网格计算

（初稿，内部交流，请勿传播）

都志辉　陈渝　刘鹏
编著

李三立 审

内容提要

本书分三大部分，分别对网格的基本原理、网格的实现方法以及网格的应用技术进行论述，总结概括了在网格计算领域的主要研究成果，是国内第一本关于网格计算的参考书。

第一部分网格原理，向读者介绍了网格的基本概念、特点和相关知识，网格的结构组成原则以及当前最重要的网格体系结构实例，最后是网格技术所研究的主要方面和内容。通过本章的介绍，目的是树立读者对网格计算的整体认识，帮助读者把握网格计算最主要、最基本和最核心的内容。

第二部分网格实现，在第一部分的基础上，对网格技术进行更具体的分析。它以当前最重要的网格计算工具包 Globus Toolkit 为基础，介绍了如何在网格基本原理的指导下，去实现一个具体的网格支撑平台。内容涉及网格资源管理、网格数据管理、网格信息服务以及网格安全等多个方面的网格技术。

第三部分网格应用，目的是为读者展示具体的网格功能，通过一个个具体的应用实例，来展示网格广泛的应用领域和应用前景。内容包括分布式超级计算、分布式仪器系统、数据密集型计算以及远程沉浸等几个主要的领域。

本书既可以作为高等院校理工科高年级本科生以及研究生关于网格计算课程的教材和教学参考书，又可以作为网格计算领域科研人员的参考资料，还可以作为相关领域人员了解网格计算知识的参考和自学材料。

希望本书能够对国内网格计算的研究作出一点贡献。

前言

首先感谢李三立院士最早将我们带到网格计算这一研究领域。

网格作为一种建立在互联网之上的新一代基础设施，在国内外的学术界和工业界都引起了广泛的关注。国外的网格计算研究最早从 90 年代初开始，而国内大概在 2000 年左右开始这方面的研究。

李三立院士在国内很早就指出网格计算是一个重要的研究方向，并多方呼吁对这一研究方向加强支持。在李三立院士的建议下，国家教育部在 2000 年 6 月决定启动教育部重点项目“先进计算基础设施北京上海试点工程”来支持这一研究领域。作者很幸运地成为这一项目的研究成员。在李三立院士的领导下，经过近一年的紧张工作，我们成功地完成了这一项目，并在 2001 年通过了国家教育部的验收和鉴定，这是国内第一个通过鉴定的网格计算项目。

之后我们在该项目研究成果的基础上，继续在网格计算领域开展多方面的深入研究，并且密切关注国内外的最新发展动态。

“信息技术专题”是作者为清华大学计算机系高年级本科生讲授的一门课程，根据我们研究的成果，作者将网格计算这一前沿的研究内容在课堂上向同学们进行介绍，引起了同学很大的兴趣，取得了较好的讲课效果。同时在作者的研究生课堂上，对这一研究方向也进行了介绍和共同探讨，研究生对这一领域问题的讨论也非常热烈。在我们研究组的内部也经常就这一领域的不同问题进行交流，这一切都是我们在这一研究领域的宝贵积累，有助于我们更全面、准确、深入地把握这一研究方向。

网格计算权威 Ian Foster 博士的热心帮助是本书问世最直接的原因。由于作者在给同学们上课时需要更丰富的网格计算材料，而关于网格计算的经典著作就是 Ian Foster 和 Carl Kesselman 共同编著的“The Grid：Blueprint for a New Computing Infrastructure”，当时在国内很难找到，Ian Foster 热心地将他的这一著作寄给了作者，我们传阅了此书。由于国内目前还没有这方面的中文图书，通过阅读这一经典著作，我们觉得有义务尽快将这一重要的研究方向以图书的形式，向国内学术界进行介绍，为国内这一领域研究的作出我们的一点贡献。

清华大学出版社的薛慧编辑是一位对当前的学术前沿具有敏锐辨别力的编辑，通过与她的联系，我们的想法得到了薛慧编辑的大力支持和鼓励，并且立即对本书的撰写计划达成一致。在我们紧张的撰写过程中，始终得到了薛慧编辑极大的支持和信任，这使我们非常感动。

感谢和我共同探讨与交流的老师和同学们，感谢 Ian Foster 的热心帮助，感谢薛慧编辑的大力支持。没有他们，就没有这本书。

本书第一篇由都志辉撰写，第二篇陈渝撰写，第三篇刘鹏撰写。

“抛砖引玉”是本书的目的，网格计算是一个崭新的研究领域，欢迎对这一领域有兴趣的朋友和同行为我们多提意见，大家相互切磋，共同促进国内的网格计算研究。

作者

2002 年 8 月 17 日于清华大学

目录

表格索引

图索引

第一篇 网格原理

作者简介

都志辉，博士，清华大学计算机系副教授，主要研究方向为 SPMD 集群式并行算法与程序设计、HPF 编译技术、用户层网络通信技术、跨学科高性能计算应用技术与网格计算技术。1998 年北京大学计算机系博士毕业，然后到清华大学计算机系从事博士后研究。2000 年 5 月博士后出站后留在清华大学计算机系从事教学与科研工作。作为项目负责人完成或者作为骨干成员参加的科研项目有近 20 项，其中通过部委级鉴定的科研项目有 6 项，其中一项获得“上海市科技进步一等奖”，一项获得“教育部科技进步二等奖”。参加工作以来，因表现突出获得“清华之友－优秀教师”奖。

完成了国内第一本关于 MPI 并行程序的著作“高性能计算并行编程技术－－MPI 并行程序设计”，发表的科技论文有三十余篇。目前在清华大学为本科生和研究生讲授关于信息技术与高性能计算技术的两门课程，并承担大量的科研工作。

欢迎读者的批评、指正，作者很乐意与读者就相关领域的问题进行学术讨论。

作者的电子邮件为地址：

duzh@tsinghua.edu.cn　　duzh@bigfoot.com

个人主页：

http://hpclab.cs.tsinghua.edu.cn/~duzh/

第1章 网格基础

“网格”是一个新出现的概念，代表了一种先进的技术和基础设施，是继 Internet 之后又一次重大的科技进步。本章的目的是树立读者关于网格的基本概念，建立宏观的整体认识，抓住网格最主要的特征。为此，本章从网格的内涵、网格的需求以及网格的特点等几个方面对网格进行论述，希望能够回答什么是网格？为什么需要网格以及网格有什么值得注意的特点等重要而基本的问题，为后面深入学习、掌握和运用网格技术建立基本的概念，提供必需的基础性知识。

1.1 网格内涵

在“网格”这两个字的背后，到底代表了一种什么样的技术？网格技术所追求的是一种什么样的前景和应用效果？什么是网格？这些问题，是网格领域研究者需要明确回答的问题，也是目前讨论得非常热烈的问题。本节从基本的概念定义开始，结合网格所希望达到的目的，对网格应该满足的要求，网格的意义以及目前关于网格概念的各种争议等不同的角度展开论述，使读者可以较为全面地从整体上对网格的基本内涵进行把握。

1.1.1 网格的概念

什么是网格（Grid）？网格就是一个集成的计算与资源环境，或者说是一个计算资源池[22]。网格能够充分吸纳各种计算资源，并将它们转化成一种随处可得的、可靠的、标准的同时还是经济的计算能力。除了各种类型的计算机，这里的计算资源还包括网络通信能力、数据资料、仪器设备、甚至是人等各种相关的资源。

什么是网格计算（Grid Computing）呢？基于网格的问题求解就是网格计算。

这里给出的网格和网格计算的概念是相对抽象的，而且是广义的定义，其实网格计算还有狭义的定义。狭义网格定义中的网格资源主要是指分布的计算机资源，而网格计算就是指将分布的计算机组织起来协同解决复杂的科学与工程计算问题。狭义的网格一般被称为计算网格（Computational Grid），即主要用于解决科学与工程计算问题的网格。

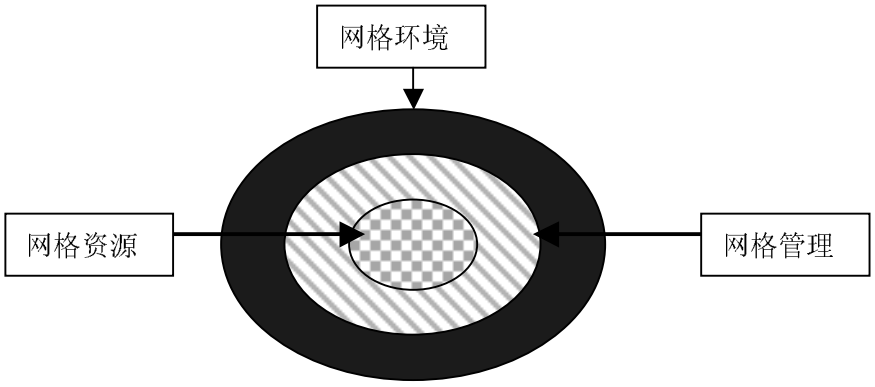


图 1-1 网格资源、网格管理与网格环境三者间的关系

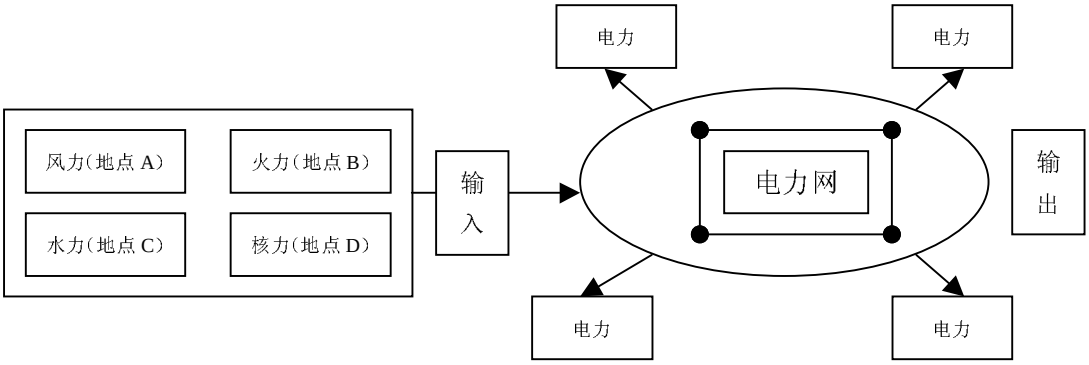
根据求解问题的特点，人们又提出了多种名称的网格，比如以数据密集型问题的处理为核心数据网格[2]，以解决科学问题为核心的科学网格[56]，以全球地球系统模型问题求解为主要目的的地球系统网格[1]等等。此外还有地震网格[3]、军事网格[4]、NASA（National

Aeronautics and Space Administration）的 IPG[27]等行业网格。

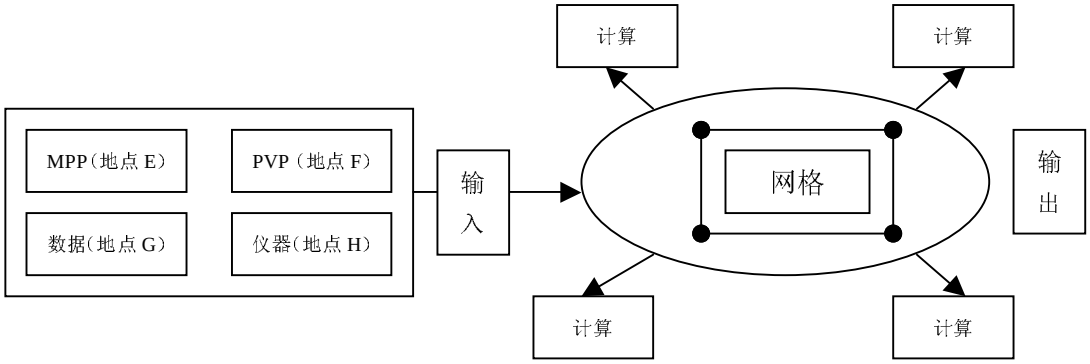
那种认为网格就是仅仅通过网络把计算机、人、仪器、数据等连接起来的观点是过时的，它过分强调了物理的网络和离散的网格资源，而没有将它们作为一个有机的统一整体来看待。另外一种观点就是把网格看作是中间件系统，这种观点也是不全面的。中间件的确在网格中占有很重要的地位，但是网格决不仅仅只是中间件。这两种观点都存在一定的片面性，第一种观点是过分强调网格物理上的资源组成，第二种观点过分强调网格逻辑上的功能，只有将两者结合起来才是完整的网格系统。物理资源本身和对资源的管理与逻辑上的抽象都是十分重要的，而且两者也是密不可分的，它们是网格环境的两大核心组成要素（如图 1-1 所示）。

1.1.2 网格的目的

网格是借鉴电力网（Electric Power Grid）的概念提出来的[22]，网格的最终目的是希望用户在使用网格计算能力时，就如同现在使用电力一样方便。我们在使用电力时，不需要知道它是从哪个地点的发电站输送出来的，也不需要知道该电力是通过什么样的发电机产生的，不管是水力发电，还是通过核反应发电，我们使用的是一种统一形式的“电能”。网格也希望给最终的使用者提供的是与地理位置无关、与具体的计算设施无关的通用的计算能力。图 1-2是对电力网和网格组成的简单对比示意图。



电力网构成示意图



网格组成示意图

图 1-2电力网与网格组成的对比

网格和电力网都有各自资源的消费者和资源提供者，对于电力网来说资源提供者就是发电站，对于网格来说资源提供者是计算机等；对于电力网来说资源消费者就是各种消耗电能的设备，对于网格来说资源消费者就是使用网格计算能力求解问题的用户。不管是电力网还是网格，他们都有覆盖范围广泛，而且组成资源多样的特点。正如同电力网中需要有大量的变电站等设施对电网进行调控一样，网格中也需要大量的管理结点来维护网格正常运行。与电力网相比，网格的结构更复杂，需要解决的问题也更多，但是它也会给我们带来更大的便利和帮助。

1.1.3 网格的基本要求

对于网格提供的计算能力，有四个基本的要求，它们分别是可靠性要求，标准化要求，易访问性要求和价格低廉的要求[22]。

网格的可靠性是指网格提供的计算能力必须保证是持续、稳定和安全的，不应该因为网格内部个别资源的变化而对网格应用造成影响，即网格内部局部资源的变动对网格应用应该是透明的，就如同我们日常使用电灯时不应该因为个别发电厂临时出现什么故障而造成整个电网电力供应的中断一样，电力网应该能够保证实时地从别的发电厂或者其它地区的电网引入电力来弥补本地电力的不足，网格也应该能够保证提供持续、稳定的计算能力。网格还应该满足各种形式的安全要求，比如数据传输的加密，权限的认证，避免非法入侵和非法使用等，如果没有安全性保障，这种先进的计算服务就不能得到广泛的推广。

网格的标准化要求一方面是指网格资源之间应该有一个统一的可以相互访问的接口或者协议标准，因为只有这样才能够实现网格资源的互操作从而实现充分的资源共享，标准化是共享的前提；标准化的另外一个含义是指网格对用户提供的计算能力应该满足一定的标准，有一种比较统一的形式，从而便于以一种统一的方式进行访问，对于访问者来说，不能因为时间、地点、具体的访问系统等的不同而要求不断改变访问形式，访问形式应该有一致性，当然一致性的前提是网格必须提供给用户一个相对稳定的标准化接口。

网格的易访问性要求是指用户可以在任何时间，任何地点，以自己习惯的统一的形式访问和使用各种网格资源。网格计算能力可以通过网格系统输送到任何角落，随处可得。换句话说，在网格上是没有资源处在什么位置的概念的，只有“在网格上”或者“不在网格上”的区别，无论你在什么地方，网格资源都在你的旁边。人们以前在解决特定问题时或许不得不到特定的地点来进行，比如到某一个单位去登记和使用特殊的仪器设备等，但是在网格上解决问题时，不应该因为访问者或者资源所在地位置不同而受到限制。

网格费用的低廉性要求是网格能够被普遍接受和推广的前提，不管网格有多少优点，如果大多数的使用者无法承受其费用，网格就不可能被普及，它的各种优势也就根本无法得到体现。网格技术通过将资源充分共享，最大限度发挥资源的使用价值，可以将原来闲置和浪费的资源收集起来供网格用户使用，而且可以避免以前由于地理位置限制所带来的各种额外开销，显然网格对使用者存在着很大的降低开销的潜力。

这些要求，都是网格需要解决的问题，也是网格技术发挥作用的地方。网格作为一种新型而重要的基础设施，不是一夜之间就能够奇迹般地突然出现的，需要各个方面联合起来，共同努力才可以实现。

1.1.4 网格的意义

网格概念的提出将从根本上改变人们对“计算”的看法，因为网格提供的是与以往根本

不同的计算方式。Randy Bramley 认为网格提供的计算能力是以前所无法得到的，而且也是不能够通过其它的方式得到的。网格概念的核心就是突破了以往强加在计算资源之上的种种限制，使人们可以以一种全新的更自由、更方便的方式使用计算资源，解决更复杂的问题。

首先是计算能力大小的限制，以前大部分的用户无法得到足够的计算能力，因此许多问题的解决是不能够通过计算或者是不能完全靠计算来实现的，对模型以及算法的化简是最常见的近似方法。而网格所提供的计算能力要远远超过以前我们所能够想象的程度，对于大多数用户来说，网格提供给他们的计算能力足以满足其计算需求，在这种计算能力的支持下，人们可以做许多以前无法想象和无法完成的工作。

其次是地理位置的限制，计算资源是分布在各处的，有些资源是稀缺或不可复制的，有些资源甚至是无法和特定的地理位置分开的，因此要使用这些资源，在以前许多情况下必须到相应的地方去，这在很大程度上限制了这些资源的使用。而网格把“到资源所在的位置”对资源进行使用的限制打破了，对资源的使用和使用者所在地位置以及资源所在地位置无关。突破了在使用资源时对位置的限制，是网格的具有突出意义的功能。

最后也是非常重要的一点就是网格打破了传统的共享或协作方面的限制，以前对资源的共享往往停留在数据文件传输的层次，而网格资源的共享允许对其它的资源进行直接的控制，而且共享资源的各方在协作时可以以多种方式更广泛地交流信息，充分利用网格提供的各种功能。比如为了分析臭氧层问题可以通过网格将各个领域的专家、各种大型专业数据库、大型计算设备、各种模型库和算法库等充分结合起来，协同研究这一问题。网格使得共享与协作的方式和方法更广泛了，而且为这种合作提供了各种控制策略与手段，可以根据需要，动态地与不同的组织与个人建立各种级别的工作关系。

过去人们往往很自然地把计算资源和特定的有形的计算机等联系起来，而网格就是在剥去了各种具体的计算资源外在的“形”的基础上，将其内在的“神”即计算能力抽取出来，形成一种分布在网上的抽象的计算能力，在实现了“形”和“神”分离的同时，将原来有形的、专用的计算能力转化为一种无形的、更通用的计算能力，正如同电力网将具体的各种类型的发电机的电力转化为一种我们认为根本没有什么差别的统一的电力一样。

这种观念和使用方式上的改变，是由网格技术支持的，不是凭空产生的。网格的意义，就如同互联网改变了人们传统的通信方式和通信手段一样，它将改变人们传统的计算方式和计算手段，网格技术将为人们提供更强大、更方便、更高级的问题求解手段。

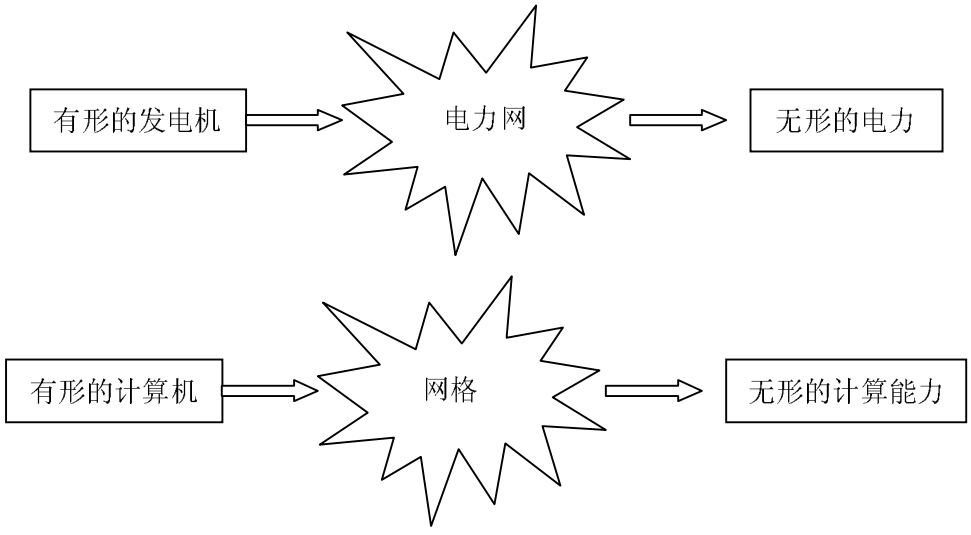


图 1-3电力网和网格作用的对比

1.1.5 网格概念的分歧

到目前为止，关于什么是网格和什么是网格计算，还没有一个普遍接受的定义，关于网格概念的分歧和争议仍然存在。

下面看一下其它的关于网格或者网格计算的定义。

- 网格就是下一代的 Internet[18]。这是一种非常简洁的定义方式，借助于人们对 Internet 的了解和认识，来说明网格可能对我们造成的巨大影响，但是它没有对网格到底是什么给出直接的说明，这一定义强调的是网格的重要性和意义。
- 网格计算就是在动态变化的、拥有多个部门或者团体的复杂虚拟组织（Virtual Organization）内，灵活、安全的协同资源共享与问题求解。所谓虚拟组织就是一些个人、组织或者资源的动态组合[19]。这一概念强调的是网格是为虚拟组织服务的，网格必须具备动态、协同资源共享的特点。
- 网格就是方便资源管理，有效支持广域分布的、多领域的科学与工程问题解决的中间件系统[57]。这一定义强调的是中间件系统在网格中的作用，正是因为中间件的存在，所以才使网格可以表现出与以往系统不同的特点，实现资源的充分共享，网格的功能是在中间件的支持和管理下完成的。
- 网格是建造分布式科学计算环境的一种一体化的集成方法，这一环境包括计算、数据管理、科学仪器以及人类的协作[56]。这一定义强调的是主要用于科学计算的网格所应该具备的特点。
- 网格是一种无缝的、集成的计算与协作环境。这种定义强调的是网格的集成化特点，说明网格是由各种不同的资源组成的，但是这些资源的集成是无缝的。
- 网格是基于硬件支持的各种服务和功能的提供者（Randy Bramley）。网格可以提供以前不能得到的特有功能，而且是无法通过其它方式得到的。

关于网格或者网格计算的定义为什么会这么多？首先，这说明了不同的定义者看问题的角度以及侧重点的不同，也就是说网格具有多方面的特点，网格功能具有多面性；其次，网格是面向问题的，网格是在真实的问题求解中存在的，网格所面临的问题不同，对网格提出的要求就不同，因此在不同问题求解者眼中的网格是有所不同的；还有，网格还正在发展阶段，并没有定型，因此出现各种各样的关于网格的定义就不足为奇了，这也从另外一个方面说明了当前网格研究的热烈程度。

随着大量网格项目的展开和网格技术的发展，关于网格概念和技术的一些重要方面已经取得了相当大的相互认同，这说明了人们对网格认识的深入。应该指出，网格概念应该是面向问题的，随着人们解决问题的重点发生变化，网格的概念也必然会产生转变，问题求解才是网格的最终目的。

1.2 网格需求

我们为什么需要网格？首先是因为通过计算来解决问题已经成为一种重要的解决问题的方法和手段，而目前大量问题的解决，只有网格提供的能力才可以满足要求；其次就是相关技术的发展为网格的出现奠定了基础，可以满足网格的要求；最后就是网格可以有广泛的应用领域，几乎各种人群都可以利用网格来解决他们面临的各种问题，网格具有很大的发展潜力，开发网格技术，建设网格，发展网格，已经成为学术界、工业界以及政府部门的共识。下面从几个不同的方面来进行详细论述。

1.2.1 计算的重要性

“计算”与理论和实验并列，已经成为第三种重要的科学研究方式[58]。并且计算将理论和实验连接起来，成为二者之间的桥梁。通过计算，可以完成许多单纯依靠理论或者实验无法进行的科学研究。比如许多问题根本就无法给出解析解，但是通过计算机模拟，就可以得到比较可靠的近似解，而且还可以把误差控制在一定的范围之内。再比如为了研究超高温或者超高压等特殊条件下材料的性质，很难进行真实的物理实验，而且实验成本往往也特别高，但是可以通过计算模拟的方法来得到所需要的结果。人们已经用计算的方法创造了一个又一个的奇迹，比如复杂的科学与工程问题的建模与模拟，医疗诊断，工业设备控制，天气预测，股市管理等，这些都说明，计算已经在各个研究领域取得了越来越重要的地位。

不仅在科学研究中，而且在越来越多的社会与经济活动中，“计算”也已经成为一种重要的甚至是不可替代的解决问题的方法与工具。虽然计算机的发明一开始最直接的目标是为科学计算服务的，但是，后来计算机最广泛的应用还是在事务处理等非科学计算领域。现在的银行系统，订票系统，办公系统，电子政务等等，都需要计算机提供的非数值计算能力。在我们的日常生活中，已经无法离开计算。目前的环境污染和整治问题，它是一个动态的、非线性的、多学科、时间和空间多尺度的难题，科学地解决这一问题的方法就是计算。

计算在问题求解中的重要地位，为网格这种主要以提供计算能力为特征的基础设施的出现奠定了基础，只要我们需要计算，网格就有用武之地。

1.2.2 问题的需求

随着人们求解问题领域的不断拓展，所遇到的也问题越来越复杂，而且规模越来越大，解决这些问题所需要的计算能力也在大幅度提高。比如在天文学研究中，天文望远镜每年所产生的数据不少于 10 Petabytes（1Petabytes = 10^{15}bytes，1Terabytes=10^{12}bytes，1Gigabytes=10^{9}bytes，1Megabytes=10^{6}bytes，）[59]。假设计算机处理 1Megabytes 的数据需要 1 秒钟，则处理 10Petabytes 的数据需要 10^{10} 秒约 3 百多年才能够处理完毕，显然这样的计算机是不能满足要求的，这里还没有考虑数据的读写所需要的时间。假设目前一块硬盘的容量为 100Gigabytes，则存放 10Petabytes 的数据需要（10×10^{15}）/（100×10^{9}）$=1\times10^{5}$ 块硬盘。

在建立数字化人脑的研究中，如果人脑体素的分辨率为微米，则建立彩色的数字化人脑需要大小为 4.5petabytes 的数据[59]。

同时，目前在线数据的数据量也在急剧增长，2000 年为 0.5petabyte，估计到 2005 年为 10petabytes，2010 年为 100petabytes。在高能和核物理研究中，在重力波的研究中，在与时间有关的三维系统研究中（地球观察，气候模型，地球物理，地震模型，流体，空气动力设计，污染物扩散分析），天文学，医学，晶体学，基因组研究（人类及其它的物种基因数据库），虚拟实验室[59]等的研究中，需要的都是具有超大规模的计算和数据分析能力。其它的问题还包括计算密集型分析，大量的数据整理和收集，地理分布的协作等等。

在这些新问题的求解过程中，局部的计算资源是无法满足这样的需求的，因此必须使用广大的分布资源，将他们集中起来协同解决问题。

同时也应当看到，由于各种因素的限制，有些资源由于成本过高或者其它的原因，往往是不可复制的，因此为了有效地使用这些资源，打破地域的限制来实现更大粒度和更大范围的资源共享就成为一种必需的要求。

因而，网格这种以更大范围的资源共享为目的的计算方式的出现就具有一定的必然性，

但是它的出现还必然受到具体的技术条件的限制，只有相关的技术成熟了，才能够推动网格的发展，网格才能够真正为大家服务。

1.2.3 相关技术的发展

这里主要从网络和计算机两个方面的发展对网格的支持进行论述。网络的发展主要是看其网络带宽和覆盖的范围来论述，计算机则主要从其计算能力和计算形式的改变来论述。

首先回顾一下网络的发展。ARPANET 是 Internet 的前身，开始于 20 世纪 70 年代初，它是一些科学家和 DoD（Department of Defense）的实验性网络。它开发了传输协议 TCP/IP，并提出了一些重要的概念，一些后来的产品和研究都是在此基础上进行的。其中一个重要的网络就是 NSFNET，1986 年建立的带宽为 56Kb/s 的主干网，连接了美国 5 个 NSF 超级计算中心。主干网的带宽分别在 1980 年和 1990 年左右提高到 1.5Mb/s 和 45Mb/s 左右，后来扩展到现在的 Internet。

1998 年，在美国一种新的主干网络 vBNS(very high speed backbone network service)又建立起来了，它连接了大概 100 多家研究机构。后来在此基础上，一种连接美国、加拿大、新加坡、台湾以及韩国等许多国家和地区的网络 STAR TAP（Science, Technology, And Research Transit Access Point）[52][53]建立起来，它为世界范围的网格的出现提供了基础。

目前主干网的带宽已经从原来的 56Kb/s 发展到十兆、百兆乃至千兆，网络速度的提高为跨地域的资源共享提供了基础和前提，正如 George Gilder 所说的那样，当外部网络的速度和计算机内部网络的速度一样时，分布在网络上的计算机将形成一种具有特定目的的联盟[59]。

根据摩尔定理，计算机的芯片的集成度或者说计算机 CPU 的处理速度大约每 18 个月翻一番，从 1986 年到 2000 年，计算机的速度提高了 500 倍，网络的速度提高了约 340,000 倍。估计从 2001 年到 2010 年，计算机的速度将提高 60 倍，而网络的速度将提高 4000 倍[59]。目前的 PC 已经比 10 年前的 Cray 超级计算机还要快，而大量的微机在许多情况下其计算能力是闲置的，因此闲置的计算能力是可以通过共享手段让其它的计算用户受益的。由于计算机绝对速度的提高，将大量闲置的计算资源充分利用起来就成为网格的另外一个重要目的。

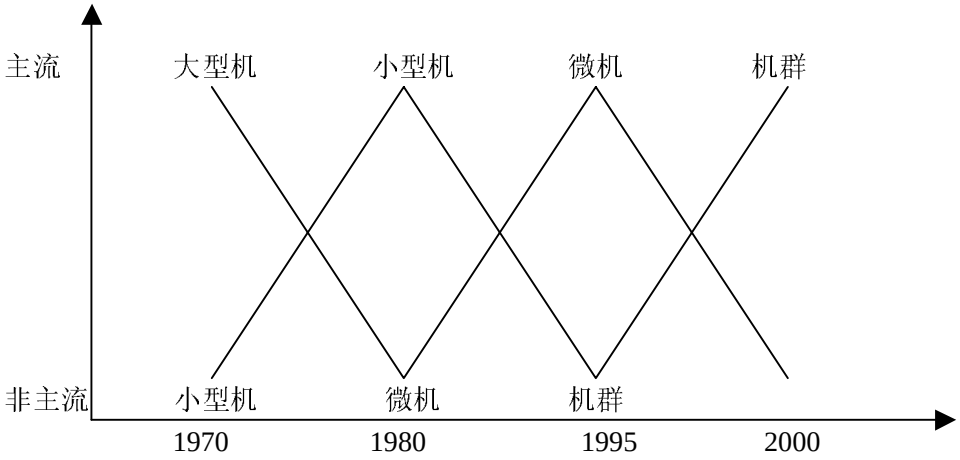


图 1-4主流计算形式的变化

计算机的发展也经历了许多阶段，在不同的阶段，都有一种形式的计算占主导地位。比如在 70 年代左右，大型机占据主导地位，代表了当时的主要计算机技术；到了 80 年代，小

型机的逐渐崛起，计算机的能力不再以体积的庞大为特征了，特别是到了 90 年代，随着微处理器技术的迅速发展，CPU 的集成度越来越高，微机成为一种广为普及的计算机类型，而微机性能的提高和高速系统级网络计算的发展以及成熟和免费的操作系统的出现，为机群计算形式的出现提供了基础。每一个发展过程都大大提高了计算机的普及程度和计算机的性能，都为更大，更复杂问题的解决提供了支持。我们估计在不久的将来，网格会成为主流的计算形式。图 1-4给出了这种发展变化过程的示意图。

1.2.4 网格的应用领域

为什么需要网格，还因为网格可以有非常广泛的应用领域。一旦建立起了网格，就可以开展许多以前无法进行的工作和研究。

在科学计算领域，网格可以在如下几个方面得到应用[22]。

- 分布式超级计算。这和以前的高性能计算的作用十分类似，不同的是以前的高性能计算大多是集中式的，主要靠一个地方的高性能计算机完成计算任务。目前遇到的许多科学与工程计算问题是无法在任何一台超级计算机上解决的，因此需要更多的超级计算机一起来完成，网格可以把分布式的超级计算机集中起来，协同解决复杂的大规模问题。从集中计算到分布计算，是网格功能的重要体现。
- 高吞吐率计算。高吞吐率计算和高性能（超级）计算的侧重点是不同的，高性能计算关心的是每秒能够完成的计算量，度量的时间单位很小。而对于高吞吐率计算，它关心的是几个月、一年甚至是几年完成的计算量，度量的时间单位比较大。只所以会提出这种计算方式是因为在许多实际的问题求解过程中，人们关心的是在一段相对较长的时间内（比如一年）解决问题的多少，而对短期内求解问题的多少并不是十分关心。对于这样的问题，可以利用 CPU 周期窃取的技术，将大量空闲计算机的计算资源集中起来，提供给对时间不太敏感的问题，作为计算资源的一种重要来源。[17]就是这样的系统。
- 数据密集型计算。对于数据密集型问题，数据采集地点、数据处理地点、数据分析与结果存放地点、可视化设备的地点等往往不在同一个地方，数据密集型问题的求解往往同时会产生很大的通信和计算需求，需要网格能力才可以解决。许多高能物理实验，数字化天空扫描，气象预测等都是数据密集型问题，网格可以在这类问题的求解中发挥巨大作用。

在社会经济生活领域，网格可以在如下领域得到应用。

- 基于广泛信息共享的人与人交互。原来的人与人的交互受到地理位置、交互能力、共享对象等等许多条件的限制。一个国际会议往往需要许多人在旅途上消耗大量的时间，如果每个人都可以在自己的工作地点，与参加会议的其它人员在一个虚拟的共享空间中进行交互，共同讨论问题，可以产生面对面的效果，无疑将会是十分理想的。一个原来物理上集中的大会场被网格技术分散在世界各地，但是又不影响开会的效果，一个原来在物理会场中传递的话筒可以在世界不同地点的人们之间传递。这显然会对大家的工作方式产生很大的影响。
- 更广泛的资源贸易。计算能力闲置的机器可以共享出来，通过网格让更多的人来租用；需要计算能力的人可以不必购买更大的计算机，只要根据自己计算任务的需求，向网格购买计算能力就可以满足要求。除了计算资源，包括贵重仪器、程序、数据、信息、文化产品等等各种资源都可以在贸易的基础上广泛共享。

网格是一种面向问题和应用的技术，随着网格技术的不断完善和应用领域的不断扩展，网格可以在更多的领域得到应用，发挥更大的作用。

1.2.5 网格的用户群

为什么需要网格，还因为网格有十分广大的潜在用户群。这些用户群涉及到几乎所有的领域，从尖端的科研到日常的生活，广大的应用领域必然有广大的用户群。

在科学研究领域，就有许多不同类别的研究者需要网格[22]。

网格就是从计算科学与工程领域逐渐发展起来的，计算科学家和工程师需要网格。因为他们需要实时地查看不同应用的运行情况，有些应用还需要对计算过程进行监控，这样，通过高速主干网络，可以把图形文件发送到局部的可视化设备上观看正在模拟的结果，其它的分布在不同实验室的科学家也可以使用不同的计算机来解决同一问题。将研究者和远处的仪器、设备、传感器等与网格建立连接，进行三维可视化模拟，召开远程会议。这种经常性的需求需要网格技术的支持。

实验科学家也需要网格。理论或者计算科学家和与实验科学家的比例大概是 1:10，有影响的科学需要有影响的实验或者观察科学。实验科学家希望将远程仪器与超级计算机连接起来，通过高级的用户界面来控制仪器，远程沉浸或者远程脑外科手术就是这样。一些天文仪器或者设备，常常设置在一些比较偏远的地方，这些地方往往离科研机构或者大学比较远，这对实验科学家来说是十分不方便的，在网格的帮助下，实验科学家可以将这些数据交给其它地方的计算机进行处理，根据计算机的处理结果又可以对仪器进行控制。

此外，社会和经济团体也需要网格，比如协会，公司，人类公共问题研究机构等。

网格具有天然的增强协作的能力和作用，协会之间通过网格可以共享分布的信息与资源。对于全球性的大公司，通过网格技术可以使公司方便地对全球各个部分的子公司统一管理，共享资源与信息，高效地运转。此外由于网络与网格技术的发展，也为建立在网格技术之上的各种新型公司的出现提供了基础。

大范围的环境保护已经成为一个全球性的问题，比如臭氧枯竭，全球变暖，空气以及水污染等。通过建立交互式的可靠的大型知识库，所有领域的研究者可以充分利用知识库集成的关于这些问题的科学知识，协同起来，解决这些多学科性质的问题，这一问题解决所涉及的机构和专家都需要网格的支持。

培训和教育需要网格。如果我们能够建造这样一个培训和教育环境，培训者或者教师只需走到一面充满魔力的接通网格的电子墙的前面，就可以为分布在各地的员工进行技能培训，或者为不同地区的学生讲授一门基础课，讲完后可以直接回到自己的办公室继续工作，而不是在各地奔波，这将是非常理想的情况。这里问题的重点不是如何用网格技术来重新实现经典的教育，而是如何使用网格来实现何种新类型的教育或者培训形式。

进一步，我们可以说，不同的国家，乃至世界都需要网格。正如互联网的发展一样，网格的发展不会局限在一个国家的范围之内。网格是永久性的基础设施，世界范围内网格的建立，将对其它各个方面的发展提供基础性的支持，反过来也必将促进网格的全面发展。

1.3 网格特点

网格作为一种新出现的重要的基础性设施，和其它的系统相比，有什么样的重要特点？这些特点对网格技术、网格应用以及网格建设又有什么样的影响？只有了解了网格的特点，才能够更好地认识和把握好网格的开发和应用。下面分别从网格的分布性、自相似性、动态多样性（不可预测性）以及管理的多重性等多个方面，对网格特点展开介绍，读者可以通过对网格特点的了解，更深入地认识和把握网格。

1.3.1 分布与共享

分布性是网格的一个最主要的特点。网格的分布性首先是指网格的资源是分布的。组成网格的计算能力不同的计算机，各种类型的数据库乃至电子图书馆，以及其它的各种设备与资源，是分布在地理位置互不相同的多个地方，而不是集中在一起的。分布的网格一般涉及的资源类型复杂，规模较大，跨越的地理范围较广。

因为网格资源是分布的，因此基于网格的计算一定是分布式计算而不是集中式计算。在网格这一分布式环境下，需要解决资源与任务的分配和调度问题，安全传输与通信问题，实时性保障问题，人与系统以及人与人之间的交互问题等等。

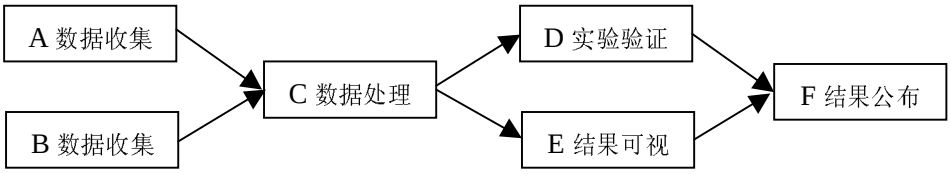


图 1-5 网格的分布性

如图 1-5所示，一个问题的求解需要从 A 和 B 两个不同的地方获取得数据，然后将这些数据送到专门的机构 C 进行数据的分析和处理，对于处理后的结果，需要进一步在 D 处通过实验进行验证，并利用 E 处的高级可视化设备进行结果显示，而问题最终得到的结果可能是在 F 处进行公布。这一问题求解过程涉及到了 A,B,C,D,E,F 六个不同的地方，这些地方有可能相距千万里之遥，甚至有些时候还需要一些移动设备的介入，这些都说明了网格的分布性特征。

网格资源虽然是分布的，但是他们却是可以充分共享的。即网格上的任何资源都可以提供给网格上的任何使用者。共享是网格的目的，没有共享便没有网格，解决分布资源的共享问题，是网格的核心内容。这里共享的含义是非常广泛的，不仅指一个地方的计算机可以用来完成其它地方的任务，还可以指中间结果，数据库，专业模型库，以及人才资源等各方面的内容。

分布是网格硬件在物理上的特征，而共享是在网格软件支持下实现的逻辑上的特征，这两者对于网格来说都是十分重要的。

1.3.2 自相似性

分形模型有一个非常重要的特征就是自相似性，这在一些分形图形中体现的十分直观和醒目（如图 1-6所示）。自相似性在许多自然和社会现象中大量存在，一些复杂系统在都具有这种特征，网格就是这样。网格的局部和整体之间存在着一定的相似性，局部往往在许多地方具有全局的某些特征，而全局的特征在局部也有一定的体现。

可以认为国家级的网格是在省一级的网格基础之上建造起来的，国家级主干网要有更大的带宽，只有这样才可以将不同省份的子网格连接起来提供满意的通信服务；国家级和省级网格都会有各自的计算中心，只不过在计算能力上有差异而已；他们也都需要管理结点，只不过国家级的管理结点管理功能需要更多、更强大而已。除了相似性之外，整体和部分之间必然有不同的地方，如图 1-7所示。

再比如我们可以在一个实验楼里建立一个小规模的实验网格，然后可以把整个学校的多

个实验网格联系起来形成一个全学校的教学科研网格，不同学校之间的内部网格可以互相连接起来形成一个高校之间的网格联盟，这一网格联盟又可以成为全国网格的一个部分。这种整体和部分之间的相似性可以在多个阶段看到。网格的自相似性在网格的建造和研究过程中有重要的意义。

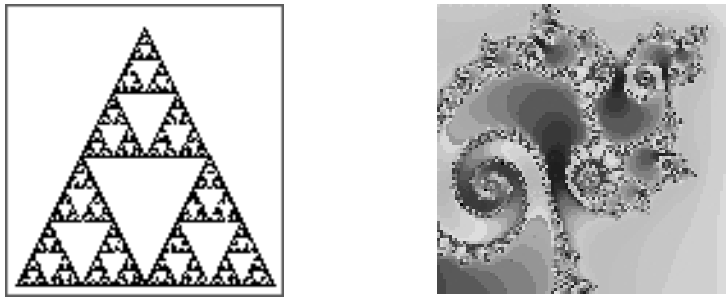


图 1-6 分形图形

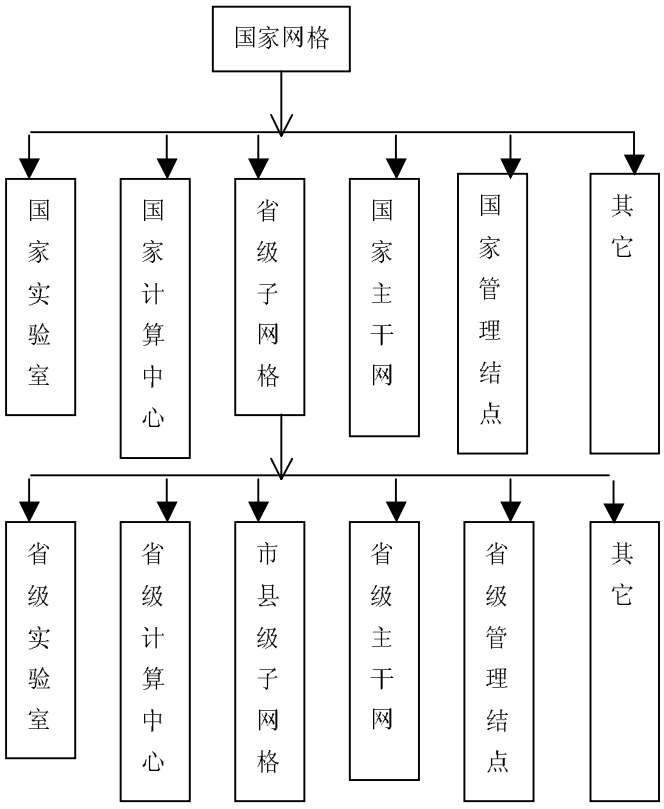


图 1-7 网格的自相似性

1.3.3 动态性与多样性

对于网格来说，决不能假设它是一成不变的。原来拥有的资源或者功能，在下一时刻可能就会出现故障或者不可用；而原来没有的资源，可能随着时间的推移会不断地加入进来。网格的动态性包括动态增加和动态减少两个方面的含义。

网格资源的动态变化特点要求网格管理必须充分考虑并解决好这一问题，对于网格资源的动态减少或者资源出现故障的情况，要求网格能够及时采取措施，实现任务的自动迁移，做到对高层用户透明或者尽可能减少用户的损失。

网格资源的动态增加需要提高网格的扩展性问题，也就是说在网格的设计与实现时，必须考虑到新的资源能否很自然地加入到网格中来，并且可以和原来的资源融合在一起，共同发挥作用。网格扩展要求体现在规模、能力、兼容性等几个方面。一开始网格的规模往往不是特别大，不需要也不可能一步到位，但是网格应该能够允许对它自身进行多种形式的扩展，网格规模扩展后网格的相应管理软件也应该能够满足扩展性的要求，网格软件的升级要能够向下兼容。

网格资源是异构和多样的。在网格环境中可以有不同体系结构的计算机系统和类别不同的资源，因此网格系统必须能够解决这些不同结构、不同类别资源之间的通信和互操作问题。正是因为异构性或者说资源多样性的存在，为网格软件的设计提出了更大的挑战，只有解决好这一问题，才会使网格更有吸引力。

1.3.4 自治性与管理的多重性

网格上的资源，首先是属于某一个组织或者个人的，因此网格资源的拥有者对该资源具有最高级别的管理权限，网格应该允许资源拥有者对他的资源有自主的管理能力，这就是网格的自治性。

但是网格资源也必须接受网格的统一管理，否则不同的资源就无法建立相互之间的联系，无法实现共享和互操作，无法作为一个整体为更多的用户提供方便的服务。

因此网格的管理具有多重性，一方面它允许网格资源的拥有者对网格资源具有自主性的管理，另一方面又要求网格资源必须接受网格的统一管理。

1.4 小结

本章主要是介绍关于网格的一些基本知识和基本概念，通过阅读本章，希望读者能够对网格产生一个宏观的整体上的认识和理解，为以后深入探讨网格技术提供基础。

由于网格是一个新出现的概念和技术，其中还有大量存在争议的内容，作者在这里介绍了一些自己的认识和观点，目的是希望能够起到抛砖引玉的作用，让更多的人加入到网格的研究之中来，共同推动网格在我国的发展。

阅读完本章，如果读者能够在自己的脑海中描绘出一幅自己心目中的关于网格的蓝图，那就达到了本章的目的。当然读者头脑中一定还有许多问题存在，后续的章节会一步步更具体地回答这些问题。

思考题

1. 什么是网格？你对网格内涵和外延是怎样认识的？
2. 你认为网格概念和已有的什么概念最相似？这些相似之处体现在什么地方？对我们有什么启发？
3. 你认为网格最核心的思想是什么？网格思想是不是一场革命？
4. 网格计算到底能够为我们的问题求解带来什么样的好处？
5. 网格有哪些基本的特点？基于这些特点，在网格设计、建造和应用过程中需要注意什么

问题？

6. 网格最基本的构成要素是什么？这些要素之间是怎样的关系？
7. 网格和互联网的关系是怎样的？
8. 你认为哪些技术对网格的出现起到了决定性的作用？
9. 网格作为一种基础设施，和已有的高速公路、铁路、银行等有什么相同和不同之处？
10. 请讨论和想象一下，网格有哪些可能的应用形式？网格的大规模应用和推广会对我们的生活会产生怎样的影响？
11. 你认为比网格更理想的计算方式或者说基础设施应该是怎样的？网格计算方式是否有缺点或者局限性？如果有，如何进行改进？改进后可以到什么样的效果？

第2章 网格体系结构

本章是在读者了解关于网格的基本概念和知识的基础上，为读者讲述网格这一基础设施的整体框架结构即网格体系结构。在详细了解网格的各种技术之前，先了解网格的总体结构框架，会给读者建立一个关于网格的整体性认识，这对于深入了解和掌握网格的精髓是十分重要的。本章重点介绍了两种目前最重要、最有影响的结构形式，即五层沙漏结构和开放网格服务结构，通过对它们的学习，有利于我们更好地把握网格的核心技术和整体特征，为设计和开发高效实用的网格系统提供基础。

2.1 网格体系结构的概念

在讨论网格体系结构之前，必须先说明什么是网格体系结构，网格体系结构包括哪些关键的要素与部分，网格体系结构的精髓是什么，下面将分别回答这些问题。

2.1.1 定义

和体系结构或者系统结构对应的英文单词是 Architecture，在定义网格体系结构之前，先看一下关于体系结构的定义。在韦氏词典（Merriam-Webster’s Collegiate Dictionary）中有如下几个相关的定义：

- 建造的艺术或者科学（The art or science of building）
- 建造的方法或者风格（a method or style of building）
- 计算机或者计算机系统各部分组织与集成的方式（the manner in which the components of a computer or computer system are organized and integrated）

在我们的《现代汉语词典》中，对结构是这样定义的：

- 各个组成部分的搭配和排列
- 建筑物上承载重力或外力部分的构造。

综合以上几个定义，结合网格的特点，这里给出如下关于网格体系结构的定义：

网格体系结构就是关于如何建造网格的技术。它给出了网格的基本组成与功能，描述了网格各组成部分的关系以及它们集成的方式或方法，刻画了支持网格有效运转的机制。

这里我们主要是从科学的角度而不是艺术的角度来探讨网格体系结构，因此我们认为网格体系结构是一门技术，而不是艺术。

明确对网格体系结构进行论述的文献并不多，在[19]中，Foster 将网格体系结构定义为“划分系统基本组件，指定系统组件的目的与功能，说明组件之间如何相互作用的技术”。和我们的定义相比，这一定义强调的是组件与组件之间的相互作用，而我们的定义还要求说明各个部分如何集成为一个整体以及通过何种机制实现整体的功能，这在设计网格时是必须考虑的问题。

2.1.2 讨论

网格体系结构，贯穿着两条主线，一是“分”，另外一个就是“合”。网格是一个整体的概念，网格体系结构的作用在一定程度上就是对网格的解剖。网格体系结构必须要能够标识

出网格的基本组成成分，要能够清楚地说明网格整体是由那些关键部分结合在一起形成的，但是这还远远不够，网格体系结构还必须能够对各个部分的功能、目的、特点等进行清晰地描述，使人们能够了解各个组成部分的作用。这些都是“分”的作用，在“分”的基础上，网格体系结构还需要进一步描述“合”起来的功能，即在充分了解网格的各个部分的作用机理、作用方式等的基础上，将这些部分按照一定的方式进行组织和集成，形成一个具有特定功能的整体对外提供服务。

网格体系结构，就是一个“分”与“合”的统一体，没有“分”，就无法深入的网格的内部去，没有“合”，就无法说明网格的整体特征。只有充分把握好“分”与“合”的这两方面的关系，才能够具体、深入、全面地把握好网格这一概念，才能够设计出真正实用、有效的网格体系结构。

到目前位置，比较重要的网格体系结构有两个，一个就是 Foster 等在早些时候提出的五层沙漏结构[19]，然后就是在以 IBM 为代表的工业界的影响下，在考虑到 Web 技术的发展与影响后，Foster 等结合 Web Service 提出的开放网格服务结构 OGSA（Open Grid Services Architecture）[20]。本章将重点介绍这两种结构。

2.2 五层沙漏结构

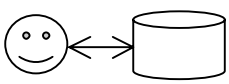
五层沙漏结构是一种影响十分广泛的结构，它的主要特点就是简单，主要侧重于定性的描述而不是具体的协议定义，因此很容易从整体上进行理解。下面分别从五层沙漏结构的基本思想、结构描述、与 Globus 的对比以及应用实例等几个方面进行介绍。

2.2.1 基本思想与概念

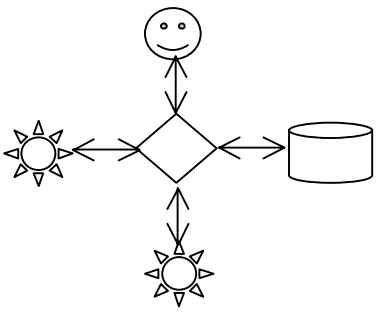
在五层沙漏结构[19]中，一个最重要的思想就是以“协议”为中心，也十分强调服务与API（Application Programming Interfaces）和SDK（Software Development Kits）的重要性。

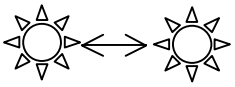
1 共享

首先介绍一下这里“共享”的含义。这里的共享不只是交换文件，而是更强调对计算机、软件、数据以及其它资源的直接访问。这种需求在工业、科学以及工程等许多领域中都会遇到。而且这种共享还必须是高度受控制的，需要在资源控制者和使用者之间小心地定义什么是可以共享的，那些人可以共享，在什么条件下可以共享。而“虚拟组织”就是基于这样的一些共享规则，由一些个人或者团体形成的集合体。



C/S 共享关系





P2P 共享关系

Proxy 共享关系

图 2-1 三种基本的共享关系

这里的共享关系又是十分灵活的，包括三种基本的形式，即客户端与服务器（C/S）的共享，端到端（P2P）的共享以及代理（Proxy）共享。

这里的共享是一种随时间变化的动态的共享，而不是静态的共享。网格具有动态性的特点，网格资源的共享也必然满足动态性的要求。在一些ASP/SSP的应用中，其共享是基于静态的配置建立起来的，因此缺乏灵活性，利用网格可以建立动态的计算与存储资源共享。而且这种共享不能够局限在某一机构或者单位之内，它是一种更广泛的共享，可以跨越不同的单位与组织的管理范围。当然这种共享也是跨越不同的地理位置的。

因此，在五层沙漏结构中的共享，是深层次、广泛、动态、具有多种形式的有条件受控制的共享。这是和我们以前所见到的共享形式有很大区别的。

2 互操作

另外一个重要的概念就是互操作，这里将共享定义为对各种资源的直接访问，也就是支持互操作。共享关系需要在任意的组织、团体之间在一开始就建立，可以动态增加新的成员，并且可以跨越不同的平台、语言和编程环境，在这样的情况下，如果不能够提供一种互操作机制，使得可以跨越不同的组织边界、使用策略以及资源类型，就不能够达到共享的目的。没有互操作机制的保证，动态虚拟组织的形成是不可能的，而且可以形成的虚拟组织的类型是非常有限的。

3 协议

而为了实现互操作就必须有相互遵守的协议。这里的协议是指为了实现特定的操作而定义的分布式系统元素之间交互的方式以及交互过程中交换的信息的结构（如图 2-2 所示）。它侧重于外部的（相互之间的）行为而不是内部的特征，这对定义网格体系结构很有实用价值。

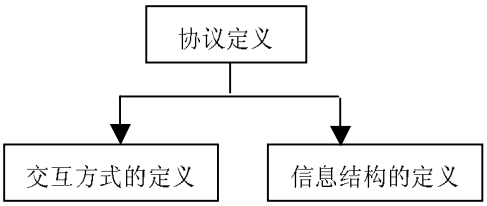


图 2-2 协议定义的两个方面

这里提出的五层网格结构首先是一个“协议结构”，通过协议可以实现一种机制，使得虚拟组织的用户与资源之间可以进行资源使用的协商，建立共享关系，并且可以进一步管理和开发新的共享关系。这一标准化的开放结构对网格的扩展性、互操作性、一致性以及代码共享都很有好处。

共享需要互操作，而实现互操作需要定义协议，因此五层沙漏结构中特别重视协议的定义。正如同Web通过提供统一的协议和语法来进行信息共享，从而引起了信息共享的革命那样，在网格中，也需要标准化的协议和语法用于通用的资源共享。

4 服务

服务是由它使用的协议和实现的行为定义的（如图 2-3所示），标准协议还使得定义标准服务更加容易。标准服务的定义（比如对计算的访问，存取数据，资源发现，协同调度，数据重复等）可以进一步提供增强的能力，使虚拟组织参加者得到更多的服务。由于这些服务抽象掉了与资源相关地细节，所以非常有利于虚拟组织应用的开发。

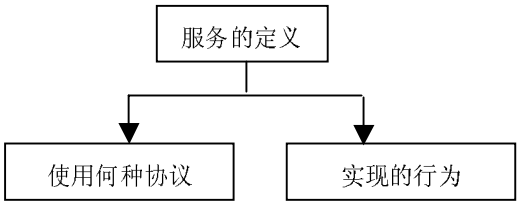


图 2-3 服务定义的两个方面

5 API/SDK

在五层沙漏结构中，同时还提供API（Application Programming Interfaces）和SDK（Software Development Kits），使得在建立网格应用时可以在抽象的基础上提高编程的级别。因为有更多的应用是针对虚拟组织的，而不是针对低级的互操作、协议或者服务。开发者要能够在复杂、动态执行的环境中开发高级的应用，借助于API、SDK就可以加速代码开发，实现代码共享，以及增强应用的移植性。API和SDK是附属于协议的，而不是协议的替代。

6 五层划分

五层沙漏结构并不提供严格的规范，它不是对全部所需协议的完整罗列，而是对该结构中各部分组件的通用要求进行定义，而且将这些组件形成一定的层次关系，每一层的组件具有相同的特征，上层组件可以在任何一个底层组件的基础之上建造。

在以上基本概念的基础上，下面介绍一下五层沙漏结构中“五层”的含义。五层沙漏结构根据该结构中各组成部分与共享资源的距离，将对共享资源进行操作、管理和使用的功能分散在五个不同的层次，越向下层就越接近于物理的共享资源，因此该层与特定资源相关的成分就比较多；越向上层就越感觉不到共享资源的细节特征，也就是说上层是更加抽象共享资源的表示，因此就不需要关心与底层资源相关的具体实现问题。

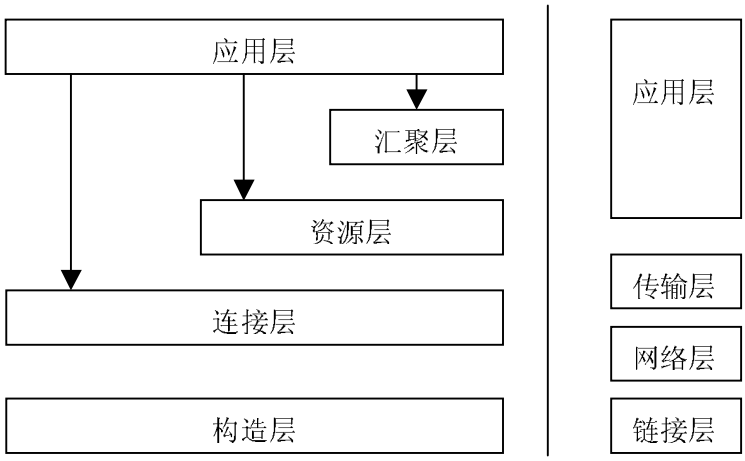


图 2-4五层结构及其与 TCP/IP 网络协议的对比

在五层沙漏结构中，最底层是构造层（fabric），构造层面对的是一个个具体的物理（也可以是逻辑的）资源，它通过对这些局部资源的管理，向上层提供对这些资源的管理和控制界面。构造层的上面是连接层（connectivity），主要是为下层的物理资源提供安全的数据通信能力，这是资源之间进行互操作的前提，连接层使得孤立的单个资源之间建立了联系。连接层的上面是资源层（resource），它反映的是抽象的局部资源的特征，而资源层上面的汇聚

层（collective）完成的功能是如何将下面以单个资源形式表现出来的资源集中起来，协调解决多个资源之间的问题。最上面的应用层（application）和资源的距离最远，它关心的是有什么样的资源可以由下面提供给虚拟组织，解决不同虚拟组织的具体问题。

为了便于理解，该结构还将这五层与广为使用的 TCP/IP 网络协议结构进行了粗略的对比。如图 2-4 [19]所示。

7 沙漏形状

五层结构的另外一个重要特点就是沙漏形状。其内在含义就是因为各部分协议的数量是不同的，对于其最核心的部分，要能够实现上层各种协议向核心协议的映射，同时实现核心协议向下层其它各种协议的映射，核心协议在所有支持网格计算的地点都应该得到支持，因此核心协议的数量不应该太多，这样核心协议就形成了协议层次结构中的一个瓶颈，在五层结构中，资源层和连接层共同组成这一核心的瓶颈部分。如图 2-5 [21]所示。

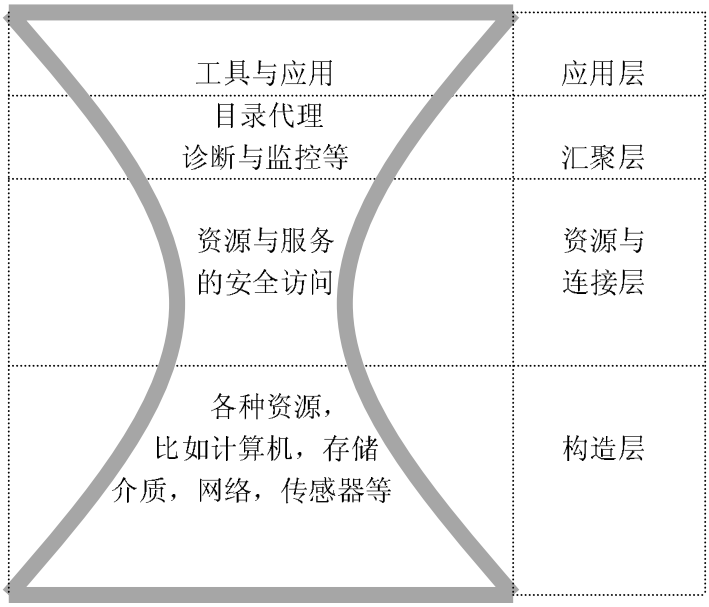


图 2-5沙漏形状的五层结构

沙漏核心的思想可以和微内核的操作系统进行类比，即操作系统只实现一些关键的基本功能，而把大量与特定设备有关和与应用有关的部分交给其它部分来完成。一个小的核心是有利于移植的，也可以比较容易地实现和得到支持。资源是多种多样地，应用需求更是复杂多变，因此定义好这样一个核心部分的意义是很大的。

2.2.2 结构描述

如前所述，五层沙漏结构的五层从下到上分别是构造层，连接层，资源层，汇聚层，以及应用层。下面对这五层的功能特点[19]分别进行描述。

1 构造层：局部控制的界面

网格构造层的基本功能就是控制局部的资源，向上提供访问这些资源的接口。构造层资源是非常广泛的，可以是计算资源，存储系统，目录，网络资源以及传感器等等。

这里需要说明的是构造层资源可以是一个比较复杂的系统，比如由多台微机通过系统级网络连接形成的机群系统，在机群系统的内部，为了实现通信和管理，必然有自身

的协议，这种协议是内部协议，和网格体系结构中资源之间的外部协议是不同的。其它的构造层实体比如分布式文件系统，或者分布式计算池等在实现过程中都可以有自己的内部协议。如图 2-6所示。

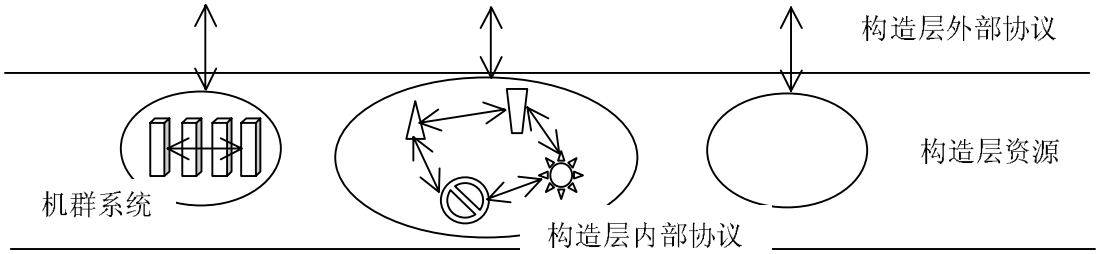


图 2-6 构造层资源的内部与外部协议

另外一点就是如果构造层资源提供的功能越丰富，则构造层资源可以支持的高级共享操作就越多，比如如果在资源层支持提前预留功能，则很容易在高层实现资源的协同调度服务，否则在高层实现这样的服务就会有较大的额外开销；如果构造层资源提供的功能较少，则网格结构的组织就可以比较简单，实现起来也就相对容易一点。

构造层应该实现的基本功能包括：查询机制（发现资源的结构和状态等信息）、控制服务质量的资源管理能力等。

表格 2-1 特定构造层资源及其功能特征

构造层资源举例	功能特征
计算资源	启动程序，监测和控制进程的执行，控制进程资源分配的管理机制，提前预留机制，查询功能。
存储资源	存放与获取文件的机制，第三方高性能传输方式，读写文件子集机制，以及远程数据选取与规约机制，对分配用于数据传输资源的控制管理机制，提前预约机制，查询功能。
网络资源	对网络传输资源的控制管理机制，查询功能（用来得到网络特征和负载）。
代码库	源代码与目标代码版本的管理机制，比如CVS控制系统。
目录	目录查询与更新操作机制，比如关系数据库。

2 连接层：支持便利安全的通信

连接层的基本功能就是实现相互的通信。它定义了核心的通信和认证协议（如图 2-7所示），用于网格的网络事务处理之中。

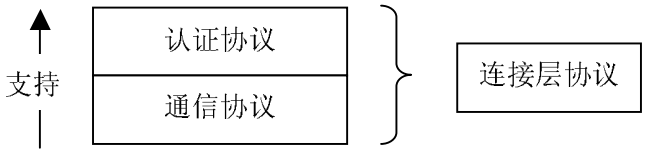


图 2-7 连接层协议组成与层次关系

通信协议允许在构造层资源之间交换数据，要求包括传输，路由，命名等功能。在实际中这些协议大部分是从TCP/IP协议栈中抽出的，比如网络层（IP、ICMP），传输层（TCP，UDP）和应用层（DNS）。建立在通信服务之上的认证协议提供加密的安全

机制，用于识别用户和资源。这里的安全也是一个十分复杂和重要的问题。

在网格环境中，连接层的安全认证需要解决的问题或者说应该具备的特征如表格 2-2 所示。

表格 2-2 连接层安全认证特征

特点	描述
单一登录点	用户只需要登录一次，就可以以该身份访问不同的构造层网格资源，不需要对不同的资源多次重复登录，也不需要用户的进一步介入。
代理	用户必须有让程序以自己身份运行的能力，因此程序就能够访问用户认证的不同资源。 该程序还能够有条件地将它的部分权限授予另一个程序（受限制的代理）。
与局部安全方法的集成	不同的资源可以使用其局部的安全方案，但是网格安全方案必须能够与那些局部的方案进行互操作。不要求网格安全方案完全替代局部安全方案，但是它必须能够实现向局部的映射。
基于用户的信任机制	用户可以使用多个提供者提供的资源，但并不要求资源提供者在安全环境中协同操作或者互操作。即如果一个用户有权使用地点A和B的资源，用户就能够将A和B资源结合起来使用，并不要求A和B的安全管理相互作用。

网格安全方案应该为通信保护提供各种灵活的支持，比如保护级别的控制，不可靠协议独立数据单元的保护，支持TCP之外的可靠传输等，允许网格使用者来控制认证策略等。

3 资源层：共享单一资源

资源层的主要功能就是实现对单个资源的共享。资源层建立在连接层的通信和认证协议之上，定义的协议包括安全初始化、监视、控制单个资源的共享操作、审计以及付费等。值得注意的是，资源层协议考虑的完全是单个的局部资源，因此忽略了全局状态和跨越分布资源集合的原子操作（这些问题是由汇聚层考虑的）。

表格 2-3 资源层的协议类型与描述

协议类型	描述
信息协议	得到资源的结构和状态信息，比如配置，当前负载，使用策略等。
管理协议	通过谈判访问共享资源，指出资源需求以及将执行的操作。初始化共享关系，保证要求的协议操作与底层共享资源提供的共享策略一致。还要考虑记帐和付费的问题，协议还可能需要具有监控操作的状态并控制某些操作的功能。

资源与连接协议形成了沙漏模型的瓶颈部分，因此这个协议集合要小，而且尽量标准化。这些协议要能够抓住含盖不同资源类型的基本共享机制，但是又不能够对高层协议的类型和性能有约束。

表格 2-1列出的构造层资源的功能是资源层协议应该支持的，但是对其中的许多操作，都可以加上特定的语义约束，并且当操作失败时具有可靠的错误报告功能。

4 汇聚层：协调各种资源

汇聚层的主要功能是协调“多种”资源的共享，而资源层的主要功能则是与“单个”资源的交互。汇聚层协议与服务（包括API/SDK）描述的是资源的共性（汇聚层实现的共同的功能和共享行为如表格 2-4所示），并不涉及资源的具体特征，说明不同资源集合之间是如何相互作用的。由于汇聚层建立在资源和连接层形成的协议瓶颈之上，因此不需要在资源上强加其它新的要求。

表格 2-4 汇聚层服务和协议

服务与功能名称	描述
目录服务	允许虚拟组织参加者发现存在的资源或者是存在资源的特性，允许用户根据名字或者属性来查询资源。
协同分配，调度以及代理服务	允许虚拟组织参加者申请分配一个或者更多的资源，并且在相应的资源上进行任务调度。
监控和诊断服务	用于监视虚拟组织资源的失败，恶意的攻击，入侵检查，过载等等。
数据复制服务	支持虚拟组织存储、网络与计算的管理，按照响应时间、可靠性、费用等标准优化数据访问的性能。
网格支持下的编程系统	可以在网格中提供熟悉的编程模型，使用不同的网格服务解决资源发现、安全、资源分配以及其它的问题。
负载管理系统与协同分配工作框架	提供描述、使用以及管理多步、异步以及多组件工作流。
软件发现服务	基于求解问题的参数发现和选择最好的软件实现与执行平台。
协作服务	用于潜在较大的用户社团内的协同交换信息，包括同步与异步两种方式。

表格 2-4展示了在实际中遇到的广泛而多样的汇聚层协议与服务。资源层协议必须是通用的而且可以广泛使用，汇聚层协议在资源层通用目的协议的基础上，实现更高级的应用，并能够面向特定的领域，这些功能或者协议有可能只在特定的虚拟组织中存在。

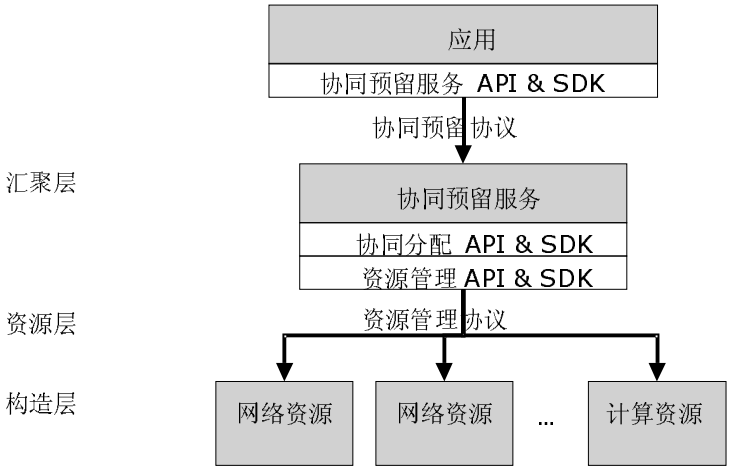


图 2-8 服务层、资源层协同工作示例

汇聚层功能可以作为永久的服务来进行实现，包括相关的协议，以及与应用相关联的API/SDK。图 2-8 [19]展示了如何将各层次的功能组织起来为应用提供支持与服务。汇聚层的协同分配API和SDK可以使用资源层的管理协议来操纵底层的资源，同时，在

此功能之上，可以定义协同预留协议并实现一个协同预留服务，它可以调用底层的协同分配API来实现协同分配操作，还可能提供附加的功能，比如授权，容错，以及日志等。这样，在应用层就可以使用协同预留服务来要求提供端到端的网络预留功能。

汇聚层组件可以通过裁减来满足特定的用户社团、虚拟组织或者应用领域的需求，比如，一个汇聚层SDK可以用来实现特定应用的协议，或者特定网络资源集合的协同预留服务，而另外的汇聚层组件可以实现更通用的目的，比如跨国界多组织的存储系统的复制服务，或者用于发现虚拟组织的目录服务等。一般地，用户社团的规模越大，基于标准化基础来建造汇聚层组件协议和API就越重要。

5 应用层

应用层是在虚拟组织环境中存在的。从应用程序员的观点看网格结构，应用是根据在任一层次上定义的服务来构造的（图 2-9 [19]）。在每一层，都定义了协议，以提供对相关服务的访问，这些服务包括资源管理，数据存取，资源发现等。 在每一层，可以将 API 定义为与执行特定活动的服务交换协议信息的具体实现。 这里的应用可以调用更高级的框架和库调用。这些框架自身可以定义协议、服务和 API，这里只是提出网格中要求的基本服务与协议。

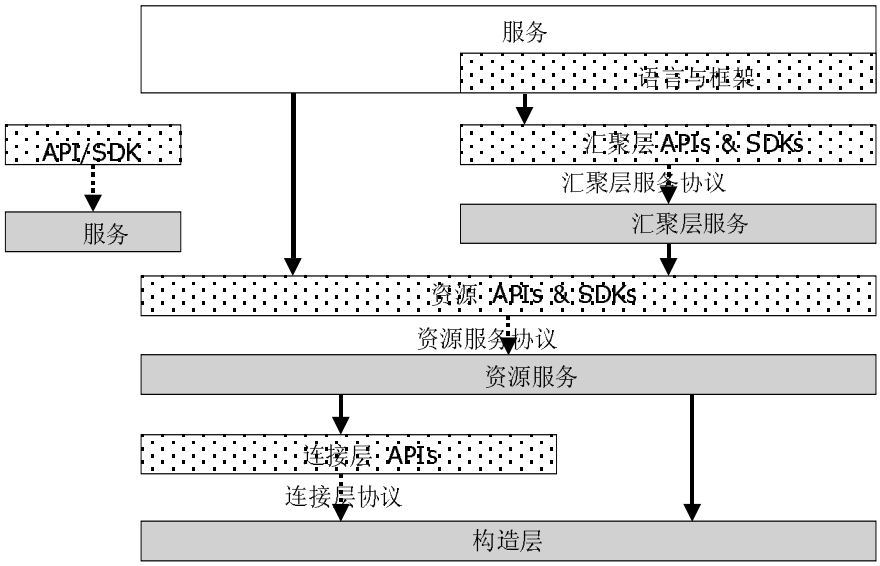


图 2-9 从应用程序员的观点看网格结构

2.2.3 与 Globus 的对应关系

为了加深对五层结构理解，该结构还将各个层次的功能与目前最重要的一种网格开发工具包 Globus Toolkit 各部分的功能进行了对应（如表格 2-5所示），这样读者可以对各个层次有一个具体的直观理解。

表格 2-5 五层结构各部分与 Globus 功能对应关系

五层结构	对应 Globus 的功能与特点
构造层	1 使用现有的构造层组件，包括卖方支持的协议与界面。 2 如果卖方不提供必要的构造层行为，Globus 将提供缺省的功能。 3 GARA（General-purpose Architecture for Reservation and Allocation），通过提供插槽管理，为不支持这一能力的资源提供实现资源预约的功能。
连接层	1 Globus 使用基于公钥的 GSI（Grid Security Infrastructure）协议用于认证、通信保护以及授权；支持与不同安全方案的集成，使用基于用户的信任机制；使用 X.509 格式的识别证书。 2 可以通过授权工具包来控制授权，它允许资源拥有者通过 GAA(Generic Authorization and Access) 控制界面集成局部的策略。
资源层	1 Globus中已被采用的协议包括 a)GRIP （A Grid Resource Information Protocol）， GRRP（Grid Resource Registration Protocol）。 b) 基于http的GRAM （Grid Resource Access and Management ）协议，用于分配计算资源并且监视和控制这些资源上的计算行为。 c)GridFTP 数据访问管理协议。 d)LDAP用于目录访问协议。 2 Globus 定义了客户端这些协议的 C 和 Java 的 APIs 与 SDKs，Globus 同时提供了服务端这些协议的 SDK 和服务的实现，比如 GRIS（Resource Information Service），GSS（Generic Security Services ）API。
汇聚层	1 元目录服务 MDS（Meta Directory Service），通过引入 GIIS（Grid Information Index Servers）来支持任意资源子集视口。 2 目录复制和复制管理服务用于支持网格环境中的数据集复制管理。 3 在线信任仓库服务为代理信任提供安全存储。 4 DUROC 协同分配库为资源协同分配提供 SDK 和 API。
应用层	无（因为 Globus 是用于支持网格应用和开发，它本身并不是网格应用）

2.2.4 基于五层结构的应用例子

建立网格体系结构的意义在哪里？下面通过具体的例子，给出较为直观的示例性说明。对于两个不同的应用，一个是多学科模拟，另外一个是光线追踪。在具体的应用人员开来，恐怕很难找到这两者的任何相同之处，但是按照体系结构的观点，表格 2-6 [19]给出了两者之间的关系。

注意这里将汇聚层进行了进一步的细分，分为通用汇聚层功能和面向特定问题的汇聚层功能。从通用汇聚层向下，这两个在高层看来互不相干的应用，却可以使用完全相同的基本功能。只是在面向具体问题的汇聚层这里，二者的功能才进行了进一步的细分。这说明了当我们建立好一个合理的体系结构之后，根据它来实现网格功能，可以充分实现相关功能的共享，不需要每次面对不同的问题都重新进行开发。

上面的例子说明的是不同的应用可以在统一的网格体系结构框架下使用相同的底层功能。下面的例子说明的是通过对标准化的网格体系结构各层功能的裁减和重组，也可以方便地实现不同的网格应用。

表格 2-6 五层结构应用例 1

应用层	多学科模拟	光线追踪
汇聚层（面向问题）	联合求解器， 分布式数据文档	检查点，作业管理，故障避免，分段运输（staging）
汇聚层（通用）	资源发现，资源代理，系统监视，社团授权，收回证书	
资源层	访问计算，访问数据，访问系统结构、状态与性能信息	
连接层	通信 (IP)，服务发现 (DNS)，认证，授权、代理	
构造层	存储系统，计算机，网络，代码库，目录	

表格 2-7五层结构应用例 2

应用层	高吞吐率系统
汇聚层（面向问题）	动态检查点，作业管理，故障避免，分段运输
汇聚层（通用）	代理，证书授权
资源层	数据访问，计算机访问，网络性能数据访问
连接层	通信，服务发现 (DNS)，认证，授权，代理
构造层	存储系统，调度

表格 2-8五层结构应用例 3

应用层	特定学科的数据网格应用
汇聚层（面向问题）	移植性控制，选择复制，任务管理，虚拟数据目录，虚拟数据代码目录
汇聚层（通用）	目录复制，复制管理，协同分配，证书授权，元数据目录
资源层	数据访问，计算机访问，网络性能数据访问
连接层	通信，服务发现 (DNS)，认证，授权，代理
构造层	存储系统，机群，网络，网络缓存

如表格 2-7 [21]和表格 2-8 [21]所示，对于高吞吐率系统和特定学科的数据网格应用，他们在构造层使用的资源都是不同的，但是在连接层和资源层却使用了完全相同的服务，这充分说明了沙漏的存在，即在这一瓶颈部分汇集了所有的应用都需要的少数功能，但是在汇聚层其功能又分散开来。这个例子说明了上层不同功能向瓶颈部分的映射，瓶颈部分又向下层不同的具体资源映射的过程（如图 2-10所示）。这个例子也说明了如果能够定义好合理的体系结构，对于具体的应用，在这一框架下通过裁减，就可以方便地实现各自的具体功能。

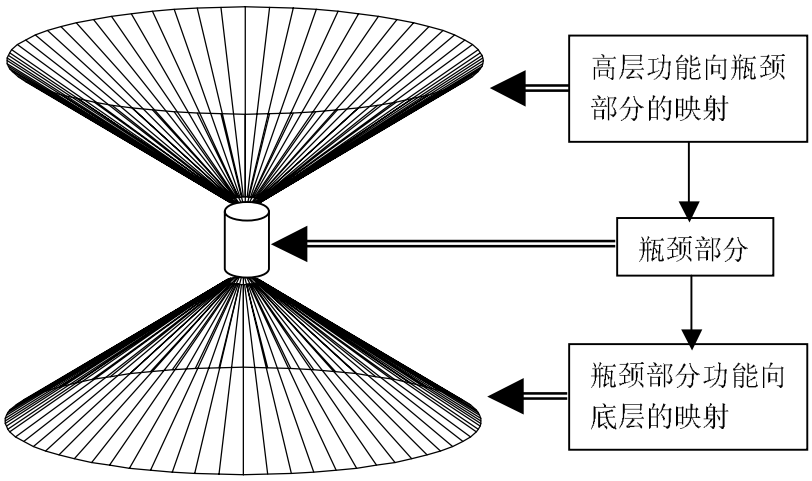


图 2-10 五层沙漏结构功能映射示意图

2.3 开放网格服务体系结构

开放网格服务结构 OGSA(Open Grid Services Architecture)[60][61]是 Global Grid Forum 4 的重要标准建议[62]，是继五层沙漏结构之后最重要，也是目前最新的一种网格体系结构，被称为是下一代的网格结构。

2.3.1 基本思想

1 以服务为中心的模型

如果说五层沙漏结构是以协议为中心的“协议结构”，则 OGSA 就是以服务为中心的“服务结构”。

这里的服务是指具有特定功能的网络化实体。在五层沙漏结构中，强调的是被共享的物理资源（或者是这些资源所支持的服务），在 OGSA 中，服务所指的概念更广，包括各种计算资源、存储资源、网络、程序、数据库等等，简而言之，一切都是服务。五层模型试图实现的是对资源的共享，而在 OGSA 中，实现的将是对服务的共享。从资源到服务，这种抽象，将资源、信息、数据等统一起来，十分有利于灵活的、一致的、动态的共享机制的实现，使得分布式系统管理有了标准的接口和行为。

为了使服务的思想更加明确和具体，OGSA定义了“网格服务”（Grid Service）的概念。网格服务是一种Web Service[9]，该服务提供了一组接口，这些接口的定义明确并且遵守特定的惯例，解决服务发现、动态服务创建、生命周期管理、通知等问题。在OGSA中，将一切都看作是网格服务，因此网格就是可扩展的网格服务的集合，即网格＝{网格服务}。网格服务可以以不同的方式聚集起来满足虚拟组织的需要，虚拟组织自身也可以部分地根据它们操作和共享的服务来定义。

简单地说，网格服务＝接口/行为＋服务数据。图 2-11 [59]是对网格服务的简单描述。

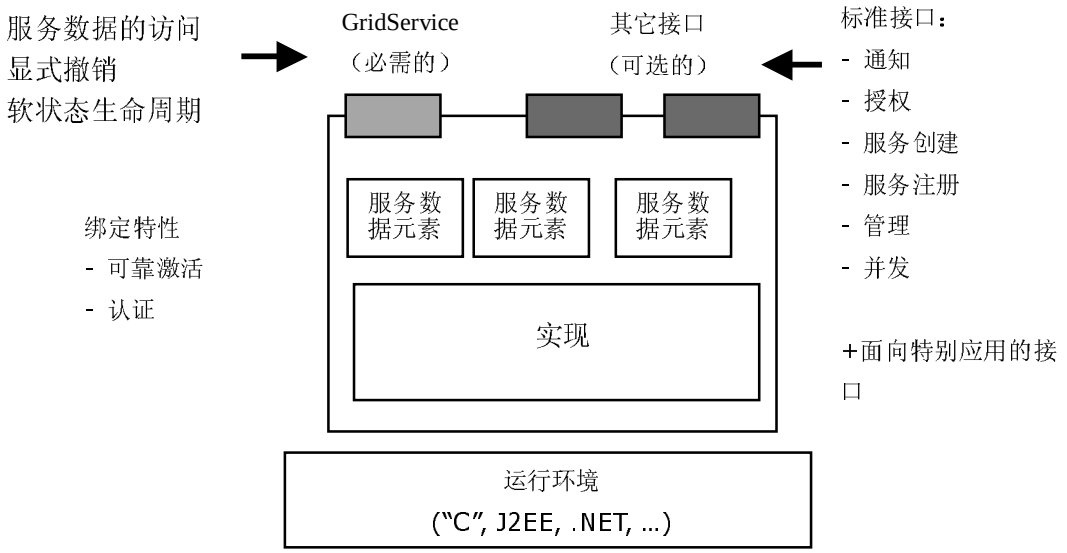


图 2-11 网格服务示意图

与五层模型一样，在 OGSA 中也非常重视互操作性，但是从服务的观点，OGSA 将互

操作性问题转化为两个子问题，即定义服务的接口和识别激活特定接口的协议。五层模型是按照支持虚拟组织组件互操作的协议要求来构造的，更多的体现出一种解剖学的特点，而OGSA 强调的是与协议消息相对应的服务，侧重于实体表现出来的行为特征，即特定对象的生理机能。

以网格服务为中心的模型具有如下好处[60]：1 由于网格环境中所有的组件都是虚拟的（这里的具体含义是指对相同接口不同实现的封装），因此，通过提供一组相对统一的核心接口，所有的网格服务都基于这些接口实现，就可以很容易地构造出具有层次结构的、更高级别的服务（如图 2-12所示），这些服务可以跨越不同的抽象层次，以一种统一的方式来看待。2 虚拟化也使得将多个逻辑资源实例映射到相同的物理资源上成为可能，在对服务进行组合时不必考虑具体的实现，可以以底层资源组成为基础，在虚拟组织中进行资源管理。通过网格服务的虚拟化，可以将通用的服务语义和行为，无缝地映射到本地平台的基础设施之上。

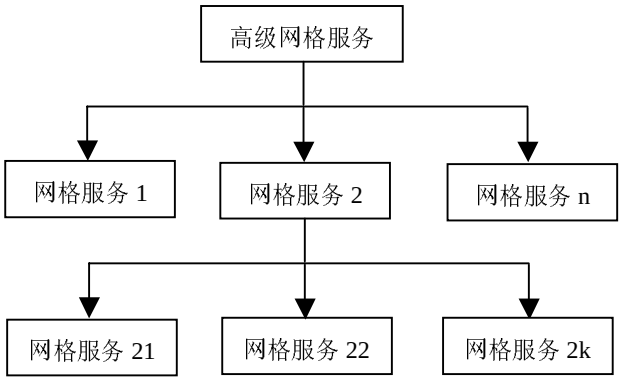


图 2-12 OGSA 的服务结构示意图

2 统一的 Web Service 框架

Web Service 的定义是这样的[9]：一个 Web Service 就是一个可以被 URI 识别的软件应用，它的接口和绑定可以被 XML（eXtensible Markup Language）描述与发现，并且可以通过基于 Internet 的协议直接支持与其它基于 XML 消息的软件应用的交互。 Web Service 在 W3C 中由三个工作组和一个协调组组成，这三个工作组分别是 Web Service 结构工作组，XML 协议工作组，Web Service 描述工作组。

Web Service 标准正在 W3C 内部以及其它的标准体内部被定义，他们形成了新的主要工业提议的基础，比如 Microsoft 的.NET, IBM 的 Dynamic eBusiness，Sun 的 Sun One，三个与网格服务有关的标准分别是 SOAP（Simple Object Access Protocol），WSDL（Web Services Description Language）和 UDDI（Universal Description Discovery and Integration）。

Web Service 描述了一种新出现的、重要的分布式计算范式，和 DCE，CORBA，JAVA RMI 等方法不同，它更强调基于单个 INTERNET 标准（XML）来解决异构分布计算的问题。Web Service 定义了一种技术，用于描述被访问的软件组件、访问组件的方法以及找到相关服务提供者的发现方法，Web Service 并不倾向于特定的编程语言、编程模型以及系统软件。

OGSA 是符合标准的 Web Service 框架的。Web Service 解决了发现和激发永久服务的问题，但是在网格中，大量的是临时服务，因此 OGSA 对 Web Service 进行了扩展，提出了网格服务（Grid Service）的概念，使得它可以支持临时服务实例，并且能够动态创建和删除。表格 2-9 [60]列出了网格服务的接口，其中只有 GridService 接口是必需的，而其它的接口都是可选的。网格服务是由他们提供的能力来刻画的。一个网格服务实现一个到多个接口，每一个接口定义了一些操作，这些操作通过交换定义好的一系列消息来激活。

网格服务接口和 WSDL 的 portTypes 相对应，网格服务提供 portTypes 的集合，包括一些与版本有关的附加信息，在网格服务中用 serviceType 来描述，serviceType 是 OGSA 定义的 WSDL 的扩展元素。

表格 2-9 网格服务的接口

PortType	操作	描述
GridService	FindServiceData	查询网格服务实例的各种信息，包括一些基本的内部信息，大量的关于每个接口的信息以及与特定服务有关的信息。
	SetTerminationTime	设置并得到网格服务实例的终止时间
	Destroy	终止网格服务实例
NotificationSource	SubscribeToNotificationTopic	根据感兴趣的消息类型和内容说明，向相关事件的通知发送者进行登记。
	UnSubscribeToNotificationTopic	取消登记
NotificationSink	DeliverNotification	异步发送消息
Registry	RegisterService	网格服务句柄的软状态注册
	UnRegisterService	取消注册的网格服务句柄
Factory	CreateService	创建新的网格服务实例
PrimaryKey	FindByPrimaryKey	返回根据特定键值创建的网格服务句柄
	DestroyByPrimaryKey	撤销特定键值创建的网格服务实例
HandleMap	FindByHandle	返回与网格服务句柄相联系的网格服务实例

由于 OGSA 采用统一的 Web Service 框架，因此很自然就具备了原来 Web Service 的所有有利因素，比如服务描述和发现；可以从服务描述中自动产生客户与服务端的代码；将服务描述和互操作的网络协议绑定在一起；和新出现的高级开放标准、服务和工具兼容；有广泛的工商业支持等等。

Web Service 框架有如下好处[60]：1 网格环境需要支持服务的动态发现和组织，在异构的动态环境里，这就需要一些必须的机制，用于注册和发现接口的定义和端点实现的描述，以及基于特定的接口绑定动态产生代理。WSDL 提供的标准机制支持这种要求，可以将接口定义和特定绑定的实现分开。2 广泛接受的 Web Service 机制意味着基于 Web Service 的框架可以开发大量的工具和服务，比如可以对不同的语言产生语言绑定的 WSDL 处理器，在 WSDL 之上的工作流系统，Web Service 的主机环境等。使用 Web Service 并不意味着在所有的通信中都必须使用 SOAP，如果需要，可以使用替代的传输方法，以赢得更高的性能或者在特定的网络协议上运行。

3 突破科技应用领域

企业以前的计算是在高度集成的企业计算中心进行，但是 Internet 的增长和电子商务的出现，使得大家更认识到企业的 IT 基础设施同时包括外部的网络、资源和服务。这样高度集成的企业内部 IT 基础设施就分解为异构和分离的系统。企业必须再重新集成那些分布的服务和数据资源，解决导航、分布安全等问题。

另一个重要的 IT 工业倾向就是不同企业间的 B2B 合作，比如多个组织支持的链式管理，虚拟 WEB 商业街，电子市场拍卖。B2B 关系是一种虚拟组织，只不过对安全，审计，可用

性，服务级协议以及复杂事务处理流有严格的要求。这样 B2B 计算就代表了另一种对分布式系统集成的要求。

网格概念和技术开始是用于科学协作中的资源共享，后来人们发现网格概念对于商业计算来说也是特别重要的，不仅可以作为一种增强企业能力的方式，而且是一种构造可靠、可扩展、安全的分布系统的重要解决方案。

正如 Web 技术一开始是为科学协作而出现的，但是后来在商业领域却大量使用一样，OGSA 将原来主要在科技领域应用的网格技术转移到工商业领域。

OGSA 的重点是商业应用而不是象以前那样更侧重于科学与技术应用，但是其原则和机制可以同时适用于两种环境。但是在商业的应用中需要无缝地和已有的服务与资源，以及负载、安全、网络 QOS、可用管理工具等的集成。OGSA 支持服务发现的特性方便了将高级网格服务功能向原始平台设施的映射与应用。OGSA 面向服务的特点允许我们在不同的层次上虚拟化资源，因此相同的机制与抽象可以应用于多个组织之间的分布式网格支持的协作，或者是跨越多个结点的主机环境。

2.3.2 OGSA 的两大支撑技术

建造 OGSA 的两大支撑技术，网格技术（即 Globus 软件包）和 Web Service。Globus 是已经被科学与工程计算广泛接受的网格技术求解方案，Web Service 是一种标准的存取网络应用的框架。

1 Globus

Globus 是一种基于社团的，开放结构，开发源码的服务的集合，也是支持网格与网格应用的软件库，该工具包解决了安全，信息发现，资源管理，数据管理，通信，错误检测以及可移植等问题。Globus 工具包在世界上的许多网格项目，包括几百个地点被使用。关于 Globus 的详细介绍请见本书的后续部分。

和 OGSA 关系密切的 Globus 组件是 GRAM 网格资源分配与管理协议和门卫服务，他们提供了安全可靠的服务创建和管理功能，元目录服务通过软状态注册、数据模型以及局部注册来提供信息发现功能，GSI 支持单一登录点、代理和信任映射。这些功能提供了面向服务结构的必要元素，但是比 OGSA 中的通用性要小。

2 Web Service

关于 XML 协议方面的工作是 Web Service 的基础。由于 XML 在分布式应用之间被广泛用于作为信息交换的方式，在 2000 年 9 月 Web Service 的 XML 工作组成立。

Web Service 中几个比较重要的协议标准是 SOAP（Simple Object Access Protocol），WSDL（Web Services Description Language），WS-Inspection，UDDI（Universal Description., Discovery, & Integration）。

SOAP 是基于 XML 的 RPC（Remote Process Call）协议，用于描述通用的 WSDL 目标。通过将 SOAP 进行扩展，比如数字签名，加密等支持 Web Services 框架的安全性。

WSDL 用于描述服务，包括接口和访问的方法。复杂的服务可以由几个服务组成。它是 Web Services 的接口定义语言。

WS-Inspection 给出了一种定义服务描述的惯例，包括一种简单的 XML 语言和相关的管理，用于定位服务提供者公布的服务。UDDI（Universal Description Discovery and Integration）定义了 Web Services 的目录结构。

3 网格计算与 Web Service 的关系

网格项目 AVAKI 认为网格计算和 Web Service 之间存在着密切的联系，并且进行了如下归纳[46]：Web Service 的核心是在大的异构网络上将各种应用连接起来，借助于 Web 标准 UDDI、 WSDL 和 XML/SOAP 等将 Internet 从一个通信网络进一步发展到一个应用平台。当越来越多的 Web Service 实现后，应用的各种特征将会随着改变。一些应用就可以根据需要或者是根据可以得到的功能，从可得的服务中来动态构造，比如，可以动态地将一个新的服务加入到基因组分析或者金融市场模拟的分析功能或者服务库中。

一旦 Web Service 在更大的范围内得以实现，应用连接标准的制定就会成为一个突出问题。由于 Web Service 一开始就设计为在整个异构网络上工作，当前标准还没有考虑其它方面的复杂性，比如网格已经解决的不同操作系统之间的通信问题，访问基于不同文件系统的文件等，因此 Web Service 自然地需要下层网格软件提供的服务。

现在需要的是高级的协议：（1)在 XML/SOAP 之上可以运行其它的协议比如 JXTA[10]；（2）可以突破 DNS 限制的命名和绑定方式；（3）实现可扩展的命名、相互的安全认证、位置透明、以及透明迁移四个关键的能力，它可以使 Web Service 可扩展、更安全、更可靠，而且具有高性能，网格软件的设计要满足这些要求，这样网格软件也将成为 Web Service 的很好补充。

2.3.3 服务接口与功能机制

OGSA不同的功能是由不同网格服务的接口实现的，下面介绍这些接口与惯例以及相应的机制。

1 必需的服务接口 **GridService**

GridService是OGSA 服务接口中唯一必需的服务，下面分别从服务的生命周期管理和服务数据管理等方面来对这一接口功能进行介绍。

由于OGSA中主要是临时服务，临时服务必然会带来服务生命周期的问题，也就是说需要决定什么时候服务应该终止，并同时释放相关的资源。

在正常操作情况下，一个临时服务实例的创建是为了完成特定的任务，可以在任务结束时终止，也可以显式地由请求方或者请求方指定的特定服务来终止。但是通过显式请求终止特定的服务是有问题的，因为在分布式系统中，个别组件出现故障的情况是无法避免的，这时可能引起消息的丢失，如果让服务终止的消息请求丢失，该服务就永远看不到对其终止的要求，其后果就是该服务永久地占用相关的资源，造成资源的浪费。还有一种情况就是由于某种原因希望一个服务的生命周期能够进一步扩展。那么OGSA是如何实现既避免资源浪费，又能够根据需要不断扩展服务的生命周期呢？

OGSA引入了软状态（soft state），网格服务通过维护一个内部状态来管理服务的生命周期，该状态将一个服务实例与另一个具有相同接口的服务实例区别开来。同时为了解决这一问题，又定义了两个标准操作：Destroy和SetTerminationTime，它们用来显式地撤销和终止网格服务实例的生命周期（软状态协议通过不断收到“keepalive”消息，不断进行状态刷新来维持存活）。

下面结合图 2-13，来看一下OGSA对服务生命周期的管理过程。

首先在A点，在网格实例创建时，被创建的实例获得一个特定的生命周期，图中所示在E点该服务终止。最初的生命周期可以由客户端来明确提出，或者由另外一个服务代表该客户端来申请。通过在网格服务接口中的SetTerminationTime操作，可以实现软状态生命周期管理。它定义了如何通过谈判为一个新的服务实例获取初始的生命周期，如何扩展服务实例的生命周期，以及当生命周期到达时如何处理服务实例的方法。如何

通过谈判获取初始生命周期？当通过Factory创建一个新的网格服务实例时，客户端指出最小和最大的可以接受的生命周期时间，Factory选择一个初始的生命周期并且将它返回给客户端。

如果以后对该服务的生命周期不再有任何改变，则到达E时间点后，不管该服务是否收到终止服务的消息，都会结束其自身的运行。这样的机制所带来的好处就是：1）即使是系统的某些部分出现故障，也不妨碍该服务占用资源的回收，不会造成资源浪费；2）客户端可以明确知道该服务的存活时间，因此可以决定何时需要对其扩展生命周期，即使是和服务失去联系但是也可以知道它何时终止，避免由于客户端对服务状态的无知而造成各种无用的工作。

然后在D时间点，客户端知道服务将会在E时间点终止，但是希望该服务能够继续存活，这时客户端可以通过向服务发送keepalive消息使得该服务能够继续存活，比如新的终止时间为时间点F。keepalive消息的发送周期可以根据客户端与服务实例的最初生命周期协商的结果以及网络可靠性来决定。时间间隔的大小可以根据当前消息是否畅通以及附加开销来加以平衡。另外，通过客户端也可以通过SetTerminationTime发送消息给网格服务实例来扩展其生命周期的时间，它指出了最大和最小可以接受的新生命周期的时间，服务实例选择一个新的生命周期并且返回给客户端。这和一个keepalive消息在效果上是等价的。

当然服务的存活期也可以被更改，比如在时间点B，可以通过SetTerminationTime来提前终止该服务的生命周期，使它在C时间点结束。

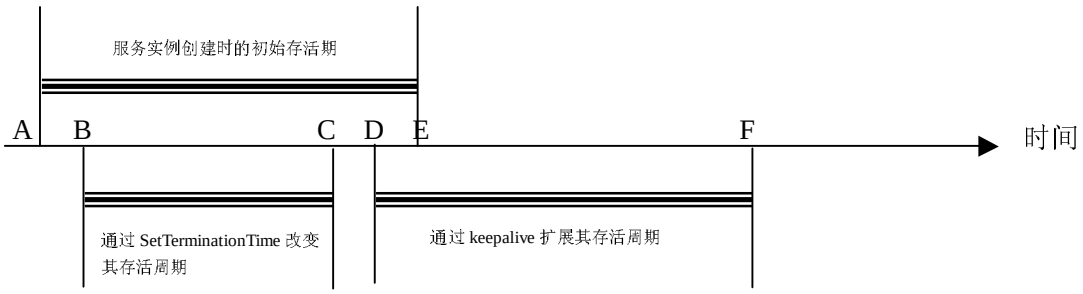


图 2-13 服务生命周期变化

这种生命周期管理方法使服务有相当大的自主性。对于来自客户端的生命周期扩展需求，服务可以进一步应用自身的某些策略。一个服务可以自己决定在任何时间扩展其生命周期，比如可以是根据客户端的请求来进行扩展，或者是其它的原因进行扩展。一个服务实例也可以在任何时候取消它自身，比如由于资源限制和优先级的原因使得它不得不放弃它的资源。服务被取消以后，客户端再要求引用这一服务是就会失败。

每一个网格服务实例都有一些服务数据与之联系，这些服务数据是一些被封装的XML元素的集合。每一数据元素包括一个对于网格服务实例来说是独一无二的名字，一个类型，还有一个用于生命周期管理的存活时间信息。

网格服务规范为每一个网格服务接口定义了0个或者多个服务数据元素，只要是支持该接口的服务，其服务实例就必须支持这些服务数据元素。

一个应用是如何发现它能够使用的服务并且知道这些服务的特征（服务数据）呢？这就需要提供一种发现机制，这一机制的建立问题可以通过如下3个方面的定义来解决：

- 关于服务数据的标准表示
- 在所有服务都必须具有的 GridService 接口内定义一个标准的 WSDL 操作 FindServiceData，用于查询和检索服务数据。

- 用注册服务来管理关于网格服务实例信息的标准接口，用于从句柄到引用的映射。

服务发现定义为根据GSH的特定属性，从一个大的GSH集合中得到GSH子集的过程。这些属性可以是所提供的接口，被满足的服务请求的个数，服务的负载，或者是声明的某种策略（比如允许未响应的请求的个数）。GridService接口FindServiceData操作的一个具体应用就是服务发现。

2 句柄映射 HandleMap

网格服务是有状态的，并且可以动态地创建和撤销，这样就需要通过一种方式来将一个动态创建的服务与另外一个服务区别开来。因此，每一个网格服务都被赋以一个全局唯一的名字，即网格服务句柄 (GSH，Grid Service Handle)，它将网格服务实例区别开来。

由于网格服务可以在生命周期内被升级（比如增加新的协议版本或者是增加可替代的协议），这样GSH就不能携带与特定协议或者实例相关的信息，比如网络地址，支持的协议绑定等。OGSA所采取的方法就是将这些信息封装起来，和其它与特定实例相关的信息一起，形成一个称为网格服务引用（GSR，Grid Service Reference）的抽象实体。不象GSH，网格服务实例的GSR可以在该服务的生命周期内改变，每一个GSR都有一个显式的存活期限，通过OGSA定义的映射机制，可以得到一个更新后的GSR。

有了GSR，就可能实现对网格服务的访问，但是并不都会成功，如表格 2-10所示的几种情形。

表格 2-10 使用 GSR 访问服务时可能出现的异常情况

GSR	访问结果
服务过期	不定义
禁止服务请求	无法访问
服务出现故障	

由于在OGSA中GSH和GSR是分开的，那么如何通过GSH得到GSR呢？如图 2-14所示，针对不同的GSH，OGSA给出了两种不同的方法。首先，OGSA定义一个引用句柄映射接口（HandleMap）。这一接口提供的操作就是给出一个GSH返回一个有效的GSR。这里需要说明的是，由于可以被访问的映射操作是受到控制，因此映射请求可能会被拒绝。一个HandleMap接口实现应该能够跟踪真正存在地网格服务实例，不要返回已经终止的实例的引用。

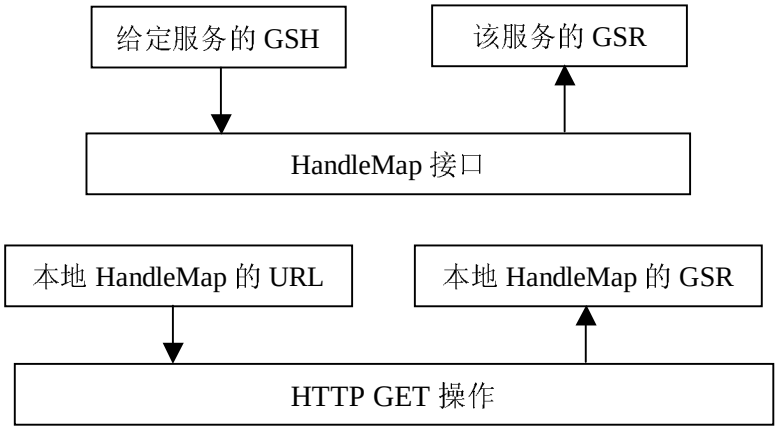


图 2-14 如何获取 GSR

其次，需要一种不需要通过映射接口就可以获得GSR的方法，即获取本地HandleMap的GSR的方法。本地HandleMap接口是指为了保证能够将一个GSH映射到GSR，OGSA要求每一个网格服务实例至少有一个HandleMap接口被注册，这个接口叫作本地HandleMap接口。因为实现HandleMap接口的服务必然也是一个网格服务（因为OGSA中一切都是网格服务），因此它也需要一个GSH。如果采用和前面相同的方法来获得GSR，就陷入了一个无限递归的过程，为了解决这一问题，就需要一种不需要HandleMap就可以得到GSR的方法。为达到这一目的，要求所有的本地HandleMap服务都能够被一个URL识别，并且支持一个自举（bootstrapping）操作，该操作只需要支持一个简单的众所周知的协议（比如HTTP/HTTPS）。这样，只需用一个HTTP GET操作作用于一个指向本地HandleMap的URL，相应HandleMap的GSR便以WSDL的形式返回。

3 注册服务 **Register**

支持服务发现的网格服务叫作注册（Registry）。一个注册服务根据两个东西来定义：一是注册接口，它提供GSH的注册操作，二是相关的服务数据元素，它包括注册的GSH的信息。可见，Registry接口主要是用于注册一个GSH，而GridService接口的FindServiceData操作用于检索已注册的GSH的消息。

一个服务可以将它的存在以及它所提供的服务通过Registry操作通知给其它感兴趣的团体。这些感兴趣的团体主要是不同形式的发现服务，它们收集服务信息并且将这些信息结构化，用于有效地响应发现服务的请求。GSH注册是一个软状态操作，必须进行周期性地刷新以保持其存活，它带来的一个很自然的结果就是发现服务可以处理动态的服务。

与一个GSH相联系的属性和该GSH向服务的注册是分开的。这一特征很重要，因为属性值可能是动态的，而且有多种获取属性值的方式。

4 创建临时服务 **Factory**

OGSA更强调的是临时服务，而不是象Web Service那样的永久服务。那么一个临时服务是如何从无到有的呢？OGSA定义了一类网格服务，专门用于实现创建新网格服务实例的接口，他们被称为Factory接口，实现这一接口的服务就是Factory。

Factory接口的CreateService操作可以根据请求，创建一个网格服务，并且返回新创建服务实例的GSH和初始的GSR（如图 2-15所示）。

Factory接口并不规定服务实例是如何创建的。一种常见的方案就是在具体的运行环境（比如.NET或者J2EE）中实现Factory接口，根据运行环境提供的标准机制来创建（并且管理）新的服务实例。在运行环境中可以详细定义服务是如何实现的（比如用何种语言），但是如何实现的细节对OGSA的服务请求者是透明的，服务请求者只能看到Factory接口。另外，可以通过对其它Factory服务的代理请求来构造更高级别的Factory。

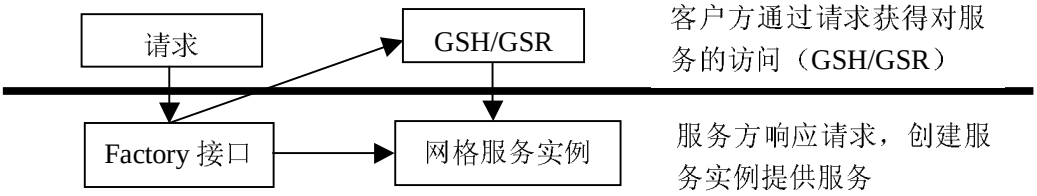


图 2-15 服务实例的创建

5 主键服务 **PrimaryKey**

利用主键查询是指当使用一个Factory时，如果这个Factory支持网格服务实例发现，而且这个网格服务实例就是通过这个Factory创建的，则客户端可以指定一个网格服务主键给Factory作为创建请求的一部分，这时客户端可以用这个键查询特定网格服务实例的Factory。

6 通知机制 **NotificationSource/NotificationSink** 接口

动态分布的服务必须能够将他们状态的改变通知对方。OGSA将消息的发布方服务接口称为NotificationSource，而将消息的接收方服务接口称为NotificationSink，通过这两个接口实现通知机制，以便于以一种标准的方式来处理通知。

通知机制允许客户端通过注册来获取特定的消息，并且支持消息的异步单向传输。如果一个服务愿意向外提供消息通知，它必须支持NotificationSource接口，对消息的订阅行为进行管理。一个服务如果愿意接收消息通知，它必须实现NotificationSink接口，通过该接口可以获取通知的消息。

如图 2-16所示通知从源到目的传递过程。首先第一步是消息的接收方用自己的GSH在通知源接口上激活订阅操作，然后第二步就是通知消息流就从“源”发向“目”，但仅仅这样这还不行，为了能够持续获得通知，就需要第三步即“目”周期性地给源发送保持存活消息，让“源”知道它仍然对接受通知感兴趣。

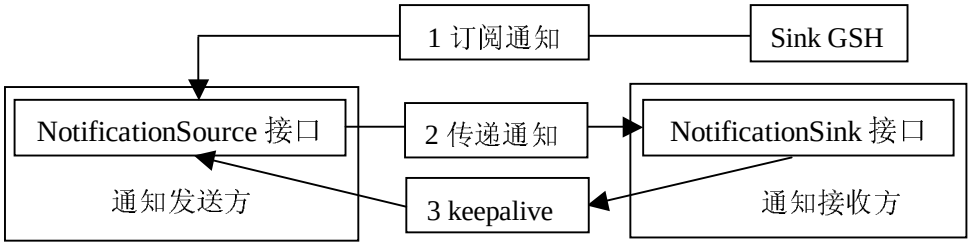


图 2-16 通知传送的过程

7 升级的管理与其它

在一个复杂分布系统中的服务可以被独立地升级。因此，要对服务之间的版本和兼容性要进行管理并且要能够表示出来，这样客户端不仅可以发现特定的服务版本，而且可以发现兼容的服务。进一步，服务（包括服务运行的主机环境）的升级不能够对客户端的操作有影响。这里，OGSA是通过定义惯例，使得我们可以识别什么时候服务改变了，那些改变是否在接口和语义上向后兼容。OGSA也定义了客户端服务信息刷新的机制。

服务的描述指明了可以和服务进行通信的协议绑定，在这样的绑定中最好能够具有可靠的服务激活以及认证这两种特性。

为了支持网格服务的发现和改变管理，网格服务接口必须是全局唯一命名的。在 WSDL 中，接口用 portType 定义，它是全局唯一的，用 portType 的 qname 命名。对网格服务定义的任何改变，不管是改变其接口，或者是操作语义的显著变化，一定要通过新的接口名字反映出来（比如新的 portTypes）。

OGSA还有待于进一步扩展，支持更多的功能，比如以后可以定义可选的Manageability接口，支持一些管理操作，通过这些操作，就为从管理控制台或者用自动控制工具对大量的网格服务实例进行监视和管理提供了支持。

2.3.4 网络协议绑定

Web Service框架可以在不同的协议绑定上进行实例化。SOAP+HTTP用TLS解决安全问题就是一个例子，当然也可以定义其它的方式。这里讨论一个OGSA上下文中的问题。

在OGSA上下文中选择网络协议的绑定，需要满足四个要求：

（1）可靠传输，网格服务抽象要求支持可靠的服务激活，解决这一问题的一种方式就是在网络协议绑定中结合进相应的支持。（2）认证与代理，网格服务抽象要求支持对远程地点的通信，该通信是基于代理信任的 。（3）普遍性，由于网格要求支持从分布式资源中动态形成虚拟组织，这就意味着任何一对服务之间都可以相互作用。（4）GSR格式，网格服务引用可以采用特别的绑定格式，一种可能的GSR格式就是WSDL文档。

定义少量用于网格服务发现和激活的标准协议绑定，将会对成功使用大粒度、互操作的OGSA实现有好处。正如同普遍使用Internet协议允许任何两个实体通信，广泛地使用网格间协议将会允许任何两个服务进行通信，这样客户端可以非常简单，因为它们只需要知道一些协议的集合。但是这样的网格间协议是否能够定义出来以及是否能够获得广泛的接受还是一个问题。

2.3.5 高级服务

抽象和基本的服务是建造不同高级网格服务的基本块。定义和实现这些不同的基本服务块，可以解决电子商务和电子科学应用中不同的问题。这些问题包括：

分布数据管理服务，支持访问和操纵分布式数据，这些数据可以是在数据库或者是文件中。一些重要的服务包括：数据库访问，数据传输，复制管理，地点复制，事务处理等等；工作流服务，支持在多个分布式网格资源上协同执行多个应用任务；审计服务，支持对数据的记录，数据的安全存储，用于防止欺诈和入侵检查等等；仪器使用和监控服务，支持在分布式环境中发现传感器，收集和分析传感器的信息，当异常情况出现时产生警报等等；分布式计算的问题测定服务，包括信息转储（dump），跟踪，具有事件标记和关联能力的日志机制等；安全协议映射服务，该服务能够将分布式安全协议透明地映射到本地平台的安全服务上，支持分布式安全认证和访问控制机制。

本框架的灵活性意味着这样的服务可以有不同方式的实现和组成。比如，支持同时分配和使用多个计算资源的协同服务可以作为一个服务实例被初始化，该服务也可以和其它的应用进行连接，或者结合到更高级的服务中。

2.3.6 运行环境的作用

OGSA定义了网格服务实例的语义：即如何创建，如何命名，如何决定生命周期的结束，如何进行相互通信等等。但是OGSA对服务做什么以及该服务如何执行并不做要求。换句话说，OGSA并不解决关于如何实现编程模型、编程语言、实现工具，执行环境等问题。

实际上，网格服务是在特定的运行执行环境或者主机环境中被实例化的。特定的运行环境不仅定义了编程模型、编程语言、开发工具以及调试工具的实现，而且定义了一个网格服务的实现如何满足网格服务语义的要求。

现在的网格应用特别依赖于作为他们的运行环境的本地操作系统进程，比如创建新的服务实例经常涉及到创建一个新的进程。在这样的环境中，服务自身可以以不同的语言实现，比如C, C++, Java, 或者Fortran。网格语义可以直接作为服务的一部分进行实现，或者通过连接特定的“库”来实现。典型的语义不是通过外部服务提供的，操作系统提供的功能除外。这样，生命周期管理功能必须在应用内部来解决。

从另一个方面，Web Service可以在更复杂的容器或者基于组件的运行环境中实现，比如J2EE, WebSphere, .NET，Sun One等等。这样的环境定义了一个框架（容器），在这一框架内，与环境定义接口标准相关的组件可以被实例化，并组合起来形成复杂的应用。与本地主机环境提供的低级功能相比，容器/组件主机环境可以提供更高级的编程性、管理性、灵活性和安全性。因此，基于组件/容器的运行环境广泛地被用于建造电子商务的服务。

通过定义服务的语义，OGSA给出了与任何主机环境无关的服务交互方式。但是，通过规定所有运行环境都具有的基本特征，可以使网格服务的成功实现变得容易。这些特征最后表现为不同的实现技术。

OGSA期望主机环境能够实现如下功能：网格名字向与实现相关的实体的映射，将网格激活和通知事件转化为与实现相关的活动，协议处理以及网络传输数据的格式化，网格服务实例的生命周期管理，服务间认证等等。

2.3.7 基于 OGSA 建立虚拟组织

应用和用户应该能够创建临时服务，并且可以发现这些服务并得到其特性。OGSA的Factory，Registry，GridService，HandleMap接口支持临时服务实例的创建，可以发现并且描述与虚拟组织有关的服务实例，图 2-17 [60]展示了用这些接口构造不同的虚拟组织服务结构。

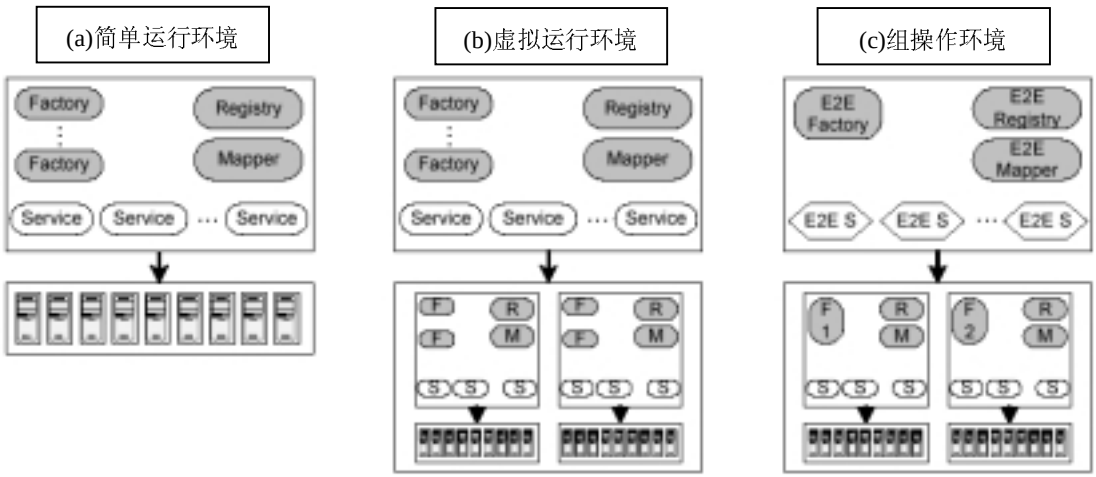


图 2-17 从简单到复杂的三种虚拟组织结构

首先对于简单运行环境，可以认为就是一些在一个简单的管理范围内的资源的集合，比如一个J2EE应用服务，Microsoft .NET系统或者Linux机群。在OGSA中，这一环境的用户接口将被构造成一个Registry，一个或者多个Factory，一个HandleMap服务。每一个Factory在Registry中记录，用于客户端发现可用的Factory。当一个Factory收到客户端要求创建网格服务实例的请求时，Factory就会激活相应运行环境的接口来创建新的实例，并且赋给它一个句柄，用Registry来注册该实例。这些不同服务的实现可以直接映

射到局部操作。

所谓虚拟运行环境，是指与虚拟组织相关连的资源可能跨越异构、地理分布的多个运行环境（例子是跨越两个不同的运行环境），但是这一虚拟运行环境为客户端提供相同的访问接口。相对于简单运行环境，虚拟运行环境中的一个或者多个更高级的Factory可以用于代理创建低级的Factory请求。类似地，可以创建一个高级的Registry，它知道已经创建的高级Factory和服务实例，以及特别的虚拟组织服务策略，它们用于管理虚拟组织的服务。客户端可以使用虚拟组织的Registry功能来发现Factory和其它与虚拟组织相关的服务实例，然后使用Registry返回的句柄直接和服务实例进行交互。高级的Factory和Registry实现了标准的接口，因此从用户的角度看，在使用上和其它的Factory和Registry是没有区别的。

这里的Registry句柄可以在虚拟组织维护的服务集合中作为一个全局唯一的名字使用，它属于虚拟组织维护的服务集合。资源管理的策略可以在拥有虚拟组织服务的平台上被定义和增强。

组操作环境是一种更高级的形式。在这一环境中，可以提供给虚拟组织参加者以更复杂的、虚拟的、组或者端到端的服务。在这种情况下，Registry跟踪并且公布创建高级服务实例的Factory。这种服务实例是通过将底层Factory创建的多个服务实例组合起来实现。

这三个例子，展示了如何通过网格服务机制来集成分布式的资源，这些资源可以是跨越多个虚拟组织边界的，也可以是在一个商业化IT基础设施的内部。不管是那种情况，注册后的网格服务组，可以跨越多个分布式的资源池，支持满足服务质量要求的功能。可以借助于一些应用和中间件来开发这些服务，在本地/远程透明以及局部优化的基础上，实现跨越异构平台的资源管理。和多个虚拟组织相联系的服务集合可以映射到相同的底层物理资源上，这些资源在逻辑上可以是不同的，但是在更低的物理层次上共享资源。

2.3.8 基于 OGSA 的应用例子

下面给出一个数据挖掘的例子（如图 2-18 [60]所示），它展示了基本的远程服务激发，生命周期管理以及通知功能。

（1）初始状态，该环境包括四个简单的运行环境：一个用户的应用程序，一个封装了计算和存储资源的简单运行环境（支持两个Factory服务，一个用于创建存储预留，另一个用于创建数据挖掘服务），另外两个封装了数据库服务的简单运行环境。

（2）用户程序在第二个运行环境中激发创建网格服务的请求，要求创建数据挖掘服务，执行数据挖掘操作，并且为计算分配临时空间。每一个请求都会激发用户和相关Factory之间的相互认证（使用Factory服务中描述的认证机制），然后就是对相应请求的授权。请求的成功将导致网格服务实例的创建，并给该实例赋以一个初始的生命周期。新的数据挖掘服务实例就拥有了委托代理证书，该证书允许服务实例以用户的身份执行进一步的远程操作。

（3）新创建的数据挖掘服务使用委托代理证书开始向其它的两个数据库服务提出数据请求，并将中间结果放在自己的局部存储器，数据挖掘服务同时使用通知机制向用户应用周期性地提供更新后的状态。同时，数据应用产生周期性的 “keepalive”请求，使得它创建的两个网格服务实例能够不断存活下来。

（4）由于某种原因，用户应用失败，但是数据挖掘计算仍继续进行，但是其结果已无人关心，继续存活的消息请求也不再产生。

（5）由于应用失败，继续存活消息停止，两个网格服务实例最终因为超过其生命

周期规定的时间而终止，同时释放相关的计算和存储资源。回到（1）状态，只不过用户应用已不存在。

以上就是一个简单的基于OGSA框架的应用的例子，例子虽然简单，但是基本描述了应用在OGSA框架下的工作过程和执行机制。

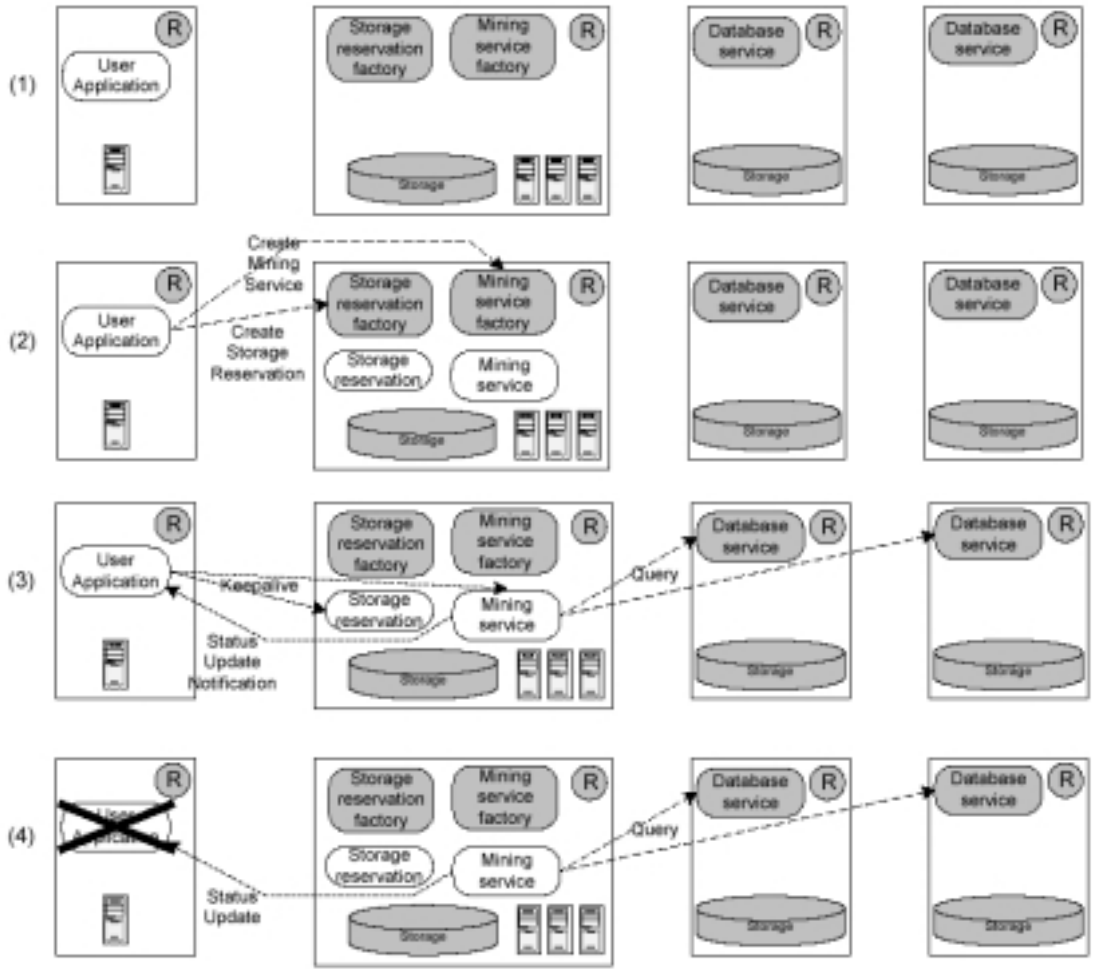


图 2-18 OGSA 应用例子

2.4 建造网格的几点建议

网格是一个有重要影响的基础设施，网格基础设施包括硬件、软件和网格组织。关于如何建造网格，必须遵照一定的原则来进行，否则就很难把网格建好。下面给出一些基于其它基础设施的建设过程得出的一些经验[22]。

2.4.1 国家行为

网格建设是一个国家行为，必须有政府的大力支持，这是因为有如下几个方面的原因。

首先网格是一个涉及到整个国家（甚至是跨越多个国家）的基础性设施，只有政府出面才能够协调好各方面的关系。网格建设一定会有工程施工，这需要工程建设部门和相应的规

划部门的介入。网格必然需要和传统的通信部门连接，因此也需要邮政、电信等提供积极的配合并且加入到网格的庞大网络体系中来。网格是一个技术性很强的基础设施，网格的建设没有科技部门的投入和参与是不可能建好的，它需要组织一大批的科技工作者投入到这一工程中来。由于国家网格的建设要覆盖到整个国家的版图，因此除了中央政府之外，还需要地方各级政府的配合。

其次就是网格这一基础设施的前期投入十分庞大，没有国家和政府的支持，即使是实力雄厚的公司或者企业，一般也不愿意承担这么大的风险，从投入到获取经济回报，周期较长，而且还要承担项目失败的风险，这不是一般的以盈利为目的单位所愿意投资的。网格是一个巨大的工程，以前的公路系统、电话与电报系统、电力网络、银行系统以及因特网，对整个社会都产生了巨大的影响，网格对整个社会造成影响的深度和广度，决不逊色于传统的基础设施，它需要政府首先大力支持，使网格能够运转起来，然后逐步走向成熟和完善。

再者，网格的建设和使用会涉及到国家安全、国家政策以及新的法令和法规，相应的规章制度只有国家和政府才有资格制定，网格从一开始的规划到建设直至完成和使用，都需要国家和政府进行全盘规划，并制订相应的政策和规定对网格进行规范化，这样才能够为网格这一基础设施的健康发展创造条件，提供保证。

2.4.2 从局部到整体

网格的局部和整体之间具有一定的自相似性，因此大规模网格的建造一定要先从局部开始，等到局部网格得到一定发展之后再进行整体网格的建设。比如高速公路系统，在建设跨越各个省，沟通全国的高速线路之前，各个省内部的子公路网络必须先建立起来，否则主干公路系统就无法和各个子公路系统相通，也就不能充分发挥全部公路系统以及主干公路系统的作用。对于以前的各种重要的基础设施，比如邮政系统，航运系统，银行系统等等都具有这样的特点。

从局部到整体的思路，具有如下好处，在局部网格的建设过程中，会遇到各种各样的问题，这些问题在整体网格的建设过程中一定会出现，而且有可能表现得更厉害，首先意识到这些问题并且进行解决对于整体网格的建设有很大的借鉴作用。局部网格建好之后，一旦和整体网格接通，就可以融入到整体网格之中，一方面可以增加整体网格的功能和资源，另一个方面就是局部网格的原有的用户就自然而然地成了整个网格的用户，可以访问更多的资源。

2.4.3 利用市场与经济杠杆

正如文献[22]所指出的那样，网格的出现，决不会是由于一些研究人员聚在一起经过热烈讨论后认为，“网格是件值得做的事情”，然后网格就奇迹般地出现了。

在网格建设的初期，需要政府的大力投入，使网格能够初步发挥作用，能够显示出市场的潜力，在此基础上吸引大量的投资者在市场这只“看不见的手”的作用下，自觉地投入到网格的建设和完善过程中，只有这时，网格才能够步入一个良性发展的轨道，不断扩大和健全。举例来说，美国政府投资建成 NSFNET 后，社会各界后来在其上投资的总和达到了美国政府一开始投资的一百多倍，从此可以看出：1 政府开始的投入会带动以后的迅速发展；2 基础设施项目需要社会各界的共同介入才会产生重大的效果。

在网格的建造和完善过程中，发明、研究和标准的制定在网格发展过程中有重要的作用。网格的正常运转使得这些发明等很容易产生经济利益，在经济利益的驱动下，会使得一些组

织或者个人自觉地投入到这一过程中来，整个社会的参与与介入，是推动网格发展最重要的力量。

可以通过应用研究刺激网格应用的发展，这些应用研究者可以是来自于计算科学或者是实验与观察科学，教育领域等。网格需要这样的使用人群，首先他们是分布在全球或者是一个国家的不同地方，他们相互之间必然有进行通信或者使用远程资源的需求（大量的需求会带来经济利益和动力），这些任务需要在网格上进行。这些人从一开始就使用网格将会大大缩短这种新的基础设施的开发时间。如果仅仅等待网格自然地发展，那恐怕需要太长的时间。

我们不能够假设网格一旦建好就会马上有大量的使用者，建成并不意味着大家愿意使用。如果没有人愿意使用它，那么网格就是一个失败。人们之所以会使用网格，是因为网格能够为他们的社会经济活动带来便利，增加收益，降低开销，利用市场这个推进器，会对网格的发展起到重要的推进作用。

2.5 小结

网格体系结构是网格系统的骨架，把握住它，就可以建立关于网格系统的整体概念，对网格有一个全局的观点，不至于在一些细节上纠缠。

本章介绍的两种网格体系结构具有很密切的关连性和继承性，也是目前最常见最流行的两种结构，但是并不是说网格就没有其它的体系结构，比如还有以积木块为基础的结构，以概念空间为基础的结构等等都是网格体系结构的重要补充。

希望本章的介绍会为后面网格的具体技术的介绍打下基础。使读者能够有整体和全部的观念后再去仔细了解不同的网格技术和应用。

思考题

1. 什么是网格体系结构？
2. 如何设计一个好的网格体系结构？目前主要的网格体系结构有哪些？
3. 五层结构具体包括哪五层？
4. 五层结构的基本思想是什么？
5. 开放网格体系结构的基本思想是什么？
6. 开放网格体系结构的运行机制是怎样的？
7. 在建造网格时有哪些历史的经验可以借鉴？

第3章 网格技术

本章在前面介绍的网格基础知识和网格体系结构的基础上，对网格所涉及到的各种技术进行介绍。网格技术是推动网格前进的主要力量，本章首先对网格技术进行了分类，然后分别对各类网格技术及其研究重点进行了介绍，包括网格应用技术、网格编程技术、网格核心管理技术以及网格底层支撑技术。这里对网格技术的介绍不是讲述网格技术的实现细节，而是侧重于网格技术所涉及到的主要方面，使读者对网格技术所涉及的范围有一个较为全面的把握。关于一些重要的网格技术实现方法将在后续的章节进行介绍。

3.1 网格技术分类

网格问题是以分布为基础的，广泛、方便、灵活，可以支持大规模、大粒度、大范围的资源共享问题。网格概念是由实际应用中的、具体的网格问题驱动的，为解决这些网格问题而出现的技术就是网格技术。

为了更好地解决网格问题，需要为网格设计一个可扩展的、开放的体系结构，然后以此体系结构为基础，开发相应的网格技术。关于网格体系结构，目前比较重要的有两个，一是文献[19]在早期提出的以协议为中心的 5 层结构；然后就是文献[60][61]进一步结合 Web Service 技术提出的以服务为中心的 OGSA。但是这些网格结构对于传统的软件开发人员来说也许有些不太习惯，在这里我们给出一种大家容易理解的、从网格开发角度体现出来的网格层次结构（图 3-1），根据这一结构来简单探讨一下网格技术的内涵。

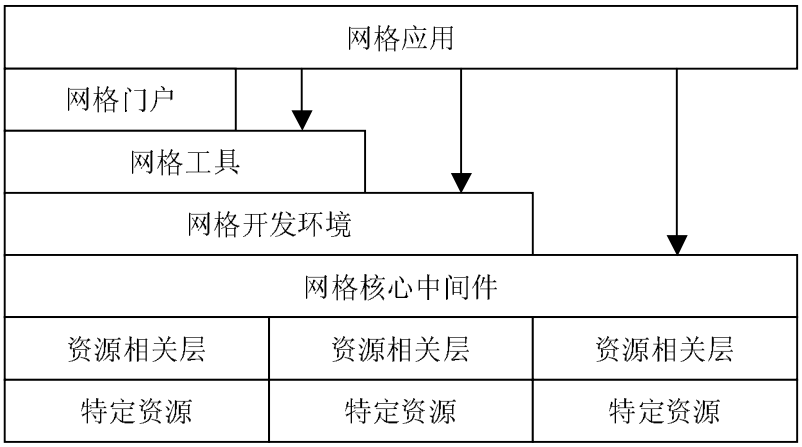


图 3-1 网格层次结构

在这一结构中，最底层就是各种可以用来共享的网格资源，它可以包括极其广泛的内容，从超大型仪器设备到一个简单的探测器都可以成为共享的目标，为了实现该资源的共享，需要提供相应的支持手段使得其它地方的使用者或者是应用程序可以访问、甚至是操纵它。

实现各种资源的广泛共享，需要提供一层与特定的资源相关的功能，它既可以实现对该资源的有效控制，又能够对上面的需求提供统一的接口，便于对资源的访问。这就是资源相关层。

网格中最关键的一层就是网格核心中间件，这层软件设施能够对分布的各种资源进行有效管理，为整个网格应用提供高效、安全、可靠的服务。网格核心中间件，是网格系统中连接上层应用和下层资源的纽带，它提供对网格的管理功能，但是这种功能一般层次上比较低，

因此不容易被普通的开发者掌握，可以把它比作计算机语言中的汇编语言，虽然功能足够强大，但是编程级别太低，因此应用程序开发效率太低。为了克服这一缺点，网格开发环境便应运而生。

网格开发环境是为了方便使用网格的各种功能而提出的一种集成的，高效的网格应用技术人员工作环境，使用这一环境，可以更容易地实现各种网格功能。

对于经常使用的网格功能，可以进一步以网格工具的形式固定下来，这样即使是非专业的人员，也可以通过网格工具方便地对网格进行一些基本的管理。

网格门户是为最终的网格用户提供的，网格门户就是用来访问网格服务与资源的可定制、个性化 Web 接口。对于网格用户来说，不需要知道网格的具体实现与细节，只需要为他们的经常性的工作、生活提供最方便，最容易理解以及最快捷的支持就行了。网格门户可以针对不同网格用户群的特点，给他们提供他们熟悉的容易理解和方便操作的界面，网格门户的界面友好与方便使用是网格应用成功的一个重要因素。

最上面的一层是各种各样的网格应用，也是网格最终服务的对象，比如物理学家利用世界范围的计算资源来进行 petabytes 级别数据的处理和分析；分布在不同应用领域的工程师用来合作设计、执行、分析一个挑战性的工程项目；保险公司可以从合作医院得到相关数据以防止被诈骗等等。网格应用可以解决更复杂的问题，使网格真正成为一种重要的基础性设施，为科研、社会以及经济的发展服务。

因此，基于图 3-1所示的结构，我们可以说，网格技术，就是在以上各个层次上，向上提供支持，向下提出相应要求的各种功能与手段。网格技术不排斥已有的各种成熟技术手段，但是仅仅依靠现有的技术是无法完全满足网格计算的要求的。

网格技术应该吸收和集成现有的大量成熟技术，那种认为网格技术和目前已有技术毫不相干的观点是不对的。从网格的发展看，网格概念就是在已有概念的基础上不断演化和发展而来的，网格技术也必然依托在已有的技术之上，网格技术和相关领域的研究是相辅相成的。另一方面，网格技术必然是创新的技术，因为大量网格问题的解决没有创新是不可能解决的，网格问题都是很富有挑战性的问题，解决这些问题必须有更先进的技术来支持。

综上，我们将网格技术从其所处的不同层次，进行如下划分，分别是网格应用技术，网格编程技术，网格核心管理技术以及网格底层支持技术，其相互之间的层次关系如图 3-2 所示。下面分别对这些技术进行介绍。

图 3-2 网格技术层次划分

3.2 网格应用技术

在[22]中，将网格应用分为四个主要的部分，分别是分布式超级计算应用、实时广域分布式仪器系统、数据密集型计算以及远程沉浸等。当然随着网格技术的发展，网格应用领域将会更广。

分布式超级计算是网格计算最早开始也是比较成熟的应用领域，主要是一些科学与工程

计算问题的解决。而远程仪器包括远程可视化以及远程制造等，其核心是网格支持的远程控制操作。数据密集型应用主要包括一些大型的天体物理、人体医学、图象等大型数据库的分析和处理。远程沉浸对性能和服务质量有特别的要求，比如应用于交互的科学计算可视化、教育、培训、场景模拟、艺术与娱乐等等领域。

关于网格应用的详细内容，在本书的后续章节会给出更具体的论述，故这里从略。

3.3 网格编程技术

如何对网格进行编程或者说如何在网格上进行程序设计？这是网格技术需要解决的一个重要问题。

如果没有编程工具和编程环境的支持，这样的程序设计必然是非常困难、低效而又容易出错的，这是我们在传统的程序设计中得出的经验，因此网格编程也需要网格编程工具和网格编程环境的支持。这里从几个方面对网格编程技术进行介绍，包括编程支持系统、面向对象编程技术、商品化技术以及数值计算编程技术等。由于这里介绍的目的是让读者对这些技术有一个总体上的了解和把握，而不是以详细介绍技术的实现为目的，因此不会过多地涉及技术细节（读者如果想了解更详细的内容，请参见文献[22]的 157－256 页）。

3.3.1 编程支持系统

对于编程支持系统，一般有如下几个要求和目标：

- 可以使应用程序的开发更简单
- 可以使开发出来的程序在不同体系结构和不同配置的运行环境中方便移植
- 可以使开发出来的应用具有很高的性能，即和程序设计专家直接利用底层提供的各种功能编写出来的程序的性能应该是接近的。

为了达到这些目标，必须从程序设计语言、编译器、运行库以及相关的支持工具等多个方面进行努力。

在传统的串行程序和后来的并行程序设计过程中，积累了不少的经验，这些都可以在网格编程环境中作为借鉴。已经存在的技术包括自动并行化技术，以消息传递为代表的显式通信并行编程技术，分布式共享内存编程技术，数据并行和任务并行编程技术，对通信延迟的控制技术（通过计算和通信的重叠来隐藏延迟，通过数据重用来降低延迟），负载平衡技术，运行时编译技术，功能强大的程序包或者库支持等。

网格环境将会引入更多的问题，首先是网格系统的异构性，这可以从计算能力、计算结构以及数据表示等各个方面体现出来；然后是网格系统的大数据访问延迟问题，而且它还会受到底层网络通信状况的影响；此外还有网格组件之间有限的网络带宽所带来的问题，动态负载的影响等等。这些问题使得网格编程将会面临更大的挑战。

在网格环境下，如果将运行在网格上的程序看作是一些任务，就可以充分利用已有的任务并行和任务分解技术，这样可以首先构造该程序的任务图，然后根据网格的配置特点对网格程序进行重新构造，最后就是将任务图中的任务分配到网格结点上进行运行。

如果将网格看作是一个分布式共享内存的系统，则可以将以前的分布式共享内存编程技术应用于网格编程，这里的最大问题就是程序的性能，基于共享内存操作产生的系统缺页将转化为通信调用，这一机制能否取得很好的性能，将直接影响这一编程模式的成败，这一模式在同构系统上取得了一定的成功。

当然专门为网格提供一种语言和编译器也是可能的，这种语言应该同时支持数据分解和

任务分解，需要语言、编译器、运行时系统等各个环节进行协调一致地高效运转，而且网格程序语言的接口不应该和目前主流程序语言的接口有太大的变化，这样会便于目前的程序员向网格程序员的转变。

由此可见，网格编程支持系统是非常复杂的，会遇到比高性能并行编程系统更多的问题，但是大量的网格应用的开发必须依赖于这样的支持系统，因此它对网格应用的成败有重要的意义。

3.3.2 面向对象技术及 Legion

面向对象的思想是软件领域的一次重要革命，这一思想当然也可以应用于网格编程技术之中。用面向对象的思想可以构造一个个的组件，然后通过将一个个的组件集成起来，就可以构造复杂的应用，就如同搭积木一样，通过这种方式构造出来的系统一般被称为组件系统。

在网格系统中，应该考虑如下问题：首先是组件框架下的对象之间要能够交换数据和成员函数的消息，因此它们之间应该能够建立某种联系；然后就是网格系统对象组件有的是永久性的，而有的是临时的；效率也是面向对象思想中应该集中解决的一个问题，许多面向对象系统都因为无法提供令人满意的效率而使这种技术本身受到影响；此外面向对象系统还得解决比如命名、永久对象和存储管理、对象共享、进程与线程管理、对象分布与对象迁移、事件日志、容错、支持并行等等问题。这些都是面向对象思想应用于网格时需要解决的问题。

下面以 Legion[11]为例来说明面向对象技术是如何在网格中得到应用的。Legion 是面向对象技术在网格中应用的重要实例，它将网格看作是一个世界范围的抽象计算机， 希望用户在 Legion 环境中感觉到的是“一台”大的计算机，而网格用户是在这台大计算机上进行程序设计。

在 Legion 中，一切都是对象，Legion 规定了对象交互的消息格式与高级协议，但是对编程语言和具体的通信协议没有规定。Legion 的类负责管理其实例并提供系统级的功能，而且 Legion 允许用户提供自己的类对象来改变系统级的对象支持机制。Legion 定义了核心对象的接口和基本的功能，用来支持基本的系统服务，比如命名、绑定、对象创建、激活、删除等。

Legion 通过多种手段来提高其性能，首先是支持并行库，包括 MPI 和 PVM；其次是对并行语言的支持，并且包装了一些并行组件；最后，Legion 开放了其运行库的接口，从而允许第三方直接基于对运行库的操作来开发自己的高效用户程序。Legion 库支持基本的通信，编码和解码，认证，异常检测与传播等特征，这样就为用户的二次开发提供了很大的支持。

Legion 提供两种类型的资源，分别是计算资源和存储资源。Legion 的调度基于两个原则：（1）地点自治，地点自治使得资源的提供者对如何使用其拥有的资源有最终的决定权；（2）用户自治，用户自治使得用户自己可以选择调度策略从而为赢得性能提供可能。当然在用户不提供调度策略的情况下系统可以提供一种缺省的调度方法。

Legion 面向对象思想在网格中的具体实现为我们开发面向对象的网格编程技术提供了借鉴，当然其它传统的编程技术和经验都可以为网格编程提供重要的参考。

3.3.3 基于商品化技术集成的网格编程

另外一种网格编程的方法就是充分利用已有的商品化技术，通过将这些技术的有机集成，为网格编程提供支持。这一方法在高性能计算领域进行了应用实践并且已经取得不少经

验，在网格计算中，也可以借鉴这种方法。这些商品化的技术包括 VRML，Java3D，JDBC，CORBA，COM，JavaBean，Web 以及各种网络技术等。

这一方法的一个重要成果就是提出了一种逻辑结构形式，如图 3-3所示，被称为三结框架结构。这一结构对于商品化技术的集成方法具有重要的指导意义。

最上面的一层形成一个结，由于它的功能一般不是十分复杂，因此被称为瘦前端，主要包括一些图形化的用户界面，应用程序和相关的工具。中间一层形成另外一个结，主要功能是用来协调相对简单的前端和较为复杂的后端的功能，包括一些高级的 Agent，用来实现负载平衡、变换、度量以及检测等服务。最下面一个结形成本结构的最后一个结，一般称为后端服务，后端服务是最复杂的，它负责对来自客户端的各种需求进行支持。

值得指出的是，三结框架只是一个逻辑上的结构，在具体系统的物理构成上，有可能并不是严格按照逻辑结构来实现的。

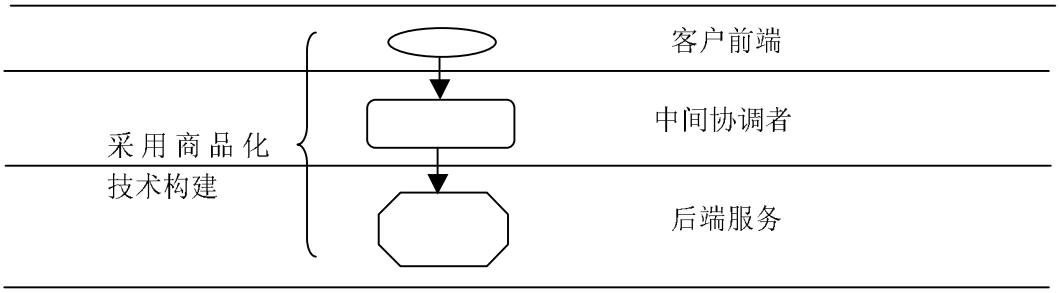


图 3-3 三结框架

分布式对象技术和分布式服务技术是构建三结框架的基本技术，分布式对象技术目前主要是指基于面向对象的 CORBA，COM 以及 JavaBean 等技术，而分布式服务技术主要是指 Web 技术，比如将分布的 HTTP 服务器聚集起来提供中间层的服务。当然将这两种基本技术结合起来使用也是一种重要的趋势。

通过商品化技术构造网格编程环境是否能够取得成功还在探索之中，它的成败在很大程度上取决于商品化技术的发展程度和应用程度。

3.3.4 数值计算编程环境 NetSolve

面向特定应用领域的网格工具可以大大方便普通用户对网格资源的访问，而且可以使这些领域用户能够以他们熟悉的方式，使用更丰富更强大的网格资源从事本领域的研究。这样就需要不同应用领域网格工具的支持，为网格编程提供方便。

网格应用工具的设计，一般都必须要求能够很好地支持该领域的应用模型，这是网格工具成败的关键；然后就是要能够方便地查找、组织并且使用相关的资源，能够充分利用这些领域的资源完成应用任务；由于网格资源的多样性和复杂性，很难避免错误不会发生，因此网格应用工具必须考虑如何避免各种错误造成的损失，如何消除网格错误对应用的影响；最后就是网格应用工具的自适应性，因为网格应用和网格资源具有很大的动态变化性和不可预测性，但是网格工具又必须为最终的应用提供性能和质量的保证，使得网格工具能够适应来自应用和网格资源两个方面的变化。

下面以一个具体的网格应用工具 NetSolve[23]为例，来说明网格工具是如何在网格应用中发挥作用的。NetSolve 有三个组成部分（如图 3-4所示），分别是服务的请求者或者发起者－客户端，服务调度和维护者－Agent 以及服务的提供者－Server。

首先是数值计算的用户即客户端发起服务请求，比如在 NetSolve 中，为了计算 a＝b×c，

可以用 NetSolve 提供的语言 a＝netsolve（‘matmul’，b，c）来表示。这样表示的优点是，对于底层的服务提供者，不管是用何种方式或者语言实现这一功能的，比如 C，FORTRAN，Java 或者是数学包 MATLAB，都可以被 NetSolve 调用来给出结果。

对于客户端以 NetSolve 语言表达出来的请求，将交给 Agent，Agent 的作用是依靠它维护的各种计算资源及其相关特征的列表，接受客户端的请求，并且将这一请求交给合适的计算资源去处理。

客户端请求的最终完成是在 Server 上实现的，它代表各种各样具体的计算资源，对于 NetSolve 来说，这些计算资源提供的计算服务必须是预先安装好的，可以随时被调用，而且这些计算服务是可配置的，对于新的服务可以随时增加进来。

这样，最终的用户可以从客户端，用 NetSolve 提供的方式，将其计算需求表达出来，并且交给一个最近的 Agent，这样 Agent 就会自动发现并且寻找合适的计算资源，让该资源完成用户的计算请求，并将结果返回给用户。这就是使用 NetSolve 进行网格编程的主要过程。

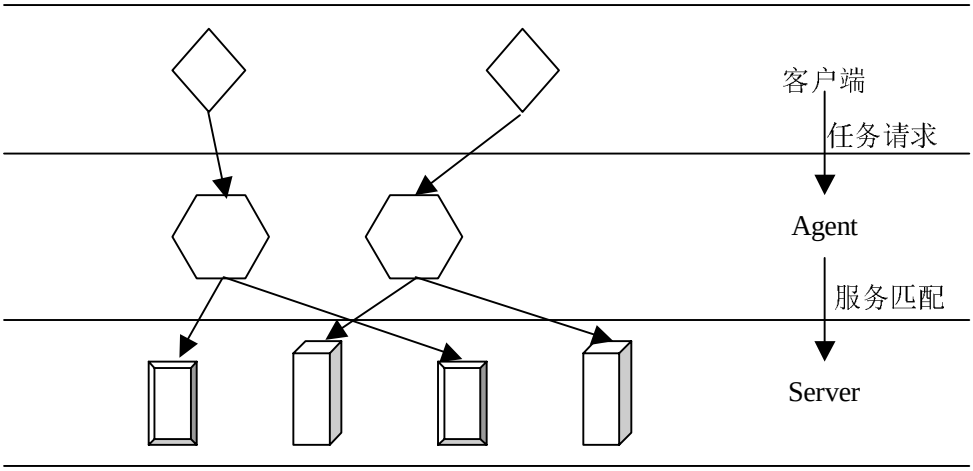


图 3-4 NetSolve 工作过程

从 NetSolve 例子可以看出，面向特定的网格应用领域，提供更有针对性的网格支持工具，可以使网格的编程工作变得十分简单，而且很容易实现跨平台，对于网格的使用者来说，也会更容易地实现对网格资源的使用。

3.4 网格核心服务技术

网格核心服务非常重要，它是连接网格底层与高层功能的纽带，是协调整个网格系统有效运转的中枢，对这部分网格技术的研究具有重要的意义。下面分别从高性能调度技术、高吞吐率管理技术、数据收集分析以及可视化技术、安全技术几个方面对它进行介绍，更详细的细节问题可以查阅文献[22]的 257－420 页。

3.4.1 高性能调度技术

在网格系统中，有大量的应用在运行，这些应用又共享网格的各种资源，如何才能够使得这些应用获得最大的性能？这就是调度需要解决的问题。

调度问题一般可以分为两大步，第一步就是在空间上对计算和数据进行分配，包括选取

给定任务所需要的资源组合，将任务交给这些资源去执行，并分配相关的数据和计算等。第二步就是在时间上为计算和通信进行排序，包括在计算资源上为不同的任务进行排序，同时为不同任务之间的通信进行排序。

网格的调度问题可以参照传统的高性能计算中的调度策略和技术，但是网格调度却更复杂，这是因为任何一个网格调度器都无法对所有的网格资源进行管理，而只能是一定范围内的网格资源，另外，网格资源还是动态变化的，由于其它应用引起的资源竞争对性能影响很大，而且会经常出现，网格资源的类型多样、复杂，不同类型的资源会展示出不同的性能特征，而且相同类型的资源由于共享等原因所展示的性能也是随时间变化的。

因此网格的调度，需要建立随时间变化的性能预测模型，充分利用网格的动态信息来表示网格性能的波动，而且网格的调度必须考虑到多种多样的环境和条件，这是由网格的异构性和多样性特征所决定的。

在网格调度中，还存在着移植性、扩展性、效率、可重复性以及网格调度和本地调度的结合等等问题，这些都是网格技术需要解决的问题。

3.4.2 高吞吐率资源管理技术

另外一种比较重要的计算形式就是高吞吐率计算（High Throughput Computing），这种计算与传统的高性能计算相比，侧重点有所不同，它更关心的是在一段相对较长的时间内计算系统或者是计算资源所能够提供的服务或者是完成任务的多少。高性能计算对响应时间特别敏感，而高吞吐率计算更侧重于整个系统的效用（utilization）。

高吞吐率系统具有强健性、扩展性以及可移植的要求。强健性是为了使系统的服务失效时间最小化，一种常见的技术就是检查点技术，它通过定期保存应用状态的“快照”，从而使得当应用或者系统出现异常时，可以重新从一个最近保留的状态继续执行下去而不是一切从头开始。而扩展性使得系统可以增加更多的资源，和移植性相结合就可以为更多的用户提供服务。

资源管理在高吞吐率系统中的占有非常重要的地位，从下到上，一般将资源管理分为六层，分别是局部资源管理层，拥有者层，系统层，客户层，应用管理层和应用层。这种层次划分对于设计资源管理系统具有重要的指导意义。

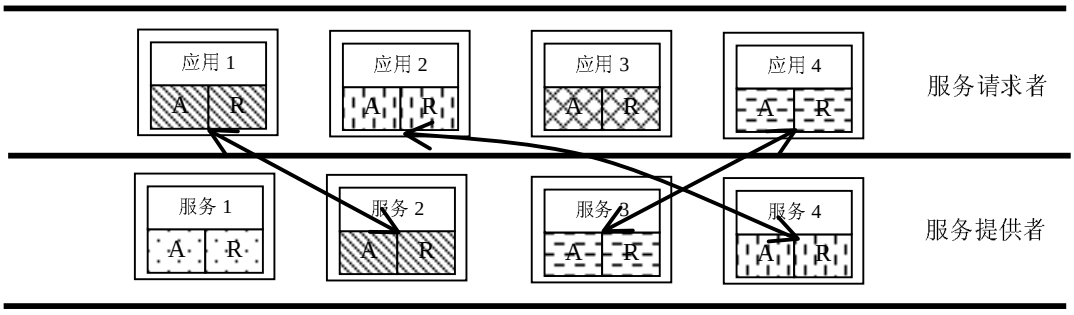


图 3-5 高吞吐率系统的匹配机制

高吞吐率系统的有效运转必须解决好服务的请求者和提供者的表示和匹配问题，不管是请求者还是提供者，都需要对其自身的属性以及它对对方的要求明确提出来，这样就需要一种机制，使得应用方可以找到满足它需要的服务，服务方为符合特定要求的请求者提供服务。如图 3-5所示，首先是几个不同的应用请求者和服务提供者分别都给出自己的自身属性 A 和特定的请求 R，即所谓的公告（advertising），然后根据供需双方各自的需求，进行相互的

匹配（matchmaking），一旦匹配上，则双方相互声明达成一致（claiming）。图中<应用 1，服务 2>，<应用 2，服务 4>，<应用 4，服务 3>可以相互匹配而达成一致，可以使用对方的服务或者为对方提供服务，应用 3 和服务 1 因为找不到匹配的对象而不能被服务或者不能提供服务。

在网格环境中，高吞吐率系统发挥着越来越重要的作用，因此对它的研究就有着普遍性的意义，这些都为相应网格技术的研究提供了动力和舞台。

3.4.3 性能数据收集、分析与可视化技术

在网格环境下，为了提高不同应用或者系统的性能，常常需要获取其运行状态下的相关性能数据，这样通过对这些性能数据分析，就可以设法提高下次程序运行的效率，或者为系统调度提供指导，或者使应用/系统的运行过程通过动态调整来提高性能。这就是性能数据的收集和分析的意义所在。

最常见的获取性能数据的源泉包括运行的程序、操作系统、处理器以及网络等。好的性能收集工具应该能够从各个地方和各个环节获取相关的性能数据。有时候性能数据的获取并不是十分容易，在有些情况下性能数据的获取是在特定的硬件支持下才可以实现的。在进行性能数据采集时，时钟的分辨率、访问延迟以及多结点同步对数据采集的结果影响较大。

除了收集，性能数据的表示对于采集数据的时间有效性、需要的存储空间以及可能采取的分析类型有重要的影响。

在一定的误差范围内，工具的使用者对性能数据是如何获取的并不关心，即数据获取的细节对于他们来说并没有多少意义，因此对软件、硬件以及网络等进行抽象就具有现实的意义，以便于工具使用者在更高的抽象级别对性能数据的理解。

对于得到的性能数据，需要进行各种分析，包括定量分析、自动性能诊断、扰动分析等。但是性能数据收集和分析的主要挑战就在于数据可视以及建立与源代码的关联。由于现在的硬件集成和编译转换技术，常常使性能数据很难和最一开始的源代码建立关联，这是性能数据分析遇到的一个重要障碍。

关于在网格环境收集性能数据一个需要注意的地方就是，在提供性能数据的同时，必须为应用程序和主机提供安全保障。

网格性能数据一般数据量庞大，关系复杂，在分析的基础上，借助可视化技术，可以清晰地展示这些数据所代表的意义，因此可视化也是网格中一种重要的技术。

性能数据的收集、分析与可视化，是三个密切相关的环节，而且这三个环节应该和网格的动态运行过程建立很好的联系，准确反映网格的运行状况，这样才能能够更深入地分析和指导网格应用和网格核心管理部分的开发。

3.4.4 安全技术

网格安全包括的内容非常广泛，比如认证、授权、保证、记帐、审计、完整性、机密性等等，在网格的各个部分都涉及到安全技术，因此它非常重要，在本书的第二篇对安全技术进行了较为详细的介绍，故这里从略。

3.5 网格底层支撑技术

前面讲的是网格的上层和中层技术，在网格的底层，也需要相应的支撑技术，这些技术

是构建网格的基础。

首先是关于网格计算结点的构建技术，计算结点包括计算、通信、存储等多种基本的组成元素，计算结点就是由这些基本的元素组合起来形成的。强大、灵活、可扩展的计算结点是网格服务的重要支撑者，是一种最重要的资源提供者，当然还有数据库服务、仪器等其它的资源提供者，如何使它们能够有效地为整个网格提供资源服务是网格技术的一个重要研究方面。

网络是网格的基础，网络协议技术对于网格有重要的意义，一般将网络协议分为数据传输协议、流协议、组通信协议、分布式对象协议等，这些协议为分布实体之间的联系建立了基础，是网格管理和网格应用的基础。除了网络协议，网络的服务质量对网格的正常运行也非常重要，基于网络服务质量的保证，可以提高网格和网格应用的性能。网络是网格的另外一种重要的资源，也为其它的资源建立了物理上的联系。

此外局部结点的操作系统和网络接口，底层网络基础设施以及网格试验床等都对网格系统起到重要的作用，这里所涉及到的各种技术都是网格底层支持技术所需要研究的范围。

3.6 网格技术发展

网格一词最早出现在20世纪90年代中期[22]，而网格计算的概念在1995年的I-WAY[24]项目中被提出。网格概念是在问题和应用的推动下不断发展、丰富和完善的。

网格计算一开始更多地被称为元计算[55]，目前最著名的网格计算工具包 Globus[21]在一开始被称为元计算工具包。过去对元计算的研究可以认为是网格计算的初级阶段。还有一些和网格计算相关的概念是分布式计算（Distributed Computing），对等式计算（Peer-to-Peer Computing），因特网计算（Internet Computing），全球计算（Global Computing），无缝的可扩展计算（Seamless Scalable Computing），基于 Web 的并行计算（web-based parallel computing）等。由于关于网格以及网格计算本身还没有一个公认的定义，比如 GGF（Global Grid Forum）[51]认为网格计算就是分布式计算，对等式计算是工业界对网格计算的另外一种叫法，况且网格和网格计算本身还在发展之中，因此我们在本书中并不去仔细区分这些概念的严格界限，统一把他们叫做相关研究领域。

网格的发展到目前为止基本上可以划分为以下几个阶段[59]：

- 萌芽阶段：在90年代早期，主要是千兆网的测试床，以及一些元计算的实验。
- 早期实验阶段：在90年代中期到晚期，比如I-WAY项目，还包括一些学术性的软件项目，比如，Globus, Legion，还有一些应用实验。
- 迅速发展阶段：2002年以来，出现了大量的应用社团和项目，主要基础设施的开发和使用，工业界对网格计算的兴趣在增长，比如IBM，Platform，Microsoft，Sun，Compaq等重要的公司。同时也出现了一些比较显著的技术基础，比如Globus Toolkit，形成了具有相当规模和世界影响的GGF组织，它大概有500个人，共有20多个国家。

在世界范围内的网格计算项目很多。由美国自然科学基金资助的PACI（Partnership for Advanced Computational Infrastructure）项目，包括两个重要的部分，分别是NCSA（National Computational Science Alliance）[25]和NPACI（National Partnership for Advanced Computational Infrastructure）[26]，这一项目通过将学术界、政府部门和工业界的力量结合起来，建立一个网格计算基础设施的伙伴联盟，来促进科学发现和工程研究的发展。

美国的NASA（National Aeronautics and Space Administration）构造了一个网格计算实验床，称为IPG（Information Power Grid）[27]，它可以将NASA分布在各地的资源通过网络（包括无线通信手段）连接起来，解决NASA目前无法解决的科学与工程计算与数据管理等问题。

由美国能源部（Department of Energy）和三个国家重点实验室Sandia 、Livermore与Los Alamos共同承担的ASCI（Accelerated Strategic Computing Initiative）计划[28]，是一个主要用于军事目的的高性能计算发展计划，其目的是在不进行物理核试验的情况下，通过计算模拟，来开展核武器的全方位研究，以维持其在军事领域的核威慑地位。美国军方的GIG[4]也是一个重要的网格项目。

在国外的网格计算研究项目，简单总结一下可以发现，一些通用目的网格技术研究和项目有：Access Grid[16]，Condor[17]，EcoGrid[32]，Globus [21]，Legion [11]，NMI 计划（NSF Middleware Initiative） [33]，SinRG [34]，Polder [35]，MOL[36]；一些网格应用和库有AppLeS[37]， Cactus 计算工具包[38]， CAVERNsoft [39] ， GrADS 网格应用开发软件[40]， 网格协作门户(包括 NASA, NCSA, SDSC) [41]，NEOS[42]， Netsolve [23]，Nimrod/G [43]， PUNCH[44]；还有一些商业界在网格计算方面的努力，包括P2P 工作组[45]，Avaki[46]， Entropia[47]， Gridware [48]， InSors[49]。在日本的网格项目有 Ninf[54]。

关于网格论坛在欧洲有欧洲网格论坛[50]，更大的论坛有全球网格论坛GGF（Global Grid Forum）[51]。

在网格建设方面，NCSA建造了NTG（National Technology Grid），后来又出现了STAR TAP（Science Technology And Research Transit Access Point）[52]，它将进一步扩展为连接全世界的网格即iGrid（International Grid）[53]。

在国内，网格计算也正处于快速发展时期，主要有中科院牵头的“国家高性能计算环境（National High Performance Computing Environment，简称 NHPCE）”项目[29]，NHPCE 的长期目标是提高计算网格系统的性能、可扩展性及可用性。目前包括北京、长沙、成都、合肥、上海、西安等几个试验结点。另一个重要的项目是由清华大学牵头，由教育部支持的重点项目“先进计算基础设施北京上海试点工程”[30]，其目的是建立一个主要为教育系统各单位提供资源共享的科研、教育、培训等高性能计算基础设施，实现跨学科、跨地域合作与人才培养。

2002 年国内启动的 863 信息领域高性能计算机及其核心软件专项就是一个网格计算项目[31]，它以“需求牵引、技术跨越、多方协作、聚焦网格”为指导思想，以实现高性能计算机及其核心软件技术跨越为目标；研制能有效支持科学工程计算、新一代因特网信息服务和数据库应用，具有资源共享、协同工作能力的国家高性能计算环境（亦称网格）；将高性能计算服务送到科教、企业、政府等各方面用户的桌面上，推动我国网格应用及其产业的发展，提高我国的综合国力和国际竞争能力。

这些都说明了我国现在对网格计算研究的重视，相信经过努力，我国的网格基础设施就会发挥作用，为科研、教育、生产服务。

不难看出，目前国内外在网格计算方面的研究十分活跃，研究范围跨度很大。目前呈现这样的趋势，一是网格计算标准化的呼声越来越高，目的就是为了规范和统一现在大量的网格计算研究，Globus 在一定程度上成为事实上的标准；二是专用网格的研究与开发成为一个重要的方向，因为网格是面向具体问题的应用的，而专用网格在这一方面具有独特的优势，可以为通用网格技术提出最直接最具体的需求；三是开放的面向 Web Service 的框架结构和与工商业界应用的结合是网格技术研究的一个重要趋势，原来的网格计算主要集中在科学计算等学术领域，而目前正在走向实用并与市场结合，直接服务于生产和各种商业活动。

3.7 小结

本章是在对网格技术有一个总的分类的基础上，对网格技术可能涉及到的各个方面进行介绍。从中可以发现网格技术涉及的范围是极其广泛的，在网格技术领域有大量的开创性工

作有待于各方面的人员去探索和研究。本章给出的是网格技术的一个全貌，对目前已经取得进展的网格技术成果，将在下面的章节进行详细介绍。

思考题

1. 什么是网格问题？什么是网格技术？网格技术可以大致分为哪几类？
2. 你认为理想的网格编程环境应该是怎样的？为达到这样的目的需要提供怎样的技术支持？
3. 你认为网格核心部分的功能应该是什么？这些核心功能的实现需要何种技术的支持？
4. 网格底层的支持包括哪些方面？
5. 目前的网格技术发展，主要包括哪些重要的项目？研究领域主要集中在哪些方面？网格技术发展具有什么样的趋势？

参考文献

[1] 美国地球系统网格“ESG”。http://www.earthsystemgrid.org/

[2] 欧盟的数据网格项目“DataGrid”。http://eu-datagrid.web.cern.ch/eu-datagrid/

[3] 美国地震网格“NEESgrid”。http://www.neesgrid.org/

[4] 美国军事网格“GIG(Global Information Grid)” http://www.mitre.org/pubs/edge/july_01/miller.htm, http://dmsweb.belvoir.army.mil/gigcrd.pdf

[5] Leigh, J., Johnson, A. and DeFanti, T.A. CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments. *Virtual Reality: Research, Development and Applications*, *2* (2). 217-237. 1997.

[6] Lopez, I., Follen, G., Gutierrez, R., Foster, I., Ginsburg, B., Larsson, O., S. Martin and Tuecke, S., NPSS on NASA's IPG: Using CORBA and Globus to Coordinate Multidisciplinary Aeroscience Applications. In *Proc. NASA HPCC/CAS Workshop*, (NASA Ames Research Center, 2000). http://accl.lerc.nasa.gov/IPG/CORBA/

[7] Benger, W., Foster, I., Novotny, J., Seidel, E., Shalf, J., Smith, W. and Walker, P., Numerical Relativity in a Distributed Environment. In *Proc. 9th SIAM Conference on Parallel Processing for Scientific Computing*, (1999)

[8] Bolcer, G.A. and Kaiser, G.E. SWAP: Leveraging the Web To Manage Workflow. *IEEE Internet Computing*, *3* (1). 85-88. 1999.

[9] Web Service 工作组，http://www.w3.org/2002/ws/

[10] JXTA Project。http://www.jxta.org/

[11] Grimshaw, A., Ferrari, A.,Knabe, F., Humphrey, M.: Legion: An Operating System for Wide Area Computing. Dept. of Comp. Sci., U. of Virginia, Charlottesville, Va. Available at ftp://ftp.cs.virginia.edu/pub/techreports/CS-99-12.ps.Z and The Legion Project: http://legion.virginia.edu/http://legion.virginia.edu/

[12] 10 Gigabit Ethernet Alliance Homepage。http://www.10gea.org/

[13] Virtual Interface Architecture Specification ， http://www.viarch.org/， http://developer.intel.com/design/servers/vi/

[14] D. Dunning et al., “The Virtual Interface Architecture,” *IEEE Micr*o, Mar.-Apr. 1998, pp. 66-76.

[15] Myrinet HomePage。http://www.myri.com/

[16] Access Grid HomePage。[web-accessgrid]http://www-fp.mcs.anl.gov/fl/accessgrid/， http://www.accessgrid.org

[17] M. J. Litzkow, M. Livny, and M. W. Mutka, “Condor - A hunter of idle workstations,” 8th International Conference on Distributed Computing Systems, 1988, pp. 104-111. http://www.cs.wisc.edu/condor/

[18] IAN FOSTER， “Internet Computing and the Emerging Grid” Nature Web Matters, 7 December 2000， http://www.nature.com/nature/webmatters/grid/grid.html

[19] I. Foster, C. Kesselman, S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". International J. Supercomputer Applications, 15(3), 2001。Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on , 2001 ,Page(s): 6 –7。

[20] I. Foster, C. Kesselman, J. Nick, S. Tuecke; "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration". January, 2002. http://www.Globus.org/research/papers/ogsa.pdf
[21] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, 11(2):115-128, 1997。The Globus project: http://www.Globus.org/ http://www.globus.org/training/grids-and-globus-toolkit/IntroToGridsAndGlobusToolkit.ppt
[22] I. Foster and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Fransisco, CA, 1999. http://mkp.com/grids, http://www.gridforum.org/，http://www.ccgrid.org/
[23] Netsolve Project: http://www.cs.utk.edu/netsolve/
[24] I-WAY Project。http://www.iway.org/, http://archive.ncsa.uiuc.edu/General/Training/SC95/I-WAY.nextgen.html
[25] NCSA Project。http://www.ncsa.uiuc.edu/
[26] NPACI Project。http://www.npaci.edu/
[27] W. E. Johnston, D. Gannon, and B. Nitzberg, “Information Power Grid Implementation Plan: Research, Development, and Testbeds for High Performance, Widely Distributed, Collaborative, Computing and Information Systems Supporting Science and Engineering,” NASA Ames Research Center, http://www.nas.nasa.gov/IPG , 1999.
[28] ASCI Project。http://www.lanl.gov/projects/asci/asci.html http://www.llnl.gov/asci/，http://www.sandia.gov/ASCI/，http://www.sandia.gov/ASCI/，http://www.lanl.gov/asci/，http://www.lanl.gov/projects/asci/asci.html
[29] Chinese NHPCE Project。http://www.grid.ac.cn
[30] Chinese ACI of MOE Project。http://aci.cs.tsinghua.edu.cn, http://www.hpclab.shu.edu.cn
[31] 863 网格计算专项。http://www.863.org.cn/863_105/applygide/applygide2/information2_app/200206270024.html
[32] Economic Grid。http://www.csse.monash.edu.au/~rajkumar/thesis/thesis.zip
[33] NMI Project。http://www.nsf-middleware.org/
[34] SinRG Project。http://icl.cs.utk.edu/sinrg/
[35] Polder Project。http://www.wins.uva.nl/projects/polder/
[36] MOL Project。http://www.uni-paderborn.de/pc2/projects/mol/
[37] AppLes Project。http://apples.ucsd.edu/
[38] Cactus Project。http://www.cactuscode.org/
[39] CAVERNSoft Project。 http://www.openchannelsoftware.org/projects/CAVERNsoft_G2/
[40] GrADS Project。http://www.isi.edu/grads/
[41] Grid Portal。http://www.ipg.nasa.gov/portals/
[42] NEOS Project。http://www-neos.mcs.anl.gov/neos/
[43] Abramson, D., Sosic, R., Giddy, J. and Hall, B. Nimrod: A Tool for Performing Parameterised Simulations Using Distributed Workstations. In *Proc. 4th IEEE Symp. on High Performance Distributed Computing*, 1995. http://www.csse.monash.edu.au/~davida/nimrod/
[44] PUNCH Project。http://punch.ecn.purdue.edu/
[45] P2P Computing Project。http://www.peer-to-peerwg.org/
[46] Avaki Project。 http://www.avaki.com/，http://www.avaki.com/products/aboutgrid.html

[47] Entropia Project。http://www.entropia.com/
[48] Gridware Project。http://wwws.sun.com/software/gridware/
[49] Insors Project。http://www.insors.com/
[50] European Grid Forum。http://www.egrid.org/
[51] Global Grid Forum HomePage。 http://www.globalgridforum.com/
[52] STAR TAP HomePage。http://www.startap.net/
[53] International Grid。 http://www.Globus.org/documentation/incoming/inet99.pdf
[54] Ninf: Network Infrastructure for global computing. http://ninf.etl.go.jp/
[55] L. Smarr and C. Catlett, Metacomputing, Communication of the ACM, Vol.35 No. 6(1992), pp. 44-52.
[56] 美国能源部的科学网格“DOE Science Grid”。http://doesciencegrid.org/
http://doesciencegrid.org/Grid/papers/DOE_Science_Grid_Collaboratory_Pilot_Proposal_03_14.nobudget.pdf
[57] William Johnston 等关于计算网格的论述
http://www.itg.lbl.gov/~johnston/Grids/DOE_Science_Grid_PImeeting_Jan,2002.1.ppt
http://www.itg.lbl.gov/~johnston/Grids/DOEScienceGrid.and.ComputationalScience.ppt
[58] B.H. McCormick，T.A. DeFanti，M.D. Brown （Eds）:“Visualization in Scientific Computing”，IEEE Computer，Vol 23，No 8，August，1989.
[59] Foster 做的关于网格计算的演讲材料，可从如下网址下载：
http://www-fp.mcs.anl.gov/~foster/Talks/WWWGridsMay2002.ppt
http://www-fp.mcs.anl.gov/~foster/Talks/GridTutorial.ppt
[60] OGSA 结构描述，http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf，
http://www.Globus.org/ogsa/
[61] OGSA 规范，http://www.gridforum.org/ogsi-wg/drafts/GS_Spec_draft03_2002-07-17.pdf
[62] GGF 的 OGSI 工作组，http://www.gridforum.org/ogsi-wg/

第二篇 网格实现

作者简介

陈渝

清华大学计算机系高性能计算所助教。

1993 年在国防科技大学获得学士学位

1997 年在国防科技大学获得硕士学位，导师及兰盛研究员和杨学军教授

2000 年在国防科学技术大学获博士学位，导师是陈福接教授和杨学军教授

2000 年－2002 年初在清华大学计算机系做博士后研究，合作导师是李三立院士

现在清华大学计算机系高性计算能所工作。目前的主要研究方向是大规模并行系统中的通信优化、容错、操作系统和网格计算等方面的研究。在清华大学作博士后研究的初期，由于在内核级容错支持的并行消息传递库 MPI 和高性能用户层并行通信协议 VIA 的实现这两方面工作出色，于 2001 年前期的清华大学博士后中期考核中被评为特优。
目前已经在国内核心期刊、国际学术会议和杂志上发表学术论文 20 余篇。完成译著（第一译者）“Numerical Methord using MATLAB”，电子工业出版社。编著（第三作者）“JSP 编程实践－－活动网页的引擎”，清华大学出版社。

第4章 Globus 项目介绍

本章主要对 Globus 项目[1][7]进行简要的介绍，讲述了 Globus 项目产生的目的和发展情况[8]，使读者可以了解 Globus 项目与网格计算之间的关系、Globus 项目与网格计算工具软件－Globus Toolkit 之间的关系。同时初步讲解了总体结构和主要的组成部分[5]，为下面各章的具体介绍做了一个铺垫。

4.1 Globus 的起源和发展

4.1.1 Globus 项目与 Globus 工具包

Globus 项目是目前国际上最有影响的与网格计算相关的项目之一。它发起于九十年代中期，其最初的目的是希望把美国境内的各个高性能计算中心通过高性能网络连接起来，方便美国的大学和研究机构使用，提高高性能计算机的使用效率.当时在美国建立了一个试验环境－I-WAY，它把位于美国 17 个不同地点的 60 多个组织的超级计算机和资源通过高性能网络联系起来，进行大规模科学模拟、协同工程、并行计算等科学研究[54][55][56]，这实际上是 Globus 项目的前身。随着对 Globus 项目的深入研究，针对它的目标也进一步扩展，希望通过 Globus 项目可方便对地理上分布的研究人员建立虚拟组织，进行跨学科的虚拟合作。目前，Globus 项目把在商业计算领域中 Web Service 技术融合在一起，希望不仅仅局限于科学计算领域，而且能够对各种商业应用进行广泛的、基础性的网格环境支持，实现更方便的信息共享和互操作，从而对商业模式、人的工作方式和生活方式产生深远的影响。

Globus 项目是美国 Argonne 国家实验室的研发项目，在初始阶段，全美有十多所大学和研究机构参与了该项目的研究工作。Globus 对信息安全、资源管理、信息服务、数据管理以及应用开发环境等网格计算的关键理论和技术进行了广泛的研究，开发出能在多种平台上运行的网格计算工具包软件（Globus Toolkit），能够用来帮助规划和组建大型的网格试验和应用平台，开发适合大型网格系统运行的大型应用程序。Globus 工具包是 Globus 最重要的实践成果，其第一版在 1999 年推出，其后的主要版本有 1.1.3 版和 1.1.4 版，目前最新版本 2002 年初是推出 2.0 版，并将于 2002 年末到 2003 年初推出基于 OGSA 体系结构，并且融合了 Web Service 技术的 Globus 工具包 3.0 版[1]。

Globus 工具包的源码向公众开放，遵循 Globus Toolkit Public License (GTPL) 协议，任何人都可以从其网站上下载源代码并进行研究和改进。这里需要注意的是其 GTPL 协议与开源软件的 OpenSource 协议、自由软件的 GNU General Public LicenseGPL 协议有一定的不同之处，所以在使用 Globus 工具包的时候，首先应该分析一下 GTPL 协议。目前，Globus 工具包已在 NASA 网格(NASA IPG)、欧洲数据网格（Data Grid）、美国国家技术网格(NTG)等众多项目中得到应用。

4.1.2 Globus 对网格计算的理解

根据 Globus 的观点，在网格计算环境中，所有可用于共享的主体都是资源，如计

算机、高性能网络设备、昂贵的仪器、大容量的存储设备、各种科学数据、各种软件等是资源，分布式文件系统、数据库缓冲池等也可以理解为资源。实际上，资源这个概念有一点含糊，只要在网格计算环境中对用户存在利用价值的东西都可理解为资源。Globus 实际上关心的不是资源的实体本身，而是如何把资源安全、有效、方便地提供给用户使用，所以从共享的角度考虑，对 Globus 而言，其主要研究资源的访问接口或访问界面。

通常的网格计算主要侧重于大型的分布式应用，而根据 Globus 的观点，大型应用项目应该由许多组织协同完成，这些组织通过网格计算环境形成一个统一的“虚拟组织”－Virtual Organization，网格计算环境中的用户、成员、资源可随时加入虚拟组织。在网格计算环境中，各组织拥有的计算资源、存储资源等各种资源可以被虚拟组织中的成员共享，并且各成员可方便地协同完成各种分布式应用和工作。按照这种理念，在网格计算环境中，各成员和组织之间的存在的时间、拥有的权限和资源的数量、种类等都会不断动态地发生变化，这使得虚拟组织中的实体（包括用户、成员、资源、组织等）需要保持一种非常动态的共享关系。所以如何有效地对虚拟组织和其中的成员进行管理是 Globus 需要研究的一个重要课题。

与 Globus 的目标相比，现有的数据共享和互操作方案，比如 P2P、COM/DCOM、CORBA、DCE，J2EE 等，在共享配置的灵活性、动态性和在共享资源种类上不能完全满足动态虚拟组织的需要。同时，Globus 并不试图取代现有技术，而是希望在现有技术之上建立更高层次的共享。从技术的角度讲，共享是资源或主体间的互操作，比如用甲计算机的分析程序调用乙计算机的数据库并把图形显示在丙计算机上。Globus 认为，要实现网络环境中的互操作就意味着：首先需要开发一套支持网格计算的通用协议（如网格计算安全协议、网格计算的数据传输协议、网格计算的信息获取协议等），用它来描述消息的格式和消息交换的规则；然后在这些协议之上，需要开发一系列支持网格计算的服务（如网格安全服务、网格信息服务、网格数据传输服务等），这与建立在 TCP/IP 协议上的 Web 服务原理相同；由于需要通过具体的软件实现这些服务，这就需要定义 API，基于这些 API 再构建各种软件开发工具（SDK）（如 Globus 工具包和 CoG 开发工具包等）。

所以 Globus 项目的主要工作就是建立支持网格计算的通用协议，开发支持网格计算的服务，实现支持网格计算环境的软件开发工具。

4.2 Globus 系统结构

为了有效地支持网格计算环境[27]，Globus 工具包针对 Globus 项目中提出的各种协议[28]，提供了一系列的服务（service）、软件库、编程接口（API）和使用例子[10]。从总体上讲，Globus 工具包的实现主要有四方面的内容：（1）网格安全，这是网格计算环境正常运行的保证，Globus 主要是结合目前成熟的分布式安全技术，并进行一定的扩展，以适合网格计算环境的特点；（2）网格信息获取与分布，在网格计算环境中如何发布资源信息，如何查询、检索资源信息是有效使用各种资源的前提条件；（3）网格资源管理，由于网格环境中的资源主要分布在广域网环境中，采用目前常用的局域网资源管理技术不能有效地对其进行管理，为此 Globus 在局域网资源管理之上实现了更高层次的资源管理技术，在信息服务的支持下，可有效地支持广域范围内的资源管理；（4）网格远程数据传输，实现广域网环境下的高速、可靠的数据传输和实现对应用程序基本透明的远程文件 I/O 访问是 Globus 考虑的重要内容。

上述四方面的技术可以使得开发在网格计算环境下的应用更加方便，而且使得网格

应用程序的执行效率会更好。针对上述四个方面的内容，Globus 项目实现的主要组成部分介绍如下（如图 4-1所示）：

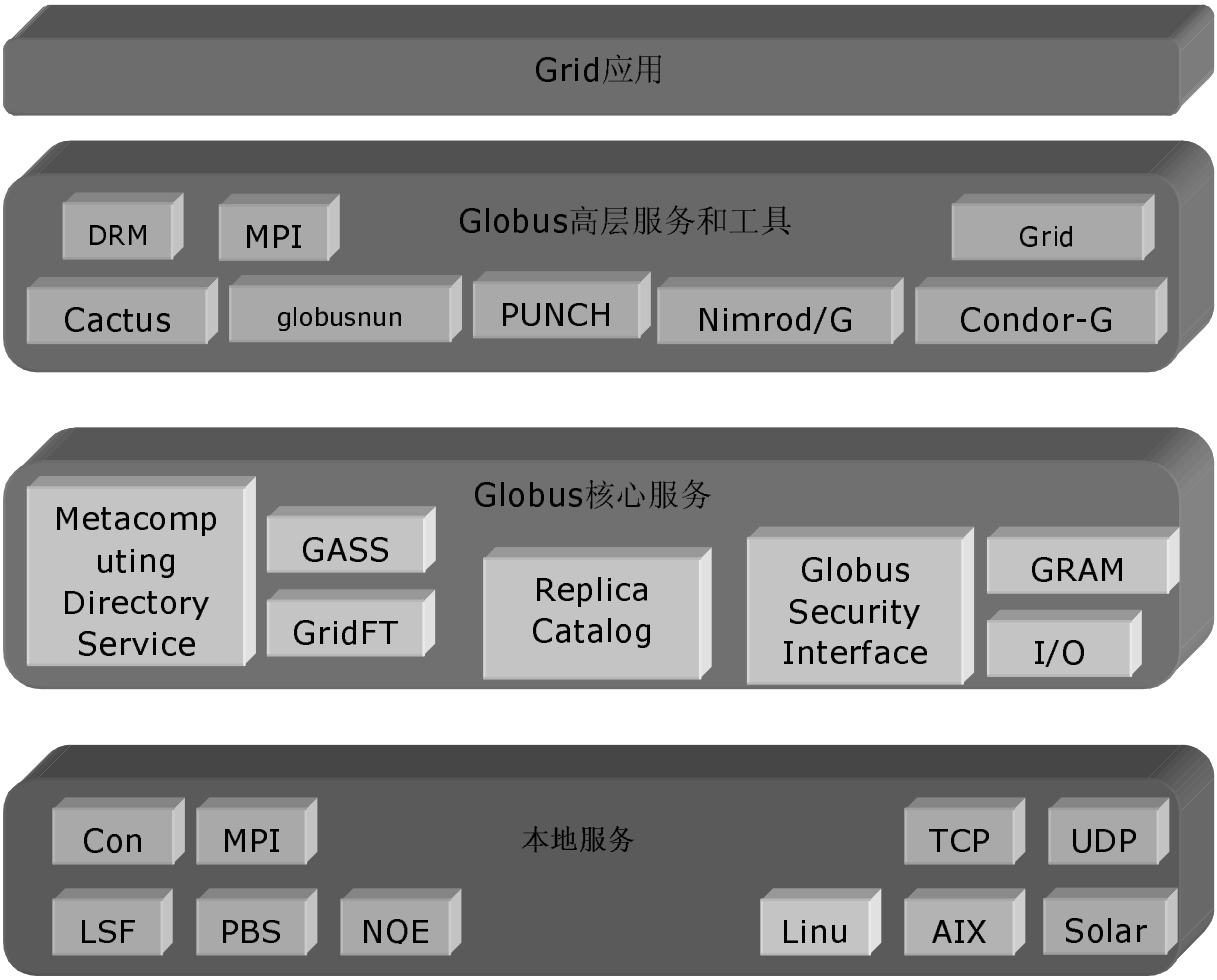


图 4-1 Globus 工具包 在 网格计算逻辑结构中组成部分

1 网格安全基础设施—Grid Security infrastructure (GSI)：GSI 负责在广域网络下的安全认证和加密通信，提供单点登录功能、远地身份鉴别功能、数据传输加密功能等，提供了基于 GSI 协议的 Generic Security Services API（GSS-API）接口。是保证网格计算环境安全性的核心部分。

2 Globus 资源分配管理—Globus Resource Allocation Manager (GRAM)：GRAM 负责远程应用的资源请求处理、远程任务调度处理、远程任务管理等工作，负责对 Resource Specification Language (RSL)信息的解析和处理工作，是网格计算环境中的任务执行中心。

3 元计算目录服务—Metacomputing Directory Service (MDS)：MDS 主要完成对网格计算环境中信息的发现、注册、查询、修改等工作，提供对网格计算环境的一个真实、实时的动态反映。主要基于 Lightweight Directory Access Protocol（LDAP）协议，其处理的信息主要是网格计算环境中的各种资源（包括数据资源、计算资源等）、服务和其它主体（entity）的描述。MDS 是网格计算环境中的信息服务中心。

4 全局二级存储服务—Global Access to Secondary Storage (GASS)：GASS 简化了在 Globus 环境中应用程序对远程文件 I/O 的操作，使得使用 UNIX 和标

准 C 语言 IO 库的应用程序基本不用改动就可在 Globus 环境中执行。GASS 是一个支持网格计算环境远程 I/O 访问的中间件。

5 网格 FTP 服务—GridFTP：GridFTP 是一个高性能、安全、可靠的数据传输协议，并针对高带宽的广域网络环境进行了优化。具有支持第三方传输、断点续传、并行传输、与 GSI 结合的安全认证、缓存等特性。是网格计算环境中的数据传输工具。

6 Globus 复制管理—Globus Replica Management：复制管理是一大类科学应用程序中需要考虑的重要问题，由于存在对大型远程文件的访问。Globus Replica Catalog 通过把部分相关数据智能地放置在离科学应用程序最近的位置，使得科学应用程序可快速地对数据进行访问。

4.3 小结

本章主要介绍了 Globus 的发展过程、Globus 项目对网格的理解和 Globus 的主要组成部分。从中可以看到 Globus 项目正逐渐得到广泛的支持，并且 Globus 工具包目前已逐步成为主要的网格开发工具和网格构建平台。Globus 项目目前正在飞速发展，并与 Web Service 技术结合[4]，提供更有效的各种资源和信息访问手段。

思考题

1 Globus 项目的起因是什么？

2 Globus 工具包的实现目标是什么？

3 Globus 系统结构中主要包括哪些组成部分？

4 Globus 工具包 3.0 的特点是什么（可从[2]中查找）？

第5章 网格安全基础设施

本章主要对 Globus 项目的网格安全基础设施－GSI（Grid Security Infrastructure）进行全面的介绍[25][26][31]。首先对目前的分布式安全技术进行了一个简要的描述和比较；然后讲述了网格安全基础设施的需求和目标；并且对 Globus 项目中的网格安全基础设施的技术特点进行了介绍；接下来对 Globus 项目的安全策略设计细节进行了分析；最后对 Globus 项目的安全策略的实现和基于 Globus 工具包的 API 进行了简单的介绍。

5.1 分布式安全技术

TCP/IP 协议群在网际互联中的使用的迅速崛起，导致了通常被称为 Internet 的由主机和网络组成的全球网际互联系统。今天，一个与 Internet 相连的信息网络正面临着来自多方面的安全威胁：

技术缺陷 复杂信息系统中的任一部分发生问题都会危及整个系统的安全，这除了要求采用高可靠性的部件外，还需要在设计上采取适当措施，例如当系统的某些部分出错或失效时仍能正常运行。

外界入侵 这包括无组织或自发的入侵活动，即黑客和病毒的入侵，也包括有组织或敌意的入侵活动，即信息战。而信息战在规模和危害程度上更为严重，它可以使敌对方的机密信息外泄或信息网络瘫痪，达到不战而胜的目的。在外界入侵方面还有一种特殊的威胁是来自人为的"陷井"，特别是在信息系统的核心部件中，如果不是自主技术，那么就有可能被开设"秘密通道"或埋藏"信息炸弹"，这实际上相当于一种特殊的间接入侵或破坏性入侵。

内部泄密 内部人员有意无意把机密信息外泄也是需要防范的问题。信息网络必须具备先进的监测、管理手段，并建立完善的法规和管理制度，对工作人员也必须加强教育，以防止或减少这类事件的发生。

用来为 Internet 提供访问控制服务和通信安全服务的安全技术研究已经进行了多年。如防火墙技术和网际、传输、应用等各层上的安全协议。

网络层(IP)安全机制的主要优点是它的透明性，即安全服务的提供不要求应用做任何改变。而对传输层来说是应用不作修改是做不到的，原则上，任何 TCP/IP 应用，只要应用传输层安全协议，比如说 SSL 或 PCT，就必定要进行若干修改以增加相应的功能，并使用稍微不同的 IPC 界面。所以，传输层安全机制的主要缺点就是要对传输层 IPC 界面和应用程序两端都进行修改。可是，比起网络层和应用层的安全机制来，这里的修改还是相当小的。不过，这种方式还存在一个缺点是：基于 UDP 的通信很难在传输层建立起安全机制来。同网络层安全机制相比，传输层安全机制的主要优点是它提供基于进程对进程的(而不是主机对主机的)安全服务。这一成就如果再加上应用级的安全服务，就可以再向前跨越一大步了。表格 5-1是不同网络层次安全的一个简单比较。

表格 5-1 网络层次安全比较

	网络层	传输层	应用层
实现方法	修改网络层	修改传输层	修改应用层
与上层关系	透明	不透明	不透明

修改难度	大	中	小
安全关系	主机对主机	进程对进程	服务对服务

目前的安全技术相当繁多，与 GSI 相近的分布式安全技术主要有 Kereros、DCE 和 Secure Shell 等。Kerberos 在 80 年代中期被广泛使用。但目前的版本依然基于传统的 AS/TGS 加密技术。尽管某些 Kerberos 扩展，如 PKINT、PKTAPP、PKCROSS 等，支持公钥加密体系，Kerberos 的执行开销还是很大，适合于小规模范围内的安全保护管理，对基于跨 Internet、系统动态改变的网格计算环境并不很适合。DCE 是由 Open Group 开发的一个成熟的分布式计算环境。它的安全保密技术主要基于传统的第三方 shard-key 加密体系，同样只适合于小规模范围内的安全保护管理。SSH 广泛应用于 Unix 系统的安全登录，它满足网格计算环境中的某些需求，提供加密的远程登录和远程数据拷贝，而且它是基于公钥安全体系，安装使用方便。但它需要用户拷贝管理跨节点的认证关系，并需要用户拷贝其公钥到所有用户需要访问的节点，这比较麻烦。而且 SSH 不提供基于站点（site）的授权控制，这可能导致用户的非法访问。另一方面是 SSH 的能力有限，对其它一些服务（如协同环境和 Web Browser）不能提供有效的身份鉴别。

当然其它的分布式安全技术还很多如网络安全入侵检测、网络防火墙等。尤其是防火墙技术，已广泛应用在从大企业、公司到用户个人的计算机系统中。防火墙的作用是在一个被认为是安全和可信的内部网络和一个被认为是不那么安全和可信的外部网络 Internet 之间提供一个封锁工具。假如没有防火墙，内部网络就暴露在不那么安全的 Internet 协议和设施面前，面临来自 Internet 其他主机的探测和攻击的危险。此时，根据木桶原理，整个内部网络的安全性等于该内部网络内所有主机中安全性最低的哪一台主机的安全性。那么这个内部网络越大，安全性就越难管理。随着安全性问题上的失误和缺陷越来越普遍，对网络的入侵不仅来自高超的攻击手段，也有可能来自配置上的低级错误或不合适的口令选择。因此，防火墙的作用是防止不希望的、未授权的通信进出被保护的内部网络。尽管存在争议，防火墙的拥护者和批评者都承认，防火墙不能替代谨慎的安全措施。防火墙在当今 Internet 世界中的存在是有生命力的。它是一些对高级别的安全性有迫切要求的机构出于实用的原因建造起来的，因此，它不是解决所有网络安全问题的万能药方，而只是网络安全政策和策略中的一个组成部分。

5.2 网格计算环境的安全需求和 Globus 的安全目标

从本质上讲，Internet 的安全保障一般提供下面两方面的安全服务：（1）访问控制服务，用来保护各种资源不被非授权使用；（2）通信安全服务，用来提供认证，数据保密性与完整性和各通信端的不可否认性服务。但这两方面的安全服务不能完全解决网格计算环境下的安全问题。网格计算环境必须能够满足用户安全、高效地使用其提供的各种资源的要求。同时，为了用户使用的方便，网格计算环境必须连接在 Internet 上并提供方便使用的服务供用户使用。为此，网格计算环境必须具有抗拒各种非法攻击和入侵的能力，并且在受到攻击和入侵时采取某些措施以维持系统的正常高效运行和保证系统中各种信息的安全。因此，在网格计算环境中，安全问题比一般意义上的网络安全问题的覆盖面更广，解决方案也更加复杂。

在网格计算环境中，各种资源都动态连接到 Internet 上，而且不同网格节点之间的通信是通过 Internet 连接的，而用户向网格计算环境提交任务和监控管理任务也是通过 Internet 来完成的。同时网格计算环境中的所有主体都可以动态加入或撤离网格中的虚拟组织，从而使得网格计算环境对安全的要求除 Internet 的安全要求外更进了一步。简

而言之，在一个网格计算环境中的网格安全体系必须考虑网格计算环境的如下特性：（1）网格计算环境中的用户数量很大，且动态可变；（2）网格计算环境中的资源数量很大，且动态可变；（3）网格计算环境中的计算过程可在其执行过程中动态地请求，启动进程和申请、释放资源；（4）一个计算过程可由多个进程组成，进程间存在不同的通信机制，底层的通信连接在程序的执行过程中可动态地创建并执行；（5）资源可支持不同的认证和授权机制。（6）用户在不同的资源上可有不同的标识；（7）资源和用户可属于多个组织。正是由于网格计算环境的特殊性，所以在设计网格安全机制时特别要考虑网格计算环境的动态主体特性，并要保证网格计算环境中不同主体之间的相互鉴别和各主体间通信的保密性和完整性。

Globus 项目通过提出网格安全基础设施－GSI，来提供在网格计算环境中的安全认证和安全通信等能力。为了保证网格计算环境的安全，GSI 的主要目标是：（1）支持在网格计算环境中主体之间的安全通信，防止主体假冒和数据泄密；（2）支持跨虚拟组织的安全，这样就不能采用集中管理的安全系统；（3）支持网格计算环境中用户的单点登录，包括跨多个资源和地点的信任委托和信任转移等。为此，GSI 为网格计算环境提供了一系列的安全协议、安全服务、安全 SDK 和命令行程序，如安全应用编程接口、相互安全身份鉴别技术、单点登录（single sign-on）技术等。通过使用这些安全技术，可有效地保证网格计算环境的安全性和方便性。

5.3 网格安全基础设施的安全技术

通过综合考虑，Globus 项目中的 GSI[12]主要集中在网络的传输层和应用层，并强调与现有分布式安全技术的融合。GSI 基于公钥加密体系，采用 X.509 认证和 Secure Sockets Layer （SSL）通信协议，并对它们进行了一定的扩展，使得 GSI 可以支持单点登录。而且 GSI 的实现符合 Generic Security Service API （GSS-API），而 GSS-API 是由 Internet Engineering Task Force (IETF)提出的用于安全系统的标准 API。GSI 中的主要安全技术手段包括安全认证、安全身份相互鉴别、通信加密以、私钥保护及委托与单点登录等。下面对这些特点进行逐一介绍。

5.3.1 安全认证

GSI 认证（certificates）的一个关键点是认证证书。在网格计算环境中的每个用户和服务都需要通过认证证书来验明正身，我们可以把认证证书类比为国家公民的身份证。为了防止对认证证书的假冒和破坏，GSI 认证证书主要包含四部分信息：

- 主体名称（subject name）：用来明确认证证书所表示的人或其它对象。
- 属于这个主体的公钥（public key）：用于 X.509 认证
- 签署证书的认证中心标识：其记录了认证中心的名称
- 签署证书的认证中心的数字签名：可用来确认认证中心的合法性

GSI 证书采用了 X.509 的证书格式，可被其它基于公钥的软件（如 Internet Explorer、Netscape 等）共享。下面是一个签署过的安全认证证书的例子：

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 28 (0x1c)
```

```
        Signature Algorithm: md5WithRSAEncryption
        ***********认证中心名称*********
        Issuer: C=CN, O=Globus, CN=Globus Certification Authority
        ***********证书的合法使用时间*********
        Validity
            Not Before: Apr 22 19:21:50 1998 GMT
            Not After : Apr 22 19:21:50 1999 GMT
        ***********主体名称*********
        Subject: C=CN, O=Globus, O=TSINGHUA, OU=HPCLAB, CN=Yu Chen
        ***********公钥信息*********
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:bf:4c:9b:ae:51:e5:ad:ac:54:4f:12:52:3a:69:
                    <snip>
                    b4:e1:54:e7:87:57:b7:d0:61
                Exponent: 65537 (0x10001)
***********数字签名*********
Signature Algorithm: md5WithRSAEncryption
        59:86:6e:df:dd:94:5d:26:f5:23:c1:89:83:8e:3c:97:fc:d8:
        <snip>
        8d:cd:7c:7e:49:68:15:7e:5f:24:23:54:ca:a2:27:f1:35:17:
```

5.3.2 安全身份相互鉴别

如果双方主体都有证书，而且都信任彼此的认证中心，则双方可相互明确彼此的身份,这实际上是相互鉴别(mutual authentication)问题。GSI 采用 SSL(Secure Sockets Layer) 协议作为它的相互认证协议。

在相互认证可以进行之前，双方首先要相信彼此的认证中心。在实现上，要求双方有彼此认证中心自身的证书，认证中心自身的证书中包含认证中心的公钥。这样才能确保双方由认证中心签署的证书具有合法性。

简单的安全身份相互鉴别过程描述如下：为了进行相互鉴别，A 方与 B 方首先建立一个连接，然后 A 方给 B 方自己的证书。A 方的证书告诉 B 方 A 是谁、A 的公钥是什么、签署 A 证书的认证中心是谁。B 方收到证书后，B 方首先要通过检查认证中心的数字签名来确认证书是合法的。一旦 B 检查了 A 的证书的合法性，B 需要确定 A 是其认证证书所指的主体。为此，B 生成一个随机信息并把它发给 A，要求 A 对这个信息进行加密。A 用自己的私钥加密信息后，把加密信息发给 B。B 用 A 的认证证书中 A 的公钥对加密信息进行解密。如果解密的结果与初始的信息一致，则 B 就可以信任 A 了。同理，可反向采用上述过程，使得 A 信任 B。这样 A 和 B 就可以相互信任，并建立安全连接通道。

5.3.3 通信加密

在缺省情况下，GSI 在通信双方之间不建立加密通道。一旦相互认证成功，则 GSI 就脱离出来，这样通信双方在进行通信时不会有额外的加解密开销。如果通信双方需要进行加密通信，则 GSI 可很容易地建立一个共享的密钥用于对信息进行加解密，这里一般采用公钥技术与对称加密技术结合的加密方式，在保证安全性的同时尽量减少加解密的开销。另一个需要 GSI 考虑的是通信的完整性。通信完整性表示窃听者可能会了解到通信双方的通信内容，但不能对通信内容进行修改。GSI 在缺省情况下保证通信的完整性，这会带来一定的开销，但这比加解密开销要小得多。

5.3.4 私钥保护

在一般情况下，GSI 要求 Globus 用户的私钥保存在本地计算机的一个文件中。为了阻止本地计算机的其它用户窃取私钥，此文件必须经过一个用户知道的口令（password）进行加密和保护。这样用户在使用 GSI 中的认证证书时，必须输入口令来解密包含私钥的加密文件。GSI 还可以采用外部介质如加密的 smartcard 来保存私钥，这样使得其他人窃取私钥的难度大大增加。

5.3.5 安全委托与单点登录

在通常情况下，如果用户之间或用户与资源管理者之间在建立联系之前，都必须通过相互鉴别的过程。这样，如果一个用户要与多个资源管理者进行联系的话，就必须多次访问保存私钥的文件，即需要多次输入密码，而且在代表用户的程序的运行过程中，也可能进行相互认证，使得相互认证的过程很烦琐。

GSI 对标准的 SSL 协议进行了扩展，使得 GSI 具有安全委托能力，减少了用户必须输入口令来得到私钥的次数。如果一个网格运算需要用多个网格资源，或者说需要一个代表用户的代理来请求资源，GSI 通过创建代理（proxy）来避免输入口令，这样可以在不同的节点之间形成一个安全信任链，如图 5-1所示。

一个代理包含一个新的证书，这个新的证书由用户来签署，而不是认证中心。证书中有新的公钥和私钥、用户的标识、代理标记和时间戳，时间戳表示在一定时间范围内，这个代理是有效的。

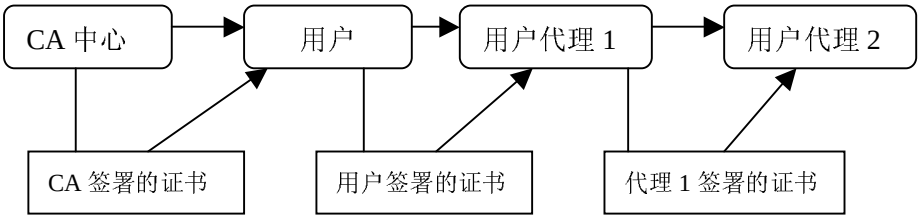


图 5-1 基于 GSI 安全代理的安全信任链

代理的私钥必须保证安全，但由于代理的有效性是有时间限制的，所以限制了代理

的私钥不可能象用户的私钥那样安全。一个比较简单的处理方法是把代理的私钥保存在本地的文件系统中，对私钥不进行加密，不允许其它用户对其进行访问。一旦代理被创建并存储，则用户可以使用代理证书和使用进行相互鉴别，并且不需要输入口令。当采用代理进行相互鉴别时，鉴别过程与用户和代理之间的鉴别过程有所不同，被请求方将会收到用户的证书和代理的证书（由用户签署）。因为在鉴别过程中，用户证书中用户的公钥用来确认代理证书中数字签名的合法性，这样实际上形成了一个认证中心－用户－代理的信任链。

目前 GSI 和基于它的软件（包括 Globus 工具包、GSI-SSH 和 GridFTP）是唯一支持委托扩展的 SSL，Globus 项目正在努力使 GSI 的委托扩展成为 SSL 的标准扩展，并希望得到 IETF 的确认。

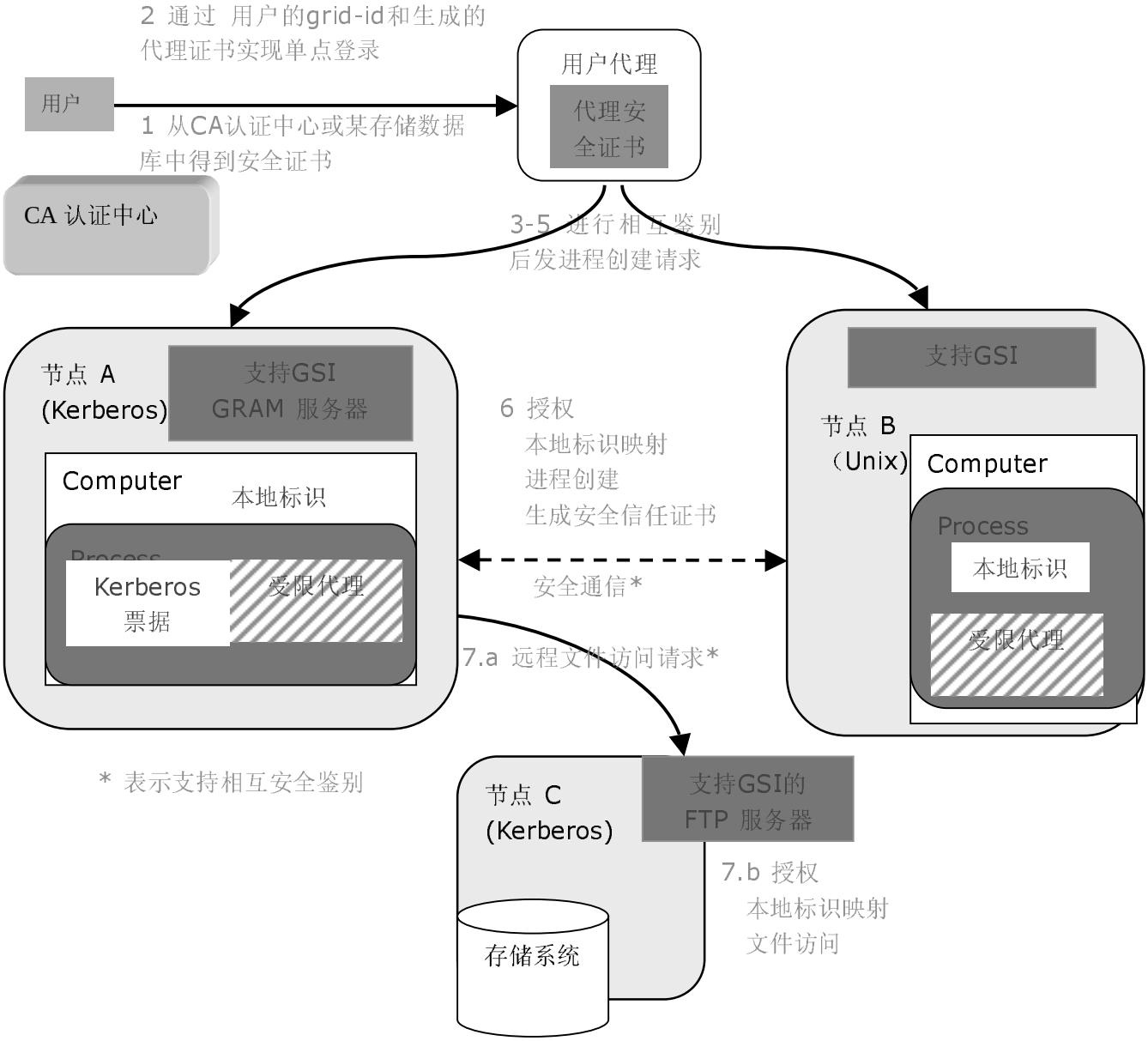


图 5-2　Globus 环境中的安全鉴别过程

5.4 基于 GSI 的任务提交与执行过程描述

由于 Globus 的 GSI 对安全处理做了大量的工作，使得在网格环境中的任务提交与执行的安全性和方便性有了很大的提高。下面上进行安全任务提交与执行的一个简单描述（任务的安全认证过程如

图 5-2所示），从中可以看出 GSI 在网格环境中的作用：

1 在进行 globus 任务提交与执行之前，用户与服务节点和服务设施需要获得安全认证证书。即可使用命令行命令 grid-cert-request 或对应的安全函数创建用于安全鉴别的公钥、私钥和未签署的安全证书。然后通过 email 或其它安全途径把提交安全认证中心。

2 安全认证中心收取签署安全认证证书的请求后，会对用户或服务节点进行考察，考察合格后，把签署过的安全认证证书返回给请求方。

3 用户在提交任务前，即可通过命令行命令 grid-proxy-init 或对应的安全函数创建一个临时的，局部的，有时间期限的用户代理（proxy），用户代理的安全证书由用户签署，有给定的有效期。

4 用户代理与远端服务节点的 gatekeeper 之间进行相互安全鉴别，即对二者的安全证书和身份进行鉴别。

5 通过相互安全鉴别后，用户代理把任务提交给 gatekeeper，由 gatekeeper 把任务交给任务管理者进行具体的处理。

6 如果任务在执行过程中需要其它的远程资源，也必须在任务进程与资源代理之间进行相互安全鉴别，通过安全鉴别以后，任务进程才可以使用资源。

7 如果任务在执行过程中需要访问远端数据或文件，也必须在任务进程与远端文件服务资源代理之间进行相互安全鉴别。通过安全鉴别以后，还要进行授权、本地 id 映射后，任务进程才可以进行远端数据或文件访问。

8 当任务执行完后，用户可通过 grid-proxy-destory 或对应的函数撤销用户代理。

5.5 GSI 的安全策略设计思想

本节主要描述了 Globus 中各种交互情况下的安全策略设计思想[15]。首先对一些相关的术语进行了介绍，然后介绍了 GSI 要达到的安全要求，最后对主要的 GSI 安全问题进行了讨论，这些 GSI 中的安全问题主要包括（1）用户与用户代理之间的安全关系：由于安全代理机制是 GSI 很有特点的一个地方，是实现单点登录的关键，所以本节对其策略设计有较详细的介绍。（2）Globus 主体与本地资源主体的映射：由于用户或资源在网格计算环境和一般的局域网环境下的身份描述是不一致的，所以在对用户或资源进行安全身份认证和其它处理之前，首先要在这两种身份之间建立某种映射关系（即某种对应联系），这样才能进行各种本地安全处理。（3）资源代理与资源分配安全：由于资源分配与安全紧密相关，所以在介绍资源管理之前，先把它与 GSI 的相关部分－－资源代理的安全策略做一简要介绍。（4）进程与用户代理之间的安全鉴别：进程之间的相互安全鉴别与用户之间的相互安全鉴别有不同之处，它主要考虑如下交互情况：节点 a 的 A 用户在节点 b 创建了 B 进程，然后 B 进程要求创建在节点 c 上的 C 进程或其它资源。所以在这里也单独进行了分析。（5）网格信息服务的安全保证：网格信息服务主要保证只有合法的用户才能访问信息，只有合法的资源才能提供信息。这里对其进行了简单介绍。

5.5.1 相关术语和安全策略介绍

在 Globus 安全策略设计中采用了如下术语:

- 主体（**Subject**）：在 Globus 环境中，一个主体一般指用户或代表用户的进程。一个 Globus 主体是被 Globus 系统中的组件识别的主体；一个资源主体是指一个被特定资源识别的主体。
- 证书（**Credential**）：又可称为信任状，证书是用来证明一个主体（subject）身份的一段信息，一般包括口令和私钥等。
- 信任（**Trust**）：如果主体 A 假设主体 B 相信任何由主体 B 签署的信息这个前提是正确的，则称 A 信任 B。
- 信任域（**Trust Domain**）：一个信任域指在一个本地安全策略维护下的管理结构。例如在一个信任域下的主体之间不需要鉴别。

Globus 安全策略定义了在网格计算环境中一个主体能够对另一个主体做什么。下面介绍了 Globus 安全策略中的五个基本组成部分：

- 用户 **User**（**U**）：一个 Globus 计算过程的请求者，可以人或代理（一个进程）。
- 用户代理 **UserProxy**（**UP**）：在一个有限时间内代表用户行使一定权限的一个进程。
- 进程 **Process**（**P**）：一个逻辑主体，提供进程管理 API 创建，并代表一个用户在特定资源上进行计算。
- 资源 **Resource**（**R**）：在一个计算过程中使用的计算节点、文件系统、网络或其它主体。
- 资源代理 **ResourceProxy**（**RP**）：一个有一定权限的资源管理者。

在 globus 环境中，目前的 GSI 主要考虑的问题是各种主体间如何进行身份确认和鉴定，并且针对各种情况初步定义的主要安全策略包括：（1）用户代理和资源代理之间的所有连接需要安全鉴定；（2）所有的安全鉴定是相互的；（3）一个用户代理在一个限定时间内可代表一个用户与一个资源代理进行交互；（4）具有相同资源的资源代理之间相互信任；（5）一个进程管理者是一个负责创建进程的资源代理，而一个进程信任创建它的进程管理者；（6）一个进程管理者和它创建的进程假设在一个单一的信任域中执行；（7）由相同的用户代理/资源代理创建的所有进程之间彼此信任；（8）资源的存取控制权限由其所处环境中的本地安全策略确定，并通过本地安全机制来实现；（9）所有的 Globus 特定主体可通过映射到一个资源特定主体（如用户 ID）来实现本地存取控制机制。

除了上述的安全策略，从易用性、灵活性、可移植性和可靠性等方面考虑，GSI 还需要满足如下的安全需求：（1）Globus 环境中，用户只需登录一次（即单点登录），就可以进行请求、使用和释放资源等操作，不需要进行多次身份认证。（2）用户的证书保存有口令，密钥等关键信息，必须有完善的保护措施。（3）Globus 安全策略必须与 Globus 环境中某节点的本地安全策略、机制和实现进行交互，这样能够保证对各种本地资源的安全性。（4）Globus 的安全策略不局限于某一个特定的安全实现技术，而是可同时适合多种安全技术，如 X5.09、PGP 公钥技术、Kerberos 技术等。（5）Globus 安全策略在 X.509 和 Kerberos 之上建立了一个抽象接口，以增强 GSI 的可移植性。（6）Globus 中的安全算法可保证 Globus 的组件在重新执行的情况下维持以前的安全性，这需要把组件的系统状态保存在本地存储空间中。

5.5.2 用户与用户代理之间的安全关系

在 Globus 环境中，存在着动态变化的各种计算过程。当某个计算过程需要申请某个资源时，它必须具有某个“有资格用户”的对应权限（即某个计算过程属于某个用户）。但是在一个 Globus 计算过程中，通常不可能要求用户直接与计算所需资源进行安全交互。为了解决这个问题，Globus 引入了用户代理的概念，即通过用户代理代表用户与资源进行安全交互，这样用户不需直接参与安全交互了。同时从安全角度考虑，用户代理的合法性要有一定的时间限制，这在一定程度上提高了 Globus 环境的安全性，并可适合 Globus 环境动态性的特点。

用户代理代表用户的最直接方法是把用户的安全证书（如口令、私钥等）转移给用户代理，这样用户代理可直接与各种资源进行安全交互。这种方法的问题在于用户代理可能会泄密用户的安全证书，而且这种方法不允许用户限制用户代理的合法存在时间。所以采用这种方法是不合适的。

Globus 通过生成用户代理的临时安全证书的方法来解决上述问题。首先，用户负责签署用户代理的临时安全证书，然后设定用户代理的临时安全证书的合法存在时间。这样经过用户签署的临时安全证书说明用户同意用户代理可代表用户与资源代理进行安全交互，且限制了用户代理的合法存在时间，保护了用户的安全信息。创建用户代理的具体过程如下：

1. 用户首先获得将要创建用户代理的节点的使用权；
2. 用户为用户代理创建临时安全证书，用户代理的临时安全证书用 $C'UP$ 表示；
3. $C'UP$ 由用户进行签署，签署的安全证书表示为

   $C_{UP} = C_U\{C'_{UP},\ start\text{-}time,\ end\text{-}time\}$

   C_{UP} 包含临时安全证书和证书合法存在的时间段，并经过了用户的签署；
4. 创建用户代理，并通过用户签署的临时安全证书提供安全身份保证。

5.5.3 Globus 主体与本地资源主体的映射

为了执行本地访问安全控制，资源代理要把一个 Globus 主体映射为一个本地资源“知道”的主体。所以要把主体的全局名称转换成本地的局部名称。为了完成名称映射，资源代理必须能够对 Globus 证书、Globus 主体、资源证书和资源主体进行访问。

映射算法的基本思想是通过进程管理者完成对 Globus 安全证书和本地安全证书或本地安全帐号的访问并建立映射关系。映射算法描述如算法 5-1所示：

算法 5-1　Globus 主体与本地资源主体的映射算法

1. 用户代理与进程管理者进行安全鉴别。 2. 用户代理向进程管理者发出经过签署的 *MAP-SUBJECT-UP* 请	1. 用户通过资源的鉴别方法对资源进行记录，然后启动一个映射注册进程。 2. 映射注册进程发出 *MAP-SUBJECT-P* 请求给进

求，并把 Globus 主体和资源主体作为参数	程管理者，并把 Globus 主体和资源主体作为参数。

3. 进程管理者调度 *MAP-SUBJECT-UP* 和 *MAP-SUBJECT-P* 请求以及对应的参数。
4. 进程管理者确保映射注册进程属于映射请求中指定的资源。.
5. 如果二者匹配,则进程管理者建立映射并向映射注册进程和用户代理发送响应信息。
6. 如果在 MAP-TIMEOUT 时间没有发现匹配，则进程管理者清除未完成的请求，并对等待的进程发送响应。
7. 如果在 MAP-TIMEOUT 时间内没有收到响应信息，则请求被认为失败。

这里需要注意的是：一个匹配的 *MAP-SUBJECT-P* 请求必须从用户代理和映射进程发出。这主要是为了确保同一用户拥有 Globus 证书和本地证书。如果把映射算法的结果放入一个资源代理可访问的数据库中，则对每个资源只需执行一次映射算法。映射有效的持续时间由本地系统管理策略决定，而且本地系统管理策略一般需要在 Globus 安全证书或用户本地帐号的有效生命期内保持这个映射关系。

5.5.4 资源代理与资源分配安全

在 Globus 环境中，对资源的操作是通过资源代理进行仲裁的，资源代理负责调度资源的使用，而且负责把计算映射到资源上，具体细节可参见“资源分配管理”一章。简单的资源请求过程如下：请求资源的用户（更精确地说是代表用户的用户代理）首先要确定资源代理的身份；然后用户向资源代理发出资源请求。如果请求成果，则一个资源分配标记返回给用户，这个标记保证了用户有权利向进程管理者提出请求，要求进程管理者在这个资源上创建一个进程。

资源分配请求可把资源分配说明映射为 *PROCESS-HANDLE*。*PROCESS-HANDLE* 的定义如下：

$$PROCESS\text{-}HANDLE = CRM\{host\text{-}identifier, process\text{-}identifier\}$$

这里 host identifie 和 process identifier 由进程管理者定义并由资源代理进行签署。资源分配请求失败的三种原因有：（1）请求的资源不存在；（2）用户不是资源的合法使用者（鉴别失败）；（3）用户没有权利使用请求的资源（授权失败）。

授权不属于 Globus 安全策略的范围，这主要由资源代理通过本地授权需求具体执行。根据资源和本地安全策略的特性，授权可在资源分配时间、进程创建时间等进行检查，也可能根本不检查。

资源代理不一定位于资源调度的节点，但安全策略要求进程管理者和被管理的进程位于相同的信任域中，Globus 资源代理采用的算法可满足这个要求。为了简化讨论，假设有一个用户代理向资源代理发出了所有的资源请求。扩展下面的算法可处理由任何用户进程发出的资源管理请求。用于资源分配的安全算法描述如下：

算法 5-2　用于资源分配的安全算法

1. 用户代理和资源代理使用 C_{UP} 和 C_{RM}进行相互鉴别。资源代理要确保用户代理的临时安全证书没有过期。

2. 用户代理向资源代理提交签署的请求，请求的形式如下：

   $C_{UP}\{allocation\ specification\}$

3. 资源代理检查用户签署的用户代理临时安全证书，并检查用户是否可通过本地安全策略的授权。如果资源代理在资源分配阶段检查授权，这可能需要把用户安全证书中的用户标识映射到一个本地用户 id 或帐号名。当然，授权检查也可推迟到进程创建时间。

4. 如果资源请求成功，则资源代理构造一个 *PROCESS-HANDLE* 并返回给用户代理。

5.5.5 进程与用户代理之间的安全鉴别身份转移

Globus 提供了一种机制使得一个进程可以与用户代理或另一个进程进行安全鉴别，以确定创建进程的用户。但这并不意味着这进程之间存在信任委托。

为了实现进程间的相互鉴别，在 Globus 环境中创建的进程有一个临时的证书 *CP*。这个证书由创建进程的用户代理签署（而用户代理的证书由用户签署）。下面的算法说明了如何创建进程证书：

算法 5-3 进程的安全证书签署算法

1. 进程创建一个没有被签署的证书集合：UC_P。
2. 然后把证书集合传给创建进程的进程管理者。
3. 进程管理者发送一个 PROCESS-CREDENTIALS 请求和 UC_P给用户代理，PROCESS-CREDENTIALS 由进程管理者签署。
4. 用户代理检查 PROCESS-CREDENTIALS 请求，如果同意请求，这用户代理签署 UC_P产生 C_P（一个完整的安全证书）。
5. 用户代理通过进程管理者把签署的安全证书返回给进程。

需要注意的是，算法 5-3的前提是进程和进程管理者在同一个信任域中。算法 5-3允许一个用户代理请求分配资源，然后根据资源创建进程。在 Globus 的运行环境中，还存在一个进程需要请求创建另一个远程进程或请求远程资源的情况，而算法 5-3没有考虑这种情况。为此，Globus 通过对算法 5-3进行扩展来处理这种情况，即进程把远程请求发给创建进程的用户代理，由用户代理代替它完成与远端资源代理的相互认证和请求交互，并返回结果给进程。这种方法缺少扩展性，但比较简单。为了使进程可安全的把请求传给用户代理，Globus 采用了如下的算法：

算法 5-4 进程的远程请求处理算法

1. 进程和用户代理相互进行安全鉴别。
2. 进程向用户代理发送请求，请求的形式为：

   CP{operation, operation args}

   这里 operation 可以是 allocate 或 create。
3. 用户代理使用算法 5-3对请求进行处理。
4. 用户代理签署结果并把结果返回给请求进程。

5.5.6 网格信息服务的安全保证

Globus 的网格信息服务（即 MDS，具体细节可参考“元计算目录服务”一章）的安全性是通过 GSI 和其它安全技术来保证的，使得只有合法的用户才能访问信息，只有合法的资源才能提供信息。Globus 的 MDS 采用了基于 LDAP v3 协议的 OpenLDAP 软件作为其前端实现，而 OpenLDAP 的安全性是通过 SASL 协议来保证的。SASL 协议在实现上采用了 GSS-API（在下一节介绍），支持面向连接协议（如 TCP）的安全认证。在 Globus 中采用了 Cyrus SASL 软件作为 SASL 协议的具体实现。SASL 要求程序的运行库是运行

时动态加载的，SASL 本身不提供任何安全性，它主要依赖其下面的支持技术来提供实际的身份鉴别和消息保护服务。应用程序通过要求特定的安全服务或使用 SASL 实现提供的缺省安全服务来保证安全性，而且 Globus 的 MDS 也使用了基于 SSL 协议来提供安全的数据通信。OpenLDAP 通过 Cyrus SASL 软件支持的安全鉴别服务（Cyrus SASL 也支持 GSS-API），而且通过 OpenSSL 软件实现基于 SSL 协议的保密和完整性保护，同时 OpenLDAP 本身提供基于 LDAP 认证信息、IP 地址、域名和其它属性的数据访问控制。MDS 充分利用了 OpenLDAP 提供的这些安全措施来保证 MDS 自身的安全。

5.6 GSI 的安全策略实现

Globus GSI 安全策略的实现主要包含三部分的内容：通用安全服务编程接口－GSS-API （Generic Security Service API）、安全认证证书管理和用户代理的实现。为了支持用户认证，Globus 的安全认证证书管理实现的安全手段主要包括建立 Globus 安全认证中心、用 TIS MOSS 软件生成使用证书、绑定本地名字和证书上的名字、撤回证书、从本地节点消除用户等。有关用户代理的实现可参见 Globus 中的 GSI 相关源代码和文献[25]，这里就不作赘述了。下面主要介绍 GSS-API。

5.6.1 通用安全服务编程接口

Globus 的 GSI 安全策略没有解决可移植性的问题，当然这也不是它关注的问题。如果提供一组通用的安全编程接口来实现和完成对上述 Globus 安全组成部分的访问，则可以解决可移植性问题，为此 Globus 采用 GSS-API 作为其安全编程接口。GSS-API 定义提供了通用的安全服务，支持各种安全机制和技术，这样可支持应用程序在源码级的可移植性。GSS-API 主要面向主体之间的安全鉴别和安全通信操作，并主要提供的功能包括：获得证书、执行安全鉴别、签署消息和加密消息。

GSS-API 在安全传输和安全机制上是独立的，这主要体现在两个方面：（1）传输独立性： GSS 不依赖于特定的通信方法或通信库，而且某个 GSS 调用会产生一系列的标记并进行通信，目前可支持 TCP、UDP 和 Nexus 等通信协议；（2）机制独立性：指 GSS 不依赖于特定的安全算法，如 Kerberos、SESAME、DES、RSA 公钥密码等。GSS 是根据安全操作过程定义相应的函数。每个操作可通过不同的安全机制进行实现。

GSS-API 符合简单公钥体制 SPKM（Simple Public Key Mechanism），而 SPKM 的密钥管理与 X.509 和 PEM 兼容。GSS-API 支持在安全上下文建立过程中的安全授权数据通信，当然，GSS-API 也将支持其它的安全机制。而 GSS-API 中安全上下文的建立与用户本地的安全证书相关。用户也可以有多个安全上下文，但这需要相同或不同的安全证书集合。

为了支持点到点的安全上下文的建立，应用程序必须确定底层安全机制（如 SPKM、Kerberos、SESAME 等），并且交互双方都要支持某种机制，这样才能进行安全通信。为了保证对通信内容的保密性和高效性，GSS-API 允许用户选择是否对通信内容进行加密，并提供某种程度的服务质量 Quality of Service（QoS）。

直接使用 GSS-API 进行基于网格计算环境的安全编程比较复杂，而且 GSS-API 函数不支持 I/O 操作，主要由 gss_init_sec_context 和 gss_accept_sec_context 返回标记。为了进一步简化 Globus 中的安全编程，Globus 提出了用于 Globus 环境的 gss_assist 库，gss_assist 库封装了 GSS-API，提供了更简单的编程接口来方便应用程序的开发。

GSS-API 主要包含两大方面的控制：（1）安全证书（credential）的控制：每个进程一个安全证书，有本地用户和服务器的标识，通过 gss_cred_id_t 结构来表示，这可能是一个静态变量；（2）安全上下文（security context）的控制：每个连接一个安全上下文，在安全鉴别过程中建立，通过 gss_ctx_id_t 结构来表示。

通过使用 gss_assist 库可极大地方便基于网格计算环境的安全编程。一般的客户端与服务器端的交互流程如下：

1. 在客户端和服务器端获得安全证书
2. 在客户端和服务器端建立安全上下文
3. 双方进行安全鉴别
4. 通过安全鉴别后，双方可进行交互
5. 交互完后，双方清除安全上下文

如果读者想进一步了解有关 GSS-API 的细节，可参考 RFC 2744 文档。

5.7 小结

本章主要讲述了网格安全问题、GSI 的安全策略和实现。网格安全问题是网格计算中的一个核心问题,Globus项目的GSI是一个解决网格计算中安全问题的一个集成方案。其特点在于在保证网格计算安全性的同时，尽量方便用户和各种服务的交互（如提供单点登录等手段)。而且 GSI 充分利用现有的网络安全技术（如 X.509、Kerberos、SSL 等），并对某些部分进行扩充（如增加客户方与服务方的相互认证、支持对 SSH 和 FTP 的 GSI 扩充等），使得在网格计算环境下 GSI 具有一个一致安全性界面，极大地方便了网格应用的开发和使用。GSI 下一步的发展方向主要体现在提高 Web Service（如 SOAP）的安全性方面[2]。

思考题

1 网格计算环境中的安全问题与一般的网络安全问题有和异同？
2 GSI 的安全技术与现有安全技术的关系是什么？
3 GSI 的安全策略是什么？
4 GSI 的安全技术主要有哪些，并对其优缺点进行分析。
5 试在 Linux 系统上安全 GSI。
6 讨论在 Windows 操作系统上如何实现 GSI？

第6章 元计算目录服务

本章主要介绍Globus项目中的信息服务－－元计算目录服务MDS（Metacomputing Directory Service） [13][32]。元计算目录服务主要完成对网格计算环境中信息的发现、注册、查询、修改等工作，提供对网格计算环境的一个真实、实时的动态反映。其处理的信息主要是网格计算环境中的各种资源（包括数据资源、计算资源等）、服务和其它主体（entity）的描述。

本章首先对元计算目录服务进行了总述，然后介绍了元计算目录服务的特点和可以提供的信息类型，接下来对元计算目录服务的主要实现模块网格资源信息服务－GRIS（Grid Resource Information Service）和网格目录信息服务－GIIS（Grid Index Information Service）以及它们的底层基础－轻量目录管理协议 LDAP（Lightweight Directory Access Protocol）进行了简要的分析；最后介绍了元计算目录服务的简单使用。

6.1 元计算目录服务介绍

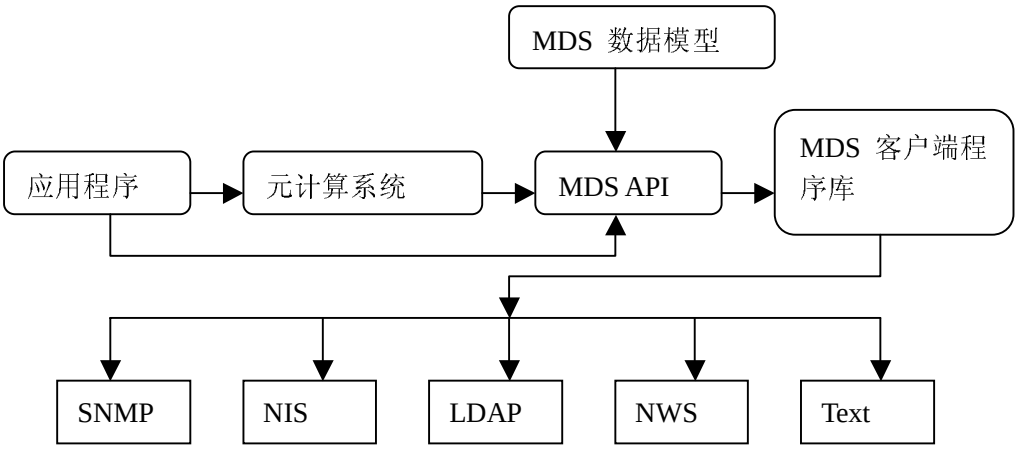
元计算目录服务是Globus项目中相当重要的一部分。在网格计算环境中存在各种动态资源，它们在地理上分散，又可以动态地加入或离开不同的虚拟组织。如何使网格应用程序方便地使用各种资源是Globus项目必须解决的问题。为此在Globus项目中提出了元计算目录服务－MDS，它主要是一种基于网格介绍环境的信息服务框架，面向网格计算环境中数目巨大、地理上分布、且具有动态性的各种资源和服务。MDS的内容主要包括资源（服务）发现、资源（服务）描述和资源（服务）监视与更新。这样网格应用程序可方便地利用Globus提供的MDS信息服务满足自己的各种需求。Globus的第三个版本将基于开发网格服务结构－OGSA 框架，用 XML 来描述各种信息，并且与 Web Service技术中的SOAP、WSDL和WS-Inspection紧密结合，提供更加方便和有效的元计算信息服务。

在网格计算环境中，信息服务（Information Services）应该具有的功能和特点包括：可访问服务系统组件的各种静态和动态信息、可针对异构和动态环境对信息服务进行配置和调整、具有统一和有效的存取信息的实现接口、对动态数据的访问具有可扩展性、可访问多个信息资源以及基于分布式的管理方式。

作为信息基础设施的一部分，MDS采用动态可扩展的框架来管理网格计算环境中各种资源（计算、网络、存储、仪器等）的静态和动态信息。目前MDS可提供如下服务信息：

- 网格环境中存在的资源
- 网格计算环境的状态信息
- 基于当前的网格计算环境的网格应用的优化信息

在实现上，MDS主要使用LDAP作为网格信息访问与存储的统一界面。MDS的基本特征包括数据生成、数据分布、数据存储、数据搜索、数据查询和数据显示等。MDS提供了一个可配置的信息提供者（information provider）组件，称为GRIS（Grid Resource Information Service）和一个可配置的集合目录组件，称为GIIS（Grid Index Information Service）。当然完整的MDS也可以搜集和发布基于其它协议的信息，如SNMP、NIS、NWS等，其简要逻辑结构如图 6-1所示。



不同的信息服务协议

图 6-1 MDS 简要逻辑结构图

6.2 元计算目录服务的特点

网格环境相关信息是网格计算环境中的一个关键资源。MDS 支持对虚拟组织的创建，使得虚拟组织中的用户可以相互合作并共享资源。一个虚拟组织可以收集并从一个统一的视角来表示各种网格资源。提供一个信息服务的基础设施来谐调跨虚拟组织的系统信息是很有必要的，因为这可以帮助网格应用程序根据这些系统信息进行自我调整和配置。MDS 正是这样的一个信息服务基础设施。通过查询 MDS 可以发现要使用的机器、计算机和网络等的当前情况和特性。通过使用 LDAP 服务器，MDS 提供了一个统一界面来表述分散的资源。

MDS 提供了一些必要的工具以构建基于 LDAP 的网格信息基础设施。MDS 使用 LDAP 协议作为查询各种系统组件信息的统一手段，并可以创建一个统一的、跨虚拟组织的资源信息名字空间（namespace）。MDS 因此定义了一个方法（基于 LDAP 和特定的模式（schema））来表示数据。在 Globus 工具包中与 MDS 相关的主要有 GRIS 和 GIIS 服务。

GRIS 服务提供了一个统一的手段来查询网格中资源的配置、能力和状态。GRIS 是一个分布的信息服务，通过在网格计算环境中对 GRIS 进行部署（deploy），可满足相关的查询请求，如主机名称、节点操作系统版本号等静态信息和可用 CPU 数和内存大小等动态信息。

GIIS 提供了一种把各种 GRIS 服务结合起来的手段，并提供一个连贯的系统映像以方便网格应用程序进行搜索和查询。GIIS 可鉴别特定类型的资源，如 GIIS 可列出属于某个虚拟组织的所有实验室中的计算资源，或者某个结构的所有分布的存储系统等。GIIS 可把属于某个虚拟组织的所有网格资源进行汇总，并提供一个连贯的网格资源系统映像。

6.3 信息提供者和提供的信息类型

MDS 的信息主要由信息提供者提供，MDS 中的信息提供者包括核心信息提供者、通用信息提供者和自定义信息提供者。其中核心信息提供者是必须的，它提供各种关键信息；通用信息提供者提供一些不是必须的信息；自定义信息提供者主要提供某些特定应用的特殊信息。

MDS 中的核心信息提供者提供有关网格资源中关键的信息和操作状态。这些信息提供者把相关数据提交给 LDAP 服务器，并由 LDAP 服务器提供可供查询的数据包括操作平台的类型和系统机构指令集、操作系统（主机上）名字和版本号、CPU 信息－CPU 类型、CPU 数目、CPU 版本、CPU 速度、CPU 的 Cache、内存－物理内存和虚拟内存的实际空间和可用空间、网络界面信息－机器名和网络地址以及文件系统信息－文件系统大小和可用空间等。

GRIS 基于信息类型和它的缓存情况对信息提供者进行调用，并对信息提供者的信息进行过滤，再把结果返回给信息查询者。目前核心信息提供者可在 Linux、Solaris、Irix、AIX 和 Tru64 等平台上运行。当然通过 Globus MDS 也可以创建自定义的信息提供者（Custom Information Providers），具体细节可参见文献[57]。

在 MDS 中，高层服务（或用户）与信息提供者之间通过两个基本协议进行交互，一个是软状态注册协议（soft-state registration protocol）：信息提供者通过软状态注册协议对 MDS 进行进行信息注册；另一个是查询协议（enquiry protocol）：高层服务或用户通过查询协议从 MDS 中查询或预订自己感兴趣的主体信息。信息提供者采用注册协议通知高层服务方它的存在性，而高层服务方通过查询协议可得到相关信息，并可对信息进行归纳合并。

MDS 不让高层服务方与信息提供者直接进行交互，而是通过上述两种协议进行间接交互。这主要基于如下考虑：把信息查询与信息提供分开处理可以不用修改组成网格计算环境的各种资源和服务，在实现上改动很小，并层次分明，开销不大，有利于将来不同的高层服务和信息提供者的实现。

MDS 实现的信息源包括静态主机信息（操作系统及其版本号、CPU 类型、CPU 数目、内存大小等）、动态直接信息（平均负载、运行的进程数等）、存储系统信息（可用磁盘空间、总磁盘空间等）和基于 Network Weather Service 得到网络信息（当前测量的和预测的网络带宽、当前测量的和预测的网络延迟等）。

在 MDS 中有一系列的信息提供者程序，这些程序可用来向 MDS 发布各种类型的网格信息。用户也可创建自己特定的信息提供者，并分布特定的网格信息。

6.4 元计算目录服务的信息模型和目录信息树

6.4.1 LDAP 介绍

MDS 目录结构遵从 LDAP 模型，主要由目录信息树－DIT（Directory Information Tree）层次和对象类定义组成。MDS 中的 GRIS 和 GIIS 也是基于 OpenLDAP 软件（一个符合 LDAP 协议的自由软件）实现的。所以有必要对 LDAP 进行简要介绍。有关 LDAP 的

详细信息可参考 RFC 2251、RFC 2252、RFC 2254 和 RFC 2256 文档。

轻量目录访问协议－LDAP（Lightweight Directory Access Protocol）是一个独立于厂家和平台的开放网络协议标准。**LDAP 是用来访问存储在信息目录（也就是 LDAP 目录）中的信息的协议。**更为确切和正式的说法应该是：“通过使用 LDAP，可以在信息目录的正确位置读取(或存储)数据”。它在对 X.500 标准进行简化的基础上,基于 TCP/IP 定义了一个目录服务标准，主要包括以下几个部分：

- LDAP 信息模型：定义了目录中数据的类型。
- LDAP 命名模型：定义了目录的组织方式。
- LDAP 函数模型：定义了如何访问和更新目录。
- LDAP 安全模型：定义了如何防止未授权用户对目录信息的访问和修改。

在 LDAP 目录中信息是存储在一个树形结构中的，一般称为 DIT(Directory Information Tree)，DIT 由很多主体(entry)组成（**这里所指的主体与网格计算中定义的主体是两个概念！**)。主体（entry）表示 LDAP 中的资源信息，每个主体具有唯一的标识（distinguish name)。每个主体包含零个或多个“属性—值（Attribute-Value)”对，表示资源的属性。主体的类型，称为对象类（ObjectClass)，确定了主体（entry）和属性（Attribute）的必选项和可选项。

就象 Sybase、Oracle、Informix 或 Microsoft 的数据库管理系统（DBMS）是用于处理查询和更新关系型数据库那样,LDAP 服务器也是用来处理查询和更新 LDAP 目录的。换句话来说 LDAP 目录也是一种类型的数据库，但不是关系型数据库。LDAP 主要优化了数据读取的性能，适合于更新频率远小于读取频率的情况。由于 LDAP 和普通数据库有差别，在性能上也有不同，它主要的优点包括：(1) LDAP 是跨平台的协议，可以在任何平台的计算机上，用 LDAP 客户端软件去访问 LDAP 服务器；(2) 对 LDAP 的读操作的完成速度比普通的数据库要快很多，LDAP 专门为读操作进行了优化，适合用于那种需要频繁读取场合；(3)LDAP 服务器可以是分布的,用户访问到的信息可以是本地的 LDAP 服务器，也可以是全局的。各地服务器之间可以通过 LDAP 内部的机制很容易的实现内容的同步(不过同步更新的速度不一定很快)；(4)LDAP 的存储是以一条条纪录(entry)存储的，各条 entry 可以存储的属性是可变的。

如何完成资源的 LDAP 定义和表示后，就需要根据资源的表示方法对资源信息进行有效的组织。根据资源分类的特性，使用层次化的树形结构对资源信息进行组织是比较合理的，便于系统对资源信息的查找、添加等操作。资源信息的组织需要遵循：(1) 资源的标识应保证这个资源在系统中的唯一性,不能出现多个资源标识指向同一个资源或一个资源标识指向多个资源的情况。(2) 资源分类应具有多个层次，同时树形组织结构应清楚的体现不同类型资源的差异。(3) 资源的属性可以冗余，保证资源的完整表示。

资源目录管理系统对资源信息采用了多层次的树形组织方式。在树形结构中，每个结点代表一个对象类，每个对象类中都定义了其父结点和子结点。每个对象类都可以对应多个主体。上述的结构体现了面向对象的思想，每个主体都可能存在对某个对象类的继承关系；同时，这种树形的结构便于资源信息的查找，可提供高效的资源定位方式。

在 LDAP 协议中存在两种通信模式:客户－服务器通信和服务器－服务器通信。基本的客户－服务器通信允许用户程序连接 LDAP 服务器进行创建、检索、修改、删除数据等操作。服务器－服务器通信定义理多个服务器如何共享一个 LDAP 目录信息树 (DIT)，以及如何更新和复制服务器之间的信息。

6.4.2 LDAP 目录信息树

LDAP 的简单结构是一个目录信息树－DIT。从根节点开始，它包含一个对其所有数据的层次视图，而且提供一个基于树形的搜索系统。这个树形结构本身称为 DIT。DIT 的子树可在一个 LDAP 服务器中，也可分布在多个 LDAP 服务器中。目录的内容称为对象类（object class）和项(entry)。对象类描述了什么信息可存储在目录中。而项把相关信息组合在一起。对象根据其在中的位置来命名。一个具体的例子如图 6-2所示。

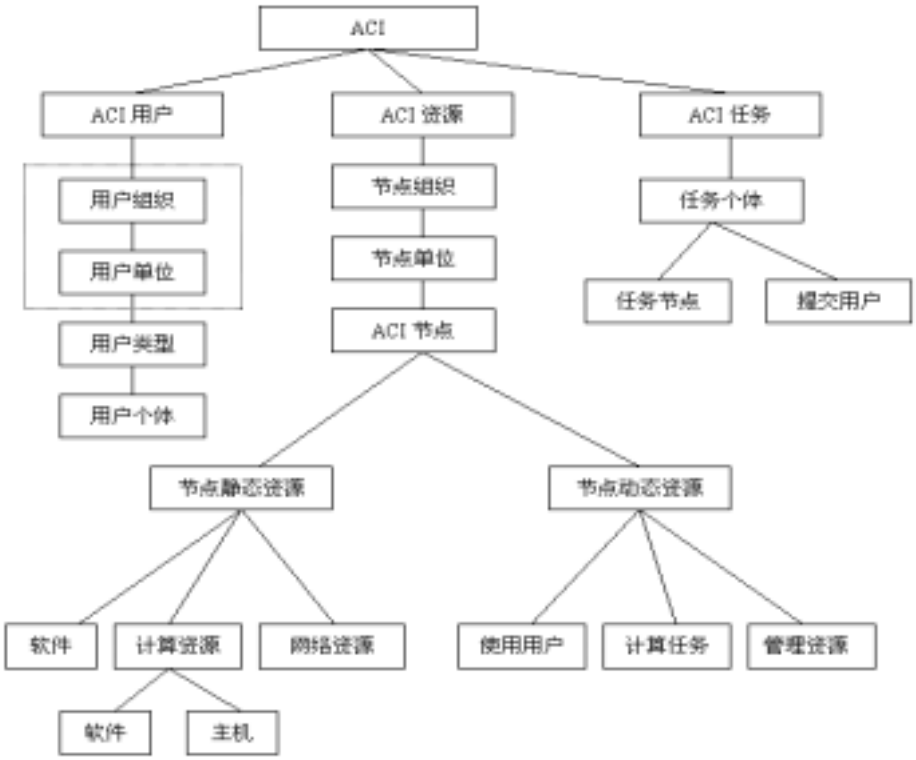


图 6-2 LDAP 目录信息树(DIT)的一个具体表示

信息树中的每一个节点是一数据项，或是一个目录服务项－DSE（ Directory Service Entry ）。这些项包含描述计算环境中真实或抽象对象的实际记录，如用户、计算机、网络、参数等。记录的内容作为一个属性/值对－attribute/value pairs（用<name, value>表示）来存储。

信息树中的根节点称为 root DSE（Root Directory Specific Entry），root DSE 包含对这个信息树的布局和内容的描述。而某个数据项包含一系列的属性/值对。可以简单地把每一项看成是变量的数据结构。

DIT 的某个节点都有一个唯一的路径连接到根节点。这个路径可作为与此节点相关联的项的名称。在 LDAP 的概念中这个名称就是节点的 DN（Distinguished Name）。

DIT 的容器（container）是在 DIT 中的一个分支节点关联的数据项。一个容器可存储其它项或其它容器。在 Globus 环境中，ou（organization unit） 对象通常就是一个 DIT 中的容器。LDAP 中的 LDIF（Data Interchange Format ）格式定义了 DIT 中记录的格式和语法。

6.4.3 MDS 的信息模型

目前 MDS 的信息模型中有三种类型的信息：（1）结构信息（Structural information）：结构信息通过映射到对象的层次结构来表示，而对象是通过 DIT 中的有名位置来表示；（2）合并信息（Merged information）：合并信息通过把携带子节点数据的父节点联合（join）起来表示，通常用于简化查询模式；（3）辅助信息（Auxiliary information）：辅助信息采用 LDAP 辅助对象类来统一表示 leaf/parent 数据，可用于表示相关属性信息的集合。一个对象必须有一个结构类型，但可有零个或多个辅助类型。

在 MDS 中，在 GRIS 主机对象层次中使用的信息模型如下：

```
Mds-Host-name=hostname
    Mds-Software-Deployment=operating system
    Mds-Device-Group-name=processors
        Mds-Device-name=cpu 0
    Mds-Device-Group-name=memory
        Mds-Device-name=physical memory
        Mds-Device-name=virtual memory
    Mds-Device-Group-name=filesystems
        Mds-Device-name=/scratch1
        Mds-Device-name=/scratch2
    Mds-Device-Group-name=networks
        Mds-Device-name=eth0
```

在上述层次中，主机包含一系列的设备，注意 Mds-Device-Group-name、eMds-Host-name 和 MDS-Software-Deployment 都是结构类型，Mds-Device-name objects 是辅助类型。

GRIS 结构类层次包含如下结构类型：

```
Mds
    Attr: Mds-validfrom (like createtime)
    Attr: Mds-validto (accuracy metadata)
    Attr: Mds-keepto (discard metadata)
MdsHost
MdsDevice
MdsDeviceGroup
MdsSoftwareDeployment
Every MDS object: name, time metadata
```

每一个结构类型游有一个 name 属性，用来指定一系列的属性。下面是 GRIS 辅助类中一个类型的例子：

```
MdsCpu
    Attr: Mds-Cpu-vendor - Once per CPU
    Attr: Mds-Cpu-model
```

```
    Attr: Mds-Cpu-speedMHz
MdsCpuCache
    Attr: Mds-Cpu-Cache-L1kB - Once per CPU
MdsCpuSmp
    Attr: Mds-Cpu-Smp-size - Once per SMP
MdsCpuTotal
    Attr: Mds-Cpu-Total-count - Once per MPP
MdsCpuFree - Once per SMP
    Attr: Mds-Cpu-Free-1minX100
    ……
MdsCpuTotalFree - Once per MPP
    Attr: Mds-Cpu-Total-Free-1minX100
    ……
```

注意在上述例子中的 MdsCpu、MdsCpuCache 等是类名，而 Mds-Cpu-vendor、Mds-Cpu-model 等是属性名。

6.4.4 MDS 信息层次

MDS 的信息模型把计算资源的物理和逻辑组件用一种层次元素来表示。其中只有一小部分元素类型对应 LDAP 结构对象类：

```
class MdsVo
    contains attr Mds-Vo-name
class MdsHost
    contains attr Mds-Host-hn
class MdsDevice
    contains attr Mds-Device-name
class MdsDeviceGroup
    contains attr Mds-Device-Group-name
...
```

一个辅助类型的补充部分增加了特殊元素的信息。LDAP 辅助类型客用于对结构类型对象的类型进行扩展。在 MDS 的信息模型中使用这个特性把信息”向上”融合，使得在页节点可包含单个资源实例的信息时，父节点可包含多个资源实例的合成信息，例如：

```
dn: Mds-Device-Group-name=memory, ...
    objectclass: MdsMemoryRamTotal
    objectclass: MdsMemoryVmTotal
    objectclass: MdsDeviceGroup
    Mds-Device-Group-name: memory
    Mds-validfrom: 200210030128.12Z
    Mds-validto: 200210030128.12Z
    Mds-keepto: 200210030128.12Z
    Mds-Memory-Ram-Total-sizeMB: 1024
    ……
```

```
        Mds-Memory-Vm-freeMB: 1592
……
 dn: Mds-Device-name=virtual memory,
 Mds-Device-Group-name=memory, ...
        objectclass: Mds
        objectclass: MdsDevice
        objectclass: MdsMemoryVm
        Mds-Device-name: virtual memory
        ……
        Mds-keepto: 200210030128.12Z
```

这样通过辅助类型的帮助，父节点对象可反映不同子节点的信息。设一个主机系统是一个 SMP 计算机资源，则通过辅助类型的描述，可得到如图 6-3 所示的反映主机特性的的 DIT 视图：

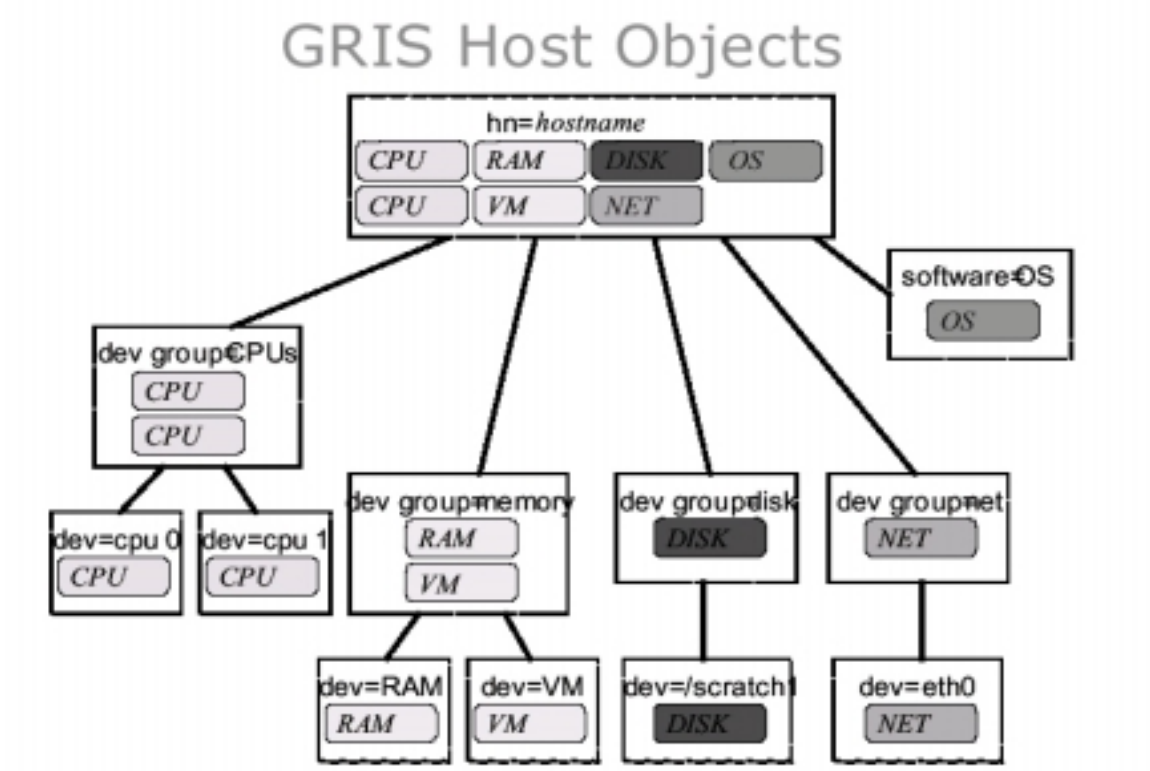


图 6-3 反映主机特性的的 DIT 视图

6.5 GRIS 和 GIIS 介绍

为了实现数据生成、数据分布、数据存储、数据搜索、数据查询和数据显示等功能。MDS 提供了一个可配置的信息提供者（information provider）组件，称为网格资源信息服务 Grid Resource Information Service (GRIS)和一个可配置的集合目录组件，称为网格目录信息服务 Grid Index Information Service (GIIS)。用户访问 GRIS 和 GIIS 的简单情况如图 6-4所示。GRIS 基于 LDAP 协议，并提供资源的相关信息。GIIS 提供一个服务缓存，就象一个 WEB 搜索引擎。资源可通过 GRIS 或直接把其信息注册到

GIIS 中，或者 GIIS 得到一个用户请求，且自身的缓存信息已经过期，就通过 GRIS 获得相关更新信息。

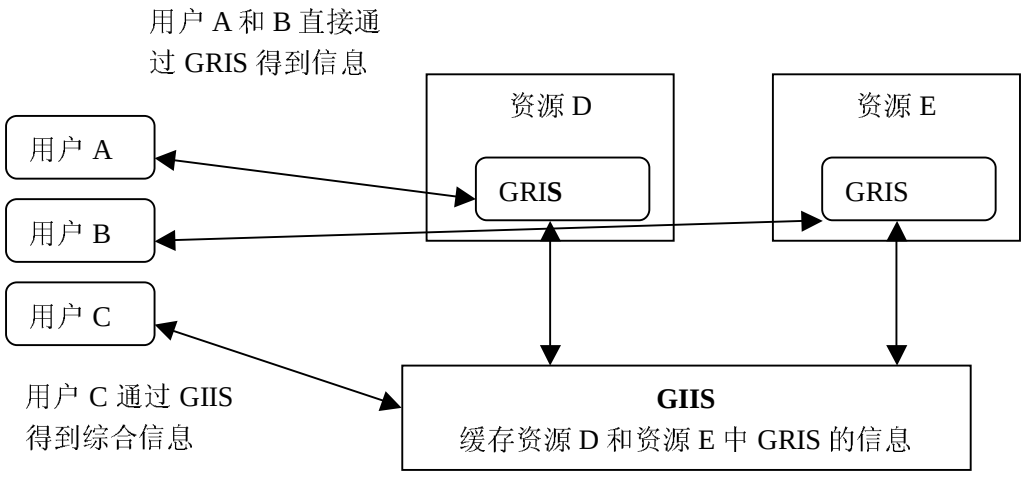


图 6-4 用户访问 GRIS 和 GIIS 的情况

6.5.1 网格资源信息服务－GRIS

MDS 包括一个标准的、可配置的信息提供者框架，称之为网格资源信息服务－GRIS。GRIS 是通过 OpenLDAP 实现的，并且加入了特定的信息源和模式。某个 MDS 资源可运行一个本地的 GRIS。在缺省情况下，一个 GRIS 服务可自动配置，且在网络端口号 2135 上进行监听。

GRIS 可响应网格计算环境中其它系统的信息查询请求，且可配置成自身为聚集目录服务（如 GIIS），这样可接收其它 GRIS 上的资源信息。一个 GRIS 对信息查询请求进行安全鉴别和解析后，根据请求信息的类型把查询请求分发到一个或多个本地信息提供者。然后 GRIS 把信息提供者返回的信息汇总，并返回给信息查询请求者。网络有效执行查询处理，可根据查询的范围对查询过程进行裁减。

GRIS 与信息提供者之间通过一个定义好的 API 进行通信。GRIS 通过指定信息提供者提供的信息类型和信息提供者定义的程序来进行配置。为了控制 GRIS 操作，提高响应时间和部署的灵活性，某个信息提供者的结构可能会在一定时间间隔进行缓存，以提高系统的效率。由信息提供者返回的结果要通过 GRIS 进行过滤，以消除与客户不需要的信息。这里需要注意的是过滤操作由 GRIS 完成，而不是信息提供者，因此（1）信息提供者的实现会比较简单；（2）对信息提供者的信息缓存可更好地提高查询性能。

6.5.2 网格目录信息服务－GIIS

MDS 提供了一个聚集目录的框架，称之为网格目录信息服务－GIIS，通过 GIIS 可构成一个层次结构的简单聚集目录框架，形成分布式的信息服务。这样 GIIS 可通过一个简单的命令从多个 GRIS 中得到信息。一个 GIIS 可设置为单节点范围或多节点合作范围的信息服务器。

GIIS 可提供信息集合查询并支持对多个 GRIS 的有效查询。GIIS 框架有三个主要部

分组成：通用注册处理、可插入的目录构造和可插入的搜索处理。这些都是基于OpenLDAP服务器的特定后端服务来实现的。

OpenLDAP前端对注册信息进行解码并发送到后端交与GIIS进行处理。GIIS和GRIS在实现上有许多共同之处，都依赖LDAP前端进行协议处理、认证和结果过滤，所以实际上二者可在一个服务器上实现。使用LDAP通用协议可提高系统的互操作性，同时简化了实现。

6.6 使用 MDS

通过MDS可定位和查询资源的各种特性。例如，可向MDS发出“哪些资源具有特定的体系结构、软件或网络带宽？”来定位资源；也可向MDS发出“这个资源的物理特征和网络情况是什么？”来查询资源。更具体地讲，通过MDS可查询信息包括：（1）计算资源的信息：IP 地址、可使用的软件、系统管理者、连接的网络、操作系统名称和版本号、存储系统信息、系统负载、进程信息、内存信息、任务队列等；（2）网络资源信息：网络带宽、网络协议、网络延迟、网络的逻辑拓扑结构等；（3）Globus基础设施信息：主机信息、资源管理者等。Globus用户和高层服务可通过MDS-API或通过命令行对MDS进行访问。为了对MDS进行交互，用户首先需要合法的安全证书，并且创建用户代理进行授权访问，同时在服务方启动MDS。

在Globus环境中，可通过grid-info-search命令对信息进行查询，命令的接收方由节点名、信息服务方和端口号确定。下面简单介绍grid-info-search命令的各个参数：

-config file

确定不同的配置文件，而不使用缺省的配置文件，注意此文件中的定义会覆盖用户环境和前面命令行选项中的定义。

-mdshost host (-h)

确定MDS服务方运行的主机名称。缺省名称是$GRID_INFO_HOST

-mdsport port (-p)

确定MDS服务方的端口号。缺省端口号是$GRID_INFO_PORT

-anonymous (-x)

采用匿名绑定，而不使用GSS-API

-mdsbasedn branch-point (-b)

指定进行搜索的DIT的基址。缺省值为"${GRID_INFO_BASEDN}"

-s scope

确定搜索的范围。Scope可以是base（基于dn层）、one（基于dn层和其下一层）或sub（基于dn层和所有下面各层）。

“attribute”

指定搜索返回的单一属性值，如MdsCpu等。

“filter”

设置搜索的属性关系

“(attribute=value)”

“(attribute>=value)”

“(attribute<=value)”

“(attribute~=value)”

“(attribute=*)” Searches for the presence of an item.搜索任意值

“(attribute=*a)” Searches for a substring item.搜索以 a 结尾的的值

-mdstimeout seconds (-T)

执行 MDS 请求的限定时间。缺省值是$GRID_INFO_TIMEOUT。

下面的例子描述了如何显示本地一个节点上的信息查询过程，命令如下：

```
grid-info-search -h giis-demo.globus.org -p 8463 -b 'Mds-vo-name=local, o=Grid'
```

这次搜索指出了 MDS 服务方的主机和端口号。Mds-Vo-name=local 表示在 GRIS 上搜索。搜索的结果如下：

```
SASL/GSI-GSSAPI authentication started
SASL SSF: 56
SASL installing layers
version: 2
# filter: (objectclass=*)
# requesting: ALL
# dc-user2.isi.edu, local, grid
dn: Mds-Host-hn=dc-user2.isi.edu,Mds-Vo-name=local,o=Grid
objectClass: MdsComputer
objectClass: MdsComputerTotal
objectClass: MdsCpu
……
Mds-Computer-isa: IA32
Mds-Computer-platform: i686
Mds-Computer-Total-nodeCount: 1
……
```

6.7 小结

本章对 Globus 的 MDS 进行了介绍，主要包括 MDS 的信息模型、信息层次和其中的主要实现部分 GRIS 和 GIIS。目前的 MDS 主要是基于 LDAP 协议的 OpenLDAP 软件实现的，完成对网格计算环境中信息的发现、注册、查询、修改等工作，可对各种临时性主体和永久性主体的各种信息进行描述和存储，符合网格计算环境的动态性特点。Globus 信息服务下一步的发展是对网格环境中的各种服务进行描述，主要采用 LDAP 和 WSDL 结合的方法来更好地实现面向 Web Service 的信息服务。

思考题

1 LDAP 协议的特点是什么，与 MDS 有何关系？
2 请分析 MDS 的信息模型和层次。
3 GRIS 和 GIIS 的特点是什么？
4 试在 Linux 系统上建立一个 MDS 的例子。
5 在 Windows 操作系统上如何实现 MDS？
6 试分析 Web Service 与 MDS 的关系。

第7章 资源分配管理

Globus 中的资源分配管理者－GRAM（Resource Allocation Manager）位于 Globus 资源管理体系结构的底层[17]，主要处理资源请求、执行远程应用执行、分配资源和管理活动等任务，并根据计算资源的情况把资源更新信息发送给 MDS。GRAM 提供 API 来方便上层服务或用户提交任务请求、删除任务请求和检查提交任务情况。用户可根据资源描述语言 RSL（Resource Specification Language）来发出任务请求并交与 GRAM 处理[29]。本章主要对 Globus 中的资源分配管理者－GRAM、动态协同分配器－DUROC（Dynamically-Updated Request Online Coallocator）和资源描述语言－RSL 进行逐一介绍。

7.1 资源分配管理者－GRAM

7.1.1 GRAM 的组成和执行流程

GRAM 的组成部分主要包括 gatekeeper、任务管理者、资源管理者等。Gatekeeper 位于资源管理者或任务管理者所在节点，用于与请求资源的用户或用户代理进行安全鉴别。资源描述语言－RSL 是一种结构化语言，用来描述资源请求和相关参数。任务请求的简单执行流程如图 7-1所示。简单地说，当用户要提交一个任务时，则用户发一个执行任务的请求（用 RSL 进行描述）给远程计算机的 gatekeeper；gatekeeper 处理这个请求，并针对这个任务创建一个任务管理者 job manager。任务管理者对任务请求中的 RSL 描述进行解析，然后启动并监视任务的执行，而且还把任务的状态信息发送用户。当远程任务正常结束或失败后，任务管理者也结束运行。

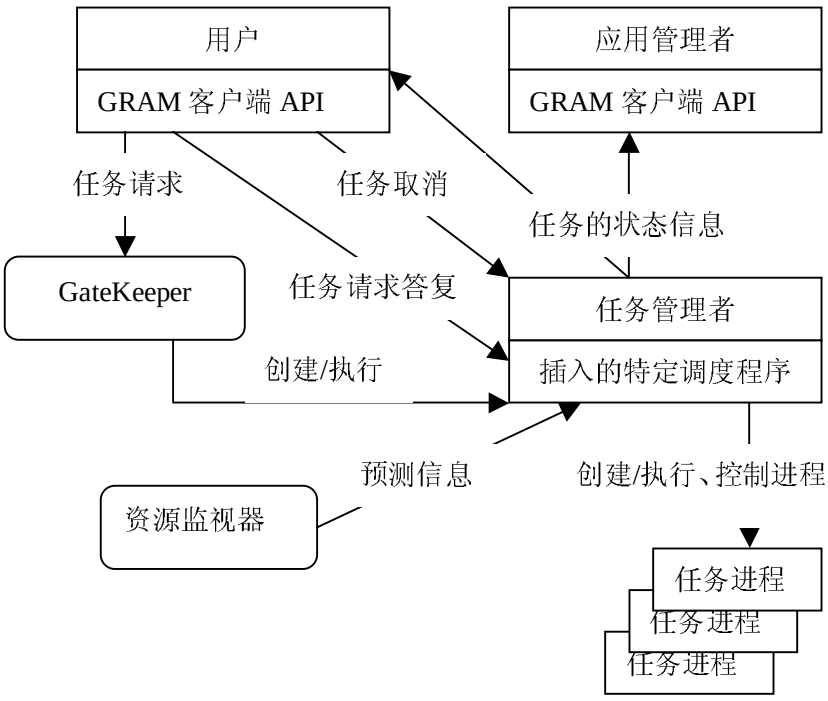


图 7-1 任务请求的简单执行流程

GRAM 主要负责的事务包括：（1）解析和处理任务请求中的 RSL 描述，在任务请求中选择的资源、任务处理和任务控制等 RSL 描述，针对可用资源等情况对请求做出拒绝处理或执行等操作；（2）管理远程监控启动的任务；（3）根据所管理的资源可用情况更新 MDS。GRAM 主要涉及的对象和组成部分描述如下：

1 Resource －这里仅仅是指可代表用户运行一个或多个进程的主体（entity）
2 Client － 这里仅指调用客户端资源分配API的进程。
3 Job －根据一个任务请求产生的一个进程或一个进程组。任务是可以使分组的，这样当一个任务产生错误后，会使得整个任务组结束运行。如果一个任务被用户删除，这任务中的所有进程都会结束运行。
4 Job Request －发给gatekeeper的请求，用于创建一个或多个任务进程，请求用RSL语言描述，主要给出了资源的选择（何时何地创建任务进程）、任务进程创建（创建什么任务进程）和任务控制（进程如何执行）的描述。
5 Gatekeeper － 一个拥有root权限的进程，主要负责处理任务分配请求。当gatekeeper接收到一个client发出的任务分配请求后，它负责的操作包括：与client进行相互安全鉴别、把client映射为本地的一个用户、在本地启动一个拥有一个本地用户权限的任务管理者、把根据RSL描述解析出的任务分配参数传递给新创建的任务管理者。
6 Job Manager －一个任务管理者由gatekeeper根据任务请求进行创建。它主要负责在本地系统中启动任务，并处理其它与用户的通信。任务管理者主要由两部分组成：（1）通用组件Common Component－把从gatekeeper接收的信息转变成一个由机器相关组件Machine-Specific Component执行的内部API，通用组件也负责转换由机器相关组件发出的callback请求为发给应用管理者（application manager）的消息；（2）机器相关组件Machine-Specific Component－主要在本地环境实现一些内部API，主要包括对本地系统的调用、向resource monitor发送信息，查询MDS。
7 Resource Monitor －负责监视本地调度系统和本地资源，向任务管理者提供预估的延迟时间。
8 Application Manager －通过多个资源管理者管理一个在不同资源上运行的应用。作为一个集中的管理者，它从任务管理者接收callback请求，并指示任务的状态改变。

GRAM 支持如下的调度模型。用户和资源掮客（resource broker）提交一个任务请求，并可指定初始任务状态为某种任务状态（如挂起 pending）。则 GRAM 可根据任务请求来设置任务的状态。四种任务状态包括：（1）Pending－表示此任务的资源还没有分配；（2）Active－表示任务得到了资源，且应用程序正在运行；（3）Failed－表示由于错误和用户取消执行使得任务非正常结束；（4）Done－表示任务成功结束。

7.1.2 GRAM-API

GRAM 主要提供了三类 API，分别面向客户端、任务和任务管理者。客户端 API 主要提供处理向服务端发请求的功能；与任务管理器相关的 API 主要提供了一个统一界面与底层的本地调度器进行交互；与 myjob 相关的 API 主要提供了查询任务中进程信息的功能。现分别介绍如下：

globus_gram_client 系列函数：

globus_gram_client_job_request：提交任务给远程资源

globus_gram_client_job_status：查询请求任务的状态

globus_gram_client_job_cancel：取消一个挂起或运行的任务

globus_gram_client_callback_allow

globus_gram_client_callback_disallow

globus_gram_client_callback_check：这三个函数控制用于异步状态改变的回调函数

globus_gram_client_job_callback_register

globus_gram_client_job_callback_unregister：这两个函数向任务管理器注册/撤销回调函数

globus_gram_myjob 系列函数

globus_gram_myjob_size：得到当前任务的进程数

globus_gram_myjob_rank：得到当前进程在任务中的位置号

globus_gram_myjob_send：一个进程向属于同一任务的另一个进程发信息

globus_gram_myjob_receive：一个进程向属于同一任务的另一个进程收信息

globus_gram_myjob_kill：如果产生错误，则返回一个错误信息

globus_gram_ jobmanager 系列函数

globus_gram_jobmanager_request_init：初始化任务管理者的请求信息

globus_gram_jobmanager_request_destroy：清除任务管理者的请求信息

globus_gram_jobmanager_request：向本地调度器发请求

globus_gram_jobmanager_request_cancel：取消任务请求

globus_gram_jobmanager_request_check：检查任务状态，并更新请求结构

7.2 动态协同分配代理－DUROC

Globus 环境中包含资源管理者组件，用来提供与系统相关的调度器，一个资源管理者提供一个访问界面来把任务提交到特定的物理资源上。如果要执行需要分布式资源的任务，则需要一个协同分配代理（co-allocator）负责各个资源管理者之间的协同交互。而协同调度器必须提供一个方便的界面来获得资源并在多个管理者之间执行任务[22]。

7.2.1 DUROC 的管理结构

一个智能的协同分配代理的任务主要有两个部分：一是处理资源描述，并确定一个任务如何分布到各个资源上，具体而言，就是把抽象的资源描述转换成具体的资源管理者中的资源描述；另一个是处理底层具体的资源描述，并进行资源分配，即把任务请求进行分解，并发送到各个相关的资源管理者上进行任务调度。

对抽象的资源描述进行具体化实际上是对资源描述的进一步细化。可允许用户进行干预，对细化的资源进行调整，提供引入用户的启发式信息，可使得基于最终资源选择的任务执行更加有效。如下面的例子描述了把一个抽象的资源描述进行具体化，转换成在两个具体的资源请求：

lower ((count=5)) -->

```
(+ (& (count=3) (resourceManagerContact=RM1 ))
   (& (count=2) (resourceManagerContact=RM2 )))
```

DUROC 实现可跨单个资源分配器的分配操作，但把选择资源的决定留给高层工具实现。一旦一个资源描述被细化，则协同分配代理必须指定资源描述进行资源分配。在通常情况下，资源由它们本地的资源管理者进行管理，而协同分配代理对任务的调度过程是一个原子操作。由于 GRAM 不直接提供对多个资源管理者的原子操作的支持，使得用户必须完成任务启动的原子操作，使得所有的资源都得到后，任务才算启动成功。下面是一个例子片段：

```
main :
    资源请求操作…
    job_start_barrier ( )
    …
    user_operations ( )
```

当所有的资源都得到后，job_start_barrier 函数才能通过。用户在实现任务启动的原子操作时要注意三点：（1）由于任务创建需要保证原子性，用户不能执行不可恢复的操作；（2）job_start_barrier 函数可用来在任务创建失败（无法得到某些资源）后，释放已得到的资源；（3）job_start_barrier 函数可初始化任务集合通信函数来使用协同分配的资源。

DUROC 使用 RSL 作为它的协同分配描述语言，DUROC 可对底层（具体的）资源进行描述，即根据抽象的资源描述产生一个具体的资源描述，这需要 DUROC 可对资源进行之间访问，并确定与哪个资源管理者进行交互。下面的例子把一个任务中的五个进程分配到两个计算节点上执行：

```
( &(count=5) (executable=myprog))   -->
    +(&(resourceManagerContact=RM1)(count=3)(executable=myprog.sparc))
    (&(resourceManagerContact=RM2)(count=2)(executable=myprog.rs6000))
```

7.2.2 DUROC-API

DUROC 主要以 API 的形式存在，当任务请求传递给 DUROC API 处理后，任务请求被分解成多个 GRAM 请求，并且把每个请求提交给 GRAM API 处理。所有请求都成功后，就达到一个任务启动同步点，一旦达到任务请求同步点后，任务就可正常执行了。但如果没有任务启动同步点，则没有办法确保所有的资源都已获得。DRUOC 还提供了简单的通信库以方便子任务之间进行通信，资源协同分配应用编程接口－DUROC-API 提供的功能包括：把任务请求发给资源掮客（broker）、对提交的请求进行编辑、取消资源或任务请求、得到任务状态信息等。

DUROC API 与 Resouce Management API 很类似，，但增加了 subjob-add、subjob-delete 和 barrier-release 操作（用于任务开始前的同步）。而 job-structure 和 inter-subjob 等通信操作给任务自组织提供了手段。DUROC 的 API 主要分三类：Duroc Control-library API、DUROC runtime-library API 和 DUROC bootstrap-library，现简要介绍如下：

Duroc Control-library API：

globus_module_activate (LOBUS_DUROC_CONTROL_MODULE)：激活

DUROC control-library API 实现

globus_module_deactivate (GLOBUS_DUROC_CONTROL_MODULE)：失效 DUROC control-library API 实现

globus_duroc_control_init：初始化 globus_duroc_control_t 对象

globus_duroc_control_job_request：请求协作分配资源

globus_duroc_control_subjob_add：增加一个子任务

globus_duroc_control_subjob_delete：减少一个子任务

globus_duroc_control_barrier_release：在一个同步点消除一个任务请求中一个挂起的任务请求或杀死一个进程，释放相关的资源

globus_duroc_control_job_cancel：消除一个任务请求中一个挂起的任务请求或杀死一个进程，释放相关的资源

globus_duroc_control_subjob_states：得到一个任务中每个子任务的状态

DUROC runtime-library API：

globus_module_activate (GLOBUS_DUROC_RUNTIME_MODULE)：激活 DUROC runtime-library API 实现

globus_module_deactivate (GLOBUS_DUROC_RUNTIME_MODULE)：失效 DUROC runtime-library API 实现

globus_duroc_runtime_barrier：实现任务启动的原子性并谐调分布式进程

globus_duroc_runtime_inter_subjob_structure：得到 DRUOC 任务的布局

globus_duroc_runtime_inter_subjob_send：发送一个字节向量给另一个子任务

globus_duroc_runtime_inter_subjob_receive：接收另一个子任务的字节向量

globus_duroc_runtime_intra_subjob_rank：得到表示子任务进程的等级

globus_duroc_runtime_intra_subjob_size：得到本地子任务进程的个数

globus_duroc_runtime_intra_subjob_send：发送字节向量给 DUROC 子任务中的其它进程

globus_duroc_runtime_intra_subjob_receive：接收 DUROC 子任务中的其它进程的字节向量

DUROC bootstrap-library API：

globus_module_activate(GLOBUS_DUROC_BOOTSTRAP_MODULE)：激活 DUROC bootstrap-library API 实现

globus_module_deactivate(GLOBUS_DUROC_BOOTSTRAP_MODULE)：失效 DUROC bootstrap-library API 实现

globus_duroc_bootstrap_subjob_exchange：交换子任务之间的信息

globus_duroc_bootstrap_master_sp_vector：构造一个在主节点的 Nexus 启动点向量（Nexus 比较过时了）

globus_duroc_bootstrap_ordered_master_sp_vector：按一定顺序构造一个在主节点的 Nexus 启动点向量

7.3 资源描述语言－RSL

RSL 是一个描述资源的通用可交换语言，它提供了一个框架性的语法描述，可用来

组成复杂的资源描述。资源管理组件在 RSL 描述中引入特定的<attribute, value>对，每个属性/值对作为某种控制参数以控制对资源的各种操作。在 Globus 资源管理体系结构下的不同组件可以使用 RSL 描述来执行各种资源管理功能。下面是一些简单的 RSL 例子，讲解了如何对资源进行描述，这里主要是给读者一个感性认识，当阅读完下面各节后，相信会有一个清晰的理解。典型的资源描述至少包含如下一些关系式的合取式：

```
(* this is a comment *)
& (executable = a.out (* <-- that is an unquoted literal *))
  (directory    = /home/nobody )
  (arguments    = arg1 "arg 2")
  (count = 1)
```

当某个字符串在资源描述中多次使用时，可用替代来简化 RSL 构造：

```
& (rsl_substitution    = (TOPDIR    "/home/nobody")
                         (DATADIR $(TOPDIR)"/data")
                         (EXECDIR $(TOPDIR)/bin) )
  (executable = $(EXECDIR)/a.out
          (* ^-- implicit concatenation *))
  (directory    = $(TOPDIR) )
  (arguments    = $(DATADIR)/file1
          (* ^-- implicit concatenation *)
                   $(DATADIR) # /file2
          (* ^-- explicit concatenation *)
                   '$(FOO)'     (* <-- a quoted literal *))
  (environment = (DATADIR $(DATADIR)))
  (count = 1)
```

对上述 RSL 描述执行变量替换并去除注释后调度的 RSL 字符串：

```
& (rsl_substitution    = (TOPDIR    "/home/nobody")
                         (DATADIR "/home/nobody/data")
                         (EXECDIR "/home/nobody/bin") )
  (executable = "/home/nobody/bin/a.out" )
  (directory    = "/home/nobody" )
  (arguments    = "/home/nobody/data/file1"
                  "/home/nobody/data/file2"
                  "$(FOO)" )
  (environment = (DATADIR "/home/nobody/data"))
  (count = 1)
```

7.3.1 RSL 语法、标记和属性

1 语法

RSL 语法的核心是关系（relation）。关系把属性和属性值联系起来，如“=”关系的一个例子为” executable=a.out”，它指出了资源请求中要执行的程序名字。在 RSL 中有两个生成的合成结构可用来基于节点关系构造更复杂的资源描述，一个是复合请求

（compound requests），另一个是值序列（value sequences）。而且 RSL 包含一个手段可同时引入和废除字符变量。

复合请求的最简单形式是合取式请求（conjunct-request）。合取式请求可用来表示简单关系的联合，使得这些简单关系式之间是一种逻辑与的联系。在 Globus RSL 描述中，合取式请求是最常用的一种，如一个简单的任务服务请求是一种合取式请求，通常包括如下关系式：执行程序的名称、节点数、执行程序的参数、输出文件等。同理，核心 RSL 语法包括析取式请求（disjunct-request），这主要用来表示简单关系式之间是一个逻辑或的联系。然而当前的资源管理组件没有使用析取式请求。

最后一种复合请求的形式是多重请求（multi-request）。多重请求主要用于 DUROC 中。DRUOC 是资源管理系统中的协同分配组件，可实现对资源的并行使用。多重请求的形式与合取式请求和析取式请求的不同之处主要有两点：首先在多重请求中引入了变量范围的概念，这样可把变量的使用范围限定到多重请求的一个子句（clause）中，这样其它的子句不能“看到“这个变量；另一个不同之处是在资源描述中引入了不可约层次（non-reducible heirarchy）的概念，而在合取式请求中的关系式可看作是对资源的一种“约束”，多重请求中的子句可看成理解为简单的资源描述的集合。这样属于不同子句的相同属性不会产生逻辑上的冲突。

RSL 语法中属性值的最简单形式是字符串。用引号括起来的字符串可包含任何字符，而属性值也可以用宏来表示，这时属性值实际上是宏的定义。RSL 语法也包括用来表示属性值集合的值序，值序列语法用于定义变量和程序的参数。

一个 RSL 字符串包括一系列的 RSL 标记（token）、空格（whitespace）和注释（comments）。RSL 标记可以是特别的语法形式（如正则表达式等）或规则的没有引号括起来的字符。特别语法可一个和多个特殊的字符和常规字符。完整的特殊字符集有

“+”	“&”	“|”	“(”	“)”
“=”	“<”	“>”	“!”	“$”
“″”	“’”	“^”	“#”	

这些字符只用于特定的语法形式，或者用在括号中。括号字符可以是双括号或单括号。RSL 语句的注释用“(*”表示前缀，用“*)”表示后缀。

2 标记

RSL 字符串可被替换。通过使用"rsl_substitution"属性，字符串替换变量（string substitution variable）被定义为一个特殊的关系式，而且这些定义会影响同一合取式请求（或析取式请求）中的变量引用，也会影响多重请求中的合取式（析取式）子请求。每一个多重请求对某个子请求引入一个新的变量范围，且变量定义只在某个范围内。

在任何给定的范围内，资源描述中的变量定义按从左至右的顺序处理。最外层的范围比内层的范围先被处理，而且在内层中定义可增大新的或更新的继承的变量定义。

变量定义和变量引用只被处理一次，在处理下一个定义之前，先要根据前面的定义对环境进行更新。变量定义中的值可能包含对前一个定义的变量的引用。如果在变量处理过程中发现对某个变量的引用，还该变量还没有被处理，这用空字符串代替对这个变量的引用。

3 属性

RSL 语法是可扩展的。每一个 Globus 资源管理组件都引入了附加的属性，可被 RSL-aware 的组件识别。由于数量众多，，因此很难把所有的属性列出来。当一个 RSL 描述传给某个管理组件后，它先使用它识别出来的属性，然后把把没有识别出来的属性不经改变地传递给下一个组件。这样可把不同的资源管理功能组合成一个强大的功能集合。下面是已存在的资源管理组件使用的属性名：

通用**RSL**属性：

rsl_substitution

GRAM属性：

Arguments	count	directory	hostCount
executable	environment	jobType	maxTime
maxWallTime	maxCpuTime	gramMyjob	stdin
stdout	stderr	queue	project
dryRun	maxMemory	minMemory	

DUROC属性：

Label	resourceManager	subjobStartType
Contact	subjobCommsType	

7.3.2 RSL 语法和标记规则

RSL 语法可用一种修改过的 BNF 进行描述。下面是 RSL 的语法描述和标记规则。

specification **[RSL String]**

=> relation

=> '+' spec-list [multi-request]

=> '&' spec-list [conjunct-request]

=> '|' spec-list [disjunct-request]

spec-list

=> '(' specification ')' spec-list

=> '(' specification ')'

relation

=> 'variables' = binding sequence [variable def'n s]

=> attribute op value-sequence [relation]

binding-sequence

=> binding binding-sequence

=> binding

binding

=> '(' string-literal simple-value ')' [variable def'n]

attribute

=> string-literal [attribute]

op

=> '=' | '!=' | '>' | '>=' | '<' | '<='

value-sequence

=> value value-sequence

=> value

value

=> '(' value-sequence ')'

=> simple-value

simple-value

=> string-literal

=> simple-value '#' simple-value [concatenation]

=> implicit-concat

=> variable-reference

variable reference

=> '$(' string-literal ')' [variable ref.]

=> '$(' string-literal simple-value ')' [ref. w/ default]

implicit-concat

=> (unquoted-literal)? (implicit-concat-core)+ [implicit concat.]

implicit-concat-core

=> variable-reference

=> (variable-reference)(unquoted-literal)

string-literal

=> quoted-literal

=> unquoted-literal

quoted-literal

=> ' ' ((^']|('' '))* ' '

=> ' " ' ((^"]|(' "" '))* ' " '

=> '^' c((^c]|(cc))* c [user delimiter]

unquoted-literal

=> ([^ \t\v\n+&|()=<>!"'^#$])+ [nonspecial chars]

Comment

==> '(*' ((^*])|('*'[^]))* '*)' [comment]

7.3.3 GRAM RSL 参数

下面是 GRAM 的任务管理者用于定位并使用资源的 RSL 参数(“*”表示在 Globus2.0 中才有），读者从中可以了解到在 Globus 的资源描述主要反映了资源的哪些特性和属性：

1 **(directory=*value*)**

确定任务所在的目录

缺省值：由 gatekeeper 指定的当前工作目录。

2 **(executable=*value*)** *GASS enabled * required parameter *

远端的执行文件名字。如果属性值是一个 GASS URL，则在任务执行前，执行文件会传输到远端的 gass cache 中，且在任务结束后，会清除 gass cache 中的执行文件。

缺省值：无

3 **(arguments=value [*value*] [*value*] ...)**

执行程序的命令行参数。如果字符串中有空格，则要用引号括起来。

例子: (arguments= "a and b" ccc d)

argv[1]="a and b"

argv[2]="ccc"

argv[3]="d"

缺省值：NULL

4 **(stdin=*value*)** *GASS enabled

远端执行程序的标准输入的文件名。如果这个值是一个 GASS URL，则在任务

执行前，输入文件会传送到远端的 gass cache 中，且在任务结束后，在远端 gass cache 中的输入文件会被删除。

缺省值：NULL

5 **(stdout=*value*)** *GASS enabled

远端执行程序的标准输出的文件名。如果这个值是一个 GASS URL，则在任务执行前，输出文件会传送到远端的 gass cache 中；且在任务结束后，在远端 gass cache 中的输出文件会被删除。

缺省值：NULL

6 **(stderr=*value*)** *GASS enabled

远端执行程序的标准错误输出的文件名。如果这个值是一个 GASS URL，则错误输出在任务的执行期间动态地传输给用户。

缺省值：NULL

7 **(count=*value*)**

执行程序的个数

缺省值：1

8 **(environment=(*var* value) [(var value)] ...)**

除了任务管理者提供的环境变量外，附加的执行程序的环境变量

例子： (environment= (VAR_A value_a))

(environment= (JOE mama)(PI 3.1415))

C-shell 的例子：

```
setenv VAR_A value_a
setenv JOE mama
setenv PI 3.1415
```

Bourne shell 的例子:

```
VAR_A=value_a; export VAR_A
JOE=mama; export JOE
PI=3.1415; export PI
```

缺省值：NULL

9 **(maxTime=*value*)**

单个执行程序的最大墙上执行时间或最大 CPU 时间。由 GRAM 调度器决定选择哪个时间。时间单位是分钟。

缺省值：无

10 **(maxWallTime=*value*)**

单个执行程序的最大墙上执行时间。时间单位是分钟。如果 GRAM 调度器不能设定墙上时间，则返回一个错误。

缺省值：无

11 **(maxCpuTime=*value*)**

单个执行程序的最大 CPU 执行时间。时间单位是分钟。如果 GRAM 调度器不能设定 CPU 时间，则返回一个错误。

缺省值：无

12 **(jobType=*single*|*multiple*|*mpi*|*condor*)**

决定任务管理者如何启动任务。

single - 表示即使 count > 1，只启动一个进程或线程

multiple - 表示启动 count 个进程或线程

mpi　　- 表示使用适当的方法启动 MPI 程序。MPI 程序的个数有 count 值指定。

condor　- 表示在"condor"环境中启动 condor 任务。

缺省值：multiple

13 **(gramMyJob=*independent*|*collective*)**

这个值确定了 gram myjob 接口中启动进程中的行为。

independent - 表示即使 count > 1，只启动一个进程或线程。

collective - 表示函数 gram_myjob_count()将返回任务中整个进程的数目，函数 gram_myjob_rank()将返回本地进程的进程号(在 0 到 count-1 之间)

缺省值：collective

14 **(queue=*value*)**

确定任务将放置到调度器的哪个队列中。

缺省值：无

15 **(project=*value*)**

确定任务将分配到哪个项目帐号中。

缺省值：无

16 **(hostCount=*value*)**

只用于由 SMP 节点构成的集群环境中，如一个 IBM SP 系统。用于定义节点数以分配"count"个进程。

缺省值：无

17 **(dryRun=*yes*|*no*)**

如果 dryrun 等于 yes，这任务管理者不会执行任务，并成功返回。

缺省值：no

18 **(minMemory=*value*)**

确定任务运行需要的最小内存大小，单位是 Mege bytes。

缺省值：无

19 **(maxMemory=*value*)**

确定任务运行需要的最大内存大小，单位是 Mege bytes。

缺省值：无

20 **(save_state=*yes*|*no*) ***

决定任务管理者是否把任务状态保存到一个持久存储设备中。如果任务管理者崩溃，则用户可重新执行新的任务管理者，并根据保存的信息继续监视任务。

缺省值：no

21 **(two_phase=<*int*>) ***

对任务提交和任务完成实现两阶段任务提交

对于任务提交：任务管理者首先得到一个促使任务提交。然后在提交任务前，等待用户的信号。**two_phase** 的属性值指出任务管理者最多等待的时间（单位为秒）。如果任务管理者在规定时间内没有得到提交信号（或用户发出取消信息），则任务管理者将清除任务文件并退出。

对于任务完成操作：任务管理者发出 DONE 或 FAILED callback 信息后，就等待用户的提交信号。如果任务管理者得到信号后，它将清除相关文件并正常退出。如果在给定时间内没有得到信号且 save_state 等于 enable，则不清除任务的相关文件，并退出。当出现如下错误时，任务管理者在退出时，不会删除任务状态文件，因为它可能还会重新执行。

错误类型：

GLOBUS_GRAM_PROTOCOL_ERROR_COMMIT_TIMED_OUT
GLOBUS_GRAM_PROTOCOL_ERROR_TTL_EXPIRED
GLOBUS_GRAM_PROTOCOL_ERROR_JM_STOPPED
GLOBUS_GRAM_PROTOCOL_ERROR_USER_PROXY_EXPIRED

缺省值：无

22 **(restart=<*old JM contact*>) ***

启动一个新的任务管理者，而不是提交一个新的任务，同时对存在的任务进行监视。新的任务管理者将搜索前一个与本任务相关的任务管理者留下的任务状态文件。如果新的任务管理者发现这些文件，这可继续对任务进行监视，并可发送任务状态信息和 stdout/err 信息流。当如果老的任务管理者还存在，则返回错误值。

缺省值：无

23 **(stdout_position=<*int*>) ***

(stderr_position=<*int*>)

是一个 job restart RSL 的一部分，用来指出 stdout/err 信息流需要重新启动的位置。

缺省值：空

24 **(remote_io_url=<*url base*>) ***

把给定的 URL 指向一个文件，然后把 GLOBUS_REMOTE_IO_URL=<path to file> 放入任务的执行环境中。如果这作为 job restart RSL 的一部分，这需要更新文件的内容。这主要用于用户代理可能崩溃又重新启动的时候，GASS 服务器的端口号可能改变的情况。

因此通过使用环境变量 GLOBUS_REMOTE_IO_URL 使得任务可以从文件的内容中得到 url_base。例如，url base 的描述形式如下：

https://ept.mcs.anl.gov:43744

则任务可根据 url base，通过 GASS 或 globus-url-copy 命令得到多个文件。例如：

```
globus-url-copy http://ept.mcs.anl.gov:43744/bin/ls \
                        file:/some/dir/transferred_ls
/some/dir/transferred_ls
```

缺省值：空

7.4 小结

本章主要介绍了 Globus 中的资源分配管理、协同分配合 RSL 描述，GRAM 的主要任务是处理资源请求、执行远程应用执行、分配资源和管理活动等，并根据计算资源的情况把资源更新信息发送给 MDS。用户可根据 RSL 描述来发出任务请求并交与 GRAM 处理。Globus 中的 RSL 很有特点，它通过一些简单的语法组合就可以描述各种复杂的资源情况，这极大地方便了任务的分配和资源的访问。而 Globus 的资源分配并没有取代已有的资源调度软件（如 PBS、LSF、Condor 等），而是在它们的上面建立一个抽象层，来统一管理各种不同的本地资源调度软件。Globus 资源分配管理的下一步发展方向是与 Web Service 技术结合，使得目前的资源分配的界面能够与 Web Service 访问界面融合，并提供更大的灵活性和动态性。

思考题

1 GRAM 的特点和组成部分是什么？
2 请描述 RSL 的语法和参数。
3 试在 Linux 系统上建立一个 GRAM 的例子。
4 试用 GRAM-API 和 DUROC-API 编写简单的应用程序。
5 讨论在 Windows 操作系统上如何实现 GRAM？
6 试分析 Web Service 与 GRAM 的关系。

第8章 数据管理

Globus 的数据管理[9][16] [20][21]主要与远程数据传输[34][35][36]、远程文件 I/O 相关[30][33]。主要的组成部分有：全局二级存储服务－Global Access to Secondary Storage (GASS)、 网格 FTP 服务－GridFTP 和 Globus 复制管理－Globus Replica Management 等。通过 GASS 可简化在 Globus 环境中应用程序对远程文件 I/O 的操作，使得使用UNIX和标准C语言I/O库的应用程序基本不用改动就可在Globus环境中执行。GridFTP 支持第三方传输、断点续传、并行传输、与 GSI 结合的安全认证、缓存等特性。是网格计算环境中的数据传输工具。Globus 复制管理是一大类科学应用程序中需要考虑的重要问题，由于存在对大型远程文件的访问。Globus 复制目录－Globus Replica Catalog 通过把部分相关数据智能地放置在离科学应用程序最近的位置，使得科学应用程序可快速地对数据进行访问。本章主要针对这三个方面[14]进行介绍。

8.1 全局二级存储服务－GASS

8.1.1 GASS 简介

GASS 服务[24]主要用来支持网格计算环境下的远程 I/O 问题，而且针对网格计算环境中的文件访问模式进行了优化支持，具有用户控制管理网络带宽的能力。设计 GASS 的目标不是要建立一个通用目的的分布式文件系统，而是对下列高性能网格计算中常见的四种 I/O 模式进行有效的支持（如图 8-1 [24]所示）：（1）只读(read-only)访问整个文件模式：例如某些地理信息数据库、配置文件等，应用程序只需读取其中的信息，而不需要对其进行修改。可能会有多个用户同时对文件进行只读访问，但由于不会修改文件，所以不存在一致性问题。（2）共享写（shared write）访问单个文件模式：这是指如果多个用户对文件进行写存在，但文件的最终结果由最后一个用户所写的数据决定。例如一个并行应用程序中的所有进程都产生相同的结构，则它们写到同一个文件的内容是一样的。在这种情况下的一致性比较简单，只需在文件这一级实现就可以了。（3）只添加（Append-only）访问模式：例如对于一个记录程序活动情况的 log 文件，如果有多个用户对其进行写操作，则可能的输出结果是混乱的或是按记录顺序的。实际上这种情况下的一致性也比较简单。（4）无限制读写（unrestricted read/write）访问文件（只有一个用户）模式：由于只有一个用户对文件进行访问，所以不存在一致性问题。

上述访问模式在一致性问题上都比较简单，不用考虑全局一致性问题，所以 GASS 的实现也相对比较容易。GASS 不考虑如下两种 I/O 操作：多个用户对一个文件进行并行读写操作；多个用户通过不仅仅是只添加操作的其它操作合作产生一个文件。目前的策略主要是为了简化 GASS 的实现。

上述访问描述的另一个特点是所有的操作都是针对整个文件进行的，这样在单个读操作与写操作之间没有通信。简单地说，对于读操作，简单地把整个文件的数据传输到读用户所在节点，然后进行读操作；对于写操作，用户写完之后，要把整个文件传输到文件原来的位置。这使得上述操作不太适合于小规模的部分文件访问。但 GASS 也实现了一些功能使得可用来优化小规模的部分文件访问，这主要体现在 GASS 的数据移动策略和文件缓存操作上，下面两小节将对数据移动策略和文件缓存做进一步阐述。

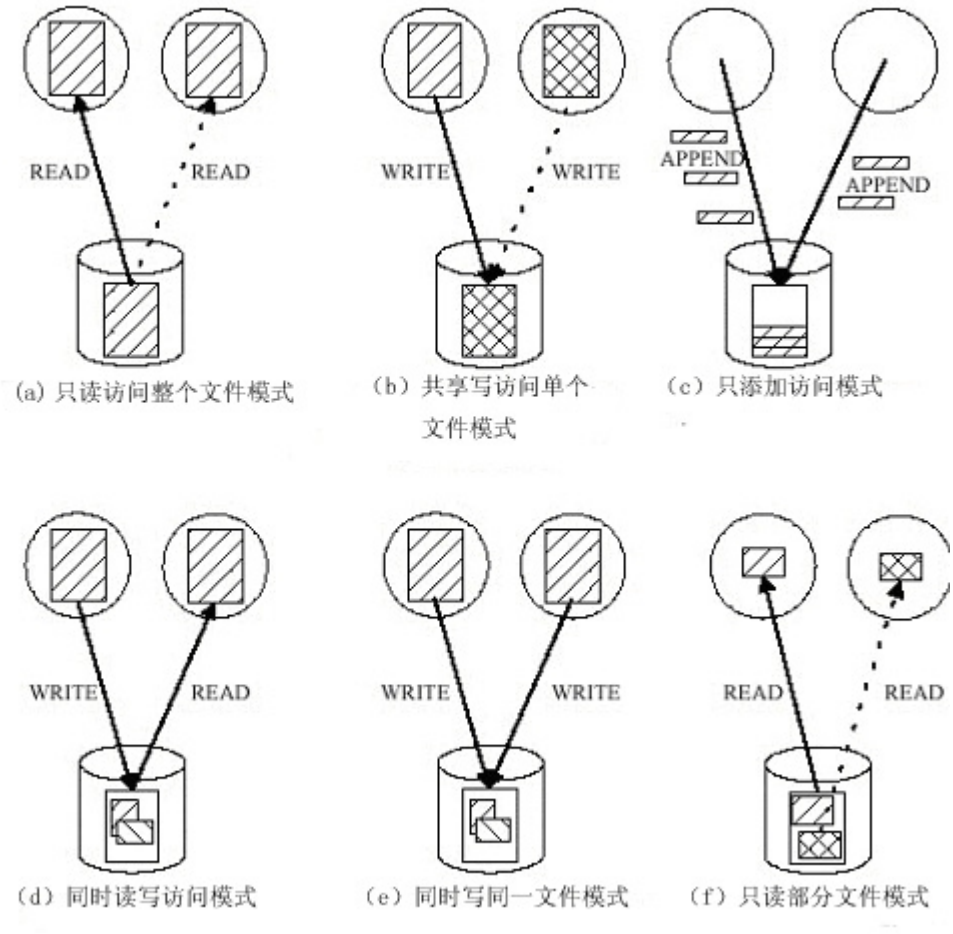


图 8-1 GASS 可优化的访问模式(a)-(c)和支持不够的访问模式(d)-(f)

8.1.2 GASS 缺省的数据移动策略

由于网格计算环境中的网络带宽有限，且具有动态和广域网的特点，所以 GASS 必须针对这些情况进行优化数据传输。典型的分布式文件系统对应用开发者隐藏了数据传输操作，而且一般只向应用开发者提供减少延迟而不是带宽的控制方法。

GASS 通过提供文件缓存－file cache 来解决带宽管理问题。文件缓存是一个“本地”二级存储系统，主要用于存储远程文件的拷贝。在缺省情况下，根据两种标准策略在文件打开时把相关数据放入文件缓存中，在文件关闭时，把文件缓存中的相关数据清除。GASS 缺省的数据移动策略主要有两个：

策略 1: 在第一次为读操作而打开文件时把文件拷贝取到本地并放入文件缓存中。本地文件缓存管理系统首先看这个文件是否已经在本地，如果没有则把整个文件的拷贝取过来，然后打开放在本地的文件拷贝，并把文件描述符返回给应用程序，最后增加整个本地文件拷贝的引用计数。然后应用程序就可以象访问本地文件一样对远程文件进行访问。这个策略适合于并行计算中多个进程读取输入文件的情况。由于一般需要读取整个文件，这样采用上述策略可有效避免多余的通信开销。但如果文件很大，或本地文件缓存太小，则应用程序可能要等很长一段时间才能进行访问。这就需要考虑其它策略了。

策略 2: 最后的应用程序对所写的文件进行关闭时对文件缓存中的文件拷贝进行传输并清除。当一个进行写操作的远程文件被关闭时，文件缓存管理系统对文件的引用计数进行检查莫如果计数值为 1，则文件拷贝要传输到相关的远程位置，然后从文件缓存中删除；如果计数值不为 1，则简单地把计数值减 1。这个策略减少了在同一位置的多个进程对远程文件的写操作开销。保证写之间的一致性有本地文件系统或应用程序完成，而且文件只传输一次，所以通信开销大大地减少了。如果一个远程文件用附加写模式打开，且文件在本地文件缓存中无拷贝，则采用通信数据流（如基于 TCP socket）对远程文件进行访问可能更好。

8.1.3 GASS 特殊的数据传输策略

GASS 提供了多种机制允许程序员对数据传输策略进行细化，并使得程序员可对特殊情况进行直接管理。这些机制主要分为两大类：相对高层的机制关注程序访问文件前和程序结束对文件访问后的数据处理（简称文件访问预处理和文件访问后处理）。文件访问预处理和后处理对大数据文件很有用。文件访问预处理等价于读模式文件打开调用，它负责在 cache 中无对应文件拷贝时，创建一文件项并传输文件到本地，这样接下来的文件 open 操作可直接在本地文件拷贝上进行。同理，文件访问后处理等价于写模式文件关闭调用。这两个处理可在应用程序内或外调用。如果在程序外调用，则可以避免对应用程序的源码进行修改。

底层机制主要是实现各种具体的数据传输策略。底层的 cache 机制对 cache 的行为提供了更细力度的控制。例如，一个用户可指定何时何地文件可被 cache。这种机制很重要，因为在 grid 环境中的应用程序访问的文件可能很大，这样采用典型的分布式文件系统策略来复制 cache 数据到预先分配的文件域可能不太切合实际。所以 GASS 允许在用户指定的地方缓存文件拷贝，即基于单个文件或单个用户的 cache 机制。

用户层控制 cache 位置的第二个好处是可以利用本地分布式文件系统的能力。例如，应用程序可通过 NFS、AFS、DFS 或 DPSS 等分布式文件系统访问缓存的文件拷贝。这也符合 Globus 的初衷：在各种本地服务之上建立一个跨域的服务层，把本地服务集成为一个统一的整体。

8.1.4 与 Globus 工具包其它服务的集成

GASS 服务可以与其它 Globus 服务进行集成。例如 GRAM 主要用于分配计算资源，如果一个应用程序想在一个远端资源上运行程序，它必须先对应的 GRAM 发出请求，并指出执行程序的名字、执行参数等参数。通过 GASS 可对 GRAM 进行扩展，使得 GRAM 的 API 允许执行程序名、标准输出、标准输入和错误输出等可用 URL 来命名，并且 GASS 可把执行程序缓存在 GRAM 本地，重定向标准输出和错误输出。

8.1.5 GASS-API

网格应用程序通过使用 GASS 来访问远程文件，GASS 的文件访问函数与 C 语言库中的标准文件访问函数类似，如 globus_gass_open、globus_gass_close、globus_gass_fopen、globus_gass_fclose）。这些函数进一步执行基于缺省的数据移动策略

的 GASS 管理操作。从应用程序的观点看，GASS 的 open 和 close 操作与标准的 UNIX I/O 操作中的对应操作类似，只是用 URL 描述代替你一般的文件名。而文件描述符和基于文件描述符的读写操作与 UNIX I/O 操作中的对应操作一样。

使用特殊的 open 和 close 函数调用意味着如果使用 GASS，就要修改某些程序。但这些修改相对很小，也是与 GASS 的文件描述语义一致。当然也可采用重新连接库技术（Condor 采用的技术）或通过操作系统内核支持（DFS 和 WebFS 采用的技术）来避免修改应用程序，但这增加了实现的复杂性并且可移植性不强。

GASS 的 open 操作用 URL 描述远端文件名、文件的物理地址和访问的协议。这样的一个好处是把文件服务器的位置显式地表示出来，这样应用程序可使用某些启发式信息对多个拷贝进行选择。如在图 8-2 [24]中，文件 F3 被复制到两个不同的服务器上，不同的进程可能选择离它们“最近”的文件拷贝。而且 GASS 可以访多种传输协议如 FTP、HTTP、HPSS、DPSS 等，这使得远程文件访问的灵活性更大。

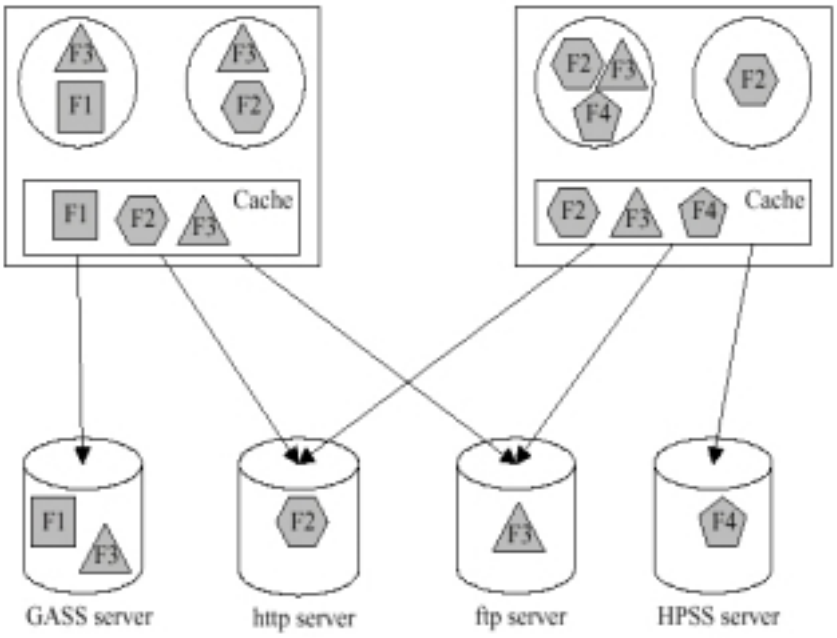


图 8-2 GASS 缓存结构

GASS 也提供了附加的底层 API，可用来实现特定的数据传输和访问策略。简而言之，GASS 提供了一系列的 API 来完成各种功能。

File Access API 主要提供高层的文件描述符合基于文件流的数据访问界面。caching 决定自动基于打开文件的参数。File Access API 是基于 cache management API 和 client implementation API 实现的，API 的操作是同步操作。

Cache Management API 主要提供了操作本地 cache 的底层函数，它允许用户直接控制插入操作、锁操作、删除操作和引用计数操作等。API 的操作时同步操作。

Client Implementation API 底层函数允许应用程序消除数据拷贝、选择传输单位尺寸、使用 proxy server 等。使用 Cache Management API 的操作主要是异步操作，主要可以与本地 I/O 操作进行重叠，以提高 GASS 操作的性能。

Server Implementation API 底层函数用来实现 GASS 远程文件服务器。通过使用 Server Implementation API 可以实现服务器方的 GASS 远程文件访问协议，并对应于应用相关服务代码以实现数据服务。Client Implementation API 和 Server Implementation

API 共享一个统一的数据查询模型。Server Implementation API 操作主要是异步的。

8.2 GRID FTP

许多应用程序使用不同的存储系统，并可对大型数据集合进行操作。虽然目前存在某些大型存储系统如 Distributed Parallel Storage System (DPSS)、High Performance Storage System (HPSS)等，但这些存储系统一般不兼容，采用了不同的协议和软件[18]。如果应用程序需要对多个存储系统进行访问，这必须采用多种访问方法。一个简单的处理方法是在这些特殊的传输系统上建立一个抽象层，但这样主要存在两个问题：（1）性能问题：由于增加了一个抽象层，也就增加了一层程序开销，而且对实现跨系统的数据传输的代价较大；（2）复杂性问题：建立和保持用户支持多种存储系统需要大量的编程工作，而且保证支持各种存储系统的更新和支持多种编程语言也需要大量的工作。

有没有一种方法既支持多种存储系统，又给用户提供一个统一的访问界面呢？为此，Globus 提出了一 GridFTP 机制[11]，它基于规范的 FTP 协议，并对其进行了全面的扩展，支持安全、高速的数据传输，可避免上面提到的问题。GridFTP 选择 FTP 协议作为基础，主要有三个方面的原因：（1）FTP 协议是在 Internet 环境中最规范的数据传输协议，且最可能满足 GRID 环境的需要；（2）FTP 协议有着大量成熟的实现且是易于理解的，是数据传输的标准协议之一，具有方便扩展的体系结构；（3）目前已有针对 FTP 协议的大量扩展协议和实现，其中一些对于 GRID 环境的数据传输很有用。

8.2.1 GridFTP 协议功能和实现

GridFTP 协议相对于 FTP 协议有许多新的特点，其中一些已经成为标准（可参考文献 RFC959）。首先 GridFTP 支持 GSI 和 Kerberos 安全机制，在 GridFTP 中支持灵活可靠的安全鉴别和完整性检查。而且用户可以控制 GridFTP 在不同层次上的数据完整性。GridFTP 的安全鉴别机制定义在 RFC 2228, “FTP Security Extensions”中

而且 GridFTP 支持第三方控制的数据传输。为了管理分布式通信中的大数据集，必须提供经过鉴别的第三方控制的数据传输。这样可允许一个用户或应用程序在某地监控其它两地之间的数据传输。在 GridFTP 实现中，增加了 GSS-API 安全认证，这样可更可靠和安全地支持第三方数据传输功能。

在数据传输方面，GridFTP 支持并行数据传输、条状（strip）数据传输和部分文件传输。在广域网环境中，GridFTP 使用多个并行的 TCP 流来提高数据传输的总的带宽，支持基于 FTP 的并行数据传输和数据通道扩展。在 Globus 环境中，大规模的数据可分布放置在多个存储点上，这称之为条状数据存储，GridFTP 可支持这样的条状数据传输来提高数据的传输速度，这样 GridFTP 的数据传输速度和带宽可进一步增加。某些应用可能只需要访问某个远程文件的一部分，这需要一定的数据传输支持，GridFTP 支持从文件的任意位置开始传输数据，可有效地支持部分文件传输。

而且 GridFTP 可自动调整 TCP buffer/window 大小，使用优化的 TCP buffer/window 大小设置可有效的提高数据传输性能。GridFTP 对标准的 FTP 协议进行了扩展，针对具体的文件大小和类型，可以支持对 GridFTP 中 TCP buffer/window 大小的手动和自动设置。GridFTP 还支持可靠传输和数据重传，对于许多应用程序而言，必须保证数据传输的可靠性，并需要支持容错的数据传输。GridFTP 扩展失败数据传输的重传，并把它扩展到新的数据通道协议中，这样可有效地支持可靠传输和数据重传。

为了实现 GridFTP 协议，Globus 项目主要实现了 GridFTP 函数库 globus_ftp_control_library 和 globus_ftp_client_library 、GridFTP 客户端、GridFTP 服务器端和一系列的相关工具。GridFTP 服务器端主要是基于自由软件－wuftpd 进行改写和扩展来实现的。Grid FTP 的客户端主要是基于自由软件－ncftp 进行改写和扩展来实现的。

GridFTP 函数库 globus_ftp_control_library 实现了控制通道 API。控制通道 API 主要提供了管理 GridFTP 连接（包括相互鉴别、创建控制和数据通道、这数据通道上读写数据）的功能。并且支持并行数据传输、条状数据传输和第三方数据传输等。

GridFTP 函数库 globus_ftp_client_library 主要实现 GridFTP 客户端 API。GridFTP 客户端 API 主要提供了高层客户端数据传输功能，包括完整的文件 get 操作和 put 操作，对并行数据传输进行控制和设置，部分文件传输操作等。

8.2.2 GridFTP 性能

图 8-3 [6]显示了 GridFTP 在两台工作站上进行 GridFTP 数据传输的性能。其中一台工作站在美国 Illinois 州的 Argonne National Laboratory，另一台在美国 California 州的 Lawrence Berkeley National Laboratory，二者之间提供 ES-Net network 进行连接。每个工作站运行 Linux 操作系统，具有 RAID 存储系统，其 RAID 存储系统的数据读写速度大约在 60Mbytes/Sec。在两台工作站之间最慢的网络部分是千兆以太网。图 8-3中下面的曲线是当并行的 TCP 流增加时 GridFTP 的数据传输性能。

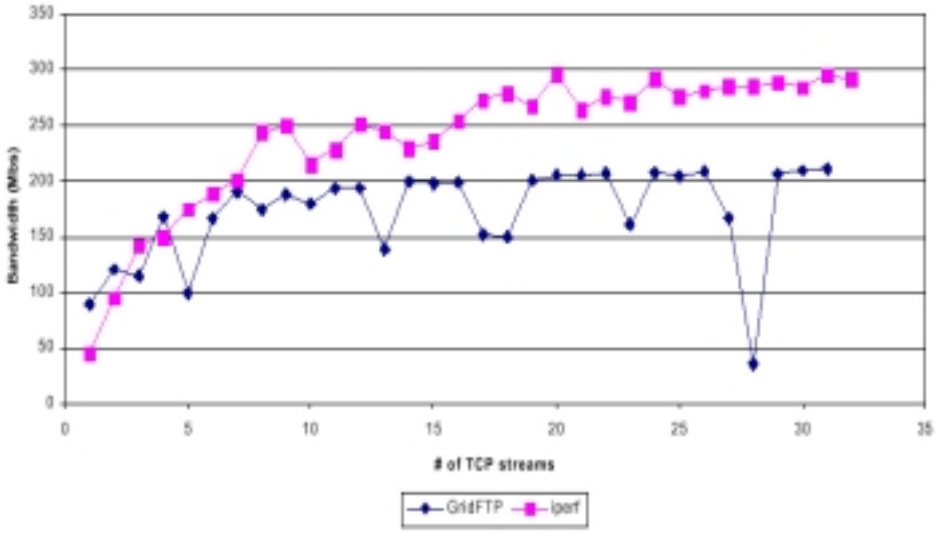


图 8-3 GridFTP 数据传输的性能

上面的曲线表示随着并行 TCP 流的增加，通过 iperf 得到的数据传输性能。iperf 是一个测量网络性能的工具，通过它可以得到网络的最大吞吐量。通过图 8-3可以看出当并行 TCP 流达到 7-10 条时，GridFTP 可达到 200Mbit/Sec 的通信性能。iperf 和 GridFTP 之间的性能差异主要是由于 GridFTP 中存在的安全鉴别开销、发送性能状态信息的开销和检查点设置开销等。根据计算，GridFTP 的性能可达到 iperf 性能的 78%。

图 8-4表示的是在一段时间内，两个节点之间进行 GridFTP 传输的情况，主要演示了 GridFTP 的可靠性。二者之间的带宽大约为 80Mbit/Sec，小于图 8-3显示的性能，

这主要受限于测试节点的磁盘性能。通过图 8-4 [6]可以看出当在传输过程中出现不同的网络问题（网络设备掉电、DNS 出错等）时，传输性能陡降，但当网络恢复时，数据传输又继续开始。这主要是因为 GridFTP 协议支持数据传输自动重传，这样一旦网络恢复正常，中断地数据传输可继续进行。

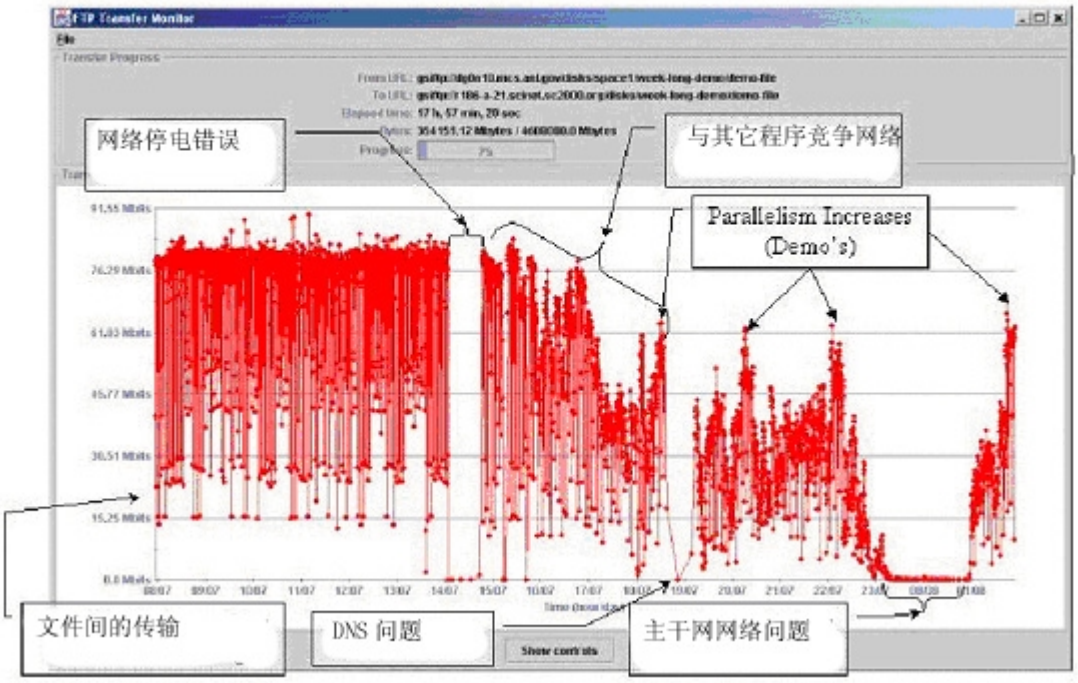


图 8-4　Dallas 和 Chicago 之间 14 小时内数据传输的带宽测量值

8.3 复制管理服务

8.3.1 复制管理服务简介

Globus 中数据管理的另一个基本服务是复制管理－Replica Management，Globus 复制管理主要针对大型远程数据文件的访问[38]。Globus 复制目录服务－Globus Replica Catalog 是 Globus 复制管理的一个核心部分[47]，它通过把部分相关数据智能地放置在离科学应用程序最近的位置，使得科学应用程序可快速地对数据进行访问。简而言之，复制管理主要管理数据集合拷贝的完整复制或部分复制，复制管理服务的功能主要包括：创建全部或部分文件集合的新拷贝、注册新的拷贝到复制目录（replica catalog）中以及允许用户和应用程序查询复制目录来找到所有存在的部分获全部文件集合的拷贝。

复制管理服务在网格体系结构中的 collective 层（如图 8-5 [6]所示），复制管理独立于具体存储系统的存储技术和数据移动协议。复制管理服务主要包括的组件有复制管理－replica management、元数据管理－metadata management、复制选择－replica selection[40]和复制与分布目录管理－management of replicated and distributed catalog。而与复制目录服务相关的组件有元数据目录－metadata catalog 和复制目录

－replica catalog，它们位于网格体系结构中的构造层。元数据目录组件主要保存与文件相关的描述性信息，而复制目录组件保存由复制管理组件注册的复制信息。

复制管理体系结构中的数据模型是基于文件的，即数据是按照文件来组织的。为方便起见，用户可把一组文件作为一个集合（collections）。一个 replica 或 location 指的是一个集合的子集，被存储在一个特定的物理存储系统中。由于存在多个存储系统，所以集合中的子集可能有重叠。

逻辑文件名是在数据网格的名字空间中的一个全局唯一的标识符。在复制管理体系结构中使用逻辑文件名，而表示物理文件名。而复制管理服务的一个主要任务就是把一个逻辑文件名映射到特定存储系统中的某个物理文件名上。

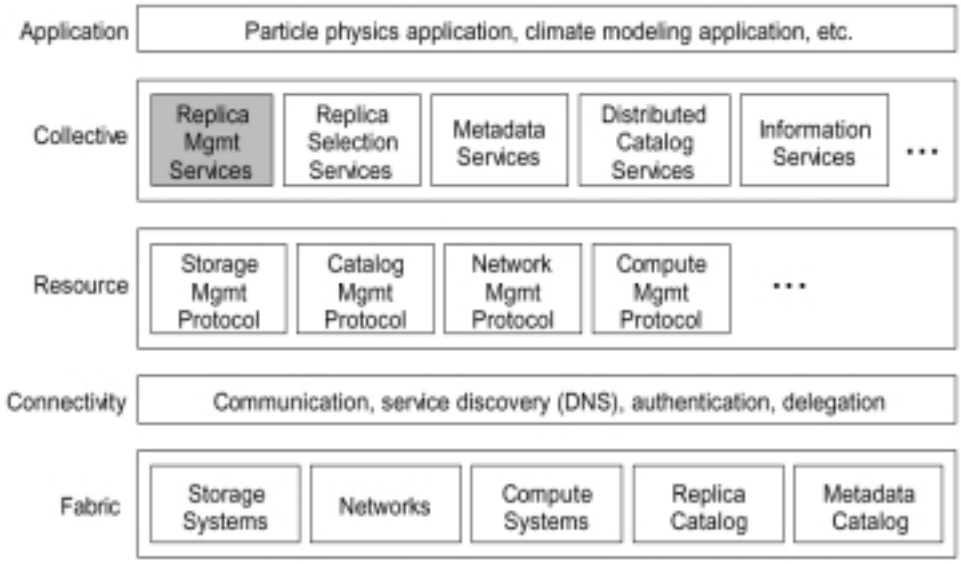


图 8-5 复制管理服务在网格体系结构的位置

8.3.2 复制管理服务的特点

1　复制信息与元数据（metadata）信息分离：

在复制目录中只保存在映射逻辑文件到物理文件过程中需要的信息。其它元数据信息（如文件的内容描述，文件的创建时间等）则由元数据管理服务进行保存。

在典型情况下，一个用户可首先查询元数据管理服务，根据文件的属性选择逻辑文件。一旦用户确定了逻辑文件，则用户可接着查询复制管理服务，以找到一个或多个存储逻辑文件的物理位置。

2　复制语义

一般而言，“replica”的语义是指“一个文件的拷贝要与文件本身保持一致”，这就要求支持上述语义的复制管理体系结构，要实现一个全面的支持广域网的分布式数据库，且在对拷贝进行修改或原子操作时要对文件加锁。

由于实现这样的分布式数据库过于复杂，所以在 Globus 项目中没有采用这样的方案。Globus 中的复制管理服务实际上没有按照“replica”的一般语义进行实现，而是对“replica”的语义放宽了限制。换句话说，对于文件的多个拷贝，Globus 中的复制管理服务并不保证它们的一致性。当用户注册文件为一个逻辑集合的复制时，这些文件之间的一致性由用户来保证。Globus 的复制管理服务不会主动执行任何操作来检查、并确保文件的一致性。

3　复制管理服务的一致性

虽然 Globus 的复制管理服务保证注册文件拷贝的一致性，但要保证存储在复制管理服务中信息的一致性。因为主分布式计算环境中，计算和网络失败是不可避免的，复制管理服务必须能够从计算或网络错误中恢复，并达到一个一致的状态。

4　回滚操作－ Rollback

在复制管理中的某些操作是原子操作，如果这些操作正常结束，则复制管理服务的状态才会更新。如果这些操作失败，则复制管理服务的状态保持不变。这些原子操作的例子有：对复制管理服务增加新的项，删除项或者对存在的项进行修改等。

而复制管理中的其它一些操作由多个步骤组成，且这些操作中任何一个步骤的失败都会导致整个操作的失败。Globus 体系结构不保证这些复杂的操作必须是原子性的，这样可能导致复制管理服务中的注册信息可能崩溃。使用 Globus 的复制管理服务保证：如果复制的操作失败，复制管理服务的状态会回滚到没有执行这个操作的前一个一致的状态。这要求复制管理服务要保存更多的信息。

5　无分布式锁机制

由于 Gloubs 的复制管理服务没有实现分布式锁机制，所以当用户改变或删除文件且没有调整复制管理服务时，会导致复制复制管理服务中的数据与实际情况不一致，使得复制管理服务中的数据可能会崩溃。Globus 不会阻止但不鼓励用户执行这些操作。如果通过分布式锁机制可以避免上述情况，但实现太复杂，所以 Gloubs 复制管理体系结构没有实现分布式锁，且不保证复制信息与实际情况的一致性。

8.3.3 复制管理服务的功能

复制管理服务中包含多种操作功能，主要有注册、修改、查询、存储等。现简要描述如下：

在复制管理服务中注册一个新项：

这个新项主要包括：一个包含逻辑文件列表的新逻辑集合、一个包含存在映射信息的新位置（这个映射信息反映了一个存在的逻辑集合中的文件子集）、在一个已有的逻辑集合中的新逻辑文件项（主要保存与文件相关的一些信息，如文件大小等）。

在复制管理服务中修改一个已有的项：

主要的操作包括：从一个已有的逻辑集合和位置项中增加或删除一个文件、从已有的逻辑文件项中删除与文件相关的描述属性。

查询复制管理服务：

主要的操作包括：根据特定的逻辑文件、逻辑集合或位置查找对应的项、寻找一个特定逻辑文件的物理拷贝的所有位置、返回与某项联系的需要的属性（对于一个逻辑集合项而言，返回集合中的文件名；对于位置项而言，服务映射逻辑文件名到物理文件名的属性。对于逻辑文件项而言，返回描述逻辑文件的属性）。

结合存储和注册操作：

主要的操作包括：把一个文件拷贝到目的存储系统，并把文件注册到对应的位置项中、把文件拷贝到目的存储系统，并把文件注册到对应的位置项和逻辑集合项中、从复制管理服务中删除项。

8.3.4 复制目录服务简介

复制目录（Replica Catalog）的功能主要是在逻辑文件名或逻辑集合与保存在物理存储系统中的物理文件之间建立映射关系。复制目录服务主要注册三种类型的项：逻辑集合（logical collection）、位置（location）、逻辑文件（logical file）。

逻辑集合是一个用户定义的文件组。这主要考虑到用户或应用程序可能需要处理一组文件，如果把这一组文件组合成一个逻辑集合统一处理，则可以减少复制管理的许多开销。**位置项**主要包含映射逻辑集合或逻辑文件到物理文件的映射信息。位置项可注册有关物理存储系统的信息，如主机名、端口号和协议等。位置项还包含可访问物理文件的 URL 表示。每一个位置项表示一个逻辑集合的完整或部分拷贝。一个位置项唯一对应于一个物理存储系统。位置项可显式地列出来一个逻辑集合中的所有文件。某个逻辑集合可有任意个相关的位置项，某个位置项包含一个文件子集。通过所有多个位置项，用户可方便地注册跨物理存储系统的逻辑集合。尽管注册和操作文件集合有对应用程序许多好处（减少文件访问开销等），但有时用户和应用程序也需要访问一些特定的单个文件。所以在复制目录中包含描述单个逻辑文件的可选项。**逻辑文件项**包含描述物理文件的全局唯一的逻辑文件名。

总地来说，复制目录可提供如下服务：注册一组文件为一个逻辑集合、注册一个逻辑集合（可以是部分集合内容或全部集合内容）的物理位置、注册一个逻辑集合中的特定逻辑文件的信息、修改复制目录中注册项的内容、对复制目录查询进行响应等。

Globus 工具包提供了一个复制目录的命令行工具：globus-replica-catalog，可通过它完成上述服务。globus-replica-catalog 有三个参数：HOST、OBJECT、ACTION，HOST 指出复制目录中的一个逻辑集合的位置，当前的描述形式是基于 LDAP 协议的，所以 HOST 的描述形式为：“ldap://host[:port]/dn”。OBJECT 指出复制目录中要进行操作的项，其格式为：

-collection
对集合进行操作
-location <location name>
对给定的位置进行操作
-logicalfile <logical file name>
对给定的逻辑文件进行操作

命令中 ACTION 参数指出来命令的具体操作，这里的操作主要分为三类：创建/删除整个项、增加/列出/删除项中的属性、增加/列出/删除逻辑集合和位置项中的文件名属性，具体格式如下：

创建/删除整个项

-create [<input file>] | -create <location url> [<input file>]| -create <size>

-delete

增加/列出/删除项中的属性

-add-attribute <attribute name> <attribute value>

-delete-attribute <attribute name> [<attribute value>]

-list-attributes [<attribute names>]

增加/列出/删除文件名属性

-add-filenames <input file> [-add-anyway]

-delete-filenames <input file>

-list-filenames

下面是使用 globus-replica-catalog 命令的一些简单例子：

注册逻辑集合

```
globus-replica-catalog \
-host “ldap://ldap.cs.tsinghua.edu.cn:389/lc=newCollection,\
rc=myCatalog,o=HPC National Laboratory,o=Globus, c=CN”\
-manager “cn=Diretory Manager,o=HPC National Laboratory,\
o=Globus, c=CN” -password pwfile \
-collection -create listOfCollectionFilenames
```

列出位置的所有属性

```
globus-replica-catalog \
-host “ldap://ldap.cs.tsinghua.edu.cn:389/lc=newCollection,\
rc=myCatalog,o=HPC National Laboratory,o=Globus, c=CN”\
-manager “cn=Diretory Manager,o=HPC National Laboratory,\
o=Globus, c=CN” -password pwfile \
-location “location1” -list-attributes
```

删除逻辑集合

```
globus-replica-catalog \
-host “ldap://ldap.cs.tsinghua.edu.cn:389/lc=newCollection,\
rc=myCatalog,o=HPC National Laboratory,o=Globus, c=CN”\
-manager “cn=Diretory Manager,o=HPC National Laboratory,\
o=Globus, c=CN” -password pwfile \
-collection -delete
```

图 8-6 [6]显示了一个用于天气模型的复制目录的例子。这个目录包含两个逻辑集合，是 1998 年和 1999 年的二氧化碳的测量数据。1998 年的逻辑集合有两个物理位置：一个部分集合放置在 jupiter.isi.edu，另一个完整集合放置在 sprite.llnl.gov。位置项包含描述存储在特定物理位置的文件列表的属性，还包含从逻辑文件名道 URL 表示的物理文件位置的映射属性。这个例子还包含在集合中的每个文件的逻辑文件项，这些文件项描述了文件的大小等信息。

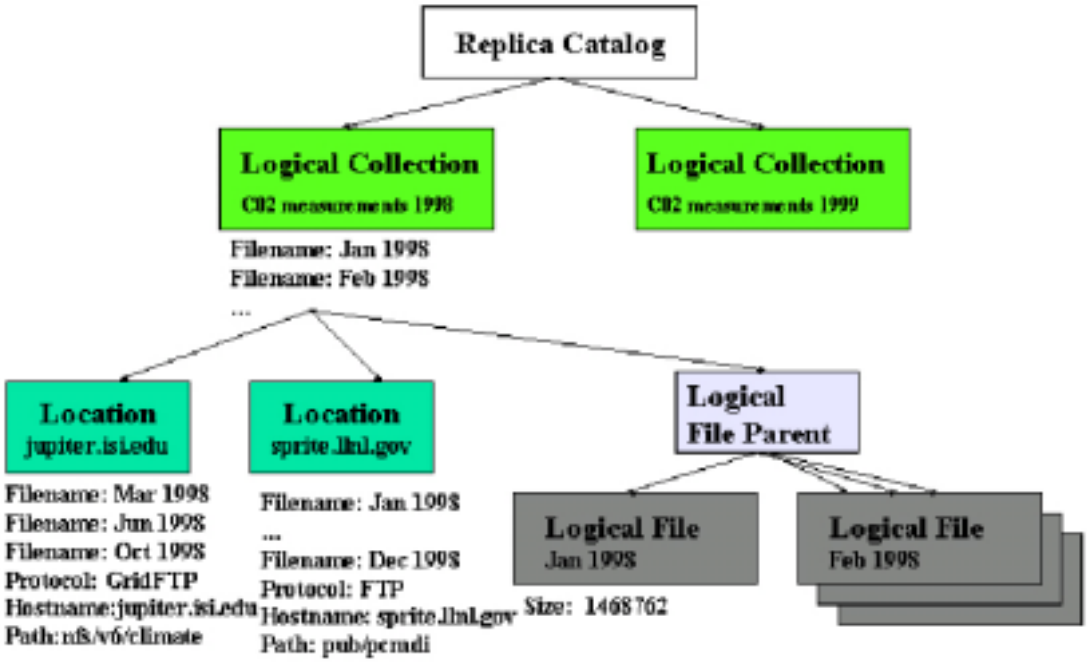


图 8-6　一个用于天气模型的复制目录的例子

8.3.5 复制管理服务 API

复制管理服务提供了一组 API 以方便用户开发网格应用程序。复制管理分配 API 可分为两大类：会话管理应用编程接口－session management API 和目录创建/文件管理应用编程接口－catalog creation/file management API。

会话管理应用编程接口提供了管理会话句柄（session handle）和会话属性（session attribute)的功能，可以缓存与 GridFTP 服务器和 LDAP 服务器的连接状态，会话句柄是指一个数据结构，它包含连接复制管理系统的状态和配置信息，而且内嵌一个处理属性的数据结构，这些属性描述了 FTP 客户端和连接复制目录的属性。而且目录创建应用编程接口还提供了对一个操作回卷（rolling back）的功能，这样可使得当前操作失败后，复制目录可返回到当前操作的前一个状态。会话管理应用编程接口提供的重起（restart）功能可重新这些当前操作，这主要是通过查询存储在复制目录中的 rollback 记录来实现的。

目录创建/文件管理应用编程接口主要提供了对复制目录进行各种操作的功能，首先是对目录项的操作功能－创建目录项，在复制目录中创建空的逻辑集合项和位置项，这样子复制目录中会增加一个包含逻辑文件列表的新逻辑集合、一个包含存在映射信息的新位置（这个映射信息反映了一个存在的逻辑集合中的文件子集）和在一个已有的逻辑集合中的新逻辑文件项（主要保存与文件相关的一些信息，如文件大小等）。另一类功能主要是文件操作，包括注册文件，即把一个复制目录已知的存储系统上的文件信息记录下来，不拷贝文件本身；发布文件，即把一个复制目录未知的存储系统上的文件信息记录下来，由于复制目录中没有文件所在的存储系统的记录，所以需要把文件拷贝到一个复制目录已知的存储系统中，并进行记录；拷贝文件，即把对应同一逻辑集合的文件在两个复制目录已知的存储系统之间进行拷贝；删除文件，即把文件信息从复制目录中清除，也可以选择把文件从对应的存储系统中删除。

复制管理服务 API 还提供了查询复制目录信息的功能：（1）根据特定的逻辑文件、逻辑集合或位置查找对应的项；（2）寻找一个特定逻辑文件的物理拷贝的所有位置；（3）返回与某项联系的需要的属性，其中对于一个逻辑集合项而言，返回集合中的文件名；对于位置项而言，服务映射逻辑文件名到物理文件名的属性；对于逻辑文件项而言，返回描述逻辑文件的属性。

8.3.6 复制管理服务的处理流程

复制管理服务与其它服务(replica selection 和 metadata management 等)是正交的（即相互之间不依赖对方）。图 8-7描述了当一个网格应用访问这些服务来确定最佳数据传输位置的情况。设存在一个天气模型模拟应用程序需要 1998 年的数据，应用程序不知道确切的文件名和需要的数据位置，为了得到 1998 年的数据，需要执行如下步骤：

1. 应用程序首先描述出需要的数据特征，然后把属性描述传递给元数据目录服务。
2. 元数据目录服务根据应用程序提供的属性描述查询包含这些属性的索引,产生出包含符合应用程序所需特征的数据的逻辑文件列表,并把逻辑文件列表返回给应用程序。
3. 应用程序收到元数据目录服务传来的逻辑文件列表后,把这些逻辑文件名传递给复制管理服务。
4. 复制管理服务收到应用程序的逻辑文件后，查询注册表，把注册的逻辑文件相对应的物理位置信息返回给应用程序。
5. 应用程序把返回的复制位置列表传给复制选择－replica selection服务
6. 复制选择服务根据列表确定从复制位置到应用程序之间的连接,并把这些信息发送给信息服务（即MDS）。
7. 信息服务通过这些连接之间的传输性能评估数据发送给复制选择服务。
8. 复制选择服务根据这些评估选择一个最佳的位置并把包含选择数据的位置信息发送给应用程序。
9. 应用程序收到包含数据的最佳位置后，就公钥通过GirdFTP或其它手段进行数据传输了。

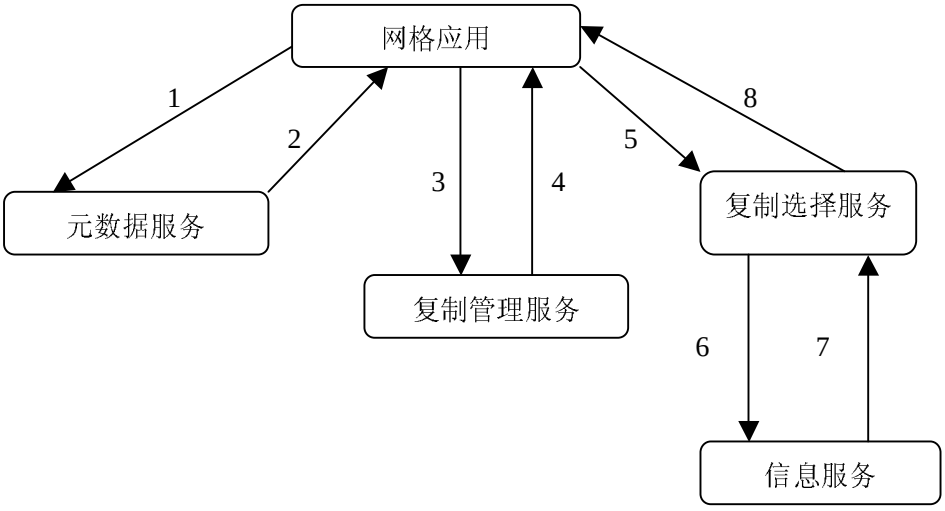


图 8-7 一个网格应用确定最佳数据传输位置的服务访问流程

8.4 小结

Globus 的数据管理主要包括远程数据传输、远程文件 I/O 等。主要的组成部分有：全局二级存储服务、 GridFTP 和 Globus 复制管理等。通过 GASS 可简化在 Globus 环境中应用程序对远程文件 I/O 的操作，使得使用 UNIX 和标准 C 语言 I/O 库的应用程序基本不用改动就可在 Globus 环境中执行。GridFTP 支持第三方传输、断点续传、并行传输、与 GSI 结合的安全认证、缓存等特性。是网格计算环境中的数据传输工具。Globus 复制管理通过把部分相关数据智能地放置在离科学应用程序最近的位置，使得科学应用程序可快速地对数据进行访问。网格环境下的远程数据传输一直是一个研究的热点，在理论有许多策略和方法，如何在网格计算环境中实现还有待于下一代 Globus 数据管理的研究与发展。

思考题

1 GASS、GridFTP 和复制管理的特点是什么？
2 试在 Linux 系统上建立 GridFTP 服务器。
3 试用 GASS-API 和复制管理服务 API 编写简单的应用程序。
4 讨论在 Windows 操作系统上如何实现 GASS、GridFTP 和复制管理组件？

第9章 网格应用程序开发工具

本章讲述的网格应用程序开发工具和编程环境不属于 Globus 工具包，但与 Globus 项目紧密相关，是建立在 Globus 工具包之上的高层网格应用开发工具和应用环境，如 CoG Kits（Commodity Grid Toolkits）、MPICH-G2、Condor-G[37]、MyProxy.[39]等。其中 MyProxy 主要用于提供一个保存、检索和访问与用户相关的安全认证证书的服务器开发工具和环境；Condor-G 主要在用于集群环境的支持容错的任务调度工具－Condor 的基础上，结合 Globus 工具包实现的一个支持网格计算环境的任务调度工具和开发环境；CoG Kits 主要用来开发基于网格的特定应用门户站点和各种网格管理应用；MPICH-G2 是一个用于网格环境的 MPI 编程环境和工具，可用于开发基于网格环境的并行程序。本章主要就 CoG Kits 和 MPICH-G2 这两个方面进行介绍。

9.1 CoG Kits 简介

Grid 计算环境给开发者提供了一个新的领域，网格开发考虑的重点在于端到端的性能、先进的网络服务支持、动态自适应等问题，但在目前的商业应用开发中，主要考虑的问题是商业应用的可扩展性，基于组件的封装、基于桌面的表示等。这使得网格计算的基本的开发技术和手段与通常的基于商业分布式计算的技术不一致。Commodity Grid 项目的创立，就是为了在二者之间建立一个桥梁，使开发者比较容易地建立起各种网格应用。Commodity Grid 项目主要推出了 Commodity Grid Toolkits（简称 CoG Kits），它定义了 Grid 和实际商业框架之间的一个映射和界面，提供多种语言支持，并定义和实现了一系列的通用构件，在很大程度上方便了网格应用开发者的开发工作（如图 9-1所示[19]）。

在 CoG Kits 的发展计划中[19]，将要实现 Web/CGI CoG Kit、Java CoG Kit、CORBA CoG Kit、DCOM CoG Kit、Python CoG Kit 等。目前的开发主要集中在 Java CoG Kit 和 Python CoG Kit，其主要原因在于这两种语言的跨平台性和可扩展性比较好，并且使用这两种语言的开发者众多[46][52]。

9.2 基于 CoG Kits 的应用

目前使用 CoG Kits 开发[53]的典型应用是基于网格的科学门户－Science portal：它是一个方便特定领域的科学家或研究人员的访问点（可以基于桌面、Web 界面或掌上设备 PDA 等），通过这个访问点可以无缝地对各种相关的信息资源和计算资源进行访问和协作[23]。

例如 一个农业研究人员使用农业 secience portal 对农作物生产进行规划。他要解决如何使农作物产量的价值最大化的问题。首先他需要得到一个准确的天气预报来决定何时种植庄稼；同时由分布在农田里的传感器帮助他判断庄稼对肥料的需求情况。传感器的数据首先被研究人员存储在数据库中，可接受反馈已改进数据模型。而访问点是一个计算机终端，通过网络（可以是无线网或其它网络设备）连接到数据库、经济市场监视服务系统（可监视农作物的市场价值）。用 CoG Kit 和 Globus 工具包可设计一个专业的农作物网格科学门户，使得农业研究人员根据农作物的市场价值、气候、土地情

况等各种因素选择种植何种农作物、何时种植等问题，并得到最大的产值。

一般而言，Secience Portal 的构建需要把不同领域的技术结合起来。由于需要访问不同的数据，使用 CoG Kit 和 Globus 工具包要能够访问不同的信息资源，而且能够通过计算资源进行复杂的计算。为了有效地支持远程访问，高性能网络是一个重要的条件。如果从 CoG Kit 的角度考虑，主要是实现两种技术的结合：（1）商业技术－强调基于桌面环境的易用性和代码可重用性，主要包括 GUI 组件、组件库、描述性语言、被工业界接受的分布式计算框架，企业级数据库、面向对象的语言和框架等。（2）网格技术－强调高性能、大规模、跨学科的广域网环境，主要包括远程计算、信息服务、高速数据传输、信息安全等。

由于目前的商业开发技术和网格技术中有一定的重叠，且可以相互借鉴，所以对各种技术的选择是需要认真考虑的。最终的目的是通过 CoG Kit 使得 1+1>2。为了支持快速原型开发和基于 CGI 的 WEB 编程，CoG Kit 提供了 python 接口；为了支持图形界面、跨平台和分布式开发，CoG Kit 提供了 Java 接口，而且 Java CoG Kit 是其中最主要的部分。

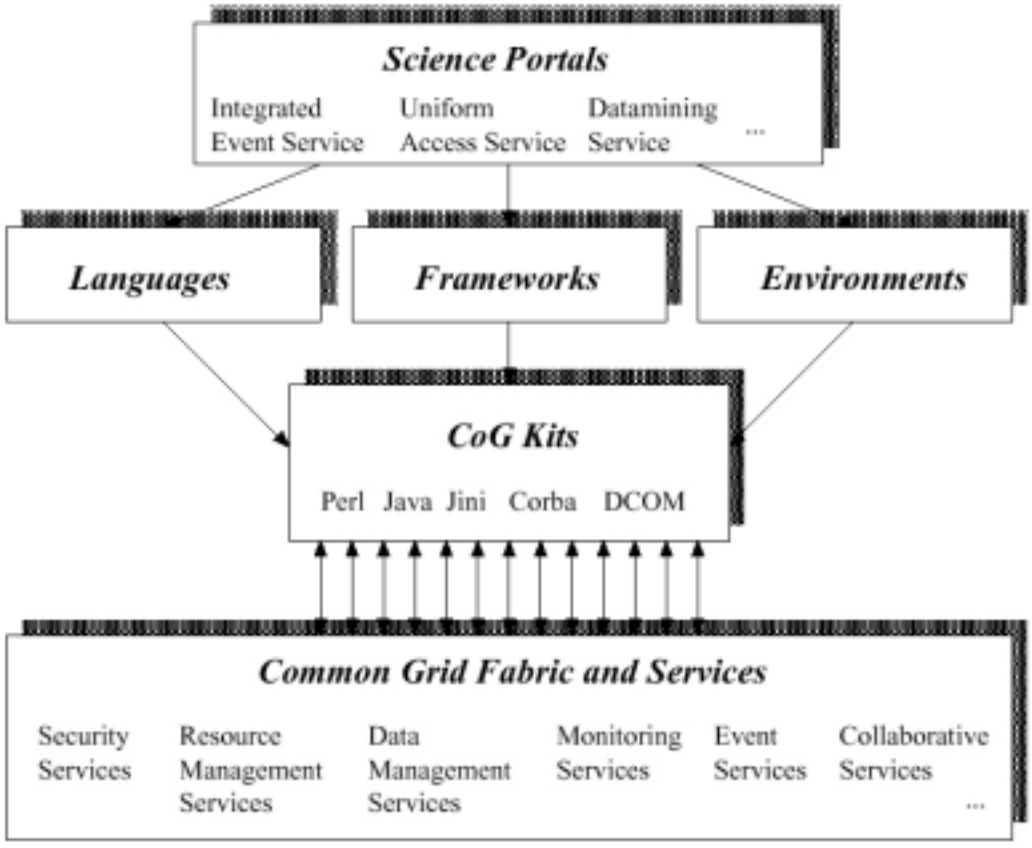


图 9-1　CoG Kit 在网格环境中的位置

9.3 Java CoG Kit 设计与实现

Java CoG Kit 的设计目标是方便开发基于 Java 和 WEB 界面的网格应用，并且支持迭代的开发过程。Java CoG Kit 主要提供了如下与 Globus 服务进行连接的组件：

- 底层网格界面组件（Low-Level Grid Interface Components）：提供了一个

到网格服务资源的映射，这里的网格服务资源是基于 LDAP 的信息服务－Globus MDS、资源管理服务－Globus GRAM 和 Globus DUROC、数据访问服务－Globus GASS、安全服务－Globus GSI 等。

- 底层实用工具组件（Low-Level Utility Components）：可被重用的各种有用的组件，如对基于 XML 或 RSL 的任务描述进行验证的工具组件、使用信息服务寻找计算资源的组件、定位计算资源的组件等。
- 通用底层 GUI 组件（Common Low-Level GUI Components）：提供一组可重用的底层 GUI 组件可用于 LDAP 属性编辑器、RSL 编辑器、LDAP 浏览器等软件的开发。
- 应用相关 GUI 组件（Application-specific GUI Components）：用于简化应用程序和基本 CoG Kit 组件组件的差异。如股票监视器、图形化的天气数据显示组件、对天气数据的图形化搜索引擎等。

下面的代码片段显示了如何用 CoG Kit 实现一个气象网格访问点(Climate Portal)[49]，从中可以看出通过 Java CoG Kit 可简单快速地服务各种基本的网格服务并实现特定的访问点服务。假设有一个高性能计算机保存有天气模型和数据，并通过计算得到结果，程序的输出格式上 GrADS（一种用于描述三维空间气象数据的文件格式），通过对这个例子的讲解，对 CoG Kit 中的各类组件也进行了分析。

用 CoG Kit 实现气象网格访问点的代码片段

```
// 步骤 0. 初始化
MDS mds=new MDS(“www.globus.org”,
        389,“o=Grid”);
// 步骤 1. 寻找一个可用的机器
result = mds.search
  (“(objectclass=GridComputeResource)
  (freenodes=64))”,“contact”);
//步骤 1.a) 选择一个机器
machineContact=< select the machine with
  minimal execution time from
  the contacts that are
  returned in result>

// 步骤 2. 准备试验数据
// 步骤 2.a) 搜索天气数据并返回
// attributes: server,port,directory,file
dn = mds.search
  (“(objectclass=ClimateData)(year=1999)
    (region=midwest”,“dn”, MDS.SubtreeScope);
result = mds.lookup (dn,“server port directory file”);

// 步骤 2.b) 把数据下载到机器中
url = result.get(“server”)+“:”
  + result.get(“port”)+“:”
  + result.get(“directory”)+“/”
  + result.get(“file”);
```

```
  data = server.fetch (url, machineContact);

// 步骤 3. 准备运行天气模型的脚本
RSL rsl = new RSL(“(executable=climateModel)
  (processors=64)
  (arguments=-grads)(arguments=-out map.grads)
  (arguments=-in ” + data.filename +”)”);
// 步骤 4. 提交任务
GramJob job = new GramJob();
job.addJobListener(new GramJobListener() {
   public void stateChanged(GramJob job) {
    // react to job state changes
   }
 });
try{
  job.request(machineContact, rsl);
  }catch (GramException e){
  problem submitting the job
}
```

下面是 CoG Kit 中各种组件的一个简单介绍：

1. 底层网格界面组件主要有如下几个：

RSL：

org.globus.rsl 包用来提供创建 RSL 表达式、操作 RSL 表达式、检查 RSL 表达式合法性的功能。如用 CoG Kit 实现气象网格访问点的代码片段的第三步所示。

GRAM：

org.globus.gram 包提供了一个道 Globus GRAM 服务的映射，这样允许用户调度和管理远程计算。具体的功能有提交任务、绑定已提交的任务、取消任务、判断是否可提交需要特定资源的任务、计算任务的状态（任务状态包括 pending、active、failed、done、suspended）。如用 CoG Kit 实现气象网格访问点的代码片段中的第四步根据一个 RSL 的描述用 Gram 类创建了一个 job 对象。在 Java 的实现中，采用了事件模型来传输 Globus GRAM 中的回调信息，即提供 GramJobListerner interface 来搜集回调信息。

DRUOC：

org.globus.duroc 包可用于协同分配多个资源。其实现机制与 org.globus.gram 包类似，也是基于事件模型。与 org.globus.gram 包不同的地方是 org.globus.duroc 包允许程序员创建和监视多请求的任务。

MDS：

org.globus.mds 包简化了对 Globus MDS 的访问。它的主要功能包括：与一个 MDS 服务器建立连接、查询 MDS 内容、显示 MDS 内容、与 MDS 服务器断开连接。org.globus.mds 包提供了一个中间应用层，可用统一的函数方便地访问不同的 LDAP client 库，如 JNDI、Netscape SDK、Microsoft SDK 等。根据用 CoG Kit 实现气象网格访问点的代码片段中的第一步可以看出，初始化 MDS 类的参数十 MDS 服务器的 DNS 名称、建立连接的端口号和指定目录树的 DN（distinguished name）。

第二步的 a 部分执行的查询操作，第一个参数是要查询的目录树的顶层信息，第二个参数指定了 LDAP 查询信息，第三个参数指定了查询的范围。

GASS：

Globus Access to Secondary Storage（GASS）服务简化了应用程序的移植和运行。org.globus.gass 提供了一个 GASS 服务的基本子集，支持不同计算机组件的文件拷贝等。get 方法用于把远程文件拷贝到本地，put 方法用于把本地文件拷贝到远地，fetch 方法与 get 方法类似。

2. 底层实用工具包

当前 CoG Kit 提供的 low-level utility 类提供了把抽象数据类型表示成无环图和对 XML 进行解析的函数。graph 类可访问任务之间的依赖关系，这在科学访问点应用中很有用。用 XML 中的 DTD（Document Type Definition）可有效地检查是否客户端与服务器端之间数据传输的格式。任务间依赖性分析是对 Globus 底层应用界面的有效扩展。而且 low-level utility 类定义了一个机器和任务 broker 界面。这使得程序员可自定义机器和任务依赖性。

3. 底层 GUI 组件

Java CoG Kit 的底层 GUI 组件可提供基本的图形组件用来构建基于 GUI 的网格应用程序。这些基本的图形组件包括显示格式化 RSL 表达式的 text panel 组件、显示 MDS 查询项的 table 组件、显示 MDS 目录信息树的 tree 组件等。在将来的 Java CoG Kit 可能把基于 GUI 的 Java bean 集成到 JBuilder 或 VisualCafe 等 Java 集成开发环境中。

4. 高层的图形应用组件

高层的图形应用组件[50]把不同的底层 CoG Kit 组件组合起来形成一个特定图形网格应用。

Graph Enabled Console Component（GECCO）是一个执行和监视一组有依赖关系的任务地图形工具。通过 GECCO 可以用图形化的方式或基于 XMLde 配置文件指定任务和它们的依赖性；而且在任务提交前可调试任务描述并纠错；并且可根据 log 文件，用图形化的方式执行并监视任务。如图 9-2所示[19]，每个任务在图中表示为一个节点，一个任务的父节点执行后，其自身才能执行。有关任务的描述可通过点击节点弹出一个窗口，可对窗口中的 RSL、label 和其它参数进行修改。

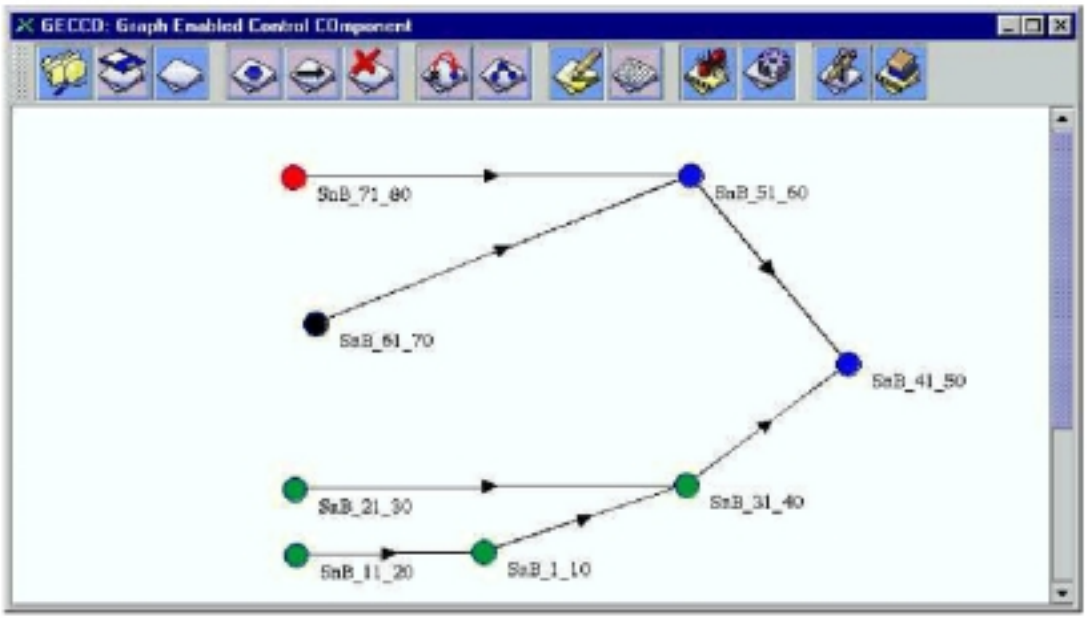


图 9-2　GECCO 组件

交互式的图形化资源协同分配器－interactive Graphical Resource Co-allocator（GRC）[51]如图 9-3所示[19]。GRC 允许用户来建立一个应用程序所需资源的网络化表示，并描述出这些资源是如何使用的。通过自动或手动技术来保证进行资源选择，并自动生成 RSL 描述。MDS 服务可用于自动发现可能满足用户需求的资源。一旦用户找到合适的资源集合，则通过 GRAM 和 DUROC 客户端的库函数实现网格应用程序的执行、监视和终止等。

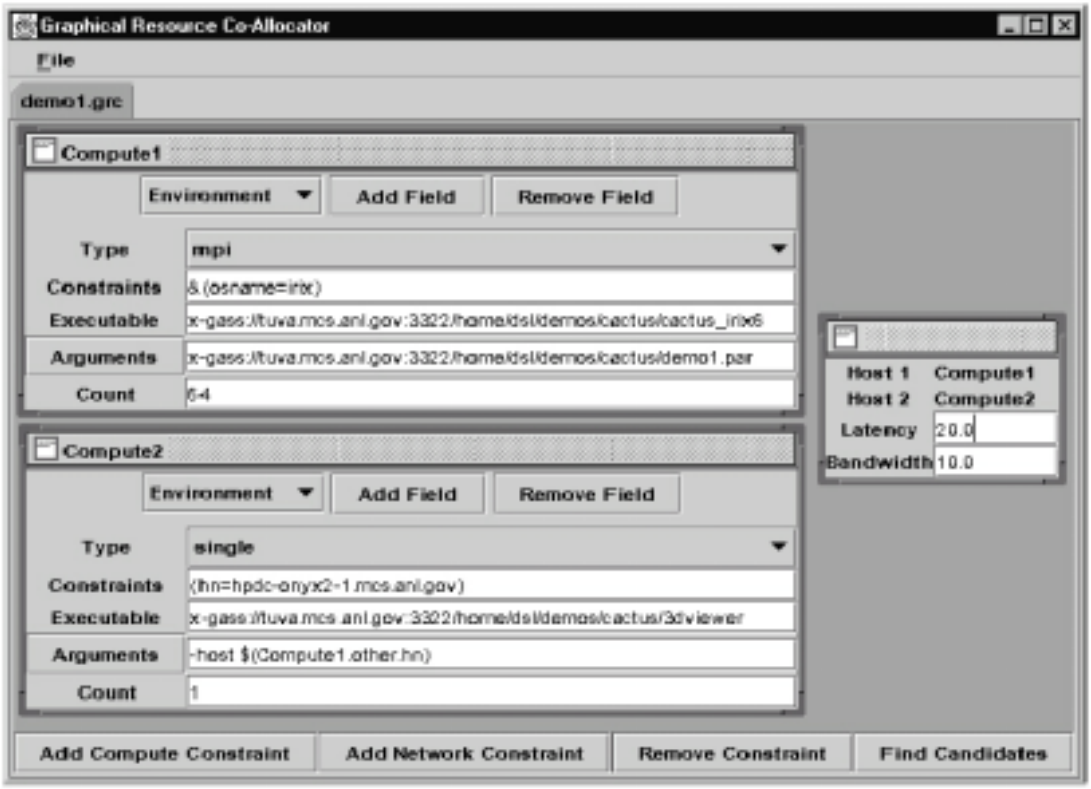


图 9-3 GRC 组件

9.4 MPICH-G2 简介

MPI（Message Passing Interface）是消息传递并行编程模型的一种具体实现和主要代表，MPI 在并行计算领域已广为普及并成为事实上的标准。MPI 标准化涉及到大约 60 个国家的人们,他们主要来自于美国和欧洲的四十多个组织，这包括并行计算机的多数主要生产厂商、大学、政府实验室等。1994 年 MPI 标准正式推出，并于 1995 年做了进一步的修正，定为 MPI1.1 标准。目前，MPI 的讨论仍在继续，补充了一些新特征的 MPI2 标准也已正式推出。

MPICH-G2 是一个用于网格计算环境的 MPI 实现[43][41]，符合 MPI 1.1 标准。通过使用 Globus 服务,MPICH-G2 可以把不同体系结构的计算节点联合起来进行并行计算，并提供标准的 MPI 编程接口，所以一般熟悉 MPI 编程的人员可很容易地使用 MPICH-G2。

MPICH-G2[44]主要用来解决两类具有分布特性的问题,一类是本身就具有分布式特性的问题，如远程可视化应用问题（计算与显示在不同的节点上完成），具体的应用项目有 Idesk 和 CAVE 等。另一类是设计上具有分布式特性的问题，如设计一个应用，通过跨广域网的多个节点协同完成某个计算工作，具体的应用项目有 Grid Application Development Software (GRADS) Project 等。而且通过 MPICH-G2 可以使得以前基于局域网系统或 MPP 系统的 MPI 并行程序很容易地在网格计算环境中运行,例如一个有名的并行计算应用项目 Cactus[48]完成了把本身应用程序移植到网格环境的转移，获得了 SuperComputing 2001 国际会议的 Gordon Bell Award 奖项。

MPICH-G2 是基于 MPICH 开发的，相对于基于局域网的 MPICH，MPICH-G2 实现了 MPI 2.0 标准中的 Client/Server 连接方式，在服务器端提供了 MPI_Open_port, MPI_Close_port, and MPI_Comm_accept 函数，在客户端提供了 MPI_Comm_connect 函数。并且对 MPI 的集合（collective）操作进行了功能优化，使得 MPICH-G2 的集合操作能够针对运行时的网格计算环境拓扑结构进行识别和优化，提高集合操作的性能。目前的 MPICH-G2 中优化过的集合操作函数有：MPI_Barrier、MPI_Bcast、MPI_Gather、MPI_Scatter、MPI_Reduce。将来还会对其它九个集合操作函数进行优化[42]。

由于基于网格计算环境的并行计算程序会跨越不同的物理通信域进行数据交换，MPICH-G2 针对这些情况[45]，假设：

广域网的 TCP 通信性能 < 局域网的 TCP 通信性能（跨网段） < 跨机器的 TCP 性能（在一个网段内） < 专门的 MPI 实现的性能（如基于高性能 Myrinet 网络的 MPI 实现）

并可以采用 RSL 语言描述出 MPI 进程的不同分布，使得 MPI 集合操作函数可基于 RSL 语言描述的分布进行优化集合通信。如设 MPICH-G2 的机器配置文件中的内容为(第一项表示计算节点网络名称，第二项表示可运行的 MPI 进程数)：

```
"aci.cs.tsinghua.edu.cn" 10
"hpclab.cs.tsinghua.edu.cn" 10
```

则如果执行如下命令：mpirun -dumprsl -np 12 app1 123 456，则可能会产生如下输出：

```
+
( &(resourceManagerContact=" aci.cs.tsinghua.edu.cn")
  (count=10)
  (jobtype=mpi)
```

```
    (label="subjob 0")
    (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0))
    (arguments=" 123 456")
    (directory=/home/chenyu)
    (executable=/home/chenyu/myapp)
)
( &(resourceManagerContact="m hpclab.cs.tsinghua.edu.cn ")
    (count=2)
    (jobtype=mpi)
    (label="subjob 1")
    (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1))
    (arguments=" 123 456")
    (directory=/homes/users/duzh)
    (executable=/homes/users/duzh/myapp)
)
```

这实际上是一个任务请求的 RSL 描述，可以被 GRAM 解释和执行。当然 MPICH-G2 的运行离不开 Globus 环境，只有配置好 Globus 工具包，才可能安装 MPICH-G2。具体的安装过程可访问[3]进行进一步了解。

9.5 小结

商业的分布式计算技术可快速构建复杂的 client-server 应用程序，而网格技术提供了支持大规模、广域网、跨学科环境的高级网络服务，可协同各种资源完成复杂的各种应用。Commodity Grid 项目地目的是在这两者之间建立一个桥梁，充分发挥二者的优势。而 Java Commodity Grid Toolkit（CoG Kit）是实现这一目的的第一步尝试，随着采用 Web Service 新技术（SOAP、UDDI、WSDL 等），CoG Kit 的功能会更加强大，建立基于 Grid 的应用程序会更加方便和快捷。MPICH-G2 也在进一步发展，对 MPI 2.0 标准的支持也会越来越强。这里只是介绍了网格应用程序开发工具中的 CoGKits 和 MPICH-G2，而其它工具也层出不穷，有兴趣的读者可参考[1]。

思考题

1 CoG Kits 的特点是什么？
2 MPICH-G2 的特点是什么，针对网格计算环境做了何种优化？
3 试用 CoG Kits 编写简单的应用程序。
4 试用 MPICH-G2 编写简单的应用程序。
5 试分析 MyProxy 的实现和特点。
6 试分析 Condor-G 的实现和特点。

参考文献

[1] http://www.globus.org

[2] http://www.globus.org/ogsa/

[3] http://www.niu.edu/mpi/

[4] I. Foster, C. Kesselman, J. Nick, S. Tuecke; January, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. 2002.

[5] I. Foster, C. Kesselman, S. Tuecke.The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001.

[6] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data Management and Transfer in High-Performance Computational Grid Environments. Parallel Computing, 2001.

[7] I. Foster, C. Kesselman.The Globus Project: A Status Report. Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, pp. 4-18, 1998.

[8] I. Foster, C. Kesselman.Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[9] W. Allcock, J. Bester, J. Bresnahan, A. Chernevak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke,Data Management and Transfer in High Performance Computational Grid Environments.; Parallel Computing, 2002.

[10] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman;Grid Service Specification. Technical Report. February, 2002.

[11] W. Allcock, J. Bresnahan, I. Foster, L. Liming, J. Link, P. Plaszczac.GridFTP Update, Technical Report. January 2002.

[12] L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke.A Community Authorization Service for Group Collaboration. Submitted to IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2001.

[13] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman.Grid Information Services for Distributed Resource Sharing. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[14] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. IEEE Mass Storage Conference, 2001.

[15] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch. A National-Scale Authentication Infrastructure. IEEE Computer, 33(12):60-66, 2000.

[16] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. Journal of Network and Computer Applications, 23:187-200, 2001.

[17] I. Foster, A. Roy, V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. 8th International Workshop on Quality of Service, 2000.

[18] V. Sander, I. Foster, A. Roy, L. Winkler. A Differentiated Services Implementation for High-Performance TCP Flows. TERENA Networking Conference, 2000.

[19] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. ACM 2000 Java Grande Conference, 2000.

[20] I. Foster, A. Roy, V. Sander, L. Winkler. End-to-End Quality of Service for High-End Applications. Technical Report.

[21] G. Hoo, W. Johnston, I. Foster, A. Roy. QoS as Middleware: Bandwidth Reservation System Design. Proc.of the 8th IEEE Symposium on High Performance Distributed Computing, pp. 345-346, 1999.

[22] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. Intl Workshop on Quality of Service, 1999.

[23] G. von Laszewski, I. Foster. Grid Infrastructure to Support Science Portals for Large Scale Instruments. Proc. of the Workshop Distributed Computing on the Web (DCW), 1999.

[24] J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke.GASS: A Data Movement and Access Service for Wide Area Computing Systems. Sixth Workshop on I/O in Parallel and Distributed Systems, May 5, 1999.

[25] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids. Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.

[26] I. Foster, N. T. Karonis, C. Kesselman, S. Tuecke. Managing Security in High-Performance Distributed Computing. Cluster Computing, 1(1):95-107, 1998.

[27] C. Lee, C. Kesselman, J. Stepanek, R. Lindell, S. Hwang, B. Scott Michel, J. Bannister, I. Foster, A. Roy. The Quality of Service Component for the Globus Metacomputing System. Proc. IWQoS '98, pp. 140-142, 1998.

[28] P. Stelling, I. Foster, C. Kesselman, C.Lee, G. von Laszewski. A Fault Detection Service for Wide Area Distributed Computations. Proc. 7th IEEE Symp. on High Performance Distributed Computing, pp. 268-278, 1998.

[29] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. A Resource Management Architecture for Metacomputing Systems. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, po. 62-82, 1998.

[30] I. Foster, D. Kohr, R. Krishnaiyer, J. Mogill. Remote I/O: Fast Access to Distant Storage. Proc. Workshop on I/O in Parallel and Distributed Systems (IOPADS), pp. 14-25, 1997.

[31] I. Foster, N. Karonis, C. Kesselman, G. Koenig, S. Tuecke. A Secure Communications Infrastructure for High-Performance Distributed Computing. 6th IEEE Symp. on High-Performance Distributed Computing, pp. 125-136, 1997.

[32] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. Proc. 6th IEEE Symp. on High-Performance Distributed Computing, pp. 365-375, 1997.

[33] I. Foster, J. Geisler, C. Kesselman, S. Tuecke. Managing Multiple Communication Methods in High-Performance Networked Computing Systems. J. Parallel and Distributed Computing, 40:35-48, 1997.

[34] I. Foster, C. Kesselman, S. Tuecke, The Nexus Approach to Integrating Multithreading

and Communication. J. Journal of Parallel and Distributed Computing, 37:70--82, 1996.

[35] I.Foster, C. Kesselman, S. Tuecke. The Nexus Task-Parallel Runtime System.. Proc. 1st Int'l Workshop on Parallel Processing, pp. 457-462, 1994.

[36] Nexus: Runtime Support for Task-Parallel Programming Languages. Technical Report: Mathematics and Computer Science Division, Argonne National Laboratory, 1994.

[37] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[38] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, B. Tierney. File and Object Replication in Data Grids. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[39] J. Novotny, S. Tuecke, V. An Online Credential Repository for the Grid: MyProxy. Welch. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[40] S. Vazhkudai, S. Tuecke, I. Foster. Replica Selection in the Globus Data Grid. Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001), pp. 106-113, IEEE Computer Society Press, May 2001.

[41] A. Roy, I. Foster, W. Gropp, N. Karonis, V. Sander, and B. Toonen. MPICH-GQ: Quality-of-Service for Message Passing Programs. Proceedings of the IEEE/ACM SC2000 Conference, November 4-10, 2000.

[42] B. de Supinski, N. Karonis. Accurately Measuring MPI Broadcasts in a Computational Grid. Proc. 8th IEEE Symp. on High Performance Distributed Computing, pp. 29-37, August 1999.

[43] I. Foster, N. Karonis.A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems. Proc. 1998 SC Conference, November, 1998.

[44] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, S. Tuecke. Wide-Area Implementation of the Message Passing Interface. Parallel Computing, 24(12):1735-1749, 1998.

[45] J. Bresnahan, I. Foster, J. Insley, S. Tuecke, B. Toonen. Communication Services for Advanced Network Applications. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications 1999, Volume IV, pp1861-1867.

[46] I. Foster, A. Iamnitchi, and M. Ripeanu. Locating Data in (Small-World?) Peer-to-Peer Scientific Collaborations. Workshop on Peer-to-Peer Systems, Cambridge, Massachusetts, March, 2002.

[47] I. Foster, A. Iamnitchi, and K. Ranganathan. Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities. Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop, Berlin, May, 2002.

[48] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel and B. Toonen. Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus. Proceedings of SC 2001, November 10-16, 2001.

[49] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Leigh,

A. Sim, A. Shoshani, B. Drach, D. Williams. High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies. SC 2001, November 2001.

[50] I. Foster, J. Insley, G. vonLaszewski, C. Kesselman, M. Thiebaux. Distance Visualization: Data Exploration on the Grid. IEEE Computer Magazine, 32 (12):36-43, 1999).

[51] K. Czajkowski, I. Foster, and C. Kesselman. Resource Co-Allocation in Computational Grids. Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8), pp. 219-228, 1999.

[52] G. Mahinthakumar, F.M. Hoffman, W.W. Hargrove, N. Karonis. Multivariate Geographic Clustering in a Metacomputing Environment Using Globus. Proc. SuperComputing 99, November 1999.

[53] S. Brunett, K. Czajkowski, S. Fitzgerald, I. Foster, A. Johnson, C. Kesselman, J. Leigh, S. Tuecke. Application Experiences with the Globus Toolkit. To appear in 7th IEEE Symp. on High Performance Distributed Computing, July 1998.

[54] J. Nieplocha and R. Harrison, Shared memory NUMA programming on the I-WAY, in Proc.5th IEEE Symp. on High Performance Distributed Computing, IEEE Computer Society Press,1996, pp. 432-441.

[55] M. Norman et al., Galaxies collide on the I-WAY: An example of heterogeneous wide-area collaborative supercomputing, International Journal of Supercomputer Applications, 10 (1996), pp. 131-140.

[56] I. Foster, J. Geisler, and S. Tuecke, MPI on the I-WAY: A wide-area, multimethod implementation of the Message Passing Interface, in Proceedings of the 1996 MPI Developers Conference, IEEE Computer Society Press, 1996, pp. 10-17.

[57] “S 2.1 GRIS Specification Document:Creating New Information Providers”，Technical Report. May 2002.

第三篇 网格应用

作者简介

刘鹏

清华大学计算机系高性能所博士研究生，师从李三立院士，主要研究方向为分布/并行处理体系结构。

1992 年　本科毕业于南京解放军通信工程学院。
1996 年　获解放军通信工程学院硕士学位，导师为谢希仁教授。
　　此后四年任解放军理工大学（原通信工程学院）计算机系讲师。
2000 年　考入清华大学攻读博士学位。

刘鹏共编著出版专业书籍 12 本(其中 6 本为第一作者),参与完成了多个重大科研项目，在核心期刊和国际会议发表论文 10 多篇。曾率清华团队赢得 2002 PennySort 世界排序比赛冠军，他们是世界排序比赛开展 17 年以来第一个赢得此项世界纪录的美、日以外团队。2002 年，刘鹏被评为清华大学“十学术新秀”，还率队夺得了该年度清华大学学生课外科技作品 “挑战杯”比赛第一名。《中国青年》、《解放军报》等媒体报道了刘鹏事迹。

Email:　pengliu@ieee.org　　pengliu@acm.org
个人主页: http://hpclab.cs.tsinghua.edu.cn/~liupeng/

第10章 分布式超级计算

分布式超级计算(Distributed Supercomputing)是指将分布在不同地点的超级计算机用高速网络连接起来，并用网格中间件软件“粘合”起来，形成比单台超级计算机强大得多的计算平台。事实上，网格的最初设计目标就是要满足更大规模的计算需求，所以 Globus 到目前为止，还是主要针对这些应用设计的。本章10.1节介绍分布式超级计算的需求背景。读者肯定对于怎样将大型的计算任务分解成较小的块，以便在多台超级计算机上运行非常感兴趣，10.2会就这个问题进行讨论。另外，在建设针对分布式超级计算的网格时，也会有一些突出的问题需要考虑，这将在10.3节中介绍。10.4节所介绍的 SF Express，是一个分布式超级计算的典型应用，它将大型军事仿真任务分解到分布式环境中运行，从而在规模上创下了该领域的世界纪录。10.5节介绍了另一个应用，它利用网格的计算能力模拟了黑洞的碰撞。该应用非常有特色，而支撑这个应用的平台更有特色，它就是 Cactus，也将在该节中介绍。

10.1 背景

人类对未知世界的探索是永无止境的，对计算能力的需求也是这样，会随着计算科技的迅猛发展而不断膨胀。1946 年，当第一台电子计算机 ENIAC 问世时，有科学家对这台每秒运算 5000 次的新机器发出感慨：这种机器的运算速度实在惊人，全世界只需要有两台，就可以满足所有计算之需。几十年过去了，虽然现在最快计算机的运算速度已经接近原来的 100 亿倍，但科学家们仍然对着越来越大的计算问题望洋兴叹。

例如，虽然爱因斯坦早在 1916 年就提出了相对论，它的重力定律能够描述宇宙中各种物质的相互作用，小到中子星和黑洞，大至星系，但在过去的 80 多年里，这个定律却很难得到具体应用[1]。究其原因，是因为它实在太复杂了，需要求解包含上千个成份的十组非线性偏微分方程。模拟最简单的中子星或黑洞，就需要有万亿次的计算能力，更不要说稍微复杂一些的天体系统了。类似的超大规模计算问题还很多，如精确的长期天气预报、新药筛选、核爆炸模拟、飞行器数字模拟、作战模拟、高分子材料分析，等等。

对于这些超出单台计算机能力的计算任务，网格分布式超级计算无疑是一个有效的解决方案。建立网格的最初目标就是要把分散在各地的大型计算机用网络和中间件联结起来，构成一个虚拟的超级大型计算机，获得前所未有的处理能力，从而有可能解决更大型的问题。

除此之外，由于网格集成了不同类型的大型计算机，形成了一个异构的计算环境，这对于解决不同类型的问题带来了很大的好处。例如，有些问题适合在价格低廉的集群计算机上运行；而有些问题却只有在矢量计算机上才会得到更为高效的解决。

还有，网格为交叉学科研究提供了一个极好的平台。一些大型的综合性问题，只有在来自不同学科的科技工作者的共同参与下才能得到很好的解决，例如，研究宇宙飞船的空气动力特性，就需要有物理学家、天文学家、计算机专家等的共同参与。网格这台特殊的“超级计算机”，摆脱了空间的限制，把它的触须（网络终端）延伸到了众多的科技工作者面前，以中间件融合他们的工作，集众人的智慧攻克前人无法企及的问题。

10.2 应用程序在网格上的分解

大型并行计算问题在网格上可以用三种方式分解[4]：

第一种是按步骤分解。某些应用由一系列处理构成，我们可以将这些处理分布到网格上的不同并行计算机上，就像工厂的流水线一样。当然，我们造这种流水线的目的肯定不是生产一个“产品”，而是希望它能源源不断地生产相同类型的“产品”。例如，遥感卫星不断地拍下照片，数据送到超级计算中心 A 进行预处理，然后送到 B 进行分析，最后送到 C 进行存储处理。

第二种是按功能分解。某些计算任务能够分解成相互基本独立的不同功能单元，例如，全球气候模型可以分解为大气模型和海洋模型，可以分别在不同的大型计算机上处理，相互之间的数据传输不是很频繁。

第三种是按数据分解。有许多应用的数据量极大，需要把不同数据块分配到不同的计算机上处理。如果数据块之间的耦合度很小，就能充分体现网格并行计算的优势，例如，要模拟由 100 万个战斗单位构成的战争态势，虽然理论上讲任何两个战斗单位 a 和 b 的行为是相关的，但相比之下，a 的行为与它所在的战斗团队 A 的其他成员的行为相关性要比与 b 的相关性大得多，同样，b 与其团队 B 的其他成员的相关性比与 a 的相关性要大得多。也就是说，可以分别用不同的并行机模拟不同的战斗团队，使得不同并行机之间的耦合度较小。

但是，并不是所有的大规模应用都能够这么轻松地被分解。有些应用的确需要经过多个步骤的处理，但它们却只需要处理一次，不具备可重复性，使得按流水线分解失去意义；有些应用的确需要有多个功能部件，但功能之间的关联很大，N 台计算机还不如在一台计算机运行得快；有些应用的确需要并行处理大量的数据，但数据之间的全局相关性非常大，如果把它们分布到不同并行机上，频繁通信将使运行效率异常低下。这几类问题暂时还没有很好的解决方案。

不过，随着技术的发展，这些问题还是有希望得到解决的。一方面，在不远的将来，网络带宽的瓶颈问题很有可能不复存在。在过去的 20 年，光纤通信传输速率的成长几乎是每隔 10 年就增加上百倍。尤其到了近期，DWDM（密集波分复用）技术使我们开始憧憬无限的网络带宽。如今，成熟的 DWDM 商用产品在单条光纤上的传输能力已经超过 Tbit/s。随着网格带宽的增大，网格的运行效率会越来越接近单台超级计算机的效率；另一方面，更强大的超级计算机、算法研究上的进展和更成熟的网格平台，也会给这类问题的解决带来曙光。

10.3 分布式超级计算的核心技术

网格是一个异构的超级计算机集合，它相对单台的超级计算机的复杂度，就像广域网相对于局域网一样。首先，网格环境由许多不同体系结构、不同厂商、不同操作系统、不同编程环境的超级计算机构成，使用和管理自然有诸多不便；其次，由于网格基于广域网，相对于每个超级计算机节点的强大功能而言，连接这些节点的公共通信设施就显得太单薄了(带宽小、延迟大)，就像一些大城市之间只有羊肠小道连接。因此，从整体上看来，虽然已经朝向正确的方向迈出了第一步，网格这种“有机体”在目前还不是一个健康的躯体；再次，网格环境不是一种稳定的环境：由于网格资源由众多参与者共享，所以存在争用资源的问题；网格的通信建立在 Internet 之上，所以存在时断时续的情况；网格是一个复杂的异构系统，所以存在频繁出错的问题。

就像在复杂、不可靠的广域网上需要设计 TCP/IP 和 HTTP 一样，网格的实用化依赖于对中间件和应用的深入研究。

10.3.1 适应性算法

网络最重要的两个属性是带宽和延迟。目前，干线网的带宽以 Gb/s 为单位。但这不是超级计算机之间的真正带宽，因为干线网是由众多计算机共享使用的，除非以极大的代价租用专门的线路。长远地看，带宽不会是一个严重问题，干线带宽达到 Tb/s 量级指日可待，相比之下，延迟更有可能成为羁绊网格发展的重要因素。网络传输速度受限于光速，距离越长延迟越大，例如，横跨美洲大陆大约需要 20ms[4]。传输所经的大量中继站点的转发过程，更为速度的提升制造了不小的障碍。相比并行计算机内部以微秒为单位的通信延迟，这种广域延迟的确让研究人员感到头疼。

虽然没有彻底消除障碍的方法，在一定的程度上削弱问题的影响还是有可能的，其基本方法是仔细分析应用和网格的特性，用一些适应性算法，减少对网络带宽的依赖，隐藏部分网络延迟。例如，可以巧妙协调应用，使计算和通信过程重叠，亦即在计算的同时进行通信，隐藏网络时延；可以增大计算的粒度，减少模块之间的耦合程度，降低通信开销，甚至可以用冗余的计算减少通信开销；可以将通信传输集结成组传输，增加一次通信数据的长度，提高传输效率；可以通过数据压缩算法，减少数据传输量；可以在计算精度和通信能力之间做出平衡，只传输关键数据，免除对大量外围数据的传输需求；可以根据应用程序和网络的特性，调整某些协议的参数（例如，改变 TCP/IP 的时间窗口大小）或制定性能更优的特殊协议。

适应性算法与具体的应用相关性极大，本书将在后面的实例中详细描述。

10.3.2 资源调度策略

网格的能力，最终还是要由应用程序的运行性能来体现。为了获得好的运行性能，需要仔细分析应用的特性、可用的高性能计算机类型、处理器数量和存贮能力、网络带宽和延迟、输入/输出数据的位置，等等。然后在应用和网格资源之间做出合理匹配，并在运行过程中不断微调和优化。

在有众多资源和众多参与者及协调者的情况下，资源调度是一个极其繁琐复杂的问题。调度的基本原则是保障资源为完成尽可能的任务服务。不能出现死锁是起码的要求，还要考虑资源在时间和空间上的合理搭配，以期达到更好的效果。更为复杂的调度算法，还要考虑应用的优先级、资源的预留、自动调度(减少人工参与工作量)等问题。

下面的例子说明了资源调度策略的重要性。

假设有三个应用 A、B 和 C，它们各自在不同的运行步骤对共享资源(a)、(b)、(c)、(d)、(e)和(f)的需求如图 10-1所示。

步骤	应用 A	应用 B	应用 C
第 1 步	(a) (b)	(b) (c)	(c)
第 2 步	(c)	(d) (e)	(a) (b) (e)
第 3 步	(d) (e)		(f)

图 10-1 三个应用程序在不同步骤对资源的需求

显然，如果没有经过优化的资源调度策略，资源满足谁的条件就给谁，会带来效率低下的问题。例如，首先把(a)和(b)分配给应用 A，然后试图把(b)和(c)分配应用 B，由于(b)已经由 A 占用，故 B 只好等待。此时，应用 C 也获得资源(c)。第 1 步完成之后，应用 B 得到(b)和(c)，而应用 A 和 C 都因需要 B 所占有的成份而被迫等待。第 2 步完成后，A 和 B 进入

下一步，而 C 只好等待……完整的步骤如图 10-2所示，一共需要 6 步才能完成。(注：这里假定只有当资源完全满足应用程序某一步的需求时才进行分配，使得每 1 步都有应用能够继续下去，不会出现两个应用互相占有对方的资源的情况，从而消除了形成死锁的条件。)

步骤	应用 A	应用 B	应用 C
第 1 步	(a) (b)	等待	(c)
第 2 步	等待	(b) (c)	等待
第 3 步	(c)	(d) (e)	等待
第 4 步	(d) (e)		等待
第 5 步			(a) (b) (e)
第 6 步			(f)

图 10-2 无调度策略时应用程序所需要经历的步骤

现在来看在一定调度策略指导下的调度方案，如图 10-3所示。可见，经过全局考虑，完成 3 个应用的总步骤缩减为 4 步，系统的运行效率大大提高。

步骤	应用 A	应用 B	应用 C
第 1 步	(a) (b)	等待	(c)
第 2 步	(c)	等待	(a) (b) (e)
第 3 步	(d) (e)	(b) (c)	(f)
第 4 步		(d) (e)	

图 10-3 在某种调度策略下应用程序所经历的步骤

10.3.3 容错

在单台并行计算机中，容错是一个重要的问题，因为它由多个处理部件构成，增加了出错的可能性。而许多计算任务又需要绵延数日的运行才能得到最终结果，如果没有容错措施，任何中途出现的故障就会使前面的计算前功尽弃。

网格是由若干台单台并行计算机用不可靠的 Internet 连接而成，出错的概率比单台计算机要大得多，如果没有一定的容错措施，就没有任何实用价值了。

在单台并行计算机中，检查点(Checkpointing)的使用是对付计算故障的重要手段。当计算每完成一个中间步骤或运行一定时间，就将机器的运行状态保存在外存中，这就形成了一个检查点。如果在下一个检查点到来之前出现了故障，就从外存中读出上一个检查点的数据恢复机器运行的现场，从而避免了从头开始。检查点既可以在系统软件级实现，也可以由应用程序自己实现。前者适用面广，但实现非常困难；后者增加了应用程序的编写复杂度，但实现相对容易些。

在网格环境中，理论上讲，可以通过全局的检查点实现网格的容错，但其实现难度却与单机检查点不在同一个数量级上。整个网格内需要分散实现检查点，却很难保证它们之间的一致性，甚至连一个统一的时钟都难以提供。就算所有结点能够同时做检查点，由于网络传输需要时间，如果在 A 机发出消息之后及 B 机收到消息之前保存了一个检查点状态，则这条消息将被检查点遗漏，造成信息完整性的破坏。

更为可行的方法似乎存在于网格协议和应用程序本身，而不在费力低效的检查点上。就网格协议而言，不仅应该随时能够检出当前所有可用的资源，还应该即时反映出资源失效的情况。例如，Globus 使用了监听心跳的措施来跟踪并行计算机失效的情况。对于任何协议，

容错措施并非可有可无，对于网格这种复杂的异构和广域环境更是如此。从这个意义上讲，TCP/IP 协议应该算得上成功的典范。Internet 之所以能够实用化并流行，仰仗 TCP/IP 用 CRC 检错，用重发消错，用迂回线路避错，在不可靠线路上提供了可靠的服务。

在网格环境中，应用程序本身也应该具有容错功能。如果一个算法容不得半点错误，那它肯定不是个好算法。一个好算法应该假设每个计算结点和网格传输都有可能出错，它应该能够检验出这些错误来，并通过冗余计算等手段寻找正确的方案。P2P 计算在容错上所取得的成功经验对网格无疑是一个启示。不管搜寻外星生命的 SETI@home 程序(http://setiathome.ssl.berkeley.edu/)，还是用以攻克癌症的 Intel 博爱 P2P 程序(http://www.intel.com/cure/)，都不会指望每一个下载程序的用户最终都将提交正确的结果。所以，它们得有手段检测结果的正确性，并把出错或没有答复的数据片段交由其他用户进行冗余计算——虽然简单，却非常实用。应用程序就应该充分利用网格的冗余性，检错并纠错，从而达到容错的目的。

10.4 SF Express——大规模军事仿真

本节和下一节给出最有影响的两个网格分布式超级计算应用，它们体现了网格计算超越传统计算的神奇威力。本节介绍的应用称为综合军力表示(Synthetic Forces Express)，简称 SF Express[7]，它基于网格模拟了多达 10 万个战斗单位的作战场面，创造了该领域的世界纪录。

10.4.1 背景

SF Express 项目始于 1996 年，是由美国国防部下属的国防先进研究计划处 DARPA(Defense Advanced Research Projects Agency)资助、由加利福尼亚理工学院负责完成的，其目标是模拟尽可能多的战斗单位。DARPA 一直对大规模的军事模拟非常感兴趣，因为它对于军事指挥、训练、演习和试验都有先验指导意义。

1996 年 11 月，SF Express 使用拥有 1024 个处理器的 Intel Paragon 并行机，模拟了 10,000 个战斗单位。1997 年，SF Express 扩展到横跨 7 个时区的 6 台超级计算机，共使用了 1,094 个处理器，模拟了 50,000 年战斗单位。这时，SF Express 已经初步具备网格特征，但它采用手工配置的方法管理资源，无法适应网格的变化，不能动态容错，在演示过程中就暴露出许多问题。

之后，SF Express 加强了与 Globus 的融合。在场景分发、资源配置、资源管理、信息服务、日志服务、监视和容错等方面都利用了 Globus 的动态管理功能。1998 年 3 月 16 日，SF Express 集合 13 台并行计算机之力，使用了 1,386 个处理器，终于成功模拟了 100,298 个战斗实体，实现了历史上最大规模的战争模拟。此时的 SF Express 已经能够适应网格的动态变化，能够自动选择资源，自动提交可执行代码和运行数据，自动调整运行状态，自动屏蔽网格中的出错情况。

从 SF Express 走过的历程看，由单台并行计算机转向网格环境，将能够解决的问题规模扩大了一个数量级。而它与 Globus 的融合程度，决定了它对网格环境的适应能力。这对后来的大规模应用无疑是有借鉴意义的。

10.4.2 任务的分解

SF Express 的模拟基于分布式交互模拟平台 DIS(Distributed Interactive Simulations)。DIS 是一个网络化的模拟环境，它支持对复杂地形、高保真独立建模实体(如坦克、舰船、导弹等)的模拟，还支持人在回路(真实的人作为模拟的一部分加入到虚拟的环境中)，模拟的战争场面可以用可视化工作站展现出来。

DIS 的模拟实体之间需要频繁沟通以反映战争的进程，既要向外发送自身位置和行为的变化，又要接收友军和敌情的变化信息，如军队的调动、导弹飞行、爆炸等。DIS 通过在网络上传输标准协议数据单元 PDU(Protocol Data Units)来完成通信。如果没有特殊的处理方法，随着场景复杂度和实体数量的增加，实体之间的通信量会迅速增加，很快，网络就会被 PDU 塞满，而并行计算机的处理能力将被耗尽。

有幸的是，实体之间的通信是有局部性的。例如，一辆坦克只需要掌握它的同伴和面对的敌军的运行状态，而不需要了解远在太平洋的航空母舰的调度情况。这样，就能把所有实体分门别类地放入到不同的“袋子”里，每个“袋子”内部是强相关的，而“袋子”之间是弱相关的，SF Express 设计了如图 10-4所示的体系结构[5]，把一些“袋子”放在某台超级计算机上运行，而另一些“袋子”放到另外的超级计算机上运行，从而在扩大了系统规模的同时避免了通信量呈指数增长。

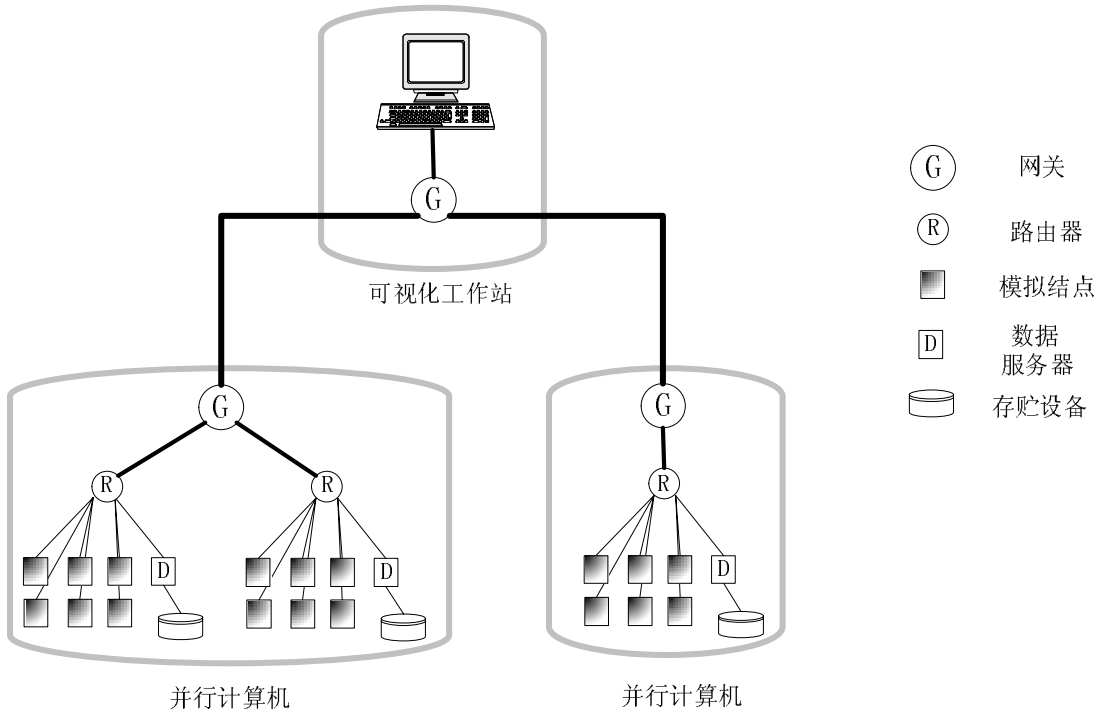


图 10-4 SF Express 体系结构

图 10-4中功能结点的作用解释如下[6]：

1 模拟器

并行计算机内部的绝大多数处理器都用作模拟器，运行标准的 ModSAF 模拟程序。ModSAF 是一个流行的 DIS 应用程序，它提供真实感很强的军事模拟。每个模拟器大概能

够模拟 50-150 个战斗实体，具体数量取决于军事行动的复杂性及处理器的能力。

2　路由器

路由器为一组模拟器传递消息。每个模拟器不断向路由器报告所模拟的实体的状态和活动，又从路由器获得其他模拟器上实体的状态和活动。在具体实现时，可能为一组模拟器指派 3 个路由器，一个负责直接与下属的模拟器打交到，一个专门计算本组模拟器的感知范围，而另一个专门报告所有实体的运动状态。这样，每组路由器能够很容易地管理 1000 到 2000 个战斗实体。

3　网关

网关是一种特殊的路由器，它负责本地路由器与其他并行计算机之间的消息传递。

4　数据服务器

专门向模拟器提供数据存贮和访问服务，它包含大容量存贮设备和缓冲设备，保存地形图等数据。

需要指出的是，不管是模拟器、路由器，还是网关、数据服务器，都是用并行计算机的处理器实现的，只不过各自数量和所运行的程序不同罢了。并行计算机所能模拟的实体数量取决于它的处理能力，举例来说，Intel Paragon 机能够模拟大约 18,000 个战斗实体。

SF Express 利用上述分层次的体系结构，非常有利于削减网格中的通信量。打个比方，由于政府机构划分为部、厅、局、处、科几个层次且有明确分工，这样，公文就不需要发给所有的单位，只需要发给同级或上下级的相关单位，提高了办事效率。同样，在 SF Express 中，不同功能结点分配了不同的任务：

“基层单位”是模拟器，它们周期性地计算所模拟的每个实体的感知范围，并将所有范围合并，得到该模拟器的整体感知范围(如图 10-5所示)，并将这个范围及实体的活动情况报告路由器结点。同时，它们也从路由器接受处于其感知范围内的实体的活动情况的报告，并根据地形和活动情况计算每个实体的下一步对策。

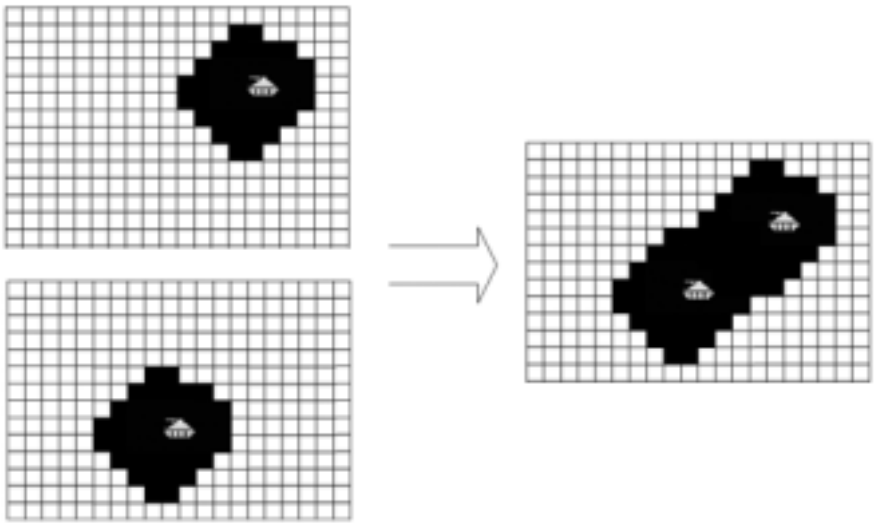


图 10-5 两辆坦克感知范围的合并

“中层单位”是路由器结点，它们从每个模拟器接受感知范围信息并保存到一个大的缓存中，然后合并出所辖全部模拟器的整体感知范围并上报网关结点。当路由器收到经网关转发的、来自其他路由器的活动报告，便会根据活动的发生地点是否落在某个模拟器的感知

范围之中，决定是否将该报告转发给该模拟器。

“高层单位”是网关结点，它是网格中并行计算机之间的信息中转站和过滤器。它向其他并行计算机的网关发布本机的集合感知范围，并周期性接收并保存其他计算机的感知范围。如果本地一个实体的活动位置正好落在某台并行计算机的感知范围之中，才将它的活动报告转发给该计算机。同理，网关如果收到其他计算机发来的实体活动报告，则该实体的位置一定处于下属某个网关的感知范围之中，于是网关就将该报告转发给该网关。

就这样，经过三层信息过滤，网格内部基本上不存在无用信息传递，即使是模拟了 10 万个实体也没有使系统瘫痪。即便如此，读者可能还是会觉得这个系统还是很复杂，因为网关、路由器、模拟器之间存在许多信息交互，单是通信格式定义起来就很麻烦。幸运的是，这是个基于 DIS 的系统，所有的活动情况报告都是用标准的 PDU 格式来传递的，因而大大简化了信息交互过程。就像邮政系统，虽然信件的内容千变万化，但信封的制式是统一的，在邮寄过程中所经历的层层转递，只是为了让信件发到正确的收信人手中，并没有增加信件本身的复杂性。

10.4.3 与 Globus 的融合

有了上一节的基础，实现 SF Express 好像已经水到渠成，为何还需要 Globus 的支持呢？那么，让我们先设想一下，如果没有 Globus，SF Express 的实现将遇到什么问题：

SF Express 项目纳入了横跨 7 个时区的 13 台超级计算机，如何才能使它们同时空闲下来以便运行 SF Express？唯一办法是逐一与这些计算机的管理人员联系，定下时间，预留机器。

如何把 SF Express 程序代码及初始数据传送到每台并行计算机上并启动之？看来只好逐一登录，用手工完成。

如果在程序运行过程中，有并行计算机出现异常情况(比如硬件故障或停电)，怎么办？这时 SF Express 只好停下来，找个机会重新开始。

如果连接某台并行机的网络出现拥塞或根本不通怎么办？这种情况也很常见，例如 SF Express 在 Supercomputing 97 国际会议上演示 50,000 个实体的模拟时，连接德州的 HP Exemplar 超级计算机的网络就出现了不畅通的情况[5]。

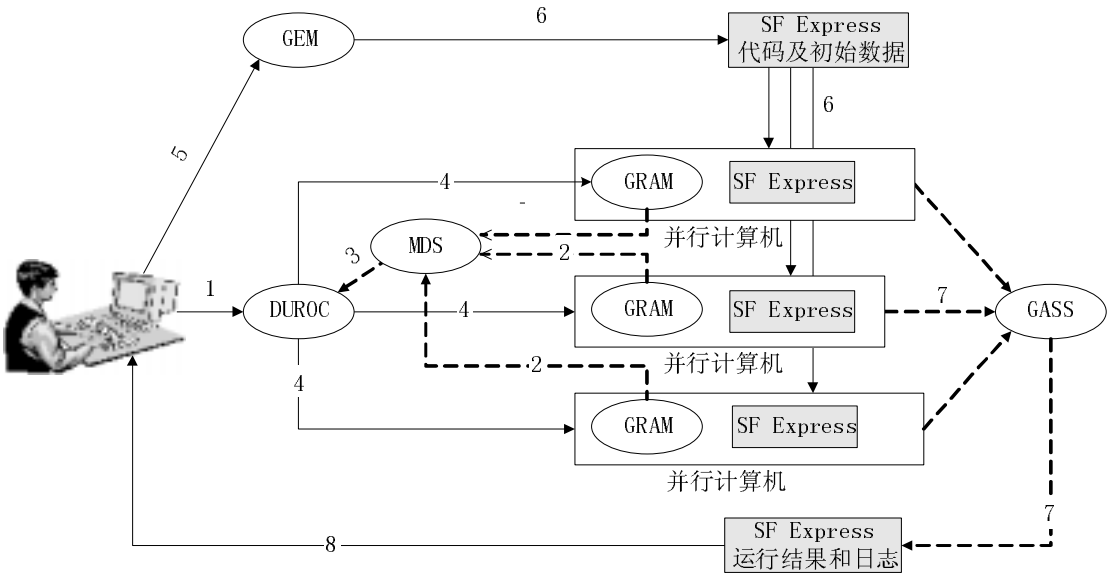
如果加入网格的并行计算机因为有更重要的任务，要求提前退出 SF Express 的执行怎么办？只好希望这种事情最好不要发生，否则将前功尽弃。

如何获得 SF Express 的运行结果和日志数据？如果没有另外的程序支持，这些结果和日志将分别保存在不同的并行计算机上，需要用手工把它们集中起来。至于动态分析和调整所有机器的运行状态，暂时是不敢奢望的事。

还有，其实超级计算机不太可能完全为某个应用空出来，它还会运行其他的任务。也就是说，它的负载会时轻时重，此时 SF Express 在这些结点上的运行就会出现时快时慢的情况，从而影响全局的性能，使仿真结果偏离实际(试想某支部队在某段时间突然没有动作的情况)，这时该怎么办？好像没有办法，因为 SF Express 在这些并行机上的任务划分是固定的，无法适应环境的变化。

从上述的描述可以看出，离开 Globus 的 SF Express 虽然也具备完成仿真任务的机会，但它缺少适应变化的“弹性”，这样，众多不确定因素的出现将使它变得捉襟见肘。幸运的是，Globus 是一个富有“弹性”的网格中间件集合，SF Express 通过与它的结合避免了上述问题。

SF Express与Globus的结合情况如图 10-6所示，图中椭圆部分是Globus的功能块。



DUROC: Dynamically-Updated Request Online Coallocator 动态更新请求在线协同分配器
GRAM: Globus Resource Allocation Manager (Globus 资源分配管理器)
MDS: Metacomputing Directory Service 元计算目录服务)
GEM: Globus Execution Management (Globus 执行管理)
GASS: Global Access to Secondary Storage 全局访问二级存贮器)

1	向DUROC发出资源申请	5	要求GEM传送文件
2	GRAM向MDS动态报道资源状况	6	文件传送到并行计算机上
3	MDS接受DUROC的查询	7	通过GASS保存结果
4	DUROC向GRAM申请资源	8	访问结果数据

图 10-6 SF Express 与 Globus 的结合

在Globus环境中，每个并行计算机上安装了资源分配管理器GRAM(Globus Resource Allocation Manager)。GRAM负责管理本地资源，并把资源的动态情况反映给元计算目录服务MDS(Metacomputing Directory Service)。这样，网格中资源的动态变化情况，可以及时反映到MDS上。由于涉及资源协同分配，Globus还专门提供了动态更新请求在线协同分配器DUROC(Dynamically-Updated Request Online Coallocator)，它负责与所有的GRAM打交到。

申请到足够的资源之后，就可以利用Globus执行管理GEM(Globus Execution Management)模块把SF Express可执行代码和初始数据自动传送到每一台并行计算机上，并为它们设置不同的初始参数。

由于MDS是动态更新的，就使得应用程序在运行过程中，可以根据资源的变化自动调整运行状态，如果个别并行计算机出现性能下降的情况，仿真程序可以将一些运行任务转移到其他计算机上，从而保障了全局的正常运行。

仿真程序的输出数据及运行过程中产生的日志，利用了Globus提供的全局访问二级存贮器GASS(Global Access to Secondary Storage)功能，自动转存到指定的计算机上，非常方便实时监控、调节仿真的状态和用可视化工具将战争场面实时地展现出来（如图 10-7[25]所示）。

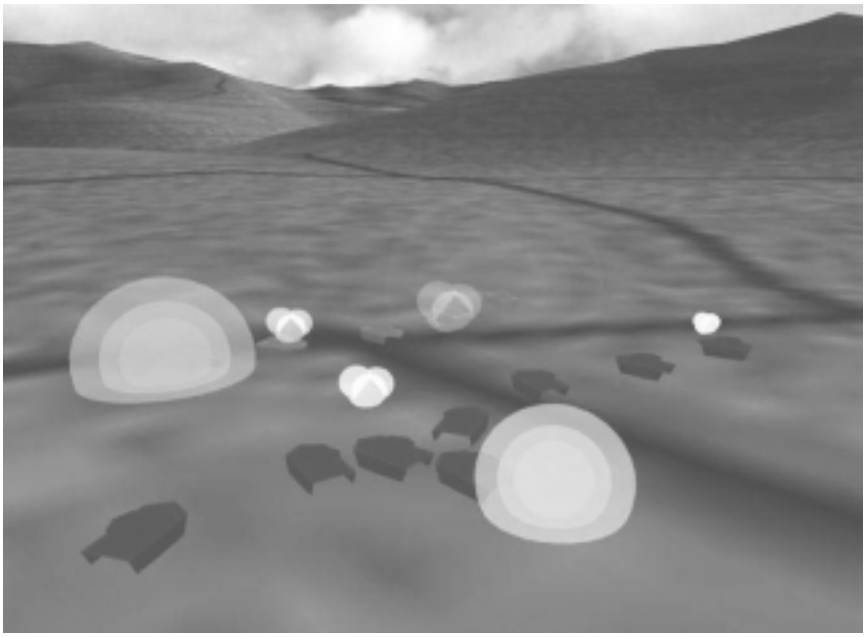


图 10-7　SF Express 模拟的战争场面

正是由于 Globus 的支持，SF Express 才具备了适应网格环境变化的“弹性”，大大增强了其实用性。

在对 SF Express 后续研究中，还试图与 Globus 的通信机制 Nexus 相结合[6]。Nexus 能够根据网络的通信服务质量自动作出调整。在 Nexus 的支持下，SF Express 不再自己编写通信程序代码，而全部改用 MPI 编程。不过，有一些应用（如[57]）表明，Nexus 并不是一个成功的通信平台，它太滞重（heavy）了，这好像让人明白了：为什么后来 SF Express 与 Nexus 结合的事情不了了之，为什么 Globus 在新版本中不再采用 Nexus……

10.5 分布式异构计算环境 Cactus 及其应用

Globus 是一个构成网格基础设施的平台，它的作用相当于网格操作系统，担负管理网格资源的重任，但它离具体应用的距离还比较远。如果直接基于 Globus 编写应用程序，就象直接使用底层 API 编写 Windows 程序一样麻烦，那么，为什么不在 Globus 和应用程序之间搭建一个桥梁呢？Cactus（仙人掌）[8]正是这样一个桥梁。有了 Cactus 的支持，编写应用程序几乎就不需要考虑网格的具体问题，甚至以前的应用程序可以不加修改就能使用。Cactus 项目由德国爱因斯坦研究所领衔开发，参加单位众多，已有多年积累，系统庞大而应用界面简单，因而受到普遍关注，成为最有影响的网格应用项目之一。其代表性应用就是10.1节所介绍的求解爱因斯坦方程并模拟出天体的运动规律，相关论文[2]在 2001 年超级计算会议(Supercomputing 2001)上获得了 Gordon Bell 奖[78]。

10.5.1 Cactus 的思想

Cactus 是一种集成的、通用的、开放源码的计算科学和工程问题解决环境。Cactus 允许

用户将原来在个人计算机上开发的程序(不管它们原来是用 C 语言、C++写的，还是用 Fortran、Java 语言写的)，转换成能够在虚拟的“网格计算机”上运行的并行程序；Cactus 提供了简单、抽象的 API 调用接口，屏蔽了系统的复杂特性，从而简化了用户界面，获得很强的可移置性；Cactus 能够在不同体系结构的机器上运行，如单处理器、集群计算机和其他体系结构的并行计算机；Cactus 具有许多先进特征，诸如先进的数字计算技术、自适应的网眼细化、并行 I/O、实时远程可视化、远程操控、Web 接入等[3]；Cactus 中还集成了许多现成的成熟应用模块，只要提供初始数据并修改参数，就可以得到计算结果，减少了重复开发的现象；Cactus 环境既可以用来数字相对论中，求解爱因斯坦方程，又可以用在其他领域，诸如天体物理学、化学工程、气象模拟、密码科学等。

Cactus 不仅在虚拟的网格超级计算机上建立了庞大的问题求解环境，还带来了一种崭新的“大科学”思想。例如，求解数字相对论这样的问题，不仅需要精通物理学、天文学、数学，还需要精通数字算法、网络工程、计算科学等，这些是任何个人或团体都无法单独完成的。Cactus 提供了一个平台，把不同学科的、不同地域的科技工作者凝结在一起。当一个物理学家正在研究爱因斯坦方程的一种新的表达式时，数字分析师可能正在探寻求解其中某种椭圆方程的更有效方法，而计算机科学家正在寻找更有效的并行 I/O 算法，而软件工程师正在开发一种新的并行编程环境，硬件生产商正在研制更高效的并行体系结构[1]……Cactus 使人类探索未知世界的能力达到了前所未有的高度。

Cactus 在其实现上，也具有非常独到之处。Cactus（仙人掌）这个名字本身就体现了它的实现之巧妙：正象仙人掌一样，Cactus 由一个主干（flesh）和若干生长在主干上的枝干（thorn）构成，枝干可以任意多个，直接插上主干就能使用，如图 10-8所示[9]。

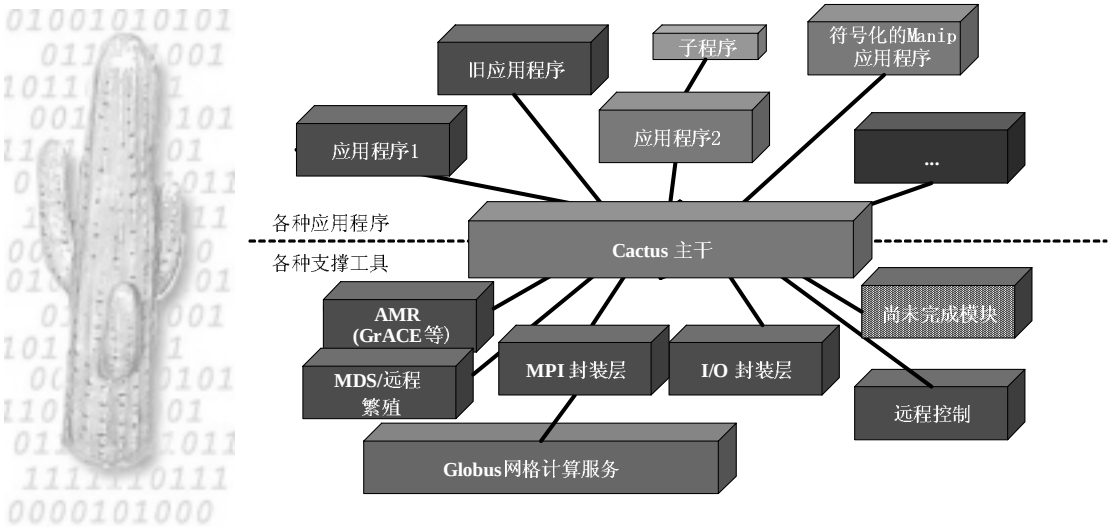


图 10-8　Cactus 的插件式思想

主干是 Cactus 的基础，用标准的 C 语言开发，提供一组 API，其目的是使枝干能够动态地结合在一起。为此，它需要能够控制由哪些枝干、用何种方式、怎样传递数据、何时执行哪些程序，等等。一个枝干是一个软件模块或一些子程序的集合，可以用任何常见的语言编写，如 Fortran 77、Fortran 90、C、C++、Java、Perl 及 Python 等。枝干分为两类，一类是计算网格的中间件（例如：实现 MPI、PVM、SHMEM 的驱动程序、并行 I/O 驱动程序，以及 Web 服务支持程序、网格监控工具等），另一类是各种现成的应用程序（例如黑洞模拟程序、椭圆方程求解程序等），它们插入到主干上即可工作。奇妙的是， Cactus 不是建立在 Globus 的基础上的，而是将 Globus 作为 Cactus 的一个枝干插在主干上，一旦 Cactus 插上了 Globus 枝干，使用 MPI 的应用程序立即就可以不加修改地运行于网格之上。

10.5.2 Cactus 体系结构

Cactus 的体系结构如图 10-9所示[2]。

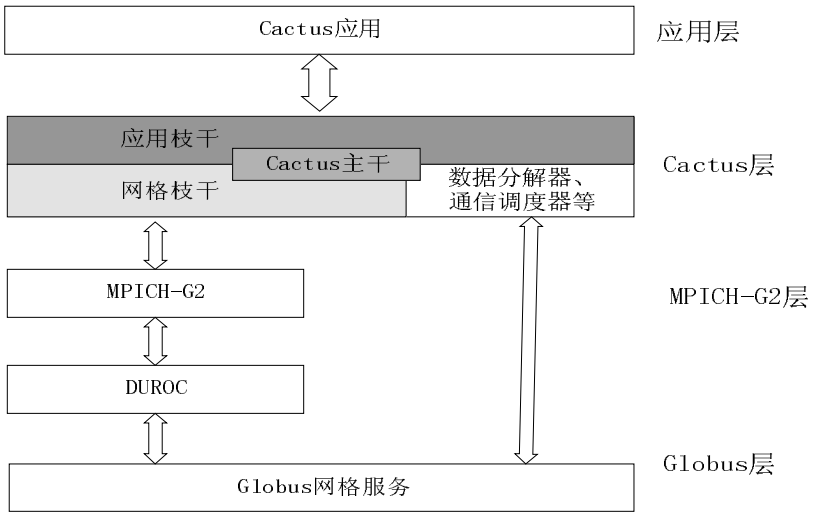


图 10-9 Cactus 体系结构

体系结构最上面的一层是应用层，也就是普通用户所能感知的层面：他们只需要指定初始状态、提交初始数据、指定求解问题的精度等，就可以得到计算的结果，而不需要考虑网格计算是在什么样的异构环境中、集结了多少动态的资源、用了什么样的优化算法完成的。

接下来就是 Cactus 层，它由 Cactus 主干及主干所融合的大量枝干插件构成。枝干大致可以分为两类：应用枝干和网格枝干。应用枝干用来完成具体的科学计算，比方说求解某种物理方程。应用枝干可用常用的编程语言书写，理论上讲，不用考虑网格的复杂情况，只需要针对抽象的虚拟机器编程。如果说应用枝干更接近于具体的应用的话，网格枝干则更接近于网格本身，它从诸多方面完善网格的功能，以更好地支持应用，如并行性、I/O、Web 接口、可视化、高性能通信、数据映射等。应用枝干之所以能够在抽象的层次上进行编程，得益于网格枝干对细节的封装。

再下面是 MPICH-G2 层。MPICH-G2 提供了与 MPI 完全一致的并行编程界面，只不过它已经从并行计算机扩展到了异构的网格环境。MPICH-G2 与 MPICH-G 一样融入了 Globus 的资源发现、资源分配、运行管理、安全验证等功能，但它的通信效率更高，加强了对服务质量的控制。

除了通过 MPICH-G2 实现统一的编程界面外，Cactus 也通过直接调用 Globus 的网格服务功能，实现了数据分解器、通信调度器等功能模块。

最下面是 Globus 层。MPICH-G2 需要通过动态更新请求在线协同分配器 DUROC 进行资源的协同分配。DUROC 既屏蔽了在不同并行计算机上分配资源的细节，又屏蔽了启动、监视和管理进程的细节。Globus 的网格服务功能相当于网格操作系统，它管理的是更底层的功能，如资源发现、验证、预留以及存贮、通信、安全等问题。

10.5.3 Cactus 应用举例——模拟黑洞

根据爱因斯坦相对论，当质量很大的星球的原子能用尽，将向内部发生坍塌，星体

所有的物质和能量会向内聚合成无限小的一个区域，称为黑洞。这个区域磁场引力太大，任何物质都无法逃脱，包括光和其他形式的射线都将被吸进这个区域中，黑洞之名即由此而来。科学家们推测，每一个星系的中心都至少有一个黑洞。根据测算，我们的银河系中心至少有一个质量 300 万倍于太阳的黑洞。

黑洞特性研究，是当今天文学的热点问题之一。由于无法直接观察到黑洞，只好从一些间接现象来证实它的存在。目前世界范围内正在兴建一些新型引力波探测器。它们可通过检测黑洞碰撞产生的引力波，来帮助科学家发现和研究黑洞。此时，超级计算就大显身手了，它可以事先模拟出黑洞碰撞所产生引力波的特征波形。这些波形不仅能指导实际观测，还可用来对观测结果进行检验判断。科学家们认为，超级计算机模拟黑洞碰撞等新型研究，有可能推动实验物理学进入一个全新的发展阶段。

对黑洞的模拟是一个典型的网格问题：一方面，它需要很大的计算能力。例如，要模拟一大一小两个黑洞的碰撞，其计算量不是单台超级计算机所能完成的。为此，就必须借助网格，把计算任务分解到多台超级计算机上；另一方面，要使分布在各地的、不同专业背景的研究人员能够紧密协作，就必须有一个能够在动态的多机构虚拟组织中进行资源共享和协同解决问题的公共平台。仅从这个意义上讲，网格也是必不可少的。

利用 Cactus 并与 Globus 结合模拟黑洞的项目，是在德国 Max Planck 引力物理研究所（即阿尔伯特．爱因斯坦研究所）的带领下，由德国和美国多个研究机构共同完成的。在 Supercomputing 2001 会议上，他们展示了最新成果，并获得了 Gordon Bell 奖。可以说，这个项目相当引人注目。

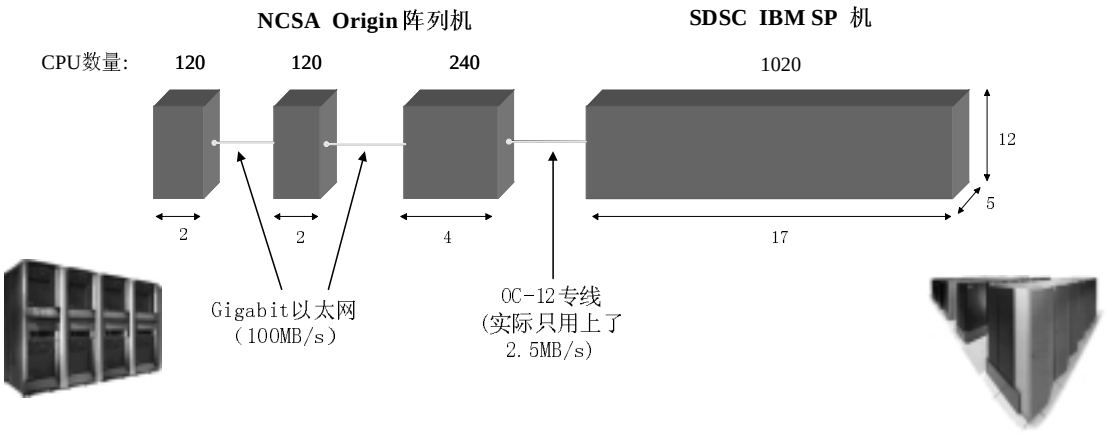


图 10-10　黑洞模拟网格拓朴结构

该实验的网格拓朴结构如图 10-10[2]所示。该网格一共使用了四台并行计算机，它们分布在两个地方：三台位于美国伊利诺斯州 Champaign-Urbana 的国家超级计算应用中心 NCSA(National Center for Supercomputing Applications)，分别是 128、128 和 256 个 CPU 的 SGI Origin 2000 机，相互之间使用 Gigabit 以太网连接，带宽达到 100MB/s；另一台位于加州的圣地亚哥超级计算中心 SDSC（San Diego Supercomputing Center），它是具有 1024 个 CPU 的 IBM Power-SP 机。NCSA 和 SDSC 之间使用 OC-12 专线连接，它的带宽是 622Mb/s，但实际测试效果不理想，只达到 2.5MB/s。

所有的处理器组成了一个虚拟的三维长方体阵列。每台并行机首先在 X 和 Y 方向都分别安排了 5 个和 12 个处理器，构成一个 5×12 平面，然后，视该机处理器总数，将该平面向 Z 方向延伸。为此，NCSA 的三台 SGI Origin 机分别用了 5×12×2、5×12×2 和 5×12×4 个处理器，SDSC 的 IBM Power-SP 机用了 5×12×17 个处理器。所以，处理器阵列的尺寸为 5×12×25，用上的处理器总数为 1500。

需要计算的问题本身可以划分为 360×720×3345 个子问题，由于 IBM Power-SP 每个处理器的处理能力大约是 SGI Origin 2000 单个处理器的两倍，故在 IBM 机的每个处理器上放了 70×60×155 个子问题，而在 SGI 的每个处理器上只放了 70×60×95 个子问题。

由于问题本身的特殊性，每个处理器只需要与紧邻它的处理器通信。这个特性使得该应用非常适合分布到不同的并行计算机上处理，因为绝大多数的通信都只发生在并行计算机内部，即使发生在 SGI Origin 2000 机之间，问题也不大，因为它们之间的网络带宽高达 100MB/s，唯一需要重点考虑的问题是 NCSA 最右侧的 5×7 处理器阵列与 SDSC 最左侧的 5×7 处理器阵列之间的通信效率问题。事实上，这个问题的解决也是比较容易的，只需要给这两侧的处理器阵列分配较少的任务即可。由于任务少，所以它们的通信需求就相对少些。

该项目还采用了其他一些手段来优化性能。例如，使用 TCP/IP 协议跨越广域网进行通信时，TCP 建立连接的时间非常长。为此，可以增加数据包大小，减少发送次数，从而提高通信效率；再如，将计算结果数据进行可视化，也面临海量数据传送问题。好在黑洞引力波是连续的，可以采用压缩后再传输的方法，有时压缩后节省的数据量竟高达 99%。由于 Cactus 非常易于扩展，只需要编写一个压缩插件，并插入到 Cactus 主干中即可实现压缩功能。

需要指出的是，如果优化只是针对固定的配置手工指定的，则有两个问题：一方面很难使优化参数正好与实际情况匹配；另一方面，网络和高性能计算机在运行过程中会不断出现变动，手工优化无法适应变动。为此，该项目利用了 Globus 的动态更新请求在线协同分配器 DUROC(Dynamically-Updated Request Online Coallocator)功能，实现了自适应优化——自动实现负载平衡，自动根据网络带宽和时延优化配置和设定通信参数，自动选择压缩方式（在压缩比与压缩开销之间作出平衡），自动容错，等等。

实验结果表明，优化后的系统性能大大优于优化前。优化前，应用程序的整体运行效率只有 15%，而优化后的整体效率达到了 63%（1500 个 CPU 时）[10]。这个效率是这样算出来的：

$$E = \frac{t_{comp}}{t_{tot}} = \frac{249GFlop/s}{480\times 168MFlop/s + 1020\times 306MFlop/s} = 63.3\%$$

其中，168MFlop/s 是该应用在单个 SGI Origin 2000 处理器上的运行速度，这种处理器一共有 480 个；306MFlop/s 是在单个 IBM Power-SP 处理器上的运行速度，这种处理器一共有 1020 个；249GFlop/s 是应用程序在整个网格上的运行速度。

试验中，尝试过只使用一台 SGI Origin 2000（120 个 CPU）和 IBM Power-SP（1020 个 CPU），曾创下高达 88%的使用效率，这无疑非常惊人。

图 10-11[3]展示了模拟的漂亮结果。

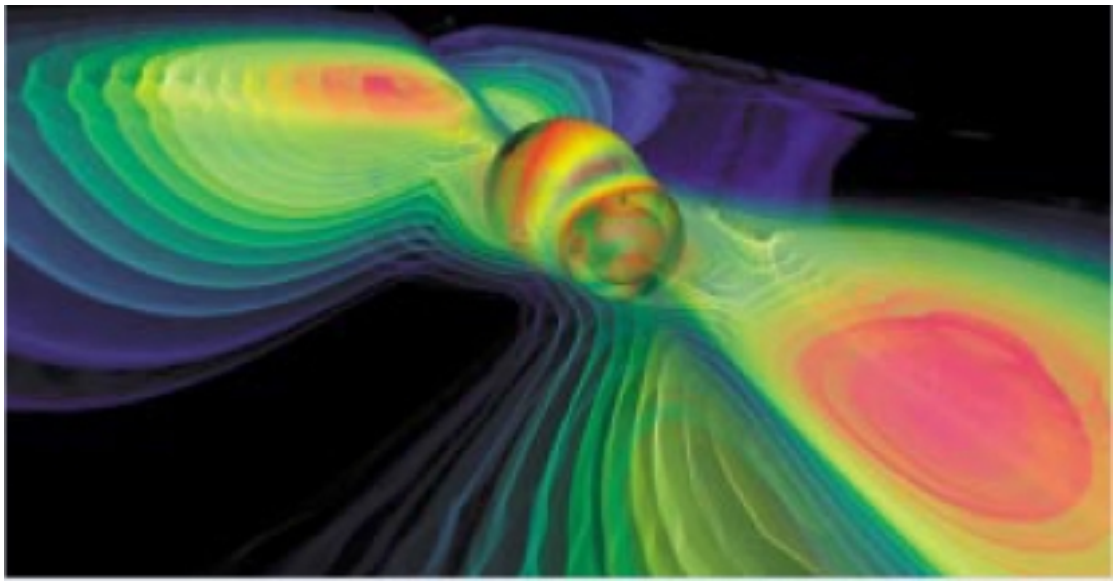


图 10-11　模拟的黑洞

尽管已经非常成功，但作者认为该项目还存在一定局限性：

1. 它虽然使用了四台并行计算机，但由于其中三台类型相同来自同一地点，不能算是一个复杂的异构环境。而且，有一句话是在 Cactus 的获奖论文中没有明白地讲出来的：只使用两台并行计算机，得到的性能（285.5GFlop/s，这个数值是从其 88%的高使用效率反推出来的）居然比四台（249GFlop/s）还高。
2. 系统只使用了 1500 个处理器，规模甚至比不上某些单台超级计算机，如当时排名世界第一的 ASCI White 就拥有 8192 个处理器。因此，这个实验并不能充分证明网格的优越性。
3. 由于应用的特殊性，只是相邻处理器之间需要通信，因而 NCSA 和 SDSC 之间的远程连接没有成为一个大的障碍，优化工作相对容易。
4. 实验工作做得不够细，例如 NCSA 和 SDSC 之间使用的 OC-12 专线带宽的实际传输性能只是理论值的 3%多一点，应当找出其原因。

尽管如此，我们还是认为 Cactus 及其应用，是网格发展史上的一个里程碑。

10.6　小结

虽然计算机的计算能力在过去的 50 多年里呈指数增长，但计算需求却膨胀得更快，单台超级计算机很难满足一些新的计算需求。于是，计算网格出现了。它把大型计算任务按步骤、按功能或按数据分解成较小的任务，分配到不同的超级计算机上运算，从一定程度上克服了计算能力不足的问题。

不过，并不是用网络把超级计算机连接在一起就变成了网格，还必须开发一套中间件软件将它们融合起来。中间件软件需要管理所有的网格资源，用合理的调度策略进行资源分配，用适应性算法来应对网络的动态变化。另外，在复杂的异构环境中出现错误是不可避免的，需要有容错措施。

本章介绍了两个典型的分布式超级计算应用：一个是军事仿真，另一个是天文模拟。它们都基于 Globus，不同的是后者在 Globus 之上还有一个应用支撑平台 Cactus。Cactus 是专

门的计算科学和工程问题解决环境，它大大简化了开发网格应用的难度。

思考题

1 网格在大规模科学计算中能够扮演什么样的角色？

2 对于分布式超级计算而言，哪些技术是至关重要的？

3 假设有三个应用 A、B 和 C，它们各自在不同的运行步骤对共享资源(a)、(b)、(c)、(d)、(e)和(f)的需求如下所示：

步骤	应用 A	应用 B	应用 C
第 1 步	(a)	(a) (b)	(c) (e)
第 2 步	(b) (c)	(c) (d)	(d) (f)
第 3 步	(d) (e)	(e)	(a) (b)

有没有一种调度方案，能将总体完成步骤控制在 4 步之内？

对于你所画出来的任何一种调度方案，如何判断它是否正确？

试编写一个程序，使资源调度策略能够自动实现。要求程序能够显示出所有可能的最佳调度方案，请用此例作为测试数据。

4 SF Express 用什么方法削减网格上的通信量？ 事实上，即使没有网格，照样可以完成 SF Express 的功能，为什么它还需要在 Globus 的支持下实现网格化？

5 你认为 Cactus 最精彩的思想是什么？

第11章 分布式仪器系统

分布式仪器系统（Distributed Instrumentation System）是指以网格管理分布在各地的贵重仪器系统，提供远程访问和控制仪器的手段，提高仪器的利用率，大大方便用户的使用。其实，正如11.1节所介绍，在网格出现之前，人们就试图通过网络访问一些仪器设备或仪器数据，但当时的软硬件环境都不成熟，只能实现一些低要求应用罢了，而网格将分布式仪器系统变成了一个非常易于管理和有弹性的系统。建立分布式仪器系统涉及一些关键技术，特别是网络化的海量数据存贮技术，因为伴随贵重仪器系统的往往是大量的数据，这些技术将在11.2节介绍。11.3节介绍了一个有代表性的分布式仪器系统 XPort。

11.1 背景

我们以远程医疗为例，来看看网格能带来什么样的变化。

远程医疗(Telemedicine)是指是用远程通讯技术以及计算机技术提供医学及有关信息服务，它包括[27]：远程诊断(Remote Diagnosis)、远程医疗会诊(Remote Consultation)、远程教学(Distance Learning)、远程医疗信息服务(Remote Medical Information Services)等等。远程医疗不仅有助于克服边远地区医疗资源缺乏的困难，还可以通过远程会诊和远程教学等手段提高医疗水平。

远程医疗系统一般包括以下几部分[28]：(1) 视频电话，使各方人员能进行面对面的讨论；(2) 远程出席系统及重要生理参数的远程监护系统，使 A 地的专家能观察到 B 地病人的现状，进行检查及指导；(3) 远程放射学(Teleradiology)和图片归档及通信系统 PACS(Picture Archiving and Communication System)系统。在许多场合 X 光片是一个重要的诊断依据。有了远程放射学的支持，偏远地区的医护人员就可以将放射影像数字化后传送给中心医院，这样中心医院的专家就可以对着图像进行讨论。而 PACS 可以将医学影像进行有效管理，方便本地和远程用户随时查阅。可见，远程放射学和 PACS 是实现远程医疗系统的关键技术。

远程放射学已经有几十年的历史[29]。最初人们用微波、卫星等手段传送图像，并幻想能用电视机显示 X 线胶片，先后使用了 525 线普通电视机及 945 线高清晰电视机作显示设备，结果发现电视摄影系统提供的模拟信号的图像清晰度难以满足诊断的需要。1970 年后，计算机技术与医疗成像技术的结合，大大促进了以 CT 为代表（包括 US、DSA、MR 等）的数字成像技术的发展。1980 年代初在传输速率为 9600bits/s 的低速网上试验传输 512×512×8bits 的数字化影像，虽然影像还不如传统的 X 线片，传输过程也太长，但已经跨出了至关重要的一步。至 1980 年代后期，采用激光系统读取胶片，数字化程度可达到 4090×4090×12bits, 而计算机化 X 射线成像系统 CR(computed radiography)甚至可以不再经过胶片这个中间环节，直接产生数字化图像。另外，从传输上看，DS－3 的传输率可达 44.736Mbits/s。所以，那时实现真正意义上的远程放射学的基础技术已经成熟。

然而，远程医疗仅在欧美发达国家的某些偏远地区使用得较多，在我国更是凤毛麟角：上海医疗卫星会诊中心于 1996 年底开通，并由上海华山医院与安徽巢湖地区人民医院通过卫星实现会诊。1997 年 4 月，北京协和医院与美国医学专家通过电视会议系统为一名中国男孩进行了成功的会诊[30]。

与其巨大潜力相比，远程医疗还远不够普及，究其原因，一方面是成本问题，另一方面是技术问题。从成本上讲，远程医疗涉及各类技术、医疗和服务人员的参与，人力成本很高。另外，需要购置昂贵的设备，使用和维修设备也需要有资金支持，正常运行还需要租用昂贵的宽带通信线路。这些因素造成远程医疗的成本居高不下；从技术上讲，如果远程医疗完全基于 Internet(特别是未来的宽带网)，无疑能为它的普及插上腾飞的翅膀。但在公共的信息传输基础设施上，既会出现网络资源的不稳定问题(如出现拥塞)，又有访问控制和信息安全问题，还有海量存贮和信息检索问题，等等。技术问题如果得到很好解决，远程医疗的成本就会大大降低，从而吸引更多的人参与其中，使资源(设备、人员、图像信息和医疗方案等)得到更好的共享……成本在这个良性循环中还会进一步降低。

网格在远程医疗中将扮演这样的角色：管理系统各种设备，动态调度资源，提供资源预留服务，自适应网络的拥塞，提供海量数据的实时存贮和检索服务，在设备和远程存贮系统之间提供传输服务，对动态可视化和分析提供支持，支持对远程仪器的控制，促成专业人员之间的协作。有了网格的支持，远程医疗系统就可以建立在宽带 Internet 之上，而不必租用专用线路，各种资源的利用率以及协作水平也将大幅提高。

近期，英国剑桥大学与西英格兰癌症网 WACN(West Anglia Cancer Network)正在联合进行基于网格的远程医疗试验，对 160 万人口提供癌症医疗服务[31]。该系统将支持多方视频交流，以及放射图像的实时传输、存贮和检索。特别地，该系统将远程接入医疗模拟计算应用程序，并对病人记录进行数据挖掘以辅助临床决策。有了该系统的支持，病人就可以到最近的癌症治疗分部，通过网络接受癌症专家的诊断和治疗(如图 11-1[31]所示)。

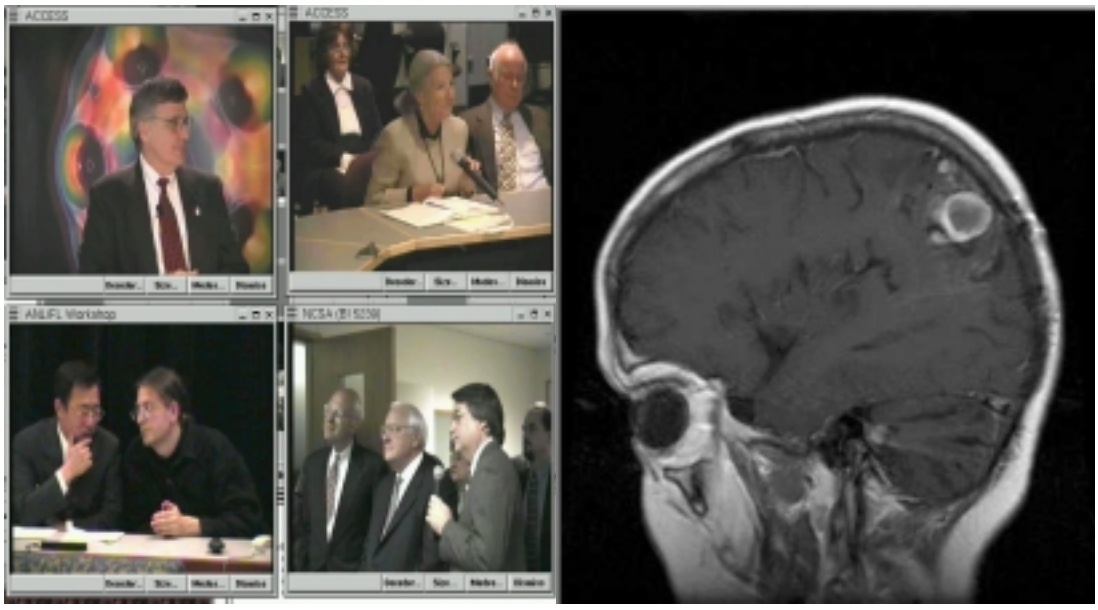


图 11-1　脑肿瘤的会诊

11.2 分布式仪器系统的核心技术

对分布式的仪器进行实时管理，需要网格在底层和中间件层提供支持[32]。对底层而言，需要有灵活的传输机制、广域网广播服务、资源预留服务，并提供计算、存贮和通信的 QoS 保障，还要提供底层的安全机制，等等；对中间件层而言，需要提供与第三方海量网络存贮系统的接口，并建立检索索引，对分布仪器、数据库、计算机等构件进行自动监控和管理，用基于策略的访问控制(如 QoS)支持资源调度和分配，提供安全措施，保障分布式系统的完整性，提供支持远程出席和协作的实时多媒体措施，等等。

11.2.1 基于网络的海量存贮系统 HPSS

几乎是所有分布式仪器系统都需要使用基于网络的海量存贮系统，之所以如此，是因为这些仪器几乎都会产生大量的数据，而且这些数据都需要存贮、管理并支持来自异地的访问，所以使用网络化海量存贮系统是很自然的选择。

目前比较流行的网络化海量存贮系统有高性能存贮系统 HPSS(High Performance Storage System)和分布式并行存贮系统 DPSS(Distributed-Parallel Storage System)等，目前也在研制网格化的更大容量、更高性能的数据网络(Data Grid)。

这里简单介绍一下 HPSS，数据网格留在第 12 章介绍。

人们对海量存贮能力和高速处理能力的需求是没有止境的：1994 年数据存贮系统的容量达到 10TB(1TB=10^{12} Bytes)，吞吐量达到 10MB/s；1997 年容量达到 100TB，吞吐量达到 100MB/s；2000 年容量达到 1PB(1PB=10^{15} Bytes)，吞吐量达到 1GB/s。现在，人们又向更高目标迈进，HPSS 便是其中的一个里程碑。

HPSS 项目始于 1993 年，由政府和工业界联合研制，目前已推出 5 个版本。参加 HPSS 联盟的机构达 20 多家，包括美国能源部 DOE、美国国家科学基金 NSF(National Science Foundation)超级计算机中心、著名公司(如 IBM)、一些联邦实验室和知名大学[33]。HPSS 目前已经被广泛采用。

HPSS 是一个层次化的存贮系统，主要为高性能计算产生的海量数据(TB 甚至 PB 量级)提供快速存贮服务，弥补越来越严重的计算能力与 I/O 吞吐能力之间的差距(计算能力每年提高 50%，而存贮能力仅为 20%)。

HPSS 的主要设计目标是让海量数据能够在网络化存贮、高性能计算机、集群计算机、海量数据库之间快速传递。例如，HPSS 能够以 1Gb/s 的速度从多台网络连接的磁盘阵列传出数据，实现高清晰度的实时视频图像传递。HPSS 有以下特征[34][35]：

1　以网络为中心

HPSS 架构中的所有计算机和存贮结点都直接连接在网络上，数据在提供者与需求者之间直接传递，无需经过中间结点。HPSS 的架构如图 11-2[36]所示。可以看出，HPSS 完全是以网络为中心的。

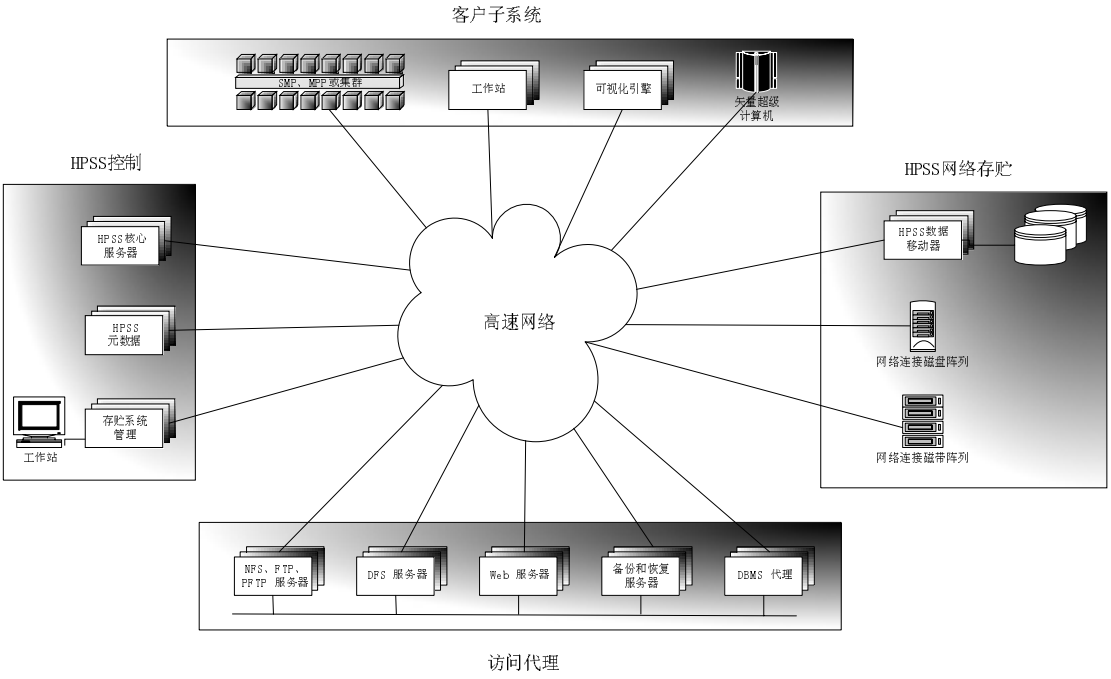


图 11-2　HPSS 架构

2　并行处理

HPSS 把数据分散在多个结点上，这些结点并行传送数据，因而吞吐量接近这些结点吞吐量之和。

3　支持事务操作

HPSS 支持事务的原子操作——对于一次操作，要么彻底完成，要么回滚到事务开始之前的状态，从而保证了数据的完整性。

4　模块化

HPSS 是一些软件模块的集合，每个模块担负一个任务，模块之间松散耦合，以 API 提供调用接口，可以象搭积木一样组合使用。 由于实现模块化，可以很容易地替换其中的软件模块，也可以很方便地把 HPSS 的功能融入到其他数据管理系统中。

5　可移置性

HPSS 不需要特定操作系统的支持，它用 ANSI C 书写，调用 POSIX 功能，因而可以运行在许多平台上。

6　可扩展性

HPSS 的性能还可以动态扩展，允许随时将新的服务器、磁盘阵列、磁带阵列加入到 HPSS 的网络存贮池中，从而满足不断增长的应用需求。

简言之，对于分布式仪器系统而言，HPSS 能够提供高性能的数据缓冲和存贮服务——容量高达 PB 量级，吞吐量高达 GB/s 量级。

11.2.2 分布式监控

在分布式仪器系统中，仪器、数据采集器、仪器控制器、网络存贮器、大型计算机、可视化终端等各种设施分散于各地，需要进行全局协调和管理，这些工作可以由分布于各地的

代理程序来完成。除了完成特定的功能外，代理程序必须对全局性的监控进行支持。

当分布式系统出现故障时，往往很难知道哪里出了问题，或者需要花费很大的时间和精力才能知道。当分布式系统的性能下降时，由于网络环境的复杂性和不稳定性，往往很难找到系统的瓶颈所在。这时，就需要代理程序担负起分布式监控的任务，它们实时记录下每个有潜在意义的事件，为全局的实时监控提供支持。处于系统性能枢纽位置的代理程序，还需要对某些关键性能参数作下记录，支持对系统性能的分析和调整。

11.2.3 基于策略的访问控制

分布式仪器系统的安全管理非常复杂，它涉及来自不同机构、位于不同地方的多个管理者和多个参与者，而且不同的设施许可了不同的人使用，且指定了不同的权限。显然，不可能人工完成每一次授权，也不可能让用户只要有一个口令就通行无阻。

比较好的解决办法是让所有设施的拥有者签署数字授权文档，指定被授权人、用户性质、使用前提等。系统资源访问控制部件负责收集所有这些授权文档，并在用户发起授权申请时，用一个策略控制引擎代替所有的授权人完成授权操作。

采用这个机制的好处是：能够及时完成授权，不影响合法用户正常访问资源；“不多不少”——资源既不被滥用，也不被过度限制；不增加设施拥有者的额外负担。

11.3 XPort——X 射线设备的科学门户

11.3.1 背景

XPort 项目由美国能源部 DOE（Depart of Energy）下一代因特网 NGI（Next Generation Internet）试验床计划[12]资助，由印第安那州大学、Argonne 国家实验室 ANL(Argonne National Laboratory)和劳伦斯市伯克利国家实验室 LBL（Lawrence Berkeley National Laboratory）共同完成，其目标是用让远程使用科学仪器达到前所未有的方便程度，拿他们的话来说，就是“比到那儿用还方便”。

XPort 使用的科学仪器是几个高亮度 X 射线结晶学设备，包括 ANL 的先进光子源 APS[16]（Advanced Photon Source）和 LBL 的先进光源 ALS[17]（Advanced Light Source），以及印第安那州大学的分子结构中心 MSC[18]（Molecular Structure Center）[13]。晶体在 X 射线的照射下产生衍射，其图像可以用来确定晶体的结晶单元的尺寸和原子的位置。对于大型的分子，衍射图案非常紧密，图像需要极高的分辨率才能被辨识。为此，象先进光源 ALS 和先进光子源 APS 这样的高亮度 X 射线源是必不可少的[19]。

XPort 平台基于 NGI 和 Globus，能提供远程仪器使用规划、仪器操作、数据获取、筛选和分析等功能，它将大大简化巨型分子晶体结构的设计和实施。为此，XPort 需要结合先进的网络技术、中间件服务和远程仪器操控技术。XPort 所涉及的技术问题主要包括：高速数据的采集、筛选、存贮、可视化和实时仪器控制等。

虽然像 APS、ALS 和 MSC 这些设施非常昂贵，具有很大的科学价值，然而在没有 Xort 之前，使用这些设施是非常不方便的。通常，一个科技工作者要想用上这种设备，需要预约排队好几个月，然后千里迢迢赶过去，接着采集数据又得花上好些天（同时也造成仪器的利

用率低下），最后分析这些数据还得花很长的时间。有了 XPort 之后，科技工作者们就可以远程提交任务，交互式控制任务的执行，接收和分析初步的运行结果以确定任务的正确性，存贮和管理产生的数据，而且还可以与其他科技工作者即时共享其研究结果，探讨新的解决方法。

可见，XPort 带来的好处是明显的，一方面它大大缩短了研究时间，提高了设备的利用率，使普通的科技工作者能够用上先进设备；另一方面它提供了一个协同研究的平台，使研究能够以团队的方式开展，为交叉学科研究创造了条件。应该说，正是 NGI、Globus、Cactus、XPort 这样的研究项目，为未来的“大科学”研究奠定了坚实的基础。

11.3.2 XPort 试验流程

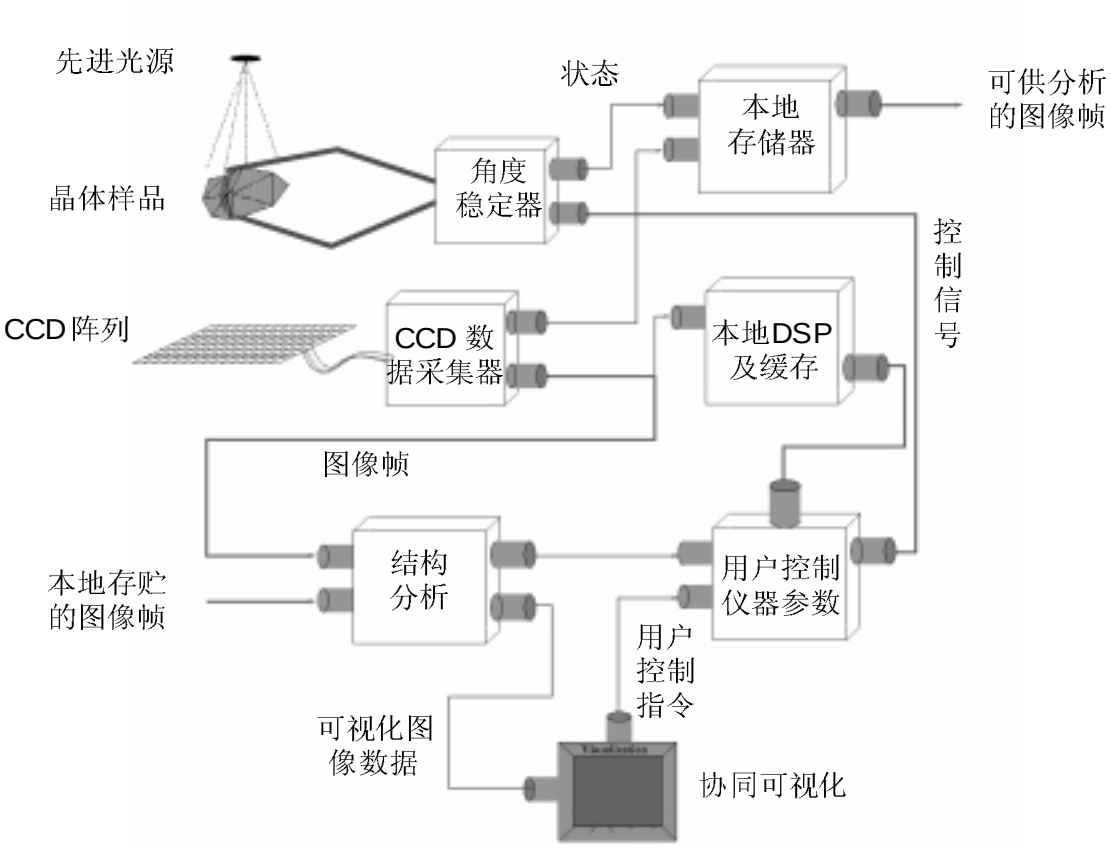


图 11-3　XPort 处理流程

XPort 的处理流程如图 11-3所示[15]。电荷耦合组件 CCD（Charge Coupled Device）每两秒采样一次，每帧数据大约 9-32MB，平均一个实验过程大约需要 12 个小时。为了保证数据传输，XPort 平台建立在 DOE 的 NGI 宽带网络试验平台上，专用 128Mbit/s 带宽[11]。采集的数据既在本地存贮，也使用宽带网远程传输给数据处理中心进行晶体的结构分析。分析的结果以可视化的形式呈现出来，提供给各地的研究人员协同分析。

试验的流程如下：

1. 用户把试验晶体样品用快递邮包发给 LBL
2. 通过远程可视会议，用户指导 LBL 技术人员在 ALS 设备上放置样品

3. 用户指定初始的扫描参数
4. 初步的扫描图像帧传回给用户
5. 用户检查样品的正确性、放置的正确性以及样品是否形成了孪晶
6. 如是，用户重新指定扫描参数、让技术人员重新放置样品或终止扫描过程
7. 与异地研究人员协同处理采集到的大量数据

11.3.3 XPort 的实现

XPort 的体系结构如图 11-4[13]所示。

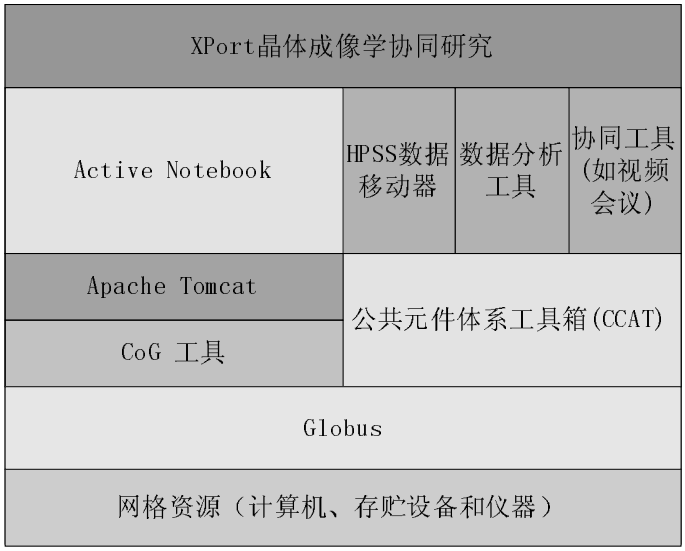


图 11-4 XPort 体系结构

底层主要使用了 Globus 的身份验证和任务执行功能。CoG 工具包(Commodity Grid Kits)[20]的功能是使应用程序可以通过 COBRA、Java、Perl 和 Python 等访问网格服务，包括 MDS、GRAM、GridFTP 和 GASS 等。

通常，运行一个分布式应用程序是很困难的，科学门户（Science Portals）[21]提供一个 Web 界面让用户运行应用程序。Active Notebook 是印第安那州大学开发的科学门户软件，它基于 Apache Tomcat 提供一个便携式网格计算问题解决环境。这个门户是一个安装在用户桌面计算机或笔记本电脑上的 Web 服务程序，它提供像“笔记本”一样简单的界面，供用户编写应用程序脚本及管理数据等。有了科学门户，用户只需要在 Web 页面上填写应用程序的运行参数，并指定初始数据的路径，就可以运行之。因而，用户不必关心程序的运行是如何完成的，大大降低了网格应用的门槛。

XPort 最关键的中间件是公共元件体系工具箱 CCAT(Common Component Architecture Toolkit)。元件体系 CA(Component Architecture)是一种软件工程方法，它采用构件的思想，所有的构件都是经过封装的软件模块，其行为和接口经过严格定义。这样，元件体系能够很好地支持软件复用，降低应用程序的复杂度。CCAT 是美国能源部公共元件体系计划 CCAP（Common Component Architecture Project）[22]中的一部分，也是由印第安那州大学完成。相对于已有的桌面构件体系（如微软的 COM、Java Beans、Gnome CORBA 构件）和分布式构件体系（如 Enterprise Java Bean、DCOM,、CORBA Component Model），CCAP 专门针对用于高性能计算的并行编程环境，它更强调性能的提高。目前，CCAT 已经演化成 XCAT[24]。

图 11-3所示的 XPort 处理流程中的每一个处理框都是一个 CCAT 元件，它们用 C++或 Java 书写，运行于不同的计算机上。采集 CCD 数据的 CCAT 元件是与仪器结合的关键，它的实现却非常简单：CCD 器件本身就带有相应的驱动程序，它把图像帧存入指定的目录。而 CCAT 数据采集元件只需要监视该目录的变动情况，把最新产生的文件传输给存贮服务器即可。

还有一个问题是 CCAT 基于 Windows NT 平台，而它使用的 Globus 功能却是基于 Linux 平台的，如何把这二者结合起来？解决方法[13]是开发一个基于 Globus 的代理程序，它通过 SSH(Secure SHell)实例化 NT 上的元件功能。还有一种更为通用的机制，就是用 Servlet 来实例化 CCAT 元件，而不通过 Globus 资源管理器。

在 CCAT 之上，还需要使用其他软件系统提供的一些服务，例如，高性能存贮系统 HPSS（High Performance Storage System）[26]提供的海量数据高性能网络存贮服务(数据移动器负责把 CCD 图像传送到海量存贮)，巨型分子生物学家所使用的衍射图案分析工具包(如 MOSFLG、CCP4 和 D*TREK 等)，以及用于用户之间、用户与设备提供者之间视频沟通的工具，等等。

图 11-5 [15]是一个在 XPort 上完成的晶体分析可视化结果。

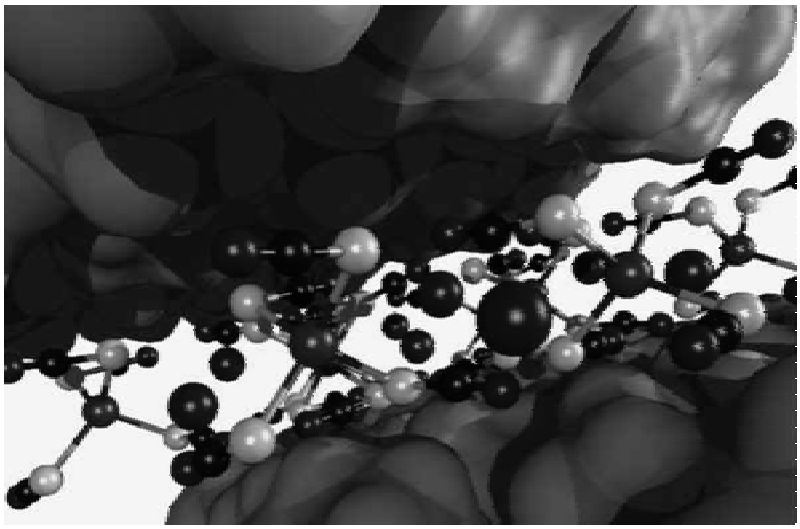


图 11-5　XPort 上完成的晶体可视化结果

11.4　小结

远程医疗经历了一个从简单到复杂、从低质量到高质量、从单一到综合的演化过程。从它的发展史可以看出：网格扮演了一个综合各种技术、揉合各种平台、屏蔽各种差异的角色。试想，既然有一种平台，它既融合了存贮海量数据的能力，又能提供有保障的传输服务，还能解决系统的安全性和完整性问题，为什么不基于这个平台建立分布式仪器系统，却还要另辟蹊径呢？所以，网格化是分布式仪器系统的必经之路。

XPort 应用体现了网格带来的仪器使用方式的革命。从它的实现上看，有两点值得注意：(1)分布式仪器系统的中间件是由一系列分布在网上的代理程序构成的，在 XPort 中，它们以公共元件体系构件的形式出现，非常符合软件工程的思想；(2) 看 XPort 怎样连接基于 Windows 平台的 CCAT 和基于 Linux 平台的 Globus。

思考题

1 对于分布式仪器系统而言，哪些技术是至关重要的？
2 HPSS 相对于传统的存贮系统有哪些优点？
3 同样是进行 X 射线结晶学实验，使用网格化的 XPort 相对于传统的做法，有什么样的变革？
4 XPort 如何解决 CCAT 所依赖的 Windows NT 平台与 Globus 所依赖的 Linux 平台的共存问题？

第12章 数据密集型计算

并行计算技术往往是由一些计算密集型应用推动着的，特别是一些带有重大挑战（Grand Challenge）性质的应用，它们大大促进了对高性能并行体系结构、编程环境、大规模可视化等领域的研究。但是，相比之下，数据密集型计算(Data Intensive Computing)的应用好像比计算密集型应用多得多[38]。它对应的数据网格更侧重于数据的存贮、传输和处理，而计算网格则更侧重于计算能力的提高，所以它们的侧重点和实现技术是不同的。本章12.1节首先介绍是什么样的应用推动了数据网格的发展，特别地，在12.2节里详细介绍了欧洲原子能研究机构 CERN (European Organization for Nuclear Research)[37]的数据网格应用需求，该应用促使数据网格将规模扩大到了前所未有的程度，12.3节介绍 CERN 所建立的 DataGrid 数据网格[40]的相关技术问题。对于规模巨大、参与单位众多的网格开发项目，如何进行有效的协调管理，一定其独到之处，这将在12.4节介绍。

12.1 背景

通过本节的 4 个实例，可以看出有了数据网格的支持，信息共享和信息处理能力将达到前所未有的水平，许多领域的研究水平将大幅提升。本节的前两个例子是欧洲数据网格 DataGrid 上的试验床项目，后两个例子是美国 NPACI[41]网格的试验床项目。

12.1.1 生物和医学

虽然人类基因组计划在媒体的曝光率很高，相对于人的细胞里每个DNA所含的35亿对基因而言，科学家们已经完成的基因分析工作只是极少的一部分。要全部标注这些基因需要很大的数据存贮及处理能力。

DataGrid将为该类计划提供新的高性能数据库支持、代码管理、数据挖掘的新方法、交互式图形界面、新的实验手段以及与国际同行共享研究成果的手段。

数据网格在生物、医学领域的另一个应用是处理医学图像[39]，它在实时图像获取、处理、存贮、共享、检索等多个方面具有独到的优势。相比11.2.1节所介绍的基于网络的海量存贮系统HPSS，数据网格的存贮能力和处理能力与之不在同一个数量级上。

12.1.2 地球观察

欧洲空间局管理了几个地球观察卫星。这些卫星每天会下传大约100GB的图像，新卫星ENVISAT发射之后，数据量又增大了5倍。为此，地面站建立了专用的设施来处理这些数据，目前已经保存了上千万亿字节的数据。

DataGrid将使处理卫星数据的能力大大提高[39]，它不仅将海量数据分散到整个欧洲范围内保存，提供更高效的访问方法，还提供了更强的处理能力。在此基础上，对卫星数据的研究水平将大大提高，例如，目前已经利用大气中臭氧层的数据建立了一个专用试验床。

12.1.3 数字天空

建立大范围的数字天空是当今天文学的热点。天文学们使用光学、射电、远红外望远镜测定并记录所有能与噪声相区别的天体，而NPACI万亿字节的存贮能力及万亿次的处理能力，给研究这些数据提供了前所未有的支持，例如，现在就有条件对全部数据进行某种统计分析，筛选出具有某些属性的天体来，像这种研究方法以前是不敢想象的[42]。

12.1.4 大脑映射

近年人类对以前的“黑盒子”——大脑——所了解的结构信息和功能信息呈爆炸式膨胀。在一些试验系统里，神经学家们针对老鼠和蟋蟀等动物已经积累了相当多的映射数据。然而，没有网格的支持，处理这么大量的数据（一个实验就达 GB 量级）是非常困难的。

在 NPACI 的支持下，不同地方的神经学家们组成了研究联盟，使得不同实验的数据能够积累下来，并因此发展出一些全新的大脑分析手段，例如，使用空间变形算法比较不同物种（如人与猴子）之间大脑的异同[42]。

12.2 CERN 与 DataGrid

12.2.1 欧洲原子能研究机构 CERN

欧洲原子能研究机构 CERN 成立于 1954 年，是世界最大的粒子物理研究中心。CERN 是欧洲第一个联合研究机构，由 20 个成员国提供资金，其卓越成绩已经成为国际合作的典范[43]。

CERN 位于法国和瑞士的交界处，就在日内瓦的郊区。CERN 主要研究物质是怎样构成的，以及是什么把它们结合起来的。CERN 建有世界上最大的正负电子对撞机 LEP（Large Electron-Positron collider）和超级质子同步加速器 SPS(Super Proton Synchrotron)，如图 12-1 所示，大圆是 LEP，周长达到 27 公里，粒子能够加速到每秒运动 11000 周，接近光速。小圆是 SPS。与加速器配套的是四层楼高的粒子检测器，用于对粒子性质进行分析。

图 12-1　CERN（大圆为 LEP，小圆为 SPS）

来自全世界 80 多个国家、500 多所大学及研究机构的 6500 多名科学家（占全球粒子物理学家的一半）在 CERN 进行各种各样的实验。CERN 自身拥有各种各样的技术支持人员，包括物理学家、工程师、程序员、技术人员、管理人员、工人等，他们负责建立各种复杂的设施并保障其正常运转。在 CERN 完成的实验也是前所未有的，通常需要数百名科学家在巨大的设备上共同完成，一个实验昼夜不停，持续数月乃至数年。

顺便说一句，CERN 同时也是寰球网 WWW(World Wide Web)的发源地[44]。1990 年，CERN 的计算机科学家 Tim Berners-Lee，为了方便分布于世界各地的高能物理学家之间的协作，设想和开发了 WWW 客户端和服务器端，还定义了 URL、HTTP、HTML 等。正是由于 Tim 等人的贡献，Internet 才变成了大家今天所习惯的模样。

12.2.2　大型强子对撞机 LHC

目前 CERN 有 1800 名物理学家正在为下一代全新的加速装置作实验准备。该装置称为大型强子对撞机 LHC（Large Hadron Collider），将于 2005 年投入使用，这将是人类历史上最强大的粒子加速器[45]。未来在 LHC 上进行的实验的参加单位、人数及产生的数据量也是前所未有的，这一点可以从 LHC 与 LEP 的对比看出来（表格 12-1[46]）。做个类比，LHC 将要产生的数据量，将是目前 CERN 所有设备产生的数据量的总和还要多一到两个数量级。

表格 12-1　LEP 与 LHC 的对比

	LEP	LHC
原始数据产生速率	1MB/sec	100MB/sec
每年的事件数	$<10^7$	$\sim10^9$
每年的数据量	0.2-0.3 TB	1 PB
平均事件大小	20 – 50 kB	1 MB
实验参加人数	400 - 600	~2000

实验参加组织数	30-50	~150

12.2.3 DataGrid

LHC 的出现将给计算科技带来全新的挑战，为此，需要有空前的计算能力来处理这些数据，空前的人类智慧来分析这些数据，以及空前的存贮能力来保存这些数据。解决这些问题的基本思想是把海量数据分散到全球的计算机上进行处理，并由全球的物理学家共同分析之。在这个背景下，欧洲的数据网格 DataGrid 应运而生了，它成为实现这个“大科学”目标的基础平台。

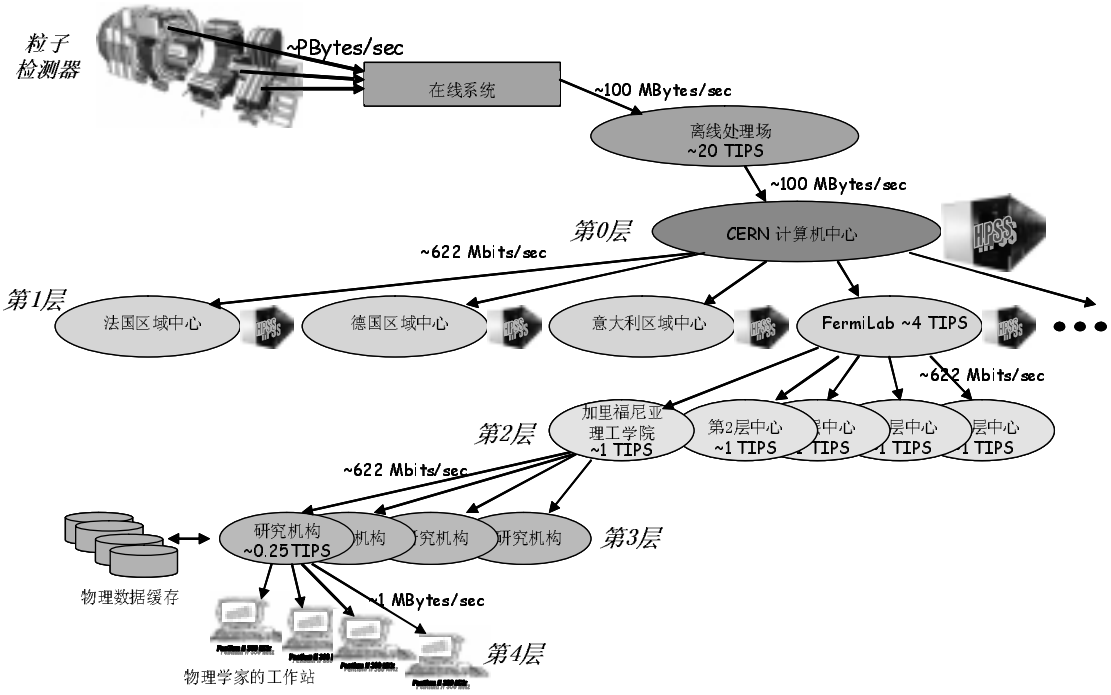


图 12-2 DataGrid 的分布处理策略

DataGrid 对海量数据的分解过程如图 12-2 [47]所示。粒子检测器产生的原始数据具有 PB/s 量级，经过在线系统过滤后，并经具有 20 万亿次处理能力的离线处理场的处理，最终以大约 100MB/s 的速率永久写入磁带，这个 100MB/s 就是 DataGrid 真正需要处理的数据速率。CERN 计算机中心负责将这些数据通过高速网络分配给欧洲、北美、日本等国的区域中心，后者再将任务作进一步分解，到物理学家的桌面时，数据量只有 1MB/s，已经可以很方便地进行处理了。

DataGrid 需要解决许多问题，诸如[46]：

1 DataGrid 需要管理成千上万个处理器和磁盘、千万亿字节（PB）的数据和每秒万亿比特（Tb/s）的网络带宽，面对如此复杂的系统，如何才能保证它的高可扩展性、低成本和易管理性？

2 广域网的带宽只是局域网的 1%－10%，不同的研究机构有不同的管理者和管理政策，如何保证数量要在它们之间安全地分发、复制、缓存并保持同步和完整性？

3 如果才能协调好不同国籍、不同研究机构的科学工作者的工作，使他们及时分

析数据并汇总结果？

这些问题解决好了，不仅对于在 LHC 上开展的实验是至关重要的，对于其他领域的研究也具有指导意义。可以这样说，虽然在高性能计算机、集群、网格等领域里美国一直处于领先地位，但欧洲的 CERN 所开创的大科学研究是美国难以望其项背的。

12.3 DataGrid 的设计

DataGrid 负责人 Gagliardi 这样描述 DataGrid 所要完成的工作[48]：

当用户提交一个任务时，DataGrid 首先分析完成任务所需要的计算资源。然后，找到这些资源并分配给任务。同样地，运行任务所需要的数据也被检索出来并传送给计算资源。在这个过程中，DataGrid 需要具备：分析任务的能力，随时掌握网格中资源的能力，执行任务程序的能力，任意传输数据的能力，判定和保障服务质量的能力，从错误中恢复的能力，记录出错情况的能力，等等。

论文[45]分析了 DataGrid 所应具备的主要功能：

1　负载调度和管理

DataGrid 在管理负载时所面临的新问题有：数据经常需要动态重新分配，系统中可调度组件的数量非常之大，会出现许多用户同时提交任务的情况，不同国家不同机构有不同的管理策略，等等。负载管理在分解和分发任务时，必须基于计算能力和数据的可用性。为此，需要扩展作业描述语言，使之能够描述数据的相关性。负载管理应能比较不同任务分解方法的利弊，为此，它需要预测并综合考虑任务在不同机器的执行时延、生成数据缓存副本的开销、在二级存贮和第三方存贮之间迁移数据的开销，等等。更进一步，负载管理应该有新手段支持资源的协同分配和预留，以及在组件失效时的恢复策略。

2　数据管理

DataGrid 需要开发中间件以支持对海量数据的访问，既要有统一的名字空间和统一的数据格式，又要能在不同站点之间高速移动和复制数据，还要保持远程数据拷贝的一致性，等等。DataGrid 应该有一些优化措施，支持自动的广域网数据高速缓存机制，并能根据用户的使用模式选择数据的分发方式。

3　网格监控

DataGrid 需要有监控网格运行的窗口，它既能站在一定高度纵览全局，又能深入某个局部分析细节状况，为此，需要开发底层 API，提供对计算构件、网络和海量存贮的性能和状态信息。有了这些支持，就能协助制定工作负载和数据管理的调度策略，以及调整应用程序的运行性能。

4　构造层的管理

网格的构造层(fabric)是网格存在的物理实体，没有它们，就谈不上计算和存贮，谈不上资源的可用性和性能，谈不上安全认证，谈不上资源分配。为此，需要有创新性的中间件，对所有的基础构件提供灵活和有弹性的管理。虽然现有网格的研究成果（如 Globus）本身就具有动态配置、自动容错、自适应资源变化以及自动调整性能特性的

能力，但对于 DataGrid 所要面临的问题而言，这些能力就显得不够强了，它不仅要面对成千上万个基础构件，还要满足严格的时间约束条件，必须要有创新性方法，实现自动发现和隔离错误、自动重组构造层、自动重新运行任务、自动把新加入的系统带入基础设施中。

5　海量存贮管理

粒子物理处理大规模数据已经有几十年的经验了。然而，在过去的一些年里，由于每一个加速器中心都有各自的数据格式，不同格式之间的转换给 CERN 的合作数据处理机构带来了许多麻烦。DataGrid 应该提供数据之间的转换接口，同时它还要负责将本地的海量数据存贮系统集成到网格的数据管理系统中。换言之， DataGrid 要用统一的接口屏蔽不同站点的数据存贮方式和处理方式之间的差异，使分布的存贮资源能够无缝融合。

DataGrid 的体系结构[50]如下所示。由于 DataGrid 尚在研制过程中，它还不是最终版本。

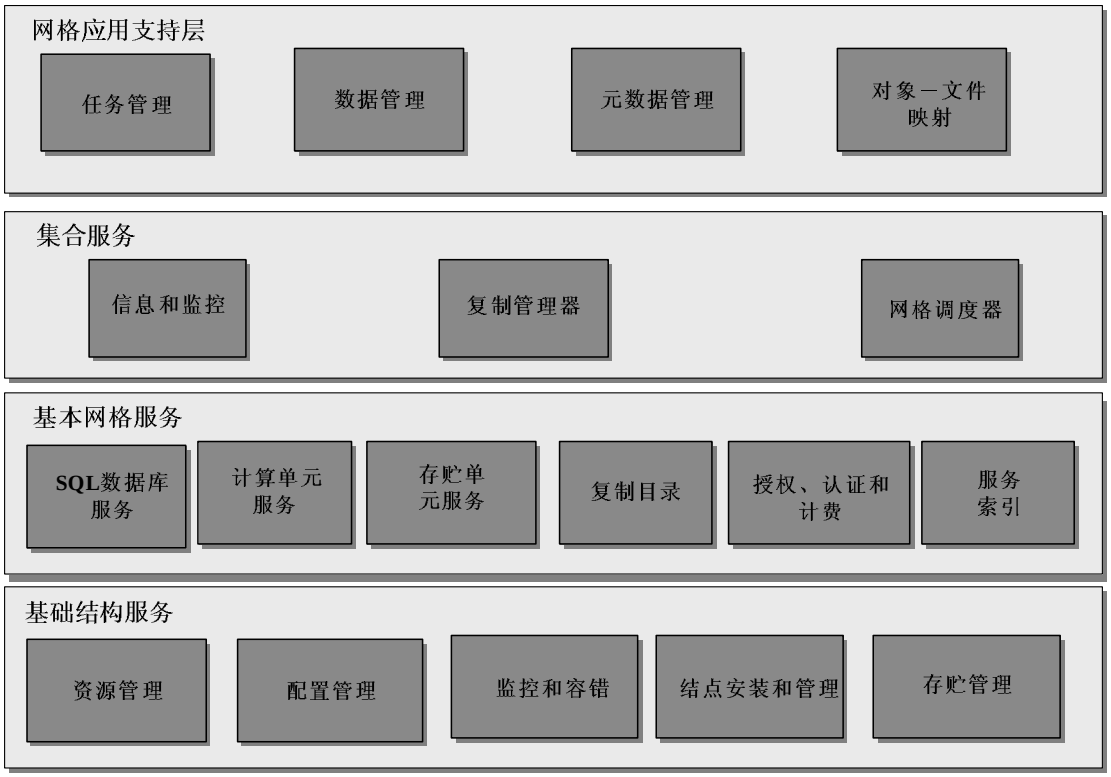


图 12-3　DataGrid 的体系结构

需要指出的是，DataGrid 的体系结构与 Globus 的体系结构息息相关，它的规划需要充分考虑与 Globus 配合的要求；从 Globus 的角度看，它原本是针对计算网格而设计的，但 Globus 项目组好像非常想把它变成一个通用的网格平台——这也是未来的 Globus 3.0 叫做开放网格服务体系结构 OGSA（Open Grid Service Architecture）的原因——为此，Globus 项目组也为让它提供数据网格服务做了许多工作，感兴趣的读者可以参考[52][53][54]。

12.4 DataGrid 的项目管理

欧洲 DataGrid 于 2000 年 12 月 29 日正式立项，由欧盟提供 980 万欧元资金，项目完成期限为 3 年。项目主要完成者除了 CERN 外，还有法国国家科学研究中心 CNRS（French National Centre of Scientific Research）、 欧洲空间研究中心意大利分部 ESA/ESRIN（Centre of the European Spatial Agency in Italy）、意大利国家原子物理研究所 INFN（Italian National Institute of Nuclear Physics）、荷兰国家原子物理和高能研究所 NIKHEF（Dutch National Institute of Nuclear Physics and High Energies）和英国粒子物理和天文研究委员会 PPARC（British Council of Research in Particle Physics and Astronomy）。除了这六家外，还与其他十几家研究机构和工业界建立了合作研究协议[48]。

这样一个多个国家共同参与、规模庞大的网格项目，如何协调不同研究机构的关系，如何保障研究进度，如何简化研究的对象，一定需要有独到的考虑，而这些考虑对于我国正在建设或准备建设的一些网格项目，无疑是有借鉴意义的。

1　明确应用背景

DataGrid 主要针对 CERN 的高能物理应用，解决海量数据的分解存贮和处理问题，同时将之扩展到其他应用，如地球观察应用和生物应用，并寻找将其推广的可能。可以说，CERN 的应用，特别是未来的 LHC 应用，是 DataGrid 的立足之本。解决好这个应用，DataGrid 的研究就可以取得战略性胜利。一句话，有应用背景的项目才有生命力，这点特别值得我们重视。

2　站在巨人的肩上

进行 DataGrid 的研究，有两个选择：一个是从头到尾，完成全新的数据网格中间件，不借助第三方的网格平台；另一个选择是基于 Globus，在其基础上扩展数据网格所特有的中间件代码。虽然这个问题看似简单，但国内的确有些研究人员喜欢从头开始（start from scratch），前些年甚至有人试图重写 TCP/IP 的 IP 层代码——精神可嘉，但未必是好的选择，有这么旺盛的研究精力，应该投向更容易出成果的领域。

DataGrid 毫不犹豫地选择了基于 Globus 平台。试想，Globus 已经完成了 DataGrid 所要解决的 80%的网格问题，这样，DataGrid 就可以专心解决好剩下的 20%的问题，这样成功的把握性要大得多。除了与 Globus 项目组织联合开展研究外，DataGrid 甚至与美国的“竞争对手”GriPhyN[51]建立了合作关系。GriPhyN 是一个与 DataGrid 类似的物理数据网格。

3　分而治之

将复杂任务分解为相对独立的模块，这是软件工程中常用的手段，在 DataGrid 中也不例外： DataGrid 的任务划分成为 12 个工作包 WP（Work Package），分为五组，如表格 12-2 所示。

表格 12-2　DataGrid 开发任务分配

分组	工作包	任务	完成者
中间件	WP1	网格负载管理	INFN
	WP2	网格数据管理	CERN
	WP3	网格监控服务	PPARC
	WP4	构造层管理	CERN
	WP5	海量存贮管理	PPARC
网格构造层试验床	WP6	集成试验床	CNRS
	WP7	网络服务	CNRS
科学应用	WP8	高能物理应用	CERN

	WP9	地球观察应用	ESA
	WP10	生物应用	CNRS
管理	WP11	信息传播	CNR
	WP12	项目管理	CERN

这些工作组之间既相互独立，又相互依存，如图 12-4[49]所示。工作组 WP11 和 WP12 专门负责整体协调，既保障 WP1-10 能够正常运转，又负责与其他研究机构的协作。

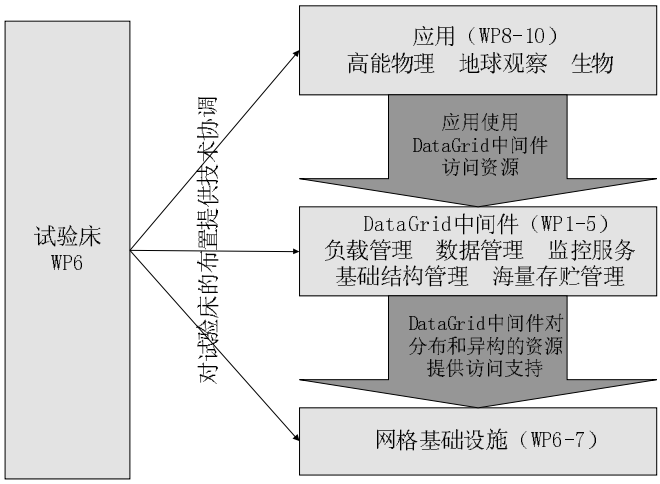


图 12-4　DataGrid 项目工作包的技术配合

4　里程碑式渐近开发

将时间划分成相对独立的阶段，简化每一阶段所要面对的问题，相当于再次将问题简化（如表格 12-3所示）。

表格 12-3　DataGrid 开发计划

里程碑	计划完成日期	任务
试验床 0	2001 初	配置试用 Globus，不开发软件
试验床 1	2002 年 3 月	针对特定用途(WP8-10)的 DataGrid 中间件
试验床 2	2002 年 9 月	在试验床 1 的基础上扩展 DataGrid 设施
试验床 3	2003 年 3 月	
试验床 4	2003 年 9 月	

5　多种协调手段

DataGrid 项目通过一组网站协调不同工作组(WP)之间的研究和进度，并保持与其他研究机构和工业界的同步。这些网站同时也是共享、交流研究成果的平台：每个工作组有各自的主页，可供发布新闻，或将已经开发好的软件提供下载。另外，各个工作组或整个项目经常定期或不定期召开会议，还要按计划在每个阶段向欧盟报告进度和进行演示。

12.5 小结

纯粹的计算密集型问题（例如寻找最大素数）是比较少的，科学研究往往需要基于科学观测的结果进行。换句话说，科学研究的很大一部分，是要处理电子显微镜、天文望远镜、粒子对撞机等各种设备所源源不断产生的大量数据。从这个意义上讲，数据网格应该比计算

网格更重要。但是，正因为如此，数据网格与计算网格是没有截然的分界线的，它们只是侧重点不同罢了。

很大程度上，欧洲的 DataGrid 是为大型强子对撞机 LHC 准备的。CERN 有开展大科学研究的传统，DataGrid 无疑会提供一个更强有力的协同研究平台。

在设计上，DataGrid 立足于 Globus，同时扩展处理海量数据和开展协同研究的能力。在项目实施上，DataGrid 所开展的广泛的国际合作是非常值得我们借鉴的。

思考题

1 CERN 为什么需要建立 DataGrid？
2 DataGrid 准备怎样处理 LHC 产生的海量数据？
3 DataGrid 需要具备哪些主要功能？
4 在管理庞大的 DataGrid 项目时，采用了哪些手段来降低项目实施的复杂度？

第13章 远程沉浸

远程沉浸(Tele-immersion)是一种特殊的网络化虚拟现实环境。这个环境可以是对现实或历史的逼真反映，可以是对高性能计算结果或数据库的可视化，也可以是个纯粹虚构的空间。“沉浸”的意思是人可以完全融入其中：各地的参与者通过网络聚在同一个虚拟空间里，既可以随意漫游，又可以相互沟通，还可以与虚拟环境交互，使之发生改变。打个比方，远程沉浸是一部观众可以进入其中的科幻电影。远程沉浸可以广泛应用于交互式科学可视化、教育、训练、艺术、娱乐、工业设计、信息可视化等许多领域。本章13.1节介绍了远程沉浸的来历，13.2节介绍了它的应用一些。从实现上看，虽然没有网格的支持，照样可以开发远程沉浸应用，但有了网格的支持，实现这类应用会简单得多，所以13.3节重点讨论了远程沉浸与网格的结合问题。

13.1 背景

远程沉浸这个术语是在 1996 年 10 月，由伊利诺州大学芝加哥分校的电子可视化实验室 EVL (Electronic Visualization Laboratory）最早提出来的[55]。远程沉浸建立在高速网络基础上，是协同可视化环境 CVE (collaborative virtual environments)、音频、视频会议以及超级计算机及海量数据存贮的有机融合[56]。远程沉浸使分布在各地的使用者能够在相同的虚拟空间协同工作，就像是在同一个房间一样[57]，甚至可以将虚拟环境扩展到全球范围内，创造出“比亲自到那儿还要好”的环境[55]。EVL 的负责人 Tom DeFanti 预测远程沉浸将成为计算网格的关键应用之一[59]。顺便说一句，Tom DeFanti 是著名的计算机图像专家，早在 1977 年就参与过第一部《星球大战》中特技效果的制作，曾获 ACM 杰出贡献奖。

远程沉浸与一些研究人员所提到的协同虚拟环境 CVE (Collaborative Virtual Environment) 或分布式虚拟环境 DVE (Distributed Virtual Environment)是不同的，它使用了更多视频和图像技术，所创造的虚拟环境更为逼真[55]。更重要的是，它将“人/机交互”模式扩展成为“人/机/人协作”模式，不仅提供协同环境，还将对数据库的实时访问、数据挖掘、高性能计算等集成了进来，为科技工作者提供了一种崭新的协同研究模式[58] 。

远程沉浸的基本构想，最早可以追溯到 1995 年的 I-WAY 项目[59]。I-WAY 为 Information Wide Area Year 的缩写，它是一个为期一年的研究计划，试验用异步转移模式 ATM (asynchronous transfer mode) 宽带网络连接全国范围内的超级计算机中心、虚拟现实研究机构和应用开发机构，在上面试验了约 60 个应用项目[60]。在圣地亚哥举行的 Supercomputing 95 年会上，I-WAY 对其应用项目作了精彩展示，其中，那些将超级计算机与虚拟现实设备相结合、用共享的虚拟空间推动计算科学协同研究的项目尤其引人注目。虽然相比现在的远程沉浸环境而言，I-WAY 还只算是个原型，它的网络服务质量没有保障，带宽不够充足，对人机交互的平台支持不够，但 I-WAY 开创了一个崭新的局面，之后有 50 多家研究团体沿着 I-WAY 所指明的方向，继续研究新的软硬件平台、虚拟现实技术和应用[59]。

在 Supercomputing 95 上，有一种虚拟现实设备大放异彩，它的名字叫 CAVE 自动虚拟环境(CAVE Automatic Virtual Environment)，是由 EVL 的Carolina Cruz-Neira、Daniel J. Sandin 和 Tom DeFanti 于 1991 年共同提出的[61]，如图 13-1[60]所示。

图 13-1　CAVE 虚拟现实环境

CAVE 是一个 10×10×9 英尺的小房间，它的四面和顶上使用高清晰背投产生一个 360 度的三维立体环境，在该场景中有一个与该人对应的卡通角色(avatar)，它代表真实的人在三维空间活动。人置身其中，头戴一副 LCD 快门眼镜，两臂绑有电磁跟踪系统，再配以三维环境声音，角色的位置、方向以及举手投足都与真人相同，会让人产生非常逼真的临场感[60]。

13.2 应用举例

过去几年里，EVL 与十几家合作伙伴一起，开发了一些具有远程沉浸特征的虚拟现实应用[62]，这些应用的典型特征是[58]：

1　它们依据科学计算结果和数据库构造虚拟环境，当参加者沉浸在虚拟环境中时，他们不仅能够看见对方，还能用做手势或说话进行交互，当他们离开时，虚拟环境本身可以随着科学计算和数据的变化仍然存在并继续演化；

2　人在虚拟环境中以角色代表，这个角色比人在真实环境中的能力要强得多：它们能够站在分子链可视化图像的某个环节前进行讨论；也能钻进设计好的汽车模型里，体验汽车是否舒适；还能站在云层的顶端，观察风暴的形成。他们既可以让自身像星系一样大，以便纵览宇宙风云；也可以让自己像虫子一样小，以便钻进一个做跌落试验的物品中，并与物品一起跌落，找到防止物品在跌落过程中损坏的办法。

3　人与虚拟环境是交互的，可以通过改变超级计算的参数等手段，动态改变虚拟环境的形成方式。

本节将介绍几个典型的远程沉浸应用。

13.2.1 虚拟历史博物馆

共享的爱奥尼亚(Shared Miletus)项目[63]让2000年前的希腊爱奥尼亚古城复活，参观者可以在虚拟的城池中畅游，如图 13-2[64]所示。

图 13-2　参观者进入虚拟希腊古城

该项目由希腊世界基金会FHW(Foundation of the Hellenic World)与CVL共同完成。FHW是一个非盈利组织，其任务是保存和再现希腊历史和文化[65]。最初，FHW已将爱奥尼亚古城建模，并用ImmersaDesk[66]和ReaCTOR虚拟环境在博物馆里展示出来，参观者可以在城市上空“飞行”或进入殿堂参观。CVL的加入是为了让该环境体现远程沉浸的特征，让参观者也可以从网络上进行参观：所有来自不同地方的人进入同一个虚拟环境，他们互相之间可以自由交流。为了方便浏览，既可以由真人担任虚拟环境中的向导，也可以指定一个虚拟的向导，“他”能够带领大家到不同地点参观，会说多国语言，还可以根据听众的要求将解说简化或细化[64]。

除了希腊古城外，中国敦煌莫高窟价值连城的壁画也被数字化并用在远程沉浸环境中[67]，如图 13-3[68]所示。之所以选择它，是因为敦煌被认为是亚欧丝绸之路的起点，而敦煌石窟是人类历史上不可多得的珍宝。该项目于1998年完成，是历史学家、艺术家和计算机科学家协同研究的结晶。

图 13-3　虚拟的敦煌莫高窟

13.2.2 协同学习环境 NICE

备受关注的 NICE——叙事式沉浸的建设者及协同环境 (Narrative Immersive Constructionist / Collaborative Environments)——是伊利诺州大学芝加哥分校电子可视化实验室 EVL 和交互计算环境实验室 CEL(Interactive Computing Environments Laboratory)合作推出的。NICE 是一个探索性质的协同学习环境，专为 6 至 10 岁孩子设计[70]。

图 13-4　一个儿童正在 NICE 虚拟花园中活动

孩子们进入 CAVE 虚拟现实环境中（如图 13-4[70]所示），戴着特制的眼镜，手中拿着一个特制的小棒用于交互操作，每一个孩子都有一个角色，来自不同地方的孩子可以在这个虚拟的花园中打招呼、交谈、互相学习。他们既可以将鲜花或蔬菜拿在手上观察，又可以将自己缩小，“深入”观察植物的根系；他们既可以除去杂草并将它扔到肥料堆，又可以顺手摘下天边飘过的雨云给花儿浇灌，或者将天边的太阳拉过来让花朵享受阳光，等等。除此之外，这些植物本身也会有一些反馈信息，例如，当给花浇水太多时，花儿会撑起一把小伞，让孩子们得到一个直接的反馈——过多浇水会让花儿受不了。NICE 有趣又有益，孩子们不仅非常喜欢在里边玩耍，还可以轻松地了解原本很复杂的生态关系[69]。

13.2.3 数据可视化协同分析环境 CAVE6D

CAVE6D[73]是 CAVE5D 与远程沉浸环境 CAVERNsort 相结合的产物。

CAVE5D[72]由弗吉尼亚大学 ODU(Old Dominion University)和威斯康星－麦迪威大学 WISC(University of Wisconsin-Madison)联合研制的一个可配置虚拟现实应用框架，它是在 Vis5D[71]的支持下运作的。Vis5D 是一个强大的图形库，能够提供显示三维数字数据的可视化支持，广泛应用于大气、海洋及其他类似模型的可视化中[75]。

CAVERNsort[74]是一个用来建造协作式网络应用的开放源码平台，主要为高吞吐量的协作应用（不一定是 CAVE 应用）提供网络支持。另外，它还提供了建造远程沉浸应用的专门模块。

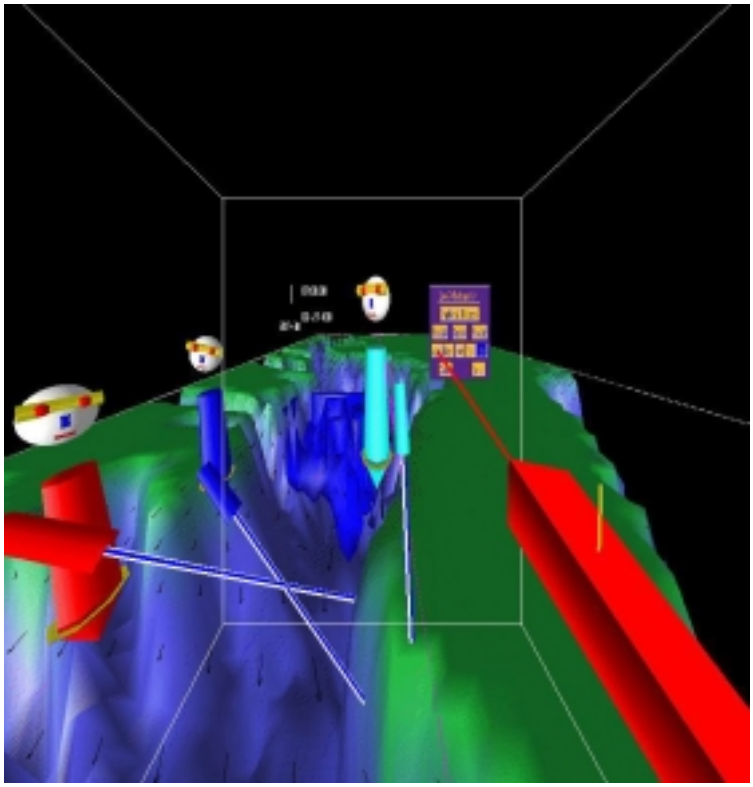


图 13-5　　几个参与者就可视化图像的某一区域进行讨论

由于 CAVERNsoft 的支持，CAVE6D 变成了一个远程沉浸环境，如图 13-5所示。它允许多个用户在虚拟的环境里对超级计算数据进行可视化并与数据进行交互。参与者们不仅以角色的方式进入三维可视化场景中自由漫游，还可以改变可视化参数，如循环矢量、温度、风的速度、鱼群的分布等[75] 。CAVE6D 不仅提供了交互式的可视化手段，还提供了让各地参与者协同交流和研究的手段[76]。

13.3 远程沉浸与网格的结合

从网格的角度来看，远程沉浸算得上是个“另类”，它与分布式超级计算、分布式仪器系统和数据密集型计算的性质有很大的不同：一方面，虽然它能把高性能计算的结果用一种全新的方式可视化出来，但它本身对高性能计算的要求并不高；另一方面，它并没有集网络上的大量资源于一身。那么，远程沉浸问题到底还算不算网格问题呢？

让我们回到网格的定义上来：动态多机构虚拟组织中的资源共享和协同问题解决(Resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organizations.[77])。显然，参与远程沉浸应用的各方是来自不同的虚拟组织的，他们所要共享的资源就是这个共同的虚拟环境，而共享的目的就是要协同解决问题，因此，远程沉浸还

是一个典型的网格问题，只不过它所共享的是一个集中的虚拟环境（这个虚拟环境是所共享资源的映象，如超级计算机、数据库等），而主要不是共享参与者的资源罢了。

更进一步，网格是未来 Internet 的发展方向，也就是说，网格会成为未来的网络基础设施，几乎所有的应用都将基于网格，因而极富创新精神、有广阔发展空间的远程沉浸应用，也会毫无例外地依赖于网格的支持。另外，远程沉浸以崭新的手段极大地促进了网络化协作，这也正是网格的精髓所在。

由于远程沉浸与其他网格应用有很大的不同，它对网格有比较特殊的需求[59]：

1　服务质量(QoS)机制

远程沉浸对服务质量的要求主要有三个方面：(1) 提供最低服务保障，并能为每一个数据流指定相对优先级；(2) 如果服务质量因为网络的动态变化而出现改变，能够通知服务接受方，并给出弥补建议；(3) 能够评估服务质量预测的可靠性。

2　网格编程环境

远程沉浸应用将大大依赖于网格分布式计算编程环境的支持：(1) 远程沉浸应用需要查找或发现资源，如远程沉浸服务器、计算服务器、数据库、仪器等，因此网格的资源定位器和资源目录将大有用场；(2) 需要有编程工具使远程沉浸应用能够与复杂的服务质量体系相结合，并能监视网格资源的实际使用情况，如网格带宽、计算能力、数据等；(3) 需要有分布式资源管理工具和调度工具的支持，使远程沉浸应用能够用一种抽象的方法指定资源，例如，用类型指定，而不是用名字指定。

3　先进的网络协议

远程沉浸应用需要不断扩展用户容量，以便让更多用户加入进来，但是如果用户数增加得太多，数据传输量将出现组合爆炸，这时就需要有革新的网络协议，例如，现有的 Internet 尚不能很好支持多播（multicast），如果网格能够提供这样的功能，应用程序的能力将大大增强。

4　性能监控和度量

当前的 Internet 不太可能给出端到端通信性能下降的原因，更不能预测某个分布式应用将能达到的网络性能，而远程沉浸应用是非常需要知道这些信息的——不仅包括某个数据流的性能，还包括整个应用的性能。所以，网格平台必须具有监控和度量网络性能的能力。

5　资源调度

资源调度对于远程沉浸应用是非常关键的，因为它不仅要满足整个应用的资源需求，还需要实时完成调度。在很多情况下，需要有一些特别的调度策略，如交互式调度和资源预留服务。

13.4 小结

远程沉浸的雏形形成于 I-WAY 计划期间，后者既提供了充足的网络带宽，又提倡将科学计算与虚拟现实相结合。尔后，EVL 成为这个领域的领跑者，它不仅提出了远程沉浸这个概念，还完成了一批具有说服力的远程沉浸应用。

远程沉浸与其他的网格应用有很大的不同，因而它对网格平台提出了不同的要求。网格如果要成为下一代 Internet 的基础平台，一定会不断满足各种应用的需求，包括远程沉浸。

思考题

1 远程沉浸应用的典型特征是什么？
2 远程沉浸为什么需要与网格相结合？
3 远程沉浸对网格提出了什么样的要求？

参考文献

[1] Gabrielle Allen, Tom Goodale, Gerd Lanfermann, Thomas Radke, Edward Seidel, Werner Benger, Hans-Christian Hege, Andre Merzky, Joan Massó and John Shalf, Solving Einstein's Equations on Supercomputers, IEEE Computer, 32, (1999) [cover story]

[2] Gabrielle Allen, Thomas Dramlitsch, Ian Foster, Nick Karonis, Matei Ripeanu, Ed Seidel, Brian Toonen, Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus, Supercomputing 2001. [Winning Paper for Gordon Bell Prize (Special Category)]

[3] Gabrielle Allen, Edward Seidel, John Shalf, Scientific Computing on the Grid, Byte, Spring 2002

[4] I. Foster and C. Kesselman, editors, The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1998. Chapter 3

[5] Sharon Brunett and Steven Fitzgerald, Metacomputing Supports Large-Scale Distributed Simulations, May 1998, http://www.cacr.caltech.edu/SFExpress/pubs/sc98/sc98.html

[6] Sharon Brunett and Steven Fitzgerald, Balancing the Load in Large-scale Distributed Entity-level Simulations, May 1998, http://www.cacr.caltech.edu/Publications/techpubs/cacr163.html

[7] http://www.cacr.caltech.edu/SFExpress/

[8] http://www.cactuscode.org

[9] Gabrielle Allen, Cactus Grid Computing, http://www.cactuscode.org/Presentations/Europar_August01.ppt

[10] Thomas Dramlitsch, Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus http://www.cactuscode.org/Presentations/SC2001_November01.ppt

[11] DOE NGI Application Information, http://www.itg.lbl.gov/NGI/templates/RRngi_apps.html

[12] DOE NGI Testbeds Project, http://www-itg.lbl.gov/NGI/

[13] Donald McMullen, Randall Bramley, et al. The Xport Collaboratory for High-Brilliance X-ray Crystallography, http://www.cs.indiana.edu/ngi/sc2000/index.html

[14] http://www.cs.indiana.edu/ngi/

[15] Dennis Gannon, Distributed Problem Solving Environment, collaborative workbenches/frameworks and Portals: how they will be used in the future? http://www-itg.lbl.gov/DOE_Security_Research/WorkshopIV/presentations/DennisGannon.ppt

[16] http://www.aps.anl.gov/

[17] http://www-als.lbl.gov/

[18] http://www.iumsc.indiana.edu/

[19] Donald McMullen, Randall Bramley, et al. Xport Collaboratory for X-ray Crystallography, http://www.iumsc.indiana.edu/XPort/xport_demo_talk.pdf

[20] CoG Kit Homepage, http://www.globus.org/cog/
[21] Science Portals Project , http://www.extreme.indiana.edu/an/
[22] The DOE Common Component Architecture Project, http://www.extreme.indiana.edu/~gannon/cca_report.html
[23] The Common Component Architecture Toolkit Project, http://www.extreme.indiana.edu/ccat/
[24] The XCAT Project, http://www.extreme.indiana.edu/xcat/
[25] SF-Express Application, http://www.globus.org/research/applications/sfexpress.html
[26] HPSS WorldWide Web Site, http://www.sdsc.edu/hpss/
[27] http://go4.163.com/xxys/ycyl/ycyl.htm
[28] http://www.netdoctor.com.cn/telemed/index.php?ps_operate=showarticle&pi_articleID=66
[29] http://www.imaging-nsmc.com.cn/telmedicine/tel.htm
[30] http://www.netdoctor.com.cn/telemed/index.php?ps_operate=showarticle&pi_articleID=66
[31] Cambridge eScience Centre, http://www.escience.cam.ac.uk/projects/telemed.html
[32] I. Foster and C. Kesselman, editors, The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1998. Chapter 4
[33] http://www.sdsc.edu/hpss/HPSS_main.html
[34] http://www.sdsc.edu/hpss/Papers/fact_sheet.html
[35] http://www.sdsc.edu/hpss/HPSS_objectives.html
[36] http://www.sdsc.edu/hpss/hpss1.html
[37] Gateway to CERN, http://welcome.cern.ch/welcome/gateway.html
[38] Panel Discussion: Data Intensive vs. Scientific Computing: Will the Twain Meet for Parallel Processing? http://ipdps.eece.unm.edu/1998/papers/kumar.pdf
[39] DataGrid Official Project Presentation, http://150.146.1.82/grid/internal/Deliverables/D11.1/DataGrid-11-PRE-0103-1_1-ProjectPresentation.htm
[40] The DataGrid Project, http://www.datagrid.cnr.it
[41] NPACI: A National Partnership, http://www.npaci.edu/
[42] R.W. Moore, T.A. Prince, and M. Ellisman. Data-Intensive Computing and Digital Libraries. Communications of the ACM, Vol. 41(11):pages 56--62, November 1998.
[43] CERN in 2 minutes, http://public.web.cern.ch/Public/whatiscern.html
[44] The World Wide Web, http://public.web.cern.ch/Public/ACHIEVEMENTS/web.html
[45] Ben Segal, Grid Computing: The European Data Project, IEEE Nuclear Science Symposium and Medical Imaging Conference, Lyon, 15-20 October 2000.
[46] Ben Segal, The European DataGrid project, First Presentation At Data Mining Workshop-CSC Scientific Computing - Otaniemi 05-APR-01, http://web.datagrid.cnr.it/pls/portal30/docs/902.PDF
[47] GRIDS Center: Grid Research Integration Development & Support, http://www.grids-center.org/Net2002%20Talk%204-18-02.ppt
[48] First results and future perspectives of the European DataGrid project, http://www.hoise.com/primeur/02/articles/weekly/AE-PR-04-02-22.html
[49] Project Description,

http://web.datagrid.cnr.it/servlet/page?_pageid=873,875&_dad=portal30&_schema=PORTAL30&_mode=3

[50] Robert Jones, Status of EU DataGrid project and test-bed 1, First Presentation At 2nd NorduGrid, Oslo, Norway 01-NOV-01, http://web.datagrid.cnr.it/pls/portal30/docs/2048.PPT

[51] Grid Physics Network, http://www.griphyn.org

[52] S. Vazhkudai, S. Tuecke, I. Foster. Replica Selection in the Globus Data Grid. Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001), pp. 106-113, IEEE Computer Society Press, May 2001.

[53] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, B. Tierney. File and Object Replication in Data Grids. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[54] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. Journal of Network and Computer Applications, 23:187-200, 2001.

[55] Jason Leigh, Thomas A. DeFanti, Andrew E. Johnson, Maxine D. Brown, and Daniel J. Sandin, Global Tele-Immersion: Better Than Being There, appeared in proceedings of 7th International Conference on Artificial Reality and Tele-Existence. Tokyo, Japan, Dec 3-5, 1997, Pp 10-17

[56] Johnson, A. E., Leigh, J., and DeFanti T., "Multi-Disciplinary Experiences with CAVERNsoft Tele-Immersive Applications," Proc. of Fourth International Conference on Virtual System and Multimedia, November 1998, pp. 498-503.

[57] Kyoung S. Park, Yong J. Cho, Naveen K. Krishnaprasad, Chris Scharver, Michael J. Lewis, Jason Leigh, Andrew E. Johnson, CAVERNsoft G2: A Toolkit for High Performance Tele-Immersive Collaboration, Proc. of the Symposium on Virtual Reality Software and Technology 2000, Oct 22-25, 2000, Seoul, Korea

[58] Maxine D. Brown, Global Tele-Immersion: Working in CyberSpace, http://www.pdc.kth.se/conference/1999/Content/abstracts.html#maxine

[59] I. Foster and C. Kesselman, editors, The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1998. Chapter 6

[60] T. DeFanti, I. Foster, M. E. Papka, R. Stevens, and T. Kuhfuss, Overview of the I-WAY: Wide Area Visual Supercomputing, International Journal of Supercomputing Applications, 10(2), 1996.

[61] http://www.pdc.kth.se/conference/1999/Content/speakers.html#tom

[62] http://www.evl.uic.edu/research/telei.html

[63] Shared Miletus, http://www.fhw.gr/fhw/en/projects/3d/miletus/

[64] http://www.evl.uic.edu/research/template_res_project.php3?indi=138

[65] D. Pape, Anstey, J., D'Souza, S., DeFanti, T., Roussou, M., Gaitatzes, A., Shared Miletus: Towards a Networked Virtual History Museum, Proceedings of the International Conference on Augmented, Virtual Environments and Three-Dimensional Imaging (ICAV3D), Mykonos, Greece, 05/30/01-06/01/01

[66] Immersadesk (tm), http://www.evl.uic.edu/pape/CAVE/idesk/

[67] http://www.evl.uic.edu/samt/silkshrine/mogao.html

[68] http://www.evl.uic.edu/research/template_res_project.php3?indi=132

[69] Maria Roussos, Andrew E. Johnson, Jason Leigh, Christina A.Vasilakis, Craig R. Barnes, and Thomas G. Moher, NICE: Combining Constructionism, Narrative, and Collaboration in a Virtual Learning Environment, Computer Graphics, Vol. 31 Num. 3 August 1997, pp. 62-63, ACM SIGGRAPH.

[70] Johnson, A., Roussos, M., Leigh, J., Barnes, C., Vasilakis, C., Moher, T., The NICE Project: Learning Together in a Virtual World,. In the proceedings of VRAIS '98, Atlanta, Georgia, March 14-18, 1998, pp.176-183.

[71] Vis5D Home Page, http://www.ssec.wisc.edu/~billh/vis5d.html

[72] Cave5D Home Page, http://www.ccpo.odu.edu/~cave5d/homepage.html

[73] Cave6D, http://www.evl.uic.edu/akapoor/cave6d/

[74] Open Channel Foundation: CAVERNsoft G2, http://www.openchannelsoftware.org/projects/CAVERNsoft_G2/

[75] http://www.evl.uic.edu/research/template_res_project.php3?indi=21

[76] http://www.evl.uic.edu/akapoor/cave6d/desc.html#links

[77] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001.

[78] Awards Cap SC2001 HPC and Networking Conference, http://www.npaci.edu/online/v5.24/sc2001.awards.html