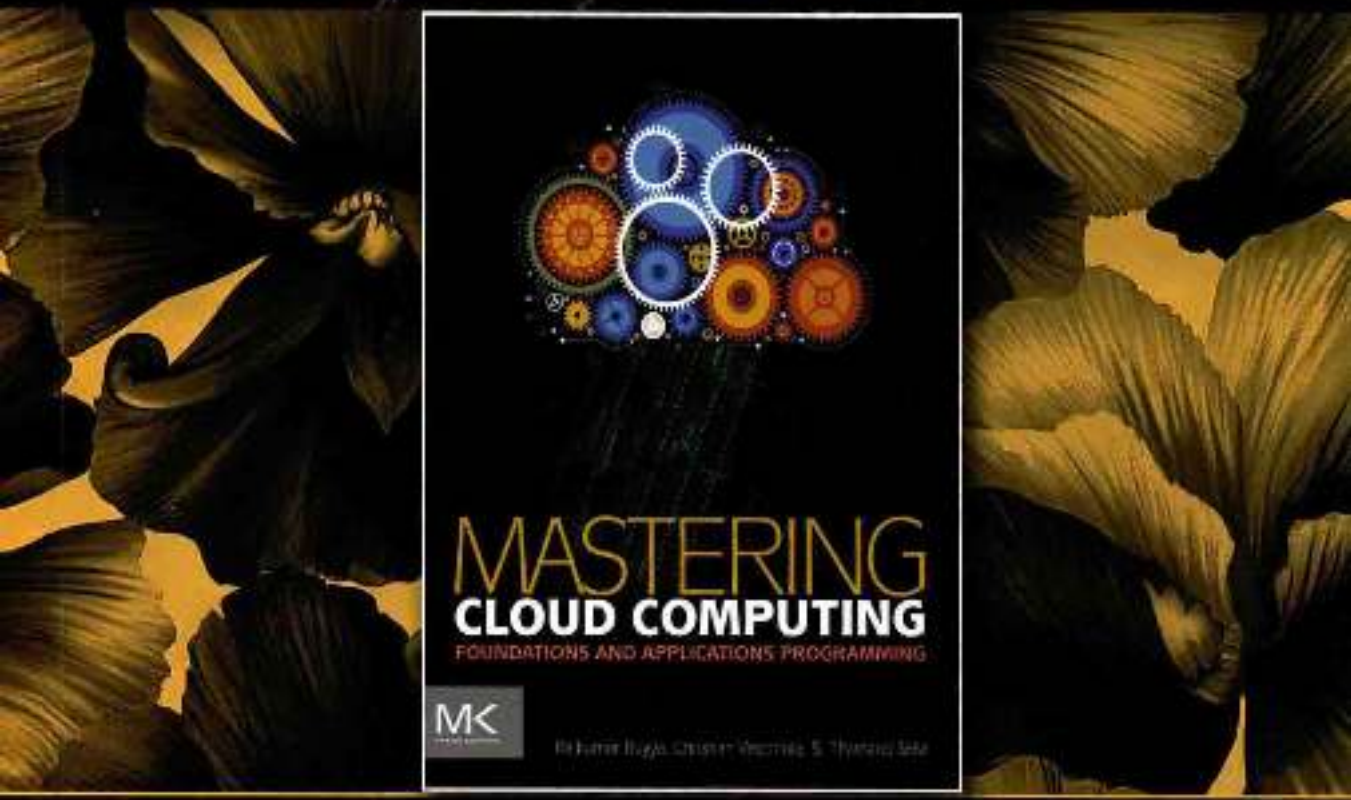
计 算 机 科 学 丛 书

深入理解云计算

基本原理和应用程序编程技术

[澳] 拉库马·布亚（Rajkumar Buyya）
克里斯坦·维奇拉（Christian Vecchiola） 著
[印] S. 泰马莱·赛尔维（S. Thamarai Selvi）
刘丽 米振强 熊曾刚 译

Mastering Cloud Computing
Foundations and Applications Programming



机械工业出版社
China Machine Press

深入理解云计算 基本原理和应用程序编程技术

Mastering Cloud Computing Foundations and Applications Programming

Buyya等人将我们带上云计算的旅途，一路从理论到实践，从历史到未来，从计算密集型应用到数据密集型应用，激发我们产生学术研究灵感，并指导我们掌握工业实践方法。从虚拟化和编程理论基础，到云计算在从图表达到客户关系管理中的应用，都进行了深入的探索。

—— Dejan Milojicic，HP实验室，2014年IEEE计算机学会主席

本书介绍云计算基本原理和云应用开发方法。未来的应用开发将不再依赖于单一计算机，而是在云数据中心的一台或多台虚拟服务器上进行，并且可以在任何时间、从任何地点访问。未来的开发者必须掌握云计算技术，包括并行编程、高性能计算和数据密集型系统。本书提供与这些技术密切相关的实例、练习以及Aneka云平台实验环境。

本书特点

- 详细解析虚拟化云计算环境中应用程序的设计和实现方法。
- 提供实验和测试的真实云系统环境——Aneka云平台。
- 展示丰富的云应用实例，涉及科学、商业、能效等众多方面。
- 配套网站（www.buyya.com/MasteringClouds/）提供多种免费教辅资源。

作者简介



拉库马·布亚（Rajkumar Buyya）博士，澳大利亚墨尔本大学云计算与分布式系统实验室负责人，Manjrasoft公司创始人及首任CEO。



克里斯坦·维奇拉（Christian Vecchiola）博士，IBM澳大利亚研究院成员，澳大利亚墨尔本大学工程学院研究员。



S. 泰马莱·赛尔维（S. Thamarai Selvi）博士，印度安那大学金奈分校Madras技术学院院长。

本书译自原版*Mastering Cloud Computing: Foundations and Applications Programming*并由Elsevier授权出版

ELSEVIER 爱思唯尔

ELSEVIER

投稿热线：(010) 88379604
购书热线：(010) 88378991 88361066
读者热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

上架指导：计算机\云计算

ISBN 978-7-111-49658-8

9 787111 496588 >

定价：69.00元

计 算 机 科 学 丛 书

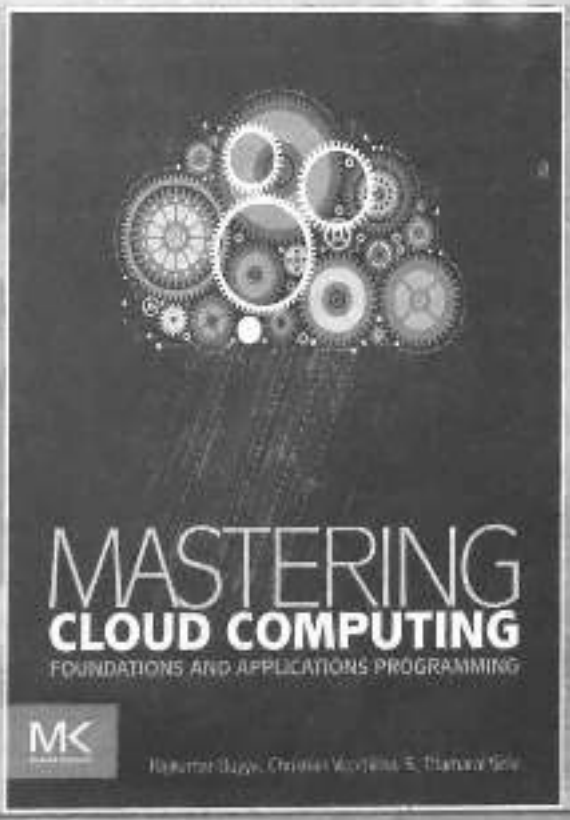
深入理解云计算

基本原理和应用程序编程技术

[澳] 拉库马·布亚（Rajkumar Buyya）
克里斯坦·维奇拉（Christian Vecchiola） 著
[印] S. 泰马莱·赛尔维（S. Thamarai Selvi）
刘丽 米振强 熊曾刚 译

Mastering Cloud Computing

Foundations and Applications Programming



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

深入理解云计算：基本原理和应用程序编程技术 /（澳）布亚（Buyya，R）等著，刘丽，米振强，陶雪刚译．—北京：机械工业出版社，2015.4
（计算机科学丛书）
书名原文：Mastering Cloud Computing: Foundations and Applications Programming

ISBN 978-7-111-49658-8

I. 深… II. ① 布… ② 刘… ③ 米… ④ 陶… III. 计算机网络－研究 IV. TP393

中国版本图书馆 CIP 数据核字（2015）第 052134 号

本书版权登记号：图字：01-2013-7841

Mastering Cloud Computing: Foundations and Applications Programming
Rajkumar Buyya, Christian Vecchiola, S. Thamarai Selvi
ISBN:978-0-12-411454-8
Copyright © 2013 by Elsevier Inc. All rights reserved.
Authorized Simplified Chinese translation edition published by the Proprietor.
Copyright © 2015 by Elsevier (Singapore) Pte Ltd. All rights reserved.
Printed in China by China Machine Press under special arrangement with Elsevier (Singapore) Pte Ltd. This edition is authorized for sale in China only, excluding Hong Kong SAR, Macau SAR and Taiwan. Unauthorized export of this edition is a violation of the Copyright Act. Violation of this Law is subject to Civil and Criminal Penalties.
本书仅供中文版由 Elsevier (Singapore) Pte Ltd. 授权机械工业出版社在中国大陆境内独家出版和发行。本版仅限在中国境内（不包括香港特别行政区、澳门特别行政区及台湾地区）出版及标价销售。未经许可之出口，视为违反著作权法，将受法律之制裁。
本书封底贴有 Elsevier 防伪标签，无标签者不得销售。

本书从云基础知识、云应用编程和云平台三个方面，介绍云计算的起源、发展、核心技术、编程技巧和实际应用，基于 Aneka 平台详细讲解并行计算、高存叶量计算和数据密集型计算的范式。此外，还涉及亚马逊 Web 服务、谷歌 AppEngine 和微软 Azure 三大云平台，以及云计算在科学、工程、游戏、社交等领域的最新应用。

本书内容严谨，结构清晰，实例丰富，既可作为高等院校计算机相关专业的教材，也适合云技术研发人员阅读参考。

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：曲　熠　　责任校对：董纪丽
印　　刷：北京瑞德印刷有限公司　　版　　次：2015 年 4 月第 1 版第 1 次印刷
开　　本：185mm×260mm　1/16　　印　　张：22
书　　号：ISBN 978-7-111-49658-8　　定　　价：69.00 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）88378991　88361066　　销售热线：（010）68379504
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@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

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出，独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系。从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

华章教育
华章科技图书出版中心

译 者 序

Mastering Cloud Computing: Foundations and Applications Programming

云计算是在传统计算机科学基础上发展起来的新兴技术。随着移动设备的大量普及、网络与计算成本的大幅降低以及用户计算需求的不断提升，传统的 PC 计算模式必将向以云数据中心为核心的云计算模式转移。同时，云计算技术所提供的强大的计算能力、低廉的成本以及按需服务的模式将会从根本上推动计算密集型应用的进一步发展，以及全球计算能力的全方位释放。

由于涉及计算机科学领域的专业知识和技术，所以对普通研发人员而言，云计算技术颇有些阳春白雪的意味。究其根源，云计算技术是学术研究与工程开发的集合体。一方面，研究人员在有好的想法或者概念的情况下，需要一个良好的云计算模型与原型系统来验证所提出的方法；另一方面，工程人员则由于专业背景知识的缺乏，而很难进入真正的云计算技术开发领域。

作为云计算技术领域的先驱者和领导者，本书作者——澳大利亚墨尔本大学计算机科学系著名教授 Rajkumar Buyya，长期从事网格计算及云计算技术研究，研究成果得到学术界与工业界的广泛认可。本书集成了 Buyya 教授在云计算领域多年的研究成果，从云计算基础、应用平台和编程技术三个方面，利用 11 个章节详细介绍云计算技术的起源、发展、核心技术及其范式、典型云计算平台以及编程技巧等重要内容。本书内容丰富、深入浅出，并包含配套习题，适合不同层次的云计算技术研发人员使用。同时，本书清晰的思路、严谨的技术框架和详尽的实例讲解也使其成为适用于计算机相关学科本科生及研究生教学的不可多得的优秀教材。

参与本书翻译工作的人员均多年从事云项目相关工作，对云计算前沿问题的研究及教学都有较深刻的理解。本书前言、第 1 ～ 4 章及索引由刘丽（北京科技大学自动化学院副教授）翻译，第 5 ～ 8 章由米振强（北京科技大学计算机与通信工程学院讲师）翻译，第 9 ～ 11 章由熊曾刚（湖北工程学院计算机与信息科学学院教授）翻译，全书由刘丽统稿。北京科技大学的研究生张森、翟颖奇、夏毓娴、徐安琪、谢翔、潘梦圆、李萌、何苗、陆源等同学协助进行资料收集，并参与了部分章节的初稿翻译工作。

感谢机械工业出版社引进此书并为本书的出版付出大量努力，使 IT 从业人员和计算机相关专业学生从中受益。特别感谢本书作者 Rajkumar Buyya 教授对翻译过程中遇到的问题进行解答。

需要说明的是，本书翻译工作和云计算研究工作得到以下项目的资助和支持：国家自然科学基金项目“互联云环境中基于效用模型的跨云协同服务优化研究（No. 61370132）”、“大数据环境下基于视觉主题模型的视觉数据分类方法研究（No. 61370092）”、“互联云环境下面向数据中心的服务资源分配与调度机制研究（No. 61472033）”，国家高技术研究发展计划（863 计划）项目“城市多模式数据系统互联技术与支撑环境（No. 2013AA01A601）”，以及湖北省自然科学基金项目“云计算环境下内容语义信任度量与评估方法研究

(No. 2013CFC005)”和湖北省高等学校优秀中青年科技创新团队计划项目“云计算环境下智能信息处理技术研究（No. T201410）”。

由于译者对云计算相关变革性技术的理解有待加深，而且许多新出现的专业术语还没有公认的译法，所以在翻译过程中难免出现一些不够清楚的表述，若有不妥之处，恳请广大读者批评指正，电子邮箱 liuli@ustb.edu.cn。

刘丽

2015 年 1 月

随着互联网与 Web 技术的快速发展和普及，以及手持计算机、移动设备、传感器设备功能的不断强大，人与人之间的交互方式、商业行为以及获取和提供服务的模式都在发生变化。低成本的计算与通信驱动了从单一计算方式向以数据中心为核心的计算方式的转变。尽管并行与分布式计算在 IT 行业已经存在多年，但其新的形式——多核和云计算为 IT 行业带来了彻底的变革。这种发展趋势将促使 IT 行业从 PC 应用开发模式转变为支持数百万用户同时使用软件的云数据中心模式。

计算向商业服务模式变革，这种计算服务类似于传统的公共基础设施服务，如水、电、煤气和电话。因此 IT 服务被当作与水、电、煤气和电话一样的“计算公共基础设施”，通过共有传输网络来交付使用并计费。在这种计算环境下，用户按需获取服务，而不管该服务由哪里提供。一些计算模式已经提出交付这种效用计算服务的构想，云计算是最新出现的实现这一目标的计算模式。

云计算如今已成为 IT 行业的又一个流行术语。众多 IT 厂商承诺提供存储、计算及应用托管服务，其服务范围涵盖几大洲，并能提供基于服务等级协议（Service Level Agreement，SLA）的服务性能保障和运行时间承诺。云服务模式允许用户基于订阅方式访问基础设施、平台及应用，也就是通常所说的基础设施即服务（IaaS）、平台即服务（PaaS）、软件即服务（SaaS）。这种服务模式大大降低了计算和应用的成本，但是要实现应用和服务开发及交付的一致性、可扩展性、可靠性是极其复杂的。

已有一些云计算技术和云平台产品，如谷歌 AppEngine、微软 Azure 和 Manjrasoft Aneka。谷歌 AppEngine 利用大量 IT 基础设施为托管的 Web 应用提供可扩展的运行环境。微软 Azure 为在云计算环境中开发和部署应用提供了大量 Windows 服务实例。Manjrasoft Aneka 能够灵活地创建云应用并将其部署在各种基础设施上，如亚马逊公共云 EC2。

随着应用开发从 PC 向云数据中心迁移，需要大量掌握云计算技能的人员。面对这一挑战，大学教育在培养下一代 IT 专业人员方面发挥了重要作用，帮助学生学习和掌握新的相关技术与工具。这样，大学需要以较小的投入建立云计算教学环境，而 Manjrasoft Aneka 比较适合建立这种云应用平台，它允许用户利用已有计算机网络组建私有云/企业云，提供软件开发工具包（SDK），支持多种编程模型（如线程、任务、MapReduce）的应用编程接口（API），支持在多核服务器、私有云、公共云等不同基础设施上无缝地部署和执行应用。

如今，专业开发人员需要创建云应用和服务。云计算研究人员、从业者以及供应商努力让用户了解云计算的好处并充分利用其潜在能力。然而，由于云计算是一种新兴的计算范式，所以对于云计算的精确定义，不同的云计算专家会给出不同的答案。因此，尽管现在比以往更好地实现了真正的效用计算，但是，与云计算服务方交互的复杂性使得对于云计算的认可和应用还只限于领域专家。本书旨在用简单的方式向读者讲解云计算基础知识、技术及编程技能，让更多普通程序员和软件工程师能轻松地开发云应用程序。

本书结构

本书介绍云计算的基本原理及相关范式，阐述云计算架构模型中虚拟化技术的概念，并

展示包括 Aneka 云计算应用平台在内的著名云计算技术产品，详细讲解并行计算、高吞吐量计算和数据密集型计算的范式，以及如何将这些范式应用于云应用程序开发。本书还研究了来自科学界、工程界、游戏和社交网络领域的多个应用案例，阐述了各应用案例的架构以及云计算技术的应用方式。这些案例研究有助于读者对云计算原理的理解。最后，本书详细阐述了许多源于云计算快速应用的开放性研究问题和机遇，我们希望这有助于激发读者在未来的研发过程中解决这些问题。另外，了解本书相关内容可浏览 Web 站点（http://www.buyya.com/MasteringClouds），其中包含更多在线资源。

全书分为三部分，共 11 章：

第一部分　基础
- 第 1 章　导论
- 第 2 章　并行计算与分布式计算原理
- 第 3 章　虚拟化
- 第 4 章　云计算架构

第二部分　云应用编程与 Aneka 平台
- 第 5 章　Aneka：云应用平台
- 第 6 章　并行计算：线程编程
- 第 7 章　高吞吐量计算：任务编程
- 第 8 章　数据密集型计算：MapReduce 编程

第三部分　工业云平台与新进展
- 第 9 章　工业云平台
- 第 10 章　云应用
- 第 11 章　云计算高级主题

本书将引导读者进入云计算领域，从理论基础讲解开始，让学生和专业人员通过在 Aneka 平台上实际开发云应用程序来理解和掌握相关概念。第三部分介绍业界其他云技术和解决方案（亚马逊 Web 服务、谷歌 AppEngine 和微软 Azure）及其实际应用，阐述云计算的发展趋势和发展前景。

读者对象

由于云计算迅速崛起为一种主流计算模式，所以必须深入理解其核心概念和特性，并掌握如何设计和实现云计算的应用程序与系统。这是如今的软件架构师、工程师和开发人员应具有的基本技能，因为多数应用都将被迁移到云环境。随着技术的成熟，具备此技能尤其重要。本书涵盖云计算的起源、理论和实际开发技术，读者对象更广泛，可作为研究生、IT 从业者、开发人员、工程师等设计和实现云计算解决方案的参考书。此外，书中最后关于相关研究的展望更加吸引云计算领域的研究人员探究其将带来的新挑战。

云计算正在不断获取相当可观的商业利益且发展势头强劲，本书为云计算领域做出了非常适时的贡献。本书主要针对研究生和 IT 专家，例如系统架构师、软件工程师、应用程序员等。在未来的 20 年里，云计算将是对科学研究和社会生活产生重大影响的五大新兴技术之一，因此认真理解和掌握本书的知识将帮助读者置身 IT 领域的前沿。

用书指南：理论、实验室、项目

鉴于云计算范式的重要性及其在业界的快速崛起，教育机构应该更新其课程体系，增加

云计算或相关领域的一门或多门专业课程，例如“并行计算”和“分布式系统”。我们建议大学设置面向本科生或研究生的云计算专业，及计算科学学士和相关硕士学位，相信此书将是该专业的优秀教材。如果学生已经了解并行和分布式计算的概念，可以跳过第 2 章。

对于想用云计算丰富其课程体系的学校，建议分别在两个学期开设两门课程：“云计算基础课程”和“云计算高级课程”。本书第 1 ～ 6 章适用于云计算基础课程，第 7 ～ 11 章适用于云计算高级课程。

除了理论学习以外，我们强烈推荐实验室使用本书，书中给出了很多实验指导。实验练习和作业题包括数学函数的并行执行、大量数据排序的并行处理、图像处理和数据挖掘。在 Aneka 云软件系统上，学校很容易利用已有的 Windows 系统的计算机局域网络搭建私有云（企业云）计算环境。学生可利用此环境学习各种云应用编程模型和接口的实例并在 Aneka 平台上实现，如第 6 章的线程编程、第 7 章的任务编程、第 8 章的 MapReduce 编程。章后编程习题可作为实验作业让学生自己编写程序实现。

学生也可以在其本科毕业设计中开发处理实际问题的云应用程序。例如，学生可以协助其他科学领域或工程领域（比如生命医学科学、机械科学）的研究人员，利用云计算强大的计算能力开发符合实际需求的应用。请阅读并学习第 10 章的各种应用案例。

教辅资源[⊖]

教师与学生需要的教辅资源可以从 Elsevier 网站下载（booksite.elsevier.com/9780124114548），也可以从作者为本书所做的主页（http://www.buyya.com//MasteringClouds）下载。

⊖ 关于本书教辅资源，使用教材的教师需通过爱思唯尔的教材网站（www.textbooks.elsevier.com）注册并通过审批后才能获取。具体方法如下：在 www.textbooks.elsevier.com 教材网站查找到该书后，点击“instructor manual”便可申请查看该教师手册。有任何问题，请致电 010-85208853。——编辑注

首先感谢所有学者和云计算开发者为本书探讨的各种概念与技术所做出的贡献。特别感谢 Manjrasoft 公司、墨尔本大学云计算与分布式系统实验室（CLOUDS）、墨尔本风险投资公司的所有员工和顾问，他们为 Aneka 云应用平台的开发、应用示范和文档的准备以及 Aneka 技术的商业化做出了贡献。他们是：Chu Xingchen、Srikumar Venugopal、Krishna Nadiminti、Christian Vecchiola、Dilchan Karunamoorthy、Chao Jin、Rodrigo Calheiros、Michael Mattess、Jessie Wei、Enayat Masoumi、Ivan Mellado、Richard Day、Wolfgang Gentzsch、Laurence Liew、David Sinclair、Suraj Pandey、Abhi Shekar、Dexter Duncan、Murali Sathya、Karthik Sukumar、Ravi Kumar Challa 和 Sita Venkatraman。

感谢澳大利亚研究委员会（ARC）与创新、工业、科学和研究部（DIISR）对我们的研究工作及其商业化尝试的支持。

感谢墨尔本大学的所有同事，尤其是 Rao Kotagiri 教授、Iven Mareels 教授 和 Glyn Davis 教授，他们对我们的研究和研究成果的传播给予指导与支持。

感谢 Aneka 技术的研发同事和用户，他们对本书的应用案例做出了直接或间接的贡献。特别感谢 ADRIN/ISRO 的 Raghavendra Kune，他利用 Aneka 建立了卫星图像处理应用并发表了文章。感谢 MSRIT 的 Srinivasa Iyengar，他利用 Aneka 开发了数据挖掘应用程序，并在云计算早期将 Aneka 引荐到学术界。

感谢 CLOUDS 实验室人员对本书部分章节的校对，他们是：Rodrigo Calheiros、Nikolay Grozev、Amir Vahid、Anton Beloglazov、Adel Toosi、Deepak Poola、Mohammed AlRokayan、Atefeh Khosravi、Sareh Piraghaj 和 Yaser Mansouri。

感谢家人 Smrithi Buyya、Soumya Buyya 和 Radha Buyya 在本书编写过程中对我的爱和理解。

我们诚挚地感谢 Elsevier 出版社的外审专家对本书提出的意见和建议，使得很多章节具有更好的组织方式和呈现效果，这为我们提高全书质量带来了极大帮助。

最后，我们要感谢 Elsevier 出版社的员工在书稿准备过程中给予的热情帮助和指导。特别感谢 Todd Green 鼓励我们承接此项目并促成此书出版。Elsevier 出版社的员工太棒了！

Rajkumar Buyya 教授
澳大利亚墨尔本大学和 Manjrasoft 公司
Christian Vecchiola 博士
澳大利亚墨尔本大学和 IBM 研究院
S. Thamarai Selvi 教授
印度安那大学金奈分校 Madras 技术学院

出版者的话
译者序
前 言
致 谢

第一部分 基础

第 1 章 导论 …… 2
1.1 云计算简介 …… 2
1.1.1 云计算构想 …… 3
1.1.2 云计算定义 …… 4
1.1.3 进一步了解云计算 …… 6
1.1.4 云计算参考模型 …… 7
1.1.5 特性和优势 …… 9
1.1.6 面临的挑战 …… 10
1.2 云计算起源 …… 10
1.2.1 分布式系统 …… 11
1.2.2 虚拟化 …… 12
1.2.3 Web 2.0 …… 13
1.2.4 面向服务的计算 …… 14
1.2.5 效用计算 …… 15
1.3 构建云计算环境 …… 16
1.3.1 应用程序开发 …… 16
1.3.2 基础设施和系统开发 …… 16
1.3.3 云计算平台和技术 …… 17
本章小结 …… 18
习题 …… 19
第 2 章 并行计算与分布式计算原理 …… 20
2.1 计算时代 …… 20
2.2 并行计算与分布式计算 …… 21
2.3 并行计算基本要素 …… 21
2.3.1 什么是并行处理 …… 21
2.3.2 并行处理硬件架构 …… 22
2.3.3 并行编程方法 …… 25
2.3.4 并行性的级别 …… 25
2.3.5 注意事项 …… 26
2.4 分布式计算基本要素 …… 26
2.4.1 通用概念和定义 …… 27
2.4.2 分布式系统组件 …… 27
2.4.3 分布式计算架构模式 …… 28
2.4.4 进程间通信模型 …… 35
2.5 分布式计算技术 …… 37
2.5.1 远程过程调用 …… 37
2.5.2 分布式对象框架 …… 38
2.5.3 面向服务的计算 …… 42
本章小结 …… 48
习题 …… 48
第 3 章 虚拟化 …… 50
3.1 简介 …… 50
3.2 虚拟化环境特点 …… 51
3.2.1 更强的安全性 …… 52
3.2.2 执行管理 …… 53
3.2.3 可移植性 …… 54
3.3 虚拟化技术分类 …… 54
3.3.1 执行虚拟化 …… 54
3.3.2 其他类型的虚拟化 …… 61
3.4 虚拟化和云计算 …… 62
3.5 虚拟化的利与弊 …… 64
3.5.1 虚拟化技术的优点 …… 64
3.5.2 虚拟化技术的缺点 …… 64
3.6 技术实例 …… 66
3.6.1 Xen：半虚拟化 …… 66
3.6.2 VMware：完全虚拟化 …… 67
3.6.3 微软 Hyper-V …… 72
本章小结 …… 75
习题 …… 75
第 4 章 云计算架构 …… 76
4.1 简介 …… 76

4.2 云计算参考模型……76
4.2.1 架构……76
4.2.2 基础设施即服务和硬件即服务……78
4.2.3 平台即服务……80
4.2.4 软件即服务……83
4.3 云的种类……85
4.3.1 公共云……85
4.3.2 私有云……86
4.3.3 混合云……88
4.3.4 社区云……90
4.4 云计算经济特性……91
4.5 云计算面临的挑战……93
4.5.1 云计算定义……93
4.5.2 云计算互操作性和标准……93
4.5.3 可扩展性和容错性……94
4.5.4 安全、可信和隐私……94
4.5.5 组织方面……95
本章小结……95
习题……95

第二部分 云应用编程与 Aneka 平台

第 5 章 Aneka：云应用平台……98
5.1 框架概述……98
5.2 Aneka 容器结构……100
5.2.1 Aneka 平台基础：平台抽象层……101
5.2.2 构造服务……101
5.2.3 基础服务……103
5.2.4 应用服务……105
5.3 构建 Aneka 云平台……106
5.3.1 基础设施组织……107
5.3.2 逻辑组织……107
5.3.3 私有云部署模式……109
5.3.4 公共云部署模式……110
5.3.5 混合云部署模式……111
5.4 云编程和云管理……112
5.4.1 Aneka SDK……112
5.4.2 管理工具……115
本章小结……116
习题……116
第 6 章 并行计算：线程编程……117
6.1 单机并行计算简介……117
6.2 线程编程应用……118
6.2.1 什么是线程……119
6.2.2 线程 API……120
6.2.3 线程并行计算技术……121
6.3 Aneka 多线程方式……130
6.3.1 线程编程模型简介……131
6.3.2 Aneka 线程和普通线程……132
6.4 Aneka 线程编程应用……135
6.4.1 Aneka 线程应用模型……135
6.4.2 域分解：矩阵乘法……136
6.4.3 功能分解：Sine、Cosine、Tangent……142
本章小结……147
习题……148
第 7 章 高吞吐量计算：任务编程……149
7.1 任务计算……149
7.1.1 任务特性……150
7.1.2 计算类别……150
7.1.3 任务计算框架……151
7.2 基于任务的应用模型……152
7.2.1 高度并行应用……153
7.2.2 参数化应用……153
7.2.3 消息传递接口应用……155
7.2.4 具有任务依赖性的工作流应用……156
7.3 基于任务的 Aneka 编程……159
7.3.1 任务编程模型……159
7.3.2 用任务模型开发应用……160
7.3.3 开发参数化应用……174
7.3.4 管理工作流……176
本章小结……178
习题……179
第 8 章 数据密集型计算：MapReduce 编程……181
8.1 什么是数据密集型计算……181

8.1.1 数据密集型计算特性 …… 182
8.1.2 未来的挑战 …… 182
8.1.3 历史背景 …… 183
8.2 数据密集型计算技术 …… 186
8.2.1 存储系统 …… 186
8.2.2 编程平台 …… 193
8.3 Aneka MapReduce 编程 …… 199
8.3.1 MapReduce 编程模型简介 …… 199
8.3.2 应用实例 …… 216
本章小结 …… 226
习题 …… 226

第三部分 工业云平台与新进展

第 9 章 工业云平台 …… 228
9.1 亚马逊 Web 服务 …… 228
9.1.1 计算服务 …… 229
9.1.2 存储服务 …… 232
9.1.3 通信服务 …… 239
9.1.4 其他服务 …… 240
9.1.5 总结 …… 241
9.2 谷歌 AppEngine …… 241
9.2.1 架构和核心概念 …… 241
9.2.2 应用程序生命周期 …… 245
9.2.3 成本模型 …… 247
9.2.4 结论 …… 247
9.3 微软 Azure …… 248
9.3.1 Azure 核心概念 …… 248
9.3.2 SQL Azure …… 252
9.3.3 Windows Azure 平台设备 …… 253
9.3.4 结论 …… 253
本章小结 …… 254
习题 …… 254
第 10 章 云应用 …… 256
10.1 科学应用 …… 256
10.1.1 医疗保健：心电图分析 …… 256
10.1.2 生物学：蛋白质结构预测 …… 257
10.1.3 生物学：基因表达数据用于癌症诊断分析 …… 258
10.1.4 地球科学：卫星图像处理 …… 259
10.2 商业和消费应用 …… 260
10.2.1 CRM 和 ERP …… 260
10.2.2 效率型应用 …… 262
10.2.3 社交网络 …… 265
10.2.4 媒体应用 …… 265
10.2.5 多人在线游戏 …… 268
本章小结 …… 269
习题 …… 269
第 11 章 云计算高级主题 …… 270
11.1 云能效 …… 270
11.2 基于市场的云管理 …… 273
11.2.1 面向市场的云计算 …… 273
11.2.2 MOCC 参考模型 …… 274
11.2.3 支持 MOCC 的技术和实现 …… 278
11.2.4 结论 …… 282
11.3 云联盟和互联云 …… 282
11.3.1 特性和定义 …… 282
11.3.2 云联盟栈 …… 283
11.3.3 关注点 …… 288
11.3.4 云联盟技术 …… 301
11.3.5 结论 …… 304
11.4 第三方云服务 …… 304
11.4.1 MetaCDN …… 304
11.4.2 SpotCloud …… 306
本章小结 …… 307
习题 …… 307
参考文献 …… 309
索引 …… 317

第一部分

Mastering Cloud Computing: Foundations and Applications Programming

基　础

第 1 章

Mastering Cloud Computing: Foundations and Applications Programming

导　论

计算正在发生变革，它将转化为一种商业化服务模式，像提供水、电、煤气和电话等基础设施服务一样来交付计算服务。这种模式下，用户根据需求获得计算服务，而不需要知道该服务由哪里提供。一些计算模式（如网格计算）提出了交付这种效用计算服务的构想。近年来新兴的云计算模式则将实现了这一蓝图。

云计算是一项技术进步，它关注如何设计计算系统和开发应用程序，以及如何利用现有的服务构建软件。云计算基于动态交付概念，不仅是服务的动态交付，还包括计算能力、存储、网络、信息技术（IT）基础设施等。资源可以通过互联网并基于使用付费的方式由云计算供应商提供。今天，任何人都可以用信用卡订阅云服务，并在一定时间内为应用程序部署和配置服务，还可根据需求增加或减少配置资源，且只需要按资源的使用时间和使用量付费。

本章简要概述了云计算模式，介绍其发展前景，讨论其核心特征，并追溯其发展历程。本章还介绍了云计算的关键技术，以及关于云计算发展方向的深入见解。

1.1　云计算简介

1969 年，原高级研究计划署网络（ARPANET）的首席科学家之一，互联网奠基人 Leonard Kleinrock 说：

> 目前，计算机网络仍处于起步阶段，但是随着其成长和成熟，我们可能会看到计算服务的普及，像电力和电话等公共事业服务一样，跨国界服务于每个家庭和办公室。

这种计算模式的设计蓝图是基于服务的交付模式，在 21 世纪，整个计算机行业将产生巨大变革，计算服务将按需求即时交付，类似于水、电、电话、燃气等公共服务。同样，只有当用户（消费者）访问计算服务时，他们才需要向服务供应商支付费用。此外，消费者不再需要将大量投资用于构建和维护复杂的 IT 基础设施。

在这样的交付和使用模式中，用户可以按需求获得服务，而不用考虑服务提供方位于何处。这种模式曾经被称为效用计算。近期（2007 年起）改称为云计算。IT 基础设施称为“云”，企业和用户可以从任何地方按需访问各种应用服务。因此，可以认为云计算是动态交付计算服务的新模式，它拥有采用虚拟化技术构建的先进数据中心，可实现资源的整合和有效利用。

云计算允许用户租用基础设施、运行环境和服务，按需使用与付费。云计算原理得到广泛应用，不同的人对云计算有不同的认识。大型企业首席信息官和技术主管看到了云计算的机遇，按企业商业需要部署和扩展基础架构。终端用户应用云计算服务，可以从任何地方通过互联网随时访问自己的文件和数据。还有许多关于云计算的观点[⊖]，其中，最直白的观点可

⊖ 2007 年 5 月 Web 2.0 发布会上，Joyent 公司副总裁 Rob Boothby 组织了一系列云计算研讨，有趣的是，不同的人对云计算有不同的认识。首席执行官、技术主管、IT 公司创始人、IT 分析师都给出了各自的观点，正是从那时开始，云计算概念逐渐被人们熟知。研讨会视频发布在 YouTube 上，网址为 www.youtube.com/watch?v=6PNuQHUiV3Q。

以概括如下：

> 我不在乎服务器在哪儿、谁管理它们、文档存储在哪儿，或者应用程序托管在哪儿。我只是希望它们始终可用，并且可以通过连接到互联网的任何设备访问它们。我愿意为我需要的服务付费。

这一描述与我们使用其他服务的方式非常相似，例如水、电等。换句话说，云计算已经由IT服务转变为公共事业服务。通过多种技术的有效融合，云服务模式已经可以实现，而且已经相对成熟。Web 2.0技术在应用云计算构建计算系统中发挥了核心作用。云计算可以把互联网转化成一个丰富的、足以满足各种复杂需求的应用和服务的交付平台。面向服务架构将计算资源抽象为服务，使得云计算能提供计算服务能力；而虚拟化赋予了云计算用于构建生产和企业系统时必要的可定制化特性、可控性和灵活性。

除了能灵活地构建新的系统和应用之外，云计算环境还能对现有系统进行扩展。动态配置IT资源比购买额外的硬件和软件更有吸引力，这些硬件和软件资源的需求量很难估计并且受限于时间因素。这是云计算最重要的优势之一，也使其成为了技术热点。随着云计算系统的广泛部署，支持云计算的基础技术与系统逐步完善和标准化，这是实现云计算长远目标的基础。云计算提供了一个开放的环境，计算、存储和其他服务都将作为计算服务进行交易。

1.1.1 云计算构想

云计算为任何一个拥有信用卡的人提供虚拟硬件、运行环境和服务，用户需要时就可以使用，不需要事先委托。计算系统总体架构被转换成各种计算服务的集合，这些服务被配置和组合在一起以便快速部署系统，以小时为单位而不是以天为单位，并且没有维护成本。这种模式起初遭到怀疑，而现在已经在多个领域和行业中得到实际应用（见图1-1）。需求紧紧跟随着技术的发展而发展，丰富了云计算所提供的服务集合，使云服务变得更加成熟，成本也更低。

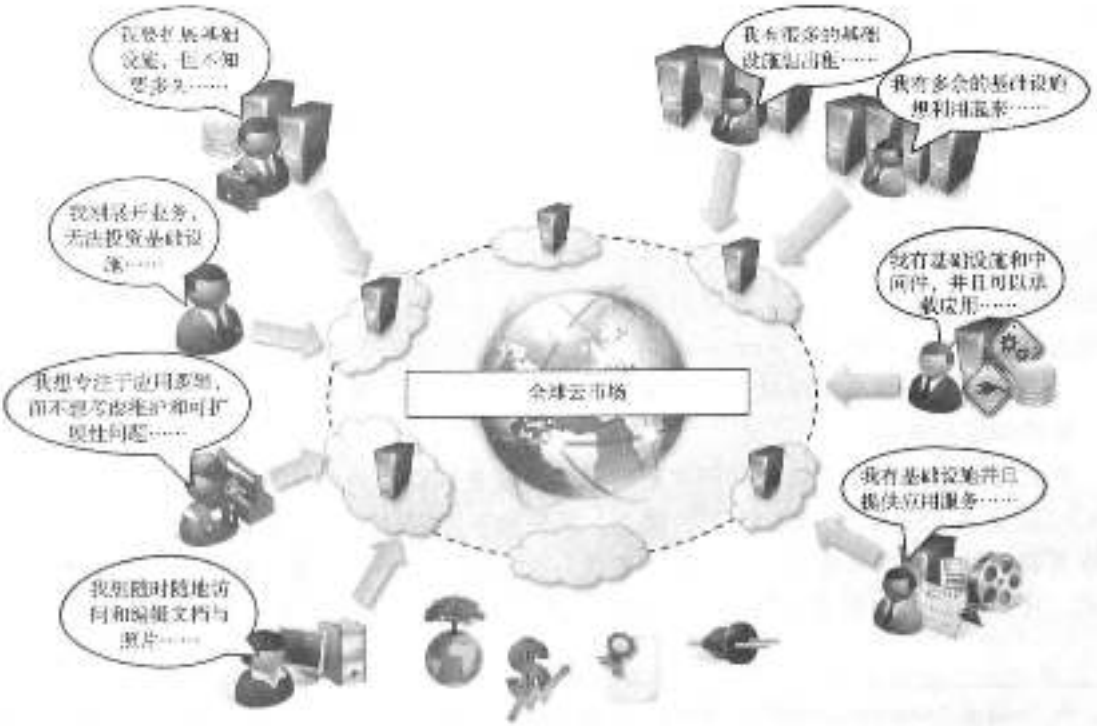


图1-1 云计算构想

尽管技术在不断演变，然而云计算的使用往往仅限于在某一时刻提供单一服务，更常见的是，一组相关服务由同一供应商提供。缺乏有效的标准化措施，使得服务从一个供应商迁移到另一个供应商变得非常困难。云计算的长期构想是将 IT 服务作为公用基础设施服务，在没有技术和法律限制的开放市场中进行交易。在这样的云市场环境中，云服务供应商和消费者把云服务作为公共基础设施服务进行交易，发挥其核心作用。

一些实现云计算模式的技术已经存在，可以从不同的角度利用云的各种服务功能。对于无处不在的存储和按需计算能力的需求是应用云计算的最常见场景。对于应用和系统开发者而言，在不具有计算资源或者没有能力扩展现有资源的情况下，一个可扩展的应用程序运行系统是非常有吸引力的选择。终端用户更愿意基于 Web 方式访问文件，并采用成熟的应用软件处理文件。

在这种情况下，服务的发现主要通过人为干预完成：人们浏览互联网，以确定能够满足其需求的服务。可以想象，未来只需在提供云计算服务的全球化数字市场中输入我们所需要的服务，就可以自动匹配服务。这种云服务市场能自动发现服务，并集成现有的软件系统，从而使用户能够在应用系统中透明地利用云资源。云服务交易的全球化平台使服务供应商更容易被消费者发现，潜在地增加了收益。也消除了消费者和供应商之间的界限，即不再只能拥有这两种身份中的一种。例如，为了履行对客户的承诺，一个云服务供应商可能会成为其竞争供应商的消费者。

可以通过制定描述云服务的统一标准，以及不同云技术之间的交互技术标准，来建立全球云计算交易平台。已有大量应用向云计算转移，并迅速普及。此外，将云计算的核心功能应用于大型数据中心，可以减少或消除消费者对于基础设施的需求，优化数据中心资源，充分利用其为多用户提供共享服务的能力。这种模式将减少能源浪费和碳排放，从而一方面促进了绿色 IT 的实现，另一方面提高了收益。

1.1.2 云计算定义

云计算已经成为流行术语，广泛用于指代各种技术、服务和概念。它往往与硬件虚拟化或按需计算、效用计算、IT 外包、平台和软件即服务，以及其他 IT 热门技术有关。图 1-2 描述了已有云计算定义中的不同概念。

术语云曾用于电信行业，作为系统结构图的网络抽象表示。之后，它成为最流行的计算机网络——互联网（Internet）的象征符号。这个含义也适用于云计算，指一种以互联网为中心的计算方式。在云计算中，互联网扮演着基础性的角色，代表介质或平台，众多云计算服务通过互联网交付和访问。Armbrust 等人 [28] 给出的定义也体现了这方面的含义：

> 云计算是指应用以服务形式通过互联网交付使用，并且数据中心的硬件和软件能提供这些服务。

此云计算定义涵盖了从底层硬件到高层软件服务和应用的总体架构。产生了一切皆服务（Everything as a Service，XaaS[⊖]）的理念，系统的不同组件——IT 基础设施、开发平台、数据库等都可以作为服务来交付、定量和定价。这种新模式不仅显著地影响着我们构建软件的方式，还对我们如何部署软件、使用软件、设计 IT 系统结构产生影响，甚至决定着公司投

⊖ XaaS 是 X-as-a-Service 的缩写，其中 X 可以被以下任一字符代替：S（软件）、P（平台）、I（基础设施）、H（硬件）、D（数据库），等等。

资IT需求的方式。以云计算为核心的新型应用不断产生：包括单个用户在云中托管文档的需求，以及某CIO决定在公共云中部署部分或整个IT基础设施的需求。美国国家标准与技术研究院（NIST）对云计算的定义重点强调了多参与者使用共享云计算环境的观点，具体描述如下：

> 云计算是一种能够便捷地按需访问共享可配制计算资源池（如网络、服务器、存储、应用、服务）的服务模式，并只需要很少的管理工作或与服务供应商的很少交互就可以快速提供和发布这些服务。



图 1-2 云计算技术、概念和设想

云计算的另一重要特性是面向效用。相对于分布式计算方式，云计算更加侧重于以给定的定价模式提供服务，多采用“按使用量付费”的策略。这样可以在线访问存储、租用虚拟硬件，或者使用软件开发平台，并且只按实际使用付费，没有或很少需要先期投入。所有操作只需通过Web浏览器访问云服务，并输入信用卡详细信息进行付费。云计算具有不同的、更实用的特征。Reese[29]认为服务是否能按照云计算模式进行交付需满足三个标准：

- 可通过Web浏览器或Web服务应用程序接口（API）访问服务。
- 不需要先期投入。
- 按资源使用量付费。

尽管许多云计算服务可免费提供给单个用户，但企业级服务通常按照特定的定价方案交付使用。在这种情况下，用户订阅服务并与服务供应商签订服务等级协议（SLA），协议中规定了交付服务的服务质量参数。Buyya等人[30]清晰地描述了云计算的面向效用特性：

> 云计算是由一组互连的虚拟机组成的并行和分布式系统，该系统依据服务供应商和消费者之间协商确定的服务等级协议，动态配置和提供一种或多种统一的计算资源。

1.1.3 进一步了解云计算

云计算帮助企业、政府、公共机构、私营机构和研究机构形成更有效的、由需求驱动的计算系统。现在，访问和集成云计算资源和系统像在互联网上用信用卡进行交易一样简单，在很多市场活动中都存在这样的系统实例。

- 大型企业可以基于云计算系统实现业务活动。近日，《纽约时报》已经把旧版本的数字图书馆转换成了 Web 访问形式，这需要在较短的时间内具有较强的快速计算能力。《纽约时报》通过租用亚马逊的 EC2 和 S3 云资源，在 36 小时内完成了任务，并在完成任务后释放这些资源，没有额外成本。
- 小型企业和初创企业可以更快地将他们的想法转化为商业模式，而不需要过多的前期投入。Animoto 是将用户提交的照片、音乐和视频片段制作成视频影像的公司。编辑制作视频过程中需要大容量的存储设备和后台快速处理设备。Animoto 没有自己的服务器，其计算基础设施完全由亚马逊 Web 服务提供，根据计算处理的负载按需定制 Web 服务。计算负载差别很大，并具有快速可扩展性[⊖]。对多数公司而言，前期 IT 资源投入显然不是最佳策略，而云计算系统成为有效的解决方案。
- 系统开发者可以专注于业务逻辑，而不是处理复杂的 IT 基础设施管理和可扩展性。Little Fluffy Toys 是伦敦的一家公司，开发了一个小软件，为用户提供附近自行车租赁服务的信息。该公司成功地在谷歌 AppEngine 上开发了该软件，仅用一周时间就推向市场。
- 终端用户可以在任何地方、任何设备上访问文件。苹果 iCloud 提供一种服务，允许用户把他们的文件存储在云中，并通过连接到云的任何设备访问这些文件。用户可以在旅行中用智能手机拍照，回家后用笔记本式计算机编辑这些照片，并在平板电脑上更新和显示。这个过程对用户是完全透明的，不必通过电缆将这些设备相互连接。

如何让这一切成为可能？基于以上四种不同的场景，无论是计算能力、存储，还是应用
9 程序的运行环境，这些 IT 资源的相同特性是按需服务，并按使用量付费。云计算不仅具有便捷地按需访问 IT 服务的能力，同时还产生了一种把 IT 服务和资源视为公共基础设施服务的新模式。云计算环境的鸟瞰图如图 1-3 所示。

云计算部署和访问模型包括三种：公共云、私有云 / 企业云和混合云（见图 1-4）。公共云是最常见的部署模式，其必要的 IT 基础设施（如基于虚拟化技术的数据中心）由第三方服务供应商建立，任何用户都可以以订阅方式使用。该云服务模式对用户非常有吸引力，用户能快速地使用计算、存储和应用服务。在这种环境下，用户的数据和应用程序部署在服务供应商的云数据中心。

拥有庞大的计算基础设施的大型企业也可以从云计算中获益，方式是在其内部建立 IT 服务交付的云计算模式。这就产生了相对于公共云的私有云概念。2010 年，美国联邦政府，
10 世界上最大的 IT 消费机构之一（在超过 10000 个系统花费约 760 亿美元），启动了云计算计划，旨在为政府机构提供更高效地利用现有计算设施的方案。企业内部保护机密信息的需求推动了私有云的发展。像政府和银行这种需要高度安全性、隐私性和可监管性的机构，更愿意建立和使用自己的私有云或企业云。

⊖ 有报道称，由于客户需要，Animoto 在一周内便将服务器规模从 70 台增加到 8500 台。

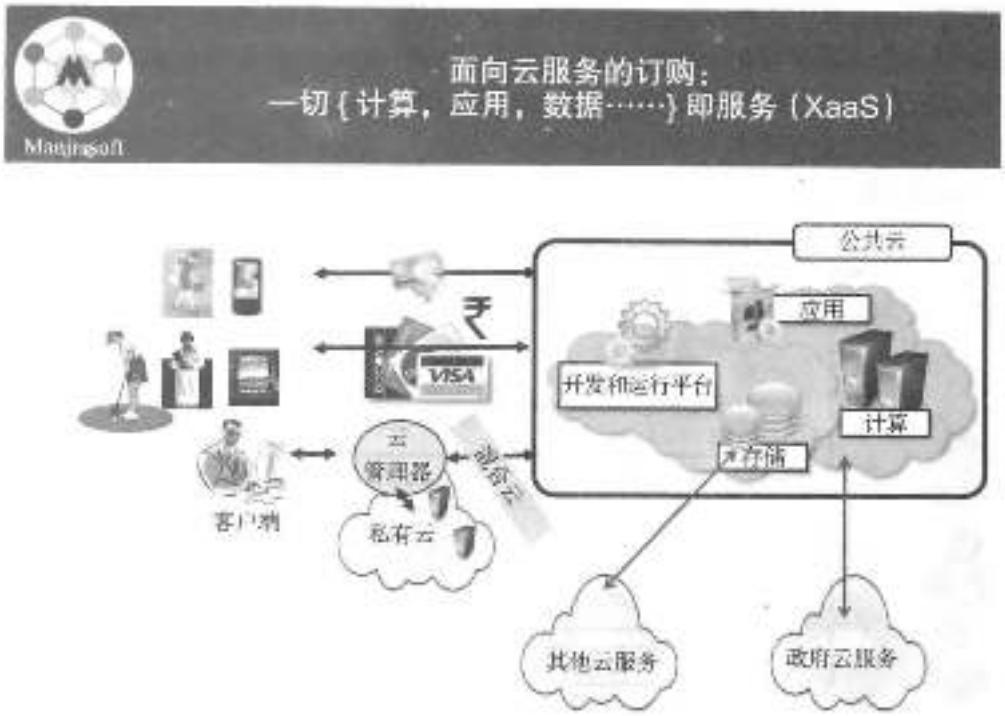


图 1-3 云计算鸟瞰图



图 1-4 云计算的主要部署模型

当私有云资源无法满足用户服务质量（QoS）要求时，可将公共云与私有云的计算能力相结合，这样就形成了混合云。混合云正在逐渐成为一种各方参与者充分利用云计算提供的各种服务能力的通用服务模式。

1.1.4 云计算参考模型

云计算的本质特征是具有按需提供各种 IT 服务的能力，这些 IT 服务彼此之间存在很大

差异，使得用户对于云计算的理解各不相同。尽管云服务缺乏统一性，但还是可以将云计算服务分为三大类：基础设施即服务（IaaS），平台即服务（PaaS）和软件即服务（SaaS）。如图 1-5 所示，这三类云服务彼此相关，构成了云计算的整体组织结构，称为云计算参考模
11 型。本书按此模型结构展开，介绍云计算技术和相关研究。该模型将各式各样的云计算服务归纳为自底向上的层次结构。

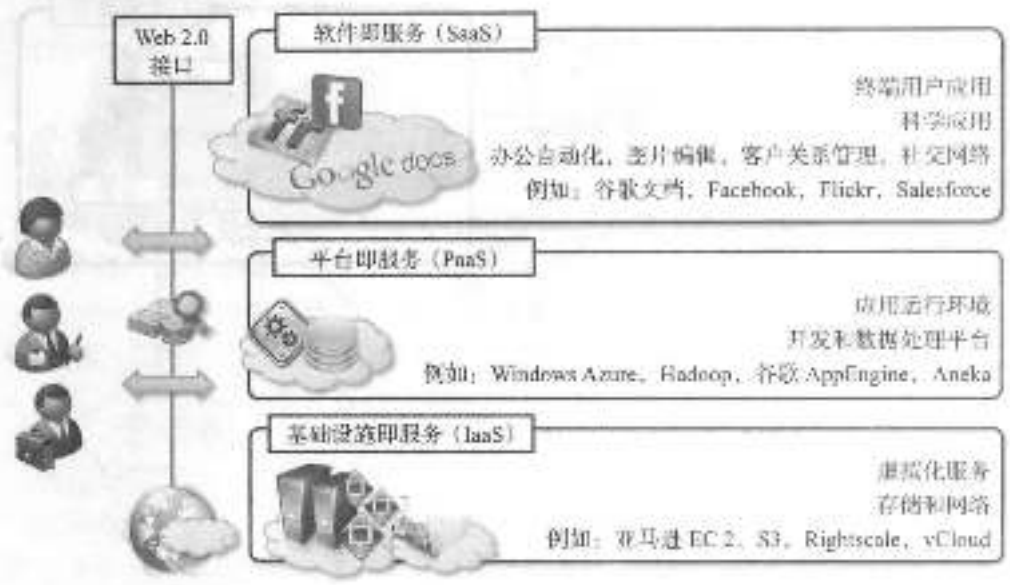


图 1-5　云计算参考模型

在云计算参考模型的最底层，IaaS 按需求提供虚拟化资源，如硬件、存储和网络。虚拟化硬件以虚拟机实例的形式按需提供计算能力。这些虚拟机实例是根据用户对基础设施的请求创建的，用户通过给定的工具和接口来配置安装在虚拟机中的软件。价格通常以美元 / 小时计算，其中每小时的费用由虚拟硬件的性能决定。虚拟存储按裸磁盘空间或对象存储的形式交付使用。前者扩充了用于持久存储的虚拟硬件产品；后者是一种更高层次的抽象，用于存储实体而不是文件。虚拟网络标识服务的集合，这些服务管理虚拟实例之间的网络，及其与互联网或专用网络之间的连接。

PaaS 是参考模型层次结构中的下一个平台服务层，它能够根据需要提供可扩展的、弹性的运行环境，执行用户的应用程序。PaaS 以核心中间件平台作为支持，该平台负责创建用于部署和执行应用程序的抽象环境。PaaS 服务供应商负责系统的可扩展性和容错管理，而用户只需利用服务的 API 和库，专注于应用程序开发中的业务逻辑。这种方法提供了一种更高层次的抽象，发挥了云计算的优势，也使用户易于在更可控的软件环境中编程。

位于参考模型顶部的 SaaS 按需提供应用程序和服务。大多数通用桌面应用程序，如办
12 公自动化、文件管理、照片编辑和客户关系管理（CRM）软件等，都统一部署在服务供应商的服务器上，使这些软件更易于升级并可按需通过浏览器访问使用。这些应用程序被众多用户共享，这些用户的交互独立于其他用户。社交网络站点常采用 SaaS 模式，基于云的基础设施减轻了由于大量用户使用所带来的高负载。

云计算参考模型中的每一层都为用户提供了不同的服务。用户采用 IaaS 是希望利用云计算构建动态可扩展的计算系统。因此，IaaS 服务可用来开发可扩展的 Web 站点或进

行后台处理。PaaS 为应用开发提供了可扩展的编程平台，在需要开发新系统的情况下更为适用。用户采用 SaaS 多是希望无需做任何软件开发、安装、配置和维护，就可以使用云计算应用软件服务，充分利用云的弹性扩展能力。该方案适用于满足用户需要的 SaaS 服务（如电子邮件、文档管理、CRM，等等）已存在，或者只需要少量软件客户化工作的情况下。

1.1.5 特性和优势

云计算具有众多值得关注的特性，可以同时为云服务消费者（CSC）和云服务供应商（CSP）带来利益。这些特性包括：

- 无需预先承诺。
- 按需获取。
- 合理的定价。
- 快速应用和弹性扩展能力。
- 有效的资源分配。
- 能源效率。
- 无缝构建和第三方服务的使用。

使用云计算系统和技术所带来的最明显的优势是因减少相关 IT 软件和硬件的维护成本和运营成本而带来的经济收益增长。主要原因是 IT 资产（包括软件和硬件）被转换成使用费用，在使用这些资源时才需要支付，不需要前期资金投入。资本成本是与资产相关的成本，在经营活动前就需要投入。在云计算出现之前，企业为了维持运营需要预先购买计算基础设施，这些 IT 硬件和软件投资产生资本成本。随着时间的推移，再将业务收入用于补偿资本成本。企业总是最大限度地减少资本成本，因为它们往往涉及折旧。举一个关于硬件折旧的例子：今天买的 1000 美元的服务器，当它被一个新的硬件替换时，其公允价值会低于其购买原价。为了获得正收益，企业会弥补由于时间产生的折旧，因而从收入中获得的净收益就减少了。因此，最大限度地降低资本成本是最基本的方法。云计算将 IT 基础设施和软件转化为使用费用，从而为增加企业净收益做出了显著贡献。此外，它也给小企业和创业者提供了机会：业务刚开展时，不需要大量投资，随着业务扩展，投资可以逐渐加大。最后，维护成本也显著减少：通过租用基础设施和应用服务，企业不再需要对其进行维护。云服务供应商负责维护工作，由于规模经济效应，供应商能够承受维护成本。

软件系统构建灵活性的提高是云计算的另一显著优势。由于企业租用了 IT 服务，他们可以更加动态地、灵活地构建自己的软件系统，而不受 IT 资产的资本成本限制。关于系统能力规划的需求减少了，因为云计算能快速响应不断变化的需求。例如，企业可以租用更多的服务器来处理负载高峰，并在不需要时退租。易于扩展是云计算的另一特性。通过利用云计算潜在的巨大计算能力，企业可以更加轻松地扩展其 IT 系统能力。扩展性在云计算参考模型的各层中均有体现。IaaS 供应商使得用户可以用简单的方法来定制硬件资源，并将其集成到现有系统中。PaaS 供应商为可扩展应用程序提供了运行环境和编程模型。SaaS 供应商可以根据需求弹性设计软件，而不需要用户提供用于实现可扩展功能的硬件和应用程序。

终端用户利用云计算能随时随地通过各种设备访问数据并对数据进行操作。存储在云中的信息和服务通过 Web 页面呈现给用户，用户可以使用便携式设备或家中的台式机来访问。

由于常用软件（办公自动化、照片编辑、信息管理等）也包含在云中，因此终端用户可以用这些软件处理任务，而过去完成这些任务则需要相当大的软件投资。云计算模式下的成本非常小，因为云服务供应商的运行成本由其服务的所有用户分担，使得共享基础设施的运营能力和灵活性得到更好的利用。将 IT 基础设施和服务部署在大型数据中心，极大地优化了资源分配以及能源效率，并减少了环境污染。

最后，面向服务和按需访问为灵活地构建系统和应用程序带来了新的机遇，如果没有云计算技术，这一切是不可能实现的。可以在整合现有服务的基础上着重增加新价值来创建新服务。由于云计算可以按需提供参考模型中的任何组件，因此我们只要专注于附加值层面，就可以用极小的投入将想法转化为产品。

1.1.6　面临的挑战

任何新技术的发展和流行都会遇到新问题，云计算也不例外。云共同体随时会面临新的、有趣的问题和挑战，这一共同体包括 IT 从业者、管理者、政府和监管机构。

除了系统配置、网络和规模等实际问题之外，与云计算服务和资源的动态提供有关的一系列新问题也出现了。例如，为获得最大收益，IaaS 需要配置多少资源？这些资源应该使用多长时间？云服务供应商在管理大型的计算基础设施和资源虚拟化技术方面也面临着挑战。另外，还应该从安全性和合法性的角度考虑真实与虚拟基础设施的集成问题。

云环境中的安全性、保密性以及数据的保护是另一项重要的挑战。使用云服务的机构并不拥有用来处理数据和存储信息的基础设施，这使得机密数据的保护面对挑战，机密数据泄露的后果是任何机构都无法承受的。因此，需要寻求确保数据机密性且符合安全标准的方法，这是对云计算系统的信息处理的最低要求。这个问题并不那么明显，尽管密码可以保护从客户端到云硬件设备之间的数据传输，然而为了处理信息，还需要在内存中进行解密。这是整个过程中的薄弱环节，因为虚拟化允许透明地获取实例内存页，恶意的商家可以轻易地得到这些数据。

还可能出现法律问题。云计算具有无处不在的特性，即计算基础设施部署在不同的地理位置，这会带来法律问题。不同国家关于隐私有不同的法律法规，可能会在第三方（包括政府部门）对数据拥有什么权利的问题上引发潜在的争议纠纷。众所周知，当可疑操作可能会对国家安全造成威胁时，美国立法赋予了政府机构极端的权力去获取机密数据。而欧洲国家则更加严格地保护隐私权。当美国机构使用云服务将其数据存储在欧洲时，有趣的情况出现了，如果美国政府对该机构产生怀疑，它也很难甚至不可能对位于欧洲云数据中心存储的数据进行控制。

1.2　云计算起源

利用大量分布式计算设施租用计算服务的理念已经存在很长一段时间了，其历史可以追溯到 20 世纪 50 年代初。技术的不断发展和完善为实现云计算创造了一系列有利条件。

图 1-6 描述了分布式计算技术的演化发展及其对云计算产生的影响。通过追溯技术发展历程，我们简要地回顾了在实现云计算过程中发挥了重要作用的五大核心技术：分布式系统、虚拟化、Web 2.0、面向服务的计算和效用计算。

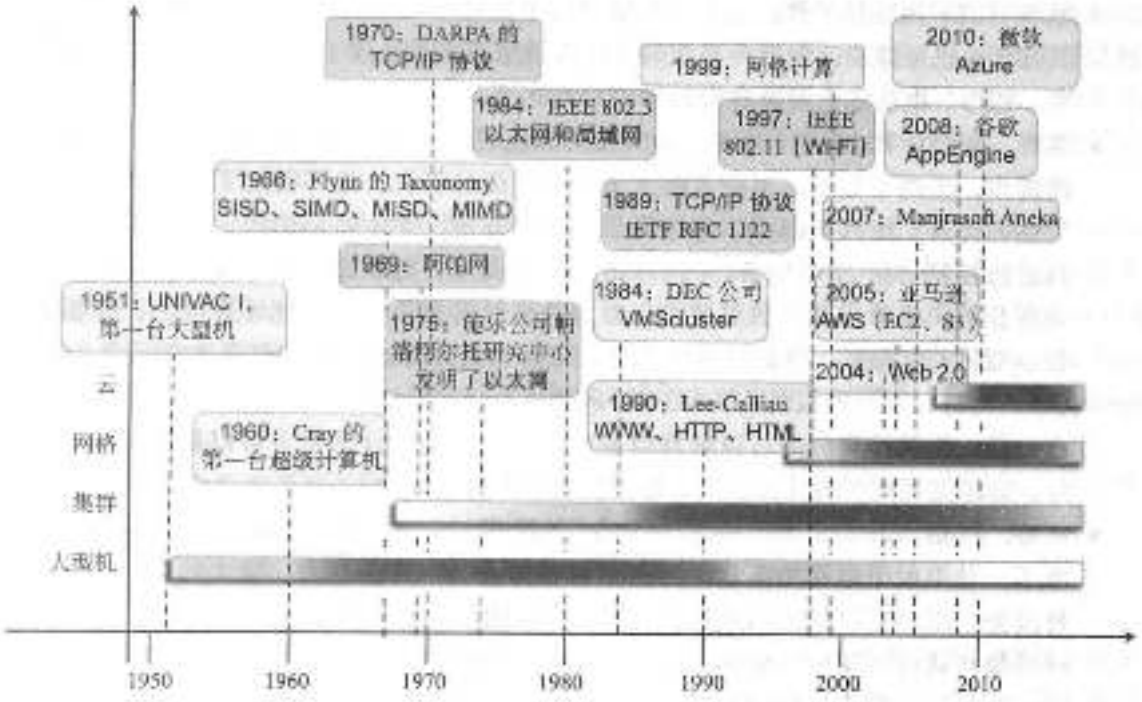


图 1-6 分布式计算技术的演进（1950 ~ 2010）

1.2.1 分布式系统

云本质上是大型的、可以按需为第三方提供服务的分布式计算系统。可参考 Tanenbaum 等人 [1] 提出的分布式系统的特性：

> 分布式系统是一些独立计算机的集合，但是对这个系统的用户来说，系统就像一台计算机一样。

15

这是一个包括各种计算系统的通用定义，说明了分布式系统的两个非常重要的基本特征：由多个独立的计算机系统组成，用户面对的是一个整体。该定义也适用于云计算，云计算隐藏了系统的复杂架构，给用户提供了单个系统视图。分布式系统的主要目的是共享资源，并更好地利用资源。云计算体现了这一观念，并且将其发挥到了极致，将资源（基础设施、运行环境以及服务）出租给用户。事实上，IT 巨头（亚马逊、谷歌等）提供大型计算设施服务是驱动云计算发展的因素之一，把计算能力作为一种服务提供给用户，能更有效地利用计算基础设施。分布式系统具有一些特性，如异构性、开放性、可扩展性、透明性、并发性、持续可用性以及故障独立性。从某种程度上讲，这也体现了云计算的特性，尤其是可扩展性、并发性和持续可用性。

三大里程碑技术奠定了云计算的基础：大型机计算系统、集群计算和网格计算。

- 大型机。大型机利用多个处理单元完成计算，是具有超级计算能力以及高可靠性的计算机，适合大量数据传输和大规模 I/O 操作。大型企业用大型机系统完成大数据量处理任务，如在线交易系统、企业资源计划系统，以及其他涉及大量数据处理的操作。尽管大型机不是分布式系统，但是它采用多个处理器为用户提供强大的计算能力，并且作为一个单独的实体呈现给用户。大型机最吸引人的特点之一是高可靠性，具有“不间断”服务和显而易见的容错能力。在更换故障组件时不用关闭系统，

16

系统可以不间断地工作。大型机主要应用于批处理任务。现在，大型机已不太流行，其部署也已经减少，但这种系统的升级版仍用于事务处理（例如，网上银行、航空订票、超市、电信运营商及政府服务）。

- 集群。集群计算[3][4]最初作为大型机和超级计算机之间的低成本选择。随着技术的进步，出现了更快、更强大的大型机和超级计算机，最终促进了低价商业机的可用性的提升。这些机器可以通过高带宽网络连接，并由特定的统一系统管理软件工具进行控制。从20世纪80年代开始，集群成为并行和高性能计算的技术标准。作为商业机，集群比大型机更便宜，并为众多机构提供了高性能计算能力，包括大学和小型研究实验室。集群技术大大促进了分布式计算系统工具和框架的形成，包括：Condor[5]、并行虚拟机（PVM）[6]和消息传递接口（MPI[⊖]）[7]。集群的显著特点之一是，集群计算能力可以处理以前只有昂贵的超级计算机才可以解决的问题。此外，如果需要更多的计算能力，集群扩展很容易实现。
- 网格。网格计算[8]出现在20世纪90年代初，由集群计算演进而来。网格计算提出了一种类似于电网的新计算方法，以获得强大的计算能力、庞大的存储设施及各种服务。用户可以像使用其他公共服务（如能源、燃气和水）一样来“消费”资源。网格最初通过互联网将地理上分散的集群节点连接在一起。这些集群属于不同的机构，通过对它们进行管理来共享计算能力。与“大集群”不同，计算网格是异构计算节点的动态聚合，其规模是全国性的甚至是世界性的。以下几个原因使计算网格的大规模应用成为可能：集群普遍存在；集群往往利用不足；单一集群不足以提供解决新问题所需要的计算能力；网络的发展和互联网的大规模使用使得远距离高带宽连接成为可能。这些因素促进了网格的发展，使其服务于当今世界各地的众多用户。

17

通常认为云计算是网格计算的继任者，它体现了以上三大技术的各个方面。机构托管的大型数据中心部署云计算环境为用户提供服务。正如大型机一样，云的特征是拥有几乎无限的容量、很强的容错能力和不间断的服务。在许多情况下，构成云计算基础设施的计算节点是由商业机组成的集群。云供应商提供的服务按使用量计费，充分体现了网格计算所提出的效用计算的构想。

1.2.2　虚拟化

虚拟化是云计算的另一个核心技术。虚拟化是将一些计算的基本构件（如硬件、运行环境、存储和网络）进行抽象化的方法。虚拟化技术的存在已经超过40年了，但其应用一直受到技术限制，不能得到有效利用。目前，这些技术限制已经基本解决，虚拟化成为支撑云计算最重要的技术基石，对于按需提供IT基础设施的解决方案尤为重要。虚拟化带来了一定程度的可自定义性和可控性，使得云计算对用户而言充满吸引力。同时，对云服务供应商而言，采用云计算也可以实现持续发展。

虚拟化实质上是一种创建不同计算环境的技术。之所以称为虚拟环境，是因为其模拟了用户所需要的接口。最常见的虚拟化实例是硬件虚拟化，它可以模拟多个操作系统界面。硬

⊖ MPI是多计算机之间互相通信的应用程序接口（API）规范。MPI定义了语言无关协议，支持点到点以及多点间通信。MPI为高性能、可扩展、便携应用而设计，目前已成为开发并行应用的主要范式。

件虚拟化允许在单个物理平台上运行不同的软件堆栈。这些软件堆栈包含在内部的虚拟机实例中，其操作彼此完全隔离。高性能服务器可以管理多个虚拟机实例，从而为按需定制软件堆栈提供了可能。这是云计算解决方案可以按需提供虚拟服务器的基本技术，如亚马逊EC2、RightScale、VMware vCloud等。硬件虚拟化、存储和网络虚拟化一起，构成了仿真IT基础设施所需的技术。

虚拟化技术也可用于复制程序的运行环境。在Java或.NET等技术的流程虚拟机应用中，应用程序不在操作系统中执行，而是在称为虚拟机的特定程序环境中执行。这种技术允许隔离应用程序的执行，并对它们所访问的资源进行更加精确的控制。相对于硬件虚拟化，流程虚拟机提供了更高层次的抽象，因为它要处理的对象仅由一个应用程序构成，而不是各种软件堆栈。云计算中采用这种方法来为按需扩展的应用程序提供平台，如谷歌AppEngine和Windows Azure。

具有对性能影响很小的隔离和可定制的环境，使虚拟化成为非常流行的技术。云计算利用虚拟化技术构建平台服务，为世界各地的众多用户提供按需定制的虚拟化服务。 18

1.2.3 Web 2.0

Web是云计算提供服务的主要接口。目前，它包含了一整套技术和服务，能够实现交互信息的共享、协同、以用户为中心的设计和应用组合。这些功能将Web转化为用于应用程序开发的丰富平台，称为Web 2.0。这个术语体现了一种新的途径，开发人员可以构建应用并通过互联网提供服务，为用户提供全新的服务体验。

Web 2.0技术使Web页面具有交互性和灵活性，桌面应用程序中的所有常见功能都可以基于Web访问，提升了用户体验。这些功能是通过集成一系列的标准和技术来实现的，如XML、异步JavaScript和XML（AJAX）、Web服务等。这些技术允许利用用户提交的内容来构建应用，用户变成了内容供应商。此外，互联网技术的广泛应用为网络开辟了新的机遇和市场，使用户可以通过各种设备获得服务，如移动电话、汽车仪表盘、电视机等。这种新应用场景要求应用程序具有更强的动态特性，这是Web 2.0技术的另一关键要素。Web 2.0应用程序是动态的，随着用户需求的不断变化，Web应用程序得到不断改进，新功能不断增加，而不需要在客户端安装新的软件版本。通过与云应用程序交互，用户可以轻松地利用软件的新功能。轻量级部署和编程模型可以有效地支持这种动态特性。松耦合是Web 2.0的另一基本属性，通过组合现有服务并将它们集成在一起，可以轻松地“合成”新的应用程序，从而带来新附加值，更容易满足用户的需求。最后，Web 2.0应用程序的目标是依靠广大互联网用户的“长尾效应”，使其无论在媒体访问方面还是费用方面都可以被用户接受。

Web 2.0应用程序的例子有：谷歌文档、谷歌地图、Flickr、Facebook、Twitter、YouTube、de.li.cious、Blogger和维基百科。特别是在社交网站中，Web 2.0技术应用最多。例如Facebook、Flickr等网站的信息交互程度，如果没有AJAX、RSS（Really Simple Syndication）和其他有效的信息交互工具的支持，是不可能实现的。此外，社区网站充分利用了社区群体智慧为应用程序提供内容：Flickr提供用于存储数字图片和视频的高端服务；Facebook是一个利用用户活动提供内容的社交网站；Blogger像其他博客网站一样，用户可在线编写日记。

Web作为一种传送方式能够完成并加强信息交互，这一想法于1999年由Darcy

DiNucci[⊖]提出，并从2004年开始得到充分认识。今天，Web是支持云计算需求的成熟平台，
19 充分利用Web 2.0应用程序和框架来提供丰富的互联网应用（RIA），这是为更广泛的公众提供云服务的基础。从社会角度看，Web 2.0应用程序做出了巨大贡献，使得人们更习惯于在日常生活中使用互联网，并且为接受云计算服务模式开辟了道路，即使是IT基础设施也可以通过Web接口提供。

1.2.4 面向服务的计算

面向服务是云计算系统的核心参考模型，它将服务作为应用和系统开发的主体模块。面向服务的计算（SOC）支持快速、低成本、灵活、可交互和可扩展的应用和系统开发[19]。

服务是对自描述和平台无关组件的抽象描述，从简单功能到复杂业务流程都可以以服务形式来实现。事实上，任何一段完成某一任务的代码都可以转换成服务，并通过网络访问协议发布其功能。服务应该是松耦合、可重用、独立于编程语言和位置透明（不需要知道提供服务的位置）的。松耦合使得服务更容易满足不同的场景并且可重用，独立于特定的平台增加了服务的可访问性，因此可以服务于更广泛的用户。用户查找在全球管理机构注册的服务，然后以位置透明的方式消费服务。服务采用面向服务的架构（SOA）[27]来组成和聚集。SOA是一种通过已发布和可发现的接口构建软件系统，为终端用户或网络用户提供服务的逻辑方式。

面向服务的计算引入和推广了两个重要的概念，服务质量（QoS）和软件即服务（SaaS），这两个概念也是云计算的基础。

- QoS定义了可从不同角度评估服务行为的功能性和非功能性属性集合。QoS可以是性能指标，如响应时间、安全属性、事务完整性、可靠性、可扩展性和可用性。QoS需求由客户和供应商之间建立的服务等级协议（SLA）描述，是在使用服务时需要满足的QoS属性的最小值（或可接受的范围）。
- SaaS的概念引入了应用程序的新交付模式。该术语源于应用服务供应商（ASP），ASP在其中央数据中心提供跨广域网的基于软件服务的解决方案，并且允许用户基于订阅或租用方式来获取。ASP负责维护硬件资源并提供应用程序，使客户摆脱系统维护成本和升级的困扰。这种软件交付模式是可行的，因为通过多租户的方式实现了规模经济。基于面向服务的计算（SOC）方式，SaaS获得了全面发展，松耦合的软件组件可以单独提供和定价，而不是处理整个应用程序。这样，复杂的业务流程和交易都以服务形式交付，同时应用程序可以随需应变，任何人在任何地方都可以
20 重复使用这些服务。

面向服务最通常的表示方法是Web Services（WS）[21]。WS将SOC的概念引入到万维网中，使得应用程序可以被处理成服务。Web服务是软件组件，通过使用超文本传输协议（HTTP）的方法调用模式来发布和交付服务。Web服务的接口可以通过由Web服务描述语言（WSDL）[22]表示的元数据语义来编程实现。WSDL是一种XML语言，定义了服务和方法的特征，以及参数、描述和返回类型，以服务的形式发布，并通过简单对象访问协议

⊖ 在《Design & New Media》杂志中，Darcy DiNucci将Web描述为："我们所了解的安装于Windows浏览器中的Web本质是静态页面，这只是Web技术的初期。Web 2.0技术的出现带来了新的曙光，见证了Web技术的发展。Web不再只是文本和图形的页面显示，而是作为信息交互的传输机制。Web信息可以在计算机、电视机、汽车仪表盘、手机、手持游戏机甚至微波炉上显示。"

（SOAP）[23] 与 Web 服务进行交互。XML 语言定义了如何调用 Web 服务的方法，并返回结果集。通过在 HTTP 上使用 SOAP 和 WSDL，Web 服务具有平台独立特性，并可以通过 Web 访问。Web 服务的标准和规范由 W3C 制定，最常用的开发 Web 服务的架构包括 ASP.NET[24] 和 Axis[25]。

分布式服务系统的组合是 SOC 在实现云计算的过程中产生的重大贡献。Web 服务技术提供了合适的工具，使多服务组合变得简单，并且更容易集成到主流的万维网（WWW）环境中。

1.2.5 效用计算

效用计算定义了一种计算服务的提供模式，将存储、计算能力、应用程序和基础设施等资源封装为服务，并基于使用量付费。把计算作为像天然气、水、电和电话一样的公共基础设施的想法已经存在很长时间了，但是直到今天，随着云计算的到来，这个想法才变为了现实。提出这一设想的先行者美国科学家 John McCarthy 于 1961 年在麻省理工学院百年庆典上讲道：

> 如果我所提倡的那种计算方式成为未来的计算机，那么有一天，计算可能会被构建成公共基础设施，就像电话系统那样……计算基础设施服务可能会成为一个新的重要产业的基础。

这种计算服务提供模式的起源可以追溯到大型机时代，IBM 和其他大型机供应商为银行和政府等机构的数据中心提供大型机。基于效用计算的商业模式带来了新的需求，并促进了大型机技术的改进：增加了操作系统、控制处理和用户计量设施等新功能。随着集群计算的出现，计算作为公共基础设施的想法从业务领域扩展到学术界。不仅是企业，科研院所也产生了根据需要利用外部 IT 基础设施的想法。计算科学作为建立计算集群的主要推动因素之一，仍然需要巨大的计算能力去解决具有挑战性的难题，而且并非所有机构都能够独立拥有满足计算需求的设施，通常需要利用外部集群系统进行计算。互联网和 Web 的广泛应用，提供了通过简单接口在世界范围内实现效用计算的技术手段。前面已经提到，计算网格提供的全球规模的分布式计算基础设施可以实现按需访问。计算网格把效用计算的概念提升到一个新的水平：面向市场 [15]。随着效用计算的广泛普及，易于建立竞投或出售存储、计算、服务等网格产品的交易系统架构。此外，电子商务技术 [25] 为效用计算提供了支持。在 20 世纪 90 年代中后期，公众对于在网上购买各种商品产生了巨大的兴趣：食品、服装、多媒体产品以及诸如存储空间和 Web 托管之类的在线服务。在互联网泡沫（dot-com bubble）[⊖] 破灭后，人们的在线购物兴趣降低了，但这种现象使得大众更热衷于购买在线服务。其结果是，通过信用卡进行网上支付的系统架构更加方便，经受了实践的考验。

从应用程序和系统开发的角度来看，面向服务的计算和面向服务的架构（SOA）引入了利用外部服务执行软件系统内部特定任务的理念。应用程序不仅是分布的，而且由不同实体提供的服务可以组成一个服务网，并可以付费使用这些通过互联网访问的服务。SOC 拓宽了有关计算机系统中什么可以作为基础设施服务被访问的概念：不仅是计算能力和存储，服务和应用程序组件也可以按需使用和集成。随着计算服务的发展，QoS 成为一个重要研究

21

⊖ 互联网泡沫现象始于 20 世纪 90 年代后期，在 2000 年到达顶峰。在此期间，大多数公司都在开展在线服务和电子商务业务，并且迅速扩张，以至于后期无法维持其业务增长，结果多数公司倒闭，一部分原因是公司收入不能支持其业务支出，或是因为没有足够的客户需求来支持其拓展业务。

课题。

这些因素促进了效用计算概念的发展，并提出了实现云计算的重要步骤。在云计算中，效用计算的概念得到了充分体现。

1.3　构建云计算环境

云计算环境的构建既包括基于云计算解决方案的应用和系统开发，也包括框架、平台及交付云计算服务的架构。

1.3.1　应用程序开发

面向云计算的应用开发可以从按需动态扩展的能力中获益。其中一类最能充分利用这一特点的应用是 Web 应用程序，其性能通常受到由不同的用户需求产生的工作负载的影响。随着 Web 2.0 技术的推广，Web 已经成为开发丰富而复杂的应用的平台，包括利用 Internet
[22] 进行服务交付和用户交互的企业应用。这些应用的特点是处理过程复杂，由用户信息交互触发，通过 Web 前端后面几个层次之间的交互进行开发。这些应用程序大多对不合适的系统架构、服务部署或者负载变化敏感。

另一类体现云计算潜在优势的应用是资源密集型应用程序，包括数据密集型和计算密集型应用程序。在这两种情况下，需要相当多的资源在一段合理的时间内来完成执行。值得注意的是，并不是持续地、长时间地需要大量资源。例如，科学应用程序可能需要巨大的计算能力，在一段时间内进行一次大规模的实验，所以购买支持实验的硬件设备是不可行的。在这种情况下，云计算可以作为解决方案。资源密集型应用程序不是交互式的，其特点大多是批处理。

云计算提供了对整个参考模型按需创建和动态扩展的解决方案。实现方式包括：①提供租用计算能力、存储和连接网络的方法；②提供支持可扩展性和动态性的运行环境；③提供模仿桌面应用程序的服务，但是这些服务完全由服务供应商来托管和管理。基于面向服务架构，所有功能都可以简单地、无缝地集成到现有系统中。开发人员通过简单的 Web 界面，利用表征状态转移（Representational State Transfer，REST）Web 服务来实现这些功能。众所周知的抽象能力使得云应用程序和系统的开发和管理变得实用而简单。

1.3.2　基础设施和系统开发

分布式计算、虚拟化、面向服务和 Web 2.0 构成了在全球各个地方提供云服务所需要的核心技术。开发基于云模式的应用程序和系统需要掌握这些技术。此外，需要从设计和开发的角度处理新的问题和挑战。

分布式计算是云计算的基础模型，因为云计算系统是一种分布式系统。除了大多与访问云中资源相关的管理任务之外，云系统的极端动态特性，即按需提供新的节点和服务，是工程师和开发人员面临的主要挑战。这是云计算解决方案特有的性质，并且大多需要在计算系统的中间件层处理。IaaS 解决方案具有添加和删除资源的功能，由那些能够明智地、高效地在可扩展的 IT 基础设施上部署系统的用户进行资源操作。PaaS 解决方案将算法、配置过程的控制规则和资源租赁规则嵌入核心产品中。这些对开发人员可以是完全透明的，或得到较好的控制。云资源和已有系统部署之间的集成是另一个值得关注的因素。

云计算服务采用 Web 2.0 技术接口进行交付、管理和配置。除了通过 Web 浏览器与各

种接口交互之外，从编程的角度来看，Web 服务已经成为云计算系统的主要访问方式。因此，面向服务是定义云计算系统体系结构的基础。云计算经常表示为 XaaS，即一切皆作为服务，明确地强调了面向服务的核心作用。尽管访问不同云服务供应商的资源没有统一的标准，但是互通性技术降低了学习的困难，并且将云计算集成到现有系统中变得简单了。 23

虚拟化是云计算的另一个重要技术基础。这项技术是云服务供应商使用的硬件基础设施的核心特色。如前所述，虚拟化概念已有 40 多年的历史，但云计算带来了新的挑战，特别是在管理虚拟环境上，不论是虚拟硬件的抽象，还是运行环境的抽象。云应用程序的开发者需要注意所采用虚拟化技术的局限性，以及系统中一些组件的波动性带来的影响。

这些都是我们编写基于云计算技术的应用程序和系统时需要考虑的因素。云计算本质上提供了一种机制，通过复制计算机系统在高负载状态下所需的组件来解决需求的激增。组件的动态性、可扩展性和波动性是指导系统设计的主要因素。

1.3.3 云计算平台和技术

利用提供不同类型服务的平台和框架，目前已开发了多种云计算应用程序，从 IT 基础设施服务到用户定制的应用服务。

1. 亚马逊 Web 服务（AWS）

AWS 提供全面的云计算 IaaS 服务，从虚拟计算、存储、网络连接到云参考模型的各层服务。AWS 主要以其按需的计算和存储服务而闻名，即弹性计算云（EC2）和简单存储服务（S3）。EC2 为用户提供了可定制的虚拟硬件，可作为在云中部署计算系统的 IT 基础设施。用户可以从大量虚拟硬件配置（包括 GPU 和集群实例）中选择资源，通过访问 AWS 服务的 Web 门户网站 AWS 控制台部署 EC2 实例，或者通过使用支持多种编程语言的 Web 服务 API 部署 EC2 实例。EC2 还具有把特定的运行实例另存为图像的功能，从而允许用户创建自己用于部署系统的模板。这些模板存储在 S3 中，可以提供随需应变的持久性存储服务。S3 划分为不同的桶（bucket），这些桶是以二进制形式存储对象的容器，含有丰富的属性。用户可以存储任意大小的对象，从简单文件到整个磁盘镜像，并可以从任何地方访问这些存储实体。

除了 EC2 和 S3 之外，还可以利用广泛的服务来构建虚拟计算系统，包括网络支持、缓存系统、DNS、数据库（关系和非关系）支持等。 24

2. 谷歌 AppEngine

谷歌 AppEngine 是一个用于执行 Web 应用程序的可扩展运行环境，它利用了谷歌大型计算基础设施的优势，随时间变化按需求动态扩展。AppEngine 提供了安全的执行环境和各种服务的集合，简化了可扩展的高性能 Web 应用程序的开发。这些服务包括：在内存中缓存、可扩展的数据存储、作业队列、消息传递和后台任务。开发人员可以使用 AppEngine 软件开发包（SDK）在自己的机器上构建和测试应用程序，SDK 复制了系统运行环境，有助于测试和配置应用程序。一旦开发完成，开发人员可以轻松地将其应用程序迁移到 AppEngine 上，设置服务价格，使其服务在世界范围内可用。AppEngine 目前支持的语言包括 Python、Java 和 Go。

3. 微软 Azure

微软 Azure 是一个云操作系统和一个用于开发云应用程序的平台，一般为 Web 应用和分布式应用提供可扩展的运行环境。Azure 上的应用程序以角色的概念组织起来，用以标识

应用的一个分配单元，体现应用的逻辑。目前，有三种类型的角色：Web 角色、工作者角色，虚拟机角色。Web 角色用来承载 Web 应用程序；工作者角色是一个更通用的应用程序容器，还可以用来执行工作负载的处理；虚拟机角色提供了一个可以完全定制云参考模型中各层服务（包括操作系统）的虚拟环境。除了这些角色之外，Azure 还提供了一套执行补充应用的附加服务，如用于存储（关系数据和非关系数据）的支持、网络、缓存、内容服务等。

4. Hadoop

Apache Hadoop 是一个开源框架，适于处理商用机上的大型数据集。Hadoop 是谷歌设计的应用程序编程模型 MapReduce 的实施方案，MapReduce 提供了数据处理的两种基本操作：map 和 reduce。前者转换并合成用户输入的数据，后者聚合由 map 操作得到的输出结果。Hadoop 提供了运行环境，开发人员只需要提供输入数据，指定需要执行的 map 和 reduce 函数。Yahoo! 是 Apache Hadoop 项目的倡导者，致力于将该项目应用于数据处理的企业级云计算平台。Hadoop 是 Yahoo! 云计算基础设施的组成部分，支持该公司的多项业务处理。目前，Yahoo! 管理全球最大的 Hadoop 集群，也供学术机构开展研究工作。

5. Force.com 和 Salesforce.com

Force.com 是用于开发社交网络应用程序的云计算平台。作为 SalesForce.com 的基础平台，SalesForce.com 是客户关系管理的 SaaS 解决方案。开发者可以用 Force.com 组合已有模块创建应用程序；可以使用一套支持企业所有业务活动的完整软件组件。用户也可以开
[25] 发自己的组件，或整合 AppExchange 中已有的组件创建自己的应用程序。该平台为应用开发提供了全面支持：从数据布局的设计到业务规则和工作流程的定义，以及用户接口的定义。Force.com 平台完全托管在云端，可通过 Web 服务技术获取其功能组件并在应用程序中实现。

6. Manjrasoft Aneka

Manjrasoft Aneka[165] 是用于快速创建可扩展应用的云应用平台，可以无缝且弹性地部署到不同类型的云平台。Aneka 支持各种开发应用程序的编程抽象模型，并提供可以部署在异构硬件（集群、联网的台式计算机和云资源）上的分布式运行环境。开发人员可以选择不同的程序抽象模型来设计应用程序：任务、分布式线程和 map-reduce。然后，应用程序在面向服务的分布式运行环境中执行，在该运行环境中可以动态按需集成附加资源。面向服务的体系结构具有极大的灵活性，并易于集成新的功能，如新的编程模型的抽象和相关的运行管理环境。在运行过程中对服务的管理主要包括：调度、执行、记账、结算、存储和 QoS。

以上这些平台是应用云计算技术的主要实例，涵盖了云计算参考模型中的三大部分：IaaS、PaaS、SaaS。本书使用 Aneka 作为参考平台，探讨分布式应用程序的实际实现过程，给出利用 Aneka 提供的各种编程模型和抽象实例创建云计算应用的不同方法。

本章小结

本章讨论了云计算的发展前景和机遇，及其特点和面临的挑战。云计算模式是支撑其发展的模型和技术逐渐成熟和融合的结果，包括分布式计算、虚拟化、Web 2.0、面向服务和效用计算。

目前没有统一的云计算概念。本书将探讨云计算不同的定义、解释及实现方法。在云计算的所有不同解释中，共同的特性是，云系统支持 IT 服务（不论是虚拟基础设施、运行环境还是应用服务）的动态交付，并采用了基于效用的成本模型对这些服务定价。这一理念贯

穿于整个计算层次中，为开发可扩展应用及其服务动态地提供基于云托管平台的IT基础设施和运行环境服务。这一构想促进了云计算参考模型的产生。该模型确定了云计算的三类主要市场划分（和服务提供）：IaaS、PaaS和SaaS。它们对应着云计算所提供的不同服务的广义分类。

26

云计算的长远目标是充分实现公共基础设施服务模型以促进服务的提供。可以预见，随着新技术的发展和对云计算交付模式理解程度的加深，将会形成计算公共基础设施交易的全球市场。有关这一领域的研究称为面向市场的云计算，面向市场进一步强调了云计算服务是作为公共基础设施服务进行交易的。要实现这一构想仍然需要很长时间，但云计算已经带来了经济、环境和技术利益。通过将IT资产变成公共基础设施服务，企业能够降低运营成本、增加收入。云计算的各种优点在实际应用过程中还没有完全体现出来。安全性和合法性是云计算在技术层面之外需面临的挑战。

从软件设计和开发的角度来看，新的挑战出现在工程计算系统中。云计算提供不同技术的融合，利用这些技术是一项具有挑战性的工程任务。云计算为软件应用架构和系统体系结构设计带来了新的机遇、技术和方法。必须考虑的一些关键因素是：虚拟化、可扩展性、动态配置、大数据集以及费用模型。为了更好地理解这些概念，本书将以Aneka作为参考平台来说明云系统和应用编程环境。

习题

1. 云计算的创新性是什么？
2. 云计算依赖的技术有哪些？
3. 简要描述分布式系统特性。
4. 云计算的定义及其核心特征是什么？
5. 促进云计算发展的主要分布式计算技术有哪些？
6. 什么是虚拟化？
7. Web 2.0技术带来的重大变革是什么？
8. 举几个Web 2.0应用程序的例子。
9. 描述面向服务的主要特点。
10. 什么是效用计算？
11. 描述云计算的发展前景。
12. 简要概括云计算参考模型。
13. 云计算的主要优势是什么？
14. 简要概括云计算面临的挑战。
15. 云应用开发和传统软件开发有什么区别？

27
|
28

并行计算与分布式计算原理

云计算是一种新的技术趋势，可以更好地利用 IT 基础设施、服务和应用。云计算采用了一种按使用付费的服务交付模式，用户无需拥有自己的基础设备、平台或应用，只在需要时使用这些服务即可。这些 IT 基础设施由将其连入互联网的服务供应商所有和维护。

本章主要阐述并行分布式计算的基本原理，讨论用于构建云计算系统和应用的模型和基本概念。

2.1 计算时代

串行和并行是两种基本的主要计算模型。串行计算起源于 20 世纪 40 年代，比并行（分布式）计算早了近十年（见图 2-1）。当时，架构、编译器、应用程序和问题解决环境成为计算发展的四个关键要素。

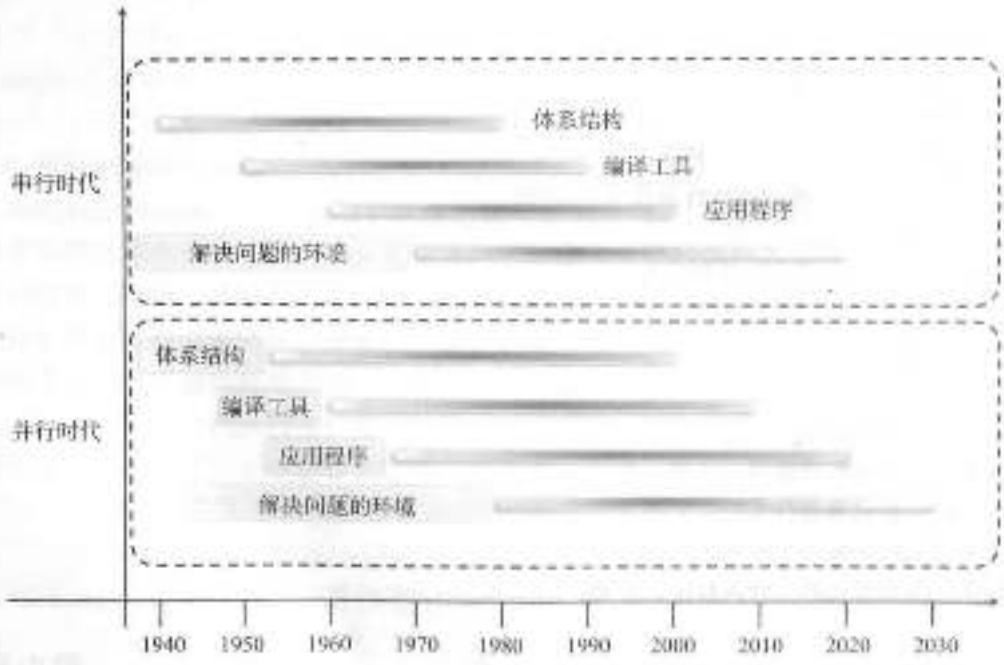


图 2-1 计算时代（1940 — 2030）

计算时代的兴起离不开硬件架构的发展，最终产生了系统软件，特别是在编译器和操作系统领域，实现了系统管理和应用开发。应用和系统的研发是最令人感兴趣的地方，当设计出问题解决环境并且可以为工程师们提供便利时，这种研发就会被逐渐整合。这标志着计算范式已经成熟并成为主流。另外，计算时代的每个方面都经历着三个阶段：研究和开发（R&D）、商业化、商品化。

2.2　并行计算与分布式计算

虽然并行计算和分布式计算存在细微的差别，但两个概念往往可以互换。并行代表一类紧耦合系统，而分布式则代表更广泛的一类系统，包括紧耦合系统。 29

更准确地说，并行计算指的是将计算任务分配给几个共享相同内存的处理器的计算模式。并行计算系统的架构通常表现为组件的同构性：每个处理器都是相同类型的，且拥有相同的处理性能。共享内存有一个独立的地址空间，可供所有处理器访问。并行程序被分成若干执行单元并分配给不同的处理器，它们之间依靠共享内存相互通信。起初，只有具有共享同一物理内存的多处理器的架构才可称为并行系统。随着时间的推移，这些限制条件已经放宽，只要是基于共享内存这一概念的架构，无论是物理内存系统，还是由库、特定的硬件和高效的网络基础设施组成的系统，都可以称为并行系统。例如，一个集群中节点通过无限带宽网络连接，且配置了分布式共享内存系统，就可以称作并行系统。

分布式计算是指那些将计算任务进行划分，并在不同计算单元中同时执行的架构或系统。不论计算单元是不同节点上的处理器，或是同一计算机上的处理器，再或是同一处理器中的内核。因此，相比并行计算，分布式计算包含的系统和应用程序的范围更广，是更为通 30
用的概念。尽管没有规定，但分布式这一术语通常意味着计算单元的位置不同，且这些单元在硬件和软件功能上也可能各不相同。典型的分布式系统实例是计算网格或互联网计算系统，分布式系统在全球范围内集成各种架构、系统和应用。

2.3　并行计算基本要素

现在人们清楚地知道，硅处理器芯片已经逐渐达到其物理极限。处理器的处理速度受制于光速，处理器中封装晶体管的密度也受到热力学温度极限的约束。为了克服这些限制，人们将多个处理器连接起来互相协调工作，成功地解决了这个难题。在这一方向上的初步探索引领了并行计算的发展，包括技术、架构和用于并行执行多个任务的系统。正如之前所述，并行计算和分布式计算在概念上没有十分清晰的界定，前者经常用来代替后者。本节将介绍并行计算的特性，以及协调多个处理器工作的单个计算机中的并行性。

2.3.1　什么是并行处理

同时在多个处理器上处理多个任务称为并行处理。并行程序由多个活跃进程（任务）组成，同时解决给定的问题。运用分而治之方法将一个给定的任务分为多个子任务，每个子任务在不同的中央处理单元（CPU）中进行处理。运用分而治之方法在多处理器系统上编程称作并行编程。

传统的串行通信计算机所能提供的计算能力不足以满足如今许多应用程序的需要。并行处理通过增加计算机中 CPU 的数量，并在 CPU 之间建立高效的通信系统，可以有效地解决这个问题。不同处理器之间分担了工作量，可以得到比单处理器系统更强的计算能力和性能。

很多因素影响着并行处理的发展，最主要有以下几点：

- 在科学和商业计算领域，计算需求不断增加。在专业计算领域，如生命科学、航空航天、地理信息系统、机械设计和分析等领域，需要高速计算能力。
- 由于受到光速和热力学定律的限制，串行架构逐渐达到物理极限。串行 CPU 的处理

速度也达到饱和（不能再垂直增长）。因此，并联多个 CPU 成为另一种提高计算速度
31 的方法（有可能水平增长）。

- 运用流水线技术和超标量技术等设计的硬件不可扩展，且需要先进的编译技术。此类编译技术的研发是一项艰巨的任务。
- 运用向量处理解决这类问题效果很好。该方法适用于解决大量科学问题（包括大矩阵运算）和图形处理。但在其他领域并不适用，例如数据库。
- 并行处理技术成熟且可进行商业开发；在开发工具及开发环境方面，已经展开了卓有成效的研发工作。
- 网络技术的长足发展为异构计算铺平了道路。

2.3.2　并行处理硬件架构

并行处理的核心元素是 CPU。根据可以同时处理的指令流和数据流的数量，计算机系统可以分为以下四类：

- 单指令流单数据流（SISD）系统。
- 单指令流多数据流（SIMD）系统。
- 多指令流单数据流（MISD）系统。
- 多指令流多数据流（MIMD）系统。

1. SISD 系统

SISD 计算系统是一个能在单数据流上执行单指令的单处理器机器（见图 2-2）。在 SISD 系统中，机器指令按顺序进行处理，因此采用这种模式的计算机通常称为串列计算机。大多数传统计算机采用 SISD 模型构建。所有需要处理的指令和数据必须存放在主存储器上。SISD 模型中，处理单元的速度受到计算机内部信息传递速率的限制。典型 SISD 系统有
32 IBM PC、Macintosh 和工作站。



图 2-2　SISD 构架

2. SIMD 系统

SIMD 计算系统是可以在不同的数据流上操作而在多 CPU 上执行同一指令的多处理器机器（见图 2-3）。由于包括大量的向量和矩阵运算，所以基于 SIMD 模型的机器适用于科学计算。例如，语句

$$Ci = Ai \times Bi$$

可以传递给所有处理单元（PE），向量 A 和 B 中有组织的数据元素可以分成多组（N 组对应 N 个 PE 系统），一个 PE 可以处理一个数据集。典型 SIMD 系统有 Cray 的向量处理机和 Thinking Machines 的 cm*。

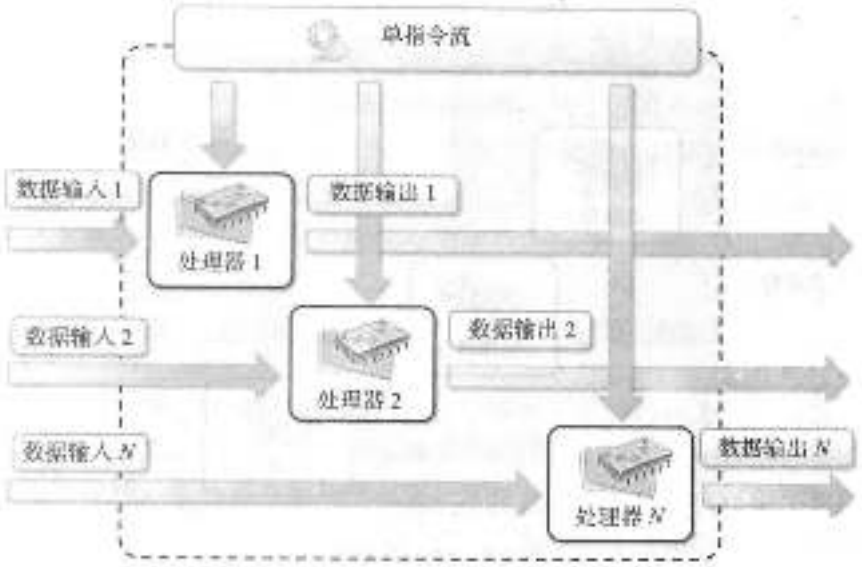


图 2-3　SIMD 构架

3. MISD 系统

MISD 计算系统是能在同一数据集上操作而在不同 PE 执行不同指令的多处理器机器（见图 2-4）。例如，语句

$$y = \sin(x) + \cos(x) + \tan(x)$$

在同一数据集上执行不同的操作。使用 MISD 模型构建的机器不适于大多数应用程序，已经设计的几台机器没有一个可以商业化，它们更像是智能测试而非实用的配置。

33

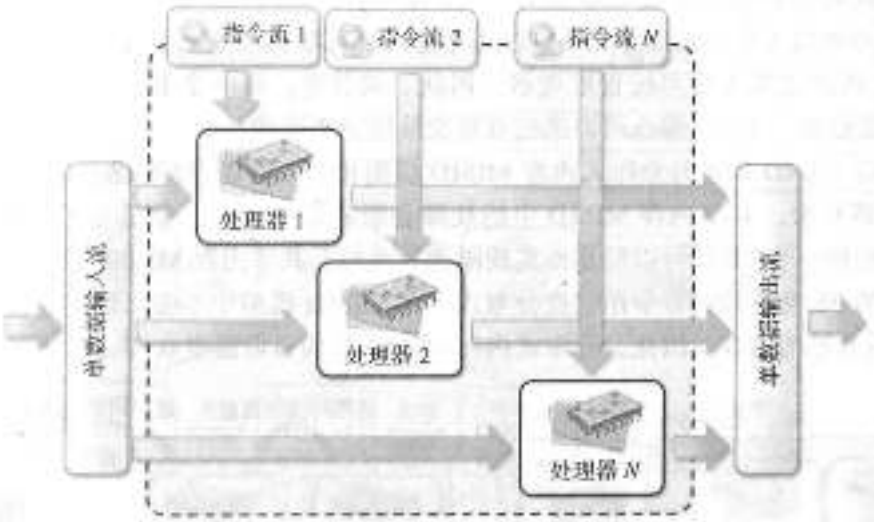


图 2-4　MISD 构架

4. MIMD 系统

MIMD 计算系统是能在多个数据集上执行多个指令的多处理器机器（见图 2-5）。MIMD 模型中的每一个 PE 都有单独的指令和数据流，因此使用该模型的机器适用于所有类型的应用程序。与 SIMD 和 MISD 模型不同，MIMD 机器中的 PE 是异步工作的。

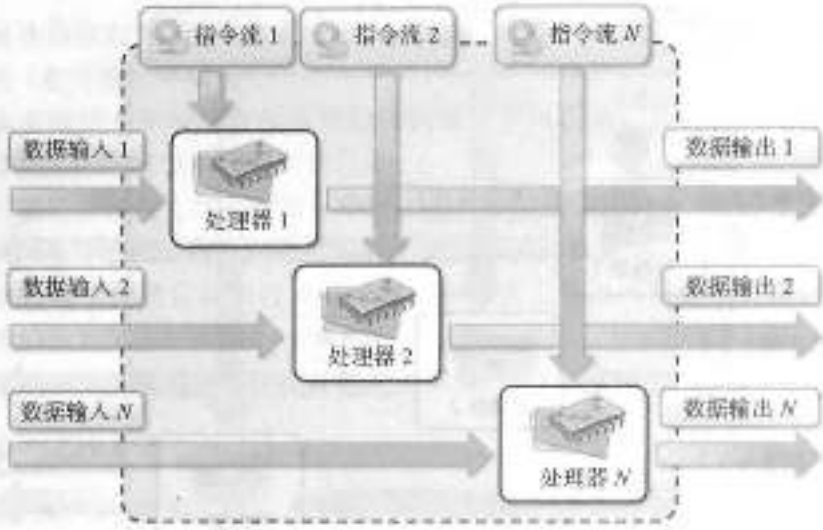


图 2-5　MIMD 构架

MIMD 机器按照 PE 与主存耦合方式不同大致可分为共享内存 MIMD 和分布式内存 MIMD。

（1）共享内存 MIMD 计算机

在共享内存 MIMD 模型中，所有执行单元都连接到一个可供访问的单一全局内存上（见图 2-6）。基于这个模型的系统也称为紧耦合多处理器系统。模型中 PE 之间通过共享内存进行通信，一个 PE 修改存储在全局内存中的数据，对所有其他 PE 都是可见的。共享内存 MIMD 模型的典型系统主要有 Silicon Graphics 计算机和 Sun/IBM SMP（对称多处理器）。

（2）分布式内存 MIMD 计算机

在分布式内存 MIMD 模型中，所有 PE 都有一个本地内存。基于这种模型的系统也称为松耦合多处理器系统。模型中的 PE 之间通过内部互联网络进行通信（进程间通信通道 / IPC）。PE 之间的网络连接可配置成树状、网状、块状等。每一个 PE 进行异步操作。如果任务之间需要通信 / 同步，那么可以通过互相交换信息来实现。

共享内存 MIMD 架构与分布式内存 MIMD 模型相比，前者更易编程，但对故障的容忍度更低且更难扩展。共享内存 MIMD 中的故障会影响整个系统，但是分布式模型则不然，分布式模型的每一个 PE 都可以轻易地实现隔离。此外，共享内存 MIMD 架构难以扩展，因为增加更多的 PE 会导致内存争用。在分布式内存 MIMD 模型中不会出现这种情况，因为每一个 PE 都有自己的内存。因此，分布式内存 MIMD 架构目前最受欢迎。

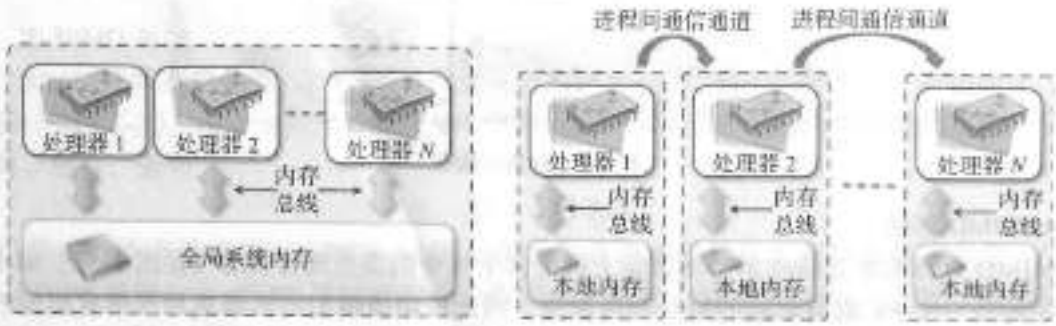


图 2-6　共享（左）和分布式（右）内存 MIMD 构架

2.3.3　并行编程方法

顺序程序运行在单个处理器上且有单一控制线程。为了让几个处理器共同处理同一个程序，就必须将该程序分成更小的、独立的块，这样每个处理器只需处理单独的一小块程序。以这种方式分解的程序就是并行程序。

并行程序设计有多种方法，最著名的有以下几种：

- 数据并行。
- 处理并行。
- “农场主和工作者”模式。

三种模型都适用于任务级并行。在数据并行的情况下，分治算法将数据分成多个集，然后在不同的处理单元中使用相同的指令处理每一个数据集。此方法非常适合在采用 SIMD 模型的计算机上进行处理。在处理并行的情况下，一个给定运算的多个（但不同的）操作可由多个处理器处理。在“农场主和工作者”模式中，工作分配方式如下：一个处理器配置为主站，所有其他 PE 被指定为从站；主站分配工作给从站，工作结束时，从站通知主站，主站进而收集处理结果。这些方法可用于不同级别的并行处理。

2.3.4　并行性的级别

并行等级由并行代码块（颗粒规模）决定。表 2-1 列出了并行性代码粒度的分类。这些方法的共同目标是通过降低延迟来提高处理器的效率。为了降低延迟，无论何时开始长时间的操作，都必须有准备好运行的另一个线程。这样做是为了能同时执行两个或两个以上的单线程应用程序，如编译、文本格式化、数据库查询和设备仿真。

如表 2-1 和图 2-7 所示，应用程序的并行性可以分为几个等级：

- 大颗粒（任务级）。
- 中颗粒（控制级）。
- 小颗粒（数据级）。
- 极小颗粒（发出多条指令）。

表 2-1　并行性级别

颗粒大小	代码	并行化执行者
大	独立的或重量级进程	程序设计员
中	函数或过程	程序设计员
小	循环或指令块	并行编译器
极小	指令	处理器

36

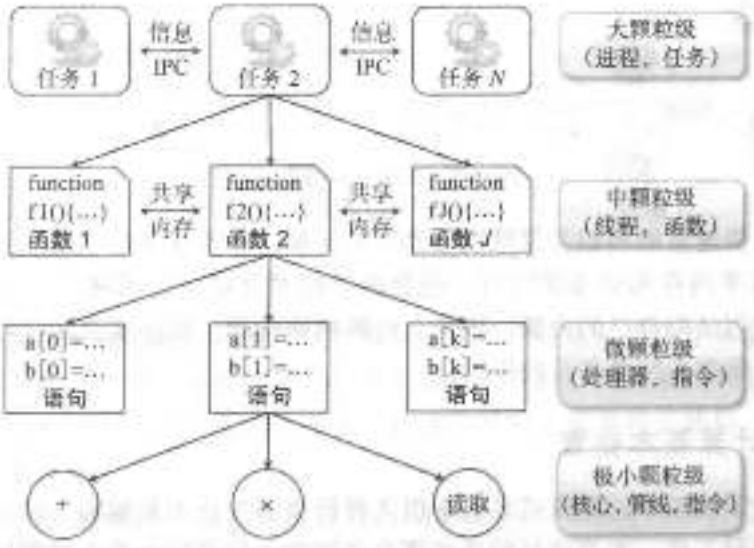


图 2-7　应用程序中的并行性级别

本书认为并行和分布式属于前两个级别，涉及多个线程或进程中的计算分布。

2.3.5 注意事项

现在，我们已经介绍了一些并行计算架构和模型方面的基本内容，下面我们分享几点系
37 统设计和实施方面的经验。这些注意事项可以作为指南，帮助我们了解并行性给应用程序或软件系统带来的好处。尤其需要注意，并行性通过同时执行多个活动来增加系统吞吐量和计算速度。但是，速度增长和控制因素之间并不是线性关系。例如，对于给定的 n 个处理器，用户希望速度也增加 n 倍，但这只是理想情况，由于通信开销而不太可能实现。

需要考虑以下两个重要原则：

- 计算速度和系统成本的平方根成正比，二者从来不呈线性增加。因此，系统的运算速度越快，提高系统速度的成本就越高（见图 2-8）。

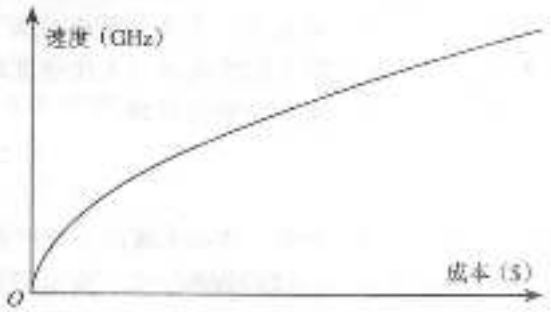


图 2-8　成本和速度

38 - 并行计算速度和处理器数量成对数关系（$y=k\log(N)$）。如图 2-9 所示。

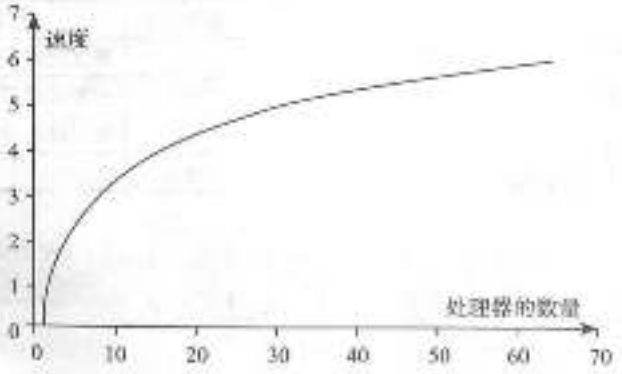


图 2-9　处理器数量和速度

并行处理的快速发展和相关领域概念界限的模糊，导致许多术语混乱不清。即使明确定义了区别，如共享内存和分布式内存，也会由于技术发展而逐渐融合。在并行处理这一领域，任何人都可以贡献自己的力量。因此，计算机架构师、操作系统设计师、语言设计师和计算机网络设计师都发挥着各自的作用。

2.4 分布式计算基本要素

上一节讨论了在单个计算机或系统中引入并行机制的技术和架构，以及如何在不同级别的计算堆栈中并行工作。本节将延伸这些概念并探究如何利用由多个异构计算机和系统组成

的系统来执行多个活动。我们将探讨什么是分布式计算，并从软件设计师的角度更准确地介绍最常见的实现分布式计算系统的原则和模式。

2.4.1　通用概念和定义

分布式计算主要研究用于构建和管理分布式系统的模型、架构和算法。Tanenbaum 等 [1] 给出了分布式系统的通用定义：

> 分布式系统是独立计算机的集合，对于用户来说是一个整体系统。

该定义大致包括了多种类型的分布式计算系统，这些系统统一使用和集成分布式资源。本章重点讲解将多个独立计算机转换为整体系统的架构模型。通信是分布式计算的基础之一。由于分布式系统由协同工作的多个计算机组成，所以有必要利用网络实现多计算机之间的数据和信息交换（Coulouris 等 [2]）：

> 分布式系统组件位于仅能通过传递消息来通信和协调活动的网络计算机中。

正如定义所述，分布式系统组件通过某种消息传递进行通信。该定义包括几种通信模型。

2.4.2　分布式系统组件

分布式系统是从硬件到软件的整个计算层次模型中多个组件交互的结果。许多组件协同工作，为用户呈现出一个单一的整体系统。图 2-10 描述了提供分布式系统服务的不同层的概况。

39

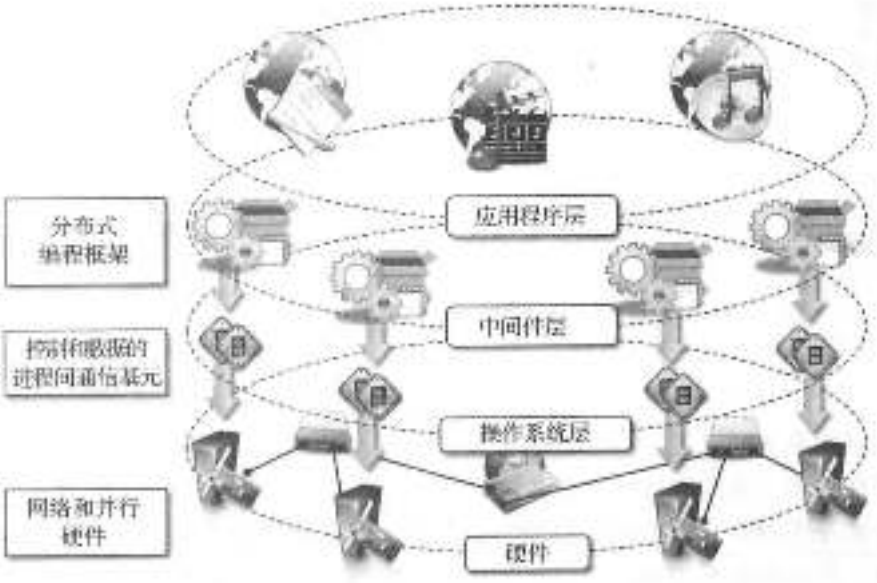


图 2-10　分布式系统的分层视图

在底层，计算机和网络硬件构成了物理基础设施，这些组件由操作系统直接管理。操作系统负责提供基础服务，用于进程间通信、进程调度和管理、文件系统和本地设备的资源管理。将网络和计算机这两层合并为一个平台，在这个平台上配置特定软件，便可将联网计算机组成一个分布式系统。

把公认的标准应用到操作系统层甚至硬件网络层中，利用异构组件可以很容易地构造

一个统一的集成系统。例如，不同设备之间的网络连接由协议进行控制，实现设备无缝交互。在操作系统层，进程间通信服务在标准化通信协议下执行，例如 TCP/IP 协议、UDP 协议等。

中间件层利用这些服务构建了一个开发和部署分布式应用程序的统一环境。这一层支持分布式系统的编程范式，我们将在本书第 5 ～ 7 章进行描述。依靠操作系统提供的服务，中间件层可开发协议、数据格式以及用于开发分布式应用程序的编程语言或框架。这些为分布式应用程序开发人员提供了统一接口，完全独立于底层操作系统且屏蔽了底层的异构性。

40 分布式系统层次架构的顶层是利用中间件设计和开发的应用或服务。设计应用层可实现多个目的，并且具有通过本地或 Web 浏览器可访问的图形用户接口（GUI）。例如，在云计算系统中，不论是为终端用户提供分布式应用接口，还是为构建分布式系统提供平台服务，都强烈推荐采用 Web 技术。IaaS 的供应商给出了很好的实例，如亚马逊 Web 服务（AWS）便于创建虚拟机、将虚拟机组织成集群以及在集群中部署应用和系统。图 2-11 说明了分布式系统的一般参考架构如何应用于云计算系统。

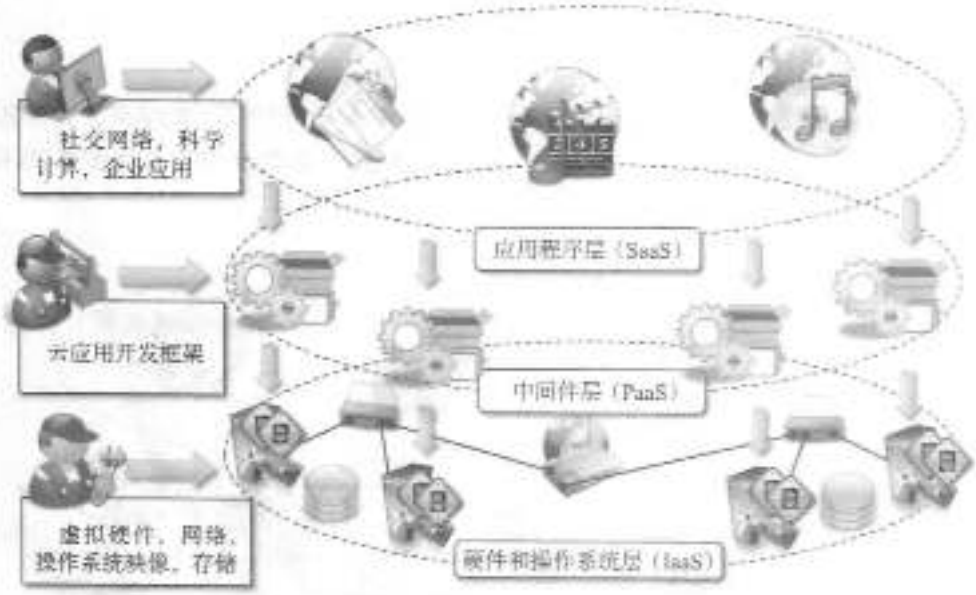


图 2-11　分布式云计算系统

硬件和操作系统层组成了一个或多个数据中心的最基本结构，其中服务器通过高速网络部署和连接在一起。这些硬件由操作系统管理，操作系统提供了基本的管理计算机和网络的能力。核心业务逻辑在管理虚拟化层的中间件上实现，虚拟化部署在物理机上，以实现最大化资源利用，并且提供可定制的应用运行环境。按照为客户提供的服务类型，中间件为应用开发人员提供了不同的工具。这些工具通过 Web 2.0 兼容接口提供了一系列服务，包括虚拟机创建、应用程序开发和运行环境部署。

2.4.3　分布式计算架构模式

尽管分布式系统包含若干层之间的交互，但只有中间件层才能够进行分布式计算，因为中间件层为应用提供了统一的整体运行环境。可采用多种不同的方式来组织组件，整合并构
41 成分布式计算环境。这些组件之间的交互以及组件的职责决定了中间件的结构和类型，换句

话说，也就是定义了中间件架构。了解架构模式[104]有助于从总体上理解和划分软件系统结构，尤其是分布式计算结构。

架构模式主要用于决定组件和连接器，组件、连接器及其结合条件一起作为架构模式的实例[105]。

设计模式[106]有助于软件工程师和开发者为如何在应用中构建组件关系以及理解软件应用程序的内部结构达成共识。架构模式在软件系统的整体架构中起到相同的作用。本节介绍分布式计算的相关架构模式，关注每个模式特有的组件和连接器。分布式系统的架构模式可以帮助我们理解系统中组件的不同作用，以及如何跨多机分配组件。架构模式分为两大类：

- 软件架构模式。
- 系统架构模式。

第一类与软件的逻辑结构有关，第二类包括从组件方面描述分布式软件系统物理结构的所有方式。

1. 组件和连接器

在详细讨论架构模式之前，需要给出一个恰当的术语。首先解释组件和连接器的含义，因为组件和连接器是定义架构模式的基本构建模块。组件代表一个封装了系统的功能或特性的软件单元。组件实例可以是程序、对象、进程、管道或过滤器。连接器是一种允许组件之间进行合作和协调的通信机制。与组件不同的是，连接器不封装在单一实体中，而是以分布式方式由多个系统组件实现。

2. 软件架构模式

软件架构模式是基于软件组件的逻辑结构。它提供了整个系统的直观视图，而忽略了系统的物理部署，因此软件架构模式十分有用。软件架构还能确定构成系统的组件和组件间交互模式的主要抽象模型。Garlan 和 Shaw [105]给出了软件架构的分类，如表 2-2 所示。

表 2-2　软件架构模式

种类	最常用的架构模式
数据中心	仓库 黑板
数据流	管道和过滤器 顺序批处理
虚拟机	基于规则的系统 解释器
调用和返回	主程序和子程序调用 / 自上而下的系统 面向对象系统 分层系统
独立组件	通信进程 事件系统

这些模型是从逻辑角度设计分布式系统的基础，下一节将进行描述。

（1）数据中心架构

这类架构将数据作为软件系统的基本元素，可访问的共享数据是以数据为中心架构的核心特征。因此，数据的完整性是系统的整体目标，特别是在分布式和并行计算系统中。

仓库架构模式是此类架构中相关程度最高的参考模型。其特点是有两个主要组件：可以表示系统当前状态的中央数据结构，以及操作中央数据的独立组件集合。独立组件与中央数据结构进行交互的方式有多种。根据对共享数据结构控制方法的选择，还可进一步将基于仓库的架构划分为子类。特别值得关注的是数据库和黑板。数据库系统的动态性由独立组件控制，在中央仓库发出动作触发特定进程的选择以实现数据操作。在黑板系统中，中央数据结构是选择待执行进程的主要触发器。

黑板架构由三个主要部件组成：

- 知识源。用于更新黑板中维护的知识库的实体。

- 黑板。表示在知识源中共享的、存储应用知识库的数据结构。
- 控制。触发器和过程的集合，负责管理与黑板的交互，更新知识库的状态。

在此参考模型中，知识源表示共享黑板的智能代理，对知识库的改变作出响应，这与一些专家在黑板前的头脑风暴非常相似。黑板模型已经广泛用于人工智能应用，以领域专家给出的声明和规则的形式来维护领域知识。这些操作通过一个控制外壳程序来控制系统的任务
[43] 处理活动。此模型现已成功应用于语音识别和信号处理领域。

（2）数据流架构

数据流架构中，控制计算的是数据的可用性。相对于以数据的访问为核心特征的数据中心架构，数据流架构显然包括数据流，因为此架构设计是由数据从组件到组件的有序移动决定的，数据的移动也是组件之间的通信方式。数据流架构可按照如下方式分类：系统控制方法、组件之间的并发度和描述数据流的拓扑结构。

顺序批处理模式。顺序批处理模式的特点是由独立程序组成的有序序列的按序执行。这些程序以文件的形式串联在一起，某一程序的输出作为下一个程序的输入，最后一个程序执行完的输出作为结果。这种设计在大型机时代的计算领域中广为流行，至今还在应用。例如，许多进行科学计算的分布式程序被定义为由程序序列表示的作业，如数据的预过滤、解析和后处理。通常用批处理方式组合执行这些程序序列。

管道过滤器模式。管道过滤器模式是之前所描述的架构模式的一种变形，它将软件系统的活动描述成数据变换的序列。加工链中的每个组件称为过滤器，用数据流表示过滤器之间的连接。就顺序批处理模式而言，数据按增量顺序被处理，一旦获得输入流中的数据，每个过滤器就会立即进行数据处理。只要一个过滤器产生了可使用的数据，下一个过滤器就会开始处理。过滤器通常没有状态表示，且没有前后顺序，它们按照内存中的数据结构连接，如先进/先出（FIFO）结构或其他数据结构等。这种特定的排序称为流水线，它利用了过滤器执行时的并发性。流水线微处理器是这种架构的经典例子之一，通过完成每个指令的不同阶段使得多重指令同时执行。我们可以识别过滤器的指令段，而数据流则由处理器内部共享的寄存器表示。另一个例子是 UNIX shell 管道（例如 cat<file-name>|grep<pattern>|wc-l），其过滤器由单一外壳程序组合而成，链接在一起的输入和输出数据流实现过滤器之间连接。该架构可用于编译器设计（例如 lex/yacc 模型基于扫描、解析、语义分析、代码生成各阶段形成管道）、图像和信号处理、音视频流处理。

当系统设计包含多阶段处理时，数据流架构是最理想的。数据流架构可以看作是将单独组件编排在一起的集合。在此方式下，组件是经明确定义的接口，包含输入和输出端口，连接器由端口之间的数据流表示。顺序批处理和管道过滤器模式之间的区别如表 2-3 所示。

表 2-3　顺序批处理和管道过滤器方式的比较

批处理	管道过滤器
粗粒度	细粒度
高延迟	由于增加了输入处理，延迟降低
外部访问输入	输入本地化
非并发	可能并发
非交互	不易交互但有可能

（3）虚拟机架构

虚拟机架构方式的特点是利用抽象的执行环境（通常称为虚拟机）模拟硬件或软件中不
[44] 可见的功能。应用程序和系统在这一层上实现，并且在各种实现了虚拟机接口的硬件和软件环境中具有可移植性。实现该模式系统的一般交互流程是：程序（或应用）定义操作和状态的抽象格式，该抽象可由虚拟机引擎处理。执行过程是对程序的解释和处理。在此情形下，

通常由引擎维护程序状态的内部表示，常见虚拟机架构的例子是基于规则的系统、解释器和命令－语言处理器。

基于规则模式。此架构的特点是将抽象执行环境看作一个推理机，以适当的规则或谓词的形式表达程序。应用程序的输入数据通常表示为一组声明或事实，推理机使用这些声明或事实激活规则或谓词，以此来处理数据。输出是规则激活产生的结果或者是输入数据的一组声明。规则或谓词的集合构成了用于推理系统属性的知识库。该方法比较特别，允许以行为方式而不是组件来描述系统或域。基于规则的系统在人工智能领域非常流行，在过程控制领域有许多实际应用。其中，通过PLC[⊖]收集和处理的传感器数据，在传感器数据异常时产生警告，基于规则的系统能够监控物理设备的状态。另一个基于规则系统的应用是在互联网领域：网络入侵检测系统（NIDS）常依赖一组规则来识别可能入侵计算系统的异常行为。

解释器模式。解释器模式的核心特征是存在一个引擎将伪代码解释成解释器可识别的格式。解释后的伪代码程序才能执行。根据此模式构造的系统主要包括四个组件：核心解释引擎、存储待解释伪代码的内部存储器、引擎的当前状态表示、正在执行程序的当前状态表示。此模型在为高级编程语言（Java、C#）和脚本语言（Awk、PERI等）设计虚拟机时非常有用。在此情形下，虚拟机将终端用户的抽象和执行此抽象的软/硬件环境连接起来。 45

虚拟机架构的特征是应用程序和执行环境之间的媒介层。这样设计的优点是应用独立于底层的硬件和软件环境；但同时也存在缺点，如性能的下降，还可能在提供虚拟执行环境后无法获取底层系统的特性。

（4）调用和返回架构

此架构确定了被组织成组件的所有系统，由方法调用将这些组件连接起来。此架构的系统活动特征在于一连串方法调用，这些方法调用的执行和组合标识了一个或多个操作的执行。组件的内部组织和连接方式可能不同，但按照系统架构和调用方法可分为三类：自上而下结构、面向对象结构和分层结构。

自上而下结构。此结构以命令式编程并发的系统为代表，采用分而治之的方法解决问题。根据这种架构开发的系统由一个主程序通过调用子程序或过程来完成任务。该架构的组件是过程和子程序，连接方式是方法调用。调用程序传递参数信息，并从返回值或参数中得到数据。方法调用还可以利用远程方法调用技术扩展单一进程的边界，如远程过程调用（RPC）及相关衍生方法。程序执行的整体结构可以用树形结构表征，树的根节点是主程序的主要功能。此架构从设计角度看非常直观，但是难以在大型系统中维护和管理。

面向对象结构。此架构利用面向对象编程（OOP）方法设计和实现系统，包含的系统类型非常广泛。系统用类来声明，用对象实现。类通过定义组件的类型，指定数据的状态及对数据的操作。与自上而下结构相比，面向对象结构的主要优点是数据及对数据的操作之间相互联系。对象实例隐藏了内部状态表示，并在操作其他组件时保证其完整性。这样可以更好地将进程分解，并且更易于管理系统。此方式的缺点主要有两个：每一个对象要调用另一个对象时，需要知道另一个对象的身份；要谨慎地设计共享对象，以保证其状态一致性。

分层结构。分层系统模式按层设计和实现软件系统，给出了系统抽象的不同级别。通常每一层最多与两层进行操作：其低一层的抽象层次和高一层的抽象层次。特定的协议和接

⊖ 可编程控制器（PLC）是用于自动化或机械电子化处理的数字计算机。不同于普通计算机，PLC能管理多个输入线并产生多个输出。尤其是PLC具有鲁棒性，适用于工厂等特殊环境。PLC是硬件实时系统，在输入后的给定时间内就能产生输出。

口定义了相邻层如何交互。可以将这样的系统建模为层次结构，每层对应每个抽象级别。因
46 此，组件是层，连接器是接口和用于邻近层间交互的协议。用户通常与最高抽象层进行交互，而高层则与低层进行交互并使用低层的服务完成活动。这个过程反复执行（如果必要的话）直到到达最底层。此过程也有可能反向进行：低层的事件和反馈可以触发高层的活动，并逐层向上传递信息。分层结构的优点在于，与面向对象结构一样，分层结构支持系统模块化设计，并可以根据不同级别的抽象将属于特定级别的所有操作封装在一起，以实现系统分解。层可以替换，只要它们符合所要求的协议和接口，这使得系统具有灵活性。分层结构的主要缺点是可扩展性差，在不改变层间协议和接口的情况下，不能够增加层[⊖]，这使得增加操作变得复杂。分层结构的例子包括现代操作系统内核、国际标准组织 / 开放系统互联（ISO/OSI）及 TCP/IP 栈。

（5）基于独立组件的架构模式

这类架构模式用具有生命周期的独立组件进行系统建模，且通过组件间的交互来执行活动。此类架构包括两类：通信进程和事件系统。二者对组件交互的管理方法不同。

通信进程。在此架构模式中，组件被表示为由 IPC 设备协调管理的独立进程。这种抽象方法特别适合分布式系统建模。该系统分布在计算节点构成的网络中，由若干个必要的并发进程组成，每一个进程都为其他进程提供服务，并可以利用其他进程提供的服务。这些进程的概念结构和通信方式依据其使用的特定模式而有所不同，可以是对等模式（peer-to-peer）或是客户端 / 服务器模式（client/server）[⊜]。这些进程使用连接器作为 IPC 设备进行通信。

事件系统。在这种架构模式中，系统组件松散地耦合、连接。除了处理数据和状态的操作，每一个组件还会发布（或声明）一组事件，其他组件也可以注册事件。通常，当事件被激活时，其他组件会给出执行的返回结果。在组件活动期间，特定的运行条件会激活其中一个组件发布事件，这样就会触发注册在事件上的调用返回并开始执行。事件的激活可能伴随着用于在调用返回中处理事件的上下文信息。这个信息可以作为调用返回的形参传递，或者使
47 用一些组件间的共享库来传递。基于事件的系统已经非常流行，API 层和编程语言层[⊝]都支持该系统的实现。此架构模式的主要优势是促进了开放系统的发展：可以增加新模块，并易于将新模块集成到系统中，只要这些模块具有事件注册的兼容接口。该架构解决了自上而下和面向对象结构的部分局限问题。首先，调用模式是隐式的，调用与被调用之间的联系不是硬编码，这就提供了很大的灵活性，可以在不改变应用程序源码的情况下添加或移除事件的处理程序。其次，事件源无需知道调用返回的事件处理程序。该架构的缺点是对系统计算缺乏控制。当一个组件触发一个事件时，并不知道有多少事件处理程序将被调用，以及是否有任何注册过的处理程序，只有在运行时才可能知道上述信息。从静态设计的观点来看，明确组件间的联系以及交互的正确性推理变得更为复杂。

本节介绍了最常见的软件架构模式，用于系统组件的逻辑结构建模。本节只描述了部分架构模式，其他模式参见文献 [105]。

⊖ 唯一的操作就是将层划分成子层，这样外部接口不变，而内部框架被重组为定义不同抽象级别的不同层。从相邻层观点看，新的重组层始终是一个单独的块。

⊜ 对等模式和客户端 / 服务器模式这两个术语将在下一节详细介绍。

⊝ 观察者模式 [106] 是软件设计的基本元素，其中编程语言（如 C#，VB.NET）和其他实现通用语言架构的语言 [53] 将事件语言结构转换为隐式调用模式。

3. 系统架构模式

系统架构模式包括组件的物理组织结构以及分布式基础设施上的进程。系统架构模式为系统的部署提供了一组参考模型，这不仅使工程师们有了描述系统物理布局的通用词汇，而且可以帮助他们快速识别给定部署的主要优缺点，以及系统部署是否适用于特定种类的应用程序。在本节中，我们介绍两个基本的参考模式：客户端 / 服务器模式和对等模式。

（1）客户端 / 服务器

在分布式计算中客户端 / 服务器模式十分流行，适用于多种应用程序。如图 2-12 所示，客户端 / 服务器模式的主要组成部分是服务器和客户端。这两个组件通过使用特定协议的网络进行连接和交互。通信是单向的：客户端向服务器发出请求，服务器在处理请求之后返回响应结果。可以是多个客户端组件向一个处于被动等待状态的服务器发送请求。因此，客户端 / 服务器模式的主要操作包括：请求和接受（客户端），监听和响应（服务器端）。

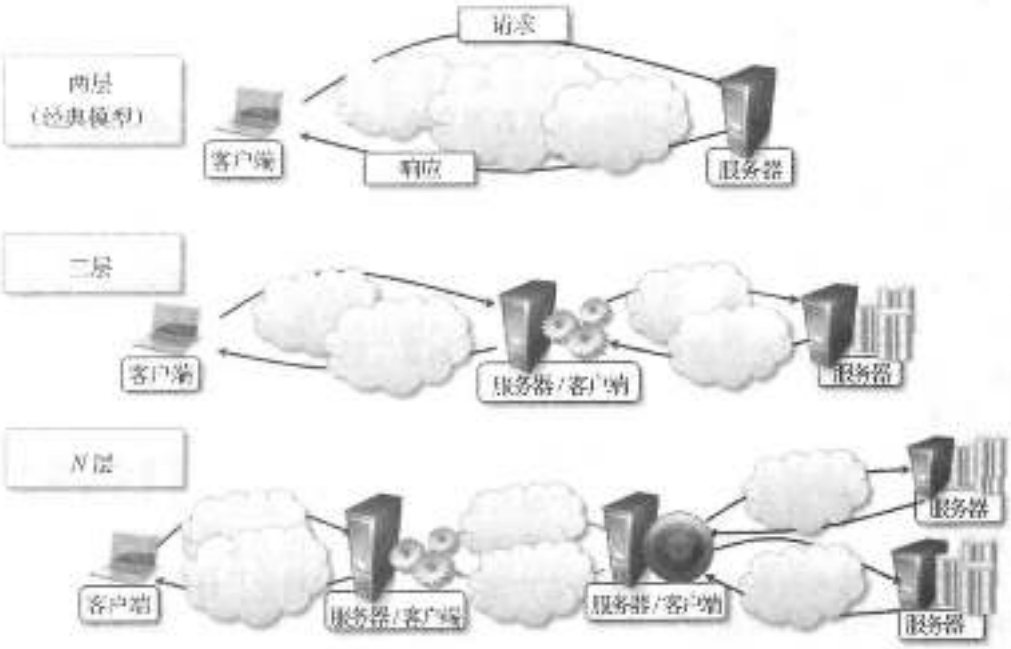


图 2-12　客户端 / 服务器模式

客户端 / 服务器模型适用于多对一的情形，其中，信息和感兴趣的服务可以集中后通过单一的接入点（服务器）被访问。通常会有多个客户需要该服务，所以必须对服务器进行适当地设计以便有效地处理来自不同客户的请求。应用中需要同时考虑客户端和服务器的需求设计。对于客户端设计，我们给出两个主要的模型：

- 瘦客户端模型。在此模型中，数据的处理和转换任务放在了服务器端，客户端轻量处理，即主要关注所需数据的检索和返回，不需更进一步的处理。
- 胖客户端模型。在此模型中，客户端组件负责在数据返回用户之前对其进行处理和转化，而服务器有着相对轻量的执行，主要管理数据访问。

客户端 / 服务器模型包括三部分：显示、应用逻辑和数据存储。在瘦客户端模型中，客户端只具有显示功能，而服务器负责应用逻辑和数据存储。在胖客户端模型中，客户端封装了显示功能和应用逻辑，而服务器主要负责数据存储和维护。

显示、应用逻辑和数据维护可以看作概念层（layer），称为层（tier）更加合适。按照概

念层及其在模块和组件的物理实现之间的映射关系。客户端/服务器模式可分成几种类型的多层架构。其中两种主要类型如下：

- 两层结构。此架构将系统分为两层：客户端和服务器端。客户端提供用户接口，主要处理显示层的所有功能；服务器负责应用逻辑和数据存储层功能。服务器组件通常部署在高性能主机上，用于处理用户请求、数据访问和执行应用逻辑，对客户端进行响应。该结构适用于规模不大且受可扩展性困扰的系统。尤其是随着用户数量的
49 增加，服务器的性能可能会显著下降。另一个限制是与数据相关的维护、管理和访问等问题，单个计算节点难以处理大规模数据，以提供满足应用需求的高性能服务。
- 三层结构/*N* 层结构。三层结构将数据显示、应用逻辑和数据存储分为三个层次。如果需要进一步区分应用逻辑和存储层，可将三层结构扩展为 *N* 层模型，此模型一般情况下比两层模型扩展性好，因为它可以把各层功能分布到多个计算节点，以便降低性能瓶颈。但是，理解和管理这些系统比较复杂。一个经典的三层结构实例是由关系型数据库管理系统存储数据的中等规模 Web 应用。在这种情况下：客户端是嵌入表示层的网络浏览器；应用服务器封装了业务逻辑层，实现具体业务；数据库服务器（双机备份以提供高可用性）完成数据存储、维护和管理。依赖第三方（或外部）服务来满足客户需求的应用服务器是 *N* 层结构的例子。

客户端/服务器架构已经成为设计和部署分布式系统的主流参考模型，基于该参考模型的应用实例很多，最常见的是源于其原始概念的 Web 应用。如今，客户端/服务器模型是复杂系统的重要架构，通过服务器以及网络交互的客户端进程来实现业务功能。此模型通常适用于多对一的情形，其中的交互是单向的，由客户端发起。然而，该模型存在可扩展性问题，因此不适用于大型系统。

（2）对等

对等模型（见图 2-13）是一种对称的架构，模型中所有组件都称为对等点，具有相同的功能，包含了客户端/服务器模型中的客户端与服务器的功能。更准确地说，某一对等点可作为服务器处理从其他对等点发来的请求，而这个对等点又可作为客户端向其他对等点发出请求。与区分客户端和服务器 IPC 职责的客户端/服务器模型相比，对等模型赋予每一个组件同样的职责。因此，对等模型适用于高分布性架构，并可以增加节点数目以提高扩展性。对等模型的缺点是算法的执行管理比客户端/服务器模型复杂。

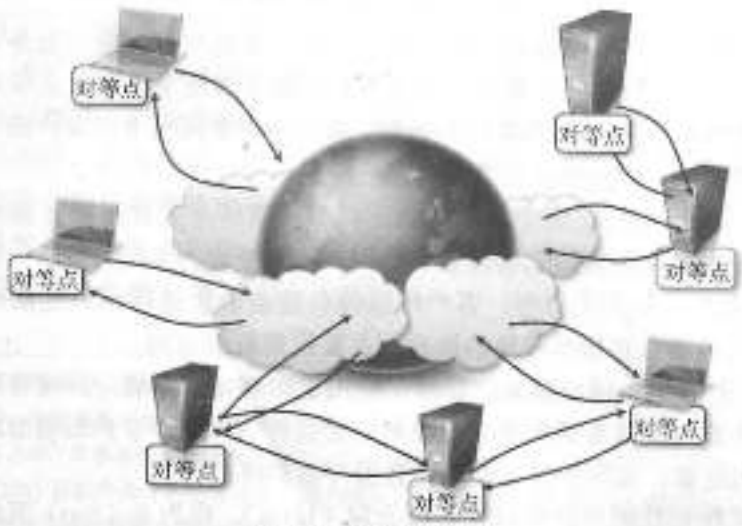


图 2-13　对等架构模型

与对等系统[87]最为相关的例子是文件共享应用，如 Gnutella，BitTorrent 和 Kazaa。除了节点间通信网络、文件共享信息及网络位置不同之外，这些系统都提供了一个客户端，该客户端同时也是服务器端，在为其他对等点提供文件时是服务器，而在从其他对等点下载文件时是客户端。为了处理大量的对等点，设计了不同的文件共享架构，并且与对等模型有些许差别。例如，在 Kazaa 中，不是所有的对等点都起到相同的作用，有的对等点用于收集一组对等点的辅助功能信息。关于对等模型的另一个值得关注的例子是 Skype 网络。 50

本节介绍的系统架构模式是一个参考模型，可根据设计和实施应用的特定需求进一步加强和多样化。比如，客户端/服务器架构起初只包含两种类型的组件，随着系统复杂性的增加，设计了多层架构将其进一步扩展和丰富。目前，该模型仍然是分布式系统和应用的主流参考架构。服务器和客户端抽象可用于大型和小型系统建模。单纯对等系统的实例非常罕见，常见的是对等系统的演化模型，如 Kazaa 在对等节点之间增加了层次结构。

2.4.4 进程间通信模型

分布式系统由一组通过网络连接的、交互的并发进程组成。因此，进程间通信（IPC）是分布式系统设计和实现的基础。IPC 既用于交换数据和信息，也用于协调进程活动。IPC 将分布式系统的不同组件连接起来，使其成为一个系统。进程之间交互的模型有多种，分别对应着 IPC 的不同抽象。其中最常见的包括共享存储、远程过程调用（RPC）和消息传递。在低层，IPC 由网络编程工具实现。套接字（socket）是最常用的在分布式进程间实现通信信道的 IPC 原语。套接字基于请求 - 应答通信模型，在较低层次上模拟客户端/服务器的抽象交互模式，使交互模式更加便捷。套接字提供了基本的传输字节序列的能力，这些字节序列在更高层上被转化为更有意义的数据（如进程参数、返回值或消息）。如此强大的抽象使得系统工程师只需要处理逻辑协同的分布式组件及其交换的信息，而不需要知道网络细节。这两个元素定义了 IPC 模型。本节介绍进程间通信架构的主要参考模型。 51

1. 基于消息的通信

消息在实现分布式计算的技术和模型演进过程中发挥了重要作用。Couloris 等[2]将分布式系统定义为“联网计算机上的组件仅通过传递消息进行沟通和协调组件活动的系统”。消息在这里指任何从一个实体传递到另一个实体的信息离散量。它包含任意形式的数据表示，数据有大小和时间限制，是对远程过程、序列化的对象实例或通用消息的调用返回的结果。因此，基于消息的通信模型可以用来指代本节讨论的任何 IPC 模型，并且不必依赖数据流的抽象。

一些分布式编程范式使用基于消息的通信，而不是呈现给开发者的用于实现分布式组件交互的编程模型。下面是一些主要编程模型：

- 消息传递。此范式用消息的概念作为模型的主要抽象。实体交换信息，以消息的形式对被交换的数据进行编码。不同的模型，其结构和消息的内容有所不同。该模型的例子包括消息传递接口（MPI）和 OpenMP。
- 远程过程调用（RPC）。此范式在单一进程以外扩展了过程调用的概念，由此触发代码在远程进程中的执行。在这种情况下，RPC 也称为客户端/服务器模式。一个远程进程承载一个服务器组件，允许客户进程请求方法调用，并返回执行结果。由执行 RPC 自动生成的消息来传递过程执行所需参数以及返回值。这样，消息也可以指被封送的参数和返回值。

- 分布式对象。这是 RPC 模型基于面向对象范式的实现，将方法的远程调用看作对象。每个进程都包括一组可远程访问的接口。客户进程可以请求指向这些接口的指针，并可以通过接口调用方法。底层的运行架构负责将本地方法调用转化为对远程进程的请求，并收集执行结果。通过消息实现调用者和远程进程的通信。相对于无状态信息的 RPC 模型，分布式对象模型引入了复杂的对象状态管理和生命周期管理。远
52 程执行的方法以实例的形式进行操作，该实例是为了这个方法的单独执行而创建的，具有有限的生命周期，并独立于请求。分布式对象架构的例子包括，通用对象请求代理架构（CORBA）、组件对象模型（COM、DCOM 和 COM+）、Java 远程方法调用（RMI）和远程 .NET。
- 分布式代理和活动对象。基于代理和活动对象的编程范式涉及对实例的定义，不管它们是否是对象代理，也不考虑是否有请求。这意味着对象有其自己的控制线程来执行动作。这些模型常显式地利用消息来触发方法的执行，并且这些消息附带着复杂的语义。
- Web 服务。Web 服务技术是 RPC 概念在 HTTP 上的实现，允许由不同技术开发的组件进行交互。Web 服务作为 Web 服务器上的一个远程对象，在 HTTP 请求时被转化为方法调用，使用特定的协议进行封装，如简单对象访问协议（SOAP）或表征状态转移协议（REST）。

消息的概念是 IPC 的基本抽象，它既被显式又被隐式地使用。在任何所讨论的例子中，消息的主要作用是定义分布式组件的交互协议，以便协调活动和交换数据。

2. 基于消息的通信模型

我们已经了解了基于消息的通信是多种分布式编程范式的基础。另一个分布式组件交互的特征是信息交换的方法，以及在多少个组件中进行交互。多数情况下，客户端 / 服务器模式可作为这种交互的相关参考模型。严格来讲，允许用多对一的交互模式来表示点到点通信模型。客户端 / 服务器模型的演变产生了不同的交互模式。本节简要介绍重要的并反复出现的一些交互模型。

（1）点对点消息模型

此模型在单一的组件中进行通信。每一个消息从一个组件传送到另一个组件，且有一个直接寻址来标识消息接收者。在点对点通信模型中，必须知道系统中的另一个组件的位置或如何获取其地址。模型中没有分发消息的中心设备，且通信是由消息发送者发起的。模型包括两个主要子类：直接通信和基于队列的通信。直接通信中，消息直接发送到接收设备上，一旦接收就立即处理。基于队列的通信中，接收设备维护一个消息队列，接收的消息存放在消息队列中，以便后续处理。点对点消息模型用于实现一对一或多对一通信的系统。

（2）发布和订阅消息模型

此模型包含基于组件间通知的不同策略。模型有两个主要角色：发布者和订阅者。发布者为订阅者提供工具，让订阅者注册某个感兴趣的主题或事件。适用于发布方的特定条件可
53 以触发特定事件上的消息创建。消息可被所有注册了相应事件的订阅者得到。用于给订阅者分派事件的两大策略如下：

- Push 策略。发布者负责通告所有的订阅者，例如方法调用。
- Pull 策略。发布者只需为特定事件提供消息，订阅者负责检查注册的事件上是否有消息。

发布者 – 订阅者模型非常适合实现一对多通信系统，且简化了间接通信模式的实现。实

际上，对于发布者而言，没有必要为了进行通信而获得订阅者的身份。

（3）请求 – 应答消息模型

请求 – 应答消息模型是对于每个由进程发送的消息都有应答的通信模型。该模型非常通用。它包括不同的类别，与通信组件的数量无关，只关注交互是怎样动态演变的。点对点消息模型更倾向于以请求 – 应答模式进行交互，尤其是在直接通信的情况下。发布 – 订阅模型不太可能基于请求 – 应答模式，而是依赖于通知模式。

本节描述了分布式系统的组件之间通信结构的参考模型。一种单一的模式通常难以满足一个系统的所有通信需求，更常见的情况是综合使用各种模式来设计和实现系统的不同通信功能。

2.5　分布式计算技术

在本节中，我们将具体介绍实现交互模型的相关技术，主要是基于消息的通信模式。它们是远程过程调用（RPC）、分布式对象框架和面向服务的计算。

2.5.1　远程过程调用

RPC 是用以执行客户请求过程的基本方法。RPC 允许在进程和单一内存地址空间之外完成过程调用。被调用的过程可以在同一个系统上，或是在网络内的不同系统上。1976 年起已经有 RPC 的概念，并在 20 世纪 80 年代早期由 Nelson[111] 和 Birrell[112] 进行规范化。从那时起，RPC 的主要组件一直没有改变。即使不是一项新技术，如今 RPC 仍然是复杂系统中 IPC 的基本组成部分。

图 2-14 描绘了 RPC 系统的主要组件。该系统基于客户端 / 服务器模型。服务器进程维护所有可被远程调用的过程的注册表，并监听从客户端发来的请求，说明调用哪一个过程，以及过程所需要的参数值。RPC 维护 IPC 的同步模式和函数调用。因此，正在执行的线程调用锁定状态，直到在服务器进程上的过程执行完成，并且将结果（如果有）返回给客户端。

54

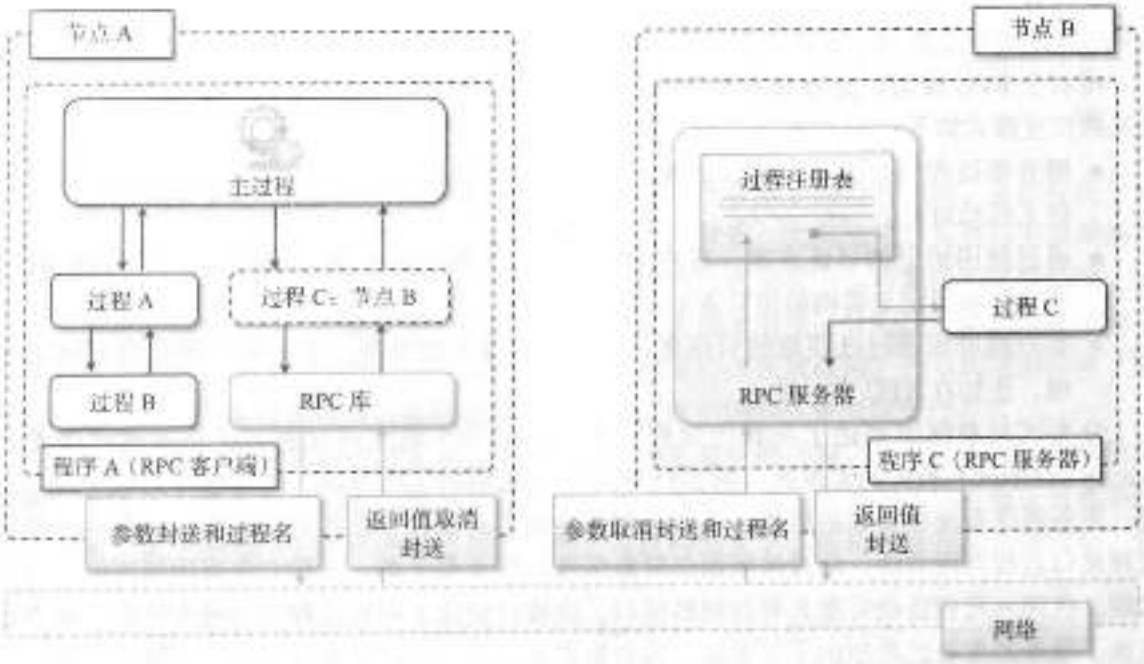


图 2-14　RPC 参考模型

封送处理（marshaling）是 RPC 的一个重要方面，封送处理将参数和返回值转换为一种更适合网络传输的字节流。术语解封（unmarshaling）指的是与封送相反的过程。封送和解封由 RPC 运行设备来执行，客户端和服务器用户代码并不需要执行这些任务。另一方面，RPC 运行时不仅负责参数的打包和解包，而且负责以完全透明的方式处理客户端和服务器进程之间的请求 - 应答交互。因此利用 RPC 开发系统 IPC 由以下几步组成：

- 设计和实现可被远程调用的服务器过程。
- 在节点上注册 RPC 服务器的远程过程，这些过程将在节点上可用。
- 设计和实现可调用远程过程的客户端程序。

每个 RPC 通常提供客户端和服务器应用程序接口（API），便于使用这种简单却实用的模型。考虑到参数和返回值的传递问题，服务器和客户端进程分布在两个独立的地址空间
55 中，使用由引用或指针传递的参数不适合这种情况，因为一旦解封这些参数，将导致内存位置在服务器进程中无法访问。其次，在用户定义的参数和返回值类型中，必须确保 RPC 运行时可以封送。这通常是可行的，尤其是当用户定义类型是由简单类型组成时，这自然就提供了封送处理。

在相当长的时间里，RPC 成为 IPC 中的主流技术，多种编程语言和环境提供库和附加包以支持这种交互模式。例如，RPyC 是 Python 的 RPC 实现。同时也存在与平台无关的解决方案，如 XML-RPC 和 JSON-RPC，分别在 XML 和 JSON 上提供了 RPC 功能。Thrift[113] 是 Facebook 开发的框架，用于透明的跨语言 RPC 模型。目前，术语 RPC 的实现涵盖了多种包含框架的解决方案，如分布式对象编程（CORBA、DCOM、Java RMI 和远程 .NET）以及从最初的 RPC 概念演变而来的 Web 服务，后续将讨论这些方法的特点。

2.5.2　分布式对象框架

分布式对象框架扩展了面向对象的编程系统，允许对象分布在异构的网络中，框架的便捷性使其如同在同一地址空间内一样。分布式对象框架利用 RPC 的基本机制，并对其进行扩展以实现对象方法的远程调用，以及通过网络连接跟踪可访问对象的引用。

相对于 RPC 模型，分布式对象模型中的管理实例采用接口而不是过程来展现。因此，常见的交互模式如下：

- 服务器进程维护活动对象注册表，其他进程可使用这些对象。可以通过接口或类来定义活动对象。
- 通过使用给定的寻址方案，客户端进程获取远程活动对象的引用。此引用由指向共享接口和类定义实例的指针表示。
- 客户端进程通过已获取的引用来调用活动对象上的方法。参数和返回值进行封送处理，正如在 RPC 中一样。

分布式对象框架描述了在调用远程方法时如何与本地实例交互。这是通过代理骨架（proxy skeleton）的机制实现的。图 2-15 概述了此机制的工作原理。代理和骨架总是成对出现：服务器进程维护骨架的组件，负责远程方法调用的执行；而客户端维护代理组件，通过代理接口远程调用方法。通过使用面向对象编程的继承和子类，远程方法调用的透明性得以实现。代理和远程活动对象具有相同的接口，该接口定义了可被远程调用的方法集。客户端
56 生成了服务器发布的类型的子类对象，该对象将本地方法调用转化成一个远程活动对象上的对应方法的 RPC 调用。在服务器端，无论何时接收到一个 RPC 请求，先打开请求包，然后

将方法调用分发给发出请求客户端的骨架。一旦服务器端的方法执行完成，返回值将被打包并发送回客户端，并且本地方法调用返回给代理。

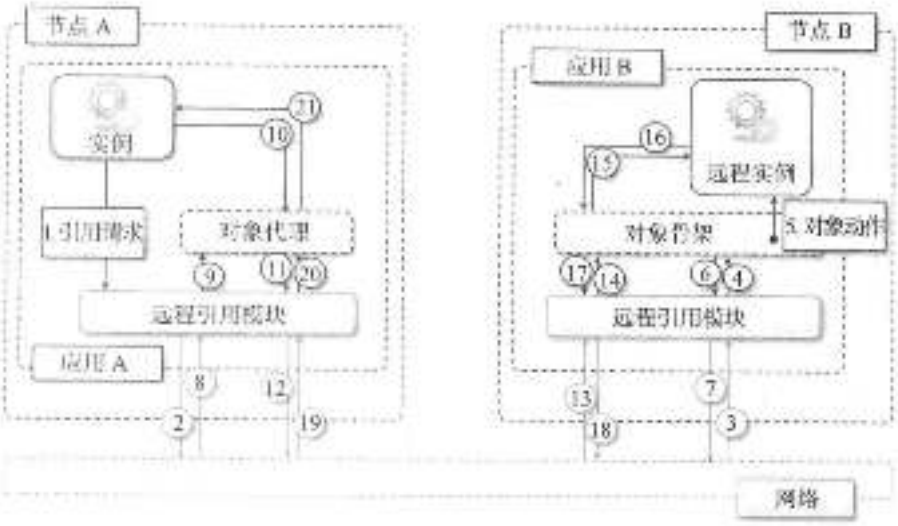


图 2-15　分布式对象编程模型

分布式对象框架将对象作为 IPC 的第一类实体。对象是调用远程方法的主要途径，同时也可作为参数和返回值传递。这样就需要注意一个问题。由于对象实例是复杂的实例，封装了一个状态并可能被其他组件引用，因此将对象作为参数或返回值传递时，会在另一执行环境中创建该对象实例的副本，产生两个状态无关的对象[⊖]。而实例复制是必要的，因为实例需要改变进程的边界。在设计分布式对象系统时，这是需要考虑的重要方面，因为变更进程边界可能会导致不一致。可将按值封送这个标准进程替代为按引用来封送[⊖]。另外一种情况是不复制对象实例，在服务器端（对参数）或在客户端（对返回值）创建对象实例的代理。按引用封送是更为复杂的技术，通常会增加运行时基础设施的负担，因为远程引用需要被跟踪。由于更加复杂且对资源的要求较高，只有在参数和返回值的复制会导致系统出现意外且不一致的行为时，才会使用按引用封送方法。

1. 对象激活和生命周期

在远程节点上过程的简单调用方面，分布式对象管理带来了额外的挑战。方法存在于对象实例的进程内，并且方法的执行可以改变对象的内部状态。尤其是对象实例的生命周期是分布式面向对象系统中的关键元素。在单一内存地址空间的情况下，对象由程序员创建，对象的引用通过对象从一个对象实例传递到另一个对象实例来实现。分配给对象的内存空间可以被程序员收回，或者当不再引用该实例时自动地被运行时系统收回。分布式情形带来了额外的问题，即对于通过远程接口提供的对象，需要采取不同的生命周期管理方式。

首先要考虑的因素是对象的激活，即如何创建远程对象的过程。有两种不同的对象激活方法：基于服务器的激活和基于客户端的激活。基于服务器激活方式的活动对象在服务器进程中创建，被注册为一个超出进程边界的实例。在这种情况下，活动对象有其生命周期，偶

57

⊖ 重新创建一个和对象一样的副本向外传递的方式称为按值封送（marshaling by value）。——译者注

⊖ 按引用封送（marshaling by reference）指当参数和返回值是其他数据的引用类型（如指针类型等）时，只传递这些数据是不够的。要使接收方能够处理，还需要将引用地址数据传递过去。——译者注

尔会执行方法，并将其作为远程方法调用的结果。在基于客户端的激活方式中，活动对象在服务器端起初是不存在的，当有来自客户端的方法调用请求时才创建活动对象。客户端的激活方式通常更适用于活动对象是无状态的，以及从远程客户端调用方法时才创建活动对象的场景。例如，如果远程对象仅仅是访问和修改服务器进程中的其他组件的一个入口，那么基于客户端的激活方式是更有效的模式。

其次要考虑的因素是远程对象的生命周期。基于服务器激活方式中，对象的生命周期通常由用户决定，因为远程对象创建过程明确地由用户控制。在基于客户端激活方式中，远程对象的创建是隐含的，因此其生命周期由运行时基础设施的某些策略来控制。可以考虑不同的策略：最简单的是为每个方法调用都创建一个新的实例。这种方法对对象实例要求较高，通常将结合一些契约管理策略，这些策略允许对象在特定的时间间隔（契约）内重复用于后续方法调用。另一项策略是，可以考虑一次只有一个实例，则对象的生命周期由方法调用的次数和频率控制。对此，不同框架可提供不同的控制等级。

目前在一定程度上，对象激活和生命周期管理是支持几乎所有分布式对象编程框架的两大特征，因为它们是理解分布式系统行为的关键。尤其是这两个方面是设计可被其他进程访问和维护状态的组件的基础。理解创建了多少个代表同一个组件的对象以及对象的生命周期，是跟踪由实例内部数据的错误更新造成的不一致的关键。

2. 分布式对象框架实例

支持分布式对象编程的语言随着时间在演变。现在分布式对象编程是主流编程语言的一个共同特征，如 C# 和 Java 将此功能作为基类库的一部分。这种集成标志着分布式对象编程技术的成熟。该技术最初被设计为可以在几种编程语言中使用的一个独立的组件。本节简要
58 地回顾了与分布式对象编程最为相关的方法和技术。

（1）公共对象请求代理架构（CORBA）

CORBA 是由对象管理组织（OMG）提出的规范，用于在分布式组件中提供跨平台和跨语言的互操作能力。最初设计此规范的目的是提供有效的工业级的互操作标准。CORBA 规范的当前版本为 3.0 版，目前该技术还不是很流行，主要是因为开发过程相当复杂，且不同语言开发的组件之间的互操作尚未达到所提出的透明性级别。CORBA 架构中的一个基本组件是对象请求代理（ORB），其作用相当于中央对象总线。CORBA 对象注册到公布的 ORB 接口，客户端可以获取该接口的引用，并调用该接口上的方法。ORB 负责将引用返回给客户端，管理所有低层操作以执行远程方法调用。为了简化跨平台的互操作，接口由接口定义语言（IDL）定义，此语言提供了平台无关的组件规范。特定 CORBA 编译器将 IDL 规范转换成存根 - 骨架（stub-skeleton），编译器在特定的编程语言中生成所需的客户端（stub）和服务器（skeleton）组件。某些编程语言实施了这些模板，这使得 CORBA 组件仅需要简单地使用与开发语言匹配的客户端和服务器，就可以在不同的运行时环境中使用。规范意味着要符合工业标准，CORBA 提供了在其运行时的不同实现方法之间的互操作性。特别地，在最底层 ORB 实现相互通信使用的是因特网 ORB 间协议（IIOP），这个协议标准化了不同 ORB 实现之间的交互。此外，CORBA 提供了额外的抽象和独立 ORB，能利用便携式对象适配器（POA）在网络节点间移动，POA 是部署和管理服务器的运行时环境。这两层的接口定义清晰，这样就提供了更大的灵活性，并允许不同实现方法一起无缝地运作。

（2）分布式组件对象模型（DCOM/COM+）

DCOM 后期被整合并演变为 COM+，是引入 .NET 技术之前，微软为分布式对象编程

提供的解决方案。DCOM 的特征是允许在进程边界之外使用 COM 组件。一个 COM 对象标识一个组件，该组件封装了一组连贯且相关的操作，利用 COM 对象的接口特征可轻易地嵌入另一个应用程序中。为了支持互操作性，COM 采用二进制格式，可在不同的编程语言中使用 COM 对象。通过添加所需的 IPC 支持，DCOM 在分布式环境中实现了这些功能。DCOM 的构架与 CORBA 十分相似，但更加简化，因为 DCOM 的目的并不是为了促进同一级别的互操作。DCOM 架构的实现由微软垄断，提供单一的运行时环境。一个 DCOM 服务器对象可以公布多个接口，每一个接口都代表着对象的一种不同行为。为了调用由接口公布的方法，客户端包含一个指向该接口的指针，在使用这个指针时，将其看作一个指向客户端地址空间内的某个对象的指针。DCOM 运行时负责执行该指针所需的所有操作。该技术在基于微软的环境中提供了适当的互操作性，并且存在第三方实现，使得 DCOM 即使在基于 UNIX 的环境中也可以使用。目前，即使 DCOM 仍然在行业中使用，但该技术已经不再流 59
行，并且已经被其他方法所取代，比如远程 .NET 和 Web 服务。

（3）Java 远程方法调用（RMI）

Java RMI 是由 Java 提供的标准技术，用于在分布式 Java 对象中实现 RPC。RMI 定义了对象上方法调用的基础设施，其中对象位于不同的 Java 虚拟机（JVM）上，这些虚拟机部署在本地节点或远程节点上。与 CORBA 相同，RMI 基于存根 - 骨架概念。开发者定义了一个接口，扩展了定义 IPC 契约的 java.rmi.Remote。Java 允许只发布接口，同时它依赖于服务器的实际类型和客户端的部分实现。一个实现了先前接口的类，代表了可在 Java 虚拟机边界之外被访问的骨架组件。存根由使用 rmic 指令行工具定义的骨架类生成。一旦存根 - 骨架准备好，骨架实例就会被注册到 RMI 注册接口 URI 上，通过 URI 可以获取实例，映射到对应的对象上。RMI 注册表是一个独立的组件，用于跟踪所有可以在节点上获取的实例。客户端联系 RMI 注册表，并以 rmi://host:port/serviceName 的形式指定一个 URI，来获取一个相应对象的引用。RMI 运行时将自动检索与给定 URI 的骨架配对的存根组件的类信息，然后返回正确配置为与远程对象进行交互的实例。在客户端代码中，骨架提供的所有服务都可以通过调用在远程接口中定义的方法来获取。RMI 提供了十分透明的交互模式。一旦完成开发过程，并获得了一个远程对象的引用，客户端代码就可以像本地实例一样与远程对象交互，而且 RMI 将执行所有 IPC 需要的操作。此外，RMI 也允许定制用于远程对象的安全性。这通过利用标准的 Java 安全架构实现，允许制定策略，定义承载远程对象的 JVM 所拥有的权限。

（4）.NET 远程处理

远程处理是一项在 .NET 应用程序中实现 IPC 的技术。为开发者提供了一个访问远程对象的统一平台，这些对象来自支持 .NET 的任意语言所开发的任意应用程序。相对于其他分布式对象技术，远程处理是一个可完全定制的架构，允许开发者控制用来在代理和远程对象之间交换信息的传输协议，定义对数据进行编码的序列化格式，管理远程对象的生命周期和服务器。除了远程处理架构的模块化和可定制特性，远程处理允许不同应用域的对象透明地交互。一个应用域代表了一个孤立的只能通过远程处理信道访问的执行环境。一个单一的进程可以承载多个应用程序域，且必须至少有一个程序域。

远程处理允许位于不同应用域的对象以完全透明的方式进行交互，不论两个应用域是否在同一进程中，同一机器中或同一节点上。这种参考构架基于经典的客户端 / 服务器模型，即承载远程对象的应用域是服务器，访问远程对象的应用域是客户端。开发者定义一个 60

类，这个类继承了 MarshalByRefObject，基类提供内置工具以获得来自另一个应用域实例的引用。不继承 MarshalByRefObject 类型的实例将跨应用域的边界被复制。没有必要手动生成某种需要远程公布的存根。远程处理架构将自动地提供在客户端应用域生成代理所需的全部信息。为了通过远程处理访问组件，要求组件在远程处理运行时注册，并且以 scheme://host:port/ServiceName 的形式将其映射到一个特定的 URI 上，其中的骨架通常是 TCP 或 HTTP。可以使用不同的策略来发布远程组件：开发者可以提供开发类型的实例，或仅是简单的类型信息。当仅提供类型信息时，对象基于客户端方式自动激活，开发者可以通过重写 MarshalByRefObject 的缺省行为来控制对象的生命周期。为了与远程对象进行交互，客户端应用域需要通过识别远程对象的 URI 来查询远程设备，并获得远程对象的代理。这样，与远程对象的交互完全透明化。如在 Java RMI 中一样，远程处理允许自定义安全措施，用于由远程调用触发的代码的执行。

这些是最为流行的实现分布式对象编程的技术。CORBA 是一项工业标准技术，用来开发跨越不同平台的分布式系统，此技术可在多种实现和语言中互操作。Java RMI 和远程 .NET 是 IPC 的基础设施，用于创建基于某一种技术的分布式应用程序，如 Java 和 .NET。对于 CORBA，其使用和配置并不复杂，但是技术本身不可互操作。而 Java 和远程 .NET 依赖于一个统一的平台非常简单且直观，并且提供了适合支持语言结构的透明交互模式。尽管这两个构架是相似的，但它们也有一些细微差别：Java 使用名为 RMI 注册表的外部组件来确定远程对象位置，并仅允许发布接口；而远程 .NET 不使用注册表，并允许开发者公开类的类型。这两种方法都广泛地用于分布式应用程序开发。

2.5.3　面向服务的计算

面向服务的计算以服务的形式组织分布式系统，此服务表征了构建系统的主要抽象。面向服务将应用和软件系统表示为在面向服务架构（SOA）中协同工作的服务的聚集。尽管还没有专门用于开发面向服务软件系统的技术，但 Web 服务却是实际上用来开发基于 SOA 系统的方法。Web 服务作为实现云计算系统的重要组件，通过互联网建立用户与系统之间的主要交互通道。

1. 什么是服务

服务封装了一组具有紧密相关功能的软件组件，其功能可被重用并被集成到更大、更复杂的应用中。术语服务是一个普遍的抽象概念，包括使用不同技术和协议的多种不同的实现。Don Box[107] 给出了标识一个服务的四大特点：

61

- 边界明确。面向服务的应用程序通常由分布在不同领域、不同受信机构和执行环境的服务组成。一般情况下，跨越上述边界代价很高，因此，通过设计和利用消息传递可以完成服务调用。分布式对象编程的远程方法调用是透明的，在面向服务的计算环境中，服务的交互是显式的，服务的接口保持最小化，以促进其重用并简化交互。
- 服务自治。服务是已有的提供某种功能的组件，服务被聚合并协同以构建更为复杂系统。这些组件不是为某个特定系统而设计的，甚至可以同时被集成到多个软件系统中。相对于面向对象架构假定应用程序的部署是原子的，面向服务将这种情况视为例外而不是规则，关注的是作为自治组件的服务的设计。自治的概念也影响了服务处理故障的方式。服务运行于未知的环境中，并与第三方应用程序交互。考虑到

这种环境，可能会出现如下问题：应用程序可能在无通知的情况下发生故障，消息可能存在格式错误，客户端可能未被授权。面向服务的设计解决上述问题的方法包括：使用事务、持久队列、冗余部署、故障转移、管理不同的域之间的信任关系。

- 服务共享模式和契约，不是类集或接口定义。服务不像在面向对象系统中那样被表示为类或接口，而是由模式和契约来定义。一个服务通告一个契约，描述可被发送或接收以及附加约束的消息的结构。服务没有以类型和类的形式表示，因此更容易用于更广泛和异构的环境中。同时，由于更改可能传播给所有可能的客户，所以面向服务要求契约和模式保持稳定。为解决这个问题，允许契约和模式在不破坏已配置代码的前提下演化服务。比如，XML 和 SOAP 技术提供了合适的工具来支持这些特性，而不是类的定义或接口的声明。
- 服务的兼容性由策略决定。面向服务将结构兼容性从语义兼容性中分离出来。结构兼容性基于契约和模式，可以基于机器学习技术进行验证或强制执行。语义兼容性以定义了服务功能和服务需求的策略形式表示。策略以表达式的形式组织起来，为了使服务可以正常运作，这些表达式必须为真。

如今，服务是最流行的用于复杂和互操作系统设计的方法。分布式系统意味着异构性、可扩展性和动态性。通过从特定实现技术和平台上抽象出来，服务提供了更为有效的方法来实现集成。此外，由于服务被设计成自治的组件，所以更易于重用和集成。这些特性不是从智能系统的设计和实现中提取出来的，而只是服务特性的一部分，就像分布式对象编程一样。 62

2. 面向服务的架构（SOA）

SOA 是支持面向服务的架构[⊖]。它将一个软件系统组织成可互操作服务的集合。SOA 给出一系列设计原则，涵盖系统开发结构和将组件集成为连贯的非集中化系统的方法。基于 SOA 的计算将功能封装成一组互操作的服务，这些服务可以集成到不同业务领域的不同软件系统中。

SOA 有两个主要角色：服务供应商和服务消费者。服务供应商是服务的维护方，是提供一个或多个服务的组织机构。为了通告服务，服务供应商可以将服务和服务契约发布在注册表中，服务契约指定服务类型、服务使用方法、服务要求和费用。服务消费者可以在注册表中查询服务的元数据，并发所需的客户端组件来绑定并使用该服务。服务供应商和服务消费者可以属于不同的组织机构或业务领域。在基于 SOA 的计算系统中，组件通常同时扮演着服务供应商和服务消费者的角色。服务可以收集从其他服务中检索的信息和数据，或者创建满足服务消费者特定需求的服务工作流。该过程称为服务协同，通常用来描述自动化安排、协调，以及对复杂计算机系统、中间件和服务的管理。另一个重要的互操作模式是服务编排，这是无单点控制的服务协调和互操作。

SOA 提供了用于多种软件系统架构的参考模型，尤其是企业业务应用和系统。在此环境中，互操作性、标准化和服务契约扮演了重要角色。特别是以下指导原则 [108]，它们表征了 SOA 平台的特性，在企业环境中突显了其优势：

⊖ SOA 定义由开放工作组 Open Group（www.opengroup.org）给出，该工作组是供应商和技术中立联盟，包括 300 家成员机构。其工作内容包括管理、创新、研究、标准化、认证和开发测试。Open Group 是 UNIX 商标认证机构，因为它是 UNIX 系统官方定义的创建者。SOA 相关文档和标准参见网址 www.opengroup.org/soa/soa/def.htm。

- 标准化服务契约。服务遵守一个给定的通信协议，该协议由一个或多个服务描述文件详细说明。
- 松散耦合。服务被设计成自包含组件，维护最小化依赖于其他服务的关系，仅要求可相互识别。服务契约约束服务之间所需的互操作。这简化了服务的柔性聚合，实现了更灵活的支持企业业务发展的设计策略。
- 抽象。一个服务是由服务契约和描述文件完整定义的。服务隐藏了自己的逻辑，将其封装在服务实现中。服务描述文件和契约的使用使得我们不需要考虑技术实现细节，并提供了一个定义企业环境中的软件系统的更直观的框架。
- 可重用性。服务被设计为组件的形式，可以更为有效地实现重用，这样就减少了开发时间和相关的成本。可重用性能使得设计、实施和部署成本有效系统变得更加灵活。因此，可以通过支付合理的费用，利用第三方服务来交付所需的功能，而不需要自行开发此功能。
- 自治性。服务控制封装逻辑，服务消费者没有必要了解执行情况。
- 状态缺失。通过提供无状态的互操作模式，提高了服务被重用和集成的机会，尤其是在单一服务被多个属于不同机构和业务领域的消费者使用的情况下。
- 可发现性。服务由描述文件定义，其中包括有效发现服务的元数据。服务发现为利用第三方资源提供了有效的方法。
- 可组合性。使用服务作为基础模块可以实现复杂的操作。服务协同和编排为组合服务并实现业务目标提供了坚实的支撑。

63

与上述原则配套的，还有将 SOA 用于企业应用集成（EAI）的其他参考资料。SOA 说明文档[⊖]集成了之前所描述的原则，并全面考虑了将面向服务方法用于企业应用软件设计的总体目标和价值。此外，建模框架和方法，如面向服务的模型框架（SOMF）[110]，以及由先进架构标准化组织（OASIS）引入的参考架构，提供了用于有效实现 SOA 的方法。

SOA 可以通过几种技术来实现。SOA 的最初实现利用了分布式对象编程技术，如 CORBA 和 DCOM。尤其是 CORBA 已经成为了实现 SOA 系统的合适平台，因为它在不同的实现中促进了互操作性，目前已经成为支持工业应用开发的一种规范。如今，SOA 大多通过 Web 服务技术实现，Web 服务技术为连接系统和应用提供了互操作平台。

3. Web 服务

Web 服务[21]是实现 SOA 系统和应用的主要技术。Web 服务利用了网络技术和标准构建分布式系统。Web 服务成为实现 SOA 的首选技术的原因有以下几个方面。首先，Web 服务允许跨越不同平台和编程语言进行互操作；第二，Web 服务基于众所周知的且与供应商无关的标准，比如 HTTP、SOAP[23]、XML 和 WSDL[22]；第三，Web 服务提供了一种直观和简单的方法来连接异构的软件系统，可以在一个分布式环境中快速组合服务；最后，Web 服务提供了在工业环境中使用的企业业务应用所需的功能。Web 服务定义了服务发现功能，这使得系统架构师能够更有效地组合 SOA 应用，并且具有评价服务质量功能，可以评估一个特定的服务是否遵守了服务供应商和服务消费者之间的契约。

64

⊖ SOA 说明文档由 17 位 SOA 实践者编写的，定义了基于面向服务的方法来设计软件系统架构的指南和原则。获取该文档网址 www.soa-manifesto.org

Web 服务的原理非常简单，以面向对象的抽象模型为基础，Web 服务公布了一系列操作，基于网络协议可以调用这些操作。方法操作支持以复杂或简单类型形式表示的参数和返回值。调用 Web 服务方法的语义通过互操作标准化语言（如 XML 和 WSDL）来描述，这些标准化语言也提供了一个完整的框架，用于描述平台无关环境中的简单或复杂的类型。将 Web 服务托管在 Web 服务器上便可以访问这些 Web 服务，因此，HTTP 是与 Web 服务进行交互的最流行的传输协议。图 2-16 描述了常见的 Web 服务应用案例。

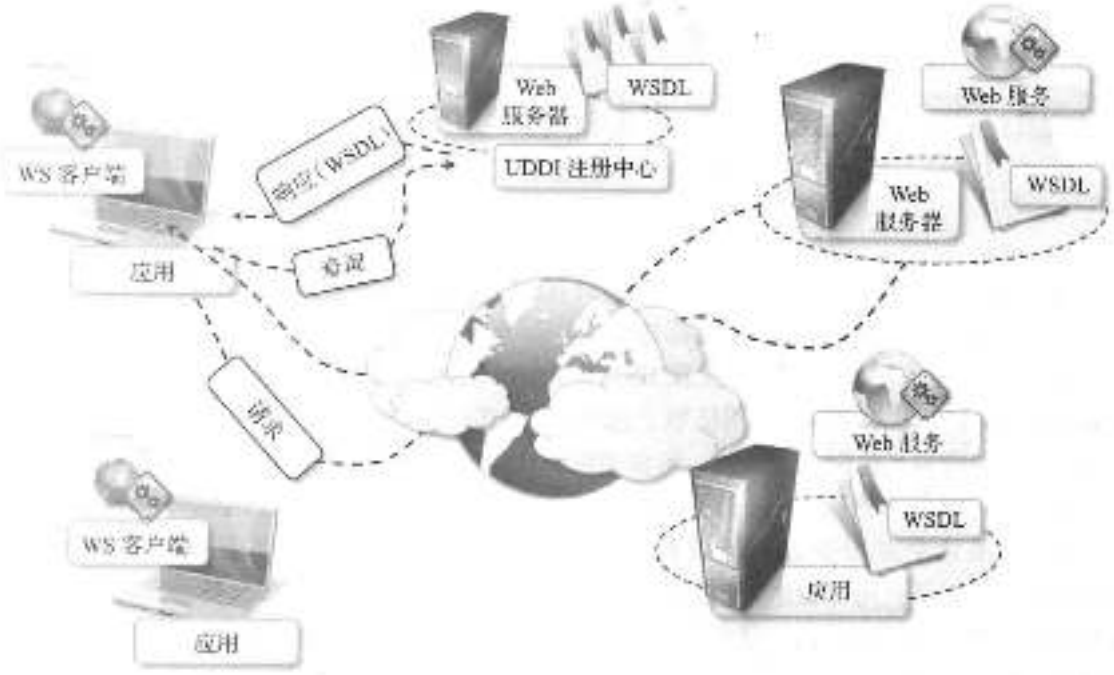


图 2-16　Web 服务交互场景

系统架构用其首选技术开发 Web 服务，并且在兼容的 Web 或应用服务器上部署 Web 服务。通过 Web 服务描述语言（WSDL）定义的服务描述文档，既可以上传至全局注册表，又可以作为服务的元数据。服务消费者可以在全局目录中使用通用描述发现和集成（UDDI）、查找和发现服务，或者通过首先询问 Web 服务的方式直接检索服务元数据。Web 服务描述文档允许服务使用者自动生成指定服务客户端，并将其嵌入现有的应用程序。Web 服务现在非常流行，主流编程语言以库或开发支持工具的形式绑定 Web 服务功能。相对而言，CORBA 需要更多集成技术，而 Web 服务的技术更加无缝化和简单化。而且，由于可互操作性，相对于其他几种分布式对象框架，如只适用于单一平台或环境的远程 .NET、Java RMI 和 DCOM/COM +，Web 服务是实现 SOA 的更好的解决方案。

除了通过利用基于 Web 的技术和互操作标准实现远程方法调用的主要功能外，Web 服务还综合了多种技术，更容易集成异构应用，实现面向服务的计算。图 2-17 所示为 Web 服务技术栈，列出了所有描述和实现 Web 服务抽象的概念框架组件。这些技术涵盖了在分布式环境中执行 Web 服务的所有方面，从网络连接的具体要求到服务的发现。这些技术的核心都是 XML，XML 也是使得 Web 服务流行并易于使用的原因之一。基于 XML 的语言可用于管理 Web 服务方法调用（SOAP）的底层交互、提供服务的元数据（WSDL）、发现服务（UDDI），以及其他核心操作。在实际中，使 Web 服务运作的核心组件是 SOAP 和 WSDL。

65

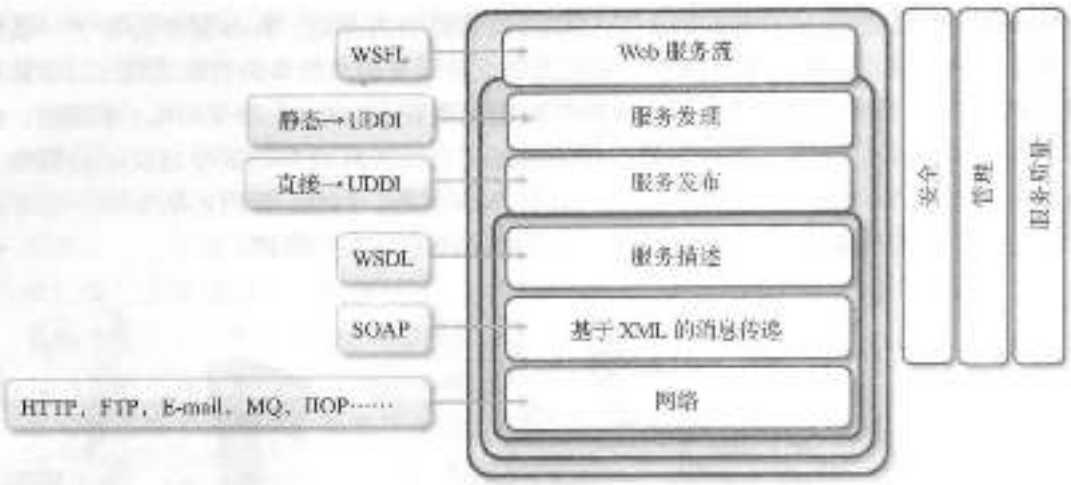


图 2-17 Web 服务技术栈

简单对象访问协议（SOAP）[23] 是一种基于 XML 的语言，以平台无关的方式交换结构化信息，是用于 Web 服务方法调用的协议。在分布式网络环境下，SOAP 被认为是一种应用层协议，利用传输层协议 HTTP 实现 IPC。SOAP 以结构化消息的形式交互信息，而消息本身是 XML 文档，这种 XML 文档仿照了包含信封、信头和信体内容的信件结构。信封定义了 SOAP 消息的边界。信头是可选的，包含如何处理消息的相关信息。此外，还包含如路由和传送设置、身份验证和授权声明，以及处理环境等信息。信件主体包含了实际待处理的消息。

66 SOAP 消息主要用于方法调用和结果返回。图 2-18 给出了一个调用 Web 服务方法的 SOAP 消息的例子，用于检索给定股票的价格并返回结果。尽管 XML 文档易于在任何平台或编程语言中生成和处理，人们还是经常认为 SOAP 效率不高，因为其过度使用 XML 强加的标记将信息组织为较好形式的文档。因此，已经提出了支持 Web 服务的 SOAP/XML 轻量级替代
67 方案。最相关的替代是表征状态转移协议（REST），REST 提供了用客户端 / 服务器模型设计基于网络的软件系统的模型，并无附加条件地利用由 HTTP 为实现 IPC 而提供的功能。

在 RESTful 系统中，客户使用标准 HTTP 方法（PUT、GET、POST 和 DELETE），通过 HTTP 发送一个请求，服务器发出一个包含资源表现形式的响应。依靠这种最小支集，可以提供任意所需的替换 SOAP 的基本功能也是最重要的功能，即方法调用。GET、PUT、POST 和 DELETE 方法构成了检索、添加、修改和删除数据的最小操作集合。利用 URI 识别资源可以实现 Web 服务所需的所有原子操作。数据的内容仍然由 XML 传输，并作为 HTTP 内容的一部分，但是 SOAP 需要的附加标记被删除了。因此，REST 代表了轻量级 SOAP，在超出 HTTP 管理范围的某些方面，它可以有效地工作。例如安全性，除 HTTP 所提供的安全性以外，RESTful Web 服务可以在不需要附加安全性的环境中执行。RESTful Web 服务十分流行且用于企业级传输功能，Twitter、Yahoo！（搜索 API、地图、照片等）、Flickr 和 Amazon.com 都使用了 REST。

Web 服务描述语言（WSDL）[22] 是用于描述 Web 服务的基于 XML 的语言。WSDL 按照被调用的方法以及所需参数和返回值的类型和结构来定义 Web 服务的接口。在图 2-18 中，我们注意到，调用 GetStockPrice 方法和接收结果的 SOAP 消息中，并不包含任何有关参数和返回值的类型和结构的信息。这一信息被保存在 Web 服务附带的 WSDL 文档中。因此，Web 服务消费者的应用程序已经知道需要哪些类型的参数，以及如何呈现结果。作为基

于XML的语言，WSDL允许自动生成可以轻易嵌入现有应用中的Web服务客户端。此外，XML是一种与平台和语言无关的规范，因此Web服务客户端可以由能够解释XML数据的任意语言来生成。这一基本特征使得Web服务具有互通性，也是这项技术成为实现SOA的首选方案的原因之一。



图2-18 用于Web服务方法调用的SOAP消息

除了直接支持Web服务的技术以外，其他Web 2.0[27]技术为Web应用和基于SOA的系统构建提供了丰富的和增强的功能。例如异步JavaScript和XML（AJAX），以及JavaScript标准对象符号（JSON）等框架。AJAX是一个基于JavaScript和XML的概念框架，利用Web浏览器的计算能力，在Web应用中实现异步操作，将简单的Web页面转换为成熟且丰富的应用。AJAX利用XML来交换Web服务和应用的数据，JSON可代替XML，以平台无关的方式表示对象和对象集合。通常更倾向于在AJAX环境中传递数据，因为与XML相比，AJAX是一个轻量级表示法，它能够以更简洁的形式传输等量的信息。

4. 面向服务和云计算

Web服务和Web 2.0相关技术构成了云计算系统和应用的基本构件。Web 2.0应用程序是云计算系统的前端，既可以通过Web服务交付服务，又可以与基于AJAX的客户端进行

有效的交互。从本质上讲，云计算提出了一切皆可作为服务（XaaS）的构想：基础设施、平台、服务和应用都可以作为服务。整个IT计算栈——从基础设施到应用——都可以依靠云计算服务来组合实现。在这种情况下，SOA是首选方法，因为它以服务的观点提出架构、组合和部署软件系统的设计原则。因此，面向服务是构建云计算系统的本质方法，它提供了灵活地组合并将更多功能集成到现有软件系统的方法。云计算也被用来按需地弹性化扩展和增加现有软件应用的功能。通过采用平台无关的技术，面向服务模型提高了互操作性。在这种情况下，它是解决集成问题并促进云计算应用的基本方法。

本章小结

本章介绍了并行计算和分布式计算，以便更好地理解云计算。并行和分布式计算的出现，首次使用了多个处理单元以及网络中多个计算节点，为复杂（“巨大挑战”）问题提供了解决方案。从串行到并行以及分布式处理的转变，为应用提供了高效的性能和可靠性。但在硬件架构、进程间通信技术以及算法与系统设计方面，这种转变也带来了新的挑战。本章讨论了支持并行处理的技术的演化，并介绍了用于设计和实现分布式系统的主要参考模型。

并行计算引入了在单一计算节点或在一组同构硬件的紧耦合节点上执行多任务的模型和架构。并行性通过能够并行处理多条指令的硬件来实现。不同的架构利用并行性来提高计算系统的性能，而这取决于数据并行、指令并行，或数据与指令同时并行处理。并行应用的开发常要求特定的环境和编译器，以提供对底层基础设施的高级功能的透明访问。

并行和分布式计算的结合，使人们可以利用一组网络化的异构计算机，并将这些作为统一的资源来处理。分布式系统组成了一个庞大的体系，对多种不同的软件系统进行了分类。架构模式有助于这种分类，并为分布式系统提供了参考模型。更准确地说，软件架构模式定义了组件和角色的逻辑结构，而系统架构模式更关注此类系统的物理部署。本章简要地回顾了主要的参考软件架构模式，并讨论了最为重要的系统架构模式：客户端/服务器模型和对等模型。这两种架构模式是任何分布式系统的基本配置单元。尤其是客户端/服务器模型，它是分布式系统中最流行的组件间交互模式的基础。

进程间通信（IPC）是分布式系统中的基本要素：它使得独立的进程结合在一起，并被
69 视为一个整体。基于消息的通信是与IPC最为相关的抽象概念，形成了关于IPC的几种不同技术，如远程过程调用、分布式对象、服务。本章回顾了分布式系统的组件间通信的参考模型，并介绍了每个抽象模型的主要特征。

云计算利用这些模型、抽象和技术，提供了设计和使用分布式系统的更加有效的方法，即按需提供整个系统或组件。

习题

1. 并行计算和分布式计算的区别是什么？
2. 阐述并行处理成为值得关注的计算技术的原因。
3. 什么是SIMD架构？
4. 列举并行计算系统的主要分类。
5. 描述计算系统中可以获得的并行级别。
6. 什么是分布式系统？具有分布式系统特征的组件有哪些？
7. 什么是架构模式？在分布式系统中它的作用是什么？

8. 列举最主要的软件架构模式。
9. 什么是基本系统架构模式?
10. 在分布式系统中，和进程间通信最相关的抽象模型是什么?
11. 论述基于消息通信的最重要的模型。
12. 论述 RPC 及其实现进程间通信的方法。
13. 分布式对象和 RPC 的区别是什么?
14. 什么是对象激活和生命周期? 它们如何在分布式系统中影响状态的一致性?
15. 与分布式对象编程最相关的技术是什么?
16. 详细叙述 CORBA。
17. 什么是面向服务的计算?
18. 什么是面向市场的云计算?
19. 什么是 SOA?
20. 论述支持服务计算的相关技术。

第3章

Mastering Cloud Computing: Foundations and Applications Programming

虚 拟 化

虚拟化技术是云计算的基本组件，在基础设施即服务方案中，虚拟化技术尤为重要。虚拟化技术为应用程序的运行创建了安全的、可定制的、独立的执行环境，即使某一应用程序是不可信的，也不会影响其他用户应用程序的运行。这项技术的基础是模拟独立于程序托管系统的应用执行环境的计算机程序或是软硬件集成系统。例如，我们可以在虚拟机上运行Windows操作系统，即便该虚拟机本身在Linux操作系统上运行。虚拟化技术使我们能以最小的成本建立弹性的、可扩展的、提供额外功能的系统。虚拟化技术广泛用于按需提供的可定制计算环境。

本章介绍虚拟化的基本概念及其发展过程，以及在云计算环境中使用的多种模型和技术。

3.1 简介

虚拟化技术具有庞大的技术与概念体系，提供用于应用程序运行的抽象环境，既包括虚拟硬件也包括操作系统。术语虚拟化通常指硬件虚拟化，在为云计算环境有效地提供基础设施即服务（IaaS）的方案中起到关键作用。事实上，虚拟化技术在计算机科学史中已经有长久的发展，并已在操作系统级、编程语言级和应用级实现了虚拟化环境。而且，虚拟化技术不仅为应用执行提供了虚拟环境，也可用于存储、内存和网络的虚拟化。

最初，虚拟化技术很少得到研究和应用，但最近几年，这项技术在持续地快速发展。以下现象使虚拟化技术重新受到关注：

- 更高的性能和计算能力。如今，一般用户的台式计算机（PC）已经可以满足几乎所有日常计算需求，很少需要额外的计算能力。几乎所有PC都有足够的资源承载虚拟机管理器，并以可接受的性能执行虚拟机。高端计算机市场亦是如此，例如超级计算机能提供可执行数百或数千虚拟机的巨大计算能力。
- 未充分利用的硬件和软件资源。硬件和软件未充分利用是由于：性能和计算能力提高；资源有限或零星使用。如今计算机如此强大，大多数情况下应用或系统只使用了小部分计算能力。此外，如果我们考虑企业的IT基础设施，许多计算机都只是部分被使用，然而它们却可以365天不间断地工作。例如，台式PC主要用于办公自动化，行政人员在工作时间操作这些机器，夜晚则完全无人使用，若工作时间之外将这些资源用于其他用途，将可以提高IT基础设施的效率。为了透明地提供这样的服务，需要部署一个完全独立的环境，这可以通过虚拟化来实现。
- 空间不足。对存储或计算能力的持续需要，使数据中心快速增长。谷歌和微软等公司通过建立大如足球场、能够承载上千个节点的数据中心，来扩大其基础设施服务。虽然这对于IT巨头是可行的，但在大多数情况下，一般企业没有能力再建一个数据中心来获得额外的资源能力。这种情况再加上硬件的利用不足，产生了所谓的服务

器整合技术[⊖]，虚拟化技术是其关键。

- 绿色节能措施。最近，许多公司正在寻找减少能源消耗以及减少碳排放量的方法。数据中心是主要的能源消费者，并且持续对环境产生影响。维护数据中心的运营不仅涉及保持服务器运行，还包括提供处理机冷却所消耗的大量能源。设备冷却对数据中心的碳排放量有显著影响。因此，通过服务器整合来减少服务器的数量必然会减少数据中心的冷却影响和能源消耗。虚拟化技术为服务器整合提供了有效的方法。
- 管理成本上升。现在能耗和冷却成本已经变得比IT设备的成本更高。此外，随着数据中心对服务器能力需求增加，管理成本显著上升。计算机尤其是服务器不是自己运行，而是需要系统管理员对其进行管理。常见的系统管理任务包括硬件监控、更换故障设备、服务器设置和更新、服务器资源监控和备份。这些工作需要大量劳动密集的操作，并且需要管理的服务器的数量越多，管理成本就越高。对于给定的工作负载，虚拟化技术减少了所需的服务器数量，从而降低了管理成本。

以上因素促进了硬件虚拟化以及其他类型的虚拟化方案的盛行。虚拟化技术被人们接受，首先是由于基于虚拟机的编程语言的广泛传播。1995年Sun公司发布了Java，很快得到开发人员认可。Java小应用程序（applet）的集成功能使Java成为一个非常成功的平台， 72
2000年年初，Java在应用服务器市场发挥了重要作用，从而证明了现有技术可以支持企业级应用程序执行。2002年，微软发布了可替代Java技术的.NET框架的第一版。与Java的原理相同，.NET能够支持多种编程语言，并可以与其他微软技术完全整合，.NET框架很快成为微软的主要开发平台，并迅速被开发人员采用。2006年，谷歌软件开发的二种“官方语言”中，Java和Python都基于虚拟机模型。从编程语言向虚拟化迁移的趋势表明了一个重要事实：技术已经准备好支持虚拟化解决方案了，且不会对性能带来显著的负面影响。这为实现底层实现虚拟化技术铺平了道路，目前虚拟化已经成为所有数据中心基础设施管理的最基本条件。

3.2 虚拟化环境特点

虚拟化是一个广义的概念，指的是构建硬件、软件环境、存储或网络的虚拟环境。虚拟化环境中包括三个主要组件：客户机、主机和虚拟化层。客户机与虚拟化层交互而不与主机交互，表示虚拟化的机器。主机是管理客户机的原始环境。虚拟化层负责重新创建相同的或不同的客户机运行环境（见图3-1）。

这种一般性的抽象概念体现了虚拟化技术的不同应用和实现。最直观和最常见的是硬件虚拟化，这是虚拟化概念的最早实现[⊖]。在硬件虚拟化环境中，客户机是一个包括操作系统和已安装应用程序的系统镜像。它们被安装在由虚拟化层控制和管理（称为虚拟机管理器）的虚拟设备上。主机指物理硬件，在某些情况下指操作系统，是虚拟机管理器的运行环境。在虚拟存储的情况下，客户机是与虚拟存储管理软件进行交互的客户端应用程序或用户，虚拟存储管理软件被部署在真实存储系统上。虚拟网络的情况与之相似：客户机的应用程序和用户与虚拟网络交互，如虚拟专网（VPN），由特定的软件（VPN客户端）利用可用的物理

⊖ 服务器整合技术就是将多个最初部署在不同服务器的服务和应用集成到一台物理服务器上的技术。服务器整合能够降低数据中心的功耗并解决硬件利用率不足问题。

⊖ 虚拟化技术最初是大型机时代的技术成果。IBM的CP/CMS大型机首先引入硬件虚拟化和虚拟机程序的概念。这些系统能够同时运行多个操作系统，提供了一个向后兼容的环境，允许客户运行其应用程序的早期版本。

网络节点进行管理。VPN 用于创建不同物理网络的虚拟环境，以访问其中资源，否则将无
73 法使用物理网络的资源。

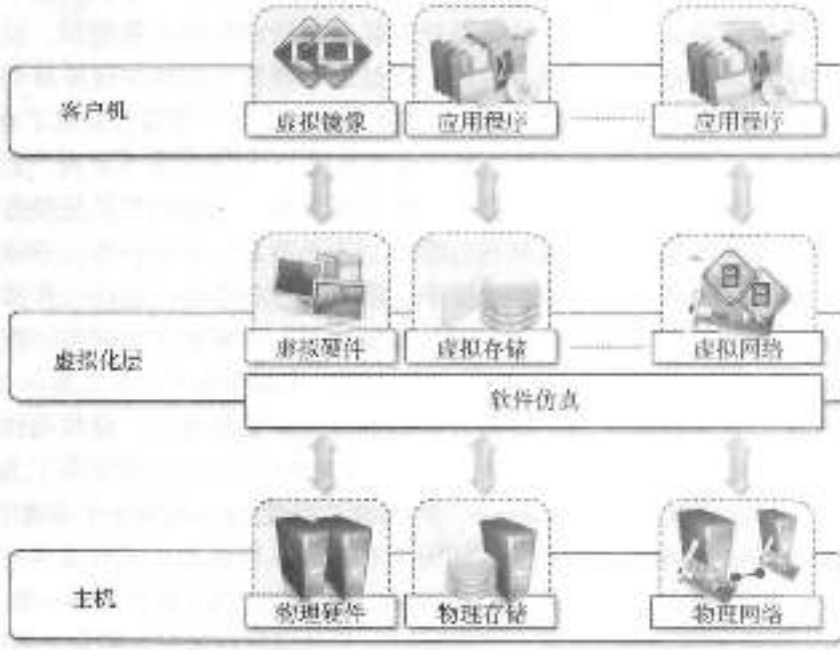


图 3-1　虚拟化参考模型

这些不同的虚拟化实现方式的共同特点是虚拟环境由软件编程创建。用软件来模拟各种环境的方法创造了诸多机遇，而之前其吸引力不足是由于虚拟化层的开销过多。今天的技术使得人们可以利用虚拟化获益，并能够充分利用虚拟化技术的优势。这些优势也一直是虚拟化解决方案的特点。

3.2.1　更强的安全性

虚拟化技术能够以完全透明的方式控制客户机的执行。这为提供一个安全的、可控的执行环境开辟了新道路。虚拟机就是客户机运行的模拟环境。客户机的所有操作都针对虚拟机，虚拟机将会把这些操作转换并应用到主机中。这种间接的方法允许虚拟机管理器控制
74 和过滤客户机的活动，从而阻止一些非正常操作的执行。由主机公开的资源可以被客户机隐藏或被简单地保护。此外，主机包含的敏感信息可以自然地隐藏而不需要配置复杂的安全策略。处理不受信任的代码时，需要更高的安全性。例如，从网络下载 Java 小应用程序运行在 Java 虚拟机（JVM）的沙箱版本[⊖]时，它们只能访问主机操作系统的有限资源。JVM 和 .NET 运行时都为定制的应用执行环境提供了广泛的安全策略。硬件虚拟化解决方案，如 VMware Desktop、Virtual Box 以及 Parallels 都可以用定制的虚拟硬件创建虚拟计算机，并能够在虚拟计算机上安装新的操作系统。默认情况下由虚拟计算机管理的文件系统完全从主机中分离，这是运行程序的理想环境，不会影响到环境中的其他用户。

⊖ 术语沙箱是指一个隔离的执行环境，其中的指令在实际执行环境中翻译和执行之前将被过滤和封锁。Java 虚拟机（JVM）的沙箱版本指的是通过安全策略封锁潜在威胁指令的 Java 虚拟机。

3.2.2 执行管理

执行环境的虚拟化不仅可以提高安全性，也可以实现更广泛的功能特性。共享、聚合、仿真和隔离是其最相关的特征（见图3-2）。

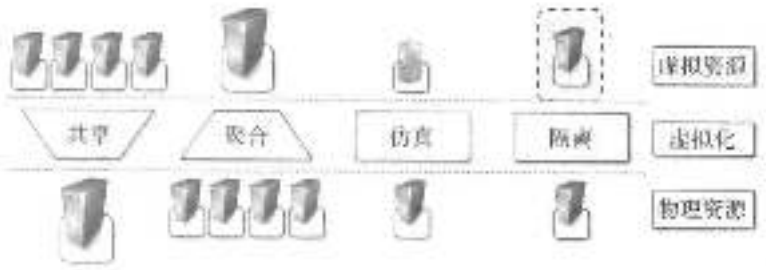


图3-2 执行管理的功能

- 共享。虚拟化技术可以在同一主机创建多个独立的计算环境。这样可以开发没有被充分利用的客户机能力。在后面的章节中会看到，共享是虚拟化数据中心非常重要的特征，可以减少活动服务器的数量并降低能耗。
- 聚合。虚拟化技术不仅可以在多个客户机之间共享物理资源，也可以聚合资源，这是两个相对的过程。一组独立的主机可以连在一起，作为一个单一的虚拟主机呈现给客户。此功能自然适用于分布式计算的中间件实现，集群管理软件就是一个经典例子，它能够利用一组同构机器的物理资源，并将其呈现为单一资源。
- 仿真。客户机程序在由虚拟化层控制的环境中执行，虚拟化控制本身也是一个程序。通过该程序可以控制和调整提供给客户机的环境。例如，主机可以仿真完全不同的环境，使得需要特定条件的客户机程序得以执行，这种特定环境在物理主机中无法实现。此特点可以用于测试，一个特定的客户机程序需要在不同的平台和架构上进行测试，程序中的很多功能在开发时很难验证。另外，硬件虚拟化能够提供虚拟硬件和仿真特定类型的设备，如用于文件I/O的小型计算机系统接口（SCSI）设备，而主机并没有安装该设备。不满足当前系统要求的旧软件可以在仿真硬件上运行而无需更改代码，这可以通过仿真需要的硬件架构或在特定的操作系统沙箱内实现，如Windows 95/98的MS-DOS模式。仿真的另一个例子是街机游戏模拟器，可以在普通的个人电脑上玩街机游戏。
- 隔离。虚拟化技术为客户机提供了完全独立的可执行环境，无论是操作系统、应用程序还是其他。客户机程序通过与抽象层交互来执行任务，该抽象层支持对底层资源的访问。隔离有许多好处，例如，允许多个客户机在同一主机上运行而相互不影响。此外，还可将主机与客户机分离。虚拟机可以过滤客户机的活动，从而阻止对主机造成损害的操作。

除了这些特点，虚拟化的另一重要特性是性能调整。支持虚拟化的硬件和软件的巨大进步促进了该特性的实现。通过调整虚拟环境资源的属性，很容易控制客户机的性能。这种特性使得我们能更有效地实现高服务质量（QoS）的基础设施，以满足客户机所要求的服务等级协议（SLA）。例如，硬件虚拟化程序仅仅给客户机操作系统分配一部分主机内存，或是设置虚拟机处理器的最大主频。执行管理的另一个优点是可以轻松捕获客户机程序的状态，并维持和重新执行。例如，虚拟机管理器（如Xen Hypervisor）能停止客户机操作系统的执

行，将其虚拟镜像到另一台计算机，以完全透明的方式来重新执行。这种技术称为虚拟机迁
[76] 移，这是虚拟数据中心的一个重要特征，用于有效优化应用需求。

3.2.3　可移植性

根据不同类型的虚拟化，可移植性的概念有不同的应用方式。在硬件虚拟化方式下，客户机被转成虚拟机镜像文件，并在不同虚拟机上安全地迁移和执行。不考虑文件的大小，可以很容易地在不同的计算机上显示图像文件。虚拟镜像通常是需要特定的虚拟机管理器来执行的特有格式。在编程级虚拟化的情况下，由 JVM 或 .NET 运行时实现，二进制代码表示的应用程序组件（jar 包或组件）不需要重新编译就可以在任何虚拟机上执行。这使得应用程序的开发周期更灵活，其部署也非常简单；在大多数情况下，某一版本的应用程序无需改变就可以在不同平台上运行。最后，只要具有虚拟机管理器，可移植性就使得系统随时可以使用。通常来讲，不管在哪里都不需要完全具备所需的应用和服务。

3.3　虚拟化技术分类

虚拟化技术涵盖了计算机不同应用领域的多种仿真技术。对这些技术的分类有助于更好地了解其特性与应用（见图 3-3）。

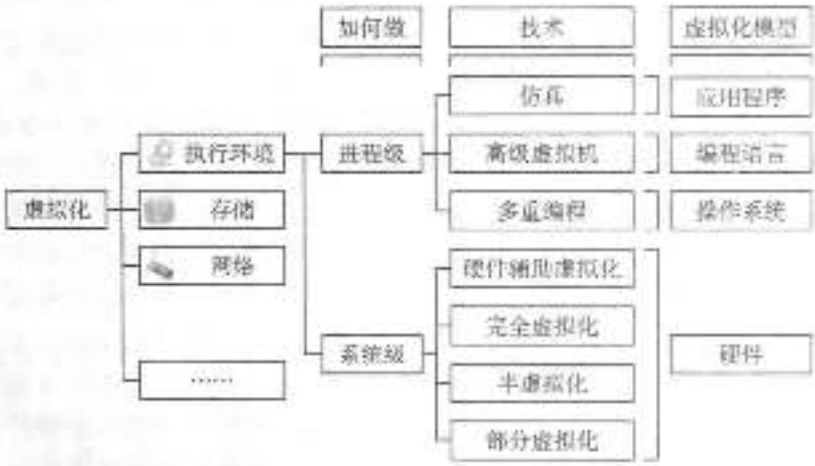


图 3-3　虚拟化技术分类

第一种方法根据仿真的服务和实体分类。虚拟化主要用于模拟执行环境、存储和网络。其中，执行环境虚拟化是流行最早和发展最快的领域，因此值得我们重点研究和进一步细分。特别是可以按照所需要的主机类型将执行环境虚拟化技术划分为两大类：进程级技术在能够完全控制硬件的操作系统上实现；系统级技术直接在硬件中实现，不需要或只需要很少的操作系统支持。根据这两类可以列出为客户机提供不同类型的虚拟计算环境的各种技术：硬件级虚拟化、操作系统级虚拟化、编程语言级虚拟化、应用级虚拟化。

3.3.1　执行虚拟化

执行虚拟化包含了所有用于模拟执行环境的技术，该执行环境与虚拟层分离。这些技术主要是为执行程序提供支持，不管是操作系统、编译的二进制程序还是应用程序。因此，执
[77] 行虚拟化可直接在硬件上通过操作系统、应用程序、动态库或静态链接到应用的镜像来实现。

1. 机器参考模型

在不同计算栈层建立虚拟化执行环境，需要定义抽象层之间接口的参考模型，以隐藏实现细节。从这个角度来讲，虚拟化技术实际上是替换其中的某一层，并拦截对该层的请求。因此，层与层之间的独立性简化了虚拟化的实现，这只需要接口的仿真和与底层之间的适当交互。

现代计算系统可以描述为如图 3-4 所示的参考模型。底层硬件模型以指令集架构（ISA）表示，ISA 定义了处理器、寄存器、存储器、中断管理的指令集。ISA 是硬件和软件的分界线。对于操作系统开发者（系统 ISA）和直接管理底层硬件的应用程序开发者（用户 ISA）很重要。应用程序二进制接口（ABI）将操作系统层与由操作系统管理的应用程序和库分隔开来。ABI 涵盖了各种细节，如低级数据类型、对齐调用约定、可执行程序的定义格式。系统调用在 ABI 上定义。该接口使得应用程序和库可以在兼容 ABI 系统中实现可移植性。抽象模型的最高层是应用编程接口（API），API 将应用程序与库和底层操作系统连接起来。 78

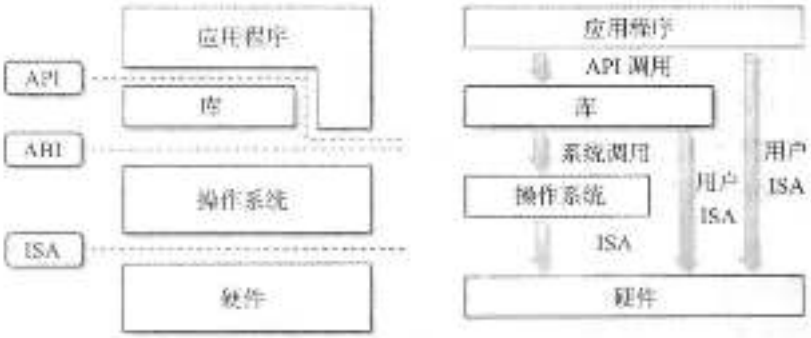


图 3-4 机器参考模型

ABI 和 ISA 负责完成对应用程序级 API 的任何操作。高层抽象被转化为机器级的指令，以执行处理器支持的实际操作。机器级的资源，如处理器寄存器和主存储器容量，都用来执行中央处理单元（CPU）的硬件级操作。这种分层方法简化了计算系统的开发和实现，并简化了多执行环境中多任务的实现和共存。事实上，此模型不仅不需要完全理解整个计算技术栈，而且提供了实现管理和访问共享资源的最精简安全模型的方法。

为此，硬件指令集被分为不同的安全等级，定义谁可以对其进行操作。首先可以分为特权指令和非特权指令。非特权指令不能改变共享资源的值或状态，用于不受其他任务干扰的操作，因为它们不访问共享资源，如浮点指令、定点指令和算术指令。特权指令是在特定的限制条件下执行用来访问共享资源的值或状态的指令，主要用于敏感操作，显示（行为敏感）或修改（控制敏感）权限状态。例如，行为敏感指令是那些操作 I/O 的指令，而控制敏感指令是改变 CPU 寄存器状态的指令。某些系统拥有不止一类特权指令，因此可以更好地控制对指令的访问。例如，实现权限等级的基于环的安全体系结构（见图 3-5）将特权级分为 4 级：0 环、1 环、2 环、3 环。0 环特权级别最高，3 环特权级别最低。操作系统的内核工作在 0 环，1 环和 2 环用于操



图 3-5 安全环和特权模式

作系统级的服务，用户应用程序工作在3环。目前大部分操作系统都支持0环和3环两个层
79 次，分别对应管理员模式和用户模式。

目前所有的系统都支持至少两种模式：管理员模式和用户模式。管理员模式中，所有指令（特权与非特权）可以没有任何限制地执行。这种模式也称为主模式或内核模式，通常用于操作系统（或虚拟机管理程序）对硬件级资源的敏感操作。在用户模式下，对机器级资源的控制受限。如果在用户模式下运行的代码调用特权指令，将会产生硬件中断和自陷指令的潜在风险。尽管这样，依然存在一些指令在一些条件下作为特权指令被调用，在其他条件下作为非特权指令被调用。

用户和管理员模式之间的区别，使我们了解了虚拟机管理程序（hypervisor）的作用及其由来。从概念上讲，虚拟机管理程序在管理员模式上执行，并使用前缀 hyper-。实际上，必须以管理员权限运行虚拟机管理程序，特权和非特权指令之间的区别为虚拟机管理器设计带来了挑战。所有敏感指令都应在特权模式下执行，通过管理员权限来避免自陷。否则不可能完全模拟和管理客户机操作系统的CPU状态。然而原有的指令集不能实现，因为有17条敏感指令不属于特权指令。这样就不能由单一虚拟机管理程序管理多操作系统的独立运行，因为它们能够访问并改变处理器的特权状态[○]。最近ISA（Intel VT和AMD Pacifica）的实现方案已经解决了这个问题，方法是将这些指令重新设计为特权指令。

通过这个参考模型，能够探索和更好地理解虚拟化执行环境的各种技术及其与系统其他
80 组件的关系。

2. 硬件级虚拟化

硬件级虚拟化是一种为运行客户机操作系统的计算机提供抽象执行环境的虚拟化技术。在此模型中，客户机指操作系统，主机指物理存在的计算机硬件，虚拟机指模拟的计算机，虚拟机管理程序指对虚拟机的监控管理（见图3-6）。虚拟机管理程序通常是一个程序或者是软件和硬件的集成，可以执行底层物理硬件的抽象。

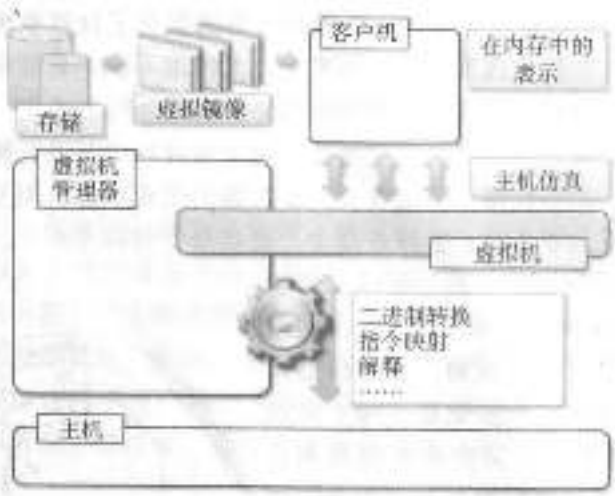


图3-6　硬件虚拟化参考模型

○ 虚拟机管理程序的管理环境中，所有客户机操作系统的代码将运行在用户模式下，以防止直接访问CPU的状态。如果有敏感指令以用户模式被调用（即以非特权指令执行），那么将不可能完全隔离客户机操作系统。

硬件虚拟化也称为系统虚拟化，虚拟机支持完整的ISA。这与进程虚拟机是不同的，进程虚拟机提供ABI支持。

（1）虚拟机管理程序

硬件虚拟化最关键的组件是虚拟机管理程序，或称为虚拟机管理器（VMM）。它能在客户机操作系统安装环境中重新创建硬件模拟环境。虚拟机管理程序主要有两种类型：Ⅰ型和Ⅱ型（见图3-7）。

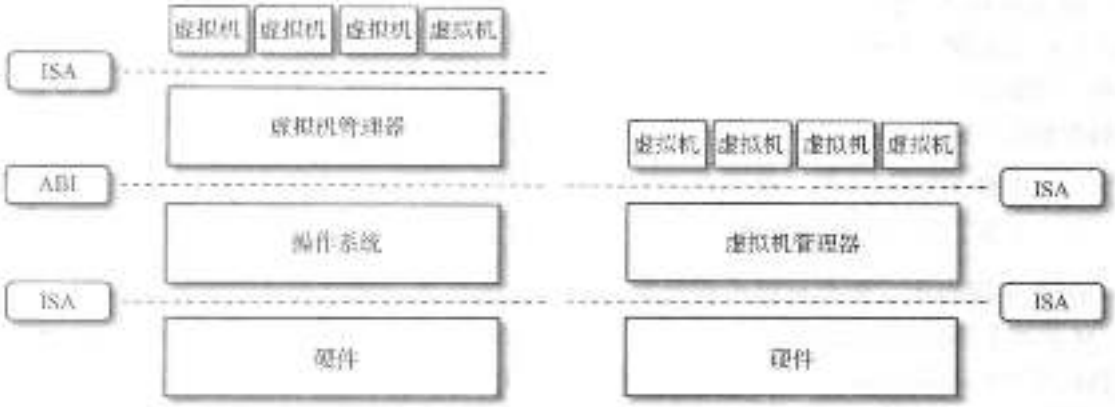


图3-7 宿主（左）和原始（右）虚拟机。此图是两种类型虚拟机管理程序的图形表示

- Ⅰ型的虚拟机管理程序直接在物理机器上运行。因此，虚拟机管理程序代替操作系统直接与底层硬件的ISA接口交互，并且模拟这个接口以实现对客户机操作系统的管理。此类虚拟机管理程序也称为原始虚拟机，因为它只运行在本机的硬件上。 81
- Ⅱ型的虚拟机管理程序依赖主机操作系统提供虚拟化服务。这意味着虚拟机管理程序是由操作系统管理的程序，通过ABI与操作系统交互，并为客户机操作系统模拟虚拟硬件的ISA。此类虚拟机管理程序也称为宿主虚拟机，因为它由操作系统托管。

从概念上讲，虚拟机管理器的功能主要包括三个模块（见图3-8）：调度器、分配器和解释器，用于协调活动以模拟底层硬件。调度器是虚拟机监控器的入口点，将虚拟机实例发出的指令分发给分配器或解释器。分配器负责分发要提供给虚拟机的系统资源；当虚拟机试图执行旨在改变其硬件资源分配的指令时，分配器就由调度器调用开始工作。解释器模块由一组解释例程组成，当虚拟机执行特权指令时将会触发自陷（trap），并执行相应的解释例程。

虚拟机管理器的设计和架构以及主机底层硬件的设计，决定了硬件虚拟化的实现方式。客户机操作系统可以透明地在VMM上执行，就像在底层硬件上运行一样。有效地支持虚拟机管理器虚拟化的标准由Goldberg和Popek于1974年建立[23]。虚拟机管理器必须具备三个特性：

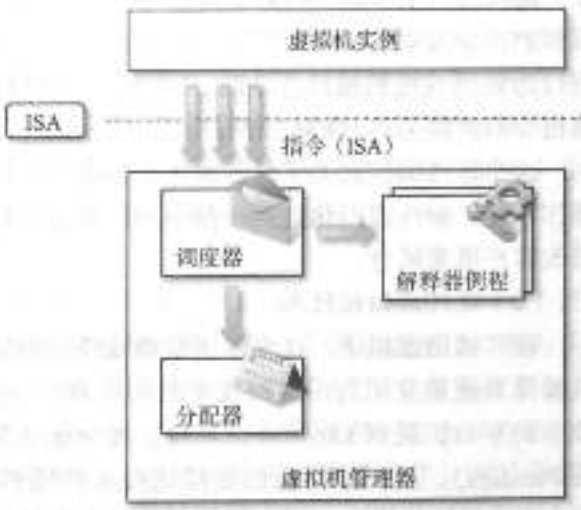


图3-8 虚拟机管理程序参考架构

- 等价性。运行于VMM的程序，其行为应与其直接运行于物理主机上的行为完全一致。
- 资源控制。VMM对虚拟资源应该进行完全控制。
- 效率。经常使用的机器指令应该在没有VMM的干预下执行。

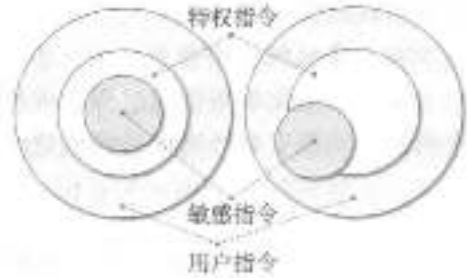


图3-9　可虚拟化的计算机（左）和不可虚拟化的计算机（右）

要满足这些特性，托管虚拟机管理器的主机的ISA结构是关键。Popek和Goldberg对指令集进行了分类，并提出三个定理，定义了有效支持虚拟化的硬件指令特征（见图3-9）。

定理3.1　对于任何传统的第三代计算机，只要其敏感指令集是特权指令集的一个子集，就可以为其构建虚拟机管理器。

这条定理指出，所有更改系统资源配置的指令应该以用户模式产生一个自陷，并在VMM控制下执行。这使得虚拟机管理程序只能有效控制那些展现抽象层的指令，同时在执行其他指令时不以性能损失为代价。当虚拟机管理程序处于最高特权模式（0环）时，该定理始终保证了对资源的有效控制。非特权指令必须在不受虚拟机管理程序控制下执行。等价特性也得到满足，在这两种情况下代码的输出是一样的，因为代码没有改变。

定理3.2　传统的第三代计算机可以递归虚拟化，条件如下：

- 可虚拟化。
- 可以为其建立一台不具有任何时间依赖性的VMM。

递归虚拟化是在另一个VMM上运行VMM的能力。它允许虚拟机管理程序嵌套，只要底层资源可以容纳。可虚拟化硬件是递归虚拟化的前提条件。

定理3.3　基于混合虚拟化技术的VMM可以在任何现有传统第三代计算机上构造，只要其用户敏感指令集是特权指令集的子集。

还有另一个术语——混合型虚拟机（HVM），它比虚拟机系统效率低。在HVM情况下，更多的指令需要被解释而不是直接执行。所有指令在虚拟管理员模式下都需要解释。每当执行行为敏感或控制敏感指令时，HVM直接控制其执行或通过自陷进行控制。这里，所有敏感指令都由模拟的HVM控制。

这个参考模型代表了最普遍的经典虚拟化模型，也就是能够在独立的环境中执行客户机操作系统。硬件虚拟化包括一些不同的策略，根据底层硬件要求、主机的抽象以及是否应该修改客户机来区分。

（2）硬件虚拟化技术

硬件辅助虚拟化。这个术语是指硬件提供架构支持，帮助创建虚拟机管理器并允许客户机操作系统独立运行。这种技术最初由IBM System/370引入。目前，支持硬件辅助虚拟化技术的平台扩展到x86-64位系统，如Intel VT（以前称为Vanderpool）和AMD V（以前称为Pacifica）。这两家厂商的虚拟化技术扩展不同，但都是为了减少用虚拟机管理程序模拟x86硬件的性能损失。引入硬件辅助虚拟化技术之前，x86硬件的虚拟化程序在性能方面代价昂贵。因为x86的设计架构不符合Popek和Goldberg提出的虚拟化规范要求，并且早期

的产品都是用二进制代码转换来获得敏感指令和提供模拟环境。如 VMware 虚拟平台，这是 VMware 在 1999 年推出的虚拟化技术，开创了 x86 虚拟化技术领域。2006 年以后，英特尔和 AMD 推出了扩展处理器，大量的虚拟化解决方案都采用了基于内核的虚拟机（KVM）、VirtualBox、Xen、VMware、Hyper-V、Sun xVM、Parallels 等。

完全虚拟化。完全虚拟化指运行程序的能力，类似操作系统无需修改就可以直接在虚拟机上运行，就像在物理机上运行一样，即所抽象的虚拟机具有完全的物理机特性。要做到这一点，VMM 必须模拟整个底层硬件资源。完全虚拟化的优点主要是完全独立，这可以增强安全性，易于模拟不同系统结构，并使得同一平台上的不同系统可以共存。而这是许多虚拟化解决方案的预期目标，完全虚拟化也带来了相关的性能和技术实现问题。一个关键的挑战是特权指令的拦截，如 I/O 指令，因为它们改变了主机提供资源的状态，因此必须在 VMM 中运行。实现完全虚拟化的简单方法是为所有指令提供一个虚拟环境，但性能也有一定的限制。成功和有效的完全虚拟化方法包括硬件和软件，并且不允许有潜在危害的指令直接在主机上执行。这就需要通过硬件辅助虚拟化来协助完成。

半虚拟化。这是一种不透明的虚拟化解决方案，允许实现简化 VMM。半虚拟化技术为虚拟机提供软件接口，操作系统需要进行修改才能够运行在半虚拟化环境中。半虚拟化的目标是将对性能要求较高的操作直接运行于主机上，从而减少托管运行的性能损失，使得 VMM 更容易将这些难以虚拟化的操作迁移到主机上执行。要利用此特性，需要修改客户机 85
操作系统，并通过虚拟机软件接口重新映射和迁移性能要求苛刻的操作。如果提供操作系统的源代码就可以采取此方式，这也是半虚拟化技术开放源代码并主要用于学术研究的原因。虽然这种技术最早在 IBM 的 VM 操作系统系列中得到应用，但半虚拟化一词首先在华盛顿大学的 Denali 项目 [24] 文档中被提出。该技术已成功应用于 Xen，提出了将 Linux 操作系统迁移到 Xen 的虚拟机管理程序上运行的虚拟化方案。不能移植的操作系统仍然可以利用半虚拟化，使用特定的设备驱动将指令运行于虚拟机管理程序提供的半虚拟化应用程序接口。Xen 提供了在 x86 系统上运行 Windows 操作系统的解决方案。其他使用半虚拟化技术的方案包括 VMWare、Parallels 及一些嵌入式和实时系统，如 TRANGO、Wind River 和 XtratuM。

部分虚拟化。部分虚拟化技术对部分底层硬件环境进行模拟，因而不支持独立的客户机操作系统。部分虚拟化支持多个应用程序透明地运行，但它不具有操作系统运行于完全虚拟化方式的所有特性。部分虚拟化可在分时系统中实现地址空间虚拟化，这使得多个应用程序和用户可以在独立的内存空间中同时运行，并共享相同的硬件资源（磁盘、处理器和网络）。在技术发展历程中，部分虚拟化是实现完全虚拟化的一个重要里程碑，曾应用于试验平台 IBM M44/44X。地址空间的虚拟化是现代操作系统的一个共同特征。

（3）操作系统级虚拟化

操作系统级虚拟化用来为那些同时管理的应用程序创建不同的单独执行环境。不同于硬件虚拟化，它没有 VMM 或虚拟机管理程序。这种虚拟化在单一操作系统中工作，该操作系统内核支持多个独立的用户空间实例。内核还负责在多个用户空间实例之间共享系统资源，并且防止实例间相互影响。用户空间实例一般是完全独立的文件系统视图，包含独立的 IP 地址、软件配置、设备访问方式。支持这种类型虚拟化技术的操作系统通常是分时操作系统，并能提供更强的命名空间和独立资源。

这种虚拟化技术可看作是 UNIX 系统中 chroot 机制的演变。chroot 操作将一个进程及其

子进程的文件系统根目录改变到一个指定目录。因此，该进程和子进程只能访问新的根目录下的文件系统。因为 UNIX 系统把设备作为文件系统的部件，所以使用该方法可以完全隔离一组进程。按照相同的原理，操作系统级虚拟化将应用程序运行于独立的多个执行容器中。与硬件虚拟化相比，这种策略几乎没有开销，因为应用程序可以直接使用操作系统调用，不需要模拟执行环境。没有必要像硬件辅助虚拟化技术一样，为了执行而修改应用程序，也不
86 需要修改任何特定的硬件。另一方面，操作系统级虚拟化不如硬件虚拟化技术灵活，因为所有的用户空间实例都必须共享相同的操作系统。

此技术是服务器整合的有效解决方案，多台应用服务器共享相同的技术：包括操作系统、应用服务器体系结构和其他组件。不同的服务器被集成到一台物理服务器，其中每台服务器都运行在不同的用户空间，并且与其他服务器完全隔离。

操作系统虚拟技术的例子包括：FreeBSD Jails、IBM Logical Partition（LPAR）、SolarisZones and Containers、Parallels Virtuozzo Containers、OpenVZ、iCore Virtual Accounts、Free Virtual Private Server（FreeVPS）等。这些技术所提供的服务有所不同。其中大部分基于 UNIX 系统，如 Solaris 和 OpenVZ，允许不同版本的同一操作系统同时运行。

3. 编程语言级虚拟化

编程语言级虚拟化主要用来轻松实现应用程序的部署和管理执行，以及跨不同平台和操作系统的移植。它包含一个运行进程编译程序的二进制代码的虚拟机。编译器利用这种技术来产生抽象系统的二进制机器代码。这种架构的特点因实现方法不同而各不相同。一般来说，这些虚拟机组成了简单的底层硬件指令集，并提供了一些高级指令以满足编译语言的特性。在运行时，二进制代码可以被实时或者及时（jitted）[⊖]地解释或编译为底层硬件指令集。

编程语言级虚拟化在计算机科学发展史上有很长的历史，最初于 1966 年用于基本组合编程语言（BCPL），这种语言用于编写编译器，是 C 语言的原型之一。使用这种技术的其他例子有 UCSD 的 Pascal 和 Smalltalk。Sun 公司 1996 年推出 Java 平台后，虚拟机编程语言再次流行起来。Java 最初创建时是开发互联网应用程序的平台，后来成为首选的企业级应用开发技术之一，并围绕它形成了开发者的大型社区。Java 虚拟机最初只能执行用 Java 语言编写的程序，但后来也能处理其他语言，如 Python、Pascal、Groovy 和 Ruby。支持多种编
87 程语言的能力是公共语言架构（CLI）的关键部分，CLI 属于 .NET 框架规范。目前 Java 平台和 .NET 框架是企业应用开发最流行的技术。

Java 和 CLI 是基于栈结构的虚拟机：抽象架构的参考模型是基于用于完成操作的执行栈结构。系统编译器生成的二进制代码包含了一组指令，这些指令用来将操作对象放入栈、执行操作并将结果放在栈中。此外，还包括调用方法、管理对象和类的指令。基于栈结构的虚拟机更容易被解释，语法分析也更简单，因此很容易移植到不同的架构。另一种是基于寄存器的虚拟机，该参考模型基于寄存器。这种虚拟机与我们今天使用的底层架构更加接近。Parrot 是基于寄存器的虚拟机实例，它最初被设计为支持 PERL 的编程级虚拟机，然后推广到动态语言的宿主执行。

编程级虚拟机也称为进程虚拟机，其主要优点是能跨不同的平台提供统一执行环境。被

⊖ jitted 这个词是 just-in-time（JIT）的缩写，非常通用。它是指调用方法时将方法的二进制代码编译为底层机器代码的特定执行策略，也就是恰好及时。编程级虚拟化最初的实现是基于指令译码，这将减慢执行速度。JIT 编译的优点是已编译的机器代码在未来实用相同方式调用时可以重复使用。实现 JIT 编译的虚拟机一般用方法缓存来存储每个方法生成的代码，且只在调用方法触发编译之前查询该缓存。

编译的二进制代码程序可在任何操作系统和平台上执行，只要该操作系统和平台部署了可以执行代码的虚拟机。从开发周期的角度来看，该方法简化了开发和部署工作，因为不需要提供相同代码的不同版本。虚拟机在不同平台上的实现仍然是一个高代价的任务，但只需做一次，而且不是针对所有的应用程序。此外，进程虚拟机支持更多的程序控制，因为其不允许直接访问内存。安全是编程语言虚拟机的另一个优势。通过过滤的I/O操作，进程虚拟机可以很容易地支持应用程序的沙箱操作。例如，Java和.NET提供了灵活的安全策略和代码访问安全框架。这些优势都是以性能为代价的。虚拟机编程语言比编译为实际系统语言的性能差，但这种性能差异正在变得越来越小，并且处理器的高运算能力使得这种差异不再重要。

这种模式的实现也称为高层次虚拟机，因为高层编程语言被编译成ISA，并进一步将ISA解释或动态地翻译成主机平台的具体指令。

4. 应用级虚拟化

当原有环境不支持应用所需特性时，应用虚拟化技术为此应用提供了虚拟的运行环境。在这种情况下，应用程序虽然未安装在预期的运行环境中，但却像运行在其中一样。一般情况下，这些技术大多涉及局部文件系统、库和操作系统部件模拟。该模拟是负责执行应用的 88
程序或操作系统组件层，也可用于执行基于不同硬件系统而编译的程序二进制代码。在这种情况下，可实现以下策略：

- 解释。在这种技术中，每一条指令都由模拟器解释成能执行的ISA指令，这将导致性能下降。解释过程开销很大，因为每条指令都要被模拟。
- 二进制翻译。在这种技术中，每一条源指令转换为具有同等功能的本地指令。许多指令翻译后被缓存并可以重新使用。二进制翻译的初始开销较大，但经过一段时间后性能会提高，因为以前翻译过的指令块可以直接执行。

如上所述，模拟与硬件级虚拟化不同。前者只是针对不同硬件的编译程序的执行，而后者可模拟完整的硬件环境，并可以在其中安装整个操作系统。

当主机操作系统缺少库时，应用程序虚拟化是很好的解决方案，在这种情况下，替代库可以与应用程序链接，或库的调用可以重新映射到主机系统的已有功能函数。另一优点是，虚拟机管理器要简单得多，因为与硬件虚拟化相比，应用程序虚拟化仅提供运行环境的部分模拟。此外，这种技术可以运行不兼容的应用程序。与虚拟机跨应用开发的编程级虚拟化技术相比，应用级的虚拟化提供了运行应用程序的特定环境。

Wine是最流行的应用程序虚拟化方案，允许UNIX操作系统执行基于Windows编写的程序软件。Wine中软件应用的容器为Winelib，包括客户机应用程序集合和库集，开发人员可以用它来编译应用程序以便移植到UNIX系统。Wine的灵感来自Sun的一款产品，Windows应用程序二进制接口（WABI），它是在Solaris上实现Win 16 API的规范。类似的Mac OS X环境的解决方案是CrossOver，支持直接在Mac OS X操作系统上运行Windows应用程序。VMware ThinApp是应用虚拟化的另一款产品，它将安装的应用程序封装成与主机操作系统无关的可执行镜像文件。

3.3.2 其他类型的虚拟化

除执行环境虚拟化外，其他类型的虚拟化提供了可交互的抽象环境，主要包括存储、网络以及客户端/服务器交互的虚拟化。

1. 存储虚拟化

存储虚拟化是一种系统管理方法，能够将硬件的物理结构表示为逻辑形式。使用这种技
89 术时，用户不必担心其数据的特定位置，只需使用逻辑路径来标识。存储虚拟化使得我们能利用大量的存储设备，并在单一的逻辑文件系统下管理和描述这些存储设备。有许多存储虚拟化技术，其中最常见的是基于网络的虚拟化，即存储区域网络（SAN）。SAN 通过高速带宽连接网络设备来提供存储能力。

2. 网络虚拟化

网络虚拟化技术将硬件设备和特定的软件结合以创建和管理虚拟网络。网络虚拟化将不同的物理网络集成为一个逻辑网络（外部网络虚拟化）或让操作系统分区具有类似于网络的功能（内部网络虚拟化）。外部网络虚拟化通常是一个虚拟局域网（VLAN）。VLAN 是主机的集合，主机之间相互通信，就像位于同一个广播域下。内部网络虚拟化通常与硬件和操作系统级虚拟化一起应用，为客户机提供虚拟的通信网络接口。有几种方式可实现内部网络虚拟化：客户机共享主机的相同网络接口，并使用网络地址转换（NAT）来访问网络；虚拟机管理器可以安装在主机、网络设备或者驱动上；或者客户机拥有一个专用网络。

3. 桌面虚拟化

桌面虚拟化技术将个人电脑的桌面环境抽象化，以便采用客户端/服务器的方式来访问。桌面虚拟化与硬件虚拟化具有同样的虚拟化环境，但服务目的不同。类似于硬件虚拟化，桌面虚拟化可以访问不同的系统，就好像它们都安装在本地主机上一样，实际上该系统存储在不同的远程主机上，并通过网络连接来访问。此外，桌面虚拟化实现了从任何地方都可以访问相同的桌面环境。尽管桌面虚拟化严格意义上是指远程访问桌面环境的能力，但通常这个桌面环境是被存储在远程服务器或数据中心，该数据中心能提供高可用性基础设施并确保数据的可访问性和持久性。

在这种情况下，支持硬件虚拟化的设备非常重要，它能访问托管在同一台服务器上的多个桌面环境。特定的桌面环境存储在虚拟机镜像中，在客户端连接到桌面环境时，按需进行加载和启动。这是典型的云计算场景，用户利用虚拟化设备执行其计算机上的日常任务。桌面虚拟化的优点是高可用性、持久性、可访问性和易于管理。正如 4.5.4 节将要讨论的，安全问题阻碍了桌面虚拟化技术的应用。用于远程访问桌面环境的基本服务的软件组件包括：Windows Remote Services、VNC 以及 X Server。基于云计算的桌面虚拟化基础设施包括：Sun Virtual Desktop Infrastructure（VDI）、Parallels Virtual Desktop Infrastructure（VDI）、
90 Citrix XenDesktop 等。

4. 应用服务器虚拟化

通过使用负载均衡策略和为应用服务器中的服务提供高可用性的基础设施，应用服务器虚拟化技术将多台提供相同服务的应用服务器抽象成一台虚拟应用服务器。这是虚拟化的具体形式，与存储虚拟化有相同的目的：提供更好的服务质量，而不仅是模拟一个不同的环境。

3.4 虚拟化和云计算

虚拟化在云计算中扮演着重要的角色，因为虚拟化能够支持适当的可定制特性、安全性、独立性和可管理性，这些都是按需提供 IT 服务最基本的特性。虚拟化技术主要用于提供可配置的计算环境和存储。网络虚拟化的应用不太广泛，大多数情况下只是构建虚拟计算机系统时提供的附加功能。

更重要的是虚拟计算环境和执行虚拟化技术所发挥的作用。其中，硬件和编程语言的虚拟化是云计算系统所采用的技术。硬件虚拟化是提供基础设施即服务（IaaS）的产品解决方案，而编程语言的虚拟化是用于平台即服务（PaaS）的技术。这两种情况下，提供可定制的和沙箱环境的能力为企业带来了机会，使企业拥有能够维持和处理庞大工作负载的大型计算基础设施。此外，虚拟化支持独立性和控制能力，从而简化服务的契约及服务供应商的责任。

除了提供所需的计算资源，虚拟化也可以通过整合手段透明地为云计算服务用户设计出更高效的计算系统。虚拟化允许创建独立和可控的环境，可以在这些环境中使用相同的资源而互不干扰。如果底层资源有足够的能力，则不需共享。对于资源未充分利用的情况，虚拟化特别有吸引力，通过在较少数量的资源上建立虚拟机来减少活动资源数并充分利用资源，这种方法也称为服务器整合，而虚拟机实例的移动则称为虚拟机迁移（见图3-10）。由于虚拟机实例处于可控环境，所以无论是暂时停止其执行并移动数据到新的资源，还是通过控制在运行时移动其实例，资源整合都不会产生什么影响。在运行时移动实例称为动态迁移，一般更难实现，却更加有效，因为虚拟机实例的活动从未中断[⊖]。 91

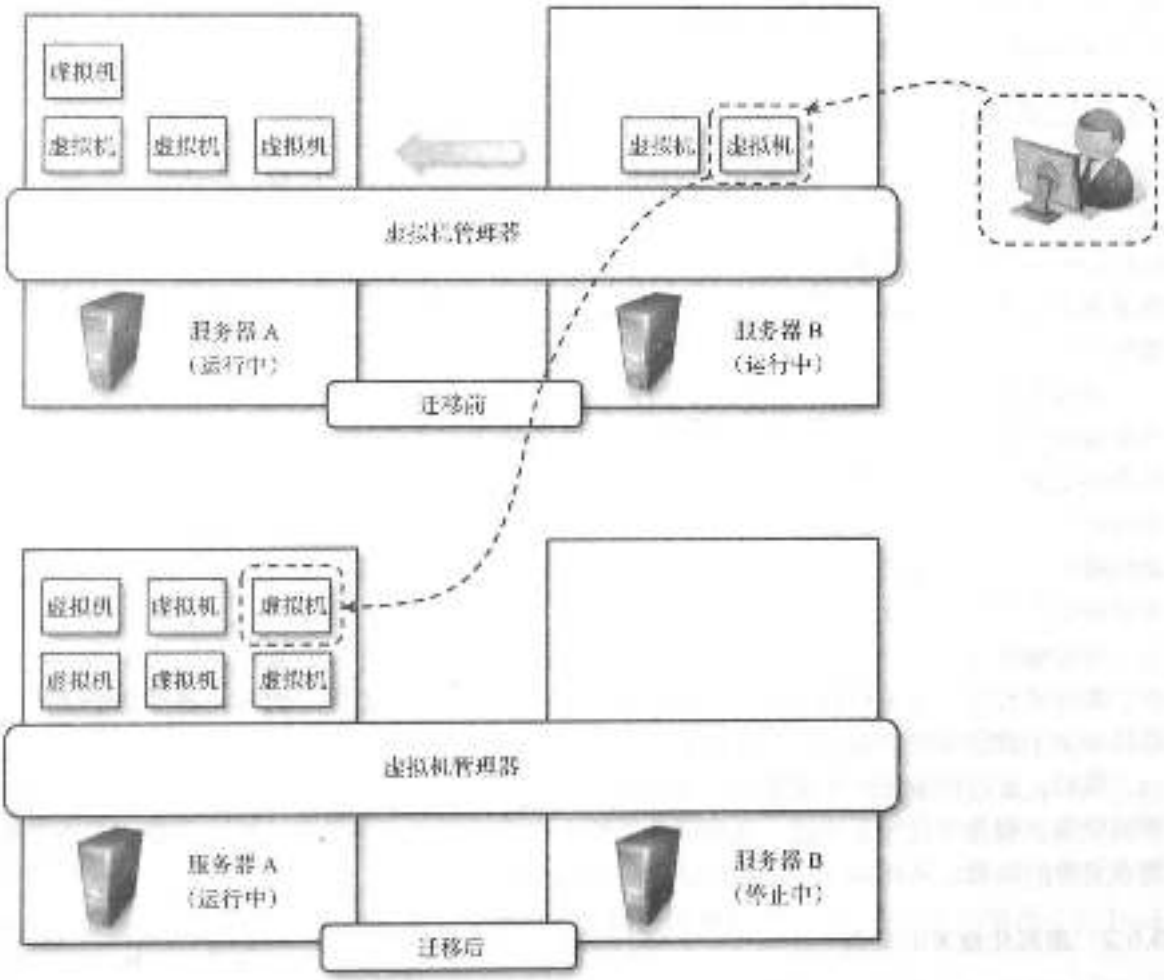


图3-10 实时迁移和服务器整合

⊖ 注意，云计算对开发按需扩展的应用程序非常重要。在大多数情况下，应用程序必须处理不断增加的负载或为更多的请求服务，这使得它们成为服务器应用程序。显而易见，在这种情况下实时迁移提供了一个更好的解决方案，因为它不会在服务整合过程中产生任何中断。

服务器整合和虚拟机迁移主要用于硬件虚拟化（见图 3-9），尽管其也可用于编程语言虚拟化。

存储虚拟化通常是执行环境虚拟化的补充。即使在这种情况下，拥有大型计算设备的供应商也能够利用其设备的巨大存储能力来提供虚拟存储服务，并可以轻松地分片。每个存储切片都可以动态地作为服务来提供，还可以确保主机设备的安全性，因为很容易确定每一分片服务的责任。

最后，云计算改变了桌面虚拟化的概念，这一概念最早在大型机时代提出。重新按需创建从基础设施到应用服务的整个计算栈的能力，使用户能够从瘦客户端经由互联网络来访问
92 托管在供应商设备中的虚拟计算机。

3.5 虚拟化的利与弊

现在，虚拟化技术已经非常流行且应用广泛，尤其是在云计算环境中，其主要原因是攻破了过去虚拟化技术缺乏有效性和可行性的技术壁垒。最直接的技术障碍是性能问题。如今，互联网的普及和计算技术的进步使得虚拟化能够按需提供 IT 基础设施和服务。尽管虚拟化技术再度流行，但是这种技术在发挥其优势的同时也存在缺点。

3.5.1 虚拟化技术的优点

执行的可控性和独立性是虚拟化的最重要的优点。在技术支持创建虚拟执行环境的情况下，利用这两个特点可以构建安全可控的计算环境。虚拟执行环境可以配置为沙箱，从而防止任何非法操作进入虚拟主机。此外，资源在不同客户机之间的分配被简化为由程序控制的虚拟主机来完成。这样能够按照服务质量的要求调度资源，这在服务器整合方案中非常重要。

可移植性是虚拟化的另一个优点，特别是对于执行虚拟化技术。虚拟机实例通常表示为可由物理机传送的一个或多个文件。此外，这些实例往往是自包含的，除了其使用的虚拟机管理器之外，没有其他依赖关系。可移植性和自包含性简化了管理。Java 程序“一次编译，多次运行”，只需要在主机上安装 Java 虚拟机。这同样适用于硬件级的虚拟化。事实上，无论在哪里，只要有笔记本电脑，就可以在虚拟机实例中建立执行环境。这个概念也推动了服务器整合方案的迁移技术的发展。

可移植性和自包含性也有助于减少维护成本，因为预期的主机数量比虚拟机实例的数量少。客户机程序在虚拟环境中执行，所以几乎不可能破坏底层硬件。此外，虚拟机管理器的数目要少于被管理的虚拟机实例的数量。

最后，通过虚拟化技术能够更有效地利用资源。多个系统可以安全地共存并共享底层主机的资源，而且不会相互干扰。这是服务器整合的首要条件，根据当前系统负载动态地调整
93 物理资源的数量，从而减少能耗且减轻对环境的影响。

3.5.2 虚拟化技术的缺点

虚拟化技术也有缺点。最明显的是由于虚拟化层协调资源而导致客户机系统性能下降。此外，由于虚拟化管理软件抽象层而引起主机没有被优化使用，使得主机利用率低或降低了用户服务质量。不明显但是更加危险的是隐含的安全问题，这大多是由于模拟不同的执行环境所产生的。

1. 性能降低

性能问题是使用虚拟化技术所需关注的主要问题之一。由于虚拟化在客户机和主机之间增加了抽象层，这将增加客户任务的操作延迟。

例如，在硬件虚拟化情况下，当模拟一个可以安装完整系统的裸机时，性能降低归咎于下列活动产生的开销：

- 维持虚拟处理器的状态。
- 支持特权指令（自陷和模拟特权指令）。
- 支持虚拟机分页。
- 控制台功能。

此外，当硬件虚拟化是通过在主机操作系统上安装或执行的程序实现时，性能降低的主要原因是虚拟机管理器同其他应用程序一起被执行和调度，从而共享主机的资源。

更高层的虚拟化技术也有类似的问题，如在编程语言级虚拟化情况下（Java、.NET 等）。二进制翻译和解释也降低应用程序的执行速度。此外，由于其执行被运行环境过滤，访问存储器和其他物理资源也会导致性能降低。

由于技术进步以及计算能力的提升，这些问题变得不再突出。例如，用于硬件虚拟化的特定技术，如半虚拟化技术，可以提高客户机程序的性能，无需修改便可将客户机上的大部分执行任务迁移到主机上。编程级的虚拟机，如 JVM 或 .NET，当性能要求比较高时，可以选择编译本地代码。

2. 低效和用户体验下降

虚拟化有时会导致主机的低效使用。特别是当某些主机的特定功能不能由抽象层展现，进而变得不可访问时。在硬件虚拟化环境中，设备驱动程序可能会出现这种情况，虚拟机有时仅仅提供只映射主机部分特性的默认图形卡。在编程级虚拟机环境中，一些底层的操作系统特性变得不可访问，除非使用特定的库。例如，在 Java 第一版本中，图形化编程的支持 94
是非常有限的，应用程序的界面和使用感觉非常差。用于设计用户界面的 Swing 新框架解决了这个问题，在软件开发工具包中集成了 OpenGL 库，加强了图形化功能。

3. 安全漏洞和威胁

虚拟化滋生了新的难以预料的恶意网络钓鱼（phishing）[⊖]，它能够以完全透明的方式模拟主机环境，使得恶意程序可以从客户机提取敏感信息。

在硬件虚拟化环境中，恶意程序可以在操作系统之前预安装，并作为一个微虚拟机管理器。这样该操作系统就可以被控制和操纵，并从中提取敏感信息给第三方。这类恶意软件包括 BluePill 和 SubVirt。BluePill 针对 AMD 处理器系列，将安装的操作系统的执行移到虚拟机中完成。微软与美国密歇根大学合作研发的 SubVirt 早期版本是原型系统。SubVirt 影响客户机操作系统，而当虚拟机重新启动时，它将获得主机的控制权。这种类型恶意软件的传播是因为原来的硬件和 CPU 产品并未考虑虚拟化。现有的指令集不能通过简单的改变或更新以适应虚拟化的需求。最近，英特尔和 AMD 相继分别推出了针对虚拟化的硬件支持 Intel VT 和 AMD Pacifica。

编程级的虚拟机也存在同样的问题：运行环境的改变可以获得敏感信息或监视客户应用

⊖ 网络钓鱼是用于获取用户敏感信息（如用户名和密码）的恶意行为，方法是重新创建一个在功能和外观上与管理这些信息的环境相同的环境。网络钓鱼通常发生在 Web 上，用户被重定向到与原有网站一样的恶意网站，目的是要获得进入原有网站（例如银行网站）的用户信息，从而获得用户的机密数据。

程序所使用的内存位置。这样，运行时环境的原始状态将被修改和替换，如果虚拟机管理程序内存在恶意软件或主机操作系统的安全漏洞被利用，将会经常发生安全问题。

3.6　技术实例

广泛的虚拟化技术特别适用于虚拟化计算环境。在本节中，我们将讨论该领域相关的技
95 术和方法。云计算的具体解决方案将在下一章中讨论。

3.6.1　Xen：半虚拟化

Xen 是开源的基于半虚拟化技术的虚拟化平台。由英国剑桥大学的研究小组开发，现在拥有支持 Xen 的强大开源环境。Citrix 还提供了 Xen 商业解决方案 XenSource。基于 Xen 的技术可用于桌面虚拟化以及服务器虚拟化，最近利用 Xen 云平台（XCP）还可提供云计算解决方案。这些解决方案的基础是作为 Xen 核心技术的虚拟机管理程序。最近 Xen 已经使用硬件辅助虚拟化技术支持完全虚拟化。

Xen 是最流行的半虚拟化实现技术，与完全虚拟化相比，它支持客户操作系统的高性能运行。需要执行特殊管理指令时，Xen 能减少性能损失。这是通过修改涉及该指令执行的 Xen 客户机操作系统来实现的。因此，它不是实现虚拟化的透明方案，尤其是对于 x86 系列商品机和服务器。

图 3-11 描述了 Xen 的结构及其在传统 x86 特权模型上的应用。基于 Xen 的系统由 Xen
96 虚拟机管理程序管理，在最高特权模式下运行，并控制客户机操作系统对底层硬件的访问。客户机操作系统都作为虚拟机实例在域内执行。此外，能访问主机并控制所有其他客户机操作系统的特定控制软件在域 0 执行。当虚拟机管理器已经完全启动时，首先加载域 0，配置超文本传输协议（HTTP）服务器，它可以接受虚拟机的创建、配置和终止请求。该组件包含在分布式虚拟机管理器的初期版本中，是提供基础设施即服务（IaaS）的云计算系统的重要组成部分。

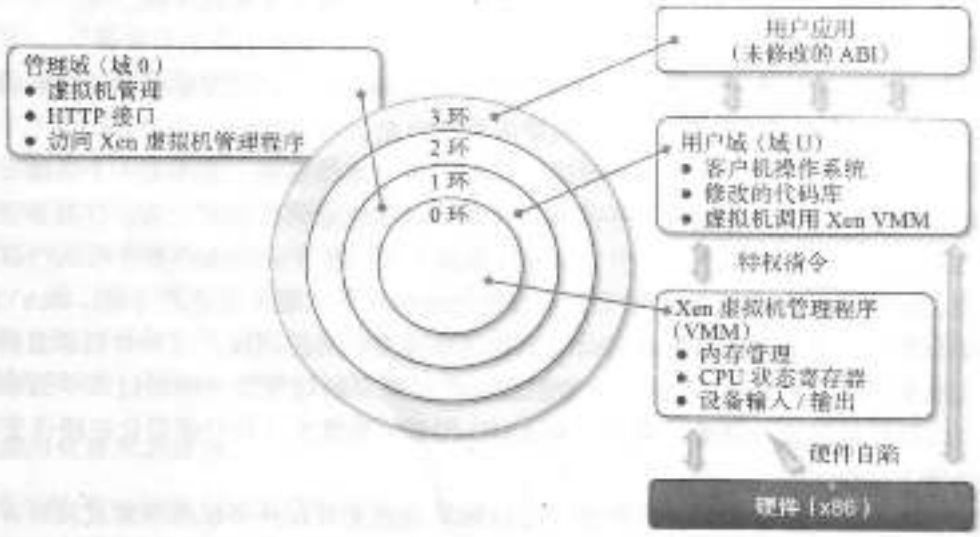


图 3-11　Xen 结构及客户机操作系统管理

x86 系统有四种不同的安全级别，称为环，其中 0 环级别最高，3 环级别最低。几乎所

有常见的操作系统（除 OS/2 外）都只支持两种安全级别：0 环运行内核代码，3 环是用户应用程序和非特权操作系统的代码。这样 Xen 可以通过执行 0 环和域 0 的虚拟机管理程序实施虚拟化，其他运行客户操作系统的域通常称为域 U，运行在 1 环，而用户应用程序在 3 环中运行。这样 Xen 可以保持 ABI 不变，从而从应用程序的角度可以很容易地转换为 Xen 虚拟化方案。在 x86 指令集的结构中，有些指令允许在 3 环执行的代码跳到 0 环（内核模式）执行。这样的操作在硬件层面完成，因此虚拟环境中会产生一个自陷或故障，从而阻止客户机操作系统的正常运行，因为其目前在 1 环上运行。这种情况一般由系统调用触发。为了避免这种情况，操作系统需要在执行过程中改变模式，而敏感系统调用需要重新由 Xen 虚拟机调用。调用后 Xen 的虚拟机管理程序能够控制和管理所有敏感指令的执行，并通过所提供的处理程序返回到客户机操作系统。

半虚拟化需要修改操作系统的代码库，因此并非所有的操作系统都可以用于 Xen 环境。更确切地说，当不能利用硬件辅助虚拟化技术，只允许在 1 环上运行虚拟机管理程序，而在 0 环上运行客户机操作系统时，会出现这种问题。因此，Xen 对于传统硬件和操作系统表现出一定的局限性，而这些操作系统不能被安全地转到 1 环上运行，因为其系统代码库不可访问。同时，底层硬件不支持在比 0 环更高特权的模式下运行虚拟机管理程序。开源操作系统（如 Linux）很容易修改，因为其代码是公开的，Xen 完全支持其虚拟化。Xen 一般不支持 Windows 系列组件，除非可以使用硬件辅助虚拟化。可以看出，这个问题现在已经不再突出了，因为新版本操作系统支持虚拟化，而且新的 x86 系列硬件也支持虚拟化。

3.6.2 VMware：完全虚拟化

VMware 技术基于完全虚拟化的概念，底层硬件被复制并提供给客户机操作系统，客户机操作系统的运行与抽象层无关，也不需要修改。VMware 可以利用Ⅱ型虚拟机管理程序实 97
现桌面环境的完全虚拟化，或者在服务器环境利用Ⅰ型虚拟机管理程序实现完全虚拟化。这两种情况都是通过直接执行（对于非敏感指令）和二进制翻译（对于敏感指令）的方式实现完全虚拟化，因而也支持 x86 系统的虚拟化。

除了这两种核心解决方案，VMware 还提供其他工具和软件以简化虚拟化技术的应用。在桌面环境中，集成虚拟客户机与主机工具；在服务器环境中，建立和管理虚拟计算基础设施的解决方案。

1. 完全虚拟化和二进制翻译

VMware 支持 x86 系统虚拟化，在运行虚拟机管理程序时无需做任何修改。随着 2006 年出现的新一代硬件架构和硬件辅助虚拟化技术（Intel VT-x 和 AMD V），在硬件的支持下全虚拟化得以实现。但在此之前，动态二进制翻译是使得 x86 客户机操作系统无需修改就可以在虚拟环境中运行的唯一解决方案。

如之前所讨论的，x86 系统设计不满足虚拟化第一定理，因为它的敏感指令集不是特权指令的子集。当该指令不在 0 环上执行时，将产生不同的结果，这是客户机操作系统运行于 1 环的常见虚拟化情况。通常系统会产生自陷，其管理方式与 x86 系统虚拟化不同。在动态二进制翻译时，自陷将错误指令翻译成实现同样目的的等效指令集，并且不会产生异常。此外，为了提高性能，这个等效指令集会被缓存，这样出现相同的指令时就不再需要转换。图 3-12 描述了该过程。

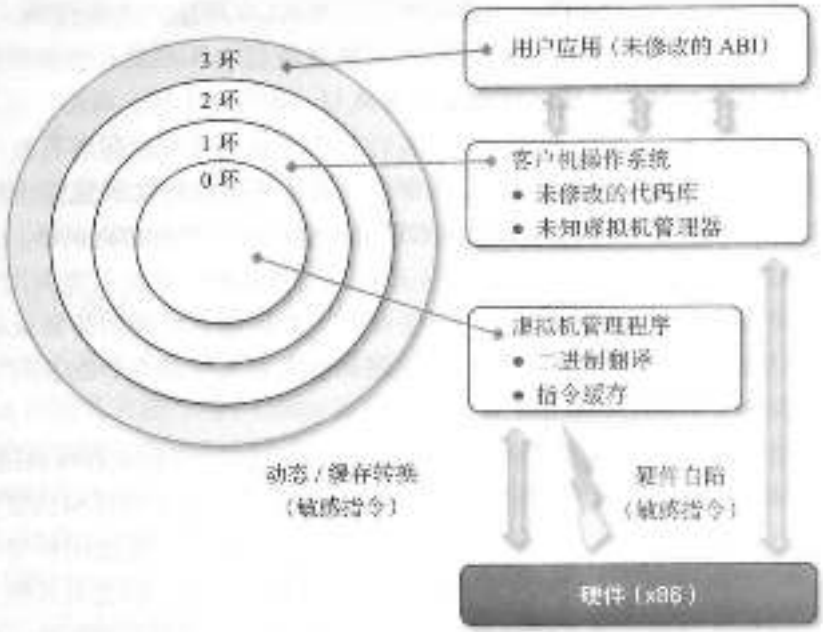


图 3-12 全虚拟化参考模型

这种方法既有优点也有缺点。最大的优点是客户机无需修改就可以在虚拟化环境中运行，这对于非开源代码的操作系统是非常关键的。Windows 系列操作系统便是一个例子。二进制翻译是使全虚拟化更具有可移植性的解决方案。另一方面，运行时的指令翻译相对于其他方法（半虚拟化或硬件辅助虚拟化）产生了额外开销。即使存在这样的缺点，二进制翻译还是应用在部分指令集中，而其他指令集则直接在底层硬件上执行。这在某种程度上降低了二进制翻译产生的性能问题。

CPU 虚拟化只是完全虚拟化硬件环境的一部分。VMware 通过虚拟内存和虚拟输入/输出设备实现完全虚拟化。存储虚拟化是虚拟化技术的另一挑战，如果没有相应的硬件支持将会严重影响性能。主要原因是内存管理单元（MMU）需要被模拟为虚拟硬件。尤其是在托管虚拟机管理程序（II 类）的情况下，其虚拟 MMU 和主机操作系统 MMU 依次访问物理内存页，对性能产生了较大影响。为了避免指令重复嵌套翻译，虚拟 MMU 的转换后援缓冲器（TLB）直接映射到物理页，这样只有 TLB 中不存在待转换指令时性能才会下降。最后，VMware 还支持 I/O 设备的全虚拟化，例如网络控制器和其他外围设备，如键盘、鼠标、磁盘和通用串行总线（USB）控制器。

98

2. 虚拟化解决方案

VMware 是虚拟化技术的先驱，提供了从桌面计算虚拟化到企业计算虚拟化，以及基础设施虚拟化的全面解决方案。

（1）终端用户（桌面）虚拟化

VMware 支持用户计算机的操作系统环境和应用程序的虚拟化。操作系统虚拟化是最流行的，并能在独立于主机操作系统的环境中安装各种操作系统和应用程序。VMware 系列软件中，VMware Workstation 适用于 Windows 操作系统，VMware Fusion 适用于 Mac OS X 环境，这些软件都安装在主机操作系统中以创建虚拟机并管理其执行。除了能够建立独立的计算环境外，这两款产品都允许客户机操作系统利用主机资源（USB 设备、文件夹共享、集

成主机操作系统的图形用户界面（GUI）。这些系统的体系结构如图 3-13 所示。

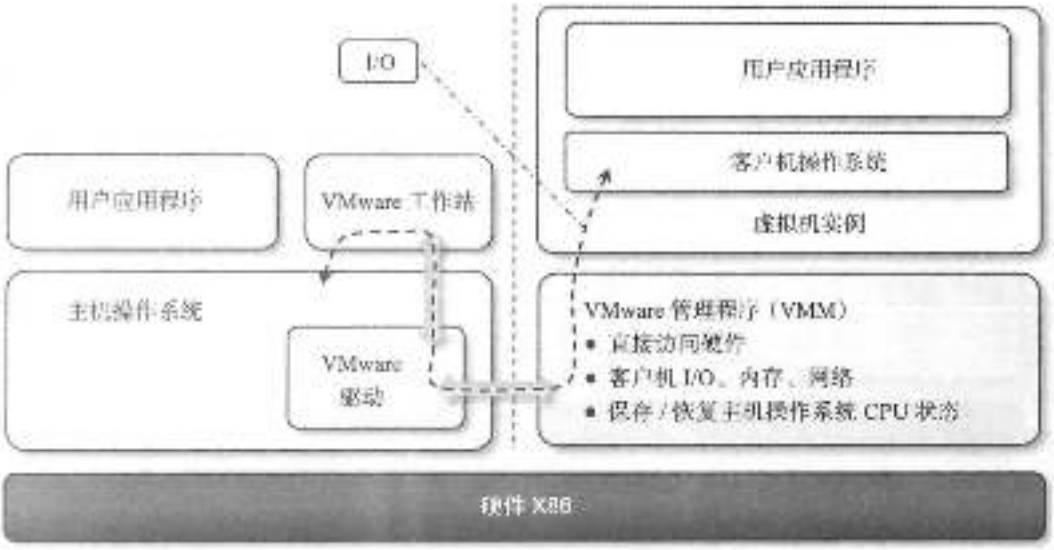


图 3-13 VMware 工作站结构

虚拟化环境由安装在客户机操作系统的应用程序创建，为这些操作系统提供底层硬件全虚拟化模式。这是通过在主机操作系统中安装特定的驱动程序实现的，该主机主要提供以下两项服务： 99

- 配置在特权模式下运行的虚拟机管理器。
- 为 VMware 应用程序提供处理 I/O 请求的方法，通过系统调用将此 I/O 请求传送到主机操作系统。

采用这种架构（也称为托管虚拟机架构）不仅可以将虚拟机实例独立于单一应用存储空间内，也能提供合理的性能，因为 VMware 应用程序只需要指令，如 I/O 设备只用二进制翻译。可直接执行的指令由虚拟机管理器管理，虚拟机管理器和主机操作系统交替控制 CPU 和 MMU。虚拟机镜像文件保存在主机文件系统，VMware Workstation 和 VMware Fusion 都能创建新的镜像、暂停其执行、创建快照，并可以复原操作回到虚拟机之前的状态。

其他终端用户计算环境的虚拟化方案包括 VMware Player、VMware ACE 和 VMware ThinApp。VMware Player 是 VMware Workstation 的简化版本，支持在 Windows 或 Linux 操作环境创建和执行虚拟机。VMware ACE 类似于 VMware Workstation，为部署安全的用户计算虚拟化环境建立虚拟机策略。VMware ThinApp 是应用级虚拟化解决方案，为应用程序提供一个独立的环境，以避免因版本和不兼容而产生的冲突。它可以检测所有由特定的应用程序安装产生的操作环境的变化，并将其与应用程序一起以二进制分组形式在 VMware ThinApp 上运行。 100

（2）服务器虚拟化

VMware 提供了不同的服务器虚拟化解决方案。最初支持服务器虚拟化的是 VMware GSX Server，它模拟了用户计算机应用功能，并增加了远程管理和脚本功能。VMware GSX Server 的架构如图 3-14 所示。

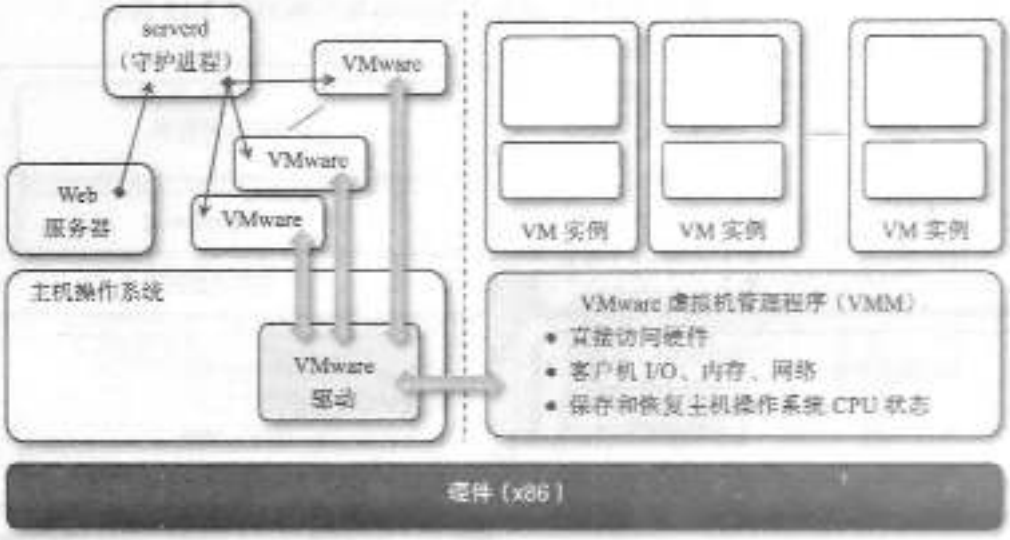


图 3-14　VMware GSX 服务器架构

该架构用于 Web 服务器虚拟化。名为 serverd 的守护进程负责控制和管理 VMware 应用程序进程。这些应用由安装在主机操作系统的 VMware 驱动连接到虚拟机实例。虚拟机实例由 VMM 管理。用户通过 Web 服务器向虚拟机发出请求，用 serverd 通过 VMM 对虚拟机进行管理和配置。

VMware ESX Server 及其增强版本 VMware ESXi Server 都是基于虚拟机管理程序的方法。既可以安装在裸机服务器上也可以为虚拟机管理提供服务。这两种方案提供相同的服务，但它们的内部结构不同，尤其是虚拟机管理程序的内核结构不同。VMware ESX 嵌入了 Linux 操作系统的修改版，可以通过服务控制台访问虚拟机管理程序。VMware ESXi 实现了简单的操作系统层，用远程管理接口和服务替代服务控制台，从而大大降低了虚拟机管理程序代码的大小和内存占用。

VMware ESXi 的架构如图 3-15 所示，其核心是 VMkernel，是简单的可移植操作系统接口（POSIX）的兼容操作系统，提供了进程和线程管理、文件系统、I/O 栈和资源调度等
101 最基本功能。通过 User world API 的 API 访问内核。系统代理利用这些 API 提供对虚拟机的管理。CIM Broker 负责对 ESXi 服务器进行远程管理。系统代理作为客户 VMkernel 网关，使用通用信息模型（CIM）[⊖]协议。ESXi 安装也可以通过直接用户端接口（DCUI）在本地管理，DCUI 提供了类似于 BIOS 的界面。

（3）基础设施虚拟化和云计算解决方案

VMware 提供了涵盖云计算全部协议栈的一组产品，从云计算环境中的基础设施管理到软件即服务（SaaS）解决方案。不同解决方案及其相互关系如图 3-16 所示。

⊖ 通用信息模型（CIM）是用于定义系统、应用程序和服务的管理信息的分布式管理任务标准。详见 http://dmtf.org/standards/cim。

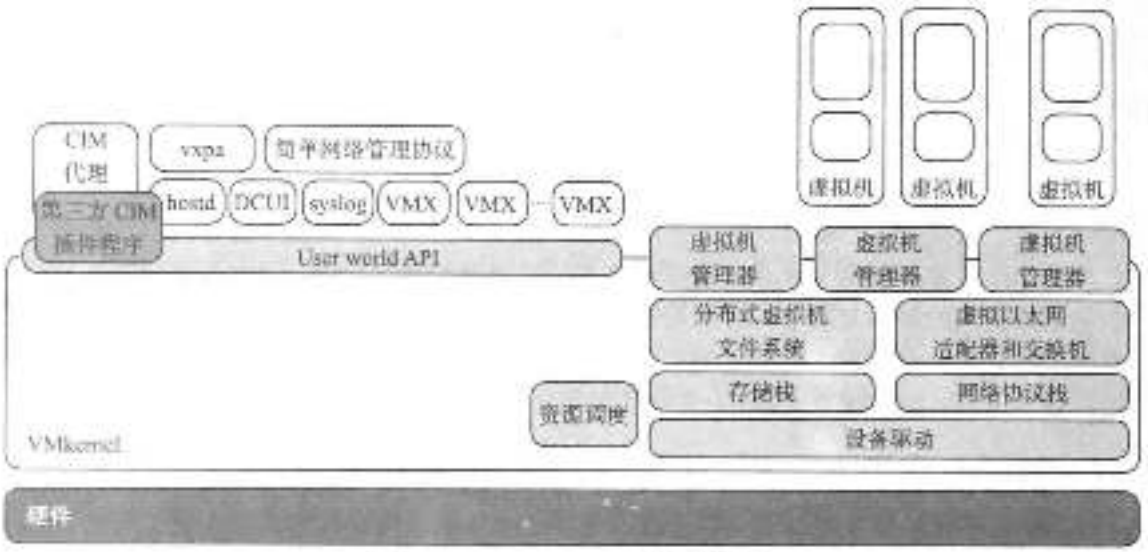


图 3-15 VMware ESXi 服务器架构

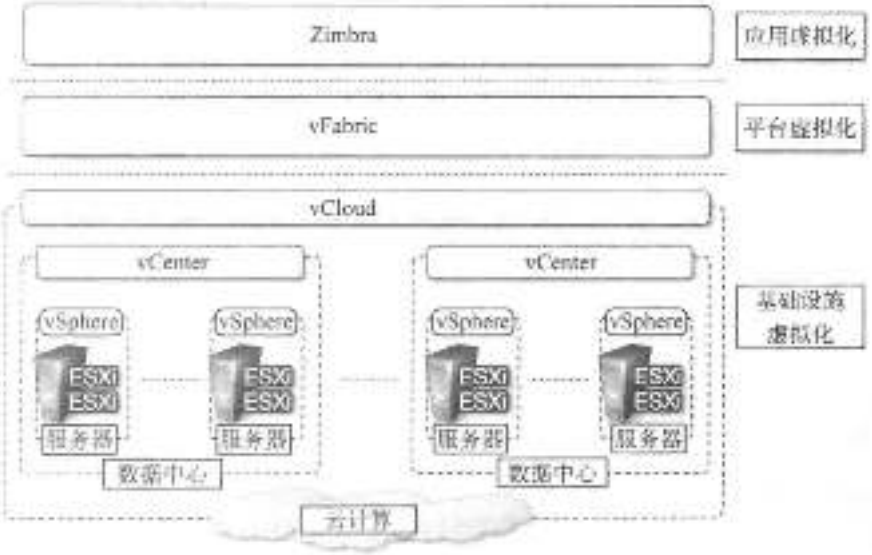


图 3-16 VMware 云计算解决方案层次结构

ESX 和 ESXi 是虚拟基础设施管理组件：虚拟化服务器池连接在一起并由 VMware vSphere 远程管理。作为虚拟化平台，VMware 提供了包括虚拟计算服务的一组基础服务：虚拟文件系统、虚拟存储和虚拟网络，它们构成基础设施的核心；vSphere 提供了应用服务，如虚拟机迁移、存储迁移、数据恢复和安全性。VMware vCenter 负责管理基础设施，在数据中心环境中安装 vSphere 并对其进行集中管理。VMware vCloud 将虚拟化数据中心转变为云计算基础设施即服务（IaaS），允许服务供应商为用户按需提供即用即付的虚拟计算环境。用户通过 Web 门户访问 vCloud 服务，而用户可以通过模板选择或在虚拟实例之间设置虚拟网络来自行建立虚拟机。

102 / 103

VMware 还提供了云计算中应用程序开发的解决方案 VMware vFabric，它是一套组件，用于在虚拟化基础设施上开发可扩展的 Web 应用程序。vFabric 包括应用程序监控、可扩展的数据管理、可扩展的执行和 Java Web 应用组件。

最后，VMware 云计算方案的最上层是 Zimbra，用于办公自动化、消息传递及组件协同，完全托管在云计算环境中，在任何地方都能访问。Zimbra 是 SaaS 解决方案，在单一软件平台集成各种功能，包括电子邮件和协同管理。

3. 结论

从 x86 硬件完全虚拟化方案开始，随着 VMware 的不断发展，目前可以提供一个完整虚拟化方案，包括硬件、基础设施、应用程序和服务的虚拟化，涵盖了云计算市场的多数产品。尽管 x86 全虚拟化是 VMware 的核心技术，但随着技术的发展，半虚拟化功能已集成到一些供应商提供的解决方案中，特别是在硬件辅助虚拟化出现之后。例如，一些设备仿真的实现和 VMware Tool 组件，加强了客户机和主机操作环境的集成。此外，VMware 已经开发了与厂商无关的虚拟机接口（VMI）并对其规范化，这是通用的主机无关的半虚拟化方法。

3.6.3　微软 Hyper-V

Hyper-V 是微软开发的用于服务器虚拟化的基础设施虚拟化方案。正如其名，它采用基于虚拟机管理程序（hypervisor）的方法实现硬件虚拟化，利用多种技术来支持各种客户机操作系统。Hyper-V 目前作为 Windows Server 2008 R2 的一个组件，将虚拟机管理程序作为服务器的一个角色。

1. 架构

Hyper-V 通过分区的方式支持多个客户机操作系统的并发执行。分区是一个完全隔离的操作系统的安装和运行环境。

104

Hyper-V 的架构如图 3-17 所示。尽管其直接作为主机操作系统的一个组件安装，但是 Hyper-V 控制硬件，并且将主机操作系统配置成具有特殊权限的虚拟机实例，称为父分区。

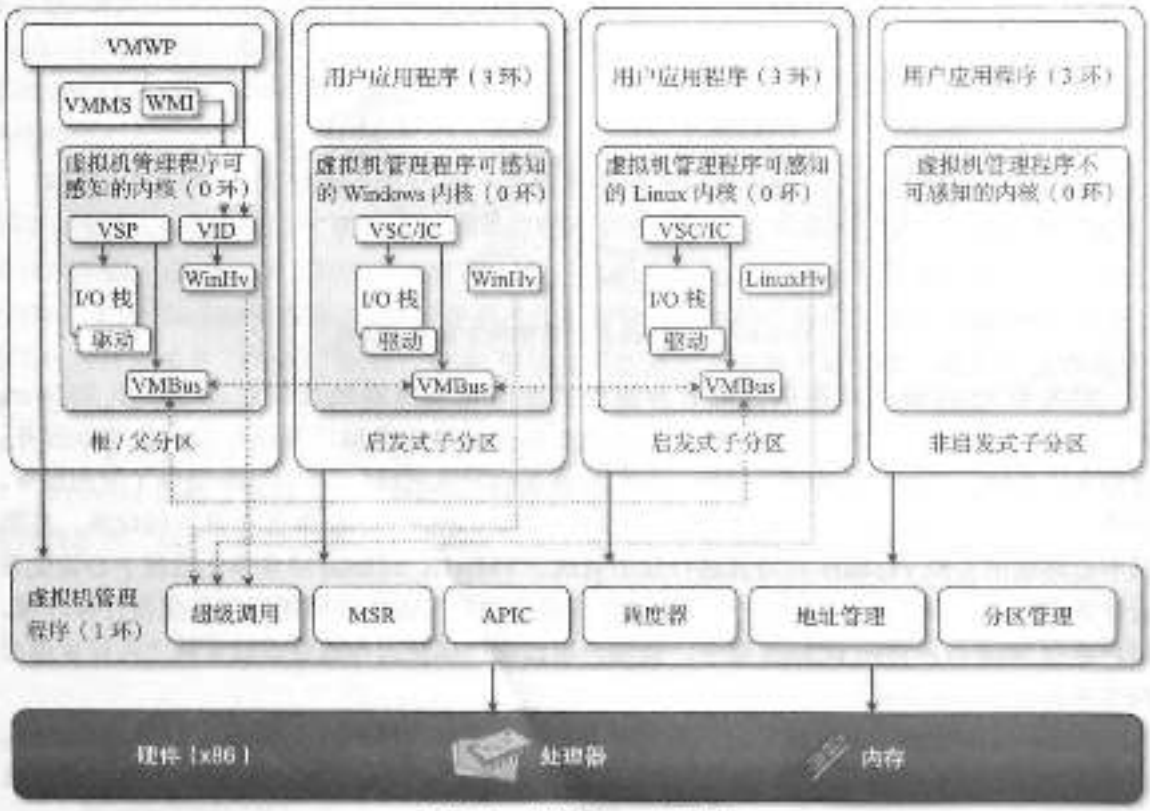


图 3-17　微软 Hyper-V 架构

父分区（也称为根分区）是唯一可以直接访问底层硬件的分区。它运行在虚拟化层，装有配置客户机操作系统的驱动程序，并通过虚拟机管理程序创建子分区。子分区安装客户机操作系统且不能访问底层的硬件，可由父分区和虚拟机管理程序控制子分区之间的交互。

（1）虚拟机管理程序

虚拟机管理程序是直接管理底层硬件（处理器和内存）的组件。包括以下部分：

- 超级调用接口。这是执行敏感指令的所有分区的入口点，用 Xen 半虚拟化方法来实现。分区操作系统中驱动程序使用该接口并采用标准的 Windows 调用方式，与虚拟机管理程序联系。父分区也使用该接口来创建子分区。
- 内存服务例程（MSR）。这是控制内存和从分区访问内存的功能集。利用硬件辅助虚拟化，虚拟机管理程序通过翻译虚拟内存地址、使用输入/输出内存管理单元（I/O MMU 或 IOMMU）快速从分区访问设备。
- 高级可编程中断控制器（APIC）。该组件是指中断控制器，管理底层硬件发生的某些引起中断的事件（计时器到时、I/O 就绪、异常和自陷）。每个虚拟处理器都配置了合成中断控制器（SynIC），是本地 APIC 的扩展。当 SynIC 发生物理中断时，虚拟机管理程序负责分派。
- 调度器。该组件对虚拟处理器进行调度，使其在具有高可靠性的物理处理器上运行。调度由父分区设置的策略控制完成。
- 地址管理。该组件用于管理分配给每个客户机操作系统的虚拟网络地址。
- 分区管理。这个组件负责分区的创建、删除、显示和配置。通过前面讨论过的超级调度的 API 接口来获得该服务。

虚拟机管理程序在 1 环上运行，因此需要相应的硬件技术支持。通过执行高特权模式，虚拟机管理程序可以支持 x86 原有操作系统。新一代操作系统可以利用 Hyper-V 的新体系结构，特别适用于由子分区完成的 I/O 操作。

（2）启发式 I/O 和集成设备

启发式 I/O 提供了一种执行 I/O 操作的优化方法，允许客户操作系统利用一个分区通信通道，而不是通过由虚拟机管理程序提供的硬件模拟技术栈。这种方式仅适用于虚拟机管理程序可感知的客户机操作系统。启发式 I/O 设备使用了 VMBus，分区通信通道用于分区（子分区和父分区）之间交换数据，而且多用于实现客户机操作系统的虚拟设备驱动。

105
|
106

启发式 I/O 的架构如图 3-17 所示。包括三个基本组件：VMBus、虚拟服务供应商（VSP）和虚拟服务客户端（VSC）。VMBus 是分区之间的通信通道，它定义了通信协议。VSP 是内核级驱动程序，部署在父分区，可访问相应的硬件设备。子分区中的客户机操作系统的虚拟设备驱动与 VSC 进行交互。支持 Hyper-V 的操作系统利用该通信通道执行 I/O 存储、网络通信、图形和输入子系统。客户机操作系统之间的虚拟网络可以增强子系统的子分区间通信 I/O 的性能。原有操作系统不是基于感知的虚拟机管理程序，但仍然可以通过 Hyper-V 依靠设备驱动模拟来运行，虚拟机管理程序对设备驱动模拟进行管理，只是效率较低。

（3）父分区

父分区执行主机操作系统，并实现客户机操作系统中的虚拟机管理程序活动的虚拟化。此分区是 Windows Server 2008 R2 的一个实例，负责管理子分区可使用的虚拟化栈。这是唯一一个直接访问设备驱动的分区，并可以通过 VSP 的子分区协调访问。

父分区用于管理子分区的创建、执行和删除，通过虚拟化设备驱动（VID）来实现其功

能，VID 控制对虚拟机管理程序的访问，并管理虚拟处理器和内存。对于创建的每个子分区，虚拟机工作进程（VMWP）被定义为父分区的实例，通过 VID 管理子分区与虚拟机管理程序的交互。可通过 WMI[⊖]远程获取虚拟机管理服务，WMI 支持远程主机访问 VID。

（4）子分区

子分区用于执行客户机操作系统。子分区是客户机的安全和可控的独立执行环境。根据客户机操作系统是否支持 Hyper-V，子分区可分为两种类型。分别称为启发式分区和非启发式分区。启发式分区受益于启发式 I/O 设备，非启发式分区则由虚拟机管理程序充分利用硬件模拟来执行。

2. 云计算和基础设施管理

Hyper-V 是微软基础设施虚拟化的基本模块。其他组件则用于创建功能全面的服务器虚拟化平台。

为了提高虚拟化环境的性能，微软发布了 Windows Server 2008 的新版本——Windows
107 Server Core。这是一个特殊版本的操作系统，它减少了功能集并使用了较少的脚本，尤其是去掉了服务器环境中不需要的功能，如 GUI 组件和其他占内存较大组件，如 .NET 框架及在该框架上开发的应用程序（例如 PowerShell）。这样的设计既有优点也有缺点。有利的一面是减少了维护（例如更少的软件补丁），减少了被攻击的可能性，减少了管理并节约了磁盘空间。缺点是嵌入式功能变弱了。尽管如此，仍可以通过功能齐全的 Windows，以远程管理的方式来获得“缺少的功能”。例如，管理员可以使用 PowerShell 中的 WMI 来远程管理 Windows Server Core 的安装。

支持虚拟机管理的另一个组件是系统中央虚拟机管理器（SCVMM）2008，它是 Microsoft System Center 的一个组件，从 IT 生存期管理方面实现虚拟基础设施的管理功能。从本质上讲，SCVMM 完成了 Hyper-V 的基本管理功能，包括：

- 创建和管理虚拟实例的门户。
- 虚拟机到虚拟机（V2V）和物理机到虚拟机（P2V）的转换。
- 授权管理。
- 库和深度 PowerShell 集成。
- 虚拟机在管理环境中的智能配置。
- 主机容量管理。

SCVMM 与其他虚拟化平台兼容，如 VMware 的 vSphere（ESX 服务器），但是主要优势源于 Hyper-V 的虚拟基础设施管理。

3. 结论

与 Xen 和 VMware 相比，Hyper-V 是一个综合的解决方案，利用了半虚拟化技术和全硬件虚拟化技术。

Hyper-V 虚拟机管理程序的架构基于半虚拟化技术。客户机操作系统通过超级调用可以获取虚拟机管理程序的服务。同样半虚拟化内核可以利用 VMBus 进行快速的 I/O 操作。此外，其分区概念类似于 Xen 的域：父分区对应域 0，而子分区对应域 U。唯一的区别是，Xen 虚拟机管理程序安装在裸硬件上，并过滤掉所有对底层硬件的访问，而 Hyper-V 是安装

⊖ WMI 指 Windows 管理工具（Windows Management Instrumentation），是 Windows 环境访问底层硬件的规范。该规范面向服务供应商，对用户授权来访问硬件的特定子系统。

在操作系统上的一个角色，它与分区的交互方式与 VMware 非常相似。

Hyper-V 所采用的方法既有优点也有缺点。优点是它是一个灵活的虚拟化平台，可以支持多种客户机操作系统。缺点是对硬件和软件的要求。Hyper-V 只与 Windows Server 2008 和 x64 系统的 Windows Server 平台兼容。此外，它需要 64 位处理器以支持硬件辅助虚拟化和数据执行保护。最后，如上所述，Hyper-V 是安装在操作系统上的一个角色，而 vSphere 和 Xen 可以安装在裸硬件上。 108

本章小结

虚拟化术语包括了各种技术和概念。所有虚拟化形式的共同点是通过使用某种模拟或抽象层来提供特定的虚拟环境，无论是运行环境、存储能力、网络连接或远程桌面。这些概念为创建云计算的基础设施和服务发挥了重要作用。云计算环境中的硬件、IT 基础设施、应用和服务都通过互联网或是通用的网络连接被按需交付。

习题

1. 什么是虚拟化？其优点是什么？
2. 虚拟化环境的特点是什么？
3. 描述不同层次的虚拟化分类。
4. 描述执行虚拟化的机器参考模型。
5. 什么是硬件虚拟化技术？
6. 列出并详述不同类型的虚拟化。
7. 云计算环境中虚拟化的优势是什么？
8. 虚拟化的缺点是什么？
9. 什么是 Xen？描述其虚拟化原理。
10. 描述完全虚拟化的参考模型。
11. 论述 Hyper-V 的架构及其在云计算中的应用。

109 - 110

第 4 章
Mastering Cloud Computing: Foundations and Applications Programming

云计算架构

云计算这个词包含许多不同的含义，最近它非常流行，常被公众误解为现有技术和想法的翻新。是什么让 IT 经营者和研究者对云计算如此感兴趣？它是怎样在分布式计算领域内创新的？这一章将围绕这些问题展开，归纳这一现象的特点，并提供一个参考模型作为讨论云计算技术的基础。

4.1 简介

面向效用的数据中心是云计算的首例成果，并且作为基础设施服务平台来实现和提供服务。任何云服务，不管是虚拟硬件、开发平台还是应用软件，都依赖于服务供应商拥有的或从第三方租用的分布式基础设施。按照之前的定义，云计算可以使用一个数据中心、若干集群或者是由 PC、工作站和服务器组成的异构分布式系统来实现。通常由一个或多个数据中心构建云计算。大多数情况下，硬件资源通过虚拟化实现任务的分离和基础设施的最有效利用。对于终端的具体服务，在虚拟基础设施的顶层又分为不同层次：虚拟机管理器、开放平台或者是具体的应用中间件。

前几章已经提及，云计算范式的出现是各种现有的模型、技术和概念共同集成的结果，它改变了我们交付和使用 IT 服务的方式。其广义的定义如下：

> 云计算是一种面向效用的以互联网为中心的按需交付 IT 服务的方法。它所提供的服务包含了整个计算栈的各个层次：从底层硬件基础设施集成为一组虚拟机，到顶层软件服务，如开发平台和分布式应用。

这个定义概括了云计算最重要和最基本的特征。现在我们讨论一个参考模型，进而对云计算技术、应用及服务进行分类。

4.2 云计算参考模型

云计算支持任何 IT 服务作为公共基础设施服务通过网络被使用和交付。这种服务包括几个不同方面：基础设施、开发平台、应用软件和服务。

4.2.1 架构

可以把云计算所有具体实现组织成分层次的堆栈结构（见图 4-1），从硬件应用到软件系统。云计算资源被用于提供服务的“计算马力”。通常，这一层采用数据中心实现，数据中心有成千上万个聚集在一起的节点。云基础设施本质上是异构的，由于包括各种不同的资源，比如集群甚至互联的 PC，这些都可以作为云基础设施。此外，数据库系统和其他存储服务也可以是云基础设施的一部分。

物理基础设施由核心中间件管理，其目的是为应用软件提供一个适当的运行环境以实现资源的最优化利用。在云计算层次结构的底部，采用虚拟化技术来保证运行环境的可定制、



图 4-1 云计算架构

应用程序的独立性、沙箱作业和服务质量。硬件虚拟化在这一层应用最为普遍。虚拟机管理程序管理资源池并把分布式基础设施展现为虚拟机集合。采用虚拟机技术能够更精细地划分硬件资源，例如 CPU、存储器、虚拟化设备，从而满足用户和应用程序的要求。该方案通 112
常与存储和网络虚拟化策略一起应用。支持对基础设施的完全虚拟化和控制。根据提供给最终用户的具体服务，也可以使用其他虚拟化技术，例如，编程级虚拟化有助于创建可移植运行环境，以运行和控制应用程序。这种情况通常意味着在云计算中托管的应用程序可以用特定的技术或编程语言（如 Java、NET 或 Python）进行开发，此时用户不必从裸机构建系统。基础设施管理是核心中间件的关键功能，支持很多功能，如服务质量协商、准入控制、执行管理和监控、计账、计费。

云托管平台和资源结合在一起通常被归类为基础设施即服务（IaaS）方案。可以把 IaaS 的不同应用实例分为两大类：①其中一些同时提供管理层和物理基础设施；②其他的只提供管理层（IaaS（M））。在这第二种情况下，管理层通常与其他提供物理基础设施的 IaaS 方案结合在一起。

IaaS 解决方案适用于设计系统的基础设施，但对于构建应用程序却只能提供有限的服务。应用程序服务由云计算的编程环境和编程工具提供，这样就形成了一个新的层——为用户提供应用程序的开发平台。工具包括基于 Web 界面、命令行、并行和分布式编程框架。在这种情况下，用户通过用户级中间件的 API 开发云计算的应用程序。由于这个原因，这种方法也称为平台即服务（PaaS），因为提供给用户的服务是一个开发平台而不是基础设施。PaaS 解决方案一般也包括基础设施，这个基础设施也是提供给用户的服务的一部分。在纯 PaaS 的情况下，只提供了用户级中间件，并且必须辅以一个虚拟或物理基础设施。

在图 4-1 中所示的参考模型的顶层包含为应用层提供的服务，通常称为软件即服务（SaaS）。在大多数情况下，这些基于 Web 的应用程序都依赖于云计算将服务提供给用户终端。在 IaaS 和 PaaS 解决方案的支持下，云计算允许独立软件供应商通过互联网提供应用服务。属于这一层的其他应用都充分利用互联网，其核心功能以云计算为依托支持大量用户使

用。游戏门户网站就是一个例子，通常还有社交网站。

作为一种设想，云计算提供的任何服务应该能够自适应地改变和自动化执行，特别是其可用性和性能要高。作为参考模型，希望有一个按需的弹性可扩展的自适应性管理层。SaaS可以自动地支持这样的操作，而 PaaS 和 IaaS 一般作为用户 API 的一部分提供这样的功能。

参考模型如图 4-1 所示，还引入了一切即服务（XaaS）的概念。这是云计算中最重要的元素之一：来自不同供应商的云服务可以结合起来，提供一个完全集成的涵盖系统所有计算层次的解决方案。IaaS 供应商可以提供支持虚拟机的裸机，PaaS 部署在虚拟机上。如果不必包含 PaaS 层，则可以直接定制运行应用程序所需软件的虚拟基础设施。虚拟 Web 是由
113 Web 服务器、数据库服务器和负载均衡服务器组成的分布式系统，在该系统安装运行 Web 应用程序的软件。云计算是降低 IT 企业初始投资成本的非常有吸引力的选择，使企业能够迅速实现商业化，并根据自己的收益增加基础设施。

表 4-1 总结了云计算解决方案三大类别的特点。在下面的章节中，我们将参考实际实施简要地讨论这些特点。

表 4-1　云计算服务分类

类别	特性	产品类型	供应商和产品
SaaS	随时随地为客户提供应用服务	Web 应用和服务（Web 2.0）	SalesForce.com（CRM）Clarizen.com（项目管理） 谷歌 App
PaaS	向用户提供一个平台以开发托管在云中的应用	编程 API 和框架部署系统	谷歌 AppEngine 微软 Azure Manjrasoft Aneka Data Synapse
IaaS/HaaS	向用户提供虚拟化硬件和存储，可以在其上建立基础设施	虚拟机管理 基础设施 存储管理 网络管理	亚马逊 EC2 和 S3 GoGrid Nirvanix

4.2.2　基础设施即服务和硬件即服务

基础设施即服务和硬件即服务（IaaS/HaaS）方案是最流行的云计算产品。IaaS 能按需提供定制的基础设施，服务范围从单一服务器到整个基础设施，包括网络设备、负载均衡、数据库和 Web 服务器。

用于交付和实施这些方案的主要技术是硬件虚拟化：一个或多个虚拟机互相连接构成一个分布式系统，再在此系统上安装和部署应用程序。根据虚拟硬件的特性（如内存、处理器数量和磁盘存储）对虚拟机进行部署和定价。IaaS/Haas 方案中硬件虚拟化的优点包括：工作负载分区、应用程序隔离、沙箱和硬件协同。从服务供应商的角度来看，IaaS/Haas 可以更好地利用 IT 基础设施，并提供了一个更安全的环境来执行第三方应用程序。从客户的角度看，它降低了管理和维护成本，以及购买硬件的成本。与此同时，用户可以充分利用虚拟化提供的完全定制的优势，在云中部署其基础设施。在大多数情况下，虚拟机只配备了选定
114 的操作系统，此系统可按照所需要的软件包和应用程序进行配置。其他方案提供了预安装系统镜像，包含大多数常用的软件：Web 服务器、数据库服务器、LAMP[⊖]。除了基本的虚拟

⊖ LAMP 是 Linux Apache MySql and PHP 首字母缩写，表示服务器配置运行 Linux 操作系统，Apache 作为 Web 服务器，MySQL 作为数据库服务器，PHP（超文本预处理）作为开发 Web 应用程序的脚本技术。LAMP 是常见的快速开发 Web 应用程序的解决方案。

机管理功能。这些方案还提供了附加服务，一般包括以下内容：基于 SLA 的资源分配，工作负载管理，通过先进的 Web 界面支持基础设施设计，以及集成第三方 IaaS 方案的能力。

实施 IaaS 方案的参考模型如图 4-2 所示。模型可划分为三个主要层次：物理基础设施、基础设施管理软件和用户界面。在顶层的用户界面可以访问基础设施管理软件提供的服务。 115
这种界面一般基于 Web 2.0 技术：Web 服务、RESTful APIs 和 mash-ups。这些技术允许任何应用程序或终端用户获取底层基础设施提供的服务。Web 2.0 应用程序支持开发全功能的浏览器或 Web 页面方式的管理控制台。Web 服务和 RESTful API 允许程序与服务交互而无需人工干预，这样就可以实现软件系统一体化。IaaS 方案的核心功能是在基础设施管理软件层实现的。尤其是虚拟机的管理是该层执行的最重要功能。调度程序作为核心角色负责分配虚拟机实例的执行任务。调度程序和其他组件交互完成各种任务：

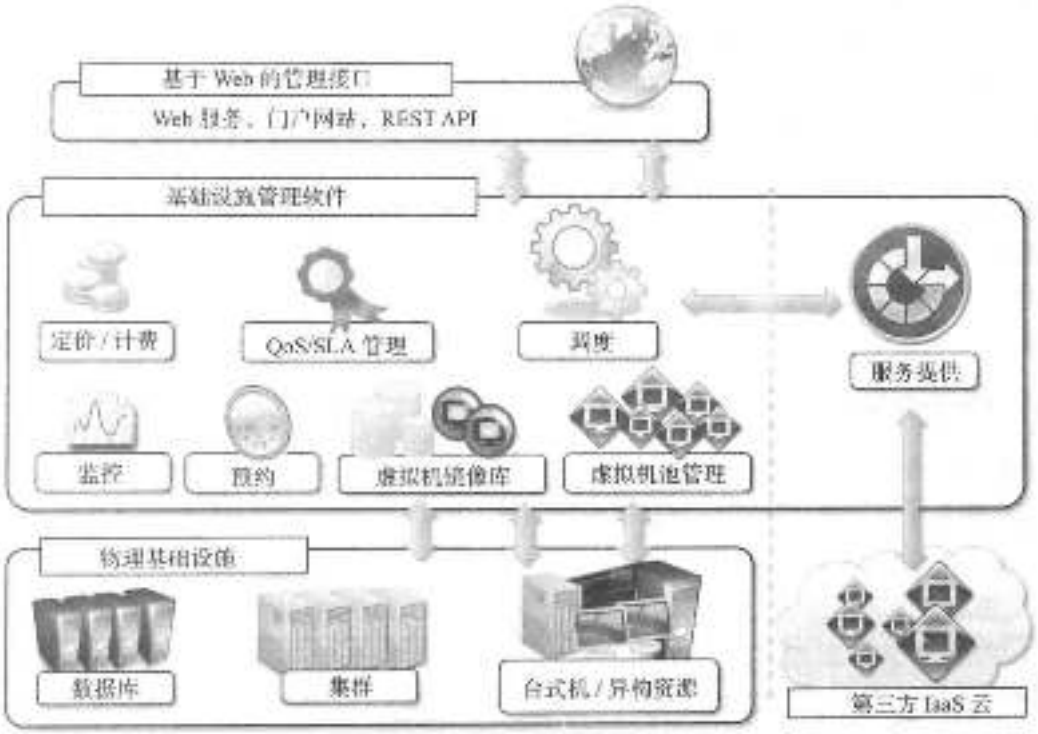


图 4-2 实施 IaaS 的参考模型

- 定价和计费组件负责记录每个虚拟机实例的执行成本，并维护用于向用户收取费用的数据。
- 监控组件跟踪每个虚拟机实例的执行情况，并维护所有用于报告和分析系统性能的数据。
- 预约组件存储所有已执行或未来将要执行的虚拟机实例的信息。
- 如果提供了支持基于 QoS 的执行，QoS/SLA 管理组件将维护所有用户要求的服务等级协议（SLA）库，与监控组件一起确保给定的虚拟机实例能达到期望的服务质量。
- 虚拟机资源库组件提供虚拟机镜像的目录，用户可以使用它来创建虚拟实例。有些还允许用户上传特定的虚拟机镜像。
- 虚拟机池管理器组件负责跟踪所有的活动实例。
- 最后，如果系统支持集成第三方 IaaS 供应商提供的资源，服务提供组件与调度程序

相互作用就可以提供一个虚拟机实例，该虚拟机实例是由资源池直接管理的本地物理设备的外部表示。

最底层是由物理基础设施构成的，在其上运行管理层。如前面所讨论的，基础设施可以是不同的类型，使用什么样的基础设施取决于具体所使用的云计算服务。服务供应商很可能会使用含有成百上千个节点的大规模的数据中心。小中型企业或大学所开发的云基础设施服务，最有可能依赖于一个集群。在底层也可以考虑由不同类型的资源组成的异构环境，包括个人电脑、工作站以及集群，这些资源都可以集成在一起。这种情况主要是指网格计算的演化。任何可用的计算资源（如 PC 和处于空闲状态的工作站）都可以被利用，以提供巨大的计算能力。从架构角度来看，物理层还包括从外部 IaaS 供应商租用的虚拟资源。

在完全 IaaS 方案中，所有三个层次都作为服务。Amazon、GoGrid、Joyent、Rightscale、Terremark、Rackspace、ElasticHosts 和 Flexiscale 等公共云供应商拥有大型数据中心并且可以通过 IaaS 访问计算基础设施。而其他解决方案只包括用户界面和基础设施软件管理层，需要认证才能访问第三方 IaaS 或拥有安装了管理软件的基础设施。这样的情况有 Enomaly、Elastra、Eucalyptus、OpenNebula 和来自 VMware、IBM 和微软的 IaaS（M）方案。

116

这个架构只是 IaaS 实现方式的一个参考模型。通过该参考模型可以了解云计算服务的普遍特性，以及实现云服务的方法。不同的解决方案可以配置额外的服务特性，甚至本书没有提及的一些特性。最后，用于实现 IaaS 的参考架构提供了计算资源以及调度组件。当存储作为主要服务时，仍然可以分为这三个层。基础设施管理软件的作用不是跟踪和管理虚拟机的执行情况，而是对大型基础设施进行访问并在物理层之上实现存储虚拟化解决方案。

4.2.3　平台即服务

平台即服务（PaaS）解决方案提供了一个开发和部署平台，用来在云计算中运行应用程序。它们构成了能在其上构建应用程序的中间件。PaaS 方案其功能特性如图 4-3 所示。

117

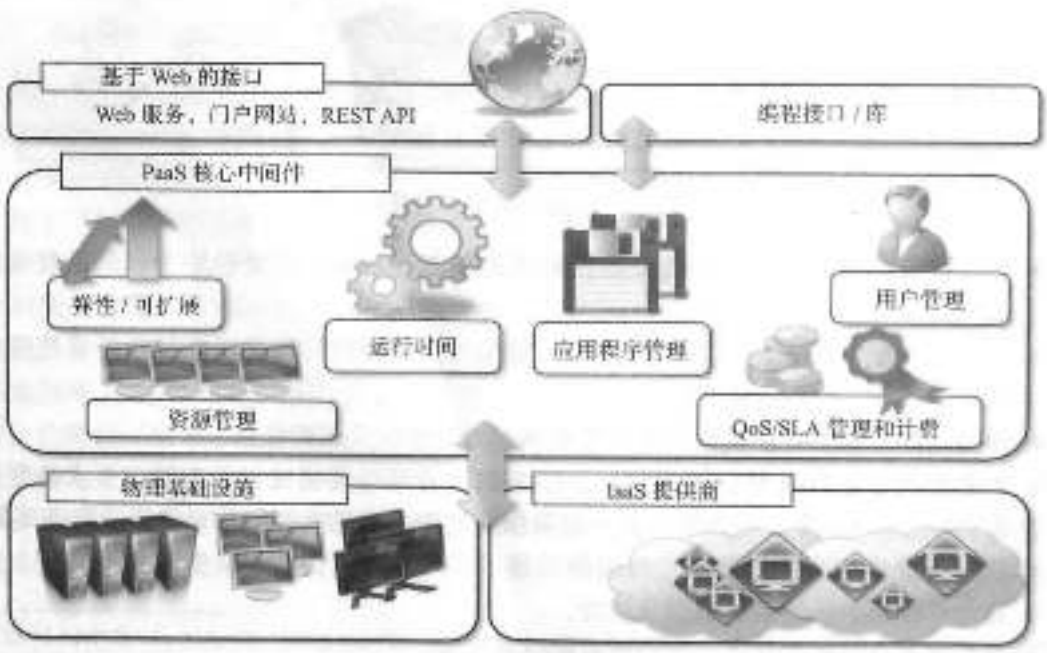


图 4-3　PaaS 参考模型

应用程序管理是中间件的核心功能。PaaS 为应用程序提供了运行环境，但并不提供任何管理底层基础设施的服务。PaaS 自动将应用程序部署到基础设施，配置应用程序组件，提供和配置支撑技术，比如负载均衡和数据库，并根据用户设置的策略调整管理系统。开发人员设计系统时针对的是应用方面，他们并不关心硬件（物理的或虚拟的）、操作系统以及其他低层服务。根据对用户的承诺，核心中间件负责管理资源并按需自动地扩展应用。从用户的角度来看，核心中间件提供接口，允许用户在云计算上编程和部署应用。这些接口可以是基于 Web 的界面形式，也可以是 API 编程和库的形式。

应用程序的具体开发模型决定了提供给用户的接口。有些实现提供了完全基于云计算的 Web 界面，并且提供各种服务。可以基于 4GL 和可视化编程概念集成开发环境，或基于快速原型环境，其中应用程序由 mash-ups 和用户自定义组件及后续客户化订制完成组装。PaaS 模型的其他实现方案提供了一个完整的对象模型来表示应用程序，并提供一种基于编程语言的方法。这种方法通常提供更多的灵活性和机会，但会有较长的开发周期。开发人员一般都掌握了丰富的编程语言，如 Java、.NET、Python 或 Ruby，但对提供更好的可扩展性和安全性有一定的限制。在这种情况下，传统的开发环境可用于设计和开发应用程序，然后通过使用 PaaS 供应商提供的 API 将其部署在云中。组件可以和开发库一起工作以更好地利用由 PaaS 环境所提供的服务。有些时候，模拟云计算的本地运行环境可用于用户在部署应用程序前对程序进行测试。这样的环境只局限于测试软件的功能，一般不用于优化软件的可扩展性。

PaaS 解决方案可以提供开发应用程序的中间件和基础设施，或按照用户契约简单地为用户提供安装软件。在第一种情况下，PaaS 供应商拥有运行应用程序的大型数据中心；而在第二种情况下，也就是本书提到的纯 PaaS，中间件是其提供的核心价值，也可能有同时提供中间件和基础设施的供应商，但可以只向私有平台提供中间件。

表 4-2 所示是最常见的 PaaS 实现方法的分类。这些解决方案可以分为三大类别：PaaS-Ⅰ、PaaS-Ⅱ和 PaaS-Ⅲ。PaaS-Ⅰ的实现完全遵循云计算模式的应用开发和部署，提供了基于 Web 浏览器的集成开发环境，可以进行应用程序的设计、开发、集成和部署。Force.com 和 Longjump 就属于这一类，两者都能够提供由中间件和基础设施组成的平台。在第二类中，表中列出了所有专注于为 Web 应用程序提供可扩展基础设施的解决方案，大多数是网站。在这种情况下，开发者通常使用服务供应商的 API 来开发应用程序。谷歌 AppEngine 是这一类最具代表性的产品。它提供了基于 Java 和 Python 编程语言的可扩展的运行环境，并且已经过修改，现可提供更安全的运行环境，以及丰富的 API 和组件，以增强其可扩展性。AppScale 是谷歌 AppEngine 的开源实现，提供接口兼容的中间件，必须安装在物理基础设施上。Joyent Smart Platform 提供了类似谷歌 AppEngine 方法。Heroku 和 Engine Yard 采用了不同的方法，在基于 Rails 的网站对 Ruby 程序提供了可扩展性。在这种情况下，开发人员用传统方法设计和创建应用程序，然后上传和部署到服务供应商的平台。 118

第三类是由所有提供云计算编程平台的解决方案组成的，它支持任何类型的应用程序，而不只是 Web 应用程序。其中，最典型的就是微软 Windows Azure，它提供了全面的构建面向服务的云应用的框架，此框架基于 .NET 技术，托管在微软的数据中心。在同一类别中的其他解决方案，如 Manjrasoft Aneka、Apprenda SaaSGrid、Appistry Cloud IQ 平台、DataSynapse 和 GigaSpaces DataGrid，仅提供不同服务的中间件。表 4-2 只描述了 PaaS 市场的部分产品。

表 4-2　PaaS 实现方法分类

类型	描述	产品类型	供应商和产品
PaaS-I	基于 Web 应用程序开发平台的运行时环境。快速应用程序原型设计	中间件 + 基础设施	Force.com
		中间件 + 基础设施	Longjump
PaaS-II	扩展 Web 应用程序的运行环境。运行时可以通过提供扩展功能的附加组件来加强性能	中间件 + 基础设施	谷歌 AppEngine
		中间件	AppScale
		中间件 + 基础设施	Heroku
		中间件 + 基础设施	Engine Yard
		中间件 + 基础设施	Joyent Smart Platform
		中间件	GigaSpaces XAP
PaaS-III	用于在云计算中开发分布式应用程序的中间件和编程模型	中间件 + 基础设施	微软 Azure
		中间件	DataSynapse
		中间件	Cloud IQ
		中间件	Manjrasoft Aneka
		中间件	Apprenda SaaSGrid
		中间件	GigaSpaces DataGrid

PaaS 包括了云计算中开发和部署应用程序的各种解决方案。尽管存在不同的方案，但
119 也可以建立统一的执行标准。正如 Appistry.com[⊖] 的产品经理 Sam Charrington 所说，PaaS 解决方案中有一些共同的基本特征：

- 运行时框架。这个框架代表 PaaS 模型的“软件堆栈”，是人们对于 PaaS 解决方案的最直观的印象。运行时框架根据用户和供应商设置的策略执行终端用户代码。
- 抽象。根据所提供的抽象高层应用的不同可以区分不同的 PaaS 解决方案。而 IaaS 解决方案的重点是提供对虚拟或物理基础设施的“原始”访问，PaaS 重点关注云计算必须支持的应用程序。这意味着 PaaS 解决方案提供的是部署和管理云计算应用的方法，而不是在其上构建和配置 IT 基础设施的虚拟机。
- 自动化。PaaS 环境能够在基础设施上自动部署应用程序，并且通过按需配置更多资源来扩展应用。这个过程是根据客户和供应商之间签订的 SLA 自动完成的。IaaS 通常没有此功能，它只负责如何提供更多资源。
- 云服务。PaaS 向开发人员和架构师提供服务和 API，帮助他们简化创建和交付具有可扩展性和高可用性的云应用。这些服务是不同的 PaaS 解决方案的关键区别，一般包括用于开发应用程序的特定组件，以及用于应用监控、管理和报告的高级服务功能。

PaaS 的另一个重要功能是能够利用面向服务架构集成其他服务供应商的第三方云服务。这种集成通过标准接口和协议完成，使得应用程序的开发更加灵活，并能根据客户和用户的需求来开发。许多 PaaS 产品都提供了这一功能，并将其内置到云计算解决方案的框架中。

利用 PaaS 解决方案实现应用程序的主要关注点之一是厂商绑定。这与 IaaS 方案不同，IaaS 提供裸虚拟服务器，并支持软件协议栈的可定制服务。PaaS 环境提供的是用于开发应用程序的平台，公开了定义好的 API 集，而且在大多数情况下，应用程序与 PaaS 供应商的具体运行环境绑定。即使基于平台的方法可以简化应用程序的开发和部署周期，但这些应用存在完全依赖于服务供应商的风险。如果服务供应商终止了契约，需要把应用迁移到另

⊖ 详细分析参见 Cloud-pulse 博客，网址 http://Cloudpulsehlog.com/2010/02/the-essential-characteristics-of-paas。

一个运行环境时，这种依赖关系就成为一个显著的障碍。不同解决方案对服务绑定应用的影响不同。例如，Force.com 依靠专有的运行框架，这使得应用迁移非常困难。再如，谷歌 AppEngine 和微软 Azure 依靠行业标准运行，但要使用专用的数据存储设备和计算基础设施。在这种情况下，可以利用基于 PaaS 的解决方案实现相同的接口，但性能可能不同。而 Appistry Cloud IQ 平台、Heroku 和 Engine Yard 则完全依赖于开放的标准，从而使应用程序更容易移植。 120

最后，从财务方面看，虽然 IaaS 解决方案允许通过外包将资金成本纳入运营成本，但 PaaS 解决方案在应用开发、部署用户方面是透明的，能够集中精力于业务的核心价值。当 PaaS 与底层的 IaaS 解决方案捆绑在一起时，将有助于小型创业公司在托管平台上以很小的成本迅速为客户提供集成的解决方案。这使得 PaaS 可以针对不同市场提供可行的方案。

4.2.4 软件即服务

软件即服务（SaaS）是一种基于 Web 的软件交付模式，提供通过互联网访问应用程序的服务。这种方案将复杂的硬件和软件管理任务交给第三方，从而减轻用户的负担，通过 Web 浏览器允许多用户访问同一应用程序。在这种情况下，用户既不需要在终端安装任何东西，也不必支付相当大的前期成本购买软件和所需的许可。他们只需访问应用程序网站，输入证书和账户详细信息，就可以立即使用应用程序，在大多数情况下，还可以进一步定制软件以满足其需要。在供应商端，每个用户应用程序的具体细节和功能特性都在基础设施中维护，并按需提供。

SaaS 模式对服务多用户的应用程序是很有吸引力的，并且几乎不需要进一步定制就能满足特定需求。SaaS 模式的特点是“一对多”的软件交付模式，即一个应用程序由多个用户共享。CRM[⊖]和 ERP[⊜]应用程序就属于这种情况，它们已成为所有企业的共同需求，不论是小型、中型还是大型的企业都需要 CRM 和 ERP 系统，即使需求不同也可以通过进一步的客户化得到满足。此方案提供了一个通用的功能集并且支持新组件的专业化集成，易于实现软件平台的开发。此外，它是托管方案的最佳选择，因为交付给用户的应用程序是相同的，而且应用程序本身为用户提供了一种方法，可以根据用户需要来创建应用程序。因此，SaaS 应用自然有很多租户。多租户是 SaaS 相对于传统软件的一个特点，它使服务供应商能够集中精力去管理大型硬件基础设施、维护和升级应用程序，并在庞大的用户群之间分担成本以优化资源。在客户端，这样可以使软件的使用费用最小化。 121

正如前面所指出的（见 1.2 节），在云计算之前，软件即服务的概念出现在 20 世纪 90 年代末，那时它开始获得市场认可 [31]。其缩写 SaaS 于 2001 年由软件与信息产业协会（SIIA）[32] 创建，其含义如下：

> 在软件即服务模型中，应用程序或服务通过网络（互联网、内联网、局域网或 VPN）在一个集中的数据中心被部署、访问并且产生使用费用。用户从服务集中供应商那里租用或订阅应用程序的使用权。商业模式根据软件的精简程度而有所不

⊖ CRM 是客户关系管理（customer relationship management）的首字母缩写，关注与客户和销售预期的交互，是用于简化客户管理和识别销售策略的软件系统。

⊜ ERP 是企业资源计划（enterprise resource planning）的缩写。ERP 通常指管理企业内部和外部资源的集成计算机系统，管理的资源包括有形资产、原材料、财务和人力资源。ERP 软件提供了集成的企业管理视图，易于实现商业活动与资源之间的信息流管理。

同，从而降低价格和提高效率，或通过客户化定制增加附加值，进一步改进数字化业务流程。

SIIA 的分析主要涵盖应用服务供应商（ASP）的不同解决方案，在广泛意义上理解软件应用作为一种服务的概念。ASP 具有 SaaS 的核心特征：

- 销售给用户的产品是应用程序的访问权。
- 应用程序被集中管理。
- 提供的服务是一对多的。
- 提供的服务是基于承诺契约的综合方案。

最初，ASP 提供的是应用程序软件包的托管解决方案，以便为多个客户服务。之后，其他方案（如基于 Web 的第三方应用服务）受到广泛关注，给独立软件供应商和服务供应商带来了新的机会。这些机会最终演变成一个更灵活的模式，就是将应用作为一种服务：SaaS 模式。ASP 提供访问软件包的解决方案以满足用户的各种需求。最初服务供应商能承受这种服务方法，但当客户化定制成本增加时就不太容易了。SaaS 引入了完全由用户自定义的更灵活的交付服务方式，用户可以集成新的服务，加入自己的组件，并可以设计应用程序和信息的工作流。这种新的方法在 Web 2.0 技术的支持下得以实现，将 Web 浏览器转变为一个功能齐全的界面，甚至支持应用程序的组合和开发。

云计算与 SaaS 有什么关系呢？根据图 4-1 所示的服务分类，SaaS 处于云计算层次结构
[122] 的顶层，它符合 XaaS（一切即服务）的云计算构想，并且 SaaS 是把应用程序作为服务来交付的。最初，只有领先的用户和早期采用者对 SaaS 模式感兴趣。在那个阶段，SaaS 带来的好处如下：

- 降低软件成本和总体成本。
- 服务水平的改进。
- 快速实施。
- 独立的、可配置的应用。
- 应用程序和数据初步集成。
- 订阅和即用即付的定价方式。

随着云计算的到来，SaaS 作为一种可行的软件交付模式日益得到人们的接受。这使得 SaaS 过渡到 SaaS2.0[40]，但并没有引入新的技术，只是 SaaS 的应用方式有所改变。

SaaS2.0 尤其专注于提供更具鲁棒性的基础设施和应用平台。与其说 SaaS2.0 能更快地实现和部署应用环境，还不如说 SaaS2.0 能更快地实现业务目标。这就是为什么这样的演进并不需要任何新技术，而是将现有的技术集成在一起，从而有效地实现业务目标。这一观点的基础是利用现有方案和集成增值业务服务的能力。现有的 SaaS 基础设施不仅支持应用程序的开发和定制，而且还易于集成第三方提供的服务。这样，SaaS 应用程序是不同的应用和组件相互连接、协同作用的结果，并且一起为用户提供附加值的服务。这种方法极大地改变了 SaaS 市场的软件生态系统，不再由少数厂商垄断，而是众多服务供应商形成服务网络，为客户提供应用程序服务。在这种情况下，集成在 SaaS 应用的每个单一组件都向用户承诺确保履行其 SLA，同时可以根据不同的服务等级定价。然后用户确定需要集成哪些组件和服务的应用服务。

软件即服务的应用程序可以满足不同需求。CRM、ERP 和社交网络应用程序无疑是最典型的应用。SalesForce.com 可能是 CRM 服务最成功和最典型的例子。它提供了广泛的应

用服务：客户关系和人力资源管理、企业资源计划，以及其他功能。SalesForce.com 建立在 Force.com 平台上，它提供了一个用于构建应用程序的功能齐全的环境，将编程语言和可视化环境结合在一起来构建应用程序。除了基本功能外，支持与第三方应用程序集成的特性使 SalesForce.com 功能更加丰富。特别地，用户可以通过 AppExchange 发布、搜索，以及把新的服务或功能整合到其现有的应用程序中。这使得 SalesForce.com 的应用拥有完全的可扩展性和可定制性。NetSuite 和 RightNow 也提供类似的方案。NetSuite 是集成软件业务，包括财务、客户关系管理、库存管理和电子商务等一体化的功能。RightNow 是以客户体验为中心的 SaaS 应用程序，它将不同功能集成在一起，从聊天软件到 Web 社区软件，以支持企业的业务活动。

另一类 SaaS 应用程序用于社交网络，如 Facebook 和专业网站 LinkedIn。除了提供网络基本功能，它们还支持集成第三方应用程序来整合和扩展其应用服务功能，这些应用服务功能被开发为托管平台的插件并提供给用户。如 Facebook，用户可以选择所需应用程序加入到自己的配置文件。因此，集成的应用程序可以完全访问联系人的网络和用户配置文件数据。这些应用包括不同的类型：办公自动化、游戏或其他已有服务的集成。

办公自动化应用也是 SaaS 应用的主要方面，谷歌 Documents 和 Zoho Office 是基于 Web 应用的实例，旨在解决用户所需要的文档管理、电子表格和演示管理。它们提供了一个基于 Web 的界面，用于创建、管理和修改文档，这些文档易于在用户间共享并且在任何地方都能访问。

SaaS 解决方案发挥着很重要的作用，它提供了整合第三方服务以及共享信息的环境。一个成功的例子是 Box.net，它为用户提供 Web 空间和配置文件的 SaaS 应用程序，并利用第三方应用程序提供丰富的可扩展应用，如办公自动化、集成 CRM 系统、社交网站和照片编辑。

4.3 云的种类

云计算是一种并行和分布式系统，它将物理机和虚拟机作为统一的计算资源。云构建基础设施并以服务的形式实现和交付给客户。这种基础设施可以是不同类型的，并且通过云提供关于属性和服务的信息。云类型根据云管理域划分，它确定了云计算服务实施的范围，确定适合提供某种服务的底层基础设施，并且对它们都做了规定。有四种不同类型的云：

- 公共云。开放给广大公众的云。
- 私有云。云计算是在一个机构内实施的，通常机构成员或子成员可以访问。
- 混合或异构云。这种云是前两个方案的组合，通常是私有云资源或服务经扩展后可在公共云中部署。
- 社区云。这种云的特点是多管理域包括不同的部署模型（公共、私有和混合型），专为满足特定行业的需求而设计。

几乎所有的云实现方式都可以按此分类，后续章节将简要介绍不同类型的云的特征。

4.3.1 公共云

云计算的第一种表现形式是公共云。云计算的权威观点是可以通过网络从任何地方、在任意时间、向任何人提供可用服务。从结构上看，它是一个分布式系统，由一个或多个连接在一起的数据中心构成，云计算服务在数据中心的基础上得以实现。任何客户可以方便地登

录云服务供应商，输入详细的账户和计费信息，就可以使用云服务供应商所提供的服务。

公共云是历史上实现云计算的第一类云。公共云提供了最大限度地降低 IT 基础设施成本的解决方案，并且是一种可行的处理本地基础设施高峰负荷的有效方法。对于小企业而言，公共云已经成为一种有吸引力的选择，企业可以完全依赖于公共基础设施来满足 IT 需求，从而不需要大量的前期投资就可以开展业务。相比于私有设施的改造和软硬件的购置，公共云具有如此吸引力的原因是它可以根据相关业务的需求进行增长或收缩。通过租用基础设施或订购应用程序服务，客户能够根据实际情况动态地增加或缩减自己的业务。目前，公共云既可以完全取代企业的 IT 基础设施，也可以在需要的时候对其进行扩展。

公共云的基本特征是多租户。公共云的目的是服务众多用户，而不是单一用户。任何客户都能拥有一个与其他使用者分离的虚拟计算环境，这是有效监控用户活动以及保证用户期望性能和其他 QoS 指标的基本要求。QoS 管理是公有云的一项非常重要的功能。因此，软件系统的重要功能之一是监控云资源，根据用户契约收费，并保存每个用户使用云计算的完整记录。这是公有云的基本功能，支持供应商向用户提供按需即用即付服务。

公共云可以提供任何类型的服务：基础设施服务、平台服务或应用服务。例如，亚马逊 EC2 是提供基础设施即服务（IaaS）的公共云，谷歌 AppEngine 是提供应用开发平台即服务（PaaS）的公共云、SalesForce.com 是提供软件即服务（SaaS）的公共云。公共云的特性是其使用方式：公共云为每个用户提供服务，并支持大量用户使用。其特点是按需扩展能力和支持峰值负载。

从系统架构的角度来看，没有限制实现公共云的分布式系统的类型。最常见的是一个或多个数据中心构成物理基础设施，然后在其上部署实现服务并交付使用。公共云可以由地理上分散的数据中心构成，这些数据中心分担用户任务负载，根据用户的位置更好地为其服务。例如，亚马逊 Web 服务的数据中心安装在美国、欧洲、新加坡和澳大利亚，允许用户在三个不同区域之间进行选择：美国 – 西 -1，美国 – 东 -1 或欧盟 – 西 -1。这些地区的价格有所不同，并被进一步划分为可用性区域，以对应特定的数据中心。根据云提供的具体服务类型，需要安装不同层次的软件来管理基础设施，包括：虚拟机管理器、分布式中间件或
125 分布式应用程序。

4.3.2 私有云

公共云非常流行，它降低了 IT 成本并减少了资本支出，但并不适用于所有情况。举例来说，其公认缺点是云用户采用云计算时失去了控制权，在公有云的情况下，服务供应商控制着基础设施，甚至用户的核心业务逻辑和敏感数据。虽然有监管程序保证公平管理和尊重客户隐私，但这种情况仍然存在威胁或不可接受的风险，所以一些组织不愿意采用公共云。特别是，政府和军事机构不会使用公有云处理或存储敏感数据。供应商可能把安全基础设施中的信息暴露给其他人，这样的风险被认为是不可接受的。

在其他情况下，失去对虚拟 IT 基础设施的控制可能导致其他问题。更确切地说，一个数据中心的地理位置通常决定了数字信息管理的规则。其结果是，根据数据的具体位置，如果用特定的加密技术进行处理，政府机构可访问一些敏感的甚至超出法律范围的信息。例如，美国爱国者法案（USA PATRIOT Act）[⊖]允许政府和某些机构获取存储在美国本土的属

⊖ 美国爱国者法案是由美国政府制定的法规，用于加强司法机构搜索电话、电子邮件、医疗、财务和其他记录的力度，限制外国情报机构在美国收集情报。法案全文在国会数据库网站上，网址 http://thomas.loc.gov/cgi-bin/bdquery/z?d107:hr03162:（2010 年 4 月 20 日检索）。

于任何公司的信息。因此，现有的拥有大规模计算基础设施或安装有大型软件的企业并不想简单地切换到公共云，但他们可以利用现有的IT资源提高其收益。这些因素导致使用公共计算基础设施并不一定总是可行的。然而，云计算的总体思路依然引人注目。更具体地讲，拥有基础设施并能够按需提供IT服务仍然是一个成功的解决方案，即使是在一个机构内部实施。这种想法导致了私有云的产生，它与公共云很相似，但其资源提供在组织内实现。

私有云是虚拟的分布式系统，依赖于私有基础设施并向内部用户动态提供计算资源。考虑到企业内不同部门的云计算使用情况，取而代之的可能是其他方案，而不是公共云中即付即用的计费模式。一旦云计算建立，便可依托现有的IT基础设施且维护成本降低，使得私有云具有保持企业内部核心业务的优势。在这种情况下，安全问题并不重要，因为敏感信息不会从私有基础设施流出。此外，现有的IT资源可以得到更好的利用，因为私有云可以为不同的用户群提供服务。与公共云相比，私有云的一个优势是可以以一个较低的价格测试应 126
用程序和系统。Forreste报告[34]强调了一些使用私有云计算基础设施的关键优势：

- 客户信息保护。尽管公共云服务供应商对安全性做出了承诺，但很少令人满意，由于推行云服务的时间不够长，也很少有人能为其系统特定安全等级提供保证。私有云的安全保障更容易维持且值得信赖。
- 保证SLA的基础设施服务。服务质量意味着提供特定的操作，如适当的集群和故障处理、数据复制、系统监控和维护，以及灾难恢复和其他满足应用需求的正常运行的服务。虽然公共云供应商提供了一些功能，但不是所有功能都能满足需求。
- 服从标准程序和操作。如果组织必须遵守第三方的合规标准，那么部署和执行应用程序时，特定的标准程序就需要被替代。这在虚拟公共基础设施的情况下是不可能的。

这些方面使得私有云成为可行的选择。

从系统架构的角度来看，私有云可以在更多异构硬件上实现，它们通常依赖于已经部署在私有场所的现有IT基础设施。这些基础设施可能是数据中心、集群、企业网格或者是它们的组合。根据交付给云计算使用者的服务类型，物理层可以由基础设施管理软件（例如IaaS（M），见4.2.2节）或者PaaS解决方案实现。

私有云的实现可以采用不同的方案。图4-4提供了解决方案的全面视图以及一些用来部署私有云的典型参考软件。在软件层次结构的底层，虚拟机技术（如Xen[35]、KVM[36]和VMware）作为云计算的基础。虚拟机管理技术（如VMware vCloud，Eucalyptus[37]以及OpenNebula[38]）可以用来控制虚拟基础设施，并提供IaaS解决方案。VMware vCloud是一个专有的解决方案，但Eucalyptus提供完整的与亚马逊Web服务兼容的接口并支持不同的虚拟机技术，如Xen、KVM和VMware。像Eucalyptus一样，OpenNebula是一个开源的解决方案，用于虚拟基础设施管理，支持KVM、Xen和VMware，这种设计是为了方便地集成第三方IaaS供应商。OpenNebula模块化架构允许扩展软件的附加功能，如使用Haizea[39]调度器预约虚拟机实例。

OpenPEX[40]和InterGrid[41]依赖于以前的虚拟机管理器并且提供附件功能。OpenPEX是基于Web的系统，允许预留虚拟机实例，并能够支持不同的后台终端（目前只实现了对Xen的支持）。InterGrid在OpenNebula和亚马逊EC2基础上提供附加价值服务，支持虚拟机实例的预留和多管理域云计算环境的管理。PaaS方案可以提供一个附加层，

并为私有云提供高层服务。在部署私有云的可选方案中，可以考虑 DataSynapse、Zimory
127 Pools、Elastra 和 Aneka。DataSynapse 是应用虚拟化软件服务的全球供应商。依靠 VMware 的虚拟化技术，DataSynapse 提供了一个可以在数据中心上构建私有云的灵活环境。Elastra Cloud Server 是易于在云计算环境中配置和部署分布式应用基础设施的平台。Zimory 基于 Xen、KVM 和 VMware 虚拟化技术，提供了可以自动使用资源池的软件基础设施层。它允许创建一个由稀少的私人和公共资源组成的内部云，并提供用于在现有的基础设施中迁移应用程序的工具。Aneka 是一个软件开发平台，可以用于在异构硬件（数据中心、集群和网络）上部署云基础设施。它提供了一个插件式的面向服务的架构，主要支持基于不同编程模型（任务背包、MapReduce 等）的分布式应用程序的执行。



图 4-4 私有云硬件和软件层次结构

私有云可以为云计算提供内部解决方案，但与公共云相比，私有云在按需进行弹性扩展方面能力较为有限。

4.3.3 混合云

公共云是有巨大能力的、足以服务于多个用户需求的大型软硬件基础设施，但是受到安全威胁和管理缺陷的影响。尽管对于不希望产生 IT 投入成本或刚开始考虑其 IT 需求的企业（如初创企业），完全依靠公共虚拟基础设施是有吸引力的选择，但在大多数情况下，依靠企
128 业已有 IT 基础设施构建私有云却相对占优势。

当需要在企业内部处理信息或者需要使用现有的硬件和软件基础设施时，私有云是比较完美的解决方案。私有云部署的一个主要缺点是不能按需扩展和有效地解决峰值负荷。在这种情况下，根据需要利用公共云的能力很重要。因此，混合解决方案可以最合理地利用私有云和公共云的优势，使得混合云得以发展和推广。

混合云使企业能够利用现有的 IT 基础设施、保存敏感信息、按需自动扩展和缩减资源使用，按需增加配置外部资源并在不需要时释放资源。安全问题只限于云计算的公共操作部分，用于不严格约束的执行操作，但该部分仍然是系统工作负载的一部分。混合云的总体结构如图 4-5 所示，它是一个异构分布式系统，是集成了一个或多个公共云的附加服务或资源的私有云，因此也称为异构云。如图 4-5 所示，动态分配是混合云环境的基本组成部分。通

过充分利用外部资源满足超负荷需求，混合云解决了可扩展性问题。这些资源或服务被暂时 129
租用一段时间，然后释放。这种做法也称为云爆发[⊖]。

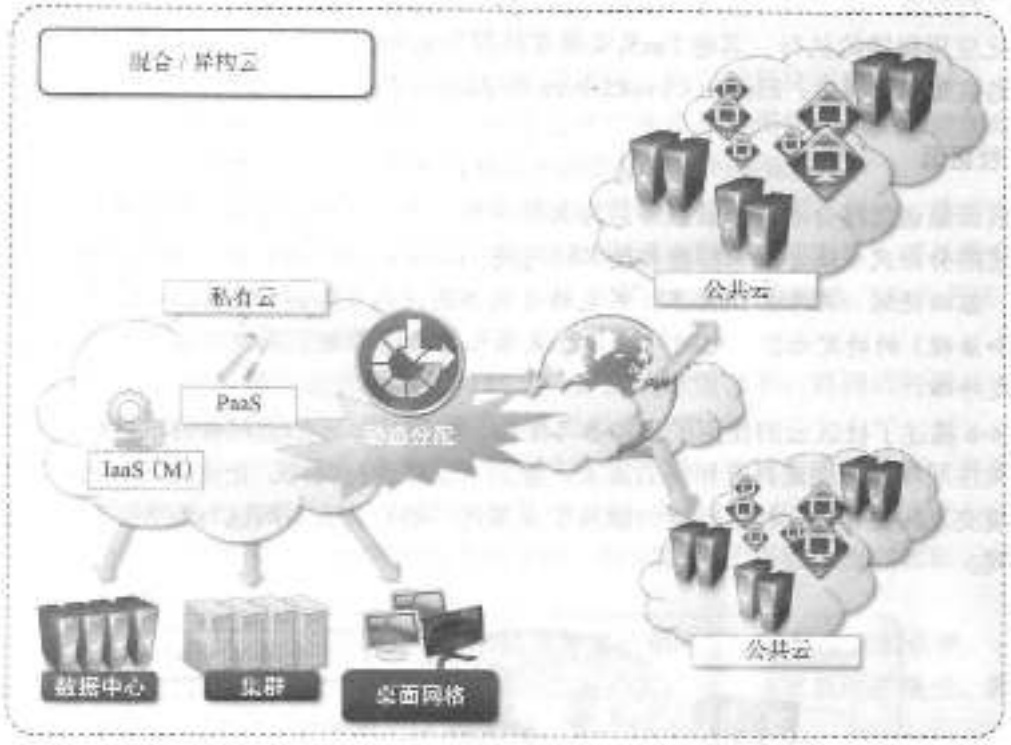


图 4-5 混合云/异构云结构简图

虽然混合云的概念是通用的，但它主要适用于IT基础设施，而不是软件服务。面向服务的计算已经引入了将付费软件服务与企业内部已有应用程序集成的概念。在IaaS的环境中，动态分配指的是按需获取或释放虚拟机的能力，以增强分布式系统的功能。基础设施管理软件和PaaS解决方案是部署和管理混合云的基本模块。特别是对于私有云，动态配置包括更复杂的调度算法和策略，其目标是优化用于租用公共资源的预算。

基础设施管理软件（如OpenNebula）已经展现出整合公共云（如亚马逊EC2）资源的能力。在这种情况下，从公共基础设施中获得的虚拟机和所有其他虚拟机实例一样在本地进行统一管理。不过，缺少能够区分这些资源并按预算进行智能资源分配的先进调度引擎。例如，OpenNebula集成了Haizea的先进调度引擎来提供基于成本的调度算法。InterGrid则采取了一种不同的方法。它本质上是在多个网络节点中管理和分配虚拟机的分布式调度引擎，网络节点可以是本地集群、公共云的网关或两者的结合。请求一旦被提交给InterGrid的某一网关，就可能为其分配所有网络节点上的虚拟实例，根据用户的预算和网络节点结构为请求分配资源。

动态分配是支持混合云的PaaS解决方案中最常见的功能。如前面所讨论的，PaaS中间件的基本功能是实现分布式应用程序到云基础设施的映射。在这种情况下，动态分配成为满

⊖ 根据Wiki对云计算的解释，云爆发有双重含义，也指“云计算环境不能处理峰值突发需求时出现故障”（http://sites.google.com/site/Cloudcomputingwiki/Home/Cloud-computing-vocabulary）。本书中云爆发都是指公共云资源的动态提供。

足用户 QoS 的应用执行的根本保障。例如，Aneka 提供了一个调配服务，可以利用不同的 IaaS 供应商扩展现有的云计算基础设施 [42]。该调配服务和调度器一起保证应用程序的特定 QoS。特别地，每个用户应用程序都有预算，调度器可以使用该预算通过按需租用虚拟节点来优化应用程序的执行。其他 PaaS 实现支持混合云的部署且提供动态分配功能。在这些
130 私有云的实施和管理中，Elastra CloudServer 和 Zimory Pools 是很好的实例。

4.3.4　社区云

社区云是通过整合不同的云服务，为满足一个行业、一个社区或一个业务部门的特定需求而创建的分布式系统。国家标准和技术研究院（NIST）[43] 对社区云的描述如下：

> 基础设施由几个组织共享，并支持关注共同问题（例如，任务、安全要求、政策和法规）的特定社区。它可以由组织或第三方负责管理，并且可能存在于组织内部或外部。

图 4-6 描述了社区云的使用场景和参考架构。特定社区云的使用者分成标识明显的不同社区，关注同样的问题或具有相同的需求。他们可以是政府机构、企业或简单用户，但关注的云环境交互问题相同。社区云既与服务于众多用户的公共云不同也与服务于机构内部的私有云不同。



图 4-6　社区云

131 从架构的角度来看，社区云可在多个管理域上实施。这意味着，政府机构、私有企业、科研机构甚至是公共的虚拟基础设施供应商等不同的组织，都可以利用他们的资源来构建云基础设施。

采用社区云的行业如下：

- 媒体行业。媒体行业公司都在寻找低成本、灵活和简单的解决方案，以提高产品效率。大多数媒体产业是与合作伙伴组成的业务扩展生态系统。特别地，数字内容的创建是一个合作的过程，包括大数据移动、大规模的计算密集型渲染任务以及复杂的工作流执行。社区云可以提供一个促进企业和企业合作的共享环境，并有效提供媒体制作所需的带宽、CPU 和存储能力。

- 医疗行业。社区云也可以应用在医疗行业。特别地，社区云可以提供一个全球性平台，在不泄露保存在私有基础设施内的敏感数据的前提下共享信息和知识。社区云的混合部署模式可以很容易地实现将病人相关数据存储在私有云中，而使用共享基础设施来提供非关键医疗服务和自动处理流程。
- 能源等核心产业。在这些部门，社区云可以提供一整套的解决方案，同时自上而下执行管理、部署和服务协同及执行。由于这些产业涉及不同的供应商、销售商和组织，社区云可以提供合适的支持开放和公平的市场环境的基础设施。
- 公共部门。公共部门的法律和政治条款可以限制公共云的使用。此外，政府流程涉及多个机构和组织，目的是在地方、国家和国际层面提供战略方案。涉及商业对行政、公民对政府和商业对商业流程。例如发票审批、基础设施规划和公开听证会。社区云提供了用于创建此类操作交流平台的分布式环境。
- 科学研究。科学计算云是社区云流行的实例。在这种情况下，共同的科研兴趣促使不同的组织共享大型分布式基础设施的方式是科学计算。

社区云是云计算的一个更具体的类型，它关注在云计算中对销售商的控制，并把云计算的案例研究和数字生态系统[⊖][44]原则相结合。社区云利用用户机器[45]未充分使用的 132
资源，提供用户、生产者和服务协同代理都能在同一时间访问的基础设施。社区云的优点如下：

- 开放性。社区云是开放的系统，存在不同的方案，消除了对云供应商的依赖。
- 共同体。社区云基于公共平台提供资源和服务，其基础设施更具可扩展性，随着其用户群的扩大，系统易于扩展。
- 非单点故障。由于不是由单一供应商来控制基础设施，所以没有单点故障。
- 便利和可控。在社区云中便利性和可控性之间没有冲突，因为云是共享的，由社区共同拥有，所有决定都由集体民主决定。
- 环境的可持续性。社区云碳排放量小，因为它能利用未充分使用的资源。此外，这些云根据一种共生的关系扩展或收缩资源来支持社区的需求，具有可持续性。

这是社区云的另一种设想，更注重云的社会层面，集成社区成员的资源以提供云环境。之前的定义也体现出了建立异构基础设施以服务于社区用户的理念，但更关注社区用户的共同利益。在这两种情况下，基础都是社区的概念。

4.4 云计算经济特性

云计算的主要驱动因素是规模经济以及简单的软件交付和操作。其实，最大的好处体现在经济方面：由云服务供应商采用即用即付模式。具体而言，云计算优点如下：

- 减少 IT 基础设施的投入成本。
- 消除了与 IT 固定资产相关的折旧成本。
- 以订阅方式替代软件许可。
- 降低了 IT 资源的维护和管理成本。

资本成本是购买资产的成本，用于产品生产或提供服务。资本成本通常是预先支付的一

⊖ 数字生态系统是分布式的、自适应的开放技术系统，受自然生态系统启发，具有自组织、可扩展、可持续特性。数字生态系统的主要目的是维持小型和中型企业的区域性发展。

次性花费，通常经过一段时间后产生利润。IT 基础设施和软件就是资本资产，因为企业需要它们来开展业务。目前，企业的主营业务是否涉及 IT 并不重要，因为企业内部肯定会有一个 IT 部门来处理许多信息化业务：工资核算、客户关系管理、企业资源规划、产品跟踪
133 和库存管理等。因此，对于任何企业，IT 资源都构成了其资本成本。尽量保持低资本成本是很好的做法，因为企业投入的资金要经过一段时间后才能产生利润；但更重要的是，这些投入的资产会随着时间不断贬值，且该成本会从企业的收入中减去，从而降低了企业的利润。IT 资本成本的折旧费用，就是随着时间的推移带来的硬件贬值，以及因为需要增加新功能而带来的软件产品更新。

云计算在企业中实现普及之前，花费在 IT 基础设施和软件上的预算构成了中型和大型企业的主要成本支出。许多企业都拥有一个小型或中型的数据中心，并且涉及维护、电力和冷却方面的运营成本，以及维护 IT 部门和 IT 技术支持的成本。此外，还有其他购买软件的昂贵费用。采用云计算方式后，这些成本显著降低，甚至完全消失。其中云计算模式的优势之一，是将先前用于购买硬件和软件的资本成本转移为租用基础设施和订阅软件的费用。这些费用可以根据业务需求和企业的情况得到更好的控制。云计算还降低了管理和维护成本。也就是说，不需要或很少需要管理人员去维护云基础设施。同时，IT 技术支持人员的成本也降低了。企业将不会面临 IT 资源的贬值问题，因为在这种情况下，IT 需求都由云计算来提供服务，不存在固定资产折旧。

云计算为企业节约成本的多少与使用云计算服务的具体方案有关，还与它们为企业产生利润与方式有关。小型企业创业初期可以在以下方面采用云计算模式，例如：

- IT 基础设施。
- 软件开发。
- 客户关系管理（CRM）和企业资源规划（ERP）。

在这种情况下有可能完全消除资本成本，因为没有初始 IT 投入。如果企业中已经有相当规模的 IT 资产，情况便会完全不同。此时，云计算（特别是 IaaS 解决方案）可以帮助管理企业短期内所产生需求的计划外资本成本，将这些费用转变成运营成本。例如，租用 IT 基础设施有助于更有效地管理高峰负荷而没有额外资本支出。只要增加的负载并不使用额外的资源，这些资源就可以释放，这样与资源相关的成本就没有了。这是大多数企业采用的云计算模型，因为很多企业已经拥有 IT 设施。另一种方式是当投入的 IT 资产贬值并需要更换时，缓慢地过渡到云计算解决方案。在这两种情况之间，有各种各样的云计算方案可以帮助
134 企业创造利润。

云计算的另一个重要方面是降低了由 IT 资产所产生的一些间接成本，如软件许可、碳排放。有了云计算，企业基于订阅方式使用应用软件，无需支付任何许可费用，因为软件仍然是由服务供应商拥有。充分利用 IaaS 解决方案，可以节省数据中心的空间，从而减少碳排放。有些国家（如澳大利亚）对碳排放征税，因此，通过减少或完全消除碳排放，企业可以少缴税款。

关于云计算的定价模式，服务供应商采用三种不同的策略：

- 分等级定价。在此定价模型中，云服务按服务等级提供，每个等级服务都有固定的计算服务规范，并且按 SLA 以小时为单位定价。亚马逊按此形式对 EC2 服务定价，不同的计算能力配置（CPU 的类型、速度、内存），其服务定价不同。
- 服务单位定价。这种模式更适合于云服务供应商提供的具体服务可以以单位计算的

情况，如数据传输和内存分配。在这种情况下，客户可以根据应用的需要更有效地配置系统。例如，GoGrid 采用的就是该模型，使得客户可以按 RAM/ 小时使用部署在 GoGrid 中的云服务器。

- 基于订阅定价。SaaS 供应商大多采用此模型，用户定期支付订阅费来使用软件或集成在应用程序中的特定组件服务。

所有成本都基于即用即付的模式，这是一种更灵活的按需交付 IT 服务的方法。这使得 IT 资本成本转换为运营成本，因为购买硬件的成本变成了租用硬件的成本，购买软件的成本变成了为使用软件所支付的订阅费。

4.5　云计算面临的挑战

云计算仍处于起步阶段，在工业界和学术界还存在许多挑战。目前学术界的一个重要工作就是确定云计算面临的挑战 [46 ～ 49]。本节关注的重点包括：云计算的定义和形式、不同云之间的互操作、标准的建立、安全性、可扩展性、容错以及组织方面。

4.5.1　云计算定义

如前所述，已经有一些云计算的定义及其所提供的服务和技术的分类。其中 NIST 给出了最全面的云计算描述 [43]。它概括的云计算特征包括：按需自助服务、广泛的网络访问、资源池、快速灵活和计量服务；服务分类有 SaaS、PaaS 和 IaaS；部署模型分为公共云、私有云、社区云和混合云。这种观点被许多 IT 从业者和学者认同。 135

除了 NIST 给出的定义，还有其他云服务分类。David Linthicum，BlueMountains 实验室的创始人，提出了更为详细的分类[⊖]，描述了 10 种不同的更适合企业实施的云计算分类。加州大学圣巴巴拉分校（UCSB）采取了不同方法 [50]，脱离了 XaaS 的概念，并试图定义云计算的本体概念。他们把云的概念分解为五个主要层次：应用程序、软件环境、软件基础设施、软件内核和硬件。每一层都针对云计算社区内不同用户的需求，并且大多数都建立在基础层上。作者描述这项工作是首次在不同云实体之间实现了功能层面和语义层面的交互模型。

这些特性和分类反映了云计算现阶段的含义，但处于起步阶段的云计算也在不断发展，这也是云计算的真正本质。值得注意的是，本书描述的云计算定义及主要特征只是一个参考，它会不断地随时间变化而完善。

4.5.2　云计算互操作性和标准

云计算以一种基于服务的模式来提供 IT 基础设施和应用，就像公用事业基础设施服务一样，如能源、水、电。为了充分实现这一目标，建立不同服务供应商解决方案之间的互操作性标准至关重要。对服务供应商的依赖性是各个阶段无缝采用云计算技术的障碍之一，这将妨碍用户使用其他服务供应商的解决方案，或即使可以使用，也会产生相当大的转换成本，并需要大量的时间。而改变服务供应商的情况是有可能发生的，因为客户需要找到一个更合适的解决方案来满足他们的需要，或者服务供应商已经不能继续提供用户所需的服务。云计算采用并实际执行的标准可以实现互操作并减少服务依赖性带来的风险。

⊖ David Linthicum，云计算概念框架，http://Cloudcomputing.sys-con.com/node/811519。

目前云计算中标准和互操作性的情况类似于早期的互联网时代，当时在协议和技术的使用上并没有共同的标准，每个组织都有自己的网络。然而，走向标准化进程的第一步已经迈出。云计算互操作论坛（CCIF）[⊖]、开放云联盟[⊜]和 DMTF 云标准孵化器[⊜]等组织都是标准化
[136] 工作的先行者。另一个值得关注的倡议是开放云计算声明[⊜]，它体现了在该领域开放标准对各利益相关方的好处。

标准化的工作大多涉及云计算架构的底层。特别是在 IaaS 市场，供应商所有的虚拟机形式导致了对服务供应商的依赖。努力在 IaaS 供应商之间提供兼容的虚拟机镜像可以提高互操作能力。开放虚拟化格式（OVF）[51] 试图给出描述虚拟机镜像的存储信息和元数据的通用格式。尽管 OVF 提供了完整的与平台无关的封装和分发虚拟机镜像的规范，但很少有厂商参照此规范导入静态虚拟机镜像。云计算面临的挑战是制定支持运行实例迁移的标准，即完全透明地从一个基础设施供应商迁移到另一个基础设施供应商。

另外，标准化工作的另一目标是为云计算系统设计通用的参考架构，并提供标准的交互接口。目前，不同解决方案之间的兼容性非常有限，缺乏统一的用于与云计算服务供应商进行交互的 API。在 IaaS 产品市场中，AWS 发挥着主导作用，其他 IaaS 解决方案大多是开源的，提供兼容 AWS 的 API，从而可作为有效的替代产品。即使在这种情况下，设计与 IaaS（通常是 XaaS）交互的通用 API 依然没有统一的趋势，将来这一领域会有相当大的发展。

4.5.3 可扩展性和容错性

按需扩展能力是云计算最具吸引力的特点之一。云计算支持在现有 IT 资源之外进行扩展，无论是基础设施（计算和存储）还是应用程序服务。要实现这样的功能，云中间件必须按具有不同规模的可扩展性的原则进行设计，例如性能、规模和负载。云中间件管理着数量庞大的资源和用户，这是依赖于云计算才能获得的效果，而企业系统要想具有此功能必须承担相当大的管理成本和维护成本。而这些成本是实际用于开发、管理和维护云计算中间件并向客户提供服务的开销。在这种情况下，容错能力是基本要求，有时甚至比提供一个非常有效和优化的系统更重要。因此，云计算所面临的挑战是设计高度可扩展的和容错的系统，且
[137] 易于管理，同时可以提供具有竞争力的性能。

4.5.4 安全、可信和隐私

安全、可信和隐私问题是云计算普及的主要障碍。传统的加密技术用于防止篡改数据和访问敏感信息。虚拟化技术的大量使用使现有系统面临着新的威胁，这是以前没有考虑过的。例如，在云中托管的应用程序可以处理敏感信息，这样的信息可以存储在云存储设备中，并使用先进的密码技术来保护数据，不经允许不能访问。虽然这些数据是在内存中进行处理，但它们必须由合法的应用程序解密，然而由于应用程序托管在虚拟环境中，虚拟机管理器就可以访问这些应用程序的内存页。这种情况的出现是因为缺乏对应用程序执行环境的控制，使得利用现有技术的新方法又给应用程序的安全性带来了新威胁。数据和流程控制的缺乏使用户对供应商的信任和需要的隐私级别成为严重的问题。

⊖ www.Cloudforum.org。

⊜ www.opencloudconsortium.org。

⊜ www.dmtf.org/about/cloud-incubator。

⊜ www.opencloudmanifesto.org。

一方面，用户需要决定是否信任服务供应商；另一方面，具体规定可以直接控制服务供应商与用户签订的信息保密性协议。而且，交付给终端用户的云服务是复杂的多层次服务，由第三方云服务供应商提供。在这种情况下，服务的交付将涉及多责任链，包括数据的安全管理、隐私规则的实施以及对服务供应商的信任，每个环节都可能带来了较严重的脆弱性。特别是检测到侵犯隐私或非法访问敏感信息时，很难确定谁应该为这样的违法行为负责。因此云计算所面临的挑战是如何从不同的角度（如技术、社会和法律等）设计安全和可信的系统。

4.5.5 组织方面

云计算引入了一个新的方式来使用和管理IT服务。更具体地说，存储、计算能力、网络基础设施和应用程序都通过互联网以可计量的服务形式来交付。这就引入了计费模式，这在传统企业的IT部门内是种新的方式，需要一定程度的文化和组织处理程序。特别是为了接受云计算模式，业务流程和组织结构将需要进行显著地改变。那么，IT部门在这个新方案中的作用是值得考虑的问题。具体而言，以下几个问题必须考虑：

- 在全部或显著依赖于云计算的企业中，IT部门扮演着什么角色？
- 当缺乏对应用程序工作流的控制时，监察部门将如何履行职责？ 138
- 服务的某些方面失去控制将会产生哪些影响（政治、法律等）？
- 终端用户对服务的满意程度？

从组织结构的角度来看，在数据和流程的管理上缺乏控制不仅存在安全威胁，而且还会带来前所未有的新问题。以前当计算机系统出了问题时，组织制定战略和解决方案来处理，往往依靠的是企业IT人员的专业技术和知识。将IT基础设施和服务转移到云计算环境的一个主要优点是减少或完全消除相关的维护和技术支持成本。因此，云服务用户无需排除IT故障。同时，现有的IT人员需要有不同类型的能力，但一般只需要较少的技能，从而降低了他们的价值。这是云计算从组织结构的角度必须面对的挑战，这将显著改变企业不同部门之间的关系。

本章小结

本章讨论了云计算的基本特征，并介绍了云服务分类和组织结构的参考架构。为了更好地总结本章内容，我们可以回顾NIST给出的云计算定义，它概述了如下几个基本方面：

- 五个基本特征。按需自助服务、广泛的网络访问、资源池、快速灵活和计量服务。
- 三种服务模式。软件即服务（SaaS）、平台即服务（PaaS）和基础设施即服务（IaaS）。
- 四个部署模型。公共云、私有云、社区云和混合云。

采用云计算的主要驱动力在于经济和简便的软件交付和操作。云计算通过降低IT资产的资本成本并将其转化成运营成本，从而增加企业利润。基于这些原因，本章还讨论了云计算的经济和成本模型。

虽然云计算已经被工业界迅速采用，但仍然存在开放性的挑战问题，如云计算系统的管理、安全性以及社会和组织问题。在支持云计算的软件基础设施和模型方面也有显著的发展空间。

习题

1. 缩写XaaS代表什么含义？

139 2. 云计算参考模型中有哪些基本组件？
3. 基础设施即服务指的是什么？
4. 云计算 IaaS 解决方案的基本组成部分是什么？
5. 举出实现 IaaS 的例子。
6. 平台即服务方案的主要特征是什么？
7. 描述在 PaaS 市场中可选的不同类型有哪些？
8. 缩写 SaaS 是什么意思？它与云计算有什么关系？
9. 列举一些典型的软件即服务解决方案。
10. 云的不同分类有哪些？
11. 举出公共云的例子。
12. 私有云最常见的案例是什么？
13. 哪些需求是通过异构云解决的？
14. 描述云计算的经济和商业模式的基本特征。
15. 云计算是如何减少产品投入市场的时间和降低资本支出的？
140 16. 列出云计算存在的挑战。

第二部分

Mastering Cloud Computing: Foundations and Applications Programming

云应用编程与 Aneka 平台

Aneka：云应用平台

Aneka 平台是 Manjrasoft 公司推出的关于云应用开发、使用、管理的解决方案。Aneka 包括一个可在异构计算平台上部署的可扩展云中间件。它为云应用提供了一系列辅助功能来协调应用程序的执行，帮助管理者监视云中的状态，同时也提供现有云技术的整合。Aneka 的核心优势之一是针对不同编程模型（例如任务、线程、MapReduce）提供了可扩展应用编程接口（API），用于不断发展分布式应用并将新的技术整合到云应用中，同时 Aneka 还支持不同的云部署方案（公共云、私有云、混合云，如图 5-1 所示）。这些特性使 Aneka 不同于 IaaS 平台上的管理软件，成为一个可在多种云平台上进行云应用开发、部署和管理的综合平台。

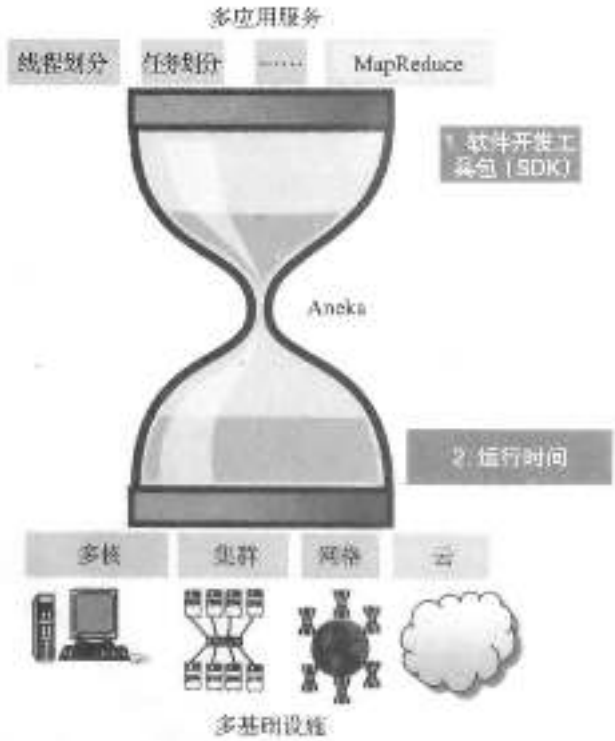


图 5-1 Aneka 功能一览

本章对 Aneka 系统框架进行了描述和总结，介绍了构成 Aneka 云平台的组件和基本服务，同时讨论了一些具有普遍性的云应用部署场景。

5.1 框架概述

Aneka 是一个开发云应用的软件平台，它可以利用分布式系统资源在一个独特的运行云应用的虚拟域——Aneka 云平台中对其进行管理。根据第 1 章所提供的云计算参考模型，

Aneka 是一个纯平台即服务的云计算解决方案。Aneka 是一个可以部署在异构资源平台上的云中间件产品，例如局域网中的计算机、多核服务器、数据中心、虚拟化的云设备或上述几种的混合。Aneka 框架不仅提供了管理和扩展分布式云应用的中间件，同时也提供了开发分布式云应用的编程接口集合。

图 5-2 对 Aneka 框架的组件进行了全面的概括性展示，系统的核心框架提供了统一的接口层，使 Aneka 框架可以部署在不同的平台和操作系统上。物理和虚拟的资源池代表了云中有限的资源，这些资源由 Aneka 容器（一种轻量级虚拟化技术）管理。Aneka 容器安装在每个节点上，构成了云中间件的基本单元。一系列互联的容器构成了 Aneka 云平台：一个向用户和开发者开放服务形式的统一域。容器提供了三种不同特性的服务形式：构造服务、基础服务和执行服务。这些服务分别负责设备管理、云支撑服务、云应用管理和运行。这些服务通过应用管理和开发接口层提供给开发者和管理人员，包括云应用开发 API 和云平台控制管理工具。

141
~
143

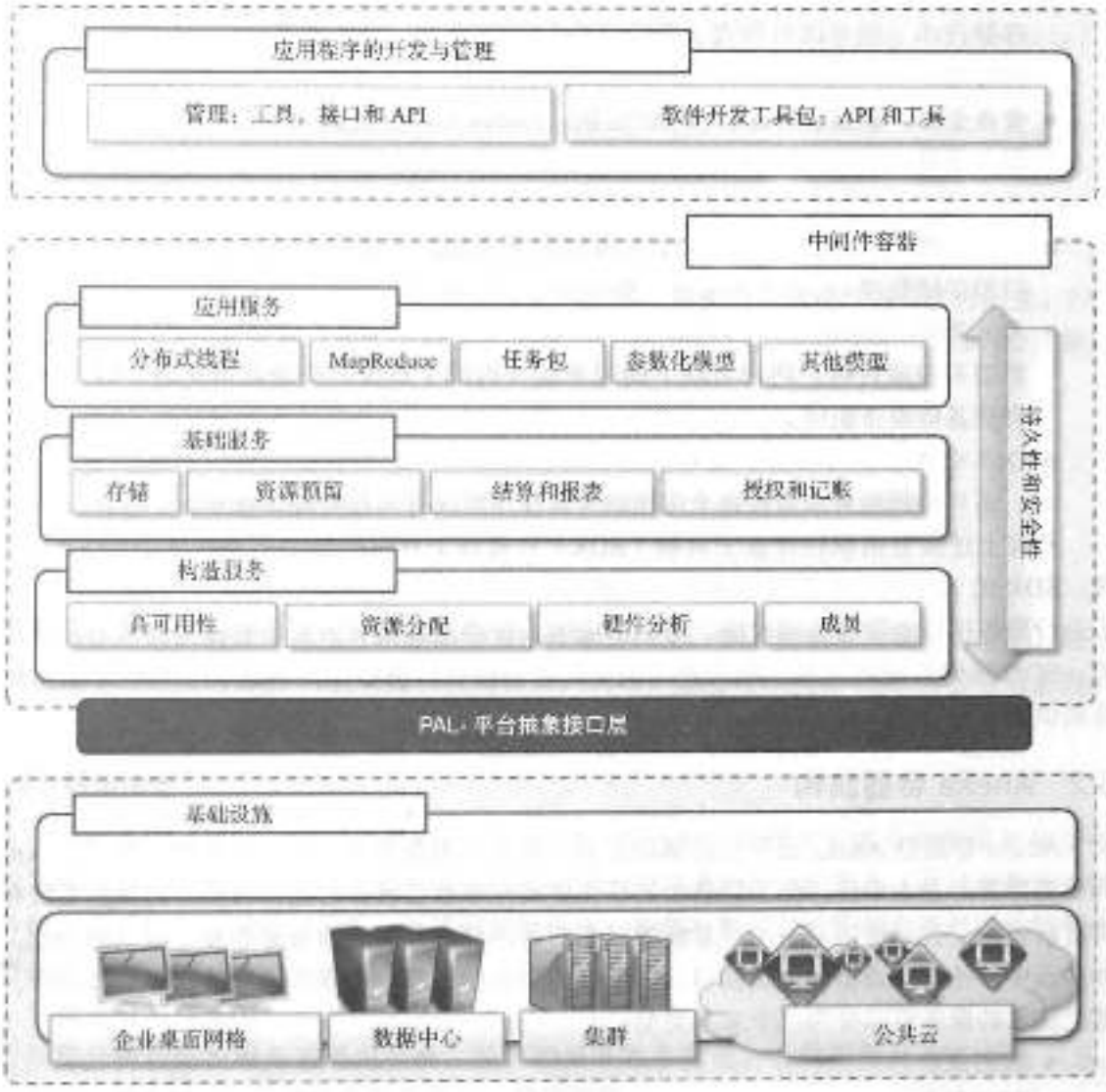


图 5-2 Aneka 框架概述

Aneka 继承了面向服务架构（SOA）的全部特性，服务是 Aneka 云平台的基本组件。服务接口建立在容器层上，框架中平台抽象层的特性对于用户、开发者和管理者是不可见的。Aneka 云平台同时提供了扩展和重新定义服务实现的服务接口，允许多种新服务的整合以及用不同的实现替代现有实现。该框架包括基础设施、节点管理、应用程序执行、计费和系统监控等基本服务，现有的服务可以进行扩展，同时新的功能可以通过动态插入容器的方式添加进云平台。容器虚拟化技术的可扩展性和灵活部署使 Aneka 云平台能够有效支持不同应用的编程模型和执行模型。这里的编程模型是指一个抽象集合，程序员可以用它来表述分布式应用。编程模型的运行支持由一个集合构成，该集合通过执行与服务相互作用来推动应用
144 的执行。因此，一个新模型的实现需要由应用开发人员和服务所使用的特定编程抽象的发展来为其提供运行支持。编程模型只是云应用管理和运行的一个方面，Aneka 云平台还提供云应用的弹性资源分配和分布式协作服务。这些服务包含以下内容：

- 弹性和可扩展性。通过动态分配服务，Aneka 支持对应用提供资源容量的增减。
- 运行管理。运行机制需要维护基础设施的可用性以支撑应用运行。Aneka 平台上要由容器技术、服务成员管理、基础设施监控和维护、程序概要分析等服务接口实现运行管理。
145 • 资源管理。根据应用需求和用户定制，Aneka 可以弹性提供设备资源。为了提供基于服务质量（QoS）的运行环境，系统不仅允许动态资源备份也允许为某些应用保留独占节点。
- 应用管理。应用管理对应一个专属的服务子集，子集中的服务包括调度、执行、监控和存储管理。
- 用户管理。Aneka 是一个多租户的分布式环境，提供一个扩展的用户系统来定义用户、群组和权限许可。用户管理下的服务接口构成了系统的安全基础机制，以及计算管理的基础服务组件。
- QoS/SLA 管理和结算。云环境将会对执行的应用进行计量和资费结算。Aneka 提供一系列协同服务来监控每个应用的资源使用情况并向用户提供账单。

所有上述服务由软件开发工具箱（SDK）和管理工具箱提供的应用程序接口（API）实现。SDK 的 API 向开发者提供现有编程模型和对象模型来创建新的模型。管理工具箱集成了运行服务接口来管理基础设施、用户和应用。管理工具箱对云平台进行整体状态描述和监控，SDK 则更多地提供对于单个应用执行的服务接口。两组工具箱致力于为核心虚拟化容器提供交互和管理的易用接口。

5.2　Aneka 容器结构

Aneka 容器是 Aneka 云平台的基础组件，代表了服务及应用的运行机制。容器是 Aneka 的资源部署的基本单位，作为轻量级的软件中间件来承载服务功能，与接入的操作系统和硬件进行交互。容器提供了一个轻量级的服务部署环境，可用于部署服务和一些基础功能，比如和云中其他节点通信的信道接口。云平台的所有操作都由容器所承载的服务接口提供。容器上加载的服务可以分为三个基本类型：

- 构造服务。
- 基础服务。
- 应用服务。

上述服务构成的服务栈位于平台抽象层（PAL）之上，提供了对潜在操作系统和硬件的接口，从容器的角度提供了软硬件资源。持久性和安全性横跨了所有服务栈的内容，提供了一个安全可靠的框架结构。下面章节将具体讨论这些服务层次的细节。 146

5.2.1 Aneka 平台基础：平台抽象层

系统的核心架构基于 .NET 技术，使 Aneka 能够在不同平台和操作系统上迁移。任何满足 ECMA-334[52] 和 ECMA-335[53] 规范的平台环境均可在 Aneka 容器上加载和运行。

ECMA-334 标准引入了通用语言基础设施（CLI），定义了一个通用代码运行环境，CLI 不会向运行在其框架上的应用程序提供任何接口来访问硬件或收集宿主操作系统的运行信息[⊖]。此外，不同的操作系统有不同的文件系统结构和存储方式。平台抽象层（PAL）可以屏蔽这种异构性，向容器提供统一接口来访问硬件和操作系统信息，因此 Aneka 容器可以在任何 PAL 所支持的平台上运行。

PAL 负责检测宿主环境的兼容性并实现与宿主环境的交互，从而支持上层容器的运行。PAL 具有如下特征：

- 提供统一且与平台无关的执行接口来访问宿主平台。
- 对宿主平台的扩展和附加属性提供统一访问途径。
- 提供统一且与平台无关的途径来访问远端节点。
- 提供统一且与平台无关的管理接口。

PAL 是 Aneka 中一个小的接口层构成的检测引擎，启动后会自动对容器进行配置。PAL 实现了具体硬件资源的接入和对不同操作系统（Windows、Linux 和 Mac OS X）的抽象层构建。

PAL 向上层提供的信息包括：

- 内核数量、频率和 CPU 用途。
- 内存的大小和使用情况。
- 全局层面的硬盘可用空间。
- 网络地址和节点上的设备。

此外，通过查询硬件属性可以获得附加硬件配置信息。PAL 提供定制实现的方法，通过使用表述平台信息的命名 - 值来提取附加信息。例如，属性可以包含 CPU 的型号和系列信息，以及容器所在的进程信息。

5.2.2 构造服务

构造服务定义了 Aneka 容器最底层的软件栈。构造服务提供了资源分配子系统和设备监控系统的接口。资源提供服务负责利用虚拟化技术按需求动态分配节点。监控服务可以对硬件信息进行分析，同时，作为一个基础监控设施，安装在容器中的任何应用都可以使用这一服务。 147

1. 概要分析和监控

概要分析和监控服务主要由心跳、监控和报告服务构成。心跳服务从 PAL 收集信息，监控和报告服务对云中任意服务行为实行一般性监控。

⊖ .NET 虚拟机 CLR 是 CLI 的一种实现。——译者注

心跳服务周期性地从节点收集动态运行信息并将这些信息发布给云中的成员服务。这些信息由云的索引节点收集，为资源预留和调度服务提供必要依据，实现异构基础设备的优化应用。如前文所述，除了收集内存、硬盘剩余信息、CPU 和操作系统等基础信息以外，一些附加信息（例如系统中安装的软件）也可以加入到动态报文信息中。更确切地说，任意以文本描述的属性信息都可以在报文中传递。一个名为 node resolver 的服务组件负责收集上述运行信息并传递给心跳服务。Aneka 为该服务组件提供了多种方法以满足不同异构环境下的实现。PAL 使用不同接口来对接不同操作系统，部署在不同节点上的 Aneka 组件可以屏蔽操作系统的异构性，以获取类型一致的信息。例如，在物理机和 IaaS 服务商（EC2、GoGrid）提供的虚拟机上获取公共 IP 的机制是不同的。Aneka 在虚拟容器部署中为每个节点配置一个 resolver 服务组件，以保证系统其他组件工作的透明性。

系统集成的监控和概要分析服务通过一个通用监控平台实现，该平台允许任何用户服务报告其行为。该平台由报告和监控服务接口构成，报告服务负责监控数据的存储并将监控数据提供给其他服务接口或者分析应用。每个节点上会实例化一个监控服务作为报告服务的网关，向报告服务传送采集节点上的监控数据。任何想要提供监控信息的服务可以借助本地的监控服务接口而不必了解整个基础设施的细节。目前已经有以下集成服务通过这种途径来提供相关信息：

- 成员目录记录节点运行信息。
- 执行服务在任务执行过程中选取时段进行监控。
- 调度服务记录任务的状态信息。
- 存储服务监控和提供数据迁移中的信息，例如上传和下载次数、文件名和大小。
- 资源分配服务记录虚拟节点的分配和生命周期信息。

上述信息可以存储在一个关系型数据库管理系统（RDBMS）或平文件[⊖]中，以便应用
148 进行深入分析。例如，管理控制台提供了管理层面的数据监控信息。

2. 资源管理

资源管理是 Aneka 云平台的另一项基本特征，它需要完成以下几个任务：资源组成、资源预留和资源分配。Aneka 提供了一系列服务接口负责资源管理，包括索引服务（即成员目录）、预留服务和资源分配服务。

成员目录是 Aneka 资源管理的基本组件，它记录了所有节点（无论连接与否）的基本信息。成员目录构成了目录服务的一个基础服务，允许通过节点名等属性字段来搜索服务。当容器启动时，每个容器实例都会将自身信息发布到成员目录并在生命周期内不断更新。服务和外部应用可以查询成员目录以发现可用容器并与其进行交互。为了加速查询，成员目录设计成一个分布式数据库：查询先在本地记录的缓存中进行，如果没有结果再向主索引节点查询，主索引节点包含全局的成员目录。成员目录同时保存每个节点的动态信息，这些信息被发送到本地监控服务实例中并被长期保存。

资源的索引和划分是资源管理的基本功能，这些功能是在基础索引服务的接口上实现的。容器实例的部署和配置在设备管理层实现，并不属于构造服务。

动态资源分配允许将从 IaaS 服务商处租用的虚拟资源整合到 Aneka 云平台中并进行管理，该服务改变了 Aneka 云平台中的拓扑结构，允许其根据不同需求（例如处理节点故障、

⊖ 数据库的无结构化的普通文件存储形式。——译者注

保证应用的服务质量、维持云的稳定性能或恒定吞吐量）来进行扩展部署。Aneka 定义了一个非常灵活的框架来实现资源分配，其中资源动态申请的触发机制、添加容器数量的选择以及添加容器后的新运行策略都是可以修改的。Aneka 集成的资源分配平台主要集中在资源分配服务，该服务提供了分配虚拟容器实例资源的所有操作。资源分配服务的实现是基于资源池的设计理念，资源池提供了一个通用抽象接口来实现与具体 IaaS 服务商资源的交互，使所有服务商的资源池可以被统一管理。资源池并不一定要映射到 IaaS 服务商，也可以将由 Xen Hypervisor 管理的私有云或者只是偶尔使用的物理机资源作为动态资源。系统基于开放协议，允许使用元数据来描述资源池和定制资源分配请求。该平台简化了附加特征的实现，以及对能够被集成到现有系统的不同实现方法的支持。

资源分配功能是为了支持面向 Qos 服务需求的应用执行。因此，资源分配服务多数是从预留服务和调度服务发送的请求。此外，外部应用可以直接利用 Aneka 的资源分配功能，动态检索客户端，并与相关基础设施进行交互。这使得资源分配功能不仅可被 Aneka 使用，也可以被虚拟机管理器使用。 149

5.2.3 基础服务

构造服务是 Aneka 云平台的基础服务，定义了系统管理平台的基础功能。基础服务主要用于对构建在基础设施上的分布式系统进行逻辑管理，并为分布式应用的执行提供支持。所有支持的编程模型都可以集成进服务接口，同时利用这些服务接口提供高级且全面的应用管理。这些服务包括：

- 应用的存储管理。
- 资源的定价、支付和结算。
- 资源预留。

基础服务提供统一途径来管理分布式应用，使开发者只需关心编程模型的逻辑差异。结合构造服务和基础服务构成了 Aneka 中间件的核心。这些服务接口主要由执行服务和管理平台所使用。外部应用也可使用这些开放接口来提供高级应用管理。

1. 存储管理

数据管理是分布式系统和云环境中的重要方面。应用对数据的操作多数以文件存储和移动的形式进行。因此，任何支持分布式应用的平台应该提供数据 / 文件迁移管理和持久化存储的服务。Aneka 提供了两组存储管理的服务模式：集中式文件存储，主要适用于计算密集型应用；分布式文件系统，适合于数据密集型应用。两种应用的需求截然不同。计算密集型服务主要是需要高性能处理器，对存储没有太高要求，在许多实用案例中仅需要存储小型文件，文件在节点间的迁移也十分方便。在这种情况下，设置一个中心存储节点或者存储节点集群是较为合适的解决方案。相反，数据密集型应用的特点是处理大型数据文件（GB 或 TB），任务对处理能力的需求并不构成性能瓶颈。在这种情景下，利用分布式系统来管理所有云节点的存储空间是一个更好且更具可扩展性的解决方案。

集中式存储通过 Aneka 的存储管理服务接口来实现，该服务构成了 Aneka 的数据分级服务。存储服务向分布式应用提供了基本的文件传输功能以及面向终端用户或其他系统组件的抽象协议接口，接口可由安装到云中的服务实例动态配置（工厂模式）。目前默认的协议配置是 FTP。

为了支持不同的协议，系统引入文件通道的概念并定义了两个组件：文件通道控制器和

文件通道句柄。文件通道控制器构成了通道的服务组件，包含文件存储和访问权限。文件
150 通道句柄代表了顾客组件，用户应用和系统其他组件可以调用句柄实现上传、下载和浏览文件。存储服务利用工厂模式下的文件通道首先创建管理存储的服务组件，然后按需求创建用户组件。需要文件传输支持的用户应用会自动分配文件通道句柄，实现输入文件的上传和外部文件的下载。同时，管理平台将会对工作节点进行配置，使节点能够完成执行文件的取回和结果上传。

文件通道抽象的一个令人感兴趣的特征是，链接两个文件通道以实现通过两种不同的文件协议来传输文件。Aneka 的每个文件中都包含元数据，可帮助管理平台选择合适的通道来移动文件。例如，一个输出文件可以利用内部 FTP 协议从工作节点传递给服务实例，再通过服务实例管理的 FTP 通道控制器输出到 AWS S3 服务的桶下。存储服务接口支持基于任务划分和线程模型的编程以及基于参数化的应用。

数据密集型的应用采用分布式文件系统进行存储，其参考模型是谷歌文件系统（GFS）[54]，具有基于商用硬件的高度可扩展的基础设施。文件系统架构基于主节点，主节点中包含用于跟踪所有存储节点状态的文件系统全局映射，以及用于提供分布式存储空间的数据块服务器资源池。文件逻辑上存放在一个目录结构中，但同时通过一个唯一的 ID 产生分结构化域名，并以此作为索引存放在文件系统中。每个文件由一系列相同大小的块构成，每个文件块分配唯一的 ID 且存放在不同的服务器上，并进行副本存储来提供高可用性和容错性。该模型由 GFS 提出并能够对具有以下特征的应用提供优化支持：

- GB 级别以上的文件。
- 修改文件时只需要增加文件内容而不需要对原有文件内容进行修改。
- 以大量读取访问和少量随机访问为主的负载类型。
- 持续稳定的带宽比低响应延迟更为重要。

此外，考虑到文件系统中有大量的商用机器共同工作，因此进程故障和硬件故障是一种常态化现象。这些现象影响了存储机制的设计，使系统能够为应用的数据操作提供更好的性能。目前，只有 MapReduce[55] 能够使用分布式文件操作系统，MapReduce 也是 GFS 的主要应用。Aneka 在 Windows 文件系统基础上提供了一个简单的分布式文件系统。

2. 记账、结算和资源定价

记账服务记录了云中应用的状态，收集的信息对分布式节点的使用情况做出了细致划分，对资源管理的优化至关重要。

151 记账服务所收集的信息主要与设备使用和应用执行有关。应用执行过程中的完整历史信息以及存储等资源的使用参数将会被记录服务抓取和分析。这些信息构成了用户计费的基础信息。

结算是记账服务的另一个重要功能。Aneka 是一个多用户云编程平台，云中应用的执行可能直接涉及从 IaaS 服务商处调度更多资源，并产生协作成本。Aneka 结算服务为每个用户提供资源使用及其相关费用的详细信息。每项资源可以根据服务接口的类型或者节点上安装的软件来定价。结算模型提供了对各项应用的综合结算，对于具体用户提供每项任务的结算信息。

记账和结算功能由记账服务和报告服务提供。前者记录了应用的执行信息，例如任务在可用资源上的分配，各项任务时序及协作成本。后者提供了从监控服务接口获取的用于结算的信息，如存储使用情况和 CPU 性能。这些信息主要由管理控制台提取。

3. 资源预留

资源预留支持分布式应用的执行，使某些应用可以独占预留资源。资源预留由两种服务接口实现：资源预留服务和部署服务。资源预留服务记录了所有的预留时隙并提供系统的全局视角。部署服务安装在每个执行服务的节点上，管理本地节点的时隙分配。有截止时间的应用可以在一个时间段内通过资源预留申请一系列节点资源。如果预留请求可以满足，预留服务实例将会返回一个标识符作为资源预定凭据。应用执行期间，该标示符将会被用来选择执行应用的预留节点。在每个预留节点上，执行服务使用分配服务接口通过检验分配标示符来判断每项任务在执行时隙内是否具有占用许可权。尽管这是一个资源预留框架下的通用参考模型，但 Aneka 允许不同的服务实现方式，包括利用多种资源预留协议，以及在预留请求中配置相关参数。不同的协议和策略可以透明化地进行集成。Aneka 提供扩展 API 来支持高级服务。目前，系统框架支持三种不同的实现：

- 基本预留。代表节点时隙预留的基本方法，执行双备份协议，提供预留资源的备份选项以避免初始的预留请求不能被满足。
- 权重（计价）预留。根据节点的硬件资源进行节点定价并提供预留。
- 中继预留。实现简单，可以使资源代理商在 Aneka 中预留节点并控制节点资源，该 152
接口在互联云环境下的集成交互中比较有用。

资源预留是保证应用服务质量（预先经过协商）的基本功能，它可以保证云平台提供一个可预见的环境来控制应用的执行。预留请求允许与否的判断机制是基于请求时刻的可用资源静态分布，还要考虑目前和未来的工作负载。该方案对节点故障十分敏感，可能会使云平台无法满足服务等级协议 SLA。实际的服务应用机制倾向于尽可能推迟为预留请求分配节点，以应对临时故障和局部断电。但是对于大范围断电导致剩余节点不足以负载需求的情况，该策略将无法应对。为了应对这种情况，资源调度服务提供了一种更有效的解决途径：可以由外部资源服务商提供调度节点来应对断电和满足应用的 SLA 需求。当系统中的现有资源不足以满足预留请求时，现有的资源预留机制会使用构造层的调度方法来满足预留需求，解决资源不足和临时故障的问题。

5.2.4 应用服务

应用服务对应用的执行进行管理，并针对不同的分布式负载应用对应的编程模型提供相应接口。由于每个编程模型的需求和特征不同，所以提供的服务接口的种类和数量也不同。在所有的支持模型中，可以划分出两类共性的类型：调度服务和执行服务。Aneka 定义了一个支持编程模型运行的参考模型，并抽象出调度和执行两个服务接口集合。此外，Aneka 还定义了基础接口来实现对新的编程模型的扩展支持。

1. 调度

调度服务负责规划分布式应用的执行并将应用所包含的任务分配给节点。调度服务同时构成了几个其他基础服务和构造服务子集的调用结合点，例如资源分配、预留、记录、汇报。调度服务组件的共同任务如下：

- 节点映射任务。
- 失败任务的重新调度。
- 任务状态监控。
- 应用状态监控。 153

Aneka 并不提供集中式的调度引擎，每个编程模型拥有自己的调度服务接口，以便和中间件中的其他服务进行协作。如上文所述，协作服务主要属于图 5-2 所示框架中构造和集成服务接口层。为不同的编程模型设计不同的调度引擎给调度和资源分配机制的设计带来了很大的灵活性，但同时也需要对共享资源的使用进行精确设计。在调度过程中，对以下情况需要进行适当管理：多任务同时发送给同一节点，没有预留申请的任务发送给预留节点，接受任务的节点没有安装预先所需的服务内容。Aneka 基础服务提供了有效的信息收集接口来避免上述情况，但是应用运行平台并不具有检测和纠正上述情况的能力。Aneka 目前的设计机制是使每个调度引擎完全相互隔离，旨在需要时可以使用其他服务接口，以此来保证任意时刻在每个节点上对每个编程模型只有一个任务在执行，但应用的运行并不是互斥的，除非使用资源预留机制。

2. 执行

执行服务控制应用中单个任务的执行，负责设定执行任务的运行环境。调度服务实例运行时，每个编程模型都有自身需求，但在所有的编程模型中可以提取出一些共性操作：

- 从调度器获取任务后的分发。
- 取回所需的输入文件。
- 任务的沙盒运行。
- 任务执行完毕后输出文件的提交。
- 执行故障管理（例如抓取有效的上下文信息进行故障鉴别）。
- 性能监控。
- 任务打包并回传给调度器。

相对于调度服务，执行服务构成了一个自成体系的单元。执行服务需要处理的信息较少并且只需要和存储服务、本地部署和监控服务接口交互。Aneka 为执行服务提供了集成上述所有服务的关联接口集合，目前已有两个编程模型实现了专门的关联接口集合。

应用服务构成了对云中编程模型的运行支持，目前所支持的编程模型如下：

- 任务模型。包含很多任务划分应用和计算应用。应用被划分为相互独立的任务的集合，任务的执行可以按任意排序进行。
- 线程模型。对分布式框架下的经典多线程编程模型进行扩展支持，使用线程抽象模型来装载远端执行的程序模块。

154

- MapReduce 模型。MapReduce 在 Aneka 上的实现。
- 参数化模型。任务模型应用的一类实例，将多组参数集合输入任务实例，每组参数代表了选定域中一个具体的点。

其他编程模型正处于开发和测试阶段，其中包括数据流模型 [56]，信息传递接口和 Actor 模型。

5.3　构建 Aneka 云平台

Aneka 是一个开发分布式云应用的平台。作为一个软件平台，它需要部署在相应的硬件设施上，同时硬件设施需要相应的管理组件。构建云平台是云管理者的主要任务之一，Aneka 支持公共云、私有云和混合云的多种部署方式。

5.3.1 基础设施组织

图 5-3 从基础设施的角度描述了 Aneka 云平台，为不同的部署模式提供了一个参考模型。管理控制台作为核心角色执行所有需要的管理操作。云部署的一个基本元件是仓库集合，仓库存储了安装和部署云平台基本服务所需的所有库，这些库构成了用于节点管理和容器程序的软件镜像。仓库可以通过传输协议通道（HTTP、FTP、通用文件共享等）来共享库。管理平台可以管理多个仓库并为具体的部署选择一个合适的仓库。设备部署的过程是为一系列节点安装节点管理器，也称为守护进程。守护进程是远程管理服务实例的一部分，以实现容器实例的部署和管理。容器的集合构成了 Aneka 云系统。

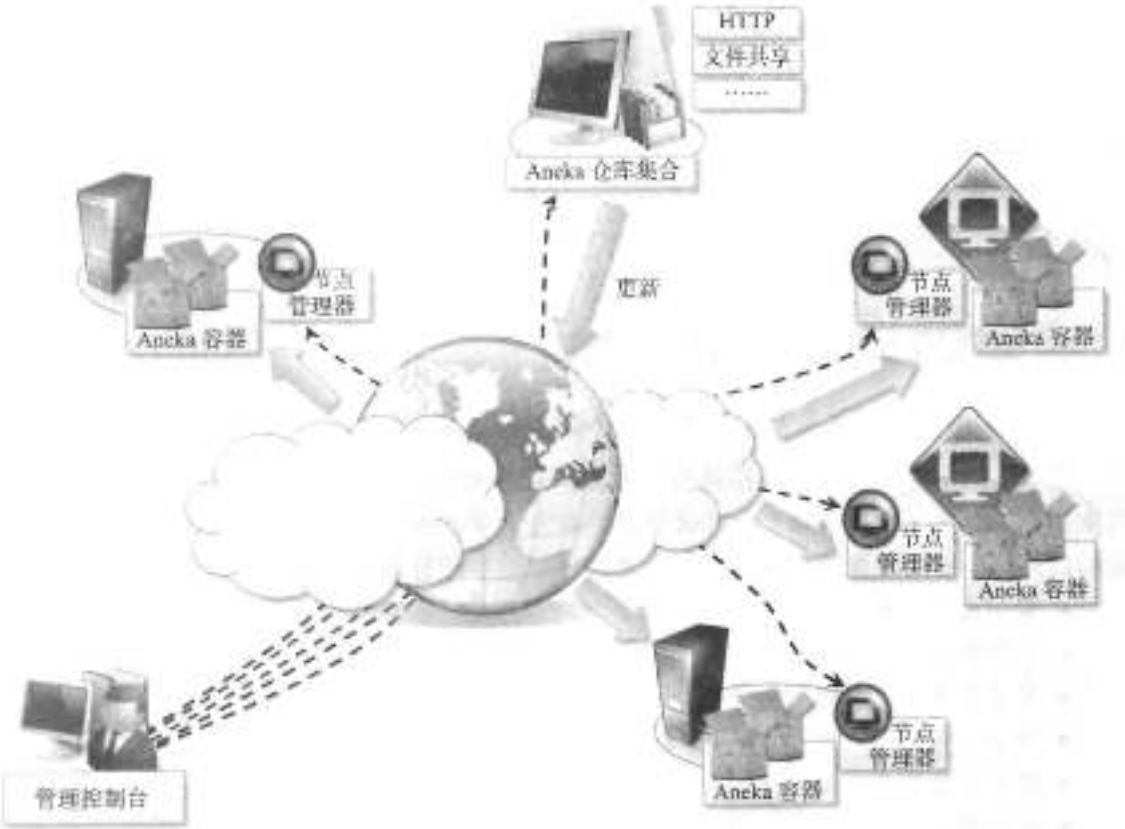


图 5-3 Aneka 云平台基础设施概览

从基础设施的角度而言，只要可以通过网络连接和远程管理接入节点，那么对物理节点和虚拟节点的管理就是一致的。而在虚拟化实例的动态分配过程中，容器由包含 Aneka 服务组件的打包资源镜像所创建。动态分配只需要将资源镜像配置给实例，以将实例加入到云环境中。容器和守护进程的安装也可以采取简化模式，这主要取决于虚拟资源的生命周期。

5.3.2 逻辑组织

云的逻辑组织非常多样化，主要取决于容器实例上的配置选择。常见的模式是采用主从 155
配置，并另外划分存储节点，如图 5-4 所示。

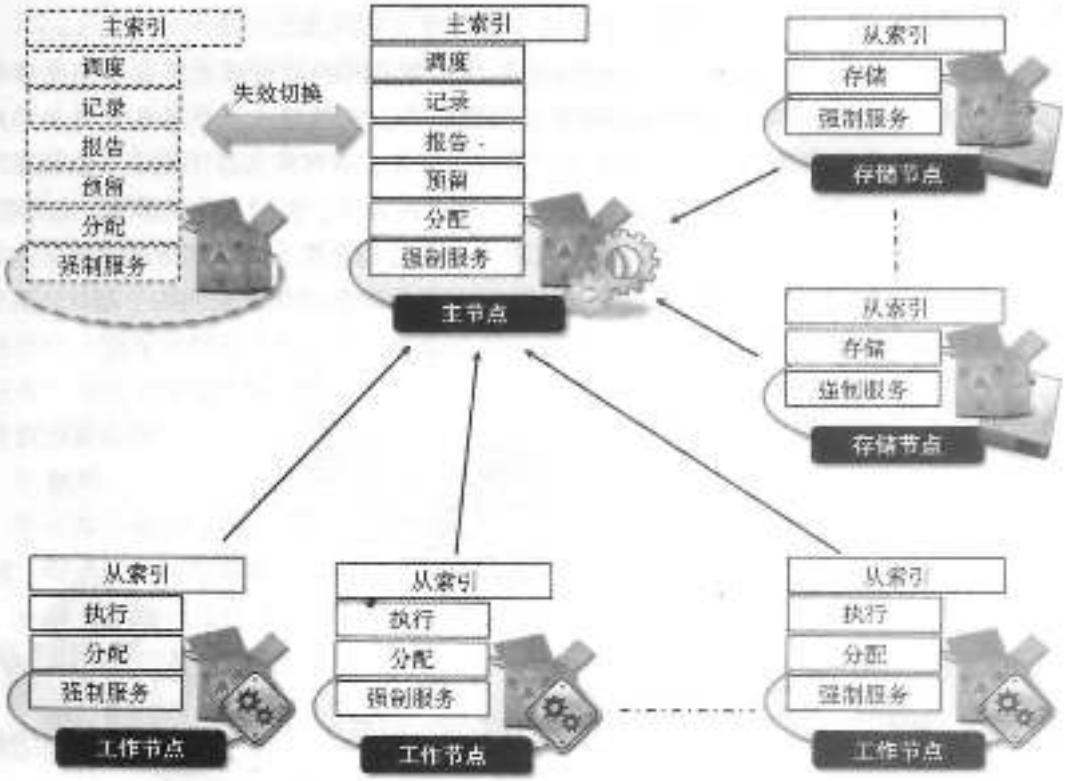


图 5-4　Aneka 云逻辑组织

主节点的服务通常具有一份拷贝，并提供对云内的智能管理。将一个节点设置为主节点需要在索引服务（成员目录）中将该节点配置为主节点模式。多拷贝的服务除了需要强制部署的以外，一般应配置到从节点。主节点的常规服务配置如下：

- 索引服务（主拷贝）。
- 心跳服务。
- 日志服务。
- 预留服务。
- 资源配置服务。
- 记录服务。
- 报告和监控服务。
- 编程模型对应的调度服务。

156

主节点通过一个关系型数据库（RDBMS）访问设备的运行状态数据。实际上，所有的调度服务都在主节点内完成。调度服务接口共享关系型数据库中的应用模块，以提供一种容错机制。主节点的配置可以被保存在几个其他节点上，通过故障转移机制提高系统的可靠性。

工作节点构成了 Aneka 云的处理单元，主要是为支持应用执行而配置相关服务，包括强制部署服务和具体编程模型的执行服务。常规配置如下：

- 索引服务。
- 心跳服务。
- 日志服务。

- 部署服务。
- 监控服务。
- 编程模型对应的执行服务。

157

通过选择不同的执行服务来使用不同的工作节点，可以实现负载均衡以及为一些具体应用保留相关节点。

存储节点为应用提供了优化存储支持，在强制服务和常规服务以外提供了存储服务接口。存储节点的数量取决于对工作负载的预测和应用对存储资源的消耗。存储节点主要配置在有足够硬盘空间可以容纳大量文件的设备上。存储节点的常规配置如下：

- 索引服务。
- 心跳服务。
- 日志服务。
- 监控服务。
- 存储服务。

实际使用中，如果没有数据传输需求，存储节点可能只有一个。在小型部署案例中，没有必要设置一个独立的存储节点，因此存储服务可能被安装和运行在主节点上。

所有节点都在主节点中进行注册，并且可以透明地关联到所有故障转移备用设备上，以保证设备的高可靠性。

5.3.3 私有云部署模式

私有云部署模式主要由本地物理机资源和提供资源池节点（多数情况下经过了虚拟化）访问的设备管理软件构成。这种情况下，Aneka 云需要在异构设备（例如桌面设备、集群和工作站）的资源池上构建。这些异构设备可以被划分为不同的组，云平台可以根据应用的需求来使用这些资源。此外，通过使用资源配置接口，可以从一些开源分布式虚拟化系统（如 XenServer、Eucalyptus 和 OpenStack）中获取虚拟节点资源。

图 5-5 显示了私有 Aneka 云的常规部署。在系统负载可以预测且本地虚拟机管理器可以处理超出预测负载的资源请求时，这种部署是可行的。大多数 Aneka 云中的节点由物理机

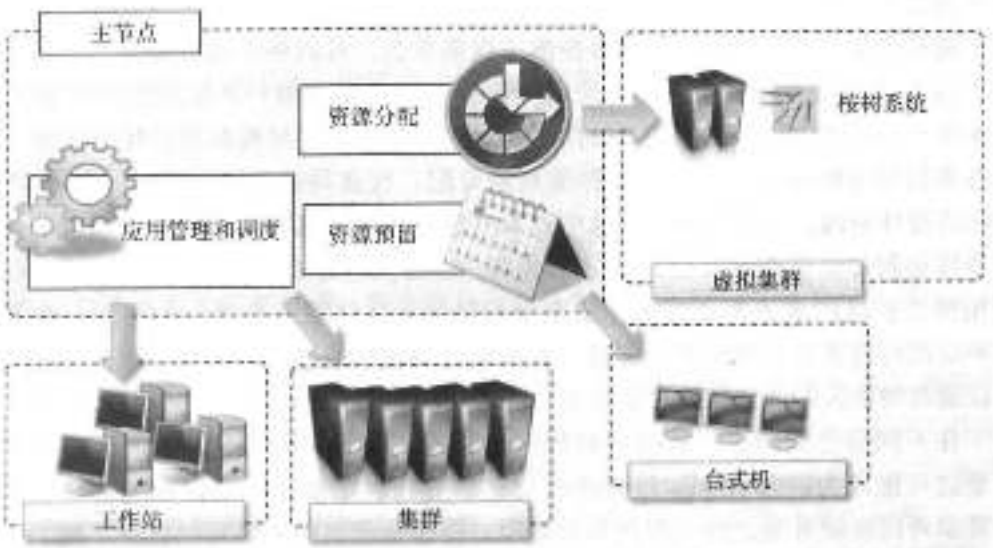


图 5-5 私有云部署

构成，具有很长的生命周期和静态配置并且不需要经常进行重新配置。针对私有云环境中物理机的不同属性，可以通过资源预留服务接口制定相应的资源管理和使用策略。例如，需要在白天进行办公自动化工作的台式机可以在标准工时以外执行分布式任务。工作站集群可能具有一些支持应用执行的既有软件，因此更适合有专属需求的应用。

5.3.4　公共云部署模式

158 在公共云部署模式下，Aneka 的主节点和工作节点完全部署在由一个或多个 IaaS 服务商（如亚马逊 EC2、GoGrid）提供的虚拟化设备上。所有节点都被事先提供以便静态部署。这种部署方式与在没有动态资源扩展能力的物理设备上安装 Aneka 是一致的。有趣的是，由于 IaaS 服务商的弹性服务特征，云的创建是完全动态化的。图 5-6 显示了这一场景。

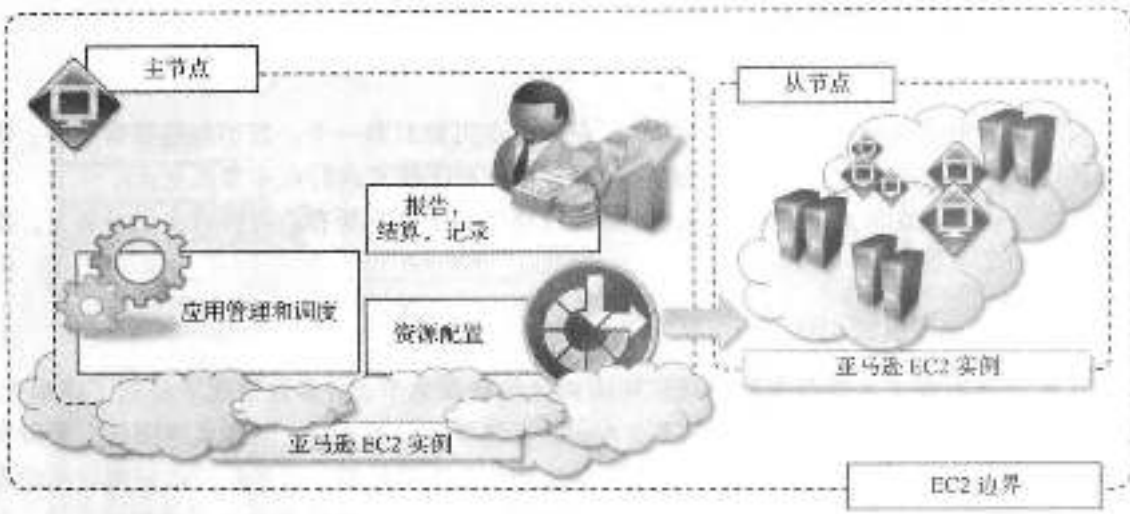


图 5-6　公共云部署

图 5-6 所示的部署是在单一 IaaS 服务商所提供的设备环境下进行的，目的在于避免不同服务商之间的数据迁移。服务商之间的数据迁移成本很高，同时部署在一个服务商的环境中网络性能更好。公共云环境下 Aneka 可以部署在单一节点上，完全使用动态分配接口按需求弹性调整设备规模。资源分配服务扮演了重要角色，可以将不同的镜像和模板配置给实例。主节点中必须包含的服务还有记录和报告服务。它们向用户和应用提供资源的使用细节，在多用户云环境中起着十分基础的作用，而用户依据它们对资源的消耗情况进行结算。

动态实例的分配将主要应用于工作节点的分配，在亚马逊 EC2 服务中，不同的镜像包含了不同的硬件资源，工作容器可以从中选取并进行配置。具有计算或存储需求的应用将会向调度器传递附加信息来触发合适的分配请求。应用执行并不是唯一使用动态实例的情况，任何使用弹性扩展的服务都会使用到动态分配机制。存储服务是另一个实例，在多用户云
159 中，多种应用可能需要存储服务的支持，有可能造成瓶颈或达到存储节点的配额限制。动态分配可以通过增加云的计算能力来解决这一问题。

由于在不同服务商间进行数据迁移的巨大成本，目前在不同服务商上部署云还不太可能，但是这可能成为联盟 Aneka 云的蓝图 [58]。在这种场景下，通过相应协议和更优惠的价格，资源可以在服务商之间实现共享和租用。资源分配服务策略可以区分不同的资源供应商，通过对不同的 IaaS 服务商的映射提供需求分配的最佳解决方式。

5.3.5　混合云部署模式

混合部署模式是 Aneka 最常见的部署模式，在很多实例中都需要一个易用的计算框架来处理应用的计算需求。该框架可以是一个 Aneka 部署的可按需求动态扩展的框架。这种部署如图 5-7 所示。

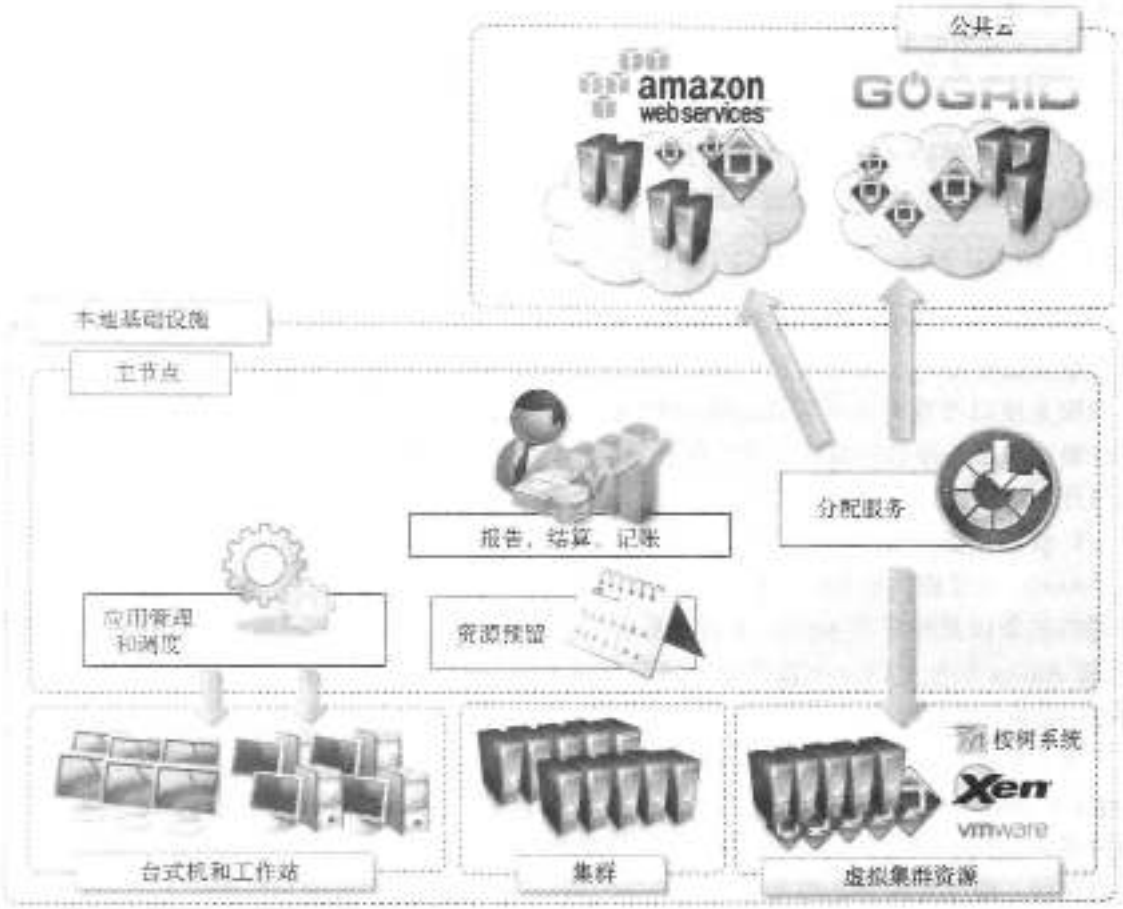


图 5-7　混合云部署

这种部署是最完整的模式，能够使用 Aneka 框架的所有功能，包括：

- 动态资源分配。
- 资源预留。
- 工作负载划分。
- 记录、监控和报告。

160

此外，如果在本地能够管理一些虚拟机资源，则可能实现对资源的充分利用，从而降低应用执行的费用。

在混合部署场景下，可以为不同用途提供不同的异构资源。正如私有云部署中所介绍的，桌面设备可以在办公时间以外预留给低优先级的工作负载，大多数应用在工作站和集群中运行，这些节点始终和 Aneka 云相连接。任何增加的计算能力需求都可以首先被本地虚拟化设备所处理，如果需要更多的计算能力，可以使用外部 IaaS 服务商的资源。

混合云部署与公共云部署的区别是利用了多元化的资源供应商来提供虚拟资源。由于部

分设施是本地的，数据迁移到外部 IaaS 服务商的设备上产生的花费是不可避免的。因此在
161 处理应用请求时要选择合适的方案。Aneka 中的资源分配服务提供了同时使用若干资源池以及设置最佳资源池配置策略的能力。上述特性可以简化用户策略的开发，更好地满足混合部署的需求。

5.4 云编程和云管理

Aneka 的初衷是提供一个执行分布式应用的可扩展中间件产品。对开发者和管理者而言，应用开发和管理是 Aneka 最主要的产品特性。为了简化上述过程，Aneka 向开发者提供了全面的可扩展的 API 集合，向管理者提供了有力的和直观的管理工具。开发者的 API 主要集中在 Aneka SDK 中，管理工具主要通过管理控制台提供。

5.4.1 Aneka SDK

Aneka 提供 API，用于基于现有编程模型开发应用、实现新的编程模型以及开发新的云平台集成服务接口。应用开发主要关注应用现有特性和使用中间件服务接口，而新的编程模型和服务接口开发将会丰富 Aneka 的特性。SDK 通过应用模型和服务模型的方式来提供对编程模型和服务接口的编程支持。前者与应用开发和新编程模型开发有关，后者则通过服务接口开发定义通用框架。

1. 应用模型

Aneka 通过提供抽象编程模型实现对云中分布式执行的支持。编程模型定义了开发者所使用的抽象以及程序在 Aneka 上运行所需的运行支持。应用模型表示适用于所有编程模型（代表 Aneka 平台上的分布式任务）的最小化 API 集合。根据需求和每个编程模型的特性，可进一步定制应用模型。

图 5-8 显示了编程模型所定义的组件，Aneka 平台运行的每个分布式应用都是 ApplicationBase<M> 类的实例，M 定义了应用管理器的类型。应用类构成了开发者视角下的分布式应用，而应用管理器是类的内部成员，用来和 Aneka 云进行交互以监视和控制应用的执行。应用管理器是编程模型实例中的第一个元素，并根据编程模型的不同选择来不对同的管理器进行配置。

无论使用那种编程模型，分布式应用都可以在接收端被分解为一系列任务。Aneka 将应
162 用进一步划分为两类：基于用户任务的应用和基于运行平台任务的应用。这两类应用对应了不同的应用基类以及不同的应用管理实现。

第一类应用最为常见并关联了几种编程模型：任务模型、线程模型以及参数化模型。该类应用由一系列用户提交的工作单元组成，每个单元都是 WorkUnit 类的子类。每个工作单元具有输入和输出文件，文件的传输由运行环境进行透明管理。工作单元的具体子类取决于编程模型（任务模型对应 AnekaTask class，线程模型对应 AnekaThread Class）。用户任务的应用都是 AnekaApplication（W, M）的子类或实例。W 表示 WorkUnit 类的子类（多态性），M 表示 IManalApplicationManager 接口的实现类。

第二类应用包含了 MapReduce 以及其他基于运行平台的应用，这种情况下工作单元没有通用的类型，应用开发者所使用的具体类型取决于编程模型的需要。例如在 MapReduce 模型中，开发者用两种函数表示它们的分布式应用，map 和 reduce。因此，MapReduceApplication 类为 Mapper<K,V> 和 Reducer<K,V>，提供相应接口以及应用所需的输入文件。其他编程

模型可能具有不同的需求和接口。因此，除了ApplicationBase<M>（M表示IAutoApplicationManager），这一类别中没有其他通用基类。

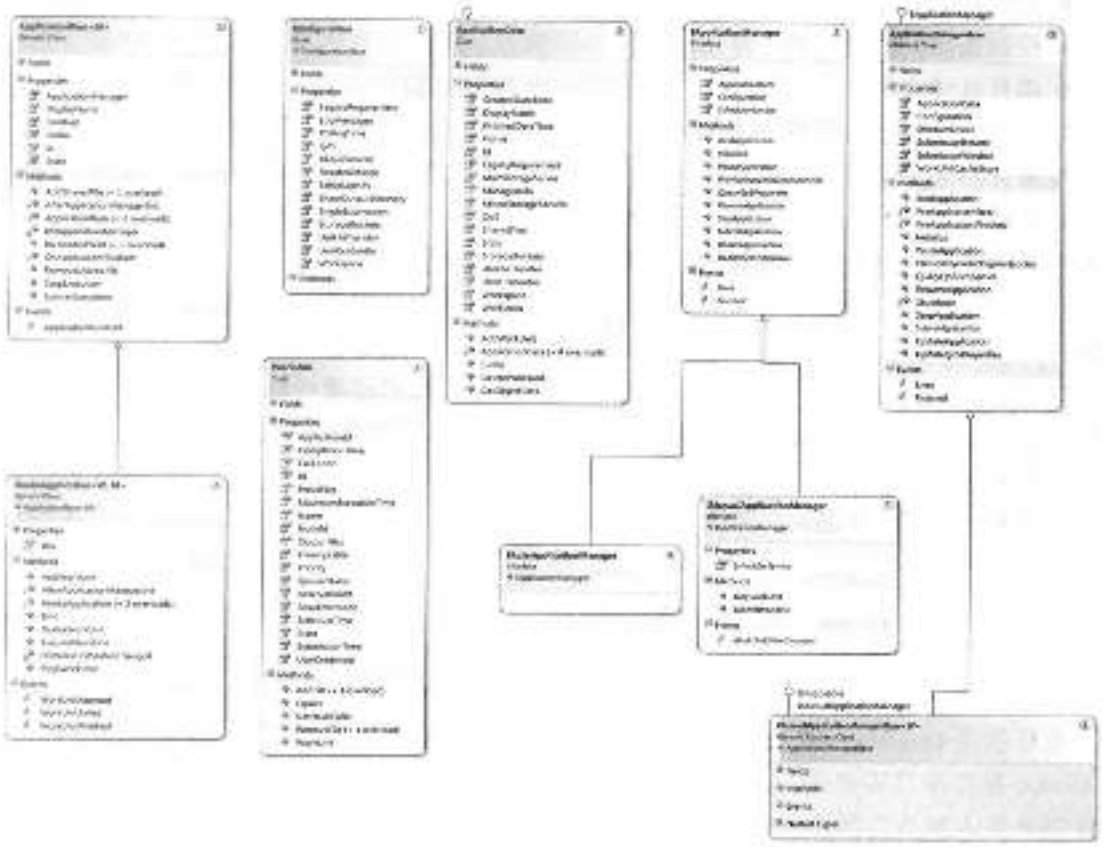


图5-8 Aneka应用模型

一系列附加类组成了对象模型，在这些类中，最值得关注的是Configuration类，它用来初始化应用并配置其行为。以及ApplicationData类，它包含应用的执行信息。

表格5-1概括了目前Aneka可支持的应用模型的特征以及对应的编程模型。编程模型是可以扩展的，因此可以利用表格中的基类构造新的编程模型。编程模型扩展可以通过增加（具体化）现有编程模型的实现特征，或者使用基类来定义新的编程模型和抽象来实现。例如，参数化模型是任务模型的扩展，在Aneka云应用管理环境中已经实现。此类模型向终端用户提供特定接口，用户只需定义相关模板任务和参数。

163
164

表5-1 Aneka应用模型特征

类别	描述	基本应用类型	工作单元	编程模型
手动	工作单元由用户产生，并通过应用提交	AnekaApplication <W, M> IManualApplicationManager <W> ManualApplicationManager <W>	是	任务模型 线程模型
自动	工作单元由运行中的基础设施产生，并在其内部进行管理	ApplicationBase <M> IAutoApplicationManager	否	参数化模型 MapReduce模型

2. 服务模型

Aneka云服务模型定义了实现一个云服务所需的基本要求。容器定义了加载服务的运

行环境。每项加载在容器中的服务必须加载一个 IService 接口，该接口呈现了以下方法和属性：

- 名称和状态。
- 控制操作，例如开始、停止、暂停和继续。
- 消息处理方法。

与终端用户直接交互的服务可以直接提供给用户，例如资源分配和预留服务。除了使用容器来开启和关闭服务的控制操作外，服务的核心逻辑都包含在消息处理机制中，位于 HandleMessage 方法下。每项需要使用服务的操作由具体消息所触发，操作结果也通过消息回执给调用者。

图 5-9 描述了容器中每个服务实例的生命周期。阴影圆圈表示临时状态，空白圆圈表示稳定状态。服务实例初始化后可以进入未知或初始化状态，一种在容器配置阶段触发容器中相应构造器的状态。一旦容器启动，容器会对每项服务调用启动函数，之后服务实例会进入启动状态。启动过程完成后进入运行状态。在容器运行周期内，服务将一直处于运行状态。运行状态也是服务唯一能够处理消息的状态。如果在服务启动过程中发生了异常，服务实例将会返回未知状态，并返回一个错误标识符。

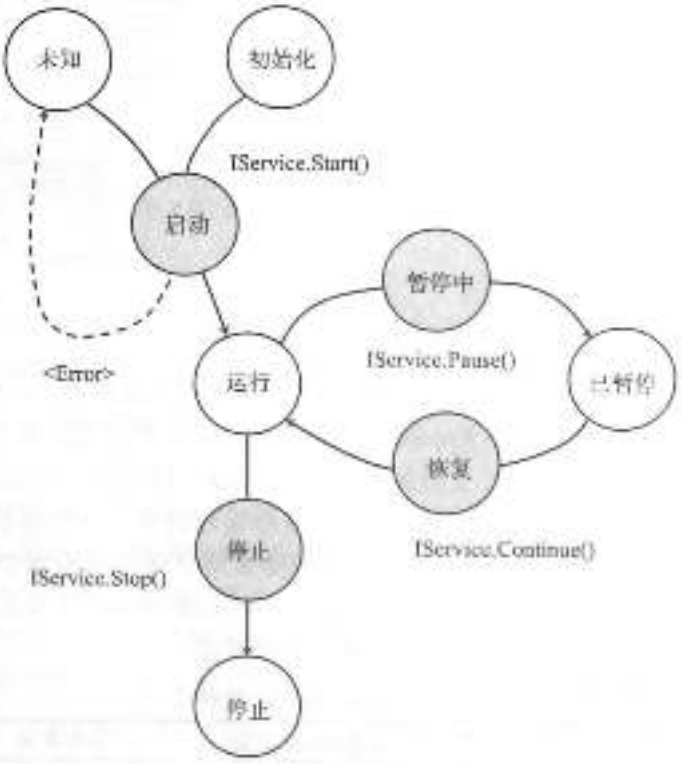


图 5-9　服务生命周期

服务在运行过程中可以使用暂停和继续方法使服务暂停和继续。如图 5-9 所示，服务实例首先进入暂停中状态，然后进入已暂停状态，再之后进入恢复状态并恢复其活动直至返回运行状态。不是所有服务都需要支持暂停 / 继续操作。目前框架中并不是所有服务都提供上述功能。

当容器注销时，容器对每项服务调用停止方法，并进入到最终的停止状态。所有预先分

配的资源将会被释放。

Aneka 提供了一个默认基类来简化服务的实现，同时对服务开发者设定了一些引导规则以使所开发的服务能够与 Aneka 兼容。特别是定义了可扩展的 Servicebase 类来实现服务，该类是框架中一些服务的基类并提供了一些内置特性：

165

- IService 接口所呈现的基本属性的定义。
- 对访问能力和状态控制相关操作的实现。
- 传递服务用户信息的内置框架。
- 支持服务监控。

开发者利用系统提供的定义控制操作行为的模板方法，实现自定义的消息处理机制，并提供对应的服务客户端。

Aneka 使用了一个强类型的信息传递通信模型，每项服务定义自己的消息体系，不同消息的设定是独占的。因此，新服务接口的开发者也需要定义所开发服务使用的消息类型，以便和其他服务和客户端通信。每个消息类型都是 Message 基类的集成子类。Message 类定义了如下特性：

- 源节点和目标节点。
- 源服务和目标服务。
- 安全凭据。

166

每个信息类型还包含附加信息。在 Aneka 平台中，信息传递应用广泛。对于应用直接使用的服务，它们会提供一个服务客户端，提供面向对象的操作接口。Aneka 提供了一个易用平台，可以通过中间件将服务客户端动态注入应用。继承自 ServiceBase 类的服务已经支持了上述特征，只需要为服务客户端类定义接口和具体实现。服务客户端在将 Aneka 服务集成进已有应用方面十分方便，已有应用不需要分布式应用的支持或者接入额外的服务接口。

Aneka 在服务配置方面也提供了丰富的支持。开发者可以定义 editors 类和 configuration 类，通过容器所需的工作流将服务配置信息集成到管理工具中。

5.4.2　管理工具

Aneka 是一个纯 PaaS 实现，使用虚拟或物理设备资源进行部署。因此，基础设备管理，以及在设备上安装逻辑云的相关设备管理，是管理接口层的重要特征。该层也包括了管理云中服务和应用实例的能力。

1. 设备管理

Aneka 使用虚拟或物理设备资源来部署云平台。虚拟硬件总体上通过资源分配服务进行管理，应用根据需求获取资源。物理硬件通过 PAL 层的管理 API，由管理控制台直接管理。管理的内容主要是物理硬件的分配和 Aneka 在硬件上的远程安装。

2. 平台管理

设备管理提供了云部署环境上的一个基础接口层。云创建的过程就是将一系列服务部署在物理设备上，以实现容器的安装和管理。一系列互联的容器定义了应用的执行平台。平台管理的主要内容是云的逻辑组织和框架。Aneka 可以将可用的硬件资源划分到若干云中作为不同用途使用。服务接口实现了 Aneka 云的核心特征，管理接口层提供了一些服务的操作接口，例如云监控、资源分配和预留、用户管理、应用概要分析。

3. 应用管理

应用定义了用户对云的贡献。管理 API 提供监控和分析功能，帮助管理者记录用户和
167 应用对资源的使用。在以资源使用作为用户结算依据的云环境下，这是十分重要的功能。Aneka 提供了对应用执行和资源使用生成宏观和微观分析信息的能力。

上述特征通过主管理控制台的管理工作室来实现。

本章小结

本章介绍了 Aneka，一个支持云应用编程的平台。Aneka 是一个纯 PaaS 实现，构成了一个能够在异构资源（台式机、集群、公共虚拟机）上创建云平台的中间件产品。

Aneka 框架的一个核心特征是可配置运行环境，可以创建基于服务的应用执行中间件。平台的一个基础元素是容器，代表了云的部署单元。容器装载了一系列服务，定义了中间件的处理能力。Aneka 中间件的基础服务如下：

- 构造服务，支持监控、资源分配、硬件分析和成员管理。
- 基础服务，支持存储、资源预留、结算、记录和报告。
- 应用服务，支持调度和执行。

从编程应用的角度，Aneka 提供对不同编程模型的支持，允许开发者通过不同抽象模型扩展分布式应用。该框架目前支持三种模型：相互独立任务的模型、多线程模型和 MapReduce。

Aneka 平台是可扩展的，通过提供应用模型和服务模型来实现扩展，将新的服务接口和编程模型集成到系统内部。

习题

1. 请简单描述 Aneka 的主要特点。
2. Aneka 容器是什么？它的用处是什么？
3. Aneka 容器承载的服务类型有哪些？
4. 描述 Aneka 的资源分配功能。
5. 描述 Aneka 的存储架构的实现。
6. 什么是编程模型？
7. 列举 Aneka 支持的编程模型。
8. Aneka 的基础结构组成组件有哪些？
9. 讨论 Aneka 云的逻辑组织结构。
10. 工作节点承载了哪些服务？
11. 讨论 Aneka 云的私有部署。
168 12. 讨论 Aneka 云的公共部署。
13. 讨论混合部署中动态资源分配的角色。
14. Aneka 为开发提供哪些设施？
15. 讨论 Aneka 应用模型的主要特征。
169 ~ 170 16. 讨论 Aneka 服务模型的主要特征。
17. 讨论 Aneka 管理工具在基础设施、平台和应用中的特征。

并行计算：线程编程

吞吐量计算指的是以事务的形式传送大量的计算。由于设计之初就和事务处理有关[60]，所以吞吐量计算自诞生以来就不断得到扩展，并已经超出了原有的领域。硬件技术的进步导致多核系统的产生，这使得即使是在单个计算机系统内，高吞吐量计算传输成为可能。在这种情况下，吞吐量计算可使用多进程和多线程实现。多进程是指多个程序在一个机器上执行，而多线程与同一个程序内多个指令流的可能性有关。

这一章介绍多线程的概念，并描述多线程如何支持高吞吐量计算应用的发展。本章还讨论了如何将最初设想为包含在一台机器边界内的多线程编程扩展到具有局限性的分布式环境中。Aneka 线程编程模型将会作为一种参考模型来描述云计算多线程模型的一种可行性实现。

6.1 单机并行计算简介

自 20 世纪 60 年代初期，当 Burroughs 公司设计出有史以来第一台 MIMD（多指令多数据）多处理器 D825 时，并行已成为提高计算机性能的一种技术。从那时起，人们提出了各种并行战略。尤其是多进程，即在单个机器内多处理单元的使用，已得到大量的关注并产生了几种并行架构。

最重要的区别之一是在处理单元的对称性方面。非对称多进程包括专门执行不同功能的不同处理单元的同时使用。对称多进程则以使用类似或相同的处理单元分担计算负载为特征。其他例子包括非均匀存储器访问（NUMA）和集群多进程，它们各自定义了一个特定的架构，以便在不同处理器间访问共享存储器，以及将多个计算机结合到一起作为单个虚拟机来使用。

对称和非对称多进程是用于提升商用计算的硬件性能的技术。图形处理单元（GPU）作为实际处理器，是非对称处理的一个应用，而多核技术是对称多进程的演变。因为物理限制对频率扩展[⊖]的影响，现在多处理器（特别是多核技术）显得至关重要，近年来这种技术已经成为实现性能增益的惯例。提高处理器时钟频率而对电源功耗和冷却没有影响已不再可能。2004 年 5 月，当英特尔正式取消两个新微处理器的开发转而支持多核时[⊜]，无代价增加时钟频率变得不再可行。人们普遍认为这一时期是频率扩展时代的结束和多核技术的开端。其他问题也已确定了频率扩展时代的结束，比如不断增加的处理器和内存速度差距，以及增加指令层并行[⊝]来保持单个高性能内核工作的难度。

⊖ 频率扩展是指增加处理器的时钟频率从而提高其性能。时钟频率的增加导致高功耗和可能使得机器停止运转的高温度，这成为时钟频率不能持续增加的限定值。频率扩展也称为频率斜坡，是从 20 世纪 80 年代中期到 2004 年末这段时期内实现性能增益的主导技术。

⊜ www.nytimes.com/2004/05/08/business/08chip.html?ex=1399348800&en=98cc44ca97b1a562&ei=5007。

⊝ 指令层并行是对一个计算机程序一次可以执行多少个操作的一种度量。有几种可以用来在微结构层面增加指令层并行的技术。其中之一是指令流水线技术，涉及指令划分层阶段，以便单个处理单元可以在同一时间内通过对每个指令开展不同阶段而执行多个指令。

多核系统由一个以共享内存的多处理器核为主的单处理器组成。每个核一般有自己的缓存 L1，缓存 L2 为以共享总线方式连接到缓存上的所有核共有，如图 6-1 所示。双核和四核结构现今十分流行，已成为组成商用计算机的标准硬件结构。多核架构也是可行的，但不是为了商用市场所设计。多核技术不仅仅用于支持处理器设计，而且在其他设备（如 GPU 和网络设备）中也成为一个提高性能的标准惯例。

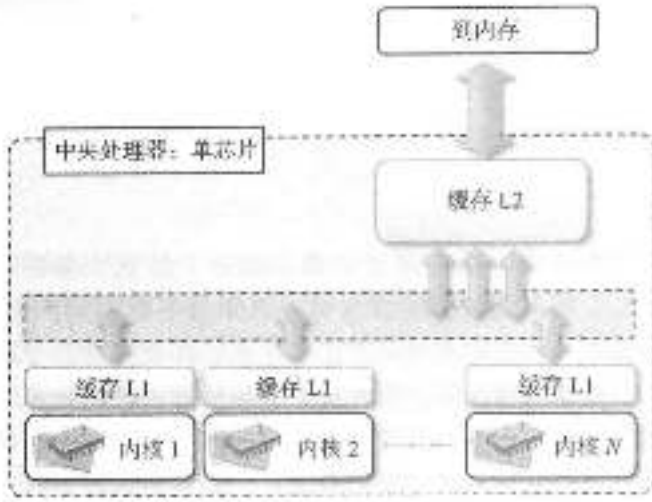


图 6-1　多核处理器

多进程只是一种可以用来实现并行的技术，方法是使用并行硬件架构。当程序设计考虑了这一特点时，并行架构才能被更好地利用。特别地，操作系统作为一个重要的角色，通过抽象的进程和线程定义了应用程序的运行架构。进程是一个应用程序运行时的图像，或者更大程度上是一个正在运行的程序，而线程标识一个进程内执行的单个流。允许多进程执行的系统，同时也支持多任务处理。当在一个进程内明确定义了多个线程结构时，它就支持多线程。

注意，引入多核技术前，多任务和多线程可以在由单个处理器和单核组成的计算机硬件上作为惯例实现。在这种情况下，操作系统可通过交叉执行不同进程和同一进程内不同线程的指令呈现并发执行的假象。这也是在多处理器 / 多核系统中的情况，因为线程或进程的数量高于处理器或内核的数量。如今，几乎所有常用的操作系统都支持多任务和多线程处理。此外，所有主流编程语言都在其 API 中纳入了进程和线程的抽象概念，然而对于开发者而言，多处理器和多核的直接支持却是十分有限的，还经常减少并局限于特定的库，如编程语言 C/C++ 的子集。

本章主要关注多线程编程，这是现已得到完全支持并且在单进程中实现并行最简单的方法（不管底层的硬件结构如何）。

6.2　线程编程应用

现代应用程序可以同时执行多个操作。开发人员按照线程组织程序以便表达程序内部的并发性。线程的使用可以是隐式或显式的。隐式线程发生在底层 API 使用内部线程来执行特定任务以支持应用程序的执行时，如图形用户界面（GUI）的呈现，或在虚拟机语言情况下的垃圾回收。显式线程以应用程序之间的线程使用为特点，开发人员使用这种抽象形式引入并行结构。线程的显式调用一般与设备 I/O 和网络连接相关，对长时间的计算或后台操作的执行结果没有特定的时间约束。线程的使用最初被定向为允许异步操作——尤其是为异步 I/O 或长时间计算提供设施，以便应用程序的用户界面不会被阻挡或没有反应。随着并行架构的到来，多线程的使用已经成为一种有效的技术，它能够提高系统吞吐量，同时也是吞吐量计算的一种可行选择。为达到此目的，线程的使用会对需要重构的算法的设计产生显著影响。本节讨论用于支持并行和分布式算法设计的线程的使用。

6.2.1　什么是线程

线程标识进程内的单个控制流，是指令的一个逻辑序列。指令的逻辑序列指的是按照程序员的设计一个接一个执行的指令序列。通俗来讲，线程标识一种用于缝纫的纱线，而由纱线的互锁纤维表现出的连续感，则对应着由线程指令表达出的操作所具有的逻辑连续序列的概念。

支持多线程的操作系统将线程识别为表示运行代码的最小程序块。这意味着除了开发人员使用的显式线程之外，操作系统执行的任何指令序列都在一个线程中。因此，每个进程至少包含一个线程，但在某些情况下，进程会由多个具有不同生存周期的线程组成。同一个进程中的线程共享内存空间和执行内容，除此之外，不同进程的线程没有实质性的区别。

在一个多任务环境中，操作系统为每个进程分配不同的时隙并交错执行。暂停一个进程的执行的进程，在寄存器中保存所有信息（通常包括 CPU 的状态以便以后恢复），并把它替换为与另一个进程相关的信息，这称为上下文。一般认为这一操作是费时的，多线程的使用可使内容切换产生的延时减到最小，从而更易于实现多任务的执行。表征一个线程的执行的状态比描述一个进程要简单。因此，相比于进程之间的切换，更倾向于选择线程之间的切换。显然，当且仅当执行的任务在逻辑上是彼此相关的且需要共享内存或其他资源时，使用多线程取代多进程是合乎情理的。如果不是这样，更好的设计是将它们分离到不同的进程中。

图 6-2 提供了线程和进程之间的关系概览，以及一个多线程应用运行时的简化描述。正在运行的程序标识为一个进程，其中包含至少一个线程，也称为主线程。此类线程是由编译器或正在执行程序的运行环境隐式创建的。这个线程很可能持续在程序的整个生存周期，并且是其他线程的起源，这些程序一般显示较短的持续时间。主线程可以衍生出其他线程。主线程与进程生存周期中产生的其他线程没有区别，它们每个都有自己的本地存储和要执行的指令序列，都共享分配给整个进程的存储空间。当所有的线程都完成时才可以认为进程执行终止。

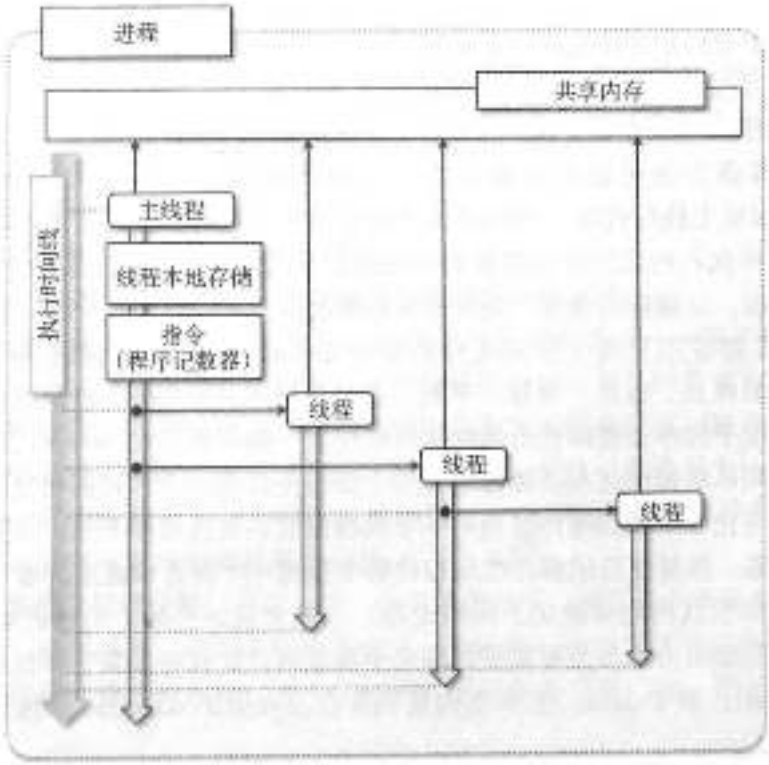


图 6-2　进程与线程的关系

6.2.2　线程 API

尽管多线程的支持依操作系统和用于开发应用的特定编程语言的不同而不同，但是确定
174 一个所有实现通用的最小限度的功能集是可能的。

1. POSIX 线程

UNIX 可移植操作系统接口（POSIX）是一套应用编程接口标准，用于 UNIX 操作系统环境下的应用程序便携式开发。POSIX 1.c 标准（IEEE Std 1003.1c-1995）解决了线程的执行问题，允许程序员开发可移植的多线程应用程序。这个标准针对基于 UNIX 的操作系统，但已有提供基于 Windows 系统的相同技术参数的实现。

UNIX 可移植操作系统接口标准定义了以下操作：创建带属性的线程、线程的终止和等待线程结束（连接操作）。除了线程的逻辑结构之外，还将引进其他抽象概念，如信号量、条件、读写锁等，以支持线程之间的正确同步。

UNIX 可移植操作系统接口提出的模型已成为其他实现的参考，这可能会为开发者提供具有类似功能的不同接口。从编程角度要重点记忆的是以下几点：

- 一个线程标识指令的一个逻辑序列。
175 • 一个线程被映射到一个包含将要执行的指令序列的函数。
- 一个线程可以创建、终止和连接。
- 一个线程有一个可以确定其当前情况的状态，无论该线程是在执行、中断、终止、等待还是输入 / 输出等。
- 线程正在经历的状态序列，部分由操作系统调度器决定，部分由应用程序开发人员决定。
- 线程共享进程的内存，由于二者同时进，因此需要同步结构。
- 提供不同同步抽象概念以解决不同的同步问题。

C 语言提供了 UNIX 可移植操作系统接口标准 1.c 的默认实现。所有可用函数和数据结构都在 pthread.h 头文件中，这是标准 C 语言实现的一部分。

2. Java 和 .NET 中的线程支持

Java 和 C# 等语言通过面向对象的方法为多线程编程提供丰富的函数。由于 Java 和 .NET 都在虚拟机上执行代码，所以库提供的应用程序接口涉及托管或逻辑线程。由这些语言开发的程序所执行的运行环境映射到物理线程上（即由底层操作系统支持的抽象）。暂不论这个映射过程，从编程的角度，托管的线程将被视为物理线程并呈现相同的功能。

Java 和 .NET 都使用呈现了形同线程操作的 Thread 类来表达线程抽象：开始，结束，暂停，恢复，强制终止，休眠，连接，中断。开始和结束 / 休眠用于控制线程实例的生存时间，暂停和恢复用于程序化暂停然后继续执行线程。一般不推荐在 Java 和 .NET 中使用这两种操作，而是使用休眠操作适当的锁技术。这个操作允许在一个预定的时间周期内终止线程的执行。这一点与让一个线程等待直到另一个线程完成的连接操作不同。使用中断操作可以中断这些等待状态，恢复执行线程并在线程代码中生成一个异常以通知异常恢复。

两个框架对线程间的同步提供不同的支持。一般来说，互斥、关键区域和读写器 – 编写器锁这些基本功能是由基本类库或附加库完全覆盖的。比线程抽象更高级的构造对于这两种语言都是可行的。对于 Java，大多数构造包含在 java.util.concurrent[⊖] 包中，而一组丰富

⊖ http://download.oracle.com/javase/6/docs/api/java/util/concurrent/package-summary.html。

的 .NET 并发编程的应用程序接口通过 .NET 并行扩展框架[⊖]得到进一步扩大。 176

6.2.3　线程并行计算技术

开发并行应用需要对问题及其逻辑结构有足够的认知。理解应用中任务的依赖项和关联关系是设计正确的程序结构并在适当情况下引入并行的基础。分解是一项有用的技术，用于分析问题是否可以分成部分（或任务）来同时执行。如果这种分解是可能的，那么它也为并行实现提供一个起始点，因为它允许线程并发执行的任务分解为独立单元。两个主要的分解/分割技术是域分解和功能分解。

1. 域分解

域分解过程用来识别功能重复但独立的数据计算模式。就吞吐量计算而言，这是最常见的分解方式，并且与解决问题所需的重复计算识别相关。

当这些计算相同，只是操作的数据不同，且可以按任何顺序执行时，这种问题称为高度并行[59]。高度并行问题是最简单的并行实例，因为没有必要同步没有共享任何数据的不同线程。此外，线程之间的协调和通信是极少的，这大大简化了代码逻辑且允许高计算吞吐量。

在许多情况下，为解决此类问题制定一个一般结构是可能的，并且在一般情况下，可以通过域分解实现并行。主从模型是一个非常常见的组织结构，对于这些情况：

- 系统分为两个主要的代码段。
- 一个代码段包含分解逻辑和协调逻辑。
- 另一个代码段包含要执行的重复计算。
- 主线程执行第一个代码段。
- 作为主线程执行结果，需要创建足够多的从属线程来执行重复计算。
- 每个从属线程的结果收集和最终结果的合成由主线程执行。

尽管重复计算的复杂性严格取决于问题的性质，但协调和分解逻辑通常很简单，只涉及确定要创建的作业的单位数量。一般来说，常使用一个 while 循环或 for 循环表达分解逻辑，且每次迭代产生的新作业单元将被分配到一个从属线程。这个过程的优化涉及线程池的使用，以限制用于执行重复计算的线程的数量。

就高度并行问题而言，几个实际问题包括：

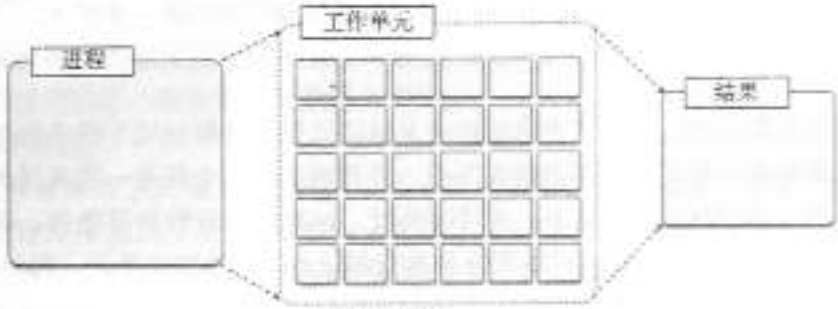
- 二维（或更高维）数据集的几何变换。
- 一个域的独立和重复计算，如曼德勃罗集和蒙特卡罗计算。 177

虽然高度并行问题是很普遍的，但是它们基于强有力的假设——每个迭代的分解方法中，隔离作业的一个独立单位是有可能的。这使得获得高速计算吞吐量成为可能。如果所有迭代的值都依赖于以前迭代获得的值，那么这样的条件是不符合假设的。这种情况称为固有顺序问题，且不能直接应用前面所述的方法。除此之外，仍然可能将整个计算分解为一系列独立的作业单元，它们可能有不同的粒度，比如通过分组得到单个计算依赖的迭代。图 6-3 所示为高度并行的分解示意图和固有顺序问题。

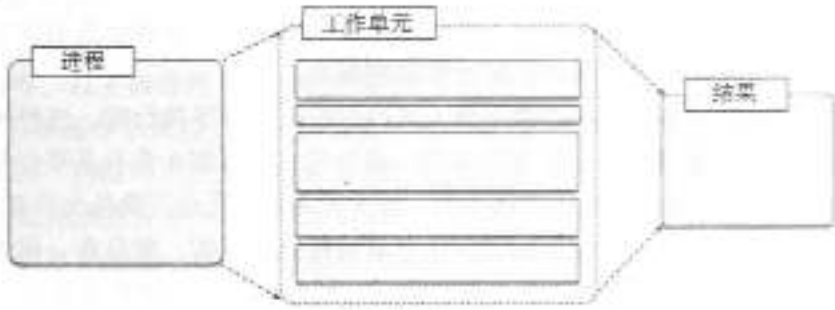
为了展示如何将应用域分解，可以创建一个简单的程序，使用多个线程来执行矩阵乘法。

矩阵乘法是二进制操作，由两个矩阵生成另一个矩阵。原始矩阵经线性变换后组成计算结果。有几种执行矩阵乘法的技术，其中，矩阵的乘积是使用最广的。图 6-4 为矩阵积的执行示意图。 178

⊖ http://msdn.microsoft.com/en-us/concurrency/default.aspx。



a) 高度并行



b) 固有顺序

图 6-3　域分解技术

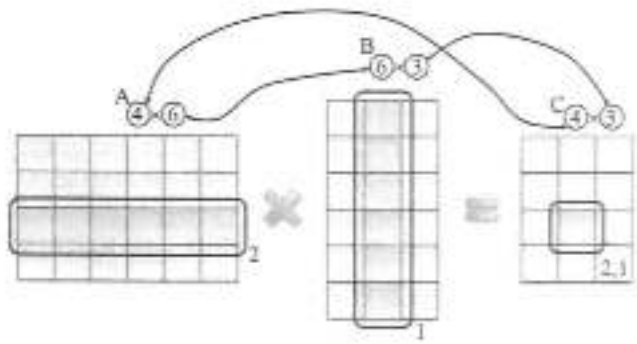


图 6-4　矩阵乘积

矩阵积计算结果矩阵中的每个元素，分别是第一个和第二个输入矩阵相应的行与列的线性组合。应用于每个结果矩阵元素的公式如下：

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} B_{kj}$$

因此，执行矩阵积需要遵守两个条件：

- 输入矩阵必须包含类具有内积定义的可比性数值。
- 第一个矩阵的列数量必须与第二个矩阵的行数量相匹配。

鉴于这些条件，结果矩阵具有第一个矩阵的行数量和第二个矩阵的列数量，并且每个元素由前面等式所述的方式计算。

显然，迭代运算的结果由矩阵中的每个元素计算得到。它们都使用相同的公式，并且计算不依赖于从结果矩阵其他元素计算得来的值。因此，对于高度并行问题，可以按以下步骤从逻辑上组织多线程程序：

- 通过执行前面的等式定义执行结果矩阵中单个元素计算的功能。
- 创建两个循环（第一个索引遍历第一个矩阵的行，第二个索引遍历第二个矩阵的列）产生计算结果矩阵元素的线程。
- 结合所有线程完成计算，并编写结果矩阵。

为了举个这种解决方案的实际例子，我们将使用 .NET 线程进行演示。.NET 框架提供 System.Threading.Thread 类，可用一个函数指针配置，也称为一个代表，以异步方式执行。 179
这样，一个代表必须引用一些类中的定义方法。因此，我们可以定义一个简单的类，以行列乘积性质和结果值的方式公开。这个类也定义了执行实际计算的方法。程序 6-1 所示为 ScalarProduct 类。

程序 6-1　ScalarProduct 类

```
///<summary>
/// Class ScalarProduct. Computes the scalar product between the row and the column
/// arrays.
///</summary>
public class ScalarProduct
{
   /// <summary>
   /// Scalar product.
   /// </summary>
   private Double result;
   /// <summary>
   /// Gets the resulting scalar product.
   /// </summary>
   public double Result{ get { returnthis.result; } }

   /// <summary>
   /// Arrays containing the elements of the row and the column to multiply.
   /// </summary>
   private double[] row, colum;

   /// <summary>
   /// Creates an instance of the ScalarProduct class and configures it with the given
   /// row and column arrays.
   /// </Summary>
   /// <param name='row'>Array with the elements of the row to be multiplied.</param>
   /// <param name='column'>Array with the elements of the column to be multiplied.
   /// </param>
   public ScalarProduct(double[] row, double[] colum)
   {
      this.row = row;
      this.colum = colum;
   }
   /// <summary>
   /// Executes the scalar product between the row and the colun.
   /// </summary>
   /// <param name='row'>Array with the elements of the row to be multiplied.</param>
   /// <param name='column'>Array with the elements of the column to be multiplied.
   /// </param>
   public void Multiply()
   {
      this.result = 0;
      for(int i=0; i<this.row.Length; i++)
      {
         this.result += this.row[i] * this.column[i];
      }
   }
}
```

控制主线程的类是很简单的。在这种情况下，我们跳过需要从标准输入或从文件读取矩阵的样例代码，并着眼于主控制逻辑，通过分解计算、创建线程并等待其完成来编写结果矩阵。

为了控制线程，需要追踪线程以便完成计算后可以确认线程的状态并获得结果。可以创建一个简单的程序读取矩阵，在一个合适的数据结构中跟踪所有线程，一旦线程完成，则组成最终结果。程序 6-2 所示为 MatrixProduct 类（有所省略）。

程序 6-2　MatrixProduct 类（主程序）

```
using System;
using System.Threading;
using System.Collections.Generic;

///<summary>
/// Class MatrixProduct. Performs the matrix product of two matrices.
///</summary>
public class MatrixProduct
{
    ///<summary>
    /// First and second matrix of the product.
    ///</summary>
    private static double[,]a, b;
    ///<summary>
    /// Result matrix.
    ///</summary>
    private static double[,] c;
    ///<summary>
    /// Dictionary mapping the thread instances to the corresponding ScalarProduct
    /// instances that are run inside.
    ///</summary>
    private static IDictionary<Thread, ScalarProduct>workers;

    ///<summary>
    /// Read the command line parameters and perform the scalar product.
    ///</summary>
    ///<param name='args'>Array strings containing the command line parameters.</param>
    public static void Main(string[] args)
    {
        // reads the input matrices a and b.
        MatrixProduct.ReadMatrices();
        // executes the parallel matrix product.
        MatrixProduct.ExecuteProduct();
        // waits for all the threads to complete and
        // composes the final matrix.
        MatrixProduct.ComposeResult();
    }

    ///<summary>
    /// Executes the parallel matrix product by decomposing the problem in
    /// independent scalar product between rows and colums.
    ///</summary>
    private static void ExecuteThreads()
    {
        MatrixProduct.workers = new List<Thread>();
        int rows = MatrixProduct.a.Length;
        // in .NET matrices are arrays of arrays and the number of columns is
        // is represented by the length of the second array.
        int columns = MatrixProduct.b[0].Length;
        for(int i=0; i<rows; i++)
            for(int j=0; j<columns; j++)
            {
                double[] row = MatrixProduct.a[i];
                // because matrices are stored as arrays of arrays in order to
                // to get the columns we need to traverse the array and copy the
                // the data to another array.
```

```
            double[] column = new double[common];
            for(int k = 0; k<common; k++)
            {
                column[k] = MatrixProduct.b[k, j];
            }
            // creates a ScalarProduct instance with the previous rows and
            // columns and starts a thread executing the Multiply method.
            ScalarProduct scalar = new ScalarProduct(row, column);
            Thread worker = new Thread(new ThreadStart(scalar.Multiply));
            worker.Name = string.Format("{0},{1}",row,column);
            worker.Start();
            // adds the thread to the dictionary so that it can be
            // further retrieved.
            MatrixProduct.workers.Add(worker, scalar);
        }
    }

    /// <summary>
    /// Waits for the completion of all the threads and composes the final
    /// result matrix.
    /// </summary>
    private static void ComposeResult()
    {
        MatrixProduct.c = new double[rowsA,colsB];
        foreach(KeyValuePair<Thread,ScalarProduct> pair in MatrixProduct.workers)
        {
            Thread worker = pair.Key;
            // we have saved the coordinates of each scalar product in the name
            // of the thread now we get them back by parsing the name.
            string[] indices = worker.Name.Split(new char[] {','});
            int i = int.Parse(indices[0]);
            int j = int.Parse(indices[1]);
            // we wait for the thread to complete
            worker.Join();
            // we get the result computed at the given coordinates.
            MatrixProduct.c[i,j] = pair.Value.Result;
        }
        MatrixProduct.PrintMatrix(MatrixProduct.c);
    }
    /// <summary>
    /// Reads the matrices.
    /// </summary>
    private static void ReadMatrices()
    {
        // code for reading the matrices a and b
    }
    /// <summary>
    /// Prints the given matrix.
    /// </summary>
    /// <param name="matrix">Matrix to print.</param>
    private static void PrintMatrix(double[,] matrix)
    {
        // code for printing the matrix.
    }
}
```

大部分的复杂程序留在线程的管理中，而域分解是十分简单的。之前的实现产生过几个问题：

- 矩阵布局。由于使用多维度数组储存方式，检索标量积的列不像获得行那样简单。一种简单的解决办法是将第二个矩阵记录为列 × 行的行式，而不是行 × 列。
- 结果组成。结果的组成在主线程上完成，且需要跟踪所有的工作线程。保持引用所有工作线程一般是好的编程习惯，因为有必要在应用完成之前终止所有线程，但是

在这种情况下可以用同步结构修改应用，允许更新工作线程的结果矩阵。新的设计意味着存储行与列的索引信息，以及 ScalarProduct 类中结果矩阵的引用。作为结果，没有必要维护线程字典，且在主线程中不需要 ComposeResult 方法。

矩阵积的例子已作为描绘基本逻辑的一个模型，这需要执行域分解为高度并行问题并在 .NET 中使用线程实现吞吐量计算。这个例子可以当作开发更复杂的应用程序的参考。

2. 功能分解

功能分解是确定功能不同但独立的计算的过程。这里的重点是计算的类型而不是计算操作的数据。这种分解不常见且不会导致大量线程，因为单个程序中执行不同的计算会受限制。

[180] 功能分解促进了独立任务单元中问题的自然分解，因为这不涉及数据集分割，而是由不同的逻辑操作清晰定义的。图 6-5 展示了分解是如何执行以及如何并行的。

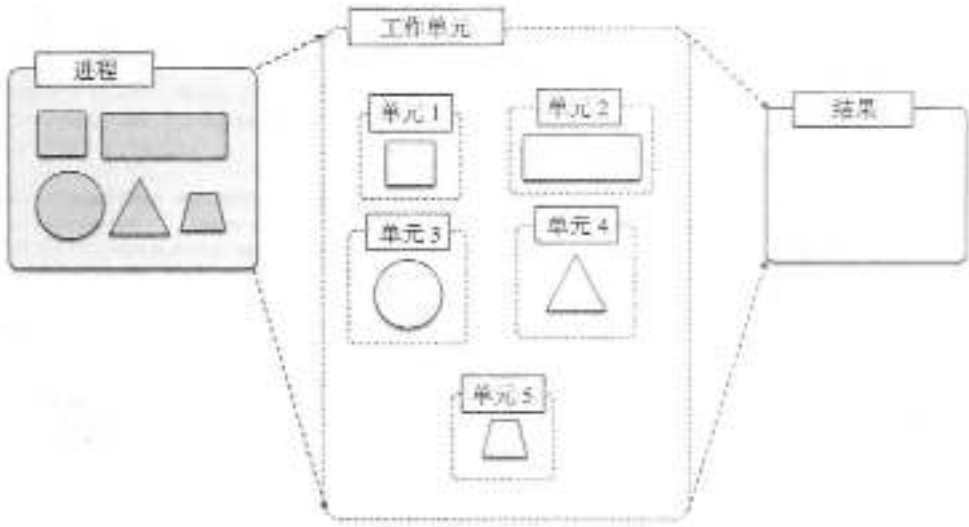


图 6-5　功能分解

正如图 6-5 所描述的原理，受功能分解影响的问题也可以要求将任务的每个独立单元的成果组合在一起构成一个组成阶段。在域分解的情况下，这个阶段常常导致一个聚合过程。结果由哪种方式组成，很大程度上取决于定义这个问题的操作类型。

接下来，我们举一个简单的例子，说明一个数学问题如何通过使用分解功能实现并行化。假设在给定 x 值的情况下需要计算下列函数的值：

$$f(x) = \sin(x) + \cos(x) + \tan(x)$$

显然，一旦 x 值设定，三个不同运算可以彼此独立执行。这是一个功能分解的例子，因为整个问题可以分解成三个不同的运算。并行计算方法一种可能的实现如程序 6-3 所示。

程序在三个独立线程中分别计算 sine、cosine 和 tangent 函数然后合计结果。这一实现构成了在前面样例程序中所讨论的替代技术的示例。把一个函数指针传递给每个线程以便可以在计算结束时更新最终结果，而非通过使用一种数据结构用于跟踪已创建的任务线程。

[181 / 184] 这个技术引入了一个同步问题，该问题通过函数指针引用方法中的 lock 语句得到正确处理。lock 语句创建了一个临界区，一次只能被一个线程访问且保证最终结果会及时更新。

程序 6-3　数学函数

```
using System;
using System.Threading;
using System.Collections.Generic;

/// <summary>
/// Delegate UpdateResult. Function pointer that is used to update the final result
/// from the slave threads once the computation is completed.
/// </summary>
/// <param name="x">partial value to add.</param>
public delegate void UpdateResult(double x);

/// <summary>
/// Class Sine. Computes the sine of a given value.
/// </summary>
public class Sine
{
   /// <summary>
   /// Input value for which the sine function is computed.
   /// </summary>
   private double x;
   /// <summary>
   /// Gets the input value of the sine function.
   /// </summary>
   public double X { get { return this.x; } }
   /// <summary>
   /// Result value.
   /// </summary>
   private double y;
   /// <summary>
   /// Gets the result value of the sine function.
   /// </summary>
   public double Y { get { return this.y; } }
   /// <summary>
   /// Function pointer used to update the result.
   /// </summary>
   private UpdateResult updater;
   /// <summary>
   /// Creates an instance of the Sine and sets the input to the given angle.
   /// </summary>
   /// <param name="x">Angle in radians.</param>
   /// <param name="updater">Function pointer used to update the result.</param>
   public Sine(double x, UpdateResult updater)
   {
      this.x = x;
      this.updater = updater;
   }
   /// <summary>
   /// Executes the sine function.
   /// </summary>
   public void Apply()
   {
      this.y = Math.Sin(this.x);
      if (this.updater != null)
      {
         this.updater(this.y);
      }
   }
}

///<summary>
/// Class Cosine. Computes the cosine of a given value.
///</summary>
public class Cosine
{
   /// <summary>
```

```
    /// Input value for which the cosine function is computed.
    /// </summary>
    private double x;
    /// <summary>
    /// Gets the input value of the cosine function.
    /// </summary>
    public double X { get { return this.x; } }
    /// <summary>
    /// Result value.
    /// </summary>
    private double y;
    /// <summary>
    /// Gets the result value of the cosine function.
    /// </summary>
    public double Y { get { return this.y; } }
    /// <summary>
    /// Function pointer used to update the result.
    /// </summary>
    private UpdateResultupdater;
    /// <summary>
    /// Creates an instance of the Cosine and sets the input to the given angle.
    /// </summary>
    /// <param name="x">Angle in radiants.</param>
    /// <param name="updater">Function pointer used to update the result.</param>
    public Cosine(double x, UpdateResultupdater)
    {
        this.x = x;
        this.updater = updater;
    }
    /// <summary>
    /// Executes the cosine function.
    /// </summary>
    public void Apply()
    {
        this.y = Math.Cos(this.x);
        if (this.updater != null)
        {
            this.updater(this.y);
        }
    }
}

///<summary>
/// Class Tangent. Computes the tangent of a given value.
///</summary>
public class Tangent
{
    /// <summary>
    /// Input value for which the tangent function is computed.
    /// </summary>
    private double x;
    /// <summary>
    /// Gets the input value of the tangent function.
    /// </summary>
    public double X { get { return this.x; } }
    /// <summary>
    /// Result value.
    /// </summary>
    private double y;
    /// <summary>
    /// Gets the result value of the tangent function.
    /// </summary>
    public double Y { get { return this.y; } }
    /// <summary>
    /// Function pointer used to update the result.
```

```
    /// </summary>
    private UpdateResult updater;
    /// <summary>
    /// Creates an instance of the Tangent and sets the input to the given angle.
    /// </summary>
    /// <param name="x">Angle in radians.</param>
    /// <param name="updater">Function pointer used to update the result.</param>
    public Tangent(double x, UpdateResult updater)
    {
        this.x = x;
        this.updater = updater;
    }
    /// <summary>
    /// Executes the cosine function.
    /// </summary>
    public void Apply()
    {
        this.y = Math.Tan(this.x);
        if (this.updater != null)
        {
            this.updater(this.y);
        }
    }
}
/// <summary>
/// Class Program. Computes the function sin(x) + cos(x) * tan(x).
/// </summary>
public class Program
{
    /// <summary>
    /// Variable storing the computed value for the function.
    /// </summary>
    private static double result;
    /// <summary>
    /// Synchronization instance used to avoid keeping track of the threads.
    /// </summary>
    private static object synchRoot = new object();
    /// <summary>
    /// Read the command line parameters and perform the scalar product.
    /// </summary>
    /// <param name="args">Array strings containing the command line parameters.</param>
    public static void Main(string[] args)
    {
        // gets a value for x
        double x = 3.6;

        // creates the function pointer to the update method.
        UpdateResult updater = new UpdateResult(Program.Sum);

        // creates the sine thread.
        Sine sine = new Sine(x, updater);
        Thread tSine = new Thread(new ThreadStart(sine.Apply));

        // creates the cosine thread.
        Cosine cosine = new Cosine(x, updater);
        Thread tCosine = new Thread(new ThreadStart(cosine.Apply));

        // creates the tangent thread.
        Tangent tangent = new Tangent(x, updater);
        Thread tTangent = new Thread(new ThreadStart(tangent.Apply));

        // shuffles the execution order.
        tTangent.Start();
        tSine.Start();
        tCosine.Start();
```

```
            // waits for the completion of the threads.
            tCosine.Join();
            tTangent.Join();
            tSine.Join();

            // the result is available, dumps it to console.
            Console.WriteLine("f({0}): {1}", x, Program.result);
        }
        /// <summary>
        /// Callback that is executed once the computation in the thread is completed
        /// and adds the partial value passed as a parameter to the result.
        /// </summary>
        /// <param name="partial">Partial value to add.</param>
        private static void Sum(double partial)
        {
            lock (Program.synchRoot)
            {
                Program.result += partial;
            }
        }
    }
```

3. 计算与通信

在设计并行和一般分布式应用程序时，仔细评估问题分解过程中已确定的组件之间的通信模式是十分重要的。本节介绍的两种分解方法以及相应的示例应用程序都基于组件是独立的这一假设。这意味着：

- 一个计算所需要的输入值并不依赖于另一个计算产生的输出值。
- 作为分解结果而产生的不同任务单元并不需要彼此交互（即数据交换）。

这两个假设极大地简化了实现过程且能够实现高度并行和高吞吐量。让所有工作线程成为相互独立的操作系统（或虚拟运行环境），这为调度程序中所有线程提供了最大的自由。不同线程之间所需要的数据交换引入了它们之间的依赖关系，且最终可以导致性能瓶颈。比如，不为线程引入任何队列技术，但如果在队列中的一些线程需要进行数据交换，那么队列线程有可能构成执行应用程序的一个问题。更常见的缺点是当一个线程与另一个线程交换数据时，会使用某种可能导致阻碍其他线程执行的同步策略。需要交换的数据越多，越会阻碍线程同步，最终会影响整体的吞吐量。

作为经验法则，当实现并行和分布式应用时，最小化需要交换的数据量是很重要的。不同线程之间没有通信是实现最高吞吐量的条件。

6.3 Aneka 多线程方式

由于应用程序日益复杂，因此对一台多核计算机所能提供计算能力的需求也变得更大了。一台机器的计算能力一般不能解决这种需求，所以自然而然地需要利用像云这样的分布式基础设施。分解技术可以应用于将一个给定的应用分割成几个工作单元，而不是在一个节点上以多线程来运行，于是这个应用就可以利用云来运行。

即使分布式设备可以显著地增加应用的并行度，但它的使用却会在应用设计和性能方面造成影响。例如，如果不同的工作单元不是运行在同一个处理环境中而是在不同的节点上，代码和数据就需要移动到不同的运行环境中。同样，运行结果也要从远端返回主进程。不仅如此，如果在不同的工作单元之间存在通信，最后就有必要利用中间件提供的 API 来重新

设计通信模型。换句话说，这种从单进程多线程执行方式到分布式执行方式的转变并不简单，往往需要应用的重新设计与重新实现。 185 ~ 189

转变一个应用的代价一般取决于管理分布式基础设施的中间件提供的设施。Aneka作为一个管理集群、网格和云的中间件，向开发者提供了实现分布式应用的高级性能。它在传统的线程编程上更进一步：允许用户以传统方式来编写多线程应用，同时通过附加方式使得这些线程可以各自脱离父进程运行于不同的机器上。事实上，这些“线程”都是在不同节点上执行的独立进程，并且不共用内存和其他资源，但是Aneka允许用户写出像传统线程一样拥有并发和同步的线程结构的线程应用。Aneka线程使用户可以用最小的转变代价轻松地将现有的多线程的、计算密集的应用转向可以自发地利用多个机器更快地运行的分布式版本。

6.3.1 线程编程模型简介

Aneka以线程编程模型的方式提供了在云上实现多线程应用的能力。这个模型提出了分布式线程的概念，也可以称为Aneka线程，Aneka线程模拟了本地线程的行为，但却在分布式基础设施上执行。线程编程模型能够在分布式基础设施上运行高吞吐量线程级并行应用，同时为高度并行的应用提供了最佳环境。

如5.4.1节所述，为Aneka设计的每一个应用都由一个可以和中间件通信的本地对象表示。根据框架所支持的各种编程模型，每一个接口提供了不同的功能，为了有效地支持程序的设计和实现，它们都以一种特定的编程方式进行了优化。在线程编程模型的情况下，程序被设计成一个线程集合，它们的共同执行代表了应用的运行。线程由应用开发者来创建和控制，一旦启动，Aneka将负责调度它们的执行。开发者通过本地对象来控制透明移动和远程执行的线程，这些本地对象就像是远程线程的代理一样。这种方法使得由本地多线程应用向分布式应用的转变变得简单且无缝。

线程编程模型的API模拟了线程的.NET基类库。通过这种方法开发者不需要完全重写应用即可达到使用Aneka的目的。移植本地多线程应用的过程很简单，只需要替换System.Threading.Thread类并引进AnekaApplication类。有三种主要元素组成了基于线程编程模型的应用程序对象模型：

- 应用。这一类代表了Aneka中间件的接口并且构成了分布式应用的本地视图。在线程编程模型中，单个工作单元是由程序员来创建的。因此，需要使用的特定类是Aneka.Entity.AnekaApplication<T,M>，T和M应该根据情况合理选择。
- 线程。线程表示模型的主要抽象并且构成了分布式应用的基本单位。Aneka提供了Aneka.Threading.AnekaThread类，该类代表分布式线程。这个类呈现了System.Threading.Thread类所呈现的方法的子集，这个子集已经缩减为只包含有意义的、可以有效地在分布式环境中实现的操作和属性。 190
- 线程管理器。线程管理器是一个用来追踪分布式线程执行并向应用提供反馈的内部组件。Aneka为这个模型提供了一个特定版本的管理器，它在Aneka.Threading.ThreadManager类中实现。

因此，将一个本地多线程应用移植到Aneka涉及定义AnekaApplication<AnekaThread, ThreadManager>类的一个实例，以及用Aneka.Threading.AnekaThread替换所有的System.Threading.Thread。类似于本地线程，开发者可以创建线程，控制它们的生命周期，并调整它们的执行。

Aneka 应用还具有一些其他的附加属性，例如通知线程执行结束和失败、整个应用执行的完成以及线程状态转变的事件。这些操作对于线程编程模型和构建一些可以用于转变本地多线程应用的附加特征都是可用的。同样，AnekaApplication 类提供了对于文件的支持，并且这些支持都是透明、自动地转移到分布式环境中的。

6.3.2 Aneka 线程和普通线程

为了在分布式基础设施上高效运行，Aneka 线程相对于普通线程有一些特定的限制。这些限制和平常用于多线程应用的通信和同步策略相关。

1. 接口兼容性

Aneka.Threading.AnekaThread 类除了不支持一小部分操作以外，呈现的接口和 System.Threading.Thread 类几乎一样。表 6-1 对比了这两个类提供的操作。其中定义的支持所有线程的命名空间都是指 Aneka.Threading 而不是 System.Threading。

表 6-1　线程 API 对比

Net 线程 API	Aneka 线程 API
System.Threading	Aneka.Threading
Thread	AnekaThread
Thread.ManagedThreadId (int)	AnekaThread.Id(string)
Thread.Name	AnekaThread.Name
Thread.ThreadState (ThreadState)	AnekaThread.State
Thread.IsAlive	AnekaThread.IsAlive
Thread.IsRunning	AnekaThread.IsRunning
Thread.IsBackground	AnekaThread.IsBackground[false]
Thread.Priority	AnekaThread.Priority[ThreadPriority.Normal]
Thread.IsThreadPoolThread	AnekaThread.IsThreadPoolThread[false]
Thread.Start	AnekaThread.Start
Thread.Abort	AnekaThread.Abort
Thread.Sleep	未提供
Thread.Interrupt	未提供
Thread.Suspend	未提供
Thread.Resume	未提供
Thread.Join	AnekaThread.Join

对于本地线程来说，像开始和中止这样的基本操作都有一个直接映射，然而却不支持一些涉及线程执行的临时中断操作。这种设计策略的原因有两个方面。首先，即使对于本地线程，使用暂停/恢复操作也并不是一个好的选择，因为暂停操作会毫无征兆地中断线程的执行状态。其次，在分布式环境中发生线程的暂停会导致对基础设施的低效使用，而在基础设施中资源是被不同的用户和应用所共享的。这也是不支持睡眠操作的原因。因此，没有必要支持会强力地将线程从等待或睡眠状态恢复的中断操作。为了支持线程之间的同步操作，Aneka 提供了一个相应的连接操作的实现。

除了这些基本的线程控制操作之外，一些最相关的属性也已实现，例如名字、特定标识符和状态。名字是自由分配的，标识符是由 Aneka 生成的，用一个字符串而不是整型变量来代表一个全局的唯一标识符。IsBackground、Priority 和 IsThreadPoolThread 等属性只是为

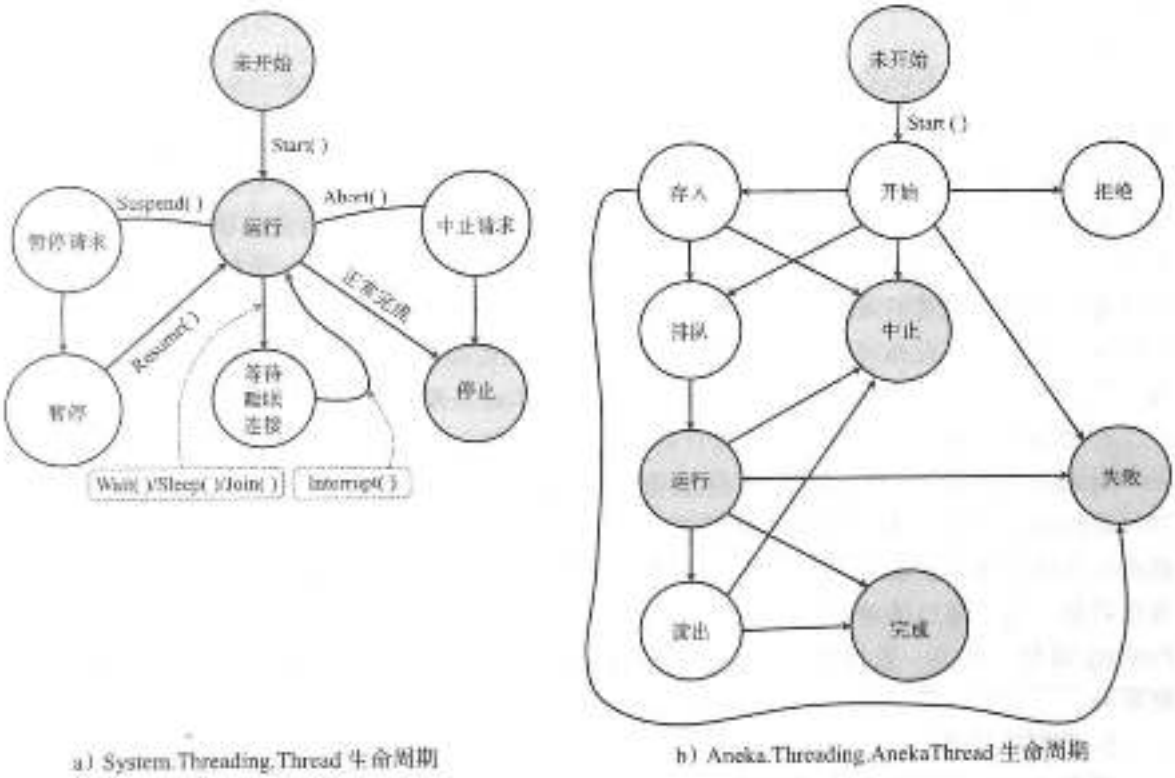
了接口的兼容性而被支持，它们对于线程调度没有什么实际作用，始终用来显示表中的一些值。其他和线程状态有关的属性，比如 IsAlive 和 IsRunning，显示了线程预期的行为，然而相对于 State 属性的 ThreadState 属性实现了一个略有不同的行为。System.Threading.Thread 类（.NET 2.0）中的其余方法不被支持。 191

最后，要注意线程创建的差异是非常重要的。本地隐式线程属于主进程，它们的行动范围限制在进程内。创建本地线程只需要提供一个指针，这个指针指向一个以 ThreadStart 或者 ParameterizedThreadStart 代理的形式来执行的方法。Aneka 线程存在于分布式应用的上下文中，多重分布式应用可以由一个单独进程来管理。基于这个原因，创建线程时应当指定其属于哪个应用。

Aneka 线程 API 和基类库之间的接口兼容性使得对于大部分本地多线程应用来说，只要简单地替换类名并修改线程构造函数就可以快速地完成向 Aneka 的移植。

2. 线程生存周期

因为 Aneka 线程是在分布式环境中生存和执行的，所以它们的生存周期和本地线程必然是不同的，不可能将本地线程的状态值直接映射到 Aneka 线程上。图 6-6 对两者的生命周期进行了比较。



a）System.Threading.Thread 生命周期　　b）Aneka.Threading.AnekaThread 生命周期

图 6-6　线程生命周期比较

图 6-6 中的白球说明这个状态在其他生命周期中并无映射，黑球说明存在相同的状态。除此之外，在本地线程中大部分的状态转变都是由开发者控制的，开发者可以通过调用线程实例中的方法来触发状态转变，然而在 Aneka 线程中，很多状态转变是由中间件来控制的。 192
如图 6-6 所示，Aneka 线程显示了比本地线程更多的状态，因为 Aneka 线程支持由中间件调度的文件暂存，中间件可以让它们排队等候相当一段时间。由于 Aneka 支持为一个特定应

用的线程执行预约节点，因此引入了一个指示因预约失败而造成的执行失败的显式状态。这种情况在线程被送入执行节点时会发生，此时节点需要特定的预约证明才可以执行线程。

一个 Aneka 线程最初以未开始状态创建。一旦 Start() 状态被调用，线程就转变到开始状态，在开始状态下，线程要么直接转换到排队状态，要么在需要为其执行上传文件的情况下转变成存入状态。如果在上传文件时发生错误，那么线程以失败状态终止执行，失败也可能在触发 Started() 时因意外情况而发生。

另一种可能转到拒绝状态的情况是线程获得了一个无效的预约。拒绝是一个最终的状态，它说明线程因权限不足而执行失败。一旦线程位于队列中，如果存在一个可以执行它的自由节点，中间件就把所有对象数据和相关文件移动到远程节点并开始执行，这样就切换到运行状态。如果线程发生异常或者没有得到预期结果，线程的执行就被认为是失败的并且最终状态被设置为失败。如果执行成功，最终状态将是完成。如果有输出文件需要取回，线
193 程状态变为读出，同时文件被收集起来送往它们的最终目标，然后再转变为完成状态。在任何时候，如果开发者停止应用执行或者直接调用 Abort() 方法，线程、并以中止为最终状态。

在大多数情况下，Aneka 线程一般的状态变化过程和本地线程一样：未开始→开始→排队→运行→完成 / 中止 / 失败。

3. 线程同步

.NET 基类库以监视器、信号量和读写锁的方式提供高级设施来支持线程同步，基本的同步是基于语言级的，Aneka 为仅限于实现的线程抽象中的连接操作的线程同步提供了最小支持。.NET 框架提供的大部分结构和类用于提供对不同线程间的共享数据的可控访问权来保护它们的完整性。这种约束在分布式环境中要稍微宽松一点，因为在线程实例间没有共享存储。不仅如此，将本地多线程应用转换成 Aneka 隐式线程的理由，还包含了对可执行大量线程的分布式设施的需求，这些线程可能不是全部在同一时间执行。在这样的环境中提供同步锁机制的协调设施可能会导致难以检测的分布式死锁。因此，Aneka 线程在执行线程间没有特别设置任何同步设施，只是简单地实现了一个连接操作。

4. 线程优先级

System.Threading.Thread 类支持线程优先级机制，这意味着调度的优先级可以在 ThreadPriority 的以下枚举集中选择任意一个值：最高、高于正常、正常、低于正常和最低。然而，并没有要求操作系统承认一个线程的优先级，当前版本的 Aneka 也并不支持线程优先级机制。为了接口的兼容性，Aneka.Threading.Thread 类带有一个类型为 ThreadPriority 的 Priority 属性，其值一般设为正常，对它做出的改变不会对 Aneka 中间件的线程调度产生任何影响。

5. 类型序列化

因为 Aneka 线程在分布式环境中执行，所以以库形式存在的对象代码和实例是通过网络传递的。这个条件给 .NET 框架中最受关注的序列化类型强制增加了一些限制。

本地线程都在同一个地址空间和共享内存中执行，因此，它们不需要对象复制或移动到一个不同的地址空间中。Aneka 线程是分布式的并且在远程的计算节点上执行的，这意味着将要执行的方法相关的对象代码需要通过网络传递。既然代理可以指向实例的方法，那么封闭对象的状态需要在远程执行环境中传递和重构。这属于类级别上的特定功能，术语称为类型序列化。

如果一个.NET类型可以将它的实例类型转换成一个二进制阵列，且阵列中包含了将其从原始形式转换到一个完全不同的执行上下文所需的所有信息，就称它是序列化的。这个特征一般在.NET框架中的几个类型定义中被指明为序列化属性。如果一个类显示了一个特定 194
的特征集，.NET框架会自动给那种类型的实例提供序列化和反序列化的设施。或者说，对于用户定义的任何类型可以实现自定义序列化。

Aneka线程执行以序列化的类型来定义的方法，因为它必须将封闭实例转移到远程的执行方法。在大多数情况下，提供可序列化功能就像在类定义中直接加上序列化属性一样简单。其他的情况下，可能需要实现ISerialization接口并为该类型提供构造函数。这并不是一个很严格的限制，因为只有非常少数的类不可以定义为可序列化的。例如，本地线程、网络连接和流是不可以序列化的，因为它们是直接访问本地资源的，而这些资源不可隐式移动到不同的节点上。

6.4 Aneka线程编程应用

为了证明快速地将普通多线程转换成Aneka线程是可能的，我们提供了前面讨论过的本地线程实例的一个分布式实现。

6.4.1 Aneka线程应用模型

线程编程模型是程序员可以在其中创建以Aneka线程为工作单位的编程模型。因此，需要使用AnekaApplication<W,M>类。这个类是所有此类编程模型的应用参考类。Aneka的API充分利用了泛型并且通过模板特化显示了对不同编程模型的支持。因此，如果想要利用Aneka线程来开发分布式应用程序，有必要依据以下方式来特化模板类型：

```
AnekaApplication<AnekaThread,ThreadManager>
```

这就是所有使用线程编程模型的分布式应用的类的类型。这两个类型是在Aneka SDK的Aneka.Threading.dll库中的Aneka.Threading命名空间中定义的。

另一个应用模型的重要组件是Configuration类，该类是在Aneka.Entity（Aneka.dll）命名空间中定义的。这个类中包含一个属性集，这个属性集允许应用类来配置它和中间件的交互。例如组成了Aneka云主要入口点的Aneka索引服务的地址，需要通过中间件验证的应用程序的用户证书，一些附加的调节参数，以及一个可能会向中间件传达附加信息的属性扩展集。程序6-4中的代码演示了如何创建一个简单的应用实例并且配置它与索引服务在本地的Aneka云进行连接。

程序6-4 应用创建和配置

```
// namespaces containing types of common use
using System;
using System.Collections.Generic;
// common Aneka namespaces
using Aneka;
using Aneka.Util;
using Aneka.Entity;
// Aneka Thread Programming Model user classes
using Aneka.Threading;

// ...

///<summary>
///Creates an instance of the Aneka Application configured to use the
```

```
/// Thread Programming Model.
/// </summary>
/// <returns>Application instance.</returns>
private AnekaApplication<AnekaThread,ThreadManager> CreateApplication()
{
    Configuration conf =new Configuration();
    // this is the common address and port of a local installation
    // of the Aneka Cloud.
    conf.SchedulerUri = new Uri("tcp://localhost:9090/Aneka");
    conf.Credentials =new UserCredentials("Administrator", string.Empty);
    // we will not need support for file transfer, hence we optimize the
    // application in order to not require any file transfer service.
    conf.UseFileTransfer = false;
    // we do not need any other configuration setting

    // we create the application instance and configure it.
    AnekaApplication<AnekaThread,ThreadManager> app =
                    new AnekaApplication<AnekaThread,ThreadManager>(conf);
    return app;
}
```

一旦创建应用，就可以通过指定程序的引用和各线程执行的方法来创建线程。对于应用执行的管理主要集中在对各个线程实例执行的控制。程序 6-5 提供了一个关于如何创建 Aneka 线程的简单例子。

程序 6-5　线程创建和执行

```
// ..... continues from the previous listing
///<summary>
///Thread worker method (implementation skipped).
///</summary>
private void WorkerMethod()
{
    // .....
}
///<summary>
///Creates a collection of threads that are executed in the context of the
/// the given application.
///</summary>
/// <param name="app">Application instance.</param>
private void CreateThreads(AnekaApplication<AnekaThread,ThreadManager> app);
{
    // creates a delegate to the method to execute inside the threads.
    ThreadStart worker = new ThreadStart(this.WorkerMethod);
    // iterates over a loop and creates ten threads.
    for(int i=0; i<10; i++)
    {
        AnekaThread thread = new AnekaThread(worker, app);
        thread.Start();
    }
}
```

195　其余与线程实例管理相关的操作类似于之前讨论的本地多线程应用。

6.4.2　域分解：矩阵乘法

为了将多线程的矩阵乘法转换成 Aneka 线程，需要应用到前面章节的内容。因此，首先通过适当改变 ScalarProduct 类来回顾一下代码。程序 6-6 展示了 ScalarProduct 的一个修

改过的版本。

程序 6-6　ScalarProduct 类（修改过的版本）

```
using System.Runtime.Serialization;

///<summary>
/// Class ScalarProduct. Computes the scalar product between the row and the column
/// arrays. The class uses custom serialization. In order to do so it implements the
/// the ISerializable interface.
///</summary>
[Serializable]
public class ScalarProduct : ISerializable
{
   /// <summary>
   /// Scalar product.
   /// </summary>
   private double result;
   /// <summary>
   /// Gets the resulting scalar product.
   /// </summary>
   public double Result{get { returnthis.result; }}

   /// <summary>
   /// Arrays containing the elements of the row and the column to multiply.
   /// </summary>
   private double[] row, column;

   /// <summary>
   /// Creates an instance of the ScalarProduct class and configures it with the given
   /// row and column arrays.
   /// </summary>
   /// <param name="row">Array with the elements of the row to be multiplied.</param>
   /// <param name="column">Array with the elements of the column to be multiplied.
   /// </param>
   public ScalarProduct(double[] row, double[] column)
   {
      this.row = row;
      this.colum = colum;
   }
   /// <summary>
   /// Deserialization constructor used by the .NET runtime to recreate instances of
   /// of types implementing custom serialization.
   /// </summary>
   /// <param name="info">Bag containing the serialized data instance.</param>
   /// <param name="context">Serialization context (not used).</param>
   public ScalarProduct(SerializationInfo info, StreamingContext context)
   {
      this.result = info.GetDouble("result");
      this.row = info.GetValue("row", typeof(double[])) as double[];
      this.column = info.GetValue("column", typeof(double[])) as double[];
   }
   /// <summary>
   /// Executes the scalar product between the row and the colum.
   /// </summary>
   /// <param name="row">Array with the elements of the row to be multiplied.</param>
   /// <param name="column">Array with the elements of the column to be multiplied.
   /// </param>
   public void Multiply()
   {
      this.result = 0;
      for(int i=0; i<this.row.Length; i++)
```

```
        {
            this.result += this.row[i] * this.column[i];
        }
    }
    /// <summary>
    /// Serialization method used by the .NET runtime to serialize instances of
    /// of types implementing custom serialization.
    /// </summary>
    /// <param name="info">Bag containing the serialized data instance.</param>
    /// <param name="context">Serialization context (not used).</param>
    public ScalarProduct(SerializationInfo info, StreamingContext context)
    {
        this.result = info.GetDouble("result");
        this.row = info.GetValue("row", typeof(double[])) as double[];
        this.column = info.GetValue("column", typeof(double[])) as double[];
    }
    /// <summary>
    /// Executes the scalar product between the row and the colum.
    /// </summary>
    /// <param name="row">Array with the elements of the row to be multiplied.</param>
    /// <param name="column">Array with the elements of the column to be multiplied.
    /// </param>
    public void Multiply()
    {
        this.result = 0;
        for(int i=0; i<this.row.Length; i++)
        {
            this.result += this.row[i] * this.column[i];
        }
    }
    /// <summary>
    /// Serialization method used by the .NET runtime to serialize instances of
    /// of types implementing custom serialization.
    /// </summary>
    /// <param name="info">Bag containing the serialized data instance.</param>
    /// <param name="context">Serialization context (not used).</param>
    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        info.AddValue("result",this.result);
        info.AddValue("row", this.row,typeof(double[]));
        info.AddValue("column", this.column,typeof(double[]));
    }
}
```

这个类已经被标注了序列化属性，同时用实现定制序列化所需要的方法进行了扩展。支持定制序列化意味着：

- 包括 System.Runtime.Serialization 命名空间。
- 实现了 ISerializable 接口。这个接口只有一个 void GetObjectData（SerializationInfo,StreamingContext）方法，在运行时需要序列化实例时调用它。
- 提供一个名为 ScalarProduct（SerializationInfo，StreamContext）的构造函数，这个构造函数会在实例并行化时调用。

196

SerializationInfo 类有一个中心角色，这个角色提供了一个仓库，用来存放定义一个类的所有序列化格式的属性，这些属性可以通过名字来存储并索引。通过框架提供的默认序列化方法，对类的布局可能产生的影响最小。为了利用这个功能，有必要使得一个实例定义状

态的所有属性可以通过 get 和 set 两种方法进行访问。在这种情况下，可以简单地将一个类标记为可序列化的，因为组成这个实例的状态的所有字段都是可以序列化的。可以这么理解，除了序列化之外，没有必要对类的运行方式做出任何改变。

第二步是利用 Aneka 线程改变 MatrixProduct 类。首先应当创建一个配置合理的应用，然后在 System.Treading.Thread 类出现的地方用 Aneka.Threading.Thread 来代替（见程序 6-7）。

程序 6-7 MatrixProduct 类（修改过的版本）

```
using System;
// we do not anymore need the reference to the threading namespace.
// using System.Threading;
using System.Collections.Generic;

// reference to the Aneka namespaces of interest.
// common Aneka namespaces.
using Aneka;
using Aneka.Util;
using Aneka.Entity;
// Aneka Thread Programming Model user classes
using Aneka.Threading;

/// <summary>
/// Class MatrixProduct. Performs the matrix product of two matrices.
/// </summary>
public class MatrixProduct
{
   /// <summary>
   /// First and second matrix of the product.
   /// </summary>
   private static double[,] a, b;
   /// <summary>
   /// Result matrix.
   /// </summary>
   private static double[,] c;
   /// <summary>
   /// Dictionary mapping the thread instances to the corresponding ScalarProduct
   /// instances that are run inside.The occurrence of the Thread class has been
   /// substituted with AnekaThread.
   /// </summary>
   private static IDictionary<AnekaThread, ScalarProduct> workers;
   /// <summary>
   /// Reference to the distributed application the threads belong to.
   /// </summary>
   private static AnekaApplication<AnekaThread, ThreadManager> app;

   /// <summary>
   /// Read the command line parameters and perform the scalar product.
   /// </summary>
   /// <param name="args">Array strings containing the command line parameters.</param>
   public static void Main(string[] args)
   {
      try
      {
         // activates the logging facility.
         Logger.Start();

         // creates the Aneka application instance.
         MatrixProduct.app =Program.CreateApplication();

         // reads the input matrices a and b.
         MatrixProduct.ReadMatrices();
         // executes the parallel matrix product.
         MatrixProduct.ExecuteProduct();
         // waits for all the threads to complete and
         // composes the final matrix.
```

```
            MatrixProduct.ComposeResult();
        }
        catch(Exception ex)
        {
            TCUtil.DumpErrorReport(ex, "Matrix Multiplication - Error executing " +
                                       "the application");
        }
        finally
        {
            try
            {
                // checks whether the application instance has been created
                // stops it.
                if (MatrixProduct.app != null)
                {
                    MatrixProduct.app.Stop();
                }
            }
            catch(Exception ex)
            {
                TCUtil.DumpErrorReport(ex, "Matrix Multiplication - Error stopping " +
                                           "the application");
            }
            // stops the logging thread.
            Logger.Stop();
        }
    }

    /// <summary>
    /// Executes the parallel matrix product by decomposing the problem in
    /// independent scalar product between rows and columns.
    /// </summary>
    private static void ExecuteThreads()
    {
        // we replace the Thread class with AnekaThread.
        MatrixProduct.workers = new Dictionary<AnekaThread, ScalarProduct>();
        int rows = MatrixProduct.a.Length;
        // in .NET matrices are arrays of arrays and the number of columns is
        // is represented by the length of the second array.
        int columns = MatrixProduct.b[0].Length;

        for(int i=0; i<rows; i++)
        for(int j=0; j<columns; j++)
        {
            double[] row = MatrixProduct.a[i];
            // because matrices are stored as arrays of arrays in order to
            // to get the columns we need to traverse the array and copy the
            // the data to another array.
            double[] column = new double[common];
            for(int k=0; k<common; k++)
            {
                column[i] = MatrixProduct.b[j][i];
            }
            // creates a ScalarProduct instance with the previous rows and
            // columns and starts a thread executing the Multiply method.
            ScalarProduct scalar = newScalarProduct(row, column);
            // we change the System.Threading.Thread class with the corresponding
            // Aneka.Threading.AnekaThread class and reference the application instance.
            AnekaThread worker = newAnekaThread(newThreadStart(scalar.Multiply), app);
            worker.Name =string.Format("{0},{1}",row,column);
            worker.Start();
            // adds the thread to the dictionary so that it can be
            // further retrieved.
            MatrixProduct.workers.Add(worker, scalar);
        }
```

```
}
/// <summary>
/// Waits for the completion of all the threads and composes the final
/// result matrix.
/// </summary>
private static void ComposeResult()
{
    MatrixProduct.c = new double[rows,columns];
    // we replace the Thread class with AnekaThread.
    foreach(KeyValuePair<AnekaThread,ScalarProduct>pair in MatrixProduct.workers)
    {
        AnekaThread worker = pair.Key;
        // we have saved the coordinates of each scalar product in the name
        // of the thread now we get them back by parsing the name.
        string[] indices = string.Split(worker.Name, new char[] {','});
        int i = int.Parse(indices[0]);
        int j = int.Parse(indices[1]);
        // we wait for the thread to complete
        worker.Join();
        // instead of using the local value of the ScalarProduct instance
        // we use the one that has is stored in the Target property.
        // MatrixProduct.c[i,j] = pair.Value.Result;
        MatrixProduct.c[i,j] = ((ScalarProduct) worker.Target).Result;
    }
    MatrixProduct.PrintMatrix(MatrixProduct.c);
}
/// <summary>
/// Reads the matrices.
/// </summary>
private static void ReadMatrices()
{
    // code for reading the matrices a and b
}
/// <summary>
/// Prints the given matrix.
/// </summary>
/// <param name="matrix">Matrix to print.</param>
private static void PrintMatrices(double[,] matrix)
{
    // code for printing the matrix.
}
/// <summary>
/// Creates an instance of the Aneka Application configured to use the
/// Thread Programming Model.
/// </summary>
/// <returns>Application instance.</returns>
private AnekaApplication<AnekaThread,ThreadManager> CreateApplication();
{
      Configuration conf =new Configuration();
      // this is the common address and port of a local installation
      // of the Aneka Cloud.
      conf.SchedulerUri = new Uri("tcp://localhost:9090/Aneka");
      conf.Credentials =new UserCredentials("Administrator", string.Empty);
      // we will not need support for file transfer, hence we optimize the
      // application in order to not require any file transfer service.
      conf.UseFileTransfer = false;
      // we do not need any other configuration setting

      // we create the application instance and configure it.
      AnekaApplication<AnekaThread,ThreadManager> app =
                  new AnekaApplication<AnekaThread,ThreadManager>(conf);
      return app;
   }
}
```

如程序 6-7 所示，需要对程序逻辑做出的改变是最小的，大部分的修改是有关异常管理和 Aneka 记录设施的合理使用。MatrixProduct 类集成了之前章节讨论过的应用创建和设置相关的方法，引入了一个 try...catch...finally 块来处理应用执行时发生的异常。除了重命名 Thread 类的地方以外，代码的其余部分是不需要改变的。

[197] 只有一个需要注意的重要改变：一旦 Aneka 线程实例完成，对包含远程执行方法的对象更新后的引用由 AnekaThread.Target 属性显示，而不是由引用最初用于创建委托的对象的局部变量来显示。

6.4.3　功能分解：Sine、Cosine、Tangent

将此例子移植到 Aneka 线程所要做出的改变和之前讨论的例子基本一样。在这个例子中只有一个显著的不同点：各个线程都有一个指向委托的引用，用来在计算结束时更新全局的和。因为我们是在分布式环境中操作的，所以对象操作的实例并不是线程共享的，但是各个线程实例有自己的本地副本。这样就阻止了在主线程中更新全局的和，并且需要对应用的更新策略做出调整。

这个例子还阐述了如何修改 Sine、Cosine、Tangent 类，以便让它们利用框架默认的序列化功能（见程序 6-8）。

程序 6-8　数学函数（修改过的版本）

```
using System;
// we do not anymore need the reference to the threading namespace.
// using System.Threading;
using System.Collections.Generic;

// reference to the Aneka namespaces of interest.
// common Aneka namespaces.
using Aneka;
using Aneka.Util;
using Aneka.Entity;
// Aneka Thread Programming Model user classes
using Aneka.Threading;


// this is not needed anymore.
// /// <summary>
// /// Delegate UpdateResult. Function pointer that is used to update the final result
// /// from the slave threads once the computation is completed.
// /// </summary>
// /// <param name='x'>partial value to add.</param>
// public delegate void UpdateResult(double x);

/// <summary>
/// Class Sine. Computes the sine of a given value.
/// </summary>
[Serializable]
public class Sine
{
    /// <summary>
    /// Input value for which the sine function is computed.
    /// </summary>
    private double x;
    /// <summary>
    /// Gets or sets the input value of the sine function.
    /// </summary>
    public double X { get { return this.x; } set { this.x = value; } }
    /// <summary>
    /// Result value.
    /// </summary>
```

```
    private double y;
    /// <summary>
    /// Gets or sets the result value of the sine function.
    /// </summary>
    public double Y { get { return this.y; } set { this.y = value; } }
    // we can't use this anymore.
    // /// <summary>
    // /// Function pointer used to update the result.
    // /// </summary>
    // private UpdateResult updater;

    // We need a default constructor, which is automatically provided by the compiler
    // if we do not specify any constructor.
    // /// <summary>
    // /// Creates an instance of the Sine and sets the input to the given angle.
    // /// </summary>
    // /// <param name="x">Angle in radians.</param>
    // /// <param name="updater">Function pointer used to update the result.</param>
    // public Sine(double x, UpdateResult updater)
    // {
    //     this.x = x;
    //     this.updater = updater;
    // }
    /// <summary>
    /// Executes the sine function.
    /// </summary>
    public void Apply()
    {
        this.y = Math.Sin(this.x);
        // we cannot use this anymore because there is no
        // shared memory space.
        // if (this.updater != null)
        // {
        //     this.updater(this.y);
        // }
    }
}

///<summary>
/// Class Cosine. Computes the cosine of a given value. The same changes have been
/// applied by removing the code not needed anymore rather than commenting it out.
///</summary>
[Serializable]
public class Cosine
{
    /// <summary>
    /// Input value for which the cosine function is computed.
    /// </summary>
    private double x;
    /// <summary>
    /// Gets or sets the input value of the cosine function.
    /// </summary>
    public double X { get { return this.x; } set { this.x = value; } }
    /// <summary>
    /// Result value.
    /// </summary>
    private double y;
    /// <summary>
    /// Gets or sets the result value of the cosine function.
    /// </summary>
    public double Y { get { return this.y; } set { this.y = value; } }
    /// <summary>
    /// Executes the cosine function.
    /// </summary>
    public void Apply()
    {
```

```
        this.y = Math.Cos(this.x);
    }
}

///<summary>
/// Class Cosine. Computes the cosine of a given value. The same changes have been
/// applied by removing the code not needed anymore rather than commenting it out.
///</summary>
[Serializable]
public class Tangent
{
    /// <summary>
    /// Input value for which the tangent function is computed.
    /// </summary>
    private double x;
    /// <summary>
    /// Gets or sets the input value of the tangent function.
    /// </summary>
    public double X { get { return this.x; } set { this.x = value; } }
    /// <summary>
    /// Result value.
    /// </summary>
    private double y;
    /// <summary>
    /// Gets or sets the result value of the tangent function.
    /// </summary>
    public double Y { get { return this.y; } set { this.y = value; } }
    /// <summary>
    /// Executes the tangent function.
    /// </summary>
    public void Apply()
    {
        this.y = Math.Cos(this.x);
    }
}

///<summary>
/// Class Cosine. Computes the cosine of a given value. The same changes have been
/// applied by removing the code not needed anymore rather than commenting it out.
///</summary>
[Serializable]
public class Tangent
{
    /// <summary>
    /// Input value for which the tangent function is computed.
    /// </summary>
    private double x;
    /// <summary>
    /// Gets or sets the input value of the tangent function.
    /// </summary>
    public double X { get { return this.x; } set { this.x = value; } }
    /// <summary>
    /// Result value.
    /// </summary>
    private double y;
    /// <summary>
    /// Gets or sets the result value of the tangent function.
    /// </summary>
    public double Y { get { return this.y; } set { this.y = value; } }
    /// <summary>
    /// Executes the tangent function.
    /// </summary>
    public void Apply()
    {
        this.y = Math.Tan(this.x);
    }
```

```
}

/// <summary>
/// Class Program. Computes the function sin(x) + cos(x) + tan(x).
/// </summary>
public class Program
{
    /// <summary>
    /// Variable storing the computed value for the function.
    /// </summary>
    private static double result;

    // we do not need synchronization anymore, because the update of the global
    // sum is done sequentially.
    // /// <summary>
    // /// Synchronization instance used to avoid keeping track of the threads.
    // /// </summary>
    // private static object synchRoot = new object();

    /// <summary>
    /// Reference to the distributed application the threads belong to.
    /// </summary>
    private static AnekaApplication<AnekaThread, ThreadManager> app;

    /// <summary>
    /// Read the command line parameters and perform the scalar product.
    /// </summary>
    /// <param name="args">Array strings containing the command line parameters.</param>
    public static void Main(string[] args)
    {
        try
        {
            // activates the logging facility.
            Logger.Start();
            // creates the Aneka application instance.
                app = Program.CreateApplication();

            // gets a value for x
                double x = 3.4d;

                // creates the function pointer to the update method.
            UpdateResult updater = new UpdateResult(Program.Sum);

            // creates the sine thread.
                Sine sine = new Sine(x, updater);
            AnekaThread tSine = new AnekaThread(new ThreadStart(sine.Apply), app);

            // creates the cosine thread.
            Cosine cosine = new Cosine(x, updater);
            AnekaThread tCosine = new AnekaThread(new ThreadStart(cosine.Apply), app);

            // creates the tangent thread.
            Tangent tangent = new Tangent(x, updater);
            AnekaThread tTangent = new AnekaThread(new ThreadStart(tangent.Apply), app);

            // shuffles the execution order.
                tTangent.Start();

        tSine.Start();
        tCosine.Start();

        // waits for the completion of the threads.
        tCosine.Join();
        tTangent.Join();
        tSine.Join();
```

```
        // once we have joined all the threads the values have been collected back
        // and we use the Target property in order to obtain the object with the
        // updated values.
        sine = (Sine) tSine.Target;
        cosine = (Cosine) tSine.Target;
        tangent = (Tangent) tSine.Target;

        Program.result = sine.Target.Y + cosine.Y + tangent.Y;

        // the result is available, dumps it to console.
        Console.WriteLine("f({0}) = {1}", x, Program.result);
    }
    catch(Exception ex)
    {
        IOUtil.DumpErrorReport(ex, "Math Functions - Error executing " +
                               "the application");
    }
    finally
    {
        try
        {
            // checks whether the application instance has been created
            // stops it.
            if (app != null)
            {
                app.Stop();
            }
        }
        catch(Exception ex)
        {
            IOUtil.DumpErrorReport(ex, "Math Functions - Error stopping " +
                                   "the application");
        }
        // stops the logging thread.
        Logger.Stop();
    }
}
// we do not need this anymore.
// /// <summary>
// /// Callback that is executed once the computation in the thread is completed
// /// and adds the partial value passed as a parameter to the result.
// /// </summary>
// /// <param name="partial">Partial value to add.</param>
// private static void Sum(double partial)
// {
//     lock(Program.synchRoot)
//     {
//         Program.result += partial;
//     }
// }
    /// <summary>
    /// Creates an instance of the Aneka Application configured to use the
    /// Thread Programming Model.
    /// </summary>
    /// <returns>Application instance.</returns>
    private AnekaApplication<AnekaThread,ThreadManager> CreateApplication();
    {
        Configuration conf = new Configuration();
        // this is the common address and port of a local installation
        // of the Aneka Cloud.
        conf.SchedulerUri = new Uri("tcp://localhost:9090/Aneka");
        conf.Credentials = new UserCredentials("Administrator", string.Empty);
        // we will not need support for file transfer, hence we optimize the
        // application in order to not require any file transfer service.
        conf.UseFileTransfer = false;
```

```
    // we do not need any other configuration setting

    // we create the application instance and configure it.
    AnekaApplication<AnekaThread,ThreadManager> app =
                        new AnekaApplication<AnekaThread,ThreadManager>(conf);
    return app;
}
```

这个例子演示了如果辅助方法在有指向本地对象（这些本地对象作为执行结果来进行更新）的引用的线程中执行，该如何改变应用逻辑。为了让具有 Aneka 线程的此类应用能够执行，有必要从辅助方法中抽出更新的逻辑并使之在主线程中实现。

本章小结

这一章简要介绍了多线程编程以及在单台机器上运行多进程所需的技术。我们介绍了多核技术的基础，它是在单台机器上实现并行化的最新技术，并且讨论了如何利用这样的并行化来加速使用多线程编程的应用。一个线程定义了一个进程中的单个控制流，而进程又是在现代操作系统中代表一个运行程序的逻辑单元。当前几乎所有的主流操作系统都支持多线程，无论底层的硬件是否显式支持真正的并行。真正的并行是同时由多个可用的处理器或处理器核来实现的，或者通过在同一个处理单元上交叉执行多个线程来实现多线程。

为了支持多线程编程，编程语言在它们的类库中定义了进程和线程的抽象。关于线程和线程同步操作的一个上流标准是 POSIX，它被所有的 Linux/UNIX 操作系统支持，同时作为一个附加库对 Windows 操作系统系列可用。POSIX 的通常实现是在 C/C++ 中以函数库形式提供。新一代的语言（例如 Java 和 C#(.NET)）为线程管理和同步提供了一个兼容的抽象集，并且遵循这些语言特有的面向对象设计方法。这些实现对于任何提供了这些语言所需运行时环境实现的操作系统都是可移植的。

多线程编程是一种允许在单机范围内实现并行化的实践。常规的多线程编程不能满足有高度并行需求的应用，这个时候就需要依赖分布式基础设施，例如集群、网格或最近常提到的云。要使用这些设施就有必要重新设计应用并使用特定的 API，这就可能需要对现有的程序做出显著的改变。为了解决这些问题，Aneka 提出了线程编程模型，它扩展了多线程编程的原理，使之能够超越单节点的界限并利用异构的分布式基础设施来执行。为了最大限度地减少应用程序转换，线程编程模型模仿了 System.Threading 命名空间的 API，由于线程在分布式设施上执行，所以这样的模仿自然存在局限。高吞吐量的程序可以很容易地在最小程度或没有改变其原有逻辑的情况下移植到 Aneka 线程。这类功能的例子以及转换本地多线程应用到 Aneka 线程的基本步骤，在之前章节中讨论并行问题的域和功能分解时已经给出。

作为一个分布式编程框架，Aneka 提供了很多内置的功能，但是这些功能在架设一个并发线程意义上的应用时一般不会使用。这些功能包括对事件通知和文件传输的支持，它们作为 Aneka 程序模型的核心功能是可用的，但是在线程编程模型中没有体现，该模型对算法的执行进行了分割以加速执行。然而，它们确实在下一章讨论的“任务包”应用中发挥了巨大作用。

198 ↓ 203

204 ↓ 209

习题

1. 什么是吞吐量计算？它旨在实现什么目的？
2. 什么是多重处理？描述实现多重处理的不同技术。
3. 什么是多核技术？它与多重处理有什么联系？
4. 简要描述多核系统的架构。
5. 什么是多任务？
6. 什么是多线程？它与多任务的联系是什么？
7. 描述进程和线程的关系。
8. 应用的并行性依赖于并行硬件架构吗？
9. 从编程的角度以及并行应用执行时线程使用的角度描述线程的主要特点。
10. 什么是 POSIX？
11. 描述在新一代语言（如 Java 或 C#）中给予线程编程的支持。
12. 逻辑线程和物理线程分别指什么？
13. 对于线程实现的通常操作是什么？
14. 描述定义计算机算法并行实现的两个关键技术。
15. 什么是高度并行问题？
16. 描述如何使用域分解实现并行矩阵标量积。
17. 通信如何影响并行或分布式算法的设计与实现？
18. Aneka 为多线程提供哪些支持？
19. 描述 Aneka 线程和本地线程的主要不同点。
20. 线程编程模型的局限是什么？
21. 设计使用简单线程制表的高斯函数的并行实现，而后将其转换为 Aneka 线程。

210

高吞吐量计算：任务编程

任务计算是涵盖了几种不同模型架构设计的分布式应用程序的分布式系统编程，这些应用程序最终都基于相同的基本抽象：任务。一个任务通常代表一个程序，需要输入文件，并生成输出文件作为其执行的结果。应用程序由一系列任务组成。这些任务在提交后执行，执行结束时可收集其输出数据。通过生成任务的方式、任务执行顺序以及任务中是否需要交换数据，可以区分任务编程类型的不同应用程序模型。

本章描述了任务的抽象化，并简要介绍了基于任务抽象的分布式应用程序模型。通常把 Aneka 任务编程模型作为参考来说明任务包（BoT）应用程序在一个分布式基础设施上的执行。

7.1 任务计算

对于并发并行和分布式计算应用程序，在任务方面组织一个应用程序是最直观、最普遍的做法。一个任务标识一个或多个操作，不同的操作产生不同的输出，并且这些输出可以被分离为单个逻辑单元。在实践中，一个任务被表示为一个代码不同的单元，或一个程序，它可以分离，并可以在一个远程的运行环境中执行。程序是解析任务最常见的选择，尤其是在科学计算领域，该领域为了方便计算已经使用了分布式计算。

多线程编程主要涉及在一台机器内提供并行支持。任务计算通过利用多个计算节点的计算能力来实现分布式计算。因此，分布式基础设施是明确存在于该模型中的。之前用于执行任务的基础设施是集群、超级计算机和计算网格。为了获得执行分布式应用程序所需要的巨大计算能力，云已成为一个有吸引力的解决方案。为了实现这一目标，需要合适的中间件。图 7-1 展示了利用任务计算的一个参考场景。

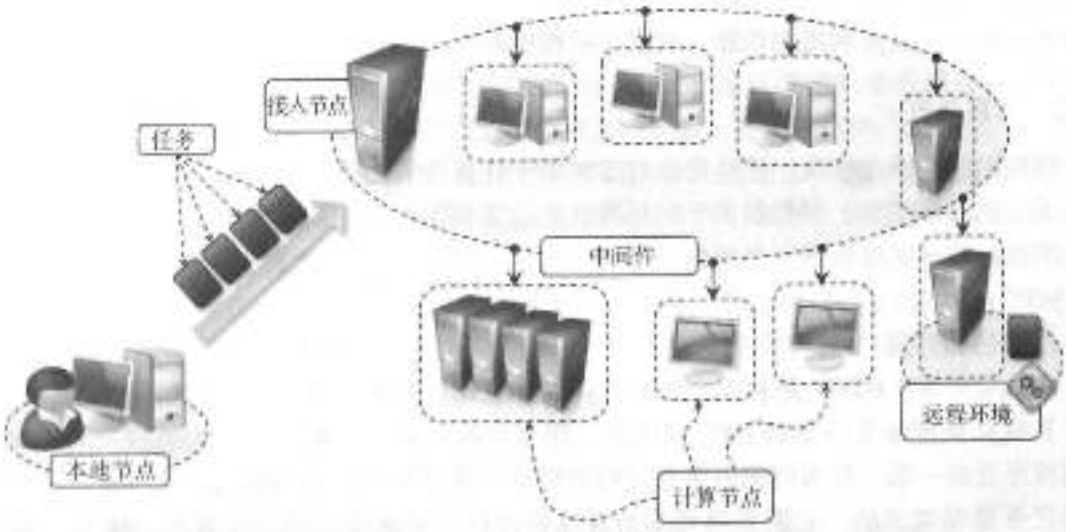


图 7-1 任务计算场景

211 中间件是一个软件层，能够协调使用从数据中心或地理上分散的网络计算机中提取的多种资源。用户将任务集合提交到中间件的接入点，这需要注意调度和监控任务的执行。每个计算资源提供了一个适当的运行环境，不同实现（简单的 shell、沙箱环境或虚拟机）之间可能会有所不同。无论是 Web 还是编程语言接口，通常都采用中间件提供的 API 提交任务、监控任务状态和收集结果。

由于任务的抽象性，在任务计算的保护下存在着不同的分布式应用程序模型。这个特性使得识别一组常用操作成为可能，中间件支持创建和执行基于任务的应用程序，主要包括：

- 协调和调度任务在一组远程节点上的执行。
- 移动程序到远程节点并管理它们的依赖关系。
- 创建在远程节点上执行任务的环境。
- 监控每个任务的执行，并通知用户其执行状态。
- 访问由任务产生的输出。

任务计算模型可能会因任务调度方式而有所不同，而这又取决于任务是否相互关联，或是否需要彼此沟通。

7.1.1 任务特性

一个任务是识别由具有有形输出的分布式应用程序的计算单元构成的一个程序或一组程
212 序的一般抽象。任务表示逻辑上可分离并分别执行的应用程序组件。分布式应用程序由任务组成，任务的集体执行和相互关系定义了应用程序的性质。一个任务可以由不同的元素来表示：

- 几个应用程序共同执行组成的一个 shell 脚本。
- 单个程序。
- 在一个特定运行环境的上下文中执行的代码单元（一个 Java/C++/.NET 类）。

任务通常的特征是输入文件、可执行代码（程序、shell 脚本等）和输出文件。在许多情况下，任务执行的常见运行环境是操作系统或等效的沙箱环境。除了可以传送到该节点的库依赖之外，任务也需要远程执行节点上的特定软件设施。

一些分布式应用程序可能有额外的限制。例如，在编程水平上（通过类继承或接口实现）描述任务抽象性的分布式计算框架，可能需要额外的限制（即符合继承规则），以及一组更丰富的可供开发人员利用的功能。根据应用程序的具体模型，任务之间可能有依赖关系。

7.1.2 计算类别

根据问题的特定性质，已经提出过多种用于任务计算的类别。这些类别不强制要求任何特定的应用程序模型，但提供关于问题特性的总体视图。它们隐式地将要求强加给基础设施和中间件。这一类应用程序主要包括高性能计算（HPC）、高吞吐量计算（HTC）和多任务计算（MTC）。

1. 高性能计算

高性能计算（HPC）是利用分布式计算设施来解决需要大量计算能力的问题。以往，超级计算机和集群是专门支持 HPC 应用的，用于解决科学与工程中的“大挑战”问题。HPC 应用程序是由一系列在短时间内需要进行处理的计算密集型任务构成的。这种并行和紧密耦合的任务是很常见的，它需要低延迟的互连网络以尽量减少数据交换时间。用来评估 HPC

系统的指标是浮点运算/秒（FLOPS），兆次浮点运算/秒甚至千兆次浮点运算/秒，表示计算系统每秒可以执行浮点运算的次数。

2. 高吞吐量计算

高吞吐量计算（HTC）是利用分布式计算设施来实现在长时间内要求大量计算能力的应用。HTC 系统必须具有足够的健壮性，并且能在一段较长的时间上进行可靠操作。传统上，异构资源（集群、工作站和志愿者台式机）组成的计算网格已用于支持 HTC。HTC 应用程序是由大量任务组成的，这些任务的执行可能会持续相当长的时间（几周或数月）。科学模 213
拟和统计分析是这种应用的典型例子。可以在分布式资源中调度的独立任务是很常见的，因为它们不需要进行通信。HTC 系统以每月完成的作业数来衡量性能。

3. 多任务计算

多任务计算（MTC）[61] 最近才开始受到关注并得到广泛应用。它旨在消除 HPC 和 HTC 之间的差距。MTC 类似于 HTC，但是它着重于在短时间内使用许多计算资源来完成许多计算任务。简言之，MTC 表示通过文件系统操作连接多个不同活动的高性能计算。MTC 的特征是任务的异质性，即可能会有大量不同的性质：大任务或小任务，单处理器或多处理器，计算密集型或数据密集型，静态或动态，同构或异构等。MTC 应用程序是松耦合的，一般是通信密集型，但不能通过 HPC 领域中常见的消息传递接口来自然表达。需要注意许多计算是多样化的但不是高度并行的。考虑到 MTC 通常由大量的任务构成，所以任何具有大量计算元件的分布式设施也能够支持 MTC。这些设施包括超级计算机、大型集群和新兴的云基础设施。

7.1.3 任务计算框架

一些框架可以支持基于任务的应用程序在包括云在内的分布式计算资源上执行。支持任务计算框架的大众软件有 Condor[5]、Globus 工具包 [12]、Sun Grid Engine（SGE）[13]、BOINC[14]、Nimrod/G[164] 和 Aneka。

这些系统的架构类似于图 7-1 所示的通用参考架构。它们包括两个主要组件：调度节点（一个或多个）和工作节点。系统组件的组成可能会发生变化。例如，在层次结构中可以包含多个调度节点。这种配置在计算网格的中间件中相当普遍，它利用一个或多个组织或站点的各种分布式资源。每个站点可以有自己的调度引擎，特别是当该系统有助于网格计算而且能为本地用户服务的时候。

一个典型的例子是集群的设置，系统可能安装 Condor 或 SGE 进行批处理作业提交，这些服务通常在站点使用，但集群能够集成到一个更大的网格，在该网格中，GRAM[⊖]（Globus 资源分配管理器）等元调度器可以把作业集合调度到集群。其他选项包括没有任何调度能力的网关节点，它们简单地构成系统的接入点。这些节点有索引服务，允许用户确定系统中的可用资源，以及其当前状态和可用调度器。工作节点一般提供一个沙箱环境，在沙箱环境 214
中，任务的执行是代表特定用户或在特定的安全上下文中，限制程序执行的操作，如文件系统访问。文件传输也是这些系统支持的基本特征。集群通常配有共享文件系统和并行 I/O 设备。网格为用户提供各种临时设施，如凭据访问远程工作节点或透明地从用户的本地计算机将文件移动到远程节点的自动化升级服务。

⊖ GRAM 是 Globus 工具集的一个组件，负责网格计算系统中的作业定位、提交、监控和取消。

Condor 可能是使用最广泛、存在时间最长的用于管理集群、闲置工作站和集群集合的中间件。Condor-G 是 Condor 的一个版本，支持由 Globus 管理的网格计算资源集成。Condor 支持批处理队列系统的共同特点，以及检查点工作的能力和管理超载节点。它提供了一个功能强大的资源匹配机制，只对具备合适运行环境的资源进行调度。Condor 可以处理大量资源的串行和并行作业，用于工业界、政府和学术界的很多组织，管理从几个到上千个工作站的基础设施。

Sun 网格引擎（SGE），即现在的 Oracle 网格引擎，是工作量和分布式资源管理的中间件。最初开发 SGE 是为了支持集群上作业的执行，因此集成了额外的功能，现在 SGE 能够管理异构资源，并构成网格计算的中间件。它支持并行、串行、交互和参数化作业的执行，并具有先进的调度功能，例如基于预算和组的调度、有期限地调度应用程序、自定义策略以及提前预订。

Globus 工具包是一组技术的集合，能够进行网格计算。它提供了一套全面的工具，用于跨企业、机构和地理边界共享计算能力、数据库和其他服务，无需影响本地自治。该工具包提供软件服务、库和资源监测、发现和管理工具，以及安全性和文件管理工具。Globus 工具包解决了网格计算的核心问题：管理分布在不同组织的异构资源组成的分布式环境，同时考虑了安全性和互操作性方面的所有隐含条件。为了在这种情况下提供对网格计算的有效支持，该工具包定义了接口集合和互操作协议，使不同的系统能够相互整合，并获得其边界之外的资源。

Nimrod/G[164] 是在全球计算网格内自动建模和执行参数化应用程序的工具。它提供了一种简单的参数化建模声明语言，用于表述参数性实验。领域专家可以很容易地创建一个参数化的实验计划，并使用 Nimrod/G 系统部署在分布式资源上执行的作业。近年来它已广泛用于各种应用程序，从量子化学到政策和环境。此外，它还采用了基于经济原理的新型资源管理和调度算法。具体来说，它支持在分布式网格资源上对应用程序进行期限和预算约束的调度，以尽量减少执行成本，并且能够及时交付成果。

用于网络计算的伯克利开放式架构（BOINC）是志愿计算和网格计算的框架。它使得无效的台式机变成可被用于运行作业的志愿计算节点。BOINC 由两个主要部分组成：BOINC 服务器和 BOINC 客户端。前者是中心节点，用于跟踪所有可用的资源和作业调度；后者是部署在台式机上，为作业提交创建 BOINC 执行环境的软件组件。鉴于 BOINC 客户端的波动性，BOINC 支持作业检查点和复制。即使主要集中在志愿计算，BOINC 系统也可以方便地建立，通过利用专用机创建计算网格为作业的执行提供更加稳定的环境支持。要充分利用 BOINC，需要建立一个应用程序项目。在安装 BOINC 客户端时，用户可以决定将计算机 CPU 周期指助给哪个应用程序项目。目前，医药学、天文学和密码学等项目都在 BOINC 基础设施上运行。

215

7.2　基于任务的应用模型

很多基于任务概念的模型都是构成分布式应用程序的基本单元。使这些模型相互不同的是任务生成方式、任务之间的联系、任务间依赖性的存在及其他条件，例如在运行环境中的一组特定的服务。本节快速回顾一下基于任务最常见的模型。

7.2.1 高度并行应用

高度并行应用构成了分布式应用中最简单也最直接的一类。正如第6章讨论过的，高度并行应用由一系列相互独立且可以按任何次序执行的任务构成。各任务的类型可能相同也可能不同，它们之间可能不需要通信。

该类应用被大多数分布式计算框架所支持。由于任务间不需要通信，因此任务的调度具有很大的自由度。任务可以在任意时间执行，并不要求某些任务必须同时执行。因此，该类应用的调度相对简单，主要是关于任务与可用资源间的优化映射。支持高度并行框架的应用包括 Globus 工具包、BOINC 和 Aneka。

高度并行处理的问题模型包括图像/视频渲染、演进优化和模型预测。在图像和视频渲染方面，任务分别表示对像素（图像的一部分区域）或者帧的渲染。对于演进优化的元启发式算法和模型预测应用而言，任务表示该算法在给定参数下的单次执行。总之，科学应用构成了很多高度并行应用的场景，尽管这些应用可以被更具体地划分到参数化应用当中。 216

7.2.2 参数化应用

参数化应用是高度并行应用的一个分支，所划分的任务本质相同只是执行参数不同。参数化任务由模板任务和参数集所定义。模板任务定义了远程节点上所需要执行的操作。模板任务需要参数，参数集是由将模板任务配置为计算实例的变量构成的集合。参数以及它们各自的可行域构成了应用多维域，多维域中的每个点代表了一个任务实例。

任何支持高度并行应用（任务可以相互独立执行）的分布式计算框架也支持参数化应用的执行。参数化应用相对于高度并行应用的不同在于，所有的执行任务通过对参数集中所有可能的合理组合遍历生成。上述操作可以由框架本身或者分布式计算的中间件完成。例如，Nimrod/G 支持参数化应用的执行。Aneka 提供了一个基于客户端的工具，可以可视化地编写模板任务，设定参数集以及遍历此类参数的所有可能组合。

参数化应用类型众多，大部分来自科学计算领域，如演进优化算法、天气预报模型、流体力学应用、蒙特卡罗方法等。例如，在演进式算法中可以将相关计算参数的集合定义为应用的域。在遗传算法中，优化群包含一定数量的个体，优化器具有设定的迭代次数。使用参数化方法执行遗传演进算法（优化群的数量和迭代次数可人为设置）的伪代码如下：

```
individuals = [100, 200, 300, 500, 1000]
generations = [50, 100, 200, 400]
foreach indiv in individuals do
foreach generation in generations do

 task = generate_task(indiv, generation)
 submit_task(task)
```

示例中描述的方法定义了一个由离散变量参数构成的二维域，之后对所有群体数量和迭代次数的组合进行遍历，生成应用中的全部任务。本例中有 20 个任务生成。generate_task 函数由应用决定，通过替换模板定义中相应变量的 indiv 和 generation 值来创建任务实例。函数 submit_task 由中间件定义，并执行实际的任务提交。

总体而言，模板任务由遗留应用（具有恰当参数）的执行操作和和执行数据迁移的文件系统操作组成。因此，对参数化应用提供原生支持的框架通常提供了一系列文件操作命令。 217
此外，模板任务通常记录在一个文件中，并包含了系统所提供的命令。通常系统可用的命令

如下：

- 执行。在远程节点上执行一段代码。
- 复制。从远程节点导入或导出文件。
- 替换。用参数在文件中的占位符替换参数值。
- 删除。删除文件。

上述所有命令可以带参数操作，用每个任务实例的实际值替换参数。

图 7-2 和图 7-3 定义了两个可能的任务模板，前者根据 Nimrod/G 的符号体系定义，后者是 Aneka 所需要的格式。

```
parameter x float range from 1 to 10 step 1;
parameter y float range from -4 to 5 step 1;

task main
  node:execute /bin/echo X:$(x) Y:$(y) > output
  copy node:output output.`expr $(y)\*10-$(x)`
endtask
```

图 7-2　Nimrod/G 任务模板定义

```
<psm>
  <name>Aneka Blast</name>
  <description>BLAST simulation</description>
  <workspace>C:\Projects\Explorer\blast</workspace>
  <parameters>
    <single name="p" type="String" comment="The name of the program" value="blastn"/>
    <single name="d" type="String" comment="The database file" value="ecoli.nt"/>
    <range name="s" type="String" comment="The sequence file" from="0" to="2" interval="1"/>
  </parameters>
  <sharedFiles>
    <file path="blastall.exe" vpath="blastall.exe"/>
    <file path="ecoli.nt.nhr" vpath="ecoli.nt.nhr"/>
    <file path="ecoli.nt.nin" vpath="ecoli.nt.nin"/>
    <file path="ecoli.nt.nnd" vpath="ecoli.nt.nnd"/>
    <file path="ecoli.nt.nni" vpath="ecoli.nt.nni"/>
    <file path="ecoli.nt.nsd" vpath="ecoli.nt.nsd"/>
    <file path="ecoli.nt.nsi" vpath="ecoli.nt.nsi"/>
    <file path="ecoli.nt.nsq" vpath="ecoli.nt.nsq"/>
  </sharedFiles>
  <task>
    <inputs>
      <file path="seq($s).txt" vpath="seq($s).txt"/>
    </inputs>
    <outputs>
      <file path="output($s).txt" vpath="output($s).txt"/>
    </outputs>
    <commands>
      <execute cmd="blastall.exe" args="-p ($p) -d ($d) -i seq($s).txt -o output($s).txt"/>
    </commands>
  </task>
</psm>
```

图 7-3　参数化文件

模板文件由两个部分构成，header 部分用于参数的定义，任务定义部分使用了系统 shell 命令和 Nimrod/G 命令，“node：”前缀定义了执行任务的远程节点，${...} 前缀定义了参数。

示例介绍了远程执行 echo 命令，并将命令的本地输出结果保存在文件中（文件名由参数 x、y 决定），然后复制到本地用户目录。

Aneka 参数化文件定义了执行示例中的 BLAST 应用的模板文件。该文件是一个 XML 文档，包含 sharedFiles、parameters 和 task 等部分。parameters 中包含了定义模板任务所需参数的定义，即定义了 single 和 range 两个属性。sharedFile 部分定义了执行任务所需的文件。task 部分描述了模板任务的具体操作。除了包含命令集之外（本例中只有一条简单的 execute 命令），task 也包含一系列本地和远程路径来定义输入、输出文件。对于图 7-2 所示的例子，不必显式地将文件发送至远程（节点）目的地，Aneka 可以自动执行该操作。

7.2.3 消息传递接口应用

消息传递接口（MPI）是开发通过交换消息进行通信的并行程序的规范。相对于其他早期模型，MPI 在其分布式任务中引入了通信约束使任务必须同时执行。MPI 规范已经联合了几个支持分布式计算（支持分布式存储器共享和消息传递）的平台，致力于创造一种共同规范。目前，MPI 事实上已经成为了开发可移植的、高效消息传递的高性能计算应用。接口规范已经被 C、C++ 和 Fortran 语言定义和实现。

MPI 向开发者提供了一系列规范：

- 管理 MPI 程序所运行的分布式环境。
- 向点对点通信提供服务。
- 向群组通信提供服务。
- 支持数据结构定义和内存分配。
- 为块调用的同步提供基本支持。

MPI 的总体参考框架如图 7-4 所示。MPI 的分布式应用由一系列 MPI 进程构成，MPI 进程并行地运行在支持 MPI 的分布式框架上（更像是从云中租用的节点或集群）。

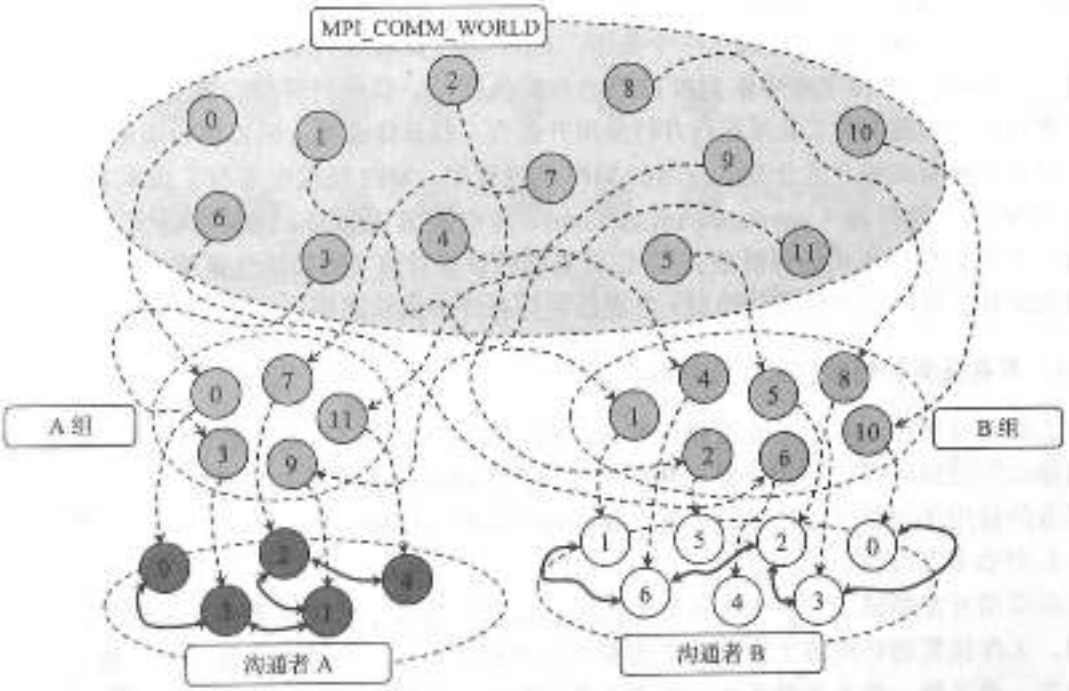


图 7-4 MPI 参考场景

MPI 应用共享相同的 MPI 运行环境，默认为是 MPI_COMM_WORLD 的一部分。该进程组内，所有的分布式进程都有一个独特的标签使 MPI 环境能够定位和处理它们。在全局任务域内可以创建具体的任务子集，例如将属于同一应用的所有 MPI 进程划分为一组。每个 MPI 进程根据所属的分组分配一个标签，该标签是一个独特的标识符，允许组内进程间相互通信。通信过程是通过为每个进程组设定的传播组件实现的。

创建一个 MPI 应用需要定义 MPI 并行执行进程的代码。程序的总体框架如图 7-5 所示。并行执行的代码部分显然位于两个操作符（开启和关闭 MPI 环境）之间，期间可以使用所有 MPI 函数以同步或者异步的方式来发送或者接受消息。



图 7-5　MPI 程序结构

图 7-5 表明 MPI 支持均匀应用，即每个节点上执行的代码部分是相同的。事实上，通过程序标号（运行时可获取）区分节点上执行的操作来实现基于负载通信类型的应用在 MPI 框架下是可行的。MPI 的常用模式之一是主从模式，由一个进程（通常标号为 0）协作其他进程的运行。

MPI 实现框架中的程序将会被相关语言的改进版本的编译器所编译。编译器将引入附加代码来管理运行环境。编译过程的输出文件可以通过 MPI 实现框架提供的工具实现分布式应用。

MPI 应用的执行环境构成了一个集群。此时 MPI 安装在共享文件系统中，每个节点上配置一个 MPI 后台程序来协作 MPI 应用的并行执行。一旦运行环境设置完成，就可以使用 MPI 实现框架所提供的工具来运行并行应用并确立一些具体选项，例如执行应用的节点数量。

目前已经有几种开发分布式应用的 MPI 实现框架，MPI 规范也发布了版本 2。一个较为流行的 MPI 软件环境（www.mcs.anl.gov/mpi）是由美国 Argonne 国家实验室开发的。MPI 作为一个并行和分布式编程模型在 CPU 密集型的数学计算（例如线性系统求解、矩阵运算、有限元计算、线性代数和数值模型）方面已经拥有许多成功案例。

7.2.4　具有任务依赖性的工作流应用

工作流应用由一系列具有依赖性的任务构成。这些依赖（主要是数据依赖，例如一项任务的输出构成另一项任务的前提）决定了应用调度的方式和分布。本例中所关注的是提供一个任务的易用序列并优化任务的放置以获取最小的数据迁移代价。

1. 什么是工作流

在商用开发领域，工作流有很长的历史，它通过描述一系列服务的组合构成了一个商业流程。工作流管理联盟将工作流定义为商业过程的自动化，在整体或局部上，按照一系列流程规范，将文件、信息或任务从一个参与者（资源、人或机器）传递给另一个[64]。工作流

将具有相互依赖关系的任务构成一个执行框架的理念对于表述许多科学实验过程十分有效，并催生了科学工作流。许多科学实验都是由许多特定连接次序的子问题解决组件构成的，连接次序是实验的重要属性。但这些科学实验本质上具有并行关系，当需要执行大量操作或者处理大量数据时，就有必要在分布式框架上执行。对于科学工作流，流程由一个执行应用所决定，参与者间传递的元素主要是任务和数据，参与者主要是计算和存储节点。流程规范由工作流定义框架所定义，以便指导任务调度。科学工作流主要涉及数据管理、分析、仿真和支持工作流执行的中间件。

科学工作流通常由有向无环图（DAG）表示，可描述任务或操作间的依赖关系。有向无环图中的节点表示工作流应用中需要执行的任务，连接节点的箭头定义了任务间的依赖关系以及连接任务的数据路径。有向无环图中最常见的依赖关系是数据依赖，即一项任务的输出文件构成了另一项任务的输入文件。这种依赖关系由从第一项任务指向第二项任务的有向箭头所描述。

图 7-6 所示的有向无环图描述了一个简单的图像合成（蒙太奇）工作流[⊖]。蒙太奇是一个合成图像的工具集，是专门为宇航员设计的，将从不同望远镜或观察视角获取的图像合成为连续的拼接图像。该工具集提供了几种应用以进行图像操作和拼接，其中一些应用可以执行背景重置、视角转换及亮度和色彩的调整。图 7-6 绘制的工作流描述了构成一个蒙太奇的整体流程。右侧的表示描述了构成蒙太奇的不同任务。整个流程充分利用了分布式框架的执行优势，对于每个图像文件依次执行下述操作：图像文件转换，重投影，差异计算和共同平面放置。每张图像可以分别并行上述任务。由此构成一个分布式框架执行工作流的实例。 222

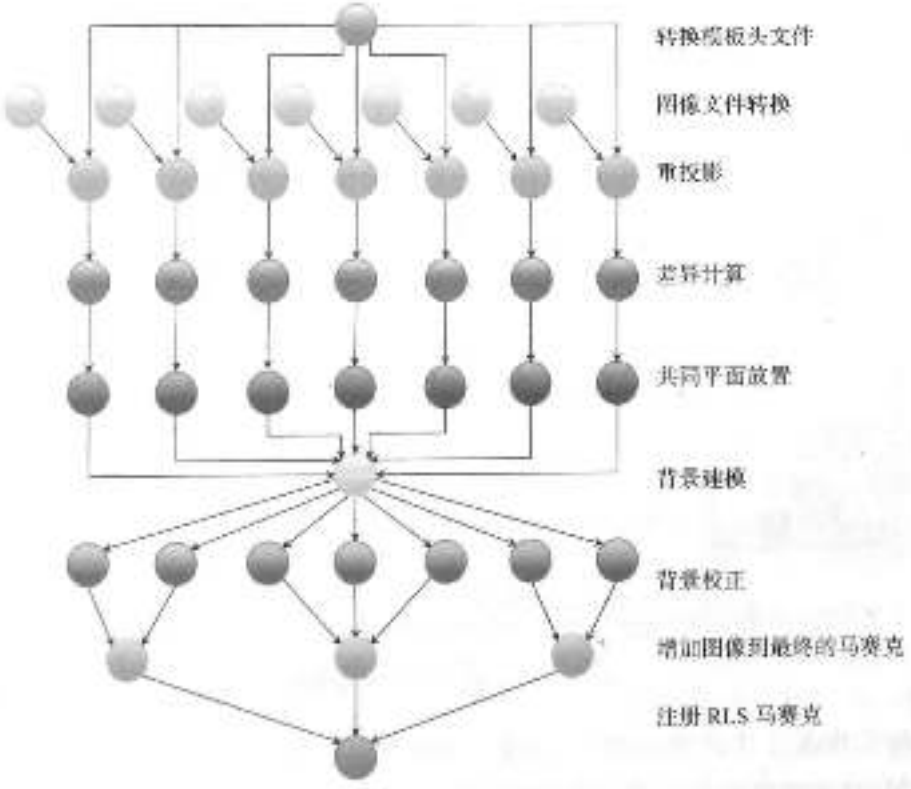


图 7-6 蒙太奇工作流

⊖ 蒙太奇工作流：http://vgrads.rice.edu/research/applications/images/montage-workflow/view。

在分布式框架下执行工作流的另一个原因是易实现在数据本地化原则下的计算任务分发。例如，一项操作需要通过特定的节点才能访问相关资源，该操作不能在其他节点上进行，而其他操作不具有这种要求。一项科学实验可能会涉及若干任务组件，单个任务组件可能需要使用特定设备。此时具有这种约束的任务需要在可行的设备上执行，产生了在原则上
223 不完全并行化的分布式应用。

2. 工作流技术

面向商用的计算工作流被定义为服务组合，目前已经有具体的语言和标准可定义这种工作流，例如 BPEL（业务流程执行语言）[65]。在科学计算领域对工作流的定义并没有形成共识，但也存在几种描述语言和应用实现 [66]。尽管有上述差异，仍然可以为工作流管理系统建立一个抽象参考模型 [67]，如图 7-7 所示。设计工具允许用户可视化地组织工作流应用，这种配置通过基于特定工作流语言的 XML 文件的配置项实现，XML 文件构成了工作流引擎的输入，工作流引擎利用分布式框架来控制工作流执行。大多数情况下，工作流引擎是一个客户端组件，可与资源或者执行工作流的中间件组件进行交互。一些框架提供一个
224 能够处理工作流实例的调度器以实现对工作流应用的支持。

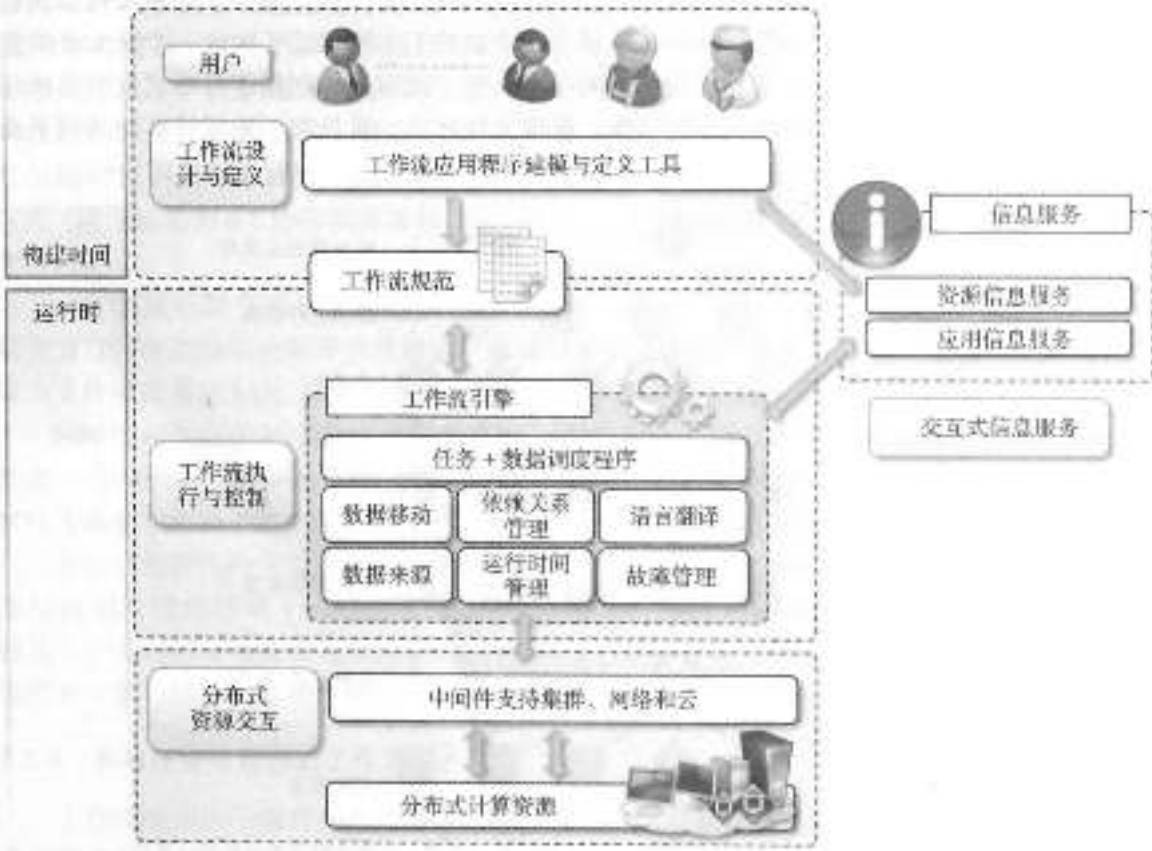


图 7-7　工作流系统的抽象模型

设计和实现基于工作流应用方面的一些相关技术包括 Kepler、DAGMan、Cloudbus Workflow Management System 和 Offspring。

Kepler [68] 是基于几个科研项目的开源科学工作流引擎。该系统在 Ptolemy II [72] 系统上构造，Ptolemy II 系统为开发面向数据流的工作流提供了一个固有平台。Kepler 提供了

一种基于 actor 概念的构造环境，actor 是独立且可重用的计算模块，可加载网络服务、数据库命令等。actor 通过端口通信从输入端口获取数据，并将处理结果和数据写入输出端口。Kepler 的创新在于利用执行工作流的协作逻辑来分离数据流，因此，对同一个工作流，Kepler 提供了不同的模式，例如同步和异步。其工作流规范由 XML 语言所描述。

DAGMan（有向无环图管理）[69] 是 Condor[5][⊖]项目的一部分，扩展了 Condor 调度器来处理任务内部的关联性。Condor 能够为执行程序分配设备但并不支持一定序列的任务调度。因此，DAGMan 作为 Condor 系统的元调度器可以将任务以合适的次序提交给调度器。DAGMan 的输入文件是简单文本文件，包含了任务提交文件指针和任务的依赖性描述。

WfMS（Cloud 工作流管理系统）[70] 是部署在分布式的云或网格系统上的用于处理大型应用工作流的中间件平台。WfMS 通过基于 Web 的门户提供相应软件工具来帮助终端用户组织、调度、执行、监控工作流。门户提供了上传工作流以及使用图形编辑器定义新工作流的方法。WfMS 基于 Gridbus Broker 来执行工作流，Gridbus Broker 是一种网格或云环境下的资源代理器，能够在异构分布式计算环境（如 Linux 集群、Globus 集群或亚马逊 EC2）下按照服务等级属性执行应用。WfMS 使用一种私有 XML 语言作为其工作流规范。

OffSpring[71] 从另一种视角展示了一种基于规划的工作流开发方式。使用者可以设定策略并将它们引入具体执行环境（通过具体的分布式引擎来执行策略）。Offspring 相对于其他工作流引擎的优势在于可以定义动态工作流，即一种半结构化的工作流，可以在任务运行过程中改变其行为。因而允许开发者动态控制任务间的依赖关系。OffSpring 能够与任何支持简单任务划分应用的分布式中间件集成。它能够与 Aeka 集成并在开发过程中支持测试策略的仿真分布式引擎。由于 OffSpring 允许以插件的形式定义工作流，所以它不需要使用任何 XML 语言规范。

7.3　基于任务的 Aneka 编程

Aneka 通过任务编程模型为基于任务的编程提供全方位支持。任务编程通过对 Aneka.Task.ITask 的抽象实现。通过使用这种抽象作为执行遗留的应用程序的基础支撑，参数化应用程序和工作流已经被集成到框架中。本节介绍了该模型的基本概念，并提供了如何为所有先前讨论的模型开发应用程序的例子。 225

7.3.1　任务编程模型

任务编程模型为在 Aneka 平台上快速开发分布式应用程序提供了一个非常直观的抽象。它提供了一个最小 API 集合，大多集中在 Aneka.Tasks.ITask 接口上。此接口连同支持中间件中任务执行的服务，构成了该模型的核心功能。图 7-8 提供了在应用程序执行时任务编程模型组件及其相应角色的整体视图。

开发人员创建了在 ITask 实例方面的分布式应用程序，其中集体执行描述了一个正在运行的应用程序。这些任务连同所有必需的依赖关系（数据文件和库文件）进行了分组，并通过 AnekaApplication 类进行管理，该类专门用于支持任务的执行。另外两个组成部分，AnekaTask 和 Task Manager，构成了一个基于任务的应用程序的客户端视图。前者构成了 Aneka 用来表示中间件中的任务的运行封装，后者是与 Aneka 交互、提交任务、监控执行过

⊖ 2012 年更名为 HTCondor。——译者注

226 程和收集结果的底层组件。在中间件中，成员目录、任务调度、执行服务和存储服务四项服务负责协调活动，以执行基于任务的应用程序。成员目录构成了云的主要接入点，并作为一个服务目录来定位任务调度服务，任务调度服务负责管理基于任务的应用程序的执行。其主要职责是为任务实例分配可用资源，这些资源的特点是执行服务，用于任务执行和监测任务状态。如果应用程序需要数据文件、输入文件或输出文件形式的数据传输支持，存储服务可被用作应用的暂存设施。

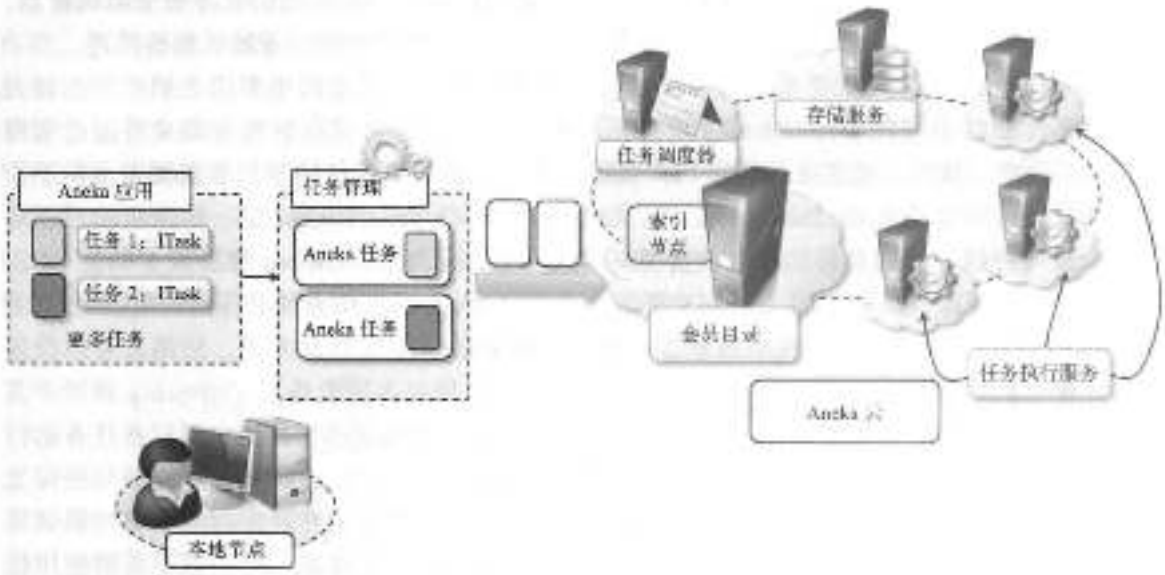


图 7-8　任务编程模型场景

任务模型提供的功能通过 Web 服务完成，它允许任何客户端将任务的执行提交到 Aneka。通过 Web 服务提交任务的步骤与使用框架 API 相同。用户在 Aneka 上创建一个应用程序，并在该应用程序中提交任务。Web 服务限制了可以提交任务的类型，只有有限的任务集合可用于提交。尽管这样，这些任务还是包括了其他分布式计算系统中常见的功能。

7.3.2　用任务模型开发应用

基于任务的应用程序的执行涉及多个组件，这些应用的开发受以下操作的限制：

- 定义类实现 ITask 接口。
- 创建一个正确配置的 AnekaApplication 实例。
- 创建 ITask 实例并将其封装到 AnekaTask 实例中。
- 执行应用程序并等待其完成。

而且，从设计的角度来看，定义一个任务应用程序的过程最终压缩为定义实现 ITask 的类，这将有助于减轻由应用程序生成的工作负荷。

1. ITask 和 AnekaTask

几乎所有用于在 Aneka 上开发基于任务的应用程序的客户端功能都包含在 Aneka.Tasks 命名空间（Aneka.Tasks.dll）中。设计任务最重要的部分是 ITask 接口，这在程序 7-1 中定义。此接口只公开一个方法：执行。在远程节点上执行任务时将会调用该方法。

程序 7-1　ITask 接口

```
namespace Anoka.Tasks
{
    ///<summary>
    ///Interface ITask. Defines the interface for implementing a task.
    ///</summary>
    public interface ITask
    {
        ///<summary>
        ///Executes the sine function.
        ///</summary>
        public void Execute();
    }
}
```

ITask 接口提供了一个编程方案，用于开发本地任务，即 .NET 框架支持的由任何编程语言实现的任务。不同于实施 ITask 接口，实施任务类的限制是最小的，自任务实例被创建且在网络上移动开始，它们就需要被序列化。程序 7-2 描述了一个简单的任务类的实现，能计算出对于给定点 x 的高斯分布。

程序 7-2　ITask 接口实现

```
// File: GaussTask.cs
using System;
using Anoka.Tasks;

namespace GaussSample
{
    /// <summary>
    /// Class GaussTask. Implements the ITask interface for computing the Gauss function.
    /// </summary>
    [Serializable()]
    public class GaussTask : ITask
    {
        /// <summary>
        /// Input value.
        /// </summary>
        private double x;
        /// <summary>
        /// Gets the input value of the Gauss function.
        /// </summary>
        public double X { get { return this.x; } set { this.x = value; } }
        /// <summary>
        /// Result value.
        /// </summary>
        private double y;
        /// <summary>
        /// Gets the result value of the Gauss function.
        /// </summary>
        public double Y { get { return this.y; } set { this.y = value; } }

        /// <summary>
        /// Executes the Gauss function.
        /// </summary>
        public void Execute()
        {
            this.y = Math.Exp(-this.x*this.x);
        }
    }
}
```

ITask 提供了关于如何实现一个任务类和从正在运行的封装类中解编任务的特定操作的最小限制。这由 AnekaTask 类来完成，该类代表了支持 Aneka 应用程序模型 API 的任务实例。这个角色由 AnekaTask 类扮演，它代表与 Aneka 应用程序模型 API 相一致的任务实例。这个类扩展了 Aneka.Entity.WorkUnit 类，并提供了嵌入 ITask 实例的功能。AnekaTask 主要是内部使用，并为最终用户提供了用于指定任务的输入和输出文件的
227 设施。

程序 7-3 说明了如何将一个 ITask 实例封装到 AnekaTask 中，也说明了如何将输入和输出文件添加到给定任务。任务编程模型利用了 WorkUnit 类中的文件管理功能，这是从 AnekaTask 类继承而来的，正如我们提出 Aneka 应用程序模型时所讨论的（见第 5 章）。WorkUnit 有两类文件集合：InputFiles 和 OutputFiles。开发人员可以将文件添加到这些集合中，运行环境会自动将这些文件移到需要它们的地方。输入文件将被上传到 Aneka 云并移动到执行任务的远程节点，输出文件将从执行节点收集数据并将其移动到本地计算机或远程 FTP 服务器。

程序　7-3

```
// create a Gauss task and wraps it into an AnekaTaskInstance
GaussTaskgauss = new GaussTask();
AnekaTask task = newAnekaTask(gauss);
// add one input and one output files
task.AddFile('input.txt', FileDataType.Input, FileAttributes.Local);
task.AddFile('result.txt', FileDataType.Output, FileAttributes.Local);
```

2. 控制任务执行

任务类和 AnekaTask 定义了基于任务的应用程序的计算逻辑，而 AnekaApplication 类提供了用于实现应用程序的协调逻辑的基本特征。

AnekaApplication 是可以用于支持不同的编程模型的通用类。在任务编程中，假定形式为 AnekaApplication<AnekaTask，TaskManager>。所提供的能用于任务模型以及其他编程模型的操作是：

- 静态和动态任务提交。
- 应用程序状态和任务状态监测。
- 任务完成或失败的事件通知。

通过将这些特征组合在一起，可以定义执行特定任务应用程序所需要的逻辑。静态提交是任务型应用程序中非常常见的模式，它涉及所有任务的创建，这些任务都需要在一个循环中执行，并作为一个单一的包提交。更多复杂的任务提交策略都需要实施基于工作流的应用
228 程序，其中任务的执行是通过它们之间的依赖关系确定的。在这种情况下，动态提交任务是一种更有效的技术，涉及任务的提交，即 AnekaApplication 类中实现基于事件的通知机制的结果。

程序 7-4 说明了如何通过静态提交方案将 400 个高斯任务作为一个包来创建和提交。AnekaApplication 类包含所有已提交的用于执行的任务的集合。每个任务可以通过唯一标识符（WorkUnit.Id）检索得到，该标识符由应用程序类中使用的索引操作符 [] 产生。在静态提交的情况下，这些任务将被添加到该应用程序，并且调用 SubmitExecution()
229 方法。

程序 7-4　静态提交

```
// get an instance of the Configuration class from file
Configuration conf = Configuration.GetConfiguration("conf.xml");
// specify that the submission of task is static (all at once)
conf.SingleSubmission = true;
AnekaApplication<AnekaTask, TaskManager> app =
new AnekaApplication<Task,TaskManager>(conf);
for(int i=0; i<400; i++)
{
    GaussTask gauss = new GaussTask();
    gauss.X = i;
    AnekaTask task = new AnekaTask(gauss);
    // add the task to the bag of work units to submit
    app.AddWorkunit(task);
}
// submit the entire bag
app.SubmitExecution();
```

另一种情况是动态提交，其中任务提交是由在执行过程中发生的其他事件完成的，例如，先前提交的任务完成、失败或其他与 Aneka 产生的作用不相关的情况。在这种情况下，开发人员在选择最合适的任务提交策略上有很大的自由度。例如，任务的初始包可能如程序 7-4 中所述的那样被提交。接着，某些任务的完成会导致其他任务的生成和提交。为了实现此方案，有必要依靠由 AnekaApplication 类提供的基于事件的通知系统，并且根据引发的特定事件来触发任务提交。具体而言，我们感兴趣的是 WorkUnitFailed 和 WorkUnitCompleted 事件。

程序 7-5 扩展了前面的例子，并实现了动态任务提交策略，用于提炼高斯分布的计算。其中静态和动态任务提交策略都有应用：包含 400 个高斯任务的初始包被提交到 Aneka，这些任务一完成，就会提交计算分布中间值的新任务。为了捕捉故障并完成任务，有必要听从 230
WorkUnitFailed 和 WorkUnitFinished 事件。该事件要求方法有一个对象参数（对于所有事件句柄），包含应用程序实例和 WorkUnitEventArgs<AnekaTask> 参数，参数中包含用于触发事件的关于 WorkUnit 的信息。这个类公开了 WorkUnit 属性，即如果不为 null，将进入相应的 231
任务实例。对于任务失败的事件，句柄将简单地告知控制台任务已失败，如果可能的话，还会附带上错误的相关信息。对于任务完成后的检查事件，无论在原始包中任务完成是否被提交，句柄都会使用 ExecuteWorkUnit（AnekaTask 任务）方法提交另一项任务。为了将初始包中提交的任务和其他任务进行区分，需要用到 GaussTask.X 的值。如果 X 包含不带小数位的值，它就是一个初始任务，否则就不是初始任务。

程序 7-5　动态任务提交

```
///<summary>
///Main method for submitting tasks.
///</summary>
public void SubmitApplication()
{
    // get an instance of the Configuration class from file
    Configuration conf = Configuration.GetConfiguration("conf.xml");
    // specify that the submission of task is dynamic
    conf.SingleSubmission = false;
    AnekaApplication<AnekaTask, TaskManager> app =
                    new AnekaApplication<Task,TaskManager>(conf);
    // attach methods to the event handler that notify the client code
    // when tasks are completed or failed
```

```
    app.WorkUnitFailed +=
                    new EventHandler<WorkUnitEventArgs<AnekaTask>>(this.OnWorkUnitFailed);
    app.WorkUnitFinished +=
                    new EventHandler<WorkUnitEventArgs<AnekaTask>>(this.OnWorkUnitFinished);
    for(int i=0; i<400; i++)
    {
        GaussTask gauss = new GaussTask();
        gauss.X = i;
        AnekaTask task = newAnekaTask(gauss);
        // add the task to the bag of work units to submit
        app.AddWorkunit(task);
    }
    // submit the entire bag
    app.SubmitExecution();
}
/// <summary>
/// Event handler for task failure.
/// </summary>
/// <param name="sender">Event source: the application instance.</param>
/// <param name="args">Event arguments.</param>
private void OnWorkUnitFailed(object sender, WorkUnitEventArgs<AnekaTask>args)
{
    // do nothing, we are not interested in task failure at the moment
    // just dump to console the failure.
    if (args.WorkUnit != null)
    {
        Exception error = args.WorkUnit.Exception;
        Console.WriteLine("Task {0} failed - Exception: {1}",
        args.WorkUnit.Name, (error == null ? "(Not given)" : error.Message));
    }
}
/// <summary>
/// Event handler for task completion.
/// </summary>
/// <param name="sender">Event source: the application instance.</param>
/// <param name="args">Event arguments.</param>
private void OnWorkUnitFinished(object sender, WorkUnitEventArgs<AnekaTask> args)
{
    // if the task is completed for sure we have a WorkUnit instance
    // and we do not need to check as we did before.
    GaussTask gauss = (GaussTask) args.WorkUnit.Task;
    // we check whether it is an initially submitted task or a task
    // that we submitted as a reaction to the completion of another task
    if (task.X - Math.Abs(task.X) == 0)
    {
        // ok it was an original task, then we increment of 0.5 the
        // value of X and submit another task

        GaussTask frag = GaussTask();
        frag.X = gauss.X + 0.5;
        AnekaTask task =new AnekaTask(frag);

        // we call the ExecuteWorkUnit method that is used
        // for dynamic submission
        app.ExecuteWorkUnit(task);
    }
    Console.WriteLine("Task {0} completed - [X:{1},Y:{2}]",
    args.WorkUnit.Name, gauss.X, gauss.Y);
}
```

静态和动态提交会影响应用程序申请终止条件的方式。静态提交时，申请条件的确定是自动的：当所有最初提交的任务失败或完成时，应用程序将被终止。在这种情况下，可以通过将该应用程序配置中的 SingleSubmission 标志设置为 true 来进行激活。这使得程序在运行时能自动判断应用程序是否完成。动态提交时，运行时不能确定应用程序的终止，因为它始

终有可能提交新的任务。在这种情况下，就需要并发者在适当的时候调用 StopExecution 方法来告诉应用程序什么时候终止应用程序。

232

设计应用程序的协调逻辑时，需要注意的是，任务的提交标识着异步执行模式，这意味着，当提交的任务完成时，方法 SubmitExecution 以及方法 ExecuteWorkUnit 返回结果，但不是任务实际执行完成。这就要求开发人员将正确的同步逻辑放在合适的位置，以便应用程序的主线程处于等待状态，直到所有任务终止且应用程序完成。这种行为可以通过使用 System.Threading 命名空间提供的同步 API 来实现：System.Threading.AutoResetEvent 或 System.Threading.ManualResetEvent。这两个 API 五同最小的逻辑，计算所有被收集的任务，并且一旦所有任务终止，将会通知主线程（通过调用方法 WaitHandle.Wait() 使其处于等待状态）。它也可以提供适当的用于管理应用程序的执行流程所需的基础设施。程序 7-6 提供了一个完整的任务提交程序，实现了动态提交和相应的同步逻辑。

程序 7-6　高斯应用

```
// File: GaussApp.cs
using System;
using System.Threading;

using Aneka.Entity;
using Aneka.Tasks;

namespace GaussSample
{
    /// <summary>
    /// Class GaussApp. Defines the coordination logic of the
    /// distributed application for computing the gaussian distribution.
    /// </summary>
    public class GaussApp
    {
        /// <summary>
        /// Semaphore used to make the main thread wait while
        /// all the tasks are terminated.
        /// </summary>
        private ManualResetEvent semaphore;
        /// <summary>
        /// Counter of the running tasks.
        /// </summary>
        private int taskCount = 0;
        /// <summary>
        /// Aneka application instance.
        /// </summary>
        private AnekaApplication<AnekaTask, TaskManager> app;

        /// <summary>
        /// Main entry point for the application.
        /// </summary>
        /// <param name='args'>An array of strings containing the command line.</param>
        public static void Main(string[] args)
        {
            try
            {
                // initialize the logging system
                Logger.Start();

                string confFile = "conf.xml";
                if (args.Length > 0)
                {
```

```
                confFile = args[0];
            }
            // get an instance of the Configuration class from file
            Configuration conf = Configuration.GetConfiguration(confFile);
            // create an instance of the GaussApp and starts its execution
            // with the given configuration instance
            GaussApp application = new GaussApp();
            application.SubmitApplication(conf);
        }
    catch(Exception ex)
    {
        IOUtil.DumpErrorReport(ex, "Fatal error while executing application.");
    }
    finally
    {
        // terminate the logging thread
        Logger.Stop();
    }
}

/// <summary>
/// Application submission method.
/// </summary>
/// <param name="conf">Application configuration.</param>
public void SubmitApplication(Configuration conf)
{
    // initialize the semaphore and the number of
    // task initially submitted
    this.semaphore = new ManualResetEvent(false);
    this.taskCount = 400;

    // specify that the submission of task is dynamic
    conf.SingleSubmission = false;
    this.app = new AnekaApplication<Task,TaskManager>(conf);
    // attach methods to the event handler that notify the client code
    // when tasks are completed or failed
    this.app.WorkUnitFailed +=
        new EventHandler<WorkUnitEventArgs<AnekaTask>>(this.OnWorkUnitFailed);
    this.app.WorkUnitFinished +=
        new EventHandler<WorkUnitEventArgs<AnekaTask>>(this.OnWorkUnitFinished);

    // attach the method OnAppFinished to the Finished event so we can capture
    // the application termination condition. this event will be fired in case of
    // both static application submission or dynamic application submission
    app.Finished += new EventHandler<ApplicationEventArgs>(this.OnAppFinished);

    for(int i=0; i<400; i++)
    {
        GaussTask gauss = new GaussTask();
        gauss.X = i;
        AnekaTask task = newAnekaTask(gauss);
        // add the task to the bag of work units to submit
        app.AddWorkunit(task);
    }
    // submit the entire bag
    app.SubmitExecution();

    // wait until signaled, once the thread is signaled the application is completed
    this.semaphore.Wait();
}
```

```
/// <summary>
/// Event handler for task failure.
/// </summary>
/// <param name="sender">Event source: the application instance.</param>
/// <param name="args">Event arguments.</param>
private void OnWorkUnitFailed(object sender, WorkUnitEventArgs<AnekaTask> args)
{
    // do nothing, we are not interested in task failure at the moment
    // just dump to console the failure.
    if (args.WorkUnit != null)
    {
        Exception error = args.WorkUnit.Exception;
        Console.WriteLine("Task {0} failed - Exception: {1}",
        args.WorkUnit.Name,
        (error == null ? "[Not given]" : error.Message));
    }
    // we do not have to synchronize this operation because
    // events handlers are run all in the same thread, and there
    // will not be other threads updating this variable

    this.taskCount--;
    if (this.taskCount == 0)
    {
        this.app.StopExecution();
    }
}
/// <summary>
/// Event handler for task completion.
/// </summary>
/// <param name="sender">Event source: the application instance.</param>
/// <param name="args">Event arguments.</param>
private void OnWorkUnitFinished(object sender, WorkUnitEventArgs<AnekaTask> args)
{
    // we do not have to synchronize this operation because
    // events handlers are run all in the same thread, and there
    // will not be other threads updating this variable
    this.taskCount--;

    // if the task is completed for sure we have a WorkUnit instance
    // and we do not need to check as we did before.
    GaussTask gauss = (GaussTask) args.WorkUnit.Task;
    // we check whether it is an initially submitted task or a task
    // that we submitted as a reaction to the completion of another task
    if (task.X - Math.Abs(task.X) == 0)
    {
        // ok it was an original task, then we increament of 0.5 the
        // value of X and submit another task
        GaussTask frag = GaussTask();
        frag.X = gauss.X + 0.5;
        AnekaTask task =new AnekaTask(frag);
            this.taskCount++;
            // we call the ExecuteWorkUnit method that is used
            // for dynamic submission
            app.ExecuteWorkUnit(task);
        }
        Console.WriteLine("Task {0} completed - [X:{1},Y:{2}]",
                          args.WorkUnit.Name, gauss.X, gauss.Y);
        if (this.taskCount == 0)
        {
```

```
                this.app.StopExecution();
            }
        }
        /// <summary>
        /// Event handler for the application termination.
        /// </summary>
        /// <param name="sender">Event source: the application instance.</param>
        /// <param name="args">Event arguments.</param>
        private void OnWorkUnitFinished(object sender, ApplicationEventArgs args)
        {
            // unblock the main thread, because we have identified the termination
            // of the application
            this.semaphore.Set();
        }
    }
}
```

上述程序提供了 GaussApp 类的完整定义，其中包含应用程序的主入口点，这是一个用于控制已实现的应用程序执行的非常简单的逻辑。GaussApp 应用程序通过使用 taskCount 字段来追踪当前运行的任务数量。当这个值达到零时，没有更多的任务等待执行，通过调用 StopExecution，应用程序停止执行。此方法触发 ApplicationFinished 事件，该事件句柄（OnApplicationFinished）通过信号量来疏导主线程。taskCount 的初始值为 400，这是任务初始包的大小。每当任务失败或完成一次，这个值减小一个单位；如果有一个新的任务提交，该字段增加一个单位。在这两个事件句柄（OnWorkUnitFailed 和 OnWorkUnitFinished）的最后，对 taskCount 的值进行检查，看它是否等于零，以及是否有必要停止应用程序。可以看到，除了使用 ManualResetEvent，不需要其他同步结构。因为操纵 taskCount 值的代码在一个单一的线程中执行，在递增或递减值时不会有其他事件对值产生影响。

控制任务应用程序的执行需要考虑的最后一个方面是重新提交策略。默认情况下，应用程序的配置中将重新提交策略设置为手动模式。这意味着如果在任务执行过程中由于异常而导致任务失败，任务实例将被发送回客户端应用程序，并且如果必要的话需要开发人员重新提交任务。在自动重新提交的模式下，Aneka 将一直重新提交任务，直到达到尝试的最大次数为止。如果任务一直失败，任务失败事件将会被触发。这个属性可以通过将 Configuration.ResubmitMode 设置为 ResubmitMode.Manual 或 ResubmitMode.Auto 进行控制。如前所述，此属性默认设置为 ResubmitMode.Manual。

3. 文件管理

基于任务的应用程序通常通过处理文件来执行操作。正如我们已经讨论过的，文件可能包括任务的输入数据、计算结果，还可能代表可执行代码或函数库的依赖。因此，为基于任务的应用程序提供文件传输功能是必不可少的。Aneka 提供了在分布式基础设施
233 中进行文件管理的内置功能，显然任务编程模型利用了这些功能。任何基于 WorkUnit 和 ApplicationBase 类的模型都支持文件管理。它可以利用 WorkUnit.InputFiles 和 WorkUnit.OutputFiles 集合通过 ApplicationBase.SharedFiles 集合提供适用于所有 WorkUnit 实例的输入文件，也可以提供特定实例的输入和输出文件。

文件管理的一个基本组成部分是 FileData 类，它构成了物理文件的逻辑表示，正如 Aneka.Data.Entity 命名空间（Aneka. Data.dll）中所定义的。FileData 实例提供如下有关文件

的信息：

- 性质：是否是共享文件、输入文件或输出文件。
- 在本地和远程文件系统中的路径，包括不同的名字。
- 一个提供其他信息的属性集合（例如文件的最终目的地以及文件是否是暂存的，等等）。

使用 FileData 类，用户可以指定任务的文件相关性和应用程序，并且在需要时 Aneka API 会自动将它们转移到 Aneka 云或从 Aneka 云中转移出来。Aneka 允许指定本地文件或保存在 FTP 服务器或亚马逊 S3 上的远程文件。一个 FileData 实例必须有三个要素：所有者，名称和类型。通过相应的 ID，所有者确定哪些是需要文件的计算元件：应用实例或工作单位。类型指定了文件是共享文件、输入文件或输出文件，名称表示对应物理文件的名称。

234 ↓ 237

程序 7-7 演示了如何将文件的依赖关系添加到应用程序和任务中。它可以同时添加 FileData 实例，从而对文件信息具有更大的控制权，或使用更直观的方式，这种方式仅仅需要文件的名称和类型。

程序 7-7　文件依赖管理

```
// get an instance of the Configuration class from file
Configuration conf = Configuration.GetConfiguration("conf.xml");
AnekaApplication<Task,TaskManager>app =new AnekaApplication<Task,TaskManager>(conf);

// Attach shared files with different methods by using the FileData class and directly
// using the API provided by the AnekaApplication class

// create a local shared file whose local and remote name is "pi.tab"
FileDataapiTab = newFileData("pi.tab",FileDataType.Shared);
app.AddSharedFile(piTab);
// once the file is added to the collection of shared files, its OwnerId property
// references app.Id

// create a remote shared file by specifying the attributes whose name is "pi.dat"
FileDataapiDat = newFileData("pi.dat",FileDataType.Shared, FileAttributes.None);
// the StorageBucketId property points a specific configuration section that is
// used to store the information for retrieving the file from the remote server
piDat.StorageBucketId = "HTTPStore";
app.AddSharedFile(piDat);
// once the file is added to the collection of shared files, its OwnerId property
// references app.Id

// adds a local shared file
app.AddSharedFile("pi.xml");

for(int i=0; i<400; i++)
{
   GaussTask gauss = new GaussTask();
   gauss.X = i;
   AnekaTask task = newAnekaTask(gauss);

   // adds a local input file for the current task whose name is "<i>.txt"
   // where <i> is the value of the loop variable
   FileData input = new FileData(string.Format("{0}.txt", i);
   FileDataType.Input, FileAttributes.Local);
   // once transferred to the remote node, the file will have the name
   // "input.txt". Since tasks are executed in separate directories there
   // will be no name clashing
```

```
        input.VirtualPath ="input.txt";
        task.AddFile(input);
        // once the file is added to the task, it will be stored in the InputFiles
        // collection and its OwnerId property will referenced task.Id

        // adds anoutput file for the current task whose name is "out.txt" that will
        // be stored on S3
        FileData output =new FileData("out.txt", FileDataType.Input, FileAttributes.None);
        // once transferred to the remote server, the file will have the name
        // "<i>.out"where <i> is the value of the loop variable. In this way we
        // easily avoid name clashing while storing output files into a single
        // directory
        output.VirtualPath =string.Format("{0}.out", i);
        output.StorageBucketId = "S3Store";
        task.AddFile(output);
        // once the file is added to the task, it will be stored in the InputFiles
        // collection and its OwnerId property will referenced task.Id

        // adds a localoutput file for the current task whose name is "trace.log".
        // The file is optional, this means that if after the execution of the task the file

        // is not present no exception or task failure will be risen.
        FileData trace =new FileData("trace.log", FileDataType.Input,
        FileAttributes.Local | FileAttributes.Optional);
        // once transferred to the local machine, the file will have the name
        // "<i>.log"where <i> is the value of the loop variable. In this way we
        // easily avoid name clashing while storing output files into a single
        // directory
        trace.VirtualPath =string.Format("{0}.log", i);
        task.AddFile(trace);
        // once the file is added to the task, it will be stored in the InputFiles
        // collection and its OwnerId property will referenced task.Id

        // add the task to the bag of work units to submit
        app.AddWorkunit(task);
    }

    // submit the entire bag, files will be moved automatically by the Aneka APIs
    app.SubmitExecution();
```

文件管理的一般交互流程如下：

- 一旦提交应用程序，共享文件就被上传到 Aneka 云中。
- 如果文件是本地的，它将在由 Configuration.Workspace 确定的目录位置中搜索得到；如果文件是远程的，使用 FileData.StorageBucketId 映射的特定配置设置可以访问远程服务器和文件内容。
- 如果上传输入文件失败，应用程序会被终止并显示错误。
- 对于应用程序的每个任务，相应的输入文件会被上传到 Aneka 云上。
- 一旦任务被调度到远程节点上执行，运行时应用程序的所有共享文件和任务的输入文件将被传送到任务的工作目录，并且，如果 FileData.VirtualPath 属性不为空，它们将会被重新命名。
- 任务执行后，运行时刻会寻找所有已添加到 WorkUnit.OutputFiles 集合的文件。如果不为空，则 FileData.VirtualPath 属性的值将被用于查找文件；否则，FileData.FileName 属性将作为参考。需要存在不包含 FileAttributes.Optional 属性的文件，否则，认为该任务执行失败。

- 不管任务执行成功或失败，运行时刻都尝试将所有找到的文件收集并移动到各自的目的地。包含 FileAttributes.Local 属性的文件都从文件保存的目录位置（由 Configuration.Workspace 标识）移动到本地计算机上。设置了 StorageBucketId 属性的文件将被上传到相应的远程服务器上。

用于文件管理的基础设施为数据的移动提供了透明的和可扩展的服务。文件管理的结构基于工厂和存储桶的概念。工厂，即 IFileTransferFactory，是用来抽象客户机的创建以及用于文件传输的服务组件的组成部分，它使整个架构通过接口就能工作而不需要特定的实施方式。存储桶是通过配置文件来区分这些组件的字符串属性的集合，在提交应用程序之前，由用户或配置文件指定或者通过编程方式将此信息添加到配置对象。通过 StorageService 使用工厂可以提取出远程输入和共享文件，并得到远程输出文件。

程序 7-8 所示为一个简单配置文件，其中包含通过 FTP 和 S3 协议来访问远程文件的所需设置。在 <Groups> 标签内，有一个名为 StorageBuckets 的特定组，该组维护每个在文件传输中需要使用到的存储桶的配置设置。每个 <Group> 标签代表一个存储桶，并且 name 属性包含在 FileData.StorageBucketId 属性中。每个组的内容都取决于所使用的存储桶的类型，这是由 Scheme 属性标识的。这些值被用于 FTP 和 S3 协议的取决实现，以此来访问远程服务器和传输文件。

238

程序 7-8　Aneka 应用配置文件

```
<?xml version="1.0" encoding="utf-8" ?>
<Aneka>
  <UseFileTransfer value="true" />
  <Workspace value="." />
  <SingleSubmission value="false" />
  <ResubmitMode value="Manual" />
  <PollingTime value="1000" />
  <LogMessages value="true" />
  <SchedulerUri value="tcp://localhost:9090/Aneka"/>
  <UserCredential type="Aneka.Security.UserCredentials" assembly="Aneka.dll">
    <UserCredentials username="Administrator" password=""/>
  </UserCredential>
  <Groups>
    <Group name="StorageBuckets">
      <Groups>
        <Group name="FTPStore">
          <Property name="Scheme" value="ftp"/>
          <Property name="Host" value="www.remoteftp.org"/>
          <Property name="Port" value="21"/>
          <Property name="Username" value="anonymous"/>
          <Property name="Password" value="nil"/>
        </Group>
        <Group name="S3Store">
          <Propertyname="Scheme" value="S3"/>
          <Propertyname="Host"value="www.remoteftp.org"/>
          <Propertyname="Port"value="21"/>
          <Propertyname="Username"value="anonymous"/>
          <Propertyname="Password"value="nil"/>
        </Group>
      </Groups>
    </Group>
  </Groups>
</Aneka>
```

4. 任务库

Aneka 提供了一组现成的任务，用于执行远程文件管理的最基本操作。这些任务是 Aneka.Tasks.BaseTasks 命名空间（Aneka.Tasks.dll 库的一部分）的组成部分。下面的操作可实现：

- 文件复制。LocalCopyTask 执行远程节点上的文件副本，它将一个文件作为输入，并在不同的名称或路径下产生它的一个副本。
- 传统应用程序的执行。ExecuteTask 允许使用 System.Diagnostics.Process 类执行外部和传统应用程序。它需要运行可执行文件的位置，还可以指定命令行参数。ExecuteTask 还收集执行应用程序产生的标准错误和标准输出。
- 替代操作。SubstituteTask 通过将生成的文件以一个不同的名字保存，来执行一个给定文件的搜索和替换操作。它可以指定基于字符串的名称 – 值对的集合来表示搜索的字符串及其相应的替换。
- 文件删除。DeleteTask 通过远程节点上的文件系统删除可访问文件。
- 定时延迟。WaitTask 引入定时延迟。这个任务可以在多种情况下使用，例如，它可用于分析或简单地用于仿真执行过程。此外，如果需要的话，它也可以用于在两个应用程序的执行之间引入停顿。
- 任务合成。CompositeTask 实现了集成模式[⊖]，并允许将顺序执行的多个任务组合成一个任务。这个任务对于执行复杂的任务（包含其他任务中的组合操作）非常有用。

239 / 240

241

基本任务库不提供对数据传输的任何支持，因为这个操作是基础设施根据需要自动执行的。除了这些简单的任务，AnekaAPI 允许通过实现 ITask 接口和支持对象序列化来创建用户自定义任务。

5. Web 服务集成

Aneka 可以通过 Web 服务与其他技术和应用程序进行集成，也可以对寄存在 Aneka 云上的一些服务进行平台无关的访问。其中，对于任务提交，Web 服务允许第三方应用程序提交任务，这一点与传统的计算网格相同。

任务提交 Web 服务是一个可以部署在任何 ASP.NET Web 服务器上并公开了一个用于作业提交的简单接口的附加组件，这符合 Aneka 应用程序模型。任务 Web 服务提供了一个更符合网格计算的传统方式的接口。因此，术语作业的新概念被引入，即预定义任务的集合。基于 Web 提交的参考场景如图 7-9 所示。用户在云上创建一个分布式应用程序实例，在这个应用程序的上下文中，可以提交多个或单个作业查询应用程序的状态。由用户决定是在所有作业完成后终止该应用程序还是直接退出应用程序。

通过将基本任务库中的任务组合在一起可以创建新的作业。Web 服务接口支持的操作如下：

- 本地文件复制到远程节点。
- 文件删除。
- 通过通用的 shell 服务执行传统应用程序。
- 参数替换。

⊖ 软件工程中，集成模式是一种将多组件组合成单一组件的软件设计模式。采用组合模式的优点是构造软件基础设施支持将操作执行转向一组对象，该对象以完全透明的方式被处理为一个单元。参考：E.Gamma,R.Helm,R.Johnson,andJ.M.Vlissides, Design Patterns: Elements of Reusable Software Design, Addison-Wesley,1995,ISBN:0201633612。



图 7-9　Web 服务提交场景

也可以为每个作业指定输入和输出文件，在这种情况下，唯一的限制是，输入和输出文件都必须驻留在远程 FTP 服务器上。这使得 Aneka 可以自动从这些服务器下载文件，而无需用户干预。图 7-10 给出了关于通过 Web 服务进行作业提交的对象模型的详细信息。

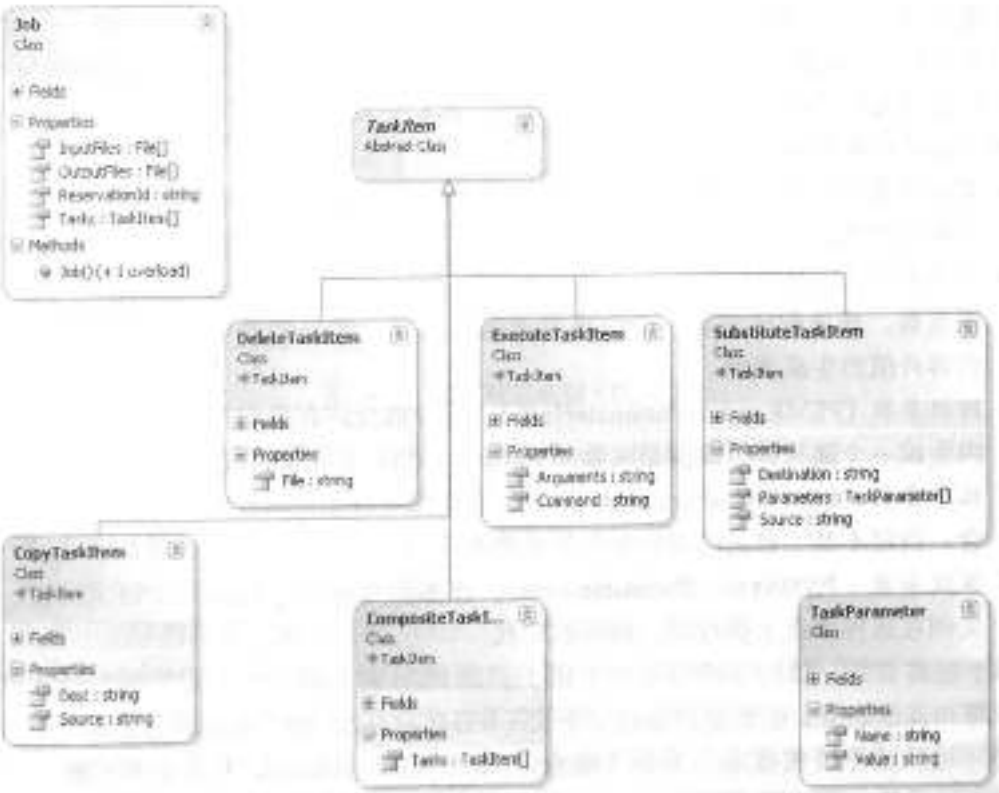


图 7-10　工作对象模型

传统网格技术，如 Gridbus Broker[15] 和工作流引擎 [70]，可以利用任务 Web 服务来
242 提交任务，用于在由 Aneka 管理的云节点执行。

7.3.3 开发参数化应用

Aneka 通过客户端组件的集合（该集合允许开发人员通过 API 编程或图形用户界面（GUI）快速开发出应用程序的原型）将对参数化应用程序的支持集成到任务模型的顶端。支持参数化应用程序发展的该组抽象和工具构成了参数化设计模型（PSM）。

根据 Aneka.PSM，PSM 被组织成多个命名空间，更确切地说：

- Aneka.PSM.Core（Aneka.PSM.Core.dll）包含通过给定的一组参数来定义模板任务和客户端组件（用于管理任务）的基本类。
- Aneka.PSM.Workbench（Aneka.PSM.Workbench.exe）和 Aneka.PSM.Wizard（Aneka.PSM.Wizard.dll）包含设计和监测参数化应用程序的用户接口支持。它们大多包含设计资源管理器所需要的类和组件。这是开发参数化应用程序的主要 GUI。
- Aneka.PSM.Console（Aneka.PSM.Console.exe）包含支持在控制台模式下执行参数化应用程序的组件和类。

243 这些命名空间定义了在 Aneka 顶端对于开发和控制参数化应用程序的支持。

1. 对象模型

参数化模型的基本要素在 Aneka.PSM.Core 命名空间中定义。该模型引入了作业（Aneka.PSM.Core.PSMJobInfo）的概念，它标识着一个参数化应用程序。作业包括文件依赖关系和参数的定义，及其受理域名和模板任务的定义。图 7-11 所示为对象模型的相关的组件。

应用程序设计的根组件是 PSMJobInfo 类，其中包含有关共享文件和输入、输出文件
244 （PSMFileInfo）的信息。按照 Aneka 应用程序模型，共享文件适用于任务模板的所有实例，而输入和输出文件只针对特定的任务实例有效。因此，这些文件可以表示为参数的函数。目前，可以指定五种不同类型的参数：

- 常量参数（PSMSingleParameterInfo）。该参数标识了在设计阶段设定的特定值，在应用程序的执行过程中不会改变。
- 范围参数（PSMRangeParameterInfo）。该参数允许定义一个允许值范围，可能是整数或实数。该参数标识着一个由离散值组成的域，并且需要设定下界和上界，以及所有容许值的生成步骤。
- 随机参数（PSMRandomParameterInfo）。该参数允许在给定范围（由上界和下界定义）内生成一个随机值，该值是实数。
- 枚举参数（PSMEnumParameterInfo）。该参数允许指定由任何类型的值组成的离散集合。指定不基于数值的离散集合是非常有用的。
- 系统参数（PSMSystemParameterInfo）。该参数允许用于映射一个特定的值，当任务实例在远程节点上执行时，该值被取代。

除了这些参数，该对象模型保留了用于识别 PSM 对象模型的特定值的特殊参数，如任务标识符和其他数据。参数通过执行上下文（PSMContext，负责提供默认和运行时刻值）访问执行环境。任务模板被定义为命令集合（PSMCommandInfo），它复制和扩展了基础任务库中提供的功能。用于撰写任务模板的可用指令形成了以下操作：

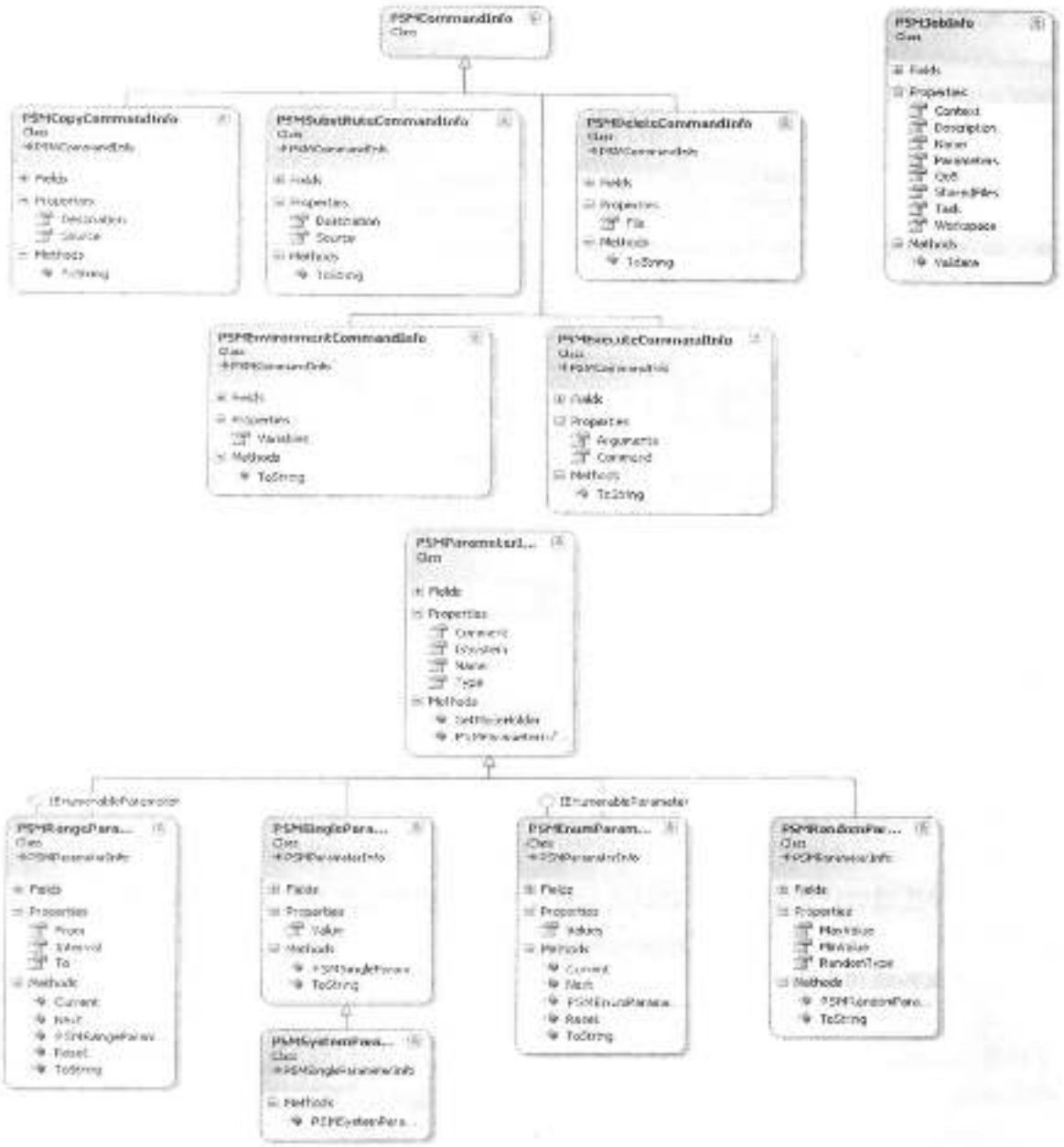


图 7-11 PSM 对象模型（相关类）

- 本地文件复制到远程节点上（PSMCopyCommandInfo）。
- 远程文件删除（PSMDeleteCommandInfo）。
- 通过 shell 执行程序（PSMExecuteCommandInfo）。
- 远程节点上的环境变量设置（PSMEnvironmentCommandInfo）。
- 文件中的字符串模式替换（PSMSubstituteCommandInfo）。

根据任务创建方案中的描述，可以通过组合这些基本块来定义任务模板。这些指令公开的属性可以包括前面定义的参数，其值将在任务实例的生成期间提供。

一个参数化应用程序由作业管理器（IJobManager）执行，开发者可以通过任务模型的底层接口来实现。图 7-12 通过作业管理器的特定参考和任务模型 API 展示了 PSM 的 API 之

间的关系。

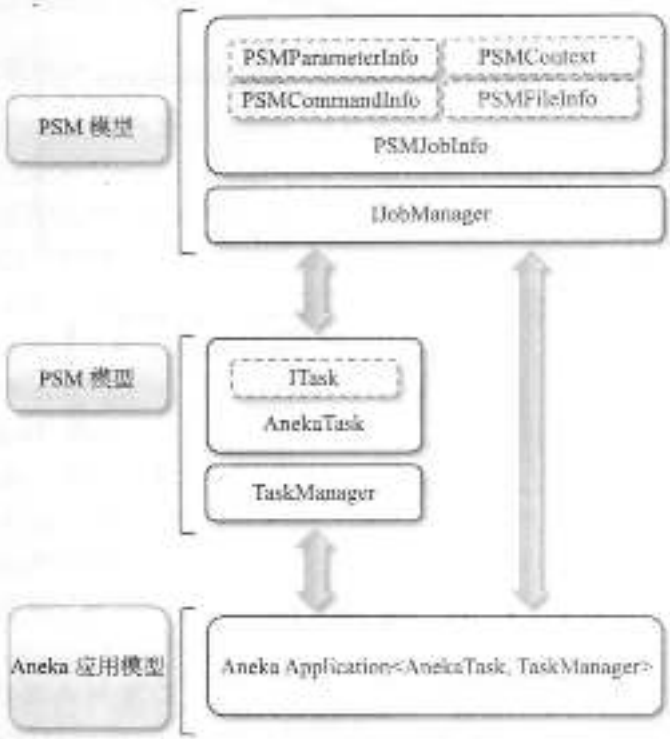


图 7-12　参数化模型 API

通过 IJobManager 接口，可以指定与 Aneka 中间件进行交互的用户凭据和配置。IJobManager 的实现将创建一个相应的 Aneka 应用实例，并利用任务模型 API 来提交所有从模板任务中生成的任务实例。该接口还公开了用于控制和监测参数化应用程序执行的设施，以及对登记有关应用程序的统计信息的支持。

245
|
246

2. 开发和监测工具

核心库允许开发者直接编写参数化应用程序并将其嵌入其他应用程序。Aneka 设计资源管理器和 Aneka PSM 控制台则通过提供对应用程序可视化设计以及交互式和非交互式应用程序执行的支持，简化了参数化应用程序的设计和开发。

Aneka 设计资源管理器是一个集成可视化环境，用于参数化应用程序的设计、执行及状态监测。它提供了一个简单的向导，可以帮助用户直观地定义参数化应用程序的任何方面，如文件的依赖关系和结果文件、参数和模板任务。该环境还提供了一组组件集合，用于帮助用户监控应用程序的执行，对应用程序执行进行汇总统计，获得详细的任务转变监测，以及获得对应用程序日志的大规模访问。

Aneka PSM 控制台是一个命令行实用工具，旨在在非交互模式下运行参数化应用程序。该控制台提供了运行应用程序（有监控其执行的功能）的简化接口。对于设计资源管理器，控制台提供的对于保持和可视化综合统计数据的支持较少，但它提供了一个更加简化的文本形式来显示相同的数据。

247

7.3.4　管理工作流

Aneka 本身不支持工作流，但具备相关插件，允许基于客户端的工作流管理器将任务提

交到 Aneka。目前，有两种不同的工作流管理器可以利用 Aneka 执行任务：工作流引擎 [70] 和 Offspring[71]。前者会利用 Aneka 提供的任务提交 Web 服务，后者直接与 Aneka 编程 API 进行交互。工作流引擎插件是框架提供集成功能的一个例子，它允许使用任何技术和语言开发的客户端应用程序利用 Aneka 来执行任务。为 Offspring 开发的集成构成了另一个例子，它说明了如何在框架现有的 API 之上构建另一种编程模型。因此，我们将讨论此解决方案的更多细节。

图 7-13 描述了 Offspring 的架构。该系统由两种类型的组件组成：插件和分配引擎。插件用于丰富功能环境，分布引擎代表用于任务执行的分布式计算设施的访问接口。现有的插件 AutoPlugin 提供了在策略方面用于工作流定义的设施。策略生成了执行的任务，并且定义了通过引擎提交任务的逻辑（在顺序、协调性和依赖性方面）。一个特定组件，即 StrategyController，能从分布引擎中解耦策略。因此，策略可独立于所使用的特定中间件来 248
定义。与 Aneka 的连接是通过 AnekaEngine 实现的，它实现了 IDistributionEngine 中用于 Aneka 中间件的操作，并且依赖于由任务模型编程 API 公开的服务。

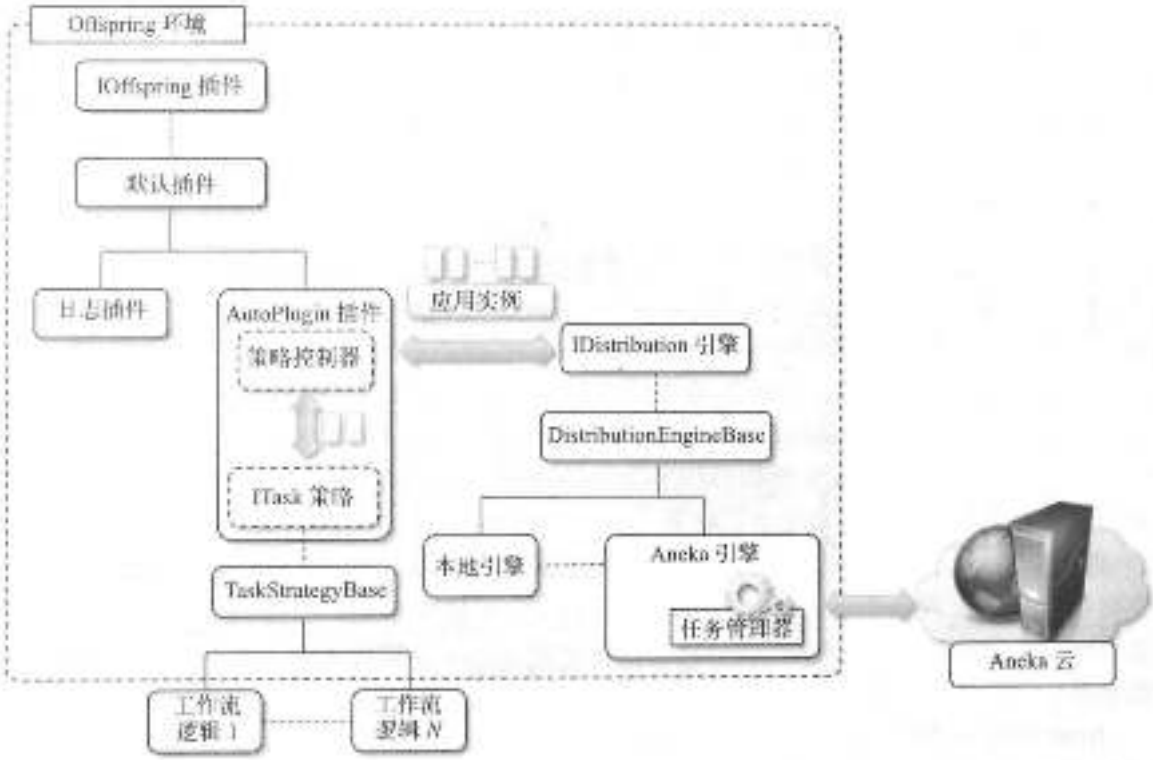


图 7-13　Offspring 架构

该系统允许动态工作流的执行，其结构被定义为执行工作流。可以定义两种不同类型的任务：native 任务和 legacy 任务。native 任务在托管代码中实现。legacy 任务管理文件的依赖性以及封装所有在远程节点上执行 legacy 程序所必要的数据。此外，策略可以定义共享文件依赖关系，这对所有工作流生成的任务都是必需的。任务之间的依赖关系由 StrategyController 和分布式引擎所触发的事件隐式定义。

图 7-14 描述了这些组件之间的交互。两个主要的执行线程控制策略的执行，控制线程管理策略的执行，而监视线程收集分布引擎的反馈信息，并允许策略对以前提交任务的执行 249

进行动态反应。从整个工作流开发者的角度看，逻辑很简单。策略的执行分为三大步：建立、执行和结束。第一步涉及策略的设置和应用程序到策略的映射。相应地，结束步骤负责释放所有分配给策略的内部资源和关闭应用程序。工作流执行的核心是执行步骤，它被分解成一组迭代。在每次迭代期间，一组任务集合被提交，这些任务彼此之间不具有依赖性，并且可以并行执行。一旦任务完成或失败，该策略将查询是否需要执行一组新的任务。以这种方式实现了任务间的依赖关系。如果有更多要执行的任务，提交这些任务并且控制器等待来自引擎的反馈；否则，该策略的迭代完成。在每次迭代结束时，控制器检查该策略是否已经完成了执行，如果完成，执行结束步骤。

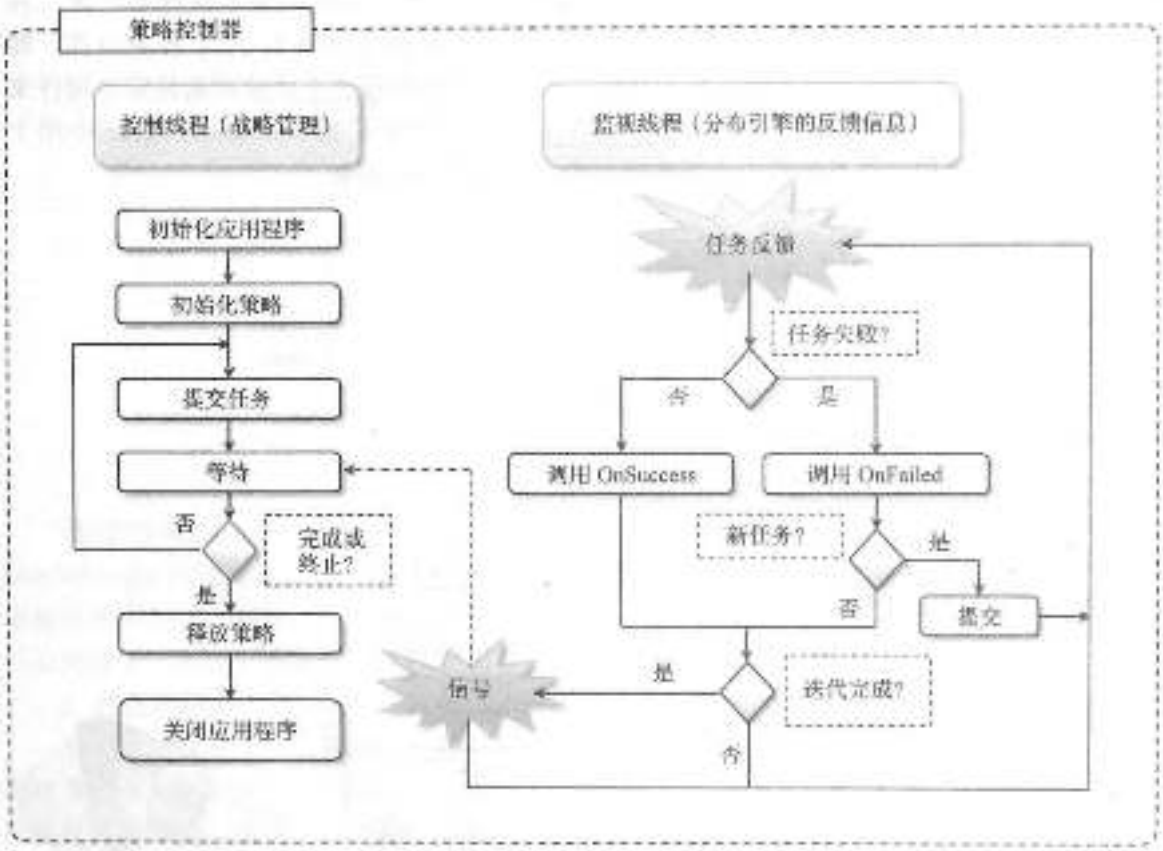


图 7-14　工作流协调

AnekaEngine 创建了一个 AnekaApplication 类的实例，用于策略的每一次执行，用一个 TaskManager 的具体实现来配置模板类，覆盖了实现文件管理的行为，并且优化了输出文件的上传。为了支持不依赖分布引擎的工作流执行，应用程序配置设置由分配引擎控制，并且在所有通过引擎执行的策略之间共享。

本章小结

本章介绍了基于任务编程的概念，并在任务概念的基础上，概述了分布式应用程序的开发技术。基于任务的程序设计是将应用程序的计算分布到一组节点上最为直观的方法。基于任务的程序设计的主要抽象是任务的概念，它代表了可以分离的并作为一个单元执行的一组操作。任务可以是通过 shell 或更复杂的代码（需要特定的运行环境来执行）来执行的一个简

单程序。很多时候，任务的执行需要输入文件并且产生输出文件作为结果。根据这个模型，应用程序表示为任务的集合，这些任务相互关联的方式，及其具体的性质和特点区分了不同的模式（基于任务编程的一种表现形式）。

传统上，基于任务的编程模型已经被成功地应用于分布式应用程序开发的许多领域。本章确定了可以利用任务模型的三个主要计算类别。高性能计算（HPC）是指利用分布式计算设施解决那些需要大量计算能力的问题。常见的HPC应用程序具有一个很大的计算密集型任务的集合，其持续时间相对较短。高吞吐量计算（HTC）适用于将分布式计算设施用于执行在长时间内需要大量计算能力的应用程序的情况。任务可能不是很多，但是它们持续的时间长，因此基础设施的可靠性变得至关重要。多任务计算（MTC）是最新出现的趋势，它能够识别一个异构应用程序集合和应用程序的要求，MTC填补了HPC和HTC之间的差距。 250

本章简要回顾了与任务编程相关的常见模型。高度并行应用程序由不相互关联的任务集合组成，它们可以以任何顺序执行，并且不需要共同分配。参数化应用是高度并行模型的一个特例。它们的特点是一个相互独立的任务集合，其中的任务是通过改变参数值的组合自动地从一个任务模板中生成的。在这种情况下，所执行的任务在计算逻辑上是相同的，但数据不同。因此，参数化应用程序也可以被认为是单程序多数据（SPMD）模型。MPI应用程序的特点是需要一起执行，并通过消息传递来交换数据的任务集合。尽管通过MPI应用程序任务执行的程序可能是相同的，但是提供一个实现逻辑，且该逻辑根据其等级区分每个任务的行为是很常见的。工作流应用程序的特点是，任务集合中任务之间的依赖关系可以用一个有向非循环图的术语表示。依赖关系大多通过文件表示，这些文件是特定任务的输出，并且都需要对相关任务的计算。任务的性质和每个任务所使用的计算种类通常是不一致的。

本章介绍了任务模型和Aneka服务的实现，Aneka支持基于任务的编程作为框架的实际例子，实现了基于任务的分布式应用程序的开发和执行。任务模型包括一组服务（目录、调度、执行和存储），它们相互协调构成了高度并行应用程序执行的运行时刻支持。就任务定义、提交、执行和文件的依赖关系管理方面而言，任务模型的基本特征由一个实际例子证明。在此基础上，客户端组件以及与其他技术的集成使供应商能够支持参数化和工作流应用。参数化应用程序通过参数化模型（PSM）实现，其特征在于具有一个能够为这种应用程序提供不同的，且更合适的接口的客户端组件集合。Aneka本身并不支持工作流应用程序，但与其他技术的集成使用户能够利用Aneka执行工作流。例如，一个使用Aneka任务提交Web服务的插件允许工作流引擎将Aneka作为工作流执行的后端使用。在Offspring中实现的Aneka分布引擎提供了另一个例子，说明了如何通过利用任务模型的基本API来快速原型化另一种编程模型（基于工作流的模型）。

习题

1. 什么是任务？任务计算如何与并行计算关联？
2. 列举并解释任务计算的计算类型。
3. 支持任务计算的框架的主要功能是什么？
4. 列举几种任务计算最常见的框架。
5. 任务包表示什么？
6. 举一个参数化应用程序的例子。
7. 什么是MPI？它的主要特点是什么？ 251

8. 什么是工作流？与高度并行应用相比，工作流应用模型的特性是什么？
9. 描述工作流管理系统的参考模型。
10. Aneka 如何支持任务计算？
11. 任务编程模型的主要组件是什么？
12. 描述 ITask 与 AnekaTask 的区别。
13. 描述静态与动态任务提交的区别。
14. 描述 Aneka 提供的基于任务应用的数据移动功能和一般架构。
15. 如何用任务编程模型运行旧版本应用程序？
16. Aneka 的任务编程模型与其他技术和平台有什么不同？
17. 用任务编程模型设计和实现简单的应用，按照 Riemann[⊖]提出的积分运算算法实现积分的非连续计算。
18. Aneka 为执行参数化应用程序提供的功能是什么？
19. Aneka 支持工作流任务执行吗？
252 20. 参考图 7-6 描述的蒙太奇工作流，简要设计在 Aneka 上执行工作流策略的控制流程。

⊖ http://en.wikipedia.org/wiki/Riemann_inegral。

数据密集型计算：MapReduce 编程

数据密集型计算是一类专注于处理大量数据的应用程序。从计算机科学到社交网络，这些应用领域产生了大量需要被有效地存储、访问、索引和分析的数据，而且信息量随时间以更高的速率累积和增加，使这些任务变得颇具挑战性。分布式计算无疑为应对这些挑战提供了帮助，它在数据计算和处理方面提供了更多的可扩展和高效的存储架构，具有更好的性能。尽管如此，利用并行和分布式技术来支撑数据密集型计算并不简单，仍然需要面对以数据表示、高效算法以及可扩展基础设施为形式的一些挑战。

本章描述了数据密集型计算的本质，并概述了大量数据的产生及其为存储系统和计算模型所带来的挑战。本章描述了 MapReduce，这是一个流行的编程模型，用于创建数据密集型应用以及将应用部署在云端。数据密集型计算的 MapReduce 应用实例使用的是 Aneka MapReduce 编程模型。

8.1 什么是数据密集型计算

数据密集型计算涉及数百 MB 到 PB 甚至更大规模数据的产生、处理和分析 [73]。术语数据集通常用于表示信息元素的集合，这些元素与一个或多个应用相关。数据集通常保存在库里，这是支持大量信息存储、检索以及索引的基础。为了便于分类和搜索，一部分相关信息需连接到数据集，这就是元数据。

数据密集型计算出现于许多应用领域，计算科学是最流行的应用领域之一。人们开展科学模拟和实验时经常会产生、分析和处理海量的数据。通过望远镜观测天空，每秒有数百 GB 的数据产生，一年多的天空图像的集合很容易达到 PB 级的数据规模。生物信息学应用挖掘到的数据库最终可能会包含 TB 级的数据。地震模拟器处理了大量的数据，这些数据是全球记录地球振动的结果。

除了科学计算，一些 IT 产业也需要数据密集型计算的支持。电信公司的顾客数据很容易达到 10 ～ 100TB 的范围，这些信息不仅用于生成账单报表，也可以通过挖掘来确定一些帮助公司提供更优服务的场景、趋势和模式。此外，据报道，美国手机移动流量已经达到每月 8PB 并且预计到 2015 年将增长到每月 327PB[⊖]。对于谷歌这样的 IT 巨头，PB 级的数据量更为普遍，据报道谷歌每天处理大约 24PB 的信息 [55]，每小时都在整理 PB 级的数据量[⊖]。目前数据密集型计算在社交网络和游戏这两个领域里已得到应用。Facebook 的收件箱搜索业务涉及抓取大约 150TB 的数据，并且存储在分布式基础设施的整个未压缩的数据达到了 36PB[⊖]。Zynga 这个社交游戏平台每天移动 1PB 的数据，据报道，Zynga 每周增加 1000 个服

⊖ Coda ResearchConsultancy, www.codaresearch.co.uk/usmobileinternet/index.htm。

⊖ Google's Blog, http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html。

⊖ OSCON 2010, DavidRecordon（SeniorOpenProgramsManager,Facebook）:Today'sLAMPStack,KeynoteSpeech。网址 www.oscon.com/oscon2010/public/schedule/speaker/2442。

务器用来存储像 Farmville 和 Frontierville 这样的游戏产生的数据[⊖]。

8.1.1　数据密集型计算特性

数据密集型应用程序不仅处理海量数据，而且经常表现出计算密集型的性能[74]。图 8-1 在两个上象限标识了数据密集型计算的域。

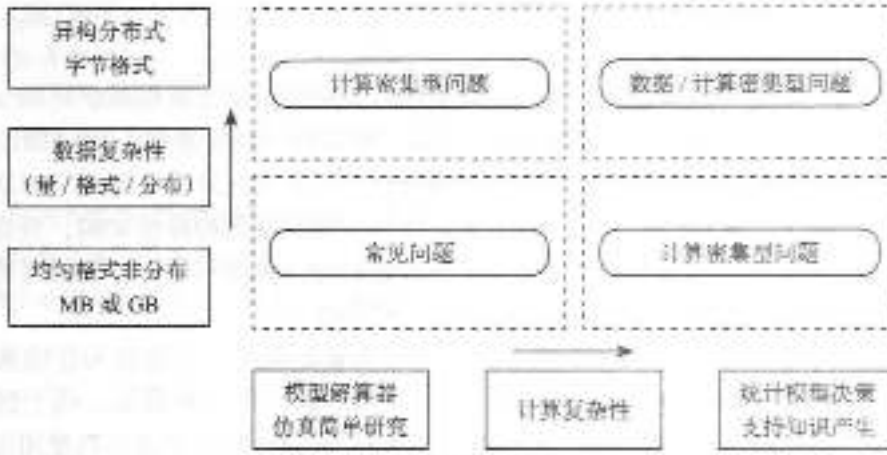


图 8-1　数据密集型研究问题

数据密集型应用处理多个 TB 和 PB 级规模的数据集。数据集通常存留几种格式并分布在不同的位置。应用程序以多步分析的渠道处理数据，包括转换和融合阶段。这样的处理要求数据大小几乎呈线性变化，并且容易并行处理。处理过程中还需要对于数据管理、过滤和融合的有效机制，以及高效的查询和分布[74]。

8.1.2　未来的挑战

海量数据的产生、分析或存储加强了对于支持基础设施和中间件的需求，而分布式计算的传统解决方案很难实现这一需求。例如，数据的位置是至关重要的，因为对于 TB 级数据的移动需求成为了高性能计算的障碍。数据分区、内容复制和可扩展的算法，有助于提高数据密集型应用程序的性能。数据密集型计算中开放的挑战由 Ian Gorton 等人[74]提出，如下：

254

- 可以搜集和处理大量数据集的可扩展算法。
- 可以扩展到处理复杂的、异构的和分布式数据源的新的元数据管理技术。
- 高性能计算平台的优势，旨在为访问内存中多 TB 数据结构提供更好的支持。
- 高性能、高可靠性、千亿万次的分布式文件系统。
- 用于数据还原和快速处理的数据签名生成技术。
- 实现软件可移植性的新途径，用于传送能够将计算移动到数据所在的位置的算法。
- 专门的混合互连架构，为过滤来自高速网络和科学仪器的多 GB 数据流提供的更好支持。
- 灵活高效的软件集成技术，便于结合不同平台上运行的软件模块以迅速形成分析通道。

⊖ http://techcrunch.com/2010/09/22/zynga-moves-1-petabyte-of-data-daily-adds-1000-servers-a-week/。

8.1.3　历史背景

数据密集型计算包括海量数据的产生、管理和分析。存储、网络技术、算法和基础软件 [255]
一起为数据密集型计算提供支持。本节通过重点介绍在存储、网络及基础软件领域最为相关的贡献来追踪这一现象的演变。

1. 早期时代：高速宽带网络

数据传输和数据流方面的技术、协议和算法的演变已经成为数据密集型计算的推动者[75]。1989 年，作为科学数据远程可视化支持的第一个高速网络实验在业界遥遥领先。两年之后，利用高速宽带网络的潜在性在 1991 年的超级计算会议（SC91）中被证实，它使得高速且基于 TCP/IP 协议的分布式应用程序成为现实。当时，大量复杂的科学数据集（一个高分辨率的磁共振图像，或称为 MRI，人类大脑的扫描）的远程可视化在匹兹堡超级计算中心和新墨西哥州阿尔伯克基（那次会议的召开地）之间建立。

Kaiser 项目[76]迈出了更深远的一步，使得远程数据源的高数据传输速率和在线仪器系统成为可能。该项目利用广域大型数据对象（WALDO）系统[77]，以提供以下性能：元数据的自动生成；实时处理数据，同时对数据和元数据自动编目；通过提供本地和远程用户对数据的访问来促进合作研究；把数据存入数据库和其他文件的机制。

据报道，MAGIC 项目构建了第一个数据密集型环境，这是一个 DARPA 资助的研究大规模、高速网络中分布式应用程序的合作项目。在这一背景下，开发出了分布式并行存储系统（DPSS），之后被用来支持 TerraVision[78]，这是一个让用户探索和导航立体实景的地形可视化应用。

另一个重要的里程碑是 Clipper 项目[⊖]的建立，这是几个科学研究实验室合作努力的结果，目标是设计和实现一个独立的但结构一致的服务组件的集合，用来支持数据密集型计算。Clipper 项目提出的挑战包括大量计算资源的管理、高速率高容量数据流的产生和消费、用户交互管理、分散资源的聚合（多数据归档、分布式计算容量、分布式缓存容量和保证网络容量）。Clipper 的焦点是开发一个服务的协调集合，可以用于各种应用程序，以构建按需、大规模、高性能、广域解决问题的环境。

2. 计算网格

随着网格计算[8]的出现，人们可以跨管理域利用异构资源获得大量计算能力和存储容量。在这样的背景下，数据网格[79]作为支持数据密集型计算的基础设施出现了。数据网格提供了帮助客户发现、转换和操作存储在分布式数据库的大量数据集的服务，并且可以创建和管理它们的备份。数据网格提供了两个主要功能：高性能和可靠的文件传输以实现大量 [256]
数据的移动，可扩展的副本发现和管理机制以便容易地访问数据集[80]。因为数据网格跨越不同的管理边界，所以访问控制和安全是重要的内容。

大部分数据网格提供存储和数据集管理能力以作为产生大量数据的科学实验的支持。图 8-2 为一种可能的一参考情景。海量数据由科学设备（望远镜、粒子加速器等）产生，之后，可以在本地处理的信息被存储在数据库中，以供本地甚至远程的科学家实验和分析。科学家可以利用特定的发现和信息服务，这有助于确定他们最感兴趣的数据集的位置。数据集由基础设施复制，以提供更好的可用性。由于处理这些信息也需要很强的计算能力，因此可以访问特定的计算站点以便执行分析和实验。

⊖ www.nersc.gov/news/annual_reports/annrep98/16clipper.html。

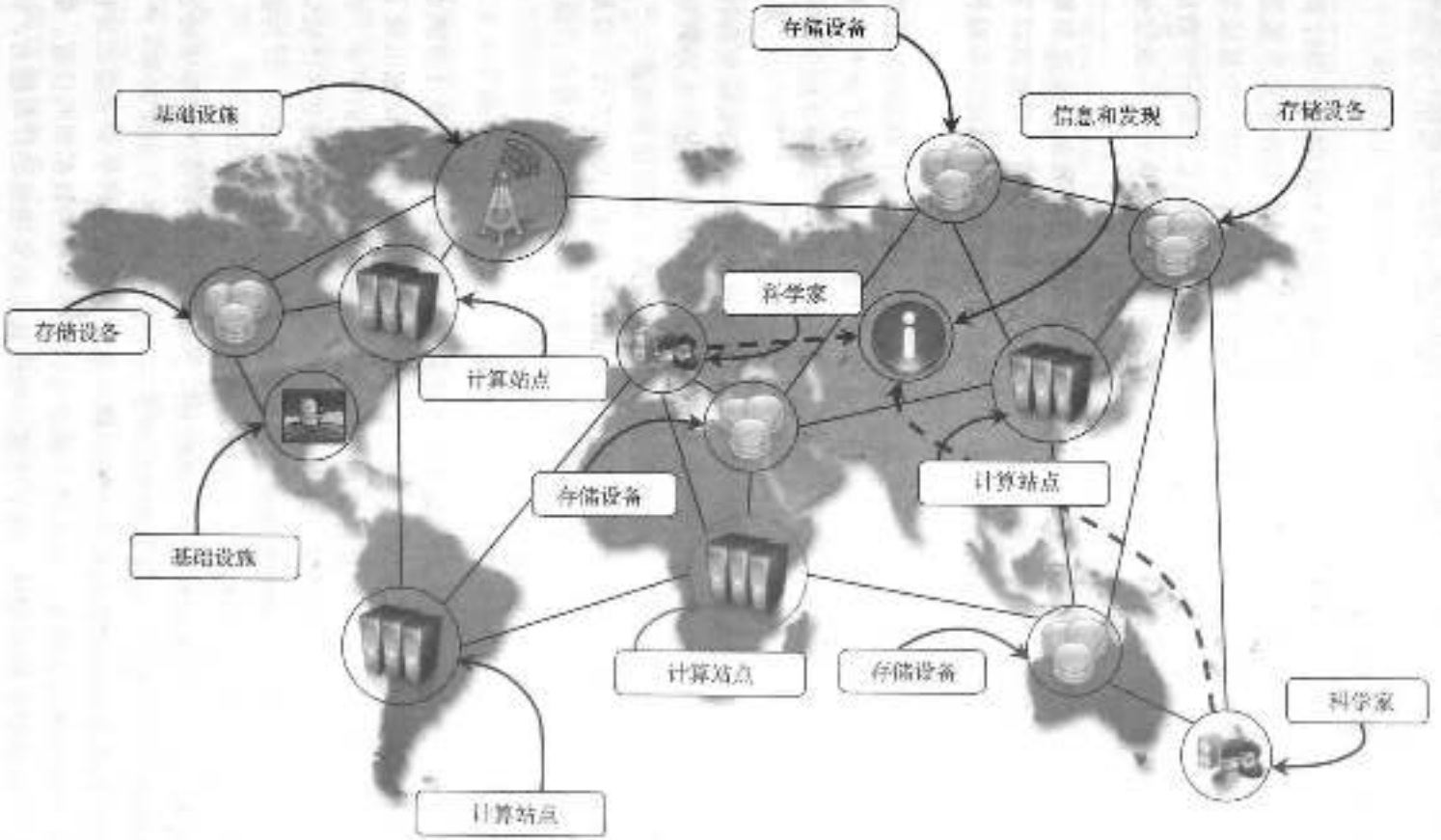


图 8-2 数据网格的参考场景

像任何其他的网格基础设施一样，由于资源和不同管理域的异构性，因此需要适当强调安全措施和虚拟组织（VO）的使用。除了异构性和安全性，数据网格还具有自己的特征并且引入了新的挑战 [79]：

- 大量数据集。数据集的大小很容易达到 GB、TB 以及更大的规模。因此在批量传输、用恰当的方法复制内容以及管理存储资源时将延迟降到最低是必要的。
- 共享数据集。资源共享包括数据的分布式收集。例如，数据库可以用来存储和读取数据。
- 统一的命名空间。数据网格利用了一个统一的定位数据集合和资源的逻辑命名空间。每一个数据元素有一个单独的逻辑名称，它最终被映射到不同的物理文件名，以便进行复制和访问。
- 访问限制。虽然数据网格的目的之一是便于实验结果和数据的共享，但一些用户可能想要确保其数据的机密性，并限制他们的合作者对这些数据的访问。身份验证和数据网格授权涉及共享数据集合中粗粒度和细粒度的访问控制。

相对于高速网络计算设施的组合，数据网格是一种更加结构化和综合化的进行数据密集型计算的方法。因此，包括高能物理、生物和天文学等科学研究领域都利用了数据网格，简要讨论如下：

- LHC 网格。这是由欧洲联盟资助的一个项目，目的是开发一个全球的网格计算环境，为与 LHC 实验合作的来自全球的高能物理研究者提供服务。它支持由 LHC 实验产生的从几百 TB 到 PB 的大规模数据集的存储和分析（http://lhc.web.cern.ch/lhc 1）。
- 生物信息学研究网络（BIRN）。BIRN 是美国建立的通过数据共享和在线合作来推动生物医学研究的创新项目。由国家研究资源中心（NCRR）资助，作为美国国家健康机构（NIH）的构成要素，BIRN 提供数据共享基础设施、软件工具、方法和咨询服务（www.birncommunity.org）。 257
- 国际虚拟天文台联盟（IVOA）。IVOA 组织旨在提供更好的对于不断扩展的天文数据资源的网络访问。实现方法是提高虚拟天文台的标准，虚拟天文台是互操作数据归档和软件工具的集合，这些软件工具利用网络形成能引导天文学研究项目的科学研究环境，允许科学家发现、访问、分析和组合来自异构数据集合的实验数据。

数据网格的完全分类法可以参考 Venugopal 等 [79] 的文章。

3. 数据云和“大数据”

大量数据集大部分已经成为科学计算的范畴。由于海量数据正在被那些提供搜索、网上广告和社会媒体等服务的公司产生、挖掘、处理，这一情景最近开始改变。对于这样的公司，有效地分析这些大量数据集是至关重要的，因为它们构成了有关其客户信息的宝贵资源。日志分析是在这样的背景下普遍进行的数据密集型操作的一个例子：像谷歌这样的公司每天都会产生由其分布式基础设施处理的日志形式的海量数据。因此，他们采用分析型基础设施，而不是科学界使用的基于网格的基础设施。

随着支持数据密集型计算的云计算技术的扩散，大数据 [82] 一词已经流行起来。这一词汇描述了当今数据密集型计算的本质，而且可以鉴别现在的数据集，它们的增长量太大以至于用手持的数据集管理工具来工作会变得复杂。关系型数据库和桌面统计 / 可视化软件包对于海量信息变得无效，取而代之的，是“运行在十、百甚至上千的服务器上的大量并行软件”[82]。

大数据问题出现在非科学的应用领域，例如网络日志、射频识别（RFID）、传感器网络、社交网络、互联网文字和文件、互联网搜索索引、呼叫详细记录、军事侦察、医疗记录、照片档案、视频档案和大型电子商务。除了数量巨大，这些例子表现出来的是新数据随时间累积，而不是代替旧数据。通常，大数据一词适用于数据集的大小超出了常用软件工具在可容忍的运行时间内捕获、管理和处理的能力。因此，大数据的量是一个不断移动的目标，目前在一个数据集中从几十 TB 到多 PB 级变化。

云计算以如下几种方式支持数据密集型计算：

- 提供大量的实时计算实例，用于并行处理和分析大量数据集。
- 提供优化的存储系统，用于保存大文件数据以及其他分布式数据存储架构。
- 提供构架和可编程 API，以便处理和管理海量数据。这些 API 大多数附加一个特定的用来完善系统整体性能的存储基础设施。

258
/
259

数据云是这些因素的结合，MapReduce 构架 [55] 就是一个例子，它提供了在谷歌大型计算设施的顶层利用谷歌文件系统 [54] 的最好性能。另一个例子是 Hadoop 系统 [83]，它是最成熟的大型开源数据云。它由 Hadoop 分布式文件系统（HDFS）和 Hadoop 的 MapReduce 实现构成。Sector[84] 提出了一个相似的方法，Sector 由 Sector 分布式文件系统（SDFS）和一个名为 Sphere[84] 的计算服务构成，Sphere 允许用户通过由 SDFS 管理的数据执行任意的用户定义功能（UDF）。Greenplum 使用基于商用硬件的无共享的大型并行处理（MPP）架构。这个架构也把类似 MapReduce 的性能加入自己的平台。Aster 开发了一个相似的架构，它采用了基于 MPP 的支持 MapReduce 的数据仓库设备，数据目标为 1PB。

4. 数据库和数据密集型计算

传统意义上，随着仅由一个系统构成的分布式数据集 [85] 变得难以管理，分布式数据库已被认为是数据库管理系统的自然进化。分布式数据库是存储在不同地点的计算机网络中的数据的集合。每个站点可能会提供一定程度的自主权，为本地应用程序的执行提供服务，但是也参与全球应用程序的执行。一个分布式数据库可以通过分裂和散射在不同站点的现有数据库的数据或由多个现有的数据库联合建立。这些强大的系统提供分布式事务处理、分布式查询优化和资源的高效管理。然而，它们大多关注可以用关系模型 [86] 来表示的数据集，并且需要执行数据的 ACID 特性限制了它们的扩展能力，就如同数据云和网格一样。

8.2　数据密集型计算技术

数据密集型计算关注那些侧重海量数据处理的应用程序的发展。因此，存储系统和编程模型成为支持数据密集型计算技术的两大分类。

8.2.1　存储系统

传统上，数据库管理系统构成了对几种类型的应用程序的实际存储支持。由于日志、网页、软件日志和传感器读数等形式的非结构化数据的爆炸，因此原有形式的关系模型似乎不是支持大规模数据分析的首选解决方案。针对数据库和数据管理行业的研究确实处于一个转折点，新的机遇出现了。导致这一变化的因素有：

- 大数据越来越受欢迎。大量数据的管理不再是罕见的情况，相反在很多领域都变得普遍：科学计算、企业应用程序、媒体娱乐、自然语言处理和社交网络分析。大量的数据都在利用新的和更有效的数据管理技术。

260

- 商业性的数据分析日益重要。数据管理不再被认为是代价，而是商业利益的关键元素。这一情况出现在受欢迎的社交网络中，就像 Facebook，它把焦点集中在管理用户利益、兴趣和人们之间的联系。不断挖掘出来的海量数据需要新的技术和策略来支持数据分析。
- 数据形式多样，不只是结构化。正如先前提到的，如今构成相关信息的事物显示出了异构的性质，并以几种形式和格式出现。随着传统企业应用程序和系统的继续使用，结构化数据不断增长，但同时互联网技术和民主化的发展，让每个人都可以提取信息，产生了海量的非结构化的和本来不适合关系模型的信息，互联网成为了一个每个人都能收集信息的平台。
- 计算的新方法和新技术。云计算确保了可以获取所需的海量计算能力。这允许工程师设计可以逐步扩展到任意并行度的软件系统。建立动态部署在数百或数千节点上的软件应用和服务不再罕见，这些节点可能只在几个小时或几天中属于这个系统。传统的数据库设施将不会用于对这样的不稳定环境提供支持。

这些因素确定了对于新的数据管理技术的需求。这不仅意味着数据库技术的一个新的研究议程和一个更全面的信息管理方法，还为关系模型的替换（或补充）留有空间。特别地，为了管理以文件、分布式对象存储和 NoSQL 运动的扩展为形式的原始数据的分布式文件系统的发展，构成了支持数据密集型计算的主要方向。

1. 高性能分布式文件系统和存储云

分布式文件系统是数据管理的基本支持。它们提供了一个接口来存储文件形式的信息，之后可通过读和写的操作访问数据。在文件系统的若干实现中，很少能够具体解决大量节点上的海量数据管理。大多数文件系统构成了数据存储支持，这些支持针对大量计算群、超级计算机、大量并行架构，以及最近出现的存储 / 计算云。

Lustre。Lustre 文件系统是一个大量并行分布式文件系统，它覆盖了从小的工作集群到大规模计算集群的需求。这个文件系统用于几个排名前 500 超级计算系统[⊖]，包括在 2012 年 6 月的名单上排名第一的超级计算机。Lustre 可提供存储 PB 级的访问，以每秒数百 GB 的 I/O 传送率服务数以千计的用户。这个系统由元数据服务器组成，包括文件系统的元数据，以及负责提供存储的对象存储服务器的集合。用户通过兼容 POSIX 的客户端来访问文件系统，此客户端既可以是安装在内核中的模块，又可以通过库来操作。这个文件系统实现了一个强大的故障转移策略和恢复机制，使得服务器的故障和恢复对于客户端是透明的。 261

IBM 通用并行文件系统（GPFS）。GPFS[88] 是 IBM 开发的高性能分布式文件系统，它提供对于 RS/6000 和 Linux 计算集群的支持。GPFS 是一个历经多年的学术研究建立的多平台分布式文件系统，并提供了先进的恢复机制。GPFS 建立在共享磁盘上，其中磁盘的集合由某些交换方式附加到文件系统的节点上。该文件系统使这一基础设施对于用户透明，并且通过将部分文件复制到磁盘阵列上为大型文件加条纹，以保证高可用性。通过这个基础设施的方法，该系统能支持 PB 级的高传送率存储，而且不会丢失数据的一致性。与其他实现相比，GPFS 把整个文件系统的元数据进行分布并提供对它的透明访问，以此消除单故障点。

谷歌文件系统（GFS）。GFS[54] 是一个支持谷歌计算云中分布式应用程序执行的存储设施。这个系统是一个建立在商用硬件和标准 Linux 操作系统上的容错的、高可用性的、分

⊖ Top 500 supercomputers list: www.top500.org (accessed in June 2012)。

布式的文件系统。这不是一个分布式文件系统的通用实现，GFS 专门针对谷歌在应用程序的分布式存储方面的需求，按照以下假设进行设计：

- 该系统建立在经常出错的商用硬件的顶层。
- 该系统存储适量的大文件。多 GB 文件是普遍的而且应该得到有效的处理，小的文件必须被支持，但没必要优化它。
- 工作负载基本由两种读取构成：大型流读取和小型随机读取。
- 工作负载也具有将数据追加到文件的许多大量和连续的写入。
- 带宽的高持续比低延迟更重要。

该文件系统的架构组织成一个单一的主设备，它包含整个文件系统的元数据，以及一个提供存储空间的块服务器的集合。从逻辑上看，该系统由软件后台程序的集合组成，既实现了主服务器又实现了块服务器。文件是块的集合，块的大小可以由文件系统级别确认。块被复制在多个节点以实现容错功能。客户查询主服务器并鉴别他们想要访问的文件的具体块。一旦这个块被确定，客户和块服务器之间的联系就发生了。应用程序通过文件系统与一个特定的界面交互，该界面支持对文件的建立、删除、读取和写入的常规操作。这一界面也支持快照和记录追加的操作，这些操作被应用程序频繁地执行。GFS 一直认为大型分布式基础设施中的失败是普遍的而不是罕见的，因此，一个高可用性的、轻量级的和容错的基础设施被给予特别的关注。通过给予在其他任意属于该设施的节点上复制主节点的可能性，单主架构故障的潜在单一节点已得到解决。此外，无状态的守护进程和广泛的日志记录能力有助于故障系统的恢复。

262

Sector. Sector[84] 是支持执行数据密集型应用程序的存储云，它根据 Sphere 构架定义。这是一个跨广域网的可应用于商用硬件的用户空间文件系统。与其他文件系统相比，Sector 不把文件分割成块，而是在多个节点上复制整个文件，这就允许用户自定义复制策略以达到更优的性能。该系统的架构由四个节点组成：一个安全服务器，一个或更多的主节点，从节点和客户端机器。安全服务器维护所有对用户和文件的访问控制策略的信息，而主服务器协调和服务于用户的 I/O 请求，这些请求根本上是与从节点交互进行文件的访问。用于与从节点交换数据的协议是 UDT[89]，它是一个轻量级的广域网优化的面向连接的协议。

亚马逊简单存储服务（S3）。亚马逊 S3 是亚马逊提供的在线存储服务。尽管它的内在细节没有公开，但该系统声称支持高可用性、可靠性、可扩展性、无限存储和低延迟的商业价值。该系统提供组织成桶的平面存储空间，并连接到亚马逊网络服务（AWS）账户。每一个桶能够存储多个对象，每个对象以一个唯一的关键词来识别。对象以唯一的 URL 被识别并通过 HTTP 呈现出来，允许非常简单的读 / 取语义。由于 HTTP 的使用，因此不需要任何特定的库来访问存储系统，其中的对象也可以通过 Bit Torrent 协议[⊖]检索。尽管语义简单，但作为本地文件系统的一部分，一个类似 POSIX 的客户端库已经发展并安装到 S3 存储桶。除了最小语义，安全性是 S3 的另一个限制。对象的可见性和可访问性与 AWS 客户相连，桶的拥有者可以将其设置为对于其他客户或公众可见。定义验证的 URL 也是可能的，这提供了在有限的（可设置的）时间内对于任何人的公共访问权。

除了这些存储系统的例子，还有其他分布式文件系统和存储云的实现，它们与这里讨论

⊖ Bit Torrent 是 P2P 文件共享协议，用于分布式大量数据。该协议的关键特性是支持用户从多主机并行下载文件。

的模型架构类似。除了 S3 服务，可以在所有提到的系统中简述一个通用的参考架构，它确定了两个主要角色，在这两个角色中可对所有节点进行分类。元数据或主节点包含了关于文件和文件块位置的信息，而从节点用于提供对存储空间的直接访问。这个架构通过客户库来完成，它提供了一个简单的访问文件系统的接口，这个接口在一定程度上可以完全兼容 POSIX 规范。参考架构的变化可以包括支持多主节点、分割元数据到多节点，以及节点角色的轻松转换能力。所有不同的实现中，最重要的是提供容错性和高可用性的存储系统的能力。

2. NoSQL 系统

NoSQL（Not Only SQL）一词最早于 1998 年为定义一个关系型数据库而提出，它没有公开用于操作和查询数据的 SQL 接口，而是依靠一套 UNIX 软件脚本和命令来操作包含实际数据的文本文件。从非常严格的意义上来说，NoSQL 不是关系型数据库，因为它不是一个根据关系模型来组织信息的软件，而是一个脚本的集合，它允许用户将文本文件作为信息存储来管理大部分最简单的和更普通的数据库任务。之后，在 2009 年，为了标注某些数据库管理系统，术语 NoSQL 再次被引入，那些系统没有使用关系模型，而是提供了对于数据处理的更简单和更快速的选择。如今，NoSQL 一词是一类存储和数据库管理系统的总称，这类系统在某些方面与关系模型不同。它们的一般理念是克服由关系模型带来的限制，并提供更多有效的系统。这意味着不使用固定模式的标签，以适应更大范围的数据类型或避免横向扩大性能和规模。 [263]

两个主要的因素决定了 NoSQL 运动的发展：在很多情况下简单的数据模型足够代替应用程序中使用的信息，而且非结构格式中包含的信息数量在最近十年已经大量增长。这两个因素使软件工程师要寻找更适合他们正在从事的特定应用领域的替代品。因此，一些不同的倡议探索了非关系型存储系统的使用，这些系统各自不同。Wikipedia 给出了一个广义的分类[⊖]，这个分类把 NoSQL 的实现分为：

- 文档存储（Apache Jackrabbit，Apache CouchDB，SimpleDB，Terrastore）。
- 图表（AllegroGraph，Neo4j，FlockDB，Cerebrum）。
- 键 - 值存储。这是一个宏分类，可进一步分类成磁盘上的键 - 值存储、RAM 中的键 - 值缓存、键 - 值存储等级、最终一致性键 - 值存储和排序键 - 值存储。
- 多值数据库（OpenQM，Rocket U2，OpenInsight）。
- 对象数据库（ObjectStore，JADE，ZODB）。
- 表格存储（谷歌 BigTable，Hadoop HBase，Hypertable）。
- 元组存储（Apache River）。

下面介绍一些支持数据密集型应用程序的显著成就。

Apache CouchDB 和 MongoDB。 Apache CouchDB[91] 和 MongoDB[90] 是文档存储的两个例子。它们都提供无模式存储，其中主要的对象是组成一个键 - 值域集合的文件。每个域的值可以是字符串类型、整数、浮点数、日期或数值数组。该数据库公开一个 RESTful 接口，并用 JSON 格式表示数据。这些都允许通过 MapReduce 编程模型查询和搜索数据，使用 JavaScript 而不是 SQL 作为数据查询和操作的基本语言，并支持大文件文档。从基础设施的角度，这两个系统支持数据复制和高可用性。CouchDB 获得了数据的 ACID

⊖ http://en.wikipedia.com/wiki/NoSQL。

属性。MongoDB 支持分片，这是分割不同节点集合的内容的能力。

亚马逊 Dynamo。Dynamo[92] 提供分布式键 – 值存储，它支持几种由亚马逊公司提供的商业服务的信息管理。Dynamo 的主要目标是提供一个可扩展规模的高可用性的存储系统。
264 这个目标有助于实现大规模的可靠性，其中，由数千服务器和网络部件构建的基础设施为每天千万的请求提供服务。Dynamo 提供一个基于读 / 取语义的简单化接口，对象通过接口以唯一的标志（键）被存储和取回。构建一个极其可靠的基础设施的主要目标已经对这些系统的性质强加了一些限制。例如，为了追求更可靠和有效的基础设施，牺牲了 ACID 的数据性能。这创建了最终一致性模型，它意味着在很长一段时间内所有用户将看到一样的数据。

Dynamo 系统的架构如图 8-3 所示，由存储节点的集合组成，这些节点组成了一个为应用程序共享键间距的环。键间距在存储节点间分开，而且键通过环进行复制，而不是通过临近的节点。每一个节点具有本地存储设施的访问权，设施中存储着原始对象和复件。更进一步，每一个节点提供分割环间的更新并探测错误和未达到节点的能力。随着一些用于复制和对象版本控制的一致性模型规范的放松，Dynamo 实现了总是写入存储的性能，该存储在后台解决数据的一致性问题。这个方法的缺点是存储模型的简单化，它需要应用程序在存储系统提供的简单构件的顶层建立自己的数据模型。例如，没有可参照的完整性约束，关系没有被嵌入存储模型中，因此不支持连接操作。这些限制在亚马逊服务的情况下没有被禁止，因
265 此单一键 – 值模型是可以接受的。

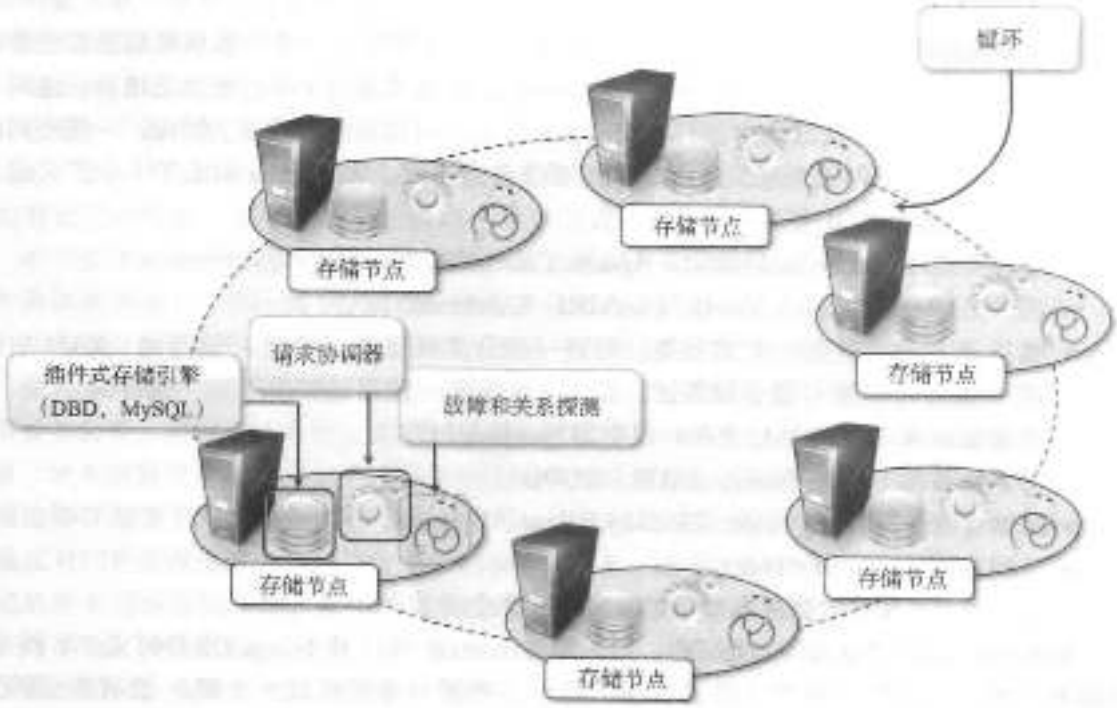


图 8-3　Amazon Dynamo 架构

谷歌 Bigtable。Bigtable[93] 是分布式存储系统，用于在数千服务器上处理高达 PB 级规模的数据。Bigtable 提供对于一些可承受不同类型工作负载的谷歌应用程序的存储支持：从面向吞吐量的批处理作业到终端用户数据的延迟敏感性。Bigtable 的设计目标是广泛的适用性、扩展型、高性能和高可用性。为了达到这些目标，Bigtable 用表格组织数据存储，表

格的行在支持中间件的分布式文件系统（谷歌文件系统）中被分割。从逻辑角度看，一个表格是一个以关键字索引的多维有序映射，关键字以任意长度的字符串表示。表格被组织成行和列，列可以被组织成列族，允许特定的优化以实现更好的访问控制、存储和索引数据。一个简单的数据访问模型构成了客户应用程序的接口，应用可以通过一行中单一列的粒度水平找到数据。此外，每一个列值以多版本存储，这些版本可以由 Bigtable 或客户应用程序自动标记时间。

除了基本的数据访问，Bigtable 的 API 也允许更复杂的操作，例如通过 Sazwall[⊖]脚本语言 [95] 或 MapReduce 的 API 进行的单行事务和高级数据操作。

图 8-4 给出了 Bigtable 基础设施的描述。服务是一些进程的集合，这些进程与其他进程共存于一个基于集群的环境。Bigtable 定义了两种进程：主进程和子表服务进程。一个子表服务器负责服务一个子表请求，子表是一个表格中一部分连续改行。每一个服务器可以管理多个子表（通常是 10 ~ 1000 个）。主服务器负责跟踪子表服务器状态和子表服务器配置的子表的状态。服务器不断地监视子表服务器以检查它们是否正常，如果发现子表服务器不可连接，配置的子表将被重新安排并最终分配给其他服务器。

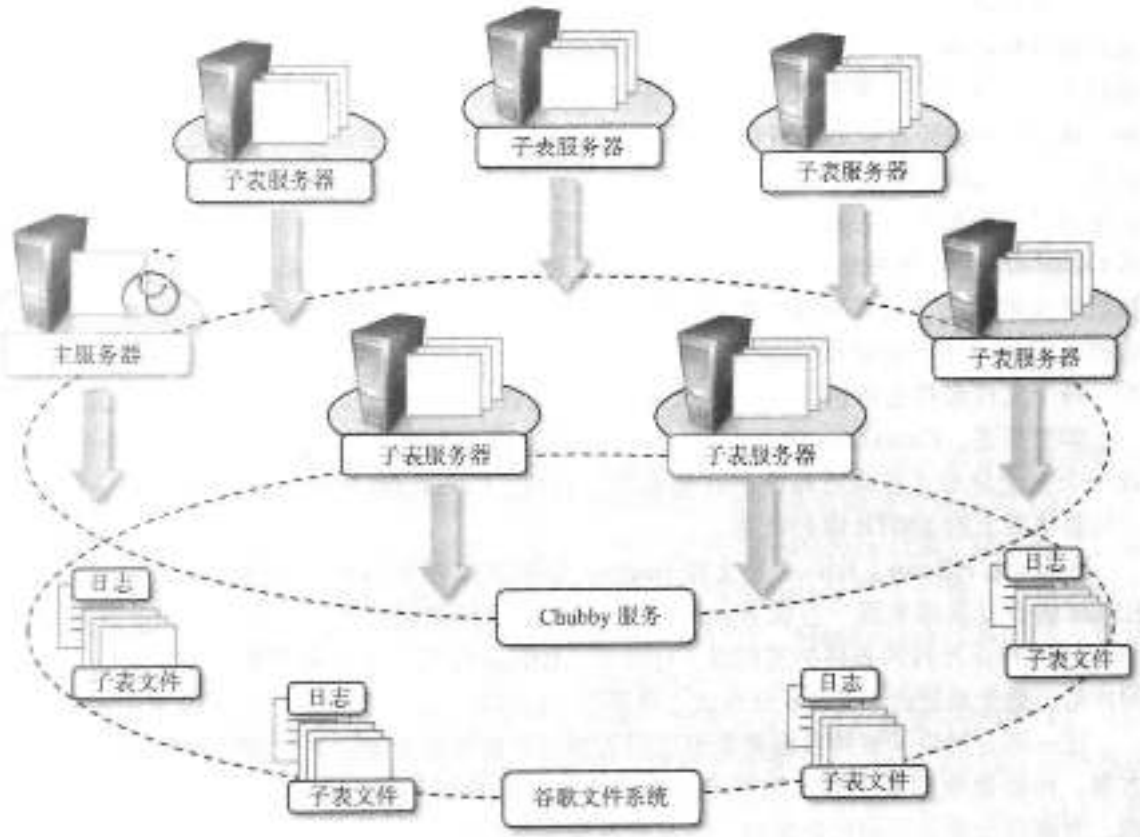


图 8-4　Bigtable 架构

Chubby[96] 是一个分布式、高可用性且持久的锁服务——支持主服务器和子表服务器的活动。Chubby 负责系统监视和数据访问，也负责管理复件并为其提供一致性。在最底层，

⊖ Sazwall 是 Google 开发的一种解释程序编程语言，用于操作大数据。其特定功能是支持由读取值或者从输入计算值得到统计和，其他功能则为简单的 PB 级数据的并行处理。

数据以文件的形式存储在谷歌文件系统，且所有更新操作都被复制在文件中，以便遇到失败或当子表需要被重新分配给其他服务器时，数据能够简单地进行恢复。Bigtable 使用特定的文件形式以存储子表数据，它可以被压缩以便优化数据的访问和存储。

Bigtable 是谷歌针对分布式应用需求的研究结果。它作为一个存储的后端服务于 60 个应用程序（如谷歌个性化搜索、谷歌分析、谷歌财经和谷歌地球），并管理 PB 级的数据。

Apache Cassandra。Cassandra[94] 是一个分布式对象存储，以管理分散在许多商用服务器中的海量结构化数据。这个系统用于防止单一故障点并提供高可靠性的服务。Cassandra
266 最初由 Facebook 开发，现在它是 Apache Incubator 计划的一部分，为 Facebook、Digg 和 Twitter 等超大型的 Web 应用提供存储支持。Cassandra 被定义为第二代分布式数据库，它建立在遵循完全分布式设计的亚马逊 Dynamo，以及由此继承了“列族”概念的谷歌 Bigtable 的理念上。Cassandra 揭示的数据模型基于表格的理念，实现了以关键字索引的分布式多维表。对应于一个键的值是一个高度结构化的对象，并构成了表中的行。Cassandra 将表格的行组织为列，并且一组列可以被组织成列族。系统提供的用于访问和操作数据的 API 非常简单：插入、取回和删除。插入以行级别执行，取回和删除可以在列级别操作。

在基础设施方面，Cassandra 和 Dynamo 很相似。Cassandra 用于增值扩展，以及组织节点的集合在环中共享键间距。每一个节点管理键间距的多个且不连续的部分，并复制它的数据到 N 个其他节点。复制使用不同的策略，可以是机架可知、数据中心可知，或机架不可
267 知。这意味着决策要考虑复制是否需要在相同的集群或数据中心，或可以不考虑节点的地理位置。在 Dynamo 中，节点关系信息基于 gossip 协议[⊖]。其他任务也可利用 Cassandra 的信息扩散模式，例如传播系统的控制状态。每个节点的本地文件系统用于实现数据持久性，而且 Cassandra 广泛使用提交日志，以使系统能够从短暂的故障中恢复。每个写操作只有已经被记录在磁盘后才能在内存中应用，以便遇到故障时可容易地重新生成。当内存中的数据达到特定的大小，它就被转存到磁盘上。读操作先在内存中后在磁盘上执行。为了加快进程，每个文件包括它所包含的键的概要，使得搜索一个关键字时可避免不必要的文件扫描。

如前所述，Cassandra 建立在 Dynamo 和 Bigtable 的设计理念上并把它们结合起来以实现一个完全分布式和高可靠性的存储系统。目前，Cassandra 的最大部署可管理分布在 150 个机器集群上的 100TB 级的数据。

Hadoop HBase。HBase 是支持 Hadoop 分布式编程平台的存储需求的分布式数据库。HBase 的设计灵感来源于谷歌 Bigtable，它的主要目标是通过利用商用硬件的集群来提供对于十亿行和百万列的表格的实时读 / 写操作。HBase 内部的架构和逻辑模型与谷歌 Bigtable 很相似，整个系统由 Hadoop 分布式文件系统（HDFS）支持，模仿了 GFS 的结构和服务。

这一部分讨论了支持数据密集型应用管理的存储解决方案，尤其那些涉及大数据的解决方案。传统数据库系统最可能基于关系模型，是处理海量数据的首要方法。如同我们讨论的，当提到海量非结构化数据时，关系型数据库变得不现实并且性能较差。可替代的和更有效的方法已经显著地检验了基于分布式文件系统和存储系统的基本概念。下一阶段是提供编程平台，该平台利用本节提到的存储系统，通过开发者的努力来处理海量数据。在编程平台中，MapReduce 及其所有演化版本扮演了一个重要的角色。

⊖ Gossip 协议是一种受社交网络中流言传播启发的通信协议，是一种用于分布式系统的可替代的信息分布和传播方法，效率与洪泛及其他类似算法相当。

8.2.2　编程平台

数据密集型应用的编程平台提供了抽象概念，有助于描述大量信息的计算，且能够有效管理海量数据的运行时系统。传统上，基于关系模型的数据库管理系统已经被用来描述数据模型实体之间的结构和联系。这种方法在大数据的情况下被证明是不成功的，这里的信息大部分是非结构化的或半结构化的，并且在数据库中数据大多可能以大型文件或大量中等大小文件的形式组织，而不是行的形式。分布式工作流经常用来分析和处理大量数据[66，67]。该方法引入了针对工作流管理系统的大量框架，如 7.2.4 节所讨论的，最终利用云计算提供 268
的弹性特征合并各项功能[70]。这些系统主要基于任务的抽取，任务给开发者带来很大的负担，他们需要处理数据管理和经常出现的数据转换的问题。

数据密集型计算的编程平台提供更高级别的抽取，专注于数据处理并将传输管理移动到运转时系统，这样使数据在需要的地方总是具备可用性。这是 MapReduce[55] 编程平台遵循的方法，它用两种简单的功能形式描述计算——map 和 reduce——并隐藏了管理支持这一平台的分布式文件系统中海量数据文件的复杂性。本节讨论 MapReduce 的特性及其变化，这些变化扩展了 MapReduce 的性能，使其应用范围更加广泛。

1. MapReduce 编程模型

MapReduce[55] 是谷歌引入的用于处理海量数据的编程平台，它通过两个简单的函数表达应用的计算逻辑：map 和 reduce。数据转换和管理完全通过分布式存储基础设施处理（如谷歌文件系统），负责提供数据访问、文件复制以及将最终将文件移动到需要的地方。因此，开发者不必再处理这些问题，而是可以利用更高级别的数据表示接口：键 – 值对的集合。MapReduce 应用程序的计算被组织成 map 和 reduce 操作的工作流，它完全由运转时间系统控制，开发者仅需要详细说明 map 和 reduce 如何操作键 – 值对。

MapReduce 模型精确表示为两个函数形式，定义如下：

$$\text{map}(k1, v1) \rightarrow \text{list}(k2, v2)$$

$$\text{reduce}(k2, \text{list}(v2)) \rightarrow \text{list}(v2)$$

map 函数读取一个键 – 值对并产生一列不同类型的键 – 值对。reduce 函数读取由键和一列值组成的一个对，并产生一个相同类型的值的列表。两个函数描述中使用的（k1，v1，k2，v2）类型说明了这两个函数如何结合起来执行以进行 MapReduce 计算：map 任务的输出根据相应的键将值分组，以这种方式合并在一起，并构成 reduce 任务的输入，对于每个找到的键，此任务就是将附加值的列表减少为单个值。因此，MapReduce 计算的输入被描述成键 – 值对 <k1，v1> 的集合，最终输出由值的列表表示：list（v2）。

图 8-5 描绘了 MapReduce 计算的参考工作流程。如图所示，用户提交一个以 <k1，v1> 对形式描述的文件集合并指定 map 和 reduce 函数。这些文件进入支持 MapReduce 的分布式系统，并且在必要时被分割为 map 任务的输入。map 任务产生了存储 <k2，list（v2）> 对的中间文件，这些文件被保存在分布式文件系统。MapReduce 运行时可最终聚集与相同键相关的值。这些文件组成 reduce 任务的输入，以 list（v2）的形式产生输出文件。由 reduce 269
任务执行的操作通常代表由一个特定键 map 的所有值的集合。map 的数量和创建的 reduce 任务、文件根据其任务被划分的方式以及连接到单一 reduce 任务的 map 任务的数量，都是 MapReduce 运行时需要负责的。此外，支持 MapReduce 的分布式文件系统还要负责文件的存储和移动。

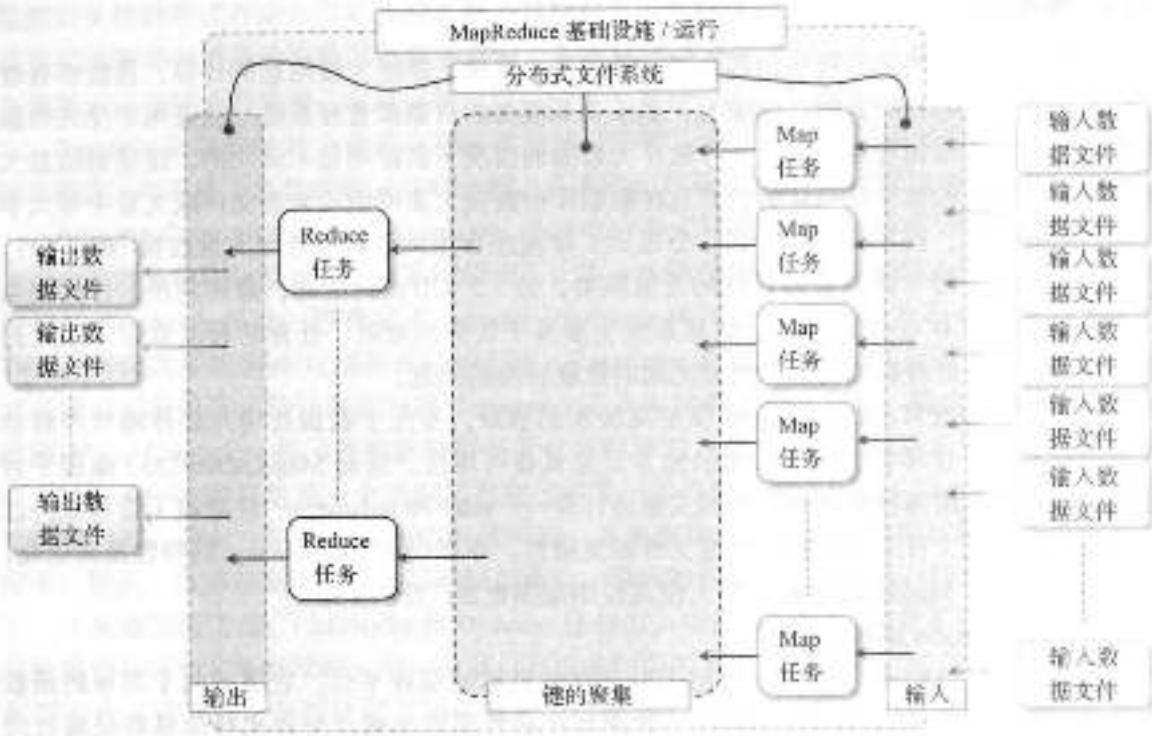


图 8-5　MapReduce 计算工作流

MapReduce 表示的计算模型非常简单，为编码处理大量数据算法的人员提供了更高的效率。这一模型在谷歌的案例中已被证明是成功的，这里需要被处理的大部分信息以文本的形式存储，并以网页或日志文件表示。一些展示 MapReduce 灵活性的例子如下 [55]：

- 分布式 grep。在文本流中执行模式识别的 grep 操作，通过广泛的文件集来运行。MapReduce 用于提供这一操作的并行和更快的执行。在这一情况下，输入文件是一个简单的文本文件，每一次识别出给定的模式时，map 函数发射一条线到输出。reduce 任务将 map 任务发出的所有线汇聚到一个单一文件。
- URL 访问频率计数。MapReduce 用于分配 Web 服务器日志分析的执行。在这一情况下，map 函数需要输入一个 Web 服务器的日志，并对于日志中记录的每一页访问发射一个键 – 值对 <URL，1> 到输出文件。reduce 函数通过相应的 URL 汇聚所有线，由此得出单一访问，并输出 <URL，total-count> 对。

270

- 反向网络链接图。反向网络链接图跟踪所有可能导致一个给定链接的网页。在这一情况下输入文件是简单的由 map 任务扫描的 HTML 页面，它为网页源发现的每个链接发射 <target，source> 对。reduce 任务将校对所有的对，如果这些对有相同目标就成为 <target，list（source）> 对。最后的结果给出一个或更多包含这些映射的文件。
- 每个主机的词向量。词向量重述最重要的文字，它们出现在一组 <word，frequency> 列表形式的文档中，其中出现的单词数量是其重要性的度量。MapReduce 用于提供一组文档的源和其相应的词向量之间的映射，这个源是作为文档 URL 的主机部分被获得的。在这种情况下，map 任务为每一个检索的文本文档创建一个 <host，term-vector> 对，并且 reduce 任务汇聚了与从相同主机检索的文档相对应的词向量。
- 反向索引。反向索引包含文档中文字表示的信息。相比于直接扫描文件，这些信息

在允许快速全文搜索时是有用的。在这一情况下，map 任务需要输入一个文件，并为每一个文档发射一个 <word,document-id> 集合。reduce 函数汇集出现的相同文字，并产生一个 <word，list（document-id）> 对。

- 分布式排序。在这种情况下，MapReduce 用于对大量记录进行排序操作的并行执行。这一应用多依赖于 MapReduce 的运行时特性，即排序和创建部分中间文件，而不是由 map 和 reduce 任务执行操作。事实上，这些是非常简单的：map 任务从一个记录中抽取键并为每一个记录分配一个 <key，record> 对，reduce 任务将简单地复制所有对。实际的排序处理由 MapReduce 运行时完成，它会通过键对记录排序以分配键 – 值对。

报道的例子大多关注基于文本的处理。通过一些改进，MapReduce 也可以用来解决更广泛的问题。一个有趣的例子是它在机器学习领域中的应用程序 [97]，其中支持向量机（SVM）、线性回归（LR）、朴素贝叶斯（NB）和神经网络（NN）等统计算法以 map 和 reduce 函数的形式表示。其他有趣的应用可以在计算密集型应用程序领域中发现，比如高精确度 Pi 的计算。据报道雅虎 Hadoop 集群已经被用来计算 Pi 的 $10^{15}+1$ 位[⊖]。Hadoop 是 MapReduce 平台的一个开源实现。

通常，任何能以两个主要阶段形式表示的计算都可以用 MapReduce 计算形式表达。这两个阶段是：

- 分析。这一阶段直接对数据输入文件进行操作，并与 map 任务执行的操作保持一致。此外，这一阶段的计算期望是高度并行的，因为 map 任务没有执行任何顺序或排序。 271
- 聚集。这一阶段对中间结果进行操作，特点为聚集、总结和前述前一阶段获得的数据并以最终形式表示这些数据。这是由 reduce 函数完成的任务。

这一模型的改进大多关注适当的键的识别，在原始问题没有这样的模型时创建合理的键，还有发现在 map 和 reduce 函数之间分割计算的方式。此外，更多的复杂算法可以被分解为多个 MapReduce 程序，这里一个程序的输出构成下一个程序的输入。

MapReduce 提出的抽象提供给开发者一个很小的接口，此接口更关注算法的实现而不是执行它的基础设施。这是一个非常有效的方法，但同时需要大量的常见任务，这些常见任务集中于分布式应用运行时的管理，允许用户只通过指定配置参数来控制应用程序的行为。这些任务在分布式基础设施上管理数据传输，并且安排 map 和 reduce 任务。图 8-6 根据谷歌提出的实现方案 [55] 提供了更完整的 MapReduce 描述。

如图 8-6 所示，用户通过使用客户库提交 MapReduce 工作的执行，这些客户库负责提交输入数据文件，记录 map 和 reduce 函数，一旦完成工作便向用户返还控制。一般的分布式基础设施配有工作规划能力，而且分布式存储可用于运行 MapReduce 应用程序。两个不同种类的进程在分布式基础设施上运行：主进程和工作进程。

主进程负责控制 map 和 reduce 任务的执行、分割以及重新组织由 map 任务产生的中间输出，以便满足 reduce 任务。工作进程用于保持 map 和 reduce 任务的执行，并用输入和输出文件提供用于连接 map 和 reduce 任务的基本 I/O 设施。在 MapReduce 计算中，输入文件最初被分割（通常 16 ～ 64MB）并存储在分布式文件系统中。主进程产生 map 任务并通过

⊖ 详细的计算方法见 Yahoo 开发者网络博客，网址：http://developer.yahoo.com/blogs/hadoop/posts/2009/05/hadoop_computes_the_10151st_bit/。

均衡负载分配输入块。

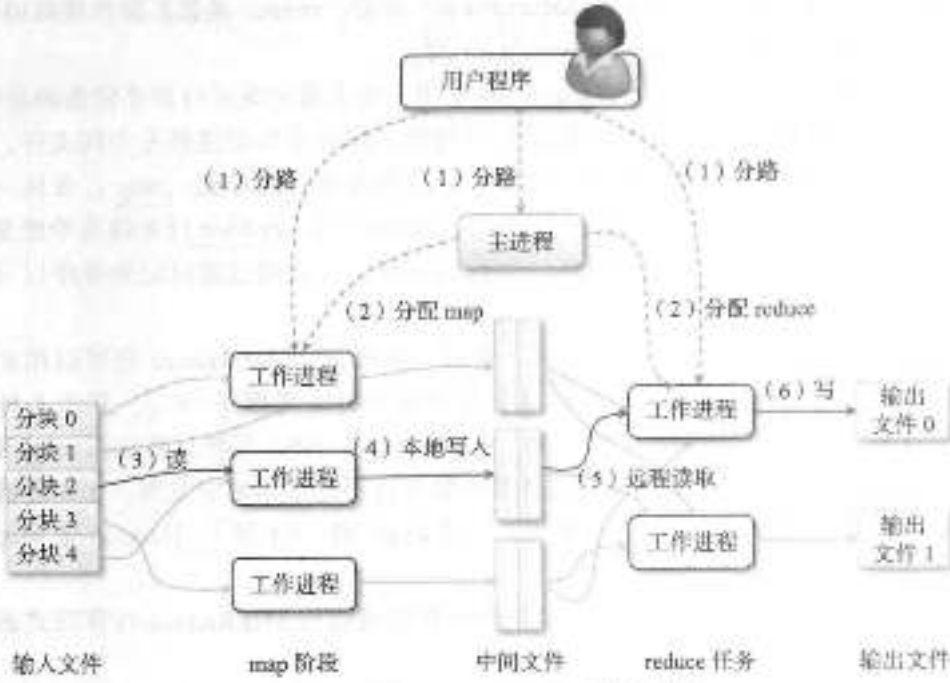


图 8-6　谷歌 MapReduce 基础设施概览

工作进程有输入和输出缓冲区，缓冲区用于优化 map 和 reduce 任务的性能。特别地，map 任务的输出缓冲区会定期转存到磁盘以创建中间文件。中间文件通过用户定义的函数被分割，以均衡 map 任务的输出。然后这些对的位置被通知给主进程，主进程把这一信息发送给 reduce 任务，为了从 map 任务的本地存储中读取，reduce 任务可以通过远程过程调用收集需要的输入。键的范围进而被排序，并且相同的键被组织在一起。最后，执行 reduce 任务以产生最终输出，输出被存储在全局文件系统。这一进程是完全自动的，用户可通过参数配置来控制它，这些参数允许指定（在 map 和 reduce 函数之间）map 任务数量，分割成最终输出的分割数量，以及中间键范围的分割函数。

除了精心安排 map 和 reduce 任务按照先前描述的方式执行，MapReduce 运行时还通过
272 提供一个容错的基础设施来保证应用程序的可靠执行。主进程和工作进程的故障处理方式同使中间输出不可访问的机器故障的处理方式一样，即在其他地方重新安排 map 任务。由于有效的 map 任务的中间输出变得无法访问，所以这也是一种用于找出机器故障的技术。主进程故障反而使用检查点来寻找，检查点技术允许以最小的数据和计算的丢失来重启 MapReduce 工作。

2. MapReduce 的变形和扩展

MapReduce 是一个简化的处理海量数据的模型，并且施加了组织分布式算法以运行在 MapReduce 基础设施的约束。虽然这一模型可以被运用在几个不同的问题场景中，但它仍然有局限性，大多是因为处理数据的抽象概念非常简单，对于复杂问题，只用 map 和 reduce 形式表示可能需要付出相当大的努力。因此，人们提出了一系列原始 MapReduce 模
273 型的扩展和变形。它们旨在扩展 MapReduce 应用程序的空间，并提供给开发者一个更简单的接口以便设计分布式算法。本节简要介绍类似 MapReduce 构架的集合，并讨论它们与原

始 MapReduce 模型的不同。

Hadoop。Apache 的 Hadoop[83] 是一个用于可靠和可扩展分布式计算的软件工程的集合。总体来说，整个集合是一个由类似 GFS 分布式文件系统支持的 MapReduce 架构的开源实现。该倡议主要包括两个项目：Hadoop 分布式文件系统（HDFS）和 Hadoop MapReduce。前者是谷歌文件系统 [54] 的实现，后者提供与谷歌 MapReduce 相同的特性与抽象。Hadoop 最初由雅虎开发和支持，现在已成为最成熟和最大的数据云应用，并有一个支持它的开发者和用户组成的非常强大的团体。雅虎现在运行着世界上最大的 Hadoop 集群，由 40000 台机器和 300000 多个中心组成，全世界的学术机构都可使用这一集群。除了 Hadoop 的核心工程，其他相关工程的集群也为分布式计算提供服务。

Pig。Pig[⊖] 是一个分析大量数据集的平台。作为一个 Apache 开发项目，Pig 由表示数据分析程序的高级语言以及用于评估这些程序的基础设施组成。Pig 的基础设施层包括一个用于产生 MapReduce 工作序列的高级语言的编译器，它可以运行在如同 Hadoop 的分布式基础设施的顶层。开发者可以用一种叫作 Pig Latin 的文本语言编写数据分析程序，这需要一个类似 SQL 的接口，其特点是以表现性为主、较少的编程工作，以及一个熟悉的 MapReduce 接口。

Hive。Hive[⊖] 是 Apache 的另一个倡议，它在 Hadoop MapReduce 的顶层提供一个数据仓库基础设施。它提供了用于简单数据汇总的工具，ad hoc 查询以及存储在 Hadoop MapReduce 文件的大量数据集的分析。虽然这一构架提供与传统数据仓库相同的功能，却没有展现出相同的性能，尤其是查询延迟，由于这一原因，Hive 不是一个解决网上交易处理的有效方法。Hive 的主要优势在于向外扩展的能力，原因是它基于 Hadoop 构架，并且在已有 Hadoop 系统运行的环境下具有提供数据仓库基础设施的能力。

Map-Reduce-Merge。Map-Reduce-Merge[98] 是 MapReduce 模型的扩展，将第三阶段（Merge 阶段）引入标准 MapReduce 渠道，允许对已分割的数据进行有效地融合并用 map 和 reduce 模块分类（或哈希排序）。Map-Reduce-Merge 框架简化了异构的相关数据集的管理，并提供一个能够表示普通关系代数运算符和一些连接算法的抽象。

Twister。Twister[99] 是 MapReduce 模型的扩展，允许创建 MapReduce 工作的迭代执行。相对于正常的 MapReduce 渠道，Twister 提出的这一模型加入了以下扩展：

- 配置 map。
- 配置 reduce。
- 条件为真时，执行以下操作：
  - 运行 MapReduce。
  - 对结果应用连接操作。
  - 更新条件。
- 关闭。

除了迭代 MapReduce 计算，Twister 还提供附加功能。比如查询静态内存数据的 map 和 reduce 任务的能力；引入名为 combine 的附加阶段，该阶段在 MapReduce 工作的结尾运行，将输出聚集在一起；以及用于数据管理的其他工具。

274

⊖ http://pig.apache.org/。

⊖ http://hive.apache.org/。

3. MapReduce 的替代品

MapReduce 和其他抽象为处理大量数据集提供支持并执行数据密集型工作负载。这些替代品从不同的程度展现出一些与 MapReduce 方法的相似性。

Sphere。 Sphere[84] 是利用 Sector 分布式文件系统（SDFS）的分布式处理引擎。Sphere 不是 MapReduce 的改版，它实现了流处理模型（单一程序，多个数据）并允许开发者以用户定义函数（UDF）的形式表示计算。UDF 是针对分布式基础设施运行的。UDF 的一个特定组合允许 Sphere 表示 MapReduce 计算。Sphere 充分利用 Sector 分布式文件系统，并建立在 Sector 用于数据访问的 API 的顶层。UDF 根据读写流的程序来表示。流是一种数据结构，此数据结构被提供在 SDFS 中映射到一或多个文件数据段的集合的访问。UDF 的集体执行通过 Sphere 处理引擎（SPE）的分布式执行来实现，SPE 以一个给定的流段被分配。这一执行模型是一个客户端控制的主 – 从模型，Sphere 客户端向主节点发送处理请求，主节点返回可用的从节点列表，客户端选择从节点，在从节点上将执行 Sphere 进程并安排整个分布式执行。

All-Pairs。 All-Pairs[100] 是一个抽象和运行时环境，以优化数据密集型工作负载的执行。它提供一个简单的抽象，即 All-Pairs 函数，这在很多科学计算领域是普遍存在的：

All-pairs（A:set,B:set,F:function）→ M：matrix

能用这一模型表示的问题的例子出现在生物统计学领域，这里相似性矩阵由几个包含主题图片的图像比较结果构成。另一个例子是数据挖掘中的几个应用程序和算法。由 All-pairs 函数表示的模型可以用以下算法容易地解决：

```
For each $i in A
  For each $j in B
    Submit job F $i $j
```

这一实现非常不成熟，并且在一般情况下会产生较差的性能。此外，数据分割、发送延迟、可用计算节点的数量以及故障几率等其他问题并没有进行专门处理。All-pairs 模型试图通过引入一个问题本质的规范和一个引擎解决这些问题，该引擎根据这一规范在传统集群或网格设施上优化任务的分配。分布式应用的执行由引擎控制，经过四个发展阶段：模拟系统、分配数据、调度批处理作业和清理系统。这一模型的前两个阶段更受关注，在这两阶段中系统的性能模型被建立并且数据被随机分配，以便创建最佳的任务数量来安排给每个节点并优化基础设施的利用。

275

DryadLINQ。 Dryad[101] 是一个微软研究项目，它为并行写入和从小的集群到大的数据中心的分布式程序研究编程模型。Dryad 的目标是为自动并行应用程序的执行提供基础设施，而不要求开发者知道分布式和并行编程。

在 Dryad 中，开发者可将分布式应用程序表示成一组由信道方式连接的顺序程序。更精确的是，一个 Dryad 计算可以用一个有向无环图的形式表示，其中节点是顺序执行的程序，顶点表示连接这类程序的通道。由于这一结构，Dryad 被认为是超级 MapReduce 模型，因为它的一般应用模型也允许表示代表 MapReduce 计算的图。Dryad 的一个有趣的特征是支持图形（在一定程度上）和分区的动态修改的能力，如果可能，还支持图形到阶段的执行。这一基础设施用于服务并行编程的不同应用和工具。在它们中间，DryadLINQ[102] 是一个从语言综合查询（LINQ）扩充到 C#[103] 的产生 Dryad 计算的编程环境。由此产生的

框架提供了一个完全与 .NET 框架融合的解决方法，并能够表示几个分布式计算模型，包括 MapReduce。

8.3　Aneka MapReduce 编程

Aneka 遵循由谷歌引进和 Hadoop 实现的参考模型，提供了一个 MapReduce 抽象的实现，支持 MapReduce 作为一个可用于开发分布式应用的程序模型。

8.3.1　MapReduce 编程模型简介

MapReduce 编程模型为开发 Aneka 顶层的 MapReduce 应用程序定义了抽象和运行时支持。图 8-7 提供了 Aneka MapReduce 基础设施的概览。谷歌 MapReduce 或 Hadoop 中的作业与 Aneka MapReduce 应用程序的执行相协调。应用通过识别使用的 map 和 reduce 函数的组件实例被具体化。这些函数表示成 Mapper 和 Reducer 类的形式，两个类是从 Aneka MapReduce 的 API 扩展来的。运行时支持由三个主要元素组成：

- MapReduce 计划服务，扮演谷歌和 Hadoop 实现中主进程的角色。
- MapReduce 执行服务，扮演谷歌和 Hadoop 实现中工作进程的角色。
- 一个用于移动数据文件的特定分布式文件系统。

276

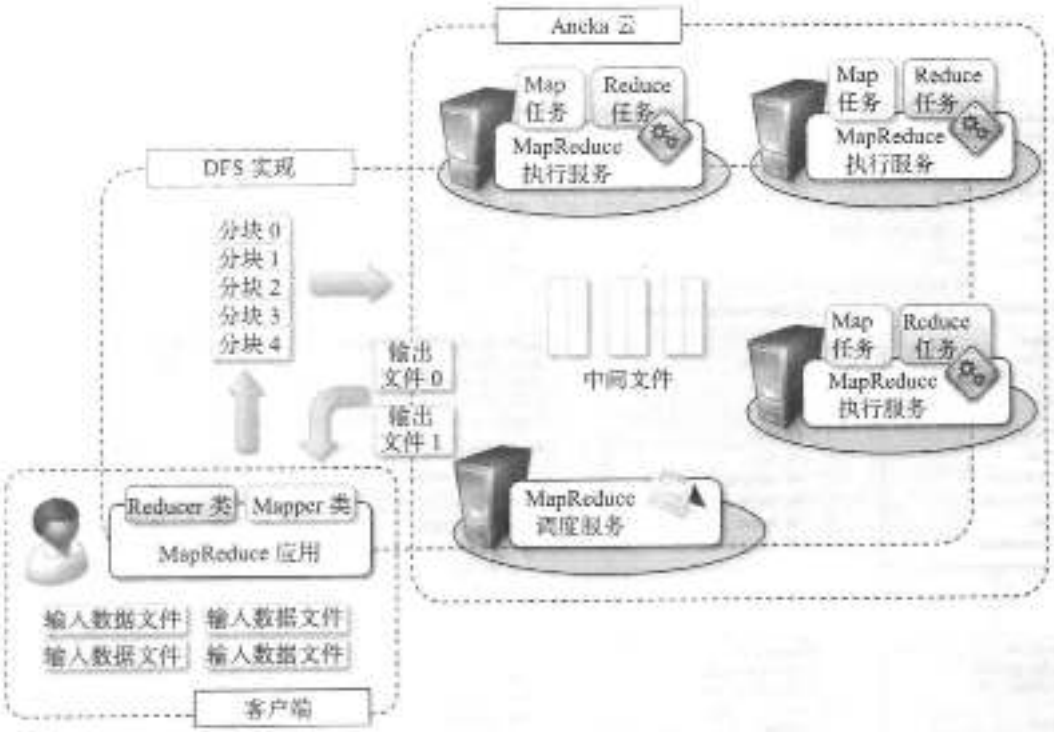


图 8-7　Aneka MapReduce 基础设施

客户端组件，即 MapReduce 应用程序，用于提交 MapReduce 作业的执行，上载并监视数据文件。数据文件的管理是透明的：本地数据文件自动加载到 Aneka，有请求时，输出文件自动下载到客户机。

以下将介绍这些主要的组件并描述它们是怎样协作执行 MapReduce 作业的。

1. 编程抽象

Aneka 在分布式应用的背景中执行任何一段用户代码。这一方法甚至在 MapReduce 编程模型中也得到保持，此编程模型中 MapReduce 作业（在谷歌 MapReduce 和 Hadoop 使用）的思想与 Aneka 应用的思想之间有自然映射。一旦用户定义了 map 和 reduce 函数，这一任务创建就不是用户的而是基础设施的责任了，这一点与其他编程模型不同。因此，Aneka MapReduce 的 API 为开发者提供开发 Mapper 和 Reducer 类型的基本分类并使用特定类型的
277 应用分类，即 MapReduce 应用程序，以更好地满足这一编程模型的需求。

图 8-8 提供了定义 MapReduce 编程模型的客户端组件的概览。有三个类是应用开发感兴趣的：Mapper<K，V>，Reducer<K，V> 和 MapReduceApplication<M，R>。其他类则用于内部实现这一模型需要的所有函数，并提供简单的接口，这些接口需要最少的代码去实现 map 和 reduce 函数以及控制作业提交。Mapper<K，V> 和 Reducer<K，V> 是应用设计和实现的起点。模板特化用于记录这两个函数操作的键和值。泛型以对象操作的形式为 map 和 reduce 方法提供了一个更自然的方法，并通过去除计算必要性和其他类型的检查操作以简化编程。MapReduce 作业的提交和执行通过 MapReduceApplication<M，R> 类来完成，此类

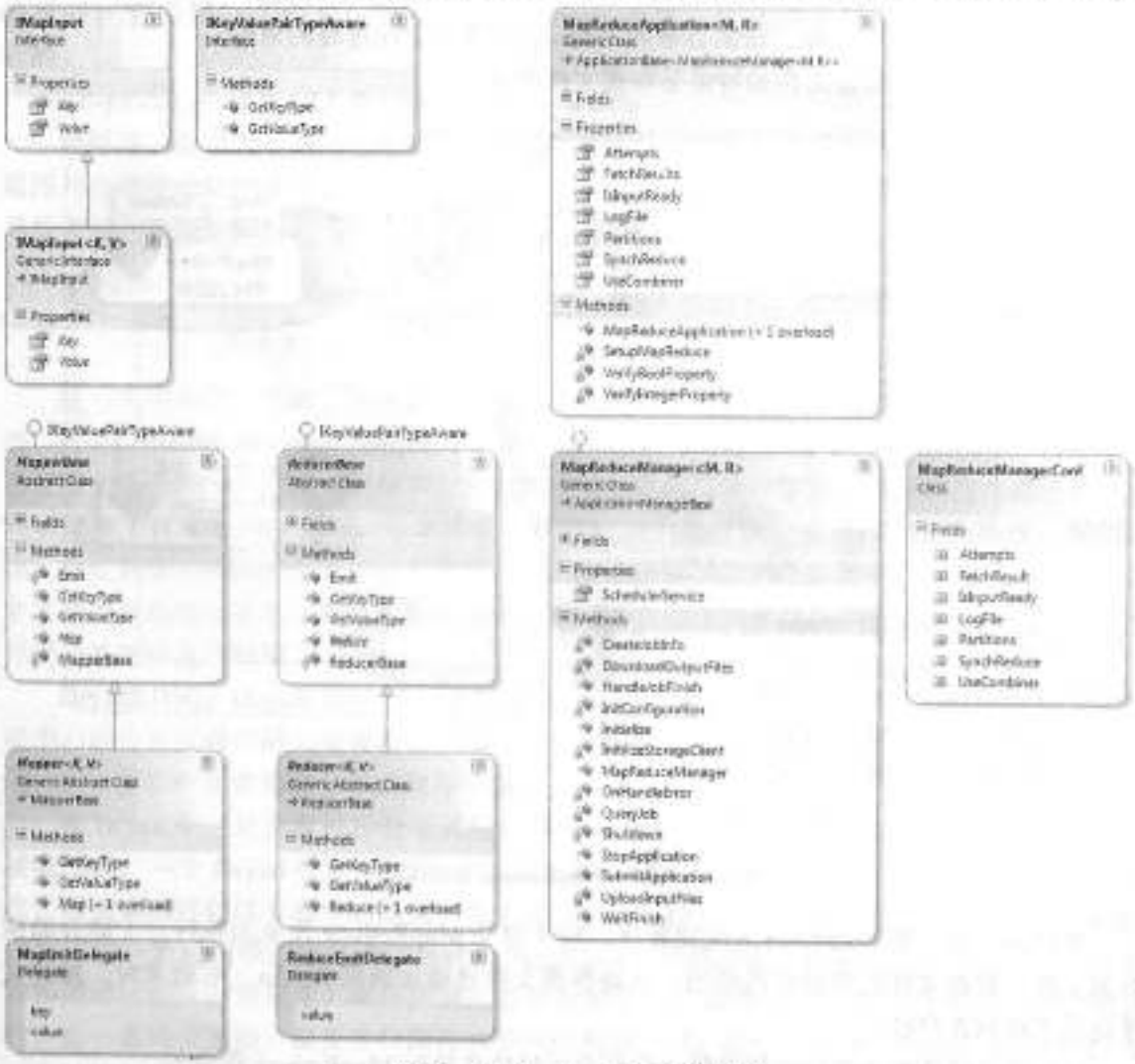


图 8-8　MapReduce 抽象对象模型

提供了支持 MapReduce 编程模型的 Aneka 云的接口。这个类型揭示了两个一般的类：M 和 R。这两个占位符将用于识别应用的特定类型的 Mapper<K，V> 和 Reducer<K，V> 类。

程序 8-1 展示了 Mapper<K，V> 类的细节和开发者应该知道的实现 map 函数的相关类型的定义。为实现一个特定的 mapper，有必要继承这个类并为 K 键和 V 值提供实际的类。map 操作通过重写 void Map（IMapInput<K，V>input）抽象方法得以实现，而其他方法则在框架内部使用。IMapInput<K，V> 提供对于完成 map 操作的输入键 – 值对的访问。

程序 8-1　map 函数 API

```
using Aneka.MapReduce.Internal;

namespace Aneka.MapReduce
{
    /// <summary>
    /// Interface IMapInput<K,V>. Extends IMapInput and provides a strongly
    /// typed version of the extended interface.
    /// </summary>
    public interface IMapInput<K,V>: IMapInput
    {
        /// <summary>
        /// Property <i>Key</i> returns the key of key/value pair.
        /// </summary>
        K Key { get; }
        /// <summary>
        /// Property <i>Value</i> returns the value of key/value pair.
        /// </summary>
        V Value { get; }
    }

    /// <summary>
    /// Delegate MapEmitDelegate. Defines the signature of a method
    /// that is used to doEmit intermediate results generated by the mapper.
    /// </summary>
    /// <param name='key'>The <i>key</i> of the <i>key-value</i> pair.</param>
    /// <param name='value'>The <i>value</i> of the <i>key-value</i> pair.</param>
    public delegate void MapEmitDelegate(object key, object value);

    /// <summary>
    /// Class Mapper. Extends MapperBase and provides a reference implementation that
    /// can be further extended in order to define the specific mapper for a given
    /// application. The definition of a specific mapper class only implies the
    /// implementation of the Mapper<K,V>.Map(IMapInput<K,V>) method.
    /// </summary>
    public abstract class Mapper<K,V> : MapperBase
    {
        /// <summary>
        /// Emits the intermediate result source by using doEmit.
        /// </summary>
        /// <param name='source'>An instance implementing IMapInput containing the
        /// <i>key-value</i> pair representing the intermediate result.</param>
        /// <param name='doEmit'>A MapEmitDelegate instance that is used to write to the
        /// output stream the information about the output of the Map operation.</param>
        public void Map(IMapInput input, MapEmitDelegate emit) { ... }
        /// <summary>
        /// Gets the type of the <i>key</i> component of a <i>key-value</i> pair.
        /// </summary>
        /// <returns>A Type instance containing the metadata about the type of the
        /// <i>key</i>.</returns>
        public override Type GetKeyType(){ return typeof(K); }
```

```
        /// <summary>
        /// Gets the type of the <i>value</i> component of a <i>key-value</i> pair.
        /// </summary>
        /// <returns>A Type instance containing the metadata about the type of the
        /// <i>value</i>.</returns>
        public override Type GetValueType() { return typeof(V); }

        #region Template Methods
        /// <summary>
        /// Function Map is overrided by users to define a map function.
        /// </summary>
        /// <param name="source">The source of Map function is IMapInput, which contains
        /// a key/value pair.</param>
        protected abstract void Map(IMapInput<K, V> input);
        #endregion
    }
}
```

程序 8-2 展示了用于字计数器样本的 Mapper<K，V> 组件的实现。该样本用于在一组大型文本文件中统计字的频率。文本文件被分割成行，每一行将变成一个键 - 值对的值组件，然而该键将通过在行开始的文件中的偏移量的方式来表示。因此，mapper 通过使用长整型作为键类型以及字符串作为值而被详细说明。为统计字的频率，map 函数将为行中包含的每个字发射一个新的键 - 值对，方法是将字作为键、数字 1 作为值。如果字在行中出现两次，这一实现将为相同的字发射两个对。在适当的时候对这些情况进行求和将是 reducer 的责任。

程序 8-2　简单的 Mapper<K，V> 实现

```
using Aneka.MapReduce;

namespace Aneka.MapReduce.Examples.WordCounter
{
    /// <summary>
    /// Class WordCounterMapper. Extends Mapper<K,V> and provides an
    /// implementation of the map function for the Word Counter sample. This mapper
    /// emits a key-value pair (word,1) for each word encountered in the input line.
    /// </summary>
    public class WordCounterMapper: Mapper<long,string>
    {
        /// <summary>
        /// Reads the source and splits into words. For each of the words found
        /// emits the word as a key with a value of 1.
        /// </summary>
        /// <param name="source">map source</param>
        protected override void Map(IMapInput<long,string> input)
        {
            // we don't care about the key, because we are only interested in
            // counting the word of each line.
            string value = input.Value;

            string[] words = value.Split(" \t\n\r\f\"\'!-/()[]<>:{},#.".ToCharArray(),
                                         StringSplitOptions.RemoveEmptyEntries);

            // we emit each word without checking for repetitions. The word becomes
            // the key and the value is set to 1, the reduce operation will take care
            // of merging occurrences of the same word and summing them.
            foreach(string word in words)
            {
                this.Emit(word, 1);
            }
        }
    }
}
```

程序 8-3 展示了 Reducer<K，V> 类的定义。reducer 的具体实现需要制定泛型类并重写 Reduce（IReduceInputEnumerator<V>input）抽象方法。由于 reduce 操作应用于映射到相同键的值的集合，IReduceInputEnumerator<V> 允许开发者遍历这样的集合。程序 8-4 展示了怎样为字 - 计数器实例实现 reducer 函数。

程序 8-3　reduce 函数 API

```
using Ancka.MapReduce.Internal;
namespace Ancka.MapReduce
{
    /// <summary>
    /// Delegate ReduceEmitDelegate. Defines the signature of a method
    /// that is used to emit aggregated value of a collection of values matching the
    /// same key and that is generated by a reducer.
    /// </summary>
    /// <param name="value">The <i>value</i> of the <i>key-value</i> pair.</param>
    public delegate void ReduceEmitDelegate(object value);

    /// <summary>
    /// Class <i>Reducer</i>. Extends the ReducerBase class and provides an

    /// implementation of the common operations that are expected from a <i>Reducer</i>.
    /// In order to define reducer for specific applications developers have to extend
    /// implementation of the Reduce(IReduceInputEnumerator<V>) method that reduces a
    /// this class and provide an collection of <i>key-value</i> pairs as described by
    /// the <i>map-reduce</i> model.
    /// </summary>
    public abstract class Reducer<K,V> : ReducerBase
    {
        /// <summary>
        /// Performs the <i>reduce</i> phase of the <i>map-reduce</i> model.
        /// </summary>
        /// <param name="source">An instance of IReduceInputEnumerator allowing to
        /// iterate over the collection of values that have the same key and will be
        /// aggregated.</param>
        /// <param name="emit">An instance of the ReduceEmitDelegate that is used to
        /// write to the output stream the aggregated value.</param>
        public void Reduce(IReduceInputEnumerator input, ReduceEmitDelegate emit) { _ }
        /// <summary>
        /// Gets the type of the <i>key</i> component of a <i>key-value</i> pair.
        /// </summary>
        /// <returns>A Type instance containing the metadata about the type of the
        /// <i>key</i>.</returns>
        public override Type GetKeyType(){return typeof(K);}
        /// <summary>
        /// Gets the type of the <i>value</i> component of a <i>key-value</i> pair.
        /// </summary>
        /// <returns>A Type instance containing the metadata about the type of the
        /// <i>value</i>.</returns>
        public override Type GetValueType(){return typeof(V);}

        #region Template Methods
        /// <summary>
        /// Reduces the collection of values that are exposed by
        /// <paramref name="source"/> into a single value. This method implements the
        /// <i>aggregation</i> phase of the <i>map-reduce</i> model, where multiple
        /// values matching the same key are composed together to generate a single
        /// value.
        /// </summary>
        /// <param name="source">An IReduceInputEnumerator<V> instancethat allows to
        /// iterate over all the values associated with same key.</param>
        protected abstract void Reduce(IReduceInputEnumerator<V> input);
        #endregion
    }
}
```

程序 8-4　简单的 Reduce<K，V> 实现

```
using Aneka.MapReduce;

namespace Aneka.MapReduce.Examples.WordCounter
{
    /// <summary>
    /// Class <b><i>WordCounterReducer</i></b>. Reducer implementation for the Word
    /// Counter application. The Reduce method iterates all over values of the
    /// enumerator and sums the values before emitting the sum to the output file.
    /// </summary>
    public class WordCounterReducer: Reducer<string,int>
    {
        /// <summary>
        /// Iterates all over the values of the enumerator and sums up
        /// all the values before emitting the sum to the output file.
        /// </summary>
        /// <param name="source">reduce source</param>
        protected override void Reduce(ReduceInputEnumerator<int> input)
        {
            int sum = 0;

            while(input.MoveNext())
            {
                int value = input.Current;
                sum += value;
            }
            this.Emit(sum);
        }
    }
}
```

在这种情况下，Reducer<K，V> 类被指定使用字符串作为键类型，以整型作为值。reducer 简单地遍历所有的可通过枚举器访问的值并将它们求和。一旦遍历完成，所求和将被转储到文件。

注意，用于指定 mapper 的类和用于指定 reducer 的类之间有一个重要的连接。reducer 使用的键和值类定义了 mapper 发出的键 - 值对类。由于 mapper 产生一个键 - 值对 (string，int)，因此 reducer 是 reducer< string，int> 类。

Mapper<K，V> 和 Reducer<K，V> 类为定义由 MapReduce 作业执行的计算提供能力。为提交、执行和监视进程，Aneka 提供了 MapReduceApplication<M，R> 类。同其他编程
278 模型一样，这一类代表了 Aneka 顶部的分布式应用的局部视图。因为 MapReduce 模型的简易，该类提供的能力有限，大多关注开始 MapReduce 作业并等待它的完成。程序 8-5 展示了 MapReduceApplication<M，R> 的接口。

该类的接口仅展示了 MapReduce 的特定设置，然而控制逻辑封装在
279 ApplicationBase<M> 中。通过这个类可设置 MapReduce 当前执行的行为。可以控制的参数如下：

- Partitions。该属性存储了一个整型数字，它包含最终结果被分割成的部分的数量。这个值也决定了将由运行时设施创建的 reducer 任务的数量。默认值为 10。
- Attempts。该属性包含声明失败之前运行时将重试执行一个任务的次数。默认值为 3。
- UseCombiner。该属性存储一个布尔值，指示 MapReduce 运行时是否应该将一个

combiner 阶段添加到 map 任务执行，以便减少传到 reduce 任务的中间文件的数量。默认值为 true。

- SynchReduce。该属性存储一个布尔值，指示是否同步 reducer。默认值为 ture，现在还没有用于决定 MapReduce 的行为。
- IsInputReady。这是一个布尔属性，指示输入文件是否已被存储在分布式文件系统，或在作业可被执行前必须由客户端管理器上载。默认值为 false。
- FetchResults。这是一个布尔属性，指示客户端管理器是否需要将由作业执行产生的结果文件下载到本地计算机。默认值为 true。
- LogFile。这一属性包含一个定义日志文件名称的字符串，该文件用于存储在 MapReduce 作业执行过程中记录的性能统计信息。默认值为 mapreduce.log。

程序 8-5　MapReduceApplication<M，R>

```
using Alea.MapReduce.Internal;

namespace Alea.MapReduce
{
    /// <summary>
    /// Class <b><i>MapReduceApplication</i></b>. Defines a distributed application
    /// based on the MapReduce model. It extends the ApplicationBase<M> and specializes
    /// it with the MapReduceManager<M,R> application manager. A MapReduceApplication is
    /// a generic type that is parameterized with a specific type of MapperBase and a
    /// specific type of ReducerBase. It controls the execution of the application and
    /// it is in charge of collecting the results or resubmitting the failed tasks.
    /// </summary>
    /// <typeparam name="M">Placeholder for the mapper type.</typeparam>
    /// <typeparam name="R">Placeholder for the reducer type.</typeparam>
    public class MapReduceApplication<M, R> : ApplicationBase<MapReduceManager<M, R>>
        where M: MapReduce.Internal.MapperBase
        where R: MapReduce.Internal.ReducerBase
    {
        /// <summary>
        /// Default value for the Attempts property.
        /// </summary>
        public const int DefaultRetry = 3;
        /// <summary>
        /// Default value for the Partitions property.
        /// </summary>
        public const int DefaultPartitions = 10;
        /// <summary>
        /// Default value for the LogFile property.
        /// </summary>
        public const string DefaultLogFile = "mapreduce.log";

        /// <summary>
        /// List containing the result files identifiers.
        /// </summary>
        private List<string> resultFiles = new List<string>();
        /// <summary>
        /// Property group containing the settings for the MapReduce application.
        /// </summary>
        private PropertyGroup mapReduceSetup;

        /// <summary>
        /// Gets, sets an integer representing the number of partions for the key space.
        /// </summary>
        public int Partitions { get { ... } set { ... } }
        /// <summary>
        /// Gets, sets an boolean value indicating in whether to combine the result
        /// after the map phase in order to decrease the number of reducers used in the
        /// reduce phase.
```

```
        /// </summary>
        public bool UseCombiner { get { ... } set { ... } }
        /// <summary>
        /// Gets, sets an boolean indicating whether to synchronize the reduce phase.
        /// </summary>
        public bool SynchReduce { get { ... } set { ... } }

        /// <summary>
        /// Gets or sets a boolean indicating whether the source files required by the
        /// required by the application is already uploaded in the storage or not.
        /// </summary>
        public bool IsInputReady { get { ... } set { ... } }
        /// <summary>
        /// Gets, sets the number of attempts that to run failed tasks.
        /// </summary>
        public int Attempts { get { ... } set { ... } }
        /// <summary>
        /// Gets or sets a string value containing the path for the log file.
        /// </summary>
        public string LogFile { get { ... } set { ... } }
        /// <summary>
        /// Gets or sets a boolean indicating whether application should download the
        /// result files on the local client machine at the end of the execution or not.
        /// </summary>
        public bool FetchResults { get { ... } set { ... } }

        /// <summary>
        /// Creates a MapReduceApplication<M,R> instance and configures it with
        /// the given configuration.
        /// </summary>
        /// <param name="configuration">A Configuration instance containing the
        /// information that customizes the execution of the application.</param>
        public MapReduceApplication(Configuration configuration) :
                                    base("MapReduceApplication", configuration) { ... }
        /// <summary>
        /// Creates MapReduceApplication<M,R> instance and configures it with
        /// the given configuration.
        /// </summary>
        /// <param name="displayName">A string containing the friendly name of the
        /// application.</param>
        /// <param name="configuration">A Configuration instance containing the
        /// information that customizes the execution of the application.</param>
        public MapReduceApplication(string displayName, Configuration configuration) :
                                    base(displayName, configuration) { ... }

        // here follows the private implementation.
    }
}
```

管理 MapReduc 作业执行的核心控制逻辑驻留在 MapReduceApplicationManager<M，R>中，该类与 MapReduce 运行时相互作用。开发者可以通过使用 ApplicationBase<M>类公开的方法和属性来控制应用程序。程序 8-6 展示了执行 MapReduce 作业时与该类有关的方法的集合。

除了关心所有应用程序的构造函数和常用属性，程序 8-6 中以粗体显示的两种方法最常用于执行 MapReduce 作业。这是 InvokeAndWait 方法的两个不同的重载：第一个简单地开始 MapReduce 作业的执行并在完成后返回；第二个在任务的结尾执行应用于客户端的回调。InvokeAndWait 的使用正在受到限制，因此，不可能通过在同一个线程调用 StopExecution 来停止应用程序。如果有必要实现对 MapReduce 作业的更复杂的管理，可能会用到 SubmitExecution 方法，此方法提交应用的执行，而不需要等待它的完成就可返回。

程序 8-6　ApplicationBase<M>

```
Namespace Aneka.Entity
{
    /// <summary>
    /// Class <b><i>ApplicationBase<M></i></b>, defines the base class for the
    /// application instances for all the programming model supported by Aneka.
    /// </summary>
    public class ApplicationBase<M> where M : IApplicationManager, new()
    {
        /// <summary>
        /// Gets the application unique identifier attached to this instance. The
        /// application unique identifier is the textual representation of a System.Guid
        /// instance, therefore is a globally unique identifier. This identifier is
        /// automatically created when a new instance of an application is created.
        /// </summary>
        public string Id { get { ... } }
        /// <summary>
        /// Gets the unique home directory for the AnekaApplication<W,M>.
        /// </summary>
        public string Home { get { ... } }
        /// <summary>
        /// Gets the current state of the application.
        /// </summary>
        public ApplicationState State{get{ ... }}
        /// <summary>
        /// Gets a boolean value indicating whether the application is terminated.
        /// </summary>
        public bool Finished { get { ... } }

        /// <summary>
        /// Gets the underlying IApplicationManager that coordinates the execution of the
        /// application instances on the client side.
        /// </summary>
        public M ApplicationManager { get { ... } }

        /// <summary>
        /// Gets, sets the application display name. This is a friendly name which is
            /// to identify an application by means of a textual and human intelligible
            /// sequence of characters, but it is NOT a unique identifier and so check about
            /// uniqueness of the value of this property is done. For a unique identifier
            /// please check the Id property.
        /// </summary>
        public string DisplayName { get { ... } set { ... } }

        /// <summary>
        /// Occurs when the application instance terminates its execution.
        /// </summary>
        public event EventHandler<ApplicationEventArgs> ApplicationFinished;

        /// <summary>
        /// Creates an application instance with the given settings and sets the
            /// application display name to null.
        /// </summary>
        /// <param name="configuration">Configuration instance specifying the
            /// application settings.</param>
        public ApplicationBase(Configuration configuration): this(null, configuration)
        { ... }
        /// <summary>
        /// Creates an application instance with the given settings and display name. As
            /// a result of the invocation, a new application unique identifier is created
            /// and the underlying application manager is initialized.
        /// </summary>
        /// <param name="configuration">Configuration instance specifying the application
            /// settings.</param>
        /// <param name="displayName">Application friendly name.</param>
        public ApplicationBase(string displayName, Configuration configuration){ ... }
        /// <summary>
        /// Starts the execution of the application instance on Aneka.
```

```
        /// </summary>
        public void SubmitExecution() { ... }
        /// <summary>
        /// Stops the execution of the entire application instance.
        /// </summary>
        public void StopExecution() { ... }
        /// <summary>
        /// Invoke the application and wait until the application finishes.
        /// </summary>
        public void InvokeAndWait() { this.InvokeAndWait(null); }
        /// <summary>
        /// Invoke the application and wait until the application finishes, then invokes
            /// the given callback.
        /// </summary>
        /// <param name="handler">A pointer to a method that is executed at the end of
        /// the application.</param>
        public void InvokeAndWait(EventHandler<ApplicationEventArgs> handler) { ... }

        /// <summary>
            /// Adds a shared file to the application.
        /// </summary>
        /// <param name= file">A string containing the path to the file to add.</param>
        public virtual void AddSharedFile(string file) { ... }
        /// <summary>
            /// Adds a shared file to the application.
        /// </summary>
        /// <param name="file">A FileData instance containing the information about the
        /// file to add.</param>
        public virtual void AddSharedFile(FileData fileData) { ... }
        /// <summary>
            /// Removes a file from the list of the shared files of the application.
            /// </summary>
           /// <param name="file"> A string containing the path to the file to
        /// remove.</param>
        public virtual void RemoveSharedFile(string filePath) { ... }

            // here come the private implementation.

    }

}
```

提到文件的管理，MapReduce 实现将自动上载 Configuration 中出现的所有文件。工作区目录将忽略由 AddSharedFile 方法附加的文件。

程序 8-7 展示了如何为了运行字 – 计数器实例创建一个 MapReduce 应用，此实例由之前的 WordCounterMapper 和 WordCounterReducer 类来定义。

需要关注的列在 try{…}catch{…}finally{…} 块中以粗体显示。如同程序 8-7 所示，MapReduce 作业的执行仅需要三行代码，这里用户读取配置文件，创建一个 MapReduceApplication<M，R> 实例并进行配置，之后开始执行。所有剩下的代码关注的大多是设置记录和异常处理。

程序 8-7　字 – 计数器作业

```
using System.IO;
using Aneka.Entity;
using Aneka.MapReduce;

namespace Aneka.MapReduce.Examples.WordCounter
{
    /// <summary>
    /// Class <b><i>Program</i></b>. Application driver for the Word Counter sample.
```

```
/// </summary>
public class Program
{
    /// <summary>
    /// Reference to the configuration object.
    /// </summary>
    private static Configuration configuration = null;
/// <summary>
/// Location of the configuration file.
/// </summary>
private static string confPath = "conf.xml";

/// <summary>
/// Processes the arguments given to the application and according
/// to the parameters read runs the application or shows the help.
/// </summary>
/// <param name="args">program arguments</param>
private static void Main(string[] args)
{
    try
    {
        Logger.Start();

        // get the configuration
        configuration = Configuration.GetConfiguration(confPath);

        // configure MapReduceApplication
        MapReduceApplication<WordCountMapper, WordCountReducer> application =
            new MapReduceApplication<WordCountMapper, WordCountReducer>
                ("WordCounter", configuration);
        // invoke and wait for result
        application.InvokeAndWait(new
                                  EventHandler<ApplicationEventArgs>(OnDone));

            // alternatively we can use the following call
            // application.InvokeAndWait();
    }
    catch(Exception ex)
    {
        Usage();
        IOUtil.DumpErrorReport(ex, "Aneka WordCounter Demo - Error Log");
    }
    finally
    {
        Logger.Stop();
    }
}
/// <summary>
/// Hooks the ApplicationFinished events and Process the results
/// if the application has been successful.
/// </summary>
/// <param name="sender">event source</param>
/// <param name="e">event information</param>
private static void OnDone(object sender, ApplicationEventArgs e) { ... }
/// <summary>
/// Displays a simple informative message explaining the usage of the
/// application.
/// </summary>
private static void Usage() { ... }
}
```

2. 运行时支持

283
?
284

MapReduce 作业执行的运行时支持是由能够处理安排和执行 MapReduce 任务服务的组合构成的。这些任务包括 MapReduce 计划服务和 MapReduce 执行服务。这两个服务与该框架现存的服务相结合，以便能够提供持久性、应用记账以及可用于其他编程模型开发的应用程序的功能。

285
?
286

作业和任务计划。作业和任务的计划是由 MapReduce 计划服务负责的，这与谷歌 MapReduce 实现中的主处理过程扮演的角色一致。计划服务的架构被组织成两个主要部分：MapReduce 调度服务和 MapReduce 调度器。前者是调度程序的封装，实现 Aneka 所要求的作为服务应公开一个软件的接口；后者控制作业执行并调度任务。因此，封装服务的主要作用是将来自 Aneka 执行时或者客户应用的信息转化为针对调度程序的呼叫或事件，反之亦然。这两个组件之间的关系如图 8-9 所示。

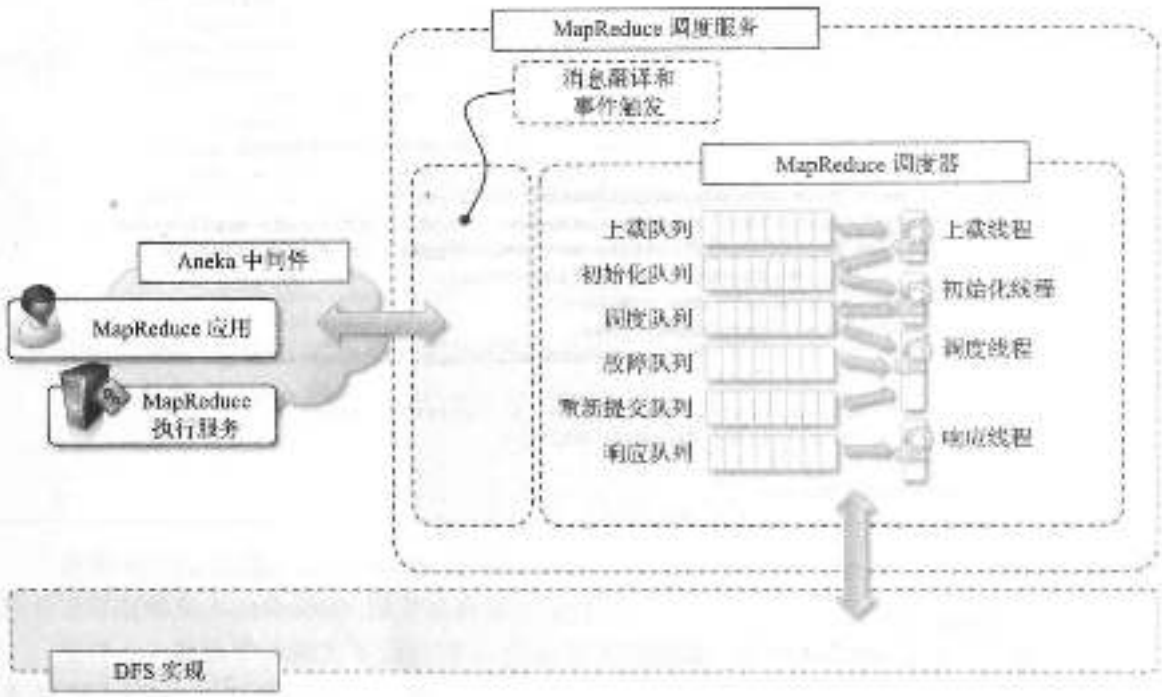


图 8-9　MapReduce 调度服务架构

工作和任务调度的核心功能在 MapReduceScheduler 类中实现。针对一些操作，调度程序管理多个队列，如上传输入文件到分布式文件系统、调度之前进行初始化、调度 map 和 reduce 任务、跟踪不可达的节点、重新提交失败的任务和报告执行统计信息。所有这些操作都是异步执行并通过发生在 Aneka 中间件的事件来触发的。

任务执行。任务的执行是由 MapReduce 执行服务控制的。此组件起到谷歌 MapReduce 实现中工作进程的作用。该服务管理 map 和 reduce 任务的执行，并执行其他操作，如分类和合并中间文件。该服务位于内部组织，见图 8-10。

三个主要组件一起协调执行任务：MapReduce 计划服务，执行管理器和 MapReduce 执行器。MapReduce 计划服务用 Aneka 的中间件与执行管理器连接，通过获取 MapReduce 执行器中任务的特定执行，向调度服务发送有关的执行统计数据，执行管理器负责对正在执行

的任务保持追踪。配置多于一个 MapReduce 执行器的实例是可行的，且在多核节点的情况下是有益的，在这些节点上多个任务可以在同一时间执行。

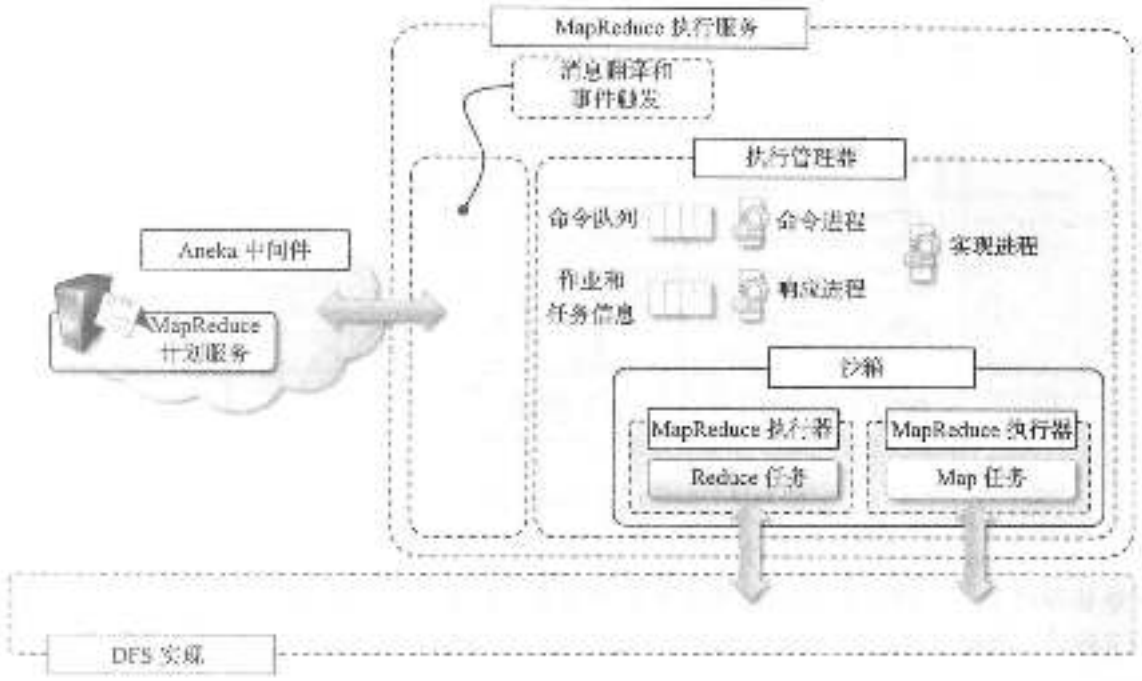


图 8-10　MapReduce 执行服务架构

3. 分布式文件系统支持

不同于 Aneka 支持的其他编程模型，MapReduce 模型不利用默认存储服务进行存储和数据传输，而是采用分布式文件系统来实现。这样做是因为 MapReduce 在文件管理方面的要求与其他模型有显著不同。特别地，MapReduce 用于处理储存在大型文件的大量数据，因此，分布式文件系统提供的支持更加合适，它可以利用多个节点存储数据。分布式文件系统的实现通过复制和分配的方式保证高可用性和更好的效率。此外，原来的 MapReduce 实现假定一个分布和可靠的存储的存在，因此，实现存储层分布式文件系统的使用是很自然的。

287 ~ 290

Aneka 提供了不同存储实现的接口能力，如第 5 章（5.2.3 节）所述，并为分布式文件系统的整合保持相同的灵活性。MapReduce 所需的集成级别需要具有执行以下任务的能力：

- 检索文件和文件块的位置。
- 通过流的方式访问文件。

对于根据数据位置优化 map 和 reduce 任务的调度来说，第一个操作是非常有用的；第二个操作是常见的 I/O 操作，且是数据文件所需的。在分布式文件系统中，如果该文件块不存储在本地节点上，流还可能会访问网络。通过提供合适的实现方法，Aneka 提供了可以进行这些操作的接口，并有能力阻塞它们背后不同的文件系统。当前的实现还提供了与 HDFS 的绑定。

291

在这些低层次的接口上，MapReduce 编程模型提供了以顺序方式读取和写入文件的类：SeqReader 和 SeqWriter。它们提供了对读写键值对的顺序访问，并且要求得到一个特定的文件格式，如图 8-11 所示。

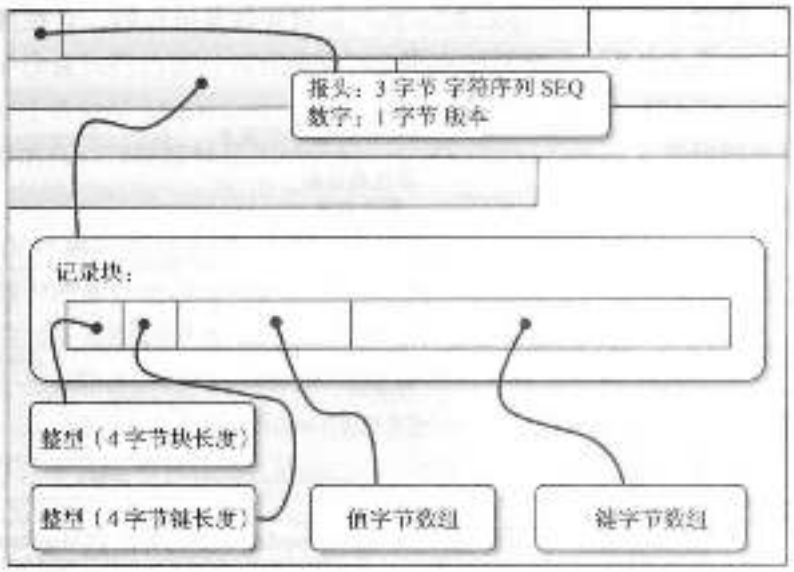


图 8-11　Aneka MapReduce 数据文件格式

一个 Aneka MapReduce 文件由用于标识文件的报头和记录块的序列组成，并且每一个记录块存储了一个键 - 值对。报头由 4 字节组成：前 3 个字节表示字符序列 SEQ，第四个字节标识文件的版本。记录块的组成如下：前 8 个字节用于存储表示该块的剩余部分长度和紧随其后的键长度的两个整数。该块的剩余部分存储由成对的值组成的数据。SeqReader 和 SeqWriter 类用于读写这种格式的文件，读写方法是透明地处理文件格式信息，以及键 - 值实例和它们的二进制表示之间的翻译转换。MapReduce 支持所有 .NET 内置类型。由于 MapReduce 作业对于普通文本文件中表示的数据操作非常频繁，SeqReader 类和 SeqWriter 类的特定版本已经可以作为键 - 值对的序列来读取和写入文本文件。在读操作的情况下，每个对的值通过在文本文件中的一行来表示，而键则是自动生成并分配到文件中该行开始的
292 字节位置。在写操作中，键的写入被跳过且值被保存为单行。

程序 8-8 显示了 SeqReader 和 SeqWriter 类的接口。SeqReader 类提供一个基于枚举的方法，该方法可通过调用 NextKey() 和 NextValue() 方法分别访问键值和数值。它也可以通过使用 NextRawKey() 和 NextRawValue() 来访问键值和数值的原始字节数据。HasNext() 返回一个布尔值，指示是否能够读取更多的对。SeqWriter 类提供了 Append 方法的不同版本。

程序 8-8　SeqReader 和 SeqWriter 类

```
using Aneka.MapReduce.Common;

namespace Aneka.MapReduce.DiskIO
{
    /// <summary>
    /// Class <b><i>SeqReader</i></b>. This class implements a file reader for the sequence
    /// file, which is a standard file split used by MapReduce.NET to store a partition of a
    /// fixed size of a data file. This classprovides an interface for exposing the content
    /// of a file split as an enumeration of key-value pairs and offersfacilities for both
    /// accessing keys and values as objects and their corresponding binary values.
    /// </summary>
    public class SeqReader
    {
        /// <summary>
        /// Creates a SeqReader instance and attaches it to the given file. This constructor
        /// initializes the instance with the default value for the internal buffers and does
```

```
    /// not set any information about the types of the keys and values read from the
    /// file.
    /// </summary>
    public SeqReader(string file) : this(file, null, null) { ... }
    /// <summary>
    /// Creates a SeqReader instance, attaches it to the given file, and sets the
    /// internal buffer size to bufferSize. This constructor does not provide any
    /// information about the types of the keys and values read from the file.
    /// </summary>
    public SeqReader(string file, int bufferSize) : this(file,null,null,bufferSize) { ... }
    /// <summary>
    /// Creates a SeqReader instance, attaches it to the given file, and provides
    /// metadata information about the content of the file in the form of keyType and
    /// valueType. The internal buffers are initialized with the default dimension.
    /// </summary>
    public SeqReader(string file, Type keyType, Type valueType)
        : this(file, keyType, valueType, SequenceFile.DefaultBufferSize) { ... }
    /// <summary>
    /// Creates a SeqReader instance, attaches it to the given file, and provides
    /// metadata information about the content of the file in the form of keyType and
    /// valueType. The internal buffers are initialized with the bufferSize dimension.
    /// </summary>
    public SeqReader(string file, Type keyType, Type valueType, int bufferSize){ ... }
    /// <summary>
    /// Sets the metadata information about the keys and the values contained in the data
    /// file.
    /// </summary>
    public void SetType(Type keyType, Type valueType) { ... }
    /// <summary>
    /// Checks whether there is another record in the data file and moves the current
    /// file pointer to its beginning.
    /// </summary>
    public bool MaxNext() { ... }
    /// <summary>
    /// Gets the object instance corresponding to the next key in the data file,
    /// in the data file.
    /// </summary>
    public object NextKey() { ... }
/// <summary>
/// Gets the object instance corresponding to the next value in the data file,
/// in the data file.
/// </summary>
public object NextValue() { ... }
/// <summary>
/// Gets the raw bytes that contain the value of the serializedinstance of the
/// current key.
/// </summary>
public BufferInMemory NextRawKey() { ... }
/// <summary>
/// Gets the raw bytes that contain the value of the serialized instance of the
/// current value.
/// </summary>
public BufferInMemory NextRawValue() { ... }

/// <summary>
/// Gets the position of the file pointer as an offset from its beginning.
/// </summary>
public long CurrentPosition() { ... }
/// <summary>
/// Gets the size of the file attached to this instance of SeqReader.
/// </summary>
public long StreamLength() { ... }
/// <summary>
/// Moves the file pointer to position. If the value of position is 0 or negative,
/// returns the current position of the file pointer.
/// </summary>
public long Seek(long position) { ... }
```

```
    /// <summary>
    /// Closes the SeqReader instanceand releases all the resources that have been
    /// allocated to read fromthe file.
    /// </summary>
    public void Close() { ... }

    // private implementation follows
}

/// <summary>
/// Class SeqWriter. This class implements a file writer for the sequence
/// sequence file, which is a standard file split used by MapReduce.NET to store a
/// partition of a fixed size of a data file. This classprovides an interface to add a
/// sequence of key-value pair incrementally.
/// </summary>
public class SeqWriter
{
    /// <summary>
    /// Creates a SeqWriter instance for writing to file. This constructor initializes
    /// the instance with the default value for the internal buffers.
    /// </summary>
    public SeqWriter(string file) : this(file, SequenceFile.DefaultBufferSize) { ... }
    /// <summary>
    /// Creates a SeqWriter instance, attachesit to the given file, and sets the
    /// internal buffer size to bufferSize.
    /// </summary>
    public SeqWriter(string file, int bufferSize) { ... }

        /// <summary>
        /// Appends a key-value pair to the data file split.
        /// </summary>
        public void Append(object key, object value) { ... }
        /// <summary>
        /// Appends a key-value pair to the data file split.
        /// </summary>
        public void AppendRaw(byte[] key, byte[] value) { ... }
        /// <summary>
        /// Appends a key-value pair to the data file split.
        /// </summary>
        public void AppendRaw(byte[] key, int keyPos, int keyLen,
                              byte[] value, int valuePos, int valueLen) { ... }

        /// <summary>
        /// Gets the length of the internal buffer or 0 if no buffer has been allocated.
        /// </summary>
        public long Length() { ... }
        /// <summary>
        /// Gets the length of data file split on disk so far.
        /// </summary>
        public long FileLength() { ... }
        /// <summary>
        /// Closes the SeqReader instance and releases all the resources that have been
        /// allocated to write to the file.
        /// </summary>
        public void Close() { ... }

        // private implementation follows
    }
}
```

程序 8-9 显示了通过在字 - 计数器例子中使用的回调函数来实现的 SeqReader 类的一个实际运用。为使应用程序的结果可视化，使用 SeqReader 类来读取输出文件的内容并转储为对任何文本编辑器可视的一个适当的文本形式，如记事本应用程序。

程序 8-9　字－计数器作业

```
using System.IO;
using Aneka.Entity;
using Aneka.MapReduce;

namespace Aneka.MapReduce.Examples.WordCounter
{
    /// <summary>
    /// Class Program. Application driver for the Word Counter sample.
    /// </summary>
    public class Program
    {
        /// <summary>
        /// Reference to the configuration object.
        /// </summary>
        private static Configuration configuration = null;
        /// <summary>
        /// Location of the configuration file.
        /// </summary>
        private static string confPath = "conf.xml";

        /// <summary>
        /// Processes the arguments given to the application and according
        /// to the parameters read runs the application or shows the help.
        /// </summary>
        /// <param name="args">program arguments</param>
        private static void Main(string[] args)
        {
            try
            {
                Logger.Start();

                // get the configuration
                Program configuration = Configuration.GetConfiguration(confPath);

                // configure MapReduceApplication
                MapReduceApplication<WordCountMapper, WordCountReducer> application =
                  new MapReduceApplication<WordCountMapper, WordCountReducer>("WordCounter",
configuration);
                // invoke and wait for result
                application.InvokeAndWait(new EventHandler<ApplicationEventArgs>(OnDone));

                    // alternatively we can use the following call
                    // application.InvokeAndWait();
            }
            catch(Exception ex)
            {
                Program.Usage();
                IOUtil.DumpErrorReport(ex, "Aneka WordCounter Demo - Error Log");
            }
            finally
            {
                Logger.Stop();
            }
        }
/// <summary>
/// Hooks the ApplicationFinished events and process the results
/// if the application has been successful.
/// </summary>
/// <param name="sender">event source</param>
/// <param name="e">event information</param>
private static void OnDone(object sender, ApplicationEventArgs e)
{
    if (e.Exception != null)
    {
        IOUtil.DumpErrorReport(e.Exception, "Aneka WordCounter Demo - Error");
```

```
        }
        else
        {
            string outputDir = Path.Combine(configuration.Workspace, "output");
            try
            {
                FileStream resultFile = new FileStream("WordResult.txt",FileMode.Create,
                                                       FileAccess.Write);
                StreamWriter textWriter = new StreamWriter(resultFile);

                DirectoryInfo sources = new DirectoryInfo(outputDir);
                FileInfo[] results = sources.GetFiles();
                foreach(FileInfo result in results)
                {
                    SeqReader seqReader = new SeqReader(result.FullName);
                    seqReader.SetTypes(typeof(string), typeof(int));

                    while(seqReader.HasNext() == true)
                    {
                        object key = seqReader.NextKey();
                        object value = seqReader.NextValue();

                        textWriter.WriteLine("{0}\t{1}", key, value);
                    }

                    seqReader.Close();
                }
                textWriter.Close();
                resultFile.Close();

                // clear the output directory
                sources.Delete(true);

                Program.StartNotepad("WordResult.txt");
            }
            catch(Exception ex)
            {
                IOUtil.DumpErrorReport(ex.Exception, "Aneka WordCounter Demo - Error");
            }
        }
    }

    /// <summary>
    /// Starts the notepad process and displays the given file.
    /// </summary>
    private static void StartNotepad(string file) { ... }
    /// <summary>
    /// Displays a simple informative message explaining the usage of the
    /// application.
    /// </summary>
    private static void Usage() { ... }
}
```

OnDone 回调会检查应用程序是否已成功终止。如果没有错误，它将遍历下载到工作区中输出目录的结果文件。默认情况下，文件保存在工作区目录的输出子目录。对于每一个结果文件，它会在上面打开一个 SeqReader 实例，并将键 - 值对的内容转储成一个文本文件，此文本文件可以通过任何文本编辑器打开。

8.3.2 应用实例

MapReduce 是处理大量数据的一个非常有用的模型，这些数据在很多情况下都保持一个半结构化的形式，例如日志或 Web 页面。为了说明如何用 Aneka MapReduce 编写实际应

用程序，我们考虑一个很常见的任务：日志分析。设计一个 MapReduce 应用程序来处理由 Aneka 容器产生的日志，以提取一些有关云行为的摘要信息。本节将详细描述要解决的问 293
题，并设计用于执行文件分析和数据抽象操作的 Mapper 和 Reducer 类。

1. 分析 Aneka 日志

Aneka 组件（守护进程、容器实例和服务）产生大量以日志文件形式存储的信息。最相关的信息被存储在容器实例的日志中，这些信息与在云中执行的应用程序有关。在这个例子中，我们通过分析这些日志来提取有关应用执行和云服务使用的有用信息。

整个构架利用了 log4net 库，用于收集和存储日志信息。在 Aneka 容器中，系统配置产生一个日志文件，每次容器重启时该日志文件被划分成块。此外，包含在日志文件中的信息可根据它的出现来定制。目前默认的布局如下： 294 ~ 296

DD MMM YY hh: mm: ss level-message

格式化日志消息中的某些例子如下：

15 Mar 2011 10:30:07 DEBUG - SchedulerService: ...
HandleSubmitApplication-SchedulerService: ...
15 Mar 2011 10:30:07 INFO-SchedulerService: Scanning candidate storage ...
15 Mar 2011 10:30:10 INFO-Added {WU: 51d55819-b211-490f-b185-8a25734ba705, 4e86fd02...
15 Mar 2011 10:30:10 DEBUG-StorageService:NotifyScheduler-Sending FileTransferMessage...
15 Mar 2011 10:30:10 DEBUG-IndependentSchedulingService:QueueWorkUnit-Queueing...
15 Mar 2011 10:30:10 INFO-AlgorithmBase::AddTasks{64} Adding 1 Tasks
15 Mar 2011 10:30:10 DEBUG-AlgorithmBase:FireProvisionResources-Provision Resource: 1

在示例日志行的内容中，可以看出几乎所有日志行的消息部分都呈现出类似的结构，并且它们与进入日志行的组件信息一起开始。通过查看不包含空格的字符序列之后首次出现的 ":" 字符，可以很容易地将该信息提取出来。

从这些日志中提取的信息可能是以下情况：

- 根据级别的日志消息分布。
- 根据组件的日志消息分布。

这些信息可以通过创建 Mapper 任务很容易地提取并组成一个单一的视图。此任务计数日志级别和组件名称的出现次数，并为每次的出现发射一个（level-name，1）或（component-name，1）形式的简单的键 - 值对。Reducer 任务简单地将所有具有相同键的键值对相加。对于这两个问题，map 和 reduce 函数的结构如下所示：

$$map:(long, string) \Rightarrow (string, long)$$

$$reduce:(string, long) \Rightarrow (string, long)$$

此后 Mapper 类将收回包含文件中行位置的键 - 值对作为键，并且提取日志消息作为值 297 ~ 299
组件。它会产生一个键 - 值对，这个键 - 值对包含了一个表示日志级别名称或组件名称的字符串，并且以 1 作为值。Reducer 类将总结所有具有相同名称的键 - 值对。通过修改之前讨论的典型结构，可以在同一时间执行两种分析，而不是开发两种不同的 MapReduce 工作。注意到由 Reducer 类进行的操作在这两种情况下是相同的，而 Mapper 类的操作发生了变化，但是这两个作业所生成的键 - 值对的类型是相同的。因此，将 map 函数执行的两个任务组合到单一的 Mapper 类是可能的，此类将为每个输入行产生两个键 - 值对。此外，可以通过

日志级别的名称区别 Aneka 组件的名称，方法是使用一个初始下划线字符。为了表示和组织数据，后处理 reduce 函数的输出将会是非常容易的。

2. Mapper 设计和实现

map 函数执行的操作是一个非常简单的文本提取，用于标识日志的级别和进入日志中信息的组件的名称。一旦这个信息被提取，则一个键 - 值对（string，long）被函数发出。既然决定合并两个 MapReduce 作业为一个单一的作业，那么每个 map 任务将最多发出两个键 - 值对。这是因为有些日志行不记录所输入行的组件名称，对于这些行，只有对应的日志级别的键 - 值对将被发出。

程序 8-10 显示了 Mapper 类的日志分析任务的实现。Map 方法简单地把日志级别标签的位置定位到行，提取并发出相应的键 - 值对（label，1）。然后 Map 方法会尝试通过查找由“：”限制的字符序列，找出输入记日志行的 Aneka 组件名称的位置。如果这一序列中不包含空格，则代表了 Aneka 组件的名称。在这种情况下，另一个键 - 值（component-name，1）被发出。如前所述，为了从组件名称来区别日志级别的标签，在第二种情况下将一个下划线前缀于键的名称。

程序 8-10　日志分析的 Mapper 实现

```
using Aneka.MapReduce;

namespace Aneka.MapReduce.Examples.LogParsing
{
    /// <summary>
    /// Class LogParsingMapper. Extends Mapper<K,V> and provides an
    /// implementation of the map function for parsing the Aneka container log files.
    /// This mapper emits a key-value (log-level, 1) and potentially another key-value
    /// (_aneka-component-name,1) if it is able to extract such information from the
    /// input.
    /// </summary>
    public class LogParsingMapper: Mapper<long,string>
    {
        /// <summary>
        /// Reads the input and extracts the information about the log level and if
        /// found the name of the aneka component that entered the log line.
        /// </summary>
        /// <param name='input'>map input</param>
        protected override void Map(IMapInput<long,string>input)
        {
            // we don't care about the key, because we are only interested on
            // counting the word of each line.
            string value = input.Value;
            long quantity = 1;

            // first we extract the log level name information. Since the date is reported
            // in the standard format DD MMM YYYY hh:mm:ss it is possible to skip the first
            // 20 characters (plus one space) and then extract the next following characters
            // until the next position of the space character.
            int start = 21;
            int stop = value.IndexOf(' ', start);
            string key = value.Substring(start, stop - start);

            this.Emit(key, quantity);

            // now we are looking for the Aneka component name that entered the log line
            // if this is inside the log line it is just right after the log level preceeded
            // by the character sequence <space><dash><space> and terminated by the <colon>
            // character.

            start = stop + 3; // we skip the <space><dash><space> sequence.
```

```
            stop = value.IndexOf(' ', start);

            key = value.Substring(start, stop - start);

            // we now check whether the key contains any space. if not then it is the name
            // of an Aneka component and the line does not need to be skipped.
            if (key.IndexOf('_') == -1)
            {
                this.Emit("_" + key, count);
            }
        }
    }
}
```

3. Reducer 设计和实现

Reduce 函数的实现更加简单，需要执行的唯一操作是添加被关联到相同键的所有值，并发出一个键 – 值对的总和。该基础设施将合计已经发给一个给定键的所有值，因此，只需要简单地遍历值的集合并将它们相加。

如程序 8-11 所示，要执行的操作非常简单，并且实际上对于从日志行中提取的两种不同的键 – 值对是相同的。区分集合在输出文件中的不同类型的信息将由驱动程序负责。

程序 8-11　Aneka 日志分析的 Reducer 实现

```
using Aneka.MapReduce;

namespace Aneka.MapReduce.Examples.LogParsing
{
    /// <summary>
    /// Class <b>LogParsingReducer</b> extends Reducer<K,V> and provides an
    /// implementation of the reduce function for parsing the Aneka container log files.
    /// The Reduce method iterates all over values of the enumerator and sums the values
    /// before emitting the sum to the output file.
    /// </summary>
    public class LogParsingReducer : Reducer<string,long>
    {
        /// <summary>
        /// Iterates all over the values of the enumerator and sums up
        /// all the values before emitting the sum to the output file.
        /// </summary>
        /// <param name="input">reduce source</param>
        protected override void Reduce(IReduceInputEnumerator<long> input)
        {
            long sum = 0;

            while(input.MoveNext())
            {
                long value = input.Current;
                sum += value;
            }
            this.Emit(sum);
        }
    }
}
```

4. 驱动程序

LogParsingMapper 和 LogParsingReducer 构成 MapReduce 作业的核心功能，只需要在主程序中进行适当配置，以处理和产生文本工程。另外，由于已经设计了 mapper 组件来提取两种不同类型的信息，所以驱动应用中执行的另一任务是将这两个统计分离成不同的文件

300 来进一步地分析。

程序 8-12 显示了驱动程序的实现。相对于前面的例子，有三个事项值得注意：

程序 8-12 驱动程序实现

```
using System.IO;
using Aneka.Entity;
using Aneka.MapReduce;

namespace Aneka.MapReduce.Examples.LogParsing
{
    /// <summary>
    /// Class Program. Application driver. This class sets up the MapReduce
    /// job and configures it with the <i>LogParsingMapper</i> and <i>LogParsingReducer</i>
    /// classes. It also configures the MapReduce runtime in order sets the appropriate
    /// format for input and output files.
    /// </summary>
    public class Program
    {
        /// <summary>
        /// Reference to the configuration object.
        /// </summary>
        private static Configuration configuration = null;
        /// <summary>
        /// Location of the configuration file.
        /// </summary>
        private static string confPath = "conf.xml";

        /// <summary>
        /// Processes the arguments given to the application and according
        /// to the parameters read runs the application or shows the help.
        /// </summary>
        /// <param name="args">program arguments</param>
        private static void Main(string[] args)
        {
            try
            {
                Logger.Start();

                // get the configuration
                Program.configuration = Program.Initialize(confPath);
                // configure MapReduceApplication
                MapReduceApplication<LogParsingMapper, LogParsingReducer> application =
                 new MapReduceApplication<LogParsingMapper, LogParsingReducer>("LogParsing",
configuration);
                // invoke and wait for result
                application.InvokeAndWait(new EventHandler<ApplicationEventArgs>(OnDone));

                    // alternatively we can use the following call
                    // application.InvokeAndWait();
            }
            catch(Exception ex)
            {
                Program.ReportError(ex);
            }
            finally
            {
                Logger.Stop();
            }
        Console.ReadLine();
    }
    /// <summary>
    /// Initializes the configuration and ensures that the appropriate input
    /// and output formats are set
    /// </summary>
    /// <param name="configFile">A string containing the path to the config file.</param>
```

```
/// <returns>An instance of the configuration class.</returns>
private static Configuration InitializeStringConfigFile()
{
    Configuration conf = Configuration.GetConfiguration(confPath);
    // we ensure that the input and the output formats are simple
    // text files.
    PropertyGroup mapReduce = conf["MapReduce"];
    if (mapReduce == null)
    {
        mapReduce = new PropertyGroup("MapReduce");
        conf.Add("MapReduce") = mapReduce;
    }
    // handling input properties
    PropertyGroup group = mapReduce.GetGroup("Input");
    if (group == null)
    {
        group = new PropertyGroup("Input");
        mapReduce.Add(group);
    }
    string val = group["Format"];
    if (string.IsNullOrEmpty(val) == true)
    {
        group.Add("Format", "text");
    }
    val = group["Filter"];
    if (string.IsNullOrEmpty(val) == true)
    {
        group.Add("Filter", "*.log");
    }
    // handling output properties
    group = mapReduce.GetGroup("Output");
    if (group == null)
    {
        group = new PropertyGroup("Output");
        mapReduce.Add(group);
    }
    val = group["Format"];
    if (string.IsNullOrEmpty(val) == true)
    {
        group.Add("Format", "text");
    }
    return conf;
}

/// <summary>
/// Hooks the ApplicationFinished events and process the results
/// if the application has been successful.
        /// </summary>
        /// <param name="sender">event source</param>
        /// <param name="e">event information</param>
        private static void OnDone(object sender, ApplicationEventArgs e)
        {
            if (e.Exception != null)
            {
                Program.ReportError(ex);
            }
            else
            {
                Console.Write("Aneka Log Parsing Job Terminated: SUCCESS");

                FileStream logLevelStats = null;
                FileStream componentStats = null;
                string workspace = Program.configuration.Workspace;
                string outputDir = Path.Combine(workspace, "output");
                DirectoryInfo sources = new DirectoryInfo(outputDir);
                FileInfo[] results = sources.GetFiles();
```

```
                try
                {
                    logLevelStats = new FileStream(Path.Combine(workspace,"loglevels.txt"),
        FileMode.Create,FileAccess.Write);
                    componentStats = new FileStream(Path.Combine(workspace,"components.txt"),
                                                    FileMode.Create,
    FileAccess.Write);
                    using(StreamWriter logWriter = new StreamWriter(logLevelStats))
                    {
                        using(StreamWriter compWriter = new StreamWriter(componentStats))
                        {
                            foreach(FileInfo result in results)
                            {
                                using(StreamReader reader =
                                    new StreamReader(result.OpenRead()))
                                {
                                    while(reader.EndOfStream == false)
                                    {
                                        string line = reader.ReadLine();
                                        if (line != null)
                                        {
                                            if (line.StartsWith("_") == true)
                                            {
                                                compWriter.WriteLine(line.Substring(1));
                                            }
                                            else
                                            {
                                                logWriter.WriteLine(line);
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                    // clear the output directory
                    sources.Delete(true);

                    Console.WriteLine("Statistics saved to:{loglevels.txt, components.txt}");

                    Environment.ExitCode = 0;
                }
                catch(Exception ex)
                {
                    Program.ReportError(ex);
                }
                Console.WriteLine("<Press Return>");
            }
        }
        /// <summary>
        /// Displays a simple informative message explaining the usage of the


        /// application.
        /// </summary>
        private static void Usage()
        {
            Console.WriteLine("Aneka Log Parsing - Usage Log.Parsing.Demo.Console.exe"
            + " [conf.xml]");
        }
        /// <summary>
        /// Dumps the error to the console, sets the exit code of the application to -1
        /// and saves the error dump into a file.
        /// </summary>
```

```
        /// <param name="ex">runtime exception</param>
        private static void ReportError(Exception ex)
        {
            IOUtil.DumpErrorReport(Console.Out, ex, "Aneka Log Parsing Job Terminated: "
                + "ERROR");
            IOUtil.DumpErrorReport(ex, "Aneka Log Parsing-Job Terminated: ERROR");
            Program.Usage();
            Environment.ExitCode = -1;
        }
    }
}
```

- MapReduce 作业的配置。
- 结果文件的后处理。
- 错误管理。

作业的配置由 Initialize 方法执行。这个方法从本地文件系统中读取配置文件，并确保文件的输入和输出格式被设置为文本。MapReduce 作业可以通过使用名为 MapReduce 的配置文件的制定部分进行配置。在这一部分，两个小部分控制输入和输出文件的属性，并分别命名为 Input 和 Output。输入和输出部分可以包含下列属性：

- 格式（字符串）定义了输入文件的格式。如果这一属性被设置，那么唯一支持的值是文本。
- 过滤器（字符串）定义了检索模式，用于过滤输入文件以处理工作区目录。此属性仅适用于输出属性组。
- 换行（字符串）定义了用于在文本流中检测（或写入）一个新行的字符序列。当输入/输出格式被设置为文本，并且默认值选自执行环境时，这个值是有意义的。
- 分离器（字符）属性只出现在部分，并定义了需要用于在输出文件中从值分离出键的字符。如同前一属性，当输入/输出格式设置为文本时该值是有意义的。

301 / 302

除了用于输入和输出文件的具体设置，也可以控制 MapReduce 作业的其他参数。这些参数在主要的 MapReduce 的配置部分中定义，它们的含义在 8.3.1 节中讨论过。

除了初始化配置的编程方法，也可以将这些设置嵌入到标准 Aneka 配置文件，如程序 8-13 所示。

程序 8-13　驱动程序配置文件（conf.xml）

```
<?xml version="1.0" encoding="utf-8" ?>
<Aneka>
  <UseFileTransfer value="false" />
  <Workspace value="Workspace" />
  <SingleSubmission value="AUTO" />
  <PollingTime value="1000" />
  <LogMessages value="false" />
  <SchedulerUri value="tcp://localhost:9090/Aneka" />
  <UserCredential type="Aneka.Security.UserCredentials" assembly="Aneka.dll">
    <UserCredentials username="Administrator" password="" />
  </UserCredentials>
  <Groups>
    <Group name="MapReduce">
      <Groups>
        <Group name="Input">
        <Property name="Format" value="text" />
        <Property name="Filter" value="*.log" />
      </Group>
```

```
      <Group name='Output'>
        <Property name='Format' value='text' />
      </Group>
    </Groups>
    <Property name='LogFile' value='Execution.log' />
    <Property name='FetchResult' value='true' />
    <Property name='UseCombiner' value='true' />
    <Property name='SynchReduce' value='false' />
    <Property name='Partitions' value='1' />
    <Property name='Attempts' value='3' />
  </Group>
    </Groups>
</Aneka>
```

如程序 8-13 所示，可以打开一个 < Group name =" MapReduce" > ... < /Group> 标签并输入所有执行所需的属性。该 Aneka 配置文件基于一个灵活的框架，允许简单地输入名称 - 值属性组。如果可能，Aneka.Property 和 Aneka.PropertyGroup 类还具备将代表属性值的字符串转换成相应的内置类型的能力，这简化了读取和写入配置对象的任务。

程序 8-12 中所示的第二个元素由输出文件的后处理表示。这一操作由 OnDone 方法执行，其调用被触发的条件是 MapReduce 作业的执行过程中发生错误，或执行成功完成。该方法从 Aneka 组件名称的出现中分离日志级别标签的出现，通过将它们在工作区目录下保存成两个不同的文件（loglevels.txt 和 components.txt）来完成这一功能，然后删除已下载的 reduce 阶段的输出文件所在的输出目录。这两个文件包含了分析的汇总结果，并可用于提取有关日志文件的内容的统计信息，并以图形方式显示，下一节将讨论这一问题。

可以考虑的最后一个方面是错误管理。Aneka 提供了一个包含在 Aneka.Util 库中的 API 集合，代表用于自动化繁琐任务的实用类，例如与异常关联的堆栈跟踪信息或关于异常抛出类型的信息的适当收集。在这个例子中，例外情况下被触发的报告功能是用 ReportError 方法实现的。该方法利用了 IOUtil 类提供的能力，以便把一个简单的错误报告转储到控制台和正在使用以下模式命名的日志文件中：error.YYYY-MM-DD_hh-mm-ss.log。

5. 运行应用程序

Aneka 产生数量相当可观的日志信息。日志基础设施的默认配置是为容器处理的每个活动创建一个新的日志文件，或一旦日志文件的范围超出 10M，也将创建一个新的日志文件。因此，通过简单地持续几天运行 Aneka 云，收集足够的数据用于示例应用的挖掘是很容易的。此外，此方案也是 MapReduce 的一个真实案例，因为它最常见的实际应用之一是从日志和执行痕迹中提取半结构化信息。

在测试的执行中使用了一个分布式基础设施，它包括七个工作节点和一个通过 LAN 互连的主节点。我们将不同大小的 18 个日志文件处理成 122 MB 的总大小。MapReduce 作业在收集数据上的执行产生存储在 loglevels.txt 和 components.txt 文件中的结果，并分别在图 8-12 和图 8-13 中用图形表示。

这两个图表明，在存储器处理产生的日志文件中有相当数量的非结构化信息。特别地，约 60% 的记录内容在分类中被跳过。此内容更可能根据堆栈跟踪的结果而转储到日志文件中，作为错误和警告记录的结果产生一个不被识别的行序列。图 8-13 展示了使用日志 API 的组件之间的分布。该分布在识别为有效的日志项中的数据上计算，该图形显示这些项中只

有约 20% 没有被 map 函数执行的解析器识别。之后可以推断，从日志分析中提取的有意义的信息约占整个日志数据的 32%（分析过的全部行的 40% 中的 80%）。

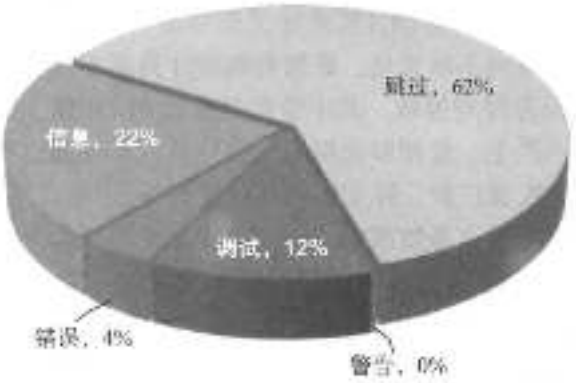


图 8-12　日志级别记录分布

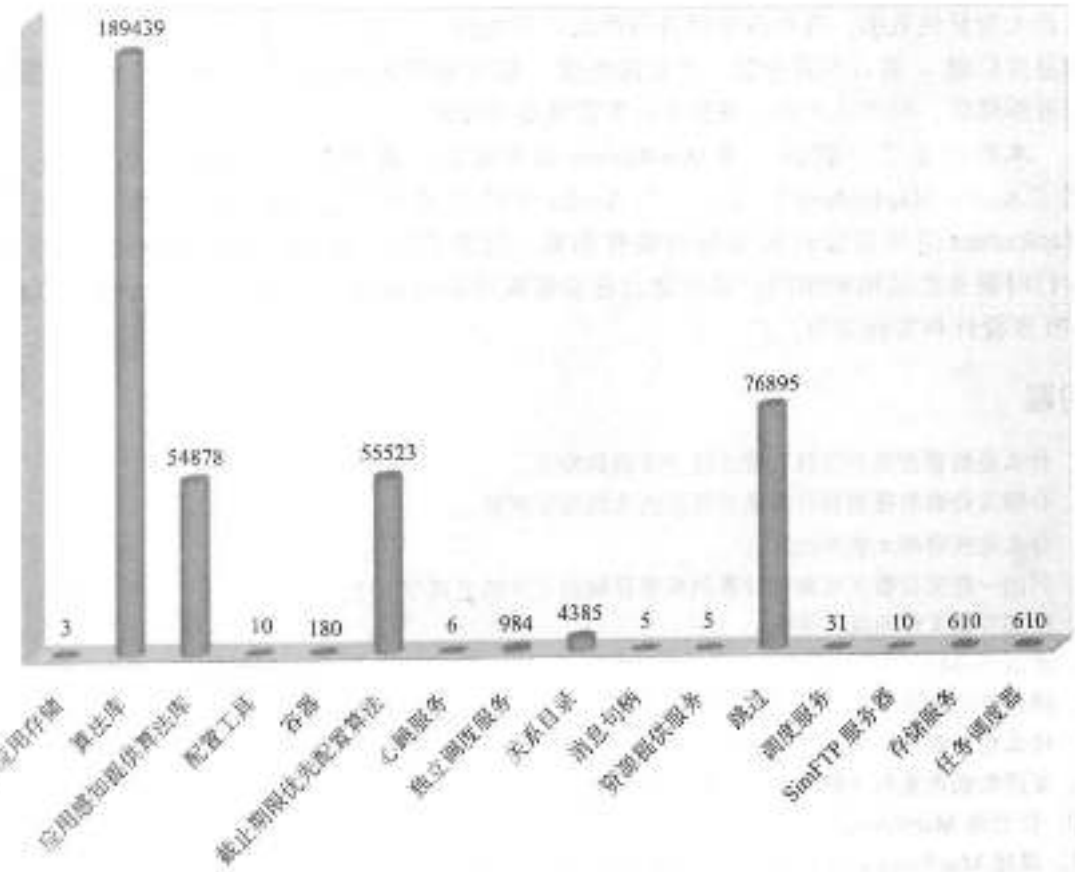


图 8-13　组件记录分布

尽管 map 任务实现的分析函数比较简单，但这个实际例子说明了 Aneka MapReduce 编程模型如何用于简单地执行海量数据分析任务。研究该案例的目的不是为了创建一个功能非常强大的分析函数，而是为了演示如何在逻辑上和编程上使用 MapReduce 处理真实的数据分析案例，以及如何在 Aneka 的 API 上实现它。

本章小结

本章介绍了数据密集型计算的主要特点。数据密集型应用处理或产生海量数据，或呈现计算密集型的性质。触发数据密集型计算的定义数据量，以及用于数据密集型计算的技术、编程和存储模型，都在随时间不断变化。数据密集型计算原本是在高速广域网的应用中突出的一个领域，现在则是云存储的领域，其中数据的量达到 TB 级（如果不是 PB 级），而且被称为大数据。这个词表示产生、处理和挖掘的大量信息，不仅通过科学应用也通过提供互联网服务的公司，如搜索、在线广告、社交媒体和社交网络。

大数据世界的一个有趣特点是数据以一个半结构化或非结构化的形式表示。因此，基于关系数据库的传统方法不能够有效地支持数据密集型应用。为了应对这些挑战，人们研发了新的方法和存储模型。在存储系统的背景下，最重要的努力方向是高性能分布式文件系统、存储云和基于 NoSQL 系统的实现。对于编写数据密集型应用的支持，最相关的创新是 MapReduce 的引进及其所有的改进版本，旨在将该方法的适用性扩展到更大范围的场景。

MapReduce 由谷歌提议并提供了一种简单的方法来处理基于 map 和 reduce 两个函数定义的大批量的数据。这些数据可进行两段式的处理。首先，map 阶段从数据中提取有价值的信息并以键 – 值对形式存储，之后这些键 – 值对最终在 reduce 阶段聚集起来。尽管这一模型有所局限，但在几个应用场景中已被证明是成功的。

本章讨论了谷歌提出的 MapReduce 参考模型，并指明了其相关变化方向。之后描
309 述了 Aneka MapReduce 的实现，与 Aneka 中的线程和任务编程模型类似。讨论了支持 MapReduce 应用的设计和实现的编程抽象。接着介绍了为 MapReduce 作业执行的 Aneka 运行时服务的结构和组织。最后通过逐步实现的案例总结了如何运用 Aneka MapReduce 的 API 来设计和实施应用。

习题

1. 什么是数据密集型计算？描述这个术语的特点。
310 2. 介绍支持数据密集型计算最重要的技术的历史背景。
3. 什么是所谓的大数据的特征？
4. 列出一些支持数据密集型计算的云存储技术并描述其中一个。
5. 描述谷歌文件系统的架构。
6. 术语 NoSQL 是什么意思？
7. 描述亚马逊简单存储服务（S3）的特点。
8. 什么是谷歌 Bigtable？
9. 支持数据密集型计算的编程平台的要求是什么？
10. 什么是 MapReduce？
11. 描述 MapReduce 可以解决的问题类型并给出一些实例。
12. 列出 MapReduce 的一些变化或扩展。
13. Aneka MapReduce 编程模型的主要组成部分是什么？
14. MapReduce 模型与 Aneka 支持的和本书中讨论的其他模型有什么不同？
15. 描述构成支持 MapReduce 的运行时基础设施的调度和执行服务的组件。
311 16. 描述用于设计 Aneka MapReduce 的数据存储层的架构和用于处理 MapReduce 文件的 I/O API。
312 17. 使用 MapReduce 设计和实现一个简单的程序来完成 Pi 的计算。

第三部分

Mastering Cloud Computing: Foundations and Applications Programming

工业云平台与新进展

第 9 章

Mastering Cloud Computing: Foundations and Applications Programming

工业云平台

云计算允许终端用户和开发者利用大规模分布式计算的基础设施，使得基础设施管理软件和分布式计算平台提供按需的计算、存储及更高级的服务成为可能。关于建立企业级的云计算应用程序或者使用云计算技术来集成和扩展现有的行业应用，有几种不同的观点。表 9-1 概括了一些著名的云计算平台和它们提供的服务类型的简单描述。可以使用单一的技术和供应商或者它们的组合来开发一个云计算系统。

表 9-1 云计算产品案例

供应商 / 产品	服务类型	描述
亚马逊网络服务	IaaS，PaaS，SaaS	亚马逊网络服务（AWS）是一个网络服务的集合，为开发者提供计算、存储和更加高级的服务。IaaS 服务是 AWS 最流行的服务，弹性计算服务 EC2 是 AWS 最基本的服务
谷歌 AppEngine	PaaS	谷歌 AppEngine 是一个分布式可扩展的运行平台，基于 Java 和 Python 运行时环境来开发可扩展网络应用程序。这些丰富的访问服务形式，以可扩展的方式简化了应用程序的开发
微软 Azure	PaaS	微软 Azure 是一个云操作系统，为开发基于专有的 Hyper-V 虚拟化技术和 .NET 框架的可扩展的应用程序提供服务
SalesForce.com 和 Force.com	SaaS，PaaS	SalesForce.com 是一个“软件即服务”的解决方案，可以使用 CRM 应用的原型。它利用 Force.com 平台来开发新的 CRM 应用程序组件和功能
Heroku	PaaS	Heroku 是一个可扩展的运行时环境，基于 Ruby 构建应用程序
RightScale	IaaS	RightScale 是一个云管理平台，用一个单独的控制面板来管理公共云和混合云

本章给出了一些市场上具有代表性的基础设施即服务（IaaS）和平台即服务（PaaS）的云计算解决方案，并围绕云计算的主要技术和服务架构等实际问题展开讨论。

9.1 亚马逊 Web 服务

亚马逊 Web 服务（Amazon Web Services，AWS）是一个平台，通过提供弹性基础架构的可扩展性、消息传递和数据存储解决方案，开发灵活的应用程序。该平台通过 SOAP 或 RESTful Web 服务接口访问，并提供了基于 Web 的控制台，用户可以对所需的资源及基于“即用即付”计算方式的费用进行管理和监控。

图 9-1 给出了 AWS 生态系统提供的所有服务。解决方案的底层提供原始的计算和存储服务：亚马逊弹性计算云服务（Amazon Elastic Compute，EC2）和亚马逊简单存储服务（Amazon Simple Storage Service，S3），这是两个最流行的服务，一般与其他产品互相补充一起构成一个完整的系统。在更高层上，弹性 MapReduce 和自动伸缩（AutoScaling）为构建更智能、更弹性的计算系统提供了更好的附加功能。在数据方面，弹性块存储（Elastic BlockStore，EBS），亚马逊简单数据库（Amazon SimpleDB），亚马逊关系数据库（Amazon

RDS）和亚马逊弹性缓存（Amazon ElastiCache）提供了可靠的数据快照和结构化及半结构化的数据管理。覆盖在网络层的亚马逊虚拟私有云（Amazon Virtual Private Cloud，VPC）、弹性负载均衡（Elastic Load Balancing）、Amazon Route 53 和亚马逊直接连接（Amazon Direct Connect）完成通信需求。更加高级的连接应用程序服务包括：亚马逊简单队列服务（Amazon Simple QueueService，SQS），亚马逊简单通知服务（Amazon Simple Notification Service，SNS）和亚马逊简单邮件服务（Amazon Simple E-mail Service，SES）。其他服务包括：

313 ? 315

- 亚马逊 Cloud Front，内容发布网络解决方案。
- 亚马逊 CloudWatch，几个亚马逊服务的监控解决方案。
- 亚马逊弹性 BeanStalk 和 Cloud Formation，灵活的应用打包和部署。

如上所示，AWS 包括一系列广泛的服务。通过分析 AWS 解决方案相关的计算、存储、通信和配套服务，本章将讨论其中最重要的服务。



图 9-1 亚马逊 Web 服务生态系统

9.1.1 计算服务

计算服务构成云计算系统的基础服务。这里的基础服务指的是亚马逊 EC2，它提供了一个 IaaS 解决方案，已成为同一细分市场中若干其他厂商产品的参考模型。亚马逊 EC2 允许以虚拟机的形式部署服务器，虚拟机即创建一个特定映像的实例。映像配有预装的操作系统和软件栈，实例可以配置内存、多个处理器和存储器。用户提供凭证远程访问实例，如果需要，可以进一步配置或安装软件。

1. 亚马逊机器映像

亚马逊机器映像（Amazon Machine Images，AMIS）是用于创建虚拟机的模板。它们存储在 Amazon S3，以唯一的标识符“ami-xxxxxx”格式和一个清单 XML 文件来识别。一个 AMI 包含一个物理文件系统格式，可以安装预定义的操作系统。这些由亚马逊内存映像（Amazon Ramdisk Image，ARI，id：ari-yyyyyy）和亚马逊内核映像（Amazon Kernel Image，
316 AKI，id：aki-zzzzzz）指定，这二者是模板的部分配置。AMI 要么从头开始创建，要么从现有的 EC2 实例“捆绑”。通常的做法是：准备新的 AMI，从一个已经存在的 AMI 创建一个实例，一旦启动和运行就登录到实例，并安装所需的所有软件。使用亚马逊提供的工具，可以将实例转换到一个新的映像。一旦一个 AMI 被创建，它将存储在一个 S3 存储桶，用户可以决定其是否供其他用户使用或保留供个人使用。最后，还可以将产品代码与给定的 AMI 相关联，从而每当 AMI 创建 EC2 实例时，AMI 的拥有者将获得收益。

2. EC2 实例

EC2 实例代表虚拟机。EC2 使用 AMI 作为模板创建实例，模板是专门用来选择内核数量、计算能力以及所安装的存储器的。处理能力表现在虚拟内核和 EC2 计算单元（EC2
317 Compute Unit，ECU）。ECU 是一个虚拟内核的运算能力的度量方式，表示分配给一个实例的真实 CPU 的预测数量。通过使用计算单元，而不是实际频率值，亚马逊可以随着时间变化将这些单元映射为底层实际分配的计算能力值，从而保持 EC2 实例的性能与当前制定的标准相一致。随着时间的推移，支撑相关基础设施的硬件会被更强大的硬件替代，而有了 ECU 的帮助，EC2 实例提供给用户的性能是一致的。由于用户是租用计算能力，而不是购买硬件，所以这样的做法是合理的。一个 ECU 的性能与一个 1.0 ~ 1.2 GHz 2007 Opteron 或 2007 Xeon 处理器的运算能力[⊖]相同。

表 9-2 列出了 EC2 实例的所有当前可用的配置，可以分成六大类：

- 标准实例。此类提供了一组配置，适合于大多数应用。EC2 提供了三种不同的类型来增加计算能力、存储和内存。
- 微实例。此类型适合那些消费有限的计算能力和内存容量，偶尔需要打断 CPU 周期来处理激增的工作负荷的应用。微实例可用于小型 Web 应用程序与有限的通信量。
- 超大内存实例。这个类型针对需要处理巨大的工作量、需要大量内存的应用程序。高流量的三层 Web 应用程序是目标配置文件。三类方法可以增加内存和 CPU，其中占用内存的比例大于计算能力。
- 超强 CPU 实例。本类型针对计算密集型应用程序。有两种配置可供选择，其中运算能力比例的增加超过内存。
- 集群计算实例。这个类型用来提供虚拟集群服务。特点是高 CPU 计算能力和大容量内存，并具有极高的 I/O 和网络性能，这使得它适合于高性能计算应用程序。
- 集群 GPU 实例。这个类型提供的实例具有图形处理单元（GPU）和高计算能力，以及大容量存储器和非常高的 I/O 和网络性能。此类型特别适合于集群应用程序执行大量图形计算，如渲染集群。由于 GPU 可用于通用计算，这些实例的用户可以从附加的计算能力获益，这使得此类型适合 HPC 应用程序。

⊖ http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it.

表 9-2 亚马逊 EC2(按需)实例特征

	实例类型	ECU	平台	内存	磁盘存储	价格(美国东部)(USD/小时)
标准实例	小型	1(1×1)	32bit	1.7GB	160GB	$0.085 Linux $0.12 Windows
	大型	4(2×2)	64bit	7.5GB	850GB	$0.340Linux $0.48 Windows
	超大	8(4×2)	64bit	15GB	1690GB	$0.680Linux $0.96 Windows
微实例	微型	≤2	32/64bit	613MB	仅 EBS	$0.020Linux $0.03 Windows
超大内存实例	超大型	6.5(2×3.25)	64bit	17.1GB	420GB	$0.500Linux $0.62 Windows
	四倍超大型	13(4×3.25)	64bit	34.2GB	850GB	$1.000Linux $1.24 Windows
	八倍超大型	26(8×3.25)	64bit	68.4GB	1690GB	$2.000Linux $2.48 Windows
超强 CPU 实例	中型	5(2×2.5)	32bit	1.7GB	350GB	$0.170Linux $0.29 Windows
	超大型	20(8×2.5)	64bit	7GB	1690GB	$0.680Linux $1.16 Windows
集群实例	四倍超大型	33.5	64bit	23GB	1690GB	$1.600Linux $1.98 Windows
集群图像实例	八倍超大型	33.5	64bit	22GB	1690GB	$2.100Linux $2.60 Windows

EC2 实例的定价根据其类别按小时收费。在开始使用的每小时，用户将付整个小时的费用。一个实例每小时的费用是固定的。实例拥有者负责提供备份策略，因为不能保证实例将运行整个小时。另一种方法是用即时实例表示。这些实例在价格和有效期方面更具动态性，因为它们是根据 EC2 负载和资源可用性提供给用户使用的。用户定义其愿意支付的价格上限，只要目前的价格（即时价格）在给定的上限之下，实例就保持运行。即时实例在每小时开始时进行计费。即时实例比正常实例更不稳定，正常实例中 EC2 会尽量保持实例处于活 318
跃状态，即时实例就不能保证了。因此，执行备份和检查点策略是不可避免的。

EC2 实例可以通过使用亚马逊提供的命令行工具连接亚马逊网络服务。亚马逊网络服务允许远程访问 EC2 基础设施或 AWS 控制台。AWS 控制台可管理其他服务，如 S3。默认情况下，一个 EC2 实例由 AMI 相关的内核和磁盘创建。这些为实例定义架构（32 位或 64 位）和可用的磁盘空间，这个磁盘是暂时的，一旦实例关闭，磁盘的内容将丢失。或者，它可能将实例附加在一个 EBS 卷上，其内容将存储在 S3。如果默认的 AKI 和 ARI 不匹配，EC2 通过指定不同的 AKI 与 ARI 来提供运行 EC2 实例的能力，从而保证实例创建的灵活性。 319

3. EC2 环境

EC2 实例在虚拟环境中执行，该环境提供了实例运行应用程序所需的服务。EC2 环境负责分配地址、附加存储卷、配置访问控制和网络连接方面的安全。

默认情况下，实例由内部 IP 地址创建，这使得它们能够在 EC2 内部网络通信和作为客户端访问互联网。可将弹性 IP 关联到每个实例，超时的情况下可以重新映射到不同的实例。弹性 IP 允许运行在 EC2 的实例作为服务器访问互联网和执行故障转移能力，因为它们没有完全绑定到具体的实例。EC2 实例与外部 IP 一起指定了一个域名，一般形式是 ec2-xxx-xxx-xxx.compute-x.amazonaws.com，其中 xxx-xxx-xxx 通常代表用英文破折号分开的外部 IP 地址的四部分，compute-x 提供了实例部署的可用地区信息。目前，有五个价格不同的可用地区：两个在美国（弗吉尼亚州和加利福尼亚北部），一个在欧洲（爱尔兰），两个在亚太地区（新加坡和东京）。

实例所有者只能部分控制在哪里部署实例。不过，他们可以更精细地控制实例的安全性以及网络访问性。创建实例时，实例所有者可以将一个密钥对联合一个或多个实例。一旦实例运行，密钥对允许所有者远程连接并获得 root 访问权。亚马逊 EC2 通过基本防火墙

配置控制一个虚拟实例的可访问性，规定源地址、端口和协议（TCP、UDP、ICMP）。规则也可以连接到安全组，在实例部署之前作为一个或多个组的一部分。安全组和防火墙规则构成了一种灵活的方式，为 EC2 实例提供基本的安全保障，实例本身必须辅以适当的安全配置。

4. 高级计算服务

EC2 实例和 AMI 构成了建立 IaaS 云计算的基本模块。此外，亚马逊网络服务提供了更加完善的服务，使得应用程序和执行基于 MapReduce 应用程序的计算平台易于封装和部署。

（1）AWS CloudFormation

AWS CloudFormation 是一个具有 EC2 特征的实例的简单部署模型的扩展。CloudFormation 引入了模板的概念。模板是 JSON 格式的文本文件，描述在 EC2 运行一个应用程序或服务及它们之间的关系所需要的资源。CloudFormation 可以很容易地显式链接 EC2 实例并作为它们之间的依赖关系。模板提供了一个简单的声明方式来构建复杂的系统，整合 EC2 实例与其他 AWS 服务，如 S3、SimpleDB、SQS、SNS、Route 53、Elastic Beantalk 等。

（2）AWS 弹性 Beanstalk

320 AWS 弹性 Beanstalk 是一个在 AWS 云上打包和部署应用程序的简单方法。该服务可以简化供应实例和部署应用程序代码的过程，并提供适当的访问权限。目前，这项服务仅适用于 Java/Tomcat 技术栈并发的 Web 应用程序。开发人员可以方便地将 Web 应用程序打包成一个 WAR 文件，并使用 Beanstalk 自动将其部署在 AWS 云上。

相对于该自动云部署的其他解决方案，Beanstalk 简化了繁琐的任务，不移除用户的访问能力，接管了底层的 EC2 实例。EC2 实例构成应用程序在其上运行的虚拟基础设施。对于 AWS CloudFormation，AWS Elastic Beanstalk 提供了在云上部署应用程序的高级方法，它不要求用户指定 EC2 实例及其依赖方面的基础设施。

（3）亚马逊弹性 MapReduce

亚马逊弹性 MapReduce 为 AWS 用户的 MapReduce 应用提供云计算平台。它采用 Hadoop 作为 MapReduce 的引擎，部署在 EC2 实例组成的虚拟基础架构上，并使用亚马逊 S3 实现存储需求。除了支持连接到 Hadoop（Pig、Hive 等）的所有的应用程序栈，弹性 MapReduce 允许用户根据自己的需要动态地估计 Hadoop 集群的大小，以及选择 EC2 实例的适当配置来组成集群（小型、超大内存、超强 CPU、集群计算和集群 GPU）。在这些服务之上，弹性 MapReduce 还提供基本的 Web 应用程序，让用户快速运行数据密集型应用程序而无需编写代码。

9.1.2　存储服务

AWS 提供了一系列的数据存储和信息管理服务，其核心服务由亚马逊简单存储服务（S3）表示。S3 是一个分布式对象存储，允许用户将信息以不同的格式存储。S3 的核心部件有两个：桶和对象。桶代表存储对象的虚拟容器，对象代表实际存储的内容。对象可以通过元数据不断充实，元数据用来为所存储的内容标记附加信息。

1. S3 核心概念

顾名思义，S3 的设计是为了提供一个简单的存储服务，通过一个表征状态转移（Representational State Transfer，REST）的接口进行访问，非常类似分布式文件系统，但一些重要的差别使基础设施更高效：

- 存储器组织为两层结构。S3 将存储空间组织成不能进一步分割的桶。这意味着它不能通过创建目录或其他类型的物理分组在桶中存储对象。尽管如此，S3 对命名对象限制很少，允许用户模拟目录，并创建逻辑分组。
- 存储的对象不能像标准文件那样操作。S3 为对象提供的存储基本不随时间变化。因此，它不允许重命名、修改或重新定位对象。一旦对象添加到桶，其内容和位置是 321
不变的，改变它的唯一方法是从存储中删除该对象并重新添加。
- 内容不能立即供用户使用。S3 的主要设计目标是提供一个最终一致的数据存储。其结果是，因为它是一个大型的分布式存储设施，所以更改不会立即表现出来。例如，S3 在全球范围内使用复制来提供冗余和高效率的服务对象，这种做法导致了延迟，当在存储中添加一个对象时，特别是大对象时，不能在全球范围内立即使用。
- 请求偶尔会失败。由于采用大型分布式基础架构管理，对对象的请求可能偶尔会失败。在一定条件下，S3 可以决定通过返回一个内部服务器错误而放弃一个请求。因此，预计在日常操作中会有一个小故障率，通常不会是持久的故障。

S3 访问接口提供 RESTful 网络服务，即对存储器执行的所有操作以 HTTP（GET、PUT、DELETE、HEAD 和 POST）请求的形式进行，根据它们的地址元素而执行不同的操作。一般来说，PUT/POST 请求用于在存储中增加新的内容，GET/HEAD 请求用于检索内容和信息，DELETE 请求用于删除和链接它们的元素或信息。

（1）资源命名

桶、对象和附加的元数据通过一个 REST 接口进行访问。因此，它们由 s3.amazonaws.com 域中的统一资源标识符（URI）的表示。然后，所有操作的执行被表示为指向 URI 请求的实体。

亚马逊提供二种不同方式来表示桶地址：

- 标准形式：http://s3.amazonaws.com/bukect_name/。桶名称作为域名 s3.amazonaws.com 的路径组成部分。这个命名约定可使用的字符限制较少，所有的字符都可以在路径组成中使用。
- 子域名形式：http://bucketname.s3.amazon.com/。另外，还可以使用子域名 s3.amazonaws.com 引用桶。用这种形式表示桶名称时，该名称必须遵守下列规则：
  - 字符长度 3 ～ 63。
  - 只能包含字母、数字、英文句点和英文破折号。
  - 以字母或数字开头。
  - 至少包含一个字母。
  - 以英文破折号开头或结尾，或者是空字符串，句点之间没有碎片。

  使用时这种形式等同于之前的形式，但首选是这种形式，因为对表示存储在 S3 中的资源的地理位置更有效。
- 虚拟主机形式：http://bucket-name.com/。亚马逊还允许资源引用自定义的 URL。这是通过输入一个 CNAME 记录到指向桶 URI 的子域名形式的 DNS 来实现的。 322

由于 S3 逻辑上组织为扁平的数据存储，所有的桶都在 s3.amazonaws.com 域下进行管理。因此，桶的名称必须在所有的用户间是唯一的。

对象指的是给定桶的当地资源。因此，它们总是显示为一个 URI 的资源分量的一部分。桶可以表示为三种不同的方式，对象间接地继承这种灵活性：

- 标准形式：http://s3.amazonaws.com/bukect_name/object_name。
- 子域形式：http://bucket-name/s3.amzonaws.com/object_name。
- 虚拟主机形式：http://bucket-name.com/object_name。

除了用“?”号将URI资源路径从用户请求的参数集中分离出来，桶名后符号“/”之后所有的字符构成对象的名称。例如，表示部分对象名称的路径分隔符在桶存储中没有相应的物理布局。尽管如此，它们仍然可以用来创建与目录类似的逻辑分组。

最后，是关于一个给定对象的具体信息，比如它的访问控制策略或桶中定义的服务器日志设置，可以使用特定的参数来引用。更确切地说：

- 对象ACL：http://s3.amazonaws.com/bukect_name/object_name?acl。
- 桶服务日志：http://s3.amzonaws.com/bucket_name?logging。

对象元数据不能通过一个特定的URI直接访问，而要通过在URL的请求中添加属性来访问，添加的属性不属于标识符的一部分。

（2）桶

桶是对象的容器。它可以看作是一个托管在S3分布式存储的虚拟驱动器。为用户提供可以添加对象的扁平存储方式。桶是S3存储架构的顶层元素，不支持嵌套。也就是说，它不可能创建“子桶”或其他类型的物理分割。

桶位于一个特定的地理位置，桶的复制是为了实现更好的容错能力和内容分布。用户可以选择创建桶的位置，默认情况下在亚马逊的美国数据中心创建。一旦创建了桶，所有属于该桶中的对象将被存储在桶的相同可用区域。用户创建桶的方式是发送一个PUT请求到http://s3.amazonaws.com/，并加上桶的名称和其他信息。如果想指定可用区，还可加上首选位置信息。通过发送指定桶名称的GET请求获得桶内容的列表。桶一旦创建，不能重命名或迁移。如果有必要的话，只能删除后再重新创建桶。通过DELETE请求执行删除桶，当且仅当桶为空时，才可以成功删除。

（3）对象和元数据

对象构成存储在S3中的内容元素。用户存储文件或向S3发送文本流都表示对象内容。
323 对象通过存储内容的桶内的唯一名称进行标识。该名称用UTF-8编码，不能超过1024个字节，允许使用几乎任何字符。由于桶不支持嵌套，所以甚至允许字符作为路径分隔符。这实际上弥补了无结构化的文件系统的缺憾，因为可以通过适当命名的对象模拟目录。

用户通过指定对象名称、桶名称、内容及其他属性的PUT请求创建一个对象。一个对象的最大容量为5GB。一旦对象创建，不能被修改、重命名或移动到另一个桶。可以通过GET请求检索对象，通过DELETE请求删除对象。

对象可以用元数据标记，通过PUT请求的属性传递。该属性通过GET请求或HEAD请求来检索，检索结果只返回该对象的元数据而无内容。元数据由系统和用户定义。S3使用系统定义的元数据控制与对象的交互；用户定义的元数据对用户更有意义，每个元数据属性可以存储2KB数据，属性表示为键－值对的字符串。

（4）访问控制和安全性

亚马逊S3通过访问控制策略（ACP）访问桶和对象。ACP是一组授予权限，连接到一个由XML配置文件表示的资源。一个策略允许定义多达100个访问规则，每个规则可以授权给允许授权者。目前，有五种可用的不同权限：

- 读。允许被授权者检索一个对象的内容及其元数据，列出桶的内容，获取其元数据。
- 写。允许被授权者将对象添加到一个桶并进行修改和删除。
- 读访问控制策略。允许被授权者读资源的访问控制策略。
- 写访问控制策略。允许被授权者修改资源的访问控制策略。
- 完全控制。授予上述所有权限。

被授权者可以是单个用户或组。通过在S3注册时的规范的ID或电子邮件地址来识别用户。对于组而言，只有三个可能的选项：所有用户，身份验证的用户和日志传送用户[⊖]。

资源一旦创建，S3只能将默认ACP授予具有完全控制权限的主人。通过使用资源的URI请求时加上“?acl”可以改变ACP。GET方法可以检索ACP，PUT方法上传一个新的ACP来取代现有的ACP。或者，当创建资源时可以使用预定义的一组名为缮发策略的权限设置ACP。这些策略是S3资源最常见的访问模式。

ACP提供一套强大的规则来控制S3用户对资源的访问。但非验证用户不能详细区分，他们被视为一组。这种情况下，为了提供更细粒度的区分，S3允许定义签名的URI，提供一个临时的访问令牌，授权在有限的时间内的所有资源访问请求。 324

（5）高级功能

除了桶、对象和ACP的管理，S3还提供了其他有用的附加功能，包括服务器访问日志和集成比特流（BitTorrent）文件共享网络。

服务器访问日志让桶所有者获得有关桶及其所有包含对象所提出的请求的详细信息。默认情况下，此功能处于关闭状态；若要激活，需要发出一个后续“?logging”的PUT请求到桶的URI。该请求应包含一个指定目标桶的XML文件，XML文件中保存日志文件和文件名前缀。到同一个URI的GET请求可以让用户检索桶中现有的日志配置。

第二个有趣的功能是可以把S3对象公开到比特流网，从而允许使用BitTorrent协议下载存储在S3中的文件，方法是在S3对象的URI后附加“?torrent”。要实际下载对象，它的ACP必须授予每个人读权限。

2. 亚马逊弹性块存储

亚马逊弹性块存储（EBS）允许AWS用户提供EC2实例，以卷的形式持久存储，在实例启动时安装卷。EBS可为实例提供高达1 TB的空间，并通过一个块设备接口进行访问，从而使用户能够根据它们连接的（原始存储、文件系统或其他）实例的需要对其进行格式化。EBS卷的内容伴随着实例的生命周期，并一直持续到S3。EBS卷可以被克隆，用作引导分区，并构成持久存储，因为它们依赖于S3，并且内容可以进行增量快照。

EBS卷通常存储在EC2实例的同一可用性区域内，这将利用它们的最大化I/O性能。另外，也可以连接位于不同可用区的卷。一旦安装为卷，其内容根据操作系统的要求在后台延迟加载。这降低了进入网络的I/O请求的数量。卷的镜像不能在实例之间共享，但可以从镜像创建多个（独立的）活动卷。此外，可以将多个卷连接到单个实例或从一个给定的快照建立卷，如果格式化文件系统允许，还可以修改卷的大小。

卷的相关费用包括在S3中所占的存储量和对卷执行I/O请求的数量所产生的成本。目前，亚马逊的收费标准是：对分配的存储空间，0.10美元/GB/月；对卷，每1万个请求0.10美元。

⊖ 这里，组标识一个特定的组帐户，自动处理执行桶的访问日志。

3. 亚马逊弹性缓存

弹性缓存实现了一种基于EC2实例的集群弹性内存缓存。通过Memcached兼容协议，可从其他EC2实例实现快速数据访问，因此，基于这种技术的现有应用程序无需修改就可透明地迁移到弹性缓存。

弹性缓存是基于运行缓存软件的EC2实例的集群，通过Web服务提供。一个弹性缓存
325 集群可以根据客户端应用程序的需求进行动态调整。此外，自动补丁管理、故障检测和缓存节点的恢复等功能支持缓存集群持续运行，而无需AWS用户的管理。仅当需要的时候，用户才弹性地改变群集大小。弹性缓存节点根据EC2的成本核算模式定价，由于使用安装此实例的缓存服务不同，因此会有一点小的价格差异。不同类型的实例可供用户选择，表9-3给出了这些实例和定价。

表9-3显示的价格是在2011～2012年期间亚马逊的产品价格，内存的容量表示考虑系统软件开销后的可用内存。

表9-3　亚马逊EC2（按需）缓冲实例特征（2011～2012）

	实例类型	ECU	平台	内存	I/O能力	价格（美国东部，美元/小时）
标准实例	小型	1(1×1)	64 bit	1.3 GB	中等	$0.095
	大型	4(2×2)	64 bit	7.1GB	高	$0.380
	超大型	8(4×2)	64 bit	14.6GB	高	$0.760
超大内存实例	超大型	6.5(2×3.25)	64 bit	16.7GB	高	$0.560
	两倍超大型	13(4×3.25)	64 bit	33.8GB	高	$1.120
	四倍超大型	26(8×3.25)	64 bit	68GB	高	$2.240
超强CPU实例	超大型	26(8×3.25)	64 bit	6.6GB	高	$0.760

4. 结构化存储解决方案

企业应用程序往往依赖于数据库来存储结构化数据、索引数据，并对其执行分析。即使最近已提出了更具扩展性和轻量级解决方案，传统上，RDBMS一直是大量应用程序通用的后台数据。亚马逊提供了三种不同形式的结构化存储服务的应用程序：预先配置的EC2 AMI，亚马逊关系型数据存储和亚马逊简单数据库。

（1）预先配置的EC2 AMI

预先配置的EC2 AMI是预定义的模板，特征是安装一个指定的数据库管理系统。从这些AMI创建的EC2实例可通过持久存储的EBS卷来完成。可用的AMI包括IBM DB2、Microsoft SQL Server、MySQL、Oracle、PostgreSQL、Sybase和Vertica。实例根据EC2的成本模式按小时定价。该解决方案给EC2用户增加了管理负担，必须自己配置、维护和管理关系数据库，但提供了最多种产品以供选择。

（2）亚马逊RDS

RDS是关系数据库服务，依赖于EC2基础设施，由Amazon.Developers管理，用户不必担心配置高可用性的存储、设计故障切换的策略或更新具有最新补丁的服务器。此外，该服务为用户提供了自动备份、快照、时间点恢复以及执行复制的能力。通过AWS控制台或一个特定的Web服务，这些服务和常见的数据库管理服务都可用。有两个关系数据库引擎可供选择：MySQL和Oracle。

326 RDS的两个关键的高级功能是multi-AZ部署和读副本。第一个功能为用户提供了

RDBMS 解决方案的故障转移的基础设施。高可用性解决方案是：一旦主服务停止运行，就激活在不同可用地区服务的同步备用副本。第二个功能针对那些严重依赖于数据库读取的应用程序，为用户提供了更高的性能。在这种情况下，亚马逊部署主服务副本，它仅可用于数据库中读取，从而减少了服务的响应时间。

表 9-4 给出了 2011 ~ 2012 年期间两个高级功能的服务价格，以及按需实例的成本细节。也可以通过在打折时提前支付来使用长期预留（一至三年）的实例。

表 9-4 亚马逊 EC2（按需）实例特征（2011 ~ 2012）

	实例类型	ECU	平台	内存	I/O 能力	价格（美国东部，美元 / 小时）
标准实例	小型	1(1 × 1)	64 bit	1.3 GB	中等	$0.11
	大型	4(2 × 2)	64 bit	7.1GB	高	$0.44
	超大型	8(4 × 2)	64 bit	15GB	高	$0.88
超大内存实例	超大型	6.5(2 × 3.25)	64 bit	17.1GB	高	$0.65
	双倍大型	13(4 × 3.25)	64 bit	34GB	高	$1.30
	四倍大型	26(8 × 3.25)	64 bit	68GB	高	$2.60

相对于以前的解决方案，用户不需负责管理、配置和维护数据库管理软件，这些操作都是通过 AWS 执行的。另外，RDS 还简化了对服务器弹性管理的支持。因此，对基于 Oracle 和 MySQL 引擎的应用程序迁移到 AWS 的基础设施，且需要一个可扩展的数据库解决方案的情况，该方案是最佳解决方案。

（3）亚马逊简单数据库

亚马逊简单数据库用于数据不需要完全关系模型的应用程序，是一种轻量级的、高度可扩展的、灵活的数据存储解决方案。简单数据库支持半结构化数据，基于域、项和属性的概念。与关系模型相比，该模型对输入数据的结构限制较少，在查询大量数据方面具有较高性能。而亚马逊 RDS 服务对 AWS 用户的数据存储无需进行配置、管理和高可用性设计。

327

简单数据库使用域作为顶层元素来组织数据存储，这些域大致相当于关系模型中的表。但与表不同，域允许项具有不同的列结构，因此，每个项表示为一个键 – 值对属性的集合。每个域都可以增至 10 GB 数据，默认情况下单个用户可以分配到最多 250 个域。客户端可以创建、删除、修改、并进行域的快照，可以插入、修改、删除和查询项和属性，也支持批量插入和删除操作。查询数据功能是该模型最重要的功能之一，而 select 子句支持以下测试操作符：=，!=，<，>，<=，>=，like，not like，between，is null，is not null 和 every( )。下面是关于如何查询数据的简单例子：

```
select from domain_name where every(attribute_name)= 'value'
```

此外，select 操作符可以超出单个域的边界查询，从而使用户能够有效地查询大量数据。

为了有效地为 AWS 用户提供可扩展性和容错服务，简单数据库中实现了一个宽松的约束模型，从而产生最终一致的数据。最终一致是指非常短时间内对同一数据的多个访问可能读取的值不相同，但经过一段时间最终会相同。这是因为简单数据库在一次更新期间，数据在后台传播，且不锁定数据的所有副本。因此，存在一个短暂期间，在不同的客户端可以访问具有相同数据不同值的不同副本。这种做法具有很强的可伸缩性，只有轻微缺陷，但也是合理的，因为 SimpleDB 应用方案的主要特点是数据查询和索引操作。也可以改变默认情况，并确保在更新过程中锁定所有读取操作。

尽管简单数据库不是一个事物模型，但它允许客户端表示有条件的插入或删除，当有多个写入时，可以防止丢失更新。此时，当且仅当条件验证有效时才会执行该操作。这种情况可以用来检查项的属性的预先存在的值。

表 9-5 给出了 2011 ~ 2012 年间简单数据库服务的数据传输定价方案。无论是数据传输服务还是存储数据服务都需要收费，但 AWS 网络中的数据传输是不收取费用的。此外，对用户的使用机器也要收费。每月的前 25 个简单数据库实例是免费的，之后，按小时收费（在美国东部地区每小时 0.140 美元）。

表 9-5　亚马逊简单数据库的数据传输费用（2011 ~ 2012）

	实例类型	价格（美国东部，美元）		实例类型	价格（美国东部，美元）
数据流入			数据流出	下一个 100TB/ 月	$0.070
	所有的数据流入	$0.000		下一个 350TB/ 月	$0.050
				下一个 524TB/ 月	特殊设定
数据流出	第一个 GB/ 月	$0.000		达到 4PB/ 月	特殊设定
	达到 10TB/ 月	$0.012		超过 5PB/ 月	特殊设定
	下一个 40TB/ 月	$0.090			

如果将这个成本模式与一个典型的 S3 作比较，很明显 S3 用于存储大对象更便宜。这有助于区分简单数据库和 S3 的本质：前者适合小对象的半结构化数据快速访问，而不是用于大对象的长期存储。

5. 亚马逊 CloudFront

CloudFront 在亚马逊分布式存储基础设施之上实现内容交付网络。它按照一定策略利用位于全球范围的边界服务器集合，更好地满足静态和流媒体网页内容请求，尽可能地减少传
328 输时间。

AWS 为用户提供简单的 Web 服务 API 来管理 CloudFront。为了实现 CloudFront 内容的可用性，需要创建一个分发来标识原始服务器，其中包含正在分发内容的原始版本，它由一个 DNS 域下的 Cloudfront.net 域名（即 my-distribution.Cloudfront.net）表示。另外，也可以给定分发一个域名。一旦创建了分发，就可以确定分发名称，CloudFront 引擎将请求重定向到最近的副本，如果已选定的边界服务器上的内容没有找到或已过期，最终将从源服务器上下载原始版本。

通过 CloudFront 交付的内容是静态的（HTTP 和 HTTPS）或流（实时消息协议或 RMTP）。托管分发内容的原始副本的源服务器可以是一个 S3 的桶，一个 EC2 实例或一个亚马逊网络外部的服务器。用户可以限制只有一个或几个可用的协议访问分发，也可以设置访问规则进行更精细的控制。还可以从分配中删除无效内容或在过期之前迫使其更新。

表 9-6 提供了 2011 ~ 2012 年期间的价格细目。请注意，CloudFront 比 S3 便宜。这反映了其用途的不同：CloudFront 用来优化经常下载的、非常流行的内容分发，不仅是从亚马逊网络，还有可能从全球网络下载。

表 9-6　亚马逊 CloudFront 按需价格（2011 ~ 2012）

	定价项目	联合国	欧盟	香港和新加坡	日本	南美
需求	每 10000 个 HTTP 请求	$0.0075	$0.0090	$0.0090	$0.0095	$0.0160
	每 10000 个 HTTPS 请求	$0.0100	$0.0120	$0.0120	$0.0130	$0.0220

（续）

	定价项目	联合国	欧盟	香港和新加坡	日本	南美
区域数据流出	第一个 10TB/ 月	$0.120/GB	$0.120/GB	$0.190/GB	$0.201/GB	$0.250/GB
	下一个 40TB/ 月	$0.080/GB	$0.080/GB	$0.140/GB	$0.148/GB	$0.200/GB
	下一个 100TB/ 月	$0.060/GB	$0.060/GB	$0.120/GB	$0.127/GB	$0.180/GB
	下一个 350TB/ 月	$0.040/GB	$0.040/GB	$0.100/GB	$0.106/GB	$0.160/GB
	下一个 524TB/ 月	$0.030/GB	$0.030/GB	$0.080/GB	$0.085/GB	$0.140/GB
	下一个 4PB/ 月	$0.025/GB	$0.025/GB	$0.070/GB	$0.075/GB	$0.130/GB
	大于 5PB/ 月	$0.020/GB	$0.020/GB	$0.060/GB	$0.065/GB	$0.125/GB

9.1.3 通信服务

Amazon 提供设施来组织和促进 AWS 基础架构中现有的应用程序和服务之间的通信。这些设施可以分为两大类：虚拟网络和消息。

1. 虚拟网络

虚拟网络由一组服务组成，允许 AWS 用户控制计算和存储服务以及它们之间的连通性。在基础设施方面，亚马逊虚拟网络（VPC）和亚马逊直接连接提供连接解决方案，Route 53 在命名方面使连接更容易。 329

亚马逊 VPC 提供了很大的灵活性，可在其基础设施内部和外部创建虚拟专用网络。服务供应商设计了大部分常用网络场景的模板，或者配置一个完全可定制的网络服务。设计的模板包括公共子网、独立网络、通过网络地址转换（NAT）接入因特网的专用网络，以及包含 AWS 资源和私有资源的混合网络。还可通过使用身份访问管理（IAM）服务控制不同的服务（EC2 实例和 S3 桶）之间的连通性。2011 年，亚马逊 VPC 的成本是每小时连接 0.50 美元。

亚马逊直接连接允许 AWS 用户创建在用户私网和亚马逊直接连接位置之间的称为端口的专用网络。这种连接可以进一步分割多个逻辑连接，并提供访问 Amazon 基础设施的公共资源。采用直接连接相比其他解决方案的优点是用户位置和直连位置之间的连接的一致性能。该服务与 EC2、S3 和亚马逊 VPC 等其他服务兼容，可以在亚马逊网络与外部世界之间需要高带宽的情况下使用。美国只有两个可用端口，但用户可以利用外部供应商提供的高带宽端口。有两种不同的带宽可以选择：1Gbps，售价为每小时 0.30 美元；10Gbps，售价为每小时 2.25 美元。入站流量是免费的，对外输出流量的售价为每 GB 0.02 美元。

亚马逊 Route 53 实现动态域名解析服务，允许通过不同于 amazon.com 域的域名来获取 AWS 资源。利用亚马逊 DNS 服务器的大型全球分布式网络，AWS 用户可以公开 EC2 实例或 S3 存储桶作为其属性域内的资源，为此，亚马逊 DNS 服务器成为权威[⊖]。EC2 实例很可 330
能比物理机器更加动态，S3 桶也可能只在有限的时间内存在。为了处理这种不稳定性，当在 EC2 上启动实例或在新桶中创建 S3 时，该服务使得 AWS 用户将名称动态映射到资源。

⊖ DNS 服务器负责解析名称和对应的 IP 地址。由于 DNS 服务器实现分布式数据库，所以没有一个单一的全局控制。单个 DNS 服务器仅有域名和 IP 地址之间的所有映射，但它有其中的一小部分直接映射。因此，这样的 DNS 服务器对这些域名来说是权威，因为它可以直接解析域名。解析其他的域名则需要联系最近的权威 DNS。

通过与 Route 53Web 服务进行交互，用户可以管理一组托管区，它表明用户域由服务控制，然后通过它编辑资源。目前，单个用户最多可以有 100 个区。成本核算模型包括一个固定金额（每月每区 1 美元）和按托管机处理查询的数量计费的动态组件（一个月中，前十亿的查询 0.50 美元每万次，超过十亿的查询，每亿查询 0.25 美元）。

2. 消息

消息服务构成了利用 AWS 功能连接应用程序的下一步。三种不同类型的消息服务是亚马逊简单队列服务（SQS），亚马逊简单通知服务（SNS）和亚马逊简单电子邮件服务（SES）。

亚马逊 SQS 是非连接模型，通过消息队列的方式在应用程序之间交换消息。SQS 是在 AWS 基础设施中提供的功能。使用 AWS 控制面板或直接使用底层的 Web 服务 AWS，用户可以创建无限数量的消息队列，并配置它们来控制访问。应用程序可以将消息发送到它们有权访问的任何队列。这些消息在有限的一段时间内安全地、有冗余地存储在 AWS 基础设施，可以通过其他（授权）的应用程序访问来访问它们。当一个消息被读取时，它将保持锁定，以避免来自其他应用程序的恶意处理，设定的时间结束后将自动解锁。

亚马逊 SNS 提供了用于连接异构应用程序的发布 – 订阅方法。亚马逊 SQS 必须不断轮询指定的队列以处理一个新消息，当有感兴趣的新内容时，亚马逊 SNS 会通知应用程序。此功能可以通过 Web 服务获取，这样 AWS 用户可以创建一个其他应用程序可以订阅的主题。无论何时，只要应用程序就一个给定的主题发布内容，订阅者便可自动接到通知。该服务为订阅者提供不同的通知模型（HTTP/HTTPS、email/email JSON 和 SQS）。

亚马逊 SES 为 AWS 用户提供充分利用 AWS 基础设施的可扩展的电子邮件服务。一旦用户注册了这项服务，他们必须提供一个电子邮件地址，SES 用来发送代表用户的邮件。要激活该服务，SES 发送一封电子邮件，以确认给定的地址并为用户提供必要的激活信息。一经核实，将为用户提供一个 SES 沙箱测试服务，用户可以请求访问产品版本。使用 SES，
[331] 可以通过指定的邮件信头和多用途因特网邮件扩展（MIME）类型来发送 SMTP 兼容的电子邮件或原始邮件。电子邮件排队等待发送，若失败会通知用户。SES 还提供了多种统计数据，帮助用户提高与客户沟通的电子邮件活动的有效性。

关于成本核算，三项服务都不要求最低保证，但都是基于即付即用的模式。目前对用户不收费，直到他们达到一个最低门槛。此外，数据流入不收费，但数据流出按范围收费。

9.1.4　其他服务

除了计算、存储和通信服务，AWS 还提供一系列服务允许用户利用聚合的服务。相关的两个服务是亚马逊 CloudWatch 和亚马逊灵活支付服务（FPS）。

亚马逊 CloudWatch 提供了一种全面的统计数据服务，帮助开发人员理解和优化托管在 AWS 上的应用执行。CloudWatch 从 EC2、S3、简单数据库、CloudFront 等几个 AWS 服务收集信息。开发人员使用 CloudWatch 可以看到在 AWS 上租用的服务及其使用情况的详细分析，并能设计出更高效和节约成本的应用。CloudWatch 的早期服务只通过认购提供，但现在已免费提供给所有 AWS 用户。

亚马逊 FPS 的基础设施允许 AWS 用户利用亚马逊的计费设施向其他 AWS 用户出售商品和服务。使用亚马逊 FPS，开发人员不必设立其他的付款方式，可以通过一个计费服务向用户收费。可通过 FPS 的支付模式，包括一次性付款、延迟付款、定期付款、通过签约和基于使用的服务，交易和综合多种支付模式。

9.1.5 总结

亚马逊利用大型分布式AWS基础架构，提供用于开发、部署、管理云计算系统的一整套服务。开发人员可以使用EC2控制和配置托管在云上的计算基础设施。如果不需要完全控制计算栈，可以利用其他服务，如AWS CloudFormatio，弹性Beanstalk或弹性MapReduce，托管在AWS云的应用程序可以利用S3、简单数据库或其他存储服务来管理结构化和非结构化数据。这些服务主要用于存储，其他服务，如亚马逊SQS、SNS和SES，为AWS云内外动态连接应用程序提供解决方案。与AWS应用程序的网络连接由亚马逊VPC和亚马逊直接连接处理。

9.2 谷歌 AppEngine

谷歌AppEngine是一个PaaS的实现，为开发和托管可扩展的Web应用程序提供服务。AppEngine基本上是一个分布式和可扩展的运行时环境。它利用谷歌的分布式基础设施扩展应用程序，面对大量的请求，通过分配更多的计算资源来平衡它们之间的负载。运行时由一组服务完成，让开发人员能够设计和实施很容易在AppEngine上扩展的应用程序。开发者 332
可以使用Java、Python、Go来开发应用程序。Go是由谷歌公司开发的简化Web应用程序开发的新编程语言。谷歌资源的使用和服务质量由AppEngine度量，在用户的应用程序完成免费限额之后向他们收费。

9.2.1 架构和核心概念

AppEngine是一个开发可扩展的Web访问应用程序的平台（见图9-2）。该平台逻辑上分为四个主要部分：基础设施，运行时环境，底层存储，以及一套用于开发可扩展应用程序的服务。



图 9-2 谷歌 AppEngine 平台架构

1. 基础设施

AppEngine托管Web应用程序，它的主要功能是有效地服务用户请求。要做到这一点，AppEngine的基础设施利用了谷歌数据中心内很多服务器的可用优势。对每个HTTP请求，

AppEngine 定位托管处理请求的应用程序的服务器，评估其负载，如果需要，分配额外的资源（如服务器）或重定向请求到现有服务器。应用程序的特殊设计简化了基础设施的工作，到相同的应用程序的请求之间不会隐式地维持任何状态信息，每个请求可重定向到任何托管目标应用程序的服务器，甚至分配一个新的服务器。

333 基础设施还负责监控应用程序性能和收集账单的统计信息。

2. 运行时环境

运行时环境表示托管在 AppEngine 上的应用程序的执行上下文。参照 AppEngine 上的基础设施代码，运行时环境始终活跃和运行，在请求处理程序开始执行运行时就存在，处理程序完成后立即终止。

（1）沙箱

运行时环境的主要职责是为应用程序提供孤立和受保护的上下文环境，执行时不会对服务器构成威胁，也不受其他应用程序的影响。换句话说，它为应用程序提供一个沙箱。

目前，AppEngine 只支持管理或解释型语言开发的应用程序，要求运行时将其代码翻译转换成可执行的指令。因此，沙箱是通过禁用一些通常默认实现的共同特征，以修改应用程序的运行时来实现的。如果应用程序试图执行任何一个潜在有害的操作，沙箱将抛出异常，并中断执行。沙箱不允许的操作包括：写入服务器文件系统，使用邮件 UrlFetch 和 XMPP 以外的方式通过网络访问计算机，在请求、排队任务、cron 作业范围之外执行代码，处理时间超过 30 秒的请求。

（2）支持运行时

目前，开发 AppEngine 上的应用可以使用三种不同的语言和相关技术：Java、Python 和 Go。

AppEngine 目前支持 Java 6，开发人员可以使用常见的工具在 Java 中开发 Web 应用程序，如 Java 服务器页面（JSP），以及使用 Java Servlet 标准的环境与应用交互。此外，对 AppEngine 的访问服务是由 Java 库提供的，针对一个给定的抽象层的特定提供者的实现，Java 库公开特定接口。开发人员可以通过 Java SDK 创建 AppEngine 上的应用程序，Java SDK 允许使用 Java 5 或 Java 6 开发应用程序，以及不超过该沙箱限制的任何 Java 库。

对 Python 的支持由一个优化的 Python 2.5.2 解释器提供。同 Java 一样，运行时环境支持 Python 标准库，但删除了有些可能执行有害操作的模块，并且在试图导入这样的模块或调用特定的方法时产生异常。为了支持应用程序的开发，AppEngine 提供了一组丰富的连接到 AppEngine 服务的应用程序库。另外，开发人员可以使用一个特定的 Python Web 应用框架，称为 webapp，简化了 Web 应用程序的开发。

Go 运行时环境允许用 Go 编程语言开发 AppEngine 上托管和运行的应用程序。目前 AppEngine 支持的 Go 版本是 r58.1。该 SDK 包括编译器，以及在 Go 中开发应用和 AppEngine 上的服务接口的标准库。与 Python 环境一样，一些功能已删除或将生成运行时
334 异常。此外，开发人员可以在其应用程序中包含第三方库，只要它们是用纯 Go 实现的。

3. 存储

AppEngine 提供多种类型的存储，根据数据的波动性而进行不同的操作。有三种不同级别的存储：存储器缓存，半结构化数据存储和静态数据长期存储。本节介绍数据存储和静态文件服务器的使用，在应用服务部分涉及存储器高速缓存。

（1）静态文件服务器

Web 应用程序由动态和静态数据组成。动态数据是应用程序逻辑与用户交互的结果。静态数据大多是构成应用程序的图形设计的组件（如 CSS 文件、纯 HTML 文件、JavaScript 文件、图像、图标和声音文件）或数据文件。这些文件可以在静态文件服务器托管，因为它们不经常修改。这种服务器对静态内容服务进行了优化，上传应用程序到 AppEngine 上时，用户可以指定动态内容应提供的服务。

（2）数据存储

数据存储是允许开发者存储半结构化数据的服务。该服务旨在通过扩展和优化以快速访问数据。数据存储可以看作是一个大对象数据库，其中存储能够由指定的键来检索的对象。键的类型和对象的结构可以变化。

对于一个由关系数据库支持的传统 Web 应用程序，数据存储对数据规则施加较少约束，但同时，没有实现一些关系模型的特征（如参照约束和连接操作）。经过认真分析 Web 应用程序数据使用模式，为获得更大的可扩展性和高效的数据存储，产生了这些数据存储的设计决策。数据存储的底层基础设施基于 Bigtable[93]，Bigtable 是冗余的、分布式和半结构化的数据存储，以表的形式组织数据（见 8.2.1 节）。

数据存储提供了高层次的抽象，简化了与 Bigtable 的交互。开发者通过实体和属性定义数据，而这些都转化成 Bigtable 里的表来存储和维护。一个实体构成存储的粒度级别，并确定定义存储数据的属性的集合。根据该服务支持的几个原始类型中的一种定义属性。每个实体关联一个键，由用户提供或由 AppEngine 自动创建。每个实体都有一个命名类型，AppEngine 用其通过 Bigtable 优化检索。虽然实体和属性看起来类似于 SQL 的行和表，但 [335]
还是有一些差别。相同种类的实体可能有不同的属性，并且具有相同名称的属性可能包含不同类型的值，属性可以存储相同值的不同版本。最后，键是不可变的元素，一旦建立，就不可以改变。

数据存储还提供了在数据上创建索引并更新一个事务上下文中的数据的功能。索引用于支持和加快查询速度。查询可以返回相同类型的零个或多个对象或简单地返回对应的键。它可以通过指定键或属性值条件来查询数据存储。返回的结果集可以通过按键值或属性值排序。尽管查询非常类似 SQL 查询，但其执行有本质不同。数据存储的设计可极速返回结果集，要做到这一点，需要事先知道对给定类型的所有可能的查询，因为每个查询存储单独的索引。该索引是上传应用程序到 AppEngine 时由用户提供的，并且通过开发服务器自动定义。当开发者测试应用程序时，服务器监控所有不同类型的对该模拟数据存储的查询，为它们创建一个索引。索引的结构保存在配置文件中，开发者上传应用程序之前可以进一步修改。利用预先计算的索引使得查询执行时间和所存储的数据的大小无关，仅受结果集的大小的影响。

为了保持存储的可扩展性和高速性，事务的实现是受限的。AppEngine 确保以原子方式执行对单一实体的更新。对相同实体的多个操作可以在事务的范围内进行。也能以原子方式更新多个实体，但仅当这些实体属于同一个实体组。实体创建时指定实体所属的实体组，且不能更改。关于并发，AppEngine 上使用了乐观并发控制：如果用户尝试更新一个已经被更新的实体，那么控制返回且操作失败。检索实体不会导致异常。

4. 应用服务

托管在 AppEngine 上的应用程序可以充分利用运行时环境所提供的服务。这些服务简

化了在 Web 应用程序执行的大多数普通操作：获取数据、账户管理、外部资源整合、信息和通信、图像处理和异步计算。

（1）UrlFetch

Web 2.0 已经推出组合 Web 应用的概念。把不同的资源放在一起，并组织成一个单一 Web 页面中的网格。网格是以不同方式产生的 HTML 片段，它们可以从远程服务器中直接得到，或者从 Web 服务中检索 XML 文档得到，或者可以由浏览器的嵌入及远程组件得到。所有这些例子的共同特点是该资源不在本地服务器，甚至不在同一管理域。因此，这是 Web 应用程序检索远程资源的基础。

沙箱环境不允许应用程序通过套接字打开任意多个连接，但它确实为开发人员提供通过 HTTP/HTTPS 以 UrlFetch 服务方式获取远程资源的能力。应用程序可以进行同步和异步 Web 请求，并将通过这种方式获得的资源集成到应用程序中的正常请求处理周期。UrlFetch 有趣的功能之一是可以设定请求的最后期限，使其在给定的时间内完成（或中止）。此外，执行这种异步请求允许应用程序继续执行，而在后台检索资源。UrlFetch 不仅用于将网格集
336 成到 Web 页面，还能根据分布式应用程序的 SOA 参考模型利用远程 Web 服务。

（2）存储器高速缓存

AppEngine 的数据存储为开发人员提供访问快速且可靠的存储服务。尽管如此，该服务的主要目的是作为一个可伸缩的和长期的存储器，其中的数据被冗余地保存到磁盘，以确保出现故障时数据的可靠性和可用性。对比其他解决方案，特别是频繁访问的对象，如每一个页面请求，本设计限制了存储速度。

AppEngine 通过存储器高速缓存提供缓存服务。这是一个分布式内存缓存，经过了优化以便快速访问，针对频繁访问对象，为开发人员提供一个易失性存储器。存储器高速缓存中实现的缓存算法将自动删除那些很少访问的对象。使用存储器高速缓存可以显著地减少数据访问的时间。开发人员可以构建自己的应用程序，首先在存储器高速缓存中查找每个对象，如果没找到，将从数据存储中检索并放入缓存中以备将来查询。

（3）邮件和即时消息

通信是 Web 应用程序的另一个重要方面。通常使用电子邮件跟进用户有关应用程序执行的操作。电子邮件也可以用来触发 Web 应用程序的活动。为了促进这些任务的执行，AppEngine 为开发者提供通过邮件来发送和接收邮件的能力。该服务允许代表应用程序发送电子邮件给特定的用户账户。它也可能包括几种附件类型和针对多个接收者。邮件以异步方式运行，一旦交付失败，可通过电子邮件将详细的错误通知给发送地址。

AppEngine 还提供了另一种与外部世界通信的方式：可扩展通信和表示协议（XMPP）。任何支持 XMPP 的聊天服务（如 Google Talk）都可以从 Web 应用程序发送和接收聊天消息，并用自己的地址标识。尽管聊天主要是用于人工交互的通信介质，但 XMPP 还可以方便地用于连接 Web 应用程序与聊天机器人，或实现一个小型管理控制面板。

（4）账户管理

Web 应用程序通常保存与用户交互定制的各种数据，这些数据通常在用户配置文件中并连接到一个账户。AppEngine 简化了账户管理，允许开发者利用谷歌账户进行管理。与服务的集成还允许 Web 应用程序将认证功能的实施转移到 Google 认证系统。

使用谷歌账户，Web 应用程序可以方便地以键 - 值对的形式存储配置文件设置，将它们附加到一个给定的谷歌账户，一旦用户进行身份验证便可迅速检索。对于自定义解决方

案，使用谷歌账户管理的用户只需要有一个谷歌账号，不需要任何其他操作。在谷歌 App 的企业环境中，使用谷歌账户开发 Web 应用尤其有利。在这种情况下，应用程序易于与其他所包含在谷歌 App 中的服务（和配置文件设置）集成。 337

（5）图像处理

Web 应用程序用图形渲染页面。用户往往需要一些简单的操作，如添加水印或使用简单的过滤器。AppEngine 允许应用程序通过图像处理进行图像缩放、旋转、镜像和增强，其他谷歌产品也使用此服务。图像处理主要处理轻量级图像，并且提高了速度。

5. 计算服务

Web 应用程序大多通过无所存在的 Web 页面来实现用户应用程序界面。大部分交互是同步进行的：用户浏览网页，并得到响应他们行动的即时反馈。这种反馈往往是一些 Web 应用程序计算的结果，实现了依期的逻辑以服务用户请求。有时这种方法是不适用的，例如，在计算耗时长或某些操作需要在给定时间点触发时。对于这种情形，一个好的设计应该是：一旦所需的操作完成就立即给用户提供反馈和通知。AppEngine 提供了附加的服务，例如任务队列和 Cron 作业，对关闭带宽或那些无法在 Web 请求的时限内完成的计算进行简化执行。

（1）任务队列

任务队列允许应用程序提交一个较晚执行的任务。这项服务对于那些耗时长的计算特别有用，即一个请求处理程序不能在最大响应时间内完成。该服务允许用户最多有 10 个队列，以可配置的速度执行任务。

事实上，一个任务是对一个给定 URL 的 Web 请求，队列通过将有效载荷作为一部分 Web 请求交给处理程序的方式来调用请求处理程序。请求处理程序的职责是完成“任务执行”，从队列的角度看是一个简单的 Web 请求。为了避免瞬时故障导致任务不能成功完成，一旦任务失败，队列可重新执行任务。

（2）Cron 作业

有时候计算的长度可能不是 Web 请求范围之内未进行操作的主要原因，而是需要操作在一天中的特定时间执行，这可能与 Web 请求的时间不一致。在这种情况下，可以通过使用 Cron 作业服务在特定的时间调度所需的操作。这项服务的工作方式类似任务队列，但不同之处是在给定的时间调用任务中的唤醒请求处理程序，且任务失败时不重新执行。这种行为可用于实施维护操作或定期发送通知。

9.2.2 应用程序生命周期

AppEngine 提供了应用程序生命周期中几乎所有阶段的功能支持：测试、开发、部署、监控。谷歌发布的 SDK 为开发者提供了这些任务所需的大部分功能。目前有两种软件开发 338
工具包可用于开发：Java SDK 和 Python SDK。

1. 应用程序开发和测试

开发人员可以在本地开发服务器上建立自己的 Web 应用程序。这是一个自包含环境，开发人员不必把应用上传到 AppEngine 上就能进行调试。开发服务器模拟 AppEngine 运行时环境，提供了数据存储、存储器高速缓存和 UrlFetch 的模拟实现，并通过 Web 应用程序利用其他服务。除了托管 Web 应用程序，开发服务器还包含了一套完整的监控功能，有利于描述应用程序的行为，特别是对数据存储服务的访问和查询。对于应用程序到 AppEngine

的部署，这是一个特别重要的特征。如前所述，AppEngine 为给定的应用执行的每个查询建立索引，以加快对相关数据的访问。这种能力通过应用程序执行的所有查询的先验知识而获得，而开发商在上传应用程序时就把这些知识提供给了 AppEngine。开发服务器分析应用程序的行为和运行，跟踪测试过程和开发中进行的所有查询，从而为将要建立的索引提供所需的信息。

（1）Java SDK

Java SDK 为开发人员提供使用 Java 5 和 Java 6 运行时环境构建应用程序的设施。另外，也可以使用谷歌 AppEngine 插件，在 Eclipse 开发环境中开发应用程序。它集成了功能强大的 Eclipse 环境中 SDK 的功能。使用 Eclipse 的软件安装程序，可以在 Eclipse 中下载和安装 Java SDK、谷歌 Web Tookit 和谷歌 AppEngine 的插件。这三个组件使开发人员能够为 AppEngine 编写强大而丰富的 Java 应用程序。

SDK 支持的应用程序通过使用 servlet 抽象来开发应用程序，servlet 是一个通用开发模式。servlet 和许多其他功能可用于构建应用程序。此外，开发人员可以使用 Eclipse Web 平台轻松地创建 Web 应用程序，Eclipse Web 平台提供了一组工具和组件。

该插件允许开发、测试以及在 AppEngine 上部署应用程序。其他任务（如应用程序日志检索）都可以使用命令行工具，命令行工具是 SDK 的一部分。

（2）Python SDK

Python SDK 允许用 Python 2.5 开发 AppEngine Web 应用程序。它提供了一个单独工具，称为谷歌 AppEngineLauncher，可在本地管理 Web 应用程序，并将它们部署到 AppEngine。该工具提供了方便的用户界面，列出了所有可用的 Web 应用程序，控制程序的执行，并与默认的代码编辑器集成来编辑应用程序文件。此外，启动器可访问一些重要的服务，如日志、SDK 控制面板和仪表板，来监控和分析应用程序。日志控制面板捕获所有应用程序运
339 行时记录的信息。控制面板 SDK 为开发人员提供一个 Web 界面，通过界面可以看到资源使用方式的应用程序配置文件。此功能特别有用，一旦应用程序部署在 AppEngine 上，便允许开发者预览应用程序的行为，并且可以通过运行时调整应用程序。

SDK 的 Python 实现还自带了一个集成 Web 应用的框架，名为 webapp，它包括一组简化了 Web 应用程序开发的模型、组件和工具，且执行一套统一的做法。它不是开发 Web 应用程序的唯一 Web 框架，还有几十个可用的 Python Web 框架。然而，由于沙箱环境的限制，它们都不能无缝地使用。webapp 框架已经被重新实现，并在 Python SDK 中可用，因此也可用于 AppEgnine。另一个 Web 框架 Django[⊖]也很好用。

SDK 是一组命令行工具，允许开发人员通过启动器以及更多的 shell 命令执行所有操作。

2. 应用程序的部署和管理

一旦应用程序完成开发和测试，就可以用一个简单的点击或命令行工具部署在 AppEngine 上。执行此操作之前，需要创建一个应用程序标识符，用于在 Web 浏览器键入地址 http://<application-id>.appspot.com 定位应用程序。另外，也可以将应用程序映射到注册的 DNS 域名。这对于商业开发特别有用，用户通过提供一个更合适的名称使应用程序可用。

⊖ www.djangoproject.com。

应用标识符是强制性的，它是应用程序与 AppEngine 进行交互的唯一标识。开发人员可以使用一个应用标识符来上传和更新应用程序。标识符是唯一的，同时也要符合域名命名规则。登录 AppEngine，选择“创建应用程序”选项，便可注册一个应用程序标识符。也可通过一个应用程序标题来描述应用程序，标题可以随着时间而改变。

一旦创建应用程序标识符，就可以在 AppEngine 上部署应用程序。这个任务可以使用相应的开发环境（谷歌 AppEngineLauncher 和谷歌 AppEngine 插件）或命令行工具来完成。应用程序一旦上传，不需要再做任何事情去实现程序的可用性。AppEngine 会负责一切。然后，开发人员可以通过使用管理控制面板管理应用程序。管理控制面板是用于应用程序监控的主要工具，让用户深入了解资源使用情况（CPU、带宽）和服务，以及其他功能的使用情况。另外，也可以管理一个单一应用的多个版本，选择其中一个进行发布，并且管理与其相关的计费问题。

9.2.3 成本模型

AppEngine 提供的限量免费服务每隔 24 小时重置一次。一旦应用程序已经由 AppEngine 测试和调整，便可建立一个缴费账户，按使用时间收费，获得更多的补贴。这使得开发人员可以明确为给定的应用分配适当的每日预算。

应用程序按计费配额、固定配额、每分钟配额度量。谷歌 AppEngine 使用这些配额以确保用户花费不会超过所分配的预算，且应用程序性能不受影响。计费配额是指每日配额，由应用程序管理员设置，通过分配给应用程序的每日预算定义。AppEngine 会确保应用程序不会超过这些配额。免费配额是计费配额的一部分，规定用户不收费的配额部分。固定配额是由 AppEngine 设置内部配额，确定基础设施的边界和定义应用程序可以在基础设施（服务和运行时）上执行的操作。这些配额一般都大于计费配额，并由 AppEngine 设置，以避免应用程序影响彼此的性能或基础设施超载。成本核算模型还包括每分钟配额，定义它是为了避免应用程序在非常有限的时间内消费所有存款、独占资源或为其他应用程序创建服务中断。

一旦应用程序对于一个给定的资源达到配额，该资源被耗尽，用户将无法使用该应用程序，直到配额得到补充。一旦资源耗尽，后续对该资源的请求将产生错误或异常。例如，CPU 时间和流入或流出带宽将向用户返回“HTTP 403”错误页面。其他资源和服务也将生成一个异常，这些异常在代码中被捕获，并向用户反馈更多的有用信息。

资源和服务的配额被组织成免费的默认配额和可计费的默认配额。对于这两种配额，定义日限制和最大速率。配额如何工作、配额的极限、向用户收取金额的详细说明参考 AppEngine 网站：http://code.google.com/appengine/docs/quotas.html。

9.2.4 结论

利用谷歌的基础设施，AppEngine 是一个开发可扩展的 Web 应用程序的框架。服务的核心部件是一个可扩展的沙箱运行时环境，用于执行应用程序，并实现大部分 Web 开发的共同特征的一系列服务，帮助开发人员构建易于扩展的应用程序。AppEngine 的特征元素之一是利用简单的接口，允许应用程序执行用于优化和设计可扩展性能的具体操作。基于这些模块，开发人员可以构建应用程序，需要时还可利用 AppEngine 扩展应用程序。

相对于传统的 Web 开发方法，丰富而强大的应用程序的实现需要创新思维和更多的努

340

力。开发人员必须熟悉 AppEngine 的功能，并且用符合 AppEngine 应用程序模型的方式实现所需的功能。

9.3　微软 Azure

微软 Windows Azure 是一个云操作系统，建立于微软数据中心的基础设施之上，并为开
341 发者提供一组用云技术构建应用程序的服务。服务范围从计算、存储、网络到应用程序的连接、访问控制以及商业智能。建立在微软技术之上的任何应用程序都可以使用 Azure 平台进行扩展，它将可扩展性功能集成到通用微软技术中，如微软的 Windows Server 2008、SQL Server 和 ASP.NET。

图 9-3 所示为 Azure 中所提供的服务概览。这些服务由 Windows Azure 管理门户管理和控制，它是 Azure 平台提供的所有服务的管理控制面板。本节将介绍 Azure 的主要服务的核心功能。

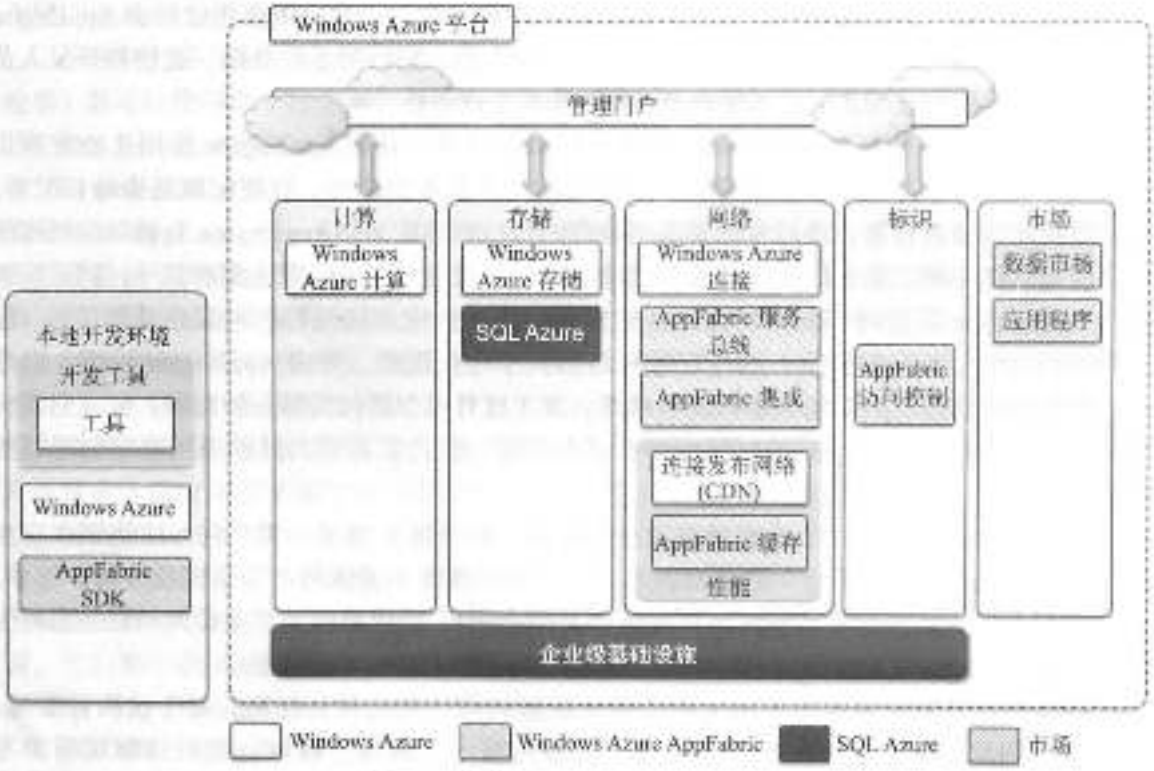


图 9-3　微软 Windows Azure 平台架构

9.3.1　Azure 核心概念

Windows Azure 平台由基础层和一组可用于构建可扩展应用程序的开发者服务组成。这些服务涵盖计算、存储、网络和身份管理，通过称为 AppFabric 的中间件绑在一起。这种可扩展的计算环境托管在微软数据中心内部，可通过 Windows Azure 管理门户访问。另外，开发人员可以在自己的机器上重新创建一个用于开发和测试的 Windows Azure 环境（功能有
342 限）。本节将概述 Azure 中间件及其服务。

1. 计算服务

计算服务是微软 Windows Azure 的核心组件，以角色抽象的方式发布。角色是定制特定计算任务的运行时环境，由 Azure 操作系统管理和按需实例化，以解决激增的应用程序需求。目前，有三种不同的角色：Web 角色、工作者角色和虚拟机（VM）角色。

（1）Web 角色

Web 角色旨在实现可扩展的 Web 应用程序。Web 角色代表了 Azure 的基础设施中部署 Web 应用程序的单元。它们都托管在 IIS 7 Web 服务器。IIS 7 Web 服务器是一个支持 Azure 基础设施的组件。当 Azure 中检测到一个给定的应用程序发出请求的高峰负荷，它为应用程序实例化多个 Web 角色，并通过负载均衡的方式在它们之间的分配负载。

由于 3.5 版中 .NET 技术原生支持 Web 角色，开发人员可以直接在 Visual Studio 中开发应用程序，在本地测试并上传到 Azure。可以开发 ASP.NET（ASP.NET Web 角色和 ASP.NET MVC 2 Web 角色）和 WCF（WCF 服务 Web 角色）等应用程序。由于 IIS 7 还通过 FastCGI 模块支持 PHP 运行环境，所以 Web 角色可用于在 Azure（CGI Web 角色）上运行和扩展 PHP Web 应用程序。未与 IIS 集成的其他 Web 技术仍然可以托管在 Azure（即 Apache Tomcat 上的 Java 服务器页面），但使用超过一个 Worker 角色的 Web 角色没有什么优势。

（2）工作者角色

工作者角色用来在 Azure 上托管普通的计算服务。它们可以快速提供计算能力或者托管不通过 HTTP 与外部世界进行通信的服务。对于工作者角色通常的做法是使用它们为 Web 角色开发的 Web 应用程序提供后台处理。

开发工作者角色就像是开发服务。Web 角色的计算由 HTTP 客户端（即浏览器）的交互触发，而工作者角色从实例创建开始就连续运行，直到被关闭。Azure SDK 为开发人员提供方便的 API 和库，允许连接角色和运行时所提供的服务，方便地控制它的启动以及通知托管环境的变化。对于 Web 角色，.NET 技术提供了对工作者角色的完整支持，但是在基于 Windows Server 栈上运行的任何技术都可以用来实现其核心逻辑。例如，工作者角色可用于托管 Tomcat 和服务基于 JSP 的应用程序。

（3）虚拟机角色

虚拟机角色使开发人员能通过定义 Windows Server 2008 R2 操作系统的定制镜像和应用程序所需的所有服务栈，完全控制计算服务的计算栈。虚拟机角色基于 Windows Hyper-V 虚拟化技术（见 3.6.3 节）和 Azure Windows 服务器技术天然集成。开发人员可以镜像一个 Windows 服务器安装，完成所有必需的应用程序和组件，将其保存到虚拟硬盘（VHD）文 343
件，并把它上传到 Windows Azure 上创建按需计算实例。实例有不同的类型，表 9-7 概述了 2011 ～ 2012 年期间提供的可选实例。

表 9-7 Windows Azure 计算实例的特性，2011 ～ 2012

计算实例类型	CPU	内存	实例存储	I/O 性能	每小时费用（美元）
特小型	1.0 GHz	768 MB	20 GB	低	$0.04
小型	1.6 GHz	1.75 GB	225 GB	中等	$0.12
中型	2 × 1.6 GHz	3.5 GB	490 GB	高	$0.24
大型	4 × 1.6 GHz	7 GB	1 000 GB	高	$0.48
超大型	8 × 1.6 GHz	14 GB	2 040 GB	高	$0.96

比起工作者角色和 Web 角色，虚拟机角色提供了对部署在 Azure 云上的计算服务和资

源的更好的控制。但需要额外的服务配置、安装和管理。

2. 存储服务

计算资源配备了本地存储，在本地文件系统上以目录的形式暂时存储信息，用于角色的当前执行周期。如果角色在不同的物理机器上重新启动或激活，该信息将丢失。

相对于本地存储，Windows Azure 提供了不同类型的存储解决方案，使计算服务具有更持久和冗余的选择。这些服务可以由来自世界各地的多个客户端同时访问，从而成为一个通用的存储解决方案。

（1）二进制大对象

Azure 允许通过 blob 服务方式以二进制大对象（BLOB）的形式存储大量数据。这项服务对于存储大型文本或二进制文件是最优的。有两种类型的 blob：

- 块 blob：块 blob 由块组成，对顺序访问进行了优化，因此适用于流媒体。目前，块大小为 4 MB，而一块 blob 可以达到 200 GB。
- 页面 blob：页面 blob 由页面组成，由 blob 起始处的偏移确定。一个页面的 blob 可分成多页或由单页构成。这种 blob 类型可随机存取，用于托管不同于流的数据。目前，一个页面 blob 的最大尺寸可以为 1TB。

blog 存储允许用户通过添加元数据来描述数据。另外，也可以对一个 blob 的快照进行备份。此外，为了优化分布，blob 存储可以利用 Windows Azure CDN 使 blob 接近用户需求，
344 以有效地提供服务。

（2）Azure 驱动器

页面 blob 以一个单一虚拟硬盘驱动器（VHD）文件的形式存储整个文件系统，因此可以作为 Azure 计算资源的 NTFS 文件系统的一部分来安装，从而提供持续永久的存储。安装为 NTFS 树的一部分的一个页面 blob，称为 Azure 驱动器。

（3）表

表是一个半结构化存储解决方案，允许用户以实体和一组属性的形式存储信息。实体存储为表中的行，标识为一个键，键也构成表的唯一索引。用户可以对存储在表中的行的子集进行插入、更新、删除和选择。和 SQL 表不同的是，这里的表没有强制实施约束实体属性的模式，也不能表示实体之间的关系。因此，表更类似于电子表格，而不是 SQL 表。

这项服务专门用来处理大量数据和返回大量结果集的查询。此功能由部分结果集和表分区支持。部分结果集和一个连续标记一起返回，允许用户重新查询大结果集。为了负载均衡，表分区允许表分散在几个服务器。一个分区由一个键标识，键由表中的三列属性来表示。

目前，一个表可以包含多达 100 TB 的数据，并且每行可以具有多达 255 个属性，每一行最大 1 MB。行键和分区键最大为 1 KB。

（4）队列

队列存储允许应用程序通过持久队列交换消息进行通信，从而避免丢失或未经处理的消息。应用程序输入的消息放入一个队列，其他应用程序可以按先进先出（FIFO）顺序读取消息。

为了确保消息得到处理，当应用程序读取一条消息时，它将被标记为不可见，这样其他客户端便不能读取。一旦应用程序处理完消息，需要显式地从队列中删除该消息。这两个阶段确保消息从队列中删除之前得到处理，并且客户端故障不会妨碍消息的处理。同时，这也是该队列不执行严格的先进先出模式一个原因：若应用程序在处理消息过程中崩溃，此消息超时后再次可用，在此期间，其他消息可以被其他用户读取。另一种读取消息方法是偷看

(peeking)，消息允许检索，但只在队列中保持可见。被偷看的消息不认为是被处理了。

上述所有服务都在地理位置上复制一次，以确保重大灾害时可用。地域复制将数据复制到不同的数据中心，可能离原来的数据中心数百或数千英里远。

3. 核心基础设施：AppFabric

AppFabric 是一个全面的中间件，用于开发、部署和管理应用程序。应用程序可以是云上的或集成现有的应用程序和云服务。AppFabric 实现了一个优化的基础设施，支持向外扩展和高可用性、沙箱和多租户、状态管理，动态地址解析和路由。在此基础之上，中间件提 345
供一组服务，简化了许多在分布式应用程序中的常见任务，如通信、身份验证、授权以及数据访问。这些服务都可以通过语言无关的接口使用，从而允许开发人员构建异构应用程序。

（1）访问控制

AppFabric 允许用户对 Web 应用程序的资源和服务进行访问控制，该功能被编码成一组规则，在应用程序代码库之外表示。这些规则在保护应用程序的组件以及为用户和组定义访问控制策略方面有很大的灵活性。

访问控制服务还将多种身份验证提供者集成到一个单一的、一致的身份管理框架。应用程序可以利用活动目录、Windows Live、Google、Facebook 和其他服务来验证用户身份。此功能更易建设混合系统，一部分存在于私有云上，其他的部署在公共云上。

（2）服务总线

服务总线是消息传递和连接的基础设施，由 AppFabric 提供，为 Azure 云以及私有设施与 Azure 云之间构建分布和非连接的应用程序。在一个可靠的保证发布的通信通道上，服务总线允许应用程序与不同的协议和模式交互。

这项服务的目的是支持透明的网络和简化松耦合应用程序的开发，它没有否认安全性和可靠性，而是让开发人员专注于交互性而不是实现细节的逻辑。服务总线允许服务通过简单的 URL 访问，并从 URL 部署的位置解析。服务总线可支持发布 - 订阅模型和全双工通信的点对点模型，以及对等网络环境中单向通信的单播和多播消息传递和异步消息，从而解耦 / 分离应用程序组件。

为了充分利用这些特性，应用程序需要连接到提供这些服务的总线。连接是服务总线元素，Azure 按即付即用定价。用户账单按每个月的连接计算，如果能提前估计他们的需求也可以提前购买折扣价的“连接包”。

（3）Azure 缓存

Windows Azure 提供了一组持久存储解决方案，允许应用程序保存数据。这些解决方案是基于磁盘存储的，这对按客户要求和数据集的大小进行扩展的应用程序来说，可能是一个瓶颈。

Azure 缓存是一种服务，允许开发人员快速访问存储在 Windows Azure 或 SQL Azure 中的数据。该服务实现了一个分布式内存缓存，大小可以由应用程序根据自己的需要进行动态调整。它可以存储任何 .NET 管理对象，以及许多常见的数据格式（表的行、XML、二进制数据），并通过应用程序控制其访问。Azure 缓存作为服务交付，并且它可以与应用程序轻松 346
集成。特别是 ASP.NET 应用程序，已经基于 Azure 缓存集成了会话状态供应商和页面输出缓存。

该服务根据应用程序每月分配缓存的大小定价，尽管它们能够有效地使用缓存。目前，有几个缓存尺寸可供选择，范围从 128 MB（45 美元 / 月）到 4 GB（325 美元 / 月）。

4. 其他服务

计算、存储和中间件服务构成了 Windows Azure 平台的核心组成部分。除此之外，其他的服务和组件简化了 Azure 云计算应用程序的开发和集成。这些服务的一个重要领域是应用程序的连接，包括虚拟网络和内容发布。

（1）Windows Azure 虚拟网络

应用程序的网络服务由 Windows Azure 虚拟网络提供，包括 Windows Azure 连接和 Windows Azure 流量管理器。

Windows Azure 连接可以方便地设置托管在私有基础设施以及部署在 Azure 云的角色之间的基于 IP 的网络连接。这项服务对于虚拟机角色特别有用，其中托管在 Azure 云的机器成为企业私有网络的一部分，并且可以使用与私人处所中相同的工具进行管理。

Windows Azure 流量管理器为 HTTP 或 HTTPS 端口的服务监听提供负载平衡功能，并托管在多个角色上。开发者可以从三个不同的负载均衡策略中选择：性能、轮转和故障转移。

目前，这两项服务仍处于 β 测试阶段，只有通过邀请才免费提供。

（2）Windows Azure 内容交付网络

Windows Azure 内容交付网络（CDN）是内容交付网络解决方案，它提高了 Windows Azure 存储和其他 Microsoft 服务的内容交付能力，如 Microsoft Windows 更新和 Bing 地图。该服务允许通过使用分布在世界各地的 24 个网点提供 Web 对象（图像、静态 HTML、CSS 和脚本）以及流媒体服务。

9.3.2　SQL Azure

SQL Azure 是关系数据库服务，托管在 Windows Azure，并基于 SQL Server 技术构建。这项服务将 SQL Server 功能扩展到云计算，并为开发者提供可扩展、高度可用和容错的关系数据库。可从 Windows Azure 云或有权访问到 Azure 云的其他任何位置访问 SQL Azure。它与 SQL Server 所提供的接口完全兼容，所以 SQL Server 建立的应用程序可以透明地迁移到 SQL Azure。此外，通过使用 REST API，开发人员能够控制部署在 Azure 云的数据库，并设置其访问防火墙规则的服务是完全可控的。

347 图 9-4 显示了 SQL Azure 的架构。对 SQL Azure 的访问基于表格格式数据流（TDS）协议，这是应用程序使用的所有不同接口之下的通信协议，用来连接基于 SQL Server 的安装，如 ODBC 和 ADO.NET。SQL Azure 通过服务层访问数据，提供了服务开通、计费和连接路由服务。这些服务逻辑上是服务器实例的一部分，由 SQL Azure Fabric 管理。它是分布式数据库中间件，构成 SQL Azure 的基础设施，在微软数据中心部署。

开发人员必须注册一个 Windows Azure 账户才能使用 SQL Azure。一旦该账户被激活，
348 便可使用 Windows Azure 管理门户或 REST API 创建服务器，登录并配置对服务器的访问。SQL Azure 服务器类似于物理 SQL Server 的抽象：具有 database.windows.net（即 server-name.database.windows.net）域名下的一个完全合格的域名。这就简化了管理任务以及客户端应用程序与 SQL Azure 的交互。SQL Azure 确保在 Azure 云内维护每个服务器的多个副本，当客户端应用程序中插入、更新和删除数据时，确保这些副本是同步的。

目前，SQL Azure 的服务是根据空间的使用和编辑的类型计费。有两种不同的版本可供选择：Web 版和商业版。前者适合小型 Web 应用程序，支持最大 1 GB 或 5 GB 的数据库。后者适合独立的软件供应商、业务线应用程序和企业应用程序，支持 10 ～ 50 GB 的数据库，

增量为 10 GB。此外，带宽费用适用于误访问 / 删除 Windows Azure 云或数据库所在区域的任何数据传输。每用户 / 数据库的月租费收取基于本月数据库达到峰值大小。



图 9-4 SQL Azure 架构

9.3.3 Windows Azure 平台设备

Windows Azure 平台还可以部署为第三方数据中心的设备，构成监管数据中心的物理服务器的云基础设施。Windows Azure 平台设备包括：Windows Azure、SQL Azure 和微软的网络、存储和服务器硬件的特定配置。该设备是一个解决方案，提供给那些想拥有自己的云计算基础设施的政府和服务供应商。

如前所述，Azure 已经提供了一个开发环境，允许用户在自己的处所为 Azure 构建应用。本地开发环境的目的不是生产中间件，而是设计用来专门开发和测试应用程序的功能，应用程序最终将被部署在 Azure 上。Azure 设备没有实现 Windows Azure 的全部功能。它的目标是在第三方基础设施上复制 Azure，使其服务内容超出微软云的范围。该设备专注于两个主要场景：有非常大计算需求的机构（如政府部门），以及支付不起将数据向处所外转移的机构。

9.3.4 结论

Windows Azure 是微软用于开发云计算应用程序的解决方案。Azure 是 PaaS 层的实现，为开发人员提供了一组服务和托管在 Microsoft 数据中心的可扩展的中间件。Microsoft 数据中心负责计算、存储、网络化和处理应用程序的身份管理需求。Azure 服务可以单独或一起构建整合云功能的应用程序和在云中完全托管的弹性计算系统。

该平台的核心组件是计算服务、存储服务和中间件。计算服务基于角色的抽象，它确定了一个沙箱环境，开发人员可以构建自己的分布式和可扩展的组件。这些角色对 Web 应用
349 程序、后端处理和虚拟计算是有用的。存储服务包括静态和动态内容解决方案，以表格形式组织，比关系模型组织的约束更少。这些服务和其他服务通过 AppFabric 实现和获得，构成了 Azure 分布式和可扩展的中间件。

SQL Azure 是 Windows Azure 中的另一个重要元素，支持云中的关系数据。SQL Azure 是为适合云计算环境而设计的动态扩展 SQL Server 功能的延伸。

该平台主要基于 .NET 技术和 Windows 系统，也支持其他技术和系统。因此，Azure 是迁移到基于 .NET 技术的云应用程序的选择方案。

本章小结

本章介绍了一些业界广泛应用的建设真正的商业应用的云计算平台：亚马逊 Web 服务，谷歌 AppEngine 和微软 Windows Azure。

AWS 提供了亚马逊云中基础设施建设的解决方案。亚马逊 EC2 和亚马逊 S3 代表 AWS 的核心产品。前者允许开发人员创建虚拟服务器，并根据需要定制自己的计算堆栈。后者是一个存储解决方案，允许用户存储任意大小的文件。这些核心服务通过一组大量服务，包括网络、数据管理、内容发布、计算中间件和通信服务，从而构成在亚马逊基础架构之上开发整个云计算系统的 AWS 完整解决方案。

谷歌 AppEngine 是在云中构建 Web 应用程序的分布式和可扩展的平台。AppEngine 是一个可扩展的运行环境，为开发人员提供一系列服务，以简化 Web 应用程序的开发。这些服务旨在考虑可扩展性，构成了定义应用程序可重复使用的功能块。开发人员可以使用 Java 或 Python 构建自己的应用程序，先在本地使用 AppEngine SDK，一旦应用程序完成并经过全面测试，就可以在 AppEngine 上部署应用程序。

Windows Azure 部署在微软的数据中心，是用来构建动态可扩展的应用程序的云操作系统。Azure 的核心部件以计算服务表示，包括角色、存储服务以及 AppFabric。AppFabric 是联系所有服务并构成 Azure 基础设施的中间件。角色是一个沙箱运行环境，专门在特定的并发场景使用：Web 应用程序、后台处理和虚拟计算。开发人员以角色定义 Azure 应用程序，然后在 Azure 上部署这些角色。存储服务代表角色的正常补充。除了存储静态数据和半结构化数据，Windows Azure 还提供 SQL Azure 服务方式存储关系数据。

AppEngine 和 Windows Azure 是 PaaS 解决方案。尽管以 EC2 和 S3 为代表的 AWS 因提
350 供 IaaS 而众所周知，但 AWS 服务在整个云计算参考模型的全部三层中都有所扩展。

习题

1. 什么是 AWS？它提供什么类型的服务？
2. 描述亚马逊 EC2 及其基本功能。
3. 什么是桶？它提供什么类型的存储？
4. 亚马逊简单数据库和亚马逊 RDS 之间的差异是什么？
5. 亚马逊 VPC 解决什么类型的问题？
6. 介绍 AWS 为支持应用程序之间的连接所提供的服务。
7. 什么是亚马逊 CloudWatch？

8. AppEngine 是什么类型的服务?
9. 描述 AppEngine 的核心部件。
10. AppEngine 目前支持什么开发技术?
11. 什么是数据存储? 其中可以存储什么类型的数据?
12. 讨论 AppEngine 提供的计算服务。
13. 什么是 Windows Azure ?
14. 详细说明 Windows Azure 的架构。
15. 什么是角色? 可使用哪些类型的角色?
16. 什么是 AppFabric ? 它提供哪些服务?
17. 讨论由 Windows Azure 提供的存储服务。
18. 什么是 SQL Azure ?
19. 举例说明 SQL Azure 的架构。
20. 什么是 Windows Azure 平台设备? 此设备可用于什么场景?

351
2
352

第 10 章

Mastering Cloud Computing: Foundations and Applications Programming

云 应 用

由于能托管应用程序，服务可迅速地以最低成本提供给消费者，云计算已经在业界受到极大的欢迎。本章讨论了一些应用案例，详细介绍了它们的架构以及如何利用各种云技术。应用程序涉及一系列领域，从科学到工程、游戏和社交网络。

10.1 科学应用

科学应用越来越多地使用云计算系统和技术。研究人员和学者从无限的可用计算资源和存储能力中获得潜在的收益，与自己部署资源相比，利用资源的花费是可接受的。云计算系统满足科学领域不同类型的应用需求：高性能计算（HPC）应用，高吞吐量计算（HTC）应用，数据密集型应用。使用云资源非常吸引人，因为为了充分利用云资源，对现有的应用程序只需要做最小的改变。

最相关的解决方案是 IaaS，它提供了运行任务包应用程序和工作流的最佳环境。IaaS 定制虚拟机实例来托管运行应用程序所需的软件栈，并与分布式计算中间件协同，以与云基础设施进行交互。PaaS 的解决方案已经在考虑之列。这使得科学家要探索新的编程模型来解决大计算量的挑战性问题。应用程序已经经过重新设计，并在云应用程序编程模型和平台之上实现，充分利用其独特的功能。例如，MapReduce 编程模型为科学家提供了一个非常简单而有效的模型，来构建处理大数据集的应用程序。因此，它已被广泛地用于开发数据密集型科学应用。若计算模型的结构方面需要更高程度的弹性，可以利用 Aneka 平台，它支持 MapReduce 和其他编程模型。下面讨论在 Aneka 中已经使用的一些有趣的案例。

10.1.1 医疗保健：云心电图分析

医疗保健是计算机技术的应用领域之一，从支持业务功能到协助科学家开发疾病治愈方案。一个重要的应用是利用云技术为医生提供更有效的诊断过程，特别是云计算环境下的心电图（ECG）数据分析[160]。

随着互联网的普及，任何时候从任何设备都可访问的特性，使云技术成为开发健康监测系统的一个有吸引力的选择。很自然地，ECG 数据分析和监测成为适合这个情形的案例。ECG 是心脏的心肌收缩活动的电子体现，这项活动产生一个特定的波形，波形随着时间的推移重复，表示心跳。ECG 波形的形状分析是用来识别心律失常和检测心脏疾病的最常见方式。云计算技术可远程监控病人的心跳数据，在最短的时间内分析数据，如果这些数据有潜在的危险则通知急救人员和医生。病人通过不断监测，不去医院就可以进行心电图分析。与此同时，医生和急救人员可以即时获得他们关注的信息。

支持远程心电监护的基础设施和模型如图 10-1 所示。配备了心电图传感器的可穿戴计算设备不断地监测病人的心跳，这些信息被发送到患者的移动设备，最终转发到云托管 Web 服务以供分析。Web 服务形成了完全托管在云平台的前端，并且充分利用了三层云计算堆栈：

SaaS，PaaS 和 IaaS。Web 服务构成了 SaaS 应用程序，在亚马逊 S3 服务存储心电数据，并给可扩展的云平台发出处理请求。运行平台由运行在工作流引擎和 Aneka 上的大量的动态实例组成。工作流引擎实例的数量根据请求的每个实例的队列数目进行控制，而 Aneka 控制 EC2 实例的数量，用于执行由工作流引擎为单个 ECG 处理作业中指定的单个任务。这些作业中的每个作业都由一组操作组成，涉及从心跳数据提取波形，并和参考波形比较来检测异常的波形。如果发现异常，医生和急救人员可以收到通知，对特定的患者采取措施。

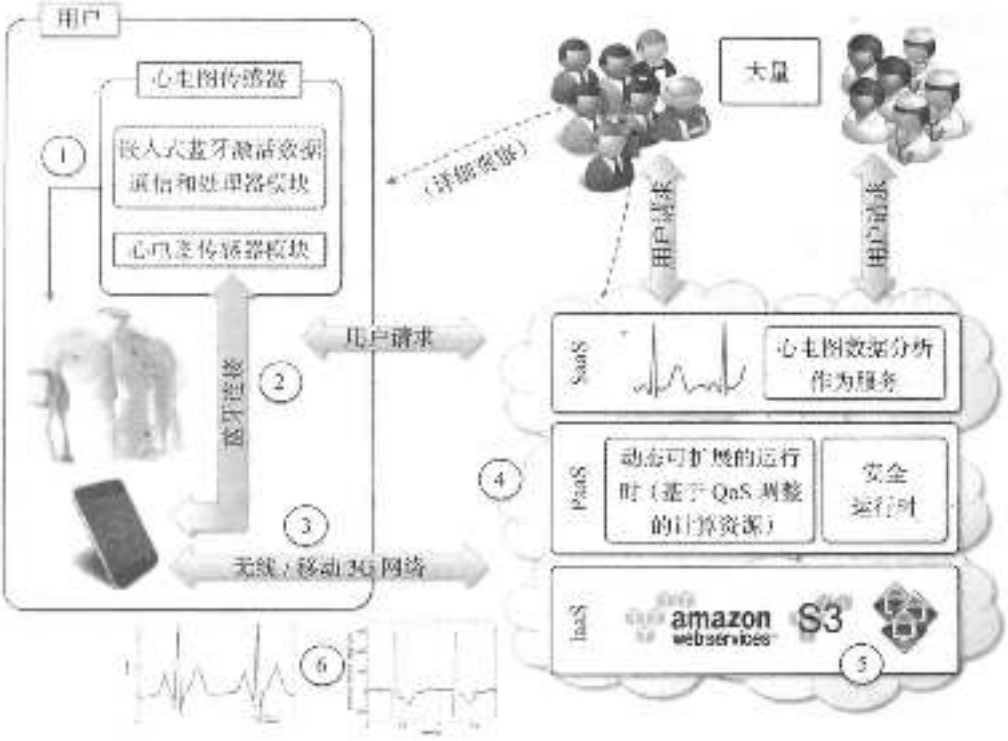


图 10-1 云在线健康监测系统

尽管远程心电监护并不一定需要云技术，但云计算提供的优势是其他技术很难实现的。第一个优势是云基础设施具有弹性，根据服务请求可以增大和缩小。这样，医生和医院不必规划容量后投资大量计算基础设施，从而更有效地利用预算。第二个优势是无处不在。现在的云计算技术已经成为容易访问且保证在最短时间或没有停机时间的发布系统。托管在云的计算系统可以从任何网络设备通过简单的接口（如 SOAP 和基于 REST 的 Web 服务）访问。这使得云系统不仅普遍存在，而且可以很容易地与保存在该医院的其他系统集成。最后，医疗保健中使用云技术的另一原因是节约成本。云服务定价按使用时间计算费用，大量的服务请求按量定价。这两款模式提供了一套用于定价服务的灵活选择，实际上收费成本是基于有效使用，而不是建设成本。

354

10.1.2 生物学：蛋白质结构预测

生物学中的应用往往需要较高的计算能力，并经常在大数据集上进行大量的 I/O 操作。所以生物学的应用程序经常广泛使用超级计算和集群计算基础设施。类似的功能以动态方式按需使用云计算技术，从而开辟生物信息学应用的新机遇。

蛋白质结构预测是一种计算密集型的任务，是生命科学领域不同类型研究的基础。其中之一是为治疗疾病设计新的药物。蛋白质的几何结构不能从构成其结构的基因序列直接推断出，它是复杂计算的结果，目的是确定所需要能量最小的结构。此任务需要研究数量庞大的状态空间，从而为每个状态进行大量的计算。蛋白质结构预测所需的计算能力现在可以按需
355 获取，不需要拥有集群或跨越政府部门访问并行和分布式计算设施。云计算允许按用量付费来使用这种基础设施。

使用云技术的蛋白质结构预测项目之一是 Jcova [161]———一个集成的 Web 门户网站，使科学家能够将预测任务转移到一个基于 Aneka 的计算云（见图 10-2）。预测任务使用机器学习技术（支持向量机）确定蛋白质的二级结构。这些技术将问题转变成模式识别，其中一个序列已被分类成三种可能的类型（E、H 和 C）之一。一种常见的实现是基于支持向量机的，把模式识别问题划分为三个阶段：初始阶段、分类阶段和终止阶段。即使这三个阶段按顺序执行，在分类阶段也能利用并行执行的优点，让多个分类器并发执行，可以明显地降低预测的计算时间。然后将预测算法转换成一个任务图提交到 Aneka，一旦完成任务，用户可通过门户查看由中间件提供的结果。

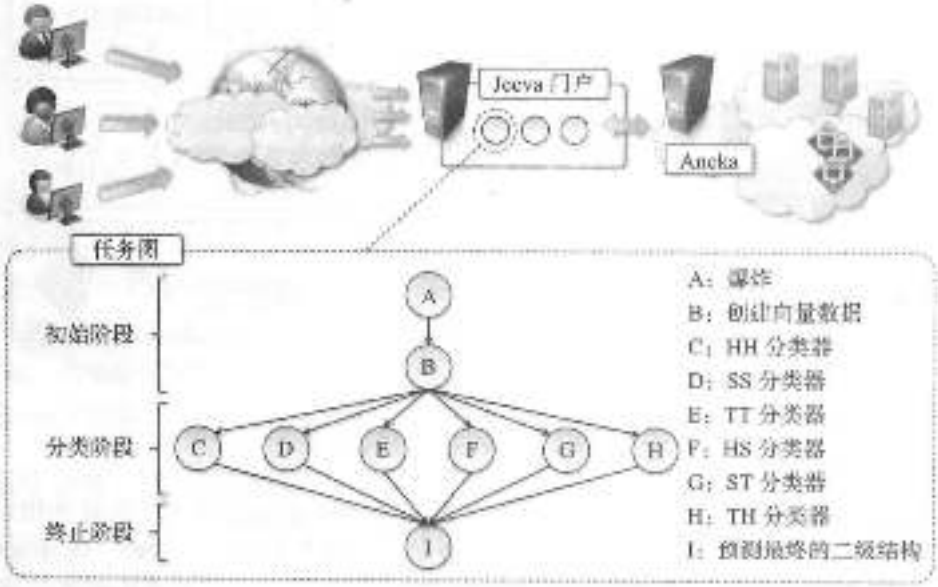


图 10-2　Jcova 门户架构概览

使用云技术（即 Aneka 作为可扩展的云中间件）相比传统网格基础设施的优点是：可扩展的计算基础设施能根据需求增加和缩小。这个概念是云技术特有的，当应用程序作为服务
356 提供和交付时，它就成为一项战略优势。

10.1.3　生物学：基因表达数据用于癌症诊断分析

基因表达谱是数千种基因的表达水平的一次测量，用于分析药物治疗在细胞层引发的生物学过程。连同蛋白质结构预测，此活动是药物设计的一个基本组成部分，它使科学家可以识别特定的治疗效果。

基因表达谱的另一个重要应用是癌症的诊断和治疗。癌症的特点是细胞的生长和扩散不受控制，原因是基因调节细胞生长发生变异，这意味着所有癌症细胞中都含有突变基因。在

这种情况下，基因表达谱可以用来提供更精确的肿瘤分类。把基因表达数据样本分类成不同的类别是一项具有挑战性的任务。典型的基因表达数据集的维数范围从几千到几万基因。然而，通常只有小样本量可用于分析。

这个问题经常和分类学习相关，产生条件－活动规则的种群来指导分类处理。其中，扩展分类系统（XCS）已成功地在生物信息学和计算机科学领域用于分类大型数据集。然而，当面临高维数据集（例如微阵列基因表达数据集）时，XCS 的有效性并未得到详细探讨。这种算法的一个变种 CoXCS [162]，已证明能有效地解决这个问题。CoXCS 将整个搜索空间划分成子域，并在每个子域采用标准的 XCS 算法。这样的处理是计算密集型的，但很容易并行化，因为子域的分类问题可以并发解决。云 CoXCS（见图 10-3）是 CoXCS 基于云计算的实现，Aneka 以并行方式解决分类问题并将结果组合起来。该算法由策略控制，定义结果的组合方法和该进程是否需要迭代。

357

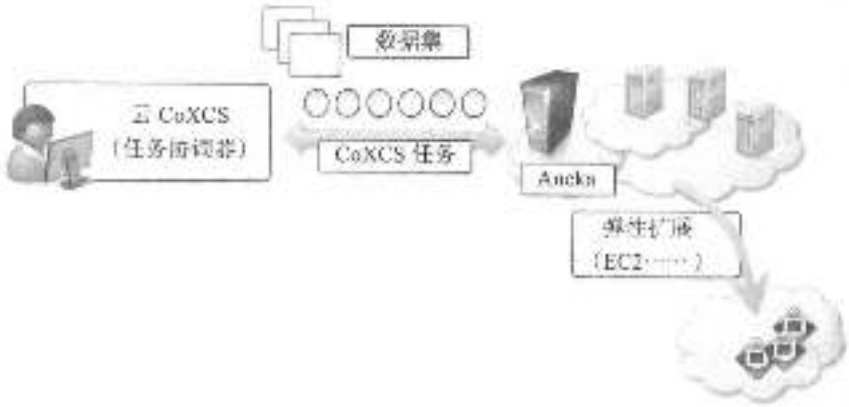


图 10-3 云 CoXCS：云上的微阵列数据处理

因为 XCS 的动态特性，所以要求执行的计算资源的数量可以随时间变化。因此，使用可扩展的中间件（如 Aneka）具有独特的优势。

10.1.4 地球科学：卫星图像处理

地球科学应用程序收集、制作和分析地理空间和非空间的海量数据。随着技术的进步和星球变得更加仪器化（通过部署传感器和卫星监测），需要加以处理的数据量显著增加。特别地，地理信息系统（GIS）是地球科学应用的主要组件。GIS 应用系统采集、存储、处理、分析、管理和呈现所有类型的地理空间参考数据。现在这种类型的信息和各种应用领域越来越相关：从先进的农业到民事安全和自然资源管理。结果是，相当数量的地理参考数据被摄取到计算机系统中进行进一步的处理和分析。为了执行这些苛刻的任务并提取有意义的信息来支持决策者，云计算是一种有吸引力的选择。

卫星遥感产生数几百吉字节原始图像，需要进一步加工变成几种不同的 GIS 产品的基础。这个过程既要求 I/O 也要求计算密集型任务。大图像需要从一个地面站的本地存储迁移到计算设备，应用几次转换和校正。云计算提供恰当的基础设施支持这样的应用场景。这种基于云实现的工作流程已经由印度政府航天部开发出来了 [163]。如图 10-4 所示的系统在整个计算栈集成了多种技术。SaaS 应用程序为这种任务提供一系列服务，如地理编码生成和数据可视化。在 PaaS 层，Aneka 控制数据导入虚拟化基础设施和图像处理任务的执行，

即从原始卫星图像生成所期望的结果。该平台利用 Xen 私有云和 Aneka 技术根据需求动态地提供所需的资源（即增大或缩小）。

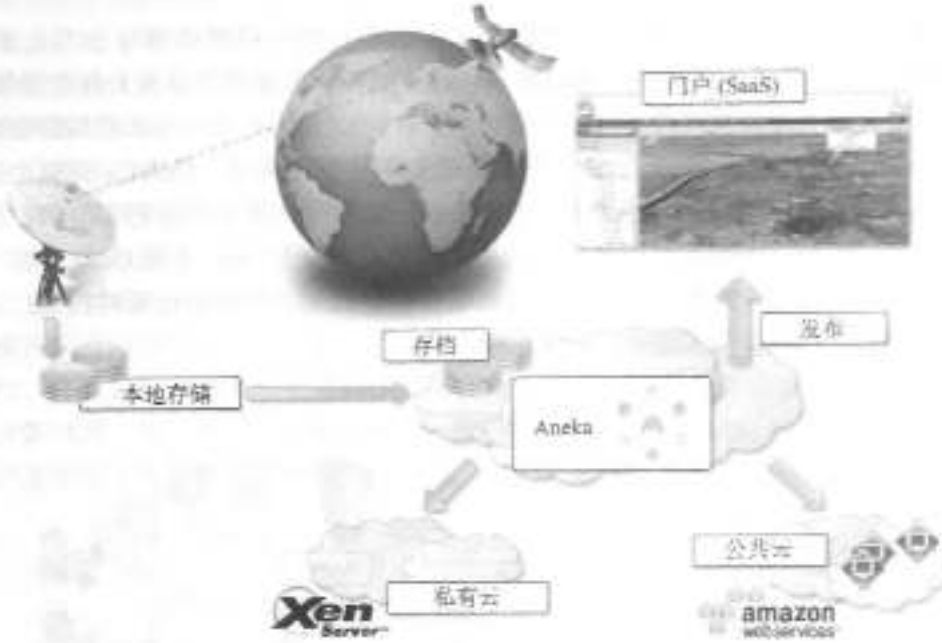


图 10-4　一个卫星数据处理的云环境

该项目证明了云计算技术如何有效地将过多的工作负荷从本地计算设备转移，并充分利用更有弹性的计算基础设施。

10.2　商业和消费应用

商业和消费是最可能从云计算技术受益的领域。一方面，对以 IT 为中心的所有企业来说，能将建设成本转化为运营成本，云是一项有吸引力的选择。另一方面，从普遍意义上说，云提供数据和服务访问，使得终端用户也对它有兴趣。此外，云技术的弹性性质并不需
358 要巨大的前期投资，新的想法能迅速转化为产品和服务，并根据需求增长。这些因素使云计算成为首选技术，适用于大范围的应用程序，从 CRM 和 ERP 系统到生产类型和社交网络型应用程序。

10.2.1　CRM 和 ERP

在云中，客户关系管理（CRM）和企业资源规划（ERP）应用程序是两个蓬勃发展的市场方向，其中 CRM 应用更为成熟。云 CRM 应用程序对小企业和刚起步的企业而言是一个大好机会，因为有功能齐全的 CRM 软件，无需大量的前期成本，只需定期付费。此外，CRM 不需要指定特定需求，它可以很容易地转移到云。这样的特性，再加上从世界各地的任何设备访问企业的业务和客户数据的可能性，共同促进了云 CRM 应用的传播。云上的 ERP 解决方案不太成熟，必须与完善的企业内部解决方案竞争。ERP 系统整合企业的几个方面：财务及会计、人力资源、生产制造、供应链管理、项目管理和 CRM。ERP 的目标是为所有操作提供一个统一的视图和访问，以维持一个复杂的组织。因为目标是组织机构，所
359 以 ERP 过渡到基于云的模型是比较困难的：长期的成本优势可能不是很清楚，如果组织机

构已经拥有大型ERP，将其转移到云可能是困难的。出于这个原因，目前云计算ERP解决方案没有CRM解决方案普及。

1. Salesforce.com

Salesforce.com可能是目前最流行和最先进的CRM解决方案，有10多万客户选择Safesforce.com实施他们的CRM解决方案。该应用程序提供定制的CRM解决方案，可与第三方开发的附加功能集成。Salesforce.com基于Force.com云开发平台，这代表了可扩展性和高性能的中间件能执行所有Salesforce.com应用程序的所有操作。

Force.com平台的架构如图10-5所示。该平台最初设计为支持可扩展的CRM应用，通过实施灵活的、可扩展的基础设施，已经演化到能支持大量云应用的整个生命周期。该平台的核心在于其元数据架构，它为系统提供灵活性和可扩展性，而不是建立在特定的组件和表之上。应用程序的核心逻辑和业务规则作为元数据保存到Force.com存储。应用程序的结构 360
和应用程序数据存储在存储区中。运行时引擎通过检索元数据执行应用程序逻辑，然后在数据上执行操作。虽然在隔离容器运行，但不同的应用逻辑上共享相同的数据库结构，运行时引擎一致地执行所有的应用程序。全文搜索引擎支持运行时引擎。尽管有大量数据需要被抓取，但这使得应用程序的用户获得有效的用户体验。搜索引擎在一个单独的存储空间维护其索引数据，并通过用户交互触发后台进程不断更新。

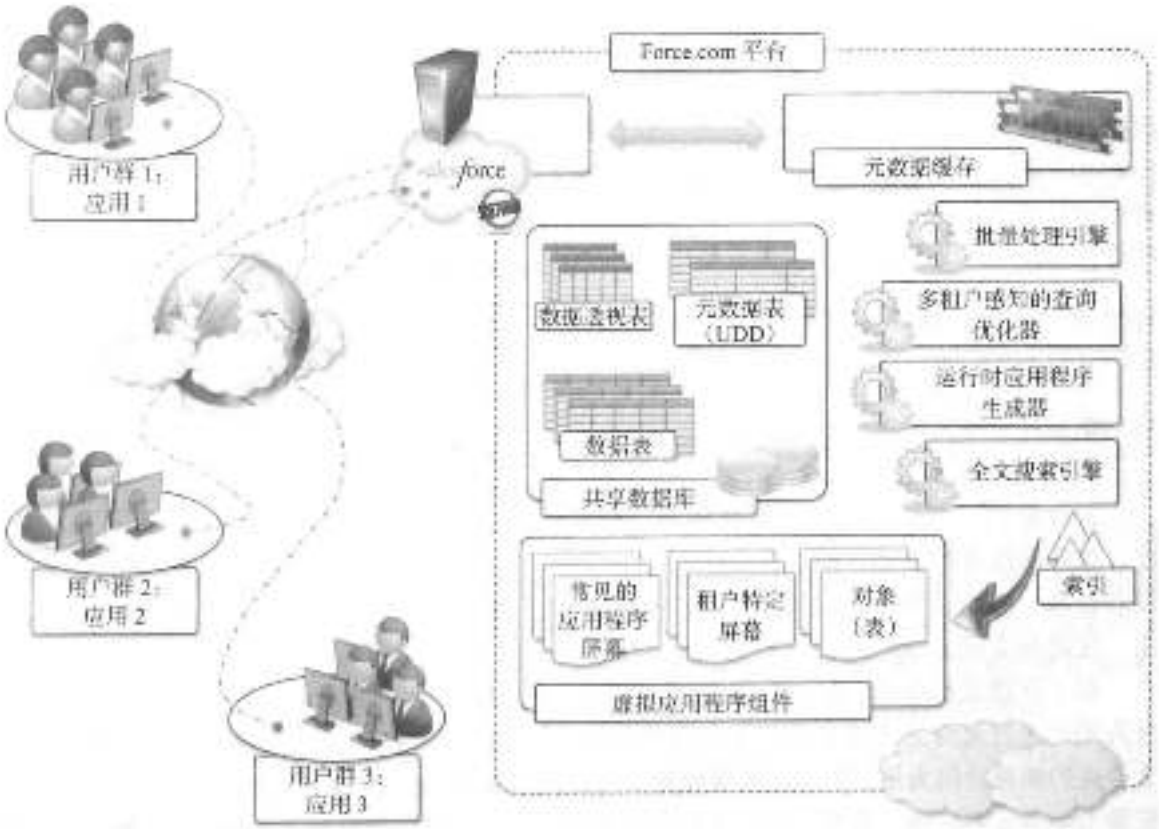


图10-5 Salesforce.com和Force.com架构

用户可以通过“本地”Force.com应用框架或在最流行的编程语言中使用编程API定制应用程序。应用程序框架允许用户可视化地定义数据或者一个Force.com应用程序的核心结构，而编程API提供了开发应用程序的更传统的方式，即应用程序依赖于Web服务与平台

交互。应用程序和逻辑的定制也可以通过在 APEX 开发脚本来实现。这是一个类似 Java 的语言，它提供面向对象和过程的功能来定义按需的脚本或触发器。APEX 还提供了表达搜索和查询的功能，以完整地访问 Force.com 平台管理的数据。

2. 微软动态 CRM

微软动态 CRM 是微软为客户关系管理实施的解决方案。动态 CRM 可在企业场所安装或作为按每月每用户订阅定价的在线解决方案。

该系统完全托管在世界各地的微软数据中心，提供给客户的 SLA 达到 99.9%，如果系统不履行协议将给用户优惠。每个 CRM 的实例在一个单独的数据库部署，应用程序为用户提供了市场、销售以及先进的 CRM 设施。动态 CRM Online 的功能可以通过 Web 浏览器界面访问，也可以通过 SOAP 编程方式和 RESTful Web 服务进行访问。这使得动态 CRM 可以很容易地与其他微软产品和行业应用集成。动态 CRM 可以通过开发插件进行扩展，插件可执行触发给定事件发生的给定行为。动态 CRM 还可以利用 Windows Azure 开发和集成新功能。

3. NetSuite

NetSuite 公司提供一系列应用程序，帮助客户管理企业的每个环节。其产品分为三大系列：NetSuite 全球 ERP 系统、NetSuite 全球 CRM+ 和 NetSuite 全球电子商务。此外，集所有功能于一身的解决方案 NetSuite One World 集成了所有三种产品。

NetSuite 提供的服务由美国东西海岸的两个大数据中心通过冗余链路连接供电。这使 NetSuite 能向客户保证 99.5% 的正常运行时间。除了预封装的解决方案，NetSuite 还提供了基础设施和开发环境，实现应用程序的定制。NetSuite 商业操作系统（NS-BOS）是一组完整
361 堆栈技术，充分利用 NetSuite 产品的功能构建 SaaS 业务应用程序。在 SaaS 基础设施之上，NetSuite 商业套装组件提供会计、ERP、CRM 和电子商务功能。在线开发环境 SuiteFlex 能将此种功能集成到新的 Web 应用程序，然后由 SuiteBundler 打包进行发布。整个基础设施托管在 NetSuite 数据中心，提供有关应用程序的正常运行时间和可用性的保证。

10.2.2　效率型应用

效率型应用程序在云中复制一些我们在桌面上执行的最常见的任务：从文件存储到办公自动化和托管在云中完整的桌面环境。

1. Dropbox 和 iCloud

云计算的一个核心功能是任何地方、任何时间、任何连接到因特网的设备都可用。因此，这样的技术使文档存储成为一个自然的应用。在线存储解决方案在云计算之前就有，但没有普及。随着云技术的发展，在线存储解决方案已经变成了 SaaS 应用程序，变得更加实用、先进和方便。

对于在线文档存储，最流行的解决方案也许是 Dropbox。Dropbox 是一个在线应用程序，允许用户通过无缝的方式在任何平台和任何设备同步任何文件（见图 10-6）。Dropbox 通过对文件夹的抽象访问为用户提供免费存储量。用户可以访问 Dropbox 文件夹，通过浏览器或下载安装 Dropbox 客户端，利用一个特殊的文件夹访问在线存储。所有对这个文件夹的修改是悄然
362 同步的，因此修改会通知所有设备 Dropbox 文件夹中的所有本地实例。Dropbox 的主要优点是它在不同平台（Windows、Mac、Linux 和移动平台）的可用性和无缝透明跨平台工作的能力。

在这方面的另一个有趣的应用是 iCloud，苹果公司提供的基于云的文件共享应用程序以完全透明的方式同步 iOS 设备。与 Dropbox 不同，它通过一个本地文件夹的抽象提供同步，

一旦设置，icloud 便完全透明。更改文档、照片和视频时自动同步，没有任何显式的操作。这使得该系统能够有效地自动执行常见操作，而无需任何人为干预：用 iPhone 拍一张照片，它会自动地出现在家里 Mac 的 iPhoto 里；在家里用 iMac 电脑编辑文档，iPad 便可自动更新。遗憾的是，这种功能仅限于 iOS 设备，目前没有计划提供 iCloud 与基于 Web 的接口，不支持的平台无法访问用户内容。

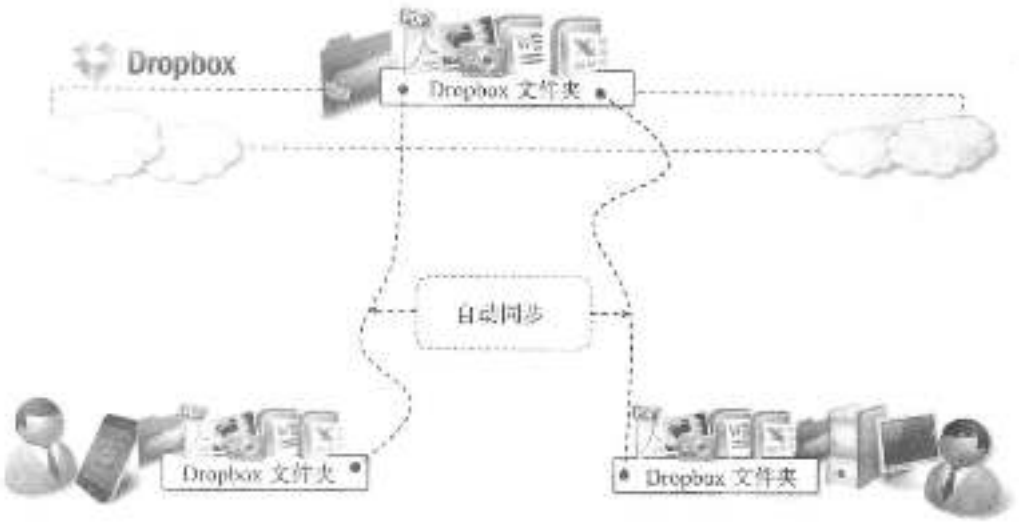


图 10-6 Dropbox 使用场景

在线文件共享还有其他的解决方案，Windows Live、亚马逊 Cloud Drive 和 CloudMe 都是很受欢迎的，本节没有涉及。这些解决方案提供的功能与我们已经讨论过的多少有些相同，只是在平台和设备之间的集成层次不同。

2. 谷歌 Docs

谷歌 Docs 是一个 SaaS 应用程序，它提供了基本的办公自动化功能，支持在 Web 上协同编辑。该应用程序在谷歌分布式计算基础设施的顶部执行，这使得系统能够根据使用该服务的用户数量动态扩展。

谷歌 Docs 允许用户创建和编辑文本文档、电子表格、演示文稿、表格和图。它的目标是取代台式产品，如微软 Office 和 OpenOffice，提供类似的界面和功能的云服务。对该套件中包含的大多数应用程序，它支持在 Web 上协同编辑。这消除了当文件需要由多个用户进行编辑时繁琐的收发电子邮件和同步任务。通过在谷歌基础设施上存储，对于任何地方以及从连接到互联网的任何装置，这些文件始终可以使用。此外，如果互联网连接不可用，该套件支持用户脱机工作。谷歌 Docs 支持多种格式，如那些最流行的桌面办公解决方案产生的格式，使用户能够轻松地导入和导出谷歌 Docs 文档，从而消除了使用这种应用程序的障碍。

谷歌 Docs 是云计算可以提供给最终用户的一个很好的例子：获取资源无处不在、弹性、免安装、免维护成本，及核心功能作为服务发布。

3. 云桌面：EyeOS 和 XIOS / 3

异步 JavaScript 和 XML（AJAX）技术已经大大增强了 Web 应用程序能实现的功能。这是云计算通过 Web 浏览器提供的大量服务中的一个基本方面。加上大规模存储和计算的使用，这项技术已使通过 Web 浏览器在云中复制复杂的桌面环境成为可能。这些称为云桌面的应用程序正在迅速普及开来。

EyeOS[⊖]是基于云技术的最流行的 Web 桌面解决方案之一。它复制了经典桌面环境的功能，并为最常见的文件和文件管理任务预安装了应用程序（见图 10-7）。单个用户可以在任何地方通过任何联网设备访问 EyeOS 的桌面环境，而企业可以在其处所创建一个私有的 EyeOS 云，虚拟员工的桌面环境并集中管理。

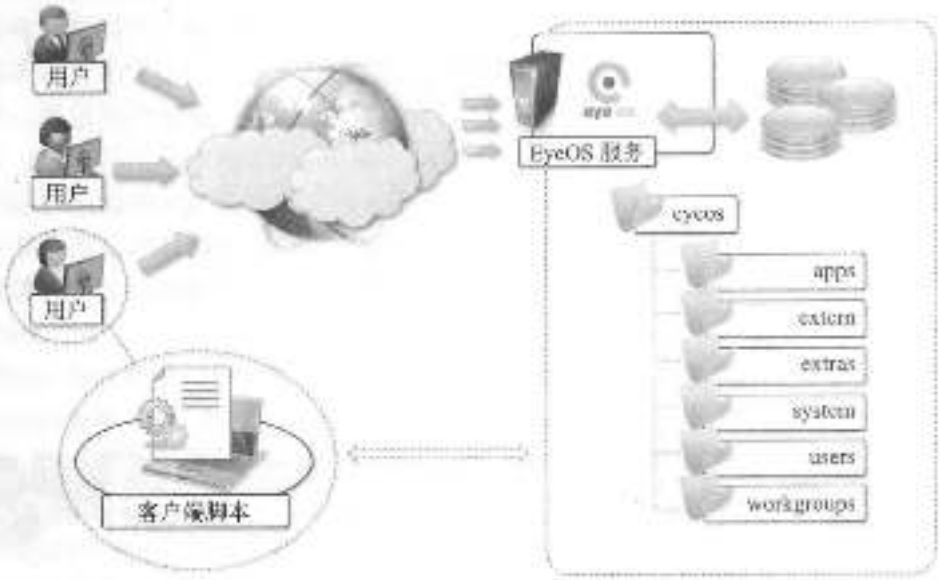


图 10-7　EyeOS 体系结构

EyeOS 的架构非常简单：在服务器端，EyeOS 的应用程序维护用户配置文件及其数据信息，客户端则是用户和管理员与系统交互的接入点。EyeOS 存储有关服务器文件系统上的用户和应用程序的数据。一旦用户提供认证登录，桌面环境便呈现在客户端的浏览器上。这是通过下载构建用户界面和实现 EyeOS 的核心功能所需的所有 JavaScript 库来实现的。在环境中加载的每个应用程序使用 AJAX 与服务器通信。这种通信模式用于访问用户数据，以及执行应用程序操作：编辑文件、可视化图像、复制和保存文件、发送电子邮件和聊天。

EyeOS 还提供了 API，用于开发新的应用和在系统中集成新的功能。EyeOS 应用程序的
364 服务器端组件至少由两个文件（存储在 eyeos/apps/appname 目录）所定义：appname.php 和 appname.js。第一个文件定义和实现了所有应用程序公开的操作；JavaScript 文件包含需要在浏览器中加载的代码，以供用户和应用程序交互。

Xcerion XML Internet OS/3（XIOS/3）是另一个 Web 桌面环境的例子。服务作为 CloudMe 应用程序的一部分发布，CloudMe 是一个云文件存储解决方案。XIOS/3 与其他服务的主要区别是其 XML 强大的影响力，它实现了许多操作系统的任务：绘制用户界面、定义应用程序的业务逻辑、结构化文件系统的组织甚至应用程序并发。OS 架构在客户端集中了大部分功能，由 XML Web 服务实施基于服务器的功能。在客户端呈现用户界面、安排流程、提供 XML 数据的数据绑定功能，该功能通过 Web 服务交换 XML 数据。服务器负责实现核心功能，例如协作模式文档编辑事务管理和在环境中安装应用程序的核心逻辑。XIOS/3 还提供了一个环境，用于开发应用程序（XIDE），它允许用户通过可视化工具快速开发复杂的应用程序，用于用户界面和 XML 业务逻辑文档。

⊖ www.eyeos.org。

XIOS/3 作为开源软件发布，实现了一个交易市场，第三方可以轻松地部署应用程序，将其安装在虚拟桌面环境的顶部。它可以开发任何类型的应用程序，并提供通过 XML Web 服务访问的数据；开发人员必须定义用户界面，绑定 UI 组件到服务调用和操作，并提供数据处理逻辑。XIDE 将这些信息打包到一个适当的 XML 文档集，其余的将在 XIOS 实现一个 XML 的虚拟机来执行。

XIOS/3 是一个先进的 Web 桌面环境，通过基于 XML 的服务致力于将服务整合到环境中，简化同伴的协作。

10.2.3 社交网络

过去几年，社交网络应用程序大幅度增长，成为网络上最活跃的网站。为了维持通信，无缝地服务数百万用户，Twitter 和 Facebook 等服务已经利用了云计算技术。而系统运行中不断增加容量的可能性是社交网络最吸引人的特点，这使得网站的用户群不断增加。

1. Facebook

Facebook 可能是最出名和最有趣的社交网络环境。它拥有超过 800 万用户，已成为世界上最大的网站之一。为了维持令人难以置信的增长，Facebook 的首要工作是不断增加容量和开发新的可扩展的技术和软件系统，同时保持较高的性能以确保流畅的用户体验。

目前，社交网络有两个数据中心备份，这两个数据中心已通过优化来减少成本和对环境的影响。在这种高效的基础设施之上，又建造和设计了廉价的硬件，利用适当修改和完善的开源技术，完全自定义的协议栈构成了最大的社交网络的后台。综合起来，这些技术构成了一个开 [365]
发云应用程序的强大的平台。该平台主要支持 Facebook 自身，也提供 API 被合第三方应用程序与 Facebook 的核心基础设施，以提供额外的服务，如社交游戏和其他人创建的线上测验。

Facebook 的参考模型栈基于 LAMP（Linux、Apache、MySQL 和 PHP）。这组技术又结合了内部开发的其他一系列服务，这些服务由不同的语言开发，实现搜索、新闻、通知等具体功能。当处理页面请求时，用户的社交图就形成了。社交图以相互关联的一组信息标识，与给定用户相关。大部分的用户数据由查询 MySQL 实例的分布式集群获得，MySQL 实例大多包含键 – 值对。然后这些数据被缓存以便快速检索。相关信息的其余部分，使用前面提到的服务组合在一起。这些服务在位置上更接近数据，而且多语言开发提供了比 PHP 更好的性能。

一组内部开发的工具使服务开发更方便。其中 Thrift 是一个核心元素，它是抽象（和语言绑定）的集合，允许跨语言开发。Thrift 能用不同的语言开发服务以方便通信和交换数据。不同语言绑定的 Thrift 可处理数据序列化和反序列化、通信、客户端和服务器的样板代码。这简化了开发人员的工作，能够实现快速原型服务，并充分利用现有服务。其他的相关服务和工具是 Scribe，它汇集流日志作为输入、应用程序报警和监控。

10.2.4 媒体应用

媒体应用程序是一个有利可图的市场，它利用云计算技术获得了相当的优势。特别是视频处理操作，如编码、转码、合成和渲染，更适合采用云计算环境。这些计算密集型任务可以很容易地转移到云计算基础设施完成。

1. Animoto

Animoto[⊖]也许是云媒体应用中最流行的例子。该网站为用户提供了一个非常简单的接

⊖ www.animoto.com。

口，用于快速创建视频，由用户提交图像、音乐和视频片段。用户选择一个视频的特定主题，上传照片和视频，并规定它们的出现顺序，选择歌曲作为配乐，然后渲染视频。该过程是在后台执行的，一旦渲染视频完成，便通过电子邮件通知用户。

Animoto 的核心价值是能快速制作效果出众的视频，而无需用户干预。一个专门的人工智能（AI）引擎根据图片和音乐选择动画和过渡效果以驱动渲染操作，用户只需要按所需的顺序组织图片和视频来定义脚本。如果用户不喜欢该结果，视频可以再次渲染，引擎会选择
366 不同的组合，从而每次都产生不同的结果。该服务允许用户免费创建 30 秒的视频。通过按月或按年支付订阅，可以产生任意长度的视频，且可供选择的模板更广泛。

支持 Animoto 的基础设施是复杂的，它由不同的系统组成，且所有的系统都需要扩展（见图 10-8）。其核心功能是在亚马逊 Web 服务基础设施之上实现的。特别是使用了亚马逊 EC2 作为 Web 前端和工作节点，亚马逊 S3 用于存储图片、音乐和视频，而亚马逊 SQS 用于连接所有组件。该系统的自动缩放功能通过 RightScale 管理，它负责监控负载，控制新的工作实例的建立及回收。前端节点采集制作视频所需的组件并将它们存储在 S3 中。一旦视频的故事板完成后，视频渲染请求被输入到一个 SQS 队列。工作节点获取渲染请求，并执行渲染。当处理完成时，另外一个消息进入到一个不同的 SQS 队列，另一个请求获得服务。最后一个队列通常被清除，并通知用户完成。EC2 实例的生命是由 RightScale 控制的，它持续地监控该系统的负载和性能，并决定是否有必要增大或缩小。

该系统的架构已经被证明具有非常大的可扩展性和可靠性，通过使用多达 4000 台服务
367 器，在 EC2 的高峰时段不丢弃请求，渲染过程中的暂时延迟都是可接受的。

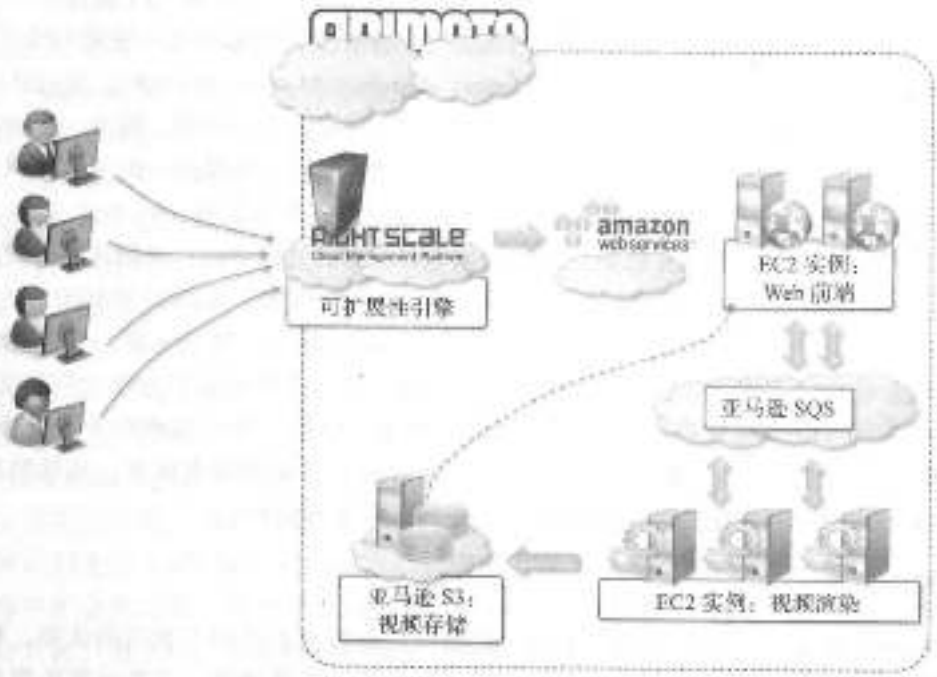


图 10-8　Animoto 参考架构

2. 用 Aneka 实现 Maya 渲染

在工程学科和影视制作行业也有有趣的媒体处理应用程序。如模型渲染操作，现在是设计流程的一个组成部分，并已成为计算需求。力学模型的可视化不仅用在设计过程结束时，

还可通过反复使用来改善设计。渲染操作需要尽可能快地完成任务，云计算为工程师提供了必需的计算能力来实现这一目标。

一个用来渲染列车设计的私有云解决方案已经由中国南车集团株州公司实施（见图 10-9）。该部门负责设计高速电力机车、地铁车辆、城市交通车辆和电机列车模型。设计过程中的原型要求高品质和三维（3D）图像。分析这些图像可以帮助工程师发现问题并改正自己的设计。三维渲染任务需要花费大量的时间，尤其是在帧的数量巨大的情况下，减少这些迭代所花费的时间对工程部非常关键。这个目标已经利用云计算技术实现了。云计算技术将该部门原来的桌面网络变成一个由 Aneka 管理的桌面云。所实现的系统包括一个株州工程师可使用的进入渲染 368
过程（帧的数目、摄像机的数量以及其他参数）的所有细节的专用客户端接口。该应用程序将渲染任务提交到 Aneka 云，通过所有可用的机器分担负载。每一个渲染任务触发本地 Maya 批渲染的执行并收集执行的结果。然后检索该渲染并把所有的渲染集中起来可视化。

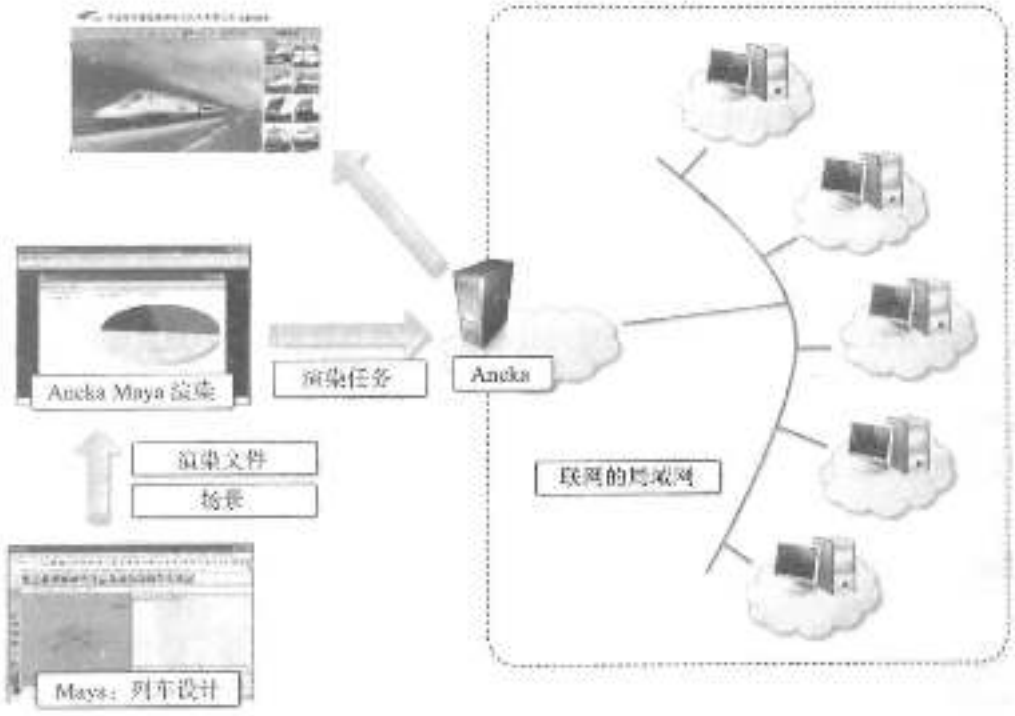


图 10-9 私有云上的渲染

通过将本地网络转换成私有云，它的资源可以用于非高峰期（即晚上或台式机没有使用时），可显著地减少渲染过程中所花费的时间，使其从几天缩短到几小时。

3. 云上视频编码：Encoding.com

视频编码和转码操作可以极大地从云技术中获益：因为这些操作是计算密集型的，并可能需要大量的存储空间。此外，随着移动设备性能的不断提高以及互联网的普及，对视频内容的请求显著增加。具有视频回放能力的不同设备需要不同的视频传播格式。进行视频编码和转码的软件和硬件往往成本高昂或不能灵活地支持将任何格式转换到需要的任何格式。目前云技术能将这些繁琐而往往要求苛刻的任务转换成服务，很容易地集成到各种工作流程，或根据需要提供给用户。

Encoding.com 是一个软件解决方案，提供点播视频转码服务，并利用云计算技术以提

供所需的视频转换能力和分期视频存储。该服务整合了亚马逊网络服务技术（EC2、S3 和 CloudFront）和 Rackspace（云服务器、云文件和 Limelight CDN 访问）。用户可以通过多种接口访问服务：Encoding.com 网站、Web 服务的 XML API、桌面应用程序和监视文件夹。使用该服务时，用户必须指定需转码视频的位置、目标格式和视频的目标位置。Encoding.com 还提供了其他视频编辑操作，如插入缩略图、水印或标志。此外，它还扩展支持音频和图像的转换。

该服务提供了各种定价方案：月租费、即付即用（分批）以及大批量的特殊价格。Encoding.com 现拥有 2000 多家客户，已处理超过 10 万个视频。

10.2.5　多人在线游戏

多人在线游戏吸引着世界各地的几百万玩家，玩家们在超出普通局域网范围的虚拟环境中一起玩游戏并分享经验。在线游戏在同一时段可扶持数百玩家，通过用于转发交互的特定架构实现，基于游戏日志处理完成。玩家更新托管游戏的游戏服务器，而服务器将所有更新集成到日志中，这样能通过 TCP 端口提供给所有玩家。客户端软件用于将游戏连接到日志
369 端口，通过读日志，根据其他玩家的动作更新本地的用户界面。

游戏日志处理用于建立玩家统计数据并进行排名。这些特征构成网络游戏门户网站吸引更多玩家的额外价值。游戏日志的处理是一个潜在的计算密集型操作，在很大程度上取决于监控的玩家数量和游戏次数。此外，游戏门户网站是 Web 应用程序，可能会遭受到用户的突发行为，随机产生大量不稳定的且不满足能力计划的工作负载。

使用云计算技术可以提供弹性，无缝地处理这些工作负载和当用户数量增加时所需的规模。一个基于云的游戏日志处理原型已经由 Titan 公司（现在是 Xfire 公司）实现。其总部设在加利福尼亚，公司扩展了游戏门户网站以将游戏日志处理转移到 Aneka 云。该原型（见图 10-10）使用私有云部署，允许 Titan 公司并发处理多个日志和维持更多的用户。

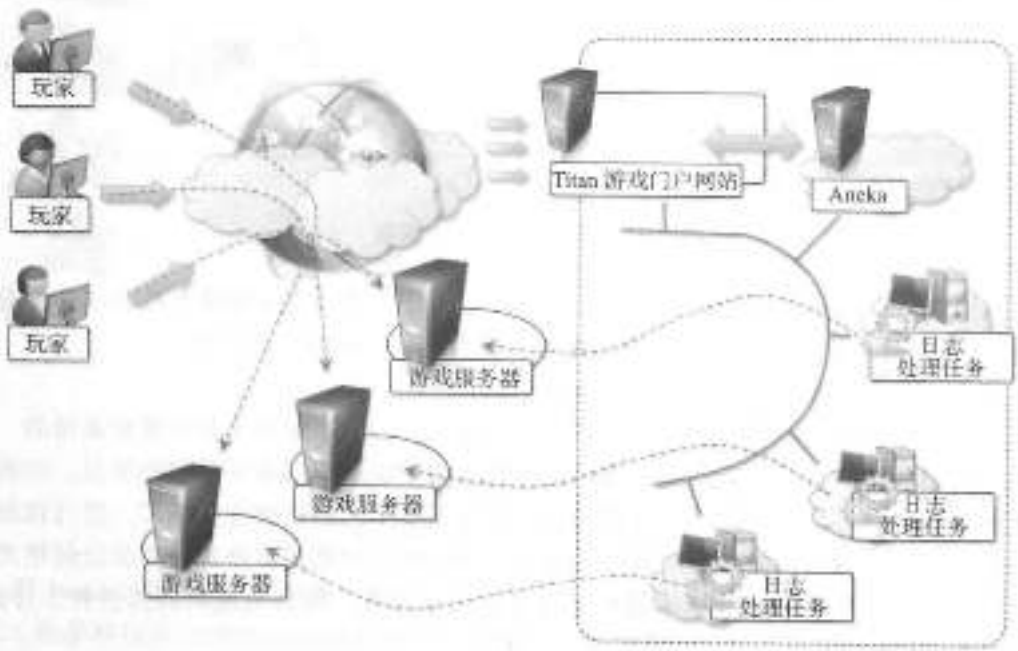


图 10-10　网络游戏的可扩展日志处理

本章小结

本章简要概述了为云开发的应用程序或利用某种形式的云技术的应用程序。这些应用程序涵盖了不同的应用领域，从科学到商业和消费者应用程序都可以利用云计算。 370

科学应用从云环境的弹性和可扩展性得到巨大的好处，还提供按需定制服务以部署和执行科学实验。商业和消费者应用程序可以利用其他几个特点：云 CRM 和 ERP 应用中可以减少甚至消除硬件管理、系统管理和软件升级的维护成本。此外，云无处不在，可以从任何设备和任何地方访问。效率型应用程序，如办公自动化产品，不仅允许从任何地方访问文档，还能修改文档，设备之间的文件不需要复制。视频编码等媒体应用程序可以将冗长和计算密集型任务编码转移到云端。社交网络能利用不断增加的容量，不会有重大的服务中断，可维持预期的性能水平。

这些新的机会已经改变了我们在日常生活中使用这些应用的方式，但也对开发者提出了新的挑战，他们不得不重新考虑设计方式，以从弹性、可扩展性、按需的资源配置和无所不在的特性中收益更多。这也是云技术在一些领域成为一个有吸引力的解决方案的重要特征。

习题

1. 什么类型的应用程序可以从云计算获益？
2. 云技术给科学应用带来的根本优势是什么？
3. 描述云计算技术如何用于支持远程心电监护。
4. 描述云计算技术在生物领域的一个应用。
5. 云计算在地球科学领域有什么优势？用一个例子来说明。
6. 描述基于云计算技术的 CRM 和 ERP 的一些例子。
7. 什么是 Salesforce.com？
8. 什么是 Dropbox 和 iCloud？它们利用云技术解决哪类问题？
9. 描述谷歌 Apps 的重要特征。
10. 什么是 Web 桌面？它们和云计算有什么关系？
11. 对社交网络应用，云计算技术最重要的优势是什么？
12. 给出一些使用云技术的媒体应用程序的例子。 371 ~ 372
13. 描述一个用于网络游戏的云技术的应用。

第 11 章 |
Mastering Cloud Computing: Foundations and Applications Programming

云计算高级主题

云计算发展迅速，新的技术不断进步，新的应用服务不断推出。尤其是在云计算市场和数据中心节能管理的背景下，有许多开放的挑战。

本章概述了云计算需要长期研究的各种开放问题。讨论的主题涉及高能效的云计算问题，并提出了一个“绿色”的云计算架构。还从云联盟、云和云之间协议的角度，讨论了利用云计算系统实现开放市场的市场模型。最后概述了一些实现云之间互操作功能的现有标准，并简单介绍了第三方云服务。

11.1 云能效

云计算模型下操作的现代数据中心承载了各种应用，包括那些运行几秒钟（例如，电子商务和社交网络门户等 Web 应用程序的服务请求）和那些在共享硬件平台运行时间较长（例如，仿真或大数据集的处理）的应用。在一个数据中心管理多个应用程序的需求对资源按需配置和分配随时间变化的工作负载提出了挑战。通常情况下，基于峰值负载特性静态地为应用程序分配数据中心的资源，以保持独立性并提供性能保证。直到最近，高性能成为数据中心部署的唯一关注，这种需求已经得到满足，但十分不重视电力消耗。根据麦肯锡关于“彻底改变数据中心电力效率”[118]的报告，一个典型的数据中心消耗的能量多达 25000 户。供给一个典型数据中心的电力成本每五年增加一倍。由于电力成本不断增加，而供应不断下降，因此有必要单独从性能角度优化数据中心资源管理，将重点转移到优化电力效率方面，同时保持高水平的服务性能（见图 11-1）。

数据中心不仅维护费用昂贵，且会对环境产生不利影响。世界各地的数据中心的碳排放量已经超过阿根廷和荷兰的排放量总和 [118]。产生高电力成本和巨大碳排放量的原因是需
373 要大规模供电量发动和冷却托管在这些数据中心的大量服务器。云服务供应商需要采取措施，以确保他们的利润空间不会因为高电力成本而显著降低。据亚马逊的估计，其数据中心的电力相关成本达到总预算的 42%，其中既包括直接的功耗也包括平分在 15 年间的冷却基础设施的费用。因此，谷歌、微软和雅虎公司正在荒芜的沙漠上建设大型数据中心，围绕美国哥伦比亚河利用廉价的水力来发电。来自世界各国政府的减少碳排放的压力越来越大，因为碳对气候变化产生了重要影响。为了解决这些问题，全球领先的 IT 厂商最近成立了一个全球联盟，称为绿色网格组织，以提高数据中心的电力效率，并最大限度地减少其对环境的影响。Pike 研究公司预测，采用云计算模式提供 IT 服务，全球数据中心的电力支出会从 2010 年的 233 亿美元降低至 2020 年的 160 亿美元，产生温室气体（GHG）将比 2010 年的排放量减少 28%。

降低数据中心的电力使用是一个具有挑战性和复杂的问题，因为计算应用程序和数据的增长如此迅速，在给定时间内的以足够快的速度来处理它们，需要更大的服务器和磁盘。绿
374 色云计算设想的实现不仅要有效地处理和利用计算基础设施，而且要最大限度地减少电力消

耗。这点对确保云计算未来的可持续增长至关重要。随着日益普遍的前台客户端设备，如 iPhone 手机与后端数据中心进行交互，云计算将导致电力消耗的巨大升级。为了解决这个问题，数据中心资源需要以高能效的方式进行管理，以带动绿色云计算。特别是，云资源分配不仅要满足用户通过服务水平协议（SLA）指定的 QoS 要求，而且还要减少电力消耗。可以通过应用基于市场的实用模型来接受用户的请求，既增加了收入，同时也可以提高云基础设施的能效利用率。

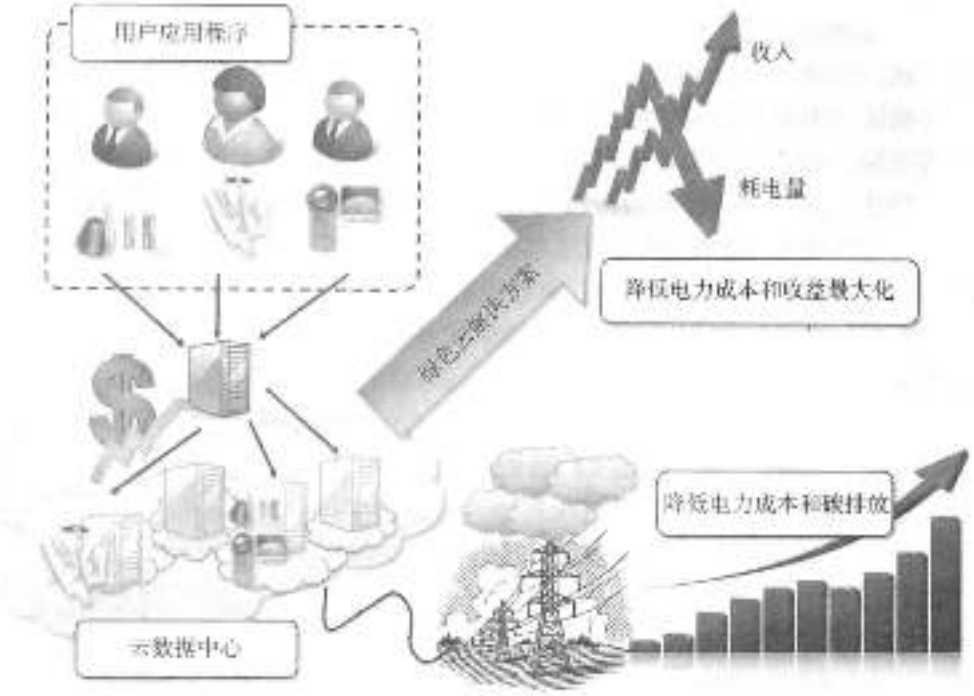


图 11-1　“绿色”云计算方案

节能和绿色云计算架构

在一个绿色的云计算基础设施内支持能效资源分配的高层次架构如图 11-2 所示。它由四个主要部分组成：

- 消费者 / 代理。云消费者或其代理从世界上任何地方向云提交服务请求。注意，云消费者和部署服务的用户之间是有差异的。例如，消费者可以是部署 Web 应用程序的 [375]
  公司，根据“用户”访问数目呈现不同的工作负载。
- 绿色资源分配器。作为云计算的基础设施和消费者之间的接口，需要以下组件来支持高效节能资源管理的交互：
  - 绿色谈判。根据消费者的 QoS 要求和节能方案，与消费者 / 代理协商敲定云服务供应商和消费者之间规定的价格和处罚（违反 SLA）。例如，在 Web 应用中，QoS 的度量可以是 95% 的请求的服务时间小于 3 秒。
  - 服务分析软件。在决定是否接受或拒绝要求之前解释和分析一个提交请求的服务。因此，它分别需要来自虚拟机管理器和电力监测仪的最新负载和电力信息。
  - 消费者侦查。收集消费者的具体特征。重要的消费者可以被授予特权，优先于其他消费者。

- 定价。决定服务请求如何收费，管理计算资源的供给和需求，并有效地促进服务优先次序分配。
- 电力监测仪。观察并确定开启或关闭哪些物理机器的电源。
- 服务调度。为虚拟机分配请求，并确定虚拟机的权利。决定何时添加或删除虚拟机以满足需求。
- 虚拟机管理器。跟踪虚拟机及其资源配额的可用性，也负责跨物理机器迁移虚拟机。
- 记账。维护请求资源的实际使用情况，计算使用成本。历史使用信息也可以用来改善服务分配决策。

- 虚拟机。为满足多个请求，一台物理机器可以动态地启动和停止多个虚拟机，根据服务请求的不同具体要求在同一台物理机器上配置资源的不同分区，从而提供最大的灵活性。单个物理机器上不同的操作系统环境下，多个虚拟机还可以并发地运行应用程序。此外，通过在物理机器间动态迁移虚拟机，工作负载可以整合，未使用的资源可以处于一个低功耗状态、关闭或者配置在低性能水平运行（例如，使用动态电压和频率缩放，即 DVFS），以节省电力。
- 物理机器。底层的物理计算服务器提供硬件基础设施，用于创建虚拟化资源，以满足服务需求。

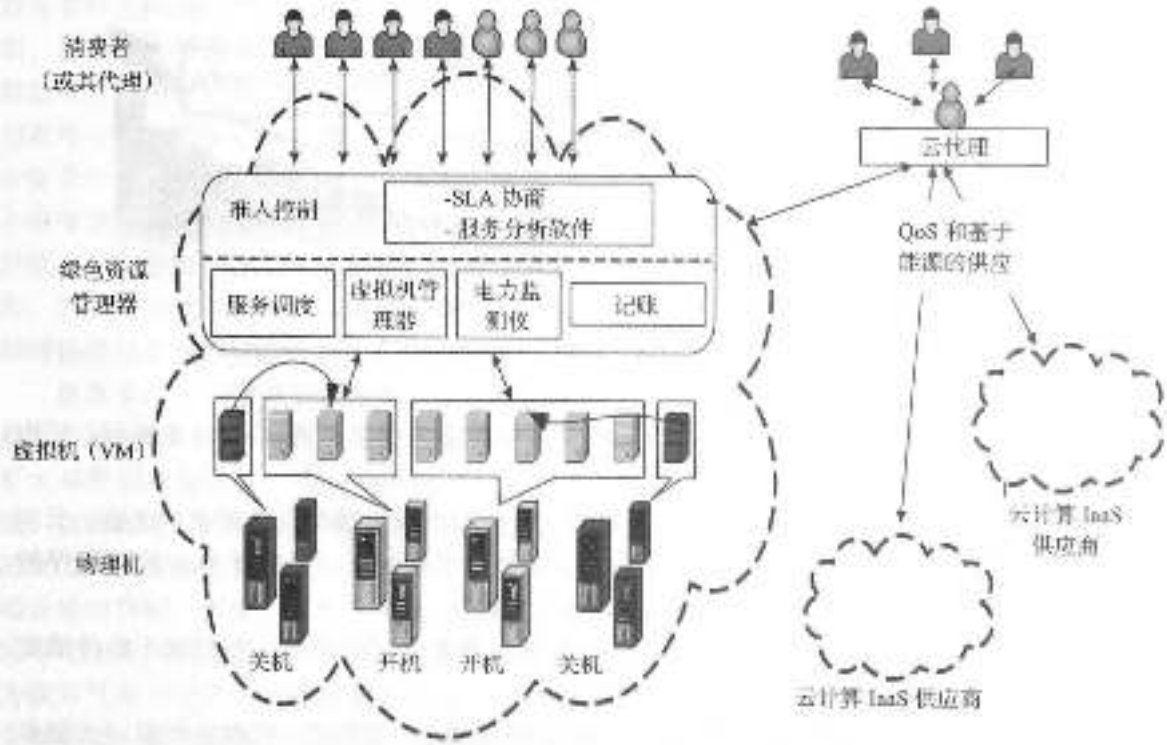


图 11-2　绿色云计算高层次的系统架构框架

1. 能量感知的动态资源分配

最近虚拟化方面的发展已使其可以跨数据中心使用。虚拟化使根据 QoS 要求部署的虚拟机在物理节点间能动态迁移。空闲的虚拟机可以在逻辑上调整和整理至最少的物理节点数，而闲置的节点可以关闭（或休眠）。通过整合虚拟机，大量用户可以共享一台物理服务

器，从而提高利用率，又减少了所需服务器的数量。此外，虚拟机的整合可以通过动态捕捉工作负载的变化和使用迁移在运行时适应虚拟机的配置。

376

目前，云数据中心资源分配的目的是提供高性能的同时使虚拟机分配满足 SLA，很少或没有考虑电力消耗。然而，探讨性能和能效，有两个关键问题必须得到解决。首先，在动态环境中关闭资源需要面临 QoS 下降的风险，积极整合可能会导致某些虚拟机获得的资源不足以服务于顶峰负载。第二，接受 SLA 为虚拟化环境应用程序性能管理带来了挑战。这些问题都需要有效地整合策略，以最大限度地减少电力的使用而不会影响用户的 QoS 要求。目前的动态虚拟机整合方法不能提供性能保证。一个证明性能界限的方式是把高能效的 VM 动态合并的问题分解成可以分别进行分析的几个子问题。通过分析对问题建模，得出最优及接近最优的近似算法，并提供可证明的效率，这是很重要的。为了实现这一目标，云需要新的分析模型和基于 QoS 的资源分配算法，优化虚拟机布置，目标是在性能约束下最小化电力消耗。

2. 互联云和资源整合配置

云服务供应商已经在遍布全球的多个地点部署数据中心。例如，亚马逊 EC2 云服务通过设在美国、欧洲和新加坡的数据中心提供服务。这种分布产生一个概念，称为互联云（InterCloud），一个或多个供应商可利用多个数据中心发布可扩展应用服务。除了提高性能和可靠性，InterCloud 提供了强有力的手段来减少电力相关成本。原因之一是，对电力的本地需求随每天的时间和天气而变化。这会导致每个地方电力价格随时间而变化，而且每个站点的电力来源不同（例如煤、水力发电或风），具有不同的环境成本。因此可以调整发送到每个地方的负载和每个地方通电服务器数目，以提高效率。

在这样的环境中，需考虑用户的位置，每一个位置硬件的电力效率、电力结构，以及当前每个位置需要的服务器的数目，综合这些因素做出路由决策算法。一个特别有希望的方法就是用这个路由使工作“遵循可再生能源”。可再生能源的一个主要问题是，大部分来源是间歇性的和不可控制的。可再生能源到站点的动态路由请求可以大大减少不可再生能源的使用，有利于清洁能源的推广。

发送负载到远程数据中心导致延误成本和能源成本，原因是在网络上增加了大量数据传送。改进能源效率传输技术应该能显著减少云软件服务的功耗 [120]。

11.2　基于市场的云管理

云计算尚处于起步阶段，其突出的用途是双重的：完全替代内部的 IT 基础设施和服务，租用服务供应商的服务以实现相同的功能；现有计算系统的弹性扩展，以解决高峰期的工作负荷。研究机构和业界大多在设计和实施系统方面不断努力，实际上应让业务供应商和企业来实现这些目标。云计算的真正潜力在于，它实际上促进了 IT 公共基础设施交易市场的建立。目前已经有一些此方面的研究，称为面向市场的云计算 [30]。

377

11.2.1　面向市场的云计算

云计算已经体现出 IT 资产作为公共基础设施服务的概念。那么，云计算与面向市场的云计算有什么不同？首先，很重要的一点是要理解市场的概念。牛津英语词典（OED）[⊖]定义市场为“进行贸易的地方”（定义 1）。更确切地说，市场指的是人们聚集在一起购买和销售商品。

⊖ 根据牛津英语词典的定义，市场可以在 www.oed.com/view/Entry/114178?rskey=13s2al&result=1#eid 找到（2011 年 7 月 5 日检索）。

一个更广泛的市场定义为：买入和卖出的行为、商业交易、购买或讨价还价。因此，市场这个词主要是指在环境中进行交易的行为，无论是物理环境或虚拟环境都专门致力于此类活动。

如果考虑将IT资产和服务作为公共基础设施服务消费的方式，很明显，服务供应商和消费者之间有交易，这使得在特定的SLA下，用户能使用该服务。因此，云计算已经表达了贸易的概念，尽管消费者和供应商之间的交互不像真正的市场中那么复杂：用户通常从一组相互竞争的供应商中选择一个云计算供应商，只要他们需要就利用其服务。而且，目前大多数服务供应商定价都不够灵活，一般仅限于基于统一费用或使用阈值收费。此外，许多供应商的服务有专用接口，从而限制了消费者以最小的转换成本从一个供应商快速转移到另一个供应商。这种死板规定称为厂商绑定，破坏了云计算作为一个开放市场服务的自由流通潜力。因此，为了取消这些限制，要求供应商通过标准接口公开服务。这使得全面商品化得以实现，从而为交易服务创建市场基础设施做好准备。

面向市场的云计算（MOCC）和云计算的区别就是存在一个动态交易和代理IT服务的虚拟市场。这种方式仍有待实现并且将显著提高为消费者提供服务的云计算服务方式。更确切地说，缺少的是云服务交易市场，在该市场中能够发布需要的服务，然后通过匹配用户
[378] 和供应商的要求自动竞价。目前，一些云计算厂商已经在朝这个方向努力[⊖]，IaaS领域已经出现了更加统一和成熟的云计算市场，但一直没有完全成功。我们可以清楚地描绘云计算和MOCC之间的关系如下：

> 面向市场的计算具有与云计算相同的特征，即允许动态地支付计算资源，让用户管理软件和数据存储，如同在跨公共和私有基础设施而形成的“实时”的大容量基础设施上执行。MOCC更进一步地扩展到由交易服务动态组成的多个公共和混合云环境。[122]

这一构想的实现在技术上是可行的，但由于缺乏标准和市场整体不成熟，现在还不可能实现。尽管如此，预计在不久的将来，随着标准的出台，不必再担心安全和信任问题，企业会更加便利地利用市场化模式来整合云IT基础设施和服务。此外，基于需求的市场使企业有机会构建其基础设施，以便动态应对工作负载高峰和减少维护成本。暂时租赁低使用率的一些内部设施，将会更好地回报投资。这一趋势将彻底实现市场化的云计算。

11.2.2　MOCC参考模型

市场化云计算起源于几个组件的协调：服务消费者和服务供应商，以及其他使这两个群体之间可能交易的实体。市场定位不仅影响了云计算市场的全球规模的组织，也形成了云计算供应商的内部架构。供应商需要更灵活地分配资源，这是由那些定义服务质量的附加属性参数所驱动的。

1. 面向市场云计算的全局视图

图11-3给出了在全球范围内实现MOCC的参考方案，是关于MOCC如何在实践中实现的指导。

定义全球面向市场云计算架构包括几个组件和实体。基本组成部分是虚拟市场，表示为云交易所（CEx），将服务的生产者和消费者组织在一起。虚拟市场上的主要参与者是云协调商和云代理。云协调商代表云服务供应商，发布供应商提供的服务信息。云代理代表消费

⊖ 亚马逊提出了即时实例的概念，供应商根据自己的能力动态提供服务，由客户竞标。这些有效使用和消费之后会由亚马逊建立的即时价格和客户提供的最高价格决定。

者操作，根据服务描述和服务质量要求发现满足客户要求的服务。代理执行与现实世界中相同的功能，是协调商和消费者之间媒介，从前者获得服务，并将其转租给后者。代理可以接受来自许多用户的请求，与此同时，用户可以利用不同的代理。在协调商和云计算服务供应商之间也存在类似的关系。协调商代表厂商负责服务发布及广告，并能从转售服务给代理获利。每一个单独的参与者都有自己的效用函数，他们都希望最优化投资回报。协商和交易在一个安全可靠的环境中进行，大多是由服务等级协议（SLA）推动的，每一方都必须履行。 379
实体之间的协商模型可能不同，尽管拍卖模式似乎更适合于目前的情况。定价模式同样可以考虑，价格可以是固定的，但期望价格可能会根据市场情况改变。

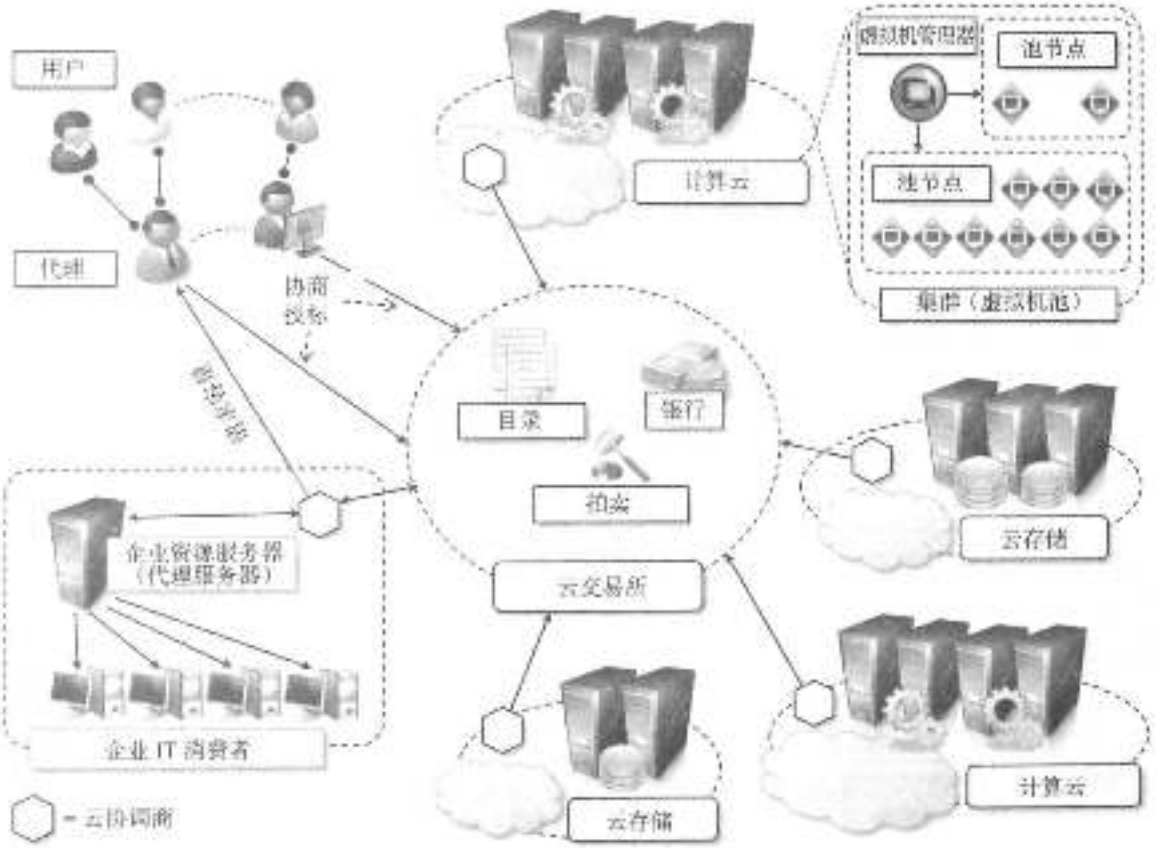


图 11-3　市场化的云计算方案

几个组件有助于实现云交易所并实施其功能。图 11-3 中给出的参考模型可以分为三个主要组件：

- 目录。市场目录包含所有已发布的服务在云市场中的可用列表。该目录不仅包含服务的名称和相应的提供服务的供应商（或云协调商）之间的简单映射，还提供了额外的元数据，可以帮助代理或终端用户从感兴趣的服务中筛选出那些能够真正满足所要求的服务质量的服务。此外，目录还提供一些索引方法，根据各种条件优化服务的发现。目录由服务供应商修改内容，供服务消费者查询。
- 拍卖商。拍卖商负责记录市场上进行的拍卖并核实拍卖服务可正常进行，禁止恶意的市场参与者进行非法活动。
- 银行。银行组件负责在虚拟市场发生的所有业务的财务方面。它还确保所有的金融

交易都在一个安全可靠的环境下进行。消费者和供应商可以在银行注册，并有一个或多个可在虚拟市场执行交易的账户。

上面所描述的组织只构成了一个参考模型，用来帮助系统架构师和设计人员设计一个基础云交易系统。在现实中，系统的架构会更加错综复杂，因为必须考虑其他因素。例如，由于云计算市场支持交易，这最终涉及不同当事人之间的金融交易，因此安全就变得至关重要。安全的电子交易机制非常重要，不只是 MOCC 系统的设计和实现会涉及这些方面，任何分布式计算系统都会涉及，这里不再赘述。

2. 数据中心的面向市场架构

数据中心是计算基础设施的组件，备份云计算供应商提供的所有类型（IaaS、PaaS 或 SaaS）的服务。本节介绍这些系统是因为它们是实现支持 MOCC 计算基础设施的基本元素。这些标准管理系统的逻辑组织，而不是它们的物理布局和硬件特性，以市场为导向指导设计架构。换句话说，为 MOCC 数据中心描述了一个参考架构。

380
|
381

图 11-4 提供了组件的整体视图，该架构支持云计算供应商使其在服务市场化基础上可用 [123]。具体地说，该模型适用于 PaaS 和 IaaS 供应商，明确地利用虚拟化技术来满足客户的需求。该架构有四个主要组成部分：

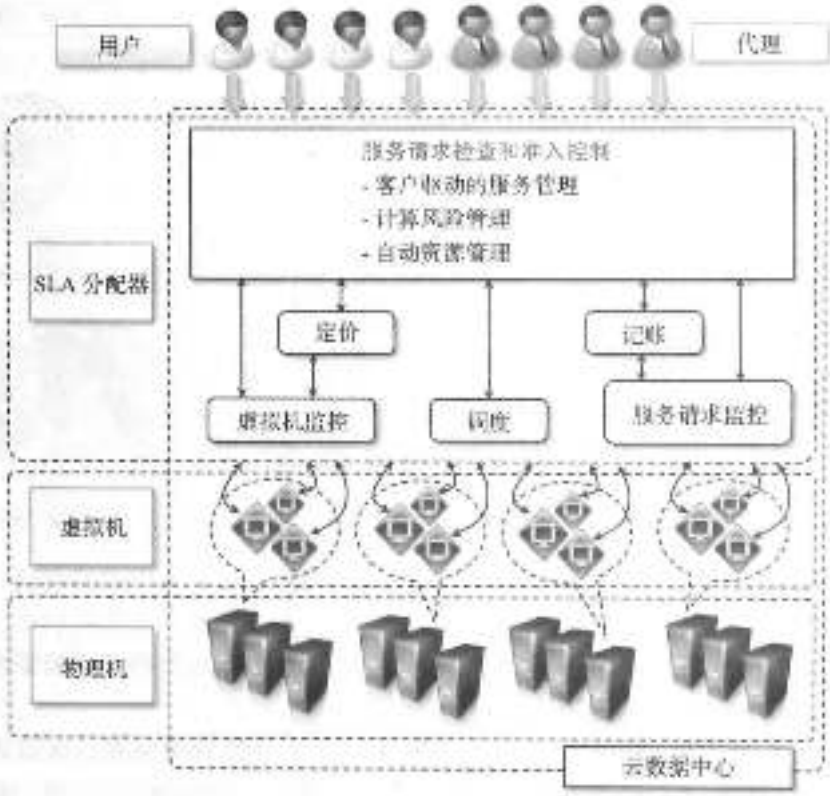


图 11-4　云数据中心参考架构

- 用户和代理。他们发起在云数据中心管理的工作负荷。用户要么需要虚拟机实例来部署系统（IaaS 的情况下），要么在供应商提供的虚拟环境中部署应用程序（PaaS 的情况下）。这些服务请求由代表用户的服务经纪公司发布，并为他们寻找最好的交易。
- SLA 资源分配器。分配器代表了数据中心和云服务供应商和外部世界之间的接口。它的主要职责是确保服务请求按照 SLA 约定让用户满意。为达到这一目标，几个组

件协调分配器活动：

- 服务请求检查和准入控制模块。该模块工作在前台，过滤用户和代理请求，以接受系统和已处理工作负荷的当前状态下可行的请求。接受的请求会分配，然后调度执行。IaaS 服务供应商分配一个或多个虚拟机实例给用户使用。PaaS 供应商确定一个合适的计算节点集合在其上部署用户的应用程序。 [382]
- 定价模块。该模块负责根据用户签署的 SLA 向用户收费。用户收费根据不同的参数，例如，通常情况下，IaaS 供应商根据虚拟机的特性收费，按照请求的内存、磁盘大小、计算能力以及使用时间。常见的是按一个小时的使用时间来计算，但也有其他几种定价机制。PaaS 供应商可以根据应用程序服务的请求数，或应用程序运行时提供的开发平台的内部服务使用量向用户收费。
- 记账模块。该模块保存资源使用情况的实际信息，为每个用户存储计费信息。评估用户的请求时，这些数据提供给服务请求检查和准入控制模块。此外，这些数据构成了丰富的信息来源，这些信息可进一步挖掘，用于识别使用趋势和改进供应商提供的服务。
- 调度。这个组件负责实现允许的服务请求所需要的低级操作。在 IaaS 方案中，该模块命令基础设施部署尽可能多的虚拟机以满足用户的请求。在 PaaS 方案中，该模块将激活并在选定的一组节点上部署用户的应用程序。可以在一个虚拟机实例部署，或在一个适当的沙箱环境中部署。
- 资源监控。该组件监控计算资源的状态，无论是物理或虚拟资源。IaaS 供应商主要关注跟踪虚拟机的可用性及其资源的权利。PaaS 供应商监控分布式中间件的状态，从而使应用程序弹性执行和装载每个节点。
- 服务请求监控。该组件跟踪服务请求的执行进度。通过服务请求监控器收集的信息帮助分析系统性能和提供有关供应商能否满足请求的质量反馈。例如，服务请求监控感兴趣的内容是满足的请求数而不是收到的请求数，以及请求的平均处理时间或执行时间。这些数据是调节系统的重要信息来源。

  SLA 分配器执行主要逻辑，主要逻辑管辖一个单一的数据中心或数据中心集合的操作。故障管理功能是最有可能被其他软件模块处理的，它可以是一个单独的层，或者和 SLA 资源分配器集成。

- 虚拟机（VM）。虚拟机构成云计算基础设施的基本构件，特别是对于 IaaS 供应商。虚拟机表示处理用户请求的部署单位。基础设施管理软件负责维护计算基础设施的运营，支持供应商的商业服务产品。正如我们所讨论的，虚拟机在为用户应用提供适当的托管环境方面起到重要作用，同时，应用程序执行时从基础设施隔离，可防止应用程序损害托管环境。此外，虚拟机是影响服务用户请求的 QoS 的最重要的部件之一。模拟硬件特性方面，虚拟机可以调整，使分配给用户的物理硬件的计算资源量可精确控制。PaaS 的供应商不直接向终端用户公开虚拟机，但他们可以在内部利用虚拟化技术，以充分和安全地利用自己的基础设施。如前面所讨论的，PaaS 的供应商通常利用指定的中间件执行用户应用程序，并满足应用执行的不同的 QoS 参数要求，而不只是模拟硬件配置。 [383]
- 物理机。在参考架构的最低层驻留着物理基础设施，可包含一个或多个数据中心。本层提供满足服务需求的资源。

此架构为云服务的厂商提供了一个参考模型，使他们的基础设施适合面向市场的计算（MOC）。如前所述，这些研究主要适用于 PaaS 和 IaaS 供应商，而 SaaS 厂商在更高的抽象层操作。但仍可以识别一些 SLA 资源分配器元素，对其进行修改以处理供应商提供的服务。例如，不链接用户请求到虚拟机实例和平台节点，分配器在供应商的 SaaS 框架内将主要关注调度请求的执行，并在技术栈较低层负责控制计算基础设施。记账、定价和服务请求监控仍将履行其职责。

不管哪种具体的服务产品类别，这里讨论的参考架构旨在支持云计算厂商提供商业解决方案，能够 [123]：

- 基于客户描述及所提出的服务需求支持客户驱动的服务管理。
- 定义计算的风险管理策略，以识别、评估和管理风险，包括应用程序执行有关服务需求和客户需求。
- 制定适当的基于市场的资源管理策略，既包括客户驱动的服务管理，也包括计算的风险管理，以保持面向 SLA 的资源分配。
- 合并自主资源管理模式，有效地自我管理变化的服务要求，同时满足新的业务需求和现有服务的职责。
- 在适当的时候充分利用虚拟机技术，根据服务需要动态分配资源。

这些功能在云计算市场具有至关重要的竞争力，用于解决 MOCC 设想的动态 SLA 协议特征的场景。目前，不存在或很少有支持该动态协议的模型，这一设想是完全实现云计算的下一个步骤。

11.2.3　支持 MOCC 的技术和实现

384

现有的云计算解决方案对以面向市场策略向用户提供服务的支持非常有限。目前大多数解决方案主要侧重于云计算基础设施、分布式运行时环境以及服务的交付。由于云计算最近已经被广泛采纳，因此技术的整合是全面实现云计算的第一步。到现在为止，多数交易针对的是 IaaS 解决方案，这是云计算市场的稳固部分，有一些不同的参与者和竞争者。新的 PaaS 解决方案势头很猛，但很难渗透谷歌、微软和 Force.com 等巨头主导的市场。

1. 交易计算基础设施框架

从学术角度已经对定义交易计算公共基础设施的模型进行了大量研究，特别集中于网格计算系统的市场化调度的设计。正如引言中所讨论的，计算网格聚集了在地理上分布的，可能属于不同组织的一组异构资源。这些资源通常是组织之间按协议方式租赁为长期使用。在此背景下，调查和实施了面向市场的调度程序，调度程序知道给定的计算资源的价格，根据预算调度用户应用程序。这方面的研究和 MOCC 有关，因为云计算充分利用了已经存在的分布式计算技术，包括网格计算。

GARG 和 Buyya[124] 提供了这种调度的完整的分类和分析，如图 11-5 所示。根据分配决策、目标、市场模型、应用模型和参与重点，将这些调度进行了主要分类。特别令人感兴趣是根据市场模型的分类，它是用于用户和供应商之间的交易机制。用这种方法可以将调度器分为以下几类：

- 博弈论。那些基于博弈论的市场模型，参与者以分配博弈的形式互动，用不同的报酬作为运用各种策略具体行动的结果。
- 按比分享。这种市场模式起源于按比例分配调度，旨在在一组资源上公平地分配工

作。这种原始的概念已经置于面向市场的方案中，其中集群的分配是和用户的出价成正比的。

- 商品市场。在该模型中，资源供应商规定资源的价格，并根据资源消耗量向用户收费。供应商决定的价格是考虑投资和管理成本、目前需求和供应后综合决策的一个结果。此外，价格可能会随时间变化。
- 发布价格。这种模式类似于小商品市场，但供应商可能对新的客户提供优惠和折扣。此外，对于商品市场，一段时间内价格是固定的。
- 合同网。在基于合同网 [125] 协议的市场模型中，用户公布他们的需求，并邀请资源拥有者进行投标。资源拥有者根据他们的要求检查这些公告。如果公告是对他们有利的，该供应商将响应投标。然后，用户将整合所有的投标并比较，选择对自己 385
最有利于的标。投标的结果将通知供应商，他们可以接受或拒绝。
- 议价。在某 1 议价的市场模型中，资源消费者和供应商之间的谈判一点进行直到相互达成协议或因任何一方不再感兴趣而终止。
- 拍卖。在基于拍卖的市场模型中，资源的价格是未知的，有竞争力的投标由第三方（拍卖商）监管，有助于确定资源的最终价格。最终设定资源价格的出价中标，相应的用户获得该资源访问权。

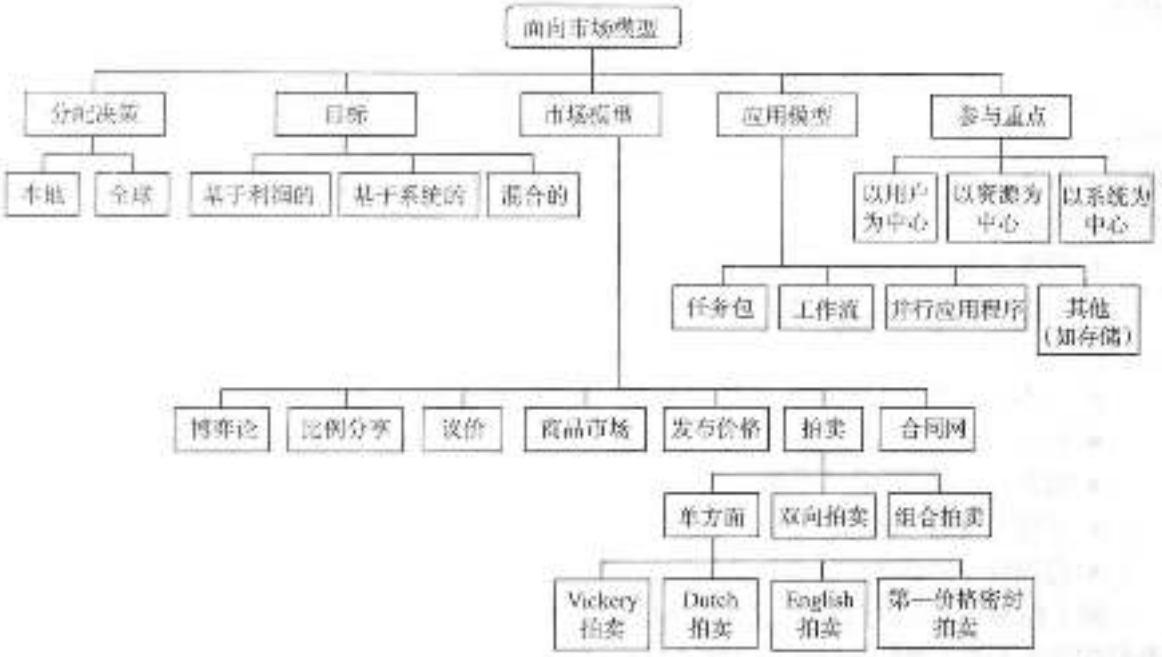


图 11-5　面向市场的调度分类

最流行和最有趣的交易计算市场模型是商品市场、发布价格以及拍卖模型。前两种模型的变化或组合推动了今天大部分的云计算服务产品。而以拍卖为基础的模型可能构成 MOCC 参考市场模型，因为它们能够无缝地支持动态协议。

学术研究已经带动了相当数量的软件项目的开发，用于实现用户资源代理或资源管理系统，能够根据前面所描述一个或多个市场模型交易计算基础设施。其中，SHARP [126]、Tycoon [127]、Bellagio[128] 和 Shirako[129] 都集中在虚拟机资源片交易，并且可能更适于 MOCC 方案。此外，值得注意的是一些相关的研究项目，侧重于资源的代理，如

Nimrod-G [164] 和 Gridbus 代理 [15]，已经能够集成功能以租用云计算资源。

2. 工业实现

尽管面向市场模型大多在学术领域进行研究，但 MOCC 的某些工业实现已经具备可用性，并且越来越受欢迎。特别是一些有趣的特性显示了 MOCC 的不同之处，如灵活的定价模式、虚拟市场和市场目录，这些都已经提供给广大公众。

（1）灵活的定价模式：亚马逊即时实例

AWS 是 IaaS 市场中最强大产品，最近推出了即时实例[⊖]的概念，它允许 EC2 客户竞标亚马逊未使用的 EC2 容量和运行这些实例，只要他们的出价超过了当前的即时价格。根据 EC2 实例的需求和供应，即时价格周期性变化，一个小时内保持不变。即时实例服务可以在任何时候终止，它们的价格通常低于传统（按需和保留）的实例价格，因为它们依靠的是 EC2 基础设施中额外的可用容量。因此，用户的责任是要定期维持即时实例中应用程序的执行状态。对亚马逊和 EC2 用户而言，即时实例代表了一个有趣的机会，双方都能从市场目前的状况中获益：供应商可以从容量获利，如果定价在正常水平容量就浪费了；而消费者有机会通过承担主要风险少付钱。

386
387

尽管价格频繁变动，但即时实例已经被证明是相当可靠的，且可用于执行具有低优先级的非紧急任务。换句话说，它们适合于可容忍 QoS 限制的应用程序。此外，还适合以较低的成本扩充现有基础设施的容量 [130]。

（2）虚拟市场：SpotCloud

SpotCloud[⊜]实现了一个虚拟市场的在线门户网站，卖家和买家可以注册和交易云计算服务。该平台是一个在 IaaS 层运作的市场。买家正在寻找能够满足他们的应用需求的计算能力，而卖家可以提供其基础设施，以服务于买家的需求而赚取收入。SpotCloud 提供了一套全面的针对虚拟市场的特征，包括：

- 所有买家的交易详细记录。
- 对任何资源性能的计量、计费。
- 完全控制资源性能的市场定价和可用性。
- 为供应商管理资源配额和利用率水平。
- 联合管理（多供应商、多客户，但只有一个平台）。
- 混合云支持（内部和外部资源管理）。
- 完整的市场管理和报告。
- 应用程序和预建的设备目录。

除了作为一个在线门户网站，由 SpotCloud 实现的虚拟市场也可以在私人场所复制。交易仍按现金并基于存款进行，一旦买卖双方创建一个账户就必须存满。

尽管有一些局限，但 SpotCloud 仍然是最具代表性的 MOCC 平台实现。SpotCloud 的工作原理是，为了加入门户网站并提供可用的计算功能，卖家需要共享一个共同的且特别的平台。SpotCloud 目前支持 Enomaly ECP[⊝]和 OpenStack[⊛]。

⊖ 在 AWS 可以看到即时实例服务产品的完整细节，参考下面的链接：http://aws.amazon.com/ec2/spot-instances/（2011 年 5 月 9 日检索）。

⊜ www.spotCloud.com。

⊝ www.enomaly.com。

⊛ www.openstack.org。

（3）市场目录：App Spot，云计算市场

SpotCloud 是作为一个托管在 AppSpot 的应用程序来实现的。这是一个巨大的构建在谷歌 AppEngine 基础设施之上的门户服务应用程序，appspot.com 是一个命名空间，互联网社区用户可访问其下所有使用谷歌 AppEngine 技术开发的可扩展的 Web 应用程序。更加面向列出可用的云构件的解决方案是云市场[⊖]，其特点是全面列出亚马逊 EC2 映像。尽管这些解 388
决方案不提供完整的市场目录实施，但它们是实现 MOCC 的一个步骤，因为它们提供了方便查找组件的方法，这对构建云计算系统十分有用。

3. 案例研究：Cloudbus 工具包

一个有趣的案例是 Cloudbus 工具包 [131]，它包括一系列技术和组件，全面尝试理接实现面向市场的云计算构想的挑战。

图 11-6 提供了 Cloudbus 组件的综合视图，以及它们如何交互从而为 MOCC 提供平台。现实生活中的应用涉及各种场景，如金融、科学、教育、工程、多媒体和其他，需要云计算设施处理工作负载和存储数据。

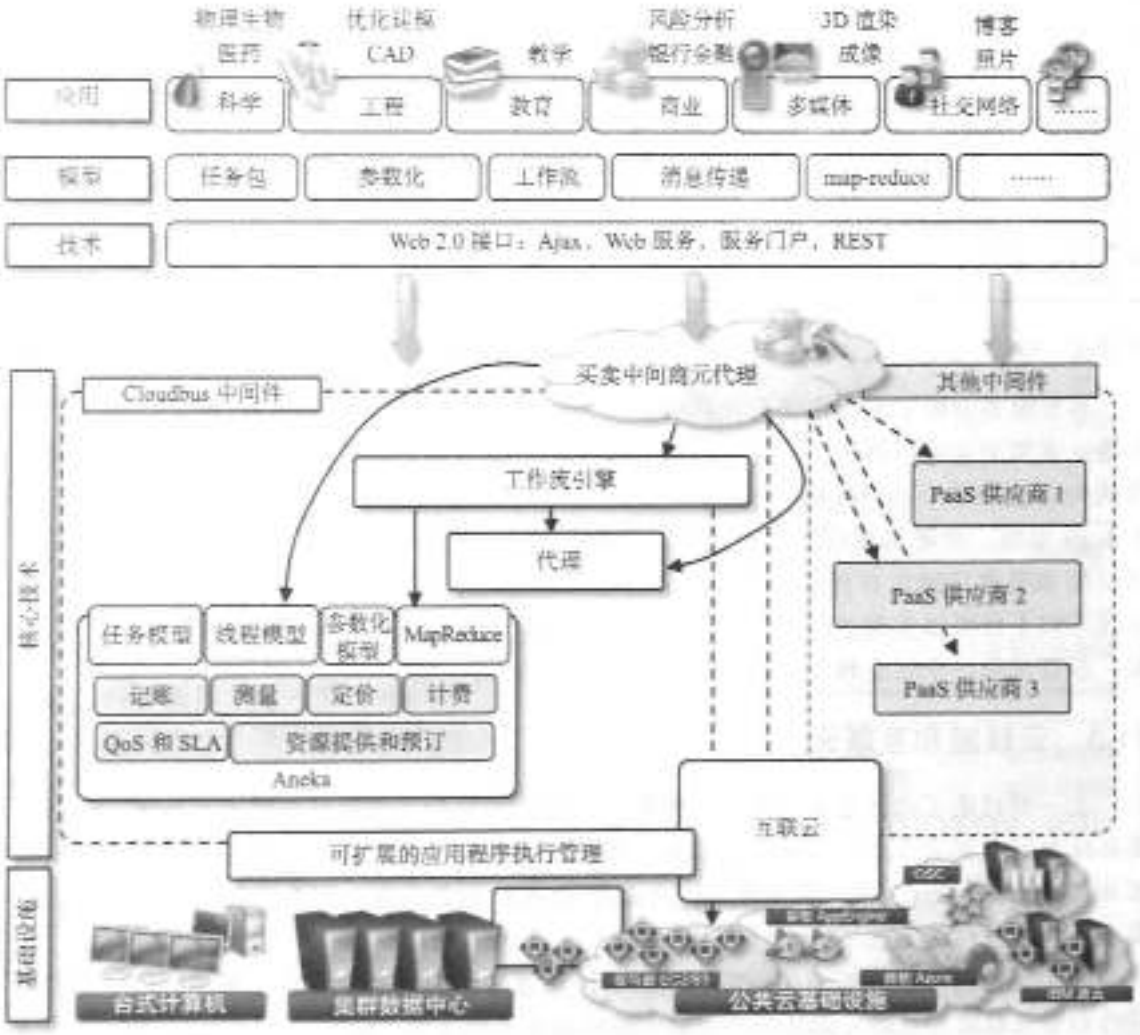


图 11-6　Cloudbus 工具

⊖ www.theCloudmarket.com。

Cloudbus 工具包可以作为一个通用的前端处理器，通过 Cloudbus 工具可以获取满足应用程序需要的云计算服务。它提供可用的工具和技术来实现服务代理基础设施和部署云应用程序的中间件。代理服务由买卖中间商实现，使用户可以充分利用云计算市场。它依赖于不同的中间件实现以完成用户的请求，这些中间件可以是 Cloudbus 技术或第三方实现。Aneka 或工作流引擎技术提供在云中执行应用程序的服务。这些可以是公共云、私有内部网，或者所有可以在互联云 [114] 范围统一管理的数据中心。联合属于不同组织的云计算形成一个唯一的域，描述各方之间的协议。

表 11-1 概述了构成 Cloudbus 工具包的所有组件，并简要说明了其功能。该工具包还包括开发算法和 CloudSim[132] 在模拟环境中部署的功能。这个工具包使用户能够模拟云计算环境的许多方面，从云基本构件（数据中心、计算节点、核、网络和虚拟机）到资源分配算法和策略。由于它的多功能性，该工具包已用于模拟电源感知调度策略 [133]，用于减少大型数据中心的能耗。

表 11-1　Cloudbus 工具包组件和技术

技术	描述
Aneka	云应用程序开发和部署的中间件
代理	跨异构系统实现分布式应用程序的中间件，基于任务包模型
工作流管理系统	执行、组合、管理和监控跨异构系统的工作流的中间件
买卖中间商 / 元代理	在市场范围内匹配用户的需求与服务供应商能力的媒介
互联云	建立云计算联盟的框架
MetaCDN	利用存储云智能地传送用户内容的 MetaCDN 中间件，基于其 QoS 和预算首选项
数据计算	正在研究的开发技术和处理可扩展性和能效的技术

11.2.4　结论

本节简要回顾了面向市场云计算的基本概念。与其他任何技术一样，概念存在一个巩固阶段，这期间会进行研究、试验、吸收和运用。云计算是分布式计算的演变，是以更有效的方式向企业和单个用户提供 IT 基础设施服务的新方法。最初，大量的研究兴趣和力量集中在 IaaS 实施，并将其集成到现有的计算系统和工作流过程中。目前，PaaS 解决方案日益普及，用来更稳定地发布商业和主流的软件应用和系统。在这个合适的环境中，支持 MOCC 的技术和系统可以实现并具备可用性。这个意义上所做的工作大部分仍然属于学术研究的范
389 畴，尽管其第一个商业和工业实现有较广泛的支持者。

11.3　云联盟和互联云

上一节讨论了如何定义计算资源作为公共基础设施服务来交易的模型和系统，以及云计算系统如何互相交互产生这种交易。本节从行政和组织角度来解决同样的问题，并介绍云联盟和互联云（InterCloud）的概念。这些概念 MOCC 都能实现，因为它们提供了不同云供应
390 商之间相互操作的方法。

因为服务可以按需按使用付费，所以云计算意味着服务商之间存在金融协议，这些概念特性对于云联盟和互联云都适用，但对于构建属于不同管理领域的云联盟具有一定的局限性。

11.3.1　特性和定义

术语云联盟和互联云往往可以交替使用，表达的都是单独管理域的云计算供应商聚合的

一般含义。澄清这两个术语的含义，以及它们如何应用于云计算是非常重要的。

术语联盟意味着创建一个整体组织，取代单个实体的决策和行政权力[⊖]。在云计算背景下，这个词并没有这么强的内涵，但意味着不同的云供应商之间达成了协议，允许他们有特权利用彼此的服务。术语云联盟的定义由 Enomaly 公司的创始人兼 CTO Reuven Cohen 给出[⊜]：

> 当两个或更多独立的位于不同地理位置上的云共享身份验证、文件、计算资源、指挥和控制，或访问存储资源时，由云联盟管理一致性和访问控制。

这个定义足够广泛，包括了云服务联盟的所有不同的表述，云服务联盟由云服务供应商之间签订的协议管理，而不是由用户组成。

互联云常常是云联盟概念的替代术语。互联云由思科引入，用来表达相互连接的云服务组合，云组合通过开放标准提供利用云计算服务的统一环境。模仿互联网“网络中的网络”术语，互联云代表“云中云”[⊝]。表达将属于不同管理域的云连接在一起的概念。这在许多情况下是可以接受的，一些从业者和专家——如 Ellen Rubin，CloudSwitch[⊞]产品的创始人和副总裁——更喜欢赋予这两个术语不同的内涵：

> 互联云和云联盟之间的主要区别是，互联云基于标准和开放的接口，而云联盟使用供应商的控制平台。在互联云环境中，所有云都知道应用程序如何部署。最终提交给云的工作任务将包括关于资源、安全、服务等级、地理位置等信息的描述，足以使云计算能够处理请求和部署应用程序。这将创建真正的效用模型，其中所有的需求都可以定义和描述，应用程序可在任何云上执行，只要有资源支持它。

391

因此，术语互联云大多指全球概念，其中由标准控制不同的云供应商之间的互操作性，从而创建一个开放的平台，应用程序可以迁移工作负载，并从不同的资源来源自由组合服务。而云联盟的概念比较普遍，包括基于私有协议和专有接口的云计算供应商之间的特殊集成。

11.3.2　云联盟栈

创建云联盟涉及不同层面的研究和开发：概念层、逻辑层、操作层以及基础设施层。图 11-7 提供了设计和实现云联盟体系结构所面临的挑战，该体系结构协调属于不同管理域的云服务，使它们在一个统一的服务中间件环境中操作。

每个云联盟层提出了不同的挑战，需在不同的 IT 堆栈层操作，并使用不同的方法和技术。总的来看，面对每

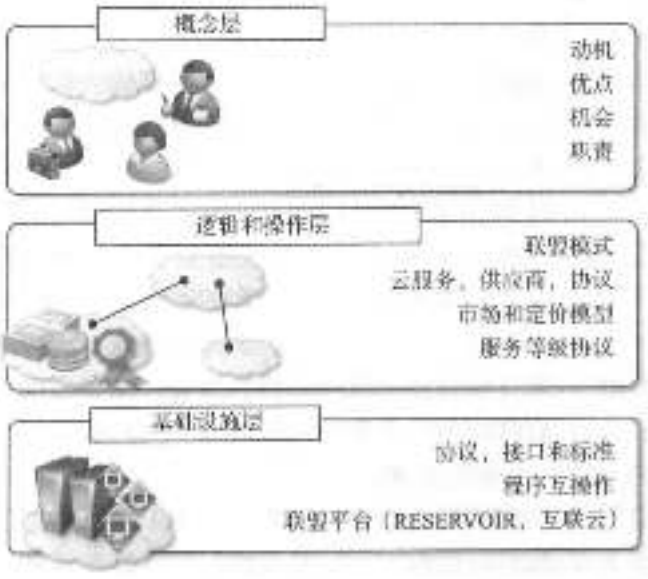


图 11-7　云联盟参考栈

⊖ 这个定义由韦氏在线词典提供：www.merriam-webster.com/dictionary/federation（2011 年 5 月 10 日检索）。

⊜ www.elasticvapor.com/2008/08/standardized-Cloud.html（2011 年 5 月 10 日检索）。

⊝ http://www.samj.net/2009/06/interCloud-is-global-Cloud-of-Clouds.html。

⊞ CloudSwitch 是一个云公司，致力于发布企业门户网站到几个云计算系统：www.Cloudswitch.com/page/Cloud-federation-and-the-interCloud（2011 年 5 月 10 日检索）。

一层挑战的解决方案构成了云联盟的参考模型。

1. 概念层

在单个云供应商租用服务使用方面，概念层提出了云联盟作为良好解决方案需面临的挑战。对于本层而言，无论是服务供应商还是服务消费者，当他们加入联盟时，使其明确云联盟的优势是很重要的，还要描述清楚对于单个供应商的解决方案，联盟环境可以创造的新机会。本层相关的元素是：

392
- 云供应商加入联盟的动机。
- 服务消费者充分利用联盟的动机。
- 供应商将其服务租用给其他供应商的优势。
- 一旦加入联盟，供应商的职责。
- 供应商之间的信托协议。
- 相对消费者的透明度。

其中，最关注的是服务供应商和消费者加入联盟的动机。

从云服务供应商的角度来看，如果有助于增加他们的收入，或提供了新的机会增加他们的业务，那么加入云联盟就是有利的。此外，如果有助于在高峰负荷期间维持客户满意的QoS，即把极端需求放在单个供应商的基础设施上，那么加入云联盟也是有利的。更确切地说，要识别云服务供应商在这些动机的后面具有的功能和非功能性要求。功能要求包括：

- 为客户提供低延迟访问，无论他们身在何处。这对单个云供应商不太可能，他们的数据中心没有广泛分布。因此，需要低延迟的服务可能会因为地理位置问题而表现
393 不佳。在这种情况下联盟将有助于单一供应商提供同样的服务，满足预期的QoS。
- 处理突发需求。尽管云计算提出了无限容量和持续可用性的设想，但服务供应商依靠的仍是有限的IT基础设施，并使其最终得到充分利用。解决这个问题正常的办法是添加更多的容量来增加基础设施。例如，为了跟上存储和计算需求的增加，谷歌已经在5年内将服务器的数量从8000增加到超过45万，从4个服务器场迁移到60多个数据中心，Facebook最近也增加了一倍的数据中心容量。如此庞大的供应对大型IT企业是负担得起的，他们能适当的预测需求的增加。不常见的需求可以租用其他供应商的容量以得到更好的解决，因为并不是每一个云供应商都处于IT巨头的地位。云联盟通过提供一个鼓励租用资源或服务的环境，有利于此类活动的进行。
- 扩展现有的应用程序和服务，超越自身的基础设施能力。对额外容量的需求也可能来自现有应用程序规模的增长，现有应用程序是指那些临时托管的，不构成服务供应商的核心业务的组成部分。从联盟供应商租用额外服务的机会再次成为云联盟的一个潜在优势。
- 从不使用资源的容量获益。为了提供连续的可用性和无限容量，云服务供应商一般拥有自己的大型计算系统，尽管这些系统非常有用，但却会产生维护和功耗方面的成本。能效计算解决方案有助于降低成本和减少IT对环境的影响。云联盟则提供了不同的机会，供应商可以在一段有限的时间内将他们的服务租赁给其他供应商，即使没有直接的客户，也可从中收益。

加入云联盟的动机还包括非功能需求，最相关的部分如下：

- 满足数据位置的强制性规则。地理位置可能会限制供应商为消费者提供服务。在这种特殊情况下，借助云联盟的供应商不会缺乏容量，但是由于供应商的数据中心位

置问题，不能确定某一位置供应商是否能为用户提供服务。当云服务处理特定级别的机密数据时，数据的地理位置成为一个重要的问题。不同的国家有不同的规定，例如，可能存在政府机构可否访问机密数据层的问题。

- 存在运营成本瞬态尖峰。运营成本也可能遇到临时尖峰，比如由于自然灾害，电力会有突然变化。这种情况不方便充分利用指定的数据中心，因此有机会利用联盟资源提供价格更便宜的服务。
- 灾难恢复。一旦发生自然灾害，如果数据中心在同一地点，那么灾难将导致整个数据中心或更多数据中心不定时中断服务。在这种情况下，根据供应商之间的协议来处理灾难状况更有可能在联盟范围内解决，而不是在竞争激烈的市场中解决。 394

对于以上所有情况，云联盟提供的不仅是概念上的解决方案，同时也是实现这些目标的切实可行的手段。

云联盟是云服务供应商利益最大化的叠加，并假设对服务消费者透明。除了终端用户间接利益，确实有一些潜在的直接受益于云联盟的概念。间接利益大多与终端用户感知的服务相关。真正的 QoS 通过执行准入控制，确保如果请求被接受，将按照与客户签订的 SLA 中定义的 QoS 属性提供服务。目前，各大云服务供应商保证 QoS 大多是基于可用性，而不是其他质量因素。例如，在 IaaS 场景，虚拟机实例的发布硬件功能可能无法反映其真实的性能。由于没有 SLA 保证此性能，供应商将始终尝试服务于请求，甚至提供低性能的服务。在云联盟的环境下，请求可以利用其他供应商服务，从而确保预期的性能指标得到满足。因此，对于用户的间接效益，当要求服务时，云联盟可以提高整体服务质量。直接利益也给终端用户带来好处，因为用户能够感知云联盟的存在。

提供不同服务的云供应商可以互相支持，因为他们不是竞争对手。航空公司和酒店细分市场之间的合作是一个很好的例子。航空公司为客户提供与预订机票配对的住宿选项。这一般可能是相互合作的酒店和航空公司之间协议的结果，从而给顾客提供更好的服务。这两个公司不会相互竞争，如果他们为客户提供一个完整的解决方案，他们可以双赢。云联盟计算环境可以复制这种类型的合作。例如，驻留在不同的细分市场（IaaS、PaaS、SaaS）的供应商可以互相打广告，给用户提供更好的服务。有遗留系统的企业将主要考虑用 IaaS 解决方案来部署和扩展他们的系统。IaaS 供应商可以在其产品服务之外支持对 PaaS 服务的访问，方法是选择那些可能会互补的、用户感兴趣的，云联盟供应商提供的服务。

未来，AWS 可能会支持以一定价格优势访问 AppEngine，或简单地提供一个更好的与谷歌的交互，提供数据传输、网络连接和带宽方面的服务。这将给客户带来哪些帮助？在亚马逊 EC2 上已经托管 Web 应用程序的公司以后可能需要整合新的功能，并采用可扩展的开发技术。由于具有从 EC2 部署利用 AppEngine 的性能优势，这可能是可选的解决方案。在云联盟范围内这种可能性更高。而且云联盟可以为用户提供更好的服务，即使他们托管在同一个细分市场，但提供不同的服务。例如，在 IaaS 方案中，具体供应商可能无法服务于 VM 模板托管一个特定的操作系统，但它可以建议或为客户指定另一个能够提供这种功能供应商。如果这两个供应商都服从属于同一个联盟的协议，那么这种情况便是可行的。

作为联盟的一部分，意味着供应商有责任避免寄生行为。例如，每个供应商都希望成为 395
为联盟贡献资源的活跃成员。这使得联盟更可靠，增加了每个供应商对它的信任。供应商的职责，如经常向联盟贡献一小部分可用资源和服务，可能会被认为是损失，但也可能是潜在的好处。例如，大公司（如谷歌）按照峰值要求而不是详细的实际使用收取每个月的电力使

用费[135]。这意味着如果在一个月内数据中心达到90%的峰值容量，平均工作60%，那么该公司将按90%的容量支付整整一个月运营成本的电费。这导致公司投入了大量的精力来优化数据中心的利用率。云联盟可能是这种优化的替代方案，因为它可通过联盟的其他成员来使用内部资源。从出租这些资源获得的收入可以弥补电力成本的峰值请求。

这些方面都是云联盟存在的理由。概念层的障碍是安全和信任的影响。例如，在联盟范围内供应商可能将一部分服务消费者的请求转移到与它有协议的另一个供应商。这对用户是透明的，用户可能不希望这样做。为了使云联盟的概念可行，从而有效地利用云计算技术，这些挑战必须得到妥善解决。

2. 逻辑和操作层

云联盟的逻辑和操作层需处理如下挑战：设计一个框架，使属于不同管理域的供应商聚集在单一基础设施环境下，即云联盟。互操作策略和规则在本层定义。此外，这一层还决定如何以及何时租赁一个服务给另一个供应商或如何充分利用来自另一个供应商的服务。逻辑组件定义了供应商之间的协议签订和服务谈判，操作组件特征化并且形成联盟的动态行为，联盟是单一供应商的选择的最终结果。该层实施和实现MOCC。

在本层解决以下重要挑战：

- 联盟应该如何表示？
- 如何建模和表示云服务、云服务供应商或一个协议？
- 如何定义允许供应商加入联盟的规则和政策？
- 解决供应商之间协议的合适机制是什么？
- 供应商彼此的责任是什么？
- 供应商和消费者什么时候应该利用联盟的优势？
- 哪些类型的服务更可能被租赁或购买？
- 出租的资源应该如何定价？应该租赁哪一部分资源？

逻辑和业务层为学术界和工业界提供了机会。我们需要一个联盟，或者更一般地说，某种形式的互操作现在已经经过评估，但没有共同和明确的指导来定义一个云联盟模型及如何应对这些挑战。事实上，若干方案正在研究中。在学术研究方面，正在研究云互操作的组织
396 模型和经济模型，用于描述一个联盟环境下云服务供应商和服务消费者的行为。行业内，各大IT巨头都在努力促成标准，并为可互操作的云计算起草建议。

在这一层上有必要特别关注SLA和它们的定义[150]。SLA的必要性在学术界和工业界是公认的事实，因为SLA更清楚地定义了不同供应商之间的买卖和租用。此外，SLA允许我们评估交易服务是否根据预期的质量配置交付。可以规定监管供应商的交易政策，对降低服务交付建立惩罚。这一点尤其重要，因为它会增加各方对云计算联盟的信任等级。

自1980年以来SLA一直在使用，它起源于电信领域[152]，用来定义附加到消费和网络供应商之间的合同的服务质量。从那以后，SLA已被用于多个领域，包括Web服务、网格计算和云计算。在不同的领域SLA的具体特性不同，但它可以普遍定义为"存在于两个组织（服务供应商和服务消费者）之间的业务关系的期望和职责的明确声明。[⊖]"SLA定义了供应商的性能交付能力、消费者的性能配置和监视手段和交付性能度量。SLA的实施应该指

⊖ Dinesh V,Supporting Service Level Agreements on IP Networks, Proceedings of IEEE/IFIP Network Operations and Management Symposium, 92(9):1382-1388, NY,USA, 2004.

定[153]：

- 目的。实现使用 SLA 的目标。
- 限制。必要的步骤或需要采取的措施，确保服务请求按级别交付。
- 有效期。SLA 有效的时间段。
- 范围。将要交付给消费者的服务及 SLA 之外的服务。
- 参与方。任何有关组织或者个人以及他们的角色（如供应商、消费者）。
- 服务等级目标（SLO）。双方都同意的服务等级。这些都是通过服务水平等级指标表示的，如可用性、性能和可靠性。
- 处罚。如果交付的服务没有达到定义的 SLO 会受到处罚。
- 可选服务。不是强制性的服务，但可能需要。
- 管理。保证 SLO 的实现，确认相关控制过程的组织责任。

参与方执行 SLA 的过程包括 SLA 的发现（创建）、执行和终止。一旦其 SLA 有效期结束或者违反了合同就终止[154]。Sun 互联网数据中心工作组给出了更加细化的过程，SLA 的生命周期包括六个步骤：

1）发现服务供应商。

2）定义 SLA。

3）建立协议。

4）监控 SLA 执行。

5）终止 SLA。

6）SLA 违约处罚。

397

目前有一个非常基本的 SLA 管理框架，不同的云供应商之间的互操作性使得这种特殊集成非常复杂。已提出的处理过程只是一个参考模型，仍然有待具体实现。此处定义的每个步骤[150]，相当多的工作在学术领域已完成，但在工业界，SLA 仍是由服务供应商强加的单方面约定，用户只可以接受。

3. 基础设施层

基础设施层解决参与实现异构的云计算系统无缝互操作的技术挑战。它处理保持各个云计算系统属于不同管理域的技术障碍。由于具有标准化的协议和接口，这些障碍是可以克服的。换句话说，联盟的基础设施层类似互联网的 TCP/IP 协议栈：一个模型和一个系统互操作技术的参考实现。

基础设施层奠定了其在云计算参考模型（4.2 节中讨论）的 IaaS 和 PaaS 的基础。互操作和接口服务也可能在 SaaS 层实施，特别是对协商和云联盟的实现。本层需重点解决以下问题：

- 应采用什么样的标准？
- 接口和协议如何设计成互操作？
- 用于互操作的技术有哪些？
- 如何实现一个能互操作的软件系统？如何设计平台组件和服务？

只有开放标准和接口，不同的云计算厂商间的互操作和组合才是可能的。此外，接口和协议在云计算参考模型的每一层变化很大。作为比较成熟的技术层，IaaS 层在此方面已经很先进。几乎每一个 IaaS 供应商都公开 Web 接口，用于封装虚拟机模板，以及启动、监控和终止虚拟实例。即使没有标准化，这些接口都利用 Web 服务技术，彼此也十分相似。使用

共同的技术简化了供应商的互操作，实现互操作需要的代码量最小。这些 API 允许定义一个抽象层，统一地访问几个 IaaS 供应商的服务。目前已经有工具（开源和商业都有）和在这层实施互操作性的规范。

全面支持云联盟的构想需要实现更复杂的功能。例如，在不同的供应商之间动态迁移虚拟机实例是必要的，以支持在不同的 IaaS 厂商中动态负载平衡。在这方面，开放虚拟化格式（OVF）[51] 的目标就是解决这个问题，OVF 最终可能会成功，因为一些云计算供应商已经了采用该规范。如果考虑 PaaS 层，互操作变得更加困难，因为每一个云计算供应商提
398 供其自己的运行时环境，不同的云服务供应商开发应用程序的语言和抽象不同，目标也不同。目前，这层的互操作还没有实现，也没有简化供应商之间互操作的标准。至于 SaaS 层，提供的不同服务使互操作不太重要。

SaaS 层实现互操作性的案例是网上办公自动化，如谷歌 Documents，Zoho Office 等，能在不同的格式间导出和导入文件，从而简化数据的交换。另外，应用服务组合似乎更有吸引力，不同的 SaaS 服务可以组合在一起，为用户提供更复杂的应用。服务组合对于在不同层跨云计算平台互操作更为重要。即使在这种情况下，需要注意的是，目前，在某一层操作的云计算供应商往往在其基础设施更低的任何层实施，需要提供服务给终端用户，他们都不愿意开放自己的技术栈来支持互操作。

云服务供应商的联盟环境提出的构想在每层都带来了许多挑战，尤其是在逻辑和基础设施层，需要设计适当的系统组织和部署有效的技术。大量的研究工作一直在逻辑和操作层上进行，目前已设计了 IaaS 方案的初步实现和互操作技术的草案。

11.3.3　关注点

很多因素对云联盟的成功实现做出了贡献。除了动力和技术，还要考虑其他因素。定义云供应商之间的互操作平台时，必须考虑互操作性标准、安全性和法律问题。

1. 标准

标准是建立联盟的基础。其主要作用是构建实现互操作的平台，解决复杂聚合和供应商迁移问题。标准化的接口和协议易于实现开放的组织，供应商可以很方便地加入。建立标准的优点主要在于技术方面，可以促进互连的软件和服务系统的开发。此外，标准定义了新的供应商加入的方法，从而有助于开放系统的实现。

供应商之间的互操作一直是企业关注的因素，也是最初妨碍他们完全接受云计算模式的原因。更具体地说，在可移植方式中，没有开发应用程序和系统的共同标准，导致了厂商绑定问题。应用程序和系统的开发部署在一个特定的云计算供应商的基础设施上，无法轻易迁移到另一个供应商。IaaS 解决方案还利用了虚拟机实例和模板的专有格式，防止实例从一个供应商的平台移动到另一个。PaaS 解决方案中，技术性障碍更要考虑，即使开发技术可能会有所不同。这些技术壁垒导致人们开始研究试图克服这些障碍的解决方案，至少可用于有限的供应商。例如，Rightscale[⊖]，为客户提供了一个可以透明地在不同的 IaaS 供应商开发平
399 台部署的解决方案。在 PaaS 方案中，Aneka 提供了一个中间件，可以跨 IaaS 供应商利用异构硬件并支持跨平台部署。JClouds[⊖]项目定义了一组库和组件，允许统一使用不同的 IaaS

⊖ www.rightscale.com。

⊖ http://code.google.com/p/jclouds/。

供应商在云上用 Java 开发应用程序。

这些技术改进帮助系统设计人员和开发人员开发少受厂商绑定问题影响的系统。但这些努力对于建立一个联盟还有距离，建立联盟只能通过为互操作设计标准、协议和实现格式的方法来实现。在开放组织和主要行业合作伙伴联盟的环境下，人们正在努力创建标准。在学术界可以发现一些小的努力。

（1）开放云计算声明

开放云计算声明[⊖]是实现云互操作平台的第一步。该声明由不同的云供应商协调制定，于2009年起草，最近列出了超过400个云计算服务供应商，他们都支持此声明的构想。该声明不是提出标准，而只是表达意向，由云计算领域的商家签署，实现互操作的开放云计算平台。声明的目的是指导首席信息官、政府、IT用户和想使用云计算建立一套云计算核心标准的领导者。

该声明列举了云计算的优势，探讨了它面临的障碍和挑战。这个开放的云平台目标可以概括如下：

- 选择。随着开放式技术的应用，当企业经营环境变化时，将有可能为消费者选择最好的供应商、体系结构或使用模型。此外，使用开放技术简化了云计算解决方案的整合，不同的供应商提供解决方案或现有基础设施，从而促进各供应商采用声明提供的平台。
- 灵活性。如果不同供应商不使用封闭的专有技术，一个供应商和另一个供应商之间的转换将变得更容易，封闭的专有技术意味着相当大的转换成本。
- 快速和敏捷。如前所述，开放的技术促进云计算解决方案和现有的软件系统的整合，从而实现快速和灵活性的缩放需求。
- 技能。开放技术简化学习过程，有助于达成共识以设计、开发、部署跨多个供应商的系统。这方便组织机构找到他们需要的具有适当技能的人。

声明以一组推荐结束，鼓励云计算厂商追求开放的协作和适当的标准，鼓励使用现有的标准而不是专有的解决方案，合适的新标准的起草应作为基于社区的努力结果。 400

通过证明开放云计算平台的优势，声明在概念上奠定了云联盟方案的基础。事实上，使用开放的技术将创造一个更灵活的环境，IT消费者可以更方便地选择云计算技术，不会感觉到供应商绑定的威胁。云计算联盟的概念促进了这个初始远景的发展，这意味着一个更加结构化和明确的合作。

（2）分布式管理任务组

分布式管理任务组（DMTF）是一个包括4000多个成员、44个国家和近200个团体机构的组织。它是一个工业组织，领导互操作管理标准的制定、采纳和推广。具体到云计算，DMTF介绍了开放虚拟化格式（OVF），支撑了几个云技术互操作的项目，如开放云标准孵化器，云管理工作组（CMWG）和云审计数据联合工作组（CADFWG）。

开放虚拟化规范（OVF）[51]是一个与服务供应商无关的规范，封装设计标准以易于虚拟设备在不同的虚拟平台之间迁移和部署。独立软件供应商（ISV）使用OVF封装安全地分发应用程序，系统映像可以导入和部署在多个平台上，从而实现跨平台可移植性。Dell、HP、IBM、Microsoft、VMWare和XenSource协同努力制定了该规范，定义了封装软件的

⊖ www.opencloudmanifesto.org。

平台独立的虚拟化规范。2008 年，他们向 DMTF 提交了一个初步的草案，目前 DMTF 已经发布 1.1.0 版本的标准，并被批准为美国国家标准协会（ANSI）INCITS 469-2010 标准。该标准规范的主要目的是为分配封装软件系统提供一个平台无关的规范。因此，该规范的主要特点如下：

- 优化分配。规范支持验证和完整性检查，基于行业标准的公钥基础设施提供了一个基本的许可证管理方案。
- 优化简单的、自动的用户体验。支持验证整个包、每个虚拟机或元数据的组件。
- 支持单虚拟机和多虚拟机的配置。能封装单一映像应用程序或复杂的多层系统。
- 便携式厂商和平台独立的虚拟机封装。OVF 设计是平台无关的，但允许特定平台增强的嵌入。
- 可扩展。尽管目前的标准已经能够捕捉封装一个虚拟机需要的所有数据，但 OVF 仍然为未来的需求提供了扩展和定义新功能的方法。
- 本地化。该规范允许在多个地方嵌入用户可见的描述，因此，可大范围跨越系统和市场立即准备进行处理。
- 开放标准。该规范格式已经引起业界主要供应商的合作，它的发展将带动一个公认的行业论坛，以保障未来的传播和使用。

401

从技术的角度来看，OVF 定义了一个虚拟机模板的传输机制。一个单独的 OVF 包可以包含一个或多个虚拟机映像，一旦部署到主机系统，便可增加为实现特定目标的一个自包含的、自我一致的软件解决方案。采用 OVF 的例子有 Linux、Apache、MySQL、PHP（LAMP）栈或划分为一个或多个虚拟映像的任何其他组件的组合。OVF 的重点是交付一个封装好的、可移植的、随时使用并可验证的软件装置。为了实现这一目标，它允许封装多个虚拟实例，以及在虚拟硬件部署所需的描述。目前，OVF 大多采用多层体系结构实现应用软件或系统，系统组件可以分布在多个计算节点，每个节点需要不同的操作系统和硬件特性。通过支持单一包内的多个虚拟映像，ISV 可以封装和保证整个系统是一个整体，将其作为一个单一组件发布。

可移植性是 OVF 的基本性质，它完全解决了在很宽的虚拟硬件范围内部署软件应用所产生的潜在问题，确定不同层次的可移植性：

- 1 级。封装设备运行在一个特定的产品或 CPU 架构或虚拟硬件上。
- 2 级。封装设备运行在一个特定的虚拟硬件系列。
- 3 级。封装设备上运行多个虚拟硬件系列。

决定 OVF 包的可移植级别的是封装在设备中的信息及其特定需求。1 级的可移植性主要是虚拟硬件具体信息不支持 OVF 的部署，即便是在相似的虚拟机管理程序中的部署。这些信息可能是虚拟机的暂停状态或快照。2 级的可移植性在管理域是可以接受的，这决定了虚拟化技术的采用可能是最为集中的。3 级的可移植性是预期和独立软件开发商需要的，封装设备能够在任何虚拟平台部署。在云联盟的背景下，对于无缝的不同供应商之间的合作，这是预期的可移植性的级别。OVF 规范自公布以来，已经相当肯定地被接受了，一些开源项目和商业产品能导入软件作为 OVF 包发布。

开放云标准孵化器是另一个重要的项目，它是由 DMTF 推动的，目的是促进云技术间的互操作性。通过开发云管理用例、体系结构、定义合作，孵化器的重点是云计算环境之间的合作的标准化。在 2010 年 7 月结束的孵化器活动中，产生了一系列白皮书（[137]、[138]

和[139]），指导开发可互操作的云系统。这些白皮书是云管理工作组（CMWG）的出发点。其目的是提供一套约定俗成的规范，赋予架构语义和定义实施细节，实现服务请求者 / 开发者和供应商之间的云互操作管理。另一个相关的 DMTF 提议是云审计数据联合（CADF）工作组，将主要定义可互操作的 API，制作和共享具体审计事件、日志信息和每一个租户的报表，从而简化异构云计算环境的监测。

DMTF 对实现一个可互操作的云环境做出了重要贡献，其唯一具体的活动是发布 OVF 规范。不过，工作组和相关文件的活动，为实现开放的方式云联盟铺平了道路。

（3）开放云计算接口

开放云计算接口（OCCI）[⊖]是一个开放的组织，包括一组社区推动的和通过开放网格论坛发布的规范。这些规范定义了一些协议和几种管理任务的 API。最初的设想是为 IaaS 式服务创建远程管理 API，现在 OCCI 已演变成一个更广泛的针对集成、可移植性和互操作性的 API。目前的规范覆盖了云计算的三个产品市场：IaaS、PaaS 和 SaaS。OCCI 目前已发布三份文档，其中定义了：

- OCCI 核心模型的形式化定义。
- IaaS 领域的 OCCI 基础设施扩展的定义。
- OCCI 实施模型，它定义了如何通过 HTTP 的 REST 与 OCCI 核心模型交互。

除了这些，OCCI 还致力于其他标准，涵盖计费、监控、提前预订、协商和协议。

图 11-8 给出了一个开放云计算接口参考模型，定义了访问给定的云供应商所管理的资源的单一接入点。

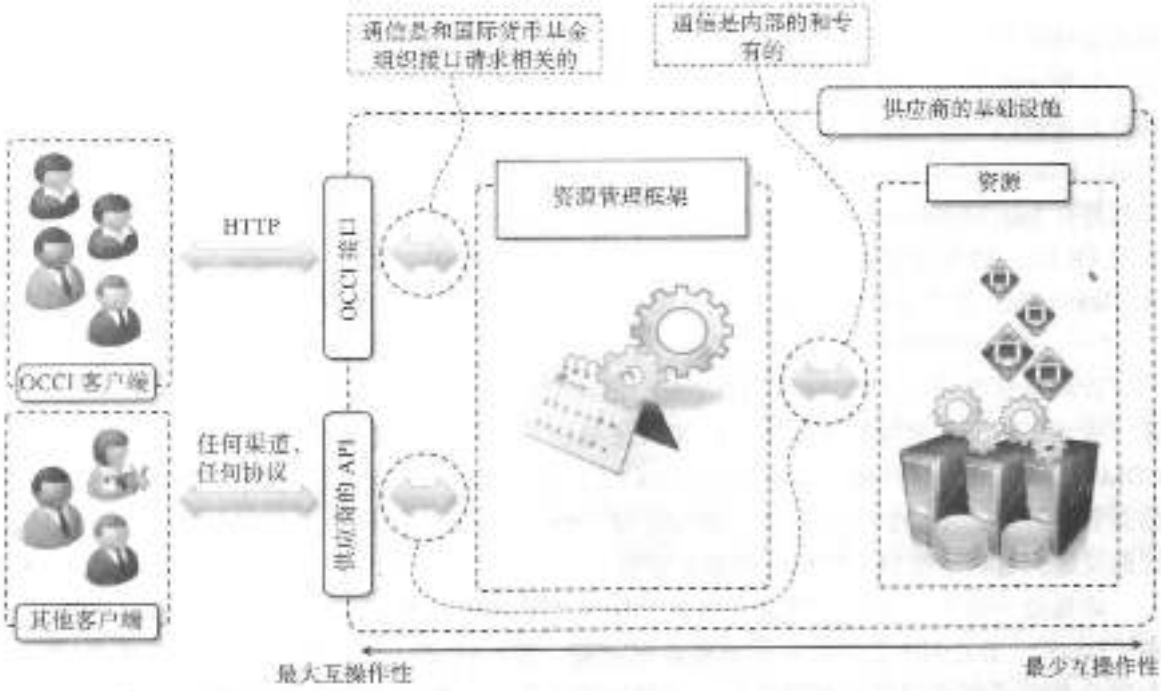


图 11-8　OCCI 参考模型

定义可互操作的平台最相关的规范是 OCCI 核心模型（GFD.183—OCCI 核心）。该文档

⊖ http://occi-wg.org/。

定义了边界协议和 API，API 是到供应商的内部管理框架的服务前端。此接口设计用来同时服务消费者和其他系统。本规范重点不是定义任何特定的领域，其主要范围是提供一种内置式分类系统，允许对特定领域使用安全扩展。因此，它提供了一个参考模型，构建一种类型尽可能灵活的系统，并支持可扩展性。这使得客户端可以动态发现支持这一标准的资源模型。资源是模型的基本元素，可以是任何类型的，并根据资源供应商操作的上下文不同而有所不同，比如，IaaS 供应商是虚拟机或模板，作业管理系统是作业或任务，等等。这些资源通过联系相连接，它允许客户端查看它们是如何连接在一起的。资源和联系构成实体，可以用来定义一个特定的类型。除了实体，类型由活动定义，表示可对该类型的实例进行的操作。为了支持灵活性和扩展性，混合用于聚合已经存在的类型和增强它们的功能。

这个参考模型由 OCCI 基础设施文档（GFD.184—OCCI 基础设施）指定，该文档定义了 403 OCCI 如何根据前一参考模型进行建模并实现一个 IaaS API。在这种情况下，公共资源是计 404 算，存储和网络。联系可以是类型 NetworkInterface 和 StorageLink。

这些组件可以用来表示任何 IaaS 供应商提供的任何服务，并描述它们的行为和属性。

OCCI 模型的实际执行可通过基于 HTTP 的 RESTful 接口进行访问。该 OCCI 实施模型标准（GFD.185—OCCI HTTP 提供[⊖]），定义了该模型可以被传输并通过 HTTP 进行序列化。因此，为了兼容 OCCI，云资源供应商必须实施以下两个步骤：

1）根据 OCCI 核心模型定义所提供的服务和资源。

2）根据 OCCI 的 HTTP 模型，提供一个 RESTful 接口，允许客户端发现供应商发布的资源集合。

如果云供应商是一个 IaaS 的供应商，那么 OCCI 基础设施文档是基础，供应商可以按照该文档开发应用。

目前，开放云计算接口支持与几种构建云计算服务的技术互连，主要用在 IaaS 方案。各种开源产品（jCloud、libvirt、OpenNebula 和 OpenStack）、研究项目（RESERVOIR、Big Grid、Morfeo Claudia、Emotive）和财团（SLA @ SOI）都在提供的服务中支持 OCCI 接口。最初是在 Sun Microsystems、Rabbit MQ 和 Univesidad Complutense de Madrid 共同努力下开发的 OCCI，如今在学术界和工业界涉及超过 250 个组织，如 Rackspace、Oracle、GoGrid 和 Flexiscale。有了这种强大的支持，OCCI 将定义云联盟的云互操作性标准。

（4）云数据管理接口

云数据管理接口（CDMI）[136] 是一个函数接口标准，应用程序将使用它来从云中创建、检索、更新和删除数据元素。此接口还提供了设备用于发现给定的云存储产品的性能。CDMI 已经由云存储网络工业协会（SNIA）的云存储技术工作组提出，SNIA 是一个推动 IT 信息管理标准的协会，尤其专注于数据存储。SNIA 还提出了 CDMI 参考实现，从而通过社区的反馈，有利于加快工作标准的制定过程。

该规范介绍和定义了云存储的概念，作为“按需交付虚拟存储”，也称为数据存储即服务（DaaS）。DaaS 的主要概念是抽象数据存储，其后面是一组接口，并使其按需提供。这个定义封装了相当广泛的存储架构。云存储参考模型如图 11-9 所示。云数据管理服务通过 RESTful 接口向用户提供 CDMI。这样的接口能够获取信息、数据和存储服务，可以利用它 405 来访问存储云。最终可以依靠几种不同的技术来实现数据和存储服务。

⊖ 据报道该文档即将公布，但在写此书时还不可用。

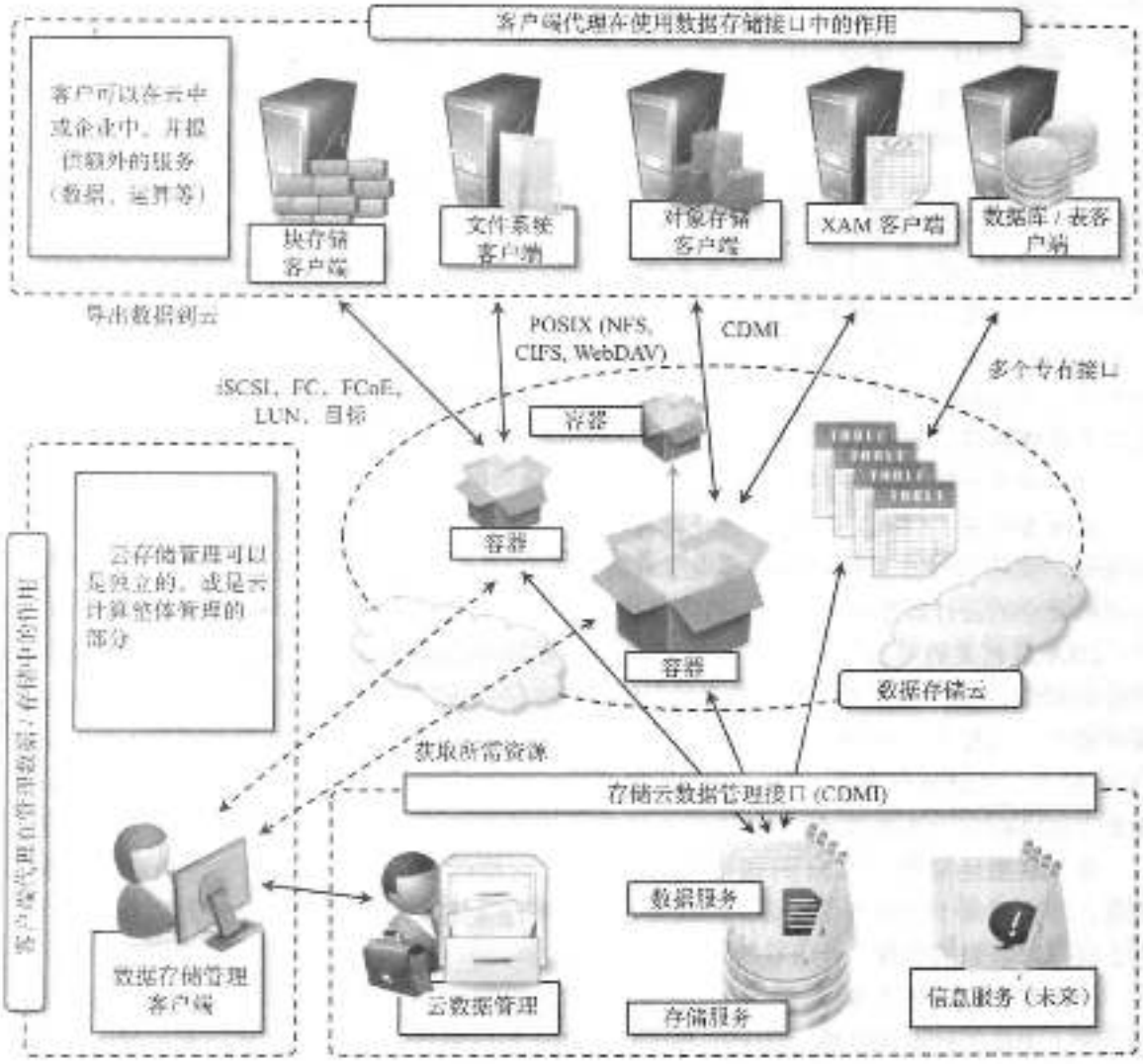


图 11-9　云存储参考模型

该接口包含对象模型，允许客户操作和发现数据组件。对象模型包含以下组件：

- 数据对象。这些都是 CDMI 中的基本存储组件，类似文件系统中的文件。数据对象有一套明确定义的单值的域。此外，它们可以支持描述对象的元数据，元数据由存储系统或客户提供。
- 容器对象。容器对象是用于存储数据的基本抽象。一个容器可以有零个或多个子对象和一组明确定义的域。至于数据对象，容器也支持元数据。容器支持嵌套和一个 406
继承其父容器的所有数据系统元数据的子容器。
- 域对象。域对象和容器对象十分相似，它们用来代表一个 CDMI 存储系统中管理的所有权。由于数据的聚合视图对管理有用，所以容器支持嵌套和促进信息流向上流。
- 队列对象。队列是一类特殊的容器，在存储和检索数据时提供先进先出（FIFO）访问。队列在支持读者 – 写者和生产者 – 消费者模式的存储数据管理方面是有用的。队列总是有一个父对象，从父对象继承系统元数据。
- 功能对象。功能对象是一类特殊的容器对象，允许 CDMI 客户端发现 CDMI 供应商

实现的 CDMI 标准的子集的内容。功能是交互集的描述符，系统能够执行它们所连接的 URI。对象模型中定义的每个实体要求有一个表示 URI 的域，该对象的功能可以通过 URI 检索。每一个 CDMI 标准的接口必须能列出每个特定对象的功能，但支持标准中列出的所有功能则是可选的。

使用简单的 REST 定义的操作，客户端可以发现和处理这些对象，如创建、检索、更新和删除对象（CRUD：Create，Retrieve，Update，Delete）。所支持的操作集由连接到各实体的功能来定义。除了 REST 允许的基本操作，相比其他协议和标准，CDMI 还支持快照、序列化和反序列化、日志和互操作。

CDMI 最初于 2010 年提出，是从其他几个机构收集的云计算技术的标准化的共识，已被列入发展和研究蓝图。目前，通过与各大标准化组织的互操作，SNIA 正在努力将 CDMI 转型为法律标准，如 ISO/IEC 和 INCITS[⊖]。

（5）云安全联盟

云安全联盟（CSA）是非营利性机构，使命是推广最佳使用实践，提供云计算安全和教育保证，使用云计算来帮助保护其他形式的计算。CSA 提供了讨论安全实践的环境和开发可靠和安全的云计算系统的指导，而不是作为一个标准化团体。

CSA 最相关的实现是云控制矩阵（CCM）。矩阵专门用来提供基本的安全原理，指导云供应商和协助可能的云服务消费者评估利用云服务供应商隐含的整体风险。该文档准备考虑最重要的安全标准、法规和控制框架，如 ISO 27001/27002、ISACA COBIT、PCI 和 NIST。CCM 加强了云计算中现有的信息安全控制环境，列举了在云中应该采用的安全方法，这些
407 安全方法已存在于其他领域。

在云联盟环境中，CCM 的相关性非常明显。它提供了一个标准化的方法来评估安全性措施，落实到每个云服务供应商，并帮助定义云联盟方案中最低的安全配置文件，从而在概念上提高对联盟的信任。

（6）其他标准化工作

除了主要的标准化工作，还有一些工作，通过特定的组织机构的认可，或在非常具体的环境中使用云计算。其中最相关的是：

- 美国国家标准和技术研究院（NIST）。正如在第 4 章讨论的，NIST 提出了目前广泛接受的云计算定义。另一重要的举措是加速采用云计算的标准战略（SAJACC）。此工作大多与现有云计算中标准的评估有关，积极推动建立开放标准，并找出现有标准的差距。
- 云计算互操作论坛（CCIF）。这个工业论坛最初由几名云计算领域参与者支持，并形成全球云计算生态系统，使得企业可以无缝地协同运作，以促进该技术被更广泛地采用。论坛的活动仅限于 2009 年一个单独的事件，产生了统一云接口（UCI）[⊖]提议，这是一种尝试，为各种不同厂商公开的 API 提供统一的接口。亚马逊 EC2 和 Enomaly ECP 已经实现了该功能。
- 开放云联盟（OCC）。这个非营利组织管理云计算基础设施以支持科学研究。该公司还为云计算技术研究实现参考模型、基本准则以及标准化。

⊖ 国际信息技术标准委员会（INCITS）是美国在信息和通信技术（ICT）领域的主要标准化组织。

⊖ 参考 http://code.google.com/p/unifiedCloud/（2011 年 5 月 17 日检索）。

- 全球跨云技术论坛（GICTF）。GICTF 是日本的组织，其目的是汇集行业、学术界及政府的研究成果，支持研究和开展云互操作性技术可行性测试。更确切地说，该论坛着重于网络协议的标准化，使云之间能通过接口发生互操作。
- 云工作组。开放组的倡议旨在在买家和供应商之间建立共识，主题是所有大小和规模的企业如何利用云计算技术。该工作组已进行了若干活动，以提高对云计算技术的业务理解、分析和领会。

还有一些其他举措，它们涵盖了云计算领域中标准和规范的创建，这与支持不同的云服务供应商的互操作不是完全相关，但仍然是支持云联盟方案技术发展的步骤之一，其方案基于云联盟的规则、策略和模型。

2. 安全

云计算通过利用大量计算基础设施支持弹性可扩展的系统的开发，最终托管应用程序、服务和数据。在这种情况下，安全问题是不容忽视的基本要求。在云联盟的情况下，安全管理更加复杂，其中的机密信息在多个云计算厂商间动态移动。

首要问题之一是，云供应商是否能提供足够的透明度来实现所需的安全管理流程，以保护用户的数据和应用程序。实施和部署所需的安全基础设施对理解谁负责提供云计算服务和谁使用云计算服务是非常重要的。云计算是安全问题的一个新领地，相对于管理服务供应商，其中管理安全基础设施由服务供应商完全处理，在云安全部署管理中则在供应商和客户之间划分。双方的责任根据服务产品的类型变化。在访问虚拟机实例或云存储的审计和日志方面，IaaS 供应商要求提供基本安全。PaaS 供应商期望为应用程序提供一个安全的开发平台和运行时环境。SaaS 的供应商在安全性方面负有主要责任，因为他们必须要建立一个安全的计算栈（基础设施、平台和应用程序），用户根据他们的需求定制计算栈。

尽管云计算引入了新内容，但也可能从考虑现有的安全框架和实践为出发点，来安全地运行应用程序和系统。特别地，ISO/IEC 27001/27002 标准和信息技术基础设施库（ITIL）服务管理框架是行业标准，能在安全管理过程和实践方面提供指导。ITIL 是一组定义了安全框架的指南，包括策略、流程、过程和工作说明。ITIL 不是一个标准，因此，组织和管理系统不能被认证为“ITIL 兼容的。”尽管如此，这些指南还是构成了实施安全措施的参考模型。ISO/IEC27001/27002 正式定义了对于信息安全管理系统的强制性要求，提供了在安全系统中实现的安全控制指南。总之，这些标准有助于组织机构确保当前安全级别是否适合他们的需求，并实现安全保障。

云安全管理的关键要素如下[156]：

- 可用性管理（ITIL）。
- 访问控制（ISO/IEC 27002，ITIL）。
- 脆弱性管理（ISO/IEC 27002）。
- 补丁管理（ITIL）。
- 配置管理（ITIL）。
- 事件响应（ISO/IEC 27002）。
- 系统使用和访问监控（ISO/IEC 27002）。

这些元素构成了云计算系统的安全参考框架，覆盖云计算参考模型的所有层，并且每种类型的云服务都有不同的支持方案。

安全方面的总体视图以及云计算供应商如何在三个不同层次（IaaS、PaaS 和 SaaS）解决

安全问题的描述见表 11-2。在确定客户的安全责任及规划适当的措施方面，较好的做法是与供应商签订服务合同。该合同明确规定了供应商的活动范围和对客户的保证。此外，云服务供应商的 API 和服务有助于管理安全性和责任。云供应商提供了服务控制面板，可以有效
410 整合到组织内部的管理流程和工具。目前，从安全的角度来看，云计算系统的安全管理缺乏企业级的接入，而企业系统管理员能为云计算应用程序定义安全的整体视图。

表 11-2　云中客户安全管理责任

活动	IaaS	PaaS	SaaS
可用性管理	管理虚拟机可用性与容错架构	为部署在 PaaS 平台的应用程序管理活动（供应商负责运行时引擎和服务）	供应商的责任
补丁和配置管理	管理虚拟机映像，使用客户建立的安全性强化流程强化虚拟机、应用程序和数据库 使用客户建立的安全管理流程管理虚拟机、应用程序和数据库的活动	为部署在 PaaS 平台的应用程序管理活动 为 OWASP 十大漏洞[⊖]测试应用程序	供应商的责任
脆弱性管理，访问控制管理	利用客户建立的脆弱性管理过程管理操作系统、应用程序和数据库的漏洞 管理网络和用户对虚拟机的访问控制，安全特权访问管理控制台，安装主机入侵检测系统 (IDS) 和管理主机防火墙策略	为部署在 PaaS 平台的应用程序管理活动（供应商负责运行时引擎和服务） 管理开发人员访问配置 限制使用访问身份验证方法（用户和基于网络控制）访问 如果支持 SAML[⊜]，联合认证和启用 SSO	供应商的责任 管理用户配置 限制使用访问身份验证方法（用户和基于网络的控制） 如果支持 SAML，联合认证和启用 SSO

云联盟安全

云联盟产生了新的必须要解决的问题：提供一个安全的环境，以便可以在一组联盟供应商之间移动应用程序和服务。这需要联盟中所有云供应商来保证基本安全。

一个有趣的方面是跨不同组织、安全领域和应用程序平台的数字身份管理。特别地，联盟身份管理是指基于标准的方法在联盟环境中处理身份验证、单点登录（SSO），以及基于角色的访问控制和会话管理 [157]。这使用户能够在联盟环境中更有效地利用服务，通过认证，只需一次便可登录到几个交易实体组成的网络。实现这种功能依靠的是开放的工业标准或公开发布的规范（自由联盟统一联合框架、OASIS 安全性断言标记语言和 WS-Federation），使得互操作可以实现。无论具体的协议和框架是什么，可以考虑以下两种主要方法：

- 集中式联盟模型。这是多种身份联盟标准采取的做法。它在 SSO 交易中区分两种操作角色：身份供应商和服务供应商。
- 基于声明的模型。这种方法从不同的角度解决用户认证的问题，要求用户提供声明回答他们是谁，以及为了访问内容或完成交易他们可以做什么。

⊖ 开放 Web 应用安全项目（OWASP）是一个开放的社区，致力于帮助企业设计、开发、维护、获取和操作可以信任的应用程序。2010 年组织出版了 OWASP 十大工程，其中列出了排名前十位漏洞的应用程序。www.owasp.org/index.php/Category:OWASP_Top_Ten_Project。

⊜ SSO 是单点登录（single sign-on）的缩写，表示被服务的集合提供功能，允许用户只进行一次身份验证，而不是显式地提供每一项服务的凭据。SAML 是安全声明标记语言（security assertion markup language）的缩写，用于在联盟环境中提供可移植框架以定义安全的 XML 方言。

现在正在使用第一种模型，第二种模型是云中身份管理的未来构想。

数字身份管理是云联盟安全管理的一个基本方面。为了跨越不同的管理域透明地进行操作，建立身份验证和授权机制框架是重要的，联盟身份管理解决了这个问题。之前的安全性考虑有助于设计和实施由各层云供应商和用户应用程序组成的安全系统；联盟身份管理能够将不同厂商的计算栈结合在一起，从安全的角度为用户呈现一个单独的环境。

3. 法律问题

除了云计算所需要的技术复杂性，访问权限、隐私和控制是云计算所特有的法律问题。本节概述了这个话题，并讨论了云计算联盟案例潜在的法律问题。

企业和终端用户利用云计算将系统、应用程序或个人资料交付给第三方控制，这不是正常的外包做法，这一点已得到了广泛的认同。此外，可以观察到应用服务供应商（ASP） 411
有很多的相似性，如果比较云计算和传统的外包和 ASP 服务交付，可以得出以下结论[140]：

- 外包一般涉及一个企业组织的整体业务或 IT 流程，而且为了客户的利益他们都是面向业务的。该软件通常属于顾客，并且在客户的设备上部署和管理。此外，这种安排高度可协商，并通过长期和复杂的合同予以规定。
- ASP 模式构成一个早期的 SaaS 交付模式的实现，这并不意味着能进行任何类型的按需扩展。在此方案中供应商拥有的软件和硬件，可能位于多个事先静态已知的位置。约定是可以商量的，但并不像管理外包那么复杂。
- 云计算涵盖了多种服务模式：软件、基础设施和运行平台。消费者和供应商之间的安排都相当有限，主要是基于按使用付费的模式，该模式根据适当的单位（时间、带宽、存储容量或交易）计算。性能的经济特性驱动供应商的活动，他们可能会选择定位或重新定位应用程序和系统，最大限度地降低成本和提高效率。此外，供应商可能有多个分散数据中心，更好地服务于各种各样的客户或利用有利的环境条件。

云计算和传统方法的不同在于服务供应商经营和提供服务的全球规模特性。大量潜在的客户需要服务，促使供应商寻找那些保证交付最佳服务和低运营成本的解决方案。因此，通常跨越多个国家的稀疏的基础设施是相当普遍的。其次，因为云计算提供 IT 服务作为公共基础设施服务，动态服务租赁和消费统一安排，因此不适用于商业用途。其结果是，多个服务承包模式已经出台，包括许可、服务协议以及在线协议。

这种新的情况使现有的法律管理问题更复杂，并引入了新的问题。例如，地理分散的数据中心使得它很难保证数据的保密性和私密性，特别是当数据分布在不同的国家的不同司法管辖区时。目前已经有法律法规可管理敏感信息，但异构环境中的云计算服务交付，从法律的角度需要不同的方法。此外，服务消费者和供应商之间的动态合同导致了新的挑战，需要更加灵活的约定，而无法通过复杂的谈判取得冗长的合同规定。正如我们已经看到的，厂商绑定是云计算中的敏感问题，当供应商申请破产或由另一供应商接管时会发生什么事？以前的 SLA 仍然可行？这种情况为客户提供什么保证？此外，服务供应商可以规定他们之间的协议，这必将导致客户的敏感数据管理更复杂。举例来说，一般客户不会在其所租赁的服务以外进行控制，在联盟方案中，服务由某一服务供应商提供，该供应商不同于 SLA 约定中的供应商。在这种情况下，其他供应商的机密性、保密性、完整性和数据保留有什么保证？ 412

我们把与云计算相关的法律问题归为以下三个方面：隐私、安全和知识产权；商业和贸易；立法和管辖权。至于出现任何颠覆性的技术，支配参与者的交互规则总是滞后的，这种技术

会带来新的机遇和方法。在云计算的情况下，法律滞后于技术创新，尽管人们正在努力。

（1）隐私、安全和知识产权相关问题

这类问题包括因数据、应用程序和服务供应商提供的服务而产生的所有法律问题。如前所述，云计算意味着对数据和应用程序的控制交给第三方。为客户提供一个可靠的和商业上可行的服务，云计算供应商有望实施适当的程序以保护数据的完整性、数据保密性以及用户的隐私。在这种情况下，应用不同的法律有时也可能发生冲突。

关于数据的私密性和安全性的法律也许是最先进的，这是因为它不仅适用于以云计算为基础的系统。同样，不同的国家、同一国家内不同的州，有不同的方法和发展阶段。美国有相当数量的立法用于管理隐私和安全措施，这些法律也适用于云计算。一个重要的美国参与者是联盟贸易委员会（FTC），如果有组织破坏安全和隐私法规，它负责进行调查来评估任何提供金融服务或从事销售产品目的的商业活动的组织责任。特别地，数据泄露、数据保护和数据访问是现行立法关注的对象：

- 数据泄露，或者丢失未加密的电子存储信息，这是一个非常敏感的问题。避免数据泄漏对云服务消费者和供应商都非常重要。前者具有引起第三方的敏感信息可能被滥用的直接损失（社会安全号码被盗、信用卡误用或访问个人医疗数据或财务状态）。后者带来的损害是由于财务损失、潜在的诉讼、声誉受损、FTC 调查和客户的流失。因此利益双方都关心如何采取防止数据泄漏的措施。立法也支持这些措施：美国几乎所有 50 个州都要求在数据泄露发生时通知受影响的人士。因此，云供应商的系统受到损害须通知受影响的人，并与提供这些数据的人协调。数据破坏的有关法律包括健康保险流通与责任法案（HIPAA）[145] 和经济和临床健康信息技术（HITECH）法案 [144]。
- 数据保护。其他相关的注意事项是确保个人信息保密措施落实到位。格雷姆 – 里奇 – 比利雷 (GLB) 法 [141] 要求金融机构采取适当措施以保护个人信息，防止未经授权访问数据。要求管理敏感信息的云供应商：①证明他们已经部署了适当的保护敏感数据的措施；②合同规定同意防止未经授权的访问；③以上二者。根据 GLB 法案，美国联邦贸易委员会要求参与提供金融服务和产品的所有的企业都有一个书面的安全计划，以保障有关客户（保障规则 [142]）的信息。另一个重要的保护措施是红旗规则 [143]，这追使金融机构监控和阻止潜在的可能盗窃身份的可疑活动。组织机构需要记载他们采取的防止身份盗窃的策略。
- 数据访问。除了各自所有者之外，确定谁可以访问敏感数据时，这会引起法律问题。美国法律对这个问题采取了积极行动。美国爱国者法案 [146] 赋予任何代表美国政府的组织以权利，可获得存储在电子系统中的个人财务信息和学生信息。这类信息查询不会受到任何怀疑。该法案要求组织机构阻止政府访问的证明可以是有关正在进行的刑事调查。持有财务信息的机构可能无法通知客户政府对其数据的访问。这个法案所赋予的权力使云供应商处于不能给相应的用户发出通知以提供敏感信息的困境。

413

其他国家和洲际组织有不同的做法。欧洲联盟（EU）于 1995 年通过了欧盟方针保护个人数据处理和这类数据隐私转移原则。属于欧盟的国家必须通过数据保护法，涵盖处理业务和消费者数据的政府和私营实体。此外，欧盟个人数据发送到任何地域，必须有欧盟标准检测得到的数据保护级别。这直接影响了许多 EU 云供应商合法进入和运营的可能性。因此，

必须仔细研究在欧盟开展业务的可能性。

除了欧盟，许多其他国家和地区也已经通过立法保护个人隐私和数据的安全性。阿根廷有类似欧盟的做法。香港有个人资料条例 [148]，涵盖电子和非电子文档的公共和私人的数据处理。巴西将隐私权表达为宪法权利，但关于这个问题没有一个全面的法律。加拿大采取了不同的方法，有一个组织对组织的方式。每当和一个加拿大的公司进行交互时，不管其地理位置，都要使用个人信息保护与电子文件法案（PIPEDA）[149] 保护个人数据的级别。总之，加拿大组织负责保护他们转让给第三方的个人信息，不管双方是在加拿大国内还是国外。这些不同的法规不仅是法律的国际冲突的根源，也表明了计算公共基础设施服务交易的全球化在法律支持方面的不成熟状态。

另一个值得关注的因素直接关系到敏感信息在云中的处理方式，这就是知识产权（IP）权限的管理。在这种情况下，不是保护云服务供应商外部的实体数据，而是保证附属在数据上的知识产权，或者通过应用程序生成的托管在云中仍属于云计算的用户的数据，这是非常重要的。适用于保护知识产权的通常是经验法则，需要由服务使用者和服务供应商之间适当 414
的合同条款强制执行。这是特别重要的，万一云服务供应商申请破产或托管的知识产权被用于增加公司的价值，相关法律将有利于用户从破产的供应商退出。

（2）商业和贸易相关问题

云服务供应商与用户之间的具体合同条款和约定的结果可能引起法律问题。需要合同条款能够处理的问题，如：

- 双方之间约定的服务定义了什么？
- 双方各自的责任是什么？
- 依靠指定的供应商，用户运行有什么风险？
- 保证用户访问其数据的措施是什么？

合同可以认为是法律问题的第一种类型。这些问题涉及云供应商和消费者之间的协议的具体性质。SOA 普及之前，许可协议作为软件合约最常用的形式已不再适用。服务协议在云计算方案中是处理约定更合适的形式，因为来自合约实体的软件产品没有真正的传输，并且 IT 服务是作为实用工具来提供和定价的。在云计算（Iaas、Paas 或 SaaS）的任何细分市场，控制和接入点由云供应商提供，协议涵盖使用本服务的所有基本条款及条件，如果服务中断，明确规定供应商的责任。一般情况下，用户的选择非常有限，用户无法洽谈在线协议形式的合同条款。由于这是结合了两个实体的唯一文件，对供应商采用的政策有明确的规范和遵守现有法律的相关政策是非常重要的。同样重要的是，确认如果出现数据泄露，供应商的责任是什么。在大多数情况下，供应商的责任非常有限，最好的情况是有赔偿用户的服务存款。由于这些条款是不可协商的，消费者使用该服务协议时要仔细评估是否签署与供应商的协议。随着越来越多的企业和关键任务应用程序迁移到云中，必须有一个更强大和平衡的合同形式。

一些企业和商业方面的考虑也可能影响供应商和消费者之间的合约安排。例如，旨在尽量减少消费者对供应商的生存能力及存储在云中的数据的未来可用性的考虑的措施是值得关注的因素。这些问题必须在协议中涉及。有助于最大限度地降低风险的要素是旨在保护数据的完整性的措施，具体的 SLA 给出计划中断时服务可用性保证的百分比和灾难恢复措施。该供应商的生存能力是另一个方面，有助于与给定的供应商达成协议的最终决定。由于厂商

绑定仍然是一个现实，所以供应商的业务生存能力是基础，这也是因为源代码托管[⊖]在这种
415 情况下无效。未来破产的风险不是阻止持久和连续地访问数据和应用的唯一威胁。其他威胁包括终止供应商的业务活动或供应商被另一家公司收购。在这些情况下，客户的 SLA 应明确制定政策，以保证能持久访问客户数据。

（3）管辖权和程序问题

管辖权和程序的法律问题起因于云计算的两大方面：云供应商提供服务的数据和应用程序的地理位置，以及一旦诉讼适用的法律和法规。

管辖权问题大多涉及数据的位置和应用在该位置的具体法律。云服务供应商位于他们的数据中心，以降低运营成本。数据中心的位置受以最佳方式在全球范围内服务客户的愿望的影响。出于这个原因，在全球各地分布单个的云服务供应商的基础设施是很常见的。具体的问题都起因于应用于数据保护的不同的法律。例如，欧盟指令规定，在欧盟内部产生的任何个人资料以及这些数据导出到第三方国家时要遵守欧盟法律。如果不能保证适当的数据保护级别，就限制了位于不同国家的数据中心之间数据的流动性。此外，SLA 在特定的法律规定的范围内被接受，但由于数据的流动性，这样的法律未必有效，可能不能保护客户权利。更糟糕的情况是，没有具体说明适用法律就签署了协议。

管辖权问题也可能出现在转包的情况下。在云计算联盟这是一个很常见的情况：一个云供应商利用其他供应商的服务和设施为客户提供服务。对用户这多半是透明地完成的。若服务传递失败，将很难为云用户找出真正的原因。在这种情况下，该方案由这些事实复杂化：不同地区、不同组织参与提供服务给最终用户。

不同的司法管辖区导致的结果也称为法律冲突，它承认一个事实，即不同国家的法律可以相互对立地操作，即使它们涉及同一主题。一般的经验法则是，由于每个国家在它自己的领土上是独立自主的，所以一个国家的法律会影响到其内部所有人和财产，包括签订的合约和在其境内开展的行动。正如我们已经看到的，SLA 应清楚说明管辖法律，以及在提供服务给终端用户时可能涉及的其他司法管辖问题。

除了管辖权问题，有关法律诉讼程序还有其他重要方面。这将是一场法律纠纷的法院程序。在这种情况下，一方可以要求另一方以电子文档方式（电子邮件、报告、财务记录等）提供数据证据，这个过程也被称为电子证据。当对云计算数据和系统的控制被转移给第三方时出现了一个有趣的挑战，以缺乏电子存储信息为由而不提供电子证据显然是行不通的。在大多数情况下，这会导致一些费用。

传统上，企业组织员工及内部程序用所需的信息响应电子证据过程。在这种情况下，企
416 业利用云服务供应商，没有基础设施和雇员直接管理这些活动；相反，数据和它的辅助功能的维护都转移到云服务供应商。因此，挑战涉及长期地存储和检索敏感数据。由于存储通常收费，并在云中有运营成本，这可能会给想利用云的企业带来额外的负担。第二，除了供应商的商业生存，数据访问是电子证据的另一个值得关注元素。

（4）云联盟的影响

本节主要讨论从云服务消费者和供应商之间的相互作用的角度来看，利用云计算解决方案可能出现的法律问题。这些问题将如何影响云联盟？是否还有从法律角度来考虑的其他因素？

我们可以观察到，不同的云供应商之间的交互以转包形式发生，或以一个正常的交互发

⊖ 源代码托管是由第三方托管代理源代码保证金，万一当事人许可的软件申请破产，可确保软件的维护。

生在一个消费者和供应商之间，而消费者的角色由另一个云供应商扮演。因此，先前介绍的所有法律问题都适用。如上所述，不同的供应商提供的服务被组合在一起提供给终端的服务给云的用户带来了额外的问题。这是因为合同的相互作用可能会跨越不同的地域，必然涉及不同的组织。未来有希望建立一个联盟以帮助定义一个更好的法律实体，遵守参与交易的云服务供应商和用户所在国家的所有法律法规，且统一所有联盟。

11.3.4　云联盟技术

尽管云计算联盟或互联云的概念还很不成熟，但至少从操作的角度上有一些配套的技术能部署可互操作的云。这些举措大多起源于学术界，主要侧重于在不同的 IaaS 实现中支持互操作性，有限地实施在逻辑层中定义的功能。

1. Reservoir

资源和服务虚拟化无障碍，即 RESERVOIR [155]，是欧洲的一个研究项目，重点是开发支持云计算基础设施的供应商的架构，与对方动态地彼此合作来扩展自己的功能，同时保留其行政自主权。RESERVOIR 定义了一个软件栈，在 IaaS 层实现互操作并支持应用程序在基础设施的顶部基于 SLA 执行，且从基础设施供应商联盟产生。

RESERVOIR 是基于动态联盟的概念：每个基础设施供应商是一个独立的业务，有自己的业务目标，需要时可能会决定合作伙伴与其他企业。采用在每个站点部署 RESERVOIR 中间件的方式获得联盟。特定网站上的 IT 管理是完全独立的，并依赖该网站的业务目标的政策。需要时，内部 IT 资源在议价的 SLA 的范围内出租给其他供应商。RESERVOIR 的作用是协调这一过程，并最大限度地减少不同管理域之间的互操作障碍。 417

图 11-10 提供了一个 RESERVOIR 云的一般概述。该框架定义了一个基础设施覆盖，跨越多个管理域和不同的地理位置。每个站点运行 RESERVOIR 的软件栈，并提供按需执行环境，服务应用程序的组件可以在其中部署和执行。该模型引入了服务供应商和基础设施供应商之间的明确分工。服务供应商是实体，能够理解特定的业务需求，并提供服务应用程序以满足这些需求，他们没有自己的基础设施，而是从基础设施供应商租用。基础设施供应商经营 RESERVOIR 网站，并提供了几乎无限的计算、网络和存储资源池。服务供应商定义服务应用程序被建模为一个可以部署在分布式虚拟基础设施的组件的集合，组件可以显式或隐式配置。第一种情况下，服务供应商进行大小调整和容量规划研究，以确定一个给定的工作负荷条件下需要的组件的适当数目。该规范是通过最小的服务配置和一套弹性的规则获得的，在变化的工作负荷条件下 RESERVOIR 使用该规范动态地提供资源。第二种情况下，服务供应商提供一个最小的服务配置，但没有弹性规则。大小调整是由 RESERVOIR 中间件自动进行的，目标是减少过度配置。服务供应商按即付即用模式计费，并可以通过使用报告来验证计费。

服务应用程序组件表示为虚拟软件包，同时规定执行环境的要求。基础设施供应商可以 418
利用可选的任何虚拟化技术，并负责为这类应用程序提供所需的环境。确定服务应用程序虚拟环境的一个重要因素是服务清单，这是 RESERVOIR 模型的关键要素之一，按将被部署为虚拟执行环境（VEE）的组件类型指定服务应用程序的结构。服务清单包含对主映像的引用，这是一个自包含的软件栈（操作系统、中间件、应用程序、数据和配置），充分捕获组件类型的功能。其他的信息和规则规定部署多少个实例，以及如何动态地扩展和收缩实例数量。另外，该清单还规定组件的分组进入虚拟网络或形成服务应用程序的层。该清单是通过扩展开放虚拟化格式以简化互操作性并利用已经存在的流行的标准来表示的。

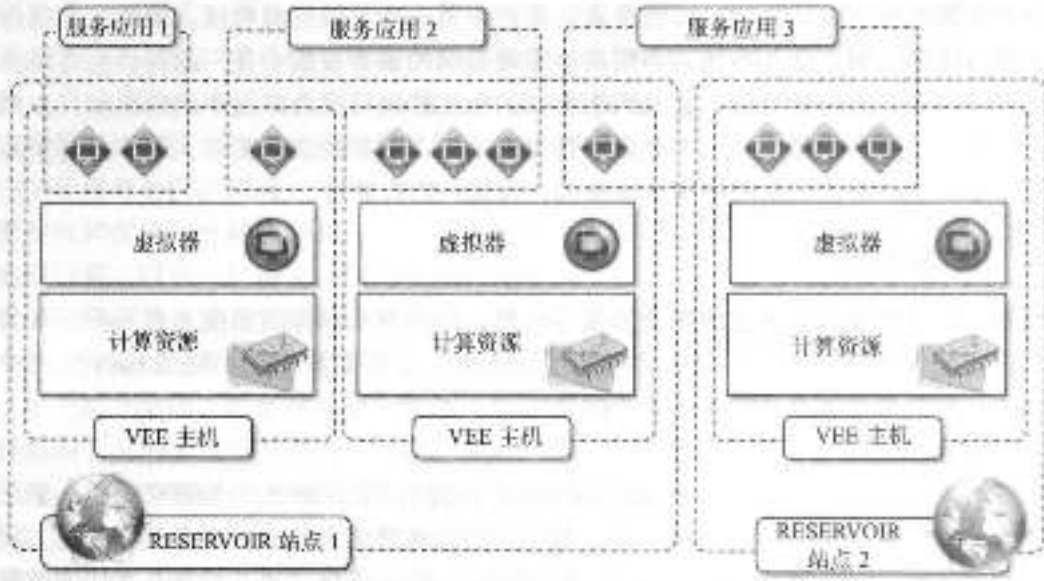


图 11-10　RESERVOIR 云部署

图 11-11 描述了一个 RESERVOIR 网站的内部架构，能执行基于联盟和 SLA 的应用程序。RESERVOIR 堆栈由三个主要部分组成：

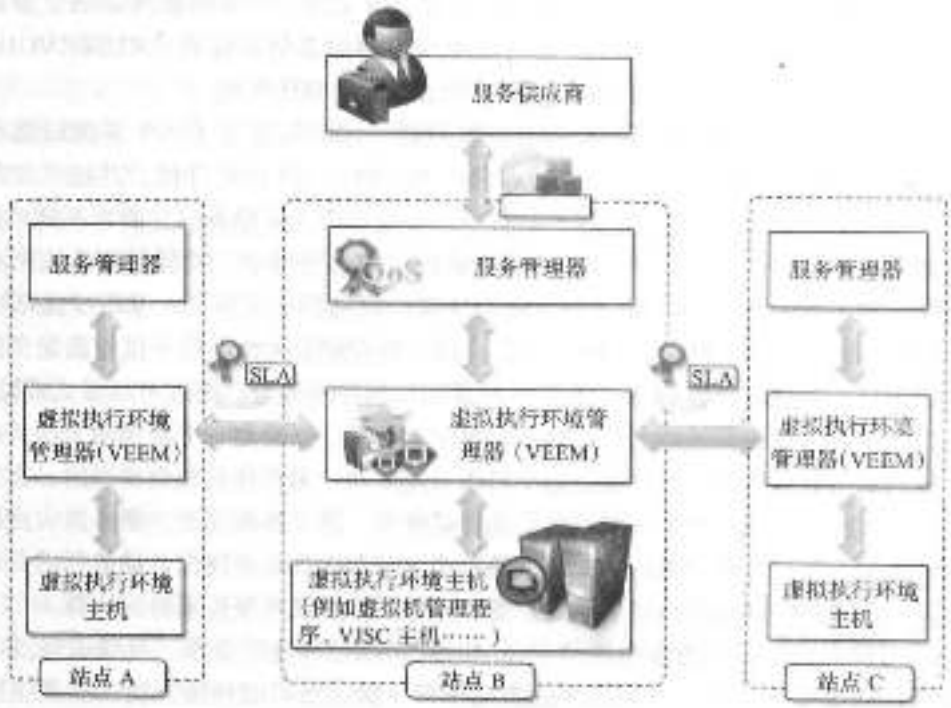


图 11-11　RESERVOIR 架构

- 服务管理器。服务管理器是最高级抽象，构成前端，服务供应商用前端来提交服务清单、协商定价和监控应用程序。该组件根据服务清单部署和规定 VEE，通过控制服务应用程序的功能监控和强制实现 SLA。
- 虚拟执行环境（VEE）管理器。这个组件是 RESERVOIR 中间件的核心，根据服务管理器表达的约束负责将最优化 VEE 放置到 VEE 主机。此外，VEE 管理器还与其他

419

网站 VEE 管理器交互为服务应用程序的执行提供另外的实例或超载时转移 VEE 到其他网站。该组件实现了云联盟的价值。

- VEE 主机（VEEH）。这是最低级抽象，与 VEE 管理器进行交互，将虚拟化平台的异构集的 IT 管理决策付诸实践。这一级也负责确保属于相同的应用程序的 VEE 之间适当和孤立的网络。该 VEEH 封装所有具体平台管理，但要求通过标准化 VEE 管理器的接口公开平台使用的虚拟化技术。

每个组件采用一个标准化的接口和协议与相邻的层互操作。这种设计决策是为了促进每一层产生多种创新方法。它让基础设施供应商自由地集成 RESERVOIR 中间件与自己现有的技术。

2. 互联云

互联云是云联盟面向服务的架构框架，支持效用驱动的云互联。互联云由一组独立元素组成，通过面向市场的系统交互来交易云资产，如应用程序的计算能力、存储和应用执行。如图 11-12 所示，互联云模型包括两个主要元素：云交易所和云协调器：

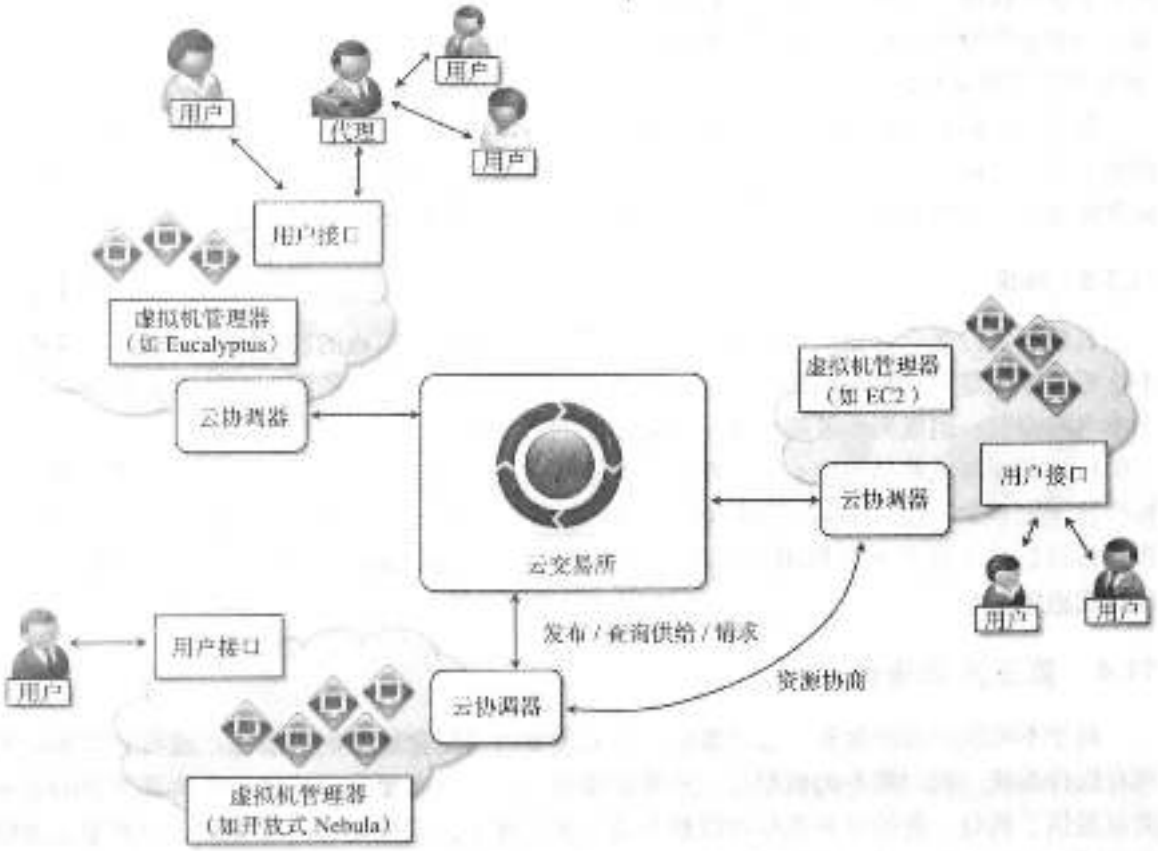


图 11-12　互联云架构

- 云交易所。这是架构中的市场组件。它提供的服务允许供应商直接交易云资产，以及允许一方注册并进行拍卖。在第一种情况下，云交易所作为一个联盟服务目录。在第二种情况下，云交易所进行拍卖。为了给联盟提供这样的服务，云交易所实现了一个基于 Web 服务的接口。云交易所允许数据中心加入和离开联盟；发布他们希望出售的资源；登记资源请求，使感兴趣的销售供应商能够为他们的资源找到潜在

的客户；查询符合具体要求的资源报价；查询匹配可用资源请求的一方；从协调器撤回供给和请求信息；在拍卖中提供资源；注册投标；咨询拍卖状况。

- 云协调器。这个组件管理联盟专用领域中的相关问题，呈现给希望加入联盟的每一方，包括前端组件（即与联盟交互的元素）和后端组件（即与相关联的数据中心进行交互的组件）。前端组件与云交易所和其他协调器交互。前者允许数据中心公布他们的供给和需求，后者让协调器了解数据中心的当前状态以决定联盟的行动是否必需。因此，不管协调器在哪检测到数据中心要求的额外的资源，它都会触发潜在的供应商发现的过程（通过与云联盟交互）。一旦发现潜在的供应商就选择最好的一个，由协调器联系远程协调和协商。同样地，当协调器检测到本地资源是能够充分利用的，便可以在云交易所发布资源报价或者寻找在交换服务登记的多个需求之间的匹配。

420 / 421

协商各方之间遵守 Alternate Offers 协议。每个协调器愿意支付的资源的价值以及各协调器要出售资源的价值，并不是联盟定义的；相反，根据利用率与效益估算，每个协调器自由地给资源确定价值，而且他们可以自由地拒绝不感兴趣的供给。供应商决定购买或出售资源的多少、时间、价格和方式，这种灵活性是供应商加入互联云联盟的一大动力。供应商如果不同意价格可以拒绝购买或出售资源，同样，他们可以倾向于选择似乎更有利可图的方法（例如拍卖或固定价格）。

此外，互联云的作用主要是在业务层。这意味着有关安全、虚拟机镜像和网络的问题不由框架处理。同样，供应商的职责和加入联盟的期望也不由框架解决。因此，这些层的现有的方法或新的解决方案，可以在不改变架构的情况下应用在互联云。

11.3.5　结论

云服务供应商也许是互相竞争的，建立一个联盟需要有明确的目标和激励机制。其中一个主要的原因是可以提高客户可感知的整体 QoS。创建联盟方案的第二步是为云联盟定义一个参考模型和一组规则和策略。这一步采用面向市场模型，也是租用资源发展的机遇。在这一阶段的发展与成熟仍非常有限。在一个非常低的层次上，不同的云供应商的联盟是通过互操作技术实现的。这一领域已经得到长足的发展，特别是支持互操作性的标准草案的形成。我们已经讨论了这个方向最相关的举措，并分析了一些在 IaaS 级可用的云联盟系统的参考和实现原型。

11.4　第三方云服务

对于不同供应商的服务，云计算的一个关键要素是组合服务的可能性，或将它们集成到现有软件系统。面向服务的模型是云计算的基础，使此方案更容易实现，并为开发新的服务

422

类型提供了机会。新的服务类型可以称为第三方云服务。这些是已经存在的云计算服务增值的结果，从而为客户提供不同的和更先进的服务。增加的值可以通过巧妙地协调现有服务去创建或在现有基础服务之上实现附加功能。

除了这个一般的定义，没有具体的特征能描述该类服务的特点，本节将描述第三方服务的一些例子。

11.4.1　MetaCDN

MetaCDN [158] 为用户提供了内容分发网络（CDN）[159]，通过利用和共同管理异构

存储云提供服务。它包括各种软件，协调不同的云存储供应商的服务产品，并用它们作为该用户存储内容的分布弹性存储。MetaCDN 为用户提供了 CDN 的高级别服务，发布内容并与低层存储云接口交互，最优地将用户的内容根据其需求的预期地理位置进行放置。利用云作为存储后端，小企业也能使用一个复杂（和一般来说昂贵的）的内容发布服务。

MetaCDN 的架构如图 11-13 所示。该 MetaCDN 接口通过 Web 用户和应用程序公开其服务，用户与门户网站进行交互，而应用程序利用 Web 服务方式提供编程访问。MetaCDN 的主要业务是建立云存储上的部署和管理。门户网站提供了一个更直观的界面，为用户提供基本的需求，而 Web 服务提供了访问 MetaCDN 的全部功能，并允许更复杂和更详细的部署。

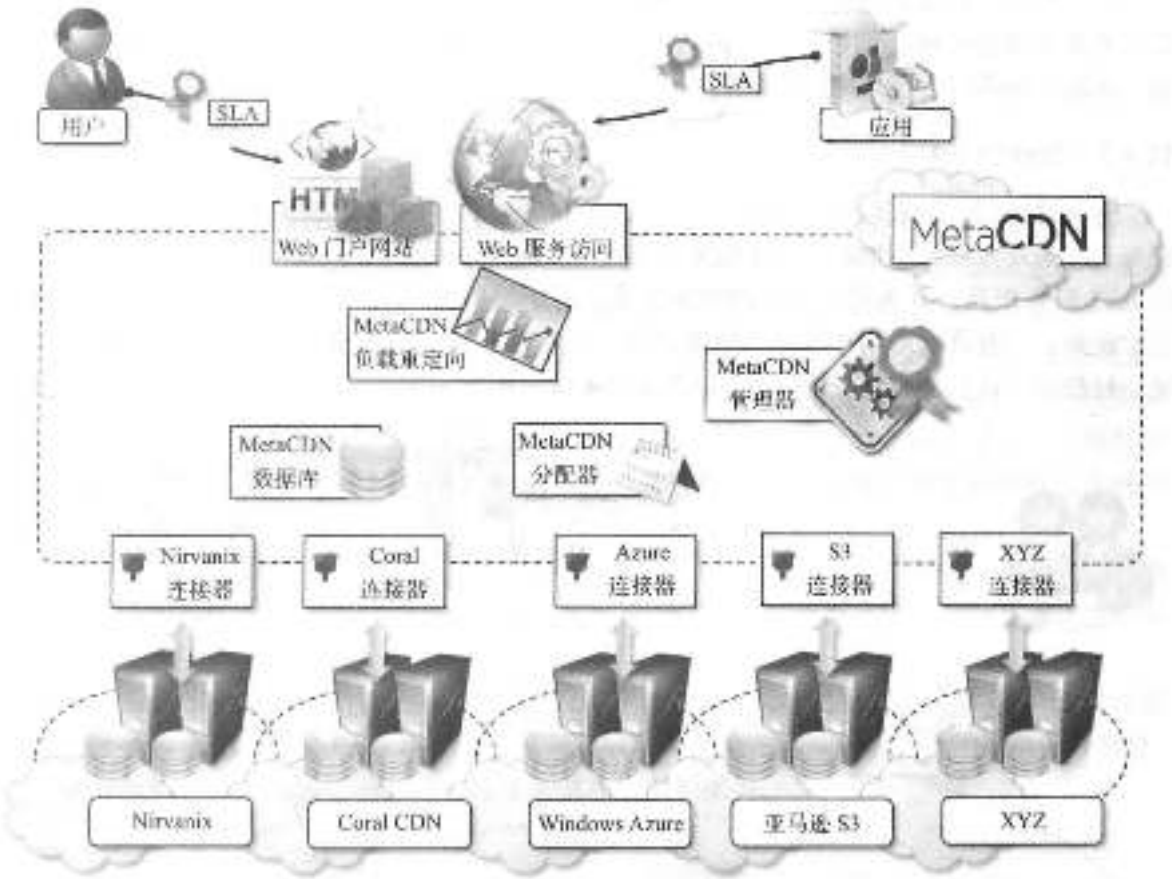


图 11-13　MetaCDN 架构

特别地，可以选择四个不同的部署选项：

- 覆盖范围和性能优化的部署。在这种情况下，MetaCDN 能部署尽可能多的副本到所有可用的位置。
- 直接部署。在这种情况下，MetaCDN 允许对内容选择部署区域和将选定区域和服务与这些区域支持的供应商匹配。
- 成本优化部署。在这种情况下，MetaCDN 部署尽可能多的副本在所确定的部署请求的位置。可用存储传输补贴和预算将用于部署副本，并让它们尽可能长时间地保持活跃。
- QoS 优化配置。在这种情况下，MetaCDN 选择能够更好地匹配连接到部署的 QoS 要求的供应商，例如特定位置的平均响应时间和吞吐量。

为提供这些服务，需要一系列组件协调其活动。这些构成了 MetaCDN 带来的附加价值。用户在云的顶端直接使用存储云。三个特别重要的组成部分是 MetaCDN 管理器、MetaCDN QoS 监控器和负载重定向器。管理器负责确保所有内容部署满足预期的 QoS。它由这个活动中的监控器支持，不断地探测存储器供应商和监控数据传输，以评估每个供应商的绩效。内容服务由负载重定向器控制，它负责重定向用户的内容请求到系统的给定条件和部署过程中指定的选项。与存储云交互是由连接器管理的，它抽象了供应商公布的不同的接口，提供 MetaCDN 系统内的一个统一的接口。

423
|
424

如前所述，MetaCDN 的核心价值在于软件集成，使统一使用的异构存储云作为一个单一的、大型的、分布式内容发布网络。其优点是不仅按访问成本提供 CDN 服务，而且还丰富了原来的服务产品，能够为现有的云服务提供额外的功能，从而创造一个新的、更复杂的、新的市场部门会感兴趣的服务。

11.4.2　SpotCloud

SpotCloud 是虚拟市场的一个实例。作为消费者和服务供应商之间的交易计算和存储的媒介，它给双方提供附加值。对于服务的消费者，它是一个市场目录，可以浏览和比较不同的 IaaS 服务产品，并选择最合适的解决方案。对于服务供应商而言，它提供宣传产品的机会。此外，它允许有可用计算能力的用户通过在基础设施上部署 SpotCloud 所需的运行时环境，轻松地将自己变成服务供应商（见图 11-14）。

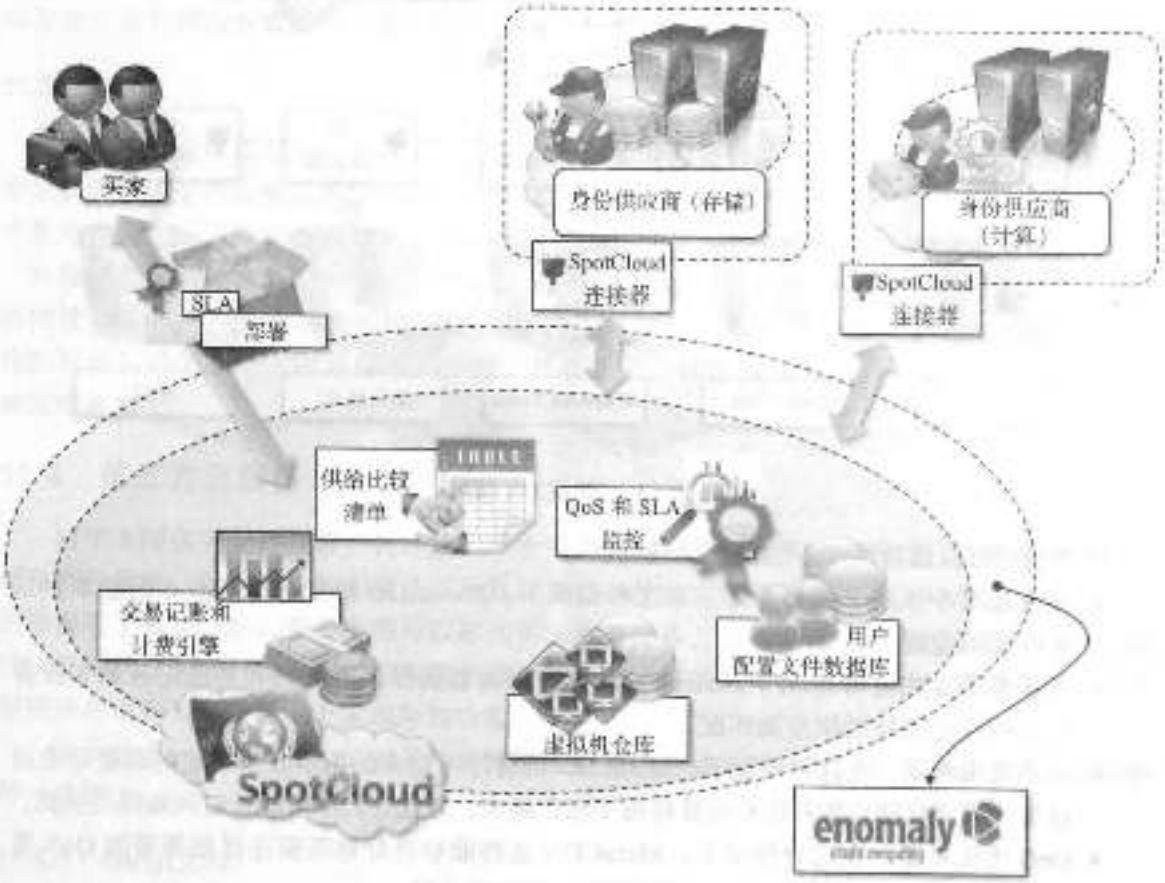


图 11-14　SpotCloud 市场架构

SpotCloud 不仅是一个 IaaS 供应商和分销商的促成者，它的中介作用还包括与资源使用相关的交易的完整记账。用户在 SpotCloud 账户中存款，并通常按照即付即用模式支付给容量的卖家。SpotCloud 保留用户计费的数量的百分比。此外，通过利用一个统一的运行环境和虚拟机管理层，它为用户提供厂商免绑定的解决方案，这可能是战略的具体应用。

本节提出的两个例子给出了不同性质的第三方服务的实现方法：MetaCDN 为终端用户提供不同于简单的云存储产品的服务；SpotCloud 不改变最终提供给终端用户的服务类型，但用附加功能丰富了服务类型，可以更有效地利用服务。这是两个市场方式的例子，云计算作为一种更智能地使用 IT 资源的方法正在不断发展。

本章小结

本章介绍了描述和驱动云计算的研究和发展的一些高级主题。当云计算技术日益巩固时，这些主题变得尤为重要。

考虑到支持云供应商服务产品的计算设备的巨大规模，节能解决方案起到了根本性的作用，包括更智能、更环保的数据中心设计，以及虚拟机优化布局和能源驱动的服务器管理。节能解决方案不仅有助于降低电费，也减少了计算设施对环境的影响。虽然高级云计算带来的这个问题还存在争议，但它有可能提供更环保的技术。

425
2
426

除了高效节能外，面向市场的云计算（MOCC）和互联云是最优服务交付的重要进展。MOCC 是云计算的正常演变，该技术使云计算得到巩固和商品化。IaaS 细分市场的成熟有利于建立更复杂的市场模式。最新进展表明，以云服务代理和拍卖为基础的模型对于更好的资源管理策略是有用的。MOCC 的最终目标是创造有利于计算实用程序交易领域的全球虚拟市场。

此外，云联盟支持 MOCC 实施，通过部署基础设施和定义组织模型使云供应商之间能够互操作。目前，客户依赖单个供应商或自行从不同的云供应商组合服务。随着云联盟的实现，用户将能够利用多个云。

本章最后讨论了第三方服务，如 MetaCDN 和 SpotCloud，这是通过利用多个云供应商提供的服务建立的。一些 SaaS 解决方案可能会被归类为第三方服务，基本上是增值分销商。这意味着他们通过组合和使用其他云服务来实现增值和销售。

习题

1. 什么是能效计算？为什么它是云计算的根本？
2. 所有功耗管理技术的基本原理是什么？
3. DVFS 代表什么？它在云计算中是如何起作用的？
4. 比较硬件和软件功耗管理技术。
5. 什么是服务器整合？为什么它也是一项功耗管理策略？
6. 什么是面向市场的云计算？
7. 实现一个基于 MOCC 的系统的主要部件是什么？
8. 什么是虚拟市场？
9. 什么是云联盟？
10. 表示互联云和云联盟的概念的三个不同层次是什么？
11. 建立一个联盟的动机是什么？

12. 业务和逻辑层面必须解决什么样的挑战？
13. 什么是基础设施级的领域？在云计算参考模型哪一层表示这个级别？
14. 什么样的标准和协议可用于实现云联盟的互操作性？
15. 什么是互联云？
16. 描述 RESERVOIR 项目的主要特征。
17. 描述在联盟环境中标准的作用。
18. 什么是云联盟环境中的安全问题？
19. 云联盟的背景下可能产生的法律问题是什么？
20. 什么是第三方云服务？
[427] 21. 给出第三方云服务的一些例子。

参考文献

[1] Tanenbaum AS, Van Steen M. Distributed systems: principles and paradigm. Upper Saddle River, NJ, USA: Prentice Hall PTR; 2001.

[2] Coulouris G, Dollimore J, Kindberg T. Distributed systems: concepts and design. 4th ed. Boston, MA, USA: Addison Wesley; 2005.

[3] Pfister G. In search of clusters. 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR; 1998.

[4] Buyya R. High-performance cluster computing: architecture and systems. Upper Saddle River, NJ, USA: Prentice Hall PTR; 1999.

[5] Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the condor experience. Concurrency Comput Pract Exper 2005;17(2–4):323–56.

[6] Sunderam VS. PVM: a framework for parallel distributed computing. Concurrency: Pract Exper 1990;2(4):315–39.

[7] Gropp W, Lusk E, Doss N, Skjellum A. A high-performance, portable implementation of the MPI message passing interface. Parallel Comput 1996;22(6):789–828.

[8] Foster I, Kesselman C. The grid: blueprint for a new computing infrastructure. San Francisco, CA, USA: Morgan Kaufmann Publishing; 1999.

[9] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. Int J High Perform Comput Appl 2001;15(3).

[10] Broberg J, Venugopal S, Buyya R. Market oriented grids and utility computing: the state of the art and future directions. J Grid Comput 2008;6(3):255–76 [Springer Netherlands].

[11] Buyya R, Bubendorfer K. Market-oriented grid computing and utility computing. Hoboken, NJ, USA: Wiley; 2009.

[12] Foster I. Globus toolkit version 4: software for service-oriented systems. IFIP International conference on network and parallel computing. Lecture notes in computer science (LNCS) 3779. Springer-Verlag; 2005. pp. 2–13.

[13] Gentzsch W. Sun grid engine: towards creating a compute power grid. Proceedings of the first IEEE/ACM international symposium on cluster computing and the grid (CCGrid 2001). Brisbane, Australia: IEEE Computer Society; 2001. pp. 35–36.

[14] Anderson DP. BOINC: a system for public-resource computing and storage. Proceedings of the fifth IEEE/ACM international workshop on grid computing. Pittsburgh, PA, USA: 2004.

[15] Buyya R, Venugopal S. The gridbus toolkit for service oriented grid and utility computing: an overview and status report. Proceedings of the first IEEE international workshop on grid economics and business models (GECON 2004). NJ, USA: IEEE Press; 2004.

[16] Popek GJ, Goldberg RP. Formal requirements for virtualizable third generation architectures. Commun ACM 1974;17(7):412–21.

[17] Whitaker A, Shaw M, Gribble D. Denali: a scalable isolation kernel. Proceedings of the tenth ACM SIGOPS European workshop. France: Saint-Emilion; 2002.

[18] Chaudhury A, Kuilboer JP. e-Business and e-Commerce infrastructure. NY, USA: McGraw Hill; 2002.

[19] Papazoglou MP, Traverso P, Dustdar S, Leymann F. Service-oriented computing: state of the art and research challenges. IEEE Comput 2007;40(11):38–45 [IEEE Computing Society]

[20] Papazoglou MP, van den Heuvel, W-J. Service-Oriented Architecture: Approaches, Technologies, and Research Issues. Very Large Data Bases (VLDB) J 2007;16(3):389–415 VLDB Endowment Inc.

[21] Papazoglou MP. Web services: principles and technology. Hoboken, NJ, USA: Prentice Hall; 2007.

[22] W3C. Web service definition language (WSDL) 1.1. [Online Document] Available at <www.w3.org/TR/wsdl/>.

[23] W3C. Simple object access protocol (SOAP) specifications. [Online Document]. Available at <www.w3.org/TR/soap/>.

[24] Liberty J, Hurwitz D. Programming ASP.NET. 3rd ed. Sebastopol, CA, USA: O'Reilly Media; 2005.
[25] Ka lok Tong K. Developing web services with apache axis2. Birmingham, UK: TipTec Development; 2008.
[26] DiNucci D. Fragmented future. Design & New Media; 1999 [Online Document]. Available at <www.cdinucci.com/Darcy2/articles/Print/Printarticle7.html>
[27] O'Reilly T. What is web 2.0? Design patterns and business model for the next generation of software. O'Reilly Media; 2005 [Online Document]. Available at <http://oreilly.com/pub/a/web2/archive/what-is-web-20.html>
[28] Armbrust M, Fox A, Griffith R, Joseph A, Katz R, Konwinski A, et al. Technical Report No. UCB/EECS-2009-28 Above the clouds: a berkeley view of cloud computing. USA: University of California at Berkeley; 2009
[29] Reese G. Cloud application architectures: building applications and infrastructure in the cloud. Sebastopol, CA, USA: O'Reilly Media Inc.; 2009.
[30] Buyya R, Yeo CS, Venugopal S. Market oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. Proceedings of the tenth conference on high performance computing and communications (HPCC 2008, IEEE Press, Los Alamitos, CA). Dalian, China: 2008.
[31] Bennett KH, Layzell PJ, Budgen D, Brereton P, Macaulay LA, Munro M. Service-based software: the future for flexible software. Proceedings of the seventh Asia-Pacific software engineering conference (ASPEC 2000). Singapore: IEEE Computer Society; December 2000.
[32] Strategic backgrounder: software as a service. Software & Information Industry Association; February 2001. [Online Document] Available at <www.siia.net/estore/pubs/SSB-01.pdf>.
[33] Koenig M, Gupeill B, McNee B, Cassell J. SaaS 2.0: Software-as-a-Service as next gen business platform. Saugatuck Technologies; 2006. Available at <www.saugatech.com/239order.htm>
[34] Staten J, Yates S, Ryme J, Gillett F, Nelson LE. Deliver cloud benefits inside your walls: economic and self-service gains are within reach. Forrester Research Inc.; 2009.
[35] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, et al. Xen and the art of virtualization. Proceedings of the 19th ACM symposium on operating systems principles (SOSP). Lake George, NY, USA: October 2003.
[36] Warnke R, Ritzau T. ISBN: 978-3-8370-0876-0 qemu-kvm& libvirt. 4th ed. Norderstedt: Books on Demand GmbH; 2010
[37] Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, et al. The eucalyptus open-source cloud computing system. In: Proceedings ninth IEEE/ACM international symposium on cluster computing and the grid (CCGrid 2009). Shanghai, China: May 2009.
[38] Llorente I, Moreno-Vozmediano R, Montero. R. Cloud computing for on-demand grid resource provisioning. Advances in parallel computing. 18. IOS Press; 2009.
[39] Sotomayor B, Keahey K, Foster I. Combining batch execution and leasing using virtual machines. HPDC '08: Proceedings of the 17th international symposium on high performance distributed computing. ACM; 2008. pp. 87–96.
[40] Venugopal S, Broberg J, Buyya R. OpenPEX: an open provisioning and execution system for virtual machines. Proceedings of the 17th international conference on advanced computing and communications (ADCOM 2009). Bengaluru, India: December, 14–18, 2009.
[41] Costanzo A, Assunção M, Buyya R. Harnessing cloud technologies for a virtualized distributed computing infrastructure. IEEE Internet Comput 2009;13(5):14–22.
[42] Vecchiola C, Chu X, Mattess M, Buyya R. Aneka: integration of public and private clouds. Cloud computing: principles and paradigms. Hoboken, NJ, USA: Wiley; 2011.
[43] Mell P, Grance T. NIST working definition on cloud computing. National Institute of Standard and Technology (NIST): [Online Document] Available at <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.
[44] Briscoe G, De Wilde P. Digital ecosystems: evolving service oriented architectures. Proceedings of the first international conference of bio inspired models of network Information and Computing Systems (BIONETICS). Cavalese, Italy: IEEE Press; December 2006.
[45] G. Briscoe and M. Alexandros. Digital ecosystems in the clouds: towards community cloud computing. Proceedings of the third IEEE international conference on digital ecosystems and technologies (DEST 2009). New York, USA: IEEE; 2009. pp. 103–108.
[46] Sriram I, Khajeh-Hosseini A. Research agenda in cloud technologies Technical Report. UK: University

of Bristol; 2010. [Online Document] Available at <http://arxiv.org/ftp/arxiv/papers/1001/1001.3259.pdf>.

[47] Khajeh-Hosseini A, Sriram I, Sommerville I. Research challenges for enterprise cloud computing. UK: University of Bristol; 2010 [Online Document] Available at <http://arxiv.org/ftp/arxiv/papers/1001/1001.3257.pdf>.

[48] Vouk AM. Cloud computing issues, research, and implementations, Proceedings of the 30th international conference of information technologies and interfaces (ITI 2008). Cavtat/Dubrovnick, Croatia: June 2008. pp. 31–40.

[49] Birman K, Chockler G, van Renesse R. Toward a cloud computing research agenda. SIGACT News 2009;40(2):68–80.

[50] Youssef L, Butrico M, Da Silva D. Towards a unified ontology of cloud computing. Grid computing environments workshop (GCE08). Austin, Texas, USA: IEEE; November 2008.

[51] Open virtualization format specification. Distributed Management Task Force; February 2009. [Online Document] Available at <www.dmtf.org/standards/published_documents/DSP0243_1.0.pdf>.

[52] Standard ECMA-334. C# Language specification. Available at <www.ecma-international.org/publications/standards/Ecma-334.htm>.

[53] Standard ECMA-335. Common language infrastructure (CLI). Available at <www.ecma-international.org/publications/standards/Ecma-335.htm>.

[54] Ghemawat S, Gobioff H, Leung S-T. The google file system. Proceedings of the 19th ACM symposium of operating systems principles (SOSP'03). Lake George, NY, USA: ACM; October 2003. pp. 29–43.

[55] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Proceedings of the 6th symposium on operating system design and implementation (OSDI'04) USENIX. San Francisco, CA, USA: December 2004.

[56] Jin C, Buyya R. Dataflow computations on enterprise grids. ISBN: 978-981-283-943-5. In: Misra S, Misra SC, Woungang I, editors. Selected topics in communication networks and distributed systems. Singapore: World Scientific; 2010.

[57] Agha G. ISBN: 0-262-01092-5 Actors: a model of concurrent computation in distributed systems. Cambridge, MA, USA: MIT Press; 1986

[58] Buyya R, Ranjan R, Calheiros RN. InterCloud: utility-oriented federation of cloud computing environments for scaling of application services. Proceedings of the tenth international conference on algorithms and architectures for parallel processing (ICA3PP'10), LNCS 6081. Springer; 2010. pp. 13–31.

[59] Fox GC, Williams RD, Messina PC. Parallel computing works. San Francisco, CA, USA: Morgan Kaufmann; 1994.

[60] Gray J, Reuter A. Transaction processing: concepts and techniques. San Mateo, CA, USA: Morgan Kaufmann; 1992.

[61] Raicu I. Many-task computing: bridging the gap between high throughput computing and high performance computing. Saarbrücken, Germany: VDM Verlag; 2009.

[62] Wilde M, Foster I, Iskra K, Beckman P, Zhang Z, Espinosa A, et al. Parallel scripting for applications at the petascale and beyond. IEEE Comp 2009;42(11):50–60 IEEE Computing Society.

[63] Raicu I, Zhang Z, Wilde M, Foster I, Beckman P, Iskra K, Clifford B. Toward loosely coupled programming on petascale systems. Proceedings of the 2008 ACM/IEEE conference on supercomputing (SC08). Austin, TX, USA: November 15–21, 2008.

[64] Hollingsworth D. The workflow reference model. Workflow Management Coalition, Document nr. tc00-1003. 1993. [Online Document] Available at <www.wfmc.org/standards/docs/tc003v11.pdf>.

[65] The OASIS Committee. Web services business process execution language (WS-BPEL) Version 2.0. 2007. [Online Document] Available at http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[66] Barker A, van Hemert J. Scientific workflow: a survey and research directions. Proceedings of the seventh international conference parallel processing and applied mathematics (PPAM 2007). September 9–12, 2007.

[67] Yu J, Buyya R. A taxonomy of scientific workflow systems for grid computing. SIGMOD Rec 2005;34(3):44–9.

[68] Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, et al. Scientific workflow management and the kepler system. Concurrency Comput Pract Exper 2005;18(10):1039–45.

[69] Couvares P, Kosar T, Roy A, Weber J, Wenger K. Workflow management in condor. Workflow for e-Science. London: Springer; 2007. pp. 357–375.

[70] Pandey S, Karunamoorthy D, Buyya R. Workflow engine for clouds ISBN-13: 978-0470887998 In: Buyya R, Broberg J, Goscinski A, editors. Cloud computing: principles and paradigms. New York, USA: Wiley Press; 2010.
[71] Vecchiola C, Kirley M, Buyya R. Multi-objective problem solving with offspring on enterprise clouds. Proceedings of the tenth international conference on high-performance computing in asia-pacific region (HPC Asia 2009). Kaoshiung, Taiwan: 2009.
[72] Buck JT, Ha S, Lee EA, Masserschmitt DG. Ptolemy: a framework for simulating and prototyping heterogeneous systems. J Comput Simul 2004;4:155–82.
[73] Moore R, Prince TA, Ellisman M. Data-intensive computing and digital libraries. Commun ACM 1998;41(11):56–62.
[74] Gorton I, Greenfield P, Szalay A, Williams. R. Data-intensive computing in the 21st century. IEEE Comput 2010.
[75] Johnston W. High-speed, Wide area, data-intensive computing: a ten years retrospective. Proceedings of the seventh symposium high-performance distributed computing. IEEE Press; 1998. pp. 280–291.
[76] Thompson M, Johnston W, Guojun J, Lee J, Tierney B, Terdiman JF. Distributed healthcare imaging information systems. PACS Design and Evaluation: Engineering and Clinical Issues. SPIE Medical Imaging; 1997.
[77] Johnston W, Jin G, Larsen C, Lee J, Hoo G, Thompson M, et al. Real-time generation and cataloguing of large object in widely distributed environments. Int J Digit Libr Spec Issue Digit Libr Med 1997; November.
[78] Lau S, Leclerc Y. Technical Note 540 TerraVision: a terrain visualization system. Menlo Park, CA: SRI International; 1994.
[79] Venugopal S, Buyya R, Kotagiri. R. A taxonomy of data grids for distributed data sharing, management, and processing. ACM Comput Surv 2006;38(1):1–53.
[80] Chervenak A, Foster I, Kesselman C, Salisbury C, Tuecke S. The data grid: towards an architecture for the distributed management and analysis of large scientific datasets. J Netw Comput Appl 2000;23(3):187–200.
[81] Stark C, Breitkreutz BJ, Chatr-Aryamontri A, Boucher L, Oughtred R, Livstone MS, et al. The BioGRID Interaction Database: 2011 Update. Nucleic Acids Res 2010.
[82] Jacobs. A. The pathologies of big data. NY, USA: ACMQueue, ACM Press; 2009.
[83] White T. Hadoop: the definitive guide. Sebastopol, CA, USA: O'Reilly & Associates, Inc.; 2009.
[84] Gu Y, Grossman RL. Sector and sphere: the design and implementation of a high performance data cloud. Phil Trans R Soc 2009;367(1897):2429–45 A 28.
[85] Ceri S, Pelagatti G. Distributed databases: principles and systems. New York, USA: McGraw-Hill; 1984.
[86] Codd. EF. A relational model for large shared data banks. Commun ACM 1970;13(6):377–87.
[87] Oram A. Peer-to-peer: harnessing the power of disruptive technologies. Sebastopol, CA, USA: O'Reilly & Associates, Inc.; 2001.
[88] Schmuck F Haskin R. GPFS: a shared-disk file system for large computing clusters. Proceedings of file and storage technologies 2002 (FAST 2002). Monterey, CA, USA: January 2002.
[89] Gu Y, Grossman RL. UDT: UDP-based data transfer for high-speed wide area networks. Comput Netw 2007;51(7) (Elsevier)
[90] Chodorow K, Dirolf M. ISBN: 978-1449381561 MongoDB: the definitive guide. Sebastopol, CA, USA: O'Reilly Media; 2010.
[91] Anderson JC, Lehnardt J, Slater. N. ISBN: 978-0596155896. CouchDB: the definitive guide: time to relax. Sebastopol, CA, USA: O'Reilly Media; 2010.
[92] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, et al. Dynamo: amazon's highly available key-value store. Proceedings of the 21st symposium on operating system principles. Stevenson, WA, USA: October 14–17, 2007.
[93] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, et al. Bigtable: a distributed storage system for structured data. Proceedings of the seventh USENIX symposium on operating system design and implementation. Seattle, USA: November 2006.
[94] Lakshman A Malik P. Cassandra: a decentralized structured storage system. Proceedings of the third ACM SIGOPS international workshop on large scale distributed systems and middleware (LADIS 2009). Big Sky, MT, USA: October 2009.
[95] Pike R, Dorward S, Griesemer R, Quinland S. Interpreting the data: parallel analysis with sawzall. Sci

Program J 2005;13(4):227–98.
[96] M. Burrows. The chubby lock service for loosely coupled distributed systems. Proceedings of the seventh USENIX symposium on operating system design and implementation (OSDI '06). Seattle, WA, USA: November 2006.
[97] Chu CT, Kim SK, Lin YA, Yu YY, Bradski G, Ng AY, et al. Map-reduce for machine learning on multicore. In: Schölkopf B, Platt J, Hoffman T editors. Advances of neural information processing systems. 19, 2007.
[98] Yang HC, Dasdan A, Hsiao RL, Stott Parker D. Map-Reduce-Merge: simplified relational data processing on large clusters. Proceedings of the 2007 ACM SIGMOD international conference on management of data. Beijing, China: June 2007.
[99] Ekanayake J, Li H, Zhang B, Gunarathne T, Bae S-H, Qiu J, et al. Twister: a runtime for iterative MapReduce. The first international workshop on MapReduce and its applications (MAPREDUCE'10), HPDC2010. Chicago, Illinois, USA: June 2010.
[100] Moretti C, Bui H, Hollingsworth K, Rich B, Flynn P, Thain D. All-pairs: an abstraction for data-intensive computing on campus grids. IEEE T Parall Distr Syst 2010;21(1):33–46.
[101] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: distributed data-parallel programs from sequential building blocks. European conference on computer systems (EuroSys). Lisbon, Portugal: March 21–23, 2007.
[102] Yu Y, Isard M, Fetterly D, Budiu M, Erlingsson U, Gunda PK, et al. DryadLINQ: a system for general-purpose distributed data-parallel computing using a high-level language. Symposium on operating system design and implementation (OSDI). San Diego, CA: December 8–10, 2008.
[103] Calvert C, Kulkarni D. Essential LINQ. Boston, MA, USA: Addison-Wesley Professional; 2009.
[104] Perry DE, Wolf AL. Foundations for the study of software architecture. ACM SIGSOFT Softw Eng Notes 1992;17(4):40–52 ACM Press.
[105] Garlan D, Shaw M. Software architecture: perspectives on an emerging discipline. Upper Saddle River, NJ: Prentice-Hall; 1996.
[106] Gamma E, Helm R, Johnson R, Vlissides JM. ISBN: 0201633612 Design patterns: elements of reusable software design. Boston, MA, USA: Addison-Wesley; 1995.
[107] Box D. A guide to developing and running connected systems with indigo, MSDN magazine. January 2004. Available at: <http://msdn.microsoft.com/en-us/magazine/cc135505.aspx>.
[108] Erl T. Service-oriented architecture: concepts, technology, and design. Upper Saddle River, NJ, USA: Prentice Hall PTR; 2009.
[109] Bell M. Introduction to service-oriented modeling. In: Service-oriented modeling: service analysis, design, and architecture. ISBN: 978-0-470-14111-3. Hoboken, NJ, USA: Wiley & Sons, 2008.
[110] OASIS. Reference architecture foundation for service oriented architecture, version 1.0, October 2009. [Online Document] Available at <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf>.
[111] Nelson BJ. PARC CSL-81-9 Remote procedure call. Palo Alto, CA: Xerox Palo Alto Resource Center; 1981.
[112] Birrell AD, Nelson BJ. Implementing remote procedure calls. ACM T Comput Syst 1984;2(1).
[113] Slee M, Agarwal A, Kwiatkowski M. Thrift: scalable cross-language service implementation. [Online Document] Available at <http://incubator.apache.org/thrift/static/thrift-20070401.pdf>.
[114] Buyya R, Ranjan R, Calheiros RN. InterCloud: utility-oriented federation of cloud computing environments for scaling of application services. Proceedings of the tenth international conference on algorithms and architectures for parallel processing (ICA3PP 2010, LNCS 6081, Springer, Germany). Busan, South Korea: May 21–23, 2010. pp. 13–31.
[115] Beloglazov A, Buyya R, Lee YC, Zomaya A. A taxonomy and survey of energy-efficient data centers and cloud computing systems. In: Zelkowitz M, editor. Advances in computers, 82. Amsterdam, The Netherlands: Elsevier; 2011.
[116] Buyya R, Beloglazov A, Abawajy J. Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges. Proceedings of the 2010 international conference on parallel and distributed processing techniques and applications (PDPTA 2010). Las Vegas, NV, USA: July 12–15, 2010.
[117] Garg S, Buyya. R. In: Murugesan S, Gangadharan G, editors. Green cloud computing and environmental sustainability, harnessing green it: principles and practices. West Sussex, UK: Wiley Press; 2011.
[118] Kaplan J, Forrest W, Kindler N. Revolutionizing data center energy efficiency. San Francisco, CA, USA: McKinsey; 2008.
[119] Oppenheimer D, Ganapathi A, Patterson DA. Why do internet services fail, and what can be done about

it? Proceedings of the fourth conference on USENIX symposium on internet technologies and systems. vol. 4, Seattle, WA: March 26–28, 2003.

[120] Baliga J, Ayre RWA, Hinton K, Tucker RS. Green cloud computing: balancing energy in processing, storage and transport. Proc IEEE 2011;99(1):149–67.

[121] Kleinrock L. A vision for the Internet. ST J Res 2005;.

[122] Vecchiola C, Duncan D, Buyya R. The structure of new IT Frontier: market oriented cloud computing–part II. Strateg Facil Mag 2010;(Issue 10):59–66 [Pacific & Strategic Holdings Pte Ltd, Singapore]

[123] Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comp Syst 2009;25 (6):599–616.

[124] Garg SK, Buyya R. Cooperative Networking Market-oriented resource management and scheduling: a taxonomy and survey. New York, USA: Wiley Press; 2011.

[125] Smith, R. The contract-net protocol: high-level communication and control in a distributed problem solver. IEEE Trans Comput 1980;4:1104–13.

[126] Fu Y, Chase J, Chun B, Schwab S, Vahdat. A. SHARP: an architecture for secure resource peering. ACM SIGOPS Oper Syst Rev 2003;37(5):133–48.

[127] Lai K, Rasmusson L, Adar E, Zhang L, Huberman BA. Tycoon: an implementation of a distributed, market-based resource allocation system. Multiagent Grid Syst 2005;1(3):169–82.

[128] AuYoung A, Chun B, Snoeren A, Vahdat A. Resource allocation in federated distributed computing infrastructures. In: Proceedings first workshop on operating system and architectural support for the on-demand IT infrastructure. Boston, MA, USA: October 2004.

[129] Irwin DE, Chase JS, Grit LE, Yumerefendi AR, Becker D, Yocum K. Sharing networked resources with brokered leases. Proceedings of 2006 USENIX annual technical conference, USENIX 2006. Boston, MA, USA: June 2006.

[130] Mattess M, Vecchiola C, Buyya R. Managing peak loads by leasing cloud infrastructure services from a spot market. 12th IEEE international conference on high performance computing and communications (HPCC 2010). Melbourne, Australia: 2010.

[131] Buyya R, Pandey S, Vecchiola C. Cloudbus toolkit for market oriented cloud computing. Proceeding of the first international conference on cloud computing (CloudCom 2009, Springer, Germany). Beijing, China: December 1–4, 2009.

[132] Calheiros R, Ranjan R, Beloglazov A, De Rose C, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Prac Exper 2011;41(1):23–50 Wiley Press, New York, NY, USA.

[133] Beloglazov A, Buyya R. Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. Concurrency Comput Prac Exper 2012;24(13):1397–420 Wiley Press, New York, NY, USA.

[134] Brown R, Masanet E, Nordman B, Tschudi B, Shehabi A, Stanley J, et al. Report to congress on server and data center energy efficiency: Public law 109–431. Berkeley, CA, USA: Lawrence Berkeley National Laboratory; 2008.

[135] Barroso LA Fan X. Power provisioning of warehouse-sized computer. Proceedings of the 34th annual symposium on computer architecture.San Diego, CA, USA: 2007. pp. 13–23.

[136] Cloud Storage Technical Working Group. Cloud data management interface (CDMI) v1.0. Storage Network Industry Association (SNIA); 2010. [Online Document]. Available at <www.snia.org/tech_activities/standards/curr_standards/cdmi/CDMI_SNIA_Architecture_v1.0.pdf>.

[137] Open Cloud Standard Incubator. Interoperable clouds: a white paper from the open cloud standards incubator; DSP-IS0101. Distributed Management Task Force (DMTF); November 2009. [Online Document] Available at <www.dmtf.org/sites/default/files/standards/documents/DSP-IS0101_1.0.0.pdf>.

[138] Open Cloud Standard Incubator. Architecture for managing clouds: a white paper from the open cloud standards incubator, DSP-IS0102. Distributed Management Task Force (DMTF); June 2010. [Online Document] Available at <www.dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf>.

[139] Open Cloud Standard Incubator. Use cases and interactions for managing clouds: a white paper from the open cloud standards incubator, DSP-IS0103. Distributed Management Task Force (DMTF); June 2010. [Online Document] Available at <www.dmtf.org/sites/default/files/standards/documents/DSP-IS0103_1.0.0.pdf>.

[140] Bowen JA. Legal issues in cloud computing. Cloud computing: principles and paradigms. New York, NY, USA: Wiley Press; 2011.

[141] Gramm-Leach-Bliley Financial Services Modernization Act (GLB Act), Title V of the Financial Services Modernization Act of 1999, Pub. L. No. 106-102, 113 Stat. 1338, U.S. Government, November 1999.

[142] Federal Trade Commission on Business Compliance with Safeguards Rule, U.S. Government, 2009. [Online Document] Available at www.ftc.gov/bcp/edu/pubs/business/idtheft/bus54.shtm.

[143] Identity theft red flags and address discrepancies under the fair and accurate credit transactions act of 2003; final rule, Federal Trade Commission, U.S. Government, November 2007. Available at www.ftc.gov/os/fedreg/2007/november/071109redflags.pdf.

[144] Health Insurance Technology for Economic and Clinical Health (HITECH) Act, Title III of Division A and Title IV of Division B of the American Recovery and Reinvestment Act (ARRA), Pub. L. 111-5, 123 Stat. 115, U.S. Government, February 2009.

[145] Health Insurance Portability and Accountability Act (HIPAA), Pub. L. No. 104-191, 110 Stat. 1936, U. S. Government, August 1996.

[146] Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism Act (USA PATRIOT Act), Pub. L. No. 107-56, 115 Stat. 272, U.S. Government, 2001.

[147] EU Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the Protection of Individuals with Regard to the Processing of Personal Data and on the Free Movement of Such Data, European Commission, October 1995. Available at <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:NOT>.

[148] Personal Data (Privacy) Ordinance, Office of the Privacy Commissioner for Personal Data, Hong Kong, 2009. Available at <www.pcpd.org.hk/english/ordinance/ordfull.html>.

[149] Personal Information Protection and Electronics Document Act, Minister of Justice, Canada, Feburary 2012. Available at <http://laws-lois.justice.gc.ca/PDF/P-8.6.pdf>.

[150] Wu L, Buyya R. Service level agreement (SLA) in utility computing systems. Performance and dependability in service computing: concepts, techniques and research directions. Hershey, PA, USA: IGI Global; 2011.

[151] Bouillet E, Mitra D, Ramakrishnan K. The structure and management of service level agreement in networks. IEEE J Sel Area Comm 2002;20(4):691–9.

[152] Dinesh V, Supporting service level agreements on IP networks. Proceedings of IEEE/IFIP network operations and management symposium, 92(2), New York, USA, 2004.

[153] Jin LJ, Machiraju VA. Technical Report HPL-2002-180 Analysis on service level agreement of web services. Software Technologies Laboratory, HP Laboratory; 2002.

[154] Ron S Aliko P. Service level agreements, Internet NG, Internet NG Project. 2001. [Online Document] Available at <http://ing.ctit.utwente.nl/WU2/>.

[155] Rochwerger B, Caceres J, Montero R, Breitgand D, Elmroth E, Galis A, et al. The RESERVOIR model and architecture for open federated cloud computing. IBM Syst J 2009;53(4):1–11.

[156] Mather T, Kumaraswamy S, Lathif S. Cloud security and privacy: an enterprise perspective on risks and compliance. O'Reilly Media Inc; 2009.

[157] Rittinghouse JW, Ransome JF. Cloud computing implementation, management, and security. CRC Press; 2010.

[158] Broberg J, Buyya R, Tari Z. MetaCDN: harnessing 'storage clouds' for high performance content delivery. J Netw Comp Appl 2009;32(5):1012–22 Elsevier, Amsterdam, The Netherlands.

[159] Buyya R, Pathan M, Vakali A, editors. Content delivery networks. Berlin, Germany: Springer; 2008.

[160] Pandey S, Voorsluys W, Niu S, Khandoker A, Buyya R. An autonomic cloud environment for hosting ECG data analysis services. Future Generation Comput Syst 2012;28(1):147–54 Elsevier Science, Amsterdam, The Netherlands.

[161] Jin C, Gubbi J, Buyya R, Palaniswami M, Jeeva: enterprise grid-enabled web portal for protein secondary structure prediction. Proceedings of the 16th international conference on advanced computing and communication (ADCOM 2008). Chennai, India: December 14–17, 2008.

[162] Vecchiola C, Abedini M, Kirley M, Chu X, Buyya R. Gene expression classification with a novel coevolutionary based learning classifier system on public clouds. Proceedings of the 2010 sixth IEEE international conference on e-Science workshops (IEEE CS Press, USA). Brisbane, Australia:

December 7, 2010. pp. 92–97.
[163] Raghavendra K, Akilan A, Ravi N, Kumar KP, Varadan G. Satellite data product generation using aneka cloud. Research demo at the 10th IEEE international symposium on cluster, Cloud, and Grid Computing (CCGrid 2010). Melbourne, Australia: 2010.
[164] Buyya R, Abramson D. The Nimrod/G grid resource broker for economic-based scheduling ISBN: 978-0470287682 In: Buyya R, Bubendorfer K, editors. Market oriented grid and utility computing. Hoboken, NJ, USA: Wiley Press; 2009.
[165] Buyya R, Venugopal S, Chu X, Nadiminti K. System and method for grid and cloud computing, Patent No: 8,230,070. United States Patent and Trademark Office; July 24, 2012. Available at <www.uspto.gov/web/patents/patog/week30/OG/html/1380-4/US08230070-20120724.html>.

索　引

索引中的页码为英文原书页码，与书中页边标注的页码一致。
备注：页码后的“f”、“t”和“b”分别表示“图”、“表”和“定理框”。

A

Access Control Policies（ACP，访问控制策略），324
ACID properties（ACID 属性），264-265
Advanced programmable interrupt controller（APIC，高级可编程中断控制器），106
Advanced Research Projects Agency Network（ARPANET，高级研究计划署网络），3
Allocator（分配器），82
All-Pairs model（All Pairs 模型），275
Amazon CloudWatch（亚马逊 CloudWatch），332
Amazon Direct Connect（亚马逊直接连接），315-316
Amazon Elastic Compute（EC2，亚马逊弹性计算），18, 125, 130, 148, 158-160, 315-316
  EC2 environment（EC2 环境），320
  EC2 instance（EC2 实例），317-319, 319t, 326t
Amazon Flexible Payment Service（FPS，亚马逊弹性支付服务），332
Amazon Machine Image（AMI，亚马逊机器镜像），316-317
Amazon Route（亚马逊路由），53, 315-316
Amazon Simple Email Service（SES，亚马逊简单邮件服务），315-316, 331-332
Amazon Simple Notification Service（SNS，亚马逊简单通知服务），315-316, 331
Amazon Simple Queue Service（SQS，亚马逊简单队列服务），315-316, 331
Amazon Simple Storage Service（S3，亚马逊简单存储服务），263, 315-316
  access control and security（访问控制与安全），324
  advanced feature（高级功能），325
  bucket（桶），323
  key concept（关键概念），321-325
  objects and metadata（对象与元数据），323-324
  resource naming（资源命名），322-323
  ways of addressing bucket（处理桶的方法），322-323
Amazon Virtual Private Cloud（VPC，亚马逊虚拟私有云），315-316, 329-330
Amazon Web Service（AWS，亚马逊 Web 服务），9, 24, 40-41, 125, 137, 315-332, 395
  advanced compute service（高级计算服务），320-321
  Amazon Elastic Block Store（EBS，亚马逊弹性数据块存储），325
  Amazon Elastic MapReduce（亚马逊弹性 MapReduce），321
  Amazon Relational Data Storage（RDS，亚马逊关系数据存储），326-327, 327t
  Amazon SimpleDB（亚马逊简单数据库），327-328, 329t
  AWS CloudFormation, 320
  AWS Elastic Beanstalk（AWS 弹性 Beanstalk），320-321
  CloudFront, 328-329, 330t
  Communication service（通信服务），329-332
  Compute service（计算服务），316-321
  ecosystem（生态系统），315-316, 317f
  ElastiCache（弹性缓存），325-326
  Identity Access Management(IAM) service（身份访问管理服务），330
  messaging service（消息服务），331-332
  Preconfigured EC2 AMI（预配置 EC2（AMI），326
  spot instance（即时实例），387-388
  storage service（存储服务），321-329
  structured storage solution（结构化存储方案），326-328

virtual networking（虚拟网络），329-331
Amazon.com, 68, 116-117
AMD V, 85
Aneka (Manjrasoft), 26, 119, 127-128, 214
accounting service（记账服务），151-152
allocation service（分配服务），152-153
AnekaApplication class（Aneka 应用类），226-230
Aneka.PSM.Console, 243
Aneka.PSM.Core, 243
Aneka.PSM.Workbench, 243
Aneka.Tasks, 227-228
Application Finished event（应用完成事件），233
application management（应用管理），146, 153-155, 162-168
application model（应用模型），162-165, 163f, 164t
AutoPlugin（自动插件），248-249
billing service（收费服务），151-152
built-in service, 148
capability（构建服务）(容量），144f
classes of service（服务类别），143-144
Configuration.ResubmitMode（重配置模型），233
controlling of task execution（任务执行控制），228-233
Design Explorer（设计研究），247
development and monitoring tool（开发和监控工具），247
dynamic resource provisioning（动态资源提供），149
elasticity and scaling（弹性和可扩展），145
execution of task-based programming（基于任务编程的执行），151
execution service（执行服务），154-155, 226-227
fabric service（组织服务），147-149
file channel controller（文件通道控制器），150-151
file channel handler（文件通道句柄），150-151
file copying（文件复制），240
FileData class（FileData 分类），237
FileData.Storage BucketId（FileData.Storage BucketId 性质），238
FileData.VirtualPath property（FileData.VirtualPath 性质），238
file deletion（文件删除），241
file dependencies management（文件依赖性管理），240f
file management（文件管理），233-239
foundation service（基础服务），150-153
framework（框架），143-146, 145f
GaussApp application（GaussApp 应用），233, 237f 439
Heartbeat Service（心跳服务），148
hybrid cloud deployment model（混合云部署模型），160-162, 161f
IDistribution Engine（IDistribution 引擎），248-249
IJobManager interface（IJobManager 接口），246
Indexing and categorizing resource（资源索引和分类），149
infrastructure（基础设施），155, 156f
interactions among execution thread and strategy（执行线程与策略间交互），249-250
ITask interface（ITask 接口），227-228, 229f, 230f
job object model（作业对象模型），244-246, 244f
legacy application execution（遗留应用执行），240
logical organization（逻辑组织），155-158, 157f
management kit（管理套件），146
ManualResetEvent（手动重置事件），233
MapReduce programming（MapReduce 编程），见 MapReduce programming of Aneka
MapReduce programming model（MapReduce 编程模型），155, 164
master node（主节点），156-157
Membership Catalogue（成员目录），149, 226-227
message handling（消息处理），165-167
multithreadingwith, 见 Multithreaded applications with Aneka
Offspring, 245-249, 248f
parameter-sweeping applications（参数化应用），243-247, 247f
parameter sweep programming model（参数化编程模型），155, 164
Platform Abstraction Layer(PAL) interface（平台抽象层接口），146-147
private cloud deployment mode（私有云部署模式），

158, 159f
profiling and monitoring service（概要分析和监控服务）, 148
programming model（编程模型）, 154-155
PSM Console（PSM 控制台）, 247
PSM object model（PSM 对象模型）, 245f
PSMCopyCommandInfo, 246
PSMDeleteCommandInfo, 246
PSMEnumParameterInfo, 246
PSMEnvironmentCommandInfo, 246
PSMExecuteCommandInfo, 246
PSMRandomParameterInfo, 246
PSMRangeParameterInfo, 246
PSMSingleParameterInfo, 246
PSMSubstituteCommandInfo, 246
PSMSystemParameterInfo, 246
public cloud deployment mode（公共云部署模式）, 158-162
QoS/SLA management and billing（QoS/SLA 管理和记账）, 146
relational database management system（RDBMS，关系型数据库管理系统）, 148
Reporting and Monitoring Service（报告和监控服务）, 148
resource management（资源管理）, 145-146, 149
resource pricing（资源定价）, 151-152
resource reservation service（资源预留服务）, 152-153
ResubmitMode.Manual, 233
runtime management（运行时管理）, 145
scheduling service（调度服务）, 153-154
service life cycle（服务生命周期）, 166f
service model（服务模型）, 165-167
service-oriented architecture（SOA，面向服务架构）, 144-146
SingleSubmission flag（Single Submission 标志）, 232
software development kit（SDK，软件开发套件）, 146, 162-167
StopExecution method（StopExecution 方法）, 232
Storage BucketId property（StorageBucketId 属性）, 238
storage node（存储节点）, 158
storage service（存储服务）, 150-151, 226-227
StrategyController（策略控制器）, 248-249
submission of task（任务提交）, 230-232, 232f
SubmitExecution method（SubmitExecution 方法）, 233
substitute operation（替代操作）, 241
task composition（任务组合）, 242
task library（任务库）, 240-242
Task Programming Model（任务编程模型）, 154, 164, 226-227
task-based programming（基于任务的编程）, 225-250
TaskScheduler service（TaskScheduler 服务）, 226-227
thread programming model（线程编程模型）, 154, 164
timed delay（时间延迟）, 241
tools（工具）, 167-168
user management（用户管理）, 146
Web services integration（Web 服务集成）, 242, 243f
worker node（工人节点）, 157-158
Workflow Engine（工作流引擎）, 248
Animoto, 9, 366-367, 367f
ApacheCouchDB, 264
ApacheHadoop, 25
AppEngines software development tkit（SDK，AppEngines 软件开发工具包）, 25
AppExchange, 123
Appistry CloudIQ Platform（Appistry CloudIQ 平台）, 119-121
Apple iCloud, 9
Applets（Java 小应用程序）, 72-73
Application binary interface（ABI，应用二进制接口）, 78-79
Application programming interface（API，应用编程接口）, 78-79, 115-116, 192t, 281f, 283f
Application server virtualization（应用服务器虚拟化）, 91
Application service provider（ASP，应用服务供应商）, 20, 122
Application-level virtualization（应用级虚拟化）, 88-89
emulation function（仿真功能）, 88-89
strategies implemented（策略实施）, 88-89

Apprenda SaaS Grid, 119
Architectural styles for distributed system（分布式系统架构模式）, 41-51
based on independent component（基于独立组件）, 47-48
batch sequential style（批处理方式）, 44
blackboard architectural style（黑板架构模式）, 43
call &return architecture（调用和返回架构）, 46-47
client/server（客户机/服务器）, 48-50
communicating processes（通信处理）, 47
components and connector（组件和连接器）, 42
data-centered architecture（数据中心架构）, 42-43
data-flow architecture（数据流架构）, 44
event system（事件系统）, 47-48
fat-client model（胖客户机模型）, 49
interpretation style（解释模式）, 45-46
layered style（分层模式）, 46-47
multitiered architecture（多层架构）, 49-50
object-oriented style（面向对象模型）, 46
peer-to-peer model（点对点模型）, 50-51, 51f
pipe-and-filter style（管道和过滤模型）, 44
repository architectural style（仓库架构模式）, 43
rule-based（基于规则）, 45
software（软件）, 42-48, 43t
system architectural style（系统架构模式）, 48-51
thin-client model（瘦客户端模型）, 48
three-tier（三层结构）, 50
top-down style（自上而下模式）, 46
two-tier（两层结构）, 49-50
virtual machine（虚拟机）, 44-46
Asymmetric multiprocessing（非对称多处理）, 171
Asynchronous JavaScript and XML（AJAX，异步JavaScript和XML）, 19, 68
Azure (Microsoft), 25, 119-121, 341-350
access control service（访问控制服务）, 346
AppFabric, 342, 345-347
architecture（架构）, 342f
Azure Cache（Azure缓存）, 346-347
Azure Drive（Azure驱动）, 345
blobs service（blob服务）, 344
compute instance（计算实例）, 344t
compute service（计算服务）, 343-344
core concept（核心概念）, 342-347
.NET support（.NET支持）, 343
platform appliance（平台设备）, 349
queue storage（队列存储）, 345
Service Bus（服务总线）, 346
SQL Azure, 347-349, 348f
storage service,（存储服务）344-345
table（表）, 345
virtual machine role（虚拟机角色）, 343-344, 347
Web role（Web角色）, 343
Windows Azure Connect（Windows Azure连接）, 347
Windows Azure Content Delivery Network（CDN，Windows Azure内容分发网络）, 347
Windows Azure Management Portal（Windows Azure管理门户）, 342
Windows Azure Traffic Manager（Windows Azure流量管理器）, 347
Worker role（工作者角色）, 343

B

Basic Combined Programming Language（BCPL，基本组合编程语言）, 87-88
Batch processing（批处理）, 16-17
Behavior-sensitive instruction（行为敏感指令）, 79
Berkeley Open Infrastructure for Network Computing (BOINC)（用于网络计算的伯克利开放基础设施）, 214-216
Big Data（大数据）, 259-260, 268-269
Bigtable (Google)（Bigtable（谷歌））, 266, 267f
Binary translation（二进制翻译）, 89, 97-99
advantage and disadvantage（优点和缺点）, 98
BioInformatics Research Network（BIRN，生物信息学研究网络）, 257-258
BitTorrent, 50, 325
Blackboard architectural style for distributed system（分布式系统的黑板架构模式）, 43
Blobs service（blob服务）, 344
Block blobs（blob块）, 344
Blogger（博客）, 19
Box.net, 124
Business and consumer applications of cloud computing（云计算的商业和消费应用）,

358-370
customer relationship management (CRM, 客户关系管理), 359-362
enterprise resource planning (ERP, 企业资源规划), 359-362
media application (多媒体应用), 366-369
online multiplayer gaming (多人在线游戏), 369-370
productivity application (效率型应用), 362-365
social networking application (社交网络应用), 365-366
Business Process Execution Language (BPEL, 业务流程执行语言), 224

C

CCassandra (Apache), 266-268
Chroot operation (Chroot 操作), 86-87
Chubby service (Chubby 服务), 266
Citrix, 96
Clipper project (Clipper 项目), 256
Cloud, defining a (云, 定义), 7-9, 135-136
Cloud computing (云计算), 3
application development (应用开发), 22-23
benefit (收益), 13-14
bird's-eye view of (鸟瞰图), 10f
building (构建), 22-26
business and consumer application (商业和消费应用), 358-370
categories of service (服务类别), 11-12
challenge (挑战), 14-15
characteristic (特征), 13-14
characterization of (特性), 8
concept and idea (概念和想法), 7f
confidentiality of data (数据机密性), 15
defined (明确的), 3, 7-8, 111
defining, cloud and end user (云定义和终端用户), 7-9, 14
enterprise application (企业应用), 22-23
historical development (遗留开发), 15-22
infrastructure and system development (基础设施和系统开发), 23-24
interoperability and standard (互操作性和标准), 136-137
IT infrastructure and software (IT 基础设施和软件), 13-14
legal issue (法律问题), 15
models for deploying and accessing (部署和访问模型), 10-11, 11f
on-demand and dynamic scaling (按需和动态可扩展), 23
organizational aspect (组织层面), 138-139
pay-per-use basis (按使用付费), 3
platforms and frameworks for (平台和框架), 24-26
principle of (原理), 4
reference model (参考模型), 11-13, 12f
representational state transfer (REST) Web service (表征状态转移 Web 服务), 23
resource-intensive application (资源密集型应用), 23
scalability and fault tolerance (可扩展性和容错), 14, 137
scientific application (科学应用), 23, 353-358
security, trust, and privacy issue (安全, 信任和隐私问题), 138
security issue (安全问题), 15
service orientation (面向服务)
service-provisioning model of computing utility (效用计算的服务提供模型), 4
utility-oriented approach (面向效用的方法), 8-9
vision of (前景, 设想), 5-7, 6f
Web application (Web 应用), 22-23
Cloud Computing Interoperability Forum (CCIF, 云计算互操作论坛), 136-137, 408
Cloud federation (云联盟), 390-422
airline and accommodation market segment, example (航空公司和酒店细分市场, 例子), 395
characterization and definition (特点和定义), 391-392
Cloud Audit Data Federation (CADF) working group (云审计数据联盟工作组), 402
Cloud Computing Reference Model (云计算参考模型), 398
Cloud Data Management Interface (CDMI, 云数据管理接口), 405-407
Cloud Management Working Group (CMWG, 云管理工作组), 402

Cloud Security Alliance（CSA，云安全联盟），407-408
at conceptual level（概念级），392-396
Distributed Management Task Force（DMTF，分布式管理任务组），401-403
functional requirement（功能需求），393-394
at infrastructural level（基础设施级），398-399
interoperability at SaaS layer（SaaS 层互操作），399
interoperable platform（互操作平台），403
interoperation and composition among vendor（厂商间的互操作和组合），398
legal issue（法律问题），411-417
at logical and operational level（逻辑和操作层），396-398
nonfunctional requirement（非功能需求），394-395
object model（对象模型），406-407
Open Cloud Computing Interface（OCCI，开放云计算接口），403-405
Core Model（核心模型），403-405, 404f
Open Cloud Manifesto（开放云声明），400-401
Open Cloud Standards Incubator（开放云标准孵化器），402
Open Virtualization Format（OVF，开放虚拟化论坛），401-402
QoS agreement（QoS 协商），395
reference stack（参考栈），393f
security arrangement（安全设置），409-411
SLA（服务等级协议），395, 397-398
standardization effort（标准化工作），399-409
storage reference model（存储参考模型），406f
technologies for（技术），417-422
vendor-independent format for packaging standard（服务商无关的封装标准），401-402
virtual machine instances among provider（服务商的虚拟机实例），398-399
CloudFront(Amazon), 328-329, 330t
Cloud infrastructure（云基础设施），112-114, 112f
physical infrastructure（物理基础设施），112-113
Cloud platform（云平台），316t
Amazon Web Service（AWS，亚马逊 Web 服务），315-332
Google AppEngine（谷歌 AppEngine），332-341
Microsoft Windows Azure（微软 Windows Azure），341-350
Cloud-based in-house solution（基于云的内部解决方案），10-11
Cloudbursting（云爆炸），129-130
Cloudbus toolkit（Cloudbus 工具），389, 390f, 391t
Cloudbus Workflow Management System（WfMS，Cloudbus 工作流管理系统），225
Cloud, economic of（云计算经济性），133-135
capital cost（建设成本），133-134
cost saving（成本节约），134
elimination of indirect cost（降低间接成本），135
IT infrastructure leasing（IT 基础设施租赁），134
operational cost（运营成本），134
pricing model（定价模型），135
Cloud, type of（云类型），124-133
community（社区），124, 131-133
hybrid or heterogeneous（混合或异构），124, 128-130
private（私有），124, 126-128
public（公共），124-125
Cloud service consumer（CSC，云服务消费者），13
Cloud service provider（CSP，云服务供应商），13
Cloud WorkGroup, 408
Cluster computing（集群计算），17
Clustered multiprocessing（集群多处理器），171
Common Information Model(CIM) protocol（通用信息模型协议），101-103
Common Language Infrastructure(CLI)（通用语言基础设施），87-88
Common Object Request Broker Architecture（CORBA，通用对象请求代理架构），52-53, 59
component（组件），59
Communication protocol（通信协议），40
Community cloud（社区云），124, 131-133
architecture（架构），131-132
benefit（收益），132-133
sector for（部分），132
Component Object Model(COM, DCOM, and COM1)（组件对象模型），52-53
Computing era（计算时代），29, 30f
commercialization phase（商业化阶段），29
commoditization phase（商品阶段），29

key elements of computing（计算的关键元素）, 29
research and development (R&D) phase（研究和开发阶段）, 29
Computing service（计算服务）, 3
classification（分类）, 114t
Infrastructure- and Hardware-as-a-Service(IaaS/HaaS) solution（基础设施即服务和硬件即服务解决方案）, 114-117
Platform-as-a-Service (PaaS) solution（平台即服务解决方案）, 117-121
Software-as-a-Service (SaaS) solution（软件即服务解决方案）, 121-124
Condor, 17, 214
Condor-G, 215
Container（容器）, 87, 296-299, 406-407
Context switch（线程切换）, 174
Control-sensitivem struction（控制敏感型指令）, 79
Cray's vector processing machine(Cray 向量处理机), 33
CrossOver（跨）, 89
Cryptography,（密码 7, 15）
Customer relationship management（CRM，客户关系管理）, 12-13, 359-362
Microsoft Dynamics CRM（微软动态 CRM）, 361
NetSuite, 361-362
Salesforce.com, 360-361

D

DDAGMan（Directed Acyclic Graph Manager，有向循环图管理器）, 225
Data cloud（数据云）, 260
Data grid（数据网格）, 256-259
Big Data（大数据）, 259-260
BioInformatics Research Network（BIRN，生物信息研究网络）, 257-258
domain（域）, 257
functionality（功能性）, 256-257
heterogeneity and security（异构性与安全性）, 257
International Virtual Observatory Alliance（IVOA，国家虚拟天文台联盟）, 259
Large Hadron Collider(LHC) grid（大强子碰撞机网格）, 257
log analysis（日志分析）, 259
reference scenario（参考方案）, 258f
storage and dataset management facilities（存储和数据集管理工具）, 257
Data storage-as-a-Service(DaaS，数据存储即服务), 405
Data center（数据中心）, 373-375, 381-384
Data-flow architecture for distributed system（分布式系统的数据流架构）, 44
Data-intensive computing（数据密集型计算）
All-Pairs model（All-Pairs 模型）, 275
Application domain（应用领域）, 253
challenge（挑战）, 254-255
characterization of（特征）, 254
cloud technology supporting（云技术支持）, 259-260
data grid（数据网格）, 256-259
data partitioning（数据分区）, 254-255
defined（定义的）, 253-260
distributed database（分布式数据库）, 260
DryadLINQ model（DryadLINQ 模型）, 276
high-performance distributed file system and storage cloud（高性能分布式文件系统和存储云）, 261-263
high-speed wide-area networking(高速宽带网络), 256
historical perspective（历史观点）, 255-260
MapReduce programming model（MapReduce 编程模型）, 269-275, 270f
Not Only SQL(NoSQL) system（NoSQL 系统）, 263-268
Platform for programming（编程平台）, 268-276
Research issue（研究问题）, 255f
in scientific computing（科学计算中）, 253-254
Sphere model（Sphere 模型）, 275
storage system（存储系统）, 260-268
variation and extension of MapReduce(MapReduce 演变和扩展), 273-275
DataSynapse, 119, 127-128
De facto processor（De facto 处理器）, 171-172
De.li.cious, 19
Desktop virtualization（桌面虚拟化）, 90, 92

DiNucci, Darcy, 19-20
Direct Client User Interface（DCUI，直接用户端接口），101-103
Dispatcher（分发器），82
Distributed component object model（DCOM/COM+，分布式组件对象模型），59-60, 65-66
  architecture（架构），59-60
  feature（特征），59-60
  runtime（运行时），59-60
  server（服务器），59-60
Distributed computing system（分布式计算系统），15-18, 23
  architectural style（架构模式），41-51
  characterization of（特征），15-16
  cluster computing（集群计算），17
  component of（组件），39-41
  defined（定义的），15-16, 39
  elements characterizing（类素特征），16, 39-54
  evolution（演变），16f
  example of（例子），30-31
  general concept and definition（通用概念和定义），39
  grid computing（网格计算），17
  hardware and operating system layer（硬件和操作系统层），40-41
  interprocess communication model（进程间通信模型），51-54
  layered view of（分层视图），40f
  mainframe computing（大型主机计算），16-17
  property（属性），16
  technology（技术），54-69
  vs parallel computing（对比并行计算），29-31
Distributed Management Task Force（DMTF，分布式管理任务组），401-403
  Cloud Standard Incubator（云标准孵化器），136-137
Distributed memory MIMD model（分布式内存MIMD模型），35-36, 35f
Distributed object framework（分布式对象框架），56-61, 57f
  client process（客户机进程），56
  client-based activation（客户端激活），58
  common interaction pattern（通用交互模式），56
  marshaling by reference（封送处理参考），57
  object activation and lifetime（对象活动和生命周期），57-61
  objects as first-class entity（对象作为第一类实体），57
  proxy and skeleton mechanism（代理和骨架机制），56-57
  server process（服务器进程），56-57
  server-based activation（基于服务器的激活），58
Distributed Parallel Storage System(DPSS)（分布式并行存储系统），256
Django, 340
Dot-com bubble burst（互联网泡沫），21-22
Dropbox, 362-363
DryadLINQ model（DryadLINQ 模型），276
Dynamic provisioning（动态提供），3, 130
Dynamo (Amazon), 264-265, 265f

E

Elastic Block Store（EBS(Amazon)，弹性数据块存储亚马逊），325
Elastic Compute Cloud（EC2，弹性计算云），24
Elastic Load Balancing（弹性负载均衡），315-316
ElastiCache(Amazon)（弹性缓存（亚马逊）），325-326
ElasticHosts（弹性主机），116-117
Elastra Cloud Server（Elastra 云服务），127-128, 130
Embarrassingly parallel application（高度并行应用），177-178, 216
Encoding.com, 369
Energy efficiency in cloud（云计算的能效），373-377
  architecture for（架构），375-377
  energy-aware dynamic resource allocation（能量感知的动态资源分配），376-377
  integrated allocation of resource（资源整合配置），377
  InterClouds（互联云），377
Engine Yard（Yard 引擎），118-121
Enterprise resource planning(ERP，企业资源计划），359-362
Eucalyptus（桉树），127
Execution mode in virtualization technique（虚拟化

技术执行模式），80
Execution virtualization（运行虚拟化），77-89
application-level virtualization（应用级虚拟化），88-89
hardware-level virtualization（硬件级虚拟化），81-87
machine reference model（机器参考模型），78-80
programming language-level virtualization（编程语言级虚拟化），87-88
External network virtualization（外部网络虚拟化），90

F

Facebook, 19, 365-366
Federal Trade Commission(FTC，贸易联盟委员会)，413-414
Flexiscale, 116-117
Flickr, 19, 68
Floating-point operation per second（FLOPS，浮点操作/秒），213
Force.com, 25-26, 118-119
Free Virtual Private Server（FreeVPS，Free虚拟专用服务器），87
FreeBSD Jails, 87
Frequency scaling（频率扩展），171-172
Full virtualization（完全虚拟化），85, 97-104

G

Geoscience application of cloud computing（云计算地理科学应用），358
geographic information system(GIS)（地理信息系统），358
satellite remote sensing（卫星远程感知），358, 359f
GigaSpaces DataGrid, 119
Global Inter-Cloud Technology Forum（GICTF，全球互联云论坛），408
Globally unique identifier（GUID，全局唯一标识），191-192
Globus Resource Allocation Manager（GRAM，全局资源分配器），214-215
Globus Toolkit（Globus工具），214-215
Gnutella, 50
Go, 25
GoGrid, 116-117, 148
Google AppEngine（谷歌AppEngine），9, 18, 25, 118-121, 125, 332-341
account management（记账管理），337
application deployment and management（应用部署和管理），340
application development and testing（应用开发和测试），339-340
application service（应用服务），336-338
architecture（架构），333f
billable quota（计费额），341
compute service（计算服务），338
cost model（成本模型），340-341
Cron Jobs service（Cron作业服务），338
DataStore, 335-336
free service（免费服务），340
Go runtime environment（Go运行时环境），334
image manipulation（图像操作），338
infrastructure（基础设施），333
Java support（Java支持），334, 339
life cycle of application（应用程序生命周期），338-340
mail and instant messaging（邮件和瞬时消息），337
MemCache service（缓存服务），337
Python support（Python支持），334, 339-340
runtime environment（运行时环境），334
sandboxing（沙箱），334
static file server（静态文件服务器），335
storage level（存储层），335-336
supported runtime（运行时支持），334
task queue handling（任务队列处理），338
UrlFetch service（UrlFetch服务），336
Google Documents（谷歌文档），19, 363
Google File System（GFS，谷歌文件系统），151, 262
Google MapReduce infrastructure（谷歌MapReduce基础设施），273f, 274
Google Maps（谷歌地图），19
Gramm-Leach-Bliley(GLB) Act（GLB法案），413-414
"Grand Challenge" problem（巨大挑战问题），21-22, 213

Graphical processing unit（GPU，图形处理单元），171-172
Green cloud computing（绿色节能云计算），374-375, 374f, 375f
  architecture for（架构），375-377
  Green Resource Allocator(绿色节能资源分配器)，376
Grep operation（检索目标行命令操作），270
Grid computing（网格计算），17-18
  diffusion of computing grid（计算网格的普及），17
Groovy（流行的），87-88

H

Hadoop(Apache), 274
Hadoop Distributed File System（HDFS，Hadoop 分布式文件系统），268, 274
Hadoop HBase, 268
Hadoop MapReduce, 274
Haizea, 130
Hardware-assisted virtualization（硬件辅助虚拟化），81-87, 92
Hardware virtualization（硬件虚拟化），18
Health Information Technology for Economic and Clinical Health (HITECH) Act（医疗信息技术促进经济和临床医疗法），413
Health Insurance Portability and Accountability Act（HIPAA，健康保险流通与责任法案），413
Heroku, 118-121
High-level virtual machine（高层虚拟机），88
High-performance computing（HPC，高性能计算），17, 213, 353
High-performance server（高性能服务器），18
High-speed wide-are anetworking（高速宽带网络），256
High-throughput computing（HTC，高吞吐量计算），213-214, 353
Hive, 274
Hosted hypervisor（托管虚拟机管理程序），98-99
Hosted virtual machine（托管虚拟机），82
  architecture（架构），100
Hybrid/heterogeneous cloud（混合云 / 异构云），10-11, 124, 128-130, 129f
  advantage（优点），129-130
Hybrid virtual machine（HVM，混合虚拟机），84, 84b
HyperText Transfer Protocol（HTTP，超文本传输协议），21, 96-97
  method（方法），68
Hypervisor（虚拟机管理程序），81-85
  reference architecture（参考架构），83f
Hyper-V (Microsoft), 85
  address manager（地址管理），106
  advanced programmable interrupt controller（APIC，高级可编程中断控制器），106
  advantage and disadvantage（优点和缺点），108
  architecture（架构），104-107, 105f
  child partition（子分区），104-107
  cloud computing and infrastructure management（云计算和基础设施管理），107-108
  enlightened I/O and synthetic device（启发式 I/O 和集成设备），106-107
  hypercalls interface（超级调用接口），106
  hypervisor（虚拟机管理程序），106
  memory service routine（MSR，内存服务例行程序），106
  parent partition（父分区），104-107, 106
  scheduler（调度器），106
  synthetic interrupt controller（SynIC，合成中断控制器），106
  Virtual Machine Worker Process（VMWP，虚拟机工作进程），107
  Virtual Service Client（VSC，虚拟服务客户端），107
  Virtual Service Provider(VSP，虚拟服务供应商)，107
  Virtualization Infrastructure Driver（VID，虚拟化设备驱动），107
  VMBus, 107
  Windows Server Core（Windows 服务内核），107-108

I

IBM General Parallel File System（GPFS，通用并行文件系统），262IBM
IBM Logical Partition（LPAR，IBM 逻辑分区），87
ICloud, 362-363
ICore

Virtual Account（虚拟账户），87
InfiniB and network（InfiniB 和网络），29-30
Information Technology Infrastructure Library (ITIL，信息技术基础设施库），409
Infrastructure-and Hardware-as-a-Service (IaaS/HaaS) solution（基础设施即服务和硬件即服务（IaaS/HaaS）解决方案），114-117, 115f
  hardware（硬件），114-115
  levels of service（服务级别），116-117
  monitoring component（监控组件），116
  pricing and billing component（定价和计费组件），116
  provisioning component（配置组件），116
  QoS/SLA management component（QoS/SLA 管理组件），116
  reservation component（预约组件），116
  software management infrastructure（软件管理基础设施），115-116
  VM pool manager（VM 池管理器），116
  VM repository component（仓库组件），116
Infrastructure -as-a-Service (Iaas) solution（基础设施即服务解决方案），11-13, 26, 40-41, 71, 113
Instruction Set Architecture(ISA)（指令集结构），78-79
Instruction-level parallelism（指令级并行），171-172
Intel VT, 85
InterCloud（互联云），377, 390-422
  Alternate Offers protocol（Alternate Otters 协议），422
  architecture（架构），421f
  characterization and definition（特征和定义），391-392
  CloudCoordinator（云协调器），420-422
  CloudExchange（云交易所），420
Interface Definition Language(IDL，接口定义语言)，59
InterGrid, 130
Internal network virtualization（内部网络虚拟化），90
International Virtual Observatory Alliance（IVOA，国际虚拟天文台联盟），259
Internet Inter-ORB Protocol（IIOP，互联网内部对象请求代理协议），59
Internet-centric way of computing（以互联网为中心的计算方法），7-8
Interpreter（解释器），82
Interprocess communication model,distributed computing system（进程间通信模型，分布式计算系统），51-54
  message-based communication model（基于消息的通信模型），52-54
  point-to-point communication model（点到点通信模型），53
  publish-and-subscribe message model（发布和订阅消息模型），53-54
  request-reply message model(请求回应消息模型)，54
Inverted index（反向索引），271
ISO/IEC 27001/27002standard（ISO/IEC 标准），409

J

Java, 25, 72-73, 112-113, 118-119
Java remote method invocation（RMI，Java 远程方法调用），52-53, 60, 65-66
Java Virtual Machine（JVM，Java 虚拟机），74-75, 87-88
JavaScript Standard Object Notation（JSON，Java 脚本标准对象定义），68, 264
Jitted, 87
Joyent Smart Platform, 116-119

K

Kaiser project（Kaiser 项目），256
Kazaa, 50
Kepler, 225
Kernel-based Virtual Machine（KVM，基于内核的虚拟机），85, 127
Kleinrock, Leonard, 3

L

Large Hadron Collider(LHC) grid（大规模强子对撞网格），257
Legal issues in cloud scenario（云计算中的法律问题），15, 411-417

in Aregntina（阿根廷），414
in Brazil（巴西），414
business and commerce-related issues（商业与商务相关问题），415-416
in Canada（加拿大），414
data access（数据访问），414
data breach（数据泄漏），413
data protection（数据保护），413-414
in European country（欧洲国家），15
European Union(EU) directive（欧盟），414
Gramm-Leach-Bliley(GLB) Act（GLB 法案），413-414
Health InformationTechnology for Economic and Clinical Health (HITECH)Act（医疗信息技术促进经济和临床医疗法案），413
Health Insurance Portability and Accountability Act（HIPAA，健康保险流通与责任法案），413
in HongKong（香港），414
implication（含义），417
intellectual property related issue（知识产权保护相关问题），414-415
jurisdictional and procedural-related issue（司法和程序相关问题），416-417
Personal Information Protection and Electronic Document Act（PIPEDA，个人信息保护和电子文件法案），414
privacy and security related issue（隐私和安全相关问题），413-415
U.S. legislation（美国立法），15
Libra reservation（秤量（计价）预留），152
Linear Regression（LR，线性回归），271
Linthicum, David, 136
Live migration（移居），91-92, 92f
Longjump, 118-119
Loose coupling（松耦合），19-20
Loosely coupled multiprocessor system（松耦合多处理器系统），35
Lustre filesystem（Lustre 文件系统），261-262

M

Mac OSX operating system（Mac OSX 操作系统），89
MAGIC project（MAGIC 项目），256
Mainframe computing（大型机计算），16-17, 21-22
Maintenance cost（维护成本），13-14
Many-task computing（MTC，多任务计算），214
MapReduce programming model（MapReduce 编程模型），25, 127-128, 151, 353
MapReduce programming of Aneka
Aneka MapReduce 编程
Aneka.Property and Aneka.PropertyGroup classes（Aneka. Property 类和 Aneka. PropertyGroup 类），303
Aneka.Util library, 303
API, 281f
application example（应用案例），293-308
component entry distribution（组件入口分配），310f
configuration file（配置文件），303
Configuration.Workspace directory（配置空间目录），284
distributed file system support（分布式文件系统支持），290-293
driver program（驱动程序），300-303, 307f
execution of task（任务执行），290
file format（文件格式），293f
formatted log message（格式化日志消息），299
infrastructure（基础设施），277f
input and output section（输入和输出部分），302-303
InvokeAndWait method（调用和等待方法），282-284
Log Parsing Mapper and Log Parsing Reducer（日志分析 Mapper 和 Reducer），300-302
map and reduce method（映射和归约方法），278
Mapper class design and implementation（Mapper 类设计和实现），300
MapReduce abstraction object model（MapReduce 抽象对象模型），279f
MapReduce Execution Service（MapReduce 执行服务），284-286, 292f
MapReduce Scheduling Service（MapReduce 调度服务），284-286, 291f
MapReduceApplication class（MapReduce Application 类），277-278
MapReduceScheduler class（MapReduce Scheduler

类），290
NextKey() and NextValue() method（NextKey() 和 NextValue() 方法），293
OnDone callback check（OnDone 回调检查），293
OnDone method（OnDone 方法），303
parameter（参数），279-281
parsing of log file（日志文件分析），296-300, 301f
programming abstraction（编程抽象化），276-284
reduce function API（reduce API 函数），283f
Reduce class design and implementation（Reduce 类设计与实现），300
Report Error method（错误报告方法），303
running application（运行应用），303-308
runtime support（运行时支持），276-277, 284-290
scheduling of job and task（作业和任务调度），286-290
SeqReader class（SeqReader 类），292-293, 296f
SeqWriter class（SeqWriter 类），292-293, 296f
StopExecution method（StopExecution 方法），282-284
storage implementation（存储实现），291
WordCounter job（单词计数作业），299f
Map-Reduce-Merge（映射 - 归约 - 融合），274
Market-based management of cloud（基于市场的云管理），377-389
application to PaaS and IaaS provider（PaaS 和 IaaS 供应商支持的应用），381-384
datacenter（数据中心），381-384
market-oriented cloud computing（MOCC，面向市场的云计算），378-384
Market-oriented cloud computing（MOCC，面向市场的云计算），378-379, 380f
AppSpot, 388-389
auctioneer（拍卖），381
bank（银行），381
for datacenter（数据中心），381-384
flexible pricing model（灵活的价格模型），387-388
framework for trading computing utility（交易计算基础设施的框架），385-387
global view of（全球视角），379-381
industrial implementation（工业实现），387-389
market directory（市场目录），381, 388-389
reference model（参考模型），379-384
scheduler taxonomy（调度器分类），386f
SLA resource allocator（SLA 资源分配器），382-383
SpotCloud, 388
technology and initiative supporting（技术和创新支持），384-389
virtual machine（VM，虚拟机），383-384
virtual marketplace（虚拟市场），388
MarshalByRefObject, 60-61
Marshaling by reference（由参考封送），57
Master/ kernel mode invirtualization technique（虚拟化技术中的主 / 内核模式），80
Media application of cloud computing（云计算媒体应用），366-369
Animoto, 366-367, 367f
Encoding.com, 369
Maya rendering with Aneka（用 Aneka 完 成 Maya 图形渲染），368-369
3D rendering on private cloud（在私有云上的 3D 渲染），368f
train design（火车设计），368-369
video encoding and transcoding（视频编码和解码），369
Memory management unit（MMU，内存管理单元），98-99
Message Passing Interface（MPI，消息传递接口），17, 218-222, 221f
Message-based communication model（基于消息的通信模型），52-53
distributed agent and active object（分布式代理和活动对象），53
distributed object model（分布式对象模型），52-53
Message-Passing Interface（MPI，消息传递接口），52
point-to-point communication model（点到点通信模型），53
publish-and-subscribe message model（发布和订阅模型），53-54
remote procedure call（RPC，远程过程调用），52
request-reply message model（请求和应答模型），54

MetaCDN, 423-425, 424f
Metadata（元数据）, 253
Microsoft Dynamics CRM（微软动态客户关系管理）, 361
MongoDB, 264
Multicore processor（多核处理器）, 172f
Multicore system（多核系统）, 172
Multiple-instruction, multiple-data (MIMD) system（多指令多数据系统）, 34-36
  architecture（架构）, 35f
  distributed memory MIM Dmodel（分布式内存 MIMD 模型）, 35-36, 35f
  shared memory MIMD model（共享内存 MIMD 模型）, 34, 35f
  Silicon Graphics machine（硅图形机）, 34
  Sun/IBM's SMP(Symmetric Multi-Processing)（Sun/IBM SMP（对称多处理器））, 34
Multiple-instruction, single-data(MISD) system（多指令单数据系统）, 33-34
  architecture（架构）, 34f
Multitasking（多任务）, 172-173
Multitenancy（多租户）, 121-122, 125
Multithreaded applications with Aneka（多 Aneka 线程应用）
  Aneka.Threading.AnekaThread class, 191
  Aneka.Threading.Thread class, 194
  application model（应用模型）, 195
  custom serialization（定制序列化）, 196-197
  domain decomposition（域分解）, 196-203
  functional decomposition（功能分解）, 203
  interface compatibility（接口兼容性）, 191-192
  mathematical function（数学函数）, 203, 209f
  MatrixProduct class（MatrixProduct 类）, 197
  Multi threaded matrix multiplication（多线程矩阵乘法）, 196-203
  ScalarProduct Class, 199f, 203f
  SerializationInfoclass, 197
  System.Threading.Threadclass, 191-192, 194
  thread creation and execution（线程创建和执行）, 197f
  thread lifecycle（线程生命周期）, 192-194, 193f
  thread priority（线程优先权）, 194
  Thread Programming Model（线程编程模型）, 190-191
  thread synchronization（线程同步）, 194
  type serialization（类型序列化）, 194-195
  vs .NET threadingAPI（和 .NET 线程 API）, 192t
Multithreading（多线程）, 172-173

N

Naïve Bayes(NB), 271
National Institute of Standards and Technologies（NIST，国家标准与技术研究院）, 8, 131, 408
Native virtual machine（本地虚拟机）, 81
.NET framework（.NET 框架）, 72-73, 87-88, 112-113
  serialization of type（类型序列化）, 194-195
.NET remoting（.NET 远程）, 52-53, 59-61, 65-66
.NET threading（.NET 线程）, 176
NetSuite, 123, 361-362
NetSuite Business Operating System（NS-BOS，NetSuite 商业操作系统）, 361-362
NetSuite Global CRM（NetSuite 全球 CRM）, 361
NetSuite Global Ecommerce（NetSuite 全球电子商务）, 361
NetSuite Global ERP（NetSuite 全球 ERP）, 361
Network Address Translation(NAT，网络地址转换), 90
Network Intrusion Detection System（NIDS，网络入侵检测系统）, 45
Network virtualization（网络虚拟化）, 90
Neural Network（NN，神经网络）, 271
Nimrod/G, 214-215
Node Resolver, 148
Nonprivileged instruction（非特权指令）, 79
Nonuniform memory acces（NUMA，不一致内存访问）, 171
Not Only SQL(NoSQL) system（NoSQL 系统）, 263-268

O

Object Request Broker（ORB，对象请求代理）, 59
Offspring, 225
Online multiplayer gaming（多人在线游戏）, 369-370
Open Cloud Computing Interface（OCCI，开放云计

算接口），403-405
Open Cloud Consortium（OCC，开放云联盟），136-137, 408
Open Virtualization Format（OVF，开放虚拟化格式），137, 398-399, 401-402
OpenNebula, 127, 130
OpenPEX, 127-128
OpenVZ, 87
Operating system-level virtualization（操作系统级虚拟化），86-87
Operating system(OS) developer（SystemISA，操作系统开发系统），78-79
Operational cost（运营成本），13-14
Oracle GridEngine, 215
Organization for Advancement of Structured Information Standard (OASIS)（高级结构化信息标准化组织），64

P

Page blob（页面块），344
Parallel computing（并行计算），29-30
  approach（方法），36
  computational requirement（计算需求），31
  cost vs speed（成本相对于速度），38, 38f
  data parallelism（数据并行），36
  element of（要素），31-39
  factor influencing parallel programming（影响并行编程的因素），31-32
  farmer-and-worker model（农场主和工作者模型），36
  guideline（准则），37-39
  hardware improvement（硬件改进），32-36
  level of parallelism（并行级别），36-37, 37f, 37t
  number of processor vs speed（处理器数量相对于速度），38, 38f
  parallel programming（并行编程），31-32
  process parallelism（过程并行），36
  sequential architecture（串行架构），31, 36
  technology of（技术），32
  vector processing（向量处理），32
Parallelism（并行）（见 Multithreaded applications with Aneka），171
Parallel program（并行程序），29-30，见 Parallel computing
Parallel（并行），85
Parallels Virtuozzo Container（并行 Virtuozzo 容器），87
Parallel Virtual Machine（PVM，并行虚拟机），17
Parameter sweep application（参数化应用），217-218
Paravirtualization technique（半虚拟化技术），85-86, 96-98
Parrot, 88
Partial virtualization（部分虚拟化），86
Pascal, 87-88
Pay-per-use basis, services on（即用即付服务），3-4, 8-10, 21, 126-127
PERL, 88
Peta-FLOPS, 213
Pig, 274
Pig Latin, 274
Pittsburgh Supercomputing Center（PSC，匹兹堡超级计算中心），256
Platform-as-a-Service(PaaS) solution（平台即服务解决方案），11-14, 26, 113, 117-121, 117f
  application management（应用管理），118
  automation（自动），120
  characteristic（特征），119-120
  service（服务），120
  core middleware（核心中间件），118
  level of abstraction（抽象级别），120
  implementation（实现），118-119, 119t
  Pure PaaS（纯 PaaS），118, 143.（见 Aneka (Manjrasoft) runtime framework），120
  vendor lock-in（供应商绑定），120-121
  Web-based interface（基于 Web 的接口），118
Point-to-point communication model（点到点通信模型），53
Portable Object Adapter（POA，可移植对象适配器），59
Portable Operating System Interface for UNIX（POSIX，用于 Unix 的可移植操作系统接口），101-103
  thread（线程），175-176
Pricing model（定价模型），135
  per-unit pricing（单位定价），135

subscription-based pricing（基于订阅定价）, 135
tiered pricing（分等级定价）, 135
Private/enterprise cloud（私有云/企业云）, 10-11, 124, 126-128, 128f
advantage（优势）, 126-127
architecture（架构）, 127
Privileged instruction（特权指令）, 79, 80f
Process virtual machine（流程虚拟机）, 18, 81, 88
Productivity application of cloud computing（云计算应用产品）, 362-365
cloud desktop（云桌面）, 363-365
Dropbox, 362-363
EyeOS, 363-365
Google Docs, 363
iCloud, 362-363
Xcerion XML Internet OS/3(XIOS/3), 363-365
Programming language-level virtualization（编程语言级虚拟化）, 87-88
Public cloud（公共云）, 10, 124-125
Publish-and-subscribe message model（发布和订阅消息模式）, 53-54
Pure PaaS（纯 PaaS）, 113, 118
Python, 25, 72-73, 87-88, 112-113, 118-119

R

Rackspace, 116-117
Rails-based Website（Rail 网站）, 118-119
Really Simple Syndication（RSS，真正简单聚合内容）, 19
Reference model of cloud computing（云计算参考模型）, 11-13, 12f
Register-based virtual machine（基于寄存器的虚拟机）, 88
Relay reservation（中继预留）, 152
Representational State Transfer (REST) system（具象状态传输协议）, 53, 66-68, 321-322
Web service（Web 服务）, 23
Request-reply message model（请求转发消息模型）, 54
Resources and Services Virtualization Without Barriers（RESERVOIR，资源和服务虚拟化无障碍）, 417-420
architecture（架构）, 419f
concept（概念）
of dynamic federation（动态联盟）, 417
general overview（一般概述）, 418f
separation between service provider and infrastructure provider（服务供应商与基础设施供应商的分工）, 418
service application component（服务应用组件）, 418-419
Service Manager（服务管理者）, 419-420
service manifest（服务清单）, 418-419
VEE Host (VEEH), 420
Virtual Execution Environment (VEE) Manager（虚拟执行环境管理器）, 420
RESTful API（RESTful 应用编程接口）, 115-116
RESTful Web service（RESTful Web 服务）, 68
Reverse Web-linkgraph（反向 Web 连接图）, 271
Rich Internet applications（RIA，丰富的互联网应用）, 19-20
RightNow, 123
RightScale, 18, 116-117
RMI registry（RMI 注册表）, 60-61
Ruby, 87-88, 118-119

S

SalesForce.com, 25-26, 123, 360-361, 360f
Scientific application of cloud computing（云计算科学应用）, 353-358
in biology（生物学）, 355-358
Cloud-CoXCS, 357f
data-intensive application（数据密集型应用）, 353
ECG monitoring（ECG 监控）, 353-354
gene expression data analysis for cancer diagnosis（癌症诊断的基因表达数据分析）, 357-358
geoscience（地理科学）, 358
healthcare（健康医疗）, 353-354
high-performance computing (HPC) application（高性能计算应用）, 353
high-throughput computing (HTC) application（高吞吐量计算应用）, 353
IaaS solution（IaaS 解决方案）, 353
Jeeva Portal, 356, 356f
online health monitoring system（在线医疗监控系

统），355f
in protein structure prediction（蛋白质结构预测），355-356
Sector Distributed File System（SDFS，Sector 分布式文件系统），275
Sector files ystem（Sector 文件系统），263
Security management in cloud scenario（云计算环境中的安全管理），409-411
centralized federation model（集中式联盟模型），411
claim-based model（基于声明的模型），411
customer responsibility（用户责任），410t
digital identity management（数字身份管理），411
element（要素），409
Liberty Alliance Identity Federation（自由联盟统一联合框架），411
OASIS Security Assertion Markup Language（OASIS 安全性断言标记语言），411
at three different level（IaaS, PaaS, and SaaS，三个不同层次（IaaS, PaaS 和 SaaS）），410t
WS-Federation（WS 联盟），411
Server consolidation（服务联合），72
Serverd process（服务过程），101
Service, defined（服务定义），20
Service choreography（服务编排），63
Service orchestration（服务协同），63
Service-level agreement（SLA，服务等级协议），8-9
Service-oriented architectures（SOA，面向服务架构），22, 26, 63-64
abstraction function（抽象功能），63-64
autonomy（自治），64
composability（组合能力），64
discoverability（发现能力），64
for enterprise application integration（EAI，企业应用整合），64
implementations of（实现），64
lack of state（状态丢失），64
loose coupling function（松耦合功能），63
platform（平台），63-64
reusability（可重用），64
service consumer role（服务消费者角色），63
service provider role（服务供应商角色），63
standardized service contract（标准化服务契约），63
Service-oriented computing（SOC，面向服务计算），20-21, 61-69
autonomous component（自治组件），62
boundary（范围），62
characteristic（特征），62
interface（类/接口），62
and cloud computing（云计算），68-69
quality of service（QoS，服务质量），20
semantic compatibility（语义兼容），62
service, defined（服务定义），61-63
service-oriented architecture（SOA，面向服务架构），63-64
Software-as-a-Service（软件即服务），20
structural compatibility（结构兼容），62
Web service（Web 服务），64-68
Service-Oriented Modeling Framework（SOMF，面向服务建模框架），64
Service-provisioning model of computing utility（计算基础设施的服务提供模型），3-4
Service in cloud computing（云计算服务），11-12
Infrastructure-as-a-Service（IaaS，基础设施即服务），11-12
Platform-as-a-Service（PaaS，平台即服务），11-12
Software-as-a-Service（SaaS，软件即服务），11-12
Shared memory MIMD model（共享内存 MIMD 类型），34, 35f
Simple Object Access Protocol（SOAP，简单对象访问协议），21, 53, 66
messages for Web service method invocation（Web 服务方法调用消息），67f
uses（使用），66-68
Simple Storage Service（S3，简单存储服务），24
Single-instruction, multiple-data(SIMD) system（单指令多数据系统），33
architecture（架构），33f
vector and matrix operation（矢量和矩阵运算），33
Single-instruction, single-data (SISD) system（单指令单数据系统），32
architecture（架构），32f
Small Computer System Interface（SCSI，小型计算机系统接口），76

Smalltalk, 87-88
Social networking application of cloud computing（云计算的社会网络应用）, 365-366
  Facebook, 365-366
Social networking Website（社交网络网站）, 19
Software architectural style for distributed system（分布式系统的软件架构模型）, 42-48, 43t
Software-as-a-Service (SaaS) solution（软件即服务解决方案）, 11-14, 26, 113, 121-124, 363
  application（应用）, 121-122
  application service provider（ASP，应用服务供应商）, 122
  CRM, ERP, and social networking application（CRM、ERP 和社会网络应用）, 123
  office automation application（办公自动化应用）, 124
  SaaS 2.0, 123
Software Information & Industry Association（SIIA，软件信息与工业协会）, 122
SolarisZones, 87
Sphere model（Sphere 模型）, 275
Sphere Process Engines(SPE), 275
SpotCloud, 388, 425, 426f
Stack-based virtual machine（基于栈的虚拟机）, 88
Standards Acceleration to Jump start Adoption of Cloud Computing（SAJACC，促进云计算应用的标准推进）, 408
Storage Network IndustryAssociation（SNIA，存储网络工业协会）, 405
Storage virtualization（存储虚拟化）, 89-90, 92
Stub-skeleton concept（存根－骨架概念）, 60
SubVirt, 95
Sun GridEngine (SGE), 214-215
Sun xVM, 85
Supercomputing 1991（SC91，超级计算 1991）, 256
Supervisor mode in virtualization technique（虚拟化技术的管理模式）, 80
Support Vector Machine（SVM，支持向量机）, 271
Symmetric multiprocessing（对称多处理器）, 171
Synthetic interrupt controller（SynIC，合成中断控制器）, 106
System Center Virtual Machine Manager（SCVMM，系统中央虚拟机管理器）, 108
System virtualization（系统虚拟化）, 81

T

Task, characterizing a（任务特征）, 212-213
Task computing（任务计算）, 211-216
  characterizing task（特征化任务）, 212-213
  frameworks for（框架）, 214-216
  high-performance computing(HPC，高性能计算), 213
  high-throughput computing（HTC，高吞吐量计算）, 213-214
  many-task computing（MTC，多任务计算）, 214
  middleware needs for（中间件要求）, 211-212
  models for（模型）, 212
  scheduling node（调度节点）, 214
  task submission（任务提交）, 211-212
  task-based application model（基于任务的应用模型）, 216-225
  worker node（工作节点）, 214-215
Task-based application model（基于任务的应用模型）（见 Aneka (Manjrasoft)）, 216-225
  for CPU-intensive mathematical computation（CPU 敏感型数学计算）, 221-222
  embarrassingly parallel application（高度并行应用）, 216
  message Passing Interface（MPI，消息传递接口）, 218-222, 221f
  parameter sweep application（参数化应用）, 217-218
  workflow application with task dependency（工作流应用的任务依赖关系）, 222-225
Technologies for cloud federation（云联盟技术）, 417-422
  InterCloud（互联云）, 420-422
  RESERVOIR, 417-420, 418f, 419f
Technology for distributed computing system（分布式计算系统技术）, 54-69
  distributed object framework（分布式对象框架）, 56-61, 57f
  remote procedure call（RPC，远程过程调用）, 54-56, 55f
  service-oriented computing（面向服务计算）, 61-

69
Tera-FLOPS（浮点），213
Term vector recap（词向量信息），271
TeraVision, 256
Terremark, 116-117
Thinking Machines'cm* 33
Third-party cloud service（第三方云服务），422-425
MetaCDN, 423-425, 424f
SpotCloud, 425, 426f
Threading, programming application with（线程编程应用）（见 Multithreaded application with Aneka），173-189
computation vs communication（计算相对于通信），189
context switch（线程切换），174
domain decomposition（域分解），177-180, 178f
embarrassingly parallel problem（高度并行问题），177-178
explicit（明确），173
functional decomposition（功能分解），180-188, 184f
implicit（隐含），173
Java and.NET, threads for(Java 和 .NET 线程)，176
main thread（主线程），174
mathematical function（数学函数），184, 188f
matrix multiplication（矩阵乘法），178-180, 179f
MatrixProduct Class（MatrixProduct 类），180, 183f
multithreaded program（多线程编程），179
operating system（操作系统），174
parallel application, developing（并行应用开发），177-189
Portable Operating System Interface for UNIX (POSIX) thread（用于 UNIX 线程的可移植操作系统接口），175-176
relation between thread and process（线程与进程之间的关系），174
ScalarProduct class（ScalarProduct 类），179-180, 181f
thread, defined（线程定义），174
Thrift, 366
Throughput computing（吞吐量计算），171
Translation look-aside buffer（TLB，转换后援缓冲器），98-99
Transmission Control Protocol/Internet Protocol（TCP/IP，传输控制协议 / 互联网协议），40
Twister, 274-275
Twitter, 19, 68
Type I hypervisor（I 型管理程序），81
Type II hypervisor（II 型管理程序），82

U

UCSD Pascal, 87-88
Unified Cloud Interface(UCI)（统一云计算接口），408
Universal Description Discovery and Integration（UDDI，通用描述发现和集成），65-66
UNIX-like operating system（UNIX 类操作系统），89
USA Patriot Act（美国爱国者法案），126
User Datagram Protocol（UDP，用户数据报协议），40
User ISA, 78-79
User mode in virtualization technique（虚拟化技术中的用户模式），80
Utility computing（效用计算），4
Utility cost（公共设施费用），13-14
Utility-oriented computing（面向效用的计算），21-22
Utility-oriented data center（面向效用的数据中心），111

V

VirtualBox（虚拟机），85
Virtual hardware（虚拟硬件），12
Virtualization（虚拟化），18, 24
advantage（优势），93
aggregation（集成），76
application server（应用服务器），91
characteristic（特征），73-77
and cloud computing（云计算），91-92
desktop（桌面），90
disadvantage（缺点），94-95
emulation function（仿真功能），76
example（例子），95-108
execution（执行），77-89
full（完全），97-104

hardware（硬件）, 18, 71-73
inefficiency and degraded use rexperience, issue of（低效和降低用户体验问题）, 94-95
isolation（隔离）, 76
malicious program, threat（恶意程序威胁）, 95
managed execution（执行管理）, 75-76
network（网络）, 90
performance degradation, problem of（性能降低问题）, 94
portability concept（可移植概念）, 77
process virtual machine（流程虚拟机）, 18
replication of runtime environment（运行时环境备份）, 18
security aspect（安全方面）, 74-75, 95
server consolidation（服务器稳定性）, 91-92
sharing function（共享功能）, 75
significance of（重要性）, 71-72
software program（软件编程）, 74
storage（存储）, 89-90
and network（网络）, 18
taxonomy（分类）, 77-91, 78f
of third-generation computer（第三代计算机）, 83b, 84b, 84f
virtual machine manager（虚拟机管理器）, 73
virtual private network（VPN，虚拟私有网络）, 73
virtualization reference model（虚拟化参考模型）, 73, 74f
Virtual LAN（VLAN，虚拟局域网）, 90
Virtual machine architectural style for distributed system（分布式系统的虚拟机架构模式）, 44-46
Virtual Machine Interface（VMI，虚拟机接口）, 104
Virtual machine migration（虚拟机迁移）, 76
Virtual machine role（Microsoft Azure，虚拟机角色（微软 Azure））, 25
Virtual Machine Worker Process（VMWP，虚拟机工人处理）, 107
Virtual machine-based programming language（基于虚拟机的编程语言）, 72-73
Virtual networking（虚拟网络）, 12
Virtual private network（VPN，虚拟私有网络）, 73
Virtual storage（虚拟存储）, 12
Virtualization Infrastructure Driver（VID，虚拟基础设施驱动）, 107
VMware, 85, 127, 376
VMware's technology, 97-104
dynamic binary translation, use of（动态二进制翻译）, 98-99
end-user (desktop)virtualization（终端用户（桌面）虚拟化）, 99-100
full virtualization（完全虚拟化）, 97-99
hardware-assisted virtualization, use of（硬件辅助虚拟化）, 98
infrastructure virtualization and cloud computing solution（基础设施虚拟化和云计算方案）, 103-104, 103f
reference model（参考模型）, 99f
server virtualization（服务器虚拟化）, 101-103
vFabric, 104
VMware ACE, 100
VMWare ESXiServer, 101-103, 102f
VMware Fusion, 99-100
VMware GSXServer, 101, 101f
VMware Player, 100
VMware ThinApp, 100
VMware Workstation, 99
x86 architecture, 98
VMware ThinApp, 89
VMware vCloud, 18

W

Web 2.0, 19-20
capillary diffusion of Internet（互联网普及）, 19
interactivity and flexibility（交互性和灵活性）, 19
interface（接口）, 23-24
loose coupling property（松耦合特性）, 19
Web role（Microsoft Azure，Web 角色）, 25
Web Service Definition Language（Web 服务定义语言）, 65-66
Web Service Description Language（Web 服务描述语言）, 21, 68
Web Services（WS，Web 服务）, 21, 64-68, 115-116
（见 Amazon Web Services(AWS)）, 65
concept（概念）, 65
directly supporting（直接支持）, 68

interoperability aspect（互操作能力方面）, 64-65
reference scenario（参考方案）, 65f
RESTful, 68
semantic for（语义）, 65
technology stack（技术栈）, 66f
WSDL（Web 服务描述语言）, 68
Wide Area Large Data Object (WALDO) system（广域大数据对象系统）, 256
Wikipedia, 19
Windows Application Binary Interface（WABI，Windows 应用二进制接口）, 89
Windows Azure, 18
Wine, 89
Winelib, 89
Worker role(Microsoft Azure)（工作者角色）, 25
Workflow application with task dependency（工作流应用的任务依赖）, 222-225
abstract model（抽象模型）, 224f
business-oriented computing workflow（面向商业的计算工作流）, 224
Cloudbus Workflow Management System（WfMS，云工作流管理系统）, 225
DAGMan（Directed Acyclic Graph Manager，有向无环图管理）, 225
directed acyclic graph（DAG，有向无环图）, 222
on distributed infrastructure（分布式基础设施）, 223
Kepler, 225
Montage workflow（蒙太奇工作流）, 222-223, 223f
Offspring, 225
scientific workflow（科学工作流）, 222
workflow, defined（工作流定义）, 222-223

X

X86 hardware（X86 系列硬件）, 85, 97
XaaS (Everything-as-a-Service)（一切即服务）, 8, 23-24, 68-69, 113-114, 122-123, 136 Xen, 85, 127
Xen CloudPlatform(XCP), 96-97, 96f Xen Hypervisor（云平台）, 76, 96-97Xen

Y

Yahoo!, 68
cloud infrastructure（云基础设施）, 25
YouTube, 19

Z

Zimory, 127-128, 130
ZimoryPools, 127-128
Zoho Office, 124

推荐阅读





云计算：概念、技术与架构

作者：Thomas Erl 等 译者：龚奕利 等 ISBN：978-7-111-46134-0 定价：69.00元

"我读过Thomas Erl写的每一本书，云计算这本书是他的又一部杰作，再次证明了Thomas Erl选择最复杂的主题和以一种符合逻辑而且易懂的方式提供关键的核心概念和技术信息的罕见能力。"

—— Melanie A. Allison，Integrated Consulting Services

在本书中，世界上最畅销的IT书籍作者之一Thomas Erl联合云计算专家和研究者，详细分析了业已证明的、成熟的云计算技术和实践，并将其组织成一系列定义准确的概念、模型、技术机制和技术架构，所有这些都是以工业为中心但是与厂商无关的。

本书理论与实践并重，重点放在主流云计算平台和解决方案的结构和基础上，除了以技术为中心的内容以外，还包括以商业为中心的模型和标准，以便读者对基于云的IT资源进行经济评估，把它们与传统企业内部的IT资源进行比较。此外，本书提供了一些用来计算与SLA相关的服务质量的模板和公式，还给出了大量的SaaS、PaaS和IaaS交付模型。

本书包括超过260幅图、29个架构模型和20种机制，是一本不可或缺的指导书，是对云计算技术的详细解读。

云计算与分布式系统：从并行处理到物联网

作者：Kai Hwang 等 译者：武永卫 等 ISBN：978-7-111-41065-2 定价：85.00元

"本书是一本全面而崭新的教材，内容覆盖高性能计算、分布式与云计算、虚拟化和网格计算。作者将应用与技术趋势相结合，揭示了计算的未来发展。无论是对在校学生还是经验丰富的实践者，本书都是一本优秀的读物。"

—— Thomas J. Hacker，普渡大学

本书是一本完整讲述云计算与分布式系统基本理论及其应用的教材。书中从现代分布式模型概述开始，介绍了并行、分布式与云计算系统的设计原理、系统体系结构和创新应用，并通过开源应用和商业应用例子，阐述了如何为科研、电子商务、社会网络和超级计算等创建高性能、可扩展的、可靠的系统。

推荐阅读



并行编程模式

作者：Timothy G. Mattson 等 译者：吕云翔 等 ISBN：978-7-111-48018-0 定价：78.00元



高性能科学与工程计算

作者：Georg Hager 等 译者：张云泉 等 ISBN：978-7-111-46662-9 定价：69.00元



多处理器编程的艺术（修订版）

作者：Maurice Herlihy 等 译者：金海 等 ISBN：978-7-111-41868-0 定价：69.00元



虚拟机：系统与进程的通用平台

作者：James E. Smith 等 译者：安虹 等 ISBN：978-7-111-25988-7 定价：78.00元

推荐阅读



作者：Abraham Silberschatz 著
中文版书名：数据库系统概念（第6版）
书号：978-7-111-37509-6
定价：55.00元
英文版书名：Database System Concepts, 6th Edition
书号：978-7-111-44089-1
定价：55.00元



作者：Jiawei Han 等著
中文版书名：数据挖掘：概念与技术（第3版）
书号：978-7-111-39140-1
定价：79.00元



作者：Ian H. Witten 等著
中文版书名：数据挖掘：实用机器学习工具与技术（第3版）
书号：978-7-111-45301-5
定价：79.00元



作者：Randal E. Bryant 等著
书名：深入理解计算机系统（第2版）
书号：978-7-111-32133-0
定价：99.00元



作者：David A. Patterson, John L. Hennessy
中文版书名：计算机组成与设计：硬件/软件接口（第4版）
书号：978-7-111-37555-3
定价：99.00元



作者：James F. Kurose 著
书名：计算机网络：自顶向下方法（第6版）
书号：978-7-111-45578-5
定价：79.00元



作者：Thomas H. Cormen 等著
书名：算法导论（第3版）
书号：978-7-111-40701-0
定价：128.00元



作者：Brian W. Kernighan 等著
书名：C程序设计语言（第2版·新版）
书号：978-7-111-12806-9
定价：30.00元



作者：Roger S. Pressman 著
书名：软件工程：实践者的研究方法（第7版）
书号：978-7-111-33581-6
定价：79.00元

华章科技
HZBOOKS | Science & Technology

