基于多核的共生虚拟机通信加速机制 XenVMC 的优化

游资奇 任 怡 刘仁仕 管剑波 刘礼鹏
(国防科技大学计算机学院 长沙 410073)

摘 要 在当前的虚拟化平台中,采用共享内存加速位于同一台物理机上的共生虚拟机间的通信是一种被普遍采用的通信加速思路。XenVMC是这些优化方案中的一种,具有效率高、多层透明、支持在线迁移的特点。多核技术的发展为XenVMC提供了进一步的改进空间。基于XenVMC特殊的通信场景,设计了一种多核优化方法,通过设计多核场景下XenVMC的环形共享内存缓冲区,并调度接收方的多个CPU运行,使接收方可以多核并发地接收数据。实验结果表明,使用多核优化后,XenVMC显著地提高了通信事务的吞吐率,并在一定条件下提高了数据的吞吐率。

关键词 虚拟机通信,多核,环形缓冲区

中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.03.017

Optimization of Co-resident Inter-VM Communication Accelerator XenVMC Based on Multi-core

YOU Zi-qi REN Yi LIU Ren-shi GUAN Jian-bo LIU Li-peng
(School of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract Nowadays, on virtualized platform, using shared memory channels to accelerate communication between virtual machines(VMs) located on the same physical machine is a wildly used solution. XenVMC is such a project, which is implemented with high efficiency, full transparency and VM live migration supported. With the development of multi-core technology, XenVMC can be improved further. This paper proposed a multi-core optimization solution on XenVMC based on its individual communication model. By scheduling other CPUs in receiving VM, and updating the design of shared memory channels, XenVMC can receive data concurrently with mutli-cores. Experiment results illustrate that connection transaction throughout is improved obviously and communication throughout is also improved in some cases with multi-core optimization.

Keywords Inter-VM communication, Multi-core, Ring buffer

1 引言

云计算已经成为了当今学术界和工业界的研究热点。作为云平台的重要支撑技术之一,虚拟化技术提供了资源隔离、容错、提高资源利用率等诸多重要功能。Xen是目前主流的开源虚拟化平台之一。在Xen平台上,每个虚拟机被称为一个域;通常称处于同一台物理机上的虚拟机为共生虚拟机。当前的云计算环境中,硬件水平的提升使得每台物理机上所能容纳的虚拟机数量越来越多,共生虚拟机间通信的频率也越来越高。Xen平台上任意两台虚拟机,无论是否存在共生关系,都使用传统的TCP/IP协议栈进行通信。这种通信方式存在着通信路径长、数据拷贝次数多、虚拟机域切换频繁等问题[1],因此通信性能存在很大的优化空间。

一种被普遍采用的优化思路是:在共生虚拟机之间建立共享内存数据通道,当通信双方是共生虚拟机时,通信数据通过共享内存通道进行交换[1]。由于数据旁路减少了通信路径和数据拷贝次数,同时也避免了虚拟机管理器频繁地在多个域之间切换,因此这种优化思路通常能够大幅度提高通信吞吐率。

大部分共生虚拟机通信优化机制的共享内存数据通道采用环形缓冲区实现。数据通信过程中,发送方是缓冲区的生产者,接收方是缓冲区的消费者。环形缓冲区的读写过程是一个生产者-消费者模型。

多核技术的发展为进一步提高虚拟化平台的整体性能提供了新的切入点。随着硬件水平的提升,在实际生产的云环境中,虚拟机可运行在多个虚拟CPU(简称为VCPU)上。通过将针对环形缓冲区的读写任务合理分配给多个VCPU来并发完成,可提高虚拟机双方的通信效率。

然而,现有的共生虚拟机间的通信优化方法[2-6]缺乏对多核环境的支持,未能充分利用多核的优势。这体现在:共生虚拟机通信优化机制的共享内存缓冲区缺乏对多个核高效并发读写的支持。

到稿日期:2016-12-14 返修日期:2017-02-19 本文受国家自然科学基金项目(61502510)资助。

游资奇(1992-),男,硕士生,主要研究方向为云计算、操作系统、虚拟化,E-mail:youziqi529@outlook.com;**任 怡**(1977-),女,博士,副研究员,主要研究方向为云计算、操作系统和虚拟化,E-mail:renyi_nudt@163.com(通信作者);**刘仁仕**(1989-),男,硕士生,主要研究方向为操作系统和虚拟化;**管剑波**(1977-),男,博士,副研究员,主要研究方向为计算机网络、高性能计算和系统软件。

本文围绕基于共享内存的共生虚拟机间的通信优化机制，结合多核技术，给出了开源软件项目面向共生虚拟机间通信优化机制 XenVMC[7] 的一种多核改进方法，并在 XenVMC 系统中加以实现。通过对改进的结果进行比较和分析，总结了多核优化给 XenVMC 带来的通信效率和事务吞吐率的提升。

2　相关工作

2.1　Linux 网卡驱动程序的多核优化

早期版本的 Linux 内核(2.4 版本之前)的网卡驱动程序中，每收到一个数据包就产生一次硬中断。内核在收到硬中断时，会在中断的下半文将此数据包拷贝到内核设置的一个接收队列中；然后内核通知协议栈对此接收队列中的数据包进行处理。这种设计存在一个问题：当网络处于高流量负载的场景时，CPU 会频繁地被硬件中断干扰，从而降低 CPU 的效率。

后续内核版本(自 2.4 版本起)为了解决这一问题，提出了 NAPI 技术。NAPI 技术的核心理念是，使用轮询(poll)的方式批量地获取待接收数据包。在第一次接收到网卡发出的硬件中断后，内核会将轮询函数加入到当前 CPU 的调度队列中；等到调度时机来临后，驱动程序会先关闭硬件中断，之后调用轮询函数从网卡的硬件缓冲或者 DMA 的输入队列读取数据，直到数据读取完或时间片超时。在 Linux 内核的默认设置中，网卡的硬件中断总是由 CPU_0 响应，因此轮询过程也总是由 CPU_0 完成，即使是在多核环境中也如此。由于 NAPI 的设计限制，在多核的环境中，早期的内核网络子系统在接收数据时无法充分发挥多核的优势。使用 NAPI 技术后的 Linux 内核网络子系统的收包流程如图 1 所示。

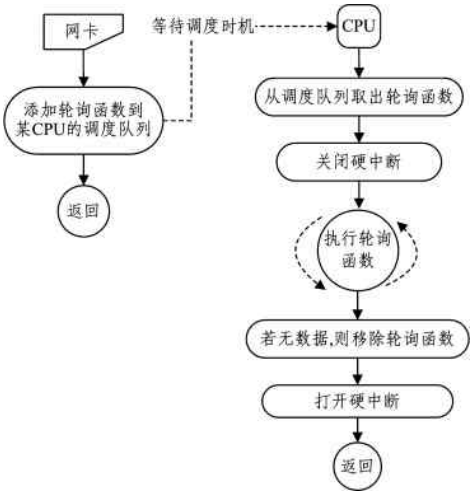

图 1　使用 NAPI 后内核网络子系统的收包流程

Fig. 1　Package receiving process in Linux kernel network subsystem with NAPI

目前，已有一些针对多核场景中 Linux 网络驱动程序所存在问题的优化解决方案。Irqbalance 是一套用于优化中断分配的方案。Irqbanlance 自动收集系统数据并分析，通过修改中断对于 CPU 的亲和性使网卡硬件中断被合理地分配到各个 CPU，均衡各个 CPU 核的负载，以充分利用多核 CPU，提升系统的整体性能。

自 2.6.21 版本起，内核实现了网卡多队列技术来充分发挥多核的优势。网卡多队列需要网卡自身维护多个硬件接收队列，这些队列以包头部的相关信息作为标识区分。同时，网卡需要向内核注册多个硬件中断号，中断号可以与 CPU 核心绑定。不同的队列可以通过不同的硬件中断号唤醒对应的 CPU 读取数据。网卡多队列技术能够使 Linux 网络子系统在数据接收时充分发挥多核的优势，但是需要网卡硬件的支持。

2.2　共生虚拟机间通信优化的相关工作

目前已有一些研究工作采用了共享内存来优化共生虚拟机间的通信。XWay[4] 为每对共生虚拟机间的 TCP 通信连接设置了一组环形缓冲区，在判断出通信双方位于同一台物理机时，发送方在传输层将发送的数据写入环形缓冲区中。XenSocket[5] 以 socket 为模板，设计了一套专用于共生虚拟机间通信的 API。在使用这套 API 通信时，上层应用的数据会直接在双方建立的共享内存中读写。XenLoop[6] 借助 Linux 内核提供的 netfilter 编程接口，在协议栈的网络层截获 IP 数据包，并将发往共生虚拟机的数据包写入共享内存缓冲区中。

XWay[4]，XenSocket[5]，XenLoop[6] 的共享内存缓冲区均设计成环形结构，缓冲区的读写问题使用经典的生产者-消费者模型解决。XWay 和 XenSocket 的缓冲区是面向连接的，每一对通信连接需要维护两个环形缓冲区。XenLoop 的环形缓冲区是面向共生关系的，每两个共生虚拟机间维护两个环形缓冲区。

上述相关工作的环形缓冲区设计，无论是面向连接的还是面向共生虚拟机的，都采用经典的 lock-free 环形缓冲区模型。在实现上，XWay，XenSocket 与 XenLoop 都尚未考虑多核环境中的数据并发读写问题，使用的是面向环形缓冲区的单写单读模型，没有充分发挥多核环境的优势。

2.3　多核场景下环形缓冲区设计的相关工作

有一部分研究工作从理论模型的角度出发，研究多核环境下环形缓冲区数据的并发读写。John Giacomoni 等人[9] 提出了 FastForward，其借助硬件 cache 将缓冲区的读写控制变量存储在 cache 中，提高了数据读写的速度。Patrick 等[10] 对 FastForward 进行了进一步改进，提出 MCRingBuffer。MCRingBuffer 批量地更新存储在 cache 中的读写控制变量，使得数据读写的效率更高。FastFoward 与 MCRingBuffer 在同一时间只允许一个写者或读者操作缓冲区，多个核在同一时间是互斥写入或读取数据。Tsigas 等人[11] 使用 CAS 原语实现了一个环形队列，该队列设计了一定的规则裁定所有写者对队列尾指针和所有读者对队列头指针的竞争，因此允许同一时间多个写者或读者写入或读取数据。

上述理论方法不能很好地用于虚拟化环境。虚拟化环境下支持多核的环形缓冲区设计是一个特定场景下的问题：由于运行在虚拟化环境中，虚拟机的体系架构与真实物理机不同，虚拟 CPU 的 cache 的访问效率并不高，借助 cache 提升缓冲区读写的效率不可行；接收方的数据接收过程发生在虚拟中断的下半文，使用原语控制读写过程可能会引发阻塞，造成错误；并且 Xen 半虚拟化环境中没有对类似 CAS 类原语的指令支持。

3 XenVMC 的多核优化

3.1 多核优化的目的

多核技术的发展为进一步提高云计算虚拟化平台的整体性能提供了新的切入点。在多核环境中，虚拟机的多个 VCPU 映射到物理机的多个 CPU 上，虚拟机操作系统和应用程序运行在一个虚拟机的多 CPU 环境中。而现有的共生虚拟机间的通信优化方法缺乏对多核环境的支持，未能充分利用多核的优势，这主要体现在共生虚拟机通信优化机制的共享内存缓冲区缺乏对多个核高效并发读写的支持。

已有大多数共生虚拟机通信优化工作[2-6]的共享内存数据通道都采用环形缓冲区实现。一对共生虚拟机准备通信时，将建立两个共享内存通道，一个用于发送 VM1 和接收 VM2，另一个则用于发送 VM2 和接收 VM1。数据通信过程中，发送方是缓冲区的生产者，接收方是缓冲区的消费者。现有共生关系感知的相关机制的环形缓冲区的读写过程是一个单写者单读者的生产者-消费者模型。

XenVMC 已有的设计方案没有考虑虚拟机的多核环境，在通信的效率上存在进一步提升的空间。

3.2 多核环境中 XenVMC 的问题分析

XenVMC 的数据收发过程由发送方驱动。发送方在将数据写入缓冲区后，通过事件通道通知接收方。事件通道相当于一个硬件中断，接收方在收到事件通道的通知后，在中断下半文完成数据接收过程。

在发送方，发送者是用户进程，由操作系统调度。XenVMC 无法控制发送者何时以及使用哪个 CPU 向缓冲区写入数据。因此，在不修改操作系统调度的前提下，XenVMC 无法实现多 CPU 并发写。在发送方，修改操作系统调度复杂、工作量大，不但需要考虑写的正确性，而且可能带来额外的性能开销。为此，本文工作暂不考虑支持 XenVMC 的发送方多 CPU 的并发写。

而在接收方，XenVMC 的接收数据的过程发生在中断的下半文，由事件通道绑定的回调函数完成，并且使用了 NAPI 技术。受 Xen 平台设计的限制，接收方的事件通道只能由 CPU_0 响应[13]。因此，目前 XenVMC 的接收方的数据读取总是由 CPU_0 完成，这意味着即使是在多核环境中，仍然是由 CPU_0 单独负责读取数据，并发性低；并且由于使用了 NAPI，轮询函数执行之前会先关闭硬件中断，CPU_0 必须完成轮询过程才能被调度，这导致接收方的 CPU_0 负载过高，而其他 CPU 即使空闲也无法用于数据接收过程，未能充分利用多核 CPU 的优势，一定程度上限制了 XenVMC 的通信效率。而不同于数据发送的是，通过改进 XenVMC 接收方的 CPU 唤醒机制，接收方可支持多 CPU 核并发读取数据。为了保证数据并发读取的有序性，接收方并发读取数据需要缓冲区的支持。

综上，多核环境中，通过对 XenVMC 的共享内存缓冲区机制的优化可以做到互斥写、并发读，即发送方是虚拟机中的多个进程，其写入环形缓冲区中的多个写操作是有序的，即宏观上并发，微观上是互斥的，不允许两个进程同时写；接收方的数据接收过程发生在中断的下半文，由 NAPI 的轮询函数完成，数据接收过程能否并发与其使用的 VCPU 是否被调度有关，没有被调度的 VCPU 核上的进程是不能参与并发的，读并发需要修改接收方的 CPU 唤醒机制，增加缓冲区机制。因此，与经典的多写者多读者并发读写理论模型不同，XenVMC 多核环境中的环形缓冲区设计是一个特定场景下的问题，是经典多写者多读者并发读写问题的一个特殊案例，其读写算法的设计与实现需要考虑 CPU 调度和数据接收的有序性，目前常见的基于原语的阻塞式读写理论模型并不适合。

因此，在多核环境中，需要对 XenVMC 进行优化，重点解决以下几个问题：1）建立合理的环形缓冲区读写并发模型；2）合理调度多个核接收数据；3）正确控制缓冲区的并发读写，从而使其缓冲区机制支持多核并发读取数据，以充分发挥多核的优势。

3.3 多核环境中 XenVMC 的环形缓冲区优化模型

XenVMC 的共享内存在逻辑上组织成环形。每一对共生虚拟机间需要创建两个环形缓冲区，一个供发送方写、接收方读；另一个供发送方读、接收方写。XenVMC 使用 Xen 提供的事件通道机制为每个环形缓冲区相应地设置了一个事件通道，用于通知对方及时地接收数据。发送方向缓冲区写完数据之后，通过事件通道通知接收方。接收方收到事件通知获知数据已经写入缓冲区，进而启动接收，将缓冲区中的数据读入到内存中。

XenVMC 的总体架构示意图如图 2 所示。

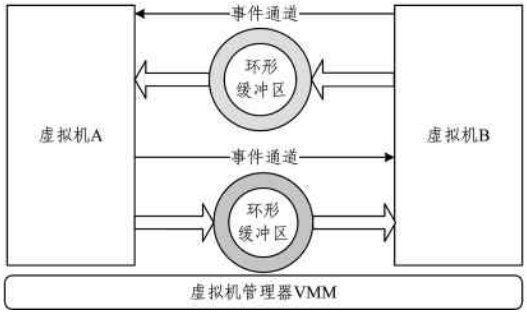

图 2 XenVMC 的总体架构示意图

Fig. 2 Overview of XenVMC architecture

在 XenVMC 的架构中，环形缓冲区相当于一个虚拟的网络设备。接收方接收数据的过程与 Linux 网卡驱动程序的逻辑类似。XenVMC 写入缓冲区的数据是有结构的，发送方写入数据时，首先写入元数据组。元数据组描述了此次写入数据的端口、长度等信息。元数据组固定为 16 字节，其结构描述如图 3 所示。

图 3 XenVMC 通信过程中使用的元数据组

Fig. 3 Structure of metadata used in communication with XenVMC

接收方在接收缓冲区中的数据时，首先从缓冲区中读取 16 字节的元数据组，依据元数据组中的数据长度字段获知接下来需要读取的数据长度。

针对多核环境，对 XenVMC 已有读写模型进行优化，提出了一种无锁的基于标识变量的单写者多读者同步读写模型，如图 4 所示。

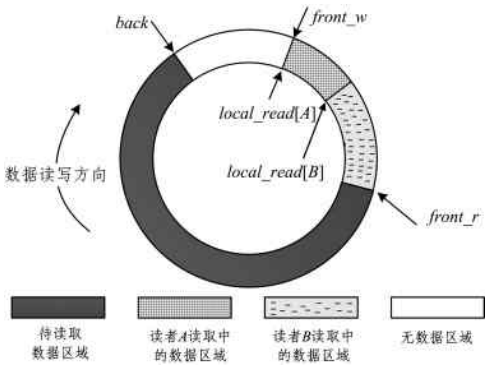

图 4　多核环境中的 XenVMC 环形缓冲区模型

Fig. 4　Ring buffer model in XenVMC multi-core optimization algorithm

如图 4 所示，多核环境下的环形缓冲区设计了两种标识变量：全局标识 *back*，*front_r*，*front_w* 和局部标识 *local_read*[0…*N*−1]（*N* 为 CPU 的核心个数）。每次读写任务都由一个 CPU 完成。*front_w* 标识了读者在此次任务开始前所在的位置；*front_r* 标识了读者在此次任务完成后应在的位置；*back* 则标识了当前写者所在的位置。写操作时，*back* 不能超过 *front_w*；读操作时，*front_r* 不能超过 *back*。

3.4　面向多核环境的 XenVMC 数据发送与接收优化

3.4.1　接收方 CPU 唤醒问题的优化

采用事件分发方法解决 3.2 节指出的接收方已有 CPU 唤醒机制引发的数据读并发受限问题：接收方接收到事件通知后，添加一组操作用于唤醒其他 CPU，从而使多个 CPU 响应事件。Linux 内核为多处理机环境设计了处理器间中断（Inter-Processor Interrupt，IPI），用于各个处理器间的相互通知。Xen 也为虚拟 CPU 提供了虚拟 IPI 中断，可以实现虚拟 CPU 间的相互通知。多核环境下，XenVMC 的接收方在接收到事件通知后，可以在回调函数中通过 IPI 调用唤醒其他空闲的 CPU。数据接收方因此可以多核并发接收数据。使用多核唤醒策略后，接收方的包处理流程如图 5 所示。

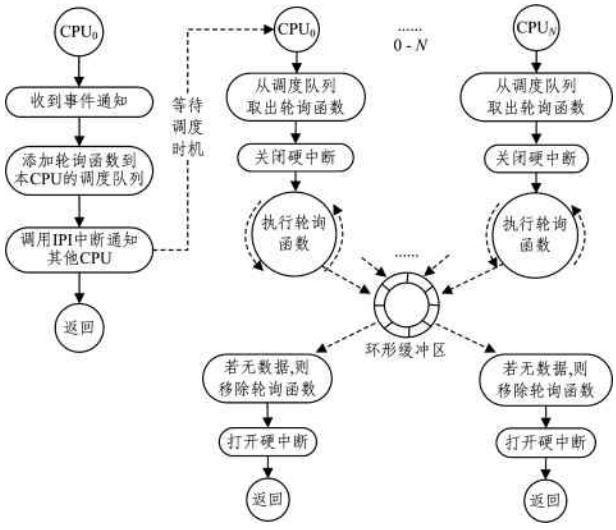

图 5　多核优化后 XenVMC 的数据接收流程

Fig. 5　Data receiving process in XenVMC after enabling multi-core optimization

3.4.2　环形缓冲区的多核读写优化算法

XenVMC 使用多核优化待解决的另外一个问题是，需要设计环形缓冲区的读写算法来支持多 CPU 的并发读。本文基于 XenVMC 数据收发特殊场景中的缓冲区模型，使用一种无锁（lock-free）的基于标识变量的单写者多读者同步读方案。图 6－图 8 详细描述了 XenVMC 环形缓冲区模型的多核读写算法及其中的一些相关定义。

定义 1（缓冲区大小）　共享内存缓冲区的大小，为一固定值，记为 *RING_SIZE*；

定义 2（待写入数据长度）　某一次写任务需要拷贝到缓冲区的数据长度，记为 *data_len*；

定义 3（缓冲区剩余空间）　缓冲区中没有数据覆盖的区域的长度总和。缓冲区的剩余空间在逻辑上不一定连续，这是因为当多个读者在并发读数据时，各个读者的任务完成顺序是不确定的。缓冲区的剩余空间可由 *front_w* 与 *back* 计算出：*remain_space*=*RING_SIZE* − (*back*− *front_w*)；

定义 4（局部变量）　读者在每次读任务开始前，将当前 *front_r* 的值记录在一组变量中，称为局部变量，记为 *local_read*[0…*N*−1]（0…*N*−1 为读者编号）。

图 6　XenVMC 环形缓冲区多核读写算法中的相关定义

Fig. 6　Related definitions of multi-core read-write algorithm in XenVMC ring buffer

数据接收：

1）检查缓冲区是否为空，若是，则直接返回；

2）将 *front_r* 记录在当前读者 *i* 的局部变量 *local_read*[*i*]中；

3）更新 *front_r*：*front_r* = *front_r* + *data_len*；

4）将以 *local_read*[*i*]为起始地址的长为 *data_len* 的区域中的数据拷贝至上层协议的数据队列中；

5）更新 *front_w*：*front_w* = *front_w* + *data_len*；

6）将 *local_read*[*i*]更新为 0。

图 7　XenVMC 环形缓冲区多核读写算法的数据接收部分

Fig. 7　Data receiving of multi-core read-write algorithm in XenVMC ring buffer

数据发送：

1）判断待写入数据长度是否大于缓冲区的剩余空间；若大于，则选择等待或者直接返回；

2）比较 *local_read*[0…*N*−1]与 *back*，若存在 *local_read*[*i*]使得：*back*+*data_len*>=*local_read*[*i*]+*RING_SIZE*，并且 *local_read*[*i*]≠0，则写者进入等待状态；

3）将待写入数据拷贝至缓冲区中起始位置为 *back* 的长为 *data_len* 的区域；

4）通知接收方接收数据。

图 8　XenVMC 环形缓冲区多核读写算法数据发送部分

Fig. 8　Data sending of multi-core read-write algorithm in XenVMC ring buffer

3.5　多核环境下 XenVMC 缓冲区读写的算法正确性分析

3.5.1　数据接收

数据接收算法首先判断缓冲区是否为空，即 *front_r*=*back* 是否成立。若成立，则说明缓冲区为空。由于 *front_r* 的值指示了最早开始读取数据任务的读者在完成读取任务后的位置，因此，若 *front_r*=*back* 成立，则说明所有存在数据的区域都已经有读者负责读取。

读者 *i* 在开始读取任务前，会将任务的起始地址记录在局部变量 *local_read*[*i*]中。对于其他读者而言，由于每位读者在读任务开始前都能计算出待读取数据的长度，因此各个读者读取的数据区域不会重叠。对于写者而言，局部变量

$local_read[0\cdots N-1]$能保证写者待写的区域与所有读者待读的区域不会重叠。

读者在完成读任务后，将 $front\_w$ 更新为 $front\_w+data\_len$，这保证了写者在计算剩余空间时能够准确地计算出未使用的空间，避免写者进程饥饿。

3.5.2 数据发送

数据发送算法首先判断待写入数据的长度是否大于缓冲区的剩余空间，即 $RING\_SIZE-(back-front\_w)>data\_len$ 是否成立。若前者小于后者，程序会直接返回或选择等待，此时的读写冲突由上层应用处理，上层应用会决定下一步的操作。若前者大于后者，存在两种情况：

1）$RING\_SIZE-(back-\min\{local\_read[0\cdots N-1]\})>data\_len$，这意味着剩余空间很富余，即使存在读者没有完成此次读操作，也不会发生读写冲突，因此不会产生错误。

2）存在 $local\_read[i]$ 使得：$RING\_SIZE-(back-local\_read[i])<=data\_len$，这意味着缓冲区的剩余空间不连续，存在着读者 $j(i<j<N)$ 已经完成此次读任务，而读者 i 尚未完成。由上式推导得：$back+data\_len>=local\_read[i]+RING\_SIZE$。根据数据发送算法的第 2）步，此时写者会进入循环等待。此循环等待在读者 i 将 $local\_read[i]$ 赋值为 0 时终止。由于读者的读操作由一个 CPU 在中断的下半文完成，而读者数据的数据接收算法能够正确完成，可以保证 $local\_read[i]$ 在有限的时间内被赋值为 0。因此，写者能够在有限的时间内完成写入操作。

4 多核优化的效果评估

4.1 实验目标与实验设计

XenVMC 多核优化策略的目的是提升多核场景下通信的效率。本文设计实验来验证采用多核优化策略后，XenVMC 的通信吞吐率与通信连接事务处理吞吐率能有一定程度的提升和改善。

通信吞吐率表示单位时间内通信传输的数据量。通信吞吐率提升是采用共享内存加速共生虚拟机间通信的最基本的设计目标，也是 XenVMC 最重要的特性。在单 CPU 接收的通信模式中，只有 CPU_0 能接收数据，一定程度上限制了通信吞吐率。通信连接事务处理吞吐率标识了单位时间内通信双方所能处理的通信连接建立过程的数量。在单 CPU 接收的通信模式中，由于 CPU_0 的负载过高，通信连接建立的响应速度慢，并且存在通信连接事务处理吞吐率抖动的问题。

实验设计从通信吞吐率和通信连接事务处理吞吐率两个方面来验证 XenVMC 优化策略的改进效果。Benchmark 选用 netperf-2.6。Netperf 被广泛应用于网络性能方面的评测，支持多种类型的测试，包括网络通信吞吐率和通信事务吞吐率测试等。Netperf 采用 C/S 架构，通信的两台虚拟机分别运行服务端和客户端。测试过程中客户端向服务端建立一系列连接并发送一定量的字符数据，服务端通过连接数量和接收数据量计算出通信连接事务处理吞吐率和通信吞吐率。

Netperf 可通过参数设置测试过程中传输的数据块大小。通信吞吐率与数据块大小密切相关，本实验在测试时以传输的数据块大小作为自变量来比较采用多核优化策略前后 XenVMC 的通信吞吐率。对于通信连接事务处理吞吐率，Netperf 会根据传输数据块大小的不同对通信连接的建立过程和数量做相应调整。因此在实验中，以传输数据块大小为自变量，考查多核优化前后 XenVMC 在不同通信连接建立场景中的通信连接事务处理吞吐率的变化。

4.2 实验环境

本文实验平台的硬件配置为 Intel Core i5-4460 四核处理器，主频为 3.2GHz。使用的 Xen 版本为 4.6.1。物理机上运行两个虚拟机，它们均以半虚拟化的模式运行。虚拟机中使用的操作系统为 CentOS 6.5，并且内核替换为 Linux-3.18.21。实验时，为每台虚拟机分配两个虚拟 CPU。实验环境的详细参数信息如表 1 所列。

表 1 实验环境参数

Table 1 Parameters of experiment environment

实验环境	参数
物理机 CPU	Intel Core i5-4460，3.20GHz，四核处理器
物理机内存	4GB，DDR3 1333
虚拟机操作系统	CentOS 6.5
虚拟机操作系统内核	Linux-3.18.21
虚拟机管理器版本	Xen-4.6.1
虚拟机 VCPU 数量	2
虚拟机内存	600MB
虚拟化平台网络配置	桥接模式

4.3 实验结果与分析

图 9 展示了单核与多核条件下 TCP 通信的网络吞吐率的结果对比。从图中可以看出，使用多核优化策略后，TCP 的吞吐率并没有提升，反而在一定程度上有所下降。TCP 协议是面向连接的协议，接收方的上层应用和协议栈会对通信连接进行流量控制，XenVMC 也实现了这一语义。因此，XenVMC 在 TCP 通信测试时，发送方的发送速度受到接收方接收进度的反馈。多核环境中，多位读者在环形缓冲中读取数据时，每位读者都会对写者进行流量控制的反馈，因此写者的写入速度受读取进度最慢的读者所限制。另外，使用 TCP 进行通信时，XenVMC 接收方设计了方案以确保数据接收的顺序正确。在多核场景中，这套方案所花费的时间开销更高。因此，使用多核优化后的 XenVMC，其 TCP 通信数据吞吐率的平均测量值不如单核环境中的测试结果。

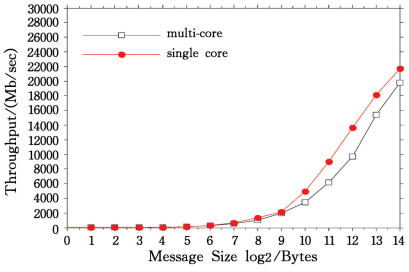

图 9 多核优化前后 XenVMC 的 TCP 数据吞吐率对比

Fig. 9 Comparison of TCP throughput of XenVMC before and after enabling multi-core optimization

图 10 展示了单核与多核条件下 UDP 通信的网络吞吐率。图中结果表明，在传输数据块的大小处于一定范围时，使用多核优化后，UDP 的吞吐率有较为明显的提升。与 TCP 不同，UDP 不是面向连接的协议。接收方的接收速率不会直

接反馈给发送者。因此，使用 XenVMC 的 UDP 语义进行通信时，接收方的接收速率是通信的性能瓶颈。使用多核优化后，接收方的接收过程并发执行，可以在一定程度上提高通信数据的吞吐率。

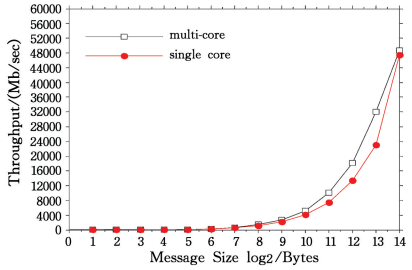

图 10　多核优化前后 XenVMC 的 UDP 数据吞吐率对比

Fig. 10　Comparison of UDP throughput of XenVMC before and after enabling multi-core optimization

图 11 和图 12 显示了使用多核优化前后通信事务吞吐率的对比。测量的数值为每秒 TCP 请求/响应数量和 UDP 请求/响应数量。使用多核优化后，XenVMC 对于连接的请求/响应处理速度显著提高。这是因为使用多核优化后，每个核都能够单独进行响应；单核时，响应需要等待轮询函数在此 CPU 上的调度时机；多个核参与调度时，只需要任意一个核的调度时机到来即可。另外，从图中的结果可以看出，多核优化后，通信的事务吞吐率更加平稳。这是因为多个核参与调度时，一般负载更轻的核会更快地等到调度时机。

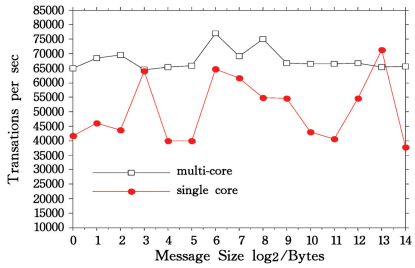

图 11　多核优化前后 XenVMC 的 TCP 事务吞吐率对比

Fig. 11　Comparison of TCP transcation throughput of XenVMC before and after enabling multi-core optimization

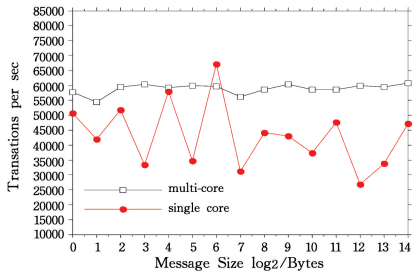

图 12　多核优化前后 XenVMC 的 UDP 事务吞吐率对比

Fig 12　Comparison of UDP transcation throughput of XenVMC before and after enabling multi-core optimization

结束语　本文基于目前业界领先的共生虚拟机通信加速机制 XenVMC，针对 XenVMC 在多核环境中的特殊通信模型提出了优化设计方案，主要对接收方共享内存环形缓冲区的并发读取提出了相关解决方法。接收方在使用优化方案后可以唤醒其他空闲的 CPU 接收数据，支持多个核并发地接收数据。

实验证明，在采用多核优化后，XenVMC 能够明显提升通信事务吞吐率，并在一定程度上提高通信数据吞吐率。

参 考 文 献

[1] REN Y，LIU L，ZHANG Q，et al. Shared-Memory Optimizations for Inter-Virtual-Machine Communication[J]. ACM Computing Surveys，2016，48(4)：1-42.

[2] REN Y，LIU L，LIU X J，et al. A Fast and Transparent Communication Protocol for Co-Resident Virtual Machines[C]// International Conference on Collaborative Computing：Networking. IEEE，2012：70-79.

[3] REN Y，LIU L，ZHANG Q，et al. Residency-Aware Virtual Machine Communication Optimization：Design Choices and Techniques[C]// 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD). IEEE，2013：823-830.

[4] KIM K，KIM C，JUNG S I，et al. Inter-domain Socket Communication Supporting High Performance and Full Binary Compatibility on Xen[C]// Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. ACM，2008：11-20.

[5] ZHANG X L，SUZANNE M，PANKAJ R，et al. XenSocket：A high-throughput interdomain transport for virtual machines [C]// Middleware 2007. Springer，2007：184-203.

[6] WANG J，KWAME-LANTE W，KARTIK G. XenLoop：a transparent high performance inter-vm network loopback[C]// Proceedings of the 17th International Symposium on High Performance Distributed Computing. ACM，2008：109-118.

[7] XenVMC Source Code[OL]. https://github.com/XenVMC-Group/XenVMC.

[8] LAMPORT L. Proving the Correctness of Multiprocess Programs[J]. IEEE Transaction on Software Engineering，IEEE，1977，3(2)：125-143.

[9] GIACOMONI J，MOSELEY T，VACHHARAJANI M. Fast Forward for Efficient Pipeline Parallelism－A Cache-Optimized Concurrent Lock-Free Queue[C]// ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP). ACM，2008：43-52.

[10] LEE P P C，BU T，CHANDRANMENON G. A Lock-Free，Cache-Efficient Multi-Core Synchronization Mechanism for Line-Rate Network Traffic Networking[C]// Proceedings of International Parallel and Distributed Processing Symposium. IEEE，2010：1-12.

[11] TSIGAS P，ZHANG Y. A Simple，Fast and Scalable Non-Blocking Concurrent FIFO queue for Shared Memory Multiprocessor Systems[C]// 3rd ACM Symposium on Parallel Algorithms and Architecture. ACM，2001：134-143.

[12] LIU R S，REN Y，YOU Z Q，et al. XenVMC：a Co-location Aware Inter Virtual Machine Communication Optimization[C]// HPC China. 2015. (in Chinese)
刘仁仕，任怡，游资奇，等. XenVMC：一种共生关系感知的虚拟机域间通信优化机制[C]// HPC China. 2015.

[13] 石磊，等. Xen 虚拟化技术[M]. 武汉：华中科技大学出版社，2009.