计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(7):10−16 [doi: 10.15888/j.cnki.csa.005857]

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

Xen的I/O虚拟化性能分析与优化①

刘　佩[1], 章建雄[1], 马　鹏[1], 阎燕山[2]

[1](华东计算技术研究所, 上海 200000)
[2](中国航空无线电电子研究所615所, 上海 200241)

摘　要: 在Credit算法应用中, 由I/O事务唤醒的VCPU处于最高优先级BOOST状态, 优先抢占PCPU资源, 使I/O操作的响应速度提高, 但多个虚拟机同时进行I/O操作时, 会引起较长延时和公平性原则被破坏问题. 针对这个问题, 研究分析SEDF算法、Credit算法、Credit2算法, 提出L-Credit调度算法解决多个虚拟机同时进行I/O操作引起响应延迟的问题. 通过监测I/O设备环共享页面中响应和请求的个数的方法, 对处于BOOST状态下I/O操作进一步细化排序, 使稀疏型I/O操作较密集型I/O操作先调用执行. 通过对L-Credit算法与Credit算法在同一应用场景下反复对比实验, 得出L-Credit算法可以提高I/O响应性能, 并且继承了Credit算法负载均衡和按比例公平共享的特点.

关键词: I/O虚拟化; Credit; SEDF; Credit2; L-Credit

引用格式: 刘佩,章建雄,马鹏,阎燕山.Xen的I/O虚拟化性能分析与优化.计算机系统应用,2017,26(7):10–16. http://www.c-s-a.org.cn/1003-3254/5857.html

Performance Analysis and Optimization on I/O Virtualization of Xen

LIU Pei[1], ZHANG Jian-Xiong[1], MA Peng[1], YAN Yan-Shan[2]

[1](East China Institute of Computer Technology, Shanghai 200000, China)
[2](China Aeronautical Radio Electronics Research Institute, Shanghai 200241, China)

Abstract: In the application of Credit algorithms, the VCPU awakened by the I/O transaction is in the highest priority BOOST state, which gives it priority to gain access to the PCPU resources and improves the response speed of the I/O operation, but that will cause long time delay and destroy fairness principle, when multiple virtual machines are doing the operation of I/O at the same time. To solve this problem, SEDF algorithm, Credit algorithm, Credit2 algorithm are analyzed and L-Credit scheduling algorithm is proposed to reduce the response delay caused by multiple virtual machines' concurrent I/O operation. By monitoring I/O device ring sharing page to get the number of requests, which can further refine the sort in the I/O state BOOST operation, so that sparse type I/O operation can the implementation before the I/O intensive operations. According to the comparison report of the L-Credit algorithm and Credit algorithm in the same application scenario experiment, L-Credit algorithm can improve the performance of the I/O response, and inherits the Credit algorithm load balance and the characteristics of proportional fair sharing.

Key words: I/O virtualization; Credit; SEDF; Credit2; L-Credit

虚拟化技术是云计算中最关键、最核心的技术原动力[1]. 云计算的提出推动了系统虚拟化技术的发展, 而虚拟化的层次将决定虚拟机性能[2]. 虚拟化技术的研究方向主要分为: CPU虚拟化、内存虚拟化和I/O虚拟化[3], 其中CPU虚拟化的核心问题是如何保证VCPU (virtual CPU)的正确执行; 内存虚拟化的核心问题是如何利用分块共享的思想来虚拟计算机的物理内存; I/O虚拟化的核心问题是如何协调多个虚拟机对同一

① 收稿时间: 2016-11-08; 收到修改稿时间: 2016-12-12

套硬件硬件设备的应用同时保证I/O操作的实时性、正确性. Intel和AMD先后分别推出了Intel VT (Virtualization Technology)和AMD VT产品[5], 很大程度上解决了CPU虚拟化和内存虚拟化问题. 而I/O设备虚拟化由于I/O设备具有异构性强、内部状态不易控制的特点, 成为虚拟化的技术难点之一. 虚拟机是系统级虚拟技术的应用平台, 其中Xen虚拟机管理器VMM是一个完全开源项目, 具有较好的兼容性和运行效率, 在学术界和业界受到广泛重视[2]Xen. 在过去十几年时间中, 对VCPU调度算法不断的优化改进, 按照时间排序: BVT算法、SEDF算法、Credit算法、Credit2算法, 目前最新版的Xen虚拟机的默认算法为Credit算法, Credit2算法目前还在实验阶段[4].

本文以Xen虚拟机管理器为基础, 通过对VCPU调度算法: SEDF、Credit、Credit2调度算法研究分析, 提出一种新的调度算法L-Credit算法, 用于解决多台虚拟机同时进行I/O操作时, 存在的I/O操作延时长和公平性原则被破坏等问题.

1 Xen虚拟机结构分析

在Xen系统中, 存在一个轻量级的软件层虚拟机管理器(VMM或Xen Hypervisor), 向运行在它之上的虚拟机提供虚拟硬件资源, 同时分配和管理这些资源, 并保证虚拟机之间的相互隔离[3]. 虚拟机则称之为域(Dom). 虚拟机管理器VMM存在于操作系统与硬件之间, 主要作用是为运行的操作系统内核提供硬件环境. Xen采用混合模式, 设置一个Dom为特权域Dom0, 其它域称之为DomU. Dom0用于辅助Xen管理其他域DomU, 提供相应的虚拟资源服务, 特别是DomU的I/O操作.

根据VMM向虚拟机提供硬件资源的方式, Xen虚拟化方式分为FV(Full Virtualization)、PV(Para-Virtualization)、HAV(Hardware Assisted Virtualization). 因为结构特点, FV技术虽然可以向虚拟机虚拟出和真实硬件完全相同的硬件环境, 以及给虚拟机提供完整的硬件支持服务, 但对于I/O设备而言, 仍然是基于PV环境开发的前后端驱动模式, 因此, 最初的Xen完全虚拟化需要使用Qemu来仿真计算机硬件, 其I/O设备的性能比PV技术要低. 而HAV技术需要CPU支持Intel-VT技术或者AMD-V技术. 对比而言, PV技术实现的I/O虚拟化性能更优良和方便.

以PV为例对Xen的I/O虚拟化过程进行说明. 如图1所示, 在Xen系统中, I/O操作采用的是前后端(Frontend-Backend)I/O技术, 其中前端设备驱动(Front-end Device Driver)位于DomU中将I/O请求发送给位于Dom0的后端设备驱动(Back-end Device Driver), 在由后端设备驱动接收I/O请求, 权限检查, 通过之后交由原生设备驱动. Dom与VMM之间的数据传输采用的是基于循环队列结构的生产者消费者模型. 接收操作系统指令的传递, 是由Xen为每个Domain建立的VCPU完成. VCPU的调度算法对于I/O虚拟化的性能有着决定性的影响. 在Xen中, 关于VCPU的调度算法主要有SEDF调度算法、Credit调度算法、Credit2调度算法.

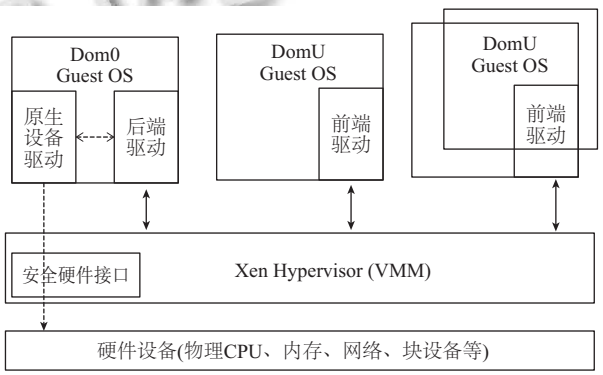

图1 Xen半虚拟化结构图

2 算法分析

I/O虚拟化技术的性能主要体现在实时性、负载均衡等特性上. SEDF算法可以在单核应用场景中保证I/O操作的实时性; Credit算法可以在多核应用场景中很好实现I/O虚拟化的负载均衡特性; Credit2算法结合Credit算法在多核应用场景中保证负载平衡的基础上, 解决了I/O虚拟化操作的延时性等问题, 但是带来了缓存压力过大的问题.

2.1 SEDF算法

SEDF算法是一种动态优先调度算法, 最初为每个VCPU进行初始化时设置一个截止期限. 截止期限最早的VCPU在VCPU调度时具有较高的优先级会优先调用. SEDF算法通过设置参数(s, p, x)控制VCPU的运行, 其中s指代时间片, p指代周期时间, x是布尔值. 在时间p内, 设置的VCPU至少可以获取s单位的CPU运行时间. x值代表额外的CPU运行时间(p-s)是否可以继续持有. 调度器中维护一个可运行队列和一个等待队列, 可运行队列中按照截止期限顺序保存当前周期仍有运行时间的队列, 等待队列中保存当前周期运行时间已消耗

完VCPU, 并结合x参数以及截止时间依次排列. 每次调度时, 处理器获取运行队列中的头元素进行调度. 假设系统中存在两个VCPU, 初始化时设vcpu1(2, 7, 1)、vcpu2(1, 9, 0), 如图2, vcpu1的截止时间是第5个时刻, vcpu2的截止时间是第8个时刻, 按照算法的优先原则, vcpu1先被调度执行, 执行2个时间单位, 接着调度执行vcpu2, 执行1个时间单位. 那么在时刻3处, vcpu1和vcpu2调度完成, 那么剩余的时间CPU处的状态由x决定, x都为0时, 代表周期剩余的时间是不能被VCPU使用, CPU处于idle状态. 若x为1, 代表剩余周期的时间, 被某VCPU使用. 由于vcpu1中x=1, 代表剩余周期vcpu1将占用CPU, 直到有新的VCPU被释放.

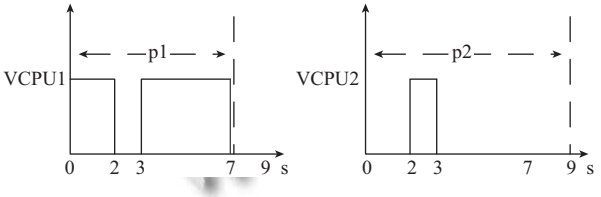

图2 SEDF算法示例

SEDF算法在负载较轻时, 处理器利用率很高, 但是当多个负载超过50%时, 会导致有些进程错过截止时间, 忽略执行, 对性能有很大影响. 而且算法只能对单个CPU进行SEDF调度, 不能够进行多个CPU之间的负载平衡操作.

2.2 Credit算法

Credit调度算法是以VCPU调度为单位, 在调度过程中, 每个VCPU都包含两个配置参数: weight(VCPU的权重)和cap(VCPU能运行的时间的上限). 由weight决定参数Credit(VCPU能够运行的时间)的值, 当VCPU被调度过程中credit值会减少. Xen4.1虚拟机之后, 每个VCPU包含了3种运行状态: BOOST、UNDER、OVER. UNDER状态: 当VCPU处于正常等待运行并且credit值不为负时, VCPU处于UNDER状态. OVER状态: 只要当credit值为负时, VCPU切换为OVER状态, 不再被调用执行. BOOST状态: 当被事件唤醒的虚拟机具有较高的优先级时, VCPU进入BOOST状态, 如果当前运行的VCPU为UNDER状态, 就抢占执行, 如图3所示. 若当前运行状态为BOOST时, 按照队列先到先得的规则执行.

Credit调度算法在执行调度时, 只关心VCPU所处状态, 按照被调度的优先级由高到低排序, 依次是BOOST状态、UNDER状态、OVER状态. Credit算法为每个物理CPU维护一个运行队列, 按照先后顺序依次是BOOST、UNDER、OVER 3个区域. 每个区域内的VCPU按照先后顺序排列, 按照队列的特性, 每10 ms(一个时间片)响应一次中断, 执行选择队列第一个VCPU运行并且消耗credit值. 如果被调用VCPU的credit值为负处于OVER状态, 那么它将不再被继续执行, 重新计算credit值, 重新调度队列VCPU. 若进行了3个时间片, 一直执行VCPU的credit值仍为非负值, 中止调度运行, 重新计算credit值, 重新调度队列VCPU. 系统每隔10 ms会中断一次, 当前正在运行的VCPU会被消耗100个credit值, 当所有的credit值得总和为负, 按照比例加速最初设置的weight值. 如果某个VCPU的credit值累积到一定值域, 将其减半, 然后处于睡眠状态.

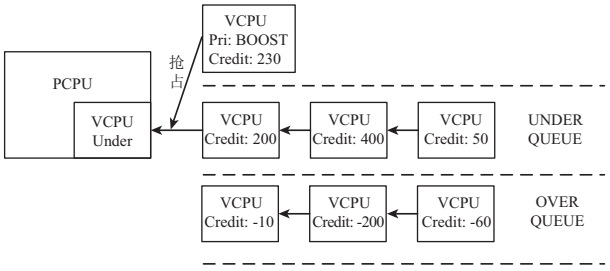

图3 Credit算法运行状态

加入了BOOST状态的Credit算法大大降低了响应延迟平均值, 提高了虚拟I/O操作性能. 但是当多台虚拟机同时运行I/O操作时, 它们按照虚拟机的优先权限都被设置为BOOST状态, 而按照同一状态模式下, 执行顺序按照先到先得服务机制, 会造成很长延迟和公平性原则的破坏.

2.3 Credit2算法

Credit2算法任然沿用Credit算法参数weight和credit. 此时weight含义是credit值消耗速度, weight值越高消耗速度越慢. Credit2调度算法任然采用队列去组织所有VCPU, 但是没有按照优先级状态处理VCPU. 在Credit算法各个状态的队列排序不会关心credit值大小, 但是Credit2算法中会按照credit值由大到小进行队列排序. 当有新的VCPU加入时, 会队列进行从头到尾的遍历, 按照credit值大小插入合适的位置, 然后选择CPU选择队头元素进行运行, 如图4所示. 当处于队列中的下一个vcpu值是小于或者等于零时, credit命令是reset, 意味着, 此时每个vcpu的credit值减去最小值, 然后加上固定的credit值.

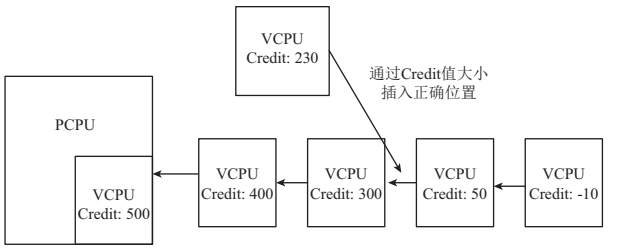

图4 Credit2算法运行状态

对于I/O虚拟化来说, I/O操作相比于其它任务占用的CPU资源较少, 会优先运行, 而且在同样是I/O请求的情景下, 对于稀疏型I/O任务会优先运行, 解决了Credit算法产生的多个I/O操作同时进行时, 延迟和公平性原则得破坏的问题, 但是由于频繁的切换, 同时大大的增加了缓存的压力, 目前Credit2算法仍处于改进与测试中, 还没有被设定为Xen默认的调度器.

3 算法优化

3.1 算法优化的基本思想

通过对SEDF算法、Credit算法、Credit2算法分析比较, 针对解决虚拟机I/O操作延时问题, 在Credit算法的基础上可以根据唤醒VCPU的事件类型, 密集型I/O操作或者稀疏型I/O操作, 进一步细分排序BOOST状态下VCPU. 根据Xen结构的I/O操作结构特点, Xen采用设备分离的策略如图5, 原生的设备驱动存在于特权域Dom0中同时还包含后端设备驱动, 在DomU中存在的是前端设备驱动. 前端驱动通过环形队列缓冲区、授权表以及事件通道与后端驱动进行交互. 当虚拟机的某个操作系统要发起I/O操作时, 首先前端驱动建立通信链接, 分配一个空闲页面作为I/O设备环、分配授权表引用并在Xenstore中存放、分配允许后端驱动连接的未被绑定的事件通道, 然后后端通信建立链接, 从Xenstore中读取前端提供的授权表引用、映射到I/O设备环的共享页面、获取前端绑定的事件通道. 整个过程在分离设备驱动的共享内存中实现, 交换请求和响应, 然后通过事件通道进行异步通知, 每个域都有各自的I/O设备环. 而且操作系统允许将一系列类似I/O请求形成请求队列, 然后发送一个Hypercall, 使Xen以批量的方式来接受和响应请求[3]. 在I/O设备环的代码实现过程中, 定义了三种结构体, 共享环页面结构体、前端私有变量结构体、后端私有变量结构体如, 表1和表2所示. 通过获取结构体元素nr_ents知道共享页面中响应和请求的个数, 这个将作为条件加入到新算法L-Credit算法中.

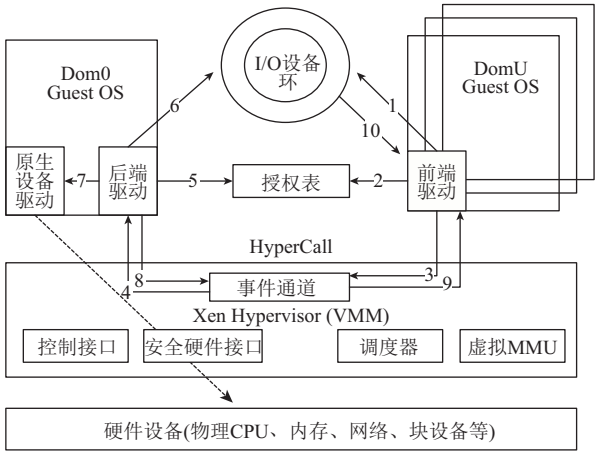

图5 I/O设备环

表1 前端私有变量结构体

struct__name##_front_ring(前端环结构体)	
RING_IDX req_prod_pvt	表示请求生产者的索引
RING_IDX rsp_cons	表示响应消费者的索引
unsigned int nr_ents	共享页面中响应和请求的个数
struct__name##_sring*sring	共享环的指针

表2 后端私有变量结构体

struct__name##_back_ring(后端环结构体)	
RING_IDX rsp_prod_pvt	表示响应生产者的索引
RING_IDX req_cons	表示请求消费者的索引
unsigned int nr_ents	共享页面中响应和请求的个数
struct__name##_sring *sring	共享环页面的指针

3.2 优化算法L-Credit

Credit算法的特点在于VCPU可以有三个运行状态BOOST、UNDER、OVER, 其中处于BOOST状态的VCPU优先被CPU调度执行. 而Credit算法中I/O操作的VCPU处于最高优先级BOOST, 确实降低了响应延迟平均值, 提高了虚拟I/O性能, 但是当面临多个I/O操作同时进行时, 它们按照虚拟机的优先权限都被设置为BOOST状态, 而按照同一状态模式下, 执行顺序按照先到先得服务机制, 会造成很长延迟和公平性原则得破坏. 为了解决这个问题L-Credit算法可以加入一个新的运行顺序标准: 当VCPU被新的事务唤醒时, 在Credit算法对于事务优先等级的划分进行队列排序基础上, 在优先等级同样为BOOST状态时, 通过判断当前域中, 对应的共享页面中响应和请求的个数对VCPU进行排序,

使处于BOOST状态的VCPU调度顺序进行进一步细分, 数目少的先调度执行, 数目相同时任选一个调度执行, 从而达到区分发起的I/O事务是属于稀疏型还是密集型操作的目的, 保证稀疏型I/O操作先于密集型I/O操作先被调度执行, 如图6所示, 部分代码如下文所示.

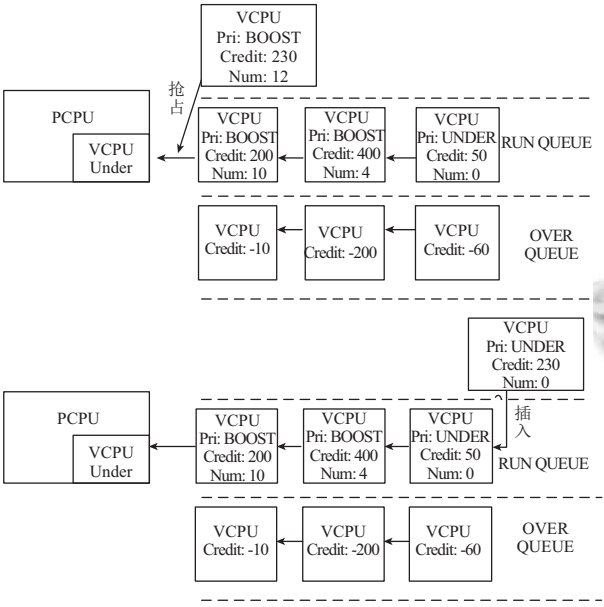

图6 L-Credit算法运行图

```
List_for_each(iter, runq)
{
  const struct csched_vcpu *const iter_svc = __runq_
elem(iter);
  if((svc->pri = iter_svc->pri) & compare (svc,
iter_svc) | (svc->pri > iter_svc->pri))
    break;
}
```

L-credit算法的实现主要分为四个部分: Sburn_credit、runq_insert、runq_elem、Csched_load_balance, 如图7所示. 在runq_insert加入compare监控函数, 用于获取VCPU之间共享页面中响应和请求的数值, 根据返回整数值作为同一优先级BOOST状态的操作排序依据. Compare监控函数伪代码如下:

```
/* time-share waking up */
#define CSCHED_PRI_TS_BOOST 0
/* time-share w/ credits */
#define CSCHED_PRI_TS_UNDER -1
/* time-share w/o credits */
```

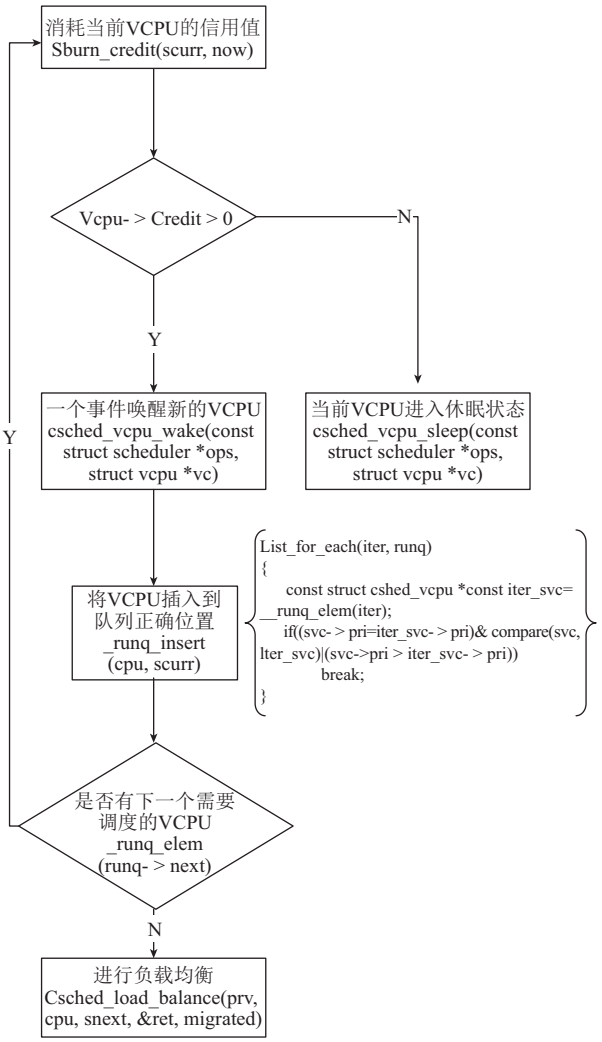

图7 L-Credit算法结构

```
#define CSCHED_PRI_TS_OVER -2
/* idle */
#define CSCHED_PRI_IDLE -64
Boolean Compare(struct csched_vcpu *vcpu1, struct
csched_vcpu *vcpu2)
{
/*判断新唤醒的VCPU是否是由I/O操作引起*/
If(vcpu1->pri < CSCHED_PRI_TS_BOOST)
    return false;
/*判断新唤醒的vcpu1与vcpu2之间共享页面中响
应和请求的数值*/
elseIf( vcpu1->sdom->evtchn->nr_events >
vcpu2->sdom->evtchn->nr_events)
    Return true;
else
```

```
    return false;
}
```

Sburn_credit消耗信用值, 设$C_s(P)$为每个PCPU更新时全部可分配的credit值, $W_s(P)$为PCPU对应所有VCPU全部weight值之和, 且每个域的VCPU数相同, 若第k个域对应的weight值为$W_k(P)$, 那么第k个域第i个VCPU每次更新时可分配的credit值为$\Delta C_{ki}(P)$, 满足:

$$\Delta C_{ki}(P) = C_s(P) * W_k(P) / W_s(P)$$

4 L-Credit实验结果分析

为验证L-Credit算法性能, 搭建测试环境如下:

硬件平台: thinkpad服务器, 处理器Inter Core i3 CPU, 内存 4GB, CPU工作频率: 2.4GHZ

软件平台: 安装Xen4.4.2版本开源Xen虚拟机管理器, 特权域Dom0为系统Ubuntu 14.04, 其它3个域DoumU系统Ubuntu 13.1, 使用iozone命令模拟I/O操作, iostate命令用于实时监控每个I/O请求处理的平均时间及其它信息.

实验场景: 分别就经典算法Credit和L-Credit算法进行比较, 在Dom0中进行稀疏型I/O操作, DomU进行I/O密集型操作, 运行iozone命令产生对应的I/O操作, iostate命令用于实时监控每个I/O请求处理的平均时间.

操作:

(1)开源Xen虚拟机管理器VCPU调度算法选择: Credit调度算法和L-Credit调度算法二选一, 进行实验比较.

```
static const struct scheduler *schedulers[] = {
&sched sedf_def,
&sched credit_def,
&sched credit2_def,
&sched l-credit_def,
};
/*选择Credit调度算法作为VCPU调度算法*/
static char __initdata opt_sched[10] = “credit”;
/*选择L-Credit调度算法作为VCPU调度算法*/
static char __initdata opt_sched[10] = “l-credit”;
string_param(“sched”, opt_sched);
```

(2) 进行I/O读写测试, 测试文件大小是128 M, 记录块从2 K到8 M, 并将测试数据输出到Excel文件中

命令行: /opt/iozone/bin/iozone -a -s 128 m -i 0 -i 1 -f /tmp/testfile -y 2k -q 8m -Rb output.xls

(3) 进行I/O操作的实时监控

命令行: iostat -d -x -k 1 10

其中, await(ms): 每一个I/O请求的处理的平均时间; svctm(ms): 表示平均每次设备I/O操作的服务时间; %util: 在统计时间内所有处理I/O时间, 除以总共统计时间. 如果svctm的值与await很接近, 表示几乎没有I/O等待, 如果await的值远高于svctm的值, 则表示I/O队列等待太长.

(4) 多次重复测试, 实验结果统计如图8、图9和图10所示.

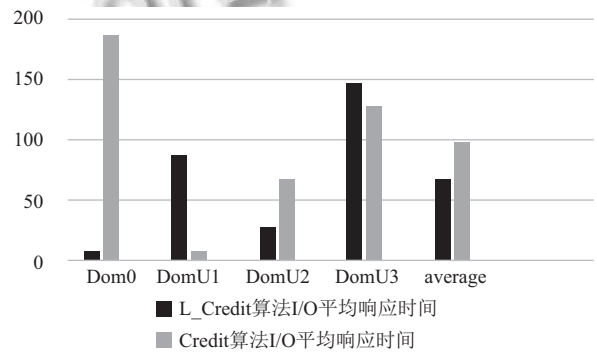

图8 Credit算法出现最坏情况下I/O平均响应

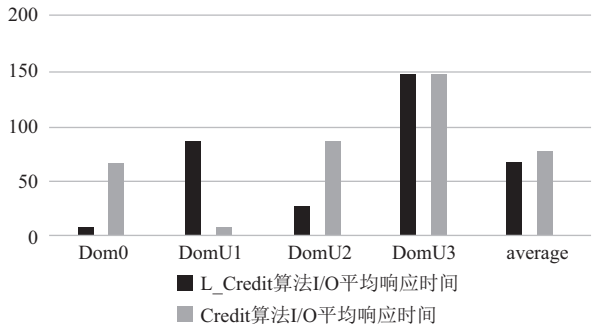

图9 Credit算法与L-Credit算法平均响应

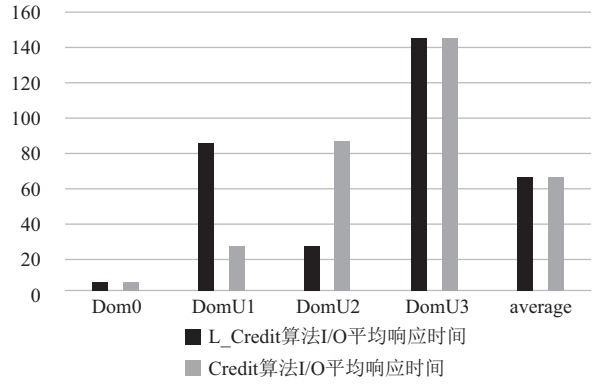

图10 Credit算法出现最好情况下I/O平均响应

设T_{dl}为延时响应时间长度, T_{slice}为时间片长度, T_{prd}为VCPU被抢占前使用CPU的时间长度. 而在Credit算法下, 唤醒VCPU的I/O事件到达后, 若同一时间没有其他的I/O操作产生既VCPU处于非阻塞状态, 那么I/O事件的响应将等待PCPU对其的调度, 由T_{dl}值由T_{slice}和队列中的VCPU数目决定. 对于I/O事务类型, 稀疏型I/O操作所耗费时间一定要小于I/O密集型操作所耗费时间既value_min=T_{slice}. 根据Credit算法的特点, 当同时出现多个I/O操作时, 会发生阻塞, 最坏的情况是I/O稀疏型操作在所有密集型I/O操作完成之后才被PCUP调度执行, 在本次实验场景中, Credit算法可能出现最坏情况如图11所示, 按照执行先后顺序T_{dl}=T_{prd1}+T_{slice1}+T_{slice2}+T_{slice3}, 实验结果如图8所示. Credit算法可能出现最好的情况是稀疏型I/O操作最先被执行如图12所示, 按照执行先后顺序T_{dl}=T_{prd0}+$T_{slice.}$+T_{slice1}+T_{slice2}, 实验结果如图10所示. 改进之后的L-Credit算法, 在同时发生多个I/O操作事件时, 可以自主的让I/O稀疏型任务先执行如图12所示, 实验结果如图8、9和10所示, 能够确保减少平均响应时间.

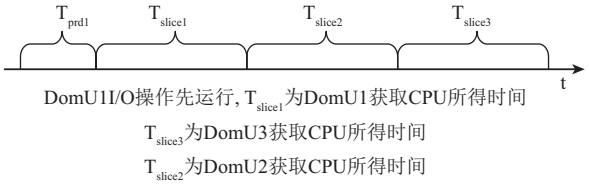

图11 最坏情况下时间延迟

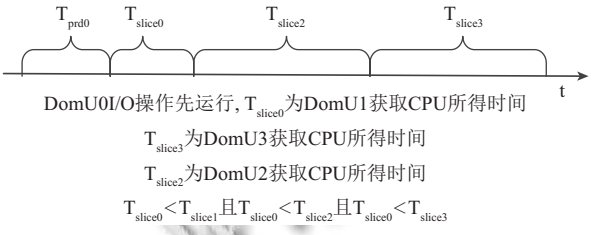

图12 最好情况下时间延迟

5 结束语

本文主要对Xen虚拟机相关的VCPU调度算法(Credit、Credit2、SEDF调度算法)进行了研究分析, 新提出的L-Credit调度算法相比于Credit调度算法, 增加了compare(struct csched_vcpu *vcpu1, struct csched_vcpu*vcpu2), 用于对处于同一BOOST状态的VCPU队列进一步细分排序, 解决多个I/O操作同时进行时, 延迟和公平性原则得破坏的问题. 在未来的工作中, 对Credit算法与L-Credit算法在多核处理器应用场景下, 对同时发起I/O操作的虚拟机数目依次进行增加, 进行重复试验, 对L-Credit算法进一步验证.

参考文献

1 惠新忠. Xen虚拟I/O优化策略[硕士学位论文]. 大连: 大连理工大学, 2010.
2 丁晓波, 马中, 戴新发. 面向响应延迟的虚拟机动态时间片调度算法. 计算机工程, 2015, 41(7): 11–16, 24.
3 石磊, 邹德清, 金海. Xen虚拟化技术. 武汉: 华中科学技术大学出版社, 2009.
4 Hnarakis R. In Perfect Xen, A Performance Study of the Emerging Xen Schedule. California: California Polytechnic State University, 2013.
5 胥平勇. 基于缓存关联的Xen虚拟机调度优化[硕士学位论文]. 南京: 南京大学, 2012.
6 广小明. 虚拟化技术原理与实现. 北京: 电子工业出版社, 2012.
7 张天宇, 关楠, 邓庆绪. Xen虚拟机Credit调度算法的实时性能分析. 计算机科学, 2015, 42(12): 115–119.
8 Hess K, Newman A. 虚拟化技术实战. 徐炯, 译. 北京: 人民邮电出版社, 2012.
9 郑兴杰. 基于SMP架构的半虚拟化CPU调度算法研究[硕士学位论文]. 哈尔滨: 哈尔滨工程大学, 2009.
10 黄漾. 多核平台下XEN虚拟机动态调度算法研究. 计算技术与自动化, 2014, 33(3): 84–87.
11 宋聿, 蒋烈辉, 董卫宇, 等. 一种独立式I/O虚拟化方法研究. 计算机工程, 2014, 40(10): 81–85. [doi: 10.3969/j.issn.1000-3428.2014.10.016]
12 李莹莹, 乔平安. 云环境下网络I/O虚拟化的研究与改进. 计算机与数字工程, 2015, 43(4): 684–688.
13 王凯, 候紫峰. Xen虚拟机的虚拟CPU松弛协同调度方法. 计算机研究与发展, 2012, 49(1): 118–127.
14 张灿群. Xen虚拟CPU调度算法的研究与改进[硕士学位论文]. 武汉: 华中科技大学, 2012.
15 顾振宇, 张申生, 李晓勇. Xen中Credit调度算法的优化. 微型电脑应用, 2009, 25(2): 1–3.
16 叶汉民, 宋子航. 数据包聚合算法提高云计算环境下的网络I/O虚拟化. 计算机应用与软件, 2016, 33(1): 127–130, 133.
17 郭御风, 郭诵忻, 邓宇. 众核处理器中硬件支持的I/O虚拟化优化技术研究. 计算机科学, 2012, 39(1): 299–304.
18 李超. SR-IOV虚拟化技术的研究与优化[硕士学位论文]. 长沙: 国防科学技术大学, 2010.
19 王彤. Xen虚拟机调度算法的实时性能研究[硕士学位论文]. 沈阳: 东北大学, 2014.
20 张文涛. 基于I/O性能的虚拟机资源调度算法研究[硕士学位论文]. 武汉: 华中科技大学, 2013.
21 陈慧星. 多核环境下虚拟机调度算法研究[硕士学位论文]. 长沙: 湖南大学, 2013.