 需要更多的 CPU 负载来处理网卡中断以保证网络性能. 软件辅助虚拟化一般将控制路径和数据路径分别处理. 在数据路径上, 像 HyV, virtio-RDMA 和 FreeFlow 等框架通过内存映射或共享内存实现了零拷贝, 因此, 延迟和吞吐量较好, 像传统半虚拟化方案的 vRDMA 和 vSocket 等, 其数据需要经过监视器或主机内核, 因而延迟较高. 在控制路径上, 通过共享内存的方式交互延迟更低, 但依赖 CPU 轮询, 通过套接字等转发的方式则可以减少 CPU 负载, 但延迟更高.

针对云环境特点, 硬件辅助的虚拟化方案往往难以达到所有的维度, 例如, 容器与 SR-IOV 网卡 VF 设备的 IP 绑定而不能灵活迁移. 软件辅助的虚拟化方案中, 混合虚拟化 HyV 等没有对数据路径进行统一控制, 因而难以实现云环境下的网络限速, QoS 等数据策略, FreeFlow 则通过对数据路径的控制改进了这些不足. 纯软件模拟的方式, 由于没有硬件限制, 容易具备云环境下的各个特点.

由于 RDMA 技术的多样性以及 RDMA 虚拟化应用场景和实现平台的差异性, 本文选择定性分析, 分别从上述七个维度对不同虚拟化方案进行评估, 其中每个维度定性地分为差, 较差, 较好, 好四个级别, 以尽可能体现出不同方案在各个维度上的差异. 各 RDAM 虚拟化方案的评估结果如表 1 所示.

表 1 RDMA 虚拟化方案评估表

方案	代表	传输延迟	吞吐量	CPU 负载	隔离性	可移植性	控制性
设备直连	—	低	高	低	好	差	差
SR-IOV	—	较低	较高	较低	好	较差	差
软件模拟	SoftRoCE, SoftiWARP	高	低	高	好	好	好
半虚拟化	vRDMA, vSocket	较高	较低	较高	较好	好	好
混合虚拟化	HyV, virtio-RDMA	较低	较高	较低	较好	较好	较差
容器环境	FreeFlow	较低	较高	较高	较好	好	好

5 展望

目前, 针对不同的云计算环境, 有不同的 RDMA 虚拟化解决方案. 例如, 针对虚拟机环境的 vRDMA 和 HyV、针对容器环境的 FreeFlow、针对普通以太网环境的 SoftiWARP 和 SoftRoCE 以及基于硬件的 SR-IOV 虚拟化等. 我们对以后的 RDMA 虚拟化技术研究做出了以下展望:

(1) 通用性: 通用性包含针对不同 RDMA 网络的通用性以及针对不同云计算环境的通用性. 目前, 最新的 RDMA 网络虚拟化方案都考虑到了对不同 RDMA 网络的通用性, 例如 HyV 和 FreeFlow 都支持 3 种不同的 RDMA 网络. 但是, 对于不同云计算环境, 例如不同虚拟方式 (容器或虚拟机), 不同的操作系统等, 还没有通用的 RDMA 解决方案. 通用的 RDMA 虚拟化方案, 有利于统一部署, 对 RDMA 应用统一管理, 能适用于复杂的云计算环境.

(2) 安全性: RDMA 原生的通信模式和流程是安全的, 但是不同的虚拟化方案可能会引入不同程度的通信安全问题. 通过软件辅助的 RDMA 虚拟化网络, 如 FreeFlow, HyV 等, 在转发控制命令的过程中可能被恶意劫持; 通过硬件辅助的方案和纯软件模拟方案在通信方式上与原生 RDMA 没有区别. 因此, 可以尝试通过额外的安全流程来弥补虚拟化中的安全漏洞, 保证 RDMA 通信安全.

(3) 编排管理: 无论是使用 RDMA 应用的虚拟机, 还是容器, 在云计算环境中都需要管理或编排. 编排管理的目的: 通过管理调度来高效利用集群的 RDMA 网卡资源, 提高整个集群的运行效率, 实现负载均衡等. RDMA 网卡作为重要的网络资源, 在如 Kubernetes 之类的容器编排工具里面如何调度管理, 这是一个值得不断研究优化的方向. 而在虚拟机环境中, 直接使用 SR-IOV 技术会导致主机无法对虚拟机使用的 RDMA 资源进行统一管理, 即便在使用类似 HyV 的软件辅助的虚拟化方法时, 虚拟机无法完全感知主机 RDMA 资源, 因此也需要思考如何对主机的 RDMA 资源进行高效动态的分配.

(4) 嵌套虚拟环境: 在实际应用中, 许多 RDMA 应用容器并非部署于裸机, 而是分布于虚拟机中. 此外, 还有虚拟机的嵌套. 在这些复杂的嵌套虚拟环境下, 如何使 RDMA 虚拟化, 让最内层的虚拟环境同样能够利用到 RDMA 网络的高性能, 也是需要考虑的问题.

(5) 智能网卡: 智能网卡或 FPGA 应用越来越广泛, 已有一些研究将其应用到 RDMA 网络虚拟化[33], 协助 CPU 处理网络负载. 与传统网卡相比, 智能网卡可以处理更加复杂的网络负载并支持可替换的数据平面. 在 RDMA 虚拟化方案中, 也可以思考如何与智能网卡的可编程功能高效结合, 以提高云环境下的 RDMA 网络性能.

参考文献

1 Wang ZG, Liu R, Chen YF, *et al*. COREMU: A scalable and portable parallel full-system emulator. Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming. San Antonio, TX, USA. 2011. 213–222. [doi: 10.1145/2038037.1941583]

2 Song CH, Wang WW, Yew PC, *et al*. Unleashing the power of learning: An enhanced learning-based approach for dynamic binary translation. Proceedings of the 2019 USENIX Annual Technical Conference. Renton, WA, USA. 2019. 77–89.

3 Fang ZM, Min QH, Zhou KY, *et al*. Transformer: A functional-driven cycle-accurate multicore simulator. Proceedings of DAC Design Automation Conference 2012. San Francisco, CA, USA. 2012. 106–114. [doi: 10.1145/2228360.2228381]

4 Zhang WH, Ji XF, Lu YP, *et al*. Prophet: A parallel instruction-oriented many-core simulator. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(10): 2939–2952. [doi: 10.1109/TPDS.2017.2700307]

5 Wang HJ, Min QH, Li Y, *et al*. RPSim: A rapid prototyping full-system simulator for SoC software development. 2014 9th IEEE International Conference on Networking, Architecture, and Storage. Tianjin, China. 2014. 259–267. [doi: 10.1109/NAS.2014.45]

6 Zhang WH, Wang HJ, Lu YP, *et al*. A loosely-coupled full-system multicore simulation framework. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(6): 1566–1578. [doi: 10.1109/TPDS.2015.2455499]

7 Docker. https://www.docker.com/, 2020. [2020-04-18].

8 rKt, a security-minded, standards-based container engine. https://coreos.com/rkt/, 2020. [2020-04-18].

9 Abadi M, Barham P, Chen JM, *et al*. TensorFlow: A system for large-scale machine learning. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation. Berkeley, CA, USA. 2016. 265–283.

10 RDMA-based apache spark. http://hibd.cse.ohio-state.edu/, 2020. [2020-04-18].

11 RDMA-based apache hadoop. http://hibd.cse.ohio-state.edu/, 2020. [2020-04-18].

12 PaddlePaddle.https://github.com/PaddlePaddle/Paddle.[2020-03-09].

13 Top500. https://www.top500.org/statistics/list/. [2020-03-09].

14 Introduction to InfiniBandTM. https://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf, 2010. [2020-04-18].

15 陈游旻, 陆游游, 罗圣美, 等. 基于 RDMA 的分布式存储系统研究综述. 计算机研究与发展, 2019, 56(2): 227–239.

[doi: 10.7544/issn1000-1239.2019.20170849]
16 Mitchell C, Geng YF, Li JY. Using one-sided RDMA reads to build a fast, CPU-Efficient key-value store. Proceedings of the 2013 USENIX Annual Technical Conference. San Jose, CA, USA. 2013. 103–114.
17 Lu YY, Shu JW, Chen YM, *et al*. Octopus: An RDMA-enabled distributed persistent memory file system. Proceedings of the 2017 USENIX Conference on Annual Technical Conference. Santa Clara, CA, USA. 2017. 773–785.
18 Wei XD, Shi JX, Chen YZ, *et al*. Fast in-memory transaction processing using RDMA and HTM. Proceedings of the 25th Symposium on Operating Systems Principles. Monterey, Canada. 2015. 87–104.
19 Dragojević D, Narayanan M, Hodson O, *et al*. FaRM: Fast remote memory. Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation. Berkeley, CA, USA. 2014. 401–414.
20 Verbs library: Libibverbs. https://www.openfabrics.org/downloads/libibverbs/, 2011. [2020-04-18].
21 RoCE Introduction. http://www.roceinitiative.org/roce-introduction/, 2020. [2020-04-18].
22 iWARP* RDMA here and now. https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/iwarp-rdma-here-and-now-technology-brief.pdf, 2019. [2020-04-18].
23 Zhang BB, Wang XL, Lai RF, *et al*. A survey on I/O virtualization and optimization. Proceedings of the Fifth Annual ChinaGrid Conference. Guangzhou, China. 2010. 117–123. [doi: 10.1109/ChinaGrid.2010.54]
24 Single root I/O virtualization. http://pcisig.com/specifications/iov/single_root/, 2020. [2020-04-18].
25 Jose J, Li MZ, Lu XY, *et al*. SR-IOV support for virtualization on infiniband clusters: Early experience. Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. Delft, the Netherlands. 2013. 385–392. [doi: 10.1109/CCGrid.2013.76]
26 Musleh M, Pai V, Walters JP, *et al*. Bridging the virtualization performance gap for HPC using SR-IOV for InfiniBand. Proceedings of the 2014 IEEE International Conference on Cloud Computing. Anchorage, AK, USA. 2014. 627–635. [doi: 10.1109/CLOUD.2014.89]
27 Toward a paravirtual vRDMA device for VMware ESXi guests. http://download3.vmware.com/software/vmw-tools/papers/VMTJ_issue_2.pdf, 2020. [2020-04-18].
28 Wang DY, Fu BZ, Lu G, *et al*. vSocket: Virtual socket interface for RDMA in public clouds. Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Providence, RI, USA. 2019. 179–192. [doi: 10.1145/3313808.3313813]
29 王冬洋. 公有云中高性能网络系统研究 [博士学位论文]. 合肥: 中国科学技术大学, 2019.
30 Pfefferle J, Stuedi P, Trivedi A, *et al*. A hybrid I/O virtualization framework for RDMA-capable network interfaces. Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Istanbul, Turkey. 2015. 17–30. [doi: 10.1145/2731186.2731200]
31 Pfefferle J. vVerbs: A paravirtual subsystem for RDMA-capable network interfaces[Master's thesis]. Zurich, Switzerland:ETH Zurich, 2014.
32 Towards a Lightweight RDMA Para-Virtualization for HPC. https://mediatum.ub.tum.de/doc/1344417/1344417.pdf, 2020. [2020-04-18]
33 Mouzakitis A, Pinto C, Nikolaev N, *et al*. Lightweight and generic RDMA engine para-virtualization for the KVM hypervisor. Proceedings of 2017 International Conference on High Performance Computing & Simulation. Genoa, Italy. 2017. 737–744. [doi: 10.1109/HPCS.2017.112]
34 Kim D, Yu TL, Liu HH, *et al*. FreeFlow: Software-based virtual RDMA networking for containerized clouds. Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation. Boston, MA, USA. 2019. 113–126.
35 Containing RDMA and high performance computing. http://caxapa.ru/thumbs/808632/containing_rdma_final.pdf, 2020. [2020-04-18].
36 RDMA container support. https://downloads.openfabrics.org/Media/Monterey_2015/Tuesday/tuesday_08.pdf, 2020. [2020-04-18].
37 SoftRoCE: Software RDMA over Converged Ethernet. https://github.com/SoftRoCE, 2017. [2020-04-18].
38 Softiwarp. https://www.openfabrics.org/downloads/Media/Sonoma2009/Sonoma_2009_Mon_softiwrp.pdf, 2020. [2020-04-18].
39 VMwarevSphere.https://www.vmware.com/products/vsphere.html, 2020. [2020-04-18].
40 KVM. http://www.linux-kvm.org/index.php?title=Main_Page&oldid173792. (2016-11-07)[2020-03-09].
41 Virtio. http://www.linux-kvm.org/index.php?title=Virtio&oldid173787. (2016-10-12)[2020-03-09].
42 UsingVhost. http://www.linux-kvm.org/index.php?title=UsingVhost&oldid3513. (2011-03-08)[2020-04-18].
43 Vhost-user. https://wiki.qemu.org/Features/VirtioVhostUser. (2018-02-06)[2020-04-18].