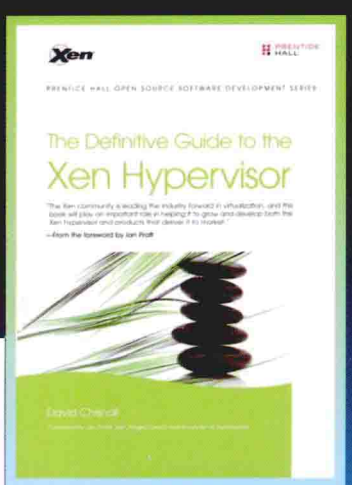
Xen 虚拟化技术完全导读

[美] Chisnall David 著
张 炯 吕紫旭 胡彦彦 文成建 译



THE DEFINITIVE GUIDE TO THE XEN HYPERVISOR

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

Xen 虚拟化技术完全导读

The Definitive Guide to The Xen Hypervisor

[美] Chisnall David 著

张 炯 吕紫旭 胡彦彦 文成建 译

北京航空航天大学出版社

内 容 简 介

本书主要介绍了目前IT技术热点虚拟化技术领域中最受关注的虚拟化系统软件Xen，包括在Xen中对于各种虚拟化技术的实现的分析，尤其基于研究热点的考虑用大量篇幅专门讨论了虚拟化的I/O和Xen内核的一些关键技术，并讨论了未来的方向，其中又以特别的章节分析了硬件虚拟化(HVM)。

读者对象以研究虚拟化技术的科研人员和工程人员为主，尤其是从事系统软件分析和开发的以及服务器端高可靠性软件研发的人员。

图书在版编目(CIP)数据

Xen虚拟化技术完全导读 /（美）大卫（David，C.）著；张炯等译. -- 北京 ：北京航空航天大学出版社，2014.1

ISBN 978-7-81124-563-9

Ⅰ. ①X… Ⅱ. ①大… ②张… Ⅲ. ①数字技术 Ⅳ. ①TP391.9

中国版本图书馆CIP数据核字(2014)第002889号

北京市版权局著作权登记号：图字：01-2008-3388

Xen虚拟化技术完全导读

The Definitive Guide to The Xen Hypervisor

[美] Chisnall David 著

张 炯 吕紫旭 胡彦彦 文成建 译

责任编辑 卫晓娜

*

北京航空航天大学出版社出版发行

北京市海淀区学院路37号(邮编100191) http://www.buaapress.com.cn

发行部电话：(010)82317024 传真：(010)82328026

读者信箱：emsbook@gmail.com 邮购电话：(010)82316936

涿州市新华印刷有限公司印装 各地书店经销

*

开本：710×1 000 1/16 印张：15 字数：320千字

2014年1月第1版 2014年1月第1次印刷 印数：3 000册

ISBN 978-7-81124-563-9 定价：45.00元

译者序

虚拟化技术(Virtualization Technology)无疑是最近几年以及未来 10 年计算机系统软件的热点。虽然它并非新出现的概念,甚至可以说是已经出现 40 年之久,但是它的热度似乎昭示 IT 界,一波新的技术浪潮将伴随着虚拟化而席卷计算系统的主要组成部分乃至各个角落。从高性能到嵌入式,从云计算到移动,从处理器到存储,从显示到无线,虚拟化技术在各种相当成熟或仍然活跃的系统中迸发出全新的解决方案和热点结合,这种情景让人感慨虚拟化的洗礼也许会全面颠覆计算机系统的应用模式乃至开发模式,未知的全新的计算机系统将脱胎换骨于这个过程,也带来可以参与其中的无数机会。

很巧合的是,虚拟化技术并非此时唯一的主角,多核(Multi-core,包括众核 Many-core)技术的出现使得 IT 的技术舞台上,两颗新星交相辉映,堪称一时瑜亮。所不同的是,这一对瑜亮完全是相互配合的组合。毫无疑问,虚拟化应用需要较之以往更多的系统计算能力来支撑,而多核处理器和多核技术则适时现身,弥补了这一需求。硬件计算能力的提升,系统资源管理能力的进步,使得人们可以基于这两个热点技术去构造属于个人的高性能服务器、工作站和计算集群,构造之前难以得到的实验环境。所以,可以预见,这波浪潮除了在计算机系统自身的演化过程中有巨大影响,还将进一步影响各种计算机应用系统的应用模式,从而以全新的方式影响人们的日常生活。具体怎样,我们还无从详细的全面预测,各位读者可以在了解两者以后,结合自己的专业和工作生活背景去开放的思考一下,也许就会出现一个很好的创意乃至创新。

本书主要介绍了目前最炙手可热的虚拟化系统软件 Xen,其作者 David Chisnall 应该算是距离 Xen 的剑桥核心团队最近的专业人士之一,按照 XenSource 的 Xen 项目领导者和奠基人 Ian Pratt 的评价,"他(David Chisnall)的这本书是有关 Xen 系统管理程序目前已经面世的书籍中,最为深入、详尽的一本,完全配得上书名中权威指南的说法"。从翻译完这本书的感受来说,我们很同意他这个评价!

参与翻译本书的译者均为从事虚拟化技术前沿研究的年青人,他们是吕紫旭、胡

彦彦、文成建、吕孟轩、刘铭。他们了解 Xen 的过去和现在，都对 Xen 抱有巨大的期望，并对 Xen 进行着各种修改的尝试，试图在其中加入自己的创意。很显然的，他们是这本书的第一批受益者，并愿意将这本书介绍给各位读者。如果有一天你发现他们围绕 Xen 开展工作得到的成果，这将不是意外的事情。与读者共勉！

张　炯

于北京航空航天大学新主楼

前 言

伴随着 Xen3.1 版本的发布，Xen 社区(Community)给业界贡献了最先进的(虚拟机系统)管理程序(Hypervisor)，并成为虚拟化技术开源软件的标准。Xen 社区获得了世界范围内超过 20 家领先的 IT 厂商的支持，得益于遍布世界各地的厂商和研究机构的研究贡献，已经成为业界虚拟化技术创新的推动力量。

Xen 的持续发展和优异性能是对 Xen 项目组件策略(Component Strategy)成功最好的证明。Xen 项目不止是开发了一个完整的开源产品，还给出了一种有效的(虚拟机)系统集成方法，即将 Xen 系统管理程序作为系统的“发动机”集成到不同的产品和项目中。例如，Xen 可以作为系统管理程序与很多操作系统内置的集成，包括 Linux、Solaris 和 BSD，也可以像 XenSorces 的 XenEnterprise 作为独立的虚拟化平台打包发布。这使得 Xen 可以服务于很多不同的虚拟化使用场合和用户需求。

Xen 支持很多类型的计算机体系结构，从拥有成千上万个安腾处理器的超级计算机系统，到 PowerPC 以及业界标准的 x86 服务器和客户端，甚至基于 ARM9 的 PDA。Xen 项目对跨体系结构的支持和对多种操作系统的支持是其另一个强有力的特征，使其影响到其他一些企业专有产品的设计，包括将要发布的微软视窗系统管理程序(Microsoft Windows Hypervisor)，也使其可以受益于来自处理器、芯片组和 fabric 厂商提供的硬件辅助的虚拟化技术。Xen 项目还在分布式管理任务组织(Distributed Management Task Force，DMTF)中有很多贡献，参与发展面向虚拟化系统的工业标准管理框架(ISMF)。

Xen 系统管理程序的不断成功很大程度上依赖于 Xen 社区中大量的高水平开发者，他们既能为 Xen 项目做出贡献和积累，也可以将这些技术应用于他们自己的产品中。尽管有一些书籍说明了在特定厂商产品的具体环境中如何使用 Xen，但是虚拟机技术开发社区对于 Xen 系统管理软件本身内在机制的权威性指南类书籍依然有很大的需求。继续使用 Xen 系统管理软件被看作是“发动机”这个比喻的话，这就好比有很多书籍用于说明集成了 Xen 的“汽车”，却没有手册用于说明如何使用和维护“发动机”本身。因而，这本书的出版对于 Xen 社区和围绕这个社区的业界厂商来说无疑是一件重要的事情。

David Chisnall 在书中给出了用户和读者深入 Xen 的内部所必须了解的专家级分析，使得他们可以理解 Xen 复杂的子系统，并整理出工作文档。拥有计算机科学

的博士学位，并且作为活跃的系统软件开放者，David 已经将 Xen 的各种复杂问题简明化，使得熟练的系统开发人员可以准确的了解 Xen 的工作机制以及如何与关键的硬件系统接口配合，甚至于如何去开发 Xen。为了完成这项工作，David 花费了很可观的时间与 XenSource 在英国剑桥大学的核心团队一起工作。在那里，他以独特的视角阐述了 Xen 的发展历史、体系结构和内在工作机制。毫无疑问，他的这本书是有关 Xen 系统管理程序目前已经面世的书籍中，最为深入、详尽的一本，完全配得上书名中“权威指南”的说法。

我希望，也相信这本书可以为 Xen 项目的更进一步发展和获得世界范围内的认同和接受做出重要的贡献。开源虚拟化技术的前途充满机会，而开源社区是重要的基础，快速的创新和不同解决方案的发布在这样的基础上将不断涌现。Xen 社区正在虚拟化领域不断引领业界向前发展，而这本书必将在帮助 Xen 社区成长、开发 Xen 系统管理程序和相关产品投放市场的过程中起到重要的作用。

Ian Pratt

Xen Project Lead and Founder of XenSource

目 录

第 1 章 虚拟化技术的现状 …… 1

1.1 什么是虚拟化技术 …… 1
1.1.1 CPU 的虚拟化 …… 2
1.1.2 I/O 的虚拟化 …… 3
1.2 为什么要虚拟化 …… 4
1.3 历史上第一台虚拟机 …… 5
1.4 x86 架构虚拟化的问题 …… 6
1.5 一些解决 x86 架构虚拟化问题的方案 …… 6
1.5.1 二进制翻译 …… 7
1.5.2 泛虚拟化 …… 8
1.5.3 硬件辅助虚拟化 …… 10
1.6 Xen 的理念 …… 11
1.6.1 方案和机制的分离 …… 11
1.6.2 做得越少越好 …… 12
1.7 Xen 的系统结构 …… 13
1.7.1 Hypervisor,操作系统,应用程序之间的关系 …… 13
1.7.2 Domain 0 的角色 …… 15
1.7.3 非特权级的 Domain …… 17
1.7.4 HVM 的 Domain …… 18
1.7.5 Xen 的结构配置 …… 18

第 2 章 探索 Xen 虚拟体系结构 …… 22

2.1 作为泛虚拟化客户端启动 …… 22
2.2 利用特权级限制操作 …… 23
2.3 用超级调用取代特权指令 …… 24
2.4 探索 Xen 事件模型 …… 27
2.5 与共享内存进行通信 …… 28

2.6 拆分设备驱动模型 …… 29
2.7 VM 生命周期 …… 30
2.8 练习:最简单的 Xen 内核 …… 31
2.8.1 客户机入口点 …… 33
2.8.2 把所有内容放在一起 …… 35

第 3 章 理解 Shared Info Pages …… 39

3.1 获取启动时钟信息 …… 39
3.2 Shared Info Page …… 42
3.3 Xen 中的时间管理 …… 44
3.4 练习:实现函数 gettimeofday() …… 45

第 4 章 使用授权表(Grant Table) …… 49

4.1 内存共享 …… 49
4.1.1 映射(Mapping)一个页面 …… 51
4.1.2 domain 间的数据传递(Transferring) …… 52
4.2 设备 I/O 环 …… 54
4.3 授权以及撤销授权 …… 55
4.4 练习:映射授权页面(granted page) …… 57
4.5 练习:在 VM 之间共享内存 …… 59

第 5 章 Xen 的内存管理 …… 62

5.1 x86 环境下的内存管理 …… 62
5.2 伪物理地址模型(Pseudo - Physical Memory Model) …… 65
5.3 32 位 x86 系统中的分段模式 …… 66
5.4 使用 Xen Memory Assist …… 67
5.5 使用 Ballon Driver 控制内存使用 …… 69
5.6 其他内存操作 …… 70
5.7 更新页表 …… 73
5.7.1 创建新的虚拟机(VM)实例 …… 77
5.7.2 处理页故障 …… 77
5.7.3 暂停(suspend)、恢复和迁移 …… 78
5.8 练习:映射 Shared Info Page …… 79

第 6 章 理解设备驱动 …… 80

6.1 分离设备模型 …… 80

6.2 将驱动程序移出 Domain 0 …… 82
6.3 理解共享存储器环形缓冲区 …… 83
6.3.1 分析 Xen 的实现 …… 85
6.3.2 通过内存栅障(Memory Barriers)实现顺序操作 …… 87
6.4 通过 XenBus 连接设备 …… 88
6.5 处理来自消息的通知 …… 90
6.6 通过 XenStore 进行配置 …… 90
6.7 练习:控制台设备 …… 91

第 7 章 使用事件通道 …… 96

7.1 事件和中断 …… 96
7.2 处理陷阱(Trap) …… 96
7.3 事件类型 …… 99
7.4 请求事件 …… 100
7.5 绑定事件通道到 VCPU 上 …… 103
7.6 绑定通道上的操作 …… 104
7.7 获取通道状态 …… 105
7.8 屏蔽事件 …… 106
7.9 事件和调度 …… 108
7.10 示例:一个完整的控制台驱动 …… 109

第 8 章 深入学习 XenStore …… 116

8.1 XenStore 接口 …… 116
8.2 浏览 XenStore …… 117
8.3 XenStore 设备 …… 119
8.4 读和写一个键 …… 121
8.4.1 用户空间方法 …… 122
8.4.2 从内核调用的方法 …… 124
8.5 其他操作 …… 132

第 9 章 支持核心设备 …… 133

9.1 虚拟块设备驱动 …… 133
9.1.1 设置块设备 …… 134
9.1.2 数据传输 …… 136
9.2 使用 Xen 网络 …… 139
9.2.1 虚拟网络接口驱动 …… 140

9.2.2 设置虚拟接口 …… 140
9.2.3 发送和接收 …… 141
9.2.4 NetChannel2 …… 144

第 10 章 其他 Xen 设备 …… 147

10.1 CD 的支持 …… 147
10.2 虚拟帧缓冲器(Frame Buffer) …… 147
10.3 TPM 驱动 …… 152
10.4 原生硬件 …… 152
10.4.1 PCI 支持 …… 153
10.4.2 USB 设备 …… 155
10.5 添加新的设备类型 …… 155
10.5.1 广播设备 …… 156
10.5.2 设置环形缓冲区 …… 156
10.5.3 困 难 …… 157
10.5.4 访问设备 …… 158
10.5.5 设计后端驱动 …… 159

第 11 章 Xen API …… 162

11.1 XML - RPC …… 162
11.1.1 XML-RPC 数据类型 …… 162
11.1.2 远程过程调用 …… 164
11.2 探索 Xen 接口层次 …… 164
11.3 Xen API 类 …… 166
11.4 Xend 的功能 …… 169
11.5 Xm 命令行 …… 171
11.6 Xen CIM 提供者 …… 172
11.7 练习:枚举正在运行的虚拟机 …… 173
11.8 总 结 …… 177

第 12 章 虚拟机调度 …… 178

12.1 调度器接口概述 …… 178
12.2 历史上的调度器 …… 180
12.2.1 SEDF …… 181
12.2.2 Credit 调度器 …… 182
12.3 使用调度器 API …… 183

12.3.1 运行一个调度器 …… 184
12.3.2 Domain 0 交互 …… 187
12.4 练习:添加一个新的调度器 …… 188
12.5 总 结 …… 191

第 13 章 HVM …… 192

13.1 运行未经修改的操作系统 …… 192
13.2 Intel VT - x 和 AMD SVM …… 194
13.3 HVM 设备支持 …… 195
13.4 混合虚拟化 …… 196
13.5 BIOS 仿真 …… 199
13.6 设备模型和传统的 I/O 仿真 …… 200
13.7 半虚拟化 I/O …… 201
13.8 Xen 中 HVM 支持 …… 202

第 14 章 未来的发展方向 …… 206

14.1 真实到虚拟,周而复始 …… 206
14.2 仿真和虚拟化 …… 207
14.3 移植的努力 …… 207
14.4 桌 面 …… 209
14.5 功耗管理 …… 211
14.6 关于 Domain0 的问题 …… 213
14.7 Stub 域 …… 215
14.8 新的设备 …… 216
14.9 特殊的架构 …… 217
14.10 前 景 …… 219

附录 泛虚拟化客户操作系统移植概述 …… 221

A.1 Domain 创建工具(Domain Builder) …… 221
A.2 启动环境 …… 222
A.3 设置虚拟中断描述符表(IDT) …… 222
A.4 页表管理 …… 223
A.5 驱 动 …… 223
A.6 Domain0 的责任 …… 224
A.7 效 率 …… 224
A.8 小 结 …… 225

第1章

虚拟化技术的现状

Xen是一个虚拟化技术工具，这代表什么含义呢？这一章将探索虚拟化技术的历史，以及过去和现在人们都在不断发现虚拟化技术实用性的一些原因。其中，本章将特别关注x86，或者说IA32架构，讨论为什么在虚拟化这类架构的机器中会遇到一些问题和限制，以及围绕着这些问题和限制，对由Xen以及其他的虚拟化系统提供的一些可能的解决方案展开讨论。

1.1 什么是虚拟化技术

虚拟化技术从概念上看非常类似于仿真技术(emulation)。在仿真时，一个系统"假扮成"另一个系统。而在虚拟化时，一个系统"假扮成"两个或者多个相同的系统。

大多数现代操作系统都已经包含了一个简单的虚拟化系统。每一个正在运行的进程都认为它是系统唯一运行的进程，这是因为CPU和内存被虚拟化了。如果一个进程尝试着占用所有的CPU时间，那么一个现代的操作系统将会阻止它这样做，并且让其他进程也公平地享用CPU时间。同样地，一个正在运行的进程有它自己特有的虚拟地址空间，操作系统将其映射到物理内存上，由此给该进程一个错觉，让其觉得自己独享内存。

操作系统也经常虚拟化硬件设备。一个进程可以通过使用伯克利套接字应用程序接口(Berkeley Sockets API)(或者别的等价物)在不用担心其他应用程序干扰的情况下访问一个网络设备。一个视窗系统或一个虚拟终端系统向屏幕和输入设备也提供了类似的多路复用技术。

所以，大家每一天都在使用着某种形式的虚拟化技术，并且也能感觉到这是非常实用的。这种技术带来的空间隔离常常能够防止一个应用程序受到来自其他应用程序的bug，或者恶意行为的影响。

不幸的是，并不是只有应用程序才会包含bug，操作系统同样也有bug，并且这通常会使得应用程序级的隔离不起作用。即使在没有bug的情况下，提供一个比操作系统提供的隔离层次更深的隔离也是非常容易做到的。

1.1.1 CPU的虚拟化

虚拟化一个CPU从某种程度上来看是非常简单的。一个进程独占CPU运行一段时间，然后被中断。此刻的CPU状态被保存，然后另一个进程开始运行。过了一段时间之后，原先被中断的进程又恢复运行。

在现代操作系统中，上述过程每隔大约10 ms发生一次。然而值得注意的是，虚拟的CPU和物理CPU是不一样的。当操作系统正在运行、交换进程的时候，CPU处在特权级模式(privileged mode)。此时系统允许某些特定操作的执行，如直接通过物理地址访问内存(这个操作通常是不被允许的)。对于完全虚拟化CPU，Popek和Goldberg在他们1974年的论文“Formal Requirements for Virtualizable Third Generation Architectures。”[①]中提出了一整套需要满足的要求。他们首先将CPU的指令划分为3类：

(1) 特权级指令：指的是那些可能运行在特权级模式，但是一旦退出特权级模式之后将发生陷入的指令。

(2) 控制敏感指令：指的是那些尝试着改变系统资源配置的指令，比如更新虚拟地址到物理地址映射的指令、与设备通信的指令、操作全局配置寄存器的指令等。

(3) 行为敏感指令：指的是那些根据资源配置的不同有不同的表现行为的指令，包括所有对虚拟地址的load和store操作的指令。

为了让一个体系结构可被虚拟化，Popek和Goldberg认为所有的敏感指令必须同时是特权级指令。直观地说，任何指令用一种会影响其他进程的方式改变机器状态的行为都必须遭到hypervisor的阻止。

DEC[②] Alpha是最容易虚拟化的架构之一。Alpha通常情况下没有特权级指令。它有一个专用的指令用于跳转到专用的固件(PALCode，privileged architecture library code，特权系统库代码)地址并进入到一个特殊的模式，在这个模式下，一些通常状态下不可见的寄存器变成了可用的。

一旦处在特权级模式下，CPU将不可以被抢占。在运行了一系列的普通指令之后，一条指令会使CPU返回到初始的模式。在执行上下文切换到内核时，用户空间的代码将产生异常，并自动跳转到PALCode。并在隐藏的寄存器中设定一个标记，然后将CPU控制权传递给内核。内核可以调用其他的PALCode指令来检查隐藏寄存器中标记的值，以及允许访问专门的一些特征，在最终调用PALCode指令之前，系统会将隐藏寄存器中的标记复位，并将CPU控制权返还给用户空间的程序。通过在隐藏寄存器中设置特权级，并在任何PALCode指令运行前检查这个特权级，

---

① 出版于 *Communication of the ACM*。

② Digital Equipment Corporation(DEC)之后重命名为Digital，并被HP公司买下，接着HP与Compaq合并。

可是使得这套机制得以扩展，从而能够非常容易的成为一个类似多特权级的机制。

特权级指令执行的每一项操作都由一组存储在 PALCode 中的指令来完成。如果用户想虚拟化 Alpha，所有需要做的只是用一组指令代替 PALCode，通过一个抽象层来传递操作。

1.1.2 I/O 的虚拟化

一个操作系统的运行不止需要 CPU，还需要主存，以及一组外设。虚拟化内存相对简单，只需要把它分割为多个区域，然后每一个访问物理内存的特权级指令发生陷入，并由一个映射到所允许的内存区域的指令所代替。一个现代的 CPU 包括一个内存管理单元(MMU)，正是内存管理单元基于操作系统提供的信息实现了上述的翻译过程。

其他的外设就较为复杂了。绝大多数外设在设计的时候并未考虑虚拟化这个因素，并且对于其中的一些外设来说，如何支持虚拟化技术并不是十分明朗。比如像硬盘这样的块设备，能够有潜力被虚拟化，采用的方法类似于主存，即被划分为多个区域，每个区域能被各自的虚拟机访问。然而对于显卡来说，问题就没这么简单了。一个简单的帧缓冲设备的做法是向每个虚拟机提供一个虚拟的帧缓冲设备，并且允许用户在它们之间进行切换或者将它们映射到物理设备上。

然而，现代的显卡要比帧缓冲设备复杂得多；它们具有 2D 和 3D 加速的功能，并且有许多内部状态。更糟糕的是，绝大部分的显卡不提供一个保存和恢复这些状态的机制，因此即使是在虚拟机之间切换也是个问题。这个问题在电源管理中就出现过。假设你正在运行一个图形用户界面(GUI)，比如 X11，一些状态也许会保存在显卡中(至少当前的视频模式会被保存)，但是当设备断电之后这些保存的状态就丢失了。所以，为了保证能够将这些状态保存在别的地方以及在需要的时候可以恢复(如使每个窗口都自我刷新)，GUI 必须要经过修改。而这对一个真实的虚拟环境来说显然是不可能的，因为虚拟化的系统不能意识到它已经同硬件断开了连接。

另一个问题来源于设备同系统交互的方式。例如数据通过 DMA 传入和传出设备。设备在由驱动程序提供的物理地址处写入一堆数据。因为设备位于通常意义上的操作系统的框架之外，所以它必须使用物理地址而不是一个虚拟的地址空间。

如果操作系统对平台有完全的控制能力，那么这个机制会工作的很好，但是如果不是就会产生一些问题。在一个虚拟化的环境中，内核运行在由 hypervisor 提供的虚拟地址空间中，而用户空间的进程以几乎同样的方式运行在内核提供的虚拟地址空间中。允许 guest OS(guest OS)内核指示设备向一个物理地址空间中的专门区域内写数据是一个严重的安全漏洞。如果内核，或者设备驱动没有意识到它们正运行在一个虚拟的环境中的话，那么情况将会更糟糕。在这种情况下，一个被信任的指向内核地址空间的地址将会被提供，但实际上指向的却是完全不同的地方。

理论上，一个 hypervisor 有可能发生陷入写入设备并将 DMA 地址重写入允许

的地址范围。实际上,这是不可行的。即使不考虑这将导致的重大的性能损失,监测一条 DMA 指令也是非常困难的。每一个设备为了同驱动程序交流都定义了自己的协议,所以 hypervisor 必须了解这些协议,解析指令流,然后执行指令代换。这比一开始就写驱动程序要花费更多的精力。

一些平台可以使用输入/输出存储管理单元(IOMMU)。它同一个标准的存储管理单元(MMU)表现出了相似的特征,即在物理地址空间和虚拟地址空间之间进行映射。不同之处在于它的应用:MMU 为运行在 CPU 上的进程执行这种映射,而 IOMMU 为设备执行这种映射。

第一个 IOMMU 出现在一些早期的 SPARC 系统中。这些系统都带有一个网络接口,这些接口没有足够的地址空间来写入所有的主存。IOMMU 的添加允许真实的地址空间页被映射到设备地址空间中。当 8 位和 16 位的 ISA 卡同 32 位系统一起使用的时候,在 x86 平台上使用的是不同的方法,即简单的为 I/O 保留一块靠近地址空间底部的内存。

AMD 的 x86-64 位的系统因为一个类似的目的也有一个 IOMMU。同 x86-64 位机器连接的很多设备可能都是传统的仅仅支持 32 位地址空间的 PCI 设备。没有 IOMMU,这些设备访问物理内存最底部的 4 GB 空间是受限制的。大多数时候当执行 mmap 系统调用或者实现虚拟内存的时候这显然是一个问题。当页故障发生时,块设备的驱动程序仅仅能通过 DMA 传输访问物理内存的底部。如果页故障发生在别的地方,那么只好使用 CPU 来负责写数据,并且是以每次一个字的速度写入正确的地址,这是非常慢的。

一个类似的机制曾在 AGP 卡中使用过。图形地址重映射表(GART)就是一个简单的 IOMMU,负责使用 DMA 将纹理结构装载进 AGP 图形卡中,并且允许这种卡轻松地使用主存。但是,这对满足虚拟化的要求并没有做多大的贡献,因为并不是所有同 AGP 卡或者 PCIe 图形卡的交互作用都是通过 GART 来完成的。事实上这主要是通过使用板载 GPU(而不是通过 BIOS 默认设置)来允许操作系统分配更多的内存给图形卡。

1.2 为什么要虚拟化

虚拟化技术的根本目的和多任务操作系统的目的是一样的,即计算机拥有不止能满足一个任务需要的处理能力。第一台计算机仅仅用于完成一个任务。第二代计算机是可编程的,这些计算机能够完成一个接一个的任务。最终,硬件变得足够快以至于一台机器在做一个任务的同时仍旧有空闲的资源。多任务机制使得利用这些空闲资源变成了可能。

现在很多组织和机构发现他们有许多的服务器都只在做一件任务,或者一个相关任务的小的集群。虚拟化技术能够将许多的虚拟服务器整合到一个单独的物理主

机上，并通过运行环境的彻底隔离而充分保障它们的安全。几个网络主机公司正在对虚拟化技术展开广泛的应用，因为该技术允许他们为每一个客户提供一个独享的虚拟主机，从而取代过去占据着数据中心磁盘阵列空间的物理主机。

有些时候情况会更糟糕。比如，一个机构需要两个或者更多的服务器来运行一个特殊的任务，以防任务失败，即使所有服务器的资源都有空闲但也只好这样。虚拟化技术能够在这里发挥作用，因为从一台物理机器上移植一个虚拟机到另一台物理机器上是相对比较简单的事情，通过物理机器保持冗余虚拟服务器镜像的同步是非常简单的。

一台虚拟机有一些特征，比如以非常低的开销进行整机克隆。如果不确定在安装一个补丁之后是否会破坏生产系统，则可以将这台机器虚拟化整机克隆，然后在虚拟机上安装这个补丁，看看会发生什么。这比试着保证一个生产机器和一个试验机都处于同样的状态要简单得多。

虚拟化技术的另一个巨大优势就是可移植性。如果物理主机硬件出现故障或者需要进行升级，那么虚拟机可以移植到另一台物理主机上。当原先的物理主机恢复正常之后，该虚拟机又可以移植回来。

能源开销低也是虚拟机技术的一个吸引人之处。一个空闲的服务器依然在消耗能源，而将多个服务器整合到一台或几台物理主机上，使之成为多个虚拟的服务机器，这样做能够给能源的消耗状况带来相当大的改观。

一台虚拟机要比一台物理机器更易于携带，用户可以把一台虚拟机的状态保存在一个 USB 闪盘中，或者一些类似于 iPod 的东西里。这样携带一台虚拟机将会比携带一台笔记本都要方便。当用户想要使用它的时候，只需要将 USB 插入电脑然后恢复虚拟机的运行就可以了。

总之，相比于在一个操作系统上运行的进程来说，一台虚拟机提供了更深层次的隔离。这使得创建虚拟应用成为了可能：即具有网络服务的虚拟机。一个虚拟的应用程序，不像其对应的物理应用程序占据着磁盘空间，并且虚拟的应用程序能够更易被复制，以及如果负荷太重可以更容易的分担在不同的虚拟机节点上（或者只是在一个大型的机器上分配更多的运行时间）。

1.3 历史上第一台虚拟机

第一台完全支持虚拟化的机器是 IBM 的 VM，它作为 IBM 360 系统项目的一部分而诞生。360 系统（System/360，S/360）的理念是向 IBM 的用户提供一个稳定的架构和简单的升级方式。IBM 提供多种具有相同体系结构的系列机，一些小型的公司或企业可以购买一台满足他们需要的小型机，并且日后也可以在不需要改动运行软件的情况下升级到大型机。

当时 IBM 确立的一个关键性的市场就是人们想要将 IBM 360 系统的机器进行