第32卷 第24期 计 算 机 工 程 2006年12月
Vol.32 No.24 Computer Engineering December 2006
·工程应用技术与实现· 文章编号：1000—3428(2006)24—0251—03 文献标识码：A 中图分类号：TP309+.2

基于 Xen 的 X86 虚拟机性能调优

杨晓伟，廖光灯，胡越明
(上海交通大学电子信息与电气工程学院，上海 200030)

摘 要：目前针对虚拟机环境的性能测试工具很少，测试主要还停留在运行一些基准测试程序的基础上，对虚拟机性能的调优帮助有限。文章提出了一个 Xen 虚拟机环境下的性能调试工具 Exentrace，介绍了其 Lasylog 和 Smartfilter 两大特性。演示了利用 Exentrace 在不同负载的虚拟机上收集、分析数据。利用这些数据发现并解决了一个关键的性能问题。

关键词：Xen VT-x；Exentrace；虚拟机

Performance Tuning of Xen-based X86 Virtual Machine

YANG Xiaowei, LIAO Guangdeng, HU Yueming
(School of Electronic Information and Electric Engineering, Shanghai Jiaotong University, Shanghai 200030)

【Abstract】 Now there are few efficient ways to tune the performance of virtual machine, excepts utilizing those general benchmark tools. This paper presents Exentrace, a low level Xen virtual machine performance tuning tools, with two keys features: Lasylog and Smartfilter. It is used to collect and parse data of virtual machines with different workloads. Looking into the data, it successfully resolves a critical performance issue.

【Key words】 Xen VT-x; Exentrace; Virtual machines

1 概述

从IBM在其S360 大型机上第 1 次实现虚拟机模型算起，虚拟机的发展已经有近 40 年的历史了[1]。互联网发展起来以后，随着新兴的虚拟机应用不断出现，虚拟机相关技术得到不断发展，可以说虚拟机正在经历一次复兴。同时，随着X86服务器市场的快速增长，X86 虚拟机更是为人们看好。

虽然目前虚拟机的实现平台和实现细节多种多样，但基本上都是基于图 1 这个经典的虚拟机模型。虚拟机管理器(Virtual Machine Monitor，VMM)介于物理硬件和虚拟机之间，为每个虚拟机虚拟出一套独立于实际硬件的虚拟硬件环境(包括 CPU，内存，I/O 设备)。

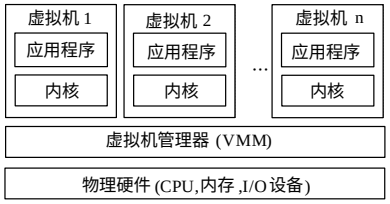


图 1 经典虚拟机模型

X86 作为一个从面向PC市场发展起来的体系结构，并没有考虑对虚拟机的支持，这确实带来一些问题[2]。为了绕开这些问题，VMware、VirtualPC 等 X86 下的纯虚拟(Full virtualization)实现采用实时的机器指令翻译的方法，动态修改虚拟机中编译好的二进制内核。该做法的好处是对操作系统的开发透明，但它引入了新的复杂性和实时的开销。

Xen[3]是一个开源的半虚拟(Para-virtualization)[4]实现，通过少量修改虚拟机中的内核，让其主动与VMM协同工作来达到很高的性能。但它需要修改少量内核代码的缺点，让在虚拟机中运行Windows这样非开源的操作系统变得不可行。

市场的驱动催生了VT-x[5]，这是Intel在芯片级对虚拟机提供支持的一种技术。VT-x大大提升了VMM对虚拟机的掌控能力，而且充分考虑了如何有效减小虚拟机的开销。结合Xen高性能、开源的优点和VT-x的优良特性和基于Xen的X86 纯虚拟机解决方案，本文提出了一个在其基础上的性能调试、分析工具Exentrace。

2 工具设计

2.1 Xen+VT-x 虚拟机模型

为了更好地理解工具，首先介绍所基于的虚拟机模型。

效法 IBM 大型机，VT-x 提供了 2 个运行环境：根(Root)环境和非根(Non-root)环境。根环境专门为 VMM 准备，而非根环境作为一个受限环境用来运行多个虚拟机。运行环境之间可以相互转化，从根环境到非根环境叫 VMEntry；从非根环境到根环境叫 VMExit。每个虚拟机对应一个 CPU 维护的 VMCS 数据结构，其中记录根环境配置、非根环境配置、VMExit 发生控制、VMExit 原因等信息。

在图 2 所示的方案中，VMM 运作在根环境的 ring0。作为 Xen 的一个特色，根环境中还有一个比较特殊的管理虚拟机 Domain0。它的内核是修改过的 Linux(称为 Xen0)，通过 VMM 提供的 Hypercall 跟 VMM 协同完成管理工作。它的用户空间有个叫 Control Panel(CP)的进程，用户可以通过它来创建、动态配置、关闭其它普通虚拟机 DomainN。VMM 允许 Domain0 访问硬件，其用户空间还有一个特殊进程 Device Model(DM)，用于为 DomainN 虚拟 I/O 外设的工作。DomainN 通常运行在非根环境不受到打扰，只有当 VMM 设置的 VMExit 条件满足时才由 CPU 执行环境切换。重获控制的 VMM 通过读取 VMCS 判断 VMExit 原因。为了尽量把 VMM 做得精简、可靠，VMM 中没有外设驱动，当一次访问硬件的 VMExit 发生时，VMM 需要 DM 的帮助，由其完成外设的

作者简介：杨晓伟(1979－)，男，硕士，主研方向：计算机系统结构；廖光灯，硕士；胡越明，副教授

收稿日期：2005-12-25 **E-mail：**bit@sjtu.edu.cn

虚拟。DM 平时阻塞等待 DomainN 对外设的访问请求。当 VMM 发现 DomainN 要访问硬件，比如读某个 I/O 端口时，先阻塞该 DomainN，并调度到 Domain0，把该请求往上传递给 DM，DM 从阻塞中返回，以发生 VMExit 的那个 DomainN 的名义访问硬件，把结果传给 VMM，VMM 再写入 VMCS 中，最后解除 DomainN 的阻塞，调用 VMRESUME 返回 DomainN。

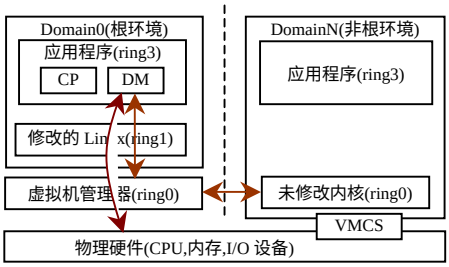


图 2 Xen+VT-x 模型

2.2 Exentrace

虽然 VT-x 技术尽可能细分 VMExit 的条件，以求通过 VMM 配置让 DomainN 尽可能少发生 VMExit，但 VMExit 还是无法避免的。而每次 VMExit/VMEntry 时都要利用 VMCS 切换环境，开销还是相当大的。因此如果有一个工具可以记录下 DomainN 运行过程中各种 VMExit 的次数和各个阶段的处理时间，那么对调试 Xen 虚拟机性能将提供非常大的帮助。

设计工具之初发现了 Xen 自带的 Xentrace。它由开发者在 VMM 中放置记录点(如虚拟机调度、Hypercall 操作)。每条记录都以一定的格式写到一个固定的环形缓冲区里面。Domain0 的应用程序 Xentrace 启动时通过调用一个特殊的 Hypercall 把这段物理内存映射到自己的虚拟空间中，接着就以一定的时间间隔读出缓存中的数据，写到一个日志文件中。开发者通过该记录文件就可以分析实际的 VMM 操作。

2.3 Lasylog

为了记录 VMExit/VMEntry 事件，很容易想到利用 Xentrace 的方法，在 VMM 中插入记录点。VMExit 发生时的入口函数 vmx_vmexit_handler 中包括一个很大的 switch 语句，通过 VMCS 读取 VMExit 原因，调用不同的处理子函数。有些处理函数很短，简单地执行几行代码即可返回；有些需要 DM 的帮助，这时就需要在虚拟机之间调度。每种 VMExit 都需要记录尽可能多的信息，而这些信息只有在各个处理子函数中一步步分析才能收集到，因此自然想到的是在各个处理子函数中增加一个记录点，但这样有多少种 VMExit 就要增加多少个记录点，分布很凌乱，管理也不方便。

VT-x 规定 VMExit 与 VMEntry 是交错发生的，也就是说 VMExit 发生后，VMEntry 前不会再次发生 VMExit，而且每次 VMExit 处理完毕后发生 VMEntry 返回的地点也是固定的，即 vmx_asm_do_resume，这样就想到了另一种放置记录点的方式，称之为 Lasylog：

(1)在 VMExit 发生的最初时放置一个记录点，除了记录下 VMExit 发生时间，不管其他额外信息；

(2)在各个处理函数中获得各种 VMExit 信息的同时把它们集中收集到一个全局数组中。在 VMEntry 前，也就是一次 VMExit 处理即将结束时才放置另一个记录点，把当前时间和全局数组中的数据组装成另一条记录。这样做的好处是：每次 VMExit 的处理都严格对应到有且仅有进/出 2 条记录 .通过这 2 条记录之间的时间戳差值，可以获得尽可能准确的 VMExit 处理开销。

2.4 Smartfilter

但最初实验的结果却并没有想象的那么理想，提取出来的记录并不是严格按照一进一出的顺序排列的，而且其中掺杂着其他诸如虚拟机调度之类我们不感兴趣的记录。通过分析发现了其中的原因：VMExit/VMEntry 频繁发生将产生大量记录填充缓冲区，而 VMM 的环形缓冲区的大小很有限，如果 Xentrace 不能及时把数据取掉，缓冲区就会溢出，把以前的记录覆盖掉。对此可能的解决方法有：

(1)增大缓冲区，从默认的为每个 CPU 分配 10 个页面大小的空间增大到 100 个页面；

(2)提高轮询频率，从默认轮询频率为 100ms 提高到 5ms。

实验表明这确实一定程度上解决了缓冲区溢出的问题，但这样做也带来了明显的负面影响：增加了系统开销，而且大量的原始数据导致了很长的解析时间。

矛盾的焦点是短时间产生了大量的记录，如果能有效地控制记录的产生数目，问题也就解决了，为此在 Exentrance 中实现了另一项特性：Smartfilter。

如图 3 所示，Smartfilter 定义了一套 Exentrace 和 VMM 都能识别的规则，在启动过程中用户以参数形式把过滤器传递给 Exentrace，由其通过 Hypercall 通知 VMM。过滤器包括：选择记录某些类事件；在有多个逻辑 CPU 的时候(如 HT、多核、SMP)只记录某些 CPU 上的事件。只有通过过滤器筛选之后的记录才会被放到缓存区中。这种灵活的设置为以后调试虚拟定时器开销或者虚拟内存 Page fault 等情况提供了很好的接口。

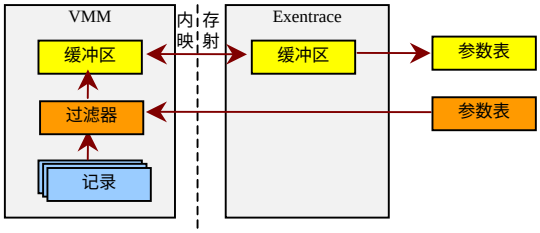


图 3 Exentrace 模型

另外，Exentrace 还包括一套脚本用于分析原始记录文件，统计出一定时间间隔内各种 VMExit 事件发生的次数，占总次数的百分比，每种 VMExit 消耗时间的平均值、方差，占总时间的百分比。

3 性能分析

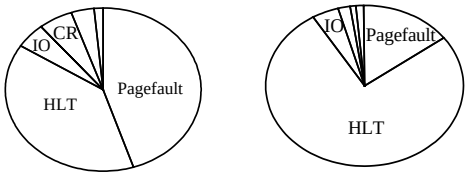
接下来利用 Exentrace 来调试 Xen 虚拟机的性能。为此我们设计了 2 个实验。

(1)在虚拟机中启动一个 Linux 后，运行一个 LTP(Linux Test Project)中的基本套件 Quickhit，同时用 Exentrace 收集数据。LTP 是 Linux 环境下测试内核最常用的一组套件，用它做实验负载得到的数据具有一定的代表性。

(2)在同样长度的时间内在虚拟机中启动 Linux 到运行级别 3，不给它任何实际任务，同时用 Exentrace 收集数据。

实验在一台带有 VT-x 功能的 Intel Prescott 机器上完成，Exentrace 启动时设置过滤器只记录所有逻辑 CPU 上的 VMExit/VMEntry 事件，而过滤掉所有其它类型记录，这样既有效地降低了产生记录的数量，又减少了分析数据的时间。

图 4 的实验结果饼图中，每块表示该类型的 VMExit 次数占总发生次数的比例。可见 VMExit 的主要原因按百分比从高到低排列分别是：缺页错误，执行 HLT 指令，I/O 操作，访问 CR 系列寄存器，而且有负载情况下因为缺页错误产生的 VMExit 明显增多。



(a)实验1(Quickhit)　　(b)实验2(无负载)

图 4 实验数据

最引人关注的是 HLT 指令引起的 VMExit，在 2 个实验中都占了很大比例，尤其在实验 2 中，占到了绝对多数。在 Linux 内核中，当系统中没有可运行的进程时，Idle 进程被调度到，它循环执行 HLT，告诉 CPU 停止运行直到下一个外部事件(中断)到来。而在虚拟机的设计中，显然不希望一台虚拟机处于空闲状态就让整台物理机器停止工作，因为这时很可能其它虚拟机上有任务需要运行。所以一旦 Idle 进程调用 HLT 指令发生 VMExit，VMM 调度算法选择运行绑定在该逻辑 CPU 上的其它虚拟机。但如果一个逻辑 CPU 上只绑定了一台虚拟机(我们的实验正是这种情况)又会怎样呢？这时，VMM 仍将调度到那台空闲的虚拟机，VMEntry 返回后又进入了 Idle 进程，Idle 调用 HLT 又产生 VMExit，这样如此反复，频繁发生 VMExit/VMEtry，却没有完成任何实际工作。

症结一旦发现，解决方法似乎很顺理成章：当发生 HLT 引起的 VMExit 时，VMM 把该虚拟机从所在逻辑 CPU 的运行队列里面摘除，直到其有一个外部事件到来。此时如果运行队列里面还有其他虚拟机，则调用调度算法选择下一个虚拟机运行，否则停止该逻辑 CPU，这样就省下了很多宝贵的 CPU 周期。

4 展望与结论

我们的工具虽然已经足够灵活与强大，利用它可以显示和分析很多 Xen 虚拟机底层性能方面的数据，但现在的记录点仅限于在 VMM 内，对于分析像一次 I/O 产生的 VMExit 操作过程，还不能提供 VMM 中的开销、Domain0/DomainN 之间切换的开销、DM 的执行开销这样更细分的信息。一种解决方案是在 Domain0 的内核 Xen0 和 DM 中增加记录点，再通过 Exentrace 汇总、组织这些记录。但因为这要 VMM、Xen0 和应用程序 DM 这 3 个空间协同配合，在实现过程中更要格外小心。如果组织事件的先后按照记录事件的时间戳为准，首先要保证的是这 3 个空间的时间精确一致，TSC 应该是个不错的时间源，但如果涉及到多 CPU，还要考虑 TSC 的同步等问题。

本文提出了一个 Xen 虚拟机下底层的性能分析工具 Exentrace 及其两大关键特性，并演示了如何以实验的形式，通过 Exentrace 收集、分析数据，从而切实有效地解决了 Xen 虚拟机关键的性能问题。相信随着 X86 虚拟机的流行，像这样的底层性能分析工具将对虚拟机起到越来越重要的作用。

参考文献

1 Creasy R J. The Origin of the VM/370 Time-sharing System[J]. IBM Journal of Research and Development, 1981, 25(5): 483-490.

2 Robin J S, Irvine C E. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor[C]. Proc. of the 9th USENIX Security Symposium, Denver, Colorado, USA, 2000-08-14.

3 Barham P, Dragovic B, Fraser K, et al. Xen and the Art of Virtualization[C]. Proc. of SOSP'03, ACM, 2003.

4 Whitaker A, Shaw M, Gribble S D. Denali: Lightweight Virtual Machines for Distributed and Networked Applications[R]. University of Washington, Tech Rep: 02-02-01, 2002.

5 Intel. Vanderpool for IA-32 Processor(VT-x) Preliminary Specification [R]. 2005.

(上接第 205 页)

200，400。由于数据是随机抽取，因此其中含有大量的模糊、残缺车牌。第 2 部分实验采用的数据基本上不含模糊不清的车牌和残缺车牌，其他条件与第 1 部分实验相同。

表 1 第 1 部分实验识别率比较（单位：%）

车牌数量	本文方法				B		
	a	b	c	d	a	b	c
100	70	85	90	99	80	85	85
200	90	98	99	99	80	85	85
400	97	98	99	99	80	85	85

表 2 第 1 部分实验识别时间比较（单位：ms）

车牌数量	本文方法				B		
	a	b	c	d	a	b	c
100	130	65	28	12	268	272	269
200	159	76	36	11	270	271	272
400	170	78	35	12	269	270	268

表 3 第 2 部分实验识别率比较（单位：%）

车牌数量	本文方法				B		
	a	b	c	d	a	b	c
100	90	95	99	99	99	99	99
200	99	99	99	99	99	99	99
400	99	99	99	99	99	99	99

表 4 第 2 部分实验识别时间比较（单位：ms）

车牌数量	本文方法				B		
	a	b	c	d	a	b	c
100	140	60	38	13	261	264	260
200	170	75	39	10	260	263	261
400	171	76	37	12	262	261	262

5 结论

通过表 1~4 的实验数据可以看出，本文基于 Rough 集的方法对含有大量模糊和残缺车牌的数据有较好的识别率；对于比较清晰的数据有不低于 B 方法的识别率 在识别时间上，比 B 方法要快。

参考文献

1 王　成，王润生. 基于 MAP 框架的图像序列超分辨率和模板匹配[J]. 计算机学报, 2003, 26(8): 961-967.

2 Wang Jue, Wang Ju. Reduction Algorithms Based on Discernibility Matrix: The Ordered Attributes Method[J]. J. Comput. Sci. & Technol., 2001, 16(6): 489-504.

3 Pawlak Z. Rough Sets-Theoretical Aspects of Reasoning about Data[M]. Boston, London Dordrecht: Kluwer Academic Publishers, 1992.

4 支天云，苗夺谦. 二进制可辨矩阵的变换及高效属性约简算法的构造[J]. 计算机科学, 2002, 29(2): 140-142.

5 王希雷，王　磊，马　涛等. 一种高效属性约简算法[J]. 微机发展, 2002, 12(增刊): 12-16.

6 王国胤. 决策表核属性的计算方法[J]. 计算机学报, 2003, 26(5): 611-615.

7 赵　军，王国胤，吴中福等. 基于粗集理论的特征子集选择算法[J]. 计算机科学, 2002, 29(11): 83-86.

8 李　佐，王姝华，蔡士杰. 基于特征行必要-充分匹配的字符识别方法[J]. 软件学报, 2002, 13(1): 85-91.