软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(2):398−410 [doi: 10.13328/j.cnki.jos.005059]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种灵活高效的虚拟 CPU 调度算法*

刘珂男, 童 薇, 冯 丹, 刘景宁, 张 炬

(武汉光电国家实验室(华中科技大学),湖北 武汉 430074)

通讯作者: 童薇, E-mail: weitong@163.com

摘 要: 目前,虚拟化已经广泛应用于数据中心,但主流的虚拟 CPU 调度策略并没有实现对 I/O 性能的保障,尤其是当延时敏感型负载的虚拟机和计算敏感型负载的虚拟机竞争 CPU 资源时,其性能显著下降.针对上述问题,提出了一种灵活、高效的虚拟 CPU 调度算法 FLMS(flexible I/O latency and multi-processor sensitive scheduler).FLMS 通过采用虚拟机分类、虚拟 CPU 绑定、多类时间片等技术降低了虚拟机的响应延时,同时基于多处理器架构重新设计了负载均衡策略,优化了虚拟 CPU 迁移.FLMS 通用于目前主流的虚拟化方案,在软件虚拟化方式下,与最新的优化方案相比,延时降低了 30%,带宽有 10%的提升;在使用硬件辅助虚拟化的系统中,通过FLMS能够获得接近原生系统的 I/O 性能,并且保证了整个系统的公平性.

关键词: I/O 虚拟化;设备直接分配;虚拟 CPU 调度

中图法分类号: TP316

中文引用格式: 刘珂男,童薇,冯丹,刘景宁,张炬.一种灵活高效的虚拟 CPU 调度算法.软件学报,2017,28(2):398−410. http://www.jos.org.cn/1000-9825/5059.htm

英文引用格式: Liu KN, Tong W, Feng D, Liu JN, Zhang J. Flexible and efficient VCPU scheduling algorithm. Ruan Jian Xue Bao/ Journal of Software, 2017,28(2):398−410 (in Chinese). http://www.jos.org.cn/1000-9825/5059.htm

Flexible and Efficient VCPU Scheduling Algorithm

LIU Ke-Nan, TONG Wei, FENG Dan, LIU Jing-Ning, ZHANG Ju

(Wuhan National Laboratory for Optoelectronics (Huazhong University of Science and Technology), Wuhan 430074, China)

Abstract: At present, virtualization technology has been widely applied in data centers. However, VCPU (virtual CPU) scheduling strategy still faces intolerable I/O delay, especially for I/O-latency sensitive VMs which suffer from significant performance degradation when competing with CPU-intensive VMs. This paper presents a flexible and efficient VCPU scheduling algorithm FLMS (flexible I/O latency and multi-processor sensitive scheduler) which utilizes VM classification, VCPU binding and flexible slicer to reduce VM response delay. The work also redesigns the load balancing strategy to ensure optimal VCPU migration. FLMS is suitable for the current mainstream virtualization solutions. It has a 30% improvement comparing with the latest software virtualization. With hardware-assisted virtualization, FLMS makes it possible for VMs to achieve near bare-metal performance and ensures the fairness of the whole system

Key words: I/O virtualization; direct device assignment; VCPU (virtual CPU) scheduling

I/O 虚拟化[1]是虚拟化技术的一个核心部分,使得虚拟机[2]可以共享服务器 I/O 资源.目前,I/O 虚拟化技术的发展仍然相对滞后,出于安全性的考虑,虚拟机监控器[3]会干预虚拟机的 I/O 操作,同时,虚拟机之间的资源竞争均对 I/O 性能有负面影响,降低了虚拟化平台的整体性能.

* 基金项目: 国家高技术研究发展计划(863)(2015AA015301, 2015AA016701), 武汉市应用基本研究计划(2015010101010004)

Foundation item: National High-Tech R&D Program of China (863)(2015AA015301, 2015AA016701), Wuhan Applied Basic Research Project (2015010101010004)

收稿时间: 2015-06-24; 修改时间: 2015-09-08, 2015-10-30; 采用时间: 2016-03-19; jos 在线出版时间: 2016-05-03

CNKI 网络优先出版: 2016-05-04 08:44:15, http://www.cnki.net/kcms/detail/11.2560.TP.20160504.0844.008.html

© 中国科学院软件研究所 http://www.jos.org.cn

早期的虚拟化通过软件方式实现,分别是软件模拟方式和半虚拟化方式.前者用软件方式模拟设备的接口,这种方式下,虚拟机的所有 I/O 操作都被虚拟机监控器截获,性能损失比较严重.后者通过前后端驱动交互的方式与真实的硬件设备进行通信.由于后端和前端是一对多的关系,当虚拟机数目较多时,后端会成为系统的瓶颈.近年来随着越来越多对 I/O 性能有严苛要求的应用被迁移到虚拟化环境中,软件方式实现的虚拟化已无法满足应用需求.为了满足虚拟机的 I/O 性能需求,硬件辅助 I/O 虚拟化[4]被提出来,设备直接分配技术将一个物理 I/O 设备直接分配给一台虚拟机,使得虚拟机可以直接访问设备,I/O 性能得到了极大的提升.但一个物理设备只能分配给一台虚拟机,无法支持多台虚拟机共享.SR-IOV(single root I/O virtualization)[5]技术则通过虚拟功能(virtual function,简称 VF)支持资源共享,它可以创建多个 VF,直接分配给每台虚拟机,兼顾了性能与扩展性.然而,SR-IOV 的 I/O 处理仍然需要虚拟机监控器的干预,物理设备产生的中断需要由虚拟机监控器注入到虚拟机之中,产生上下文切换,导致其性能与原生系统相比仍有一定的差距.

目前已有很多针对 I/O 虚拟化性能优化的研究,其中优化虚拟 CPU(virtual CPU,简称 VCPU)调度算法是主要途径之一,但现有的方法仍存在一些问题,如缺乏灵活性[6]、应用场景单一[6−8]、破坏系统公平性[9]等.

虚拟机操作系统管理和访问的系统资源都是由虚拟机监控器提供的虚拟硬件资源,包括虚拟 CPU、虚拟内存和虚拟 I/O.当分配给虚拟机的虚拟 CPU 被调度到物理 CPU 上运行时,虚拟机操作系统才能运行虚拟机中的各个进程,因此,虚拟机性能与虚拟 CPU 调度密切相关.根据目前主流的虚拟机监控器可以将虚拟 CPU 调度算法分为两类,一类直接使用内核中的调度器并将虚拟 CPU 作为线程调度,如 KVM[10]使用的 $O(n)$,$O(1)$,CFS 算法;另一类虚拟机监控器专门设计了一套调度算法,如 Xen[11]使用的 BVT,SEDF,Credit 算法[12],其中 CFS 算法和 Credit 算法是目前的主流.

CFS 调度算法的核心思想是围绕着虚拟时间展开调度,每个进程有自己的虚拟时间,其增长速度与进程权重成反比,进程的权重对应进程的优先级.CFS 调度器为每个 CPU 分配一个按虚拟时间从小到大排序的红黑树结构,所有可运行进程都被插入红黑树之中,红黑树最左边的节点表示下一个被调度的进程.CFS 调度器在原生 Linux 环境中表现良好,但是在虚拟化环境中存在不足.(1) 在与计算型负载虚拟机共存的环境下,I/O 型负载虚拟机的 I/O 性能受损.因为红黑树根据虚拟时间排序,而运行 I/O 负载的进程被唤醒时所获得的虚拟时间往往不足以抢占当前运行的进程[13].(2) 多处理器虚拟机可能引发锁占用的问题[14].由于虚拟 CPU 是否正在持有锁对调度器是不可见的,如果一个正在持有自旋锁的虚拟 CPU 进入休眠,正在等待这个自旋锁的其他虚拟 CPU 则会因此需要等待更长的时间.

Credit 调度算法按照虚拟机权重共享 CPU 时间.Credit 调度算法的参数主要包括表示权重的 weight 和表示 CPU 时间上限的 cap.Credit 调度器根据每个虚拟机的 weight 值计算其应获得的信用值.cap 值限定其所能获得的信用值值上限.其 CPU 时间片分配由指定定时器完成,该定时器触发周期为 30ms.首先根据物理 CPU 个数计算系统总信用值,然后再根据每个虚拟机的权重和虚拟 CPU 个数计算虚拟机应获得的信用值,继而取信用值和 cap 中较小的一个作为虚拟机的信用值,再将虚拟机的信用值平均分配到每个虚拟 CPU.在 Credit 算法中,映射到每个物理 CPU 上的虚拟 CPU 会加入该物理 CPU 的运行队列中,并按照其优先级由高到低顺序排列.虚拟 CPU 的优先级共有 4 种,分别是 IDLE,OVER,UNDER 以及 BOOST.其中,IDLE 状态表示该虚拟 CPU 仅用于空闲时占位,具有最低的优先级;OVER 状态表示该虚拟 CPU 已消耗完 Credit 值并且无法被调度;UNDER 状态下的虚拟 CPU 按照队列中的先后顺序周期性地被调度;当虚拟 CPU 因事件到来被唤醒后会进入 BOOST 状态,BOOST 状态的虚拟 CPU 具有最高优先级,会被优先调度.当一个虚拟 CPU 消耗完其时间片时,会被重新插回运行队列中同等优先级虚拟 CPU 尾部.当一个运行队列空闲时,会从其余运行队列中寻找非 IDLE 的虚拟 CPU 并迁移调度.

与直接使用内核的调度算法相比,Credit 调度算法具有如下优势:(1) 它是一种能够通过设置虚拟机权重来按比例分享 CPU 时间的调度算法,特别适用于按需分配的云计算环境;(2) 其 Boost 机制支持及时抢占.这也是选择其作为研究对象的原因.

© 中国科学院软件研究所 http://www.jos.org.cn

1 虚拟 CPU 调度器面临的挑战

虚拟机在竞争 CPU 资源时,运行 I/O 型负载的虚拟机性能明显下降,其原因如下:首先,每次处理 I/O 请求仅消耗极短的 CPU 时间,使得虚拟机实际获得的 CPU 时间远远少于分配的时间,同时其调度延时取决于虚拟 CPU 数目以及时间片长短;其次,尽管负载均衡有利于充分利用 CPU 资源,但是负载均衡过程中频繁的虚拟 CPU 迁移所引入的缓存污染、上下文切换等额外开销会在很大程度上抵消负载均衡所带来的好处.

1.1 调度延时高且不稳定

虚拟 CPU 调度延时对虚拟机的 I/O 性能具有重要影响.目前的虚拟 CPU 调度器注重公平共享 CPU 时间,对所有的虚拟 CPU 采用简单的循环轮转算法,每次从可运行的运行队列中选取队列头,然后运行一定的时间,再插入到运行队列.因此,虚拟 CPU 的调度延时与运行队列中的可运行虚拟 CPU 数目以及时间片长短密切相关.

图 1 的测试显示了不同情况下虚拟机的 I/O 延时.测试分成 3 组,除了被测虚拟机 VM 1 以外,每组测试均另外创建 1 个~4 个虚拟机.第 1 组从同一局域网中另一台服务器对这些虚拟机进行 ping 测试;作为对比,第 2 组在除 VM 1 外的其他虚拟机中运行计算型负载 lookbusy,也对 VM 1 进行了 ping 测试.第 3 组的所有的虚拟机中均运行 lookbusy,并对 VM 1 进行了 ping 测试.在第 1 组实验中,随着运行队列中虚拟 CPU 数目的增长,往返延时几乎没有变化,这是由于虚拟机运行的是纯 I/O 负载,虚拟 CPU 运行时间都极短.然而,一旦其他虚拟机运行计算型负载,无论 VM 1 中运行的应用是纯 I/O 负载还是混合型负载,其应用延时都将随着虚拟机个数的增加而上升.

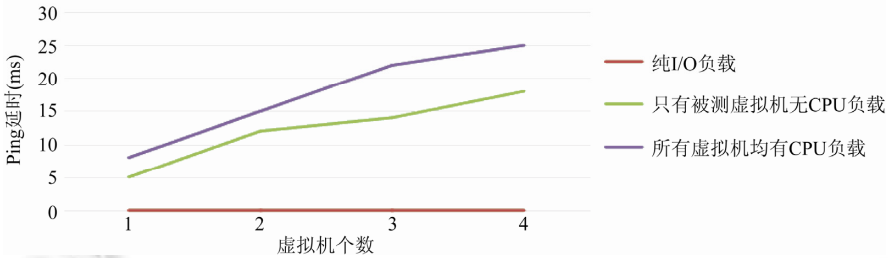


Fig.1 Latency of VM 1 changes with the number of VMs

图 1 虚拟机 1 的延时随虚拟机个数变化

1.2 负载均衡带来的性能损失

为了充分利用 CPU 资源,虚拟 CPU 调度器会在物理 CPU 空闲时从其他忙碌的物理 CPU 上迁移虚拟 CPU 来实现负载均衡.现在的计算机通常是多处理器架构.物理机通常具有多个物理封装(socket),一个物理封装具有多个物理核(core),在支持超线程的服务器中,一个物理核可以模拟两个逻辑处理器(logic processor).在多处理器系统中,虚拟 CPU 迁移的开销取决于源和目的物理 CPU 的相对位置.对于在同一物理核上的逻辑处理器之间的虚拟 CPU 迁移,由于源和目的物理 CPU 共享 L1 缓存,所以迁移开销较小;而对于不同物理封装之间的虚拟 CPU 迁移,高速缓存污染的开销则很大.因此,为了降低虚拟 CPU 迁移所带来的额外开销,一方面应尽量消除不必要的迁移,另外一方面必须要考虑多处理器架构对迁移开销的影响.自 4.3 版本开始,Credit 调度器加入了针对多处理器架构优化的负载均衡机制,大幅度消除了跨物理封装的虚拟 CPU 迁移.但其负载均衡策略仍有不足:(1) 粒度还不够细,物理封装内部仍然有物理核结构,如果能够尽量在同一个物理核中进行虚拟 CPU 迁移,其开销会更小;(2) 没有考虑目标虚拟 CPU 上运行的负载对迁移的影响.这也是很多研究工作都忽视的一个问题.

2 虚拟 CPU 调度算法优化相关工作

虚拟 CPU 调度算法一直是广受关注的研究热点,同时也是影响 I/O 虚拟化性能的一个重要因素.早期的虚拟 CPU 调度器没有考虑 I/O 负载的需求.2008 年,Ongaro 等人在虚拟 CPU 调度器中引入 BOOST 优先级和抢占机制来提高延时敏感型虚拟机的性能[15],当有剩余信用值的虚拟 CPU 从挂起状态被 I/O 事件唤醒时,则标记为

© 中国科学院软件研究所 http://www.jos.org.cn

BOOST优先级,并抢占物理CPU,更快地进行I/O处理.但当虚拟机中运行的是计算密集型负载或者混合负载时,虚拟CPU往往会耗尽时间片,不会达到触发BOOST优先级的条件.针对BOOST优先级的局限性,Ding等人在2013年提出了ECredit模型[7]:一种通过统计I/O请求数目来设置I/O优先级的改进算法.I/O优先级根据虚拟机的I/O请求的数目来进行设定,数目越大,I/O优先级越高,会排在运行队列前面.这种方案解决了BOOST的局限性问题,但是需要虚拟机监控器与虚拟机的通信合作,仅适用于半虚拟化环境.2013年,Zeng等人[16]考虑到I/O执行时间通常很短,将BOOST优先级的执行时间改为2ms,这样也避免了非I/O进程在BOOST优先级的虚拟CPU上执行时间过长.该方案针对BOOST优先级进行了优化,但是并未解决计算密集型负载与I/O密集型负载共存时的I/O性能问题.

另一种解决I/O响应延时的方案是修改时间片.最直观的方式就是缩短所有虚拟CPU的运行时间片长度[17],每一个虚拟CPU的调度延时都可以降低,I/O性能提升明显,但该方案同时也增加了上下文切换的发生频率.2012年,Xu等人提出了vSlicer方案[18],设计两类时间片来满足负载的特性:对运行I/O密集型负载的虚拟CPU设定很短的时间片;运行计算密集型负载的虚拟CPU仍然使用正常的时间片.如此就降低了整个系统的I/O响应时间,并且没有影响计算密集型负载的性能.但是,由于该方案没有考虑负载的隔离,当I/O密集型负载和计算密集型负载共享相同物理CPU时,I/O密集型负载仍需较长的排队时间,因此,该方案不适合混合型负载的虚拟机.作为对vSlicer模型的改进,Xu等人在2013年提出了多处理器环境下的vTurbo方案[6].该方案基于负载隔离与专用I/O核心的思想,预留1个或多个物理核作为运行时间片很短的专用I/O核心,并为每个虚拟机额外分配一个I/O型虚拟CPU来处理I/O中断,将I/O型虚拟CPU绑定到I/O核心上调度,其余的虚拟CPU不在I/O核心上调度并拥有正常的运行时间片.该方案解决了vSlicer不适合混合型负载的问题.但是,其I/O核心资源是静态预留的,不会随着I/O负载的变化而灵活分配,并且该方案需要修改虚拟机操作系统,仅适用于半虚拟化环境.

优化Credit调度器中的负载均衡策略也能实现I/O性能的提升.Rao等人在2013年提出了(bias random VCPU migration,简称BRM)算法[8].该方案对影响NUMA性能的因素,包括数据局部性、三级缓存竞争、数据共享这3个方面进行了综合测试和分析,据此设定每个虚拟CPU的迁移开销.然后根据迁移开销来决定虚拟CPU如何迁移.达到整个系统的迁移开销最小,以此提升I/O性能.但该方案需要修改虚拟机操作系统,仅适用于半虚拟化环境.2013年,Cheng等人[19]也指出,虚拟机访问远程NUMA节点比本地NUMA节点延时更高,影响I/O性能,但虚拟CPU调度器的负载均衡策略并未考虑NUMA架构的影响,针对该问题,Cheng等人提出了“最佳NUMA节点”方案,确保虚拟CPU被迁移到最合适的NUMA节点中调度.该方案有较高的缓存命中率,但是粒度还不够细.

针对目前的优化方案存在上述问题,本文实现了一种兼容软硬件虚拟化方式、低延时、灵活性好并且能保证系统公平性的虚拟CPU调度算法.

3 FLMS(flexible I/O latency and multi-processor sensitive scheduler)设计与实现

本文提出了一种灵活、高效的虚拟CPU调度算法(FLMS).FLMS通过采用虚拟机分类、虚拟CPU绑定、多类时间片等技术降低了虚拟机的响应延时,同时,基于多处理器架构重新设计了负载均衡策略,考虑了目标虚拟CPU上运行的负载对迁移的影响,优化了虚拟CPU迁移.FLMS不需要修改操作系统,通用于软硬件虚拟化方式,在软件虚拟化方式下与近年来提出的vTurbo方案相比,效果提升明显,在使用硬件辅助虚拟化的系统中,通过FLMS能够获得接近本地环境的I/O性能,并且保证虚拟机公平共享系统资源.

3.1 动态负载隔离

针对I/O延时敏感型的负载和计算密集型负载相互影响的问题,本文提出一种动态负载隔离方案.FLMS调度架构如图2所示.将虚拟机分成3种类型:延时敏感型虚拟机(latency-sensitive virtual machine,简称LSVM)、非延时敏感型虚拟机(non latency-sensitive virtual machine,简称NLSVM)和混合型负载虚拟机MixVM.对应地,将物理CPU分成两组:SP(short-slice PCPU)和LP(long-slice PCPU).将LSVM绑定到SPs上运行,给SPs设置较短的时间片,从而降低LSVM的调度延时,提升其调度频率;将NLSVM绑定到LPs上运行,给LPs设置较长的时间片,以保证较高的缓存命中率和较低的切换开销.对于MixVM,则需要对其虚拟CPU进行划分,将其运行I/O

© 中国科学院软件研究所 http://www.jos.org.cn

密集型负载的虚拟 CPU 绑定到 SPs 上,将运行计算密集型负载的虚拟 CPU 绑定到 LPs 上.MixVM 虚拟 CPU 划分原则如下:虚拟机启动时,静态预留指定数目的虚拟 CPU,缺省为 1 个,该虚拟 CPU 绑定到 SPs,其余虚拟 CPU 绑定到 LPs.所有进程默认绑定到该虚拟机非预留虚拟 CPU 上运行,当检测到虚拟机中有由用户创建并且 CPU 利用率极低的进程时,将其迁移到预留的虚拟 CPU 上运行.因为预留的虚拟 CPU 在空闲时不会分配物理 CPU 资源,因此并不会造成资源浪费.

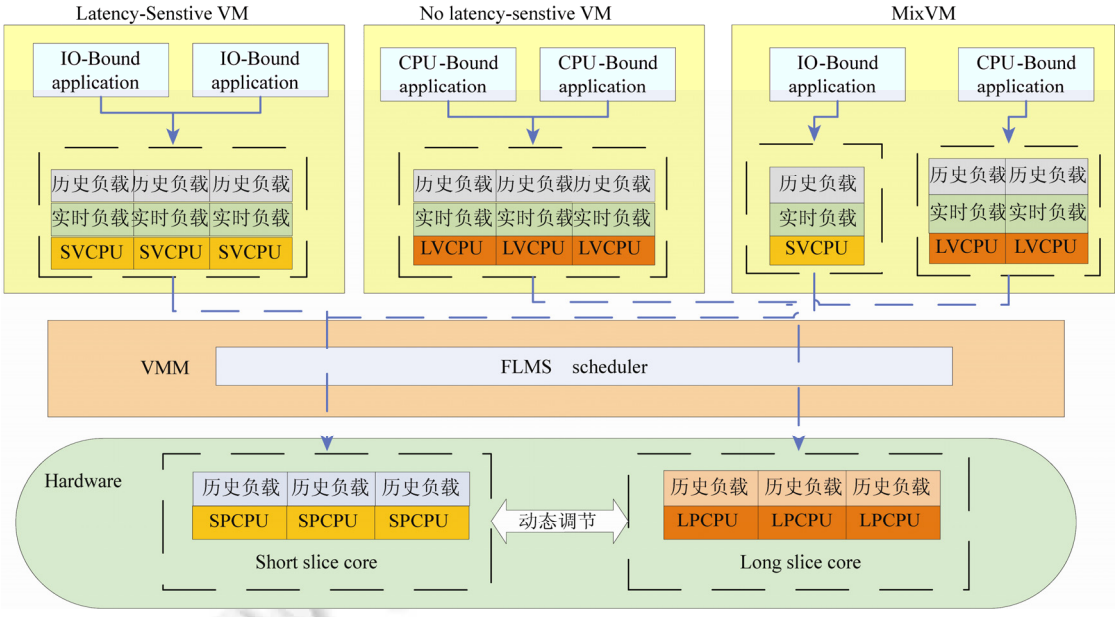


Fig.2 FLMS scheduling framework

图 2 FLMS 调度框架

FLMS 根据整个系统中的 I/O 密集型负载以及计算型负载的比重来动态分配 SPs 以及 LPs 的比例.其中,I/O 密集型负载统计量为一个周期内系统所有运行 I/O 型负载的虚拟 CPU 的运行时间之和,即 LSVM 和 MixVM 中运行 I/O 密集型负载的虚拟 CPU 的运行时间总和,而计算密集型负载统计量为 NLSVM 和 MixVM 中运行计算密集型负载的虚拟 CPU 的运行时间总和.由于要在当前周期内确定下一周期的 LPs 和 SPs 的数目,FLMS 需要根据历史负载信息来预测下一周期的负载状况.

假设物理 CPU 的个数为 PNum,SPs 和 LPs 的数目分别为 SPNum 和 LPNum,系统中的 I//O 密集型负载量和计算密集型负载量分别为 EstimatedIO 和 EstimatedCPU,当前周期内 I/O 密集型负载统计量和计算密集型负载统计量定义为 SampleIO 和 SampleCPU.可以通过以下公式计算 SPs 和 LPs 的数目,其中,α代表当前周期内的负载统计值对下一周期负载的影响因子,$0\leqslant\alpha\leqslant1$.

$$EstimatedIO=(1-\alpha)\cdot EstimatedIO+\alpha\cdot SampleIO \tag{1}$$

$$EstimatedCPU=(1-\alpha)\cdot EstimatedCPU+\alpha\cdot SampleCPU \tag{2}$$

$$SPNum=EstimatedIO\cdot PNum/(EstimatedIO+EstimatedCPU) \tag{3}$$

$$LPNum=PNum-SPNum \tag{4}$$

为了标识 SPs 以及 LPs 的物理 CPU,FLMS 使用了两个互补的 CPU 位图:ForIO 和 ForCPU.虽然也可以仅用 1 个位图,但由于调度器会频繁查询位图,将产生大量取反操作.而且只有在执行 CPU 分组动态调整时会同时更新两个物理 CPU 位图.系统初始化时,ForIO 设置为全 0,ForCPU 设置为全 1,表示系统初始默认运行计算密集型负载.在系统运行过程中,通过用户设定虚拟机类型,系统中 I/O 型负载与计算型负载的比重来更新两个物理 CPU 位图.

© 中国科学院软件研究所 http://www.jos.org.cn

采用这种调度结构的优点有如下两点.

(1) 性能改善.FLMS 实现了负载隔离,I/O 密集型负载获得了更高的调度频率与更少的排队等待时间,对于计算密集型负载,也保证了较少的上下文切换与较高的缓存命中率.

(2) 灵活性好.相对于专门分配一定数目的物理 CPU 作为 I/O 核心的方法而言,FLMS 根据系统负载比重对物理 CPU 分组,并定时根据负载变化调节两组物理 CPU 的数目,保证了系统的灵活性以及更高的资源利用率.

3.2 增强I/O优先级

在 Credit 算法中引入 Boost 机制的目的在于期望 I/O 事件到来时,目标 VCPU 可以及时抢占物理 CPU 资源.但是在 FLMS 的调度架构中,因为实现了负载的隔离,原有的抢占机制已不再适用,因此考虑采用一个新的变量来表示 I/O 任务繁重程度.对于运行 I/O 型负载的虚拟 CPU 而言,I/O 任务越重,被调度的次数就越多,其信用值消耗速度也越快;而 I/O 任务很轻的虚拟 CPU 会长期挂起,信用值的消耗很慢,因此,消耗信用值的多少可以作为衡量虚拟 CPU 的 I/O 任务繁重程度的指标.FLMS 在虚拟 CPU 调度结构增设一个字段 IO_Boost 来标识虚拟 CPU 的 I/O 繁重程度.IO_Boost 对于 NLSVM 的虚拟机是毫无意义的,可以忽略其虚拟 CPU 的 IO_Boost 值.IO_Boost 值值越小,表示该虚拟 CPU 的 I/O 任务越繁重,系统启动时初始值为 0,当动态负载隔离计时器触发,发生了 SPs 以及 LPs 类型的物理 CPU 数目调节时,也重新修改 IO_BOOST 的值,取值的依据也是各个虚拟 CPU 的历史负载.每当虚拟 CPU 被调度 1 次时,也适当增加其 IO_Boost 值,计算公式如下:

$$IO\_Boost=IO\_Boost+1/(totalCredit-remainCredit) \tag{5}$$

$$historyLoad=historyLoad+(totalCredit-remainCredit) \tag{6}$$

$$IO\_Boost=1/historyLoad \tag{7}$$

其中,totalCredit 表示在当前周期内虚拟 CPU 被分配的信用值以及上一周期剩余信用值之和,remainCredit 表示当前周期结束后剩余的信用值,当前周期到期后,通过将 totalCredit 减去虚拟 CPU 剩余信用值即可得到当前周期虚拟 CPU 消耗掉的信用值;historyLoad 字段用于表示虚拟 CPU 的历史负载,其值根据当前周期内消耗的信用值进行计算.

当虚拟 CPU 运行完成,重新插入运行队列时,也需要按照 IO_Boost 值从小到大的顺序,引入 IO_Boost 机制后,调度系统中 SPs 的物理 CPU 运行队列如图 3 所示.

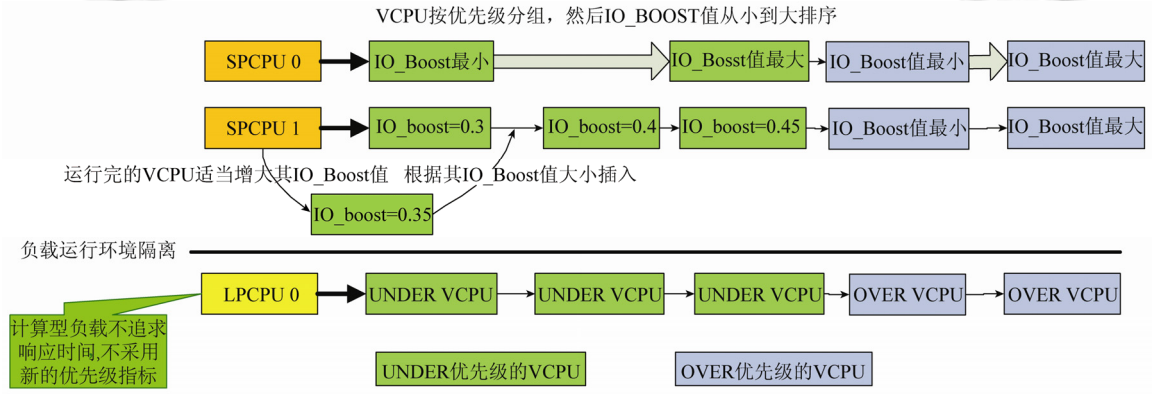


Fig.3 IO_Boost mechanism

图 3 IO_Boost 机制

3.3 高效负载均衡

目前负载均衡的优化方案大多关注目标物理 CPU 的选择,而忽略了目标虚拟 CPU 的选择对整个系统的影响.在 FLMS 中除了考虑到多处理器架构之外,还设计了目标虚拟 CPU 选择策略.

对于目标物理 CPU 的选择依然考虑多处理器架构的特点,同时还需要保证当 SPs 中的物理 CPU 触发负载

© 中国科学院软件研究所 http://www.jos.org.cn

均衡时不会影响到 LPs 中的物理 CPU,以避免计算密集型负载与 I/O 密集型负载之间相互干扰.当物理 CPU 空闲时,FLMS 根据下述迁移策略选择目标物理 CPU.

(1) 判断触发负载均衡的源物理 CPU 属于 SPs 还是 LPs,如果属于 SPs,则将 SPs 的其他所有物理 CPU 作为目标集合;否则将 LPs 中的所有物理 CPU 作为目标集合.

(2) 调度器遍历物理 CPU 目标集合中的物理 CPU,寻找是否存在与源物理 CPU 在同一个物理核中的物理 CPU,若存在,则将该物理 CPU 作为目标物理 CPU 并转(7);否则转(3).

(3) 调度器遍历物理 CPU 目标集合中的物理 CPU,寻找是否存在与源物理 CPU 在同一个物理封装上的物理 CPU,若存在,则找到该物理封装中负载最重的物理 CPU 作为待选物理 CPU,转(4);否则转(5).

(4) 若待选物理 CPU 负载减去源物理 CPU 负载超过阈值 A,则说明两者负载相差较大,此时将待选物理 CPU 作为目标物理 CPU,转步骤(7);否则,该物理封装上的所有物理 CPU 均不能作为目标物理 CPU,转(5).

(5) 调度器遍历物理 CPU 目标集合中剩余的物理 CPU,寻找负载最重的物理 CPU,作为待选物理 CPU.

(6) 若待选物理 CPU 的负载减去源物理 CPU 负载超过阈值 B,则说明两者负载相差较大,此时将待选物理 CPU 设置为目标物理 CPU,转(7);否则转(8).

(7) 在目标物理 CPU 上寻找合适的虚拟 CPU 并迁移,迁移结束.

(8) 若未找到合适的目标物理 CPU,则迁移结束.

其中,A 和 B 分别代表了在不同物理核与在不同物理封装之间进行虚拟 CPU 迁移的阈值.为了实现对物理 CPU 负载的统计,FLMS 需要在物理 CPU 调度数据结构中添加相关字段 historyBusy,currentBusy 以及一个定时器.currentBusy 用于表示一个计时器周期内产生的负载,由该物理 CPU 正在调度的虚拟 CPU 消耗的信用值累加而得,historyBusy 用于表示物理 CPU 的历史负载,其值根据当前周期内的历史负载和相对于当前周期的负载 currentBusy 进行计算.定时器用于每隔一定周期进行一次物理 CPU 历史负载记录 historyBusy 的更新.计算方法参考如下公式,其中,β表示历史负载权重因子,$0\leqslant\beta\leqslant1$.

$$historyBusy=historyBusy\cdot\beta+currentBusy\cdot(1-\beta) \tag{8}$$

选定了目标物理 CPU 之后还需要进一步选择目标虚拟 CPU.Credit 算法每次从运行队列头开始依次遍历,只要队列元素的优先级大于源物理 CPU 队首虚拟 CPU 的优先级就将其作为目标虚拟 CPU.这种策略未顾及到负载的特性,虽然简单但并不高效.FLMS 针对上述问题提出了目标虚拟 CPU 的选择策略.

FLMS 负载均衡机制如图 4 所示.如果触发负载均衡策略的源物理 CPU 属于 SPs,则说明当前需要迁移 I/O 密集型负载的虚拟 CPU,对于 I/O 任务而言,需要的是极短的 I/O 响应时间,因此选择 I/O 任务紧迫并且在等待调度的虚拟 CPU 进行迁移,此时,在目标物理 CPU 中选择 IO_Boost 值最小的虚拟 CPU 作为目标虚拟 CPU 执行迁移.如果触发负载均衡策略的源物理 CPU 属于 LPs,则说明当前需要迁移运行计算密集型负载的虚拟 CPU,对于计算密集型任务而言,对响应时间没有特别的依赖,所考虑的应该是迁移后源物理 CPU 与目标物理 CPU 的负载是否均衡,若选取的虚拟 CPU 负载过小,那么源和目的物理 CPU 的负载依然不均衡;反之,若迁移的虚拟 CPU 负载过重,则可能导致迁移后源物理 CPU 的负载超过目的物理 CPU,从而可能导致反复迁移.为了让源和目的物理 CPU 的负载尽量相等,应该选择令下面的等式成立的目标.

$$historyBusy_{\text{源物理 CPU}}+historyLoad_{\text{目标虚拟 CPU}}=historyBusy_{\text{目标物理 CPU}}-historyLoad_{\text{目标虚拟 CPU}} \tag{9}$$

FLMS 在 Xen-4.4.2 中实现,在物理 CPU 调度数据结构中增加实时负载与历史负载变量,并增加一个定时器来更新物理 CPU 的历史负载;在虚拟 CPU 调度数据结构中增加历史负载变量、统计信用值变化的变量以及表示虚拟 CPU 优先级的变量;在虚拟域调度数据结构中增加一个变量,表示其虚拟机类别;增加两个全局位图,表示两组物理 CPU,两个栈保存绑定在两组物理 CPU 位图上的虚拟 CPU;两个负载变量保存一个周期内系统中的总 I/O 型负载统计值与计算型负载统计值;一个全局定时器,设定动态调整的周期.FLMS 与默认的 Credit 调度器相比,增加的代码行数为 1 171 行.

© 中国科学院软件研究所 http://www.jos.org.cn

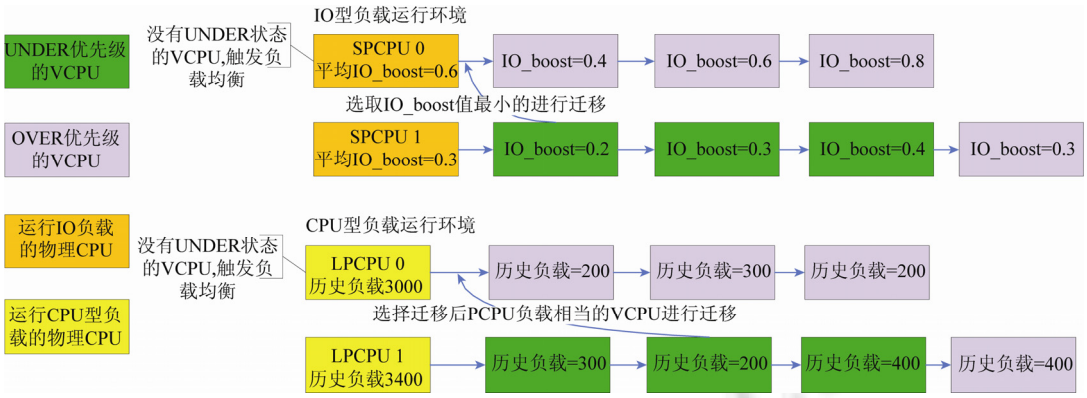


Fig.4 FLMS load balance strategy

图 4 FLMS 负载均衡策略

4 测 试

本节主要是针对 FLMS 调度算法进行性能评价.测试环境为两台使用万兆网卡连接在同一局域网的主机,其中一台搭建虚拟化环境,运行测试虚拟机;另一台主机使用原生系统配合测试.测试中开启处理器的超线程.系统负载统计值影响因子α、物理 CPU 历史负载权重因子β、在不同物理核与不同物理封装之间进行虚拟 CPU 迁移的阈值 A,B 分别设定为 0.125,0.875,40,80.这是在本文的测试环境中经过多次实验,考虑到系统稳定性以及实时负载对系统的影响而进行的取值.在不同软硬件环境下,这些因子应该随之变化.在本文的后续工作中,将会考虑动态设定这些因子的值.目前这些因子的值都是静态预设的.

4.1 I/O性能测试

如第 1.1 节所述,当主机的所有虚拟机都没有计算型负载时,虚拟机的 I/O 延时普遍很低,并且也不符合实际的应用场景,所以在本节测试中使用 lookbusy[20]来为每台虚拟机产生计算型负载,并配置同等数量的 LSVM,NLSVM 以及 MixVM.如果未作特殊声明,虚拟机的 CPU 使用率参数均为 50%.

4.1.1 I/O 读写测试

本节主要测试FLMS在I/O读写方面的性能,测试工具为I/O Zone[21].I/O Zone是一个文件系统的benchmark工具,可以测试文件系统的读写性能.测试中运行 6 台虚拟机.对比 LSVM 与默认 Credit 调度器虚拟机的 I/O 读写性能,读写文件大小为 2GB;Record 大小分别为 1M,2M,4M,8M;吞吐量单位为 KBps.测试结果如图 5 所示.其中,LS 为优化后的调度器,Nor 为默认调度器.从图中可以看出,I/O 写性能与默认调度器相比有 20%的提高,读性能提高为 6%.这是因为 FLMS 对 I/O 优化后,写处理函数调用更加及时,但是读文件往往因内存不命中而访问硬盘,因此会对性能有较大影响,从而导致性能提升不够明显.

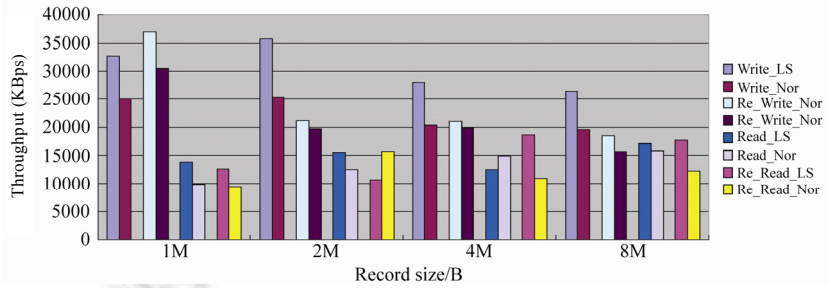


Fig.5 I/O read-write test

图 5 I/O 读写测试

© 中国科学院软件研究所 http://www.jos.org.cn

4.1.2 I/O 延时和带宽测试

本测试主要对比 FLMS 的 LSVM 与默认 Credit 调度器、I/O bounding 方案、vTurbo 以及 KVM 使用 virtio 环境下虚拟机的 I/O 延时和带宽性能.测试工具为 Ping 和 Netperf[22].Netperf 是一种网络性能测量工具,主要针对基于 TCP 或 UDP 的传输.测试结果如图 6、图 7 和表 1 所示.FLMS 的平均延时比 Credit 减少了 80%,比 KVM 减少了 30%,比 I/O bounding 减少了 17%,比 vTurbo 减少了 49%;FLMS 的带宽相对于默认的 Credit 调度器以及 KVM 有 50%的提升,相对于 vTurbo 以及 I/O bounding 方式,也提高了 10%.

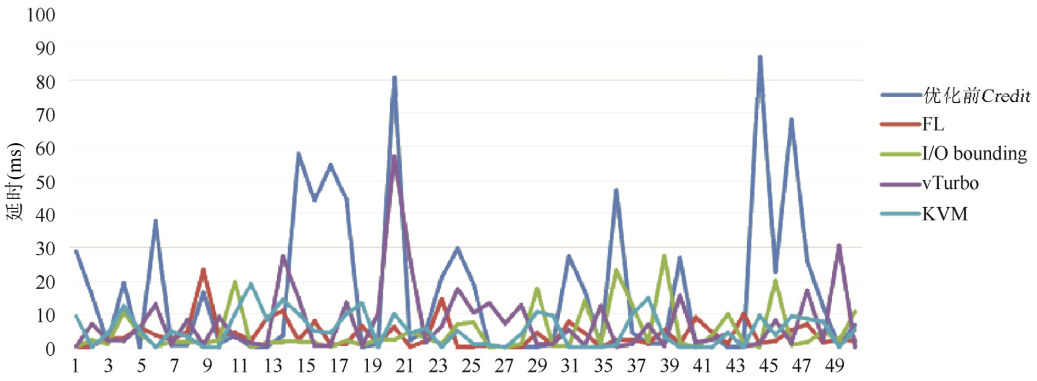


Fig.6 I/O latency test

图 6 I/O 延时测试

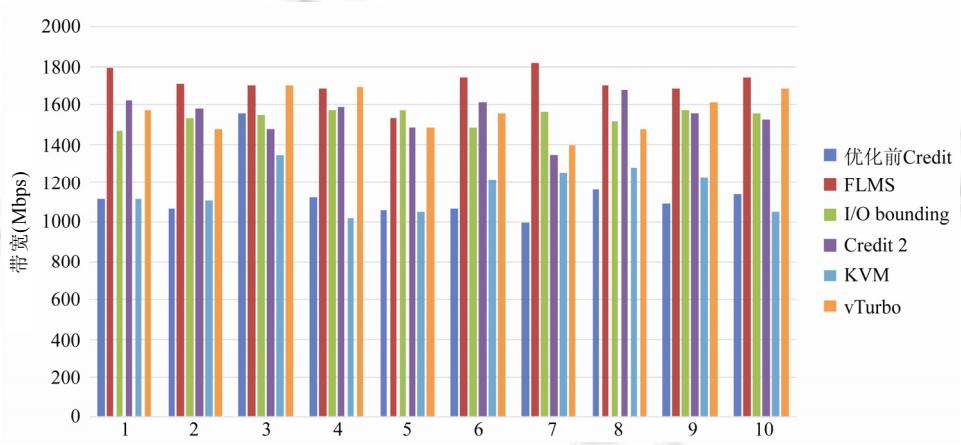


Fig.7 Throughput test

图 7 带宽测试

Table 1 Average latency and bandwidth comparison between the schedulers

表 1 调度器之间平均延时和带宽对比

	Credit	FLMS	IO bounding	KVM	vTurbo
平均延时(ms)	16.853	3.939	4.794	5.283	7.762
平均带宽(Mbps)	1 135.554	1 708.92	1 535.858	1 165.149	1 561.675

4.1.3 Web 压力测试

本节测试使用 Webbench[23]进行虚拟机的 Web 压力测试.Webbench 是一款网站压力测试工具,可以模拟并发请求,通过每分钟响应请求数和每秒传输数据量来对比性能.测试中运行 8 台虚拟机,在目标虚拟机中搭建

© 中国科学院软件研究所 http://www.jos.org.cn

Web服务,并在同一个子网中另一台主机上利用Webbench来产生压力,测试LSVM与默认Credit调度器的Web压力性能,测试结果如图8所示.FLMS相对于默认的Credit调度器在传输速度上有10%的提升.

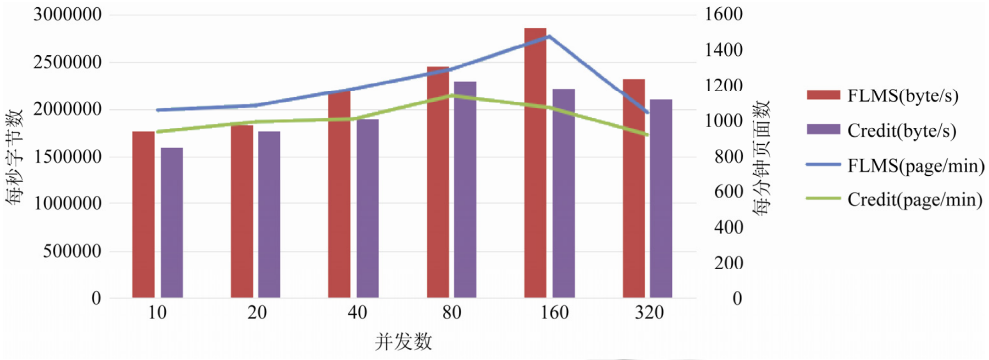


Fig.8 Web pressure test

图8 Web压力测试

4.2 负载均衡测试

Xen虽然在4.3版本之后加入了对负载均衡的优化,但仍然不够完善.本节主要测试FLMS在负载均衡中所带来的改善,包括在非NUMA和NUMA架构的机器上的测试对比.测试中运行8台虚拟机,其中4台虚拟机作为空闲虚拟机.另外4台虚拟机作为忙碌的虚拟机,利用Lookbusy产生系统负载,在CPU利用率分别为50%,65%,80%以及95%的情况下,进行整个系统中优化前后10s内负载均衡触发次数对比.测试结果如图9、图10所示,其中LS为优化后的调度器,Nor为默认调度器.从图中可以看出,FLMS相对于默认的Credit调度器,减少了虚拟CPU迁移频率以及远程的虚拟CPU迁移.采用FLMS后,非NUMA架构机器的跨物理封装VCPU迁移仅占总数的4%左右.相对而言,默认的Credit调度器负载均衡触发频率相当高,并且跨物理封装之间的虚拟CPU迁移达到迁移总数的31%.NUMA架构的机器有利于Credit调度器的负载均衡,在一定程度上减少了跨物理封装的虚拟CPU迁移,仅占到迁移总数的12%,但是仍然有较高的迁移频率.

FLMS减少了不必要的负载均衡以及远距离的VCPU迁移,同时带来了性能的提升.如图11所示,在NUMA以及非NUMA架构机器上进行了性能测试.测试中运行8台虚拟机,其中4台虚拟机作为空闲虚拟机.另外4台虚拟机作为忙碌的虚拟机,运行sysbench[24],保持100%的CPU使用率,测试其中一台虚拟机获取最大素数的时间.从测试中发现,优化后的负载均衡机制相对于默认的Credit算法有17%的提升.

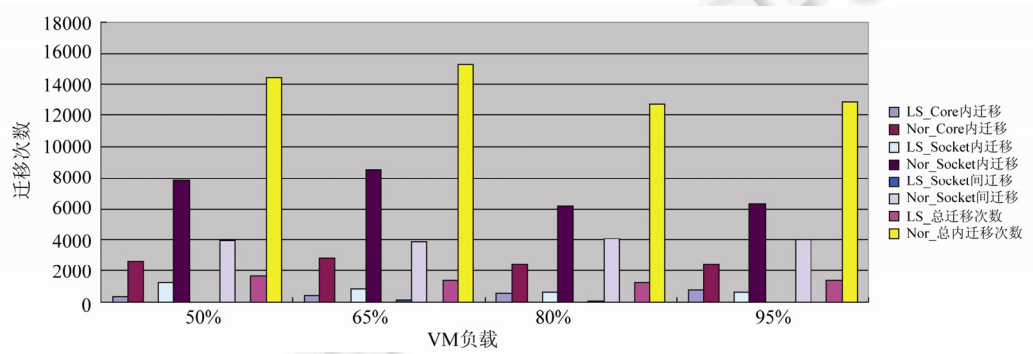


Fig.9 Load balance test on a non-NUMA machine

图9 非NUMA结构机器负载均衡测试

© 中国科学院软件研究所 http://www.jos.org.cn

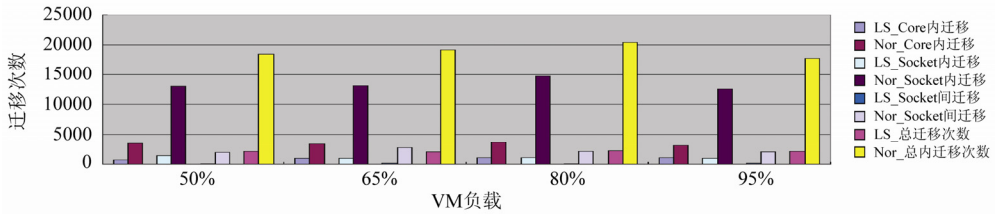


Fig.10 Load balance test on a NUMA machine

图 10 NUMA 结构机器负载均衡测试

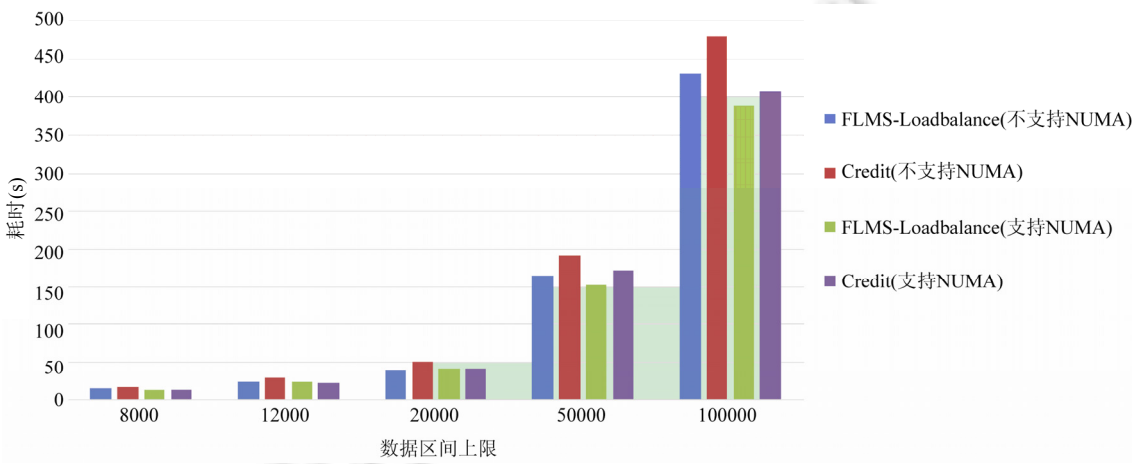


Fig.11 Load balance test

图 11 负载均衡测试

4.3 计算性能测试

FLMS 不仅在 I/O 性能方面优于默认的 Credit 调度器,而且由于负载均衡带来性能的提升,其计算性能也有一定的提高,本测试主要对比测试 FLMS 中 NLSVM 类型的虚拟机与默认 Credit 调度器的计算性能.为了保证测试全面,设计了只有 NLSVM 以及同时有 3 类虚拟机情况下的对比.测试中运行 8 台虚拟机,在目标虚拟机中运行 sysbench 达到满负荷求最大素数,比较两者求出最大素数所用的时间.其余虚拟机运行 Lookbusy.每个虚拟机均有 4 个虚拟 CPU,sysbench 测试时线程参数设置为 4.测试结果如图 12 所示.从图中可以看出,在 FLMS 方案下,无论是否有 LSVM 或者 MixVM 的干预,NLSVM 计算出最大素数都耗时更短,计算性能提升了 15%.

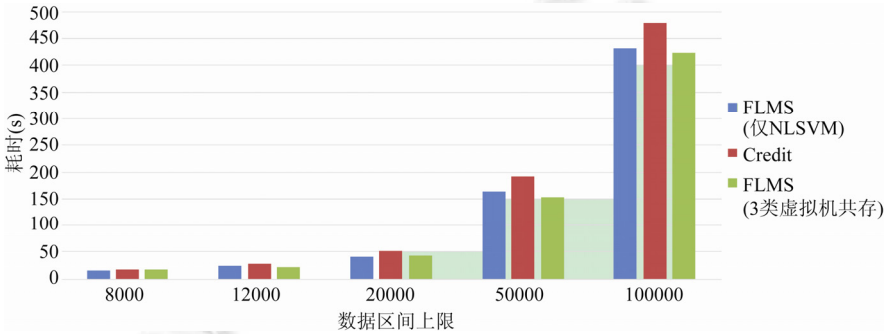


Fig.12 Computing capability test

图 12 计算性能测试

© 中国科学院软件研究所 http://www.jos.org.cn

4.4 公平性测试

FLMS同样继承Credit调度器公平性原则,虽然对虚拟机进行了类别区分,并修改了不同类别虚拟机的时间片机制,但未改变信用值的分配,因此,虚拟机之间的公平性仍然可以得到保障.在测试中创建了6个虚拟机,其中VM 1和VM 2类型为NLSVM,VM 3和VM 4类型为LSVM,VM 5和VM 6类型为MixVM,每个虚拟机中运行Lookbusy作为虚拟机负载,在5个不同的时间测试各个虚拟机的CPU使用率.测试结果如图13所示.从图中可以看出,系统中不同类别的虚拟机仍然得到几乎一致的物理CPU资源,系统公平性得到了保证.

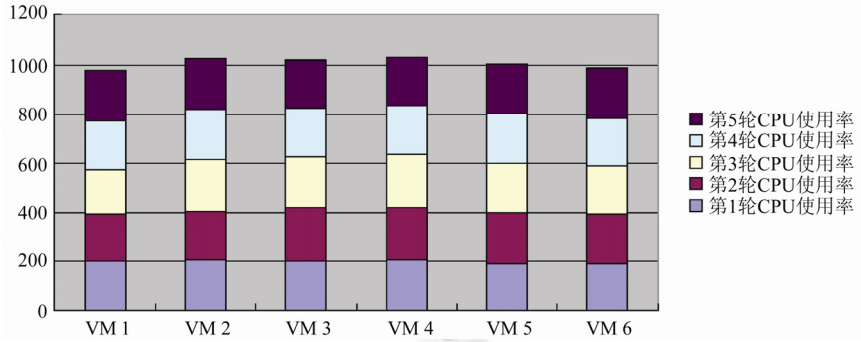


Fig.13 Fairness test

图13 公平性测试

5 总 结

目前CPU虚拟化与内存虚拟化技术已日趋成熟,而I/O虚拟化技术的发展却相对滞后,成为制约虚拟化技术部署的一个瓶颈.现今已有不少研究通过对虚拟CPU的调度算法进行优化来提升I/O性能,但这些方法大都存在一些局限性.本文针对I/O性能问题提出了一种灵活、高效的虚拟CPU调度算法(FLMS).FLMS通过将I/O型负载与计算型负载运行环境隔离,为运行I/O负载的虚拟CPU提供了短时间片与高频率的调度,并且不会影响计算型负载的性能.同时,根据FLMS的调度架构重新设计了优先级机制,让I/O任务能够以更高效率执行.此外也考虑到了目前主流的多处理器架构,优化了负载均衡机制,减少了远程虚拟CPU迁移,择优选择目标虚拟CPU.FLMS保证了虚拟机资源公平共享,具有很好的灵活性,并且不需要修改操作系统,有很好的兼容性.

致谢 在此,我们向对本文的工作给予支持和建议的同行,尤其是武汉光电国家实验室(华中科技大学)信息存储及应用实验室的老师和同学们表示感谢.

References:

[1] Waldspurger C, Rosenblum M. I/O virtualization. Communications of the ACM, 2012,55(1):66−73. [doi: 10.1145/2063176.2063194]

[2] Smith JE, Nair R. The architecture of virtual machines. Computer, 2005,38(5):32−38. [doi: 10.1109/MC.2005.173]

[3] Rosenblum M, Garfinkel T. Virtual machine monitors: Current technology and future trends. Computer, 2005,38(5):39−47. [doi: 10.1109/MC.2005.176]

[4] Uhlig R, Neiger G, Rodgers D, Santoni AL, Martins FCM, Anderson AV, Bennett SM, Kagi A, Leung FH, Smith L. Intel virtualization technology. Computer, 2005,38(5):48−56. [doi: 10.1109/MC.2005.163]

[5] Kutch P. PCI-SIG SR-IOV Primer: An introduction to SR-IOV technology. Application note 321211-002, Intel Corporation, 2011.

[6] Xu C, Gamage S, Lu H, Kompella R, Xu DY. vTurbo: Accelerating virtual machine I/O processing using designated turbo-sliced core. In: Proc. of the 2013 USENIX Annual Technical Conf. (USENIX ATC 2013). Berkeley: USENIX, 2013. 243−254.

[7] Ding XX, Xiong AP, Yang C. Optimation of Xen scheduler for multitasking. In: Proc. of the 4th IEEE Int'l Conf. on Software Engineering and Service Science (ICSESS). Piscataway: IEEE, 2013. 754−757. [doi: 10.1109/ICSESS.2013.6615415]

© 中国科学院软件研究所 http://www.jos.org.cn

[8] Rao J, Wang K, Zhou X, Xu C. Optimizing virtual machine scheduling in NUMA multicore system. In: Proc. of IEEE the 19th Int'l Symp. on High Performance Computer Architecture (HPCA 2013). Piscataway: IEEE, 2013. 23−27. [doi: 10.1109/HPCA.2013.6522328]

[9] Guan HB, Ma RH, Li J. Workload-Aware credit scheduler for improving network I/O performance in virtualization environment. IEEE Trans. on Cloud Computing, 2014,2(2):130−142. [doi: 10.1109/TCC.2014.2314649]

[10] KVM homepage. http://www.linux-kvm.org/page/Main_Page

[11] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. In: Proc. of the Symp. on Operating Systems Principles (SOSP). 2003. [doi: 10.1145/945445.945462]

[12] Ackaouy E. The xen credit CPU scheduler. 2006. http://www-archive.xenproject.org/files/summit_3/sched.pdf

[13] Shim H, Lee SM. CFS-v: I/O demand-driven VM scheduler in KVM. 2014. http://events.linuxfoundation.jp/sites/events/files/slides/kvmforum14_hshim.pdf

[14] Jiang W, Zhou YS, Cui Y, Feng W, Chen Y, Shi YC, Wu QB. CFS optimizations to KVM threads on multi-core environment. In: Proc. of the 15th Int'l Conf. on Parallel and Distributed Systems. Piscataway: IEEE, 2009. 348−354. [doi: 10.1109/ICPADS.2009.83]

[15] Ongaro D, Cox AL, Rixner S. Scheduling I/O in virtual machine monitors. In: Proc. of the 4th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. New York: ACM, 2008. 1−10. [doi: 10.1145/1346256.1346258]

[16] Zeng LF, Wang Y, Shi W, Feng D. An improved Xen credit scheduler for I/O latency-sensitive applications on multicores. In: Proc. of the Int'l Conf. on Cloud Computing and Big Data. Piscataway: IEEE, 2013. 267−274. [doi: 10.1109/CLOUDCOM-ASIA.2013.40]

[17] Hu Y. Long X, Zhang J, He J, Xia L. I/O scheduling model of virtual machine based on multi-core dynamical partitioning. In: Proc. of the 19th ACM Int'l Symp. on High Performance Distributed Computing. New York: ACM, 2010. 142−154. [doi: 10.1145/1851476.1851494]

[18] Xu C, Gamage S, Rao PN, Kangarlou A, Kompella RR, Xu D. vSlicer: Latency-Aware virtual machine scheduling via differentiated-frequency CPU slicing. In: Proc. of the 21st ACM Int'l Symp. on High Performance Distributed Computing. New York: ACM, 2012. 3−14. [doi: 10.1145/2287076.2287080]

[19] Cheng YX, Chen WZ, Chen X, Xu B, Zhang SY. A user-level NUMA-aware scheduler for optimizing virtual machine performance. In: Proc. of the Int'l Workshop on Advanced Parallel Processing Technologies. Berlin, Heidelberg: Springer-Verlag, 2013. 32−46. [doi: 10.1007/978-3-642-45293-2_3]

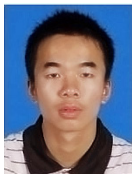
[20] https://www.devin.com/lookbusy/

[21] http://www.iozone.org/

[22] http://www.netperf.org/netperf/

[23] http://home.tiscali.cz/~cz210552/webbench.html

[24] https://launchpad.net/sysbench



刘珂男(1991－),男,湖南湘乡人,硕士,主要研究领域为 I/O 虚拟化.

刘景宁(1957－),女,教授,博士生导师,CCF 专业会员,主要研究领域为计算机系统结构,计算机存储系统.

童薇(1977－),女,博士,讲师,CCF 专业会员,主要研究领域为海量存储系统,固态存储,I/O 虚拟化.

张炬(1988－),男,硕士,主要研究领域为 I/O 虚拟化.

冯丹(1970－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为信息存储系统,网络存储,固态存储及性能评价.

© 中国科学院软件研究所 http://www.jos.org.cn