网格实例与关键技术分析

杜 江 周念生 刘 然 王小鸽 都志辉
(清华大学计算机系 北京 100084)

Grid Example and the Analysis of Key Technology

DU Jiang ZHOU Nian-Sheng LIU Ran WANG Xiao-Ge DU Zhi-Hui
(Department of Computer, Tsinghua University, Beijing 100084)

Abstract Grid computing technology is developing in an impressing speed these years. This paper is going to do some detailed analysis on two of the major Grid projects, Globus and Legion, in order to find out the key technologies of Grid computing. In this paper, three key technologies will be discussed: Grid Architecture, Grid Data and Resource Services, and Grid Security Services. Different implementation of these technologies will be discussed and compared with each other, some new ideas about these technologies will be proposed as well.

Keywords Grid computing, Grid architecture, Data grid components, Resource management, Grid data and resource services, Grid security services

过去的几年中,随着网格计算的出现和网格技术的成熟应用,很多现成的网格工具能够支持在局域网甚至广域网上搭建网格,众多的客户端和服务器的计算资源可以被这些网格整合起来,以实现安全、高效、大吞吐量的并行大规模计算。

本文将对两个主要的网格项目实例——Globus 和 Legion——所使用的技术进行分析和讨论;并试图通过这些讨论,总结出在构建一个网格时所需要的几类关键性技术,同时对这些技术的各种可能的实现方式进行评价和展望。

下面将分 3 个部分对各类网格技术进行论述,第 1 部分(第 1 节)将着重分析现有的网格体系结构的构架,第 2 部分(第 2,3 节)则会讨论网格数据及资源服务的有关内容,关于网格安全服务的实现技术将在第 3 部分(第 4 节)详细叙述。最后将在前面论述的基础上,对这类网格技术的可能发展提出一些新的建议和看法。

1 网格体系结构(Grid Architecture)

1.1 网格体系结构的大体分类

在搭建一些基础构件进行网格计算之前,首先需要考虑的是网格上这台虚拟计算机的体系结构问题。按照核心的构建思想来划分,网格系统的体系结构可以分为开放式和封闭式两种,而这两者最明显的区别就在于它们实现的基础不一样[1]。

对封闭式的网格系统来说,下层资源的整合和调整对上层的用户是不透明的,从上往下看就是一个个封装好的服务。用户可以通过它们声明的外部变量来访问或调用它们,而不用关心它们的内部实现。然而这样的易用性是有代价的,因为一个封装好的服务很难分散地分布,所以封闭式的网格由重权(heavyweight)服务器组成,每个服务器都要提供至少一个完整服务的所有资源。而开放式的网格系统则恰好相反,下层的资源对上层的调用是透明的,这就是说在这种网格系统上提供服务的程序员必须了解下层资源的整合和调整情况。在开放式的网格系统里,服务器是轻权(lightweight)的,可以由多个服务器共同承担一个完整服务所需的资源,甚至还能互为备份,这样可以降低对服务器的要求,提高系统的稳定性。然而很明显,在开放系统上维持一个服务必须要和最底层的资源打交道,还要考虑到系统的实时变化,这些都是相当繁琐的工作。

另外,实用的系统除了要能提供基本的服务以外,还要有良好的可扩展性以支持不同时期的不同应用需求。两种不同的系统结构在添加新的服务以扩展功能的过程中所采取的策略也各有特色。

封闭系统由于其封装性,每一层对外都不透明,通过固定的应用接口(API)很难甚至有时不可能从上往下进行功能扩展。因此,很多封闭系统采取了"插件"的形式,由系统程序员根据特定的需求从下往上开发一个个完整的功能模块提供相应的服务。这也使得封闭系统可以被"分割"为很多小模块,而这些模块间都可以相互独立地在网格上提供服务,不同的应用系统实质上就是不同模块的集合。

在开放系统上添加服务或者增加应用接口都相对容易得多,因为程序员可以很容易地了解到下层的结构并且能够在相应的地方做出适当的修改,这样从上往下的开发就很方便。但是在开放系统中,开发出来的服务的移植性很差,为适应不同的基层构件,常常需要对整个系统做重大的调整;而且应用层次越高,开发的工作就越复杂,导致很高的成本。

1.2 网格体系结构举例

当然,实际的网格系统并不是严格划分的,往往一些封闭的网格系统也在尽力实现分布式的轻权服务器模式;而开放的网格系统为降低开发的成本也有统一的规范和越来越明显的层次。下面介绍几个具体的网格系统的实例。

1.2.1 Legion 体系结构 Legion 的目标是进行高速(十亿,甚至万亿次)的大规模(全美国的,甚至覆盖整个因特网)的并行计算。为此 Legion 开发出一个网格计算的平台,将互联网上孤立的计算机或者封闭的工作站整合起来,形成一个拥有庞大计算资源的虚拟计算机,同时满足成千上万用户的不同需求[3]。

Legion 在实现网格平台时,力图将它设计成一个封闭的系统,虚拟的计算机除了向用户提供强大的分布并行运算能力外,还要对底层的操作系统有很好的兼容性,让用户在使用网格系统时像管理本地资源一样方便自如。

分析 Legion 的系统结构,可以划分为非常明显的三个层次:最核心的底层,资源管理的中间层和面向应用的上层。这种三层的结构在分布式系统中是很典型的,它对应网格上三个不同的物理层面,分别完成网格系统的基本功能,如图 1 所示[2]。

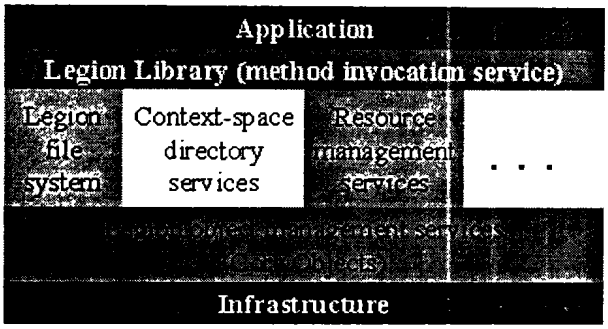


图 1 Legion 系统结构

其中底层提供的是网格系统的核心功能，在 Legion 中被称为“元系统”。它位于网格服务器的物理资源和网格应用之间，负责将网格上分布的资源整合到一个虚拟的系统中，跨操作平台实现一个自主的系统。这一层向上提供的是一些系列类似系统调用的核心服务，使其上层的系统在此基础上实现物理无关的功能级服务。

中间层包含了一系列底层的系统服务调用的入口和描述。在这一层中网格系统通过管理和共享系统调用实现了基本的网格功能，比如文件的传输和资源管理等。同时这一层还向上提供这些功能级服务的调用接口，使上层的应用可以直接调用这些网格服务。

上层是最接近用户的接口层。在这一层中，网格系统通过接口和用户交互，获取用户的需求并将它分解成可并行的多种网格服务，通过下层的接口调用特定的网格功能模块，实现大规模的分布运行。而网格系统所有内部的实现机制在这一层都被封装起来，用户通过标准的接口就像使用本地的操作系统一样来提交任务需求而感知不到虚拟计算机内部复杂的调度管理和实际上千差万别的硬件系统。

Legion 通过这样的三层结构，逐级将分布、异构、共享的网格资源封装成统一的服务接口，实现了一个虚拟的并行的而且与平台无关的网格计算系统。而实际上这种体系结构正是当前被最广泛采用的一种结构，除了 Legion 外，很多著名的网格计算项目也都是基于类似的结构搭建的，只是每一层的具体功能在实现上可能略有差异并且根据不同的应用需求采取了不同的策略。

1.2.2 Globus 体系结构 Globus 2.0 提出了一个“虚拟对象”的概念，即使用一套网格计算的软件环境，将大规模的计算拆分成虚拟的对象，使其可以分布、跨平台地在一个虚拟的计算机上进行[5]。

实现这个软件环境的核心是一套完备的网格系统的“工具集”。工具集是由很多的“工具”，实际就是能完成一定功能的模块组成。Globus 中每个模块都能相对独立地提供一个或多个网格服务用以实现对网格资源的整合和基本的管理。同时 Globus 2.0 还定义了一系列标准的应用接口，使用户很容易利用预制的模块搭建出所需的网格系统而不必特意关心其内部的实现。Globus 在封装后的易用性和它提供的强大的功能模块使它很受工程人员的欢迎，在大规模并行计算和仿真工程中都被广泛应用。

深入 Globus 的内部，可以发现它的功能模块实际上分四个层次来实现，这其中就包括：构造层、连接层、资源层和汇聚层[4]，而每个层次都负责具体的功能并且在封装后都有着规范的应用接口，如图 2 所示[5]。

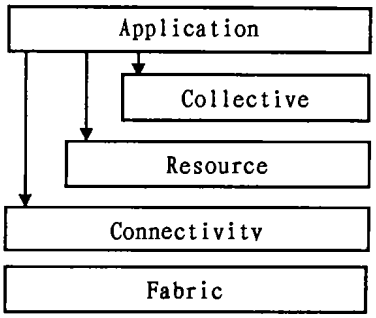


图 2 Globus 系统结构

构造层是 Globus 中最基本的结构，管理着本地服务器上的程序运行和数据交换等工作。在这个层面上，Globus 通过标准的文件协议将不同的底层的系统资源都整合到统一的 APIs 中，封装以后就实现了基本的平台无关性。

连接层紧贴在构造层的上面，一方面利用构造层的 APIs 控制本地资源，另一方面，又通过一系列网络协议(IP，DNS，routing，等)将各个孤立的服务器联系起来，实现了资源的简单共享。同时，通过协议的传输、授权、认证，在这一层上 Globus 还解决了大部分的系统安全问题。

资源层上，Globus 关注的是资源的分配和管理，包括资源的获取、传输和分布储存都在这里得到解决。在这一层上，Globus 针对资源的不同操作，采取了一系列算法和策略，力图实现网络资源高效安全的共享。

汇聚层中的服务建立在下层资源的共享基础上，是由各种资源根据应用的需求协调同步来完成一个定制功能的模块。因此，这一层上的资源实际上是“复合”的资源，虽然对应用来说它们具有资源的共享性和封装性，但实际上在 Globus 内部，这些资源还可以细化并分布在不同的服务器上。

通过把不同的功能模块封装到明晰的内部层次中，Globus 2.0 实现了面向协议的内部结构，就是在不同的层次上，分布的服务器通过不同的协议来实现相互协调，共同解决系统内的问题。其中关键的协议包括：连接层中实现的网格安全机制(GSI)；资源层中应用的网格资源分配管理(GRAM)、网格资源描述协议(GRIP)以及网格文件传输协议(Grid-FTP)等等。

2 网格数据服务(Data Grid Components)

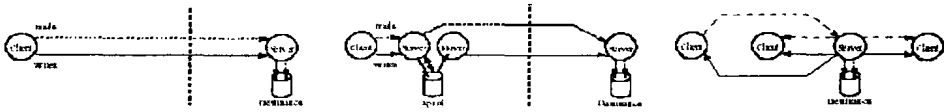
在网格系统中，作为重要的网格资源，对于网格数据的服务是必不可少的。通常这类服务利用底层的系统调用，根据不同的上层应用需求采取相应的策略，以完成对大规模分布式的网格数据进行高效安全的查找、传输和存储等操作。

2.1 网格数据传输

在网格系统中，各种各样的数据资源需要在不同的用户和不同的底层系统间进行快速的传递，为此设计一些专门应用于传输数据的服务在系统中就显得尤为重要。一般情况下，网络中的数据元按大小可以分为字节流、XML 文档和网络文件等级别，因此不同的传输服务可以根据具体的应用需求选择在不同的层次上支持数据元的分布共享和高效传输。比如 ATM 系统中用到的数据处理子系统(DPSS)，Globus 采用的 Grid-FTP 协议，NCSA 系统采用的虚拟文件传送器等等，都是各个项目为了保障级别的网格数据的传输而制定的一套实用的传输协议。

在实际应用中，大多数面向大规模网格计算的系统为了

寻求异构性和高效性的平衡更倾向于采取规模适中而性能较好的 XML 文档作为传输的数据元。这主要是因为，XML 文档本身在封装后保障了平台无关性，对它的操作不用再像字符串操作一样需要考虑下层硬件系统的功能实现。另外受限于网络的带宽，传递可伸缩的 XML 文档又比传递整个的网络文件更快捷更安全，同时还能更好地满足分布数据传输的要求。在 XML 的基础上，实用的网格数据传输机制还需要实现系统中的服务器间的直接传输、第三方中转的传输和广播式的传输等功能，从而支持网络的各种拓扑结构和数据的大范围内分散布局(图 3)[6]。



I. 直接传输　　II. 第三方中转传输　　III. 广播式传输

图 3　网格中的数据传输方式

其中直接传输的方式多应用于客户端和服务器的直接通信，传输的速度快效率高，但对带宽的要求也高，因此一般数据流量不太大而且实时性强的控制数据多采用这种传输方式。

第三方中转的传输是网格数据传输中最常见也是性能较好的一种方式。在传递数据的服务器之间插入第三方的中转服务器提供数据的缓存和断点续传的功能，这样可以在一定程度上容忍带宽的限制，即使中间某环节的网络临时被阻塞也不会影响整个传递任务的完成，从而保证单向的数据传输更流畅也更安全。

广播式的数据传输是一种更贴近应用的数据服务，尤其针对网格上分散的存储空间，这种一对多的数据传输在数据的分布存储中有非常实际的应用价值。其次，高层的网格服务中也常常调用这类的数据传输，用以广播通知和多个并行进程间的相互通信等。

2.2　网格数据存储

网格数据的存储也是数据服务的重要功能之一。对于分布式的网格，为了提高存储和调用过程中的并行度或者基于数据安全和容错的考虑，数据资源的存储和备份被分散在网格的不同节点服务器中，为此实用的网格系统在考虑网格系统的数据服务时，往往要制定一些相关的策略来管理数据资源的存储。下面以 Condor 的数据动态分配存储为例，简要介绍网格数据存储策略的设计规范和实际应用[6]。

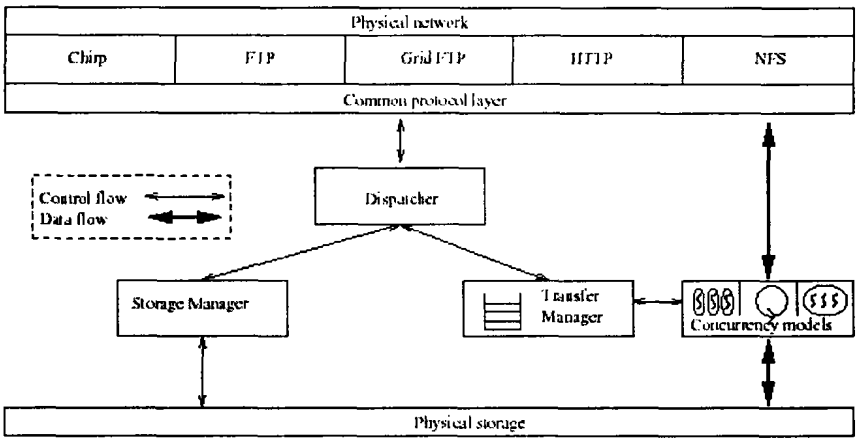


图 4　网格数据存储过程

在很多网格系统中，数据的动态分配存储都是必要的存储策略，它主要针对网格中的数据资源的灵活性和多样性采取尽量低的代价来保证数据的安全有效的存储。在 Condor 的设计中数据的存储过程由分派器、传输管理器和存储管理器共同来完成。如图 4[6]所示，当网格数据通过不同的协议传输后，由分派器根据数据类型和规模选择合适的一个或多个网格上的存储空间，然后通知当地的传输管理器和存储管理器，由后两者协同决定网格数据的存储速度、粒度和物理存储地址。以后再调用这些数据时，又经历逆向的过程由分派器将分散在网格中的数据片断拼接起来再通过不同的协议完成读取数据的服务。

3　网格资源管理

所谓对资源的管理，主要包括资源的搜索、选择、分配，而这几部分又是彼此结合、相互关联的。资源管理主要依托 agent 来实现，随着网格的迅速发展，原来像 NetSolve，Javelin 所使用的类似中心管理器(centralized manager)的集中形式已经不能适应分散的、动态变化的网格资源搜索的要求。原始的方式是通过一个集中的 server 来实现资源的搜索、选择、分配，它那里登记了所有 server 的资源情况以及所处位置，server 本身的一些特征等信息。后来随着网格的发展，为适应不同的应用需求和资源类型，产生了很多特定的中间件，这些中间件虽然在某个特定的领域有很好的性能，但是不具有普遍性，因此也不可能是网格发展的最终方向。由于在网格定义下，所有的服务都是可以随时退出和进入的，而且所使用的资源可能是一种也可能是多种，因此 P2P 的方式亦不适用。在此背景下，分散的 agent 也就应运而生。下面主要从搜索、分配、选择几方面进行一些探讨。

3.1　网格资源选择(Resource Selection)

由于资源蕴涵有多重性，并且资源的选择是基于客户的

要求的，而客户的要求通常具有个性化，这些都需要在资源选择框架中予以考虑。从 Globus 建立的资源选择框架中，我们可以看到这些要求实现的可能方式。

Globus 建立的资源选择框架定义了资源选择服务。在框架中存在一个发展了 Condor Match-making 的 set matching，支持单一和多重资源的选择。它同时提供了为用户个性化选择资源的接口。选择、构造、映射是整个资源选择的主要步骤。在 Condor 里，用 ClassAd(这可以认为是一种描述语言)来描述用户的请求，然后利用 match-maker 找到合适的资源，如果满足的资源多于一个，将对其进行排名，但是 Condor 只支持单一的资源选择。在此框架中发展了 Condor 的 ClassAd 语言，提出了 extended-ClassAd，以支持多资源选择。匹配是在一个 set 请求和 classAd set 间产生的，对 set 的整体和 set 中个体的限制要在 set 请求中表达，匹配的过程就是寻找满足要求的 set。(extended-ClassAd 提供了一些特殊的语法，Type 用来描述 set 的类型，Max，Min，Sum 可以来描述某种整体要求；Suffix 类似于普通的字符包含，Setsize 则是取得当前 set 的大小)。

真正的匹配算法分两步：1)过滤：去掉那些不满足个体要求的 set；2)建立 set 的过程是不断从剩余 set 中寻找最优的 set 放入 candidate set，再将 candidate set 与 best set 比较，如果更优就替代当前的 best set，这样不断进行直到没有剩余的资源存在。

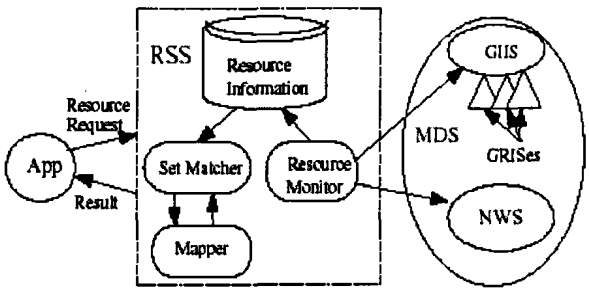


图 5　资源选择框架结构

图 5[7]描述的是此框架的结构：MDS 是 Monitoring and Discovery Service。GIIS，GRIS 分别提供信息、资源服务；而 RSS(Resource Selection Service)中的 Resource Monitor 从 MDS 中取得资源和信息并完成暂存更新的任务，Set Matcher 执行匹配算法，Mapper 为可看作特殊化的 library，面向用户。

在此过程中，为用户的个性化要求提供了特殊的 ranking 定义接口。

```
1. [
2. ServiceType="Synchronous";
3. Type="Set";
4. iter=100;alpha=0;x=100;y=100;z=100;
5. A=370;B=254;startup=30;C=0.0000138;
6. computetime=x*y*alpha/other.cpuspeed*A;
7. comtime = (other.RLatency + y * x * B/other.RBandwidth +
        other.LLatency+y*x*B/other.Lbandwidth);
8. exctime=(computetime+comtime)*iter+startup;
9. Mapper=[type="dll".libname="cactus";func="mapper"];
10. requirements=Suffix(other.machine,domains)
        && Sum(other.MemorySize)>=(1.757+C*z*x*y);
11. domains={cs.utk.edu.ucsd.edu};
12. rank=Min(1/exectime)
13. ]
```

从上面的描述，我们可以看到：用户在自己的请求中只需声明：1)种类，同步的或非同步的；2)任务描述；3)限制条件；4)所要使用的 Mapper 程序；5)排名标准。

上面的例子是一个很好的发展网格普遍性和兼容性的方向，当然这种普遍性的框架实际上也为一些特殊用途的网格提供了接口。

3.2　网格资源搜索(Resource Discovery)

在假定用户的请求已经用合适的方式表达出来了以后，资源的搜索则在很大程度上表现为一种独立于具体问题的算法。文[8]提出了一种具有普遍意义的资源查找框架。

假设用户将服务要求发给一个已知的节点，如果它有满足要求的资源，就将其作为结果返回，如果没有，它将在临近的节点中选择一个继续查找，直到达到这一要求的生存期限，如果期间找到了合适的资源，再将这些资源返回到用户，如果没有找到，则返回失败的结果。这一搜索算法主要包含两方面的机制：1)节点之间有一个互通的成员关系；2)决定选择哪一个前进节点的算法。对于前一个机制，可以通过一个软件协议(membership protocol)实现，在一个新节点进入时，要对其它连接节点进行注册，而已存在节点之间要通过定期的收发信息来保持联系。对于如何选择下一节点的问题，有四种可能的算法：1)随机选择；2)对以前的节点访问与相应资源要求在本节点存有记录，选择最接近所要求资源的节点＋随机选择；3)选择响应要求最多的节点；4)2)＋3)。

文[8]通过实验检测算法的性能如图6所示[8]。

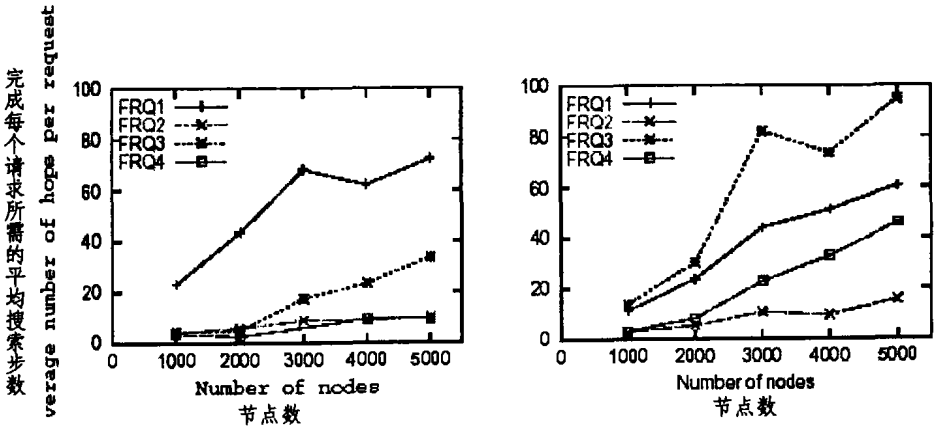


图6　各算法性能比较

首先考虑在某个给定频率的资源要求下，左图是不平衡的节点资源，右图是平衡的情况。纵坐标代表所需经过的节点(hops 是一个衡量节点数量的单位)，FRQ 分别代表4种不同的查找算法。从结果分析可知，无论在哪种情况下，随机选择

都是效率最低的，尽管很廉价。第二种都是效率最高，而耗费最大的第四种算法却并没有收到很好的效果。

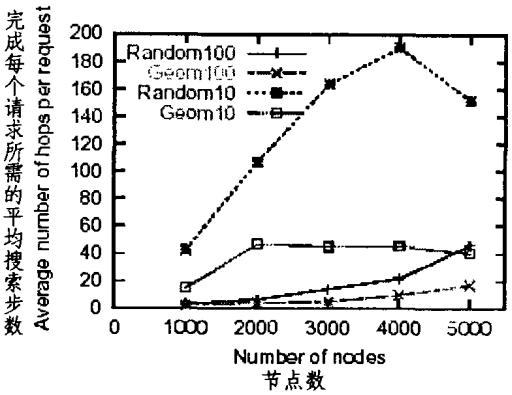


图7　资源出现频率对算法性能的影响

图7[8]采用的是第二种选择节点的算法，使用的是非平衡的资源分布环境，Random 和 Geom 分别代表两种不同的用户请求资源，其他同上。从结果中可以看出，资源访问频率越低，用户的请求影响就越大。

实际上，这种资源搜索的方式将 agent 的工作分摊到了网格中的各个节点，是 agent 在逻辑上分散化的一种形式，这对于提高效率的功效是显而易见的。

3.3　网格资源分配

在经典的 client-agent-server 即客户-代理-服务器的结构下，agent 起着决定性的作用。当新的 server 产生时，它要向某个 agent 发出请求，并提供它的资源信息和其它相关信息，最终成为可以为用户使用的资源。agent 的作用就是实现资源的匹配，它使用 ranking 的技术返回给用户一个可用资源的列表，在其中它要考虑用户的要求（大多是关于计算的要求，比如输入数据量，要解决的问题大小等）和 server 的静态、动态情况（在 agent 上第一次注册时所登记的情况和一些动态变化的性能比如网络的延迟，主机的工作负荷）。

在程序执行过程中有必要引入容错的机制。当程序在一台主机上运行失败后，agent 会自动调用可用 server 列表中的一台主机并将相关数据、信息等发送给新的主机，这一过程对用户来说是透明的。在早期，这种资源的调度、纠错与再分配甚至可能是用户的工作，他们自主提供规划的策略，而负载平衡只是一个简单的实现，当本地主机工作忙时，它可以从邻近的主机"盗用"资源。容错机制可以由用户手工修改完善。

容错和再分配机制是保证整个系统正常运行所不可或缺的，其实现不仅牵扯到分配的问题，而且还可能要重新进行部分的搜索和资源的再次选择（选择总是与分配相伴而生的）。

4　网格安全服务

网格可以看作是在一个动态的虚拟组织（Virtual Organization，VO）中实现资源的共享以及对计算问题的协同处理，而网格安全问题的提出，则是基于以下考虑[11]。

网格中的每个个体都可能有自己独立的安全策略和安全认证方式。首先，作为资源的拥有者，它必然要求对资源拥有足够的控制权；同时还需要考虑对于处于同一个域（domain）中的其余个体（它们可能是资源的要求者，也可能是资源的提供者），应该如何解决在共同处理某一问题时的相互协调。最后，从解决更大规模问题的要求来看，必然需要考虑域和域之间在资源共享时所要面临的问题。在考虑到以上这些资源共享的情形时，最先遇到的一个问题就是如何通过资源的要求者和相应资源的提供者之间的安全认证过程使得资源的要求者获得对所需资源的足够的操作权限。

从网格使用者（比如一个基于网格的应用程序）的角度来看，它希望网格本身能够支持对于虚拟组织中所有可使用资源的动态分配调度。这使得对于网格所要提供的安全服务的要求进一步提高了，因为它一方面要能够协调统一网格中跨域间或者同一域中的个体之间不同安全认证机制，同时也需要对外部的网格使用者提供一套相对统一的安全认证接口以将内部的实现细节屏蔽起来。这些要求很容易使人考虑使用代理，标准的协议，甚至采用专门的中间件来实现整个网格的安全服务。

4.1　对网格安全服务的要求

网格所要实现的安全服务主要包含了两个基本的内容：对资源使用的安全认证，和对资源操作的权限控制[12]。安全认证所要实现的是提供给资源的使用者（更确切地说，这时应该只是请求者）以及所要求的资源之间一套能够识别并确认对方身份的机制；权限控制则要求做到能够将资源使用者对该资源的权限明确地映射成该资源在其拥有者本地的操作权限。

基于以上的讨论，同时考虑到网格本身的一些特性，对于网格中的安全服务来说，它应该可以实现以下的一些功能[12]：

1）一次性的身份认证（Single sign-on）：资源的使用者应能在第一次身份认证被确认后就获得对资源相应的操作权限，之后在释放该资源的控制权之前，使用者对于该资源的各种操作（请求、使用、释放、内部通讯）都不需要再次通过身份认证的过程。

2）对用户信用信息的保护（Protection of credentials）：这里的用户信用信息主要包括用户的密码等信息，对这类信息的保护需要注意到存储时的保护以及在网络传输时的加密保护措施。

3）统一的安全服务接口的提供：这点在前面已经提到过，要满足这一点的要求，必须要引入对用户信用信息和安全策略的统一描述规范及编码标准。

4.2　网格安全服务的具体实现

在这里，将主要对 Globus 中的网格安全服务进行描述，以便总结出网格安全服务所应该遵循的一般性原则，目前所提出的一些框架结构和所采用的技术，最后对今后网格安全服务各方面可能的发展进行讨论。

4.2.1　Globus 中的网格安全服务　Globus 的构建主要是基于资源的概念，资源的提供者之间通过各种标准协议来实现资源的共享。因此，网格安全机制也可以基于这些概念来描述如下[12]：

1）整个网格环境中包含了多个逻辑上的域，每个域都包含了一系列的资源使用者（一般是某个程序或进程）和资源的提供者，更重要的一点在于，每个域中还包含了一套独立的资源管理机制和网格安全策略。

2）在每个域中对资源的操作都受到该域内对应的本地安全策略的限制，这些安全策略的具体实现则可以通过防火墙、SSH 等方法。

3）在每个域中，既有本地域内的局部使用者，也有整个网格内全局使用者。对于每个特定的域来说，需要建立一套映射机制，将全局使用者映射为本地域内的局部使用者，这样做的

目的是使得全局使用者与本地域内部的操作不发生直接联系，同时也保证了对全局使用者和本地域之间的一次性的身份认证(Single sign-on)特性。

4)在不同域之间的资源共享以及相应操作需要不同域之间的相互身份认证。

5)对于通过了身份认证的全局使用者来说，它在某个域所映射的局部使用者将被作为一般的局部使用者来对待，也就是说，在域内部对于资源的操作中，只有局部使用者的概念的存在，所有的资源使用者都遵循该域的网格安全策略。

6)所有的控制权限的判断都在资源的提供者一方根据本地域的安全策略在本地进行，这事实上保证了资源的提供者对资源的最终控制权。

7)当资源的使用者包含了多个进程时，如果这些进程使用同一个域内的资源，则这些进程可以共享同一套用户信用信息，而不是为每个进程都创建单独的用户信用信息。这样可以避免对这套用户信用信息的过多重复复制。

以上叙述的实际上是Globus中关于安全策略实现的一些原则，这些原则在网格构建时应该是普遍都需要遵循的。这是因为目前的安全认证的技术基本上都是基于类似用户信用信息和相应的用户权限管理这样的框架，如果在这样的框架内构建网格，上面的原则都是很有必要的。当然，为了实现更为灵活的控制策略，可以考虑在原有的基础上进行扩充，不过这事实上已经可以划入具体实现技术的范畴了，我们将在下面的讨论中进行更详细的描述。

4.2.2 Globus中网格安全服务的结构 前面已经提到，Globus主要是基于资源和协议来构架的，其网格安全服务的结构也不例外，图8[12]显示了Globus中网格安全的大体结构。可以看到，该结构中除了包含资源使用者(user)、提供者(resource)、用户信用信息(credential)以及各类协议(protocol)外，还包含两个重要的组成部件：用户代理(user proxy)和资源代理(resource proxy)。

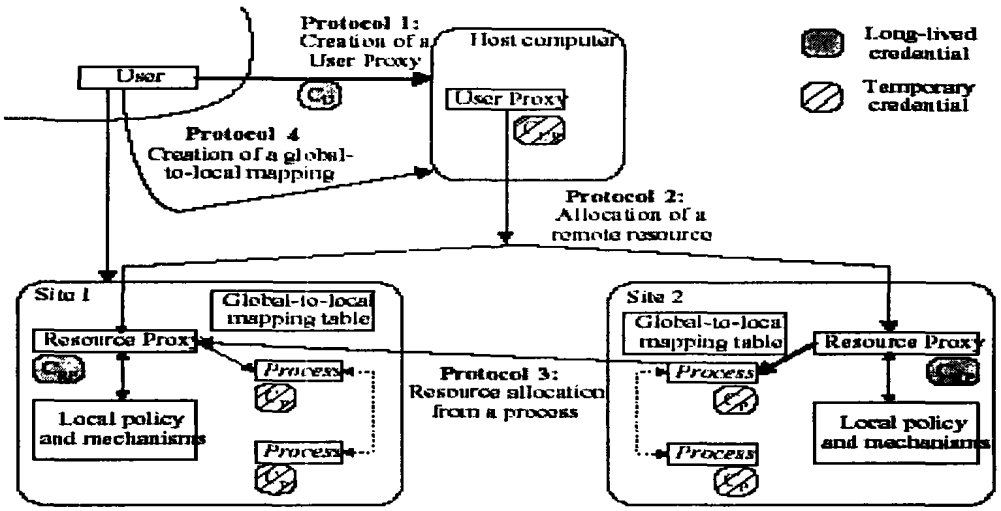


图8 Globus中的网格安全结构

用户代理是一个拥有自己独立的用户信用信息的管理进程，它在规定的时间段内代表资源的使用者进行各类操作。引入用户代理的意义在于，它不仅实现了实际用户和资源之间的一个中间层，还可以适应实际用户在某一个时间段内与资源所在域失去连接的情形。同时因为用户代理拥有单独的用户信用信息，所以在之后申请资源和对资源进行操作时所需的用户信用信息可以直接从用户代理处得到，对实际用户而言，这样也满足了网格安全服务所要求的一次性的身份认证特性。

资源代理是作为网格安全结构和资源提供者本地的安全结构之间的接口提出的，主要实现的功能是进行域间安全操作和本地域内安全操作机制的相互转换。

如图8所示，用户首先通过协议1登录到整个网格系统中，在系统中创建相应的用户代理，生成的用户代理使用自己的用户信用信息，通过协议2进行资源的分配(目的是为了创建新进程)，在资源的所有者一方，由资源代理受理用户代理的资源分配请求，并应用本地的安全机制分配出所要求的资源(在这里用于创建出新进程)。创建出的新进程则可以通过协议3向其他的资源所有者请求新的资源。在整个过程中，全局使用者到本地局部使用者之间的映射则是由用户通过协议4来创建的。

可以看出，在整个过程中，用户、用户代理、资源代理等之间的交互都是遵循一些标准的协议来进行的。在Globus中，协议1到4的作用如下：

协议1(User Proxy Creation Protocol)：创建用户代理。

协议2(Resource Allocation Protocol)：资源分配(请求由用户代理发出)。

协议3(Resource Allocation from a Process Protocol)：资源分配(请求由新创建的进程发出)。

协议4(Mapping Registration Protocol)：全局使用者到本地局部使用者的映射表注册。

综上所述，可以看出Globus的网格安全结构主要依赖于用户代理，资源代理通过各类标准协议将分散的各资源拥有者的本地安全策略加以分装统一，借以实现一套标准的对外安全服务接口。单从Globus的网格安全结构来说，对它的架构思想可以有两种理解：一种是将其看成以用户信用信息的流动为核心，由它驱动而建立起来的，这实质上是突出了用户所要提供的主要信息，可以认为是从用户的角度来看的；另一种理解则可以从网格的构建者角度来看，它所要面临的问题就是如何才能把有差异的一组资源提供者的不同界面接口隐藏起来，对外(对基于网格的应用程序开发)提供一套标准的接口，前面所描述的网格安全结构中的各种标准协议的应用则恰好满足了统一安全接口方面的要求。

总结以及进一步的讨论 以上主要从网格体系结构，网

格数据及资源服务，以及网格安全服务这三个方面对网格所使用的技术进行了分析。从分析中可以看出，网格体系结构实际上解决的问题是如何将网格中分布的各类资源提供者有效地组织起来，对外形成一个虚拟的整体组织，提供具有统一接口的各类服务；网格数据及资源服务则为网格内部各资源提供者之间，以及资源提供者与用户之间的资源和信息传输提供了一整套机制；网格安全服务的功能对于多数实用的网格系统是不可缺少的，它的目的在于协调网格中资源的提供者和资源的请求者对于资源所要求的控制权限，对于通用的网格安全服务而言，更进一步的要求就是提供尽可能灵活的控制策略。同时，这三类技术又是有着紧密联系的，网格数据及资源服务和安全服务都必然地要在网格体系结构的框架中来实现，而在网格数据及资源服务中也自然会涉及到网格安全的问题。

下面将对这三类技术可能的进一步发展进行讨论。实际上，其中任何一类技术的发展都可能会影响到其它技术的实现方式的改变。

以网格安全服务为例，对于网格安全结构的构建思想的讨论有助于思考对于网格整体的结构可能的扩展，甚至可能导出全新的结构，下面我们从另外一个角度对整个安全结构做一些讨论。

如果从资源提供者的角度来看，4.2.2节中所描述的结构的作法是由资源代理对外部的资源请求进行处理，也就是说，资源是在请求到来之后再动态分配的，这样保证了足够的灵活性，却使得资源分配的效率有所下降，这样在发生某一用户对某些资源的不断重复请求的情形时，灵活性的存在反而成了缺点。基于这种考虑，可以在资源代理对外部请求的处理中对以往的请求加以记录分析，对某一用户常用的资源，可以预先进行分配，并在资源代理处注册，这样当下一次该用户对该资源的请求到来之时，就可以直接对资源进行相应的操作，效率也将有所提高。但需要考虑的是，对以往资源请求的记录处理这一操作对于整体效率也将会有一定影响，如果可能的话可以在进行资源分配的时候进行。这里的讨论实际上已经与资源分配有关(但是对于用户信息的记录处理，则应该在网格安全服务的框架内进行)，这就是属于网格数据及资源服务的范畴了。

关于网格体系结构，现在提出的主要发展方向是开放式网格服务体系，即将构建网格所围绕的中心从网格资源转移到网格服务。这一改变同样对于网格的数据及资源服务和安全服务的实现方式会有相应的影响。

从以上这些论述可以看出，网格体系结构，网格数据及资源服务，和网格安全服务这三类技术对于网格构架的成功与否起着关键性的作用，对于这几类技术的具体实现方式的选择除了需要考虑其各自独立性能，还需要对各类技术之间的相互影响加以分析。

参 考 文 献

1 Beck M, Moore T, Plank J S. Exposed versus Encapsulated Approaches to Grid Service Architecture. GRID 2001, LNCS 2242,2001. 124～132

2 Grimshaw A S, Lewis M J, Ferrari A J, Karpovich J F. Architectural Support for Extensibility and Autonomy in Wide-Area Distributed Object Systems: [UVa CS Technical Report]. 1998(http://legion. virginia. edu/papers/CS-98-12. pdf)

3 Grimshaw A S, et al. Legion: The Next Logical Step Toward a Nationwide Virtual Computer: [UVa CS Technical Report]. 1994, (http://legion. virginia. edu/papers/CS-94-21. pdf)

4 Middleware Overview: Globus/Grids, Ian Foster, (http://www-fp. mcs. anl. gov/middleware98/glogr. ppt)

5 Globus: A Meta Computing Infrastructure Toolkit, Ian Foster, Carl Kesselman. (http://www. globus. org/)

6 Bent J, et al. Flexibility, Manageability, and Performance in a Grid Storage Appliance. HPDC'02, July 2002

7 Thain D, et al. The Kangaroo Approach to Data Movement on the Grid. In: Proc. of the Tenth IEEE Symposium on High Performance Distributed Computing. San Francisco, California, Aug. 2001

8 Iamnitchi A, Foster I. On Fully Decentralized Resource Discovery in Grid Environments. C. A. Lee(Ed.): Grid Computing - GRID 2001 Second International Workshop, Denver, CO, USA, November 12, Proceedings, Lecture Notes in Computer Science, LNCS2242, p. 51 ff. (http://link. springer-ny. com/link/service/series/0558/papers/2242/22420051. pdf)

9 Dongarra H C J. Netsolve's Network enabled Server: examples and applications. IEEE Computational Science & Engineering, 1998, 5(3): 57～67

10 Frey J, et al. Condor-G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing, 2002, 5(3): 237～246

11 GSI Roadmap Introduction talk from Grid Forum 5(Oct 2000). http://www. gridforum. org

12 Foster I, et al. A Security Architecture for Computational Grids. In: Proc. 5th ACM Conf. on Computer and Communications Security Conf. Describes techniques for authentication in wide area computing environments. 1998. 83～92

(上接第8页)

如何搜集到所需的MIB信息是一个值得关注的问题。例如为不同厂商的设备提供不同的MIB采集程序；对于不支持MIB信息读取的网桥，在其网段上设置代理侦听所有的生成树协议数据包提取相应MIB信息等。

结束语 准确的拓扑结构信息无论是对于网络管理还是一些网络相关的应用都是十分必要的。在本文中我们提出了一个基于标准生成树协议MIB组[3]的第2层拓扑结构发现算法。与基于地址转发信息的拓扑发现算法[4,5]相比，本算法信息采集和处理的开销小。在网络正常运行的情况下，生成树已经确定，运用本算法所得拓扑结构是准确无误的。

IETF于2000年推出物理拓扑MIB[2]，试图解决物理层拓扑结构的发现问题。但是由于没有确定如何获取这些MIB对象的机制，关于网络第2层、第1层拓扑结构的自动发现还有待更多的研究。

参 考 文 献

1 Media Access Control(MAC) Bridges. IEEE 8021. D-1990, May 1990

2 Bierman A, Jones K. Physical Topology MIB. Internet RFC-2922, Sept. 2000

3 Decker E, Langillle P, Rijsinghani A, McCloghrie K. Definitions of Managed Objects for Bridges. Internet RFC-1493, July 1993

4 Breitbart Y, et al. Topology Discovery in Heterogeneous IP Networks. In: Proc. of INFOCOM 2000, March 2000

5 Lowekamp B, O'Hallaron D R, Gross T R. Topology Disovery for Large Ethernet Networks. ACM SIGCOMM '2001

6 耿素云. 集合论与图论. 北京大学出版社, 1998

7 Tanenbaum A S. Computer Networks. Prentice Hall PTR, 1996

8 McCloghrie K, Rose M. Management Information Base for Network Management of TCP/IP-based internets: MIB-II Internet RFC-1213, March 1991