Linux/Unix
技术丛书

深度实践KVM

核心技术、管理运维、性能优化与项目实施

肖力 汪爱伟 杨俊俊 赵德禄 著

Inside Explore the KVM

Core Technology, Operation and Maintenance,
Performance Optimization, Project Implementation

- 中国运维领域里划时代著作，国内顶尖KVM专家和运维专家多年大规模生产环境实践经验总结，国内数十位运维专家鼎力支持和推荐
- 从核心技术、管理运维、性能优化、项目实施等多个维度系统、深入讲解KVM虚拟化技术的工作原理、应用方法和技术生态，并针对生产环境中的各种难题给出了解决方案

机械工业出版社
China Machine Press

1107114476@qq.com
dolphinebook.com

内容简介

本书在中国运维领域将具有“划时代”的重要意义：第一，作者团队是国内顶尖的KVM技术专家、虚拟化技术专家和运维专家，是国内第一批研究并将KVM应用到大规模生产环境的团队之一；第二，书中的所有知识、经验和案例都来自金山、盛大等国内一线互联网公司，这些公司的业务规模巨大，生产环境复杂；第三，从核心技术、管理运维、性能优化、项目实施等多个维度系统、深入讲解KVM虚拟化技术的工作原理、应用方法和技术生态，并针对生产环境中的各种难题给出了解决方案；第四，本书内容得到了来自金山、盛大、阿里、猎豹移动、触控科技、英特尔、浪潮等大型IT企业的数十位VP和运维专家的高度认可，并联袂推荐。

本书共22章，分为五篇：

第一篇（第1～8章）：虚拟化技术选型与KVM虚拟机技术。详细介绍了虚拟化技术的选型，KVM的CPU和内存虚拟化技术、磁盘虚拟化技术、网络虚拟化技术以及这些技术的生产环境应用场景，物理机转虚拟机，KVM的桌面虚拟化实践。

第二篇（第9～10章）：开源分布式文件系统在KVM中的应用。详细讲解了常见的开源分布式文件系统在KVM环境中的使用，重点讲解比较热的Ceph文件系统，包括配置、使用及常见故障处理。

第三篇（第11～13章）：开源的虚拟化管理平台在生产环境的实践经验。重点讲解了OpenStack、OpenNebula在生产环境的实践与故障处理，还有oVirt、CloudStack、Convirt、WebVirtMgr等几个管理平台的架构和理念。

第四篇（第14～21章）：KVM虚拟化项目的实施与经验。系统讲解了业务性能评估、压力模型建立、虚拟机性能测试、宿主机基准测试、业务测试环境测试验证、单机虚拟化技术、集群虚拟化技术、虚拟化正式环境上线等一套完整的虚拟化项目实施过程。

第五篇（第22章）：KVM虚拟化的典型案例。讲解了KVM虚拟化过程中比较典型的一些案例，针对案例中的问题给出了解决思路和方法，方便读者在碰到类似的问题时找到解决方案。

1107114476@qq.com
dolphinebook.com

Linux/Unix
技术丛书

深度实践KVM

核心技术、管理运维、性能优化与项目实施

Inside Explore the KVM

Core Technology, Operation and Maintenance,
Performance Optimization, Project Implementation

肖力 汪爱伟 杨俊俊 赵德禄 著

机械工业出版社
China Machine Press

1107114476@qq.com
dolphinebook.com

图书在版编目（CIP）数据

深度实践 KVM：核心技术、管理运维、性能优化与项目实施 / 肖力等著．—北京：机械工业出版社，2015.9
（Linux/Unix 技术丛书）

ISBN 978-7-111-51193-9

I. 深… II. 肖… III. 虚拟处理机 IV. TP338

中国版本图书馆 CIP 数据核字（2015）第 188798 号

深度实践 KVM
核心技术、管理运维、性能优化与项目实施

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：高婧雅　　责任校对：董纪丽
印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2015 年 9 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：35
书　　号：ISBN 978-7-111-51193-9　　定　　价：95.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

版权所有 • 侵权必究
封底无防伪标均为盗版
本书法律顾问：北京大成律师事务所　韩光 / 邹晓东

1107114476@qq.com
dolphinebook.com

Praise 本书赞誉

这本书干货太多，都是作者亲身经历和经验之分享，运维人说实话，做实事，拒绝浮夸，在这本书体现得很好，希望能引起你的共鸣。

——赫宝生，金山西山居运维总监

云计算是一个令人兴奋的领域，作者在书中系统、详尽且量化地总结了一系列的方法与准则，这些方法与准则都在盛大游戏中进行过大规模的应用实践，不仅实用且大大开拓了读者视野。

更难能可贵的是，作者肖力是一位很注重实践和乐意分享的人。他发自内心、长期热爱这份事业，这一点在盛大游戏任职期间表现得淋漓尽致，盛大游戏 G 云的发展也得益于肖力及其团队对技术的深刻理解与不断探索。我相信，正是这份质朴的热爱和执著的追求成就了这本极具含金量的专业书籍。感谢他为盛大游戏所做的贡献，同时，诚挚分享此书给所有相关从业人员，希望更多的朋友能精读此书，从中得到更多启发和借鉴，一起成为云计算的弄潮儿。

——陈桂新，盛大游戏 G 云负责人、技术保障高级总监

很高兴看到越来越多的朋友加入云计算的大军，但对技术人员来说，从理论、实现到支撑业务会遇到不少的挑战。肖力、俊俊等把 G 云在技术方面走过的路、踩过的坑总结并分享出来，对有志于加入这个领域的同学有很大的实用参考价值。

感谢本书作者肖力、俊俊、爱伟和德禄对盛大游戏 G 云做出的贡献；G 云正是在你们的努力下一步步成长起来，并成为用户首选的、高性价比的定制云的。

——冯祯旺，盛大游戏 G 云 COO、技术保障副总监

在这个风起云涌的年代，虚拟化是各家云公司的核心和门槛。可以说没有虚拟化就谈不

1107114476@qq.com
dolphinebook.com

上 IaaS 的云服务。该书很好地诠释了时下最流行的 Hypervisor——KVM 的前生与今世，并且由浅入深，从实战的角度展现给读者怎么搭建、使用、运维大规模的 KVM 集群，非常值得读者深入研读。

——宋伟，金山云合伙人、技术 VP、政企事业部总经理

KVM 技术虽已出现多年，并在云计算浪潮中发挥重要作用，但相关技术深度实践的书籍并不多见。本书作者为国内 KVM 技术早期研究者及实践者之一，本书内容由浅入深、图文并茂，适合各种级别的 KVM 技术人员查阅学习。

——萧田国，触控科技运维总监、InfoQ 金牌专栏《高效运维最佳实践》作者

不管你是否承认云时代已经来临，VMware、Xen、KVM、Docker 等虚拟机技术层出不穷，KVM 作为内核级的虚拟化技术，一直受到很多工程师的青睐，不管是性能还是安全性都算是其中的佼佼者。

本书作者有多年 KVM 运维实践经验，把自己对 KVM 的理解及工作中遇到的问题，通过这本书深入浅出地告诉大家，特别是还介绍了 Ceph 和 OpenStack 等案例，希望大家看完这本书后可以对 KVM 有一个新的认识，对虚拟化技术有一个新的认识。

——南非蜘蛛，开源爱好者

认识肖力已经很长时间，从盛大游戏时期参与建设盛大 G 云的前身，到如今在西山居从事游戏虚拟化工作，肖力是我在业界见过为数不多的既有实操能力，又有宏观格局，同时具备海量运维经验的云计算从业者。作为国内最早一批 KVM 实践者，本书是肖力多年实战经验和思考的积累，站在应用者的角度剖析了利用 KVM 建设私有云平台的过程和经验，这本书不仅适合 KVM 爱好者，对于所有的云计算爱好者和从业人员都大有裨益，郑重推荐。

——吴磊，阿里云多媒体云总经理，兼通用业务总监

现在是思维横行的时代，我们每天在微博和朋友圈刷着各种信息，很容易变得浮躁。认识老肖快十年了吧，从盛大到西山居，一直在踏踏实实地研究虚拟化相关领域，可以说在老肖身上能看到互联网运维圈子里出来的人所固有的执着与沉静。

其实，现在这类技术书籍很多，但这本书的宝贵之处在于是一群互联网资深匠者，以真诚与严谨的方式，对待自己孩子般地去雕琢技术细节，里面珍贵的经验或许可以帮助我们少踩很多坑。而更重要的是，你在这里收获的不仅仅是技术，还有互联网资深老兵的经验和感悟。本书是适合从事互联网行业以及服务器领域的技术工程师们仔细研读的一部作品。

——李刚，浪潮互联网行业部技术总监

1107114476@qq.com
dolphinebook.com

随着近些年互联网的发展，虚拟化技术的日益成熟，虚拟化和云也逐步被大家接受和应用。

伴随需求的增长，公有云厂商也不断崛起和扩大。海外的亚马逊云、RackSpace、谷歌云、微软云，都在拓展中国市场；国内的阿里云、金山云、盛大云、腾讯云、UCloud 也日渐壮大，还有百花齐放的私有云，甚至近两年流行的容器技术，这些现象向我们暗示着未来的技术走向。

虚拟化作为云服务的基石，其重要性不言而喻。或许你很早就知道 VMware，然后可能你开始听说 Xen/Citrix、Hyper-V。没错，然后 KVM 横空出世，一发不可收拾！

相信现在的运维、开发或测试人员，或多或少会用到虚拟化。无论你使用哪种虚拟化方案，KVM 作为当下主流的虚拟化方案都值得你去了解和学习。

肖力同学专注于虚拟化工作多年，有着极其丰富的实战经验。善于总结和分享的他，提笔撰写本书必定会成为 KVM 运维技术的经典之作。

本书涵盖知识面非常广，能很好地拆分讲解并实现由浅入深，可用“从入门到精通”来形容。开篇介绍了虚拟化选型和 KVM 的发展史，让读者很容易融入进来；接下来，很贴切地从“第一台虚拟机”开始动手实战；然后，分类解读各种虚拟化技术及应用场景、怎样做资源控制；而后接地气地介绍了物理机转换到虚拟机的实践以及桌面虚拟化实践。实现了从技术的基本介绍深入到“你想用，怎么做”的自然延伸。

第二篇介绍分布式文件系统，是云平台的另一个基础组件，和 KVM 密不可分。文中详细介绍了各大主流分布式存储系统与 KVM 的结合使用，分布式存储完善 KVM 的健壮性，同时保证数据的安全性。随后，高大上地介绍了主流的 KVM 管理平台，以及虚拟化项目实施的实践经验分享。本书内容丰富精彩、层次清晰，无论初学者还是“老司机”，都能从中受益。

——胡凯，猎豹移动运维总监

“基石”这个词，可以形象地描述开源虚拟化技术对互联网的价值，也代表着互联网基础设施技术的未来。但“开源”对普通的技术人员来说，意味着需要在缺乏路标的海洋中航行。

这本书难得之处在于：是由一群奋战在互联网第一线的牛人总结的经验宝典，本书的内容，让我不断回想起多年来和各个互联网公司的小伙伴们在虚拟化技术实验、部署、侦错、管理过程中不下几百次的各种研讨，历历在目。他们成功运营着成千上万台的服务器，处理上亿客户的海量业务请求，他们所做的技术探索，在目前的互联网行业具有广泛的代表性和实用性。

而今天，有幸他们能够把自己的经验和体会全面地分享，全都是干货。我想，这就是互

1107114476@qq.com
dolphinebook.com

联网所崇尚的开放精神吧。这本书的作者和读者，以及这一代在互联网开源技术领域勤奋工作的人们，将会构成中国互联网基础架构技术的基石。

——陆晴，英特尔互联网及云计算业务部经理

知其因，晓其理；漫步在云端，然不知处处都是虚无缥缈的浮夸；落地、上云都离不开底层技术的支持，一线大量实战经验的总结是对垂直领域最好的诠释。这本书恰恰是这一领域的佼佼者，深入、全面并包含大量实践，不仅能帮助我们晓其理，一步步学习虚拟机技术，而且能帮助我们快速将云落地。

——刘宇，西山居运维架构师

基于 KVM 的虚拟化方案在业界已应用多年，而且越来越多的云计算服务商使用该方案对外提供服务，这对运维人员来说是一个大的挑战，因为国内还没有一本关于 KVM 运维实践的权威的书籍，大部分的问题都是靠运维人员自己摸索，不成体系。这本书的出版对从事 KVM 虚拟化方案的运维人员来说是个福音，从内容看，作者很用心，而且对运维和从事虚拟化平台开发的人员也有很好的参考作用，相信此书的出版能推动我国虚拟化乃至云计算方面的运维水平和国际接轨。

——叶理灯，UCloud 云主机开发部经理

得知力哥要写一本关于 KVM 的书，我的第一感觉是国内做 KVM 人有福了。当年我和力哥一起搞 KVM 的时候，就深感他功底扎实，思维缜密，而且动手能力快得惊人。后来，我转向 PaaS 领域，力哥又不间断地在 KVM 上深耕数载，如今已是该领域的专家。相信无论是虚拟化领域的新手还是资深从业人员，都能从他的书中得到很多借鉴。

——刘晓光，Cloud Foundry 工程师

从力哥的博客和“KVM 虚拟化实践”的公众号就可以看出这是一位注重实践、乐于分享、很有互联网精神的人。

实践出真知，本书全面性佳、实战性强，相信对 KVM 有兴趣以及正在做 KVM 相关虚拟化、云计算相关的兄弟来说必定会备感亲切，喜爱有加，强烈推荐！

——秦洁，安居客运维负责人

在虚拟化道路上，选型 KVM 是一个深思熟虑的决定，通过小规模部署验证，最终大规模部署到生产环境中。在虚拟化的道路上，我们学习到很多，想要尽可能和大家去分享。本书将会给大家带来 KVM 在生产环境中的、资深运维角度所理解的最佳实践。

云计算改变了传统运维的思维方式，短短几分钟可以实现快速部署，扩容生产业务。想

1107114476@qq.com
dolphinebook.com

要通过 KVM 虚拟化技术结合存储构建一个稳定、高效的生产集群，支撑业务弹性伸缩，这本书不可不读。

——杨树林，动视 SRE

说来有缘，肖兄的这本书还算是我牵的线。自知我在 KVM 方面并非专家，因此不敢多加点评。我和肖兄认识的时间并不长，但从肖兄的微信公众号分享的持续性可见他对知识积累的专注和专业，让我非常敬佩，也让我相信此书绝对能成为国内 KVM 领域的巅峰之作，对诸多同行们有所帮助。

——叶金荣，http://imysql.com 创始人，Oracle ACE（MySQL）

1107114476@qq.com
dolphinebook.com

前　言 Preface

目前，国内新一代的公有云底层采用的都是KVM，据笔者了解，国内原来一些采用Xen的公有云也逐步在向KVM迁移。许多企业的私有云项目招标中，已经明确要求使用KVM。在全球范围内，OpenStack就是云建设的标准，而当前OpenStack底层默认就是KVM。经过几年的快速发展，KVM技术已经非常成熟稳定，在任何Linux发行版中，KVM都是标配。虚拟化已经成为一项必备的技术技能，而学习Linux就必须学习KVM。

为什么写这本书

2009年，当虚拟化在互联网业内鲜有使用时，我们虚拟化小组在国内已经在生产环境率先开始KVM虚拟化技术的探索。通过实施虚拟化，我们几年内为公司节省了上亿元的运营成本，帮助公司多个处于生命周期末的游戏项目实现扭亏为盈。

当时，KVM还是一个小众产品，国内文档较少，我们只能通过钻研国外的英文资料甚至阅读源代码来学习。RedHat是KVM的重要支持者，因此与其公司RHEL系统同源的CentOS成为了研究KVM的重要资料。从当时刚发布的CentOS 5.5版本开始，我们一路追随CentOS系统的升级，每有一个版本发布，我们都要测试评估新的KVM虚拟化特性，然后逐步在生产环境部署，这样的工作方法一直持续到最新的CentOS 6.6及CentOS 7.1。这也是我们小组人员快速成长的一个阶段，我们对KVM技术的热爱一直保持了下来，我们在周围普及KVM技术的热情一直没有改变。

时至今日，KVM经过几年的快速发展，已经成为了最炙手可热的虚拟化引擎，国内新一代的公有云全部选择了KVM。而与KVM如火如荼的发展形成鲜明对比的是中文资料依然匮乏，尤其是介绍实践经验的资料。在我们周围和各种技术论坛上，经常碰到一些运维工程师，在实施一个KVM虚拟化项目时，不知道从哪里开始；也碰到对KVM技术感兴趣的运维人员在重复问一些相近的问题。于是，我们几个人萌生了一个想法，为什么不把我们的KVM项

1107114476@qq.com
dolphinebook.com

目运维实践经验一一记录下来，分享给大家。与此同时，我们也犹豫过，因为有许多问题我们也并未完全理解。但是，经过认真思考后的我们认为，即便在这种情况下，我们几年积累下来的实践经验对于想要了解和学习 KVM 虚拟化技术的工程师来说也是非常有意义的，毕竟实践更能检验真理。回想当初自己遇到问题的时候，我们也得到了一些论坛朋友、圈内专家的热心解答。互联网精神崇尚开放，这更加坚定了我们创作此书的决心。我们愿意将自己对 KVM 技术的理解和认识，以及在 KVM 虚拟化中解决问题的思路和方法分享出来，让大家少走一些我们走过的弯路，同时能够与所有运维工程师共同学习提升！

本书特色

本书是 KVM 一线项目运维实践经验总结，在 KVM CPU、内存、网络、磁盘方面拥有丰富的技术积累，在 KVM 虚拟化开源分布式与商业存储、KVM 桌面虚拟化方面有的大量实践，我们还会指出踩过的“技术坑”，这些都是弥足珍贵的经验。

本书对 Linux 系统和虚拟化有一定了解，但是对 KVM 不了解或者是了解不多，而准备在测试或者生产环境部署 KVM 虚拟化的读者，本书会介绍虚拟化项目如何评估、如何逐步实施，并介绍相关的技术与运维实践，希望对读者的生产环境虚拟化实施有所帮助。

对于正在做 KVM 虚拟化日常运维的读者，本书有相应的技术、实践经验、问题案例分享，便于这些读者碰到相似的问题时查询或者找到解决思路。

本书面向的读者

KVM 已经是 Linux 系统的标配，是 Linux 的必备技能，希望本书对广大 Linux 从业者能够有所帮助。

本书适用于以下读者：

- Linux 系统工程师。
- 从事 Linux 虚拟化开发的程序员。
- 虚拟化工程师。
- IDC 机房管理员。
- 有志于从事虚拟化的在校大学生。

如何阅读本书

本书分为五篇。

第一篇（第 1 ~ 8 章），介绍虚拟化选型及 KVM 虚拟机技术。实施虚拟化可带给我们哪

1107114476@qq.com
dolphinebook.com

些收益，虚拟化引擎选择，KVM 虚拟化相关的 CPU、内存、磁盘、网络技术及生产环境应用场景，物理机转虚拟机，KVM 桌面虚拟化实践。

第二篇（第 9 ~ 10 章），介绍开源分布式文件系统在 KVM 虚拟化的应用。介绍一些常见的开源分布式文件系统如何在 KVM 环境使用，重点详细介绍最近一两年比较热的 Ceph 文件系统，包括配置、使用和常见故障处理。

第三篇（第 11 ~ 13 章），介绍开源的虚拟化管理平台在生产环境的实践经验。重点介绍 OpenStack、OpenNebula 在生产环境的实践及故障处理，还有 oVirt、CloudStack、Convirt、WebVirtMgr 几个管理平台的架构和理念。

第四篇（第 14 ~ 21 章），介绍 KVM 虚拟化项目实施经验，涵盖业务性能评估、压力模型建立、虚拟机性能测试、宿主机基准测试、业务测试环境测试验证、单机虚拟化技术、集群虚拟化技术、虚拟化正式环境上线等一套完整的虚拟化项目实施过程。

第五篇（第 22 章），介绍 KVM 虚拟化过程中比较典型的一些案例。主要介绍一些在生产环境中的案例及解决思路、方法，方便读者碰到类似的问题时找到解决方案。

根据本书内容，我们给出如下阅读建议：

- 如果是刚准备开始实施虚拟化的读者，请从头开始阅读本书。
- 如果对分布式文件系统在虚拟化中的应用感兴趣的读者，可以阅读第二篇的内容。
- 如果对虚拟化管理平台使用感兴趣的读者，可以阅读第三篇的内容。
- 如果是对虚拟化技术有一定了解，但是对不知道如何实施虚拟化的读者，建议阅读第四篇的内容。
- 如果在虚拟化实施碰到问题，需要寻找解决方法的读者，可以阅读第五篇的内容。

资源和勘误

由于时间紧张，笔者水平有限，书中难免有错误和不足之处，笔者运营着一个微信订阅号，名字为“KVM 虚拟化实践”，订阅号为“kvm_virt”二维码为：

1107114476@qq.com
dolphinebook.com

欢迎读者扫描关注，“KVM 虚拟化实践”会定时发布笔者对 KVM 虚拟化新的认识和经验总结。也欢迎读者通过这个微信订阅号和笔者交互，书中的勘误和更新也会通过这个订阅号发布。

致谢

感谢盛大游戏高级总监陈桂新，盛大游戏副总监、G 云 COO 冯祯旺对我们的支持，我们四个人都曾经或者现在在盛大游戏虚拟化小组工作，桂总、冯总一路上支持我们，坚定地支持我们沿着 KVM 的方向前进，即使中间碰到多次挫折。回首看，我们坚持了下来，也很高兴我们这些年的成果终于开花结果，孵化出了 G 云（www.gicloud.com.cn），一款专门针对游戏行业物理机和云主机的混合云。

感谢金山西山居运维总监赫宝生对本书写作的鼓励和支持。感谢金山西山居运维架构师刘宇对本书的帮助和建议。

感谢曾经在盛大游戏虚拟化小组工作过的刘晓光、秦洁、杨树林，感谢现在还在盛大游戏虚拟化小组工作的夏学峰、闫强，这本书也是我们所有人的经验总结。

感谢盛大游戏网络技术负责人、G 云网络架构师苏永华，金山西山居资深网络工程师李正奇在网络方面的支持和帮助。

感谢金山西山居资深系统工程师、Zabbix 权威专家薛群在宿主机监控方面的帮助。

特别感谢金山西山居高级系统工程师赵斌，提供了部分虚拟化生产环境实践案例及测试结果。

感谢盛大游戏资深工程师杨帆、季青、武文对我们在服务器知识方面的支持和帮助。

感谢国内顶尖的数据库专家叶金荣在本书成书过程中的支持和协助。

感谢机械出版社的杨福川和高婧雅，福川兄出版了一系列互联网技术图书，在认识福川兄之前，我们已经购买了多本福川兄出版的书，和福川兄的合作是我们的梦想。相信随着时间的推移，会有更多的人认识到福川兄对中国互联网技术巨大的贡献。高婧雅编辑的专业、细致、认真、耐心令我们非常敬佩，和高婧雅合作的大半年内，我们受益匪浅，很感谢、很庆幸有机会和杨兄、高编辑合作。

致我们的家人

感谢我的舅舅、舅妈、表哥、表姐，你们对我的人生观影响很大；感谢我的父亲、母亲，是你们一直在默默支持我；感谢我妻子和女儿一直以来的理解和支持，你们是我的动力。

——肖力，2015 年 6 月

1107114476@qq.com
dolphinebook.com

感谢父母辛苦的培养，没有你们就没有今天的我。感谢老婆在我一无所有的时候默默陪在我的身边，还有可爱的儿子，你们是我一直前进的动力。

——汪爱伟，2015 年 6 月

感谢生我养我的父亲母亲，这些年，你们辛苦了，儿子长大了，你们却老了。感谢所有一路和我走过来的人，你们的支持和信任，给了我前进的勇气和力量。感谢相信我，愿意和我一路走下去的人，无论明天怎样，我都希望一路有你。

——杨俊俊，2015 年 6 月

参与这本书的写作对我来说非常荣幸，首先要感谢肖力等前辈传授经验并耐心支持，其次还要感谢家人对我业余时间不能陪伴的理解！

——赵德禄，2015 年 6 月

1107114476@qq.com
dolphinebook.com

Contents 目　录

本书赞誉
前　言

第一篇　KVM 技术详解与实践

第 1 章　企业虚拟化选型与 KVM 介绍 …… 2

1.1　KVM 的前世今生 …… 2
1.2　KVM 与常用企业级虚拟化产品的 PK …… 7
1.3　判断企业是否适合使用 KVM 的标准 …… 9
1.4　本章小结 …… 10

第 2 章　开始自己的第一台虚拟机 …… 11

2.1　服务器 BIOS 设置 …… 11
2.2　宿主机 CentOS 6.5、CentOS 7 系统安装与配置技巧 …… 13
2.3　第一台虚拟机安装 …… 15
2.4　本章小结 …… 21

第 3 章　CPU、内存虚拟化技术与应用场景 …… 22

3.1　NUMA 技术与应用 …… 22
3.1.1　KVM 虚拟机 NUMA 调优 …… 24
3.1.2　CPU 绑定操作方法 …… 27
3.2　CPU 热添加与应用 …… 32
3.3　CPU host-passthrough 技术与应用 … 36
3.4　CPU Nested 技术与配置方法 …… 38
3.5　KSM 技术与应用 …… 39
3.6　内存气球技术详解与应用 …… 40
3.7　内存限制技术与应用 …… 43
3.8　巨型页内存技术与应用 …… 44
3.9　本章小结 …… 47

第 4 章　网络虚拟化技术与应用场景 … 48

4.1　半虚拟化网卡技术详解 …… 48
4.1.1　半虚拟化网卡的配置 …… 49
4.1.2　全虚拟化网卡、半虚拟化网卡性能比较 …… 50
4.1.3　全虚拟化网卡、半虚拟化网卡的应用场景 …… 63

1107114476@qq.com
dolphinebook.com

4.2 MacVTap 和 vhost-net 技术原理与应用 …… 63
4.2.1 MacVTap 技术与应用 …… 64
4.2.2 vhost_net 技术 …… 66
4.3 网卡的中断与多队列 …… 66
4.4 网卡 PCI Passthrough 技术 …… 71
4.5 SR-IVO 虚拟化技术 …… 72
4.6 虚拟化软件交换机 Open vSwitch 的安装与配置 …… 75
4.6.1 Open vSwitch 安装 …… 75
4.6.2 Open vSwitch 典型配置 …… 77
4.7 多网卡绑定与建桥 …… 79
4.8 本章小结 …… 82

第 5 章 KVM 磁盘虚拟化技术与应用场景 …… 83

5.1 磁盘虚拟化方式性能比较与应用场景 …… 83
5.1.1 QEMU 磁盘虚拟化方式概述 …… 83
5.1.2 IDE 与 Virtio 性能比较测试 …… 84
5.2 Virtio 磁盘缓存方式详解、性能比较与应用场景 …… 85
5.2.1 磁盘缓存详解 …… 86
5.2.2 缓存方式的性能比较测试 …… 87
5.2.3 缓存方式的应用场景 …… 89
5.3 磁盘镜像格式 …… 89
5.3.1 QEMU 支持的磁盘镜像格式 …… 89
5.3.2 裸设备使用 …… 96
5.3.3 KVM 虚拟机镜像管理利器 guestfish …… 97
5.3.4 raw、qcow2、裸盘、lvm 性能比较 …… 106
5.3.5 磁盘镜像格式的应用场景 …… 107
5.4 文件系统块对齐 …… 108
5.4.1 什么是块对齐 …… 108
5.4.2 块对齐与块不对齐性能比较 …… 109
5.4.3 生产环境如何配置块对齐 …… 110
5.5 SSD 在 KVM 虚拟化中的使用实践 …… 111
5.5.1 SSD 原理与写放大 …… 111
5.5.2 SSD 在 KVM 中的使用 …… 112
5.5.3 Flachcache 配置与性能测试 …… 117
5.5.4 DM-cache 配置 …… 120
5.5.5 LVM cache 配置与性能测试 …… 122
5.5.6 生产环境中 SSD 使用要点 …… 123
5.6 本章小结 …… 126

第 6 章 KVM 虚拟机的资源限制 …… 127

6.1 哪些场景要做虚拟机的资源限制 …… 127
6.2 使用 CGroups …… 130
6.3 KVM 虚拟机 CPU 资源限制配置 …… 142
6.4 KVM 虚拟机网络资源限制 …… 144
6.5 KVM 虚拟机磁盘资源限制 …… 150
6.6 本章小结 …… 151

第 7 章 物理机转虚拟机实践 …… 152

7.1 哪些应用场景适合做物理机转虚拟机 …… 152
7.2 P2V 的技术实施方案 …… 152
7.3 Windows 系统物理机转虚拟机的操作方法 …… 153

1107114476@qq.com
dolphinebook.com

7.3.1 WinPE 的制作方法 …… 153
7.3.2 Windows 系统 P2V 操作方法 … 155
7.4 Linux 物理服务器转换虚拟机操作 …… 155
7.5 使用 virt-p2v 工具进行 P2V 转换 … 157
7.6 P2V 经验总结 …… 160
7.7 关于 V2V …… 160
7.8 本章小结 …… 160

第 8 章 KVM 桌面虚拟化实践 …… 161

8.1 桌面虚拟化简介及 Spice 协议 …… 161
8.2 桌面虚拟化实践经验 …… 162
8.3 本章小结 …… 167

第二篇 分布式文件系统

第 9 章 几种常见开源文件系统在 KVM 中的应用 …… 170

9.1 演示环境基础配置 …… 170
9.2 DRBD 在 KVM 中的应用 …… 172
9.2.1 DRBD 介绍 …… 172
9.2.2 DRBD 搭建方法及在 KVM 中的应用 …… 173
9.2.3 DRBD 在 KVM 虚拟化中的优化方案 …… 183
9.3 GlusterFS 在 KVM 虚拟化中的应用 …… 184
9.3.1 GlusterFS 介绍 …… 184
9.3.2 GlusterFS 集群搭建与使用 …… 189
9.3.3 GlusterFS 在 KVM 虚拟化中的应用 …… 195
9.3.4 GlusterFS 在 KVM 虚拟化中的优化方案 …… 198
9.4 Sheepdog 在 KVM 中的应用 …… 199
9.4.1 Sheepdog 介绍 …… 199
9.4.2 Sheepdog 搭建方法及在 KVM 中的应用 …… 201
9.5 MooseFS 在 KVM 中的应用 …… 207
9.5.1 MooseFS 介绍 …… 208
9.5.2 MFS 搭建方法及在 KVM 虚拟化中的应用 …… 208
9.6 本章小结 …… 213

第 10 章 Ceph 在 KVM 虚拟化中的应用与故障处理 …… 214

10.1 Ceph 简介 …… 214
10.2 Ceph 部署方法及在 KVM 中的应用 …… 217
10.2.1 使用 ceph-deploy 工具部署 Ceph …… 217
10.2.2 挂载 CephFS …… 221
10.2.3 通过 Librbd 方式使用 Ceph RBD …… 223
10.2.4 使用内核方式挂载并使用 Ceph RBD …… 226
10.3 CRUSH 算法介绍及调优方法 …… 228
10.3.1 CRUSH 算法介绍 …… 228
10.3.2 CRUSH 算法调优方法 …… 230
10.4 多角度评估 Ceph 在 KVM 虚拟化环境中的应用 …… 234
10.5 Ceph 生产环境搭建案例 …… 240

1107114476@qq.com
dolphinebook.com

10.5.1 使用 CephFS 集中备份虚拟机 …… 240
10.5.2 使用 Librbd 将虚拟机运行在 Ceph RBD …… 241
10.5.3 在 CephFS 上创建 qcow2 文件作为云硬盘 …… 242
10.6 Ceph 常见故障处理 …… 242
10.6.1 Ceph 集群监控状态检查 …… 243
10.6.2 Ceph 常见错误提示的含义与处理方法 …… 244
10.7 本章小结 …… 245

第三篇 KVM 虚拟化管理平台

第 11 章 利用 OpenStack 管理 KVM …… 248
11.1 OpenStack 搭建 …… 248
11.1.1 OpenStack 简介 …… 248
11.1.2 控制节点的安装 …… 249
11.1.3 计算节点的安装 …… 266
11.2 OpenStack 实用技巧 …… 269
11.2.1 RabbitMQ Cluster 搭建 …… 269
11.2.2 利用 Haproxy 和 Keepalived 实现控制节点高可用 …… 271
11.2.3 冷迁移虚拟机的方法 …… 277
11.3 本章小结 …… 277

第 12 章 利用 OpenNebula 管理 KVM …… 278
12.1 OpenNebula 架构 …… 278
12.1.1 OpenNebula 架构介绍 …… 278
12.1.2 OpenNebula 组件介绍 …… 279
12.2 在 KVM 中使用 OpenNebula …… 280
12.2.1 在 CentOS 6.5 上快速搭建 OpenNebula …… 280
12.2.2 使用 OpenNebula 创建第一台 KVM 虚拟机 …… 285
12.2.3 使虚拟机硬盘创建在宿主机本地硬盘上 …… 288
12.3 OpenNebula 常见故障处理 …… 291
12.4 本章小结 …… 293

第 13 章 其他管理平台介绍 …… 294
13.1 ConVirt 管理平台介绍 …… 294
13.1.1 ConVirt 的理念 …… 294
13.1.2 ConVirt 的架构 …… 295
13.2 CloudStack 管理平台介绍 …… 295
13.3 oVirt 管理平台介绍 …… 297
13.4 WebVirtMgr 管理平台介绍 …… 299
13.5 如何选择管理平台 …… 303
13.6 本章小结 …… 304

第四篇 KVM 虚拟化项目实施

第 14 章 业务性能评估与压力模型建立 …… 306
14.1 Linux 系统性能数据采集原则与方法 …… 306
14.2 Linux 系统压力模型建立 …… 311
14.3 Windows 系统性能数据采集原则与方法 …… 314
14.4 Windows 系统压力模型建立 …… 317

1107114476@qq.com
dolphinebook.com

14.5 本章小结 …… 319

第 15 章 宿主机选型与基础性能测试 …… 320

15.1 宿主机 CPU 选型原则 …… 320
15.2 宿主机内存选型注意事项 …… 322
15.3 宿主机硬盘选型 …… 324
15.4 RAID 卡选型 …… 326
15.5 服务器网卡选型 …… 327
15.6 宿主机性能基准测试实践 …… 328
15.7 本章小结 …… 345

第 16 章 虚拟机镜像制作、配置与测试 …… 346

16.1 Windows 虚拟机制作方法 …… 346
16.1.1 Windows 镜像制作 …… 346
16.1.2 Windows 虚拟机 sysprep 初始化封装 …… 361
16.2 Linux 镜像制作方法 …… 362
16.2.1 RHEL/CentOS 镜像制作方法 …… 362
16.2.2 Ubuntu、Debian 虚拟机配置注意点 …… 364
16.3 虚拟机自动配置 IP 的实现办法 …… 365
16.3.1 通过 DHCP 给虚拟机配置 IP …… 365
16.3.2 通过 Libvirt 实现虚拟机自动配置 IP …… 365
16.3.3 笔者生产环境虚拟机 IP 自动配置的方法 …… 365
16.4 虚拟机镜像测试 …… 375
16.5 本章小结 …… 375

第 17 章 单机虚拟化技术与生产环境实践 …… 376

17.1 单机虚拟化技术与应用场景 …… 376
17.1.1 单机虚拟化的优势和劣势 …… 377
17.1.2 单机虚拟化技术 …… 378
17.1.3 单机虚拟化应用场景 …… 380
17.2 单机虚拟化生产环境注意点 …… 381
17.2.1 单机虚拟化灾备与应急方案 …… 381
17.2.2 单机虚拟化生产环境部署实践 …… 382
17.3 本章小结 …… 385

第 18 章 KVM 虚拟化集群技术与应用场景 …… 386

18.1 虚拟化集群设计思路与架构 …… 386
18.2 虚拟化集群的实施流程与技术方案 …… 389
18.3 虚拟化集群搭建 …… 392
18.3.1 计算节点配置与管理 …… 392
18.3.2 网络设计与配置 …… 403
18.3.3 虚拟机迁移管理 …… 404
18.3.4 虚拟化集群的备份与恢复 …… 407
18.4 Dell EqualLogic 存储的管理实践经验 …… 414
18.5 本章小结 …… 419

第 19 章 业务迁移到虚拟化环境流程 …… 420

19.1 性能评估与测试环境测试 …… 420

1107114476@qq.com
dolphinebook.com

19.2 上线前的检查工作 …… 428
19.3 小规模部署及逐步切换到虚拟化环境 …… 429
19.4 本章小结 …… 430

第 20 章 宿主机自动化运维管理 …… 431

20.1 Puppet 简介与安装部署 …… 431
20.2 利用 Puppet 管理宿主机的运维架构设计 …… 436
20.3 利用 Puppet 配置宿主机初始化操作 …… 441
20.4 利用 Puppet 管理宿主机配置与更新 …… 442
20.5 利用 Puppet 推送宿主机的非一致性配置 …… 445
20.6 本章小结 …… 449

第 21 章 虚拟化监控、报警与应急响应方案 …… 450

21.1 虚拟化监控内容 …… 450
21.1.1 宿主机 CPU 监控要点 …… 450
21.1.2 宿主机内存监控注意点 …… 455
21.1.3 宿主机网络监控注意点 …… 458
21.1.4 宿主机磁盘监控注意点 …… 460
21.1.5 宿主机层面监控虚拟机的方法 …… 461
21.2 监控软件 Cacti、Zabbix …… 463
21.2.1 用 Cacti 监控宿主机 …… 464
21.2.2 用 Zabbix 监控宿主机 …… 471
21.3 应急方案 …… 487
21.3.1 应急方案制定注意要点 …… 487
21.3.2 单机虚拟化应急方案 …… 491
21.3.3 集群虚拟化应急方案 …… 495
21.4 本章小结 …… 497

第五篇 KVM 虚拟化典型案例

第 22 章 生产环境问题案例与分析 …… 500

22.1 游戏在虚拟机上非常慢的案例 …… 500
22.2 虚拟机流量过高引起网络风暴的案例一 …… 502
22.3 虚拟机流量过高引起网络风暴的案例二 …… 517
22.4 其他生产环境问题案例 …… 520
22.4.1 共享存储集群虚拟机迁移故障案例 …… 520
22.4.2 宿主机异常关机后虚拟机无法启动案例 …… 522
22.4.3 宿主机使用 CentOS 5.6 系统问题案例 …… 523
22.4.4 Windows 虚拟机网络闪断、不通问题案例 …… 523
22.4.5 Windows 7 虚拟机只能使用 2 个 CPU 案例 …… 524
22.4.6 生产环境非常奇怪的一个案例 …… 524
22.4.7 U 盘在虚拟机上挂载问题案例 …… 526

1107114476@qq.com
dolphinebook.com

22.4.8 HP ILO4 固件 Bug 引起宿主机重启案例 …… 528
22.4.9 一次机房断电引起的问题案例 …… 529
22.4.10 CPU 绑定问题案例 …… 531
22.4.11 生产环境存储故障导致 50 个虚拟机丢失案例 …… 532
22.4.12 生产环境 Ubuntu 虚拟机故障问题案例 …… 533
22.5 本章小结 …… 538
附录 笔者推荐的 KVM 学习资料 …… 539

1107114476@qq.com
dolphinebook.com

1107114476@qq.com
dolphinebook.com

第一篇 *Part 1*

KVM 技术详解与实践

- 第 1 章　企业虚拟化选型与 KVM 介绍
- 第 2 章　开始自己的第一台虚拟机
- 第 3 章　CPU、内存虚拟化技术与应用场景
- 第 4 章　网络虚拟化技术与应用场景
- 第 5 章　KVM 磁盘虚拟化技术与应用场景
- 第 6 章　KVM 虚拟机的资源限制
- 第 7 章　物理机转虚拟机实践
- 第 8 章　KVM 桌面虚拟化实践

1107114476@qq.com
dolphinebook.com

Chapter 1 第 1 章

企业虚拟化选型与 KVM 介绍

虚拟化技术从 2008 年开始越来越热，经过一个大爆发的阶段，目前已经是企业 IT 环境的必备技术，在许多企业里面，虚拟机的数量已经远远大于物理机。同许多技术一样，虚拟化也分为开源和闭源技术。KVM 就是一种开源的虚拟化技术，本章将为读者介绍 KVM 技术的演进过程、KVM 的发展历史及 KVM 的应用场景。

1.1 KVM 的前世今生

如图 1-1 所示，根据 OpenStack 平台上 2013 年 10 月虚拟化引擎选择的调查统计数据，KVM 已经在 OpenStack 平台占到 71% 的份额。

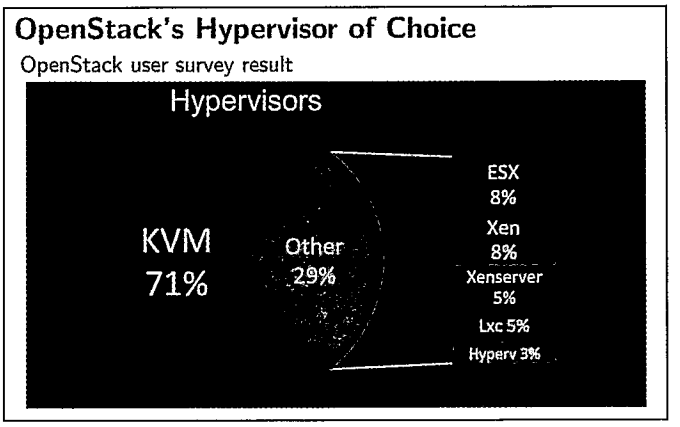


图 1-1 OpenStack 平台虚拟化引擎，用户调查结果

1107114476@qq.com
dolphinebook.com

KVM 是一种年轻的虚拟化技术，在出生的时候就吸取了其他虚拟化技术的优点，所以 KVM 的架构简单，没有历史兼容性的包袱，性能表现优异。

本节介绍一下 KVM 的演进过程。

1. 虚拟化技术的演变过程

虚拟化技术的演变过程可以分为软件模拟、虚拟化层翻译、容器虚拟化三个大的阶段。

其中，虚拟化层翻译又可以分为：

- 软件捕获翻译，即软件全虚拟化。
- 改造虚拟机系统内核加虚拟化层翻译，即半虚拟化。
- 硬件支持的虚拟化层翻译，即硬件支持的全虚拟化。

下面分别介绍一下这几种技术方式。

（1）软件模拟的技术方式

软件模拟是通过软件完全模拟 CPU、芯片组、磁盘、网卡等计算机硬件，如图 1-2 所示。

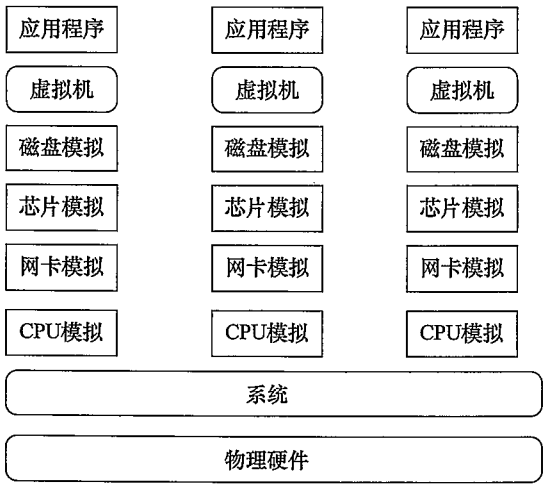


图 1-2　QEMU 的虚拟机架构

因为是软件模拟，所以理论上可以模拟任何硬件，甚至是不存在的硬件。但是因为这种方式全部是软件模拟硬件，所以非常低效，一般只用于研究测试的场景。采用这种技术的典型产品有 Bochs、QEMU 等。

（2）虚拟化层翻译

先介绍一下 X86 平台的指令集权限划分。如图 1-3 所示，X86 平台指令集划分为 4 个特权模式：Ring 0、Ring 1、Ring 2、Ring 3。操作系统一般使用 Ring 0 级别，应用程序使用 Ring 3 级别，驱动程序使用 Ring 1 和 Ring 2 级别。X86 平台在虚拟化方面的一个难点就是**如何将虚拟机越级的指令使用进行隔离**。

VMware 公司找到了最早的解决方法，图 1-4 所示为对虚拟机指令的使用进行捕获和翻译的示意图。

1107114476@qq.com
dolphinebook.com

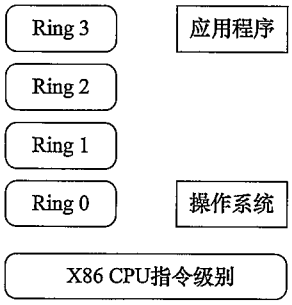


图 1-3 X86 CPU 指令级别

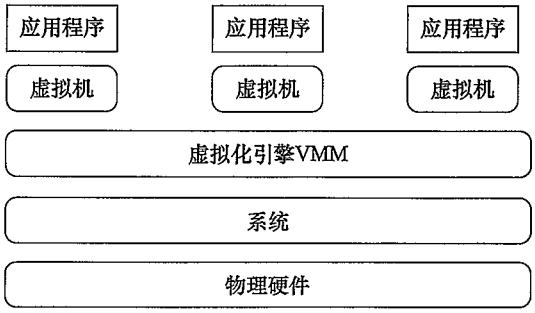


图 1-4 软件全虚拟化方案架构

通过虚拟化引擎，捕获虚拟机的指令，并进行处理，这也是为什么在虚拟机上虽然使用的是物理机一样的指令，但是虚拟机不能对硬件进行操作的原因，比如重启虚拟机不会引起宿主机的重启。这种解决方案也叫软件全虚拟化方案。

（3）改造虚拟机操作系统的方式

通过虚拟化引擎进行指令捕获和翻译的方式虽然可行，但是在虚拟化层要做大量的工作，Xen 项目提供了另外一种思路，就是对虚拟机的操作系统内核进行改造，使虚拟机自己对特殊的指令进行更改，然后和虚拟化层一起配合工作，这也是 Xen 早期一直要使用一个特殊内核的原因，并且不支持 Windows 系统虚拟化。改造的虚拟机虽然使用上有限制，配置比较麻烦，但是这种方式效率非常高，这种方式也被称为半虚拟化方案。

（4）对 CPU 指令进行改造

2005 年，Intel 推出了硬件的方案，对 CPU 指令进行改造，即 VT-x，如图 1-5 所示。VT-x 增加了两种操作模式：VMX root operation 和 VMX non-root operation。VMM 运行在 VMX root operation 模式，虚拟机运行在 VMX non-root operation 模式。这两种操作模式都支持 Ring 0 ~ Ring 3 这 4 个特权级。

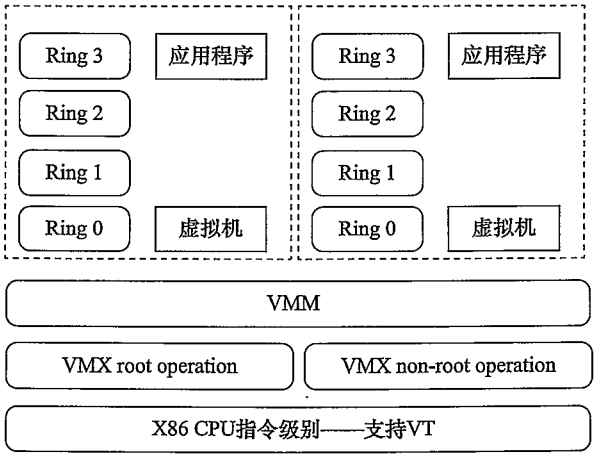


图 1-5 硬件虚拟化 CPU 指令说明

1107114476@qq.com
dolphinebook.com

这种方案因为是基于硬件的，所以效率非常高，这种方案也称为硬件支持的全虚拟化方案，如图 1-6 所示。现在的一个发展趋势是不仅 CPU 指令有硬件解决方案，I/O 通信也有硬件解决方案，称为 VT-d；网络通信也有硬件解决方案，称为 VT-c。

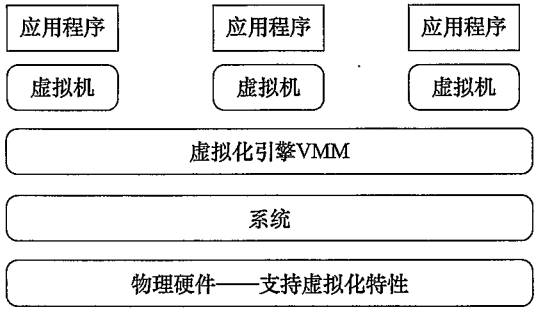


图 1-6　硬件全虚拟化方案架构

当前的虚拟化引擎，都是使用硬件支持的虚拟化解决方案。并且最新的操作系统一般都支持一些半虚拟化的特性，所以宿主机和虚拟机使用比较新的版本，性能也会好一些。

（5）容器虚拟化

容器虚拟化的原理是基于 CGroups、Namespace 等技术将进程隔离，每个进程就像一台单独的虚拟机一样，有自己被隔离出来的资源，也有自己的根目录、独立的进程编号、被隔离的内存空间。基于容器的虚拟化可以实现在单一内核上运行多个实例，因此是一个更高效率的虚拟化方式。目前最热的容器虚拟化技术就是 Docker。Docker 的优势是可以将一个开发环境进行打包，很方便地在另外一个系统上运行起来，并且有版本的概念，可以在前一个版本的基础上累加。但是 Docker 在生产环境的使用还需要一个过程，主要是磁盘、网络性能上还受到很多限制。

2. KVM 的历史

KVM（Kernel-based Virtual Machine）最初是由以色列的公司 Qumranet 开发的。KVM 在 2007 年 2 月被正式合并到 Linux 2.6.20 核心中，成为内核源代码的一部分。2008 年 9 月 4 日，RedHat 公司收购了 Qumranet，开始在 RHEL 中用 KVM 替换 Xen，第一个包含 KVM 的版本是 RHEL 5.4。从 RHEL 6 开始，KVM 成为默认的虚拟化引擎。KVM 必须在具备 Intel VT 或 AMD-V 功能的 X86 平台上运行。它也被移植到 S/390、PowerPC 与 IA-64 平台上。在 Linux 内核 3.9 版中，加入了对 ARM 架构的支持。

KVM 包含一个为处理器提供底层虚拟化、可加载的核心模块 kvm.ko（kvm-intel.ko 或 kvm-amd.ko），使用 QEMU（QEMU-KVM）作为虚拟机上层控制工具。KVM 不需要改变

1107114476@qq.com
dolphinebook.com

Linux 或 Windows 系统就能运行。

3. KVM 的架构

KVM 的架构非常简单，如图 1-7 所示，KVM 就是内核的一个模块，用户空间通过 QEMU 模拟硬件提供给虚拟机使用，一台虚拟机就是一个普通的 Linux 进程，通过对这个进程的管理，就可以完成对虚拟机的管理。实际上德国有家公司开发了一个管理平台 Proxmox VE，就是通过对 KVM 进程的管理来实现对虚拟机的管理的。

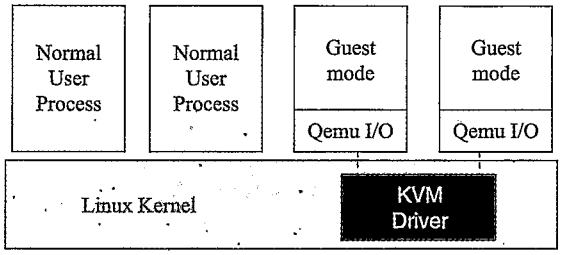


图 1-7 KVM 的架构

因为对进程的管理非常麻烦，RedHat 发布了一个开源项目 Libvirt。Libvirt 有 API，也有一套命令行工具，可以完成对虚拟机的管理，大多数的管理平台都是通过 Libvirt 来完成对 KVM 虚拟机的管理的，比如 OpenStack、CloudStack、OpenNebula 等。

4. QEMU 与 KVM

QEMU 是一个开源项目，实际就是一台硬件模拟器，可以模拟许多硬件，包括 X86 架构处理器、AMD64 架构处理器、MIPS R4000、ARM v6/v7（Cortex-A8，A9，A15）、SPARC sun3 与 PowerPC 架构，还支持其他架构，可以从 QEMU 主页获取完整的列表。

QEMU 可以在其他平台上运行 Linux 的程序，可以存储及还原虚拟机运行状态，还可以虚拟多种设备，包括网卡、多 CPU、IDE 设备、软驱、显卡、声卡、多种并口设备、多种串口设备、多种 USB 设备、PC 喇叭、PS/2 键盘鼠标（默认）和 USB 键盘鼠标、蓝牙设备。

QEMU 还内建 DHCP、DNS、SMB、TFTP 服务器。

比较有意思的是，有人还将 QEMU 编译成 Windows 版本，在 Windows 平台上也可以运行 QEMU。

QEMU 的好处是因为是纯软件模拟，所以可以在支持的平台模拟支持的设备，比如还有人利用 QEMU 在安卓上安装一个 Windows XP 虚拟机出来。

QEMU 的缺点是因为是纯软件模拟，所有非常慢。QEMU 1.0 的时候有一个 QEMU 和 KVM 结合的分支。KVM 只是一个内核的模块，没有用户空间的管理工具，KVM 的虚拟机可以借助 QEMU 的管理工具来管理。QEMU 也可以借助 KVM 来加速，提升虚拟机的性能。QEMU-KVM 的分支版本发布了 3 个正式的版本 1.1、1.2、1.3，随后和 QEMU 的主版本合并，也就是说现在的 QEMU 版本默认支持 KVM，QEMU 和 KVM 已经紧密地结合起来了。

1107114476@qq.com
dolphinebook.com

KVM 的最后一个自己的版本是 KVM 83，随后和内核版本一起发布，和内核版本号保持一致，所以要使用 KVM 的最新版本，就要使用最新的内核。

5. Libvirt 与 KVM

Libvirt 是一套开源的虚拟化的管理工具，主要由 3 部分组成：

- 一套 API 的 lib 库，支持主流的编程语言，包括 C、Python、Ruby 等。
- Libvirtd 服务。
- 命令行工具 virsh。

Libvirt 的设计目标是通过相同的方式管理不同的虚拟化引擎，比如 KVM、Xen、HyperV、VMware ESX 等。但是目前实际上多数场景使用 Libvirt 的是 KVM，而 Xen、HyperV、VMware ESX 都有各自的管理工具。

Libvirt 可以实现对虚拟机的管理，比如虚拟机的创建、启动、关闭、暂停、恢复、迁移、销毁，以及虚拟机网卡、硬盘、CPU、内存等多种设备的热添加。

Libvirt 还支持远程的宿主机管理，只要在宿主机上启动 Libvirtd 服务并做好配置，就可以通过 Libvirt 进行虚拟机的配置。通道可以是以下方式：

- SSH。
- TCP。
- 基于 TCP 的 TLS。

Libvirt 将虚拟机的管理分为以下几个方面：

第一，存储池资源管理，支持本地文件系统目录、裸设备、lvm、nfs、iscsi 等方式。在虚拟机磁盘格式上支持 qcow2、vmdk、raw 等格式。

第二，网络资源管理，支持 Linux 桥、VLAN、多网卡绑定管理，比较新的版本还支持 Open vSwitch。Libvirt 还支持 nat 和路由方式的网络，Libvirt 可以通过防火墙让虚拟机通过宿主机建立网络通道，和外部的网络进行通信。

1.2　KVM 与常用企业级虚拟化产品的 PK

1. 常用企业级虚拟化产品的比较

目前常见的企业级的虚拟化产品有 4 款：分别是 VMware、HyperV、Xen、KVM。

（1）VMware

VMware 是最早的 X86 平台上的虚拟化引擎，1999 年就发布了第一款产品，经过十几年的发展和市场检验，产品成熟、稳定，兼容性也不错。VMware 的产品线非常全面，不仅有虚拟化的解决方案，在 IaaS、SaaS、PaaS 层都有自己的产品。并且 VMware 在网络、存储方面都有相关的解决方案，VMware 和网络存储厂商在协议层面也有一些私有协议，许多主流的厂商都支持 VMware 一些专用的协议，和 VMware 一起形成了一个生态链。

1107114476@qq.com
dolphinebook.com

VMware目前被EMC控股，虚拟化产品线主要有针对个人使用的VMware Workstation，针对苹果用户的VMware Fusion，针对企业级用户的VMware ESXi服务器。管理工具主要是VMware vSphere套件。

VMware的产品基本上都是非开源产品，并且大部分都是收费产品，一般在传统关键行业使用比较多一些，在中小型企业、互联网行业使用得比较少一点。

（2）HyperV

HyperV是微软的虚拟化产品，最近几年发展非常迅速，在Windows Server 2012 R2中的HyperV支持许多非常新的虚拟化特性。HyperV必须使用64版的Windows产品，HyperV也支持Linux系统的虚拟机。

HyperV也是一款非开源的收费产品，HyperV的集群管理工具SCVMM配置非常复杂，需要配置Windows域、Windows Server集群，然后才能管理多台宿主机。因为HyperV的成本相对比较低，所以最近几年市场占有率也在提升，主要是一些使用Windows系统的企业使用比较多。

（3）Xen

Xen是最早的开源虚拟化引擎，由剑桥大学开发，半虚拟化的概念也是Xen最早提出的。Xen后来被思杰收购，推出了一套叫作XenServer的管理工具，XenServer于2013年年底宣布免费。Xen因为推出的时间比较长，兼容性、稳定性都不错，目前使用Xen的主要是一些在Xen上面技术积累较多的企业。

（4）KVM

KVM比较年轻，所以出生的时候就吸取了其他虚拟化技术的优点，一开始就支持硬件虚拟化技术，没有历史兼容包袱。所以KVM推出来的时候，性能就非常优异。目前，KVM是OpenStack平台上首选的虚拟化引擎。国内新一代的公有云全部采用KVM作为底层的虚拟化引擎。KVM已经成为开源解决方案的主流选择。

2. KVM优势

KVM的优势主要体现在以下几点。

（1）开源

KVM是一个开源项目，这就决定了KVM一直是开放的姿态；许多虚拟化的新技术都是首先在KVM上应用，再到其他虚拟化引擎上推广。

虚拟化一般网络和存储都是难点。网络方面，SRIOV技术就是最先在KVM上先有应用，然后再推广到其他虚拟化引擎上。再比如SDN、Open vSwtich这些比较新的技术，都是先在KVM上得到应用。

磁盘方面，基于SSD的分层技术，都是最早在KVM上得到应用。

KVM背靠Linux这棵大树，和Linux系统紧密结合，在Linux上的新技术都可以马上应用到KVM上。围绕KVM的是一个开源的生态链，从底层的Linux系统，到中间层的

1107114476@qq.com
dolphinebook.com

Libvirt 管理工具，到云管理平台 OpenStack，莫不是如此。

（2）性能

KVM 吸引许多人使用的一个动因就是性能，在同样的硬件条件下，能提供更好的虚拟机性能，主要是因为 KVM 架构简单，代码只有 2 万行，一开始就支持硬件虚拟化，这些技术特点保证了 KVM 的性能。

（3）免费

KVM 因为是开源项目，绝大部分 KVM 的解决方案都是免费方案，随着 KVM 的发展，KVM 虚拟机越来越稳定，兼容性也越来越好，因而也就得到越来越多的应用。

（4）广泛免费的技术支持

免费并不意味着 KVM 没有技术支持。在 KVM 的开源社区，数量巨大的 KVM 技术支持者都可以提供 KVM 技术支持。另外，如果需要商业级支持，也可以购买红帽公司的服务。

1.3　判断企业是否适合使用 KVM 的标准

1. 业务类型

目前在互联网行业，KVM 虚拟化技术是使用最广泛的，因为互联网行业有对新技术追求的冲动。另外中小企业也推荐使用 KVM 技术，因为使用 KVM 不需要支付额外的费用。

还有哪些企业、个人适合使用 KVM 呢？可以这样总结：KVM 天生就是为 Linux 而生的，凡是可以使用 Linux 的地方，就可以使用 KVM，KVM 天生和 Linux 在一起。

2. 企业对成本的关注度

随着虚拟化这几年的快速发展，在企业的 IT 环境中，虚拟化已经是一个标配的技术。虚拟化带来的好处是巨大的，虚拟化真正实现了资源池化，通过虚拟化可以将服务器资源进行切割，做到资源随取随用，有效节省成本，提高资源利用率。

KVM 虚拟化技术经过几年的发展，已经非常成熟，使用 KVM 技术更不需要在虚拟化方面支付额外的费用，可以进一步节省企业的 IT 成本。

3. 企业对快速部署的关注度

虚拟化还有一个重要的功能就是快速部署。在宿主机层面看，虚拟机就是一个镜像文件，要得到另外一台虚拟机，只需要将镜像文件复制一份就可以了，通常只有几分钟。而按照传统方式部署一台物理机，最起码都要一个小时。通过磁盘差量的技术，甚至可以做到秒级生成虚拟机。

通过虚拟化技术，还可以实现在虚拟化层做高可用和在线迁移。虚拟化层的高可用是系统层面的高可用，比基于应用层的高可用配置要简单很多。虚拟机的在线迁移，更是虚拟化技术的独有手段。通过在线迁移，物理机的维护、系统的维护、网络的维护，都可以做到不中断服务，进一步提高了业务应用的可用性。

1107114476@qq.com
dolphinebook.com

因为所有的虚拟机都是相同的虚拟硬件，实现了硬件层级的标准化，降低了自动化的难度，很容易搭建私有或者公有的云平台，所以通过虚拟机技术，很容易实现以下功能：

- ❑ 资源使用自动化和自助服务。
- ❑ 资源池可随时扩展。
- ❑ 资源使用标准化。
- ❑ 资源使用保持了很好的兼容性。

1.4 本章小结

本章介绍了虚拟化及 KVM 技术的演进过程，介绍了 KVM 技术的优势，可以简单地概括：凡是使用 Linux 系统的场景，如果上虚拟机项目，就天然适合使用 KVM 虚拟化技术。

下一章将介绍如何开始自己的第一台虚拟机，在安装第一台虚拟机的时候有哪些地方应该注意。

1107114476@qq.com
dolphinebook.com

第 2 章 Chapter 2

开始自己的第一台虚拟机

通过第 1 章对 KVM 的发展历史、架构及应用场景的介绍，读者对 KVM 技术有了初步的了解和认识。本章将介绍如何创建一台 KVM 虚拟机、操作步骤及要点。对于熟悉或者使用过 KVM 虚拟化的读者，本章内容可以快速阅读或者跳过。

2.1 服务器 BIOS 设置

KVM 的使用必须有硬件虚拟化支持，所以需要打开 CPU 的硬件虚拟化特性。对于大多数服务器，如 Dell、HP、IBM、浪潮、联想、华为等，在开机启动的第一个画面都会有一些提示的按键，图 2-1 所示是 Dell R610 开机启动的第一个画面。

图 2-1 Dell R610 开机启动的第一个画面

1107114476@qq.com
dolphinebook.com

该界面提供了如下几个选项：

- 按 F2 键进入 System Setup，进去之后可以进行一些 BIOS 相关的配置。
- 按 F10 键进入 System Services，进去之后可以配置一些系统服务，主要用于服务器的配置和检测。
- 按 F11 键进入 BIOS Boot Manager，进入之后可以选择启动介质，比如从硬盘启动、虚拟 DVD 设备、USB 设备，等等。
- 按 F12 键进入 PXE，进入之后启动网络引导。

提示 不同的厂商服务器，功能键的定义会有一些差别，可以根据提示来操作。

在安装 KVM 虚拟化之前，需要先确认 CPU 虚拟化支持是否开启。在如图 2-1 所示的服务器开机界面中按 F2 键，然后进入 BIOS 配置，选择 Processor Setting，按 Enter 键进入子菜单，其中一项 Virtualization Technology 必须配置为 Enabled（选择菜单项，按方向键右键可更改配置），如图 2-2 所示。

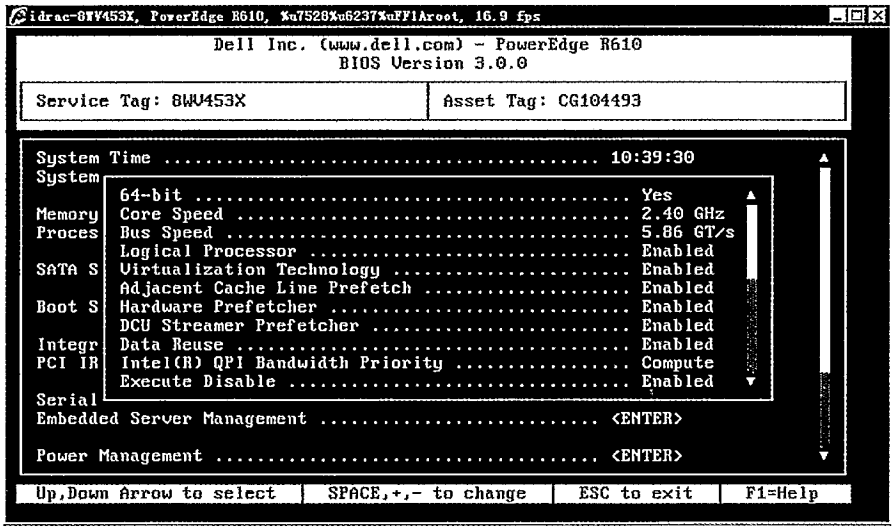


图 2-2 开启 CPU 虚拟化支持

提示 不同厂商的服务器，CPU Virtualization Technology 开关的位置会有些差别，一般都在处理器配置菜单之下。近年来服务器上的 CPU 一般都支持 CPU 虚拟化，在系统中可以执行 egrep '(vmx|svm)' /prov/CPUinfo 命令查看，如果有输出内容，说明 CPU 是支持虚拟化的。

1107114476@qq.com
dolphinebook.com

2.2　宿主机 CentOS 6.5、CentOS 7 系统安装与配置技巧

通过 2.1 节的配置，已经打开了服务器 CPU 对虚拟化的支持，下面开始安装宿主机的操作系统。CentOS 6.x 是目前使用比较多的宿主机操作系统版本，当前最新版本是 CentOS 6.6。CentOS 7 从系统跨度来说，是 CentOS 的一个大版本升级，其中主要包括内核版本的跨越。CentOS 6.x 系列使用的是 2.6.32.x 的内核，CentOS 7.x 则使用 3.10.x 的内核。

1. CentOS 6.5 宿主机系统安装及配置

在生产环境中，安装 CentOS 6.5 系统的宿主机，笔者采用 pxe 方式来完成批量的宿主机部署。pxe 配置过程的资料很多，本书就不做详细介绍了，下面分享一个宿主机安装 kickstart 文件的内容，是笔者实际在生产环境中使用的。

```
# Kickstart file automatically generated by anaconda
# 系统安装自动生成的 kickstart 文件
install
url --url=http://10.10.10.1/system/CentOS6564
# 指定安装镜像的目录，可以将不同的系统放在不同的目录，这样可以完成多个系统安装
lang en_US.UTF-8
# 宿主机建议使用英文
keyboard us
network --onboot yes --device eth0 --mtu=1500 --bootproto dhcp
network --onboot yes --device eth1 --noipv4 --noipv6
# 设置网络，笔者装机的时候，一般第一个网卡使用 dhcp 分配 IP，第二个网卡关闭。读者可以根据自己的
  实际情况进行修改
rootpw  cEmXc2pkKets    # 配置系统密码
text
reboot
# 文本方式安装，安装后重启
firewall --disabled
authconfig --useshadow  --passalgo=sha512
# 用户密码加密 seLinux --disabled  # 关闭 SELinux
#######################################################
# Installation logging level
logging --level=info
# Do not configure the X Window System
skipx
#######################################################
timezone Asia/Shanghai  # 配置时区
bootloader --location=mbr --driveorder=sda --append="crashkernel=auto rhgb quiet"
clearpart --all --initlabel  # 清除硬盘数据，并创建相应分区
part /boot --fstype ext3 --size=256
part pv.3 --size=40960
part pv.4 --size=100 --grow
volgroup datavg --pesize=32768 pv.4
volgroup KVMvg --pesize=32768 pv.3
logvol /datapool --fstype ext4 --name=datapool --vgname=datavg --size=10240
--grow  # 此处创建了一个目录，用于存放虚拟机，笔者习惯的目录是 /datapool
```

1107114476@qq.com
dolphinebook.com

```
logvol swap --fstype swap --name=swap --vgname=KVMvg --size=8192
logvol / --fstype ext4 --name=root --vgname=KVMvg --size=30720
%packages
# 安装虚拟化需要的包，主要有：
@virtualization
@Base
@Core
@additional-devel
@base
@large-systems
@storage-client-iscsi
@system-management-snmp
@virtualization
@virtualization-client
@virtualization-platform
@virtualization-tools
########################################################
%end
```

使用上面的 kickstart 配置文件安装完宿主机系统，可以通过 rpm -q 命令检查一下具体安装了哪些包。

```
[root@localhost ~]# rpm -qa|grep -E 'qemu|libvirt|virt'
libvirt-client-0.10.2-29.el6.x86_64
#Libvirt 的客户端，最重要的功能之一就是就在宿主机关机时可以通知虚拟机也关机，
# 使虚拟机系统正常关机，而不是被强制关机，造成数据丢失
gpxe-roms-qemu-0.9.7-6.10.el6.noarch # 虚拟机 iPXE 的启动固件，支持虚拟机从网络启动
libvirt-python-0.10.2-29.el6.x86_64    #libvirt 为 Python 提供的 API
python-virtinst-0.600.0-18.el6.noarch  # 一套 Python 的虚拟机安装工具
qemu-KVM-0.12.1.2-2.415.el6.x86_64 #KVM 在用户空间运行的程序
Virt-manager-0.9.0-19.el6.x86_64  # 基于 Libvirt 的图像化虚拟机管理软件
libvirt-0.10.2-29.el6.x86_64    # 用于管理虚拟机，它提供了一套虚拟机操作 API
virt-viewer-0.5.6-8.el6.x86_64  # 显示虚拟机的控制台 console
virt-top-1.0.4-3.15.el6.x86_64  # 类似于 top 命令，查看虚拟机的资源使用情况
virt-what-1.11-1.2.el6.x86_64  # 在虚拟机内部执行，查看虚拟机运行的虚拟化平台
qemu-img-0.12.1.2-2.415.el6.x86_64  # 用于操作虚拟机硬盘镜像的创建、查看和格式转化
```

在 KVM 环境中，以上 rpm 包都是必需的。读者可以检查当前系统里面是否已经安装 rpm，如果没有，可使用“yum install 组件名称”命令直接安装。

2. CentOS 7 宿主机系统安装及配置

CentOS 7 宿主机的安装与 CentOS 6 的系统安装方式类似，系统安装完之后，确认已经安装了如下的 rpm 相关包。Libvirt 还包含了很多工具的库，可以使用 yum install libvirt* 命令安装。

```
[root@KVM-host-CentOS7 ~]# rpm -qa|grep -E 'qemu-img|libvirt-[0-9]|virt-install'
qemu-img-1.5.3-60.el7_0.10.x86_64
virt-install-0.10.0-20.el7.noarch
libvirt-1.1.1-29.el7_0.3.x86_64
```

1107114476@qq.com
dolphinebook.com

```
[root@KVM-host-CentOS7 ~]# lsmod |grep KVM  # 查看 KVM 模块是否载入
KVM_intel 138567  6
KVM 441119  1 KVM_intel
```

Libvirt 及 guestfish 相关的工具在平时的运维过程中经常会用到，建议使用 yum install libguest* libvirt* 命令安装。

2.3　第一台虚拟机安装

本节介绍如何开始安装第一台虚拟机。首先需要新建一台虚拟机，然后通过诸如物理机安装的方式为虚拟机安装操作系统。但是一般不会使用 pxe、cobbler 等网络引导方式安装，因为这样安装虚拟机系统太慢了。虚拟化相比于物理机，其中一个优势就是创建快速。所以，一般都会使用 ISO 镜像文件安装第一台虚拟机，然后将这台虚拟机做成虚拟机模板，之后的虚拟机都是由这个模板生成的。后面第 16 章会详细介绍虚拟机模板的创建。本节将介绍两个 KVM 虚拟化中常用的管理工具，用它们来创建虚拟机。

模板的概念在虚拟化中非常重要，实际上绝大多数虚拟机都是通过模板克隆出来的，而不是像物理机那样安装出来的。有的云管理平台还有镜像市场的概念，就像应用商店，可以上传、下载各种模板。

1. Virt-Manager 使用介绍

Virt-Manager 是一个图形化的虚拟机管理工具，它提供了一个简易的虚拟机操作界面。要使用它，需要先安装图形化界面，下面以 CentOS 6.5 系统为例，需要安装如下组件。

```
[root@KVM-host ~]#yum groupinstall -y "Desktop" "Desktop Platform" "Desktop
Platform Development" "Fonts" "General Purpose Desktop" "Graphical Administration
Tools" "Graphics Creation Tools" "Input Methods" "X Window System" "Chinese Support
[zh]" "Internet Browser"
```

一般来说，服务器都是在 IDC 机房中的，为了看到宿主机的图形化界面，还需要安装配置 VNC。

```
[root@KVM-host ~]#yum install -y tigervnc
[root@KVM-host ~]#yum install -y tigervnc-server
 <!-- 安装 VNC 程序软件包 tigervnc 和 tigervnc-server  -->
[root@KVM-host ~]#vim /etc/sysconfig/vncservers
VNCSERVERS="1:root" # 配置宿主机 VNC 虚拟机显示器为 1，端口是 5901
VNCSERVERARGS[2]="-geometry 800x600 -nolisten tcp -localhost"
# 虚拟机显示器 [2] 监听的 IP 是 0.0.0.0，虚拟显示器 [1] 监听 127.0.0.1，屏幕分辨率为 800×600
[root@KVM-host ~]# vncpasswd  # 设置 VNC 密码
Password:
Verify:
```

1107114476@qq.com
dolphinebook.com

```
[root@KVM-host ~]# service vncserver restart  #启动 vnc-server
Shutting down VNC server:                                  [  OK  ]
Starting VNC server: 1:root xauth:  creating new authority file /root/.Xauthority
New 'whcq-netinst-121:1 (root)' desktop is KVM-host:1
Creating default startup script /root/.vnc/xstartup
Starting applications specified in /root/.vnc/xstartup
Log file is /root/.vnc/ KVM-host:1.log
                        [  OK  ]
```

配置完成之后，使用 VNC Viewer 登录宿主机的图形界面，在 VNC Server 组合框中输入宿主机的“IP:端口”，端口是之前配置的，本例中输入 192.168.106.221:5901，如图 2-3 所示。

提示 VNC Viewer 使用中，有时候会出现 VNC Viewer 界面闪退，可以通过修改网络速度解决这个问题。在如图 2-4 所示的位置，取消勾选该复选框，调节滑块到 Best quality。

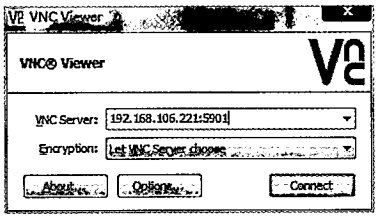


图 2-3 使用 VNC Viewer 登录宿主机图形界面

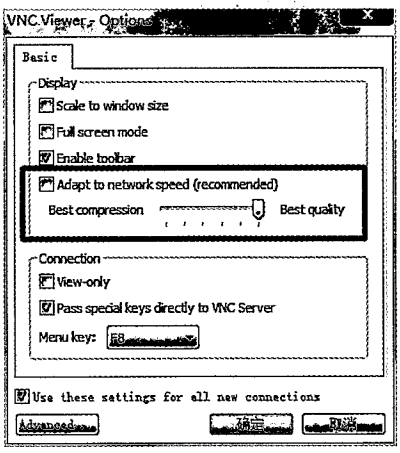


图 2-4 VNC Viewer 配置网络速率

VNC Viewer 配置完成之后，回到图 2-3 所示的界面，单击 Connect 按钮，出现如图 2-5 所示界面，提示输入密码，此处的密码是之前通过 vncpasswd 命令设置的 VNC Server 密码。

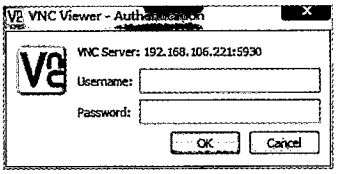


图 2-5 VNC Viewer 登录界面

输入正确密码之后，应该就可以登录成功了，出现如图 2-6 所示界面。

1107114476@qq.com
dolphinebook.com

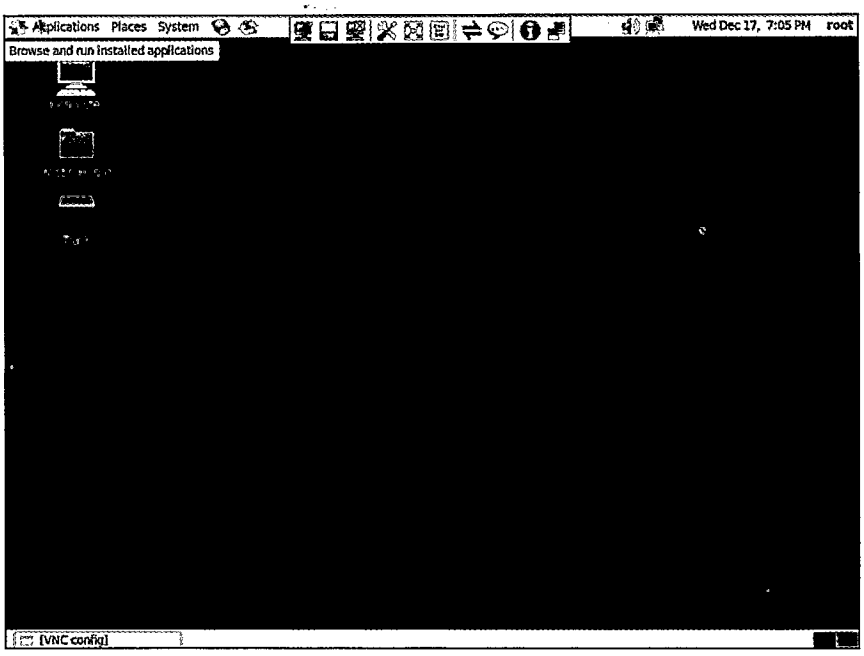


图 2-6　宿主机 VNC 登录之后界面

在菜单中，选择 Applications 菜单中的 System Tools 子菜单，选择打开 Virtual Machine Manager，如图 2-7 所示。其中列出了当前宿主机上所有的虚拟机，并显示了简单的 CPU 利用率信息。在图形界面中可以对虚拟化做常规操作，如创建、删除、编辑、配置及查看远程终端等。

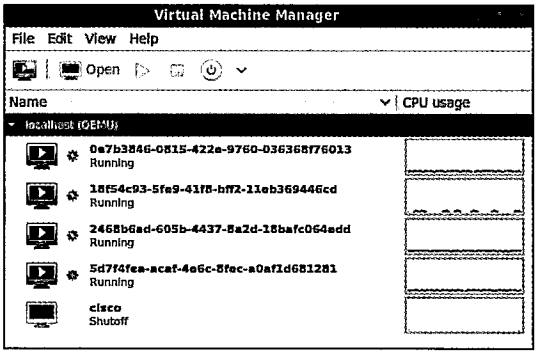


图 2-7　Virtual Machine Manager 界面

2. virt-install 命令使用介绍

virt-install 是一个在命令行创建 KVM 虚拟机的工具，使用 virt-install 配合一些配置参数，最终可以生成一个完整的 .xml 虚拟机配置文件。

```
#virt-install --name=testvm --ram=2048 --vCPUs=4 --os-type=Windows  --hvm
--cdrom=/root/W2003cnent.iso --file=/root/SDG100.img --file-size=10
```

1107114476@qq.com
dolphinebook.com

```
--bridge=br0  --vnc --vncport=5920
```

参数说明如下。

- --name：设置虚拟机名称。
- --ram：配置虚拟机内存，单位是 MB。
- --vCPUs：配置 CPU 个数。
- --hvm：配置使用全虚拟化。
- --os-type：指定操作系统类型，如 Linux、Windows。
- --cdrom：使用 cdrom 安装系统，指定 ISO 位置。
- --file：设置虚拟机硬盘文件路径。
- --file-size：配置虚拟机硬盘文件大小，单位是 GB。
- --bridge：配置桥接的网卡。
- --vnc：打开 VNC 支持。
- --vncport：配置 VNC 端口。

执行上述命令之后，virt-install 会创建一台名为 testvm 的虚拟机，并使用 W2003cnent.iso 镜像文件安装系统。此时使用 VNC Viewer，在 VNC Server 中输入宿主机 ip:vncport，便可登录虚拟机的控制台，此时虚拟机开始从 ISO 引导，安装虚拟机系统的步骤和安装普通服务器系统是一样的，这里就不详述了。

3. Windows 虚拟机安装注意事项

第一次安装 Windows 虚拟机的时候，经常会碰到以下几个问题。

（1）qcow2 格式的磁盘如何操作

Virt-Manager 默认创建的磁盘格式是 RAW 格式，如果需要使用 qcow2 格式的磁盘，必须用 qemu-img create 手工先创建一个 qcow2 格式的磁盘镜像。

```
qemu-img create Windows-test.qcow2 -f qcow2 50G
```

然后如图 2-8 所示，在 Virt-Manager 上指定 qcow2 格式。

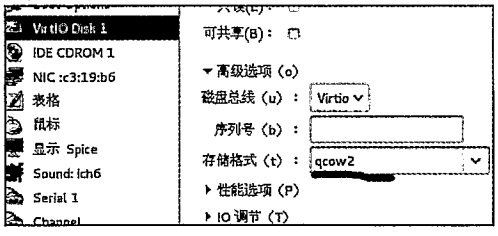


图 2-8 Virt-Manager 需要手工选择 qcow2 方式

提示 在使用 virt-install 命令，磁盘镜像格式为 qcow2 时，在 virt-install 命令中要特别指明磁盘格式，否则会出现镜像复制之后虚拟机系统不能启动的现象，这往往是初学者容易忽视的地方。

（2）光驱自动消失问题

Windows 系统安装的时候，重启后找不到光盘。

1107114476@qq.com
dolphinebook.com

KVM 新创建虚拟机，第一次挂载的光驱，重启后自动消失，这是一个功能，专门针对 Linux 系统，但是 Windows 系统安装的时候需要多次重启，所以安装时第一次重启后，会出现如图 2-9 所示的界面，需要再手工挂载一下 Windows 系统 ISO 镜像。

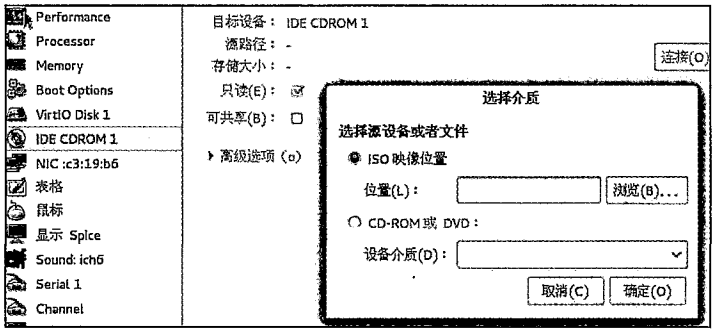


图 2-9　Virt-Manager 需要手工选择再加载光驱一次

也可以修改虚拟机的 xml 配置文件，光驱配置的 xml 文件如下：

```
<disk type='file' device='cdrom'>
<driver name='qemu' type='raw' cache='none'/>
<source file='/home/CentOS-7.0-1406-x86_64-DVD.iso'/>
<target dev='hdb' bus='ide'/>
<readonly/>
</disk>
```

（3）鼠标不同步问题

Windows 在 KVM 上会出现鼠标不同步问题，如图 2-10 所示，再添加一个 USB 鼠标设备就可以解决。

图 2-10　Virt-Manager 需要手工添加 USB 指针设备

添加一个 USB 鼠标设备的 xml 文件如下：

```
<input type='tablet' bus='usb'/>
```

提示　如果添加两次 USB 设备，Windows 系统虚拟机系统启动会蓝屏，所以只能添加一次 USB 设备。

4. Linux 虚拟机安装注意事项

Linux 系统的安装除了会碰到 Windows 系统安装的 qcow2 磁盘格式问题、鼠标不同步的问题，Linux 虚拟机的安装还可以使用一种非常有意思的方式，就是可以直接指定内核文件

1107114476@qq.com
dolphinebook.com

路径，然后直接加载，如图 2-11 所示。

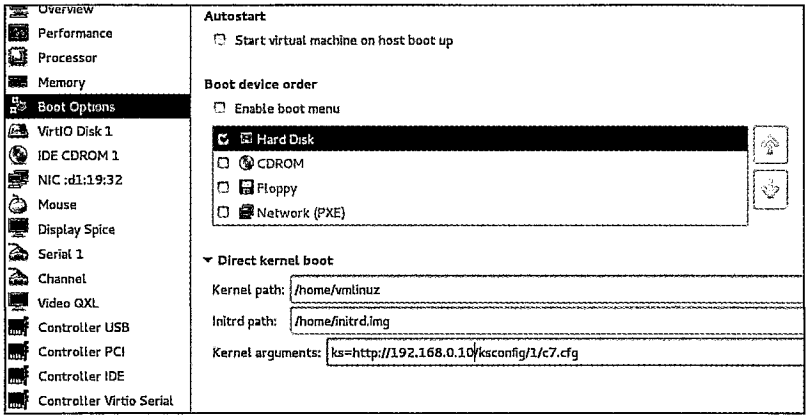


图 2-11 通过指定内核文件安装虚拟机

提示 内核文件需要使用安装光盘中的 images 下的 pxe 内核文件。

系统引导起来之后，会根据 kickstart 文件来下载系统，然后安装，如图 2-12 和图 2-13 所示。

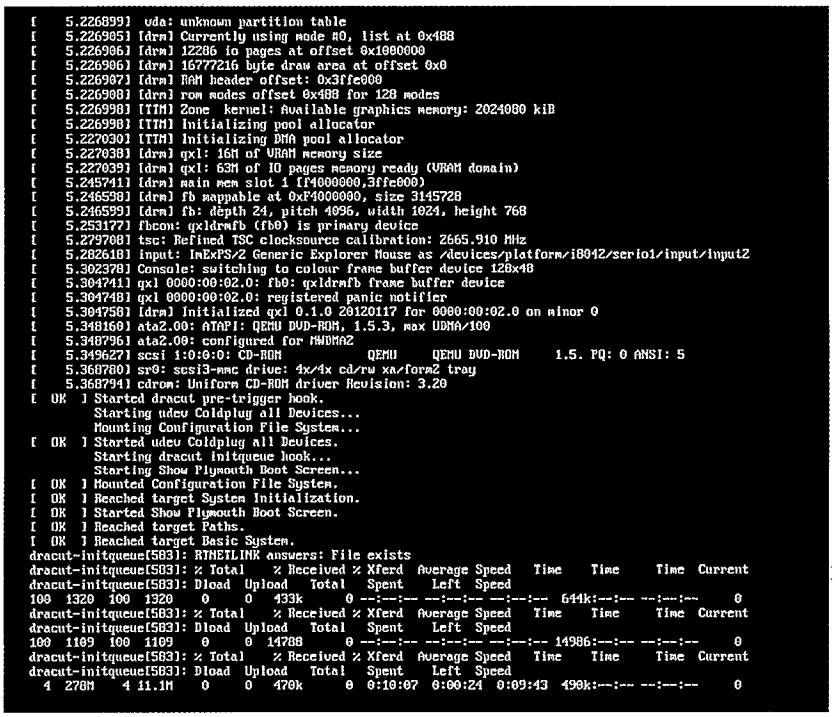


图 2-12 · 虚拟机启动后根据 kickstart 文件路径下载安装文件

1107114476@qq.com
dolphinebook.com

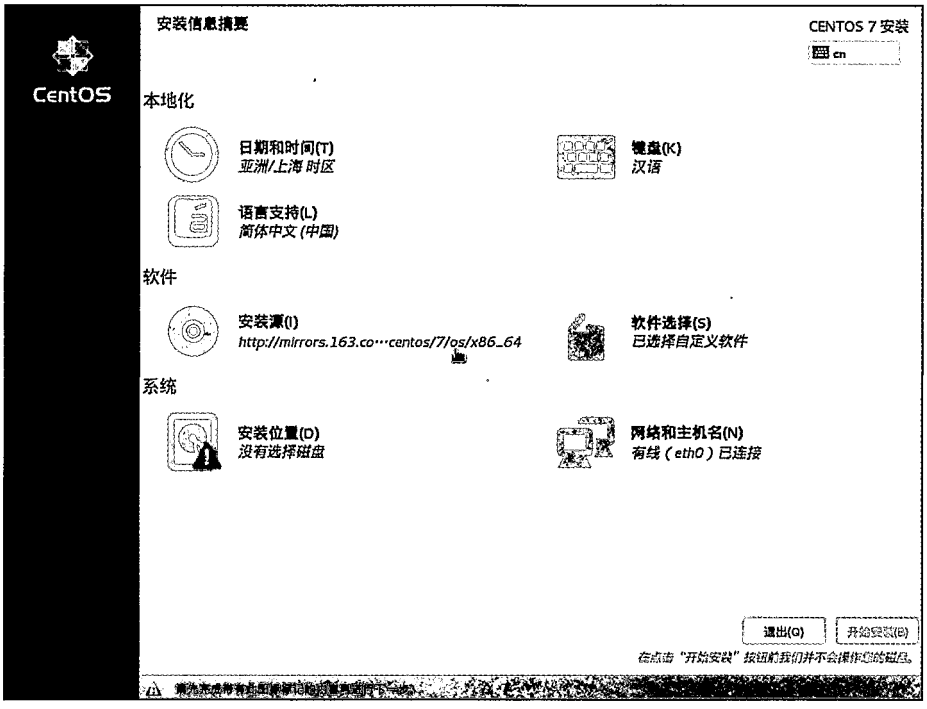
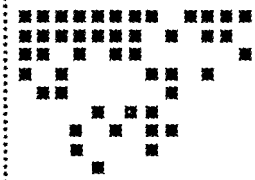


图 2-13　随后根据 kickstart 文件安装系统

2.4　本章小结

本章介绍了如何安装宿主机、如何安装虚拟机及虚拟机安装时的一些注意事项。读者可以正式开始自己的虚拟化环境搭建了，在搭建的时候，还有许多技术点需要了解，下一章将为读者介绍 KVM 虚拟机 CPU、内存方面的技术及应用场景。

1107114476@qq.com
dolphinebook.com



Chapter 3 第 3 章

CPU、内存虚拟化技术与应用场景

本章将为读者详细介绍 KVM 虚拟化中的 CPU 和内存技术，以及每种技术的应用场景。

KVM 虚拟机 CPU 的软件调优首先需要对 NUMA 技术有一定的了解，调优的主要手段就是虚拟机对物理机 CPU 逻辑核的手工绑定。

CPU 的 Nested 特性使用也是非常有意思的一个特性，KVM 虚拟机的嵌套在理论上可以无限层地嵌套下去，只要物理机的性能足够。

内存方面的调优手段主要是 KSM，即相同内存页合并、内存气球技术及大页内存的使用。

3.1 NUMA 技术与应用

NUMA 是一种解决多 CPU 共同工作的技术方案，首先回顾一下多 CPU 共同工作技术的架构历史。多 CPU 共同工作主要有 3 种架构，分别是 SMP、MPP、NUMA 架构。SMP、MPP、NUMA 都是为了解决多 CPU 共同工作的问题。

1. SMP 技术

早期的时候，每台服务器都是单 CPU，随着技术的发展，出现了多 CPU 共同工作的需求，最早的多 CPU 技术是 SMP。

如图 3-1 所示，SMP 即多个 CPU 通过一个总线访问存储器，因此 SMP 系统有时也被称为一致内存访问（UMA）结构体系。一致性意指无论在什么时候，处理器只能为内存的每个

1107114476@qq.com
dolphinebook.com

数据保持或共享唯一一个数值。

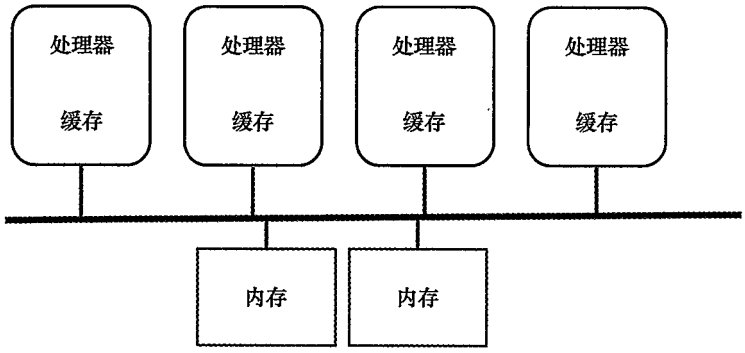


图 3-1　多 CPU 的 SMP 架构

SMP 的缺点是扩展性有限，因为在存储器接口达到饱和的时候，增加处理器并不能获得更高的性能，因此 SMP 方式支持的 CPU 个数有限。

2. MPP 模式

MPP 模式则是一种分布式存储器模式，能够将更多的处理器纳入一个系统的存储器。一个分布式存储器模式具有多个节点，每个节点都有自己的存储器，可以配置为 SMP 模式，也可以配置为非 SMP 模式。单个的节点相互连接起来就形成了一个总系统。MPP 可以近似理解成一个 SMP 的横向扩展集群。MPP 一般要依靠软件实现。

3. NUMA 技术

如图 3-2 所示，NUMA 模式则是每个处理器有自己的存储器，每个处理器也可以访问别的处理器的存储器。

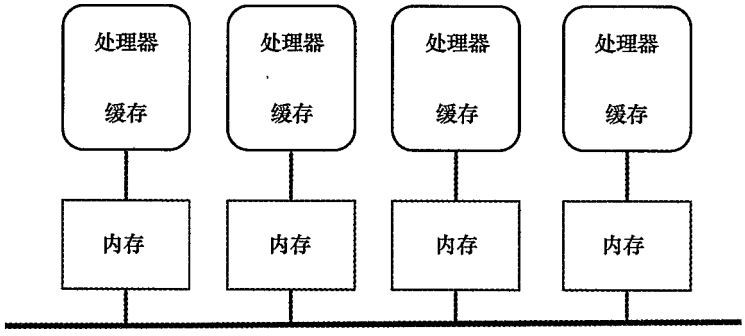


图 3-2　多 CPU 的 NUMA 架构

图 3-3 所示是多核 CPU 的 NUMA 架构。

NUMA-Q 是 IBM 最早将 NUMA 技术应用到 i386 上的商业方案，可以支持更多的 X86 CPU 一起工作。

1107114476@qq.com
dolphinebook.com

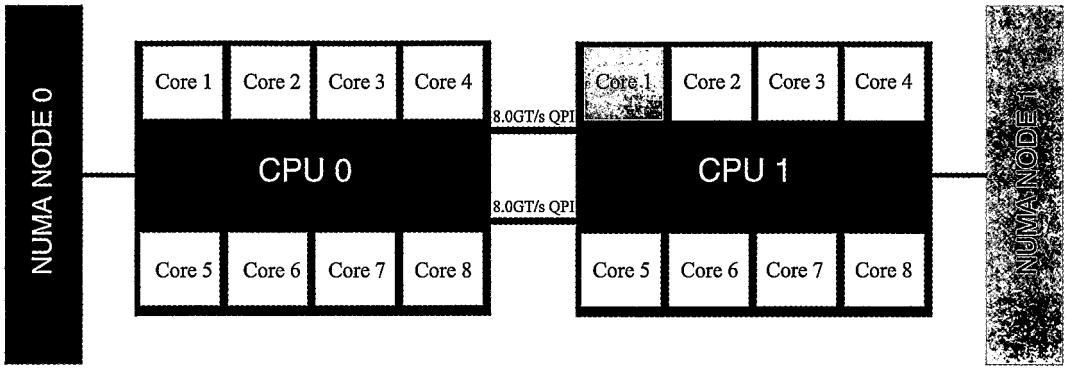


图 3-3 多核 NUMA CPU 架构（来自于 Intel 网站资料）

3.1.1 KVM 虚拟机 NUMA 调优

1. 宿主机的 NUMA 信息查看与配置

因为 NUMA 架构每个处理器都可以访问自己和别的处理器的存储器，访问自己的存储器要比访问别的存储器快很多，速度相差 10 ~ 100 倍，所以 NUMA 调优的目标就是让处理器尽量访问自己的存储器，以提高处理速度。

如图 3-4 所示，通过 numactl --hardware 命令可以看到当前 CPU 硬件的情况，CPU 有两颗，每颗有 8 个核，每个核有 64GB 内存可以使用。

```
[root@localhost ~]# numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23
node 0 size: 65501 MB
node 0 free: 689 MB
node 1 cpus: 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
node 1 size: 65535 MB
node 1 free: 4829 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

图 3-4 CPU NUMA 情况

使用 numastat 命令可以查看每个节点的内存统计。

```
                  node0           node1
numa_hit      4911255660       490070817 # 使用本节点内存次数
numa_miss       20552767       946830474 # 计划使用本节点内存而被调度到其他节点次数
numa_foreign   946830474        20552767 # 计划使用其他节点内存而使用本地内存次数
interleave_hit     14625           14612 # 交叉分配使用的内存中使用本节点的内存次数
local_node     911244851       490037204 # 在本节点运行的程序使用本节点内存次数
other_node      20563576       946864087 # 在其他节点运行的程序使用本节点内存次数
```

numastat 命令使用 -c 参数可以查看相关进程的 NUMA 内存使用情况。

1107114476@qq.com
dolphinebook.com

```
numastat -c qemu-kvm
Per-node process memory usage (in MBs)
PID               Node 0 Node 1 Total
---------------  ------ ------ -----
3135 (qemu-kvm)    2317   1443  3760
3187 (qemu-kvm)    2508   2454  4962
3245 (qemu-kvm)    4424   2287  6711
20830 (qemu-kvm)    942   1124  2066
31717 (qemu-kvm)    852   1114  1967
---------------  ------ ------ -----
Total             11044   8422 19466
```

Linux 系统默认是自动 NUMA 平衡策略。如果要关闭 Linux 系统的自动平衡，可以使用如下命令：

```
# echo 0 > /proc/sys/kernel/numa_balancing
```

如果要开启自动 NUMA 平衡策略，可以使用如下命令：

```
echo 1 > /proc/sys/kernel/numa_balancing
```

2. 虚拟机 NUMA 信息查看与配置

使用 virsh numatune 命令可以查看或者修改虚拟机的 NUMA 配置。

```
virsh # numatune 4
numa_mode      : strict
numa_nodeset   : 0-1
```

NUMA 工作方式可以是 strict 指定 CPU，或者 auto 使用系统的 numad 服务。

```
<numatune>
        <memory mode='strict' placement='auto'/>
</numatune>
<numatune>
        <memory mode='strict' nodeset='0,2-3'/>
</numatune>
```

可以使用 numatune 命令配置虚拟机的 NUMA。

```
virsh numatune rhel7 --nodeset '0, 2-3'
```

vpcu 的设置如下：

```
<vcpu placement='auto'>8</vcpu>
<vcpu placement='static' cpuset='0-10,^5'>8</vcpu>
```

<vcpu> 和 <numatune> 需要保持一致，<numatune> 配置的是物理 CPU，<vcpu> 配置的是 CPU 的核，包括超线程产生的核。<numatune> 使用 static 模式，<nodeset> 也必须是。

可以设置一个虚拟机给 32 个虚拟 CPU，但是一开始只能使用 8 个，然后根据系统压力，

1107114476@qq.com
dolphinebook.com

热添加 CPU 给虚拟机。

```
<vcpu placement='auto' current='8'>32</vcpu>
```

也可以给每个虚拟机 CPU 指定具体的物理机 CPU pinning 策略。

```
<cputune>
        <vcpupin vcpu="0" cpuset="1-4,^2"/>
        <vcpupin vcpu="1" cpuset="0,1"/>
        <vcpupin vcpu="2" cpuset="2,3"/>
        <vcpupin vcpu="3" cpuset="0,4"/>
</cputune>
```

也可以使用 emulatorpin 的方式。emulatorpin 标签可以指定一个特定的物理 CPU 范围，比如一个物理 CPU 内部所有的核，使虚拟机使用的 CPU 和存储器都在一个物理机 CPU 内部，xml 文件如下：

```
<cputune>
        <emulatorpin cpuset="1-3"/>
</cputune>
```

可以在线调整，命令方式如下：

```
virsh emulatorpin CentOS7 1-3
```

1 ~ 3 的核都在一个物理 CPU 内部。也可以设置虚拟机对 NUMA 资源的使用。

```
<cpu>
        ...
    <numa>
      <cell cpus='0-3' memory='512000'/>
      <cell cpus='4-7' memory='512000'/>
    </numa>
    ...
</cpu>
```

标签项意义如下。

- cell：numa 的 cell 或者 numa 节点。
- cpus：一个物理 CPU 可以使用的 CPU 范围。
- memory：可以使用的内存大小，单位为 KB。

3. 虚拟机 NUMA 和内存 KSM

KSM 技术可以合并相同的内存页，即使是不同的 NUMA 节点，如果需要关闭跨 NUMA 节点的内存合并，设置 /sys/kernel/mm/ksm/merge_across_nodes 参数为 0。或者可以关闭特定虚拟机的 KSM 内存合并，在虚拟机的 xml 配置文件中添加以下内容就可以：

```
<memoryBacking>
        <nosharepages/>
</memoryBacking>
```

1107114476@qq.com
dolphinebook.com

3.1.2　CPU 绑定操作方法

CPU 绑定是一项非常神奇的技术，最神奇的地方就是可以在线配置，并且立即生效，可以解决生产环境 CPU 利用率严重不平均的问题。

1. CPU 信息查看

使用 virsh vcpuinfo 命令查看虚拟机 VCPU 和物理 CPU 的对应关系。

```
virsh  vcpuinfo 21
VCPU: 0
CPU: 25
State: running
CPU time: 10393.0s
CPU Affinity: --------yyyyyyyy--------yyyyyyyy

VCPU: 1
CPU: 8
State: running
CPU time: 7221.2s
CPU Affinity: --------yyyyyyyy--------yyyyyyyy

VCPU: 2
CPU: 26
State: running
CPU time: 7321.4s
CPU Affinity: --------yyyyyyyy--------yyyyyyyy

VCPU: 3
CPU: 8
State: running
CPU time: 6808.9s
CPU Affinity: --------yyyyyyyy--------yyyyyyyy
```

可以看到 VCPU 0 被调度到物理机 CPU 25 上，目前是使用状态，使用时间是 10 393.0s。最后一行是 CPU 亲和性的信息：

```
CPU Affinity: --------yyyyyyyy--------yyyyyyyy
```

yyyyyyy 表示可以使用的物理 CPU 内部的逻辑核，可以看到这台虚拟机允许在 8 ~ 15、24 ~ 31 这些物理 CPU 之间调度。为什么不能使用 0 ~ 7、16 ~ 23 这些 CPU 呢？是因为系统的自动 NUMA 平衡服务在发生作用，一个虚拟机默认只能使用同一颗物理 CPU 内部的逻辑核。

使用 emulatorpin 命令可以查看虚拟机可以使用哪些物理逻辑 CPU。

```
virsh # emulatorpin 21
emulator: CPU Affinity
----------------------------------
       *: 0-31
```

可以看到 0 ~ 31 都可以使用，意味着可以强制将 VCPU 调度到任何物理 CPU 核上。

1107114476@qq.com
dolphinebook.com

2. 在线绑定虚拟机的 CPU

（1）让虚拟机只能在部分物理 CPU 之间调度

可以强制让虚拟机只能在部分物理 CPU 之间调度。比如使用下面的命令，使编号为 21 的虚拟机 CPU 在 26 ~ 31 这些物理 CPU 之间调度：

```
virsh  emulatorpin 21 26-31 --live
```

通过以下命令查看绑定是否生效：

```
virsh  emulatorpin 21
emulator: CPU Affinity
----------------------------------
       *: 26-31
```

我们可以通过 vcpuinfo 命令和虚拟机的 xml 配置文件验证。

1）查看虚拟机的 VCPU 调度信息。

```
virsh  vcpuinfo 21
VCPU: 0
CPU: 28
State: running
CPU time: 10510.5s
CPU Affinity: ----------------------------yyyyyy

VCPU: 1
CPU: 28
State: running
CPU time: 7289.7s
CPU Affinity: ----------------------------yyyyyy

VCPU: 2
CPU: 27
State: running
CPU time: 7391.3s
CPU Affinity: ----------------------------yyyyyy

VCPU: 3
CPU: 27
State: running
CPU time: 6886.6s
CPU Affinity: ----------------------------yyyyyy
```

2）通过虚拟机的 xml 配置文件查看 VCPU 调度信息。

```
virsh # dumpxml 21
<domain type='kvm' id='21'>
  <name>cacti-230</name>
  <uuid>23a6455c-5cd1-20cd-ecfe-2ba89be72c41</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
```

1107114476@qq.com
dolphinebook.com

```
  <vcpu placement='static'>4</vcpu>
  <cputune>
    <emulatorpin cpuset='26-31'/>
  </cputune>
...
```

（2）强制 VCPU 和物理机 CPU 一对一绑定

我们也可以强制 VCPU 和物理机 CPU 一对一地绑定。比如强制 VCPU 0 和物理机 CPU 28 绑定，强制 VCPU 1 和物理机 CPU 29 绑定，强制 VCPU 2 和物理机 CPU 30 绑定，强制 VCPU 3 和物理机 CPU 31 绑定，命令如下：

```
virsh  vcpupin 21 0 28
virsh  vcpupin 21 1 29
virsh  vcpupin 21 2 30
virsh  vcpupin 21 3 31
```

查看 xml 文件，验证 <cputune> 块中的 vcpupin 配置信息。

```
virsh # dumpxml 21
<domain type='kvm' id='21'>
  <name>cacti-230</name>
  <uuid>23a6455c-5cd1-20cd-ecfe-2ba89be72c41</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static'>4</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='28'/>
    <vcpupin vcpu='1' cpuset='29'/>
    <vcpupin vcpu='2' cpuset='30'/>
    <vcpupin vcpu='3' cpuset='31'/>
    <emulatorpin cpuset='26-31'/>
  </cputune>
...
```

验证一下虚拟机重启后，配置是否继续生效。将虚拟机关机再开机，检查一下配置是否一直有效，可以看到配置一直生效。

```
virsh  vcpuinfo 22
VCPU: 0
CPU: 28
State: running
CPU time: 1.8s
CPU Affinity: ----------------------------y---
VCPU: 1
CPU: 29
State: running
CPU time: 0.0s
CPU Affinity: ----------------------------y--
VCPU: 2
CPU: 30
```

1107114476@qq.com
dolphinebook.com

```
State: running
CPU time: 0.0s
CPU Affinity: --------------------------------y-
VCPU: 3
CPU: 31
State: running
CPU time: 0.0s
CPU Affinity: --------------------------------y
```

3. CPU 绑定技术的原理

CPU 绑定实际上是 Libvirt 通过 CGroup 来实现的，通过 CGroup 直接去绑定 KVM 虚拟机进程也可以。通过 CGroup 不仅可以做 CPU 绑定，还可以限制虚拟机磁盘、网络的资源控制，本书第 6 章将详细介绍 CGroup 在 KVM 虚拟化中的使用。

4. CPU 绑定技术的应用场景

在 CPU 压力比较大的时候使用，就要考虑使用 CPU 绑定技术。根据笔者的经验，不同的业务场景，使用 CPU 绑定效果不一样，有的业务性能有 20% 以上的提升。

笔者还碰到过一个案例，在一台 32 核的宿主机上，有一台虚拟机压力很大，宿主机的前 3 个 CPU 利用率达到了 90% 以上，中间几个 CPU 利用率为 60% ~ 80%，后面第 27 ~ 31 的 6 个 CPU 利用率只有 40%，当时业务已经出现性能问题，通过在线使用 CPU 绑定技术，将压力大的虚拟机绑定到了后面几个 CPU 上，解决了问题。

所以 CPU 绑定技术适用于以下场景：

- 系统的 CPU 压力比较大。
- 多核 CPU 压力不平衡，可以通过 cpu pinning 技术人工进行调配。

NUMA 在虚拟化应用之外的提示：

某些应用，比如数据库，为了保证性能，需要尽量使用更多的内存，如果使用系统默认的 NUMA 自动平衡策略，有可能一个 CPU 的内存消耗光，另外一个 CPU 还有大量的内存可以使用，但是系统却去调度 swap 来使用，造成性能严重降低。使用这些内存消耗型的应用时，可以考虑直接将系统的 NUMA 自动平衡策略关闭。

5. CPU 绑定对性能提升的测试案例

CPU 绑定到底对性能影响有多大，笔者进行了一个简单的测试。

（1）测试环境

硬件：Intel(R) Xeon(R) CPU X5650 @ 2.67GHz 2 颗。

软件：CentOS 7 更新到内核 3.10.0-123.8.1.el7.x86_64。

虚拟机：CentOS 7 更新到内核 3.10.0-123.8.1.el7.x86_64。

虚拟机：CPU 一颗。

测试工具：unixbench 5.1.2（使用方法及分项测试意义在第 15 章有介绍）。

1107114476@qq.com
dolphinebook.com

（2）测试结果

1）不做 CPU 绑定 unixbench 的得分：

```
1 CPU in system; running 1 parallel copy of tests
Dhrystone 2 using register variables       28890881.0 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     3880.4 MWIPS (9.0 s, 7 samples)
...
                                                                 ========
System Benchmarks Index Score                                     1444.7
```

2）做了 CPU 绑定 unixbench 的得分：

```
1 CPU in system; running 1 parallel copy of tests
Dhrystone 2 using register variables       29812559.6 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     3928.7 MWIPS (8.9 s, 7 samples)
...
                                                                 ========
System Benchmarks Index Score                                     1464.1
```

（3）简单比较

- unixbench 综合得分（性能提升 1.34%）：绑定——1464.1；不绑定——1444.7。
- CPU 浮点运算（性能提升 1.24%）：绑定——3928.7；不绑定——3880.4。

6. CPU 绑定的一个生产环境使用案例

图 3-5 和图 3-6 所示是一台 32 核的宿主机没有做 CPU 绑定之前的利用率，从第一个核和最后一个核 CPU 利用率的比较中可以看出差距较大。

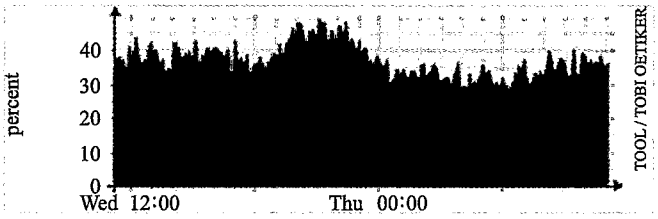


图 3-5　未做 CPU 绑定之前的 CPU 1 利用率

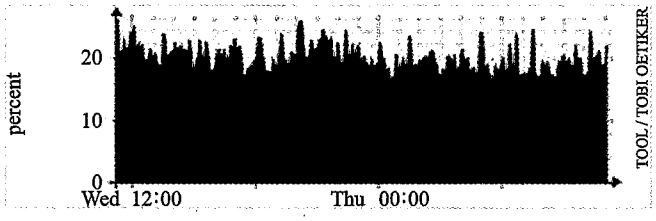


图 3-6　未做 CPU 绑定之前的 CPU 31 利用率

图 3-7 和图 3-8 所示是做了 CPU 绑定之后，第一个核和最后一个核 CPU 利用率差距已

1107114476@qq.com
dolphinebook.com

经减少，拉平了整体的 CPU 利用率。

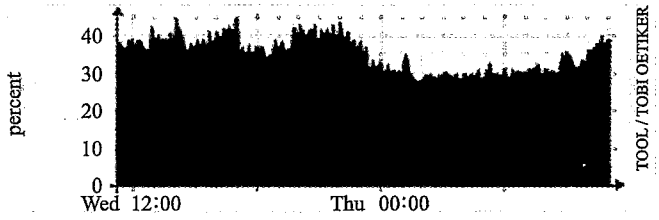


图 3-7 做 CPU 绑定之后 CPU 1 利用率

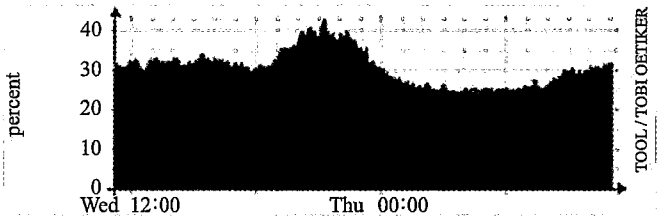


图 3-8 做 CPU 绑定之后 CPU 31 利用率

3.2 CPU 热添加与应用

CPU 热添加最早的时候是大型机上的硬件技术，随后一些高端的 X86 服务器也支持硬件的 CPU 热添加。CPU 热添加也需要操作系统的支持，在虚拟化上 CPU 热添加反而变得容易，因为虚拟化只需要修改软件配置就可以，不需要做硬件配置变更。

CPU 热添加是 CentOS 7 的一个新特性，Linux 系统要求宿主机和虚拟机都是 CentOS 7，Windows 虚拟机系统要求是 Windows Server 2008 数据中心版或者 Windows Server 2012 标准版和数据中心版。

1. Linux 系统的 CPU 热添加

如图 3-9 所示，在给虚拟机分配 CPU 的时候，就要预留足够的 CPU。

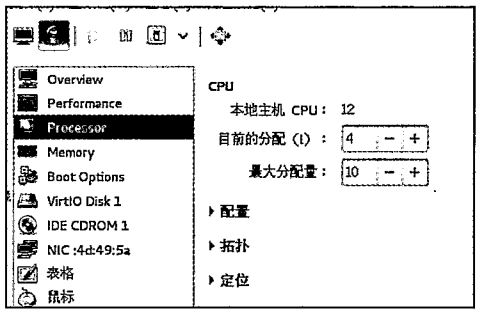


图 3-9 CPU 热添加配置

1107114476@qq.com
dolphinebook.com

如图 3-10 所示，在虚拟机中可以看到 4 个 CPU。

```
[root@localhost ~]# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:        123          0          0          0   IO-APIC-edge      timer
  1:         10          0          0          0   IO-APIC-edge      i8042
  6:          3          0          0          0   IO-APIC-edge      floppy
  8:          0          0          0          0   IO-APIC-edge      rtc0
  9:          0          0          0          0   IO-APIC-fasteoi   acpi
 10:       4007          0          0          0   IO-APIC-fasteoi   virtio3
 11:         62         10          0          0   IO-APIC-fasteoi   uhci_hcd:usb1, qxl
 12:        144          0          0          0   IO-APIC-edge      i8042
 14:          0          0          0          0   IO-APIC-edge      ata_piix
 15:        144          0          0         67   IO-APIC-edge      ata_piix
 40:          0          0          0          0   PCI-MSI-edge      virtio2-config
 41:       7342          0        140          0   PCI-MSI-edge      virtio2-requests
 42:          0          0          0          0   PCI-MSI-edge      virtio4-config
 43:        205          0          0          0   PCI-MSI-edge      virtio4-requests
 44:          0          0          0          0   PCI-MSI-edge      virtio0-config
 45:        294          0          0          0   PCI-MSI-edge      virtio0-input.0
 46:          1          0          0          0   PCI-MSI-edge      virtio0-output.0
 47:          0          0          0          0   PCI-MSI-edge      virtio5-config
 48:        205          0          0          0   PCI-MSI-edge      virtio5-requests
 49:          0          0          0          0   PCI-MSI-edge      virtio6-config
 50:        205          0          0          0   PCI-MSI-edge      virtio6-requests
 51:          0          0          0          0   PCI-MSI-edge      virtio7-config
```

图 3-10　CPU 热添加前进行虚拟机 CPU 查看

把 CPU 在线修改成 5 个：

```
virsh setvcpus CentOS7 5 --live
```

在虚拟机里面将第 5 个 CPU 激活：

```
echo 1 >/sys/devices/system/cpu/cpu4/online
```

如图 3-11 所示，可以看到虚拟机的 CPU 已经变成了 5 个：

```
[root@localhost ~]# cat /proc/interrupts |less
           CPU0       CPU1       CPU2       CPU3       CPU4
  0:        123          0          0          0          0   IO-APIC-edge      timer
  1:         10          0          0          0          0   IO-APIC-edge      i8042
  6:          3          0          0          0          0   IO-APIC-edge      floppy
  8:          0          0          0          0          0   IO-APIC-edge      rtc0
  9:          1          0          0          0          0   IO-APIC-fasteoi   acpi
 10:       4007          0          0          0          0   IO-APIC-fasteoi   virtio3
 11:         64         35          0          0          0   IO-APIC-fasteoi   uhci_hcd:usb1, qxl
 12:        144          0          0          0          0   IO-APIC-edge      i8042
 14:          0          0          0          0          0   IO-APIC-edge      ata_piix
 15:        144          0          0        233          0   IO-APIC-edge      ata_piix
 40:          0          0          0          0          0   PCI-MSI-edge      virtio2-config
 41:       7342          0        172          0          6   PCI-MSI-edge      virtio2-requests
 42:          0          0          0          0          0   PCI-MSI-edge      virtio4-config
 43:        205          0          0          0          0   PCI-MSI-edge      virtio4-requests
 44:          0          0          0          0          0   PCI-MSI-edge      virtio0-config
 45:        866          0          0         76          0   PCI-MSI-edge      virtio0-input.0
 46:          1          0          0          0          0   PCI-MSI-edge      virtio0-output.0
 47:          0          0          0          0          0   PCI-MSI-edge      virtio5-config
 48:        205          0          0          0          0   PCI-MSI-edge      virtio5-requests
 49:          0          0          0          0          0   PCI-MSI-edge      virtio6-config
 50:        205          0          0          0          0   PCI-MSI-edge      virtio6-requests
 51:          0          0          0          0          0   PCI-MSI-edge      virtio7-config
 52:        205          0          0          0          0   PCI-MSI-edge      virtio7-requests
 53:          0          0          0          0          0   PCI-MSI-edge      virtio8-config
 54:        205          0          0          0          0   PCI-MSI-edge      virtio8-requests
 55:          0          0          0          0          0   PCI-MSI-edge      virtio1-config
```

图 3-11　热添加 CPU 到 5 个

1107114476@qq.com
dolphinebook.com

我们也可以查看 cpuinfo 的信息来确认 CPU 增加了。

```
cat /proc/cpuinfo
...
processor       : 4
vendor_id       : GenuineIntel
cpu family      : 6
model           : 44
model name      : Intel(R) Xeon(R) CPU           X5650  @ 2.67GHz
stepping        : 2
microcode       : 0x1
cpu MHz         : 2665.908
cache size      : 4096 KB
physical id     : 4
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 4
initial apicid  : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                  pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb
                  rdtscp lm constant_tsc arch_perfmon rep_good nopl pni
                  pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 x2apic popcnt aes
                  hypervisor lahf_lm tsc_adjust
bogomips        : 5331.81
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

因为一开始预留的是 10 个，所以最多的时候，可以热添加 CPU 到 10 个。

```
virsh # setvcpus ct7 10 --live
```

在虚拟机上执行如下命令：

```
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu4/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu5/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu6/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu7/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu8/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu9/online
[root@localhost ~]# echo 1 >/sys/devices/system/cpu/cpu9/online
```

查看 cpuinfo，已经是 10 个 CPU 了。

```
cat /proc/cpuinfo  |grep processor |wc -l
10
```

1107114476@qq.com
dolphinebook.com

目前还不支持 CPU 热去除，可以在虚拟机里面关闭 CPU，使用如下命令：

```
echo 0 >/sys/devices/system/cpu/cpu4/online
```

但是在宿主机上如果要把 CPU 个数改小，是不支持的。

```
virsh # setvcpus ct7 4 --live
错误：内部错误：无法更改这个域的 vcpu 计数
```

2. Windows 系统的 CPU 热添加

Windows 系统也一样，开始的时候就要预留 CPU，需要的时候直接添加就可以了。如图 3-12 所示，未做 CPU 添加之前，有 4 个 CPU。

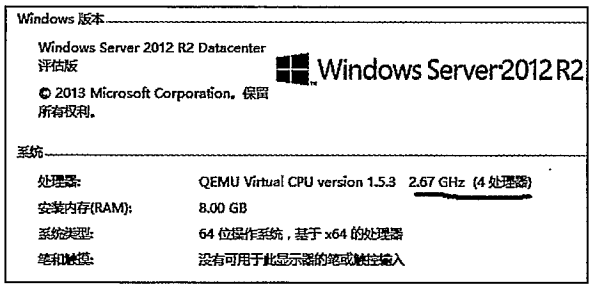


图 3-12　Windows 系统热添加 CPU 之前

将 CPU 在线添加到 6 个。

```
virsh # setvcpus window2012 6 --live
```

如图 3-13 所示，添加之后，虚拟机 CPU 已经变成 6 个。Windows 系统 CPU 添加后直接就可以自动刷新出来，不需要做特殊配置。同样 Windows 系统 CPU 也不能在线减少。

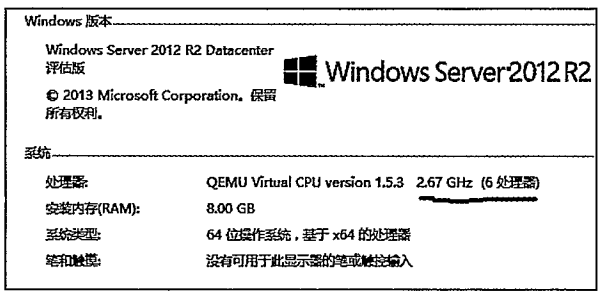


图 3-13　Windows 系统热添加 CPU 之后

3. CPU 热添加技术的应用场景

如果在虚拟机运行的应用非常重要，不能停机而性能严重不足的场景，CPU 热添加的技术是一个很好的解决方案。CPU 热添加的技术是比较新的技术，要求的宿主机、虚拟机操作

1107114476@qq.com
dolphinebook.com

系统比较新，还要待在生产环境中逐步实践。

3.3 CPU host-passthrough 技术与应用

1. 关于 KVM 虚拟机的 CPU 型号的定义

为了保证虚拟机在不同宿主机之间迁移时候的兼容性，Libvirt 对 CPU 提炼出标准的几种类型，在 /usr/share/libvirt/cpu_map.xml 中可以查到。cpu_map.xml 不仅是 CPU 型号，还有生产商信息、每种型号的 CPU 特性定义等信息。

```
<cpus>
  <arch name='x86'>
    <!-- vendor definitions -->
    <vendor name='Intel' string='GenuineIntel'/>
    <vendor name='AMD' string='AuthenticAMD'/>

    <!-- standard features, EDX -->
    <feature name='fpu'> <!-- CPUID_FP87 -->
      <cpuid function='0x00000001' edx='0x00000001'/>
    </feature>
    <feature name='vme'> <!-- CPUID_VME -->
      <cpuid function='0x00000001' edx='0x00000002'/>
    </feature>
...
<!-- models -->
    <model name='486'>
      <feature name='fpu'/>
      <feature name='vme'/>
      <feature name='pse'/>
    </model>
...
 <model name='Haswell'>
      <model name='SandyBridge'/>
      <feature name='fma'/>
      <feature name='pcid'/>
      <feature name='movbe'/>
      <feature name='fsgsbase'/>
      <feature name='bmi1'/>
      <feature name='hle'/>
      <feature name='avx2'/>
      <feature name='smep'/>
      <feature name='bmi2'/>
      <feature name='erms'/>
      <feature name='invpcid'/>
      <feature name='rtm'/>
    </model>
...
```

CentOS 6.6 所带的 Libvirt 主要规定了以下几种 CPU 型号：'486'、'pentium'、'pentium2'、'pe

1107114476@qq.com
dolphinebook.com

ntium3'、'pentiumpro'、'coreduo'、'pentiumpro'、'n270'、'coreduo'、'core2duo'、'qemu32'、'kvm32'、'cpu64-rhel5'、'cpu64-rhel6'、'kvm64'、'qemu64'、'Conroe'、'Penryn'、'Nehalem''Westmere'、'SandyBridge'、'Haswell'、'athlon'、'phenom'、'Opteron_G1'、'Opteron_G2'、'Opteron_G3'、'Opteron_G4'、'Opteron_G5'、'POWER7'、'POWER7_v2.1'、'POWER7_v2.3'。

2. CPU 模式配置

CPU 配置模式可以有以下几种。

（1）custom 模式

xml 配置文件如下：

```
<cpu mode='custom' match='exact'>
    <model fallback='allow'>kvm64</model>
 ...
    <feature policy='require' name='monitor'/>
  </cpu>
```

（2）host-model 模式

根据物理 CPU 的特性，选择一个最靠近的标准 CPU 型号。如果没有指定 CPU 模式，默认也是使用这种模式，xml 配置文件如下：

```
<cpu mode='host-model' />
```

（3）host-passthrough 模式

直接将物理 CPU 暴露给虚拟机使用，在虚拟机上完全可以看到的就是物理 CPU 的型号，xml 配置文件如下：

```
<cpu mode='host-passthrough'/>
```

使用 host-model 看到的 VCPU 如下：

```
processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 44
model name      : Westmere E56xx/L56xx/X56xx (Nehalem-C)
...
```

使用 host-passthrough 看到的 VCPU 如下：

```
processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 44
model name      : Intel(R) Xeon(R) CPU           X5650  @ 2.67GHz
```

可以看到，使用 host-model 模式，Libvir 会根据物理 CPU 的型号，从规定的 CPU 中选择一种最接近的 CPU 型号，而使用 host-passthrough 模式直接看到的就物理 CPU 的型号。

1107114476@qq.com
dolphinebook.com

3. CPU host-passthrough 技术的应用场景

HOST 技术适用于以下场景：

- 需要将物理 CPU 的一些特性传给虚拟机使用，比如使用虚拟机嵌套的 nested 技术的时候。
- 需要在虚拟机里面看到和物理 CPU 一模一样的 CPU 品牌型号，这个在一些公有云很有意义，用户体验比较好。

使用 CPU host-passthrough 技术需要注意，不同型号 CPU 的宿主机之间虚拟机不能迁移。

3.4 CPU Nested 技术与配置方法

Nested 技术，简单地说，就是在虚拟机上运行虚拟机，即 KVM on KVM。KVM 虚拟机嵌套和 VMWare 原理不同，VMWare 第一层是用的硬件虚拟化技术，第二层就是完全软件模拟出来的，所以 VMWare 只能做两层嵌套。KVM 是将物理 CPU 的特性全部传给虚拟机，所以理论上可以嵌套 N 多层，但是事实上笔者测试的时候，跑了两层就已经非常慢了。

CentOS 7 官方宣称不正式支持 Nested 技术，所以测试的时候建议使用 Fedora 测试，笔者测试使用的是 Fedora 21。

Nested 配置方法如下。

第一步：打开 KVM 内核模块的 Nested 特性。

```
rmmode kvm-intel
modprobe kvm-intel nested=1
```

或者修改 modprobe.d，编辑 /etc/modprobe.d/kvm_mod.conf，添加以下内容：

```
options kvm-intel nested=y
```

然后使用 rmmod kvm-intel 命令删除 kvm-intel 模块，再通过 modprobe kvm-intel 命令加载 kvm-intel 模块，编辑 modprobe.d 配置文件，就不用在加载模块的时候带参数。

检查是否打开 Nested 功能，可以查看 /sys/module/kvm_intel/parameters/nested 的内容，为 Y 表示打开 Nested 特性。

```
cat /sys/module/kvm_intel/parameters/nested
Y
```

第二步：第一层的虚拟机配置文件，要将物理机 CPU 的特性全部传给虚拟机，使用 CPU HOST 技术。

```
<cpu mode='host-passthrough'/>
```

1107114476@qq.com
dolphinebook.com

第三步：和宿主机一样，将第一层虚拟机按照宿主机配置，安装相应的组件，然后就可以安装第二层的虚拟机了。

图 3-14 所示是三层 Fedora 21 嵌套的效果（fedora21 on fedaro21 kvm on fedaro21 kvm）。

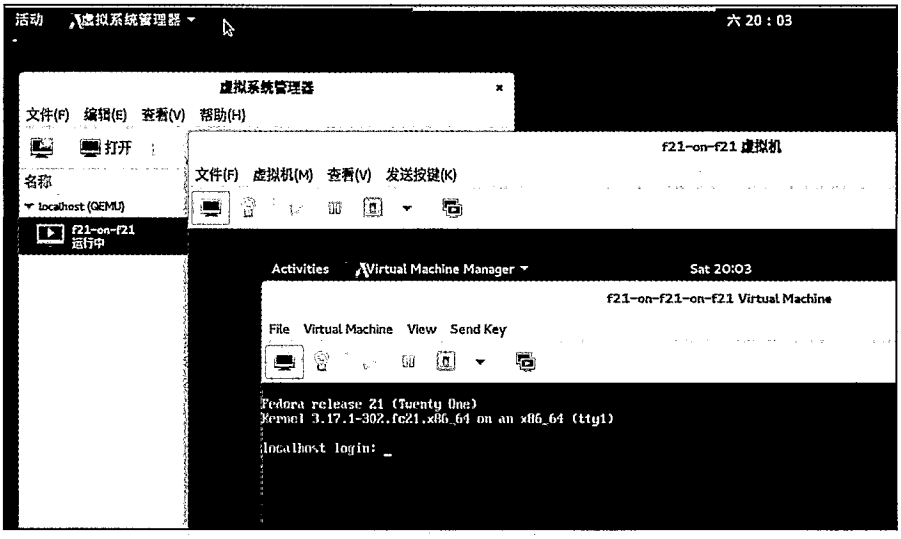


图 3-14　Fedora 21 中嵌套两层虚拟机的情况

3.5　KSM 技术与应用

1. 宿主机内存合并（压缩）

宿主机的内存压缩主要采用 KSM（Kernel SamePage Merging）技术，原理和软件压缩类似，就是将相同的内存分页进行合并。KSM 在 CentOS 6、CentOS 7 上默认是打开的，主要有两个服务：

- KSM 服务。
- ksmtuned 服务。

要关闭 KSM，关闭相关的两个服务就可以。

```
service ksm stop
service ksmtuned stop
chkconfig ksm off
chkconfig ksmtuned off
```

如果需要打开或者关闭 KSM，在生产环境中可以直接在线操作，系统会逐步合并或者释放内存页，对业务是没有影响的。所以当宿主机内存不足时，临时打开 KSM 也是一种应急方案。

1107114476@qq.com
dolphinebook.com

2. 阻止个别虚拟机进行内存压缩的方法

使用 nosharepages 关键字阻止宿主机将特定的虚拟机内存页合并，xml 配置文件如下：

```
<memoryBacking>
        <nosharepages/>
</memoryBacking>
```

3. 查看 KSM 运行的情况

在 /sys/kernel/mm/ksm/ 中可以看到 KSM 运行的情况。

- pages_shared：有多少共享内存页正在被使用。
- pages_sharing：how 有多少节点被共享并且多少被保存。
- pages_unshared：内存被合并时有多少内存页独特但是反复被检查。
- pages_volatile：多少内存页改变太快被放置。
- full_scans：多少次可以合并区域被扫描。

4. KSM 技术的应用场景

在生产环境中一般要慎用 KSM 技术，因为打开 KSM 就是为了内存超用，这在业务压力比较大，虚拟机内存变化非常频繁的时候，将导致两个结果：

- 一是会消耗一定的计算机资源用于内存扫描，加重 CPU 的消耗。
- 二是因为是内存超用，当内存不够的时候，只能是频繁地使用 swap 交互，导致虚拟机性能严重下降。

但是在测试环境和桌面虚拟化这样的场景中就很适合使用 KSM 技术，因为测试环境一般压力都不太高，能容忍虚拟机性能突然下降，打开 KSM 可以节约大量的内存，多部署一些虚拟机出来。在桌面虚拟化中，往往虚拟机的操作系统都是一样的，打开 KSM，内存压缩的效果就非常明显，只要控制好内存超用的范围就好了。

KSM 应用场景总结如下：

- 生产环境慎用。
- 测试环境推荐使用。
- 桌面虚拟机推荐使用，但是要根据实际情况控制内存超用的比率。

3.6 内存气球技术详解与应用

1. 宿主机内存气球技术配置

KVM 的内存气球技术可以在虚拟机之间按照需要调节内存大小，提高内存的利用率。

使用的时候，虚拟机需要安装 virt balloon 的驱动，内核开启 CONFIG_VIRTIO_BALLOON。CentOS 7 默认已经开启，并且默认已经安装 virt balloon 驱动。如图 3-15 所示，在虚拟机中可以看到有一个名为 Virtio memory balloon 的 PCI 设备。

1107114476@qq.com
dolphinebook.com

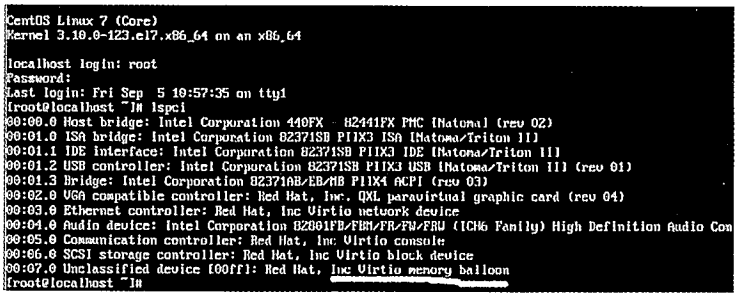


图 3-15　虚拟机的 virt balloon 设备

虚拟机 .xml 配置文件需要增加以下配置：

```
  <memballoon model='virtio'>
      <alias name='balloon0'/>
 </memballoon>
```

balloon 有两种操作。

- 膨胀：虚拟机的内存被拿掉给宿主机。
- 压缩：宿主机的内存还给虚拟机。

气球技术最大的优点是内存可以超用；缺点是有可能造成内存不够使用而影响性能。

2. 虚拟机内存气球配置

（1）Linux 系统配置

查看当前内存大小，使用 virsh qemu-monitor-command 命令。

```
virsh  qemu-monitor-command ct7 --hmp --cmd info balloon
balloon: actual=4096
```

限制内存大小为 2GB 并查看。

```
virsh  qemu-monitor-command ct7 --hmp --cmd balloon 2048
virsh  qemu-monitor-command ct7 --hmp --cmd info balloon
balloon: actual=2048
```

图 3-16 所示是在虚拟机中通过 free 命令查看内存限制效果。

图 3-16　限制内存大小为 2GB 时虚拟机内存情况

限制内存大小为 4GB 并查看。

```
virsh # qemu-monitor-command ct7 --hmp --cmd balloon 4096
virsh # qemu-monitor-command ct7 --hmp --cmd info balloon
balloon: actual=4096
```

图 3-17 所示是在虚拟机中通过 free 命令查看内存限制效果。

1107114476@qq.com
dolphinebook.com

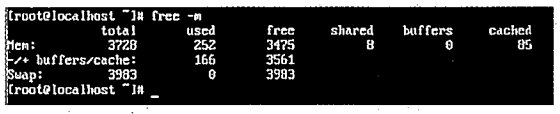


图 3-17 限制内存大小为 4GB 时虚拟机内存情况

限制内存大小为 8GB 并查看。

```
virsh # qemu-monitor-command ct7 --hmp --cmd balloon 8192
virsh # qemu-monitor-command ct7 --hmp --cmd info balloon
balloon: actual=8192
```

图 3-18 所示是在虚拟机中通过 free 命令查看内存限制效果。

图 3-18 限制内存大小为 8GB 时虚拟机内存情况

要实现虚拟机内存气球的自动平衡，可以根据自己的业务情况，编写脚本来实现。

（2）Windows 系统配置

Windows 系统的使用方法与 Linux 系统类似，但是 Windows 系统能看到整体分配的内存，然后通过内存气球技术来限制系统可以分配的内存。具体步骤如下。

第 1 步：安装 virt balloon 设备驱动，安装完成后，如图 3-19 所示，就可以看到一个 virt balloon 的 PCI 设备。

图 3-19 Windows 系统的 balloon 设备

第 2 步：安装内存气球服务，如图 3-20 所示。

图 3-20 Windows 系统 balloon 服务安装

1107114476@qq.com
dolphinebook.com

通过命令在线调整内存气球为 512MB、2GB、4GB、8GB。

```
virsh #  qemu-monitor-command 14 --hmp --cmd balloon 512
virsh #  qemu-monitor-command 14 --hmp --cmd balloon 2048
virsh #  qemu-monitor-command 14 --hmp --cmd balloon 4096
virsh #  qemu-monitor-command 14 --hmp --cmd balloon 8192
```

如图 3-21 所示，可以明显看到内存变化。

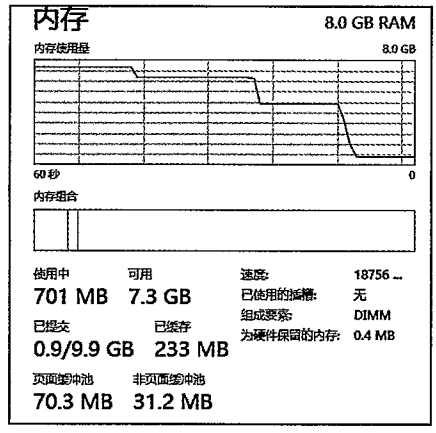


图 3-21　Windows 系统 balloon 内存变化

3. 内存气球技术应用场景

如果有两种或者几种不同业务的虚拟机在同一台宿主机上，可以考虑使用气球技术，在不同的时间段释放或者申请内存，这样可以提高内存利用率。往往要达到效果，需要编写一个脚本，通过时间或者一定的触发条件和虚拟机互动来调整内存。

3.7　内存限制技术与应用

1. 内存限制技术详解

可以将虚拟机的内存限定在一定的范围内，可以通过 virsh 命令行限制内存，也可以编辑 .xml 文件，命令行格式为：

```
virsh memtune virtual_machine --parameter size
```

可选的参数如下。

- hard_limit：虚拟机可以使用的最大内存，单位为 kibibytes（blocks of 1024 bytes）。
- soft_limit：竞争时的内存，单位为 kibibytes（blocks of 1024 bytes）。
- swap_hard_limit：最大内存加 swap，单位为 kibibytes（blocks of 1024 bytes）。
- min_guarantee：最低保证给虚拟机使用的内存，单位为 kibibytes（blocks of 1024 bytes）。

应用示例：

1）限制虚拟机 c7 最大使用 9GB 内存，写到配置文件中，下次重启虚拟机进程生效。

```
memtune c7 --hard-limit 9437184 --config
```

2）限制虚拟机 c7 竞争时为 7GB 内存。

```
memtune c7 --soft-limit 7340032 --config
```

3）限制虚拟机 c7 最大内加可以使用的宿主机 swap 不超过 10GB 内存。

```
memtune c7 --swap-hard-limit 10488320 --config
```

4）保证虚拟机 c7 最少可以使用 4GB 内存。

1107114476@qq.com
dolphinebook.com

```
memtune c7 --min_guarantee 4194304 --config
```

注意，memtune 生效方式有 3 种。

- --config：写到配置文件中，下次重启虚拟机进程生效。
- --live：影响正在运行的虚拟机，虚拟机进程停止后，效果消失，这是默认的方式。
- --current：影响停止和正在运行的虚拟机，如果虚拟机运行，虚拟机进程停止后，效果消失。

如果虚拟机未运行，写入到 .xml 配置文件中，对应的 .xml 文件如下：

```
<memory unit='KiB'>8388608</memory>
<currentMemory unit='KiB'>4194304</currentMemory>
<memtune>
    <hard_limit unit='KiB'>9437184</hard_limit>
    <soft_limit unit='KiB'>7340032</soft_limit>
    <min_guarantee unit='KiB'>4194304</min_guarantee>
    <swap_hard_limit unit='KiB'>10488320</swap_hard_limit>
</memtune>
```

还可以限制虚拟机对宿主机 swap 的使用，主要是通过 .xml 文件的 <memoryBacking> 标签来实现的。<memoryBacking> 标签包含一些元素影响宿主机对虚拟内存页的支持。

```
<memoryBacking>
        <locked/>
</memoryBacking>
```

locked 阻止宿主机将 swap 内存（交换分页内存）分配给虚拟机。设置 locked 参数，必须在 <memtune> 块中设置 hard_limit。

2. 内存限制技术的应用场景

内存限制技术可以和内存气球技术结合，将内存气球技术限制在一定范围内，避免内存被气球无限压缩。

3.8 巨型页内存技术与应用

1. 巨型页与透明巨型页

X86 默认的内存页大小是 4KB，但是也可以使用 2MB 或者 1GB 的巨型页，系统的巨型页可以传输过虚拟机，KVM 虚拟机可以通过分配巨型页提高性能。在 CentOS 5 上面，需要手工配置巨型页。在 CentOS 6 上面启用一种叫作透明巨型页的技术，默认开启巨型页，并且可以自动调整。

使用巨型页可以提升内存的分配效率，提升系统性能。巨型页可以手工配置，也可以使用透明巨型页技术。

巨型页可以手工配置的坏处：必须手工配置，虚拟机的数量、可用的内存、虚拟机的启

1107114476@qq.com
dolphinebook.com

动、关闭、迁移都需要重新配置，并且不能使用 swap。

使用透明巨页内存的好处：

- 可以使用 swap，内存页默认大小是 2MB，需要使用 swap 的时候，内存被分割为 4KB。
- 对用户透明，不需要用户做特殊配置。
- 不需要 root 权限。
- 不需要依赖某种库文件。

2. 透明巨型页内存配置

CentOS 6.x 默认启用透明巨型页内存。查看目前状态：

```
cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never
```

修改配置：

```
echo never  >/sys/kernel/mm/transparent_hugepage/enabled
```

参数说明如下：

- never：关闭，不使用透明内存。
- alway：尽量使用透明内存，扫描内存，有 512 个 4KB 页面可以整合，就整合成一个 2MB 的页面。
- madvise：避免改变内存占用。

使用情况监控：

可以查看 /sys/kernel/mm/transparent_hugepage/khugepaged 下的信息。

- pages_to_scan（默认 4096=16MB）：一个扫描周期被扫描的内存页数。
- scan_sleep_millisecs（默认 10 000=10sec）：多长时间扫描一次。
- alloc_sleep_millisecs（默认 60 000=60sec）：多长时间整理一次碎片。

也可以查看 /proc/meminfo 信息。

```
grep Huge /proc/meminfo
AnonHugePages:    266240 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

使用注意要点：虚拟机和宿主机都需要启动 THP，这样可以获得以下好处。

- 虚拟机的 CPU 可以使用 2MB 的 TLB。
- 相同的算法。
- 相同的代码。

1107114476@qq.com
dolphinebook.com

❑ 相同的内核镜像。

3. 巨型页手工配置

（1）虚拟机巨型页手工配置

有的场景需要手工配置虚拟机可以使用的巨型页数量。通过配置虚拟机 .xml 文件，可以指定虚拟机可以使用的巨型页数量。

```
<memoryBacking>
        <hugepages/>
</memoryBacking>
```

（2）巨型页使用检查

要查看当前巨型页使用情况，使用如下命令：

```
cat /proc/sys/vm/nr_hugepages
```

查看当前的巨型页值，命令如下：

```
View the current huge pages value:
# cat /proc/meminfo | grep Huge
AnonHugePages:      2048 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

（3）修改宿主机巨型页数量

巨型页默认大小是 2MB，可以通过如下命令修改可以使用的巨型页数量：

```
echo 25000 > /proc/sys/vm/nr_hugepages
```

或者使用 sysctl 命令，N 是要设置的巨型页数量：

```
sysctl vm.nr_hugepages=N
```

挂载巨型页，命令如下：

```
mount -t hugetlbfs hugetlbfs /dev/hugepages
```

重启 libvirtd 服务和虚拟机：

```
systemctl start libvirtd
virsh start virtual_machine
```

（4）关闭巨型页

关闭巨型页，命令如下：

```
sysctl vm.nr_hugepages=0
umount hugetlbfs
```

1107114476@qq.com
dolphinebook.com

4. 内存巨型页的应用场景

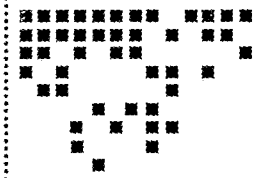
对于生产环境是 CentOS 6、CentOS 7 的宿主机，默认是打开透明巨型页并且是自动调整的，所以不需要我们做过多的设置。但是可以监控巨型页的使用分配，在内存遇到瓶颈的时候，方便查找原因。

3.9　本章小结

本章介绍了 KVM 虚拟化 CPU、内存技术，如果在生产环境中遇到 CPU 压力比较高的情况，CPU 的绑定是一个有效的技术解决手段。一般在生产环境中，内存限制技术建议慎用。目前服务器内存相对比较便宜，建议宿主机的内存尽量配置高一些。

下一章将为读者介绍 KVM 虚拟化网络技术及应用场景。

1107114476@qq.com
dolphinebook.com



Chapter 4 第 4 章

网络虚拟化技术与应用场景

本章将为读者详细介绍 KVM 虚拟化中的网络技术，以及每种技术的应用场景。如图 4-1 所示，一个完整的数据包从虚拟机到物理机的路径是：虚拟机→ QEMU 虚拟网卡→虚拟化层→内核网桥→物理网卡。

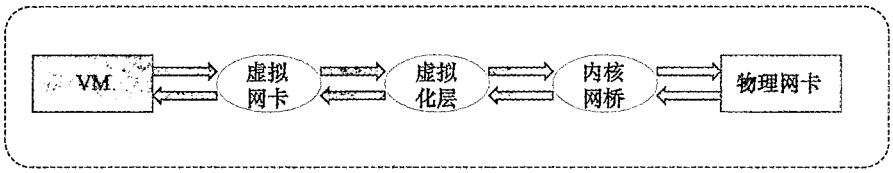


图 4-1　数据包从虚拟机到物理机的过程

KVM 的网络优化方案，总的思路就是让虚拟机访问物理网卡的层数更少，直至对物理网卡单独占领，和物理机一样使用物理网卡，以达到和物理机一样的网络性能。

本章也会介绍笔者积累的网络性能测试方法和测试数据，读者一方面可以对不同网络配置的性能差距有所认识，另一方面也可以对网络测试需要重点测试哪些指标有所了解，这些都可以应用到自己的虚拟化网络环境测试部署中。

Open vSwitch 是最近两三年虚拟化的一个热点，它主要是解决虚拟化网络的管理问题。Open vSwitch 一方面将宿主机内部的网络和物理网络打通，方便网络管理员的管理；另一方面整合了虚拟化网络管理方面的需求，实现了通过一个统一的工具管理虚拟化网络。

4.1　半虚拟化网卡技术详解

在 KVM 中，默认情况下网络设备是由 QEMU 在 Linux 的用户空间模拟出来并提供给虚

1107114476@qq.com
dolphinebook.com

拟机的。这样做的好处是通过模拟可以提供给虚拟机多种类型的网卡，提供最大的灵活性。但是由于网络 I/O 的过程需要虚拟化引擎参与，由此产生了大量的 vm exit、vm entry，效率低下。所以产生了全虚拟化网卡和半虚拟化网卡技术。全虚拟化和半虚拟化在第 1 章已经有了详细介绍，再简单回顾一下。

- 全虚拟化：即客户机操作系统完全不需要修改就能运行于虚拟机中，客户机看不到真正的硬件设备，与设备的交互全是由纯软件虚拟的。
- 半虚拟化：通过对客户机操作系统进行修改，使其意识到自己运行在虚拟机中。因此，全虚拟化与半虚拟化的根本区别在于客户机是否需要修改才能运行在宿主机中。

半虚拟化网卡与全虚拟化网卡的区别是全虚拟化网卡是虚拟化层完全模拟出来的网卡，半虚拟化网卡通过驱动对操作系统做了改造。在实际应用中，使用较多的是半虚拟化网卡技术，即 Virtio 技术。如图 4-2 所示，Virtio 驱动因为改造了虚拟机操作系统，让虚拟机可以直接和虚拟化层通信，从而大大提高了虚拟机的性能。

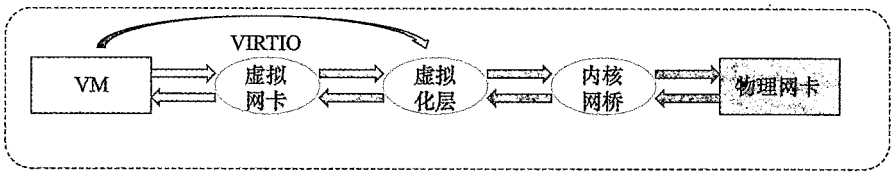


图 4-2　Virtio 网卡的原理示意图

4.1.1　半虚拟化网卡的配置

解决全虚拟化网卡效率低下的一个办法就是使用半虚拟化驱动程序 Virtio。要使用 Virtio 必须要在宿主机和客户机中分别安装 Virtio 驱动，这样客户机的 I/O 就可以以 Virtio 的标准接口来进行，而虚拟化引擎不需要捕捉这些 I/O 请求，这样就提高了性能。

1. 系统对 Virtio 的支持

Linux 内核从 2.6.24 开始支持 Virtio，因此宿主机只需要较新的 Linux 内核即可。对于虚拟机来说，Linux 发行版如 RHEL 6.x、Ubuntu 等都自带 Virtio 驱动模块。为了确定系统是否支持 Virtio，可以查看内核配置文件，示例如下，如果输出包含 Virtio 关键字，就是支持 Virtio。

```
[root@localhost ~]# grep -i Virtio /boot/config-2.6.32-358.2.1.el6.x86_64
CONFIG_NET_9P_VIRTIO=m
CONFIG_VIRTIO_BLK=m
CONFIG_SCSI_VIRTIO=m
CONFIG_VIRTIO_NET=m
CONFIG_VIRTIO_CONSOLE=m
CONFIG_HW_RANDOM_VIRTIO=m
CONFIG_VIRTIO=m
CONFIG_VIRTIO_RING=m
CONFIG_VIRTIO_PCI=m
CONFIG_VIRTIO_BALLOON=m
```

1107114476@qq.com
dolphinebook.com

Windows 虚拟机则需要额外安装 Virtio 的 Windows 版驱动，具体安装方法请参见第 16 章 Windows 镜像制作。

2. 配置半虚拟化网卡

配置半虚拟化网卡的方法有两种。

1）在虚拟机启动命令中加入 virtio-net-pci 参数。

```
[root@localhost ~]# QEMU-system-x86_64 -boot c -drive \
file=/images/xpbase.qcow2,if=virtio -m 384 -netdev \
type=tap,script=/etc/KVM/QEMU-ifup,id=net0 -device \
virtio-net-pci,netdev=net0
```

2）使用 Libvirt 管理的 KVM 虚拟机，可以修改 xml 配置文件，配置文件如下：

```
<interface type='bridge'>
    <mac address='fa:16:3e:fc:e0:c0'/>
    <source bridge='br0'/>
    <model type='Virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

<model type='Virtio'/> 指定了使用 Virtio 网卡。如果要使用全虚拟化网卡，使用 <model type='e1000'/>。如果没有指定 model type 关键字，默认是 8139 的全虚拟化网卡。e1000 模拟的是 Intel 公司的千兆网卡，8139 模拟的是早期的 Realtek 公司的百兆网卡。

由于实际中多数使用 Libvirt 管理虚拟机，因此第二种方法用得更多。

4.1.2 全虚拟化网卡、半虚拟化网卡性能比较

全虚拟化网卡的性能与 Virtio 半虚拟化网卡的性能差距非常明显。通常用两个方面的数据来衡量网卡性能：

- 吞吐率（即带宽）bit/s。
- 每秒发包数 pps。

通常情况下，更多地使用前者来衡量网卡性能，但是实际使用中经常发现，如果服务器承载的是游戏或者 Web 服务等需要大量收发小包的业务，即使带宽没有被撑满也会出现性能瓶颈，这时就是受到了“每秒发包数”这个指标的限制。因此下面将从带宽和每秒发包数两方面对比全虚拟化网卡和半虚拟化网卡的性能。测试使用的是 Intel X540 的万兆网卡。

1. 测试环境介绍

测试环境使用两台服务器，一台宿主机，一台物理机，宿主机上运行虚拟机，用物理机给虚拟机打压力。

宿主机和物理机配置一模一样，配置信息如下。

- CPU：CPU 1 Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz 2 颗。

1107114476@qq.com
dolphinebook.com

- 内存：DIMM000 DIMM000，Hynix，8192 MB，1332 MHz 6 根。
- 网卡：Intel Corporation Ethernet Controller 10-Gigabit X540-AT2。

2. 测试环境拓扑

测试环境如图 4-3 所示，宿主机编号为 A，物理机编号为 B，全部虚拟机使用桥接模式。

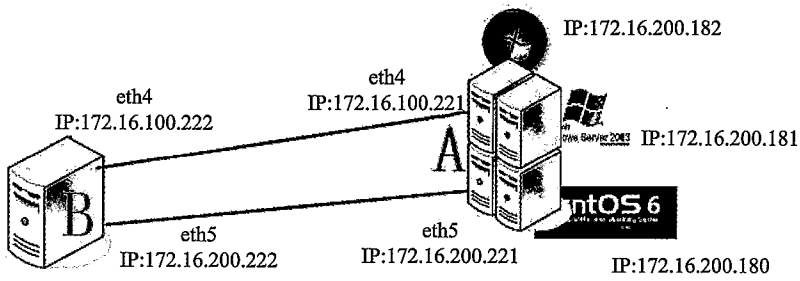


图 4-3　网卡性能测试环境拓扑图

（1）宿主机 A 配置及虚拟机信息

宿主机 A 网卡 eth4 配置 IP：172.16.100.222；网卡 eth5 和网桥 br5 对应，配置 IP：172.16.200.222。

在宿主机 A 上创建 3 台虚拟机的配置如下。

1）CentOS6.2。

- IP：172.16.200.180。
- 内存：4GB。
- 网卡驱动：Virtio。
- 网卡连接方式：桥接。

2）Windows 2003。

- IP：172.16.200.181。
- 内存：4GB。
- 网卡驱动：Virtio。
- 网卡连接方式：桥接。

3）Windows 2008。

- IP：172.16.200.182。
- 内存：4GB。
- 网卡驱动：Virtio。
- 网卡连接方式：桥接。

（2）服务器 B 配置

网卡 eth4 配置 IP：172.16.100.221；网卡 eth5 配置 IP：172.16.200.221。

1107114476@qq.com
dolphinebook.com

3. 测试用例一：物理网卡之间发包率及吞吐测试

提示 以下测试内容和结果来自笔者实际的测试报告，内容比较详细，主要目的是希望能够介绍一下进行网络性能测试的时候应该关注的测试指标及测试内容、笔者的数据展现方式，以及各种驱动、系统下虚拟机网卡性能的差异。

（1）服务器 B 网卡 eth4 到服务器 A 网卡 eth4 发包率测试

测试内容如下（具体的测试工具使用 FIO，使用方法见本书第 15 章）：

- 64B、128B、256B、512B、1024B 小包，发包率测试。
- 测试队列总数 50 个，取稳定状态下的最大值为测试结果。

发包率测试结果如图 4-4 所示。

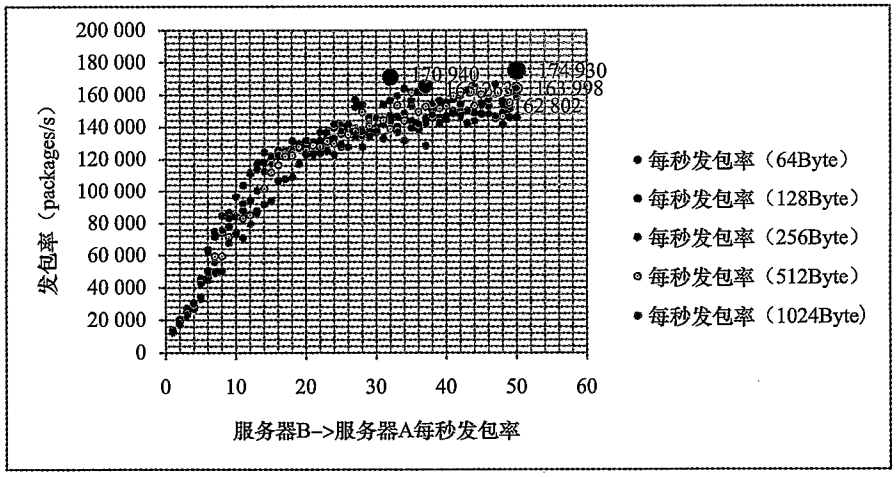


图 4-4 物理网卡发包率测试

数据结果图表如表 4-1 所示。

表 4-1 物理网卡发包率测试结果

测试内容	Max package	测试内容	Max package
64Byte	174 930	512Byte	163 998
128Byte	170 940	1024Byte	162 802
256Byte	165 263		

（2）服务器 B 网卡 eth4 到服务器 A 网卡 eth4 吞吐测试

测试内容如下：

- 16KB、64KB、128KB、256KB、512KB、1MB、5MB 包，发包率测试。
- 测试队列总数 20 个，取稳定状态下的最大值为测试结果。

1107114476@qq.com
dolphinebook.com

吞吐测试结果如图 4-5 所示。

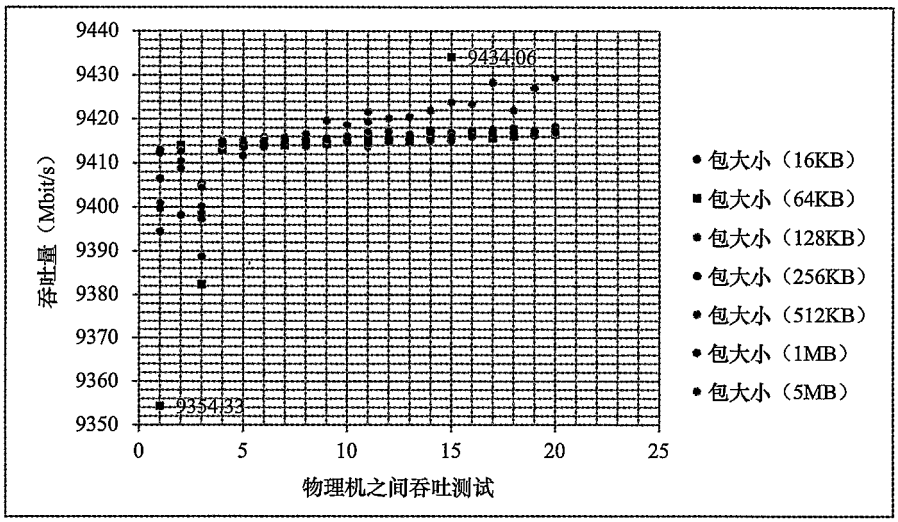


图 4-5　物理网卡吞吐测试

数据结果图表如表 4-2 所示。

表 4-2　物理网卡吞吐测试结果

测试内容	最大吞吐	测试内容	最大吞吐
16KB	9417.55	512KB	9417.94
64KB	9434.06	1MB	9419.34
128KB	9417.07	5MB	9429.38
256KB	9418.22		

4. 测试用例二：物理网卡到 CentOS 6.2 虚拟机之间发包率及吞吐测试

（1）服务器 B 网卡 eth5 到 CentOS 6.2 虚拟机网卡发包率测试

测试内容如下：

- 64B、128B、256B、512B、1024B 小包，发包率测试。
- 测试队列总数 50 个，取稳定状态下的最大值为测试结果。

测试结果如图 4-6 所示。

测试结果图表如表 4-3 所示。

表 4-3　物理网卡到 CentOS 6.2 虚拟机之间发包率测试结果

测试内容	Max package	测试内容	Max package
64Byte	87 551	512Byte	88 319
128Byte	87 992	1024Byte	88 049
256Byte	86 234		

1107114476@qq.com
dolphinebook.com

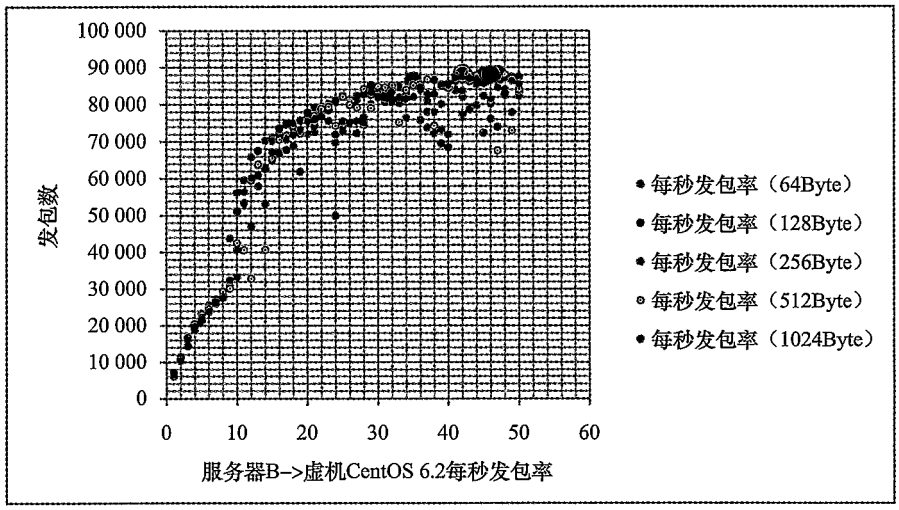


图 4-6 物理网卡到 CentOS 6.2 虚拟机之间发包率测试结果

（2）服务器 B 网卡 eth4 到虚拟机 CentOS 6.2 网卡吞吐测试

测试内容如下：

- 16KB、64KB、128KB、256KB、512KB、1MB、5MB 包，发包率测试。
- 测试队列总数 20 个，取稳定状态下的最大值为测试结果。

吞吐测试结果如图 4-7 所示。

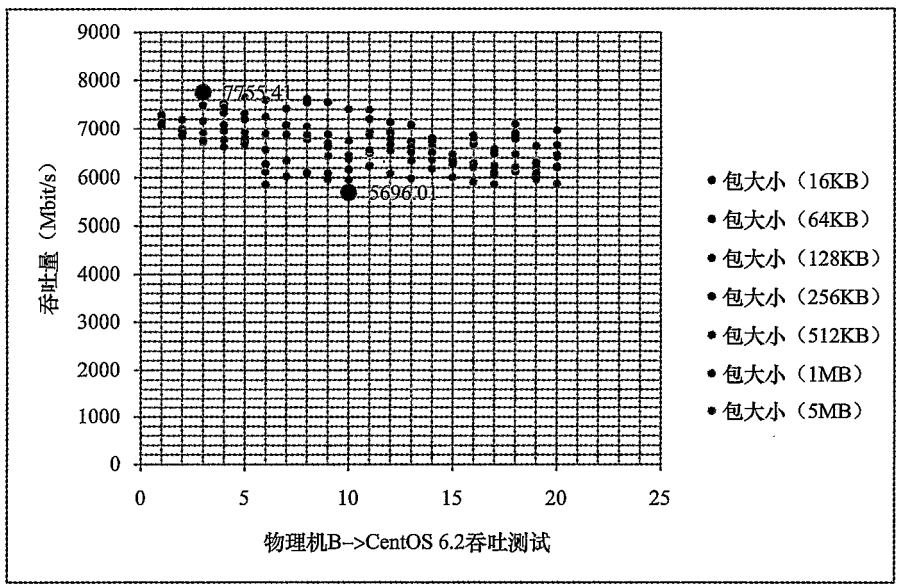


图 4-7 物理网卡到 CentOS 6.2 虚拟机之间吞吐测试

测试结果图表如表 4-4 所示。

1107114476@qq.com
dolphinebook.com

表 4-4　物理网卡到 CentOS 6.2 虚拟机之间吞吐测试结果

测试内容	最大吞吐	测试内容	最大吞吐
16KB	7600.7	512KB	7755.41
64KB	7553.18	1MB	7403.53
128KB	7656.1	5MB	7458.82
256KB	7616.8		

5. 测试用例三：物理网卡到 Windows 2003 虚拟机之间发包率及吞吐测试

（1）虚拟机网卡驱动：Virtio

1）服务器 B 网卡 eth5 到虚拟机 Windows 2003 网卡发包率测试。

测试内容如下：

- 64B、128B、256B、512B、1024B 小包，发包率测试。
- 测试队列总数 50 个，取稳定状态下的最大值为测试结果。

测试结果如图 4-8 所示。

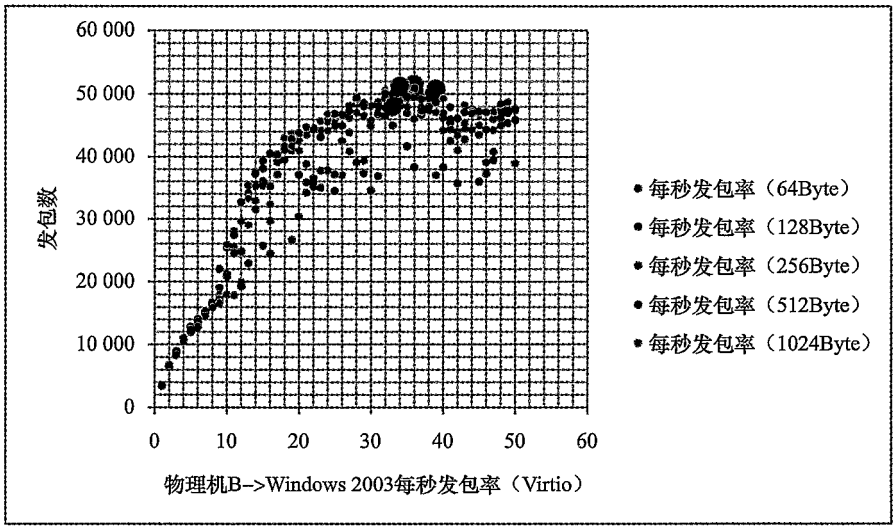


图 4-8　Windows 2003 Virtio 网卡发包率测试

测试结果图表如表 4-5 所示。

表 4-5　Windows 2003 Virtio 网卡发包率测试结果

测试内容	Max package	测试内容	Max package
64Byte	50 432	512Byte	51 273
128Byte	50 824	1024Byte	48 056
256Byte	51 537		

1107114476@qq.com
dolphinebook.com

2）服务器 B 网卡 eth4 到虚拟机 Windows 2003 网卡吞吐测试。

测试内容如下：

- 16KB、64KB、128KB、256KB、512KB、1MB、5MB 包，发包率测试。
- 测试队列总数 20 个，取稳定状态下的最大值为测试结果。

吞吐测试结果如图 4-9 所示。

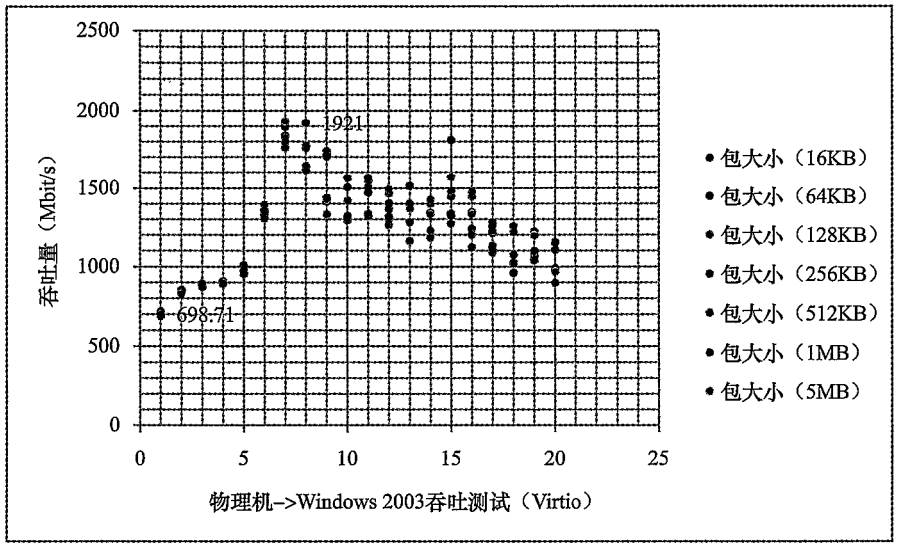


图 4-9 Windows 2003 Virtio 网卡吞吐测试

测试结果图表如表 4-6 所示。

表 4-6 Windows 2003 Virtio 网卡吞吐测试结果

测试内容	最大吞吐	测试内容	最大吞吐
16KB	1909.08	512KB	1824.59
64KB	1921	1MB	1795.09
128KB	1930.17	5MB	1763.11
256KB	1839.94		

（2）虚拟机网卡驱动：e1000

1）服务器 B 网卡 eth5 到虚拟机 Windows 2003 网卡发包率测试。

测试内容如下：

- 64B、128B、256B、512B、1024B 小包，发包率测试。
- 测试队列总数 30 个，取稳定状态下的最大值为测试结果。

测试结果如图 4-10 所示。

1107114476@qq.com
dolphinebook.com

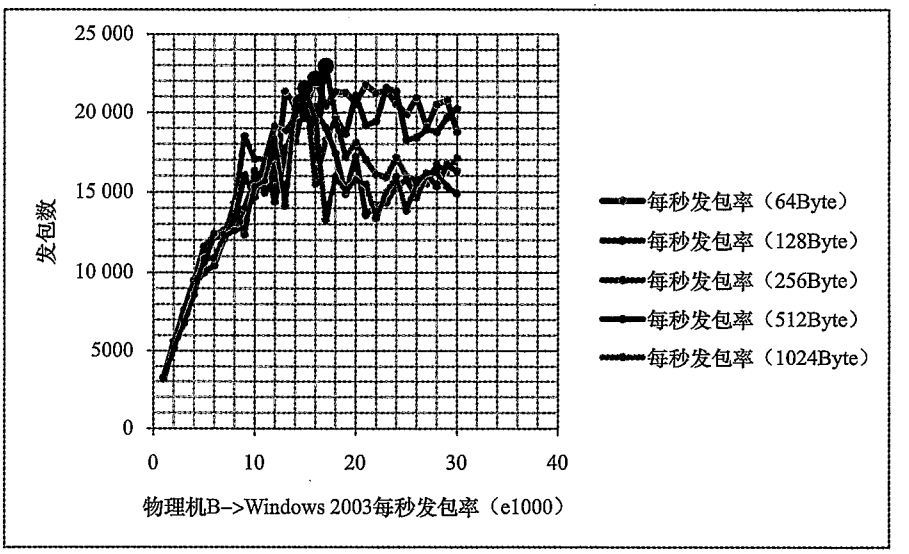


图 4-10　Windows 2003 e1000 网卡发包率测试

测试结果数据图表如表 4-7 所示。

表 4-7　Windows 2003 e1000 网卡发包率测试

测试内容	Max package	测试内容	Max package
64Byte	22 148	512Byte	20 063
128Byte	22 944	1024Byte	21 477
256Byte	21 547		

2）服务器 B 网卡 eth4 到虚拟机 Windows 2003 吞吐测试。

测试内容如下：

- 16KB、64KB、128KB、256KB、512KB、1MB、5MB 包，发包率测试。
- 测试队列总数 20 个，取稳定状态下的最大值为测试结果。

吞吐测试结果如图 4-11 所示。

测试结果图表如表 4-8 所示。

表 4-8　Windows 2003 e1000 网卡吞吐测试结果

测试内容	最大吞吐	测试内容	最大吞吐
16KB	2898.93	512KB	2715.63
64KB	3259.28	1MB	2109.3
128KB	2698.52	5MB	1770.69
256KB	2966.67		

1107114476@qq.com
dolphinebook.com

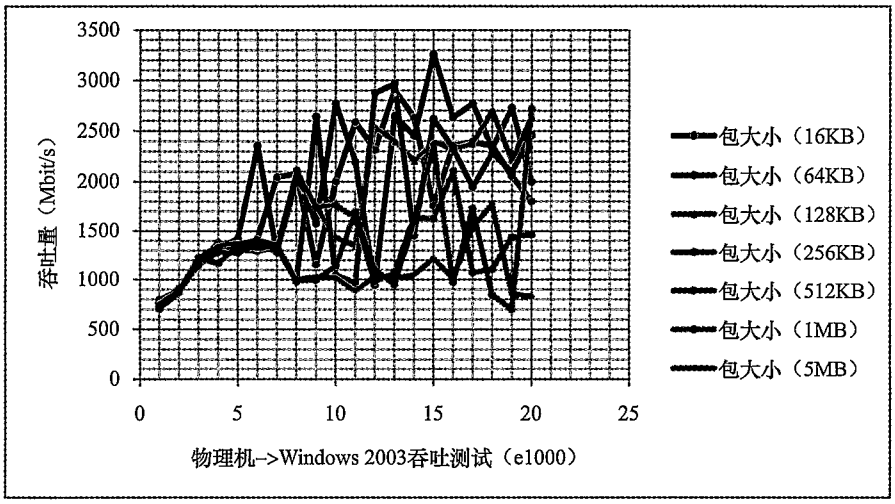


图 4-11 Windows 2003 e1000 网卡吞吐测试

6. 测试用例四：物理网卡到 Windows 2008 虚拟机之间发包率及吞吐测试

（1）虚拟机网卡驱动：virtio

1）服务器 B 网卡 eth5 到虚拟机 Windows 2008 网卡发包率测试。

测试内容如下：

- 64B、128B、256B、512B、1024B 小包，发包率测试。
- 测试队列总数 50 个，取稳定状态下的最大值为测试结果。

测试结果如图 4-12 所示。

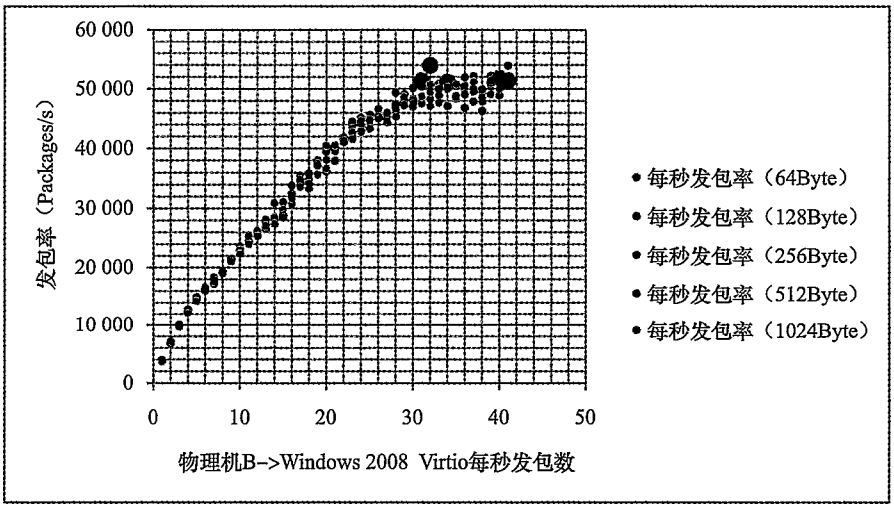


图 4-12 eth5 到虚拟机 Windows 2008 网卡发包率测试

测试结果图表如表 4-9 所示。

1107114476@qq.com
dolphinebook.com

表 4-9　eth5 到虚拟机 Windows 2008 网卡发包率测试结果

测试内容	Max package	测试内容	Max package
64Byte	51 365	512Byte	51 134
128Byte	53 889	1024Byte	51 345
256Byte	51 781		

2）服务器 B 网卡 eth4 到虚拟机 Windows 2008 网卡吞吐测试。

测试内容如下：

- 16KB、64KB、128KB、256KB、512KB、1MB、5MB 包，发包率测试。
- 测试队列总数 20 个，取稳定状态下的最大值为测试结果。

吞吐测试结果如图 4-13 所示。

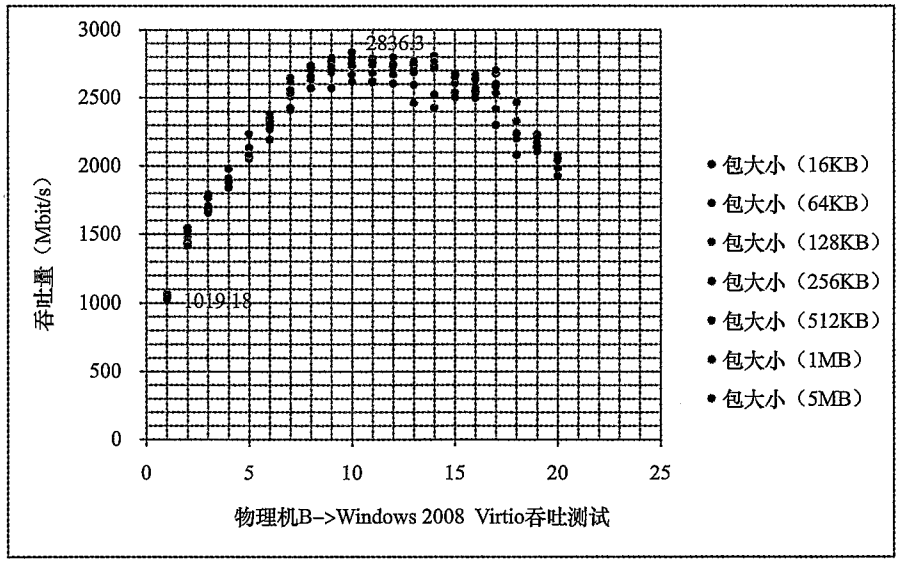


图 4-13　eth4 到虚拟机 Windows 2008 网卡吞吐测试

测试结果图表如表 4-10 所示。

表 4-10　eth4 到虚拟机 Windows 2008 网卡吞吐测试结果

测试内容	最大吞吐	测试内容	最大吞吐
16KB	2836	512KB	2797
64KB	2807	1MB	2741
128KB	2757	5MB	2769
256KB	2746		

（2）虚拟机网卡驱动：e1000

1）服务器 B 网卡 eth5 到虚拟机 Windows 2008 网卡发包率测试。

1107114476@qq.com
dolphinebook.com

测试内容如下：

❑ 64B、128B、256B、512B、1024B 的小包，发包率测试。

❑ 测试队列总数 30 个，取稳定状态下的最大值为测试结果。

测试结果如图 4-14 所示。

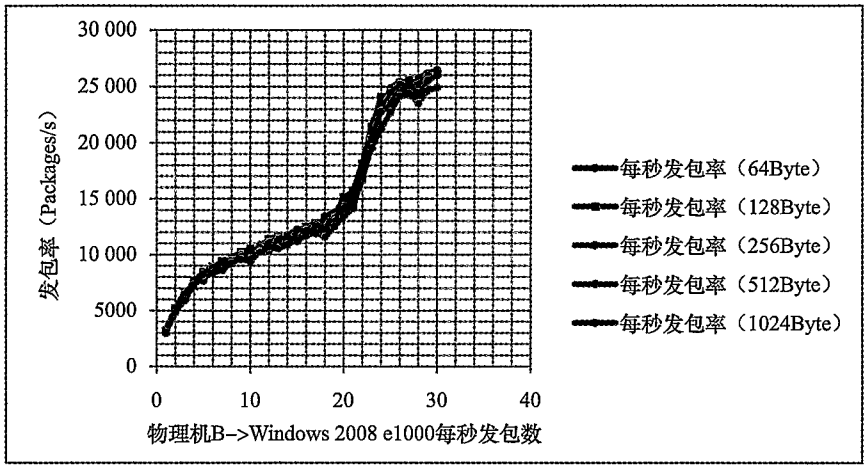


图 4-14 Windows 2008 e1000 网卡发包率测试

测试结果图表如表 4-11 所示。

表 4-11 Windows 2008 e1000 网卡发包率测试结果

测试内容	Max package	测试内容	Max package
64Byte	26 468	512Byte	26 010
128Byte	26 179	1024Byte	24 911
256Byte	26 029		

2）服务器 B 网卡 eth4 到虚拟机 Windows 2008 网卡吞吐测试。

测试内容如下：

❑ 16KB、64KB、128KB、256KB、512KB、1MB、5MB 包，发包率测试。

❑ 测试队列总数 20 个，取稳定状态下的最大值为测试结果。

吞吐测试结果如图 4-15 所示。

测试结果图表如表 4-12 所示。

表 4-12 Windows 2008 e1000 网卡吞吐测试结果

测试内容	最大吞吐	测试内容	最大吞吐
16KB	763.97	512KB	816.62
64KB	783.22	1MB	785.23
128KB	806.11	5MB	798.58
256KB	812.66		

1107114476@qq.com
dolphinebook.com

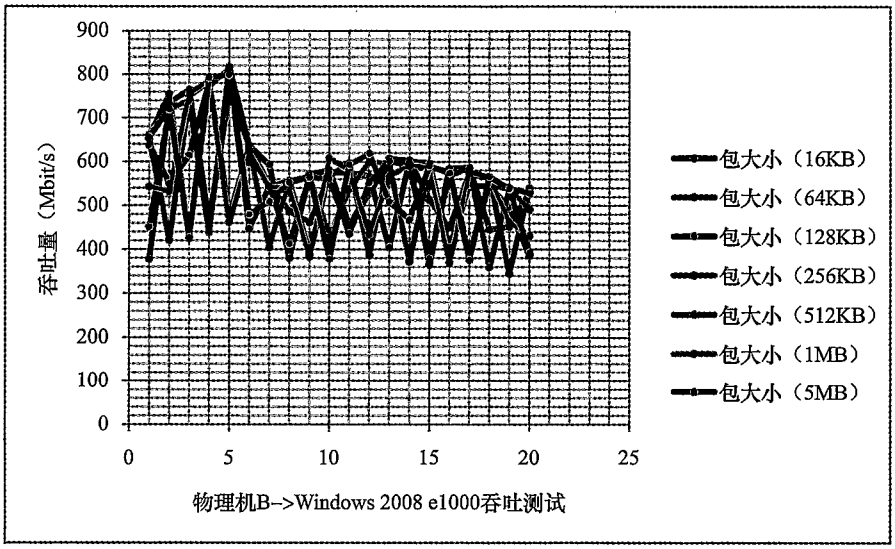


图 4-15　Windows 2008 e1000 网卡吞吐测试

7. 测试总结

（1）发包率性能测试数据及结论（见图 4-16 和表 4-13）

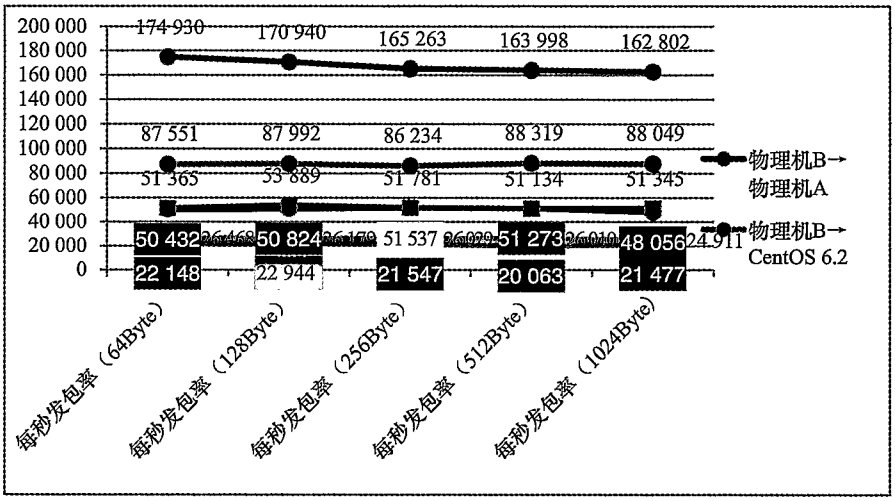


图 4-16　发包率性能测试数据

表 4-13　X540 发包率性能测试数据对比

测试类型	包大小（64Byte）	包大小（128Byte）	包大小（256Byte）	包大小（512Byte）	包大小（1024Byte）
物理机 B–> 物理机 A	174 930	170 940	165 263	163 998	162 802
物理机 B–>CentOS 6.2	87 551	87 992	86 234	88 319	88 049

1107114476@qq.com
dolphinebook.com

（续）

测试类型	包大小（64Byte）	包大小（128Byte）	包大小（256Byte）	包大小（512Byte）	包大小（1024Byte）
物理机 B->Windows 2003（Virtio）	50 432	50 824	51 537	51 273	48 056
物理机 B->Windows 2003（e1000）	22 148	22 944	21 547	20 063	21 477
物理机 B->Windows 2008（Virtio）	51 365	53 889	51 781	51 134	51 345
物理机 B->Windows 2008（e1000）	26 468	26 179	26 029	26 010	24 911

Intel X540 在虚拟机以桥的方式连接网络的情况下性能表现如下：

- CentOS 6.2 的发包率性能可以达到物理网络的 50% 左右。
- 虚拟机使用 Virtio 驱动，对于发包率性能有明显提高，基本是使用 e1000 驱动性能的 2 倍。
- 整体可以看到，在 KVM 虚拟化下，对于 CentOS 系统要比 Windows 的好 3 倍。

（2）吞吐性能测试数据及结论（见图 4-17 和表 4-14）

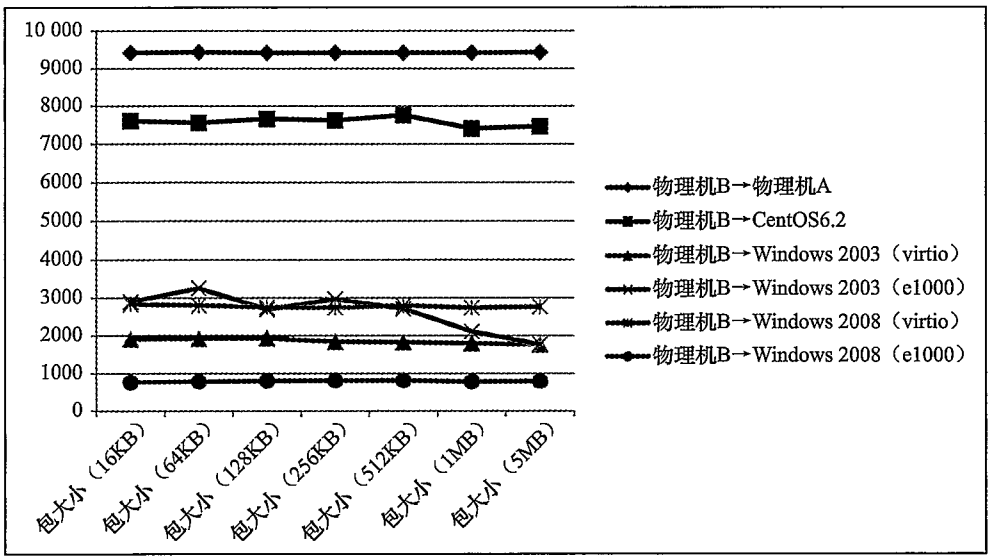


图 4-17 吞吐性能测试数据

表 4-14 吞吐性能测试数据对比

测试类型	包大小（16KB）	包大小（64KB）	包大小（128KB）	包大小（256KB）	包大小（512KB）	包大小（1MB）	包大小（5MB）
物理机 B-> 物理机 A	9418	9434	9417	9418	9418	9419.34	9429.38
物理机 B->CentOS 6.2	7600.7	7553.18	7656.1	7616.8	7755.41	7403.53	7458.82
物理机 B->Windows 2003（Virtio）	1909.08	1921	1930.17	1839.94	1824.59	1795.09	1763.11
物理机 B->Windows 2003（e1000）	2898.93	3259.28	2698.52	2966.67	2715.63	2109.3	1770.69
物理机 B->Windows 2008（Virtio）	2836	2807	2757	2746	2797	2741	2769
物理机 B->Windows 2008（e1000）	763.97	783.22	806.11	812.66	816.62	785.23	798.58

1107114476@qq.com
dolphinebook.com

Intel X540 在虚拟机使用桥方式连接的时候，吞吐性能表现如下：

- 物理网络的性能非常出色，几乎达万兆。
- CentOS 6.2 的性能可能达到物理网卡 80% 左右的吞吐性。
- Windows 2003 使用 e1000 时，吞吐要比使用 Virtio 性能高 10%。
- Windows 2008 使用 Virtio 驱动时，吞吐要比使用 e1000 驱动的性能高两倍左右。
- 由上面两条可以看出，Windows 系统吞吐性能和驱动是有很大关系的，不同驱动表现的性能差异十分明显。

4.1.3　全虚拟化网卡、半虚拟化网卡的应用场景

在实际生产环境中，根据系统不同，甚至系统的版本不同，网卡的选择方式也不一样。

1. Windows 虚拟机网卡使用

2014 年 10 月之前，笔者在生产环境中，Windows 虚拟机一直使用 e1000 的全虚拟化网卡，因为 Windows 虚拟机使用 Virtio 网卡，一直有网络闪断的情况发生。Windows 虚拟机 Virtio 驱动也在不断地改进，到 2014 年 10 月，笔者测试最新版的 Virtio 的 Windows 驱动，已经工作得比较稳定了，所以笔者随后又在生产环境使用 Virtio。如果 Windows 虚拟机的压力很高，本书 4.5 节介绍的网卡独占技术也是非常不错的性能解决方案，稳定性也非常好。

2. Linux 虚拟机网卡使用

Viritio 网卡在 Linux 系统上表现一直很稳定，目前 Linux 内核默认集成 Virtio 驱动，大部分 Linux 发行版可以直接使用 Virtio 网卡。比如从 RHEL 5.4 开始就正式支持 Virtio，RHEL 5.4 之后的 RHEL 系统都建议使用 Virtio 网卡。对于一些比较老的但是还在用的 Linux 系统，比如 RHEL 4.x，为了保证兼容性，建议使用 e1000 的虚拟网卡。

4.2　MacVTap 和 vhost-net 技术原理与应用

图 4-18 所示是 MacVTap 与 vhost_net 技术原理比较。

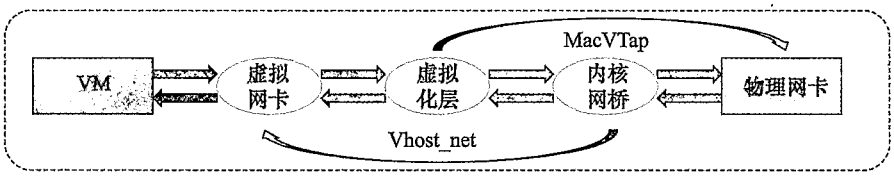


图 4-18　MacVTap 与 vhost_net 技术原理比较

如图 4-18 所示，vhost_net 技术使虚拟机的网络通信绕过用户空间的虚拟化层，可以直接和内核通信，从而提供虚拟机的网络性能，MacVTap 则是跳过内核的网桥。使用 vhost_net，必须使用 Virtio 半虚拟化网卡。

1107114476@qq.com
dolphinebook.com

MacVlan 的功能是给同一个物理网卡配置多个 MAC 地址，这样就可以在软件上配置多个以太网口，属于物理层的功能。MacVTap 是用来替代 TUN/TAP 和 Bridge 内核模块的。MacVTap 是基于 MacVlan 这个模块，提供 TUN/TAP 中 TAP 设备使用的接口，使用 MacVTap 以太网口的虚拟机能够通过 TAP 设备接口，直接将数据传递到内核中对应的 MacVTap 以太网口。

vhost_net 是对于 Virtio 的优化。Virtio 本来是设计用于进行客户系统的前端与 VMM 的后端通信，减少硬件虚拟化方式下根模式和非根模式的切换。vhost-net 是对于 Virtio 的优化，Virtio 是虚拟化层的前端优化方案，减少硬件虚拟化方式下根模式与非根模式的切换，而 vhost-net 是虚拟化层后端优化方案。不使用 vhost-net，进入 CPU 的根模式后，需要进入用户态将数据发送到 tap 设备后，再次切入内核态，使用 vhost-net 方式后，进入内核态后不需要在进行内核态用户态的切换，进一步减少特权级切换的开销。

4.2.1 MacVTap 技术与应用

MacVTap 同传统的桥接方案相比，其优点主要是支持新的虚拟化网络技术，可以将宿主机 CPU 的一些工作交给网络设备。

1. 传统桥接方案

传统的 Linux 网络虚拟化技术采用的是 TAP+Bridge 方式，将虚拟机连接到虚拟的 TAP 网卡，然后将 TAP 网卡加入到 Bridge。Bridge 相当于用软件实现的交换机。这种解决方案实际上就是用服务器的 CPU 通过软件模拟网络。

下面是在 4.1 节出现过的虚拟机网卡配置文件。

```
<interface type='bridge'>
     <mac address='00:16:3e:d6:6f:37'/>
     <source bridge='br0'/>
     <targetdev='tap0'/>
     <model type='Virtio'/>
     <alias name='net0'/>
     <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
 </interface>
```

其中 <target dev='tap0'/> 的 tap0 就是一个 TAP 设备，只不过其中的一些参数已经换成了具体的设置。可以通过 ifconfig 查看 tap0 的情况。

```
root@localhost:~# ifconfig tap0
tap0  Link encap:EthernetHWaddr 52:13:ab:00:85:83
inet6 addr: fe80::5013:abff:fe00:8583/64 Scope:Link
UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
RX packets:11 errors:0 dropped:0 overruns:0 frame:0
TX packets:27 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:846 (846.0 B) TX bytes:2050 (2.0 KB)
```

1107114476@qq.com
dolphinebook.com

tap0 被加入到了 br0 中，可以理解为 tap0 是交换机 br0 上的一个端口，虚拟机的网卡则连接到了这个端口。

```
[root@localhost ~]# brctl show br0
bridge name     bridge id               STP enabled     interfaces
br0             8000.000c2962b4f0       no              tap0
```

以上就是传统的 TAP+Bridge 虚拟化网络技术，这种技术有 3 个缺点：

- 每台宿主机内都存在 Bridge，会使网络拓扑变复杂，相当于增加了交换机的级联层数。
- 同一宿主机上的虚拟机之间的流量直接在 Bridge 完成交换，使流量监控、监管困难。
- bridge 是软件实现的二层交换技术，会加大服务器的负担。

2. MacVTap 方案

针对云计算中的复杂网络问题，业界主要提出了两种扩展技术标准：802.1Qbg 与 802.1Qbh。802.1Qbh（Bridge Port Extension）主要由 VMware 与 Cisco 提出，尝试从接入层到汇聚层提供一个完整的虚拟化网络解决方案，尽可能达到软件定义一个可控网络的目的。它扩展了传统的网络协议，因此需要新的网络设备支持，成本较高。802.1Qbg（Edge Virtual Bridging，EVB）主要由 HP 等公司联合提出，尝试以较低成本利用现有设备改进软件模拟的网络。

802.1Qbg 的一个核心概念是 VEPA（Virtual Ethernet Port Aggregator），简单来说它通过端口汇聚和数据分类转发，把宿主机上原来由 CPU 和软件来做的网络处理工作转移到接入层交换机上，减少宿主机 CPU 负载，同时使得在一级的交换机上做虚拟机网络流量监控成为可能，从而更清晰地分割服务器与网络设备的工作范围，方便系统的管理。

为支持新的虚拟化网络技术，Linux 引入了新的网络设备模型：MacVTap。MacVTap 的实现基于传统的 MacVlan。和 TAP 设备一样，每一台 MacVTap 设备拥有一台对应的 Linux 字符设备，并拥有和 TAP 设备一样的 IOCTL 接口，因此能直接被 KVM/QEMU 使用，方便地完成网络数据交换工作。引入 MacVTap 设备的目标是：简化虚拟化环境中的交换网络，代替传统的 Linux TAP 设备加 Bridge 设备组合，同时支持新的虚拟化网络技术，如 802.1 Qbg。

MacVTap 设备有 3 种不同的工作模式：VEPA（Virtual Ethernet Port Aggregator）、Bridge、Private。

（1）VEPA

在这种模式下，同一物理网卡下的 MacVTap 设备之间的流量也要发送到外部交换机，再由外部交换机转发回到服务器，前提是交换机必须支持 hairpin 模式。

（2）Bridge

这种模式类似于传统的 Linux Bridge，同一物理网卡下的 MacVTap 设备可以直接进行以太网帧的交换，不需要外部交换机的介入。

（3）Private

在 Private 模式下，同一物理网卡下的 MacVTap 设备互相无法连通，无论外部交换机支不支持 hairpin 模式。

以下是创建一个 MacVTap 端口的过程。

1107114476@qq.com
dolphinebook.com

```
[root@localhost ~]# ip link add link eth1 name MacVTap0 type MacVTap
[root@localhost ~]# ip link set MacVTap0 address 1a:46:0b:ca:bc:7b up
[root@localhost ~]# ip link show MacVTap0
12: MacVTap0@eth1: <BROADCAST,MULTICAST,UP>mtu 1500 qdiscnoqueue state UNKNOWN
link/ether 1a:46:0b:ca:bc:7b brdff:ff:ff:ff:ff:ff
```

KVM 虚拟机使用 MacVTap 接入网络的方法也很简单，在 Libvirt 中使用 MacVTap 的配置文件如下：

```
<interface type='direct'>
 <mac address='1a:46:0b:ca:bc:7b'/>
 <source dev='eth1' mode='bridge'/>
 <model type='e1000'/>
 <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

虚拟机创建后，宿主机系统会自动创建出一台 MacVTap 设备给虚拟机使用，这台 MacVTap 设备附属于母设备 eth1，工作模式为 Bridge。

目前 MacVTap 在 Windows 虚拟机中性能表现不理想，因而 Windows 虚拟机不建议使用。

4.2.2 vhost_net 技术

1. 什么是 vhost_net

运行一台虚拟机是由用户空间的 QEMU 和内核的 KVM 共同完成的，QEMU 负责模拟各种设备提供给虚拟机，KVM 负责完成 CPU 和内存的虚拟化。Virtio 的后端处理程序一般是由用户空间的 QEMU 提供的。为了进一步减少延迟、提高性能，比较新的内核中增加了一个 vhost_net 的驱动模块，在内核中实现了 Virtio 的后端处理程序。

2. vhost_net 配置

Libvirt 配置文件的默认配置就是使用 vhost_net，当然也可以显式配置。

```
<interface type='bridge'>
  <mac address=''/>
  <source bridge='br0'/>
  <model type='Virtio'/>
  <driver name="vhost"/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

4.3 网卡的中断与多队列

在介绍网卡中断与多队列之前，需要先了解一下中断。X86 体系结构的计算机采用中断

1107114476@qq.com
dolphinebook.com

机制来协同处理器与其他设备的工作。当一台设备需要与处理器通信时，就会向处理器发出一个中断信号。例如敲击键盘时，键盘就会产生一个中断，通知操作系统有键被按下。在响应一个中断时，Linux 内核会调用一个叫作中断处理程序的函数来处理这个中断。由于中断会频繁发生，因此要求中断处理程序执行要快速。为了实现快速执行，必须要将一些繁重且不非常紧急的任务从中断处理程序中剥离出来，这一部分在 Linux 中称为“下半部”，有 3 种方法处理下半部——软中断、tasklet 和工作队列。

以网卡为例，当网卡收到数据包时会产生中断，通知内核有新数据包，然后内核调用中断处理程序进行响应，把数据包从网卡缓存复制到内存。因为网卡缓存大小有限，如果不及时复制出数据，后续数据包将会因为缓存溢出被丢弃，因此这一工作需要立即完成。剩下的处理和操作数据包的工作就会交给软中断。

高负载的网卡是软中断产生的大户，很容易形成瓶颈。并且在相当长的时期内，网卡的中断都是通过 CPU 0 来处理的，造成 CPU 0 的压力很高、其他 CPU 相对空闲的情况。后来出现了网卡多队列技术，网卡多队列实际就是网卡的数据请求可以通过多个 CPU 处理。

1. 物理网卡的中断与多队列

RSS 是网卡的硬件特性，实现多队列，将不同的流分发到不同的 CPU 上，同一数据流始终在同一 CPU 上，避免 TCP 的顺序性和 CPU 的并行性发生冲突。基于流的负载均衡，解决了顺序协议和 CPU 并行的冲突及 Cache 热度问题。

如果服务器的网卡支持 RSS，会在系统中看到网卡对应多个发送和接收队列。

```
[root@localhost ~]# ls /sys/class/net/eth0/queues/
rx-0  rx-1  rx-2  rx-3  rx-4  rx-5  rx-6  rx-7  tx-0  tx-1  tx-2  tx-3  tx-4
tx-5  tx-6  tx-7
```

多队列网卡的好处是可以将每个队列产生的中断分布到 CPU 的多个核，实现负载均衡，避免单个核被占用过高、而其他核还处于空闲的情况。多队列网卡的每个队列的中断都是使用单独的中断线。

2. 绑定中断

CentOS 系统中依靠 irqbalance 服务优化中断分配，irqbalance 服务用于优化中断分配，它会自动收集系统数据以分析使用模式，并依据系统负载状况将工作状态置于 Performance mode 或 Power-save mode。

- 处于 Performance mode 时，irqbalance 会将中断尽可能均匀地分发给各个 CPU core，以充分利用 CPU 多核，提升性能。
- 处于 Power-save mode 时，irqbalance 会将中断集中分配给第一个 CPU，以保证其他空闲 CPU 的睡眠时间，降低能耗。

irqbalance 服务在大压力情况下，尤其是万兆网卡上，有中断漂移及分配不平均的现象。如果压力比较大，可以手工调整系统的网卡中断。

1107114476@qq.com
dolphinebook.com

github 上有一个非常经典的网卡绑定中断的脚本，脚本的主要功能是关闭 irqbalance 服务。如果不关闭 irqbalance 服务，由脚本设置的中断 CPU 亲和性的值会不断被 irqbalance 改写，使脚本的作用失效。然后脚本找到每个网卡的每个队列对应的中断线，分别设定 CPU 亲和性。脚本链接是 https://gist.github.com/syuu1228/4352382，读者可以直接下载使用。

3. 中断优化脚本效果测试

运行脚本之前，用 Netperf 向服务器打压力（Netperf 的使用在第 15 章有详细介绍）。使用 mpstat -p ALL 观察服务器的 CPU 情况。

在看下面的数据之前，先介绍一下 mpstat 命令的输出。mpstat 命令使用 -p 参数选择要看 CPU 的哪个核心，ALL 表示显示所有 CPU 核心的数据。下面是 mpstat 命令的全部输出项。

- %usr：表示处理用户进程所使用 CPU 的百分比。
- %nice：表示使用 nice 命令影响的用户进程所使用 CPU 的百分比，nice 值可以影响进程的优先级。
- %sys：表示内核进程使用的 CPU 百分比。
- %iowait：表示等待 I/O 使用的 CPU 百分比。
- %irq：表示处理中断使用的 CPU 百分比。
- %soft：表示处理软中断使用的 CPU 百分比。
- %guest：表示虚拟机消耗的 CPU 百分比。
- %idle：表示 CPU 空闲的百分比。

图 4-19 所示是 mpstat -p ALL 观察服务器未运行脚本绑定中断时的 24 个 CPU 的运行情况。

09:56:30 AM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
09:35:23 AM	all	0.09	0.00	4.33	0.00	0.00	0.56	0.00	3.82	91.21
09:35:23 AM	0	0.00	0.00	13.13	0.00	0.00	0.00	0.00	72.73	14.14
09:35:23 AM	1	1.02	0.00	8.16	0.00	0.00	0.00	0.00	4.08	86.73
09:35:23 AM	2	0.00	0.00	8.16	0.00	0.00	0.00	0.00	9.18	82.65
09:35:23 AM	3	0.00	0.00	38.89	0.00	0.00	19.44	0.00	0.00	41.67
09:35:23 AM	4	0.00	0.00	6.00	0.00	0.00	0.00	0.00	0.00	94.00
09:35:23 AM	5	0.00	0.00	1.03	0.00	0.00	0.00	0.00	0.00	98.97
09:35:23 AM	6	1.10	0.00	6.59	0.00	0.00	0.00	0.00	0.00	92.31
09:35:23 AM	7	1.05	0.00	8.42	0.00	0.00	0.00	0.00	0.00	90.53
09:35:23 AM	8	0.00	0.00	4.21	0.00	0.00	0.00	0.00	0.00	95.79
09:35:23 AM	9	0.00	0.00	3.09	0.00	0.00	0.00	0.00	0.00	96.91
09:35:23 AM	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:35:23 AM	11	1.01	0.00	4.04	0.00	0.00	0.00	0.00	0.00	94.95
09:35:23 AM	12	1.03	0.00	3.09	0.00	0.00	0.00	0.00	3.09	92.78
09:35:23 AM	13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:35:23 AM	14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:35:23 AM	15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:35:23 AM	16	0.00	0.00	3.03	0.00	0.00	0.00	0.00	0.00	96.97
09:35:23 AM	17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:35:23 AM	18	0.00	0.00	5.10	0.00	0.00	0.00	0.00	0.00	94.90
09:35:23 AM	19	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:35:23 AM	20	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	99.01
09:35:23 AM	21	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:35:23 AM	22	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:35:23 AM	23	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00

图 4-19 未运行脚本绑定中断时的 24 个 CPU 的运行情况

1107114476@qq.com
dolphinebook.com

%soft 一列的数据显示，在不绑定中断的情况下，软中断集中在一个 CPU 核上，很容易形成性能瓶颈。

图 4-20 所示是运行脚本后 24 个 CPU 的测试结果。

09:56:30 AM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
09:56:31 AM	all	0.04	0.00	4.59	0.00	0.00	0.89	0.00	3.66	90.82
09:56:31 AM	0	0.00	0.00	14.29	0.00	0.00	7.14	0.00	0.00	78.57
09:56:31 AM	1	0.00	0.00	13.13	0.00	0.00	0.00	0.00	42.42	44.44
09:56:31 AM	2	0.00	0.00	19.78	0.00	0.00	8.79	0.00	18.68	52.75
09:56:31 AM	3	0.00	0.00	16.67	0.00	0.00	5.56	0.00	5.56	72.22
09:56:31 AM	4	0.00	0.00	7.07	0.00	0.00	0.00	0.00	19.19	73.74
09:56:31 AM	5	0.00	0.00	2.06	0.00	0.00	0.00	0.00	0.00	97.94
09:56:31 AM	6	0.00	0.00	6.19	0.00	0.00	0.00	0.00	0.00	93.81
09:56:31 AM	7	0.00	0.00	5.10	0.00	0.00	1.02	0.00	1.02	93.88
09:56:31 AM	8	0.00	0.00	5.05	0.00	0.00	0.00	0.00	0.00	94.95
09:56:31 AM	9	0.00	0.00	3.09	0.00	0.00	0.00	0.00	0.00	96.91
09:56:31 AM	10	0.00	0.00	3.09	0.00	0.00	0.00	0.00	0.00	96.91
09:56:31 AM	11	0.00	0.00	3.06	0.00	0.00	0.00	0.00	0.00	96.94
09:56:31 AM	12	1.00	0.00	2.00	0.00	0.00	0.00	0.00	0.00	97.00
09:56:31 AM	13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.04	97.96
09:56:31 AM	14	0.00	0.00	3.09	0.00	0.00	0.00	0.00	0.00	96.91
09:56:31 AM	15	0.00	0.00	1.01	0.00	0.00	0.00	0.00	0.00	98.99
09:56:31 AM	16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:56:31 AM	17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:56:31 AM	18	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:56:31 AM	19	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:56:31 AM	20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:56:31 AM	21	0.00	0.00	4.04	0.00	0.00	0.00	0.00	0.00	95.96
09:56:31 AM	22	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
09:56:31 AM	23	0.00	0.00	6.00	0.00	0.00	0.00	0.00	0.00	94.00

图 4-20　运行脚本后 24 个 CPU 的测试结果

可以看出，软中断被均衡到了多个 CPU 核上，这样就可以充分利用多个 CPU 核心带来的性能提升。

4. 网卡软中断分发的软件解决方法

RSS 需要网卡硬件的支持，在使用不支持 RSS 的网卡时，为了充分利用多核 CPU，CentOS 6.1 开始提供了 RPS（Receive Packet Steering）和 RFS（Receive Flow Steering）。

RPS 使网卡可以把一个 rx 队列的软中断分发到多个 CPU 核上，从而达到负载均衡的目的。RFS 是 RPS 的扩展，RPS 只依靠 hash 来控制数据包，提供了好的负载平衡，但是它没有考虑应用程序的位置（这个位置是指程序在哪个 CPU 上执行）。RFS 则考虑到了应用程序的位置。RFS 的目标是通过指派应用线程正在运行的 CPU 来进行数据包处理，以此来增加数据缓存的命中率。

开启 RPS/RFS 的脚本（脚本来自网上，出处不可考证）如下：

```
#!/bin/bash
# Enable RPS (Receive Packet Steering)

rfc=4096
cc=$(grep -c processor /proc/cpuinfo)
rsfe=$(echo $cc*$rfc | bc)
sysctl -w net.core.rps_sock_flow_entries=$rsfe
```

1107114476@qq.com
dolphinebook.com

```
for fileRps in $(ls /sys/class/net/eth*/queues/rx-*/rps_cpus)
do
    echo ffff > $fileRps
done

for fileRfc in $(ls /sys/class/net/eth*/queues/rx-*/rps_flow_cnt)
do
    echo $rfc > $fileRfc
done
      tail /sys/class/net/eth*/queues/rx-*/{rps_cpus,rps_flow_cnt}
```

其中，所有队列的 rps_flow_cnt 之和不能大于 rps_sock_flow_entries。rps_cpus 的值采用的是 CPU 位图，转换成二进制后，每一位代表一个 CPU 核心，1 表示软中断可以调度到这个核心，0 表示不可以。ffff 转换成二进制是 1111111111111111，表示软中断可以调度到 16 个核心。如果你的服务器 CPU 核心数跟笔者不同，可以修改该值，例如 8 个 CPU 核心的服务器，这个值可以设为 ff。

对虚拟化来说，RPS/RFS 的应用场景主要是虚拟机系统内，借助 RPS/RFS 软件方式可以实现虚拟机的软件中断分发。图 4-21 和图 4-22 所示的测试结果是从一台物理机向虚拟机用 netperf 打压力时的数据。

开启前：

10:58:00 AM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
10:58:01 AM	all	2.54	0.00	38.58	0.00	0.00	8.63	0.00	0.00	50.25
10:58:01 AM	0	0.00	0.00	1.02	0.00	0.00	0.00	0.00	0.00	98.98
10:58:01 AM	1	5.10	0.00	76.53	0.00	0.00	17.35	0.00	0.00	1.02

图 4-21 未开启虚拟机 RPS/RFS 中断情况

开启后：

11:09:29 AM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
11:09:29 AM	all	4.55	0.00	66.48	0.00	0.57	10.23	0.00	0.00	18.18
11:09:29 AM	0	5.88	0.00	68.63	0.00	0.00	7.84	0.00	0.00	17.65
11:09:29 AM	1	4.00	0.00	62.67	0.00	1.33	13.33	0.00	0.00	18.67

图 4-22 开启虚拟机 RPS/RFS 中断情况

从图 4-21 和图 4-22 中可以看出，开启前中断波动比较大，开启之后中断比较平衡，软中断被分配到了两个 CPU 核，从而使 CPU 的中断负载更加均衡。

5. 多队列 Virtio 网卡

RedHat 的两位工程师已经实现了多队列 Virtio 网卡驱动程序。其使用方法如下。

1）判断内核是否支持多队列。如果命令输出为空，则表示不支持多队列。

```
[root@localhost ~]# grep IFF_MULTI_QUEUE /usr/include/Linux/if_tun.h
```

2）多队列 Virtio 网卡配置。

```
<interface type='bridge'>
```

1107114476@qq.com
dolphinebook.com

```
    <mac address='52:54:00:43:6e:3f'/>
    <source bridge='clients'/>
    <model type='Virtio'/>
    <driver name='vhost' queues='N'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
</interface>
```

N 1 - 8 最多支持 8 个队列，在虚拟机上执行以下命令开启多队列网卡：

```
#ethtool -L eth0 combined M
```

M 1 - N，M 小于等于 N。

3）目前多队列 Virtio 网卡在小包转发上性能还不理想，有待优化，因此还不建议在生产环境使用。随着 CentOS 7 的版本迭代，相信 Virtio 网卡的性能会得到改善。

4.4　网卡 PCI Passthrough 技术

如果虚拟机对网络的要求非常高，通过 PCI Passthrough 技术将物理网卡直接给虚拟机使用，如图 4-23 所示，虚拟机单独使用网卡，可以达到几乎和物理网卡一样的性能。

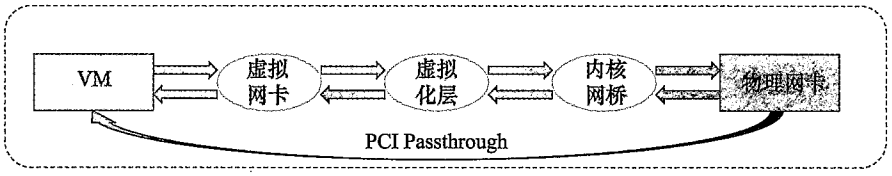


图 4-23　网卡 PCI Passthrough 技术示意图

1. 网卡 PCI Passthrough 配置

PCI Passthrough 配置方法如下。

1）查看网卡设备信息。使用 lspci 命令查看网卡设备信息。

```
04:00.0 Ethernet controller: Intel Corporation 82571EB Gigabit Ethernet
Controller (rev 06)
04:00.1 Ethernet controller: Intel Corporation 82571EB Gigabit Ethernet
Controller (rev 06)
```

也可以使用 virsh nodedev-list -tree 命令得到相关信息。

```
+- pci_0000_00_07_0
|   |
|   +- pci_0000_04_00_0
|   |   |
|   |   +- net_p1p1_00_1b_21_88_69_dc
|   |
|   +- pci_0000_04_00_1
|       |
```

1107114476@qq.com
dolphinebook.com

```
|        +- net_p1p2_00_1b_21_88_69_dd
```

2）使用 virsh nodedev-dumxml pci_0000_04_00_0 命令得到 .xml 配置信息。

```
[root@]# virsh nodedev-dumpxml pci_0000_04_00_0
<device>
 <name>pci_0000_04_00_0</name>
 <parent>pci_0000_00_07_0</parent>
 <driver>
   <name>e1000e</name>
 </driver>
 <capability type='pci'>
   <domain>0</domain>
   <bus>4</bus>
   <slot>0</slot>
   <function>0</function>
   <product id='0x105e'>82571EB Gigabit Ethernet Controller</product>
   <vendor id='0x8086'>Intel Corporation</vendor>
 </capability>
</device>
```

3）编辑虚拟机 xml 文件，加入 PCI 设备信息。

```
<hostdev mode='subsystem' type='pci' managed='yes'>
    <source>
      <address domain='0x0000' bus='0x04' slot='0x00' function='0x0'/>
    </source>
</hostdev>
```

Domain bus slot function 信息从 nodedev-dumxml 处的 .xml 文件获取，然后使用 virsh define 命令定义虚拟机，再开启虚拟机就可以了。

因为附加上去的是物理设备，所以需要在虚拟机操作系统里面安装相应的物理网卡驱动。

2. 网卡 PCI Passthrough 的应用场景

PCI Passthrough 技术是虚拟化网卡的终极解决方案，能够让虚拟机独占物理网卡，达到最优的性能，可以在网卡性能要求非常高的场景使用。但是因为 PCI Passthrough 是独占网卡，所以对宿主机的网卡数量有要求。目前主流的服务器都有 4 块以上的网卡，实际使用的时候，可以将 1 ~ 2 块网卡通过 PCI Passthrough 技术分给网络压力非常大的虚拟机，然后其他虚拟机共享剩余的网卡。

4.5 SR-IVO 虚拟化技术

SR-IOV 是比较好的虚拟化硬件性能解决方案，使用需要特定的硬件支持，缺点是目前

1107114476@qq.com
dolphinebook.com

还不支持虚拟机在线迁移。在单机虚拟化的场景，可以考虑使用 SR-IOV 来解决性能及虚拟机之间资源平衡分配的问题。

1. SR-IOV 的技术的原理及优点

（1）SR-IOV 的原理

SR-IVO 是 the Single Root I/O Virtualization 的简写，是一个将 PCI-E 设备共享给虚拟机使用的标准，多用在网络设备上。理论上也可以支持其他的 PCI 设备，目前也有一些显卡使用 SR-IOV 方案，主要用于桌面虚拟化，但是大多还是实验原型，相信很快就能看到显卡的 SR-IOV 商业化产品。

如图 4-24 所示，SR-IOV 是一种硬件解决方案，提供了一种从硬件上绕过系统和虚拟化层，并且使每个虚拟机能有单独的内存地址、中断、DMA 流。SR-IOV 在架构上简单讲就是一块物理网卡，开多个 VFs（Virtual Functions）实例。SR-IOV 上有两个功能类型。

- Physical Functions（PFs）：拥有全功能 PCI-E 功能，用于配置管理 SR-IOV。
- Virtual Functions（VFs）：只有轻量级的 PCI-E 功能，只包含数据传输必要的资源，但是资源可以非常细致地配置，每个 PF 最多可有 64 000 个与其关联的虚拟功能（Virtual Function，VF）。

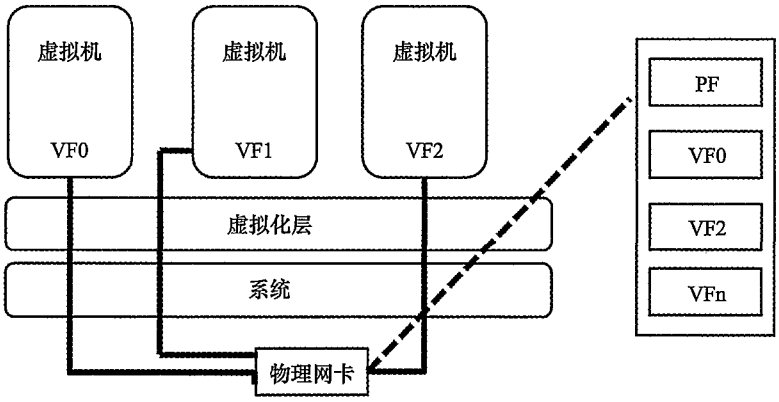


图 4-24　SR-IOV 原理

（2）SR-IOV 的优点

SR-IOV 标准允许高效共享 PCI-E 设备，有以下优点：

- 良好的性能，虚拟机绕过虚拟化层和系统，直接访问硬件，没有虚拟化层软件模拟的开销。
- 降低成本，减少了设备数量，例如网卡的 SR-IOV 减少了网卡数量、交换机端口、网线。

2. 网卡 SR-IOV 的配置

SR-IOV 的配置需要先配置宿主机 PF，然后将子网卡通过网卡独占的方式供虚拟机使用。

1107114476@qq.com
dolphinebook.com

（1）加载 SR-IOV 内核模块

通过 modprobe 命令加载 igb 模块。

```
modprobe igb
```

实际加载的时候，需要激活虚拟功能（VF）。

```
modprobe igb max_vfs=7
```

千兆网卡最多支持 8 个 VF（0 ~ 7）。千兆网卡目前支持比较好的是 Intel I350，Intel 82576 虽然也支持 SR-IOV，但是只支持虚拟机是 Linux 系统，Windows 系统不支持。

万兆网卡最多支持 64 个 VF（0 ~ 63），Intel 的新一代万兆网卡 X520（82599）、X540 都支持 SR-IOV 技术。

提示 如果是主板集成的网卡，一般在 BIOS 中 SR-IOV 功能是关闭的。如果要使用，需要在 BIOS 中打开 SR-IOV 选项。

如果需要重新设置 VF，可以删除模块再重新加载。

```
modprobe -r igb
```

将配置永久写入配置文件。

```
echo "options igb max_vfs=7" >>/etc/modprobe.d/igb.conf
```

通过 lspci 命令可以看多主网卡和子网卡。

```
# lspci | grep 82576
0b:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
 0b:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network
 Connection (rev 01)
0b:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
...
0b:10.7 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
...
```

（2）子网卡的使用

虚拟机可以通过网卡独占的方式使用子网卡。

```
# virsh nodedev-list | grep 0b
pci_0000_0b_00_0
...
pci_0000_0b_10_6
pci_0000_0b_11_7
pci_0000_0b_11_1
...
```

1107114476@qq.com
dolphinebook.com

虚拟机网卡 xml 配置文件如下：

```
<interface type='hostdev' managed='yes'>
    <source>
      <address type='pci' domain='0' bus='11' slot='16' function='0'/>
    </source>
  </interface>
```

虚拟机需要安装网卡驱动才能使用子网卡，高版本的 Linux 系统可以自动识别子网卡，Windows 系统只能是 2008 Server 以上版本，并且要安装最新的 Intel 网卡驱动。

4.6　虚拟化软件交换机 Open vSwitch 的安装与配置

Open vSwitch 是一个由软件实现的虚拟交换机，由 Nicira Networks 主导开发。它的目的是通过编程扩展让大型网络的管理变得自动化，同时仍然支持标准的管理接口和协议（例如 NetFlow、sFlow、SPAN、RSPAN、CLI、LACP、802.1ag）。此外，Open vSwitch 还可以跨物理服务器，实现类似于 VMware 的 vNetwork 分布式 vSwitch 或 Cisco Nexus 1000 V 的功能。

KVM 虚拟机可以通过 Open vSwitch 接入网络。相比传统的桥接方式，Open vSwitch 具有更高的灵活性，例如可以在线更改虚拟机的 VLAN 等。在使用 Open vSwitch 管理虚拟机网络之前，先介绍一下 Open vSwitch 的一些基本概念。

- Bridge：Bridge 代表一个以太网交换机（Switch），一个主机中可以创建一个或者多个 Bridge 设备。
- Port：端口相当于物理交换机的端口，每个 Port 都隶属于一个 Bridge。
- Interface：连接到 Port 的网络接口设备。在通常情况下，Port 和 Interface 是一对一的关系，只有在配置 Port 为 bond 模式后，Port 和 Interface 才是一对多的关系。
- Controller：OpenFlow 控制器。OVS 可以同时接受一个或者多个 OpenFlow 控制器的管理。
- datapath：在 OVS 中，datapath 负责执行数据交换，也就是把从接收端口收到的数据包在流表中进行匹配，并执行匹配到的动作。
- Flow table：每个 datapath 都和一个 Flow table 关联，当 datapath 接收到数据之后，OVS 会在 Flow table 中查找可以匹配的 Flow，执行对应的操作，例如转发数据到另外的端口。

Open vSwitch 的理解和使用非常复杂，不仅需要系统知识，还需要网络方面的知识，受篇幅限制，本书将着重讲解 Open vSwitch 的安装及几个典型环境的配置方法。

4.6.1　Open vSwitch 安装

一般情况推荐使用包管理工具 yum 或 apt 安装 Open vSwitch，这样做方便而且易于维护。如果想要尝试比较新的 Open vSwitch 版本，可以使用源代码安装的方式。

1107114476@qq.com
dolphinebook.com

1. 通过源代码的方式安装 Open vSwitch

1）安装 gcc 等相关软件包。

```
[root@localhost ~]# yum install gcc kernel-devel
```

2）安装 OVS。

```
[root@localhost ~]# wget http://KVM.org/releases/KVM-2.3.0.tar.gz
[root@localhost ~]# tar xvf KVM-2.3.0.tar.gz
[root@localhost ~]# cd KVM-2.3.0
[root@localhost KVM-2.3.0]# ./configure --with-Linux=/lib/modules/`uname -r`/build
[root@localhost KVM-2.3.0]# make&&make install
# 如果这一步报错：
/root/KVM-2.3.0/datapath/Linux/compat/include/Linux/skbuff.h:298: error:
'structsk_buff' has no member named 'l4_rxhash'
# 需要把 /root/KVM-2.3.0/datapath/Linux/compat/include/Linux/skbuff.h 的第 298 行注释掉
# 加载模块
[root@localhost KVM-2.3.0]# modprobeKVM
```

3）配置。

```
[root@localhost ~]#mkdir -p /usr/local/etc/KVM
# 创建数据库文件
[root@localhost ~]# ovsdb-tool create /usr/local/etc/KVM/conf.db
# 启动数据库
[root@localhost ~]# ovsdb-server \
 --remote=punix:/usr/local/var/run/KVM/db.sock --pidfile --detach
# 初始化数据库
[root@localhost ~]# ovs-vsctl --no-wait init
# 启动 ovs-vswitchd 进程
[root@localhost ~]# ovs-vswitchd --pidfile --detach
# 检查两个进程是否正常
[root@localhost ~]# psaux|grepovs
root 10898 0.0 0.1 11952 1268 ? Ss 14:03 0:00 ovsdb-server --remote=punix:/usr/
local/var/run/KVM/db.sock --pidfile --detach
root 10930 0.0 0.1 12368 1372 ?Ss 14:06 0:00 ovs-vswitchd --pidfile --detach
root 10970 0.0 0.0 103252 876 pts/0 S+ 14:21 0:00 grepovs
```

2. 通过 yum 的方式安装 Open vSwitch

通过源码安装 OVS 过程复杂，容易遇到问题，而且不易管理，因此不推荐使用。实际使用中最好用 yum 或 apt 等包管理工具进行安装。下面介绍如何使用 yum 安装 Open vSwitch。

1）添加 repo 源。由于工作的关系，笔者一般用的是 Fedora 的 openstack-grizzly 的源。

```
[root@localhost ~]# vim /etc/yum.repos.d/rdo-release.repo
[openstack-grizzly]
name=OpenStack Grizzly Repository
baseurl=http://repos.fedorapeople.org/repos/openstack/EOL/openstack-grizzly/epel-6/
enabled=1
```

1107114476@qq.com
dolphinebook.com

```
skip_if_unavailable=1
gpgcheck=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-RDO-Grizzly
priority=98
```

2）安装。

```
[root@localhost ~]# yum install KVM
```

3）启动服务并设置为开机自启动。

```
[root@localhost ~]# chkconfig KVM on
[root@localhost ~]# /etc/init.d/KVM start
```

OVS 有一套强大的命令行管理工具，命令数量比较多且难记，因此不准备专门花篇幅介绍命令，在接下来的几节中将直接在配置案例中演示。

4.6.2　Open vSwitch 典型配置

1. KVM 虚拟机通过 Open vSwitch 接入网络

宿主机环境操作系统为 CentOS 6.4，双网卡 eth0、eth1，其中 eth0 作为宿主机的管理口，eth1 外连交换机的 trunk 端口，预留给虚拟机流量使用。

1）创建网桥。

```
[root@localhost ~]# ovs-vsctl add-br br0
```

2）把 eth1 加入 br0。

```
[root@localhost ~]# ovs-vsctl add-port br0 eth1
```

3）编辑虚拟机配置文件。需要注意，Libvirt 从 0.9.11 版本开始才支持以下的配置。

```
<interface type='bridge'>
    <mac address='fa:16:3e:fc:e0:c0'/>
    <source bridge='br0'/>
    <virtualport type='KVM'/>
    <vlan>
        <tag id='2'/>
    </vlan>
    <targetdev='tap1'/>
    <model type='Virtio'/>
</interface>
```

启动虚拟机后可以看到 OVS 的配置情况。

```
[root@localhost ~]# ovs-vsctl show
ca36676e-cb43-448b-aefe-23ff0b96bac5
Bridge "br0"
Port "eth1"
Interface "eth1"
```

1107114476@qq.com
dolphinebook.com

```
Port "br0"
Interface "br0"
type: internal
Port "tap1"
tag: 2
Interface "tap1"
type: internal
```

在 Open vSwitch 中，port 默认属于 trunk 模式，除非指定其 VLAN tag。前面提到 eth1 外连到了交换机的 trunk 端口，因此这里没有给 eth1 设置 tag。根据 VLAN 的知识，除了 native VLAN（一般为 VLAN 1）之外，所有二层帧必须包含正确的 VLAN tag 才能正常通过 trunk 链路，所以虚拟机对应的 tap1 端口也必须有 VLAN tag。笔者给虚拟机的网段属于 VLAN 2，所以这里 tag 设置为 2。

既然 eth1 是 trunk 模式，这台宿主机上就可以同时运行属于不同 VLAN 的虚拟机了，只需要给虚拟机对应的 tap 网卡设置 VLAN tag 即可。如果用传统的 Linux bridge 要实现类似的功能，配置相当复杂，而且无法进行在线更改，而在 OVS 中使用命令 ovs-vsctl set port tap1 tag=3 即可改变虚拟机的 VLAN，这是 OVS 所强调的灵活性的一个体现。

2. Open vSwitch bond 配置

Linux bond 应用已经非常广泛，它在提高网络可靠性的同时还能提高性能。OVS 同样提供了对 bond 的支持，下面是配置过程。

```
# 创建一个网桥
[root@localhost ~]# ovs-vsctl add-br br0
# 将网卡 eth2、eth3 配置成一个 bond0，并加入到 br0 中，lacp 状态设为 active
[root@localhost ~]# ovs-vsctl add-bond br0 bond0 eth2 eth3 lacp=active
# 将 bond mode 设为 balance-slb
[root@localhost ~]# ovs-vsctl set port bond0 bond_mode=balance-slb
# 查看 bond0 的具体信息
[root@localhost ~]# ovs-appctl bond/show bond0
```

Open vSwitch bond 常用的有以下几种模式。

- active-backup：将所有流量分配给同一块物理网卡，当这块物理网卡失效时，将流量切换至其他网卡。
- balance-slb：将流量根据源 mac 地址和输出 vlan ID 在物理网卡之间进行负载均衡。
- balance-tcp：与 balance-slb 类似，除了根据 L2 的信息外，这种模式还可以根据 L3、L4 的信息进行负载均衡，如 IP 地址、TCP 端口等。但是需要注意的是，这种模式的正常工作需要上游交换机支持 802.3ad 且 lacp 协商成功，如果不满足条件，KVM 将会把 bond mode 回退到 balance-slb。

lacp 的值有 3 种：active、passive 和 off。在本节刚开始的配置案例中，上游交换机不需要任何额外配置，lacp 的状态是任何一种都可以；但是，如果交换机上对应端口配置了 port channel，Open vSwitch bond 的 lacp 就必须设置为 active 网络才能正常。这一点需要特别注

1107114476@qq.com
dolphinebook.com

意，刚开始学习 Open vSwitch bond 的配置很容易犯错。

3. 连接两个 Open vSwitch 网桥

将两个网桥连接起来，就相当于现实中用网线将两台真实交换机连接起来。

（1）利用 VETH 设备

VETH（Virtual Ethernet）设备出现较早，它的作用是反转通信数据的方向，需要发送的数据会被转换成需要收到的数据重新送入内核网络层进行处理，从而间接地完成数据的注入。VETH 设备总是成对出现，送到一端请求发送的数据总是从另一端以请求接收的形式出现。该设备不能被用户程序直接操作，但使用起来比较简单。创建并配置正确后，向其一端输入数据，VETH 会改变数据的方向并将其送入内核网络核心，完成数据的注入。在另一端能读到此数据。

下面是配置过程。

```
# 创建两个网桥
[root@localhost ~]# ovs-vsctl add-br br0
[root@localhost ~]# ovs-vsctl add-br br1
# 创建一对 veth 设备
[root@localhost ~]# ip link add name veth0 type veth peer name veth1
# 分别将两个 veth 设备加入到 br0 和 br1
[root@localhost ~]# ovs-vsctl add-port br0 veth0
[root@localhost ~]# ovs-vsctl add-port br1 veth1
```

这样，两个网桥就连接起来了。

（2）利用 Open vSwitch 的 patch 类型的端口

patch 类型的端口是也是成对存在的，其作用相当于现实中的 patch cable。

下面是配置过程。

```
# 创建两个网桥
[root@localhost ~]# ovs-vsctl add-br br0
[root@localhost ~]# ovs-vsctl add-br br1
# 创建一对 patch 端口，并分别加入两个网桥
[root@localhost ~]# ovs-vsctl add-port br0 patch-to-br0
[root@localhost ~]# ovs-vsctl set interface patch-to-br0 type=patch
[root@localhost ~]# ovs-vsctl set interface \
 patch-to-br0options:peer=patch-to-br1
[root@localhost ~]# ovs-vsctl add-port br1 patch-to-br1
[root@localhost ~]# ovs-vsctl set interface patch-to-br1 type=patch
[root@localhost ~]# ovs-vsctl set interface \
 patch-to-br1options:peer=patch-to-br0
```

4.7　多网卡绑定与建桥

在生产环境中，通过多网卡绑定，可以提高可靠性和带宽。在 CentOS 7 系统中，宿主机配置多网卡绑定和建桥的方法如下。

1107114476@qq.com
dolphinebook.com

（1）配置 modprobe 文件

内容如下：

```
cat /etc/modprobe.d/nicbond.conf
alias bond0 bonding
options bond0 mode=1 miimon=100
```

多网卡绑定可以有 0 ~ 6 共 7 模式。

- mode=0：多端口轮询发送数据包。
- mode=1：主备模式。
- mode=2：基于 Hash 算法发送数据包。
- mode=3：每个接口上传输每个数据包，此模式提供了容错能力。
- mode=4：基于（802.3ad）IEEE 802.3ad Dynamic link aggregation（IEEE 802.3ad 动态链接聚合），需要和交换机配合。
- mode=5：每块网卡根据当前的负载（根据速度计算）分配外出流量。
- mode=6：针对 IPv4 做了优化。

（2）挂载 bonding 模块

```
modprobe bonding
```

（3）配置多网卡绑定接口

服务器有 4 个网口。

```
enp3s0f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        ether 1c:c1:de:1e:a1:f2 txqueuelen 1000 (Ethernet)
        RX packets 6716921 bytes 745694715 (711.1 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 689079 bytes 174181455 (166.1 MiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp3s0f1: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        ether 1c:c1:de:1e:a1:f4 txqueuelen 1000 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp4s0f0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        ether 1c:c1:de:1e:a1:f6 txqueuelen 1000 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp4s0f1: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        ether 1c:c1:de:1e:a1:f8 txqueuelen 1000 (Ethernet)
```

1107114476@qq.com
dolphinebook.com

```
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

目标是绑定 enp4s0f0 和 enp4s0f1 两个网口，配置方法如下。

1）先配置聚合接口。

```
cd /etc/sysconfig/network-scripts
cat ifcfg-bond0
DEVICE=bond0
ONBOOT=YES
BRIDGE=br1
```

2）配置 enp4s0f0。

```
cat ifcfg-enp4s0f0
HWADDR=1C:C1:DE:1E:A1:F6
TYPE=Ethernet
BOOTPROTO=none
NAME=enp4s0f0
UUID=2ae88bd4-5d49-42e8-b247-702738c954ac
ONBOOT=yes
MASTER=bond0
SLAVE=yes
```

3）配置 enp4s0f1。

```
cat ifcfg-enp4s0f1
HWADDR=1C:C1:DE:1E:A1:F8
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
PEERDNS=yes
PEERROUTES=yes
NAME=enp4s0f1
UUID=d1a335d6-dec4-4ba2-aaea-0b7c14b653c6
ONBOOT=yes
MASTER=bond0
SLAVE=yes
```

4）配置虚拟网桥。

```
cat ifcfg-br1
DEVICE=br1
TYPE=Bridge
IPADDR=192.168.20.200
NETMASK=255.255.255.0
ONBOOT=yes
```

5）重启网络，然后查看配置效果。

1107114476@qq.com
dolphinebook.com

```
systemctl restart network.service
ifconfig
br1: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        inet 192.168.20.200 netmask 255.255.255.0 broadcast 192.168.20.255
        ether 1a:bf:fc:5a:da:91 txqueuelen 0 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

6）查看 /proc/net/bonding/bond0，可以看到绑定端口的状态。

```
cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)
Bonding Mode: load balancing (round-robin)
MII Status: down
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0
```

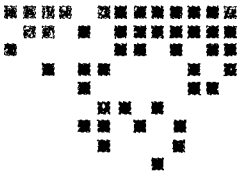
4.8 本章小结

本章介绍的网络虚拟化技术比较多，各种技术的应用场景总结如下。

- KVM 网络优化方案分为软件和硬件，软件主要是尽量让虚拟机靠近网络网卡，直至独占网络网卡，一般情况下推荐尽量使用 Virtio 驱动。
- 硬件方面，万兆网卡及 SR-IOV 的方案会越来越普及，但是 SR-IOV 的缺点是不支持在线迁移，这个问题有待解决。
- 多网卡绑定是很好的提升带宽和冗余的方案，可以在强调高可用的环境使用。
- 目前 Open vSwitch 是开源虚拟化网络事实上的标准，功能会越来越完善，使用也会越来越广泛。Open vSwitch 目前工作已经非常稳定，建议在生产环境中可以部署起来，以满足虚拟化网络管理的需求。

虚拟化的性能难点之一是网络 I/O 和磁盘 I/O，下一章将为读者介绍磁盘虚拟化技术及应用场景。

1107114476@qq.com
dolphinebook.com



第 5 章 Chapter 5

KVM 磁盘虚拟化技术与应用场景

本章将为读者详细介绍 KVM 虚拟化中的磁盘技术，以及每种技术的应用场景。

KVM 虚拟机磁盘的软件调优首先要对各种镜像方式有一定的了解，调优主要是不同业务场景的磁盘镜像方式及缓存方式选择。虚拟机和物理机文件系统的块对齐对磁盘性能影响也很大，在生产环境中也要引起足够重视。

在虚拟化实践中，经常需要对虚拟机镜像调整，修改虚拟机内部的配置文件。guestfish 是一套镜像管理的利器，不仅可以调整镜像的大小，还可以在虚拟机不运行的情况下，直接修改虚拟化内部的文件，本章将会详细介绍 guestfish 的使用。

硬件方面，SSD 这几年使用越来越广泛，本章将会详细讨论 SSD（Solid State Drives）在 KVM 虚拟化中的技术及应用场景。

5.1 磁盘虚拟化方式性能比较与应用场景

虚拟化实施的时候，经常面临一个问题，就是在不同的场景下如何选择虚拟化磁盘方式、每种磁盘方式的性能如何。本节将介绍磁盘虚拟化的方式，以及每种磁盘方式的性能对比及应用场景。

5.1.1 QEMU 磁盘虚拟化方式概述

QEMU 最初是一款软件模拟器，所以 QEMU 的磁盘也都是通过软件模拟的。后来 QEMU 和 KVM 结合，性能有了很大提升，磁盘方面也有了 Virtio 这样的半虚拟化技术。

1107114476@qq.com
dolphinebook.com

1. QEMU 可以模拟的磁盘类型

KVM 虚拟机在配置磁盘的时候，可以指定 IDE、SATA、Virtio、Virtio-SCSI 几种磁盘类型，CentOS 6.x 只支持 IDE 和 Virtio 磁盘类型，CentOS 7.x 增加支持 SATA 和 Virtio-SCSI 磁盘类型。从虚拟的方式来看，IDE、SATA 是纯软件模拟的磁盘，也称为全虚拟化的磁盘；Virtio、Virtio-SCSI 是半虚拟化的磁盘，通过改造虚拟机系统的驱动来达到提升性能的目标。

Virtio 驱动主要是绕过 QEMU 软件模拟这一层，让虚拟机的操作系统可以和内核的虚拟化层直接通信，提高通信效率。

2. 每种磁盘类型的最大磁盘数量

IDE 设备同物理机一样，一台虚拟机最多挂载 4 个 IDE 设备。Virtio 是模拟 PCI 设备给虚拟机使用，因此一个 Virtio 磁盘会占用虚拟机的一个 PCI 槽位，所以一台虚拟机能挂载的 Virtio 磁盘数量受虚拟机系统能支持的 PCI 插槽数量限制，一般最多是 30 个左右。Virtio-SCSI 使用 Virtio 的方式，但是走的是 SCSI 通道，可以挂载更多的磁盘，理论上一个 SCSI 通道可以挂载 256 块硬盘。

5.1.2 IDE 与 Virtio 性能比较测试

IDE 虚拟磁盘是兼容性最好的一种方式，在一些特定的环境，必须使用低版本操作系统的虚拟机，只能选择使用 IDE，比如 CentOS 4.x 系列、CentOS 5.3 以前的版本，以及 Windows 2000 以前的版本。对于比较新的操作系统，尽量使用 Virtio 的驱动，性能会好很多，并且 Windows 系统 Virtio 的驱动尽量使用比较新的官方驱动。

从理论上讲，Virtio 会比 IDE 磁盘性能好很多，因为 IDE 是全软件模拟，Virtio 是半虚拟化驱动。Virtio 改造了虚拟机，使虚拟机可以直接和虚拟化层通信，极大地提高通信效率。

1. 测试环境

- 宿主机和虚拟机版本：CentOS release 6.6（Final）。
- 磁盘：宿主机，6 块 300GB RAID 5；虚拟机，100GB IDE 硬盘一块，以及 100GB Virtio 磁盘一块。
- 虚拟机磁盘缓存方式为 writethough。
- 镜像方式为 qcow2。

2. 测试结果

IDE、Virtio 4K 随机读测试结果如图 5-1 所示。

IDE、Virtio 4K 随机写测试结果如图 5-2 所示。

可以明显看到 Virtio 比 IDE 性能好很多。

1107114476@qq.com
dolphinebook.com

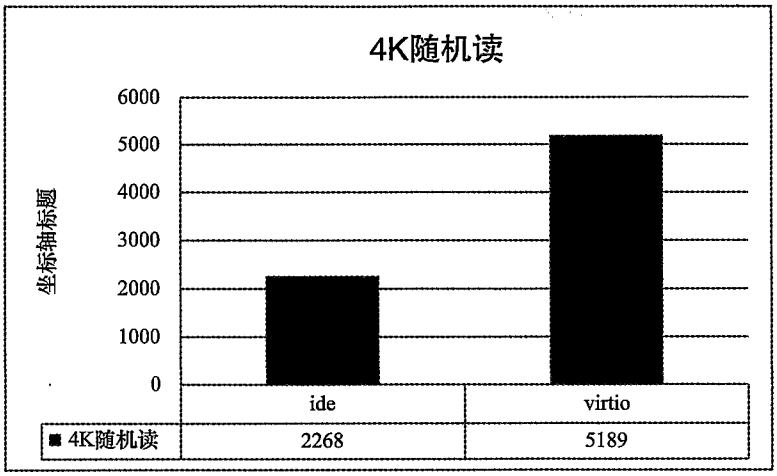


图 5-1　IDE、Viritio 4K 随机读测试

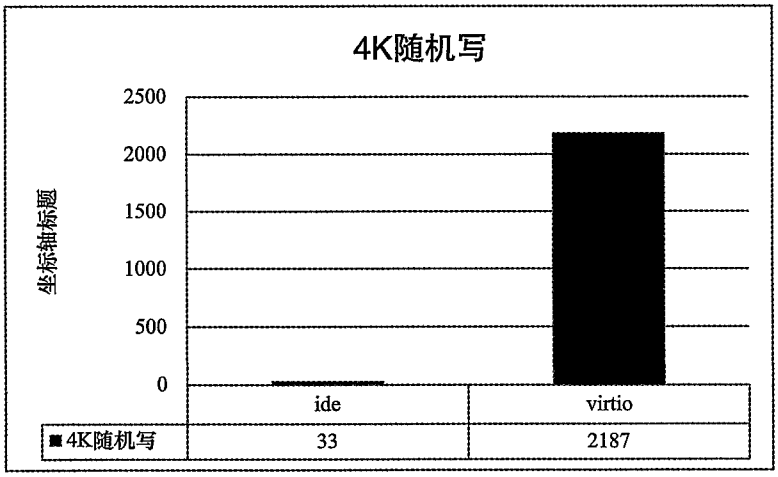


图 5-2　IDE、Viritio 4K 随机写测试

5.2　Virtio 磁盘缓存方式详解、性能比较与应用场景

KVM 虚拟机在配置磁盘的时候，可以指定几种缓存模式。每一台虚拟机的磁盘接口可以配置成 writethrough、writeback、none、directsync 或者 unsafe 缓存模式。如果没有指明缓存模式，KVM 使用默认的缓存方式。

如果虚拟机和它的存储设备失去连接会造成数据不一致，缓存模式的使用有可能会造成数据丢失。缓存方式也会剧烈影响磁盘性能。另外，在某些场景下，一些磁盘缓存模式会和在线迁移冲突。磁盘镜像、镜像位置、存储子系统、缓存方式没有简单的最佳规则，需要规

1107114476@qq.com
dolphinebook.com

划和测试每台虚拟机的各种情况以选择最优的性能。

5.2.1 磁盘缓存详解

1. 磁盘的 6 种缓存方式介绍

图 5-3 所示是磁盘 I/O 从虚拟机到宿主机物理存储的历程。

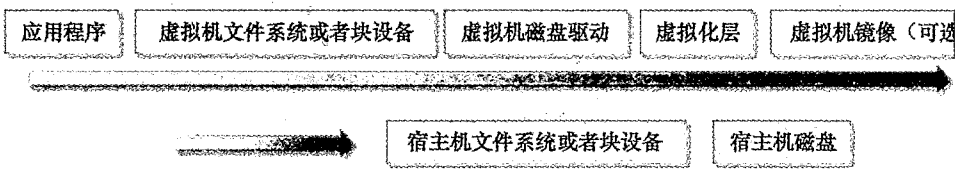


图 5-3 磁盘 I/O 从虚拟机到宿主机流向

缓存模式实际是在虚拟化层和宿主机的文件系统或者块设备打开或关闭缓存的各种组合。

（1）没有指定缓存方式

低于 1.2 版本的 QEMU-KVM，如果没有指定缓存方式，默认使用 writethrough 模式。1.2 之后的版本，大量的 writeback 或者 writethrough 缓存模式的虚拟化存储接口语义问题被修复，允许默认的缓存模式切换到 writeback。IDE、SCSI 和 Virtio 等虚拟机的磁盘驱动，虚拟机内核有能力禁止写缓存模式，导致缓存模式转换为 writethrough。但是在 CentOS 系统上虚拟机默认的缓存模式是 none。

（2）缓存为 writethrough 模式

这种模式设置虚拟机的磁盘镜像文件或者块设备为 O_DSYNC 语义，数据只有合并写入存储设备才会返回成功报告。宿主机的页面缓存在这种模式下工作在透写方式。虚拟机的磁盘驱动告知虚拟机没有回写缓存，所以虚拟机不需要发出刷盘命令以保持数据一致性。存储设备的行为就像是透过缓存。

（3）缓存为 writeback 模式

这种模式设置虚拟机的磁盘镜像文件或者块设备既不是 O_DSYNC 也不是 O_DIRECT 语义，所以可以使用宿主机页面缓存，数据到达宿主机页面缓存就给虚拟机返回写成功报告，页面缓存管理机制会管理数据的合并写入宿主机存储设备。另外，虚拟机磁盘控制器被通知可以使用回写缓存，于是虚拟机在需要保证数据一致性的时候会发出刷盘命令。其过程就像 RAID 控制器的 RAM 缓存机制。

（4）缓存为 none 模式

这种模式设置虚拟机的磁盘镜像文件或者块设备为 O_DIRECT 语义，所以宿主机的页面缓存被绕过，I/O 直接在 QEMU-KVM 的用户空间缓存和宿主机存储设备间发生。由于实际的存储设备可能在数据被放入写入队列就报告数据写操作完成，虚拟机的存储控制器被告

1107114476@qq.com
dolphinebook.com

知有回写缓存，于是虚拟机在需要保证数据一致性的时候会发出刷盘命令。相当于直接访问主机的磁盘，并且有优越的性能。

（5）缓存为 unsafe 模式

这种模式同上面讨论的 cache=writeback 非常类似，unsafe 模式关键的地方是所有的虚拟机刷盘指令会被忽略，使用这个模式意味着接受宿主机故障的时候数据丢失的风险以换取性能。这个模式可以在系统安装的时候使用，不建议在生产环境中使用。

（6）缓存为 directsync 模式

这种模式设置虚拟机的磁盘镜像文件或者块设备同时使用 O_DSYNC 和 O_DIRECT 语义，只有数据被合并写入存储设备才会报告写操作成功。这种模式也绕过宿主机的页面缓存，类似 cache=writethrough 模式，虚拟机也不需要发出刷盘命令。这种模式是最后一种语义缓存和直接访问的可能的组合，是缓存方式的一种补充。

2. 缓存模式的数据一致性

下面按模式划分来介绍数据的一致性。

（1）缓存为 writethrough、none、directsync 模式

这些是比较安全的模式，可以考虑用于保持数据一致性的场景，虚拟机可以在需要的时候刷盘。如果是一台可靠性要求非常高的虚拟机，请使用 writethough 或者 directsync 模式。注意在一些文件系统中不兼容 none 或者 directsync 的模式，当这些缓存模式被开启的时候，这些文件系统不支持 O_DIRECT 语义。

（2）缓存为 writeback 模式

这种模式通知虚拟机工作在回写模式，依靠虚拟机在必要的时候发起刷盘命令保持虚拟机镜像的数据一致性。这和现代文件系统存储设计思路保持完全一致。但是必须注意在数据报告写完成和真正地合并写到存储设备上一个时间窗口期，这种模式在宿主机故障的时候会丢失数据。

（3）缓存为 unsafe 模式

这种模式同 writeback 非常相似，只是希望忽略虚拟机的刷盘指令，通过刷盘保持数据一致性是无效的，所以宿主机故障的时候数据丢失的风险非常高。被命名为 unsafe 也是警告这是宿主机故障的时候数据丢失风险最高的一种模式。注意只有在虚拟机被关闭的时候才有数据刷盘的动作。

5.2.2　缓存方式的性能比较测试

下面是一个简单的测试，验证一下各种缓存方式的性能差异。

1. 测试环境

宿主机和虚拟机版本：CentOS release 6.6（Final）。

1107114476@qq.com
dolphinebook.com

磁盘：宿主机，6 块 300GB RAID 5；虚拟机，100GB Virtio 磁盘一块；镜像方式为 qcow2。

2. 测试方法

分别修改虚拟机磁盘缓存方式为 writeback、witethroug、none、unsafe 测试 4K 随机读、4K 随机写性能。

3. 测试结果

如图 5-4 所示，4K 随机读 none 模式最好，其他三种方式相差不大。

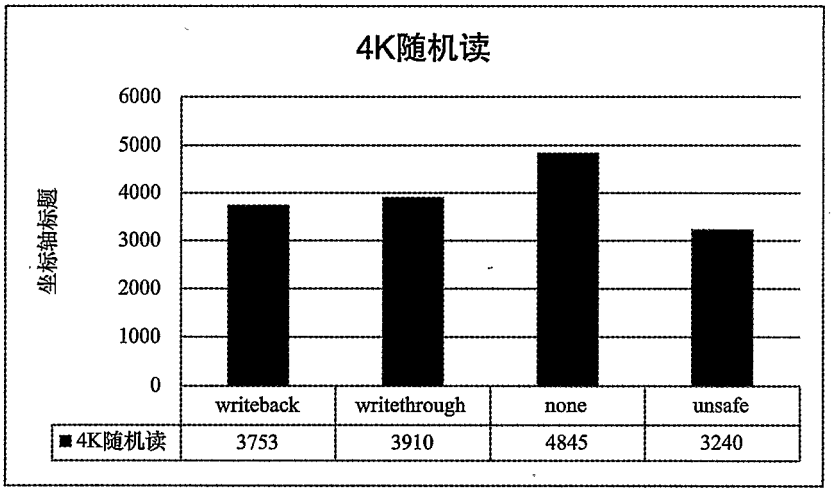


图 5-4 几种缓存方式的 4K 随机读比较

如图 5-5 所示，4K 随机写 writeback、none、unsafe 三种方式差不多，writethrough 性能最差。

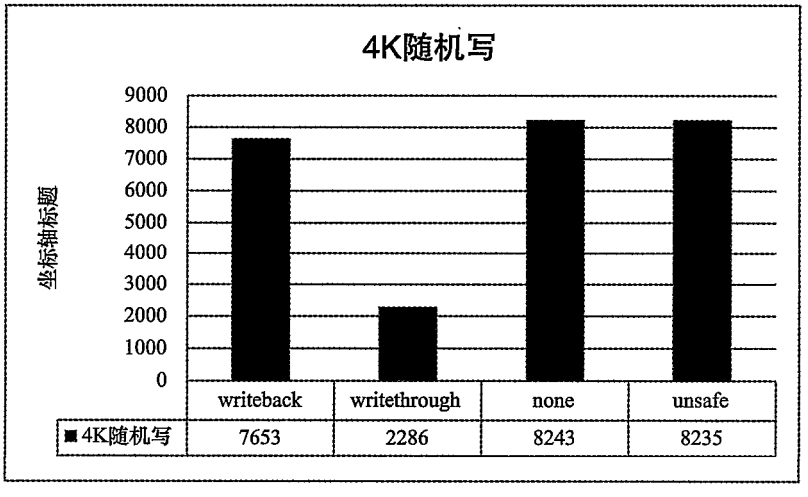


图 5-5 几种缓存方式的 4K 随机写比较

1107114476@qq.com
dolphinebook.com

5.2.3　缓存方式的应用场景

1. 缓存方式对在线迁移的影响

存储数据和元数据的缓存限制支持在线迁移的配置。当前，只有 raw、qcow2、qed 的镜像格式支持在线迁移。如果使用的是集群文件系统，所有的镜像格式都支持迁移；如果不是，只有 none 模式支持在线迁移。

Libvirt 会检查在线迁移的几种兼容性因素。如果虚拟机在集群文件系统上，且共享存储被标记为只读模式，缓存模式的检查会被忽略。相反，如果缓存模式不是 none，Libvirt 不会允许在线迁移。但是实际上也可以通过 virsh 命令使用 unsafe 参数，或者使用 API，强制执行在线迁移，例如：

```
virsh migrate --live --unsafe
```

none 模式要求 I/O 方式设置成 aio = native。如果是其他缓存模式，I/O 方式会静默切换为 aio = threads。KVM 虚拟机可以使用 fdatasync() 方式实现在宿主机上刷盘。

2. 各种缓存方式的应用场景

none 模式用于需要虚拟机在线迁移的环境，主要是虚拟化集群。writethrough 模式用于单机虚拟化，这种方式比较保守，一般突然断电或者宿主机故障不会造成数据丢失。测试环境可以考虑 writeback，这种模式性能比较好，但是如果在脏数据没有刷盘的时候宿主机掉电，有可能造成虚拟机数据丢失，脏数据的刷盘周期取决于文件系统的配置，比如 ext4 文件系统，默认的刷盘周期是 5 秒。unsafe 模式性能比较好，但是最不安全，因为一般情况下宿主机不执行虚拟机的刷盘请求，只有在虚拟机的进程结束的时候，才会执行刷盘动作，因此在安装系统的时候可以考虑这种方式。

5.3　磁盘镜像格式

KVM 虚拟机的磁盘镜像从存储方式上看，可以分为两种方式，第一种方式为存储于文件系统上，第二种方式为直接使用裸设备。裸设备的使用方式可以是直接使用裸盘，也可以是使用 LVM 的方式。存于文件系统上的镜像有多种格式，如 raw、cloop、cow、qcow、qcow2、vmdk、vdi 等，经常使用的是 raw 和 qcow2。

5.3.1　QEMU 支持的磁盘镜像格式

QEMU 支持的磁盘镜像格式有以下几种。

- raw：raw 格式是简单的二进制镜像文件，一次性会把分配的磁盘空间占用。raw 支持稀疏文件特性，稀疏文件特性就是文件系统会把分配的空字节文件记录在元数据

1107114476@qq.com
dolphinebook.com

中，而不会实际占用磁盘空间。Linux常用的ext4、xfs文件系统都支持稀疏特性，Windows系统的ntfs也支持稀疏特性。所以在ext4文件系统上，我们一开始创建的raw格式的镜像文件，用ls命令和用du命令看到的大小是不一样的。使用ls命令看到的是分配的大小，使用du命令看到的是实际使用的大小。如果希望实际分配和占用的大小一致，我们可以使用dd命令创建raw的虚拟机镜像文件。

- cloop：压缩的loop格式，主要用于可直接引导优盘或者光盘的一种镜像格式。
- cow：写时复制格式，因为历史兼容性原因还在支持，这种镜像格式不支持Windows虚拟机。
- qcow：第一代的QEMU写时复制格式，被qcow2替代，因为历史兼容性原因还在支持。
- qcow2：第二代的QEMU写时复制格式，支持很多特性，如快照、在不支持稀疏特性的文件系统上也支持精简方式、AES加密、zlib压缩、后备方式。
- vmdk：VMware产品的镜像格式。
- vdi：VirtualBox的镜像格式。
- vpc vhd：微软hyperV镜像格式。
- dmg：Mac磁盘镜像格式。
- nbd：网络块设备。
- parallels：半虚拟化镜像格式。

1. 镜像创建及查看

（1）镜像创建

镜像的创建使用qemu-img create命令，可使用-f参数指定镜像格式，不指定时默认为raw格式。比如创建一个50GB、名字为test的raw格式的镜像，命令为：

```
qemu-img create test 50G
Formatting 'test', fmt=raw size=53687091200
```

创建一个50GB的镜像，格式为qcow2，名字为test.qcow2，命令为：

```
qemu-img create test.qcow2 -f qcow2 50G
Formatting 'test.qcow2', fmt=qcow2 size=53687091200 encryption=off cluster_size=65536
```

（2）镜像信息查看

使用qemu-img info命令可以查看镜像的信息。比如查看刚才创建的镜像，可以使用以下命令：

```
qemu-img info test
image: test
file format: raw
virtual size: 50G (53687091200 bytes)
disk size: 0
qemu-img info test.qcow2
image: test.qcow2
```

1107114476@qq.com
dolphinebook.com

```
file format: qcow2
virtual size: 50G (53687091200 bytes)
disk size: 136K
cluster_size: 65536
```

分别用 ls 和 du 命令，可以看到两种镜像格式在磁盘空间上的差别。

```
ls -lh test*
-rw-r--r-- 1 root root 50G Oct 23 15:56 test
-rw-r--r-- 1 root root 194K Oct 23 15:58 test.qcow2
-rw-r--r-- 1 root root 0 Oct 16 11:19 test.txt
du -h test*
0 test
136K test.qcow2
0 test.txt
```

这是因为在 ext4 的文件系统上创建，ext4 支持稀疏特性，所以 test 的 raw 格式镜像文件，用 ls 命令显示空间是 50GB，用 du 命令查看是 0GB。而 test.qcow2 因为使用的是 qcow2 格式，无论用 ls 和 du 命令查看，都是非常小。

2. QEMU 镜像格式转换

（1）镜像格式转换

使用 qemu-img convert 命令可以转换镜像格式。比如将刚才创建的 test 镜像转换为名为 test1.qcow2 的 qcow2 镜像格式，命令如下：

```
qemu-img convert -p -f raw -O qcow2 test test1.qcow2
    (20.04/100%)
```

-p 是显示转换进度，-f 是指原有的镜像格式，-O 是输出的镜像格式，然后是输入文件和输出文件。

```
ls -lh test*
-rw-r--r-- 1 root root 50G Oct 23 15:56 test
-rw-r--r-- 1 root root 194K Oct 23 16:07 test1.qcow2
-rw-r--r-- 1 root root 194K Oct 23 15:58 test.qcow2
```

通过 ls 命令，可以看到 test1.qcow2 已经非常小了。

```
qemu-img info test1.qcow2
image: test1.qcow2
file format: qcow2
virtual size: 50G (53687091200 bytes)
disk size: 136K
cluster_size: 65536
```

通过 qemu-img info 命令查看，镜像确实已经转换成 qcow2 格式了。

（2）压缩和加密镜像的转换

如果要做镜像压缩和加密，只能使用 convert 方式。将 test1.qcow2 压缩的命令为：

1107114476@qq.com
dolphinebook.com

```
qemu-img convert -c -f qcow2 -O qcow2 test2.qcow2 test3.qcow2
```

只有 qcow2 支持压缩特性，压缩使用 zlib 算法，压缩是块级别的压缩，并且是只读的，就是当压缩的块被重写的时候，是不压缩的。qcow2 镜像的压缩在镜像被传输的时候特别有意义，因为是块级的压缩，要比用 tar 压缩效率高很多。

qcow2 镜像加密使用 aes 算法，将镜像加密的命令为：

```
qemu-img convert -f qcow2 -O qcow2 test2.qcow2 test3.qcow2 -o encryption
Disk image 'test3.qcow2' is encrypted.
password:
```

命令执行的时候，会要求输入密码。通过 qemu-img info 命令会看到镜像确实被加密了。

```
qemu-img info test3.qcow2
image: test3.qcow2
file format: qcow2
virtual size: 50G (53687091200 bytes)
disk size: 136K
encrypted: yes
cluster_size: 65536
```

镜像加密只用于镜像传输，使用的时候，必须通过 convert 方式对镜像转换，以达到解密的目的。如果直接挂载加密镜像到虚拟机，虚拟机会不能正常启动。

镜像格式转换也可以使用 virt-convert 命令，在 5.3.4 节有介绍。

磁盘转换主要用于不同虚拟化产品的虚拟机镜像转化，比如将 VMware 的 vmdk 转换成 KVM 专用的 qcow2 格式。

3. QEMU 镜像快照

下面介绍快照相关操作。

raw 格式不支持快照，只有 qcow2 格式才支持快照，快照使用 qemu-img 的参数 snapshot 管理。

1）快照创建使用 -c 参数和快照的名字。例如创建名为 s1 的快照，使用如下命令。

```
qemu-img snapshot test.qcow2 -c s1
```

2）快照查看，使用 -l 参数。

```
qemu-img snapshot test.qcow2 -l
Snapshot list:
ID TAG VM SIZE DATE VM                          CLOCK
1    s1               0 2014-10-24 08:05:14          00:00:00.000
2    s2               0 2014-10-24 08:05:50          00:00:00.000
```

3）删除快照，使用 -d 参数。

```
qemu-img snapshot test.qcow2 -d s2
```

1107114476@qq.com
dolphinebook.com

4）还原快照，使用 -a 参数。

```
qemu-img snapshot test.qcow2 -a s1
```

5）快照单独提取镜像，可以使用 convert 参数进行转换。

```
qemu-img convert  -f qcow2   -O qcow2   -s s1 test.qcow2 test-s1.qcow2
```


快照的原理是利用写时复制的机制，所以快照对性能有影响，生产环境建议最多创建一次快照。

4. 镜像一致性检查

可以使用 check 参数对镜像进行一致性检查，镜像检查只支持 qcow2、qed、vdi 格式，命令为：

```
qemu-img check [-f format] filename
```

- --output=ofmt 参数可以限制检查结果输出，ofmt 格式是 human 或者 json。
- -r 参数可以强制镜像修复，默认是块级别的修复，-r all 修复所有错误，但是有很多的风险，可能导致镜像不能被使用。

5. 后备镜像差量

（1）后备镜像简介

后备镜像差量是指多台虚拟机共用一个后备镜像，每个虚拟机根据需要，如果是读操作，读取镜像；如果是写操作，才写入自己的镜像文件，类似快照的 copy on write 机制。后备镜像可以是 raw 和 qcow2 格式，差量镜像只能是 qcow2 格式。

使用差量方式的好处：

1）快速生成虚拟机镜像。通过一条命令，瞬间生成虚拟机镜像。

2）节省磁盘空间。多台虚拟机共用一个后备镜像，节省大量空间。

（2）后备镜像的创建

差量镜像创建使用 -b 参数指定后备镜像，其他参数和创建普通镜像是一样的。

```
qemu-img create -f qcow2 -b w2k3-test w2k3-test-01
```

查看创建的镜像，可以看到后备镜像的信息。

```
qemu-img info w2k3-test-01
image:  w2k3-test-01
file format: qcow2
virtual size: 50G (53687091200 bytes)
disk size: 196K
cluster_size: 65536
backing file: w2k3-test
```

1107114476@qq.com
dolphinebook.com

（3）将差量镜像转换成普通镜像

使用 qemu-img convert 命令转换差量 w2k3-test-01 为普通镜像 w2k3-test-01-10，格式为 qcow2。

```
qemu-img convert -f qcow2 -O qcow2 w2k3-test-01 w2k3-test-01-10
qemu-img info w2k3-test-01
image: w2k3-test-01
file format: qcow2
virtual size: 8.0G (8589934592 bytes)
disk size: 122M
cluster_size: 65536
backing file: /data/win2003ent32chs
qemu-img info w2k3-test-01-10
image: w2k3-test-01-10
file format: qcow2
virtual size: 8.0G (8589934592 bytes)
disk size: 3.3G
cluster_size: 65536
```

（4）更换镜像的后备镜像

使用 qemu-img rebase 命令更换后备镜像，-u 参数非安全方式，只是更换镜像。不带参数 -u，会比较旧的后备镜像和新的后备镜像的差异，以新的镜像为准。

```
qemu-img rebase -u -b w2k3-test-01-10 w2k3-test-01
qemu-img info w2k3-test-01
image: w2k3-test-01
file format: qcow2
virtual size: 8.0G (8589934592 bytes)
disk size: 122M
cluster_size: 65536
backing file: w2k3-test-01-10
```

（5）后备镜像合并

将文件改变的内容合并到后备镜像中，主要用于需要更新后备镜像的场景，使用 qemu-img commit 命令。

```
qemu-img commit [-f format] filename
```

（6）后备镜像差量技术生产环境的使用

后备镜像差量生成虚拟机的方式可以快速产生大量的虚拟机，节省磁盘空间，特别适合桌面虚拟化这样的场景。经过测试，对后备镜像的读压力非常小，系统压力主要集中在差量镜像的写压力。使用的时候，为了取得良好效果，应将后备镜像和差量镜像分散到不同的物理硬盘上，并保证镜像的安全性。

使用差量方式的风险：

- 多台虚拟机启动的时候，I/O 压力非常大，尤其是第一次启动的时候。
- 对后备镜像安全性要求非常高。

1107114476@qq.com
dolphinebook.com

6. QEMU 镜像大小修改

qemu-img 命令的 resize 参数用于修改镜像大小，但是这种方式只能修改镜像大小，不能修改文件系统。如果要修改镜像大小的同时修改文件系统，需要使用 guestfish 套件的 virt-resize 命令。

修改 test2.qcow2 镜像的大小，先查看一下镜像目前的大小。

```
qemu-img info test2.qcow2
image: test2.qcow2
file format: qcow2
virtual size: 55G (59055800320 bytes)
disk size: 136K
cluster_size: 65536
```

镜像大小增加 5GB，使用参数 +5G。

```
qemu-img resize test2.qcow2 +5G
Image resized.
```

使用 qemu-img info 命令查一看下效果。

```
qemu-img info test2.qcow2
image: test2.qcow2
file format: qcow2
virtual size: 60G (64424509440 bytes)
disk size: 136K
cluster_size: 65536
```

也可以直接指定镜像修改后的大小。

```
qemu-img resize test2.qcow2 70G
Image resized.
```

使用 qemu-img info 命令查看一下效果。

```
qemu-img info test2.qcow2
image: test2.qcow2
file format: qcow2
virtual size: 70G (75161927680 bytes)
disk size: 136K
cluster_size: 65536
```

qcow2 镜像不支持镜像缩小，如果缩小会报错。

```
qemu-img resize test2.qcow2 50G
This image format does not support resize
```

raw 镜像可以缩小，先通过 qemu-img info 查看一下镜像。

```
qemu-img info test
image: test
```

1107114476@qq.com
dolphinebook.com

```
file format: raw
virtual size: 40G (42949672960 bytes)
disk size: 0
```

使用 qemu-img resize 命令缩小镜像。

```
qemu-img resize test 20G
Image resized.
```

查看缩小效果。

```
qemu-img info test
image: test
file format: raw
virtual size: 20G (21474836480 bytes)
disk size: 0
```

注意，镜像缩小的时候，要先将分区和文件系统缩小，否则会造成数据丢失。

5.3.2 裸设备使用

裸设备的使用可以直接挂载裸盘给虚拟机，也可以挂载 lvm。建议在生产环境中使用 lvm，因为 lvm 可以动态地进行调整，可以做快照，管理起来比较方便。

lvm 的管理使用、动态扩展等资料很多，这里就不赘述了，只介绍一下 lvm 在 KVM 虚拟化中的一些注意事项。

（1）lvm cache 配置

一般给虚拟机挂载裸盘，都是为了获得更高的读 / 写性能，所以都是比较重要的虚拟机才使用裸盘的方式。如果使用 lvm，建议将 lvm cache 关闭，防止宿主机因为突然断电，造成数据丢失。

关闭 /etc/lvm/lvm.conf 文件中 lvm 的写缓存。

```
write_cache_state = 0
```

默认是 1 为开启状态，清空 cache_dir 指向的目录内容，一般是：

```
cache_dir = "/etc/lvm/cache"
```

（2）lvm 快照

在虚拟机运行状态下，建议不要做 lvm 快照，可能会出现数据不一致的情况，尤其是数据库业务的虚拟机，lvm 快照也是采用写时复制的机制，因此生产环境建议制作一次快照，多次快照会影响虚拟机性能。

（3）lvm 的精简模式

从 CentOS 6.4 开始，支持 lvm 精简模式，CentOS 6.4 是预览技术，CentOS 7.1 已经是正式支持的技术。lvm 精简模式的原理为建立一个精简模式的资源池，然后在资源池上划分 lv，lv 可以超用空间，并且是用多少写多少。

1107114476@qq.com
dolphinebook.com

（4）clvm 介绍

clvm 是 lvm 的集群方式，一般用于集群的块设备使用，主要是多了验证功能，抑制两台服务器同时对一个 lv 进行写操作。

5.3.3　KVM 虚拟机镜像管理利器 guestfish

guestfish 套件是镜像管理的“瑞士军刀”，在没有 guestfish 这个套件之前，对虚拟机镜像的处理，要么虚拟机运行起来修改，要么使用下面介绍的办法，总之很不方便。

1. 虚拟机镜像挂载及 Windows 2008 Server 系统自检慢问题的解决办法

下面介绍的挂载办法只适应于块设备镜像的挂载。在 guestfish 出来之前，对镜像的处理，只能是先转化成 raw 格式，raw 再转化成裸格式，然后挂载，非常痛苦。

在 KVM 虚拟机上，经常碰到这样的问题，就是 Windows 2008 Server 的虚拟机重启的时候，磁盘检查需要几个小时，即使新装的系统也是这样。下面我们介绍通过裸设备挂载的方式擦除 Windows 2008 Server 磁盘检查标志的方法，绕过这个问题。

操作方法如下。

第 1 步：在宿主机上安装最新的 ntfs-3g。

```
wget http://tuxera.com/opensource/ntfs-3g_ntfsprogs-2012.1.15.tgz
tar -zxvf ntfs-3g_ntfsprogs-2012.1.15.tgz
cd ntfs-3g_ntfsprogs-2012.1.15
./configure --bindir=/usr/bin
make
make install
```

第 2 步：扩展镜像，将镜像挂载，并清除 NTFS 镜像重启检测信息。

① 使用 losetup -f 命令得到可用的 loop 设备。

```
 losetup -f
/dev/loop0
```

② 挂载镜像。

```
losetup /dev/loop0 /dev/vmVG/ptyyb-webzb-57_vda
```

第 3 步：使用 kpartx 将镜像分区映射，如果 2008 镜像是一个分区，默认会挂载到 /dev/mapper/loop0p1。因为 2008 默认安装的时候会有一个 boot 分区，那么 C 盘就在 /dev/mapper/loop0p2 上，读者可根据自己的实际情况挂载。

```
kpartx -av /dev/loop0
```

第 4 步：清除 NTFS 磁盘镜像检测标志信息，使用 ntfsfix 命令。参数 -b -d 很重要，这两个参数是清除 NTFS 的检测标志信息，只有最新版的 ntfs-3g 支持。

```
ntfsfix -b -d /dev/mapper/loop0p1
```

1107114476@qq.com
dolphinebook.com

第 5 步：分离镜像映射。

```
kpartx -dv /dev/loop0
```

第 6 步：分离镜像挂载。

```
losetup -d /dev/loop0
```

2. guestfish 镜像工具套件

guestfish 是一套虚拟机镜像管理的利器，提供一系列对镜像管理的工具，也提供对外的 API。guestfish 主要包含以下工具，如表 5-1 所示。

表 5-1 guestfish 主要包含的工具

工 具	含 义
guestfish	挂载镜像，并提供一个交互的 shell
guestmount	将镜像挂载到指定的目录
guestumount	卸载镜像目录
virt-alignment-scan	镜像块对齐扫描
virt-builder	快速镜像创建
virt-cat	显示镜像中的文件内容
virt-copy-in	复制文件到镜像内部
virt-copy-out	复制镜像文件出来
virt-customize	定制虚拟机镜像
virt-df	查看虚拟机镜像空间的使用情况
virt-diff	不启动虚拟机的情况下，比较虚拟机内部两份文件的差别
virt-edit	编辑虚拟机内部文件
virt-filesystems	显示镜像文件系统信息
virt-format	格式化镜像内部磁盘
virt-inspector	镜像信息测试
virt-list-filesystems	列出镜像文件系统
virt-list-partitions	列出镜像分区信息
virt-log	显示镜像日志
virt-ls	列出镜像文件
virt-make-fs	镜像中创建文件系统
virt-p2v	物理机转虚拟机
virt-p2v-make-disk	创建物理机转虚拟机 ISO 光盘
virt-p2v-make-kickstart	创建物理机转虚拟机 kickstart 文件
virt-rescue	进入虚拟机救援模式

1107114476@qq.com
dolphinebook.com

（续）

工　具	含　义
virt-resize	虚拟机分区大小修改
virt-sparsify	镜像稀疏空洞消除
virt-sysprep	镜像初始化
virt-tar	文件打包并传入 / 传出镜像
virt-tar-in	文件打包并传入镜像
virt-tar-out	文件打包并传出镜像
virt-v2v	其他格式虚拟机镜像转 KVM 镜像
virt-win-reg	注册表导入镜像
libguestfs-test-tool	测试 libguestfs
hivex	解压 Windows 注册表文件
hivexregedit	合并、导出注册表文件内容
hivexsh	注册表修改交互的 shell
hivexml	将 Window 注册表转化为 .xml
hivexget	得到注册表键值
guestfsd	guestfs 服务

3. guestfish 安装与注意事项

guestfish 套件安装非常简单，一条命令就可以。

```
yum install libguestfs-tools
```

默认安装是不安装 Windows 系统支持的，如果需要修改 Windows 系统镜像，需要再运行如下命令：

```
yum install libguestfs-winsupport
```

CentOS 6.5 以后及 CentOS 7 的 yum 源中没有了 libguestfs-winsupport 这个组件，可以使用如下命令安装 Windows 支持：

```
rpm -ivh \
http://people.redhat.com/~rjones/libguestfs-winsupport/7/7.0/x86_64/libguestfs-
winsupport-7.0-2.el7.x86_64.rpm
rpm -ivh \
http://people.redhat.com/~rjones/libguestfs-winsupport/6/x86_64/libguestfs-
winsupport-1.0-7.el6.x86_64.rpm
```

4. 使用 guestfish 查看虚拟机信息

虚拟机镜像信息查看，主要通过 virt-inspector 和 virt-inspector2 命令。

1107114476@qq.com
dolphinebook.com

```
virt-inspector - Display OS version, kernel, drivers, mount points,
applications, etc. in a virtual machine
```

- virt-inspector 显示 OS 版本、内核、驱动、挂载点、应用，等等。
- virt-inspector2 显示 OS 版本和其他信息。

1）virt-inspector 使用很简单，在命令后面加需要检测的镜像就可以，命令运行后会输出镜像的系统信息。virt-inspector 示例如下：

```
virt-inspector centos5332.qcow2
linux centos i386 5.3 (CentOS release 5.3 (Final)) on /dev/vmvg/root:
  Mountpoints:
    /dev/vmvg/root /
    /dev/sda1 /boot
    /dev/vmvg/swap swap
  Filesystems:
    /dev/sda1:
      label: /boot
      UUID: 152ec813-72cc-4ea1-ad76-71c470feff2f
      type: ext3
      content: linux-grub
    /dev/vmvg/root:
      label:
      UUID: 9462b99e-04d9-4648-9d5d-2ce50d8ed8f2
      type: ext3
      content: linux-root
    /dev/vmvg/swap:
      type: swap
  Modprobe aliases:
  ...
```

也可以通过 --query 输出一些固定内容，方便脚本判断。

```
virt-inspector --query centos5332.qcow2
windows=no
linux=yes
rhel=no
fedora=no
debian=no
fullvirt=yes
xen_domU_kernel=no
xen_pv_drivers=yes
virtio_drivers=yes
kernel_arch=i386
userspace_arch=i386
```

输出也可以是 xml 格式，加 --xml 参数即可。

2）virt-inspector 命令检测 Windows 虚拟机示例如下。

```
indows i386 5.2 (Microsoft Windows Server 2003) on /dev/sda1:
```

1107114476@qq.com
dolphinebook.com

```
  Mountpoints:
    /dev/sda1 /
  Filesystems:
    /dev/sda1:
      type: ntfs
      content: windows-root
  Applications:
```

virt-inspector2 输出为 xml 格式，系统相关信息比 virt-inspector 详细。

3）virt-inspector2 和 virt-inspector 命令类似，输出信息是 xml 文件形式。virt-inspector2 示例如下：

```
virt-inspector2 centos5332.qcow2
<?xml version="1.0"?>
<operatingsystems>
  <operatingsystem>
    <root>/dev/vmvg/root</root>
    <name>linux</name>
    <arch>i386</arch>
    <distro>centos</distro>
    <product_name>CentOS release 5.3 (Final)</product_name>
    <major_version>5</major_version>
...
```

4）virt-inspector2 检测 Windows 虚拟机示例如下：

```
 <arch>i386</arch>
    <distro>windows</distro>
    <product_name>Microsoft Windows Server 2003</product_name>
...
```

5. 使用 guestfish 查看虚拟机分区及文件系统

（1）虚拟机分区及文件系统查看

虚拟机分区及文件系统查看主要使用 3 个命令。

1）列出虚拟机镜像文件分区信息。

```
virt-list-partitions
```

2）列出虚拟机镜像文件系统、分区、块设备、lvm 信息。

```
virt-filesystems
```

3）查看虚拟机镜像分区是否块对齐。

```
virt-alignment-scan
```

（2）分区及文件系统查看示例

1）Linux 镜像查看。

1107114476@qq.com
dolphinebook.com

```
 virt-list-partitions centos6232.qcow2
/dev/sda1
/dev/sda2
 virt-list-filesystems centos6232.qcow2
/dev/sda1
/dev/vmvg/root
```

可以看到镜像 centos6232.qcow2 有两个分区，文件系统挂载的是 /dev/sda1 和 /dev/vmvg/root。

2）Windows 镜像查看。

```
virt-list-filesystems win2k3.qcow2
/dev/sda1
 virt-list-partitions win2k3.qcow2
/dev/sda1
```

可以看到镜像 win2k3.qcow2 有一个分区，文件系统挂载的是 /dev/sda1。

3）检查镜像是否块对齐。

```
virt-alignment-scan -a win2k3.qcow2
/dev/sda1 32256 512 bad (alignment < 4K)
```

从命令结果看，win2k3.qcow2 镜像 sda1 这个分区块没有对齐。

```
virt-alignment-scan -a centos6232.qcow2
/dev/sda1 1048576 1024K ok
/dev/sda2 210763776 1024K ok
```

从命令结果看，centos6232.qcow2 镜像 /dev/sda1、/dev/sda2 块都对齐了。

6. 去掉磁盘空洞——KVM 虚拟镜像的稀疏问题

虚拟机磁盘经常会有大量的空洞，比如 swap 分区、通过 dd 命令占有的磁盘空间等。guestfish 套件中的 virt-sparsify 命令可以消除磁盘镜像中的空洞，方便镜像传输。下面介绍一下 virt-sparsify 的使用方法。

（1）ext4 的稀疏问题

因为 ext4 的稀疏特性，使用不同方式创建的几个镜像文件对比如下。

```
 qemu-img create -f raw test.raw 5G
#qemu-img 创建 raw 格式镜像
 qemu-img create -f qcow2 test.qcow2 5G
#qemu-img 创建 qcow2 格式镜像
 dd if=/dev/zero of=/root/test.dd bs=1G count=5
# 通过 dd 命令创建 raw 镜像
```

通过 ls 和 du 命令进行查看，会发现一件非常有意思的事情。

```
ls -l
-rw-r--r-- 1 root root 5.0G Oct 13 22:46 test.dd
```

1107114476@qq.com
dolphinebook.com

```
-rw-r--r-- 1 root root 193K Oct 13 22:58 test.qcow2
-rw-r--r-- 1 root root 5.0G Oct 13 22:58 test.raw
du test*
5242880 test.dd
196 test.qcow2
0 test.raw
```

从输出结果来看，ls 命令看到两种 raw 镜像都是 5GB，但是通过 du 命令，会发现 dd 命令创建的裸格式是 5GB，Qemu-img 命令创建的 raw 格式是 0GB，为什么会这样呢？这是因为 ext4 文件系统是一个支持稀疏特性的文件系统，当文件内容全部是空的时候，ext4 并不分配实际的空间。

如果想要使用 raw 方式，并希望一次占用完所有的空间，则可以使用 dd 命令创建这样的方式。

（2）qcow2 的稀疏问题

qcow2 方式也有稀疏的问题，在磁盘级别上看，镜像会有大量连续的相同空洞，比如刚创建的镜像的 swap 分区，通过 virt-sparsify 命令可以消除这些空洞，从而缩小磁盘大小，便于我们传输磁盘镜像。

使用 virt-sparsify 前，需要先创建一个新的同样大小的镜像。

```
qemu-img create -f qcow2 test2.qcow2 5G
```

然后使用 virt-sparsify 去除镜像空洞。

```
virt-sparsify  -x   /root/test.qcow2 --convert qcow2 /root/test2.qcow2
```

7. 用 guestfish 操作虚拟机内部文件

（1）虚拟机内部文件管理命令

虚拟机内部文件管理主要使用以下命令。

- ❑ virt-df：镜像磁盘空间使用查看。
- ❑ virt-ls：列出指定目录内文件。
- ❑ virt-cat：显示指定文件内容。
- ❑ virt-edit：编辑指定文件。
- ❑ virt-copy-in：将文件复制到虚拟机内部。
- ❑ virt-copy-out：将虚拟机内部文件复制出来。
- ❑ virt-tar-in：tar 压缩文件复制进虚拟机并解压。
- ❑ virt-tar-out：镜像内指定目录文件复制并压缩。
- ❑ virt-tar：解压或者上传文件到虚拟机。

（2）示例

1）查看镜像磁盘空间。

```
virt-df -a centos6364.qcow2
```

1107114476@qq.com
dolphinebook.com

```
Filesystem 1K-blocks Used Available Use%
centos6364.qcow2:/dev/sda1 198337 32098 155999 17%
centos6364.qcow2:/dev/vmvg/root 16308136 1210172 14269548 8%
```

2）查看镜像 /root 目录内容。

```
virt-ls -a centos6364.qcow2 /root
.bash_history
.bash_logout
.bash_profile
.bashrc
.cshrc
.tcshrc
anaconda-ks.cfg
install.log
install.log.syslog
```

3）显示指定文件内容。

```
virt-cat -a centos6364.qcow2 /root/anaconda-ks.cfg
# Kickstart file automatically generated by anaconda.
...
```

4）编辑指定文件。

```
virt-edit -a centos6364.qcow2 /root/anaconda-ks.cfg
```

5）文件复制进镜像及复制出镜像。

```
touch test.txt
virt-copy-in test.txt -a centos6364.qcow2 /root/
virt-copy-out -a centos6364.qcow2 /root/test.txt /home
```

6）压缩文件复制进虚拟机并解压。

```
virt-tar-in -a disk.img data.tar /destination
```

7）镜像内指定目录文件复制并压缩。

```
virt-tar-out -a disk.img /dir files.tar
```

8）将虚拟机的 home 目录复制出来并打包。

```
virt-tar -x domname /home home.tar
```

9）上传本地的压缩文件到虚拟机并解压。

```
virt-tar -u domname uploadstuff.tar /tmp
```

8. guestfish 修改镜像格式和大小

（1）guestfish 修改镜像格式的命令

修改镜像格式和大小主要使用以下命令。

1107114476@qq.com
dolphinebook.com

❑ virt-convert：转换虚拟机镜像格式。

❑ virt-resize：修改虚拟机镜像磁盘。

（2）示例

1）raw 转 qcow2 格式 Virtio

需要先用 qemu-img 命令创建一个一样大小的空 qcow2 格式镜像文件，然后使用 virt-convert 命令进行转换。

```
virt-convert  -i raw -o qcow2 old.img new.qcow2
```

原始镜像可以是 VMware 镜像 vmx、kvm 镜像、ovf 的镜像。

2）将指定的分区扩大 5GB。

创建一个新的镜像，比原来大 5GB，然后扩展，命令如下：

```
virt-resize --expand /dev/sda2 olddisk newdisk
```

3）将 boot 增加 200MB，剩下的空间扩充给 /dev/sda2。

```
virt-resize --resize /dev/sda1=+200M --expand /dev/sda2 \
olddisk newdisk
```

4）lv 扩展。

```
virt-resize --expand /dev/sda2 --LV-expand /dev/vg_guest/lv_root \
 olddisk newdisk
```

5）扩展分区，并将 raw 格式转换成 qcow2 格式。

```
qemu-img create -f qcow2 newdisk.qcow2 15G
virt-resize --expand /dev/sda2 olddisk newdisk.qcow2
```

注意：

❑ 如果是扩展分区，目标磁盘文件必须大于原生磁盘。

❑ 磁盘缩小比较复杂，一般要求缩小到的空间远大于文件系统的大小。

9. guestfish 挂载、修改、运行救援方式

（1）挂载、修改、运行救援命令

❑ guestmount：挂载镜像到某个目录。

❑ guestfish：挂载镜像并得到一个交互的 shell。

❑ virt-rescue：运行一个镜像的救援模式。

这 3 个命令使用的时候很相似，都是使用 -a 参数指定要处理的镜像。

（2）示例

1）只读方式将镜像挂载到 /mnt 目录。

```
guestmount -a windows.img -m /dev/sda1 --ro /mnt
```

1107114476@qq.com
dolphinebook.com

2）将 Linux 虚拟机的根目录挂载到宿主机的 /mnt 目录。

```
guestmount -a linux.qcow2 -m /dev/sda2  /mnt
```

3）guestfish 编辑镜像 grub 文件。

```
guestfish --rw --add disk.img \
          --mount /dev/vg_guest/lv_root \
          --mount /dev/sda1:/boot \
          edit /boot/grub/grub.conf
```

说明 --rw 参数用可写方式挂载，--mount 参数挂载块设备到目录，edit 命令编辑 grub.conf。

4）进入镜像的救援模式。

```
$ virt-rescue --suggest -d Fedora15
    ...略去部分输出
    #进入救援模式，并可以运行一些命令
      ><rescue> lvs
      LV VG Attr LSize Origin Snap% Move Log Copy% Convert
      lv_root vg_f15x32 -wi-a- 8.83G
      lv_swap vg_f15x32 -wi-a- 992.00M
    #查看lv信息
      ><rescue> mount /dev/vg_f15x32/lv_root /sysroot
      ><rescue> mount /dev/vda1 /sysroot/boot
      ><rescue> ls /sysroot
      ><rescue> chroot /sysroot
      #挂载lv，并修改根分区
```

5.3.4 raw、qcow2、裸盘、lvm 性能比较

几种镜像方式的性能如何，做了一个简单的对比测试。

（1）测试环境

- 宿主机和虚拟机版本：CentOS release 6.6（Final）。
- 磁盘：宿主机——6 块，300GB，RAID 5；虚拟机——100GB raw 硬盘一块，100GB qcow2 磁盘一块，100GB lvm 磁盘一块，所有磁盘选择 Virtio 驱动，虚拟机磁盘缓存方式为 writethough。

（2）测试结果

如图 5-6 所示，4K 读测试 raw 性能最好，qcow2、lvm 次之。虚拟机的读性能比物理机好，是因为磁盘缓存方式为 writethough，读缓存是开启的，所以使用虚拟机的读性能比物理机要好。

如图 5-7 所示，4K 随机读写测试中，裸设备 lvm 性能最好，文件系统镜像 raw、qcow2 性能次之。虚拟机 lvm 方式比物理机性能好，是因为 lvm 还有一层缓存。

1107114476@qq.com
dolphinebook.com

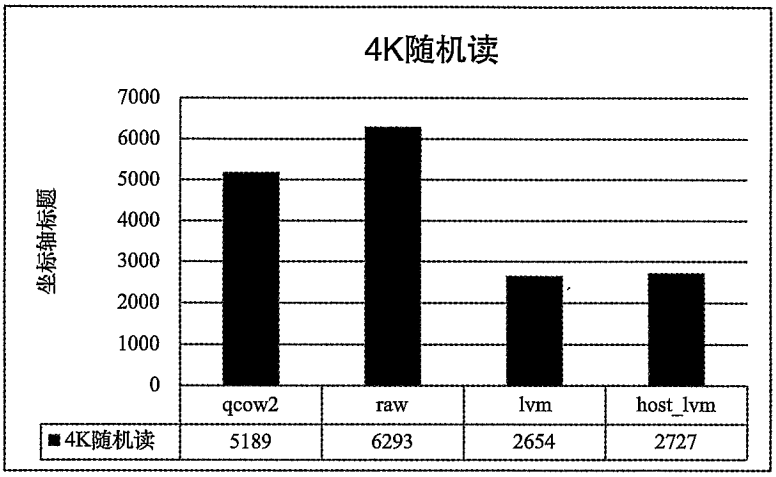


图 5-6　几种镜像方式的 4K 随机写比较

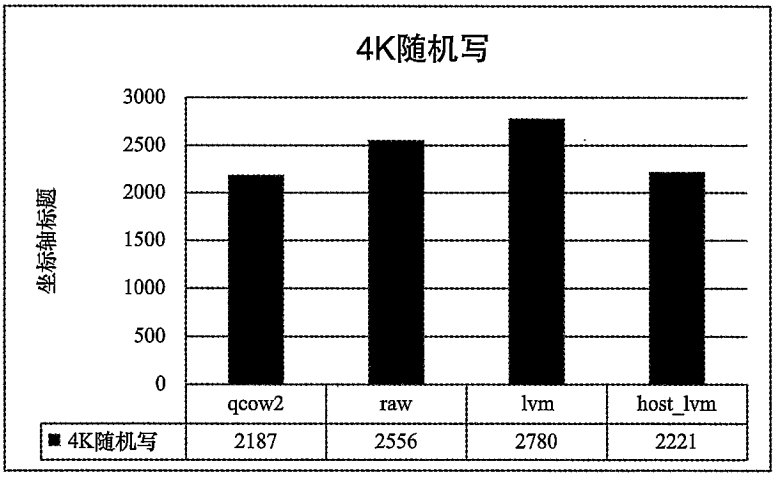


图 5-7　几种镜像方式的 4K 随机写比较

5.3.5　磁盘镜像格式的应用场景

在生产环境中，建议磁盘镜像格式按照如下原则使用：

- CPU 消耗型的虚拟机使用 qcow2 这样的文件镜像。
- 磁盘 I/O 消耗型的虚拟机使用 lvm 这样的裸设备。

CPU 消耗型的虚拟机使用 qcow2 的镜像格式，是因为 qcow2 格式相对性能不如裸设备，但是有许多非常好的特性，比如数据用多少写多少、可以做快照、可以加密、可以动态进行扩展。

磁盘 I/O 消耗型的虚拟机使用 lvm，主要是因为 lvm 是裸设备，读 / 写性能比较好，而且可以动态地进行扩展，也可以做快照。

1107114476@qq.com
dolphinebook.com

5.4 文件系统块对齐

文件系统块对齐也是虚拟化磁盘调优的重要软件手段，并且在低版本的操作系统上效果明显，在生产环境部署低版本操作系统的虚拟机时需要引起注意。

5.4.1 什么是块对齐

Windows 2008 之前的 Windows 系统，2010 年以前的 Linux 系统，第一个分区的扇区是磁盘第 63 扇区，并且扇区尺寸是 512Byte，这个是历史的原因，硬盘必须将 cylinder / head / sector（CHS）信息报告给 BIOS，这个信息在现代的操作系统中是无意义的，但是磁盘依然报告给 BIOS 每个磁盘轨道有 63 个扇区，因此操作系统依然将第一个分区的开始位置放置到第一个磁盘轨道上，从第 63 个扇区开始。

当虚拟化的时候，虚拟机操作系统和虚拟化引擎也许采用以下几种对齐方式。

- 512B 方式：虚拟机操作系统使用本地裸设备，并且裸设备使用 512bit 的扇区。
- 4KB 方式：在新的本地硬盘上使用 4KB 的物理扇区；在基于文件系统的存储方式上使用 4KB 物理扇区；在基于网络的存储方式上使用 4KB 物理扇区。
- 64KB 方式：在高速网络存储上使用，是一些高速网络存储的默认值。
- 1MB 方式：微软从 Windows Server 2008 开始默认采用 1MB 的块对齐方式，如图 5-8 所示。随后 Linux 系统做了跟进。

```
DISKPART> list partition

  分区 ###      类型              大小     偏移量
  ----------   ----------------  -------  -------
* 分区     1    主要               59 GB  1024 KB

DISKPART>
```

图 5-8 Windows Server 2008 默认偏移 1MB

假设虚拟机是 512B 的扇区，会看到 Windows 2008 第一个分区从第 2048 个扇区开始，随后的分区从 2048 倍数个扇区开始。图 5-9 所示为虚拟机和宿主机文件系统块对齐示意图。

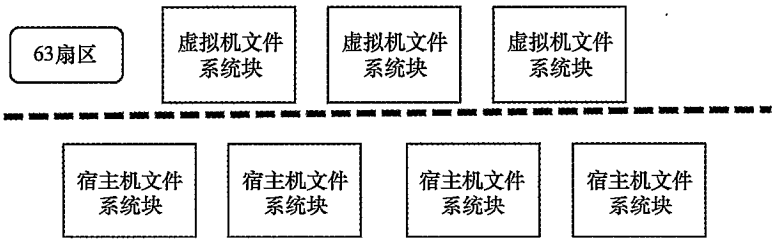


图 5-9 虚拟机和宿主机文件系统块对齐示意图

块对齐的问题如下：

举个例子，在 4KB 扇区的系统上，虚拟机的文件系统块，第 63 扇区横跨物理机文件系统的两个块，会造成一个块的读 / 写操作，横跨两个物理机的块，产生额外的 I/O 开销。

块对齐查看可以使用 virt-alignment-scan 命令。

1107114476@qq.com
dolphinebook.com

```
virt-alignment-scan -a os_images_1
/dev/sda1         32768            32K    bad (alignment < 64K)
virt-alignment-scan -a os_images_2
/dev/sda1         65536            64K    ok
virt-alignment-scan -a centos62x64
/dev/sda1       1048576          1024K    ok
/dev/sda2     210763776          1024K    ok
```

5.4.2　块对齐与块不对齐性能比较

1. 块对齐性能测试

下面我们通过块对齐性能测试来进行性能对比。

（1）测试环境

- 硬件：HP DL380G7 CPU，Intel(R) Xeon(R) E5520 *2 内存，24GB 硬盘，146GB*8 RAID 10。
- 宿主机操作系统：CentOS 6.3 升级到最新内核版本 kernel 2.6.32-279.14.1.el6.x86_64。
- 虚拟机操作系统：Windows 2003 32 中文企业版。
- 测试工具：Iometer。

（2）测试方法

采用 4K 随机写。

提示　如何横向比较磁盘的 I/O 性能？

4K 随机写是一个业界常用的比较指标，意思是 4KB 大小的数据块在磁盘上一秒钟随机写入的次数。同样，4K 随机读也是一个重要的指标，意思是 4KB 大小的数据块在磁盘上一秒钟随机读取的次数。

（3）测试结果

- 块没有对齐的虚拟机：IOPS 165。
- 块对齐的虚拟机：IOPS 187。

如图 5-10 所示，同样环境下，块对齐速度快 13.3%。

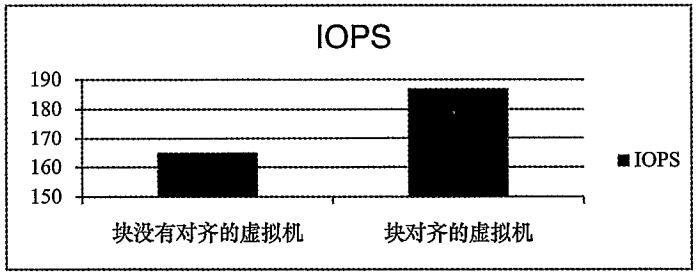


图 5-10　块对齐与块不对齐性能比较

1107114476@qq.com
dolphinebook.com

2. 哪些系统存在块对齐的问题

Windows 系统方面，Windows 2008 Server 之前的版本都存在块对齐的问题，主要是 Windows 2003 Server 和桌面的 Windows XP，Windows 2008 Server 和 Windows 7 以后的系统，默认都会有 1MB 的偏移量，解决了这个问题。

Linux 系统方面，CentOS 6 以前的系统，如 CentOS 5、RHEL 5 都存在块对齐的问题，从 CentOS 6 和 RHEL 6 以后，像 Windows 系统一样，都有一个偏移量，已经解决了这个问题。

5.4.3 生产环境如何配置块对齐

1. Windows 系统

安装的时候，使用 winpe 引导，通过 diskpart 命令先划分分区。

```
diskpart
    select disk 0
    clean

    create partition primary align=64
```

划分分区后，格式化分区，安装操作系统。

2. Linux 系统

使用 kickstart 文件。先在预处理的部分用 parted 分区，然后即可使用分区。

```
part /boot --fstype ext3 --onpart vda1
part pv.2 --onpart vda2
volgroup VolGroup00 --pesize=32768 pv.2
# 划分 vg 名字为 VolGroup00
logvol swap --fstype swap --name=LogVol01 --vgname=VolGroup00 --size=1008 --grow
--maxsize=2016
logvol / --fstype ext3 --name=LogVol00 --vgname=VolGroup00 --size=1024 --grow
# 划分两个 lv，一个是 swap，一个是根
%pre
parted /dev/vda mklabel msdos -s
parted /dev/vda mkpart primary ext3 2048s 409600s -s
# 创建分区，从 2048 扇区开始，避免块不对齐问题
parted /dev/vda mkpart primary 411648s 100% -s
parted /dev/vda set 2 lvm on s
# 按照扇区来划分磁盘空间，每个扇区是 512 字节，一般开始和结束按照 2048 个扇区的倍数来开始
```

3. 使用 virt-resize 命令更改块对齐方式

已经安装好的虚拟机镜像可以使用 virt-resize 命令更改块对齐方式。

```
# 创建一个 qcow2 镜像
qemu-img create kuai-no-vda-2-yes -f qcow2 6G
# 使用 virt-resize 调整镜像，并在调整的时候强制块对齐
```

1107114476@qq.com
dolphinebook.com

```
virt-resize --align-first always --expand /dev/vda1 kuai-no-vda kuai-no-vda-2-yes
# 查看块对齐是否成功
[root@lab-kvm-test178 datapool]# virt-alignment-scan -a kuai-no-vda-2-yes
/dev/sda1 65536 64K ok
```

使用 virt-resize 命令存在一个问题，就是 Windows 系统使用 virt-resize 命令修改磁盘大小之后的虚拟机镜像不能再次使用 virt-resize 命令；但是 Linux 系统可以多次使用 virt-resize 命令修改磁盘大小。

5.5　SSD 在 KVM 虚拟化中的使用实践

固态硬盘（Solid State Disk、Solid State Drive，SSD）是一种以存储器 Flash 作为永久性存储器的存储设备。固态盘常见接口有 SATA 及 PCI-Express 接口的加速卡。

5.5.1　SSD 原理与写放大

1. SLC、MLC、TLC 闪存区别

Flash 的最小存储单元是晶浮栅晶体管，在固态盘中，在存储单元晶体管的栅（Gate）中注入不同数量的电子，通过改变栅的导电性能，改变晶体管的导通效果，实现对不同状态的记录和识别。

能读 / 写两种状态的晶体管的 Flash 的存储单元称为单层式存储（SLC），能读 / 写 4 种状态的晶体管的 Flash 的存储单元称为多层式存储（MLC），能读 / 写 6 种状态的晶体管的 Flash 的存储单元称为三层式存储（TLC），TLC 也称作 3-bit MLC。SLC、MLC 及 TLC 的读 / 写速度依序从快至慢，使用寿命依序从长至短，成本依序从高至低，需要纠错比特数（ECC）则是相反地从低至高。SLC 的状态简单，所以读取很容易。MLC 有多种状态，读取时容易出错，需要校验，速度相对较慢。实际 MLC 的状态识别过程比上述复杂很多，读取一次 MLC 的功耗比 SLC 大很多。由于材料本身的缘故，SLC 可以接受 10 万次级的擦写，而 MLC 材料只能接受万次级擦写操作，所以 MLC 的寿命比 SLC 短很多。但是，也是最重要的，由于 MLC 中的信息量大，同一个存储单元，信息量是 SLC 的 N 倍，所以相同容量的磁盘，MLC 类型 Flach 成本更低，存储单元体积更小，这也导致市面上多数固态盘都采用了 MLC 型的 Flash 颗粒。SLC 由于其特性，仅在高端的高速存储设备中使用。目前固态硬盘的主流从 SLC 芯片转到 MLC 芯片。

Flash 颗粒是大量浮栅晶体管的阵列，一般的 U 盘中会有 1 ~ 2 粒 Flash 颗粒，视容量而定；在 SSD 硬盘中，常见会有更多粒 Flash 颗粒，并且 Flash 颗粒可以同时进行读 / 写。所以一般固态硬盘的容量越大，Flash 的颗粒也就越多，性能也就越好。

1107114476@qq.com
dolphinebook.com

2. SSD 的写放大

SSD 的硬盘有 Page 和 Block 的概念。Page 大小为 4KB，Block 大小为 512KB（即 128 个 Block）。

SSD 一次写的单位是 Page 即 512KB，为什么不像磁盘一样，每次写一个比特呢？主要原因是为了防止电子干扰，保证数据的稳定和准确。

写的时候，SSD 的 Write 只能写到空的 Page 上，对于之前写过的 Page，必须先进行一次 Erase。而 Erase 的单位是 Block，所以如果一个 Page 的数据删掉之后，要想再写到这个 Page 上，必须经过以下三步：

- 将在同一个 Block 中的其他 Page 读出来。
- 将整个 Block 擦除。
- 将整个 Block 的数据写下去。

因为一个写动作要操作三次，严重降低了效率，这个过程叫作 SSD 的写放大，也叫写惩罚。

解决写放大主要有两个方案。

（1）预留空间

一般固态硬盘上都有一块保留空间，消费机级的 SSD 为 7% ~ 10%，企业级的 SSD 保留空间在 20% 以上，甚至有些产品有 100% 的保留空间。使用保留空间的好处是，随时都能保证有未使用的空间，减少写放大，SSD 的芯片在空闲的时候再对删除的块进行清除。所以写入频繁的时候，保留空间越多的 SSD 性能越好。

（2）使用 TRIM 技术

TRIM 是现在公认的解决写放大的比较好的方案。TRIM 位于操作系统层。操作系统使用 TRIM 命令来通知 SSD 某个 Page 的数据不需要了，可以回收了。

支持 TRIM 的操作系统和以往的主要区别是删除一个 Page 的操作不同。在磁盘时期，删除一个 Page，之后在文件系统的记录信息里将该 Page 的标志位设置为可用，但是并没有将数据删除。使用 SSD 且支持 TRIM 的操作系统，在删除一个 Page 时，会同时通知 SSD 这个 Page 的数据不需要了，SSD 内部有一个空闲时刻的垃圾收集进程，在空闲时刻 SSD 会将一些空闲的数据集中到一起，然后一起 Erase。这样每次写操作，就在已经 Erase 好了的 Page 上写入新的数据。越是新的操作系统对 Trim 特性支持得越好。

5.5.2 SSD 在 KVM 中的使用

要使用 SSD，需要先对 SSD 在 KVM 虚拟化中使用进行测试，建立数据模型，根据模型再确定最好的使用方式及使用中要注意的地方。本节是笔者做过的基于 Intel 320 SSD 在 KVM 上做的测试，320 虽然是上一代的 SSD，但是测试结论在新一代的 SSD 上依旧有效。

1107114476@qq.com
dolphinebook.com

1. 测试环境

（1）硬件

1）测试服务器一。

❑ HP DL380G7：内存 24GB，CPU 是 Intel(R) Xeon(R) E5620，@ 2.40GHz。

❑ Intel 320 160GB *8 RAID 10。

2）测试服务器二。

❑ 联想 510G7：内存 8GB，CPU 是 Intel(R) Xeon(R) E5506，@ 2.13GHz。

❑ Intel 320 300GB *2 RAID 1。

3）测试服务器三。

❑ HP DL380G6：内存 16GB，CPU 是 Intel(R) Xeon(R) E5520，@ 2.27GHz。

❑ Intel 320 160GB *4 RAID 10。

（2）软件

宿主机系统：CentOS 6.3 内核版本 2.6.32-279.22.1.el6.x86_64。

测试工具：所有 Linux 系统使用 fio，所有 Windows 系统使用 Iometer。

（3）测试准备工作

为保证测试准确性，做了以下工作。

1）关闭所有测试服务器的 lvm 缓存。

```
Vim /etc/lvm/lvm.conf
write_cache_state = 0
sed -i s/write_cache_state\ =\ 1/write_cache_state\ =\ 0/g /etc/lvm/lvm.conf
rm /etc/lvm/cache/* -rf
```

2）拔掉所有测试服务器的 RAID 卡电池。

3）修改内核的缓存机制，定时清空缓存。

编辑 crontab，每分钟执行以下命令：

```
echo 3 >> /proc/sys/vm/drop_caches
echo 1 > /proc/SCSI/sg/allow_dio
```

4）因为 SSD 有写放大问题，先将 SSD 写满，再进行测试。所有 SSD 测试前，先使用脚本反复写满数据 10 次。

```
for (( i=0; i<10; i++))
{
echo $i;
dd if=/dev/zero of=/dev/datavg/sdc1 bs=10M;
}
```

2. 测试结果

（1）物理机上 5 种使用方式最大性能测试结果

这 5 种方式的性能测试结果如表 5-2 和图 5-11 所示。

1107114476@qq.com
dolphinebook.com

表 5-2 SSD 物理机上 5 种使用方式最大性能测试结果

使用方式 / 测试内容	HP 4 块 160GB RAID 10	HP 2 块 160GB RAID 1	联想 2 块 300GB RAID 1	Gluster 每节点 4 块 160GB RAID 10 复制卷	drbd 每节点 4 块 160GB RAID 10
4K 顺序读 IOPS	20 745	19 479	23 943	15 260	7975
4K 顺序写 IOPS	5100	4733	6173	788	3358
4K 随机读 IOPS	54 745	46 092	24 131	14 029	31 097
4K 随机写 IOPS	4516	3288	3563	760	2826

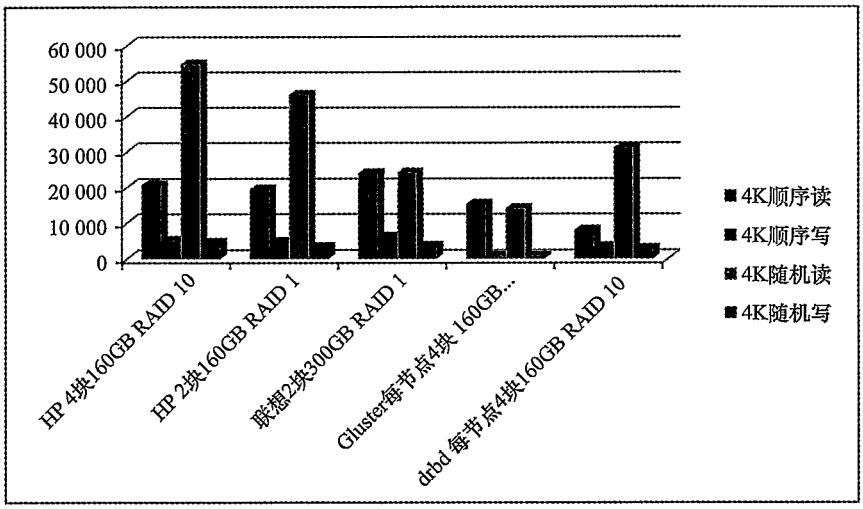


图 5-11 SSD 物理机上 5 种使用方式最大性能测试结果对比

从图 5-11 中可以得出以下结论：

- SSD 读性能好于写性能。
- 容量越大的 SSD，性能越好。

（2）不同系统的虚拟机性能测试对比

1）HP 服务器，宿主机上 4 块（160GB 硬盘）SSD 做 RAID 10，然后测试这台宿主机上不同操作系统的虚拟机性能，结果如表 5-3 和图 5-12 所示。

表 5-3 SSD RAID10 不同操作系统的性能结果

测试内容	文件系统 4K 随机读 IOPS	文件系统 4K 随机写 IOPS	Lv 4K 随机读 IOPS	Lv 4K 随机写 IOPS
Windows 2003	37 684	204	15 674	3776
Windows 2008	25 748	239	15 876	3353
CentOS 5.6	31 124	2440	44 377	2784
CentOS 6.2	35 582	1879	44 842	2918

1107114476@qq.com
dolphinebook.com

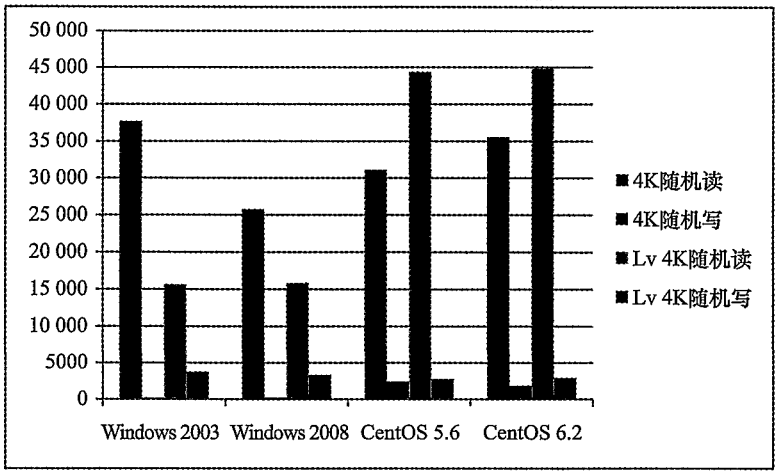


图 5-12　SSD RAID 10 不同操作系统的性能对比

2）联想服务器，宿主机上两块 SSD 做 RAID 1，然后测试这台宿主机上不同操作系统的虚拟机性能，结果如表 5-4 和图 5-13 所示。

表 5-4　联想 2 块 RAID 1 虚拟机测试

测试内容	4K 随机读	4K 随机写	Lv 4K 随机读	Lv 4K 随机写
Windows 2003	26 256	288	9747	2237
Windows 2008	21 075	293	10 629	2643
CentOS 5.6	23 504	2339	2734	25 875
CentOS 6.2	20 922	2481	26 677	3064

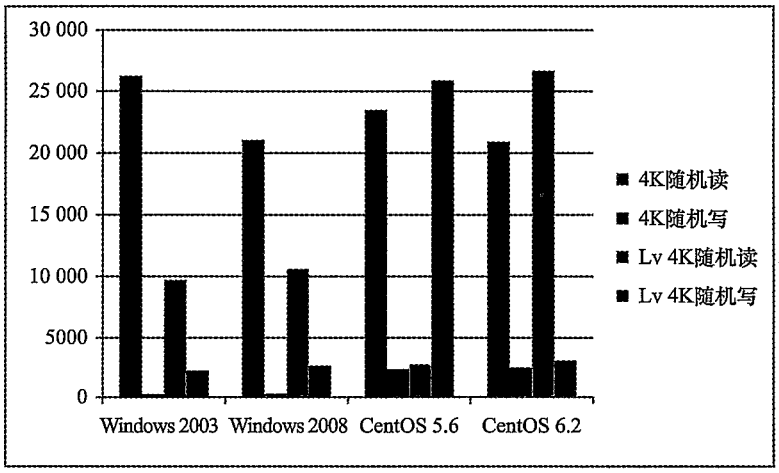


图 5-13　SSD RAID 1 不同操作系统的性能对比

1107114476@qq.com
dolphinebook.com

从图 5-12 和图 5-13 中可以得出以下结论：

- 基于文件系统，同样条件下 Linux 系统虚拟机性能好于 Windows 系统虚拟机。
- 虚拟机使用 SDD，基于 Lv 的裸设备磁盘，性能好于基于文件系统的磁盘。

（3）分布式文件系统上虚拟机性能测试

初步测试了使用 SSD 搭建 GlusterFS 和 DRBD 两种分布式文件系统下虚拟机的使用效果（分布式文件系统在虚拟化中的使用在第 9 章、第 10 章有详细介绍）。测试结果如表 5-5 所示。

表 5-5 GFS 和 DRBD 上虚拟机性能对比测试结果

测试内容	4K 随机读	4K 随机写
GFS Windows 2003	29 203	50
DRBD Windows 2003	3148	826

图 5-14 所示是 GFS 和 DRBD 上虚拟机性能对比测试。

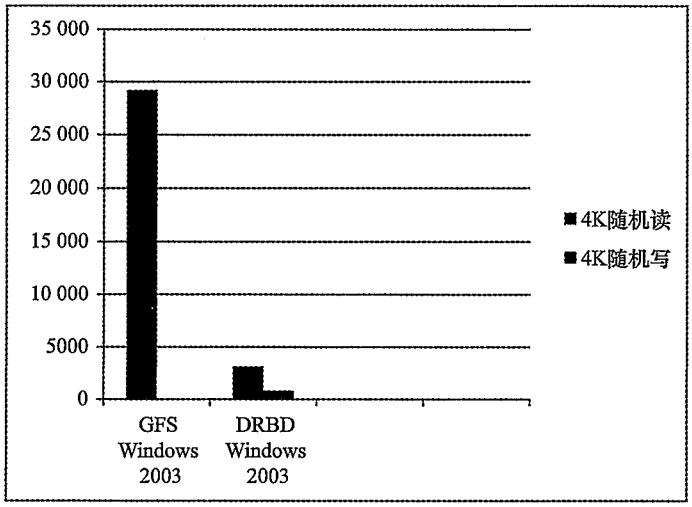


图 5-14 GFS 和 DRBD 上虚拟机性能对比测试

从图 5-14 中可以得出以下结论：

- 同物理机 SSD 性能相比，虚拟机性能相差很大。
- 测试条件下，GlusterFS 性能好于 DRBD。

3. 测试结论

综合上面的测试结果，可以得出以下结论：

- 使用 SSD 在读方面可以达到数万以上的 IOPS，在写方面可以达到数千左右的 IOPS，相对机械硬盘有很大的优势。
- 在虚拟化中，为了充分发挥 SSD 性能，虚拟机磁盘应该使用裸设备映射，而不是 qcow2 格式的文件系统。

1107114476@qq.com
dolphinebook.com

5.5.3　Flachcache 配置与性能测试

1. Flashcache 介绍

Flashcache 是一款针对 Linux 系统的块设备缓存的工具，可以编译为内核模块，使用 Device Mapper 实现。原理是用高速设备做缓存，把数据先缓存到高速设备上，到一定阈值，再写到低速设备上。因为高速设备一般比较昂贵，Flashcache 的目标是综合高速设备和低速设备的优势。

Flashcache 支持 writeback、writethrough、writearound、writeonly 4 种缓存方式。

（1）writethrough 缓存方式

writethrough 是一种比较安全的缓存方式，所有的写操作写到 SSD 并且立即写到机械硬盘，所有的读操作都被缓存。

（2）writearound 缓存方式

writearound 是一种非常安全的缓存方式，数据不写到 SSD，直接写到机械硬盘，所有的读操作都被缓存。

（3）writeback 缓存方式

writeback 速度快，但是牺牲一定的安全，数据只写到 SSD，并按照一定的策略刷到机械硬盘，所有的读操作都被缓存。

（4）writeonly 缓存方式

writeonly 是 writeback 的一个变种，只缓存写，读不缓存。

writethrough 和 writearound 缓存方式不受设备移除或者重启影响。writeback 缓存会受设备移除或者重启影响。

2. Flashcache 安装与使用

（1）安装

Flashcache 官方网址为 https://github.com/facebook/flashcache。安装方式如下：

```
wget https://github.com/facebook/flashcache/archive/master.zip
unzip master.zip
cd flashcache-master/
cd src/
make
make install
```

（2）使用

flashcache_ 命令主要有以下几个。

- ❑ flashcache_create：创建一个新的 Flashcache 卷。
- ❑ flashcache_destroy：删除 Flashcache 卷。
- ❑ flashcache_load：加载 Flashcache 卷。

1107114476@qq.com
dolphinebook.com

3. Flashcache 性能测试

（1）测试环境

1）硬件。

测试服务器：

- 联想 510G7：内存 8GB，CPU 是 Intel(R) Xeon(R) E5506，@ 2.13GHz。
- Sas 146*3 RAID 0。
- Intel SSD 3700 200GB *1 RAID 0。

2）软件。

- 宿主机系统：CentOS 6.3，内核版本 2.6.32-279.22.1.el6.x86_64。
- 测试工具：所有 Linux 系统使用 fio，所有 Windows 系统使用 Iometer。

（2）测试准备工作

同 5.5.2 节 SSD 测试。

（3）配置

用 sdc 做 sdb 的缓存。

```
# flashcache_create -p back  cachedev /dev/sdc /dev/sdb
cachedev cachedev, ssd_devname /dev/sdc, disk_devname /dev/sdb cache mode WRITE_BACK
block_size 8, md_block_size 8, cache_size 0
Flashcache metadata will use 1192MB of your 24016MB main memory
```

查看运行状态。

```
# flashstat
================================================================================
========================
Flashstat: a tool for flashcache status per second
Author   : NinGoo(seaman.ning@gmail.com)
Version  : 0.3
================================================================================
========================
SSD Device:  /dev/sdc   Disk Device:  /dev/sdb    Cache Mode: WRITE_BACK
Capacity:  303998M     Block Size:   4K   Meta Block Size:      4096b
Total Blocks:   77823488    Cached Blocks:    9   Cached Percent:     0
Set Numbers:    512    Dirty Blocks:     0       Dirty Percent:       0
cache_all:    1    reclaim_policy:     FIFO    dirty_thresh_pct:       20
max_clean_ios_set:    2 max_clean_ios_total:    4  skip_seq_thresh:   0K
time  read/s write/s diskr/s diskw/s  ssdr/s  ssdw/s uread/s uwrit/s metaw/s
clean/s  repl/s wrepl/s   hit%  whit% dwhit%
03-13 14:51:20       0       0       0       0       0       0       0       0
                     0       0       0       0   0|10    0|0     0|0
```

使用命令 fdisk -l /dev/mapper/cachedev，可以看到一个裸设备。

```
Disk /dev/mapper/cachedev: 146.8 GB, 146778685440 bytes
255 heads, 63 sectors/track, 17844 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

1107114476@qq.com
dolphinebook.com

```
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0002da0a
```

提示　Flashcahce 可以非常细致地控制读 / 写比例、脏数据刷新周期等参数，读者可以阅读官方文档。

（4）使用

可以当一个块设备来使用，划分 pv、vg、lv。

```
# pvcreate /dev/mapper/cachedev
  Physical volume "/dev/mapper/cachedev" successfully created
# vgcreate ssdcachetest /dev/mapper/cachedev
  Volume group "ssdcachetest" successfully created
# lvcreate -L 100G -n ssdcache_test_lv1 ssdcachetest
  Logical volume "ssdcache_test_lv1" created
```

（5）删除缓存设备

```
dmsetup  remove cachedev
flashcache_destroy /dev/sdc
```

（6）系统重启恢复

系统重启恢复使用 flashcache_load 命令重新装载。

语法为：

```
flashcache_load ssd 设备名字 缓存卷
```

例如：

```
flashcache_load /dev/sdd cachedev
```

也可以添加服务为 RHEL/CentOS 服务，经过笔者的 CentOS6.5 测试通过。

1）复制源代码目录下的 flashcache 脚本到 /etc/init.d。

```
cp utils/flashcache  /etc/init.d/flashcache
```

2）给脚本添加执行属性。

```
chmod +x /etc/init.d/flashcache
```

3）根据自己的环境编辑文件，加入 SSD、缓存设备、挂载点等信息。

4）注册服务。

```
chkconfig --add /etc/init.d/flashcache
```

（7）测试结果

测试结果如表 5-6 所示。

1107114476@qq.com
dolphinebook.com

表 5-6 SAS 硬盘和 SSD 硬盘 Flashcache 性能对比测试

测试内容	4K 随机读	4K 随机写	1M 随机读	1M 随机写
SAS 146GB *3 RAID 0	1751	297	121	21
SSD 200GB*1 RAID 0	89 572	22 775	192	133
Flashcache	24 517	5742	165	117

HP 4 块 RAID 10 虚拟机测试的结果如表 5-7 所示。

表 5-7 不同系统的虚拟机性能测试对比

测试内容	4K 随机读	4K 随机写	Lv 4K 随机读	Lv 4K 随机写
Windows 2003	5709	3176	40	54
Windows 2008	6692	4414	42	65
CentOS 5.6	12 945	1910	51	35
CentOS 6.2	12 030	2497	59	51

（8）测试结论

- 使用 Flashcache 在读 / 写方面可以达到数千 IOPS，是一种性价比比较高的 SSD 和机械硬盘组合的方式。
- 在虚拟化中，为了充分发挥 Flashcache 的性能，虚拟机磁盘应该使用裸设备映射，而不是 qcow2 格式的文件系统。

5.5.4 DM-cache 配置

DM-cache 是一个 device-mapper 级别的、用高速存储设备做低速存储设备缓存的解决方案，主要应用就是用 SSD 做机械硬盘的缓存，达到的性能和容量之间的平衡。

（1）测试环境

1）硬件：

- HP 380 G7。
- Intel(R) Xeon(R) CPU，X5650，@ 2.67GHz *2 24GB 内存，146GB 硬盘 *2，RAID 1 安装系统。
- 600GB*4 SAS 硬盘，RAID 10，划分名为 sasvg 的 vg 作为低速设备。
- Intel 3500 SSD 300GB /dev/sdd 裸盘作为高速设备。

2）软件：CentOS 7 kernel 3.10.0-123.13.2.el7.x86_64。

（2）DM-cache 的主要创建步骤

1）先确定要使用的 SSD 缓存的大小，使用 blockdev --getsize64 命令。

```
blockdev --getsize64 /dev/sdd
300035497984
```

假设要将 SSD 全部用作缓存，SSD 容量是 300GB，通过命令取得的大小是 300 035 497 984 字节。

1107114476@qq.com
dolphinebook.com

2）确定 metadata 的大小。

metadata 大小的计算公式是：4MB+（16Bytes × nr_blocks）。

nr_blocks 就是设备块数量。默认缓存的块大小是 256KB，即 262 144 字节，那么元数据大小计算方法是：

```
4194304 + (16*300035497984/262144)=22507017.5
```

换算成扇区：

```
22507017.5/512=43959.0185546875
```

近似等于 43 960 字节。

3）创建 matadata 分区。

```
dmsetup create ssd-metadata --table '0 43960linear /dev/sdd 0'
dd if=/dev/zero of=/dev/mapper/ssd-metadata
```

4）计算剩余的扇区。

```
300035497984/512-43960=585962872
```

5）创建 SSD 缓存设备。

使用元数据剩余的空间。

```
dmsetup create ssd-blocks --table '0 585962872 linear /dev/sdd 43960'
```

6）确定低速设备的扇区数。

```
blockdev --getsz /dev/sasvg/saslv1
629145600
```

7）创建缓存设备，并且是 256KB 的缓存块大小。

```
dmsetup create home-cached --table '0 629145600  cache /dev/mapper/ssd-metadata
/dev/mapper/ssd-blocks /dev/sasvg/saslv1 512 1 writeback default 0'
```

8）校验设备。

```
ls -l /dev/mapper/home-cached
lrwxrwxrwx 1 root root 7 Jan  6 00:52 /dev/mapper/home-cached -> ../dm-8
dmsetup status /dev/mapper/home-cached
0 629145600 cache 8 2272/5495 512 4/1144458 72 19 0 0 0 4 0 1 writeback 2 migration_
threshold 2048 mq 10 random_threshold 4 sequential_threshold 512 discard_
promote_adjustment 1 read_promote_adjustment 4 write_promote_adjustment 8
```

9）使用设备。

```
mkdir /cache
mkfs.ext4 /dev/mapper/home-cached
#格式化并挂载
mount /dev/mapper/home-cached /cache
```

1107114476@qq.com
dolphinebook.com

```
df -h
/dev/mapper/home-cached  296G   65M  281G   1% /cache
```

10）删除设备。

```
umount /dev/mapper/home-cached
#先卸载
#删除设备
dmsetup remove home-cached
dmsetup remove ssd-metadata
dmsetup remove ssd-blocks
```

5.5.5 LVM cache 配置与性能测试

（1）LVM cache 简介

LVM cache 是在 DM-cache 的基础上基于 lvm 做的配置，也是一种通过 SSD 做分层的方案。

（2）LVM cache 的配置方法

LVM cache 的配置方法如下，测试环境同 5.5.4 节的环境。

1）创建慢速的 lv。

```
lvcreate -L 300G -n slowlv sasvg
mkfs.xfs /dev/sasvg/slowlv
#格式化
mkfs.xfs /dev/sasvg/slowlv
meta-data=/dev/sasvg/slowlv      isize=256    agcount=4, agsize=19660800 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=0
data     =                       bsize=4096   blocks=78643200, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal log           bsize=4096   blocks=38400, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

2）将 SSD 磁盘加入到 sasvg 里面。

```
vgextend sasvg /dev/sdd
```

3）创建元数据 lv 和缓存 lv。

```
lvcreate -L 2G -n  metadata sasvg /dev/sdd
lvcreate -L 270G -n  cache sasvg /dev/sdd
#查看 lv
lvs
 LV     VG     Attr       LSize   Pool Origin Data%  Move Log Cpy%Sync Convert
  cache    sasvg  -wi-a-----  270.00g
  metadata sasvg  -wi-a-----    2.00g
  slowlv   sasvg  -wi-a-----  300.00g
```

4）创建缓存池。

```
lvconvert --type cache-pool --poolmetadata sasvg/metadata --cachemode writeback
```

1107114476@qq.com
dolphinebook.com

```
sasvg/cache
  Logical volume "lvol0" created
```

--cachemode writeback 使用回写模式，不声明默认是 Writethrough。

5）将慢速 lv 加入缓存池。

```
lvconvert --type cache --cachepool sasvg/cache sasvg/slowlv
lvs #查看 lv
  LV     VG     Attr       LSize   Pool  Origin        Data%  Move Log Cpy%Sync Convert
  cache  sasvg  Cwi-a-C--- 270.00g
  slowlv sasvg  Cwi-a-C--- 300.00g cache [slowlv_corig]
```

6）移除 LVM cache。

```
lvconvert --uncache VG/CacheLV
#移除的时候会自动刷盘
lvremove /dev/sasvg/cache
  Flushing cache for slowlv
  0 blocks must still be flushed.
Do you really want to remove active logical volume cache? [y/n]:
```

（3）SSD 分层性能对比测试

笔者做了一个简单的 4K 随机写的测试，对比 SSD 直接使用、Flashcache、DM-cache、LVM cache 4 种方式的性能，结果如图 5-15 所示。

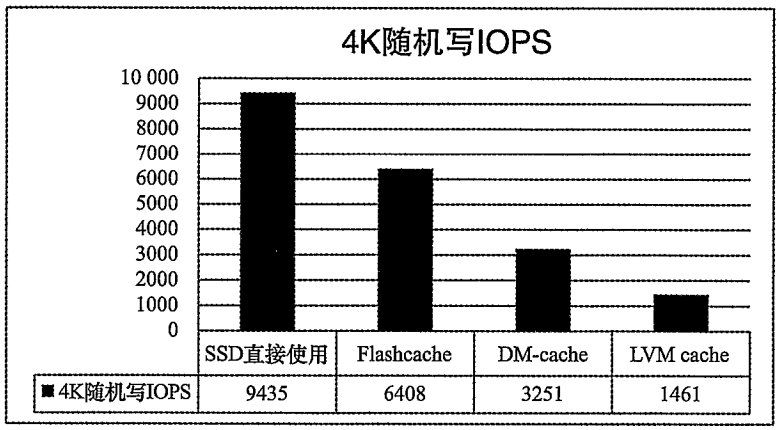


图 5-15　SSD 分层对比测试

如图 5-15 所示，直接使用 SSD 是性能最好的，其次是 Flashcache、DM-cache、LVM cache。这个性能和软件成熟度也有关系，Flashcache 推出最早，使用也最多，DM-cache 是在内核 3.9 才加入的特性，LVM cache 技术也是比较新的技术。

5.5.6　生产环境中 SSD 使用要点

SSD 的使用寿命要比机械硬盘短很多，因此对 SSD 的寿命监控就比较重要。SSD 的寿

1107114476@qq.com
dolphinebook.com

命监控主要是通过读取 SSD 硬盘的 SMART 信息。本节介绍一下 SSD 硬盘 SMART 信息的读取和分析方法。

（1）Windows 系统

Windows 系统下不带 RAID 卡，可以使用 Intel 的专用工具，界面如图 5-16 和图 5-17 所示。

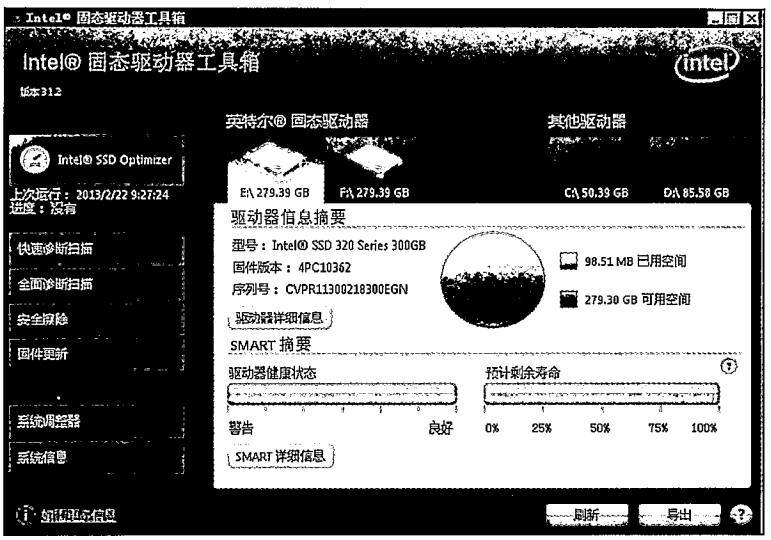


图 5-16 Windows 下 Intel SSD 状态查看工具

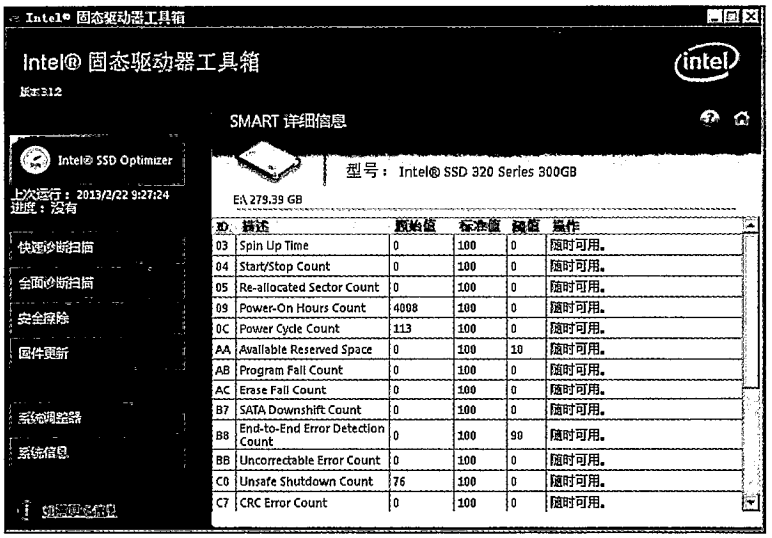


图 5-17 使用 Intel SSD 状态查看 SSD SMART 信息

这种方法只能监控 Windows 服务器，并且不能有 RAID 卡，因为 RAID 卡会将硬盘的 SMART 信息屏蔽。

1107114476@qq.com
dolphinebook.com

（2）Linux 系统

可以透过 RAID 卡读取硬盘的 SMART 信息。各种 RAID 卡的机器，SMART 信息读取方法如下。

1）低端板载的 LSI MPT RAID 卡，SMART 信息读取使用 smartctl 命令，方法如下。

第一块硬盘读取：

```
Smartctl -a -i /dev/sg0
```

第二块硬盘读取：

```
Smartctl -a -i /dev/sg1
```

2）LSI MegaRAID 卡（主要是 IBM、联想、华为、浪潮、戴尔等服务器），SMART 信息读取方法如下。

第一块硬盘：

```
Smartctl -a -d megaRAID,0 /dev/sda
```

第二块硬盘：

```
Smartctl -a -d megaRAID,1 /dev/sda
```

3）Hp 服务器 RAID 卡，SMART 信息读取方法如下（需要 CentOS 6.3 以上的版本）。

第一块：

```
Smartctl -d sat+cciss,0 -a /dev/sda
```

第二块：

```
Smartctl -d sat+cciss,1 -a /dev/sda
```

（3）SSD SMART 信息解读

图 5-18 所示是从 Intel 320 上读取的 SMART 信息。

其中比较关注的有以下 4 点。

1）Media_Wearout_Indicator：使用耗费，100 为没有任何耗费；表示 SSD 上 NAND 的擦写次数的程度，初始值为 100，随着擦写次数的增加，开始线性递减，递减速度按照擦写次数从 0 到最大的比例。一旦这个值降低到 1，就不再降了，同时表示 SSD 上面已经有 NAND 的擦写次数到达了最大次数。这个时候建议备份数据，以及更换 SSD。

2）Reallocated_Sector_Ct：出厂后产生的坏块个数，初始值为 100；如果有坏块，从 1 开始增加，每 4 个坏块增加 1。

3）Host_Writes_32MiB：已写 32MB，每写入 65 536 个扇区 raw value 增加 1。这个扇区还是个数量单位，512 字节。比如，这块盘就是 1 284 966 × 65 536 × 512=40 155.187 5GB。

4）Available_Reservd_Space：SSD 上剩余的保留空间，初始值为 100，表示 100%；阈值为 10，递减到 10 表示保留空间已经不能再减少。

1107114476@qq.com
dolphinebook.com

```
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  3 Spin_Up_Time            0x0020   100   100   000    Old_age   Offline      -       0
  4 Start_Stop_Count        0x0030   100   100   000    Old_age   Offline      -       0
  5 Reallocated_Sector_Ct   0x0032   100   100   000    Old_age   Always       -       0
  9 Power_On_Hours          0x0032   100   100   000    Old_age   Always       -       3682
 12 Power_Cycle_Count       0x0032   100   100   000    Old_age   Always       -       63
170 Reserve_Block_Count     0x0033   100   100   010    Pre-fail  Always       -       0
171 Program_Fail_Count      0x0032   100   100   000    Old_age   Always       -       0
172 Erase_Fail_Count        0x0032   100   100   000    Old_age   Always       -       0
183 Runtime_Bad_Block       0x0030   100   100   000    Old_age   Offline      -       0
184 End-to-End_Error        0x0032   100   100   090    Old_age   Always       -       0
187 Reported_Uncorrect      0x0032   100   100   000    Old_age   Always       -       0
192 Unsafe_Shutdown_Count   0x0032   100   100   000    Old_age   Always       -       55
199 UDMA_CRC_Error_Count    0x0030   100   100   000    Old_age   Offline      -       0
225 Host_Writes_32MiB       0x0032   100   100   000    Old_age   Always       -       414422
226 Workld_Media_Wear_Indic 0x0032   100   100   000    Old_age   Always       -       1085
227 Workld_Host_Reads_Perc  0x0032   100   100   000    Old_age   Always       -       83
228 Workload_Minutes        0x0032   100   100   000    Old_age   Always       -       220965
232 Available_Reservd_Space 0x0033   100   100   010    Pre-fail  Always       -       0
233 Media_Wearout_Indicator 0x0032   099   099   000    Old_age   Always       -       0
241 Host_Writes_32MiB       0x0032   100   100   000    Old_age   Always       -       414422
242 Host_Reads_32MiB        0x0032   100   100   000    Old_age   Always       -       2104239
```

图 5-18 Intel 320 上读取的 SMART 信息

提示 目前新一代的服务器都支持从带外管理卡检测硬盘监控状态，也支持对 SSD 健康状态的检测，使用方法在第 21 章有介绍。

（4）SSD 如何在生产环境中使用

根据笔者的经验，SSD 在生产环境中的使用方式有以下几种。

1）直接使用 SSD，采用 SSD 和 SAS 磁盘组合，比如两块 SSD 做 RAID 1，6 块 SAS 硬盘做 RAID 10，高 IOPS 的虚拟机使用 SSD，推荐使用 LVM 裸盘的形式直接挂给虚拟机使用，CPU 消耗型的虚拟机使用 SAS 硬盘，使用 qcow2 的镜像格式。

2）使用 SSD 分层方案，生产环境推荐使用 Flashcache，系统使用 CentOS 6，这种方式是比较经济的一种方式，经过笔者实践，稳定性也不错。LVM cache 目前还是预览技术；DM-cache 对内核版本要求较高，还有待进一步实践。

5.6 本章小结

本章介绍了 KVM 虚拟化中的磁盘技术，在实际的虚拟化实施中，磁盘虚拟化是一个热点和难点。从这几年的趋势来看，qcow2 的磁盘镜像格式性能越来越好，qcow2 的功能也非常齐全，所以 qcow2 已经是许多虚拟化平台的默认选择。解决磁盘性能问题，最有效的方案还是硬件，所以 SSD 这几年也越来越流行，会在生产环境中越来越普及。

在实施虚拟化的时候，一方面希望能获得更高的性能，另一方面还希望对虚拟机的资源能够有所控制，下一章将为读者介绍虚拟机的资源控制。

1107114476@qq.com
dolphinebook.com

第 6 章 Chapter 6

KVM 虚拟机的资源限制

在虚拟化工作中，笔者经常使用 CGroups 限制虚拟机的 I/O 资源，其次是绑定 CPU，本章还会介绍通过 Linux TC、Libvirt、iptables 限制虚拟机网络流量的方法。

本章先介绍 CGroups 的一些概念，6.4 ~ 6.6 节重点介绍如何限制 CPU、网络、磁盘 I/O，内存的限制在第 3 章已经介绍过了，本章就不介绍了。另外，本章中用到的压力测试工具 Netperf、IOmeter、fio 等在第 15 章有详细的介绍。

6.1 哪些场景要做虚拟机的资源限制

一般在生产环境中都要做虚拟机的资源限制，因为在生产环境中不希望因为某种原因造成一台虚拟机的消耗过高，而其他虚拟机没有资源使用被饿死的情况。尤其是当一台宿主机上有好几个业务的虚拟机的时候，配置虚拟机的资源限制就非常必要。配置虚拟机的资源限制，还要考虑到业务突发的情况，需要预留正常业务需求的空间，否则对业务造成影响就得不偿失了。

KVM 虚拟机的资源限制主要是通过 CGroups 去配置，Libvirt 在 CGroups 上包了一层，也可以通过修改 xml 文件去做虚拟机的资源限制。CGroups 是 Linux 内核提供的一种可以限制、记录、隔离进程组（Process Groups）所使用的物理资源（如 CPU、内存、I/O 等）的机制。最初由 Google 的工程师提出，RedHat 在 RHEL 6/CentOS 6 中开始正式支持 CGroups。

1. CGroups 运行机制

CGroups（Control Groups）是这样一种机制，它以分组的形式对进程使用系统资源的行

1107114476@qq.com
dolphinebook.com

为进行管理和控制。也就是说，用户通过 CGroups 对所有进程进行分组，再对该分组整体进行资源的分配和控制。

Linux 系统中的所有进程都是进程 init 的子进程，该进程在引导时由内核执行，并启动其他进程（这些进程会按顺序启动其子进程）。因为所有进程都归结到一个父进程，所以 Linux 进程模式是一个单一层级结构，或者树结构。

另外，init 进程之外的每个 Linux 进程都会继承其父进程的环境和某些属性。

CGroups 与进程在以下方面类似：

- 都是分级的。
- 子 CGroups 会继承其父 CGroups 的某些属性。

根本不同在于某个系统中可以同时存在不同的分级 CGroups。我们可以把 Linux 进程模式看成单一的树型结构，把 CGroups 模式看成是一个或者更多个独立的树型结构，如图 6-1 所示。

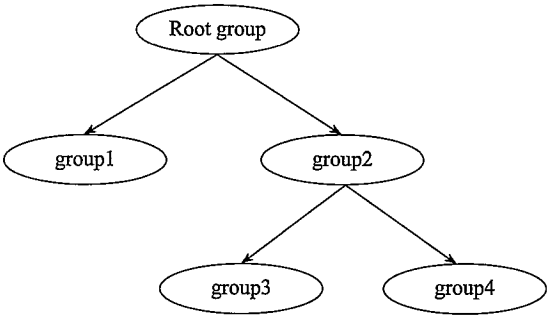


图 6-1 CGroups 树型结构

需要多个独立的 CGroups 分级，因为每个分级都会附加到一个或者多个子系统中。子系统代表单一资源，比如 CPU 时间或者内存。

CGroups 是一种对进程资源管理和控制的统一框架，它提供的是一种机制，而具体的策略（Policy）是通过子系统（Subsystem）来完成的，子系统是 CGroups 对进程组进行资源控制的具体行为。机制和策略是 Linux 操作系统中一种经典的设计思想，所谓机制就是“我要提供哪种功能”，而策略则是“我要怎样来实现这种功能”。RHEL 6 提供了 9 个 CGroups 子系统。CGroups 中每个子系统都代表一种类型的资源，具体如下。

- cpu 子系统：该子系统为每个进程组设置一个使用 CPU 的权重值，以此来管理进程对 CPU 的访问。
- cpuset 子系统：对于多核 CPU，该子系统可以设置进程组只能在指定的核上运行，并且还可以设置进程组在指定的内存节点上申请内存。
- cpuacct 子系统：该子系统只用于生成当前进程组内的进程对 CPU 的使用报告。
- memory 子系统：该子系统提供了以页面为单位对内存的访问，比如对进程组设置内存使用上限等，同时可以生成内存资源报告。
- blkio 子系统：该子系统用于限制每个块设备的输入 / 输出。与 cpu 子系统类似，该系统通过为每个进程组设置权重来控制块设备对其的 I/O 时间；其次，该子系统也可以限制进程组的 I/O 带宽及 IOPS。
- devices 子系统：通过该子系统可以限制进程组对设备的访问，即允许或禁止进程组对某设备的访问。
- freezer 子系统：该子系统可以使得进程组中的所有进程挂起。

1107114476@qq.com
dolphinebook.com

- net-cls 子系统：该子系统提供对网络带宽的访问限制，比如对发送带宽和接收带宽进程限制。
- NameSpaces：名空间子系统。

2. 子系统、层级、控制组群和进程的关系

这里简单说明一下 CGroups 的一些规则，子系统、CGroups 层级及进程之间的关系。

（1）规则 1

任何单一子系统（比如 cpu）最多可附加到一个层级中。结果是，cpu 子系统永远无法附加到两个不同的层级。

（2）规则 2

单一层级可附加一个或者多个子系统。结果是，cpu 和 memory 子系统（或者任意数目的子系统）都可附加到单一层级中，只要每个子系统不再附加到另一个层级即可。

（3）规则 3

每次在系统中创建新层级时，该系统中的所有进程都是那个层级的默认 CGroups 的初始成员。对于创建的任何单一层级，该系统中的每个进程都可以是那个层级中唯一一个 CGroups 的成员。单一进程可以在多个 CGroups 中，只要每个 CGroups 都在不同的层级中即可。只要某个进程成为同一层级中第二个 CGroups 的成员，就会将其从那个层级的第一个 CGroups 中删除。一个进程永远不会同时位于同一层级的不同 CGroups 中。结果是，如果 cpu 和 memory 子系统都附加到名为 cpu_and_mem 的层级中，且 net_cls 子系统附加到名为 net 的层级中，那么运行的 httpd 进程可以是 cpu_and_mem 中任意 CGroups 的成员，同时也是 net 中任意 CGroups 的成员。

httpd 进程所在 cpu_and_mem 中的 CGroups 可将其 CPU 时间限制为分配给其他进程时间的一半，并将其内存用量限制为最多 1024MB。另外，net 中的 CGroups 还可将其传输速率限制为 30MB/s。

首次创建层级时，该系统中的每个进程都至少是一个 CGroups 的成员，即 root CGroups，所以当使用 CGroups 时，每个系统进程至少在一个 CGroups 中。

（4）规则 4

系统中的任意进程都将自己分支创建子进程，该子进程自动成为其父进程所在 CGroups 的成员。然后可根据需要将该子进程移动到不同的 CGroups 中，但开始时它总是继承其父进程的 CGroups 环境。

cpu_and_mem 层级中名为 half_cpu_1gb_max 的 CGroups 成员的进程，以及 net 层级中 CGroups trans_rate_30 的成员。

当 httpd 进程将其自身分成几个分支时，其子进程会自动成为 half_cpu_1gb_max CGroups 和 trans_rate_30 CGroups 的成员。它会完全继承其父进程所属 CGroups。

此后，父进程和子进程就彼此完全独立，更改某个进程的 CGroups 不会影响到另一个进程。同样，更改父进程的 CGroups 也不会影响其子进程。总之，所有子进程总是可以继承其

1107114476@qq.com
dolphinebook.com

父进程 CGroups 的成员关系，但之后便可更改或者删除这些成员的关系。

6.2 使用 CGroups

使用 CGroups 最简单的方法是安装 libcgroups 软件包，该软件包包含大量与 CGroups 有关的命令行工具及其相关 man page。可以使用 shell 命令和在任意系统中使用的工具挂载层级并设定 CGroups 参数。但是，使用 libcgroups 提供的工具可简化过程并扩展功能。因此，本节着重介绍 libcgroups 命令。安装 CGroups 的命令如下：

```
# yum install libcgroups -y
```

1. CGroups 服务

由 libcgroups 软件包安装的 cgconfig 服务可提供创建层级的简便方法，并在层级中附加子系统，且在那些层级中管理 CGroups。建议使用 cgconfig 管理层级和 CGroups。

RHEL 6 默认不启动 cgconfig 服务。当使用 chkconfig 启动该服务时，它读取 CGroups 配置文件 /etc/cgconfig.conf。CGroups 因此会在不同会话间重新创建并保留。根据配置文件的内容，cgconfig 可创建层级、挂载所需文件系统、创建 CGroups 及为每个组群设定的子系统参数。

libcgroups 软件包默认安装的 /etc/cgconfig.conf 文件为每个子系统创建并挂载独立层级，并在这些层级中附加子系统。如果停止 cgconfig 服务（使用 service cgconfig stop 命令），则会卸载它挂载的所有层级。

cgconfig.conf 文件包含两个主要类型的条目：mount 和 group。挂载条目生成并挂载层级，将其作为虚拟文件系统，同时将子系统附加到那些层级中。挂载条目使用以下语法定义：

```
mount {
<controller> = <path>;
...
}
```

2. CGroups 的配置方法

（1）创建挂载条目

以下示例为 cpu 子系统创建层级：

```
mount {
cpu = /cgroups/cpu;
}
```

对等的 shell 命令为：

```
# mkdir /cgroups/cpu
# mount -t cgroups -o cpu cpu /CGroups/cpu
```

1107114476@qq.com
dolphinebook.com

效果如下：

```
# ls /cgroups/cpu
cgroups.event_control  cgroups.procs  cpu.cfs_period_us  cpu.cfs_quota_us  cpu.
rt_period_us  cpu.rt_runtime_us  cpu.shares cpu.stat  notify_on_release  release_
agent  tasks
```

（2）创建组群条目

以下示例为 sql 守护进程创建 CGroups，可为 sqladmin 组群中的用户在 CGroups 中添加进程，并让 root 用户修改子系统参数。

```
group daemons/sql {
perm {
    task {
    uid = root;
    gid = sqladm in;
    } admin {
    uid = root;
    gid = root;
    }
} cpu {
    cpu.shares = 100;
}
}
```

对等的 shell 命令为：

```
~# mkdir -p /CGroups/cpu/daemons/sql
~# chown root:root /CGroups/cpu/daemons/sql /*
~# chown root:sqladmin /CGroups/cpu/daemons/sql/tasks
~# echo 100 > /CGroups/cpu/daemons/sql/cpu.shares
```

必须重启 cgconfig 服务方可使 /etc/cgconfig.conf 中的更改生效。

```
#service cgconfig restart
```

（3）创建层级并附加子系统

要创建层级并在其中附加子系统，需要以 root 编辑 /etc/cgconfig.conf 文件的 mount 部分。mount 部分的条目有以下格式：

```
subsystem = /CGroups/hierarchy;
```

下一次启动 cgconfig 时，它会创建层级并为其附加子系统。

以创建名为 cpu_and_mem 的层级为例，并附加 cpu、cpuset 和 memory 子系统。

```
mount{
cpuset=/cgroups/cpu_and_mem;
cpu=/cgroups/cpu_and_mem;
```

1107114476@qq.com
dolphinebook.com

```
memory=/cgroups/cpu_and_mem;
}
```

对等的shell命令为：

```
# mkdir /cgroups/cpu_and_mem
# mount -t cgroups -o cpuset,cpu,memory cpu_and_mem /cgroups/cpu_and_mem
```

可以使用lssubsys命令列出系统中的所有层级、可用子系统及其当前挂载点。

```
# lssubsys -am
Ns
cpuacct
devices
freezer
net_cls
blkio
perf_event
net_prio
cpuset,cpu,memory /CGroups/cpu_and_mem
```

这里创建了/cgroups/cpu_and_mem层级，并将子系统cpu、cpuset和memory附加到了该层级中，且还没有在任何层级中附加子系统net_cls、ns、cpuacct、devices、freezer和blkio，因为缺少相应的挂载点。

（4）在现有层级中附加、删除子系统

要在现有层级中添加子系统、从现有层级中取消层级或者将其移动到不同的层级中，需要以root编辑/etc/cgconfig.conf文件的mount部分。当cgconfig下次启动时，它会根据指定的层级识别那些子系统。

要在现有层级中取消或修改附加子系统，只需要重新挂载该层级即可。

lssubsys命令显示在cpu_and_mem层级中附加了cpu、cpuset和memory子系统。

```
# lssubsys -am
ns
cpuacct
devices
freezer
net_cls
blkio
perf_event
net_prio
cpuset, cpu,memory /CGroups/cpu_and_mem
```

使用remount选项重新挂载cpu_and_mem层级，并在子系统列表中包含cpuacct：

```
#  mount -t CGroups -o remount, cpuset, cpu,memory, cpuacct cpu_and_mem /
CGroups/cpu_and_mem
```

lssubsys命令显示子系统cpuacct已经附加到cpu_and_mem层级中。

1107114476@qq.com
dolphinebook.com

```
[root@test ~]# lssubsys -am
ns
devices
freezer
net_cls
blkio
perf_event
net_prio
cpuset,cpu, cpu acct,memory /CGroups/cpu_and_mem
```

同样，也可以重新挂载该层级并使用 -o 选项忽略子系统名称。例如，要将 cpuacct 子系统从 /cgroups/cpu_and_mem 层级中取消，只要重新挂载并忽略 cpuacct 即可。

```
# mount -t CGroups -o remount,cpuset,cpu,memory cpu_and_mem /CGroups/cpu_and_mem
```

使用 umount 命令即可卸载 CGroups 中的层级。

```
# umount /CGroups/cpu_and_mem/
# lssubsys -am
cpuset
ns
cpu
cpuacct
memory
devices
freezer
net_cls
blkio
perf_event
net_prio
```

如果该层级目前为空（即它只包含 root CGroups），则在卸载它时会取消激活该层级。如果该层级包含任意其他 CGroups，则该层级在内核中仍保持活跃，即使不再挂载它也是如此。

要删除层级，请确定在卸载该层级前删除所有子 CGroups；或者使用 cgclear 命令，它可在层级非空时取消激活层级。

3. 创建、删除与卸载控制组群

使用 cgcreate 命令可以创建 CGroups 控制组群，通过 man cgcreate 可以看到这样一句话：The command creates new CGroups(s) defined by the options –g，说得很清楚了。

cgcreate 的语法为：

```
cgcreate -t uid:gid -a uid:gid -g subsystems:path
```

其中各选项含义如下。

- -t（可选）：指定用户 UID 和组 GID 以便让这个 CGroups 拥有对 tasks 文件的写权限。这个用户可在该 CGroups 中添加进程。
- -a（可选）：指定用户 UID 和组 GID 以便让这个 CGroups 拥有对 tasks 文件的写权限。

1107114476@qq.com
dolphinebook.com

这个用户可修改这个 CGroups 中的进程对系统资源的访问。

- -g：指定 CGroups 的层级，格式为与那些层级关联的用逗号分开的 subsystems 列表。如果这个列表中的子系统在不同的层级中，则要在每个层级中都创建该组群。层级列表后是一个冒号，然后是与该层级有关的子组群 path。不要在该 path 中包含层级挂载点。

例如，目录 /CGroups/cpu_and_mem/lab1/ 中的 CGroups 称为 lab1——其路径已唯一确定，因为对于给定的子系统最多有一个层级。还请注意该组群可由创建该 CGroups 的现有层级中的所有子系统控制，即使没有在 cgcreate 命令中指定这些子系统。因为同一层级中的所有 CGroups 有相同的控制器，该子组群与其父 CGroups 有相同的控制器。

cgcreate 用法：

```
# cat /etc/cgconfig.conf
mount {
    cpuset  = /cgroups/cpuset;
    cpu     = /cgroups/cpu_and_mem;
    cpuacct = /cgroups/cpuacct;
    memory  = /cgroups/cpu_and_mem;
    devices = /cgroups/devices;
    freezer = /cgroups/freezer;
    net_cls = /cgroups/net_cls;
    blkio   = /cgroups/blkio;
}
# cgcreate -g cpu,memory:/test-subgroup
```

这里创建了名为 test-subgroup 的组群，位于 cpu_and_mem 层级，另外 cpu 和 memory 子系统又附加到了 cpu_and_mem 的层级中，所以 cpu_and_mem 层级中的 test-subgroup 组群由 cpu 和 memory 子系统控制，即使在 cgcreate 命令中没有指定它也是如此。

使用 cgdelete 命令可以删除 CGroups 控制组群，其语法与 cgcreate 类似。运行 cgdelete subsystems:path，其中：

- subsystems 是用逗号分隔开的子系统列表。
- path 是到与该层级相对 root 的 CGroups 的路径。

```
# cgdelete -r cpu,memory:/test-subgroup
```

使用不带任何参数的 cgclear 命令可以清除整个 CGroups 文件系统，并将该 CGroups 中的所有进程重新分配到该层级的 root 节点中；删除所有 CGroups；从该系统中卸载这个文件系统；这样就破坏了所有之前挂载的层级。最后，实际上是删除了挂载该 CGroups 文件系统的目录。

```
# lssubsys -am                    //列出所有可用子系统及其当前挂载点
ns
perf_event
net_prio
```

1107114476@qq.com
dolphinebook.com

```
cpuset /CGroups/cpuset
cpu /CGroups/cpu
cpuacct /CGroups/cpuacct
memory /CGroups/memory
devices /CGroups/devices
freezer /CGroups/freezer
net_cls /CGroups/net_cls
blkio /CGroups/blkio
# ll /CGroups/cpu/test-subgroup/          //test-subgroup 组群
total 0
--w--w---- 1 root root 0 Dec 21 13:52 CGroups.event_control
-rw-rw-r-- 1 root root 0 Dec 21 13:52 CGroups.procs
-rw-rw-r-- 1 root root 0 Dec 21 13:52 cpu.cfs_period_us
-rw-rw-r-- 1 root root 0 Dec 21 13:52 cpu.cfs_quota_us
-rw-rw-r-- 1 root root 0 Dec 21 13:52 cpu.rt_period_us
-rw-rw-r-- 1 root root 0 Dec 21 13:52 cpu.rt_runtime_us
-rw-rw-r-- 1 root root 0 Dec 21 13:52 cpu.shares
-r--r--r-- 1 root root 0 Dec 21 13:52 cpu.stat
-rw-rw-r-- 1 root root 0 Dec 21 13:52 notify_on_release
-rw-rw-r-- 1 root root 0 Dec 21 13:53 tasks
# cgclear                                 //运行 cgclear
# ll /CGroups/cpu/test-subgroup/          //test-subgroup 组群已经被清除
ls: cannot access /CGroups/cpu/test-subgroup/: No such file or directory
 [root@test ~]# lssubsys -am             //可以看到所有的子系统已经被卸载
cpuset
ns
cpu
cpuacct
memory
devices
freezer
net_cls
blkio
perf_event
net_prio
```

4. 设置参数

运行 cgset 命令可以设置子系统参数。例如，如果有 /cgroups/cpuset/test-subgroup 组群，则可以使用如下命令指定这个组群可使用的 CPU：

```
# cgset -r cpuset.cpus=0-1 test-subgroup
# cat /CGroups/cpuset/test-subgroup/cpuset.cpus
0-1
```

上面的命令表示 /cgroups/cpuset/test-subgroup 组群只能使用 CPU 0 和 CPU 1，此时指定虚拟机运行在此组群中，则这台虚拟机只能使用 CPU 0 和 CPU 1。

cgset 的语法为：

```
cgset -r parameter=value path_to_CGroups
```

1107114476@qq.com
dolphinebook.com

其中各选项含义如下。

- parameter 是要设定的参数，该参数与给定 CGroups 的目录中的文件对应。
- value 是为参数设定的值。
- path_to_CGroups 是到相对该层级 root 的 CGroups 路径。例如，如果设定 root 组群的参数（如有 /CGroups/cpuacct/ 文件），请运行：

```
# cgset-r cpuacct.usage=0 /
```

另外，因为 . 与 root 组群相关（即 root 组群本身），还可运行：

```
# cgset -r cpuacct.usage=0 .
```

还可以使用 cgset 命令将一个 CGroups 组群中已经设置好的参数复制到另一个 CGroups 的组群中，这样就可以省去不必要的重复步骤，例如：

```
# cgcreate -g cpuset:/test-subgroup
# cgcreate -g cpuset:/test-subgroup1
# cgset -r cpuset.cpus=0-1 test-subgroup                    //不能大于 CPU 物理核数，假如 CPU 物理核数为 24，如果这里设置 cpuset.cpus=0-31，那么在组群中也只能看到 0-23
# cat /CGroups/cpuset/test-subgroup/cpuset.cpus
0-1
# cat /CGroups/cpuset/test-subgroup1/cpuset.cpus            //此时 cpuset.cpus 没有参数
# cgset --copy-from test-subgroup test-subgroup1
# cat /CGroups/cpuset/test-subgroup1/cpuset.cpus
0-1                    //可以看到 test-subgroup 中的参数已经复制到了 test-subgroup1 中
```

要直接在 CGroups 中设置参数，还可以使用 echo 命令将参数值插入相关子系统的文件中。例如，将 test-subgroup1 组群中的 cpuset. cpus 文件设置为 0-5，对等的 shell 命令为：

```
# cat /cgroups/cpuset/test-subgroup1/cpuset.cpus
0-1
# echo 0-5 > /cgroups/cpuset/test-subgroup1/cpuset.cpus
# cat /cgroups/cpuset/test-subgroup1/cpuset.cpus
0-5
```

5. 将某个进程移动到控制组群中

可以通过运行 cgclassify 命令将进程移动到指定的控制组群中，以达到限制某些进程对系统资源的调度。例如现在要将 httpd 和 mysql 进程移动到 test-subgroup 组群中，并限制这两个进程只能运行在 CPU 0 和 CPU 1 上。

```
# cgcreate -g cpu,memory:/test-subgroup
# cgset -r cpuset.cpus=0-1 test-subgroup
# cat /cgroups/cpuset/test-subgroup/cpuset.cpus
0-1
# ps -ef |grep httpd
root     11002     1  0 18:38 ?        00:00:00 /usr/sbin/httpd
# ps -ef |grep mysql
```

1107114476@qq.com
dolphinebook.com

```
root      8699     1  0 18:18 ?        00:00:00 /bin/sh /usr/bin/mysqld_safe --
datadir=/var/lib/mysql --socket=/var/lib/mysql/mysql.sock --pid-file=/var/run/
mysqld/mysqld.pid --basedir=/usr --user=mysql
# cgclassify -g cpu,memory:test-subgroup 11002 8699
# cat /CGroups/cpu/test-subgroup/tasks
8699
11002
[root@test ~]# cat /CGroups/memory/test-subgroup/tasks
8699
11002
```

cgclassify 的语法为：

```
cgclassify -g subsystems:path_to_CGroups pidlist
```

其中各选项含义如下。

- subsystems 是用逗号分隔的子系统列表，或者 * 启动与所有可用子系统关联的层级中的进程。请注意：如果在多个层级中有同名的 CGroups，则 -g 选项会将该进程移动到每个组群中。
- path_to_CGroups 是到其层级中的 CGroups 的路径。
- pidlist 表示进程 PID 列表。

还可以在 pidlist 前面添加 --sticky 选项以保证所有子进程位于同一 CGroups 中。如果没有设定这个选项且 cgred 守护进程正在运行，则会根据 /etc/cgrules.conf 中的设置将子进程分配到 CGroups 中，该进程本身则仍保留在启动它的 CGroups 中。

对等的 shell 命令为：

```
# echo 8699 > /cgroups/memory/test-subgroup/tasks
# echo 11002 > /cgroups/memory/test-subgroup/tasks
# echo 8699 > /cgroups/cpu/test-subgroup/tasks
# echo 11002 > /cgroups/cpu/test-subgroup/tasks
```

某些糟糕的进程可能导致 CPU 异常或内存泄露而影响其他进程的运行，甚至影响系统的稳定。使用 cgclassify 命令可以实现进程间的资源隔离，尽量避免上述情况的发生。

6. cgred 守护进程

cgred 是一个守护进程，它可根据在 /etc/cgrules.conf 文件中设定的参数将进程移动到 CGroups 中。

```
#  cgcreate -g cpu:/test-subgroup
# vim /etc/cgrules.conf
apache          cpu            test-subgroup/
//这个条目指定任何属于名为 apache 用户的进程根据在 test-subgroup/ 中指定的参数访问 cpu 子系统
# /etc/init.d/cgred restart
# ps -ef |grep httpd
root      2966 31708  0 22:26 pts/3    00:00:00 grep httpd
root     11002     1  0 18:38 ?        00:00:00 /usr/sbin/httpd
```

1107114476@qq.com
dolphinebook.com

```
apache    11005 11002  0 18:38 ?        00:00:00 /usr/sbin/httpd
apache    11006 11002  0 18:38 ?        00:00:00 /usr/sbin/httpd
apache    11007 11002  0 18:38 ?        00:00:00 /usr/sbin/httpd
apache    11008 11002  0 18:38 ?        00:00:00 /usr/sbin/httpd
apache    11009 11002  0 18:38 ?        00:00:00 /usr/sbin/httpd
apache    11010 11002  0 18:38 ?        00:00:00 /usr/sbin/httpd
apache    11011 11002  0 18:38 ?        00:00:00 /usr/sbin/httpd
apache    11012 11002  0 18:38 ?        00:00:00 /usr/sbin/httpd
[root@test ~]# cat /cgroups/cpu/test-subgroup/tasks  // 可以看到已经生效
11005
11006
11007
11008
11009
11010
11011
11012
```

/etc/cgrules.conf 文件中的条目可包括以下额外符号。

- @：当在 user 使用前缀时，代表是一个用户组而不是单独用户。例如，@admins 是 admins 组中的所有用户。
- *：代表“所有”。例如，subsystem 字段中的 * 代表所有子系统。
- %：代表与以上行中项目相同的项目。

7. 在控制组群中启动一个进程

有些子系统拥有强制参数，将进程移动到子系统前必须设定这些参数。例如，将进程移动到使用 cpuset 子系统的 CGroups 前，必须为那个 CGroups 定义 cpuset.cpus 和 cpuset.mems 参数。

下面将说明命令的相关语法，但只适用于本小节为所有控制器设定了相关强制参数的系统。如果还没有配置相关控制器，将无法直接将本小节中的命令示例用于您的系统。

运行 cgexec 命令在 CGroups 中启动进程。

```
cgexec -g cpu:test-subgroup lynx http://www.aliyun.com
```

这个命令启动了 test-subgroup 控制组群中的 lynx 网页浏览器，目的是限制 cpu 子系统为那个组群造成的负担。

cgexec 语法为：

```
cgexec -g subsystems:path_to_CGroups command arguments
```

其中：

- subsystems 是用逗号分隔的子系统列表或者 * 启动与所有可用子系统关联的层级中的进程。
- path_to_CGroups 是到与该层级相关的控制组群的路径。

1107114476@qq.com
dolphinebook.com

❑ command 是要运行的命令。

❑ arguments 是运行该命令时所需的参数。

cgexec 命令类似于 cgclassify，也有 --sticky 选项，用于将所有子进程都放在同一个 CGroups 中。如果没有设定这个选项，且 cgred 守护进程正在运行，则将根据在 /etc/cgrules.conf 中的设置将子进程分配到 CGroups 中，而该进程本身仍保留在启动它的 CGroups 中。

对等的 shell 命令为：

```
# echo $$ > /cgroups/cpu/test-subgroup/tasks
```

请注意：退出 lynx 后，现有 shell 中仍在控制组群 /cgroups/cpu/test-subgroup 中。可以通过下面的方法避免这种比较尴尬的情况：

```
# sh -c "echo \$$ > /CGroups/cpu/test-subgroup/tasks && lynx"
```

可在某个 CGroups 中启动某些服务。在 CGroups 中启动的服务必须：

❑ 使用 /etc/sysconfig/servicename 文件。

❑ 使用 /etc/init.d/functions 的 daemon() 功能启动该服务。

要在 CGroups 中启动某个服务，需要在 /etc/sysconfig 中编辑该服务对应的文件，使该文件包含指定的格式：CGROUPS_DAEMON="subsystem:control_group"，其中 subsystem 是与具体层级关联的子进程，control_group 是那个层级中的 CGroups。

例如，当启动 Libvirtd 服务时，会自动在各子系统下创建 Libvirt 组群，主要是因为 Libvirt 会自动检查 CGroups 的位置并自动挂载，在 /etc/libvirt/qemu.conf 配置文件中有具体说明。

```
# What cgroups controllers to make use of with QEMU guests
#
#  - 'cpu' - use for schedular tunables
#  - 'devices' - use for device whitelisting
#  - 'memory' - use for memory tunables
#  - 'blkio' - use for block devices I/O tunables
#  - 'cpuset' - use for CPUs and memory nodes
#  - 'cpuacct' - use for CPUs statistics.
#
# NB, even if configured here, they won't be used unless
# the administrator has mounted CGroupss, e.g.:
#
#  mkdir /dev/CGroups
#  mount -t CGroups -o devices,cpu,memory,blkio,cpuset none /dev/CGroups
#
# They can be mounted anywhere, and different controllers
# can be mounted in different locations. libvirt will detect
# where they are located.
#
#CGroups_controllers = [ "cpu", "devices", "memory", "blkio", "cpuset", "cpuacct" ]
```

1107114476@qq.com
dolphinebook.com

```
# This is the basic set of devices allowed / required by
# all virtual machines.
#
# As well as this, any configured block backed disks,
# all sound device, and all PTY devices are allowed.
#
# This will only need setting if newer QEMU suddenly
# wants some device we don't already know about.
#
#CGroups_device_acl = [
#    "/dev/null", "/dev/full", "/dev/zero",
#    "/dev/random", "/dev/urandom",
#    "/dev/ptmx", "/dev/kvm", "/dev/kqemu",
#    "/dev/rtc","/dev/hpet"
#]
```

所以当启动Libvirtd服务时，会将每个子系统自动挂载到Libvirt层级中，例如：

```
[root@localhost ~]# ll /cgroups/blkio/libvirt   //因为在重启cgconfig服务后并没有重启
//Libvirtd服务，所以在blkio子系统中并没有Libvirtd组群
ls: cannot access /cgroups/blkio/libvirt: No such file or directory
[root@localhost ~]# /etc/init.d/libvirtd restart
Stopping libvirtd daemon:                                  [  OK  ]
Starting libvirtd daemon:                                  [  OK  ]
[root@localhost ~]# ll /cgroups/blkio/libvirt   //重启Libvirtd服务后可以看到已经自动
                                                //挂载Libvirtd组群
total 0
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.io_merged
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.io_queued
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.io_service_bytes
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.io_serviced
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.io_service_time
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.io_wait_time
--w-------. 1 root root 0 Dec 31 11:32 blkio.reset_stats
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.sectors
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.throttle.io_service_bytes
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.throttle.io_serviced
-rw-r--r--. 1 root root 0 Dec 31 11:32 blkio.throttle.read_bps_device
-rw-r--r--. 1 root root 0 Dec 31 11:32 blkio.throttle.read_iops_device
-rw-r--r--. 1 root root 0 Dec 31 11:32 blkio.throttle.write_bps_device
-rw-r--r--. 1 root root 0 Dec 31 11:32 blkio.throttle.write_iops_device
-r--r--r--. 1 root root 0 Dec 31 11:32 blkio.time
-rw-r--r--. 1 root root 0 Dec 31 11:32 blkio.weight
-rw-r--r--. 1 root root 0 Dec 31 11:32 blkio.weight_device
--w--w--w-. 1 root root 0 Dec 31 11:32 CGroups.event_control
-rw-r--r--. 1 root root 0 Dec 31 11:32 CGroups.procs
drwxr-xr-x. 2 root root 0 Dec 31 11:32 lxc
-rw-r--r--. 1 root root 0 Dec 31 11:32 notify_on_release
drwxr-xr-x. 2 root root 0 Dec 31 11:32 qemu
-rw-r--r--. 1 root root 0 Dec 31 11:32 tasks
```

1107114476@qq.com
dolphinebook.com

需要注意的是，每次重启完 cgconfig 服务后必须在重新启动 Libvirtd 服务（虚拟机也要重新启动）。

8. CGroups（组群）相关属性

1）查找某个进程所属 CGroups。

```
[root@localhost ~]# ps -O cgroups
  PID CGROUPS                                  S TTY          TIME COMMAND
14266 blkio:/;net_cls:/;freezer:/;devices:/;cpuacct:/;memory,cpu:/;cpuset:/
S pts/2 00:00:00 /bin/bash
22610 blkio:/;net_cls:/;freezer:/;devices:/;cpuacct:/;memory,cpu:/;cpuset:/
R pts/2 00:00:00 ps -O cgroups
```

或者：

```
# cat /proc/25753/cgroups
255:blkio:/
254:net_cls:/
253:freezer:/
252:devices:/
251:memory:/
250:cpuacct:/
249:cpu:/
248:cpuuset:/
```

2）查找子系统。

```
# cat /proc/cgroupss
```

或者查找具体子系统的挂载点。

```
# lssubsys -m cpu
```

3）查找层级。

列出或者浏览包含层级组群的目录中的内容。

```
# tree /cgroups/
```

4）查找控制组群。

要查找某个系统的 CGroups，可以使用 lscgroups 命令。

```
# lscgroups
cpuset:/
cpu:/
cpu:/test-subgroup
cpuacct:/
memory:/
memory:/test-subgroup
devices:/
freezer:/
```

1107114476@qq.com
dolphinebook.com

```
net_cls:/
blkio:/
```

5）显示控制组群的参数。

可以使用 cgget 命令来显示。

```
cgget -r parameter list_of_CGroupss
```

其中 parameter 是包含子系统值的文件，list_of_CGroupss 是用空格分隔的 CGroups 列表。例如：

```
# cgget -r cpuset.cpus /
/:
cpuset.cpus: 0-15
```

如果不知道参数名称，请使用以下命令：

```
# cgget -g cpuset /
/:
cpuset.memory_pressure_enabled: 0
cpuset.memory_spread_slab: 0
cpuset.memory_spread_page: 0
cpuset.memory_pressure: 0
cpuset.memory_migrate: 0
cpuset.sched_relax_domain_level: -1
cpuset.sched_load_balance: 1
cpuset.mem_hardwall: 0
cpuset.mem_exclusive: 1
cpuset.cpu_exclusive: 1
cpuset.mems: 0-1
cpuset.cpus: 0-15
```

6.3 KVM 虚拟机 CPU 资源限制配置

在 QEMU/KVM 中，QEMU 提供对 CPU 的模拟，展现给虚拟机一定的 CPU 数目和 CPU 的特性。每个虚拟机都是一个标准的 Linux 进程（QEMU 进程），每一个 VCPU 在宿主机中是 QEMU 进程派生的一个普通线程。因此，对虚拟机 CPU 资源的限制最后归结到对该线程的限制。

1. 虚拟机 CPU 绑定

（1）CGroups 方式

开启一台测试用虚拟机 cgroups-test。

```
# ps -eLo pid,lwp,psr,args|grep qemu-kvm | grep -v grep |grep cgroups-test
5175  5175   9 /usr/libexec/qemu-kvm -name cgroups-test -S -M rhel6.5.0
5175  5200   5 /usr/libexec/qemu-kvm -name cgroups-test -S -M rhel6.5.0
```

如上所示，虚拟机 cgroups-test 在宿主机中有两个线程，其中线程 5175 代表整个虚拟机，线程 5200 是 VCPU 的线程。

1107114476@qq.com
dolphinebook.com

重复运行数遍上述命令，会发现第三列即当前线程运行在那台宿主机 CPU 核是变化的。接下来将通过操作把虚拟机绑定到第 0 个核上。

```
# echo "0" > /cgroups/cpuset/libvirt/qemu/cgroups-test/emulator/cpuset.cpus
# echo "5175" > /cgroups/cpuset/libvirt/qemu/cgroups-test/emulator/tasks
# echo "5200" > /cgroups/cpuset/libvirt/qemu/cgroups-test/emulator/tasks
# ps -eLo pid,lwp,psr,args|grep qemu-kvm | grep -v grep |grep CGroups-test
 5175  5175   0 /usr/libexec/qemu-kvm -name cgroups-test -S -M rhel6.5.0
 5175  5200   0 /usr/libexec/qemu-kvm -name cgroups-test -S -M rhel6.5.0
```

虚拟机已经运行在了 CPU 0 上，而且不会被调度到其他核上。

taskset 方式，-p,--pid 操作已存在的 PID,-c,--cpu-list 通过列表显示方式设置 CPU。

```
# ps -eLo pid,lwp,psr,args|grep qemu-kvm | grep -v grep |grep cgroups-test
31803 31803   8 /usr/libexec/qemu-kvm -name cgroups-test -S -M rhel6.5.0
31803 31828   1 /usr/libexec/qemu-kvm -name cgroups-test -S -M rhel6.5.0
# taskset -cp 0 31803
# taskset -cp 0 31828
# ps -eLo pid,lwp,psr,args|grep qemu-kvm | grep -v grep |grep cgroups-test
31803 31803   0 /usr/libexec/qemu-kvm -name cgroups-test -S -M rhel6.5.0
31803 31828   0 /usr/libexec/qemu-kvm -name cgroups-test -S -M rhel6.5.0
```

另外还可以通过直接修改 .xml 文件的方式来绑定 VCPU，例如：

```
<vcpu placement='static' cpuset='0'>1</vcpu>
```

（2）Libvirt 方式

Libvirt 实际上也是通过 CGroups 去做限制，但是可以通过 virsh 命令和虚拟机的 .xml 配置文件进行配置，配置方法已经在第 3 章进行了详细介绍。

2. CPU 时间的分配

上面讨论的是给进程分配不同的 CPU core，当多个进程运行在同一个 CPU core 上时，也可以限制每个进程得到的 CPU 时间的比例。

开启两台虚拟机，用上一小节的方法让两台虚拟机都运行在 CPU 0 上。

编写一个消耗 CPU 的脚本 cpu.sh，用于验证 CPU 时间的分配比例。

```
#! /bin/sh
# filename killcpu.sh
if [ $# != 1 ] ; then
  echo "USAGE: $0 <cpus>"
  exit 1;
fi
for i in 'seq $1'
do
  echo -ne "
i=0;
while true
do
```

1107114476@qq.com
dolphinebook.com

```
i=i+1;
done" | /bin/sh &
  pid_array[$i]=$! ;
done

for i in "${pid_array[@]}"; do
  echo 'kill ' $i ';';
done
```

然后在两台虚拟机中都运行上述脚本。

```
# bash cpu.sh 1
# bash cpu.sh 1
```

此时，在宿主机上观察两台虚拟机占用的 CPU 时间几乎是相同的。

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
10983 qemu     20   0 2396m 348m 4548 S 50.0  0.7   3:42.04 qemu-kvm
16603 qemu     20   0 2387m 344m 4548 R 49.7  0.7   2:43.32 qemu-kvm
```

修改两台虚拟机的 .xml 配置文件。

1）cgroups-test 配置文件加入如下内容：

```
<cputune>
    <shares>2048</shares>
</cputune>
```

2）cgroups-test2 配置文件加入如下内容：

```
<cputune>
    <shares>512</shares>
</cputune>
```

3）继续运行 cpu.sh 消耗 CPU，此时宿主机上的情况如下，接近 4:1。

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
17012 qemu     20   0 2410m 310m 4548 S 79.3  0.6  10:03.00 qemu-kvm
17052 qemu     20   0 2396m 252m 4548 S 19.3  0.5   2:50.14 qemu-kvm
```

可以通过设置 cpu.shares 来达到同样的目的。例如，在两个 CGroups 中都将 cpu.shares 设定为 1 的进程将有相同的 CPU 时间，但在 CGroups 中将 cpu.shares 设定为 2 的进程可使用的 CPU 时间是在 CGroups 中将 cpu.shares 设定为 1 的进程可使用的 CPU 时间的两倍。

6.4 KVM 虚拟机网络资源限制

CGroups 对网络方面的限制没有 CPU 和磁盘那么方便，限制网络流量一般不使用 CGroups，目前笔者在生产环境中主要使用以下 3 种限制方案：

❑ 通过 TC 限制虚拟机流量，TC 是内核中一套限制网络流量的机制。

1107114476@qq.com
dolphinebook.com

- 通过 Libvirt 限制虚拟机流量。
- 通过 iptables 限制虚拟机流量。

为了对比网络流量限制效果，先介绍一下不做流量限制的网络测试情况，笔者的演示环境中宿主机是千兆网卡，在不做限制的情况下，利用 Netperf 测试虚拟机的结果如下：

```
# netperf -t TCP_STREAM -H 10.0.0.161 -p 12865 -l 60
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.69.68.156
(10.69.68.156) port 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
 87380  16384  16384    60.01     947.97
```

可以看到，基本可以达到线速。

Netperf 参数介绍（Netperf 使用在第 15 章有详细介绍）。

- -t testname：指定进行的测试类型，包括 TCP_STREAM、UDP_STREAM、TCP_RR、TCP_CRR 和 UDP_RR。
- -H host：指定远端运行 netserver 的 server IP。
- -P port：指定 netserver 的端口号。
- -l testlen：指定测试的时间长度（秒）。

1. 使用 TC（Traffic Control）限制虚拟机带宽

TC（Traffic Control）是 Linux 进行流量控制的工具，可以控制网络接口发送数据的速率。每个网络接口都有一个队列，用于管理和调度待发的数据。TC 的工作原理就是通过设置不同类型的网络接口队列，从而改变数据包发送的速率和优先级，达到流量控制的目的。

Linux 内核中支持的队列如下。

- TBF（Token Bucket Flow）：令牌桶过滤器，适用于流量整形。
- Pfifo_fast，Third Band First In First Out Queue 先进先出队列。
- CBQ 队列：分类的队列，用于实现精细的 QoS 控制，配置复杂。
- SFQ（Stochastic Fairness Queueing）：随机公平队列。
- HTB（Hierarchy Token Bucket）：分层令牌桶，用于实现精细的 QoS 控制，配置比 CBQ 简单些。

TC 控制的主要是出口流量，入口流量不能通过 TC 控制。

（1）通过 TC 限制网卡流量

1）通过 TBF 限制流出速度的例子。

```
#tc qdisc add dev eth0 root tbf rate 51200kbit latency 50ms minburst 200k burst 200k
```

1107114476@qq.com
dolphinebook.com

限制网卡 eth0 流出速度为 51 200kbit/s，正确的设置方法和 minburst 这个参数有很大的关系，不同的硬件环境和系统需要具体调试。

① 删除队列的命令。

```
tc qdisc del dev eth0 root tbf
```

② 修改队列的命令。

```
tc qdisc change dev eth0 root tbf rate 2200kbit latency 5000ms burst 1540
```

③ 查看当前队列。

```
tc -s -d qdisc ls
```

2）通过 HTB 队列，针对不同 IP 源限速。

① 建立一个 htb 根队列。

```
tc qdisc add dev eth0 root handle 1: htb
```

② 建立一个父类 1：1，速度为 100Mbit/s。

```
tc class add dev  eth0 parent 1: classid 1:1 htb rate  100mbit ceil 100mbit
```

③ 建立一个子类 1：10，速度为 50Mbit/s，这里的 50Mbit/s 包含在 1：1 的 100Mbit/s 中。

```
tc class add dev  eth0 parent 1:1 classid 1:10 htb rate 50mbit ceil 50mbit
```

④ 指定 1：10 是一个随机公平队列。

```
tc qdisc add dev  eth0 parent 1:10 sfq perturb 10
```

⑤ 通过过滤器，保证 1.1.1.1 这个 IP 通过 1：1 这个队列，优先级是 2。

```
tc filter add dev eth0 protocol ip parent 1: prio 2   u32 match ip dst 1.1.1.1/32
flowid 1:1
```

通过过滤器，保证所有的其他 IP 通过 1：10 这个队列，优先级是 50。

```
tc filter add dev eth0 protocol ip parent 1: prio 50 u32 match ip dst 0.0.0.0/0
flowid 1:10
```

（2）通过 TC 限制虚拟机流量

通过 TC 的流量限制，限制虚拟机的流量，分为以下几个步骤：

1）建立队列。

2）建立分类。

3）建立过滤器，过滤虚拟机的 IP。

4）绑定网卡。

下面是一个在宿主机上限制网卡流量的脚本。

1107114476@qq.com
dolphinebook.com

```
#!/bin/bash
DEV=em1
tc qdisc del dev $DEV root
tc qdisc add dev $DEV root handle 1: cbq avpkt 1000 bandwidth 1000mbit
# 创建一个队列，限制带宽为 1000M
tc class add dev $DEV parent 1: classid 1:1 cbq rate 100kbit allot 1500 prio 5
bounded isolated
# 创建子队列，带宽为 100k
tc filter add dev $DEV parent 1: protocol ip prio 16 u32 match ip src 10.0.0.161
flowid 1:1
# 将子队列绑定到源 IP 为 10.0.0.161 的过滤器上，此 IP 为虚拟机 IP
tc qdisc add dev $DEV parent 1:1 sfq perturb 10
# 将过滤器绑定到网卡上
```

其中 DEV 是与虚拟机进行桥接的宿主机网卡名。测试结果如下：

```
# netperf -t TCP_STREAM -H 10.0.0.161 -p 12865 -l 60
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.0.0.161
(10.0.0.161) port 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
87380  16384  16384    14.00       0.09
```

流量基本被限定到 100KB。

2. 使用 Libvirt 限制虚拟机带宽

Libvirt 的虚拟机流量限制，配置非常方便，在虚拟机配置文件中的网卡部分加入如下内容：

```
<bandwidth>
    <inbound average='100' peak='50' burst='1024'/>
    <outbound average='100' peak='50' burst='1024'/>
</bandwidth>
```

目标是限制进出带宽平均不超过 100KB/s，最高不超过 50KB/s，实际结果和预想的有出入，测试结果如下。

1）流进几乎没有影响。

```
# netperf -H 10.0.0.161 -l 60
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.0.0.161 ()
port 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
87380  16384  16384    60.01     820.58
```

2）流出是 177.55MB。

1107114476@qq.com
dolphinebook.com

```
#netperf -H 10.0.0.167 -l 20
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.0.0.167 ()
port 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
87380  16384  16384    20.02     177.55
```

Libvirt实际也是使用TC限制虚拟机流量，虚拟机配置生效的时候，通过TC队列查看命令，可以看到如下TC队列：

```
tc -s -d qdisc ls
...
qdisc pfifo_fast 0: dev virbr0-nic root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0
0 1 1 1 1 1 1 1 1
 Sent 614 bytes 8 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
qdisc htb 1: dev macvtap0 root refcnt 2 r2q 10 default 1 direct_packets_stat 0
ver 3.17
 Sent 468 bytes 6 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
qdisc sfq 2: dev macvtap0 parent 1:1 limit 127p quantum 1514b depth 127 flows
128/1024 divisor 1024 perturb 10sec
 Sent 468 bytes 6 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
qdisc ingress ffff: dev macvtap0 parent ffff:fff1 ----------------
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

因为TC只能限制流出方向流量，不能限制流入方向流量，所以通过Libvirt限制流量，实际也只能限制流出方向流量。

3. 使用iptables限制虚拟机的发包率

（1）通过iptables限制虚拟机的流进方向的发包率

在宿主机上用iptables来限制虚拟机的发包率，也是限制虚拟机流进流量的一个有效方案。

默认情况下，经过桥接的虚拟机网络流量不受宿主机的iptables控制，需要改动一个内核参数。

```
# echo 1 > /proc/sys/net/bridge/bridge-nf-call-iptables
```

然后添加两条iptables规则。

```
iptables -A FORWARD -d 10.0.0.161 -m limit --limit 100/sec --limit-burst 100 -j ACCEPT
# 限定每秒100个数据包
iptables -A FORWARD -d 10.0.0.161 -j DROP
```

同时测试了一下流进、流出方向的流量，测试结果如下。

1）流进方向流量被限制得很明显，只有2.06MB。

1107114476@qq.com
dolphinebook.com

```
#netperf -H 10.20.102.161 -l 60
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.20.102.161 ()
port 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
87380  16384  16384    60.01       2.06
```

2）流出方向流量被限制得也很明显，是 213.50MB。

```
#netperf -H 10.0.0.167 -l 60
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.0.0.167 ()
port 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
87380  16384  16384    60.23     213.50
```

（2）通过 iptables 限制虚拟机的流出方向的发包率

```
iptables -A FORWARD -s 10.0.0.161 -m limit --limit 100/sec --limit-burst 100 -j ACCEPT
# 限定虚拟机源地址每秒 100 个数据包
iptables -A FORWARD -s 10.0.0.161 -j DROP
```

测试结果如下。

1）流进方向流量明显受到限制，是 19.31MB。

```
netperf -H 10.20.102.161 -l 60
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.20.102.161 ()
port 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
87380  16384  16384    60.49      19.31
```

2）流出方向流量也受到限制，是 1.58MB。

```
netperf -H 10.20.102.167 -l 60
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 10.20.102.167 ()
port 0 AF_INET
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
87380  16384  16384    60.63       1.58
```

（3）iptables 限制流量结论

- 通过 iptables 不能精确控制网速，只能控制包的个数，具体数对可以用 mtu 乘包的个数计算。

1107114476@qq.com
dolphinebook.com

- 因为网络协议是双向的，因此 iptables 限制网络流量，即使配置的是单向的，实际对流出、流进方向流量都有影响。

6.5 KVM 虚拟机磁盘资源限制

1. 限制磁盘 IOPS

磁盘限制是指定 CGroups 中某设备每秒钟可以执行的读 / 写请求数上限。其格式为：

```
major:minor operations_per_second
```

比如：

```
echo "253:4 100">blkio.throttle.read_iops_device
```

100 为虚拟机磁盘的 IOPS，blkio.throttle.read_iops_device 限制虚拟机读 IOPS。

对虚拟机磁盘进行读压力测试，同样使用 Iometer 进行压测，如图 6-2 所示。

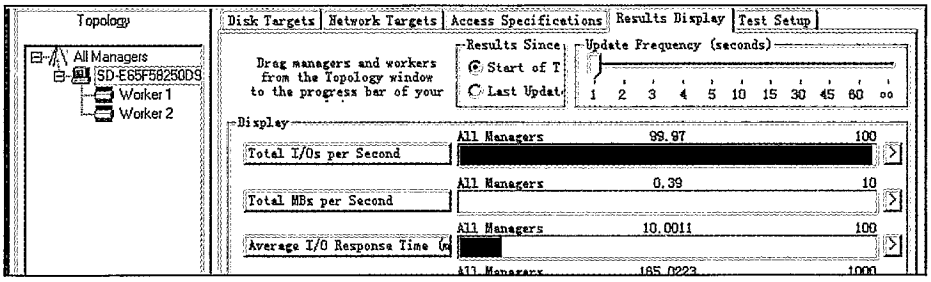


图 6-2 磁盘 IOPS 限制

2. 限制磁盘吞吐

磁盘吞吐限制主要通过以下 CGroup 参数配置：

- blkio.throttle.read_bps_device。
- blkio.throttle.write_bps_device。

指定 CGroups 中某设备每秒钟读 / 写数据的字节上限。其格式为：

```
major:minor bytes_per_second
```

比如：

```
# echo "253:2 10485760" > blkio.throttle.write_bps_device
```

10485760 指限制虚拟机吞吐为 10MB，blkio.throttle.write_bps_device 限制写的吞吐，253:2 是虚拟机磁盘的设备号，通过如下方法可获得：

```
# virsh dumpxml CGroups-test |grep file
    <disk type='file' device='disk'>
```

1107114476@qq.com
dolphinebook.com

```
      <source file='/datapool/CGroups-test'/>
# ls -l /dev/datavg/datapool
lrwxrwxrwx 1 root root 7 Dec 25 13:21 /dev/datavg/datapool -> ../dm-2
# ls -l /dev/dm-2
brw-rw---- 1 root disk 253, 2 Dec 25 13:21 /dev/dm-2
```

对虚拟机磁盘进行写压力测试（dd、fio、Iometer 都可以，使用方法在第 15 章有详细介绍），这里使用 Iometer 进行测试，如图 6-3 所示。

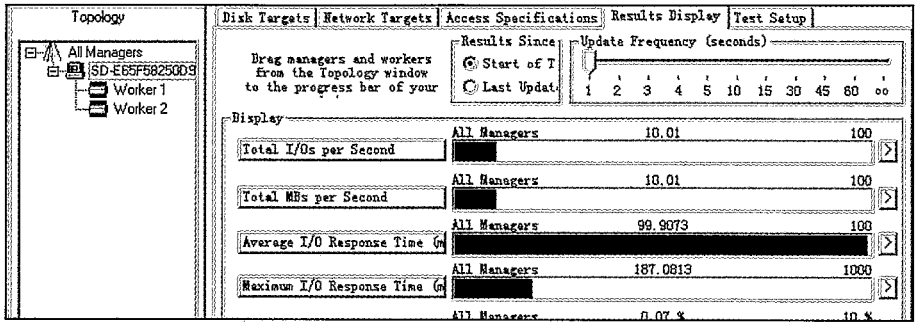


图 6-3　磁盘吞吐限制

可以看到虚拟机磁盘写的吞吐最高只能到 10MB，达到了预期效果。

通过 CGroups 可以很方便地控制磁盘 IOPS 和吞吐，在生产环境中可以通过写一些脚本来监控虚拟机磁盘的使用情况，当超过某个阈值时自动使用脚本中已经定义好的 IOPS 及吞吐的值进行限制，避免因为某台虚拟机占用过高磁盘资源而影响其他虚拟机的正常运行。

6.6　本章小结

资源限制在公有云上是必需的，在当今的公有云中几乎都在限制虚拟机的 CPU、磁盘资源、网络资源。

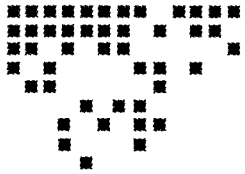
对于 CPU 资源的限制，现在通用的做法是所有虚拟机共享 CPU 资源，可以达到节省成本的目的，一般不会去绑定，当然对于特定用户或特定业务除外。

磁盘资源限制特别是对于后端使用分布式存储架构的公有云就显得十分重要。如果使用本地 SSD 磁盘，或者使用本地 SATA 磁盘（改变 I/O 路径，顺序读 / 写都落到缓存盘，再由缓存盘随机写到本地盘）的架构，提供的性能非常高，可以考虑只在必要的时候去适当限制一下就可以了。

网络资源的限制往往和计费相关，因为在公有云，带宽都是要另外计费的，所以一般的云主机都会进行网络资源限制。

下一章将为读者介绍物理机转虚拟机的操作方法和实践。

1107114476@qq.com
dolphinebook.com



Chapter 7 第 7 章

物理机转虚拟机实践

本章将为读者详细介绍物理机转换成虚拟机（Physical to Virtual，P2V）的原理及方法。P2V 有许多技巧和实际的经验，其难点是物理机如何加载虚拟化的 Virtio 驱动。

7.1 哪些应用场景适合做物理机转虚拟机

（1）P2V 是什么

P2V 即物理机（P）转虚拟机（V），要求将物理机原封不动地转换成虚拟机，保持文件系统、任何文件都不发生变化。

（2）为什么要做 P2V，或者 P2V 的价值是什么

P2V 将物理机环境搬到虚拟机环境，适用于以下场景：

- 物理机利用率非常低，转换成虚拟机可以提高资源利用率。
- 物理机的软件环境如果非常复杂，将物理机转换成虚拟机，不需要重新配置软件环境。
- 转换成虚拟机，可以在虚拟化层面做快照和高可用，可提高系统的可靠性。

7.2 P2V 的技术实施方案

1. P2V 技术方案分类

P2V 的技术实施方案分为两种：静态方案和动态方案。

1107114476@qq.com
dolphinebook.com

静态方案就是将物理机关机，使用克隆软件克隆，再在虚拟机上还原。KVM 的 virit-P2V、VMware esx 3.5 以前都属于这样的解决方案。

动态方案就是物理机处于运行状态，使用专用的 Agent 将物理机文件在线复制到虚拟机中。VMware 4.0 以后采用动态方案实施 P2V。

2. 两种方案的比较

两种方案的优缺点和难点比较如下。

（1）静态方案

优点：因为是关机操作，对现有的系统不改造，即使实施失败，也不影响现有系统。

缺点：物理机的系统没有虚拟机的磁盘驱动，比如 KVM 的 Virtio，要将驱动导入到系统中，否则克隆到虚拟机中的系统不能正常启动，Windows 会蓝屏，Linux 会报无法识别硬盘的错误。

（2）动态方案

优点：开机情况下完成物理机到虚拟机转换，转换的时候不需要加载驱动。

缺点：要在现有系统安装 Agent；有一定的概率转换不成功。

7.3　Windows 系统物理机转虚拟机的操作方法

Windows 系统物理机的 P2V 需要使用 WinPE，然后再手工加载 Virtio 或者 IDE 的驱动。

7.3.1　WinPE 的制作方法

手工执行 Windows 系统的物理机转虚拟机，需要使用 WinPE 这个工具。WinPE 即 Windows Preinstallation Environment，提供一个最小的 Windows 环境，主要是微软为了方便部署 Windows 系统而设计开发的。WinPE 支持网络，但只附带命令行、记事本、VBS 等简单维护工具。

最早的 WinPE 随着 Windows XP 系统一起发布，即 WinPE 1.0。随着 Windows XP SP1 发布的是 WinPE 1.5，这是一个经典的 WinPE 版本，有许多第三方的维护管理工具都是在这个 WinPE 版本上优化修改的。随着 Windows Vista 发布的是 WinPE 2.0，WinPE 2.0 支持 powershell。随着 Windows 7 发布的是 WinPE 3.0。随着 Windows 8 发布的是 WinPE 4.0，WinPE 4.0 中加入对 .NET 的支持，所以在 WinPE 4.0 中，可以支持 .NET 平台的编程语言。随着 Windows 8.1 发布的是 WinPE 5.0。随着 Windows 8.1 update1 发布的是 WinPE 5.1，这也是当前最新的版本。Windows 10 对应的 WinPE 版本是 6.0，随着 Windows 10 一起发布。

本书以 WinPE 5.1 的制作为例，介绍 WinPE 的制作方法。

第 1 步：到微软网站下载 wadk 的包，链接地址为：

1107114476@qq.com
dolphinebook.com

```
http://go.microsoft.com/fwlink/p/?LinkID=232339
```

下载完成直接安装就可以。

第 2 步：在开始菜单启动部署和映像工具环境，如图 7-1 所示。

启动后实际是一个命令行窗口。

第 3 步：复制 WinPE 制作文件到 C 盘。

```
copype amd64 C:\WinPE_amd64
```

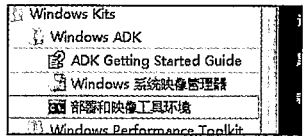


图 7-1 Windows 开始菜单的部署和映像工具环境

第 4 步：加载驱动，主要是一些比较新的服务器，WinPE 没有包含网卡和 RAID 卡，需要手工加载。

1）先要挂载镜像。

```
Dism /Mount-Image /ImageFile:"C:\WinPE_amd64\media\sources\boot.wim" /index:1 /
MountDir:"C:\WinPE_amd64\mount"
```

2）使用 dism 命令加载驱动，C:\nic\nic1.inf 是驱动文件路径。

```
Dism /Add-Driver /Image:"C:\WinPE_amd64\mount" /Driver:"C:\nic\nic1.inf"
```

提示 加载驱动的时候，一定要注意驱动程序的版本，要与 WinPE 对应的 Windows 版本匹配，并且还要注意区分 32 位版和 64 位版，否则驱动不能正确加载。

3）卸载镜像。

```
Dism /Unmount-Image /MountDir:"C:\WinPE_amd64\mount" /commit
```

第 5 步：将镜像打包。

如果制作 U 盘的 WinPE，执行以下命令：

```
MakeWinPEMedia /UFD C:\WinPE_amd64 F:
```

如果制作 ISO 的 WinPE，执行以下命令：

```
MakeWinPEMedia /ISO C:\winpe_amd64 c:\winpe_amd64\winpe.iso
```

第 6 步：WinPE 的使用。

首先需要建立一个共享，可以使用 Windows 共享功能，也可以使用 Linux SMB 服务配置。然后需要有 DHCP 服务的网络环境。在 WinPE 启动后，首先会通过 DHCP 服务获得 IP 地址，然后通过 net use 命令映射共享目录。

```
net use n: \\server\share
```

可以将 Ghost 软件放到共享目录上，然后运行 Ghost 软件，就可以开始克隆 Windows 系统到共享目录上了。

1107114476@qq.com
dolphinebook.com

7.3.2　Windows 系统 P2V 操作方法

1. 转换成 IDE 硬盘方式

第 1 步：在物理机（以下简称 P 机）上运行 ide.reg 注册表文件，注册表文件比较长，读者可以通过手机扫描如图 7-2 所示的二维码，到笔者的博客上阅读。

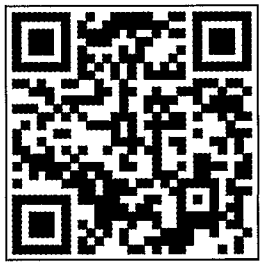


图 7-2　笔者博客的二维码

根据系统不同，复制 Windows 2003 32 位系统、64 位系统、Windows 2008 系统的 *.sys 文件到 windows\system32\driver 目录下面。驱动文件可以在 Windows 系统安装光盘或者已经有 IDE 设备的系统中复制。

第 2 步：启动 WinPE，运行 Ghost，制作镜像。

第 3 步：在 KVM 虚拟机（以下简称 V 机）上以 pxe 方式启动 WinPE，运行 Ghost，恢复镜像。

第 4 步：启动 V 机，系统会重新加载驱动，然后提示重启，完成。

2. 转换成 Virtio 硬盘方式

第 1 步：在物理机上导入 Virtio 驱动的注册表文件。

注册表文件如何得到？

注册表文件需要在一台虚拟机中，参照 IDE 驱动的注册表文件自己制作相应键值的注册表文件，并进行测试，笔者博客（http://xiaoli110.blog.51cto.com）上（读者可以扫描图 7-2 所示的二维码，连接到笔者的博客）提供已经制作好的注册表文件，读者可以下载。

根据物理机系统的不同，复制相应 Windows 2003 32 位系统、64 位系统或者 Windows 2008 系统的 *.sys 文件到 windows\system32\driver 目录下面，右击安装相应驱动下的 viostor.inf。

第 2 步：以 pxe 方式启动 WinPE，运行 Ghost，制作镜像。

第 3 步：在 V 机上启动 WinPE，运行 Ghost，恢复镜像。

第 4 步：启动 V 机，系统会重新加载驱动，在这个过程中，提示需要安装 Virtio 驱动，按照提示安装，然后根据提示重启，完成。

7.4　Linux 物理服务器转换虚拟机操作

下面介绍 Linux 系统的物理机转虚拟机的手工操作步骤。

第 1 步：克隆。

Linux 推荐使用再生龙这个克隆软件，并且配置一个 NFS，通过再生龙，将物理机克隆

1107114476@qq.com
dolphinebook.com

到 NFS 服务器上，具体操作请参考再生龙的相关操作，如图 7-3 所示。

第 2 步：还原。

① 创建一个虚拟镜像文件，空间大于物理机。

```
qemu-img create linux-P2V.qcow2.vda -f qocw2 300G
```

② 创建一个虚拟机，使用上一步创建的镜像文件，并按照要求设置 CPU 和内存。

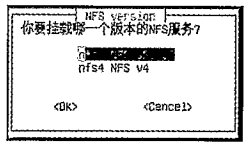


图 7-3 再升龙克隆 Linux 系统

注意：这一步硬盘采用 IDE 方式。

③ 网络引导，用再生龙将物理机镜像恢复到这台虚拟机中。

④ 关闭虚拟机。

⑤ 查看镜像分区和硬盘大小。

```
virt-df  -h linux-P2V.qcow2.vda
```

⑥ 通过 guestfish 修改 Linux 分区大小。

```
guestfish -a  linux-P2V.qcow2.vda
run
list-filesystem
e2fsck-f /dev/vda2
resize2fs-size /dev/vda2 30G
 exit
```

第 3 步：修改虚拟机硬盘为 Virtio。

通过相应版本的安装光盘引导，进入相应版本的救援模式，加载 Virtio 驱动。

```
modproble virtio virtio_pci virtio_blk virtio_net
 chroot /mnt/sysimage
sed -i "s/hda/vda/" /boot/grub/device.map
mkinitrd --with virtio --with virtio_pci --with virtio_blk --with virtio_net -f /
boot/initrd-$(uname -r).img $(uname -r)
```

Virtio 驱动加载之后请重启。

第 4 步：修改内核的时间设置。

因为虚拟机都会发生时间飘移，所以需要设置一下内核参数，使虚拟机的时间走得更精确些，同时建议设置 NTP 服务器。

按照官方文档，每个系统的内核时间参数设置如表 7-1 所示。

表 7-1 Linux 系统虚拟机时间配置方法

RedHat 企业 Linux 版本	需要添加的内核参数
6.0 AMD64/Intel 64 及上版本使用半虚拟化时钟	不需要额外添加参数
6.0 AMD64/Intel 64 及上版本不使用半虚拟化时钟	notsc lpj=*n*
5.5 AMD64/Intel 64 及上版本使用半虚拟化时钟	divider=10[1]
5.5 AMD64/Intel 64 及上版本不使用半虚拟化时钟	divider=10 notsc lpj=*n*

1107114476@qq.com
dolphinebook.com

（续）

RedHat 企业 Linux 版本	需要添加的内核参数
5.5 x86 及上版本使用半虚拟化时钟	不需要额外添加参数
5.5 x86 及上版本不使用半虚拟化时钟	divider=10 clocksource=acpi_pm lpj=*n*
5.4 AMD64/Intel 64	divider=10 notsc
5.4 x86	divider=10 clocksource=acpi_pm
5.3 AMD64/Intel 64	divider=10 notsc
5.3 x86	divider=10 clocksource=acpi_pm
4.8 AMD64/Intel 64	notsc divider=10
4.8 x86	clock=pmtmr divider=10
3.9 AMD64/Intel 64	不需要额外添加参数
3.9 x86	不需要额外添加参数

7.5　使用 virt-p2v 工具进行 P2V 转换

CentOS 6 还提供了一个工具 virt-p2v，专门用于物理机转虚拟机，就是将上面介绍的步骤自动化了。

（1）virt-p2v 支持的操作系统类型

virt-p2v 支持的操作系统类型如下：

- Red Hat Enterprise Linux 3, 4, 5, 6, 7。
- CentOS 3, 4, 5, 6, 7。
- Scientific Linux 3, 4, 5, 6, 7。
- Fedora。
- SLES 10 及以上版本。
- OpenSUSE 10 及以上版本。
- Windows XP 到 Windows 7 / Windows Server 2008 R2。

（2）支持的转换方式

virt-p2v 支持的转换方式如下：

- VMware ESXi 必须有 Vcenter，不支持从 ESXi 直接转换。
- RHEL 5 Xen。
- Citrix Xen Server 没有进一步测试。
- Hyper-V 没有测试过。
- 直接磁盘转换，只支持虚拟化引擎。
- 物理机。

（3）virt-p2v 操作前的准备工作

virt-p2v 的安装很简单，一条命令就可以。

```
yum install virt-v2v
```

1107114476@qq.com
dolphinebook.com

安装完成，在 /usr/share/virt-p2v 目录下面有一个 .iso 文件，用于在物理机上启动，将物理机转换到虚拟机。

```
cd /usr/share/virt-p2v/
ls
virt-p2v-0.9.1-2.20130730.1.el6.centos.iso
```

ISO 可以刻录成光盘，也可以转换到 U 盘上，U 盘需要先格式化成 FAT 或者 Ext2、Ext3 的格式，建议格式化成 FAT，兼容性比较高。

```
mkfs.fat /dev/sda
```

制作启动 U 盘，还需要安装 extlinux，用于 U 盘引导。

```
yum install extlinux
```

安装后运行如下脚本。

```
mkdir /mnt/p2vmount
mount -o loop virt-p2v-0.9.1-2.20130730.1.el6.centos.iso /mnt/p2vmount/
bash /mnt/p2vmount/LiveOS/livecd-iso-to-disk \
 /root/virt-p2v-0.9.1-2.20130730.1.el6.centos.iso /dev/sda
```

这样就可以完成 P2V 启动 U 盘的制作。也可以制作基于 pxe 的 P2V 工具。

（4）制作基于 pxe 的 P2V 工具

```
mkdir /mnt/p2vmount
mount -o loop virt-p2v-0.9.1-2.20130730.1.el6.centos.iso /mnt/p2vmount/
bash /mnt/p2vboot/LiveOS/livecd-iso-to-pxeboot 、virt-p2v-0.9.1-2.20130730.1.el6.
centos.iso
cp -R p2vboot/ /tftpboot/
```

宿主机需要配置 /etc/virt-v2v.conf。在文件末尾 </virt-v2v> 上面加入以下内容：

```
  -->
<profile name="myp2vtest">                                  # 配置文件名称
<method>libvirt</method>                                    # 转换方式，libvirt 或 rhev
<storage format="raw" allocation="preallocated">  # 虚拟机镜像方式 raw 或者 qcow2
default                                                     # 存储资源池名称
</storage>
<network type="default">                                    # 网络资源池名称
<network type="network" name="br0"/>                        # 网桥名称
</network>
</profile>
</virt-v2v>
```

（5）转换方法

第 1 步：使用 P2V 光盘启动，将物理机转换成虚拟机。

第 2 步：配置 IP，如图 7-4 所示。如果有 DHCP，可以使用 DHCP。

1107114476@qq.com
dolphinebook.com

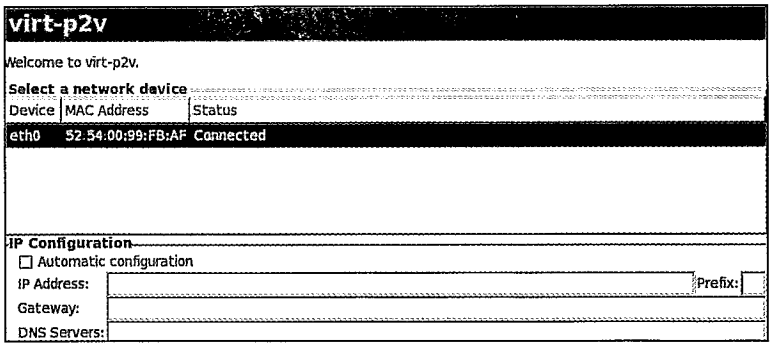


图 7-4　virt-p2v 光盘引导配置 IP 界面

第 3 步：通过 SSH 连接到宿主机，如图 7-5 所示。

宿主机配置注意点：

宿主机必须是直接用 root 连接，所以宿主机的 ssh root 登录权限必须打开。

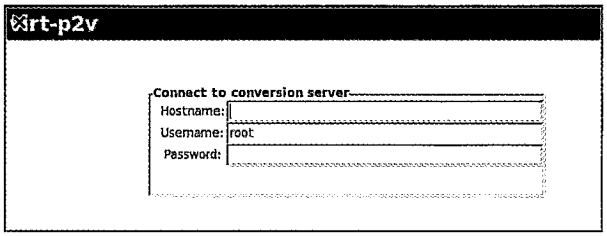


图 7-5　virt-p2v 光盘引导连接宿主机界面

第 4 步：选择磁盘和网络。

配置好，然后转换，如图 7-6 所示。完成后会提示关机，按提示操作即可。

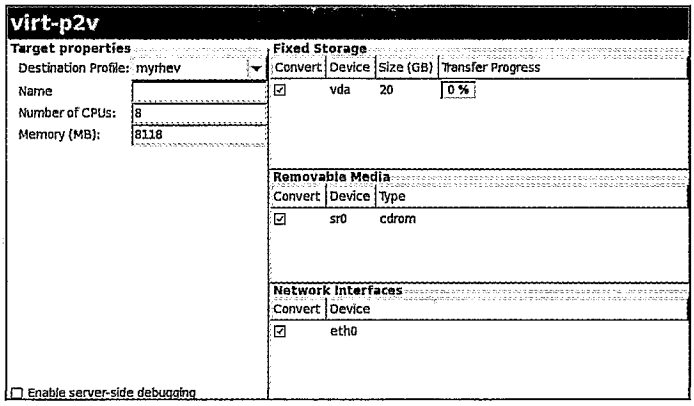


图 7-6　virt-p2v 物理机转虚拟机进行中界面

1107114476@qq.com
dolphinebook.com

到宿主机上查看，虚拟机已经生成，启动配置一下网络就可以使用了。

7.6 P2V 经验总结

操作前的准备工作很重要，物理机删除不需要的软件，Windows 系统要清空垃圾站，清空各种缓存文件，尤其是浏览器的缓存文件，笔者曾经碰到因为浏览器的缓存文件过多，导致 P2V 恢复的时候不成功的案例。

一般情况下 Windows Server 2008 R2 之前的版本及大部分 Linux 系统，用 virt-p2v 基本都能成功。如果是 Windows Server 2012，或者是服务器的 RAID virt-p2v 不能识别，就需要使用手工方式。

Windows 系统使用手工方式的时候，工具方面：

- Ghost 软件，建议使用高版本。
- WinPE 及能引导 WinPE 的 pxe 环境（环境搭建请参考微软相关文档）。

7.7 关于 V2V

V2V 即虚拟机转虚拟机，最常用的场景是将 VMware 的虚拟机转换成 KVM 的虚拟机。V2V 实际也可以按照 P2V 的方式来操作，低版本的 VMware 使用 virt-p2v 工具来操作，高版本的使用安装手工 P2V 的方法来操作。

guestfish 还提供了一个工具 virt-v2v，可以支持 VMware ESXi5.0 以前的版本直接转换成 KVM 虚拟机，但是需要有 Vcenter 支持，不能直接将单机版的 ESXi 虚拟机转换。

virt-v2v 的使用也很简单，将 VMware 虚拟机关机，运行 virt-v2v 工具，附带上 VCenter 地址密码及要转换的虚拟机名字，就可以开始转换了。

virt-v2v 也支持 Xen，但是对高版本的 Xen 支持不好，高版本的 Xen 需要使用 P2V 的方法进行转换。

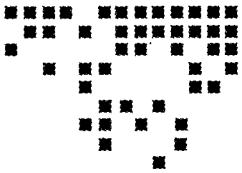
对于 Hyper-V 的虚拟机转换，没有直接转换的工具，需要使用 P2V 的方法进行转换。

7.8 本章小结

本章介绍了物理机转虚拟机的方法，物理机转虚拟机在生产环境中经常要用到，尤其是软件环境配置复杂，或者操作系统比较老，不能在较新的服务器上安装的情况。

下一章将为读者介绍 KVM 桌面虚拟化方面的实战经验。

1107114476@qq.com
dolphinebook.com



第 8 章 Chapter 8

KVM 桌面虚拟化实践

目前市场上已经有 VMware Horizon Viewer、Citrix XenDesktop 和微软 VDI 方案等成熟的商业解决方案，但是价格不菲，不是所有企业都能接受。近几年，随着 KVM 虚拟化技术的逐渐成熟，很多桌面虚拟化解决方案开始以 KVM 为虚拟化引擎。本章将为读者介绍 KVM 虚拟化技术在桌面虚拟化（Virtual Desktop Infrastructure，VDI）中应用的一些要点。桌面虚拟化与服务器虚拟化有很大不同，因此作为虚拟化引擎的 KVM 也需要一些特殊配置才能胜任桌面虚拟化的工作。本章着重介绍桌面虚拟化不同于服务器虚拟化的要点和笔者的一些生产环境的经验。

8.1 桌面虚拟化简介及 Spice 协议

桌面虚拟化将用户的桌面操作系统集中运行在服务器端，服务器端应用 VMware、Xen、KVM 和 Hyper-V 等虚拟化技术，在一台物理服务器上运行多个桌面操作系统；而用户使用 PC、瘦客户端等终端设备，通过 RDP、PCoIP 和 Spice 等远程访问协议连接到桌面操作系统。由此可见，虚拟化技术和远程访问协议是 VDI 的两大核心技术。

1. 桌面虚拟化相对于传统 PC 的优势

桌面虚拟化相对于传统 PC 有很多优势。

- 降低硬件成本。VDI 环境通常物理机的虚拟化比例非常高，例如在笔者的环境中，一台物理机要运行 50 台 Windows 7 虚拟机，给用户配置 50 台 PC 的成本显然远高于一台服务器的成本。
- 降低运维成本，提高可管理性。在传统公司中，每个员工都有自己的办公 PC，这种

1107114476@qq.com
dolphinebook.com

分布式的架构给 IT 工程师的维护工作带来很多不便。使用桌面虚拟化，所有桌面操作系统集中在服务器端，方便管理，为员工分配一个新的 PC 的工作变成了仅仅需要部署一台新的虚拟机，升级系统的操作可能变成了仅需要替换虚拟机模板；而且，服务器的故障率远低于普通 PC，增加了桌面操作系统的稳定性。

- 使移动办公变成现实。例如用户在公司编辑了一半的文档、打开的应用程序，在其他任何地方，只要通过远程连接到同一台虚拟机，就可以继续之前的工作。

桌面虚拟化的架构主要由 3 部分组成。

- 用户终端。用户终端是用户访问 VDI 的入口，可以是一台 PC、一个瘦客户端，也可以是一台平板电脑，用户通过终端连接到服务端的桌面操作系统。
- 远程访问协议。前面提到的 RDP、PCoIP 和 Spice 都是远程访问协议，用户在终端的操作和桌面操作系统的画面、声音等信息都是通过这些协议进行交互的，可以说协议的用户体验影响一个桌面虚拟化解决方案的成败。
- 虚拟化服务器。在物理服务器上通过虚拟化引擎运行多个桌面操作系统，这一部分为桌面虚拟化提供实际的计算服务。

2. Spice 协议

上文提到了，远程访问协议是桌面虚拟化的重要组成部分。基于 KVM 的桌面虚拟化方案使用的是 Spice 协议。Spice 是一个开源的远程访问协议，客户端可以通过 Spice 协议访问服务端的图像、声音及键盘、鼠标等设备。

从运维的角度，可以把 Spice 的架构分为 3 部分：Spice 客户端、Spice 服务端及 Spice Agent。

Spice 客户端（如 Linux 上的 virt-viewer）是负责发起连接的，在桌面虚拟化的架构中，Spice 客户端运行在用户终端上。

Spice 服务端负责处理客户端的连接请求，运行在宿主机上。以 Ubuntu 系统为例，需要在宿主机上安装软件包 libspice-server1。

```
root@localhost:~# apt-get install libspice-server1
```

Spice Agent 是一个可选组件，运行在虚拟机内，它的作用是增强性能，提高用户体验，并提供一些显示方面的设置。考虑到用户体验的重要性，这个组件一定要安装，具体安装方法在后面介绍 Windows镜像的小节有介绍。

8.2 桌面虚拟化实践经验

桌面虚拟化在虚拟机的视频、声音配置方面需要特殊注意，在宿主机的配置方面可以考虑使用 KSM 这样的内存压缩技术。

1107114476@qq.com
dolphinebook.com

1. 桌面虚拟化中 KVM 虚拟机的 .xml 配置文件

桌面虚拟化的虚拟机与服务器虚拟化不同，除了承载计算任务外，还要负责处理声音、图像等多媒体内容，以及考虑如何支持远程访问协议。下面先列出笔者生产环境的一台虚拟机的 xml 配置文件。

```
<domain type='kvm' id='1'>
<name>instance-00000016</name>
<uuid>3b30f4b0-ebc9-4f99-9170-e734df98c597</uuid>
<memory unit='KiB'>2097152</memory>
<currentMemory unit='KiB'>2097152</currentMemory>
<vcpu placement='static'>2</vcpu>
<sysinfo type='smbios'>
<system>
<entry name='manufacturer'>OpenStack Foundation</entry>
<entry name='product'>OpenStack Nova</entry>
<entry name='version'>1.4.0</entry>
<entry name='serial'>13d465a0-1dd2-11b2-ba78-000000821800</entry>
<entry name='uuid'>3b30f4b0-ebc9-4f99-9170-e734df98c597</entry>
</system>
</sysinfo>
<os>
<type arch='x86_64' machine='pc-i440fx-1.4'>hvm</type>
<boot dev='hd'/>
<smbios mode='sysinfo'/>
</os>
<features>
<acpi/>
<apic/>
</features>
<cpu mode='host-model'>
<model fallback='allow'/>
</cpu>
<clock offset='localtime'>
<timer name='pit' tickpolicy='delay'/>
<timer name='rtc' tickpolicy='catchup'/>
<timer name='hpet' present='no'/>
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<devices>
<emulator>/usr/bin/kvm-spice</emulator>
<disk type='file' device='disk'>
<driver name='qemu' type='qcow2' cache='none'/>
<source file='/vdesktop/3b30f4b0-ebc9-4f99-9170-e734df98c597/disk'/>
<target dev='vda' bus='virtio'/>
<alias name='virtio-disk0'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
</disk>
<interface type='bridge'>
```

1107114476@qq.com
dolphinebook.com

```
<mac address='fa:16:3e:30:29:dc'/>
<source bridge='br-int'/>
<virtualport type='openvswitch'>
<parameters interfaceid='5163a895-3a7c-47b6-adb5-264398c777e4'/>
</virtualport>
<target dev='tap5163a895-3a'/>
<model type='virtio'/>
<alias name='net0'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
<serial type='file'>
<source path='/vdesktop/3b30f4b0-ebc9-4f99-9170-e734df98c597/console.log'/>
<target port='0'/>
<alias name='serial0'/>
</serial>
<serial type='pty'>
<source path='/dev/pts/1'/>
<target port='1'/>
<alias name='serial1'/>
</serial>
<console type='file'>
<source path='/vdesktop/3b30f4b0-ebc9-4f99-9170-e734df98c597/console.log'/>
<target type='serial' port='0'/>
<alias name='serial0'/>
</console>
<channel type='spicevmc'>
<target type='virtio' name='com.redhat.spice.0'/>
<alias name='channel0'/>
<address type='virtio-serial' controller='0' bus='0' port='1'/>
</channel>
<input type='tablet' bus='usb'>
<alias name='input0'/>
</input>
<input type='mouse' bus='ps2'/>
<graphics type='vnc' port='5900' autoport='yes' listen='0.0.0.0' keymap='en-us'>
<listen type='address' address='0.0.0.0'/>
</graphics>
<graphics type='spice' port='10022' autoport='no' listen='0.0.0.0' keymap='en-us'>
<listen type='address' address='0.0.0.0'/>
</graphics>
<sound model='ich6'>
<alias name='sound0'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
</sound>
<video>
<model type='qxl' ram='65536' vram='65536' heads='1'/>
<alias name='video0'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
</video>
<memballoon model='virtio'>
<alias name='balloon0'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
```

1107114476@qq.com
dolphinebook.com

```
</memballoon>
</devices>
<seclabel type='dynamic' model='apparmor' relabel='yes'>
<label>libvirt-3b30f4b0-ebc9-4f99-9170-e734df98c597</label>
<imagelabel>libvirt-3b30f4b0-ebc9-4f99-9170-e734df98c597</imagelabel>
</seclabel>
</domain>
```

可以看到大部分地方和服务器虚拟化是一样的，需要关注几个与服务器虚拟化不同的地方。

1）sound 和 video 设备不再可有可无，而是很重要。

配置如下：

```
<sound model='ich6'>
<alias name='sound0'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
</sound>
<video>
<model type='qxl' ram='65536' vram='65536' heads='1'/>
<alias name='video0'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
</video>
```

需要修改的地方不多，video 的 model type 需要注意。如果使用 qxl，虚拟机内部需要安装 qxl 驱动程序，否则虚拟机可能启动失败。

2）对 Spice 协议的支持。

配置如下：

```
<graphics type='spice' port='10022' autoport='no' listen='0.0.0.0' keymap='en-us'>
<listen type='address' address='0.0.0.0'/>
</graphics>
```

2. Windows 7 模板制作注意事项

本节只罗列与服务器虚拟化模板不同的地方。

（1）安装 spice guest tools

在讲 Spice 协议的时候提到了 RHEV Spice Agent 和 qxl 驱动，这两个组件要安装在虚拟机中。

下载地址：

```
http://www.spice-space.org/download/windows/spice-guest-tools/spice-guest-tools-0.74.exe
```

这个安装程序集成了 RHEV Spice Agent 和 qxl 驱动。安装完成后系统中会出现 RHEV Spice Agent 服务，请确保该服务开机自动启动。

（2）系统配置调整

关闭系统的睡眠、休眠，在控制面板的“电源选项”中设置，如图 8-1 所示。

1107114476@qq.com
dolphinebook.com

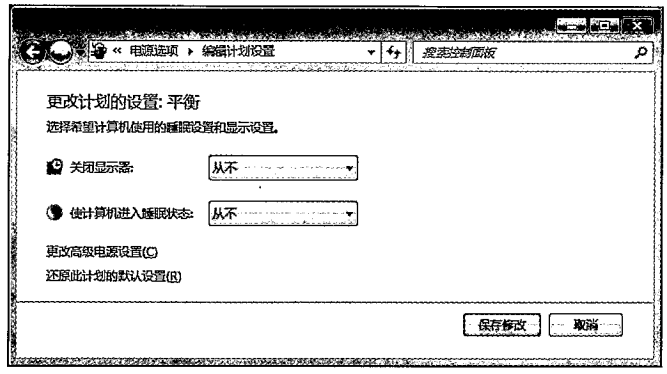


图 8-1 关闭系统休眠

关闭“系统自动关闭硬盘”功能，如图 8-2 所示。

关闭屏幕保护，如图 8-3 所示。

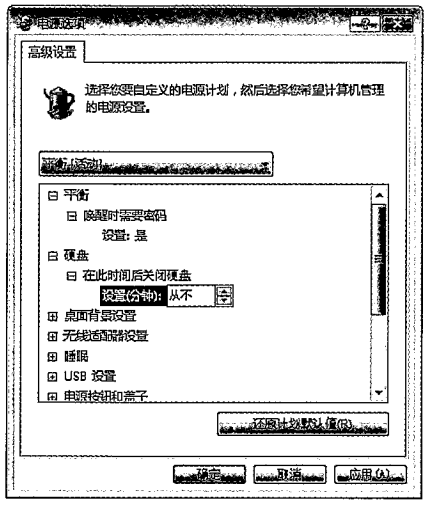


图 8-2 关闭“系统自动关闭硬盘”功能

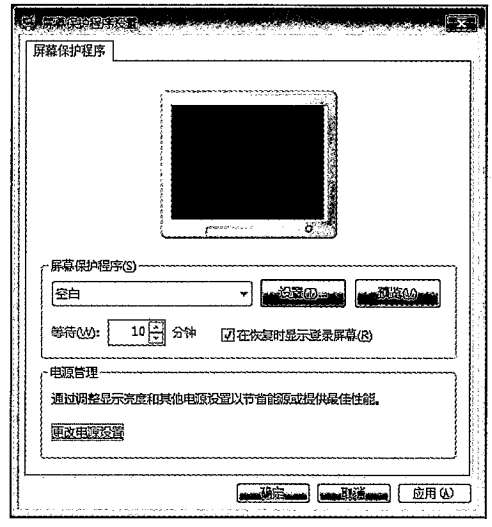


图 8-3 关闭屏幕保护

Windows 虚拟机在被硬重启或启动中有其他错误的时候容易进入恢复模式，对虚拟机来说恢复模式意义不大，而且出现后还要运维人员处理，因此可以关掉。方法是在 cmd 命令行界面中执行以下命令：

```
bcdedit /set {default} bootstatuspolicy ignoreallfailures
```

3. 宿主机系统为 Ubuntu 12.04 时的 KSM 参数调整

桌面虚拟化中由于虚拟化比例高，内存是非常珍贵的资源，运用 KSM（内存压缩技术）效果很好。但是 Ubuntu 12.06 关于 KSM 的默认配置并不能正常运行，需要进行调整。

1107114476@qq.com
dolphinebook.com

/sys/kernel/mm/ksm/pages_to_scan 指定 KSM 每次扫描多少内存页，默认值为 100。

/sys/kernel/mm/ksm/sleep_millisecs 指定 KSM 每次扫描之间的间隔时间，默认值为 200，单位是 ms。

要做的调整如下：

```
echo "1000" > /sys/kernel/mm/ksm/pages_to_scan
echo "100" > /sys/kernel/mm/ksm/sleep_millisecs
```

可以发现，调整的结果是增加了 KSM 的工作速度，调大了每次扫描的内存页数，调小了时间间隔。为了让调整永久生效，要把上述命令写入 /etc/rc.local。

4. 宿主机系统为 CentOS 6.5 的 SSD 磁盘调度算法调整

在有些对性能敏感的桌面虚拟化项目中，SSD 磁盘是一个很好的选择。SSD 磁盘在随机读 / 写方面相对于机械磁盘优势明显，但是在笔者的生产环境中曾经发现 SSD 在 CentOS 6.5 中的表现不如预期。经排查发现，CentOS 6.5 默认的磁盘调度算法 CFQ 并不适合 SSD。

最终期限算法（Deadline）和完全公平排队算法（CFQ）是常见的两种磁盘调度算法，它们最核心的工作可以总结为：为 I/O 请求进行排序以及对邻近的 I/O 请求进行合并。机械磁盘在响应 I/O 请求时最大的耗时发生在机械臂寻道，排序正是为了减少寻道时间。但是 SSD 不存在寻道的问题，排序工作便成了无意义的。因此，Linux 内核还实现了一种专为随机访问的块设备而生的调度算法——空操作（Noop），这种算法只做合并不做排序工作。修改磁盘调度的命令如下（假设 sdb 是 SSD）：

```
echo "noop" > /sys/block/sdb/queue/scheduler
```

为了让调整永久生效，可以把命令写入 /etc/rc.local。

5. 虚拟机没有声音排障

宿主机为 CentOS 6.5 时，如果通过 Spice 协议连接虚拟机发现没有声音，可以尝试调整 /etc/libvirt/qemu.conf 中的一个参数。

```
vnc_allow_host_audio = 1
```

这个值默认为 0，调整后重启 libvirt-bin 服务。如果在其他解决方法失败后，可以尝试这个调整。

8.3　本章小结

桌面虚拟化绝大部分技术和普通的虚拟化技术是一样的，本章根据笔者的生产环境经验，介绍了一些需要特别注意的地方，希望能对读者的生产环境桌面虚拟化有所帮助。

下一章开始介绍分布式文件系统在 KVM 虚拟化中的使用。

1107114476@qq.com
dolphinebook.com

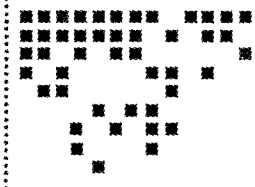
1107114476@qq.com
dolphinebook.com

第二篇 Part 2

分布式文件系统

- 第 9 章　几种常见开源文件系统在 KVM 中的应用
- 第 10 章　Ceph 在 KVM 虚拟化中的应用与故障处理

1107114476@qq.com
dolphinebook.com



Chapter 9 第 9 章

几种常见开源文件系统在 KVM 中的应用

本章将为读者介绍几种常见开源分布式文件系统在 KVM 虚拟化中的应用。本章介绍的几种分布式文件系统配置简单，都可以弹性伸缩，其中 DRBD、GlusterFS、MooseFS 几种分布式文件系统成熟稳定，在生产环境中使用比较多。Sheepdog 是专门为 KVM 定制的分布式文件系统，QEMU 原生支持，正在逐步走向成熟。本章将会介绍这些分布式文件系统的基本概念、配置方法及如何在 KVM 虚拟化中应用。

9.1 演示环境基础配置

本章介绍的 4 款分布式文件系统，笔者都在同样的环境下演示配置，4 款文件系统基础配置都一样，为了避免重复介绍，在本节为读者介绍一下笔者的演示环境及基础配置。

笔者的演示环境使用两台宿主机，一台宿主机运行 6 台虚拟机，配置分布式文件系统；一台宿主机挂载分布式文件系统，并将虚拟机运行在分布式文件系统上。

1. 硬件环境

1）宿主机 1 配置。

运行 6 台虚拟机，配置如下。

- 品牌型号：HP 380 G7。
- CPU：Intel(R) Xeon(R) X5650 @ 2.67GHz *2。
- 内存：24GB。
- 硬盘类型 1：300GB SAS*4 安装系统，存储虚拟机系统镜像。
- 硬盘类型 2：Intel 3500 SSD 300GB*2，根据测试分布式文件系统的不同，划分大小不

1107114476@qq.com
dolphinebook.com

同的 lv，挂给虚拟机使用。

2）宿主机 1 上 6 台虚拟机配置。

- CPU：4 个虚拟机 CPU。
- 内存：3GB。
- 硬盘类型 1：100GB 安装系统。
- 硬盘类型 2：根据测试分布式文件系统的不同，挂载大小不同的 lv。

3）宿主机 2 配置。

- 品牌型号：HP 380 G7。
- CPU：Intel(R) Xeon(R) X5650 @ 2.67GHz *2。
- 内存：24GB。
- 硬盘：600GB SAS*4 安装系统。

4）宿主机 2 上一台虚拟机配置。

- CPU：4 个虚拟机 CPU。
- 内存：3GB。
- 硬盘类型 1：100GB 安装系统。
- 硬盘类型 2：根据测试分布式文件系统的不同，挂载大小不同的 lv。

2. 软件环境

如果没有特殊说明，默认系统使用 CentOS 7，内核升级到 3.10.0-123.13.2.el7.x86_64，IP、主机名分配如表 9-1 所示。

表 9-1　测试环境 IP、主机名对应关系

类　型	主 机 名	IP 地址
宿主机 1	host182	10.0.0.182
分布式文件系统存储节点虚拟机 1	host183	10.0.0.183
分布式文件系统存储节点虚拟机 2	host184	10.0.0.184
分布式文件系统存储节点虚拟机 3	host185	10.0.0.185
分布式文件系统存储节点虚拟机 4	host186	10.0.0.186
分布式文件系统存储节点虚拟机 5	host187	10.0.0.187
分布式文件系统存储节点虚拟机 6	host188	10.0.0.188
宿主机 2	host251	10.0.0.251
分布式文件系统挂载虚拟机	host189	10.0.0.189

3. 基本环境配置

1）安装 epel 源。

安装 epel，执行如下命令：

```
rpm -ivh \
http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-5.noarch.rpm
```

2）配置 NTP，保证时间同步。

1107114476@qq.com
dolphinebook.com

运行命令 ntpdate ntp 服务器，并将命令添加到 crontab 中，每隔 20 分钟运行一次。

3）配置主机名 hostname。

CentOS 7 配置 hostname 有两种方法。

第一，立即生效的方法，使用 hostnamectl 命令。

查看状态，使用命令 hostnamectl status。修改 hostname 使用 hostnamectl set-hostname 加需要修改的 hostname。

第二，重启后生效的方法。

修改 /etc/hostname 配置文件，然后重启。

hostname 修改后，修改每台服务器的 /etc/hosts，将每个节点的 IP 和主机对应信息填写进去，使每个节点通过 hostname 能够互相访问。

9.2 DRBD 在 KVM 中的应用

9.2.1 DRBD 介绍

分布式复制块设备（Distributed Replicated Block Device，DRBD）是一款基于 Linux 的开源软件，它由内核模块和管理工具组成，通过网络进行块设备的同步镜像。数据写入本地的 DRBD 设备上的文件系统时，同时被发送到网络中的另外一台主机之上，并以完全相同的形式记录。本地节点（主机）与远程节点（主机）的数据可以保证实时同步，并保证 I/O 的一致性。如图 9-1 所示，DRBD 类似基于网络 RAID 1。

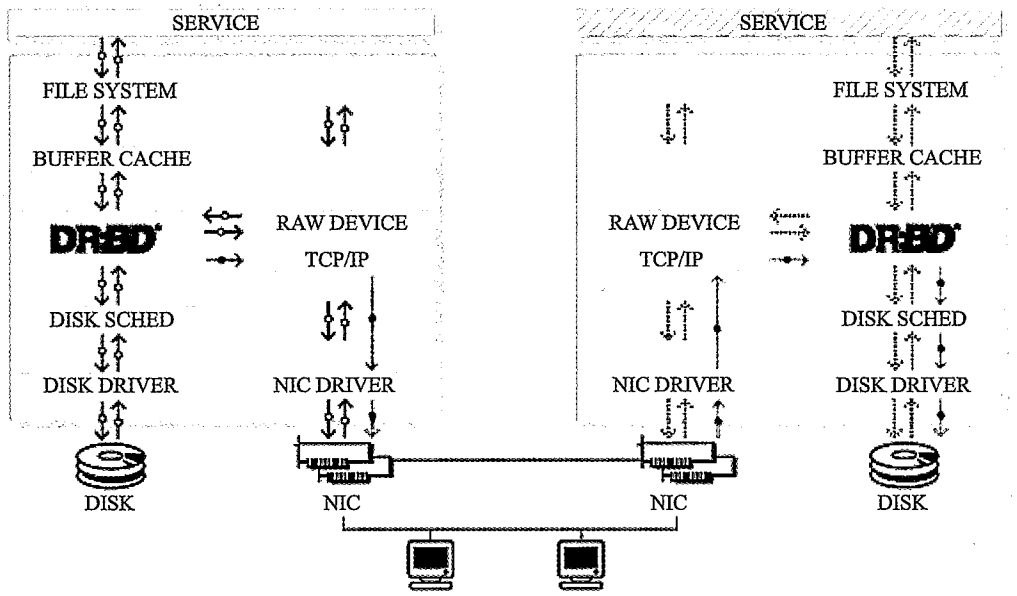


图 9-1 DRBD 架构（图片来自 DRBD 官方网站）

1107114476@qq.com
dolphinebook.com

DRBD 已经发布多个版本，目前 DRBD 9.0 版本已经发布测试版本，DRBD 成熟稳定，被亲切地称为“穷人的存储”。在高可用（HA）解决方案中使用 DRBD，可以代替使用商业存储设备。因为数据同时存在于本地主机和远程主机上，在遇到需要切换的时候，远程主机只需要使用它上面的那份备份数据，就可以继续提供服务。

（1）DRBD 复制协议

DRBD 的复制有 3 种协议。

- 协议 A：异步复制协议。本地写成功后立即返回，数据放在发送 Buffer 中，如果服务器掉电，数据可能丢失。
- 协议 B：内存同步（半同步）复制协议。本地写成功并将数据发送到对方后立即返回，如果双机掉电，数据可能丢失。
- 协议 C：同步复制协议。本地和对方写成功确认后返回。如果磁盘同时损坏，则数据可能丢失。

生产环境从数据安全性角度出发，建议使用协议 C，协议 C 也是同步速度最慢的方式，实际是牺牲速度换取了数据可靠性。

（2）DRBD 使用方式

DRBD 使用方式可以分为两种。

- 主备模式：两台存储服务器，一主一备，主出现故障的时候，备机顶替，同时只有一台服务器提供服务。
- 双主模式：两台存储服务器，同时提供服务，共同分担压力，当一台服务器故障的时候，服务性能降级，但是不会中断。

9.2.2　DRBD 搭建方法及在 KVM 中的应用

笔者生产环境使用的是 DRBD 8.4.5，DRBD 8.4.5 不支持应用层的自动切换，应用层的自动切换需要用到 Corosync、Pacemaker 工具，Corosyncs 是 Linux 集群通信套件，Pacemaker 是一套以资源管理为基础的 Linux 系统集群管理软件。

KVM 使用 DRBD 有两种用法。

- 通过 NFS 方式使用。
- 通过 iSCSI 方式使用。

笔者在生产环境中一直使用的是 iSCSI 方式，本节介绍一下使用 DRBD 搭建 iSCSI 集群的方法。

关于虚拟化后端存储使用 NFS 还是 iSCSI，业界一直有争论，争论主要集中在易用性、稳定性、性能上。笔者在生产环境中偏爱使用 iSCSI，从笔者的实践经验来看，同样条件下，iSCSI 配置虽然复杂，但是稳定性、性能、安全性都能满足需求。

1107114476@qq.com
dolphinebook.com

1. DRBD 搭建操作步骤规划

DRBD 集群的搭建规划如下。

1）两台存储服务器命名。

笔者测试环境如 9.1 节介绍的情况，在 KVM 虚拟化中使用 DRBD 一般使用主备模式，所以需要两个角色。

- 主节点（Master）：存储节点虚拟机 host185。
- 备用节点（Slave）：存储节点虚拟机 host188。

2）存储服务器镜像存储空间。

宿主机 1 的两块 SSD 分布以裸盘的形式挂给两台虚拟机使用，在存储节点虚拟机中看到的就是 /dev/vdb，作为 DRBD 的镜像盘。

3）创建同步镜像盘。

配置 DRBD 服务，并创建同步镜像盘 drbd1。

4）配置 corosync，使得两个节点可以通信。

5）配置 iSCSI 服务 ISCSITarget。

DRBD 主节点使用 /dev/drbd1 创建 pv，创建名为 iscsivg01 的 vg，划分名为 LUN1 的 lv 作为 iSCSI 的 Target 输出。

6）配置 Pacemaker。

Pacemaker 主要配置以下资源：

- 创建 DRBD 的主从资源。
- 创建心跳，浮动 IP。
- 创建 vg、lv 的资源。
- 创建 iSCSI 服务资源。

7）KVM 计算节点使用。

使用 iSCSI 方式通过浮动 IP 连接 DRBD 集群共享出来的块设备，创建 pv、vg，划分 lv 给虚拟机使用。

2. DRBD 配置

（1）DRBD 安装

因为 DRBD 的 8.4.5 版本只宣布支持 CentOS 5、6，因此 DRBD 集群的搭建使用 CentOS 6.6，内核版本升级为 2.6.32-504.3.3.el6.x86_64。

DRBD 的安装非常简单，首先安装 elrepo 的源。

```
rpm -Uvh http://www.elrepo.org/elrepo-release-6-6.el6.elrepo.noarch.rpm
```

然后通过 yum 命令安装 DRBD 内核 kmod 和管理工具，再加载一下内核就可以了。

```
yum install drbd84-utils kmod-drbd84
modprobe drbd
```

1107114476@qq.com
dolphinebook.com

（2）配置 DRBD

修改配置文件内容如下，两个节点都要操作。

```
cat /etc/drbd.conf
#include "/etc/drbd.d/global_common.conf";
include "/etc/drbd.d/*.res";
```

（3）编辑资源文件

```
cat /etc/drbd.d/iSCSIvg01.res
global {
usage-count yes;
# 是否到 DRBD 官方网站去登记
}
common {
protocol C;
# 使用协议 C 复制
disk {
on-io-error detach;
fencing resource-only;
}
net {
cram-hmac-alg sha1;
shared-secret "a6a0680c40bca2439dbe48343ddddcf4";
# 数据同步加密
}
syncer {
rate 300M;
# 同步速率控制在 300M/s
}
handlers {
fence-peer "/usr/lib/drbd/crm-fence-peer.sh";
after-resync-target "/usr/lib/drbd/crm-unfence-peer.sh";
pri-on-incon-degr "echo b > /proc/sysrq-trigger";
}
}
resource iSCSIvg01 {
device /dev/drbd1;
disk /dev/vdb;
meta-disk internal;
# 使用 /dev/vdb 创建镜像盘，镜像盘名字为 drbd1
net {
    max-buffers 8192;
    max-epoch-size 8192;
    sndbuf-size 2048k;
    unplug-watermark 127;
    }
disk {
    disk-barrier no;
    disk-flushes no;
    resync-rate 150M;
```

1107114476@qq.com
dolphinebook.com

```
    c-plan-ahead 200;
    c-max-rate 200M;
    c-min-rate 25M;
    c-fill-target 15M;
     }
on host185 {
address ipv4 10.0.0.185:7898;
}
on host188 {
address ipv4 10.0.0.188:7898;
# 两个节点信息
}
}
```

提示 c-max-rate、c-min-rate 这两个参数要保持平衡，否则会造成同步速率上不去，读者可以根据自己的实际环境调节测试。

（4）启动 DRBD 服务

两个节点都要操作。

```
/etc/init.d/drbd restart
# 创建资源
drbdadm create-md iscsivg01
drbdadm adjust iSCSIvg01
# 启动服务，并设置开机自启动
/etc/init.d/drbd restart
chkconfig drbd on
```

（5）设置主备，并同步数据

查看 DRBD 状态，可以看到两个都是 Secondary。

```
cat /proc/drbd
version: 8.4.5 (api:1/proto:86-101)
GIT-hash: 1d360bde0e095d495786eaeb2a1ac76888e4db96 build by root@host185,
2015-01-27 02:23:13
 1: cs:Connected ro:Secondary/Secondary ds:Inconsistent/Inconsistent C r-----
    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:d oos:292994436
```

提升 host185 为 DRBD 的主节点，在 host185 上执行如下命令：

```
drbdsetup /dev/drbd1 primary
```

或者下面的命令也可以：

```
drbdadm primary iSCSIvg01
```

在 host185 上查看状态，可以看到本地节点已经提升为主节点，并且开始同步镜像盘的数据。

1107114476@qq.com
dolphinebook.com

```
cat /proc/drbd
...
 1: cs:SyncTarget ro:Primary/Secondary ds:Inconsistent/UpToDate C r-----
    ns:0 nr:6259832 dw:6259712 dr:608 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:d
oos:286734724
        [>....................] sync'ed:  2.2% (280012/286124)M
        finish: 0:28:53 speed: 165,376 (160,504) want: 166,280 K/sec
```

至此，DRBR安装设置完成。

3. 配置Corosync

（1）Corosync安装

通过yum命令可以直接安装，两个节点上都要安装。

```
yum -y install Pacemaker
```

（2）配置Corosync

编辑/etc/ha.d/authkeys内容如下：

```
cat /etc/ha.d/authkeys
auth 1
1 sha1 secret
```

修改文件属性，内容如下：

```
chmod 600 /etc/ha.d/authkeys
```

编辑Corosync配置文件，内容如下：

```
cat /etc/corosync/corosync.conf
#用户、组定义
aisexec {
    user: root
    group: root
}
#服务定义，使用pacemaker服务
service {
    name: Pacemaker
    ver: 0
    use_mgmtd: yes
}
 totem {
    version: 2
    secauth: off
    threads: 0
    interface {
        ringnumber: 0
#内部网络地址段
        bindnetaddr: 10.0.0.0
         mcastaddr: 226.94.1.1
```

1107114476@qq.com
dolphinebook.com

```
        mcastport: 5405
    }
}
logging {
    fileline: off
    to_stderr: no
    to_logfile: yes
    to_syslog: yes
    logfile: /var/log/cluster/corosync.log
    debug: off
    timestamp: on
    logger_subsys {
        subsys: AMF
        debug: off
    }
}
amf {
    mode: disabled
}
```

修改日志文件属性，并启动服务。

```
chown -R hacluster. /var/log/cluster
/etc/rc.d/init.d/corosync start
chkconfig corosync on
```

配置好 Corosync 后，通过命令 crm_mon 查看资源状态，可以看到 host185、host188 两个节点都上线了。

```
crm_mon
...
2 Nodes configured, 2 expected votes
0 Resources configured
Online: [ host185 host188 ]
```

4. LVM 设置

因为是先配置 DRBD，然后配置 iSCSI，再划分 lv 给虚拟机，在虚拟机里面有可能再次使用 LVM，造成 LVM on LVM 的情况，所以需要配置 LVM，使系统不扫描 /dev/vdb 上的 LVM 信息。同时为了保证虚拟机数据一致性，关闭 LVM 缓存。

```
vi /etc/lvm/lvm.conf
#修改以下内容
filter = [ "r|/dev/vdb.*|" ]
write_cache_state = 0
```

清空已有缓存。

```
rm /etc/lvm/cache/* -r -f
```

也可以使用脚本清空缓存，脚本内容如下：

1107114476@qq.com
dolphinebook.com

```
sed -i s/write_cache_state\ =\ 1/write_cache_state\ =\ 0/g /etc/lvm/lvm.conf
rm /etc/lvm/cache/* -r -f
```

5. iSCSI 的 tgtd 服务配置

主节点上基于 DRBD 创建的块设备 drbd1，创建 pv、vg。

```
pvcreate /dev/drbd1
vgcreate iscsivg01 /dev/drbd1
```

安装 iSCSI 的 tgtd 服务。

```
yum install scsi-target-utils.x86_64 -y
```

启动 tgtd 服务，并配置开机启动。

```
/etc/init.d/tgtd start
chkconfig tgtd on
```

6. Pacemaker 安装配置

（1）crm 安装

Pacemaker 是 Linux 上的集群资源管理工具，CentOS 6.3 以前 Pacemaker 配置工具是 crm 命令，CentOS 6.4 以上的版本及 CentOS 7 系列，配置工具为 pcs 命令。因为笔者习惯使用 crm 命令，有第三方开源爱好者开发了支持 CentOS 6.4 以上版本的 crm 版本，安装步骤如下。

1）升级 autoconf 到 2.69 版，下载、解压、编译就可以了。

```
wget http://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar.gz
tar -xvf autoconf-2.69.tar.gz
./configure
make && make install
```

2）安装第三方 crm 版本。

安装很简单，git 克隆后编译安装就可以了。

```
git clone https://github.com/crmsh/crmsh.git
cd crmsh
./autogen.sh
./configure
make
make install
```

（2）下载 iSCSITarget 的 Pacemaker 脚本

Pacemaker 的理念是一切集群配置都是资源，每种资源都是脚本，需要在集群中切换资源的时候，通过相应的脚本切换，脚本默认路径是 /usr/lib/ocf/resource.d/。CentOS 6.6 安装的 Pacemaker 已经不包含 iSCSITarget 的 Pacemaker 脚本，需要手工下载，并放置在脚本资源目录中。

1107114476@qq.com
dolphinebook.com

```
git clone https://github.com/ClusterLabs/resource-agents
cd resource-agents/heartbeat/
cp iSCSITarget /usr/lib/ocf/resource.d/heartbeat/
```

（3）配置集群

1）配置集群属性。

```
crm configure
property stonith-enabled="false"
# 关闭 stonith，stonith 的主要作用是节点故障的时候，通过 ipmi 强制关机
property no-quorum-policy="ignore"
# 因为 DRBD 只有两个节点，关闭集群的投票机制
property default-resource-stickiness="200"
commit
```

2）创建 crm 资源 p_drbd_iscsivg01，基于 DRBD 的 iscsivg01。

```
crm(live)configure#
primitive p_drbd_iscsivg01 ocf:linbit:drbd \
    params drbd_resource="iscsivg01" \
    op monitor interval="29" role="Master" \
    op monitor interval="31" role="Slave"
```

3）创建 ms_drbd_iscsivg01，基于上面创建的资源 p_drbd_iscsivg01，Master/Slave 使得 DRBD 资源 iscsivg01 在两个节点上同步。

```
primitive p_drbd_iscsivg01 ocf:linbit:drbd \
    params drbd_resource="iscsivg01" \
    op monitor interval="29" role="Master" \
    op monitor interval="31" role="Slave"
```

4）创建 crm 资源 p_ip_drbd-M，配置浮动 IP 10.0.0.200。

```
crm(live)configure#
primitive p_ip_alicebob01 ocf:heartbeat:IPaddr2 \
    params ip="10.0.0.200" cidr_netmask="24" \
    op monitor interval="10s"
```

5）创建 crm 资源 p_lvm_iscsivg01，定义 vg。

```
crm(live)configure#
ms ms_drbd_iscsivg01 p_drbd_iscsivg01 \
    meta master-max="1" master-node-max="1" clone-max="2" clone-node-max="1"
    notify="true"
colocation c_iscsivg01_on_drbd inf: rg_iscsivg01 ms_drbd_iscsivg01:Master
```

6）创建 crm 资源 p_target_iscsivg01，定义 iSCSI 的 target，要登入需要使用用户名 iscsi，密码为 1qa2ws3ed4rF。

```
crm(live)configure#
primitive p_target_iscsivg01 ocf:heartbeat:iscsiTarget \
```

1107114476@qq.com
dolphinebook.com

```
    params iqn="iqn.2001-04.com.example:storage.example.iscsivg01" tid="1"
    incoming_username="iscsi" incoming_password="1qa2ws3ed4rF" \
    op monitor interval="10s"
```

如果要用 tgt 的话，必须设置 numberric target id（TID）号，全局唯一就可以。

7）添加 lun。

创建 crm 资源 p_lu_iscsivg01_lun1，定义了一个 lv 的 lun1。

```
crm(live)configure#
primitive p_lu_iscsivg01_lun1 ocf:heartbeat:iSCSILogicalUnit \
    params target_iqn="iqn.2001-04.com.example:storage.example.iscsivg01"
    lun="1" path="/dev/iscsivg01/LUN1" scsi_id="1234567890123456" \
    op monitor interval="10s"
```

8）创建资源组。

创建资源组 group rg_iscsivg01，使 iSCSI 联动。

```
crm(live)configure#
group rg_iscsivg01 p_lvm_iscsivg01 p_target_iscsivg01 p_lu_iscsivg01_lun1 p_lu_
iscsivg01_lun2 p_ip_alicebob01 \
    meta target-role="Started"
ms ms_drbd_iscsivg01 p_drbd_iscsivg01 \
    meta master-max="1" master-node-max="1" clone-max="2" clone-node-max="1"
    notify="true"
colocation c_iscsivg01_on_drbd inf: rg_iscsivg01 ms_drbd_iscsivg01:Master
```

9）指定资源组默认在 DRBD 的主节点。

资源组默认是自动托管的，没有排序，需要指定资源组在 DRBD 的主节点上运行。

```
crm(live)configure#
order o_drbd_before_iscsivg01 inf: ms_drbd_iscsivg01:promote rg_iscsivg01:start
commit
```

10）清除 crm 的配置文件的方法。

如果要清除 crm 的配置文件，执行如下命令：

```
crm(live)configure # erase
crm(live)configure# commit
```

如果要清除 Pacemaker 的所有设置，执行如下命令：

```
/etc/rc.d/init.d/corosync stop
 rm -f /var/lib/heartbeat/crm/*
 /etc/rc.d/init.d/corosync start
```

11）最终的配置。

```
configure
property stonith-enabled="false"
property no-quorum-policy="ignore"
```

1107114476@qq.com
dolphinebook.com

```
property default-resource-stickiness="200"
commit
primitive p_drbd_iscsivg01 ocf:linbit:drbd \
    params drbd_resource="iscsivg01" \
    op monitor interval="29" role="Master" \
    op monitor interval="31" role="Slave"
primitive p_ip_alicebob01 ocf:heartbeat:IPaddr2 \
    params ip="10.0.0.200" cidr_netmask="24" \
    op monitor interval="10s"
primitive p_lu_iscsivg01_lun1 ocf:heartbeat:iSCSILogicalUnit \
    params target_iqn="iqn.2001-04.com.example:storage.example.iscsivg01"
    lun="1" path="/dev/iscsivg01/LUN1" scsi_id="1234567890123456" \
    op monitor interval="10s"
primitive p_lvm_iscsivg01 ocf:heartbeat:LVM \
    params volgrpname="iscsivg01" \
    op monitor interval="30s"
primitive p_target_iscsivg01 ocf:heartbeat:iSCSITarget \
    params iqn="iqn.2001-04.com.example:storage.example.iscsivg01" tid="1"
    incoming_username="iscsi" incoming_password="1qa2ws3ed4rF" \
    op monitor interval="10s"
group rg_iscsivg01 p_lvm_iscsivg01 p_target_iscsivg01 p_lu_iscsivg01_lun1 p_lu_
iscsivg01_lun2 p_ip_alicebob01 \
    meta target-role="Started"
ms ms_drbd_iscsivg01 p_drbd_iscsivg01 \
    meta master-max="1" master-node-max="1" clone-max="2" clone-node-max="1"
notify="true"
colocation c_iscsivg01_on_drbd inf: rg_iscsivg01 ms_drbd_iscsivg01:Master
order o_drbd_before_iscsivg01 inf: ms_drbd_iscsivg01:promote rg_iscsivg01:start
commit
```

（4）验证配置

通过 crm_mon 查看集群状态，可以看到两个节点在线，6 个资源启动。

```
...
2 Nodes configured, 2 expected votes
6 Resources configured
Online: [ host185 host188 ]
 Resource Group: rg_iscsivg01
    p_lvm_iscsivg01     (ocf::heartbeat:LVM):    Started host185
    p_target_iscsivg01 (ocf::heartbeat:iSCSITarget):    Started host185
    p_lu_iscsivg01_lun1         (ocf::heartbeat:iSCSILogicalUnit):      Started host185
    p_ip_alicebob01     (ocf::heartbeat:IPaddr2):        Started host185
 Master/Slave Set: ms_drbd_iscsivg01 [p_drbd_iscsivg01]
    Masters: [ host185 ]
    Slaves: [ host188 ]
```

关闭 host185，查看 host188 是否接管。

```
...
2 Nodes configured, 2 expected votes
6 Resources configured
```

1107114476@qq.com
dolphinebook.com

```
Online: [ host188 ]
OFFLINE: [ host185 ]
 Master/Slave Set: ms_drbd_iscsivg01 [p_drbd_iscsivg01]
    Masters: [ host188 ]
    Stopped: [ host185 ]
...
```

确实是生效了。

7. KVM 计算节点使用 DRBD 集群

（1）安装 iSCSI 客户端管理工具

在宿主机上通过 yum 命令安装 Linux 的 iSCSI 客户端。

```
yum install iscsi-initiator-utils
```

修改配置文件 /rtc/iscsi/iscsid.conf，将 iSCSI 用户密码添加上去。

```
# To enable CHAP authentication set node.session.auth.authmethod
# to CHAP. The default is None.
node.session.auth.authmethod = CHAP
# 启用 CHAP 认证协议
# To set a CHAP username and password for initiator
# authentication by the target(s), uncomment the following lines:
node.session.auth.username = iscsi
node.session.auth.password = 1qa2ws3ed4rF
# 输入登录 iSCSI 服务端的用户、密码
```

（2）发现和登录 iSCSI 服务端

通过 iscsiadm 命令发现和登录 iSCSI 服务端。

```
iscsiadm -m discovery -t st -p 10.0.0.200
iscsiadm -m node -p 10.0.0.200  -l
```

登录后会发现多出一块磁盘。

```
[root@host251 ~]# fdisk -l /dev/sde
Disk /dev/sde: 214.7 GB, 214748364800 bytes, 419430400 sectors
```

然后创建 pv、vg、lv 就可以给虚拟机使用了。

9.2.3　DRBD 在 KVM 虚拟化中的优化方案

DRBD 原理是通过网络的磁盘复制，因此要优化就是磁盘和网络方面，如图 9-2 所示。在生产环境使用，最好存储网络、DRBD 同步网络、生产网络能够独立。

（1）磁盘系统优化方案

- 使用尽量多的磁盘，磁盘越多性能越好。
- 使用带电池的 RAID 卡，带电池可以开启 RAID 卡的写缓存，有利于提高性能。

1107114476@qq.com
dolphinebook.com

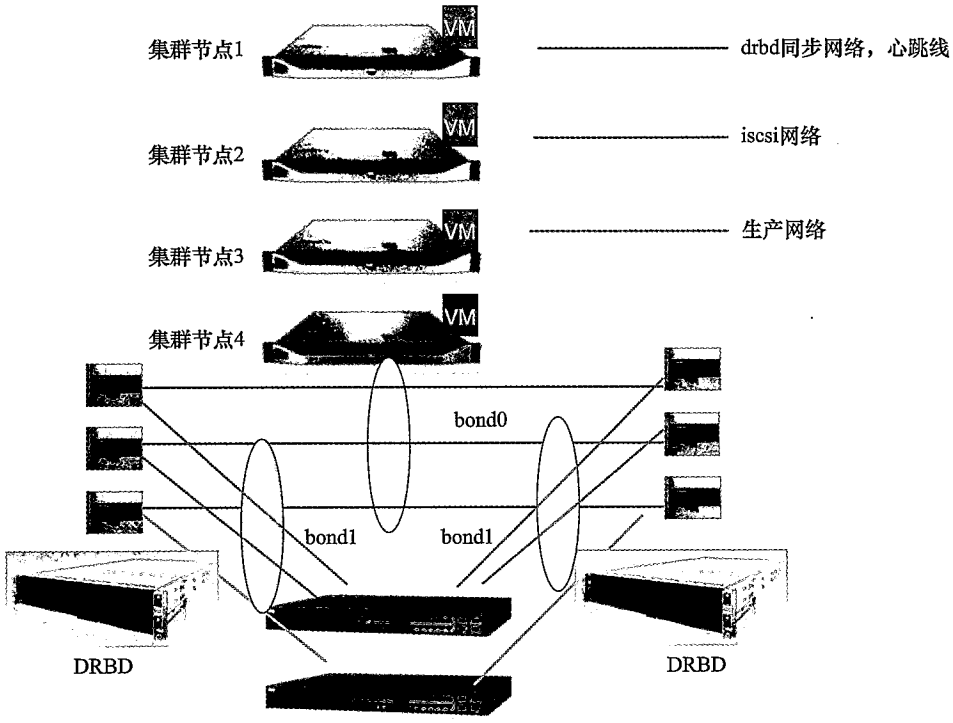


图 9-2 KVM DRBD 高可用方案示意图

- 使用 SSD 固态硬盘，SSD 固态硬盘可以有效地提升读 / 写性能，建议读者使用 5.5.3 节介绍的 Flashcache 的 SSD 分层方案，可以达到性能和成本比较好的平衡。
- 从容错、性能角度综合考虑，建议使用 RAID 10 方案，并配置两块热备磁盘。

（2）网络系统优化

- DRBD 同步使用单独网卡，并做多网卡绑定，使用 mode=0 方式（具体配置请参考 4.7 节介绍的多网卡绑定方法）。
- 使用万兆网卡。
- 修改 DRBD 同步网卡 mtu 为巨型帧 MTU=9000。
- iSCSI 网络使用单独网卡，并使用多路径技术（具体配置可以参考第 18 章）。

9.3 GlusterFS 在 KVM 虚拟化中的应用

9.3.1 GlusterFS 介绍

GlusterFS（GNU ClusterFile System，基于 GNU 协议的集群文件系统）是一款全对称的开源分布式文件系统，全对称的意思是 GlusterFS 采用弹性哈希算法，没有中心节点，所有的节点全部平等。GlusterFS 最大的特定是安装配置方便，并且稳定性非常好，可以轻松地

1107114476@qq.com
dolphinebook.com

达到 PB 级的容量，数千个节点。GlusterFS 于 2011 年 10 月被红帽公司收购，随后红帽公司推出了基于 GlusterFS 的 Red Hat Storage Server。因为 GlusterFS 和 KVM 都属于红帽公司，GlusterFS 被收购之后，增加了针对 KVM 的许多特性，并且获得了 QEMU 的原生支持。红帽公司发起的开源云管理平台 oVirt 也和 GlusterFS 之间相互支持（oVirt 在第 13 章有介绍）。

1. GlusterFS 特性

GlusterFS 特性如下：

- ❑ 全对称架构，部署使用简单。
- ❑ 弹性伸缩。
- ❑ 支持 Infiniband。
- ❑ 支持通过 SSL 证书加密的数据同步。
- ❑ 支持卷级别的压缩。
- ❑ 支持 FUSE。
- ❑ 支持 SMB。
- ❑ 支持 QEMU。
- ❑ 支持 NFS。
- ❑ 支持 Hadoop。
- ❑ 与 oVirt 深度整合。
- ❑ 支持 OpenStack。

GlusterFS 协议层面支持 IP 和 RDMA。RDMA（Remote Direct Memory Access）技术全称为远程直接数据存取，是为了解决网络传输中服务器端数据处理的延迟而产生的。RDMA 通过网络把资料直接传入计算机的存储区，将数据从一个系统快速移动到远程系统存储器中，而不对操作系统造成任何影响，这样就很少用到计算机的处理功能。它消除了外部存储器复制和文本交换操作，因而能解放内存带宽和 CPU 周期用于改进应用系统性能。

RDMA 最初是为 Infiniband 互连开发的技术。它在无须扩展缓存、干扰 CPU 或呼叫 OS 内核的情况下直接在两个系统间进行主数据转换，因此有助于缩短延时，提高系统间数据传输性能。

InfiniBand 架构是一种支持多并发链接的“转换线缆”技术，在这种技术中，每种链接都可以达到 2.5GB/s 的运行速度。这种架构在一个链接的时候速度是 500MB/s，4 个链接的时候速度是 2GB/s，12 个链接的时候速度可以达到 6GB/s。

InfiniBand 技术不是用于一般网络连接的，它的主要设计目的是针对服务器端的连接问题。因此，InfiniBand 技术被应用于服务器与服务器、服务器和存储设备以及服务器和网络之间的通信。

2. GlusterFS 的几个重要概念

- ❑ brick：GlusterFS 的基本单元，以节点服务器目录的形式展现。

1107114476@qq.com
dolphinebook.com

- 卷：多个 brick 的逻辑组合。
- Metadata：元数据，关于数据的数据，用于描述文件、目录等的信息。
- FUSE：Filesystem in Userspace，用户空间的文件系统，内核的特性，在用户态创建文件系统，而不需要修改内核。
- GlusterFS 服务器节点：数据存储服务器，GlusterFS 只有两种服务器角色，一种是数据存储服务器，一种是客户端，两种角色也可以在一台物理服务器上。
- 客户端：使用 GlusterFS 存储服务的服务器。
- Glusterd：GlusterFS 服务，所有存储节点需要运行。

3. GlusterFS 支持的卷类型

GlusterFS 支持如下几种卷类型。

（1）分布卷

分布卷也称为哈希卷，如图 9-3 所示，多个文件在多个 brick 上使用哈希算法随机存放。

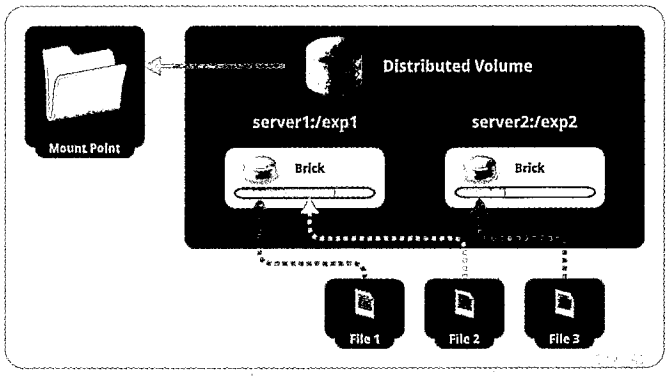


图 9-3 分布卷架构图（来自于 GlusterFS 官方网站）

- 使用场景：大量小文件。
- 优点：读 / 写性能好。
- 缺点：使用分布卷，存储的弹性扩展和冗余，需要其他层面的软件或者硬件解决。如果服务器或者磁盘故障，会导致分布在服务器或者磁盘上的数据丢失。

（2）复制卷

多个文件在多个 brick 上复制多份，如图 9-4 所示，brick 的数目要与需要复制的份数相等，并且 brick 最好是在不同的服务器上。

- 使用场景：对可靠性和读性能要求高的场景。
- 优点：读性能好。
- 缺点：写性能差，用磁盘空间换取可靠性。

（3）条带卷

条带卷将文件分成条带，存放在多个 brick 上，默认条带大小 128KB，条带卷主要存放大文件。

1107114476@qq.com
dolphinebook.com

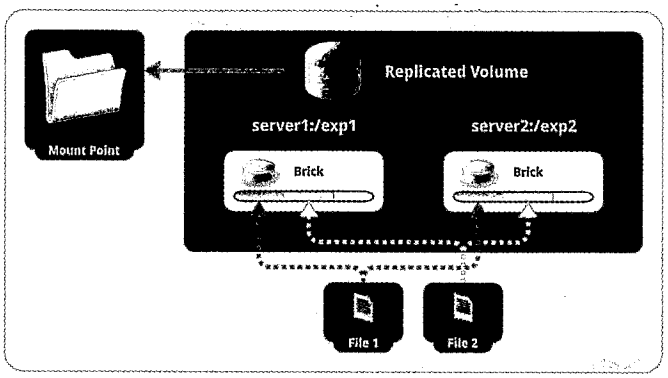


图 9-4　复制卷架构图（来自于 GlusterFS 官方网站）

- 使用场景：大文件。
- 优点：用条带的方法使大文件能够存储。
- 缺点：可靠性不高，有 brick 故障，数据全部丢失。

（4）分布条带卷

分布条带卷在一个集群内部，如图 9-5 所示，多个文件在多个节点哈希存储，每个文件再分条带在多个 brick 上存储。

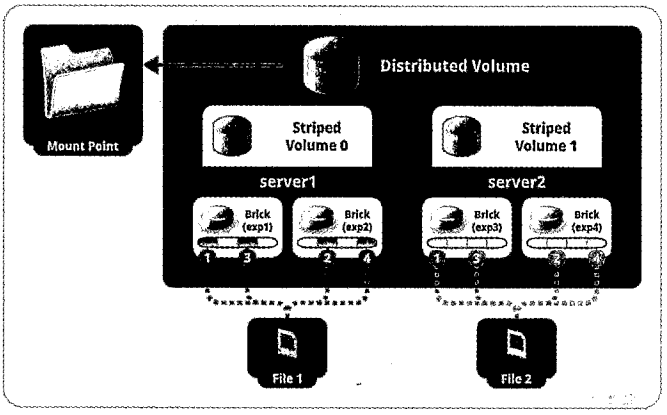


图 9-5　分布条带卷架构图（来自于 GlusterFS 官方网站）

- 使用场景：读 / 写性能高的大量大文件场景。
- 优点：高并发支持，存储池可以弹性扩展。
- 缺点：没有冗余，可靠性不高。

分布条带卷需要的 brick 数量是条带数的倍数。

（5）分布复制卷

分布复制卷如图 9-6 所示，多个文件在多个节点哈希存储，在多个 brick 复制多份存储。

1107114476@qq.com
dolphinebook.com

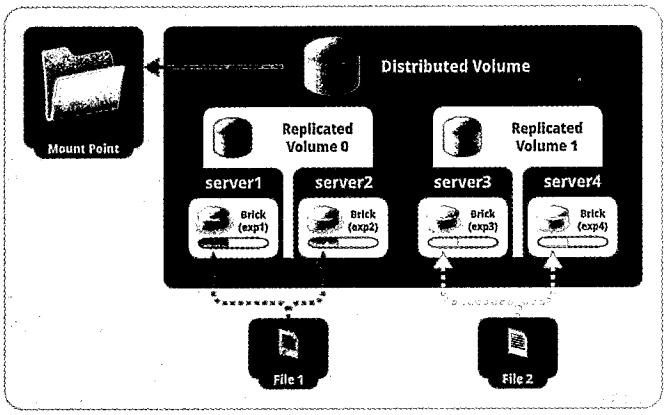


图 9-6 分布复制卷架构图（来自于 GlusterFS 官方网站）

- 使用场景：大量文件读和可靠性要求高的场景。
- 优点：存储空间弹性扩展，高可用，读性能高。
- 缺点：牺牲存储空间，写性能差。

分布复制卷需要 brick 的数量是复制数的倍数。另外，对 brick 的选择影响数据的安全性，同一台服务器有多个 brick 的时候，数据复制应该配置在多台服务器之间进行，而不是在同一台服务器内部之间进行，否则当服务器发生故障的时候，有可能会造成数据丢失。

（6）分布条带复制卷

分布条带复制卷如图 9-7 所示，多个文件在多个节点哈希存储，存储的时候划分条带，并且保存多份。

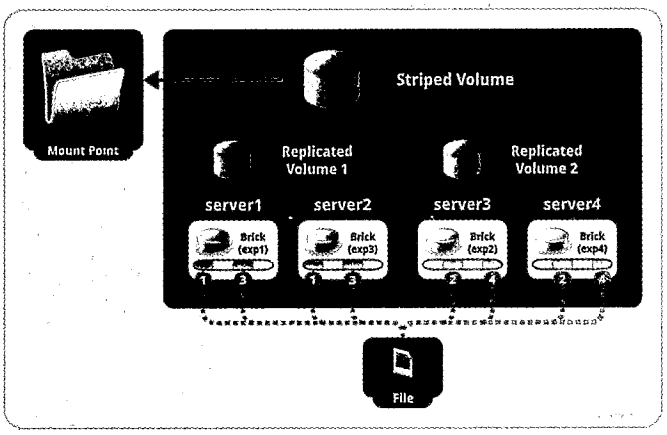


图 9-7 分布条带复制卷架构图（来自 GlusterFS 官方网站）

- 使用场景：多份大文件并且对可靠性要求高的场景，虚拟机的镜像存储就是一个典型场景。

1107114476@qq.com
dolphinebook.com

❑ 优点：读、写、可靠性比较均衡。

❑ 缺点：牺牲磁盘空间和写性能。

分布条带复制卷需要的 bircк 数量是条带数和复制数的乘积倍数。

（7）条带复制卷

条带复制卷，一个大文件存储的时候划分条带，并且保存多份。

❑ 使用场景：超大文件，并且对可靠性要求高的场景。

❑ 优点：使大文件能够存储，并且获得可靠性。

❑ 缺点：需要更大的磁盘空间，写性能差。

条带复制卷需要的 bircк 数量是条带数和复制数的乘积。

（8）分散卷

分散卷基于纠错码，它基于条带编码，添加冗余编码，并分布到多个 brick 上存储。分散卷以最小的磁盘空间消耗实现冗余，并且冗余级别可以配置。

❑ 应用场景：对冗余和磁盘空间都敏感的场景。

❑ 优点：在冗余和磁盘空间上取得平衡。

❑ 缺点：需要消耗额外的资源进行验证，对性能也有一定影响。

关于分散卷冗余，分散卷在创建的时候可以设置冗余数，冗余数取决于希望有多少 brick 出现问题的时候不引起数据丢失。冗余数决定了可用空间，可以使用下面的公式计算：

可用空间 =brick 大小 ×（brick 数量 – 冗余级别）

所有的 brick 需要相同的容量，如果不同，当最小的 brick 写满数据的时候，其他 brick 也就不能写数据。

如果是 3 个 brick 冗余数是 1，那么有 66.7% 的可用空间；如果是 10 个 brick，那么有 90% 的空间可以使用。但是 3 个 brick 的情况安全性要比 10 个 brick 的高，因为随着 brick 增加，bircк 出现问题的风险要加大。

（9）分布分散卷

分布分散卷，多个文件在多个节点上哈希分布存储，存储的时候基于条带编码并有冗余度设置。

❑ 应用场景：对冗余和磁盘空间都敏感的场景。

❑ 优点：在冗余、磁盘空间、读 / 写性能上取得平衡。

❑ 缺点：需要消耗额外的资源进行验证，对性能也有一定影响。

9.3.2　GlusterFS 集群搭建与使用

GlusterFS 集群搭建主要分为以下几个步骤：

❑ 存储节点、存储目录准备。

❑ GlusterFS 服务安装。

❑ 集群配置。

1107114476@qq.com
dolphinebook.com

- 卷创建及启用。
- 客户端挂载。

下面分别介绍一下每个步骤的操作方法。

1. 环境准备

环境准备如 9.1 节介绍，另外 6 台存储虚拟机每台挂载 SSD 上的 30GB 的 lv 3 个，在每台虚拟机上都可以看到 3 个 30GB 的磁盘，分别为 vdb、vdc、vdd。

2. gluster 目录准备

格式化 vdb、vdc、vdd 并分别挂载到 /export/vdb、/export/vdc、/export/vdd。以 vdb 上的操作为例，在每个节点上都需要执行如下命令：

```
mkfs.xfs -i size=512 /dev/vdb
mkdir -p /export/vdb && mount /dev/vdb /export/vdb
echo "/dev/vdb /export/vdb xfs defaults 0 0"  >> /etc/fstab
```

执行完成，通过 df 命令查看效果如下：

```
[root@host183 ~]# df -h
Filesystem               Size  Used Avail Use% Mounted on
/dev/vdd                  30G   33M   30G   1% /export/vdd
/dev/vdc                  30G   33M   30G   1% /export/vdc
/dev/vdb                  30G   33M   30G   1% /export/vdb
```

3. 安装 GlusterFS

可以通过 yum 命令直接安装，命令如下：

```
yum install gluster* -y
```

yum 安装的版本是 3.4，不是最新版本，可以下载官网最新版本的 rpm 包，然后安装。

```
wget -l 1 -nd -nc -r -A.rpm \
http://download.gluster.org/pub/gluster/glusterfs/3.6/LATEST/CentOS/epel-
7Server/x86_64
yum install gluster* -y
```

安装完成启动 glusterd 服务。

```
service glusterd start
```

4. 配置 GlusterFS 集群

通过在第一台存储节点服务器上执行 gluster peer probe 命令组建集群。

```
 [root@host183 ~]# gluster peer probe host184
peer probe: success.
[root@host183 ~]# gluster peer probe host185
peer probe: success.
```

1107114476@qq.com
dolphinebook.com

```
[root@host183 ~]# gluster peer probe host186
```

GlusterFS 的配置主要通过 gluster 命令操作，直接执行命令加参数可以操作。输入 gluster 命令按 Enter 键，进入 GlusterFS 的配置界面。输入 help 得到可以执行的命令。命令可以分为两个大类：一类是卷管理，包括卷的创建、删除、快照、状态查看等；一类是集群节点管理，包括节点添加、强制退出集群、状态查看等。

```
gluster> help
```

集群状态查看。

```
gluster> peer status
Number of Peers: 3
Hostname: host184
Uuid: 137e25eb-3886-4c79-aebd-7488505076ed
State: Peer in Cluster (Connected)
Hostname: host185
Uuid: 22ac500b-7284-4704-b807-e7d710e35f48
State: Peer in Cluster (Connected)
Hostname: host186
Uuid: 9f66f4fc-079b-48c0-87af-641a7652e399
State: Peer in Cluster (Connected)
```

5. 节点管理

1）添加新的节点。

在 host187、host188 节点上启动 glusterd 服务。

```
service glusterd start
```

然后在初始节点上执行探测命令。

```
gluster> peer probe host187
peer probe: success.
gluster> peer probe host188
peer probe: success.
```

查看节点状态。

```
gluster> peer stat
Number of Peers: 5
Hostname: host184
Uuid: 137e25eb-3886-4c79-aebd-7488505076ed
State: Peer in Cluster (Connected)
Hostname: host185
Uuid: 22ac500b-7284-4704-b807-e7d710e35f48
State: Peer in Cluster (Connected)
Hostname: host186
Uuid: 9f66f4fc-079b-48c0-87af-641a7652e399
State: Peer in Cluster (Connected)
```

1107114476@qq.com
dolphinebook.com

```
Hostname: host187
Uuid: d9e78eff-ff52-4e58-9524-16996339335a
State: Peer in Cluster (Connected)
Hostname: host188
Uuid: bcd2b2a7-0918-4c78-9c1d-7d45cdf10f43
State: Peer in Cluster (Connected)
```

2）删除节点。

强制 host188 退出集群。

```
gluster> peer detach host188
peer detach: success
```

可以通过 peer probe 再将 host188 添加进集群，这样在有存储节点故障的时候，可以通过删除再添加的方式进行修复。

6. 创建卷

1）创建分布卷，不指明卷类型，默认是分布卷。

```
gluster> volume create dis_vol host183:/export/vdc/dv1 host184:/export/vdc/dv1
host185:/export/vdc/dv1
volume create: dis_vol: success: please start the volume to access data
```

查看卷信息。

```
gluster> volume  info
Volume Name: dis_vol
Type: Distribute
Volume ID: 53bcf7bb-d1b0-40af-af15-da84643b6dc0
Status: Created
Number of Bricks: 3
Transport-type: tcp
Bricks:
Brick1: host183:/export/vdc/dv1
Brick2: host184:/export/vdc/dv1
Brick3: host185:/export/vdc/dv1
```

2）创建复制卷。

通过参数 replica 指定创建复制卷，复制卷的份数和 brick 数要相等。下面的命令每份数据存 3 份到 3 个 brick 上。

```
gluster> volume create rep_vol replica 3   host183:/export/vdc/rv2 host184:/export/
vdc/rv2 host185:/export/vdc/rv2
```

3）创建条带卷。

通过参数 stripe 指定创建条带卷。下面的命令将文件分条带存放到 3 个 brick 上。

```
gluster> volume create str_vol stripe 3   host183:/export/vdc/sv3 host184:/export/
vdc/sv3 host185:/export/vdc/sv3
```

1107114476@qq.com
dolphinebook.com

4）创建分布条带卷。

通过参数 stripe 和 brick 数量来创建分布条带卷，brick 数量是条带参数的倍数。下面的命令创建一个有 8 个 brick 的分布条带卷。

```
gluster> volume create dir_str_vol stripe 4   host183:/export/vdc/dsv4 host184:/
export/vdc/dsv4 host185:/export/vdc/dsv4 host186:/export/vdc/dsv4 host187:/export/
vdc/dsv4 host188:/export/vdc/dsv4 host183:/export/vdd/dsv4  host184:/export/vdd/dsv4
volume create: dir_str_vol: success: please start the volume to access data
```

查看一下卷信息。

```
gluster> volume info dir_str_vol
Volume Name: dir_str_vol
Type: Distributed-Stripe #分布条带卷
Volume ID: 6279b25d-3b38-4152-9b7e-c67f9dd2fbe6
Status: Created
Number of Bricks: 2 x 4 = 8
#文件存储的时候在 4 个 brick 上划分条带存储，多个文件在两组 brick 上哈希分布
Transport-type: tcp
Bricks:
Brick1: host183:/export/vdc/dsv4
Brick2: host184:/export/vdc/dsv4
Brick3: host185:/export/vdc/dsv4
Brick4: host186:/export/vdc/dsv4
Brick5: host187:/export/vdc/dsv4
Brick6: host188:/export/vdc/dsv4
Brick7: host183:/export/vdd/dsv4
Brick8: host184:/export/vdd/dsv4
```

5）创建分布复制卷。

通过参数 replica 和 brick 数量来创建分布复制卷，brick 数量是 replica 参数的倍数。下面的命令创建一个有 8 个 brick 的分布复制卷。

```
gluster> volume create dir_rep_vol replica 2   host183:/export/vdc/drv5 host184:/
export/vdc/drv5 host185:/export/vdc/drv5 host186:/export/vdc/drv5 host187:/export/
vdc/drv5 host188:/export/vdc/drv5 host183:/export/vdd/drv5  host184:/export/vdd/drv5
```

查看一下卷信息。

```
gluster> volume info dir_rep_vol
Volume Name: dir_rep_vol
Type: Distributed-Replicate #分布复制卷
Volume ID: ba927421-7f38-4e58-bf9e-2a3460a07617
Status: Created
Number of Bricks: 4 x 2 = 8
#多个文件存储的时候在 4 个 brick 上哈希存储，每个文件存储 2 份
...
```

6）创建分布条带复制卷。

通过 stripe、replica 参数及 brick 数量创建分布条带复制卷，brick 的数量需要是 stripe、

1107114476@qq.com
dolphinebook.com

replica 乘积的倍数。下面的命令创建一个有 8 个 brick 的分布条带复制卷。

```
gluster> volume create dis_str_rep_vol stripe 2 replica 2   host183:/export/vdc/
dsrv6 host184:/export/vdc/dsrv6 host185:/export/vdc/dsrv6 host186:/export/vdc/dsrv6
host187:/export/vdc/dsrv6 host188:/export/vdc/dsrv6 host183:/export/vdd/dsrv6
host184:/export/vdd/dsrv6
```

查看一下卷信息。

```
gluster> volume info dis_str_rep_vol
Volume Name: dis_str_rep_vol
Type: Distributed-Striped-Replicate # 分布条带复制卷
Volume ID: 76f63aea-fe7b-437a-b40d-ffc738cd20ee
Status: Created
Number of Bricks: 2 x 2 x 2 = 8
# 多个文件存储的时候在 4 组 brick 上哈希存储，每个文件存储 2 份，并且划分条带
...
```

7）创建条带复制卷。

通过 stripe、replica 参数及 brick 数量创建分布条带复制卷，brick 的数量需要是 stripe、replica 的乘积。下面的命令创建一个有 4 个 brick 的分布条带复制卷。

```
gluster> volume create str_rep_vol stripe 2 replica 2   host183:/export/vdc/
srv7 host184:/export/vdc/srv7 host185:/export/vdc/srv7 host186:/export/vdc/srv7
```

8）创建分散卷。

分散卷需要指定冗余 brick 数，冗余数要小于 brick 数量。下面的命令创建一个 4 个 bircк 冗余的分散数，最多允许 1 个 brick 故障而不丢失数据。

```
gluster> volume create disperse_vol disperse  4 host183:/export/vdc/d8 host184:/
export/vdc/d8 host185:/export/vdc/d8 host186:/export/vdc/d8
There isn't an optimal redundancy value for this configuration. Do you want to
create the volume with redundancy 1 ? (y/n) y
volume create: disperse_vol: success: please start the volume to access data
```

查看分散卷的信息。

```
gluster> volume info disperse_vol
Volume Name: disperse_vol
Type: Disperse
Volume ID: 9676d60d-590f-42c0-8d76-97bb63c5ef9d
Status: Created
Number of Bricks: 1 x (3 + 1) = 4
...
Transport-type: tcp
```

9）创建分布分散卷。

分散卷需要指定分散数和冗余数，如果分散数大于 brick 数，就是分布分散卷，冗余数也可以不指定，GlusterFS 有默认的算法。下面的命令创建一个 5 个 bircк 分布分散卷，最多

1107114476@qq.com
dolphinebook.com

允许 2 个 brick 故障而不丢失数据。

```
gluster> volume create disperse_vol3  disperse  3 host183:/export/vdc/d10 host184:/
export/vdc/d10 host185:/export/vdc/d10 host186:/export/vdc/d10 host187:/export/vdc/
d10 host188:/export/vdc/d10
volume create: disperse_vol3: success: please start the volume to access data
```

查看分布分散卷信息。

```
gluster> volume info disperse_vol3
Volume Name: disperse_vol3
Type: Distributed-Disperse
Volume ID: a8abbf71-5b83-417e-9d26-ea1b3e612814
Status: Created
Number of Bricks: 2 x (2 + 1) = 6
Transport-type: tcp
Bricks:
Brick1: host183:/export/vdc/d10
...
```

7. 卷管理

1）删除卷操作。

```
gluster> volume delete rep_vol
Deleting volume will erase all information about the volume. Do you want to
continue? (y/n) y
volume delete: rep_vol: success
```

2）启动卷操作。

```
gluster volume start 卷名称
```

3）通过 gluster 协议挂载卷。

```
mkdir -p /gfs
mount -t glusterfs host183:/test-volume /gfs
```

自动挂载，添加新到 fstab，因为是网络文件系统，需要添加 _nerdev 参数。

```
vim fstab
server1:/test-volume /mnt/glusterfs glusterfs defaults,_netdev 0 0
```

4）使用 nfs 方式挂载。

```
mount -t nfs host183:/kvm-test /mnt
df -h
Filesystem                    Size  Used Avail Use% Mounted on
host183:/kvm-test             180G   12G  169G   7% /mnt
```

9.3.3　GlusterFS 在 KVM 虚拟化中的应用

建议在生产环境中使用分布条带复制卷，并且每个虚拟机镜像文件保存 3 份，这样在性

1107114476@qq.com
dolphinebook.com

能、稳定性、安全性方面能达到比较好的平衡。GlusterFS 在 KVM 虚拟化中的应用有两种方法：一种方法是通过 QEMU 直接调用；一种是先将卷挂载到宿主机，然后像宿主机本地存储一样使用。

1. 为宿主机创建分布条带复制卷

下面的命令创建具有 18 个 brick 的分布条带复制卷，用于虚拟机的镜像创建。

```
gluster> volume create kvm-test stripe 3 replica 3 host183:/export/vdb/k1 host184:/
export/vdb/k1 host185:/export/vdb/k1 host186:/export/vdb/k1  host187:/export/vdb/
k1 host188:/export/vdb/k1 host183:/export/vdc/k1 host184:/export/vdc/k1 host185:/
export/vdc/k1 host186:/export/vdc/k1  host187:/export/vdc/k1 host188:/export/vdc/k1
host183:/export/vdd/k1 host184:/export/vdd/k1 host185:/export/vdd/k1 host186:/
export/vdd/k1  host187:/export/vdd/k1 host188:/export/vdd/k1
```

GlusterFS 的条带、分布和 brick 添加时候的顺序有关系，为了尽量容错，添加的时候，应将不同服务器上的 brick 交叉添加。

将卷启动，就可以使用了。

```
gluster> volume start kvm-test
```

通过 gluster> volume status 命令可以查看卷状态。

```
gluster> volume status kvm-test
Status of volume: kvm-test
Gluster process                                         Port    Online    Pid
------------------------------------------------------------------------------
Brick host183:/export/vdb/k1                            49153     Y       32298
...
Brick host188:/export/vdd/k1                            49154     Y       31873
NFS Server on localhost                                 2049      Y       32332
Self-heal Daemon on localhost                           N/A       Y       32340
...
NFS Server on host186                                   2049      Y       32148
Self-heal Daemon on host186                             N/A       Y       32156
Task Status of Volume kvm-test
There are no active volume tasks
```

要使用的宿主机一定要能通过域名解析 glusterd 服务器节点。

2. 通过 QEMU 使用 GlusterFS

（1）通过 qemu-img 创建镜像

QEMU 从 1.3 版本开始就支持 GlusterFS，用 qemu-img create 命令指明使用 Gluster 协议、TCP 方式、GlusterFS 服务器地址端口、GlusterFS 卷名称、需要创建虚拟机镜像的名称

1107114476@qq.com
dolphinebook.com

就可以，命令如下：

```
/usr/bin/qemu-img create    gluster+tcp://10.0.0.183:24007/kvm-test/gfs-kvm
100G -f qcow2
Formatting 'gluster+tcp://10.0.0.183:24007/kvm-test/gfs-kvm', fmt=qcow2
size=107374182400 encryption=off cluster_size=65536 lazy_refcounts=off
```

同样，通过 qemu-img info 命令可以查看一下镜像信息。

```
/usr/bin/qemu-img info    gluster+tcp://10.0.0.183:24007/kvm-test/gfs-kvm
image: gluster+tcp://10.0.0.183:24007/kvm-test/gfs-kvm
file format: qcow2
virtual size: 100G (107374182400 bytes)
disk size: 194K
cluster_size: 65536
Format specific information:
    compat: 1.1
    lazy refcounts: false
```

（2）通过 qemu-kvm 命令启动虚拟机

通过 qemu-kvm 命令可以直接启动虚拟机。

```
/usr/libexec/qemu-kvm  --drive \
file=gluster://10.0.0.183:24007/kvm-test/gfs-kvm
VNC server running on '::1:5900'
```

（3）通过 Libvirt 使用虚拟机

要通过 Libvirt 使用虚拟机，需要配置 GlusterFS。

修改 etc/glusterfs/glusterd.vol，添加以下内容，然后重启 Glusterd 服务。

```
option rpc-auth-allow-insecure on
```

打开卷 server.allow-insecure 属性。

```
gluster volume set kvm-test server.allow-insecure on
```

设置卷 UID 和 GID。

```
gluster volume set kvm-test storage.owner-uid 107
gluster volume set kvm-test storage.owner-gid 107
```

停止卷再启动卷。

```
gluster  volume stop kvm-test
gluster  volume start kvm-test
```

检查以上配置是否生效。

```
gluster> volume info kvm-test
...
Options Reconfigured:
```

1107114476@qq.com
dolphinebook.com

```
storage.owner-gid: 107
storage.owner-uid: 107
server.allow-insecure: on
```

然后修改虚拟机的 .xml 文件中的磁盘文件。

```
<disk type='network' device='disk'>
# 磁盘格式为 network
    <driver name='qemu' type='qcow2' cache='none'/>
    <source protocol='gluster' name='kvm-test/gfs-kvm'>
# 指定协议为 Gluster，指定卷名称和镜像名称
        <host name='host183' port='24007'/>
# 指定 Glusterd 服务，主机名和 IP 都可以，指定 TCP 端口
        </source>
        <target dev='vda' bus='virtio'/>
        <alias name='virtio-disk0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </disk>
```

然后就可以使用虚拟机了。

3. 通过 fuse 使用 GlusterFS

fuse 方式使用非常简单，通过 mount 命令挂载就可以。

```
mount -t glusterfs host183:/kvm-test /mnt
ls /mnt
gfs-kvm
```

可以看到通过 qemu-img 创建的镜像，通过 df 命令，可以看到挂载的分区。

```
df -h
Filesystem                  Size  Used Avail Use% Mounted on
host183:/kvm-test           180G  194M  180G   1% /mnt
```

要使用镜像，和普通的本地存储挂载一模一样，这里就不介绍了。

在生产环境中建议通过 fuse 的方式使用 GlusterFS，QEMU 直接调用 GlusterFS 的方式虽然性能好一些，但是不同的 GlusterFS 版本、QEMU 版本、文件系统之间经常会遇到兼容性问题。

9.3.4 GlusterFS 在 KVM 虚拟化中的优化方案

GlusterFS 是基于网络的分布式文件系统，因此要优化就是磁盘和网络方面。GlusterFS 高可用方案示意图如图 9-8 所示，由多台存储服务器组成 GlusterFS 存储集群，供计算节点调用，存储服务器网络建议是万兆或者多网卡绑定。

（1）磁盘系统优化方案

❑ 使用尽量多的磁盘，总体性能可以随着磁盘增加而线性增长。

1107114476@qq.com
dolphinebook.com

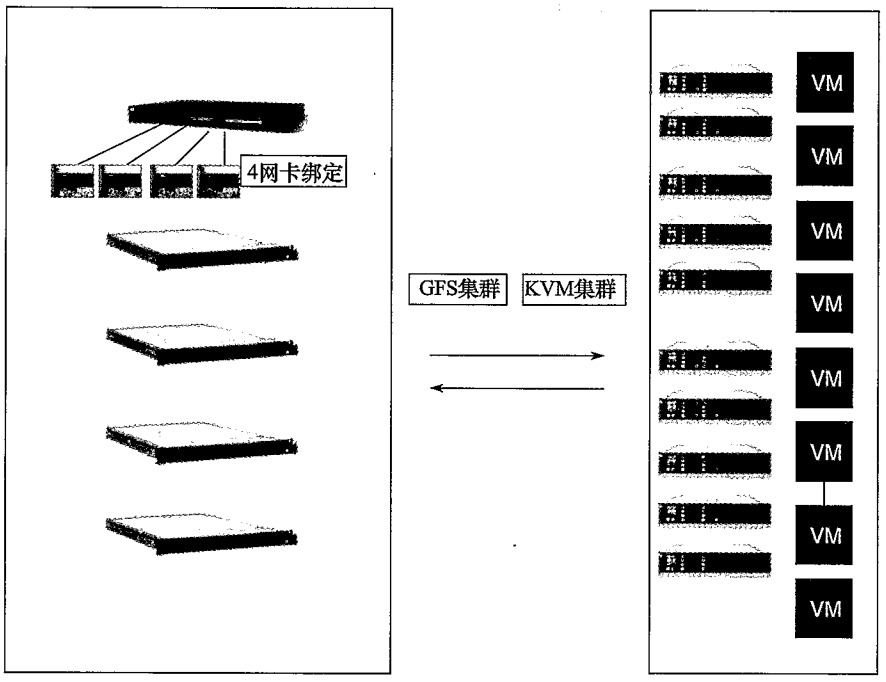


图 9-8　GlusterFS 高可用方案示意图

- 因为 GlusterFS 可以在应用层面容错，使用单块磁盘，不作 RAID，可以在容量、可靠性、性能之间取得平衡。
- 使用 SSD 固态硬盘，SSD 固态硬盘可以有效地提升读 / 写性能。

（2）网络系统优化

同 DRBD，详见 9.2.3 节。

9.4　Sheepdog 在 KVM 中的应用

Sheepdog 是一款分布式对象存储系统，最早是日本 NTT 公司发起的开源项目，国内的淘宝内核团队也贡献了大量的代码。Sheepdog 是为 KVM 虚拟化量身定做的分布式文件系统，针对 KVM 做了深入的优化。Sheepdog 的特点是使用方便、代码简单，可以非常容易地扩展到数千个节点。

9.4.1　Sheepdog 介绍

1. Sheepdog 的特性

Sheepdog 是完全对称的架构，没有元数据服务器，每个节点完全平等，主要特性如下。

- 只有两个服务，部署管理方便。

1107114476@qq.com
dolphinebook.com

- 性能和容量可以线性伸缩：当容量或者性能需要的时候，Sheepdog 可以通过增加节点线性地进行扩展。
- 没有单点故障：任何一个节点故障，数据都不会丢失。
- 管理方便：没有专门的管理角色，当新的服务器增加并启动 Sheepdog 服务的时候，Sheepdog 可以自动探测并加入存储系统。
- QEMU 原生支持，支持所有的 QEMU 镜像。
- 灵活可选的冗余级别，可以基于集群、节点的多份镜像策略。
- 支持镜像快照、克隆、大小调整、大小预分配、冗余策略调整、备份。
- 支持集群级别的快照。
- 支持虚拟机在线迁移。
- 支持专用的网卡用于数据同步。
- 被 OpenStack 支持。
- 支持 iSCSI、NBD、块设备方式提供存储服务。

2. Sheepdog 的组件

Sheepdog 包含以下组件。

1）对象存储：Sheepdog 服务（也称为 sheep 服务）是为 QEMU 创建的分布式对象存储系统。对象存储是固定大小的数据并且有全局唯一的标识。通过全局唯一的标识可以读取、写入、创建、删除对象存储。存储对象由网关和对象管理组成。每个对象存储有 64 字节的唯一标识，并且在多节点复制。

2）网关：网关从 QEMU 驱动接收 I/O 请求，通过哈希算法计算目标节点，并将 I/O 请求发送回目标节点。

3）对象管理：对象管理网关接收 I/O 请求，并在自己的本地磁盘执行读 / 写操作。

4）集群管理：集群管理管理节点成员，探测失败添加的节点，并且探知节点的变化，一些操作需要在所有节点之间同步，比如镜像创建、快照等。

5）QEMU 块驱动：QEMU 块驱动将虚拟机的镜像切割为固定的块大小，默认大小是 4MB，并且通过网关将数据块存储在存储对象上。QEMU 块驱动不关心如何存储对象，对象存储系统负责管理如何存储对象。

3. 对象的类型

Sheepdog 对象有 4 种类型。

1）数据对象：包含虚拟机实际的数据。虚拟磁盘镜像被分为固定大小的数据镜像。Sheepdog 客户端归根结底访问的是这些镜像。

2）vdi 对象：虚拟磁盘的元数据，比如磁盘名字、磁盘尺寸、创建时间、数据对象 ID 等。

3）虚拟机状态对象：存储运行虚拟机镜像状态，在管理员创建在线快照的时候使用。

1107114476@qq.com
dolphinebook.com

4）vdi 属性镜像：可以存储每个 vdi 的属性，通过 vdi 属性镜像存储，属性是 key-value 形式。

9.4.2　Sheepdog 搭建方法及在 KVM 中的应用

Sheepdog 的配置过程分为以下几个步骤。

Sheepdog 节点上的配置：

- 配置 hostname。
- 配置 corosync。
- 编译 Sheepdog。

宿主机节点上的配置：

- 编译 QEMU。
- 创建虚拟机，挂载 Sheepdog 文件系统。

下面介绍一下每个步骤的配置方法。

1. 环境准备

环境准备参见 9.1 节介绍，另外 6 台存储虚拟机每台挂载 SSD 上的 90GB 的 lv。

2. 安装配置 corosync

corosync 可以直接通过 yum 命令安装，也可以使用编译安装。通过 yum 命令安装，需要安装 CentOS 7 的 epel 源。笔者通过编译方式安装，安装很简单，下载编译就可以。

```
wget http://corosync.org/download/corosync-2.3.4.tar.gz
tar -zxvf corosync-2.3.4.tar.gz
cd corosync-2.3.4/
./configure
make
make install
```

安装之后配置 corosync.conf 文件，文件内容可以参考 Sheepdog 官网 wiki，关于 corosync 的配置地址是：https://github.com/Sheepgog/Sheepdog/wiki/Corosync-config。下面是笔者的 corosync.conf 文件内容。

```
cat << EOF >/etc/corosync/corosync.conf
cat /etc/corosync/corosync.conf
compatibility: whitetank
totem {
  version: 2
  secauth: off
  threads: 0
  # Note, fail_recv_const is only needed if you're
  # having problems with corosync crashing under
  # heavy Sheepdog traffic. This crash is due to
  # delayed/resent/misordered multicast packets.
```

1107114476@qq.com
dolphinebook.com

```
  # fail_recv_const: 5000
  interface {
    ringnumber: 0
    bindnetaddr: 10.0.0.0
    #注意这里是所有节点网段地址，不是节点IP地址
    mcastaddr: 226.94.1.1
    mcastport: 5405
  }
}
logging {
  fileline: off
  to_stderr: no
  to_logfile: yes
  to_syslog: yes
  # the pathname of the log file
  logfile: /var/log/cluster/corosync.log
  debug: off
  timestamp: on
  logger_subsys {
    subsys: AMF
    debug: off
  }
}
amf {
  mode: disabled
}
EOF
```

然后启动 corosync 服务。

```
/etc/init.d/corosync restart
```

查看日志，如果能看到各个节点之间的通信信息，则说明配置成功。

```
tail -f /var/log/cluster/corosync.log
Jan 14 19:34:42 [25485] host185 corosync notice  [TOTEM ] A new membership
(10.0.0.185:188) was formed. Members joined: 169109177
Jan 14 19:34:42 [25485] host185 corosync notice  [MAIN  ] Completed service
synchronization, ready to provide service.
Jan 14 19:34:42 [25485] host185 corosync notice  [TOTEM ] A new membership
(10.0.0.183:192) was formed. Members joined: 169109175 169109176
Jan 14 19:34:42 [25485] host185 corosync notice  [MAIN  ] Completed service
synchronization, ready to provide service.
```

3. Sheepdog 安装

Sheepdog 默认文件存储目录是 /var/lib/Sheepdog，创建目录并将第二块硬盘挂载到这个目录。

```
mkdir -p /var/lib/Sheepdog
mkfs.xfs /dev/sdb
mount /dev/sdb /var/lib/Sheepdog
```

1107114476@qq.com
dolphinebook.com

Sheepdog 安装通过源代码，笔者安装的版本是 0.9.50。

```
wget https://github.com/sheepdog/sheepdog/archive/master.zip
unzip master.zip
cd /root/sheepdog-master
./autogen.sh
./configure
make install
```

也可以编译成 rpm 包，方便批量安装，使用 make rpm 命令就可以。

```
make rpm
```

启动服务：

```
sheep /var/lib/Sheepdog
```

sheep 命令是 Sheepdog 服务的管理命令，主要参数如下。

- -b, --bindaddr——绑定特定的 IP 或者接口。
- -c, --cluster——指定特定的集群底层驱动，默认是 corosync。

提示　Sheepdog 的底层集群驱动有两个：一个是 corosync，corosync 是全对称结构，无中心节点；还有一个是 zookeeper，使用 zookeeper 需要先搭建 zookeeper 集群，然后 sheep 服务通过 IP 端口调用 zookeeper 集群服务。corosync 在 15 节点以下的情况支持得很好，zookeeper 可以支持到上千个节点。

- -D, --directio——直接 I/O，绕过文件系统缓存。
- -g, --gateway——网关，使本节成为网关模式。
- -i, --ioaddr——同步 Sheepdog 数据时使用指定的网卡，默认关闭。
- -p, --port——指定 TCP 端口，默认是 7000。
- -z, --zone——指定 zone ID，默认根据服务监听地址判断。通过 zone 可以建立多个 Sheepdog 集群。

4. 配置 Sheepdog 集群

集群管理主要通过 dog 命令（早期使用的是 collie 命令，目前也可以使用）。dog 命令常用参数如下。

- vdi check——检查修复镜像。
- vdi create——创建镜像。
- vdi snapshot——创建镜像快照。
- vdi clone——镜像克隆。
- vdi delete——删除镜像。

1107114476@qq.com
dolphinebook.com

- vdi rollback——回滚镜像到指定快照。
- vdi list——列出所有镜像。
- vdi resize——修改镜像大小。
- vdi alter-copy——设置镜像冗余级别。
- vdi lock——锁定或者解锁镜像。
- node kill——删除节点。
- node list——列出所有节点。
- node info——显示节点信息。
- cluster info——显示集群信息。
- cluster format——创建集群。
- cluster shutdown——停止集群。
- cluster snapshot——创建快照、回滚集群快照。
- cluster check——检查修复集群。
- cluster alter-copy——修改集群冗余级别。

（1）Sheepdog 集群管理

创建一个 Sheepdog 集群，默认冗余策略是 3 份。

```
dog cluster format --copies=3
```

copies 是镜像冗余数，也可以在后台创建。

```
collie cluster format -b farm --copies=3
```

查看集群节点状态。

```
dog node list
  Id   Host:Port         V-Nodes       Zone
   0   10.0.0.183:7000       128 3076920330
   1   10.0.0.184:7000       128 3093697546
   2   10.0.0.185:7000       128 3110474762
```

查看集群信息。

```
dog cluster  info
Cluster status: running, auto-recovery enabled
Cluster created at Wed Jan 14 19:52:21 2015
Epoch Time          Version
2015-01-14 19:52:21     1 [10.0.0.183:7000, 10.0.0.184:7000, 10.0.0.185:7000]
```

停止集群。

```
collie cluster shutdown
```

添加节点。节点添加很简单，在新加的节点上启动 sheep 服务就可以，Sheepdog 会自动通过。笔者又启动了 3 个节点的 sheep 服务，可以看到节点达到了 6 个。

1107114476@qq.com
dolphinebook.com

```
dog node list
  Id   Host:Port         V-Nodes       Zone
   0   10.0.0.183:7000       128 3076920330
   1   10.0.0.184:7000       128 3093697546
   2   10.0.0.185:7000       128 3110474762
   3   10.0.0.186:7000       128 3127251978
   4   10.0.0.187:7000       128 3144029194
   5   10.0.0.188:7000       128 3160806410
```

查看集群信息，可以看到集群是通过版本来标记节点变化的。

```
[root@host188 ~]# dog cluster info
Cluster status: running, auto-recovery enabled
Cluster created at Wed Jan 14 19:52:21 2015
Epoch Time          Version
2015-01-14 21:46:09     14 [10.0.0.183:7000, 10.0.0.184:7000, 10.0.0.185:7000,
10.0.0.186:7000, 10.0.0.187:7000, 10.0.0.188:7000]
...
2015-01-14 21:32:02      3 [10.0.0.183:7000]
2015-01-14 20:41:53      2 [10.0.0.183:7000, 10.0.0.184:7000]
2015-01-14 19:52:21      1 [10.0.0.183:7000, 10.0.0.184:7000, 10.0.0.185:7000]
```

通过 dog node info 命令可以查看每个节点的空间使用情况。

```
dog node info
Id      Size    Used    Avail   Use%
 0      45 GB   11 GB   33 GB    25%
 1      44 GB   12 GB   33 GB    25%
 2      45 GB   11 GB   34 GB    25%
 3      45 GB   9.1 GB  36 GB    20%
 4      45 GB   11 GB   34 GB    23%
 5      46 GB   6.4 GB  39 GB    14%
Total   269 GB  60 GB   209 GB   22%
Total virtual image size        63 GB
```

通过 dog node stat 命令可以查看集群读 / 写速度。

```
dog node stat
Request Active  Total   Write   Read    Remove  Flush   All WR  All RD
Client  0       6       6       0       0       0       0.0 MB  0.0 MB
Peer    0       0       0       0       0       0       0.0 MB  0.0 MB
```

通过 dog node kill 命令可以删除节点。

```
dog node kill  5
```

（2）Sheepdog 镜像管理

创建名为 test1、大小为 20GB 的镜像。

```
[root@host183 ~]# dog vdi create test1 20G
```

列出所有镜像。

1107114476@qq.com
dolphinebook.com

```
[root@host183 ~]# dog vdi list
  Name        Id    Size    Used  Shared    Creation time   VDI id  Copies  Tag
  Block Size Shift
  test1        0   20 GB  0.0 MB  0.0 MB 2015-01-14 20:00   fd32fc       3   22
```

因为 QEMU 原生支持 Sheepdog，所以通过 qemu-img 命令也可以创建 Sheepdog 镜像。并将创建好的镜像导入到 Sheepdog 集群中。

```
[root@host183 ~]# qemu-img convert test3 Sheepgog:test3
[root@host183 ~]# qemu-img create -f qcow2 test3 22G
Formatting 'test3', fmt=qcow2 size=23622320128 encryption=off cluster_
size=65536 lazy_refcounts=off
[root@host183 ~]# dog vdi list
  Name        Id    Size    Used  Shared    Creation time   VDI id  Copies  Tag
  Block Size Shift
  test1        0   20 GB  0.0 MB  0.0 MB 2015-01-14 20:00   fd32fc       3   22
  test3        0   22 GB   22 GB  0.0 MB 2015-01-14 20:02   fd3662       3   22
```

通过 qemu-img 命令创建镜像快照。

```
qemu-img snapshot -c test1s1 Sheepgog:test1
```

同样，qemu-img 的其他镜像管理命令 Sheepdog 都支持，这里就不详细介绍了。

5. 宿主机使用 Sheepdog

（1）编译 QEMU

因为 CentOS 7 默认携带的 QEMU 编译的时候没有开启对 Sheepdog 的支持，要使用 Sheepdog，需要重新编译 QEMU。

QEMU 安装如下：

```
wget http://wiki.qemu-project.org/download/qemu-2.2.0.tar.bz2
tar -xvf qemu-2.2.0.tar.bz2
cd qemu-2.2.0/
ls
./configure --block-drv-rw-whitelist=qcow2,raw,file,host_device,nbd,iSCSI,gluster,
rbd,Sheepgog
make
make install
```

编译的时候需要加上如下参数：--block-drv-rw-whitelist=qcow2,raw,file,host_device,nbd,iSCSI,gluster,rbd,Sheepdog，否则使用的时候会出现“Driver 'sheepdog' is not whitelisted”的错误。

（2）宿主机上通过 QEMU 使用 Sheepdog

将宿主机镜像远程转到 Sheepdog。

```
/usr/local/bin/qemu-img convert centos66.qcow2 \
```

1107114476@qq.com
dolphinebook.com

```
Sheepgog:10.0.0.183:7000:centos66-189
```

查看 Sheepdog 镜像信息。

```
qemu-img info Sheepgog:10.0.0.183:7000:test3
image: Sheepgog:10.0.0.183:7000:test3
file format: raw
virtual size: 22G (23622320128 bytes)
disk size: 22G
```

通过 qemu 命令，使用镜像启动虚拟机。

```
/usr/local/bin/qemu-system-x86_64  -name Sheepdogtest  -smp \
4,sockets=4,cores=1,threads=1  -vnc :10    --drive  \
file=Sheepgog:10.0.0.183:7000:centos66-189
```

如果遇到如下错误：

```
qemu-system-x86_64: --drive file=Sheepgog:10.0.0.183:7000:centos66-189: could
not open disk image Sheepgog:10.0.0.183:7000:centos66-189: cannot get vdi info,
VDI is already locked, centos66-189 0
```

在 Sheepdog 节点上取消镜像锁定，这是一个 bug，待修复。

```
dog vdi lock unlock centos66-189
```

（3）宿主机上通过 Libvirt 使用 Sheepdog

通过 Libvirt 使用 Sheepdog 镜像，.xml 配置文件如下：

```
<disk type='network' device='disk'>
      <driver name='qemu'/>
      <source protocol='Sheepdog name='centos66-189'>
       <hostname='10.0.0.183' port='7000'/>
      </source>
      <target dev='vda' bus='virtio'/>
    </disk>
```

说明：通过 source protocol 关键字指定使用 Sheepdog，通过 name 关键字指定镜像名称为 centos66-189，通过 hostname 关键字指定 Sheepdog 集群地址，通过 port 关键字指定集群服务端口。

Sheepdog 理念非常好，架构简单，这几年在不断地发展，但是目前成熟度还不够，暂时不建议读者在生产环境中使用。

9.5 MooseFS 在 KVM 中的应用

MooseFS 在 KVM 虚拟化中最合适的应用是镜像存储，它是 OpenStack 指定的镜像存储

1107114476@qq.com
dolphinebook.com

方式之一。

9.5.1 MooseFS 介绍

MooseFS 是一款支持容错的基于网络的分布式文件系统，它将数据分布到多台物理机上，但是对外提供统一接口，MFS 提供标准的类 UNIX 文件系统：

- 高可用，数据多份存储。
- 动态扩展，可以在线添加机器和磁盘。
- 文件可以在一定周期内回收，类似回收站的功能。
- 支持在线文件快照。

MooseFS 有 4 种角色。

- 管理节点（master server）：一台单一的服务器，存储文件系统元数据。
- 数据节点（chunk server）：多台服务器，存储数据。
- 元数据备份服务器（metalogger serve）：可以有多台，存储元数据日志，并且周期性地从管理节点下载元数据文件，可以在管理节点故障的时候提示为管理节点。
- 客户端：挂载使用 MFS 文件系统的服务器。挂载使用 mfsmount 命令，mfsmount 基于 FUSE 接口，所以 MFS 支持许多文件系统，如 Linux、FreeBSD、MacOS X 等。

元数据存储在管理节点的内存中，并周期性地刷盘，并定时同步到日志节点。数据节点将存储空间分片，每片称为 chunk，最大是 64MB，每个 chunk 在数据节点上就是一个文件。

MooseFS 在 2.0 版本以前，管理节点因为是一台，一直是性能和可靠性的瓶颈。2.0 分为社区版（Community Edition，CE）和专业版（Professional）。专业版支持多台管理节点同时工作和容错，但是要收费；社区版开源免费，但还是只支持一个管理节点。

MooseFS 读性能要好于写性能，可靠性受管理节点的限制，在 KVM 虚拟化中用于镜像的存储，MooseFS 也是 OpenStack 推荐的镜像存储文件系统。

9.5.2 MFS 搭建方法及在 KVM 虚拟化中的应用

1. MFS 搭建方法

演示环境还是 9.2 节介绍的环境，本书介绍社区版的安装方法，具体的搭建方法如下。

（1）角色划分

服务器角色划分如表 9-2 所示，每个存储节点挂载 90GB 的第二块磁盘，在虚拟机中看到的是 /dev/vdb。

表 9-2 MFS 搭建服务器角色划分

类 型	主 机 名	MFS 系统角色
宿主机 1	host182	
分布式文件系统存储节点虚拟机 1	host183	Master 服务器
分布式文件系统存储节点虚拟机 2	host184	Metadatelog 服务器
分布式文件系统存储节点虚拟机 3	host185	Chunk 服务器

1107114476@qq.com
dolphinebook.com

（续）

类　型	主 机 名	MFS 系统角色
分布式文件系统存储节点虚拟机 4	host186	Chunk 服务器
分布式文件系统存储节点虚拟机 5	host187	Chunk 服务器
分布式文件系统存储节点虚拟机 6	host188	Chunk 服务器
宿主机 2	host251	Client 服务器
分布式文件系统挂载虚拟机	host189	不使用

（2）安装 yum 源

安装 MooseFS 官方的 yum 源。

```
curl http://ppa.moosefs.com/stable/yum/MooseFS.repo >/etc/yum.repos.d/MooseFS.repo
curl http://ppa.moosefs.com/stable/yum/RPM-GPG-KEY-MooseFS >/etc/pki/rpm-gpg/
RPM-GPG-KEY-MooseFS
```

（3）管理节点 Master 安装配置

1）组件安装。

用 host183 作管理节点 Master 服务器，通过安装命令行工具，管理节点组件。

```
yum install moosefs-ce-cli moosefs-ce-master moosefs-ce-cgi
```

2）修改配置文件。

修改 /etc/mfs/mfsexports.cfg。

```
[root@host183 mfs]# vim mfsexports.cfg
# 添加以下内容，允许访问的客户端 IP
10.0.0.0/8              /         re,alldir,maproot=0
"mfsexports.cfg" 67L, 4101C written
```

配置 Master 程序开机启动。

```
chkconfig mfsmaster on
```

3）创建元数据文件。

```
[root@host183 mfs]# cd /var/lib/mfs/
[root@host183 mfs]# ls -l
total 4
-rwxr--r-- 1 mfs mfs 8 Dec 24 07:25 metadata.mfs.empty
[root@host183 mfs]# cp metadata.mfs.empty metadata.mfs
```

4）启动服务。

```
[root@host183 mfs]# systemctl start mfsmaster.service
```

（4）cgi 服务器安装

cgi 是 MooseFS 的一个 Web 界面监控程序，可以看到集群状态和性能。cgi 可以单独使

1107114476@qq.com
dolphinebook.com

用服务器，也可以和其他节点复用，笔者复用管理节点 host183。

1）安装 cgi 组件，启动服务。

通过 yum 命令直接安装，然后启动服务，并配置服务开机自启动。

```
yum install moosefs-ce-cgiserv
/etc/init.d/mfscgiserv start
chkconfig mfscgiserv on
```

2）访问 Web。

通过浏览器访问 IP 地址加 9425 端口，然后输入管理节点的 IP 地址，就可以看到 Web 界面，如图 9-9 所示，可以看到集群非常详细的情况。

```
http://10.0.0.183:9425/mfs.cgi?masterhost=10.0.0.183
```

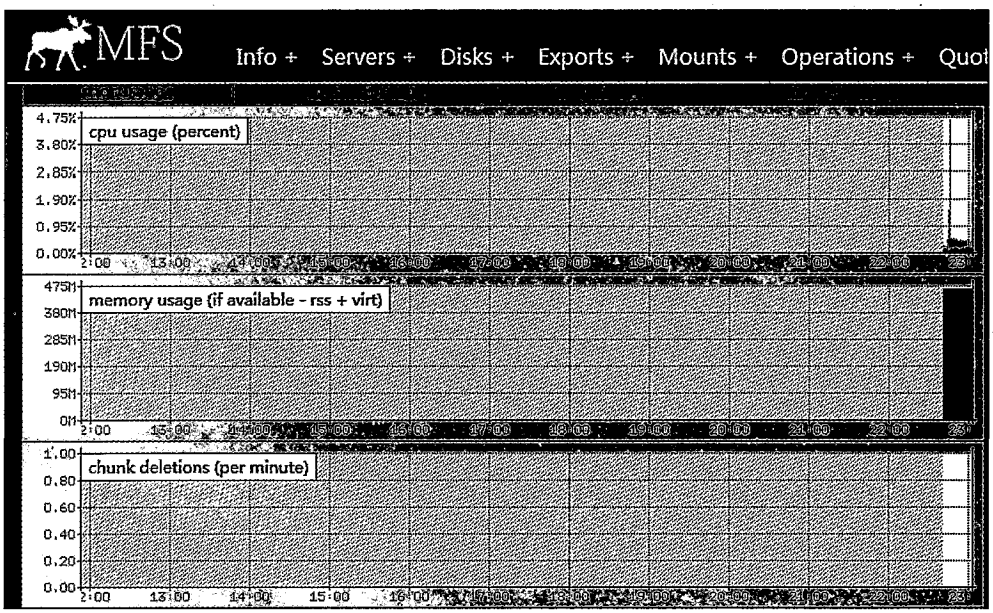


图 9-9 MFS 的 Web 监控界面

（5）Metalog 服务器安装

使用 host184 作为数据日志服务器。

1）服务安装。

通过 yum 命令安装数据日志服务组件。

```
yum install moosefs-ce-metalogger
```

2）配置并启动服务。

编辑 /etc/mfs/mfsmetalogger.cfg 文件，添加管理节点地址。

```
vim mfsmetalogger.cfg
```

1107114476@qq.com
dolphinebook.com

```
# 添加以下内容
MASTER_HOST = host183
```

启动服务，并注册为开机启动。

```
[root@host184 mfs]# /etc/init.d/mfsmetalogger start
Starting mfsmetalogger (via systemctl):  [  OK  ]
[root@host184 mfs]# chkconfig mfsmetalogger on
```

（6）存储节点（chunk server）服务器安装

使用 host185、host186、host 187、host188 作为存储节点（chunk server）。

1）安装存储组件。

通过 yum 命令安装就可以。

```
yum install moosefs-ce-chunkserver
```

2）编辑配置文件。

编辑 /etc/mfs/mfschunkserver.cfg，加入管理节点的地址。

```
vim mfschunkserver.cfg
# 添加下面的内容
MASTER_HOST = host183
```

3）配置存储。

格式化 /dev/vdb 为 xfs 文件系统，创建 /mfs 目录，并挂载 /dev/vdb 到 /mfs。

```
mkfs.xfs -f -i size=512 /dev/vdb
mkdir -p /mfs
mount /dev/vdb /mfs
```

4）编辑介质配置文件。

编辑存储介质配置文件 /etc/mfs/mfshdd.cfg，添加存储目录。

```
vim mfshdd.cfg
# 添加刚才挂载的 /mfs
/mfs
```

5）修改 /mfs 的用户属性，启动存储服务。

```
chown mfs:mfs /mfs/
/etc/init.d/mfschunkserver start
```

（7）命令行管理

通过 MFS 的命令行管理工具可以管理查看集群状态。比如查看整个集群状态，使用如下命令：

```
mfscli  -S -SIN -H host183
```

可以看到非常详细的集群状态，图 9-10 是其中的一部分信息。

Master Info	
master version	2.0.45 CE
RAM used	428 MiB
CPU used	0.54%
CPU used (system)	0.21%
CPU used (user)	0.33%
total space	308 GiB
avail space	295 GiB
trash space	0 B
trash files	0
sustained space	0 B
sustained files	0
all fs objects	28
directories	1
files	27
chunks	175
all chunk copies	243
regular chunk copies	243
last successful store	-
last save duration	-
last save status	-

图 9-10　MFS 集群状态部分信息

1107114476@qq.com
dolphinebook.com

如图 9-11 所示，通过 mfscli -S -SCS 命令，可以查看数据存储节点信息。

```
[root@host183 ~]# mfscli  -S -SCS -H host183
                                                    Chunk servers
+-----------+------+----+-----------+------+-------------+-------------------------------------+-------------------------------------+
|           |      |    |           |      |             |          'regular' hdd space        |     'marked for removal' hdd space  |
| ip/host   | port | id | version   | load | maintenance | chunks | used   | total  | % used | chunks | used | total | % used |
+-----------+------+----+-----------+------+-------------+--------+--------+--------+--------+--------+------+-------+--------+
| host185   | 9422 |  1 | 2.0.45 CE |    0 |         off |     76 | 3.7 GiB| 92 GiB |  4.03% |      0 | 0 B  | 0 B   |      - |
| host186   | 9422 |  2 | 2.0.45 CE |    0 |         off |     75 | 3.7 GiB| 92 GiB |  4.02% |      0 | 0 B  | 0 B   |      - |
| host187   | 9422 |  3 | 2.0.45 CE |    0 |         off |     51 | 2.5 GiB| 62 GiB |  4.02% |      0 | 0 B  | 0 B   |      - |
| host188   | 9422 |  4 | 2.0.45 CE |    0 |         off |     48 | 2.5 GiB| 62 GiB |  4.07% |      0 | 0 B  | 0 B   |      - |
+-----------+------+----+-----------+------+-------------+--------+--------+--------+--------+--------+------+-------+--------+
```

图 9-11 存储节点信息

命令行工具还可以移除、维护管理数据节点，具体的操作请读者阅读命令帮助信息。

（8）客户端使用

1）安装客户端组件。

在要使用 MooseFS 的客户端安装客户端组件。

```
yum install moosefs-ce-client
```

2）通过 mfsmount 挂载。

通过 mfsmount 挂载 MooseFS 到 /mnt，通过 -H 参数指定管理节点地址。

```
mfsmount -H host183 /mnt
```

通过 df -h 命令查看挂载情况。

```
[root@host251 ~]# df -h
Filesystem                    Size  Used Avail Use% Mounted on
...
host183:9421                  307G     0  307G   0% /mnt
```

可以看到管理节点作为一个网络存储文件系统，已经挂载到 /mnt 下面。

3）配置文件存储份数。

通过 mfssetgoal 命令可设置文件存储的份数。MooseFS 非常灵活，不同的目录可以设置不同的级别。设置 /mnt/mfs 目录下的文件写 3 份：

```
[root@localhost ~]# mfssetgoal -r 3 /mnt/mfs
```

通过 mfsgetgoal 命令可以得到目录的文件存储份数。

```
 [root@localhost ~]# mfsgetgoal /mnt/mfs
/mnt/mfs: 3
```

MooseFS 默认文件写 3 份，大部分分布式文件系统默认的文件都是写 3 份。

（9）MooseFS 容灾

MooseFS 社区版只有一个管理节点，当管理节点故障的时候，整个服务就会停止。存储

1107114476@qq.com
dolphinebook.com

日志服务器实际是管理节点的数据备份，通过存储日志服务可以恢复管理节点的数据，操作步骤如下。

1）重新部署管理节点。

重新安装一台服务器作为管理节点，可以和故障服务器同主机名，也可以是新的主机名和 IP。

2）传输备份数据到新的管理节点。

将存储日志服务器的备份数据传输到新的管理节点，通过 scp 命令，将 /var/lib/mfs 目录下的数据从存储日志服务器传到新的管理节点相同目录下。在存储日志服务器上执行如下命令：

```
scp /var/lib/mfs/* host183: /var/lib/mfs/
```

3）恢复数据。

在新的管理节点上执行如下命令，可以自动恢复数据。

```
mfsmaster -a
```

启动管理节点服务，如果主机名、IP 有变化，修改一下每个存储节点的管理节点配置信息，重启存储服务，集群就恢复了。

2. MFS 在 KVM 虚拟化中的使用

MFS 因为读性能好，管理节点单点，非常适合作虚拟化的镜像存储文件系统，只需要在每台宿主机上安装客户端，挂载就可以使用。

但是 MFS 不适合用于虚拟化的后端存储，主要是因为管理节点是单点，出现问题的时候，会影响整个虚拟化系统的可用性。

实际使用的时候，管理节点服务器尽量选用可靠性高、内存大的服务器。存储节点支持容错，可以使用硬盘大、可靠性相对比较差的淘汰服务器，以达到利旧的目的。

9.6　本章小结

本章介绍了 DRBD、GlusterFS、Sheepdog、MooseFS 在 KVM 虚拟化中的使用方法。DRBD、GlusterFS、MooseFS 几种分布式文件系统，稳定性都非常好，在生产环境中使用，最大的挑战是性能，解决性能最好的办法就是 SSD 和万兆网络。

当前最热的分布式文件系统是 Ceph，Ceph 架构更健壮，社区非常活跃，已经成熟，下一章将为读者详细介绍 Ceph 及其在 KVM 虚拟化中的使用方法。

1107114476@qq.com
dolphinebook.com

Chapter 10 第 10 章

Ceph 在 KVM 虚拟化中的应用与故障处理

在第 9 章介绍了 DRBD、GlusterFS、Sheepdog、MooseFS 等几种分布式文件系统之后，本章将介绍另外一种分布式文件系统 Ceph，分别从 Ceph 的架构、配置、功能、性能，以及 Ceph 在 KVM 中的应用、Ceph 的一些常见故障处理等方面介绍 Ceph。

本书之所以会用一章的内容来单独介绍 Ceph，主要原因有两点。

第一，出于对热门技术的热爱。

从架构上讲，Ceph 的架构设计均衡，没有明显的缺点，想必读者已经发现近年来各大 IT 报道中无数次将 Ceph、统一存储、云计算联系一起，这点很值得思考和重视。

第二，Ceph 提供了对象存储、块存储及文件存储的统一存储模型。

Ceph 独自提供了 3 类存储方式，而且性能和稳定性目前已经比较成熟，已经可以考虑在生产环境中使用，值得去探讨和研究。

10.1 Ceph 简介

Ceph 在设计之初便被定义为一个存储平台软件，在一个分布式集群之上，同时为用户提供对象存储、块存储及文件存储。Ceph 的主要特点包括：

- 统一存储。
- 无任何单点故障。
- 数据多份冗余。
- 存储容量可扩展。
- 自动容错及故障自愈。

1107114476@qq.com
dolphinebook.com

1. Ceph 的三大角色组件及其作用

在 Ceph 存储集群中，包含了三大角色组件，它们在 Ceph 存储集群中表现为 3 个守护进程，分别是 Ceph OSD、Monitor、MDS。它们支撑起了一个完成的 Ceph 存储集群。当然，还有一些其他的功能组件，比如 RADOS Gateway。但是最主要的还是上面的 3 个。下面详细介绍一下这 3 个角色组件的作用。

Ceph OSD：Ceph 的 OSD（Object Storage Device）守护进程。主要功能包括：存储数据、副本数据处理、数据恢复、数据回补、平衡数据分布，并将数据相关的一些监控信息提供给 Ceph Moniter，以便 Ceph Moniter 来检查其他 OSD 的心跳状态。一个 Ceph 存储集群，要求至少两个 Ceph OSD，才能有效地保存两份数据。注意，这里的两个 Ceph OSD 是指运行在两台物理服务器上的，并不是在一台物理服务器上运行两个 Ceph OSD 的守护进程。

Monitor：Ceph 的 Monitor 守护进程，主要功能是维护集群状态的表组，这个表组中包含了多张表，其中有 Moniter map、OSD map、PG（Placement Group）map、CRUSH map。

MDS：Ceph 的 MDS（Metadata Server）守护进程，主要保存的是 Ceph 文件系统的元数据。注意，对于 Ceph 的块设备和 Ceph 对象存储都不需要 Ceph MDS 守护进程。Ceph MDS 为基于 POSIX 文件系统的用户提供了一些基础命令的执行，比如 ls、find 等，这样可以很大程度上降低 Ceph 存储集群的压力。

2. Ceph 的架构及应用场景

Ceph 的架构主要分成底层数据分布及上层应用接口，如图 10-1 所示。

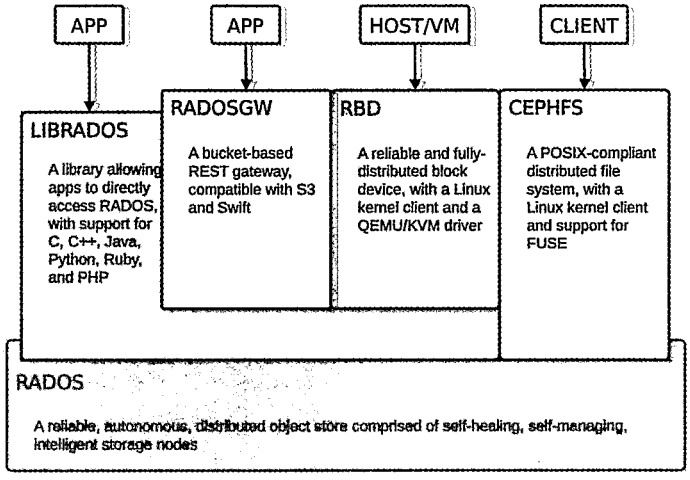


图 10-1　Ceph 架构（图片引用自 Ceph 官方文档）

- A library allowing……and PHP：LIBRADOS 库允许应用程序可以直接访问 Ceph 底层的对象存储，它支持的有 C、C++、Java、Python、Ruby 及 PHP 语言。
- A buket-based……and swift：一套基于 RESTful 协议的网关，并兼容 S3 和 Swift。

1107114476@qq.com
dolphinebook.com

- A reliable……QEMU/KVM driver：通过 Linux 内核客户端及 QEMU/KVM 驱动，来提供一个可靠且完全分布的块设备。
- A POSIX-compliant……for FUSE：通过 Linux 内核客户端结合 FUSE，来提供一个兼容 POSIX 文件结构系统。
- A reliable，autonomous……storage nodes：以其具备的自愈功能、自管理、智能存储节点特性，来提供一个可靠、自动、分布式的对象存储。

Ceph 的底层核心是 RADOS（Reliable，Autonomous，Distributed，Object，Storage），从字义上可以看出，Ceph 的本质是一个对象存储。RADOS 由两个组件组成：OSD 和 Monitor。OSD 主要提供存储资源，每一个 disk、SSD、RAID group 或者一个分区都可以成为一个 OSD，而每个 OSD 还将负责向该对象的复杂节点分发和恢复；Monitor 维护 Ceph 集群并监控 Ceph 集群的全局状态，提供一致性的决策。如图 10-2 所示展示了 RADOS 的结构组成。

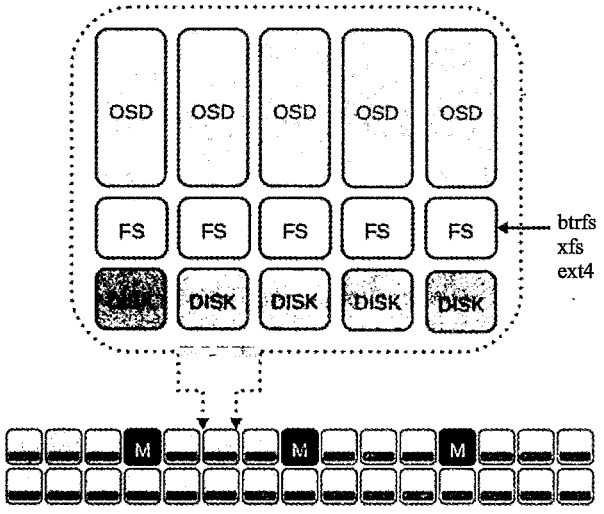


图 10-2 RADOS 结构（图片引用自 Ceph 官方文档）

RADOS 分发策略依赖于一个名为 CRUSH（Controlled Replication Under Scalable Hashing，基于可扩展哈希算法的可控复制）的算法（关于这个算法，10.3 节将进行详细的介绍和分析）。

Ceph 的应用场景主要由它的架构确定，Ceph 提供对象存储、块存储和文件存储，对应于实际应用，也以这三类存储进行区分，主要有 4 种应用，本章节着重介绍与虚拟化相关的第三和第四类应用。

第一类，Librados 应用。通俗地说，Librados 提供了应用程序对 RADOS 的直接访问，目前 Librados 已经提供了对 C、C++、Java、Python、Ruby 和 PHP 的支持。它支持单个单项的原子操作，如同时更新数据和属性、CAS 操作，同时有对象粒度的快照操作。它的实现是基于 RADOS 的插件 API，因此，其实际上就是在 RADOS 上运行的封装库。

1107114476@qq.com
dolphinebook.com

第二类，RADOSGW（Reliable Autonomic Distributed Object Storage）应用。这类应用基于 Librados 之上，增加了 HTTP 协议，提供 RESTful 接口并且兼容 S3、Swfit 接口。RADOSGW 将 Ceph 集群作为分布式对象存储，对外提供服务，如 Amazon 的 S3 范围、Swift 等。企业也可以直接将其作为媒体数据存储、分发等。

第三类，RBD（RADOS Block Device）应用。同样，这类应用也是基于 Librados 之上的，细分为两种应用场景。

1）为虚拟机提供块设备。通过 Librbd 可以创建一个块设备（Container），然后通过 QEMU/KVM 附加到 VM 上。通过 Container 和 VM 的解耦，使得块设备可以被绑定到不同的 VM 上。

2）为主机提供块设备。这种场景便是我们传统意义上理解的块存储。

以上两种方式都是将一个虚拟的块设备分片存储在 RADOS 中，都会利用数据条带化提高数据并行传输，都支持块设备的快照、COW（Copy-On-Write）克隆。最重要的是 RBD 还支持 Live migration。目前的 OpenStack、CloudStack、OpenNebula 这些管理平台都采用第一种方式为虚拟机提供块设备。

第四类，CephFS（Ceph 文件系统）应用。这类应用是基于 RADOS 实现的 PB 级分布式文件系统，其中引入了 MDS（Meta Date Server），它主要为兼容 POSIX 文件系统提供元数据，比如文件目录和文件元数据。同时 MDS 会将元数据存储在 RADOS 中，这样元数据本身也达到了并行化，可以大大加快文件操作的速度。主要指出的一点是，MDS 本身不为 Client 提供数据文件，只为 Client 提供对元数据的操作。当 Client 打开一个文件时，会查询并更新 MDS 相应的元数据（如文件包括的对象信息），然后再根据提供的对象信息直接从 RADOS 中得到文件数据。

10.2　Ceph 部署方法及在 KVM 中的应用

在 Ceph 的官网上提供了两种安装 Ceph 的方法：Installation（Quick）和 Installation（Manual）。Quick 安装是使用 ceph-deploy 工具来部署；Manual 则是一步步手动来部署 Ceph 集群，步骤相对烦琐，但有助于加深对 Ceph 各角色的理解。

此处主要介绍使用 ceph-deploy 工具来部署和管理 Ceph 集群。

10.2.1　使用 ceph-deploy 工具部署 Ceph

ceph-deploy 工具使用 SSH 的方式连接到服务器上，通过执行一系列的 Python 脚本，来完成 Ceph 集群的部署。它的一个显著特点是不需要特别配置的服务器，不需要数据库支持。ceph-deploy 是 Ceph 官方推荐的一种快速部署和管理 Ceph 集群的工具，它在 GitHub 上活跃度非常高，基本上每天都有更新。ceph-deploy 工具都是使用 Python 语言编写的，感兴趣的读者不妨读一下它的源代码，这对更好地理解 Ceph 的部署过程非常有用。

1107114476@qq.com
dolphinebook.com

Ceph 集群节点机系统采用 CentOS 6.5 64 位、Ceph 版本 0.87。总共 4 台 Ceph 节点机，每台节点机启动两个 OSD 角色，每个 OSD 角色对应一块物理磁盘。

1. 安装 ceph-deploy

1）配置主机名，配置 hosts 文件，本例 ceph-deploy 安装在其中一个节点上。

```
[root@ ceph-host-01]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
10.168.107.13   ceph-host-01
10.168.107.14   ceph-host-02
10.168.107.169  ceph-host-03
10.168.107.170  ceph-host-04
```

2）使用 ssh-keygen 生成 key，并用 ssh-copy-id 复制 key 到各节点机。例如将 key 复制到 ceph-node-173，其他节点操作相同。

```
[root@ceph-host-01 ~]# ssh-copy-id ceph-host-02
The authenticity of host 'ceph-host-02 (10.168.107.14)' can't be established.
RSA key fingerprint is 1b:54:26:c8:45:cf:12:d1:7e:02:32:a5:7f:60:95:6a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ceph-host-02, 10.168.107.14' (RSA) to the list of known
hosts.
root@ceph-host-02's password:
Now try logging into the machine, with "ssh 'ceph-host-02'", and check in:
  .ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
```

3）安装 ceph-deploy。

使用 yum 的方式安装 ceph-deploy 非常方便，在安装之前，先添加 ceph-deploy 的 yum 安装源，如下代码中将 {ceph-stable-release} 替换为目前最新的 giant 版本，{distro} 替换为 el6，最终为 baseurl=http://ceph.com/rpm-giant/el6/noarch/。配置完成之后，使用 yum install ceph-deploy 完成 ceph-deploy 的安装。

```
<!-- 配置 yum -->
[root@ceph-host-01 ~]# cat /etc/yum.repos.d/ceph.repo
[ceph-deploy]
name=Ceph Packages and Backports $basearch
baseurl= http://ceph.com/rpm-{ceph-stable-release}/{distro}/noarch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
[root@ceph-host-01 ~]# yum install ceph-deploy
[root@ceph-host-01 ~]# ceph-deploy --version
1.5.20
```

2. 创建 Ceph Monitor 角色

1）在使用 ceph-deploy 部署的过程中会生成一些配置文件，建议先创建一个目录，例如

1107114476@qq.com
dolphinebook.com

deploy_ceph_cluster，然后在这个目录中执行全部的 ceph-deploy 部署操作。

2）配置主机名，配置 hosts 文件，本例 ceph-deploy 安装在其中一个节点上。

```
[root@ceph-host-01 deploy_ceph_cluster]# ceph-deploy new ceph-host-01 ceph-host-
02 ceph-host-03
[ceph_deploy.conf][DEBUG ] found configuration file at: /root/.cephdeploy.conf
<!-- 此处省略多行 ceph-deploy DEBUG 输出 -->
[root@ceph-host-01 deploy_ceph_cluster]# ls
ceph.conf  ceph.log  ceph.mon.keyring
```

执行完成之后，配置目录下生成 3 个文件：ceph.conf 为集群配置文件；ceph.log 为 ceph-deploy 命令执行结果的日志；ceph.mon.keyring 为 ceph mon 角色的 key。

3）安装 Ceph 程序。

使用 ceph-deploy 来安装 Ceph 程序，也可以单独到每台节点机上手动安装 Ceph，根据配置的 yum 源不同，会安装不同版本的 Ceph，默认安装成功之后版本还是 v0.80，非最新版本。此处最新版本为 v0.87。

```
[root@ ceph-host-01 ceph-config]#ceph-deploy install ceph-host-01 ceph-host-02 ceph-
host-03 ceph-host-04
```

4）创建 Ceph Moniter。

```
[root@ceph-host-01 deploy_ceph_cluster]# ceph-deploy mon creat ceph-host-01 ceph-
host-02 ceph-host-03
```

5）收集节点的 key。

使用 ceph-deploy gatherkey 命令将节点的 key 文件收上来，执行成功之后，在本地目录下会新产生如下文件：ceph.client.admin.keyring（管理员 key）、ceph.bootstrap-osd.keyring（OSD 启动 key）、ceph.bootstrap-mds.keyring（MDS 启动 key）。

```
[root@ceph-host-01 deploy_ceph_cluster]# ceph-deploy gatherkeys ceph-host-01 ceph-
host-02 ceph-host-03
[root@ceph-host-01 deploy_ceph_cluster]# ls
ceph.bootstrap-mds.keyring  ceph.bootstrap-osd.keyring  ceph.client.admin.keyring
ceph.conf  ceph.log  ceph.mon.keyring
```

3. 创建 Ceph OSD 角色

1）在各节点机上，在 OSD 对应的硬盘分区，可以分 10GB 作为日志空间，剩余空间存放数据，格式化数据分区为 xfs 格式。根据 Ceph 官网的提示，生成环境中目前还是建议使用 xfs 格式。将格式化好的分区挂载到对应的目录下。使用 fdisk 命令将 sdb 分为两个区，sdb1 为 10GB，存放 Ceph Journal 不需要格式化；其余的 sdb2 存储 Ceph Data，格式化为 xfs，mkfs.xfs options: -f -i size=2048，挂载时注意 xfs mount options: -o inode64，noatime，在 fstab 文件中添加。

```
/dev/sdb2              /opt/osd-data/sas01     xfs     inode64, noatime        0 0
```

1107114476@qq.com
dolphinebook.com

使用 mount -a 命令验证 fstab 配置并挂载分区。

```
[root@ceph-host-01]# mount -a
```

2）编辑 ceph.conf，添加部分必要参数。

```
[root@ceph-host-01]# cat ceph.conf
[global]
auth_service_required = cephx
filestore_xattr_use_omap = true
auth_client_required = cephx
auth_cluster_required = cephx
mon_host = 10.168.107.13, 10.168.107.14, 10.168.107.169
mon_initial_members = ceph-host-01, ceph-host-02, ceph-host-03
fsid = 0527a494-d825-4f9d-87be-2fe6d9e47170
<!-- 以上内容都是 ceph-deploy 默认创建 -->
public_network = 10.168.107.0/24
cluster_network  = 172.16.100.0/24
<!--public 和 cluster network 是新增部分，定义了 Ceph 集群的两个网络 -->
[osd]
osd_journal_size = 10240
<!-- 定义 osd 日志的大小，10GB -->
[mon]
mon osd allow primary affinity = true
<!-- 开启 OSD 亲和度调整开关 -->
```

3）添加 OSD 分两步，第一步先初始化，第二步激活。

```
[root@ceph-host-01 deploy_ceph_cluster]# ceph-deploy  --overwrite-conf  osd prepare
ceph-host-02:/opt/osd-data/sas01:/dev/sdb1
<!--  /opt/osd-data/sas01 是 Ceph 的 Data 分区，/dev/sdb1 是 Ceph 的 Journal 分区 -->
[root@ceph-host-01 deploy_ceph_cluster]# ceph-deploy  --overwrite-conf  osd activate
ceph-host-02:/opt/osd-data/sas01:/dev/sdb1
<!-- 激活 OSD -->
```

4. 创建 Ceph MDS 角色

Ceph MDS（Metadata Server）角色不是一个必需的角色，只有当使用 CephFS 的时候才需要。CephFS 至少需要一台 Metadata 服务器，官方建议也是一台，当然，完全可以部署多台。

1）创建 Ceph MDS 角色。

命令 ceph-deploy mds create 后面跟需要创建 MDS 角色的节点名称，如果需要部署多台 MDS 的话，可以写多个节点名称。

```
[root@ceph-host-01 deploy_ceph_cluster]# ceph-deploy mds create ceph-host-01
```

使用 ceph-deploy 1.5.20 创建 MDS 之后不会自动生成 data 和 metadata 两个池，在此需要手动创建。

1107114476@qq.com
dolphinebook.com

```
[root@ceph-host-01 deploy_ceph_cluster]# ceph osd pool create data 192 192
pool 'data' created
[root@ceph-host-01 deploy_ceph_cluster]# ceph osd pool create metadata 192 192
pool 'metadata' created
[root@ceph-host-01 deploy_ceph_cluster]# ceph fs new cephfs metadata data
new fs with metadata pool 2 and data pool 1
[root@ceph-host-01 deploy_ceph_cluster]# ceph mds stat
e5: 1/1/1 up {0=ceph-host-01=up:active}
```

2）按上面的操作步骤完成之后，一个最简单的 Ceph 集群已经部署完成了，使用 ceph-s 命令查看 Ceph 集群的状态。

```
[root@ceph-host-01]# ceph -s
    cluster 0527a494-d825-4f9d-87be-2fe6d9e47170
    health HEALTH_OK
    monmap e1: 3 mons at{ceph-host-01=10.168.107.13:6789/0, ceph-host-02=10.168.
    107.14:6789/0, ceph-host-03=10.168.107.169:6789/0}, election epoch 18, quorum
    0, 1, 2 ceph-host-01, ceph-host-02, ceph-host-03
    mdsmap e5: 1/1/1 up {0=ceph-host-01=up:active}
    osdmap e24: 4 osds: 4 up, 4 in
    pgmap v44: 448 pgs, 3 pools, 1902 bytes data, 20 objects
        143 MB used, 1094 GB / 1094 GB avail
            448 active+clean
```

10.2.2　挂载 CephFS

CephFS 的挂载方式有两种：FUSE（用户空间文件系统，Filesystem in Userspace）和通过内核模块挂载，两种方式都有自己的特点。使用 FUSE 方式挂载 CephFS 不需要将内核升级到 3.x，对于虚拟化来说，目前 CentOS 或者 Red Hat 的宿主机系统大部分使用的是 6.5 或者之前的版本。直接编译升级内核，在没有充分测试之前是不可取的。那么在不改变现有系统内核的基础上，使用 FUSE 方式挂载 CephFS 是最方便的。

2015 年 3 月发布的 RHEL/CentOS 7.1 的内核已经默认支持 Ceph。

1. 使用 FUSE 方式挂载 CephFS

安装 FUSE，需要添加前面提到的 Ceph 官方源，然后直接执行 yum install ceph-fuse-y 命令即可。

复制 ceph.conf 配置文件和 ceph.client.admin.keyring 到客户端的 /etc/ceph 目录中。

```
[root@localhost ceph]# ceph-fuse /mnt/ceph-filesystem
<!-- 此处挂载过程中，ceph-fuse 会读取 ceph.conf 配置，选择可用的 Moniter 进行挂载。对于有多
个 Moniter 的集群，即使其中一个 Moniter 坏了也不影响客户端使用 -->
[root@localhost]# df -Th
Filesystem            Type            Size  Used Avail Use% Mounted on
/dev/mapper/vmvg-root ext4            46G  2.2G   41G   5% /
```

1107114476@qq.com
dolphinebook.com

```
tmpfs                   tmpfs           1.9G     0  1.9G   0% /dev/shm
/dev/vda1               ext4            194M   34M  150M  19% /boot
ceph-FUSE               FUSE.ceph-FUSE  160G   31G  130G  19% /mnt/ceph-filesystem
<!-- Filesystem 的源是 ceph-FUSE，并未绑定哪台 Moniter，文件系统类型是 FUSE.ceph-FUSE -->
```

2. 使用内核方式挂载 CephFS

对于目前使用的 CentOS 6.5 宿主机来说，需要使用内核方式挂载 CephFS，必须升级内核，CentOS 6 自带的内核都是 2.6.32-xxx 的版本。

（1）Linux 内核对 Ceph 支持现状

Ceph 官方文档目前对于 3.x 几大最主要分支的详细版本如下：

- ❑ v3.16.3 or later（rbd deadlock regression in v3.16.[0-2]）。
- ❑ NOT v3.15.*（rbd deadlock regression）。
- ❑ v3.14.*。
- ❑ v3.10.*。

为 Ceph 特别设计的文件系统 btrfs 对 Linux 内核的建议如下：

如果使用的内核不在上面推荐的 v3.x 分支的话，v3.4.20 及以上版本在 v3.4.x 中最稳定，v3.6.6 及以上版本在 v3.6.x 中最稳定。其他 v3.x 的 Linux 内核版本，官方推荐使用。

（2）升级系统内核

CentOS 6.5 升级到支持 Ceph kernel module 的版本有两个途径，下面逐一介绍。

方法一：从官网下载相应的内核版本，手动编译、安装，一般步骤如下。

```
# tar -xf linux-3.x.xx.tar.xz
# cd linux-3.x.xx
# make mrproper
<!-- 清理旧的编译生成的文件及其他配置等文件 -->
# make menuconfig
<!-- 这一步将打开配置选项菜单，选择需要编译进内核的模块 此处选择 Ceph distributed file system-->
# make all -j 8
<!-- -j 8 表示用 8 个线程去编译，一般配置为不大于服务器的 CPU 核数 -->
#make modules_install
#make install -j 8
```

编译完成后，修改 /etc/grub.conf 文件（default=0），重启系统，进入新内核。

方法二：安装编译好的 Linux 内核，例如 ELRepo（Linux 企业版的一个增强性 yum 源，有编译好的最新 Linux 内核）：http://elrepo.org/tiki/tiki-index.php。ELRepo 提供两种 Kernel 版本，一个是 kernel-lt，是长期支持版本，目前是 3.10.60；另一个是 kernel-ml，目前主线稳定版是 3.17.3。

```
#rpm -Uvh http://www.elrepo.org/elrepo-release-6-6.el6.elrepo.noarch.rpm
<!-- 安装 ELRepo 的 yum 源 -->
#yum --enablerepo=elrepo-kernel install kernel-lt
```

1107114476@qq.com
dolphinebook.com

（3）使用内核方式挂载 CephFS

内核升级完成之后，查看系统中已经加载的 Ceph 内核模块。

```
[root@localhost ~]# uname -a
Linux localhost.localdomain 3.10.60-1.el6.elrepo.x86_64 #1 SMP Sat Nov 15 10:22:
20 EST 2014 x86_64 x86_64 x86_64 GNU/Linux
[root@localhost ~]# lsmod |grep ceph
ceph 147982  1
libceph 148489  1 ceph
libcrc32c 12644  1 libceph
```

如果你的 Ceph 未使用 cephx 认证，使用如下命令挂载。

```
sudo mount -t ceph {moniter IP}:6789:/ /mnt/mycephfs
```

默认使用 ceph-deploy 部署的 Ceph 集群都是开启 cephx 认证的，可采用如下方式挂载。

```
[root@localhost ~]#mount -t ceph  {moniter IP}:6789:/ /mnt/mycephfs -o name=
admin  , secret=AQATSKdNGBnwLhAAnNDKnH65FmVKpXZJVasUeQ==
<!-- {miniter IP} 根据实际集群中 Moniter 的 IP -->
[root@localhost ~]# df -Th
Filesystem Type   Size  Used Avail Use% Mounted on
/dev/mapper/vmvg-root ext4    16G  7.0G  7.7G  48% /
tmpfs tmpfs 1004M     0 1004M    0% /dev/shm
/dev/vda1 ext4   190M     56M  120M  32% /boot
172.16.100.173:6789:/    ceph  160G  31G  130G  19% /mnt/ceph-filesystem
```

10.2.3　通过 Librbd 方式使用 Ceph RBD

Ceph RBD（Rados Block Device，基于 RADOS 对象存储的块设备）是一种对外提供块设备的存储方案。Ceph RBD 包含许多非常好用的功能，包括快照、备份、恢复，其中还有一个对于虚拟化来说最好的特性，就是对 QEMU 及 Libvirt 的支持，使虚拟机直接运行在 Ceph RBD 上成为可能。本节主要介绍如何配置 KVM 虚拟机直接运行在 Ceph RBD 之上。

1. 升级安装 QEMU 使其支持 Librbd

将 Ceph RBD 设备直接作为 KVM 虚拟机的硬盘，需要通过 Librbd 库，官方 yum 源中提供的 QEMU 用户空间程序没有添加对 Librbd 的支持，所以，要使用的话，必须手动编译 QEMU，或者使用 Ceph 官方提供的已经编译好的 rpm 包。下面介绍使用官方的 yum 源安装 QEMU 的方法。

1）安装 yum 插件。

```
[root@localhost ~]#yum install -y yum-plugin-priorities
```

2）添加 ceph-extras yum 源。

```
[root@localhost ~]# cat /etc/yum.repos.d/ceph-extras.repo
```

1107114476@qq.com
dolphinebook.com

```
[ceph-extras]
name=Ceph Extras
baseurl=http://ceph.com/packages/ceph-extras/rpm/centos6.4/$basearch
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc

[ceph-qemu-source]
name=Ceph Extras Sources
baseurl=http://ceph.com/packages/ceph-extras/rpm/centos6.4/SRPMS
enabled=1
priority=2
gpgcheck=1
type=rpm-md
gpgkey=https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc
```

3）移除系统中已经安装的 QEMU 程序。

```
[root@localhost ~]#yum remove qemu-kvm qemu-kvm-tools qemu-img
[root@localhost ~]#yum clean all
```

4）安装 Ceph 定制的 QEMU 程序。

```
[root@localhost ~]#yum install qemu-kvm qemu-kvm-tools qemu-img
[root@localhost ~]#yum install qemu-guest-agent qemu-guest-agent-win32
<!-- qemu-guest-agent 和 qemu-guest-agent-win32 是 QUMU 的虚拟机客户端代理程序 -->
```

2. 安装 Libvirt

Libvirt 是一套用于管理、操作虚拟机的常用的工具集。它支持多种虚拟化引擎，如主流的 KVM、Xen、Hyper-V 都支持。Ceph 对于 Libvirt 并没有特别的要求，使用官方提供的 yum 源，通过 yum install libvirt 命令即可安装 Libvirt 工具集。

```
[root@localhost ~]#yum install libvirt
[root@localhost ~]# rpm -qa |grep qemu
qemu-kvm-0.12.1.2-2.415.el6.3ceph.x86_64
qemu-img-0.12.1.2-2.415.el6.3ceph.x86_64
qemu-kvm-tools-0.12.1.2-2.415.el6.3ceph.x86_64
gpxe-roms-qemu-0.9.7-6.12.el6.noarch
qemu-guest-agent-0.12.1.2-2.415.el6.3ceph.x86_64
[root@localhost ~]# rpm -qa |grep libvirt
libvirt-0.10.2-46.el6_6.2.x86_64
libvirt-client-0.10.2-46.el6_6.2.x86_64
<!-- 查看系统中安装的 QEUM 及 Libvirt 包版本 -->
```

安装完 Libvirt 之后，还需要安装 Ceph，方法在上面章节已经介绍了，配置好 Ceph 官方的源，执行 yum install ceph-y 命令即可安装 Ceph。

1107114476@qq.com
dolphinebook.com

3. 配置虚拟机直接运行在 Ceph RBD 之上

1）在 Ceph Moniter 上创建用于运行虚拟机的 pool（存储池）。

```
[root@ceph-host-01 ~]#ceph osd pool create vmpool 192 192
<!-- 创建名为 vmpool 的存储池，指定 pg=192 pgs=192 -->
```

2）创建 client.libvirt 账户，并赋予 vmpool 存储的访问权限。

```
[root@ceph-host-01 ~]#ceph auth get-or-create client.libvirt mon 'allow r' osd
'allow class-read object_prefix rbd_children, allow rwx pool=vmpool'
```

3）将 Ceph 的配置文件复制到 /etc/ceph/ceph.conf。

4）将 Ceph 的授权文件复制到 /etc/ceph/ceph.client.admin.keyring。

5）将虚拟机镜像文件转换到 RBD 镜像中。

```
[root@localhost]# qemu-img convert -f qcow2 -O raw win2003ent32chs rbd:vmpool/
windows2003
<!-- 这步操作将在 vmpool 中创建名为 windows2003 的镜像，并将 Windows 2003 32 位系统转换到这
个镜像中，大小和 Windows 2003 32 位系统相同 -->
```

6）在客户端为 client.libvirt 生成一个密码，步骤 10 会用到这个步骤生成的 uuid。

```
[root@localhost ~]# cat > secret.xml <<EOF
> <secret ephemeral='no' private='no'>
> <usage type='ceph'>
> <name>client.libvirt secret</name>
> </usage>
> </secret>
> EOF
[root@localhost ~]# virsh secret-define --file secret.xml
Secret 22446aa1-1422-cbf3-3100-04147a1fa3f3 created
<!-- uuid of secret is output here -->
```

7）在 Ceph Moniter 上查看 client.libvirt 的 key。

```
[root@ceph-host-01 ~]# ceph auth get-key client.libvirt
AQBe1m1USGECDhAAgkVQW1rMf/kLYllrHvFdKg==
```

8）为 virsh 授权。

```
[root@localhost ~]# virsh secret-set-value --secret 22446aa1-1422-cbf3-3100-
04147a1fa3f3 --base64 AQBe1m1USGECDhAAgkVQW1rMf/kLYllrHvFdKg==
```

9）使用 virt-install 或者 virt-manager 创建一台虚拟机。

```
[root@localhost~]#virt-install --name=vm_name--ram=2048--vcpus=4--os-type=
windows --file=/root/vmdisk.img --file-size=10 --bridge=br0 --vnc --vncport=5920
<!-- 以上命令将创建一台名为 vm_name、内存为 2GB、4 核 CPU、操作系统为 Windows 的虚拟机 -->
```

10）编辑虚拟机的 xml 文件。

```
<disk type='network' device='disk'>
```

1107114476@qq.com
dolphinebook.com

```
 <driver name='qemu'/>
 <auth username='libvirt'>
 <!--libvirt 用户是之前创建的 Client 名字 -->
  <secret type='ceph' uuid='22446aa1-1422-cbf3-3100-04147a1fa3f3'/>
  <!-- 此处的 uuid 是步骤 6 中生成的密码 -->
</auth>
<source protocol='rbd' name='vmpool/windows2003'>
<!-- 此处指定的是虚拟机的硬盘文件路径，在 vmpool 的存储池下名为 windows2003 的镜像 -->
  <host name='10.168.107.13' port='6789'/>
  <host name='10.168.107.14' port='6789'/>
  <host name='10.168.107.169' port='6789'/>
  <!-- 上面定义的 3 个 host name 分别是 Ceph 集群的 3 个 Monitor 服务器地址 -->
 </source>
 <target dev='vda' bus='virtio'/>
 <alias name='virtio-disk1'/>
 <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
</disk>
```

11）开启虚拟机，虚拟机正常启动，已经运行在 Ceph RBD 中。

```
[root@localhost datapool]# virsh start rbd-vm-win    // rbd-vm-win 是虚拟机名称
[root@localhost datapool]# virsh qemu-monitor-command --hmp rbd-vm-win 'info block'
drive-virtio-disk0: removable=0 io-status=ok file=rbd:vmpool/windows2003:id=libv
irt:key=AQBe1m1USGECDhAAgkVQW1rMf/kLYllrHvFdKg==:auth_supported=cephx\\;none:mon_
host=10.168.107.13\\:6789\\;10.168.107.14\\:6789\\;10.168.107.169\\:6789 ro=0 drv=
raw encrypted=0
<!-- 查看虚拟机块设备信息 -->
```

10.2.4 使用内核方式挂载并使用 Ceph RBD

使用内核方式挂载 Ceph 块设备，需要一个前提，那就是需要内核支持 RBD 模块，默认 CentOS 6.5 内核不支持，需要升级内核，如何升级内核在前面章节已经介绍了。

挂载 Ceph RBD 设备之后，有两种使用方式。

第一，直接格式化当作本地磁盘使用。

第二，使用 iSCSI 软件，将这个块设备以 iSCSI 设备的形式对外提供 iSCSI 服务。

在本节中这两种使用方式都会介绍。

1. 使用内核方式映射 Ceph RBD

1）升级 CentOS 6.5 系统的内核到 3.10.x，并加装 RBD 模块。

```
[root@localhost ~]# uname -a
Linux localhost.localdomain 3.10.60-1.el6.elrepo.x86_64 #1 SMP Sat Nov 15 10:22:
20 EST 2014 x86_64 x86_64 x86_64 GNU/Linux
[root@localhost ~]# modprobe rbd
[root@localhost ~]# lsmod |grep rbd
rbd                    56747  0
libceph               148489  1 rbd
```

1107114476@qq.com
dolphinebook.com

2）执行安装 Ceph RBD 的命令。

```
[root@localhost ~]# yum install ceph-common
<!-- 配置完 Ceph 的 yum 源，因为 rbd 命令包含在 ceph-common 包中，只需安装 ceph-common-->
[root@localhost ~]# rbd -v
ceph version 0.87 (c51c8f9d80fa4e0168aa52685b8de40e42758578)
<!-- 当前安装的 rbd 是属于 ceph 0.87 的 -->
```

3）在 Ceph 集群中创建存储池及镜像。创建名为 rbdpool 的存储池，PG（Placement Group，配置组）和 PGs（配置组总和）定义为 192。

```
[root@bjgm-ceph-osd-13 ~]# ceph osd pool create rbdpool 192 192
pool 'rbdpool' created
[root@bjgm-ceph-osd-13 ~]# rbd create block-device --size 1024000 --pool rbdpool
<!-- 在存储池 rbdpool 中创建名为 block-device 的镜像，大小为 1000MB-->
[root@bjgm-ceph-osd-13 ~]# rbd ls rbdpool
block-device
<!-- 查看存储池 rbdpool 中包括的镜像 -->
[root@bjgm-ceph-osd-13 ~]# rbd info block-device --pool rbdpool
rbd image 'block-device':
    size 1000 GB in 256000 objects
    order 22 (4096 kB objects)
    block_name_prefix: rb.0.1aa6.238e1f29
    format: 1
<!-- 查看存储池 rbdpool 中镜像 block-device 的信息 -->
```

4）复制 Ceph 集群的配置及 key 到客户端。

此处为了方便实验，客户端用的 key 就是 client.admin 的 key。在生成环境中出于安全考虑，需要创建账号，重新授权。

5）客户端挂载 RBD 设备。

```
[root@localhost ~]# rbd map block-device -p rbdpool
/dev/rbd1
[root@localhost ~]# ll /dev/rbd1
brw-rw----. 1 root disk 251, 0 Nov 22 14:00 /dev/rbd1
<!-- 查看 rbd1 设备，目前已经挂载到客户端了 -->
```

2. 直接使用 Ceph RBD 设备

这种方式用得比较普遍，等于给服务器挂载了一块远程磁盘。还可以非常方便地卸载和挂载到其他服务器上，类似于云硬盘。

- ❑ 使用 fdisk /dev/rbd1 创建分区。
- ❑ 使用 mkfs.ext4 /dev/rbd1p1 格式化分区。
- ❑ mount 分区到挂载点。

```
[root@localhost ~]# mount /dev/rbd1p1 /rbd/
[root@localhost ~]# df -Th
Filesystem          Type    Size  Used Avail Use% Mounted on
```

1107114476@qq.com
dolphinebook.com

```
/dev/mapper/VolGroup-lv_root  ext4    18G  1.2G   16G   8% /
tmpfs tmpfs  2.0G  8.0K  2.0G   1% /dev/shm
/dev/vda1 ext4   477M   70M  378M  16% /boot
/dev/rbd1p1 ext4    99G   60M   94G   1% /rbd
<!-- /rbd 目录上挂载的就是 100GB 大小的 RBD 设备分区 -->
```

3. 使用 scsi-target 将 Ceph RBD 作为共享块设备对外提供服务

（1）编译安装 tgt.

编译安装 tgt 需要先用 yum 命令安装 tgt 工具，默认安装的 tgt 工具不支持 RBD，还需要从 GitHub 中下载源代码，手动编译。编译之前需打开 configure 文件将“ export CEPH_RBD”这一行修改为“ export CEPH_RBD=1”，然后编译安装。各系统环境不同，编译过程可能会报错，根据实际情况处理，编译安装过程中使用的命令为：

```
yum install scsi-target-utils # 安装 tgt 工具。
git clone https://github.com/fujita/tgt # 下载 tgt 源代码。
# 打开 configure 文件将 "export CEPH_RBD" 这一行修改为 "export CEPH_RBD=1"。
make & make install # 编译安装。
```

（2）共享 RBD 设备

```
[root@localhost ~]# tgtd
<!-- 启动 tgtd -->
[root@localhost ~]# rbd create iscsi-image --size 500000 --pool rbdpool
<!-- 创建 RBD 卷，名为 iscsi-image，大小为 500GB-->
[root@localhost ~]# tgtadm --lld iscsi --mode target --op new --tid 1 --targetname
iscsi_rbd
<!-- 创建 target，iqn 名为 iscsi_rbd -->
[root@localhost ~]# tgtadm --lld iscsi --mode logicalunit --op new --tid 1 --lun
1 --backing-store iscsi-image --bstype rbd
<!-- 创建 LUN 将 iscsi-image 加入 lun -->
[root@localhost ~]# tgtadm --lld iscsi --op bind --mode target --tid 1 -I ALL
<!-- 添加访问权限，-I 后面可以填网段，此处允许所有客户端连接 iSCSI 共享的 lun-->
```

（3）客户端挂载 iSCSI 设备

```
[root@client ~]#iscsiadm -m discovery -t st -p 10.0.0.2
[root@client ~]#iscsiadm -m node -T iSCSI_rbd -p 10.0.0.2 -l
```

10.3 CRUSH 算法介绍及调优方法

本节将介绍 CRUSH 算法的用途、特性，以及算法调优技巧。

10.3.1 CRUSH 算法介绍

CRUSH（Controlled Replication Under Scalable Hashing）是一种基于可扩展哈希的控制复制算法，在 Ceph 中，CRUSH 是其中负责数据对象实际分布与数据恢复的算法，对于大规

1107114476@qq.com
dolphinebook.com

模分布式存储系统来说，有一些问题是必须要关注，或者说必定会出现的，比如分布数据和负载（提高资源利用率），最大化系统的性能，处理系统的扩展和硬件失效，等等。而本节中介绍的 CRUSH 算法正是为了解决或者说要做得更好而出现的，要做到当发生存储设备故障、存储设备添加和移除时，能最小化地迁移数据来恢复故障其实非常必要，可以有效地减少磁盘因为频繁读 / 写而降低 Ceph 集群的性能。

1. CRUSH 算法三要素

CRUSH 算法从根本上来说，是通过存储设备的权重来计算数据对象的分布的。在计算过程中，由 Cluser Map、Data Distribution Policy 和给出的一个随机的整数 x 共同决定数据对象最终的位置。

- Cluster Map（集群映射）：它的主要作用是记录所有可用的存储资源及相互之间的空间层次架构（比如说，这个 Ceph 集群包含有多少个机架，每个机架上有多少台服务器，每个服务器上有多少个磁盘这些信息）。Cluster Map 使得 Ceph 集群存储设备在物理层面做了一层防护，这点也很容易理解，比如，数据和备份不放在同一台服务器上来避免服务器宕机受影响，数据和备份不放在同一个机柜来避免机柜掉电受影响，数据和备份不放在同一排机柜避免整排机柜跳闸受影响等。
- Data Distribution Policy（数据分布规则）：它由 Placement Rules（配置规则）组成。Placement Rules 决定了每个数据对象的副本份数，以及这些副本存储的限制条件（比如 3 个副本放在不同的服务器中，或者不同的机架中）。
- CRUSH 算法利用多参数 Hash 函数，Hash 函数中的参数含有 x，使得从 x 到 OSD 集合是确定性的和独立的。CRUSH 又是伪随机算法，所以相似输入的结果之间并没有相关性。

下面主要介绍 Cluster Map 和 Data Distribution Policy，因为这两个因素比较抽象。

2. CRUSH 算法要素说明

（1）Cluster Map 的分层结构

Cluster Map 由 Device（设备）和 Bucket（桶）组成，它们都有 ID 和权重值，并且形成一个以 Device 为叶节点、Bucket 为非叶节点的树形结构，如图 10-3 所示。

Bucket 拥有不同的类型，包括 Row、Rack、Host，如图 10-3 所示，所有机架的类型为 Rack，所有主机的类型为 Host。使用者还可以自己定义 Bucket 的类型。Device 节点的权重代表存储节点的容量和性能，其中磁盘容量是影响权重大小的重要参数。Bucket 节点的权重是其子节点的权重之和。Device 是实际存放 Placement Groups 的 OSD 设备。Cluster Map 通过 Device 和 Bucket 的定义反映了 Ceph 存储集群的物理结构。

（2）Data Distribution Policy

Data Distribution Policy 主要由 Placement Rules（放置规则）组成，它决定了数据对象的保留份数及对象副本分布的区域，每个区域可以是一个单独的故障域，这样可以保证某个区

1107114476@qq.com
dolphinebook.com

域发生故障时，在其他故障域的数据不受影响。每一个 Rule 可以定义一系列操作，而 Rule 定义的规则最终可以应用在一个池上。

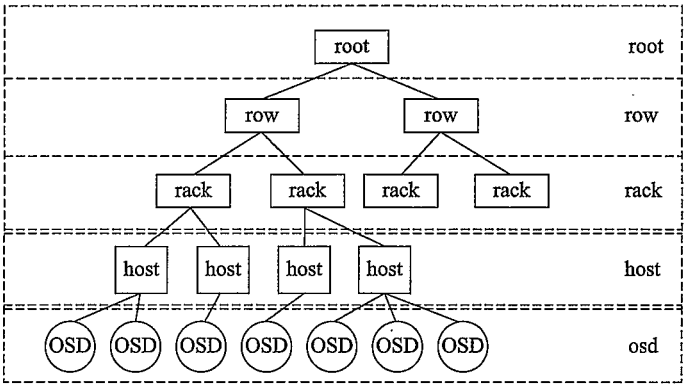


图 10-3 Cluster Map 分层结构（来自 Ceph 官方资料）

10.3.2 CRUSH 算法调优方法

CRUSH 算法不同于其他一致性算法，它提供了一些可以调优的参数，使最终的数据分布变得更加可控，通过人为干预的方式来避免一些故障的发生。

CRUSH 算法调优可以分为两类。

第一，通过调整 CRUSH MAP（描述了包含当前磁盘、服务器、机架等元素的层级结构）的配置来实现 Ceph 的架构调优。

第二，Ceph 性能调优。

1. 通过调整 CRUSH MAP 实现 Ceph 架构调优

通过调整 CRUSH MAP 实现 Ceph 架构调优的操作方法和步骤如下。

1）导出当前 Ceph 集群的 CRUSH MAP，此时的 CRUSH MAP 是一个二进制文件，还需要进一步来解析这个二进制文件。

```
[root@bjgm-ceph-mm-29 crush]# ceph osd getcrushmap -o crushmap
got crush map from osdmap epoch 5043
```

2）使用 crushtool 解析之前导出的 CRUSH MAP 二进制文件，保存为 crush.map。

```
[root@bjgm-ceph-mm-29 crush]# crushtool -d crushmap -o crush.map
```

3）打开 cursh.map 文件。

```
[root@bjgm-ceph-mm-29 crush]# vim crush.map
device 0 osd.0
device 1 osd.1
    <!--  省略相似的输出 osd.2 到 osd.3 -->
```

1107114476@qq.com
dolphinebook.com

```
device 23 osd.23
# types    <!--  Bucket types 默认包含 10 种类型，也可以自定义新的 type -->
type 0 osd
type 1 host
type 2 chassis
type 3 rack
type 4 row
type 5 pdu
type 6 pod
type 7 room
type 8 datacenter
type 9 region
type 10 root

# buckets
host bjgm-ceph-osd-13 {
        id -2                    # do not change unnecessarily
        # weight 2.120
        alg straw                //每个 Bucket 内部都支持 4 种算法：Uniform、List、Tree、
                                 //Straw，4 种算法在性能和数据重组效率方面有区别，默认情
                                 //况下 CRUSH MAP 使用 Straw
        hash 0  # rjenkins1      //Hash 算法 0 表示 rjenkins1
        item osd.4 weight 0.260  //OSD 的权重，默认情况由 OSD 的容量确定
        item osd.5 weight 0.260
        item osd.6 weight 0.260
        item osd.7 weight 0.260
        item osd.16 weight 0.540 //本例中 osd.4 是 300GB 的 SAS 盘，osd.16 是 600GB 的 SSD，
                                 //所以可以调整为两倍多一些，因为 SSD 的性能要比 SAS 高
        item osd.17 weight 0.540
}
<!-- 省略相似的 host 输出 -->
root default {                   //定义 root 类型的 bucket，命名为 default
        id -1           # do not change unnecessarily
        # weight 8.480
        alg straw                //此处也涉及 Bucket 算法的配置，默认都是 Straw
        hash 0  # rjenkins1
        item bjgm-ceph-osd-13 weight 2.120
        item bjgm-ceph-osd-14 weight 2.120
        item bjgm-ceph-osd-169 weight 2.120
        item bjgm-ceph-osd-170 weight 2.120
                                 //定义了 root 中包含 4 个 item，每个 item 表示一个 host
}

# rules
rule replicated_ruleset {        //定义 rule，命名为 replicated_ruleset，下面定义详细的规则
        ruleset 0
        type replicated          //描述规则类型是 replicated，默认就是这个
        min_size 1               //定义副本的最少个数，如果创建的 pool 不满足将不使用这个 rule
        max_size 10              //定义副本的最大个数，如果创建的 pool 不满足将不使用这个 rule
        step take default        //定义副本从哪个 bucket 往下检索 cluster map tree
```

1107114476@qq.com
dolphinebook.com

```
        step chooseleaf firstn 0 type host
                                    //选择副本存放的位置
        step emit                   //执行退出
}

# end crush map
```

其中，在选择副本存放的位置时，0 表示所有副本都放到 host 下。如果这个值 num 大于 0，小于副本份数，表示 num 份都放在 host 下；如果 num 小于 0，表示副本份数 num 放在 host 下。

4）将 cursh.map 文件编译成 CRUSH MAP 支持的二进制文件。

```
[root@ceph-mm-173 ~]# crushtool -c crush.map -o crush.new.map
```

5）应用编译好的 CRUSH MAP 策略。

```
[root@ceph-mm-173 ~]# ceph osd setcrushmap -i crush.new.map
set crush map
```

上面介绍的步骤就是通过手动编辑 CRUSH MAP 配置文件来实现的 CRUSH 算法调优，其中可以调整的部分包括 OSD 的权重、bucket 的定义、Rule 规则，等等。

2. Ceph 性能调优

Ceph 性能调优有一些通用的方法，但是大多数还是和实际的应用场景有关，以下举几个实际的应用场景来详细地介绍一下调优的理论基础和实际操作方法。

（1）场景一

在 Ceph 集群中有部分 OSD 是大容量的 SATA 盘，这些大容量的 SATA 盘的性能肯定没有 SAS 或者 SSD 盘的好，Ceph 定义了对象数据的读 / 写操作都是在主 OSD 上进行，这样就需要调整 CRUSH 算法变量来避免这部分使用 SATA 盘的 OSD 成为主 OSD，而调整 OSD 的亲和度可以使指定的 OSD 降低或者拒绝成为主 OSD。

先介绍一下 OSD 亲和度。在 Ceph 集群中，每个数据被分割为一个或多个对象，每个对象都将映射到唯一一个 PG 中，每个 PG 最终映射到 OSD，数据对象定义的保留份数决定了包含这个对象的 PG 所需要映射 OSD 的个数。简单来说，如果一个对象需要保留 3 份数据，那么这个对象数据映射的 PG 就会同时写在 3 个 OSD 中，其中一个是主 OSD，其余两个是备份 OSD。OSD 亲和度指的就是这个 OSD 成为主 OSD 的优先权。默认每个 OSD 的亲和度权重配置都是 1，意思是所有的 OSD 都有可能成为主 OSD，那么对于容量大的 OSD 来说，成为主 OSD 的概率就高。这个结果不是我们想要的。通过降低 OSD 的亲和度权重，来使配置大容量 SATA 盘的 OSD 尽可能少地被指定为主 OSD，具体实现如下。

```
[root@bjgm-ceph-mm-29 ~]# ceph pg dump |grep 'active+clean'|grep '\[4'|wc -l
dumped all in format plain
20
```

1107114476@qq.com
dolphinebook.com

```
<!-- 查看当前主 OSD 对 osd.4 的 pg 个数，结果是 20 个 -->
[root@bjgm-ceph-mm-29 ceph_config]# service ceph  restart mon
[root@bjgm-ceph-mm-29 ceph_config]# ceph osd primary-affinity osd.4 0.5
```

注意，关于上述代码有两点要说明。

1）如果在配置过程中出现如下报错：

```
[root@bjgm-ceph-mm-29 ~]# ceph osd primary-affinity osd.4 0.5
Error EPERM: you must enable 'mon osd allow primary affinity = true' on the mons
before you can adjust primary-affinity.  note that older clients will no longer
be able to communicate with the cluster.
```

如果出现如上报错，则说明 Monitor 没有打开运行调整 OSD 亲和度的开关，需要在 Ceph Monitor 角色的机器上修改 Ceph 配置文件 /etc/ceph/ceph.conf，在其中添加：

```
[mon]
mon osd allow primary affinity = true
```

添加之后重启 Moniter 进程。

2）使用 ceph osd primary-affinity osd.4 0.5 命令调整 osd.4 为 0.5 的亲和度之后，再次查看当前主 OSD 对 osd.4 的 pg 个数，结果是 8 个，说明通过 OSD 亲和度权重的调整，PG 的分布按我们预计的效果发生了变化。

```
[root@bjgm-ceph-mm-29 ceph_config]# ceph pg dump |grep 'active+clean' \
|grep '\[4'|wc -l
dumped all in format plain
8
```

（2）场景二

在 Ceph 集群中少部分 OSD 使用 SSD 硬盘（比如一台服务器只有一块 SSD 盘），这种情况下，如果只是简单地将这块 SSD 盘配置成一个 OSD，添加到 Ceph 集群中，SSD 的作用就没有完全发挥了！根据 CRUSH 算法的特点，每个 OSD 都包含两部分，一部分是 DATA，另一部分是 Journal，这两部分可以任意定义位置。在实际进行写操作时，只要数据写入 Journal 之后就会返回成功，Journal 可以理解为一个数据缓存空间，所以，将 Journal 存放在 SSD 中，性能会有显著提升，主要体现在 IOPS 性能方面。按照这个思路，可以将 SSD 划分为 10GB 大小的分区，每块非 SSD 盘在 OSD 中只用来存放 DATA，所有的 Journal 都存放在一个 SSD 的分区中。

实际操作步骤如下：

第一，手动将 SSD 盘分为 10GB 一个分区，不需要格式化。

第二，使用 ceph-deploy 添加 OSD 时执行。

```
ceph-deploy osd prepare ceph-osd-192:/opt/ceph-data1:/dev/sdg1
<!-- 使用 ceph-deploy 初始化 OSD，/opt/ceph-data 是非 SSD 盘格式化之后挂载的目录的 DATA 分
区，/dev/sdg1 是 SSD 盘上划出的一个 10GB 分区，做 Journal 用 -->
```

1107114476@qq.com
dolphinebook.com

```
ceph-deploy osd activate ceph-osd-192:/opt/ceph-data1:/dev/sdg1
 <!-- 激活 OSD -->
```

以上举了一个 OSD 的初始操作步骤，其他的 OSD 初始化也用这种方式，只是 Journal 都指到 SSD 对应的分区。如此操作完之后，进行 IOPS 压力测试，会发现 IOPS 的性能提升了不少。

10.4 多角度评估 Ceph 在 KVM 虚拟化环境中的应用

评估 Ceph 在 KVM 虚拟化环境的应用，需要从多个角度来考虑。首先是功能，对于 KVM 的某些应用场景，Ceph 是否支持；其次是性能，这点非常重要，因为所有的虚拟机最终都是针对业务的，任何因为虚拟机本身性能问题而导致业务受影响，都可能导致这个方案被否定。所以，本节将用比较大的篇幅来进行性能相关的测试，以确保实际业务上线之后的稳定和可靠。

1. KVM 对于存储相关的介绍

KVM 的虚拟机硬盘支持多种格式，最常用的格式主要包含 raw 和 qcow2。raw 格式的硬盘文件主要是一些块设备文件，比如 Linux 中的逻辑卷 lv、磁盘裸设备分区 sdb2 等。qcow2 格式文件目前应用得非常广泛，一块虚拟机的硬盘，对应于宿主机上一个 qcow2 格式的文件，不论是迁移还是备份都非常方便。从性能方面来说，raw 格式比 qcow2 格式略微好一点，但是可以在创建 qcow2 的时候，添加元数据空间预分配来提升 qcow2 的性能。预分配元数据的 qcow2 文件性能和 raw 格式的性能相同。在 Ceph 集群中提供的 CephFS 中运行 qcow2 格式的虚拟机，在 RBD 设备中直接运行 raw 格式的虚拟机，是否可行，下面主要围绕这两点进行展开。

2. 测试环境说明

下面是一些测试环境的说明。

（1）测试拓扑及服务器配置信息

首先来看一下 Ceph 集群测试环境拓扑图，如图 10-4 所示。

Ceph 集群测试环境，每台节点机都配置 3 块网卡，分别为：管理网络（管理各 Ceph 节点机）、Cluser 网络（Ceph 集群内部数据传输专用）和 Public 网络（Ceph 集群对外提供存储服务专用）。网卡全部采用千兆网卡。节点机配置配置清单如表 10-1 所示。

（2）Ceph 集群配置说明

Ceph 版本为目前最新的版本 0.87，每块硬盘为一个 OSD 角色，Data 和 Journal 都在同一块磁盘上，磁盘分区格式为 xfs。

（3）性能测试说明

测试工具使用 fio（fio 的使用在第 15 章有详细介绍）版本 1.57。所有的性能测试采用如下命令进行。

1107114476@qq.com
dolphinebook.com

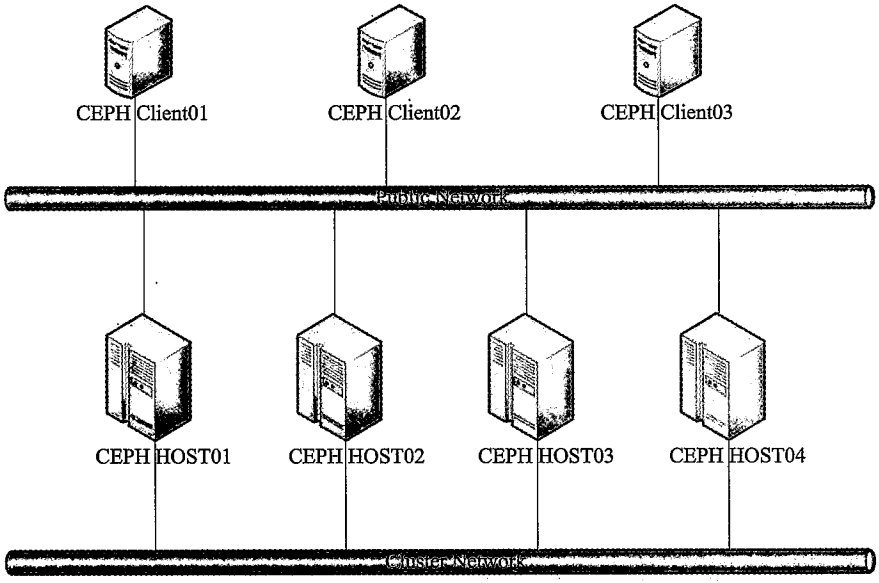


图 10-4　Ceph 集群测试环境拓扑图

表 10-1　Ceph 节点机配置清单

节点主机名及 IP 地址	硬件配置信息	Ceph 角色
Hostname：bjgm-ceph-osd-13 ClusterIP：172.16.100.13 PublicIP：10.168.107.13	CPU：E5-2620 v2 内存：64GB 硬盘：300*4 SAS	MON+MDS+OSD
Hostname：bjgm-ceph-osd-14 ClusterIP：172.16.100.14 PublicIP：10.168.107.13	CPU：E5-2620 v2 内存：64GB 硬盘：300*4 SAS	MON+MDS+OSD
Hostname：bjgm-ceph-osd-169 ClusterIP：172.16.100.169 PublicIP：10.168.107.169	CPU：E5-2620 内存：48GB 硬盘：300*4 SAS	MON+MDS+OSD
Hostname：bjgm-ceph-osd-170 ClusterIP：172.16.100.170 PublicIP：10.168.107.170	CPU：E5-2620 内存：48GB 硬盘：300*4 SAS	OSD

1）写吞吐测试，命令如下：

```
fio --filename=/mnt/ceph_filesystem/fio --direct=1 --ioengine=libaio --rw=write
--bs=4M --size=50G --runtime=300 --iodepth=16  --group_reporting --name=test-read
```

2）读吞吐测试，命令如下：

```
fio --filename=/mnt/ceph_filesystem/fio --direct=1 --ioengine=libaio --rw=read
--bs=4M --size=50G --runtime=300 --iodepth=16  --group_reporting --name=test-read
```

3）IOPS 写测试，命令如下：

1107114476@qq.com
dolphinebook.com

```
fio --filename=/mnt/ceph_filesystem/fio --direct=1 --ioengine=libaio --bs=4k
--rw=randwrite --size=50G --runtime=300 --iodepth=16  --group_reporting
--name=rand-write
```

4）IOPS 读测试，命令如下：

```
fio --filename=/mnt/ceph_filesystem/fio --direct=1 --ioengine=libaio --bs=4k
--rw=randread --size=50G --numjob=1 --runtime=300 --iodepth=16 --group_reporting
--name=rand-read
```

命令说明：

- direct=1，测试过程绕过机器自带的 buffer。
- --rw=write，表示测试任务为写。
- --bs=4M，表示测试数据块大小为 4MB。
- --size=50G，表示测试文件最大 50GB。
- --runtime=300，表示测试时间为 300s。
- --iodepth=16，表示 I/O 测试队列深度。

3. 以 FUSE 方式挂载 CephFS 并评估各项性能

（1）稳定性测试

在 CephFS 稳定性测试过程中，主要采用了长时间的反复读 / 写、删除操作，过程中还模拟各类故障，其中包括 OSD 故障、OSD 删除、OSD 添加、CRUSH 策略调整等。在持续进行近 15 天的测试过程中，未发生 CephFS 不可访问、数据丢失的情况。由此可知，除一些人为的误操作之外，CephFS 的稳定性是没有问题的。

（2）性能测试

测试内容包括 CephFS 吞吐、CephFS 磁盘 IOPS 性能。

CephFS 吞吐性能（见图 10-5），4M 顺序读最大为 51.18MB/s，4M 顺序写最大为 109.03MB/s。从结果看，顺序写的性能非常好，基本已经达到了千兆网卡的性能上限。

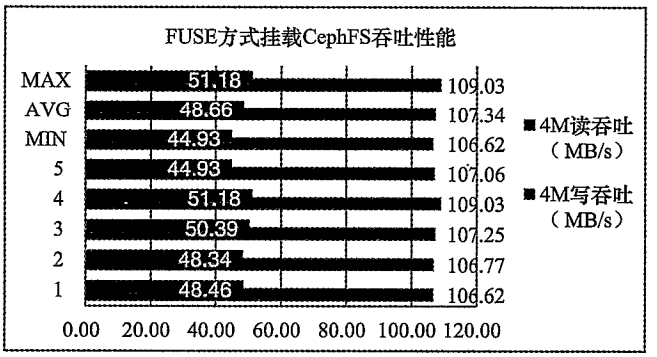


图 10-5 FUSE 方式挂载 CephFS 吞吐性能

1107114476@qq.com
dolphinebook.com

CephFS 磁盘 IOPS 性能（见图 10-6），4K 随机读最大 IOPS 为 688，4K 随机写最大 IOPS 为 9458。随机读 / 写的性能相差非常大，说明 Ceph 对于写默认是有优化的。

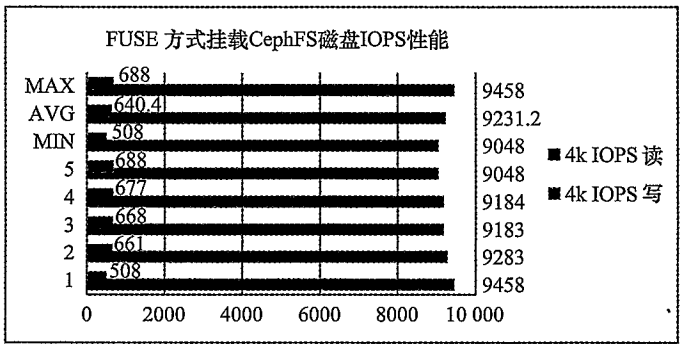


图 10-6　FUSE 方式挂载 CephFS 磁盘 IOPS 性能

4. 以内核方式挂载 CephFS 并评估各项性能

（1）稳定性测试

内核方式挂载 CephFS 做稳定性测试，测试过程和操作都与 10.3.2 节中说明的一致，从结果来看，Ceph 0.87 的稳定性和可靠性都没有问题。

在使用内核方式挂载 CephFS 时需要注意一点：在 Ceph 0.80 中，在使用 Linux 中的 cp 命令复制文件的过程中，前台会很快返回命令完成，而这实际是 Ceph 内核模块的一个异步操作，当 cp 返回成功之后，后台 kworker 工作队列进程开始往 OSD 上写数据。如果在实际数据没有写完成时，客户端直接重启，会造成操作数据不一致。当前测试环境用的 Ceph 0.87 已经修复了这个问题，实际写完成之后，才返回 cp 成功。

（2）性能测试

测试内容主要包括 CephFS 吞吐测试和 IOPS 性能测试。

CephFS 吞吐性能测试结果如图 10-7 所示，4M 顺序读最大为 33.13MB/s，4M 顺序写最大为 100.39MB/s。

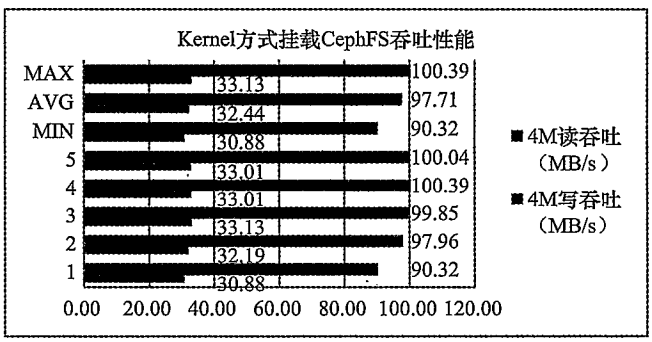


图 10-7　Kernel 方式挂载 CephFS 吞吐性能

1107114476@qq.com
dolphinebook.com

CephFS 磁盘 IOPS 性能测试结果，如图 10-8 所示，4K 随机读最大 IOPS 为 248，4K 随机写最大 IOPS 为 798。

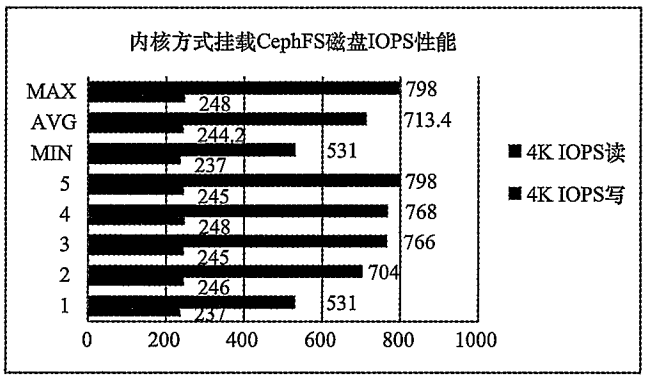


图 10-8 内核方式挂载 CephFS 磁盘 IOPS 性能

5. 虚拟机硬盘以 qcow2 格式运行在 CephFS 上的各项评估

qcow2 方式的稳定性测试方法是：虚拟机硬盘以 qcow2 格式运行在 CephFS 上，包含两个 CephFS 的挂载方式，第一是 FUSE 方式，第二是内核方式。在稳定性测试过程中创建了一台 Windows 和一台 Linux 的虚拟机，并在虚拟机里面施加磁盘读 / 写压力，长时间运行，Windows 和 Linux 两台虚拟机都没有出现死机情况。

但是在测试过程中，如果出现对 CephFS 本身进行大文件复制，以及 OSD 故障模拟的过程中，发现 Windows 虚拟机会出现蓝屏，而且概率非常高，之后进行多项调整，如 qcow2 metadata 预分配，QEMU 升级为 2.0，反复验证，Windows 虚拟机依然出现蓝屏。因此，从稳定性测试结果来看，虚拟机硬盘以 qcow2 格式运行在 CephFS 上是非常不稳定的，因为实际环境中 OSD 故障是无法避免的。

6. 虚拟机硬盘以 raw 格式运行在 RBD 上的各项评估

（1）稳定性测试

使用 Librbd 在 Ceph RBD 设备上直接运行 KVM 虚拟机，这种方案，在 OpenStack、CloudStack 等开源平台上都已经提供支持，笔者也进行了近两周不断的压力测试，以及 Ceph 集群故障模拟，都未出现 KVM 虚拟机宕机或者蓝屏的情况。所以对于在 Ceph RBD 上直接运行虚拟机，这种方案是完全可行的。

（2）性能评估

在 Ceph RBD 设备上直接运行 CentOS 6.5 虚拟机，在虚拟机内部测试磁盘 I/O 的性能，主要包括两方面。

1）吞吐测试。

性能数据如图 10-9 所示，4M 顺序读最大吞吐 2896.5MB/s，说明默认 Ceph RBD 设备

1107114476@qq.com
dolphinebook.com

对于读操作是有 cache 的，4M 顺序写最大吞吐 64.05MB/s，相较 CephFS 写吞吐，性能低 40%。

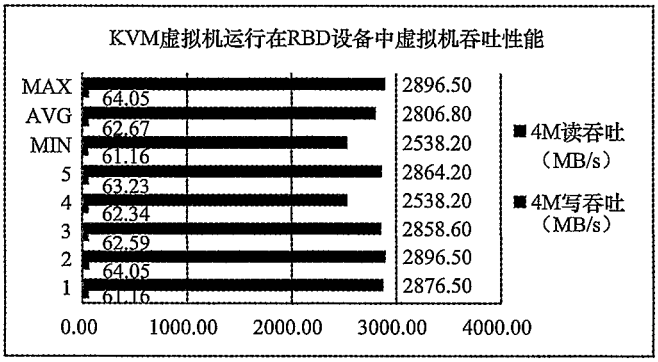


图 10-9　KVM 虚拟机运行在 RBD 设备中虚拟机吞吐性能

2）磁盘 IOPS 性能测试。

性能数据如图 10-10 所示，4K 随机读最大 IOPS 为 84873，再次印证了 Ceph RBD cache 对于读默认是开启的：4K 随机写最大 IOPS 为 2716，在实际场景中虚拟机 4K 随机写性能超过 2000，都是属于磁盘性能很高的，基本可以满足大部分游戏服务器的需求。

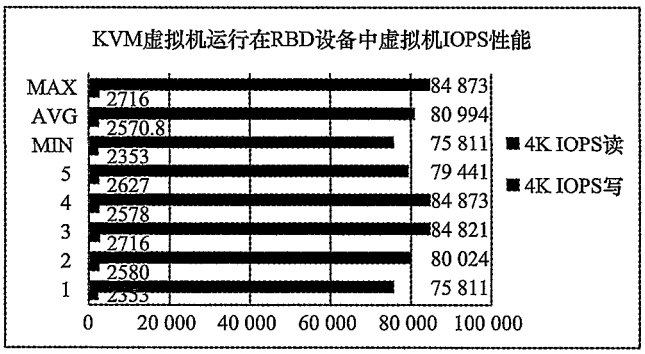


图 10-10　KVM 虚拟机运行在 RBD 设备中虚拟机 IOPS 性能

7. Ceph 在 KVM 中适用的场景

通过前面章节对于 Ceph 集群功能、性能以及稳定性的探讨，基本可以发掘目前 Ceph 在 KVM 中有 3 个适用的场景。

- 使用 CephFS，作为虚拟机的集中备份。
- 使用 Librbd 将虚拟机直接运行在 Ceph RBD 中。
- 将 Ceph RBD 作为外挂云硬盘提供给虚拟机。

后面章节将通过案例的形式，详细地介绍上面这 3 种方式在 KVM 中的应用。

1107114476@qq.com
dolphinebook.com

10.5 Ceph 生产环境搭建案例

本节主要分享 Ceph 在生成环境中针对不同应用的搭建方案，从规划、实施、优化整个过程都将进行介绍。

10.5.1 使用 CephFS 集中备份虚拟机

1. Ceph 集群前期规划

CephFS 备份集群的主要功能显然是用于备份，其实并不局限于存放虚拟机备份文件，其他的一些业务数据也同样可以使用。

对于 Ceph 节点机的选择，首要条件是大硬盘，硬盘可以选择 2TB*12 或者 4TB*12。每台节点机至少配置 3 个网口，其中两个万兆网口，一个千兆网口。两个万兆网口，一个作为 Ceph 集群内部交换数据使用，另一个作为 Ceph 集群对外提供存储设备使用；千兆网口专门作管理 Ceph 集群使用。如果条件允许的话，除管理网口之外，都使用双网口绑定，连接到不同的交换机上，交换机作堆叠。这样不仅能提高吞吐，还可以避免网口的单点故障。

Ceph 集群的网络，特别是用于 Ceph 内部数据同步的网口，连接在同一台交换机下，尽量避免过多地跨交换机流量造成性能下降。Ceph 集群默认存储池的副本是 3 份，所以至少需要 3 台 Ceph 节点才能组成一个状态健康的存储集群。

Ceph 节点的硬盘可以直接裸盘或者对于配备 RAID 卡服务器在 RAID 设置里面为每块盘都设置一个 RAID 0。

计算集群实际容量的时候，至少考虑一台 Host 宕机之后，剩余的容量足够存储所有的数据。这样即使一台 Host 宕机，也可以是让 Ceph 自动修复问题，恢复整个集群健康的状态。

2. Ceph 部署方法及注意点

部署可以使用前面章节介绍的 ceph-deploy 工具，非常方便管理和使用。可以在每台节点机上都创建 OSD 或 MON 角色，MDS 官方建议使用一个就够了，当然也可以使用两个。这样做到 Ceph 每种角色都有冗余，避免单点故障。

客户端挂载，如果目前宿主机使用的是 CentOS 6.5 或者 CentOS 6 系列，不建议升级内核，再使用内核方式挂载 CephFS，使用 Ceph-FUSE 方式挂载，对于宿主机系统的改动最小。对于宿主机内核的升级，需要反复测试之后才能进行，否则不充分的测试对于 KVM 虚拟机来说可能是致命的。

Ceph 集群搭建完成之后，需要做以下几个测试。

第一，性能测试，通过多客户端并行复制的方式，测试 CephFS 性能。

第二，故障测试，模拟 OSD 故障，单台宿主机死机，如果做网络冗余，则断开一个网口测试冗余是否有效。

1107114476@qq.com
dolphinebook.com

10.5.2　使用 Librbd 将虚拟机运行在 Ceph RBD

1. 前期规划及说明

首先需要说明一个前提，使用 Librbd 将虚拟机直接运行在 Ceph RBD 上，适用于那些对磁盘 I/O 要求一般的业务，对于超高 I/O 压力的 DB 类的应用，这类虚拟机不适合运行在 Ceph RBD 上。比较适合的应用包括 Web 站点、客户端游戏的 Game Server 服务器、大部分手游等对于 I/O 要求不是非常高的应用。

Ceph 节点机硬盘可以选择目前主流的 600GB 或者 900GB 的 SAS 盘，其中也可以混一部分 SSD 盘。

网络方面，对外提供服务的 Ceph Public 网络必须使用两个，做多网卡绑定。宿主机连接 Ceph 集群的网口同样需要使用两个网卡做绑定。这点非常重要，因为虚拟机硬盘都在 Ceph RBD 中，如果网络出问题，所有的虚拟机内部都会报 I/O 读 / 写错误。

使用 Ceph RBD 运行虚拟机，不建议在默认 CRUSH 策略下再使用 CephFS，之前测试过程中发现一旦 CephFS 有写压力时，Ceph RBD 虚拟机的性能下降非常明显，下降约 50%。所以在默认 CRUSH 算法的情况下，Ceph RBD 上运行虚拟机的集群最好是独占的。当然，可以通过配置 CRUSH 算法，将虚拟机所在的 Ceph RBD 存储池限定在指定的 OSD 中。

2. 部署方法及注意点

配置方法可以参见前面章节，有详细的说明，此处不再重复了。使用 Ceph RBD 运行虚拟机有一个非常大的优势，即可以直接使用 Ceph RBD 的一些机制来处理 KVM 虚拟机的一些日常操作，下面举一些实际操作的例子。

1）虚拟机快照操作。

```
[root@ceph-host-01]# rbd snap create vmpool/vm-centos6564@vm-centos6564.snap01
<!-- 创建虚拟机硬盘 vm-centos6564 的快照，快照在 vmpool 存储池中，命名为 vm-centos6564.snap01 -->
 [root@ceph-host-01]# rbd snap ls vmpool/vm-centos6564
SNAPID NAME                        SIZE
    2 vm-centos6564.snap01 10240 MB
<!-- 列出虚拟机硬盘 vm-centos6564 的快照列表 -->
 [root@ceph-host-01]# rbd --pool vmpool snap rollback --snap vm-centos6564.snap01
vm-centos6564
Rolling back to snapshot: 100% complete...done.
<!-- 将快照 vm-centos6564.snap01 回滚到虚拟机硬盘 vm-centos6564 -->
[root@ceph-host-01 ceph]# rbd --pool vmpool snap rm --snap vm-centos6564.snap01
vm-centos6564
<!-- 删除快照 vm-centos6564.snap01 -->
```

2）通过 RBD 镜像快照创建虚拟机。

```
[root@localhost]# qemu-img convert -f qcow2 -O raw centos6564 rbd:vmpool/centos6564
[root@ceph-host-01]# rbd info vmpool/centos6564
```

1107114476@qq.com
dolphinebook.com

```
rbd image 'centos6564':
    size 10240 MB in 2560 objects
    order 22 (4096 kB objects)
    block_name_prefix: rb.0.26ea.238e1f29
    format: 1
<!-- 默认导入的镜像为格式 1，不支持快照克隆 -->
[root@ceph-host-01]#rbd export vmpool/centos6564 -|rbd import --image-format 2 -
vmpool/centos6564.template
<!-- 转化镜像格式为 format 2 -->
[root@ceph-host-01 ~]# rbd snap create vmpool/centos6564.template@centos6564.v1
<!-- 创建模板快照 -->
[root@ceph-host-01 ~]# rbd snap protect vmpool/centos6564.template@centos6564.v1
<!-- 锁定模板快照，因为后续克隆操作都是这个快照基础上做增量，所以这个快照不能删除 -->
[root@ceph-host-01 ~]# rbd info vmpool/centos6564.template@centos6564.v1
rbd image 'centos6564.template':
    size 10240 MB in 2560 objects
    order 22 (4096 kB objects)
    block_name_prefix: rbd_data.125192ae8944a
    format: 2
    features: layering
    protected: True
<!-- 查看快照信息，format 为 2，支持 layering，已经锁定保护 -->
 [root@ceph-host-01 ~]# rbd clone vmpool/centos6564.template@centos6564.v1 vmpool/
centos6564-149.disk0
<!-- 从快照 centos6564.v1 克隆一个镜像，命名为 centos6564-149.disk0，Libvirt 可以直接将这
个镜像作为虚拟机硬盘使用，虚拟机克隆创建完成 -->
```

以上两种 RDB 相关操作，在 KVM 直接应用 Ceph RDB 设备运行虚拟机的过程中很常用，第二种克隆创建虚拟机的过程非常快。

10.5.3 在 CephFS 上创建 qcow2 文件作为云硬盘

在 CephFS 上直接创建 qcow2 格式的硬盘文件，然后直接挂载到虚拟机中作为云硬盘来使用，用于备份或者运行磁盘 I/O 不高的应用。其主要的优势在于容量可以做到不限制地扩充，在虚拟迁移的过程中也不需要复制数据。

在部署的前期规划，节点机硬盘选型可以适当使用 SSD 来提高 Ceph 集群的整体 IOPS。CPU 不要选择非常低的，比如，志强 E5504 之前的 CPU，因为在测试过程中发现，Ceph 集群 IOPS 压力大的时候，Ceph 节点机的 CPU 负载会比较高。例如 E5-2620 的 CPU，测试 IOPS 压力，当 IOPS 压力达到 4000 左右时，CPU 整体利用率接近 40%。所以为了使 CephFS 能够提供较好的性能，建议使用性能较好的 CPU。具体配置可以参见前面的章节。

10.6 Ceph 常见故障处理

任何一个系统或者应用，都无法做到 100% 的无故障，那么在出现故障的时候，如何快

1107114476@qq.com
dolphinebook.com

速地响应和恢复就变得至关重要。

对于 Ceph 集群的故障主要分为两类。

1）第一类是常规故障。例如硬盘故障，因为每个 Ceph 集群都有相当多数量的 OSD，一般而言，一个 OSD 对应一块硬盘。硬盘故障最为常见。这类故障一般的处理方式如下：

```
[root@bjgm-ceph-osd-13 ~]#  ceph osd out osd.15
<!-- 将出错的 OSD 从 Ceph 集群中踢出 -->
[root@bjgm-ceph-osd-13 ~]# ceph osd crush remove osd.15
<!-- 从 crush map 中移除错误的 OSD -->
[root@bjgm-ceph-osd-13 ~]# ceph auth del osd.15
<!-- 删除 OSD 认证信息 -->
[root@bjgm-ceph-osd-13 ~]# ceph osd rm osd.15
<!-- 删除 OSD -->
[root@bjgm-ceph-osd-13 ~]#ceph-deploy osd prepare ceph-osd-192:/opt/ceph-data:/
dev/sdc2
<!-- 使用 ceph-deploy 初始化磁盘，做此操作之前需要先格式化硬盘，并挂载到相应目录 -->
[root@bjgm-ceph-osd-13 ~]#ceph-deploy osd prepare ceph-osd-192:/opt/ceph-data:/
dev/vdc2
<!-- 使用 ceph-deploy 激活 OSD -->
```

2）第二类故障是指由 Ceph 本身原因导致的故障。比如网络故障，因为 Ceph 集群作为分布式文件系统，各种数据传输都依赖于网络，所以一旦出现网络故障，Ceph 集群基本处于僵尸的状态，数据不可读、不可写，直至网络恢复。所以避免这种故障，最好的办法就是在设计和规划的时候，必须避免网络层面的单点故障。

10.6.1　Ceph 集群监控状态检查

目前来看，并没有非常出色的监控 Ceph 集群状态的工具，这里介绍一个比较常用的 Ceph dashboard 软件 ceph-dash。

下载 Ceph dashboard 安装包：https://github.com/Crapworks/ceph-dash/archive/master.zip。

解压之后，直接运行 python ceph-dash.py。

使用浏览器访问 http://ip:5000，如果出现如下报错，是因为缺少了 autoescape 模块。

```
{
  "message": "'module' object has no attribute 'autoescape'",
  "code": 500
}
```

则解决方法如下：

```
easy_install pip
pip install --upgrade jinja2
```

最终正常之后，显示如图 10-11 所示的界面。

1107114476@qq.com
dolphinebook.com

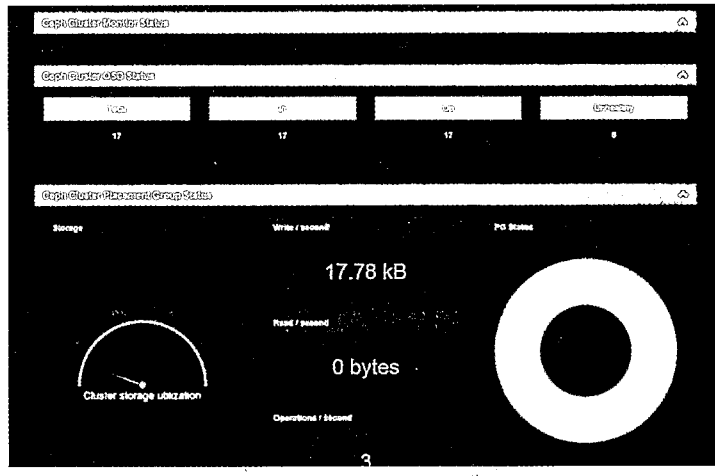


图 10-11 ceph-dash 软件展示

10.6.2 Ceph 常见错误提示的含义与处理方法

1. 节点延迟过大引起的错误

完整错误信息展示：

```
[root@bjgm-ceph-osd-13 ~]# ceph -s
    cluster 9b958522-b4b1-4622-b9fd-0e3e333cf7fe
    health HEALTH_WARN 10 requests are blocked > 32 sec
    monmap e3: 1 mons at {bjgm-ceph-osd-13=10.168.107.13:6789/0},election epoch 1,
    quorum 0 bjgm-ceph-osd-13
    mdsmap e46: 1/1/1 up {0=bjgm-ceph-osd-13=up:active}
    osdmap e6363: 24 osds: 24 up, 24 in
    pgmap v2986164: 768 pgs, 4 pools, 379 GB data, 99956 objects
        1067 GB used, 7617 GB / 8684 GB avail
            768 active+clean
```

原因分析：从提示信息上看，可以知道有 10 个请求的响应时间超过 32s，说明有 OSD 的响应速度缓慢。通过 ceph health detail 命令查看更具体的报错。

```
[root@bjgm-ceph-osd-13 ~]# ceph health detail
HEALTH_WARN 10 requests are blocked > 32 sec; 1 osds have slow requests
10 ops are blocked > 16777.2 sec
10 ops are blocked > 16777.2 sec on osd.23
```

从显示的信息可以定位是在请求 osd.23 时延迟较大。

处理方法：将响应缓慢的 osd.23 从 Ceph 集群中撤除，问题即可解决。

2. 时间不一致引起的错误

完整错误信息展示：

```
[root@ceph-node-173 ~]# ceph -s
```

1107114476@qq.com
dolphinebook.com

```
cluster 15c8236b-4238-4568-8b2f-7226b450e438
 health HEALTH_WARN clock skew detected on mon.ceph-node-174, mon.ceph-node-175
 monmap e3: 3 mons at {ceph-node-173=172.16.100.173:6789/0, ceph-node-174=
 172.16.100.174:6789/0, ceph-node-175=172.16.100.175:6789/0}, election epoch
 24, quorum 0, 1, 2 ceph-node-173, ceph-node-174, ceph-node-175
 mdsmap e6: 1/1/1 up {0=ceph-node-176=up:active}, 3 up:standby
 osdmap e37: 4 osds: 4 up, 4 in
 pgmap v1783: 192 pgs, 3 pools, 10240 MB data, 2580 objects
     30899 MB used, 129 GB / 159 GB avail
        192 active+clean
```

原因分析：这个报错是因为 Ceph mon 角色服务器之间存在时间不一致。

处理方法：在 Ceph mon 角色服务器上配置 NTP，或者在 Ceph 配置文件中 [global] 下添加 mon_clock_drift_allowed = 1，问题即可解决。

10.7　本章小结

本章从 Ceph 的架构、部署、算法调优、应用场景等方面进行了介绍，着重列举了 Ceph 在 KVM 中一些常见的应用场景，并且从功能和性能上对 Ceph 做了评估。总体来说 Ceph 是一个非常强大的分布式文件系统，而且从发展的形式来看，将来的前景是非常不错的。如果读者对分布式文件系统感兴趣，不妨把这章的内容作为深入学习 Ceph 的开始，更深入和详细的内容可以参见 Ceph 的官网 http://ceph.com/。

下一章为读者开始介绍热门的云管理平台 OpenStack。

1107114476@qq.com
dolphinebook.com

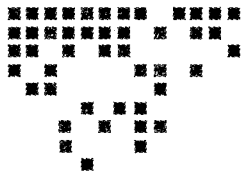
1107114476@qq.com
dolphinebook.com

第三篇 Part 3

KVM 虚拟化管理平台

- 第 11 章　利用 OpenStack 管理 KVM
- 第 12 章　利用 OpenNebula 管理 KVM
- 第 13 章　其他管理平台介绍

1107114476@qq.com
dolphinebook.com



Chapter 11 第 11 章

利用 OpenStack 管理 KVM

KVM 提供了在 Linux 系统中运行虚拟机的能力，要在生产环境中大规模使用，便捷的管理手段也是必不可少的。提到管理虚拟机的软件，目前最流行、使用最广泛的就是 OpenStack，本章将为读者介绍 OpenStack。由于本书主要内容为 KVM，而 OpenStack 又非常复杂，所以本章只介绍 OpenStack 的搭建及笔者在生产环境中的一些实践经验，如果读者想要全面地了解 OpenStack，还请阅读 OpenStack 官方相关资料。

11.1 OpenStack 搭建

本节简单介绍一下 OpenStack，然后以 OpenStack 的 Icehouse 版本为例，介绍在 Ubuntu 12.04 上搭建一套 OpenStack 环境的方法。按照功能不同，把服务器分为控制节点和计算节点，这也是实际应用中常见的 OpenStack 架构。

说明 本书中介绍的内容，基本都是基于 CentOS 系统。本章介绍的 OpenStack 是在笔者的桌面虚拟化生产环境中使用的，宿主机都使用 Ubuntu 系统，所以本章介绍的 OpenStack 搭建是基于 Ubuntu。

11.1.1 OpenStack 简介

OpenStack 是一款美国国家航空航天局和 Rackspace 合作研发的云平台管理软件，以 Apache 许可证授权，并且是自由软件和开放源代码项目。OpenStack 由 Keystone、Glance、

1107114476@qq.com
dolphinebook.com

Nova、Neutron、Cinder、Horizon 和 Swift 等组件构成：

- Keystone 负责认证、用户管理工作；
- Glance 负责镜像管理；
- Nova 负责管理计算资源；
- Neutron 负责虚拟化网络的管理；
- Cinder 提供块存储服务；
- Horizon 是 OpenStack 的 dashboard，提供了通过网页管理 OpenStack 的功能；
- Swift 提供对象存储服务。

11.1.2　控制节点的安装

在安装控制节点之前，要对服务器的环境做一些准备工作，具体如下。

服务器环境准备：

- 操作系统为 Ubuntu 12.04 LTS。
- 双网卡 eth0、eth1，以笔者的环境为例，网卡配置文件如下：

```
auto lo
iface lo inet loopback
# eth0 网卡配置
auto eth0
iface eth0 inet static
address 192.168.11.128
netmask 255.255.255.0
gateway 192.168.11.1
#eth1 不配置 IP
auto eth1
iface eth1 inet static
address 0.0.0.0
```

- 系统盘 /dev/sda 30GB，数据盘 /dev/sdb 50GB。

1. 配置 apt 的源

1）由于国外的 Ubuntu 源比较慢，笔者这里替换为国内比较常用的 163 的源，把 /etc/apt/sources.list 替换为以下内容。

```
deb http://mirrors.163.com/ubuntu/ precise main restricted
deb-src http://mirrors.163.com/ubuntu/ precie main restricted
deb http://mirrors.163.com/ubuntu/ precise-updates main restricted
deb-src http://mirrors.163.com/ubuntu/ precise-updates main restricted
deb http://mirrors.163.com/ubuntu/ precise universe
deb-src http://mirrors.163.com/ubuntu/ precise universe
deb http://mirrors.163.com/ubuntu/ precise-updates universe
deb-src http://mirrors.163.com/ubuntu/ precise-updates universe
deb http://mirrors.163.com/ubuntu/ precise multiverse
deb-src http://mirrors.163.com/ubuntu/ precise multiverse
```

1107114476@qq.com
dolphinebook.com

```
deb http://mirrors.163.com/ubuntu/ precise-updates multiverse
deb-src http://mirrors.163.com/ubuntu/ precise-updates multiverse
deb http://mirrors.163.com/ubuntu/ precise-backports main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ precise-backports main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ precise-security main restricted
deb-src http://mirrors.163.com/ubuntu/ precise-security main restricted
deb http://mirrors.163.com/ubuntu/ precise-security universe
deb-src http://mirrors.163.com/ubuntu/ precise-security universe
deb http://mirrors.163.com/ubuntu/ precise-security multiverse
deb-src http://mirrors.163.com/ubuntu/ precise-security multiverse
deb http://extras.ubuntu.com/ubuntu precise main
deb-src http://extras.ubuntu.com/ubuntu precise main
root@openstack:~# apt-get update
```

2）添加 Icehouse 的源。

```
root@openstack:~# apt-get install -y python-software-properties
root@openstack:~# add-apt-repository cloud-archive:icehouse
root@openstack:~# apt-get update
```

OpenStack 集群规模可能会达到几十到上百个节点，时间同步对于各项服务能正常工作非常重要，因此在这一步把 ntp 服务也一起安装了。

```
root@openstack:~# apt-get install ntp -y
```

2. 安装 RabbitMQ

OpenStack 各个组件之间通信靠的是支持 AMQP 的 Messaging 服务软件，RabbitMQ、Qpid、ZeroMQ 等都被 OpenStack 所支持。这里选择最常用的 RabbitMQ，本节只是介绍安装过程，RabbitMQ 的具体内容会在 11.2.1 节中展开介绍。

```
root@openstack:~# apt-get install rabbitmq-server -y
```

RabbitMQ 安装完后默认的用户为 guest，密码为 guest。为了安全，通常都要改掉这个密码，命令如下：

```
root@openstack:~#rabbitmqctlchange_password guest kvmpractice
```

3. 安装 MySQL 数据库

OpenStack 的主要组件如 Nova、Glance 等都需要数据库的支持，这里依然选择最常用的 MySQL，首先安装软件包。

```
root@openstack:~#apt-get install -y mysql-server python-mysqldb
```

安装过程中会提示设置 MySQL 的 root 密码。MySQL 默认只监听 127.0.0.1，如果要远程访问 MySQL 数据库的话，需要修改配置文件并重启服务。

```
root@openstack:~# sed-i's/127.0.0.1/0.0.0.0/g' /etc/mysql/my.cnf
```

1107114476@qq.com
dolphinebook.com

```
root@openstack:~# service mysql restart
```

4. 安装 Keystone

Keystone 是 OpenStack 的认证组件，主要起以下两个作用：

- 管理用户及其权限。
- 管理 OpenStack 的可用 service 列表，并提供这些 service 的 API endpoint。endpoint 通常是一个 URL，是访问该 service 的接口。

OpenStack 的其他组件都需要在 Keystone 中创建用户并注册 endpoint，因此要第一个安装 Keystone。

1）安装软件包。

```
root@openstack:~# apt-get install -y keystone
```

2）修改配置文件，[DEFAULT] 中有两处需要修改。

```
[DEFAULT]
admin_token=kvmpractice
connection=mysql://keystone:kvmpractice@localhost/keystone
```

Keystone 刚安装完的时候是不存在管理员用户的，admin_token 是获取 Keytone 的管理权限的唯一方法，因此这个字符串非常重要，在生产环境中要尽量设置得复杂一些。

3）创建数据库，用户名和密码与配置文件保持一致。

```
# 默认会生成一个 SQLite 数据库，为了避免冲突，先删掉
root@openstack:~# rm /var/lib/keystone/keystone.db
root@openstack:~#mysql -u root -p
mysql> CREATE DATABASE keystone;
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'192.168.11.128'IDENTIF
IED BY 'kvmpractice';
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%'IDENTIFIED BY
'kvmpractice';
mysql> exit
```

4）初始化数据库。

```
root@openstack:~# keystone-manage db_sync
```

命令执行过后，数据库就会建好 Keystone 要用的表。

```
mysql> show tables;
+------------------------+
| Tables_in_keystone     |
+------------------------+
| ec2_credential         |
| endpoint               |
| metadata               |
| migrate_version        |
```

1107114476@qq.com
dolphinebook.com

```
| role                    |
| service                 |
| tenant                  |
| token                   |
| user                    |
| user_tenant_membership  |
+-------------------------+
```

5）重启 Keystone 服务。

```
root@openstack:~#service keystone restart
```

6）验证服务是否正常。

```
root@openstack:~# service keystone status
keystone start/running, process 12996
```

然后使用 Keystone 的命令行 client 验证 Keystone 是不是正常工作，这时就要用到 admin_token 了。

```
# 设置环境变量，要访问 OpenStack 的服务就必须先通过 Keystone 的认证
root@openstack:~# vim openrc
export OS_SERVICE_TOKEN=kvmpractice export OS_SERVICE_ENDPOINT=http://192.168.11.
128:35357/v2.0
root@openstack:~# source openrc
root@openstack:~# keystone user-list
# 没有报错，说明配置成功。由于还没有创建任何用户，所以输出为空
```

7）创建一个管理员用户，用于管理 OpenStack 集群。

Keystone 中有 user、role、tenant 这几个概念，一个 user 可以在不同的 tenant 内有不同的 role（角色）。

```
root@openstack:~# source openrc
# 创建 admin 用户
root@openstack:~# keystone user-create --name=admin --pass=kvmpractice
+----------+----------------------------------+
| Property | Value |
+----------+----------------------------------+
| email | |
| enabled | True |
| id | e7a10d8df1844c6b837a2b95343cb784 |
| name | admin | | username | admin |
+----------+----------------------------------+
# 创建 admin 角色
root@openstack:~# keystone role-create --name=admin
+----------+----------------------------------+
| Property | Value |
+----------+----------------------------------+
| id | 074f9af857ba44b182241b79d2cf899c |
| name | admin |
```

1107114476@qq.com
dolphinebook.com

```
+-------------+----------------------------------+
# 创建 admin tenant
root@openstack:~# keystone tenant-create --name=admin --description="Admin
Tenant"
+-------------+----------------------------------+
| Property | Value |
+-------------+----------------------------------+
| description | Admin Tenant |
| enabled | True |
| id | 4d5675f9c8f04831ac83c8acad96d34b |
| name | admin |
+-------------+----------------------------------+
# 在 admin tenant 中赋予 admin 用户 admin 角色
root@openstack:~# keystone user-role-add --user=admin --tenant=admin --role=admin
root@openstack:~# keystone user-role-add --user=admin --role=_member_ --tenant=admin
```

8）定义 service 和 endpoint。

OpenStack 中的所有组件都要在 Keystone 中创建相应的 service 和 API endpoint，Keystone 本身要跟其他服务通信，因此也需要创建 service 和 endpoint。

```
root@openstack:~# keystone service-create --name=keystone --type=identity --de-
scription="OpenStack Identity"
+-------------+----------------------------------+
| Property | Value |
+-------------+----------------------------------+
| description | OpenStack Identity |
| enabled | True |
| id | 6d33db7219ac4dcfab41842d6e73287b |
| name | keystone |
| type | identity |
+-------------+----------------------------------+
root@openstack:~# keystone endpoint-create --service-id=$(keystone service-list
| awk '/ identity / {print $2}') --publicurl=http://192.168.11.128:5000/v2.0
--internalurl=http://192.168.11.128:5000/v2.0
--adminurl=http://192.168.11.135357/v2.0
+-------------+----------------------------------+
| Property | Value |
+-------------+----------------------------------+
| adminurl| http://192.168.11.128:35357/v2.0 |
| id | 75b8cfbb7f754882bade385a030dcfb6 |
| internalurl |http://192.168.11.128:5000/v2.0 |
| publicurl| http://192.168.11.128:5000/v2.0|
| region | regionOne|
| service_id| 6d33db7219ac4dcfab41842d6e73287b |
+-------------+----------------------------------+
```

9）验证配置是否生效。

在有了 admin 用户后，就不建议使用 admin_token 了，先把之前设置的环境变量删除，然后设置新的变量。

1107114476@qq.com
dolphinebook.com

```
root@openstack:~# vim openrc
export OS_USERNAME=admin
export OS_PASSWORD=kvmpractice
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://192.168.11.128:35357/v2.0
root@openstack:~# source openrc
root@openstack:~# keystone user-list
+----------------------------------+---------+-------+-------+
| id | enabled | email | name |
+----------------------------------+---------+-------+-------+
| e7a10d8df1844c6b837a2b95343cb784 | True | None | admin |
+----------------------------------+---------+-------+-------+
```

如果 keystone user-list 能执行成功，则说明前面创建的 admin 用户和 Keystone 的 endpoint 都生效了。

5. 安装镜像组件 Glance

Glance 是负责管理 OpenStack 镜像的服务，为用户提供存储和检索镜像的功能。用户可以通过 Glance 添加用于创建 instance 的镜像。

1）安装软件包。

```
root@openstack:~#apt-get install -y glance
```

2）编辑配置文件。

Glance 有两个主要的配置文件 /etc/glance/glance-api.conf 和 /etc/glance/glance-registry.conf，需要修改的配置项是一致的，如下：

```
[database]
#sqlite_db = /var/lib/glance/glance.sqlite    这一行注释掉
connection = mysql://glance:kvmpractice@192.168.11.128/glance

[keystone_authtoken]
auth_uri = http://192.168.11.128:5000
auth_host = 192.168.11.128
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = kvmpractice

[paste_deploy]
flavor=keystone
```

3）创建数据库，用户名和密码跟配置文件保持一致。

```
# 先删除 Ubuntu 的软件包自带的 SQLite 文件
root@openstack:~#rm /var/lib/glance/glance.sqlite
root@openstack:~#mysql -u root -p
mysql> CREATE DATABASE glance;
```

1107114476@qq.com
dolphinebook.com

```
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'192.168.11.128'
IDENTIFIED BY 'kvmpractice';
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' IDENTIFIED BY 'kvmpractice';
```

4）初始化数据库。

```
root@openstack:~#glance-manage db_sync
```

5）在 Keystone 中创建 Glance 的 user、service 和 endpoint。

① 创建名为 service 的 tenant，后面安装其他组件时也都要用到。

```
root@openstack:~# keystone tenant-create --name=service
+-------------+----------------------------------+
| Property | Value |
+-------------+----------------------------------+
| description | None |
| enabled | True |
| id | 0a76675d8f714fcd9336fb5ff5133599 |
| name | service |
+-------------+----------------------------------+
```

② 创建 glance 用户，并在 service 这个 tenant 中赋予 glance 用户 admin 角色，用户密码要与配置文件 [keystone_authtoken] 中保持一致。

```
root@openstack:~# keystone user-create --name=glance --pass=kvmpractice
+----------+----------------------------------+
| Property | Value |
+----------+----------------------------------+
| email | |
| enabled | True |
| id | d0c9969dc1f3401a87c3c7c2fd06c44d |
| name | glance |
| username | glance |
+----------+----------------------------------+
root@openstack:~# keystone user-role-add --user=glance --tenant=service --role=admin
```

③ 创建 service 和 API endpoint。

```
root@openstack:~# keystone service-create --name=glance --type=image
--description="OpenStack Image Service"
+-------------+----------------------------------+
| Property | Value |
+-------------+----------------------------------+
| description | OpenStack Image Service |
| enabled | True |
| id | 9fb4a282b9574fc6a35eb864728d3a1d |
| name | glance |
| type | image |
+-------------+----------------------------------+
root@openstack:~# keystone endpoint-create   --service-id=$(keystone service-list
| awk '/ image / {print $2}')
```

1107114476@qq.com
dolphinebook.com

```
--publicurl=http://192.168.11.128:9292
--internalurl=http://192.168.11.128:9292
--adminurl=http://192.168.11.128:9292
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
|   adminurl  |    http://192.168.11.128:9292    |
|      id     | 1377b6393f5d4f289a7a9673211ff4da |
| internalurl |    http://192.168.11.128:9292    |
|  publicurl  |    http://192.168.11.128:9292    |
|    region   |            regionOne             |
|  service_id | 9fb4a282b9574fc6a35eb864728d3a1d |
+-------------+----------------------------------+
```

6）重启服务。

```
root@openstack:~# service glance-registry restart
root@openstack:~# service glance-api restart
```

7）验证服务是否正常。

```
root@openstack:~# service glance-registry status
root@openstack:~# service glance-api status
root@openstack:~# source openrc
root@openstack:~# glance-image list
```

下面添加一个镜像。

```
root@openstack:~# mkdir /tmp/images
root@openstack:~#  cd /tmp/images/
#cirros 是一个精简的 Linux 系统，为了方便我们将以此作为测试镜像
root@openstack:~# wget http://cdn.download.cirros-cloud.net/0.3.2/cirros-0.3.2-x86_
64-disk.img
root@openstack:/tmp/images# glance image-create --name=cirros --disk-format=qcow2
--container-format bare --is-public=true < cirros-0.3.2-x86_64-disk.img
+------------------+--------------------------------------+
| Property | Value |
+------------------+--------------------------------------+
| checksum | 64d7c1cd2b6f60c92c14662941cb7913 |
| container_format | bare |
| created_at | 2014-12-30T08:19:16 |
| deleted | False |
| deleted_at | None |
| disk_format | qcow2 |
| id | f32326d6-a8af-4de6-82e3-2a978b1a7627 |
| is_public | True |
| min_disk | 0 |
| min_ram | 0 |
| name | cirros |
| owner | 4d5675f9c8f04831ac83c8acad96d34b |
| protected | False |
```

1107114476@qq.com
dolphinebook.com

```
| size | 13167616 |
| status | active |
| updated_at | 2014-12-30T08:19:16 |
| virtual_size | None |
+------------------+--------------------------------------+
root@openstack:/tmp/images# glance image-list
+--------------------+-----------+----------+--------------+----------+--------+
| ID | Name | Disk Format | Container Format | Size | Status |
+--------------------+-----------+----------+--------------+----------+--------+
| f32326d6-a8af-4de6-82e3-2a978b1a7627 | cirros | qcow2 | bare | 13167616 | active |
+--------------------+-----------+----------+--------------+----------+--------+
# 镜像加入 Glance 后，原文件就不需要保留了
root@openstack:~# rm /tmp/images/cirros-0.3.2-x86_64-disk.img
```

测试镜像已经成功地加入到了 Glance 中。如果读者想添加自己制作的镜像，可以参考本书关于制作虚拟机镜像的第 16 章内容。

6. 安装 Nova 组件

Nova 负责 OpenStack 的计算管理，由多个组件构成。控制节点上要安装除了 nova-compute 外的所有其他 Nova 组件。

1）安装软件包。

```
root@openstack:~# apt-get install nova-api nova-cert nova-conductor nova-consoleauth
nova-novncproxy nova-scheduler python-novaclient -y
```

2）修改配置文件。

修改 /etc/nova/nova.conf 文件内容如下：

```
root@openstack:~# vim /etc/nova/nova.conf:

[DEFAULT]
rpc_backend = rabbit
rabbit_host = 192.168.11.128
rabbit_password = kvmpractice
my_ip = 192.168.11.128
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = 192.168.11.128
auth_strategy = keystone

#networking
network_api_class = nova.network.neutronv2.api.API
neutron_url = http://192.168.11.128:9696
neutron_auth_strategy = keystone
neutron_admin_tenant_name = service
neutron_admin_username = neutron
neutron_admin_password = kvmpractice
neutron_admin_auth_url = http://192.168.11.128:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```

1107114476@qq.com
dolphinebook.com

```
security_group_api = neutron

#neutron metadata agent
service_neutron_metadata_proxy = true
neutron_metadata_proxy_shared_secret = kvmpractice

[database]
connection = mysql://nova:kvmpractice@192.168.11.128/nova

[keystone_authtoken]
auth_uri = http://192.168.11.128:5000
auth_host = 192.168.11.128
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = kvmpractice
```

其中 Neutron 相关的配置项要与后面要安装的 Neutron 的配置保持一致。

3）创建 Nova 数据库，nova 用户的密码与配置文件中保持一致。

```
root@openstack:~# mysql -u root -p
mysql> CREATE DATABASE nova;
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'192.168.11.128' \
IDENTIFIED BY 'kvmpractice';
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' \
IDENTIFIED BY 'kvmpractice';
```

4）初始化数据库：

```
root@openstack:~# nova-manage db sync
```

5）在 Keystone 中创建 Nova 的 user、service 和 endpoint。

① 创建 nova 用户，并且在 service 这个 tenant 中赋予 nova 用户 admin 角色。

```
root@openstack:~# keystone user-create --name=nova --pass=kvmpractice
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |                                  |
| enabled  |               True               |
|    id    | 42ab62a0c92a4e0d99a6e3a886b77e5a |
|   name   |               nova               |
| username |               nova               |
+----------+----------------------------------+
root@openstack:~# keystone user-role-add --user=nova --tenant=service --role=admin
```

② 创建 service 和 API endpoint。

```
root@openstack:~# keystone service-create --name=nova --type=compute --descrip-
tion="OpenStack Compute"
```

1107114476@qq.com
dolphinebook.com

```
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |        OpenStack Compute         |
|   enabled   |               True               |
|      id     | 3eb163185c3443249bf25b65a927cb24 |
|     name    |               nova               |
|     type    |             compute              |
+-------------+----------------------------------+
root@openstack:~# keystone endpoint-create \
--service-id=$(keystone service-list | awk '/ compute / {print $2}') \
--publicurl=http://192.168.11.128:8774/v2/%\(tenant_id\)s \
--internalurl=http://192.168.11.128:8774/v2/%\(tenant_id\)s \
--adminurl=http://192.168.11.128:8774/v2/%\(tenant_id\)s
+-------------+-------------------------------------------+
|   Property  |                   Value                   |
+-------------+-------------------------------------------+
|   adminurl  | http://192.168.11.128:8774/v2/%(tenant_id)s |
|      id     |       bf80d03ea49b409c89270633a58ee13d      |
| internalurl | http://192.168.11.128:8774/v2/%(tenant_id)s |
|  publicurl  | http://192.168.11.128:8774/v2/%(tenant_id)s |
|    region   |                 regionOne                 |
|  service_id |       3eb163185c3443249bf25b65a927cb24      |
+-------------+-------------------------------------------+
```

6）重启服务。

```
root@openstack:~# cd /etc/init.d;for i in `ls nova-*`;do service $i resetart;done
```

7）验证服务是否正常。

```
root@openstack:~# cd /etc/init.d;for i in `ls nova-*`;do service $i status;done
root@openstack:~# nova image-list
+--------------------------------------+--------+--------+--------+
| ID | Name | Status | Server |
+--------------------------------------+--------+--------+--------+
| f32326d6-a8af-4de6-82e3-2a978b1a7627 | cirros | ACTIVE | |
+--------------------------------------+--------+--------+--------+
```

nova image-list 看到的正是之前添加的镜像。

7. 安装网络组件 Neutron

Neutron 负责整个 OpenStack 的网络部分，在 Grizzly 版本及之前的版本中曾叫作 Quantum，因此在老的文档里见到 Quantum 也是指 OpenStack 的网络组件。

1）安装软件包。

```
root@openstack:~# apt-get install neutron-server neutron-plugin-ml2 neutron-
plugin-openvswitch-agent openvswitch-datapath-dkms neutron-l3-agent neutron-
dhcp-agent -y
```

1107114476@qq.com
dolphinebook.com

2）修改 Linux 内核参数。

```
root@openstack:~# vim /etc/sysctl.conf
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
#使修改生效
root@openstack:~# sysctl -p
```

3）修改配置文件。

Neutron 组件众多，因此需要修改的配置文件也很多。

① 修改 /etc/neutron/neutron.conf 文件内容如下：

```
[DEFAULT]
auth_strategy = keystone

rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_host = 192.168.11.128
rabbit_password = kvmpractice

notify_nova_on_port_status_changes = True
notify_nova_on_port_data_changes = True
nova_url = http://192.168.11.128:8774/v2
nova_admin_username = nova
nova_admin_tenant_id = 0a76675d8f714fcd9336fb5ff5133599
nova_admin_password = kvmpractice
nova_admin_auth_url = http://192.168.11.128:35357/v2.0

core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True

[database]
connection = mysql://neutron:kvmpractice@192.168.11.128/neutron

[keystone_authtoken]
auth_uri = http://192.168.11.128:5000
auth_host = 192.168.11.128
auth_protocol = http
auth_port = 35357
admin_tenant_name = service
admin_user = neutron
admin_password = kvmpractice
```

② 修改 /etc/neutron/plugins/ml2/ml2_conf.ini 文件内容如下：

```
[ml2]
type_drivers = vlan
tenant_network_types = vlan
mechanism_drivers = openvswitch
[ml2_type_vlan]
```

1107114476@qq.com
dolphinebook.com

```
network_vlan_ranges = physnet1:2:1024

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
enable_security_group = True

[ovs]
tenant_network_type = vlan
bridge_mappings = physnet1:br-eth1
```

③ 修改 /etc/neutron/l3_agent.ini 文件内容如下：

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
```

④ 修改 /etc/neutron/dhcp_agent.ini 文件内容如下：

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
use_namespaces = True
```

⑤ 修改 /etc/neutron/metadata_agent.ini 文件内容如下：

```
[DEFAULT]
auth_url = http://192.168.11.128:5000/v2.0
auth_region = regionOne
admin_tenant_name = service
admin_user = neutron
admin_password = kvmpractice
nova_metadata_ip = 192.168.11.128
metadata_proxy_shared_secret = kvmpractice
```

4）配置 Open vSwitch。

选择 Open vSwitch 作为 Neutron 的 plugin 来实现底层的网络虚拟化，Open vSwitch 需要进行配置（Open vSwitch 安装使用在第 4 章有详细介绍）。

```
root@openstack:~# service openvswitch-switch restart
root@openstack:~# ovs-vsctl add-br br-int
root@openstack:~# ovs-vsctl add-br br-eth1
root@openstack:~# ovs-vsctl add-port br-eth1 eth1
root@openstack:~# ovs-vsctl add-br br-ex
```

5）创建数据库。

```
root@openstack:~# mysql -u root -p
mysql> CREATE DATABASE neutron;
mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'192.168.11.128' IDENTIFIED
BY 'kvmpractice';
mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%' IDENTIFIED BY 'kvmpractice';
```

1107114476@qq.com
dolphinebook.com

6）在Keystone中创建neutron的user、service和endpoint。

① 创建neutron用户，并且在service这个tenant中赋予neutron用户admin角色。

```
root@openstack:~# keystone user-create --name neutron --pass kvmpractice
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |                                  |
| enabled  |               True               |
|    id    | 50a8f4934e264bc8ba5a25be94f338f3 |
|   name   |             neutron              |
| username |             neutron              |
+----------+----------------------------------+
root@openstack:~# keystone user-role-add --user neutron --tenant service --role admin
```

② 创建service和API endpoint。

```
root@openstack:~# keystone service-create --name neutron --type network
--description "OpenStack Networking"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |       OpenStack Networking       |
|   enabled   |               True               |
|      id     | 1521d75b03c346cdaf8bf621f4b8e092 |
|     name    |             neutron              |
|     type    |             network              |
+-------------+----------------------------------+
root@openstack:~# keystone endpoint-create \
--service-id $(keystone service-list | awk '/ network / {print $2}') \
--publicurl http://192.168.11.128:9696 \
--adminurl http://192.168.11.128:9696 \
--internalurl http://192.168.11.128:9696
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
|   adminurl  |    http://192.168.11.128:9696    |
|      id     | 598218212bb24fc5bd8ec146ba798215 |
| internalurl |    http://192.168.11.128:9696    |
|  publicurl  |    http://192.168.11.128:9696    |
|    region   |            regionOne             |
|  service_id | 1521d75b03c346cdaf8bf621f4b8e092 |
+-------------+----------------------------------+
```

7）重启服务。

```
root@openstack:~# cd /etc/init.d;for i in `ls nova-*`;do service $i resetart;done
```

8）验证服务是否正常。

```
root@openstack:~# cd /etc/init.d;for i in `ls nova-*`;do service $i status;done
```

1107114476@qq.com
dolphinebook.com

```
root@openstack:~# source openrc
root@openstack:~# neutron agent-list
+--------------------+--------------------+-----------+-------+----------------+
| id | agent_type | host | alive | admin_state_up |
+--------------------+--------------------+-----------+-------+----------------+
| 17e8a447-4e6e-4ae4-8b13-61112ee15fa3 | Metadata agent | openstack | :-) | True |
| 7d9b1036-dbd3-44e2-8625-c31191090d71 | Open vSwitch agent | openstack | :-) | True |
| d0ca32a9-4f07-4c7a-90a2-8bc7aabbdbd3 | DHCP agent | openstack | :-) | True |
| d7eae98f-07fe-441c-86c2-1d0b6d7f52ca | L3 agent | openstack | :-) | True |
+--------------------+--------------------+-----------+-------+----------------+
```

命令输出显示 Neutron 的服务状态都正常。

8. 安装 dashboard——Horizon

Horizon 是 OpenStack 的 dashboard 组件，使用户可以通过网页更加直观、方便地管理 OpenStack。

1）安装软件包。

```
root@openstack:~# apt-get install apache2 memcached libapache2-mod-wsgi
openstack-dashboard -y
```

2）如果不想使用 Ubuntu 的主题，可以卸载软件包。

```
root@openstack:~# apt-get remove --purge openstack-dashboard-ubuntu-theme -y
```

3）验证 Horizon 是否能正常工作。

在浏览器中输入 http://192.168.11.128/horizon，使用 admin 用户登录 dashboard，这个用户正是前面在 Keystone 中创建的管理员用户。

9. 安装块存储服务 Cinder

Cinder 为 OpenStack 提供块存储服务，由 3 个组件组成：cinder-api、cinder-scheduler、cinder-volume。其中，cinder-api 负责处理 API 请求；cinder-volume 负责实际操作后端存储；而 cinder-scheduler 则是起调度作用，决定在哪个存储节点上创建 volume。如果有专门的存储节点，可以把 cinder-volume 部署在存储节点，把 cinder-api、cinder-scheduler 部署在控制节点。由于我们的架构中没有存储节点，因此 3 个组件都部署在控制节点。

1）安装软件包。

```
root@openstack:~# apt-get install cinder-api cinder-scheduler cinder-volume -y
```

2）修改配置文件。

```
root@openstack:~# vim /etc/cinder/cinder.conf
[DEFAULT]
rpc_backend = rabbit
rabbit_host = 192.168.11.128
rabbit_port = 5672
```

1107114476@qq.com
dolphinebook.com

```
rabbit_userid = guest
rabbit_password = kvmpractice

[database]
connection = mysql://cinder:kvmpractice@192.168.11.128/cinder

[keystone_authtoken]
auth_uri = http://192.168.11.128:5000
auth_host = 192.168.11.128
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = cinder
admin_password = kvmpractice
```

3）创建数据库。

```
root@openstack:~# mysql -u root -p
mysql> CREATE DATABASE cinder;
mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' \
IDENTIFIED BY 'kvmpractice';
mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' \
IDENTIFIED BY 'kvmpractice';
```

4）初始化数据库。

```
root@openstack:~# cinder-manage db sync
```

5）在 Keystone 中创建 cinder 的 user、service 和 endpoint。

① 创建 cinder 用户，并且在 service 这个 tenant 中赋予 cinder 用户 admin 角色。

```
root@openstack:~# keystone user-create --name=cinder --pass=kvmpractice
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |                                  |
| enabled  |               True               |
|    id    | 78c5da5c5c164fc58250ba11a0e477de |
|   name   |              cinder              |
| username |              cinder              |
+----------+----------------------------------+
root@openstack:~# keystone user-role-add --user=cinder --tenant=service --role=
admin
```

② 创建 service 和 endpoint。

```
root@openstack:~# keystone service-create --name=cinder --type=volume --descrip-
tion="OpenStack Block Storage"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
```

1107114476@qq.com
dolphinebook.com

```
| description |     OpenStack Block Storage      |
|   enabled   |               True               |
|      id     | 0b087e898f1445e5b3b5e69280f9aa8a |
|     name    |              cinder              |
|     type    |              volume              |
+-------------+----------------------------------+
root@openstack:~# keystone endpoint-create \
--service-id=$(keystone service-list | awk '/ volume / {print $2}') \
--publicurl=http://192.168.11.128:8776/v1/%\(tenant_id\)s \
--internalurl=http://192.168.11.128:8776/v1/%\(tenant_id\)s \
--adminurl=http://192.168.11.128:8776/v1/%\(tenant_id\)s
+-------------+-----------------------------------------------+
|   Property  |                     Value                     |
+-------------+-----------------------------------------------+
|   adminurl  | http://192.168.11.128:8776/v1/%(tenant_id)s   |
|      id     |       5cec6f26a5a840d0a43800ad5f99f6d0        |
| internalurl | http://192.168.11.128:8776/v1/%(tenant_id)s   |
|  publicurl  | http://192.168.11.128:8776/v1/%(tenant_id)s   |
|    region   |                  regionOne                    |
|  service_id |       0b087e898f1445e5b3b5e69280f9aa8a        |
+-------------+-----------------------------------------------+
root@openstack:~# keystone service-create --name=cinderv2
--type=volumev2 --description="OpenStack Block Storage v2"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |    OpenStack Block Storage v2    |
|   enabled   |               True               |
|      id     | b40886709e174ff481a62ae6217fb623 |
|     name    |             cinderv2             |
|     type    |             volumev2             |
+-------------+----------------------------------+
root@openstack:~# keystone endpoint-create \
--service-id=$(keystone service-list | awk '/ volumev2 / {print $2}') \
--publicurl=http://192.168.11.128:8776/v2/%\(tenant_id\)s \
--internalurl=http://192.168.11.128:8776/v2/%\(tenant_id\)s \
--adminurl=http://192.168.11.128:8776/v2/%\(tenant_id\)s
+-------------+-----------------------------------------------+
|   Property  |                     Value                     |
+-------------+-----------------------------------------------+
|   adminurl  | http://192.168.11.128:8776/v2/%(tenant_id)s   |
|      id     |       24051382da9045a4b957af65585ac67a        |
| internalurl | http://192.168.11.128:8776/v2/%(tenant_id)s   |
|  publicurl  | http://192.168.11.128:8776/v2/%(tenant_id)s   |
|    region   |                  regionOne                    |
|  service_id |       b40886709e174ff481a62ae6217fb623        |
+-------------+-----------------------------------------------+
```

6）创建 Cinder 用的 Volume Group。

```
root@openstack:~# pvcreate /dev/sdb
```

1107114476@qq.com
dolphinebook.com

```
# 其中 sdb 是给 cinder 用的数据盘
root@openstack:~# vgcreate cinder-volumes /dev/sdb
```

在实际环境中，很多情况下可能用户根本不会用到 Cinder，启动 Cinder 服务只是为了不让 dashboard 报错，这时候可以用以下解决方案。

```
root@openstack:~# cd /mnt
root@openstack:~# dd if=/dev/zero of=cinder-volumes bs=1 count=0 seek=2G
root@openstack:~# losetup /dev/loop2 /mnt/cinder-volumes
root@openstack:~# vim /etc/init.d/cinder-setup-backing-file
losetup /dev/loop2 /mnt/cinder-volumes
exit 0
root@openstack:~# chmod 755 /etc/init.d/cinder-setup-backing-file
root@openstack:~# ln -s /etc/init.d/cinder-setup-backing-file /etc/rc2.d/S10cinder-
setup-backing-file
root@openstack:~# pvcreate /dev/loop2
root@openstack:~# vgcreate cinder-volumes /dev/loop2
```

7）重启服务。

```
root@openstack:~# service cinder-scheduler restart
root@openstack:~# service cinder-api restart
root@openstack:~# service cinder-volume restart
```

8）验证服务是否正常。

```
root@openstack:~# service cinder-scheduler status
root@openstack:~# service cinder-api status
root@openstack:~# service cinder-volume status
root@openstack:~# cinder service-list
+---------------+---------------+-------+------------+-------+----------------+
| Binary | Host | Zone | Status | State | Updated_at |
+---------------+---------------+-------+------------+-------+----------------+
| cinder-scheduler | openstack | nova | enabled | up | 2015-01-08T08:28:05.000000 |
| cinder-volume | openstack | nova | enabled | up | 2015-01-08T08:28:13.000000 |
+---------------+---------------+-------+------------+-------+----------------+
```

至此控制节点安装完成。

11.1.3 计算节点的安装

在计算节点安装之前，需要进行服务器环境的准备和设置。

服务器环境准备：

- 操作系统为 Ubuntu 12.04 LTS。
- 双网卡 eth0、eth1，以笔者的环境为例，网卡配置文件如下：

```
auto lo
iface lo inet loopback

#the admin interface
```

1107114476@qq.com
dolphinebook.com

```
auto eth0
#eth0 IP 配置信息
iface eth0 inet static
address 192.168.11.129
netmask 255.255.255.0
gateway 192.168.11.1
#eth1 实际不配置 IP
auto eth1
iface eth1 inet static
address 0.0.0.0
```

❑ 磁盘 30GB 以上。

首先也需要设置 apt 源，这一步骤请参考控制节点安装部分，不再赘述。

1. 安装 Neutron 和 OpenvSwitch

1）修改内核参数，开启反向过滤。

```
root@openstack:~# vim /etc/sysctl.conf
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
root@openstack:~# sysctl -p
```

2）安装 Neutron 软件包。

```
root@openstack:~# apt-get install neutron-common neutron-plugin-ml2 neutron-plugin-
openvswitch-agent openvswitch-datapath-dkms
```

3）修改 Neutron 配置文件。

① 编辑 /etc/neutron/neutron.conf 文件内容如下：

```
root@openstack:~# vim /etc/neutron/neutron.conf
[DEFAULT]
auth_strategy = keystone

rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_host = 192.168.11.128
rabbit_password = kvmpractice

core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True

[database]
connection = mysql://neutron:kvmpractice@192.168.11.128/neutron

[keystone_authtoken]
auth_uri = http://192.168.11.128:5000
auth_host = 192.168.11.128
auth_protocol = http
auth_port = 35357
admin_tenant_name = service
```

1107114476@qq.com
dolphinebook.com

```
admin_user = neutron
admin_password = kvmpractice
```

② 编辑 /etc/neutron/plugins/ml2/ml2_conf.ini 文件内容如下：

```
root@openstack:~# vim /etc/neutron/plugins/ml2/ml2_conf.ini
[ml2]
type_drivers = vlan
tenant_network_types = vlan
mechanism_drivers = openvswitch

[ml2_type_vlan]
network_vlan_ranges = physnet1:2:1024

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewall-
Driver
enable_security_group = True

[ovs]
tenant_network_type = vlan
bridge_mappings = physnet1:br-eth1
```

4）配置 Open vSwitch，增加网桥。

```
root@openstack:~# service openvswitch-switch restart
root@openstack:~# ovs-vsctl add-br br-int
root@openstack:~# ovs-vsctl add-br br-eth1
root@openstack:~# ovs-vsctl add-port br-eth1 eth1
```

2. 安装 nova compute 组件

1）安装 nova 软件包。

```
root@openstack:~# apt-get install nova-compute-kvm qemu-kvm
```

2）修改配置文件。

多数配置与控制节点相同，主要修改的是一些涉及本机 IP 地址的配置项。编辑 /etc/nova/nova.conf 文件内容如下：

```
root@openstack:~# vim /etc/nova/nova.conf
[DEFAULT]
rpc_backend = rabbit
rabbit_host = 192.168.11.128
rabbit_password = kvmpractice
my_ip = 192.168.11.129
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = 192.168.11.129
auth_strategy = keystone
glance_host = 192.168.11.128

#networking
```

1107114476@qq.com
dolphinebook.com

```
network_api_class = nova.network.neutronv2.api.API
neutron_url = http://192.168.11.128:9696
neutron_auth_strategy = keystone
neutron_admin_tenant_name = service
neutron_admin_username = neutron
neutron_admin_password = kvmpractice
neutron_admin_auth_url = http://192.168.11.128:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
security_group_api = neutron

#neutron metadata agent
service_neutron_metadata_proxy = true
neutron_metadata_proxy_shared_secret = kvmpractice

[database]
connection = mysql://nova:kvmpractice@192.168.11.128/nova

[keystone_authtoken]
auth_uri = http://192.168.11.128:5000
auth_host = 192.168.11.128
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = kvmpractice

[libvirt]
virt_type = kvm
```

3）重启 libvirt 服务和 nova-compute 服务。

```
root@openstack:~# service libvirt-bin restart
root@openstack:~# service nova-compute restart
```

至此计算节点安装完成。

11.2　OpenStack 实用技巧

11.2.1　RabbitMQ Cluster 搭建

RabbitMQ 是 OpenStack 各组件之间通信的枢纽，因此它的可用性非常重要。RabbitMQ Cluster 是一个很好的高可用解决方案。

一个 RabbitMQ Cluster 包括多个运行 RabbitMQ 的节点，这些节点之间共享 users、virtual hosts、queues、exchanges 等。接下来将介绍如何搭建一个 RabbitMQ Cluster。

1. 统一每个节点上的 Erlang cookie

RabbitMQ 是由 Erlang 语言写成的，Erlang 节点只能同与它拥有相同 cookie 的其他节点

1107114476@qq.com
dolphinebook.com

通信，因此，搭建 cluster 前要保证各个节点上的 Erlang cookie 相同。

cookie 的位置是 /var/lib/rabbitmq/.erlang.cookie。cookie 是一个任意长的英文字母串：

```
root@rabbit01:~# cat /var/lib/rabbitmq/.erlang.cookie
RZBHENFOEQAXCNAYSJCL
```

在统一 cookie 时最好是复制文件，不要复制字符串，而且复制完后要保证文件权限正确。

```
root@rabbit01:~# scp /var/lib/rabbitmq/.erlang.cookie root@rabbit02:/var/lib/
rabbitmq/.erlang.cookie
```

2. 配置 /etc/hosts

创建 RabbitMQ cluster 需要各节点之间能互相解析主机名，因此，要在 /etc/hosts 中加入配置，代码如下：

```
root@rabbit01:~#  cat /etc/hosts
127.0.0.1  localhost
192.168.11.128    rabbit01
192.168.11.129    rabbit02
root@rabbit02:~#  cat /etc/hosts
127.0.0.1  localhost
192.168.11.128    rabbit01
192.168.11.129    rabbit02
```

3. 把节点加入到集群中

RabbitMQ 在安装完后，默认属于一个 cluster，不过这个 cluster 内只有它本身这一个节点。

```
root@rabbit01 ~  # rabbitmqctl cluster_status
Cluster status of node rabbit@rabbit01 ...
[{nodes, [{disc, [rabbit@rabbit01]}]},
{running_nodes, [rabbit@rabbit01]},
{partitions, []}]...done.
```

下面要做的就是把其他节点加入到这个 cluster，加入的步骤如下：

```
root@rabbit02 ~  # rabbitmqctl stop_app
Stopping node rabbit@rabbit02 ...done.
root@rabbit02 ~  # rabbitmqctl join_cluster rabbit@rabbit01
Clustering node rabbit@rabbit02 with [rabbit@rabbit01] ...done.
root@rabbit02 ~  # rabbitmqctl start_app
Starting node rabbit@rabbit02 ...done.
```

然后现在查看一下 cluster status：

```
root@rabbit01:~# rabbitmqctl cluster_status
Cluster status of node 'rabbit@rabbit01' ...
[{nodes, [{disc, ['rabbit@rabbit01', 'rabbit@rabbit02']}]},
```

1107114476@qq.com
dolphinebook.com

```
 {running_nodes, ['rabbit@rabbit02', 'rabbit@rabbit01']},
 {cluster_name, <<"rabbit@rabbit01">>},
 {partitions, []}]
...done.
```

至此，cluster 已经搭建完成。

4. 配置 queue mirror

cluster 搭建完后还没有实现真正的高可用，默认情况下，一个 queue 只分布在一个节点上，如果这个节点出现故障，queue 中的 message 就会丢失。因此，要实现高可用，还要配置 queue mirror，把 queue 镜像到多个节点。方法是定义一个 policy，然后 RabbitMQ 会根据这个 policy 产生 queue 的镜像。

```
root@rabbit01:~# rabbitmqctl set_policy ha-all ^(?!amq\\.).* {"ha-mode":"all"}
```

这个命令定义了一个 policy，把所有名称符合 ^(?!amq\\.).* 正则表达式的 queue 镜像到 cluster 内所有节点，这个正则表达式的作用是只匹配 OpenStack 组件产生的 queue，忽略 RabbitMQ 默认产生的 queue，因为这些 queue 没有做高可用的必要。

```
root@rabbit01:~#  rabbitmqctl list_policies
Listing policies ...
/  HA all ^(?!amq\\.).* {"ha-mode":"all"} 0
...done.
```

11.2.2　利用 Haproxy 和 Keepalived 实现控制节点高可用

Haproxy 是一款开源的负载均衡软件，支持基于 TCP 和 HTTP 实现负载均衡。而 Keepalived 利用 VRRP 协议，可以检测节点状态，实现高可用。本节介绍利用 Haproxy 和 Keepalived 实现 OpenStack 控制节点的负载均衡及高可用。

假设现在有两个控制节点：ctrl01、ctrl02。每个节点上应该运行的服务如表 11-1 所示。

表 11-1　每个节点上应该运行的服务

Server	Nova	Keystone	Neutron	Glance	Cinder
ctrl01 192.168.11.128	nova-api nova-cert nova-conductor nova-scheduler nova-consoleauth nova-novncproxy	keystone	neutron-server neutron-dhcp-agent neutron-l3-agent neutron-metadata-agent neutron-openvswitch-agent	glance-api glance-registry	cinder-api cinder-scheduler cinder-volume
ctrl02 192.168.11.129	nova-api nova-cert nova-conductor nova-scheduler	keystone	neutron-server		cinder-api

从表 11-1 中可以看出，灰色的服务如 DHCP 只能运行在一个节点上，这个节点称为

1107114476@qq.com
dolphinebook.com

主控制节点，对于这样的服务难以实现高可用。负载均衡和高可用只针对其他黑色字体的服务。

ha01 和 ha02 是要安装 Haproxy 的两个节点，ha01 的 IP 地址为 192.168.11.101，ha02 的 IP 地址为 192.168.11.102。Keepalived 使用的虚拟 IP 地址（VIP）为 192.168.11.100。

下面在两个节点安装软件包。

```
root@ha01:~# apt-get install haproxy keepalived -y
```

然后配置 Haproxy，配置文件如下：

```
root@ha01:~# vim /etc/haproxy/haproxy.cfg
global
  chroot  /var/lib/haproxy
  daemon
  group  haproxy
  maxconn  4000
  ulimit-n 65536
  log   127.0.0.1  local1 info
  pidfile  /var/run/haproxy.pid
  stats socket /var/lib/haproxy/socket level admin
  user  haproxy

defaults
  log  global
  maxconn  8000
  option  redispatch
  retries  3
  timeout  http-request 20s
  timeout  queue 1m
  timeout  connect 10s
  timeout  client 1m
  timeout  server 1m
  timeout  check 10s
  timeout  tunnel 3600s # TCP timeout, used by novncproxy's websocket

frontend unsecured vdesktop-stage.cn1.global.ctrip.com:80
  mode http
redirect scheme https if !{ ssl_fc }

listen dashboard_cluster
  bind 192.168.11.100:443 ssl crt /etc/haproxy/openstack.pem
  mode http
  reqadd X-Forwarded-Proto:\ https
  balance source
  option  tcpka
  option  httpchk
  option  httplog
  server ctrl01 192.168.11.128:80 check inter 2000 rise 2 fall 3
```

1107114476@qq.com
dolphinebook.com

```
  server ctrl02 192.168.11.129:80 check inter 2000 rise 2 fall 3

listen glance_api_cluster
  bind 192.168.11.100:9292
  mode http
  balance  source
  option  tcpka
  option  httpchk
  option  httplog
  # should only have one server for glance, as we are using file sytem as image store
  server ctrl01 192.168.11.128:9292 check inter 2000 rise 2 fall 3

listen glance_registry_cluster
  bind 192.168.11.100:9191
  balance  source
  option  tcpka
  option  tcplog
  # should only have one server for glance, as we are using file sytem as image store
  server ctrl01 192.168.11.128:9191 check inter 2000 rise 2 fall 3

listen keystone_admin_cluster
  bind 192.168.11.100:35357
  mode http
  balance roundrobin
  option  tcpka
  option  httpchk
  option  httplog
  server ctrl01 192.168.11.128:35357 check inter 2000 rise 2 fall 3
  server ctrl02 192.168.11.129:35357 check inter 2000 rise 2 fall 3

listen keystone_public_internal_cluster
  bind 192.168.11.100:5000
  bind 192.168.11.100:8443 ssl crt /etc/haproxy/openstack.pem
  mode http
  reqadd X-Forwarded-Proto:\ https
  balance  roundrobin
  option  tcpka
  option  httpchk
  option  httplog
  server ctrl01 192.168.11.128:5000 check inter 2000 rise 2 fall 3
  server ctrl02 192.168.11.129:5000 check inter 2000 rise 2 fall 3

listen nova_compute_api_cluster
  bind 192.168.11.100:8774
  mode http
  balance roundrobin
  option  tcpka
  option  httpchk
  option  httplog
  server ctrl01 192.168.11.128:8774 check inter 2000 rise 2 fall 3
```

1107114476@qq.com
dolphinebook.com

```
  server ctrl02 192.168.11.129:8774 check inter 2000 rise 2 fall 3

listen nova_metadata_api_cluster
  bind 192.168.11.100:8775
  balance roundrobin
  option  tcpka
  option  tcplog
  server ctrl01 192.168.11.128:8775 check inter 2000 rise 2 fall 3
  server ctrl02 192.168.11.129:8775 check inter 2000 rise 2 fall 3

listen cinder_api_cluster
  bind 192.168.11.100:8776
  mode http
  balance  source
  option  tcpka
  option  httpchk
  option  httplog
  server ctrl01 192.168.11.128:8776 check inter 2000 rise 2 fall 3
  server ctrl02 192.168.11.129:8776 check inter 2000 rise 2 fall 3

listen ceilometer_api_cluster
  bind 192.168.11.100:8777
  mode http
  balance  source
  option  tcpka
  option  httpchk
  option  httplog
  server ctrl01 192.168.11.128:8774 check inter 2000 rise 2 fall 3
  server ctrl02 192.168.11.129:8774 check inter 2000 rise 2 fall 3

listen neutron_api_cluster
  bind 192.168.11.100:9696
  mode http
  balance  source
  option  tcpka
  option  httplog
  server ctrl01 192.168.11.128:9696 check inter 2000 rise 2 fall 3
  server ctrl02 192.168.11.129:9696 check inter 2000 rise 2 fall 3

frontend novncproxy
  bind 192.168.11.100:6080 ssl crt /etc/haproxy/vdi_pd.pem

  acl is_websocket hdr(Upgrade) -i WebSocket
  use_backend websocket-backend if is_websocket

  default_backend novncproxy-backend

backend websocket-backend
  balance roundrobin
  option http-server-close
```

1107114476@qq.com
dolphinebook.com

```
  option forceclose
  no option httpclose
  option  tcplog
  server ctrl01 192.168.11.128:6080 check inter 2000 rise 2 fall 3
backend novncproxy-backend
  mode http
  balance roundrobin
  option  httplog
  server ctrl01 192.168.11.128:6080 check inter 2000 rise 2 fall 3
```

ha01 和 ha02 上的配置文件要保持一致。Haproxy 使用 rsyslog 记录日志，另外还需要额外的配置让 Haproxy 打印日志到特定的文件，代码如下：

```
root@ha01:~# vim /etc/rsyslog.d/49-haproxy.conf
# Create an additional socket in haproxy's chroot in order to allow logging via
# /dev/log to chroot'ed HAProxy processes
$AddUnixListenSocket /var/lib/haproxy/dev/log

# Send HAProxy messages to a dedicated logfile
if $programname startswith 'haproxy' then /var/log/haproxy.log
&~
```

然后重启 rsyslog 服务，代码如下：

```
root@ha01:~# service rsyslog restart
```

这样，Haproxy 就会把日志打印到 /varlog/haproxy.log 中。

接下来配置 Keepalived，两个节点上配置文件有所不同，请注意：

```
#ha01 作为 master 节点
root@ha01:~# cat /etc/keepalived/keepalived.conf
vrrp_script chk_haproxy {
    script "sh /etc/keepalived/check.sh" # verify the pid existance
    interval 2                           # check every 2 seconds
    weight 2                             # add 2 points of prio if OK
}

vrrp_instance VI_1 {
    interface eth0                       # interface to monitor
    state MASTER
    virtual_router_id 51                 # Assign one ID for this route
    priority 100                         # 100 on master, 99 on backup
    virtual_ipaddress {
        192.168.11.100                   # the virtual IP
   }
   track_script {
        chk_haproxy
   }
}

#ha02 作为 backup 节点
```

1107114476@qq.com
dolphinebook.com

```
root@ha02:~# cat /etc/keepalived/keepalived.conf
vrrp_script chk_haproxy {
    script "sh /etc/keepalived/check.sh"    # verify the pid existance
    interval 2                              # check every 2 seconds
    weight 2                                # add 2 points of prio if OK
}

vrrp_instance VI_1 {
    interface eth0                          # interface to monitor
    state BACKUP
    virtual_router_id 51                    # Assign one ID for this route
    priority 99                             # 100 on master, 99 on backup
    virtual_ipaddress {
        192.168.11.100                      # the virtual IP
    }
    track_script {
        chk_haproxy
    }
}
```

配置文件中提到的 /etc/keepalived/check.sh 脚本是用来检查 haproxy 进程状态的，代码如下：

```
root@ha01:~# cat /etc/keepalived/check.sh
#! /bin/bash
if [ "`pgrep -f "/usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -D -p /var/run/
haproxy.pid"`" = "" ];then
    service haproxy start
    sleep 3
    if [ "`pgrep -f "/usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -D -p /var/run/
    haproxy.pid"`" = "" ];then
    killall keepalived
    fi
fi
```

其中，pgrep 匹配的进程名可以根据自己环境的不同进行更改，这是因为不同环境 haproxy 的进程名可能不同。

重启 Keepalived 服务，代码如下：

```
root@ha01:~# service keepalived restart
```

如果配置成功，会在 MASTER 节点的网卡上看到虚拟 IP，之后访问 OpenStack 的服务就可以通过这个虚拟 IP。

```
root@ha01:~# ip a s eth0
2: eth0: <BROADCAST, MULTICAST, UP, LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 06:07:85:48:0e:76 brd ff:ff:ff:ff:ff:ff
    inet 192.168.11.101/24 brd 192.168.11.255 scope global eth0
    inet 192.168.11.100/32 scope global eth0
    inet6 fe80::407:85ff:fe48:e76/64 scope link
        valid_lft forever preferred_lft forever
```

1107114476@qq.com
dolphinebook.com

11.2.3　冷迁移虚拟机的方法

在实际生产环境中，很多情况下并没有使用共享存储来存储虚拟机，因此，热迁移虚拟机变得很困难。如果这时有虚拟机迁移的需求，可以尝试冷迁移，下面将介绍冷迁移虚拟机的方法。

首先，查看虚拟机，找到要迁移虚拟机的 id，并关掉虚拟机，代码如下：

```
root@ctrl01:~# nova list --all-tenants
root@ctrl01:~# nova stop INSTANCEID
```

在 /var/lib/nova/instances 下找到对应的虚拟机的目录，目录名就是虚拟机的 id，将该目录复制到目的主机下的 /var/lib/nova/instances 中，一定要确保权限一致，复制完检查目录的 owner 是不是 nova 用户，如果不是，将会导致虚拟机起不来。

更改 nova 数据库中的表 instances，其中的 host、node 字段为新的目的计算节点名字。

```
root@ctrl01~  # mysql -u root -p
mysql> use nova;
mysql> update instances set host='COMPUTENODENAME', node='COMPUTENODENAME' where
uuid='INSTANCEID';
```

现在启动虚拟机，虚拟机就会运行在目标计算节点上了。

```
root@ctrl01~  # nova start INSTANCEID
```

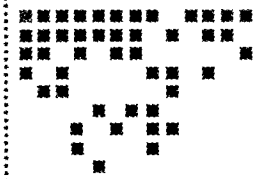
不要忘了把旧节点上的该虚拟机删除，否则可能引起冲突。

11.3　本章小结

OpenStack 非常强大而复杂，由于篇幅限制，本章只能简单介绍 OpenStack 的概念，笔者的生产环境的搭建方法和一些经验，不能完全展开介绍。好在 OpenStack 的资料非常多，如果读者有兴趣进一步研究和使用，资料很方便得到。

下一章为读者介绍 OpenNebula，OpenNebula 相对搭建使用更容易些，更适用于私有云管理。

1107114476@qq.com
dolphinebook.com



Chapter 12 第 12 章

利用 OpenNebula 管理 KVM

第 11 章介绍了如何利用 OpenStack 管理 KVM，本章将介绍另外一款支持 KVM 的开源管理平台——OpenNebula，它也是一个功能非常强大的管理平台。本章侧重从实践的角度介绍 OpenNebula，内容主要包括 OpenNebula 介绍及基本原理、OpenNebula 环境配置、OpenNebula 常见故障处理这三部分。

12.1 OpenNebula 架构

OpenNebula 是一套功能丰富且灵活性非常强的解决方案，用于构建和管理企业云及数据中心虚拟化。OpenNebula 支持 Xen、KVM、VMware ESX 虚拟化引擎，可以一起混合建立和管理私有云，同时还提供了 Deltacloud（红帽公司 2009 年 9 月推出的一套开源的 API）适配器，与 Amazon EC2 的 API 兼容。

12.1.1 OpenNebula 架构介绍

在 OpenNebula 的架构中，一个经典的 OpenNebula 集群至少需要一个 Front-End（前端）和一个用于运行虚拟机的 Host（主机）。在网络方面，至少需要一个物理网络，并且确保 Front-End 能和所有的 Host 连通。图 12-1 所示为最基本的 OpenNebula 架构图。

图 12-1 所示的架构图是 OpenNebula 最精简的结构，OpenNebula 本身具有非常强的延展性和可扩展性，OpenNebula front-end 可以配置高可用，做到负载均衡。多台主机可以组成一个集群，一个 IDC 机房中可以包含多个集群，OpenNebula 还支持管理多个 IDC 机房。

1107114476@qq.com
dolphinebook.com

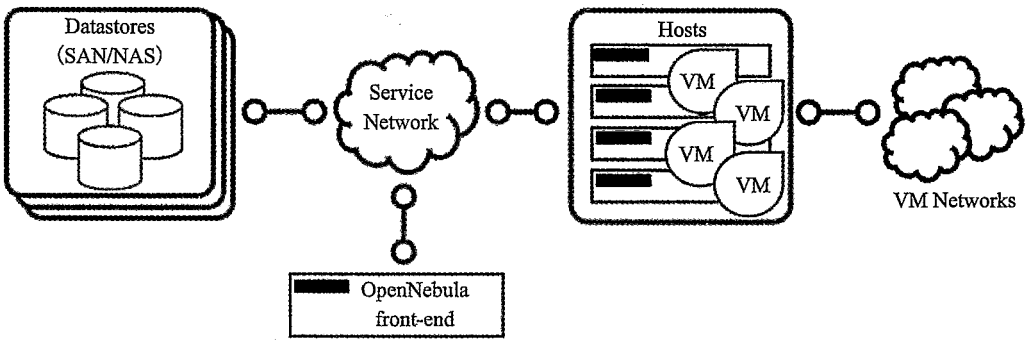


图 12-1　OpenNebula 基本架构图（来自 OpenNebula 官方资料）

OpenNebula 可扩展性灵活，能适用于多种应用场景。例如，简单的企业私有云，用 OpenNebula 可以将同一机房的多台主机统一管理，仅需简单配置，就可以实现虚拟机创建、开机、关机、删除等一系列操作，非常方便地实现完整的虚拟机生命周期管理。

OpenNebula 还提供了混合云的解决方案，方便在现有的私有云的基础上接入外部的资源，形成强大的混合云。OpenNebula 也可以用于构建公有云，通过简单的定制，就能实现多机房主机资源的管理，对外提供统一的 Web UI（界面），以及用户授权管理，对外部用户来说，只需要购买相应的权限，便可以使用云主机资源。

12.1.2　OpenNebula 组件介绍

在一个 OpenNebula 集群中包含如下 4 个基础组件：

- Front-End(前端)：用于运行 OpenNebula 服务。
- Host（宿主机）：用于运行所有的虚拟机。
- Datastores（数据存储）：用于实际存放虚拟机的硬盘。
- Network（网络）。

其中，网络需要分为两部分，一部分是连通 OpenNebula 集群中各组件，另一部分是为虚拟机提供的虚拟网络。下面就对这 4 个组件进行展开说明。

1. 前端（Front-End）

这是一个服务器角色，其中包含了诸多功能组件，如 OpenNebula 管理进程 oned、调度器 mm_sched、一个 Web 接口服务 sunstone-server。这 3 个功能组件可以理解为 OpenNebula 的大脑，对于安全性和性能要求较高的用户来说，这 3 个服务是可以部署在 3 台独立服务器上的，各组件之间通过 XML-RPC（XML Remote Procedure Call，XML 远程方法调用）接口进行通信。OpenNebula 默认数据库采用 SQLite，因为它的配置非常简单，但是如果用于生产环境或者大规模部署的话，还是推荐使用 MySQL 来替换。Front-End 本身还可以配置实现高可用，以避免 OpenNebula 出现宕机。

1107114476@qq.com
dolphinebook.com

2. 宿主机（Host）

Host 用于运行虚拟机的服务器，OpenNebula 支持多个虚拟化平台，目前主流的 Xen、KVM、VMware 都是支持的，当然 KVM 是 OpenNebula 默认支持最好的虚拟化平台。在宿主机层面，OpenNebula 也可以做到虚拟机的高可用，当一台宿主机宕机之后，上面运行的虚拟机根据 OpenNebula 中定义的一些规则，自动在正常的宿主机上开启。

3. 数据存储（Datastores）

数据存储用于存放虚拟机的硬盘文件，它可以是任意一种存储介质，NAS（网络附加存储）、SAN（存储区域网络）、直连存储设备（宿主机本地硬盘）都可以作为 OpenNebula 的数据存储。根据用途，数据存储可以细分为三大类：系统数据存储、镜像数据存储、文件数据存储。

4. 网络 Network

OpenNebula 从功能上定义了两类不同的网络。第一，服务网络，它用于 Front-End（前端）程序对宿主机进行管理、监控和镜像文件移动等操作，推荐的做法是建立单独的服务网络。第二，虚拟机实例网络，用于提供给虚拟机用的网络，通俗来说就是业务网络。这个网络相对服务网络要复杂很多，不仅需要考虑连通性，还需要保证虚拟之间必要的隔离和安全性。目前，虚拟机实例网络支持多种驱动，如 802.1q（通过 vlantag 的方式做网络隔离）、ebtables（通过 ebtables 策略来隔离虚拟机网络）、ovswitch（使用 Open vSwitch Virtual Switch 来实现网络隔离等复杂的软件定义网络功能）等。其中，Open vSwitch 在生产环境使用得最多，使用 Open vSwitch 能为虚拟机构建复杂的网络架构，同时增强网络的可管理性。

12.2 在 KVM 中使用 OpenNebula

通过上边的内容，使我们对 OpenNebula 有了一些了解，从本节开始将介绍如何从零开始在 CentOS 系统之上搭建第一个 OpenNebula 集群，并在搭建完成之后，开始创建第一台虚拟机。本节还将介绍一个非常实用的功能配置，将默认虚拟机运行在宿主机的本地硬盘上。因为对于大部分用户来说，未必都有性能非常强的共享存储资源，如果使用 OpenNebula 默认的方式将虚拟机运行在共享存储上，性能会非常不理想。所以，为了虚拟机能得到更好的性能，将虚拟机运行在宿主机的本地硬盘上，就变得非常有意义，这一点正好符合本书的宗旨——实践，所以，本节会把这部分介绍得比较细。

12.2.1 在 CentOS 6.5 上快速搭建 OpenNebula

在 CentOS 6.5 上搭建 OpenNebula 集群环境，官方建议至少需要两台服务器（如果只是测试环境的话，可以只用一台物理服务器），用于部署两个角色——frontend（前端程序）和 nodes（计算节点，也就是宿主机），下面将介绍如何使用两台服务器分别安装 frontend 和 node。

1107114476@qq.com
dolphinebook.com

1. 前端安装和配置

（1）前端安装

① 在安装前端的过程中需要依赖 EPEL（Extra Packages for Enterprise Linux）的 yum 源，所以需要先安装 EPEL，代码如下：

```
[root@front-end-01 ~]# yum install epel-release -y
```

② 添加 OpenNebula 的 yum 安装源，安装当前最新的 OpenNebula 稳定版 4.10，代码如下：

```
[root@front-end-01 ~]# cat << EOT > /etc/yum.repos.d/opennebula.repo
[opennebula]
name=opennebula
baseurl=http:// downloads.opennebula.org/repo/4.10/CentOS/6/x86_64/
enabled=1
gpgcheck=0
EOT
```

③ 使用 yum 安装核心服务和管理 UI，代码如下：

```
[root@front-end-01 ~]# yum install opennebula-server opennebula-sunstone -y
```

④ 使用 gem 安装 OpenNebula 所有的依赖包，代码如下：

```
[root@front-end-01 ~]# /usr/share/one/install_gems
Distribution "redhat" detected.
About to install these dependencies:
* gcc
* rubygem-rake
* libxml2-devel
* libxslt-devel
* sqlite-devel
* curl-devel
* mysql-devel
* ruby-devel
* make
Press enter to continue...          // 无需任何选择，一直回车到底，看提示确认安装是否成功
```

经过上面 4 步操作，前端的组件和依赖都安装完了。

（2）配置管理平台 sunstone

先开始配置 OpenNebula 的管理平台 sunstone，编辑 /etc/one/sunstone-server.conf 文件，将“: host: 127.0.0.1”改为实际环境的 IP 地址，代码如下：

```
[root@front-end-01 ~]# vim /etc/one/sunstone-server.conf
    <-- 此处省略无用输出内容 28 行 -->
# Server Configuration
#
:host: 10.168.107.84                // 此处设定 front-end 服务器 IP
:port: 9869                         // 默认端口是 9869，可以根据需要更改
```

1107114476@qq.com
dolphinebook.com

配置完毕之后，启动服务 front-end 角色中的两个服务，代码如下：

```
# service opennebula start                                    //OpenNebula 核心服务
# service opennebula-sunstone start                           //OpenNebula web UI 服务
```

（3）配置 nfs 服务器，将 oneadmin 的家目录共享出来

注意，anonuid 和 anongid 分别是 oneadmin 的 uid 和 gid，意思是用户只用 root 挂载，就会按 oneadmin 用户授权的配置完成。

nfs 配置完成之后，启动 nfs 服务，并使用 showmount 验证 nfs 共享内容，代码如下：

```
[root@front-end-01 ~]# cat /etc/exports

/var/lib/one/ *(rw,sync,no_subtree_check,root_squash,anonuid=9869,anongid=9869)
    [root@front-end-01 ~]# service rpcbind start       //启动 nfs 相关服务
Starting rpcbind:                                          [  OK  ]
[root@front-end-01 ~]# service nfs start
Starting NFS services:                                     [  OK  ]
Starting NFS quotas:                                       [  OK  ]
Starting NFS mountd:                                       [  OK  ]
Starting NFS daemon:                                       [  OK  ]
Starting RPC idmapd:                                       [  OK  ]
[root@front-end-01 ~]# showmount -e localhost
Export list for localhost:
/var/lib/one *
```

（4）配置 ssh 证书认证

切换到 oneadmin 账户下，配置家目录下的 .ssh/config 文件，其中，添加 StrictHostKeyChecking no，此处配置的作用是自动把计算机的密匙加入 .ssh/known_hosts；添加 UserKnownHostsFile /dev/null 的作用是实时删除 known_hosts 文件，每次都是新的，以避免冲突。配置完成之后，将 .ssh/config 文件的权限配置为 600，代码如下：

```
[root@front-end-01 ~]# su - oneadmin
[oneadmin@front-end-01 ~]$ cat << EOT > ~/.ssh/config
> Host *
>     StrictHostKeyChecking no
>     UserKnownHostsFile /dev/null
> EOT
[oneadmin@front-end-01 ~]$ chmod 600 ~/.ssh/config
```

经过上面的操作，front-end 角色就部署完成了。

2. 计算节点部署和配置

（1）计算节点部署

配置 OpenNebula 的 yum 安装源，同 Front-End 中一样，使用目前最新的稳定版 4.10，代码如下：

```
# cat << EOT > /etc/yum.repos.d/opennebula.repo
```

1107114476@qq.com
dolphinebook.com

```
[opennebula]
name=opennebula
baseurl=http:// downloads.opennebula.org/repo/4.10/CentOS/6/x86_64/
enabled=1
gpgcheck=0
EOT
```

在安装 node 角色过程中，OpenNebula 必需的安装包：

```
# yum install opennebula-node-kvm -y
```

启动服务，node 节点必需的服务：

```
# service messagebus start        //用于进程间通信的服务
# service libvirtd start          //KVM 的管理工具集
```

（2）配置网桥

OpenNebula 默认使用最简单的网桥作为虚拟机的网络连接。编辑对应的网卡配置文件，配置完成之后，重启网络，网桥就配置完成了。

```
[root@ceph-host-01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-em2
DEVICE=em2
BOOTPROTO=none
IPV6INIT=no
NM_CONTROLLED=no
ONBOOT=yes
TYPE=Ethernet
BRIDGE=br2
[root@ceph-host-01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-br2
DEVICE=br2
TYPE=Bridge
BOOTPROTO=static
NM_CONTROLLED=no
ONBOOT=yes
IPADDR=10.168.107.13
NETMASK=255.255.255.0
[root@ceph-host-01]# ifup br2
Determining if ip address 10.168.107.13 is already in use for device br2...
[root@ceph-host-01]# ifup em2
```

（3）挂载 front-end 中共享出来的 nfs 文件夹

编辑 /etc/fstab，添加 front-end 共享出来的目录，然后直接使用 mount -a 根据 fstab 中定义的内容挂载，如果挂载成功，说明配置没问题了。

```
10.168.107.84:/var/lib/one/ /var/lib/one/ nfs  soft,intr,rsize=8192,wsize=8192 0 0
[root@ceph-host-01 ~]# mount -a
[root@ceph-host-01 ~]# df -Th
File; system          Type    Size  Used Avail Use% Mounted on
```

1107114476@qq.com
dolphinebook.com

```
/dev/mapper/kvmvg-root
                      ext4     30G  3.5G   25G  13% /
tmpfs                 tmpfs    32G  8.0K   32G   1% /dev/shm
/dev/sda1             ext3    248M   68M  168M  29% /boot
10.168.107.84:/var/lib/one/
                      nfs      46G  2.2G   41G   6% /var/lib/one
```

提示 先编辑 /etc/fstab，再使用 mount -a 挂载，这个习惯很重要。很多时候，会先使用 mount 命令直接挂载共享目录，然后再编辑 /etc/fstab，但是，如果 /etc/fstab 的内容编辑时有问题，可能会导致系统下次重启时报错，系统启动不了。对于没有带外管理又不在身边的服务器，就非常麻烦了。

3. 登录 OpenNebula 管理平台

前端和计算节点角色部署完成之后，就可以打开 OpenNebula 管理平台了，剩下的大部分配置操作，可以直接在管理平台上操作。但是，OpenNebula 还提供了非常强大的命令行工具，两者都可以操作管理 OpenNebula，读者可以根据自己的方式来操作。下面介绍如何登录管理平台。

各服务启动完成之后，在浏览器中输入 http: //10.168.107.84: 9869/，这个地址是配置文件“/etc/one/sunstone-server.conf”中配置的 IP，如图 12-2 所示。

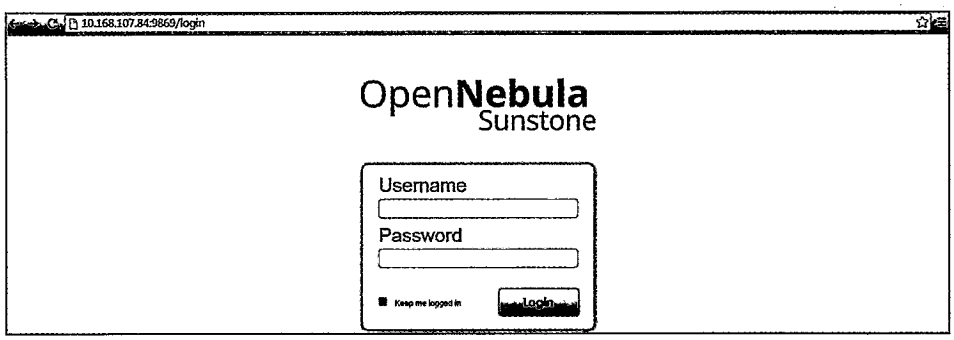


图 12-2 OpenNebula UI 登录界面

用户名密码在 oneadmin 账号下的“.one/one_auth”文件中：

```
[oneadmin@front-end-01 ~]$ cat .one/one_auth
oneadmin:9a1dcbaef562404f5ba4eb022c532ea5
```

登录之后的界面如图 12-3 所示，此时 OpenNebula 中还没有任何资源，通过 OpenNebula UI 也做不了任何事情，后续章节将介绍如何添加各类资源。

至此，OpenNebula 在 Centos 6.5 系统上的最简部署已介绍完了。

1107114476@qq.com
dolphinebook.com

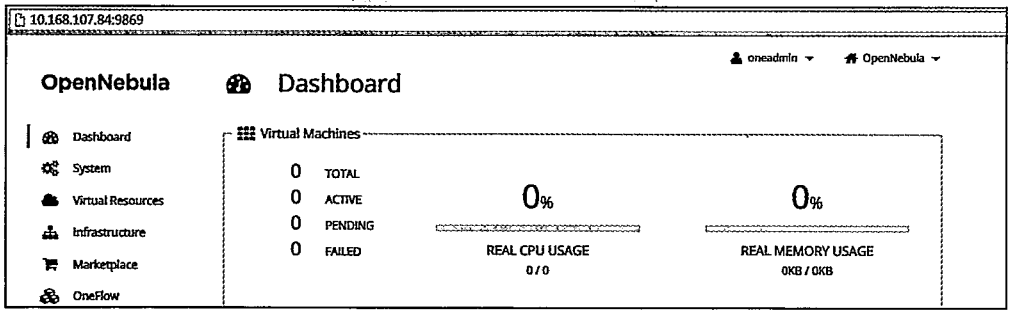


图 12-3　OpenNebula UI 主界面

12.2.2　使用 OpenNebula 创建第一台 KVM 虚拟机

在上一节中，已经完成了 OpenNebula 平台的基础搭建，但是，要通过 OpenNebula 创建第一台虚拟机，还需要添加和配置一些资源。配置 OpenNebula 有两种方式：一是在 OpenNebula UI 界面完成；二是通过 OpenNebula 提供的命令行工具完成。这两者从功能上来说，命令行工具要更加强大一些，对于技术人员来说，可能更倾向于使用命令行工具，所以，后续的演示操作都采用命令行的方式进行，同时可以通过 OpenNebula UI 进行验证。

1. 在 OpenNebula 平台中添加 KVM 宿主机

切换到 oneadmin 用户，使用 onehost create 命令添加 IP 为 10.168.107.13 的宿主机到 OpenNebula 平台，参数 -i 指定的是虚拟化驱动类型为 KVM，-v 指定的是虚拟化类型为 KVM，-n 指定虚拟机的网络类型 dummy 表示默认的网桥模式，完整的命令如下：

```
[root@front-end-01 ~]# su - oneadmin
[oneadmin@front-end-01 ~]$ onehost create 10.168.107.13 -i kvm -v kvm -n dummy
ID: 8
```

执行完添加命令之后，使用“onehost list”查看宿主机状态，会发现此时状态为 init，表示 OpenNebula 正在对宿主机进行初始化。

```
[oneadmin@front-end-01 ~]$ onehost list
 ID NAME            CLUSTER   RVM      ALLOCATED_CPU      ALLOCATED_MEM STAT
  8 10.168.107.13   -           0                  -                  - init
```

稍等片刻，再次查看，会看到状态变成 on，同时宿主机的一些配置信息也显示了。

```
[oneadmin@front-end-01 ~]$ onehost list
 ID NAME            CLUSTER   RVM      ALLOCATED_CPU      ALLOCATED_MEM STAT
  8 10.168.107.13   -           0      0 / 2400 (0%)    0K / 62.9G (0%) on
```

2. 在 OpenNebula 平台中添加虚拟机网络

（1）定义网卡的配置文件

网卡的配置文件命名为 vm-network1.one，其中，NAME 表示网卡名称，BRIDGE 指定

1107114476@qq.com
dolphinebook.com

网桥名称，本例使用宿主机中创建的网桥 br2，AR 表示网卡的 IP 定义，TYPE 指定 IP 类型是 ipv4，IP 指定详细的 IP 地址，SIZE 指定 IP 的数量。

```
NAME = "inside-network"
BRIDGE = br2
AR = [
    TYPE = IP4,
    IP = 192.168.20.100,
    SIZE = 3
]
```

（2）将网卡配置添加到 OpenNebula 平台

使用“onevnet create”命令将刚才定义的网卡配置添加到 OpenNebula 平台，再使用“onevnet list”命令可以参看结果。

```
[oneadmin@front-end-01 ~]$ onevnet create vm-network1.one

ID: 0
[oneadmin@front-end-01 ~]$ onevnet  list
  ID USER            GROUP         NAME                CLUSTER    BRIDGE    LEASES
   0 oneadmin        oneadmin      inside-network      -          br2            0
```

3. 在 OpenNebula 平台中添加虚拟机镜像

虚拟机镜像的添加有两种方式：①手动上传自己制作的虚拟机镜像；②使用 OpenNebula 提供的虚拟机镜像。一般来说，大多数公司都会对虚拟机镜像做一些针对自己业务或者安全的定制。如果自己没有做定制的话，直接使用 OpenNebula 提供的镜像也是非常不错的选择。

使用“oneimage create”命令添加虚拟机镜像，通过参数 --name 指定镜像名称；--path 指定镜像地址（OpenNebula 官方网址为 http: //appliances.c12g.com，读者可以打开这个链接，选择想要下载的镜像）；参数 --driver 指定硬盘格式，根据实际情况填写硬盘格式，本处为 qcow2；参数 --datastore 指定存储池，默认是 default，完整命令如下：

```
[oneadmin@front-end-01 ~]$ oneimage create --name "CentOS-6.5" \
>     --path " http:// appliances.c12g.com/CentOS-6.5/centos6.5.qcow2.gz" \
>     --driver qcow2 \
>     --datastore default
ID: 1
```

使用“oneimage list”命令查看 OpenNebula 中已有的镜像列表，刚才创建的 CentOS 7.0 镜像当前状态是 lock，这是正常的，因为这个镜像需要从 OpenNebula 网站下载。

```
[oneadmin@front-end-01 ~]$ oneimage list
  ID USER     GROUP      NAME            DATASTORE      SIZE TYPE PER STAT RVMS
   0 oneadmin   oneadmin   CentOS-6.5     default      10G OS    No lock     0
```

从官网下载镜像的时间比较长，待下载完成之后，再次使用“oneimage list”命令查看，状态变为 rdy，代码如下：

1107114476@qq.com
dolphinebook.com

```
[oneadmin@front-end-01 ~]$ oneimage list
  ID USER     GROUP      NAME            DATASTORE      SIZE TYPE PER STAT RVMS
   0 oneadmin   oneadmin   CentOS-6.5    default      10G OS    No rdy      0
```

4. 在 OpenNebula 平台中添加虚拟机模板

首先介绍两个概念：镜像和模板。在 OpenNebula 中镜像是指安装完系统的虚拟机硬盘文件。模板是指完整地定义了虚拟机配置，包括 CPU、内存、网卡，使用哪个镜像等信息，也可以说模板就是一台虚拟机的配置描述。在 OpenNebula 中创建虚拟机，都是通过模板定义的内容来创建的。

通过“ onetemplate create”命令创建模板，参数 --name 指定模板名称，--cpu 指物理占 CPU 的百分比，--vcpu 指虚拟 CPU 个数，--memory 指内存大小，单位为 MB，--arch 指虚拟机架构，--disk 指镜像名称，--nic 指虚拟机网卡名称 --vnc 指开启远程控制功能，--ssh 指开启 ssh 功能，完整的命令如下：

```
[oneadmin@front-end-01 ~]$onetemplate create --name "CentOS-6.5-template"
--cpu 1 --vcpu 1 --memory 512 \
--arch x86_64 --disk "CentOS-6.5" --nic "inside-network" --vnc \
--ssh
```

使用 onetemplate list 命令查看配置结果，模板添加完成。

```
[oneadmin@front-end-01 ~]$ onetemplate list
  ID USER          GROUP          NAME                                REGTIME
   3 oneadmin      oneadmin       CentOS-6.5-template          01/17 15:18:26
```

5. 在 OpenNebula 平台中创建第一台虚拟机

使用 onetemplate instantiate 命令可以创建虚拟机：

```
[oneadmin@front-end-01 ~]$ onetemplate instantiate "CentOS-6.5-template" --name
"my-first-vm"
VM ID: 5
```

创建之后立即使用 onevm list 查看虚拟机状态，此时为 prol，虚拟机创建过程中，状态依次从 PENDING（挂起）到 PROLOG（开始创建），最后到 RUNNING（运行），代码如下：

```
[oneadmin@front-end-01 ~]$ onevm list
    ID USER     GROUP    NAME            STAT UCPU    UMEM HOST             TIME
     5 oneadmin oneadmin my-first-vm     prol    0      0K 10.168.107   0d 00h00
```

稍等片刻，再使用 onevm list 查看虚拟机状态，即可看到虚拟机状态变成 runn 了，代码如下：

```
[oneadmin@front-end-01 ~]$ onevm list
    ID USER     GROUP    NAME            STAT UCPU    UMEM HOST             TIME
     5 oneadmin oneadmin my-first-vm     runn    0      0K 10.168.107   0d 00h00
```

通过上面几步操作之后，第一台虚拟机已经创建完成了，此时可以登录 OpenNebula UI，

1107114476@qq.com
dolphinebook.com

可以直观地看到创建的虚拟机。登录页面，如图 12-4 所示，选择左侧 Virtual Machines 菜单，看到创建的虚拟机 my-first-vm 状态是 RUNNING。

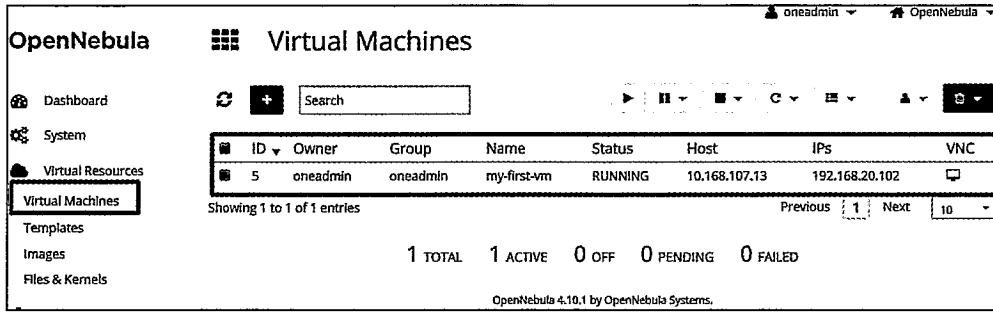


图 12-4 虚拟机展示页面

单击最右侧的显示器图标，打开虚拟机的远程终端，如图 12-5 所示。

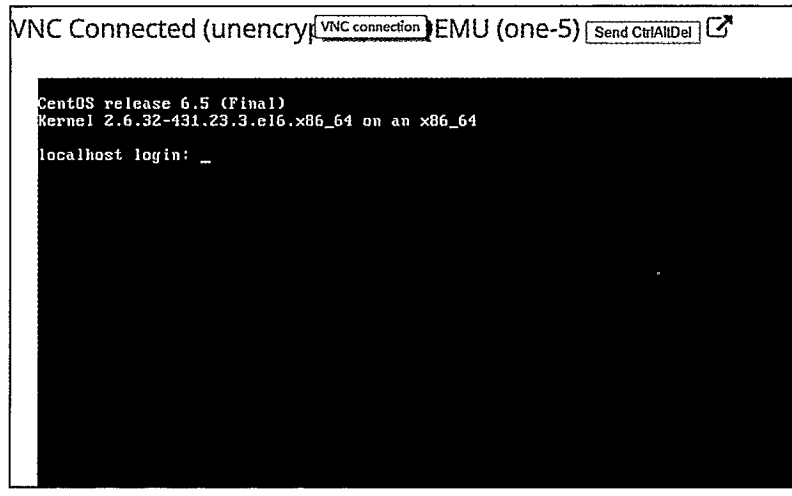


图 12-5 虚拟机远程终端界面

至此，已经使用 OpenNebula 创建了第一台 KVM 虚拟机。但是这台虚拟机显然还不能使用，第一，这台虚拟机中现在没有配置 IP，OpenNebula 对于虚拟机 IP 配置支持并不好；虚拟机 IP 配置方法主要有 3 种：① 使用 DHCP；② 手动注入 IP 配置脚本；③ 手动配置 IP。前两种方法，在实际的应用场景中都有，第三种方法适合少量测试时，本书的第 16 章将对虚拟机 IP 配置进行详细介绍，在此就不再重复了。

12.2.3 使虚拟机硬盘创建在宿主机本地硬盘上

OpenNebula 在默认配置中，将虚拟机的磁盘镜像存储在共享文件系统上，这种配置能方便实现对业务几乎无感知的故障迁移，但同时也带来如下两个问题：

1107114476@qq.com
dolphinebook.com

1）增加了一个集中式的故障点，一旦共享存储出现问题，上面的全部虚拟机都会受到影响，甚至存在数据丢失的风险，对于特别是包含 DB 类业务的场景下，需要慎用。

2）虚拟机 I/O 性能问题。普通的共享文件系统，性能普遍表现一般，虚拟机 I/O 性能与承载的虚拟机数量成反比例关系。

为了解决这两个问题，本节将着重介绍如何使用 OpenNebula 让虚拟机硬盘创建在宿主机本地硬盘上。这样既可以避免因为共享文件系统故障而导致大量虚拟机受影响，又可以通过在宿主机上配置高性能 SSD 盘，使虚拟机能获得非常高的 I/O 性能。

1. OpenNebula 数据资源类型介绍

默认情况下，OpenNebula 定义了 3 类数据资源，使用 onedatastore list 命令可以查看这些资源：system 类型的资源 ID 为 0，存放的是处于运行状态的虚拟机的磁盘映像；default 类型的资源 ID 为 1，用于存放虚拟机镜像；files 类型的资源 ID 为 2，用于存放非镜像类型的普通文件。

```
[oneadmin@front-end-01 ~]$ onedatastore list
  ID NAME                SIZE AVAIL CLUSTER      IMAGES TYPE DS      TM
   0 system              45.1G 85%   -                 0 sys  -       shared
   1 default             45.1G 85%   -                 2 img  fs      shared
   2 files               45.1G 85%   -                 0 fil  fs      ssh
```

默认配置下，system 和 default 类型的资源使用 tm_shared 驱动传输磁盘镜像，就是共享类型。所以，为了使虚拟机运行在宿主机自身的硬盘上，需要将 system 和 default 类型的资源配置为 tm_ssh 驱动传输磁盘镜像。

2. OpenNebula 数据资源传输驱动配置

更改 system 和 default 类型的资源传输驱动，配置为 ssh 类型，使镜像传输时使用 ssh 方式将镜像传输到宿主机上，然后再启动虚拟机。

使用 onedatastore update 0 编辑 id=0，即 system 资源池的属性。执行的时候可能出现类似的报错：Editor not defined。这是因为 OpenNebula 中默认定义的编辑器路径是 /usr/binvi，如果系统 vi 编辑器的路径不是这个路径，就会报错。解决办法就是创建一个软连接，代码如下：

```
[oneadmin@front-end-01 ~]$ onedatastore update 0

Editor not defined
[oneadmin@front-end-01 ~]$ grep EDITOR_PATH /usr/lib/one/ruby/cli/one_helper.rb
    EDITOR_PATH='/usr/bin/vi'
[oneadmin@front-end-01 ~]# ln -s /usr/bin/vim /usr/bin/vi
```

编辑 system 资源池属性，将默认的 share 传输驱动改成 ssh，代码如下：

```
[oneadmin@front-end-01 ~]$ onedatastore update 0
BASE_PATH="/var/lib/one//datastores/"
SHARED="YES"
TM_MAD="ssh"      //原本是 shared 现改为 ssh
```

1107114476@qq.com
dolphinebook.com

```
TYPE="SYSTEM_DS"
```

编辑 default 资源池属性，将默认的 share 传输驱动改成 ssh，代码如下：

```
[oneadmin@front-end-01 ~]$ onedatastore update 1
BASE_PATH="/var/lib/one//datastores/"
CLONE_TARGET="SYSTEM"
DISK_TYPE="FILE"
DS_MAD="fs"
LN_TARGET="NONE"
TM_MAD="ssh"                                    //原本是 shared 现改为 ssh
TYPE="IMAGE_DS"
```

3. OpenNebula 默认本地资源存储路径配置

编辑默认存储资源路径，在 /etc/one/oned.conf 配置文件中将 DATASTORE_LOCATION 配置为宿主机上用于存放虚拟机硬盘的路由，这里使用 /datapool/local_storage，代码如下：

```
[root@front-end-01]# vim /etc/one/oned.conf
DATASTORE_LOCATION   = /datapool/local_storage
```

配置修改完毕，重启 OpenNebula 服务，代码如下：

```
[root@front-end-01 bin]# service opennebula restart
Stopping OpenNebula daemon: oned and scheduler stopped  [  OK  ]
Starting OpenNebula daemon:                             [  OK  ]
```

4. 宿主机对应本地路径配置

在宿主机上创建 /datapool/local_storage 目录，并把目录所属权限制定为 oneadmin，代码如下：

```
[root@ceph-host-02 ]# chown oneadmin.oneadmin /datapool/local_storage
```

5. 创建虚拟机进行验证

配置完成后，使用 onetemplate instantiate "CentOS-6.5-template"--name"vm_run_in_local_storage" 测试创建虚拟机，观察整个创建过程中虚拟机的状态。

```
[oneadmin@front-end-01 ~]$ onetemplate instantiate "CentOS-6.5-template" --name
"vm_run_in_local_storage"
VM ID: 8
[oneadmin@front-end-01 ~]$ onevm list   //刚创建的时候 STAT 是挂起
    ID USER     GROUP    NAME            STAT UCPU    UMEM HOST             TIME
     8 oneadmin oneadmin vm_run_in_local pend    0      0K              0d 00h00
[oneadmin@front-end-01 ~]$ onevm list   //稍后 STAT 是开始创建
    ID USER     GROUP    NAME            STAT UCPU    UMEM HOST             TIME
     8 oneadmin oneadmin vm_run_in_local prol    0      0K 10.168.107   0d 00h00
[oneadmin@front-end-01 ~]$ onevm list   //最后正常状态为运行
    ID USER     GROUP    NAME            STAT UCPU    UMEM HOST             TIME
     8 oneadmin oneadmin vm_run_in_local runn   93    512M 10.168.107   0d 00h00
```

1107114476@qq.com
dolphinebook.com

验证宿主机 /datapool/local_storage 目录中是否有虚拟机硬盘文件，0 表示资源类型 system，8 表示虚拟机的 id（对应创建时的 id）。至此，完成了将虚拟机运行在宿主机本地磁盘的配置。

```
[oneadmin@ceph-host-02]$ ll /datapool/local_storage/0/8

total 1485552
-rw-rw-r-- 1 oneadmin oneadmin        873 Jan 18 17:24 deployment.0
-rw-r--r-- 1 oneadmin oneadmin 1520828416 Jan 18 17:26 disk.0
-rw-r--r-- 1 oneadmin oneadmin     372736 Jan 18 17:24 disk.1
lrwxrwxrwx 1 oneadmin oneadmin         34 Jan 18 17:24 disk.1.iso -> /datapool/
local_storage/0/8/disk.1
```

12.3　OpenNebula 常见故障处理

本节介绍笔者在使用 OpenNebula 过程中遇到的两个故障案例，读者碰到类似的问题可以参考。

1. 虚拟机无法创建

（1）故障描述

创建虚拟机过程中，发现无法创建，虚拟机状态未 fail，查看 /var/log/one/oned.log 日志文件，代码如下：

```
Sat Jan 17 15:27:16 2015 [Z0][VMM][D]: Message received: LOG I 1 Successfully
execute network driver operation: pre. //此处是正常的
Sat Jan 17 15:27:16 2015 [Z0][VMM][D]: Message received: LOG I 1 Command
execution fail: cat << EOT | /var/tmp/one/vmm/kvm/deploy '/var/lib/o
ne//datastores/0/1/deployment.0' '10.168.107.13' 1 10.168.107.13
Sat Jan 17 15:27:16 2015 [Z0][VMM][D]: Message received: LOG I 1 error: Failed
to create domain from /var/lib/one//datastores/0/1/deployment.0
<-- 此处开始报错，提示无法使用 deployment.0 文件创建虚拟机 -->
Sat Jan 17 15:27:16 2015 [Z0][VMM][D]: Message received: LOG I 1 error:
internal error process exited while connecting to monitor: qemu-kvm:
-drive file=/var/lib/one//datastores/0/1/disk.0,if=none,id=drive-ide0-0-
0,format=qcow2,cache=none: could not open disk image /var/lib/one//da
tastores/0/1/disk.0: Permission denied
<-- 此处提示权限问题：访问 disk.0 被拒绝 -->
```

（2）问题分析与处理

从日志告警提示中明显看到一个问题：访问 disk.0 被拒绝。于是登录到 10.168.107.13，在这台宿主机上找到 /var/lib/one//datastores/0/1/deployment.0 文件，手动执行 virsh create deployment.0，得到如下报错信息，这是一个和权限有关的报错。

```
[oneadmin@ceph-host-01 1]$ virsh create deployment.0
error: Failed to create domain from deployment.0
error: Unable to create tap device vnet%d: Operation not permitted
```

1107114476@qq.com
dolphinebook.com

在 Libvirt 配置文件 /etc/libvirt/qemu.conf 中涉及部分权限配置，检查其中的 user、group 的配置，发现不是 oneadmin 的话，要改为 oneadmin：

```
[root@ceph-host-01 qemu]# cat /etc/libvirt/qemu.conf
user  = "oneadmin"
group = "oneadmin"
dynamic_ownership = 0
```

修改完成之后，重启 libvirtd 服务，故障排除。

```
[root@ceph-host-01 qemu]# service libvirtd restart
Stopping libvirtd daemon:                                  [  OK  ]
Starting libvirtd daemon:                                  [  OK  ]
```

2. 计算节点状态异常

（1）故障描述

使用 onehost create 添加宿主机，等待片刻之后，查看宿主机状态为 err。

```
[oneadmin@front-end-01 ~]$ onehost create 10.168.107.14 -i kvm -v kvm -n dummy
[oneadmin@front-end-01 ~]$ onehost list
  ID NAME          CLUSTER   RVM      ALLOCATED_CPU      ALLOCATED_MEM STAT
   9 10.168.107.14  -         0                  -                  - err
```

再查看 /var/log/one/oned.log 日志：

```
Sat Jan 17 17:36:54 2015 [Z0][InM][D]: Monitoring host 10.168.107.14 (9)
Sat Jan 17 17:36:54 2015 [Z0][InM][I]: Command execution fail: scp -r /var/lib/
one/remotes/. 10.168.107.14:/var/tmp/one        //执行命令失败
Sat Jan 17 17:36:54 2015 [Z0][InM][I]: Warning: Permanently added '10.168.107.14'
(RSA) to the list of known hosts.
Sat Jan 17 17:36:54 2015 [Z0][InM][I]: Permission denied, please try again.
Sat Jan 17 17:36:54 2015 [Z0][InM][I]: Permission denied, please try again.
Sat Jan 17 17:36:54 2015 [Z0][InM][I]: Permission denied (publickey,gssapi-
keyex,gssapi-with-mic,password).                //权限拒绝
Sat Jan 17 17:36:54 2015 [Z0][InM][I]: lost connection
Sat Jan 17 17:36:54 2015 [Z0][InM][I]: ExitCode: 1
Sat Jan 17 17:36:54 2015 [Z0][ONE][E]: Error monitoring Host 10.168.107.14 (9):
```

（2）问题分析与处理

从上面的报错信息可以发现，OpenNebula 管理端在连接宿主机时，存在报错权限拒绝的错误。这个问题肯定和权限有关。

OpenNebula 管理端和宿主机之间的通信是通过 ssh 的无密码登录进行的，权限拒绝的问题，最有可能就是无密码登录有问题。

尝试在 OpenNebula 管理端上无密码登录 10.168.107.14，提示输入密码。说明 OpenNebula 管理服务器到宿主机的无密码登录配置有问题。

```
[oneadmin@front-end-01 ~]$ ssh oneadmin@10.168.107.14
Warning: Permanently added '10.168.107.14' (RSA) to the list of known hosts.
```

1107114476@qq.com
dolphinebook.com

```
oneadmin@10.168.107.14's password:
```

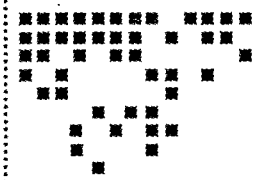
确认无密码登录配置正常之后，问题解决。

12.4　本章小结

本章主要介绍了 OpenNebula 管理平台的基本搭建和配置，使用 OpenNebula 实现对 KVM 虚拟机的管理，同时还介绍了如何使 KVM 虚拟机完全运行在宿主机上，不依赖共享文件系统。不可否认，OpenNebula 非常强大，对于 OpenNebula 还有很多值得研究的地方，鉴于本书的侧重点和篇幅的限制，本章并未对 OpenNebula 高级功能进行介绍。笔者希望本章能起到一个抛砖引玉的作用，读者可以通过 OpenNebula 官网文档库 http: //docs.opennebula.org/4.10/index.html 获取最为全面的帮助文档。

下一章将为读者再介绍另外几款常见的开源管理平台。

1107114476@qq.com
dolphinebook.com



Chapter 13 第 13 章

其他管理平台介绍

第 11 章为读者介绍的 OpenStack 平台是目前最流行的云平台，第 12 章介绍的 OpenNebula 比较适合快速部署私有云。本章将为读者介绍几款在理念和架构上比较有特色的管理平台，以及每种管理平台的适用场景。

13.1 ConVirt 管理平台介绍

ConVirt 支持 KVM、Xen、HyperV 等多种虚拟化引擎，也支持通过 vCenter 的接口管理 VMWare ESXi，同时支持对 OpenStack 的管理，也支持管理亚马逊的云主机。

ConVirt 有如下 3 个版本：

- 社区版：开源版本，可以免费使用。
- 企业版：商业版，主要用于构建企业私有云。
- 企业云版：商业版，主要用于构建公有云。

13.1.1 ConVirt 的理念

ConVirt 的理念是云管理平台，致力于在其他的云管理平台上构建一套标准接口，统一管理私有云、公有云、混合云，以实现如下目标：

- 通过一个管理界面管理多个云环境。
- 通过一个管理界面完成对模板、配置、管理、监控、迁移等整个虚拟机生命周期的管理。
- 通过一个管理界面完成网络、存储、虚拟机、宿主机资源管理。

1107114476@qq.com
dolphinebook.com

- 通过一个管理界面管理一个企业所有的外包公有云。
- 统一自动备份、恢复，并进行高可用、基于策略的负载均衡管理。
- 将公有云上的计算、网络、存储等资源池组成虚拟数据中心，通过虚拟的数据中心实现资源按需使用。

13.1.2　ConVirt 的架构

ConVirt 开源版是标准的三层架构：

- 高交互性的 Web 管理界面。
- 数据存储：存储虚拟机和服务器配置、任务历史记录、性能信息等。
- 管理机：不需要使用代理或者客户端，直接通过管理机进行管理。

ConVirt 开源版主要功能如下：

- 多用户管理。
- 服务器池管理，一个服务器池同时共享同一存储的一组服务器。
- 模板管理，支持第三方模板应用市场，支持精简模式的模板。
- 对虚拟机 24 小时监控、详尽的配置管理。
- 虚拟机资源限制、自动在线迁移。
- 存储监控管理。
- 网络监控管理。

13.2　CloudStack 管理平台介绍

CloudStack 一度是 OpenStack 有力的竞争对手。Cloud Stack 和 OpenStack 最大的区别是，OpenStack 由社区维护，CloudStack 主要由思杰公司维护。CloudStack 开发语言为 Java，此前为 Cloud.com 研发应用的商业软件，后被 Citrix 思杰收购，2012 年 4 月 5 日思杰宣布将 CloudStack 项目提交至 Apache 基金会，之后 CloudStack 成为 Apache 许可下的完全开源软件，已经加入到 Apache 基金会中的开源云计算平台。

CloudStack 使用 Java 开发，是 IaaS 层的解决方案，主要功能如下：

- 支持 XenServer/XCP、KVM、Hyper-V、VMware ESXi（需要有 vSphere vCenter）。
- 提供友好的 Web 管理界面。
- 提供 API。
- 提供亚马逊 S3/EC2 兼容 API。
- 通过主存储管理虚拟机镜像，通过第二存储管理模板、快照、ISO 镜像。
- 从二层到七层多种的网络服务支持，如 DHCP、NAT、防火墙、VPN，等等。
- 支持网络、计算、存储资源审计。
- 支持多租户隔离。
- 支持用户管理。

1107114476@qq.com
dolphinebook.com

1. CloudStack 理念

CloudStack 采用了“框架＋插件”的系统构架，通过不同的插件来提供对不同虚拟化技术的支持。对于标准的 Xen/KVM 计算节点，CloudStack 需要在计算节点上安装 Agent 与控制节点进行交互；对于 XenServer/VMWare 计算节点，CloudStack 通过 XenServer/VMWare 所提供的 XML-RPC 远程调用接口与计算节点进行交互。

CloudStack 本身是一个虚拟化管理平台，但是它通过 CloudBridge 提供了与 Amazon EC2 相兼容的云管理接口，对外提供 IaaS 服务。

2. CloudStack 架构

CloudStack 对于云有独特的理解，有主存储、第二存储、集群、Pods、Zone 等概念。

（1）主存储

主存储用来保存虚拟机镜像，支持本地磁盘、NFS、iSCSI、FC、Ceph、GlusterFS 等方式。主存储的特性如下：

- 为虚拟机提供镜像存储。
- 高 IOPS。
- 同集群都可以访问。
- 支持分层存储。

（2）第二存储

第二存储提供虚拟机模板、ISO 的存放，主要特点是一次写入，多次读取，只支持 NFS 的方式。

（3）集群

如图 13-1 所示，集群是共享同一个主存储的一批计算节点，虚拟机镜像存储在主存储上，可以在一个集群内部迁移。

（4）Pods

如图 13-2 所示，Pods 由一个或多个集群组成，通常是同一个 IP 子网内的集群。

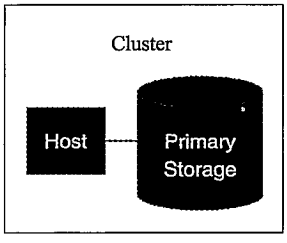


图 13-1 CloudStack Cluster 概念

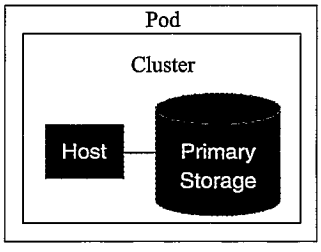


图 13-2 CloudStack Pod 概念

（5）区域

如图 13-3 所示，区域（Zone）通常是物理上的一个数据中心，包含一个或者多个 Pod。一个 Zone 通常共享一个二级存储。

1107114476@qq.com
dolphinebook.com

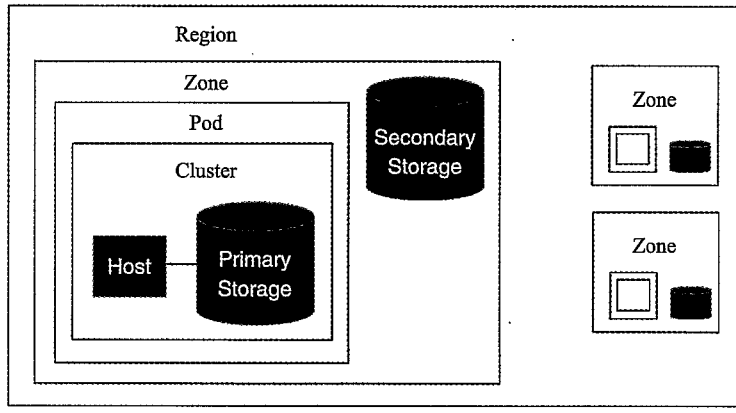


图 13-3　CloudStack Zone 概念

13.3　oVirt 管理平台介绍

oVirt（open Virtualization）是红帽公司商业的云管理平台（Red Hat Enterprise Virtualization，RHEV）的社区版本，主要使用场景是私有云管理，在架构设计上使用了节点 / 引擎（Node/Engine）架构，特点是专注 KVM、基于 Web 界面、易于搭建。一些 KVM 上的新技术，如 spice、GlusterFS 等都首先在 oVirt 上得到支持。

特别要提到的一点是，oVirt 有一个中文社区（http: //ovirt-china.org），有非常详细的中文资料及邮件列表、IRC、wiki、blog 等沟通工具。

1. oVirt 理念

oVirt 的理念是数据中心虚拟化管理平台解决方案，目标是方便、快速地构建私有云。

oVirt 的特性如下：

- 对宿主机节点的管理：支持基于数据中心、集群的管理，比较有特色的一个功能是支持宿主机故障时，通过 HP ILO、Dell IDRAC、IPMI 等硬件管理卡强制关闭宿主机，以保证集群可靠性。
- 对虚拟机的管理：可以完成虚拟机的创建、快照、删除、基于模板的克隆等常见操作。
- 迁移：在线及离线迁移虚拟机。
- 高可用：当一台宿主机宕机时，在另外一台宿主机上自动开启虚拟机。
- 宿主机、虚拟机性能查看及统计。
- 支持 NFS、iSCSI、FC、GlusterFS 等存储方式。
- 支持多端口的网口绑定。
- 支持命令行工具，可以完成图形界面的大部分功能。
- 支持活动目录。
- 提供 SDK，支持多种开发语音。
- 开源免费。

1107114476@qq.com
dolphinebook.com

2. oVirt 架构

oVirt 由客户端和管理端组成，客户端即 oVirt-Node，类似于 VMWare ESXi，最早由 Fedaro 定制而成，如图 13-4 所示。现在最新的 3.5 版本已经使用 RHEL6 定制，也说明 oVirt 已经可以在生产环境使用。oVirt 也支持在 Linux 系统上安装 VDSM 服务而得到一个 oVirt 客户端，客户端还有 Live 版本，支持从光盘直接启动。管理端 oVirt-Engine 类似基于 Web 页面的 VMWare vCenter。

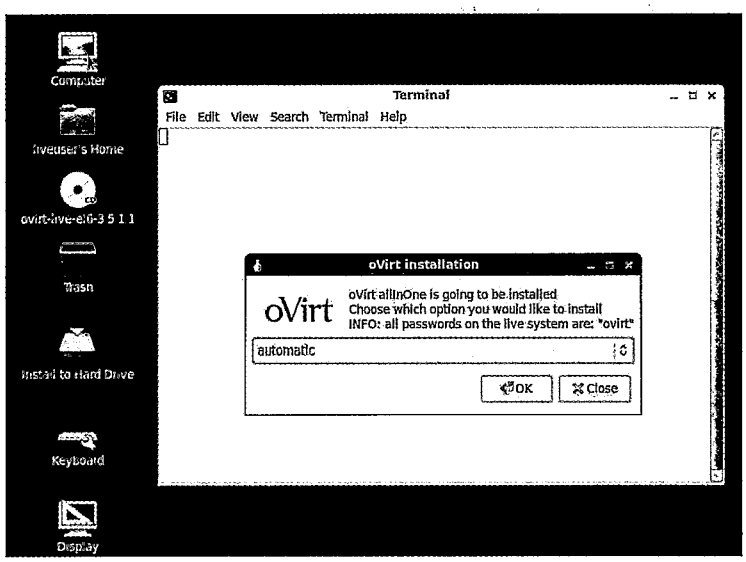


图 13-4 oVirt live 版界面

如图 13-5 所示，oVirt 按照数据中心来划分资源，每个数据中心都有整体的存储、网络、模板资源。计算资源按照集群划分，每个集群从数据中心获取存储、网络、模板资源。

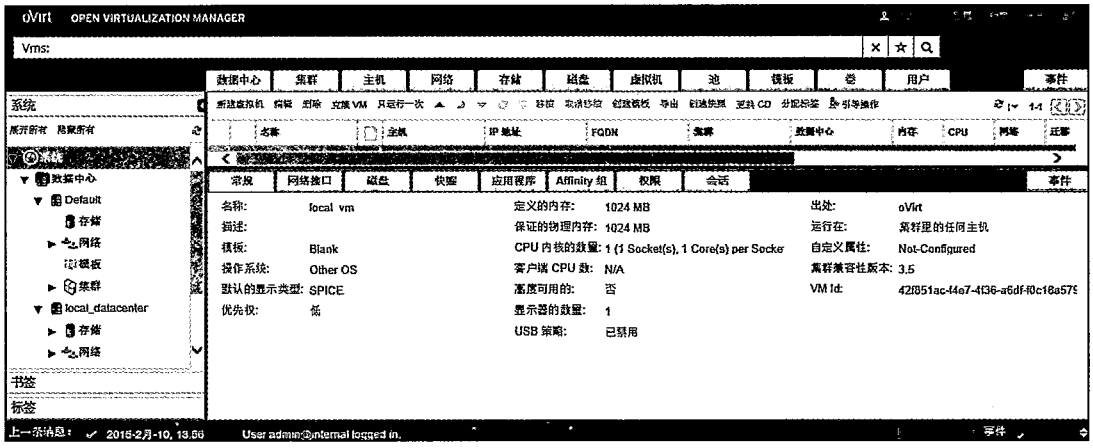


图 13-5 oVirt 数据中心管理界面

1107114476@qq.com
dolphinebook.com

如图 13-6 所示，oVirt 对宿主机管理非常详细，软件和硬件信息都很齐全，甚至可以通过 IPIM 控制宿主机的电源。

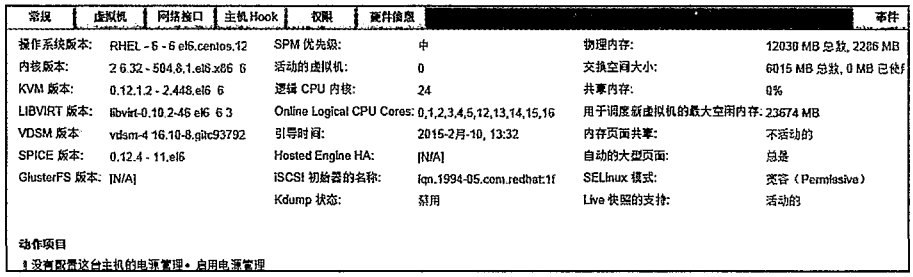


图 13-6　oVirt 宿主机管理界面

如图 13-7 所示，oVirt 对虚拟机创建是向导式的，也可以将虚拟机转换成模板，通过模块快速复制。

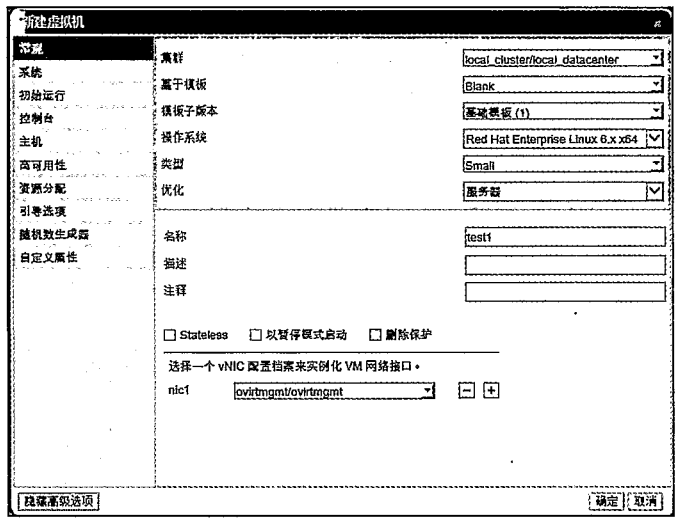


图 13-7　oVirt 虚拟机创建界面

13.4　WebVirtMgr 管理平台介绍

WebVirtMgr 是近几年来发展较快、比较活跃、非常清新的一个 KVM 管理平台。一看名字就知道，其定位非常清晰，就是要将 VirtManager 页面化。

1. WebVirtMgr 的主要功能

WebVirtMgr 几乎是纯 Python 开发的，前端是基于 Python 的 Django，后端是基于 Libvirt 的 Python 接口。风格也是 Python 的风格，特点如下：

1107114476@qq.com
dolphinebook.com

- 容易使用。
- 基于 libvirt 的连接。
- 虚拟机生命周期管理。

在宿主机管理方面支持以下功能：

- CPU 利用率。
- 内存利用率。
- 网络资源池管理。
- 存储资源池管理。
- 虚拟机镜像。
- 虚拟机克隆。
- 快照管理。
- 日志管理。

在虚拟机管理方面支持以下功能：

- CPU 利用率。
- 内存利用率。
- 光盘管理。
- 关闭虚拟机。
- 安装虚拟机。
- VNC 网页连接。
- 创建快照。

2. WebVirtMgr 的搭建方法介绍

WebVirtMgr 搭建非常简单，主要步骤如下：

（1）配置解析主机名

修改所有节点的 /etc/hosts 文件，使所有节点之间能够通过 DNS 解析主机名。

（2）客户端安装

1）安装 KVM 和 Libvirt 组件（略）。

2）Libvirtd 服务监听配置。修改 /etc/sysconfig/libvirtd 文件，去掉下面一行注释，使 Libvirtd 服务处于监听状态：

```
vim /etc/sysconfig/libvirtd
LIBVIRTD_ARGS="--listen"
```

3）配置 Libvirt 服务。配置 Libvirt 服务，允许通过 TCP 方式通信，修改 /etc/libvirt/libvirtd.conf：

```
# 允许 tcp 监听
listen_tcp = 1
# 开放 tcp 端口
tcp_port = "16509"
```

1107114476@qq.com
dolphinebook.com

```
# 监听地址修改为 0.0.0.0
listen_addr = "0.0.0.0"
# 配置 tcp 通过 sasl 认证
auth_tcp = sasl
```

启动服务：

```
service libvirtd start
```

4）创建 libvirt 管理用户。

```
saslpasswd2 -a libvirt virtadmin
```

可以使用 sasldblistuser2 命令查看创建了哪些用户：

```
sasldblistusers2 -f /etc/libvirt/passwd.db
virtadmin@webvirt: userPassword
```

如果需要禁止用户，使用如下命令：

```
saslpasswd2 -a libvirt -d virtadmin
```

（3）服务器端安装

1）安装基础组件。

```
rpm -ivh http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
yum -y install git python-pip libvirt-python libxml2-python python-websockify
supervisor nginx
```

2）WebVirtMgr 安装及初始化配置。通过 git 克隆 WebVirtMgr 代码，并安装需要的 Python 组件：

```
git clone git://github.com/retspen/webvirtmgr.git
cd webvirtmgr
pip install -r requirements.txt # or python-pip (RedHat, Fedora, CentOS, OpenSuse)
```

初始化数据库：

```
./manage.py syncdb
./manage.py collectstatic
```

添加超级管理员，运行如下命令，并按照提示添加管理员账号。

```
./manage.py createsuperuser
```

将 WebVirtMgr 复制到系统默认的 Web 服务目录：

```
cp -r webvirtmgr/ /var/www/
```

3）配置 nginx。添加 nginx 配置文件，内容如下：

```
vim /etc/nginx/conf.d/webvirtmgr.conf
```

1107114476@qq.com
dolphinebook.com

```
server {
    listen 80 default_server;
    server_name $hostname;
    #access_log /var/log/nginx/webvirtmgr_access_log;
   location /static/ {
        root /var/www/webvirtmgr/webvirtmgr; # or /srv instead of /var
        expires max;
    }
    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-for $proxy_add_x_forwarded_for;
        proxy_set_header Host $host:$server_port;
        proxy_set_header X-Forwarded-Proto $remote_addr;
        proxy_connect_timeout 600;
        proxy_read_timeout 600;
        proxy_send_timeout 600;
        client_max_body_size 1024M; # Set higher depending on your needs
    }
}
```

修改 nginx 配置文件 /etc/nginx/nginx.conf，将 default.conf 行注释掉：

```
# The default server is in conf.d/default.conf
```

修改 default.conf 文件名：

```
 mv default.conf  default.conf.bak
```

启动 nginx 服务：

```
/etc/init.d/nginx restart
chkconfig supervisord on
```

设置 WebVirtMgr 文件夹 nginx 用户权限：

```
chown -R nginx:nginx /var/www/webvirtmgr
```

4）配置超级引擎。编辑超级引擎，增加以下内容：

```
/etc/supervisord.conf
[program:webvirtmgr]
command=/usr/bin/python /var/www/webvirtmgr/manage.py run_gunicorn -c /var/www/
webvirtmgr/conf/gunicorn.conf.py
directory=/var/www/webvirtmgr
autostart=true
autorestart=true
logfile=/var/log/supervisor/webvirtmgr.log
log_stderr=true
user=nginx
[program:webvirtmgri-console]
command=/usr/bin/python /var/www/webvirtmgr/console/webvirtmgr-console
```

1107114476@qq.com
dolphinebook.com

```
directory=/var/www/webvirtmgr
autostart=true
autorestart=true
stdout_logfile=/var/log/supervisor/webvirtmgr-console.log
redirect_stderr=true
user=nginx
```

配置完成，可以通过浏览器访问测试了。

（4）WebVirtMgr 升级

因为 WebVirtMgr 处于快速发展期，所以，经常有新版本发布，需要升级，具体如下：

```
cd /var/www/webvirtmgr
git pull
./manage.py collectstatic
service supervisord restart
```

（5）排错方法

如果遇到问题，需要排错，可以以 debug 模式：

```
./manage.py runserver 0:8000
```

然后用浏览器访问服务器的 8000 端口：

```
http://x.x.x.x:8000
```

在服务端有非常详细的输出信息。

13.5　如何选择管理平台

随着虚拟机的数量增加，面临的一个问题就是如何管理虚拟机。这时往往希望有一款管理工具，能够完成以下任务：

- 能快速创建一台或者一批虚拟机。
- 能完成虚拟机的创建、维护、销毁整个生命周期的管理。
- 能够和其他的运维系统交互，为其他的运维系统提供虚拟化服务，提高整体的自动化运维水平。

运维工程师从对 KVM 的熟悉到对虚拟机的管理，必然经过以下过程：

- 管理单台虚拟机。
- 通过 Virt-Manager 或者 Libvirt 管理多台宿主机。
- 通过脚本批量的创建、管理虚拟机、宿主机。
- 通过开源或者自研的云管理平台完成虚拟机管理。
- 整合云管理平台和其他运维系统。

虚拟机数量少的时候，往往是通过 virsh 或者 Virt-Manager 管理。virsh、Virt-Manager 都

1107114476@qq.com
dolphinebook.com

是手动管理方式，所以，一次只能管理一台虚拟机，要批量管理就需要写脚本，或者和自动化的管理工具结合（比如 Puppet）。

虚拟机数量再增加，或者频繁变更，或者希望把虚拟机的申请、维护以服务的方式提供给使用方，这时候就需要管理平台。

从第 11 章到 13 章，为读者介绍了主流的开源管理平台，那么在实际的生产环境中，如何选择管理平台呢？管理平台的选择，主要取决于以下几个因素：

- 宿主机、虚拟机规模。
- 稳定性。
- 易用性。
- 使用者的喜好。

表 13-1 所示为对几款管理平台的比较。

表 13-1 常见 KVM 管理平台简单比较

虚拟机	稳定性（满分 5 分）	易用性（满分 5 分）	支持虚拟机规模	社区活跃性（满分 5 分）
OpenStack	4	2	10 000+	5
CloudStack	4.8	4	10 000+	3.5
OpenNebula	4.6	4	数千	3.8
oVirt	4.7	4	数千	4
Convirt	4.5	4	数千	3
WebVritMgr	3	4.5	数百	3

笔者对管理平台选择的建议如下：

- 规模比较小，熟悉 Python，希望做一些定制，建议选择 WebVirtMgr。
- 规模中等，完全是企业内部的私有云，建议使用 OpenNebula、CloudStack、ConVirt 开源版、oVirt 这样的管理平台。
- 规模非常大，有足够的运维和开发，建议基于 OpenStack 定制。

13.6 本章小结

本章介绍了 ConVirt、CloudStack、oVirt 管理平台的理念、架构，目的是让读者对这些相对小众化（主要是和 OpenStack 对比）的开源平台有初步的了解。另外，介绍了一个非常清新的管理平台 WebVirtMgr，简单易用，适用于规模比较小的场景。

下一篇将介绍虚拟化项目实施，是本书的重点内容之一。

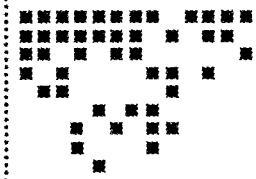
1107114476@qq.com
dolphinebook.com

第四篇 Part 4

KVM 虚拟化项目实施

- 第 14 章　业务性能评估与压力模型建立
- 第 15 章　宿主机选型与基础性能测试
- 第 16 章　虚拟机镜像制作、配置与测试
- 第 17 章　单机虚拟化技术与生产环境实践
- 第 18 章　KVM 虚拟化集群技术与应用场景
- 第 19 章　业务迁移到虚拟化环境流程
- 第 20 章　宿主机自动化运维管理
- 第 21 章　虚拟化监控、报警与应急响应方案

1107114476@qq.com
dolphinebook.com



Chapter 14 第 14 章

业务性能评估与压力模型建立

从本章开始，将正式进入第四篇——KVM 虚拟化项目实施。作为开篇的第一章，将从虚拟化项目实施的最开始阶段，项目接入这块来介绍如何评估一个业务是否适合接入虚拟化，进行必要的性能数据采集和分析，建立适当的压力模型，最终决定一个业务是否能接入虚拟化。

虽然虚拟化技术在进过了这么多年的发展之后的今天，无论从性能、稳定性、可靠性方面都得到了业界的肯定，但是不可否认，虚拟化技术不是万能的，并不是所有的业务都可以接入虚拟化。所以，在接入之前，都必须经过非常详细的性能评估，才能保证业务后续在虚拟化环境中流畅运行。

性能分析的目的，是要把一些不适合使用虚拟化的业务排除出去，避免因为盲目接入而导致业务受影响。本章围绕业务性能采集、评估、性能分析、压力模型建立等方面进行详细的介绍。

14.1 Linux 系统性能数据采集原则与方法

对于业务服务器性能的采集，必须遵循两点：第一，性能数据采集的方法尽可能简单、通用，尽量避免在业务服务器内安装程序；第二，操作步骤和方法必须简练，因为大多数情况下，性能采集的工作需要业务方来完成，过于复杂的操作，很容易造成性能数据无法采集。本节将介绍 Linux 系统下，如何采集 Linux 性能数据。

1. Linux 系统性能采集原则

首先，对于 Linux 系统性能采集，其结果必须包含 4 个部分的数据：① CPU 性能；②磁

1107114476@qq.com
dolphinebook.com

盘 I/O 性能；③网络发包率及吞吐性能；④内存及 swap 使用情况。

其次，采集过程要尽可能对系统本身无压力，所以，最好使用 Linux 系统自带的命令。

最后，操作要尽可能简单，一般来说，在 Linux 系统下，使用 Shell 脚本是一个不错的选择。因为简单的 Shell 脚本兼容性非常好，Linux 都带 Shell 的运行环境。

2. 常用的 Linux 系统性能查看命令介绍

Linux 系统性能查看的命令有很多，而且每个命令都很强大，在命令后面带不同的参数，可以查看到不同项目的性能，下面介绍几个常用的查看服务器性能的命令，这些命令采集到的数据，可以作为压力模型建立的依据。

（1）CPU 性能采集

CPU 性能采集命令：`sar -P ALL 2 10`

命令含义：每隔 2 秒显示采集一次，一共采集 10 次 CPU 所有核的性能。其中，ALL 可以用任意和核心 ID 来替换，以获取这个核的性能信息。输出的最后有一个汇总的平均性能信息。

- %user：显示在用户级别应用运行使用 CPU 总时间的百分比。
- %nice：显示在用户级别，用于 nice 操作，所占用 CPU 总时间的百分比。
- %system：在核心级别（Kernel）运行所使用 CPU 总时间的百分比。
- %iowait：显示用于等待 I/O 操作占用 CPU 总时间的百分比。
- %steal：管理程序（Hypervisor）为另一个虚拟进程提供服务而等待虚拟 CPU 的百分比。
- %idle：显示 CPU 空闲时间占用 CPU 总时间的百分比。

```
[root@localhost ~]# sar -P ALL 2 10
<-- 此处省略 10 次 cpu 性能输出 -->
Average:        CPU     %user     %nice   %system   %iowait    %steal     %idle
Average:        all      8.96      0.00      1.52      0.06      0.00     89.45
Average:          0     24.88      0.00      2.35      0.15      0.00     72.62
Average:          1     17.59      0.00      2.30      0.13      0.00     79.97
Average:          2     15.65      0.00      2.04      0.15      0.00     82.16
Average:          3     13.17      0.00      1.87      0.14      0.00     84.83
Average:          4     10.32      0.00      2.22      0.05      0.00     87.42
Average:          5      8.97      0.00      2.05      0.15      0.00     88.84
Average:          6     22.36      0.00      2.29      0.28      0.00     75.08
Average:          7     12.75      0.00      1.85      0.08      0.00     85.32
```

（2）网卡性能采集

网卡性能采集命令：`sar -n DEV 2 5`

命令含义：每隔 2 秒一次，采集 5 次所有网卡的性能数据。

- rxpck/s：每秒接收的数据包。
- txpck/s：每秒发送的数据包。
- rxbyt/s：每秒接收的字节数。

1107114476@qq.com
dolphinebook.com

- txbyt/s：每秒发送的字节数。
- rxcmp/s：每秒接收的压缩数据包。
- txcmp/s：每秒发送的压缩数据包。
- rxmcst/s：每秒接收的多播数据包。

```
[root@localhost ~]# sar -n DEV 2 10
<-- 此处省略 10 次网卡性能输出 -->
Average:   IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
Average:   lo       1.07      1.07      0.23      0.23      0.00      0.00      0.00
Average:   eth0    315.81    324.07     37.90     39.54      0.00      0.00      0.31
Average:   eth1    0.31      0.00       0.11      0.00      0.00      0.00      0.15
```

（3）磁盘性能采集

磁盘性能采集命令：`iostat -x -d -k 2 10`

各命令的含义如下。

- -x 选项：用于显示和 I/O 相关的扩展数据。
- -d 选项：显示设备（磁盘）使用状态。
- -k 选项：表示将 Block 为单位的列使用 KB 为单位。
- 2：表示数据采集每隔 2 秒一次；10：表示采集 10 次。
- rrqm/s：每秒这个设备相关的读取请求有多少被 Merge（合并）了（当系统调用需要读取数据时，虚拟文件系统将请求发到各个文件系统，如果文件系统发现不同的读取请求读取的是相同块的数据，文件系统会将这个请求合并）。
- wrqm/s：每秒这个设备相关的写入请求有多少被合并了。
- r/s：每秒读请求。
- w/s：每秒写请求。
- rKB/s：每秒从设备读入的数据量。
- wKB/s：每秒向设备写入的数据量。
- avgrq-sz：平均请求扇区的大小 avgqu-sz 是平均请求队列的长度。
- await：每一个 I/O 请求的处理的平均时间（单位是毫秒）。
- svctm：表示平均每次设备 I/O 操作的服务时间（以毫秒为单位）。
- %util：在统计时间内所有处理 I/O 时间，除以总共统计时间。

```
 [root@gcloud-bjgm-c1-132 ~]# iostat -xdk 2 10
Linux 2.6.32-431.29.2.el6.x86_64 (gcloud-bjgm-c1-132)   01/27/2015      _
x86_64_         (40 CPU)
<-- 此处省略 9 次磁盘性能输出 -->
Device:  rrqm/s  wrqm/s  r/s  w/s  rkB/s  wkB/s  avgrq-sz  avgqu-sz  await
svctm  %util
sda       0.00     0.00    0.00    0.00     0.01     0.00     8.01     0.00
0.26   0.26   0.00
sdb       0.00     1.82    3.04   27.99    85.11   530.57    39.69     0.08
```

1107114476@qq.com
dolphinebook.com

```
2.54    0.30    0.92
dm-0      0.00      0.00      0.00      0.00      0.00      0.00      8.00      0.00
1.10    1.09    0.00
dm-1      0.00      0.00      0.01      3.37      0.13     13.75      8.22      0.01
2.29    0.14    0.05
```

（4）内存性能采集

内存性能采集命令：`sar -r 2 10`

命令含义：-r 输出内存和交换空间的统计信息，2 表示采集频率，10 表示采集次数。

下面列举 3 次输出的结果并说明各项的意义。

- kbmemfree：这项值和 free 命令中的 free 值基本一致，它不包括 buffer 和 cache 的空间。
- kbmemused：这项值和 free 命令中的 used 值基本一致，所以，它包括 buffer 和 cache 的空间。
- %memused：这个值是 kbmemused 和内存总量（不包括 swap）的一个百分比。
- kbbuffers 和 kbcached：这两个值就是 free 命令中的 buffer 和 cache。
- kbcommit：保证当前系统所需要的内存，即为了确保不溢出而需要的内存（RAM+swap）。
- %commit：这个值是 kbcommit 与内存总量（包括 swap）的一个百分比。

```
[root@gcloud-bjgm-c1-132 ~]# sar -r 2 10
Linux 2.6.32-431.29.2.el6.x86_64 (gcloud-bjgm-c1-132)    01/27/2015
x86_64_         (40 CPU)
08:17:33 PM kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit
08:17:35 PM  52405092 212088412     80.19    192340 124431852 107663704     39.45
08:17:37 PM  52405164 212088340     80.19    192340 124431852 107663704     39.45
08:17:39 PM  52405180 212088324     80.19    192340 124431852 107673948     39.46
```

3. 性能采集脚本示例

上一节介绍的 4 个命令，分别用于采集 CPU、网卡、磁盘、内存 4 个方面的性能数据，其实还有很多类似的命令可以用于采集这些信息，只是笔者感觉这 4 个命令好用一点，所以，举了这几个命令。本节主要展示一个在 Linux 系统中用于采集服务器性能数据的脚本，一般情况下，在服务器负载最高的时间点，通过定义 crontab 任务，运行此脚本可以采集该时段的服务器负载，脚本内容如下：

```
#!/bin/bash
#Default frequency
# $TIMES x $INTERVAL = performance collection long
#eg:
#  TIMES=180 , INTERVAL=10  collection long is 0.5h
TIMES=3600          //此处定义采集的次数
INTERVAL=2          //此处定义采集频率
PWD=`pwd`
TIME=`date "+%F %H:%M:%S"`
TAR=`whereis tar|awk -F ":" '{print $2}'|awk '{print $1}'`
SAR=`whereis sar|awk -F ":" '{print $2}'|awk '{print $1}'`
```

1107114476@qq.com
dolphinebook.com

```
IOSTAT=`whereis iostat|awk -F ":" '{print $2}'|awk '{print $1}'`

SysInfo(){  //此函数内容，用于采集服务器的一些基本信息，诸如硬盘大小、分区、网卡信息等
        echo "sysip : $SYSIP"|tee $PWD/$SYSIP/sysinfo
        echo "starttime : $TIME" |tee -a $PWD/$SYSIP/sysinfo
        /sbin/ifconfig >>$PWD/$SYSIP/sysinfo
        echo "====================================" >>$PWD/$SYSIP/sysinfo
        /usr/sbin/dmidecode >>$PWD/$SYSIP/sysinfo
        echo "====================================" >>$PWD/$SYSIP/sysinfo
        /bin/cat /proc/cpuinfo >> $PWD/$SYSIP/sysinfo
        echo "====================================" >>$PWD/$SYSIP/sysinfo
        /sbin/fdisk -l >> $PWD/$SYSIP/sysinfo
        echo "====================================" >>$PWD/$SYSIP/sysinfo
        /bin/df -Th >>$PWD/$SYSIP/sysinfo
        echo "====================================" >>$PWD/$SYSIP/sysinfo
        /usr/bin/free -m >> $PWD/$SYSIP/sysinfo
        echo "====================================" >>$PWD/$SYSIP/sysinfo
        echo ""
}
CheckEnv(){
//此函数用于检查服务器环境获取服务器IP，判断性能采集工具是否安装，如果未安装会自动安装
        PUB_IP=`/sbin/ifconfig |grep "inet addr" | awk -F: '{print $2}'| awk
        '{print $1}'|grep -v "172\.\|10\.\|127\.\|192\."|sed -n 1p`
        PRI_IP=`/sbin/ifconfig |grep "inet addr" | awk -F: '{print $2}'| awk
        '{print $1}'|grep "10\.\|127\.\|192\."|sed -n 1p`
        if [ " $PUB_IP" == "" ];then
                SYSIP=$PRI_IP
        else
                SYSIP=$PUB_IP
        fi
        if [ -d $PWD/$SYSIP ];then
                rm -rf $PWD/$SYSIP
        fi
        mkdir -p $PWD/$SYSIP
        if ! grep iostat /usr/bin/iostat ;then
        <-- 检查系统中是否有iostat命令，如果没有，则安装sysstat包 -- >
        yum -y install sysstat
        fi
}
GetPerf(){ //此函数是采集性能的关键，用了4个命令来采集服务器性能
        CPUUSAGE="$PWD/$SYSIP/cpuusage.log"
        MEMUSAGE="$PWD/$SYSIP/memusage.log"
        DISKUSAGE="$PWD/$SYSIP/diskusage.log"
        NETWORK="$PWD/$SYSIP/network.log"
        $SAR -P ALL $INTERVAL $TIMES>> $CPUUSAGE &
        $IOSTAT -dkx $INTERVAL $TIMES>> $DISKUSAGE &
        $SAR -n DEV $INTERVAL $TIMES>> $NETWORK &
        $SAR -r $INTERVAL $TIMES>> $MEMUSAGE &
        for ((i=0;i<$TIMES;i++)) //这个循环，定义等待时间，也就是采集时长
        do
                sleep $INTERVAL
```

1107114476@qq.com
dolphinebook.com

```
        done
}
CheckEnv
SysInfo
GetPerf
if [ -d $PWD/$SYSIP ];then //打包采集结果。
        cd $PWD
        rm -f $SYSIP.tar.gz
        tar zcvf $SYSIP.tar.gz $SYSIP
fi
curl -A "SNDA-PREF-COLLECTOR" -F "result=@$SYSIP.tar.gz" "192.168.25.100:37011/
recv-pref.php?os=linux" //自动上传采集到的结果
```

从脚本功能上看，主要做了几件事：

第一，获取服务器基本信息；

第二，检查服务器环境，确保这几个采集性能的命令能运行；

第三，采集服务器性能；

第四，打包上传性能数据。

提示　脚本最后一行的作用是将采集到的结果上传到性能分析的服务器，如果只是采集性能数据，这一行可以省略。

14.2　Linux 系统压力模型建立

压力模型的建立，可以很直观地反映整个系统的负载情况，压力模型中主要包含 4 个主要元素：CPU 负载、磁盘负载、内存占用、网络负载。下面就将从每个元素来分析，最终可以得到一个系统最直观的负载情况。

1. 分析性能采集数据

14.1.3 节中介绍了 Linux 性能数据采集脚本，使用该脚本完成性能采集之后，会得到 4 个元素的性能数据文件——CPU 性能数据：cpuusage.log；磁盘性能数据：diskusage.log；网卡性能数据：network.log；内存性能数据：memusage.log。每个性能数据文件的格式，在 14.1.2 节中已经详细介绍过了，这里不再重复，只说明每个元素中，需要重点关注的项。

cpuusage.log 文件中采集了以下几项参数：

```
%user     %nice    %system    %iowait    %steal     %idle
```

一般情况下根据这些项的值可以得到一些信息。例如，如果 %iowait 的值过高，则可能硬盘存在 I/O 瓶颈；如果 %idle 的值高但系统响应慢时，则可能 CPU 等待分配内存，可以尝试添加内存；如果 %idle 的值持续低于 10，则系统的 CPU 处理能力相对较低，说明系统中最需要解决的资源是 CPU。

1107114476@qq.com
dolphinebook.com

diskusage.log 文件中采集了以下几项参数：

```
rrqm/s  wrqm/s  r/s  w/s  rkB/s  wkB/s  avgrq-sz  avgqu-sz  await svctm  %util
```

r/s 和 w/s 就是通常说的磁盘 IOPS，包括读 IOPS 和写 IOPS，是衡量磁盘性能和压力非常重要的指标。

rkB/s 和 wkB/s 表示磁盘的吞吐，每秒能写或者能读取多少数据，对于顺序读 / 写的应用，这项参数非常重要。

await 和 svctm 是一对相对的数据，await 可以理解为 I/O 的处理时间，包括队列时间和操作时间，一般系统 I/O 的处理时间应该低于 5ms，一旦超过 20ms，服务器会感觉到卡。syctm 表示设备 I/O 操作的服务时间，一般情况下，await 大于 svctm，它们的差值越小，则说明队列时间越短，性能越好。

%util 项是表示磁盘利用率，值越小磁盘越空闲，当这个值持续大于 90% 时，说明磁盘利用率很高，需要引起重视。

network.log 文件中采集了以下几项参数：

```
rxpck/s  txpck/s  rxkB/s  txkB/s  rxcmp/s  txcmp/s  rxmcst/s
```

rxpck/s 和 txpck/s 分别表示每秒发送和接收数据包的多少，这是衡量一个网卡性能的重要指标，如果这个值大于 10 万，说明网卡压力很大。rxkB/s 和 txkB/s 表示网卡的吞吐性能。一般换算一下，如果接近千兆或者万兆的上限，说明服务器存在网卡吞吐压力。

memusage.log 文件中采集了以下几项参数：

```
kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit
```

通过简单的 memused-kbbuffers-kbcached 得到大致的内存实际使用量，还有一个需要注意的项是 %commit，这是 kbcommit 与内存总量（包括 swap）的一个百分比，这个值越小说明服务器内存越充足，在接入虚拟化时，可以适当减少内存分配。

2. gnuplot 介绍

gnuplot 是一个能通过命令行的交互，实现绘图的工具，功能非常强大，可以画出复杂的三维图形，使用方法比较简单，只需要使用 gnuplot 命令带上简单的参数就能画出图形。

在 Ubuntu 系统中，可以直接使用 sudo apt-get install gnuplot 安装，CentOS 系统可以下载源码包编译安装，使用 ./configure; make; make install; 即可。简单举例说明一下。

```
/usr/local/bin/gnuplot --persist <<EOF

set term png size 800,600                    //定义输出图像大小及格式
set output "DiskIOPSPerformance.png"         //输出文件名称
set key title "Disk IOPS"                    //图像标题
set key box 3                                //设置图像外边框
set key below
<-- 设置图注位置，可选（lmargin/rmargin/tmargin/bmargin(below)）-->
```

1107114476@qq.com
dolphinebook.com

```
set ylabel "IOPS"                                    //定义 y 轴单位
plot                                                 //绘图内容区域
"sda_iops.txt" using  4 w l title "sda read IOPS", "sda_iops.txt" using  5 w l
title "sda write IOPS"
EOF
```

绘图区域各配置参数的含义如下：

- sda_iops.txt：是画图的数据文件，每个数据之间以空格或者 tab 分割，文件的内容如下。

```
sda   0.00   549.00   1.00   235.00   42.00   2982.00   25.63   0.07   0.31   0.07   1.65
```

- using 4：表示使用第四列的数据绘图。
- w l=with line：表示绘图类型是曲线，如果定义 w p，则表示话散点图。
- title "sda read IOPS"：表示图注的名称。

在一张图上可以绘制多个项，上述实例绘制了两个项：sda read IOPS 和 sda write IOPS，如果绘制多个，相互之间用逗号分隔。

3. 利用 gnuplot 展示系统压力

将采集到的服务器性能数据通过脚本进行处理，将处理结果交给 gnuplot 进行展示，通过图表的形式，可以直观地看到业务服务器的压力情况，由于脚本冗长，所以将脚本分享在笔者的博客上，并生成了二维码，链接地址和二维码如下：

链接地址：http: //xiaoli110.blog.51cto.com/1724/1656450。

二维码：

通过分析脚本和 gnulot 配合生成的磁盘 IOPS 性能如图 14-1 所示。

图 14-1 是 14.2.2 节中所举 gnuplot 实例画出来的曲线图。从曲线图上可以比较直观地看到，写的 IOPS 比较高，最大在 6200 左右。

4. Linux 压力模型建立

建立 Linux 压力模型的数据基础是 14.1.3 节性展示的采集脚本取到的性能数据，主要以 CPU、网卡、磁盘三个压力元素作为对象，进行汇总和统计。

- CPU 元素：主要参考参数是 CPU %idle（空闲时间），包括整体和单核和 CPU 核数。
- 网卡元素：主要参考参数是吞吐量和发包率。

1107114476@qq.com
dolphinebook.com

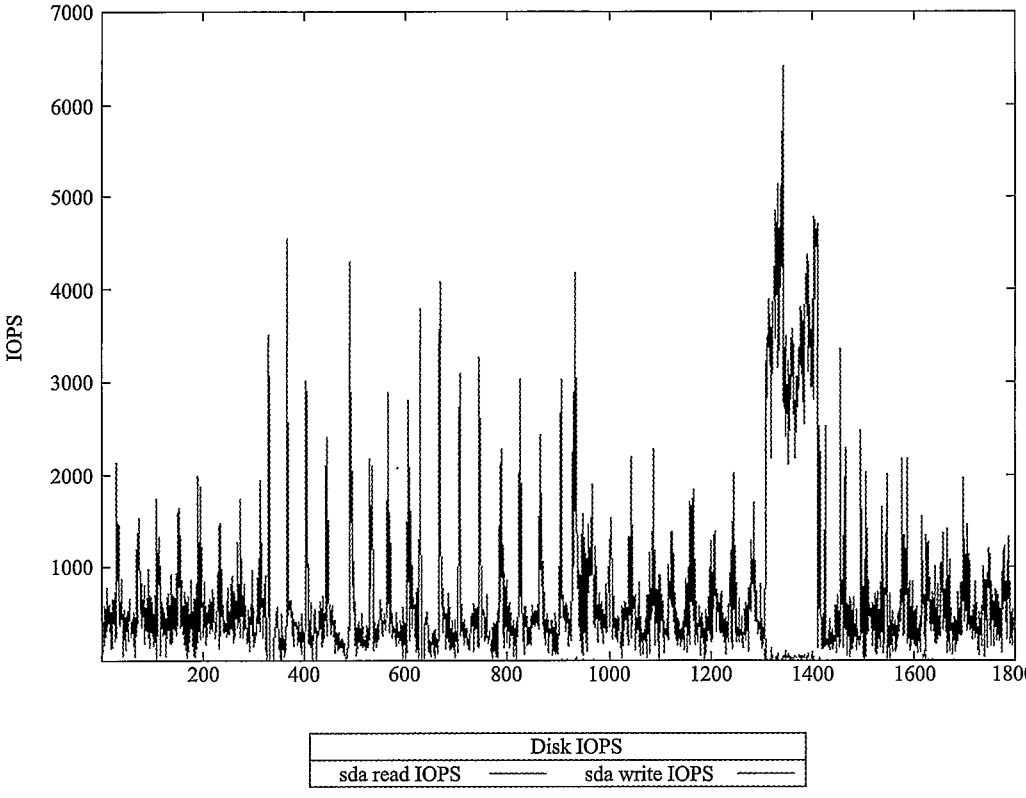


图 14-1 脚本和 gnulot 配合生成的磁盘 IOPS 性能

- 硬盘元素：主要参考参数是 IOPS 和 %util（在统计时间内所有处理 I/O 时间，除以总共统计时间）。

对于虚拟化接入来说，压力模型遵循木桶效应，即宿主机的最大虚拟承载能力，取决于虚拟机单个压力元素负载总和。由此很容易理解，宿主机的性能，在整个模型起到决定性作用。但是，也不是说将一台宿主机各元素的性能配置的越高就越好，因为对于虚拟化接入来说，性能只写一方面，稳定性和故障影响方面也是非常重要的，但是谁也无法保证服务器永远不宕机，对于宿主机宕机的情况，上面的所有虚拟机必定会受影响，那么，合理地平衡虚拟化，避免故障影响范围，是非常重要的。

一般的思维方式是从业务方出发，根据业务的压力情况，配置最合理的宿主机来承载业务的虚拟机，当然这个思路对于公有云来说就不适用了。对于公有云来说，会有另外一套逻辑来处理虚拟机分布问题。

14.3 Windows 系统性能数据采集原则与方法

Windows 系统性能采集，大致可以参考 Linux 系统的思路，只是采集方法略有区别，在

1107114476@qq.com
dolphinebook.com

Windows 系统中，自带了一个工具，可以很方便、直观地采集和观察服务器性能。本节将介绍如何在 Windows 系统中采集服务器性能，以及使用哪个工具可以直观地分析性能数据。

1. Windows 性能采集原则

对于 Windows 系统来说，采集性能的原则包括：操作简单、数据准确、低消耗、通用性强。操作简单，这一点还是最重要的，需要尽可能保证不会因为操作失误而取不到性能数据。一般通过系统自带的工具或接口取到的性能数据都比较准确，而且相对来说自身占用资源也非常少。通用性强也是非常重要的，这就需要在创建脚本的时候，考虑到每个版本的区别，对于 Windows 服务器版系统总体还是一脉相承的，所以，基本方法都是一致的，通用性也非常强。

2. logman 介绍

在 Windows 系统中提供了一个名为 Perfmon 的工具，用它可以查看 CPU 使用率、内存使用率、硬盘读 / 写速度、网络速度等性能参数。Perfmon 的管理方式有两种：控制台和命令行。

（1）控制台方式

选择“开始”→“运行”命令，在弹出的对话框中输入 Perfmon.msc，调出性能管理的控制台，如图 14-2 所示。在性能监视器中添加需要监控的项，就可以得到实时的性能曲线。

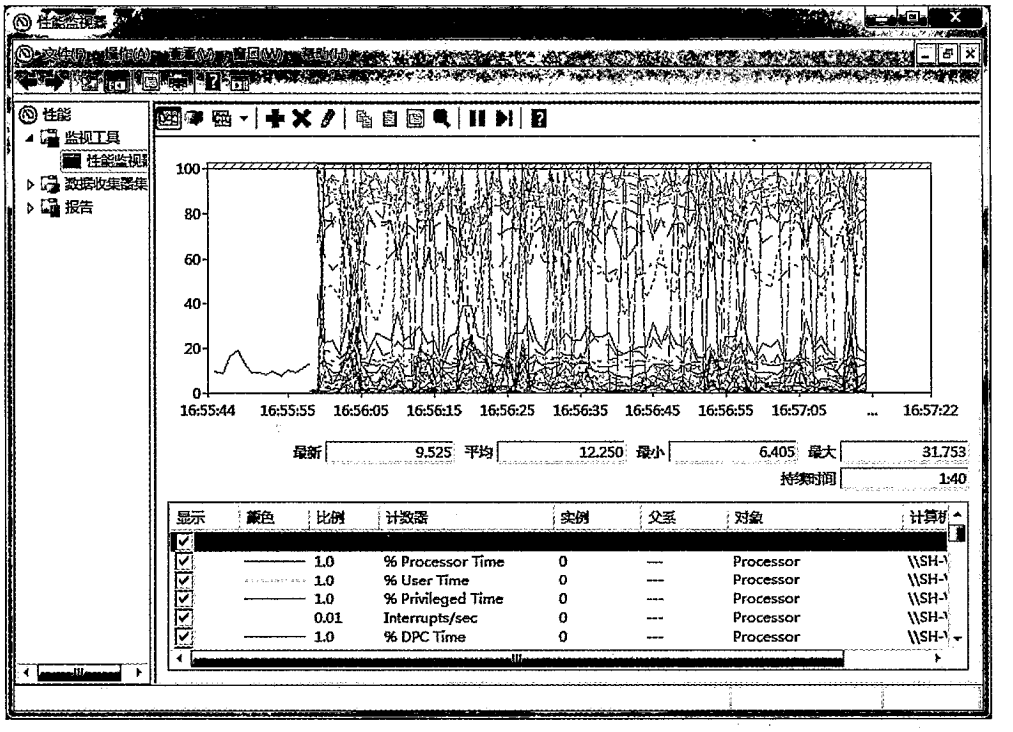


图 14-2　Perfmon 界面

1107114476@qq.com
dolphinebook.com

（2）命令行方式

命令方式就是本节需要介绍的 logman，它不仅能够在命令行中启动和停止日志会话，而且能够从命令行创建新的日志会话。通过使用 logman 可以很方便地利用 Perfmon 工具采集 Windows 系统的性能数据。

3. Windows 性能采集脚本实例分析

Windows 性能采集脚本分为两个部分。

第一部分是过滤的性能采集项，这些项可以保存在一个 txt 文件中，如 permon.txt，内容如下：

```
\Processor(*)\% Processor Time
\Memory\Committed Bytes
\Network Interface(*)\Packets Received/sec
\Network Interface(*)\Packets Sent/sec
\Network Interface(*)\Bytes Received/sec
\Network Interface(*)\Bytes Sent/sec
\Network Interface(*)\Output Queue Length
\PhysicalDisk(*)\% Idle Time
\PhysicalDisk(*)\Avg. Disk sec/Read
\PhysicalDisk(*)\Avg. Disk sec/Write
\PhysicalDisk(*)\Avg. Disk Bytes/Read
\PhysicalDisk(*)\Avg. Disk Bytes/Write
\PhysicalDisk(*)\Current Disk Queue Length
\PhysicalDisk(*)\Avg. Disk Queue Length
\PhysicalDisk(*)\Disk Reads/sec
\PhysicalDisk(*)\Disk Writes/sec
\LogicalDisk(*)\% Free Space
\LogicalDisk(*)\Free Megabytes
```

这些过滤项依次是 CPU 时间利用率、内存使用量、网卡每秒收包数、网卡每秒发包数、网卡每秒接收字节数、网卡每秒发送字节数、网卡队列、硬盘空闲时间百分比、硬盘每次读时间、硬盘每次写时间、硬盘每次读字节数、硬盘每次写字节数、当前磁盘队列长度、平均磁盘队列长度、每秒读次数、每秒写次数、逻辑磁盘剩余空间百分比、逻辑磁盘剩余空间。

第二部分是性能采集脚本，内容如下：

```
@echo off
:begin
cls
md c:\perflogs
echo.
echo 运行此脚本，请将 perfmon.txt 放于 c:\ 下
echo.
set servername=
set /p servername= 请输入服务器名称，项目 - 角色 -ip:
set mstart=
```

1107114476@qq.com
dolphinebook.com

```
set /p mstart= 请输入任务开始时间，格式 2010-12-25 20:59:30  :
set mend=
set /p mend= 请输入任务结束时间，格式 2010-12-25 21:59:30  :
logman query | find "%servername%" && logman delete %servername% && echo 计划任务
%servername% 已存在，将执行删除操作，并重新创建 ...
logman create counter %servername% -f csv -b %mstart% -e %mend% -o "c:\
perflogs\%servername%" -v mmddhhmm -cf c:\perfmon.txt  -si 00:30
echo.
echo. 计划任务 %servername% 已创建
echo 性能取样时间是 30 秒
echo 计划任务开始时间是 %mstart%，结束时间是 %mend%
echo 性能收集日志输出路径是 c:\perflogs\
echo.
pause
goto :eof
```

在运行这个脚本时，必须将第一步中的 permon.txt 放在 C:\ 下。

根据脚本的提示，选择在 Windows 服务器压力最大的时候采集性能。

14.4 Windows 系统压力模型建立

Windows 系统压力模型建立的思想和 Linux 系统非常相似，同样也是以 CPU、网卡、磁盘三方面的压力数据为主要参照，进行整理分析。

1. 分析性能采集数据

在 Windows 系统性能采集的过程中，已经通过定义 logman 采集项的方式，采集了最关键的项。

1）CPU 方面，只取了 CPU 时间，也就是通常说的利用率，这项值越大，说明 CPU 压力越大。

2）网卡方面，通过发包率（每秒收发包数量）和吞吐两项来衡量网卡压力，一般来说，Windows 系统的虚拟机，在网卡发包率这项要特别注意，因为，由于驱动的关系，运行在 KVM 平台上的 Windows 虚拟机网卡发包率远远不如 Linux 虚拟机，通过测试，大概是 Linux 虚拟机网卡发包率 1/3 的性能。

3）磁盘方面，通过 IOPS（每秒读次数、每秒写次数）和 IO 延迟（硬盘每次读时间、硬盘每次写时间），来判断磁盘的压力，如果 IOPS 不高，但是 IO 延迟很大，说明磁盘性能是不足的。

总体来说，通过上述三点分析，能得出一个正在运行的 Windows 负载情况，对于内存这项，我们并不需要太大的关注，因为虚拟化过程中，宿主机的内存，往往不会成为瓶颈，宿

1107114476@qq.com
dolphinebook.com

主机需要多少内存，完全可以根据虚拟机的内存总和来配置。

2. 利用 PAL Tool 展示系统压力

PAL（Performance Analysis of Logs）是一个用于分析 Perfmon 计数器日志的实用工具，而且完全免费，最终会产生一个报告，报告用颜色标记计数器，依次来展示问题区，PAL 默认提供了很多阈值配置模板，这些都是微软支持团队的主题专家定义的阈值，当然，阈值配置模板也可以根据自己的情况自行定义。本节将介绍如何利用 PAL Tool 来展现系统压力。

首先介绍 PAL 的安装。安装程序可以从 http: //pal.codeplex.com/ 官网获取，当前最新版是 PAL 2.7.1，PAL Tool 安装需要依赖 3 个组件：Windows PowerShell 2.0 or greater、.NET Framework 3.5 Service Pack 1 及 Microsoft Chart Controls for .NET Framework 3.5。安装完成之后，如图 14-3 所示。

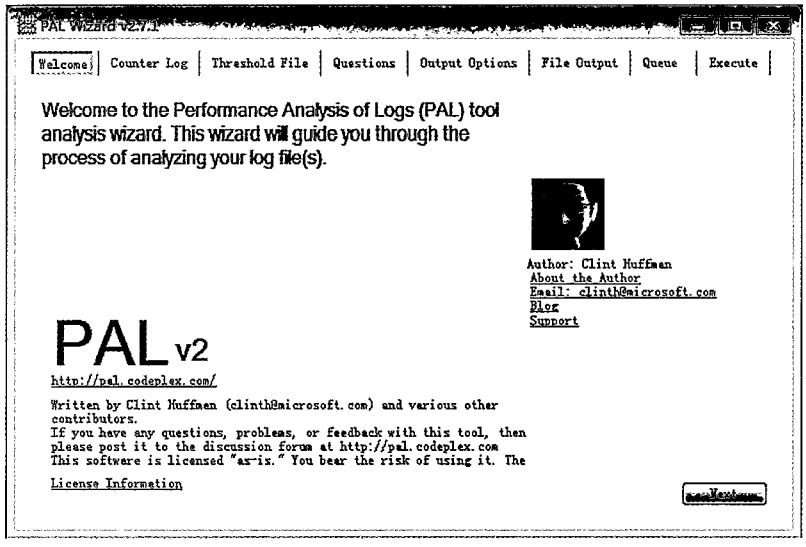


图 14-3 PAL Tool 软件界面

软件操作非常简单，根据提示单击“Next”按钮，大部分都可以保持默认配置，在 Threshold File 这一步既可以选择系统自带的阈值模板，也可以自定义。最后将生成一个 HTML 的报告页面，如图 14-4 所示。

报告的内容非常详细，可能很直观地看到系统各方面的压力情况，读者，可以尝试用 PAL Tool 分析一下 Permon 生成的日志。

3. Windows 压力模型建立

Windows 压力模型建立的思路和 Linux 系统是一致的，也是从 CPU、网卡、磁盘三方面压力来分析，再通过宿主机能承载的 CPU、网卡、磁盘压力范围，最终计算出虚拟化比例。

14.2 节中已经做了分析，本节就不再重复了。

1107114476@qq.com
dolphinebook.com

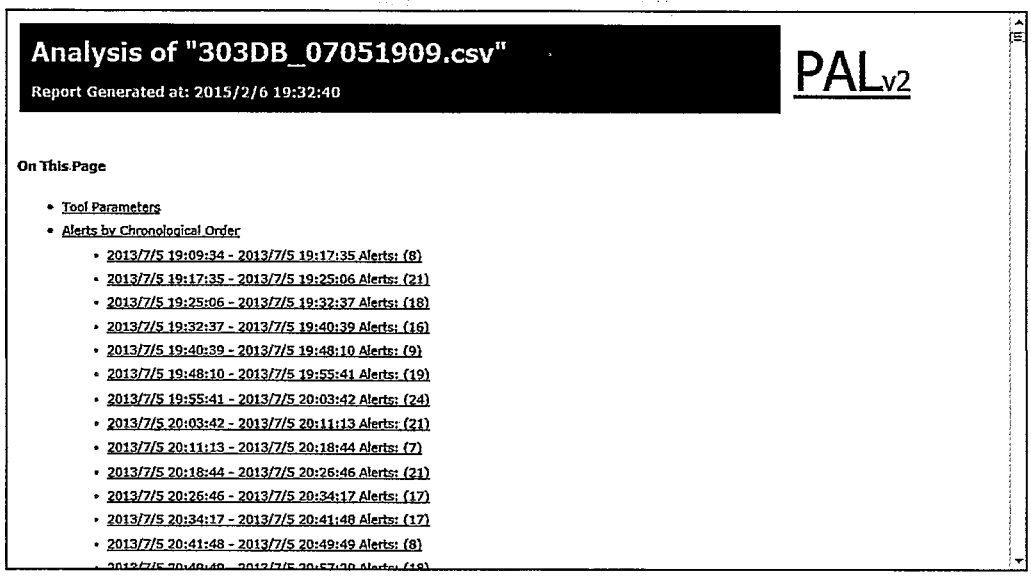


图 14-4 PAL Tool 软件生成的报告

14.5 本章小结

本章主要介绍了在虚拟化项目接入之初，对现有业务进行性能评估，以确保将业务从物理服务器迁移到虚拟机中，性能方面做到无缝衔接。另外，分别对 Linux 系统和 Windows 系统性能采集，以及使用 gnuplot 和 PAL Tool 分别对采集到的性能数据进行直观的展示。希望读者从本章的介绍中，了解性能采集和分析的原理，并最终结合自己的实际情况适当的变通，最终得到最适合自己业务场景的方案。

业务评估完成之后，下一步是实施虚拟化，首先面临的问题就是宿主机如何选型，下一章将介绍宿主机如何选型。

1107114476@qq.com
dolphinebook.com

Chapter 15 第 15 章

宿主机选型与基础性能测试

因为虚拟化是一台宿主机上运行多台虚拟机，所以，对宿主机的稳定性和性能有非常高的要求。本章介绍宿主机选型应该关注哪些指标，如何进行宿主机的性能测试，取得基准数据，对宿主机的使用做到心中有数。本章介绍的测试方法也可以用于虚拟机的测试，以取得虚拟机的基准数据。

15.1 宿主机 CPU 选型原则

CPU 选型就是首先确定自己的业务模型，然后根据业务模型决定是选频率高的 CPU，还是核数多的 CPU。

1. Intel CPU 体系

Intel CPU 的发展战略是钟摆模式（Tick-Tock），是在 2007 年正式提出的。基本上是一年做架构更新 Tock，即芯片制程的基础上，更新处理器架构，提升性能。一年做芯片制程的更新 Tick，即在 CPU 性能相同下，缩小芯片面积、减少功耗。表 15-1 所示为 Intel CPU 处理器架构制程更新路线总结。

表 15-1 Intel CPU 处理器架构制程更新路线总结

微架构更新	更新周期	微架构	制造工艺 / 制程
Tick	制程改进更新	Presler, Cedar Mill, Yonah	65 纳米
Tock	微架构更新	Core	
Tick	制程改进更新	Penryn	45 纳米
Tock	微架构更新	Nehalem	

1107114476@qq.com
dolphinebook.com

（续）

微架构更新	更新周期	微架构	制造工艺 / 制程
Tick	制程改进更新	Westmere	32 纳米
Tock	微架构更新	Sandy Bridge	
Tick	制程改进更新	Ivy Bridge	22 纳米
Tock	微架构更新	Haswell	
Tick	制程改进更新	Broadwell	14 纳米
Tock	微架构更新	Skylake	
Tick	制程改进更新	Skymont	10 纳米

Intel 的处理器产品线划分策略实际上就是一个核心，从高到低配置不同的缓存、频率、核数，由此衍生出多个系列的产品。

Intel 目前主要的产品线如下：

- 笔记本电脑为酷睿 M 处理器。
- PC 为酷睿 I3、I5、I7 处理器。
- 服务器为志强 E3、E5、E7 处理器。
- 手机、平板为凌动处理器 。

CPU 编号之所以是 I3、I5、I7，或者 E3、E5、E7，主要是市场销售策略，能够区分低端、中端、高端产品，并没有实际的含义。

服务器端产品线编号意义如图 15-1 所示。

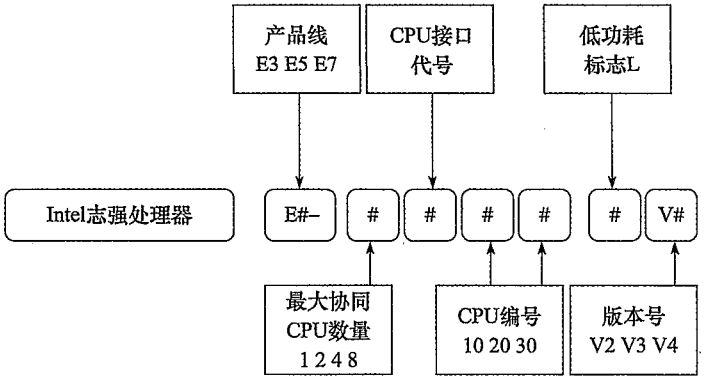


图 15-1　Intel CPU 编号意义

例如，E5-2640V2 就是位于 E5 这条产品线，最多可以两颗 CPU 共同工作，CPU 接口代号是 2，40 是 CPU 的编号，v2 是版本号。

另外，目前的志强 CPU 都支持睿频技术，英特尔睿频加速技术可以理解为自动超频。当开启睿频加速之后，CPU 会根据当前的任务量自动调整 CPU 主频，从而在重任务时发挥最大的性能，轻任务时发挥最大节能优势。例如，E5-2640V2 标称频率是 2.0GHz，当压力

1107114476@qq.com
dolphinebook.com

不大时，可能工作在 1.2GHz，随着压力增大，逐步可以跳到 1.8GHz、2.0GHz、2.5GHz，最大支持跳到 2.5GHz，并且睿频技术可以细分到每个具体的 CPU 核上，E5-2640V2 是 8 个核，可能有的核工作在 1.2GHz，有的核工作在 2.5GHz。

2. 如何选择宿主机 CPU

宿主机 CPU 选型可以分为三步：

第一，查询 Intel 的产品线，了解 CPU 的核数、频率、缓存情况，Intel 官网也可以查到售前价格。

第二，要清楚自己业务的 CPU 消耗类型，分析自己的业务，确认自己业务的 CPU 消耗类型，是多进程还是单进程，是频率消耗型还是可以横向扩展，能充分利用多核 CPU 资源的类型。

第三，结合第一步和第二步的信息，综合价格，就可以确定一款非常合适的 CPU。

15.2 宿主机内存选型注意事项

服务器的内存有许多独特的校验技术，目前内存类型和 CPU 平台是绑定的，且选择了 CPU，就只能选择配套的内存类型，所以，在选择宿主机内存时，最应该关注的是总容量、单根容量、条数、内存频率这些指标。

1. 服务器内存技术

目前最新的服务器上使用的都是 DDR4 内存，DDR4 全称是第四代双倍数据率同步动态随机存取存储器（Double-Data-Rate Fourth Generation Synchronous Dynamic Random Access Memory，简称为 DDR4 SDRAM），DDR4SDRAM 提供更低的供电电压和更高的带宽，支持 2133 ~ 4266MT/s 的数据传输率，比 DDR3 SDRAM 拥有更好的功耗表现，DDR4 只有 1.05V ~ 1.2V 的供电电压（DDR3 的为 1.2V ~ 1.65V）。对于服务器而言，DDR4 还提供了 Banks 切换特性，但这样使得服务器用 DDR4 存储器与桌面版本的 DDR4 存储器从物理层面上就无法互用。相较于 DDR3，一个 rank 单元内的 bank 单元数量增长至 16 个（4 个 bank 选择比特），每个 DIMM 模块最高拥有 8 个 rank 单元。

2. 服务器内存的纠错技术介绍

（1）Parity（奇偶校验）技术

Parity（奇偶校验）技术是比较早的纠错技术，简单来说就是每 8 位加一个校验位，每次对内存读 / 写时都会核对校验，但是这样的校验方法只能发现一位内存错误，并且不能修正内存错误。

（2）内存 ECC 技术

ECC（Error Checking and Correcting，错误检查和纠正）技术可以对内存数据进行一位的校验，发现错误还可以修正。它与 Parity 不同的是，如果数据位是 8 位，则需要增加 5 位

1107114476@qq.com
dolphinebook.com

来进行 ECC 错误检查和纠正，数据位每增加一倍，ECC 只增加一位检验位，也就是说当数据位为 16 位时，ECC 位为 6 位，当数据位是 32 位时，ECC 位为 7 位，数据位为 64 位时，ECC 位为 8 位，以此类推，数据位每增加一倍，ECC 位只增加一位。

ECC 技术的特点是，在内存中 ECC 能够容许错误，并可以将错误更正，使系统得以持续正常地操作，不致因错误而中断，且 ECC 具有自动更正的能力，可以将 Parity 无法检查出来的错误位查出并将错误修正。除了能够检查到并改正单比特错误之外，ECC 码还能检查到（但不改正）单 DRAM 芯片上发生的任意两个随机错误，并最多可以检查到 4 位的错误。当有多位错误发生时，ECC 内存会生成一个不可隐藏（Non-Maskable Interrupt）的中断（NMI），会中止系统运行，以避免出现数据恶化。

提示　一些厂商推出的入门级低端服务器使用的多是普通 PC 用的内存，不带 ECC 功能，在选购时应该注意这个指标。

3. 内存镜像技术

内存镜像（Memory Mirroring）技术很像磁盘 RAID1，将数据同时写到两块内存中，每块内存的大小要求是一样的。不同的是，两块内存做的是主备，正常工作情况下，内存数据读取只从主内存中读取，只有当活动内存出现故障时，才会从镜像内存中读取数据。

如果一块内存中有足以引起系统报警的软故障，系统会报告系统管理员，提醒管理员这个内存条将要出故障；同时服务器就会自动地切换到使用镜像内存卡，直到这个有故障的内存被更换。镜像内存允许进行热交换（Hot Swap）和在线添加（Hot-add）内存。因为镜像内存采用的两套内存中实际只有一套在使用，另一套用于备份，所以，对于系统来说也只有整个内存的一半容量是可用的。要使用镜像内存，需要在 BIOS 中进行配置。

4. 内存加速技术介绍

内存加速技术有三种：

（1）内存 Register

Register 即寄存器或目录寄存器，可以把 Register 理解成书的目录或者是数据库的索引，有了 Register，当内存接到读 / 写指令时，会先检索此目录，然后再进行读 / 写操作，若所需数据在目录中，则直接取用不再进行读 / 写操作，这将大大提高服务器内存工作效率。

（2）内存 FB-DIMM

FB-DIMM（Fully Buffered-DIMM，全缓冲内存模组）是 Intel 在 DDR2 的基础上发展出来的一种新型内存模组与互联架构，它既可以搭配 DDR2 内存芯片，也可以搭配 DDR3、DDR4 内存芯片。FB-DIMM 可以极大地提升系统内存带宽并且极大地增加内存最大容量。

（3）内存多通道

在北桥芯片中设计两个或者多个内存控制器，这些内存控制器可以相互独立工作，每个

1107114476@qq.com
dolphinebook.com

控制器控制一个内存通道。在这些内存通道上，CPU 可分别寻址、读取数据，从而使内存的带宽增加一倍或者几倍，数据存取速度也相应增加。

5. 宿主机内存选型注意点

目前，内存一般都是和 CPU 搭配使用的，一旦选择了 CPU 和服务器的品牌型号，相应地内存的频率、多通道等都是唯一的，因此，内存的选择应关注以下 4 点：

- 内存应该配置多少条。
- 每条的容量多大。
- 共需要配置内存容量到多少。
- 内存频率。

例如，需要配置宿主机内存为 128GB，如果配置两个 CPU，一般每个 CPU 是双通道，两个 CPU 就是 4 通道，所以，内存的条数必须是 4 的倍数，128GB 可以使用 16 个 8GB 的内存，也可以使用 8 个 16GB 的内存，具体如何配置也要考虑实际的价格及以后的扩展性。

15.3 宿主机硬盘选型

本节介绍宿主机硬盘的选型，一般生产环境硬盘使用 SAS 硬盘，如果磁盘 I/O 压力特别大，可以考虑使用 SSD 硬盘。

1. 磁盘类型

目前，服务器上常见的硬盘有如下 3 种类型：

（1）机械硬盘

机械硬盘（HDD）由盘片、磁头、电动机、控制器组成，数据读 / 写主要是通过盘片转动和电动机驱动磁头横向移动完成的。所以，HDD 硬盘的容量取决于盘片的密度和盘片的数量。数据读 / 写速度主要取决用盘片的转速。一般 PC 常见的 SATA 转速是 5400r/min 或者 7200r/min，服务器用的 SAS 硬盘，一般是 10 000r/min，或者 15 000r/min。

（2）固体硬盘

固体硬盘（Solid State Disk,SSD）是用存储器作为存储，类似常用的 U 盘，没有电动机，所以没有噪音，体积也很小。

（3）PCI-E 加速卡

PCI-E 加速卡是将存储器配置到一块 PCI-E 接口的卡上，与 CPU、内存进行交互，直接走 PCI-E 总线，速度要快很多。

2. 硬盘的接口

最早的硬盘是并行接口，主要是 ATA（IDE）和 SCSI 接口，后来发展到串行接口——SATA 和 SAS，另外还有光纤通道接口。

1107114476@qq.com
dolphinebook.com

（1）ATA

ATA 的全称为 Advanced Technology Attachment，是用传统的 40-pin 并口数据线连接主板与硬盘，接口速度最大为 133Mbit/s，因为并口线的抗干扰性太差，且排线占用空间较大，不利于计算机内部散热，已逐渐被 SATA 所取代。

（2）SATA

SATA 的全称为 Serial ATA，也就是使用串口的 ATA 接口，抗干扰性强，且对数据线的长度要求比 ATA 低很多，支持热插拔等功能，SATA-II 的接口速度为 375Mbit/s，而新的 SATA-III 标准可达到 750Mbit/s 的传输速度。SATA 的数据线也比 ATA 细得多，有利于机箱内的空气流通，整理线材也比较方便。

（3）SCSI

SCSI 的全称是 Small Computer System Interface（小型机系统接口），经历多代的发展，从早期的 SCSI-II 到目前的 Ultra320 SCSI 以及 Fiber-Channel（光纤通道），接口型式也多种多样。SCSI 硬盘广为工作站级个人计算机以及服务器所使用，因此，会使用较为先进的技术，如碟片转速 15 000r/m 的高转速，且资料传输时 CPU 占用率较低，但是单价也比相同容量的 ATA 及 SATA 硬盘更加昂贵。

（4）SAS

SAS（Serial Attached SCSI）是新一代 SCSI 技术，与 SATA 硬盘相同，都是采取序列式技术以获得更高的传输速度，可达到 6Gbit/s。此外，也通过缩小连接线改善系统内部空间等。

由于 SAS 硬盘可以与 SATA 硬盘共享同样的背板，因此，在同一个 SAS 存储系统中，可以用 SATA 硬盘来取代部分昂贵的 SAS 硬盘，节省整体的存储成本。但 SATA 存储系统并不能连接 SAS 硬盘。

（5）FC

FC 的全称是 Fibre Channel，即光纤通道接口。拥有此接口的硬盘在使用光纤连接时具有热插拔性、高速带宽（4Gbit/s 或 10Gbit/s）、远程连接等特点；内部传输速率也比普通硬盘更高。限制于其高昂的售价，通常用于高端服务器领域。

3. 硬盘的缓存

磁盘缓存相当于在硬盘上有一块内存，所有来往的数据都通过缓存中转，达到提升读 / 写效率的目的。一般服务器硬盘、硬盘缓存都是被关闭的，因为如果机器掉电，会造成数据丢失。服务器上都是使用 RAID 卡的缓存来提升读 / 写效率，服务器的 RAID 卡一般都有电池保护，在服务器突然断电时，能保证将缓存的数据刷到硬盘上，或者能在缓存上保存一段时间。

4. SATA、SAS、SSD 硬盘应用场景比较

SATA 硬盘在生产环境主要有如下两种应用场景：

- 第一种场景主要是备份和对性能不高，但是磁盘空间要求比较大的场景。

1107114476@qq.com
dolphinebook.com

- 第二种场景是可以与其他高速设备结合，利用高速设备做缓存，通过高低搭配，实现性能和容量的平衡。

SAS 硬盘是目前在生产环境使用最多的硬盘，其性能比 SATA 硬盘高 3 ~ 5 倍，对 CPU 的消耗也要低很多，如果是高磁盘 I/O 消耗性的应用，一般要搭配高缓存带电池的 RAID 卡使用。

SSD 硬盘正在逐步普及，在虚拟化中主要应用在以下需要高 IOPS 的场景：

- 磁盘的随机写 IOPS 大于 2000IOPS 的应用场景。
- 磁盘的随机读 IOPS 大于 10000IOPS 的应用场景。

15.4 RAID 卡选型

生产环境宿主机 RAID 级别一般使用 RAID10，RAID 卡选择带缓存和电池的卡，可以有效地提示宿主机的 I/O 性能。

1. RAID 技术介绍

RAID（Redundant Array of Independent Disks）即廉价冗余磁盘阵列，目前常用的 RAID 级别有 RAID0、RAID1、RAID5、RAID6 和 RAID10。

- RAID0：数据分到多块硬盘上，并同时读 / 写，可以提升磁盘读 / 写的性能。
- RAID1：保存一样的两份数据到两块磁盘上，当一块磁盘有故障时，数据不丢失。
- RAID5：由 3 块以上的磁盘组成，数据分块保存到磁盘上，并有数据的校验块，多个磁盘共同参与读 / 写，可以提升读 / 写性能，并且当一块磁盘出现故障时，通过数据校验块，数据可以恢复，不会造成数据丢失。
- RAID6：与 RAID5 很相似，但是算法更复杂，有两块硬盘冗余，可以在两块硬盘故障的情况下不丢数据。
- RAID10：多个两块硬盘的 RAID1 组，再组合成 RAID0，这样兼有 RAID0、RAID1 的好处，既有数据冗余，又能保证性能。

RAID 有软 RAID 和硬 RAID 之分：

- 软 RAID 是在操作系统层面完成 RAID 计算，需要消耗 CPU 的计算资源。
- 硬 RAID 是将 RAID 算法固话到硬件上，通过专门的硬件来完成，这就是 RAID 卡，RAID 卡也可以理解成一台专用的计算机，它有计算、缓存、存储、通信单元，只是的功能单一，就是用于 RAID 计算。

一般服务器上都使用硬件的 RAID 卡，可以不占用 CPU 资源。另外，硬件的 RAID 卡一般有电池保护，当服务器掉电时，可以保证写到 RAID 卡上的数据不丢失。

2. RAID 卡选型

选择 RAID 卡时要注意以下参数：

1107114476@qq.com
dolphinebook.com

- 支持的RAID级别。
- 缓存大小。
- 是否带电池。

服务器的RAID从功能上可以分为以下两种：

- 入门级的RAID卡。一般这种RAID卡只支持RAID1、RAID0，RAID10，不支持RAID5，没有缓存，不带电池，用于CPU消耗性的服务器。
- 更高级一点的RAID卡。支持RAID5甚至RAID6，带有缓存和电池。一般用在磁盘I/O密集的服务器上。

> 目前一些新出的服务器上，有些RAID卡在硬件级别支持使用SSD硬盘做缓存，磁盘I/O密集的场景也可以考虑这种服务器。

15.5 服务器网卡选型

服务器网络的一个特点是连接并发数高，对计算资源的要求非常高，较低的CPU占用率对于服务器网卡来说是非常重要的，所以，服务器网卡都有自己的计算单元，能够处理一些网络方面的计算压力。

1. 服务器网卡介绍

服务器专用网卡一般具有如下特点：

- 低延时、CPU占用率低。
- 稳定性和安全性。
- 多网卡、高带宽。

服务器专用网卡可以从主CPU中接管许多网络任务，使主CPU集中“精力”运行网络操作和应用程序，当然服务器的服务性能也就不会再受影响了。

服务器最重要的是稳定性，为了保证服务器的稳定性，所有配件都必须具备同样高的品质。服务器网卡除了芯片性能够好之外，用料充足和做工精良也是必需的。网卡的稳定性也体现在数据传输的表现上，服务器网卡的传输曲线都很平滑，升降波动少。网络传输是以数据包的形式发送的，如果网络质量不好，就容易发生丢包的现象。如果网卡性能不够好，那么在网卡负载大时，丢包的概率就会上升，发生丢包后，数据就要重新传输，反过来再次增加网卡的压力。

服务器网卡一般采用PCI Express总线，PCI Express的接口根据总线位宽不同而有所差异，包括X1、X4、X8以及X16（X2模式将用于内部接口而非插槽模式）。较短的PCI Express卡可以插入较长的PCI Express插槽中使用。PCI Express接口能够支持热拔插。

目前，常见服务器一般配置4个或者8个网卡，支持1GB/s、10GB/s甚至40GB/s的

1107114476@qq.com
dolphinebook.com

带宽。

2. 服务器网卡选型

笔者的宿主机网卡的选型经验如下：

❑ 如果选择千兆网卡，建议选择 Intel 的 I350 网卡。

❑ 如果选择需要万兆网卡，建议选择 Intel 的 x520 或者 x540 的网卡。

因为这些网卡都支持 SRIVO，SRIOV 是非常好的虚拟化性能解决方案，I350 千兆网卡支持 8 块子网卡，x520 或 x540 的万兆网卡支持 64 块子网卡。

选择 Intel 网卡，还因为 Intel 的 DPDK 技术，DPDK 技术是一套开发包，在内核、虚拟化层、VirtIO 接口都针对 Intel 硬件做了性能优化。

15.6 宿主机性能基准测试实践

在 KVM 技术早期，虚拟机的 CPU 中断和上下文切换消耗比较严重。随着目前虚拟化技术的发展，CPU 性能从物理机转化到虚拟机，损失已经非常少了，往往计算方面的瓶颈受限于物理 CPU 的性能，而 CPU 性能的测试，在不同场景下差异非常大，所以，一般 CPU 在虚拟化环境下的测试，只是简单对比一下不同型号 CPU 计算能力差异。

1. CPU 测试工具与测试方法

（1）测试工具

笔者经常使用的测试工具主要有两款：Super PI 和 UnixBench。

① Super PI。

Super PI 的官方地址为 http: //www.superpi.net/，在官网的 http: //www.superpi.net/Scores/?action=searchScores&test=-12&vendor=&platform=&series=&rpp=20 链接中可以查到大部分 CPU 的得分，有 1M 和 32M 两种。

官网下载的是 Windows 版本，Linux 版本可以在 ftp: //pi.super-computing.org/Linux/super_pi.tar.gz 链接中下载。

下载后解压就可以直接运行了，如果报如下错误：

```
./super_pi 20
./super_pi: ./pi: /lib/ld-linux.so.2: bad ELF interpreter: No such file or directory
```

是因为在 64 位的系统上，需要安装 32 位的库文件：

```
yum install glibc.i686
./super_pi 20

...
 ------ Started super_pi run : Sun Dec 28 22:35:32 CST 2014
 Start of PI calculation up to 1048576 decimal digits
```

1107114476@qq.com
dolphinebook.com

```
 End of initialization. Time=       0.235 Sec.

...
 I=19 L=   89415        Time=       0.474 Sec.
 End of main loop
 End of calculation.    Time=      11.095 Sec.
 End of data output.    Time=       0.064 Sec.
 Total calculation(I/O) time=      11.159(      0.276) Sec.
 ------ Ended super_pi run :
```

Super PI 是一个典型的 CPU 密集型基准测试工具，原理非常简单，根据设置计算圆周率 π 小数点后 N 个位数，然后统计消耗的时间，根据时间长度的比较就能初步衡量 CPU 计算能力的强弱。Super PI 主要是浮点计算能力的比较。

② Unixbench。

Unixbench（https: //code.google.com/p/byte-unixbench）是一套 UNIX 系统基准测试套件。Unixbench 的设计目标是为类 UNIX 系统提供一套基本的指标，有许多项目测试系统各方面的性能。各项的测试有得分，还有一个综合的得分，这样可以很方便地通过分数去比较。

Unixbench 也包含一些非常简单的 2D 和 3D 图形测试。

Unixbench 也支持多 CPU 系统的测试，默认的行为是测试两次，第一次是一个进程的测试，第二次是 N 份测试，N 等于 CPU 个数。这样的设计是为了以下目标：

- ❑ 测试系统的单任务性能。
- ❑ 测试系统的多任务性能。
- ❑ 测试系统并行处理的能力。

Unixbench 是一个基于系统的基准测试工具，不单纯是 CPU 内存或者磁盘测试工具。测试结果不仅取决于硬件，还取决于系统、开发库和编译器。

（2）测试项目

① Dhrystone 测试。

Dhrystone 测试聚焦在字符串处理，没有浮点运算操作。这个测试用于测试链接器编译、代码优化、内存缓存、等待状态、整数数据类型等，硬件和软件设计都会影响测试结果。

② Whetstone 测试。

这项测试项目用于测试浮点运算效率和速度，包含若干个科学计算的典型性能模块、大量的 C 语言函数（sin、cos sqrt、exp）和日志，以及使用整数和浮点的数学操作，还包含数组访问、条件分支和过程调用。

Execl Throughput（Execl 吞吐，这里的 Execl 是类 UNIX 系统非常重要的函数，非办公软件的 Excel）测试

这项测试测试每秒 Execl 函数的调用次数。Execl 是 Exec 函数家族的一部分，使用新的图形处理代替当前的图形处理。有许多命令和前端的 Execve() 函数命令非常相似。

1107114476@qq.com
dolphinebook.com

③ File Copy 测试。

这项测试衡量文件数据从一个文件被传输到另外一个，使用大量的缓存，包括文件的读、写、复制测试，测试指标是一定时间内（默认是 10 秒）被重写、读、复制的字符数量。

④ Pipe Throughput（管道吞吐）测试。

Pipe 是简单的进程之间的通信。管道吞吐测试是测试在一秒内一个进程写 512bit 到一个管道中并且读回来的次数。管道吞吐测试和实际编程有差距。

⑤ Pipe-based Context Switching（基于管道的上下文交互）测试。

这项测试衡量两个进程通过管道交换和整数倍的增加吞吐的次数。基于管道的上下文切换和真实程序很类似，测试程序产生一个双向管道通信的子线程。

⑥ Process Creation（进程创建）测试。

这项测试衡量一个进程能产生子线程并且立即退出的次数。新进程真的创建进程阻塞和内存占用，所以，测试程序直接使用内存带宽。这项测试用于典型的比较大量的操作系统进程创建操作。

⑦ Shell 脚本测试。

Shell 脚本测试用于衡量在一分钟内一个进程可以启动并停止 Shell 脚本的次数，通常会依次按照 1，2，3，4，8 的参数，运行 Shell 脚本的共同复制，Shell 脚本是一套转化数据文件的脚本。

⑧ System Call Overhead（系统调用消耗）测试。

这项测试衡量进入和离开系统内核的消耗，例如，系统调用的消耗。程序简单重复地执行 getpid 调用（返回调用的进程 id）。消耗的指标是调用进入和离开内核的执行时间。

⑨ Graphical（图形）测试。

Unixbench 图形测试由 ubgears 程序组成，测试非常粗粒度的 2D 和 3D 图形性能，尤其是在 3D 测试上非常有限。测试结果与硬件、系统是否有合适的驱动关系很大。

（3）安装和使用

Unixbench 安装很简单，下载、解压、编译就可以。

```
wget http://byte-unixbench.googlecode.com/files/unixbench-5.1.2.tar.gz
tar xvfz unixbench-5.1.2.tar.gz
cd unixbench-5.1.2
```

如果不需要测试图形，注释掉 Makefile 文件的这一行即可。

```
#GRAPHIC_TESTS = defined
```

运行命令如下：

```
./Run
```

测试结果如图 15-2 所示，有分享得分和总得分。

1107114476@qq.com
dolphinebook.com

```
Version 5.1.2 Based on the Byte Magazine Unix Benchmark
1 x Dhrystone 2 using register variables 1 2 3 4 5 6 7 8 9 10
1 x Double-Precision Whetstone 1 2 3 4 5 6 7 8 9 10
1 x Execl Throughput 1 2 3
1 x File Copy 1024 bufsize 2000 maxblocks 1 2 3
1 x File Copy 256 bufsize 500 maxblocks 1 2 3
1 x File Copy 4096 bufsize 8000 maxblocks 1 2 3
1 x Pipe Throughput 1 2 3 4 5 6 7 8 9 10
1 x Pipe-based Context Switching 1 2 3 4 5 6 7 8 9 10
1 x Process Creation 1 2 3
1 x System Call Overhead 1 2 3 4 5 6 7 8 9 10
1 x Shell Scripts (1 concurrent) 1 2 3
1 x Shell Scripts (8 concurrent) 1 2 3
------------------------------------------------------------------------
1 CPU in system; running 1 parallel copy of tests
Dhrystone 2 using register variables 28890881.0 lps (10.0 s, 7 samples)
Double-Precision Whetstone 3880.4 MWIPS (9.0 s, 7 samples)
Execl Throughput 4146.3 lps (30.0 s, 2 samples)
File Copy 1024 bufsize 2000 maxblocks 1051084.3 KBps (30.0 s, 2 samples)
File Copy 256 bufsize 500 maxblocks 286552.2 KBps (30.0 s, 2 samples)
File Copy 4096 bufsize 8000 maxblocks 2142638.4 KBps (30.0 s, 2 samples)
Pipe Throughput 1726807.0 lps (10.0 s, 7 samples)
Pipe-based Context Switching 322865.5 lps (10.0 s, 7 samples)
Process Creation 13662.4 lps (30.0 s, 2 samples)
Shell Scripts (1 concurrent) 5955.4 lpm (60.0 s, 2 samples)
Shell Scripts (8 concurrent) 713.1 lpm (60.0 s, 2 samples)
System Call Overhead 2138318.1 lps (10.0 s, 7 samples)
System Benchmarks Index Values BASELINE RESULT INDEX
Dhrystone 2 using register variables 116700.0 28890881.0 2475.7
Double-Precision Whetstone 55.0 3880.4 705.5
Execl Throughput 43.0 4146.3 964.2
File Copy 1024 bufsize 2000 maxblocks 3960.0 1051084.3 2654.3
File Copy 256 bufsize 500 maxblocks 1655.0 286552.2 1731.4
File Copy 4096 bufsize 8000 maxblocks 5800.0 2142638.4 3694.2
Pipe Throughput 12440.0 1726807.0 1388.1
Pipe-based Context Switching 4000.0 322865.5 807.2
Process Creation 126.0 13662.4 1084.3
Shell Scripts (1 concurrent) 42.4 5955.4 1404.6
Shell Scripts (8 concurrent) 6.0 713.1 1188.4
System Call Overhead 15000.0 2138318.1 1425.5
========
System Benchmarks Index Score 1444.7
```

图 15-2　Unixbench 测试结果

2. 内存测试工具与测试方法

虚拟机和物理机的性能差异主要体现在内存占用与释放上，因为虚拟机的内存占用与释放需要在虚拟机上操作一次，还需要在宿主机上操作一次，即虚拟机的内存实际是被映射了两次。一方面，随着虚拟化技术的发展，如透明大页内存技术的采用，另外一方面，因为内存是一个高速设备，主要的瓶颈主要是硬件的频率、容量等，软件上的损耗和硬件比起来比率很低，因此，宿主机的内存测试一般主要是测试内存是否有错误。

笔者常用的测试工具是 Memtester 和 Memtest86+。

Memtester 是从系统中测试内存是否有错误，主要项目有随机值、异或比较、减法、乘法、除法、与或运算等，可以给定测试内存的大小和次数。

1107114476@qq.com
dolphinebook.com

Memtester 官网网址是 http: //pyropus.ca/software/memtester/。

该工具的安装很简单，下载、解压，使用 make、make install 命令即可。

```
wget  \
http:// pyropus.ca/software/memtester/old-versions/memtester-4.3.0.tar.gz
tar xvf memtester-4.3.0.tar.gz
cd memtester-4.3.0
make
make install
```

该工具的使用也很简单，加上要测试的内存大小和测试次数就可以，内存单位可以是 B、KB、MB、GB，测试结果如图 15-3 所示，如果显示 ok，表示每项测试均正常。

```
memtester 20M 2
memtester version 4.3.0 (64-bit)
Copyright (C) 2001-2012 Charles Cazabon.
Licensed under the GNU General Public License version 2 (only).
pagesize is 4096
pagesizemask is 0xfffffffffffff000
want 20MB (20971520 bytes)
got  20MB (20971520 bytes), trying mlock ...locked.
Loop 1/2:
  Stuck Address       : ok
  Random Value        : ok
  Compare XOR         : ok
  Compare SUB         : ok
  Compare MUL         : ok
  Compare DIV         : ok
  Compare OR          : ok
  Compare AND         : ok
  Sequential Increment: ok
  Solid Bits          : ok
  Block Sequential    : ok
  Checkerboard        : ok
  Bit Spread          : ok
  Bit Flip            : ok
  Walking Ones        : ok
  Walking Zeroes      : ok
  8-bit Writes        : ok
  16-bit Writes       : ok

Loop 2/2:
  ......
Done.
```

图 15-3 Memtester 测试结果

Memtester 测试结果：Memtest86+ 测试内存不需要通过系统，而是通过光盘、软盘、U 盘引导一个小型系统来检测内存。

Memtest86+ 官方网址是 http: //www.memtest.org/，可以下载光盘、软盘、U 盘的镜像，然后引导系统，进行内存测试就可以了，如图 15-4 所示。

1107114476@qq.com
dolphinebook.com

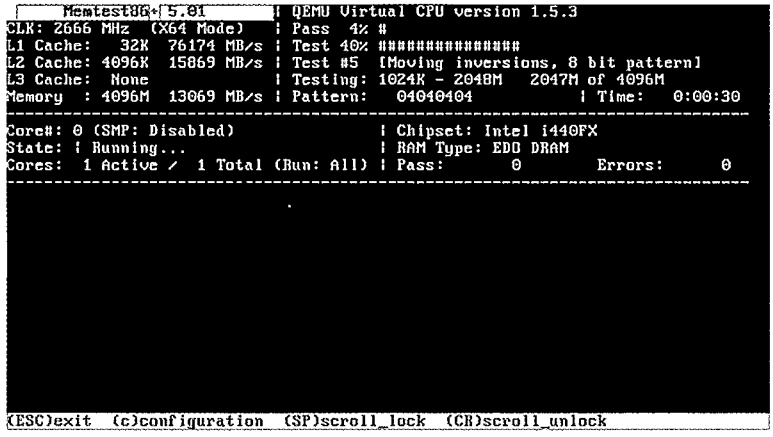


图 15-4　使用 Memtest86+ 测试内存

3. 网络 I/O 测试工具与测试方法

网络 I/O 是虚拟化中的一个难点和热点，一方面是虚拟机经常碰到网络方面的性能瓶颈，另一方面是虚拟化时，经常遇到网络稳定性方面的问题。宿主机的网络测试还有一个目的，就是提炼业务的网络压力模型，并根据自己的业务进行网络压力测试，取得基准的数据，为宿主机的使用提供依据。

网络 I/O 测试建议使用以下测试用例：

- ❑ TCP 吞吐测试。
- ❑ TCP 连接数测试。
- ❑ TCP 单连接多交易测试测试。
- ❑ TCP 发包率测试。
- ❑ UDP 吞吐测试。
- ❑ UDP 单连接多交易测试测试。
- ❑ UDP 发包率测试。
- ❑ 业务模型网络模拟测试。

提示　进行网络测试时，有的场景对 CPU 消耗很高，所以，要注意观察并记录 CPU 的利用率。

（1）测试工具

笔者常用的网络测试工具是 Netperf。

Netperf（http: //www.netperf.org/）是一款网络性能的测量工具，主要针对基于 TCP 或 UDP 的传输。Netperf 根据应用的不同，可以进行不同模式的网络性能测试，即批量数据传输（Bulk Data Transfer）模式和请求 / 应答（Request/Reponse）模式。Netperf 测试结果所反映

1107114476@qq.com
dolphinebook.com

的是一个系统能够以多快的速度向另外一个系统发送数据，以及另外一个系统能够以多块的速度接收数据。

Netperf 工具以 Client/Server 方式工作。Server 端是 Netserver，用来侦听来自 Client 端的连接，Client 端是 Netperf，用来向 Server 发起网络测试。在 Client 与 server 之间首先建立一个控制连接，传递有关测试配置的信息，以及测试的结果；在控制连接建立并传递了测试配置信息以后，Client 与 Server 之间会再建立一个测试连接，用来来回传递着特殊的流量模式，以测试网络的性能。

安装需要下载的源代码并进行编译，或者使用 epel 源通过 yum 安装也可以。

进行网络测试时，一般有如下两个关键指标：

- BPS 网络吞吐量：一分钟经过网卡的数据流量。
- PPS 发包率：一分钟经过网卡数据包的数量。

Netperf 可以模拟如下三种不同的 TCP 流量模式：

- 单 TCP 连接：传输大量数据。
- 单 TCP 连接：大量的连接次数。
- 多 TCP 连接：每个连接中一对请求 / 应答的交易方式。

Netperf 可以模拟两种 UDP 的流量模式：

- 单向批量传输。
- 请求 / 应答的交易方式。

Netperf 常用参数如下：

- -H：指定 netserver 的 server IP 地址。
- -l：指定测试的时间长度，单位为秒。
- -t：指定进行的测试类型，包括 TCP_STREAM、UDP_STREAM、TCP_RR、TCP_CRR、UDP_RR 等。

每个测试项的意义如表 15-2 所示。

表 15-2 Netperf TCP 和 UDP 测试类型

TCP_STREAM	Netperf 默认情况下进行 TCP 批量传输，即 -t TCP_STREAM。测试过程中，netperf 向 netserver 发送批量的 TCP 数据分组，以确定数据传输过程中的吞吐量
UDP_STREAM	UDP_STREAM 用来测试进行 UDP 批量传输时的网络性能。需要特别注意的是，此时测试分组的大小不得大于 socket 的发送与接收缓冲大小，否则，netperf 会报错
TCP_RR	TCP_RR 方式的测试对象是多次 TCP Request 和 Response 的交易过程，但是它们发生在同一个 TCP 连接中，这种模式常常出现在数据库应用中。数据库的 Client 程序与 Server 程序建立一个 TCP 连接以后，就在这个连接中传送数据库的多次交易过程
TCP_CRR	与 TCP_RR 不同，TCP_CRR 为每次交易建立一个新的 TCP 连接。最典型的应用就是 HTTP，每次 HTTP 交易是在一条单独的 TCP 连接中进行的。因此，由于需要不停地建立新的 TCP 连接，并且在交易结束后拆除 TCP 连接，交易率一定会受到很大的影响
UDP_RR	UDP_RR 方式使用 UDP 分组进行 request/response 的交易过程。由于没有 TCP 连接所带来的负担，所以我们推测交易率一定会有相应的提升

1107114476@qq.com
dolphinebook.com

（2）测试方法

Netperf 支持多机测试。进行测试时，一台运行服务器端程序 netserver，另一台或者多台服务器运行客户端程序 netclient/netperf.。下面介绍要测试的内容。

① TCP 吞吐量测试

Server 端运行 netserver。

Client 端运行 netclient-H server_ip-l 60-t TCP_STREAM。

② TCP 交易测试

Netperf 2.×× 版本只支持每个实例一个线程。由于一个线程产生的 TCP 交易远远达不到系统瓶颈，需要开多个 netperf 实例，并将每个实例的结果相加。

（3）测试示例

① Windows 服务端脚本 netserver.bat。

```
@echo off
for /L %%i in (12865 1 12884) do start netserver.exe -p %%i
```

② Linux 物理机客户端脚本。

```
# cat netperf.sh
#!/bin/bash

for port in $(seq 12865 1 12884)
do
      netperf -t TCP_RR -H server_ip -l 60 -p $port &
done
```

4. 磁盘 I/O 测试工具与测试方法

磁盘 I/O 测试需要测试以下两个方面：

- 基准测试：主要是比较通用的测试，如经常被提到的 4K 随机写的 IOPS。
- 业务模型测试：主要是自己的业务磁盘读 / 写的 I/O 模型是什么，然后根据自己的业务模型进行测试，可以做到有的放矢，也便于根据测试结果进行调优。

另外，Cache 对 I/O 速度的影响极大。在虚拟化环境中，一个虚拟机中的 I/O 性能受到诸多层 Cache 的影响，包括：

- Guest OS 的 buffer cache。
- Host OS 的 buffer cache。
- RAID 卡的 Cache 大小和模式。
- 物理硬盘的 Cache 和模式。

I/O 的测试数值比较，是要以保证虚拟机 OS 以下数据一致性为前提的。如果没有这个前提，比较一个带 32MB Cache 的消费级 SATA 盘和一个 Cache 被禁用的服务器专用 SAS 盘的随机写性能，是没有意义的。

1107114476@qq.com
dolphinebook.com

针对生产环境，建议采取以下配置：

❑ 禁用物理硬盘的 Cache。

❑ RAID 卡如没有电池，不要启用 writeback cache。

❑ 虚拟机硬盘模式设为 writethrough 或者 direct io 模式，以保证数据一致性。

为了使测试结果有参考意义，测试环境的软硬件的配置信息和 cache 模式必须和测试结果一起提供。表 15-3 所示是一个实际测试环境的例子。

表 15-3 测试环境配置样表

服务器型号	HP DL380G7
CPU	2 × XEON 5620
内存	24GB
RAID 卡 Cache	512MB，带电池，write back mode
硬盘	146GB，1500 转，SAS 磁盘 8 块
RAID 设置	Array1：4HD Raid10；Array2：4HD Raid5
Host OS	CentOS 5.4 64 位
Hypervisor	KVM83
虚拟机硬盘模式	vda：文件，write through mode vdb：lvm，direct io (cache=none)

宿主机的裸 I/O 性能测试，由于宿主机一般内存都比较大（16GB 甚至更大），测试时易采用 directIO 的方式，以消除宿主机系统 buffer cache 的影响。

磁盘测试 I/O 建议使用以下测试用例：

❑ 大块连续读——1024KB，顺序读。

❑ 大块连续写——1024KB，顺序写。

❑ 小块随机读——4KB，随机读。

❑ 小块随机写——4KB，随机写。

❑ 业务模拟测试用例[⊖]：20% 3KB，顺序写，79% 8KB 随机写 8KB，1% 随机读。

笔者常用的磁盘 I/O 测试工具主要有四款，下面进行介绍。

（1）fio

1）fio 介绍。

fio 的官方网址是 http: //mac.freecode.com/projects/fio，它是一款用于基准和压力测试的 I/O 工具，最新版本是 fio-2.1.10。fio 支持 19 不同的 I/O 引擎（sync、mmap、libaio、posixaio、SG v3、splice、null、network、syslet、guasi、solarisaio 等）。操作系统支持方面，fio 支持 Linux、FreeBSD、NetBSD、OpenBSD、OS X、OpenSolaris、AIX、HP-UX、Android 和 Windows 等众多操作系统。在 Linux 系统上，fio 支持测试 Linux 系统的 I/O 优先级、I/O 速率、子线程等一些比较新的内核特性。

fio 不仅支持裸设备也支持文件系统性能测试，fio 还支持文本方式的脚本，通过脚本可

⊖ 仅仅是一个例子，时间情况请读者根据自己的时间情况来确定合适的读写混合比例。

1107114476@qq.com
dolphinebook.com

以实现批量测试。

fio 的特点主要如下：

- 跨平台，支持 UNIX、Linux、Windows 系统。
- 支持多种 I/O 引擎。
- 支持命令行和脚本。
- 支持对 CPU、内存、线程、I/O 非常细致的配置。
- 支持非常丰富的测试类型，有非常细致的参数，使用简单，功能强大。

fio 的参数主要包含以下基本类型：

- I/O 类型：顺序读 / 写、随机读 / 写、混合读 / 写等。
- 块大小：测试时使用的块大小，可以是一个数值，也可以是一个范围。
- I/O 大小：读 / 写数据的大小。
- I/O 队列深度：配置队列深度数值。
- I/O 缓存类型：是否打开文件系统的缓存。
- 文件数量：同时读 / 写几个文件。
- 线程数量：同时运行几个线程。

2）fio 安装。

centos 如果安装了 epel 源，通过 yum 可以直接安装，也可以自己编译安装：

```
tar -zxvf fio-x.x.x.tar.gz
cd fio-x.x.x
make
make install
```

3）fio 参数使用。

fio 非常强大，参数也非常多，下面按照不同的类型，介绍详细的参数。

① fio 配置相关的参数及含义如下：

--debug：调试模式。

--version：显示版本信息。

② 测试任务相关的参数及含义如下：

--output=：将测试结果输出到文件，等号后是文件名字。

--runtime=：限制运行时间，单位为秒。

--name=：字符串，测试项目名称。

--description=：字符串，测试项目描述。

--filename=：测试文件，也可以指定到裸设备上。

--size=：测试文件大小。

--filesize= ：单独的文件大小，也可以是一个范围，fio 在设定的 size 之内随机地选择大小，如果没有指定，则每个子文件大小相同。

③ 读 / 写方式参数及含义如下：

1107114476@qq.com
dolphinebook.com

--readwrite=：测试读 / 写方式，可以是下面的选项：

- Read：顺序读。
- Write：顺序写。
- Trim：顺序 trim，只支持 Linux 块设备。
- Randread：随机读。
- Randwrite：随机写。
- Randtrim：随机 trim，只支持 Linux 块设备。
- rw，readnwrite：混合读 / 写。
- randrw：随机混合读 / 写。

④ 混合读写参数。如果是混合读 / 写，则默认是 50/50（即 50% 读加 50% 的写），对于特定的类型，结果会有些出入，因为速度可能不一样。可以在测试的指定一个数字：

rwmixwrite=20：混合读写，写占 20%。

⑤ 文件系统缓存方式参数。

direct=：选型可以是 1 或者 0，1 为绕过文件系统的缓存，0 为使用文件系统的缓存；

⑥ 测试块大小参数及含义如下：

- --blocksize= 或者 --bs=：测试的时候读写块大小，可以是 4K、8K、1M 等；
- --bssplit= ：字符串，需要测试混合块大小的时候使用，bssplit 参数读写的时候都可以使用。可以参考如下的格式：
  - --bssplit=4k/10: 64k/50: 32k/40，意思是 4K 占 10%，64K 占 50%，32k 占 40%；
  - 不指定百分比，fio 会平均分配，比如：--bssplit=4k/50: 1k/: 32k/，意思是 4K 占 50%，1K、32K 各占 25%；

⑦ 运行相关参数及含义如下：

- --max-job=：运行的最高线程。
- --zero_buffers ：如果使用这个参数，fio 会初始化 IO 缓存，并默认使用随机的数据填满缓存。
- --refill_buffers：强制重新填写读 / 写缓存、
- --iodepth=io：队列深度，默认是 1。
- --numjobs=：线程数量。
- --group_reporting：汇总每个进程的信息。
- --lockmem=：固件测试的时候内存大小，如 2GB。
- --nrfiles=：每个进程生成文件的数量，比如 8。

4）fio 使用案例：

文件系统 4K 顺序读测试：

```
fio --filename=/tmp/test --direct=1 --rw=read --bs=4k --size=15G --numjobs=64
--runtime=300 --group_reporting --name=test-read
```

1107114476@qq.com
dolphinebook.com

文件系统 4K 随机写测试：

```
fio --filename=/tmp/test --direct=1 --rw=randwrite --bs=4k --size=15G
--numjobs=64 --runtime=300 --group_reporting --name=test-read
```

裸设备 4K 顺序写测试：

```
fio --filename=/dev/sdb --direct=1 --rw=write --bs=4k --size=15G --numjobs=64
--runtime=300 --group_reporting --name=test-read
```

fio 的输出非常详细，图 15-5 所示为 4K 随机写的一个测试的结果，可以看到有写吞吐、IOPS 指标，还有 CPU 利用率、磁盘状态等指标。

```
fio --filename=/test --direct=1 --rw=randwrite --bs=4k --size=1G --numjobs=64 \
--runtime=300 --group_reporting --name=test-randwrite
...
  write: io=2850.2MB, bw=9726.1KB/s, iops=2431 , runt=300048msec
    clat (usec): min=159 , max=448315 , avg=410.20, stdev=1208.19
     lat (usec): min=159 , max=448315 , avg=411.15, stdev=1208.27
    clat percentiles (usec):
     |  1.00th=[  203],  5.00th=[  278], 10.00th=[  326], 20.00th=[  346],
     | 30.00th=[  366], 40.00th=[  386], 50.00th=[  402], 60.00th=[  418],
     | 70.00th=[  434], 80.00th=[  454], 90.00th=[  486], 95.00th=[  516],
     | 99.00th=[  724], 99.50th=[ 1032], 99.90th=[ 1368], 99.95th=[ 2224],
     | 99.99th=[ 8256]
    bw (KB/s)  : min=    3, max=  667, per=1.57%, avg=152.30, stdev=27.63
    lat (usec) : 250=3.92%, 500=89.07%, 750=6.14%, 1000=0.32%
    lat (msec) : 2=0.49%, 4=0.03%, 10=0.02%, 20=0.01%, 50=0.01%
    lat (msec) : 500=0.01%
  cpu          : usr=0.04%, sys=0.23%, ctx=1548182, majf=0, minf=1844
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued    : total=r=0/w=729637/d=0, short=r=0/w=0/d=0

Run status group 0 (all jobs):
  WRITE: io=2850.2MB, aggrb=9726KB/s, minb=9726KB/s, maxb=9726KB/s, mint=300048msec, maxt=300048msec

Disk stats (read/write):
    dm-0: ios=0/766673, merge=0/0, ticks=0/341324, in_queue=341316, util=86.15%, aggrios=0/731075,
aggrmerge=0/35722, aggrticks=0/268135, aggrin_queue=267337, aggrutil=85.33%
  vda: ios=0/731075, merge=0/35722, ticks=0/268135, in_queue=267337, util=85.33%
```

图 15-5　fio 测试结果

fio 也可以将测试用例写成脚本，通过运行脚本，完成多项测试任务。

```
; -- start --
[global]
[random-writers]
rw=randwrite
bs=4K
direct=1
size=30G
numjobs=63
[random-read]
rw=randread
```

1107114476@qq.com
dolphinebook.com

```
bs=4K
direct=1
size=30G
numjobs=63

; -- end --
```

fio 的 howto 参考文档对于 fio 的使用有非常详细的描述，读者可以详细阅读，连接地址如下：http: //www.bluestop.org/fio/HOWTO.txt。

（2）Iometer

1）Iometer 介绍。

Iometer（官方网址是 http: //www.iometer.org/，直接下载安装就可以使用）是一个测试系统和集群系统下的 I/O 子系统性能的工具。Iometer 既是工作负载生成器（它可以进行输入输出操作，以便增加系统的负荷），也是一个测量工具（它检查并且记录 I/O 操作的性能和对系统的影响）。它可以被配置为模拟任何程序或者基准测试程序的磁盘和网络 I/O 的负载，或者用来产生整个综合的 I/O 负载，也可以用来产生并测量单系统或者多系统（网络）的负载。

Iometer 支持测试以下内容：

- 磁盘和网络控制器性能。
- 带宽延时能力。
- 网络吞吐。
- 总线性能。
- 系统级别的磁盘性能。
- 系统级别的网络性能。

2）Iometer 使用。

笔者主要使用 Iometer 测试 Windows 系统的磁盘性能，Iometer 测试 Linux 系统 I/O 漂移比较严重，Iometer 一个非常好的功能是支持多机联合测试。在被测试的系统上执行如下命令：

```
dynamo -i local_ip -m manager_ip -n manager_name
```

可以连接到一台已经运行 Iometer 的系统，然后可以通过管理机同时给几个测试机发布测试指令。

测试时使用 1 个 Worker，Disk Targets 选择测试的磁盘，删除该盘根目录下的 iobw.tst 文件（如果已经存在），Access Specifications 按照如下配置，测试结果在 Results Display 中显示。

具体参数设置如下：

- Worker：1。
- Maximum Disk size：10240 sectors 。
- Outstanding IO：队列深度 1 ~ 128 循环加载。
- 4K 顺序读。
- 配置测试时长：5 分钟。

如图 15-6 所示为 Iometer 测试 Worker 设置，选择 Worker1，文件写到 E:。

1107114476@qq.com
dolphinebook.com

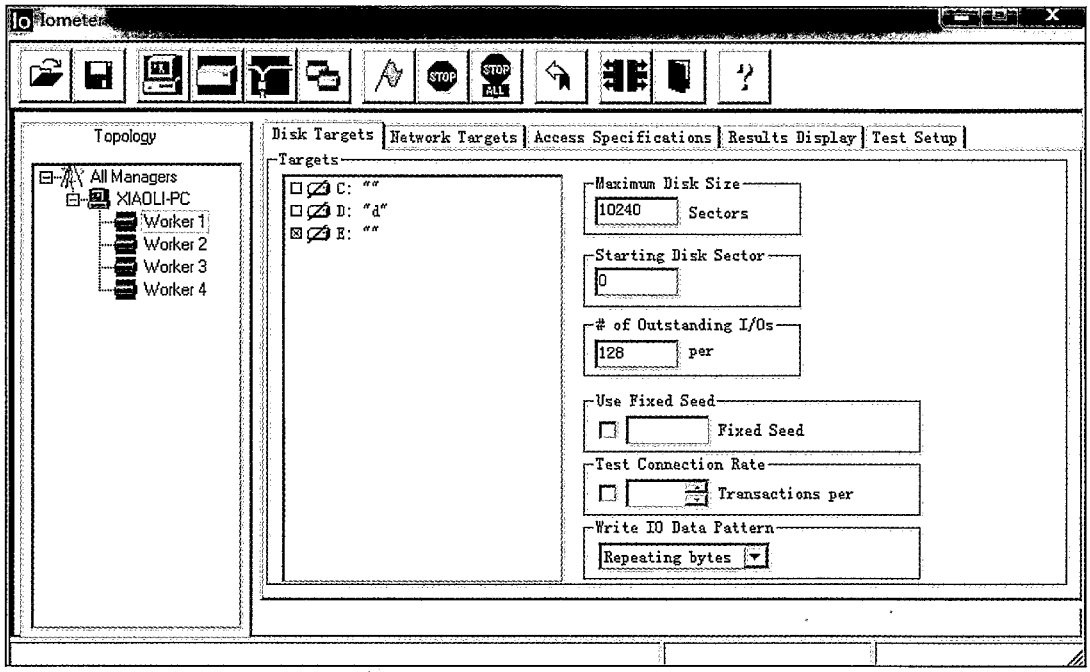


图 15-6　Iometer 测试 Worker 设置

如图 15-7 所示为 Iometer 测试内容设置，选择 4K 顺序读。

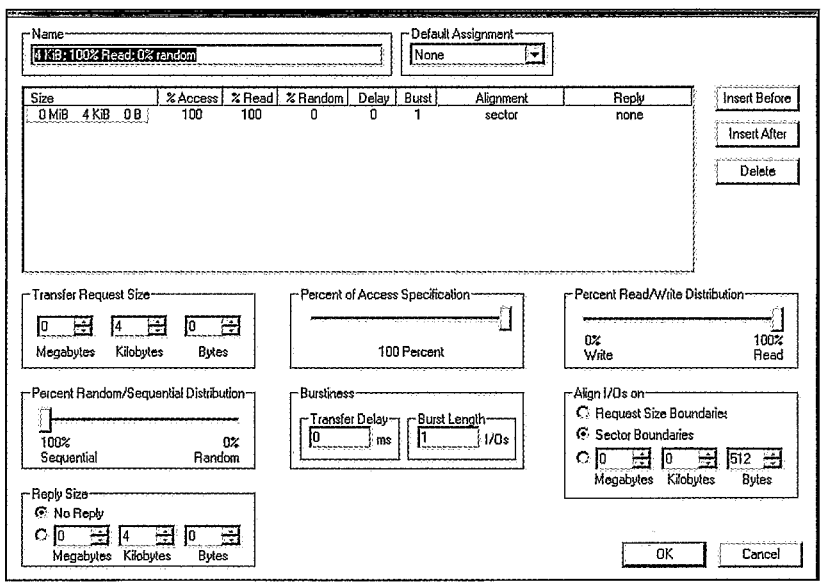


图 15-7　Iometer 测试内容设置

如图 15-8 所示为 Iometer 测试配置，选择一次跑完所有测试。

1107114476@qq.com
dolphinebook.com

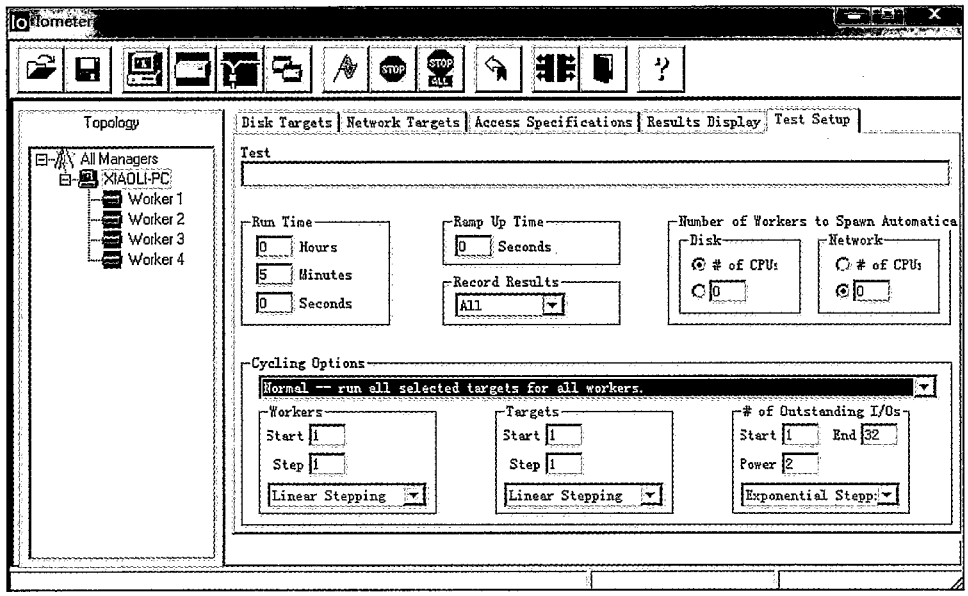


图 15-8 Iometer 测试配置

如图 15-9 所示为 Iometer 测试结果展示配置，选择每秒刷新一次。

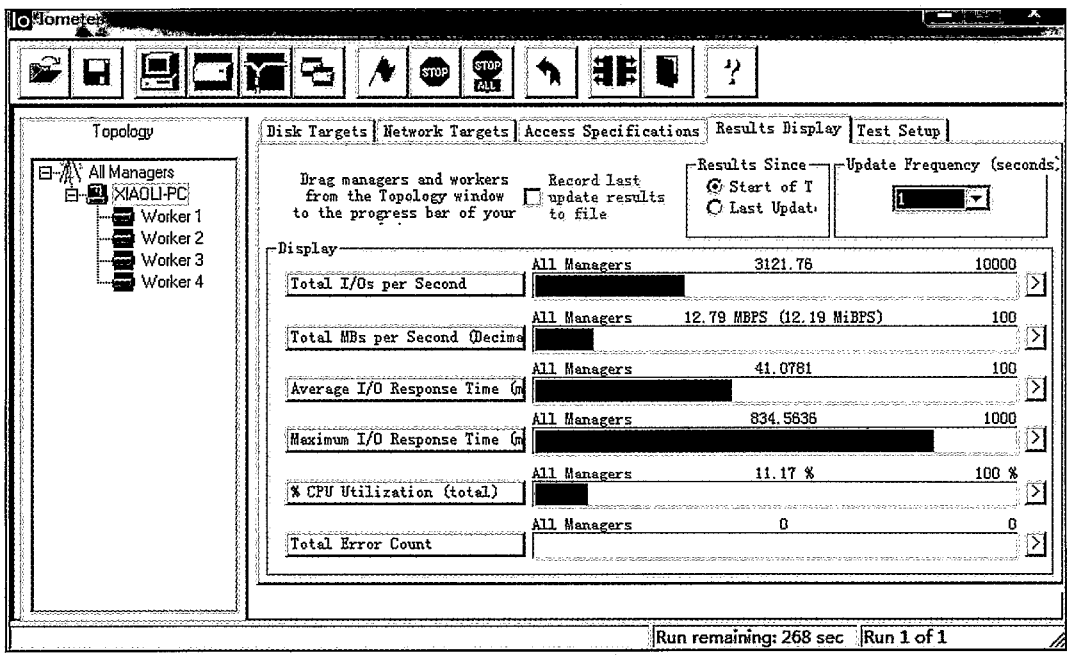


图 15-9 Iometer 测试实时结果显示

如图 15-10 所示为测试结果，为 Excel 格式。

1107114476@qq.com
dolphinebook.com

'Test Type	Test Description					
0						
'Version						
1.1.0						
'Time Stamp						
2014-12-27 16:53:37:915						
'Access specifications						
'Access specification name	default assignment					
4 KiB; 100% Read; 0% random	0					
'size	% of size	% reads	% random	delay	burst	align
4096	100	100	0	0	1	0
'End access specifications						
'Results						
'Target Type	Target Name	Access Specification Name	# Managers	# Workers	# Disks	IOps
ALL	All	4 KiB; 100% Read; 0% random	1	1	1	3268.034
MANAGER	XIAOLI-PC	4 KiB; 100% Read; 0% random		1	1	3268.034
PROCESSOR	CPU 0					
PROCESSOR	CPU 1					
PROCESSOR	CPU 2					
PROCESSOR	CPU 3					
WORKER	Worker 1	4 KiB; 100% Read; 0% random			1	3268.034
DISK	E: ""					3268.034

图 15-10　Iometer 测试结果文件

（3）IOzone

1）IOzone 介绍。

IOzone（http: //www.iozone.org/）是针对文件系统的性能测试工具。IOzone 的一个特色是测试结果可以是 Execl 格式，并且可以画出非常直观的三维测试图，如图 15-11 所示，这样磁盘 I/O 各方面的性能就非常直观，有利于找出系统的优势和瓶颈点。但美中不足的是画图的脚本是要单独收费的。

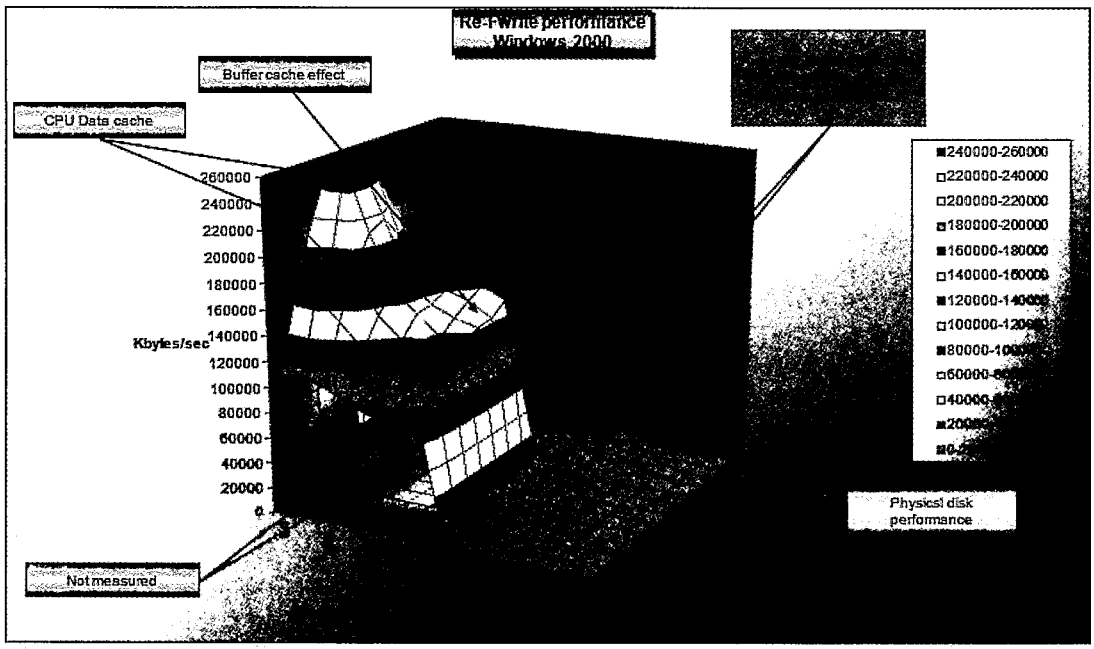


图 15-11　IOzone 测试结果的展示图

1107114476@qq.com
dolphinebook.com

2）IOzone 安装使用。

IOzone 需要通过编译的方式进行安装，不支持裸设备的测试。

```
wget http://www.iozone.org/src/current/iozone*.tar
tar  xvf  iozone*.tar
cd iozone*/src/current/
make linux-AMD64
```

IOzone 可以测试顺序和随机的文件 I/O 性能。在命令行上使用 -o 参数，可以使测试工作在 SYNC 模式上。命令输出结果中的 rewrite 指标可近似看做磁盘的 IOPS 值。

```
./iozone -i 0 -i 1 -r 4K -s 1G -o -I -f /test
```

参数说明如下：

- -i：后面加数字，表示测试内容，范围为 0 ~ 12，常用的有 3 项，0= 写或者重写，1= 读或者重读，2= 随机读写，0 选型是必需的。
- -r：测试块大小。
- -s：测试文件大小。
- -o：采用 O_sync 方式。
- -f：测试文件路径。
- -I：关闭文件系统缓存。

输出结果如下：

```
Iozone: Performance Test of File I/O
            Version $Revision: 3.429 $
            Compiled for 64 bit mode.
            Build: linux-AMD64
...
    Record Size 4 kB
    File size set to 1048576 kB
    SYNC Mode.
    O_DIRECT feature enabled
    Command line used: ./iozone -i 0 -i 1 -r 4K -s 1G -o -I -f /test
    Output is in kBytes/sec
    Time Resolution = 0.000001 seconds.
    Processor cache size set to 1024 kBytes.
    Processor cache line size set to 32 bytes.
    File stride size set to 17 * record size.
                                                        random    random
              kB  reclen    write  rewrite    read    reread    read     write
         1048576    4      1988     4885    10498    10944
```

（4）dd 命令

dd 是系统自带的命令，不需要单独安装，使用非常方便，但是只能测试顺序 I/O 的性能。采用 oflag 或者 iflag 可以指定 direct/sync/dsync 模式。例如，下面的命令测试了 4K 的块

1107114476@qq.com
dolphinebook.com

大小下文件的顺序写性能（directio 模式）。

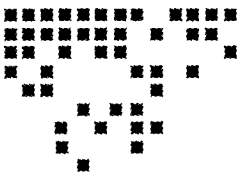
```
dd if=/dev/zero of=/foo/test.file bs=4k count=10000 oflag=direct
```

15.7　本章小结

本章介绍了宿主机 CPU、内存、磁盘、RAID 卡、网卡的选型。另外，宿主机因为可靠性要求比较高，除了选择配置，对服务器的品牌型号也要注意，尽量选择主流品牌，主流品牌和非主流品牌的稳定性从笔者的实践来看，差距比较大。即便主流品牌也要选择企业级的型号，一些入门级的服务器型号建议不要选做宿主机。

本章还介绍了宿主机的性能测试方法，性能测试方法对于虚拟机的基准测试同样适用，下一章为读者介绍虚拟机的镜像制作。

1107114476@qq.com
dolphinebook.com



Chapter 16 第 16 章

虚拟机镜像制作、配置与测试

本章为读者介绍虚拟机模板镜像的制作方法，虚拟机是通过模板镜像进行大量的复制，因此，制作镜像的目的即要求方便地进行大量复制，又要求保证复制出虚拟机的稳定性。虚拟机模板镜像的系统、驱动程序版本都要经过实际的测试，以保证通过镜像复制的虚拟机能大规模部署，高效、稳定地运行。本章向读者介绍如何制作、配置 Windows、Linux 虚拟机，以及虚拟机模板镜像的测试注意点。

16.1 Windows 虚拟机制作方法

目前，在生产环境使用的 Windows 版本主要是 Windows Server 2008 R2 及 Windows Server 2012 R2，本章以 Windows Server 2008 R 为例说明 Windows 系统镜像的制作方法，Windows Server 2012 R2 制作方法非常类似，本书就不介绍了。

本章也会简单介绍 Windows Server 2003 镜像的制作注意点。需要说明的是，最新一代服务器已经不提供对 Windows Server 2003 驱动的支持，但是虚拟化对 Windows Server 2003 支持得很好。如果有老的业务需要使用 Windows Server 2003，通过虚拟化提供支持也是很好的解决方案。

16.1.1 Windows 镜像制作

1. Windows 虚拟机的安装

（1）准备操作系统 ISO 文件

建议使用正版 Windows ISO 镜像，生产环境不要使用非正常渠道获得的 Windows ISO

1107114476@qq.com
dolphinebook.com

文件。

（2）Windows 虚拟机的配置

Windows 虚拟机建议根据表 16-1 来配置。

表 16-1　Windows 镜像资源分配

	CPU	内存大小（MB）	硬盘方式、大小（GB）	网卡方式
Windows 2003 系列	2 个	512	IDE、50	e1000
Windows 2008 系列	2 个	2048	IDE、100	Virtio
Windows 2012 系列	4 个	2048	IDE、100	Virtio

生成镜像的命令如下：

```
qemu-img create -f qcow2 sys.img -opreallocation=metadata 100G
```

为了传输管理方便，虚拟机镜像格式建议使用 qcow2 格式，要加上 opreallocation=metadata 参数，或者在 virt-install 命令参数中指明是 qcow2 格式，否则，使用 virt-install 命令安装完成后虚拟机磁盘镜像会变为 raw 格式。

1）关于 Windows 虚拟机镜像大小。关于镜像大小笔者也有一个认识过程。最开始的时候，是制作一个非常小的镜像，生成虚拟机时，通过 virt-resize 命令进行扩展。这样做的好处是因为镜像小，传输复制方便，尤其是在笔者刚开始部署虚拟化的时候，大量使用的是 Windows Server 2003 系统，镜像小有利于快速扩展。但是随着 Windows Server 2008 R2 的大量开始使用，使用扩展镜像的方法遇到两个问题：

- 扩展过程比 Windows Server 2003 慢很多。
- 因为做了镜像扩展，第二次重启的时候要做文件系统检查，Windows Server 2008 R2 的文件数目非常多，导致文件系统检查时时间很长，有时候长达几个小时。

所以，到了 Windows Server 2008 R2 时代，笔者改变做了做法，在生成虚拟机时，直接将镜像复制，不再进行磁盘和文件系统的扩展。这样做虽然牺牲了灵活性，但是生成虚拟机时速度要快很多，并且生成的虚拟机第二次重启时，没有文件系统长时间检查的问题。

笔者经过研究测试，目前国内公有云大部分也都是通过复制镜像的方式进行部署的。

2）关于 Windows 系统网卡驱动的说明。2014 年上半年以前，笔者的生产环境的 Windows 虚拟机系统网卡使用 e1000 的网卡，因为使用 Virtio 的网卡，在 Windows 系统上一直有闪断的情况。但是到 2014 年年底，最新发布的 Virtio 网卡 Windows 系统驱动，经过笔者在生产环境的检验，已经工作的非常稳定，所以，笔者又开始在生产环境使用 Virtio 网卡。

3）关于 Windows 虚拟机系统的磁盘驱动。开始安装系统的时候，建议磁盘驱动使用 IDE 方式，这样方便安装，系统安装之后，建议将驱动更换为 Virtio，因为 Virtio 性能要好很

1107114476@qq.com
dolphinebook.com

多，具体的操作方法在 16.2 节介绍。

（3）安装虚拟机

安装虚拟机有多种方法，如使用 Virt-Manager、virt-install 工具，也可以通过事先定义 xml 文件来安装虚拟机，这里笔者使用 virt-install 命令安装虚拟机：

```
virt-install \
             --hvm \
             --name windows2008 \
             --ram 2048 \
             --file=/opt/sys.img  \
             --livecd \
             --cdrom /opt/windows/windows2008.iso \
             --vnc \
             --vncport=5910
```

具体安装过程就不详细介绍了，但是 Windows Server 2008 R2 安装时候的分区需要注意，因为 2008 默认安装时会有一个 100MB 的隐藏分区，放一些系统引导文件，类似 Linux 系统的 boot 分区，但是这个分区在调整分区和文件系统的时候经常容易误导，如果不希望有这个分区，可以手工分区，方法如下：

在光盘引导起来之后，按 Shift+F10 组合键，会出现一个命令行界面，输入 diskpart 命令，进入 Windows 的命令行分区模式，diskpart 类似 Linux 下的 partd 命令，如图 16-1 所示。

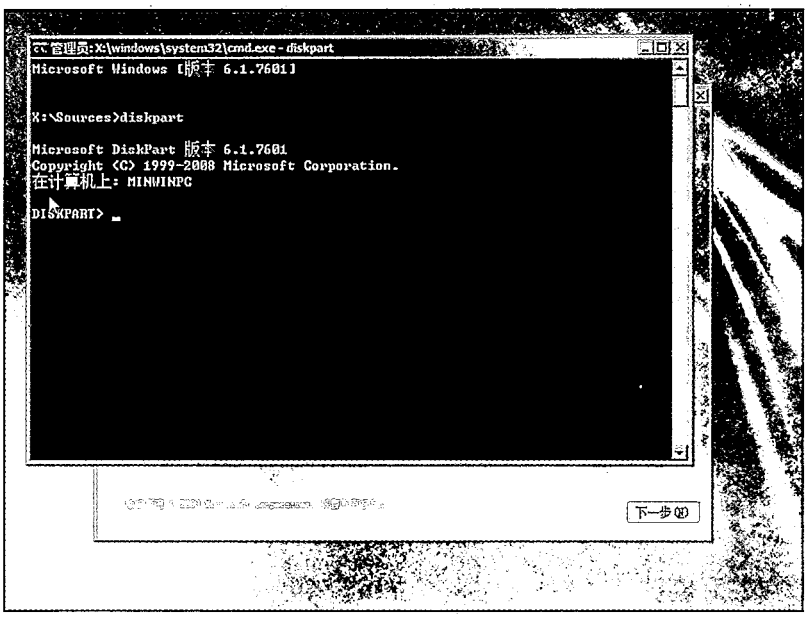


图 16-1 进入 diskpart 命令界面

输入 list disk 命令，可以看到当前的磁盘，输入 select disk 0 命令，选择第一块磁盘，如

1107114476@qq.com
dolphinebook.com

图 16-2 所示。

输入 create partition primary 命令，将所有磁盘空间划分为一个大的主分区，输入 active 命令，将这个分区激活，如图 16-3 所示。

图 16-2　使用 diskpart 命令选择磁盘

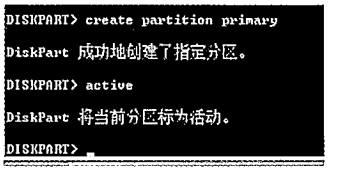


图 16-3　使用 diskpart 命令划分磁盘分区

要想了解 diskpart 命令的详细使用可以输入 help 命令，查看帮助。分区完成输入 exit 退出命令行界面，然后开始 Windows 系统的安装，到磁盘分区的步骤时，如图 16-4 所示，直接选择刚才手工的分区安装就可以。

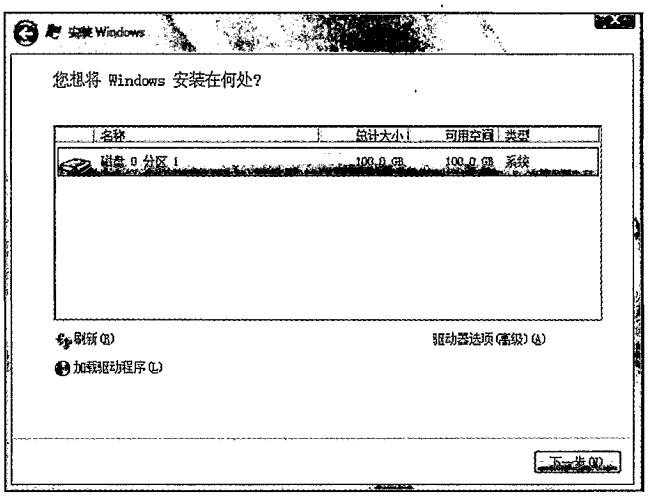


图 16-4　Windows 安装选择分区

虚拟机安装完成后，重要的就是如何配置虚拟机系统，下面介绍如何让制作的虚拟机符合自己的业务需求。

2. 配置虚拟机模板

（1）安装 Virtio 驱动

KVM 是使用硬件虚拟化辅助技术（如 Intel VT-x、AMD-V）的虚拟化引擎，在 CPU 运行效率方面有硬件支持，效率是比较高的。KVM 在 I/O 虚拟化方面，传统的方式是使用 QEMU 纯软件的方式来模拟 I/O 设备，其效率并不高。在 KVM 中，可以在虚拟机中使用半

1107114476@qq.com
dolphinebook.com

虚拟化驱动（Paravirtualized Drivers，PV Drivers）来提高 I/O 性能，因此，在虚拟机的磁盘、网络尽量应使用 Virtio 设备。

1）安装硬盘 Virtio 驱动。

首先，下载硬盘 Virtio 驱动，下载地址为：http://www.linux-kvm.org/page/WindowsGuestDrivers/Download_Drivers。

然后，新增一块临时硬盘，并将其驱动更新为 SCSI 模式：

```
qemu-img create -f qcow2 tmp.img 1G
virsh attach-disk vm --source /kvm/tmp.img --target vdb --persistent
```

通过热插拔方式动态增加一块硬盘后，会在资源管理器里提示发现新的硬盘。此时，将下载好的 virto 驱动挂载到虚拟机。打开后会发现其包括多个文件夹，其中几个文件夹对应的系统是 wnet 为 Windows 2003 Server，wlh 为 Windows Server 2008。按照目前使用的 Windows 系统，安装相应目录中的驱动就行了。

安装完成后，还需要更新虚拟机 xml 文件中有关系统盘的配置。使用 virsh edit vmname 命令，进入虚拟机 xml 编辑模式，找到如下代码行：

```
<source file='/kvm/sys.img'/>
<target dev='hda' bus='ide'/>
```

将其修改为：

```
<source file='/kvm/sys.img'/>
<target dev='vda' bus='virtio'/>
```

参照上面的配置修改其他硬盘的 xml 部分，改完后关闭虚拟机并启动。在设备管理器会发现原硬盘项已变成 SCSI。

2）安装网卡 Virtio 驱动。

首先，手动通过热插拔方式增加一块临时网卡：

```
virsh attach-interface vm --type bridge --source br1 --model virtio
```

在设备管理器中可以看到新加的网卡，然后为新的网卡安装 Virtio 驱动即可，方法同 Virtio 磁盘的安装方法。新增网卡的 xml 配置如下：

```
<interface type='bridge'>
 <mac address='32:45:11:sf:5c:3d'/>
 <source bridge='br1'/>
 <target dev='vnet3'/>
 <model type='Virtio'/>
 <alias name='net2'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
</interface>
```

在安装完网卡 Virtio 驱动后，参照上面的配置修改其他网卡的 xml 文件，重启虚拟机

1107114476@qq.com
dolphinebook.com

后，在设备管理器中会要求重新安装驱动，安装即可。

（2）取消登录任务配置及清除任务栏记录

Windows Server 2008 R2 安装完成后，默认会启动初始任务配置及服务器管理器，如图 16-5 和图 16-6 所示，为了避免打扰用户，勾选两个界面的“登录时不显示此窗口”复选框。

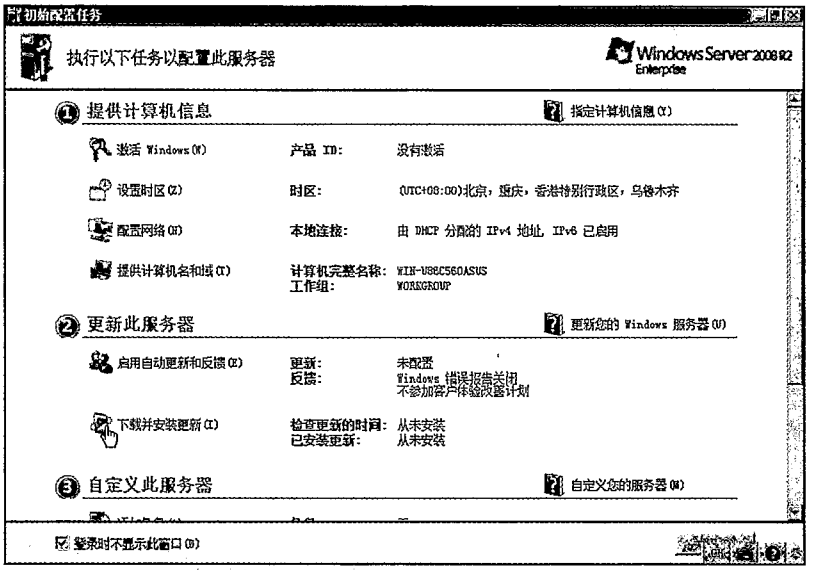


图 16-5　初始配置任务

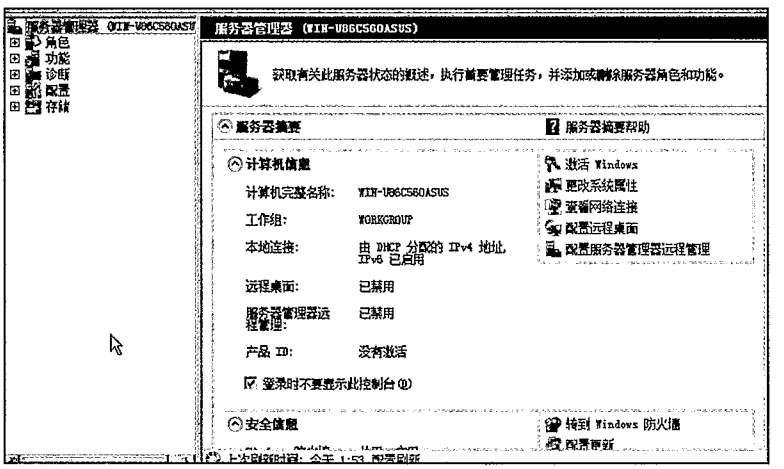


图 16-6　服务器管理器

为了让新建的虚拟机菜单栏没有残留的程序痕迹，建议删除菜单的历史程序启动记录，

1107114476@qq.com
dolphinebook.com

操作方法为，右击“开始”按钮，在弹出的快捷菜单中选择“属性”命令弹出“任务栏和「开始」菜单属性”对话框，选择“「开始」菜单”选项卡，取消勾选“隐私”复选框，单击“应用”按钮，再勾选两个选项，在应用一次，如图 16-7 所示。

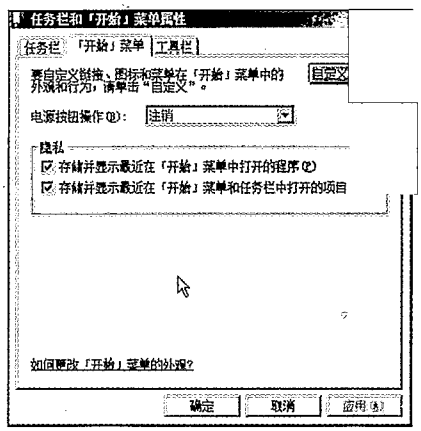


图 16-7 Windows Server 2008 R2 菜单属性

（3）配置系统更新源

为了提高同一数据中心 Windows 系统补丁更新速度，节省带宽，可以统一配置所有 Windows 虚拟机到数据中心内部的更新源进行补丁更新。

1）配置更新策略。操作方法是，同时按 Windows 键和 R 键，在弹出的“运行”对话框中输入 gpedit.msc 并按回车键，然后会启动组策略编辑器，如图 16-8 所示，在组策略编辑器中依次选择“计算机配置”→“管理模板”→“Windows 组件”→“Windows Update”，在右边的界面双击配置“自动更新”，选择“已启用”单选按钮，配置自动更新选择“2– 通知下载并通知安装”，还可以根据需要定义更新的时间。

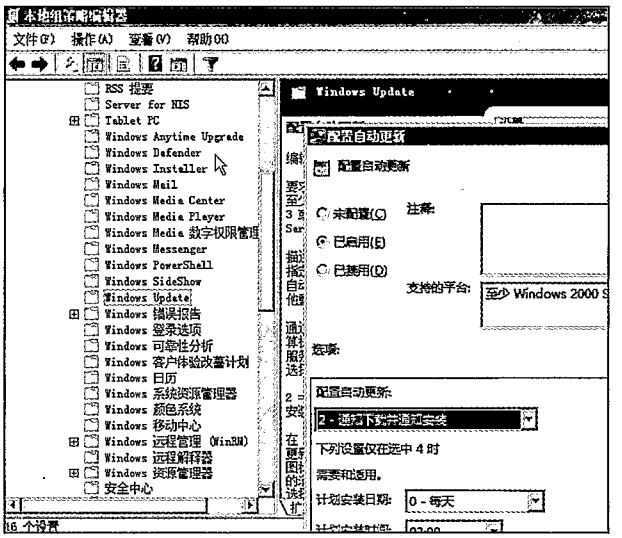


图 16-8 自动更新属性

2）配置更新源。还是上一步的 Windows Update 组策略，双击右边窗格中的“指定 Intranet Microsoft 更新位置”，如图 16-9 所示，如果选择“未配置”单选按钮，此时系统将通过连接微软站点自动更新。在生产环境中一般笔者都会搭建自动安装服务器（WSUS）来实现加快更新速度和节省带宽的目的。

1107114476@qq.com
dolphinebook.com

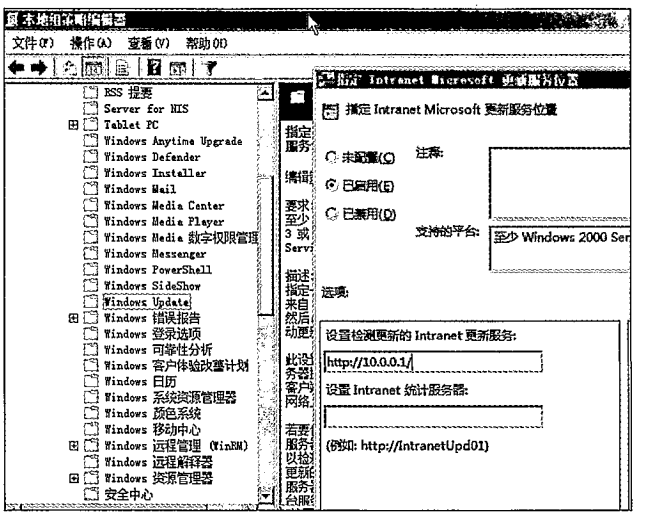


图 16-9　Intranet Microsoft 更新服务位置属性

（4）配置性能计数器

在系统遇到性能问题时，可以通过性能计数器收集到的数据方便地找出瓶颈所在。建议读者根据自己业务的需要，添加相应的性能计数器。

添加性能计数器的操作方法如下：

1）同时按 Windows 键加 R 键，在弹出的“运行”对话框中输入命令：“ %windir%\system32\perfmon.msc /s”并按回车键，打开“性能监视器”对话框。

2）选择“数据收集器集”中的“用户定义”选项，右击，在弹出的快捷菜单中选择“新建”命令，打开“创建新的数据收集器集”对话框，选择“手动创建（高级）”单选按钮，如图 16-10 所示。

3）单击“下一步”按钮，选择“创建数据日志”单选按钮并勾选“性能计数器”复选框，如图 16-11 所示。

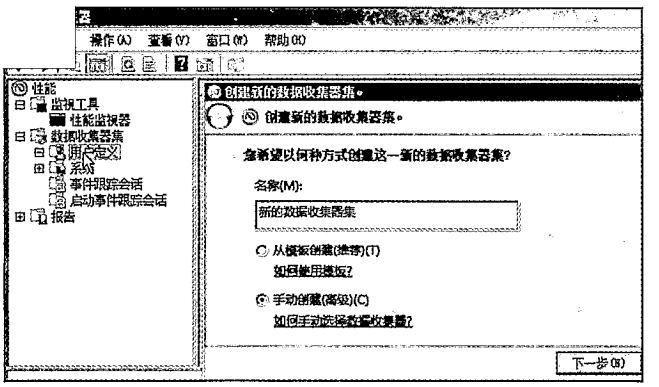


图 16-10　创建新的数据收集器集

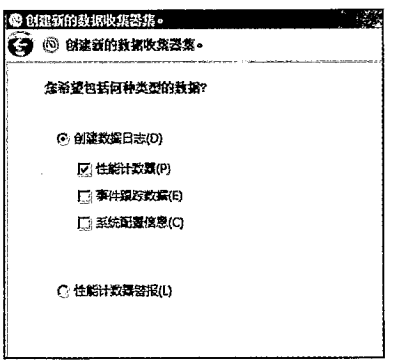


图 16-11　选择“性能计数器”

1107114476@qq.com
dolphinebook.com

4）根据需要添加响应的计数器选项，如图 16-12 所示，建议 CPU、磁盘、网络、内存这些基本性能选项都添加上，然后读者可以根据自己的业务需要，再添加相应的性能计数器。

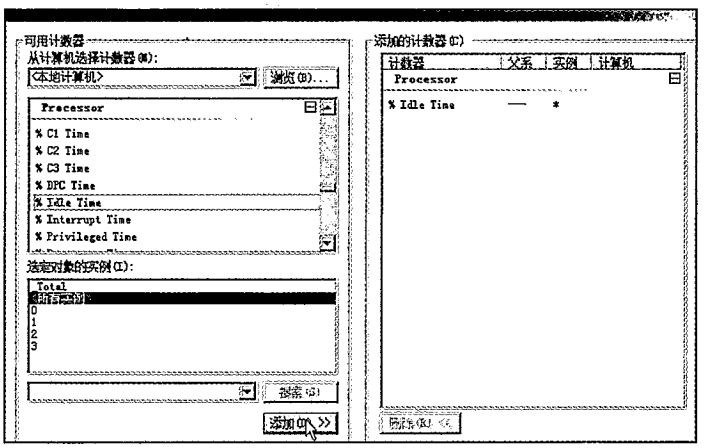


图 16-12 添加性能计数器

5）在下一步的“日志格式”选项组中选择“逗号分隔”的格式，即 csv 格式，这种格式方便脚本或者电子表格分析，如图 16-13 所示。

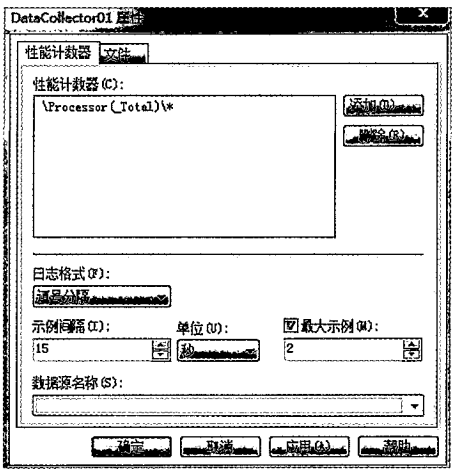


图 16-13 性能计数器日志格式

（5）添加 SNMP 组件

很多监控工具都需要通过 SNMP 来获取监控数据，建议安装 SNMP 组件。

安装方法如下：在控制面板中选择“程序和功能”，在打开的窗口中选择“打开或关闭 Windows 功能”，然后在“功能”选项区域中单击“添加功能”按钮，如图 16-14 所示。

1107114476@qq.com
dolphinebook.com

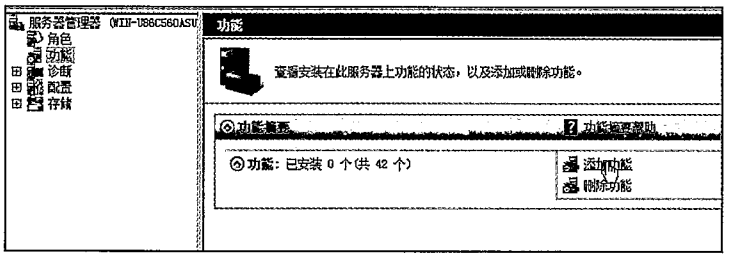


图 16-14　添加功能

如图 16-15 所示，在打开的对话框中选择 SNMP 组件，然后按照提示安装就可以。

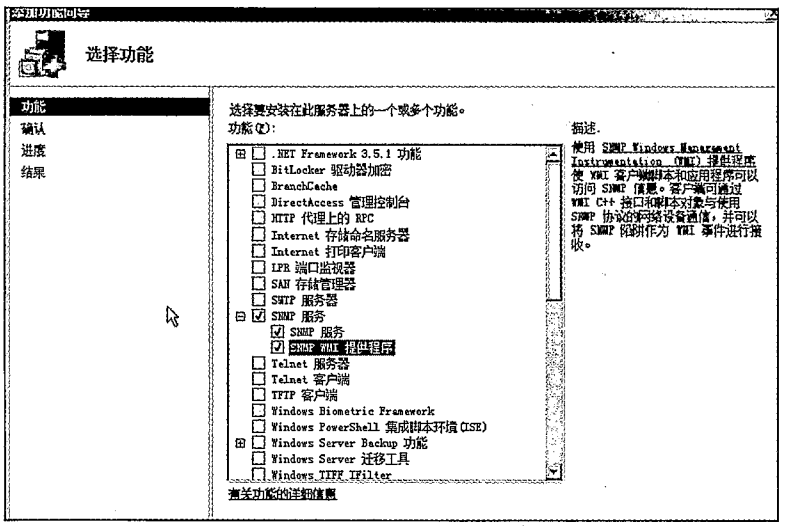


图 16-15　添加 SNMP 组件

（6）计算机属性设置

1）视觉性能优化。右击桌面，在弹出的快捷菜单中选择“属性”命令，然后选择“高级系统设置”选项，在打开的窗口中选择“高级”选项卡，在“性能”选项区域单击“设置”按钮，弹出“性能选项”对话框，选择“视觉效果”选项卡，然后选择“调整为最佳性能”单选按钮，如图 16-16 所示。配置这一步的原因是因为服务器系统主要是提供服务，更注重性能。

2）配置数据保护。接上一步，在图 16-16 中选择“数据执行保护”选项卡，然后选择“仅为基本 Windows 程序和服务启动 DEP”单选按钮，如图 16-17 所示，这么操作目的是因为有些程序和 DEP 有兼容问题，会造成程序不能运行。

提示　DEP（Data Execution Prevention，数据执行保护），是一套软硬件技术，能够在内存上执行额外检查以帮助防止在系统上运行恶意代码。笔者在生产环境多次遇到因为 DEP 程序不能正常运行的情况。

1107114476@qq.com
dolphinebook.com

图 16-16 视觉性能优化

3）打开远程桌面。连接 Windows 虚拟机的主要方式是通过远程桌面，所以需要打开远程桌面，操作方法如下：在计算机上右击，在弹出的快捷菜单中选择“属性”命令，在弹出的对话框中选择“远程”选项卡，如图 16-18 所示，选择“允许运行任意版本远程桌面的计算机连接（较不安全）”单选按钮，这个选项兼容性比较好，使用 Linux 的 rdesktop 套件都可以连接 Windows 计算机。

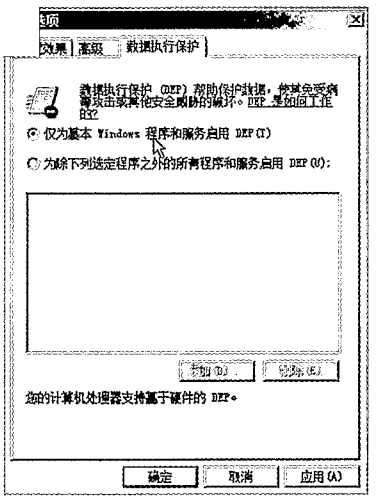


图 16-17 配置数据保护（一）

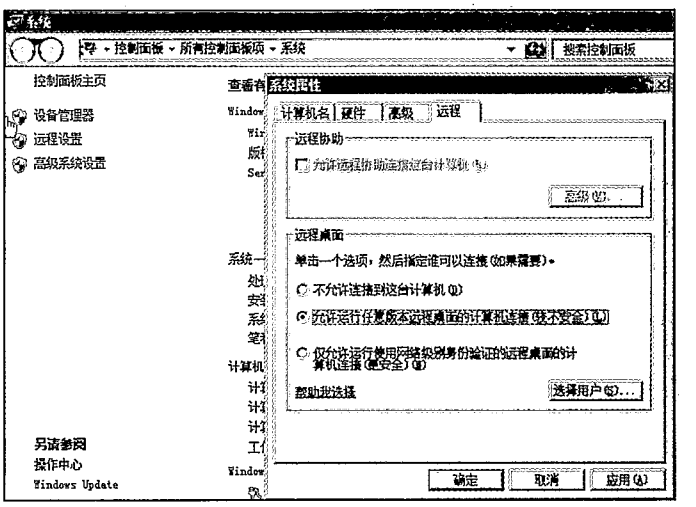


图 16-18 配置数据保护（二）

（7）通过组策略配置计算机启动执行脚本

在初始化虚拟机的过程中，用户可能需要做一些初始化配置，如用户程序和配置文件的

1107114476@qq.com
dolphinebook.com

个性化设置，操作步骤如下：

- 定制脚本名称、配置文件路径规范。
- 在虚拟机生成时，通过 virt-copy-in 命令将程序、相关配置文件、脚本复制到虚拟机中。
- 在组策略中指定开机运行脚本（如果是 Linux 系统则是通过 rc.local 指定开机运行脚本）。
- 虚拟机第一次启动执行脚本，完成配置。

提示　如何利用虚拟化进一步实现业务的自动化部署和个性化配置，将以上操作和虚拟化管理平台、业务配置平台结合，结合合适的业务逻辑步骤，就可以很好地实现业务从虚拟机生成、配置、业务运行全自动化的运维。

配置组策略开机运行脚本的操作步骤如下：在开始菜单的运行对话框中输入“ gpedit.msc”命令回车，打开组策略配置窗口。然后在组策略的“本地计算机策略”下面的“计算机配置”选项中，点开“ Windows 设置”，选择“脚本（启动 / 关机）”选项，打开启动脚本配置界面，然后单击“添加”按钮，按照提示将相关脚本添加到开机启动中，如图 16-19 所示：

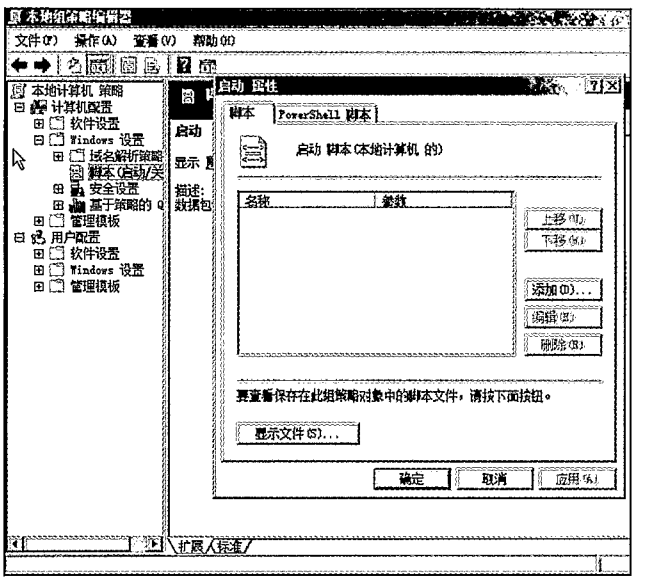


图 16-19　组策略编辑器——开机启动脚本对话框

（8）配置允许系统在未登录的情况下关闭

在宿主机维护时，因为虚拟化管理员往往没有虚拟机系统的密码，又希望系统在未登录

1107114476@qq.com
dolphinebook.com

的情况下能正常关机，修改组策略 Windows 安全选项的“关机：允许系统在未登录的情况下关闭”可以实现这个目的，如图 16-20 所示。

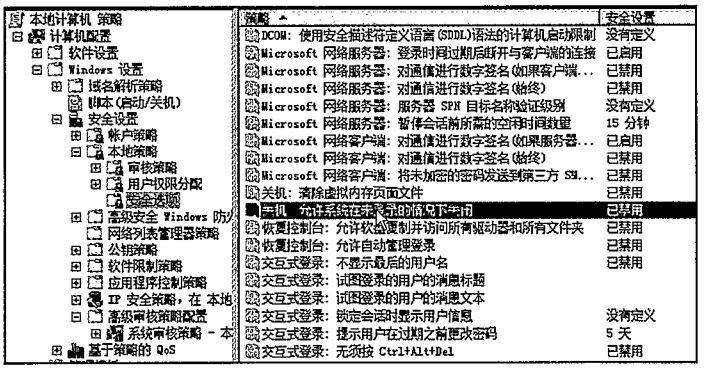


图 16-20 组策略编辑器——未登录关机对话框

（9）配置 Windows 防火墙

Windows Server 2008 R2 的防火墙默认全部禁止数据包进入，初始配置时，因为用户需要远程连接，建议开启远程桌面，为了方便判断虚拟机的状态，建议开启 ICMP，操作方法如下：在控制面板中选择“ Windows 防火墙”，然后选择“高级设置”，在“入站规则”中，右击，选择“新建规则”，如图 16-21 所示，选择“自定义”单选按钮：

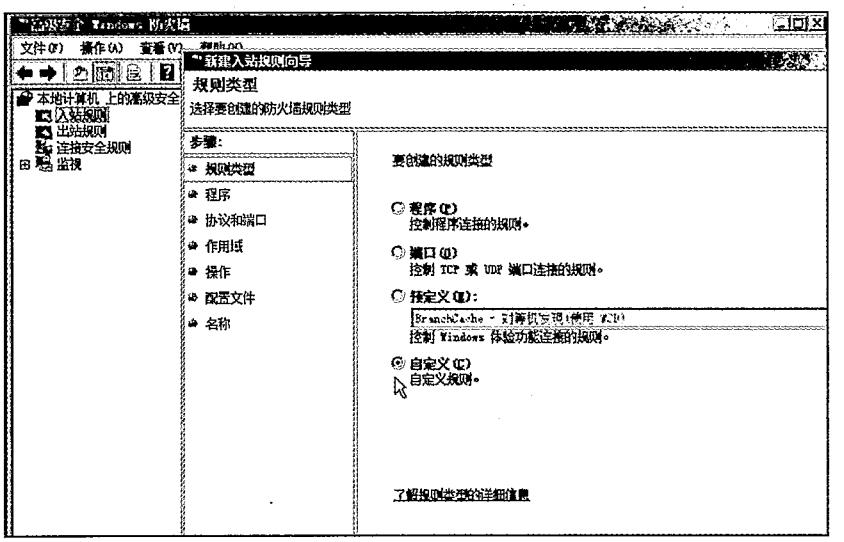


图 16-21 新建防火墙入站规则

下一步如图 16-22 所示，设置“协议类型”为“ TCP”，在“本地端口”下拉列表框中选择“特定端口”选项，输入端口号“3389”，然后按照提示完成操作即可。

1107114476@qq.com
dolphinebook.com

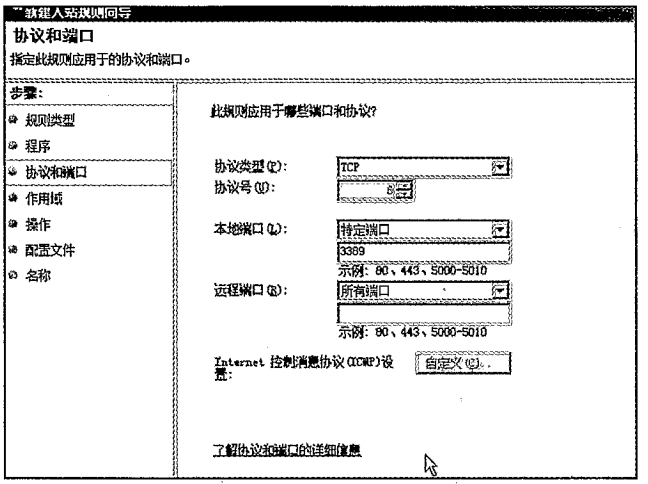


图 16-22　配置防护墙协议和端口

同样，在新建一个规则，运行 ICMP，如图 16-23 所示，协议选择 ICMPv4，其他按照提示操作即可。

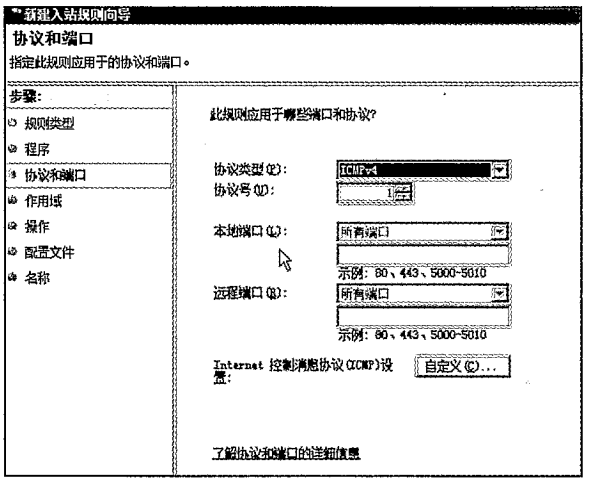


图 16-23　配置防火墙 ICMP

（10）配置精确时钟和 NTP 服务器

虚拟机都有时间漂移的现象，Windows 虚拟机需要配置精确时钟和 NTP 服务器。

1）精确时钟的配置方法。Windows Server 2003 开启精确时钟设置方法如下：修改 c:\boot.ini，在启动行最后增加 /usepmtimer 参数。另外，有时需要进入安全方式进行配置，为了方便 Windows Server 2003 进入安全方式，在 c:\boot.ini 中增加一行以设置进入安全方式，修改后的 boot.ini 如下：

1107114476@qq.com
dolphinebook.com

```
[boot loader]
rem 启动菜单时间为 5 秒钟
timeout=5
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
rem 正常启动菜单
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows Server 2003, Enterprise" /
noexecute=optout /fastdetect /usepmtimer
rem 安全方式启动菜单
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Safe Mode" /safeboot:minimal /
fastdetect /usepmtimer
```

Windows Server 2008 R2 配置精确时钟的方法为，在命令行方式下运行以下命令：

```
C:\Windows\system32>bcdedit /set {default} USEPLATFORMCLOCK on
```

2）NTP 服务器配置。在桌面右下角的时间上右击，在弹出的快捷菜单中选择“更改时间和日期”命令，在弹出的对话框中选择“Internet 时间”选项卡，输入自己的 NTP 服务器地址，如图 16-24 所示。

图 16-24 配置 NTP 服务器

（11）IE 配置

1）将 IE 浏览器主页设置为空白页面，并将 IE 浏览器中的历史记录删除。

2）关闭 IE 浏览器的增强安全配置。

操作方法：在服务器管理器中选择“配置 IE ESC”，如图 16-25 所示。

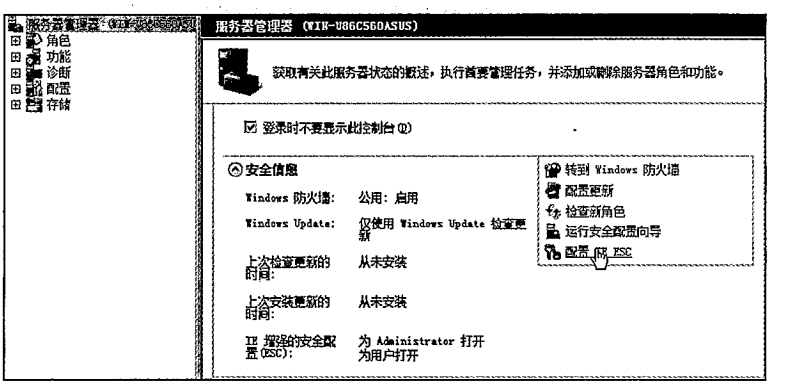


图 16-25 服务器管理器中的“配置 IE ESC”

将“管理员”和“用户”的 IE 增强的安全配置禁用，如图 16-26 所示。

1107114476@qq.com
dolphinebook.com

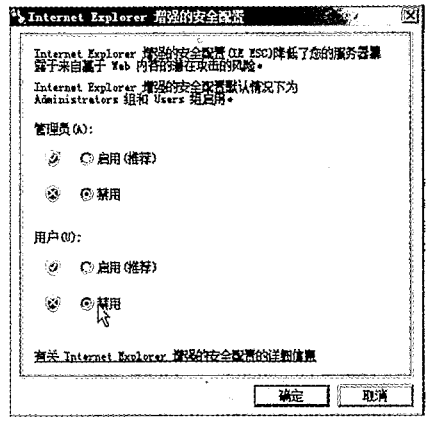


图 16-26　IE 增强的安全配置

（12）网络配置

为了安全起见，关闭 Windows 的网络共享和 WINS 解析，如图 16-27 所示，关闭网络配置中的 IPv6、微软网络客户端、文件和打印机共享选型。如图 16-28 所示，禁用 TCP/IP 中的 NetBIOS。

图 16-27　网络属性配置

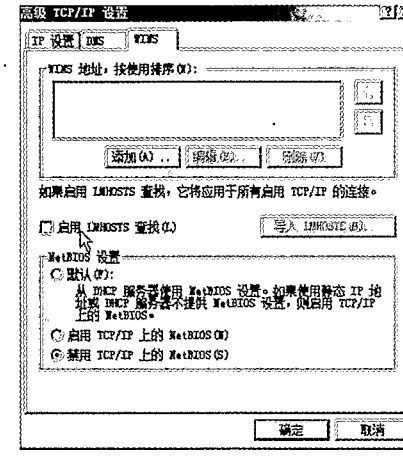


图 16-28　TCP/IP WINS 配置

16.1.2　Windows 虚拟机 sysprep 初始化封装

镜像制作中，一般安装好系统，并设定好所有标准化选项，然后进行系统封装，以便通过镜像部署的虚拟机在首次启动时自动更新 SID、配置、主机名。如果不更新 SID，通过模板部署的所有虚拟机 SID 都一样，这些虚拟机如果要加入 Windows 域就会有问题。所以，为了保险起见，在制作镜像过程中，最好进行镜像的封装，下面介绍 Windows 封装的方法。

1107114476@qq.com
dolphinebook.com

提示 SID 是每个 Windows 系统的唯一标识符，根据笔者的生产环境的经验，如果 Windows 系统不加入域，不进行系统封装也可以正常稳定地运行。但是，这样复制出来的虚拟机会出现主机名同名的问题，解决方法是通过初始化脚本实现主机名随机修改。
另外，Windows Server 2003 也可以通过 newsid 工具进行修改，Windows Server 2008 R2 如果要修改 SID 必须使用微软的封装方法。

1. Windows Server 2003 的封装

Windows Server 2003 的封装方法如下：

① 在 Windows Server 2003 系统安装光盘中找到 SUPPORT 目录下的 TOOLS 文件夹，将 DEPLOY.CAB 内的所有文件解压到到 C 盘 Sysprep 文件夹中。

② 运行其中的 sysprep.exe，在下一步的封装主界面中选择“重新封装”选项。

③ 10 ~ 20 秒封装准备完成后，系统自动关机。

通过以上步骤，可以把封装后的虚拟机转化为模板，然后就可以通过此模板部署虚拟机了。以上封装好的系统，首次启动会有对话框要求用户提供必要设定信息，如主机名、区域语言设定等。对于批量部署虚拟机，每台都手动输入非常不方便，为了解决这个问题，微软提供自动应答文件的方案，以便封装好的系统首次启动自动运行应对文件，不需要用户手动输入信息，具体设定步骤如下。

在运行封装之前，首先，运行 sysprep 文件夹下的 setupmgr.exe，安装向导会指导一步步完成自动应答文件的生成，过程很简单，笔者就不详细介绍了。

全部输入完毕后，结束向导，会在 sysprep 目录下生成一个名称为 sysprep.inf 的应答文件。自动应答文件生成后，再运行 sysprep.exe 进行封装就可以了，通过镜像复制的系统首次运行就可以自动完成配置。

2. Windows 2008 R2 封装

Windows Server 2008 的封装工具位于 C:\Windows\System32\sysprep 目录下，不需要安装光盘。

双击运行其中的 sysperp，选择“ OOBE”和下面的“通用”，确定后开始封装，结束后会自动关机。

Windows Server 2008 R2 也可以制作自动应答文件，在微软网站下载相关的工具，工具的下载参考 7.3.1 节，Windows Server 2008 的自动应答文件为 xml 格式，配置方法不在本书的讨论范围内，请读者参考微软官网的相关文档，本书就不作详细介绍了。

16.2 Linux 镜像制作方法

16.2.1 RHEL/CentOS 镜像制作方法

首先进行系统的安装，安装过程就不详细介绍了，有以下几个注意点。

1107114476@qq.com
dolphinebook.com

（1）关于镜像大小

根据经验，一般 Linux 系统盘 20GB 大小比较合适。但也不是绝对的，要根据业务类型来决定，如在生产环境中，根据业务需要，有的 Linux 系统镜像模板大小为 150GB。

（2）关于分区大小

分区建议使用自定义方式安装，boot、swap、根分区的配置经验如下。

1）boot 分区大小配置。

boot 分区建议按照表 16-2 进行配置。

表 16-2　boot 分区大小配置

系　　统	boot 分区大小（MB）
RHEL/CentOS 5 系列	100
RHEL/CentOS 6 系列	200
RHEL/CentOS 7 系列	500

2）swap 分区配置。

swap 一般和内存大小一致，或者安装的时候不创建 swap 分区，安装完成后使用文件的格式。

swap 文件分区操作方法如下：

① 创建 swap。

```
# fallocate -l 512M /swapfile
```

② 启用 swap。

```
# chmod 600 /swapfile
# mkswap -f /swapfile
# swapon /swapfile
```

③ 验证 swap 是否生效，大小是否符合预期。

```
# swapon -s
# free -m
```

④ 修改 fstab，使 swap 开机自动挂载。添加以下内容：

```
/swapfile none swap defaults 0 0
```

3）关于根分区大小和业务数据存储。

建议剩余空间全部留给根分区，业务数据存储需另外挂载一块磁盘，根据业务需求配置大小。

（3）系统升级

这步很重要，升级系统可以提升虚拟化的转换效率，使用较新的 Virtio 驱动，命令如下：

```
# yum update -y
```

1107114476@qq.com
dolphinebook.com

（4）删除旧的内核

为了节约空间，删除旧的内核，并修改启动菜单不用的内核。

（5）安装基础组件和开发组件

基础组件 Base 和开发组件一般系统都需要使用，建议安装。

（6）yum 配置

建议搭建内部 yum 源，其优点如下：

- 更新速度快。
- 节省带宽。
- 可以将自己制作的 rpm 包放入 yum 源中。

如果有自定义的 yum 源，在模板镜像中添加自建的 yum 源配置。

（7）配置 NTP

可以配置外网的 NTP 地址，也可以自己搭建 NTP 服务，以外网 NTP 为例。

将 /etc/ntp.conf 中的 NTP 服务器更换为亚洲的 NTP 源，命令如下：

```
# vim /etc/ntp.conf
server 0.centos.pool.ntp.org
server 1.centos.pool.ntp.org
server 2.centos.pool.ntp.org
```

修改如下：

```
server 0.asia.pool.ntp.org
server 1.asia.pool.ntp.org
server 2.asia.pool.ntp.org
```

开启 NTP 服务、配置 NTP 服务开机运行，命令如下：

```
# service ntpd start
# chkconfig ntpd on
```

（8）关闭 SELinux

SELinux 开启的情况下，许多程序都要进行特殊配置，一般生产环境建议关闭，方法为修改 /etc/selinux/config，将其中的 SELINUX=enforcing 修改为 SELINUX=disabled。

16.2.2 Ubuntu、Debian 虚拟机配置注意点

Ubuntu、Debian 的虚拟机安装配置和 CentOS 类似，这里只介绍以下需要注意的地方：

- 因为 Ubuntu 系统的升级比较激进，尽量选择长支持的版本和内核。笔者在生产环境中碰到多次 Ubuntu 系统使用较新内核，系统崩溃的问题。
- 使用手工分区，不要使用 lvm 方式。笔者碰到过多次使用 lvm 方式，重启系统的时候文件系统检查不通过的情况。
- 按照 16.4 节的方法，进行严格的性能和稳定性测试。

1107114476@qq.com
dolphinebook.com

16.3　虚拟机自动配置 IP 的实现办法

在生产环境中，经常需要批量的生成虚拟机，并且希望能够自动的完成虚拟机的 IP 配置，而且虚拟机的 IP 能够按照希望的规则配置上去，本节详细介绍如何自动化的批量配置虚拟机 IP 方案。

16.3.1　通过 DHCP 给虚拟机配置 IP

DHCP 的方法是一个传统方案，可以专门架设一台或者一组 DHCP 服务器，通过 DHCP 服务分发 IP，如果需要也可以控制特定的 MAC 地址配置特定的 IP。

DHCP 方案的缺点如下：

- 变更 IP 的时候操作麻烦，并且有生效周期。
- 要精确控制 MAC 和 IP 的对应关系，需要频繁修改 DHCP 配置文件。
- 有一定的几率虚拟机不能获得 IP。

所以，DHCP 方案在生产环境使用关键的问题是要精细的控制 DHCP 服务。

16.3.2　通过 Libvirt 实现虚拟机自动配置 IP

Libvirt 的网络资源池支持虚拟机的自动配置 IP，IP 资源池的 xml 配置文件如下：

```
<ip address="192.168.122.1" netmask="255.255.255.0">
<dhcp>
<range start="192.168.122.128" end="192.168.122.254" />
<host mac="00:16:3e:77:e2:ed" name="foo.example.com" ip="192.168.122.10" />
<host mac="00:16:3e:3e:a9:1a" name="bar.example.com" ip="192.168.122.11" />
</dhcp>
```

这样，可以对特定的虚拟机实现 IP 按照 MAC 配置。

16.3.3　笔者生产环境虚拟机 IP 自动配置的方法

（1）实现虚拟机 IP 自动配置的原理

在生产环境中，实现虚拟机 IP 自动配置的原理如下：

- 虚拟机在生成的过程中，通过管理工具从 IP 资源池中随机选择一个空闲的 IP 及相关掩码、网关信息放到指定的文件中。
- 通过脚本读取此文件并生成网卡的配置。
- 通过 virt-copy-in 将配置文件复制到虚拟机。
- 虚拟机启动后运行配置脚本，实现 IP 自动配置。

（2）方案技术难点和应对办法

这个方案的难点如下：

- 需要随机生成 MAC 地址，并保证 MAC 地址不重复。

1107114476@qq.com
dolphinebook.com

❑ 保证虚拟机 MAC 和 IP 地址的严格对应关系。

MAC 地址随机生成，可以使用 virtinst 工具的 API 随机生成 MAC，具体如下：

```
import virtinst.util
MAC1=virtinst.util.randomMAC("qemu")
MAC2=virtinst.util.randomMAC("qemu")
```


virtinst 工具是一套 Python 写基于 Libvirt 的虚拟机安装工具，github 地址是 https://github.com/rlaager/python-virtinst。

保证虚拟机 MAC 和 IP 地址的严格对应关系解决办法如下：

❑ 随机生成 MAC 地址，并赋值给变量。

❑ 通过 virt-install 命令生成虚拟机，并将 MAC 地址变量作为参数，使生成的虚拟机 MAC 地址作为指定的地址。

❑ 将 IP 信息传给虚拟机内部的 IP 配置脚本，在虚拟机第一次启动的时候自动运行脚本，完成 IP 配置。

以上功能通过简单的脚本可以完成，Windows 和 Linux 系统虚拟机的 IP 配置脚本是一个难点，下面介绍下 Windows 和 Linux 系统虚拟机的 IP 配置脚本。

1. Windows 虚拟机 IP 地址自动配置的方法

Windows 系统可以将 IP 的配置文件在虚拟机生成的时候，通过 virt-copy-in 复制到虚拟机中，这样，在虚拟机启动的时候，通过执行一个 vbs 配置脚本，读取配置文件，就可以完成虚拟机 IP 配置。

首先，配置组策略，让虚拟机启动的时候运行 setnicip.vbs 脚本，组策略的配置方案已经在 16.1.1 节中介绍过，通过组策略配置系统在启动的时候运行 C:\windows\setnicip.vbs 脚本。

在 C:\windows 目录下编辑生成 setnicip.vbs 文件，setnicip.vbs 内容如下：

```
'Option Explicit
call auto_setup_ip()
'读取配置文件，并根据配置文件设置 IP 地址，如果配置文件第一行 ipsetup 为 1 就直接退出
'注意目标是设置一个标志，如果配置过 IP，就不在重新配置 IP
sub auto_setup_ip
Const ForReading=1,ForWriting=2,ForAppending=8
Dim   fso,file,msg,logfile
Dim   WshShell,Path_init_file,Path_log_file
Dim   mac(10),ip(10),mask(10),gw(10)
Dim   k,m
Dim   ifrun
Dim   i,j,x,y
Dim   nicfile(20)
Dim   objFSO,logFSO
```

1107114476@qq.com
dolphinebook.com

```
Dim    rtime
Dim    name,reval

i=0
j=1
x=0
y=0
Set    WshShell=WScript.CreateObject( "WScript.Shell")  '得到程序所在的当前路径

Path_init_file=left(Wscript.ScriptFullName,len(Wscript.ScriptFullName)-
len(Wscript.ScriptName))& "nicinfo.ini"
 '设置配置文件为当前路径下的 nicinfo.ini
'nicinfo.ini 文件注意是在虚拟机生成的时候通过 virt-copy-in 注入虚拟机
Path_log_file=left(Wscript.ScriptFullName,len(Wscript.ScriptFullName)-
len(Wscript.ScriptName))& "nicinfo.log"
 '设置日子文件为当前路径下的 nicinfo.log

wrlog(" 脚本开始运行时间是 :")

Set objFSO = Createobject("Scripting.FileSystemObject")
If  objFSO.Fileexists(path_init_file) Then
'msgbox("fil now")
else
'msgbox("fil now ---")
    exit sub
end if

Set fso=CreateObject("Scripting.FileSystemObject")
Set file=fso.OpenTextFile(Path_init_file,ForReading)

While (Not file.AtEndOfLine)

  msg=file.ReadLine
  nicfile(i)=msg
  i=i+1
  msg=trim(msg)

  if instr(msg, "c_name:")>0 then
      k = Instr(msg, ":")
      m = Len(msg)
      reval = Right(msg, m - k)
  else
  end if

  if instr(msg,"ipsetup:")>0 then
      k = InStr(msg, ":")
      m = Len(msg)
      ifrun = Right(msg, m - k)
      'msgbox(ifrun)
          if (ifrun="0") then
```

1107114476@qq.com
dolphinebook.com

```
                ifrun="1"
            else
                exit sub
            end if
        'msgbox(ifrun)

    else
    end if

    if instr(msg,"mac:")>0 then
        k = InStr(msg, ":")
        m = Len(msg)
        mac(x) = Right(msg, m - k)
        x=x+1
                    'msgbox(routmac)
    else
    end if

    if instr(msg,"ip:")>0 then
        k = InStr(msg, ":")
        m = Len(msg)
        ip(x) = Right(msg, m - k)
        'msgbox(routip)
    else
    end if

    if instr(msg,"mask:")>0 then
        k = InStr(msg, ":")
        m = Len(msg)
        mask(x) = Right(msg, m - k)
        'msgbox(routmask)
    else
    end if

    if instr(msg,"gw:")>0 then
        k = InStr(msg, ":")
        m = Len(msg)
        gw(x) = Right(msg, m - k)
        'msgbox(routgw)
    else
    end if


Wend
file.Close
Set file=Nothing
Set fso=Nothing


'wrlog("Configure Compute Name :")
```

1107114476@qq.com
dolphinebook.com

```
'Set objnet = CreateObject ("WScript.Network")
'Set R = CreateObject("WScript.Shell")

'if (ifrun="1") then
'  strComputer = "."

 ' Set objWMIService = GetObject("winmgmts:" _
  '  & "{impersonationLevel=impersonate}!\\" _
   ' & strComputer & "\root\cimv2")

  'Set colComputers = objWMIService.ExecQuery _
   ' ("Select * from Win32_ComputerSystem")

  'For Each objComputer in colComputers
   ' If reval <> "" Then
    'errReturn = ObjComputer.Rename (reval)

  'R.run("Shutdown.exe -r -t 0")
   'end If
  'next
'else
'end if

wrlog(" 配置外网 ip :")
WScript.Sleep   50000

strGatewaymetric = Array(1)
For y=0 To x-1
  'Dim strComputer,objWMIService,colNetAdapters,strIPAddress,strSubnetMask,strG
  ateway,errEnable,errGateways,objNetAdapter
  strComputer = "."
  Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\CIMV2")
  Set colNetAdapters = objWMIService.ExecQuery("Select * from Win32_
  NetworkAdapterConfiguration where (MACAddress ='"+ mac(y)+"' )")
  strIPAddress = Array(ip(y))
  strSubnetMask = Array(mask(y))
  strGateway = Array(gw(y))
  For Each objNetAdapter In colNetAdapters
    errEnable = objNetAdapter.EnableStatic(strIPAddress, strSubnetMask)
    errGateways = objNetAdapter.SetGateways(strGateway, strGatewaymetric)
  next
next

wrlog("ip 配置完成 :")

Set fso=CreateObject("Scripting.FileSystemObject")
Set file=fso.OpenTextFile(Path_init_file,ForWriting)

file.write("ipsetup:"+ifrun + vbCrLf)
```

1107114476@qq.com
dolphinebook.com

```
file.Close

Set fso=CreateObject("Scripting.FileSystemObject")
Set file=fso.OpenTextFile(Path_init_file,8)
do until j > i
'msgbox(j)
file.writeline(nicfile(j))
j=j+1
loop
file.Close

wrlog(" 修改配置文件完成 :")

Set file=Nothing
Set fso=Nothing

' 删除组策略配置文件
Set fso=CreateObject("Scripting.FileSystemObject")
'fso.deletefile("C:\WINDOWS\system32\GroupPolicy\Machine\Scripts\Scripts.ini"),true
Set fso=Nothing
Set   shell=WScript.CreateObject("WScript.Shell")
'shell.run("del  C:\WINDOWS\system32\GroupPolicy\Machine\Scripts\Scripts.ini /f /q /a")
' 强制更新组策略
'shell.run("c:\windows\system32\gpupdate /force ")

WScript.Sleep   5000
' 重启

'WScript.Sleep   10000
'set win32_OS=getobject("winmgmts:{(Shutdown)}//./root/cimv2").execQuery
("select * from win32_operatingsystem where primary=true")
'for each OS in win32_OS
'OS.win32shutdown(6)
'next
'set win32_OS=nothing

end sub

function wrlog(logmsg)
Set   WshShell=WScript.CreateObject( "WScript.Shell")
 ' 得到程序所在的当前路径
Path_log_file=left(Wscript.ScriptFullName,len(Wscript.ScriptFullName)-len(Wscript.
ScriptName))& "nicinfo.log"     ' 设置日子文件为当前路径下的 nicinfo.log

' 判断日志文件是否存在，如果不存在则创建
Set logfso=CreateObject("Scripting.FileSystemObject")
If  logfso.Fileexists(Path_log_file) Then
'msgbox("fil now")
else
  set ts=logfso.CreateTextFile(Path_log_file, True)
```

1107114476@qq.com
dolphinebook.com

```
  ts.close
end if

'写日志

Set logfile=logfso.OpenTextFile(Path_log_file,8)
rtime=cstr(now())
logfile.write(logmsg+rtime + vbCrLf)
logfile.Close
end function
```

在脚本中读取 nicinfo.ini 文件，这个文件在虚拟机生成的时候，通过 virt-copy-in 命令，复制到虚拟机的 c:\Windows 目录下面，nicinfo.ini 文件格式如下：

```
ipsetup:1                 1 或者 0 控制是否运行 IP 配置脚本
ip:88.88.88.88            需要配置的外网 IP
mask:255.255.255.128      外网 IP 子网掩码
gw:88.88.88.1             外网网关
mac:00:16:3e:02:79:f4     外网的 MAC 地址，使用 MAC 地址主要是可以实现精确的网卡匹配
ip:10.88.88.1             内网 IP
mask:255.255.255.0        内网子网掩码
gw:0.0.0.0                内网网关，0.0.0.0 表示没有网关
mac:00:16:3e:a0:11:d3     内网 mac 地址
```

2. Linux 虚拟机 IP 地址自动配置的方法

Linux 系统将 IP 的配置信息作为参数传给 IP 配置脚本，通过 virt-copy-in 复制到虚拟机的 rc.local 中，这样，在虚拟机启动的时候，会运行配置脚本，完成虚拟机 IP 配置。

IP 配置脚本名为 setipbymac.sh，需要修改的 rc.local 内容如下：

```
sh /bin/setipbymac.sh vm-hostname 10.0.102.1 10.0.102.190 255.255.255.0
52:54:00:EA:AB:77 172.0.102.190 255.255.255.0 52:54:00:28:16:B7
```

在虚拟机模板配置的时候，将 setipbymac.sh 脚本复制到 /bin 目录下，并配置执行属性，setipbymac.sh 脚本内容如下：

```
#!/bin/bash
#write by xiaoli V2.1
#set hostname and nic ip

#################function set nic info#############
# 函数 set_nic_ip，功能是设置网卡信息
set_nic_ip(){
# 网卡名字为第一个参数
nic_name=$1
# IP 为第二个参数
nic_ip=$2
# 掩码为第三个参数
nic_netmask=$3
```

1107114476@qq.com
dolphinebook.com

```
# 根据网卡名字生成 IP 配置文件
nic_name_file="/etc/sysconfig/network-scripts/ifcfg-"$nic_name
# 根据网卡名字将原有的 IP 文件进行备份
nic_name_file_bak="/etc/sysconfig/network-scripts/ifcfg-"$nic_name".bak"
# 检查网卡名字是否赋值成功，如果不成功就退出
if [ -z "$nic_name" ];then
    echo "no nic name give!"
    exit 5
else
    echo " nic name check ok   "
fi
echo "nic_name is" $nic_name
echo "nic_ip is" $nic_ip
echo "nic_netmask is " $nic_netmask
echo "nic file name is " $nic_name_file
# 检查是否存在 IP 配置文件，存在先进行备份
if [ -f "$nic_name_file" ];then
    echo "nic file exist! backup it"
    rm $nic_name_file_bak -f
    mv $nic_name_file $nic_name_file_bak
else
    echo " nic file not exist create it   "
fi
# 生成 IP 配置文件
touch $nic_name_file
echo "DEVICE="$nic_name >>$nic_name_file
echo "BOOTPROTO=none" >>$nic_name_file
echo "ONBOOT=yes" >>$nic_name_file
echo "TYPE=Ethernet" >>$nic_name_file
echo "IPADDR="$nic_ip >>$nic_name_file
echo "NETMASK="$nic_netmask >>$nic_name_file
# 启动网卡
/sbin/ifdown $nic_name
/sbin/ifup $nic_name
echo "++++++++++++++++++++++++++++++++++++++++++++++++++"
cat $nic_name_file
echo "++++++++++++++++++++++++++++++++++++++++++++++++++"
}
################################################


#************************get variable*********
# 检查是否是 RHEL/CentOS7
uname -r |grep el7
#mac 地址赋值
mac1=$5
mac2=$8
# 如果是 RHEL/CentOS7，将 MAC 地址转化为小写
if [[ $? -eq 0   ]];then
    #typeset -i get_nic1_mac
```

1107114476@qq.com
dolphinebook.com

```
    #get_nic1_mac="${mac1}"
    get_nic1_mac=$(echo $mac1 | tr '[A-Z]' '[a-z]')
    get_nic2_mac=$(echo $mac2 | tr '[A-Z]' '[a-z]')
else
    get_nic1_mac="${mac1}"
    get_nic2_mac="${mac2}"
fi
# 得到主机名、外网网关、外网 IP、外网掩码、外网 MAC、内网 IP、内网掩码、内网 MAC 信息
get_host_name=$1
get_gateway=$2
get_nic1_ip=$3
get_nic1_netmask=$4
#get_nic1_mac=$5
get_nic2_ip=$6
get_nic2_netmask=$7
#get_nic2_mac=$8
#***********************get variable*********

###############set hostname#####################
# 配置主机名
echo "host name is "$get_host_name
# 检查主机名是否赋值成功，如果不成功就退出
if [ -z "$get_host_name" ];then
    echo "please give host name!"
    echo "variable order by hostname gateway nic1 ip nic1 netmask nic1 mac nic2
    ip nic2 netmask nic2 mac"
    exit 6
else
# 配置 RHEL/CentOS6 主机名，如果是 RHEL/CentOS7，不存在 HOSTNAME 关键字，不造成影响
    sed -i 's/.*HOSTNAME.*/HOSTNAME\="'$get_host_name'"/g' /etc/sysconfig/network
# 配置 RHEL/CentOS7 主机名，如果是 RHEL/CentOS6 不存在 hostnamectl 命令，不造成影响
    hostnamectl set-hostname $get_host_name
    echo "host name set OK!"
fi
################################################

###############set gateway#####################
# 配置网关
echo "gate way  is"$get_gateway
# 检查网关是否赋值成功，如果不成功就退出
if [ -z "$get_gateway" ];then
    echo "please give gateway!"
    echo "variable order by hostname gateway nic1 ip nic1 netmask nic1 mac nic2
    ip nic2 netmask nic2 mac"
    exit 7
else
    if [ 'cat /etc/sysconfig/network |grep GATEWAY |wc -l'  -eq 1  ];then
        sed -i 's/.*GATEWAY.*/GATEWAY\="'$get_gateway'"/g' /etc/sysconfig/network
    else
        echo "GATEWAY="$get_gateway >> /etc/sysconfig/network
```

1107114476@qq.com
dolphinebook.com

```
    fi
    echo "gate way set OK!"
fi
echo "----------------------------------------------"
cat /etc/sysconfig/network
echo "----------------------------------------------"
################################################

temp001='/sbin/ifconfig -a |grep Link'
# 检查是否是 RHEL/CentOS6，还是 RHEL/CentOS7，根据不同系统的 ifconfig 命令输出
# 得到每块网卡的名字、MAC 地址信息
if [[ $? -eq 0   ]];then
    allif='/sbin/ifconfig -a |grep Link |awk '{print $1}'|grep -vE "lo|inet6|sit0"'
    allmac='/sbin/ifconfig -a |grep Link |awk '{print $5}'|grep -vE "lo|inet6|sit0"'
    ifcounter='/sbin/ifconfig -a |grep Link|grep -vE "lo|inet6|sit0" |wc -l'
else
    if [[ $? -eq 1  ]];then
    allif='/sbin/ifconfig -a |grep mtu |awk -F :  '{print $1}'|grep -vE "lo|inet6|sit0"'
    allmac='/sbin/ifconfig -a |grep ether |awk '{print $2}'|grep -vE "lo|inet6|sit0"'
    ifcounter='/sbin/ifconfig -a |grep mtu|grep -vE "lo|inet6|sit0" |wc -l'
fi
fi
echo "ifname is " $allif
echo "mac add is " $allmac
echo "ifcounter is " $ifcounter

#push if and mac to array
# 将 IP MAC 信息放到数组中
x=1
while [ $x -le $ifcounter ];

      do
      #STR_TEMP='printf "%s%s" "$STR_TEMP" "$1"'
      ifname[$x]='echo $allif|awk '{print $'$x'}''
      ifmac[$x]='echo $allmac|awk '{print $'$x'}''
      echo "x" $x
      echo "ifname" ${ifname[$x]}
      echo "ifmac" ${ifmac[$x]}
# 判断系统实际的 MAC 和脚本参数的 MAC 是否相同，如果相同就调用 IP 脚本配置函数
      if [ "${ifmac[$x]}" = "$get_nic1_mac"    ];then
          set_nic_ip ${ifname[$x]} $get_nic1_ip $get_nic2_netmask
      else
          echo "not nic1"
      fi
      if [ "${ifmac[$x]}" = "$get_nic2_mac"    ];then
          set_nic_ip ${ifname[$x]} $get_nic2_ip $get_nic2_netmask
      else
          echo "not nic2"
      fi
      x='expr $x + 1'
```

1107114476@qq.com
dolphinebook.com

```
done
#b[0]='echo $a |awk  '{print $1}''
#b[1]='echo $a |awk  '{print $2}''
#echo ${b[0]}
#echo ${b[1]}
#-----------------disable rc.local-----------------------
# 将 IP 配置脚本注释掉，放置下次开机运行
sed -i '/^sh\ \/bin\/setipbymac/s/^/#/' /etc/rc.local
sed -i '/^sh\ \/bin\/setipbymac/s/^/#/' /etc/rc.d/rc.local
sed -i 's/\(127.*\)/\1 "'$get_host_name'"/' /etc/hosts
sed -i 's/"/\ /g' /etc/hosts
# 配置完成重启系统
reboot
```

16.4　虚拟机镜像测试

根据经验，不同操作系统、驱动、内核版本对稳定性和性能有不同的影响，尤其是对稳定性的影响。所有的虚拟机都是通过模板进行复制，所以，虚拟机模板的测试非常重要。

对虚拟机镜像的测试，主要通过一系列专业测试工具，测试镜像的性能、兼容性、稳定性。测试流程如表 16-3 所示，具体的测试工具和方法，在 15.6 节有详细介绍。

表 16-3　虚拟机镜像测试列表

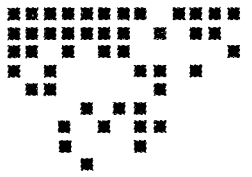
测试内容	目　　的
虚拟机系统内反复重启 50 次	测试重启稳定性
虚拟机断电重启 50 次	测试断电重启稳定性
磁盘 I/O 性能测试	测试虚拟机磁盘最大性，优选不同系统合适的磁盘格式
网络 I/O 性能测试	测试虚拟机网络最大性，优选不同系统合适的网卡驱动
磁盘 I/O 稳定性测试	磁盘持续高压力测试 5 小时，测试稳定性
网络 I/O 稳定性测试	网络高吞吐、高发布率情况下，持续测试 24 小时，测试稳定性
业务压力测试	测试虚拟机最大的业务负载压力
业务稳定性测试	业务最大压力持续压测 24 小时，测试虚拟机稳定性

16.5　本章小结

本章介绍了 Windows 和 Linux 模板镜像的制作方法和配置、测试注意点，虚拟机模板制作需要非常细致、认真、谨慎，并严格进行测试，有一点疏忽，都可能会造成线上同版本的虚拟机稳定性问题，这方面笔者有多次深刻的教训。

下一章将为读者介绍单机虚拟机技术与应用场景。

1107114476@qq.com
dolphinebook.com



Chapter 17 第 17 章

单机虚拟化技术与生产环境实践

第 16 章详细地介绍了虚拟机模板镜像的制作过程，镜像制作完成后，下一步就是在生产环境使用。生产环境虚拟化的主要形式有单机虚拟化和集群虚拟化。单机虚拟化是生产环境使用最广泛的虚拟化形式，本章将介绍单机虚拟化技术，重点介绍单机虚拟化技术及应用场景，第 18 章介绍集群虚拟化技术及生产环境实践。

17.1 单机虚拟化技术与应用场景

什么是单机虚拟化？单机虚拟化，从字面上理解，首先是单机，即单台宿主机，其次是虚拟化，连起来就是由单台宿主机实现的虚拟化方式。单机虚拟化一台宿主机就是一套虚拟化体系，这台宿主机上所有的虚拟机，使用的计算、内存、存储、网络资源都在这台宿主机内部，单机虚拟化每台虚拟机的物理边界就是所在的宿主机。

单机虚拟化部署灵活，性价比非常高，是生产环境应用最广泛的虚拟化方式，应用场景非常广泛，主要有以下场景：

- 老业务迁移到虚拟化环境，为了保持业务兼容，保持 IP 等因素不变。
- 物理位置零散分布的业务。
- 对成本敏感的业务。
- 对性能要求非常高的业务。

1107114476@qq.com
dolphinebook.com

17.1.1　单机虚拟化的优势和劣势

1. 单机虚拟化的优势

（1）单机虚拟化有很好的成本优势

单机虚拟化不需要额外投入存储、网络资源，不需要另外改造现有的网络结构。硬件成本、时间成本、人员投入成本都是最低的，是性价比最高的虚拟化方式。

（2）单机虚拟化具有很高的灵活性

单机虚拟化非常灵活，部署几台服务器就可以搞起来。例如，经常碰到这样的情况，在某个机房里面，有一批非常陈旧的服务器正在运行，这批服务器已经很老了，而且全部都过了保修期，服务器上面的应用可能都是很多年前部署上去的，环境非常复杂，可能维护这些应用的人，已经更换了好几批，重新申请新的服务器，重新部署业务难度非常大。单机虚拟化，在这种场景下就非常合适，可以采用本书第 7 章介绍的 P2V 技术，将老旧的服务器转换成虚拟机，业务环境、IP 都可以保持的和原来一模一样，而对于业务来说，只需要停机一段时间即可。

（3）单机虚拟化实现简单

对于业务初期或者刚开始尝试应用虚拟化技术的团队，单机虚拟化技术简单，可以很快地上手。它没有复杂的架构，宿主机就是一台配置符合要求的普通服务器，要使用 KVM 虚拟化，只需要安装一些必要的软件包即可，对于一般的业务运维，实现起来相对简单，可以在很短的时间内，将虚拟化部署起来。

（4）单机虚拟化性能高

不同的业务对于服务器的压力差别是非常大的。如数据库服务器，压力一般多来自磁盘的读 / 写；游戏的 GS（Game Server）服务器压力大多在 CPU；视频业务服务器，压力都在内存和网络。单机虚拟化性能高的优势主要体现在磁盘和网络。

单机虚拟化通过硬件配置和特有的虚拟化技术，可以达到接近物理机的性能，能很好地应对这些性能压力。对于磁盘读 / 写压力大的业务，单机虚拟化宿主机上可以通过配置 SSD 硬盘，大幅提升虚拟机的读 / 写性能。单机虚拟化在磁盘性能方面的优势是，和采用共享存储的方式相比，单机虚拟化可以非常灵活地调整虚拟机独占不同的 SSD 硬盘，达到和物理机近似的性能，而共享存储是多台虚拟机分享存储资源，单台虚拟机的磁盘性能会受到限制。

对于网络压力大的业务，可以通过开启网卡的 SR-IOV（Single-Root I/O Virtualization）使虚拟机网络性能得到非常高的提升，而 SR-IOV 目前的特性是不支持迁移的，非常适合单机虚拟化。

（5）单机虚拟化故障影响面小

在服务器运行过程中，故障是不可避免的，一旦出现故障，影响范围有多大，这点是非常重要的。单机虚拟化时，因为一台宿主机上运行的虚拟机的数量有限，当一台宿主机出现问题的时候，只会影响这台宿主机上的虚拟机，对其他的虚拟机不会造成影响。和集群虚拟

1107114476@qq.com
dolphinebook.com

化对比来看，集群虚拟化采用集中式的存储，虽然出现问题的几率比较小，但是一旦存储、网络出现问题，整个集群的全部虚拟机都会受到影响。

2. 单机虚拟化的劣势

（1）单机虚拟化难以统一管理

灵活性带来的另一个问题就是混乱，宿主机在物理机上分布在多个地方，在网络上分布在多个网段，一般非常分散，不方便管理，有违资源的整体规划原则。

（2）单机虚拟化故障恢复慢，有可能虚拟机彻底丢失

因为虚拟机都运行在单台宿主机上，一旦这台宿主机出现问题，恢复业务时间周期比较长。

如果是宿主机非磁盘硬件故障，有两种恢复方案。第一种方案是将虚拟机迁移到健康的宿主机上，这个过程中，虚拟机是不可用的，而迁移的过程，实际是文件的复制，当虚拟机文件非常大的时候复制过程就非常漫长。第二种恢复的方式是通过备机，将故障宿主机上的硬盘，全部换到备机上，直接开机，这种恢复方式相比虚拟机迁移要快一些，但是对于品牌多样的服务器，每种都要准备一台备机。

如果非常不幸，发生极端情况，宿主机多块硬盘同时故障，那虚拟机就会彻底丢失，只能从备份恢复虚拟机，然后再恢复业务数据，这种情况就有可能会造成业务数据丢失，但是，一般发生这种情况的几率非常低，笔者在几年的虚拟化运维实践中，因为生产环境一般都做的是 RAID10，同一个 RAID1 组两块磁盘同时故障的情况，只碰到过一次，并且碰到这种情况，还可以使用应用层面的数据进行恢复，因为应用层面都是每天备份一次，所以，恢复的时候，丢失了将近一天的数据。但是恢复业务也用了 7 个多小时。

还有一种情况，就是宿主机系统故障，需要修复系统，或者将虚拟机镜像文件复制到备用宿主机上，恢复时间也比较长。单机虚拟化的详细故障恢复方法在第 21 章有详细介绍。

17.1.2 单机虚拟化技术

1. 单机虚拟化磁盘技术

在单机虚拟化中，虚拟机的硬盘可以采用目前 KVM 支持的全部格式，包括 raw、qcow2、qed、裸盘、lvm 等，每种格式的功能和性能在第 5 章做了详细的介绍和测试，本节就不再重复介绍了。一般情况下，在单机虚拟化中，建议优先使用 qcow2 或者 lvm 裸盘的方式作为虚拟化的磁盘，主要有以下两个原因。

① qcow2 镜像文件在宿主机上实际占用空间，默认是按需占用的，除非添加参数 preallocation=full，才会预分配空间，命令如下：

```
[root@localhost ~]# qemu-img create -f qcow2 -o size=5G,preallocation=full test.qcow2
Formatting 'test.qcow2', fmt=qcow2 size=5368709120 encryption=off cluster_
size=65536 preallocation='full'
```

lvm 从 CentOS 6.4 开始也支持精简模式，使用 lvm 方式主要是 I/O 性能比较好，一些 I/O 性能要求比较高的虚拟机建议使用 lvm 的磁盘方式。

1107114476@qq.com
dolphinebook.com

② qcow2、lvm 方式的虚拟机硬盘，支持快照功能，这个特性方便进行虚拟机的备份和恢复。

提示　qcow2 方式的虚拟机创建快照备份的方式有两种，第一种是直接使用 virsh snapshot-create vm-name 命令来创建，过程会锁定虚拟机，这个方法只适合在业务已停止的情况下为虚拟机做备份。第二种是使用 virsh snapshot-create-as 命令将当前虚拟机运行镜像到新的文件中，然后再将老的虚拟机镜像文件复制备份，再使用 virsh blockcommit 命令将当前镜像合并回原镜像，但是 CentOS 6/CentOS 7 系统中自带的 qemu-img 命令版本及 Libvirt 版本不支持这样的操作，需要升级 qemu-img 到 2.0.x 版本以上，Libvirt 到 1.2.8 版本以上才能支持。

在单机虚拟化中，针对极端虚拟机磁盘压力情况，可以为宿主机配置 SSD 盘。虚拟机直接使用 SSD 裸盘，可以有效地提升虚拟机的磁盘读 / 写性能。在部署的过程中，对于重要业务，笔者一般在每台宿主机上配置 4 块 SSD 盘，做 RAID10，以应对虚拟机运行过程中突发的磁盘读 / 写压力及磁盘故障。

2. 单机虚拟化网络技术方案

单机虚拟化在网络方面非常灵活，本书的第 4 章介绍了 KVM 网络虚拟化技术，其中介绍的全部网络类型在单机虚拟化都可以使用，如 Macvtap、Vhost-net、SR-IOV、Bridge、Open vSwitch，等等。在第 4 章中，对这些网络虚拟化技术都做了详细的介绍，这节就不再重复介绍了。

上面列举的几种网络虚拟化技术，从功能上说，最强大的是 Open vSwitch，它可以结合 SDN 非常方便地实现虚拟机之间的组网和隔离；从性能上来说，SR-IOV 是最好的，测试发现，在虚拟机为 Linux 系统的情况下，虚拟机的网卡可以达到物理网卡 90% 以上的性能；从配置上说，桥接是最方便的，对于刚开始接触虚拟化的读者来说，可以尝试桥接方式。其他几种网络虚拟化技术用的相对较少一些。

提示　SDN 软件定义网络（Software Defined Network），是一种新型网络创新架构，是网络虚拟化的一种实现方式，其核心技术通过将网络设备控制面与数据面分离开来，从而实现了网络流量的灵活控制，是近几年网络方面正在进行的革命，是未来网络发展的方向之一。

3. 单宿主机虚拟机迁移方案

单宿主机迁移方案，根据虚拟机的状态，可以分为两类，即冷迁移和热迁移。

冷迁移需要虚拟机关机，然后通过程序、脚本或者命令，将虚拟机的镜像文件从一台

1107114476@qq.com
dolphinebook.com

宿主机复制到另一台宿主机，然后配置虚拟机的 xml 配置文件，将虚拟机在新的宿主机上开机，整个过程就完成了。需要注意的是，新、老宿主机上的网卡对应关系，需要一致，否则，虚拟机开机后网络会不通。

热迁移是在虚拟机运行的过程中，在线将一台虚拟机从一台宿主机上迁移到另一台宿主机上，迁移过程对虚拟机基本没有影响。例如，使用如下命令，将虚拟机 vm_centos65x64 迁移到宿主机 192.168.1.21 上：

```
[root@gcloud-qd-c1-169 ~]#/usr/bin/virsh --connect qemu:///system  migrate
--live --copy-storage-all --persistent --undefinesource vm_centos65x64
qemu+ssh://192.168.1.21/system
```

虚拟机的热迁移技术，最初只在有共享存储的集群虚拟化场景中提供支持，在 CentOS 6.5 之后，开始支持对无共享存储环境下的虚拟机热迁移。两个场景虽然不同，但是原理基本上是一致的，只是无共享存储的场景下，先要迁移磁盘存储，然后再迁移虚拟机内存，消耗的时间会长很多。

17.1.3 单机虚拟化应用场景

单机虚拟化的应用场景十分普遍，包括目前一些主流的公有云，也都采用单机虚拟机来为客户提供云资源。本节介绍 3 个最适合的应用场景并加以说明。

（1）场景一：P2V，即物理机转化为虚拟机

这种情况，通常是因为原本的物理服务器年代久远，配件过保修期或者停产等原因，但上面仍然运行了一些系统，这些系统配置非常复杂，几乎无法重新部署。这种情况下，最适合做 P2V，将原本的物理服务器用虚拟机替代，因为单机虚拟化非常灵活，将 P2V 后的虚拟机运行在单机虚拟化上非常合适。

（2）场景二：高性能应用

对于负载高的应用，使用单机虚拟化是非常合适的，因为单机虚拟化的宿主机配置的定制性非常强，可以更具应用的需要，为宿主机添加配置，如磁盘可以使用 SSD 来加强性能，网卡可以使用支持 SR-IOV 的网卡，CPU 可以选用高主频多核等。总之，单机虚拟化宿主机的可定制性强，决定了它能承载更大负载的虚拟机。

（3）场景三：定制化业务

这类应用一般与业务的架构有关系，例如，Web 类的应用，一般在业务层面都会考虑负载均衡、数据库主从、Memcached 主从等，以保证业务的可用性；或者对于游戏来说，很多游戏都有分区分组的概念，每个区组由一组角色的服务器来提供服务。以上两类业务就非常适合使用单机虚拟化，因为服务器宕机是无法完全避免的，既然无法避免，就希望一旦出现宕机的情况，对业务造成的影响尽可能小，从这个出发点看，可以将 Web 类应用的主从应用放到两台单机宿主机上，这样即使一台宿主机宕机，业务至少可以保证不中断。对于游戏业务，可以将一个区组的服务器放在一台单机虚拟化宿主机上，宕机造成的最大危害只会影响

1107114476@qq.com
dolphinebook.com

一个区组的游戏，当然，如果游戏架构内部存在冗余机制的话，宿主机宕机也是可以做到不影响游戏的。

17.2　单机虚拟化生产环境注意点

单机虚拟化在生产环境的使用已经非常普遍，但是，在使用单机虚拟化之前，还是需要做一些准备和故障恢复预案，本节介绍单机虚拟化生产环境的注意点，防患于未然，尽量避免一切可预见的故障，提高稳定性、可用性是最终的目的。

17.2.1　单机虚拟化灾备与应急方案

虚拟化的灾备和应急方案在第 21 章有详细介绍，本节只介绍一些单机虚拟化需要特别注意的地方。

1. 单机虚拟化的灾备

备份是一个老生常谈的话题，对于所有的线上业务，都必须做好备份和恢复的准备，在单机虚拟化的场景中，这点也非常重要。

首先，需要区分单机虚拟化的备份和业务层面的备份，这是两个概念，虚拟化层面的备份可以保证虚拟机的可用性，但是虚拟机内部业务层面的数据一致性还是需要业务层面的备份来保障的。

特别值得一提的是数据库应用，需要注意，数据库一致性只能由数据库备份来保证。经过长期的实践发现，**仅靠虚拟机的备份，恢复后数据库内的数据有非常高的概率是不一致的。**

（1）备份策略

对于虚拟机的备份，需要根据业务类型、业务时间段来做区分。一般来说，对于单机虚拟化，从业务分类来看，如数据库类应用，那么备份的频率不需要很高，在业务最初部署完成测试通过后，可以做一个备份，发生故障的时候，将业务层面的数据恢复回来，业务便能恢复。需要分时段进行备份的业务（如 Web 类应用），通常情况下，Web 类应用产生的数据变化量很少，但是会定期发布新的版本，这种情况下，可以在每一个新版本发布前和发布后做两个备份，发布前的备份可以作为发布失败后的回滚，发布后的备份可以在故障发生时用于恢复。在游戏业务中，GS 这个角色的服务器也非常适合用这种备份方案。

（2）备份实施

除了备份策略，也需要介绍备份如何做。对于单机虚拟化，通常情况下，每台虚拟机的硬盘在宿主机上对应一个镜像文件，最简单的备份方案是虚拟机关机，将虚拟机硬盘文件

1107114476@qq.com
dolphinebook.com

复制一份，然后将文件备份到其他机器上。这个方法操作最简单，如果业务可以接受停机的话，用这种方式备份是不错的选择，但是，对于还有一些需要 7×24 小时不能停的业务，用上面这种方式就不可行了。当然，KVM 管理工具 Libvirt 已经提供了两种在线备份虚拟机的方案，分别是在线块复制和在线外部快照，在第 21 章都有详细介绍。

2. 单机虚拟机的应急方案

应急方案主要包括故障定位及故障恢复。

故障定位。一般来说，单机虚拟化的故障包含虚拟机故障和宿主机故障两大类，如虚拟机故障，首先需要建立完整的虚拟机与宿主机的关系信息，通过虚拟机的 IP，能很快找到对应的宿主机，然后再由宿主机连接到虚拟机的 VNC（远程控制软件），查看虚拟机的运行状态。如果只是虚拟机系统层面的故障，处理方法就按照一般服务器故障处理；如果是虚拟机死机或者没有响应等问题，一般的处理方法是重启虚拟机。对于一台宿主机上多台虚拟机同时出问题的情况，那么就需要考虑问题是不是出在宿主机，或者宿主机网络上。

故障恢复。对于一般的虚拟机故障，通常处理方法与一般的服务器故障一样，在虚拟机系统内部找原因，或者重启虚拟机。对于宿主机故障，如果是宿主机的 CPU、内存、主板、电源等故障，一般采用更换部件来解决。

17.2.2 单机虚拟化生产环境部署实践

本节介绍单机虚拟化生产环境部署实践，单机虚拟化部署有两种方式，即手工部署和通过管理平台部署。

1. 手工部署

手工部署是指不借助任何管理平台，手工独立地逐台部署宿主机。这种情况下，可以编写一些部署脚本，在脚本中定义虚拟机的配置，然后通过脚本来创建虚拟机。这种情况下，虚拟机名就需要有一定含义，如以用途加配置加 IP 作为虚拟机的名称。例如，web-2c-4g-192-168-10.11，表示 Web 类应用，配置为双核，内存为 4GB，IP 为 192.168.10.11，这种命名方式可以非常直观地看到虚拟机的一些必要信息，一旦虚拟机出现问题，可以快速定位并处理。对于虚拟机的 MAC 地址，可以采用将 IP 转换为 16 进制的方式来定义虚拟机的 MAC，这样做可以方便地通过 MAC 地址来定位 IP 地址，对以后的故障排除非常有用。

2. 通过管理平台管理单机虚拟化

管理平台可以自主开发，也可以使用一些开源的平台，本书第 11 章、第 12 章、第 13 章介绍了主流的 KVM 管理平台，如 OpenNebula、OpenStack、oVirt 等，这些管理平台的功能都非常强大。通过平台管理的虚拟机，操作起来要方便很多，需要的大部分信息都可以通过平台来查询，当出现问题时，也可以快速的定位问题并处理。

以上介绍的两种使用单机虚拟化的方式各有优缺点。手工部署方式适合业务规模不大的

1107114476@qq.com
dolphinebook.com

情况，自动化程度较低，但是操作简单，故障点少；管理平台部署方式适合大规模部署，通过平台可以实现虚拟机完整的生命周期管理，但是平台本身的维护、管理和稳定性，都需要考虑，通常适合有一定虚拟化运维经验的人来使用。

3. 笔者开源的单机虚拟化部署脚本

笔者使用 Python 编写了一套单机虚拟化部署脚本，已经使用 3 年多了，运行稳定。目前，笔者生产环境的单机虚拟化都是通过这套脚本部署的。脚本已经开源，部署在 github 上，地址如下：

https://github.com/xiaoli110/kvm_vm_setup。

（1）脚本功能

脚本可以完成 KVM 虚拟机自动部署，包括 IP 注入配置，主要功能如下：

- 全自动的完成虚拟机生成、主机名、IP 地址配置。
- 支持 Windows 系统、RHEL 6/RHEL 7、CentOS 6/CentOS 7、RHEL 6/RHEL 7 衍生系统、Ubuntu 系统的部署，其他系统未做测试。

（2）脚本使用环境

脚本使用前提条件如下：

- Windows、Linux 虚拟机模板系统，需要按照第 16 章的镜像制作方法操作。
- 宿主机上，镜像、虚拟机的存储路径为 /datapool。
- 如果要使用 lvm，vg 的名字必须是 vmvg。
- 宿主机系统只支持 RHEL 6/RHEL 7、CentOS 6/CentOS 7、RHEL 6/RHEL 7 衍生系统，脚本通过调用 Libvirt 配置虚拟机，通过 guestfish 编辑虚拟机镜像，需要安装虚拟化组件、Libvirt 组件、guestfish 组件。

（3）脚本的使用方法

1）下载源代码。脚本必须用 root 用户运行，首先下载源代码，命令如下：

```
git clone https://github.com/xiaoli110/kvm_vm_setup
```

2）脚本运行环境准备。环境准备需要运行 firsh.sh 脚本，会自动判断是 CentOS 6 或者 CentOS 7 操作系统，并安装虚拟化、Linux、Guestfish 组件，然后升级操作系统到最新。

安装完成后重启系统，并启动 libvirt 服务。

```
service libvirtd restart
```

3）编辑虚拟机配置文件。虚拟机配置主要通过 vm.csv 文件配置，文件位于脚本根目录下，编辑 vm.csv 文件，格式如下：

```
#vmflag,template,name,disk1_size,disk2_size,mem_size,cpu_num,nic_type,nic1_
bridge,nic2_bridge,vnc_port,outip,outmask,outgw,inip,inmask,ingw
#vm template name must be ''win2003ent32chs' 'win2003ent64chs' 'win2008ent64chs'
'centos56x64' 'ubuntu1204X64'
```

1107114476@qq.com
dolphinebook.com

```
#keyword vmLvm means create lvm for vm,default vg is datavg
#keyword vmCpOnly means not resize images , just cp imaiges,default vg is datavg
#kerword url: means download vm images path,like
#url:http://172.16.1.100/
ftp:ftp://ftpuser1:password@ftp.myimages.com:10021/
ftp:ftp://ftpuser2:password@ftp.myimages2.com:10021/
# 此处如果指定 url，会到 url 地址上去拉去虚拟机镜像，支持 ftp 方式，拉取方式为 url 加镜像名字，
  镜像名字为下面虚拟机配置文件中的镜像名字
# 可以指定两个 url，脚本会自动比较多个 url 的速度，需要在 url 里面放置一个 200MB 大小的名为
  "ratetest" 的文件
vm,win2003ent32chs,win2003-138,20G,20G,2048,2,e1000,br1,br1,5921,10.10.10.21,25
5.255.255.0,10.10.10.1,192.168.122.138,255.255.255.0,none
vmLvm,win2003ent32chs,virt1-lab-222,10G,10G,2048,2,e1000,br2,br2,59222,10.0.0.2
22,255.0.0.0,10.0.0.1,172.16.2.222,255.255.255.0,none
vmCpOnly,centos6564.qcow2,centos65-8,30G,20G,2048,2,virtio,br1,br1,59008,10.10.10.8,
255.255.255.0,10.10.10.1,172.16.2.8,255.255.255.0,none
```

解释配置文件信息，第一个关键字意义如下：

- vm 使用 qcow2 方式，并通过 guestfish 进行文件系统的扩展。
- vmLvm 使用 lvm 方式。
- vmCpOnly 使用 qcow2 方式，直接复制镜像。

第二个关键字为镜像名字，建议名字包含操作系统信息。

后面的配置关键字依次为虚拟机主机名、第一块磁盘大小、第二块磁盘大小、CPU、内存、IP 配置等信息。

使用注意事项如下：

- 如果使用 vm 关键字进行磁盘扩展，第一块磁盘必须大于镜像。
- 每个关键字必须有内容，第二块磁盘也必须有，最小为 1GB。
- IP 地址合法性本脚本不作检查。
- CentOS 7 的虚拟机只支持在 CentOS 7 的宿主机上进行配置。

4）运行脚本。使用如下命令运行脚本，就可以自动生成虚拟机：

```
python virtauto.py
```

virtauto.py 帮助信息如下：

```
#python virtauto.py --help
kvm vm setup script
-h, --help print this
--vg,assige vg name,such as --vg=datavg,vg=vmVG
default vg name is 'datavg' if not assige
# 指定 vg 的名字，目前只支持 datvg，后面可以支持自定义 vg
--config,assige config file name ,such as --config=vm.csv
config file must in same directory and must be csv
default config file name is 'vm.csv' if not assige
# 指定配置文件名字，如果没有指定就是 vm.csv，配置文件必须是 csv 格式
```

1107114476@qq.com
dolphinebook.com

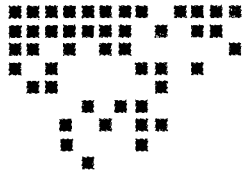
```
--url,give path to download vm images,such as --url=ftp://user1:pass@172.16.1.100/
# 指定镜像下载 url 地址，如果没有指定 url，脚本会认为 /datapool 已经有指定的镜像，如果
  /datapool 下面也没有镜像，就会报错退出
```

17.3　本章小结

本章介绍了单机虚拟化技术的特点，总结了单机虚拟机的优势及应用场景，单机虚拟化适用于性能要求高的业务，通过灵活的自定义宿主机配置，可以为虚拟机提供更高的性能。灾备是单机虚拟化的弱点，单机虚拟化的备份思路主要是做各种快照，本章详细举例了如何通过外部快照来实现虚拟机的备份。

通过本章的介绍，读者可以判断自己那些业务是否适合选择单机虚拟化方式。下一章将介绍另一种形式的虚拟化——集群虚拟化。

1107114476@qq.com
dolphinebook.com



Chapter 18 第 18 章

KVM 虚拟化集群技术与应用场景

本章将为读者介绍 KVM 虚拟化集群在生产环境中的应用场景、架构、配置及实施流程。虚拟化集群的搭建与运维，技术只是一个方面，根据笔者的实践经验，集群的规划设计、搭建流程、设备拓扑资料准备及更新都是虚拟化集群能否稳定运行的关键因素。所以，本章的内容不仅涉及 KVM 虚拟化技术，还涉及商业存储选型与运维、虚拟化集群网络设计与设备选型、虚拟化集群服务器设备选型、虚拟化集群搭建流程与运维规范等多个方面的内容。

18.1 虚拟化集群设计思路与架构

使用虚拟化集群的目的是克服单机虚拟化的局限性，利用技术手段提高虚拟机可用性，最终达到业务不中断或者减少中断时间，确保业务数据更安全的目标。

1. 虚拟化集群介绍

（1）什么是虚拟化集群

虚拟机集群最显著的特征是有共享存储。因为有了共享存储，虚拟机就可以实现非常快速的在线迁移，并在虚拟化层配置高可用。笔者在生产环境使用的集群有两种存储方式，即基于商业存储和基于开源分布式文件系统。

（2）虚拟化集群的高可用和基于应用层高可用的区别

高可用是经常用到的运维技术，在系统、网络、数据库、Web 业务等各个应用层面都有使用。高可用技术至少有主、备两个节点，当主节点故障的时候，迅速切换到备用节点。为

1107114476@qq.com
dolphinebook.com

了避免备用节点误判，有时候还有第三个节点，或者主节点和备用节点共同能访问到的存储空间，用于仲裁判断。应用层面的高可用还有一个特点，就是一般都有浮动 IP，当切换发生的时候，IP 从主节点漂移到备用节点。应用层面的高可用一般切换时间比较快，从几毫秒到几秒，同时，应用层面的高可用一般需要专用软件，如常用的 Keepalived、Heartbeat 等。

虚拟化层面的高可用是虚拟机系统层面的高可用，即当一台计算节点故障的时候，在另外一台计算节点上自动将故障节点上的虚拟机启动起来。注意，如果虚拟机上的业务不能做到开机自启动，即使虚拟机自动启动了，并不能保证业务层面的自动恢复。

另外，还有一个问题，就是即使虚拟机启动起来了，当启动到一半时虚拟机的系统卡住了，也不能及时恢复业务。

虚拟化层高可用的一般业务恢复实际是系统重启的时间，加上业务开机自启动的时间，通常是分钟级别。

虽然虚拟化层高可用有业务不能恢复的风险，业务恢复时间也相对比较长，但是，虚拟化层高可用有个非常巨大的优势，就是不需要在应用层面配置，大大地拓宽了高可用的适用范围，使原来在应用层难以使用高可用技术的应用，也能做到高可用，尤其是在某些专用的软件领域。

其实，虚拟机层高可用和应用层面高可用并不矛盾，在虚拟机的系统里面，也可以配置应用层面的高可用，做这样的配置的时候，注意主、备节点要放置到不同宿主机上。

（3）虚拟化集群后端存储的使用

最早的时候，笔者在生产环境使用的集群都是以机柜为单位的小集群，主要因为笔者一直搭建的是私有云，在 IDC 机房中机柜都是要计算费用的，为了充分利用资源，合理节省成本，私有云的一个设计思想就是能够按照机柜，灵活地扩展或者伸缩。

后来随着虚拟化的规模扩大，也进行了一些基于开源分布式文件系统集群的搭建，基于开源分布式文件系统的集群，规模可以更大，扩展性更好，适用于 KVM 的开源分布式文件系统在第 9 章、第 10 章都有详细的介绍。开源的虚拟化管理平台在本书第 11 章、第 12 章、第 13 章有详细介绍，所以，本章不再介绍开源分布式文件系统及管理平台。

2. 使用虚拟化集群的优势

虚拟化集群相对于单机虚拟化，有以下几点优势：

- 快速的在线迁移（Live Migration），设备、系统维护造成的业务计划内停机时间减少到零。
- 高可用（HA），当一台计算节点故障时，上面的虚拟机可以很快在其他计算节点上启动运行起来，极大缩短计划外停机时间。
- 动态资源调度，业务负载发生变化引起计算节点压力分布不均匀时，可手动或者自动平衡物理机负载。也可在整体压力较低的时间段，将虚拟机集中在部分计算节点上，将不用的计算节点临时关闭，达到节能的目标。

1107114476@qq.com
dolphinebook.com

- 业务快速部署，集群将资源池化，通过和管理平台结合，在集群的容量范围内，业务部署的速度非常快。
- 数据更安全，后端存储采用冗余度更高的商业存储，或者分布式文件系统，数据可靠性可以达到 99.99% 以上。
- 网络速度、可靠性更高，集群网络采用冗余架构，网络设备、网络连接都是双冗余，网络速度更高，可靠性也更高，单台网络设备、单根网线、单个网卡故障都不会引起网络中断。

提示

（1）什么是计划内停机与计划外停机

计划内停机是指可预期、可计划的停机，如定期的维护，提前通告的维护。计划外停机是指突发事件引起的停机事件，如硬件故障，网络 DDoS 攻击等。一般计划内停机因为是提前预知的，会做好预防措施，不会有数据丢失，对业务的损失是比较小的。计划外停机则因为是突发事件，对业务的损失要大很多。运维的重要职责之一就是通过技术手段减少计划外和计划内停机时间。对虚拟化来说，虚拟化集群能够做到在线虚拟机迁移，并且是全冗余设计，需要计划内硬件和软件维护的时候，可以做到计划内停机时间为零。当宿主机发生紧急硬件故障的时候，虚拟机可以很快在其他宿主机上启动起来，所以，虚拟化集群也能有效降低计划外停机。

（2）在线迁移并不是灾备手段

在线迁移实际迁移的是虚拟机的内存，当宿主机发生故障的时候，虚拟机的内存信息已经丢失，这时候是不能再去做虚拟机的在线迁移的。所以，在线迁移解决的是有计划的维护问题，如要升级宿主机内存，可以将宿主机上的虚拟机在线迁移到其他宿主机上，内存升级完成后，再将虚拟机在线迁移回来。

---

3. 集群设计与架构

（1）虚拟化集群设计

为保证虚拟机尽量的在线时间，灵活的扩展，虚拟化集群的设计需要满足以下要求：

- 有共享存储，虚拟机能够在线迁移。
- 通过增加计算节点、存储、网络设备可以横向扩展。
- 没有单点故障，计算节点有多个，商业存储为双控制器，分布式文件系统镜像写多份，网络设备冗余。
- 性能满足要求，并且通过增加设备，性能可以扩展。

（2）虚拟化集群的架构

一套虚拟化集群体系包括以下组成部分：

- 若干计算节点，承载虚拟机的计算、内存、网络资源。

1107114476@qq.com
dolphinebook.com

- 管理节点及管理平台，管理虚拟机的镜像，虚拟机生成、维护、销毁的生命周期，虚拟机的调度。
- 后端存储，存储虚拟机镜像存放。
- 网络设备。

18.2　虚拟化集群的实施流程与技术方案

1. 虚拟化集群的实施流程

虚拟化集群相比单机虚拟化复杂很多，实施不光是技术实施，也需要流程的管控，应将实施过程看作一个项目，按照以下流程实施：

- 前端计算节点方案及设备选型。
- 后端存储方案及设备选型。
- 设备及清单整理。
- 设备摆放及网络拓扑图绘制。
- 部署搭建及测试。
- 日常运维。

2. 前端计算节点技术方案

虚拟化集群前端计算节点可以使用普通机架式服务器，也可以使用刀片服务器。

（1）机架式服务器作为计算节点的优缺点

使用机架式服务器作为计算节点的优点如下：

- 架构简单，安装配置方便。
- 扩展节点数量和升级较为容易。
- 成本有一定的优势。

其缺点如下：

- 随着节点数量的增多，占用的机柜空间也在增大，单位机柜服务器密度低。
- 网络结构复杂，每台服务器有公网、私网、存储网。
- 交换机端口数量多，接线容易出错。

（2）刀片服务器作为计算节点的优缺点

使用刀片服务器作为计算节点的优点如下：

- 刀片服务器内置交换机，可以灵活的配置网络。
- 刀片服务器连线简单，占有交换机端口数量少，网络非常简洁。
- 单位机柜服务器密度大。
- 功耗低。
- 刀片服务器冗余电源和风扇，冗余交换模块，是全冗余的架构。

1107114476@qq.com
dolphinebook.com

使用刀片服务器的缺点如下：

❑ 成本较高。

❑ 配置复杂，安装配置需要专业的知识。

❑ 往往需要改造机柜电源，并受限于机柜最高电流。

另外，目前还有一种多节点服务器，就是在 1U 或者 2U 的空间里面，能够容纳 2 ~ 4 台服务器，这些服务器很像是刀片服务器，共享电源和机框，但是网络接口独立。多节点服务器密度介于机架式服务器和刀片服务器之间，使用上和机架式服务器完全一样。

服务器的配置选型，根据笔者的经验，选择比较高的配置，虽然初期投入高，但是长远看，因为能够容纳更多的虚拟机，其实是节省成本的。宿主机在运行一段时间后，往往会发现内存是瓶颈，所以，一开始配置的时候，内存尽量配置大一些。具体宿主机如何选型在第 15 章已经有详细的介绍，本章就不重复介绍了。

3. 后端存储技术方案

虚拟化集群的后端存储可以使用商业存储和分布式文件系统，商业存储有 3 类，即 NAS、IP SAN、FC SAN。

（1）NAS 共享存储

NAS（NETWORK ATTACHED STORAGE）即网络附加存储，网络上直接挂接的存储设备，相当于一个网络文件共享服务器。

测试环境可以用一台普通的主机模拟 NAS，只要这台主机有自己的磁盘和文件系统，并且对外提供访问文件系统的接口。最常见的 NAS 有 Linux 下的 NFS 和 windows 下的 CIFS。

（2）IP SAN 共享存储

SAN（STORAGE AREA NETWORK）即存储区域网络，主要是基于 TCP/IP 的网络来实现数据存取，即传输介质为 IP 网络。通过 IP 网络将计算节点和存储设备连接起来，计算节点通过发送 Block I/O 的请求到存储设备，最常见的就是用 ISCSI 技术，计算节点通过 SCSI 协议发出读取数据的请求，并用 TCP/IP 包封装 SCSI 包，就可以在 TCP/IP 网络中进行传输，即 SCSI over TCP/IP。

测试环境也可以用普通服务器模拟 ISCSI 存储。

（3）FCSAN 共享存储

FC(Fibre Channel 光纤通道)SAN 类似于 IP SAN，只是以光纤作为传输介质，性能较高，目前使用最广。计算节点上安装光纤接口的 HBA（Host Bus Adapter，提供服务器内部的 I/O 通道与存储系统的 I/O 通道之间的物理连接）卡，为了冗余 HBA 卡一般有两块，分别接两台光纤交换机，存储一般有两个控制器，也分别接两台光纤交换机，达到全容易的目标。FC SAN 计算节点直接将 I/O 请求通过 FC 网络发送到存储设备，性能非常高。

（4）生产环境如何选择存储类型

在实际部署的生产环境中，选择存储类型，取决于以下几个因素：

1107114476@qq.com
dolphinebook.com

- 业务性能及可靠性需求。
- 预算。
- 运维对技术熟悉程度。

一般来说，对性能要求非常高的业务，使用 FC SAN 存储，FC SAN 存储也是成本最高的一种方案。如果业务性能需求稍低，可以使用 NAS、IP SAN 的存储，NAS、IP SAN 的存储是性价比较高的方式。如果业务主要是 CPU 消耗型的，可以考虑使用分布式文件系统，本书第 9 章介绍的 DRBD、GlusterFS，第 10 章介绍的 Ceph，工作都很稳定，但是性能相对比较低，很适合 CPU 消耗型的虚拟机。

关于 NFS 和 ISCSI，业内一直有争论，NFS 配置简单，但是，因为是应用层的协议，有人认为性能低，其实商业存储做了许多优化，性能也不见得比 ISCSI 差。如何选择主要取决于预算、运维的技术习惯、具体的存储品牌型号，笔者生产环境喜欢使用 ISCSI 存储。

提示　商业存储在选择的时候，尽量选择双控制器，因为选择商业存储就是为了性能和可靠性，如果是单控制器，就不能做到高可用，在控制器层面没有容灾。另外，目前万兆已经是趋势了，在选择 IP 存储的时候，推荐选择万兆网络。

4. 虚拟化集群设备与规划

（1）设备清单

设备清单是技术方案的重要组成部分，集群搭建前，先将所有的设备整理成表格，方便在搭建的时候查询，也方便以后出问题的时候核对。清单应包含搭建集群的所有设备，包括设备型号、配置、数量等，如表 18-1 所示。

表 18-1　集群设备清单

资源准备			备　注
设备信息	计算节点	配置	
		数量	
	交换机	规格	
		数量	
	存储	数量	
	网线	数量	
网络及机房信息	IP 地址	数量	
	机柜	数量	

（2）设备摆放连接图

虚拟化集群不仅是技术比单机虚拟化复杂，就是物理设备的摆放、网线的连接也要复杂很多。在上线之前，根据机柜的高度和每个机柜的用电量，设计好设备摆放连接图，列出每

1107114476@qq.com
dolphinebook.com

个设备的具体 IP 信息，方便现场工程师按照连接图操作，也方便有问题的时候排查，在网络拓扑发生变化的时候，应及时更新图表。

如表 18-2 所示为集群机柜后视图。

表 18-2 机柜后视图

XX 集群机柜后视图					
外网 交换机					
内网 交换机					
带外 交换机					
存储 交换机 A					
存储 交换机 B					
计算节点	外网 IP	内网 IP	带外 IP	存储网 IP1	存储网 IP2
计算节点 1					
计算节点 10					
管理节点					
存储节点 A					

18.3 虚拟化集群搭建

18.3.1 计算节点配置与管理

1. 前期准备

在搭建集群前，需要收集业务方对虚拟机数量与配置的要求，根据需求评估集群的数量、计算节点的配置、后端存储的容量，综合评估需要搭建什么样的集群才能满足业务方的要求。

为了后面章节容易展开，本节以每台计算节点包含 4 个千兆网口、一个带外管理口，共 10 台计算节点、一台管理机，使用商业的 Dell EqualLogic ISCSI 存储，通过 virt-manager 作为管理平台为例，介绍集群的架构以及具体的搭建步骤。

2. 计算节点配置

1）环境准备。首先需要对计算节点进行必要的配置，包括计算节点系统更新、KVM 相关包的安装，笔者生产环境计算节点使用的是 CentOS 6.x，命令如下：

```
# yum update -y
# 升级到最新系统，并安装虚拟化组件
# yum -y groupinstall Virtualization "Virtualization Client" "Virtualization
  Platform" "Virtualization Tools" storage-client-iscsi
# 安装多路径服务，ISCSI 节点服务、python 开发包
# yum -y install device-mapper-multipath virt-v2v iscsi-initiator-utils kvm
  libvirt.x86_64 libvirt-python.x86_64 python-virtinst.noarch kernel
```

1107114476@qq.com
dolphinebook.com

2）新建运行相关程序所需的用户和组，设置相应的 sudo 权限，操作步骤略。

3）设置监控、日志收集的计划任务。设置系统开机自启动服务，如 messagebus、libvirtd、iptables、xinetd、portmap、multipathd、iscsid、iscsi、ntpd 等，关闭其他不需要的服务，如 ip6tables、gmp、cups 等服务以节省系统资源开销，操作步骤略。

4）配置 /etc/resolv.conf，最好是自己搭建的 DNS，操作步骤略。

5）同步计算节点的时间并写入到硬件时钟中。

```
# cp -f  /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
# /usr/sbin/ntpdate pool.ntp.org
# /usr/sbin/hwclock --systohc --directisa
```

6）修改 /etc/ssh/sshd_config 文件。增强系统安全性，包括 Port、PermitRootLogin、AllowTcpForwarding、AllowGroups 等，操作步骤略。

7）计算节点针对 KVM 的调优策略。

```
# /usr/bin/tuned-adm profile enterprise-storage
```

tuned 是 CentOS 6 推出的调优工具，可以简单、动态地调整系统。默认的 tuned 软件包包含 9 个调优配置文件，对 laptop、desktop、server 系统的省电策略、磁盘调度算法、缓存、内存脏页回写等进行调节。

这里选择 enterprise-storage 可以极大提高 I/O 吞吐量的模式。切换磁盘调度模式为 deadline，在非根和非 boot 分区上关闭 I/O barriers（极大地提高 I/O 吞吐量，但是掉电后会产生文件丢失），启用 4 倍 readahead。

8）关闭 IPv6。

```
# echo "install ipv6 /bin/true" >>/etc/modprobe.d/disable_ipv6.conf
# sed -i 's/^NETWORKING_IPV6=yes/^NETWORKING_IPV6=no/' \
 /etc/sysconfig/network
# echo "IPV6INIT=no" >>/etc/sysconfig/network
```

9）修改内核参数。内核参数调整其中最重要的两个参数如下：

```
net.ipv4.conf.$NIC2.rp_filter = 0
net.ipv4.conf.$NIC3.rp_filter = 0
```

集群设计的架构是第一块网卡为外网、第二块网卡为内网、第三、四块网卡为同一网段的连接存储的网卡，不设置此参数将导致两块网卡不能同时保持与存储的连接。

10）另外，还包括 logrotate、ntp、yum 等相关设置，请读者根据实际情况进行配置。

11）关闭 virbr0。安装和开启 Libvirt 的服务后会自动生成 virbr0，宿主机上所有的虚拟机都通过 virbr0 连起来，默认情况下 virbr0 使用的是 NAT 模式，所以，这种情况下虚拟机通过宿主机才能访问外部，生产环境一般虚拟机使用的是 bridge 方式（网桥），所以，virbr0 不

1107114476@qq.com
dolphinebook.com

是必须的，通过如下方式可以关闭 virbr0：

```
# virsh net-autostart default --disable
# virsh net-destroy default
# virsh net-undefine default
```

12）配置 ISCSI。配置 /etc/iscsi/iscsid.conf 文件，命令如下：

```
# vim /etc/iscsi/iscsid.conf
# 使用 chap 协议
node.session.auth.authmethod = CHAP
# 输入 ISCSI 存储用户密码
node.session.auth.username = username
node.session.auth.password = password
#ISCSI 的发现，也使用 CHAP 协议，输入用户密码
discovery.sendtargets.auth.authmethod = CHAP
discovery.sendtargets.auth.username = username
discovery.sendtargets.auth.password = password
# 重启 ISCSI 服务
# /etc/init.d/iscsid restart
# 先发现 ISCSI 设备，然后挂载 ISCSI 存储
# iscsiadm -m discovery -t st -p $STORAGE        // $STORAGE 为存储的 IP
# iscsiadm -m node -p $STORAGE -l                // 将存储挂载到本地，此时在本地应该可以
                                                 // 看到两块新增加的磁盘
```

13）配置计算节点的外网网桥和虚拟机的外网子网桥。假设外网为 eth0，内网为 eth1，外网子接口为 111，内网子接口为 222，子接口数值不是随便配置的，必须要和接入层交换机的 VLAN ID 一致。

```
// 配置计算节点外网网桥
# vim /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
TYPE=Ethernet
BRIDGE=br0
# vim /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE=br0
TYPE=Bridge
BOOTPROTO=static
IPADDR=$IPADDR
NETMASK=$netmask
ONBOOT=yes
DELAY=0
// 配置外网子接口网桥，主要用于虚拟机的外网桥接
# vim /etc/sysconfig/network-scripts/ifcfg-eth0.111
DEVICE=eth0.111
BOOTPROTO=static
ONBOOT=yes
TYPE=Ethernet
```

1107114476@qq.com
dolphinebook.com

```
BRIDGE=br0.111
# vim /etc/sysconfig/network-scripts/ifcfg-br0.111
DEVICE=br0.111
TYPE=Bridge
BOOTPROTO=static
IPADDR=$IPADDR
NETMASK=$netmask
ONBOOT=yes
DELAY=0
//配置计算节点内网网桥
# vim /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
BOOTPROTO=static
ONBOOT=yes
TYPE=Ethernet
BRIDGE=br1
# vim /etc/sysconfig/network-scripts/ifcfg-br1
DEVICE=br1
TYPE=Bridge
BOOTPROTO=static
IPADDR=$IPADDR
NETMASK=$netmask
ONBOOT=yes
DELAY=0
//配置内网子接口网桥，主要用于虚拟机的内网桥接
# vim /etc/sysconfig/network-scripts/ifcfg-eth1.222
DEVICE=eth1.222
BOOTPROTO=static
ONBOOT=yes
TYPE=Ethernet
BRIDGE=br1.222
# vim /etc/sysconfig/network-scripts/ifcfg-br1.222
DEVICE=br1.222
TYPE=Bridge
BOOTPROTO=static
IPADDR=$IPADDR
NETMASK=$netmask
ONBOOT=yes
DELAY=0
```

为什么内网、外网在生产环境要划分 VLAN？划分 VLAN 的目的有两个，第一是将宿主机网络和虚拟机网络隔离，一般不允许宿主机和虚拟机通信；第二是虚拟机之间也可以通过 VLAN 隔离，保证不同的业务在不同的 VLAN 中。

14）配置 Multipath 实现 LUN 设备名称的持久化。为了提供冗余，服务器会通过多条线路和存储连接，那么在系统中就会发现多个存储设备，实际上是一块存储设备。Multipath 多路径服务主要解决这个问题，通过 Multipath 服务，可以将特征形同的多块存储设备聚合成

1107114476@qq.com
dolphinebook.com

一块设备，然后给系统使用。Multipath的配置过程如下：

```
//安装multipath所需的安装包
# yum install device-mapper-multipath iscsi-initiator-utils -y

//加载multipath模块
# modprobe dm-multipath
# modprobe dm-round-robin
# modprobe dm-service-time
//新建访问接口文件，接口文件保存在/var/lib/iscsi/ifaces目录中
# iscsiadm  -m iface -I eth2 -o new
# iscsiadm  -m iface -I eth3 -o new
//配置iface，通过eth2,eth3两个接口访问
# iscsiadm -m iface -I eth2 --op=update -n iface.net_ifacename -v eth2
# iscsiadm -m iface -I eth3 --op=update -n iface.net_ifacename -v eth3
# 查看文件内容
# cat /var/lib/iscsi/ifaces/eth2
iface.iscsi_ifacename = eth2
iface.net_ifacename = eth2
iface.transport_name = tcp
iface.vlan_id = 0
iface.vlan_priority = 0
iface.iface_num = 0
iface.mtu = 0
iface.port = 0
iface.tos = 0
iface.ttl = 0
iface.tcp_wsf = 0
iface.tcp_timer_scale = 0
iface.def_task_mgmt_timeout = 0
iface.erl = 0
iface.max_receive_data_len = 0
iface.first_burst_len = 0
iface.max_outstanding_r2t = 0
iface.max_burst_len = 0
# cat /var/lib/iscsi/ifaces/eth3
iface.iscsi_ifacename = eth3
iface.net_ifacename = eth3
iface.transport_name = tcp
iface.vlan_id = 0
iface.vlan_priority = 0
iface.iface_num = 0
iface.mtu = 0
iface.port = 0
iface.tos = 0
iface.ttl = 0
iface.tcp_wsf = 0
iface.tcp_timer_scale = 0
iface.def_task_mgmt_timeout = 0
iface.erl = 0
```

1107114476@qq.com
dolphinebook.com

```
iface.max_receive_data_len = 0
iface.first_burst_len = 0
iface.max_outstanding_r2t = 0
iface.max_burst_len = 0
```

通过 scsi_id 命令，查询 ISCSI 设备 id 号，命令如下：

```
# scsi_id -g -u -s /dev/xxx  //xxx 表示通过 iscsiadm 发现并挂载到本地的磁盘
36000eb3d3cc8a1a3000000000000002a
# 编辑 multipathh.conf 配置文件 /etc/multipath.conf
# vim /etc/multipath.conf
defaults {
        find_multipaths yes
        user_friendly_names no
        path_grouping_policy multibus
        polling_interval        4
        udev_dir                /dev
        multipath_dir           /lib64/multipath
        verbosity               2
        path_selector           "service-time 0"
        path_checker            directio
        getuid_callout          "/lib/udev/scsi_id --whitelisted --device=/dev/%n"
        prio                    const
        failback                manual
        rr_min_io               1000
        rr_weight               uniform
        no_path_retry           fail
        queue_without_daemon    yes
        flush_on_last_del       no
        max_fds                 65536
}
devices {
        device {
                vendor                  "EQLOGIC"
                product                 "100E-00"
                path_grouping_policy    multibus
        }
        device {
                vendor                  "LEFTHAND"
                product                 "iSCSIDisk"
        }
}
blacklist {
        devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
        wwid "*"
        device {
                vendor  "IBM"
                product "3S42"       #DS4200 Product 10
        }
}
blacklist_exceptions {
```

1107114476@qq.com
dolphinebook.com

```
        wwid                    36000eb3d3cc8a1a3000000000000002a
                                                    // 此处填写挂载的设备 id
        device {
                vendor  "EQLOGIC"
                product "100E-00"
        }
}
multipaths {
        multipath {
                wwid                    36000eb3d3cc8a1a3000000000000002a
                alias                   ipsanmp1        // 挂载设备的别名
        }
}
# /etc/init.d/multipathd restart  // 此时在 /dev/mapper/ 下面会多出一个卷设备 ipsanmp1
```

3. 管理节点配置

（1）虚拟化集群的存储方式

使用 ISCSI 存储，管理节点和计算节点将 ISCSI 块设备挂载后，虚拟机镜像存储方式可以采用以下方式：

- 使用 qcow2 方式，格式化成文件系统，然后虚拟机以 qcow2 的方式存储。
- 使用 lvm 方式，划分 PV、VG，每个虚拟机的卷划分一个 lv 使用。
- 使用 clvm 方式。

使用 qcow2 方式的优势是可以使用 qcow2 镜像的优点，如精简模式，但是商业存储一般都有精简模式的功能，并且性能会好很多，所以，使用 qcow2 方式并不占有优势，反而因为是基于文件系统，比使用裸设备的性能会低很多，所以，一般不建议在虚拟化集群使用 qcow2 方式。

使用 lvm 方式，因为是裸设备，性能要好很多，并且 lvm 支持在线快照，从块设备层面多了一种灾备方式。lvm 方式有一个缺点，就是不支持对同时读 / 写的验证，读操作的时候，不会造成验证影响，但是，如果是两个计算节点同时写一个虚拟机的 lvm，后果将非常严重。要避免这个问题，需要在管理平台上进行严格的限制。

clvm 方式就是解决集群中多个节点同时读 / 写 lvm 的矛盾，在块设备层做了限制，但是因为做了限制，会对性能有一定影响。

在生产环境使用 lvm 方式，经过验证，许多开源的虚拟化管理平台，也使用的是 lvm 方式。使用 lvm 方式，主要是出于性能考虑。

（2）管理节点的配置

管理节点主要是虚拟机模板镜像的管理，配置过程如下：

① 先按照计算节点的配置步骤配置管理节点，配置完毕后对 /dev/mapper/ipsanmp1 创建 PV、VG，LVM 元数据包含了 LVM 的配置信息，默认情况下，每个 PV 中保存一份元数据，元数据以 ASCII 格式存储，元数据在系统上会保持备份，在 /etc/lvm 目录下保持最新版

1107114476@qq.com
dolphinebook.com

本，在 /etc/lvm/archive 目录下保持旧版本，其次，在磁盘的开始扇区中也保持元数据。如果存储的是两份元数据，第二份则存储在磁盘的最后面，这样，当通过 PVCREATE 等操作将第一份数据覆盖后，可通过第二份数据进行恢复，在集群中将元数据设置保存为两份，命令如下：

```
# pvcreate  --metadatacopies  2 /dev/mapper/ipsanmp1
# vgcreate vmVG  $PV
```

② 修改 /etc/lvm/lvm.conf 的 locking_type 参数，管理机的 locking_type=1，计算节点为 locking_type=4，1 表示对 vg 有读 / 写权限，4 为只读权限。

```
# vim /etc/lvm/lvm.conf
locking_type = 1
```

③ 此时，在管理机和计算节点都能看到创建的 PV 和 VG，接下来，需要修改主机名和 hosts 文件，将管理机节点和所有计算节点的主机名与其对应的内网 IP 配置到 /etc/hosts 文件中，虚拟机在迁移过程中是以主机名为依据的，主机名不正确则无法迁移虚拟机。

4. 认证配置

虚拟机的在线迁移，可以通过 QEMU 层进行，也可以通过 Libvirt 迁移，Libvirt 层的迁移支持加密和认证，安全性更好，一般在生产环境使用 Libvirt 层迁移的方式。Libvirt 层的迁移需要配置认证，认证有以下 3 种方案：

- TCP 方式，迁移流量直接走 TCP 通道，没有加密。
- SSH 方式，迁移流量走 SSH 通道。
- TLS 方式，TLS 是基于证书的加密，一般推荐在生产环境使用这种方式。

3 种认证方式的配置方法如下。

（1）TCP 方式配置

TCP 方式是比较简单的一种配置方式，通常用于测试环境，配置步骤如下，需要对集群内部所有的计算节点和管理节点配置：

1）配置 Libvirt 服务。

```
# vi /etc/libvirt/libvirtd.conf
...
# 关闭 tls，默认是开启的
listen_tls = 0
# 开启 tcp 认证
listen_tcp = 1
#tcp 的端口配置，默认端口是 16509
tcp_port = "16509"
# 配置是否开启口令认证，默认不开启
auth_tcp = "none"
# 是否开启 mdns 组播，默认关闭
mdns_adv = 0
...
```

1107114476@qq.com
dolphinebook.com

如果需要开启 tcp 的口令认证，也可以采用下面的配置方法，集群中每个节点都要操作。

```
# 配置 /etc/sysconfig/libvirtd 文件中 auth_tcp 关键字为 sasl:
auth_tcp = "sasl"     # 使用 sasl 作为 tcp 的连接认证方式，sasl 是一个用于网络通信协议的安全验证
# 增加 Libvirt 的认证用户
# echo PASSWORD |saslpasswd2 -a libvirt virtadmin@$HOSTNAME -p
```

2）配置 Libvirt 监听方式。配置 Libvirt 监听不管是 TCP、ssh、TLS 都需要配置，命令如下：

```
# 编辑 /etc/sysconfig/libvirtd 文件
 vi /etc/sysconfig/libvirtd
...
```

添加以下内容，使 Libvirt 处于监听状态：

```
LIBVIRTD_ARGS="--listen"
# 编辑 /etc/libvirt/qemu.conf 文件
# vi /etc/libvirt/qemu.conf
...
```

添加以下内容，使虚拟机带外 VNC 默认监听 0.0.0.0：

```
vnc_listen = "0.0.0.0"
...
# 重启服务，使配置生效
# service libvirtd restart
```

验证配置是否生效，命令如下：

```
virsh -c qemu+tcp://node2/system list
```

node2 为同集群内其他计算节点，如果能连接并显示 node2 上的虚拟机信息，说明配置成功。

（2）SSH 方式配置

SSH 方式走的是 SSH 通道，安全性有保证，使用方法如下：

```
qemu+ssh://node2/system list
```

如果生产环境使用这种方式，为了方便管理，建议对集群所有节点直接配置免密码 SSH 认证，如何配置资料非常多，请读者自己查找参考相关资料。

（3）TLS 方式配置

TLS 方式因为要生成证书，配置比较复杂，步骤如下。

1）创建授权证书模板文件。

```
cat certificate_authority_template.info
cn = ks
ca
cert_signing_key
```

1107114476@qq.com
dolphinebook.com

```
expiration_days = 3650
# expiration_days 不指定是 1 年，可以自己指定时间
```

使用 certtool 命令创建私有证书文件，命令如下：

```
umask 277 && certtool --generate-privkey > certificate_authority_key.pem
```

使用模板文件、私有证书创建授权证书，命令如下：

```
certtool --generate-self-signed \
            --template certificate_authority_template.info \
            --load-privkey certificate_authority_key.pem \
            --outfile certificate_authority_certificate.pem
```

删除授权模板文件，授权模板文件在证书生成后就不再需要使用，为了安全起见，删除授权模板文件，命令如下：

```
rm certificate_authority_template.info
```

证书文件放置到正确的位置，默认的位置是 /etc/pki/CA/cacert.pem，将证书传输到第一个集群节点并配置，命令如下：

```
scp -p certificate_authority_certificate.pem someuser@host1:cacert.pem
# 在 host1 上执行以下命令：
mv cacert.pem /etc/pki/CA
chmod 444 /etc/pki/CA/cacert.pem
# 如果开放 selinux 执行以下命令：
restorecon /etc/pki/CA/cacert.pem
```

在其他所有节点上执行同样的操作。

2）客户端证书创建。客户端证书用于一个管理节点管理多台宿主机的情况，操作步骤如下：

① 创建模板文件，管理节点模板文件。

```
cat manager_client_template.info

country = China
state = GD
locality = ZH
organization = KS
cn = webvirt
tls_www_client
encryption_key
signing_key
expiration_days = 3650
```

管理节点 1 模板文件。

```
cat node1_client_template.info
country = China
```

1107114476@qq.com
dolphinebook.com

```
state = GD
locality = ZH
organization = KS
cn = node1
tls_www_client
encryption_key
signing_key
expiration_days = 3650
```

其他所有节点也需要创建模板文件。

② 创建私钥，创建 node1 的私钥。

```
umask 277 && certtool --generate-privkey > node1_client_key.pem
```

其他节点都需要按照同样的方法创建私钥。

③ 创建管理节点的私钥。

```
umask 277 && certtool --generate-privkey > manager_client_key.pem
```

④ 创建公钥，创建 node1 的公钥。

```
certtool --generate-certificate \
            --template node1_client_template.info \
            --load-privkey node1_client_key.pem \
            --load-ca-certificate certificate_authority_certificate.pem \
            --load-ca-privkey certificate_authority_key.pem \
            --outfile node1_client_certificate.pem
```

其他所有的节点都需要创建公钥。

⑤ 创建管理节点公钥。

```
certtool --generate-certificate \
            --template manager_client_template.info \
            --load-privkey manager_client_key.pem \
            --load-ca-certificate certificate_authority_certificate.pem \
            --load-ca-privkey certificate_authority_key.pem \
            --outfile manager_client_certificate.pem
```

公钥放置到目录 /etc/pki/libvirt/clientcert.pem 下，私钥放置到目录 /etc/pki/libvirt/private/clientkey.pem 下，私钥、公钥必须是 400 的权限。传输私钥、公钥到对应的服务器上，命令如下：

```
scp -p node1_client_certificate.pem someuser@node1:clientcert.pem
scp -p node1_client_key.pem someuser@node1:clientkey.pem
#-p 为保持文件修改时间不变
```

在 node1 上执行如下命令：

```
 mv clientcert.pem /etc/pki/libvirt
clientkey.pem /etc/pki/libvirt/private
```

1107114476@qq.com
dolphinebook.com

```
chmod 400 /etc/pki/libvirt/clientcert.pem /etc/pki/libvirt/private/clientkey.pem
```

所有的节点都需要执行相同的操作，然后，在管理节点上通过以下命令验证配置是否成功。

```
virsh qemu+tls://node1/system
```

5. 管理平台

计算节点和管理节点都配置完成后，就可以手工创建 KVM 虚拟机了，最简单的方式就是使用 Virt-Manager，将所有计算节点添加到管理节点的 Virt-Manager 中。Virt-Manager 可以做手工的在线迁移及高可用，使用 virsh 命令也可以操作。为了方便后续管理和大批量的部署，需要将管理平台相关程序部署到各节点中。根据自己的业务情况开发，也可以使用开源平台，本书第 11 章、第 12 章、第 13 章对主流的开源管理平台都有介绍，一般开源平台要和业务结合得比较好，需要根据自己的需求做二次开发，这个不在本书的讨论范围之内，就不做详细介绍了。

18.3.2　网络设计与配置

1. 虚拟化集群物理网络设计与配置

虚拟化集群的外网、内网、存储网核心采用 Port Channel 技术，提供带宽的同时防止因为链路中断引起的 STP 收敛。网络设备选型上以提高集群高可用性和容错性为前提，采用主流的 CISCO、H3C 设备搭建。外网接入交换机一般采用 CISCO 或 H3C 普通的接入交换机，如 H3C S5120 或者 CISCO 2960x 系列，如果数据中心采用 CISCO Nexus 交换机系列的话也可采用 7-5-2 架构。内网接入交换机由于连接存储及主要的集群内网，推荐使用具备虚拟或堆叠功能的交换机以提高冗余容错性，如 H3C 的 IRF 技术、CISCO 堆叠或者 VSS 技术，使用 Nexus 7-5-2 架构的话即采用 vPC 技术，皆可满足需求。内、外网核心交换机皆采用的是数据中心 CLOS 架构的交换机，如 CISCO 的 Nexus 7000 系列或者 H3C 125 系列交换机，整个数据中心采用大二层架构，便于虚拟主机的漂移和灵活性。

什么是 vPC 和 VPC

vPC（virtual port channel）是 CISCO Nexus 的新技术，主要作用是可以跨不同设备的 Port Channel 技术，并且收敛更快。

VPC（Virtual Private Cloud）虚拟私有云，是公有云上的一个网络概念。

用户选择公有云最大的顾虑是安全问题，公有云安全包含以下两个层面。

① 数据安全：数据会不会被窃取，会不会丢失。

② 服务安全：因为使用公有云，依靠的是公有云的服务支持，用户会担心，如果公有云出现大面积的问题，是否会对自己的服务造成影响。

1107114476@qq.com
dolphinebook.com

VPC实际上是基于网络的安全解决方案，VPC将同一用户的云主机隔离在一个逻辑网络内部，只有用户自己能访问自己的私有网络。

VPC和VLAN的区别

VPC是应用层的概念，往往和用户的账号绑定。VLAN是网络二层的概念，两者的概念很像，都是隔离，但是，VPC主要是公有云不同用户之间的隔离，VLAN是隔离网段和广播流量，VPC有可能包含多个VLAN。

VPC往往使用SDN技术。用户可以在自己的VPC内部创建自己的子网，按照自己的需求分配IP、创建路由表、配置安全规则等。

VPC的使用场景

大量使用内网IP地址，只将需要暴露的服务通过NAT映射出去。

将VPC和自己的企业内网或者自己的私有数据中心打通，将公有云变成自己私有云的一部分。

通过VPC的使用，可以从网络层面大大提高用户使用公有云的安全性，做到了只需要暴露的暴露，不需要暴露的完全就在网络层面关闭。

---

2. 宿主机网络设计与配置

物理机配置4块网卡，前两块为内、外网卡，3、4块为存储网卡。内、外网分别配置网桥，并根据接入层交换机的VLAN ID创建子接口和子网桥，虚拟机的所有网卡分别桥接到对应的子接口上才能正常通信。具体配置可以参考18.3.1节中关于网桥的部分内容。

在18.3.1节中，需提前去申请IP段，包括外网段（即宿主机和虚拟机外网段）、内网段、存储段以及业务内网段（即虚拟机内网段）。一般会配置两个业务内网段，主要是为了应对特殊需求，如业务上有时需要有两个内网IP。

关于虚拟化实践中的IP地址问题。在虚拟化实践中，往往因为迁移到虚拟化环境，在单机柜内部，系统密度大大提高。如果是物理机，一个机柜最多十几台服务器，消耗十几个IP地址。但是，因为迁移到虚拟化环境，一个机柜内部可以容纳几十甚至上百个系统，需要消耗一大批IP地址。如果IP地址是内网IP还好处理，做好IP地址规划就可以；如果需要的是公网IP，就需要购买大量的IP地址，这也是一项很大的费用。但是从全盘考虑，同虚拟化节省的大量硬件成本对比，IP地址费用只是一小部分。权衡利弊，IP地址购买可以带来更大的收益。

18.3.3 虚拟机迁移管理

虚拟机的迁移为服务器的虚拟化和日常管理提供了极大的方便，本节为读者介绍基于共享存储和本地存储的虚拟机动态、静态迁移方法。

1107114476@qq.com
dolphinebook.com

1. 虚拟机的动态迁移

（1）什么是动态迁移

动态迁移（Live Migration）也叫在线迁移（Online Migration），即保证不中断业务的情况下将虚拟机从一台宿主机迁移到另外一台宿主机，整个过程对用户是透明、无影响的。通常，在遇到宿主机负载高、故障、配置升级并且虚拟机业务不可中断的情况下，需要动态迁移虚拟机。

动态迁移又可分为基于共享存储和基于本地存储两种方式，基于共享存储的动态迁移在迁移过程中虚拟机的磁盘因为存放在共享存储中，所以，虚拟机磁盘不需要迁移，只需要将存放在虚拟机内存中的数据迁移到目标宿主机上即可，如图 18-1 所示为共享存储的虚拟机迁移示意图。

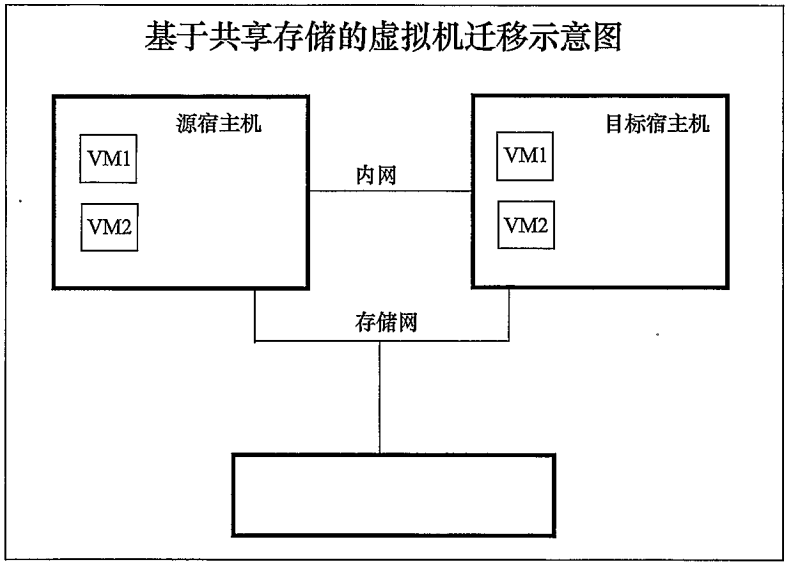


图 18-1　基于共享存储的虚拟机迁移示意图

（2）基于共享存储的动态迁移过程

基于共享存储的动态迁移过程，在源宿主机上执行迁移命令 virsh migrate，10.10.10.20 为目标宿主机的内网 IP。

```
# virsh migrate --live --copy-storage-all --unsafe --persistent centos1
  qemu+ssh://10.10.10.20/system
// centos1 运行在共享存储的 lvm 上，需要注意迁移前必须保证目前宿主机的 lvm 卷是激活的，可以通过
// lvchange -ay lvmname，不执行将会报出找不到 lvmname 的提示，虚拟机磁盘如果不是基于 lvm 的，
// 则不需要此步骤
```

此时，在目标宿主机上可以看到虚拟机目前处于 paused 状态，命令行显示如下：

```
# virsh list
```

1107114476@qq.com
dolphinebook.com

```
 Id    Name                           State
----------------------------------------------------
 4     centos1                        paused
```

在目标宿主机上还可以看到虚拟机迁移进度，命令行显示如下：

```
#tail -f /var/log/libvirt/qemu/centos1.log
isa-serial,chardev=charserial0,id=serial0 -device usb-tablet,id=input0 -vnc
0.0.0.0:0 -vga cirrus -device intel-hda,id=sound0,bus=pci.0,addr=0x4 -device
hda-duplex,id=sound0-codec0,bus=sound0.0,cad=0 -incoming tcp:[::]:49152 -device
virtio-balloon-pci,id=balloon0,bus=pci.0,addr=0x7 -msg timestamp=on
char device redirected to /dev/pts/1
Receiving block device images
Completed 100 %
```

当虚拟机迁移完成后，可以看到虚拟机在源宿主机上变为关机状态，在目标宿主机上变为 running 状态，显示如下：

```
# virsh list -all //源宿主机
 Id    Name                           State
----------------------------------------------------
 -     centos1                        shut off
 # virsh list    //目标宿主机
 Id    Name                           State
----------------------------------------------------
 4     centos1                        running
```

在迁移过程中一直 ping 虚拟机的 IP，可以观察到在切换宿主机的时候 ping 会卡顿，有时会丢几个包。如果虚拟机中部署有数据库，应用程序最好有重连机制。

动态迁移基于共享存储设备，目的是加速迁移的过程，尽量减少宕机时间。但是，在某些情况下需要进行基于本地存储的虚拟机的动态迁移，这就需要存储块动态迁移技术，简称块迁移。

基于本地存储的动态迁移和基于共享存储的动态迁移过程类似，只是本地存储的动态迁移同时还需要将虚拟机的本地磁盘迁移到目标宿主机，迁移过程较长。目前，在云计算中有基于本地存储的动态迁移公有云，也有基于共享存储动态迁移的公有云，不同厂商从成本和技术方面有不同的考量。

2. 虚拟机的静态迁移

静态迁移也叫离线迁移（Offline Migration），就是在虚拟机关机的情况下从一台宿主机迁移到另外一台宿主机。虚拟机静态迁移又可分为基于共享存储的静态迁移和基本本地存储的静态迁移，基于共享存储的虚拟机静态迁移，虚拟机磁盘是建立在共享存储上并且是关机状态的，所以，并不需要迁移内存的数据，迁移瞬间完成，这种迁移方式简单易行，适用于对服务可用性要求不严格的场合，如图 18-2 所示为基于本地存储的虚拟机迁移示意图。

1107114476@qq.com
dolphinebook.com

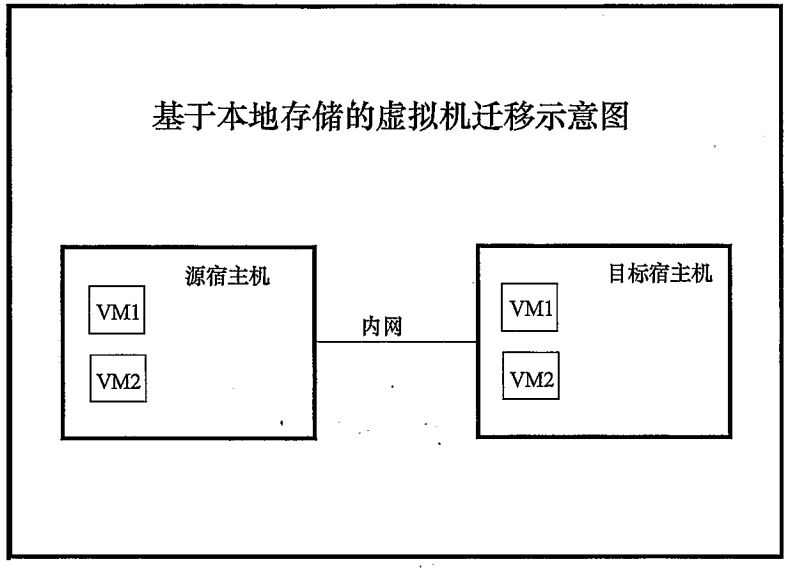


图 18-2　基于本地存储的虚拟机迁移示意图

基于共享存储的静态迁移步骤如下。

源宿主机上线关闭虚拟机，并备份虚拟机的 xml 文件。

```
virsh destroy centos1
virsh dumpxml centos1 > centos1.xml
scp centos1.xml 10.10.10.10:/root/        //将虚拟机 xml 配置文件复制到目标宿主机上
```

目标宿主机执行如下命令：

```
lvchange -ay /dev/vmVG/lvmname            //虚拟机磁盘文件为 lvmname
virsh define centos1.xml                  //根据 xml 配置文件创建虚拟机
# virsh list --all
 Id    Name                           State
----------------------------------------------------
 -     centos1                        shut off
virsh start centos1                       //开启虚拟机
```

关于本地存储的静态迁移需要在虚拟机关机的情况下将镜像和相应的配置文件复制到目标宿主机上，过程与共享存储的静态迁移类似，只是多了一步复制虚拟机镜像的步骤。

18.3.4　虚拟化集群的备份与恢复

虚拟化集群的备份因为数据都是在后端存储上，所以，集群的备份就是存储上数据的备份。开源分布式文件系统因为都是写多份，或者有自己的解决方案。

商业存储的备份有两种思路，一种思路是使用普通的廉价存储服务器，这种方案成本

1107114476@qq.com
dolphinebook.com

低，但是备份过程时间长、效率低；另一种思路是利用商业存储的功能，采用多台存储同步或者异步的方式进行备份。

本节介绍商业存储的备份方案，商业存储以 Dell EqualLogic 为例。

1. 存储服务器备份方案

存储服务器方案，使用一台大于或等于商业存储容量的存储服务器，作为商业存储的备份，这里称为备份机，计算、管理节点通过 ISCSI 挂载备份机，利用商业存储的快照机制，管理机挂载快照，然后，将快照内容复制到存储服务器中，再删除快照，如图 18-3 所示为集群大容量磁盘备份方案的示意图。

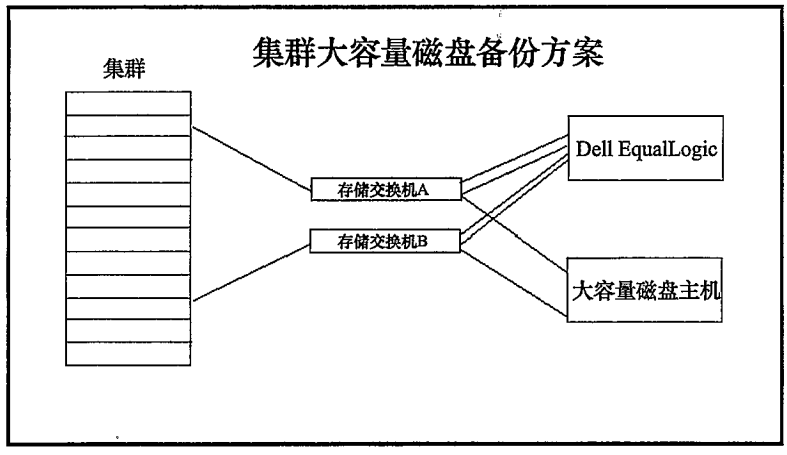


图 18-3 集群大容量磁盘备份示意图

（1）备份机上的具体设置

1）修改内核参数文件 /etc/sysctl..conf，命令如下：

```
net.ipv4.conf.default.rp_filter = 0
net.ipv4.conf.eth0.rp_filter = 0
net.ipv4.conf.eth1.rp_filter = 0
# sysctl -p
```

2）安装 iscsi 工具，命令如下：

```
yum install scsi-target-utils sg3_utils sg3_utils-libs pere-Config-General libibverbs -y
```

3）编辑 target.conf 文件，将磁盘共享出来。

```
vim /etc/tgt/targets.conf
<target iqn.2015-03-02.com.sdg:kvm-storage1>
    backing-store /dev/sdb                  //备份主机上的 sdb 盘共享出来
    write-cache off
    initiator-address 10.10.10.0/24         //只有 10.10.10.0/24 网段可以挂载存储
</target>
```

1107114476@qq.com
dolphinebook.com

4）开启 tgtd 服务，命令如下：

```
# service tgtd start
# chkconfig tgtd on
```

具体实施时，在集群中创建一台虚拟机专门用于管理集群备份和恢复使用，此虚拟机将同时挂载 Dell EqualLogic 快照和备份机共享出来的磁盘。

（2）挂载备份机磁盘和存储快照

1）挂载备份机的磁盘，命令如下：

```
# iscsiadm -m discovery -t st -p 10.10.10.50  //10.10.10.50 为备份机的 ip
10.10.10.50:3260,1 iqn.2015-03-02.com.sdg:kvm-storage1
10.10.10.50:3260,1 iqn.2015-03-02.com.sdg:kvm-storage1
//挂载备份机共享出来的大容量磁盘
# iscsiadm -m node -T iqn.2015-03-02.com.sdg:kvm-storage1 -p 10.10.10.50 -l
```

2）挂载 Dell EqualLogic 的快照，命令如下：

```
iscsiadm -m discovery -t st -p 10.10.10.220
10.10.10.220:3260,1 iqn.2001-05.com.equallogic:0-1cb196-xxx
10.10.10.220:3260,1 iqn.2001-05.com.equallogic:0-1cb196-xxx
# iscsiadm -m node -T iqn.2001-05.com.equallogic:0-1cb196-xxx -l
```

使用 scsi_id 获取快照和备份机磁盘的 wwid，命令如下：

```
# scsi_id -g -d /dev/sdc  //sdc 为挂载的 EQL 快照
36019cb71a0c1426cb80e85e291627a1c
# scsi_id -g -d /dev/sdd  //sdd 为挂载备份机共享出来的磁盘
1IET     00010001
```

3）配置 /etc/multipath.conf，部分内容如下：

```
blacklist_exceptions {
        wwid                    36019cb71a0c1426cb80e85e291627a1c
        wwid                    "1IET     00010001"
        device {
                vendor  "EQLOGIC"
                product "100E-00"
        }
}
multipaths {
        multipath {
                wwid                    36019cb71a0c1426cb80e85e291627a1c
                alias                   ipsanmp1_snapshot
        }

        multipath {
                wwid                    "1IET     00010001"
                alias                   backupstorage
        }
```

1107114476@qq.com
dolphinebook.com

启动 multipathd 服务，命令如下：

```
# /etc/init.d/multipathd restart
# chkconfig multipathd on
```

4）在 backupstorage 磁盘上创建和快照上名字一样的 PV、VG、LV，命令如下：

```
创建 PV
# pvcreate /dev/mapper/backupstorage
创建 VG
# vgcreate langchaoSTORAGE /dev/mapper/backupstorage
激活 vmVG
# vgchange -ay /dev/vmVG
根据 EQL 快照上 lvm 的名字、数量和大小，在备份机上创建一样的名字、数量和大小
```

5）用 dd 的命令来备份集群的虚拟机，命令如下：

```
dd if=/dev/vmVG/lvmxxx of=/dev/backupstorage/lvmxxx bs=1M
```

恢复的时候需要停止所有虚拟机，在所有的计算节点中修改 multipath 配置文件即可，开启虚拟机。

2. 多台 Dell EqualLogic 备份方案

EqualLogic 复制分为同步复制和异步复制，这里主要演示如何使用异步复制的功能来备份集群，如图 18-4 所示为 Dell EqualLogic 备份方案的示意图。为了描述方便，Dell EqualLogic 下面简称 Eql。

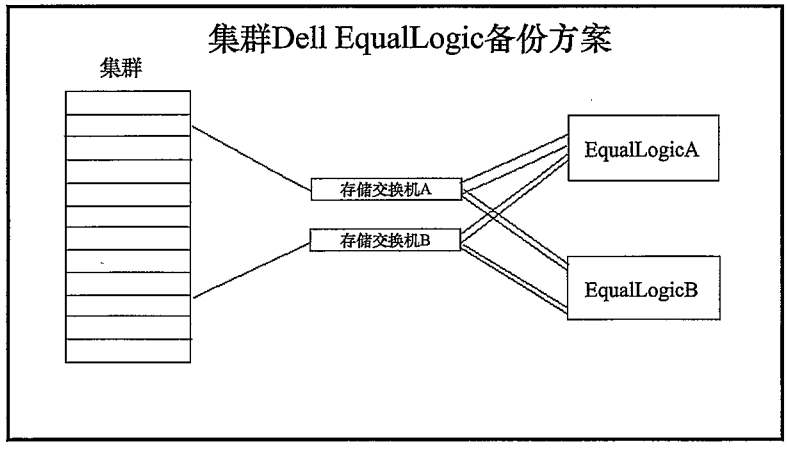


图 18-4 EQL 备份方案的示意图

假设 Eql A 承载着集群，Eql B 为备份，Eql 的异步复制具体步骤如下。

先配置 Eql A，选择 replication 选项，选择 Configure partner，如图 18-5 所示。

填写 Eql B 的 group name（注意大小写）和 IP 地址，* 号为必填项目，如图 18-6 所示为 Eql partner 相关信息。

1107114476@qq.com
dolphinebook.com

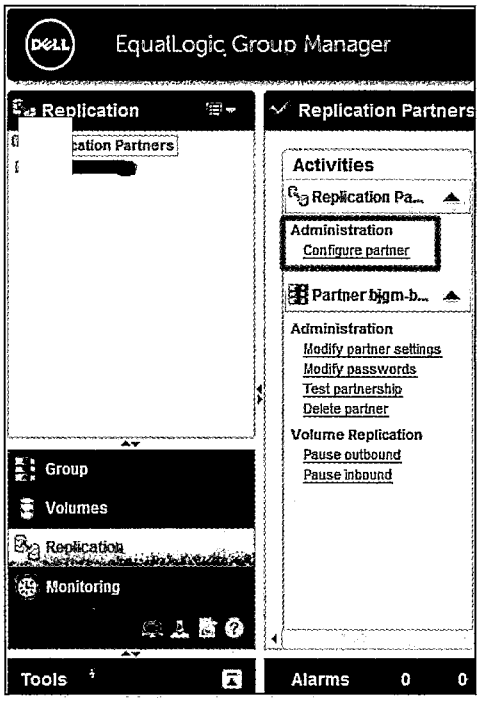


图 18-5　Eql Configure partner

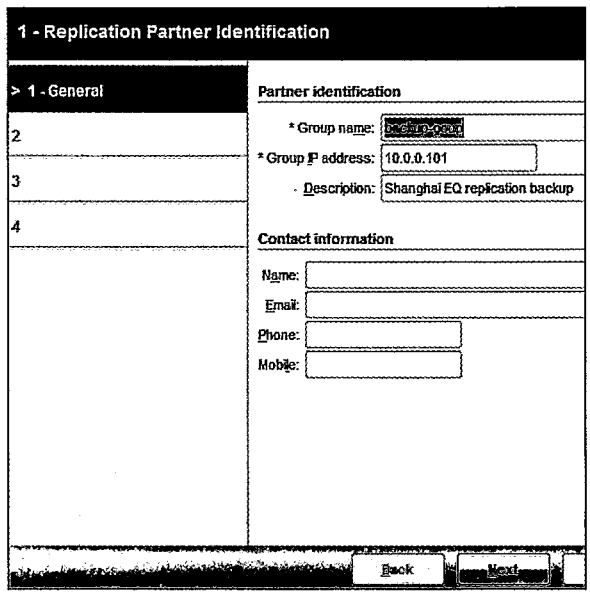


图 18-6　Eql partner 相关信息

填写 Eql partner 的验证信息，如图 18-7 所示。

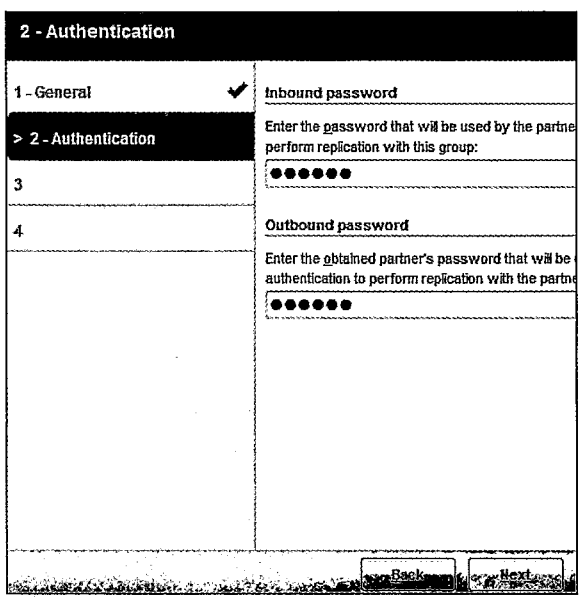


图 18-7　Eql partner 验证信息

1107114476@qq.com
dolphinebook.com

设置 Delegated space，此空间设置的大小不能小于原卷的 2 倍。该空间是用来存放源卷 data 数据的专属空间，如图 18-8 所示为 Eql partner 相关空间的设置。

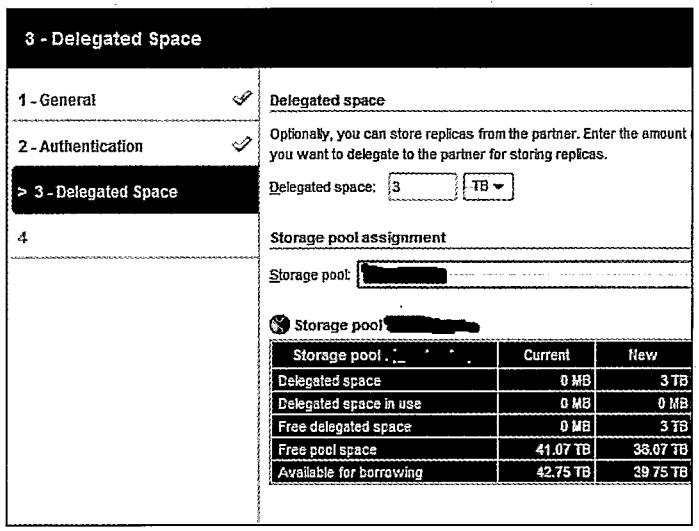


图 18-8 设置 Eql Deleted space

下一步，单击“finish”按钮就完成了 Eql A 的配置。Eql B 的操作步骤与 Eql A 相同，需要注意 Delegated space 的设置，配置完成 Eql B 后，两边的复制关系就建立好了，如图 18-9 所示为建立好的 Eql partner 的示意图。

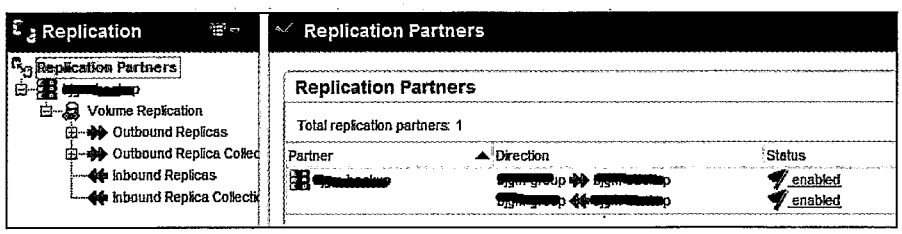


图 18-9 Eql partner 建立完成后的示意图

下面开始配置卷的异步复制，在 Eql A 的卷上右击，选择 Configure replication。

要说明一下，关于每个复制卷都有副本卷保留，大小约为正在使用的卷空间量，在 Eql B 上委派的空间为 15TB（上面的步骤没有），这里把 4.35TB 的委派空间分配到 Eql A 上面的卷做副本，一般是副本卷空间量 2 倍的大小，如图 18-10 所示，设置卷保留空间。

keep failback snapshot 根据需要选择，这里选中这个选项的作用是，当 Eql A 宕机后，如果在另外一台 Eql B 上对这个卷做了 replication，可以让 Eql B 上的卷替换 Eql A 上面的卷，

1107114476@qq.com
dolphinebook.com

等 Eql A 设备正常后，再把有数据变化的 Eql B 上面的卷和 Eql A 上面的卷同步，避免复制所有卷的内容，如图 18-11 所示，配置 Eql failback。

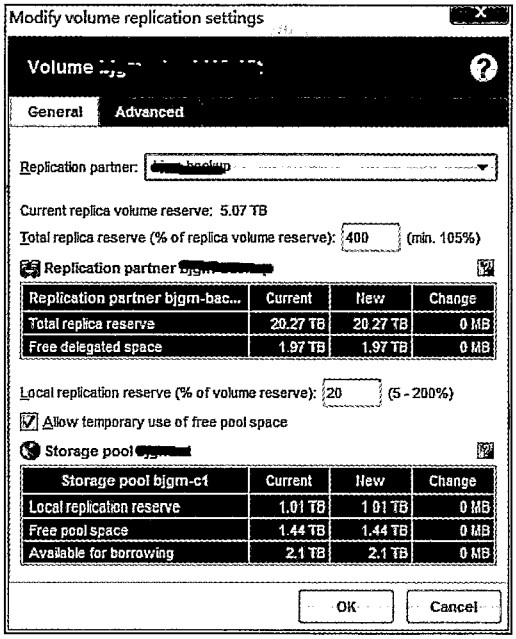


图 18-10 设置卷保留空间

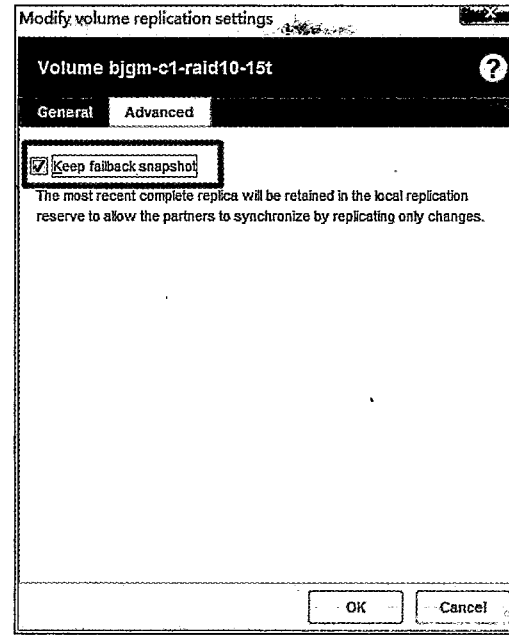


图 18-11 配置 Eql failback

单击“finish”按钮，完成设置。

在 Outbound 可以看到 enabled，表示已完成 replicate，如图 18-12 所示，如果是 In-progress，则正在 replicate 数据。

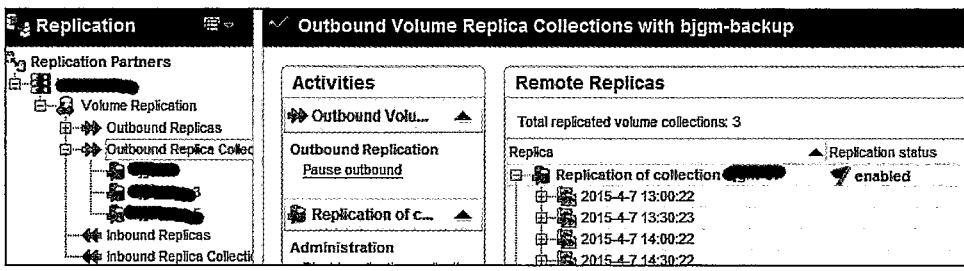


图 18-12 进行中的 Eql 异步复制

过段时间后，从 Eql B 的 Inbound 可以看到数据传输完毕，如图 18-13 所示为完成异步复制的示意图。

关于集群恢复，假设 Eql A 因为某种故障无法提供服务，这时，只需将虚拟机关闭，计算节点挂载 Eql B 上面的卷，开启虚拟机，即可恢复。

1107114476@qq.com
dolphinebook.com

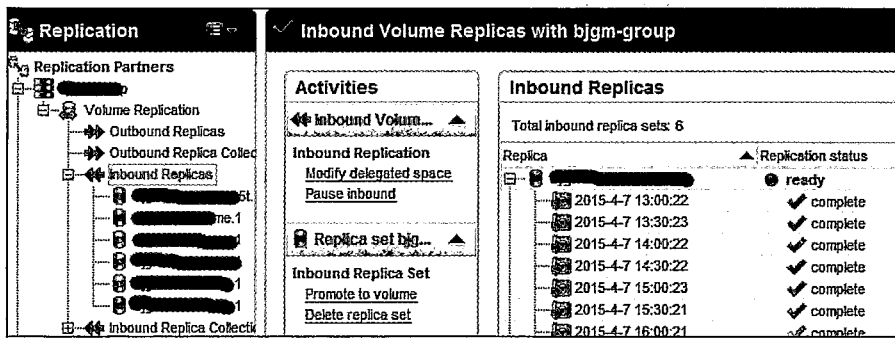


图 18-13 Eql 完成异步复制示意图

18.4 Dell EqualLogic 存储的管理实践经验

Dell EqualLogic 是一款性价比非常高的 ISCSI 存储，笔者在生产环境有大量使用，也积累了不少经验，本节分享下笔者这方面的经验，同时读者也可以对商业存储有个初步的认识。

1. Dell EqualLogic 简介

Dell EqualLogic 存储阵列定位就是 ISCSI 存储，其只有 ISCSI 的产品线，在 ISCSI 存储领域有很深的积累，管理智能自动化，并具有快速灵活的可扩展性。

EqualLogic 的一个特点就是可以横向扩展，通过网络将多台 EqualLogic 存储阵列组合到一起，形成一个大的组（Group），共同对外服务。组不仅易于管理，并且组中所有阵列可以互相配合工作，自动管理数据、跨所有资源平衡负载。扩展组后，其容量和性能会自动提升，且对用户无感知。新成员的存储空间添加到组的存储池中，可立即投入使用，如图 18-14 所示为只有一个成员的组。

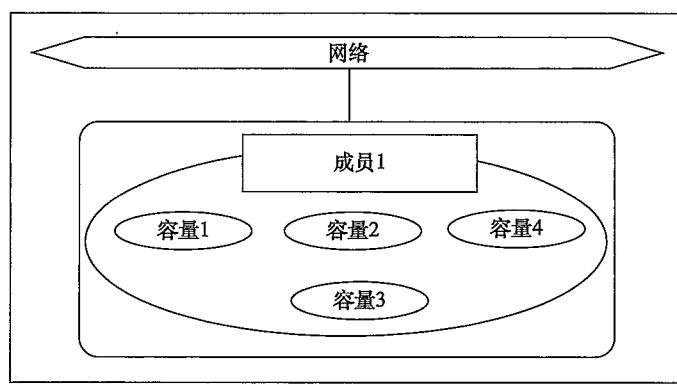


图 18-14 包含一个成员的组

要连接到卷，计算机只需要标准的 ISCSI 启动程序，通过 IP 地址、启动程序名称或

1107114476@qq.com
dolphinebook.com

CHAP（挑战握手验证协议）凭证限制对卷的访问。连接后，计算机会将卷视为常规磁盘，可以对其进行常规格式化。

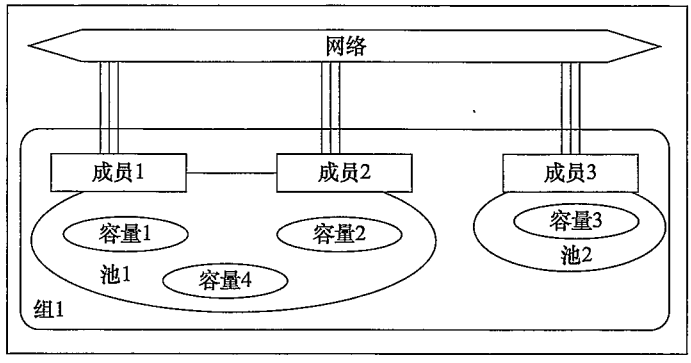


图 18-15　包含多个成员和存储池的扩展组

在多成员组中，成员之间会根据需要自动实现负载平衡。管理的额外开销始终相同。不管其大小如何，均可通过一个网络地址（称为组 IP 地址）将组作为单一系统进行管理。在多成员组中，可以将空间分为多个存储池，这样就可以根据用途来整理存储空间，从而更好地控制资源分配，并且只需要在单一系统管理视图中进行操作。成员和卷可以在各个存储池之间移动，而数据会始终处于联机状态，如图 18-15 所示为包含多个成员和存储池的组。

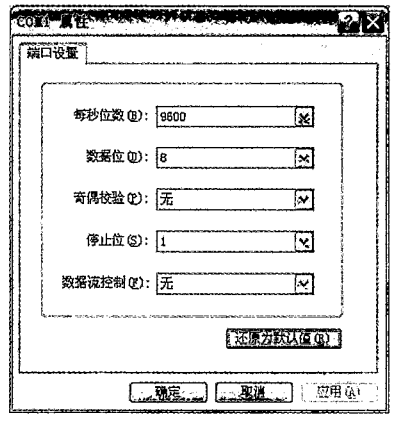


图 18-16　超级终端设置

2. Dell EqualLogic 初始化

在主机上创建超级终端，使用默认设置（9600，8，无，1，无），如图 18-16 所示为超级终端设置。

默认的初始化用户名和密码为 grpadmin，登录即可进入 Eql 的界面，如图 18-17 所示。

```
SP Loading ...Done

Dell, Inc. Storage Array

Copyright 2001-2008 Dell, Inc.

SP [secondary]:38.65:psarcp.c:429:INFO:28.2.5:Executing secondary control module
 in slot 1.
MFS set up
Building databases...
Fri Feb 13 06:36:43 EST 2009
Feb 13 06:36:43 init: kernel security level changed from 0 to 1

PS Series Storage Arrays
Unauthorized Access Prohibited

login: _
```

图 18-17　登录 Eql 界面

1107114476@qq.com
dolphinebook.com

配置 Group 初始化信息，从输入 setup 开始，如图 18-18 所示。

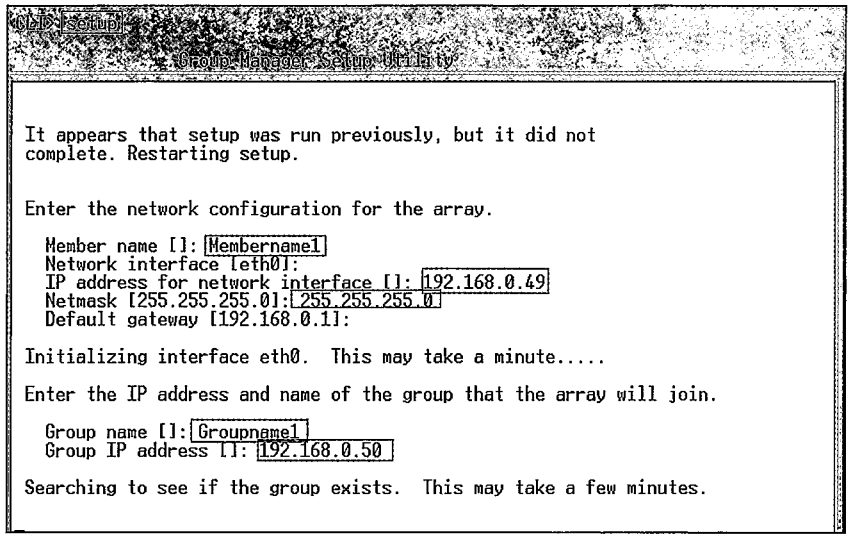


图 18-18 配置 Eql

在浏览器中输入 Group IP 即可登录 Web 管理页面，如图 18-19 所示。

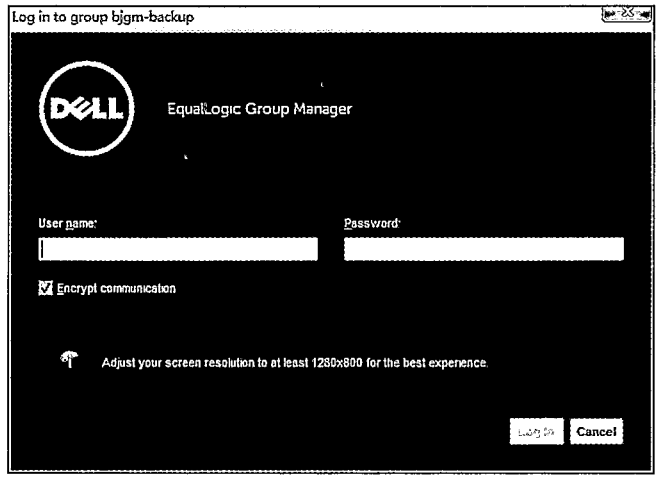


图 18-19 登录 Web 管理页面

3. Dell EqualLogic 常用功能介绍

EqualLogic 非常强悍的一个功能是可以横向扩展，多台 EqualLogic 组成一个大组，以一个 IP 对外提供服务，每个组中可以划分若干个存储池（Pool）。目前，一个组中最多支持 4 个存储池，在每个存储池中可以划分若干个卷，卷以 ISCSI 的方式挂载给计算节点使用。

1107114476@qq.com
dolphinebook.com

（1）异步复制

EqualLogic 的异步复制功能采用的是快照机制，可以做到最低 5 分钟的镜像同步，复制采用的是尽快复制的方式，第一次复制会占用大量带宽，以后复制占用的带宽取决于变化数据量。如果需要在生产环境中使用，建议放置在同一个机房内，并且保证足够的带宽。

EqualLogic 的快照功能是基于指针方式，理论上对性能没有损失，和 Linux 系统 lvm 快照采用的写时复制的快照机理不一样，所以，可以多份快照而不影响性能。

提示 关于快照的指针方式和写时复制（Copy On Write，COW）方式。

快照写时复制的方式是 Linux 系统 lvm 的快照方式，即有快照操作的时候，先指定一个块设备区间，如果是读操作，还是读取原来的数据块；如果是写操作，先把原块设备变动的地方复制到快照区间，再对原块设备对写操作。写时复制的方式，写操作的时候，对性能有影响，如果是多份快照，对性能影响更大。

指针方式是指写操作的时候，直接将变化的部分写到快照区间，然后在原块设备处通过指针标记，这样的快照没有复制过程，对性能影响要小很多。所以，使用指针的方式，要比传统的写时复制好很多，这种方案，目前笔者只在一些商业存储上看到应用。

（2）同步复制

EqualLogic 的同步复制是向两个处于不同存储池中的卷同时同步的写入数据，如果其中一个 pool 因为电源或其他原因不能访问，仍然可以从另外一个卷中获取数据。同步复制对磁盘压力比较大，网络带宽要求也高，同步复制采用的是牺牲性能来保存服务高可用的策略，在对数据可靠性、安全性要求非常高的场景可以使用同步复制功能。

（3）在线迁移卷

Eql 支持同一个 group 中，不同存储池之间在线迁移卷，一个卷可以在提供服务的同时，实现从一个存储池迁移到另外一个存储池。每个卷都与实际的存储相对应，卷迁移实际就是在不同的物理存储之间迁移数据，这个功能在实际应用场景中非常实用。

在线迁移卷，可以在存储维护期间实现业务零中断，如存储控制器固件升级，硬盘固件升级等场景，都可以将卷先迁移到同一个 group 中的其他存储池中，待升级完成后，再将卷迁移回来。

在线迁移卷，可以实现卷性能的动态调整。假定这样一个场景，当一个业务刚上线的时候，压力非常大，此时就需要将这个卷放到高性能的存储上，如 EQ 带纯 SSD 的型号，但当业务稳定后，压力下降，最终趋于平稳，此时，可以将卷迁移到低配置的存储上，如 EQ 带大容量 SATA 盘的型号，这样就实现了根据业务压力大小动态的调整卷性能，实现按需分配，在成本和灵活性上表现都非常出色。

卷迁移对网络的可靠性要求比较高，因为一旦网络断开，可能造成数据无法恢复的情况，所以，要在具有冗余设计的网络中做卷迁移。

1107114476@qq.com
dolphinebook.com

4. Dell EqualLogic 集中监控

Dell EqualLogic 提供了一个集中监控工具 SAN Headquarters（简称 SAN HQ），通过 SAN HQ 可以监控到当前存储的运作状态，如图 18-20 所示。

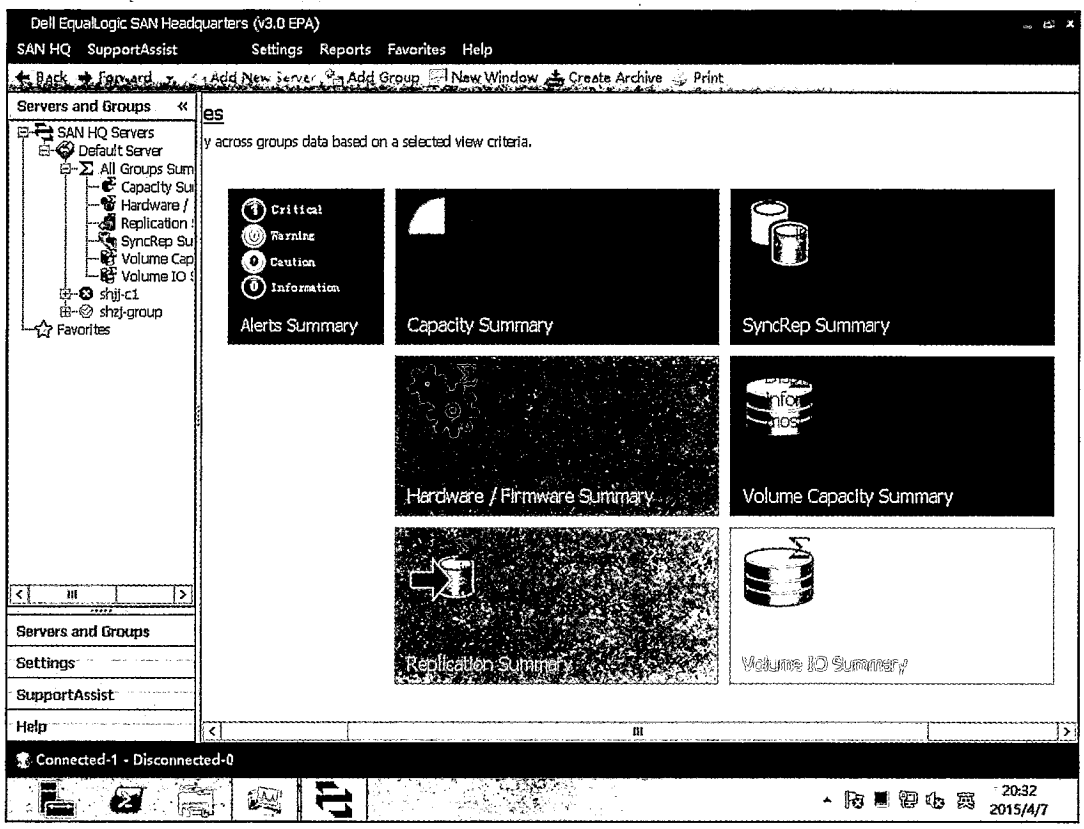


图 18-20 SAN HQ 监控 EQ

SAN HQ 的功能主要有以下几点：

- 存储容量信息，包括 group 容量使用情况、卷容量使用情况、快照容量使用情况、保留空间使用情况、本地复制空间使用情况。
- 同步复制汇总信息，展示的时候，两台 EQ 之间配置同步复制后的一些详细状态信息。
- 硬件及固件相关信息，主要包括控制器固件版本、存储型号、控制器类型、成员状态、服务编号、raid 状态、raid 级别、硬盘信息等。
- 卷详细信息，主要详细地罗列每个 group 中卷的详细信息，包括卷大小、使用空间、可用空间等。
- 卷异步复制详情，主要展示的是异步复制卷的一些状态信息。

1107114476@qq.com
dolphinebook.com

- 卷 I/O 性能信息，展示了每个 group 中各个卷的 I/O 情况，包括平均 I/O 吞吐、平均 I/O 延迟、平均 I/OPS 压力、平均 I/O 大小等。

通过 SAN HQ 可以查看存储的性能历史数据，从中判断业务的负载情况，及时的调整存储资源分配策略。

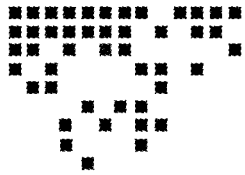
在 SAN HQ 还可以定义报警邮件，设置某些阀值，及时地给管理员邮件提醒。如果购买了高级服务，报警邮件可以发给 DELL 的 800 工程师的值班中心，值班中心的工程师会及时跟使用方工程师电话确认，确认后会立即下单维修。

18.5 本章小结

本章介绍了笔者虚拟化集群的运维实践经验，同单机虚拟化方式相比，集群虚拟化支持虚拟机在线迁移和高可用，业务可用性更高，数据安全更有保障，同时维护管理更复杂，在事件的运维工作中，需要做好流程和规范，以保证虚拟化集群稳定高效的运行。

下一章节将介绍如果把现有业务迁移到虚拟化环境。

1107114476@qq.com
dolphinebook.com



Chapter 19 第 19 章

业务迁移到虚拟化环境流程

将业务迁移到虚拟化环境的过程就像登山，所有的准备工作都是为了能够攀登到顶峰，在迁移前，要做足充分的准备和测试，迁移整个过程一定要保证业务的稳定性，慎之又慎，要在确保业务稳定的前提下完成迁移，就像在登山过程中最重要的是保护好个人的生命安全。

一般迁移的时候遵循如图 19-1 所示的几个步骤，能较好地保证业务迁移稳定性，保证迁移的成功率。

业务性能需求评估是对现有的业务进行评估，得出现有业务的压力模型。虚拟化方案设计是确定宿主机选型、配置和虚拟化比例。测试环境测试是验证业务在虚拟机上逻辑和性能方面没有问题。上线前制定计划和 checklist、小规模部署、全面部署，是为了保证尽量在保证业务稳定的前提下完成切换。下面具体介绍每个阶段如何实施。

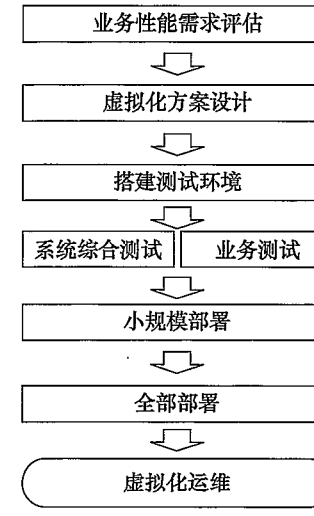


图 19-1　业务迁移到虚拟化的流程

19.1　性能评估与测试环境测试

性能评估可以通过脚本收集业务的性能压力数据，也可以通过监控系统查看长期的数据，具体的数据收集方法，已经在第 14 章做了介绍，数据收集分析之后，应形成测试报告。

1107114476@qq.com
dolphinebook.com

1. 性能评估

下面是一个 ×× 项目性能评估报告的例子，供读者参考。

第一部分：项目情况

×× 项目架构是数据库使用 MySQL 做主从，使用两台高配服务器，其他所有的角色都是业务服务器。现在使用 20 台服务器，功能分布表如表 19-1 所示。

表 19-1　×× 项目服务器功能分布表

用　途	数量	用　途	数量
业务服务器	15	业务服务器备机	2
DB	1	DB 备机	1
Slave DB	1		

业务架构图（略）。

第二部分：性能分析

在压力最大的区组抓取了高峰时期的物理机性能数据，并且抓取了监控系统一个月相应的性能图，相互对比，分析结果如下。

（1）业务服务器性能数据

如图 19-2 所示的是 CPU 压力最大时的一组数据。

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 6  0 853288 402348 178424 52992868    0    0     5    28    0    0  7  1 92  0  0
 6  0 853288 402412 178424 52992896    0    0     0    26 102298 174533 11  3 86  0  0
 1  0 853288 402880 178424 52992908    0    0     0   244 105192 181761 11  3 87  0  0
 7  0 853288 401708 178424 52992912    0    0     0   758 102838 175409 11  3 86  0  0
 8  0 853288 401788 178424 52992920    0    0     0   240 96123 171132 11  2 87  0  0
 4  0 853288 400868 178424 52992960    0    0     0  1556 101826 160840 13  2 85  0  0
 8  0 853288 397440 178424 52992968    0    0     0   132 88252 145873 12  3 85  0  0
 7  0 853288 397268 178424 52992984    0    0     2   194 92289 152471 12  4 84  0  0
 8  0 853288 396384 178424 52993096    0    0     0  1386 90228 146561 12  4 84  0  0
 6  0 853288 397548 178424 52993180    0    0     0   430 103789 157650 12  3 85  0  0
 8  0 853288 398324 178424 52993196    0    0     0   190 98299 155926 11  3 86  0  0
```

图 19-2　业务服务器 CPU 高峰压力数据

如图 19-3 所示的是一个月内 CPU 使用率。

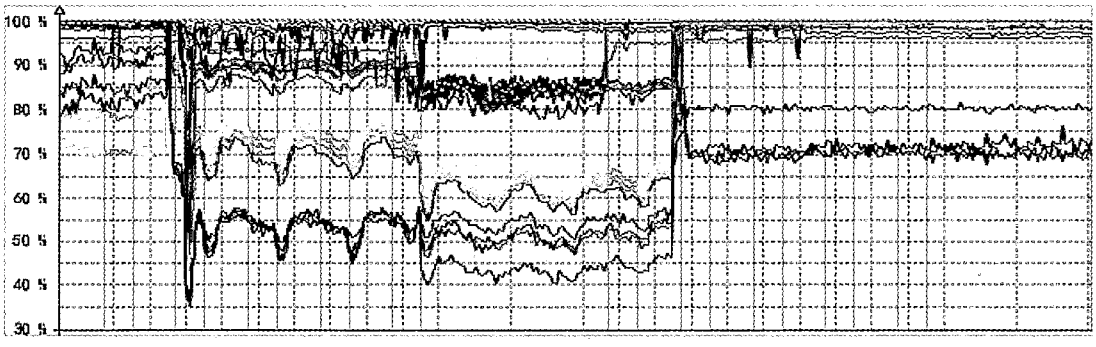


图 19-3　业务服务器 CPU 月压力数据

1107114476@qq.com
dolphinebook.com

如图 19-4 所示的是磁盘压力最大时的一组数据。

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           6.79    0.00    1.44    0.05    0.00   91.72

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.07   114.07    1.70   94.51   145.84   890.90    21.55     0.10    1.03   0.11   1.06
dm-0              0.00     0.00    1.69  208.48   145.54   890.49     9.86     1.24    5.90   0.05   1.05
dm-1              0.00     0.00    0.07    0.10     0.29     0.41     8.00     0.00    5.84   0.44   0.01

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          12.13    0.00    2.41    0.06    0.00   85.39

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.00    21.50    0.00  341.50     0.00  1452.00     8.50     0.20    0.57   0.05   1.55
dm-0              0.00     0.00    0.00  363.00     0.00  1452.00     8.00     0.27    0.74   0.04   1.60
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          11.38    0.00    2.50    0.00    0.00   86.13

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.00    28.50    0.00   41.00     0.00   278.00    13.56     0.01    0.12   0.02   0.10
dm-0              0.00     0.00    0.00   69.50     0.00   278.00     8.00     0.00    0.06   0.01   0.05
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
```

图 19-4 业务服务器磁盘压力峰值数据

如图 19-5 所示的是一个月内磁盘读 / 写吞吐。

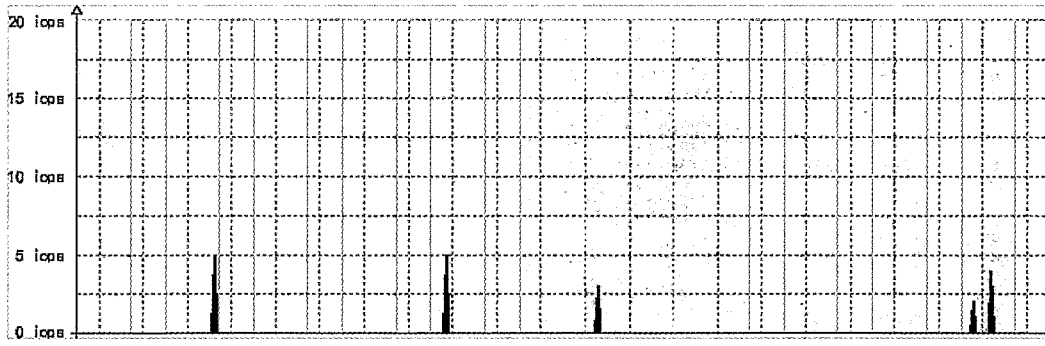


图 19-5 业务服务器磁盘月压力数据

如图 19-6 所示的是内存压力最大时的一组数据。

```
             total       used       free     shared    buffers     cached
Mem:     132119028  131717804     401224       2240     178424   52975144
-/+ buffers/cache:   78564236   53554792
Swap:     65535996     853804   64682192
```

图 19-6 业务服务器内存峰值数据

如图 19-7 所示的是一个月内内存使用情况展示。

1107114476@qq.com
dolphinebook.com

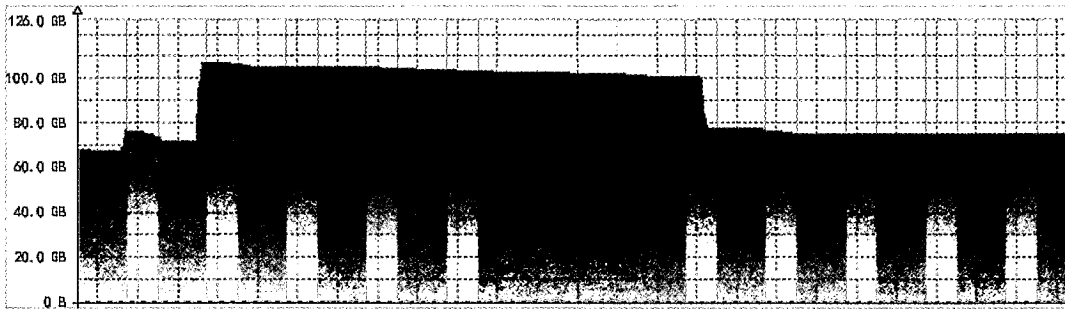


图 19-7　业务服务器内存月使用量数据

如图 19-8 所示的是网卡压力最大时的一组数据。

07:50:08 PM	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s
07:50:10 PM	lo	0.00	0.00	0.00	0.00	0.00	0.00	0.00
07:50:10 PM	eth0	4541.15	4757.81	293.72	2551.95	0.00	0.00	0.00
07:50:10 PM	eth1	2.60	0.00	0.15	0.00	0.00	0.00	0.00
07:50:10 PM	eth2	160.94	187.50	79.93	106.35	0.00	0.00	0.00
07:50:10 PM	eth3	3.65	0.00	0.27	0.00	0.00	0.00	0.00

图 19-8　业务服务器网卡峰值数据

如图 19-9、图 19-10 所示的是一个月内外网网卡吞吐和发包数据。

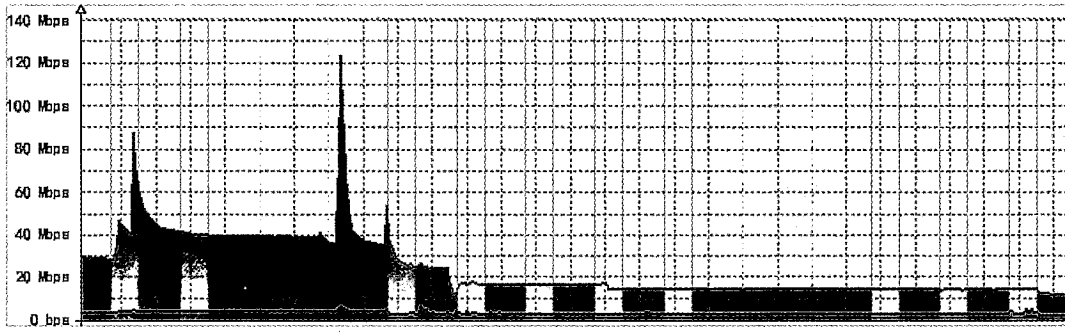


图 19-9　业务服务器外网网卡月流量数据

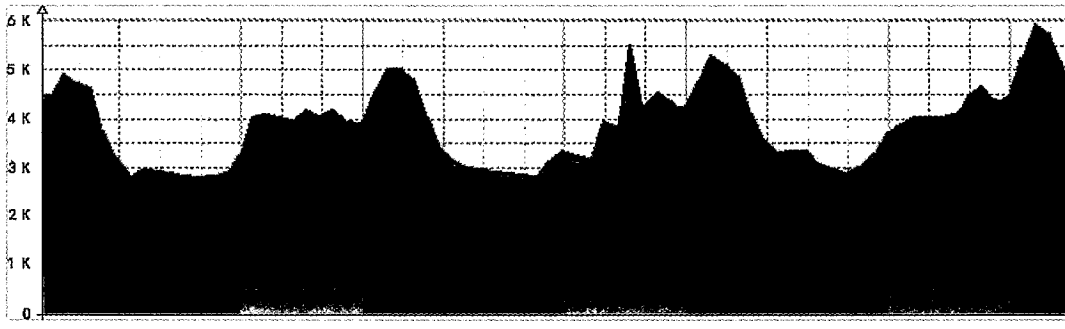


图 19-10　业务服务器外网网卡月发包数据

1107114476@qq.com
dolphinebook.com

如图 19-11、图 19-12 所示的是一个月内内网网卡吞吐和发包数据。

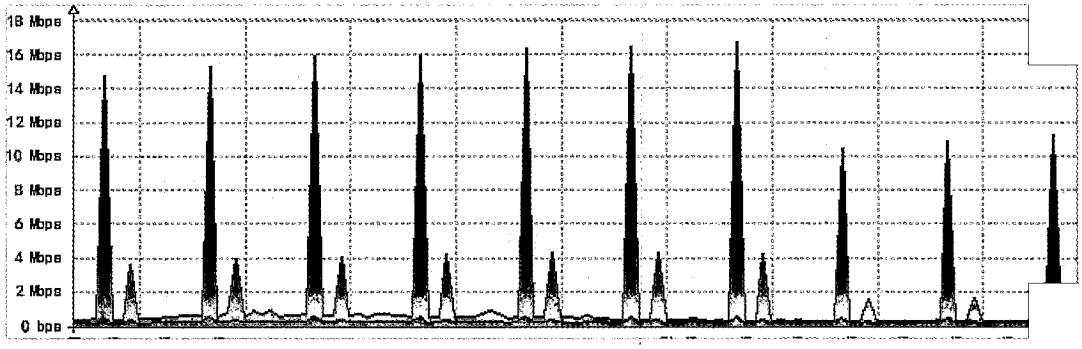


图 19-11 业务服务器内网网卡月流量数据

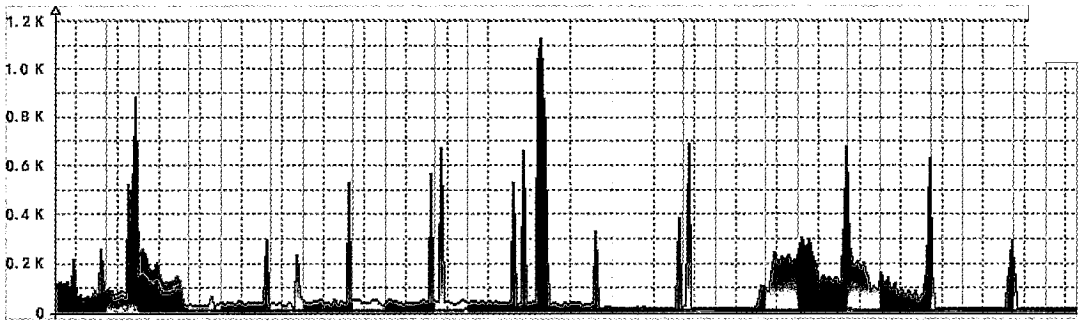


图 19-12 业务服务器内网网卡月发包数据

业务服务器性能分析结果如表 19-2 所示。

表 19-2 业务服务器性能分析结果

测试对象	业务服务器	业务环境数据	最大值参照
CPU	CPU 型号	志强 E5-2640 两颗	—
	CPU 利用率	41%	100%
内存	内存容量	12G	—
	内存使用量	63%	100%
磁盘	硬盘大小	300G	—
	写 I/O	80IOPS	2000
	读 I/O	170IOPS	10 000
	await time	12ms	20ms
网卡 1	发吞吐	50M	1000M
	收吞吐	8M	1000M
网卡 2	发吞吐	2M	1000M
	收吞吐	1M	1000M

1107114476@qq.com
dolphinebook.com

结论：从性能分析结果看，业务服务器 CPU 压力比较高，网卡流量相对较高，磁盘压力较低。

（2）DB 性能数据（略）。

DB 性能分析结果（略）。

（3）所有 ×× 项目服务器 CPU 和网络高峰使用情况统计

×× 项目服务器 CPU 和网络高峰使用情况统计样表如表 19-3 所示。

表 19-3 业务服务器性能分析结果

服务器角色	网络流量 /M	CPU 使用率 /%
业务服务器 1		
（略）		
DB 服务器 1		
（略）		

实际操作的时候，建议读者应收集每台服务器的流量和 CPU 使用率数据。

（4）虚拟化比例的确定

根据性能分析结果，根据压力大小，采用 2 ~ 6 台 GS 类服务器搭配一台 DB 服务器，按照平均 1 虚 5 的方式虚拟化是比较合适的方案。

虚拟化的时候，业务服务器每台分配 4 个 CPU、8G 内存、100GB 磁盘空间；DB 服务器每台分配 4 个 CPU、32GB 内存、500GB 磁盘空间。

2. 进行测试环境测试的必要性

虚拟化的实施是一个循序渐进的过程，大多数业务一般都有测试环境，先将测试环境切换到虚拟化平台，就是正式战争前的演练，演练得越充分，取得战争胜利的几率就越大，所以，进行测试环境的虚拟化测试是非常必要的。

无论对虚拟化技术掌握的多好，无论对业务是如何的熟悉，直接跳过测试环境，上正式环境都是不可取的。根据笔者的经验，每上一个虚拟化项目，都有许多新问题在里面，生产环境每一步操作都要小心翼翼，如果没有充分的测试，充足的准备，贸然操作，往往就是血的教训。

3. 如何进行测试环境业务逻辑测试

一版情况下，业务的测试环境就是为了进行业务的逻辑测试，所以，在虚拟化平台上，进行业务逻辑测试，对测试人员来说，完全是透明的，不需要做任何改变，业务测试人员也只需要关注业务相关的逻辑就可以了。

业务逻辑的测试一般分为两类，一类是人工测试，就是依靠人工检查；一类是自动化测试，业务程序上有任何的改变，只需要将全套的测试用例跑一遍就可以。两者往往需要结合使用。

总之，在虚拟机上的业务逻辑测试和在物理机上的测试完全一样，不需要特别关注。根

1107114476@qq.com
dolphinebook.com

据笔者的经验，往往是因为使用了虚拟机，通过快照和多份的镜像备份，使测试环境可以停留到任何一个点上，这样反而大大提高了测试效率。

4. 如何进行测试环境业务压力测试

部署虚拟化的时候，首先要熟悉业务的压力模型，通过性能评估脚本和监控工具，可以获得宿主机、虚拟机的压力情况。

虚拟机的性能，一般会比物理机低一些。虚拟化就是为了建立资源池，把物理机的资源进行切分。虚拟机能承担的业务压力，一般要比物理机低一些，所以，掌握虚拟机极限的业务承载能力非常重要。

首先，做宿主机和虚拟机系统方面的综合测试，如 CPU 性能、磁盘吞吐和 I/O、网络吞吐和 I/O 的测试，得到各方面的极限测试数据。测试的时候，测试宿主机，建立一个基准，然后，在虚拟一台虚拟机的情况下，虚拟各方面的极限性能，测试一下多台虚拟机同时的系统性能压力。比如，宿主机 4k 的随机写是 3000IOPS（Input/Output Operations Per Second），单台虚拟机可能是 1500 的 IOPS，4 台虚拟机同时打磁盘压力的时候，可能就是 2800 的 IOPS。总之，需要尽量模拟生产环境，将虚拟机系统方面的极限压力测出来，取得数据，为后面的生产环境实施做好数据依据，打好基础。

系统测试完成之后，就是业务方面的压力测试，每种业务一般都有专用的测试工具，如 Web 类、数据库都是非常成熟的应用，有许多的压测工具可以选择，游戏业务一般都有机器人。也有许多业务没有压力测试工具，这个就需要研发协商，看看能不能帮助写一个工具，或者提供一些思路。总之，尽量要取得业务在虚拟机上的最大压力数据，方便业务虚拟化的时候按照数据进行规划部署。

业务测试完成后，应形成业务测试报告，下面是一个 ×× 项目虚拟化测试环境测试报告的例子，供读者参考。

第一部分：结论

压力测试方面，经过实际测试，虚拟机最大负载在业务方面可以达到以下目标。

这里需要根据自己的业务情况，附上相关的数据，如果是游戏业务，可以附上游戏人数等数据，如果是 Web 类业务，可以附上 PV 等相关数据。

第二部分：测试环境及测试方法

1）计算节点：提供计算、内存、存储。表 19-4 为计算节点的一些信息。

表 19-4 计算节点信息

服务器型号	配　置
CPU	2 颗 Intel(R) Xeon(R) X5650 @ 2.67GHz
内存	24GB
硬盘	SAS 146*2（Raid 1）
网卡	Broadcom NetXtreme II BCM5709
服务器数量	1

1107114476@qq.com
dolphinebook.com

2）宿主机系统：RedHat RHEL 6.5，内核版本 2.6.32-431.17.1.el6.x86_64。

3）虚拟机配置如下：

- CPU：10 个。
- 内存：22GB。
- 磁盘空间：150GB。
- 系统版本：CentOS release 5.3(Final)。
- 内核版本：2.6.18-371.9.1.el5PAE。

4）测试方法：业务压力测试，通过业务压力模拟，收集虚拟机、宿主机性能压力，以及业务数据。

第三部分：测试结果

如图 19-13 所示的是宿主机 CPU 利用率。

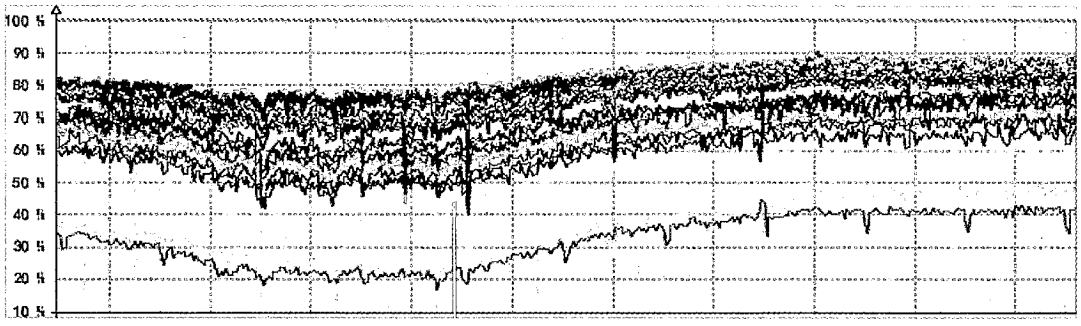


图 19-13　宿主机 CPU 利用率

宿主机有 32 个核最高利用率达到 20% 以上。

如图 19-14 所示的是宿主机内存压力。

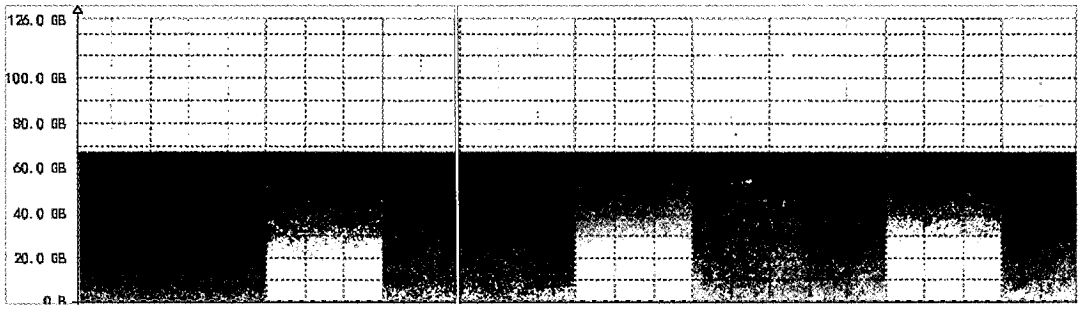


图 19-14　宿主机内存压力

如图 19-15 所示的是宿主机网卡吞吐。

如图 19-16 所示的是宿主机网卡发包。

1107114476@qq.com
dolphinebook.com

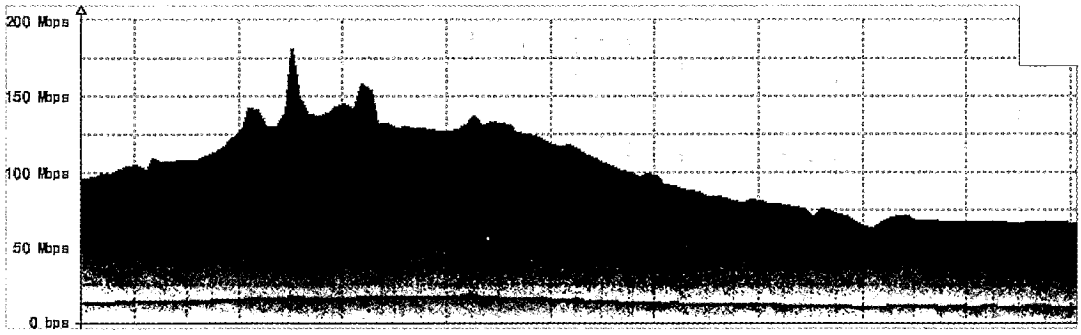


图 19-15 宿主机网络吞吐

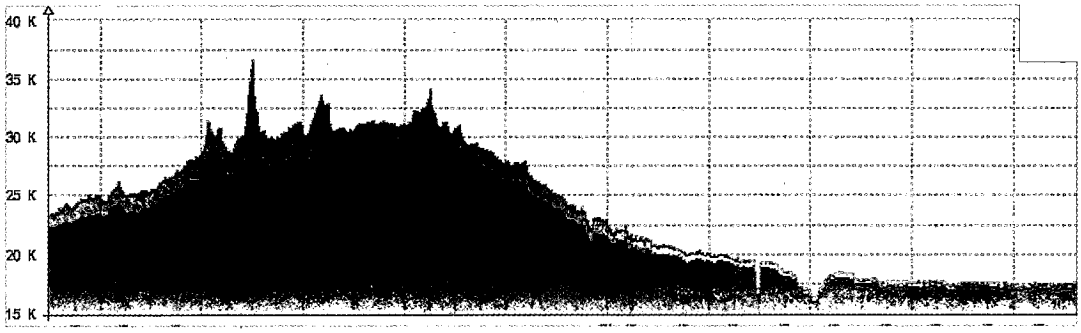


图 19-16 宿主机网络发包

第四部分：虚拟化收益预估

目前 ×× 机柜和服务器使用情况：×× 目前在线服务器 20 台，占用机柜 2 个。

虚拟化费用方面收益预估及虚拟化方案选择：如果按照平均 1 虚 5 的比例进行虚拟化，可以将机柜压缩到 1 个，服务器压缩到 5 台宿主机，其中 4 台跑生产，一台备用。

第五部分：宿主机选型

考虑总体服务器数量比较少，使用单机虚拟化方式，宿主机配置如下：

- 2U 机架式。
- 2 颗 Xeon 8C E5-2640 V2 2.0GHz。
- 16 × 8GB。
- 8 块 300GB 10K SAS 2.5 寸，RAID1G 缓存带电池。
- 无光驱，双电源，远程管理卡。

19.2 上线前的检查工作

业务正式上线之前，应该根据业务的具体情况，制定 checklist 检查，并对正式环境进行业务的逻辑和压力测试，尤其是第一台虚拟机的上线，尽量做好各种测试，努力保证上线成功。

1107114476@qq.com
dolphinebook.com

如表 19-5 所示的是一个虚拟机上线的 checklist，供读者参考。

表 19-5　某项目虚拟机上线 checklist 参考

工作流程	工作项目	完成时间	完成状态	责任人	备　注
网络准备	交换机配置检查	2015/6/30	100%	张三	
	宿主机、虚拟机 MAC、IP 对应关系网络设备确认	2015/6/30	100%	张三	
	网络链路、设备故障预案演练	2015/6/30	100%	张三	
系统准备	宿主机硬件报警检查	2015/6/30	100%	李四	
	宿主机网口状态检查	2015/6/30	100%	李四	
	宿主机系统版本检查	2015/6/30	100%	李四	
	宿主机 libvirt 配置检查	2015/6/30	100%	李四	
	虚拟机镜像检查	2015/6/30	100%	李四	
	虚拟机 autostart 配置检查	2015/6/30	100%	李四	
	虚拟机网桥检查	2015/6/30	100%	王五	
业务检查	业务逻辑验证	2015/6/30	100%	王五	
	业务压力模拟测试	2015/6/30	100%	王五	
	业务故障模拟	2015/6/30	100%	王五	
联合调试	虚拟机故障演练	2015/6/30	100%	王五、李四	
	宿主机故障演练	2015/6/30	100%	王五、李四	

19.3　小规模部署及逐步切换到虚拟化环境

业务正式迁移到虚拟化环境，需要逐步谨慎推荐，应先进行较长时间的生产环境小规模部署，一般建议小规模生产环境部署为两周到一个月比较合适。

1. 小规模部署

一般第一台正式上线后的虚拟机稳定跑上一段时间后，就可以逐步将业务进行迁移，迁移的时候要时刻观察待迁移业务机器的压力，跟业务做好后续计划的沟通。如果有特殊的活动，要对压力做好预估，防止因为压力过大，造成虚拟机反应慢，甚至卡死，给业务造成损失，从而使业务丧失虚拟化信息。

2. 逐步切换到虚拟化环境

随着虚拟化的稳定逐步进行，可以加快虚拟化进度，但是每次虚拟化的迁移都不能掉以轻心，应该还是遵循虚拟机上线、检查、测试、再迁移业务的步骤。并且从第一台虚拟机上线的时候开始，就要做好虚拟化的监控和灾备工作。

还有一种情况，就是在这个过程中出现问题。出现问题也正常，如果问题是在预案中

1107114476@qq.com
dolphinebook.com

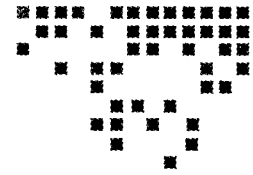
的，按照预案进行处理；如果是没有碰到的问题，根据笔者的经验，往往都是业务层面配置的问题居多，应该跟业务充分沟通，搭建测试服务器，看看能不能重现问题，再仔细地查找问题。也可以在物理机上进行对比测试，往往在虚拟机上碰到的问题，在物理机也会出现。

19.4 本章小结

本章介绍了将业务迁移到虚拟化环境的流程及注意事项，再次总结，在业务迁移的过程中应坚持稳扎稳打，步步为营的策略，确保业务稳定是第一位的。

业务迁移到虚拟化环境之后，面临的一个问题就是如何提高运维效率，下一章将为读者介绍宿主机自动化运维管理。

1107114476@qq.com
dolphinebook.com



第 20 章 Chapter 20

宿主机自动化运维管理

第 19 章介绍了将业务迁移到虚拟化环境的整个流程，迁移完成之后面临的另一个问题就是宿主机的维护管理。随着业务的发展，宿主机数量在不断增加，对于宿主机的管理，需要尽可能地做到自动化，将一些批量的、固定的、重复的事情交由自动化工具来实现，这不仅能提高效率和准确性，还能让运维人员从枯燥的工作中脱身出来，把精力放到研究及解决疑难问题上面来。

本章将以通过 Puppet 管理宿主机为主线，介绍如何实现宿主机自动化运维管理、如何设计自动化运维架构、如何定制自动化运维任务，以实际的案例，展示自动化运维的魅力。

20.1 Puppet 简介与安装部署

提到自动化运维，Puppet 无疑是目前最火、最成熟的自动化运维工具之一，笔者在生产环境中大量使用 Puppet 来管理宿主机。Puppet 有非常丰富的模块，类似一些面向对象的编程语言，Puppet 有自带的类，通过简单调用就能实现非常复杂的功能。在 Puppet 官网（https://forge.puppetlabs.com/）已经提供了几千个模块。

本章对 Puppet 的介绍，侧重于宿主机管理的应用层面，对于一些基本的 Puppet 语法、安装将不做详细介绍，读者可以直接通过 Puppet 官方提供的文档进行学习，链接地址如下：

http://docs.puppetlabs.com/puppet/latest/reference/lang_summary.html。

通过官网文档可以获取非常详细的信息，机械工业出版社的《 Puppet 实战》[⊖]也是非常

⊖ 本书由机械工业出版社华章公司出版，书号 978-7-111-44518-0。

1107114476@qq.com
dolphinebook.com

不错的一本书，调理清楚、逻辑严谨、实践经验丰富，推荐读者阅读参考。

1. Puppet 简介

Puppet 的出现，将自动化运维推到了一个全新的层面，目前和 Puppet 类似的工具有很多，如常用的 Chef、Ansiable、ControlTier 等，这些工具有一些共性，都能实现标准化和自动化的配置管理，综合比较，Puppet 配置灵活简便、稳定性好、成熟度高，目前它在生产环境得到广泛使用。

其实，在 Puppet 这类自动化运维工具出现之前，作为运维人员，都会通过写一些脚本或小程序来实现一些自动化操作，如软件安装脚本、数据库备份脚本、软件配置脚本等。但是这些脚本有一个最突出的问题——兼容性非常差，如在 CentOS 上写的脚本拿到 Ubuntu 系统上去运行，一般就会报错，除非在脚本中做了很多判断来兼容它，但是这样做是很麻烦的，基本上就是将这个脚本重写一遍。

笔者有这样的体会，如果在网上查找数据库备份脚本，会找到很多，但是能拿来能直接用的基本上没有。所以，通常的结果就是需要批量做一些事情，临时写个脚本，其他地方再用了，再重写，这样就非常低效。

Puppet 最突出或者说最成功的地方，就是它对要做的事情做了一层抽象，把一件完整的事情抽象成多个步骤，在每个步骤中提供了很多实现这个基本功能的操作方法，在 Puppet 中将这种方法称为资源，最后将这些过程连起来，即完成了一个复杂的任务。在 Puppet 3.6 中，默认提供了 52 种不同类型的资源，但是其中常用的不超过 20 种，所以，学习起来还是比较容易的。

2. 使用 Apache 及 Passenger 替换自带的 WEBrick

Puppet 属于 C/S 架构，分为服务器和客户端两部分，Puppet 服务端通常称为 Puppet Master，Puppet 客户端称为 Node。Puppet 通过内置 Ruby 的 WEBrick 库来提供完整 Puppet Master 服务。WEBrick 库是一个 HTTP 服务的工具集，用它可以配置 HTTPS 服务、代理服务和虚拟主机服务。使用内置的 WEBrick 支撑的 Puppet Master 服务，官方并不建议在生产环境中使用，其中最主要的一个原因是性能，一般小于 10 个客户端的场景或者开发环境中可以直接使用 WEBrick 支撑 Puppet Master。本节将主要介绍如何使用 Apache 及 Passenger 替换自带的 WEBrick，以支撑大量客户端。

3. 替换 WEBrick 的必要性

在实际的应用场景中，往往会有成百甚至上千的服务器需要用 Puppet 来集中管理，按照 Puppet 官方说明中提到的，WEBrick 仅在 10 个客户端以下或者开发环境中适用，实际的生产环境中就必须替换它。

Apache 及 Passenger 其实就是利用 Apache 的 Passenger 模块实现基于 Rails 框架的应用部署，简单地说就是在 Apache 上运行 Ruby 的应用。这个架构目前已经非常成熟了，在生产环境中已经被普遍应用了。笔者在实际的生产环境中，使用 Apache 及 Passenger 管理了超过

1107114476@qq.com
dolphinebook.com

500 台服务器，已经平稳运行了两年以上。所以，不管是从性能，还是从稳定性考虑，替换 WEBrick 是非常有必要的。

4. 操作步骤及注意事项

对于基本的 Puppet 安装，本书就不做详细介绍了，笔者的系统是 CentOS 6.5，基本的安装步骤如下：

- 添加 Puppet 官方的 yum 源。
- 使用 yum install puppet-server 命令，安装 Puppet Master Server。
- 更改主机名，并将主机名绑定在 hosts 文件中，或者配置统一的 DNS。
- 使用 service puppetmaster start 启动 Puppet Master 服务。

经过上面的步骤，一个最简单的 Puppet Master Server 就部署完成了。接下来开始介绍如何使用 Apache 及 Passenger 替换自带的 WEBrick。

（1）安装 Apache 及一些依赖的软件包

使用如下命令安装 Apache 及 ruby。

```
[root@puppetmaster ~]# yum install -y httpd httpd-devel
[root@puppetmaster ~]# yum install -y mod_ssl ruby-devel rubygems gcc
```

安装 Rack 和 Passenger。

```
[root@puppetmaster ~]#gem install rack
[root@puppetmaster ~]#gem install passenger -v 3.0.19
```

安装 Passenger 时，需要指定版本号，笔者在 CentOS 6.5 系统，Apache 版本是 2.2.15 的环境下，默认安装 Passenger 版本为 5.0.4，使用该版本，Passenger 启动 httpd 服务器时候提示：

“Invalid command 'RackAutoDetect', perhaps misspelled or defined by a module not included in the server configuration”

所以，建议安装时指定版本为 3.0.19，该版本是笔者线上环境中运行的稳定版。

（2）安装 Apache Passenger 模块

Passenger 运行之后，会检查依赖性，如果某些包没有安装，会提示如何安装，按提示操作即可。

```
[root@puppetmaster ~]# passenger-install-apache2-module
```

安装成功之后，会出现如下提示：

```
The Apache 2 module was successfully installed.
Please edit your Apache configuration file, and add these lines:
LoadModulepassenger_module \
```

1107114476@qq.com
dolphinebook.com

```
/usr/lib/ruby/gems/1.8/gems/passenger-3.0.19/ext/apache2/mod_passenger.so
PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-3.0.19
PassengerRuby /usr/bin/ruby
<!-- 这部分配置放在 httpd.conf 配置文件中，当重启 httpd 服务的时候，会载入 passenger 模块 --!>
<!-- 此处省略多行输出 --!>
<VirtualHost *:80>
ServerName www.yourhost.com
# !!! Be sure to point DocumentRoot to 'public'!
DocumentRoot /somewhere/public
<Directory /somewhere/public>
# This relaxes Apache security settings.
AllowOverride all
# MultiViews must be turned off.
Options -MultiViews
</Directory>
</VirtualHost>
<!-- 以上这段是虚拟机主机的配置，可以在 Apache 的配置文件目录，创建虚拟机主机配置文件，
     并将这部分内容放在这个文件中 --!>
```

（3）配置 Apache 使其能运行 Puppet Master 应用

首先，需要创建一个目录，并从 Puppet 的源码中复制配置文件 config.ru 到该目录，更改相应的权限。

```
[root@puppetmaster ~]#mkdir -p /usr/share/puppet/rack/puppetmasterd
[root@puppetmaster ~]#mkdir /usr/share/puppet/rack/puppetmasterd/public \
 /usr/share/puppet/rack/puppetmasterd/tmp
[root@puppetmaster ~]#cp /usr/share/puppet/ext/rack/config.ru \
 /usr/share/puppet/rack/puppetmasterd/
[root@puppetmaster ~]#chownpuppet:puppet \
 /usr/share/puppet/rack/puppetmasterd/config.ru
```

其次，需要创建一个虚拟主机服务搭载 Puppet Master 应用，可以通过自带的模板文件创建，并将安装完 Apache Passenger 模块时提示的内容放到虚拟主机配置文件中。

```
[root@puppetmaster~]#cp /usr/share/puppet/ext/rack/ \
example-passenger-vhost.conf /etc/httpd/conf.d/passenger-vhost.conf
[root@puppetmaster ~]# cd /etc/httpd/conf.d/
[root@puppetmasterconf.d]# vim passenger-vhost.conf
LoadModulepassenger_module /usr/lib/ruby/gems/1.8/gems/ \
passenger-3.0.19/ext/apache2/mod_passenger.so
PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-3.0.19
PassengerRuby /usr/bin/ruby
<!-- 上面的内容就是安装完 Apache Passenger 之后提示的载入 Passenger 模块的命令 --！ >
# you probably want to tune these settings
PassengerHighPerformance on
PassengerMaxPoolSize 12
PassengerPoolIdleTime 1500
# PassengerMaxRequests 1000
PassengerStatThrottleRate 120
```

1107114476@qq.com
dolphinebook.com

```
RackAutoDetect Off
RailsAutoDetect Off
Listen 0.0.0.0:8140
<!-- 配置虚拟主机监听的 ip 和端口 --!>
<VirtualHost 0.0.0.0:8140>
SSLEngine on
SSLProtocol -ALL +SSLv3 +TLSv1
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:-LOW:-SSLv2:-EXP

SSLCertificateFile      /var/lib/puppet/ssl/certs/puppetmaster.test.com.pem
SSLCertificateKeyFile   /var/lib/puppet/ssl/private_keys/puppetmaster.test.com.pem
SSLCertificateChainFile /var/lib/puppet/ssl/ca/ca_crt.pem
SSLCACertificateFile    /var/lib/puppet/ssl/ca/ca_crt.pem
# If Apache complains about invalid signatures on the CRL, you can try disabling
# CRL checking by commenting the next line, but this is not recommended.
SSLCARevocationFile     /var/lib/puppet/ssl/ca/ca_crl.pem
<!-- 以上配置的相关证书路径需要根据实际情况进行修改，本实例都是默认安装的路径 --!>
SSLVerifyClient optional
SSLVerifyDepth  1
# The 'ExportCertData' option is needed for agent certificate expiration warnings
SSLOptions +StdEnvVars +ExportCertData
# This header needs to be set if using a loadbalancer or proxy
RequestHeader unset X-Forwarded-For
RequestHeader set X-SSL-Subject %{SSL_CLIENT_S_DN}e
RequestHeader set X-Client-DN %{SSL_CLIENT_S_DN}e
RequestHeader set X-Client-Verify %{SSL_CLIENT_VERIFY}e
DocumentRoot /usr/share/puppet/rack/puppetmasterd/public/
RackBaseURI /
<Directory /usr/share/puppet/rack/puppetmasterd/>
Options None
AllowOverride None
Order allow,deny
allow from all
</Directory>
</VirtualHost>
<!-- 以上虚拟主机的配置，就是安装完 apache passenger 模块之后，提示的创建虚拟主机实例，
     需要改变的是根路径  --!>
```

Apache 配置完成后，启动 Apache 服务。

```
[root@puppetmaster ~]# service httpd start
Starting httpd:                                            [  OK  ]
[root@puppetmaster ~]# netstat -tunpl|grep http
tcp        0      0 0.0.0.0:8140        0.0.0.0:*        LISTEN      16410/httpd
```

至此，使用 Apache 和 Passenger 替换自带的 WEBrick 已经完成，在客户端测试连接，类似如下输出，说明配置已经正常了。

```
[root@puppetclient ~]# puppet agent --test
Info: Retrieving pluginfacts
```

1107114476@qq.com
dolphinebook.com

```
Info: Retrieving plugin
Info: Caching catalog for puppetclient.com
Info: Applying configuration version '1426590587'
Notice: Finished catalog run in 0.04 seconds
```

20.2 利用 Puppet 管理宿主机的运维架构设计

Puppet 任务架构这部分内容，是笔者一些实践经验的总结，通过一个合理的任务架构设计，能使得在后续的配置和管理过程中事半功倍，一个好的任务架构，需要具备以下特性：

- 灵活性。
- 可控性。
- 安全性。

（1）灵活性

对于一些基础的服务检查及通用配置管理，需要能应用于所有的 Puppet 客户端，而不需要重复的定义。由此可以将一些基础的服务，定义为模块，用于检查一些服务的状态，如必须开启 iptables 服务，关闭不必要服务等。

（2）可控性

任务可控性，这点非常重要，必须非常清楚新建或者变更一个任务的时候，有多少 Puppet 客户端受影响，针对全局任务的调整必须谨慎对待，通过分层的思想，将任务的影响面分层化，由小范围向大范围延伸。

（3）安全性

任务安全性，主要考虑以下两点：

1）Puppet 客户端配置文件是否可回退。通常的方法可以通过配置 svn 或者 git，将 Puppet 客户端任务配置文件放在 svn 或者 git 上，借助 svn 或者 git 的版本控制功能，当出现某个配置文件变更出错的时候，能及时回退。

2）不可逆任务的处理。针对一些不可逆任务，往往把 Puppet 客户端配置文件回退到某个版本，无法将这些变更复原，安装软件包就是一个明显的例子。对此，在执行某些重要更改的时候，需要实现考虑好回退方案，事先写好回退的脚本或者程序，以防止不可逆的情况发生。

1. Puppet 任务架构设计思路及注意点

Puppet Master 任务的载入是从 /etc/puppet/manifests/site.pp 文件开始的，在这个文件中，通常不定义具体的某个 Puppet 客户端的任务，而是定义一些环境变量，及 Puppet 客户端任务的调用。具体实例如下：

```
[root@puppetmaster ~]# cat /etc/puppet/manifests/site.pp
import "node/*.pp"
import "node/shanghai_CT/*.pp"
```

1107114476@qq.com
dolphinebook.com

```
import "node/beijing_UN/*.pp"
import "db/*.pp"
Exec {
    path=>\
['/bin','/usr/local/sbin','/usr/local/bin','/sbin','/usr/sbin','/usr/bin'],}
```

上面的 site.pp 中定义，引用 Puppet 客户端配置文件的路径，并定义了 exec 类型的任务执行时的环境变量，如此定义可以避免创建 exec 任务时重复定义命令路径。

在定义 Puppet 客户端任务时，首先可以按照地域做一个归类，在 Node 目录下创建多个子目录，如 shanghai_CT 表示上海电信机房，beijing_UN 表示北京联通机房。具体如何命名读者可以自行定义，如此定义的话，可以直接看到这个机房 Puppet 管理了多少个客户端，并且一个机房内相同角色的 Puppet 客户端，往往需要执行相同的配置变更和管理。当客户端数量很多的时候，不至于混乱。

2. 生产环境多机房宿主机运维架构设计

在实际的应用中，往往会面临需要管理的服务器分别在全国多个机房，而不同的机房又会带有一些地域的特征，比如，上海的电信机房，北京的联通机房。在推送 DNS 服务器地址的时候，最佳的情况就是做一些区分，推送离各自最近的 DNS 服务器地址。

由此，可以定义一个配置文件，如下路径 /etc/puppet/manifests/inherits-tasks.pp。

```
[root@puppetmaster manifests]# cat /etc/puppet/manifests/inherits-tasks.pp
node "shanghai_CT" {
    file{'/etc/resolve.conf':
        ensure  => file,
        owner   => root,
        group   => root,
        source  => 'puppet:///files/resolv_shanghai.conf'
    }
}
<!-- 定义上海电信机房的通用主机 shanghai_CT, Puppet client 可以通 inherits 引用 --!>
node "beijing_UN" {
    file{'/etc/resolve.conf':
        ensure  => file,
        owner   => root,
        group   => root,
        source  => 'puppet:///files/resolv_beijing.conf'
    }
<!-- 定义北京联通机房的通用主机 beijing_UN, Puppet client 可以通 inherits 引用 --!>
}
```

上面的配置文件中，定义的推送的 resolv.conf 文件路径是 puppet:///files/resolv_shanghai.conf，这个文件在 Puppet Master 服务器上，需要先配置 /etc/puppet/fileserver.conf，在其中定义一个挂载点，命名为 files，具体的内容如下：

```
[root@puppetmaster puppet]# cat/etc/puppet/fileserver.conf
```

1107114476@qq.com
dolphinebook.com

```
[files]
  path /etc/puppet/files
  allow *
```

定义完挂载点后，创建/etc/puppet/files目录，将需要推送的文件resolv_beijing.conf、resolv_shanghai.conf，放在这个目录下面。

```
[root@puppetmaster puppet]# cat /etc/puppet/files/resolv_shanghai.conf
nameserver      202.96.199.133
nameserver      114.114.114.114
<!-- 推送的 DNS 配置文件中，定义了两个 DNS，一个是上海地区的，还有一个是通用的 dns--!>
[root@puppetmaster puppet]# cat /etc/puppet/files/resolv_beijing.conf
nameserver      202.96.0.133
nameserver      114.114.114.114
<!-- 推送的 DNS 配置文件中，定义了两个 DNS，一个是北京联通的，还有一个是通用的 dns--!>
```

最后，需要在 Puppet Node 的任务配置文件中，引入刚才定义的机房通用主机任务。

```
[root@puppetmaster ~]# cat \
 /etc/puppet/manifests/nodes/shanghai_CT/puppetclient.pp
node "puppetclient" inherits shanghai_CT{
<!--Puppet 客户端中通过 inherits shanghai_CT 引入机房通用的 Node 任务 --!>
        }
}
```

配置完成后，所有配置文件的路径和名称如下：

```
[root@puppetmaster puppet]# pwd
[root@puppetmaster puppet]# tree /etc/puppet
/etc/puppet
├── auth.conf
├── environments                    //在 Puppet 中可以定义多种配置环境，如 development, test
                                    //环境等，默认情况下，都放在此目录中
│   └── example_env
│       ├── manifests
│       ├── modules
│       └── README.environment
├── files                           //新建的目录，用于存放下发的配置文件
│   ├── resolv_beijing.conf
│   └── resolv_shanghai.conf
├── fileserver.conf                 //定义下发文件挂载点的
├── manifests
│   ├── inherits-tasks.pp           //定义的引用节点任务
│   ├── nodes                       //新建的目录，用于存放所有 Puppet 客户端任务配置文件
│   │   ├── beijing_UN
│   │   └── shanghai_CT
│   │       └── puppetclient.pp
│   └── site.pp                     //Puppet 入口文件
├── modules                         //此目录，后续会大量使用，存放的时候 Puppet 模块的内容
└── puppet.conf                     //Puppet 配置文件
10 directories, 9 files
```

1107114476@qq.com
dolphinebook.com

3. 任务的分层推送设计思路

任务分层，首先需要了解，为什么任务需要做分层设计，分层的目的是为了将一个任务的影响范围限制在一定范围内，从小范围向大范围扩散，避免一次性全局推送，避免能造成的全部服务器受影响。本节通过一个实例，进一步介绍任务分层推送的设计思路。

（1）任务应用场景需求

假设有这样一个应用场景，在全国有 5 个 IDC 机房，每个机房里面有多个角色的服务器，如有 Web 类的、DB 类的、Memcached 类的，还有宿主机。现在有一个需求，就是对所有的 Web 类服务器做网站版本更新，所有 Memcached 类服务器需要更新一个配置文件，更新完之后载入新配置。其他角色服务器不更新。

（2）任务设计及实现思路

针对以上场景的需求，应该如何设计和实现效果最好呢？

从需求来看，需要做两个任务，分别针对 Web 类服务器和 Memcached 类服务器，先看 Web 类服务器，更新网站版本，也就是最常用的站点更新。主要是内容替换，在 Puppet 里面抽象为文件推送。还有 Memcached 类服务器，需要更新配置，然后重新载入配置，在 Puppet 里面抽象为两步，配置文件推送并触发服务重新载入。抽象之后，发现这两个任务本身很简单，但是，在具体的实施过程中，还有一些点是需要注意的，一般更新操作，需要避免一次性全部推送，因为有时候开发的新版本，未必都是完美的，有些问题在测试过程中可能没有发现，真正上线后才发现，如果一次性推送，可能会造成服务受影响。所以，需要用任务分层的思路来逐步的推送更新。

具体的设计思路是这样的，针对 Web 类服务器，要能够先推送一个机房的，在完成一个机房的任务推送后，监控业务运行是否一切正常，如果正常，则推送一半的机房，本例中可以是 3 个，待推送完成后，继续监控业务是否一切正常，依然正常的情况下完成全部机房推送。其中任何一个环节出现问题，都需要停止继续推送，找到问题，必要情况下，进行回滚操作。Memcached 类服务器推送思路也一样。

上面的这个场景，在实际应用中，是比较常见的，下面将详细介绍，用 Puppet 实现的具体思路和配置步骤。

1）定义模块。对于这类常用的任务，可以定义一个模块，名字只需要符合 Puppet 的命名规则即可（模块命名必须以小写字母开始），本处定义一个名为 update-task 的模块，目录结构如下：

```
[root@puppetmaster ~]# tree /etc/puppet/modules/update_task/
/etc/puppet/modules/update_task/
<！ -- 如下展示了一个 puppet 模块最基本的结构 -- ！ >
├── files                    // files 目录里面存放需要推送的文件
├── manifests                // manifests 目录存放模块具体的任务内容，*.pp 文件
│   └── init.pp              // 模块的初始文件，每个模块都必须有一个 init.pp 文件
└── templates                // templates 目录存放模板文件，模板文件中包含以下变量，
                             // 通过给变量赋不同的值，可以得到不同的配置文件
3 directories, 1 file
```

1107114476@qq.com
dolphinebook.com

把常用任务定义为模块主要有以下两点好处：

- 结构非常清晰，通过直观的模块名称可以清楚它的作用。
- 通过模块方式定义的任务，可以非常灵活地被 Puppet 客户端调用。

2）模块内容介绍。在 init.pp 中定义了一个 update_task 类，它可以接收两个变量，分别是机房信息（location）和服务器角色（rules），内容如下：

```
[root@puppetmaster ~]# cat /etc/puppet/modules/update_task/manifests/init.pp
class update_task($location,$rules){
<!-- 定义 update_task 类，其中可以接受两个变量，location 和 rules --!>
    case $rules {
<！ --使用 case 来区分服务器角色变量 rules，使不同角色的服务器执行不同分支中的任务--！ >
        web:{
                case $location{
<!-- 在此通过 case 来区分服务器所属机房信息 location，不同机房可定义不同任务 --!>
shanghai_CT:{include  update_task::web_20150321}
beijing_UN:{}
<!-- 罗列了两个机房，在各自机房下定义的任务，旨在该机房有效，此处实现任务分层 --!>
                }
#               include update_task::web_20150321
<!—此处任务 update_task::web_20150321 被注释掉了，一点去掉注释，则有所机房的 Web 类型服务
 器都会执行 update_task::web_20150321 任务，这步是任务分层设计中，全局任务定义的位置 --!>
        }
memcached:{
<!—memcached 分支和 web 类定义方式一致，所以省略了 --!>
        }
        default:{}
}
}
class update_task::web_20150321{
<--！将 web 类服务器的更新任务，详细操作内容定义在这个类中，具体的读者朋友可以自行定义 !-->
        }
}
class update_task::memcached_20150321{
<--！将 memcached 类服务器的更新任务，定义在这个类中，具体操作读者朋友可以自行定义 !-->
        }
}
```

更新任务的模块已经定义完成了，下面需要说明在 Puppet 节点中如何使用。在父类节点中定义变量 idc，用于标注机房信息。

```
[root@puppetmaster ~]# cat /etc/puppet/manifests/inherits-tasks.pp
node "shanghai_CT" {
    $idc="shanghai_CT"
}
```

在 Puppet 节点中，定义变量服务器角色 rules，并调用模块 update_task 的类，为其赋值 location 和 rules，其中的 location 值便是父类节点中定义的变量 idc 的值。

1107114476@qq.com
dolphinebook.com

```
[root@puppetmaster ~]# cat /etc/puppet/manifests/nodes/shanghai_CT/puppetclient.pp
node "puppetclient" inherits shanghai_CT{
        class {'update_task':location => "$idc",rules => 'web'}
}
```

（3）小结

至此，完成了任务的分层推送设计，凡是定义在 update_task 模块中 web -> location 下机房分支中的任务，只在这个机房的 Web 角色服务器生效，定义在 update_task 模块中 Web 分支下的任务，将对所有 Web 角色服务器生效；定义在 update_task 模块中 default 分支下的任务，将对全部引用了 update_task 的 Puppet 节点生效。

4. 任务回退设计

任务回退作为错误最后的补救，通常情况下，如果按照上一节中任务分层推送的思路来操作的话，不会有大量 Puppet 节点出现这样的问题，但即使这样，任务回退也是非常必要的。

对于任务回退，可以分为两类，第一类是可逆的任务回退，比如，常见服务的配置文件推送，这种情况下如果配置出错，只需要将原来的配置文件重新推送，大多数情况下就完成了任务的回退操作。第二类是一些不可逆的，比如，一个某中类型服务器的配置脚本，其中做了很多的操作，包括修改配置、改环境变量、编译安装程序等，那么对于这类任务，就需要在测试环境中，对每一处的变更都需要考虑如何还原，通常比较简单的方式就是编写脚本。在实际的环境中，任务回退一定要考虑到，否则一旦出现问题，后果是非常严重的。

20.3　利用 Puppet 配置宿主机初始化操作

对于 KVM 宿主机，本质上也是一类应用服务器，所以，也需要做一些前期配置，主要是对一些基础服务做配置。本节介绍如何用 Puppet 完成部分宿主机初始化配置，最终要达到的效果是宿主机的快速部署及加入资源池。笔者亲历过使用 Puppet 在两天时间内，完成 6 个虚拟化集群的配置和 500 台云主机交付的任务。所以深切地体会到，自动化在宿主机运维中的重要性。

1. 宿主机初始化操作内容

宿主机初始化操作主要分为以下四个大类：

- 系统层面的调优配置。
- 系统服务的检查和配置。
- KVM 相关的服务和工具组件的安装和配置。
- KVM 管理平台相关的配置。

1107114476@qq.com
dolphinebook.com

2. 哪些初始化操作可以使用 Puppet

对于上面列举的四大类宿主机的初始化操作，其中，前三类可以很方便地做到 Puppet 自动化任务，第四类任务是否能用 Puppet 实现自动化，还需要看具体接入的 KVM 的平台是什么，因为目前这类平台很多，如 OpenStack、OpenNebula、CloudStack 等，还有一些有研发实力的公司会选择自己开发 KVM 管理平台，所以，对于第四类任务，需要根据实际情况来结合 Puppet，一般情况下能实现全部或者至少也能实现部分自动化操作。

3. 如何避免任务重复执行

Puppet 对于自带的一些默认类型资源，Puppet 内部通过检查，已经具备了跳过重复执行的操作，如文件推送、软件包安装、计划任务等。此处主要讲的是针对一些自定义的脚本，或者 exec 类型的任务，如何来避免重复执行。

通常 Puppet 在执行自定义脚本的时候，每运行一次 Puppet agent --test 就会执行一次，但是可以在编写脚本的时候，定义一个标记输出，如脚本执行成功生成一个文件，然后由 Puppet 来判断这个标记输出文件是否存在，存在则自动跳过。下面的实例，判断如果存在 /root/ossh 目录则自动跳过，不再重复解压。

```
exec {'unzip-file':
          command         => 'tar zxf /root/sshd_20050524.tar.gz',
          creates         => "/root/ossh",
          path            => ['/bin','/usr/local/sbin','/usr/local/bin',
                              '/sbin','/usr/sbin','/usr/bin'],
     }
```

还有一种是通过 unless 来判断是否执行，当脚本执行完成之后会监听 39422，如果检查到已经监听了这个端口，就不再重复执行安装。

```
    exec {'install-client':
cwd                       => '/root/ossh',
      command             => 'bash install.sh',
        path              => ['/bin','/usr/local/sbin','/usr/local/bin',
                              '/sbin','/usr/sbin','/usr/bin'],
      unless              => 'netstat -tunl|grep 39422 >/dev/null',
```

20.4 利用 Puppet 管理宿主机配置与更新

对于宿主机的配置，主要包含以下三个部分：

- 宿主机前期初始化过程中一个配置文件。
- 后期虚拟化平台的应用程序及相关配置。
- 一些用于日常维护的脚本，定时任务及配置文件。

通常对于第一部分的配置，初始化结束之后一般是很少有更新的，除非出现系统级别的漏洞或者 bug 才会去更新这部分配置。第二部分是经常需要更新的，因为一般来说，虚拟

1107114476@qq.com
dolphinebook.com

化平台都会定时的有版本更新，这种情况当然是内部自己开发的虚拟化平台更新会更频繁一些。第三部分也是会经常调整的，如一些监控或者备份策略，这部分内容运维人员的自定义层度最大。本节主要就介绍第二部分和第三部分的配置管理和更新处理策略。

1. 利用 Puppet 集中管理的宿主机配置

从目的出发，使用 Puppet 集中管理宿主机的配置，是为了宿主机出现故障时，能快速恢复。Puppet 集中管理的宿主机配置，主要分为以下两部分：

- 宿主机初始化过程中的配置，通过这部分配置的集中管理，能快速地实现宿主机的初始化。
- 宿主机所部署的其他程序、脚本或者平台，通过这部分配置的集中管理，能快速地恢复业务。

所以，使用 Puppet 兼顾这两部分的配置管理，在故障恢复及宿主机横向扩展非常有用。

2. 集中管理 iptables 脚本，使更新自动生效

对于一个 Puppet 任务，首先需要分解它，完成这个任务需要哪些步骤，每个步骤都做了哪些事情，对于防护墙自动配置更新任务，需要做的是管理 iptables 脚本，并且在 iptables 脚本更新的时候，能自动生效。分解 puppet 任务，包含以下两步操作：

- Puppet 文件推送。
- 使新的 iptables 生效。

具体如下：

```
file { 'iptables':
path                    => '/opt/script_firewall.sh',
ensure                  => file,
owner                   => root,
group                   => root,
source                  => "puppet://$fileserver/modules/postinstall/script_firewall.sh",
mode                    => 755,
} //iptables 脚本推送
exec {'flush_iptables':
command  => '/bin/bash /opt/script_firewall.sh -a',
subscribe=> File['iptables'],
refreshonly       => true,       //通过定义 refreshonly 参数，使 iptables 脚本更新的时候
                                 //自动执行 script_firewall.sh -a 刷新 iptables 策略
} //刷新 iptables 策略
```

当然，Puppet 对于 iptables 的管理，也有一些现成的模块可以使用，使用 puppet module search iptables 命令可以查找可用的模块，各列依次表示模块名称、描述信息、作者、关键字。需要使用模块的话，直接使用 puppet module install module-name 即可安装。默认安装在 /etc/puppet/module 目录下。

```
[root@puppetmaster manifests]# puppet module search iptables
```

1107114476@qq.com
dolphinebook.com

```
Notice: Searching https://forgeapi.puppetlabs.com ...
NAME    DESCRIPTION                AUTHOR      KEYWORDS
arusso-iptablesiptables management  @arussoiptables security firewall
erwbgy-iptables  manage iptables allow and deny rules  @erwbgyiptablesrhel firewall centos
<!-- 以下省略多行模块列表内容 --!>
```

通常情况下，Puppet 提供的模块都比较完善，考虑了各系统之间的兼容，但是内容较为复杂，使用前需要先读懂具体的任务内容及实现过程。读者可以通过阅读 Puppet 提供的模块内容，借鉴一些好的方法，来完善自己编写的模块。

3. 集群平台程序推送、安装及更新

这个实例是笔者生存环境中的一个例子，笔者的生产环境使用了一个被称为集群平台的系统来管理所有的虚拟机，平台全部由 Python 编写，任务的操作步骤如下：

- ❑ 推送新版的集群平台代码。
- ❑ 关闭正在运行的集群平台相关服务。
- ❑ 编译 Python 程序。
- ❑ 启动集群平台相关服务。

具体任务如下：

```
file {'/etc/init.d/compute-node':        //推送 compute-node 启动程序
        ensure          => present,
        source          => "puppet:///modules/vcloudv7/compute-node",
        mode            => 0755,
        owner           => root,
        group           => root,
  }                                      //推送 compute-node 的 init 服务脚本
file {'/root/zone-manager-package':      //推送最新的平台管理程序
ensure                  => directory,
        source          => "puppet:///modules/vcloudv7/zone-manager-package",
        mode            => 0644,
        owner           => root,
        group           => root,
        purge           => true,
        recurse         => true,
        force           => true,
        backup          => ".bak_$uptime_seconds",
}
service {'stop-compute-node':            //关闭正常运行的程序
ensure                  => "stopped",
        enable          => true,
        hasrestart      => true,
        hasstatus       => true,
        subscribe       => File['/root/zone-manager-package'],
refreshonly             => true,
exec{'python-build':
        path            => ["/usr/bin","/usr/sbin","/bin"],
```

1107114476@qq.com
dolphinebook.com

```
        command         => "python setup.py build",
        cwd             => "/root/zone-manager-package",
        user            => root,
        subscribe       =>Service ['stop-compute-node'],
        refreshonly     => true,
}
exec{'python-install':                  //编译安装最新的 python 程序
        path            => ["/usr/bin","/usr/sbin","/bin"],
        command         => "python setup.py install",
        cwd             => "/root/zone-manager-package",
        user            => root,
        subscribe       => Exec['python-build'],
        refreshonly     => true,
}
service {'start-compute-node':          //编译完成，开启程序
        ensure          => "running",
        enable          => true,
        hasrestart      => true,
        hasstatus       => true,
        subscribe       => Exec['python-install'],
}
```

上面的这个例子是一个完整的程序更新任务，读者可以根据自己的实际情况做一些变通，便可以完成，只要用最新的程序覆盖老程序，程序便可以自动更新。注意，在更新的过程中，需要充分测试程序本身无故障，再做覆盖。

20.5　利用 Puppet 推送宿主机的非一致性配置

文件推送在 Puppet 中运用的非常多，但是在某些场景下，需要对不同的服务器推送适合自己的文件。比如，每套虚拟化集群配置的多路径脚本，在不同的集群上是不一样的，那么就需要在推送备份脚本的时候，安装集群单独定义，本节介绍在类似这样的应用场景下，如何实现非一致性的文件推送。

1. 自定义 Puppet 模板

Puppet 自定义模板的使用，是为了在某些情况下，根据服务器的配置或者角色推送有差异化的配置，通常自定义 Puppet 模板在 Puppet 模块里面使用。在本章的 20.1 节中介绍了模块基本的目录结构。一个模块下，一般包含三个目录，即 files、manifests 和 templates，其中 templates 目录用于存放模板文件，文件一般以 .erb 结尾，下面一个最简单的 rsync 的配置文件模板。

```
log file = /var/log/rsyncd.log
[backup]
comment = temporary file area
usechroot = true
```

1107114476@qq.com
dolphinebook.com

```
path = /backup
read only = no
hosts allow = <%=@rsync_server_allow_network%>
auth users = oneadmin
strict modes = true
uid = root
gid = root
secrets file = /etc/rsyncd.secrets
```

在这个模板里面定义了一个变量 rsync_server_allow_network，用于指定 rsync 服务允许哪些客户端连接。在使用的时候，只需要定义 $rsync_server_allow_network={ip address}，就能将这个变量赋值给 rsync 的配置文件，得到差异化推送的效果。像类似的例子很多，可以是服务的配置文件，也可以是脚本的变量。

2. 自定义 Puppet Facter 获取特定信息

Puppet 中使用 Facter 可以得到很多系统相关的信息，在 Puppet Node 上直接运行 Facter 命令可以得到很多信息，不同版本的 Facter 获得的信息量稍有不同，一般都在 85 个左右，内容基本如下：

```
[root@puppetclient ~]# facter
architecture => x86_64
hostname =>puppetclient
kernelrelease => 2.6.32-431.el6.x86_64
lsbdistdescription =>CentOS release 6.5 (Final)
manufacturer => Red Hat
memoryfree => 1.66 GB
memorysize => 1.83 GB
network_eth0 => 172.16.100.0
operatingsystem =>CentOS
operatingsystemmajrelease => 6
operatingsystemrelease => 6.5
processorcount => 2
uptime => 12 days
<!-- 以上这些项，只是 facter 能获取信息的一部分 -- ! >
```

Facter 能获取到的信息很多，包括系统版本、内存使用情况、CPU 核数、主机名等。尽管如此，还是有一些信息是 Facter 没有获取到的，那么这个时候，可以通过自定义 Facter 来获取需要的信息。

Facter 分为三类，即核心 facts、自定义 facts 和外部 facts，核心 facts 的默认是自带的，不同的 Facter 版本自带的 facts 略有区别。可以定义的 facts 主要是后面两类，这两类 facts 定义的位置有所不同。自定义 facts 可以放在模块下的 facts.d 或者 lib/facter 目录下，目录结构如下：

```
[root@puppetmaster ~]# tree /etc/puppet/modules/module-name/
├── facts.d
```

1107114476@qq.com
dolphinebook.com

```
|    └──── ifconfigpath.rb
├──── files
├──── lib
|    └──── facter
|         └──── rubypath.rb
├──── manifests
|    └──── init.pp
└──── templates
```

提示　在 Puppet 3.0 之前，需要在 puppet.conf 文件中添加 pluginsync = true 来同步 facter，Puppet 3.0 之后，默认不需要再添加这个配置。凡是定义在模块下的 facter，都会同步到 Puppet 客户端上，定义在 facts.d 内的会同步到 Node 的 /var/lib/puppet/facts.d 目录下，定义在 lib/facter 内的 Facter 会同步到客户端 /var/lib/puppet/lib/facter 目录下。

外部 facts 可以放在 /etc/facter/facts.d 目录下，这个目录是 facter 默认的，凡是在这个目录中定义的 fact，直接通过 Facter 命令即可查到变量内容。同时，还支持 Python 编写的脚本，只需输出的格式为字典格式即可：

```
#!/usr/bin/env python
data = {"key1" : "value1", "key2" : "value2" }
for k in data:
print "%s=%s" % (k,data[k])
```

同时，还支持一些标准的文件格式，如 .yaml、.json、.txt。

```
    .yaml:
        ---
        key1: val1
        key2: val2
        key3: val3
<!--.yaml 格式的文件如上 --!>
    .json: JSON data, in the following format:
        {
            "key1": "val1",
            "key2": "val2",
            "key3": "val3"
        }
<!--.json 格式的文件如上 --!>
    .txt: Key value pairs, in the following format:
      key1=value1
      key2=value2
      key3=value3
<!--.txt 格式的文件如上 --!>
```

但是，定义在这个目录下的 fact 内容不会自动同步到其他客户端，需要配置相应的

1107114476@qq.com
dolphinebook.com

Puppet 任务做文件集中分发。

fact 的使用非常简单，可以通过变量的形式直接引用，下面举个简单的例子说明一下。

在使用多链路 ISCSI 的时候，会用到 Multipath 服务来配置多链路负载均衡。配置 Multipath 的时候，需要获取磁盘的 wwid，而 wwid 需要通过在 Node 上执行 /sbin/scsi_id -g /dev/sdb 来获取。例如，通过多链路挂载 ISCSI 之后，会出现两个设备 sdb 和 sdc，可以定义一个 fact，用来获取 wwid，并将这个获取到的 wwid 直接放到 Multipath 的配置文件中。fact 的内容如下：

```
Facter.add(:getscsiid)do          //申明一个 fact 变量，为 getscsiid
    setcode do
            diskname = Facter.value('blockdevices')
            casediskname
            when "sda,sdb,sdc"    //判断系统中存在 sda，sdb，sdc 的时候获取 wwid
                    Facter::Util::Resolution.exec("/sbin/scsi_id -g /dev/sdb")
            else                  //否则返回一次自定义的 wwid
                    Facter::Util::Resolution.exec("echo 'abcdefgABCDEFG'")
            end
    end
end
```

然后，在 Multipath 的配置文件模板 multipath.conf.erb 中，通过 <%=@getscsiid%> 直接应用即可。

```
multipaths {
    multipath {
            wwid                "<%=@getscsiid%>"
            alias               ipsanmp1
    }
}
```

如果要验证在模块下定义的 fact 取到的值是什么，需要先指定一下环境变量。

```
export FACTERLIB="/var/lib/puppet/lib/facter"
```

之后，使用 factergetscsiid 即可查看获取到的 wwid 值。

3. Puppet 变量定义

Puppet 变量可以有很多种用途，通常最常用的用途包括模板赋值和条件判断。

模板赋值，就是在推送某些服务的配置文件，或者脚本的时候，将需要推送的内容定义为模板文件，需要变的部分用 <=@variable> 变量来替换，然后在 puppetnode.pp 文件中使用 $ variable='abc' 来赋值，这种用途在实际环境中应用的非常广泛。

条件判断，在某些场景上，需要在节点中定义一些变量，来指明一些参数，Puppet 再通过这些参数变量来判断，具体执行哪些任务，详细的实例可以参考本章第 20.3 节，其中介绍的任务分层推送设计思路，就通过机房变量和服务器角色变量来判断执行哪些任务。

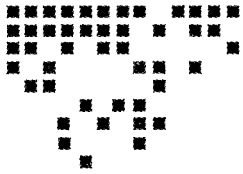
1107114476@qq.com
dolphinebook.com

20.6　本章小结

本章介绍了自动化运维在宿主机管理中的运用，主要以自动化工具 Puppet 作为基础，介绍了如何设计和优化 Puppet 的任务架构，用 Puppet 来解决在平时运维过程中一些繁琐和低效的工作内容。Puppet 任务架构，是非常值得研究的内容，一个好的任务架构，绝对可以在长远的工作中起到非常大的作用，通过本章的介绍，希望读者能从中感受到 Puppet 的魅力，结合自己的应用场景，设计出适合自己的任务架构，使运维工作变得更简单高效。

宿主机维护管理问题解决后，面临的下一个问题是监控、报警、应急处理，下章将介绍虚拟化的监控、报警、应急方案。

1107114476@qq.com
dolphinebook.com



Chapter 21 第 21 章

虚拟化监控、报警与应急响应方案

业务迁移到虚拟化环境后，最大的风险是宿主机出现问题的时候，对业务的影响面要比业务原来在物理机上大很多。因为将业务虚拟化，几台虚拟机集中在一台宿主机上。未做虚拟化之前，一台服务器故障，影响的是一项业务，迁移到虚拟化后，一台宿主机故障，影响的是好几项业务。这就要求虚拟化相关的监控、报警、应急响应方案一定要完善，并能定期进行演习。本章为读者介绍虚拟化监控、报警应该关注哪些点，以及如何制定和实施应急响应方案等。

21.1 虚拟化监控内容

虚拟化的监控可以分为以下两个方面：

- 宿主机的硬件监控，监控的硬件包括 CPU、内存、网卡、硬盘、电源、风扇等硬件。
- 宿主机的系统监控，监控的主要是系统性能，包括 CPU 利用率、内存利用率、网卡利用率、磁盘利用率等指标。

下面分别介绍每个监控项的注意点及监控办法。

21.1.1 宿主机 CPU 监控要点

宿主机 CPU 主要关注以下指标：

- 进程运行情况。
- CPU 利用率。
- 中断及上下文切换。

1107114476@qq.com
dolphinebook.com

提示　宿主机 CPU 最关注的监控指标是 CPU 利用率。

1. 使用命令查看 CPU 的进程情况

系统设计的目标就是为应用进行服务，进程就是系统正在运行中的应用程序，进程有以下 5 种状态。

- TASK_RUNNING：运行状态。
- TASK_INTERRUPTIBLE（可中断）：被阻塞，进程正在睡眠或者进程正在等待某些条件，一旦条件触发，内核会把进程投入运行。
- TASK_UNINTERRUUPTIBLE（不可中断）：与可中断状态类似，但是不会因为接到信号被唤醒。这个状态的进程必须在等待时不受干扰或者等待事件很快就会发生时出现。由于处于此状态的任务对信号不做响应，所以使用得比较少。
- TASK_ZOMBIE(僵死)：进程已经结束了，但是其父进程还没有进行 wait4() 系统调用。为了父进程能够获知它的消息，子进程的进程描述符仍然被保留着。一旦父进程调用了 wait4()，进程描述符就会被释放。
- TASK_STOPPED（停止）：进程停止执行。

下面具体介绍如何利用命令来查看 CPU 的进程情况。

（1）使用 uptime 命令

uptime 命令是查看系统从启动到当前的时间，并简单显示系统负载情况，命令如下：

```
uptime
14:43:51 up 1 day, 5:29, 1 user, load average: 0.87, 1.44, 1.56
```

load average 是 1 分钟、5 分钟、15 分钟内平均加载到 CPU 的进程数。

（2）使用 top 命令

top 命令可以动态查看系统负载，查看进程对系统资源的消耗，命令如下：

```
top - 14:51:34 up 1 day, 5:36, 3 users, load average: 2.07, 1.71, 1.62
Tasks: 234 total, 1 running, 233 sleeping, 0 stopped, 0 zombie
cpu(s): 24.3%us, 21.8%sy, 0.0%ni, 51.4%id, 0.6%wa, 0.0%hi, 1.8%si, 0.0%st
Mem: 64937040k total, 48433524k used, 16503516k free, 200624k buffers
Swap: 8388600k total, 0k used, 8388600k free, 21541852k cached
 PID  USER PR NI VIRT  RES SHR S  %CPU  %MEM  TIME+    COMMAND
  2397 qemu 20 0 17.0g 11g  5120 S 173.6 18.3 2487:27 qemu-kvm
  2851 qemu 20 0 8113m 1.2g 5112 S 79.6 2.0 1199:09   qemu-kvm
  3252 qemu 20 0 7996m 970m 5140 S 58.4 1.5 805:08.56 qemu-kvm
  2525 qemu 20 0 8130m 774m 5112 S 46.5 1.2 860:03.92 qemu-kvm
  2336 qemu 20 0 8059m 553m 5124 S 24.6 0.9 485:02.34 qemu-kvm
.....
```

第一行和 uptime 相同。

第二行是进程的状态，分别是进程总数、正在运行多进程、休眠的进程、停止的进程、

1107114476@qq.com
dolphinebook.com

僵尸进程。

第三行与 CPU 相关，主要是各个项目对 CPU 时间的利用比例，比例中的字母代表如下含义。

- us：用户模式。
- sy：系统模式。
- ni：nice 低优先用户模式。
- id：空闲。
- wa：IO 等待。
- hi：服务系统中断。
- si：服务系统软中断。
- st：保留（给其他 DOMU 实例保留的时间）。

第四行、第五行是系统内存的使用情况。第六行是每个进程的详细情况，含义如下。

- PID：进程编号。
- PPID：父进程编号。
- USER：进程用户名。
- PR：进程的优先级别，数字越小，级别越高。
- NI：进程 nice 值，负值意味着更高消耗，正值意味着更低消耗，0 意味着没有调整任务优先级。
- VIRT：虚拟内存使用。
- RES：物理内存使用。
- SHR：共享内存使用。
- S：进程状态，它包括以下状态：S，休眠；R，运行；Z，僵尸状态；T，被追踪或者停止状态；D，不可中断状态。
- %CPU：CPU 利用率，这个值有可能超过 100%，因为这个数值是几个 CPU 利用率的累加。
- %MEM：内存使用百分比。
- TIME+：总的 CPU 时间累加值。
- COMMAND：进程命令。

（3）使用 ps 命令

ps 命令可以查看进程对资源的消耗，和 top 命令不同的是，top 是动态变化，ps 只显示命令运行时的数据，运行 ps 命令，结果如下：

```
ps -aux
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.0 19356 1528 ? Ss 16:07 0:00 /sbin/init
root 2 0.0 0.0 0 0 ? S 16:07 0:00 [kthreadd]
root 3 0.0 0.0 0 0 ? S 16:07 0:00 [migration/0]
```

1107114476@qq.com
dolphinebook.com

```
root 4 0.0 0.0 0 0 ? S 16:07 0:00 [ksoftirqd/0]
root 5 0.0 0.0 0 0 ? S 16:07 0:00 [migration/0]
root 6 0.0 0.0 0 0 ? S 16:07 0:00 [watchdog/0]
root 7 0.0 0.0 0 0 ? S 16:07 0:00 [migration/1]
root 8 0.0 0.0 0 0 ? S 16:07 0:00 [migration/1]
...
```

输出内容与 top 命令差不多，进程状态说明如下。

- D：不可中断。
- R：正在运行，或在队列中的进程。
- S：处于休眠状态。
- T：停止或被追踪。
- Z：僵尸进程。
- W：进入内存交换（从内核 2.6 开始无效）。
- X：死掉的进程。
- <：高优先级。
- N：低优先级。
- L：有些页被锁进内存。
- S：包含子进程。
- +：位于后台的进程组、多线程、克隆线程。

2. 查看 CPU 利用率、中断、上下文切换

简单地说，CPU 中断是指 CPU 接收到来自硬件或者软件的请求信号，提示发生了某个事件，应该被处理。一个逻辑 CPU 在同一时间只能运行一个进程，如果要运行的进程大于逻辑 CPU 的数量，就会有进程在备用状态（被阻塞），进程从备用状态切换到运行状态，或者从运行状态切换到阻塞状态，就是上下文切换，这样做的目的是让每个进程都能使用 CPU，合理分配 CPU 时间片。

一般来说，大量的上下文切换会影响性能，但是，这也要结合 CPU 利用率来看，如果 CPU 利用率不高，大量的上下文切换也是能接受的。

（1）vmstat 命令

vmstat 命令主要用于查看系统 CPU 消耗及中断情况，使用 vmstat 命令 2 秒钟刷新一次，运行数据如下：

```
vmstat 2
procs --memory--- ---swap-- -----io---- --system-- ---cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
1 0 0 50165268 46644 4664156 0 0 334 16 1726 201 7 7 85 1 0
0 0 0 50165260 46644 4664184 0 0 0 0 35337 76635 2 2 95 0 0
6 0 0 50165096 46652 4664184 0 0 0 14 35202 74826 3 7 89 0 0
0 0 0 50165000 46660 4664184 0 0 0 18 35678 74779 2 5 93 0 0
0 0 0 50165000 46660 4664184 0 0 0 72 34344 76017 3 3 94 0 0
```

1107114476@qq.com
dolphinebook.com

```
1 0 0 50165000 46668 4664180 0 0 0 8 35118 73650 4 7 89 0 0
0 0 0 50164976 46668 4664196 0 0 0 78 37400 74469 5 9 86 0 0
0 0 0 50165188 46676 4664204 0 0 0 46 35583 74113 4 6 88 2 0
8 0 0 50165212 46684 4664208 0 0 0 62 35553 75561 4 7 88 1 0
```

和中断、上下文切换相关的选项是 procs、system 和 CPU，这 3 项的含义如下。

- procs：r 为运行队列中的进程数量，b 为等待 I/O 的进程数。
- system：in 为中断数，cs 为上下文切换。
- cpu：us 为用户进程时间百分比，sy 为系统进程时间百分比，id 为空闲时间，wa 为 I/O 等待时间。

（2）mpstat 命令

通过 mpstat 命令可以看到每个核的 CPU 中断、上下文切换情况，运行数据如下：

```
mpstat -P ALL
Linux 2.6.32-431.23.3.el6.x86_64 (bg-t-11) 10/09/2014 _x86_64_ (8 CPU)

04:43:38 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %idle
04:43:38 PM all 0.51 0.00 6.78 0.97 0.00 0.01 0.00 5.67 86.07
04:43:38 PM 0 0.44 0.00 6.52 1.08 0.00 0.01 0.00 5.57 86.39
04:43:38 PM 1 0.30 0.00 6.77 1.10 0.00 0.02 0.00 5.69 86.13
04:43:38 PM 2 0.37 0.00 6.41 1.19 0.00 0.02 0.00 5.53 86.49
04:43:38 PM 3 0.34 0.00 6.86 1.09 0.00 0.01 0.00 5.62 86.08
04:43:38 PM 4 0.92 0.00 8.11 0.80 0.00 0.01 0.00 5.81 84.35
04:43:38 PM 5 0.55 0.00 6.30 0.81 0.00 0.01 0.00 5.59 86.75
04:43:38 PM 6 0.52 0.00 6.69 0.91 0.00 0.00 0.00 5.74 86.14
04:43:38 PM 7 0.62 0.00 6.49 0.77 0.00 0.00 0.00 5.78 86.33
```

（3）sar 命令

sar 命令同 mpstat 命令一样，也可以看到 CPU 每个核的中断、上下文切换情况，运行数据如下：

```
sar -P ALL 2
Linux 2.6.32-431.23.3.el6.x86_64 (bg-t-11) 10/09/2014 _x86_64_ (8 CPU)
04:45:47 PM CPU %user %nice %system %iowait %steal %idle
04:45:49 PM all 3.91 0.00 6.76 0.13 0.00 89.20
04:45:49 PM 0 1.62 0.00 1.62 0.00 0.00 96.76
04:45:49 PM 1 1.07 0.00 2.14 0.00 0.00 96.79
04:45:49 PM 2 1.14 0.00 1.70 0.57 0.00 96.59
04:45:49 PM 3 1.69 0.00 1.12 0.00 0.00 97.19
04:45:49 PM 4 11.44 0.00 12.71 0.00 0.00 75.85
04:45:49 PM 5 4.33 0.00 8.17 0.00 0.00 87.50
04:45:49 PM 6 3.33 0.00 14.76 0.00 0.00 81.90
04:45:49 PM 7 3.48 0.00 8.96 0.00 0.00 87.56
```

（4）top 命令

通过 top 命令可以看到 CPU 每个核的中断、上下文切换情况，运行数据如下：

1107114476@qq.com
dolphinebook.com

```
top - 16:46:26 up 39 min, 2 users, load average: 0.14, 0.08, 0.60
Tasks: 231 total, 1 running, 230 sleeping, 0 stopped, 0 zombie
Cpu0 : 13.7%us, 12.8%sy, 0.0%ni, 73.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu1 : 4.6%us, 7.9%sy, 0.0%ni, 87.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu2 : 0.7%us, 1.1%sy, 0.0%ni, 98.2%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu3 : 1.8%us, 1.4%sy, 0.0%ni, 96.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu4 : 5.2%us, 11.4%sy, 0.0%ni, 83.4%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu5 : 1.8%us, 2.2%sy, 0.0%ni, 96.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu6 : 1.1%us, 0.7%sy, 0.0%ni, 98.2%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7 : 1.4%us, 1.8%sy, 0.0%ni, 96.1%id, 0.7%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 64937040k total, 14771044k used, 50165996k free, 47668k buffers
Swap: 8388600k total, 0k used, 8388600k free, 4665416k cached
```

如果要长期查看 CPU 各个指标的情况，可以使用监控工具，如使用 Cacti。如图 21-1 所示的是 CPU 的利用率、中断、上下文切换、I/OWait 的监控图。

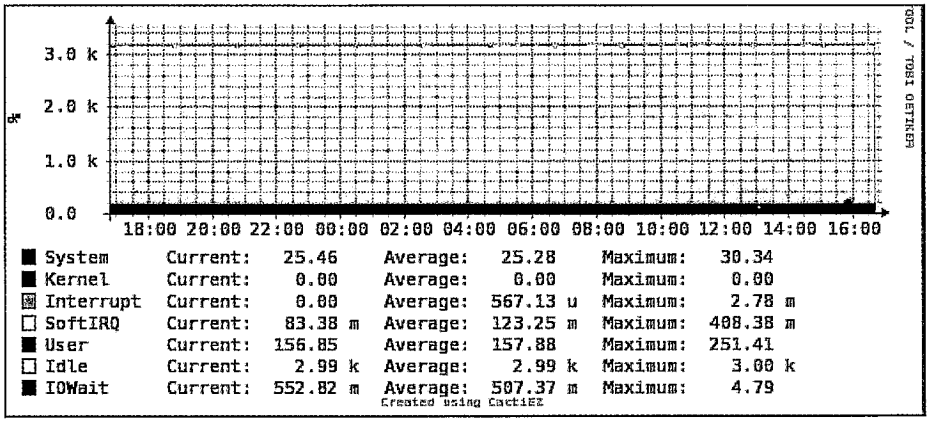


图 21-1　CPU 监控综合图形展示

图 21-2 是 CPU0 的利用率监控图。

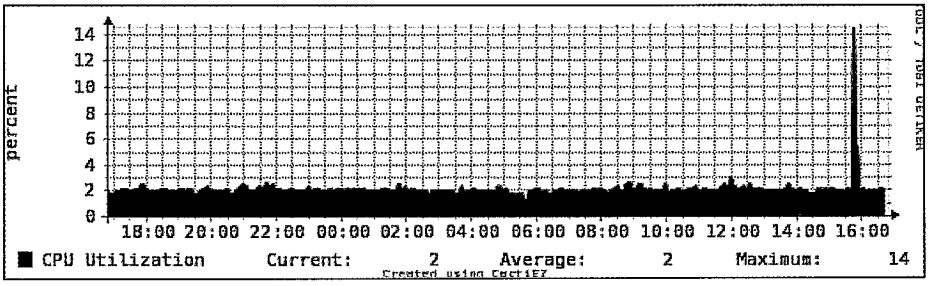


图 21-2　单 CPU 利用率监控图形展示

21.1.2　宿主机内存监控注意点

宿主机内存一般需要监控以下数据：

1107114476@qq.com
dolphinebook.com

- 物理内存的情况。
- 内存使用情况。
- Cache 使用情况。
- Swap 使用情况。

1. 物理内存的情况

为了有效使用内存，内核把物理内存分页使用，一般 32 位系统支持 4KB 的分页，64 位系统支持 8KB 的分页。

内核把内存页分成以下不同的区（zone）。

- ZONE_DMA：这个区用来执行 DMA 操作，以便设备用来访问。
- ZONE_NORMAL：这个区是给应用使用的页。
- ZONE_HIGHMEM：高端内存，内核通过映射的方式使用。

X86 上 ZONE_DMA 为 0 ~ 16MB 的内存范围，ZONE_HIGHMEM 为高于 896MB 的物理内存，中间的是 ZONE_NORMAL 区。

物理内存不够用时，内核用 swap 模拟内存使用。

物理内存信息查看方法如下：

① 使用 dmidecode 命令可以查看物理内存的详细信息。

```
#dmidecode
...
Physical Memory Array
        Location: System Board Or Motherboard
        Use: System Memory
        Error Correction Type: Single-bit ECC
        Maximum Capacity: 64 GB
        Error Information Handle: Not Provided
        Number Of Devices: 8
...
Handle 0x1100, DMI type 17, 23 bytes
Memory Device
        Array Handle: 0x1000
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 8192 MB
        Form Factor: FB-DIMM
        Set: 1
        Locator: DIMM 1A
        Bank Locator: Not Specified
        Type: DDR2 FB-DIMM
        Type Detail: Synchronous
        Speed: 667 MHz
...
```

可以看到总共有 64GB 的内存，8 根内存条，每条是 8192MB。内存的频率等物理信息

1107114476@qq.com
dolphinebook.com

也可以看到。

② 使用 /proc/meminfo 命令可以看到系统对内存使用的详细情况，包括内存使用信息、swap 使用信息等。

```
#cat /proc/mrminfo
...
MemTotal: 64937040 kB
MemFree: 31646428 kB
Buffers: 224524 kB
Cached: 9879320 kB
SwapCached: 0 kB
Active: 23831572 kB
Inactive: 8072936 kB
Active(anon): 21801004 kB
Inactive(anon): 156 kB
Active(file): 2030568 kB
Inactive(file): 8072780 kB
Unevictable: 0 kB
Mlocked: 0 kB
SwapTotal: 8388600 kB
SwapFree: 8388600 kB
Dirty: 48 kB
Writeback: 0 kB
AnonPages: 21800656 kB
Mapped: 19268 kB
Shmem: 504 kB
Slab: 296572 kB
SReclaimable: 155340 kB
SUnreclaim: 141232 kB
KernelStack: 2960 kB
PageTables: 64700 kB
NFS_Unstable: 0 kB
Bounce: 0 kB
WritebackTmp: 0 kB
CommitLimit: 40857120 kB
Committed_AS: 124825440 kB
VmallocTotal: 34359738367 kB
VmallocUsed: 620248 kB
VmallocChunk: 34359059632 kB
HardwareCorrupted: 0 kB
AnonHugePages: 0 kB
HugePages_Total: 0
HugePages_Free: 0
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 2048 kB
DirectMap4k: 8496 kB
DirectMap2M: 66050048 kB
...
```

1107114476@qq.com
dolphinebook.com

2. 内存的使用情况

内存关注的几个重要指标可以通过 free 命令查看，如图 21-3 所示。

```
             total       used       free     shared    buffers     cached
Mem:      64937040   33291156   31645884          0     224580    9879844
-/+ buffers/cache:   23186732   41750308
Swap:      8388600          0    8388600
```

图 21-3 内存利用率查看

① 从系统角度看，mem 行分别是系统总的内存 total、被使用的内存 used、剩余的内存 free，total=used+free。

- shared 表示被几个进程共享的内存，现在已经放弃了，所以一般显示是 0。
- buffers 用于存放要输出到 disk（块设备）的数据。
- cached 存放从 disk 上读出的数据。
- buffers、cached 这二者是为了提高 I/O 性能，由系统管理。

② 从应用的角度看，–/+ buffers/cache 行是内存的使用情况，具体如下：

- –buffers/cache，表示一个应用程序认为系统被用掉多少内存。
- +buffers/cache，表示一个应用程序认为系统还有多少内存。

③ swap 行分别给出了交换的总量（tatal）、使用量（used）和剩余空闲的交换区（free）。

可以使用如下命令释放掉被系统 Cache 占用的数据：

```
echo 3>/proc/sys/vm/drop_caches
```

使用监控工具可以长期观察系统内存使用。如图 21-4 所示，通过 Cacti，内存的使用量、缓存等信息都可以通过监控软件长期监控。

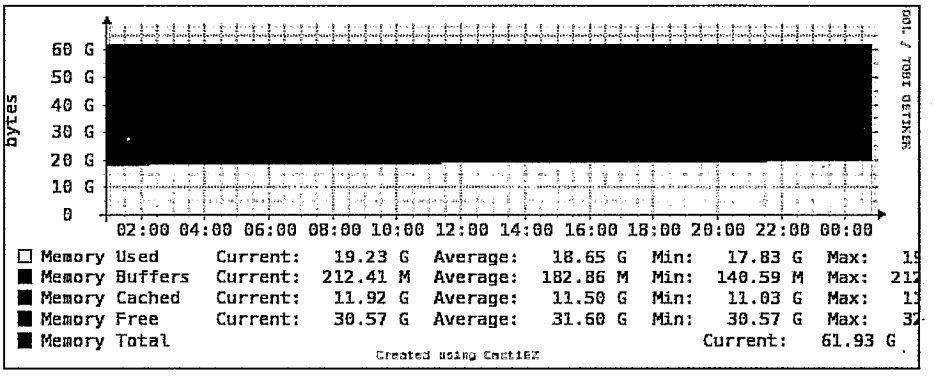


图 21-4 通过监控工具监控内存

21.1.3 宿主机网络监控注意点

宿主机的网络监控关注以下两个指标。

1107114476@qq.com
dolphinebook.com

- 网络吞吐量：一秒发送接收多少字节。
- 网络发包：一秒发送接收多少包数。

提示　一般网络流量和业务关系非常密切，故建议也对网络的急剧变化进行监控。

使用命令行工具监控网络。

（1）通过 ethtool 命令查看网卡状态

例如，查看 eth0 状态，运行 ethtool eth0 命令，如图 21-5 所示。

```
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 100Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 1
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: Unknown
        Supports Wake-on: g
        Wake-on: g
        Link detected: yes
```

图 21-5　通过 ethtool 命令查看网卡连接速度

提示　宿主机正式上线前，建议先使用 ethtool 命令查看下网卡的实际速率，因为经常发生由于网线质量问题，导致千兆网卡工作在百兆的情况。

（2）通过 ifconfig 命令查看网卡性能数据

例如，通过 ifconfig eth0 命令来查看 eth0 的数据，如图 21-6 所示。

```
eth0      Link encap:Ethernet  HWaddr 00:21:5A:D0:32:E2
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6111873092 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6108937396 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:461332915825 (429.6 GiB)  TX bytes:498457288914 (464.2 GiB)
```

图 21-6　通过 ifconfig 命令查看网卡发包吞吐

ifconfig 查看的是累加量，从开机开始一直在累加。

（3）通过 sar 命令查看网卡性能数据

sar 命令可以查看每个网卡的流量数据，图 21-7 所示为命令 sar -n DEV 2 的运行结果。

```
04:09:04 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
04:09:06 PM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
04:09:06 PM      eth0   5200.48   5206.25    380.84    413.50      0.00      0.00      1.44
04:09:06 PM      eth1      0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

图 21-7　通过 sar 命令查看网卡性能

1107114476@qq.com
dolphinebook.com

查看网络还有许多命令，如 iftop 等，这里就不详细介绍了。

也可以通过监控工具，长期用图形方式查看，如图 21-8 所示为 Cacti 的网卡监控。

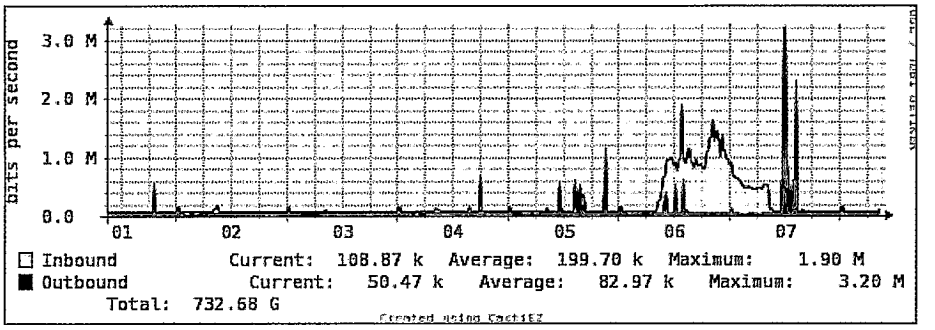


图 21-8 通过监控工具监控网卡性能

21.1.4 宿主机磁盘监控注意点

宿主机的磁盘监控的指标比较多，具体如下。

- 磁盘吞吐：一秒读 / 写多少字节。
- 磁盘 IOPS：一秒读 / 写多少次。
- 磁盘利用率：磁盘操作时间和空闲时间的百分比。
- 磁盘服务时间：一般认为比较理想的是少于 20ms。
- 磁盘扇区读写：一秒读 / 写多少个扇区。

监控方法主要包括命令行工具和使用监控软件。

使用命令行工具包括以下方面：

① 使用 iostat 命令查看磁盘性能，如图 21-9 所示。

```
Linux 2.6.32-431.23.3.el6.x86_64 (bg-t-11)      10/20/2014      _x86_64_      (8 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          16.45    0.00   17.23    0.39    0.00   65.92

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
cciss/c0d0        0.01     0.40    0.02    0.29     0.61     2.73    21.67     0.00   16.11   4.74   0.15
cciss/c0d1        0.02    14.03    0.20    4.08    13.45    72.45    40.13     0.15   35.20   8.76   3.75
dm-0              0.00     0.00    0.00    0.00     0.00     0.00     8.00     0.00    2.41   1.57   0.00
dm-1              0.00     0.00    0.03    0.68     0.58     2.73     9.27     0.05   72.83   2.04   0.15
dm-2              0.00     0.00    0.23   18.11    13.45    72.45     9.37     1.32   72.22   2.04   3.75
dm-3              0.00     0.00    0.00    0.00     0.00     0.00     7.98     0.00    1.49   0.93   0.00
```

图 21-9 通过 iostat 命令监控磁盘性能

图中所示各项含义如下。

- rrqm/s、wrqm/s：每秒合并的读 / 写操作次数。
- r/s、w/s：每秒到设备的读 / 写次数。
- rkB/s、wkB/s：每秒读 / 写千字节数。
- avgrq-sz：平均每次设备读 / 写操作的数据大小（扇区）。

1107114476@qq.com
dolphinebook.com

- avgqu-sz：平均队列长度。
- await：平均每次设备 I/O 操作的等待时间（毫秒），整个 I/O 完成时间包括等待时间和服务时间。
- svctm：平均每次设备 I/O 操作的服务时间（毫秒），I/O 操作的时间。
- %util：一秒内有百分之多少的时间用于 I/O 操作，或者说一秒内有多少时间 I/O 队列是非空的。

② 使用 sar 命令查看磁盘性能数据，如图 21-10 所示的是命令 sar -d 2 的输出结果，各项指标和 iostat 命令指标相近。

```
03:49:59 PM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
03:50:01 PM  dev104-0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
03:50:01 PM dev104-16      2.54      0.00     36.55     14.40      0.03     13.40      6.80      1.73
03:50:01 PM  dev253-0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
03:50:01 PM  dev253-1      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
03:50:01 PM  dev253-2      4.57      0.00     36.55      8.00      0.06     13.00      3.78      1.73
03:50:01 PM  dev253-3      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

图 21-10　通过 sar 命令监控磁盘性能

使用图形监控工具，如图 21-11 所示，21.2 节对使用监控工具监控磁盘有更详细的介绍。

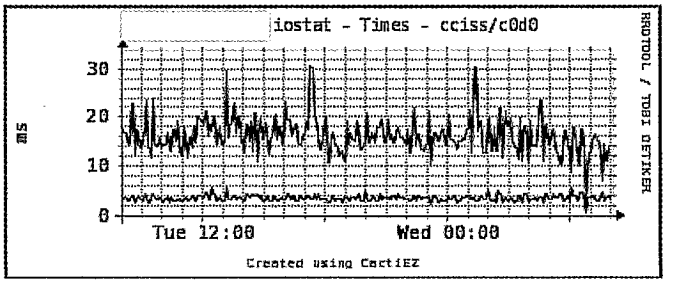


图 21-11　通过监控工具监控磁盘性能

21.1.5　宿主机层面监控虚拟机的方法

从宿主机层面也可以看到虚拟机的一些性能数据，主要有以下 3 种途径：

- VirtManager 的虚拟机管理界面。
- 使用 virt-top 命令。
- 利用 Libvirt 的 API 接口。

通过 VirtManager 的虚拟机管理界面，可以实时看到虚拟机的 CPU 利用率、内存利用率、磁盘吞吐、网络吞吐，但是不能看到更详细的信息，如每个 CPU 的利用率情况，CPU 的中断、上下文切换情况，也不能查看历史情况，如图 21-12 所示。

使用 virt-top 命令可以查看更详细的信息，virt-top 命令和 Linux 系统的 top 命令非常类似，运行后界面如图 21-13 所示。

按 1 键，可以查看每个虚拟机对每个 CPU 的详细使用情况，如图 21-14 所示。

1107114476@qq.com
dolphinebook.com

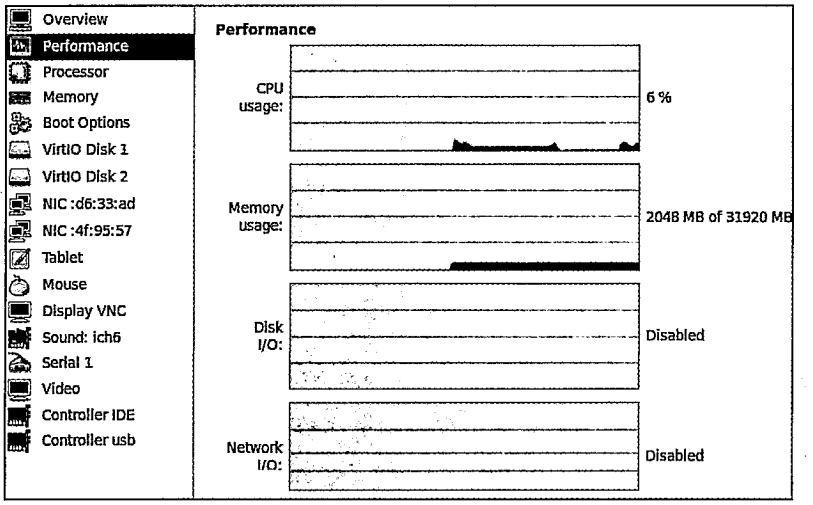


图 21-12 通过 VirtManager 查看虚拟机性能

```
virt-top 21:26:25 - x86_64 32/32CPU 1200MHz 31920MB 4.4%
6 domains, 5 active, 5 running, 0 sleeping, 0 paused, 1 inactive D:0 O:0 X:0
CPU: 1.6%  Mem: 25600 MB (25600 MB by guests)

   ID S RDRQ WRRQ RXBY TXBY %CPU %MEM     TIME   NAME
   10 R    0    3 6527 4267  0.9  6.0 897:34:00 windows-9
    5 R    0    0 7156 8539  0.2 22.0 114:01:44 centos6464-12
   11 R    0   21 4956 5941  0.2  6.0  81d11:14 centos6464-11
    4 R    0    2 7372 8789  0.2 22.0 106:35:41 centos6464-14
    3 R    0    1  15K 9723  0.2 22.0 103:31:48 centos6464-13
    -                                           (centos6464)
```

图 21-13 通过 virt-top 命令查看虚拟机性能

```
virt-top 21:27:07 - x86_64 32/32CPU 1200MHz 31920MB 1.1% 0.9% 4.4%
6 domains, 5 active, 5 running, 0 sleeping, 0 paused, 1 inactive D:0 O:0 X:0
CPU: 9.0%  Mem: 25600 MB (25600 MB by guests)

PHYCPU %CPU windows-9    centos6464-11 centos6464-13 centos6464-14 centos6464-12
   0    9.7  5.2  5.2    0.5  0.5      0.4  0.2      1.3  1.3      2.4  2.4
   1    8.1  3.8  3.2    0.4  0.2      0.2  0.2      1.2  1.1      2.6  2.5
   2    4.8  2.8  0.3    0.6  0.6      0.3  0.3      0.5  0.5      0.6  0.6
   3    3.9  0.5  0.3    0.8  0.8      0.3  0.3      2.2  2.2      0.1  0.1
   4   30.2  0.0  0.0   20.7 20.7      1.9  1.9      4.0  4.0      3.6  3.6
   5    1.0                           1.0  1.0      0.0  0.0      0.0  0.0
   6    0.2  0.1  0.1                 0.0  0.0                    0.0  0.0
   7    8.6                 8.4  8.4                 0.1           0.1  0.0
   8   12.1  1.5  1.5    9.5  9.5      0.0  0.0                    1.1  1.1
   9    2.3  1.6  0.6    0.7  0.7                                  0.0  0.0
  10   31.3               31.3 31.3                               0.0  0.0
  11    0.0                                                      0.0  0.0
  12    2.0                 2.0  1.9                             0.0  0.0
  13    0.0
  14    0.0
  15    7.8                 7.8  7.8
  16    9.9  5.0  3.3    0.0                 0.2  0.1      3.7  3.7      1.0  1.0
  17   10.1  0.7  0.5    7.6  7.5            0.2  0.2      1.2  1.2      0.5  0.5
  18   20.7  1.2  1.0   14.7 14.7            0.8  0.8      1.6  1.6      2.5  2.5
  19    5.5  2.0         3.0  3.0            0.1  0.1      0.4  0.4      0.0  0.0
  20    4.9               0.7  0.6            1.7  1.7      0.5  0.5      2.0  2.0
  21    0.0                                   0.0  0.0
  22    0.1               0.1
  23    1.0  0.0  0.0    1.0  0.6                           0.0  0.0      0.0
  24   14.5               13.3 13.3            0.0  0.0                    1.2  1.2
  25   30.6  0.1         30.5 30.5
  26   23.9               23.9 23.9                                       0.0  0.0
  27   22.6               22.6 22.5            0.0  0.0
  28    0.0
  29    0.0
  30    0.0
  31   18.8               18.8 18.8
```

图 21-14 通过 virt-top 命令查看虚拟机 CPU 详细性能

1107114476@qq.com
dolphinebook.com

按 2 键，可以查看每个虚拟机每个网卡的网络吞吐，如图 21-15 所示。

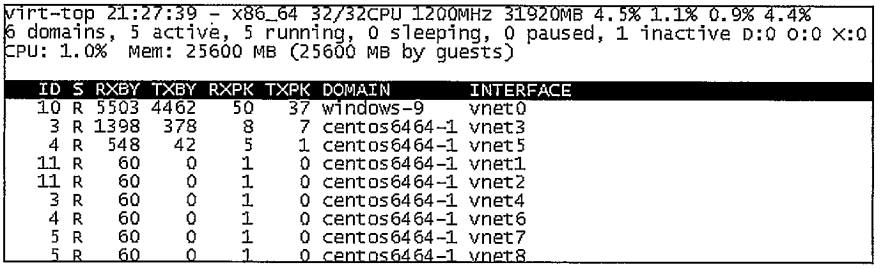


图 21-15　通过 virt-top 命令查看虚拟机网卡性能

按 3 键，可以查看每个虚拟机每块磁盘的吞吐，如图 21-16 所示。

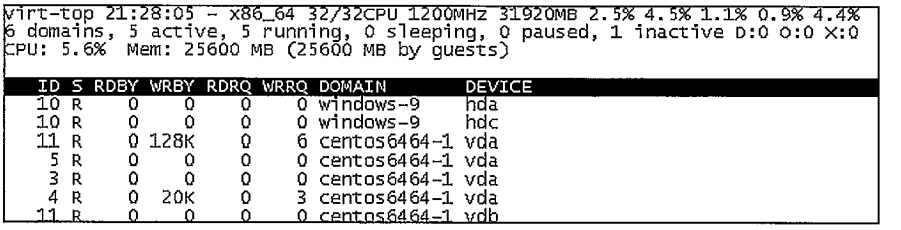


图 21-16　通过 virt-top 命令查看虚拟机磁盘性能

virt-top 默认 3 秒刷新一次，按 D 键，如图 21-17 所示，可以更改刷新频率。

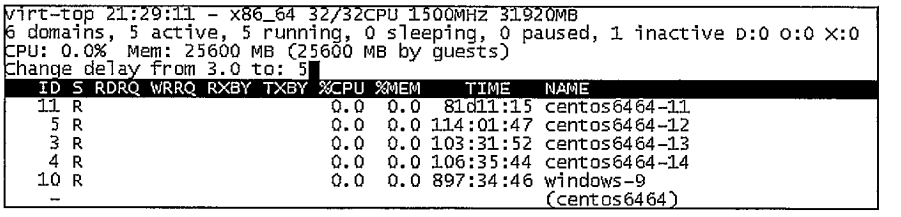


图 21-17　修改 virt-top 命令刷新频率

通过 Libvrit 的 API，也可以取得虚拟机的性能数据，然后通过监控工具以画图形式表现出来。一些第三方的监控工具也支持直接从 Libvirt 的 API 接口取得数据并画图。

通过宿主机层面监控到的虚拟机性能数据只能是比较粗的状态，没有办法取得详细的数据，如虚拟机 CPU 的上下文切换及中断数据、内存的缓存、swap 使用情况、每个进程对资源的消耗情况等，所以，要监控虚拟机的详细数据，还需要从虚拟机内部进行监控。

21.2　监控软件 Cacti、Zabbix

目前，常用的监控软件很多，比较常用的宿主机监控软件是 Cacti、Zabbix。下面分别介

1107114476@qq.com
dolphinebook.com

绍下用 Cacti、Zabbix 监控宿主机关键指标的方法。

21.2.1 用 Cacti 监控宿主机

Cacti 的特点是通过 SNMP 得到被监控设备数据，不需要在被监控的服务器上安装 agent，绘图漂亮。Cacti 安装资料比较多，本书不介绍如何安装搭建 Cacti，主要介绍如何给 Cacti 添加磁盘、CPU、内存模板，以方便能监控到宿主机更多的关心指标。

虚拟化的一个难点就是磁盘 I/O 的监控，一般需要关注磁盘 I/O 的 IOPS、延时、吞吐等指标。下面介绍 3 个 Cacti 模板，可以很好地监控磁盘、CPU、内存。

1. Cacti-iostat 模板

Cacti-iostat 模板用于对 iostat 命令的输出进行绘图，图 21-18 至图 21-23 所示的是 Cacti-iostat 模板监控到的磁盘 I/O 指标抓图，可以看到监控到的磁盘相关指标非常详细。

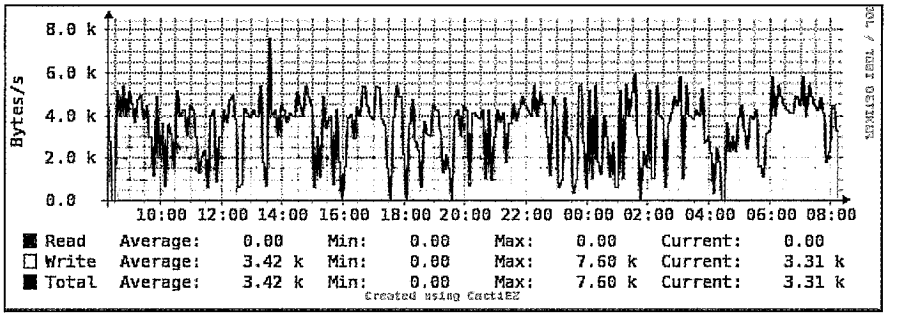


图 21-18 磁盘吞吐监控

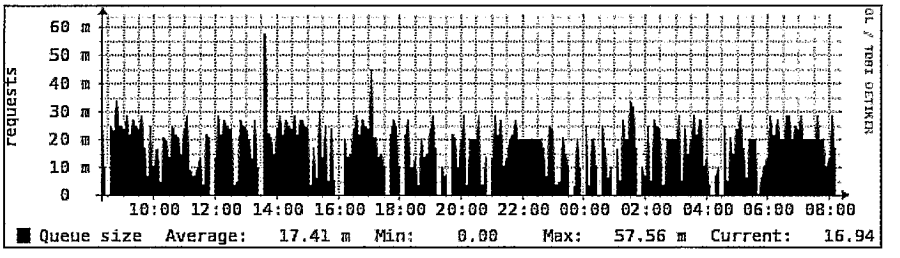


图 21-19 磁盘 Queue size 监控

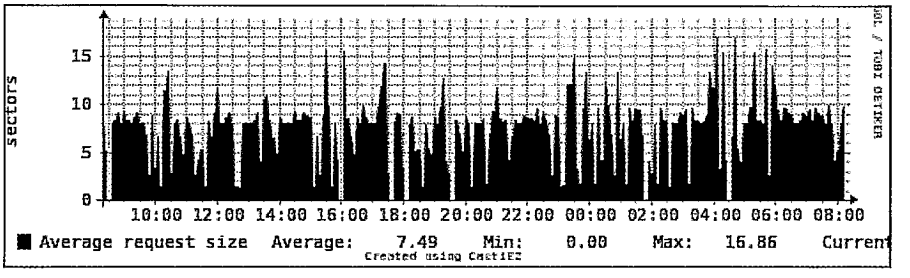


图 21-20 磁盘 Average 监控

1107114476@qq.com
dolphinebook.com

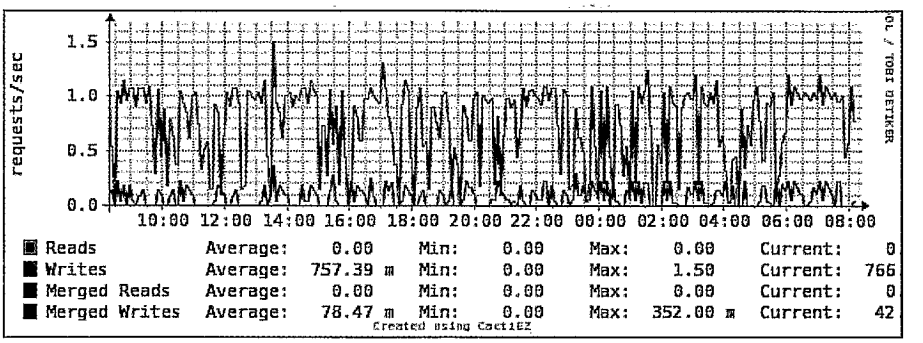


图 21-21　磁盘 IOPS 监控

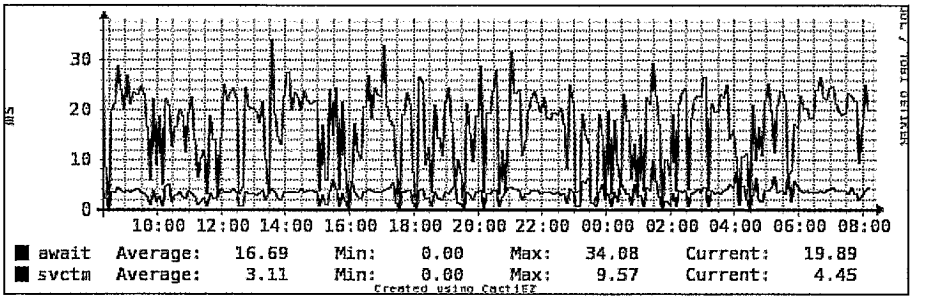


图 21-22　磁盘延时监控

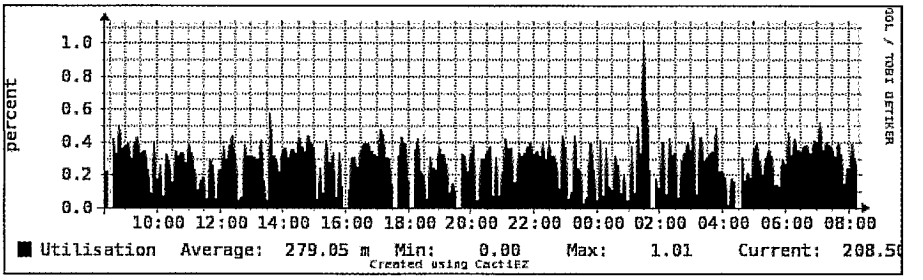


图 21-23　磁盘利用率监控

Cacti-iostat 模板配置稍微复杂一些，下面进行详细介绍。

（1）配置 Cacti 服务器端

Cacti-iostat 模板的下载地址是 https://github.com/markround/Cacti-iostat-templates，下载之后解压可以看到如下 3 个目录。

- [scripts]：客户端脚本。
- [snmp_queries]：snmp 配置文件。
- [templates]：Cacti 模板。

[scripts] 目录中是客户端脚本，[snmp_queries] 目录中是 snmp 配置文件，[templates] 目录中是 Cacti 模板。

1107114476@qq.com
dolphinebook.com

Cacti 模板有以下 13 个文件，首先，需要将 13 个模板文件依次导入 Cacti 服务器。

- cacti_data_query_iostat_-_kbytessec.xml
- cacti_data_query_iostat_-_queue_size.xml
- cacti_data_query_iostat_-_requestssec.xml
- cacti_data_query_iostat_-_request_size.xml
- cacti_data_query_iostat_-_times.xml
- cacti_data_query_iostat_-_utilisation.xml
- cacti_data_template_iostat_-_io.xml
- cacti_graph_template_iostat_-_bytessec.xml
- cacti_graph_template_iostat_-_queue_size.xml
- cacti_graph_template_iostat_-_requestssec.xml
- cacti_graph_template_iostat_-_request_size.xml
- cacti_graph_template_iostat_-_times.xml
- cacti_graph_template_iostat_-_utilisation.xml

操作步骤如下：

① 如图 21-24 所示，在 Cacti 控制台左侧的模板中选择导入模板。

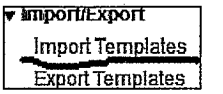


图 21-24 Cacti 导入模板

② 如图 21-25 所示，选择 Cacti-iostat 模板，再导入。

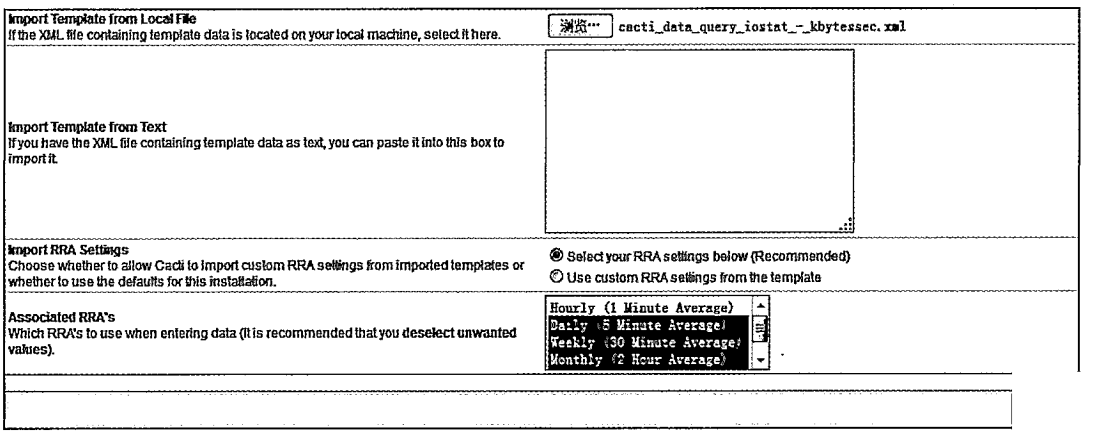


图 21-25 倒入 Cacti-iostat 模板

③ 如图 21-26 所示，导入后会提示成功。然后，重复以上几步，将所有的 Cacti-iostat 模板都导入。

（2）将 SNMP 配置文件放入 Cacti 服务器目录

将 snmp_queries/linux/ 目录下的 iostat.xml 文件传到 Cacti 服务器 resource/snmp_queries/ 目录下，并参考其他的 Cacti 的 xml 文件权限，chown 修改新导入的配置文件。

Cacti 服务器端配置成功后，被监控服务器也需要配置。

1107114476@qq.com
dolphinebook.com

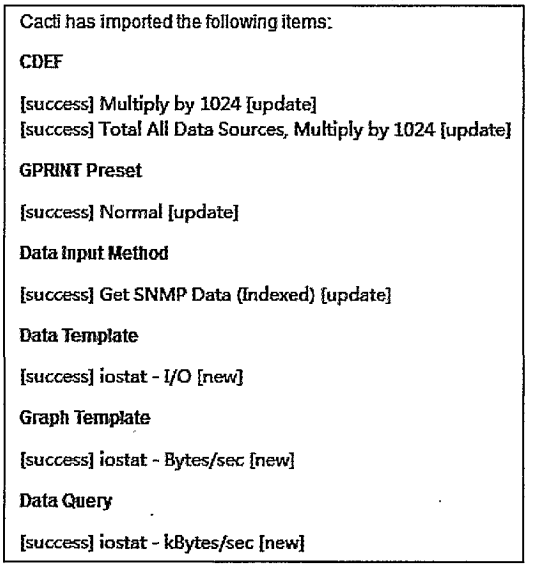


图 21-26　Cacti-iostat 模板导入成功

（3）被监控端配置

1）配置定时任务。将 script 目录下的 iostat.pl 放到被监控服务器的 /bin 目录下，配置可执行权限，并编写一个 cacti_iostat.sh 脚本，用于 crontab 的执行。

```
cat  /root/cacti_iostat.sh
#!/bin/sh
IOSTAT=/usr/bin/iostat
cd /tmp && $IOSTAT -xkd 15 4 -N |sed 2d | sed -n /^$/,/^$/p > io.tmp && mv
-f io.tmp iostat.cache
```

然后修改 crontab，让脚本一分钟跑一次。

```
crontab -l
*/1* * * * /root/cacti_iostat.sh
```

2）配置 SNMP。下面是经常使用的一个脚本，输入 SNMP 账号、密码就可以将被监控服务器的 SNMPv3 配置起来。

```
cat /root/cacti_tune.sh
YUM=/usr/bin/yum
SNMPWALK=/usr/bin/snmpwalk
#get_options 函数，得到用户输入参数
get_options() {
        # Note that we use `"$@"' to let each command-line parameter expand to a
        # separate word. The quotes around `$@' are essential!
        # We need TEMP as the `eval set --' would nuke the return value of getopt.
        TEMP=`getopt --options hvU:P: \
                              --long help,version,username:,password: \
```

1107114476@qq.com
dolphinebook.com

```
                                -- "$@"`

        if [ $? != 0 ] ; then echo "Terminating..." >&2 ; exit 1 ; fi

        # Note the quotes around `$TEMP': they are essential!
        eval set -- "$TEMP"

        while true ; do
                case "$1" in
                        -h|--help) help ; exit ;;
                        -v|--version) print_version ; exit ;;
                        -U|--username) USERNAME=$2 ; shift 2 ;;
                        -P|--password) PASSWORD=$2 ; shift 2 ;;
                        --) shift ; break ;;
                        *) echo "Internal error!" ; exit 1 ;;
                esac
        done
}

#check_options函数，检查用户是否输入参数，如果没有，则提示用户输入参数
check_options() {
        while [ -z "$USERNAME" ]; do
                printf "Please input %s: " "Snmpv3 username"
                read USERNAME
       done

        while [ -z "$PASSWORD" ]; do
                printf "Please input %s: " "Snmpv3 password"
                read PASSWORD
       done
}
#检查是否是root用户
run_check() {

  # check for user ID - has to be root
  if [ $((UID)) != 0 ]; then
    echo 'You need to run this script as ROOT user' >&2
    exit 2
  fi
#检查是否安装了需要的SNMP组件，如果没有则通过yum命令安装
  # check the packages
  for pkg in sysstat.x86_64 net-snmp.x86_64 net-snmp-utils.x86_64 net-snmp-devel.x86_64
  do
    $YUM list installed|grep $pkg >/dev/null 2>&1
    if [ $? != 0 ];then
      echo "$pkg does not existed"
      $YUM -y install $pkg >/dev/null 2>&1
      if [ $? = 0 ];then
        echo "$pkg install successful..."
      fi
```

1107114476@qq.com
dolphinebook.com

```
    fi
  done

}
# 配置 snmp v3
cacti_tune()
{

/etc/init.d/snmpd stop >/dev/null
/usr/bin/net-snmp-config --create-snmpv3-user -ro -a $PASSWORD -A MD5 $USERNAME >/dev/null
/etc/init.d/snmpd start >/dev/null
# 测试配置是否生效
if $SNMPWALK -v3 -u $USERNAME -l auth -a MD5 -A $PASSWORD 127.0.0.1 if >/dev/null;then
  echo 'Snmpwalk OK...
Snmpv3 setup done...'
else
  echo 'Snmpwalk error, you need to check it manul !!'
  exit 2
fi
# 配置磁盘性能脚本
if ! grep "pass .1.3.6.1.3.1 /usr/bin/perl /bin/iostat.pl" /etc/snmp/snmpd.conf
>/dev/null 2>&1;then
  echo "pass .1.3.6.1.3.1 /usr/bin/perl /opt/scripts/iostat.pl" >>/etc/snmp/snmpd.conf
fi

/etc/init.d/snmpd restart >/dev/null
/sbin/chkconfig snmpd on >/dev/null
echo "Cacti IO monitoring setup done... "
}

  run_check
  get_options "$@"
  check_options
  cacti_tune
```

3）测试。在 Cacti 服务器执行 snmpwalk 命令，探测配置是否成功。

```
snmpwalk -v3 -l authPriv -u test -a MD5 -A password -x DES -X password  ip
```

连接成功后会输出非常多的信息。接下来，在 Cacti 中添加相应的监控，单击相应服务器的 status，用鼠标拉到页面后面，在“AssociatedData Queries”选项中单击“Add Data Query:”项，添加相应的“query”，如 iostat- Utilisation，然后单击“VerboseQuery”项，在新的页面中可以看到类似下面的信息：

```
Running    data query [12].
+ Found type = '3' [snmp query].
+ Found data      query XML file at '/var/www/html/cacti/resource/snmp_queries
/iostat.xml'
+ XML fileparsed ok.
+ Executing SNMP walk for list of indexes @     '.1.3.6.1.3.1'
```

1107114476@qq.com
dolphinebook.com

```
+ Index found at OID: '1.3.6.1.3.1.1.1' value: '1'
+ Index found at OID: '1.3.6.1.3.1.1.2' value: '2'
+ Index   found at OID: '1.3.6.1.3.1.1.3' value: '3'
+ Index found at OID:    '1.3.6.1.3.1.1.4' value: '4'
+ Index found at OID:    '1.3.6.1.3.1.1.5' value: '5'
+ Index found at OID:    '1.3.6.1.3.1.2.1' value: 'sda'
+ Index found at OID:    '1.3.6.1.3.1.2.2' value: 'sda1'
.................
```

接着，拉到页面最上方，选择 CreateGraphs for this Host，选择相应的磁盘，单击“create”按钮即可。过一段时间，就可以看到图形出现了。如果没有图形出现，可以检查以下两个问题。

- SELinux 是否关闭。
- 将延时设置更大，有时候查询没有数据，将延时从 500ms 设置成 1000ms 或者更高便可解决。

2. Cacti_cpu_full 模板

宿主机 CPU 的 Cacti 监控，可以使用如下链接的 Cacti_cpu_full 模板：

```
http:// forums.cacti.net/viewtopic.php?f=12&t=42227
http:// download.csdn.net/detail/zhang0506/6318683
```

效果如图 21-27 所示。

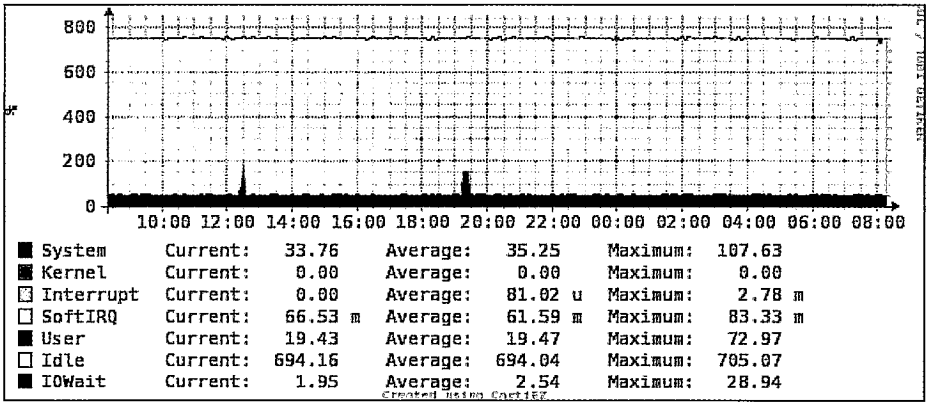


图 21-27 Cacti CPU 监控模板效果图

3. Cacti_memmory_full 模板

宿主机内存的 Cacti 监控，可以使用如下链接的 Cacti_memmory_full 模板：

```
http:// forums.cacti.net/viewtopic.php?t=24501
```

效果如图 21-28 所示。

1107114476@qq.com
dolphinebook.com

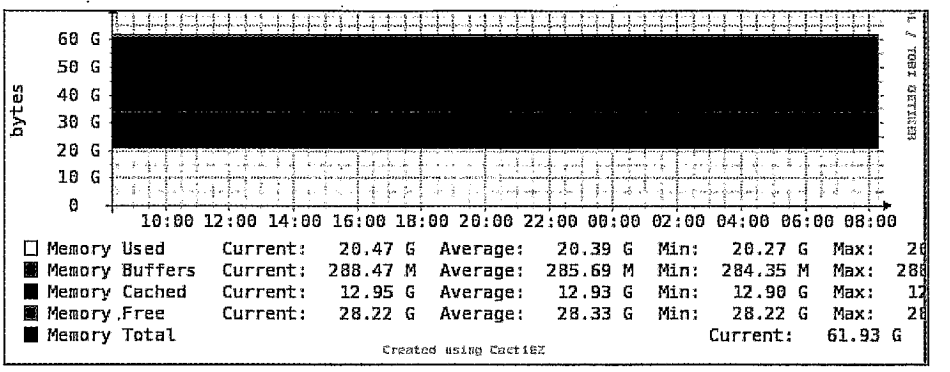


图 21-28　Cacti 内存监控模板效果图

CPU、内存模板的配置方法很简单，将模板导入 Cacti 服务器，在配置被监控机的时候选择模板就可以了。

21.2.2　用 Zabbix 监控宿主机

在众多的监控软件中，Zabbix 各项功能齐全，在被监控服务器上安装有客户端，定制非常方便，并且是开源软件，是目前国内使用最广泛的监控软件之一。

本节主要讲解通过 Zabbix 监控宿主机的以下指标：CPU 负荷，内存使用，硬盘吞吐和利用率，以及网络状况。

1. CPU 负荷监控

由于宿主机上运行了许多虚拟机，而虚拟机则被分配了一定数量的 CPU，所以对于宿主机的监控，从如下几方面入手：

（1）CPU 的负载情况

通常可以通过命令行：top、uptime、w 等来进行查看负载，所谓 CPU 负载指的是一段时间内任务队列的长度，也就是一段时间内一共有多少任务在使用或等待使用 CPU。

例如：

```
[root@localhost]# w
 21:03:05 up 187 days,  9:21,  1 user,  load average: 1.90, 0.99, 0.60
```

1.90、0.99、0.60 分别代表前 1 分钟，5 分钟，15 分钟的平均 CPU 负载，最重要的指标是最后一个数字，即前 15 分钟的平均 CPU 负载，这个数字越小越好。对于负载可以直接使用 Zabbix 自带的监控项 system.cpu.load[<cpu>，<mode>] 进行监控。其中：

- cpu——可用值：1.all（默认），2.percpu。
- mode——可用值：1.avg1（1 分钟默认值），2.avg5（5 分钟平均），3.avg15（15 分钟平均值）。
- 范例 key——system.cpu.load[，avg1]。

1107114476@qq.com
dolphinebook.com

在模板 Template OS Linux 里添加监控项，这里是对前 1 分钟、5 分钟、15 分钟的平均 CPU 负载进行采集和展示。如图 21-29 所示，添加前 1 分钟的负载值：

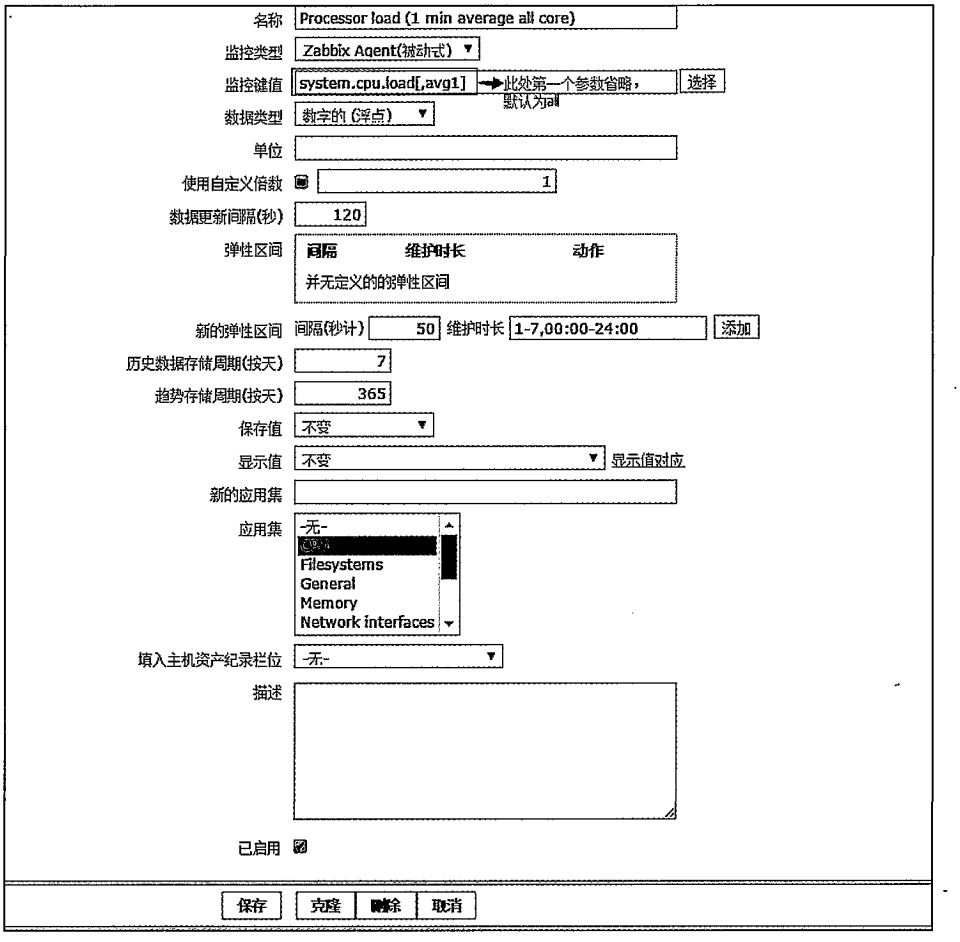


图 21-29 在模板 Template OS Linux 里添加监控项

其他两个依样填写好，即 system.cpu.load[,avg5] 和 system.cpu.load[,avg15]。然后需要通过图形进行展示，Graphs 是将数据展示为图像，给人以视觉化效果，Graphs 的配置存在于主机和模板中，在刚刚添加监控项的模板中，选择“监控图”，如图 21-30 所示：

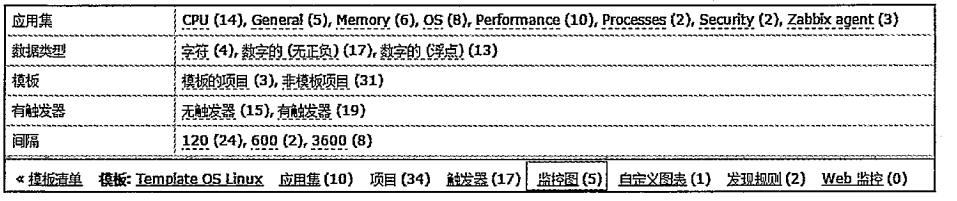


图 21-30 选择监控图

1107114476@qq.com
dolphinebook.com

然后单击创建图形，如图 21-31 所示，输入图形的名称，选择所需要的项（items）。

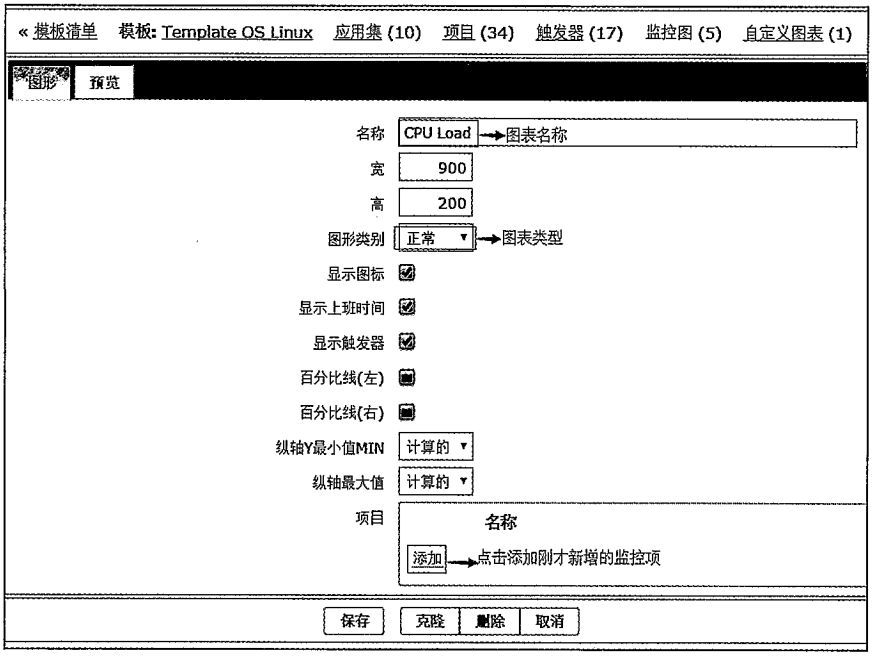


图 21-31　添加监控图

点击保存后，等待一个监控周期（interval），便可以单击“Dashboard”→“监控图”命令，然后选择刚刚添加的监控图 CPU Load 查看效果，如图 21-32 所示：

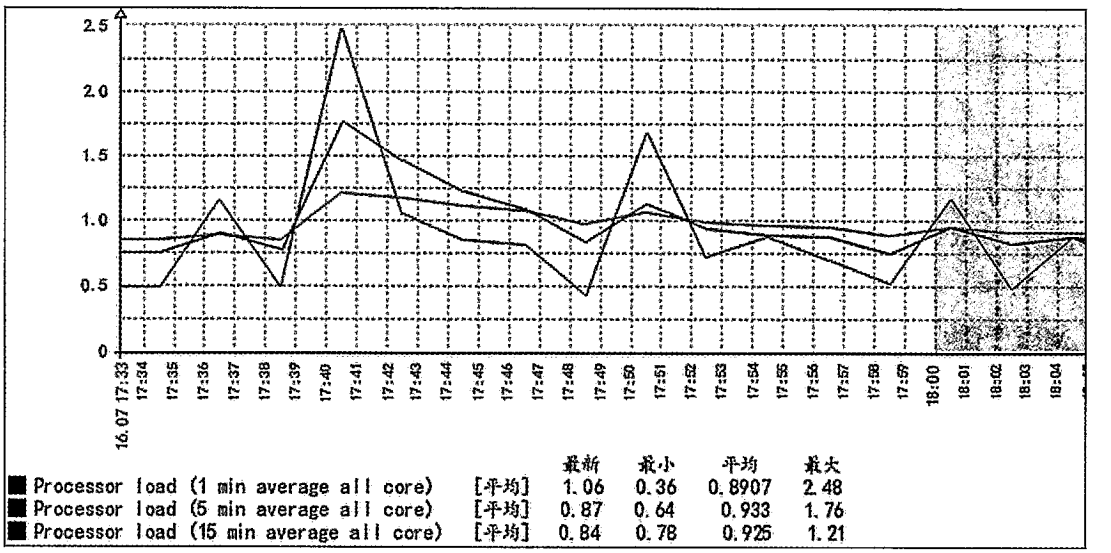


图 21-32　CPU 监控效果

1107114476@qq.com
dolphinebook.com

然后就可以添加报警阈值了，由于负载和 CPU 的数量有关，例如，24 核心的 CPU 和 8 核心的 CPU 所能够承受的负载程度肯定不一样，且线上服务器的 CPU 核数也多种多样，所以这里的阈值按照“当前负载 /CPU 个数”的比例来设定，当然，这个可以根据你线上服务器的实际情况来定。例如，我们设定：当本机最近 5 分钟的负载是 CPU 数量的 2 倍时就发出报警通知，触发器可以这样设置：

```
{Template OS Linux:system.cpu.load[,avg1].last(5m)}/{Template OS Linux:system.
cpu.num.last(0)}>2
```

（2）CPU 的使用率

同查看 CPU 的负载一样，经常使用命令 sar、top 等进行服务器的 CPU 使用情况进行查看。熟悉的人肯定知道，在一个操作系统中，处理器需要被分给系统内核、用户进程以及 I/O 等很多角色使用。

在 Zabbix 中，可以直接使用系统自带的监控项对宿主机的 CPU 整体使用情况进行监控，具体监控项目为：

- ❑ system.cpu.util[,user]
- ❑ system.cpu.util[,nice]
- ❑ system.cpu.util[,system]
- ❑ system.cpu.util[,idle]
- ❑ system.cpu.util[,iowait]
- ❑ system.cpu.util[,steal]
- ❑ system.cpu.util[,interrupt]
- ❑ system.cpu.util[,softirq]

监控的添加和上面添加负载的“监控项”类似，完成后，如图为 21-33 所示：

除了对整体情况进行监控外，还需要对每个核的使用情况进行监控，以便及时调整方式对 cpu 的灵活利用，例如，可以使用处理器的亲和性，来人工指定物理 CPU 和虚拟 CPU 的绑定，以达到合理利 CPU 资源的目标，手工绑定的时候就需要看长期的 CPU 压力监控数据。对 CPU 每个核的使用情况，我们可以通过每个 CPU 的某些指标来进行查看，如 idle 值。我们这里对每个核的 idle 情况进行了监控，也是使用 Zabbix 自带监控项“ system.cpu.util[0,idle,avg1]”，监控项中的 0 代表 cpu0，idle 表示为空闲状态，avg1 是 1 分钟，当然，这里的参数也可以根据需要的情况来定，比如比较关注 5 分钟或者 15 分钟的 CPU 使用情况，avg1 改成 avg5 或者 avg15 即可，在模板中添加完监控项后，查看效果，如图 21-34 所示：

从图 21-34 上可以清晰地看到，最近这段时间，第 29 ~ 32 颗 CPU 的利用率比较高，平均使用率在 60% 左右，实际上这是因为做了 CPU 绑定，将业务压力比较大的那个虚拟机绑定在了最后面几个 CPU 上。

1107114476@qq.com
dolphinebook.com

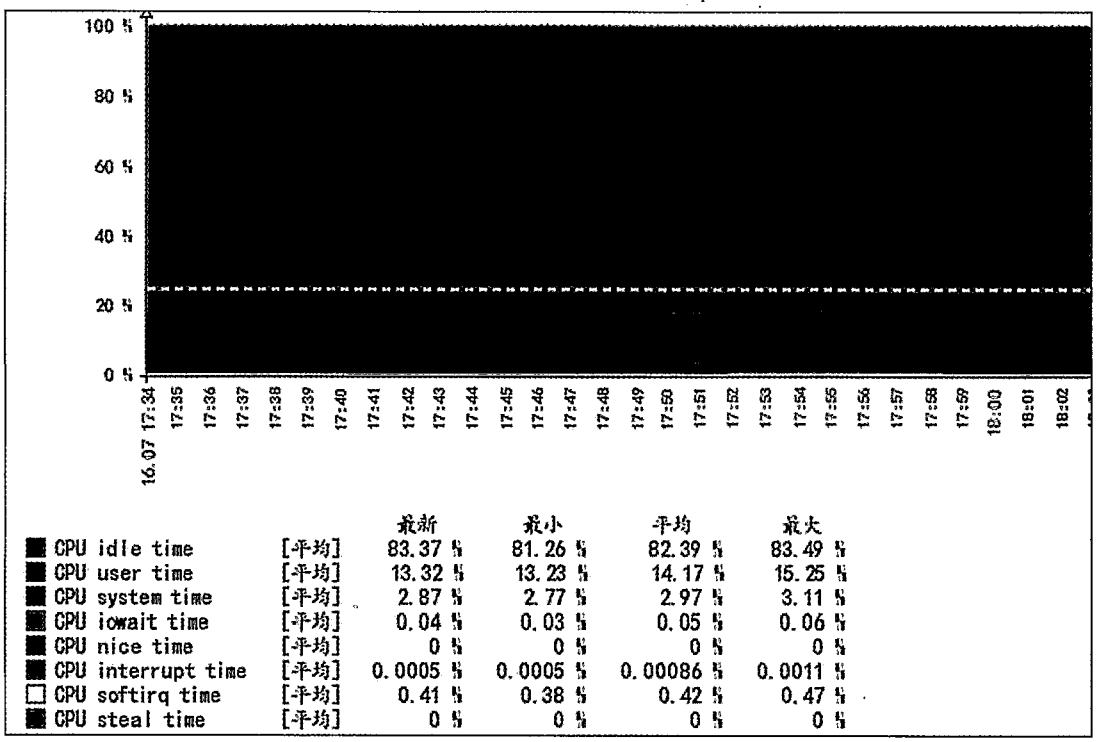


图 21-33　CPU 监控项

（3）CPU 的 jumps

CPU 的 jumps 主要是监控设备中断数和上下文交换次数。现在 Linux 大多基于抢占式，CPU 给每个任务一定的服务时间，当时间片轮转的时候，需要把当前状态保存下来，同时加载下一个任务，这个过程叫做上下文切换。时间片轮转的方式，使得多个任务利用一个 CPU 执行成为可能，但是保存现场和加载现场，也带来了性能消耗。

那线程上下文切换的次数和时间以及性能消耗如何看呢？ Zabbix 自带的监控项为：

- system.cpu.intr；
- System.cpu.switches。

读者可以自己添加，添加完成后效果如图 21-35 所示。

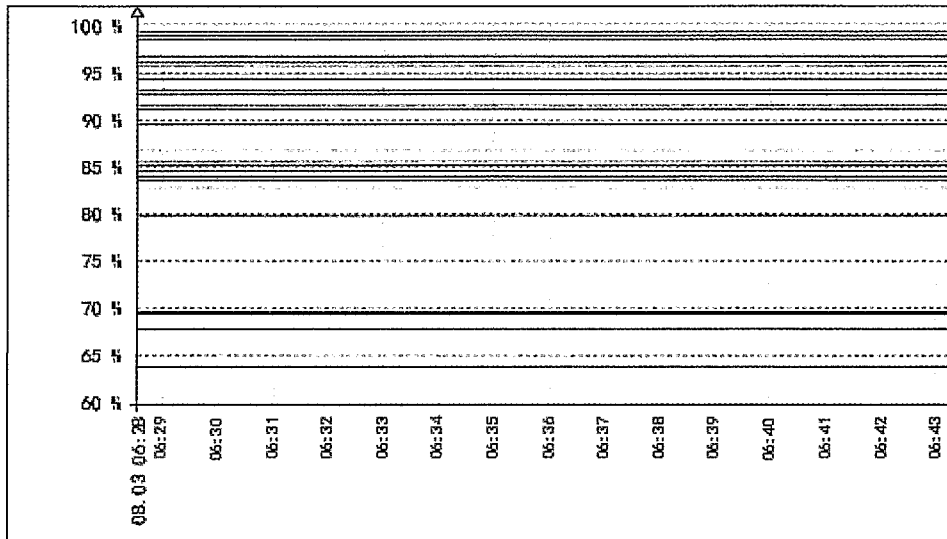
（4）CPU 主频监控

Intel 的 E5 系列 CPU 有自动调频的功能，称为睿频技术。它可以根据系统的压力，自己调整工作频率，主要是为了省电。通过 Zabbix 也可以将 CPU 的频率监控起来，效果如图 21-35 所示：

2. 内存使用监控

宿主机的内存使用主要从物理内存和虚拟内存两方面进行监控。物理内存监控的项目有总物理内存、可用物理内存、可用物理内存百分比；Swap 内存监控项目有总 Swap 内存、可用 Swap 内存、可用 Swap 内存百分比，使用 Zabbix 自带的监控项即可满足需求，如图 21-36、图 21-37 所示：

1107114476@qq.com
dolphinebook.com



		最新	最小	平均	最大
Cpu_idle_1	[平均]	67.8 %	56.99 %	67.8 %	87.56 %
Cpu_idle_2	[平均]	79.8 %	70.33 %	79.8 %	91.39 %
Cpu_idle_3	[平均]	83.92 %	71.12 %	83.92 %	92.3 %
Cpu_idle_4	[平均]	84.68 %	68.62 %	84.68 %	96.81 %
Cpu_idle_5	[平均]	85.68 %	73.36 %	85.68 %	96.72 %
Cpu_idle_6	[平均]	86.85 %	64.42 %	86.85 %	98.99 %
Cpu_idle_7	[平均]	82.86 %	59 %	82.86 %	98.92 %
Cpu_idle_8	[平均]	85.22 %	65.91 %	85.22 %	99.58 %
Cpu_idle_9	[平均]	83.56 %	52.85 %	83.56 %	94.37 %
Cpu_idle_10	[平均]	89.64 %	76.95 %	89.64 %	98.48 %
Cpu_idle_11	[平均]	91.12 %	66.24 %	91.12 %	99.11 %
Cpu_idle_12	[平均]	91.6 %	62.35 %	91.6 %	99.83 %
Cpu_idle_13	[平均]	98.63 %	92.22 %	98.63 %	99.88 %
Cpu_idle_14	[平均]	99.37 %	95.8 %	99.37 %	100 %
Cpu_idle_15	[平均]	99.04 %	91.49 %	99.04 %	100 %
Cpu_idle_16	[平均]	98.72 %	84.16 %	98.72 %	100 %
Cpu_idle_17	[平均]	92.72 %	86.35 %	92.72 %	95.63 %
Cpu_idle_18	[平均]	93.3 %	84.16 %	93.3 %	96.7 %
Cpu_idle_19	[平均]	93.59 %	87.02 %	93.59 %	97.62 %
Cpu_idle_20	[平均]	94.74 %	86.73 %	94.74 %	98.92 %
Cpu_idle_21	[平均]	91.56 %	71.67 %	91.56 %	99.2 %
Cpu_idle_22	[平均]	94.48 %	68.82 %	94.48 %	99.48 %
Cpu_idle_23	[平均]	91.55 %	50.85 %	91.55 %	99.98 %
Cpu_idle_24	[平均]	96.12 %	78.06 %	96.12 %	99.97 %
Cpu_idle_25	[平均]	96.82 %	92.81 %	96.82 %	99.21 %
Cpu_idle_26	[平均]	95.81 %	82.59 %	95.81 %	99.55 %
Cpu_idle_27	[平均]	96.94 %	84.9 %	96.94 %	99.66 %
Cpu_idle_28	[平均]	97.1 %	86.27 %	97.1 %	99.82 %
Cpu_idle_29	[平均]	63.9 %	51.78 %	63.9 %	78.84 %
Cpu_idle_30	[平均]	69.49 %	61.6 %	69.49 %	78.79 %
Cpu_idle_31	[平均]	69.47 %	59.39 %	69.47 %	88.96 %
Cpu_idle_32	[平均]	69.64 %	63.61 %	69.64 %	79.8 %

图 21-34 CPU 利用率监控图

1107114476@qq.com
dolphinebook.com

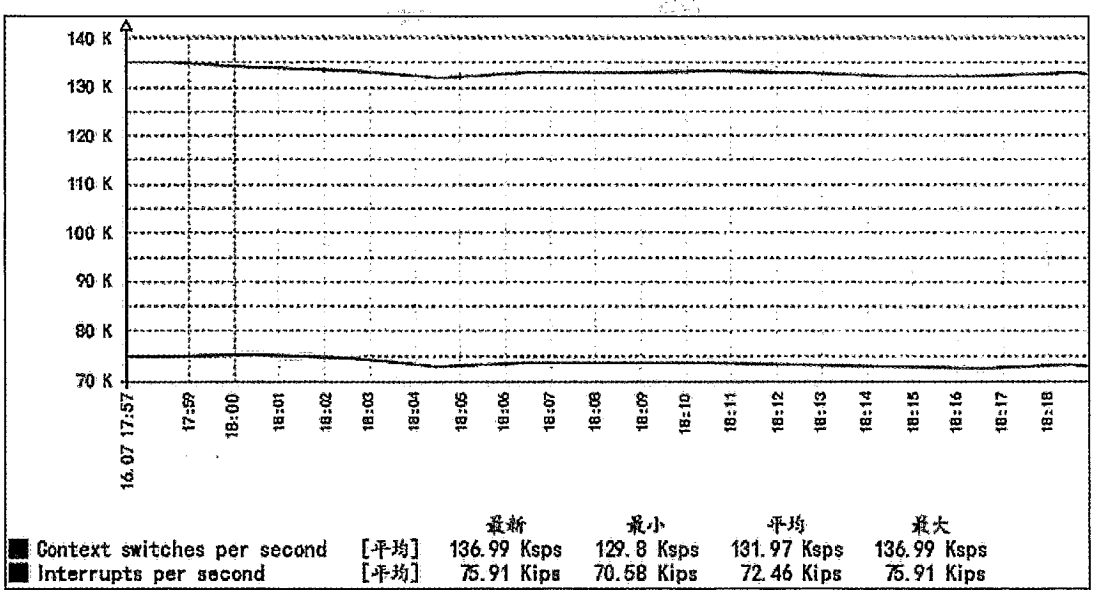


图 21-35　CPU jumps

3. 硬盘吞吐和利用率

默认的 Zabbix 监控模板中，对于磁盘主要是对空间使用情况的监控，对于 I/O 方面却没有自带的“监控项”，并且每台服务器的磁盘名称可能不一样（如 vda、sda、xvdb、cciss/c0d0 等），所以需要用到 Zabbix 的神技功能之一的 LLD 来发现磁盘并监控 I/O，它可以做到自适应不同的磁盘名称监控，其原理是通过抓取系统文件 /proc/diskstats 中的内容进行展示。例如有以下内容。

（1）Disk - IO By sda

如图 21-36 所示：

第一个监控项 Disk:sda:IO:Currently:Active 表示 I/O 的当前进度，只有这个域应该是 0。当请求被交给适当的 request_queue_t 时 I/O 增加，请求完成时 I/O 减小。

第二个监控项 Disk:sda:IO（ms）表示花在 I/O 操作上的毫秒数。

（2）Disk - Bytes/second By sda

这 2 个监控项主要用于监控磁盘的吞吐，每秒读写速率。

（3）Disk - Read/Write ms By sda

如图 21-41，监控磁盘 sda 读写花费的毫秒数。

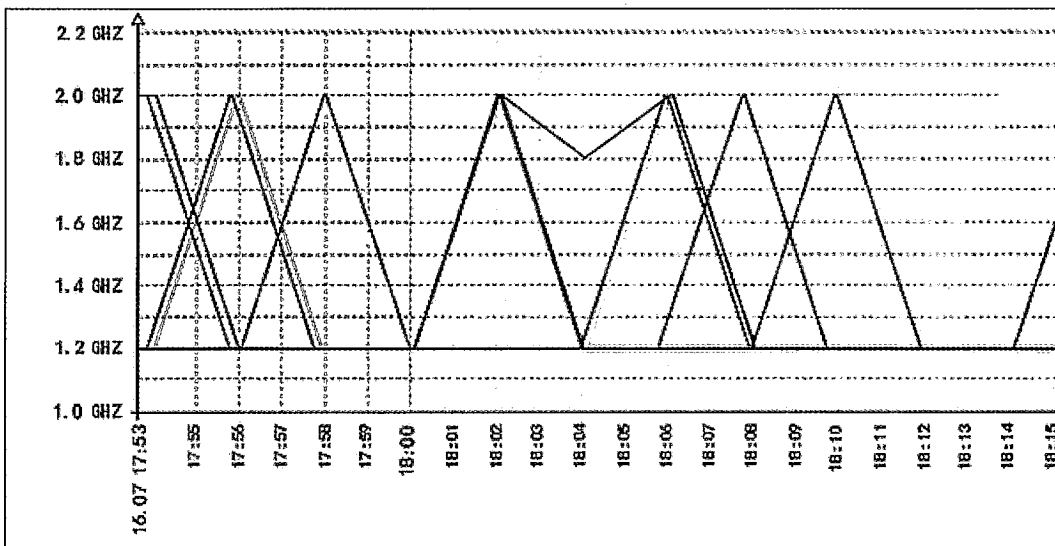
（4）Disk - iops/second By sda

如图 21-42，监控磁盘 sda 的读写 IOPS：

磁盘监控具体如何实现呢？通过如下几步：

1）收集硬盘信息。

1107114476@qq.com
dolphinebook.com



		最新	最小	平均	最大
Cpu_crufreq_1	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_2	[平均]	1.2 GHZ	1.2 GHZ	1.23 GHZ	2 GHZ
Cpu_crufreq_3	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_4	[平均]	1.26 GHZ	1.2 GHZ	1.28 GHZ	2 GHZ
Cpu_crufreq_5	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_6	[平均]	1.2 GHZ	1.2 GHZ	1.25 GHZ	2 GHZ
Cpu_crufreq_7	[平均]	2 GHZ	1.2 GHZ	1.25 GHZ	2 GHZ
Cpu_crufreq_8	[平均]	1.2 GHZ	1.2 GHZ	1.23 GHZ	2 GHZ
Cpu_crufreq_9	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_10	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_11	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_12	[平均]	1.2 GHZ	1.2 GHZ	1.23 GHZ	2 GHZ
Cpu_crufreq_13	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_14	[平均]	1.2 GHZ	1.2 GHZ	1.23 GHZ	2 GHZ
Cpu_crufreq_15	[平均]	1.2 GHZ	1.2 GHZ	1.31 GHZ	2 GHZ
Cpu_crufreq_16	[平均]	1.2 GHZ	1.2 GHZ	1.36 GHZ	2 GHZ
Cpu_crufreq_17	[平均]	1.2 GHZ	1.2 GHZ	1.23 GHZ	2 GHZ
Cpu_crufreq_18	[平均]	1.2 GHZ	1.2 GHZ	1.23 GHZ	2 GHZ
Cpu_crufreq_19	[平均]	1.2 GHZ	1.2 GHZ	1.23 GHZ	2 GHZ
Cpu_crufreq_20	[平均]	1.2 GHZ	1.2 GHZ	1.23 GHZ	2 GHZ
Cpu_crufreq_21	[平均]	1.2 GHZ	1.2 GHZ	1.23 GHZ	2 GHZ
Cpu_crufreq_22	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_23	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_24	[平均]	1.2 GHZ	1.2 GHZ	1.2 GHZ	1.2 GHZ
Cpu_crufreq_25	[平均]	1.2 GHZ	1.2 GHZ	1.28 GHZ	2 GHZ
Cpu_crufreq_26	[平均]	1.2 GHZ	1.2 GHZ	1.28 GHZ	2 GHZ
Cpu_crufreq_27	[平均]	1.2 GHZ	1.2 GHZ	1.31 GHZ	2 GHZ
Cpu_crufreq_28	[平均]	1.2 GHZ	1.2 GHZ	1.31 GHZ	2 GHZ
Cpu_crufreq_29	[平均]	1.2 GHZ	1.2 GHZ	1.31 GHZ	2 GHZ
Cpu_crufreq_30	[平均]	1.2 GHZ	1.2 GHZ	1.31 GHZ	2 GHZ
Cpu_crufreq_31	[平均]	2 GHZ	1.2 GHZ	1.31 GHZ	2 GHZ
Cpu_crufreq_32	[平均]	2 GHZ	1.2 GHZ	1.38 GHZ	2 GHZ

图 21-36 CPU 频率监控

1107114476@qq.com
dolphinebook.com

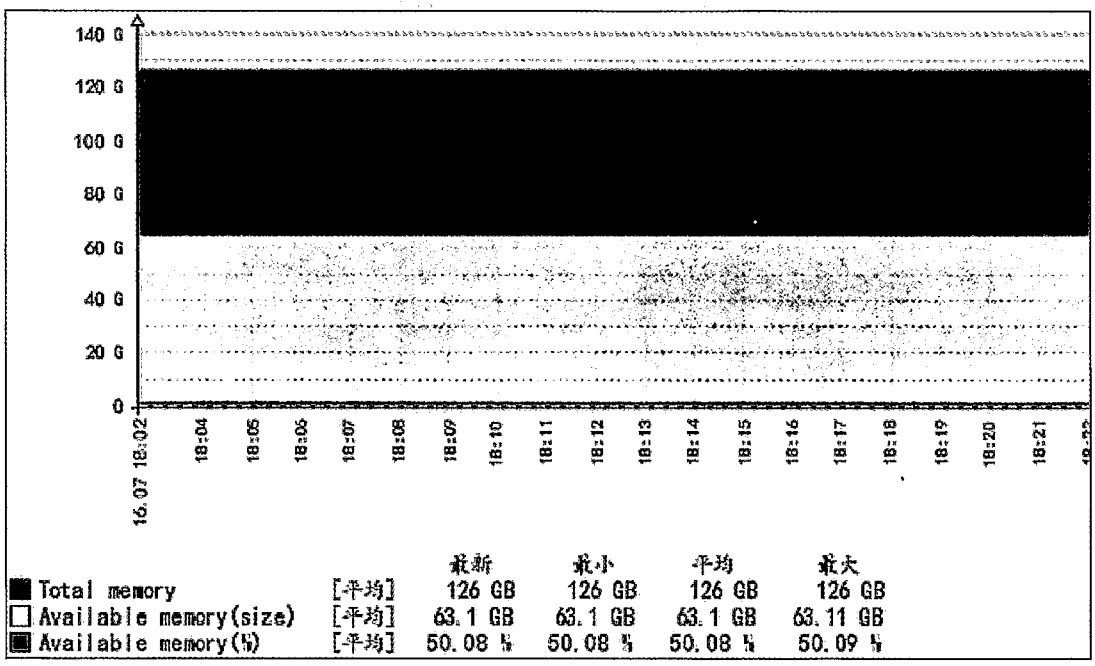


图 21-37　Zabbix 内存监控

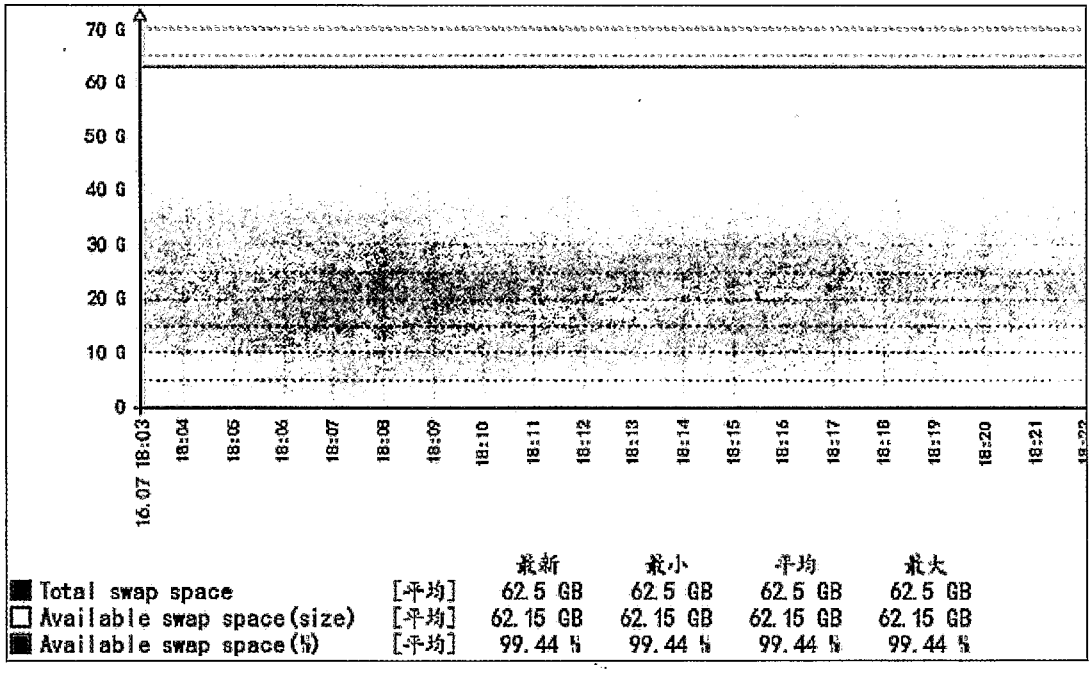


图 21-38　Zabbix swap 监控

1107114476@qq.com
dolphinebook.com

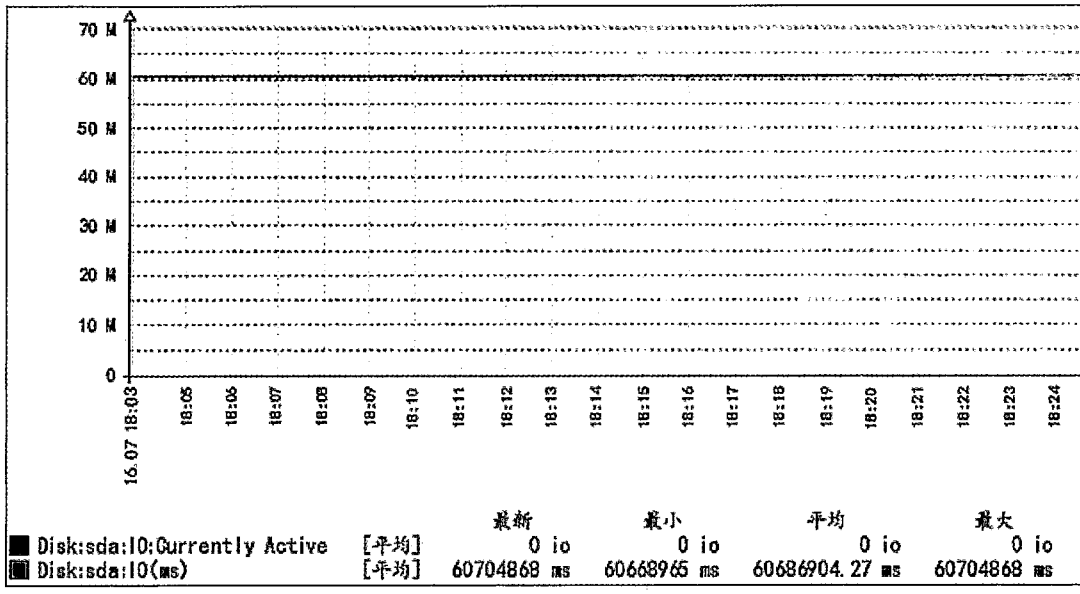


图 21-39 Zabbix 磁盘 I/O 监控

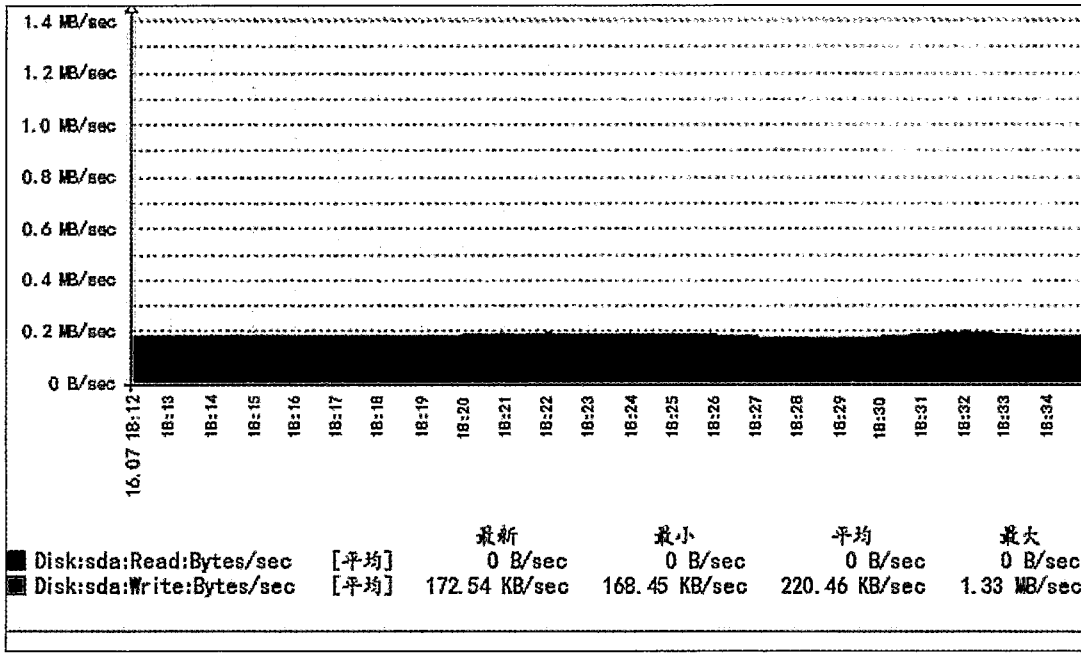


图 21-40 Zabbix 监控磁盘的读写字节数

1107114476@qq.com
dolphinebook.com

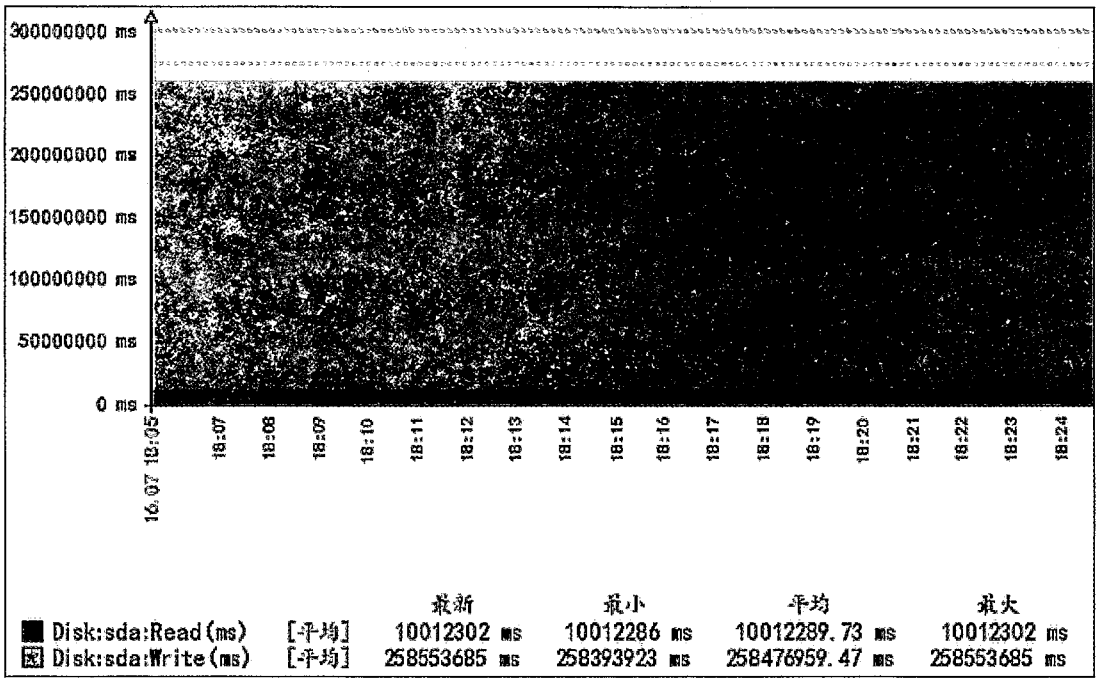


图 21-41　Zabbix 监控磁盘 sda 读写时间

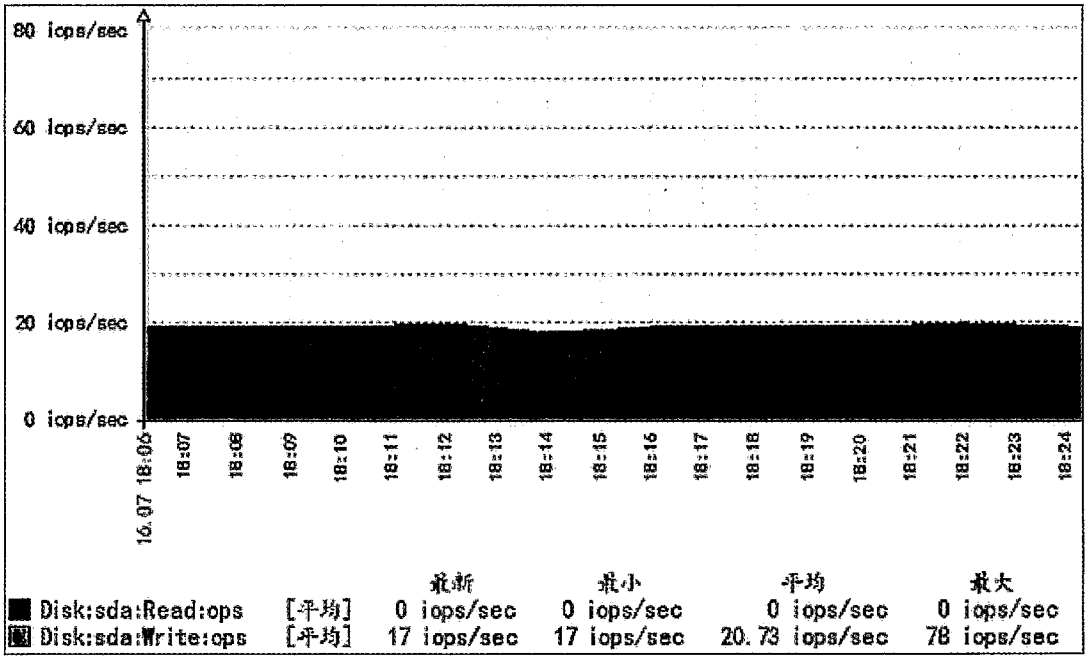


图 21-42　Zabbix 监控磁盘 sda 的读写 IOPS

1107114476@qq.com
dolphinebook.com

通过脚本收集到agent端的硬盘信息，然后把收集到的内容用JSON格式展示，JSON格式是Zabbix接受的格式，然后服务端通过正则表达式过滤出结果。脚本内容如下：

```
cat partition_low_discovery.sh
#!/bin/bash
diskarray=(`cat /proc/diskstats |grep -E "\bvd[abcdefg]\b|\bsd[abcdefg]\b|
\bc0d0p[0~9]\b"|grep -i "\b$1\b"|awk '{print $3}'|sort|uniq   2>/dev/null`)
# 本脚本核心，用于获取本机器的磁盘
length=${#diskarray[@]}
printf "{\n"
printf  '\t'"\"data\":["
for ((i=0;i<$length;i++))
do
        printf '\n\t\t{'
        printf "\"{#DISK}\":\"${diskarray[$i]}\"}"# 将获取的磁盘进行展示
        if [ $i -lt $[$length-1] ];then
                printf ','
        fi
done
printf  "\n\t]\n"
printf "}\n"
```

在宿主机上执行后的结果如下：

```
{
        "data":[
                {"{#DISK}":"sda"},
        ]
}
```

这里是Shell脚本，如果读者熟悉其他脚本，如Python、Perl，也可以使用它们来编写，只要保证结果一致即可。

2）修改zabbix_agentd.conf文件并重启。

磁盘信息收集完成后，要修改zabbix-agentd.conf，需要在agent端进行自定义key添加，具体如下：

```
#add disk discovery
UserParameter=custom.vfs.dev.discovery,/bin/sh /opt/zabbix/partition_low_discovery.sh
# reads completed successfully
UserParameter=custom.vfs.dev.read.ops[*],cat /proc/diskstats | egrep $1 | head -1 |
awk '{print $$4}'
# sectors read
UserParameter=custom.vfs.dev.read.sectors[*],cat /proc/diskstats | egrep $1 | head -1 |
awk '{print $$6}'
# time spent reading (ms)
UserParameter=custom.vfs.dev.read.ms[*],cat /proc/diskstats | egrep $1 | head -1 |
awk '{print $$7}'
# writes completed
UserParameter=custom.vfs.dev.write.ops[*],cat /proc/diskstats | egrep $1 | head -1 |
awk '{print $$8}'
```

1107114476@qq.com
dolphinebook.com

```
# sectors written
UserParameter=custom.vfs.dev.write.sectors[*],cat /proc/diskstats | egrep $1 | head -1 |
awk '{print $$10}'
# time spent writing (ms)
UserParameter=custom.vfs.dev.write.ms[*],cat /proc/diskstats | egrep $1 | head -1 |
awk '{print $$11}'
# IOs currently in progress
UserParameter=custom.vfs.dev.io.active[*],cat /proc/diskstats | egrep $1 | head -1 |
awk '{print $$12}'
# time spent doing IOs (ms)
UserParameter=custom.vfs.dev.io.ms[*],cat /proc/diskstats | egrep $1 | head -1 |
awk '{print $$13}'
```

注意　我这里的脚本 partition_low_discovery.sh 放在 /opt/zabbix 下面，然后给予此脚本 755 权限，并且将所属组和所属用户改为 zabbix，重启 agent。

3）服务端导入模板，设置正则和添加模板。

在 Zabbix 的 Web 页面上，点击“配置管理”→“模板”，如图 21-43 所示：

图 21-43　导入模板操作

然后点击右上角的“导入”，进行模板导入，如图 21-44 所示：

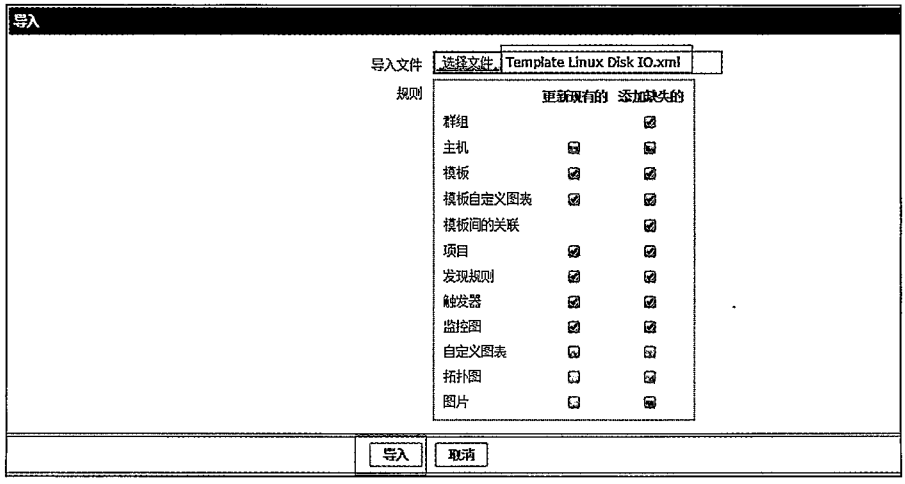


图 21-44　导入模板操作

点击刚刚导入的模板，点击 " 发现规则 "，可以查看具体监控项内容如图 21-45、图 21-46 所示：

1107114476@qq.com
dolphinebook.com

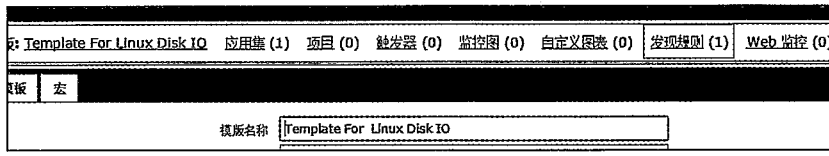


图 21-45 发现规则

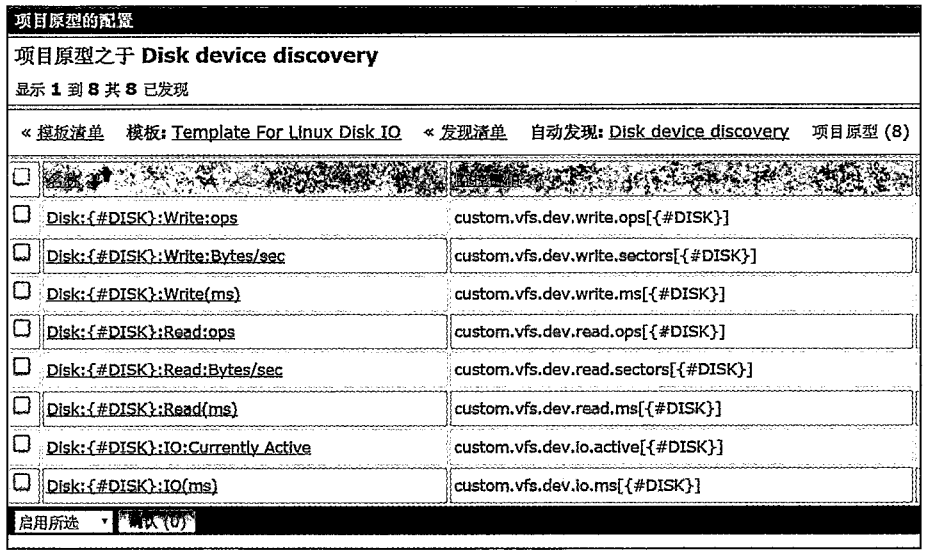


图 21-46 监控项内容

然后设置正则表达式，表示匹配需要的磁盘格式，在 Web 页面上选择“系统管理”→“基础配置”→“正则表达式”，然后创建我们需要匹配的正则格式，如图 21-47 所示：

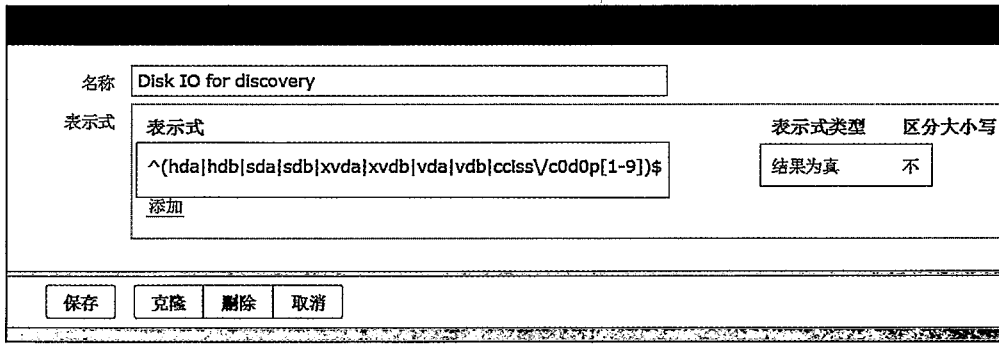


图 21-47 配置正则表达式

至此，准备工作已经做完，直接点击宿主机进行模板的关联即可。然后，单击宿主机的图形就看到了开头展示的图形，每个磁盘的吞吐量和 IOPS 就展示出来了。

1107114476@qq.com
dolphinebook.com

提示　参考链接 http://www.muck.net/19/getting-hard-disk-performance-stats-from-zabbix。

4）网络 I/O 监控。

宿主机的网络 I/O 监控也是件很头痛的事情，因为宿主机里面这么多网卡，到底该监控哪些网卡，不该监控哪些网卡？ Zabbix 也能很好地解决，可以使用 Zabbix 自带的监控项加正则表达式过滤即可满足需求。

首先，Zabbix 可以通过它的另一优秀功能，即自动发现，这一功能和上面的磁盘发现是一样的原理，不过 Zabbix 本身已经提前写好了发现功能模板，只需要在功能模板进行简单定制后即可满足需求。这个功能已经自带在 Template OS Linux 模板中了，如图 21-48 所示。

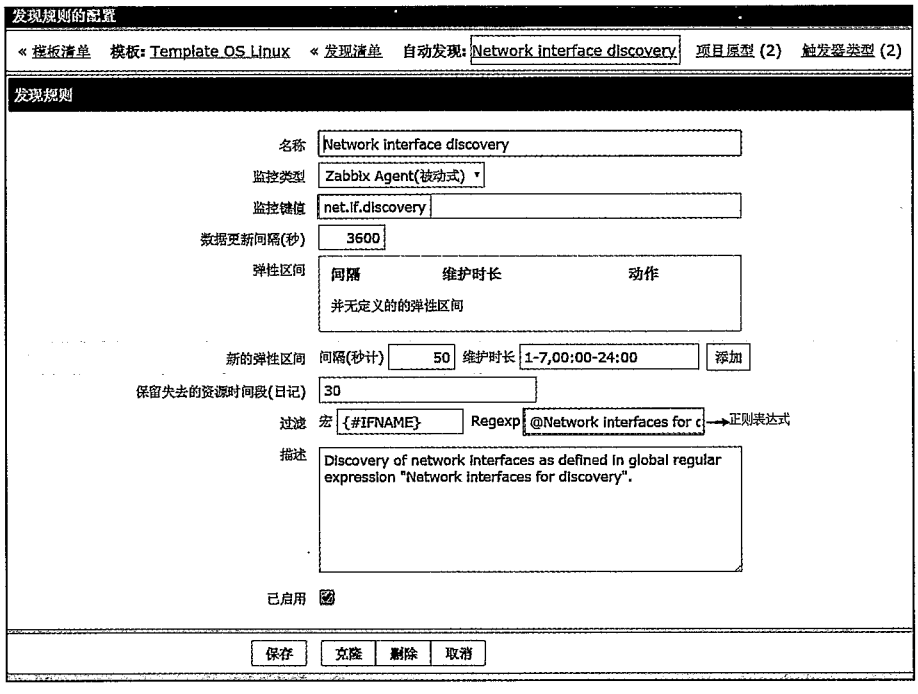


图 21-48　Template OS Linux 中的网卡自动发现

将宿主机应用以上模板后，即可自动发现宿主机上的所有网卡，然后可以看到每块网卡的流量。同样，如果你需要过滤掉某些不需要的网卡，则可以应用图 21-48 的正则表达式选项。依次点击“系统管理”→“基础配置”→“正则表达式”进行配置，如图 21-49 所示，对 lo 端口进行过滤，如果你需要过滤其他不需要的端口，依葫芦画瓢即可。

1107114476@qq.com
dolphinebook.com

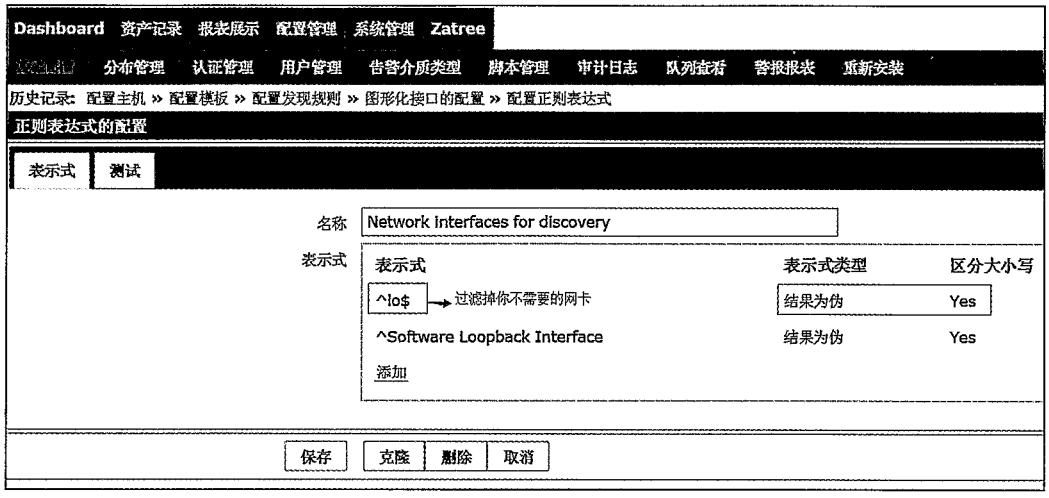


图 21-49 过滤网卡正则表达式

如果需要看到发包数、丢包数、错误包数，都可以使用 Zabbix 自带的监控项来实现。这里只监控了网卡的发包数。监控 key 为："net.if.in[if,<mode>]"、"net.if.out[if,<mode>]"，其中，if 为端口，"mode" 为模式，例如 net.if.in[eth0,packets] 表示 eth0 端口在流入方向的发包数。将监控项添加进模板后，即可获得对应的图形，如图 21-50 和图 21-51 所示：

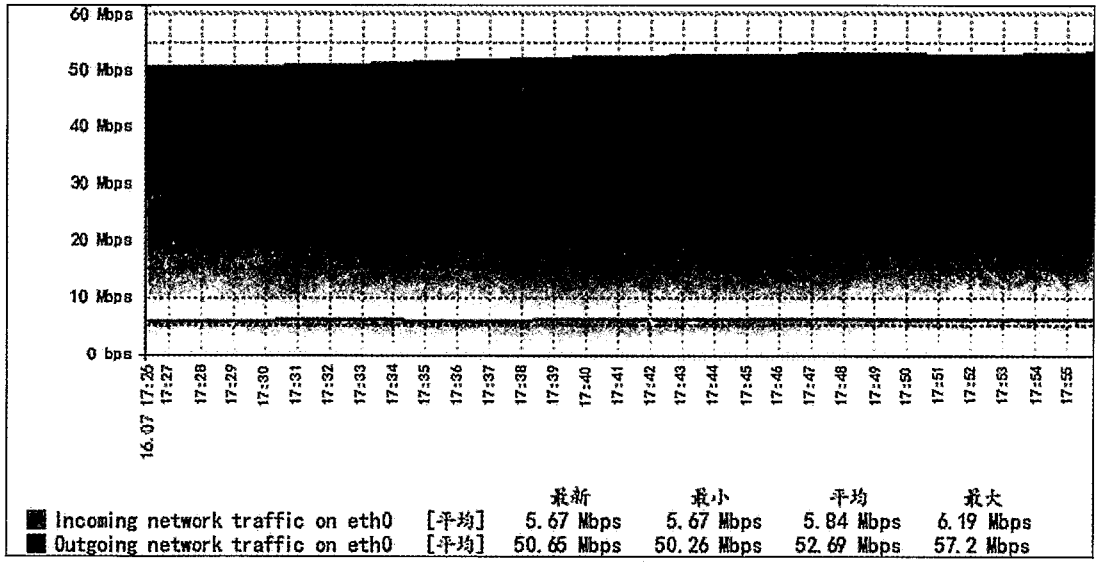


图 21-50 Zabbix 监控网卡吞吐

1107114476@qq.com
dolphinebook.com

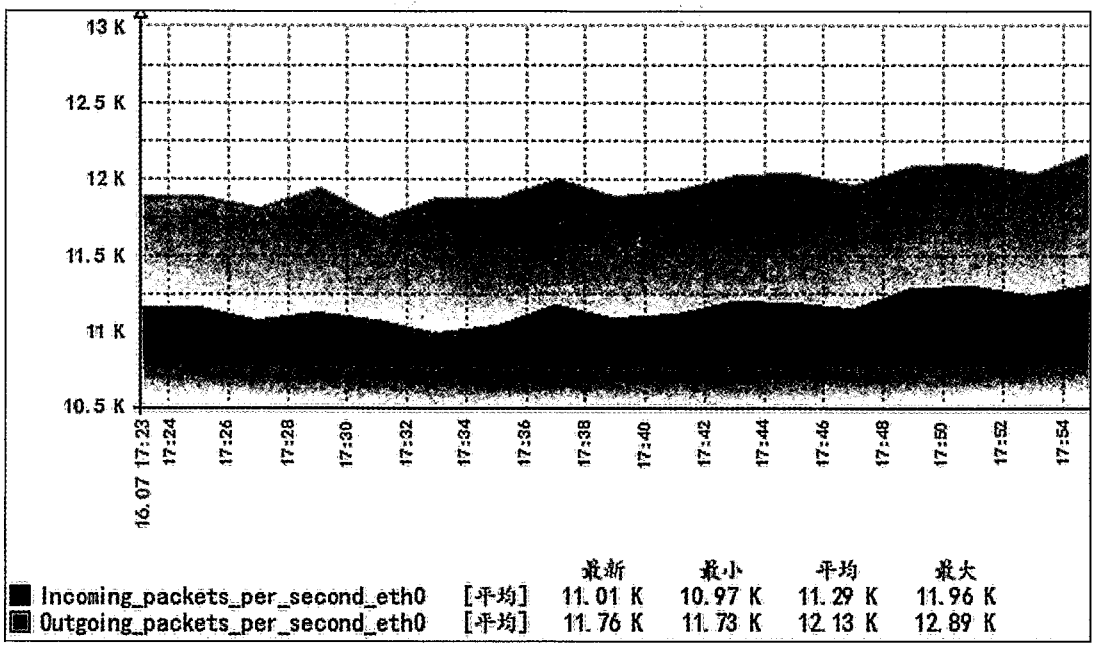


图 21-51 Zabbix 监控网卡发包数

21.3 应急方案

运维的一个重要职责就是应对突发情况，凡事预则立不预则废，比较好的应对突发情况的方法就是制定应急预案，并经常进行演习，当紧急情况发生的时候就可以从容面对。本节介绍 KVM 虚拟机应急方案的制定和演练注意点。

21.3.1 应急方案制定注意要点

虚拟机的灾备有以下两种思路：

- 虚拟化层灾备。
- 应用层灾备。

图 21-52 是虚拟化灾备方案的示意图，虚拟化层灾备就是做虚拟机镜像的多份复制及快照，应用层做灾备就是将虚拟机像物理机一样灾备，原来在物理机上如何灾备，现在在虚拟机用同样的方法进行灾备。一般建议使用应用层灾备的方法，因为在虚拟化层做灾备，要消耗大量的磁盘空间，并且做多份快照，会影响虚拟机的性能，而在应用层做灾备，消耗的资源比较少，而且可以做到精准备份，即只备份必要的部分。如果要做好应用层的灾备，就需要在应用层提升自动化运维的水平，自动化运维的水平越高，越有利于提升灾备和恢复的效率。

1107114476@qq.com
dolphinebook.com

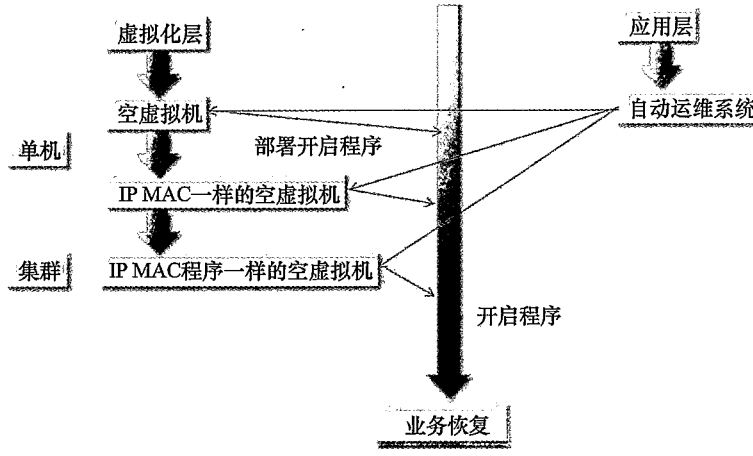


图 21-52 虚拟化灾备方案

虚拟化层灾备有以下一些技术手段：

- 虚拟机镜像多份复制。
- 虚拟机镜像快照。
- 虚拟机使用 LVM 裸设备，然后通过 LVM 快照备份虚拟机。
- 利用 qcow2 镜像的块复制功能进行备份。
- 在线迁移。

下面介绍具体的操作方法：

（1）停机备份，虚拟机镜像多份复制

在业务的维护时间，将虚拟机的镜像复制一份，并复制到其他宿主机上，这种方式操作最简单，非常有效，缺点是只能业务停机时间操作，备份的周期比较长。

（2）通过快照备份虚拟机

在线虚拟机的快照实际有两个层面，即磁盘快照和内存快照。在关机情况下，就只有磁盘快照。如果是关机情况，虚拟机的备份就很简单。下面主要讨论在线虚拟机的备份方法。

虚拟机快照操作命令如下：

```
virsh # snapshot-create-as test
Domain snapshot 1419296749 created
```

但是 CentOS6、CentOS7 中，创建虚拟机快照的时候，虚拟机会暂停几秒，命令如下：

```
virsh  list --all
 Id    Name                           State
----------------------------------------------------
 48    test                           paused
```

快照信息查看，使用 virsh snapshot-list 命令，命令如下：

1107114476@qq.com
dolphinebook.com

```
virsh  snapshot-list test
 Name                 Creation Time             State
------------------------------------------------------------
 1419296749           2014-12-22 20:05:49 -0500 running
 1419296772           2014-12-22 20:06:12 -0500 running
```

也可以通过 qemu-img 命令查看快照信息，命令如下：

```
qemu-img info test
image: test
file format: qcow2
virtual size: 100G (107374182400 bytes)
disk size: 4.0G
cluster_size: 65536
Snapshot list:
ID        TAG                 VM SIZE                DATE       VM CLOCK
1         1419296749             1.8G 2014-12-22 20:05:49   01:07:36.123
2         1419296772             1.8G 2014-12-22 20:06:12   01:07:48.249
Format specific information:
    compat: 1.1
    lazy refcounts: false
```

将快照迁移出来，并复制到其他宿主机上，就可以完成虚拟机的备份，命令如下：

```
qemu-img convert  -f qcow2   -O qcow2   -s s1 test.qcow2 test-s1.qcow2
```

因为快照对虚拟机的性能有影响，备份完成后，删除快照，命令如下：

```
virsh # snapshot-delete test 1419296749
Domain snapshot 1419296749 deleted
virsh # snapshot-delete test 1419296772
Domain snapshot 1419296772 deleted

virsh # snapshot-list test
 Name                 Creation Time             State
------------------------------------------------------------
```

（3）使用 lvm 快照备份虚拟机

如果虚拟机全部使用的是 lvm，可以利用 lvm 的特性，通过 lvm 快照备份虚拟机。

通过 lvcreate 命令创建 lvm 快照，命令如下：

```
lvcreate -s -n  CentOS7s1  -L 10G  /dev/testvg/CentOS7onlvm
  Logical volume "CentOS7s1" created
```

通过 lvs 命令查看快照，命令如下：

```
lvs
LV    VG    Attr    LSize    Pool    Origin    Data%  Move Log Cpy%Sync Convert
CentOS7onlvm testvg    owi-aos--- 200.00g
CentOS7s1        testvg    swi-a-s---  10.00g          CentOS7onlvm   0.06
```

1107114476@qq.com
dolphinebook.com

通过dd命令将lv快照备份到文件中，命令如下：

```
dd if=/dev/testvg/CentOS7s1  of=/home/CentOS7.bak
```

将文件复制到其他宿主机，在恢复的时候，可以直接使用raw格式的文件或者通过dd命令将备份文件复制到备用宿主机上的lvm。

（4）使用qcow2块复制备份

很遗憾，目前CentOS6、CentOS7默认携带的QEMU还不支持qcow2块复制的功能，如果要使用这个功能，只能手工编译QEMU或者使用Fedaro，下面的测试需要在Fedaro21上进行。

blockcopy必须是临时生成的虚拟机，不能是define状态的虚拟机，先将在线的虚拟机xml文件备份，命令如下：

```
virsh  dumpxml  --inactive win7 >/tmp/win.xml
```

然后undefine之后才能使用blockcopy，命令如下：

```
virsh  undefine win7
```

将虚拟机的一块硬盘blockcopy到/home/win72.bak文件中，命令如下：

```
virsh  blockcopy --domain win7 hda  /home/win72.bak --wait --finish --verbose
块复制：[100 %]
```

通过qemu-img命令查看，命令如下：

```
[root@f21b home]# qemu-img info win72.bak
image: win72.bak
file format: qcow2
virtual size: 50G (53687091200 bytes)
disk size: 8.3G
cluster_size: 65536
Format specific information:
compat: 1.1
lazy refcounts: false
```

再将文件复制到备用宿主机上就可以。

（5）在线的带存储迁移

有共享存储的环境，在线迁移很简单，从CentOS 6.4开始，支持在线的带存储迁移，就是不需要共享存储也能在线迁移，这是一个非常强大的功能，操作方法如下。

① 在备用宿主机上通过qemu-img create命令创建好空文件，大小和要备份的虚拟机一模一样。

```
qemu-img create -f qcow2 -o preallocation=metadata CentOS6.qcow2 150G
```

空文件需加预分配参数，要不复制过去的文件会是按照qcow2预分配文件的大小。

1107114476@qq.com
dolphinebook.com

② 设置迁移速度，一般不希望迁移的时候对正常业务造成影响，设置迁移速度限制到600M/s。

```
virsh migrate-setspeed CentOS6 600
```

③ 迁移，命令如下：

```
virsh migrate --live --copy-storage-all --unsafe --persistent CentOS6
qemu+ssh://192.168.0.101/system
```

迁移时间随磁盘空间大小而不通，磁盘空间越大，迁移时间越长。迁移的过程是先迁移磁盘，然后迁移内存，迁移内存的时候，一般业务会瞬断下。虚拟机一瞬间会在一边关闭，另一边的宿主机开启。

迁移速度和成功率与网络有很大的关系，宿主机的网络尽量是万兆或者千兆多网卡绑定，可以提高迁移速度和成功率。

21.3.2　单机虚拟化应急方案

1. 硬件故障的检查

从长期实践来看，单机虚拟化最常见的故障是硬件故障，而在硬件中，磁盘故障的概率最高，所以，对宿主机磁盘的健康情况的监控是宿主机监控的一个重点，新一代 Intel 志强 E5 v2、v3 平台上的服务器，带外管理卡都支持磁盘的健康监控，如图 21-53 所示的就是 IBM 服务器的磁盘健康监控，管理卡也支持 CPU、内存、网卡、电源、风扇等硬件报警，所以，强烈建议将服务器的带外管理卡报警配置起来，这样硬件有故障的时候，就会自动发送报警信息。

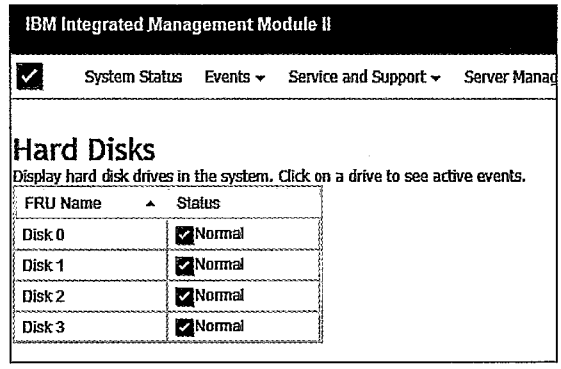


图 21-53　IBM IMM2 磁盘健康状态查看

在宿主机操作系统内部也可以使用 RAID 卡管理工具结合 crontab 来实现硬盘健康状态的监控。目前，服务器上常用的 RAID 卡主要是 LSI 和 HP 服务器专用的 RAID 卡，LSI 的低端的 RAID 卡可以使用 lsiutil 工具，高端的 RAID 卡可以使用 MegaCLi64 工具，HP 的 RAID 卡可以使用 HPACU 工具，工具都可以在 RAID 卡或者服务器厂商的网站上下载，下面是在宿主机上使用的一个脚本，通过 crontab，每天在凌晨检查一次硬盘，当发现有硬盘故障的时候，就会调用 Python 发邮件的一个脚本，发邮件处理，脚本供读者参考。

1107114476@qq.com
dolphinebook.com

```
cat /bin/checkdisk.sh
#!/bin/bash
source /etc/profile
myip=`/sbin/ifconfig |grep "inet addr"`
myname=`hostname`
echo $myip > /tmp/q.txt
echo $myname >> /tmp/q.txt
# 生成临时文件，并把 IP、服务器名字写入临时文件
check_mpt()
# 通过 mpt 的 RAID 卡管理程序，检查是否有硬盘故障
{

find /bin/lsiutil.x86_64
if [ $? = 1   ];then
bash /opt/sys_scripts/misc/check_disk/install.sh reinstall
fi

/bin/lsiutil.x86_64 -p1  -a 21,2,0,0 |grep offline

if [ $? = 0  ];then
    echo $myip > /tmp/errmsg
    /bin/lsiutil.x86_64 -p1  -a 21,2,0,0 >> /tmp/errmsg
    /usr/bin/python /bin/pymail $myname
fi
}

check_Mega()
# 通过 megaraid 的 RAID 卡管理程序检查是否有硬盘故障
{
find /bin/MegaCli64
if [ $? = 1   ];then
bash /opt/sys_scripts/misc/check_disk/install.sh reinstall
fi

/bin/MegaCli64 -PdGetMissing -aALL  |grep "No Missing"
if [ $? = 0  ];then
    errmsg=`/bin/MegaCli64 -PdGetMissing -aALL`
else
    echo $myip > /tmp/errmsg
    /bin/MegaCli64 -PdGetMissing -aALL >> /tmp/errmsg
    /usr/bin/python /bin/pymail $myname
fi
critical_disk=`/bin/MegaCli64 -AdpAllInfo -aALL | grep "Critical Disks"|sed
's/^[ ]*//g'|awk -F ":" '{print $2}'|sed 's/^[ ]*//g'`
failed_disk=`/bin/MegaCli64 -AdpAllInfo -aALL | grep -i "Failed Disks"|sed
's/^[ ]*//g'|awk -F ":" '{print $2}'|sed 's/^[ ]*//g'`
if [ $critical_disk -gt 0  -o $failed_disk -gt 0 ];then
        echo $myip > /tmp/errmsg
        echo -e "\n" >>/tmp/errmsg
        /bin/MegaCli64 -AdpAllInfo -aALL | grep -i -EA9 "Device Present">>/tmp/errmsg
```

1107114476@qq.com
dolphinebook.com

```
        /usr/bin/python /bin/pymail $myname
else
        echo `/bin/MegaCli64 -AdpAllInfo -aALL | grep -i -EA9 "Device Present"`
fi
}

check_hp()
# 通过 HP 的 RAID 卡管理程序检查是否有硬盘故障
{
find /usr/sbin/hpacucli
if [ $? = 1   ];then
bash /opt/sys_scripts/misc/check_disk/install.sh reinstall
fi

/usr/sbin/hpacucli ctrl slot=0 pd all show status |grep Failed
if [ $? = 0   ];then
    echo $myip > /tmp/errmsg
    /usr/sbin/hpacucli ctrl slot=0 pd all show status >> /tmp/errmsg
    /usr/bin/python /bin/pymail $myname
fi
}

check_RAID_type()
# 通过 lspci 确定是哪种 RAID 卡
{
lspci |grep "MPT"|grep "LSI" > /dev/bull
if [ $? = 0   ];then
    RAID_type="MPT"
    check_mpt
fi

lspci |grep "MegaRAID" > /dev/bull
if [ $? = 0   ];then
    RAID_type="Mega"
    check_Mega
fi

lspci |grep "Hewlett_packard" > /dev/bull
if [ $? = 0   ];then
    RAID_type="HP"
    check_hp
fi;

lspci |grep "Hewlett_Packard" > /dev/bull
if [ $? = 0   ];then
    RAID_type="HP"
    check_hp
fi;
lspci |grep "Hewlett-Packard" > /dev/bull
if [ $? = 0   ];then
```

1107114476@qq.com
dolphinebook.com

```
    RAID_type="HP"
    check_hp
fi;

dmidecode  |grep "ProLiant" > /dev/bull
if [ $? = 0  ];then
    RAID_type="HP"
    check_hp
fi;
}
check_RAID_type
```

2. 故障的应对方案

单机虚拟化服务器故障分为硬件故障和系统故障，处理方案如表 21-1 所示。

表 21-1 单机虚拟化故障应急方案

内　容	操作序号	解决方案
硬件故障	单块磁盘故障	更换硬盘
	RAID 卡故障	硬盘迁移到备机
	物理机宕机	硬盘迁移到备机
系统、数据故障	Linux 系统崩溃	修复 / 重装
	数据卷损坏（磁盘物理无故障）	使用数据恢复技术尝试

硬件故障操作方法包括单块硬盘故障、更换同类型的硬盘、等待重建。

RAID 卡故障、主机故障、其他硬件故障修复时间较长或无配件更换时，把故障机器的硬盘按照顺序迁移到备机后，对 RAID 信息进行导入操作，大多数 RAID 会自动识别，并提示是否要导入外部的 RAID，导入并启动即可，操作的时候要仔细查看提示，以免磁盘上的数据被错误清除。

如果机房现场工程师配合，可以采用更换硬盘的方法，更换时要对硬盘进行编号，并将对应的硬盘插到同型号的服务器硬盘插槽上。如果现场没有工程师支持，并且数据有备份，可以考虑直接把数据恢复到备用的虚拟机上，开启业务。主要的评估点取决于方案对业务的损失最小，恢复业务时间更快。

系统、数据故障操作方法如下：

- 备份磁盘分区表。
- 备份 MBR 及 LVM 信息。

备份分区表信息的操作方法如下：

```
# 导出分区表信息
sfdisk -d /dev/cciss/c0d0 > /opt/backup/`hostname`/`data "+%Y%m%d" `/mbr/c0d0.out
```

1107114476@qq.com
dolphinebook.com

```
 sdfisk -d /dev/cciss/c0d1> /opt/backup/`hostname`/`data "+%Y%m%d" `/mbr/c0d1.out
# 备份 MBR
dd if=/dev/cciss/c0d0  of=/opt/backup/`hostname`/`data "+%Y%m%d" `/mbr/c0d0.
mbr bs=512 count=1
dd if=/dev/cciss/c0d1  of=/opt/backup/`hostname`/`data "+%Y%m%d" `/mbr/c0d1.
mbr bs=512 count=1
```

备份 LVM 信息的操作命令如下：

```
cp -r /etc/lvm/backup/ /opt/backup/`hostname`/`data "+%Y%m%d" `/lvm
```

备份虚拟机配置信息的操作命令如下：

```
cp -r /etc/libvirt/qemu/*.xml /opt/backup/`hostname`/`data \
"+%Y%m%d" `/vmconfig
```

KVM 宿主机系统崩溃的修复方法如下：

（1）如果是系统故障，修复故障系统

使用引导盘进入 rescue 模式修复系统故障，无法修复或修复时间无法预计时，重装 KVM 宿主机系统。

（2）如果不能修复，重装 KVM 宿主机系统

重装系统之前，建议先用 LiveCD 启动机器，建立 nfs，先将虚拟机相关镜像文件、xml 配置文件复制到备用的宿主机上，在备用宿主机上开启虚拟机，先恢复业务，然后对系统进行彻底重装。

21.3.3　集群虚拟化应急方案

虚拟化集群的计算节点要按照单机虚拟化进行监控、报警。

虚拟化集群的计算节点故障处理相对比较简单，因为集群一般使用共享存储，虚拟机的镜像保存在共享存储上，所以，在计算节点故障的时候，可以将虚拟机在线迁移到其他计算节点上。如果计算节点彻底宕机，可以在另外的计算节点上手工开起虚拟机，或者通过管理平台配置高可用，计算节点宕机的时候，可以自动地将虚拟机在其他计算节点上开启。

虚拟化集群的共享存储是核心，对保障共享存储的安全非常重要。

如果共享存储使用的是商业存储，一般监控报警做得非常完善，建议按照厂商相关文档进行配置。商业存储也不意味着就是百分百的保险，如果使用商业存储，建议将异步或者同步的备份也配置起来，这样万一存储有多块硬盘同时故障的时候，也可以对数据进行恢复。第 22 章有一个案例，就是在生存环境中使用商业存储，有多块硬盘故障，导致 50 多台虚拟机丢失的惨痛教训。

如果共享存储使用的分布式文件系统，建议一份镜像至少写 3 份，对存储节点的报警、监控也可以参照单机虚拟化的方法，尤其是磁盘的监控。

虚拟化集群的应急预案如表 21-2 所示。

1107114476@qq.com
dolphinebook.com

表 21-2 集群虚拟化故障应急方案

故障类型	处理方案
虚拟机故障	① 虚拟化支持工程师排查故障或者开启备用虚拟机 ② 业务运维配合
宿主机性能出现瓶颈	虚拟化支持工程师在线迁移部分虚拟机到集群内其他计算节点（无停机时间）
宿主机故障	① 虚拟化支持工程师在集群内其他宿主机上重开启虚拟机 ② 业务运维配合
存储故障	① 虚拟化支持工程师从备份恢复数据，需要时间可能比较长 ② 业务运维配合

故障处理步骤如下。

（1）虚拟机故障的处置

故障场景：业务运维无法访问到虚拟机，但虚拟化支持工程师可正常访问宿主机。

处置步骤如下：

① 如果业务运维不希望立即重置虚拟机，而是保留故障现场并登录到 console 检查虚拟机情况，可用 virsh dumpxml [vmname] 查看到 vnc 端口，并提供宿主机 IP、该虚拟机 vnc 端口和 vnc 口令给业务运维，由业务运维用 vnc 访问虚拟机控制台后自行处置。

② 如果业务运维希望立即重置虚拟机，使用 virsh destroy [vm name] 强行关闭虚拟机，然后重置虚拟机。

③ 如果虚拟机重置后依然无法使用，需访问虚拟机控制台排查原因。如果该虚拟机文件系统崩溃导致无法使用，视实际情况重建新的虚拟机并由业务运维重新安装软件。

（2）性能瓶颈的处置

故障场景：虚拟化支持工程师收到性能监控报警，提示某台宿主机的某项性能超过限值。

处置步骤如下：

① 登录性能监控页面，确认性能异常。

② 登录有问题的宿主机，使用 virt-top 命令查看每个虚拟机的资源使用情况，查看集群内物理机资源使用情况，迁移部分虚拟机到其他集群，命令为 virsh migrate --live [vm name] --desturi qemu+tcp://[new node]/system。

密切关注相关性能数据，视情况觉得这个调整是否是长期的，并更新管理平台。

（3）宿主机故障处置

故障场景：虚拟化支持工程师收到宿主机离线报警，或者业务运维发现虚拟机无法访问后虚拟化支持工程师发生宿主机无法连接。

处置步骤如下：

① 通过带外管理卡检测宿主机。如果有必要则可以重启宿主机。

② 如果重启后故障依旧，视情况在相对空闲的其他宿主机上开启受影响的虚拟机。更新系统中的对应关系。

③ 如果虚拟机重开后依然无法使用，需访问虚拟机控制台排查原因。如果该虚拟机文件

1107114476@qq.com
dolphinebook.com

系统崩溃导致无法使用，视实际情况重建新的虚拟机并由业务运维重新安装软件。

业务运维确认业务恢复后，与现成工程师联系检测维修有故障的宿主机。

（4）存储故障处置

故障场景：当存储软件故障或其他无法预测的故障而无法使用时，启用备用存储及备用存储上的备份数据。

处置步骤如下：参照具体存储厂商说明文档或者分布式文件系统恢复文档。

21.4 本章小结

本章介绍了虚拟化中监控、报警的方法和笔者的经验，应急响应是对运维能力最重要的考验项之一。应急响应方案应做到逻辑清晰，考虑到各种极端情况，最重要的是要定期演练，一方面可以验证方案的有效性，另一方面可以让每个参与人员能够熟悉应急方案，再有紧急情况发生的时候，就能比较熟练地应付。

下一章将为读者介绍一个业务在虚拟化上非常慢的案例。

1107114476@qq.com
dolphinebook.com

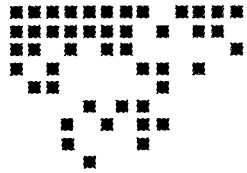
1107114476@qq.com
dolphinebook.com

第五篇 Part 5

KVM 虚拟化典型案例

■ 第 22 章　生产环境问题案例与分析

1107114476@qq.com
dolphinebook.com



Chapter 22 第 22 章

生产环境问题案例与分析

22.1 游戏在虚拟机上非常慢的案例

本章介绍一个在虚拟机上非常奇怪的问题案例，当时花了很长时间才找到原因，这个问题的解决过程，对解决类似问题的思路方法，有借鉴意义。

1. 现象

2009 年的时候，在虚拟化平台下载了一款游戏，当时，宿主机操作系统是 CentOS 5.5 64 位版本，虚拟机操作系统版本是 Windows Server 2003 中文企业版。到 2010 年底的时候，CentOS 6.0 发布，CentOS 6.0 在虚拟化方面添加了许多新的功能，所以，计划将这款游戏逐步迁移到 CentOS 6.0 上面。

在迁移的过程中，出现一个非常奇怪的现象，就是这款游戏的服务器端在虚拟机上特别慢，在 CentOS 5.5 的宿主机上，游戏服务器端程序启动需要 2 分钟，在 CentOS 6.0 的宿主机上，游戏服务器服务器端程序启动需要 10 多分钟，并且即使勉强启动成功，用客户端连上来，也是非常慢，根本不能进行游戏。

2. 原因查找过程

经过和游戏开发合作，很快将问题定位到一个游戏开发中使用的函数上，游戏中用到一个系统函数，即 QueryPerformanceFrequency()。

（1）QueryPerformanceFrequency() 基本介绍

类型：Win32API。

原型：BOOL QueryPerformanceFrequency（LARGE_INTEGER *lpFrequency）。

作用：返回硬件支持的高精度计数器的频率。

1107114476@qq.com
dolphinebook.com

返回值：若为非零，硬件支持高精度计数器；若为零，硬件不支持，读取失败。

（2）QueryPerformanceFrequency() 技术特点

供 WIN9X 使用的高精度定时器：QueryPerformanceFrequency() 和 QueryPerformance Counter()，要求计算机从硬件上支持高精度定时器，就是得到很高的 CPU 的精度。

游戏开发写了一个程序，专门调用这个函数做循环，输入的参数是循环的次数，在 CentOS 5.5 和 CentOS 6.0 上的虚拟机，分别循环测试 10 000 次，可以看到结果相差奖金 200 倍。在 CentOS 6 的宿主机上的虚拟机测试下来，在这个函数上面耗时很长，导致整个 CPU 的效率很低。

```
b: 如果绑定到CPU，输入CPU索引，（1开始）
c: 测试次数，如10000
请输入：
1,1,1000000
你的输入：1,1,1000000
绑定到CPU
指定CPU，当前运行在1CPU
本机器支持QueryPerformanceFrequency特性

开始测试QueryPerformanceFrequency性能<调用1000000次>...
测试QueryPerformanceFrequency性能完毕，次数：1000000，耗时:125

开始测试GetTickCount性能<调用1000000次>...
测试GetTickCount性能完毕，次数：1000000，耗时:15
```

图 22-1　CentOS 5.5 宿主机上虚拟机函数 10000 次耗时

```
b: 如果绑定到CPU，输入CPU索引，（1开始）
c: 测试次数，如10000
请输入：
1,1,1000000
你的输入：1,1,1000000
绑定到CPU
指定CPU，当前运行在1CPU
本机器支持QueryPerformanceFrequency特性

开始测试QueryPerformanceFrequency性能<调用1000000次>...
测试QueryPerformanceFrequency性能完毕，次数：1000000，耗时:2704
```

图 22-2　CentOS 6.0 宿主机上虚拟机函数 10000 次耗时

因此，首先怀疑的是新版的 KVM 对 QueryPerformanceFrequency() 这个函数的支持是不是有缺陷，导致出现这样的问题，但是从理论上讲，可能性似乎不大。

后来，又花了两周时间专门做对比测试，终于找到了原因。首先，将 CentOS 6 上的虚拟机复制到 CentOS 5.5 上，发现问题会浮现；然后，将 CentOS 5.5 上的虚拟机复制到 CentOS 上，发现不会出现问题。

到这一步，怀疑是不是 CentOS 6 这个版本上的虚拟机镜像制作有问题。于是，重新安装了一个纯净版的 Windows 系统，重新测试，发现不会出现问题，现在，问题应该可以定位到镜像配置上了。

然后，又对镜像配置进行了详细的检查，对每项配置进行了关闭及开启对比测试，终于找到了原因，在同时具备以下 3 个条件时，会触发问题而导致游戏很慢：

1107114476@qq.com
dolphinebook.com

① c:\boot.ini 里面的两个参数 usepmtimer、pae 同时添加、使用 QueryPerformanceFrequency() 函数。

② usepmtimer 是为了保证虚拟机走时准确，所有的虚拟机，不管是 VMWare、HyperV、XEN，还是 KVM，都有一个问题，就是虚拟机的时钟会走的不精确，因为虚拟机的时钟是虚拟出来的，不像物理机，时钟可以从主板上的计时器获取数据，所以，如果是 Linux 系统，一般都要配置 tsc 这样的半虚拟化时钟，如果是 Windows 系统，一般都要配置 usepmtimer 这个参数，并且建议所有的虚拟机都设置 ntp 时间服务器，以保证虚拟机的走势准确。

关于 CPU0 压力大时对虚拟机时钟的影响，在虚拟机的使用中，当虚拟机的 CPU0 压力持续很大的时候，也会对虚拟机的时间也会产生走时不准的情况，所以，建议在生产环境，虚拟机一定要设置 ntp。

③ pae 是为了保证 windows 2003 32 位系统能能够识别更大的内存，如图 22-3 所示。

后来，笔者又在不同的虚拟化引擎上，如 VMWare、hyperV，甚至是物理机上做了对比测试，发现只要上述 3 个条件具备，肯定会触发这个现象；然后又在 Windows Server 2008 R2 上做了对比测试，发现现象是一样的，所以，这个问题应该是 Windows 系统的一个 bug。

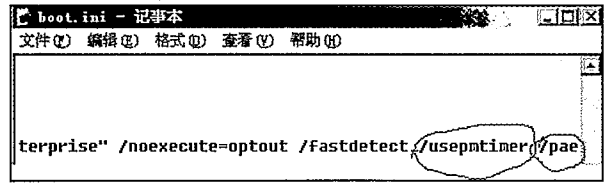


图 22-3 windows 2003 系统的 boot.ini 设置

3. 处理方法

由于当时虚拟机的内存并没有大于 4G，所以，去掉了 pae 这个参数。另外，游戏开发在后续的服务器程序不再使用 QueryPerformanceFrequency() 这个函数，采用其他的方式提高游戏精度。

4. 问题处理思路总结

这个问题确实非常怪异，一开始只是在怀疑虚拟机，经过大量的对比测试，才真正的把问题搞清楚，所以，在虚拟化实施过程中，碰到问题，需要认真细致的排查，不能轻易地下结论，尽量和业务程序多沟通，遇到问题不轻易放弃，争取能找到真正的原因。

解决过程很曲折，回过头来总结经验，对比测试是一个解决问题的法宝，通过对比测试，抽丝剥茧，可以一步一步地找到问题。

22.2 虚拟机流量过高引起网络风暴的案例一

单线双 IP 是一种典型的多线机房使用方式，本章介绍一个在单线双 IP 机房实施虚拟化的时候，碰到的一个惨痛案例，希望以后在单线双 IP 机房实施虚拟化的读者能避免出现类似

1107114476@qq.com
dolphinebook.com

的问题。

1. 现象

在单线多 IP 的机房，为了测试虚拟机之间的网络性能，做了两台宿主机上两台虚拟机之间的网络压力测试，两台宿主机在同一交换机下面，测试开始 10 分钟后，整个机房机房网络出现网络风暴，机房出口被堵死，机房所有服务器对外中断服务，然后紧急将两台宿主机关机，机房网络恢复正常。

出现问题的环境如下：

（1）硬件环境

- 服务器品牌：不知名品牌。
- CPU：E5-2640 *2。
- 硬盘：600GB*1。
- 内存：16GB。

（2）软件环境

- 宿主机：CentOS 6.4 64 位，内核版本 2.6.32-431.17.1.el6.x86_64，两台。
- 虚拟机：Ubantu 12.04，64 位服务器版，内核版本 3.10，每台宿主机虚拟机一台。
- 软件工具：Netperf。

（3）网络环境

机房网络为单线双 IP，一个运营商甲 IP，另一个运营商乙 IP，未作 VLAN 划分，两个网段在同一个 VLAN 内部，两台宿主机在同一交换机下面，带宽 1000M。

2. 问题再现与原因查找

为了测试到底是什么原因引起这样的现象，在测试环境搭建了一个环境，对各种网络结构分别做了测试，测试环境如下：

（1）硬件环境

- 服务器品牌：HP DL380G5。
- CPU：E5430。
- 内存：16GB。
- 硬盘：146GB*1。

（2）软件环境

- 宿主机：CentOS 6.5 64 位，内核版本 2.6.32-431.20.3.el6.x86_64，两台。
- 虚拟机：Ubuntu12.04，内核版本 3.10，两台。

（3）网络环境

为什么要进行如下 8 种网络环境的测试呢？主要是为了彻底测试不同的子网及桥组合配置下，是否会引起相同的网络风暴现象。

1107114476@qq.com
dolphinebook.com

① 网络结构图一：宿主机使用单网卡单 IP，创建桥 br0，虚拟机配置双网卡（都桥接到 br0），如图 22-4 所示。

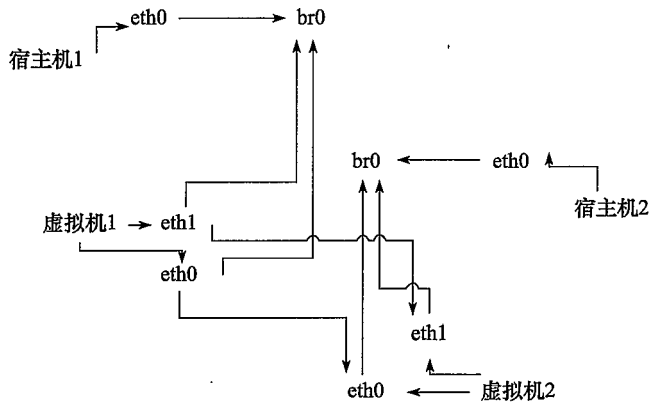


图 22-4 宿主机单网卡单桥，虚拟机双网卡桥 br0

宿主机网络配置，宿主机单网卡单 IP，命令如下：

```
ifconfig
br0         Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.0.150  Bcast:192.168.0.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:1795957 errors:0 dropped:0 overruns:0 frame:0
            TX packets:6229 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:133290840 (127.1 MiB)  TX bytes:438593 (428.3 KiB)
eth0        Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:1863264 errors:0 dropped:0 overruns:0 frame:0
            TX packets:6225 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:171318323 (163.3 MiB)  TX bytes:504379 (492.5 KiB)
...
```

虚拟机网络配置，虚拟机双网卡双 IP，虚拟机分配两块网卡，全部桥到 br0 上，命令如下：

```
virsh # dumpxml 1
...
<interface type='bridge'>
        <mac address='52:54:00:99:fb:af'/>
        <source bridge='br0'/>
        <target dev='vnet0'/>
        <model type='virtio'/>
        <alias name='net0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <interface type='bridge'>
```

1107114476@qq.com
dolphinebook.com

```
        <mac address='52:54:00:48:c2:86'/>
        <source bridge='br0'/>
        <target dev='vnet1'/>
        <model type='virtio'/>
        <alias name='net1'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </interface>
```

② 网络结构图二：宿主机配置单网卡双 IP 及两个桥 br0、br0:1，虚拟机配置双网卡（eth0 桥接到 br0；eth1 桥接到 br0:1），如图 22-5 所示。

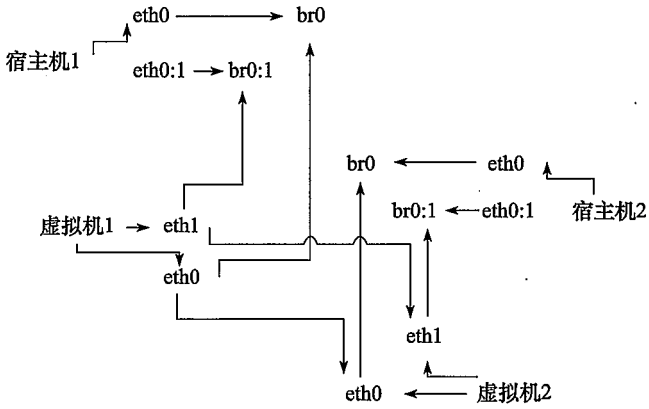


图 22-5　宿主机单网卡、网桥、子网桥，虚拟机双网卡桥 br0，br0:1

宿主机网络配置，宿主机单网卡双 IP，命令如下：

```
ifconfig
br0       Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          inet addr:192.168.0.150  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1796577 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6338 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:133340838 (127.1 MiB)  TX bytes:455527 (444.8 KiB)
br0:1     Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          inet addr:192.168.1.150  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
...
```

虚拟机网络配置，虚拟机双网卡双 IP，虚拟机分配两块网卡，分别桥到 br0，br0:1 上，命令如下：

```
virsh # dumpxml 1
...
<interface type='bridge'>
        <mac address='52:54:00:99:fb:af'/>
        <source bridge='br0'/>
```

1107114476@qq.com
dolphinebook.com

```
        <target dev='vnet0'/>
        <model type='virtio'/>
        <alias name='net0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <interface type='bridge'>
        <mac address='52:54:00:48:c2:86'/>
        <source bridge='br0:1'/>
        <target dev='vnet1'/>
        <model type='virtio'/>
        <alias name='net1'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </interface>
```

③ 网络结构图三：宿主机配置单网卡双 IP 及桥 br0、br0:1，虚拟机配置双网卡（都桥接到 br0），如图 22-6 所示。

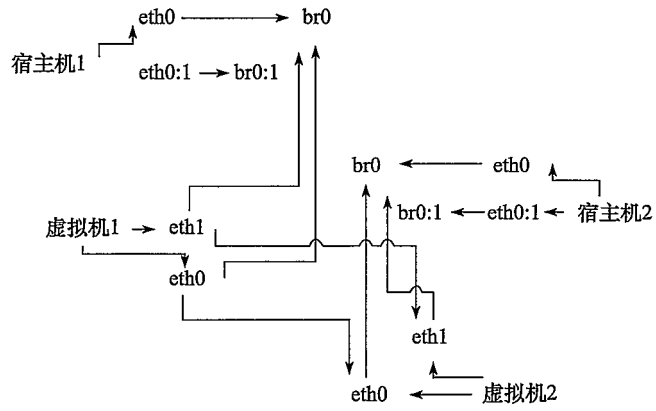


图 22-6 宿主机单网卡、网桥、子网桥，虚拟机双网卡桥 br0

宿主机网络配置，宿主机单网卡双 IP，命令如下：

```
ifconfig
br0         Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.0.150  Bcast:192.168.0.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:1796577 errors:0 dropped:0 overruns:0 frame:0
            TX packets:6338 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:133340838 (127.1 MiB)  TX bytes:455527 (444.8 KiB)
br0:1       Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.1.150  Bcast:192.168.1.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
...
```

虚拟机网络配置，虚拟机双网卡双 IP，虚拟机分配两块网卡，都桥到 br0 上，命令如下：

1107114476@qq.com
dolphinebook.com

```
virsh # dumpxml 1
...
<interface type='bridge'>
      <mac address='52:54:00:99:fb:af'/>
      <source bridge='br0'/>
      <target dev='vnet0'/>
      <model type='virtio'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <interface type='bridge'>
      <mac address='52:54:00:48:c2:86'/>
      <source bridge='br0:1'/>
      <target dev='vnet1'/>
      <model type='virtio'/>
      <alias name='net1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </interface>
```

④ 网络结构图四：宿主机配置单网卡双 IP 及桥 br0、br0:1，虚拟机配置单网卡（桥接到 br0），如图 22-7 所示。

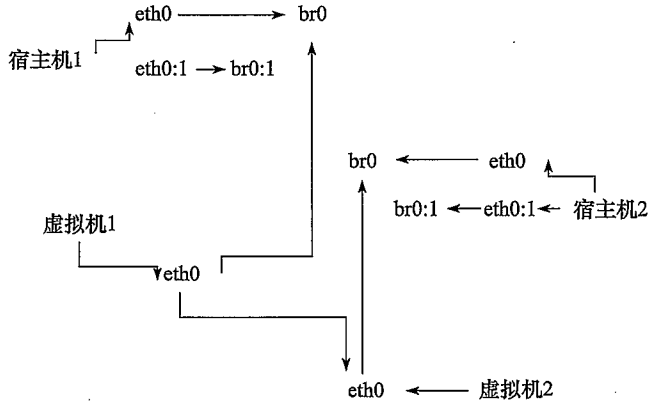


图 22-7　宿主机单网卡、网桥、子网桥，虚拟机单网卡桥 br0

宿主机网络配置，宿主机单网卡双 IP，命令如下：

```
ifconfig
br0         Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.0.150  Bcast:192.168.0.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:1796577 errors:0 dropped:0 overruns:0 frame:0
            TX packets:6338 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:133340838 (127.1 MiB)  TX bytes:455527 (444.8 KiB)
br0:1       Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.1.150  Bcast:192.168.1.255  Mask:255.255.255.0
```

1107114476@qq.com
dolphinebook.com

```
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
...
```

虚拟机网络配置，虚拟机单网卡单 IP，桥到 br0 上，命令如下：

```
virsh # dumpxml 1
...
<interface type='bridge'>
        <mac address='52:54:00:99:fb:af'/>
        <source bridge='br0'/>
        <target dev='vnet0'/>
        <model type='virtio'/>
        <alias name='net0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
...
```

⑤ 网络结构图五：宿主机配置单网卡多 IP 及桥 br0、br0:1、br0:2，虚拟机配置单网卡（桥接到 br0），如图 22-8 所示。

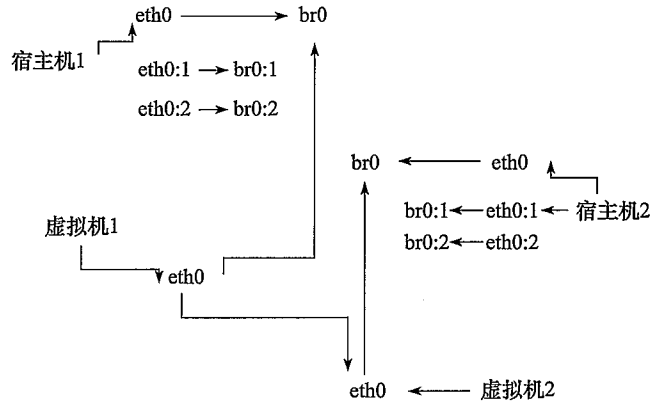


图 22-8 宿主机单网卡、网桥、多子网桥，虚拟机单网卡桥 br0

宿主机网络配置，宿主机单网卡多 IP，命令如下：

```
ifconfig
br0         Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.0.150  Bcast:192.168.0.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:1796943 errors:0 dropped:0 overruns:0 frame:0
            TX packets:6408 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:133369641 (127.1 MiB)  TX bytes:465203 (454.2 KiB)

br0:1       Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.1.150  Bcast:192.168.1.255  Mask:255.255.255.0
```

1107114476@qq.com
dolphinebook.com

```
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

br0:2     Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          inet addr:192.168.2.150  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
...
```

虚拟机网络配置，虚拟机单网卡单 IP，桥到 br0 上，命令如下：

```
virsh # dumpxml 1
...
<interface type='bridge'>
      <mac address='52:54:00:99:fb:af'/>
      <source bridge='br0'/>
      <target dev='vnet0'/>
      <model type='virtio'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
   </interface>
...
```

⑥ 网络结构图六：宿主机配置单网卡多 IP 及桥 br0、br0:1、br0:2，虚拟机配置单网卡（桥接到 br0:2），如图 22-9 所示。

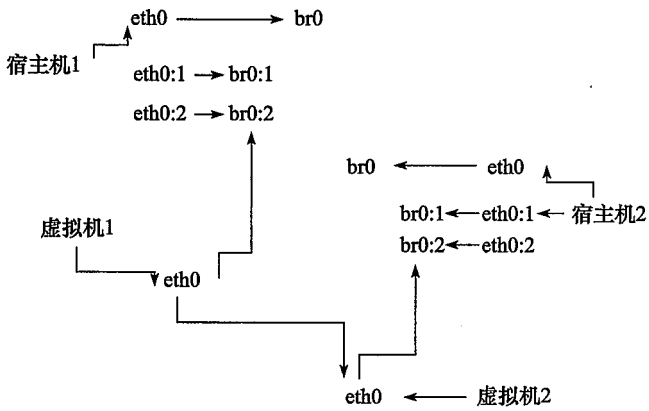


图 22-9　宿主机单网卡、网桥、多子网桥，虚拟机单网卡桥子网卡 br0:2

宿主机网络配置，宿主机单网卡多 IP，命令如下：

```
ifconfig
br0       Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          inet addr:192.168.0.150  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1796943 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6408 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

1107114476@qq.com
dolphinebook.com

```
            RX bytes:133369641 (127.1 MiB)  TX bytes:465203 (454.2 KiB)

br0:1       Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.1.150  Bcast:192.168.1.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

br0:2       Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.2.150  Bcast:192.168.2.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
...
```

虚拟机网络配置，虚拟机单网卡单 IP，桥到 br0:2 上，命令如下：

```
virsh # dumpxml 1
...
<interface type='bridge'>
        <mac address='52:54:00:99:fb:af'/>
        <source bridge='br0:2'/>
        <target dev='vnet0'/>
        <model type='virtio'/>
        <alias name='net0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
...
```

⑦ 网络结构图七：宿主机配置单网卡多 IP 及桥 br0、br0:1、br0:2，虚拟机配置双网卡（eth0 桥接到 br0，eth1 桥接到 br0:2），如图 22-10 所示。

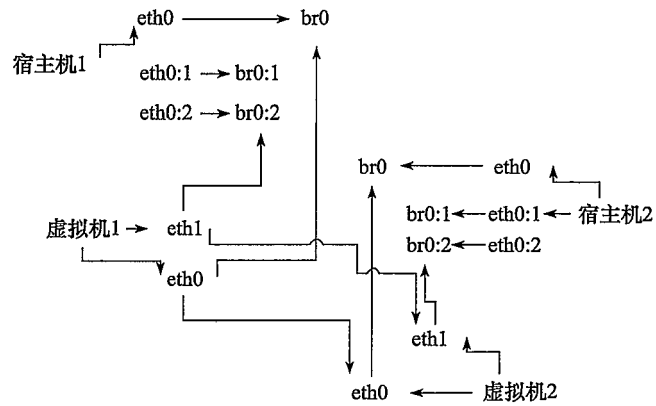


图 22-10 宿主机单网卡、网桥、多子网桥，虚拟机双网卡桥 br0、子网卡 br0:2

宿主机网络配置，宿主机单网卡多 IP，命令如下：

```
ifconfig
br0         Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:192.168.0.150  Bcast:192.168.0.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

1107114476@qq.com
dolphinebook.com

```
          RX packets:1796943 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6408 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:133369641 (127.1 MiB)  TX bytes:465203 (454.2 KiB)

br0:1     Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          inet addr:192.168.1.150  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

br0:2     Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          inet addr:192.168.2.150  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
...
```

虚拟机网络配置，虚拟机双网卡双 IP，第一个块网卡桥到 br0 上，第二块网卡桥到 br0:2 上，命令如下：

```
virsh # dumpxml 1
...
<interface type='bridge'>
      <mac address='52:54:00:99:fb:af'/>
      <source bridge='br0'/>
      <target dev='vnet0'/>
      <model type='virtio'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <interface type='bridge'>
      <mac address='52:54:00:48:c2:86'/>
      <source bridge='br0:2'/>
      <target dev='vnet1'/>
      <model type='virtio'/>
      <alias name='net1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </interface>
...
```

⑧ 网络结构图八：宿主机配置双网卡双 IP 及桥 br0、br1，虚拟机配置双网卡（eth0 桥接到 br0，eth1 桥接到 br1），如图 22-11 所示。

宿主机网络配置，宿主机单网卡双 IP，命令如下：

```
ifconfig
br0       Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          inet addr:192.168.0.150  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1796943 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6408 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:133369641 (127.1 MiB)  TX bytes:465203 (454.2 KiB)

br1       Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6E
```

1107114476@qq.com
dolphinebook.com

```
          inet addr:172.18.254.150  Bcast:172.18.254.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:168 (168.0 b)

eth0      Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1864275 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6403 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:171416918 (163.4 MiB)  TX bytes:531595 (519.1 KiB)

eth1      Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6E
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)...
...
```

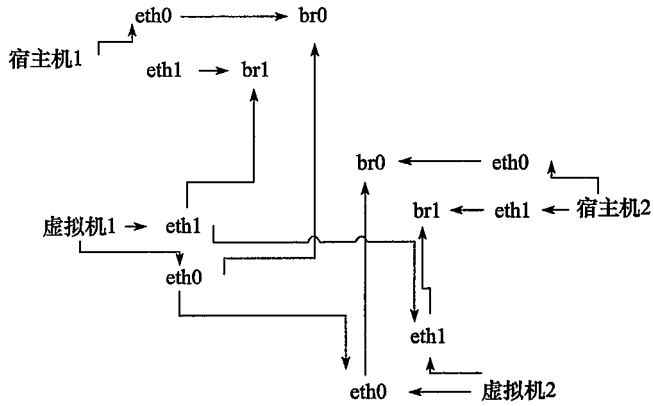


图 22-11 宿主机双网卡、双网桥，虚拟机双网卡桥 br0、br1

虚拟机网络配置，虚拟机多网卡双 IP，第一个块网卡桥到 br0 上，第二块网卡桥到 br1 上，命令如下：

```
virsh # dumpxml 1
...
<interface type='bridge'>
        <mac address='52:54:00:99:fb:af'/>
        <source bridge='br0'/>
        <target dev='vnet0'/>
        <model type='virtio'/>
        <alias name='net0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
```

1107114476@qq.com
dolphinebook.com

```
    </interface>
    <interface type='bridge'>
        <mac address='52:54:00:48:c2:86'/>
        <source bridge='br1'/>
        <target dev='vnet1'/>
        <model type='virtio'/>
        <alias name='net1'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </interface>
...
```

⑨ 网络结构图九：宿主机配置单网卡多 VLAN 及桥 br0、br0.100、br0.200，虚拟机配置双网卡（eth0 桥接到 br0.100，eth1 桥接到 br0.200），如图 22-12 所示。

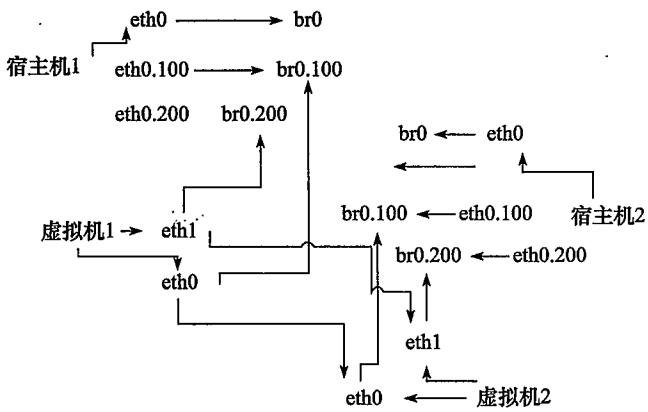


图 22-12　宿主机单网卡多 vlan、网桥、多子网桥，虚拟机双网卡桥子 van br0.100、br0.200

宿主机网络设置，宿主机单网卡多 VLAN，命令如下：

```
ifconfig
...
br0         Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:0 (0.0 b)  TX bytes:168 (168.0 b)

br1.200     Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
            inet addr:172.18.200.150  Bcast:172.18.200.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:0 (0.0 b)  TX bytes:336 (336.0 b)
...
eth0        Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
```

1107114476@qq.com
dolphinebook.com

```
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

eth0.100  Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

eth0.200  Link encap:Ethernet  HWaddr 00:21:5E:09:E7:6C
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

虚拟机网络配置，虚拟机双网卡双 IP，第一个块网卡桥到 br0.100 上，第二块网卡桥到 br0.200 上，命令如下：

```
virsh # dumpxml 1
...
<interface type='bridge'>
        <mac address='52:54:00:99:fb:af'/>
        <source bridge='br0.100'/>
        <target dev='vnet0'/>
        <model type='virtio'/>
        <alias name='net0'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <interface type='bridge'>
        <mac address='52:54:00:48:c2:86'/>
        <source bridge='br0.200'/>
        <target dev='vnet1'/>
        <model type='virtio'/>
        <alias name='net1'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </interface>
...
```

（4）测试内容、工具及方法

测试目的：测试网络结构的合理性，测试过程中是否会引起网络中断，为以后同样环境下虚拟机的使用积累经验。

测试用例（见表 22-1）：

1107114476@qq.com
dolphinebook.com

表 22-1　多网络结构测试用例

项　目	参　数
TCP_STREAM	测试 TCP 吞吐

测试工具：

- netperf-2.5.0。
- iperf-2.0.5。

测试方式：

虚拟机到虚拟机之间打压力。

Netperf 测试：

在虚拟机 1 上启动 netserver，命令如下：

```
netserver
```

在虚拟机 2 上输入测试命令（eth0—eth0），显示如下：

```
netperf  -c -H 192.168.200.207 -l 1800 -t TCP_STREAM -- -M 1048576 -m 1048576
```

在虚拟机 2 上输入测试命令（eth1—eth1），显示如下：

```
netperf  -c -H 192.168. 200.207  -l 1800 -t TCP_STREAM -- -M 1048576 -m 1048576
```

Iperf 测试：

在虚拟机 1 上启动 server 模式，命令如下：

```
iperf -s -w 1M
```

在虚拟机 2 上以 client 模式启动，输入测试命令，显示如下：

```
iperf  -c 192.168.200.207 -w 1M -t 900
```

（5）测试结论

经过测试，结论如下：

① 如表 22-2 所示的网络结构为四、五、六，并且通过端口映射进行，两台虚拟机之间进行网络压力测试会引起网络中断。

② 所有网络结构，如果不进行端口映射，则网络性能测试不会引起网络中断。

表 22-2　各种网络架构是否引起网络风暴结果

网络结构	宿主机网络配置	虚拟机网络配置	网络桥接方式	端口映射	出现现象
网络结构一	单网卡单 IP	双网卡	都桥接到 br0	否	否
网络结构二	单网卡双 IP	双网卡	eth0 桥接到 br0　eth1 桥接到 br0:1	否	否
网络结构三	单网卡双 IP	双网卡	都桥接到 br0	否	否
网络结构四	单网卡双 IP	单网卡	桥接到 br0	是	是
	单网卡双 IP	单网卡	桥接到 br0	否	否

1107114476@qq.com
dolphinebook.com

（续）

网络结构	宿主机网络配置	虚拟机网络配置	网络桥接方式	端口映射	出现现象
网络结构五	单网卡多 IP	单网卡	桥接到 br0	是	是
	单网卡多 IP	单网卡	桥接到 br0	否	否
网络结构六	单网卡多 IP	单网卡	桥接到 br0:2	是	是
	单网卡多 IP	单网卡	桥接到 br0:2	否	否
网络结构七	单网卡多 IP	双网卡	eth0 桥接到 br0　eth1 桥接到 br0:2	否	否
网络结构八	双网卡双 IP	双网卡	eth0 桥接到 br0　eth1 桥接到 br1	是	否
网络结构九	单网卡多 VLAN	双网卡	eth0 桥接到 br0.100　eth1 桥接到 br0.200	是	否
	单网卡多 VLAN	双网卡	eth0 桥接到 br0.100　eth1 桥接到 br0.200	否	否

网络正常现象：只要在宿主机上没做端口映射，在所有的网络结构下进行虚拟机之间的网络性能测试，都不会引起网络中断。

网络故障现象：在网络结构四、五、六情况下，若宿主机上设置端口映射，虚拟机之间的网络性能测试会引起网络中断。

（6）抓包及网络层面的分析

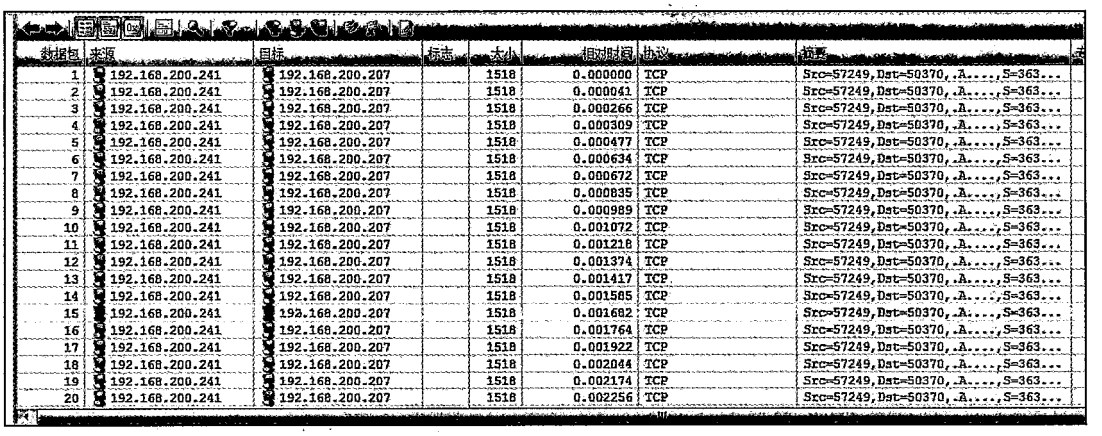
抓包分析以及查看交换机的 CAM 表信息发现，在 5 分钟后，交换机的 CAM 表中没有测试目标服务器 MAC 地址表项。根据交换机的工作机制，当收到一个数据包，若在 CAM 表中找不到包中的目标 MAC 地址，交换机会做未知单播处理，工作机制等同广播，会向交换机所有端口转发这个数据包，当另外的交换机收到这个数据包后，也会做广播处理，从而影响整个机房网络，直至机房网络完全中断。

如图 22-13 所示，在测试过程中，为什么在 5 分钟后交换机 CAM 表中就没有测试目标服务器的 MAC 地址表项呢？经抓包发现，在测试的过程中，只有测试源服务器到测试目标服务器的数据包，测试目标服务器一直没有回包。交换机是根据数据包的源 MAC 地址来更新 CAM 表的，在“学到”一个 MAC 地址并放在 CAM 表后，老化时间默认是 5 分钟，如果在 5 分钟内没再收到同样源 MAC 地址的包，交换机会将 CAM 中对应的 MAC 表项清除掉。因为目标服务器一直未回包，对应的交换机 CAM 表一直不能更新，5 分钟后自动删除目标服务器在 CAM 中的 MAC 地址表项，在测试 5 分钟后，导致整个网络出现广播风暴。

3. 问题处理方法

在单线多 IP 机房，宿主机只配置一个 IP、一个网桥，虚拟机配置两个网卡，都桥到宿主机的单网桥上，虚拟机一个网卡配置运营商甲 IP，另一个网卡配置运营商乙 IP，并做到运营商乙的 IP 段手工路由。因为宿主机的 IP 只是用于管理，所以，一个 IP 就足够了；因为主要的生产流量都是走虚拟机，所以，虚拟机配置两块网卡，分别绑定两个运营商的 IP。

1107114476@qq.com
dolphinebook.com



数据包	来源	目标	标志	大小	相对时间	协议	概要
1	192.168.200.241	192.168.200.207		1518	0.000000	TCP	Src=57249,Dst=50370,A....,S=363...
2	192.168.200.241	192.168.200.207		1518	0.000041	TCP	Src=57249,Dst=50370,A....,S=363...
3	192.168.200.241	192.168.200.207		1518	0.000266	TCP	Src=57249,Dst=50370,A....,S=363...
4	192.168.200.241	192.168.200.207		1518	0.000309	TCP	Src=57249,Dst=50370,A....,S=363...
5	192.168.200.241	192.168.200.207		1518	0.000477	TCP	Src=57249,Dst=50370,A....,S=363...
6	192.168.200.241	192.168.200.207		1518	0.000634	TCP	Src=57249,Dst=50370,A....,S=363...
7	192.168.200.241	192.168.200.207		1518	0.000672	TCP	Src=57249,Dst=50370,A....,S=363...
8	192.168.200.241	192.168.200.207		1518	0.000835	TCP	Src=57249,Dst=50370,A....,S=363...
9	192.168.200.241	192.168.200.207		1518	0.000989	TCP	Src=57249,Dst=50370,A....,S=363...
10	192.168.200.241	192.168.200.207		1518	0.001072	TCP	Src=57249,Dst=50370,A....,S=363...
11	192.168.200.241	192.168.200.207		1518	0.001216	TCP	Src=57249,Dst=50370,A....,S=363...
12	192.168.200.241	192.168.200.207		1518	0.001374	TCP	Src=57249,Dst=50370,A....,S=363...
13	192.168.200.241	192.168.200.207		1518	0.001417	TCP	Src=57249,Dst=50370,A....,S=363...
14	192.168.200.241	192.168.200.207		1518	0.001585	TCP	Src=57249,Dst=50370,A....,S=363...
15	192.168.200.241	192.168.200.207		1518	0.001682	TCP	Src=57249,Dst=50370,A....,S=363...
16	192.168.200.241	192.168.200.207		1518	0.001764	TCP	Src=57249,Dst=50370,A....,S=363...
17	192.168.200.241	192.168.200.207		1518	0.001922	TCP	Src=57249,Dst=50370,A....,S=363...
18	192.168.200.241	192.168.200.207		1518	0.002044	TCP	Src=57249,Dst=50370,A....,S=363...
19	192.168.200.241	192.168.200.207		1518	0.002174	TCP	Src=57249,Dst=50370,A....,S=363...
20	192.168.200.241	192.168.200.207		1518	0.002256	TCP	Src=57249,Dst=50370,A....,S=363...

图 22-13　流量过高引起网络风暴抓包

4. 问题处理思路总结

这个案例最大的教训是不应在生产环境做大压力测试，主要是流程上的问题，也提醒每一个运维，严格遵守相关的运维流程、规范是对自己最大的保护。

问题的技术原因分析，主要是通过在测试环境，模拟出问题的线上环境，对问题重现，并且进一步测试了各种网络情况下，是否会出现类似的问题。

通过近两周的测试，发现在以下条件下会触发故障：

- 同 VLAN 多网段配置子网卡并桥接虚拟机。
- 某个网段做了 NAT 映射。
- 网络大流量压力接近线速。

所以，在多线机房的虚拟化实施中，一定要避免出现类似的问题。

本节介绍的案例，后果非常严重，最大的教训就是对生产环境要有敬畏之心，在生产环境的操作，要充分验证再操作，切不可想当然。下一节再介绍一个没有充分验证，上线时造成严重影响的案例。

22.3　虚拟机流量过高引起网络风暴的案例二

笔者曾经在给一款知名的 RPG（Role-playing game，角色扮演游戏）手机游戏提供虚拟化支持期间遇到问题，印象非常深刻，本节分享下，希望读者在虚拟化实施时能有所借鉴。

1. 现象

一款游戏需要经过 CBT（内部测试）、OBT（公开测试）等阶段的测试才能最终上线。当时给此款游戏提供了配置为两颗 CPU、4G 内存的虚拟机，因为 RPG 类手游的压力主要是网卡的发包率，经过游戏项目组一段时间的测试没有发现虚拟机性能存在问题。代码调式以及发现的相关 bug 也都修复完成，讨论认为游戏已经能够满足上线前的要求，于是，提前广告

1107114476@qq.com
dolphinebook.com

投放、流量入口的推广，反响非常好，官网论坛也很活跃，大家都很期待。

开放时间定于当日 13：30，实际上在 12：30 左右已经开放了少许玩家进来，游戏运行稳定。随着玩家慢慢导入，到 13：30 对所有玩家全部开放。这时，论坛、客服接到众多玩家反馈无法登录，报错信息："该账号已在其他设备上登录，不允许重复登录"，虚拟机无异常。

从 15：00 开始虚拟机出现异常，表现为很卡、反应慢、无法登录、无法重启。通过 VNC 的方式登录查看虚拟机有 OOM 的日志，明显为内存不足，自动 kill 掉了很多其他进程，同时，后端运行虚拟机的存储压力飙升，IOPS 突然上升到 7000 多，已经达到存储的极限，非常危险（曾经遇到在高 I/O 的情况下存储做 raid5，损坏一块磁盘，在 raid5 还没有 rebuild 完成，又坏了一块磁盘导致整个集群数据丢失的情况），当时存储的情况如图 22-14。

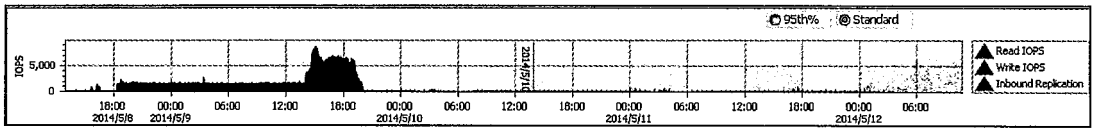


图 22-14 存储 IOPS 情况

另外，上线前预估读会比较多，但实际情况却是读 / 写比例基本在 4：6 和 5：5 之间，和预期严重不符。

根据经验，如果 I/O 延迟在 20ms 以上是不可接受的，会严重影响游戏的运行，从图 22-15 中可以看到 I/O 延迟基本都在 20ms 以上。

虚拟机运行在虚拟化集群中，具体请参考第 18 章有关 KVM 虚拟化集群技术及应用场景的介绍，以下为当时集群的环境。

（1）硬件环境

- 服务器品牌：HP C3000（刀片服务器，支持 8 片刀片服务器）。
- CPU：Intel Xeon E5-2640（2.5GHz，六核）*2。
- 硬盘：300G B*2。
- 内存：96G/ 每刀片服务器。

（2）软件环境

宿主机：centOS 6.4 64 位，内核版本 2.6.32-431.17.1.el6.x86_64，两台。

（3）网络环境

BGP 机房，集群架构（具体可以参考 18.3.2 节有关集群架构的介绍）。

2. 问题定位过程

首先，看下当时的故障报告，如表 22-3 所示。

1107114476@qq.com
dolphinebook.com

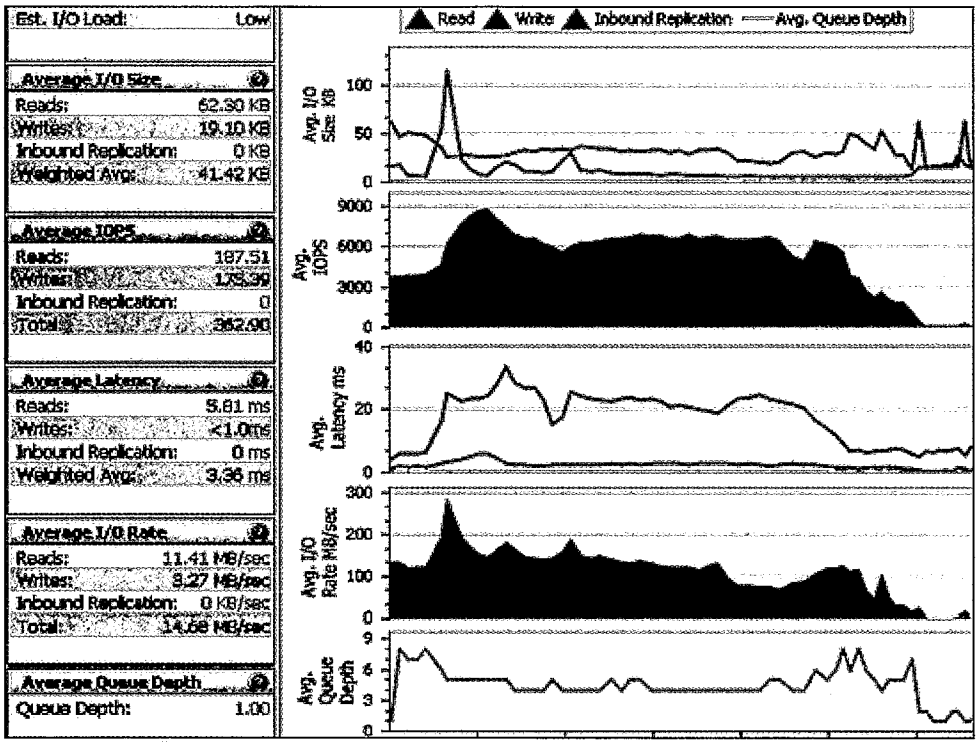


图 22-15　存储整体负载情况

表 22-3　故障报告

故障 ID	时间点	故障描述	运维团队处理过程
故障 1	13:30—14:30	众多玩家无法登录，在线人数徘徊在 30 左右，报错信息：“该账号已在其他设备上登录，不允许重复登录。”	检查服务器状态，检查相关日志 . 因为从人数曲线上看与开服前波动不大（22 ~ 45），猜测可能是某个防火墙的阻挡
故障 2	15:00—15:56	众多玩家游戏卡或者无法登录或掉线	检查发现 20 台 Web 虚拟机服务器 CPU 相继跑满，并且出现无法登录无法重启的情况 运维团队的处理过程如下： 1. 配合游戏提供方检查各个建议项，未发现问题 2. 增加 Web 服务器从 20 台到 40 台，未改善 3. 基于同属一个资源集群内的服务器并非越多越好的想法，考虑到可能会抢占资源的问题，尝试将 20 台减为 8 台 Web，卡机的情况逐步改善列一个数字也可以说明： 14:52 pcu 人数 3041，20 台服务器顶不住 17:36 pcu 人数 3150，8 台服务器非常平稳

故障 1 主要是游戏代码问题，不做分析，重点看下和虚拟化相关的故障 2。

从虚拟机的反应以及日志来看，主要是由于内存不足导致，默认提供的虚拟机都会开启虚拟内存（swap），如果虚拟机的内存不足会将内存数据存放到 swap 空间里，实际上，间

1107114476@qq.com
dolphinebook.com

接导致后端存储压力飙升，这里原因已经很明显了，这也是为什么将前端 Web 服务器从 20 台增加到 40 台没有改善，反而从 20 台减少到 8 台 Web 服务器却非常平稳的原因，存储的 IOPS 也从 7000 多降到 2800 左右。

另外，对于游戏项目组提供的说法，放在虚拟机上的都是一些 Web 业务，读的比例会比较大，不应该有大量写的请求，但是，从存储上可以看到读写比例基本在 4：6 和 5：5 之间，说明对程序和业务模型的分析没有到位，在上线前也没有测试出相关问题。

3. 问题处理方法

通过以上分析，此款游戏前端 Web 业务需要配置较高的内存以及高性能的磁盘，将写较多的业务放在带有 SSD 磁盘的物理机上，读请求较多的继续放在虚拟机上，同时将虚拟机分布在多个集群上，较好的解决了此次遇到的问题。

4. 问题处理思路总结

① 因为一个集群对应一台存储，并不是说前端 Web 的虚拟机越多越好，应提前告知游戏项目组，让其在申请虚拟机时要加一条说明（虚拟机平均分布在多个集群中）。

②.另外，对集群架构做一些改动，将两台甚至更多台存储同时加入到一个存储池，做到存储压力的负载均衡。

③ 对此款游戏没有足够多的重视，没有相关预案和专门人员去跟踪。

④ 业务上线前都需要经过严格的测试，一般虚拟化项目上线都要经过第 19 章介绍的流程，必须严格遵守，此案例就是因为没有严格的遵循流程导致的。

本节介绍了一个生产环境非常严重的事件，印象非常深刻，主要的原因就是测试的不够，对业务了解不够，希望读者在虚拟化实施中能够引以为戒。

22.4 其他生产环境问题案例

本节介绍一些在生产环境中积累的 KVM 虚拟化问题案例，有些案例曾经困扰笔者很长时间，或者造成相当严重的影响，希望这些案例能为读者提供一些借鉴，在生产环境中能够避免出现类似的问题，或者为读者在碰到类似的问题的时候提供思路。

22.4.1 共享存储集群虚拟机迁移故障案例

本节案例是笔者在生产环境正式部署的第一个虚拟化集群，集群搭建过程中及测试时碰到的问题。

1. 虚拟机不能迁

（1）虚拟机不能迁移症状现象

集群搭建完成之后，进行在线迁移测试，发现迁移不成功，具体的环境如下。

❑ 硬件环境：10 台宿主机，一台商业存储。

1107114476@qq.com
dolphinebook.com

❑ 软件环境：所有的宿主机系统为 CentOS 6.1。

❑ 问题现象：任何宿主机上的虚拟机不能在线迁移，执行迁移操作之后，提示迁移失败。

（2）虚拟机不能迁移原因分析思路

花了三四天时间，从以下几个方面进行对比测试：

❑ 详细分析系统日志、Libvirt 日志，只能看到迁移失败，没有提示具体原因。

❑ 升级宿主机系统内核，故障依旧。

❑ 切换虚拟机系统，尝试 Linux、Windows 系统的虚拟机，故障依旧。

❑ 因为迁移主要是内存迁移，给虚拟机配置不同大小的内存，故障依旧。

❑ 编译更新 Libvirt、QEMU 版本，故障依旧。

❑ 抓包查看，只能看到开始迁移，随后断开，排除是网络层的原因。

至此，已经没有更好的思路，然后在红帽、Libvirt、QEMU 等论坛，邮件组翻阅，偶然看到有人描述配置在线迁移的时候，需要能够在宿主机直接能解析主机名。修改每台宿主机的 /etc/hosts 文件，使主机名和 IP 能够解析，再尝试在线迁移，问题得到解决。

2. 虚拟机迁移过程中长达 10 秒不能 ping 通

还是这个集群，虚拟机终于能在线迁移了，但是在迁移过程中发现，虚拟机迁移成功之后，大约 10 秒后才能 ping 通。

开始以为是 arp 解析的问题，经过抓包查看，发现不是 arp 的问题。然后也是在红帽的知识库上找到原因，主要是配置网桥的时候，需要再添加一行 DELAY=0。配置后问题得到解决，虚拟机迁移的时候只丢了 1 个包。

Linux 系统网桥最大有 30 秒的延时，主要是为了防止路由环路，如果确定不会发生路由环路，可以将延时设置为 0。

后续的 CentOS 系统已经不存在这个问题了。

---

3. 虚拟机迁移后长达 1 分钟 TCP 服务不通

还是这个集群，虚拟机迁移成功后，发现网络层通信是正常的，但是 TCP 服务要等大约 1 分钟之后才能正常，反复测试，发现不同宿主机之间，不同虚拟机之间迁移，都有类似的问题，时间长短不一，最长的时候，时间差距有 2 分钟。

这个问题经过反复排查，最后定位到虚拟机的时间上，经过测试发现，TCP 服务不通的时间，就是虚拟机之间时间的差距。解决办法就是给所有的宿主机、虚拟化设置 NTP，按照 7.4 节的方法设定内核参数。

任何虚拟机，如 VMware、Xen、HyperV、KVM，都有时钟漂移的问题，主要是因为虚拟机的时钟是虚拟出来的，一般比物理机要走得快一点。随着虚拟化技术的发展，现在采用的都是半虚拟化的精确时钟，已经能比较好地解决这个问题。

1107114476@qq.com
dolphinebook.com

4. 问题总结

通过第一个生产环境集群的搭建，收获很多，Libvirt 的在线迁移需要 IP 和主机名能够解析，在此之前，一直以为迁移直接通过 IP 就可以。在以后的虚拟化集群搭建中，都做了以下 3 步的 checklist 检查：

- ❑ 主机和 IP 的解析。
- ❑ 配置网桥 DELAY 参数为 0。
- ❑ 配置 NTP。

22.4.2 宿主机异常关机后虚拟机无法启动案例

1. 症状表现描述

宿主机维护的时候，在宿主机执行 reboot 重启命令，大约有 10 分钟，宿主机一直未重启，然后强制断电重启宿主机，发现有几台虚拟机不能启动，通过 virsh start 命令启动虚拟机，提示虚拟机启动失败，命令行显示如下：

```
virsh # start win7
error: Failed to start domain win7
error: Unable to read from monitor: Connection reset by peer
```

宿主机系统为 CentOS 6.x，虚拟机系统 Windows、Linux 都有。

2. 原因及处理过程

造成这个问题主要是以下原因：

- ❑ 宿主机执行重启操作的时候，会把在线的虚拟机状态保存，以方便宿主机在启动的时候能够快速恢复虚拟机状态。
- ❑ 状态保存主要是虚拟机的内存，如果虚拟机的内存比较大，保存时间就比较长。
- ❑ 在未完成内存保存的情况下，重启宿主机，会造成虚拟机再次启动的时候，虚拟机保存的状态文件读取错误，从而虚拟机不能启动。

虚拟机的状态保存文件路径是 /var/lib/libvirt/qemu/save/，可以直接查看，命令如下：

```
ls -l /var/lib/libvirt/qemu/save/
-rw------- 1 root root 20868025 Feb 14 08:53 win7.save
```

直接删除状态保存文件，或者执行 managedsave-remove 命令，再启动虚拟机，问题解决了，命令如下：

```
virsh # managedsave-remove win7
Removed managedsave image for domain win7
virsh # start win7
Domain win7 started
```

本案例是一例常见案例，执行宿主机重启的时候，虚拟机最好能先关机，再重启宿主机，以避免出现相同的故障。

1107114476@qq.com
dolphinebook.com

22.4.3　宿主机使用 CentOS 5.6 系统问题案例

1. 问题显现及解决过程

（1）网卡驱动不定期的卡死

最早在生产环境使用 KVM 虚拟化的时候，宿主机系统版本是 CentOS 5.6，当时主流服务器的网卡都是 Broadcom 的网卡，在使用的过程中发现每过一段时间，网卡就失去响应，解决办法是重启宿主机，或者重新卸载网卡驱动模块，再加载模块就恢复正常。后来通过升级网卡驱动，问题有显著缓解，随着宿主机的系统升级到 CentOS 6，这个问题最终得到解决。

（2）Libvirt 不定时崩溃

还是 CentOS 5.6 的宿主机，经常出现 Libvirt 启动虚拟机的时候，发生崩溃现象。经过升级 Libvirt 问题得到解决。

（3）Windows 虚拟机中断占有过高

还是 CentOS 5.6 的宿主机，Windows 虚拟机响应很慢，查看 CPU 利用率不是很高，后来，使用第三方工具，发现虚拟机系统中断占有非常高。随着宿主机的系统升级到 CentOS 6，这个问题最终得到解决。

2. 总结

开始使用 KVM 的时候，主要的精力是解决稳定性问题，KVM 经过几年的发展，现在已经非常稳定，在生产环境中，推荐使用比较新的 KVM 和宿主机系统版本，往往稳定性和性能都有所提升。

22.4.4　Windows 虚拟机网络闪断、不通问题案例

1. 现象

2011 年的时候，在生产环境中，新上线了一批 Windows 虚拟机，宿主机系统为 CentOS 6.1，虚拟机系统为 Windows 2003 Server 32 位中文版。上线之后发现，经常会发生网络闪断的情况，平均每天发生 1 ~ 3 次。网卡驱动使用的 Virtio 发生闪断的时候，并不是网卡压力最大的时候，闪断时间基本上是随机分布的。

后来，尝试将部分虚拟机的网卡驱动更换为 Intel e1000，闪断情况消失。但是又发现一个问题，就是同一宿主机上，虚拟机之间不能通讯。经过多次尝试，解决办法是将虚拟机内的 Intel e1000 网卡进行升级，问题得到解决。

2. 原因分析

Intel e1000 是全虚拟化的驱动，完全由软件模拟，Virtio 是半虚拟化的驱动，让虚拟机和虚拟化层可以直接交互，提高通讯效率。根据经验，Windows 虚拟机使用 Virtio 驱动，一直有网卡闪断的情况发生，随着 Virtio 驱动的升级，情况逐步在改善。最新的 Virtio 驱动，笔者在生产环境使用，已经有两个多月没有发生问题。

1107114476@qq.com
dolphinebook.com

但是，Linux 系统使用 Virtoio 驱动一直运行稳定。Windows 系统、Linux 系统使用 Virtio 的硬盘驱动也一直工作正常。

随着 KVM 版本的升级，发现不用升级 e1000 的驱动，也不影响虚拟机直接的通讯。总的来说，随着 KVM 技术的不断完善，在生产环境的问题越来越少，越来越稳定。目前，在生产环境的 Windows 2008 R2 Server 又开始逐步使用 Virtio 网卡，已经使用有三个多月一直正常。

22.4.5 Windows 7 虚拟机只能使用 2 个 CPU 案例

1. 现象

在生产环境发现一个很奇怪的现象，无论配置几个 CPU，在 Windows 7 的系统中只能看到两个 CPU。如图 22-16 所示，配置了 4 个 CPU，在虚拟机设备管理器里面查看，处理器也是 4 个，但是，如图 22-17 所示，在任务管理器中只能看到两个，实际只能使用两个。

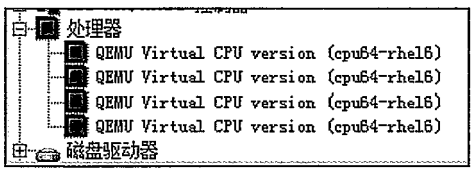


图 22-16 设备管理器处理器情况

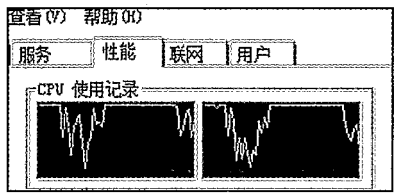


图 22-17 任务管理器只能看到 2 个 CPU

2. 原因及解决办法

通过在微软网站查找资料，发现 Windows 7 最多只支持两个 CPU socket，即两颗物理机 CPU，但是，每个物理 CPU 内可以有多个 core，如果配置给虚拟机的是模拟多颗物理 CPU，那么 Windows 7 系统就只能使用两个。

KVM 默认每个 CPU 模拟一个 socket，必须修改虚拟机 CPU 的拓扑，才能使用超过一个 CPU。

解决办法是修改为一个 socket 内有多个 Core，如图 22-18 所示。

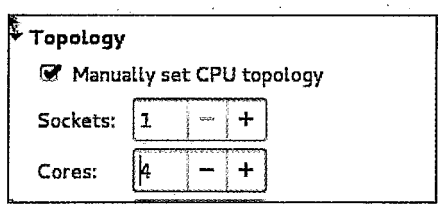


图 22-18 virt-manager CPU 配置

xml 配置文件如下：

```
<topology sockets='1' cores='4' threads='1'/>
```

问题得到解决，在虚拟机系统中可以看到 4 个 CPU，如图 22-19 所示。

22.4.6 生产环境非常奇怪的一个案例

1. 现象

生产环境一台宿主机，已经连续稳定运行几个月，突然出现 CPU 间歇的压力非常高的

1107114476@qq.com
dolphinebook.com

现象，并且非常有规律，如图 22-20 所示，每 23 分钟出现一次，每次持续将近 2 分钟。并且出现问题的时候，宿主机及宿主机上所有的虚拟机都出现 CPU 利用率非常高的情况，出现问题的时候对业务影响非常严重。

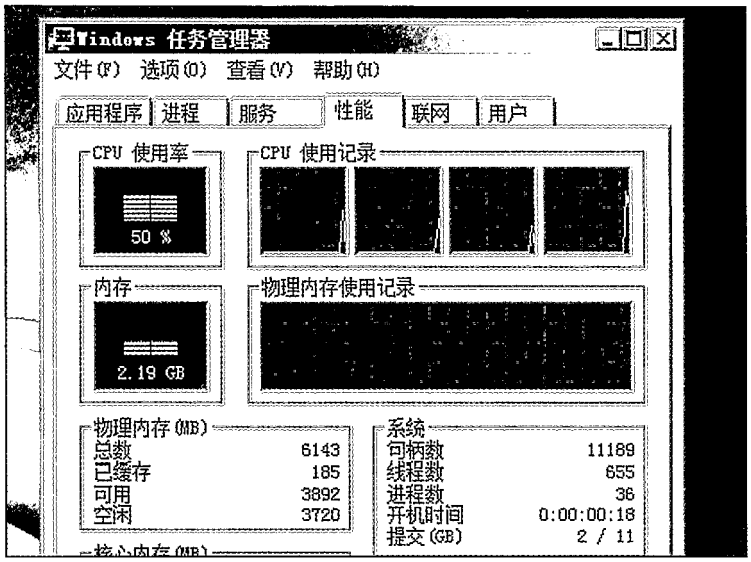


图 22-19　任务管理器可以看到 4 个 CPU

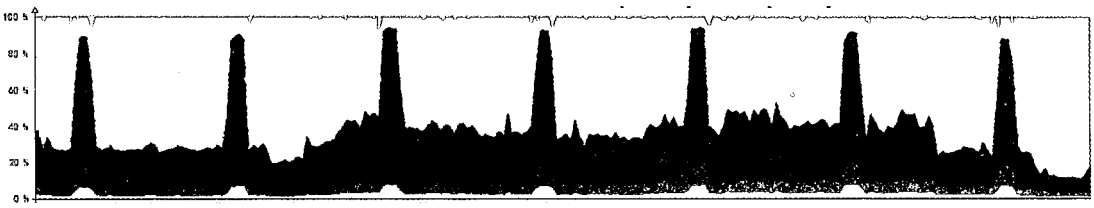


图 22-20　CPU 间歇性高压力

2. 问题解决过程

因为非常有规律，首先，怀疑业务做了定时任务，详细排查了宿主机、虚拟机的各项定时任务，未发现异常。

然后，怀疑业务是否有程序漏洞，请业务排查日志和代码，经过一天的排查，未发现问题。排查了宿主机、虚拟机各种日志，未发现异常，至此陷入困境。

问题出现的第三天，收到带外管理卡的内存报警，如图 22-21 所示。于是，怀疑内存引起的故障，将虚拟机迁移到

Temperatures	OK
Power Supplies	OK
Power Status	Redundant
Storage	OK
Network	OK
Processors	OK
Memory	Degraded
BIOS/Hardware Health	OK

图 22-21　管理卡健康情况

1107114476@qq.com
dolphinebook.com

备机，更换内存，再迁移回来，问题得到解决。

这个问题是在生产环境碰到最奇特的问题之一，从现象看，很可能是软件问题，一直在排查软件原因，直到硬件故障报警出来，才怀疑到硬件。

提示 宿主机因为运行多台虚拟机，所以，宿主机的稳定性要求非常高，新一代服务器的管理卡报警功能非常强大，充分利用管理卡的报警功能，有利于硬件问题尽早被发现。

22.4.7 U 盘在虚拟机上挂载问题案例

KVM 的 USB 设备 Passthrough 是将宿主机的 USB 接口直接给虚拟机使用，USB 接口上的设备也就直接可以在虚拟机上使用。在准备将 U 盘给虚拟机使用的时候，发现不同系统的宿主机，现象竟然不一样。

1. USB 设备 Passthrough 操作方法

生产环境如下。

- 宿主机 1：系统 CentOS 6.6 64 位，内核版本 2.6.32-504.1.3.el6。
- 宿主机 2：系统 Fedora 21，内核版本 3.17.2-300.fc21.x86_64。
- 虚拟机：Windows 7 64 位。

USB 设备 Passthrough 配置方法如下。

① 通过 lsub 命令查看有哪些 USB 口可供使用，命令如下：

```
lsusb
...
Bus 006 Device 002: ID 03f0:1027 Hewlett-Packard Virtual keyboard and mouse
Bus 001 Device 002: ID 058f:6387 Alcor Micro Corp. Flash Drive
Tue Dec 16 12:45:21 CST 2014
```

或者也可以通过 virt-manager 查看，如图 22-22 所示。

② 将需要的 USB 接口分配给虚拟机单独使用，如图 22-23 所示。

xml 配置文件如下：

```
<hostdev mode='subsystem' type='usb' managed='yes'>
      <source>
        <vendor id='0x058f'/>
        <product id='0x6387'/>
        <address bus='1' device='2'/>
      </source>
      <alias name='hostdev0'/>
    </hostdev>
```

2. 问题现象

在测试过程中，发现一个很奇怪的现象，在 CentOS 6 的宿主机上的 Windows7 中，U 盘

1107114476@qq.com
dolphinebook.com

不能识别，如图 22-24 所示。

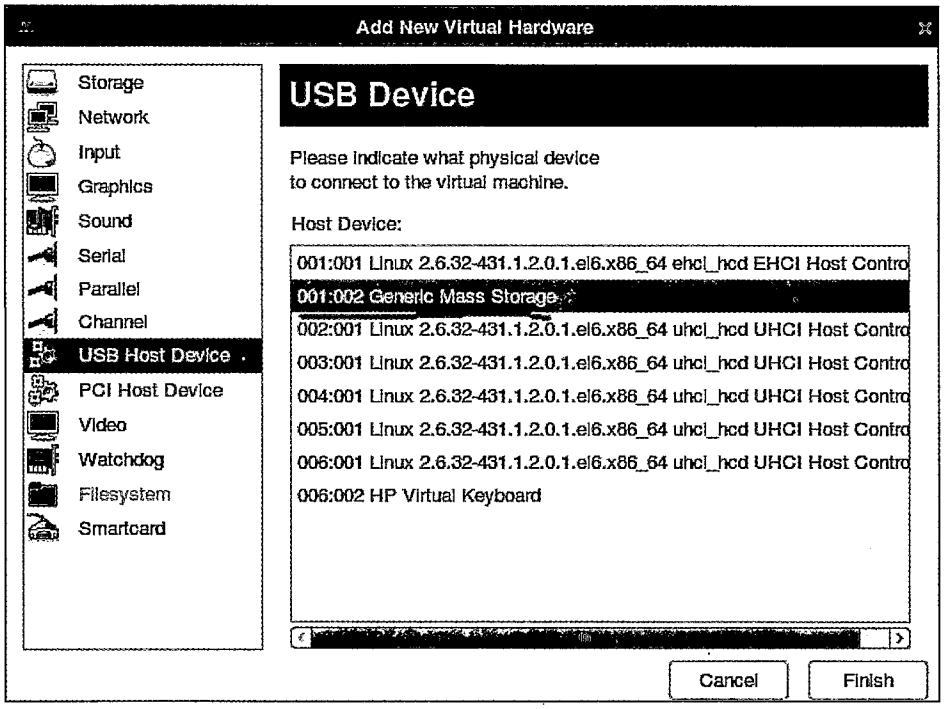


图 22-22　通过 virt-manager 查看可用 USB 端口

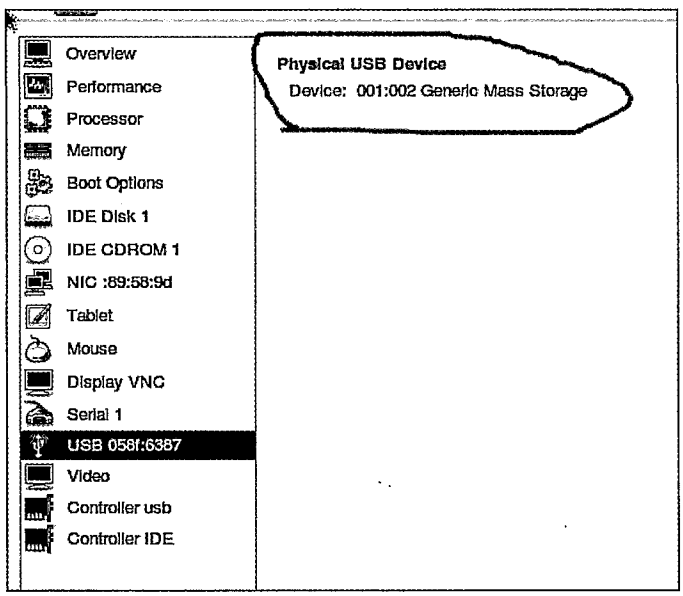


图 22-23　通过 virt-manager 分配 USB 端口给虚拟机

1107114476@qq.com
dolphinebook.com

但是，如果是 CentOS 6 的虚拟机，U 盘就可以正常使用。开始以为是 Windows 7 驱动的问题，但是尝试了各种驱动，问题依旧。

后来，尝试了在 Fedora21 上的 Windows 7 虚拟机，U 盘插上去就可以直接使用，如图 22-25 所示。

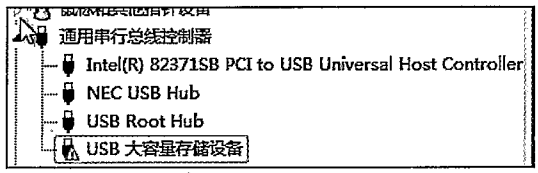


图 22-24 Windows 系统 USB 设备出现黄色感叹号

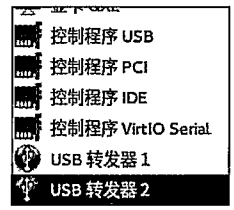


图 22-25 Fedora 21 上 U 盘可直接识别

甚至不要做任何配置，应该和 Fedora 上的 QEMU 版本有关系，新的版本做了改进。读者如果有需要给虚拟机接上 U 盘，请注意这个问题。

22.4.8 HP ILO4 固件 Bug 引起宿主机重启案例

在生产环境发生一次宿主机异常重启，经过排查发现是宿主机内核崩溃，引起宿主机重启。宿主机品牌是 HP GEN8 380，系统是 CentOS 6.6，默认开启了内核自动转存。

内核崩溃时日志文件和转存的内存文件是放在 /var/crash 目录下的，有个精确到秒的以崩溃时间命名的目录，目录下有两个文件，一个是崩溃时内存的一个转存，另一个是文本的日志文件。

查看日志文件，可以清楚地看到内核崩溃时的情况，显示如下：

```
<0>Kernel panic - not syncing: An NMI occurred. Depending on your system the
reason for the NMI is logged in any one of the following resources:
<0>1. Integrated Management Log (IML)
    <0>2. OA Syslog
<0>3. OA Forward Progress Log
<0>4. iLO Event Log
<4>Pid: 0, comm: swapper Not tainted 2.6.32-504.el6.x86_64 #1
<4>Clocksource tsc unstable (delta = -34359715026 ns).  Enable clocksource
failover by adding clocksource_failover kernel parameter.
```

从日志可以清楚看到，主要是 NMI 发生，造成内核崩溃。

提示 NMI 是系统中标准的中断屏蔽技术中不能被忽略的硬件中断，通常是不可恢复的硬件错误信号。

通过查阅资料，发现 HP GEN8 在 Linux、VMWare、Windows 上都有类似的问题，在 HP 官网文档讲得很清楚，链接如下：

1107114476@qq.com
dolphinebook.com

http://h20566.www2.hp.com/hpsc/doc/public/display?sp4ts.oid=5268290&docId=emr_na-c04332584-3&docLocale=en_US。

HP GEN8 ILO4 固件版本 1.30、1.32、1.40、1.50 的时候会触发 NMI 问题，将 ILO 固件刷新到最新的 2.04 时，问题得到解决。

提示　应定期关注宿主机硬件厂商固件更新情况，如果有重要固件更新，应及时更新。

22.4.9　一次机房断电引起的问题案例

某机房断电 40 分钟，电力恢复后，发现宿主机和虚拟机故障，解决过程记录如下。

1. 机房断电引起的宿主机问题

发现所有 IBM 3650M4 服务器的宿主机不能正常启动，通过带外管理卡的远程桌面重定向，看到如图 22-26 所示的提示。

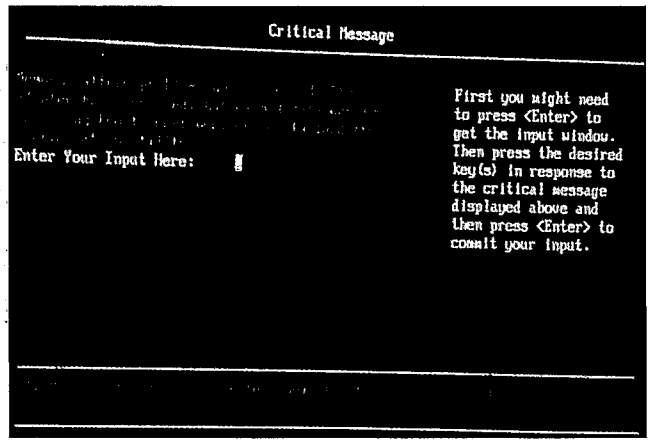


图 22-26　IBM 服务器故障

重启服务器，现象一样，仔细查看说明，发现提示的信息是，“RAID 卡电池用电耗尽，缓存的数据丢失。”

按照提示操作如下：

按 Enter 键，然后输入 desired，再按 Enter 键，然后按 Esc 键退出，系统正常启动。

那么如何避免再出现类似的情况呢，如果不解决，下次服务器突然断电，系统重启后还是会卡在这个界面。通过查阅资料，发现有办法可以绕过提示，操作方法如下。

① 在系统下安装 RAID 卡的管理工具，安装程序可以在 Lsi 官网下载，命令如下：

```
# 安装 MegaCli
unzip ibm_utl_sraidmr_megacli-8.00.48_linux_32-64.zip
cd linux/
```

1107114476@qq.com
dolphinebook.com

```
rpm -ivh Lib_Utils-1.00-09.noarch.rpm MegaCli-8.00.48-1.i386.rpm
```

② 查看 RAID 卡设置，断电后是否提示如下：

```
/opt/MegaRAID/MegaCli /MegaCli64 -AdpBios -dsply -a0
BIOS on Adapter 0 is Enabled.
BIOS will Stop on error.
#RAID 卡会提示
```

③ 修改 RAID 设置，命令如下：

```
/opt/MegaRAID/MegaCli /MegaCli64 -AdpBios -BE -a0
BIOS is set to Bypass Error on Adapter 0.
```

2. 机房断电引起的虚拟机问题

宿主机重启系统后发现几台虚拟机无法连接，通过 VNC 登录后看到如图 22-27 所示的提示。

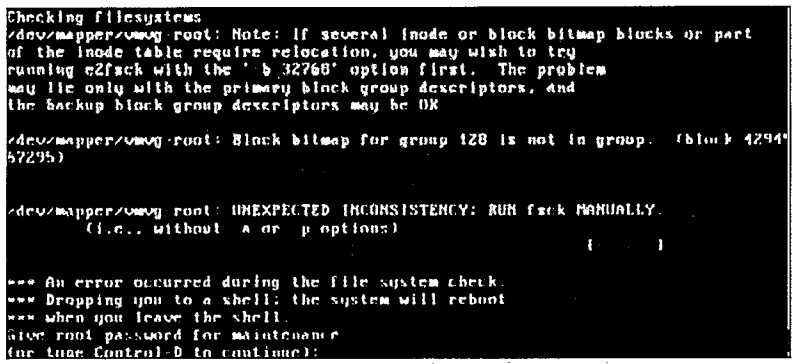


图 22-27 虚拟机启动报错

引起这种问题是由于突然断电造成虚拟机文件系统不一致，解决方法是加载并安装镜像进入救援模式，进行文件系统检查。

编辑虚拟机 xml，加载 iso 光驱，命令如下：

```
<disk type='file' device='cdrom'>
        <driver name='qemu' type='raw' cache='none'/>
        <source file='/home/CentOS-6.3.x86_64-DVD.iso'/>
        <target dev='hdb' bus='ide'/>
        <readonly/>
</disk>
```

编辑虚拟机的启动方式，在 os 的标签处添加 cdrom，并放在第一项，命令如下：

```
<os>
    <boot dev='cdrom'/>
    <boot dev='hd'/>
</os>
```

1107114476@qq.com
dolphinebook.com

激活 vg，强制检查文件系统，命令如下：

```
vgchange -ay centos6-vg    # 激活 LV
fsck -f /dev/centos6-vg/root    # 检查文件系统
reboot #重启系统，系统启动正常
```

提示　本案例记录的问题，尤其是虚拟机断电不能启动的问题，通过加载 iso 镜像修复是一个常规解决方法。

22.4.10　CPU 绑定问题案例

这个问题曾经也困扰了笔者一段时间，当时因为虚拟机压力大，配置在线虚拟机的 CPU 绑定，强制 vCPU 和物理机 CPU 一对一的绑定，命令如下：

```
virsh  vcpupin 21 0 28
virsh  vcpupin 21 1 29
virsh  vcpupin 21 2 30
virsh  vcpupin 21 3 31
```

查看一下，并没有产生效果，提示如下：

```
virsh  vcpuinfo 21
VCPU: 0
CPU: 31
State: running
CPU time: 10559.2s
CPU Affinity: ----------------------------yyyyyy
...
```

查看 xml 文件，配置是写到 xml 文件了，提示如下：

```
virsh # dumpxml 21
...
  <cputune>
    <vcpupin vcpu='0' cpuset='28'/>
    <vcpupin vcpu='1' cpuset='29'/>
    <vcpupin vcpu='2' cpuset='30'/>
    <vcpupin vcpu='3' cpuset='31'/>
    <emulatorpin cpuset='26-31'/>
  </cputune>
```

然后将虚拟机重启，可以看到配置生效了，提示如下：

```
virsh  vcpuinfo 22
VCPU: 0
CPU: 28
State: running
CPU time: 1.8s
CPU Affinity: ----------------------------y---
...
```

1107114476@qq.com
dolphinebook.com

但是，跑几分钟后，发现又不按照配置的对应了，提示如下：

```
virsh  vcpuinfo 22
VCPU: 0
CPU: 8
State: running
CPU time: 81.4s
CPU Affinity: --------yyyyyyyy--------yyyyyyyy
...
```

然后，升级了一下宿主机系统，命令如下：

```
yum update -y
```

再去做 vcpupin 的绑定，观察了 4 个多小时，发现是一直生效的。

笔者还在生产环境碰到过，使用 CGroup 做 CPU 资源限制，造成宿主机 CPU 利用率很高的情况，也是通过升级系统解决问题。读者碰到类似的情况，可以借鉴。

22.4.11 生产环境存储故障导致 50 个虚拟机丢失案例

这个案例是笔者在生产环境碰到的最严重的事故之一，直接导致 50 多个虚拟机丢失。

（1）案例环境

- 5 台计算节点服务器。
- 一台商业存储，12 块 1T 的磁盘，11 块配置 RAID5，另外一块为热备盘。

（2）故障的过程

- 一块磁盘故障。
- 按照 RAID5 的机制，热备盘加入到阵列组，并开始重建，这时候就有大量的磁盘读 / 写。
- 不到 20 分钟，另外一块磁盘故障，重建没有完成。RAID5 的机制同时只能允许一块磁盘故障，这时候已经有两块磁盘故障，所以数据丢失。

（3）故障处理办法

- 因为所有虚拟机配置文件做了异步备份，按照配置文件重建新的虚拟机。
- 联系各个业务重新部署业务，恢复业务数据，整个过程持续了 30 多小时。

（4）故障影响

- 大部分业务中断 20 ~ 30 小时；
- 部分业务数据丢失 5 ~ 20 小时，部分业务数据丢失 3 个月。

（5）总结反思及应对方案

应该考虑各种极端情况，做好如下监控：

- 所有线上存储，磁盘监控 SMART 信息，有 10 个以上的坏块就报警。
- 增加备份存储，并配置异步备份，备份级别做到 5 分钟。
- 生产环境所有使用 RAID 的存储，全部在维护时间修改为 RAID10 或者 RAID6。
- 提醒业务做好数据备份。

1107114476@qq.com
dolphinebook.com

22.4.12　生产环境 Ubuntu 虚拟机故障问题案例

这两个案例是在生产环境实施一款游戏虚拟化碰到的问题，虚拟机系统为 Ubuntu12.04，使用这个版本主要是因为游戏在这个版本开发，要求必须使用这个版本。

1. 案例一：虚拟机重启后卡住

（1）问题现象

制作镜像的时候，虚拟机分区使用的是自动分区，自动分区使用的 lvm 分区格式。虚拟机镜像制作完成后，将业务程序部署上去测试，运行很正常。决定在生产环境部署测试，测试的时候发现一个问题，虚拟机重启的时候，经常会检查文件系统，并且检查的时候会卡住，如图 22-28 所示，开始的时候以为是虚拟机非正常关机造成的，但是，测试了几次从虚拟机内部正常关机，或者直接从虚拟机内部重启虚拟机，依然有这样的现象，并且几率非常高。

```
Begin: Loading essential drivers ... done.
Begin: Running /scripts/init-premount ... done.
Begin: Mounting root file system ... Begin: Running /scripts/local-top ... done.
Begin: Running /scripts/local-premount ... done.
[    2.009371] EXT4-fs (vda2): INFO: recovery required on readonly filesystem
[    2.009453] EXT4-fs (vda2): write access will be enabled during recovery
[    2.013567] EXT4-fs (vda2): recovery complete
[    2.013789] EXT4-fs (vda2): mounted filesystem with ordered data mode. Opts: (null)
Begin: Running /scripts/local-bottom ... done.
done.
Begin: Running /scripts/init-bottom ... done.
[    2.461996] init: ureadahead main process (360) terminated with status 5
fsck from util-linux 2.20.1
fsck from util-linux 2.20.1
/dev/vda2: clean, 56638/3252224 files, 491610/12984832 blocks
/dev/vda1: recovering journal
/dev/vda1: clean, 227/121920 files, 59960/487424 blocks
[    2.601391] EXT4-fs (vda2): re-mounted. Opts: errors=remount-ro
[    2.719099] EXT4-fs (vda1): mounted filesystem with ordered data mode. Opts: (null)
 * Starting system logging daemon
 * Starting configure network device security
 * Starting Mount network filesystems
 * Starting Failsafe Boot Delay
 * Stopping Mount network filesystems
 * Stopping cold plug devices
 * Stopping log initial device creation
 * Starting Bridge socket events into upstart
 * Starting enable remaining boot-time encrypted block devices
 * Starting configure network device security
 * Starting save udev log and update rules
 * Stopping save udev log and update rules
 * Starting configure virtual network devices
 * Stopping configure virtual network devices
 * Stopping OpenSSH server
 * Starting configure network device
 * Stopping enable remaining boot-time encrypted block devices
 * Starting OpenSSH server
 * Starting Mount network filesystems
 * Stopping Mount network filesystems
 * Stopping Failsafe Boot Delay
 * Starting System V initialisation compatibility
 * Starting set sysctls from /etc/sysctl.conf
 * Starting configure network device
 * Stopping set sysctls from /etc/sysctl.conf
Skipping profile in /etc/apparmor.d/disable: usr.sbin.rsyslogd
 * Starting AppArmor profiles
 * Stopping System V initialisation compatibility
 * Starting System V runlevel compatibility
 * Starting save kernel messages
 * Starting automatic crash report generation
 * Starting ACPI daemon
 * Starting regular background program processing daemon
 * Starting deferred execution scheduler
 * Starting CPU interrupts balancing daemon
 * Starting crash report submission daemon
 * Stopping save kernel messages
 * Stopping System V runlevel compatibility
```

图 22-28　虚拟机内核崩溃

1107114476@qq.com
dolphinebook.com

（2）问题解决过程

尝试进入单用模式，然后手工检查文件系统，第二次重启虚拟机就能启动，但是再重启就又卡住了。进入系统后，检查系统日志，未发现异常。开始怀疑是系统安装的问题，反复重装系统数次，问题依旧。后来又怀疑是安装镜像的问题，从官网下载镜像，验证 MD5 进行安装，问题依旧。最后，怀疑是分区的问题，测试不使用 lvm 分区，使用手工分区，vda1 为 /boot 分区 200MB，vda2 为 swap 分区 4GB，vda3 为根分区，使用剩余空间，如图 22-29 所示。经过 20 次重启，基本上只会发生一次检查文件系统的问题，问题解决。

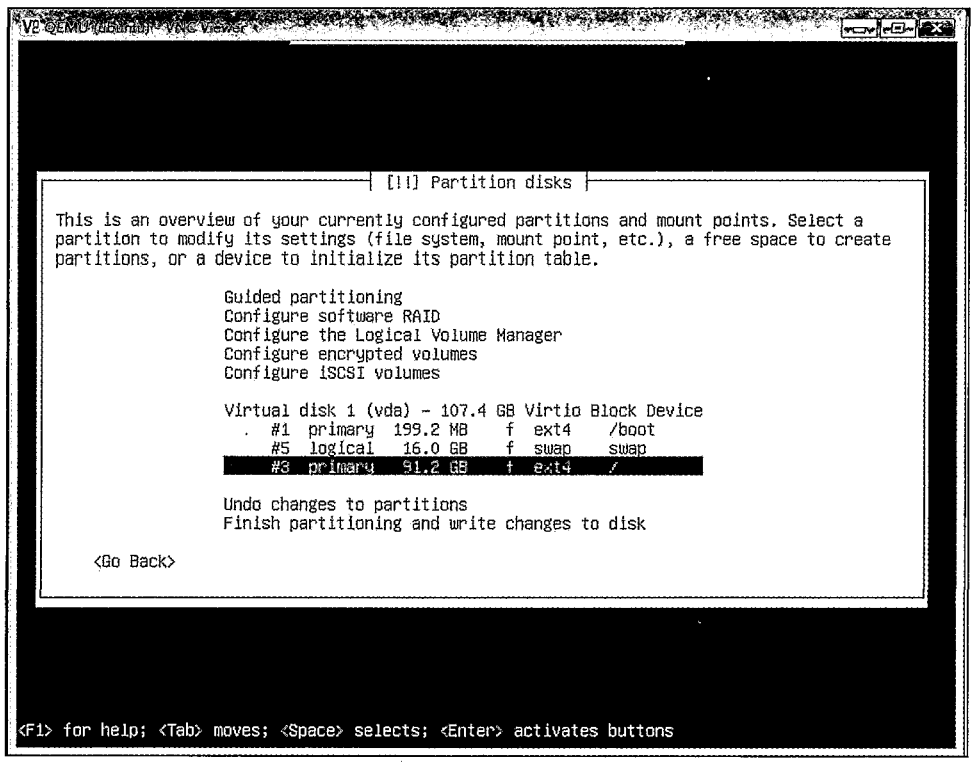


图 22-29 虚拟机磁盘分区

2. 案例二：虚拟机内核频繁崩溃

（1）问题现象与 Ubuntu 内核崩溃自动转存及重启配置

还是这个项目，在生产环境逐步开始实施，业务正式上线后，发现一个问题，几乎每台虚拟机每个月会发生内核崩溃一次。因为有几台虚拟机内存压力很大，开始怀疑是内存原因，调整了内存，问题依旧。实际虚拟机的 CPU、网络、磁盘压力均不大，于是，决定查看系统的内核崩溃日志，但是默认 Ubuntu12.04 没有配置内核崩溃自动重启及转存，发生内核崩溃的时候，没有 core dump 文件去分析，为了方便查找内核崩溃原因，需要将内核崩溃自

1107114476@qq.com
dolphinebook.com

动重启配置及内核转存配置起来，配置步骤如下：

1）配置内核崩溃自动重启。配置 /etc/sysctl.conf 文件，添加 kernel.panic 到内核参数，10 为内核崩溃 10 秒后，自动重启系统，命令显示如下：

```
vi /etc/sysctl.conf
kernel.panic = 10
```

2）验证自动重启机制是否生效。验证自动重启机制，需要配置 sysrq，sysrq 是专门用于内核调试的一种机制，打开这个参数，就可以模拟内核崩溃，添加 kernel.sysrq 到内核参数，参数为 1 时生效，命令显示如下：

```
vi /etc/sysctl.conf
kernel.sysrq = 1
```

运行 sysctl-p 命令，使配置的参数生效，或者重启系统，命令如下：

```
sysctl -p /etc/sysctl.conf
```

检查配置的参数是否生效，检查内核崩溃重启参数配置，如图 22-30，查看参数配置是否为 10。

检查 sysrq 配置，如图 22-31，查看配置是否为 1。

图 22-30　内核 panic 参数配置查看

图 22-31　内核 sysrq 参数配置

模拟系统内核崩溃，同时按 Alt+SysRq+C 组合键，或者运行如下命令：

```
echo c  >/proc/sysrq
```

可以看到内核崩溃，并读秒重启，如图 22-32，说明配置生效。

3）配置内核转存。新装的系统需要升级下，否则，不能通过 apt-get 安装软件，命令行显示如下：

```
apt-get update
```

安装内核转存组件，命令如下：

```
sudo apt-get install linux-crashdump
```

查看 /proc/cmdline 内存，检查内核转存是否生效，如果有 crashkernel 关键字，如图 22-32 所示，说明生效。

测试，模拟系统内核崩溃，同时按 Alt+SysRq+C 三个键，或者运行如下命令：

```
echo c  >/proc/sysrq
```

1107114476@qq.com
dolphinebook.com

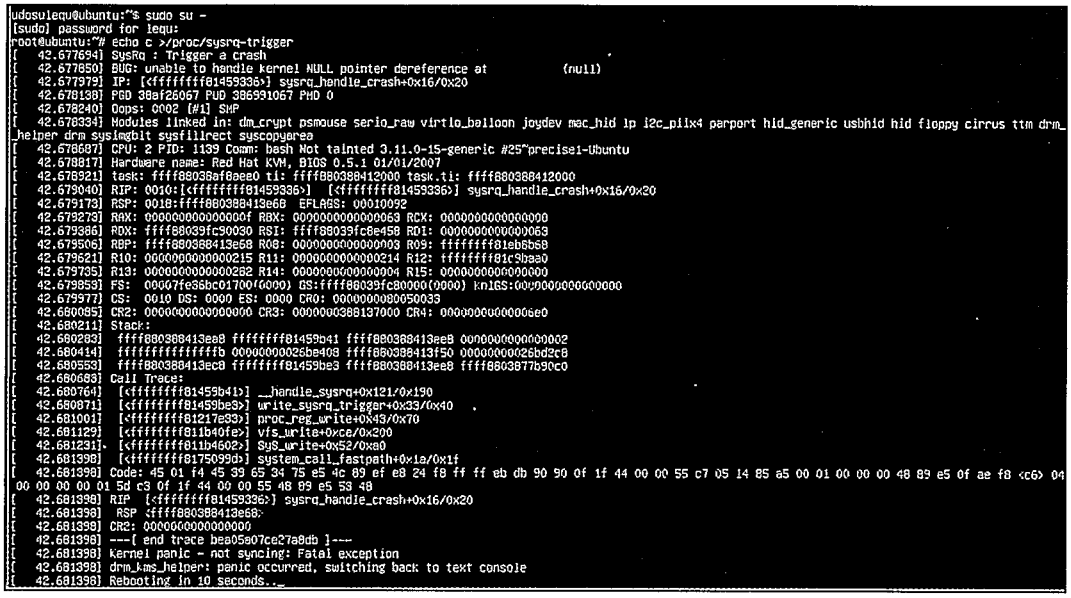


图 22-32 内核崩溃并计秒重启

图 22-33 内核转存配置

但是，发现系统崩溃，并卡住，没有发送转存，也没有重启，如图 22-34 所示。

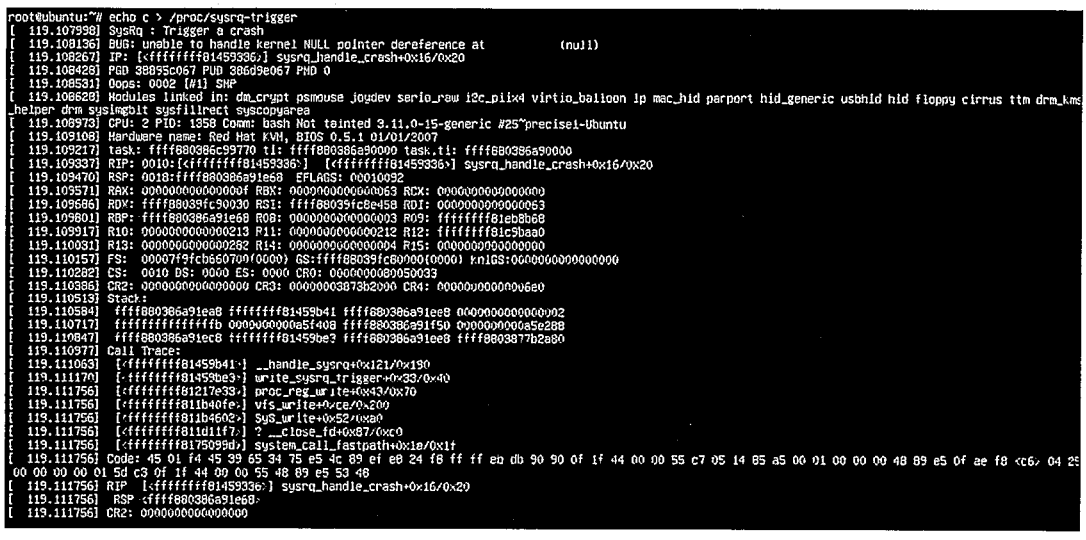


图 22-34 内核发生崩溃

1107114476@qq.com
dolphinebook.com

通过查找资料，发生这样的情况，可能和 core dump 内存配置不够有关系，于是，修改 core dum 内存配置，编辑 /etc/grub.d 下的 10_linux 文件。

修改内核转存最大文件大小为 512MB，如图 22-35。

```
# add crashkernel option if we have the required tools
if [ -x "/usr/bin/makedumpfile" ] && [ -x "/sbin/kexec" ]; then
    GRUB_CMDLINE_EXTRA="$GRUB_CMDLINE_EXTRA crashkernel=384M-2G:64M,2G-:512M"
fi
```

图 22-35　内核转存内存大小配置

重新使用 grub-mkconfig 命令生成 grub.cfg，如图 22-36 所示。

```
root@ubuntu:/etc/grub.d# grub-mkconfig -o /boot/grub/grub.cfg
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-3.11.0-15-generic
Found initrd image: /boot/initrd.img-3.11.0-15-generic
Found memtest86+ image: /memtest86+.bin
done
```

图 22-36　重新生成 grub.cfg 文件

再测试，成功，如图 22-37 所示。

```
Begin: Running /scripts/local-bottom ... done.
done.
Begin: Running /scripts/init-bottom ... [    2.072703] EXT4-fs (vda1): INFO: recovery required on readonly filesystem
[    2.072763] EXT4-fs (vda1): write access will be enabled during recovery
[    2.077740] EXT4-fs (vda1): recovery complete
[    2.078285] EXT4-fs (vda1): mounted filesystem with ordered data mode. Opts: (null)
Begin: Saving vmcore from kernel crash ... [    2.101173] EXT4-fs (vda2): re-mounted. Opts: data=ordered
```

图 22-37　内核转存生效

于是，线上所有的虚拟机都配置了内核崩溃自动重启及转存，但是，奇怪的是，发生内核崩溃的时候，依旧卡住，并没有转存内存及自动重启。

（2）问题解决过程

从现象看起来很有可能是虚拟机内核和宿主机兼容性的问题，宿主机系统是 CentOS 6.5，先尝试将宿主机系统升级到 CentOS 6.6，内核升级到最新，问题依旧。接下来就剩下改造虚拟机的内核了。经过咨询相同版本的生产环境，及对比同一个项目的物理机生产环境，大部分使用的是 3.8 的内核，并且工作稳定，于是决定将虚拟机内核更换为 3.8。在维护时将内核更换为 3.8 后，经过两个月测试，再没有出现内核崩溃问题，故障解决。

（3）问题总结

大部分生产环境使用的是 CentOS 或者 Windows 系统，长久以来笔者养成的习惯就是上线虚拟机的时候，将虚拟机系统升级到最新，因为最新系统会做许多优化和改进。Ubuntu 这个系统制作镜像的时候，笔者也是将内核和系统升级到最新，结果发生了内核兼容性问题。导致发生这个问题的原因应该是 Ubuntu 系统比较激进，升级过于频繁又没有充分测试。所

1107114476@qq.com
dolphinebook.com

以，生产环境使用 Ubuntu，应尽量使用长支持版本及内核。

22.5 本章小结

本章介绍了笔者实施虚拟化以来碰到几个典型问题案例，这些案例共同的特点是都给笔者带来深刻的印象，问题的解决消耗了笔者大量的时间和精力，希望这些案例的分享能够使读者在虚拟化实践时碰到类似的问题有所提示，也欢迎读者在虚拟化实践中碰到问题，通过笔者的微信订阅号“KVM 虚拟化实践”一起讨论。

1107114476@qq.com
dolphinebook.com

附录 Appendix

笔者推荐的 KVM 学习资料

1. 以下资料建议读者熟练掌握，反复研究

① RedHat 官网资料，虚拟化相关章节，链接如下：

```
https:// access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/
```

虚拟化相关的章节建议熟练掌握，在 RHEL 有新版本发布的时候，也建议阅读下新特性。

《Virtualization Administration Guide》(虚拟化管理指南)

《Virtualization Getting Started Guide》(虚拟机化起步指南)

《Virtualization Host Configuration and Guest Installation Guide》(虚拟化宿主机和虚拟机安装指南)

《Virtualization Security Guide》(虚拟化安全指南)

《Virtualization Tuning and Optimization Guide》(虚拟化性能调优)

② Libvirt 官网官方文档。Libvirt 是 kvm 虚拟化的管理工具，官方文档非常详细，链接如下：

```
http:// www.libvirt.org/docs.html
```

③ guestfish 官网官方文档。guestfish 是一套 kvm 虚拟化的镜像管理工具，可以对镜像进行非常方便的管理，链接地址如下：

```
http:// libguestfs.org
```

2. 参考阅读文档

① KVM 官网文档，链接如下：

1107114476@qq.com
dolphinebook.com

```
http://www.linux-kvm.org/page/Main_Page
```

② QEMU 官网文档，链接如下：

```
http://www.qemu.org
```

3. KVM 相关软件的更新新特性发布页面

① Libvirt 更新页面链接如下：

```
http://www.libvirt.org/news.html
```

② QEMU 更新页面链接如下：

```
http://wiki.qemu.org/Download
```

③ Fedora 新版本特性链接如下：

```
https://fedoraproject.org/wiki/Features
```

4. 两个非常不错的网站

① 虚拟机技术，链接如下：

```
http://virtualizationreview.com/home.aspx
```

② IBM developerworks 虚拟化专题，链接如下：

```
http://www.ibm.com/developerworks/cn/linux/theme/virtualization/index.html
```

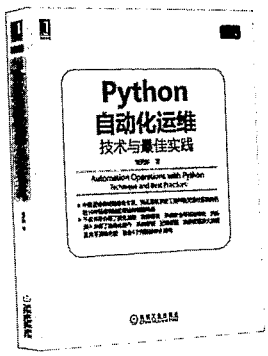
5. 虚拟化管理软件

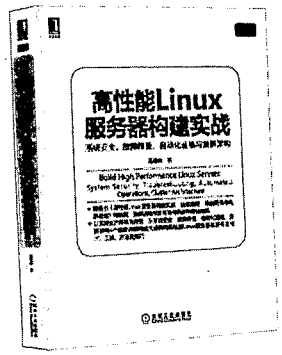
虚拟化管理软件有如下几种。

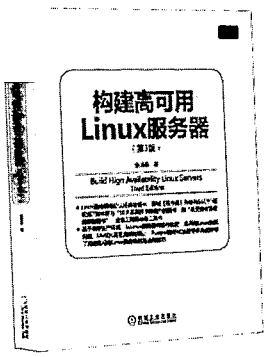
- ❑ PVE，其官网为 http://pve.proxmox.com。
- ❑ oVirt，其官网为 http://www.ovirt.org，中国社区 http://ovirt-china.org。
- ❑ Cloudstack，其官网为 http://cloudstack.apache.org。
- ❑ OpenStack，其官网为 www.openstack.org，中国社区 www.openstack.cn。
- ❑ Convirt，其官网为 www.convirture.com。
- ❑ Puppet 虚拟化组件，其官网为 https://github.com/carlasouza/puppet-virt。

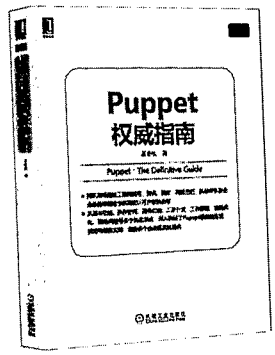
1107114476@qq.com
dolphinebook.com

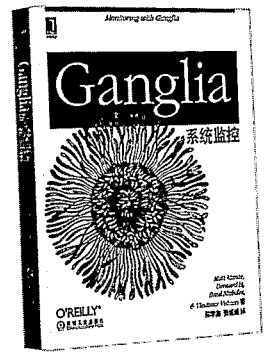
推荐阅读

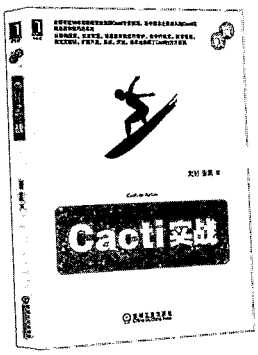






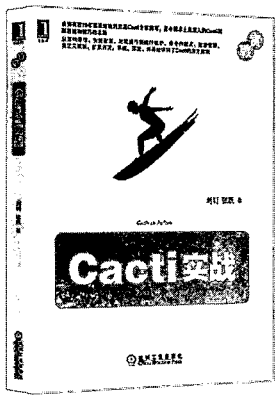


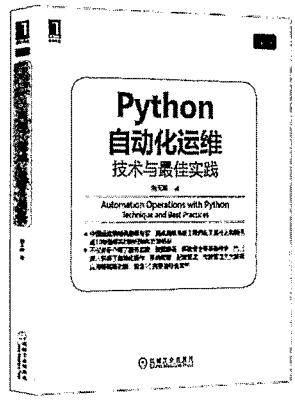


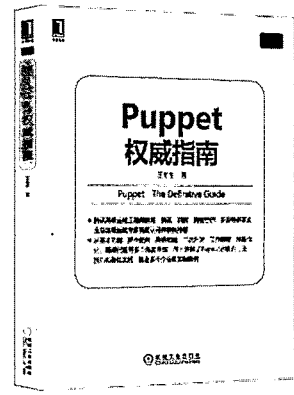


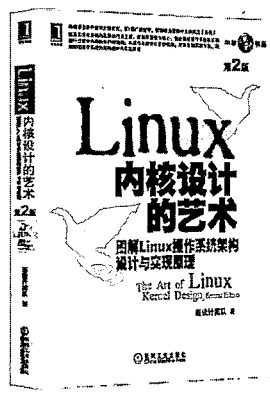
1107114476@qq.com
dolphinebook.com

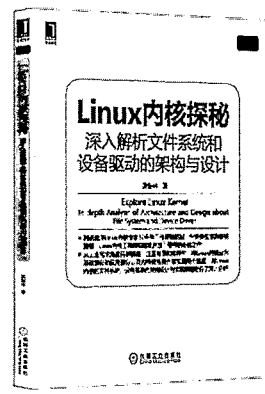
推荐阅读











Cacti实战

作者：刘钊 等 ISBN：978-7-111-50187-9 定价：69.00元

Python自动化运维：技术与最佳实践

作者：刘天斯 ISBN：978-7-111-48306-9 定价：69.00元

Puppet权威指南

作者：王冬生 ISBN：978-7-111-48598-8 定价：69.00元

Linux内核精髓：精通Linux内核必会的75个绝技

作者：Hirokazu Takahashi 等 ISBN：978-7-111-41049-2 定价：79.00元

Linux内核设计的艺术：图解Linux操作系统架构设计与实现原理（第2版）

作者：新设计团队 ISBN：978-7-111-42176-4 定价：89.00元

Linux内核探秘：深入解析文件系统和设备驱动的架构与设计

作者：高剑林 ISBN：978-7-111-44585-2 定价：59.00元

1107114476@qq.com
dolphinebook.com