



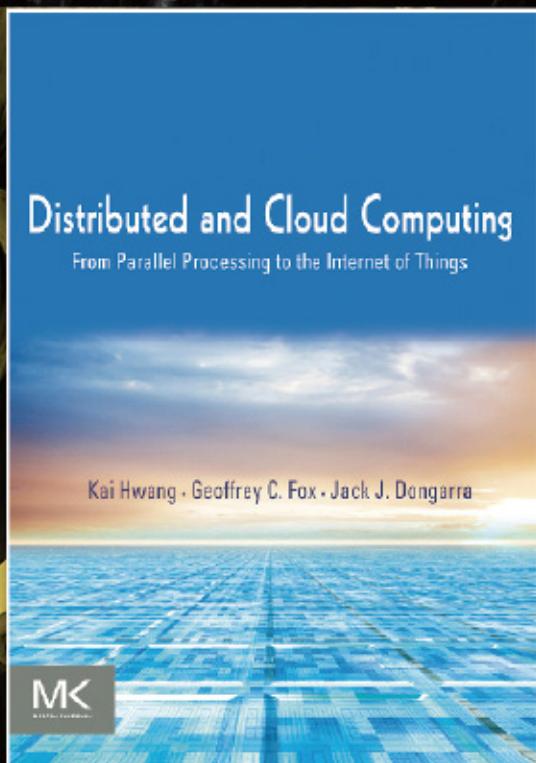
计 算 机 科 学 从 书

InfoQ 软件开发丛书

云计算与分布式系统 从并行处理到物联网

(美) Kai Hwang Geoffrey C. Fox Jack J. Dongarra 著
武永卫 秦中元 李振宇 钮艳 译
黄铠 审校

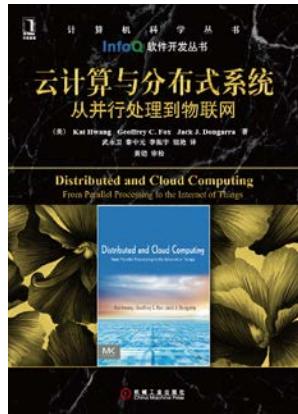
Distributed and Cloud Computing
From Parallel Processing to the Internet of Things



机械工业出版社
China Machine Press

免费在线版本

(非印刷免费在线版)



了解本书更多信息请登陆[豆瓣图书相关页面](#)

InfoQ中文站出品

InfoQ
软件开发

本书由InfoQ中文站免费发放。如果您从其他渠道获取本书，请注册InfoQ中文站以支持作者和出版商，并免费下载更多InfoQ软件开发系列图书。

本迷你书主页为

<http://www.infoq.com/cn/minibooks/distributed-and-cloud-computing>

文艺复兴以降，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；也正是这样的传统，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, Brian W. Kernighan, Dennis Ritchie, Jim Gray, Alfred 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襄助，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，有的还专程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，我们的联系方式如下：

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

中文版序 | Distributed and Cloud Computing, 1E

本书是《Distributed and Cloud Computing: From Parallel Processing to the Internet of Things》的中译本，其英文原版由美国的 Morgan Kaufmann 出版公司于 2011 年出版发行。机械工业出版社华章公司获得了本书的影印版授权，并于 2012 年在国内影印出版。本书适合作为高等院校计算机相关专业的教材，也适合作为专业技术人员的参考书籍。

我于 2007 年在美国南加州大学开始筹划此书。2008 年，我在中国科学院计算技术研究所开设了这门课，当时教材还在初期框架建构中。2009 年，我在北京大学做客座教授时完成了全部大纲，并在应聘清华大学 EMC 讲座教授时邀请了美国的 Geoffrey C. Fox 与 Jack J. Dongarra 教授加入这本书的撰写队伍。

本书中文版与英文版的版权完全是独立的，换句话说，中文版是在英文版的基础上修订、补充完成的，文责全部由本人负责。我邀请了国内四位一线学者参与中文版的翻译工作，现在简单介绍四位学者如下：

武永卫教授执教于清华大学计算机系。他是网格计算与云存储方面国内的重要学者，在国际同行中也很有影响。他负责了本书第 1 章、第 2 章和第 7 章的翻译工作。

秦中元博士是东南大学信息科学与工程学院副教授。他从事计算机系统与安全领域教学多年，曾于 2010 年访问美国南加州大学。他对本书第 9 章物联网方面内容的撰写有直接的贡献。他负责了本书第 5 章和第 6 章的翻译工作。

李振宇博士是中国科学院计算技术研究所的副研究员。他主要从事互联网通信和对等计算（P2P）方面的研究，与中国科学院计算技术研究所的谢高岗研究员合作发展未来互联网的系统与路由技术。他负责翻译本书第 8 章和第 9 章。

钮艳博士于 2012 年毕业于北京大学计算机系，她目前在工信部国家计算机网络应急技术中心担任工程师。她的博士论文是关于计算机虚拟化与云计算应用。她负责翻译本书第 3 章和第 4 章。

我目前担任清华大学分布式与云计算领域讲席教授组的客座首席讲座教授。本书中文版完成时，我恰好正在清华大学访问讲学，审校了中文版全书。对于四位同仁的精心翻译，我在此表示由衷感谢。在翻译的过程中，清华大学的赵勋与郭维超同学以及东南大学的宋云燕、郑勇鑫与杨中云同学都曾给予协助，在此对他们表示诚挚的感谢！

本书的中文版与影印版能够在国内出版，机械工业出版社华章公司做了大量工作，其中温莉芳、姚蕾和王春华给予了全力支持，在编辑、排版和制作索引方面做了很多细致工作。本人仅代表原著者与四位译者向以上所有工作人员表示感谢。

为了便于教师使用本书教学，我已将本书的英文版 PPT 和部分习题答案交给出版社，用书教师可以登录华章网站（www.hzbook.com）下载。

最后，希望读者通过阅读本书获益。读者如果在阅读过程中发现错误或不足，请发送至邮箱 hzjsj@hzbook.com。

黄铠

2012 年 12 月 12 日

于清华大学，北京

理查德·费曼（Richard Feynman）在他精彩的传记《别闹了，费曼先生》中叙述了 1944 年他如何在洛斯阿拉莫斯负责管理人类计算机执行曼哈顿计划要求的长而繁琐的计算。使用当时的机械计算器，最好的人类计算机每隔几秒钟仅可以实现一次加法或乘法。因此，费曼和他的团队研究了一些方法：将问题分解为较小的任务，这些小任务可以由不同的人同时执行（他们传递卡片，这些卡片带有人操作加法器、乘法器、整理器和分类器的中间结果）；每次以相同的计算复杂度运行乘法计算（使用不同颜色的卡片）；有效地检测和恢复错误（相关的卡片及其后代被移除，并重新开始计算）。

70 年后，计算机架构师面临类似的挑战，并且采用了类似的解决方案。虽然单个计算设备计算速度很快，但是物理约束对其速度仍然有限制。因此，如今的计算趋势是普适并行计算。单处理器包含流水线、并行指令、预测执行和多线程。本质上，从台式计算机到强大的超级计算机，每个计算机系统都包含多处理器。未来亿级超级计算机（每秒可以进行 10^{18} 次运算）的设计师告诉我们，这些计算机将需要支持 10^7 次并发运算。

并行性基本上是关于通信和协作的，并且伴随着技术的巨大变革，这两项活动过去 70 年中也在发生变化。光速没有比费曼时间更快，光纤中每纳秒 8 英寸或 20 厘米。我们很难想象在 50 毫秒内从洛杉矶向奥克兰发送一条消息。但是现今的数据传输速率已经发生了巨大变化，已从 1910 年（早期电报）每秒几个字符发展到 1960 年（ARPANET）每秒数千字符，到 2010 年光纤每秒超过 100 亿字符。

拟普适高速通信不仅允许将电话呼叫中心重新迁移至印度，而且允许将计算转移至集中式设备，以达到巨大的规模经济效益，并且允许收集和组织大量数据来支持世界范围的人做决策。因此，政府机关、研究实验室和需要模拟复杂现象的企业创建和操作庞大的超级计算机，它们都有成百上千的处理器。类似地，像谷歌、Facebook 和微软这类需要处理大量数据的公司操作众多大规模“云”数据中心，每个中心可能占地几万平方英尺，拥有几万或几十万台计算机。像费曼的洛斯阿拉莫斯团队一样，这些计算综合体把计算作为服务向多人提供，处理许多不同目的的计算。

大规模并行、超速通信和大规模集中化是现今人们做决策的基础。对于预报明天的天气、索引 Web、推荐电影、建议社会连接、预测股票市场的未来状态或者提供任意满足需要的信息产品，这些计算通常分布在上千的处理器上，并且有时依赖于从世界几百万个资源获取的数据。事实上，现代世界不能没有并行计算和分布式计算。

在这个普遍的并行和分布式世界中，对分布式计算的理解显然是计算机科学本科生教育的一个重要部分。（实际上，我觉得任何本科生都应该了解这些主题。）今天，大多数的复杂计算机系统不再是单个微处理器，而是完整的数据中心。大多数编写的计算机程序都是管理或运行于数据中心规模级系统上的。不理解这些系统和程序如何构造的计算机科学专业研究生可能无法高效地从事相关的工作。

黄铠、Fox 和 Dongarra 的这本教材出版非常及时。本书分为三部分，其内容逐步覆盖了支撑现代大规模并行计算机系统的硬件和软件体系结构；实现云计算与分布式计算的相关概念和技术；分布式计算的高级主题，包括网格、P2P 和物联网。在每一部分，本书都采用系统方法，不仅介绍概念，还阐



述代表技术和现实大型分布式计算部署。计算作为一门科学，是一门工程学科，并且这些真实系统的描述既有助于学生使用它们，也会帮助他们理解其他架构师如何操纵与大型分布式系统设计有关的各种约束。

本书还介绍了一些计算机科研人员目前面临的一些挑战。举两个例子，计算机已经成为电能的主要消耗者，在美国约占所有电能的 3%（在日本，2011 年的海啸过后，为节约电力，经常不得不关闭一些大规模超级计算机，而这些超级计算机可以帮助应对未来的自然灾害。）实际上，每年出售的处理器大约有 100 亿个，其中 98% 是用于嵌入式设备，而这些嵌入式设备日益需要通信来驱动，给物联网带来机遇和挑战。物联网将比现在的互联网更广大、更复杂、更有能力。

我希望本书的出版可以推动高校中分布式计算的教学——不仅作为一门选修课（现在通常这样设置），而是成为本科课程体系中的核心课。我还希望高校之外的其他人通过这本书来了解分布式计算，更广泛地了解目前处于前沿和尖端的计算技术：有时混乱；通常复杂；但更重要的是，令人兴奋不已。

Ian Foster

于怀俄明州杰克逊山洞

2011 年 8 月

关于本书

经过 30 年的发展，并行处理和分布式计算在计算机科学和信息技术中方兴未艾。许多高校现在已经开设相关课程。教师和学生一直在寻找一本可以全面涵盖计算理论和信息技术（包括设计、编程和分布式系统应用）的教材。本书正是为了满足这一需求而设计，而且本书还可以作为相关领域专业技术人员的参考书。

本书介绍了硬件和软件、系统体系结构、新的编程范式，以及强调速度性能和节能的生态系统方面的最新进展。这些最新发展说明了如何创建高性能集群、可扩展网络、自动数据中心和高吞吐量云/网格系统。我们还介绍了云编程以及如何将分布式系统和云系统应用于创新的互联网应用中。本书的目的是将传统的多处理器和多计算机集群转换为 Web 规模网格、云以及在未来互联网中泛在使用的对等（P2P）网络，包括近年来快速发展的大型社会网络和物联网。

本书主要内容

我们已经在单独的一卷中介绍了许多里程碑式的发展。我们呈现的成果不仅来自于我们自己的研究团队，还来自于美国、中国和澳大利亚的主要研究组织。总的来说，本书总结了近年来从并行处理到分布式计算和未来互联网的进展。

本书从现代分布式模型概述开始，揭示并行、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。本书试图将并行处理技术与基于网络的分布式系统结合。书中通过开源和商业厂商的具体例子，重点介绍了用于研究、电子商务、社会网络、超级计算等应用的可扩展物理系统、虚拟化数据中心和云系统。

全书共 9 章内容，分为三部分：第一部分覆盖系统模型和关键技术，包括集群化和虚拟化。第二部分介绍数据中心设计、云计算平台、面向服务的体系结构、分布式编程范式和软件支持。第三部分研究计算/数据网格、对等网络、普适云、物联网和社会网络。

本书有 6 章内容涉及云计算方面的相关材料，分别是第 1 章、第 3 ~ 6 章和第 9 章。书中描述的云系统包括公有云：谷歌应用引擎、亚马逊 Web 服务、Facebook、SalesForce. com 等。这些云系统在升级 Web 服务和互联网应用方面发挥着越来越重要的作用。计算机架构师、软件工程师和系统设计师可能想要利用云技术来建造未来计算机和基于互联网的系统。

本书特点

- 覆盖现代分布式计算技术，包括计算机集群、虚拟化、面向服务的体系结构、大规模并行处理器、对等系统、云计算、社会网络和物联网。
- 强调开发并行、分布式和云计算系统的普适性、灵活性、有效性、可扩展性、可用性和可编程性。

- 硬件、网络和系统体系结构方面的最新进展：
 - 多核 CPU 和众核 GPU (Intel、Nvidia、AMD)。
 - 虚拟机和虚拟集群 (CoD、Violin、Amazon VPC)。
 - Top500 体系结构 (Tianhe-1A、Jaguar、Roadrunner 等)。
 - 谷歌应用引擎、亚马逊 AWS、微软 Azure、IBM 蓝云。
 - TeraGrid、DataGrid、ChinaGrid、BOINC、Grid' 5000 和 FutureGrid。
 - Chord、Napster、BitTorrent、KaZaA、PPLive、JXTA 和 .NET。
 - RFID、传感器网络、GPS、CPS 和物联网。
 - Facebook、Force. Com、Twitter、SGI Cylone、Nebula 和 GoGrid。
- 范式、编程、软件和生态系统方面新的改进：
 - MapReduce、Dryad、Hadoop、MPI、Twister、BigTable、DISC 等。
 - 云服务和信任模型 (SaaS、IaaS、PaaS 和 PowerTrust)。
 - 编程语言和协议 (Python、SOAP、UDDI、Pig Latin)。
 - 虚拟化软件 (Xen、KVM、VMware ESX 等)。
 - 云操作系统和混搭系统 (Eucalyptus、Nimbus、OpenNebula、vSphere/4 等)。
 - 面向服务的体系结构 (REST、WS、Web 2.0、OGSA 等)。
 - 分布式操作系统 (DCE、Amoeba 和 MOSIX)。
 - 中间件和软件库 (LSF、Globus、Hadoop、Aneka)。
- 书中含有 100 多个例题，并且每章末都有习题和进一步阅读建议。
- 涵盖多个来自主流分布式计算提供商 (亚马逊、谷歌、微软、IBM、惠普、Sun、Silicon Graphics、Rackspace、SalesForce. com、netSuite 和 Enomaly 等) 的案例研究。

读者对象和阅读建议

本书适合作为分布式系统或分布式计算课程的教材，同时也适合专业系统设计师和工程师作为了解最新分布式系统技术（包括集群、网格、云和物联网）的参考书。本书均衡覆盖了这些主题，并洞察了物联网和 IT 演变的未来。

这 9 章内容适合作为高年级本科生和低年级研究生一学期课程（45 小时讲义）使用。对三学期的系统课程，本书的第 1~4 章、第 6 章和第 9 章适合 10 周课程（30 小时讲义）。除了解答习题之外，我们还建议学生在可用集群、网格、P2P 和云平台上做一些并行和分布式编程的实验。

受邀的贡献者

本书是三位主要作者花费 4 年的时间（2007—2011）共同计划、写作、编辑和校对完成的。在这期间，我邀请并得到了如下科学家、研究者、教师和博士生的帮助和技术协助，他们来自美国、中国和澳大利亚的知名大学。

下面列出了受邀对本书做出贡献的人。各章的原创作者、做出贡献的人员和编校都分别在每章结尾进行了说明。我们感谢他们在反复书写和校订过程中的敬业工作和宝贵贡献。匿名审稿人的建议对最终内容的改进也是非常有帮助的。

Albert Zomaya、Nikzad Rivandi、Young-Choon Lee、Ali Boloori、Reza Moraveji、Javid Taheri 和 Chen Wang，悉尼大学（澳大利亚）

Rajkumar Buyya, 墨尔本大学 (澳大利亚)

武永卫、郑纬民和陈康, 清华大学 (中国)

李振宇、孙凝辉、徐志伟和谢高岗, 中国科学院

喻之斌、廖小飞和金海, 华中科技大学 (中国)

Judy Qiu、Shrideep Pallickara、Marlon Pierce、Suresh Marru、Gregor Laszewski、Javier Diaz、Archit Kulshrestha 和 Andrew J. Younge, 印第安纳大学 (美国)

Michael McLennan、George Adams III 和 Gerhard Klimeck, 普度大学 (美国)

Zhongyuan Qin (秦中元)、Kaikun Dong、Vikram Dixit、Xiaosong Lou (楼肖松)、Sameer Kulkarni、Ken Wu、Zhou Zhao (赵洲) 和 Lizhong Chen (陈理中), 南加州大学 (美国)

Renato Figueiredo, 佛罗里达大学 (美国)

Michael Wilde, 芝加哥大学 (美国)

版权许可与致谢

涉及版权保护的插图的许可分别在图注中进行了公开致谢。特别地, 我们还要感谢 Bill Dally、John Hopcroft、Mendel Roseblum、Dennis Gannon、Jon Kleinberg、Rajkumar Buyya、Albert Zomaya、Randal Bryant、Kang Chen (陈康) 和 Judy Qiu, 他们慷慨地允许我们使用他们的幻灯片、原始图片和书中展示的例子。我们还要感谢 Ian Foster, 他为本书撰写了序, 向读者介绍本书。本书由 Morgan Kaufmann 出版社的 Todd Green 发起, Robyn Day 编辑, 并由 diacriTech 的 Dennis Troutman 负责生产, 在此对他们表示感谢!

没有以上诸位的共同努力, 本书是无法完成的。我希望读者能喜欢本书, 并向我们反馈书中的遗漏和错误, 以便新版修正, 进一步改进。

Kai Hwang、Geoffrey C. Fox 和 Jack J. Dongarra

目 录 |

Distributed and Cloud Computing, 1E

出版者的话

中文版序

序

前言

第一部分 系统建模、集群化和虚拟化**第1章 分布式系统模型和关键技术**

1.1 互联网之上的可扩展计算	2
1.1.1 互联网计算的时代	2
1.1.2 可扩展性计算趋势和新的范式	5
1.1.3 物联网和 CPS	7
1.2 基于网络的系统技术	8
1.2.1 多核 CPU 和多线程技术	9
1.2.2 大规模和超大规模 GPU 计算	11
1.2.3 内存、外部存储和广域网	13
1.2.4 虚拟机和虚拟化中间件	15
1.2.5 云计算的数据中心虚拟化	17
1.3 分布式和云计算系统模型	18
1.3.1 协同计算机集群	19
1.3.2 网格计算的基础设施	20
1.3.3 对等网络家族	22
1.3.4 互联网上的云计算	23
1.4 分布式系统和云计算软件环境	25
1.4.1 面向服务的体系结构 (SOA)	25
1.4.2 分布式操作系统趋势	28

1.4.3 并行和分布式编程模型	29
1.5 性能、安全和节能	31
1.5.1 性能度量和可扩展性分析	31
1.5.2 容错和系统可用性	33
1.5.3 网络威胁与数据完整性	34
1.5.4 分布式计算中的节能	35
1.6 参考文献和习题	37

第2章 可扩展并行计算集群

2.1 大规模并行集群	44
2.1.1 集群发展趋势	44
2.1.2 计算机集群的设计宗旨	45
2.1.3 基础集群设计问题	46
2.1.4 Top500 超级计算机分析	48
2.2 计算机集群和 MPP 体系结构	51
2.2.1 集群组织和资源共享	51
2.2.2 节点结构和 MPP 封装	52
2.2.3 集群系统互连	53
2.2.4 硬件、软件和中间件支持	56
2.2.5 大规模并行 GPU 集群	56
2.3 计算机集群的设计原则	59
2.3.1 单系统镜像特征	59
2.3.2 冗余高可用性	64
2.3.3 容错集群配置	67
2.3.4 检查点和恢复技术	69
2.4 集群作业和资源管理	71
2.4.1 集群作业调度方法	71
2.4.2 集群作业管理系统	73
2.4.3 集群计算的负载共享设备 (LSF)	74
2.4.4 MOSIX: Linux 集群和云的	

操作系统	75
2.5 顶尖超级计算机系统的个案研究	77
2.5.1 Tianhe-1A: 2010 年的世界最快超级计算机	77
2.5.2 Gray XT5 Jaguar: 2009 年的领先超级计算机	80
2.5.3 IBM Roadrunner: 2008 年的领先超级计算机	82
2.6 参考文献和习题	83

第3章 虚拟机和集群与数据中心

虚拟化	88
3.1 虚拟化的实现层次	88
3.1.1 虚拟化实现的层次	88
3.1.2 VMM 的设计需求和提供商	91
3.1.3 操作系统级的虚拟化支持	91
3.1.4 虚拟化的中间件支持	94
3.2 虚拟化的结构/工具与机制	95
3.2.1 hypervisor 与 Xen 体系结构	95
3.2.2 全虚拟化的二进制翻译	96
3.2.3 编译器支持的半虚拟化技术	97
3.3 CPU、内存和 I/O 设备的虚拟化	99
3.3.1 虚拟化的硬件支持	99
3.3.2 CPU 虚拟化	100
3.3.3 内存虚拟化	101
3.3.4 I/O 虚拟化	103
3.3.5 多核处理器的虚拟化	105
3.4 虚拟集群和资源管理	107
3.4.1 物理集群与虚拟集群	107
3.4.2 在线迁移虚拟机的步骤与性能影响	109
3.4.3 内存、文件与网络资源的迁移	111

3.4.4 虚拟集群的动态部署	114
3.5 数据中心的自动化与虚拟化	116
3.5.1 数据中心服务器合并	117
3.5.2 虚拟存储管理	118
3.5.3 虚拟化数据中心的云操作系统	119
3.5.4 虚拟化数据中心的可信管理	121
3.6 参考文献与习题	123

第二部分 云平台、面向服务的体系结构和云编程

第4章 构建在虚拟化数据中心上的云平台体系结构	132
4.1 云计算和服务模型	132
4.1.1 公有云、私有云和混合云	132
4.1.2 云生态系统和关键技术	135
4.1.3 基础设施即服务 (IaaS)	138
4.1.4 平台即服务 (PaaS) 和软件即服务 (SaaS)	140
4.2 数据中心设计与互连网络	142
4.2.1 仓库规模的数据中心设计	142
4.2.2 数据中心互连网络	144
4.2.3 运送集装箱的模块化数据中心	145
4.2.4 模块化数据中心的互连	146
4.2.5 数据中心管理问题	147
4.3 计算与存储云的体系结构设计	148
4.3.1 通用的云体系结构设计	148
4.3.2 层次化的云体系结构开发	150
4.3.3 虚拟化支持和灾难恢复	152
4.3.4 体系结构设计挑战	155
4.4 公有云平台: GAE、AWS 和 Azure	157
4.4.1 公有云及其服务选项	157
4.4.2 谷歌应用引擎 (GAE)	158

4.4.3 亚马逊的 Web 服务 (AWS) ...	160	5.4.3 元数据目录	215
4.4.4 微软的 Windows Azure	162	5.4.4 语义 Web 和网格	216
4.5 云间的资源管理	162	5.4.5 作业执行环境和监控	218
4.5.1 扩展的云计算服务	163	5.5 面向服务的体系结构中的	
4.5.2 资源配置和平台部署	164	工作流	219
4.5.3 虚拟机创建和管理	168	5.5.1 工作流的基本概念	219
4.5.4 云资源的全球交易	171	5.5.2 工作流标准	221
4.6 云安全与信任管理	173	5.5.3 工作流体系结构和规范	222
4.6.1 云安全的防御策略	173	5.5.4 工作流运行引擎	223
4.6.2 分布式入侵/异常检测	176	5.5.5 脚本工作流系统 Swift	224
4.6.3 数据和软件保护技术	177	5.6 参考文献与习题	226
4.6.4 数据中心的信誉指导保护 ...	179		
4.7 参考文献与习题	181	第6章 云编程和软件环境	234
第5章 面向服务的分布式体系 结构	188	6.1 云和网格平台的特性	234
5.1 服务和面向服务的体系结构	188	6.1.1 云的功能和平台的特性	234
5.1.1 REST 和系统的系统	189	6.1.2 网格和云的公共传统特性	234
5.1.2 服务和 Web 服务	191	6.1.3 数据特性和数据库	237
5.1.3 企业多层体系结构	195	6.1.4 编程和运行时支持	238
5.1.4 网格服务和 OGSA	196	6.2 并行和分布式编程范式	239
5.1.5 其他的面向服务的体系结构和 系统	199	6.2.1 并行计算和编程范式	240
5.2 面向消息的中间件	201	6.2.2 MapReduce、Twister 和 迭代 MapReduce	241
5.2.1 企业总线	201	6.2.3 来自 Apache 的 Hadoop 软件库	248
5.2.2 发布 - 订阅模型和通知	202	6.2.4 微软的 Dryad 和 DryadLINQ ...	251
5.2.3 队列和消息传递系统	202	6.2.5 Sawzall 和 Pig Latin 高级 语言	255
5.2.4 云或网格中间件应用	202	6.2.6 并行和分布式系统的映射 应用	258
5.3 门户和科学网关	204	6.3 GAE 的编程支持	259
5.3.1 科学网关样例	205	6.3.1 GAE 编程	259
5.3.2 科学协作的 HUBzero 平台 ...	206	6.3.2 谷歌文件系统 (GFS)	261
5.3.3 开放网关计算环境 (OGCE)	209	6.3.3 BigTable——谷歌的 NOSQL 系统	263
5.4 发现、注册表、元数据和 数据库	212	6.3.4 Chubby——谷歌的分布式锁 服务	265
5.4.1 UDDI 和服务注册表	212	6.4 亚马逊 AWS 与微软 Azure 中的	
5.4.2 数据库和订阅 - 发布	214		

编程	266
6.4.1 亚马逊 EC2 上的编程	266
6.4.2 亚马逊简单存储服务 (S3) ...	267
6.4.3 亚马逊弹性数据块存储服务 (EBS) 和 SimpleDB	269
6.4.4 微软 Azure 编程支持	269
6.5 新兴云软件环境	271
6.5.1 开源的 Eucalyptus 和 Nimbus	271
6.5.2 OpenNebula、Sector/Sphere 和 Open Stack	273
6.5.3 Manjrasoft Aneka 云和 工具机	276
6.6 参考文献与习题	280

第三部分 网格、P2P 和未来互联网

第 7 章 网格计算系统和资源 管理.....	292
7.1 网络体系结构和服务建模	292
7.1.1 网格历史与服务类别	292
7.1.2 CPU 清除和虚拟超级 计算机	294
7.1.3 开放网格服务体系结构 (OGSA)	296
7.1.4 数据密集型网格服务模型 ...	298
7.2 网格项目和网格系统创建	300
7.2.1 国家网格和国际项目	300
7.2.2 美国的 NSF TeraGrid	302
7.2.3 欧盟的 DataGrid	303
7.2.4 ChinaGrid 设计经验	305
7.3 网格资源管理和资源中介	305
7.3.1 资源管理和作业调度	305
7.3.2 CGSP 的网格资源监控.....	307
7.3.3 服务记账和经济模型	308
7.3.4 Gridbus 的资源中介	309
7.4 网格计算的软件与中间件	311

7.4.1 开源网格中间件包	311
7.4.2 Globus Toolkit 体系结构 (GT4)	313
7.4.3 集装器和资源/数据管理	316
7.4.4 ChinaGrid 支持平台 (CGSP)	318
7.5 网格应用趋势和安全措施	320
7.5.1 网格应用技术融合	320
7.5.2 网格负载与性能预测	322
7.5.3 网格安全执行的信任模型 ...	324
7.5.4 认证与授权方法	326
7.5.5 网格安全基础设施 (GSI) ...	327
7.6 参考文献与习题	330

第 8 章 对等计算和覆盖网络

336

8.1 P2P 计算系统	336
8.1.1 P2P 计算系统的基本概念 ...	337
8.1.2 P2P 计算面临的基础挑战 ...	340
8.1.3 P2P 网络系统分类	344
8.2 P2P 覆盖网络及其性质	345
8.2.1 无结构 P2P 覆盖网络	345
8.2.2 分布式哈希表 (DHT)	348
8.2.3 结构化 P2P 覆盖网络	349
8.2.4 混合式覆盖网络	352
8.3 路由、邻近性和容错	355
8.3.1 P2P 覆盖网络的路由	355
8.3.2 P2P 覆盖网络中的网络邻近性 ...	356
8.3.3 容错和失效恢复	358
8.3.4 抗扰动与失效	359
8.4 信任、信誉和安全管理	361
8.4.1 节点信任和信誉系统	361
8.4.2 信任覆盖网络和 DHT 实现	364
8.4.3 PowerTrust：可扩展的信誉 系统	365
8.4.4 加强覆盖网络安全，抵御 DDoS 攻击	367

8.5 P2P 文件共享和版权保护	368	9.2.5 MPI、Azure、EC2、MapReduce、Hadoop 的基准测试	404
8.5.1 快速搜索、副本和一致性 ...	368	9.3 物联网关键技术	406
8.5.2 P2P 内容分发网络	372	9.3.1 实现普适计算的物联网	406
8.5.3 版权保护问题和解决方案 ...	375	9.3.2 射频标识（RFID）	408
8.5.4 P2P 网络中的共谋盗版 预防	376	9.3.3 传感器网络和 ZigBee 技术	410
8.6 参考文献与习题	379	9.3.4 全球定位系统（GPS）	413
第9章 普适云计算、物联网与 社会网络	385	9.4 物联网创新应用	416
9.1 支持普适计算的云趋势	385	9.4.1 物联网应用	416
9.1.1 云计算在 HPC/HTC 和 普适计算中的应用	385	9.4.2 零售和供应链管理	416
9.1.2 NASA 和 CERN 的大规模 私有云	389	9.4.3 智能电网和智能建筑	417
9.1.3 灵活和可扩展的 云混搭系统	391	9.4.4 信息物理系统（CPS）	419
9.1.4 移动云计算平台 Cloudlet ...	394	9.5 在线社会网络和专业网络	420
9.2 分布式系统和云的性能	395	9.5.1 在线社会网络特征	421
9.2.1 科研云综述	396	9.5.2 基于图论的社会网络分析 ...	422
9.2.2 数据密集型扩展计算 (DISC)	399	9.5.3 社会网络社区和应用	424
9.2.3 HPC/HTC 系统的性能指标 ...	400	9.5.4 Facebook：世界上最大的 社会网络	427
9.2.4 云计算的服务质量	403	9.5.5 Twitter：微博、新闻和提醒 服务平台	430
索引	439		

第一部分

Distributed and Cloud Computing, 1E

系统建模、集群化和虚拟化

前三章介绍了系统模型，并回顾了两个关键技术。第1章讲述了分布式系统模型和云平台。第2章介绍了集群化技术，第3章描述了虚拟化技术。这两个技术可以实现分布式计算和云计算。系统模型包括计算机集群、计算网格、P2P网络和云计算平台。系统集群化需要硬件、软件和中间件支持。虚拟化用于创建虚拟机、虚拟集群、数据中心的自动化，并构建弹性云平台。

第1章 分布式系统模型和关键技术

本章介绍过去30年在并行、分布式、云计算领域发生的一些变革。我们研究了科学计算领域的高性能计算（HPC）系统和商业计算领域的高吞吐量计算（HTC）系统。我们检查了集群/MPP、网格、P2P网络和互联网云。这些系统在硬件体系结构、操作系统平台、处理算法、通信协议、安全需求、提供的服务模型等方面均有所不同。本章最后重点介绍了分布式系统中可扩展性、性能、可用性、安全、节能、负载外包、数据中心保护等方面的基本问题。

本章主要由Kai Hwang（黄铠）撰写，Geoffrey Fox（1.4.1节）和Albert Zomaya（1.5.4节）撰写了部分内容，Nikzad Rivandi、Young-Choon Lee、Xiaosong Lou（楼肖松）和Lizhong Chen（陈理中）做了一些辅助工作。最终稿由Jack Dongarra审校。

第2章 可扩展并行计算集群

集群化使得构建满足HPC与HTC应用的可扩展并行和分布式系统成为可能。现在的集群节点用物理服务器或虚拟机构建。本章主要研究集群计算系统和大规模并行处理器。我们专注于硬件、软件、中间件的设计原则和评估。我们讨论可扩展性、实用性、可编程性、单系统镜像、作业管理和容错能力。我们将研究近年来报告的3个顶尖超级计算机系统（分别名为Tianhe-1A、Gray XT5 Jaguar和IBM Roadrunner）中的集群化MPP体系结构。

本章由Kai Hwang（黄铠）和Jack Dongarra共同撰写，Rajkumar Buyya和Ninghui Sun做了部分贡献。

Zhiwei Xu（徐志伟）、Zhou Zhao（赵洲）、Xiaosong Lou（楼肖松）和Lizhong Chen（陈理中）提供了技术帮助。

第3章 虚拟机和集群与数据中心虚拟化

虚拟化技术通过在同一组硬件主机上多路复用虚拟机（Virtual Machine, VM）的方式来共享昂贵的硬件资源。近年来，虚拟机的大量涌现扩展了系统应用的范围，并且提升了计算机性能和效率。我们介绍了虚拟机、虚拟机在线迁移、虚拟集群构建、资源配置、虚拟配置适应，以及用于云计算应用的虚拟化数据中心的设计。我们强调使用虚拟集群的作用，以及构建动态网格和云平台的虚拟化资源管理。

本章由Zhibin Yu（喻之斌）和Kai Hwang（黄铠）共同撰写，金海、廖小飞、秦中元、陈理中和赵洲提供了技术帮助。

第1章 |

Distributed and Cloud Computing, 1E

分布式系统模型和关键技术

本章介绍过去 30 年在变化负载和大数据集的应用驱动下，并行、分布式、云计算领域发生的一些变革。我们研究了并行计算领域要求高性能和高吞吐量的一些计算系统，如计算机集群、SOA、计算网格、P2P 网络、互联网云和物联网。这些系统在硬件体系结构、系统平台、处理算法、通信协议、提供的服务模型等方面均有所不同。本书也介绍了分布式系统中可扩展性、性能、可用性、安全、节能等方面的基本问题。

1.1 互联网之上的可扩展计算

在过去的 60 多年间，计算技术的平台和环境经历了一系列的变革。在这一节，我们将介绍在硬件体系结构、操作系统平台、网络连接、应用负载方面的革命性创新。一个并行的、分布式的计算系统使用大量的计算机解决互联网上的大规模计算问题，而不是使用一个集中式的计算机解决计算问题。这也导致分布式计算的缺点是数据敏感和网络中心化。本节选取了一些使用并行和分布式计算模型的现代计算机系统。在今天的社会，这些大规模互联网应用提高了生活和信息服务的质量。

1.1.1 互联网计算的时代

每天有数十亿人使用互联网。因此，超级计算机和大规模数据中心必须面向巨量的互联网用户并发地提供高性能计算服务。由于这样高的需求，用于高性能计算（High-Performance Computing, HPC）系统性能测试的 Linpack 基准不再适合和最优。云计算的出现改为要求采用并行和分布式计算技术来构建的高吞吐量计算系统（High-Throughput Computing, HTC）^[5,6,19,25]。我们必须升级数据中心，采用更快的服务器、存储系统和高带宽网络。其目的是利用不断涌现的新技术来改进基于网络的计算和 Web 服务。

1.1.1.1 平台的变革

计算机技术经历了五代的发展，每一代持续 10 ~ 20 年。连续的两代之间会有 10 年左右的交迭。例如，从 1950 年到 1970 年，用于满足大公司和政府组织的计算需求的是少数大型机，包括 IBM 360 和 CDC 6400。从 1960 年到 1980 年，在小公司和大学，低成本的微型计算机（如 DEC PDP 11 和 VAX 系列）变得流行起来。

从 1970 年到 1990 年，使用 VLSI 微处理器的个人计算机到处可见。从 1980 年到 2000 年，在 4 有线和无线应用中出现了海量的便携式计算机和通用型设备。自 1990 年以来，隐藏在集群、网格或互联网云背后的 HPC 和 HTC 系统应用不断增长扩散。这些系统既被用于高端 Web 规模计算和信息服务，也提供给用户。

普遍的计算趋势是平衡共享的网络资源和互联网上的海量数据。图 1-1 阐述了 HPC 和 HTC 系统的演化。在 HPC 方面，超级计算机（大规模并行处理器（Massively Parallel Processors, MPP））逐渐地被协同计算机集群所替代，不再有共享计算资源的要求限制。集群通常是一个物理上处在近距离范围且彼此连接的同构计算节点的集合。我们将在第 2 章和第 7 章更详细地讨论

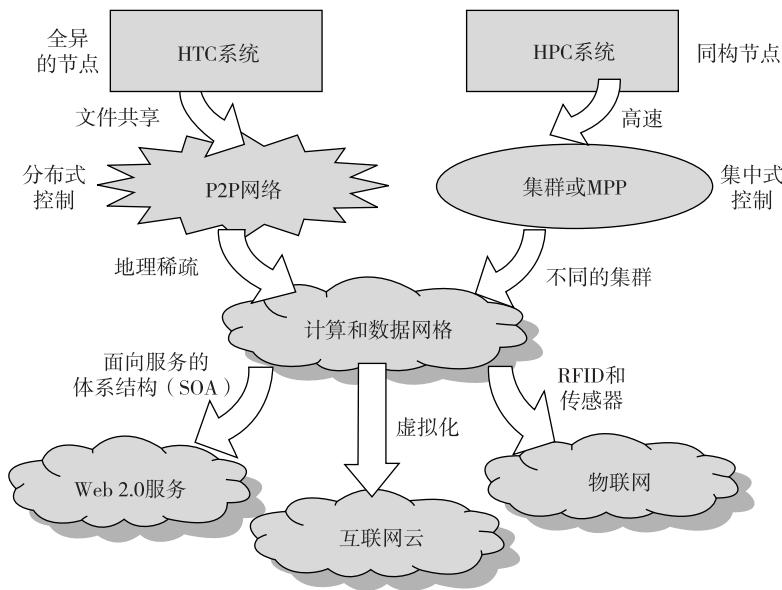


图 1-1 HPC 和 HTC 系统的演化趋势（并行、分布式、云计算，具有集群、MPP、P2P 网络、网格、云、Web 服务和物联网）

集群系统、MPP 系统和网格系统。

在 HTC 方面，对等（Peer-to-Peer, P2P）网络起源于分布式文件共享和内容分发应用。一个 P2P 系统建立在众多客户机之上（第 5 章将会进行更详细的讨论）。节点机器是完全分布式的。P2P、云计算和 Web 服务平台更关注 HTC 应用，而非 HPC 应用。集群和 P2P 技术促进了计算网格和数据网格的发展。

1.1.1.2 高性能计算

多年以来，HPC 系统强调系统的原生速度性能。HPC 系统的速度已经从 20 世纪 90 年代初每秒十亿次浮点运算（Gflops）增长到 2010 年的每秒千万亿次浮点运算（Pflops）。这个增长来自于科学、工程、制造业的需求驱动。例如，世界计算机系统 500 强评测采用的是 Linpack 基准结果中的浮点运算速度。然而，超级计算机用户数量不到全部计算机用户的 10%。今天，大多数计算机用户使用台式计算机，或者在开发互联网搜索和市场驱动计算任务时使用大型服务器。

1.1.1.3 高吞吐量计算

面向市场的高端计算系统的发展正发生策略上的从 HPC 范式到 HTC 范式的转变。HTC 范式更关注于高通量计算。高通量计算主要应用于被百万以上用户同时使用的互联网搜索和 Web 服务。性能目标因而转移到测量高吞吐量或单位时间内任务完成数。HTC 技术不仅需要提高批处理任务速度，在很多数据和企业计算中心还要考虑突发问题开销、能量节约、安全和可靠性。本书将既讲述 HPC 也讨论 HTC 系统以满足所有计算机用户的需求。

1.1.1.4 三种新的计算范式

如图 1-1 所示，随着 SOA 体系结构的引入，Web 2.0 服务变得可行。先进的虚拟化技术使互联网云作为一种新的计算范式得以不断成长。射频识别（Radio-Frequency IDentification, RFID），全球定位系统（GPS）和传感技术的成熟触发了物联网（Internet of Things, IoT）的发展。这些新的范式在这里只进行简要介绍。我们将在第 5 章详细研究 SOA；第 3 章主要讨论虚拟化；云计算系统在第 4 章、第 6 章和第 9 章加以论述；第 9 章还将介绍包含信息物理融合系统（Cyber-Physical Systems, CPS）的物联网。

从 1969 年引入互联网开始，加州大学洛杉矶分校的 Leonard Kleinrock 断言：“到目前为止，计算机网络的发展仍处于幼年时期，但随着网络的成长和成熟，我们将可以看到计算机效用的扩展，就像电和电话的广泛应用，将会惠及全世界的家庭和办公室。”从那时起，许多人已经重新定义“计算机”这个词。1984 年，Sun Microsystems 公司的 John Gage 更是提出了“网络就是计算机”的口号。2008 年，加州大学伯克利分校的 David Patterson 说：“数据中心就是计算机。这种以服务的方式将软件提供给数百万用户与之前的分发软件让用户在自己的 PC 上运行有着明显的不同。”最近，墨尔本大学的 Rajkumar Buyya 言简意赅地说：“云就是计算机。”

本书讨论的问题包括集群、MPP、P2P 网络、网格、云、Web 服务、社会网络和物联网。事实上，集群、网格、P2P 网络和云之间的区别将会越来越模糊。一些人视云计算为通过虚拟化技术适度变化的网格计算或集群。其他人认为这种改变是颠覆性的，因为云计算被预期用于处理传统互联网、社会网络和未来的物联网产生的海量数据集。在后续的章节，所有分布式和云计算系统模型之间的区别和依存关系将会更加清晰和透明。

1.1.1.5 计算范式间的区别

关于集中式计算、并行计算、分布式计算、云计算的精确定义，一些高科技组织已经争论了 6 多年。通常来讲，分布式计算和集中式计算相反。并行计算领域与分布式计算在很大程度上有交迭，云计算与分布式计算、集中式计算、并行计算都有一部分的交集。下面的列表更清晰地定义了这些术语，后续章节将对其体系结构和运行进行更深入的讨论。

- **集中式计算：**这种计算范式是将所有计算资源集中在一个物理系统之内。所有资源（处理器、内存、存储器）是全部共享的，并且紧耦合在一个集成式的操作系统中。许多数据中心和超级计算机都是集中式系统，但它们都被用于并行计算、分布式计算和云计算应用中^[18,26]。
- **并行计算：**在并行计算中，所有处理器或是紧耦合于中心共享内存或是松耦合于分布式内存。一些学者称之为并行处理^[15,27]。处理器间通信通过共享内存或通过消息传递完成。通常称有并行计算能力的计算系统为并行计算机^[28]。运行在并行计算机上的程序称为并行程序。编写并行程序的过程称为并行编程^[32]。
- **分布式计算：**这是一个计算机科学和工程中研究分布式系统的领域。一个分布式系统^[8,13,37,46]由众多自治的计算机组成，各自拥有其私有内存，通过计算机网络通信。分布式系统中的信息交换通过消息传递的方式完成。运行在分布式系统上的程序称为分布式程序。编写分布式程序的过程称为分布式编程。
- **云计算：**一个互联网云的资源可以是集中式的也可以是分布式的。云采用分布式计算或并行计算，或两者兼有。云可以在集中的或分布式的大型数据中心之上，由物理的或虚拟的计算资源构建。一些学者认为云计算是一种效用计算或者服务计算形式^[11,19]。

和前面的术语相比，一些高科技组织更喜欢并发计算或者并发编程这个术语。虽然这些术语通常代表并行计算和分布式计算，但有倾向的从业者也会给出不同的解释。普适计算是指在任何地点和时间通过有线或者无线网络使用普遍的设备进行计算。物联网是一个日常生活对象（包括计算机、传感器、人等）网络化的连接。物联网通过互联网云实现任何对象在任何地点和时间的普适计算。最后，互联网计算这一术语几乎涵盖了所有和互联网相关的计算范式。本书将覆盖前面提及的所有计算范式，重点介绍分布式计算和云计算及其运行的系统，包括集群、网格、P2P 和云系统。

1.1.1.6 分布式系统家族

自 20 世纪 90 年代中期以来，建立 P2P 网络和集群网络的技术在许多设计构建广域计算基础设施的国家项目中得以巩固，被称为计算网格或数据网格。最近，我们已经见证到一个探索互联

网云中数据敏感应用的热潮。互联网云是迁移桌面计算到使用服务集群和数据中心大规模数据库的面向服务计算的结果。本章将介绍各种并行计算和分布式计算的基础知识。网格和云则是 7 更加关注于硬件、软件和数据集方面资源共享的不同系统。

本书也涉及这些大规模分布式系统的设计理论、关键技术和案例研究。大规模分布式系统意在多机上达到高度并行和并发。在 2010 年 10 月，拥有最高性能的集群是中国制造的由 86 016 个 CPU 处理器核心和 3 211 264 个 GPU 核心组成的天河一号系统（Tianhe-1A）。最大的计算网格连接了数百个服务器集群。一个典型的 P2P 网络可能包含数百万同时运行的客户机。实验云计算集群也由数千个处理节点组成。我们在第 4~6 章将专门讨论云计算。HTC 系统的案例分析将放在第 4 章和第 9 章，包括数据中心、社会网络和虚拟云平台。

未来，HPC 和 HTC 系统都将需要每个核可以处理大量计算线程的多核或众核处理器。HPC 和 HTC 系统都强调并行和并发计算。未来的 HPC 和 HTC 系统必须满足计算资源在吞吐量、效率、可扩展性、可靠性方面的巨大需求。系统效率决定于速度、可编程性和能量因素（如每瓦能量消耗的吞吐量）。达到这些目标需要遵从如下设计原则：

- **效率：**在 HPC 系统中开发大规模并行计算时，度量执行模型内资源的利用率。对于 HTC 系统，效率更依赖于系统的任务吞吐量、数据访问、存储、节能。
- **可信：**度量从芯片到系统到应用级别的可靠性和自管理能力。目的是提供有服务质量（QoS）保证的高吞吐量服务，即使是失效的情况下。
- **编程模型适应性：**度量在海量数据集和虚拟云资源上各种负载和服务模型下支持数十亿任务请求的能力。
- **应用部署的灵活性：**度量分布式系统能够同时很好地运行在 HPC（科学和工程）和 HTC（商业）应用上的能力。

1.1.2 可扩展性计算趋势和新的范式

技术上一些可预测的趋势是驱动计算应用的。事实上，设计者和开发者想预测新系统的技术承载力。例如，Jim Gray 的论文“数据工程中的经验方法”，是一个技术如何影响应用的经典例子，反之亦然。另外，摩尔定律预测处理器速度每 18 个月翻一番。虽然在过去的 30 年摩尔定律已经得到验证，但很难说在未来一段时间是否仍然有效。

Gilder 定律预测网络带宽在过去的每年翻一番。这种趋势是否能继续？常用硬件极大的价格/性能比率由台式计算机、笔记本电脑、平板电脑等计算设备市场所驱动。这也决定了大规模计算产品技术的采纳和使用。我们将在下面的章节中更详细地讨论这些计算趋势。目前，重点是理解分布式系统如何同时强调资源分布和并发或高并行度（Degree of Parallelism, DoP）。在我们 8 讨论分布式计算的特殊需求之前，回顾一下并行度的概念。

1.1.2.1 并行度

50 年前，当硬件庞大而昂贵时，大多数计算机都采用位串行方式。在这样的场景中，位级并行（Bit-Level Parallelism, BLP）将位串行处理过程逐渐转变成字级处理。这些年来，用户经历了 4 位微处理器到 8 位、16 位、32 位、64 位 CPU 的逐步变化。这引领我们进行下一波的改进，即指令级并行（Instruction-Level Parallelism, ILP），处理器同时执行多条指令而不是一个时刻执行一条指令。在过去的 30 年间，我们已经通过指令流水线、超标量计算、VLIW（Very Long Instruction Word，超长指令字）体系结构、多线程实践了 ILP。ILP 需要分支预测、动态规划、投机预测、提高运行效率的编译支持。

数据级并行（Data-Level Parallelism, DLP）的流行源于 SIMD（Single Instruction, Multiple Data，单指令多数据）和使用向量与数组指令类型的向量机。DLP 需要更多的硬件支持和编译器辅助来实现。自从多核处理器和片上多处理器（Chip MultiProcessor, CMP）引入后，我们进行了

任务级并行 (Task-Level Parallelism, TLP) 的一些探索。一个现代处理器已经满足所有前述并行类型。事实上，BLP、ILP 和 DLP 已经在硬件和编译器层面得到很好的支持。然而由于多核片上多处理器高效执行在编程和代码复杂化上的困难，TLP 还是非常成功。随着并行处理向分布式处理的转移，我们将看到计算粒度向作业级并行 (Job-Level Parallelism, JLP) 的逐渐增长。可以说，粗粒度并行是建立在细粒度并行之上的。

1.1.2.2 创新型应用

在很多应用层面，HTC 和 HPC 系统都需要透明性。例如，数据访问、资源分配、过程定位、并发执行、作业复制，以及错误恢复对于用户和系统管理都应该是透明的。表 1-1 突出显示了这些年来驱动并行和分布式系统发展的几个关键应用。这些应用广泛应用于许多重要领域，包括科学、工程、商业、教育、卫生保健、交通控制、互联网和 Web 服务、军事，以及政府应用。

表 1-1 高性能和高吞吐量系统应用

领 域	具 体 应 用
科学和工程	科学仿真、基因分析等
	地震预测、全球变暖、天气预报等
商业、教育、服务业和卫生保健	远程通信、内容分发、电子商务等
	银行、股票交易、事务处理等
	空中交通控制、电力网格、远程教育等
互联网和 Web 服务、政府应用	卫生保健、医院自动化、远程医疗等
	互联网搜索、数据中心、决策系统等
	流量监测、病毒预防、网络安全等
关键任务应用	数字化政务、网上纳税申报处理、社会网络等
	军事指挥和控制、智能系统、危机管理等

几乎所有的应用都要求计算经济性、网络规模数据收集、系统可靠性和可扩展性能。例如，分布式事务处理经常出现在银行和财政系统中。事务描绘了可靠银行系统中 90% 的业务。分布式事务中，用户必须处理多数据库服务器。在实时银行业务中，维护事务记录副本的一致性是至关紧要的。其他的复杂因素包括缺少软件支持、网络饱和、应用中的安全威胁。我们将在后续章节更详细地讨论应用和软件支持。

1.1.2.3 效用计算趋势

图 1-2 标识了推动分布式系统及其应用研究的主要计算范式。这些范式有一些共同的特

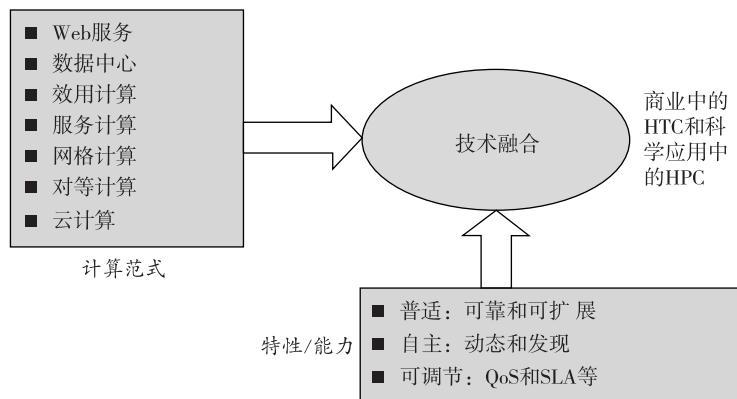


图 1-2 现代分布式计算系统中的计算机效用愿景

注：修改自 Raj Buyya (2010) 的幻灯片。

性。首先，在日常生活中它们是普遍的。在这些模型中，可靠性和可扩展性是两个主要设计目标。其次，它们都是针对自组织支持动态发现的自动化业务。最后，这些范式是 QoS 和 SLA（Service-Level Agreement，服务级协议）可调节的。这些范式及其特性实现了计算机效用的愿景。

效用计算集中于用户从付费服务提供商处获得计算资源的商业模型。所有的网格和云都被视为效用服务提供商。然而，云计算是一个比效用计算更宽泛的概念。分布式云应用运行在边际网络中任何可用的服务器上。这在计算机科学和工程的各个方面都面临着许多技术挑战。例如，用户要求新的高效网络处理器、可扩展的内存和存储方案、分布式操作系统、机器虚拟化中间件、新的编程模型、有效的资源管理和应用程序开发。要构建在所有处理级别探索大规模并行的分布式系统，必需有这些硬件和软件的支持。

9
10

1.1.2.4 新技术成熟周期

任何新出现的计算和信息技术都会经历一个成熟周期，如图 1-3 所示。这个周期展示了在 5 个不同阶段对技术的预期。这种预期在触发阶段到膨胀预期的一个高峰阶段迅速升高。经过一个短期的幻灭阶段，预期会跌入谷底，然后经历一个较长的复苏阶段的平稳增长达到生产力水平成熟期。一个新兴技术达到一个必然的阶段所需年数已经用特殊标志进行了标记。空心圆圈表示两年时间内被主流采纳的技术。灰色圆圈代表 2~5 年被主流采纳的技术。实心圆圈表示将需 5~10 年时间被主流采纳的技术，三角形表示需要 10 年以上时间的技术。十字圆圈代表在达到成熟期前就被淘汰的技术。

图 1-3 中的成熟周期展示了 2010 年 8 月的技术状态。例如，在那时用户生成媒体内容处在幻灭阶段，并被预测在两年内将达到采纳成熟期。互联网微支付系统被预测将在 2~5 年时间内从复苏阶段达到成熟期。3D 打印将需要 5~10 年时间从预期上升阶段达到主流采纳阶段，网状网络：传感器被预计需要 10 年以上时间才能从膨胀预期阶段达到主流采纳成熟阶段。

如图 1-3 所示，云技术刚经过了 2010 年预期阶段的峰值，将在 2~5 年时间达到生产力稳定阶段。预计电力线宽带技术将在 2010 年离开幻灭阶段谷底之前被淘汰。许多其他技术（图 1-3 中用灰色圆圈标识）在 2010 年 8 月处于预期峰值阶段，将可能在未来的 5~10 年达到成熟稳定期。一旦一个技术开始进入复苏期的范围，在 2~5 年将达到生产力成熟阶段。有希望的这类技术是，云计算、生物认证、交互电视、语音识别、预测分析，以及媒体平板电脑。

1.1.3 物联网和 CPS

本节将讨论互联网发展的两种趋势：物联网^[48] 和 CPS。这两个革命性的趋势都强调互联网向日常生活对象的延伸和扩展。这里我们只介绍一些基本概念，第 9 章将给出更详细的讨论。

1.1.3.1 物联网

传统的互联网是机器和机器或者网页和网页之间的连接。物联网的概念 1999 年在 MIT 被提出^[40]。物联网是指日常生活中对象、工具、设备或计算机间存在网络互连。我们可以视物联网为互联了所有我们生活中的对象的无线传感器网络。这些对象可大可小，随时间和地点的变化而变化。这个思路就是使用 RFID、相关传感器或电子技术（如 GPS）来标识每个物体。

11

随着 IPv6 协议的引入， 2^{128} 个 IP 地址足以区分地球上的每个对象，包括所有计算机和专有设备。物联网研究者已经估计出每个人身边将会有 1 000~5 000 个对象。物联网应该设计成可同时追踪百万亿条静态对象或移动对象。物联网需要对所有对象进行统一编址。为了减少标识、搜索和存储的复杂性，可以通过设置阈值过滤掉细小的对象。物联网显然扩展了互联网，在亚洲和欧洲得到了更多的发展。

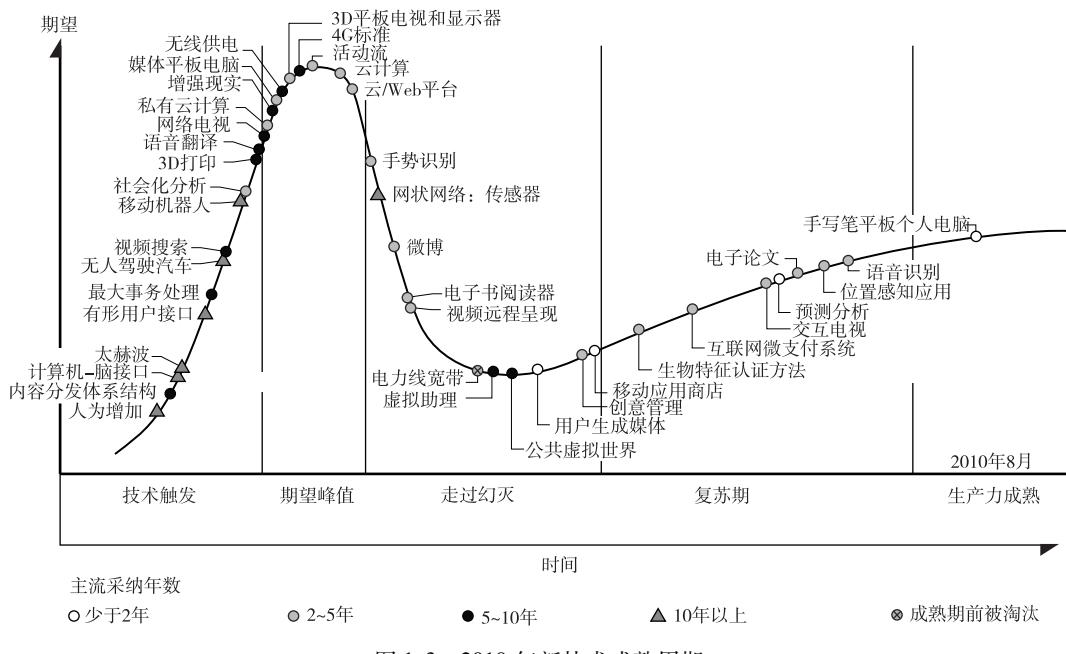


图 1-3 2010 年新技术成熟周期

在物联网时代，所有对象和设备都是工具化的、互连的和智能交互的。这种交流可以发生在人和物或者物和物之间。三种交流模式同时存在：H2H（人和人）、H2T（人和物）、T2T（物和物）。这里，物包括机器，如 PC 和手机。这里的思想是在任何时间、任何地点以较低的成本智能地连接事物（包括人和机器对象）。任何地点连接包括在 PC 上、户内（不在 PC 上）、户外，以及移动中。任何时间连接包括白天、晚上、户外和户内，也包括移动中。

动态连接将会指数型增长成为包含多个网络的一个新的动态网络，称为物联网。物联网仍处在其发展的初级阶段。在编写本书过程中，许多指定区域覆盖的物联网仍处于试验状态。云计算研究者希望用云和下一代互联网技术支持地球上人、机器、任何对象间的快速、有效、智能交互。智慧的地球应该有智能的城市、清洁的水资源、高效的能源、便利的交通、完善的食物供应、负责任的银行、快速的远程通信、绿色的信息技术、更好的学校、良好的医疗、丰富的资源等。要在世界的不同地区实现这个理想的生活环境，还需要花费一定的时间。

1.1.3.2 CPS

CPS 是计算过程和物理世界之间交互的结果。CPS 集成了“计算节点”（同构，异构）和“物理”（并发和信息密集的）对象。CPS 在物理世界和信息世界之间将“3C”技术（计算、通信、控制）融合到了一个智能闭环反馈系统中，已经在美国被积极地研究和探索。物联网强调物理对象之间的多样化连接，而 CPS 强调物理世界中虚拟现实（Virtual Reality, VR）应用的开发和研究。这将改变我们同物理世界交互的方式，就像互联网改变了我们同虚拟世界交互的方式。我们将在第 9 章研究物联网、CPS 及其同云计算之间的关系。

1.2 基于网络的系统技术

随着可扩展性计算的概念日趋成熟，是时候为分布式计算系统设计及其应用开发硬件、软件和网络技术了。尤其应关注在分布式环境中构建处理大规模并行分布式操作系统的可行尝试。

1.2.1 多核 CPU 和多线程技术

过去 30 年组件和网络技术取得了长足的进步，这些对 HPC 和 HTC 系统的发展是至关重要的。在图 1-4 中，处理器速度的测量单位是每秒执行百万条指令数（MIPS），网络带宽的测量单位是兆位每秒（Mbps）或千兆位每秒（Gbps）。单位 GE 指的是 1Gbps 以太网带宽。

1.2.1.1 先进的 CPU 处理器

今天，先进的 CPU 或微处理器芯片采取双核、四核、六核或更多处理核心的多核体系结构。这些处理器在 ILP 和 TLP 级别开拓并行。处理器速度的增长如图 1-4 靠上的曲线所绘，综合了各代微处理器和片上多处理器。我们看到增长从 1978 年 VAX 11/780 的 1MIPS 到 2002 年 Intel Pentium 4 的 1 800 MIPS，上至 2008 年 Sun Niagara 的 22 000 MIPS 的峰值。如图 1-4 所示，在这个例子中，摩尔定律已经被非常精确的验证。30 年中，处理器时钟速率从 Intel 286 的 10MHz 提升到 Pentium 4 的 4 GHz。

然而由于基于 CMOS 的芯片能量上的限制，时钟速率已经达到了极限。在编写本书的时刻，极少数的 CPU 芯片达到了 5GHz 以上的时钟速率。换句话说，除非芯片技术有所突破，否则时钟速率不会再有提高。这个限制主要归因于高频或高电压下额外热量的生成。ILP 在现代处理器中已经得到充分开拓。ILP 机制包括多路超标量体系结构、动态分支预测、猜测执行等方法。这些 ILP 技术要求硬件和编译器支持。另外，DLP 和 TLP 在图形处理单元（Graphics Processing Unit, GPU）上被充分探索，其中 GPU 采用成百上千的简单核心的众核体系结构。

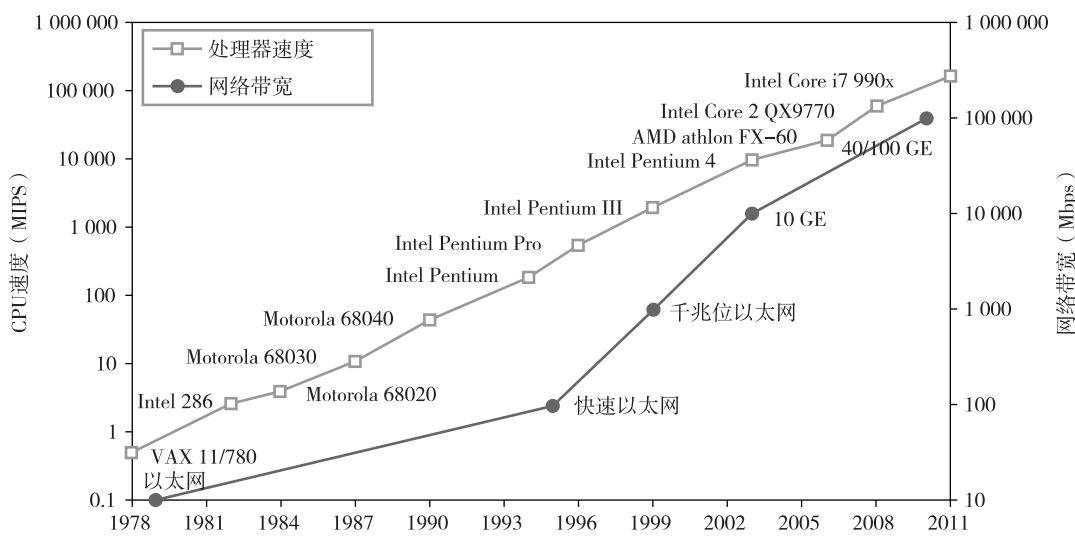


图 1-4 处理器和网络技术近 33 年的发展

注：由南加州大学的楼肖松和陈理中（2011）提供。

14

目前，多核 CPU 和众核 GPU 都可以在不同量级上处理多指令线程。图 1-5 展示了一个标准的多核处理器体系结构。每个核心本质上是一个拥有私有 cache（L1 cache）的处理器。多核与被所有核心共享的 L2 cache 布置在同一块芯片上。将来，多 CMP 甚至是 L3 cache 可以被放在同一块 CPU 芯片上。许多高端处理器都配备多核和多线程 CPU，包括 Intel i7、Xeon、AMD Opteron、Sun Niagara、IBM Power 6 和 X cell 处理器。每个核心也可以多线程。例如，Niagara II 是 8 核的且每个核心可处理 8 个线程。这意味着在 Niagara 上最大化的 ILP 和 TLP 数可以达到 64 ($8 \times 8 = 64$)。如图 1-4 最上方的方块所示，据报道 2011 年 Intel Core i7 990x 的执行速度已经达到 159 000 MIPS。

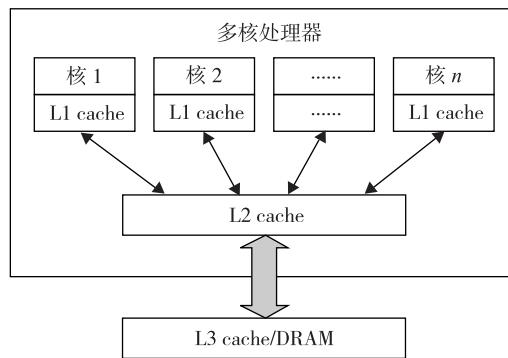


图 1-5 现代多核 CPU 芯片的层次 cache 示意图，其中 L1 cache 是每个核私有的，片上 L2 cache 是共享的，L3 cache 和 DRAM 是非片上的

1.2.1.2 多核 CPU 和众核 GPU 体系统结构

多核 CPU 将从数十个核心增长到数百个甚至更多。但由于前述的内存墙问题，CPU 已经达到大规模 DLP 开拓的极限。这也触发了有数百或更多轻量级核心的众核 GPU 的开发。IA-32 和 IA-64 指令集体系结构都被应用于商业 CPU。现在，x86 处理器已经被扩展用于 HPC 和 HTC 系统的一些高端服务器处理器。

在 Top500 系统中，许多 RISC 处理器已经被替换为多核 x86 处理器和众核 GPU。这个趋势表明在数据中心和超级计算机上 x86 升级将占支配地位。GPU 也被用于大规模集群来建造 MPP 超级计算机。在未来，处理器制造业也渴望开发异构或同构的可同时承载重量级 CPU 核心和轻量级 GPU 核心的片上多处理器芯片。

1.2.1.3 多线程技术

考虑图 1-6，分发 5 个独立指令线程到下面 5 类处理器的 4 条数据流水路径（功能单元），从 15 左到右：4 路超标量处理器、细粒度多线程处理器、粗粒度多线程处理器、双核 CMP、并发多线程（Simultaneous MultiThreaded，SMT）处理器。超标量处理器是带有 4 个功能单元的单线程处理器。三个多线程处理器都是 4 路多线程的，复用 4 条功能数据路径。在双核处理器中，两个处理核心都是单线程的 2 路超标量处理器。

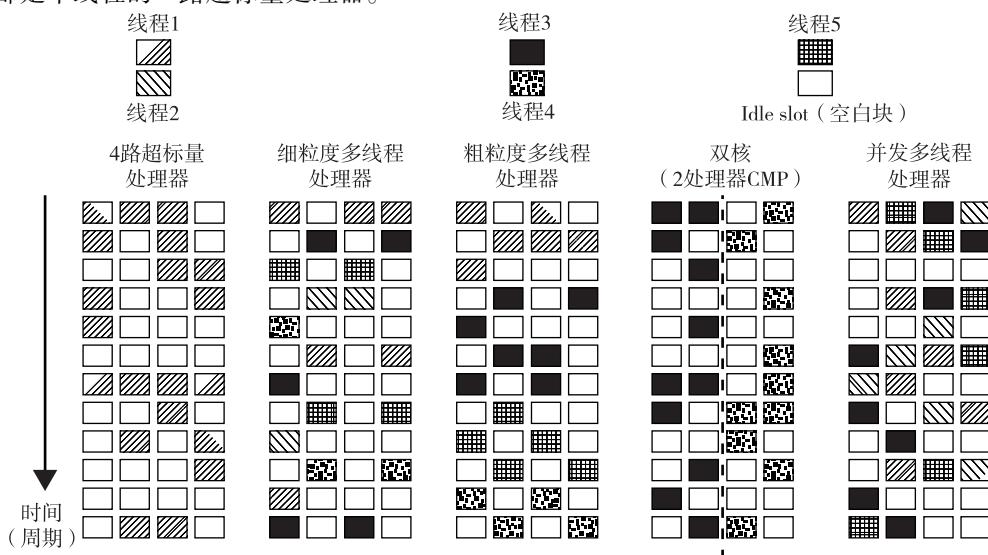


图 1-6 现代 CPU 处理器的 5 种微体系结构，通过多核和支持 ILP 和 TLP

不同线程的指令通过特定的 5 个独立线程指令的影子模式来区分。典型的指令调度模式也再次体现出来。只有同一个线程的指令才能在一个超标量处理器上执行。细粒度多线程在每个周期切换不同线程上指令的执行。粗粒度多线程在切换到下一个线程前在相当多的指令周期内执行同一个线程的多条指令。多核 CMP 完全分别从不同的线程执行指令。SMT 允许在一个时钟周期同时调度不同线程的指令。

这些执行模式近似地模拟一个普通程序。空方块对应在一个特定的处理器时钟周期，某一指令数据路径没有可执行的指令。空格单元越多，说明调度效率越低。很难达到每个处理器周期 ILP 最大化或 TLP 最大化。这里的意图是让读者理解现代处理器 5 种不同的微体系结构的典型指令调度模式。16

1.2.2 大规模和超大规模 GPU 计算

GPU 是图形协处理器或挂载在计算机显卡上的加速器。GPU 将 CPU 从视频编辑应用繁重的图形任务中解脱出来。世界上第一个 GPU (GeForce 256) 是由 NVIDIA 于 1999 年推向市场。这些 GPU 芯片每秒至少可以完成 1 000 万个多边形绘制，目前，几乎市场上的每台计算机都在使用。一些 GPU 特性也被集成到了某些 CPU 上。传统的 CPU 只由几个核构成。例如，Xeon X5670 有 6 个核。然而，一个现代 GPU 芯片集成了至少数百个处理核心。

不像 CPU，GPU 有一个慢速执行多并发线程的大规模并行吞吐体系结构，而不是在一个通常的微处理器上快速地执行一个单独的长线程。现在，并行 GPU 和 GPU 集群相对使用限制并行的 CPU 已经获得了许多关注。GPU 上的通用目的计算，简称为 GPGPU，已经在 HPC 领域出现。NVIDIA 的 CUDA 模型就是用于 HPC 中加入 GPGPU。第 2 章将会详细讨论用于大规模并行计算的 GPU 集群^[15,32]。

1.2.2.1 GPU 如何工作

早期的 GPU 功能是作为附属于 CPU 的协处理器。今天，NVIDIA 的 GPU 已经升级到单芯片集成 128 个核心。而且，GPU 上每个核心能够处理 8 个指令线程。也就是说，在一个 GPU 上最多可同时执行 1 024 个线程。相对于仅能处理几个线程的传统 CPU，这是真正的大规模并行。CPU 通过高速缓存得到优化，而 GPU 的优化则是直接管理片上内存释放更高的吞吐量。

现代 GPU 并不是仅限于加速图形和视频编码。它们还应用于 HPC 系统的多核和多线程级别大规模并行超级计算机。GPU 被设计用于处理大批量并行浮点运算。在某种程度上，GPU 让 CPU 摆脱了所有数据密集型计算，而不只是那些视频处理相关的计算。通常的 GPU 广泛用于手机、游戏终端、嵌入式系统、PC 和服务器。NVIDIA 的 CUDA Tesla 和 Fermi 用于 GPU 集群或是 HPC 系统中的海量浮点数据并行处理。

1.2.2.2 GPU 编程模型

图 1-7 所示是并行执行浮点操作时 CPU 和 GPU 间的交互。CPU 是并行拓展能力有限的通用多核处理器。GPU 拥有数百个简单处理核心组织为多处理器的众核体系结构。每个核心可执行一个或多个线程。本质上说，CPU 的浮点核心计算任务极大地被众核 GPU 承担。CPU 指示 GPU 进行海量数据处理。主板上主存和片上 GPU 内存间带宽必须匹配。这个过程在 NVIDIA 的 CUDA 编程中由 GeForce 8800 或 Tesla 和 Fermi GPU 完成。我们将在第 2 章研究 CUDA GPU 在大规模集群计算中的应用。17

例 1.1 512 CUDA 核心的 NVIDIA Fermi GPU 芯片

2010 年 11 月，世界上最快的 5 台超级计算机中有 3 台 (Tianhe-1A、Nebulae 和 Tsukuba) 使用大量 GPU 芯片加速浮点计算。图 1-8 所示是 Fermi GPU 的体系结构，NVIDIA 的下一代 GPU。这是一个流式多处理器 (Streaming Multiprocessors, SM) 模型。多个 SM 可以集成在一块 GPU 芯

片上。Fermi 芯片由 30 亿个晶体管形成 16 个 SM。每个 SM 由 512 个流式处理器（Streaming Processor, SP）组成，称为 CUDA 核心。Tianhe-1A 中使用的 Tesla GPU 有着与之相同的体系结构，有 448 个 CUDA 核心。

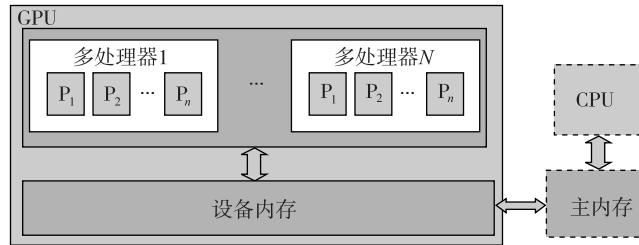


图 1-7 数百或数千处理核心的海量并行处理中协同 CPU 的 GPU 使用

注：由 B. He 等人提供，PAT'08^[23]。

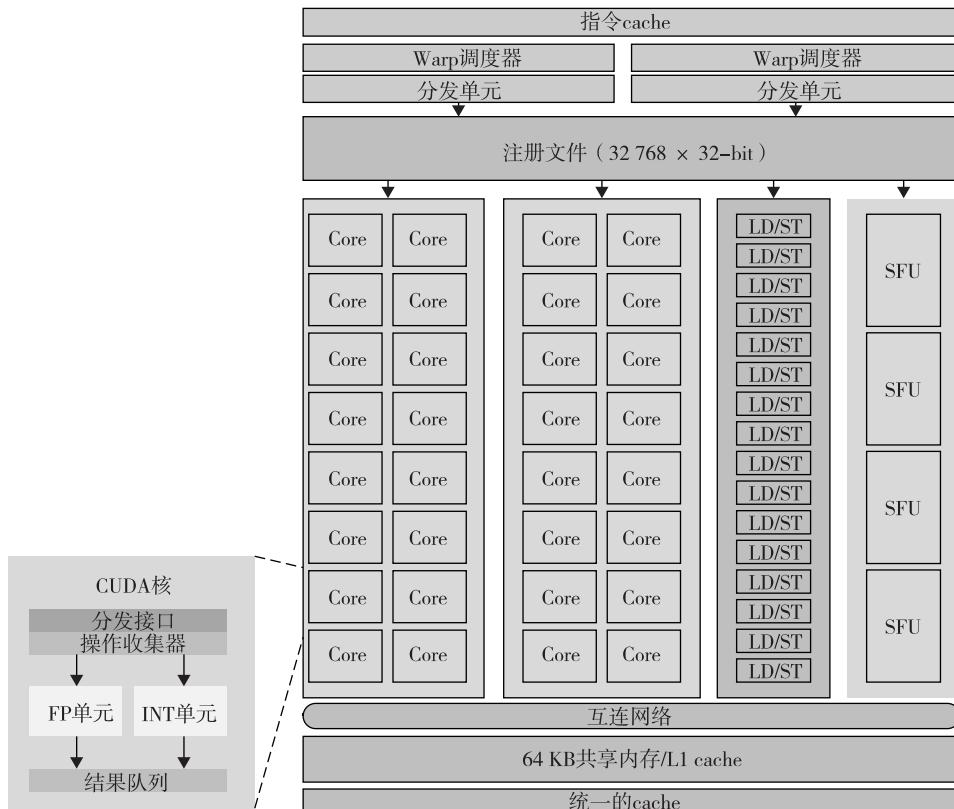


图 1-8 16 个流式多处理器 (SM)，每个有 32 个 CUDA 核的 NVIDIA Fermi GPU，只一个 SM 被展示出来。更多细节可参阅文献[49]

注：由 NVIDIA, 2009 [36] 2011 提供。

Fermi GPU 是较新一代的 GPU，首次出现在 2011 年。Tesla 和 Fermi GPU 可以用于桌面工作站，以加速浮点计算或者用于建设大规模数据中心。图 1-8 所示的体系结构是基于 2009 年 NVIDIA 发布的白皮书^[36]。每个 SM 有 32 个 CUDA 核心。图 1-8 中只是一个 SM。每个 CUDA 核心有一个简单的可用于并行的流水线整型运算器和浮点运算器。每个 SM 有 16 个用于每个时钟周期 16 个线程计算源和目的地址的读/写单元。有 4 个特殊功能单元（Special Function Unit, SFU）用于执行超越指令。

所有功能单元和 CUDA 核心通过 NoC (Network on Chip, 片上网络) 内联于大量的 SRAM 存储 (L2 cache)。每个 SM 有一个 64 KB 的 L1 cache, 768 KB 的统一 L2 cache 由所有的 SM 共享并用于处理所有的负载、存储和结构化操作。内存控制器用于连接 6GB 非片上 DRAM。SM 调度组中纵向的 32 个并行线程。总之, 256/512 FMA (混合乘法和加法) 操作可以并行执行生成 32/64 位的浮点数据结果。如果充分利用, 一个 SM 中的 512 个 CUDA 核心可以并行工作得到 515 Gflops 的双精度运算能力。16 个 SM, 单 GPU 峰值速度达 82.4 Tflops。只有 12 个 Fermi GPU 有可能达到 Pflops 的性能。 ■

在将来, 数千个 GPU 可能会出现在大规模 (Eflops 或 10^{18} flops) 系统中。这反映了未来 MPP 建造采用两种类型芯片的混合体系结构的趋势。在 2008 年 9 月 DARPA 发布的报告中, 提到了大规模计算的 4 个挑战: (1) 能源和电力, (2) 内存和外部存储, (3) 并发和位置, 以及 (4) 系统弹性。这里我们看到了伴随 CPU 在节能、性能和可编程性方面改进的 GPU 的进展^[16]。在第 18 章, 我们将讨论大规模集群中 GPU 的使用。

1.2.2.3 GPU 的节能

斯坦福大学的 Bill Dally 认为能量和海量并行是未来 GPU 相对于 CPU 的主要优势。以现有技术和计算机体系结构推测, 运行一个百亿亿次系统每个核心需要 60 Gflops/W 能量 (见图 1-10)。能量约束了我们在一个 CPU 或 GPU 芯片上所能进行的搭载。Dally 估计出 CPU 芯片每条指令大约消耗 2nJ 能量, 而 GPU 芯片则是每条指令 200pJ 能量, 是 CPU 的 1/10。CPU 针对高速缓存 (cache) 和内存延迟进行优化, 而 GPU 是针对片上内存外部管理的吞吐量进行优化。

图 1-9 比较了 CPU 和 GPU 的以每核心每瓦 Gflops 的值测量的性能/能量之比。2010 年, GPU 在核心级别是 5 Gflops/W, 比 CPU 的每核心 1 Gflops/W 少。这可能限制未来超级计算机的规模。然而, GPU 将终结 CPU 的这种局限。数据运动支配着能量消耗。这需要优化应用的存储层次和裁剪内存。我们需要促进自感知 (self-aware) 操作系统和运行时支持, 并针对基于 GPU 的 MPP, 构建位置感知编译器和自动调节器。这表明能量和软件是未来并行和分布式计算系统的真正挑战。

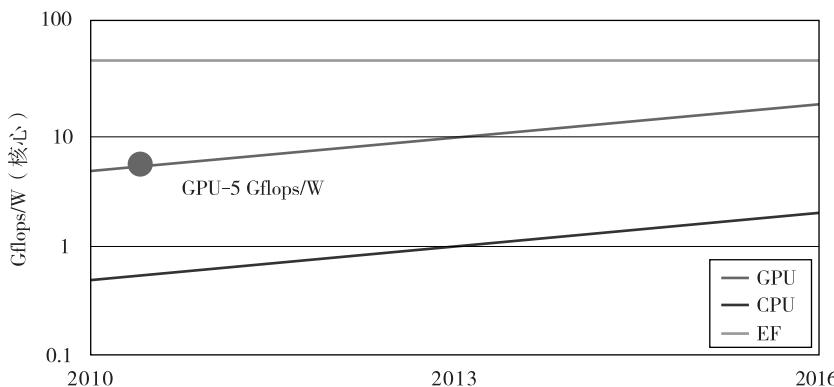


图 1-9 GPU 性能 (中间的曲线, 2011 年每个核心 5 Gflops/W), 相比较低的 CPU 性能 (下面的曲线, 2011 年每个核心 0.8 Gflops/W), 以及 2011 年预计未来每个核心 60 Gflops/W 的性能 (上面曲线中的 EF)

注: 由 Bill Dally 提供^[15]。

1.2.3 内存、外部存储和广域网

1.2.3.1 内存技术

图 1-10 中上面的曲线描绘了 DRAM 芯片容量的增长, 从 1976 年的 16 KB 到 2011 年的 64 GB。这显示了内存芯片在容量上已经历了每三年 4 倍的增长。内存访问时间没有提高太多。

事实上，由于处理器越来越快，内存墙问题变得越来越糟糕。硬盘方面，容量从 1981 年的 260 MB 增长到 2004 年的 250 GB。希捷 Barracuda XT 硬盘在 2011 年达到了 3 TB。这表示在容量上每 8 年约有 10 倍的增长。磁盘阵列容量的增长在接下来的几年将会更大。更快的处理器速度和更大的内存容量导致处理器和内存间更大的差距。内存墙将可能会成为限制 CPU 性能的更为严重的问题。

1.2.3.2 磁盘和存储技术

2011 年以来，磁盘和磁盘阵列容量已经超过了 3 TB。图 1-10 中下面的曲线显示了磁盘存储在 33 年中增长了 7 个数量级。闪存和固态硬盘（Solid-State Drive, SSD）的飞速增长也影响着未来的 HPC 和 HTC 系统。固态硬盘的损坏率并不太坏。通常的 SSD 每个块能处理 300 000 ~ 1 000 000 写操作周期。所以 SSD 能维持几年时间，即使是在高写使用率的情况下。闪存和 SSD 将会在许多应用中示范令人惊讶的速度提升。

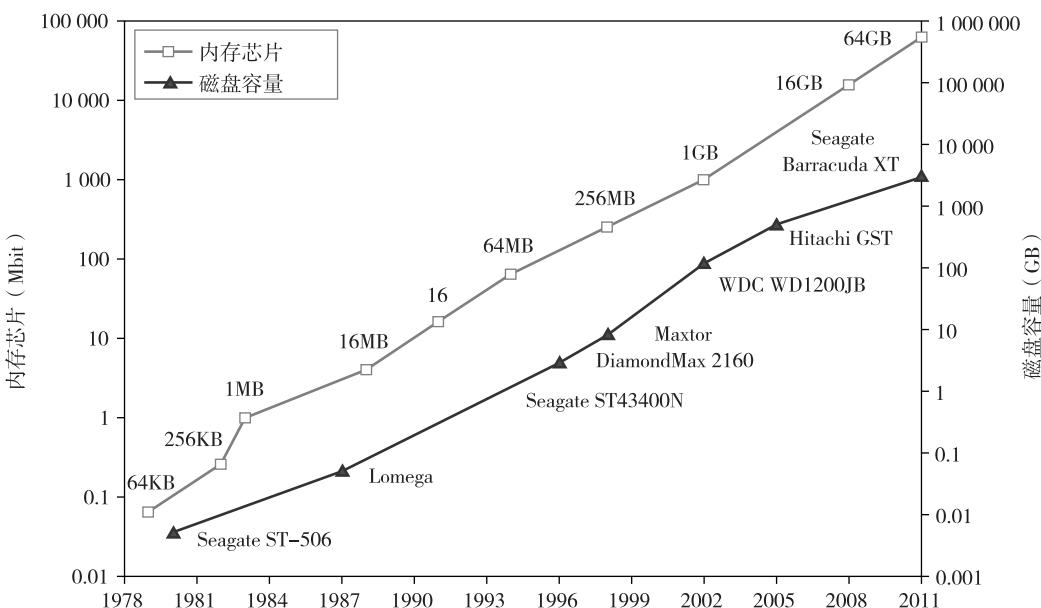


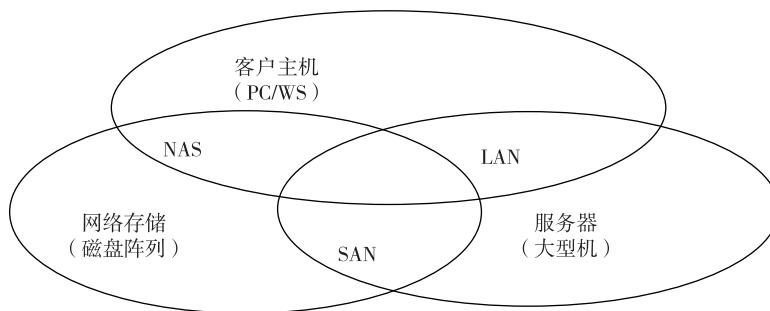
图 1-10 过去 33 年中内存和磁盘技术的进步。2011 年，Seagate Barracuda XT 磁盘容量为 3 TB

注：由南加州大学的楼肖松和陈理中提供。

最后，能量消耗、冷却和包装将会限制大系统发展。功耗关于时钟频率呈线性增长，关于片上电压呈二次方增长。时钟速率不能无限地增长。降低供电电压是非常需要的。Jim Gray 在南加州大学的一次受邀访谈中曾说，“磁带已经不复存在，现在磁盘就是磁带，闪存就是磁盘，内存就是 cache。”这清晰地描绘了未来的磁盘和存储技术。2011 年，在存储市场上用 SSD 代替稳定的磁盘阵列仍然过于昂贵。

1.2.3.3 系统区域互连

小集群中的节点大多通过以太网交换机和局域网（Local Area Network, LAN）互联。如图 1-11 所示，LAN 通常用于连接客户机和大服务器。存储区域网络（Storage Area Network, SAN）连接服务器和网络存储（如磁盘阵列）。网络附加存储（Network Attached Storage, NAS）直接连接客户机到磁盘阵列。所有三种类型的网络经常出现在采用商业网络组件的大集群中。如果没有大的分布式存储被共享，小集群可以采用多端口交换机加铜缆连接终端机器。所有三种类型网络在市场上都可获得。



1.2.3.4 广域网络

图 1-10 中较低的曲线描绘了以太网带宽的飞速增长，从 1979 年的 10 Mbps 到 1999 年的 1 Gbps，到 2011 年的 40 ~ 100 GE。我们可以推测出到 2013 年将达到 1 Tbps 的网络连接。根据 Berman、Fox 和 Hey 所著书中的记载^[6]，2006 年，1000、1000、100、10 和 1 Gbps 带宽的网络连接分别用于国际、国家、组织、光纤桌面和铜缆桌面连接。

网络性能每年增长 2 倍，快于摩尔定律在 CPU 上每 18 个月翻一番的速度。这意味着在将来更多的计算机将会被并发地使用。高带宽网络提高了建设大规模分布式系统的能力。IDC 2010 报告预测无限带宽和以太网会成为 HPC 领域两个主要互连选择。大部分数据中心都使用千兆位以太网作为服务器集群间的互连。

1.2.4 虚拟机和虚拟化中间件

通常的计算机只有一个单操作系统镜像。这提供了应用软件紧耦合于指定的硬件平台的刚性体系结构。一些软件虽然在一台机器上运行良好，但却可能无法在另一个固定操作系统下具有不同指令集的平台上执行。针对未充分利用的资源、应用灵活性、软件可管理性、存在于物理机的安全问题，虚拟机提供了新的解决方案。

目前，建立大规模集群、网格和云，我们需要以虚拟的方式访问大量的计算、存储和网络化资源。我们需要集群化这些资源，并希望提供一个单独的系统镜像。特别地，一个规定资源的云必须动态地依靠处理器、内存和 I/O 设备的虚拟化。我们将会在第 3 章介绍虚拟化。然而，虚拟资源的基本概念（如虚拟机、虚拟存储和虚拟网络，以及虚拟软件或中间件）需要首先介绍。[22]

图 1-12 说明了三种虚拟机配置体系结构。

1.2.4.1 虚拟机

在图 1-12 中，主机配置了物理硬件，如图中底部所示。一个例子是一个 x86 体系结构台式计算机运行已安装的 Windows 操作系统，如图 1-12a 所示。虚拟机可以处在任何硬件系统之上。虚拟机由客户端操作系统管理虚拟资源运行指定应用。在虚拟机和主机平台之间，需要配置一个叫做虚拟机监视器（Virtual Machine Monitor, VMM）的中间层。图 1-12b 所示是一个本地虚拟机，由在特权模式称为 hypervisor 的虚拟机监视器安装。例如，x86 体系结构硬件运行一个 Windows 系统。

客户端操作系统可以是 Linux 系统，hypervisor 是剑桥大学开发的 XEN 系统。这种管理方法也称为裸机虚拟机，因为 hypervisor 直接管理原生硬件（CPU、内存和 I/O）。另一个体系结构是主机虚拟机，如图 1-12c 所示。这里 VMM 运行在非特权模式。主机操作系统不需要修改。虚拟机也可在双模式下实现，如图 1-12d 所示。VMM 一部分运行在用户级，另一部分运行在特权级。这种情况下，主机操作系统可能在某些范围需要修改。多虚拟机可以实现给定硬件系统的接口

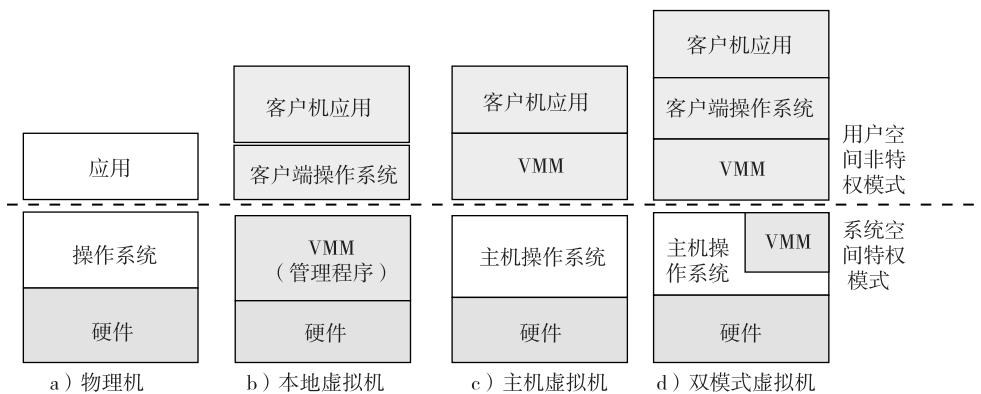


图 1-12 三种虚拟机体系结构与传统的物理机的比较

注：由南加州大学的 M. Abde-Majeed 和 S. Kulkarni (2009) 提供。

以支持虚拟化过程。虚拟机方法提供了操作系统和应用的硬件独立性。用户应用程序和其所在的操作系统可以绑定在一起作为一个可以接口任何硬件平台的虚拟应用工具。虚拟机可以在一个与主机操作系统不同的操作系统上运行。

1.2.4.2 虚拟机原始操作

VMM 提供虚拟机摘要给客户端操作系统。在全虚拟化中，VMM 提供了和物理机相同的虚拟机摘要，以至于一个标准的操作系统（如 Windows 2000 或 Linux）可以像在物理机上一样运行。

23 底层的 VMM 操作由 Mendel Rosenblum^[41]指出，说明如图 1-13 所示。

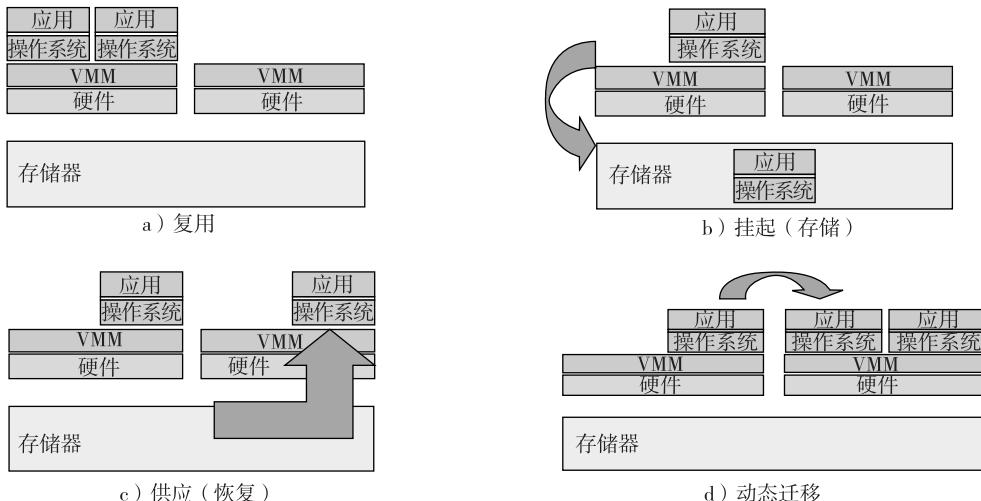


图 1-13 分布式计算环境中的虚拟机复用、挂起、供应和迁移

注：由 M. Rosenblum 提供，ACM ASPLOS2006^[41]。

- 第一，虚拟机可以在硬件机器上复用，如图 1-13a 所示。
- 第二，虚拟机可以挂起并保存在一个稳定存储器上，如图 1-13b 所示。
- 第三，挂起的虚拟机可以在一个新的硬件平台上恢复或者供应，如图 1-13c 所示。
- 第四，虚拟机可以从一个硬件平台迁移到另一个硬件平台，如图 1-13d 所示。

这些虚拟机操作使虚拟机可以提供给任何可用硬件平台，也使分布式应用执行的接口灵活。此外，虚拟机方法有效地提高了服务器资源的利用率。多服务器功能可以在相同的硬件平台上统一以获得更高的系统效率。通过虚拟机系统的开发消除了服务器的扩张，透明地共享硬件。通

过这种方法，VMware 称服务器利用率可以从现在的 5% ~ 15% 提高到 60% ~ 80%。

1.2.4.3 虚拟基础设施

图 1-14 底部用于计算、存储和网络化的物理资源都被映射到顶部集成在各个虚拟机需要的应用上，从而分离了硬件和软件。虚拟基础设施完成资源到分布式应用的连接。系统资源到特定应用的映射是动态的。这降低了成本并提高了效率和响应。用于服务器一致性和牵制策略的虚拟化就是一个很好的例子。我们将在第 3 章讨论虚拟机和虚拟化支持。对集群、云、网格的虚拟化支持将分别在第 3 章、第 4 章和第 7 章介绍。

24

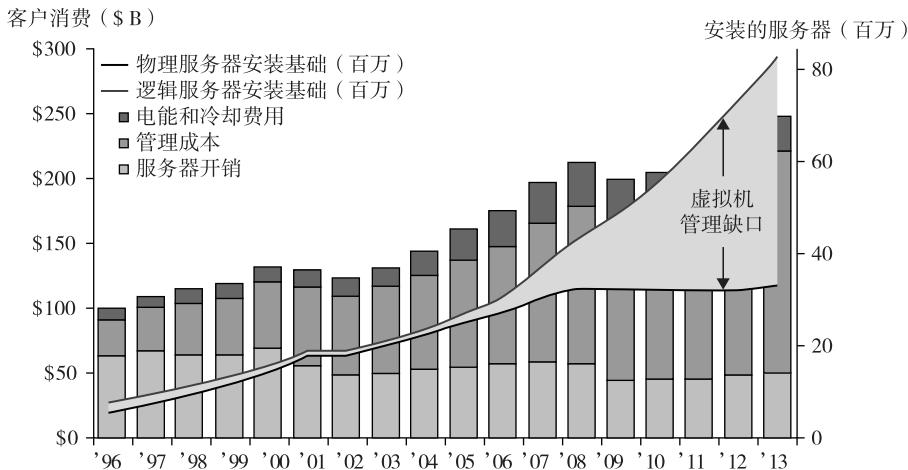


图 1-14 近年来，数据中心的服务器数量增长和成本分析

资料来源：IDC Report, 2009。

1.2.5 云计算的数据中心虚拟化

在这一节，我们讨论数据中心的基本体系结构和设计考虑。云计算体系结构由商业硬件和网络设备建立。几乎所有的云平台都选择使用流行的 x86 处理器。低成本的太字节磁盘和千兆位以太网用于建设数据中心。数据中心设计强调性能/价格比，而不是单一的速度性能。换句话说，存储和节能比只看速度性能更重要。图 1-14 显示了过去 15 年数据中心的服务器数量增长和成本分析。2010 年，全世界大约有 4300 万台服务器在使用。三年后，效用成本超过硬件成本。

1.2.5.1 数据中心的服务器数量增长和成本分析

一个大的数据中心可能有数千台服务器。较小的数据中心通常有数百台服务器。近年来，建设和维护数据中心服务器的成本已经增长。根据一份 2009 IDC 的报告（见图 1-14），通常数据中心的成本中只有 30% 用于购买 IT 设备（如服务器和磁盘），33% 用于冷却，18% 用于不间断电源供应（Uninterruptible Power Supply, UPS），9% 用于机房空调（Computer Room Air Conditioner, CRAC），余下的 7% 用于能量发送、照明、变压器耗费。因此，约 60% 的数据中心运行成本用于管理和维护。服务器购买成本没有随着时间增长太多。电能和冷却的成本在 15 年中从 5% 增长到 14%。

1.2.5.2 低成本设计原则

高端交换机或路由器对于建造数据中心来说可能成本太高。因而，使用高带宽网络可能不利于云计算的经济性。给定固定预算，数据中心更需要商用交换机和网络。类似地，相对昂贵的大型机使用商用 x86 服务器会更好。软件层处理网络流量均衡、容错和扩展性。现在，几乎所有的云计算数据中心都使用以太网作为基础网络技术。

25

1.2.5.3 技术的融合

实质上，云计算依托以下四个方面技术的融合：(1) 硬件虚拟化和多核芯片，(2) 效用和网格计算，(3) SOA、Web 2.0 和网络服务糅合，以及(4) 原子计算和数据中心自动化。硬件虚拟化和多核芯片使云中的动态配置成为可能。效用和网格计算技术是云计算必需的基础。近来在 SOA、Web 2.0、平台糅合上的进展又推动云计算向前迈出一步。最后，云计算的增长要归功于自治计算和自动化的数据中心运作的进展。

Jim Gray 曾提出下面的问题：“科学面对海量数据。如何管理和分析信息？”这说明科学和我们的社会同样都面临海量数据的挑战。数据来自于传感器、实验、仿真、私人文档，以及各种规模和格式的网络数据。保存、移动和访问海量数据集需要通用工具来支持高性能、可扩展文件系统、数据库、算法、工作流和虚拟化。随着科学变成以数据为中心，一个新的科学研究范式将以数据密集型技术为基础。

在 2007 年 1 月 11 日，《计算机科学和电子通讯》(CSTB) 推荐使用 *fostering* 工具进行数据捕获、数据生成和数据分析。四个技术领域中存在一个交互周期。云技术由海量数据发掘的热潮驱动。另外，云计算极大地影响了电子化科学的研究，开拓了多核和并行计算技术。这两个热门领域使海量数据累积。为支持数据密集型计算，需要解决工作流、数据库、算法和虚拟化问题。

通过连接计算机科学技术和科学家，计算机辅助科学和研究在生物、化学、物理、社会科学、人类学等跨学科活动已经形成新的应用前景。云计算如其所承诺的是比数据中心模型更具变革性的尝试。它的根本性改变在于我们如何与信息交互。云计算在体系结构、平台、软件级别提供按需服务。在平台级别，MapReduce 提供了一个新的编程模型，可以透明地处理数据并行并且具有自然的容错能力。我们将在第 6 章更详细地讨论。

迭代的 MapReduce 拓展 MapReduce 以支持更广泛的科学应用中常用的数据挖掘算法。云运行在一个极大的商用计算机集群上。对每个集群节点，多线程通过众核 GPU 集群的大量核心实现。数据密集型科学、云计算、多核计算在体系结构设计和编程挑战上正向下一代计算集群化和变革。[26] 它们激活了流水线：数据成为信息和知识，并促成了如 SOA 期望的机器智能。

1.3 分布式和云计算系统模型

分布式和云计算系统都建立于大量自治的计算机节点之上。这些节点通过 SAN、LAN 或 WAN 以层次方式互连。利用现在的网络技术，几个 LAN 交换机可以方便地将数百台机器连接成一个工作集群。一个 WAN 可以连接许多本地集群形成一个大的集群的集群。从这个角度看，可以建立连接边际网络的数百万台计算机的大系统。

大系统被认为高可扩展，并能在物理上或逻辑上达到 Web 规模互连。在表 1-2 中，大系统被划分为四组：集群、P2P 网络、计算网格、大数据中心之上的互联网云。借助于节点数，这四个系统分类可能包括数百、数千甚至数百万台计算机作为协同工作节点。这些机器在各个级别共同、合力、协作地工作。表 1-2 从技术和应用层面描述了这四个系统分类。

从应用的角度看，集群在超级计算应用中最流行。2009 年，Top500 超级计算机中有 417 台是采用集群体系结构构建的。可以说集群已成为建造大规模网格和云必需的基础。P2P 网络对商业应用最有吸引力。然而，内容业界难以接受 P2P 技术在自组织网络中缺少版权保护的弱点。过去十年构建的许多国家网格都没能充分利用，因为缺少可靠的中间件或编码良好的应用。云计算潜在的优势在于，对提供商和用户都是低成本和简单的。

表 1-2 并行和分布式计算系统分类

功能、应用	计算机集群 ^[10,28,38]	P2P 网络 ^[34,46]	数据/计算网格 ^[6,18,51]	云平台 ^[1,9,11,12,30]
体系结构、网络连接性和大小	计算节点网络通过 SAN、LAN、WAN 有层次地互连	灵活的客户机网络通过覆盖网络逻辑地互连	异构集群在选中的资源地址上通过高速网络链接互连	在数据中心之上满足 SLA 的虚拟服务器集群
控制和资源管理	分布式控制的同构节点，运行 UNIX 或 Linux	自治客户机节点，自由加入和退出，自组织	集中式控制，面向服务器的、授权式的安全	动态调整服务器、存储和网络资源
应用和网络为中心的服务	高性能计算、搜索引擎和 Web 服务等	最适合商业的文件共享、内容分发和社会网络	分布式超级计算、全球难题解决和数据中心服务	升级网络搜索、效用计算和外包计算服务
代表性的运行系统	谷歌搜索引擎、SunBlade、IBM Road Runner、Cray XT4 等	Gnutella、eMule、BitTorrent、Napster、KaZaA、Skype、JXTA	TeraGrid、GriPhyN、UK EGEE、D-Grid、ChinaGrid 等	Google App Engine、IBM Bluecloud、AWS 和 Microsoft Azure

1.3.1 协同计算机集群

一个计算集群由互连的协同工作的独立计算机组成，这些独立计算机作为单一集成的计算资源协同工作。在过去，集群式的计算机系统在处理重负载大数据集任务方面已经发挥了重要作用。

1.3.1.1 集群体系结构

图 1-15 所示是通常的建立在低响应时间、高带宽互连网络上的服务器集群体系结构。这个网络可以像 SAN（如 Myrinet）或 LAN（如以太网）一样简单。要建立更多节点的更大集群，互连网络可以建成千兆位以太网、Myrinet 或 InfiniBand 交换机组成的多级网络。通过使用 SAN、LAN 或 WAN 构成层次化结构，可以通过增加一定数量的节点建立可扩展的集群。集群通过虚拟专用网络（VPN）网关接入互联网。网关 IP 地址定位了集群。计算机的系统镜像由操作系统管理共享集群资源的方式决定。大多数集群的节点计算机是松耦合的。一个服务器节点的所有资源由它自己的操作系统管理。因此，大多数集群因在不同操作系统下有很多自治节点而有多个系统镜像。

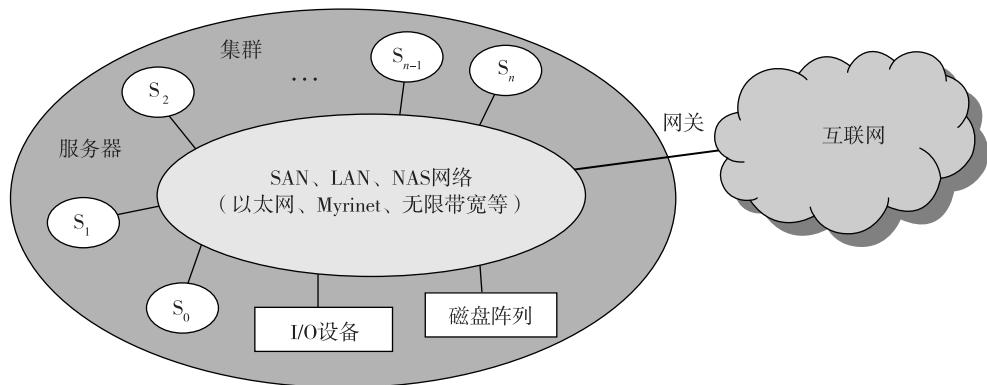


图 1-15 服务器集群通过高带宽 SAN 或 LAN 互连以共享 I/O 设备和磁盘阵列；集群以一个单独计算机的身份接入互联网

1.3.1.2 单系统镜像

Greg Pfister^[38]声称一个理想的集群应该合并多个系统镜像到一个单系统镜像（Single-System Image, SSI）。集群设计者期待一个集群操作系统或者一些中间件在各个级别支持 SSI，包括跨越

所有集群节点共享 CPU、内存和 I/O。SSI 是虚拟的、由软件或硬件资源集合为一个集成的、强大的资源镜像。SSI 使集群在用户看来像一个单独的机器。一个有多个系统镜像的集群除了一群独立的计算机什么都不是。

1.3.1.3 硬件、软件和中间件支持

在第 2 章，我们将讨论小型和大规模集群的设计原理。集群实践大规模并行通常称为 MPP。几乎所有的 Top500 中的 HPC 集群都是 MPP 的。基本的构成部件包括计算机节点（PC、工作站、服务器或 SMP）、特殊的通信软件（如 PVM 或 MPI）和每个计算机节点上的网络接口卡。大多数集群在 Linux 操作系统下运行。计算机节点通过高带宽网络（如千兆位以太网、Myrinet、InfiniBand 等）互连。

特殊的集群中间件支持是用来实现 SSI 或高可用性（High Availability, HA）。串行和并行程序都可以在集群上运行，特殊的并行环境是用来促进集群资源的使用。例如，分布式的内存有多个镜像。用户可能想要通过分布式共享内存（Distributed Shared Memory, DSM）使所有分布式内存在所有服务器上共享。许多 SSI 特性在不同集群操作级别的实现是昂贵或难以达到的。在没实现 SSI 时，许多集群是机器间松耦合的。通过虚拟化，可以根据用户要求动态地建立许多虚拟集群。我们将在第 3 章讨论虚拟集群，在第 4 章、第 5 章、第 6 章和第 9 章讨论虚拟集群在云计算中的应用。

1.3.1.4 主要的集群设计问题

遗憾的是，仍没有一个适合集群的完全资源共享的操作系统。现有的中间件和操作系统扩展都是在用户空间开发的，以在特定功能级别实现 SSI。没有这个中间件，集群节点不能一起有效工作来实现协同计算。软件环境和应用必须依靠中间件来达到高性能。集群利益来自于可扩展的性能、有效的消息传递、高系统可用性、无缝容错和集群视角的作业管理，如表 1-3 所示。我们将在第 2 章分析这个问题。

1.3.2 网格计算的基础设施

在过去 30 年，用户经历了一个从互联网到 Web 和网格计算服务的自然发展。互联网服务（如 Telnet 命令）使本地计算机可以连接到一台远程计算机。一个 Web 服务（如 HTTP）使远程访问 Web 页面成为可能。网格计算被预想用于同时在多台远距离计算机上运行的应用间进行近距离交互。《福布斯杂志》已经预测全球 IT 经济将从 2001 年的 1 万亿美元增长到 2015 年的 20 万亿美元。从互联网到 Web 和网格服务的进展在这个增长中起着重要作用。
[29]

表 1-3 关键集群设计问题和可行实现

特 性	功能描述	可行实现
可用性和支持	硬件和软件支持保证集群持续 HA	失效备援、失效回退、检查点、回滚恢复、不断操作系统等
硬件容错	自主的错误管理避免所有单点失效	组件冗余、热交换、RAID、多电源供应等
单系统镜像（SSI）	通过硬件和软件支持、中间件、操作系统拓展，实现功能级别 SSI	硬件机制或中间件支持用来在缓存一致性级别达到 DSM
有效通信	减少消息传递的系统开销和隐藏延迟	快速消息传递、动态消息、增强的 MPI 库等
集群级作业管理	使用全局作业管理系统进行更好的调度和监控	单作业管理系统应用，如 LSF、Condor 等
动态负载均衡	连同错误恢复平衡所有处理节点的负载	负载监控、处理迁移、作业备份、群组调度等
可扩展性和可编程性	随着负载或数据集的增长，加入更多服务器到集群或加入更多集群到网格	使用可扩展互连、性能监测、分布式执行环境和更好的软件工具

1.3.2.1 计算网格

像电力网格一样，一个计算网格提供一个基础设施，可以把计算机、软件/中间件、特殊指令、人和传感器结合起来。网格通常被架构在 LAN、WAN 或者地区性、全国性或全球规模的互联网骨干网络上。企业或组织将网格呈现为集成的计算资源。它们也可以被视为支持虚拟组织的虚拟平台。网格中的计算机主要是工作站、服务器、集群和超级计算机。个人计算机、笔记本电脑和 PDA 可以作为访问网格系统的设备。

图 1-16 所示是一个建立在为不同组织所拥有的多个计算资源上的计算网格的例子。资源站点提供补充的计算资源，包括工作站、大服务器、处理器网格和 Linux 集群来满足计算需求链。网格建立在各种 IP 宽带的网络上，包括互联网上已被企业和组织使用的 LAN 和 WAN。网格对用户来说是一个集成的资源池，如图 1-16 上半部分所示。

特殊指令可能包括，如在 SETI@ Home 使用射电望远镜在银河中搜寻生命和在 astrophysics @ Swinburne 搜索脉冲星。在服务器端，网格是一个网络。在客户端，我们看到的是有线的或无线的终端设备。网格作为租用服务集成了计算、通信、内容和事务。企业和消费者形成了用户基础，从而决定了使用率趋势和服务特点。许多国家级和国际级网格将在第 7 章介绍，包括美国的 NSF TeraGrid、欧洲的 EGEE 和中国的 ChinaGrid，用于多种科研网格应用。30

1.3.2.2 网格家族

网格技术要求新的分布式计算模型、软件/中间件支持、网络协议和硬件基础设施。紧随国家网格项目之后，IBM、Microsoft、Sun、HP、Dell、Cisco、EMC、Platform Computing 等开发了工业界网格平台。新的网格服务提供商（Grid Service Provider, GSP）和新的网格应用已经迅速形成，类似于过去 20 年互联网和 Web 服务的增长。在表 1-4 中，网格系统从本质上被分为两类：计算或数据网格和 P2P 网格。计算或数据网格主要是建立在国家级别。在第 7 章，我们会介绍一些网格应用和经验。

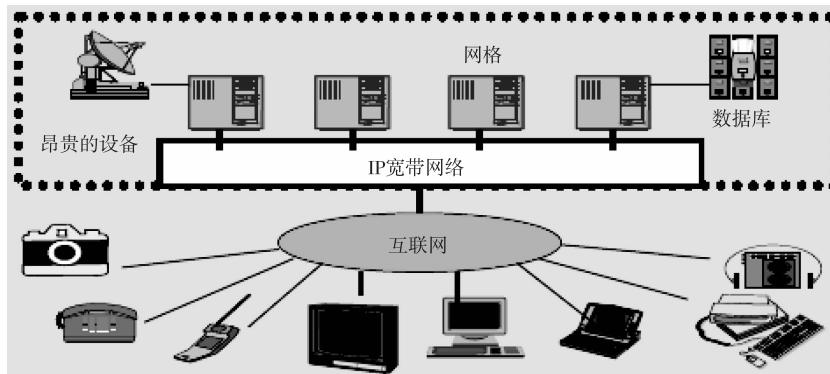


图 1-16 计算网格或数据网格通过资源共享和多个组织间合作提供了计算效用、数据和信息服务

表 1-4 两个网格计算基础设施和代表性系统

设计问题	计算和数据网格	P2P 网格
网格应用	分布式超级计算、国家网格等	开放网格带有 P2P 灵活性、所有资源来自客户机
代表性系统	美国的 TeraGrid、中国的 ChinaGrid、英国的 e-Science 网格	JXTA、FightAid@ home、SETI@ home
已知开发经验	受限的用户组、中间件漏洞、耗资源的协议	不可信的用户贡献资源、受限于少数几个应用

1.3.3 对等网络家族

一个广为大家所知的分布式系统的例子是客户端服务器体系结构。在这个场景中，客户机（PC 和工作站）被连接到一个中央服务器，用来进行计算、电子邮件、文件访问和数据库应用。P2P 体系结构提供了一个分布式的网络化系统模型。最初，P2P 网络是面向客户端而不是面向服务器。本节将在物理层和逻辑层覆盖网络介绍 P2P 系统。

1.3.3.1 P2P 系统

在一个 P2P 系统中，每个节点既是客户端又是服务器，提供部分系统资源。节点机器都是简单的接入互联网的客户机。所有客户机自治、自由地加入和退出系统。这表明对等节点间不存在主从关系。无需中心协作或中心数据库。换句话说，没有节点机器拥有整个 P2P 系统的全局视野。系统是分布式控制下自组织的。

图 1-17 在两个抽象层次展示了 P2P 网络的体系结构。初始时，节点间是完全不相关的。每个节点自由地加入或退出 P2P 网络。任何时候都只有一起参加的节点形成物理网络。不像集群或网格，P2P 网络没有使用一个专一的互连网络。物理网络是一个简单的在各种互联网域使用 TCP/IP 和 NAI 协议随机地形成的特殊网络。因此，物理网络因 P2P 网络中自由的组织关系而在大小和拓扑结构上动态变化。

1.3.3.2 覆盖网络

数据项或文件分布在一起参加的节点中。基于通信或文件共享需求，对等节点（peer）ID 在逻辑层形成一个覆盖网络。这是通过逻辑地映射每台物理机为其 ID 而形成的虚拟网络，虚拟映射如图 1-17 所示。当一个新的对等节点加入系统，它的对等节点 ID 作为一个节点加入到网络中。当一个存在的对等节点离开系统，它的对等节点 ID 会自动地从覆盖网络中移除。因此，刻画对等节点间逻辑连接的是 P2P 覆盖网络。

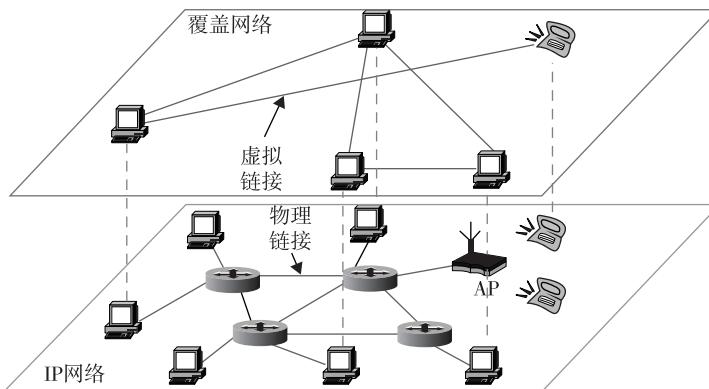


图 1-17 通过映射物理 IP 网络到一个覆盖网络建立虚拟链接的 P2P 系统结构

注：由中国科学院计算所李振宇提供。

32 覆盖网络有两类：非结构化的和结构化的。非结构化的覆盖网络可以用随机图来描述。节点间没有固定的路线发送消息或文件。通常，在一个非结构化网络中使用泛洪向所有节点发出一个查询，因而导致了巨大的网络流量和不确定的查询结果。结构化覆盖网络遵循确定的连接拓扑和从覆盖图中插入与删除对等节点（peer ID）的规则。路由机制是利用结构化重叠开发的。

1.3.3.3 P2P 应用族

基于应用，P2P 网络被分成 4 组，如表 1-5 所示。第一族是 P2P 网络上分布式数字内容（音乐、视频等）的文件共享。这包括许多流行的 P2P 网络，如 Gnutella、Napster 和 BitTorrent 等。合作 P2P 网络包括 MSN 或 Skype 聊天、即时消息和合作设计等。第三族是针对科学应用中的分

布式 P2P 计算。例如，SETI@ home 提供 25 Tflops 的分布式计算能力，集中了约 300 万台互联网主机。其他 P2P 平台，如 JXTA、.NET 和 FightingAID@ home，支持命名、发现、通信、安全，以及在一些 P2P 应用中的资源集群化。我们将在第 8 章和第 9 章更详细地讨论这些主题。

表 1-5 P2P 网络家族主要分类^[46]

系统特性	分布式文件共享	合作平台	分布式 P2P 计算	P2P 平台
有吸引力的应用	MP3 音乐、视频、开放软件等的内容分发	即时消息、协同设计和游戏	科学探索和社会网络	公共资源的开放网络
运行问题	松散的安全性和严格的在线版权侵害	缺乏信任，分布式的垃圾信息、隐私和节点共谋	安全漏洞、自私的合作者和节点共谋	缺少标准和保护性协议
样例系统	Gnutella、Napster、eMule、BitTorrent、Aimster、KaZaA 等	ICQ、AIM、Groove、Magi、Multiplayer、Games、Skype 等	SETI@ home、Geonome@ home 等	JXTA、.NET、FightingAID@ home 等

1.3.3.4 P2P 计算挑战

P2P 计算在硬件、软件和网络需求上面临三类异构问题。有太多的硬件模型和体系结构而无法选择；软件和操作系统间不相容；不同的网络连接和协议使其过于复杂而无法应用于真实应用。我们需要随着负载增加扩展系统。系统规模直接决定于性能和带宽。P2P 网络有这些性质。数据位置对集体性能的影响也很重要。数据局部性、网络邻近性和互操作性是分布式 P2P 应用的设计目标。

P2P 性能受到路由效率和组节点间自组织的影响。容错、失效管理和负载均衡都是使用覆盖网络另外面临的重要问题。对等节点间信任的缺乏形成了另一个问题。对等节点间互相是陌生的。安全、隐私和版权违反是工业界对于在商业应用中使用 P2P 技术的主要担忧^[35]。在一个 P2P 网络中，所有客户端提供的资源包括计算能力、存储空间和 I/O 带宽。P2P 网络的分布式特性也增加了健壮性，因为有限的对等节点失效不会造成单点失效。

通过在多个对等节点间复制数据，很容易丢掉失效节点上的数据。另外，P2P 网络也存在缺点。因为系统不是中央集中的，管理比较困难。此外，系统缺乏安全性。任何人都可以登录系统并引起损坏或滥用。而且，所有连接到 P2P 网络的客户机不能被认为可靠或无病毒。总之，P2P 网络对于少量对等节点是可靠的。仅对低级别安全要求并且不涉及敏感数据的应用有用。我们将在第 8 章讨论 P2P 网络，在第 9 章拓展 P2P 网络到社会网络。

1.3.4 互联网上的云计算

Gordon Bell、Jim Gray 和 Alex Szalay^[5]提倡：“计算科学正向数据密集型转变。超级计算机必须是一个平衡的系统，不仅是 CPU，还要有千兆规模的 I/O 和网络阵列。”将来，处理大数据集将通常意味着把计算（程序）发送给数据，而不是复制数据到工作站。这反映了 IT 业计算和数据从桌面向大规模数据中心移动的趋势，以服务的方式按需提供软件、硬件和数据。数据爆炸促发了云计算的思想。

许多用户和设计者对云计算给出了不同的定义。例如，IBM（云计算领域的主要发起者）给出了如下定义：“云是虚拟计算机资源池。云可以处理各种不同的负载，包括批处理式后端作业和交互式用户界面应用。”基于这个定义，云通过迅速提供虚拟机或物理机允许负载被快速配置和划分。云支持冗余、自恢复、高可扩展编程模型，以允许负载从许多不可避免的硬件/软件错误中恢复。最终，云计算系统可以通过实时监视资源来确保分配在需要时平衡。

1.3.4.1 互联网云

云计算提供了一个虚拟化的按需动态供应硬件、软件和数据集的弹性资源平台（见图 1-18）。它的思想是将桌面计算移到面向服务的平台上，使用数据中心的服务器集群和大数据库。

[34] 云计算利用它的低成本和易用性，使用户和提供商双赢。机器虚拟化使之如此划算。云计算意图同时满足多用户应用。云生态系统必须被设计成安全、可信和可靠的。一些计算机用户认为云就是一个集中式资源池。其他人则认为云是在所有使用的服务器上实践分布式计算的服务器集群。

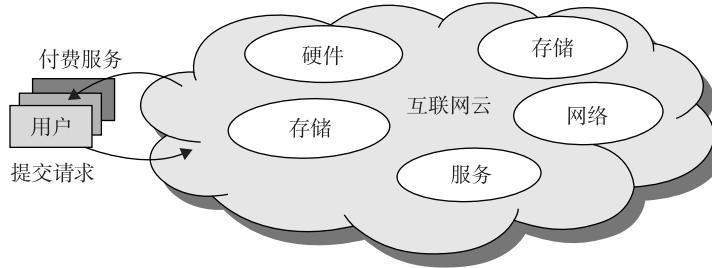


图 1-18 数据中心的虚拟化资源形成互联网云，向付费用户提供硬件、软件、存储、网络和服务以运行他们的应用

1.3.4.2 云前景

通常，一个分布式计算系统属于自治的管理域（如一个科研实验室或公司），运行一个固定的计算需求。然而，这些传统系统遭遇了几个性能瓶颈：日常系统维护、低利用率、硬件/软件升级引起的成本增加。云计算作为一个按需计算范式解决或减轻了这些问题。图 1-19 描述云前景和主要云开拓者，基于三个云服务模型。第 4 章、第 6 章和第 9 章将给出关于云服务的详细说明。第 3 章将介绍相关的虚拟化工具。

- 基础设施即服务 (IaaS)：这个模型将用户需要的基础设施（即服务器、存储、网络和数据中心构造）组合在一起。用户可以在使用客户机操作系统的多个虚拟机上配置和运行指定应用。用户不管理或控制底层的云基础设施，但可以指定何时请求和释放所需资源。
- 平台即服务 (PaaS)：这个模型使用户能够在一个虚拟的云平台上配置用户定制的应用。PaaS 包括中间件、数据库、开发工具和一些运行时支持（如 Web 2.0 和 Java）。平台包括集成了特定程序接口的硬件和软件。提供商提供 API 和软件工具（如 Java、Python、Web 2.0 和 .NET）。用户从云基础设施的管理中得以解脱。
- 软件即服务 (SaaS)：这是指面向数千付费云用户的初始浏览器的应用软件。SaaS 模型应用于业务流程、工业应用、客户关系管理（Consumer Relationship Management, CRM）、企业资源计划（Enterprise Resource Planning, ERP）、人力资源（Human Resources, HR）和合作应用。在用户这边，没有服务器或软件许可方面的前期投入。在提供商这边，同传统的用户应用服务器相比，成本相当低。

[35] 互联网云提供 4 种配置模式：私有、公有、受管理、混合。这些模式对安全有不同的要求。不同的 SLA 表明安全责任由云提供商、云资源用户和第三方云软件提供商共享。云计算的优势已经被许多 IT 专家、工业界领导者和计算机科学研究者提倡。

在第 4 章，我们将描述已经建立的主要云平台和各种云服务。下面列表突出了 8 个原因以适应云的升级的互联网应用和 Web 服务。

- 1) 理想的位置要有受保护的空间和更高的能效。
- 2) 在大用户池间共享峰值负载能力，提升总体效用。
- 3) 基础设施维护责任从特定领域应用开发中分离。

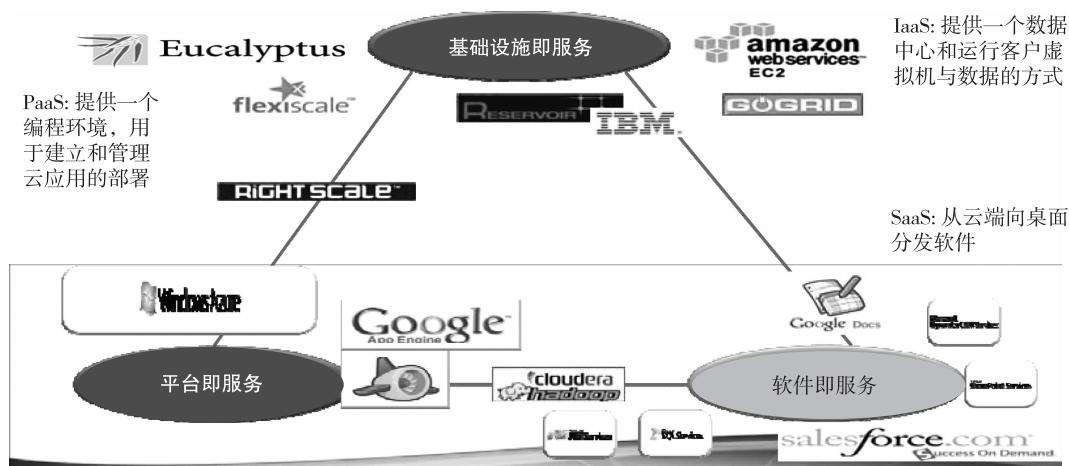


图 1-19 主要提供商的云前景中的三个云服务模型

注：由 Dennis Gannon 在 Cloudcom2010^[19] 的报告中给出。

- 4) 与传统计算范式相比，云计算的成本有效地减少了。
- 5) 云计算编程和应用开发。
- 6) 服务和数据发现与内容/服务分布。
- 7) 隐私、安全、版权和可靠性问题。
- 8) 服务协议、业务模型和价格策略。

1.4 分布式系统和云计算软件环境

本节将介绍使用分布式和云计算系统的流行软件环境。第 5 章和第 6 章将更深入地讨论这个 [36] 主题。

1.4.1 面向服务的体系结构 (SOA)

在网格/Web 服务、Java 和 CORBA 中，实体分别指服务、Java 对象和各种语言中的 CORBA 分布式对象。这些体系结构建立在传统的提供基础网络抽象的开放系统互连协议（Open Systems Interconnection, OSI）层之上。在这之上，我们有一个基础的软件环境，可能是面向 Web 服务的 .NET 或 Apache Axis、面向 Java 的 Java 虚拟机和面向 CORBA 的代理网络。在基础环境之上应建立一个反映分布式系统环境特殊特性的更高层级的环境。这开始了实体接口和内部实体通信，在实体而非位级重建了 OSI 协议栈的最上四层。图 1-20 显示了面向使用 Web 服务和网格系统的分布式实体的分层体系结构。

1.4.1.1 Web 服务和网格分层体系结构

在这些分布式系统例子中，实体接口对应 Web 服务描述语言（Web Services Description Language, WSDL）、Java 方法和 CORBA 接口定义语言（Interface Definition Language, IDL）规范。在这三个例子中，这些接口与定制化的高级通信系统连接：SOAP、RMI 和 IIOP。这些通信系统支持特殊消息模式（如远程过程调用（Remote Procedure Call, RPC））、错误恢复和专门路由。通常，这些通信系统建立在面向消息的中间件（企业总线）设备，如 WebSphere MQ，或者提供了丰富功能、支持虚拟化路由、发送者、接收者的 Java 消息服务（Java Message Service, JMS）。

在容错的情况下，Web 服务可靠性消息传递（Web Services Reliable Messaging, WSRM）框架的特性模拟 OSI 协议能力（如 TCP 容错）在实体级别修改以匹配不同的抽象（如消息对应包、虚拟化地址映射）。安全是一个关键能力，不论是使用还是重新实现此能力，如在互联网协议安

全 (Internet Protocol Security, IPSec) 和 OSI 层的安全套接概念中所见。实体通信由更高级的注册、元数据和实体管理服务所支持，这将在 5.4 节进行讨论。

这里，可以得到几个模型，例如，JNDI (Jini and Java Naming Directory Interface, Jini 和 Java 命名的目录接口) 说明了 Java 中分布式对象模型的不同尝试。CORBA 交易服务、UDDI (通用描述、发现、集成)、LDAP (轻量级目录访问协议) 和 ebXML (使用可扩展标记语言的电子商务) 也是 5.4 节中描述的其他一些发现和信息服务的例子。管理服务包括服务状态和生命周期支持；例子包括 CORBA 生命周期和持久化状态、不同的企业级 JavaBean 模型、Jini 生命周期模型和第 5

37

章的一套 Web 服务规范。上述语言或接口术语形成了实体级能力集合。

后者有性能上的优势并能提供一个“共享内存”模型，使得信息交换更方便。然而，分布式模型有两个主要优势：更高的性能（当通信不重要时来自多 CPU）和具有清晰软件重用和维护优势的更简洁的软件功能分离。分布式模型被期待如默认软件系统方法一样得到普及。早些年间，CORBA 和 Java 方法比今天的 SOAP、XML 或 REST (表征性状态转移) 更多地应用于分布式系统。

1.4.1.2 Web 服务和工具

松耦合和异构实现支持使服务比分布式对象更有吸引力。图 1-20 提到两个服务体系结构选择：Web 服务或 REST 系统（第 5 章有更深入的讨论）。Web 服务和 REST 系统都与建立可靠互操作的系统有明显差距。在 Web 服务中，意在详细说明服务的所有方面和环境。此规范带有简单对象访问协议 (Simple Object Access Protocol, SOAP) 的通信消息。主机环境成为一个使用 SOAP 消息达到完全分布式能力的通用分布式操作系统。这种方法已经混合成功，由于很难在协议关键部分达成一致并且通过软件（如 Apache Axis）更难有效实现协议。

在 REST 方法中，采用简单原则（如通用原则）并把大多数难以解决的问题交给应用（实现规范）软件。在一个 Web 服务语言中，REST 在信息头中信息量最小，消息主体（对通常的消息过程，这是不透明的）携带了所有需要的信息。显然，REST 体系结构更适合飞速发展的技术环境。然而，Web 服务的思想是重要的并将可能在一些成熟的系统被不同级别的协议栈（作为应用的一部分）所需要。注意，REST 可以用 XML 而非 SOAP 的一部分；“HTTP 之上的 XML”是这个关系中流行的设计选择。在通信和管理层之上，我们可以通过集合几个实体组成新的实体或分布式程序。

在 CORBA 和 Java 中，分布式实体由 RPC 连接，构建组合应用的最简单方式是视实体为对象并使用传统的方式将它们连接到一起。对于 Java，这可以如同用远程方法调用 (Remote Method Invocation, RMI) 替代方法调用编写一个 Java 程序一样简单，而 CORBA 支持一个类似的模型，该模型有一个反映 C++ 实体（对象）接口样式的语法。当“网格”这一术语指一个单独的服务或者代表一个服务集合时，这里的传感器就代表输出数据（如消息）的实体，网格和云代表有多个基于消息的输入和输出的服务集合。

38

1.4.1.3 SOA 变命

如图 1-21 所示，面向服务的体系结构 (Service-Oriented Architecture, SOA) 这些年已不断发展。SOA 大体应用于建立网格、云、云网格、网格云、云之云（也称为互联云）和系统的系统。

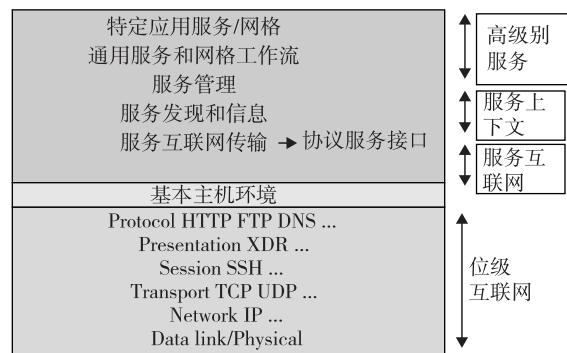


图 1-20 Web 服务和网格的层次化体系结构

大量传感器提供了数据收集服务，在图中表示为 SS（传感器服务）。传感器可以是 ZigBee 设备、蓝牙设备、WiFi 接入点、个人计算机、GPA 或无线电话等。原生数据通过传感设备收集。所有的 SS 设备与大的或小的计算机、各种各样的网格、数据库、计算云、存储云、过滤云、发现云等交互。过滤服务（图中的 fs）用于除去不需要的原始数据，以响应 Web、网格、Web 服务中指定请求。

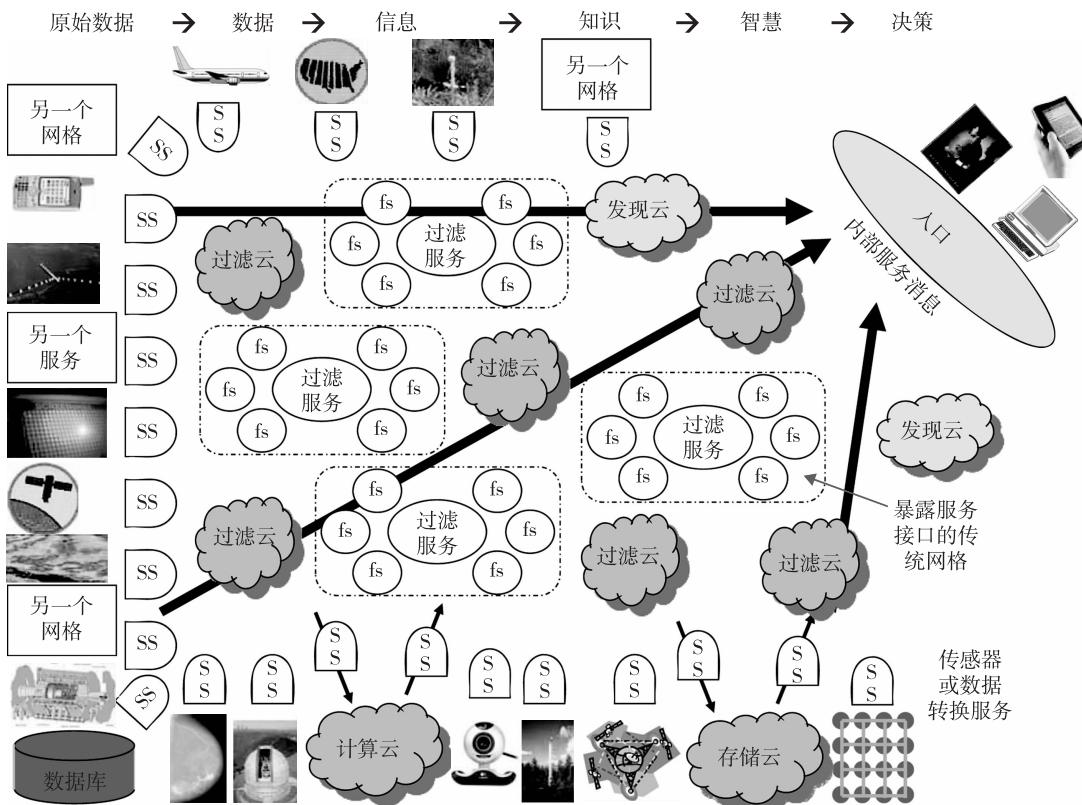


图 1-21 SOA 变革：云网格和网格，其中“SS”指传感器服务，“fs”指过滤或转化服务

过滤服务的集合形成了过滤云。我们将在第 4~6 章介绍各种用于计算、存储、过滤和发现的云，在第 7~9 章介绍各种网格、P2P 网络和物联网。SOA 意在从海量的原始数据项中寻找或挑选出有用的数据。处理这些数据将得到有用的信息，并获得我们日常使用的知识。事实上，智慧或智能是从大知识库中挑选出的。最后，我们基于生物和机器智慧做出明智的决定。读者将在下面的章节更清晰地看到这些结构。

大多数分布式系统需要一个 Web 接口或入口。对于将大量传感器收集的原始数据转化成有用的信息或知识，数据流可能经历一系列的计算、存储、过滤和发现云。最后，内部服务消息在入口聚合，被所有用户访问。如 5.3 节描述的两个入口——OGFCE 和 HUBZero，使用 Web 服务（portlet）和 Web 2.0（gadget）技术。许多分布式编程模型也建立在这些基本构造之上。

1.4.1.4 网格与云

网格和云之间的界限近年来变得越来越模糊。对于 Web 服务，工作流技术用于协调或编排具有指定规范的服务，其中这些规范用于定义关键业务流程模型，如两阶段事务。5.2 节将讨论工作流中使用的通用方法、BPEL Web 服务标准，以及几个重要的工作流方法，如 Pegasus、Taverna、Kepler、Trident 和 Swift。在所有方法中，都是建立一个服务集合来一起解决所有或部分

分布式计算问题。

一般，网格系统使用静态资源，而云强调弹性资源。对一些研究者来说，网格和云之间的不同仅限于基于虚拟化和自治计算的动态资源管理。可以通过多个云建立网格。这种网格比一个单纯的云能更好的工作，因为它能明确支持协议资源分配。从而可以建立系统的系统，如云之云、云网格、网格云，或互联云作为一个基本 SOA 体系结构。

1.4.2 分布式操作系统趋势

在大多数分布式系统中，计算机是松耦合的。因此，一个分布式系统自然有多个系统镜像。这主要是由于所有节点机器运行独立的操作系统。为了提升资源共享和节点机器间快速通信，最好有一个分布式的操作系统一致、有效地管理所有资源。这样的系统非常像一个封闭的系统，它可能依靠消息传递和 RPC 进行内部节点通信。应该指出的是，分布式操作系统对于升级分布式应用的性能、效率和灵活性是至关重要的。

1.4.2.1 分布式操作系统

Tanenbaum^[26]提出了分布式计算机系统中三种分布式资源管理方法。第一种方法是在大量的异构操作系统平台上建立一个网络操作系统，这样一个操作系统对用户提供最低的透明性，本质上是一个节点独立的以文件共享作为通信方式的分布式文件系统。第二种方法是开发一个有限度的资源共享中间件，类似于为集群系统开发的 MOSIX/OS（见 2.4.4 节）。第三种方法是开发一个真正的分布式操作系统以获取更高的使用和系统透明性。40 表 1-6 比较了三种分布式操作系统的功能。

表 1-6 三种分布式操作系统功能比较

分布式操作系统功能	Amoeba 自由大学开发 ^[46]	DCE as OSF/1 (开源软件基金会) ^[7]	MOSIX (希伯来大学用于 Linux 集群) ^[8]
历史和现存系统状态	C 实现，欧洲组织测试，1995 年发布 5.2 版本	建立在 UNIX、VMS、Windows、OS/2 等之上的用户扩展	1977 年开发，现在称为 MOSIX2，用于 HPC Linux 和 GPU 集群
分布式操作系统体系结构	基于微内核和位置透明，使用许多服务器处理文件、目录、副本、运行、启动和 TCP/IP 服务	中间件操作系统提供一个运行分布式应用的平台，支持 RPC、安全和线程	一个分布式操作系统，支持资源发现、进程迁移、运行时支持、负载均衡、泛洪控制和配置等
操作系统内核、中间件和虚拟化支持	一个特殊的微内核，处理低级别进程、内存、I/O 和通信功能	DCE 包处理文件、时间、目录、安全服务、RPC 和中间件或用户空间授权	MOSIX2 运行在 Linux 2.6 上；用于预分配虚拟机组成的多集群和云的拓展
通信机制	使用一个网络层 FLIP 协议和 RPC 实现点对点和群组通信	RPC 支持授权通信和其他用户程序安全服务	使用 PVM、MPI 进行协同通信、进程优先级控制和排队服务

1.4.2.2 Amoeba 和 DCE

DCE 是一个分布式计算环境下的基于中间件的系统。Amoeba 是荷兰自由大学学术性的开发。开源软件基金会 (Open Software Foundation, OSF) 已经推动 DCE 在分布式计算中的使用。然而，Amoeba、DCE 和 MOSIX2 仍都是主要用于学术的研究原型。没有在这些系统基础上的商业操作系统产品。

我们需要新的基于 Web 的操作系统来支持分布式环境下资源虚拟化。这仍是一个完全开放的研究领域。为平衡资源管理负载，这样一个分布式操作系统的功能必须分布在任何可用的服务器上。从这一点来讲，传统的操作系统只能运行在一个集中式的平台上。考虑到操作

系统服务的分布，分布式操作系统设计应采用类似 Amoeba 的轻量级微内核方法^[46]，或者从现有的操作系统上拓展，像 DCE 拓展自 UNIX^[7]。未来的发展趋势是将用户从大部分资源管理职责中解脱出来。

1.4.2.3 用于 Linux 集群的 MOSIX2

MOSIX2 是一个分布式操作系统^[8]，在 Linux 环境中运行一个虚拟化层。这个虚拟化层提供部分单系统镜像给用户应用。MOSIX2 支持串行和并行应用，以及 Linux 节点间的资源发现和进程迁移。MOSIX2 能管理一个 Linux 集群或多集群网格。网格的灵活管理可以使集群拥有者在多个集群拥有者中共享计算资源。只要集群拥有者之间可信，支持 MOSIX 的网格就能无限拓展，MOSIX2 被探索用于管理各类集群中的资源，包括 Linux 集群、GPU 集群、网格，以及使用虚拟机的云。我们将在 2.4.4 节研究 MOSIX 及其应用。[41]

1.4.2.4 透明编程环境

图 1-22 展示了未来计算平台的透明计算基础设施这一概念。用户数据、应用、操作系统和硬件被分成四个级别。数据是属于用户的，独立于应用。操作系统向应用程序开发者提供清晰的接口、标准的编程接口或系统调用。在未来的云基础设施中，硬件将使用标准接口从操作系统中分离出来。因此，用户可以在硬件平台上随意选择使用不同的操作系统。为了从特定的应用程序中分离数据，用户可以使云应用成为 SaaS。因此，用户可以在不同的服务间切换。数据将不会受限于特定的应用。

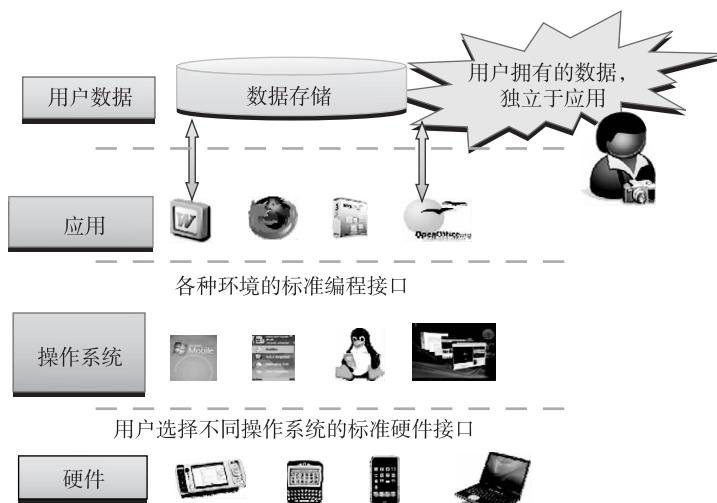


图 1-22 在时间和空间上分离了用户数据、应用、操作系统和硬件的透明计算环境——云计算的一个理想模型

1.4.3 并行和分布式编程模型

这一节，我们将介绍 4 个具有预期可扩展性能和应用灵活性的分布式计算编程模型。表 1-7 总结了三种模型，以及近年来开发的一些软件工具集。正如我们将要讨论的，MPI 是最流行的用于消息传递系统的编程模型。谷歌的 MapReduce 和 Big Table 是用于有效使用互联网云和数据中心资源的。服务云要求拓展 Hadoop、EC2 和 S3 来促进分布式存储系统之上的分布式计算。在过去一段时间，还提出并开发了许多其他模型。在第 5 章和第 6 章，我们将更详细地讨论并行和分布式编程。[42]

表 1-7 并行和分布式编程模型与工具集

模 型	描 述	特 性
MPI	一个可供 C 或 FORTRAN 调用的子程序库，用于编写运行于分布式计算系统的并行程序 ^[6,28,42]	点对点的指定同步和异步，在用户程序中收集通信请求和 I/O 操作，用于消息传递执行
MapReduce	在大数据集或 Web 搜索操作上用于大集群的可扩展的 Web 编程模型 ^[16]	Map 函数生成一个中间的键值对集合；Reduce 函数用相同的键合并所有中间值
Hadoop	一个用于在商业应用中海量数据集上编写和运行大型用户应用程序的软件库	提供给用户商业集群的易于访问的可扩展的、经济的、有效的、可靠的工具

1.4.3.1 消息传递接口（MPI）

这是开发分布式系统上运行的并行程序和并发程序的主要编程标准。MPI 本质上是一个可以被 C 或 FORTRAN 所调用的子程序库，用于编写运行于分布式系统上的并行程序。其思想是用升级的 Web 服务和效用计算应用来体现集群、网格系统和 P2P 系统。除了 MPI 外，低级别基元（如并行虚拟机（Parallel Virtual Machine, PVM））也支持分布式编程。MPI 和 PVM 都是由 Hwang 和 Xu 描述^[28]。

1.4.3.2 MapReduce

这是一个用于大数据集上大规模集群可扩展数据处理的 Web 编程模型^[16]。这个模型主要用于 Web 规模的搜索和云计算应用。用户指定 Map 函数来生成一个中间的键值对的集合。然后用户请求 Reduce 函数来合并所有相同键的值。MapReduce 在探索不同作业级的高度并行上是高可扩展的。一个典型的 MapReduce 计算过程能在成千上万台甚至更多的机器上处理太字节的数据。数百个 MapReduce 程序可以同时执行；事实上，每天有几千个 MapReduce 作业在谷歌的集群上执行。

1.4.3.3 Hadoop 库

Hadoop 提供了一个软件平台，最初由雅虎开发。这个开发包使用户能够在海量分布式数据上编写和运行程序。用户可以简单地划分 Hadoop 来存储和处理 Web 空间中千万亿字节的数据。

43 Hadoop 也是经济的，因为它是 MapReduce 的一个开源实现，最小化了任务数增长和海量数据通信的开销。它是高效的，因为它在很多服务器节点间高度并行地处理数据；它是可靠的，因为它自动保存多个数据副本，以促使发生意外系统故障时重新配置计算任务。

1.4.3.4 开放网格服务体系结构（OGSA）

网格基础设施开发由大规模分布式计算应用驱动。这些应用依靠高度的资源和数据共享。表 1-8 介绍了 OGSA 作为一个网格服务公共使用的通用标准。Genesis II 是 OGSA 的一个实现。关键特性包括一个分布式的执行环境、使用一个本地证书颁发机构（CA）的公钥机制（Public Key Infrastructure, PKI）服务、信任管理和网格计算中的安全策略。

表 1-8 用于科学和工程应用的网格标准和工具包^[6]

标 准	服 务 功 能	关 键 特 性 和 安 全 基 础 设 施
OGSA 标准	开放网格服务体系结构；为公共使用提供通用网格服务标准	支持异构分布式环境、桥接 CA、多信任媒介、动态策略、多安全机制等
Globus 工具包	资源分配、Globus 安全基础设施（GSI）和通用安全服务 API	通过 PKI, Kerberos、SSL、代理、委托和 GSS API 多点登录授权，以保证信息完整性和保密性
IBM 网格工具箱	AIX 和 Linux 网格建立在 Globus 工具包之上，自动计算，备份服务	使用简单 CA、授权访问、网格服务（ReGS），支持用于 Java 的网格应用（GAF4J），用于安全更新的 IntraGrid 中的 GridMap

1.4.3.5 Globus 工具包和扩展

Globus 是一个由美国阿贡国家实验室和 USC 信息科学研究院 10 年前联合开发的中间件库。这个库实现了一些 OGSA 标准，包括网格环境中的资源发现、分配和安全强制。Globus 包支持用 PKI 认证多点相互授权。当前 Globus 版本 GT 4 在 2008 年就已经开始使用。另外，IBM 已经扩展 Globus 用于商业应用。我们将在第 7 章更详细地讨论 Globus 和其他网格计算中间件。

1.5 性能、安全和节能

这一节，我们将结合经验法则讨论构建海量分布式计算系统的基本设计原则。内容包括集群、网格、P2P 网络和互联网云中的可扩展性、可用性、编程模型和安全问题。

44

1.5.1 性能度量和可扩展性分析

性能度量是测量分布式系统所必需的。在这一节，首先我们将讨论不同维度的可扩展性和性能法则。然后我们将检查系统可扩展性与操作系统镜像和其他限制性因素的关系。

1.5.1.1 性能度量

我们在 1.3.1 节以 MIPS 为单位度量 CPU 速度，以 Mbps 为单位度量网络带宽来估测处理器和网络性能。在一个分布式系统中，性能与许多因素相关。系统吞吐量经常用 MIPS、Tflops（每秒 T 浮点运算次数）或 TPS（Transactions Per Second，每秒事务数）测量。其他度量包括作业响应时间和网络延迟。一个比较好的互连网络是低延迟和高带宽的。系统开销通常归因于操作系统启动时间、编译时间、I/O 数据速率和运行时支持系统消耗。其他性能相关度量包括互联网和 Web 服务的 QoS、系统可用性和可靠性，以及系统抵抗网络攻击的安全弹性。

1.5.1.2 可扩展性维度

用户希望拥有一个能获得可扩展性能的分布式系统。任何系统资源升级都应该后向兼容已有的硬件和软件资源。过度设计是不合算的。系统扩展由于许多实际因素可能带来资源的增加或减少。下面的可扩展性维度都是用来刻画并行和分布式系统的：

- 规模可扩展性：指通过增加机器数量来获取更高的性能和更多的功能。“规模”指增加处理器、缓存、内存、存储器或 I/O 通道。判断规模可扩展性的最明显方式是简单地计算处理器安装数量。例如，IBM S2 在 1997 年由 512 个处理器扩展组成，但是在 2008 年，IBM BlueGene/L 系统由 65 000 个处理器扩展而成。
- 软件可扩展性：指升级操作系统或编译器，增加数学和工程库，移植新的应用软件，安装更多的用户友好的编程环境。一些软件升级可能不适合大型系统配置。新软件在更大系统上的测试和微调不是一个简单的工作。
- 应用可扩展性：指问题的规模扩展与机器的大小扩展相匹配。问题的规模影响数据集的大小或负载的增长。用户可以通过放大问题的规模，而不是增加机器的大小，来提高系统效率或成本效益。
- 技术可扩展性：指系统可以适应构建技术的变化，如 3.1 节中讨论的组件和网络技术等。何时扩展新技术的系统设计，必须考虑三个方面：时间、空间、异构性。（1）时间指生成可扩展性。当改变到新一代处理器时，我们必须考虑主板、电源、封装和冷却的影响等。根据以往的经验，大部分系统每三至五年就要升级自己的处理器。（2）空间是与封装和能量有关的问题。技术的可扩展性要求供应商之间的协调和可移植性。（3）异构性是指使用不同厂商的硬件组件或软件包。异构性可能限制可扩展性。

45

1.5.1.3 可扩展性和系统镜像数

在图 1-23 中，根据 2010 年部署的多重分布式操作系统镜像估测了其可扩展性能。可扩展的性能意味着系统可以通过增加更多的处理器或服务器、扩大物理节点的内存大小、扩展磁盘容

量或增加更多的 I/O 通道实现更快的速度。操作系统镜像数量由集群、网格、P2P 网络或云中被观测到的独立操作系统镜像数得出。SMP (Symmetric Multiprocessor, 对称多处理器) 和 NUMA (Nonuniform Memory Access, 非统一内存访问) 也被列入比较中。SMP 服务器有一个单系统镜像，可能是一个大型集群中的单个节点。根据 2010 年的标准，最大的共享内存 SMP 节点拥有有限的几百个处理器。SMP 系统的可扩展性主要是受所使用的封装和系统互连的限制。

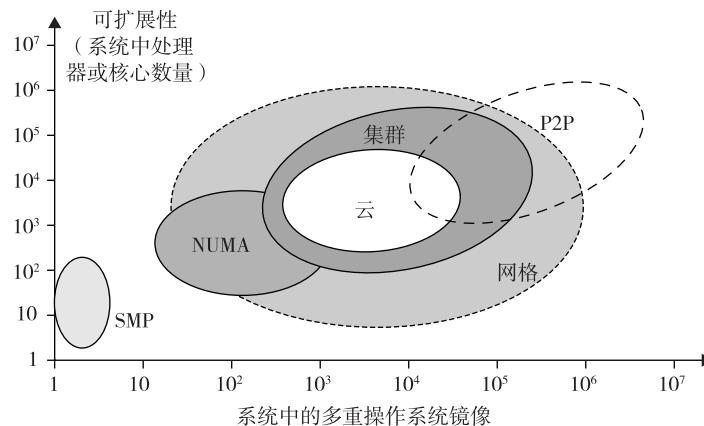


图 1-23 基于 2010 年技术的多重操作系统镜像与系统可扩展性

NUMA 机器通常由分布式、共享内存的 SMP 节点构成。NUMA 机器可以运行多个操作系统，并可以使用 MPI 库扩展到几千个处理器通信。例如，一个 NUMA 机可能有 2 048 个处理器运行 32 个 SMP 操作系统，即 32 个操作系统镜像运行在 2 048 个处理器的 NUMA 系统上。集群节点可以是 SMP 服务器或高端机器松耦合在一起。因此，集群可扩展性远远高于 NUMA 机。集群中操作系统镜像数是基于并发集群节点数的。云可以是一个虚拟的集群。截至 2010 年，最大的云能够扩展到几千个虚拟机。

请记住，许多集群节点是 SMP 或多核服务器，处理器或核心总数在集群系统中比集群上运行的操作系统镜像大 1 或 2 个数量级。网格节点可能是一个服务器集群、一台主机、一台超级计算机或 MPP。因此，大型网格结构中的操作系统镜像数可能比网格中的处理器数少成百或上千个。P2P 网络容易扩展到数百万独立对等节点，基本上是台式机。P2P 性能取决于公共网络的 QoS。低速的 P2P 网络、互联网云和计算机集群应在相同的网络水平。

1.5.1.4 Amdahl 定律

考虑单处理器执行给定程序共需 T 分钟时间。现在我们假定该方案已并行或划分到集群的许多处理节点上执行。假设必须串行执行的代码部分为 α ，称为串行瓶颈。因此， $(1 - \alpha)$ 的代码可以由 n 个处理器并行执行。总执行时间是 $\alpha T + (1 - \alpha)/n$ ，其中第一项是单处理器上串行执行的时间，第二项是 n 个处理节点并行执行的时间。

这里忽略了所有的系统或通信开销。下面加速分析中也没有包括 I/O 时间或异常处理时间。Amdahl 定律指出， n 处理器系统相对单一处理器的加速因子表示如下：

$$\text{加速比} = S = T / [\alpha T + (1 - \alpha)T/n] = 1 / [\alpha + (1 - \alpha)/n] \quad (1-1)$$

只有当串行瓶颈 α 降到零或代码完全并行时，才能达到最大加速比 n 。由于集群变得足够大，即 $n \rightarrow \infty$ ， S 接近 $1/\alpha$ ，令人惊讶的是，在加速与约束的上限，这个上限是独立于集群大小 n 的。串行瓶颈是不能并行化的代码部分。例如，如果 $\alpha = 0.25$ 或者 $1 - \alpha = 0.75$ ，即使使用数百个处理器，最大加速比也为 4。Amdahl 定律告诉我们，应该使串行瓶颈尽可能小。在这种情况下，仅增加集群的规模可能不会得到好的加速。

1.5.1.5 固定负载问题

在 Amdahl 定律中，假定串行和并行执行固定问题规模或数据集的程序需要等量负载。这被 Hwang 和 Xu^[14]称为固定负载加速比。 n 个处理器执行一个固定负载，并行处理的系统效率定义如下：

$$E = S/n = 1/[\alpha n + 1 - \alpha] \quad (1-2)$$

系统效率通常非常低，尤其是当集群规模非常大时。在 $n = 256$ 个节点的集群上执行上述程序，显然，极低效率 $E = 1/[0.25 \times 256 + 0.75] = 1.5\%$ 。这是因为只有几个处理器（比方说，4）在工作，而大多数节点空转。

1.5.1.6 Gustafson 定律

当使用一个大规模集群时，为了实现更高的效率，我们必须考虑扩大问题规模来匹配集群的能力。这促使 John Gustafson (1988) 提出了下面的加速比定律，简称为扩展负载加速比^[14]。设 W 是给定程序的负载。当使用 n 处理器系统时，用户将负载扩展为 $W' = \alpha W + (1 - \alpha) nW$ 。请注意，只有并行部分负载是在第二项中扩展 n 倍。这个扩展负载 W' 基本上是单个处理器上的串行执行时间。并行扩展负载 W' 在 n 个处理器上的执行时间由扩展负载加速比定义如下：

$$S' = W'/W = [\alpha W + (1 - \alpha) nW] / W = \alpha + (1 - \alpha) n \quad (1-3)$$

这个加速比称为 Gustafson 定律。通过固定并行处理时间在级别 W ，可以得到如下的效率表达式：

$$E' = S'/n = \alpha/n + (1 - \alpha) \quad (1-4)$$

对于扩展负载，使用 256 个节点的集群可以使前面程序使用效率提高为 $E' = 0.25/256 + 0.75 = 0.751$ 。在不同的负载条件下，应该灵活选用 Amdahl 定律和 Gustafson 定律。对于固定负载，应采用 Amdahl 定律。为了解决扩展规模的问题，应采用 Gustafson 定律。

1.5.2 容错和系统可用性

除了性能外，系统可用性和应用的灵活性是分布式计算系统另外两个重要的设计目标。

系统可用性

HA（高可用性）是所有集群、网格、P2P 网络和云系统所期望的。如果系统有一个长的平均故障时间（Mean Time To Failure, MTTF）和短的平均修复时间（Mean Time To Repair, MTTR），那么这个系统是高度可用的。系统可用性形式化定义如下：

$$\text{系统可用性} = \text{MTTF}/(\text{MTTF} + \text{MTTR}) \quad (1-5)$$

影响系统可用性的因素有很多。所有的硬件、软件和网络组件都可能会出错。影响整个系统运行的故障称为单点故障。经验法则是一个可靠的计算系统设计应没有单点故障。增加硬件冗余，提高元件的可靠性和设计可测性，将有助于提高系统的可用性和可靠性。在图 1-24 中，预测了通过增加处理器核心数来扩大系统规模对系统可用性的影响。

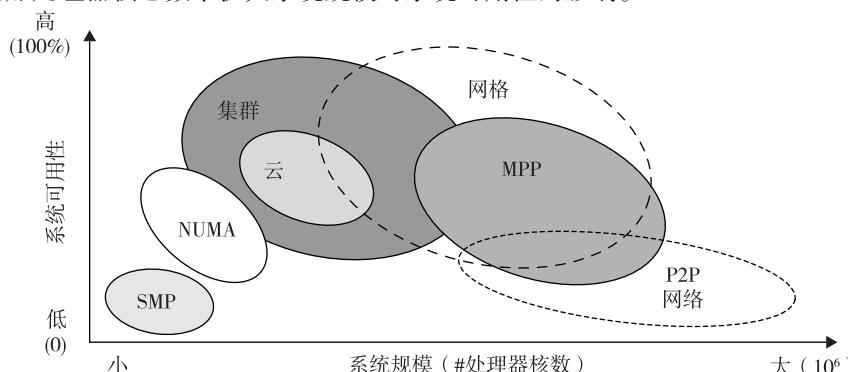


图 1-24 通过 2010 年常规配置的系统规模估计系统可用性

在一般情况下，随着分布式系统规模的增加，系统可用性会因更高的故障几率和隔离故障的难度而降低。SMP 和 MPP 在单操作系统下的集中式资源是非常脆弱的。由于使用多操作系统，NUMA 机器可用性大大提高。大多数集群是通过故障转移功能来获得高可用性。与此同时，私有云由虚拟化数据中心创造出来；因此，云有一个类似主机集群的可用性预测。网格作为层次化的集群的集群是可视化的。网格因故障隔离而有更高的可用性。因此，集群、云和网格随着系统规模的增加，可用性降低。一个 P2P 文件共享网络具有最高的客户机聚合。然而，它独立地运行，可用性很低，甚至很多对等节点退出或同时失败。

1.5.3 网络威胁与数据完整性

要在当今的数字时代使用集群、网格、P2P 网络和云，就要对其安全和版权进行保护。本节将介绍分布式或云计算系统中的系统弱点、网络威胁、防范措施和版权保护。

1.5.3.1 系统与网络的威胁

网络病毒的广泛传播影响到许多用户。这些病毒通过破坏路由器和服务器进行传播，使商业、政府和服务蒙受数十亿美元的损失。图 1-25 归纳了各种病毒袭击方式及其可能对用户造成的影响。如图 1-25 所示，信息泄露对保密性造成损失。用户变更、木马和欺诈服务会破坏数据完整性。拒绝服务（Denial of Service, DoS）会破坏系统运行和互联网连接。不正当的认证会导致非法使用。

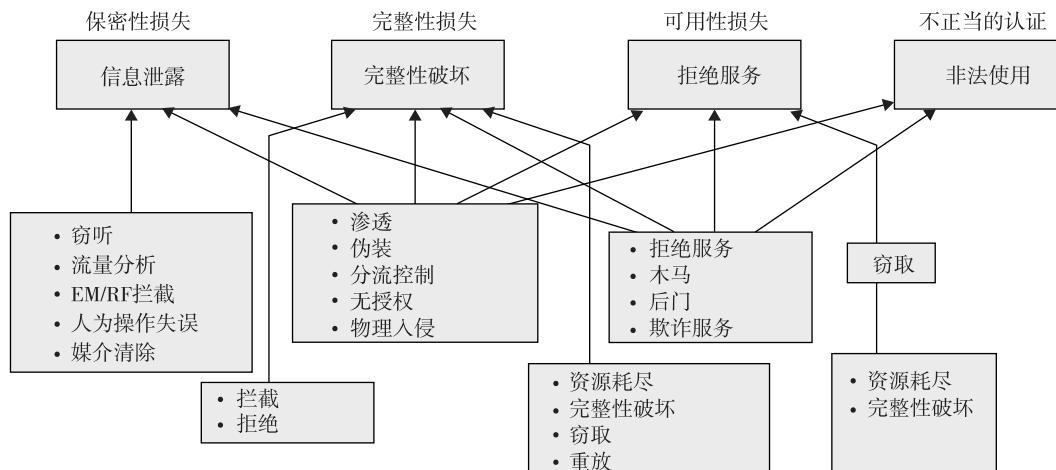


图 1-25 对计算机的各种系统袭击和网络威胁及造成的 4 种损失

缺少认证或授权会导致袭击者非法使用计算资源。开放式资源（如数据中心、P2P 网络，以及网格和云基础设施）将会成为下一个袭击目标。所以用户需要保护集群、网格、云和 P2P 系统。否则，用户不应该使用或信任它们进行外包工作。系统遭到恶意入侵可能会破坏重要的主机，以及网络和存储资源。路由、网关及分布式主机出现网络异常会阻碍这些公共资源计算服务的认可度。

1.5.3.2 安全责任

保密性、完整性和可用性是多数网络服务提供商和云用户经常考虑到的安全需求。提供商对云用户安全控制的责任按 SaaS、PaaS 和 IaaS 的顺序逐渐减小。总之，SaaS 模式依赖于云提供商来保证所有安全功能的运行。另一种极端情况则是 IaaS 模式，要求用户承担几乎所有的安全运行，而将可用性交由提供商。PaaS 模式依靠提供商维护数据的完整性和可用性，但保密性和隐私控制由用户承担。

1.5.3.3 版权保护

共谋剽窃是 P2P 网络领域内侵犯知识产权的主要来源。付费用户（共谋者）会与非付费用

户（剽窃者）非法共享内容受版权保护的文件。在线剽窃已经妨碍了以商业内容传输为目的的开放式P2P网络的使用。开发一种主动内容中毒机制可以阻止共谋者和剽窃者在P2P文件共享过程中侵犯版权。通过基于签名的身份识别和时间戳令牌，剽窃者会被及时发现。这一机制会阻止共谋剽窃的发生，而并不妨碍合法的P2P用户。第4章和第7章将讨论网格和云的安全、P2P信誉系统，以及版权保护。

1.5.3.4 系统防御技术

在之前已经出现过三代网络防御技术。第一代技术设计了阻止或避免入侵的工具，这些工具通常将自己显示为访问控制令牌、密文系统等。但入侵者仍然可以渗入安全系统，因为在安全调度进程中常有漏洞。第二代技术可以很快发现入侵并进行修补。这些技术包括防火墙、入侵检测系统（IDSe）⁴⁸、PKI服务、信誉系统等。第三代技术对入侵能做出更多智能回应。⁵⁰

1.5.3.5 数据保护基础设施

安全基础设施是保护Web服务和云服务的必要条件。在用户层，用户需要履行信任协议和对所有用户的信誉集合。在应用终端，我们需要建立安全预警以遏制蠕虫，检测病毒、蠕虫的入侵，以及分布式拒绝服务（DDoS）袭击。我们还需要阻止在线剽窃和数字内容的版权侵犯。在第4章，我们将研究信誉系统对云系统和数据中心的保护。根据三种云服务模式，安全责任被分配给云提供商和用户。提供商完全负责平台的可用性。IaaS用户更多地负责保密性，IaaS提供商更多地负责数据完整性。在PaaS和SaaS服务中，提供商和用户会平等地承担保护数据完整性和保密性的责任。

1.5.4 分布式计算中的节能

传统并行和分布式计算系统的首要目标是高性能和高吞吐量，同时还需要考虑一些性能可靠性（如容错性和安全性）。然而这些系统最近面临新的挑战，包括节能、负载和资源外包。这些新兴问题的重要性不仅在于它们本身，而且一般还关系到大型计算系统的稳定性。本节将探讨服务器和HPC系统中的能源消耗问题，这一问题也称为“分布式电能管理”（Distributed Power Management, DPM）。

对数据中心的保护需要集成的解决方案。并行和分布式计算系统中的能源消耗引起了资金、环境和系统性能方面的多种问题。例如，地球模拟器和每秒千万亿次浮点运算是以12兆瓦和100兆瓦为能源峰值的系统。如果按每兆瓦100美元计算，那么它们在峰值期间运行的能源成本是每小时1200美元和10000美元；这超出了许多（潜在的）系统运营者的预算承受范围。除了能源成本外，冷却是另一个不得不提的问题，因为高温会对电子元件造成负面影响。电路温度的上升不但使线路超出正常范围，还缩短了元件的寿命。

1.5.4.1 空转服务器的能源消耗

若要运行一个数据中心，公司不得不花费大量金钱来购买硬件、软件、运营支持，以及每年的能源。因此，公司应该仔细考虑他们所安装的数据中心（更确切地说是调配的资源量）是否在一个合理的水平上，特别是在效用方面。据估计，以前在公司里平均每天有六分之一（15%）的全天候服务器在运行时并没有被利用（即处于闲置状态）。这说明在全世界4400万台服务器中，有470万台服务器没有做任何有用的工作。

关闭这些服务器将节省下一大笔钱——全球仅能源成本就有38亿美元，运行闲置服务器的全部成本是247亿美元，这是1E公司与节能同盟（Alliance to Save Energy, ASE）合作的调查结果。这些被浪费的能源相当于每年排放1180万吨二氧化碳，相当于210万辆小汽车排放的一氧化碳的污染量。在美国，这相当于317万吨二氧化碳或者580678辆小汽车的排放量。因此，IT部门的第一步是分析他们的服务器，以找出闲置的和没有被充分利用的服务器。

48

50

51

1.5.4.2 运行服务器的节能

除了节省未使用和未充分利用的服务器的能源外，还需要用合适的技术减少分布式系统中运行服务器的能源消耗，同时要把对它们性能的影响降到最低。分布式计算平台的能源管理问题可以分为4层（见图1-26）：应用层、中间件层、资源层和网络层。

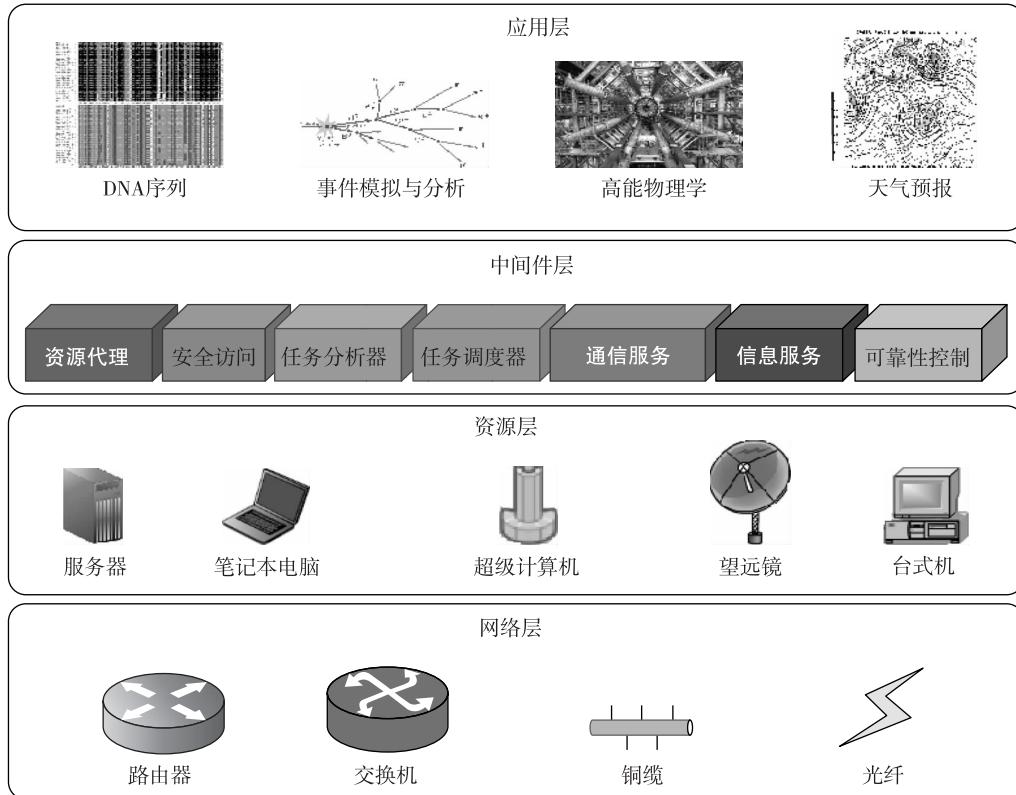


图1-26 分布式运算系统的4个操作层

注：由悉尼大学的Zomaya、Rivandi和Lee提供^[33]。

1.5.4.3 应用层

直到现在，大部分科学、商业、工程和金融领域的用户应用都倾向于提升系统的速度或质量。52通过引入能量感知应用，在不损害性能的前提下设计复杂的多层次、多领域能源管理应用成为挑战。面向这个目标的第一步是要探索出性能和能源消耗之间的关系。事实上，应用程序的能源消耗很大程度上取决于需要执行的应用和存储单元（或内存）的事务数量。这两个因素（计算和存储）是相关的，影响着完成时间。

1.5.4.4 中间件层

中间件层充当了应用层和资源层之间的桥梁。这层提供了资源代理、通信服务、任务分析器、任务调度器、安全访问、可靠性控制和信息服务能力。它也负责采用高效节能技术，特别是任务调度。直到最近，调度旨在最小化完工时间，即一组任务的执行时间。分布式计算系统需要一种新的代价函数涵盖完工时间和能源消耗。

1.5.4.5 资源层

资源层由包括计算节点和存储单元的广泛资源组成。这层通常与硬件设备和操作系统进行交互。因此，它负责控制分布式计算系统中的所有分布式资源。最近，制定了一些可以更高效管理硬件和操作系统电源的机制。它们中大多数是硬件方法，尤其是针对处理器的。

动态电能管理（Dynamic Power Management, DPM）和动态电压频率缩放（Dynamic Voltage-Frequency Scaling, DVFS）是两种纳入计算机硬件系统中的新方法^[21]。在 DPM 中，硬件设备（如 CPU）可以从空闲模式切换到一个或多个低电能模式。在 DVFS 技术中，节能是基于 CMOS 功耗与频率和供应电压平方有直接关系这一事实。执行时间和功耗是通过在不同频率和电压之间进行切换来控制^[31]。

1.5.4.6 网络层

路由、传输数据包和确保资源层的网络服务是分布式计算系统中网络层的主要责任。同样，构建节能网络的主要挑战是决定如何来度量、预测和建立一个能耗与性能之间的平衡。节能网络设计的两个主要挑战是：

- 模型应该全面地表达网络，比如，应充分考虑时间、空间和能源之间的相互作用。
- 需要探索新的节能路由算法。需要新的节能协议对抗网络攻击。

由于信息资源推动经济和社会发展，数据中心作为信息存储和处理以及服务提供所在，正变得越来越重要。数据中心已成为核心基础设施，就像电网和运输系统。传统的数据中心正遭受着高建设和运营成本、复杂的资源管理、低可用性、低安全性和低可靠性，以及巨大的能源消耗。在下一代数据中心设计上采用新技术是非常必要的。这将在第 4 章进行详细讨论。[53]

1.5.4.7 DVFS 节能方法

DVFS 方法能够利用因任务交互而招致的松弛时间（空闲时间）。具体来说，利用与任务相关的松弛时间以一个低电压、频率执行任务。CMOS 电路中能耗和电压、频率之间的关系如下：

$$\begin{cases} E = C_{\text{eff}} f v^2 t \\ f = K \frac{(v - v_t)^2}{v} \end{cases} \quad (1-6)$$

其中 v 、 C_{eff} 、 K 和 v_t 分别代表电压、电路交换能力、技术相关因素和阈值电压，参数 t 是任务在时钟频率 f 下的执行时间。通过降低电压和频率，设备能耗也能够减少。

例 1.2 分布式电源管理中的节能

图 1-27 所示为 DVFS 方法。想法就是在负载松弛时间降低频率和电压。低功率模式之间的过渡延迟非常小。因此，可以通过在运行模式之间进行切换来节约能源。在低功耗模式之间切换会影响性能。存储单元必须和计算节点交互来平衡功耗。根据 Ge、Feng、Cameron^[21]，存储设备约占数据中心能源消耗总量的 27%。由于存储需求年增长 60%，这个数字迅速增加，问题变得更加恶化。■

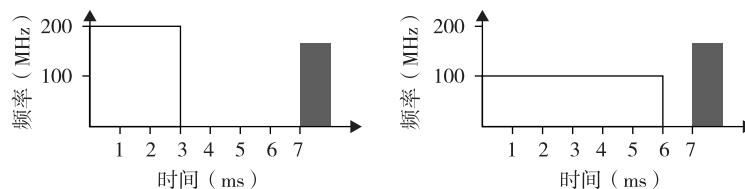


图 1-27 通过在松弛时间降低频率或电压，DVFS 技术（右）节约的能量与传统方式（左）的比较

[54]

1.6 参考文献和习题

在过去的 40 年中，并行处理和分布式计算已经成为研究和开发的热门话题。并行计算的早期工作可以在一些经典书籍中找到^[14, 17, 27, 28]。最近的分布式系统研究工作可以在文献[8, 13, 20, 22]中找到。集群计算在文献[2, 10, 28, 38]有涉及，网格计算在文献[6, 18, 42, 47, 51]中。文献[6, 34, 46]介绍了 P2P 网络。多核 CPU 和众核 GPU 处理器在文献[15, 32, 36]中

进行了讨论。Top500 超级计算机的相关资料可以在文献[50]中找到。

文献[4, 19, 26]是对数据中心的介绍，文献[24, 26]介绍了最新的计算机体系结构。文献[1, 9, 11, 18, 29, 30, 39, 44]是云计算相关研究。由 Buyya、Broberg 和 Goscinski 编辑成卷的有关云计算研究[11]是云计算研究的一个很好的资源。Chou 的书^[12]强调云服务的商业模式。文献[40 ~ 44]介绍了虚拟化技术。Bell、Gray 和 Szalay^[5]、Foster 等人^[18]，以及 Hey^[25]的文章关注有关数据密集型网格和云计算的关键问题。大规模数据并行和编程包括在文献[14, 32]中。

分布式算法和 MPI 编程的研究在文献[3, 12, 15, 22, 28]中有介绍。分布式操作系统和软件工具在文献[3, 7, 13, 46]中涉及。节能和能量管理在文献[21, 31, 52]中被研究。文献[45, 48]是对物联网的研究。Hwang 和 Li 的工作^[30]为应对云安全和数据保护问题提出了建议。在随后的章节，将提供其他参考文献。下面的列表中重点列出涉及并行、集群、网格、P2P 系统、云和分布式系统领域最新进展的一些国际会议、杂志和期刊：

- **IEEE 及相关会议刊物：**互联网计算，TPDS（并行与分布式系统汇刊），TC（计算机汇刊），TON（网络汇刊），ICDCS（分布式计算系统国际会议），IPDPS（国际并行与分布处理研讨会），INFOCOM，GLOBECOM，CCGrid（集群、云和网格），P2P 计算，HPDC（高性能分布式计算），CloudCom（云计算技术和科学国际会议），ISCA（计算机体系结构国际研讨会），HPCA（高性能计算机体系结构），计算机杂志，TDSC（可靠和安全计算汇刊），TKDE（知识与数据工程汇刊），HPCC（高性能计算和通信），ICPADS（并行与分布式应用和系统国际会议），NAS（网络、体系结构和存储）和 GPC（网格和普适计算）。
- **ACM、互联网协会、IFIP 和其他相关出版物：**超级计算会议，ACM 计算系统汇刊，USENIX 技术会议，JPDC（并行和分布式计算期刊），云计算期刊，分布式计算期刊，集群计算期刊，下一代计算机系统，网格计算期刊，平行计算期刊，平行处理国际会议（ICPP），欧洲并行计算会议（EuroPAR），并发：实践与经验（Wiley），NPC（IFIP 网络和并行计算），PDCS（ISCA 并行与分布式计算机系统）。

55

致谢

本章主要由黄铠教授撰写，Geoffrey Fox 和 Albert Zomaya 分别参与了 1.4.1 节和 1.5.4 节的撰写。南加州大学的楼肖松和陈理中对图 1-4 和图 1-10 的绘制提供了帮助。悉尼大学的 Nikzad Rivandi 和 YoungChoon Lee 参与了 1.5.4 节的写作。Jack Dongarra 审校了本章内容。

本章由清华大学武永卫教授负责翻译。

参考文献

- [1] Amazon EC2 and S3, Elastic Compute Cloud (EC2) and Simple Scalable Storage (S3). http://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud and http://spatten_presentations.s3.amazonaws.com/s3-on-rails.pdf.
- [2] M. Baker, R. Buyya, Cluster computing at a glance, in: R. Buyya (Ed.), High-Performance Cluster Computing, Architecture and Systems, vol. 1, Prentice-Hall, Upper Saddle River, NJ, 1999, pp. 3–47, Chapter 1.
- [3] A. Barak, A. Shiloh, The MOSIX Management System for Linux Clusters, Multi-Clusters, CPU Clusters, and Clouds, White paper. www.MOSIX.org/txt_pub.html, 2010.
- [4] L. Barroso, U. Holzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Morgan & Claypool Publishers, 2009.

- [5] G. Bell, J. Gray, A. Szalay, Petascale computational systems: balanced cyberstructure in a data-centric World, *IEEE Comput. Mag.* (2006).
- [6] F. Berman, G. Fox, T. Hey (Eds.), *Grid Computing*, Wiley, 2003.
- [7] M. Bever, et al., Distributed systems, OSF DCE, and beyond, in: A. Schill (Ed.), *DCE-The OSF Distributed Computing Environment*, Springer-Verlag, 1993, pp. 1–20.
- [8] K. Birman, *Reliable Distributed Systems: Technologies, Web Services, and Applications*, Springer-Verlag, 2005.
- [9] G. Boss, et al., Cloud Computing—The BlueCloud Project. www.ibm.com/developerworks/websphere/zones/hipods/, October 2007.
- [10] R. Buyya (Ed.), *High-Performance Cluster Computing*, Vol. 1 and 2, Prentice-Hall, Englewood Cliffs, NJ, 1999.
- [11] R. Buyya, J. Broberg, A. Goscinski (Eds.), *Cloud Computing: Principles and Paradigms*, Wiley, 2011.
- [12] T. Chou, *Introduction to Cloud Computing: Business and Technology*. Lecture Notes at Stanford University and Tsinghua University, Active Book Press, 2010.
- [13] G. Coulouris, J. Dollimore, T. Kindberg, *Distributed Systems: Concepts and Design*, Wesley, 2005.
- [14] D. Culler, J. Singh, A. Gupta, *Parallel Computer Architecture*, Kaufmann Publishers, 1999.
- [15] B. Dally, GPU Computing to Exascale and Beyond, Keynote Address at ACM Supercomputing Conference, November 2010.
- [16] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: Proceedings of OSDI 2004. Also, *Communication of ACM*, Vol. 51, 2008, pp. 107–113.
- [17] J. Dongarra, et al. (Eds.), *Source Book of Parallel Computing*, Morgan Kaufmann, San Francisco, 2003.
- [18] I. Foster, Y. Zhao, J. Raicu, S. Lu, *Cloud Computing and Grid Computing 360-Degree Compared*, Grid Computing Environments Workshop, 12–16 November 2008.
- [19] D. Gannon, The Client+Cloud: Changing the Paradigm for Scientific Research, Keynote Address, IEEE CloudCom2010, Indianapolis, 2 November 2010.
- [20] V.K. Garg, *Elements of Distributed Computing*. Wiley-IEEE Press, 2002.
- [21] R. Ge, X. Feng, K. Cameron, Performance Constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters, in: Proceedings Supercomputing Conf., Washington, DC, 2005.
- [22] S. Ghosh, *Distributed Systems—An Algorithmic Approach*, Chapman & Hall/CRC, 2007.
- [23] B. He, W. Fang, Q. Luo, N. Govindaraju, T. Wang, Mars: A MapReduce Framework on Graphics Processor, ACM PACT’08, Toronto, Canada, 25–29 October 2008.
- [24] J. Hennessy, D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2007.
- [25] T. Hey, et al., The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft Research, 2009.
- [26] M.D. Hill, et al., *The Data Center as a Computer*, Morgan & Claypool Publishers, 2009.
- [27] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programming*, McGraw-Hill, 1993.
- [28] K. Hwang, Z. Xu, *Scalable Parallel Computing*, McGraw-Hill, 1998.
- [29] K. Hwang, S. Kulkarni, Y. Hu, Cloud Security with Virtualized Defense and Reputation-based Trust Management, in: IEEE Conference on Dependable, Autonomous, and Secure Computing (DAC-2009), Chengdu, China, 14 December 2009.
- [30] K. Hwang, D. Li, Trusted Cloud Computing with Secure Resources and Data Coloring, in: IEEE Internet Computing, Special Issue on Trust and Reputation Management, September 2010, pp. 14–22.
- [31] Kelton Research, *Server Energy & Efficiency Report*. www.1e.com/EnergyCampaign/downloads/Server_Energy_and_Efficiency_Report_2009.pdf, September 2009.
- [32] D. Kirk, W. Hwu, *Programming Massively Processors: A Hands-on Approach*, Morgan Kaufmann, 2010.
- [33] Y.C. Lee, A.Y. Zomaya, A Novel State Transition Method for Metaheuristic-Based Scheduling in Heterogeneous Computing Systems, in: IEEE Transactions on Parallel and Distributed Systems, September 2008.
- [34] Z.Y. Li, G. Xie, K. Hwang, Z.C. Li, Churn-Resilient Protocol for Massive Data Dissemination in P2P Networks, in: *IEEE Trans. Parallel and Distributed Systems*, May 2011.
- [35] X. Lou, K. Hwang, Collusive Piracy Prevention in P2P Content Delivery Networks, in: *IEEE Trans. on Computers*, July, 2009, pp. 970–983.
- [36] NVIDIA Corp. Fermi: NVIDIA’s Next-Generation CUDA Compute Architecture, White paper, 2009.
- [37] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, SIAM, 2000.
- [38] G.F. Pfister, *In Search of Clusters*, Second ed., Prentice-Hall, 2001.
- [39] J. Qiu, T. Gunaratne, J. Ekanayake, J. Choi, S. Bae, H. Li, et al., Hybrid Cloud and Cluster Computing Paradigms for Life Science Applications, in: 11th Annual Bioinformatics Open Source Conference BOSC 2010, 9–10 July 2010.

- [40] M. Rosenblum, T. Garfinkel, Virtual machine monitors: current technology and future trends, *IEEE Computer* (May) (2005) 39–47.
- [41] M. Rosenblum, Recent Advances in Virtual Machines and Operating Systems, Keynote Address, ACM ASPLOS 2006.
- [42] J. Smith, R. Nair, *Virtual Machines*, Morgan Kaufmann, 2005.
- [43] B. Sotomayor, R. Montero, I. Foster, Virtual Infrastructure Management in Private and Hybrid Clouds, *IEEE Internet Computing*, September 2009.
- [44] SRI. The Internet of Things, in: Disruptive Technologies: Global Trends 2025, www.dni.gov/nic/PDF_GIF_Confreports/disruptivetech/appendix_F.pdf, 2010.
- [45] A. Tanenbaum, *Distributed Operating Systems*, Prentice-Hall, 1995.
- [46] I. Taylor, *From P2P to Web Services and Grids*, Springer-Verlag, London, 2005.
- [47] Twister, Open Source Software for Iterative MapReduce, <http://www.iterativemapreduce.org/>.
- [48] Wikipedia. Internet of Things, http://en.wikipedia.org/wiki/Internet_of_Things, June 2010.
- [49] Wikipedia. CUDA, <http://en.wikipedia.org/wiki/CUDA>, March 2011.
- [50] Wikipedia. TOP500, <http://en.wikipedia.org/wiki/TOP500>, February 2011.
- [51] Y. Wu, K. Hwang, Y. Yuan, W. Zheng, Adaptive Workload Prediction of Grid Performance in Confidence Windows, in: *IEEE Trans. on Parallel and Distributed Systems*, July 2010.
- [52] Z. Zong, Energy-Efficient Resource Management for High-Performance Computing Platforms, Ph.D. dissertation, Auburn University, 9 August 2008.

习题

- 1.1 简要地定义以下在计算机体系结构、并行处理、分布式计算、互联网技术、信息服务领域代表最近相关进展的基本技术：
 - a. 高性能计算 (HPC) 系统
 - b. 高吞吐量计算 (HTC) 系统
 - c. 对等 (P2P) 网络
 - d. 计算机集群与计算网格
 - e. 面向服务的体系结构 (SOA)
 - f. 普适计算与互联网计算
 - g. 虚拟机和虚拟基础设施
 - h. 公有云与私有云
 - i. 射频识别 (RFID)
 - j. 全球定位系统 (GPS)
 - k. 传感器网络
 - l. 物联网 (IoT)
 - m. 信息物理系统 (CPS)
- 1.2 在下面两个问题中选出唯一的正确答案：
 1. 2009 年最快的计算机系统排名 Top500 中，哪个体系结构占主宰地位？
 - a. 对称共享内存多处理器系统
 - b. 集中式大规模并行处理器 (MPP) 系统
 - c. 协同计算机集群
 2. 在由服务器集群形成的云中，所有服务器必须采用下面哪种方式？
 - a. 所有云机器必须构建在物理机上
 - b. 所有云机器必须构建在虚拟机上
 - c. 云机器可以是物理机也可以是虚拟机
- 1.3 越来越多的工业和商业组织采用云系统。关于云计算，回答以下问题：
 - a. 列出并描述云计算系统的主要特点。
 - b. 讨论云计算系统中的关键技术。

c. 讨论云服务提供商最大化收入的不同方式。

- 1.4 将左侧术语缩写和系统模型同右侧的描述匹配起来，将描述的标号填入术语前的空格中。

<input type="text"/> Globus	(a) 由 Apache 倡导和维护的用于编写和运行面向大量分布式数据应用程序的可扩展软件平台
<input type="text"/> BitTorrent	(b) 通过集中式目录服务器进行 MP3 音乐分发的 P2P 网络
<input type="text"/> Gnutella	(c) 谷歌用于超大数据集分布式映射和压缩的编程模型与相关实现
<input type="text"/> EC2	(d) 由 USC/ISI 和阿贡国家实验室联合开发的用于网格资源管理和作业调度的中间件库
<input type="text"/> TeraGrid	(e) 谷歌用于管理可能扩展到超大规模的结构化数据的分布式存储程序
<input type="text"/> EGEE	(f) 使用多文件索引的 P2P 文件共享网络
<input type="text"/> Hadoop	(g) 计算机集群节点容错和主机故障恢复的关键设计目标
<input type="text"/> SETI@ home	(h) 作为开放网格标准的服务体系结构说明
<input type="text"/> Napster	(i) 一个允许网络应用开发者有效获取云资源的弹性且灵活的计算环境
<input type="text"/> BigTable	(j) 用于在寻找地外文明中进行分布式信号处理的超过 300 万台台式计算机的 P2P 网格

- 1.5 考虑 4 个异构内核标记为 A、B、C 和 D 的多核处理器。假设核 A 和核 D 有相同的速度。核 B 运行速度比核 C 快 2 倍，核 C 运行速度比核 A 快 3 倍。假设所有 4 个内核同时执行下面的应用程序且在所有核运行过程中没有缓存未命中情况。假设应用程序需要计算数组中 256 个元素的平方。假设核 A 或核 D 在 1 个单元时间能计算 1 个元素的平方。因此，核 B 需要 1/2 个单元时间，核 C 需要 1/3 个单元时间计算一个元素的平方。给出 4 个核的分工：

核 A	32 个元素
核 B	128 个元素
核 C	64 个元素
核 D	32 个元素

- a. 计算使用 4 核处理器并行计算 256 个元素平方的总运行时间（单元时间）。4 个核速度不同。一些快的核完成任务后可能会空闲下来，而其他核仍进行计算直到所有平方算完。
 - b. 计算处理器利用率，所有核工作（非空闲）总时间除以执行上面应用程序时处理器中所有核总运行时间。
- 1.6 考虑在 SPMD（单程序多数据流）模式的 n 台相同 Linux 服务器组成的集群上并行执行一个使用 MPI 代码的 C 程序。SPMD 模式意味着相同的 MPI 程序同时运行在所有服务器上但处理相同负载的不同数据集。假设 25% 的程序执行是 MPI 命令的执行。为简单起见，假设所有 MPI 命令消耗相同的执行时间。运用 Amdahl 定律回答下列问题：
- a. 给定 MPI 程序在 4 服务器集群上的总执行时间是 T 分钟，在 256 个服务器的集群上执行相同 MPI 程序的加速比是多少？假定程序执行是无死锁的并且忽略计算中所有其他运行时开销。
 - b. 假设所有 MPI 命令现在通过用户空间消息句柄采用动态消息效率提升了 2 倍。提升使所有 MPI 命令的运行时间减少了一半。安装了这种 MPI 改进的 256 个服务器集群，相对于原来加速比是多少？
- 1.7 考虑一个计算两个大规模的 $N \times N$ 矩阵乘法的程序，其中 N 是矩阵大小。单服务器上串行乘法执行时间是 $T_1 = cN^3$ 分钟，其中 c 是由所用服务器决定的常量。一个 MPI 并行程序在一个 n 服务器集群系统完成执行需要 $T_n = cN^3/n + dN^2/n^{0.5}$ 分钟，其中 d 是一个由所使用 MPI 版本决定的常量。假定程序的串行瓶颈是 0 ($\alpha=0$)。 T_n 中的第二项表示 n 个服务器总的消息传递开销。

对于给定集群配置： $n = 64$ 个服务器， $c = 0.8$ ， $d = 0.1$ 。回答如下问题。a 和 b 部分有一个相应于矩阵大小 $N = 15\,000$ 的固定负载。c 和 d 部分有一个相应于矩阵大小 $N' = n^{1/3}N = 64^{1/3} \times 15\,000 = 4 \times 15\,000 = 60\,000$ 的扩展负载。假设用相同的集群配置来处理两个负载。那么，系统参数 n 、 c 和 d 保持不变。运行扩展负载，开销也会随着矩阵 N' 的增大而增长。

- a. 使用 Amdahl 定律，计算 n 服务器集群相对单服务器的加速比。

- b. a 部分使用的集群系统效率是多少?
 - c. 使用 Gustafson 定律计算相同集群配置下执行扩展的 $N' \times N'$ 矩阵计算的加速比。
 - d. 计算在 64 处理器集群上运行 c 部分扩展负载的效率。
 - e. 比较以上运算加速比和效率结果并评价它们的影响。
- 1.8 比较传统计算集群/网格和近年来兴起的计算云之间的相似和不同。考虑下面列出的所有技术和经济因素。针对这些年构建的实例系统或平台回答下列问题，并讨论两个计算范式在将来可能的融合点。
- a. 硬件、软件和网络支持
 - b. 资源分配和供给方法
 - c. 基础设施管理和保护
 - d. 计算服务效用支持
 - e. 操作和耗费模型应用
- 1.9 针对 PC 和 HPC 系统，回答下列问题：
- a. 解释为什么个人计算机和高性能计算近期的变革超过过去 30 年的变革。
 - b. 讨论处理器架构的破坏性改变的缺陷。为什么内存墙是性能可扩展的主要问题？
 - c. 解释为什么 x86 处理器仍然主宰着 PC 和 HPC 市场。
- 1.10 多核和众核处理器已经广泛应用于台式计算机和 HPC 系统中。针对先进的处理器、内存设备和系统互连设备回答下列问题：
- a. 多核 CPU 和 GPU 在体系结构和使用方面的不同之处是什么？
 - b. 解释为什么并行编程模型无法匹配处理器技术的进步。
 - c. 针对核心扩展与有效编程和使用多核的不匹配之间的问题给出建议，并面对似是而非的解决方法捍卫你的观点。
 - d. 解释为什么闪存 SSD 在一些 HPC 和 HTC 应用中可以得到更好的加速比。
 - e. 说明 InfiniBand 和以太网将继续主宰 HPC 市场这个预测是合理的。
- 1.11 在图 1-7 中，你了解了现代处理器的 5 个分类。表 1-9 中刻画了设计这些处理器的 5 个微体系统结构。评价它们的优缺点并给出每个处理器分类中两个商业处理器的例子。假设一个单核超标量处理器和三个多线程处理器，上述处理器分类是一个多核 CMP 并且每个核心处理一个线程。

表 1-9 现代处理器 5 个微体系统结构比较

处理器微体系统结构	体系结构特点	优点/缺点	典型处理器
单线程超标量			
细粒度多线程			
粗粒度多线程			
并发多线程 (SMT)			
片上多核多处理器 (CMP)			

- 1.12 讨论下列领域的主要优点和缺点：
- a. 为什么云计算系统中虚拟机和虚拟集群备受推崇？
 - b. 建立合算的虚拟云系统需要哪些突破？
 - c. 云平台对于 HPC 和 HTC 在工业界的未来有什么影响？
- 1.13 描述下列三个云计算模型：
- a. 什么是 IaaS（基础设施即服务）云？给出一个例子。
 - b. 什么是 PaaS（平台即服务）云？给出一个例子。
 - c. 什么是 SaaS（软件即服务）云？给出一个例子。
- 1.14 简要解释下面的云计算服务。在每个服务类别下给出两个云提供商的公司名。
- a. 应用云服务
 - b. 平台云服务

- c. 计算和存储服务
 - d. 分配云服务
 - e. 网络云服务
- 1.15 简要解释下列分布式计算系统中网络威胁和安全防御相关的术语：
- a. 拒绝服务（DoS）
 - b. 木马
 - c. 网络蠕虫
 - d. 服务欺诈
 - e. 授权
 - f. 认证
 - g. 数据完整性
 - h. 保密性
- 1.16 针对绿色信息技术和分布式系统节能，简要回答下列问题：
- a. 为什么数据中心运行中能量消耗是关键问题？
 - b. 动态电压频率缩放（DVFS）技术的构成？
 - c. 基于现有绿色IT研究的进展进行深度研究，并写一篇关于数据中心设计和云服务应用的报告。
- 1.17 比较GPU和CPU芯片各自的优势和弱点。特别地，讨论节能、可编程性和性能之间的权衡。并比较各种MPP架构在处理器选取、性能目标、效率和封装的约束。
- 1.18 比较三种分布式操作系统：Amoeba、DCE和MOSIX。调研它们最近的进展和对集群、网格和云中应用的影响。讨论每个系统在商业或实验性分布式应用中的适应性。并讨论每个系统的局限以及它们为什么不如商业系统成功。

第 2 章 |

Distributed and Cloud Computing, 1E

可扩展并行计算集群

计算机集群化（clustering）在科学与商业应用中实现了可扩展并行计算和分布式计算。本章主要学习建立集群结构的大规模并行处理机。我们专注于硬件、软件、中间件、操作系统支持的设计原则和评估，这些用于实现集群的可扩展性、可用性、可编程性、单系统镜像和容错能力。我们将检测 Tianhe-1A、Gray XT5 Jaguar 和 IBM Roadrunner 的集群体系结构，同时还将介绍 LSF 中间件和 MOSIX/OS，它们在构建网格和云的 Linux 集群、GPU 集群，以及集群扩展中用于作业和资源管理。本章将只介绍物理集群，虚拟集群将会在第 3 章和第 4 章介绍。

2.1 大规模并行集群

计算机集群（computer cluster）由相互联系的个体计算机聚集组成，这些计算机之间相互联系并且共同工作，对于用户来说，计算机集群如同一个独立完整的计算资源池。集群化实现作业级的大规模并行，并通过独立操作实现高可用性。计算机集群和大规模并行处理器（MPP）的优点包括可扩展性能、高可用性、容错、模块化增长和使用商用组件。这些特征能够维持在硬件、软件和网络组件上经历的生成变化。集群计算兴起于 20 世纪 90 年代中期，当时传统的大型机和向量超级计算机已被证实高性能计算中具有较低的成本效益。

2010 年发布的 Top500 超级计算机中 85% 是由同构节点构建的计算机集群或者 MPP。计算机集群依赖于当今建立在数据中心之上的超级计算机、计算网格和互联网云。现今，大量计算机的应用变得日益重要。根据最近 IDC 预测，高性能计算机市场将会在 2010 年到 2013 年，由 85 亿美元增长到 105 亿美元。大部分 Top500 超级计算机被用于科学和工程中的高性能计算。高吞吐量集群服务器在商业和 Web 服务应用中也被更为广泛地使用。

2.1.1 集群发展趋势

计算机集群化已由高端大型计算机的相互连接转变为使用大量的 x86 引擎。计算机集群化由大型计算机（如 IBM Sysplex 和 SGI Origin 3000）之间的链接发展而来。其目的是满足协同组计算的需求，并为关键企业级应用提供更高的可用性。随后，集群化的发展更多面向网络中的大量小型计算机，例如，DEC 的 VMS 集群化是由共享同一套磁盘/磁带控制器的多个 VAX 互连组成。Tandem 的 Himalaya 是为容错在线事务处理（Online Transaction Processing, OLTP）应用而设计的商业集群。

在 20 世纪 90 年代早期，接下来是建立基于 UNIX 工作站集群，其中具有代表性的是 Berkely NOW（Network of Workstations）和基于 AIX 的 IBM SP2 服务器集群。2000 年以后，集群发展趋势变为 RISC 或 x86 个人计算机引擎的集群化。集群产品目前已出现在集成系统、软件工具、可用基础设施和操作系统扩展中。集群化的发展趋势与计算机工业的削减趋势相一致。对较小节点集群的支持将使得集群配置的销售额以模块化增量递增。从 IBM、DEC、Sun 和 SGI 到 Compaq 和 Dell，计算机工业的发展使得可以采用低成本服务器或 x86 台式计算机实现具有成本效益、可扩展性和高可用性特征的集群化。

集群系统的里程碑

集群化已成为计算机体系结构中的热门研究方向。快速通信、作业调度、SSI 和 HA 是集群研究中的活跃课题。表 2-1 中列出了部分里程碑式的集群研究项目和商业集群产品。这些老集群的细节可以参阅文献[14]。这些里程碑项目在过去 20 年内带领着集群化硬件和中间件的发展。列表中的每个集群均发展了一些独特特征。现代集群朝着高性能集群的方向发展，这将在 2.5 节中介绍。

表 2-1 研究或商业集群计算机系统的里程碑^[14]

项 目	支持集群化的具体特征
DEC VAXcluster (1991)	运行 VMS/OS 及其扩展的 SMP 服务器组成的 UNIX 集群，主要用于高可用性应用
加州大学伯克利分校 NOW 项目 (1995)	工作站的无服务器网络，具有动态消息传递、合作归档和 GLUnix 开发的特征
莱斯大学的 TreadMarks (1996)	软件驱动的分布式共享内存，用于基于页面迁移的 UNIX 工作站集群
Sun Solaris MC Cluster (1995)	建立于 Sun Solaris 工作站之上的研究集群；部分集群操作系统功能得以发展，但是从未商用
Tandem Himalaya Cluster (1994)	用于 OLTP 和数据库进程的可扩展与容错集群，由不间断操作系统支持构建
IBM SP2 服务器集群 (1996)	由 Power2 节点和 Omega 网络构建的 AIX 服务器集群，并获得 IBM Loadleveler 和 MPI 扩展的支持
谷歌搜索引擎集群 (2003)	具有 4 000 个节点服务器的集群，用于互联网搜索和 Web 服务应用，提供分布式文件系统和容错
MOSIX (2010) http://www.mosix.org	分布式操作系统，用于 Linux 集群、多集群、网格和云，被研究机构使用

NOW 项目解决了集群计算的各方面问题，包括体系结构、Web 服务器的软件支持、单系统镜像、I/O 与文件系统、高效通信和高可用性。莱斯大学的 TreadMarks 是软件实现共享内存的工作站集群的一个好例子。内存共享通过用户空间运行时间库实现，它是建立在 Sun Solaris 工作站上的研究型集群。部分集群操作系统的功能得到了发展，但是却没有成功地用于商用。

使用 VMS/OS 及其扩展的 SMP 服务器组成的 UNIX 集群主要用于高可用性应用。AIX 服务器集群主要由 Power2 节点和 Omega 网络组成，并由 IBM Loadleveler 和 MPI 扩展支持。用于 OLTP 和数据库处理的可扩展容错集群由不间断操作系统支持构建。谷歌搜索引擎建立在谷歌使用的商品组件上。MOSIX 由希伯来大学在 1999 年开始使用，是用于 Linux 集群、多集群、网格和云的分布式操作系统。

2.1.2 计算机集群的设计宗旨

集群的分类方式有多种。本书利用集群的 6 个正交特性对其进行分类：可扩展性、封装、控制、同构性、可编程性和安全性。

2.1.2.1 可扩展性

计算机集群化是基于模块化增长的概念。将几百个单处理器节点的集群扩展为 10 000 多核节点的超级集群不是一个简单任务。这是由于可扩展性被一些因素限制，如多核心芯片技术、集群拓扑结构、封装方式、电力消耗和冷控制技术应用。在上述因素的影响下，若目标依然是实现可扩展性能，则还必须考虑其他的一些限制因素，如内存墙、磁盘 I/O 瓶颈和容许时延等。

2.1.2.2 封装

集群节点可以被封装成紧凑或者松散的形式。在一个紧凑 (compact) 集群中，节点被紧密

布置在一个房间内的一个或多个货架上，且节点不附外设（显示器、键盘、鼠标等）。在一个松散（slack）集群中，节点连接到它们平常的外设（如完整的 SMP、工作站和 PC），并且节点可能位于不同的房间、不同的建筑，甚至偏远地区。封装直接影响通信线路的长度，因此需要选择合适的互连技术。紧凑集群通常利用专有的高带宽、低延迟的通信网络，而松散集群节点一般由标准的局域网或广域网连接。

2.1.2.3 控制

集群能够以集中或分散的形式被控制或管理。紧凑集群通常集中控制，而松散集群可以采取另一种方式。在集中式集群中，中心管理者拥有、控制、管理和操作所有节点。在分散式集群中，节点有各自的拥有者。例如，考虑某个部门中由互连台式工作站组成的集群，其中每个工作站分别被某个职员拥有。拥有者可以在任何时间重新配置、更新，甚至关闭工作站。由于单点控制的缺陷，因此系统很难管理这样一个集群。它同样需要进程调度、负载迁移、检查点、记账和其他类似任务的特殊技术。

2.1.2.4 同构性

同构集群采用来自相同平台的节点，即节点具有相同处理器体系结构和相同操作系统。
68] 通常情况下，这些节点都来自同一提供商。异构集群使用来自不同平台的节点。互操作性是异构集群的一个非常重要的问题。例如，进程迁移通常需要满足负载均衡或可用性。在同构集群中，二进制进程镜像可以迁移到另一个节点并能够继续执行。这在异构集群中是不允许的，因为当进程迁移到不同平台的节点上时，二进制代码不继续执行。

2.1.2.5 安全性

集群内通信可以是开放的或封闭的。开放集群节点间的通信路径对外界显示。外界机器可采用标准协议（如 TCP/IP）访问通信路径，从而访问单独节点。这种集群容易实现，但有几个缺点：

- 由于开放，集群内通信变得不安全，除非通信子系统提供附加的功能来确保隐私和安全。
- 外界通信可能以不可预测的形式干扰集群内通信。例如，重 BBS 流量可能干扰生产作业。
- 标准通信协议往往具有巨大的开销。

在封闭集群中，集群内通信与外界相隔离，从而缓解了上述问题。其不利条件是目前还没有高效、封闭的集群内通信标准。因此，大多数商业或学术集群按照各自的协议实现高速通信。

2.1.2.6 专用集群和企业集群

专用集群通常安装在中央计算机机房的台前架上。专用集群由相同类型的计算机节点同构配置，并且由一个类似前端主机的独立管理组管理。专用集群被用于代替传统的大型机或超级计算机。专用集群的安装、使用、管理与单一机器相似。许多用户可以登录集群执行交互和批量作业。专用集群极大地提高了吞吐量，并且减少了响应时间。

企业集群主要利用节点的闲置资源，每个节点通常是一个完整的 SMP、工作站或 PC 及其所有必要的外部设备。这些节点通常在地理上是分散的，不一定在同一个房间甚至同一幢楼里。这些节点分别被多个所有者拥有。集群管理员只有有限的控制权，因为一个节点可以在任何时候被它的所有者关闭，且所有者的“本地”作业比企业作业具有更高的优先级。这类集群通常是由异构计算机节点配置的。这些节点通常由低成本的以太网网络连接。大多数数据中心由低成本的服务器集群构成。虚拟集群在数据中心升级中起着重要作用。我们将在第 6 章讨论虚拟集群，在第 7 章、第 8 章和第 9 章讨论云。

2.1.3 基础集群设计问题

本节将对各种集群和 MPP 分类，然后将会确定集群和 MPP 系统中的主要设计问题，包括物理和虚拟集群。这些系统经常出现在计算网格、国家实验室、商业数据中心、超级计算机网站和

虚拟云平台中。正确理解集群和 MPP 的集体运行将对整体理解后续章节介绍的网格和互联网云提供极大的帮助。在推进和使用集群的过程中应该考虑一些问题。虽然在这些问题上已有大量的研究，但这仍是一个活跃的研究和发展领域。

2.1.3.1 可扩展性能

资源扩展（集群节点、内存容量、I/O 带宽等）使性能成比例增长。当然，基于应用需求或者成本效益考虑，扩大和减少的能力都是必需的。集群化因为其可扩展性得以发展，在任何集群或 MPP 计算机系统应用中都不应忽视可扩展性。

2.1.3.2 单系统镜像（SSI）

采用以太网连接的工作站集合不一定就是一个集群。集群是一个独立的系统。例如，假设一个工作站拥有一个 300 Mflops/s 的处理器、512 MB 内存和 4 GB 硬盘，并且能够支持 50 位活跃用户和 1 000 个进程。100 个这样的工作站组成的集群能否看做为单一的系统，即相当于拥有一个 30 Gflops/s 处理器、50 GB 内存和 400 GB 的磁盘，并能支持 5 000 位活跃用户和 100 000 个进程的巨大工作站或大规模站（megastation）？这是一个吸引人的目标，但是难以实现。SSI 技术旨在达成这个目标。

2.1.3.3 可用性支持

集群能够利用处理器、内存、磁盘、I/O 设备、网络和操作系统镜像的大量冗余提供低成本、高可用性的性能。然而，要实现这一潜力，可用性技术是必需的。我们将在介绍 DEC 集群（10.4 节）和 IBM SP2（10.3 节）如何尝试实现高可用性时，具体说明这些技术。

2.1.3.4 集群作业管理

集群尝试使用传统工作站或 PC 节点实现高系统利用率，而这些资源通常是不能被很好利用的。作业管理软件需要提供批量、负载均衡和并行处理等功能。我们将在 3.4 节学习集群作业管理系统。同时管理多个作业需要特殊的软件工具。

2.1.3.5 节点间通信

集群由于具有更高的节点复杂度，故不能被封装得如 MPP 节点一样的简洁。集群内节点之间的物理网线长度比 MPP 长。这在集中式集群中也是成立的。长网线会导致更大的互连网络延迟。更重要的是，长网线会产生可靠性、时钟偏差和交叉会话等更多方面问题。这些问题要求使用可靠和安全的通信协议，而这会增加开销。集群通常使用 TCP/IP 等标准协议的商用网络（如以太网）。

2.1.3.6 容错和恢复

机器集群能够消除所有的单点失效。因为冗余，集群能在一定程度上容忍出错的情况。心跳机制可以监控所有节点的运行状况。如果一个节点发生故障，那么该节点上运行的关键作业可以被转移到正常运行的节点上。回滚恢复机制通过周期性记录检查点来恢复计算结果。[70]

2.1.3.7 集群分类

基于应用需求，计算机集群可以分为三类：

- **计算集群：**主要用于单一大规模作业的集体计算。一个很好的例子是用于天气状况数值模拟的集群。计算集群不需要处理很多的 I/O 操作，如数据库服务。当单一计算作业需要集群中节点间的频繁通信，该集群必须共享一个专用网络，因而这些节点大多是同构和紧耦合的。这种类型的集群也被称为贝奥武夫集群。

当集群需要在少量重负载节点间通信时，其从本质上就是众所周知的计算网格。紧耦合计算集群用于超级计算应用。计算集群应用中间件（如消息传递接口（Message-Passing Interface, MPI）或并行虚拟机（Parallel Virtual Machine, PVM）），将程序传递到更广的集群。

- **高可用性集群：**用于容错和实现服务的高可用性。高可用性集群中有很多冗余节点以容忍故障或失效。最简单的高可用性集群只有两个可以互相转移的节点。当然，高冗余可以提供更高的可用性。可用性集群的设计应避免所有单点失效。很多商业高可用性集群能使用各种不同操作系统。
- **负载均衡集群：**通过使集群中所有节点的负载均衡而达到更高的资源利用。所有节点如同单个虚拟机（Virtual Machine, VM）一样，共享任务或功能。来自用户的请求被分发至集群的所有计算机节点，这样就可以在不同机器间平衡负载，从而达到更高的资源利用或性能。为了在所有集群节点间迁移作业或进程来实现动态负载均衡，中间件是必需的。

2.1.4 Top500 超级计算机分析

每隔 6 个月，会在超大数据集上运行 Linpack 基准测试程序评测出世界 Top500 超级计算机，此排名每年会发生一些变化。在本节中，我们将分析在体系结构、速度、操作系统、国家和应用方面的历史因素。此外，我们将比较近年快速系统的前 5 名。

2.1.4.1 体系结构演变

观察图 2-1，Top500 超级计算机这些年在体系结构演变方面很有趣。在 1993 年，250 个系统是 SMP 体系结构的，并且这些 SMP 系统在 2002 年 6 月以后都不再使用。大多数 SMP 采用共享内存和 I/O 设备的结构。在 1993 年，有 120 个 MPP 系统，到 2000 年时，MPP 达到峰值 350 个系统，而到 2010 年又减少到不足 100 个系统。单指令多数据（SIMD）机器在 1997 年消失，而集群体系结构在 1999 年开始出现。现在，集群在 Top500 超级计算机中处于支配地位。

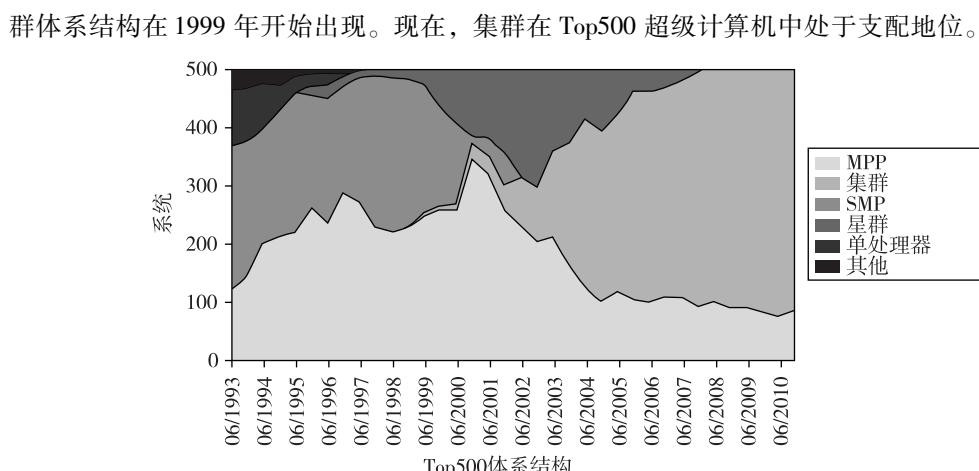


图 2-1 Top500 系统的体系结构分布

2010 年，Top500 计算机体系结构包括集群（420 个系统）和 MPP（80 个系统）。这两类系统的基本区别源于构建系统的组件。集群通常采用市场上的商用硬件、软件和网络组件。而 MPP 采用定制的计算节点、插件、模块和机壳，它们之间的互连被特定封装。MPP 要求高带宽、低延迟、更好的能效和高可靠性。在考虑成本的前提下，集群为了满足性能扩展，而允许模块化增长。MPP 因为其成本高，故而出现得较少。一般来说，每个国家只有很少的 MPP 超级计算机。

2.1.4.2 速度提升

图 2-2 绘制了 1993 年至 2010 年 Top500 快速计算机的测量性能。y 轴按照 Gflops、Tflops 和 Pflops 表示持续速度性能。中间曲线描绘了 17 年来最快速计算机的性能，峰值性能从 58.7 Gflops 增长到 2.566 Pflops。底部曲线对应于第 500 位计算机的速度，由 1993 年的 0.42 Gflops 增长为 2010 年的 31.1 Tflops。顶部曲线描述了同一时期这 500 个计算机的速度之和。在 2010 年，500 个计算机的总体速度之和达到 43.7 Pflops。有趣的是，总体速度几乎随着时间呈线性增加。

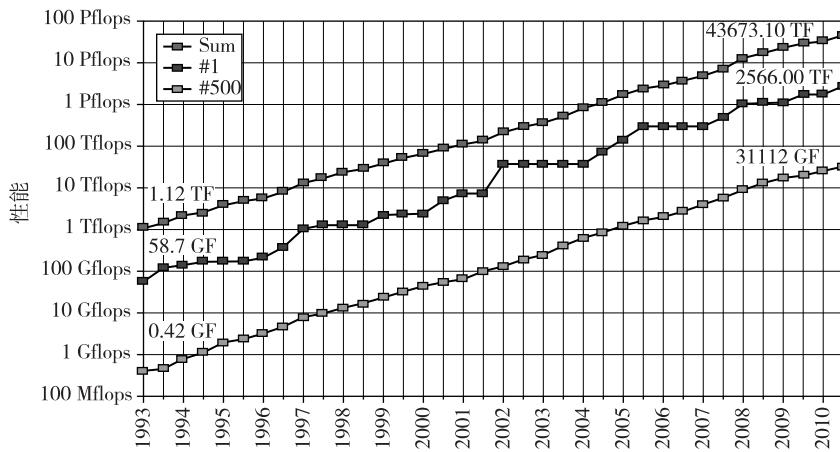


图 2-2 Top500 超级计算机的性能变化曲线 (1993—2010)

注：摘自 www.top500.org^[25]。

2.1.4.3 Top500 操作系统趋势

根据 TOP500.org (www.top500.org/stats/list/36/os) 在 2010 年 11 月发布的数据，最通用的 5 个操作系统占 Top500 计算机的市场份额已经超过 10%。根据数据，使用 Linux 的 410 个超级计算机的总处理器数超过了 450 万，82% 的系统是 Linux。排名第 2 位的是 IBM AIX/OS，被用于 17 个系统（占 3.4% 份额），处理器数超过了 94 288 个。第 3 位的是 SLEs10 与 SGI ProPack5 的联合使用，有 15 个系统（3%）和 135 200 个处理器。第 4 位的是 CNK/SLES9，用于 14 个系统（2.8%）和 113 万个处理器。最后，CNL/OS 被部署于 178 577 个处理器的 10 个系统（2%）中。余下的 34 个系统使用的其余 13 种操作系统只占 6.8%。总之，Linux 操作系统在 Top500 计算机中占据支配地位。

2.1.4.4 2010 年前 5 名超算系统

表 2-2 总结了 2010 年 11 月发布的五大超级计算机的关键体系结构特征以及持续 Linpack 基准测试性能。Tianhe-1A 在 2010 年底成为最快的 MPP，该系统由中国国防科学技术大学使用 86 386 个 Intel Xeon CPU 和 NVIDIA GPU 构建而成。我们将在 2.5 节展示某些领先的超级计算机，包括 Tianhe-1A、Cray Jaguar、Nebulae 和 IBM Roadrunner，它们在 2008 年到 2010 年都是领先的系统。表 2-3 中的五大机器已经高于 1 Pflops 的速度，其中，Pflops 量级的持续速度 (R_{\max}) 是从执行最大规模矩阵的 Linpack 基准程序测量得来的。

持续速度与峰值速度 (R_{peak}) 的比值反映了系统效率，峰值速度为系统中所有计算组件得到充分利用时的速度。前 5 位系统中，美国构建的两个系统 (Jaguar 和 Hopper) 效率最高，超过 75%。由中国构建的两个系统 (Tianhe-1A 和 Nebulae) 以及日本的 TSUBAME 2.0，效率较低。换句话说，这些系统在未来仍有改进的空间。这 5 个系统的平均功耗是 3.22 MW。这意味着未来过度的功耗可能会限制构建更快的超级计算机。为了充分利用平均每个系统的 250 000 个处理器核心，这些顶尖系统强调大规模并行性。

2.1.4.5 国家份额和应用共享

在 2010 年 Top500 名单上，274 个超级计算系统安装在美国，41 个系统在中国，103 个系统在日本、法国、英国和德国，其余国家只有 82 个系统。国家份额大致反映了该国家这些年的经济增长水平。Top500 的国家排名每 6 个月更新一次。超级计算机应用增长主要在数据库、研究、金融和信息服务等领域。

表 2-2 前 5 位超级计算机评测 (2010 年 11 月)

系统名、站点和 URL	系统名、处理器、操作系统、拓扑结构和开发者	Linpack 速度 (R_{\max})，能耗	效率 (R_{\max}/R_{peak})
1. Tianhe-1A, 中国天津国家超级计算中心, http://www.nscc-tj.gov.cn/en/	NUDT TH1A, 有 14 336 个 Xeon X5670 CPU (每个包含 6 个核) 和 7 168 个 Nvidia Tesla M2050 GPU (每个含有 448 CUDA 核), 运行 Linux, 由中国国防科技大学开发	2.57 Pflops, 4.02 MW	54.6% (超过峰值 4.7 Pflops)
2. Jaguar, DOE/SC/橡树岭国家实验室, http://computing.ornl.gov	Cray XT5-HE: MPP 有 224 162 × 6 AMD Opteron, 3-D 环形网络, Linux (CLE), 由 Cray 公司生产	1.76 Pflops, 6.95 MW	75.6% (超过峰值 4.7 Pflops)
3. Nebulae, 中国深圳国家超级计算机中心	TC3600 Blade, 120 640 个核, 55 680 个 Xeon X5650 和 64 960 个 Nvidia Tesla C2050 GPU, Linux, Infiniband, 由 Dawning 公司构建	1.27 Pflops, 2.55 MW	42.6% (超过峰值 2.98 Pflops)
4. TSUBAME 2.0, GSIC Center, 日本东京工业大学, http://www.gsic.titech.ac.jp/	HP Cluster, 3000SL, 73 278 × 6 Xeon X5670 处理器, Nvidia GPU, Linux/SLES 11, 由 NEC/HP 联合构建	1.19 Pflops, 1.8 MW	52.19% (超过峰值 2.3 Pflops)
5. Hopper, DOE/SC/LBNL/NERSC, 美国伯克利, http://www.nersc.gov/	Cray XE6 150 408 × 12 AMD Opteron, Linux (CLE), 由 Cray 公司构建	1.05 Pflops, 2.8 MW	78.47% (超过峰值 1.35 Pflops)

表 2-3 用于大型集群构造的样本计算节点体系结构

节点体系结构	主要特征	代表系统
同构节点, 使用相同的多核处理器	多核处理器安装于同一节点上, 通过交叉开关连接到共享内存或本地磁盘	Cray XT5 的每个计算节点具有 2 个 6 核 AMD Opteron 处理器
混合节点, 使用 CPU 和 GPU 或者 FLP 加速器	通用 CPU 用于整型计算, GPU 作为协作处理器以加速 FLP 操作	中国天河 1 号的每个计算节点使用 2 个 Intel Xeon 处理器与 1 个 NVIDIA GPU

2.1.4.6 2010 年前 5 名超级计算机的能耗和性能

如图 2-3 所示, 依据左边的速度 (Gflops) 和右边的能耗 (MW/系统) 排列前 5 名计算机。Tianhe-1A 速度得分最高, 为 2.57 Pflops, 其能耗为 4.01 MW。排名第二的 Jaguar 能耗最高, 为 6.9 MW。排名第四的 Tsubame 系统能耗最少, 为 1.5 MW, 并且具有和 Nebulae 系统差不多的速度性能。可以定义性能/能耗的比率来协调这两个指标。Top500 绿色排名根据超级计算机的能源利用率对其进行排名。此图显示使用混合 CPU/GPU 体系结构将会消耗较少的能源。

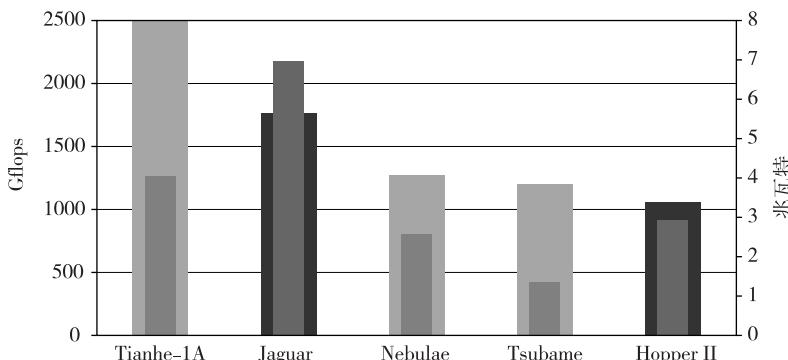


图 2-3 前 5 名超级计算机的能耗和性能 (2010 年 11 月)

注：由 www.top500.org^[25] 和 B. Dally^[10] 提供。

2.2 计算机集群和 MPP 体系结构

大多数集群关注更高的可用性和可扩展性能。集群系统由 Microsoft Wolfdog 和 Berkeley NOW 演变成 SGI Altix 系列、IBM SP 系列和 IBM Roadrunner。NOW 是加州大学伯克利分校的研究项目，旨在探索 UNIX 工作站集群化的新机制。大多数集群使用商用网络，例如千兆以太网、Myrinet 交换或者 InfiniBand 网络，连接计算和存储节点。集群化趋势从支持大型高端计算机系统转变为大容量桌面或桌边计算机系统，这正符合计算机工业中缩小体积的发展趋势。

75

2.2.1 集群组织和资源共享

在这一节中，我们将讨论基本的、小规模的个人计算机或者服务器集群。在后续的章节中，我们将讨论如何构建大规模集群与 MPP。

2.2.1.1 基本集群体系结构

图 2-4 显示了建立在个人计算机或工作站上的计算机集群的基本体系结构。该图展示了一个由商用组件构建的简单计算机集群，并且集群完全支持必需的单系统镜像特征和高可用性能力。处理节点均为商用工作站、个人计算机或服务器。这些商用节点能够很方便地替换或升级为新一代硬件。节点的操作系统支持多用户、多任务和多线程应用程序。节点由一个或多个快速商用网络连接，这些网络使用标准通信协议，并且速度比当前以太网 TCP/IP 速度高两个数量级。

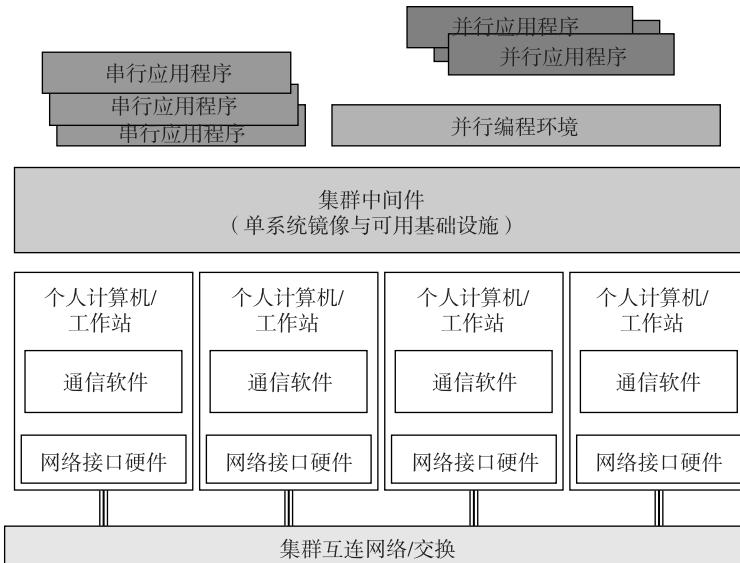


图 2-4 由商用硬件、软件、中间件和网络组件构成的计算机体系结构，支持 HA 和 SSI

网卡连接到节点的标准 I/O 总线（例如，PCI）。当处理器或操作系统发生改变，只需要改变驱动软件。我们希望在节点平台之上建立一个与平台无关的集群操作系统。但这种集群操作系统并不可以商用，我们可以在用户空间部署一些集群中间件来粘合所有的节点平台。中间件能够提供高可用的服务。单系统镜像层提供单一入口、单一文件层次、单一控制点和单一作业管理系统。单内存可由编译库或运行库辅助实现。单进程空间并不是必需的。

76

一般来说，理想中的集群包含三个子系统。首先，传统的数据库和 OLTP 监视器为用户提供一个使用集群的桌面环境。在运行串行用户程序之外，集群使用 PVM、MPI 和 OpenMP，同时支持基于标准语言和通信库的并行程序。编程环境还包括调试、仿形（profiling）、监测等工具。用户界面子系统需要综合 Web 界面和 Windows GUI 的优点。集群还应该提供不同编程环境、作业管理工具、超文本的用户友好链接和搜索支持，使得用户可以在计算集群中很容易获得帮助。

2.2.1.2 集群资源共享

小节点集群的发展将会提高计算机销量，同时集群化也增进了可用性及性能。这两个集群化目标并不一定是冲突的。部分高可用性集群使用硬件冗余来扩展性能。集群节点的连接可以采用图 2-5 所示的三种方式之一。大多数的集群采用不共享体系结构，这些集群中的节点通过 I/O 总线连接。共享磁盘体系结构有利于商业应用中小规模可用集群。当一个节点失效时，其他节点可以接管。

图 2-5a 所示的不共享结构通过以太网等局域网简单连接两个或更多的自主计算机。图 2-5b 所示的是共享磁盘集群。这类结构是大多数商业集群所需要的，可以在节点失效的情况下实现恢复。共享磁盘能存储检查点文件或关键系统镜像，从而提高集群的可用性。如果没有共享磁盘，就无法在集群中实现检查点机制、回滚恢复、失效备援和故障恢复。图 2-5c 所示的共享内存集群实现起来十分困难。节点由可扩展一致性接口（Scalable Coherence Interface，SCI）环连接，SCI 通过 NIC 模块连接每个节点的内存总线。在其他两种结构中，它们之间是通过 I/O 总线连接的，内存总线工作频率高于 I/O 总线。

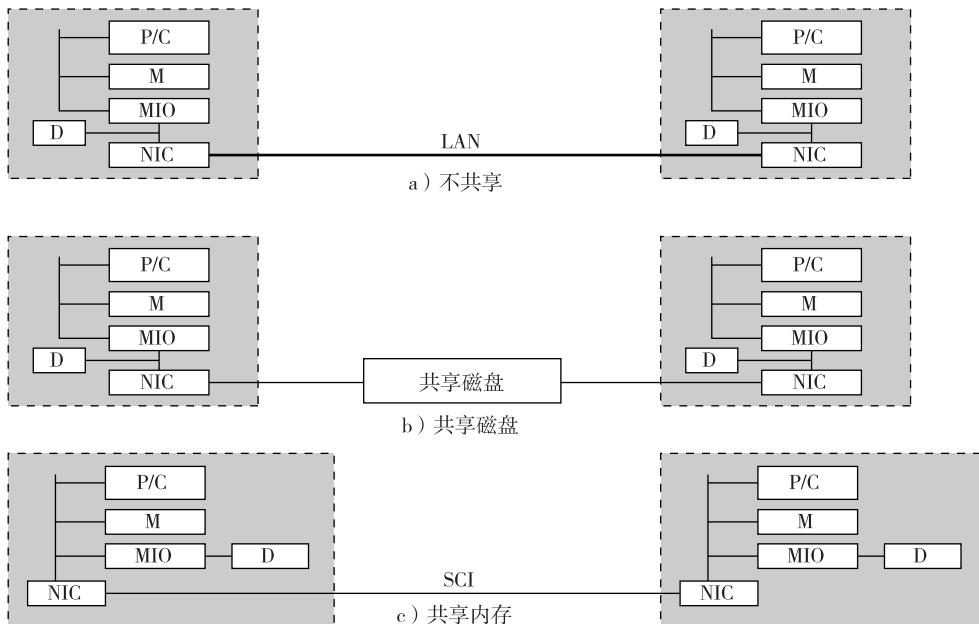


图 2-5 连接集群节点的三种方式 (P/C: 处理器和缓存; M: 内存; D: 磁盘; NIC: 网卡;

MIO: 内存-I/O 桥)

注：由 Hwang 和 Xu 提供^[14]。

目前还没有广为接受的内存总线标准，但是有 I/O 总线的标准，常用的标准为 PCI I/O 总线标准。因此，如果使用网卡连接更快的以太网与 PCI 总线，应当保证该网卡可用于使用 PCI 作为 I/O 总线的其他系统。I/O 总线相较于内存总线发展得非常缓慢。考虑一个使用 PCI 总线连接的集群。当处理器升级时，互连与网卡并不需要改变，只要新系统仍然使用 PCI。在一个共享内存的集群中，改变处理器意味着需要重新设计节点板和网卡。

2.2.2 节点结构和 MPP 封装

在构建大规模集群或者 MPP 系统时，集群节点分为两类：计算节点和服务节点。计算节点主要大量用于大规模搜索或并行浮点计算。服务节点可以使用不同的处理器来处理 I/O、文件访问和系统监控。在 MPP 集群中，计算节点占系统成本的主要部分，因为在单个大型集群系统中，

计算节点的个数可能是服务节点的 1 000 倍。表 2-3 中介绍了两种具有代表性的计算节点结构：同构设计和混合节点设计。

以往，大多数 MPP 采取同构体系结构，连接大量相同的计算节点。2010 年，Cray XT5 Jaguar 系统由 224 162 个 AMD Opteron 处理器组成，其中每个处理器有 6 个核。Tianhe-1A 采用混合节点设计，每个计算节点有 2 个 Xeon CPU 和 2 个 AMD GPU。GPU 可以用特殊浮点加速器替代。同构节点设计使得编程和系统维护变得容易。

例 2.1 IBM Blue Gene/L 系统的模块化封装

Blue Gene/L 是由 IBM 和 Lawrence Livermore 国家实验室联合开发的超级计算机。该系统是顶尖的日本地球模拟器，从 2005 年开始运行，以 136 Tflops 的速度在 Top500 名单中排名第一。到 2007 年，该系统升级至 478 Tflops 的速度。图 2-6 通过分析 Blue Gene 系列的体系结构，揭示了可扩展 MPP 系统的模块化结构。通过模块化封装，Blue Gene/L 由处理器芯片逐层构建为 64 个物理机架。该系统由 65 536 个节点构建，每个节点有 2 个 PowerPC 449 FP2 处理器。这 64 个机架通过巨大的三维 $64 \times 32 \times 32$ 环网络连接。

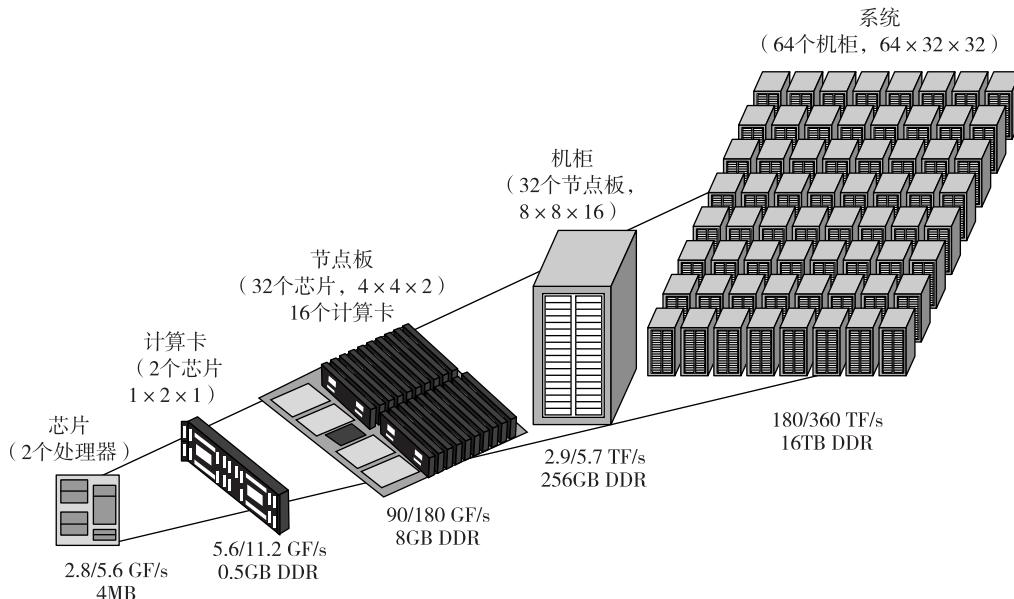


图 2-6 IBM Blue Gene/L 5 个层次上的模块化封装

注：由 N. Adiga 等人提供^[1]。

图中的左下角是一个双核处理器芯片。两个芯片安置在同一个计算机卡中。16 个计算机卡（32 个芯片或 64 个处理器）安装在一个节点板中。32 个节点板由 $8 \times 8 \times 16$ 环相互连接组成机柜箱。最后，64 个机柜（机架）组成右上角的整个系统。该封装图显示的是 2005 年的配置。客户可以选择任意规模来满足他们的计算需求。Blue Gene 集群通过保护本地失效和检查机制来确保内置的可测试性和可恢复性，从而实现可扩展性、可靠性，同时集群通过故障位置的划分与隔离实现了可服务性。

2.2.3 集群系统互连

2.2.3.1 高带宽互连

表 2-4 比较 4 种高带宽系统之间的互连。2007 年，以太网连接的速度为 1 Gbps，同时最快的 InfiniBand 连接可以达到 30 Gbps 的速度。Myrinet 和 Quadrics 的性能在以上两者之间。MPI 延迟表示了远程消息传递的状态。这 4 种技术能够实现任意的网络拓扑结构，包括纵横交换、胖树

和环状网络。InfiniBand 的连接速度最快，但是费用也最高。以太网仍是最经济有效的选择。超过 1 024 个节点的集群互连的两个例子将在图 2-7 和图 2-9 中介绍。图 2-8 将会比较 5 种集群互连的普及程度。

表 2-4 4 种集群互连技术的比较 (2007 年)

特征	Myrinet	Quadrics	InfiniBand	以太网
可用连接速度	1.28 Gbps (<i>M-XP</i>) 10 Gbps (<i>M-10G</i>)	2.8 Gbps (<i>QsNet</i>) 7.2 Gbps (<i>QsNet II</i>)	2.5 Gbps (1X) 10 Gbps (4X) 30 Gbps (12X)	1 Gbps
MPI 延迟	~3 μs	~3 μs	~4.5 μs	~40 μs
网络处理器	是	是	是	否
拓扑结构	任意	任意	任意	任意
网络拓扑	Clos	胖树	胖树	任意
路由	源路由, 直通	源路由, 直通	目的路由	目的路由
流控制	停和行	Worm-hole	基于信用	802.3x

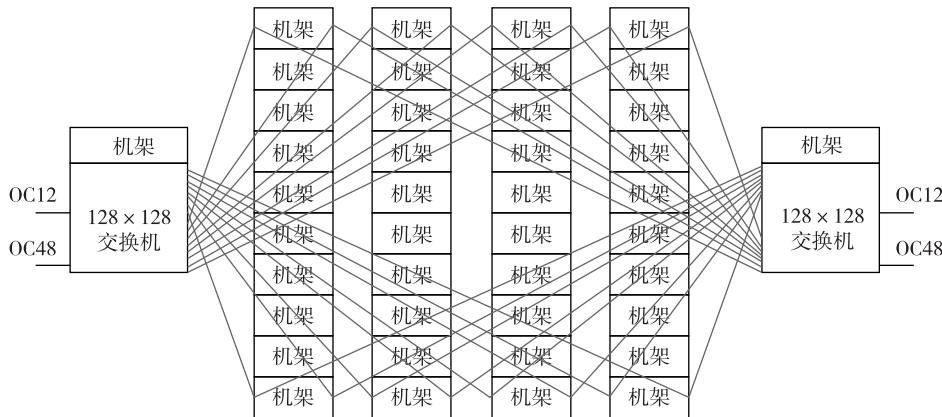


图 2-7 谷歌搜索引擎集群体系结构

注：由谷歌公司提供^[6]。

例 2.2 谷歌搜索引擎集群中的纵横交换

谷歌的很多数据中心使用低成本个人计算机引擎集群。这些集群主要用于谷歌 Web 搜索业务。图 2-7 显示了一个谷歌集群通过两个 128×128 以太网交换机连接 40 台个人计算机引擎机架。每个以太网交换机可以处理 128 个带宽为 1Gbps 的以太网链接。一台机架上有 80 台个人计算机。这是一个早期的 3 200 台个人计算机的集群。谷歌的搜索引擎集群具有比这多得多的节点。现在谷歌的服务器集群是由货柜车安装在数据中心的。

两个交换机用于提高集群的可用性。即使其中一个交换机不能提供个人计算机间的连接，集群仍可以正常工作。交换机的前端通过 2.4Gbps 的 OC12 连接到互联网，与附近数据中心网络的连接则采用 622Mbps 的 OC12。如果 OC48 连接失效，集群仍然可以通过 OC12 连接到外界。因此，谷歌集群避免了单点失效现象。■

2.2.3.2 系统互连共享

图 2-8 显示了从 2003 年到 2008 年 Top500 系统中大规模系统互连的分布情况。千兆位以太网因为低成本及符合市场需求而最受欢迎，InfiniBand 网络由于其高带宽性能而用于 150 个系统。Gray 互连是专为 Gray 系统所设计的。到 2008 年，Myrinet 和 Quadrics 网络的使用大幅下降。

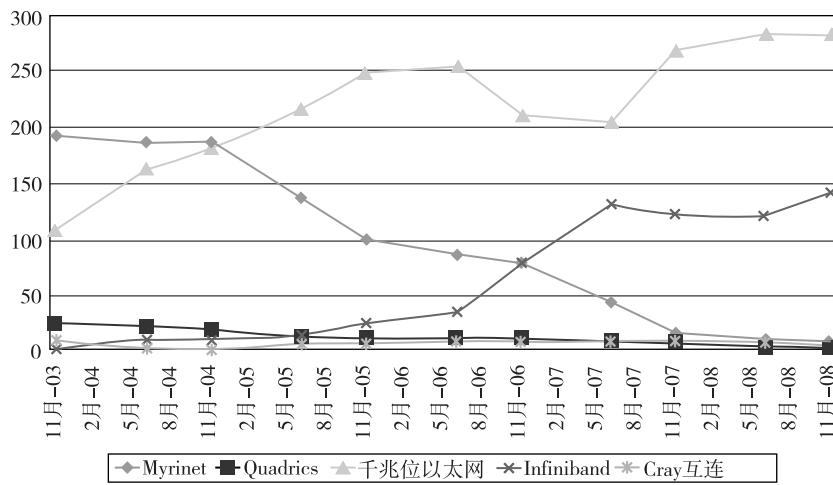


图 2-8 Top500 系统高带宽互连的分布情况 (2003—2008)

注：摘自 www.top500.org^[25]。

81

例 2.3 理解 InfiniBand 体系结构^[8]

InfiniBand 是基于交换的点对点互连体系结构。大型 InfiniBand 为分层结构，其互连支持分布式通信中的虚拟接口结构（Virtual Interface Architecture, VIA）。InfiniBand 交换和连接能够组成任何拓扑结构。其中常见的有纵横交换、胖树和环状网络。图 2-9 展示了 InfiniBand 网络的分层结构。根据表 2-5，在公布的大规模系统中，InfiniBand 提供了最快速的连接与最高速的带宽。然而，InfiniBand 网络的成本在这 4 种互连技术中最高。

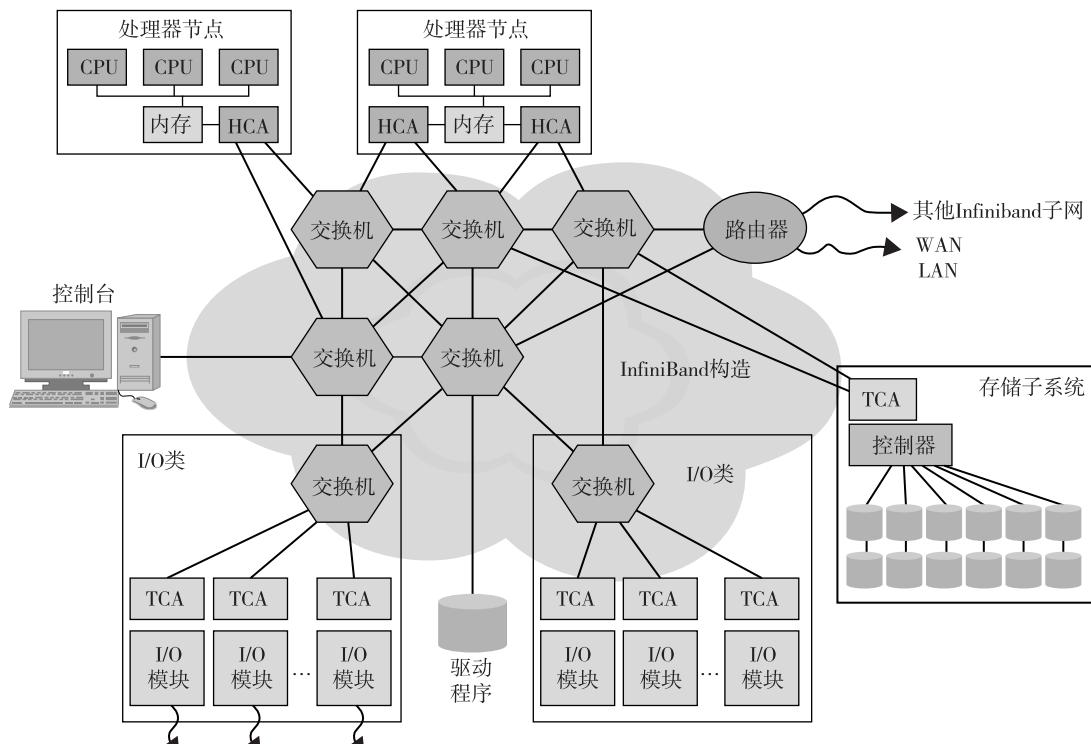


图 2-9 InfiniBand 系统构造在典型高性能计算机集群中的应用

注：由 Celebioglu 等人提供^[8]。

每个端点可能是存储控制器、网卡（NIC）或者连接主机系统的接口。主机通道适配器（Host Channel Adapter, HCA）通过标准外设组件互连（Peripheral Component Interconnect, PCI）、扩展 PCI（PCI-X）或者 PCI 专用总线提供主机接口，连接到主机处理器。每个 HCA 都至少有一个 InfiniBand 端口。目标通道适配器（Target Channel Adapter，TCA）使得 I/O 设备可以装载在网络中。TCA 包括一个 I/O 控制器，使用特定的设备协议（如 SCSI）、光纤通道或以太网。该体系结构可以很容易应用于由上千台甚至更多的主机互连构成的大规模集群。集群应用中的

[82] InfiniBand 相关内容，可以参见文献[8]。 ■

2.2.4 硬件、软件和中间件支持

实际上，集群中的 SSI 和 HA 的目标并不是免费。它们必须有相应的硬件、软件、中间件以及操作系统扩展的支持。硬件设计和操作系统扩展中的改变需由制造商完成。硬件和操作系统支持对于普通用户可能费用过高。然而，编程水平对于集群用户却是一个不小的负担。因此，应用层上中间件支持的实现费用是最少的。如图 2-10 所示的例子，在一个典型的 Linux 集群系统中，中间件、操作系统扩展和硬件支持需要达成高可用性。

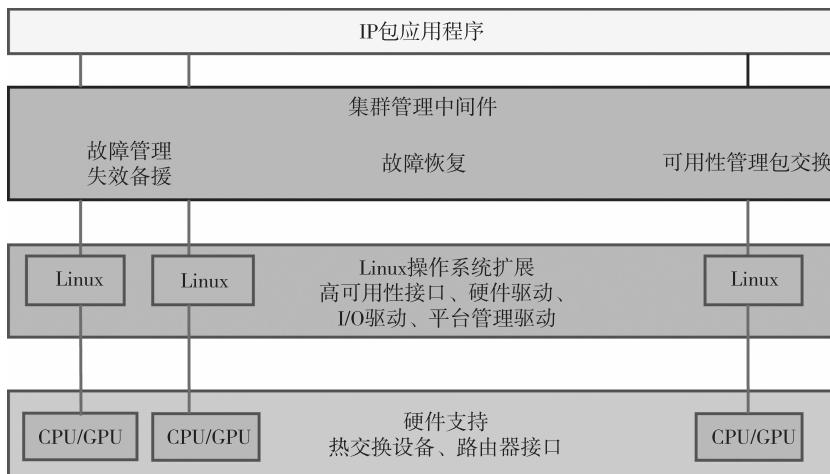


图 2-10 用于支持由 CPU 和 GPU 组成的 Linux 集群系统的大规模并行与高可用的中间件、Linux 扩展和硬件

接近用户程序端时，中间件封装在集群管理级执行：可用于故障管理，并支持失效备援和故障恢复，这将在 2.3.3 节讨论。另一个特征是使用失效检测与恢复及包交换实现高可用性。如图 2-10 中间位置所示，我们需要修改 Linux 操作系统来支持高可用性，同时需要特定的驱动支持高可用性、I/O 和硬件设备。如图 2-10 底部所示，我们需要特定的硬件来支持热交换设备和提供路由接口。在接下来的章节，我们将讨论不同的支持机制。

2.2.5 大规模并行 GPU 集群

商用 GPU 成为数据并行计算的高性能加速器。现代 GPU 芯片的每个芯片包含上百个处理器。基于 2010 年报告^[19]，每个 CPU 芯片可以进行 1Tflops 单精度（Single-Precision, SP）计算和超过 80Gflops 双精度（Double-Precision, DP）计算。目前，优化高性能计算的 GPU 包括 4GB 的板上内存，并有持续 100GB/s 以上内存带宽的能力。GPU 集群的构建采用了大量的 GPU 芯片。在一些 Top500 系统中，GPU 集群已经证实能够达到 Pflops 级别的性能。大多数 GPU 集群由同构 GPU 构建，这些 GPU 具有相同的硬件类型、制造和模型。GPU 集群的软件包括操作系统、GPU 驱动和集群化 API，如 MPI。

[83] 基于 2010 年报告^[19]，每个 CPU 芯片可以进行 1Tflops 单精度（Single-Precision, SP）计算和超过 80Gflops 双精度（Double-Precision, DP）计算。目前，优化高性能计算的 GPU 包括 4GB 的板上内存，并有持续 100GB/s 以上内存带宽的能力。GPU 集群的构建采用了大量的 GPU 芯片。在一些 Top500 系统中，GPU 集群已经证实能够达到 Pflops 级别的性能。大多数 GPU 集群由同构 GPU 构建，这些 GPU 具有相同的硬件类型、制造和模型。GPU 集群的软件包括操作系统、GPU 驱动和集群化 API，如 MPI。

GPU 集群的高性能主要归功于其大规模并行多核结构、多线程浮点算术中的高吞吐量，以及使用大型片上缓存显著减少了大量数据移动的时间。换句话说，GPU 集群比传统的 CPU 集群具有更好的成本效益。GPU 集群不仅在速度性能上有巨大飞跃，而且显著降低了对空间、能源和冷却的要求。GPU 集群相较于 CPU 集群，能够在使用较少操作系统镜像的情况下正常工作。在电力、环境和管理复杂性方面的降低使得 GPU 集群在未来高性能计算应用中非常有吸引力。

2.2.5.1 Echelon GPU 芯片设计

图 2-11 展示了一种未来的 GPU 加速器体系结构，该体系结构被建议用于为百万兆级计算服务构建的 NVIDIA Echelon GPU 集群。Echelon 项目由 NVIDIA 的 Bill Dally 领导，并在普适高性能计算（Ubiquitous High-Performance Computing, UHPC）计划中，得到美国国防部高级研究计划局（DARPA）的部分资助。该 GPU 设计中，单个芯片包含 1 024 个流核和 8 个延迟优化类 CPU 核（称为延迟处理器）。8 个流核组成流多核处理器（Stream Multi-processor, SM），128 个多核处理器组成 Echelon GPU 芯片。

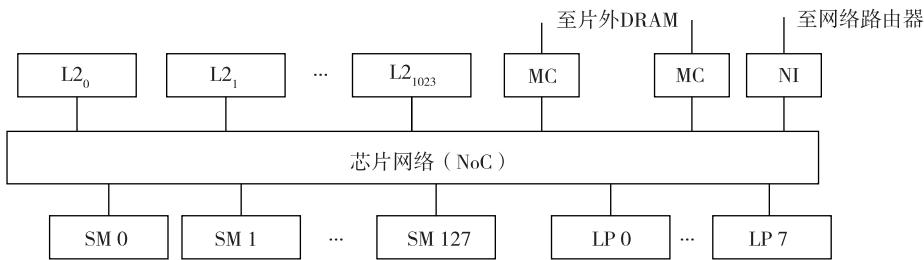


图 2-11 实现 Echelon 系统的 20Tflops 和 1.6TB/s 内存带宽的 GPU 芯片设计

注：由 Bill Dally 提供^[10]，经授权使用。

每个多核处理器被设计为含有 8 个处理器核，可以达到 160Gflops 的峰值速度。每个芯片包含 128 个流多核处理器，可以达到 20.48Tflops 的峰值速度。这些节点通过芯片网络（Network on Chip, NoC）连接到 1024 个静态随机存储器（L2 caches），每个缓存为 256KB。内存控制器（Memory Controller, MC）用来连接芯片外动态随机存储器（DRAM）和网络接口（Network Interface, NI），扩展了 GPU 集群的层次规模，如图 2-12 所示。在撰写本节内容时，Echelon 只是一个研究项目。经过 Bill Dally 的允许，我们出于学术目的展示了该设计，阐明如何探索众核 GPU 技术以达成未来 GPU 技术中的百万兆级计算，从而实现未来全面的百万兆级计算。[84]

2.2.5.2 GPU 集群组件

GPU 集群通常是一个异构系统，包含三个主要组件：CPU 主机节点、GPU 节点和它们之间的集群互连。GPU 节点由通用目的 GPU 组成，被称为 GPGPU，以完成数值计算。主机节点控制程序的执行。集群互连控制节点之间的通信。为了保证性能，多核 GPU 需要提供充足的高带宽网络和内存数据流。主机内存应该被优化，从而能匹配 GPU 芯片上的缓存带宽。图 2-12 展示了计划中的 Echelon GPU 集群，该集群使用图 2-11 所示的 GPU 芯片作为组成块，并由层次结构网络互连。

2.2.5.3 Echelon GPU 集群体系结构

Echelon 系统层次体系结构如图 2-12 所示。整个 Echelon 系统由 N 个机柜组成，分别标记为 C_0, C_1, \dots, C_N 。每个机柜有 16 个计算模块，分别标记为 M_0, M_1, \dots, M_{15} 。每个计算模块由 8 个 GPU 节点构成，分别标记为 N_0, N_1, \dots, N_7 。每个 GPU 节点是最内层的块，如图 2-12 所示，标记为 PC（细节见图 2-11）。每个计算模块可以达到 160Tflops 和 2TB 内存上 12.8TB/s 的性能。单个机柜可容纳 128 个 GPU 节点或 16 000 个处理器核。因此，每个机柜可以提供 32TB 内存上 2.6Pflops 的性能及 205TB/s 的带宽，这 N 个机柜通过光纤蜻蜓网络互连。

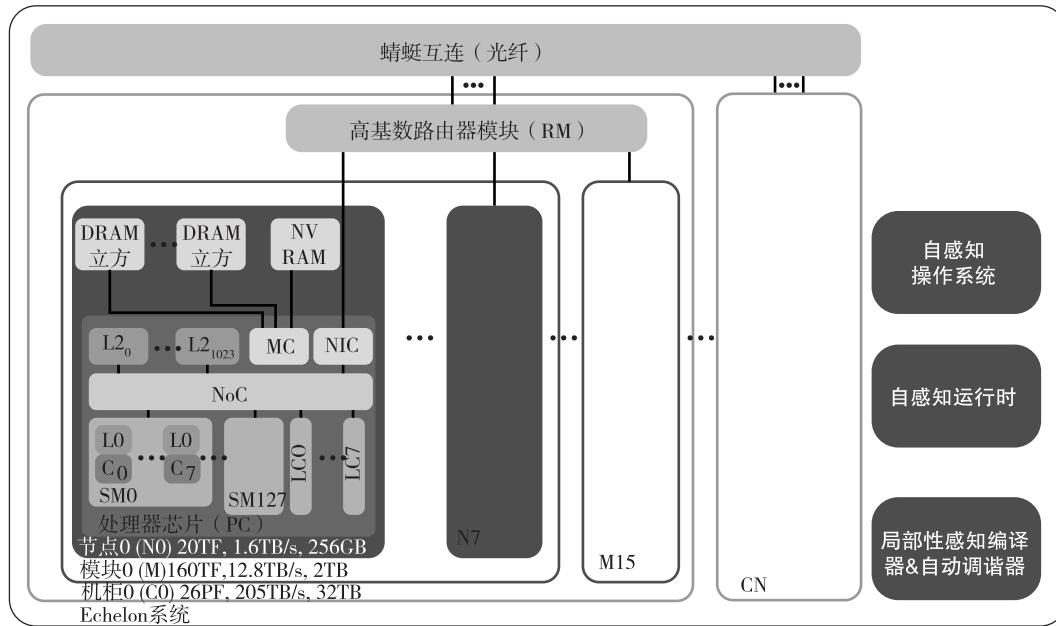


图 2-12 由 GPU 层次网络构成的 NVIDIA Echelon 系统的体系结构，其中每个机柜可以提供 2.6 Pflops 的性能，至少需要 $N = 400$ 个机柜才能实现所需的 Eflops 性能

注：由 Bill Dally 提供^[10]，经授权方可使用。

为了达到 Eflops 级别的性能，我们至少需要使用 $N = 400$ 个机柜。换句话说，百万兆级系统需要在 400 个机柜中有 327 680 个处理器核。Echelon 系统获得自感知操作系统和运行时系统的支持，同时由于其被设计为保护局部性，因此支持编译器和自动调谐器。目前，NVIDIA Fermi (GF110) 芯片已包含 512 个流处理器，因此 Echelon 设计大约快 25 倍。采用 post-Maxwell NVIDIA GPU 计划的 Echelon 很有可能在 2013 ~ 2014 年出现。

2.2.5.4 CUDA 并行编程

CUDA (Compute Unified Device Architecture, 计算统一设备体系结构) 由 NVIDIA 开发，提供并行计算体系结构。CUDA 是 NVIDIA GPU 中的计算引擎，允许开发者通过标准程序语言访问。程序员可以使用 NVIDIA 扩展和受限的 CUDA C。CUDA C 通过 PathScale Open64 C 编译器编译，可以在大量 GPU 核上并行执行。例 2.4 体现了在并行处理中使用 CUDA C 的好处。

例 2.4 在 GPU 上使用 CUDA C 并行化 SAXPY 运行

SAXPY 是矩阵乘法中频繁执行的内核操作。它本质上使用重复乘法和加法操作来产生两个长向量的点积。下列 `saxpy_serial` 程序采用了标准 C 代码。该代码只适用于单处理器核的串行执行。

```
Void saxpy_serial (int n, float a, float*x, float *y)
{ for (int i = 0; i < n; ++i), y[i] = a*x[i] + y[i] }
// Invoke the serial SAXPY kernel
saxpy_serial (n, 2.0, x, y);
```

下列 `saxpy_parallel` 程序使用 CUDA C 代码编写，可以在 GPU 芯片的多处理器核上的 256 个线程/块中并行执行。需要注意的是， n 个块由 n 个处理器核控制，其中 n 可以数以百计。

```
_global__void saxpy_parallel (int n, float a, float*x, float *y)
{ Int i = blockIdx.x*blockDim.x + threadIndex.x; if (i < n) y [i] = a*x[i] + y[i] }
// Invoke the parallel SAXPY kernel with 256 threads/block int nblocks = (n + 255)/256;
saxpy_parallel <<< nblocks, 256 >>> (n, 2.0, x, y);
```

这是一个使用 CUDA C 的很好的例子，它使用 CUDA GPGPU 作为基本部分实现了多核和多线程处理器集群上的大规模并行开发。

2.2.5.5 CUDA 编程接口

CUDA 体系结构共享一系列计算接口，其有两个竞争者：Khronos Group 的开放计算语言（Open Computing Language）和微软的直接计算（DirectCompute）。第三方包装也适用于 Python、Perl、FORTRAN、Java、Ruby、Lua、MATLAB 和 IDL 的使用。CUDA 已用于加速计算生物学、密码学等领域中一个数量级以上的非图形应用。一个很好的例子是 BOINC 分布式计算客户端。CUDA 同时提供了低级 API 和更高级 API。G8X 系列之后的所有 NVIDIA GPU 都采用了 CUDA，包括 GeForce、Quadro 和 Tesla 系列。NVIDIA 声明，由于二进制兼容性，为 GeForce 8 系列所做的程序开发无需任何改动，就可以继续用于所有未来 NVIDIA 显卡上。

85
86

2.2.5.6 CUDA 使用趋势

Tesla 和 Fermi 是基于 CUDA 体系结构分别在 2007 年和 2010 年发布的两代产品。CUDA 3.2 版在 2010 年用于单一 GPU 模块。新的 CUDA 4.0 版将解决多个 GPU 使用共享内存的统一虚拟地址空间问题。下一代 NVIDIA GPU 将会是开普勒设计，以支持 C++。Fermi 的双精度浮点运算峰值性能是 Tesla GPU 的 8 倍（5.8 Gflops/W 对 0.7 Gflops/W）。目前，Fermi 已具有 512 个 CUDA 核，共 30 亿个晶体管。

CUDA GPU 和 Echelon 系统的未来应用可能包括以下方面：

- 地球外智慧生物的研究（SETI@ Home）。
- 预测蛋白质天然构成的分布式运算。
- 基于 CT 和 MRI 扫描图像的药理分析模拟。
- 流体动力学和环境统计中的物理模拟。
- 3D 图形加速、密码学、压缩和视频文件格式的转换。
- 通过众核体系结构中的虚拟化构建单芯片云计算机（SCC）。

2.3 计算机集群的设计原则

集群设计应具有可扩展性和可用性。在这一节中，我们将会介绍通用目的计算机和协作计算机集群的单系统镜像、高可用性、容错和回滚恢复。

2.3.1 单系统镜像特征

单系统镜像指的并不是驻留在 SMP 或者工作站的内存中的操作系统镜像的单一复制。相反，它是关于单一系统、单一控制、对称性和透明性的描述，具体特征如下：

- **单一系统** 用户将整个集群作为一个多处理器系统。用户可以选择“使用 5 个处理器执行应用程序”，这不同于分布式系统。
- **单一控制** 逻辑上，一个终端用户或系统用户在一个地方只能通过单一的接口使用服务。例如，用户提交一批作业至队列；系统管理员经由一个控制点配置集群的所有硬件和软件组件。
- **对称性** 用户可以从任意节点使用集群服务。换句话说，除了受到访问权限保护的部分，所有集群服务和功能对于所有节点和所有用户是对称的。
- **位置透明性** 用户并不了解什么位置的物流设备最后提供了服务。例如，用户可以使用磁带驱动器连接到任意集群节点就像连接到本地节点一样。

87

使用单系统镜像的主要目的是，可以使用、控制和维护一个集群如同一个工作站。“单一”这个词在“单一系统镜像”中有时候等同于“总体”或者“中央”。例如，全局文件系统意味着单一文件层级，用户可以通过任意节点访问系统。单点控制允许操作者监控和配置集群系统。

虽然有单一系统的设想，但是集群服务或功能往往是通过多种组件的协作以分布式的方式实现的。单系统镜像技术的一个主要需求（和优势）是同时提供了分布式执行的性能优势和单一镜像的易用性。

从进程 P 的角度，集群中的节点可以分为三种类型。进程 P 的原始节点（home node）是创建进程 P 的节点。进程 P 的本地节点是进程 P 目前所在的节点。对于 P 来说，所有其余节点均为远程节点。可以根据不同的需求配置集群节点。主机节点通过 Telnet、rlogin，甚至 FTP 和 HTTP 为用户登录提供服务。计算节点执行计算作业。I/O 节点响应 I/O 请求，如果一个集群拥有共享磁盘和磁带单元，那么它们通常在物理上连接到 I/O 节点。

每个进程都有一个原始节点，这在进程的整个生命周期中是固定的。在任意时间，只有一个本地节点，该节点可以是也可以不是主机节点。当进程迁移时，其本地节点和远程节点可能发生变化。一个节点可以同时提供多种功能。例如，一个节点在同一时间内可以是主机节点、I/O 节点和计算节点。单系统镜像的描述可以分为几个层次，其中三层描述如下。值得注意的是，这些层次可能相互重叠。

- **应用软件层** 两个例子是并行 Web 服务器和各种并行数据库。用户通过应用程序来使用单系统镜像，甚至没有意识到他正在使用的是一个集群。这种方法需要为集群修改工作站或 SMP 的应用程序。
- **硬件或内核层** 理想情况下，单系统镜像应该由操作系统或硬件提供。遗憾的是，目前这还没有得到实现。此外，在异构集群上提供单系统镜像是极其困难的，因为大多数硬件体系结构和操作系统是专有的，所以只能被制造商使用。
- **中间件层** 最可行的方法是在操作系统内核之上建立单系统镜像层。这种方法是有发展前景的，因为它与平台无关，并且不需要修改应用程序。许多集群作业管理系统已经采用了这种方法。

集群中的每个计算机有自己的操作系统镜像。由于所有节点计算机的独立操作，因此一个集群可以显示出多个系统镜像。决定如何在集群中合并多个系统镜像，这与在社区中调节许多特征到单一特性一样困难。由于不同程度的资源共享，多个系统可以被整合，从而在不同的操作水平上实现单系统镜像。

88

2.3.1.1 单一入口

单系统镜像（SSI）包括单一入口、单文件层次、单一 I/O 空间、单一网络机制、单一控制点、单一作业管理系统、单一内存空间和单一进程空间。单一入口使得用户登录（例如，通过 Telnet、rlogin 或 HTTP）集群就像登录一个虚拟主机一样，虽然该集群可能有多个物理主机节点为登录会话服务。系统透明地分配用户登录和连接请求至不同的物理主机以平衡负载。集群可以代替大型机和超级计算机。互联网集群服务器上，成千上万的 HTTP 或 FTP 请求可能同时到达，建立多个主机的单一入口并不是一件容易的事。许多问题必须得到解决，下面只是部分问题列表：

- **主目录** 用户的主目录放在什么位置？
- **认证** 如何认证用户登录？
- **多重连接** 如果同一个用户重复登录了几次相同的用户账户，该怎么办？
- **主机失效** 如何处理一个或多个主机失效？

例 2.5 计算机集群的单一入口实现

图 2-13 描述了如何实现单一入口。集群中的 4 个节点作为接收用户登录请求的主机节点。尽管图中只显示一个用户，但数以千计的用户能以相同的方式连接到该集群。当用户登录集群时，可以输入标准 UNIX 命令，例如 telnet cluster.cs.hku.hk，使用该集群系统的符号名称。 ■

DNS 服务器翻译符号名称，并返回负载最轻的节点 IP 地址 159.226.41.150，这由节点主机 1

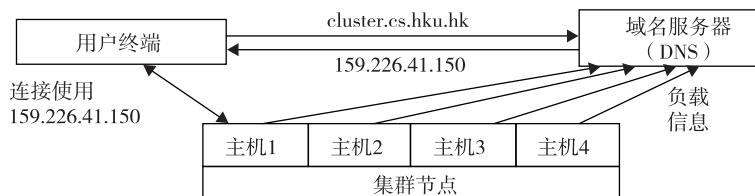


图 2-13 采用负载均衡的域名服务器 (DNS) 实现单一入口

注：由 Hwang 和 Xu^[14] 提供。

完成。用户在接下来登录时使用该 IP 地址。DNS 服务器定期从主机节点收集负载信息，从而做出相关的负载均衡决策。在理想的情况下，如果 200 个用户同时登录，登录会话将平均分布于主机，其中每个主机有 50 个会话。这使得单一主机的能力达到原来的 4 倍。

89

2.3.1.2 单文件层次

在本书中，我们使用术语“单文件层次”表示单一的、巨大的文件系统镜像，该文件系统镜像透明地整合本地和全局磁盘以及其他文件设备（例如，磁带）。换句话说，用户所需的所有文件被存储在根目录“/”的一些子目录中，可以通过普通 UNIX 调用（如 open、read 等）访问。这不能与工作站中多个文件系统作为根目录的子目录相混淆。

单文件层次的功能已经由现有的分布式文件系统（如 NFS（Network File System，网络文件系统）和 AFS（Andrew File System，Andrew 文件系统））提供了一部分。从进程的角度来看，文件能够存放在集群中的三种不同类型的位置，如图 2-14 所示。

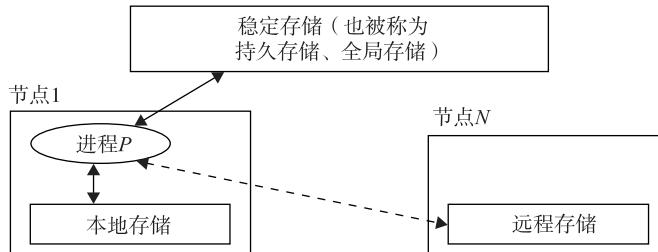


图 2-14 单文件层次中存储的三种类型。实线表示进程 P 可以访问，虚线表示 P 可能被允许访问

注：由 Hwang 和 Xu^[4] 提供。

本地存储是进程所在的本地节点上的磁盘。远程节点上的磁盘是远程存储。一个稳定的存储需要满足两个条件：它是持续的，这意味着数据一旦写入稳定存储，将会保留一段足够长的时间（例如，一个星期），即使集群关闭；利用冗余和定期磁带备份，它在某种程度上是容错的。

图 2-14 中使用了稳定存储。稳定存储中的文件称为全局文件，同理，本地存储中的文件称为本地文件，远程存储中的文件称为远程文件。稳定存储可由一个集中的大规模 RAID 磁盘实现，但也可以利用分布式集群节点的本地磁盘实现。第一种方法由于使用大规模磁盘，存在单点失效问题和潜在性能瓶颈。第二种方法比较难实现，但是可能会更经济、更有效、更可用。在许多集群系统中，按照惯例，系统允许用户进程在单文件层次上可见下列目录：传统 UNIX 工作站中常用的系统目录，如 /usr 和 /usr/local；以及拥有小磁盘配额（1 ~ 20MB）的用户主目录 ~/，用户在这里存放代码文件和其他文件。但是大数据文件必须存放在其他地方。

- 全局目录被所有用户和进程共享。该目录拥有若干 GB 的大磁盘空间。用户能够在这里存储他们的大型数据文件。
- 在集群系统中，进程能够访问本地磁盘上的特定目录。该目录具有中介的功能，与全局目录相比能够被更快地访问。

90

2.3.1.3 文件可见性

术语“可见性”在这里意味着进程能够使用 `fopen`、`fread` 和 `fwrite` 等传统 UNIX 系统或库函数访问文件。值得注意的是，集群中有多个本地擦除目录。远程节点的本地擦除目录不在单文件层次中，对进程并不能直接可见。但用户进程通过指定节点名和文件名，使用 `rcp` 等命令或一些特殊库函数仍然可以访问这些目录。

“擦除”表明该存储用来作为暂时信息存储的便笺本。本地擦除空间的信息可以在用户退出后丢弃。全局擦除空间的文件通常在用户退出后仍被保留，但是如果在一段预定时间内未被访问，将被系统删除。这对其他用户来说是免费的磁盘空间。周期的长短可由系统管理员设定，范围一般从一天到几个星期。一些系统定期或在删除文件之前，将全局擦除空间中的信息备份至磁带。

2.3.1.4 单文件层次支持

人们希望单文件层次结构具有单系统镜像属性，文件系统被重申如下：

- **单文件** 从用户的角度看，只有一个文件层次。
- **对称性** 用户能够从任意节点使用集群服务访问全局存储（例如，`/scratch`）。换句话说，除了受到访问权限保护的情况，对于所有节点和用户，所有文件服务和功能是对称的。
- **位置透明** 用户并不知道最终提供服务的物理设备的所在位置。例如，用户可以使用 RAID 连接任意节点就像连接到本地节点一样，虽然在性能上可能会存在一些差异。

集群文件系统应该维持 UNIX 语义：每个文件操作（`fopen`、`fread`、`fwrite`、`fclose` 等）均是一个事务。在 `fwrite` 修改文件之后，`fread` 访问该文件，`fread` 应该获得已经更新的文件。然而，现有的分布式文件系统并不完全符合 UNIX 语义，一些文件系统只会在关闭或清除时更新文件。在集群中组织全局存储有多种选择。一种极端方法是使用具有巨大 RAID 的主机作为单一文件服务器，该方案很简单，使用当前软件（如 NFS）能够很容易实现，但是文件服务器成为性能瓶颈，并面临单点失效的问题。另一种极端方法是利用所有节点的本地磁盘组成全局存储，这可以解决单一文件服务器的性能及可用性问题。

2.3.1.5 单一 I/O、网络和内存空间

为了实现单系统镜像，我们需要单一控制点、单一地址空间、单一作业管理系统、单一用户接口和单一进程控制，如图 2-15 所示。在这个例子中，每个节点恰好有一个网络连接。4 个节点中的两个节点分别有两个 I/O 附设。

单一网络：正确设计的集群应该表现得像一个系统（阴影区域）。换句话说，该集群就如同一个具有 4 个网络连接和 4 个 I/O 附设的大规模工作站。任意节点上的任意进程能够使用任意网络和 I/O 设备，就像它们连接在本地节点上。单一网络还意味着任意节点能够访问任意网络连接。
91

单点控制：系统管理员应该能够通过单一入口配置、监视、测试和控制整个集群和每个独立的节点。许多集群通过与所有集群节点连接的系统控制台来辅助实现这点。该系统控制台通常与外部局域网（未在图 2-15 中显示）连接，这样管理员能够在局域网的任何地方远程登录系统控制台，进行管理员的工作。

值得注意的是，单点控制并不意味着所有系统管理工作必须由系统控制台单独执行。实际上，很多管理功能分布在集群上。这意味着管理集群不应该比管理 SMP 或主机困难。管理相关的系统信息（如各种配置文件）应该保存在一个逻辑空间中。管理员使用图形工具监控集群，图形工具显示集群的完整信息，并且管理员可以随意放大和缩小。

在构建集群系统中，单点控制（或单点管理）是最富挑战性的课题之一。分布式和网络系统管理中的技术能够被转换应用于集群。由于网络管理，若干实际标准已经得到发展。简单网络管理协议（Simple Network Management Protocol，SNMP）是其中的一个例子。它需要一个整合了可用支持系统、文件系统和作业管理系统的有效集群管理软件包。

单一内存空间：单一内存空间为用户提供一个大规模集中式主内存的假象，但实际上可能是分布式本地内存空间的集合。PVP、SMP 和 DSM 在这方面比 MPP 集群优秀，因为它们允许程序利用全局或本地内存空间。测试集群是否有单一内存空间的一个好方法是，运行一个串行程序，其需要的内存空间超过任意单个节点所能提供的。

假设图 2-15 中的每个节点有 2GB 的用户可用内存。理想的单一内存镜像应该允许该集群运行需要 8GB 内存的串行程序。这使得集群运行时如同一个 SMP 系统。为了实现集群上的单一内存空间，可以尝试几种方法。其中一种方法是让编译器将应用程序的数据结构分布于多个节点之上。开发有效的、平台无关的并且支持串行二进制代码的单一内存机制是具有挑战性的任务。[92]

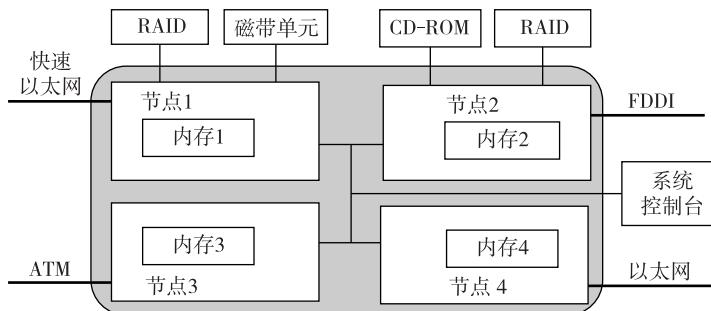


图 2-15 具有单一网络连接、单一 I/O 空间、单一内存和单点控制的集群

注：由 Hwang 和 Xu 提供^[14]。

单一 I/O 地址空间：假设集群被用来作为一个 Web 服务器。Web 信息数据库分布在两个 RAID 上。每个节点启动一个 HTTP 后台程序处理来自网络连接的 4 个 Web 请求。单一 I/O 空间意味着任意节点能够访问这两个 RAID。假设大多数请求来自于 ATM 网络。如果节点 3 具有的 HTTP 功能可以分布到所有的 4 个节点上，这将是有益处的。下面的例子介绍了 RAID-x 体系结构，该体系结构用于以 I/O 为中心的集群计算^[9]。

例 2.6 I/O 中心集群的分布式 RAID 之上的单一 I/O 空间

Hwang 等人提出一种分布式磁盘阵列体系结构^[9]，该体系结构用来建立 I/O 中心集群应用中的单一 I/O 空间。图 2-16 显示了 4 节点 Linux PC 集群的体系结构，其中三个磁盘通过 SCSI 总线连接到每个主机节点。所有 12 个磁盘形成具有单一地址空间的完整 RAID-x。换句话说，所有个人计算机可以访问本地和远程磁盘。所有磁盘块的寻址机制是水平交叉的。正交剥离和反射使得系统中可能有与 RAID-1 等效的能力。

阴影的区域表示空白块的镜像。一个磁盘块和它的镜像通过正交方式被映射在不同的物理磁盘上。例如， B_0 位于磁盘 D_0 上。 B_0 的镜像 M_0 位于磁盘 D_3 。 D_0 、 D_1 、 D_2 和 D_3 这 4 个磁盘分别连接到 4 个服务器上，因此能够被并行访问。任意单个磁盘失效不会丢失数据块，因为磁盘的镜像可以用于数据恢复。所有磁盘块被标注相应的镜像映射。基准测试程序实验表明该 RAID-x 是可扩展的，并且能够在任意单个磁盘失效后恢复数据。该分布式 RAID-x 提高了集群中所有物理磁盘的并行读/写操作的总体 I/O 带宽。 ■

2.3.1.6 其他 SSI 所需特征

SSI 的最终目标是使得集群如同台式计算机一样易于使用。下面是 SSI 额外特征，这些特征存在于 SMP 服务器中：

- **单一作业管理系统** 所有集群作业能够由任意节点提交到单一作业管理系统。
- **单一用户接口** 用户通过单一图形界面使用集群。这适用于工作站和个人计算机。发展集群 GUI 的一个好的方向是利用 Web 技术。

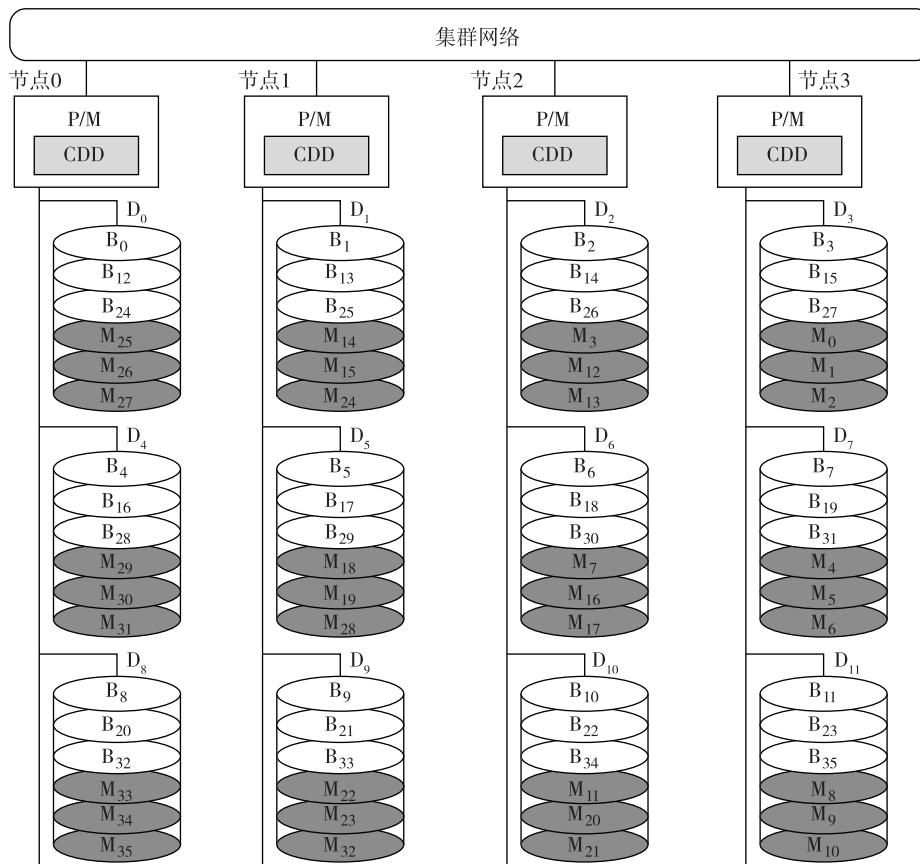


图 2-16 在连接到集群中 4 个主机的 12 个分布式磁盘之上具有单一 I/O 空间的分布式 RAID 体系结构 (D_i 表示磁盘 i , B_j 表示磁盘块 j , M_j 是 B_j 的一个镜像, P/M 表示处理器/内存节点, CDD 表示协作磁盘驱动器)

注：由 Hwang、Jin 和 Ho 提供^[13]。

- **单一进程空间** 各节点的所有用户进程形成单一进程空间，并且共享统一进程认证机制。在任意节点上能够创建进程（如通过 UNIX fork）并与远程节点上的进程通信（如通过信号、管道等）。
- **SSI 集群化的中间件支持** 如图 2-17 所示，在集群应用的三个层次上，中间件支持各种 SSI 特征。
- **管理级** 该级处理用户应用程序，并且提供作业管理系统，如 GLUnix、MOSIX、Load Sharing Facility (LSF) 或 Codine。
- **编程级** 该级提供单一文件层次（NFS、xFS、AFS、Proxy）和分布式共享内存（TreadMark、Wind Tunnel）。
- **实现级** 该级支持单一进程空间、检查点机制、进程迁移和单一 I/O 空间。这些特征必须与集群硬件和操作系统平台结合。在例 2.6 中，分布式磁盘阵列、RAID-x 实现了单一 I/O 空间。

2.3.2 冗余高可用性

当设计鲁棒的高可用系统时，三个术语经常一起使用：可靠性、可用性和可服务性（Reliability, Availability, and Serviceability, RAS）。可用性是最有意义的概念，因为它综合了可

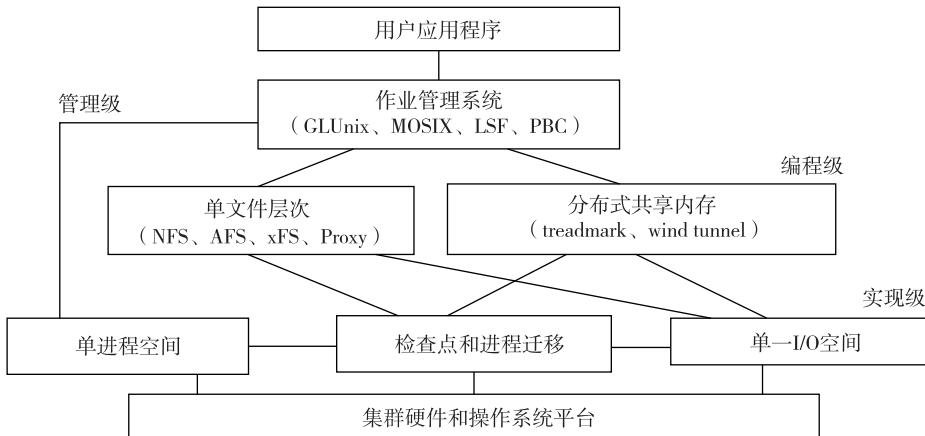


图 2-17 在作业管理、编程和实现级上集群化中间件的关系

注：由 K. Hwang、H. Jin、C. L. Wang 和 Z. Xu^[16] 提供。

可靠性和可服务性的概念，定义如下：

- 可靠性 根据系统不发生故障的运行时间衡量。
- 可用性 表示系统对用户可用的时间百分比，即系统正常运行的时间百分比。
- 可服务性 与服务系统的容易程度相关，包括硬件和软件维护、修复、升级等。

RAS 需求由实际的市场需求决定。最近的 Find/SVP 调研总结了世界 1 000 强企业中的下列情形：计算机平均每年发生 9 次故障，平均每次故障时间为 4 小时。平均每小时损失的收入是 82 500 美元。由于故障过程中可能造成巨大损失，许多公司都在努力提供 24/365 可用的系统，即该系统每天 24 小时，每年 365 天都是可用的。

2.3.2.1 可用性和失效率

如图 2-18 所示，计算机系统通常在发生故障前会运行一段时间。故障被修复后，系统恢复正常运行，然后不断重复这个运行 - 修复周期。系统可靠性由失效平均时间 (MTTF) 衡量，该时间指的是系统（或系统部件）发生故障前正常运行的平均时间。可服务性的度量标准是平均修复时间 (MTTR)，该时间为发生故障后修复系统及还原工作状态的平均时间。系统可用性定义为：

$$\text{可用性} = \text{MTTF} / (\text{MTTF} + \text{MTTR}) \quad (2-1)$$

2.3.2.2 计划停机和意外失效

学习 RAS 时，我们称任意使得系统不能正常执行的事件为失效 (failure)。这包括：

- 意外失效 由于操作系统崩溃、硬件失效、网络中断、人为操作失误以及断电等而引起的系统失效。所有这些被简单地称为失效，系统必须修复这些失效。
- 计划停机 系统没有被损坏，但是周期性中止正常运行以进行升级、重构和维护。系统也可能在周末或假日关闭。图 2-18 所示的 MTTR 是关于这类失效的计划停机时间。

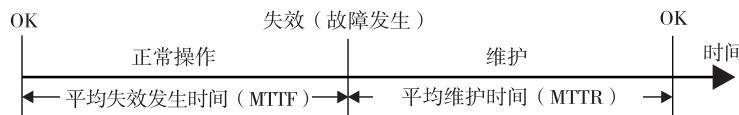


图 2-18 计算机系统的执行 - 修复周期

表 2-5 显示了几种具有代表性的系统可用性值。例如，传统工作站具有 99% 的可用性，意味着其建立和运行的时间占总时间的 99%，或者每年只停机 3.6 天。可用性的乐观定义并不考虑计划停机时间，这可能是有实际意义的。例如，许多超级计算机设置为每周几小时的计划停机时间，而电话系统却不能忍受每年几分钟的停机时间。

表 2-5 不同类型计算机系统的可用性

系统类型	可用性 (%)	每年停机时间
传统工作站	99	3.6 天
高可用性系统	99.9	8.5 小时
故障可恢复系统	99.99	1 小时
容错系统	99.999	5 分钟

2.3.2.3 暂时性失效和永久性失效

很多失效是暂时的，它们短暂出现然后消失。处理这类失效不需要更换任何组件。一个标准的方法是回滚系统至已知状态，然后重新开始。例如，我们通过重启计算机来恢复诸如键盘或窗口不响应等暂时性失效。永久性失效不能通过重启来修复，必须维修或更换某些硬件或软件组件。例如，如果系统硬盘坏了，重启也不能恢复正常工作。

2.3.2.4 部分失效和整体失效

使得整个系统不可用的失效称为整体失效。如果系统在一个较低的水平仍可以运行，那么只影响部分系统的失效称为部分失效。提高可用性的关键方法是系统地移除单点失效，使得失效尽可能为部分失效，因为硬件或软件组件的单点失效会影响到整个系统。

例 2.7 SMP 和计算机集群的单点失效

在 SMP (图 2-19a) 中，共享内存、操作系统镜像和内存总线均为单一失效点。另外，处理器并不是单一失效点。在一个工作站集群 (图 2-19b) 中，位于每个工作站的多操作系统镜像通过以太网互连。这避免了 SMP 中操作系统可能造成的单点失效。然而，以太网网络却成为单一失效点，如图 2-19c 所示，其增加了高速网络，两条通信路径消除了此单点失效。

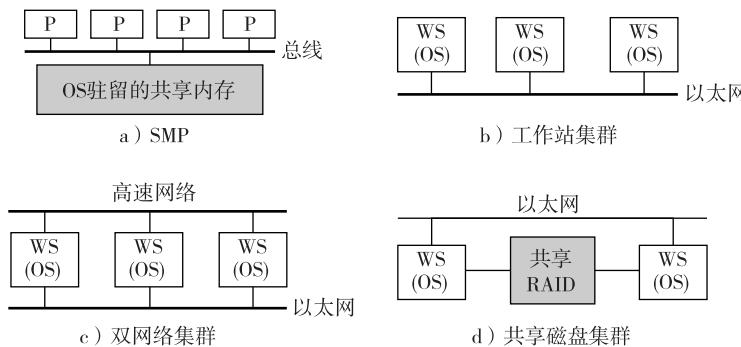


图 2-19 SMP 和三个集群中的单点失效 (SPF)，由 a 到 d，其中冗余越大，消除的单点失效也越多

注：由 Hwang 和 Xu^[14] 提供。

当图 2-19b 和图 2-19c 中的某个节点失效时，不仅该节点上的应用均失效，而且节点数据也无法使用，直至节点被修复。图 2-19d 中的共享磁盘集群为该情况提供了一种补救方案。系统在共享磁盘上存储连续数据，并且检查点周期性存储中间结果。如果一个 WS 节点失效，该共享磁盘中的数据并不会丢失。

2.3.2.5 冗余技术

考虑图 2-19d 中的集群。假设只有一个节点失效。系统的其余部分（例如，互连和共享 RAID 磁盘）是 100% 可用的。假设当一个节点失效时，该节点的工作量不需要额外时间便可转移到另一个节点上。我们提出下面的问题，如果忽略计划停机时间，集群的可用性如何？如果集群需要 1 小时/周的维护时间，可用性又如何？如果每周关闭一小时，每次只关闭一个节点，集群可用性又如何？

从表2-5可知，工作站的可用性高达99%。两个节点都停机的时间仅占0.01%。因此，可用性为99.99%。目前的故障恢复系统每年只有一个小时的停机时间。计划停机时间为52小时/年，即 $52/(365 \times 24) = 0.0059$ 。总停机时间是 $0.59\% + 0.01\% = 0.6\%$ 。集群的可用性为99.4%。如果忽略一个节点被维护时，另一个节点可能失效的情况，其可用性是99.99%。

提高系统可用性有两种基本方法：增加MTTF或减少MTTR。增加MTTF等同于增加系统可靠性。计算机工业致力于研发可靠系统，目前工作站的MTTF的范围从数百到数千小时不等。然而，进一步提高MTTF是非常困难和昂贵的。于是，集群提供了一种基于减少系统MTTR的高可用性解决方法。一个多节点集群比工作站具有较低的MTTF（因而具有较低的可靠性），然而，其失效可以被快速解决，以便提供较高的可用性。我们在集群设计中考虑了几种冗余技术。

2.3.2.6 隔离冗余

提高任何一个系统可用性的关键技术是利用冗余组件。当一个组件（主组件）失效时，该组件提供的服务可由另一个组件（备份组件）接管。此外，主组件和备份组件应该相互隔离，这意味着它们不会因为相同的原因失效。集群通过电能供应、风扇、处理器、内存、磁盘、I/O设备、网络和操作系统镜像等的冗余，提供了高可用性。在一个设计优良的集群中，冗余也是相互隔离的。隔离的冗余提供了几个好处：

- 第一，考虑隔离冗余的组件不会发生单点失效，因此该组件的失效不会导致整个系统失效。
- 第二，失效的组件可以在系统其余部分正常工作时被修复。
- 第三，主组件和备份组件可以彼此相互检测及调试。

IBM SP2通信子系统是一个很好的隔离冗余设计的例子。所有节点由两个网络连接：以太网网络与高性能交换。每个节点使用两个独立网卡分别连接到这些网络上。两种通信协议：标准IP和用户空间（User Space, US）协议；每种协议均可运行在另一种网络上。如果任一网络或协议失效，另一网络或协议可接替。

2.3.2.7 用N版本编程来增强软件可靠性

构造关键任务软件系统的通用冗余方法称为N版本编程。软件由N个独立的队列执行，这些队列甚至不知道彼此的存在。不同的队列要求使用不同的算法、编程语言、环境工具甚至平台执行软件。在一个容错系统中，这N个版本同时运行并且不断比较它们的结果。如果结果不一致，系统提示发生故障。由于隔离冗余，因此在同一时间内，某一故障导致大多数N版本失效是几乎不可能的。所以系统可根据多数表决产生的正确结果继续工作。在一个高可用非关键任务系统中，在某一时间只需运行一个版本。每一版本内置自动检测功能。当某个版本失效时，另一版本能够接管其任务。[98]

2.3.3 容错集群配置

集群解决方案的目标是为两个服务器节点提供三个不同级别的可用性支持：热备份、主动接管和容错。在这一节中，我们将考虑恢复时间、回滚特征和节点主动性。可用性水平促进从备份变化为主动和容错集群配置。恢复时间越短，集群的可用性越高。回滚指的是一个恢复节点在修复和维护后回归正常执行的能力。主动性指的是该节点在正常运行中是否用于活跃任务。

- **热备份服务器集群** 在一个热备份集群中，一般情况下只有主要节点积极完成所有有用的工作。备份节点启动（热）和运行一些监控程序来发送与接收心跳信号以检测主要节点的状态，但并不积极运行其余有价值的工作。主要节点必须备份所有数据至共享磁盘存储，该存储可被备份节点访问。备份节点需要二次复制的数据。
- **主动接管集群** 在这个例子中，多个服务器节点的体系结构是对称的。两个服务器都是主要的，正常完成有价值的任务。两个服务器节点通常都支持故障切换和恢复。当一个节点失效时，用户应用程序转移至集群中的其他可用节点。由于实施故障切换需要时间，

用户可能会遇到一些延迟或者丢失在最后检查点前未保存的部分数据。

- **故障切换集群** 故障切换可能是目前商业应用集群最重要的特征。当一个组件失效时，该技术允许剩余系统接管之前由失效组件提供的服务。故障切换机制必须提供一些功能，如失效诊断、失效通知和失效恢复。失效诊断是指失效以及导致该失效的故障组件位置的检测。一种常用的技术是使用心跳消息，集群节点发送心跳消息给对方。如果系统没有接收到某个节点的心跳消息，那么可以判定节点或者网络连接失效了。

例 2.8 双网络集群的失效诊断和恢复

集群使用两个网络连接其节点。其中一个节点被指定为主节点（master node）。每个节点都

[99] 有一个心跳维护进程，该进程通过两个网络周期性（每 10 秒）发送心跳消息至主节点。如果主节点没有接收到某节点的心跳（10 秒）消息，那么将认为探测到失效并会作出如下诊断：

- 节点到两个网络之一的连接失效，如果主节点从一个网络接收到该节点的心跳消息，但从另一个却没有接收到。
- 节点发生故障，如果主节点从两个网络都没有接收到心跳消息。这里假设两个网络同时失效的几率忽略不计。

示例中的失效诊断很简单，但它有若干缺陷。如果主节点失效，怎么办？10 秒的心跳周期是太长，还是太短？如果心跳消息在网络中丢失了（例如，由于网络拥塞），怎么办？该机制能否适用于数百个节点？实际的高可用性系统必须解决这些问题。一种常用的技术是使用心跳消息携带负载信息，当主节点接收到某个节点的心跳消息时，它不仅了解该节点存活，而且知道该节点的资源利用率等情况。这些负载信息对于负载均衡和作业管理是很有用的。

失效一旦被诊断，系统将通知需要知道该失效的组件。失效通知是必要的，因为不仅仅只有主节点需要了解这类信息。例如，某个节点失效，DNS 需要被通知，以至不会有更多的用户连接到该节点。资源管理器需要重新分配负载，同时接管失效节点上的剩余负载。系统管理员也需要被提醒，这样他能够进行适当的操作来修复失效节点。 ■

恢复机制

失效恢复是指接管故障组件负载的必需动作。恢复技术有两种类型：在向后恢复中，集群上运行的进程持续地存储一致性状态（称为检查点）到稳定的存储。失效之后，系统被重新配置以隔离故障组件、恢复之前的检查点，以及恢复正常操作。这称为回滚。

向后恢复与应用无关、便携，相对容易实现，已被广泛使用。然而，回滚意味着浪费了之前执行结果。如果执行时间是至关重要的，如在实时系统中，那么回滚时间是无法容忍的，应该使用向前恢复机制。在这个机制下，系统并不回滚至失效前的检查点。相反，系统利用失效诊断信息重建一个有效的系统状态，并继续执行。向前恢复是应用相关的，并且可能需要额外的硬件。

例 2.9 MTTF、MTTR 和失效成本分析

考虑一个基本没有可用性支持的集群。当一个节点失效，下面一系列事件将会发生：

1. 整个系统被关闭和断电。
2. 如果硬件失效，故障节点被替换。
3. 该系统通电和重启。
4. 用户应用程序被重新装载，并从开始重新运行。

[100] 假设集群中的某个节点每 100 小时发生一次故障。集群的其余部分不会发生故障。步骤 1~3 需要花费 2 小时。一般来说，步骤 4 的平均时间也是 2 小时。该集群的可用性是多少？如果每小时的停机损失为 82 500 美元，每年的失效损失是多少？

解 集群的 MTTF 是 100 小时，MTTR 是 $2 + 2 = 4$ 小时。根据表 2-5，可用性为 $100/104 = 96.15\%$ 。这相当于每年 337 小时的停机时间，失效损失为 $82 500 \text{ 美元} \times 337$ ，即超过 2 700 万美元。 ■

例 2.10 计算机集群的可用性和成本分析

重复例 2.9，但是现在假设该集群的可用性支持显著提高。当某个节点失效，其负载会自动转移到其他节点上。故障切换的时间只有 6 分钟。同时，集群具有热交换的能力：故障节点从集群分离、修复、重新插入、重启以及重新加入集群，这整个过程不影响集群的其余部分。这一理想集群的可用性是多少，并且每年的失效损失又是多少？

解 集群的 MTTF 仍是 100 小时，但是 MTTR 减少为 0.1 小时，因为当修复故障节点时，集群是可用的。根据表 2-5，可用性为 $100/100.5 = 99.9\%$ 。这相当于每年 8.75 小时的停机时间，失效损失是 82 500 美元，相较于例 2.9，减少至 $1/38$ 。 ■

2.3.4 检查点和恢复技术

检查点和恢复这两种技术必须共同发展，才能提高集群系统的可用性。我们将从检查点的基本概念入手。某个进程周期性地保存执行程序的状态至稳定存储器，系统在失效后能够根据这些信息得以恢复。每一个被保存的程序状态称为检查点。包含被保存状态的磁盘文件称为检查点文件。虽然目前所有的检查点软件在磁盘中保存程序状态，但是使用节点内存替代稳定存储器来提高性能还处在研究阶段。

检查点技术不仅对可用性有帮助，同时对程序调试、进程迁移和负载均衡也是有用的。许多作业管理系统和一些操作系统支持某种程度上的检查点。Web 资源包含众多检查点相关的 Web 网站，还包括一些公共领域软件，如 Condor 和 Libckpt。这里，我们将展示检查点软件的设计者和用户重点关注的问题。我们将首先考虑串行和并行程序的共同问题，接下来将单独讨论并行程序的相关问题。

2.3.4.1 内核、库和应用级

检查点可以由操作系统在内核级实现，操作系统在内核级透明地设立检查点并重新开始进程。这对用户来说是理想的。然而，尤其对于并行程序，大多数操作系统并不支持检查点。在用户空间，以一种较不透明地方式链接用户代码和检查点库。检查点和重启操作由运行时支持所掌控。[101]这种方法使用广泛，因为它不需要修改用户程序。

一个主要的问题是目前大多数检查点库是静态的，这就意味着应用程序的源代码（或至少对象代码）必须是可得到的。如果应用程序是可执行代码的形式，它则不能正常工作。第三种方法需要用户（或编译器）在应用程序中插入检查点函数。因此，应用程序必须被修改，透明度也就不能保证了。然而，它的优点是用户可以指定在哪个位置设立检查点。这有利于减小检查点的开销，因为检查点会消耗一定的时间和存储。

2.3.4.2 检查点开销

在一个程序的执行过程中，它的状态可能保存很多次。这被表示为保存检查点所需时间。存储开销指的是检查点需要的额外内存和磁盘空间。时间和存储开销取决于检查点文件的大小。开销可能是巨大的，尤其当应用程序需要一个大的内存空间时。已经有许多技术被推荐，用来降低这些开销。

2.3.4.3 选择最优检查点间隔

两个检查点之间的时间间隔称为检查点间隔。时间间隔增大可以降低检查点的时间开销。然而，这意味着失效后更长的计算时间。Wong 和 Franklin 推导出图 2-20 所示最优检查点间隔的表达式：

$$\text{最优检查点间隔} = \sqrt{(MTTF \times t_c) / h} \quad (2-2)$$

这里 MTTF 是系统的平均失效时间。MTTF 反映了保存一个检查点的时间开销， h 是在系统故障前的检查点时间间隔内，进行正常计算的平均百分比。参数 h 处于某个范围内。系统恢复之后，

需要花费 $h \times (\text{检查点间隔})$ 的时间来重新计算。

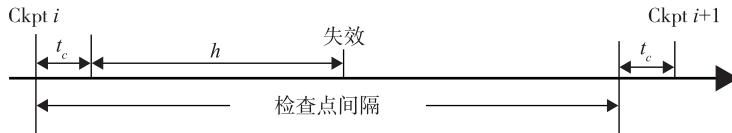


图 2-20 两个检查点间的时间参数

注：由 Hwang 和 Xu^[14] 提供。

2.3.4.4 增量检查点

相对于每个检查点保存全状态，增量检查点机制只保存与之前检查点相比发生改变的状态。然而，必须关注之前的检查点文件。在全状态检查点中，只需在磁盘上维护一个检查点文件，之后的检查点文件可以简单地覆盖此文件。在增量检查点中，之前的文件仍需要被维护，因为一个
[102] 状态可能横跨许多文件。因此，总存储需求较大。

2.3.4.5 分支检查点

大多数检查点机制是阻塞的，因为当设置检查点时，正常的计算被停止。如果有足够的内存，可以通过内存中程序状态的复制并唤起另一个异步线程同时执行检查点程序，以减少检查点的开销。一个简单的方法是使用 UNIX `fork()` 系统调用计算重复检查点。分支子进程复制父进程的地址空间并设置检查点，与此同时，父进程继续执行。由于检查点程序是磁盘-I/O 密集的，重叠操作可以实现，进一步的优化可使用写时优化机制。

2.3.4.6 用户指导检查点

如果用户插入代码（例如，库或系统调用）告知系统何时保存、保存什么以及不保存什么，检查点开销有时能够大幅度降低。检查点的准确内容应该是什么？它应该包含足够的信息帮助系统恢复。进程状态包括其数据状态和控制状态。在 UNIX 进程中，这些状态存储在其地址空间，包括文本（代码）、数据、堆栈段和进程描述符。保存和恢复全状态的代价是昂贵的，有时甚至是不可能的。

例如，进程 ID 及其父进程 ID 是不可恢复的，在许多应用中它们也不需要被保存。大多数检查点系统只保存部分状态。例如，通常不保存代码段，因为在多数应用程序中其不发生改变。什么类型的应用能够被设置检查点呢？目前检查点机制需要程序是多机通用的（well behaved），精确的定义在不同的方案中有所不同。在最低程度上，通用程序应该不需要不可恢复的状态信息，例如进程的 ID 数值。

2.3.4.7 并行程序检查点

现在我们来看并行程序检查点。通常并行程序的状态远多于串行程序的状态，因为它包括了独立进程的状态集合，以及网络通信状态。并行同时也会带来多种时间和一致性问题。

例 2.11 一个并行程序检查点

图 2-21 描述了一个三进程并行程序的检查点。标记为 x 、 y 和 z 的箭头表示进程间的点对点通信。三条分别标记为 a 、 b 和 c 的粗实线表示三个全局快照（或简称快照），这里的全局快照指检查点的集合（表示为点），每个检查点来自一个进程。此外，一些通信状态也可能需要保存。快照线与进程时间线的交点表示该进程设置（局部）检查点的位置。因此，程序快照 c 由三个局部检查点组成： s 、 t 、 u 分别是进程 P 、 Q 和 R 的检查点，以及保存的通信状态 y 。

2.3.4.8 一致快照

如果一个进程在检查点处没有接收到消息，而这个消息并没有由其他进程发出，那么全局性快照称为一致的。在图形中，这相当于没有从右至左穿过快照线的箭头。因此，快照 a 是一致的，因为箭头 x 是从左向右的。但是快照 c 是不一致的，因为 y 是从右到左的。为了确保一致

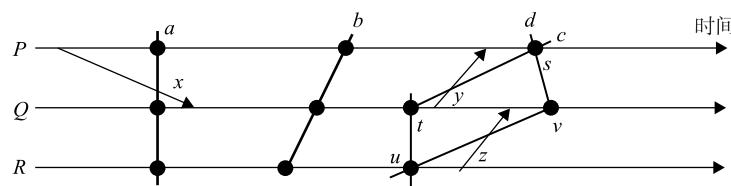


图 2-21 并行程序中的一致检查点和非一致检查点

注：由 Hwang 和 Xu^[14] 提供。

性，在两个检查点之间不应该有任何锯齿路径（zigzag path）^[20]。例如，检查点 u 和 s 不属于一个一致性的全局快照。更苛刻的一致性要求需要没有箭头穿过快照，这样只有快照 b 是一致的，如图 2-21 所示。

2.3.4.9 协作检查点和独立检查点

并行程序的检查点机制可分为两种类型。在协作检查点（也称为一致检查点）中，并行程序冻结，并且所有进程在同一时间设置检查点。在独立检查点中，进程彼此独立设置检查点。这两种类型可以通过不同方式相结合。协作检查点难以实现，并且需要巨大的开销。独立检查点则具有较小的开销，可以利用串行程序现有的检查点机制。

2.4 集群作业和资源管理

本节涵盖在集群系统上执行多个作业的多种调度方法，还将介绍描述集群计算的中间件，以及在大规模集群或云中，用于资源管理的分布式操作系统 MOSIX。

2.4.1 集群作业调度方法

集群作业可能在一个指定的时间（日历调度），或者在特定事件发生（事件调度）时被调度运行。表 2-6 总结了用于解决集群作业调度问题的各种方案。根据提交时间、资源节点、执行时间、内存、磁盘、作业类型及用户认证的优先级，作业被调度。静态优先级指的是根据预定的方案，作业被分配的优先级。其中一个简单的方案是采用先到先服务的形式调度作业。另一种方案是为用户分配不同的优先级，而作业的动态优先级可能会随时间发生变化。

表 2-6 集群节点的作业调度问题和机制

问 题	机 制	关 键 问 题
作业优先级	非抢占式	高优先级作业被延迟
	抢占式	开销、实现
资源需求	静态	负载不平衡
	动态	开销、实现
资源共享	专用	低利用率
	空间共享	铺盖、大规模作业
调度	分时	基于进程的作业控制与上下文切换开销
	独立	严重减速
	组调度	实现困难
与外界（本地）作业竞争	驻留	本地作业减速
	迁移	迁移阈值、迁移开销

共享集群节点有三种不同机制。在专用模式中，在某一时间集群中只有一个作业运行，在某一时间一个节点至多被分配一个作业进程。作业直至运行完成，才释放集群让其他作业运行。值得注意的是，即使在专用模式中，一些节点可能保留，以供系统使用，不对用户作业开放。除此

之外，所有集群资源用于运行单一作业，这可能会导致系统利用率低。作业资源需求可以是静态的，也可以是动态的。静态机制在单一作业的整个周期，固定分配一定数目的节点。静态机制可以充分利用集群资源。它无法处理所需节点变得不可用的情形，例如，单工作站的所有者关闭了机器。

动态资源允许作业在运行中获得或释放节点。然而，这是难以实施的，需要运行的作业和 Java 信息服务（Java Message Service，JMS）协作。作业对 JMS 提交异步添加/删除资源的请求。JMS 需要通知该作业何时资源变得可用，同步意味着作业会被请求/通知推迟（阻塞）。作业和 LMS 的协作需要修改编程语言/库。此协作的原始机制存在于 PVM 和 MPI 中。

2.4.1.1 空间共享

常用的方案是在日间赋予短的交互作业较高的优先级，而在晚间使用瓷砖式覆盖。在这个空间共享模式中，多个作业可以在分离的节点分区（组）同时运行。一个进程在某一时间至多被分配到一个节点。虽然部分节点只分配给一个任务，但是互连和 I/O 子系统可能被所有作业共享。空间共享必须解决瓷砖式覆盖问题和大规模作业问题。

例 2.12 集群节点上依据瓷砖式覆盖的作业调度

图 2-22 描述了瓷砖式覆盖技术。在图 2-22a 中，JMS 在 4 个节点上按照先到先服务的策略调度 4 个作业。作业 1 和 2 较小，因此被分配至节点 1 和 2。作业 3 和 4 是并行的，且每个均需要 3 个节点。当作业 3 到达时，它不能够立即运行。它必须等待作业 2 完成，并释放其使用的节点。如图 2-22b 所示，瓷砖式覆盖会增加节点的利用率。在可用节点上重新装载这 4 个作业，这些作业的整体执行时间减少了。这个问题在专用模式或者空间共享模式中不能得到解决。然而，通过分时操作，该问题能够得以缓和。

103
105

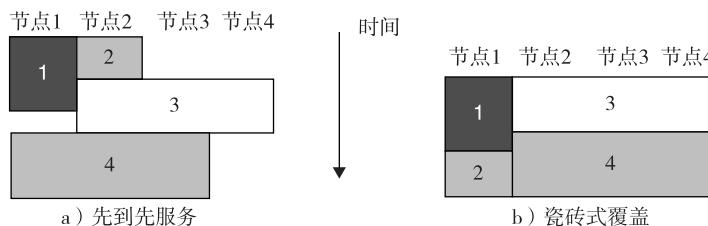


图 2-22 用于集群节点上作业调度的瓷砖式覆盖技术减少了整体时间，因此增加了作业吞吐量

注：由 Hwang 和 Xu^[14] 提供。

2.4.1.2 分时

在专用或者空间共享模式中，每个节点只分配了一个用户进程。然而，该节点上的系统进程或后台程序仍在运行。在分时模式中，多个用户进程被分配至相同的节点，引入了下列并行调度策略：

1. **独立调度** 分时的最直接实现如同传统的工作站，使用集群中每个节点的操作系统来调度不同的进程。这称为局部调度或独立调度。然而，这会导致并行作业的性能明显降低，因为执行并行作业的进程之间需要交互。例如，当一个进程试图与另一个进程界限同步，后者可能已被调度出去。于是第一个进程必须等待。而当第二个进程被重新调度时，第一个进程可能已被替换。

2. **组调度** 组调度机制共同调度并行作业的全部进程。一个进程活跃，则所有的进程都活跃。集群节点并不完全锁同步。事实上，大多数系统是异步系统，不由同一时钟驱动。虽然我们说：“所有进程将在同一时间运行，”但是它们不会十分精确地在同一时间开始。在第一个进程开始和最后一个进程开始之间，组调度倾斜具有最大差异。组调度倾斜越大，并行作业的运行时间就越长，从而导致更长的运行时间。我们应该使用同构集群，这样组调度更有效。然而，由于

实施困难，组调度在大多数集群中并未实现。

3. 与外界（本地）作业竞争 当集群作业和本地作业都运行时，调度变得更复杂。本地作业的优先级应高于集群作业。拥有者仅需击一次键，便可掌控所有工作站资源。处理这种情况一般有两种方法：集群作业可以驻留在工作站节点或者迁移到另一个空闲节点。驻留方案可以避免迁移开销，但集群进程以最低优先级运行。工作站周期可以分为三个部分：内核进程、本地进程和集群进程。然而，驻留降低了本地和集群作业的速度，尤其是当集群作业是一个需要频繁同步和通信的负载均衡的并行作业时。于是可以转移作业到可用周围节点，以平衡负载。

106

2.4.2 集群作业管理系统

作业管理也称为负载管理或负载共享。我们首先讨论作业管理系统面临的基本问题，并总结可用的软件包。作业管理系统（Job Management System, JMS）具有三部分：

- **用户服务器**：提交用户作业至一个或多个队列，为每个作业指定资源需求，将作业从队列中删除，以及询问作业或队列的状态。
- **作业调度器**：根据作业类型、资源需求、资源可用性和调度策略，执行任务调度和排队。
- **资源管理器**：分配和监控资源，执行调度策略，以及收集统计信息。

2.4.2.1 JMS 管理

JMS 的功能通常是分布的。例如，用户服务器可能在每个主机节点中，而资源管理器则可能跨越所有集群节点。然而，JMS 的管理应该是集中的，所有配置与日志文件应该维护在同一地点。并且需要一个单一用户界面以便使用 JMS。强制用户使用某一软件包运行 PVM 作业、使用另一软件包运行 MPI 作业，以及使用剩余的某一软件包运行 HPF 作业，是不受欢迎的。

JMS 应该能够在对运行作业产生最小影响的情况下，动态重新配置集群。管理员的开始和结束脚本应该能够在安全检查、统计和清除作业之前及之后运行。用户应该能够干净地清除他们自己的作业。管理员或 JMS 应该能够干净地暂停或清除任何作业。干净（clean）意味着当某一作业中止或死亡时，必须涵盖所有它的进程。否则，有些“孤儿”进程遗留在系统中，会浪费系统资源并且可能最终导致系统无法使用。

2.4.2.2 集群作业种类

一个集群上可以运行几类作业。串行作业在单个节点上运行。并行作业使用多个节点。交互作业需要快速的周转时间，并且其输入/输出指向一个终端。这些作业不需要大量资源，但用户期望作业被立即执行，不需要在队列中等待。批量作业通常需要更多的资源，如大内存空间和长 CPU 时间，但是它们不要求立即反馈结果。于是它们被提交至作业队列，当资源可用时（如在空闲时间），被调度执行。

相较于交互作业和批量作业由 JMS 管理，外界作业是在 JMS 之外被创建的。例如，当工作站网络作为集群时，用户可以向 JMS 提交交互作业或批量作业。同时，工作站的所有者可以在任意时间创建一个外界作业，该作业不通过 JMS 提交。这类作业也称为本地作业，相对于集群作业（交互或批量、并行或串行）。本地作业的特征是响应时间迅速。所有者希望所有资源执行他的作业，就好像集群作业不存在一样。

2.4.2.3 集群负载特征

为了解决作业管理问题，我们必须了解集群的工作行为。在实际集群上，基于长期操作数据来描述负载似乎是理想的。并行负载跟踪包括开发与生产作业。这些跟踪被输入一个模拟器，依据不同的串行与并行负载的组合、资源分配和调度策略，产生不同的统计结果和性能结果。下列负载特征基于 NAS 基准测试程序实验，当然，不同的负载可能有不同的统计结果。

- 约一半的并行作业在正常工作时间内提交。约 80% 的并行作业运行持续三分钟甚至更少；运行时间超过 90 分钟的并行作业占总时间的 50%。

107

- 串行负载显示，60% ~ 70% 的工作站可以在任意时间执行并行作业，即使在日间高峰时间。
- 在工作站中，53% 的空闲时间为 3 分钟或者更少，但是 95% 的空闲时间在 10 分钟之后才被使用。
- 2:1 法则，即一个包含 64 个工作站的网络，具有合适的 JMS 软件，除了原有的串行负载，能够维持一个 32 节点的并行负载。换句话说，集群化的一半被免费提供给超级计算机！

2.4.2.4 迁移机制

一个迁移机制必须考虑以下三个问题：

- **节点可用性** 这涉及作业迁移时的节点可用性。Berkeley NOW 项目声称大学校园环境内还存在这样的情形。即使在高峰时间，伯克利校园集群中 60% 的工作站是可用的。
- **迁移开销** 迁移开销的影响包括什么？迁移时间会显著影响并行作业运行。降低迁移开销（如通过提高通信子系统）或尽量少迁移是很重要的。如果一个并行作业运行在 2 倍规模的集群上，其减速时间将显著降低。例如，一个 32 节点的并行作业在一个 60 节点集群上运行，由迁移造成的减速时间不超过 20%，即使迁移有 3 分钟之长。这是因为多个节点是可用的，因此迁移需求减少了。
- **迁移阈值** 发生迁移的阈值应该是什么？最坏的情况下，当进程迁移到某一节点，该节点立即被它的所有者控制。因此，进程必须再次迁移，甚至不断循环下去。迁移阈值是在集群认为某工作站是空闲节点之前，该工作站闲置的时间。

2.4.2.5 JMS 期望特征

下面是集群计算应用中的一些特征，已应用于部分商业 JMS：

- 最大程度支持异构 Linux 集群，支持并行作业和批量作业。但是，Connect: Queue 不支持交互作业。
- 企业级集群作业由 JMS 管理。它们将影响运行本地作业的工作站所有者。然而，NQE 和 PBS 允许调整这类影响。在 DQS 中，该影响能够配置为最小。
- 软件包提供某种负载均衡机制以有效利用集群资源。某些软件包支持检查点。
- 大多数软件包不支持动态进程迁移。它们支持静态迁移：一个进程可以在其第一次被创建时，被派遣到远程节点上执行。然而，一旦它开始执行，它便驻留在那个节点中。Condor 是一个支持动态进程迁移的软件包。
- 所有软件包允许通过用户或管理员动态暂停和恢复用户作业。所有软件包允许动态添加或删除资源（如节点）。
- 大多数软件包提供命令行界面和图形用户界面。除了 UNIX 的安全机制外，大多数软件包还使用 Kerberos 认证系统。

2.4.3 集群计算的负载共享设备 (LSF)

LSF 是平台计算中的商用负载管理系统^[29]。在并行作业和串行作业中，LSF 强调作业管理和负载共享。此外，它还支持检查点、可用性、负载迁移和单系统镜像。LSF 具有高扩展性，并且能够支持上千个节点的集群。LSF 服务于各种 UNIX 和 Windows/NT 平台。目前，LSF 不仅在集群中使用，也在网格和云中使用。

2.4.3.1 LSF 体系结构

LSF 支持大多数 UNIX 平台，并采用标准 IP 进行 JMS 通信。正因为如此，它可以使 UNIX 计算机的异构网络成为一个集群，而没有必要修改潜在的操作系统内核。终端用户使用有效命令集合调用 LSF 功能。LSF 支持 PVM 和 MPI，提供命令行界面及 GUI，同时 LSF 也为熟练用户提供了名为 LSLIB（Load Sharing Library，负载共享库）的运行时库的 API。使用 LSLIB 明确要求用户

修改应用程序代码，而不是使用实用命令。集群中的每个服务器有两个 LSF 维护进程。负载信息管理器（Load Information Managers, LIM）定期交换负载信息。远程执行服务器（Remote Execution Server, RES）执行远程任务。

2.4.3.2 LSF 实用命令

集群节点可能是具有单处理器主机或多处理器的 SMP 节点，但总是只运行节点上操作系统的单一备份。下面是构建 LSF 设备的一些有趣特征：

- LSF 支持交互、批量、串行和并行作业的任意组合。不通过 LSF 执行的作业称为外界作业。服务器节点是可以运行 LSF 作业的节点。客户端节点是可以初始化或提交但不能执行 LSF 作业的节点。只有服务器节点的资源可以共享。服务器节点也可以初始化或提交 LSF 作业。
- LSF 为从 LSF 获取信息和远程执行作业提供了一套工具（lstoools）。例如，lshosts 列出了集群中每个服务器节点的静态资源（稍后讨论），命令 lsrun 用于执行远程节点上的程序。
- 当用户在客户端节点输入命令 %lsrun -R ‘swp > 100’ myjob，应用程序 myjob 将自动选择负载最轻的服务器节点运行，该节点的可用交换空间大于 100MB。
- 工具 lsbatch 允许用户通过 LSF 提交、监控及执行批量作业。该工具是常用的 UNIX 命令解释器 tcsh 的负载共享版本。一旦用户进入 lscsh shell，每个输入的命令将自动地在合适节点上运行。这是透明的：用户所看见的 shell，如同 tcsh 在本地节点上运行一样。[109]
- 工具 lsmake 是 UNIX 工具 make 的并行版本，使生成文件可在多个节点上同时执行。

例 2.13 计算机集群上的 LSF 应用

假设一个集群由 8 个昂贵的服务器节点和 100 个廉价的客户端节点（工作站或个人计算机）组成。服务器节点昂贵是因为它有更好的硬件和软件，包括应用软件。授权协议允许安装 FORTRAN 语言编译器和计算机辅助设计（CAD）模拟软件包，最多对 4 个用户有效。使用一个 JMS（如 LSF），服务器节点的所有硬件和软件资源都是透明地提供给客户端。

用户坐在客户终端前感觉本地客户端节点具有所有软件及服务器性能。输入 lsmake my.makefile，用户可以在 4 个服务器上编译他的源代码。因为 LSF 选择负载最少的节点，所以使用 LSF 也有益于资源利用。例如，如果用户想要运行 CAD 模拟，可以提交批量作业。一旦软件变得可用，LSF 将会调度这个作业。■

2.4.4 MOSIX: Linux 集群和云的操作系统

MOSIX 由希伯来大学在 1977 年开发，是一个分布式操作系统。最初，该系统扩展了 BSD/OS 系统调用，用于奔腾系列集群中的资源共享。在 1999 年，该系统被重新设计，运行在 x86 平台的 Linux 集群上。MOSIX 项目在 2011 年仍然活跃，这些年共发布了 10 个版本。最新的版本 MOSIX2 与 Linux 2.6 兼容。

2.4.4.1 Linux 集群的 MOXIS2

MOSIX2 是运行在 Linux 环境中的虚拟化层。该层利用运行时 Linux 支持，为用户和应用程序提供单系统镜像。该系统运行远程节点上的应用程序，就如在本地一样。它支持串行和并行应用程序，并且可以发现资源，在 Linux 节点之间透明地自动迁移软件进程。MOSIX2 也可以管理 Linux 集群或多集群网格。

网格的灵活管理允许集群所有者可以在多集群所有者之间共享计算资源。每个集群仍保有自治权，可以在任意时间从网格中断开自己的节点。这应当在不影响正在运行的程序的前提下完成。许可 MOSIX 的网格只要集群所有者间相互信任，就可以无限扩展。其条件是需要保证正运行于远程集群的客户应用程序不能够被修改。不友善的计算机不允许连接到本地网络。

2.4.4.2 MOSIX2 中的 SSI 特征

系统能以本机模式或作为虚拟机运行。在本机模式中，其性能较好，但是它要求修改基本 Linux 内核；而虚拟机可运行于支持虚拟化且不作任何修改的操作系统中，包括微软 Windows、

[110] Linux 和 Mac OS X。此系统中最适合运行具有少量或中等数量 I/O 操作的计算密集型应用程序。

MOSIX2 的测试表明在 1GB/s 校园网格中的若干应用程序性能和单个集群上的几乎相同。下面是 MOSIX2 的部分有趣特征：

- 用户可以从任何节点登录，并不需要知道程序的运行位置。
- 没有必要修改应用程序或链接应用程序至特殊库。
- 没有必要复制文件至远程节点，这是因为进程迁移可以自动发掘资源和分配负载。
- 用户能够平衡负载，将进程从较慢节点迁移至快速节点，并从内存耗尽的节点中迁移出进程。
- 关于迁移进程直接通信的套接字也是可迁移的。
- 该系统以客户进程的安全运行时环境为特征。
- 该系统能够运行批量作业，并可以通过检查点恢复，这由自动安装和配置脚本的工具实现。

2.4.4.3 高性能计算机上 MOSIX 的应用

MOSIX 是用于高性能计算机集群、网格和云计算的研究型操作系统。该系统的设计者声称 MOSIX 通过自动资源发现和负载均衡提供了广域网格资源的高效利用。长进程被自动分配到网格节点，在运行长进程可能造成不可预测的资源请求或运行时间的情况下，该系统可以运行应用程序。因为节点负载索引和可用内存的不同，在节点间迁移进程时，该系统也能够结合具有不同能力的节点。

MOSIX 在 2001 年成为专利软件。科学计算的应用实例包括基因序列分析、分子动力学、量子动力学、纳米技术和其他并行高性能计算机应用；工程上的应用包括 CFD、天气预报、碰撞模拟、石油工业模拟、ASIC 设计和药物设计；以及云应用，如金融建模、渲染车间和编译车间。

例 2.14 使用 MOSIX 和 PVM 的内存引导算法

当本地节点的主内存耗尽时，内存引导借用远程集群节点的主内存。远程节点的加入可通过进程迁移实现，而不用对本地磁盘进行分页或交换。引导进程可由 PVM 命令实现，也可以使用 MOSIX 进程迁移。在每一次执行中，平均内存块可以分配至使用 PVM 的节点。图 2-23 显示了使用 PVM 和 MOSIX 的内存算法所需执行时间的比较。

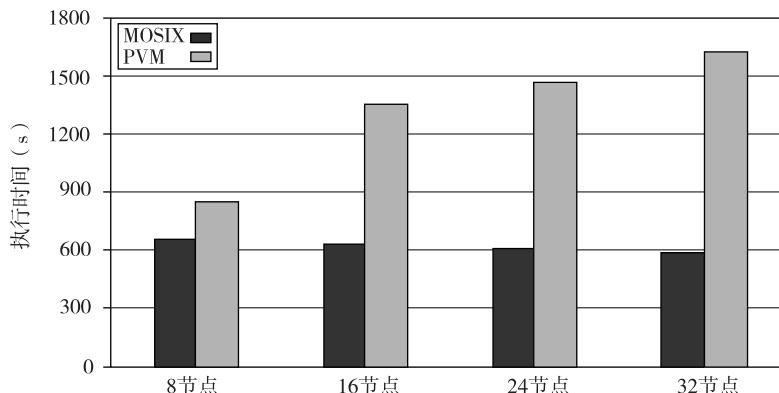


图 2-23 使用 MOSIX 或 PVM 的内存引导算法的性能

注：由 A. Barak 和 O. La'adan^[5] 提供。

对于小型的 8 节点集群，它们的执行时间相对接近。当集群规模扩展至 32 节点，MOSIX 程序在引导时间上减少了 60%。此外，当集群规模增长时，MOSIX 性能几乎相同。PVM 引导时间以平均每节点 3.8% 的速度单调上升，而 MOSIX 以每节点 0.4% 的速度持续下降。该实验结果证实了 MOSIX 的内存和负载均衡算法比 PVM 有更高的扩展性。[11]

2.5 顶尖超级计算机系统的个案研究

本节回顾了三个领先超级计算机，这三个超级计算机在 2008—2010 年已被选入 Top500 名单。IBM Roadrunner 是世界上第一个千万亿次计算机，在 2008 年排名第一。之后，Cray XT5 Jaguar 在 2009 成为领先系统。2010 年 11 月，中国 Tianhe-1A 成为世界上最快的系统。这 3 个系统均采用 Linux 集群结构，通过大量可同时运行的计算节点实现大规模并行。

2.5.1 Tianhe-1A：2010 年的世界最快超级计算机

2010 年 11 月，2010 ACM Supercomputing Conference 宣布 Tianhe-1A 为混合超级计算机。该系统在 Linpack 基准测试中，显示了 2.507 Pflops 的持续速度，因此成为 2010 年 Top500 名单中排名第一的超级计算机。该系统由中国国防科学技术大学（NUDT）开发，于 2010 年 8 月安装于天津国家超级计算机中心（NSC，www.nscc.tj.gov.cn）。该系统计划用来作为研究和教育的开放平台。图 2-24 显示了安装在 NSC 的 Tianhe-1A 系统。



图 2-24 Tianhe-1A 系统，由中国国防科学技术大学构建，安置在天津国家超级计算机中心（2011）[11]

2.5.1.1 Tianhe-1A 体系结构

图 2-25 显示了 Tianhe-1A 系统的精简体系结构。系统由 5 个主要部件组成。计算子系统覆盖了 7 168 个节点上的所有 CPU 和 GPU。服务子系统包含 8 个操作节点。存储子系统拥有一大批共享磁盘。监测与诊断子系统用于控制和 I/O 操作。通信子系统由连接所有功能性子系统的交换机组成。

2.5.1.2 硬件实现

该系统采用了 14 336 个 2.93GHz 的六核 Xeon E5540/E5450 处理器及 7 168 个 NVIDIA Tesla M2050s。它具有 7 168 个计算节点，每个节点由两个 2.93GHz 的六核 Intel Xeon X5670 (Westmere) 处理器和一个 NVIDIA M2050 GPU 通过 PCI-E 连接组成。每片有两个节点，高度为 2U (图 2-25)。整个系统有 14 336 个 Intel 插板 (Westmere)、7 168 个 NVIDIA Fermi 板和 2 048

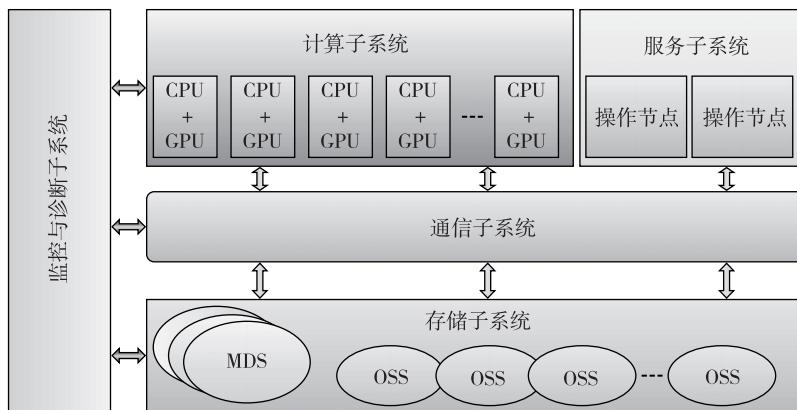


图 2-25 Tianhe-1A 的精简体系结构

个 Galaxy 插板（基于 Galaxy 处理器的节点用于系统的前端处理）。每个计算节点有两个 Intel 插板、一个 Fermi 板和 32GB 的内存。

整个系统的理论峰值为 4.7Pflops/s, 如图 2-26 中的计算所示。值得注意的是, 每个 GPU 节点中有 448 个 CUDA 核。峰值速度应该综合考虑 14 236 个 Xeon CPU (共有 380 064 个核) 和 7 168 个 Tesla GPU (每个节点有 448 个 CUDA 核, 总共有 3 496 884 个 CUDA 核)。CPU 和 GPU 芯片共有 3 876 948 个处理器核。一个操作节点拥有两个 8 核 Galaxy 芯片 (1GHz, SPARC 体系结构) 与 32GB 内存。Tianhe-1A 系统被封装为 112 个计算机柜、12 个存储机柜、6 个通信机柜和 8 个 I/O 机柜。

操作节点由两个 8 核 Galaxy FT-1000 芯片组成。这些处理器由 NUDT 设计，频率为 1GHz。八核芯片的理论峰值是 8Gflops/s。整个系统拥有 1 024 个这样的操作节点，每个节点具有 32GB 的内存。这些操作节点主要用于功能性操作，而服务节点用于作业创建和提交，它们并不是作为通用目的计算节点。其速度并不影响峰值或持续计算速度。Tianhe-1A 的峰值速度为 3.692 Pflops^[11]。它在 4 个子系统中共采用了 7 168 个计算节点（448 个 CUDA 核/GPU/计算节点），14 236 个六核 CPU。

$$\begin{aligned} & 7168(\text{节点}) \times 2(\text{CPU}) \times 2.93(\text{GHz}) \times 6(\text{核}) \times 4 \\ & = 1.008 \text{Pflops} \\ + & \\ & 7168(\text{节点}) \times 1(\text{GPU}) \times 1.15(\text{GHz}) * 448(\text{CUDA 核}) \\ & = 3.692 \text{Pflops} \end{aligned}$$

总计：
4 701 061GFlops

图 2-26 Tianhe-1A 理论峰值速度的计算

该系统通过 Lustre 集群文件系统实现了总共 2PB 的磁盘存储。该集群系统中共分布了 262TB 的主内存。Tianhe-1A 体现了现代异构 CPU/GPU 计算，在性能、规模和电能方面取得显著成绩。该系统原本只使用 CPU，需要 50 000 多个 CPU 以及 2 倍工作空间来实现相同的性能。一个 2.507 Pflops 的系统全部使用 CPU 的功率至少为 1 200 万瓦特，这比 Tianhe-1A 的功率消耗多 3 倍。

2.5.1.3 胖树互连体系结构

Tianhe-1A 的高性能归功于由 NUDT 定制设计的互连体系结构。该体系结构使用 InfiniBand DDR 4X 和 97TB 的内存。胖树体系结构如图 2-27 所示。双向带宽为 160Gbps，大约是相同数目节点上 QDR InfiniBand 网络带宽的 2 倍。该结构有 1.57 微秒的节点跳跃延迟和 61Tb/s 的集群化带宽。在胖树结构的第一层，16 个节点由一个 16 口的交换板连接。在第二层，所有部件都连接到 11 个 384 口的交换机。路由器和网络接口芯片由 NUDT 团队设计。

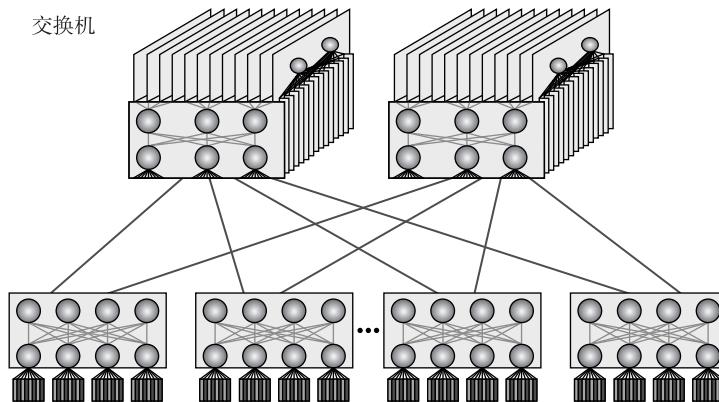
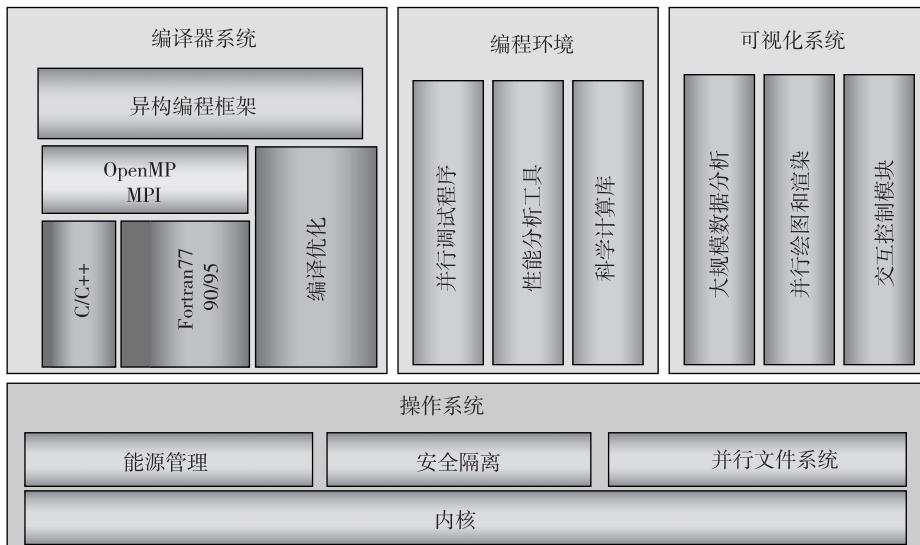


图 2-27 高带宽交换两个级别的胖树互连体系结构

2.5.1.4 软件栈

Tianhe-1A 上的软件栈在高性能系统中具有代表性。它使用由 NUDT 开发的操作系统 Kylin Linux，这在 2006 年顺利通过中国 863 高新技术研究与发展项目办公室的批准。Kylin 基于 Mach 和 FreeBSD，兼容其他主流操作系统，并且支持多个微处理器以及不同结构的计算机。Kylin 软件包包括标准开源和公共软件包，使得系统便于安装。图 2-28 描述了 Tianhe-1A 的软件体系结构。

图 2-28 Tianhe-1A 超级计算机的软件体系结构^[11]

该系统具有来自 Intel (icc 11.1)、CUDA、OpenMP 的 FORTRAN、C、C++ 和 Java 编译器，并得到 MPICH2 与 GLEX (Galaxy Express) 通道的支持。NUDT 建立者开发了数学库，该数学库基于 Intel 的 MKL 10.3.1.048 和 NVIDIA GPU 上的 BLAS，并由 NUDT 优化。此外，高效并行运行环境 (HPPRE) 也被安装。系统也提供了基于 Eclipse 的并行工具包，用于整合各种编辑、调试和性能分析工具。此外，设计师提供了对服务质量 (QoS) 评判和资源预定的工作流支持。

2.5.1.5 能耗、空间和成本

Tianhe-1A 能耗在轻负载时为 4.04MW (兆瓦)。该系统占地面积 700 平方米，并由采用加压气流的分体水冷却系统降温。该混合体系结构消耗较少的电能——大约是 12MW 的三分之一，

其中 12MW 是整个系统全采用多核 CPU 的能耗。整个系统的预算是 6 亿人民币（约 9 000 万美元）；2 亿元人民币来自于科学技术部（MOST），4 亿元来自于天津当地政府。每年大约需要 2 000 万美元的费用来运行、维护和冷却该系统。

2.5.1.6 Linpack 基准测试结果和应用计划

在 2010 年 10 月 30 日，Tianhe-1A 的 Linpack 基准测试性能为 2.566 Pflops/s，其中规模为 [115] 3 600 000， $N_{1/2} = 1 000 000$ 。系统运行的总时间是 3 小时 22 分钟，效率为 54.58%，远低于 Jaguar 和 Roadrunner 的 75% 的效率。下面是 Tianhe-1A 的一些应用，大部分经过特别设计以满足国家的需要。

- 并行 AMR (Adaptive Mesh Refinement, 自适应网格细化) 方法。
- 并行特征值问题。
- 并行快速多级方法。
- 并行计算模型。
- Gridmol 计算化学。
- ScGrid 中间件、网格门户。
- PSEPS 并行对称特征值封装解决。
- 雷达截面的 FMM-radar 快速多级方法。
- 移植大量开源软件程序。
- 沙尘暴预测、气候模型、电磁散射或宇宙学。
- 汽车制造业中的 CAD/CAE。

2.5.2 Gray XT5 Jaguar: 2009 年的领先超级计算机

2010 年 6 月，在 ACM 超级计算会议发布的 Top500 名单中，Cray XT5 Jaguar 是世界上最快的超级计算机。在 2010 年 11 月发布的 Top500 名单中，Cray XT5 Jaguar 被中国 Tianhe-1A 取代，变为快速超级计算机的第二名，它是由 Gray 股份有限公司构建的可扩展 MPP 系统，Jaguar 属于 [116] Gray 的系统模型 XT5-HE。系统安装在隶属于美国能源部门的橡树岭国家实验室，整个 Jaguar 系统共有 86 个机柜。下面是 Jaguar 系统的一些有趣体系结构和操作特征：

- 由 AMD 六核 Opteron 处理器组成，以 2.6GHz 的时钟频率运行 Linux。
- 总共有 224 162 个处理器核，超过 37 360 个处理器，分布于 4 排共 88 个机柜中（每个机柜有 1 536 或 2 304 个处理器）。
- 8 256 个计算节点和 96 个服务节点通过 3D 环形网络互连，环形网络由 Cray SeaStar2 + 芯片组成。
- 其持续速度 R_{\max} ，在 Linpack 基准测试中为 1.759Pflops。
- 最大的 Linpack 测试矩阵规模记录为 $N_{\max} = 5 474 272$ 未知。

系统的基本构建模块是计算叶片。SeaStar + 芯片（见图 2-29）中的互连路由器提供了 3D 环形网络中的 6 条高速链接，如图 2-30 所示。该系统可由小型配置扩展至大型配置。整个系统具有 129TB 的计算内存。理论上，系统的峰值速度为 $R_{\text{peak}} = 2.331 \text{ Pflops}$ 。换句话说，Linpack 实验只实现了 75% (= 1.759/2.331) 的效率。对外 I/O 接口使用 10Gbps 以太网和 InfiniBand 连接。消息传递编程使用 MPI 2.1。系统能耗为 32 ~ 43 千瓦/机柜。当有 160 个机柜时，整个系统的能耗高达 6.950MW。系统使用强制冷风降温，这也会消耗大量电能。

2.5.2.1 3D 环形互连

图 2-30 显示了系统的互连体系结构。Gray XT5 系统使用 Cray SeaStar2 + 路由器芯片集成高带宽低延迟互连。该系统由具有 8 个插槽的 XT5 计算叶片配置而成，并支持双核或四核 Opterons。XT5 使用 3D 环形网络拓扑结构，SeaStar2 + 芯片提供了 6 个高速网络链接，连接 3D 环形网络中的 6 个邻居。每个连接的双向峰值带宽是 9.6GB/s，持续带宽超过 6GB/s。每个端口由独立的路

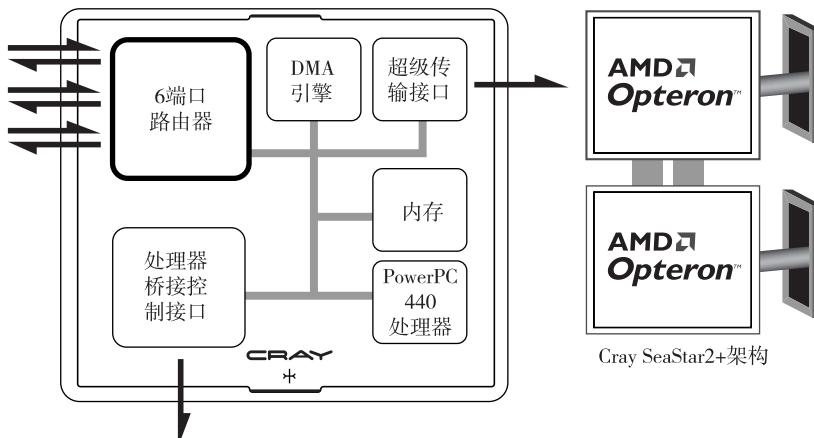


图 2-29 Cray XT5 Jaguar 超级计算机的内连 SeaStar 路由器芯片设计

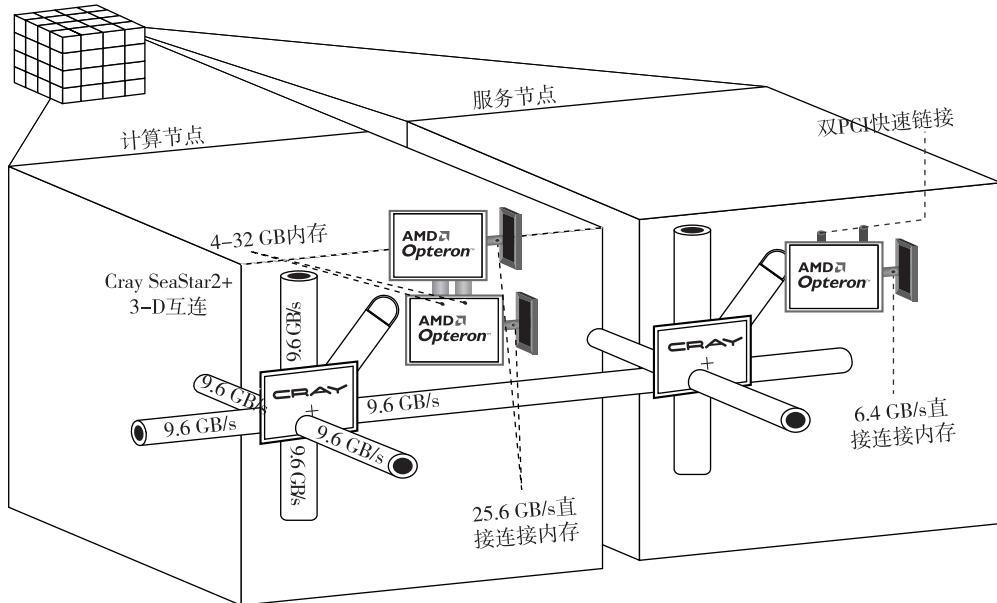


图 2-30 Gray XT5 Jaguar 超级计算机的 3D 环形互连

注：由 Gray 公司^[9]和美国橡树岭国家实验室提供，2009。

由表配置，保证无冲突存取数据包。

路由器采用支持纠错和重传的可靠链路层协议，保证消息传递通信到达它的目的地，而不采用典型集群中的暂停和重试机制。环形互连直接连接 Cray XT5 系统中的所有节点，降低了外部交换的成本与复杂性，并且容易扩展。这允许系统很好地扩展为成千上万个节点，远远超越胖树交换的能力。互连实现了在全局文件系统的消息传递和 I/O 通信。

2.5.2.2 硬件封装

Cray XT5 系列使用了节能封装技术，减少能源消耗，从而降低维护成本。系统的计算叶片只封装了必需组件，使用处理器、内存和互连构建 MPP。在 Gray XT5 机柜中，垂直冷却直接从源头（地面）接收冷空气，并有效地冷却叶片上的处理器，叶片位于最佳气流的特定位置。每个处理器也有定制的散热器，在机柜内依附于它的旁边。每个 Gray XT5 系统机柜使用单个高效的小型涡轮风扇降温，它需要直接从电网获取 400/480VAC，以避免变压器和 PDU 损失。

Cray XT5 3D 环形体系结构被设计用于实现 HPC 应用的优秀 MPI 性能。这通过集成专用的计算节点和服务节点来实现。计算节点用于高效运行 MPI 任务，并可靠地完成这些任务。每个计算节点由一个或两个 AMD Opteron 微处理器（双或四核）、直接连接内存和专用通信资源组成。服务节点用来向系统提供 I/O 连通性，并如同登录节点一样，协助作业编译与启动。每个计算节点的 I/O 带宽被设计为 25.6 GB/s。

2.5.3 IBM Roadrunner: 2008 年的领先超级计算机

2008 年，IBM Roadrunner 是世界上第一台达到千万亿次性能的通用计算机系统。该系统 Linpack 性能为 1.456 Pflops，安装在新墨西哥州的洛斯阿拉莫斯国家实验室（LANL）。随后，在 2009 年末，Cray 的 Jaguar 超越了 Roadrunner。IBM Roadrunner 主要用于评估美国核武库的衰变，其系统为混合设计，具有 12 960 个 IBM 3.2 GHz PowerXcell 8i CPU（图 2-31）和 6 480 个 AMD 1.8 GHz Opteron 2210 双核处理器。系统共有 122 400 个处理器核。Roadrunner 是一个 Opteron 集群，由 8 个浮点核的 IBM Cell 处理器加速。

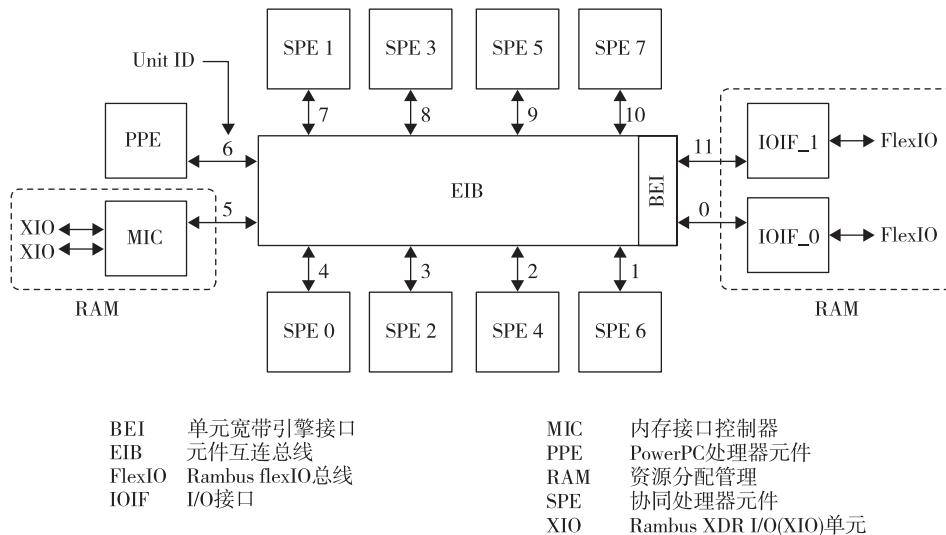


图 2-31 IBMCell 处理器体系结构的原理图

注：由 IBM 提供，<http://www.redbooks.ibm.com/redpapers/pdfs/redp4477.pdf>^[28]。

2.5.3.1 处理器芯片和计算叶片设计

Cell/B. E. 处理器提供非凡的计算能力，可以利用单一多核芯片。如图 2-31 所示，Cell/
[119] B. E. 体系结构支持非常广泛的应用。单芯片多处理器实现了 9 个处理器元件运行在一个共享内存模型上。TriBlade 服务器通过 InfiniBand 网络连接，组成了机架。为了保持这种计算能力，每个节点的连通由 4 个 PCI Express x8 链接组成，每个链接具有 2GB/s 传送速率，并有 2μs 的延迟。该扩展槽还包括 InfiniBand 互连，允许与集群的其余部分通信。InfiniBand 互连的速度为 2GB/s，延迟为 2μs。

2.5.3.2 InfiniBand 互连

Roadrunner 集群是层次结构的。InfiniBand 为集群提供交换功能，连接 270 个机架中的 18 个单元。总之，整个集群连接 12 960 个 IBM Power XCell 8i 处理器和 6 480 个 Opteron 2210 处理器，共有 103.6TB 的 RAM。此集群提供了约 1.3 Pflops 的性能。此外，系统的 18 个 Com/Service 节点使用 18 个 InfiniBand 交换机，提供了 4.5 Tflops 的性能。第二存储单元采用 8 个 InfiniBand 交换机连接。总体上，系统中共安装了 296 个机架。分层结构按两个级别构造。该系统消耗 2.35 MW

的电能，名列 2009 年建造的最节能超级计算机的第 4 位。

2.5.3.3 消息传递性能

Roadrunner 使用 MPI API 与其他 Opteron 处理器通信，应用程序以典型的单程序多数据（SPMD）形式运行。运行此应用程序的计算节点数目在程序启动时确定。Roadrunner 的 MPI 实现是基于开源 Open MPI 项目的，因此是标准 MPI。在这方面，Roadrunner 应用程序与其他典型 MPI 应用类似，如那些运行于 IBM Blue Gene 的解决方案。Roadrunner 与应用体系结构领域的不同之处是如何使用 Cell/B. E. 加速器。在应用流的任何时候，每个 Opteron 上运行的 MPI 应用程序能够分发复杂计算逻辑至其下属 Cell/B. E. 处理器。

2.6 参考文献和习题

自 1990 年以来，集群计算一直是热点研究领域。集群计算由 DEC 和 IBM 倡导，发表于 Pfister[26]。该书提供了若干关键概念的很好介绍，包括 SSI 和 HA。历史上，计算机集群的里程碑包括运行 VMS/OS 的 VAXcluster（1984）、Tandem Himalaya HA 集群（1994）和 IBM SP2 集群（1996）。这些早期的集群在文献[3, 7, 14, 23, 26]有所介绍。近年来，超过 85% 的 Top500 系统都采用了集群配置^[9,11,20,25,28,29]。

每一年，IEEE 和 ACM 举行几次国际会议讨论相关课题。它们包括“Cluster Computing”（Cluster），“Supercomputing Conference”（SC），“International Symposium on Parallel and Distributed Systems”（IPDPS），“International Conferences on Distributed Computing Systems”（ICDCS），“High-Performance Distributed Computing”（HPDC）和“Clusters, Clouds, and The Grids”（CCGrid）。同时也有与此课题相关的几个杂志，包括《Journal of Cluster Computing》、《Journal of Parallel and Distributed Computing》（JPDC）和《IEEE Transactions on Parallel and Distributed Systems》（TPDS）。

Bader 与 Pennington^[2]评测了集群应用程序。本章部分数据和实例从 Hwang 和 Xu^[14]之前的书中修改得来。Buyya 将集群计算分为两个章节编辑^[7]。关于 Linux 集群的两本书是文献[20, 23]。[120] HA 集群在文献[24]中有介绍。最近的 HPC 互连评估可以在文献[6, 8, 12, 22]中找到。谷歌集群互连由 Barroso 等人发表^[6]。超级计算机的 GPU 在文献[10]中讨论。GPU 集群在文献[19]中介绍。GPU 的 CUDA 并行编程可参见文献[31]。集群或网格计算的 MOSIX/OS 可参见文献[4, 5, 30]。

Hwang、Jin 和 Ho 开发了一种分布式 RAID 系统，以实现个人计算机或工作站集群的单一I/O 空间^[13~17]。LSF 的更多细节可以参见 Zhou[35]。Top500 名单采用了 2010 年 6 月和 11 月的发布结果^[25]。Tianhe-1A 的资料可以在 Dongarra[11] 和 Wikipedia[29] 上找到。IBM Blue Gene/L 体系结构由 Adiga 等人发表^[1]，随后被设计为更新的模型，称为 Blue Gene/P 解决方案。IBM Roadrunner 由 Kevin 等人发布^[18]，也可以参见 Wikipedia[28]。Cray XT5 和 Jaguar 系统在文献[9] 中描述。中国 Nebulae 超级计算机发表于文献[27]。具体的集群应用程序和检查点技术可以参见文献[12, 16, 17, 24, 32, 34]。集群应用可以参见文献[7, 15, 18, 21, 26, 27, 33, 34]。

致谢

这一章由南加州大学的黄铠教授和田纳西大学的 Jack Dongarra 共同撰写。部分集群资料出于黄铠教授和中国科学院徐志伟研究员之前出版的《可扩展并行计算：技术、结构与编程》一书^[14]。墨尔本大学的 Rajkumar Buyya 对这些资料更新提出了具有价值的建议。

本章由清华大学的武永卫教授负责翻译。

参考文献

- [1] N. Adiga, et al., An overview of the blue gene/L supercomputer, in: ACM Supercomputing Conference 2002, November 2002, <http://SC-2002.org/paperpdfs/pap.pap207.pdf>.
- [2] D. Bader, R. Pennington, Cluster computing applications, *Int. J. High Perform. Comput.* (May) (2001).
- [3] M. Baker, et al., Cluster computing white paper. <http://arxiv.org/abs/cs/0004014>, January 2001.
- [4] A. Barak, A. Shiloh, The MOSIX Management Systems for Linux Clusters, Multi-Clusters and Clouds. White paper, www.MOSIX.org/txt_pub.html, 2010.
- [5] A. Barak, R. La'adan, The MOSIX multicomputer operating systems for high-performance cluster computing, *Future Gener. Comput. Syst.* 13 (1998) 361–372.
- [6] L. Barroso, J. Dean, U. Holzle, Web search for a planet: The Google cluster architecture, *IEEE Micro.* 23 (2) (2003) 22–28.
- [7] R. Buyya (Ed.), *High-Performance Cluster Computing*. Vols. 1 and 2, Prentice Hall, New Jersey, 1999.
- [8] O. Celebioglu, R. Rajagopalan, R. Ali, Exploring InfiniBand as an HPC cluster interconnect, (October) (2004).
- [9] Cray, Inc, CrayXT System Specifications. www.cray.com/Products/XT/Specifications.aspx, January 2010.
- [10] B. Dally, GPU Computing to Exascale and Beyond, Keynote Address, ACM Supercomputing Conference, November 2010.
- [11] J. Dongarra, Visit to the National Supercomputer Center in Tianjin, China, Technical Report, University of Tennessee and Oak Ridge National Laboratory, 20 February 2011.
- [12] J. Dongarra, Survey of present and future supercomputer architectures and their interconnects, in: International Supercomputer Conference, Heidelberg, Germany, 2004.
- [13] K. Hwang, H. Jin, R.S. Ho, Orthogonal striping and mirroring in distributed RAID for I/O-Centric cluster computing, *IEEE Trans. Parallel Distrib. Syst.* 13 (2) (2002) 26–44.
- [14] K. Hwang, Z. Xu, Support of clustering and availability, in: *Scalable Parallel Computing*, McGraw-Hill, 1998, Chapter 9.
- [15] K. Hwang, C.M. Wang, C.L. Wang, Z. Xu, Resource scaling effects on MPP performance: STAP benchmark implications, *IEEE Trans. Parallel Distrib. Syst.* (May) (1999) 509–527.
- [16] K. Hwang, H. Jin, E. Chow, C.L. Wang, Z. Xu, Designing SSI clusters with hierarchical checkpointing and single-I/O space, *IEEE Concurrency* (January) (1999) 60–69.
- [17] H. Jin, K. Hwang, Adaptive sector grouping to reduce false sharing of distributed RAID clusters, *J. Clust. Comput.* 4 (2) (2001) 133–143.
- [18] J. Kevin, et al., Entering the petaflop era: the architecture of performance of Roadrunner, www.c3.lanl.gov/~kei/mypubbib/papers/SC08:Roadrunner.pdf, November 2008.
- [19] V. Kindratenko, et al., GPU Clusters for High-Performance Computing, National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, Urbana, IL, 2009.
- [20] K. Kopper, *The Linux Enterprise Cluster: Building a Highly Available Cluster with Commodity Hardware and Free Software*, No Starch Press, San Francisco, CA, 2005.
- [21] S.W. Lin, R.W. Lau, K. Hwang, X. Lin, P.Y. Cheung, Adaptive parallel Image rendering on multiprocessors and workstation clusters. *IEEE Trans. Parallel Distrib. Syst.* 12 (3) (2001) 241–258.
- [22] J. Liu, D.K. Panda, et al., Performance comparison of MPI implementations over InfiniBand, Myrinet and Quadrics, (2003).
- [23] R. Lucke, *Building Clustered Linux Systems*, Prentice Hall, New Jersey, 2005.
- [24] E. Marcus, H. Stern, *Blueprints for High Availability: Designing Resilient Distributed Systems*, Wiley.
- [25] TOP500.org. Top-500 World's fastest supercomputers, www.top500.org, November 2010.
- [26] G.F. Pfister, *In Search of Clusters*, second ed., Prentice-Hall, 2001.
- [27] N.H. Sun, China's Nebulae Supercomputer, Institute of Computing Technology, Chinese Academy of Sciences, July 2010.
- [28] Wikipedia, IBM Roadrunner. http://en.wikipedia.org/wiki/IBM_Roadrunner, 2010, (accessed 10.01.10).
- [29] Wikipedia, Tianhe-1. <http://en.wikipedia.org/wiki/Tianhe-1>, 2011, (accessed 5.02.11).
- [30] Wikipedia, MOSIX. <http://en.wikipedia.org/wiki/MOSIX>, 2011, (accessed 10.02.11).
- [31] Wikipedia, CUDA. <http://en.wikipedia.org/wiki/CUDA>, 2011, (accessed 19.02.11).

- [32] K. Wong, M. Franklin, Checkpointing in distributed computing systems, *J. Parallel Distrib. Comput.* (1996) 67–75.
- [33] Z. Xu, K. Hwang, Designing superservers with clusters and commodity components. *Annual Advances in Scalable Computing*, World Scientific, Singapore, 1999.
- [34] Z. Xu, K. Hwang, MPP versus clusters for scalable computing, in: *Proceedings of the 2nd IEEE International Symposium on Parallel Architectures, Algorithms, and Networks*, June 1996, pp. 117–123.
- [35] S. Zhou, LSF: Load Sharing and Batch Queuing Software, Platform Computing Corp., Canada, 1996.

习题

- 2.1 区分并举例说明以下集群的相关术语：
 - a. 紧凑集群和松弛集群
 - b. 集中式集群和分散集群
 - c. 同构集群和异构集群
 - d. 封闭集群和开放集群
 - e. 专用集群和企业集群
- 2.2 本题与冗余技术相关。假设当一个节点失效时，它需要 10s 诊断故障，并需要 30s 转移其负载。
 - a. 如果忽略计划停机时间，集群的可用性如何？
 - b. 如果集群每周需要一小时的停机维护时间，但每次只有一个节点，集群的可用性又如何？
- 2.3 这是一个评测近年来构建的 4 个超级计算机的集群体系结构的研究项目。研究 2010 年 11 月发布的 Top500 名单中排名第 1 的超级计算机（即 Tianhe-1A）的相关细节。你的研究应包括以下内容：
 - a. 深入评估 Tianhe-1A 体系结构、硬件组成、操作系统、软件支持、并行编译器、封装、冷却和新型应用。
 - b. 比较 Tianhe-1A 与在 2.5 节中介绍的 Jaguar、Nebulae 和 Roadrunner，分析 Tianhe-1A 的相对优势和局限性。如果你找到充足的基准测试数据进行比较研究，可以用电子表格或曲线图表示。
- 2.4 本题包括与集群计算相关的两个部分：
 1. 定义并区分下列可扩展性术语：
 - a. 机器规模上的可扩展性
 - b. 问题规模上的可扩展性
 - c. 资源可扩展性
 - d. 世代可扩展性
 2. 解释三种可用集群配置（热备份、主动接管和容错集群）之间体系结构和功能差异。给出每种可用集群配置的两个商用集群系统示例。评价在商业应用中，它们的相对优势和缺点。
- 2.5 基于结构、资源共享和处理器间通信，说明多处理器和多计算机之间的区别。
 - a. 解释 UMA、NUMA、COMA、DSM 和 NORMA 内存模型间的差异。
 - b. 相较于传统的自主计算机网络，集群有什么额外的功能特征？
 - c. 传统 SMP 服务器上的集群系统有什么优点？
- 2.6 研究表 2-6 列出的 5 个虚拟集群研究项目，并根据 2.5.3 节和 2.5.4 节中 COD 和 Violin 的相关经验回答下列问题：
 - a. 从动态资源供应的角度，根据公开的文献评估这 5 个虚拟集群，并讨论它们的相对优势与缺点。
 - b. 记录这 5 个虚拟集群在硬件配置、软件工具和开发实验环境，以及发布的性能结果这些方面的独特贡献。
- 2.7 本题涉及在构建 HPC 系统过程中，高端 x86 处理器的使用。回答下列问题：
 - a. 根据最新的超级计算机系统 Top500 名单，所有的系统均使用 x86 处理器。识别处理器模型和关键处理器特性，如核的数目、时钟频率和预计性能。
 - b. 有些采用 GPU 来补充 x86 CPU。识别这些具有大量 GPU 的系统。讨论在提供 flops/美元的峰值或者持续速度过程中，GPU 所起的作用。
- 2.8 假设一个顺序计算机具有 512MB 的主内存和足够的磁盘空间。对大规模数据块，其磁盘读/写带宽是

1MB/s。下面的代码需要应用检查点：

```
do 1000 iterations
A = foo (C from last iteration)      /* this statement takes 10 minutes */
B = goo (A)                          /* this statement takes 10 minutes */
C = hoo (B)                          /* this statement takes 10 minutes */
end do
```

A、B 和 C 分别都是 120MB 的数组。所有代码的其余部分、操作系统和库，最多占用 16MB 的内存。假设计算机恰好失效一次，并且忽略恢复计算机的时间。

- a. 如果执行检查点，成功运行该代码的最坏执行时间是多少？
 - b. 如果简单透明地执行检查点，成功运行该代码的最坏执行时间又是多少？
 - c. 在 (b) 中使用分支检查点是否有帮助？
 - d. 如果执行用户指导的检查点，该代码的最坏执行时间是多少？在代码中显示加入用户指示的位置。
 - e. 如果在 (b) 中使用分支检查点，该代码的最坏执行时间是多少？
- 2.9 比较最新 Top500 名单和 HPC 系统的 Top500 绿色名单。讨论在能源效率和冷却成本方面的小部分赢家与输家。显示绿色能源赢家的故事，并记录使他们获胜的特殊设计特点、封装、冷却及管理策略。为什么两张名单的排名顺序是不同的？根据公开的资料，讨论其原因和意义。
- 2.10 本题关系到为了构建最新 Top500 名单中三大具有商业互连的集群系统，所使用的处理器和系统互连。
- a. 在潜在峰值浮点性能方面，比较这些集群使用的处理器，并且说明它们的优势和缺点。
 - b. 比较这三个集群的商业互连。讨论在拓扑属性、网络延迟、对分带宽和硬件使用方面，它们的潜在性能。
- 2.11 学习例 2.6 及其原始文章 [14] 发表的分布式 RAID-x 体系结构和性能结果。根据技术上理由或证据回答下列问题：
- a. 解释在连接到集群节点的分布式磁盘上，RAID-x 系统如何实现单 I/O 地址空间。
 - b. 解释在 RAID-x 系统中实现的协同磁盘驱动器 (CCD) 的功能。基于目前的 PC 体系结构、SCSI 总线和 SCSI 磁盘技术，评论它的应用需求和可扩展性。
 - c. 解释为什么 RAID-x 具有和 RAID-5 体系结构相同的容错能力。
 - d. 通过与其他 RAID 体系结构比较，解释 RAID-x 的优势和局限性。
- 2.12 根据 2.2 节和 2.5 节的相关材料，比较 2009 年 11 月发布的 Top500 名单中，IBM Blue Gene/L、IBM Roadrunner 和 Cray XT5 超级计算机的系统互连。深入了解这些系统的细节。这些系统的互连可以使用户定制的路由器。部分也使用一些商业互连及组件。
- a. 在技术、芯片设计、路由机制和消息传递性能方面，比较这三个系统互连所使用的基本路由器或交换机。
 - b. 比较这三个系统互连的拓扑性质、网络延迟、对分带宽及硬件封装。
- 2.13 学习由 SGI 构建的最新、最大的商业 HPC 集群系统，并依据下列技术和基准测试记录集群体系结构：
- a. SGI 系统模型及其说明是什么？用块范式说明集群体系结构并描述每个模块的功能。
 - b. 讨论由 SGI 公布的峰值性能及持续性能。
 - c. 采用了什么独特硬件、软件、网络或设计特点而实现 (b) 中性能？描述或说明这些系统特征。
- 2.14 考虑图 2-32 中的服务器 - 客户端集群中两台相同服务器之间的主动接管配置。其中服务器通过 SCSI 总线共享磁盘。客户端 (PC 或工作站) 和以太网不会失效。当一台服务器失效时，它的负载将被转移至运行的服务器。
- a. 假定每台服务器的 MTTF 为 200 天，MTTR 为 5 天。磁盘的 MTTF 为 800 天，MTTR 为 20 天。此外，每台服务器由于维护每周需要被关闭一天，在此期间服务器被认为是不可用的。每次只关闭一台服务器。失效率包括正常失效和定期维护。SCSI 总线的失效率为 2%。服务器和磁盘的失效相互独立。磁盘和 SCSI 总线并没有计划关闭。客户端机器永远不会失效。
 1. 只要至少有一台服务器可用，服务器便被认为是可用的。两台服务器的组合可用性是什么？

2. 在正常操作中，集群必须同时有 SCSI 总线、磁盘和至少一台可用的服务器。在该集群中，单点失效的可能性如何？
- b. 该集群不能接受两台服务器在同一时间失效。此外，当 SCSI 总线或磁盘发生故障时，集群被认为是不可用的。基于上述情况，整个集群的系统可用性如何？
- c. 在上述失效和维护的情况下，提出一种改进的体系结构来消除（a）的所有单点失效。

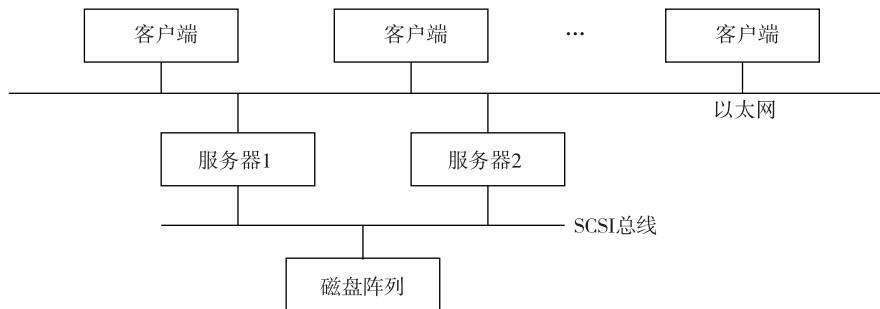


图 2-32 具有冗余硬件的 HA 集群

- 2.15 学习表 2-6 中的集群作业调度策略并回答下面的问题。如果对某个调度策略较为陌生，你可能需要从维基百科、谷歌或者其他来源获得更多信息。
- 解释非抢占和抢占调度策略的优点和缺点，并给出修正这些问题的方法。
 - 基于静态和动态调度策略，重新回答（a）中的问题。
 - 基于专用和共享空间调度策略，重新回答（a）中的问题。
 - 比较分时、独立和组调度策略的相对性能。
 - 相较于远程作业，比较本地作业上驻留和迁移策略的相对性能。
- 2.16 学习 2.3 节中集群的 SSI 特性和 HA，回答下面的问题，并给出原因。指出具有这些特性的部分实例集群系统。评论它们的实现要求，并讨论在集群系统中实现每个 SSI 特征的操作障碍。
- | | |
|----------------|-------------------|
| a. 集群环境的单一入口 | b. 集群系统的单一内存空间 |
| c. 集群系统的单文件层次 | d. 集群系统的单一 I/O 空间 |
| e. 集群系统的单一网络空间 | f. 集群系统的单一联网 |
| g. 集群系统的单点控制 | h. 集群系统的单一作业管理 |
| i. 集群系统的单一用户接口 | j. 集群系统的单一进程空间 |
- 2.17 用例子解释集群作业管理系统的下列方面：
- | | |
|---------------------|-------------------|
| a. 串行作业和并行作业 | b. 批量作业和交互作业 |
| c. 集群作业和外界（本地）作业 | d. 集群进程、本地进程和内核进程 |
| e. 专用模式、共享空间模式和分时模式 | f. 独立调度和组调度 |
- 2.18 这道题关注于 LSF 的概念：
- 给出 4 种 LSF 作业的各自示例。
 - 在 1 000 台服务器的集群中，如果（1）整个集群只有一个管理者 LIM 或者（2）所有的 LIM 均为管理者，给出 LSF 负载共享策略较好的两个原因。
 - 在 LSF 管理者选举机制中，处于“非管理者”状态的某节点成为新的管理者前的等待时间与节点数量成正比。为什么等待时间与节点数量成正比？
- 2.19 本题涉及集群计算中 MOSIX 的使用。查阅公开文献中，了解由设计者和开发者提出的支持 Linux 集群、GPU 集群、多集群甚至虚拟云的目前特征。从用户的角度讨论其优点与不足。
- 2.20 在体系结构设计、资源管理、软件环境和发布应用方面，比较中国 Tianhe-1A 和 Cray Jaguar 的相对优势和缺点。你可能需要进行一些研究，发现关于这些系统的最新发展。解释你给出评价的原因以及依据的信息。

第6章 |

Distributed and Cloud Computing, 1E

云编程和软件环境

本章将论述真实云平台下的编程，其中将介绍和评价 MapReduce、BigTable、Twister、Dryad、DryadLINQ、Hadoop、Sawzall 和 Pig Latin。我们用具体的实例来讲解云中的实现和应用需求。我们回顾了核心服务模型和访问技术。通过应用实例讲解了由谷歌应用引擎（GAE）、亚马逊 Web 服务（AWS）和微软 Windows Azure 提供的云服务。特别地，我们演示了怎样对 GAE、AWS EC2、S3 和 EBS 编程。我们综述了用于云计算的开源 Eucalyptus、Nimbus 和 OpenNebula，以及 Manjrasoft 公司的 Aneka 系统。

6.1 云和网格平台的特性

本节总结了真实云和网格平台的重要特性。在 4 个表格中，我们涵盖了功能、传统特性、数据特性以及程序员和运行时系统使用的特性。表格中的条目可以给那些想要在云上进行高效编程的人提供参考资料。为了更好地学习本章，读者需要熟悉和理解第 5 章中介绍的 SOA 和 Web 服务的相关语言和软件工具。

6.1.1 云的功能和平台的特性

商用云需要全面的功能，见表格 6-1 的总结。这些功能提供高性价比的效用计算，并可以满足计算能力上的弹性伸缩。除了这个关键功能外，商用云还一直在提供越来越多的附加功能，通常被称为“平台即服务”（Platform as a Service, PaaS）。对于 Azure 来说，现有的平台特性包括：Azure Table、队列、blob、SQL 数据库，以及 Web 和工作机角色。亚马逊常常被看做“仅”提供基础设施即服务（IaaS），但是它在不断地增加平台特性，包括 SimpleDB（类似于 Azure Table）、队列、通知、监视、内容发布网络、关系数据库和 MapReduce（Hadoop）。谷歌现在不提供更广泛的云服务，但是谷歌应用引擎（GAE）提供了一个功能强大的 Web 应用开发环境。

表 6-2 列出了一些底层的基础设施特征。表 6-3 列出了在云环境中需要支持的用于并行和分布式系统的传统编程环境。它们可以用作系统（云平台）或用户环境的一部分。表 6-4 给出了在云和一些网格中强调的特性。注意，表 6-4 中的一些特性是最近才作为主要方法提供的。特别地，这些特性并不在学术云基础设施中提供，比如 Eucalyptus、Nimbus、OpenNebula 或 Sector/Sphere（虽然 Sector 是表 6-4 中归类的一个数据并行文件系统（Data Parallel File System, DPFS））。6.5 节将介绍这些新兴的云编程环境。

6.1.2 网格和云的公共传统特性

本节我们集中关注当今计算网格和云中有关工作流、数据传输、安全和可用性方面的公共特性。

表 6-1 重要云平台功能

功 能	描 述
物理/虚拟计算平台	云环境由一些物理或者虚拟平台构成。虚拟平台有一些特别的功能来为不同的应用和用户提供独立环境
大规模数据存储服务，分布式文件系统	对于大规模数据集，云数据存储服务提供大容量磁盘及允许用户上传和下载数据的服务接口。分布式文件系统可以提供大规模数据存储服务，并可提供类似于本地文件系统的接口
大规模数据库存储服务	一些分布式文件系统足以提供应用开发者以更加语化的方式来保存数据的底层存储服务。正如在传统软件栈中的 DBMS，云需要大规模数据库存储服务
大规模数据处理方法和编程模型	云基础设施甚至为一个很简单的应用提供数以千计的计算节点。程序员需要利用这些机器的能力，而不需要考虑繁杂的基础设施管理问题，比如处理网络故障或伸缩运行中的代码来使用平台提供的所有计算设施
工作流和数据查询语言支持	编程模型提供了云基础设施的抽象。类似于数据库系统中使用的 SQL 语言，在云计算中，提供商开发了一些工作流语言和数据查询语言来支持更好的应用逻辑
编程接口和服务部署	云应用需要 Web 接口或者特殊的 API：J2EE、PHP、ASP 或者 Rails。在用户使用 Web 浏览器获取所提供的功能时，云应用可以使用 Ajax 技术来提高用户体验。每个云提供商都开放了其编程接口来访问存储在大规模存储设备上的数据
运行时支持	对于用户及其应用而言，运行时支持是透明的。这些支持包括分布式监视服务、分布式任务调度，以及分布式锁定和其他服务。这对于运行云应用是至关重要的
支持服务	重要的支持服务包括数据和计算服务，例如，云提供了丰富的数据服务和有用的数据并行执行模型，如 MapReduce

6.1.2.1 工作流

如 5.5 节所介绍的，工作流已经在美国和欧洲产生了很多项目。Pegasus、Taverna 和 Kepler 很受欢迎，并得到了广泛的认可。也有一些商用系统，如 Pipeline Pilot、AVS (dated) 和 LIMS 环境。最近的表项是来自微软研究院的 Trident^[2]，它建立在 Windows 工作流基础之上。如果 Trident 运行在 Azure 或只是其他旧版本 Windows 机器上，它将在外部 (Linux) 环境上运行工作流代理服务器。在真实的应用中工作流按需连接多个云和非云服务。

337

表 6-2 基础设施云特征

审计：包括经济学，显然是商业云的一个活跃领域
应用：预先配置的虚拟机镜像，支持多方面任务，如消息传递接口 (Message-Passing Interface, MPI) 集群
认证和授权：云对于多个系统只需要单个登录
数据传输：在网格和云之间或内部的作业组件之间进行数据传输；开发定制的存储模式，比如在 BitTorrent 上
操作系统：Apple、Android、Linux、Windows
程序库：存储镜像和其他程序资料
注册表：系统的信息资源（元数据管理的系统版本）
安全：除了基本认证和授权外的其他安全特性；包括更高层的概念，如可信
调度：Condor、Platform、Oracle Grid Engine 等的基本成分；云隐式地含有它，例如 Azure Worker Role
群组调度：以可扩展的方式分配多个（数据并行）任务；注意，这是 MapReduce 自动提供的
软件即服务 (SaaS)：这个概念是在云和网格之间共享的，并且不用特殊处理就可以被支持；注意，服务和面向服务的体系结构的使用是非常成功的，在云中的应用和之前的分布式系统非常类似
虚拟化：云的基本的支持“弹性”的特性，这被伯克利强调为定义（公有）云的特征；包括虚拟网络，如佛罗里达大学的 ViNe

表 6-3 集群、网格和并行计算环境中的传统特性

集群管理: 提供一系列工具来简化集群处理的 ROCKS 和程序包
数据管理: 包括元数据支持, 例如 RDF Triple 存储 (成功的语义 Web, 并能建于 MapReduce 上, 比如 SHARD); 包括了 SQL 和 NOSQL
网格编程环境: 从开放网格服务体系结构 (Open Grid Services Architecture, OGSA) 中的链接服务到 GridRPC (Ninf, GridSolve) 和 SAGA, 各自不同
Open MP/线程: 可包括并行编译器, 比如 Cilk; 大体上共享内存技术。甚至事务内存和细粒度数据流也在这里
门户: 可被称为 (科学) 网关。技术上还有一个有趣的变化, 从 portlets 到 HUBzero, 到现在的云上: Azure Web Roles 和 GAE
可扩展并行计算环境: MPI 和相关高层概念, 包括不走运的 HP Fortran、PGAS (不成功但也没丢脸)、HPCS 语言 (X-10、Fortress、Chapel)、patterns (包括伯克利 dwarves) 和函数式语言, 例如分布式存储的 F#
虚拟组织: 从专门的网格解决方案到流行的 Web 2.0 功能, 比如 Facebook
工作流: 支持工作流来链接网格和云之间或者内部的作业组件, 与 LIMS 实验室信息管理系统有关

338

表 6-4 云和 (有时) 网格支持的平台特性

Blob: 基本存储概念, 典型代表有 Azure Blob 和亚马逊的 S3
数据并行文件系统 (DPFS): 支持文件系统, 例如谷歌 (MapReduce)、HDFS (Hadoop) 和 Cosmos (Dryad), 带有为数据处理而优化过的计算 – 数据密切度
容错性: 如文献 [1] 中所评述的, 这个特性在网格中被大量地忽略了, 但在云中是一个主要的特性
MapReduce: 支持 MapReduce 编程模型, 包括 Linux 上的 Hadoop、Windows HPCS 上的 Dryad, 以及 Windows 和 Linux 上的 Twister, 包括新的相关语言, 例如 Sawzall、Pregel、Pig Latin 和 LINQ
监测: 很多网格解决方案, 例如 Inca, 可以基于发布 – 订阅
通知: 发布 – 订阅系统的基本功能
编程模型: 和其他平台特性一起建立的云编程模型, 和熟悉的 Web 和网格模型相关
队列: 可能基于发布 – 订阅的排队系统
可扩展的同步: Apache Zookeeper 或 Google Chubby。支持分布式锁, 由 BigTable 使用。不清楚是否 (有效地) 使用于 Azure Table 或 Amazon SimpleDB 中
SQL: 关系数据库
Table: 支持 Apache Hbase 或 Amazon SimpleDB/Azure Table 上的表数据结构模型。是 NOSQL 的一部分
Web 角色: 用在 Azure 中用来向用户描述重要链路, 并能被除了门户框架以外的架构所支持。这是 GAE 的主要目的
工作机角色: 这个概念已被亚马逊和网格隐式使用, 但第一次被 Azure 作为一个高层架构提出

6.1.2.2 数据传输

在商业云中 (较少程度上, 在商业云之外) 数据传输的成本 (时间和金钱) 经常被认为是使用云的一个难点。如果商业云成为一个国家计算机基础设施的重要部分, 我们可以预期在云和 TeraGrid 之间将出现一条高带宽链路。带有分块 (在 Azure blob 中) 和表格的云数据的特殊结构允许高性能并行算法, 但最初, 使用简单 HTTP 机制在学术系统/TeraGrid 和商业云上传输数据^[3-5]。

6.1.2.3 安全、隐私和可用性

以下技术与开发一个健康、可靠的云编程环境的安全、隐私和可用性需求有关。我们把这些技术总结如下。这些技术中的一些以及可能的解决方案已经在 4.4.6 节讨论过。

- 使用虚拟集群化来实现用最小的开销成本达到动态资源供应。
- 使用稳定和持续的数据存储, 带有用于信息检索的快速查询。
- 使用特殊的 API 来验证用户及使用商业账户发送电子邮件。
- 使用像 HTTPS 或者 SSL 等安全协议来访问云资源。

- 需要细粒度访问控制来保护数据完整性，阻止侵入者或黑客。
- 保护共享的数据集，以防恶意篡改、删除或者版权侵犯。
- 包括增强的可用性和带有虚拟机实时迁移的灾难恢复等特性。
- 使用信用系统来保护数据中心。这个系统只授权给可信用户，并阻止侵入者。

[339]

6.1.3 数据特性和数据库

在下面的内容中，我们评述了有用的编程特性，这些特性和程序库、blob、驱动、DPFS、表格和各种数据库（包括 SQL、NOSQL 和非关系数据库、特殊队列服务等）相关。

6.1.3.1 程序库

人们进行了许多努力来设计虚拟机镜像库，以便管理学术云和商业云中用到的的镜像。本章中描述的基本云环境也包含很多管理特性，允许方便地部署和配置镜像（即它们支持 IaaS）。

6.1.3.2 blob 和驱动

云中基本的存储概念是 Azure 的 blob 和亚马逊的 S3。这些都能由 Azure 的集装箱来组织（近似地，和在目录中一样）。除了 blob 和 S3 的服务接口，人们还可以“直接”附加到计算实例中作为 Azure 驱动和亚马逊的弹性块存储。这个概念类似于共享文件系统，比如 TeraGrid 中使用的 Lustre。云存储内部是有容错能力的，而 TeraGrid 需要备份存储。然而，云和 TeraGrid 之间的体系结构理念非常类似，所以简单云文件存储 API^[6] 也会变得更为重要。

6.1.3.3 DPFS

这包括对诸如谷歌文件系统（MapReduce）、HDFS（Hadoop）和 Cosmos（Dryad）等文件系统的支持，并且带有为数据处理而优化过的计算 – 数据密切度。这使得链接 DPFS 到基本 blob 和基于驱动的体系结构成为可能，但是将 DPFS 用作以应用为中心并带有计算 – 数据密切度的存储模型会更简单，同时用 blob 及驱动作为以存储为中心的视图。

一般来说，需要数据传输来连接这两种数据视图。认真地考虑这一点似乎很重要，因为 DPFS 文件系统是为执行数据密集型应用而精确设计的。然而，链接亚马逊和 Azure 的 DPFS 的重要性还不是很清楚，因为这些云现在并没有为计算 – 数据密切度提供细粒度支持。在这里，我们注意到 Azure Affinity Groups 是一个有趣的功能^[7]。我们期望最初 blob、驱动、表格和队列都在这个范围内，并且学术系统能有效地提供类似于 Azure（以及亚马逊）那样的平台。注意 HDFS（Apache）和 Sector（UIC）项目也在这个范围内。

6.1.3.4 SQL 和关系型数据库

亚马逊和 Azure 云都提供关系型数据库，这可以直接为学术系统提供一个类似的功能，但如果是需要大规模数据，事实上，基于表或 MapReduce 的方法可能会更合适^[8]。作为一个早期的用户，我们正在为观测性医疗结果伙伴关系组织（Observational Medical Outcomes Partnership, OMOP）开发一个基于 FutureGrid 的新的私有云计算模型，这是关于病人医疗数据的项目，使用了 Oracle 和 SAS，其中 FutureGrid 增加了 Hadoop 来扩展到许多不同的分析方法。

注意到，我们可以使用数据库来说明两种配置功能的方法。传统情况下，我们可以把数据库软件加入到计算机磁盘。给出你的数据库实例后，这个软件就能被执行。然而在 Azure 和亚马逊中，数据库是安装在一个独立于你的作业（Azure 中的工作机角色）的单独虚拟机上。这实现了“SQL 即服务”。在消息传递接口上可能存在一些性能问题，但是很明显“即服务”部署简化了 [340] 系统。对于 N 个平台特性来说，我们只需要 N 个服务，其中不同方式的可能镜像数量会是 2^N 。

6.1.3.5 表格和 NOSQL 非关系型数据库

在简化数据库结构（称为 NOSQL^[9,10]）上已经有了很多重要的进展，典型情况强调了分布式和可扩展性。这些进展体现在三种主要云里：谷歌的 BigTable^[11]、亚马逊的 SimpleDB^[12] 和

Azure 的 Azure Table^[13]。由天文学^[14]中的 VOTable 标准和 Excel 的普及都清楚说明了表格在科学中的重要性。然而在云之外使用表格并没有丰富经验。

当然也有非关系型数据库的许多重要应用，尤其是使用三元存储来实现元数据的存取。最近，研究人员的兴趣在于基于 MapReduce 和表格或者 Hadoop 文件系统^[8,15]来建立可扩展的 RDF 三元存储，在大规模存储上有一些早期成功的报道。当前的云表格可以分为两组：Azure 表格和亚马逊 SimpleDB 非常类似^[16]，并支持轻量级的“文档商店”存储；而 BigTable 旨在管理大规模分布式数据集，且没有大小的限制。

所有这些表格都是自由模式的（每个记录可以有不同的属性），尽管 BigTable 有列（属性）家族模式。表格对科学计算的作用似乎会越来越大，学术系统会用两个 Apache 项目来支持这一点：BigTable 的 Hbase^[17] 和文档商店的 CouchDB^[18]。另一个可能是开源的 SimpleDB 实现 M/DB^[19]。支持文件存储、文档存储服务和简单队列的新的 Simple Cloud API^[6]，可以帮助在学术云和商业云之间提供一个公共的环境。

6.1.3.6 队列服务

亚马逊和 Azure 都能提供类似的可扩展、健壮的队列服务，用来在一个应用的组件之间通信。消息长度应较短（小于 8KB），有一个“至少发送一次”语义的 REST 服务接口。这些由超时器来控制，能发送一个客户端所允许的处理时间长度。人们可以在（在一个更小、更少挑战的学术环境）建立类似的方法，基于 ActiveMQ^[20] 或者 NaradaBrokering^[21,22] 等发布 - 订阅系统，这些我们都有大量经验。

6.1.4 编程和运行时支持

我们需要编程和运行时支持来促进并行编程，并为今天的网格和云上的重要功能提供运行时支持。本节介绍了各种 MapReduce 系统。

6.1.4.1 工作机和 Web 角色

Azure 引入的角色提供了重要功能，并有可能在非虚拟化环境中保留更好的密切度支持。工作机角色是基本的可调度过程，并能自动启动。注意在云上没有必要进行明显的调度，无论是对个人工作机角色还是 MapReduce 透明支持的“群组调度”。在这里，队列是一个关键概念，因为它们提供一个自然的方法来以容错、分布式方式管理任务分配。Web 角色为门户提供了一个有趣的途径。GAE 主要用于 Web 应用，而科学门户在 TeraGrid 上非常成功。
[341]

6.1.4.2 MapReduce

“数据并行”语言日益受到广泛关注，这种语言主要的目的在于在不同数据样本上执行松耦合的计算。语言和运行时产生和提供了“多任务”问题的有效执行，著名的成功案例就是网格应用。然而，与传统方法相比，表 6-5 中总结的 MapReduce 对于多任务问题的实现有一些优点，因为它支持动态执行、强容错性以及一个容易使用的高层接口。主要的开源/商用 MapReduce 实现是 Hadoop^[23] 和 Dryad^[24~27]，其执行可能用到或者不用虚拟机。

Hadoop 现在是由亚马逊提供的，我们期望 Dryad 能在 Azure 上实现。印第安纳大学已经建立了一个原型 Azure MapReduce，我们将在下面讨论。在 FutureGrid 上，我们已经准备好支持 Hadoop、Dryad 和其他 MapReduce 方案，包括 Twister^[29]，它支持在很多数据挖掘和线性代数应用中的迭代计算。注意，这个方法和 Cloudera^[35] 有点类似，Cloudera 能提供很多 Hadoop 分发版本，包括亚马逊和 Linux。MapReduce 相对于其他云平台特性而言更接近于宽部署，因为它有相当多关于 Hadoop 和 Dryad 在云外的经验。

表 6-5 MapReduce 类型系统的比较

	谷歌 MapReduce ^[28]	Apache Hadoop ^[23]	微软 Dryad ^[26]	Twister ^[29]	Azure Twister ^[30]
编程模型	MapReduce	MapReduce	DAG 执行；扩展到 MapReduce 和其他模式	迭代 MapReduce	现在只有 MapReduce；将扩展为可迭代的 MapReduce
数据处理	GFS（谷歌文件系统）	HDFS（Hadoop 分布式文件系统）	共享目录和本地磁盘	本地磁盘和数据管理工具	Azure blob 存储
调度	数据位置	数据位置；Rack 感知；用全局队列进行动态任务调度	数据位置；运行时优化的网络拓扑；静态任务分区	数据位置；静态任务分区	用全局队列进行动态任务调度
故障处理	重新执行失败任务；慢速任务的复制执行	重新执行失败任务；慢速任务的复制执行	重新执行失败任务；慢速任务的复制执行	迭代的重新执行	重新执行失败任务；慢速任务的复制执行
HLL 支持	Sawzall ^[31]	Pig Latin ^[32,33]	DryadLINQ ^[27]	Prege ^[34] 有相关特性	N/A
环境	Linux 集群	Linux 集群，亚马逊 EC2 上的弹性 MapReduce	Windows HPCS 集群	Linux 集群；EC2	Windows Azure，Azure 本地开发虚拟环境
中间数据传输	文件	文件、HTTP	文件，TCP 管道，共享内存 FIFO	发布 - 订阅消息传递	文件，TCP

6.1.4.3 云编程模型

前面的大多数内容都是描述编程模型特性，但是还有很多“宏观的”架构，并不能作为代码（语言和库）。GAE 和 Manjrasoft Aneka 环境都代表编程模型；都适用于云，但实际上并不是针对这个体系结构的。迭代 MapReduce 是一个有用的编程模型，它提供了在云、HPC 和集群环境之间的可移植性。

6.1.4.4 软件即服务

服务在商业云和大部分现代分布式系统中以类似的方式使用。我们希望用户能尽可能地封装他们的程序，这样不需要特殊的支持来实现软件即服务。我们已经在 6.1.3 节中讨论过为什么在一个数据库服务环境下“系统软件即服务”是一个有趣的想法。我们需要 SaaS 环境提供很多有用的工具，能在大规模数据集上开发云应用。除了技术特征之外，例如 MapReduce、BigTable、EC2、S3、Hadoop、AWS、GAE 和 WebSphere2，我们还需要可以帮助我们实现可扩展性、安全性、隐私和可用性的保护特征。

6.2 并行和分布式编程范式

我们把并行和分布式程序定义为运行在多个计算引擎或一个分布式计算系统上的并行程序。这个术语包含计算机科学中的两个基本概念：分布式计算系统和并行计算。分布式计算系统是一系列由网络连接的计算引擎，它们完成一个共同目标：运行一个作业或者一个应用。计算机集群或工作站网络就是分布式计算系统的一个实例。并行计算是同时运用多个计算引擎（并不一定需要网络连接）来运行一个作业或者一个应用。例如，并行计算可以使用分布式或者非分布式计算系统，如多处理器平台。

在分布式计算系统上（并行和分布式编程）运行并行程序，对于用户和分布式计算系统都有一些优点。从用户的角度，它减少了应用响应时间；从分布式计算系统的角度，它提高了吞吐量和资源利用率。然而，在分布式计算系统上运行并行程序，是一个很复杂的过程。所以，从复杂性的角度，本章将进一步介绍在分布式系统上运行一个典型并行程序的数据流。

342
343

6.2.1 并行计算和编程范式

考虑一个由多个网络节点或者工作组组成的分布式计算系统。这个系统是用并行或分布式方式来运行一个典型的并行程序，该系统包括以下方面^[36~39]：

- **分区：**如下所示，分区适用于计算和数据两方面：
- **计算分区：**计算分区是把一个给定的任务或者程序分割成多个小任务。分区过程很大程度上依靠正确识别可以并发执行的作业或程序的每一小部分。换句话说，一旦在程序结构中识别出并行性，它就可以分为多个部分，能在不同的工作机上运行。不同的部分可以处理不同的数据或者同一数据的副本。
- **数据分区：**数据分区是把输入或中间数据分割成更小的部分。类似地，一旦识别出输入数据的并行性，它也可以被分割成多个部分，能在不同的工作机上运行。数据块可由程序的不同部分或者同一程序的副本处理。
- **映射：**映射是把更小的程序部分或者更小的数据分块分配给底层的资源。这个过程的目的在于合理分配这些部分或者分块，使它们能够同时在不同的工作机上运行。映射通常由系统中的资源分配器来处理。
- **同步：**因为不同工作机可以执行不同的任务，工作机之间的同步和协调就很有必要。这样可以避免竞争条件，不同工作机之间的数据依赖也能被恰当地管理。不同工作机多路访问共享资源可能引起竞争条件。然而，当一个工作机需要其他工作机处理的数据时会产生数据依赖。
- **通信：**因为数据依赖是工作机之间通信的一个主要原因，当中间数据准备好在工作机之间传送时，通信通常就开始了。
- **调度：**对于一项作业或一个程序，当计算部分（任务）或数据块的数量多于可用的工作机数量时，调度程序就会选择一个任务或数据块的序列来分配给工作机。值得注意的是，资源分配器完成计算或数据块到工作机的实际映射，而调度器只是基于一套称为调度策略的规则，来从没有分配的任务队列中选择下一个任务。对于多作业或多程序，调度器会选择运行在分布式计算系统上的一个任务或程序的序列。这样看来，当系统资源不够同时运行多个作业或程序时，调度器也是很有必要的。

编程范式的动机

因为处理并行和分布式编程的整个数据流是非常耗时的，并且需要特别的编程知识，所以处理这些问题会影响到程序员的效率，甚至会影响到程序进入市场的时间。而且，它会干扰程序员集中精力在程序本身的逻辑上。因此，提供并行和分布式编程范式或模型来抽象用户数据流的多个部分。

344

换句话说，这些模型的目的是为用户提供抽象层来隐藏数据流的实现细节，否则以前用户是需要为之写代码的。所以，编写并行程序的简单性是度量并行和分布式编程范式的重要标准。并行和分布式编程模型背后的其他动机还有：(1) 提高程序员的生产效率，(2) 减少程序进入市场的时间，(3) 更有效地利用底层资源，(4) 提高系统的吞吐量，(5) 支持更高层的抽象^[40]。

MapReduce、Hadoop 和 Dryad 是最近提出的三种并行和分布式编程模型。这些模型是为信息检索应用而开发的，不过已经显示出它们也适用于各种重要应用^[41]。而且这些范式组件之间的松散耦合使得它们适用于虚拟机实现，并使其对于某些应用的容错能力和可扩展性都优于传统的并行计算模型，例如 MPI^[42~44]。

6.2.2 MapReduce、Twister 和迭代 MapReduce

6.1.4节介绍的MapReduce是一个软件框架，可以支持大规模数据集上的并行和分布式计算^[27, 37, 45, 46]。这个软件框架抽象化了在分布式计算系统上运行一个并行程序的数据流，并以两个函数的形式提供给用户两个接口：*Map*（映射）和*Reduce*（化简）。用户可以重载这两个函数以实现交互和操纵运行其程序的数据流。图6-1说明了在MapReduce框架中从*Map*到*Reduce*函数的逻辑数据流。在这个框架中，数据的‘value’部分（key, value）是实际数据，‘key’部分只是被MapReduce控制器使用来控制数据流^[37]。

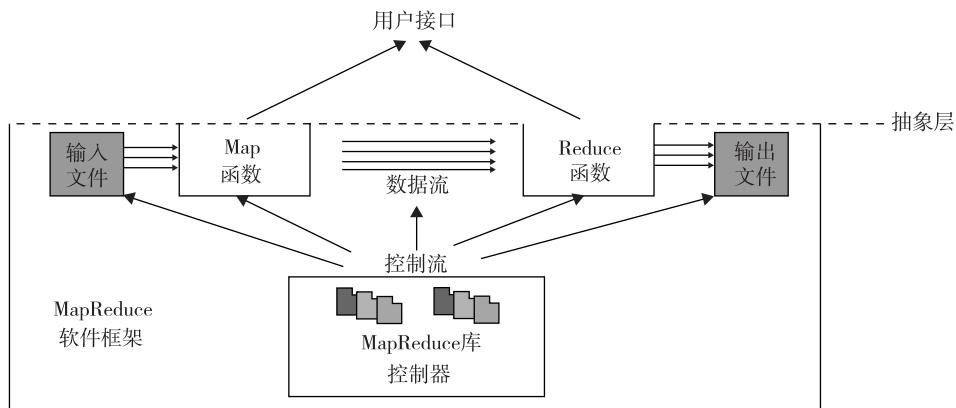


图6-1 MapReduce框架：输入数据流经*Map*和*Reduce*函数，在使用MapReduce软件库的控制流下产生输出结果。使用特别的用户接口来访问*Map*和*Reduce*资源

6.2.2.1 MapReduce的形式化定义

MapReduce软件框架向用户提供了一个具有数据流和控制流的抽象层，并隐藏了所有数据流实现的步骤，比如，数据分块、映射、同步、通信和调度。这里，虽然在这样的框架中数据流已被预定义，但抽象层还提供两个定义完善的接口，这两个接口的形式就是*Map*和*Reduce*这两个函数^[47]。这两个主函数能由用户重载以达到特定目标。图6-1给出了具有数据流和控制流的MapReduce框架。

所以，用户首先重载*Map*和*Reduce*函数，然后从库里调用提供的函数*MapReduce (Spec, & Results)*来开始数据流。*MapReduce*函数*MapReduce (Spec, & Results)*有一个重要的参数，这个参数是一个规范对象‘*Spec*’。它首先在用户的程序里初始化，然后用户编写代码来填入输入和输出文件名，以及其他可选调节参数。这个对象还填入了*Map*和*Reduce*函数的名字，以识别这些用户定义的函数和MapReduce库里提供的函数。

下面给出了用户程序的整个结构，包括*Map*、*Reduce*和*Main*函数。*Map*和*Reduce*是两个主要的子程序。它们被调用来实现在主程序中执行的所需函数。

```

Map Function (....)
{
    .....
}
Reduce Function (....)
{
    .....
}
Main Function (....)
{
    Initialize Spec object
    .....
    MapReduce (Spec, & Results)
}

```

6.2.2.2 MapReduce 逻辑数据流

Map 和 *Reduce* 函数的输入数据有特殊的结构。输出数据也一样。*Map* 函数的输入数据是以 (key, value) 对的形式出现。例如, key 是输入文件的行偏移量, value 是行内容。*Map* 函数的输出数据的结构类似于 (key, value) 对, 称为中间 (key, value) 对。换句话说, 用户自定义的 *Map* 函数处理每个输入的 (key, value) 对, 并产生很多 (zero, one, or more) 中间 (key, value) 对。这里的目的是为 *Map* 函数并行处理所有输入的 (key, value) 对 (见图 6-2)。

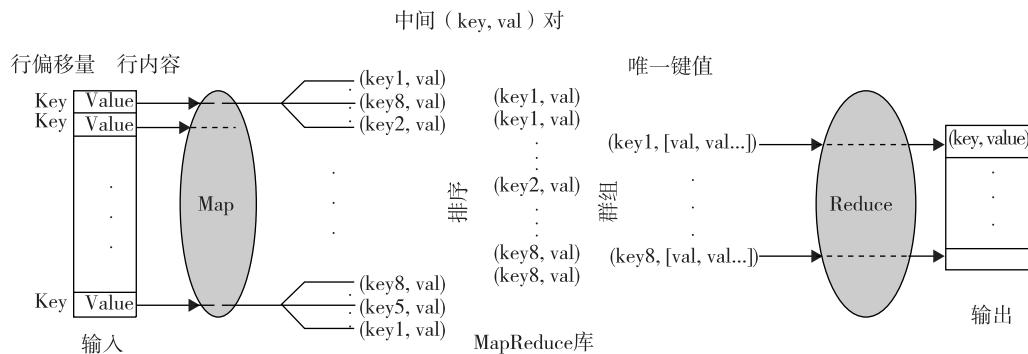


图 6-2 在 5 个处理步骤中连续 (key, value) 对的 MapReduce 逻辑数据流

反过来，*Reduce* 函数以中间值群组的形式接受中间 $(key, value)$ 对，这个中间值群组和一个中间 $key(key, [set\ of\ values])$ 相关。实际上，MapReduce 框架形成了这些群组，首先是对中间 $(key, value)$ 对排序，然后以相同的 key 来把 $value$ 分组。需要注意的是，数据的排序是为了简化分组过程。*Reduce* 函数处理每个 $(key, [set\ of\ values])$ 群组，并产生 $(key, value)$ 对集合作为输出。

为了阐明样本 MapReduce 应用中的数据流，我们这里将介绍 MapReduce 的一个例子应用，也是一个著名的 MapReduce 问题——被称为“单词计数”(word-count)，是用来计算一批文档中每一个单词出现的次数。图 6-3 说明了一个简单输入文档的“单词计数”问题的数据流，这个文件只包含如下两行：(1) “most people ignore most poetry”，(2) “most poetry ignores most people”。在这个例子里，*Map* 函数同时为每一行内容产生若干个中间 (key, value) 对，所以每个单词都用带“1”的中间键值作为其中间值，如 (*ignore*, 1)。然后，MapReduce 库收集所有产生的中间 (key, value) 对，进行排序，然后把每个相同的单词分组为多个“1”，如 (*people*, [1, 1])。然后把组并行送入 *Reduce* 函数，所以就把每个单词的“1”累加起来，并产生文件中每个单词出现的实际数目，例如 (*people*, 2)。

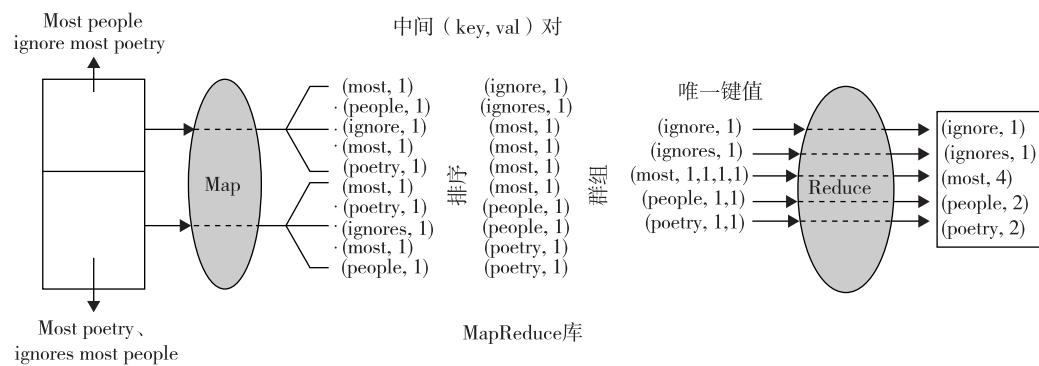


图 6-3 单词计数问题的数据流，以级联操作方式使用 MapReduce 函数 (Map, Sort, Group 和 Reduce)

6.2.2.3 MapReduce 数据流的形式化符号

对每个输入 (key, value) 对并行地应用 *Map* 函数，并产生新的中间 (key, value) 对^[37]，如下所示：

$$(key_1, val_1) \xrightarrow{\text{Map 函数}} List(key_2, val_2) \quad (6-1)$$

然后，MapReduce 库收集所有输入 (key, value) 对产生的中间 (key, value) 对，并基于键值部分进行排序。然后分组统计所有相同键值出现的次数。最后，对于每个分组并行地应用 *Reduce* 函数，来产生值集合作为输出，如下所示：

$$(key_2, List(val_2)) \xrightarrow{\text{Reduce 函数}} List(val_2) \quad (6-2)$$

6.2.2.4 解决 MapReduce 问题的策略

正如上文提及的，将所有中间数据分组之后，出现相同 key 的 value 会排序并组合在一起。产生的结果是，分组之后所有中间数据中每一个 key 都是唯一的。所以寻找唯一的 key 是解决一个典型 MapReduce 问题的出发点。然后，作为 *Map* 函数输出的中间 (key, value) 对将会自动找到。下面的三个例子解释了如何在这些问题中确定 key 和 value：

问题 1：计算一批文档中每个单词的出现次数。

解：唯一 “key” ——每个单词；中间 “value” ——出现次数。

问题 2：计算一批文档中相同大小、相同字母数量的单词的出现次数。

解：唯一 “key” ——每个单词；中间 “value” ——单词大小。

问题 3：计算一批文档中变位词 (anagram) 出现的次数。变位词是指字母相同但是顺序不同的单词（例如，单词 “listen” 和 “silent”）。

解：唯一 “key” ——每个单词中按照字母顺序排列的字母（如 “eilnst”）；中间 “value” ——出现次数。

6.2.2.5 MapReduce 真实数据和控制流

MapReduce 框架主要作用是在一个分布式计算系统上高效运行用户程序。所以，MapReduce 框架精细地处理这些数据流的所有分块、映射、同步、通信和调度的细节^[48,49]。我们总结为如下 11 个清晰的步骤：

1. **数据分区：**MapReduce 库将已存入 GFS 的输入数据（文件）分割成 M 部分， M 也即映射任务的数量。

2. **计算分区：**计算分块通过强迫用户以 *Map* 和 *Reduce* 函数的形式编写程序，（在 MapReduce 框架中）被隐式地处理。所以，MapReduce 库只生成用户程序的多个复制（例如，通过 fork 系统调用），它们包含了 *Map* 和 *Reduce* 函数，然后在多个可用的计算引擎上分配并启动它们。

3. **决定主服务器 (master) 和服务器 (worker)：**MapReduce 体系结构是基于主服务器 – 服务器模式的。所以一个用户程序的复制变成了主服务器，其他则是服务器。主服务器挑选空闲的服务器，并分配 *Map* 和 *Reduce* 任务给它们。典型地，一个映射/化简服务器是一个计算引擎，例如集群节点，通过执行 *Map/Reduce* 函数来运行映射/化简任务。步骤 4 ~ 7 描述了映射服务器。

4. **读取输入数据 (数据分发)：**每一个映射服务器读取其输入数据的相应部分，即输入数据分割，然后输入至其 *Map* 函数。虽然一个映射服务器可能运行多个 *Map* 函数，这意味着它分到了不止一个输入数据分割；通常每个服务器只分到一个输入分割。

5. **Map 函数：**每个 *Map* 函数以 (key, value) 对集合的形式收到输入数据分割，来处理并产生中间 (key, value) 对。

346
348

6. **Combiner 函数**: Combiner 函数是映射服务器中一个可选的本地函数，适用于中间 (key, value) 对。用户可以在用户程序里调用 Combiner 函数。Combiner 函数运行用户为 Reduce 函数所写的相同代码，因为它们的功能是一样的。Combiner 函数合并每个映射服务器的本地数据，然后送到网络传输，以有效减少通信成本。正如我们在逻辑数据流的讨论中提到的，MapReduce 框架对数据进行排序并分组，然后数据被 Reduce 函数处理。类似地，如果用户调用 Combiner 函数，MapReduce 框架也会对每个映射服务器的本地数据排序并分组。

7. **Partitioning 函数**: 正如在 MapReduce 数据流中提到的，具有相同键值的中间 (key, value) 对被分组到一起，因为每个组里的所有值都应只由一个 Reduce 函数来处理产生最终结果。然而在实际实现中，由于有 M 个 map 和 R 个化简任务，有相同 key 的中间 (key, value) 对可由不同的映射任务产生，尽管它们只应由一个 Reduce 函数来一起分组并处理。

所以，由每一个映射服务器产生的中间 (key, value) 对被分成 R 个区域，这和化简任务的数量相同。分块是由 Partitioning (分区) 函数完成，并能保证有相同键值的所有 (key, value) 对都能存储在同一区域内。因此，由于化简服务器 i 读取所有映射服务器区域 i 中的数据，有相同 key 的所有 (key, value) 对将由相应的化简服务器 i 收集（见图 6-4）。为了实现这个技术，Partitioning 函数可以仅是将数据输入特定区域的哈希函数（例如 $\text{Hash}(\text{key}) \bmod R$ ）。注意， R 个分区中的缓冲数据的位置被送至服务器，以便后来将数据送至化简服务器。

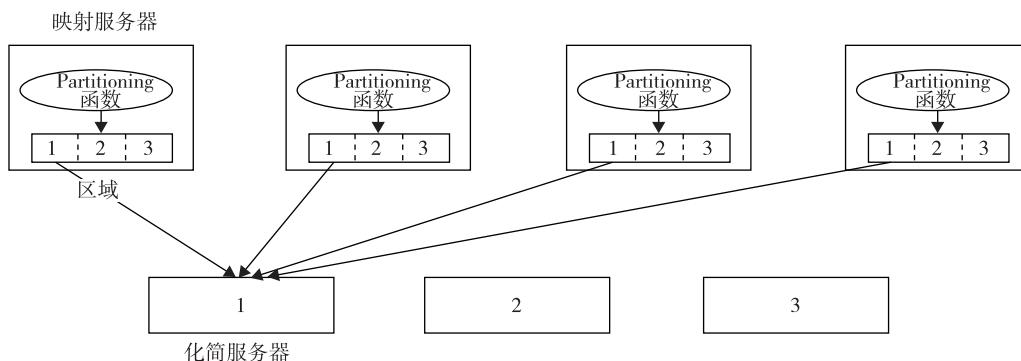


图 6-4 使用 MapReduce Partitioning 函数把映射和化简服务器链接起来

图 6-5 阐述了所有数据流步骤的数据流实现。以下是两个联网的步骤：

8. **同步**: MapReduce 使用简单的同步策略来协调映射服务器和化简服务器，当所有映射任务完成时，它们之间的通信就开始了。
[349]

9. **通信**: Reduce 服务器 i 已经知道所有映射服务器的区域 i 的位置，使用远程过程调用来从所有映射服务器的各个区域中读取数据。由于所有化简服务器从所有映射服务器中读取数据，映射和化简服务器之间的多对多通信在网络中进行，会引发网络拥塞。这个问题是提高此类系统性能的一个主要瓶颈^[50~52]。文献[55]提出了一个数据传输模块来独立地调度数据传输。

步骤 10 和 11 对应于化简服务器方面的：

10. **排序和分组**: 当化简服务器完成读取输入数据的过程时，数据首先在化简服务器的本地磁盘中缓冲。然后化简服务器根据 key 将数据排序来对中间 (key, value) 对进行分组，之后对出现的所有相同 key 进行分组。注意，缓冲数据已经排序并分组，因为一个映射服务器产生的唯一 key 的数量可能会多于 R 个区域，所以在每个映射服务器区域中可能有不止一个 key（见图 6-4）。

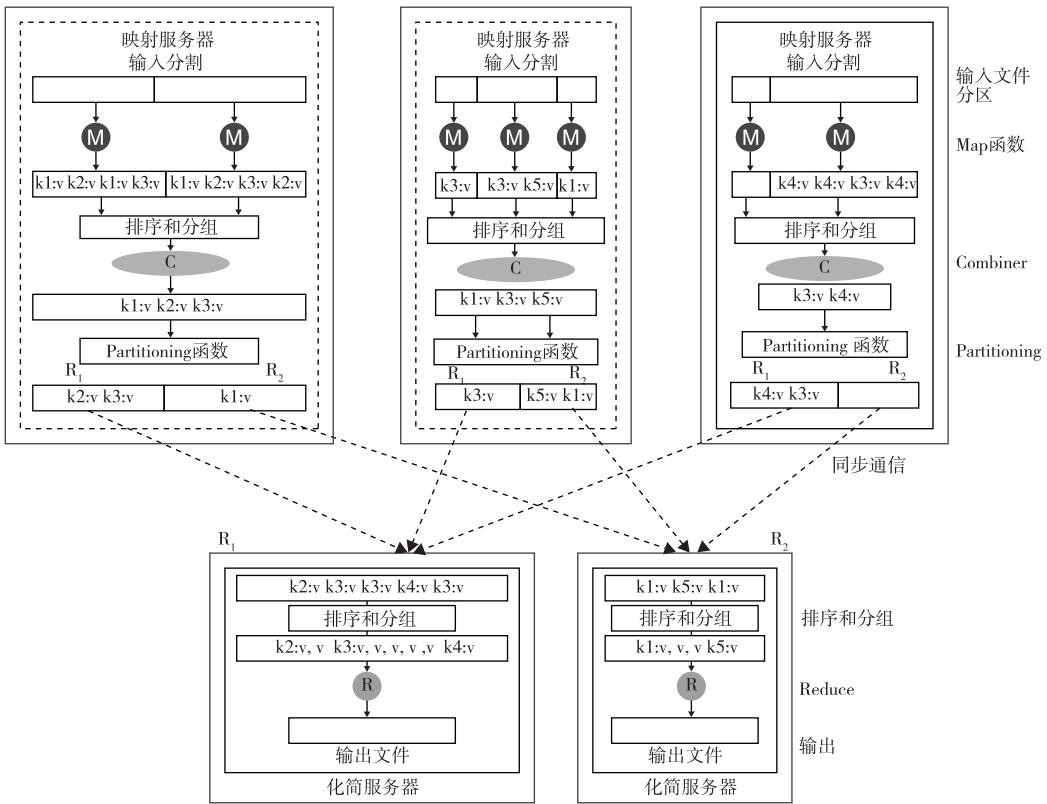


图 6-5 映射服务器和化简服务器的许多函数的数据流实现，通过分区、汇总、同步和通信、排序和分组，以及化简操作的多个序列

11. **Reduce 函数：**化简服务器在已分组的 (key, value) 对上进行迭代。对于每一个唯一的 key，它把 key 和对应的 value 发送给 Reduce 函数。然后，这个函数处理输入数据，并将最后输出结果存入用户程序已经指定的文件中。

为了更好地区分 MapReduce 框架中的相关数据控制和控制流，图 6-6 给出了这种系统中过程控制的精确顺序，与图 6-5 中的数据流相对应。

6.2.2.6 计算 - 数据密切度

MapReduce 软件框架最早是由谷歌提出并实现的。首次实现是用 C 语言编码的。该实现是将谷歌文件系统 (GFS)^[53]的优势作为最底层。MapReduce 可以完全适用于 GFS。GFS 是一个分布式文件系统，其中文件被分成固定大小的块，这些块被分发并存储在集群节点上。

如前所述，MapReduce 库将输入数据（文件）分割成固定大小的块，理想状态下是在每个块上并行地执行 Map 函数。在这种情况下，由于 GFS 已经将文件保存成多个块，MapReduce 框架只需要将包含 Map 函数的用户程序复制发给已经存有数据块的节点。这就是将计算发向数据，而不是将数据发给计算。注意，GFS 块默认为 64MB，这和 MapReduce 框架是相同的。

6.2.2.7 Twister 和迭代 MapReduce

理解不同运行时间的性能是很重要的，尤其是比较 MPI 和 MapReduce^[43,44,55,56]。并行开销的两个主要来源是负载不均衡和通信（这相当于通信同步并行单元〔线程或进程〕时的同步开销，见表 6-10 的第 2 类和第 6 类）。MapReduce 的通信开销相当大，原因有两个：

- MapReduce 是通过文件来读取和写入的，而 MPI 通过网络直接在节点之间传输信息。

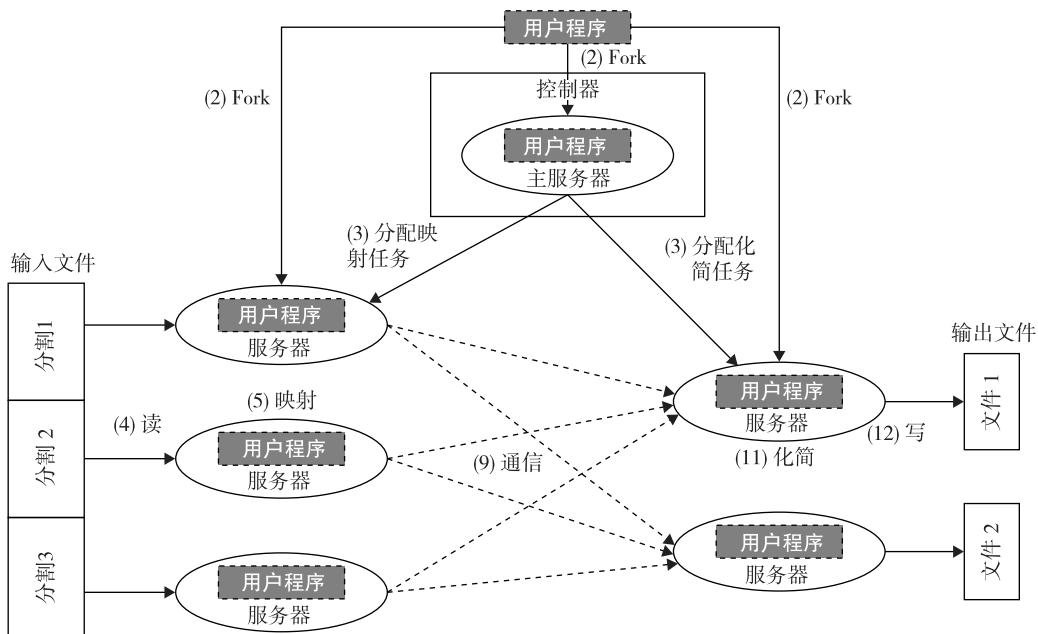


图 6-6 在映射服务器和化简服务器（运行用户程序）中 MapReduce 功能的控制流实现，在主服务器用户程序的控制下从输入文件到输出文件

注：由 Yahoo! Pig Tutorial 提供^[54]。

- MPI 并没有把所有的数据都在节点之间传输，而只是需要更新信息的数量。我们可以把 MPI 流称为 δ 流，而把 MapReduce 流称为全数据流。

在所有“经典并行”的松散同步应用中可以看到同样的现象，典型地需要在计算阶段加入一个迭代结构，然后是通信阶段。我们可以通过两个重要的改变来解决性能问题：

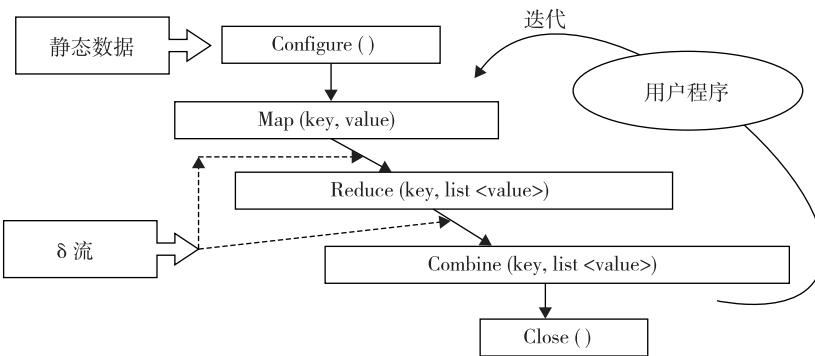
1. 在各个步骤之间的流信息，不把中间结果写入磁盘。
2. 使用长期运行的线程或进程与 δ （在迭代之间）流进行通信。

这些改变将会导致重大的性能提升，代价是较差的容错能力，同时更容易支持动态改变，如可用节点的数量。这个概念^[42]已经在多个项目^[34, 57-59]中应用，此外，文献[44]提出了在 MapReduce 应用中使用 MPI 的直接想法。图 6-7 示例了 Twister 编程范式及其运行时实现体系结构。在例 6.1 中，我们总结了 Twister^[60]，图 6-8 给出了对于 K 均值的性能结果^[55, 56]，其中 Twister 要比传统的 MapReduce 快很多。Twister 从通信的动态 δ 流中区分了从来不会被加载的静态数据。

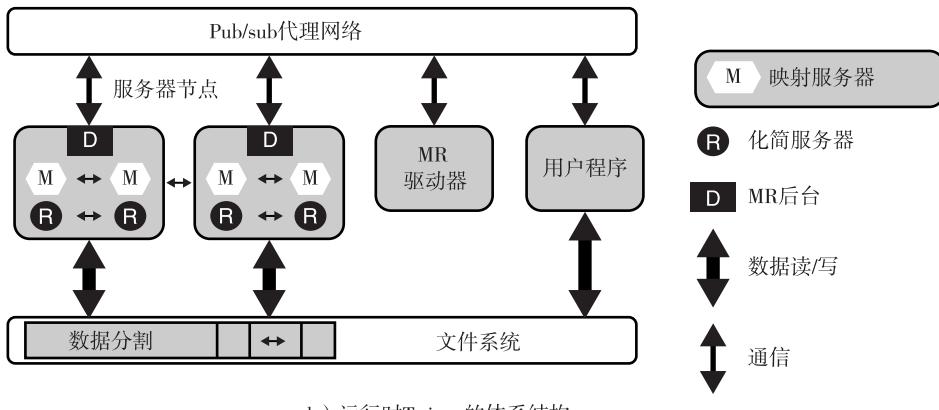
例 6.1 在 MPI、Twister、Hadoop 和 DryadLINQ 上的 K 均值集群化的性能

MapReduce 方法能实现容错和灵活调度，但对于有些应用程序来说，其性能下降相对于 MPI 来说比较严重，如图 6-8 所示的一个简单并行 K 均值集群化算法。对于大规模数据集 Hadoop 和 DryadLINQ 比 MPI 慢了 10 倍多，而对于较小的数据集慢得更多。人们在迭代 MapReduce 上可以使用很多通信机制，但是 Twister 选择了使用分布式代理集合的发布-订阅网络，如 5.2 节中所描述，达到了类似于 ActiveMQ 和 NaradaBrokering 的性能。

Map-Reduce 对在长运行的线程中迭代地执行。在图 6-9 中，我们比较了 4 种并行编程范式的不同线程和进程结构：Hadoop、Dryad、Twister（也称为 MapReduce ++）和 MPI。注意，根据文献[26, 27]，Dryad 可以使用管道，避免了昂贵的磁盘写。



a) 迭代MapReduce编程的Twister



b) 运行时Twister的体系结构

图 6-7 Twister：一个迭代的 MapReduce 编程范式，用于重复的 MapReduce 运行

353

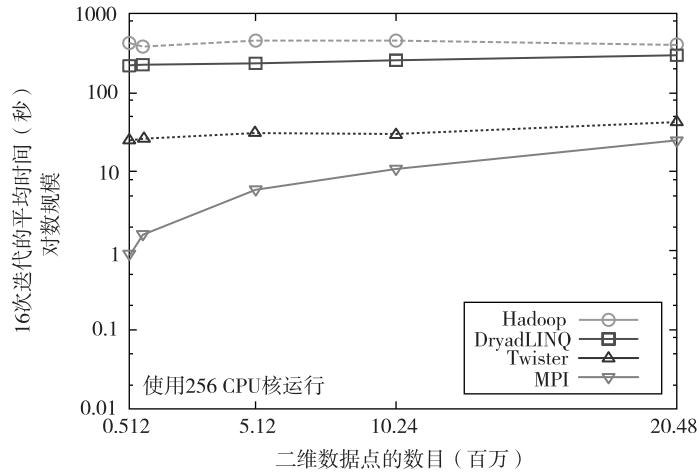
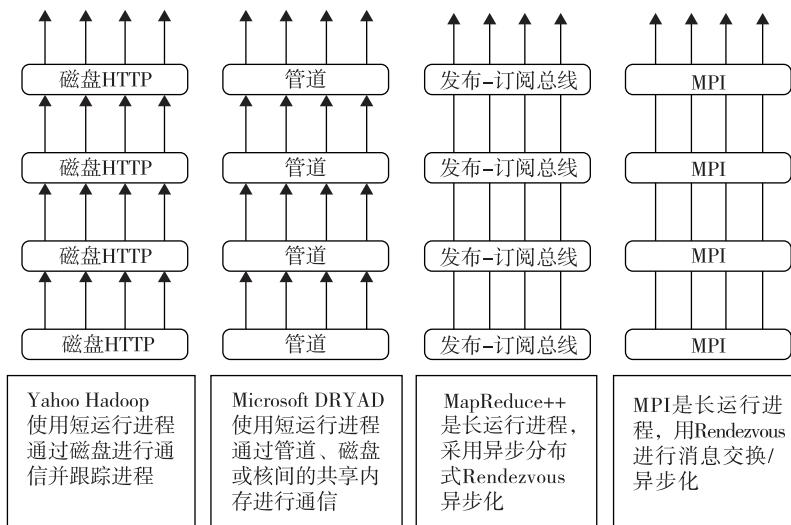


图 6-8 在 MPI、Twister、Hadoop 和 DryadLINQ 上 K 均值集群化的性能

例 6.2 在 ClubWeb 数据集上用 256 处理器核时 Hadoop 和 Twister 的性能

迭代 MapReduce 的重要研究领域包括容错能力和可伸缩的通信方法。图 6-10 表明^[55] 迭代算法是在信息检索中发现的。该图介绍了著名的 Page Rank 算法（带有迭代矩阵向量乘法内核），运行在公共 ClueWeb 数据集上，与大小无关。如图 6-10 中上下两条曲线的间距所示，Twister 比 Hadoop 快了 20 倍。



[354]

图 6-9 运行时并行编程范式的线程和处理结构

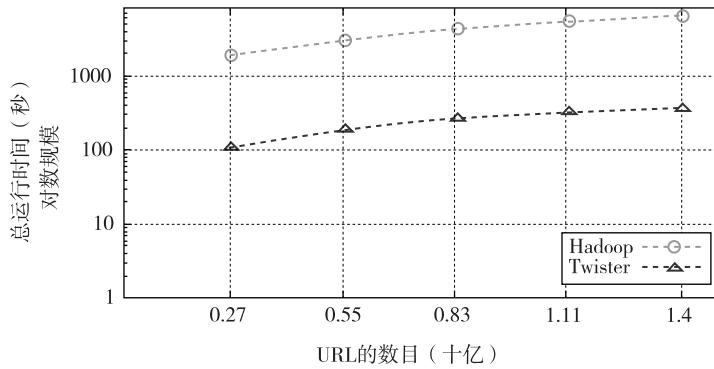


图 6-10 在 ClueWeb 数据集上使用 256 处理核时，Hadoop 和 Twister 性能

6.2.3 来自 Apache 的 Hadoop 软件库

Hadoop 是 Apache 用 Java（而不是 C）编码和发布的 MapReduce 开源实现。MapReduce 的 Hadoop 实现使用 Hadoop 分布式文件系统（Hadoop Distributed File System, HDFS）作为底层，而不是 GFS。Hadoop 内核分为两个基本层：MapReduce 引擎和 HDFS。MapReduce 引擎是运行在 HDFS 之上的计算引擎，使用 HDFS 作为它的数据存储管理器。下面两节内容涵盖了这两个基本层的具体细节。

HDFS：HDFS 是一个源于 GFS 的分布式文件系统，是在一个分布式计算系统上管理文件和存储数据。

HDFS 体系结构：HDFS 有一个主从（master/slave）体系结构，包括一个单个 NameNode 作为 master 以及多个 DataNodes 作为工作机（slave）。为了在这个体系结构中存储文件，HDFS 将文件分割成固定大小的块（例如 64MB），并将这些块存到工作机（DataNodes）中。从块到 DataNodes 的映射是由 NameNode 决定的。NameNode（master）也管理文件系统的元数据和命名空间。在这个系统中，命名空间是维护元数据的区域，而元数据是指一个文件系统存储的所有信息，它们是所有文件的全面管理所需要的。例如，元数据中的 NameNode 存储了所有 DataNodes 上关于输入块位置的所有信息。每个 DataNode，通常是集群中每个节点一个，管理这个节点上的存储。每个 DataNode 负责它的文件块的存储和检索^[61]。

[355]

HDFS 特性：分布式文件系统为了能高效地运作，会有一些特殊的需求，比如性能、可扩展性、并发控制、容错能力和安全需求^[62]。然而，因为 HDFS 不是一个通用的文件系统，即它仅执行特殊种类的应用，所以它不需要一个通用分布式文件系统的所有需求。例如，HDFS 系统从不支持安全性。下面的讨论着重突出 HDFS 区别于其他一般分布式文件系统的两个重要特征^[63]。

HDFS 容错能力：HDFS 的一个主要方面就是容错特征。由于 Hadoop 设计时默认部署在廉价的硬件上，系统硬件故障是很常见的。所以，Hadoop 考虑以下几个问题来达到文件系统的可靠性要求^[64]：

- **块复制：**为了能在 HDFS 上可靠地存储数据，在这个系统中文件块被复制了。换句话说，HDFS 把文件存储为一个块集，每个块都有备份并在整个集群上分发。备份因子由用户设定，默认是 3。
- **备份布置：**备份的布置是 HDFS 实现所需要的容错功能的另一个因素。虽然在整个集群的不同机架的不同节点上（DataNodes），存储备份提供了更大的可靠性，但这有时会被忽略，因为不同机架上两个节点之间的通信成本要比同一个机架上两个不同节点之间的通信相对要高。所以，有时 HDFS 会牺牲可靠性来降低通信成本。例如，对于缺省的备份因子 3，HDFS 存储一个备份在原始数据存储的那个节点上，一个备份在同一机架的不同节点上，还有一个备份在不同机架的不同节点上，这样来提供数据的三个副本^[65]。
- **Heartbeat 和 Blockreport 消息：**Heartbeats 和 Blockreports 是在一个集群中由每个 DataNode 传给 NameNode 的周期性消息。收到 Heartbeat 意味着 DataNode 正运行正常，而每个 Blockreport 包括了 DataNode 上所有块的一个清单^[66]。NameNode 收到这样的消息，是因为它是系统中所有备份的唯一的决定制订者。

HDFS 高吞吐量访问大规模数据集（文件）：因为 HDFS 主要是为批处理设计的，而不是交互式处理，所以 HDFS 数据访问吞吐量比延时来的更为重要。而且，因为运行在 HDFS 上的应用程序往往有大规模数据集，单个文件会被分成大块（如 64MB）以允许 HDFS 来减少每个文件所需要的元数据存储总量。这里提供了两点优势：每个文件的块列表将随着单个块大小的增加而减少，并且通过在一个块中顺序地保持大量数据，HDFS 提供了数据的快速流读取。

HDFS 操作：HDFS 操作（比如读和写）的控制流能正确突出在管理操作中 NameNode 和 DataNodes 的角色。本节进一步描述了 HDFS 在文件上的主要操作控制流，表明在这样的系统中用户、NameNode 和 DataNodes 之间的交互^[63]。

[356]

- **读取文件：**为了在 HDFS 中读取文件，用户先发送一个“open”请求给 NameNode 以获取文件块的位置信息。对于每个文件块，NameNode 返回包含请求文件副本信息的一组 DataNodes 的地址。地址的数量取决于块副本的数量。一旦收到这样的信息，用户就调用 *read* 函数来连接包含文件第一个块的最近的 DataNode。当第一块从有关的 DataNode 中传至用户后，已经建立的连接将会终止，重复同样的过程来获取所请求文件的全部块，直至整个文件都流到了用户。
- **写入文件：**为了在 HDFS 中写入文件，用户发送一个“create”请求给 NameNode，来在文件系统命名空间里创建一个新的文件。如果文件不存在，NameNode 会通知用户，并允许他调用 *write* 函数开始将数据写入文件。文件第一块被写入一个叫做“数据队列”的内部队列中，并由数据流监视其写入到 DataNode。由于每个文件块都需要由一个预定义的参数来复制，数据流首先发送一个请求给 NameNode，以获取合适的 DataNodes 列表来存储第一个块的备份。

然后，数据流存储这个块到第一个分配的 DataNode。之后，这个块由第一个 DataNode 转发给第二个 DataNode。这个过程一直会持续到所有分配的 DataNode 都从前一个 DataNode 那里收到了第一个块的备份。一旦这个复制过程结束，第二块就会开始同样的流程，并会持续，直到所有

文件块都在文件系统上存储并备份。

6.2.3.1 Hadoop 上的 MapReduce 体系结构

Hadoop 的顶层是 MapReduce 引擎，管理着分布式计算系统上 MapReduce 作业的数据流和控制流。357 图 6-11 给出了 MapReduce 引擎与 HDFS 协作的体系结构。类似于 HDFS，MapReduce 引擎也有一个主/从（master/slave）体系结构，由一个单独的 JobTracker 作为主服务器并由许多的 TaskTracker 作为服务器（slaves）。JobTracker 在一个集群上管理 MapReduce 作业，并负责监视作业和分配任务给 TaskTracker。TaskTracker 管理着集群上单个计算节点的映射和化简任务的执行。

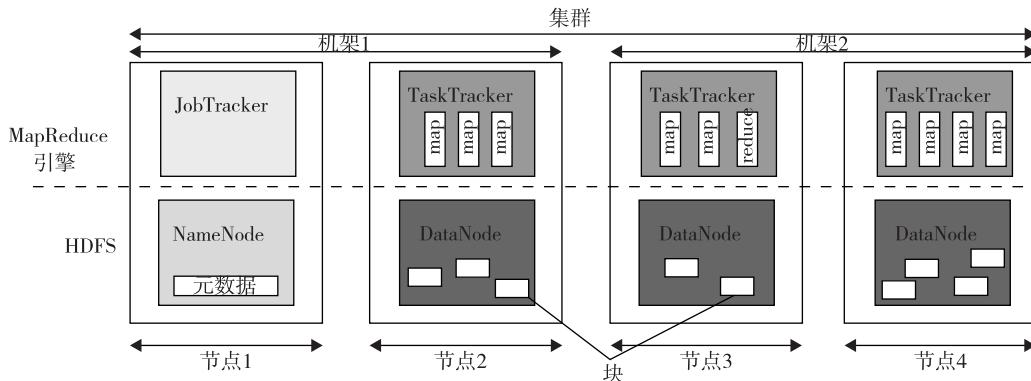


图 6-11 Hadoop 上的 HDFS 和 MapReduce 引擎体系结构，不同阴影的盒子表示应用不同数据块的不同功能节点

每个 TaskTracker 节点都有许多同时运行槽，每个运行是映射任务或者化简任务。插槽是由 TaskTracker 节点的 CPU 支持同时运行的线程数量来确定的。比如，一个带有 N 个 CPU 的 TaskTracker 节点，每个都支持 M 个线程，共有 $M \times N$ 个同时运行的槽^[66]。需要注意的是，每个数据块都是由运行在单独的一个槽上的映射任务处理的。所以，在 TaskTracker 上的映射任务和在各个 DataNode 上的数据块之间存在一一对应关系。

6.2.3.2 在 Hadoop 里运行一个作业

在这个系统中有三个部分共同完成一个作业的运行：用户节点、JobTracker 和数个 TaskTracker。数据流最初是在运行于用户节点上的用户程序中调用 `runJob(conf)` 函数，其中 `conf` 是 MapReduce 框架和 HDFS 中一个对象，它包含了一些调节参数。`runJob(conf)` 函数和 `conf` 如同谷歌 MapReduce 第一次实现中的 `MapReduce(Spec, &Results)` 函数和 `Spec`。图 6-12 描述了在 Hadoop 上运行一个 MapReduce 作业的数据流^[63]。

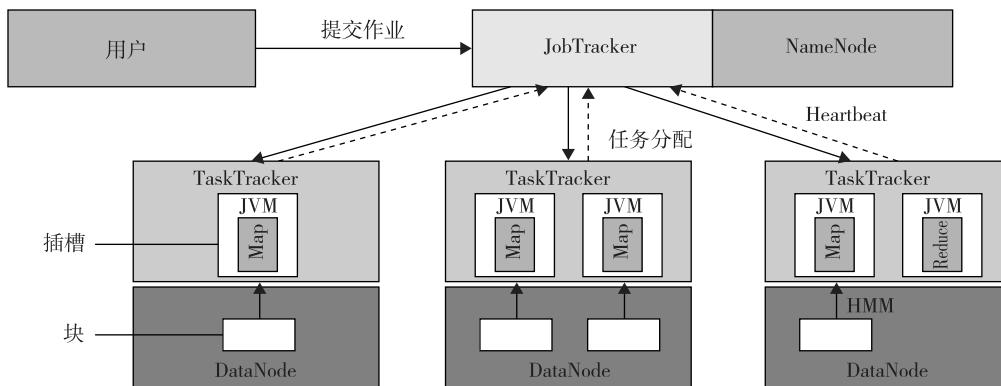


图 6-12 在不同的 TaskTracker 上使用 Hadoop 库运行一个 MapReduce 作业的数据流

- **作业提交：**每个作业都是由用户节点通过以下步骤提交给 JobTracker 节点，此节点可能会位于集群内一个不同的节点上：
 - 一个用户节点从 JobTracker 请求一个新的作业 ID，并计算输入文件分块。
 - 用户节点复制一些资源，比如用户的 JAR 文件、配置文件和计算输入分块，至 JobTracker 文件系统。
 - 用户节点通过调用 `submitJob()` 函数提交任务至 JobTracker。
- **任务分配：**JobTracker 为用户节点的每个计算输入块建立一个映射任务，并分配给 TaskTracker 的执行槽。当分配映射任务给 TaskTracker 时，JobTracker 会考虑数据的定位。JobTracker 也会创建化简任务，并分配给 TaskTracker。化简任务的数量是由用户事先决定 [358] 的，所以在分配时不用考虑位置问题。
- **任务执行：**把作业 JAR 文件复制到其文件系统之后，在 TaskTracker 执行一个任务（不管映射还是化简）的控制流就开始了。在启动 Java 虚拟机（Java Virtual Machine, JVM）来运行它的映射或化简任务后，就开始执行作业 JAR 文件里的指令。
- **任务运行校验：**通过接收从 TaskTracker 到 JobTracker 的周期性心跳监听消息来完成任务运行校验。每个心跳监听会告知 JobTracker 传送中的 TaskTracker 是可用的，以及传送中的 TaskTracker 是否准备好运行一个新的任务。

6.2.4 微软的 Dryad 和 DryadLINQ

本节将介绍并行和分布式计算中的两个运行时软件环境，即微软开发的 Dryad 和 DryadLINQ。

6.2.4.1 Dryad

Dryad 比 MapReduce 更具灵活性，因为 Dryad 应用程序的数据流并非被动或事先决定，并且用户可以很容易地定义。为了达到这样的灵活性，一个 Dryad 程序或者作业由一个有向无环图（DAG）定义，其顶点是计算引擎，边是顶点之间的通信信道。所以，用户或者应用开发者在作业中能方便地指定任意 DAG 来指定数据流。

对于给定的 DAG，Dryad 分配计算顶点给底层的计算引擎（集群节点），并控制边（集群结点之间的通信）的数据流。数据分块、调度、映射、同步、通信和容错是主要的实现细节，这些被 Dryad 隐藏以助于其编程环境。因为这个系统中作业的数据流是任意的，在这里只对运行时环境的控制流做进一步阐述。如图 6-13a 所示，处理 Dryad 控制流的两个主要组件是作业管理器和名字服务器。

在 Dryad 中，分布式作业是一个有向无环图，每个顶点就是一个程序，边表示数据信道。所以，整个作业将首先由应用程序员构建，并定义了处理规程以及数据流。这个逻辑计算图将由 Dryad 运行时自动映射到物理节点。一个 Dryad 作业由作业管理器控制，作业管理器负责把程序部署到集群中的多个节点上。它可以在计算集群上运行，也可以作为用户工作站上的一个可访问集群的进程。作业管理器有构建 DAG 和库的代码，来调度在可用资源上运行的工作。数据传输是通过信道完成，并没有涉及作业管理器。所以作业管理器应该不会成为性能的瓶颈。总而言之，作业管理器：

1. 使用由用户提供的专用程序来构建作业通信图（数据流图）。
2. 从名字服务器上收集把数据流图映射到底层资源（计算引擎）所需的信息。

集群有一个名字服务器，用来枚举集群上所有可用的计算资源。所以，作业管理器就能和名字服务器联系，以得到整个集群的拓扑并制订调度决策。有一个处理后台程序运行在集群的每一个计算节点上。该程序的二进制文件将直接由作业管理器发送至相应的处理节点。后台程序会被视为代理人，以便作业管理器能和远程顶点进行通信，并能监视计算的状态。通过收集这些信息，名字服务器能够提供给作业管理器底层资源和网络拓扑的完美视图。所以作业管理器能够：

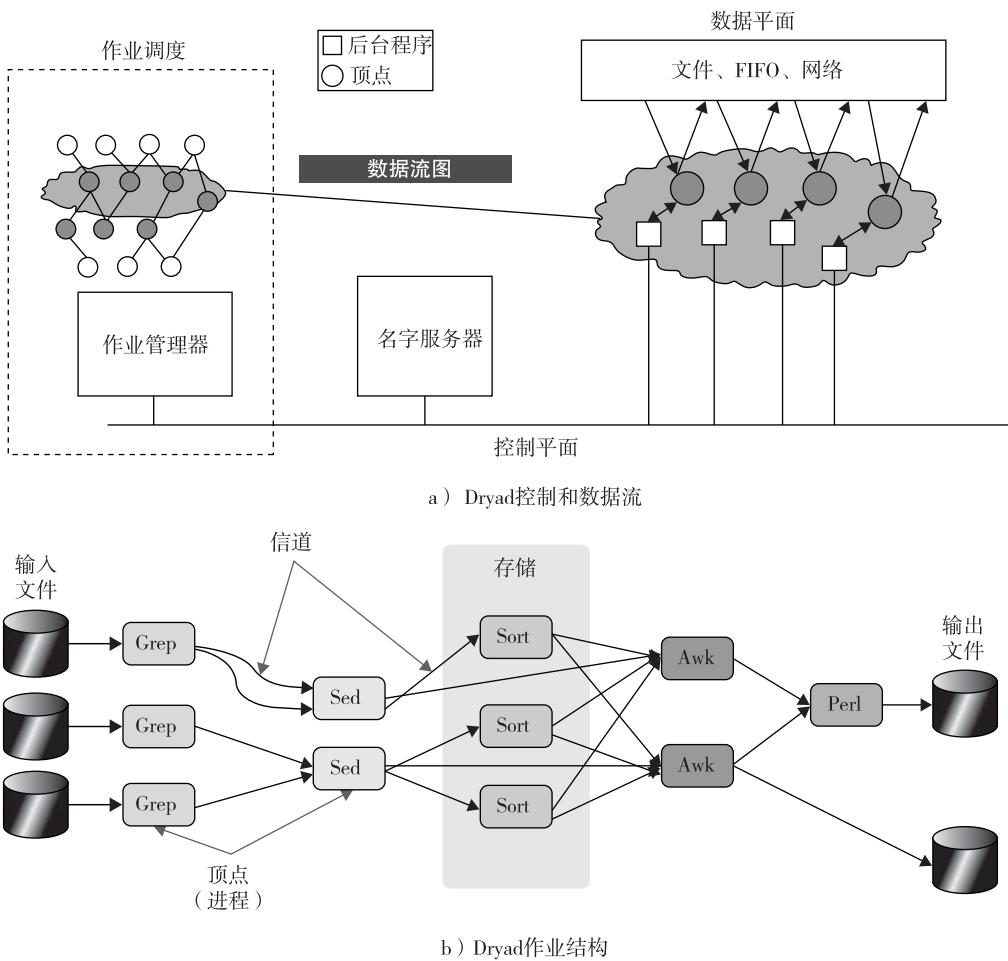


图 6-13 Dryad 体系结构及其作业结构、控制和数据流

1. 把数据流图映射到底层资源。
2. 在各自的资源上调度所有必要的通信和同步。

当映射数据流图到底层资源时，它也考虑数据和计算的位置^[26]。当数据流图映射到一系列计算引擎上时，一个小的后台程序在每个集群节点上运行，以运行分配的任务。每个任务是由用户用一个专用程序定义的。在运行时内，作业管理器和每个后台程序通信，以监视节点的计算状态及其之前和之后节点的通信。在运行时，信道被用来传输代表处理程序的顶点之间的结构化条目。另外，还有几类通信机制来实现信道，如共享内存、TCP 套接字，甚至分布式文件系统。

Dryad 作业的执行可以看做是二维分布式管道集。传统的 UNIX 管道是一维管道，管道里的每个节点作为一个单独的程序。Dryad 的二维分布式管道系统在每个顶点上都有多个处理程序。通过这个方法，可以同时处理大规模数据。图 6-13b 给出了 Dryad 二维管道作业结构。在二维管道执行的时候，Dryad 定义了关于动态地构造和更改 DAG 的很多操作。这些操作包括创建新的顶点、增加图的边、合并两个图，以及处理作业的输入和输出。Dryad 也拥有内置的容错机制。因为它建立在 DAG 上，所以一般会有两种故障：顶点故障和信道故障。它们的处理方式是不一样的。

因为一个集群里有很多个节点，作业管理器可以选择另一个节点来重新执行分配到故障节点上相应的作业。如果是边出现了故障，会重新执行建立信道的顶点，新的信道会重新建立并和相应的节点再次建立连接。除了用来提高执行性能的运行时图精炼外，Dryad 还提供一些其他的

机制。作为一个通用框架，Dryad 能用在很多场合，包括脚本语言的支持、映射 - 化简编程和 SQL 服务集成。

6.2.4.2 微软的 DryadLINQ

DryadLINQ 建立在微软的 Dryad 执行框架之上（参见 <http://research.microsoft.com/en-us/projects/DryadLINQ/>）。Dryad 能执行非周期性任务调度，并能在大规模服务器上运行。DryadLINQ 的目标是能够让普通的程序员使用大型分布式集群计算。事实上，正如其名，DryadLINQ 连接了两个重要的组件：Dryad 分布式执行引擎和 .NET 语言综合查询（Language Integrated Query, LINQ）。图 6-14 描述了使用 DryadLINQ 时的执行流。执行过程分为如下 9 个步骤：

1. 一个 .NET 用户应用运行和创建一个 DryadLINQ 表示对象。由于 LINQ 的延迟评估，表达式的真正执行还没有开始。
2. 应用调用 `ToDryadTable` 触发了一个数据并行的执行。这个表达对象传给了 DryadLINQ。
3. DryadLINQ 编译 LINQ 表达式到一个分布式 Dryad 执行计划。表达式分解成子表达式，每个都在单独的 Dryad 顶点运行。然后生成远端 Dryad 顶点的代码和静态数据，接下来是所需要数据类型的序列化代码。
4. DryadLINQ 调用一个自定义 Dryad 作业管理器，用来管理和监视相应任务的执行流。
5. 作业管理器使用步骤 3 建立的计划创建作业图。当资源可用的时候，它来调度和产生顶点。
6. 每个 Dryad 顶点执行一个与顶点相关的程序。
7. 当 Dryad 作业成功完成，它就将数据写入输出表格。
8. 作业管理器处理结束，它把控制返回给 DryadLINQ。DryadLINQ 创建一个封装有执行输出的本地 `DryadTable` 对象。这里的 `DryadTable` 对象可能是下一个阶段的输入。
9. 控制返回给用户应用。`DryadTable` 上的迭代接口允许用户读取其内容作为 .NET 对象。

[361]

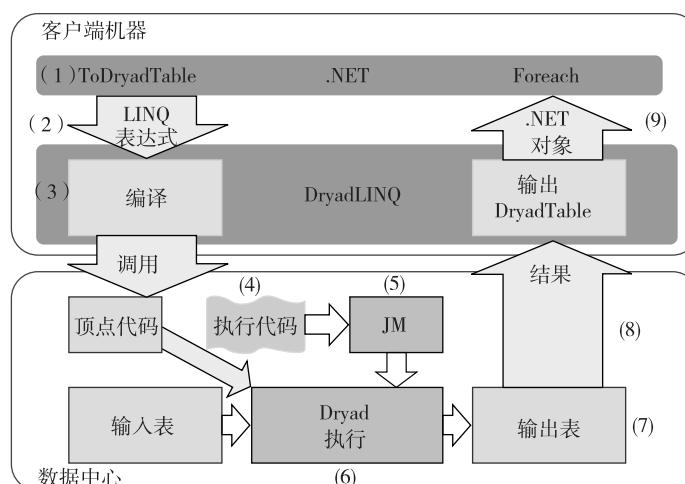


图 6-14 DryadLINQ 上的 LINQ 表达式运行

注：由 Yu 等人提供，OSDI 2008^[27]。

并不是所有程序都会进行完所有 9 个步骤。有些程序可能只进行少数几个步骤。基于以上的描述，DryadLINQ 让用户将当前的编程语言（C#）集成到一个编译器和一个运行时引擎。下面的例子显示了怎样在 DryadLINQ 中编写直方图。

例 6.3 一个用 DryadLINQ 编写的直方图

我们显示了用 DryadLINQ 编写的直方图程序，它是用来计算文本文件中每个单词出现的频

率。整个程序如下所示，用户可以阅读该程序来熟悉 Dryad 编程的高级语言。对于更多的细节，

[362] 用户可以参考 Yu 等人^[27]的文章。

```
[Serializable]
public struct Pair {
    string word;
    int count;

    public Pair(string w, int c)
    {
        word = w;
        count = c;
    }

    public override string ToString()
    {
        return word + ":" + count.ToString();
    }
}

public static IQueryable<Pair> Histogram(
    string directory,
    string filename,
    int k)
{
    DryadDataContext ddc = new DryadDataContext("file://" + directory);
    DryadTable<LineRecord> table =
        ddc.GetPartitionedTable<LineRecord>(filename);
    IQueryable<string> words =
        table.SelectMany(x => x.line.Split(' '));
    IQueryables<IGrouping<string, string>> groups = words.GroupBy(x => x);
    IQueryables<Pair> counts = groups.Select(x => new Pair(x.Key, x.Count()));
    IQueryables<Pair> ordered = counts.OrderByDescending(x => x.Count());
    IQueryables<Pair> top = ordered.Take(k);
    return top;
}
```

例 6.4 单词计数问题直方图的执行举例

这个程序的执行流和 MapReduce 框架的单词计数程序非常相似。表 6-6 显示了如何在样例文本输入上执行这个程序。表格中的每一行是程序中每条代码行的执行结果。程序的执行会涉及一个典型的 DryadLINQ 所有的步骤。

表 6-6 直方图的执行举例

操作符	输出
Table	“A line of words of wisdom”
SelectMany	[“A”, “line”, “of”, “words”, “of”, “wisdom”]
GroupBy	[[“A”], [“line”], [“of”], [“words”], [“wisdom”]]
Select	[{ “A”, 1 }, { “line”, 1 }, { “of”, 2 }, { “words”, 1 }, { “wisdom”, 1 }]
OrderByDescending	[{ “of”, 2 }, { “A”, 1 }, { “line”, 1 }, { “words”, 1 }, { “wisdom”, 1 }]
Take(3)	[{ “of”, 2 }, { “A”, 1 }, { “line”, 1 }]

程序员不用担心程序的并行执行或者考虑容错能力。伸缩性、可靠性和其他一些涉及分布

[363] 式计算机系统的困难问题，都隐藏在 DryadLINQ 框架中。程序更多地会关注应用程序逻辑。这能大大地降低并行数据处理编程所需要的编程技巧。

例 6.5 一个 MapReduce WebVisCounter 程序的 Hadoop 实现

在这个例子中，我们介绍一个在 Hadoop 上编码的实用 MapReduce 程序。这个样例程序叫做 WebVisCounter，它计算了用户使用一个特定的操作系统（例如 Windows XP 或 Linux Ubuntu）来连接或访问一个给定网站的次数。输入数据如下所示。一个典型 Web 服务器日志文件的一行有 8 个域，由制表符或空格隔开，含义如下所示：

1. 176.123.143.12 (连接机器的 IP 地址)

2. — (一个分隔器)
3. [10/Sep/2010:01:11:30 -1100] (访问时间戳, 格式是 DD:Mon:YYYY HH:MM:SS, -11:
00 是格林威治标准时间的偏移)
4. "GET /gse/apply/int_research_app_form.pdf HTTP/1.0" (GET 使用 HTTP/1.0 协议请求文
件, /gse/apply/int_research_app_form.pdf)
5. 200 (状态码, 表示用户的请求成功)
6. 1363148 (传输位的数量)
7. http://www.eng.usyd.edu.au (用户在到达服务器之前从此开始)
8. "Mozilla/4.7[en](WinXp; U)" (浏览器用来取得网址、浏览器的版本、语言版本和操作
系统)

因为输出是用户使用特定操作系统来连接到一个给定网站的次数，*Map* 函数分析了每一行，得到所使用的操作系统类型（如 WinXP）作为一个 key，并给它分配一个值（在这个例子中是 1）。反过来，*Reduce* 函数对每个唯一 key（在这个例子中是操作系统类型）的 1 的数量求和。图 6-15 描述了 WebVisCounter 程序的相关数据流。

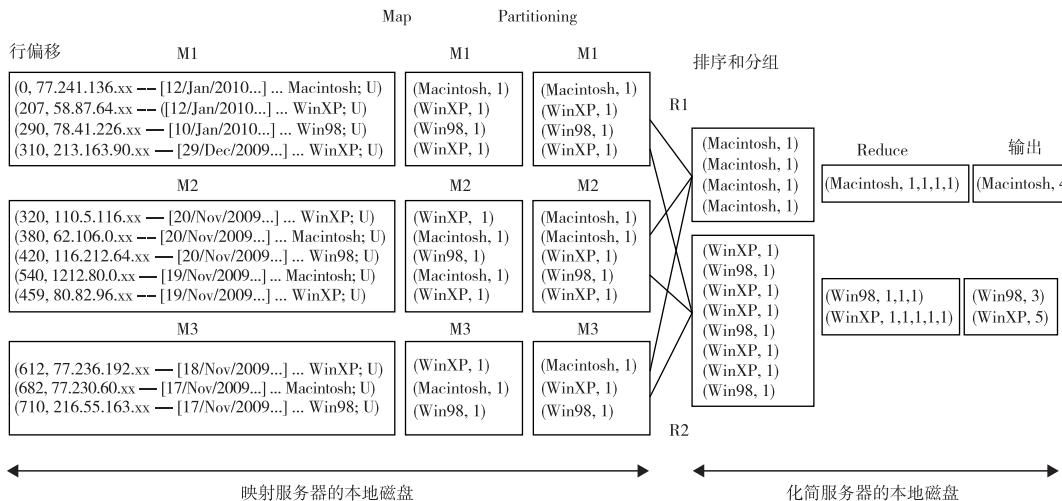


图 6-15 WebVisCounter 程序运行的数据流

6.2.5 Sawzall 和 Pig Latin 高级语言

Sawzall 是建立在谷歌的 MapReduce 框架之上的一种高级语言^[31]。Sawzall 是一种脚本语言，能进行并行数据处理。和 MapReduce 一样，Sawzall 能对大规模数据集，甚至对整个互联网上收集的数据规模进行分布式、容错处理。Sawzall 是由 Rob Pike 开发的，其最初的目标是处理谷歌日志文件。在这个方面，它已经取得了巨大的成功，并且使得大批企业变成了交互式进程，这样使用这些数据的新方法可以被开发。图 6-16 描述了 Sawzall 框架的数据流和处理过程的整个模型。Sawzall 最近作为一个开源项目被发布。

首先，数据用现场处理脚本在本地被分块和处理。本地数据会被过滤，以得到必需的信息。然后，使用整合器根据已发送的数据来获取最后的结果。很多谷歌的应用都符合这个模型。用户使用 Sawzall 脚本语言编写其应用。Sawzall 运行时引擎将相应的脚本翻译为能够运行在很多节点上的 MapReduce 程序。Sawzall 程序能够自动利用集群计算的威力，也能够从冗余服务器上获取可靠性。

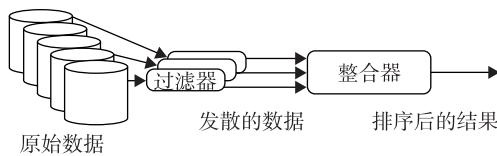


图 6-16 Sawzall 的过滤、整合和校对的整个流

例 6.6 Sawzall 上的文件摘要程序

这里是一个使用 Sawzall 来在集群上进行数据处理的简单例子（该例子来自文献[31]的 Sawzall 论文）。假定我们需要处理一系列文件，每个文件都有记录，并且每个记录都包含一个浮点数。我们想要计算记录的数量、总值和平方和。相关代码如下：

```
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```

前三行声明了整合器：`count`、`total` 和 `sum_of_squares`。`table` 是一个关键字，定义了整合器类型。这些特殊表格是 `sum` 表格，它们会自动对发送给它们的值进行求和。对于每个输入记录，Sawzall 初始化预定义的变量输入为输入的未解释字节串。`x:float = input;` 这一行将输入转换成一个浮点型数据，并存储在本地变量 `x` 中。三个 `emit` 语句将中间值发送给整合器。人们可以将 Sawzall 脚本转化成 MapReduce 程序，并运行在多服务器上。 ■

Pig Latin 是雅虎开发的一种高级数据流语言^[33]，并且已经在 Apache Pig 项目^[67]中 Hadoop 之上实现了。Pig Latin、Sawzall 和 DryadLINQ 是在 MapReduce 及其扩展之上构建语言的三种不同方法。它们之间的比较见表 6-7。

表 6-7 高级数据分析语言的比较

	Sawzall	Pig Latin	DryadLINQ
提出者	谷歌	雅虎	Microsoft
数据模型	谷歌协议缓存或基本	原子，元组，包，映射	分区文件
类型	静态的	动态的	静态的
种类	解释型	编译型	编译型
编程风格	命令式	过程的：说明步骤序列	命令和说明
和 SQL 相似程度	最少	中等	非常多
可扩展性（用户定义功能）	没有	有	有
控制结构	有	没有	有
执行模型	记录操作 + 固定集群化	MapReduce 操作序列	有向无环图
目标运行时	谷歌 MapReduce	Hadoop (Pig)	Dryad

DryadLINQ 是直接在 SQL 上建立的，而其他两种语言是 NOSQL 的继承，尽管 Pig Latin 支持主要 SQL 架构，包括 Join，而 Sawzall 则不支持。每种语言自动实现并行性，因此只需要考虑单个元素的操作，然后调用支持的共同操作。当然这是可能的，因为所需要的并行可以被独立的任务清楚地实现，“副作用”只是出现在所支持的共同操作上。这是一个重要的实现并行化的通用方法，并且很久以前就在高性能 Fortran^[68] 上得到了体现。在文献[69, 70] 中关于 Pig 和 Pig Latin 有一些讨论，这里我们总结了语言特性。表 6-8 列出了 Pig Latin 的 4 种数据类型，表 6-9 列出了 14 个操作符。

364
365

表 6-8 Pig Latin 数据类型

数据类型	描述	举例
原子	简单原子值	'Clouds'
元组	任意 Pig Latin 类型的字段序列	('Clouds', 'Grids')
包	元组的集合，每个包成员都允许不同的模式	{ ('Clouds', 'Grids') ('Clouds', ('IaaS', 'PaaS')) }
映射	和一组键值相关的数据项的集合。键值是原子数据包	['Microsoft' → { ('Windows') ('Azure') } 'Redhat' → 'Linux']

表 6-9 Pig Latin 操作符

指令	描述
LOAD	从文件系统中读取数据
STORE	将数据写入文件系统
FOREACH GENERATE	为每个记录应用一个表达式，输出一个或多个记录
FILTER	申请一个谓词，并将返回值不为真的记录删除
GROUP/COGROUP	从一个或多个输入中选出有相同键值的记录
JOIN	基于一个键值来连接两个或更多的输入
CROSS	对于两个或多个输入做叉积
UNION	合并两个或更多数据集
SPLIT	基于过滤条件将数据分裂成两个或更多的集合
ORDER	基于一个键值对记录排序
DISTINCT	删除重复元组
STREAM	通过一个用户提供的二进制来发送所有记录
DUMP	将输出写到标准输出设备
LIMIT	限制记录的数量

例 6.7 使用 Sawzall 和 Pig Latin 的并行编程

首先我们使用命令读取数据，例如：

```
Queries = LOAD 'filewithdata.txt' USING myUDF() AS (userId, queryString, timestamp);
```

LOAD 命令返回一个句柄到 `Queries` 包。`myUDF()` 是一个可选的自定义读取器，是用户自定义函数的一个例子。AS 语法定义了组成 `Queries` 包的元组模式。现在数据可以由命令来处理，例如：

```
Expanded_queries = FOREACH queries GENERATE userId, expandQueryUDF(queryString);
```

该例子映射了 `queries` 中的每个元组，这由用户自定义函数 `expandQueryUDF` 来决定。FOREACH 运行了包中所有的元组。另外，用户可以使用 FILTER 根据 `userId` 是否等于字符串 `Alice` 来删除所有元组，如下例所示：

```
Real_queries = FILTER queries BY userId neq 'Alice';
```

Pig Latin 使用 COGROUP 提供了 SQL JOIN 的等效功能，例如：

```
Grouped_data = COGROUP results BY queryString, revenue BY queryString;
```

其中 `results` 和 `revenue` 是元组包（来自 LOAD 或加载数据的处理）

```
Results: queryString, url, position  
Revenue: (queryString, adSlot, amount)
```

从 COGROUP 并不产生一系列元组 (`queryString, url, position, adSlot, amount`)，而是一个包含三个字段的元组的意义上说，COGROUP 比 JOIN 更通用。第一个字段是 `queryString`，第二个字段是来自 `Results` 的所有元组，带有 `queryString` 的值，第三个字段是来自 `Revenue` 的所有元组，带有 `queryString` 的值。FLATTEN 能将 COGROUP 的结果映射到 SQL Join (`queryString, url, position, adSlot, amount`) 语法。 ■

Pig Latin 操作符按照数据流管道中列出的顺序来执行。这和说明性 SQL 形成对比，只需要指定需要做“什么”；而不是怎么做。Pig Latin 支持用户自定义函数作为语言中第一类操作，如上面举例的代码，可能是超过 SQL 的一个优势。根据用户的偏好，用户自定义函数能放于 `Load`、`Store`、`Group`、`Filter` 和 `Foreach` 操作符中。注意，Pig Latin 允许了丰富的数据流操作符，这很类似于 5.5.5 节中介绍的一个工作流的脚本方法。Pig! Apache 项目^[69] 把 Pig Latin 映射到一组 Hadoop 实现的 MapReduce 操作序列。

6.2.6 并行和分布式系统的映射应用

过去，Fox 从 5 个应用体系结构的角度讨论了不同硬件和软件的映射应用程序^[71]。最初的 5 个类别在表 6-10 中已经列出来，紧跟着的第 6 类描述了数据密集计算^[72,73]。最初的分类大体上描述了模拟，它们并不是直接针对数据分析的。简略概括并解释新的类别还是很有好处的。类别 1 在 20 年前比较风行，但现在已经不重要了。它描述了由硬件控制的应用，可以被锁级（lock-step）操作并行化。

表 6-10 并行和分布式系统的应用分类

类别	分 类	描 述	机器体系结构
1	同步	如同 SIMD 体系结构，问题的分类可以由指令层锁级操作来实现	SIMD
2	宽同步（BSP 或块同步处理）	这些问题显示了迭代计算 - 通信策略，每个 CPU 都有与一个通信步骤同步的独立计算（映射）操作。这类问题包含了很多成功的 MPI 应用，包括偏微分方程的求解和质点动力学应用	MPP 上的 MIMD (大规模并行处理器)
3	异步	例子是计算象棋和整数规划；组合搜索通常是由动态线程所支持的。这在科学计算中算不上重要，但是这是操作系统的核心，并发出现在用户应用程序（比如 Microsoft Word）中	共享内存
4	乐意并行	每个组件都是独立的。在 1988 年，Fox 估计此项占整个应用程序的 20%。但是这个比例一直在增加，这是因为网格和数据分析应用软件的使用，比如粒子物理学的大规模强子对撞器分析	网格移至云
5	元问题	这里有第 1~4 类和第 6 类的粗粒度（异步或数据流）组合。这个领域的重要性也在增加，得到网格的很好支持，由 3.5 节的工作流描述	集群的网格
6	MapReduce ++ (Twister)	它描述了文件（数据库）到文件（数据库）的操作，有三个子分类（参见表 6-11）： 6a) 只有乐意并行映射（类似于第 4 类） 6b) 映射然后化简 6c) 迭代“映射然后化简”（当前技术的扩展，支持线性代数和数据挖掘）	数据密集型云 a) Master- Worker 或 Mapreduce b) MapReduce c) Twister

这样的配置将运行在单指令多数据（Single-Instruction and Multiple-Data，SIMD）机器上，而第 2 类现在显得尤为重要，也就相当于运行在多指令多数据（Multiple Instruction Multiple Data，MIMD）机器上的单一程序多重数据（Single-Program and Multiple Data，SPMD）模型。这里每个分解后的单元执行相同的程序，但是不是在任意时候都必须执行相同的指令。第 1 类相当于常规问题，而第 2 类包括了动态的不规律问题，包括求解偏微分方程式或者质点动力学的复杂几何。注意，同步问题依然存在，但它们是以 SPMD 模型运行在 MIMD 机器上。还要注意第 2 类包括了

计算 – 通信阶段，计算通过通信来实现同步。不需要辅助的同步。

第3类包括了异步互动对象，并且通常是一些人们关于一个典型并行问题的观点。它可能的确描述了在一个现代操作系统中的并发线程，以及一些重要的应用程序，比如事件驱动模拟和在计算机游戏与图形算法中的搜索区域。共享内存是很自然的，因为执行动态同步化常常需要低延迟。在过去这一点并不是很清楚，但是现在看来这个类别在重要的大规模并行问题中不是很普遍。

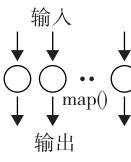
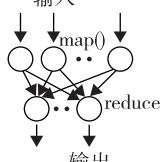
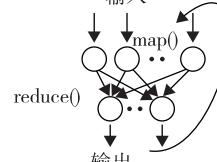
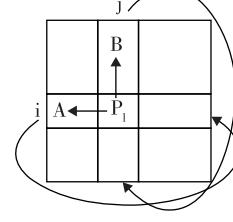
368
369

第4类在算法上是最简单的，它并不和并行组件相连接。然而，从最初1988年分析看来，这个类别可能已经变得越来越重要，因为估计它将占20%的并行计算。网格和云在这类上都很自然，它们不需要在不同节点之间进行高性能通信。

第5类元问题指的是不同“原子”问题的粗粒度连接，这完全包含在5.5节中。很显然，这个领域非常普遍，并且会变得更为重要。回想5.5节的重要观察，我们使用了一个两层编程模型，这个模型有一个指定样式的元问题（工作流）连接，以及采用本章提到的解决方法的组件问题。网格或者云都适用于元问题，因为粗粒度分解通常并不需要很高的性能。

如前所述，我们增加了第6类来包括数据密集型应用，这是由MapReduce作为一个新的编程模型所提出的。我们称这个类为MapReduce++，它有三个子类：“只映射”应用程序类似于第4类的乐意并行^[41,43,74,75]；经典MapReduce，带有文件到文件操作，包括并行映射和接下来的并行化简操作；6.2.2节介绍的捕获扩展版本的MapReduce子类。注意，第6类是第2类和第4类的子集，增加了数据的读和写，以及与数据分析相比特殊化的宽同步结构。这个比较在表6-11中有更详细的介绍。

表6-11 MapReduce++子类别和MPI使用的宽同步类型的比较

只映射	经典MapReduce	迭代MapReduce	宽同步
 <ul style="list-style-type: none"> • 文档转换（比如，PDF → HTML） • 密码学中的强力搜索 • 参数清除 • 基因组合 • PolarGrid Matlab 数据分析 (www.polargrid.org) 	 <ul style="list-style-type: none"> • 高能物理学（HEP）直方图 • 分布式搜索 • 分布式排序 • 信息检索 • 序列的逐对距离的计算（BLAST） 	 <ul style="list-style-type: none"> • 期望最大化算法 • 线性代数 • 数据挖掘，包括 <ul style="list-style-type: none"> • 集群化 • K-均值 • 确定性退火集群化 • 多维伸缩（MDS） 	 <ul style="list-style-type: none"> • 很多MPI科学应用，利用包括本地交互的多种通信架构 • 求解微分方程和带有短程力的质点动力学
← MapReduce域和迭代扩展 →			MPI

6.3 GAE的编程支持

4.4.2节在图4-20中介绍了GAE基础设施。本节我们会描述由GAE支持的编程模型。GAE平台的接入链路在第4章中已经给出。

6.3.1 GAE编程

已经有一些网络资源（例如<http://code.google.com/appengine/>）、特定书籍和文章（例

如 <http://www.byteonic.com/2009/overview-of-java-support-in-google-app-engine/>) 在讨论如何 GAE 编程。图 6-17 总结了对于两种支持语言 Java 和 Python, GAE 编程环境的一些主要特性。客户端环境包括一个 Java 的 Eclipse 插件, 允许你在本地机器上调试自己的 GAE。对于 Java Web 应用程序开发者来说, 还有一个 GWT (谷歌 Web 工具集) 可用。开发者可以使用它, 或其他任何借助于基于 JVM 的解释器或编译器的语言, 如 JavaScript 或 Ruby。Python 会经常和 Django 或者 CherryPy 之类的框架一起使用, 但是谷歌也提供一个内置的 webapp Python 环境。

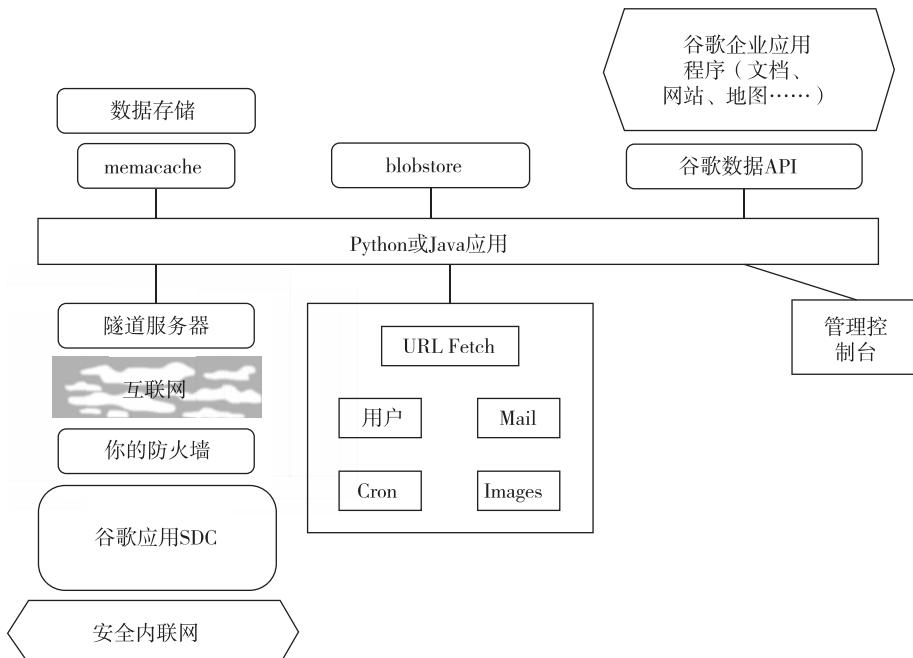


图 6-17 GAE 编程环境

关于存储和读取数据, 有一些很强大的构造。数据存储是一个 NOSQL 数据管理系统, 实体的大小至多是 1MB, 由一组无模式的属性来标记。查询能够检索一个给定类型的实体, 这是根据属性值来过滤和排序的。Java 提供一个 Java 数据对象 (Java Data Object, JDO) 和 JPA (Java Persistence API) 接口, 是由开源 Data Nucleus Access 平台来实现, 而 Python 有一个类似 SQL 的查询语言称为 GQL。数据存储非常一致, 它使用的是最优化并发控制。

如果其他进程试图同时更新同一个实体, 那么这个实体的更新是发生在一个事务处理中, 并且重试固定的次数。你的应用程序可以在单个事务处理中执行多数据存储操作, 结果要么一起成功, 要么一起失败。数据存储使用“实体群组”实现了贯穿其分布式网络的事务。事务在单个群组中操作实体。为了有效地运行事务, 这些同一群组的实体被存储在一起。当实体创建的时候, 你的 GAE 应用可以把实体分配给群组。通过使用 memcache 的内存高速缓存, 数据存储的性能可以提高, 它也可以在数据存储中被独立使用。

最近谷歌增加了 blobstore, 适用于保存大文件, 因为它的文档大小限制在 2GB。有几种机制可以用来和外部资源进行合作。谷歌安全数据连接 (Secure Data Connection, SDC) 能够和互联网建立隧道连通, 并能将内联网和一个外部 GAE 应用相连。URL Fetch 操作保障了应用程序能够使用 HTTP 和 HTTPS 请求获取资源, 并与互联网上的其他主机进行通信。有一个专门的邮件机制, 从你的 GAE 应用程序中发送电子邮件。

应用程序能够使用 GAE URL 获取服务访问互联网上的资源，比如 Web 服务或者其他数据。URL 获取服务使用相同的高速谷歌基础设施来检索 Web 资源，这些谷歌基础设施是为谷歌的很多其他产品来检索网页的。还有许多谷歌“企业”设施，包括地图、网站、群组、日程、文档和 YouTube 等。这些支持谷歌数据 API，它能在 GAE 内部使用。

一个应用程序可以使用谷歌账户来进行用户认证。谷歌账户处理用户账户的创建和登录，如果一个用户已经有了谷歌账户（如一个 Gmail 账户），他就能用这个账户来使用应用程序。GAE 使用一个专用的 Images 服务来处理图片数据，能够调整大小、旋转、翻转、裁剪和增强图片。一个应用程序能够不响应 Web 服务来执行任务。你的应用程序能够执行一个你配置的调度表上的任务，比如以每天或每小时的标准，使用由 Cron 服务处理的“时钟守护作业”（cron job）。

另外，应用程序能执行由应用程序本身加入到一个队列中的任务，比如处理请求时创建的一个后台任务。配置一个 GAE 应用消耗的资源有一定上限或者固定限额。有了固定限额，GAE 保证应用程序不会超出预算，其他运行在 GAE 上的应用程序也不会影响应用的性能。特别地，GAE 按照某个限额来使用是免费的。

6.3.2 谷歌文件系统 (GFS)

GFS 主要是为谷歌搜索引擎的基础存储服务建立的。因为网络上抓取和保存的数据规模非常大，谷歌需要一个分布式文件系统，在廉价、不可靠的计算机上存储大量的冗余数据。没有一个传统的分布式文件系统能够提供这样的功能，并存储如此大规模的数据。另外，GFS 是为谷歌应用程序设计的，并且谷歌应用程序是为谷歌而建立。在传统的文件系统设计中，这种观念不会有吸引力，因为在应用程序和文件系统之间应该有一个清晰的接口，比如 POSIX 接口。

有几个关于 GFS 的假设。其中一个与云计算硬件基础设施的特性有关（如高组件故障率）。因为服务器是由廉价的商业组件构成的，所以一直会有并发故障，这是很常见的现象。另一个关系到 GFS 中文件的大小。GFS 会拥有大量的大规模文件，每个文件可在 100MB 以上，数 GB 的文件也很常见。因此，谷歌选择文件数据块大小为 64MB，而不是典型的传统文件系统中的 4KB。谷歌应用程序的 I/O 模式也很特别。文件一般只写入一次，写操作一般是附加在文件结尾的数据块上。多个附加操作可能会同时进行。会有大量的大规模流读取以及很少量的随机存取。至于大规模流读取，高持续吞吐量比低延迟来的更为重要。

因此，谷歌关于 GFS 的设计做出了一些特殊决策。如前所述，选择 64MB 块大小。使用复制来达到可靠性（比如，每个大块或者一个文件的数据块在多于三个块服务器上进行复制）。单个主服务器可以协调访问以及保管元数据。这个决策简化了整个集群的设计和管理。开发者不需要考虑许多分布式系统中的难题，例如分布式一致。GFS 中没有数据高速缓存，因为大规模流读取和写入既不代表时间也不代表空间的近邻性。GFS 提供了相似但不相同的 POSIX 文件系统访问接口。其中明显的区别是应用程序甚至能够看到文件块的物理位置。这样的模式可以提高上层应用程序。自定义 API 能够简化问题，并聚焦在谷歌应用上。自定义 API 加入了快照和记录附加操作，以利于建立谷歌应用程序。[373]

图 6-18 描述了 GFS 体系结构。很明显在整个集群上只有一个主服务器。其他节点是作为块服务器来存储数据，而单个主服务器用来存储元数据。文件系统命名空间和锁定工具是由主机来管理的。主机周期性地和块服务器进行通信，来收集各种管理信息以及向块服务器发送指令，来完成负载均衡或者故障修复之类的工作。

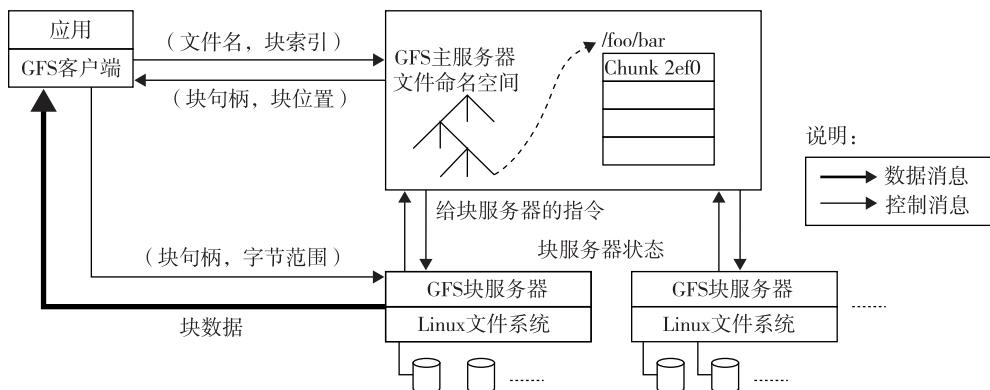


图 6-18 GFS 体系结构

注：由 S. Ghemawat 等人提供^[53]。

主机有足够的信息来保持整个集群在一个良好的状态。使用了单个的主机，就能避免很多复杂的分布式计算，系统的设计也能简化。然而这样的设计有一个潜在的缺点，因为单个 GFS 主服务器可能会成为性能瓶颈和唯一故障点。为了减轻这个缺点，谷歌使用一个影子主服务器，复制了主服务器上的所有数据。这个设计也能保证在客户端和块服务器之间的所有数据操作都能直接执行。控制消息在主服务器和客户端之间传输，并能缓存起来以备以后使用。使用市场上服务器的现有性能，单个主服务器能处理一个大小超过 1 000 个节点的集群。

图 6-19 描述了 GFS 中的数据变异（写入或增加操作）。[374] 在所有副本中都必须创建数据块。目的是尽量减少主机的参与。变异采用如下的步骤：

1. 客户端询问主机哪个块服务器掌握了当前发行版本的块和其他副本的位置。如果没有发行版本，那么主机授权给一个它挑选的副本（没有显示）。

2. 主机回复了主版本的身份和其他（第二级）副本的位置。客户端缓存这个数据以备将来的变异。只有当主版本变的不可达或回复它不再拥有一个发行版时，它才需要重新和主机联系。

3. 客户端将数据推送给所有副本。客户端可以按任意顺序推送数据。每个块服务器将数据存储在一个内部 LRU 缓存区，直到数据被使用了或失效了。通过将数据流和控制流解耦合，对基于网络拓扑的高代价数据流进行调度，我们就可以提高性能，而不用考虑哪个块服务器是主要的。

4. 一旦所有副本都确认接收数据，客户端就将写请求送至主要版本。该请求区分出之前送至所有副本的数据。主要版本分配连续序列号至它收到的所有变异，这些变异可能是来源于多个客户端，并提供了必要的序列化。它按照顺序将变异应用到它自己的本地状态。

5. 主要版本转发写请求到对所有二级副本。每个二级副本请求按照主要版本分配的相同序列号应用变异。

6. 第二级都回复主要版本，来表明操作已经完成了。

7. 主要版本回复客户端。在任何副本遇到的任何错误都会报告给客户端。如果发生错误，写会在主要版本和任意第二级副本的子集进行纠正。客户端请求会被认为失败，改进区域会停

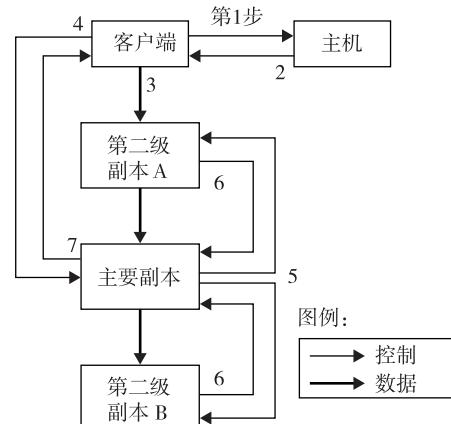


图 6-19 GFS 中的数据变异序列

留在一个不一致的状态。我们的客户端代码通过重试发生故障的变异来处理这样的错误。从返回重试最开始写之前，会从第3步到第7步做一些尝试。

所以，除了由GFS提供的写操作外，一些特殊的附加操作会用来附加数据块到文件尾部。提供这种操作是因为有些谷歌应用程序需要很多附加操作。例如，当网络爬虫从网络中收集数据时，网页内容将被附加在页面文件上。所以才提供并优化了附加操作。客户端指定附加的数据，GFS至少一次将它附加到文件上。GFS挑选偏移量，客户端不能决定数据位置的偏移量。附加操作适用于并发的书写者。

GFS是为高容错设计的，并采纳了一些方法来达到这个目标。主机和块服务器能够在数秒之内重启，有了这么快的恢复能力，数据不可使用的时间窗口将大大减少。正如上文中提到的，每个块至少在三个地方上备份，并且在一个数据块上至少能够容忍两处数据崩溃。影子主机用来处理GFS主机的故障。对于数据完整性，GFS在每个块上每64KB就进行校验和。有了前面讨论过的设计和实现，GFS可以达到高可用性、高性能和大规模的目标。GFS证明了如何在商业硬件上支持大规模处理负载，这些硬件被设计为可容忍频繁的组件故障，并且为主要附加和读取的大规模文件进行了优化。

[375]

6.3.3 BigTable——谷歌的NOSQL系统

本节我们继续讨论谷歌云环境的关键技术。我们已经在6.2.2节中介绍了最著名的MapReduce技术以及6.2.5节的Sawzall。这里，我们将关注另一个谷歌创新技术：BigTable。我们将在6.3.4节介绍Chubby和前面章节已描述的GFS。

BigTable提供了一个服务，用来存储和检索结构化与半结构化的数据。BigTable应用包括网页、每个用户数据和地理位置的存储。这里，我们使用网页来代表URL及其相关数据，比如内容、爬取元数据、链接、锚和网页评分值等。每个用户的数据拥有特定用户的信息，包括这样的数据，例如用户优先设置、最近查询/搜索结果以及用户电子邮件。地理地址是在谷歌地图软件上使用的。地理位置包括物理实体（商店、餐馆等）、道路，卫星影像数据以及用户标注。

这样的数据规模是相当大的。会有数十亿的URL，每个URL都有很多版本，每个版本的平均网页大小是20KB。用户规模也很巨大，会有上亿之多，每秒钟就会有数千次查询。相同规模也会出现在地理数据上，这可能会消耗超过100TB的磁盘空间。

使用商用数据库系统来解决如此大规模结构化或半结构化的数据是不可能的。这是重建数据管理系统的一个原因；产生的系统可以以较低的增量成本应用在很多项目中。重建数据管理系统的另一个动机是性能。低级存储优化能显著地提升性能，但如果运行在传统数据库层之上，则会困难得多。

BigTable系统的设计和实现有以下的目标。应用程序需要异步处理来连续更新不同的数据块，并且需要在任意时间访问大部分的当前数据。数据库需要支持很高的读/写速率，规模是每秒数百万的操作。另外，数据库还需要在所有或者感兴趣的数据子集上支持高效扫描，以及大规模一对一和一对多的数据集的有效连接。应用程序有可能需要不时地检测数据变化（比如一个网页多次爬取的内容）。

因此，BigTable能够看做是分布式多层映射。它像存储服务一样提供了容错能力和持续数据库。BigTable系统是可扩展的，这意味着系统有数千台服务器、太字节内存数据、拍字节基于磁盘的数据、每秒数百万的读/写和高效扫描。BigTable也是一个自我管理的系统（例如，服务器能动态地增加/移除，也能自动负载均衡）。BigTable的设计/最初实现开始于2004年初。BigTable在很多项目中使用，包括谷歌搜索、Orkut、谷歌地图/谷歌地球等。一个最大的BigTable核管理了分布在数千台机器中的大约200TB的数据。

BigTable系统建立在现有的谷歌云基础设施之上。BigTable使用如下的构建模块：

1. GFS: 存储持续状态
2. 调度器: 涉及 BigTable 服务的调度作业
3. 锁服务: 主机选择, 开机引导程序 (bootstrapping) 定位
4. MapReduce: 通常用来读/写 BigTable 数据

376

例 6.8 在大规模媒质中使用的 BigTable 数据模型

BigTable 提供了一个比传统数据库系统更简化的数据模型。图 6-20a 描述了一个 Web Table 表格实例的数据模型。Web Table 存储了有关网页的数据。每个网页都能由 URL 来访问。URL 被当做行索引。列提供了和相应的 URL 相关的不同数据——比如, 内容的不同版本和网页中出现的锚。在这个意义上, BigTable 是一个分布式多维稀疏存储映射。

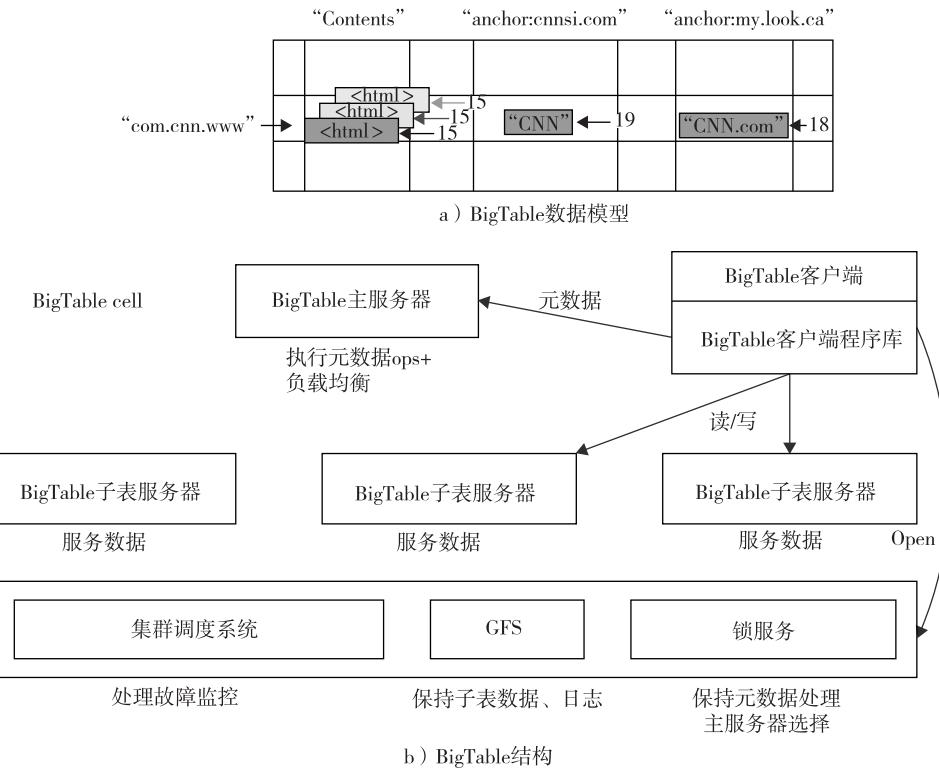


图 6-20 BigTable 数据模型和系统结构

注: 由 Chang 等人提供^[11]。

这个映射按照行键值、列键值和时间戳来索引, 即 (row: string, column: string, time: int64) 映射到 string (cell contents)。行是由行键值根据字典顺序来排序的。一个表格的行范围是动态分块的, 每个行范围被称为“子表” (Tablet)。列的语法是 (family: qualifier) 对。单元格可以存储带时间戳的多个数据版本。

这种数据模型对于大多数谷歌 (或其他组织) 应用来说都是不错的匹配。对于行, Name 是一个任意字符串, 对于行数据的访问是原子操作。这和传统关系数据库是不同的, 传统数据库提供了冗余的原子操作 (事务)。行创建在存储数据上是隐式的。行是按照字典顺序进行排序的, 通常是在一个或者少量机器上。

大表格根据行边界被分成多个子表。一个子表保持行的一个连续范围。客户端经常能选择键来达到近邻性。系统的目标是每个子表的数据量达到 100 ~ 200MB。每个服务器负责大约 100 个子表。这能实现更快的恢复时间, 因为 100 台机器每台都从故障机器中选出一个子表。这

377

也导致了细粒度负载均衡，即从过载机器中移出子表。和 GFS 的设计类似，BigTable 也有一个主服务器来决定负载均衡。

图 6-20b 描述了 BigTable 系统结构。BigTable 主服务器管理和存储 BigTable 系统的元数据。BigTable 客户端使用 BigTable 客户端程序库来和 BigTable 主服务器以及子表服务器进行通信。BigTable 依靠一个高可用的、持续的分布式锁服务，称为 Chubby^[76]，将在 6.3.4 节讨论。■

子表位置分层

图 6-21 描述了如何对从 Chubby 中存储的文件开始的 BigTable 数据进行定位。第一层是一个存储在 Chubby 上的文件，它包括根子表的位置。根子表在一个特殊元数据（METADATA）表中包含所有子表的位置。每个 METADATA 子表包含一组用户子表的位置。根子表就是 METADATA 表的第一个子表，它的处理比较特殊；它绝不会分裂，以确保子表位置分层不会多于三层。

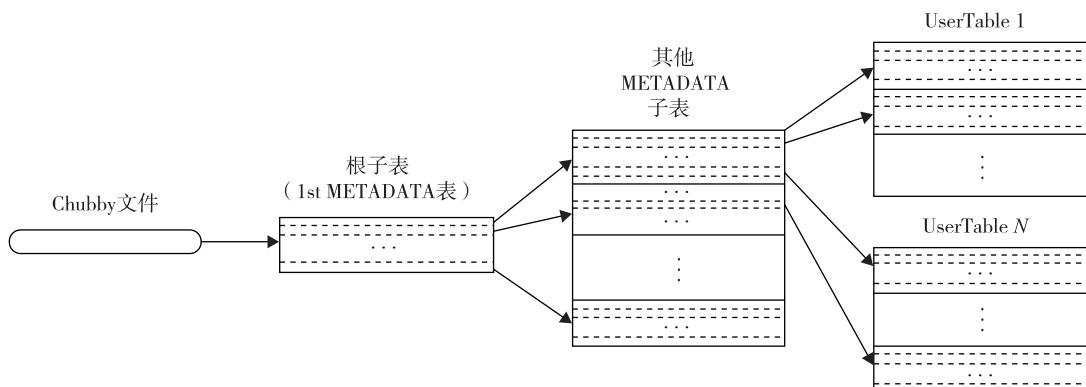


图 6-21 使用 BigTable 的子表位置分层

[378]

METADATA 表存储一个行键值下一个子表的位置，行键值是一个子表的表格标识符和其结束行的编码。BigTable 包含很多优化和容错特性。Chubby 能为寻找根子表确保文件的可用性。BigTable 主机能迅速扫描子表服务器，来确定所有节点的状态。子表服务器使用压缩来有效地存储数据。共享日志用来记录多个子表的操作，为的是减少日志空间以及保持系统的一致性。

6.3.4 Chubby——谷歌的分布式锁服务

Chubby^[76]用来提供粗粒度锁服务。它能在 Chubby 存储中存储小文件，这里提供了一个简单命名空间作为文件系统树。和 GFS 中的大规模文件相比，存储在 Chubby 上的文件是非常小的。基于 Paxos 一致协议，尽管任何成员节点都会出现故障，Chubby 系统仍然能够非常可靠。图 6-22 描述了 Chubby 系统的整体体系结构。

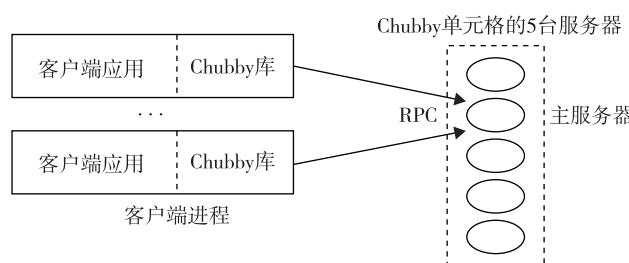


图 6-22 用于分布式锁服务的谷歌 Chubby 结构

每个 Chubby 单元内部都有 5 台服务器。在单元中每台服务器都有相同的文件系统命名空间。客户端使用 Chubby 库来和单元中的服务器进行对话。客户端应用程序能够在 Chubby 单元中的任何服务器上执行各种文件操作。服务器运行 Paxos 协议来保持整个文件系统的可靠性和一致性。

Chubby 已经成为谷歌的主要内部命名服务。GFS 和 BigTable 使用 Chubby 来从冗余副本中选择一个最主要版本。

6.4 亚马逊 AWS 与微软 Azure 中的编程

在这一节中，我们将会考虑 AWS 平台的编程支持。首先，我们回顾 AWS 平台及其提供的更新服务。然后，通过编程实例来研究 EC2、S3 和 SimpleDB 服务。回到图 4-22 和图 4-23 所示的编程环境的特点，亚马逊（和 Azure 一样）通过 6.1.3 节介绍的消息传递接口为一个关系数据库服务（Relational Database Service，RDS）。弹性 MapReduce 的功能与运行在 EC2 基础上的 Hadoop 相当。亚马逊在 SimpleDB 中支持 NOSQL。然而亚马逊并没有直接支持 6.3.3 节描述的 BigTable。

现在我们将关注表 4-6 列出的几个更多的功能。亚马逊提供了简单队列服务（SQS）和简单通知服务（SNS），它们是在 5.2 节和 5.4.5 节中讨论的服务的云实现。注意，代理系统在云中运行非常有效，它提供了一种引人注目的模型来控制传感器以及给数量不断增长的智能手机和平板电脑后台办公支持^[77]。我们进一步注意到自动伸缩和弹性负载均衡服务支持的相关功能。自动伸缩能够根据用户定义自动调节亚马逊 EC2 平台的容量大小。通过自动伸缩，用户可以确保使用的亚马逊 EC2 实例数量，在需求高峰时无缝地扩大规模以保持性能，在需求低谷时自动缩减规模以降低成本。

弹性负载均衡自动分配进来的应用流量到多个亚马逊 EC2 实例，这样可以避免闲置节点，同时在工作镜像上均衡化负载。CloudWatch 用来监控运行中的实例，通过 CloudWatch 可以实现自动伸缩和弹性负载均衡。CloudWatch 是一个提供 AWS 云资源监控的 Web 服务，与亚马逊 EC2 一起启动。它帮助用户直观地了解资源利用情况、操作性能和总体需求模式——其中包括 CPU 利用率、磁盘读/写和网络流量的度量。

6.4.1 亚马逊 EC2 上的编程

亚马逊是第一家引入应用托管虚拟机的公司。用户可以租借虚拟机而不是物理机器来运行他们的应用程序。通过使用虚拟机，用户可以自己选择加载任意软件。这类服务的弹性特点是用户可以根据需要创建、启动和终止服务器实例，并且对活动服务器按小时支付费用。亚马逊提供几种类型的预装虚拟机。实例通常称为亚马逊机器镜像（Amazon Machine Image，AMI）。这些虚拟机预先配置了 Linux 或者 Windows 的操作系统和一些附加软件。

表 6-12 定义了 3 种类型的 AMI。图 6-23 显示了运行环境。AMI 是运行虚拟机的实例模板。建立一个虚拟机的工作流是

创建一个 AMI → 创建密钥对 → 配置防火墙 → 启动 (6-3)

表 6-12 三类 AMI

映像类型	AMI 定义
私有 AMI	镜像由用户创建，默认为私有类型。可以授权给其他用户来启动你的私有镜像
公共 AMI	镜像由用户创建并发布到 AWS 社区，任何人都能以他们喜欢的方式启动实例并使用。亚马逊网站在以下网址列出了所有公共镜像： http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171
付费 QAMI	你可以创建具有特定功能的镜像。任何人只要在亚马逊收费之上按照使用的小时数支付给你费用即可使用

如图 6-23 所示，这个序列被公共、私有和付费 AMI 所支持。AMI 由图 6-23 底部所示的虚拟化计算、存储器和服务器资源构成。

379

380

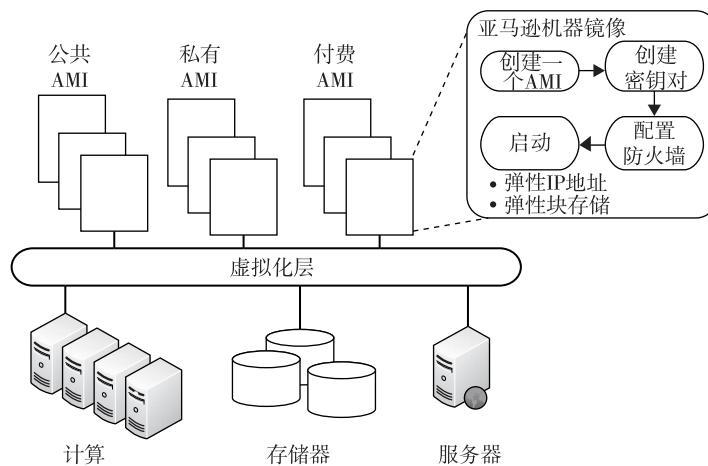


图 6-23 亚马逊 EC2 运行环境

例 6.9 在 AWS 平台使用 EC2 服务

表 6-13 列出了 2010 年 10 月在 5 个大类中可用的 IaaS 实例：

1. **标准实例** 适合大多数应用。
2. **微实例** 提供少量一致的 CPU 资源，在额外的周期允许超过 CPU 容量。适合较低吞吐量的应用程序或者定期进行大量计算的网站。
3. **大内存实例** 提供大内存容量用于高吞吐量应用，包括数据库和内存高速缓存的应用。
4. **高 CPU 实例** 提供按照比例比内存 (RAM) 更多的 CPU 资源，适用于计算密集型应用。
5. **集群计算实例** 提供网络性能增强的高 CPU 资源，适合于高性能计算应用和其他需要网络绑定的应用。它们使用 10GB 以太网互连。

表 6-13 亚马逊 EC2 实例类型 (2010 年 10 月 6 日)

计算实例	内存 (GB)	ECU 或 EC2 单元	虚 拟 核	存储 (GB)	32/64 位
标准: 小	1.7	1	1	160	32
标准: 大	7.5	4	2	850	64
标准: 特大	15	8	4	1690	64
微型	0.613	多达 2	无	只有 EBS	32 或 64
大内存	17.1	6.5	2	420	64
大内存: 双倍	34.2	13	4	850	64
大内存: 4 倍	68.4	26	8	1690	64
高 CPU: 中等	1.7	5	2	350	32
高 CPU: 特大	7	20	8	1690	64
集群计算	23	33.5	8	1690	64

第三列中的成本是根据 EC2 计算单元 (EC2 Compute Units, ECU) 计算，一个 ECU 提供相当于一个 1.0 ~ 1.2 GHz 的 2007 Opteron 或 2007 Xeon 处理器的 CPU 计算能力。表 6-14 是 CPU 每小时的使用成本。注意，在实际使用 EC2 时要支付多种资源的费用，而表 6-14 的 CPU 费用只是其中一项，其他所有费用（通常自然收费，所以读者应当在线获得最新的数据）可以在 AWS 网站中查看。 ■

6.4.2 亚马逊简单存储服务 (S3)

亚马逊 S3 提供一个简单 Web 服务接口，利用该接口可以在任意时间、任意地点通过 Web 存储和检索任意数据。S3 为用户提供面向对象的存储服务。用户可以通过带有支持 SOAP 的浏览器或者其他客户端程序的 SOAP 来访问他们的对象。SQS 用来确保两个进程间的可靠消息服务，

即使接收进程没有运行。图 6-24 为 S3 的运行环境。

表 6-14 亚马逊按需虚拟机实例类型的成本 (2010 年 10 月 6 日)

虚拟机实例类型	规 模	使用 Linux/UNIX (美元/时)	使用 Windows (美元/时)
标准实例	小 (默认)	0.085	0.12
	大	0.34	0.48
	特大	0.68	0.96
微实例	微型	0.02	0.03
大内存实例	特大	0.50	0.62
	双倍特大	1.00	1.24
	4 倍特大	2.00	2.48
集群计算实例	4 倍特大	1.60	暂无

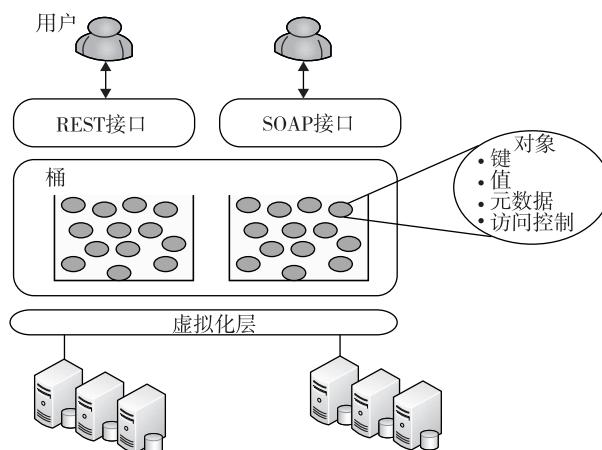


图 6-24 亚马逊 S3 运行环境

对象 (object) 是 S3 的基本操作单元。每个对象被存储在桶 (bucket) 里，通过唯一的开发者分配的键值 (key) 来被检索。也就是说，桶是对象的集装箱。除了唯一的键值属性以外，对象还有数值、元数据和访问控制信息等其他属性。从程序员的角度来看，S3 的存储可以被看做一个非常粗粒度的键 - 值对存储。通过键 - 值编程接口，用户可以读、写和删除对象，每个对象的大小可以从 1B 到 5GB。用户可以通过两类 Web 服务接口访问亚马逊云存储的数据。一个是 REST (Web 2.0) 接口，另一个是 SOAP 接口。S3 的一些关键特征如下：

- 通过地理分散冗余。
- 利用更便宜的减少冗余存储 (Reduced Redundancy Storage, RRS)，设计提供了一个在给定一年内 99.99999999% 的耐用性和 99.99% 可用性的对象。
- 认证机制用于确保数据不被非法访问。对象可以设置为私有或者公共，也可以授权给指定用户。
- 每个对象有 URL 和 ACL (访问控制列表)。
- 默认下载协议为 HTTP。BitTorrent 协议接口用来降低大规模分发的费用。
- 每月存储费用 (取决于存储总量) 是 0.055 美元/GB (超过 5000 TB) 到 0.15 美元/GB。
- 每个月最初 1GB 的输入或者输出流量是免费的，然后发往 S3 区域之外的流量价格为每 0.08 ~ 0.15 美元/GB。
- 在相同区域的亚马逊 EC2 和 S3 之间或者弗吉尼亚北部地区亚马逊 EC2 和美国标准地区亚马逊 S3 之间传输数据是不收取费用的 (2010 年 10 月 6 日)。

6.4.3 亚马逊弹性数据块存储服务（EBS）和 SimpleDB

弹性块存储（Elastic Block Store，EBS）提供卷块接口用于存储和恢复 EC2 实例的虚拟镜像。传统的 EC2 实例在使用后被销毁。现在，在机器关闭后，EC2 的状态仍被保存在 EBS 系统中。用户可以使用 EBS 保存永久性数据和安装到 EC2 的运行实例。注意，S3 是带消息传递接口的“存储即服务”。EBS 类似于传统的操作系统磁盘访问机制的分布式文件系统。EBS 允许用户创建大小为 1GB 到 1TB 的存储卷，可以安装为 EC2 实例。

多个卷可以被安装在同一个实例中。这些存储卷就像原始的未格式化的块设备，带有用户提供的设备名和块设备接口。在亚马逊 EBS 卷上，用户可以创建一个文件系统，也可以按使用 [383] 块设备（像硬驱动一样）的其他任意方式来使用存储卷。快照用来增量地保存数据，利用快照可以提高数据存储和恢复的性能。关于价格，亚马逊提供了类似 EC2 和 S3 的按需支付模式。存储卷的收费依据是用户分配的存储量，直到释放为止，价格为每月 0.10 美元/GB。EBS 还对存储器每 100 万个 I/O 请求收取 0.10 美元的费用（2010 年 10 月 6 日）。和 EBS 相似的服务也已经在开源云计算系统中提供，例如 Nimbus。

亚马逊 SimpleDB 服务

SimpleDB 基于关系数据库数据模型提供了一个简单数据模型。用户的结构化数据被组织到域中，每个域可以看做是一个表，条目（item）是表中的列，表中的单元格存放相应行的具体属性（列名）的值。这和关系数据库很相似。不同的是，它可能分配多个值到表格中的一个单元。而在传统关系数据库中，为了保持数据的一致性，这是不允许的。

许多开发人员只是希望能够快速地存储、访问和查询存储的数据，因此 SimpleDB 放弃了维持强一致性数据库模式的需求。SimpleDB 的定价是每台 SimpleDB 机器每个月最初 25 小时免费，超出部分每机每月 0.140 美元（2010 年 10 月 6 日）。和 Azure Table 一样，SimpleDB 可以被称为“LittleTable”，因为它们旨在管理存储在分布式表格中的少量信息。我们可以认为 BigTable 旨在管理基本的大数据，而 LittleTable 旨在管理元数据。亚马逊 Dynamo^[78]是沿着 SimpleDB 生产线下来的早期研究系统。

6.4.4 微软 Azure 编程支持

4.4.4 节介绍了 Azure 云系统。本节将更详细地描述这一编程模式。主要编程组件有客户端开发环境、SQLAzure，以及海量存储和编程子系统，如图 6-25 所示。我们集中在开发 Azure 程序的重要特点。首先，我们拥有底层 Azure 框架，它包括虚拟化的硬件和复杂的控制环境，可以实现资源的动态分配和容错。这实现了域名系统（DNS）和监控功能。自动服务管理允许用 XML 模板定义服务类型，还可以将多个服务复制按请求实例化。

当系统运行时，服务处于监控状态，人们可以访问事件日志、跟踪/调试数据、性能计数器、IIS Web 服务日志、崩溃转储和其他日志文件。这些信息可以保存在 Azure 存储器中。注意，运行云端应用不能进行调试，但是调试可以由跟踪完成。Azure 的基本特点可以分为存储和计算能力。Azure 应用程序通过定制的计算虚拟机连接到互联网，这个虚拟机称为 Web 角色，它支持基本的微软 Web 托管。这样配置的虚拟机一般称为工具机（appliance）。另一个重要计算类是服务器角色，它反映了在云计算中当需要时进行调度的计算资源池的重要性。服务器角色支持 HTTP(S) 和 TCP 协议。角色提供以下方法函数：

- OnStart() 方法在零件组成结构启动时调用，允许初始化任务。它报告忙碌状态到负载均衡器，直到返回 true。
- OnStop() 方法当角色要被关闭时调用，执行平稳退出。
- Run() 方法包含主要逻辑。

正如第4章所讨论的，Azure中角色的概念是一个有趣的想法，我们可以期待扩展角色类型并在其他云计算环境中使用。图6-25展示计算角色可以达到负载均衡，这点与GAE和AWS云的处理相似（见4.1节）。

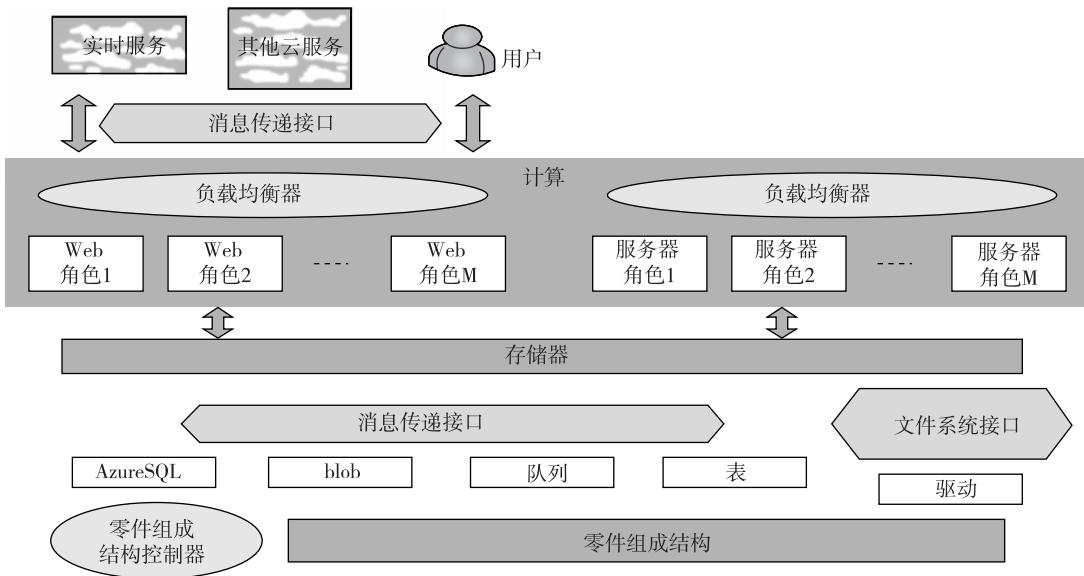


图6-25 Azure云计算平台特征

6.4.4.1 SQLAzure

如图6-25所示，Azure提供一系列非常丰富的存储功能。SQLAzure提供SQL服务器作为服务，将在例6.10中详细描述。除了最新引进的驱动外，其他所有存储形式都用REST接口访问，这点与6.4.3节中讨论的亚马逊EBS相似，另外提供一个文件系统接口作为支持blob存储的持久NTFS卷。REST接口自动关联URL，同时为了提高容错性能，保证访问的一致性，所有存储将被复制三次。

类似于亚马逊中的S3，Azure的基本存储系统建立在blob之上。blob数据模型分为三个层次：账户→集装箱→页面或块blob。集装箱类似传统文件系统的目录，账户是根目录。块blob用于流数据，每个blob由若干个有序的块组成，每个块最大为4MB，ID长度为64字节。块blob容量上限为200GB。页blob用于随机读/写访问，包含一系列页，最大为1TB。blob中的数据与相应的元数据建立关系，元数据以<name, value>形式表示，每个blob最多有8KB的元数据说明。
385

6.4.4.2 Azure表

Azure表和队列存储模式目标是非常小的数据卷。队列提供可靠的消息传递，很自然地用于支持Web和服务器角色之间的工作分配。队列可以包括数目不限的消息，队列中的消息至少可以被检索和处理一次，每个消息长度最大为8KB。Azure支持PUT、GET和DELETE消息操作以及CREATE和DELETE队列操作。每个账户可以有任意数量的Azure表，Azure表中的行称为实体，列称为属性。

表格中的实体数量没有限制，这个技术被设计为可以很好地扩展到分布式计算机中存储的大量实体。一个实体最多可以包括255个通用属性，它们是<name, type, value>三元组。每个实体必须定义另外的两个属性PartitionKey和RowKey，但在其他方面，属性的命名没有限制——这个表是非常灵活的！RowKey用来唯一标识每个实体，而PartitionKey被设计成共享，具有相同PartitionKey的实体存储在相邻的位置。用好PartitionKey可以加速搜索性能。一个实体可以至多有1MB的存储容量；如果你需要更大的容量，只需要在表格的属性值中存储一个到blob存储的

链接。ADO.NET 和 LINQ 支持表格查询。

例 6.10 SQLAzure 数据服务

Azure 提供了一个复杂的数据库编程接口。(更多细节参见 www.microsoft.com/azure/sql.mspx)。SQLAzure 数据服务可以近似看做传统的关系数据库。建立在目前成熟的商业软件包上，SQLAzure 可以被视为是一个高度可扩展的、按需数据存储和查询处理的效用服务。SQLAzure 的服务接口基于标准 Web 协议，并且 SQLAzure 支持 SOAP 和 REST。由于基于关系数据库，SQL 数据服务 (SQL Data Services, SDS) 比之前讨论的两个 NOSQL 数据管理服务 (谷歌的 BigTable 和亚马逊的 SimpleDB) 能够提供更丰富的数据模型。

SQLAzure 中的数据模型包括三个灵活的模式：授权机构 (authority)、集装箱 (container) 和实体。用户在注册完数据服务后，可以创建一个授权机构，表示为一个 DNS 名称，例如 mydomain.data.database.windows.net，其中 mydomain 是用户创建的授权机构，data.database.windows.net 指的是相应的服务。用户可以在任意时间创建多个授权机构。这个 NDS 名将解析到一个特定的 IP 地址并映射到特定的数据中心。因此一个授权机构和它的数据存储在同一个数据中心。在最高级授权机构之下的是集装箱，一个授权机构包含若干个集装箱 (或者没有)。集装箱可以使用其 ID 作为句柄，在授权机构中查找相应的集装箱。

集装箱是用户存放数据的地方。就像 SimpleDB 一样，用户可以将数据存储在集装箱，而无需考虑数据模式。实体是存储在集装箱中的单元。一个实体可以存放任意数量用户自定义的属性和相应值 (例如，像 SimpleDB 中的属性和值)。这里有两个不同类型的集装箱：同构集装箱和异构集装箱。像关系数据库表格一样，一个同构集装箱的所有实体具有相同的类型，而异构集装箱则没有这一限制。这些概念在图 6-25 中可以看到。

SDS 是 Azure 平台提供云端应用程序的构建模块之一。它的确提供了企业级数据平台。微软在全世界范围内建设了多个数据中心，用于托管第三方云端应用。多个数据中心使得数据具有高可用性和安全性，用户不用担心数据的丢失。SQLAzure 的另一个重要特征是易于开发。SDS [386] (实际上是整个 Azure 平台 SDK) 可以与微软强大的 Visual Studio 开发环境整合在一起，这样能大大提高开发人员开发云端应用的效率和有效性。 ■

6.5 新兴云软件环境

本节我们对流行的云操作系统和新兴云软件环境进行评估。我们覆盖了开源的 Eucalyptus 和 Nimbus，然后检测了 OpenNebula、Sector/Sphere 和 Open Stack。第 3 章从虚拟化的角度介绍了这些环境。现在，我们提供了更多关于编程需求的细节。我们还将涉及最近在墨尔本大学开发出的 Aneka 云编程工具。

6.5.1 开源的 Eucalyptus 和 Nimbus

Eucalyptus 是从加州大学圣巴巴拉分校一个研究项目开发出的 Eucalyptus Systems (www.eucalyptus.com) 发展而来的一个产品。Eucalyptus 最初旨在将云计算范式引入到学术上的超级计算机和集群。Eucalyptus 提供了一个 AWS 兼容的基于 EC2 的 Web 服务接口，用来和云服务交互。另外，Eucalyptus 也提供服务，如 AWS 兼容的 Walrus，以及一个用来管理用户和镜像的用户接口。

6.5.1.1 Eucalyptus 体系结构

Eucalyptus 系统是一个开放的软件环境。Eucalyptus 白皮书^[79,80] 中介绍了这个体系结构。在图 3-27 中，我们已经从虚拟集群化的角度介绍了 Eucalyptus。图 6-26 从管理虚拟机镜像的要求上给出了体系结构。如下所示该系统在虚拟机镜像管理中支持云程序员。实际上，该系统已经延伸到支持计算云和存储云的开发。

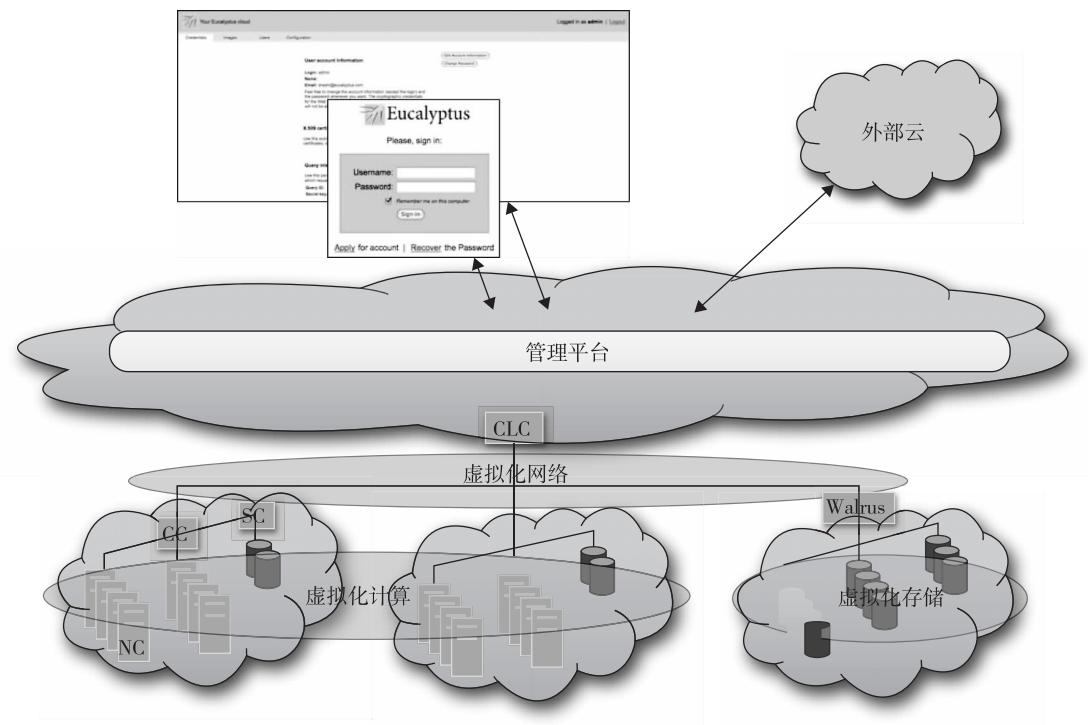


图 6-26 用于虚拟机镜像管理的 Eucalyptus 体系结构

注：由 Eucalyptus LLC 协议提供^[81]。

6.5.1.2 虚拟机镜像管理

Eucalyptus 吸收了很多亚马逊 EC2 的设计成果，且二者镜像管理系统没有什么不同。Eucalyptus 在 Walrus 中存储镜像，其块存储系统类似于亚马逊 S3 服务。这样，任何用户可以自己捆绑自己的根文件系统，上传然后注册镜像并把它和一个特定的内核与虚拟硬盘镜像连接起来。这个镜像被上传到 Walrus 内由用户自定义的桶中，并且可以在任何时间从任何可用区域中被检索。这样就允许用户创建专门的虚拟工具 (http://en.wikipedia.org/wiki/Virtual_appliance) 并且不费力地用 Eucalyptus 来配置它们。Eucalyptus 系统提供了一个商业版权版本，以及我们刚刚描述的开源版本。

6.5.1.3 Nimbus

Nimbus^[81,82]是一套开源工具，一起提供一个 IaaS 云计算解决方案。图 6-27 给出了 Nimbus 的体系结构，它允许客户租赁远程资源，通过在资源上部署虚拟机和配置它们表示用户期望的环境。为了这个目的，Nimbus 提供了一个被称为 Nimbus Web 的特殊 Web 界面^[83]。其目的是以一个友好的界面提供管理和用户功能。Nimbus Web 以一个 Python Django^[84] Web 应用为中心，其目的是部署时可以从 Nimbus 服务中完全分离出来。

正如我们在图 6-27 中所看到的，一个称为 Cumulus^[83]的存储云实现已经与其他中心服务紧密集成起来，尽管其可以单独使用。Cumulus 和亚马逊的 S3 REST API^[85]兼容，并通过包含诸如配额管理这些特征扩展了其功能。因而，那些不能和 S3 REST API 同时运行的客户端（如 boto^[86] 和 s2cmd^[87]）能够和 Cumulus 一起运行。另外，Nimbus 云客户端使用 Java Jets3t 库^[88]与 Cumulus 进行交互。

Nimbus 支持两种资源管理策略。第一种是默认的“资源池”模式。在这种模式中，服务直接控制虚拟机管理器节点池，且假设其能启动虚拟机。另一个支持模式被称为“飞行模式”。在

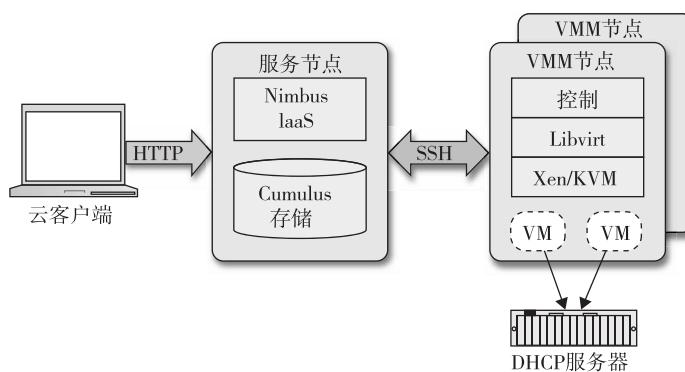


图 6-27 Nimbus 云基础设施

注：由 Nimbus 项目提供^[82]。

这里，服务向集群的本地资源管理系统（Local Resource Management System，LRMS）发出请求，以获得一个可用的虚拟机管理器来配置虚拟机。Nimbus 也提供亚马逊 EC2 接口^[89]的实现，允许用户使用为真实 EC2 系统开发的客户端，而不是基于 Nimbus 的云。

6.5.2 OpenNebula、Sector/Sphere 和 Open Stack

Open Nebula^[90,91]是一个开源的工具包，它可以把现有的基础设施转换成像类似云界面的 IaaS 云。图 6-28 显示了 OpenNebula 体系结构及其主要组件。OpenNebula^[92]的体系结构已经被设计得非常灵活且模块化，允许与不同的存储和网络基础设施配置以及 hypervisor 技术集成起来。这里，核心是一个集中式组件，它管理着虚拟机的全部生命周期，包括动态设置虚拟机群的网络，管理它们的存储需求，如虚拟机磁盘镜像的部署或者即时软件环境的生成。

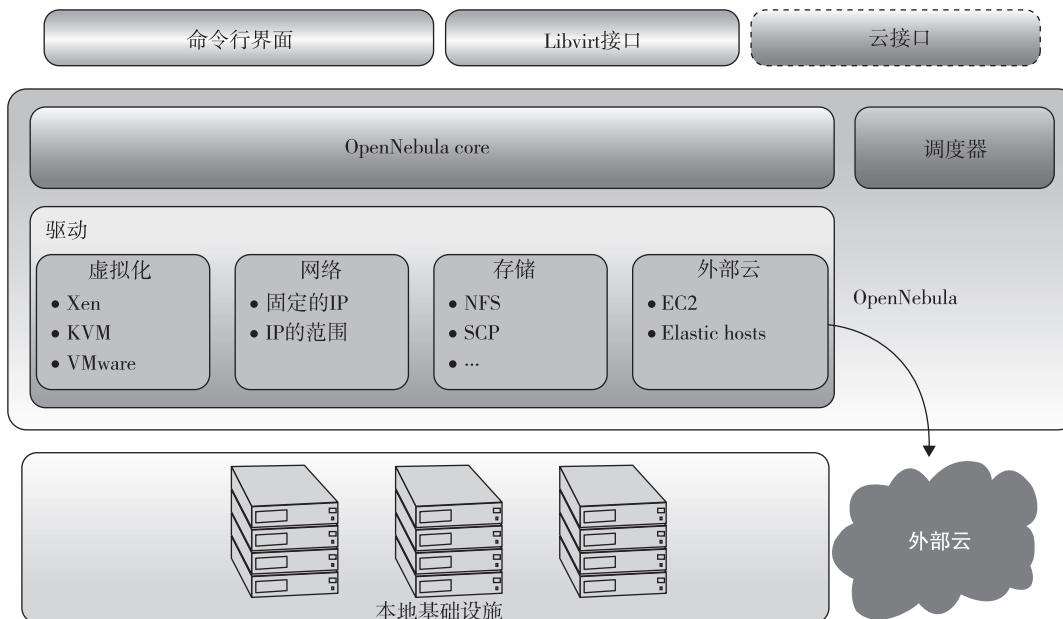


图 6-28 OpenNebula 体系结构及其主要组件

注：由 Sotomayor 提供^[94]。

另外一个重要的组件是容量管理器或调度器。它管理核心提供的功能。默认的容量调度器是一个需求/rank 的匹配者。然而，很有可能通过租赁模型和提前预留^[93]发展出更为复杂的调度

策略。最后的主要组件是访问驱动器。它们提供一个底层基础设施的抽象来显示在集群中可用的监测、存储、虚拟化服务的基本功能。因此，OpenNebula 没有绑定到任何特定环境，并且能提供一个与虚拟平台无关的统一管理层。

此外，OpenNebula 提供管理界面来整合其他数据中心管理工具的核心功能，如审计或监测框架。为此，OpenNebula 实现了 libvirt API^[94]，它是一个虚拟机管理的开放接口，也是一个命令行界面（Command Line Interface，CLI）。这些功能的一部分通过一个云接口显示给外部用户。OpenNebula 能够适应组织资源需求的变化，包括物理资源的增加或失效^[95]。一些支持变化环境的基本特性有实时迁移和虚拟机快照^[90]。

进一步，当本地资源不足时，OpenNebula 能够通过使用和外部云接口的云驱动器支持混合云模型。这使得组织能用公有云的计算容量来补充本地设施以满足峰值需求或者实现高可用策略。OpenNebula 目前包括一个 EC2 驱动器，它能向亚马逊 EC2^[89] 和 Eucalyptus^[80]，以及 ElasticHosts 驱动器^[96] 提交请求。关于存储，镜像库允许用户很容易地从一个目录中指定磁盘镜像，而不用担心低级磁盘配置属性或者块设备映射。同时，镜像访问控制被应用到仓库中注册过的镜像，于是简化了多用户环境和镜像共享。不过，用户也可以建立他们自己的镜像。

6.5.2.1 Sector/Sphere

Sector/Sphere 是一个软件平台，能够在一个数据中心内部或在多个数据中心之间，支持大量商业计算机集群上的大规模分布式数据存储和简化的分布式数据处理。该系统由 Sector 分布式文件系统和 Sphere 并行数据处理框架^[97,98] 构成。Sector 是一个分布式文件系统（DFS），它能部署在很大范围里且允许用户通过高速网络连接^[99] 从任何位置管理大量的数据集。通过在文件系统中复制数据和管理副本实现容错。

由于当放置副本时，Sector 知道网络拓扑结构，它也能提供更好的可靠性、可用性和访问吞吐量。通信执行是通过用户数据报协议（User Datagram Protocol，UDP）进行消息传递，通过用户定义类型（User Defined Type，UDT）^[100] 进行数据传输。显然，由于不需要建立连接，对于消息传递，UDP 比 TCP 来得更快，但是如果 Sector 用在互联网上，这也很可能成为一个问题。与此同时，UDT 是一个可靠的基于 UDP 的应用级数据传输协议，它被专门设计来在大范围高速网络上高速传输数据^[100]。最后，Sector 客户端提供编程 API、工具和 FUSE^[101] 用户空间文件系统模型。

另外，Sphere 是一个与 Sector 管理的数据一起工作的并行数据处理引擎。这种联合允许该系统对作业调度和数据位置做出精确的决策。Sphere 提供了一个编程框架，开发人员可以使用该框架去处理存储在 Sector 中的数据。因此，它允许 UDF 并行地运行在所有输入数据段上。一旦有可能（数据位置），这些数据段就在它们的存储位置被处理。故障数据段可以在其他节点重新启动，以达到容错要求。在 Sphere 应用程序中，输入和输出都是 Sector 文件。通过 Sector 文件系统的输入/输出交换/共享，多个 Sphere 处理段可被联合起来去处理更为复杂的应用^[102]。

Sector/Sphere 平台^[102] 被如图 6-29 所示的体系结构所支持，它包括 4 个组件。第一个组件是安全服务器，它负责认证主服务器、从节点和用户。我们也有可以认作基础设施核心的主服务器。主服务器维护文件系统元数据、调度作业并响应用户的请求。Sector 支持多个活跃的主服务器，它们可以在运行时加入或离开以及管理请求。另一个组件是从节点，在这里数据被存储和处理。从节点可以位于单个的数据中心内部，或者通过高速网络相连的多个数据中心。最后一个组件是客户端组件。这个组件为用户提供访问和处理 Sector 数据的工具和编程 API。

最后，需要指出的是作为这个平台的一部分，一个新的组件已经被开发出来。它被称为 Space^[97]，包含一个支持基于列的分布式数据表的框架。因此，表被按列存储且被分割到多个从节点中。表是独立的且它们之间不支持关系。支持一个简化的 SQL 操作集，包括但不限于表的

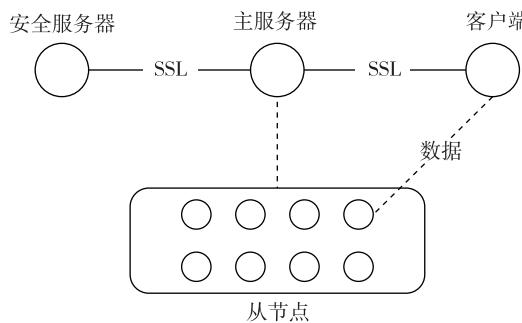


图 6-29 Sector/Sphere 系统体系结构

注：由 GU 和 Grossman 提供^[102]。

建立和修改、键值对的更新和查找，以及选择 UDF 操作。

6.5.2.2 OpenStack

在 2010 年 7 月，OpenStack^[103]已经被 Rackspace 和 NASA 提出。该项目是试图建立一个开源的社区，跨越技术人员、开发人员、研究人员和工业来分享资源和技术，其目标是创建一个大规模可伸缩的、安全的云基础设施。按照其他开源项目的惯例，整个软件是开源的且仅限开源 API，如亚马逊。[391]

目前，OpenStack 使用 OpenStack Compute 和 OpenStack Storage 的解决方案，重点开发两个方面的云计算来解决计算和存储问题。“OpenStack 计算是创建和管理大规模团体虚拟专用服务器的云内部结构”和“OpenStack Object Storage 是一个软件，使用商用服务器集群去存储太字节甚至拍字节数据来创建冗余可伸缩的对象存储”。最近，已经原型化一个镜像库。镜像库包含一个镜像注册和发现服务以及一个镜像发送服务。它们一起向计算服务发送镜像，并从存储服务获得镜像。这个开发表明该项目正在努力向产品组合里集成更多的服务。

6.5.2.3 OpenStack Compute

作为计算支持努力的一部分，OpenStack^[103]正在研发一个称为 Nova 的云计算结构控制器，它是 IaaS 系统的一个组件。Nova 的体系结构是建立在零共享和基于消息传递的信息交换的概念上。所以 Nova 中的大多数通信是由消息队列所推动。为了防止在等待其他响应时阻塞组件，引入了延迟对象。这样的对象包括回调函数，该函数在收到一个响应时会被触发。这和并行计算中已经建立起来的概念非常相似，例如“futures”，这已经在网格社区中的一些项目（如 CoG Kit）中应用。

为实现零共享范式，整个系统的状态保存在一个分布式数据系统中。通过原子事务使得状态更新保持一致性。Nova 用 Python 语言来实现，同时利用大量的外部支持函数库和组件。这包括 boto、Python 编写的亚马逊 API，以及 Tornado、一个在 OpenStack 中用来实现 S3 功能的快速 HTTP 服务器。图 6-30 显示了 OpenStack Compute 的主要体系结构。在这个体系结构中，API 服务器从 boto 接收 HTTP 请求，把命令转成或转自 API 格式，并向云控制器转发请求。

云控制器维护系统的全局状态，确保通过轻量级目录访问协议（Lightweight Directory Access Protocol，LDAP）与用户管理器交互时的授权，同 S3 服务和管理节点相互作用，还通过一个队列与存储工作机作用。此外，Nova 集成网络组件来管理私有网络、公有 IP 寻址、虚拟专用网（Virtual Private Network，VPN）连接，以及防火墙规则。它包括以下类型：

- *NetworkController* 管理地址和虚拟局域网（Virtual LAN，VLAN）分配。
- *RoutingNode* 管理公有 IP 到私有 IP 的 NAT（Network Address Translator，网络地址翻译器）转换，强制执行防火墙规则。

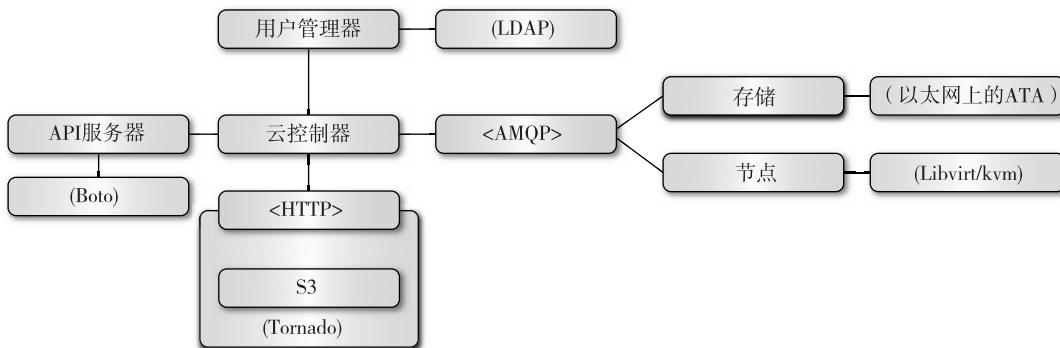


图 6-30 OpenStack Nova 系统体系结构。5.2 节描述了 AMQP（高级消息队列协议）

- *AddressingNode* 为私有网络运行动态主机配置协议（Dynamic Host Configuration Protocol, DHCP）服务。
- *TunnelingNode* 提供 VPN 连接。

网络状态（在分布式对象存储中管理）包含以下内容：

- 分配给一个项目的 VLAN。
- 在一个 VLAN 中一个安全群体的私有子网分配。
- 运行实例的私有 IP 分配。
- 一个项目的公有 IP 分配。
- 一个私有 IP/运行实例的公有 IP 分配。

6.5.2.4 OpenStack Storage

OpenStack 存储方案是在很多相互作用的组件和概念上建立起来的，包括一个代理服务器、一个环、一个对象服务器、一个集装箱服务器、一个账户服务器、副本、更新者和审计者。代理服务器的作用是使查询账户、集装箱或者 OpenStack 储存环里的对象及路由请求成为可能。因此，任何对象是直接通过代理服务器在对象服务器和用户之间来回流动。一个环代表磁盘上存储的实体名字和其物理位置的映射。

存在账户、集装箱和对象的单独环。一个环包括使用区域、设备、分区和副本的概念。于是它允许系统能够处理故障，以及代表着一个驱动器、一台服务器、一个机架、一个交换机或者甚至一个数据中心区域的隔离。在集群里可以使用权重来平衡驱动器里每个分区的分配，以支持异构的存储资源。根据文档，“对象服务器是一个非常简单的块存储服务器，它能存储、检索和删除存储在本地设备中的对象”。

对象以二进制文件形式保存，元数据存储在文件的扩展属性里。这要求底层的文件系统围绕对象服务器构建，这常常和标准 Linux 安装不一样。为了列出对象，可以使用集装箱服务器。集装箱列表是由账户服务器负责的。OpenStack “Austin” Compute 和 Object Storage 的第一个版本是在 2010 年 10 月 22 日发行的。该系统有一个很强大的开发团队。

6.5.3 Manjrasoft Aneka 云和工具机

Aneka (www.manjrasoft.com/) 是一个由 Manjrasoft 公司开发的云计算应用平台，公司坐落于澳大利亚墨尔本。Aneka 是为私有云或公有云上并行和分布式应用的快速开发和部署而设计的。它提供了一系列丰富的 API，可以透明地利用分布式资源，采用喜欢的编程抽象来表示各种应用的商业逻辑。系统管理员可以利用一系列的工具来监视和控制部署好的基础设施。该平台可以部署在像亚马逊 EC2 这样的公有云上，其订阅者通过互联网来访问，也可以部署在访问受限的一系列节点组成的私有云上，如图 6-31 所示。

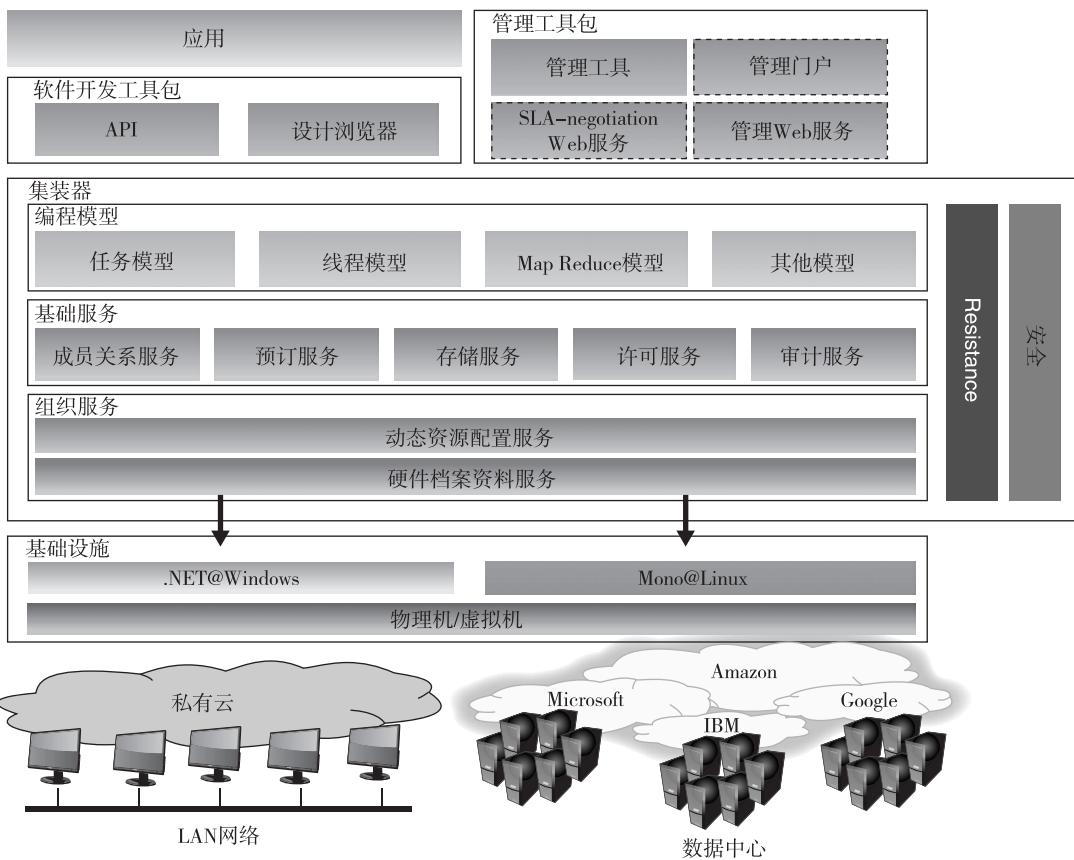


图 6-31 Aneka 体系结构和组件

注：由 Manjrasoft 公司 Raj Buyya 提供。

Aneka 作为一个工作负载的分配和管理平台，用来加速运行于 Linux 和 Microsoft .NET 框架环境下的应用。和其他负载分配解决方案相比，Aneka 有如下一些关键优势：

- 支持多种编程和应用环境。
- 同时支持多种运行时环境。
- 拥有快速部署工具和框架。
- 基于用户的 (QoS/SLA) 需求，能够利用多种虚拟机和物理机器来加速应用供应。
- 构建在 Microsoft .NET 框架之上，能够通过 Mono 支持 Linux 环境。

Aneka 提供了三种类型功能，它们是创建、加速和管理云计算及其应用所必需的：

1. **创建：**Aneka 包括一个新的 SDK，它把 SDK 和工具联合起来让用户能够迅速开发应用。Aneka 也允许用户创建不同的运行时环境，例如，通过利用网络或企业数据中心、亚马逊 EC2 的计算资源创建的企业/私有云，Aneka 管理的企业私有云和来自亚马逊 EC2 的资源组成的混合云，或使用 XenServer 创建和管理的其他企业云。

2. **加速：**Aneka 支持在多个运行时环境中不同的操作系统（比如 Windows 或 Linux/ UNIX 系统）下快速开发和部署应用程序。Aneka 尽可能地使用物理机器来达到本地环境的最大利用率。任何时候当用户设定 QoS 参数（如期限）时，或者如果企业资源不足以满足要求，Aneka 支持从公有云（如 EC2）动态租赁额外的能力，以便按时完成任务（见图 6-32）。

3. **管理：**Aneka 支持的管理工具和功能包括一个 GUI 和来设置、监控、管理和维护远程与全球 Aneka 计算云的 API。Aneka 也有一个审计机制和管理优先权以及基于 SLA/QoS 的可扩展性，

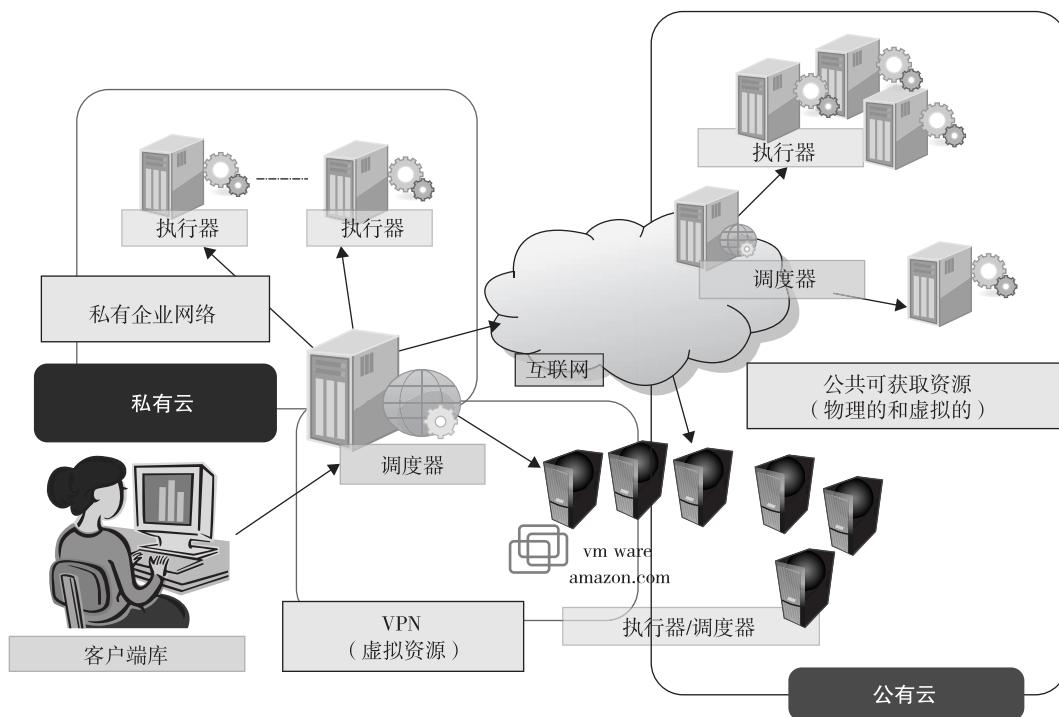


图 6-32 Aneka 使用私有云资源，并动态租赁公有云资源

注：由 Manjrasoft 公司提供，<http://www.manjrasoft.com/>。

它使动态供应成为可能。

下面是 Aneka 支持的三个重要编程模型，可以用于云计算和传统并行应用程序：

1. 线程编程模型：该模型是最好的解决方法，可用来利用计算机云中多核节点的计算功能。
2. 任务编程模型：该模型允许快速原型和实现一个独立的任务应用包。
3. MapReduce 编程模型：在 6.2.2 节中已讨论过。

6.5.3.1 Aneka 体系结构

作为一个云应用平台，Aneka 的特点是为应用程序提供同构的分布式运行时环境。这个环境通过将托管 Aneka 容器的物理节点和虚拟节点聚集在一起而建成。容器是一个轻量级层次，与主机环境进行交互，管理部署在一个节点上的服务。和主机平台的交互是由平台抽象层（Platform Abstraction Layer, PAL）调解，在它的实现中隐藏了不同的操作系统的所有异构性。

通过 PAL 使运行所有与基础设施相关的任务成为可能，如性能和系统监控。这些活动对于确保应用所需的服务质量是至关重要的。PAL 和容器一起，代表了服务的主机环境，实现了中间件的核心功能，组成一个动态和可扩展的系统。可用的服务可以归为三个主要类别：

组织服务：组织服务实现了云基础设施的基本操作。这些服务包括：高可用性和故障时提高可靠性、节点关系和目录、资源供应、性能监控和硬件档案资料。

基础服务：基础服务构成了 Aneka 中间件的核心功能。它们提供了一套基本功能，增强了在云里应用的执行能力。这些服务给基础设施提供了附加价值，并且对于系统管理员和开发者都是有用的。在这个类别里，我们可以列举出：存储管理、资源预留、报告、审计、计费、服务监控和许可制度。在所有支持的应用模型里都可以运行这个级别的服务。

应用服务：应用服务直接处理应用的执行并负责为每一个应用模型提供合适的运行时环境。他们为几个应用运行任务（如弹性可扩展、数据传输、性能监控、审计和计费）利用基础服务和

组织服务。在这个级别上，Aneka 在支持不同的应用模型和分布式编程模式上显示了其真实潜力。

依靠底层和服务来执行应用，每一个支持的应用模型由一种不同的服务集所管理。总的来说，每一个应用模型的中间件副本至少有两个不同的服务：调度与执行。此外，特定的模型需要额外服务或者一个不同类型的支持。Aneka 为最有名的应用编程模式提供支持，如分布式线程、任务包和 MapReduce。

在这个系统中可以设计并部署附加服务。基础设施就是这样添加了很多附加特点和功能而变得丰富。SDK 为快速的服务原型开发提供了直接的接口和易用的组件。新服务的部署和集成非常迅速、悄无声息。容器利用 Spring 框架，允许对类似服务这种新组件进行动态集成。

例 6.11 Maya 绘图实例的 Aneka 应用

Aneka 已在生命科学、工程和创意媒体等领域创造了多种有趣的应用。利用 Aneka 开发的各种应用能不加改变地运行在企业或公有云上。我们将简要介绍一个使用 Aneka 并软件在工程设计中进行高速渲染的实例研究。为了减少时间，GoFront 使用 Aneka 并在他们公司利用计算机网络建立了一个企业云（见图 6-33）。作为中国南方铁路的一个成员，GoFront 集团是中国领先的和最大的电动机车设备研究和制造商。

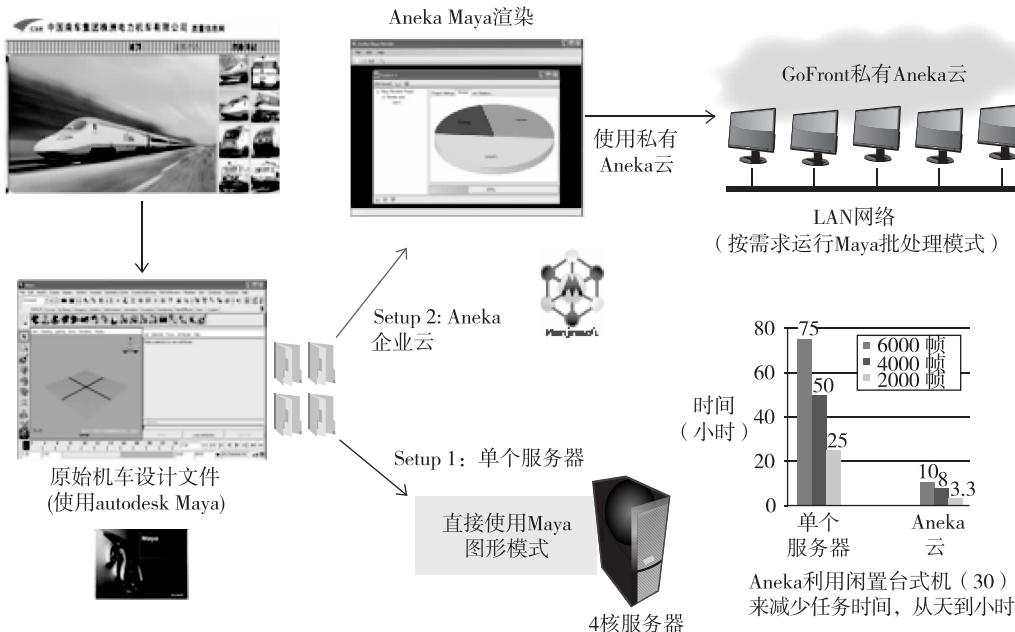


图 6-33 使用 Aneka 在 GoFront 私有云上绘制机车设计图像

注：由 Manjrasoft 公司的 Raj Buyya 提供。

GoFront 集团负责设计高速电动机车、地铁车辆、城市交通车辆和动车。最初的原型设计需要使用 Autodesk 的 Maya 绘图软件生成的高质量三维图像。通过检查三维图像，工程师找出了原始设计中的问题，并作出了合适的设计改进。然而，这样的设计要用 2 000 帧图像描绘场景，需要用一个 4 核的服务器花费 3 天时间。

为了节约时间，GoFront 使用了 Aneka 并在他们公司利用计算机网络建立了企业云。他们利用 Aneka Design Explorer，一款快速创建参数扫描应用的工具，这个工具在不同的数据项上将相同的程序执行了很多次（在这个例子中，执行 Maya 软件来描绘不同的图像）。一款定制的 Design Explorer（称为 Maya GUI）已经用于 Maya 绘图。Maya GUI 管理各种参数、产生 Aneka 任务、监视已提交的 Aneka 任务并收集最后已经画好的图像。设计图像过去需要 3 天来渲染（2 000 +

395
396

397

帧，每一帧有超过 5 种不同的摄影角度）。只使用 20 个节点的 Aneka 云，GoFront 就能够使渲染时间从开始的 3 天缩短为 3 个小时。 ■

6.5.3.2 虚拟设备

机器虚拟化提供了唯一的机会，突破了软件在应用程序和主机环境之间的依赖性。近年来，由于为商用系统开发的高效且可自由获得的虚拟机监视器（例如 Xen、VMware Player、KVM 和 VirtualBox），资源虚拟化经历了复兴，并且得到微处理器公司的支持（例如 Intel 和 AMD 的虚拟化扩展）。现代系统虚拟机提供了更好的灵活性、安全性、隔离和资源控制，同时支持大量未经修改的应用，在网格计算上的使用有着压倒性的优势。网格应用也需要网络的连通性，日益增加的 NAT 和 IP 防火墙技术打破了先前互联网中节点对等的早期模式，阻碍了网格计算系统的规划和部署。

在 Aneka 中，虚拟机和 P2P 网络虚拟化技术可以集成为一个可自我配置的、预包装的“虚拟设备”，以使在异构、广域分布式系统上同构配置的虚拟集群得到简单部署。虚拟设备是安装和配置好整个软件栈（包括操作系统、库、二进制文件、配置文件和自动配置脚本）的虚拟机镜像，这样当虚拟设备实例化时，给定的应用可以即开即用地工作。与传统的软件发行方式相比，虚拟设备的好处是大大减少了软件对主机环境的依赖。

对于大多数 VMM 存在虚拟机转化工具，正在进行标准化工作来进一步增强不同 VMM 之间的合作，使得多个平台之间的应用实例化是无缝的。虚拟应用的广域覆盖物汇集了商用硬件的功能，可以被作为局域网来编程和管理——即使节点分布在多个网域之上。预配置的网格设备镜像 (www.grid-appliance.org) 能以对用户透明的方式封装复杂的分布式系统软件，以便容易部署。对于现代的商用服务器和台式机有运行在免费 VMM 上的网格设备。

[398]

6.6 参考文献与习题

在本章中已经给出核心技术的详细文献，这里我们另外提供一些有用的参考文献，其中的大多数主题都很新，很少有综合性的整理，在这个快速发展的时代，主要的商业云的链接也经常发生变化；建议读者检查各大网站的最新链接。例如文献 [104 ~ 108]。文献 [109] 是 Chou 编写的一本很好的在商业和技术中介绍云计算的讲义。文献 [110 ~ 115] 介绍了各种云服务。文献 [110, 116 ~ 120] 讨论了云计算的好处和机会。文献 [28, 43, 73, 121 ~ 127] 中研究了 HPC 中的云技术。文献 [26, 28] 介绍云应用的分布式编程范式。文献 [25, 41 ~ 43, 74, 75] 讨论云对于数据密集型计算的应用。

致谢

本章的合著者是印第安纳大学的 Geoffrey Fox 和悉尼大学的 Albert Zomaya。此外，还得到了以下学者的帮助：陈康（清华大学，中国），Judy Qiu、Gregor von Laszewski、Javier Diaz、Archit Kulshrestha 和 Andrew Younge（印地安纳大学），Reza Moravaeji 和 Javid Teheri（悉尼大学），以及 Renato Figueiredo（佛罗里达大学）。Rajkumar Buyya（墨尔本大学）对 6.5.3 节做出了贡献。黄铠教授（USC）负责最后本章文稿的编辑。

[399]

本章是由东南大学的秦中元副教授翻译，并得到宋云燕、郑勇鑫与杨中云同学的协助。

参考文献

- [1] C. Dabrowski, Reliability in grid computing systems, *Concurr. Comput. Pract. Exper.* 21 (8) (2009) 927–959.
- [2] Microsoft, Project Trident: A Scientific Workflow Workbench, <http://research.microsoft.com/en-us/collaboration/tools/trident.aspx>, 2010.

- [3] W. Lu, J. Jackson, R. Barga, AzureBlast: A case study of developing science applications on the cloud, in: ScienceCloud: 1st Workshop on Scientific Cloud Computing co-located with HPDC (High Performance Distributed Computing), ACM, Chicago, IL, 21 June 2010.
- [4] Distributed Systems Laboratory (DSL) at University of Chicago Wiki. Performance Comparison: Remote Usage, NFS, S3-fuse, EBS. 2010.
- [5] D. Jensen, Blog entry on Compare Amazon S3 to EBS data read performance, <http://jensendarren.wordpress.com/2009/12/30/compare-amazon-s3-to-ebs-data-read-performance/>, 2009.
- [6] Zend PHP Company, The Simple Cloud API for Storage, Queues and Table, <http://www.simplecloud.org/home>, 2010.
- [7] Microsoft, Windows Azure Geo-location Live, <http://blogs.msdn.com/b/windowsazure/archive/2009/04/30/windows-azure-geo-location-live.aspx>, 2009.
- [8] Raytheon BBN, SHARD (Scalable, High-Performance, Robust and Distributed) Triple Store based on Hadoop. <http://www.cloudera.com/blog/2010/03/how-raytheon-researchers-are-using-hadoop-to-build-a-scalable-distributed-triple-store/>, 2010.
- [9] NOSQL Movement, Wikipedia list of resources, <http://en.wikipedia.org/wiki/NoSQL>, 2010.
- [10] NOSQL Link Archive, LIST OF NOSQL DATABASES, <http://nosql-database.org/>, 2010.
- [11] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D. Wallach, M. Burrows, et al., BigTable: A distributed storage system for structured data, in: OSDI'06: Seventh Symposium on Operating System Design and Implementation, USENIX, Seattle, WA, 2006.
- [12] Amazon, Welcome to Amazon SimpleDB, <http://docs.amazonwebservices.com/AmazonSimpleDB/latest/DeveloperGuide/index.html>, 2010.
- [13] J. Haridas, N. Nilakantan, B. Calder, Windows Azure Table, <http://go.microsoft.com/fwlink/?LinkId=153401>, 2009.
- [14] International Virtual Observatory Alliance, VOTable Format Definition Version 1.1, <http://www.ivoa.net/Documents/VOTable/20040811/>, 2004.
- [15] Apache Incubator, Heart (Highly Extensible & Accumulative RDF Table) planet-scale RDF data store and a distributed processing engine based on Hadoop & Hbase, <http://wiki.apache.org/incubator/HeartProposal>, 2010.
- [16] M. King, Amazon SimpleDB and CouchDB Compared, <http://www.automatthew.com/2007/12/amazonsimpledb-and-couchdb-compared.html>, 2007.
- [17] Apache, Hbase implementation of BigTable on Hadoop File System, <http://hbase.apache.org/>, 2010.
- [18] Apache, The CouchDB document-oriented database project, <http://couchdb.apache.org/index.html>, 2010.
- [19] M/Gateway Developments Ltd, M/DB Open Source “plug-compatible” alternative to Amazon’s SimpleDB database, <http://gradvs1.mgateway.com/main/index.html?path=mdb>, 2009.
- [20] ActiveMQ, <http://activemq.apache.org/>, 2009.
- [21] S. Pallickara, G. Fox, NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids, in: ACM/IFIP/USENIX 2003 International Conference on Middleware, Rio de Janeiro, Brazil, Springer-Verlag, New York, Inc., 2003.
- [22] NaradaBrokering, Scalable Publish Subscribe System, <http://www.naradabrokering.org/>, 2010.
- [23] Apache Hadoop, <http://hadoop.apache.org/>, 2009.
- [24] J. Ekanayake, A.S. Balkir, T. Gunarathne, G. Fox, C. Poulain, N. Araujo, et al., DryadLINQ for scientific analyses, in: Fifth IEEE International Conference on eScience, Oxford, 2009.
- [25] J. Ekanayake, T. Gunarathne, J. Qiu, G. Fox, S. Beason, J.Y. Choi, et al., Applicability of DryadLINQ to Scientific Applications, Community Grids Laboratory, Indiana University, 2009.
- [26] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly, Dryad: Distributed data-parallel programs from sequential building blocks, in: ACM SIGOPS Operating Systems Review, ACM Press, 2007.
- [27] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P.K. Gunda, et al., DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language, in: Symposium on Operating System Design and Implementation (OSDI), 2008.
- [28] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
- [29] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu, et al., Twister: a runtime for iterative MapReduce, in: Proceedings of the First International Workshop on MapReduce and Its Applications of ACM HPDC 2010 conference, ACM, Chicago, IL, 20–25 June 2010.

- [30] T. Gunarathne, T. Wu, J. Qiu, G. Fox, MapReduce in the Clouds for Science, in: CloudCom, IUPUI Conference Center, Indianapolis, 30 November–3 December 2010.
- [31] R.S. Dorward, R. Griesemer, S. Quinlan, Interpreting the data: parallel analysis with Sawzall, *Scientific Prog. J.* 13 (4) (2005) 227–298 (Special Issue on Grids and Worldwide Computing Programming Models and Infrastructure).
- [32] Pig! Platform for analyzing large data sets, <http://hadoop.apache.org/pig/>, 2010.
- [33] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins, Pig Latin: a not-so-foreign language for data processing, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, Vancouver, Canada, 2008, pp. 1099–1110.
- [34] G. Malewicz, M.H. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, et al., Pregel: a system for large-scale graph processing, in: Proceedings of the twenty-first annual symposium on parallelism in algorithms and architectures, ACM, Calgary, Canada, 2009, p. 48.
- [35] Cloudera, CDH: A free, stable Hadoop distribution offering RPM, Debian, AWS and automatic configuration options. <http://www.cloudera.com/hadoop/>, 2010.
- [36] A. Grama, G. Karypis, V. Kumar, A. Gupta, *Introduction to Parallel Computing*, second ed., Addison Wesley, 2003.
- [37] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: Sixth Symposium on Operating Systems Design and Implementation, 2004, pp. 137–150.
- [38] H. Kasim, V. March, R. Zhang, S. See, Survey on Parallel Programming Model, in: IFIP International Conference on Network and Parallel Computing, Lecture Notes in Computer Science, Vol. 5245, Springer-Verlag, Shanghai, China, 2008, pp. 266–275.
- [39] S. Hariri, M. Parashar, *Tools and Environments for Parallel and Distributed Computing*, Series on Parallel and Distributed Computing, Wiley, 2004, ISBN:978-0471332886.
- [40] L. Silva, R. Buyya, *Parallel Programming Models and Paradigms*, (2007).
- [41] T. Gunarathne, T. Wu, J. Qiu, G. Fox, Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications, in: Proceedings of the Emerging Computational Methods for the Life Sciences Workshop of ACM HPDC 2010 conference, Chicago, IL, 20–25 June 2010.
- [42] G. Fox, MPI and MapReduce, in: Clusters, Clouds, and Grids for Scientific Computing CCGSC, Flat Rock, NC, <http://grids.ucs.indiana.edu/ptliupages/presentations/CCGSC-Sept8-2010.pptx>, 8 September 2010.
- [43] J. Ekanayake, X. Qiu, T. Gunarathne, S. Beason, G. Fox, High Performance Parallel Computing with Clouds and Cloud Technologies, *Cloud Computing and Software Services: Theory and Techniques*, CRC Press (Taylor and Francis), 2010.
- [44] T. Hoefler, A. Lumsdaine, J. Dongarra, Towards Efficient MapReduce Using MPI, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science: vol. 5759, Springer Verlag, Espoo Finland, 2009, pp. 240–249.
- [45] S. Ibrahim, H. Jin, B. Cheng, H. Cao, S. Wu, L. Qi, CLOUDLET: towards mapreduce implementation on virtual machines, in: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing, ACM, Garching, Germany, 2009, pp. 65–66.
- [46] T. Sandholm, K. Lai, MapReduce optimization using regulated dynamic prioritization, in: Proceedings of the eleventh international joint conference on measurement and modeling of computer systems, ACM, Seattle, WA, 2009, pp. 299–310.
- [47] Wikipedia, MapReduce, <http://en.wikipedia.org/wiki/MapReduce>, 2010 (accessed 06.11.10).
- [48] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: Presentation at OSDI-2004 Conference. <http://labs.google.com/papers/mapreduce-osdi04-slides/index.html>, 2004 (accessed 6.11.10).
- [49] R. Lammel, Google’s MapReduce programming model – Revisited, *Sci. Comput. Prog.* 68 (3) (2007) 208–237.
- [50] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, Improving MapReduce performance in heterogeneous environments, in: Proceedings of the 8th USENIX conference on operating systems design and implementation, USENIX Association, San Diego, California, 2008, pp. 29–42.
- [51] N. Vasic, M. Barisits, V. Salzgeber, D. Kostic, Making cluster applications energy-aware, in: Proceedings of the 1st workshop on automated control for datacenters and clouds, ACM, Barcelona, Spain, 2009, pp. 37–42.
- [52] D.J. DeWitt, E. Paulson, E. Robinson, J. Naughton, J. Royalty, S. Shankar, et al., Clustera: an integrated computation and data management system, in: Proc. VLDB Endow., 2008, 1(1), pp. 28–41.

- [53] S. Ghemawat, H. Gobioff, S. Leung, The Google File System, in: 19th ACM Symposium on Operating Systems Principles, 2003, pp. 20–43.
- [54] Google, Introduction to Parallel Programming and MapReduce, <http://code.google.com/edu/parallel/mapreduce-tutorial.html>, 2010.
- [55] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu, et al., Twister: a runtime for iterative MapReduce, in: Proceedings of the First International Workshop on MapReduce and Its Applications of ACM HPDC 2010 Conference, ACM, Chicago, IL, 20–25 June 2010.
- [56] B. Zhang, Y. Ruan, T. Wu, J. Qiu, A. Hughes, G. Fox, Applying Twister to Scientific Applications, in: CloudCom 2010, IUPUI Conference Center, Indianapolis, 30 November–3 December 2010.
- [57] G. Malewicz, M.H. Austern, A. Bik, J.C. Dehnert, I. Horn, N. Leiser, et al., Pregel: A System for Large-Scale Graph Processing, in: International conference on management of data, Indianapolis, Indiana, 2010, pp. 135–146.
- [58] Y. Bu, B. Howe, M. Balazinska, M.D. Ernst, HaLoop: Efficient Iterative Data Processing on Large Clusters, in: The 36th International Conference on Very Large Data Bases, VLDB Endowment, Vol. 3, Singapore, 13–17 September 2010.
- [59] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: Cluster Computing with Working Sets, in: 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10), Boston, 22 June 2010.
- [60] SALSA Group, Iterative MapReduce, <http://www.iterativemapreduce.org/>, 2010.
- [61] Yahoo, Yahoo! Hadoop Tutorial, <http://developer.yahoo.com/hadoop/tutorial/index.html>, 2010.
- [62] G. Coulouris, J. Dollimore, T. Kindberg, Distributed Systems: Concepts and Design. International Computer Science Series, 4th ed., Addison-Wesley, 2004.
- [63] T. White, Hadoop: The Definitive Guide, Second ed., Yahoo Press, 2010.
- [64] Apache, HDFS Overview, <http://hadoop.apache.org/hdfs/>, 2010.
- [65] Apache, Hadoop MapReduce, <http://hadoop.apache.org/mapreduce/docs/current/index.html>, 2010.
- [66] J. Venner, Pro Hadoop, first ed., Apress, 2009, ISBN:978-1430219422.
- [67] Apache! Pig! (part of Hadoop), <http://pig.apache.org/>, 2010.
- [68] A. Choudhary, G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, S. Ranka, et al., Unified compilation of Fortran 77D and 90D, ACM Lett. Program. Lang. Syst. 2 (1–4) (1993) 95–114.
- [69] Yahoo, Pig! Tutorial. <http://developer.yahoo.com/hadoop/tutorial/module6.html#pig>.
- [70] Systems@ETH Zurich, Massively Parallel Data Analysis with MapReduce. Lectures on MapReduce, Hadoop and Pig Latin. <http://www.systems.ethz.ch/education/past-courses/hs08/map-reduce/map-reduce/lecture-slides>, 2008 (accessed 07.11.10).
- [71] G. Fox, R.D. Williams, P.C. Messina, Parallel computing works! Morgan Kaufmann Publishers, 1994.
- [72] J. Ekanayake, T. Gunarathne, J. Qiu, G. Fox, S. Beason, J. Choi, et al., Applicability of DryadLINQ to Scientific Applications, Community Grids Laboratory, Indiana University, <http://grids.ucs.indiana.edu/ptliupages/publications/DryadReport.pdf>, 2010.
- [73] J. Qiu, J. Ekanayake, T. Gunarathne, J. Choi, S. Bae, Y. Ruan, et al., Data Intensive Computing for Bioinformatics, http://grids.ucs.indiana.edu/ptliupages/publications/DataIntensiveComputing_BookChapter.pdf, 2009.
- [74] J. Ekanayake, T. Gunarathne, J. Qiu, Cloud Technologies for Bioinformatics Applications, IEEE Trans. Parallel Distrib. Syst., (2010).
- [75] J. Qiu, T. Gunarathne, J. Ekanayake, J. Choi, S. Bae, H. Li, et al., Hybrid Cloud and Cluster Computing Paradigms for Life Science Applications, in: 11th Annual Bioinformatics Open Source Conference BOSC, Boston, 9–10 July 2010.
- [76] M. Burrows, The Chubby Lock Service for Loosely-Coupled Distributed Systems, in: OSDI'06: Seventh Symposium on Operating System Design and Implementation, USENIX, Seattle, WA, 2006, pp. 335–350.
- [77] G.C. Fox, A. Ho, E. Chan, W. Wang, Measured characteristics of distributed cloud computing infrastructure for message-based collaboration applications, in: Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems, IEEE Computer Society, 2009, pp. 465–467.
- [78] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, et al., Dynamo: Amazon's highly available key-value store, SIGOPS Oper. Syst. Rev. 41 (6) (2007) 205–220.

- [79] Eucalyptus LLC, White Papers. <http://www.eucalyptus.com/whitepapers>.
- [80] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, et al., The Eucalyptus Open-Source Cloud-Computing System, in: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, Shanghai, 18–21 May 2009, pp. 124–131.
- [81] K. Keahey, I. Foster, T. Freeman, X. Zhang, Virtual workspaces: achieving quality of service and quality of life in the Grid, *Scientific Prog. J.* 13 (4) (2005) 265–275.
- [82] Nimbus, Cloud computing for science, <http://www.nimbusproject.org>, 2010.
- [83] Nimbus, Frequently Asked Questions, <http://www.nimbusproject.org/docs/current/faq.html>, 2010.
- [84] Django, High-Level Python Web Framework, <http://www.djangoproject.com/>, 2010.
- [85] Amazon, Simple Storage Service API Reference: API Version 2006-03-01, <http://awsdocs.s3.amazonaws.com/S3/latest/s3-api.pdf>, 2006.
- [86] boto, Python interface to Amazon Web Services, <http://code.google.com/p/boto/>, 2010.
- [87] S3tools project, Open source tools for accessing Amazon S3 – Simple Storage Service, <http://s3tools.org/s3tools>, 2010.
- [88] bitbucket, JetS3t: open-source Java toolkit and application suite for Amazon Simple Storage Service (Amazon S3), Amazon CloudFront content delivery network, and Google Storage. <http://bitbucket.org/jmurty/jets3t/wiki/Home>.
- [89] Amazon, Amazon Elastic Compute Cloud (Amazon EC2). <http://aws.amazon.com/ec2>.
- [90] OpenNebula, industry standard open source cloud computing tool. <http://opennebula.org/>.
- [91] I.M. Llorente, R. Moreno-Vozmediano, R.S. Montero, Cloud Computing for On-Demand Grid Resource Provisioning, in: Advances in Parallel Computing: High Speed and Large Scale Scientific Computing vol. 18, IOS Press, 2009, pp. 177–191.
- [92] R. Monteroa, R. Moreno-Vozmediano, I. Llorente, An elasticity model for high throughput computing clusters, *J. Parallel Distrib. Comput.* (May) (2010).
- [93] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Capacity Leasing in Cloud Systems Using the OpenNebula Engine, in: 2008 Workshop on Cloud Computing and Its Applications (CCA08), Chicago, IL, 2008, <http://www.cca08.org/papers/Paper20-Sotomayor.pdf>.
- [94] libvirt, virtualization API. <http://libvirt.org>.
- [95] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, *IEEE Internet Comp.* 13 (5) (2009) 14–22.
- [96] ElasticHosts, Flexible servers in the cloud. <http://www.elastichosts.com/>.
- [97] Sector/Sphere, High Performance Distributed File System and Parallel Data Processing Engine. <http://sector.sourceforge.net>.
- [98] Y. Gu, R. Grossman, Sector/Sphere: A Distributed Storage and Computing Platform. SC08 Poster, <http://sector.sourceforge.net/pub/sector-sc08-poster.pdf>, 2008.
- [99] Y. Gu, R. Grossman, Lessons learned from a year's worth of benchmarks of large data clouds, in: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, ACM, Portland, Oregon, 2009, pp. 1–6.
- [100] Y. Gu, R. Grossman, UDT: UDP-based data transfer for high-speed wide area networks, *Comput. Netw.* 51 (7) (2007) 1777–1799.
- [101] FUSE, Filesystem in Userspace. <http://fuse.sourceforge.net>.
- [102] Y. Gu, R.L. Grossman, Sector and sphere: the design and implementation of a high-performance data cloud, *Phil. Trans. R. Soc. A* 367 (2009) 2429–2445.
- [103] Open Stack, Open Source, Open Standards Cloud, <http://openstack.org/index.php>, 2010.
- [104] VSCSE, Big Data for Science. Virtual Summer School hosted by SALSA group at Indiana University, 26 July–30 July 2010, <http://salsahpc.indiana.edu/tutorial/>.
- [105] T. Hey, The Fourth Paradigm: Data-Intensive Scientific Discovery, <http://research.microsoft.com/en-us/um/redmond/events/TonyHey/21216/player.htm>, 2010.
- [106] T. Hey, S. Tansley, K. Tolle, The Fourth Paradigm: Data-Intensive Scientific Discovery, <http://research.microsoft.com/en-us/collaboration/fourthparadigm/>, 2009 (accessed 07.11.10).
- [107] SALSA Group, Catalog of Cloud Material, <http://salsahpc.indiana.edu/content/cloud-materials>, 2010.

- [108] Microsoft Research, Cloud Futures Workshop, <http://research.microsoft.com/en-us/events/cloudfutures2010/default.aspx>, 2010.
- [109] T. Chou, Introduction to Cloud Computing: Business and Technology, Active Book Press, LLC, 2010, p. 252, http://www.lulu.com/items/volume_67/8215000/8215197/1/print/8215197.pdf.
- [110] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616.
- [111] P. Chaganti, Cloud computing with Amazon Web Services, Part 1: Introduction — When it's smarter to rent than to buy, <http://www.ibm.com/developerworks/architecture/library/ar-cloudaws1/>, 2008.
- [112] Cloud computing with Amazon Web Services, Part 2: Storage in the cloud with Amazon Simple Storage Service (S3)—Reliable, flexible, and inexpensive storage and retrieval of your data, <http://www.ibm.com/developerworks/architecture/library/ar-cloudaws2/>, 2008.
- [113] P. Chaganti, Cloud computing with Amazon Web Services, Part 3: Servers on demand with EC2, <http://www.ibm.com/developerworks/architecture/library/ar-cloudaws3/>, 2008.
- [114] M.R. Palankar, A. Iamnitchi, M. Ripeanu, S. Garfinkel, Amazon S3 for science grids: a viable solution? in: Proceedings of the 2008 International Workshop on Data-aware Distributed Computing, ACM, Boston, MA, 2008, pp. 55–64.
- [115] W. Sun, K. Zhang, S. Chen, X. Zhang, H. Liang, Software as a Service: An Integration Perspective, in: Fifth International Conference Service-Oriented Computing – ICSOC, Lecture Notes in Computer Science, Vol. 4749, Springer Verlag, Vienna Austria, 2007, pp. 558–569.
- [116] G. Lakshmanan, Cloud Computing. Relevance to Enterprise, <http://www.infosys.com/cloud-computing/white-papers/Documents/relevance-enterprise.pdf>, 2009.
- [117] N. Leavitt, Is cloud computing really ready for prime time? Computer 42 (1) (2009) 15–20.
- [118] G. Lin, G. Dasmalchi, J. Zhu, Cloud Computing and IT as a Service: Opportunities and Challenges, in: Web Services, ICWS '08, IEEE, Beijing, 23–26 September 2008.
- [119] D.S. Linthicum, Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide, Addison-Wesley Professional, 2009.
- [120] L. Mei, W.K. Chan, T.H. Tse, A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues, in: Asia-Pacific Services Computing Conference. APSCC '08 IEEE, Taiwan, 9–12 December 2008, pp. 464–469.
- [121] G. Fox, S. Bae, J. Ekanayake, X. Qiu, H. Yuan, Parallel Data Mining from Multicore to Cloudy Grids, book chapter of High Speed and Large Scale Scientific Computing, IOS Press, Amsterdam, 2009, <http://grids.ucs.indiana.edu/ptliupages/publications/CetraRoWriteupJune11-09.pdf>.
- [122] J. Ekanayake, G. Fox, High Performance Parallel Computing with Clouds and Cloud Technologies, in: First International Conference CloudComp on Cloud Computing, Munich, Germany, 2009.
- [123] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, et al., BigTable: a distributed storage system for structured data, ACM Trans. Comput. Syst. 26 (2) (2008) 1–26.
- [124] E. Deelman, G. Singh, M. Livny, B. Beriman, J. Good, The cost of doing science on the cloud: the Montage example, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE Press, Austin, Texas, 2008, pp. 1–12, <http://www.csd.uwo.ca/faculty/hanan/cs843/papers/ewa-ec2.pdf>.
- [125] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Beriman, et al., On the Use of Cloud Computing for Scientific Workflows, in: Proceedings of the 2008 Fourth IEEE International Conference on eScience, IEEE Computer Society, 2008, pp. 640–645.
- [126] N. Paton, A. Marcelo, T. De Aragão, K. Lee, A. Alvaro, R. Sakellariou, Optimizing utility in cloud computing through autonomic workload execution, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, <http://www.cs.man.ac.uk/~alvaro/publications/TCDEBull09.pdf>, 2009.
- [127] B. Rochwerger, et al., The Reservoir model and architecture for open federated cloud computing, IBM J. Res. Dev. 53 (4) (2009) 4:1–4:11.

习题

- 6.1 访问 GAE 网站，下载 SDK，阅读 Python 指南或 Java 指南后开始。注意，GAE 只支持 Python、Ruby 和 Java 编程语言。该平台不提供任何 IaaS 服务。

- a. 利用 GAE 平台已有的软件服务，例如 Gmail、Docs 或 CRM，开发一个具体的云应用。在 GAE 平台上测试运行你的程序。
 - b. 报告你的应用开发经验和实验结果，报告时可选择一些性能指标，如作业排队时间、执行时间、资源利用率，或者一些 QoS 属性（如目标完成情况、成功率、容错和成本效率）。
 - c. 在你的 GAE 实验中改变问题规模或数据集大小以及平台配置来学习可扩展性和效率问题。
- 6.2 编写一个运行在 GAE 平台的程序代码，实现备份存储大量个人、家庭或公司数据和记录，例如，相片、视频、音乐、销售收据、文档、新闻媒体、存货、市场记录、财务、供应链信息、人力资源、公共数据集等。注意，在这里需要严格的隐私保护。要达到的另一个目标是最小化存储成本。在第 4 章中可以找到使用 GAE 的接入路径、软件开发工具和平台信息。需要说明你在使用 GAE 平台中的代码开发过程和结果。
- 6.3 获取使用亚马逊 Web 服务的设置，教师可以使用亚马逊提供的教育服务来为学生申请免费的教育账户，详情请访问 <http://aws.amazon.com/education/>。使用关系数据库服务来测试 AWS 平台的 SimpleDB 应用程序。此应用程序可能类似习题 6.1 和 6.2。你应该研究系统的性能、可扩展性、吞吐量、效率、资源利用率、容错和成本效率。
- 6.4 设计和请求一个 AWS 平台的 EC2 配置，用于计算两个阶数超过 50 000 的矩阵并行乘法。
- a. 报告你的实验结果，包括执行时间、速度性能、初始化的 VM 实例、计算单元和存储利用率，以及服务费用。
 - b. 同样在本次科学云实验中，你还可以研究相关问题，如可扩展性、吞吐量、效率、资源利用率，容错和成本效率。
- 6.5 在 AWS 平台上为亚马逊 S3 应用重做习题 6.4，此应用与习题 6.1 和 6.2 类似，你也可以研究性能、可扩展性、吞吐量、效率、资源利用率、容错和成本效率等相关问题。
- 6.6 研究一些商业或服务行业或大企业在 AWS 平台的 EC2 或 S3 上的应用程序的报告。在 Chou 的书^[109]中讲诉了许多案例和成功的故事。例如，Vertica Systems 在 DBMS（数据库管理系统）应用程序中使用 EC2。Eli Lilly 使用 EC2 进行药物开发，Animoto 提供在线服务来促进个人视频制作。联系几家这类服务公司，询问他们技术实现和服务细节。提交一份关于提高可选计算和存储服务应用程序的建议的研究报告。
- 6.7 访问微软 Windows Azure 开发中心，你可以下载 Azure 开发工具集来运行一个 Azure 的本地版本。设计一个应用环境并在本地计算机上测试，例如你的台式机、笔记本电脑或大学的工作站或服务器。报告你使用 Azure 平台的实验过程。
- 6.8 在 MapReduce 编程模型下，有一个特殊的情况只能实施映射阶段，这就是有名的“map-only”问题。这一实现可以增强现有的应用程序/二进制，使它们通过并行方式运行从而拥有高吞吐量；换句话说，它帮助单独的程序利用大规模的计算能力。本次练习的目的是在 Linux 或 UNIX 环境下使用 Twister 编写一个 Hadoop “map-only” 生物信息应用程序 BLAST（NCBI BLAST+；<ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/2.2.23/>）。

基本上，BLAST 二进制在每个分配输入的映射任务中，在“map-only”程序范围内调用。这里的分配输入是映射任务键值对，它可能包含原始输入、输出文件路径，或者其他任何关于二进制本质的元数据。然后每个映射任务在运行时将输入传递到外部的二进制文件，当所有输入传递完毕后，“map-only”程序就关闭。

```
MapOnly (<key, value>) {
    input -> keyValueModification (<key, value>);
    output -> invokeBlastBinary(input);
    return finish }
```

根据上面的伪码，你需要实现 RunnerMap.java (http://salsahpc.indiana.edu/tutorial/source_code/Hadoop-Blast-sketch.zip) 内的 map() 接口，用来执行外部的 Java 进程，在分配输入查询中的每次映射任务都运行单独的 BLAST。“input.fa”需要分成块 (chunk[1-n].fa) 作为一组输入查询，这组输入

查询存储在本地磁盘，你可能需要在 Hadoop Blast 程序执行前上载到 HDFS。HDFS 目录名是程序参数之一。所以对于每个服务器，Hadoop 框架以 `<key, value>` 对的形式分配一个输入查询，例如 `<filename, file path on HDFS>`。

如果你使用提供的 `DataFileInputFormat.java` 和 `FileRecordReader.java` 作为 `InputFormatClass`，则 `<key, value>` 代码已经生成。BLAST 二进制在数据库中查找关于输入查询的匹配记录，这一数据库可以存储在共享的文件系统或者使用分布式缓存的本地磁盘，两种解决方法都可以，取决于你如何编写程序。最后，你需要从每个镜像程序中收集结果并把它们整合到一个输出文件 `result.fa`。另外，请选择一个适合 Linux OS 版本 (i686/i386 或 x86_64) 的 BLAST+ 二进制 (32 位或 64 位)。更多说明请查看 SalsaHPC Hadoop BLAST 教程，带有伪码 (<http://salsahpc.indiana.edu/tutorials/hadoopblast.html>)。

- 6.9 访问 Manjrasoft Aneka 软件中心，网址 www.manjrasoft.com/。下载 Aneka 软件，在你的大学/学生实验室使用连接局域网的计算机创建企业云，运行 Aneka 的各种示例程序。使用 Aneka 任务、线程和 MapReduce 编程模型编写简单的并行程序，运行的时候改变服务器数量从 2 到 20，以步长为 2 递增。
- 6.10 使用 6.6 节描述的学术/开源程序包 Eucalyptus、Nimbus、OpenStack、OpenNebula、Sector/Sphere 重做习题 6.1~6.5 中的应用。这些软件都可以在 FutureGrid 网站 (<http://www.futuregrid.org>) 获得，网上附有大量教程。
- 6.11 尝试在两个或三个云平台 (GAE、AWS 和 Azure) 上运行大规模矩阵乘法运算程序。你也可以选择其他数据密集型应用程序 (如大规模搜索或商业处理应用)。分别在至少两个或所有三个云平台上实现上述应用。主要目标是最小化应用程序的执行时间，次要目标是最小化用户的服务成本。
 - a. 在 GAE 平台上运行服务。
 - b. 在 AWS 平台上运行服务。
 - c. 在 Windows Azure 平台上运行服务。
 - d. 比较在以上平台的计算和存储成本、设计经验和实验结果。报告它们的相关性能和 QoS 测量结果。
- 6.12 MapReduce 引擎及其扩展有三个实现：谷歌 MapReduce、Apache Hadoop 和 Microsoft Dryad。补充完成下表缺失的 14 个表项并在 6 个技术方面比较它们的相似和不同之处。下表中已经给出 4 个表项作为参考答案。不需要详细描述表项的细节，只需填写语言、模式、方法、机制和应用平台的名字。

编程环境	谷歌 MapReduce	Apache Hadoop MapReduce	微软 Dryad
使用的代码语言和编程模型			
数据处理机制			共享目录和本地磁盘
故障处理方法	重新执行失效任务和低速任务的重复执行		
数据分析高级语言			
操作系统和集群环境			Windows HPCS 集群
中间数据传输方法		文件传输或使用 http 链接	

- 6.13 使用在 Aneka 中支持的 MapReduce 编程模型，开发一个图像过滤程序，处理数码相机拍摄的数百张照片。在基于 Aneka 的企业云上做可扩展性实验，改变计算节点/工作机数量和不同分辨率或文件大小的图像，并报告实验结果。
- 6.14 登录 <http://www.futuregrid.org/tutorials>，查看教程，并在 Hadoop 上使用 Eucalyptus、Nimbus，以及 OpenStack、OpenNebula、Sector/Sphere 三者中的至少一个，比较单词计数应用程序。
- 6.15 登录 <http://www.salsahpc.org> 和 <http://www.iterativemapreduce.org/samples.html>，查看教程，比较指

导上的 Hadoop 和 Twister 的具体案例，讨论它们的相对优势和劣势。

- 6.16 下面程序在 Hadoop 中编写，也就是 WebVisCounter。跟踪程序或者在你可以访问到的云平台上运行。分析这个 Hadoop 程序的功能，学习使用 Hadoop 库。

```

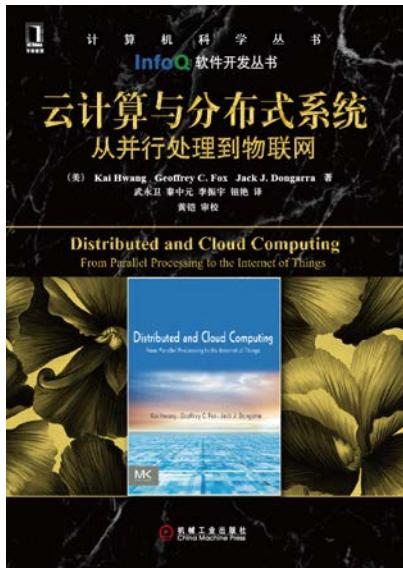
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
static class OSCountMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        Text UserInfo = new Text();
        Text OSversion = new Text();
        int StartIndex, EndIndex, i;
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens() && i != 8) {
            i++; UserInfo.set(tokenizer.nextToken());
        }
        i = 0;
        while (UserInfo.charAt(i) != ';') {
            if (UserInfo.charAt(i) != '('){StartIndex = i}
            i++;
        }
        EndIndex = i;
        OSversion = UserInfo.substring(StartIndex, EndIndex);
        output.collect(OSversion, one); }

    static class OSCountReduce
        extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context){ int sum = 0;
            while (values.hasNext()) { sum += values.next().get(); }
            output.collect(key, new IntWritable(sum)); }
    }
    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("OSCount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        JobClient.runJob(conf); }
}

```

- 6.17 Twister K 均值以迭代方式扩展 MapReduce 编程模型。许多数据分析技术需要迭代计算，例如，K 均值集群化是必须使用多个迭代的 MapReduce 来完成整体计算的应用程序。Twister 是一种有效支持迭代 MapReduce 计算的增强 MapReduce 运行时。对于本题，学习迭代 MapReduce 编程模型和怎样用 Twister 实现 K 均值算法。

- 6.18 DryadLINQ PageRank 是有名的链接分析算法，它计算网页超链接集合的每个元素的数值，这些值反映了随机访问该网页的概率。由于大规模 Web 图中的迭代结构和随机访问模型，很难在 MapReduce 中高效和可编程地实现 PageRank。DryadLINQ 提供了类似 SQL 的查询 API，帮助程序员不太费力地实现 PageRank。此外，Dryad 基础设施以一种简单的方法帮助扩展应用程序。本题将帮助你学习如何用 DryadLINQ 实现简单的 PageRank 应用。
- 6.19 利用 Aneka 中支持的线程编程模型，开发一个阶数大于 500 的两个超大方阵的并行乘法程序。做可扩展性实验，每次以 100 递增，从 500 ~ 1 000 改变矩阵阶数，在基于 Aneka 的企业云上每次以 10 递增，从 10 ~ 50 改变计算节点/服务器，并记录结果。
- 6.20 分别用 Pig Latin 和“赤裸的” Hadoop 来实现你的老师指定的数据密集型应用，并比较。讨论它们的相关优点和缺点。



云计算与分布式系统

原书作者: Kai Hwang等

原书责任编辑: 王春华

迷你书责任编辑: 杨赛

迷你书主页:

<http://www.infoq.com/cn/minibooks/distributed-and-cloud-computing>

《云计算与分布式系统：从并行处理到物联网》一书由机械工业出版社在2013年1月出版，机械工业出版社对该作品享有专有版权，未经授权，不得以任何方式复制或抄袭书中之部分或全部内容。版权所有，侵权必究。

本电子迷你书摘自《云计算与分布式系统：从并行处理到物联网》一书第1、2、6章。由InfoQ制作，属于InfoQ软件开发丛书。如果您打算订购本书完整的纸质版，请登录[豆瓣图书相关页面](#)。本书提到的公司产品或者使用到的商标为产品公司所有。

如果读者要了解具体的商标和注册信息，应该联系相应的公司。

欢迎共同参与InfoQ中文站的内容建设工作，包括原创投稿和翻译等，请联系 editors@cn.infoq.com。