DOI：10.7652/xjtuxb201504009

基于 I/O 受限进程识别的虚拟处理器调度机制

王强[1]，董小社[1]，王恩东[2]，朱正东[1]
（1. 西安交通大学电子信息与工程学院，710049，西安；
2. 浪潮集团有限公司高效能服务器和存储技术国家重点实验室，250013，济南）

摘要：针对多核平台的虚拟化环境中客户机与虚拟机管理器（virtual machine monitor，VMM）之间语义缝隙造成客户机 I/O 性能下降的问题，提出了一种基于 I/O 受限进程识别的虚拟处理器（virtual CPU，vCPU）调度机制。该机制在客户机内部利用推断技术识别 I/O 受限进程，通过客户机与 VMM 的协作实现 I/O 事件与 I/O 受限进程的关联，利用保证客户机之间公平性的虚拟对称多核处理器（virtual symmetric multi-core processor，vSMP）Internal 调度算法，优先调度与 I/O 事件关联的 I/O 受限进程所在的 vCPU 来桥接客户机与 VMM 之间的语义缝隙，提高拥有 vSMP 的客户机中 I/O 负载性能。测试结果表明，相比于 KVM 虚拟化环境的 CFS 调度机制，该机制可以在保证客户机 CPU 公平性的前提下，有效提升运行混合负载的 vSMP 客户机中 I/O 负载性能，同时只增加较小的客户机额外开销，可以应用在负载多样性和不可预测性的虚拟桌面和云计算环境中。

关键词：虚拟化；虚拟机管理器；虚拟处理器调度；I/O 受限进程
中图分类号：TP333 **文献标志码**：A **文章编号**：0253-987X(2015)04-0053-08

A Virtual CPU Scheduling Mechanism Based on I/O-Awareness

WANG Qiang[1]，DONG Xiaoshe[1]，WANG Endong[2]，ZHU Zhengdong[1]
（1. School of Electronics and Information Engineering，Xi'an Jiaotong University，Xi'an 710049，China；
2. State Key Laboratory of High-End Server and Storage Technology，Inspur Group Co.，Ltd.，Jinan 250013，China）

Abstract：A novel I/O-aware virtual CPU（vCPU）scheduling mechanism in virtualized environments based on multi-core platform is proposed to eliminate the semantic gap between guest and virtual machine monitor（VMM）that dramatically degrades the performance of I/O-bound workloads in virtual symmetric multi-core processor（vSMP）virtual machine（VM）. Inference techniques are used to identify the I/O-bound tasks，and the I/O-bound tasks and I/O events are correlated through the coordination between the guest operating system and VMM. Then the correlation information is used to bridge the semantic gap by a vSMP Internal algorithm so that a vCPU with I/O-bound task can selectively be scheduled to handle its incoming events promptly with ensured fairness among VMs. Extensive evaluations and comparisons with the CFS scheduler used by the KVM virtual machine monitor show that the proposed mechanism significantly improves I/O performance of vSMP VMs with ensured CPU fairness，and little overhead is introduced to guest. Therefore，the proposed mechanism is widely applicable in such

收稿日期：2014-08-14。 作者简介：王强（1989—），男，硕士生；朱正东（通信作者），男，高级工程师。 基金项目：国家高技术研究发展计划资助项目（2008AA01A202，2012AA01A306）；国家自然科学基金资助项目（61173039）；国家青年自然科学基金资助项目（61202041）。
网络出版时间：2015-02-10 网络出版地址：http://www.cnki.net/kcms/detail/61.1069.T.20150210.0924.002.html

environments with unpredictable and varying workloads as virtual desktop and cloud computing.
Keywords: virtualization; virtual machine monitor; virtual CPU scheduling; I/O-bound

云计算是推动 IT 转向以业务为中心模式的一次重大变革,云平台可以通过互联网为用户提供按需动态扩展的基础架构服务。虚拟化是构建云基础架构不可或缺的关键技术之一,可以使得多个客户机能同时安全相互隔离地运行在同一台物理机器上。虚拟化在资源的有效利用、动态调配和高可靠性方面存在着巨大优势,同时多核处理器的日益广泛应用扩大了虚拟化技术的优势。使用多核处理器为平台的虚拟机系统导致客户机操作系统与虚拟机管理器(virtual machine monitor,VMM)构成两级调度框架[1]。两级调度实体使用各自的调度策略和机制,相互之间缺少协同合作,导致客户机与 VMM 之间存在语义缝隙[2]。在客户机聚合的环境中,语义缝隙问题妨碍了 VMM 有效地为每个客户机分配所需资源。特别地,对于拥有虚拟对称多核处理器(virtual symmetric multi-core processor, vSMP)的客户机,VMM 给 vSMP 客户机分配多个具有相同计算能力的虚拟处理器(virtual CPU, vCPU)。vSMP 客户机操作系统为了提高客户机 I/O 受限进程的性能,优先将 I/O 受限进程调度到 vCPU 中运行,然而 VMM 缺乏对 vSMP 客户机 vCPU 内部 I/O 进程的感知,仅仅是简单地调度 vSMP 客户机的多个 vCPU 分时共享物理计算资源,无法优先调度运行客户机 I/O 负载的 vCPU。在虚拟桌面以及云计算环境中,客户机配备多个 vCPU 同时运行 I/O 和 CPU 混合负载来充分利用底层物理处理器不断增长的计算能力[3]。同时,在虚拟桌面以及云计算环境中负载一般是多样性和不可预测性的[4],这种缺少协同合作的两级调度实体会导致虚拟桌面以及云计算环境中的 vSMP 客户机 I/O 性能严重下降。

目前已有研究主要从 vCPU 调度的角度解决语义缝隙导致的问题。Kim 等对 Xen 的 Credit 调度器[5] boosting 调度机制进行改进,提出了 Partial bosting 机制[6]。Partial bosting 机制中一旦有 I/O 受限进程对应的 I/O 事件到来,立即调度对应的 vCPU 到物理 CPU 上运行,I/O 受限进程处理完 I/O 事件之后将该 vCPU 调度出物理 CPU。Partial bosting 可以在保证公平性的前提下,提高混合负载情况下客户机 I/O 受限进程的性能,但是该机制基于单核物理平台下实现,没有考虑 vCPU 的迁移和同步问题,同时只适用于单个 vCPU 的客户机,不能扩展到客户机多 vCPU 的环境中,并且客户机 I/O 负载的识别机制不适用于最新的硬件辅助 MMU 虚拟化技术[7-8]。为了适用多核物理平台多 vCPU 环境,Kim 等提出了 vAMP 机制[9],该机制根据 vCPU 中运行负载动态调整一个客户机中各 vCPU 运行时间占客户机总运行时间的比例,运行时间比例大的 vCPU 称为快速 vCPU,更容易被 VMM 调度运行。通过在 VMM 和客户机中添加扩展使得客户机中与用户交互的负载运行在快速 vCPU 中,来改善交互负载的性能。vTurbo[10] 和 vBalance[11] 从降低客户机 I/O 中断处理的延迟入手来解决语义缝隙导致的客户机 I/O 性能下降问题。二者都没有考虑客户机 I/O 进程所在 vCPU 的调度延迟而导致 I/O 性能下降的问题。

针对运行混合负载的 vSMP 客户机中 I/O 受限进程性能下降问题,提出了一种基于 I/O 受限进程识别的 vCPU 调度机制。该方法在客户机内部识别 I/O 受限进程,通过 VMM 与客户机合作实现 I/O 事件与 I/O 受限进程关联,以优先运行与 I/O 事件关联的 I/O 受限进程所在的 vCPU,可以有效提升运行混合负载的 vSMP 客户机中 I/O 负载性能。

1 I/O 受限进程识别的 vCPU 调度机制

为了在混合负载中识别客户机 I/O 受限进程,在客户机内部加载一个半虚拟化驱动程序[12]跟踪客户机进程运行,利用推断技术识别 I/O 受限进程。在 I/O 受限进程识别的基础上,vSMP Internal 调度算法对客户机内部 vCPU 进行分类,当与 I/O 受限进程关联的 I/O 事件到来后,通过对不同类别 vCPU 之间物理 CPU 资源的切换提高 I/O 受限进程的性能。

首先阐述客户机 I/O 受限进程的识别方法,然后在此基础上描述 vSMP Internal 调度算法以及 I/O 事件与 I/O 受限进程的关联方法。

1.1 I/O 受限进程识别

VMM 缺乏客户机负载的认知导致 VMM 调度 vCPU 的困难,所以客户机进程的识别技术是实现有效 vCPU 调度机制的前提。识别 I/O 受限进程比较直接的方法是客户机操作系统将 I/O 受限进程的信息通过超调用发送给 VMM。另外一种比较常见的技术是推断技术,在 VMM 层获得客户机进

程信息，根据进程信息来推断进程类型。超调用的引入需要修改客户机操作系统，同时频繁的超调用会增加性能开销。VMM 层的推断技术不用修改客户机操作系统，对客户机透明并且可扩展性强。然而，VMM 只能对有限的客户机 I/O 操作以及特权指令进行监控[13]，难以获得有助于识别进程类型的详细信息，不能准确地识别 I/O 受限进程。I/O 受限进程的识别作为 vCPU 调度的基础，错误的识别不但不会改善 I/O 性能，反而会导致系统性能的严重下降。

综合以上两种方法，本文提出一种在客户机内部识别 I/O 受限进程的方法。该方法为客户机操作系统加载一个半虚拟化驱动程序，驱动程序跟踪进程的运行，利用推断技术识别客户机 I/O 受限进程。驱动程序通过内存映射的方式与 VMM 共享 I/O 受限进程的信息。驱动程序根据 I/O 受限进程如下两个特点来推断 I/O 受限进程：①I/O 受限进程大部分时间是等待设备完成 I/O 请求，消耗很少的 CPU 时间；②I/O 受限进程因为 I/O 请求不能立即完成而被阻塞，主动调用进程调度程序来释放 CPU 资源。驱动程序监控操作系统进程调度程序的运行，根据调用进程调度函数的进程是否满足以上特点作为判断 I/O 受限进程的依据。客户机加载驱动程序的方式避免了对客户机操作系统内核的修改，同时利用内存映射的方式共享信息相比超调用减少开销。驱动程序通过监控客户机进程的运行来获得进程详细的信息，可以准确地识别 I/O 受限进程。

1.2　vSMP Internal 调度算法

在 vSMP Internal 调度算法中，将 vSMP 客户机中的 vCPU 分为以下 3 类：①处理 I/O 中断的 vCPU（简称 Intr-vCPU）；②运行 I/O 受限进程的 vCPU（简称 IO-vCPU）；③不处理 I/O 中断，也没有运行 I/O 受限进程的 vCPU（简称 Victim-vCPU）。在传统调度方法下，客户机 I/O 请求的处理过程如图 1 所示。T_1 时刻 IO-vCPU 中的 I/O 受限进程对设备发出请求，该请求最终被发送给 I/O thread 进行异步处理。T_2 时刻 I/O thread 完成 I/O 请求并通知 Main thread。T_3 时刻 Main thread 生成虚拟设备中断并挂起到 Intr-vCPU 中，通知 I/O 请求完成。T_4 时刻 Intr-vCPU 得到运行机会，调用中断处理程序处理中断，并在 T_5 时刻通知 IO-vCPU 中 I/O 受限进程请求数据到来。T_6 时刻 IO-vCPU 得到运行机会，调度 I/O 受限进程处理请求的数据，完成此次 I/O 请求。

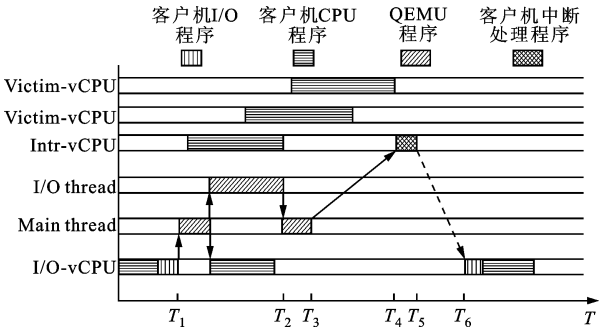


图 1　混合负载中 vCPU 处理 I/O 请求过程

显然如下两方面延迟因素影响 I/O 受限进程的性能：①I/O 事件对应的虚拟 I/O 中断到达 Intr-vCPU 后到 Intr-vCPU 开始运行中断处理程序之间的延迟（T_3-T_4）；②Intr-vCPU 处理完虚拟 I/O 中断后到 IO-vCPU 中 I/O 受限进程开始运行之间的延迟（T_5-T_6）。这两种延迟导致 I/O 受限进程等待时间增多，影响 I/O 受限进程的响应时间和吞吐率。单一负载情况下，Intr-vCPU 以及 IO-vCPU 经常处于空闲状态，VMM 会优先调度 Intr-vCPU 和 IO-vCPU 运行。在混合负载的情况下，Intr-vCPU 和 IO-vCPU 中的 CPU 受限进程消耗掉 VMM 分配给它们的 CPU 资源，导致 VMM 调度其他进程抢占 Intr-vCPU 和 IO-vCPU。

为了减少混合负载情况下这两种延迟，当 I/O 事件到来后，需要给 Intr-vCPU 和 IO-vCPU 更多的运行机会。vSMP Internal 调度算法过程如图 2 所示，当 I/O 事件对应的虚拟 I/O 中断挂起到 Intr-vCPU 后，如果 Intr-vCPU 没有在物理 CPU 上运行或者在很长的一段时间之后才会运行，则立即调度 Intr-vCPU 来完成虚拟 I/O 中断处理，以减少第一种延迟。同时，在 Intr-vCPU 完成虚拟 I/O 中断处理之后，为了减少第二种延迟也要立即调度 IO-vCPU 运行，I/O 受限进程才能处理数据并完成 I/O

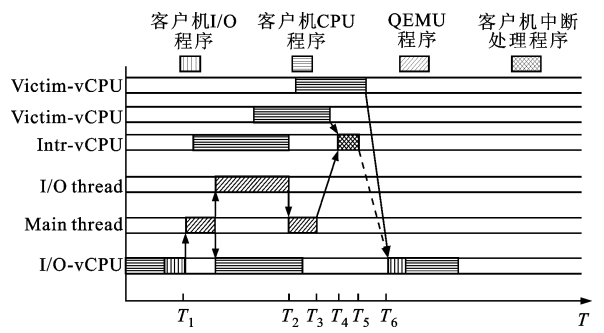


图 2　vSMP Internal 调度过程

请求。为了保证整个系统的公平性，vSMP Internal 调度算法牺牲 vSMP 客户机中 Victim-vCPU 的运行机会，Intr-vCPU 和 IO-vCPU 通过抢占 Victim-vCPU 的运行时间来立即调度到物理 CPU 上运行。这样当 I/O 事件到来后能够及时调度这两种 vCPU 运行，提高 I/O 受限进程的响应性能和吞吐率。

1.3 I/O 事件与 I/O 受限进程的关联方法

vSMP Internal 算法中为了提高客户机 I/O 负载性能，每当有 I/O 事件到达 vCPU 时，就强制调度对应的 Intr-vCPU 和 IO-vCPU 运行，但是由于某些非 I/O 受限进程也会执行 I/O 请求，即并不是每个 I/O 事件对应的进程都是 I/O 受限进程。这可能导致对非 I/O 受限进程对应的 I/O 事件做出错误的响应，影响真正需要改善的 I/O 受限进程的性能以及系统公平性。因此，为了获得低的响应延迟以及整个系统的公平性，需要实现 I/O 事件与 I/O 受限进程的关联。

设 I/O 事件集合为集合 A，I/O 受限进程集合为集合 C，A 和 C 为两个非空集合。为了实现 I/O 事件与 I/O 受限进程的关联，需要满足 $\forall a \in A$，存在唯一的 $c \in C$，使得 $(a,c) \in h$，那么 h 为从 A 到 C 的函数，记为 $h:A \rightarrow C$，由此问题转化为求从 A 到 C 的函数 h。语义缝隙的问题导致 VMM 无法获得客户机内部进程的详细信息，所以无法在 VMM 中直接实现关联函数 h。但是，如果存在一个媒介集合 B，并且可以找到两个函数 f 和 g，使得 f 是从 A 到 B 的函数(记为 $f:A \rightarrow B$)，g 是从 B 到 C 的函数(记为 $g:B \rightarrow C$)，那么，复合关系 $f \circ g$ 就是从 A 到 C 的函数(记为 $g \circ f$)。因此，为了得到 $h:A \rightarrow C$，需要找到媒介集合 B，并且实现 $f:A \rightarrow B$ 以及 $g:B \rightarrow C$，如图 3 所示。

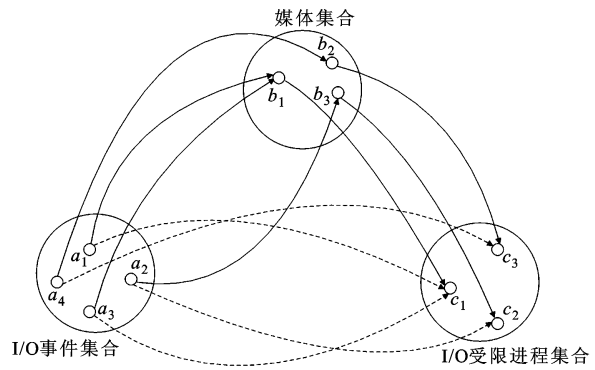


图 3 I/O 事件与 I/O 受限进程关联

媒介集合的选取是实现关联的基础，不同类型的设备需要选取不同的媒介集合来实现该类型设备发出的 I/O 事件与访问该设备的 I/O 受限进程之间的关联。设备模拟程序负责完成客户机 I/O 请求并产生 I/O 事件，因此可以很容易地在 VMM 层实现函数 f。客户机可以通过监控 I/O 受限进程与 I/O 设备的交互来实现函数 g。这种 VMM 与客户机合作的方法，可以有效地实现 I/O 事件与 I/O 受限进程的关联。

2 vCPU 调度机制实现

本节讲述基于 I/O 受限进程识别的 vCPU 调度机制在 KVM(Kernel-based Virtual Machine)[14] 虚拟化环境中的具体实现。宿主机与客户机操作系统均为 Centos 5.8，内核版本 3.10.21。KVM 版本为 kvm-kmod-3.10.21，模拟处理器的自由软件 QEMU 版本为 qemu-1.2.0。

图 4 为系统的结构框图，IO-Track 是客户机操作系统中一个动态可加载模块，负责维护 IOHash 表、BlockHash 表以及 NetHash 表。IO-Track 使用 IOHash 表实现 I/O 受限进程的识别，同时使用 BlockHash 表以及 NetHash 表实现媒介集合到 I/O 受限进程集合的函数 g；QEMU 中的后端驱动程序实现 I/O 事件集合到媒介集合的函数 f。vSMP Internal 利用 CFS 的相关特性通过对客户机 vCPU 的切换来提高客户机中 I/O 受限进程的性能。

2.1 I/O 受限进程识别

为了管理客户机内 I/O 受限进程信息，IO-Track 为每个客户机维护一个 I/O 受限进程哈希表(IO-Hash)。IOHash 表以进程标识符(PID)作为键值，表项为 io_info 结构体，该结构体记录了 I/O 受限进程的 PID 以及所属 vCPU 的标识符(vCPUID)。IO-Track 利用 kprobe[15] 内核探测机制在内核进程调度函数 asmlinkage void_sched schedule(void)的入口插入 kprobe 类型的探测点。探测点函数如算

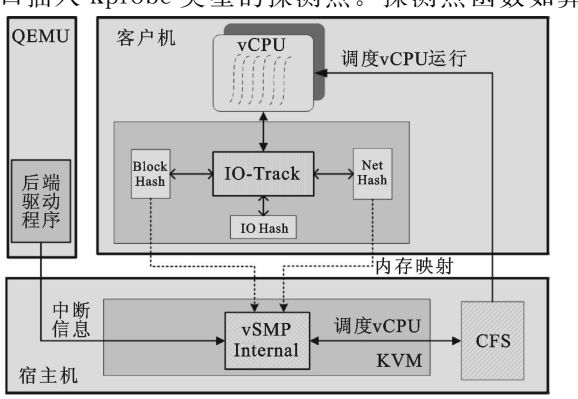


图 4 vSMP Internal 调度系统框图

法 1 所示，根据当前进程是否满足 I/O 受限进程的特点来识别 I/O 受限进程。

算法 1　识别 I/O 受限进程

输入　进程描述符 Task，哈希表 IOHash

```
function KprobeSchedule
 Threshold←Hz/2
 io_info←HashFind(Task.pid,IOHash)
 if Task.state=WaitState and Task.runtime
<Threshold then
  if io_info≠NIL then
   io_info.vcpuid=Task.cpuid
  else
   io_info←Create(Task.pid,IOHash)
   HashAdd(Task.pid,IOHash,io_info)
  end if
 else if io_info≠NIL then
  HashDel(Task.pid,IOHash)
 end if
end function
```

如果进程处于等待状态，说明进程是因不能获得必须的资源而主动调用进程调度函数来释放 CPU 资源。Threshold 代表 I/O 受限进程运行时间的阈值，I/O 受限进程由于频繁地发出 I/O 请求并等待请求的完成，所以运行时间一般小于 Threshold。由于某些进程只在一段时间内频繁地请求 I/O，所以探测点函数需要更新 IOHash 表，将不再满足 I/O 受限进程特点的进程从 IOHash 表中删除。

2.2　I/O 事件与 I/O 受限进程的关联

本节描述客户机与 QEMU 后端驱动程序合作实现块设备读和网络数据包接收这两类对延迟敏感的 I/O 事件与 I/O 受限进程的关联。

2.2.1　*块设备读*　一个块设备的 I/O 中断唯一对应一个磁盘存储区标识符，且由于高速缓存的存在，不会出现两个 I/O 受限进程对同一个磁盘存储区标识符发起读请求操作。因此，一个磁盘存储区标识符也唯一对应一个 I/O 受限进程，磁盘存储区标识符集合可以作为块设备的 I/O 事件集合与对块设备进行读操作的 I/O 受限进程集合的媒介集合。

在虚拟化环境中，I/O 事件对应的是后端驱动程序发送给 vCPU 的虚拟中断。为了实现 I/O 事件集合到磁盘存储区标识符集合的函数 $f:A\rightarrow B$，修改块设备后端驱动的中断注入函数，将虚拟中断对应的磁盘存储区标识符信息与虚拟中断一起发送给 VMM。

IO-Track 为每个客户机维护了一个 BlockHash 表，VMM 通过内存映射的方式访问每个客户机 BlockHash 表。哈希表项为 block_info 结构，包含起始扇区号、扇区数和块设备 I/O 受限进程所在 vCPU 标识符(vCPUID)，BlockHash 表以块请求的起始扇区号作为键值。BlockHash 表实现磁盘存储区标识符集合到 I/O 受限进程集合的函数 $g:B\rightarrow C$。IO-Track 利用 kprobe 内核探测机制，对内核函数 void blk_queue_bio(struct request_queue * q, struct bio * bio)添加 jprobe 探测点，根据函数的参数 bio 结构体获得此次 I/O 请求的磁盘存储区标识符。探测点函数根据 bio 结构体的 bi_rw 字段判断是否是来自文件系统的块设备读请求，通过 IO-Hash 表判断发出请求的进程是否为 I/O 受限进程，如果当前进程为 I/O 受限进程，构建 block_info 结构并插入到 BlockHash 表中。

2.2.2　*网络数据包接收*　每个网络数据包到来的 I/O 事件唯一地对应一个(传输层协议，目的端口号)二元组，对于同一个 IP 地址的机器，该二元组对应一个唯一网络 I/O 受限进程，所以该二元组集合可以作为网络数据 I/O 事件集合与网络数据包接收进程集合的媒介集合。

在虚拟化环境中，网络数据包到达的 I/O 事件对应的是网络设备后端驱动程序发送给 vCPU 的虚拟中断，网络设备后端驱动程序根据接收到的网络数据包的包头获得使用的传输层协议和目的端口号。为了实现网路数据包 I/O 事件集合到两元组集合的函数 $f:A\rightarrow B$，需要修改网络设备后端驱动的中断注入函数，将数据包对应的传输层协议和目的端口号与虚拟中断一起发送给 VMM。

IO-Track 为每个客户机维护了一个以端口号为键值的 NetHash 表，VMM 通过内存映射方式访问每个客户机 NetHash 表。哈希表项为 net_info 结构，包含传输层协议(protocol)、端口号(port)以及网络 I/O 受限进程所在 vCPU 的标识符(vCPUID)。NetHash 表实现(传输层协议，端口号)两元组集合到网络数据包接收 I/O 受限进程集合的函数 $f:B\rightarrow C$。TCP 协议的服务器程序调用内核函数 struct sock * inet_csk_accept(struct sock * sk,int flags, int * err)监听来自客户端的连接请求。TCP 和 UDP 协议的服务器程序统一调用内核函数 int sock_recvmsg(struct socket * sock,struct msghdr * msg,size_t size,int flags)接收客户端数据。IO-

Track 模块利用 kprobe 内核探测机制对以上两个函数添加 jprobe 探测点，探测点函数首先通过 IO-Hash 判断当前进程是否为 I/O 受限进程，如果当前进程为 I/O 受限进程，根据 sock 结构体获得传输层协议以及端口号，构建 net_info 结构并插入到 NetHash 表中。

2.3 vSMP Internal 调度

本节主要描述 vSMP Internal 调度流程，以及如何利用 CFS 特性实现指定 vCPU 之间的切换。

2.3.1 vSMP Internal 调度流程 当 I/O 事件对应的虚拟 I/O 中断挂起到 vCPU 时，VMM 利用 I/O 事件与 I/O 受限进程的关联方法判断是否启动如算法 2 所示的 vSMP Internal 调度。

算法 2 vSMP Internal 调度

输入 intr_vcpu，io_vcpu

```
function vSMPInternal(intr_vcpu,io_vcpu)
 if intr_vcpu.preempted=True then
  vic_vcpu←VictimFind(intr_vcpu.pcpu)
  vCPUSwitch(intr_vcpu,vic_vcpu)
 end if
 if io_vcpu≠intr_vcpu and io_vcpu.preempted
=True then
  vic_vcpu←VictimFind(io_vcpu.pcpu)
  vCPUSwitch(io_vcpu,vic_vcpu)
end if
end function
```

vSMP Internal 调用 VictimFind 时，将 Intr-vCPU 或 IO-vCPU 的所在物理 CPU 的 ID(pcpu)作为参数传递给 VictimFind，使得 VictimFind 尽量查找位于 pcpu 的 Victim-vCPU，防止 vCPUSwitch 影响其他客户机的运行。同时，为了防止某一个 vCPU 被频繁地当作 Victim-vCPU 而被抢占，vSMP Internal 调度为每个客户机记录上次作为 Victim-vCPU 的下标 last_vic，VictimFind 从 last_vic 开始轮转查找占用物理 CPU 的 Victim-vCPU。

2.3.2 vCPUSwitch 实现 Linux 内核将每个 CPU 中运行队列与 CFS 调度器[16]相关的字段组合成一个新的数据结构 CFS 运行队列(struct cfs_rq)。cfs_rq 的 next 字段代表要优先调度的进程，skip 字段代表不应该被调度的进程。CFS 在选取下一个被调度运行的进程时优先选择 next 字段代表的进程并且不选择 skip 字段代表的进程运行。因此，vCPUSwitch 将 cfs_rq 的 skip 字段设置为 Victim-vCPU，next 字段设置为要与 Victim-vCPU 切换的 vCPU，然后调用进程调度函数实现 vCPU 之间的切换。

3 测 试

测试环境中宿主机配置见表 1。

表 1 宿主机软硬件配置

配置描述	配置参数
机型	Inspur NF5588
CPU	Intel Xeon E5620 @ 2.40 GHz
内存	4 GB
磁盘	500 GB SATA
网卡	Intel 82574L Gigabit
OS	Centos 5.8 内核版本 3.10.21
KVM 版本	kvm-kmod-3.10.21
QEMU 版本	qemu-1.2.0

网络测试中客户端运行在相同配置的物理机器上，两台物理机器通过华为 Quidway S5328C-SI 千兆交换机相连。在宿主机上同时运行了 5 个客户机，每个客户机配备 8 个 vCPU 以及 1 GB 内存，客户机操作系统为 Centos 5.8，Linux 内核版本为 3.10.21，客户机网络设备为 Intel e1000 网卡，块设备为 IDE 硬盘。以一台同时运行 I/O 受限任务和 CPU 受限任务的客户机作为评估对象，另外 4 台客户机只运行 CPU 受限任务。所有的测试结果是 5 次测试的平均值，在本文测试结果中，I/O 代表测试客户机只运行单一的 I/O 负载，I/O+CPU 代表客户机同时运行 I/O 和 CPU 混合负载，Original 代表 KVM 原 CFS 调度机制，vSMP Internal 代表本文实现机制。

3.1 网络性能测试

在虚拟化环境中，客户机运行 I/O 密集型的网络服务器对外提供服务，TCP 协议能够提供端到端的可靠传输，被大量网络应用程序使用。本文通过 Netperf 工具对 vSMP 客户机 TCP 数据传输吞吐率进行测试，评估本文提出的调度机制对 I/O 性能的提升。为了模拟多种真实的 TCP 网络传输场景，采用 3 种测试模式进行批量数据传输和请求应答吞吐率测试，实验结果如图 5 所示。其中 TCP_STREAM 模式模拟 FTP 等一次传输整个文件的批量数据传输的网络应用，TCP_RR 模式模拟一个 TCP 连接传送多次交易的数据库应用，TCP_CRR 模式模拟最典型的一次交易在一条单独的 TCP 连接的 HT-

TP 应用。图 5 中是相对于客户机单一 I/O 负载性能的归一化表示，vSMP Internal 相比 Original 在 3 种模式上都有显著提高，并且接近客户机单一 I/O 负载情况下的性能。图 6 显示的是 vSMP Internal 测试中各个客户机对物理 CPU 的使用率，测试结果显示客户机对物理 CPU 的使用率大致相同，因此可以保证客户机之间的公平性。

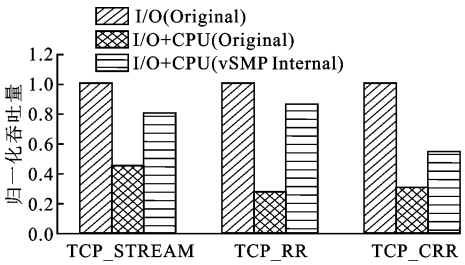
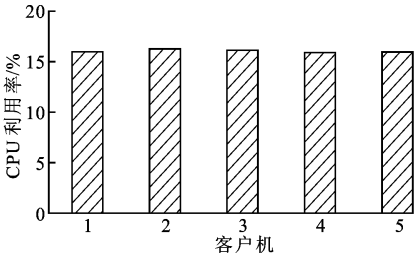


图 5　不同模式的网络测试对比



客户机 1～4：CPU；客户机 5：IO+CPU

图 6　vSMP Internal 网络测试中物理 CPU 使用情况

3.2　IO-Track 开销测试

本文提出的基于 I/O 受限进程识别的 vCPU 调度机制的开销主要来自客户机中实现 I/O 受限进程的识别，以及 I/O 事件与 I/O 受限进程的关联，即 IO-Track 带来的开销。为了证明本文提出的调度机制的轻量化，利用 IOzone 和 SuperPI 测试工具分别测试 IO-Track 对 I/O 负载和 CPU 负载的额外开销。为了防止 vSMP Internal 对 vCPU 调度导致 I/O 负载测试的不准确性，在所有测试中禁止 vSMP Internal。

IOzone 测试工具对磁盘多种不同模式的读写进行测试，可以充分评估 IO-Track 对磁盘 I/O 的影响。如图 7 所示，在磁盘 I/O 负载测试中，磁盘写操作会带来 1%～2%的性能开销，磁盘读操作会带来 5%～8%的性能开销。写操作的开销主要来自 IO-Track 中内核 schedule()函数的 kprobe 探测点的运行，而读操作的开销除了来自 schedule()函数的 kprobe 探测点以外，还来自 blk_queue_bio()内核函数 jprobe 探测点的运行，因此读磁盘的开销要大于写磁盘的开销。

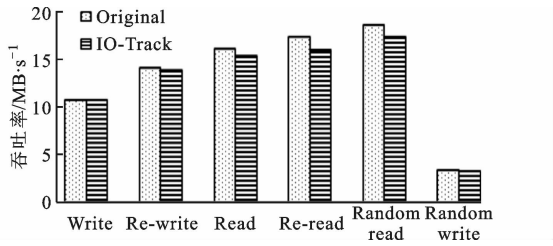


图 7　IOzone 测试的磁盘读写速度对比

Super PI 测试结果如图 8 所示，其中时间越短说明性能越好。测试结果显示 IO-Track 对 CPU 负载带来的开销为 0.5%，这主要来自 IO-Track 中 schedule()函数 kprobe 探测点的运行。运行 CPU 负载的情况下，schedule()函数的运行次数要明显少于运行 I/O 负载的情况，因此 IO-Track 对 CPU 负载带来的开销要明显少于 I/O 负载。

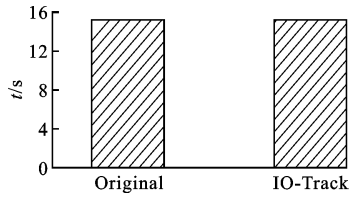


图 8　Super PI 基准测试数据对比

4　结　论

本文通过客户机与 VMM 合作实现基于 I/O 受限进程识别的 vCPU 调度机制。该机制在 I/O 受限进程与 I/O 事件关联的基础上，利用 vSMP Internal 调度算法桥接客户机与 VMM 之间的语义缝隙。测试结果表明，在保证客户机之间公平性的前提下，提高了运行混合负载的 vSMP 客户机中 I/O 负载性能，因此本文提出的 vCPU 调度机制可以应用在运行多样性和不可预测性负载的虚拟桌面和云计算环境中。

参考文献：

[1]　金海，钟阿林，吴松，等. 多核环境下虚拟机 VCPU 调度研究：问题与挑战 [J]. 计算机研究与发展，2011，48(7)：1216-1224.
JIN Hai，ZHONG Alin，WU Song，et al. Virtual machine VCPU scheduling in the multi-core environment：issues and challenges [J]. Journal of Computer Research and Development，2011，48(7)：1216-1224.

[2]　CHEN P M，NOBLE B D. When virtual is better than real [C]// Proceedings of the 8th Workshop on Hot Topics in Operating Systems. Piscataway，NJ，USA：IEEE，2001：133-138.

[3] SONG Xiang, SHI Jicheng, CHEN Haibo, et al. Schedule processes, not VCPUs [C]// Proceedings of the 4th Asia-Pacific Workshop on Systems. New York, USA: ACM, 2013: 1-7.

[4] XU Fei, LIU Fangming, JIN Hai, et al. Managing performance overhead of virtual machines in cloud computing: a survey, state of the art, and future directions [J]. Proceedings of the IEEE, 2014, 102(1): 11-31.

[5] BARHAM P, DRAGOVIC B, FRASER K, et al. Xen and the art of virtualization [C]// Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York, USA: ACM, 2003: 164-177.

[6] KIM H, LIM H, JEONG J, et al. Transparently bridging semantic gap in CPU management for virtualized environments [J]. Journal of Parallel and Distributed Computing, 2011, 71(6): 758-773.

[7] UHLIG R, SMITH L, NEIGER G, et al. Intel virtualization technology [J]. Computer, 2005, 38(5): 48-56.

[8] Advanced Micro Devices. AMD64 virtualization codenamed "Pacifica" technology: secure virtual machine architecture reference manual [M]. Sunnyval, CA, USA: AMD, 2005: 49-51.

[9] KIM H, KIM S, JEONG J, et al. Virtual asymmetric multiprocessor for interactive performance of consolidated desktops [C]// Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. New York, USA: ACM, 2014: 29-40.

[10] XU Cong, GAMAGE S, LU Hui, et al. vTurbo: accelerating virtual machine I/O processing using designated turbo-sliced core [C]// Proceedings of the 2013 USENIX Conference on Annual Technical Conference. Berkeley, CA, USA: USENIX, 2013: 243-254.

[11] CHENG Luwei, WANG Choli. vBalance: using interrupt load balance to improve I/O performance for SMP virtual machines [C]// Proceedings of the Third ACM Symposium on Cloud Computing. New York, USA: ACM, 2012: 1-14.

[12] RUSSELL R. Virtio: towards a de-facto standard for virtual I/O devices [J]. ACM SIGOPS Operating Systems Review: Research and Developments in the Linux Kernel, 2008, 42(5): 95-103.

[13] ADAMS K, AGESEN O. A comparison of software and hardware techniques for x86 virtualization [C]// Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM, 2006: 2-13.

[14] KIVITY A, KAMAY Y, LAOR D, et al. KVM: the Linux virtual machine monitor [J]. Proceedings of the Linux Symposium, 2007, 1: 225-230.

[15] KRISHNAKUMAR R. Kernel korner: kprobes-a kernel debugger [J]. Linux Journal, 2005, 2005(133): 11.

[16] MOLNAR I. Modular scheduler core and completely fair scheduler [EB/OL]. (2007-04-13) [2014-06-26]. http://lwn.net/Articles/230501.

[本刊相关文献链接]

刘强,董小社,朱正东,等.一种短作业环境下的延迟调度算法.2015,49(2):1-5.[doi:10.7652/xjtuxb201502001]

樊源泉,伍卫国,许云龙,等.MapReduce环境中的性能特征能耗估计方法.2015,49(1):14-19.[doi:10.7652/xjtuxb201501003]

丑文龙,梅魁志,高增辉,等.ARM GPU的多任务调度设计与实现.2014,48(12):87-92.[doi:10.7652/xjtuxb201412014]

郑鹏飞,尤佳莉,王劲林,等.一种多租户云的内部网络共享策略.2014,48(8):54-59.[doi:10.7652/xjtuxb201408010]

马莉,唐善成,王静,等.云计算环境下的动态反馈作业调度算法.2014,48(7):77-82.[doi:10.7652/xjtuxb201407014]

张忆文,郭锐锋.硬实时系统周期任务低功耗调度算法.2014,48(7):90-95.[doi:10.7652/xjtuxb201407016]

董皎皎,马瑞瑞,翟桥柱,等.多类型煤炭海运运输库存管理一体化模型.2014,48(6):37-42.[doi:10.7652/xjtuxb201406007]

汝海,高峰,徐寅峰,等.单水库汛期分段线性调度的在线策略与分析.2014,48(2):99-105.[doi:10.7652/xjtuxb201402017]

王兆杰,高峰,翟桥柱,等.高耗能企业关口平衡问题的双目标规划模型.2013,47(8):26-32.[doi:10.7652/xjtuxb201308005]

庄威,桂小林,林建,等.云环境下基于多属性层次分析的虚拟机部署与调度策略.2013,47(2):28-32.[doi:10.7652/xjtuxb201302005]

李健,黄庆佳,刘一阳,等.云计算环境下的大规模图状数据处理任务调度算法.2012,46(12):116-122.[doi:10.7652/xjtuxb201212020]

杜文超,陈庶樵,胡宇翔.面向网络流的自适应正则表达式分组匹配算法.2012,46(8):49-53.[doi:10.7652/xjtuxb201208009]

(编辑 武红江)