清 华 大 学

综 合 论 文 训 练

题目：网格计算管理与评价系统

系　　别：自动化系

专　　业：自动化

姓　　名：林筱

指导教师：曹军威　研究员

2010 年 6 月 17 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：林骏　导师签名：[illegible]　日　期：2010.7.1

中文摘要

在信息技术高度发达的今天，网格计算作为高性能计算未来的一个重要发展方向，正受到广泛的关注。但网格计算粗粒度的管理方式导致的网格体系僵化也带来了诸多不便，成为了将网格计算在科研领域更深入应用的阻碍。

根据用虚拟组织的思想，本文提出了解决该问题的动态虚拟组织管理方案，针对网格计算进行细粒度的项目管理。该方案按照虚拟组织的方式管理网格计算项目，以虚拟组织成员关系控制用户访问的权限。根据实际项目需要动态创建、取消虚拟组织，并通过评价函数控制用户和资源提供者的加入，使得虚拟组织的构成更有针对性。

针对 Globus 网格平台，本文给出了这个方案的一个具体实现——动态虚拟组织管理系统。该系统是针对 Globus 网格平台进行设计的网格中间件，根据虚拟组织成员变化动态修改 grid-mapfile，控制用户的访问。实际应用表明该系统与 Globus 的结合实现了 Globus 平台上细粒度的访问控制，证明了该方案的有效性。

关键词：网格计算管理；虚拟组织；细粒度；评价；Globus。

ABSTRACT

Grid computing, as a form of distrubuted computing with promising future, has attracted the more and more attention from all over the world. Meanwhile, however, the innate coarse-grained management of grid projects leads to an inflexible structure thus brings about inconvenience and may inhibit a broader application of grid computing.

With the concept of virtual organizations, this paper proposed a dynamic virtual organization management scheme which provides fine-grained project management to solve this problem. This scheme manages grid projects as virtual organizations, and controls a user's access of resources according to membership information in virtual organizations. Virtual organizations are dynamically formed and canceled when need. An evaluation system is introduced to control the join phase of users and resource providers to help virtual organizations to be more specific towards their goal.

Above the Globus grid system, a dynamic virtual organization management system constructed according to this scheme. This system is a specially designed middleware for Globus, which controls a user's access of resources by modifying grid-mapfile according to VO membership changes. This system provides fine-grained grid management as expected, which demonstrated the effectiveness of this scheme

Keywords：Grid computing management; Virtual organization; Fine-grained; Evalutaion system; Globus.

目 录

第1章 题目背景

在上世纪 90 年代早期，Ian Foster 和 Carl Kesselmen 在他们的文章“The Grid: Blueprint for a new computing infrastructure”[1]中首先提出了”网格计算”一词。网格计算技术将网络上空闲的计算资源整合起来，并通过虚拟化技术将其包装成一般形式的计算能力，服务于用户。

大多数电脑的计算能力并没有充分被利用。一些基本的文档处理、浏览网页等操作只消耗不到 5%的 CPU 时间。即便对于服务器来说，在低谷时段 CPU 也有大量的空闲。而网格技术可以将地理上可能十分分散，关联很小但相互之间存在网络连接的计算资源整合为一台超级虚拟计算机，协同完成大规模的计算工作。

在网格计算的流程中，一个网格计算的任务首先被提交到一个任务分解和调度程序。任务被分解为适合并行计算的一些子任务，然后这些子任务由调度程序分配到网格中的各个计算机上。每个计算机维护自己的任务队列，并完成自己任务队列中的任务。最终，当所有子任务均被完成后，由任务分解和调度程序进行整合，给出程序执行的最终结果。

在网格计算中，每个空闲节点都可以通过共享它们自己来提供计算能力。每个要提交任务的用户可以享受到网络上所有空闲节点的服务，省去了自己购置设备带来的大量开销。因此网格计算的一大优势在于能够在短时间内以低廉的代价收集到大量的计算资源用于计算。在个人计算机十分普及，计算机处理能力日新月异的今天，网格计算的容量也在随着计算机的普及和更新换代而飞速发展，使得网格计算的优势更加明显。

近年来，网格计算技术得到了越来越广泛的应用。企业级的网格计算已经成为一项产业，有着十分广阔的市场。IBM、微软、苹果公司和 Google 相继开展了相关项目。在网格计算的日常应用方面也有了突飞猛进的发展。杀毒软件公司纷纷推出“云安全”计划，而一些有实力的公司已经启动了云操作系统的开发工作。

网格计算能够以低廉的价格快速获取计算能力的优势使其尤其适用于科学计算。科研工作中经常出现需要大数据量、大计算量的计算。这样的计算往往只能由超级计算机来完成。而网格计算技术可以通过将网络上散布的计算机的空闲运算能力整合起来，虚拟出一台超级计算机来完成同样的功能。购置并维护一台超

级计算机需要客观的成本，但通过网格计算技术，完全可以获取同样的计算能力而不需要单独购置新设备。

LIGO[2]（Laser Interferometer Gravitational-Wave Observatory，激光干涉引力波天文台）是网格计算的一个典型应用场景。LIGO 项目的目标是直接观测被爱因斯坦相对论所预言的引力波。LIGO 是一个规模宏大的项目，由 MIT、Caltech、清华等多所世界著名大学联合承担。在 LIGO 中有很多工作需要很大的计算量才能完成，如 Montage，Rmon 等，因而网格计算对 LIGO 项目有着很重要的意义。在 LIGO 项目中，科学家们能够过网格技术使用分布在世界各地的十几台超级计算机来完成对 LIGO 数据的计算和分析工作。

另外一个著名的网格计算项目是 GIMPS[3] (Great Internet Mersenne Prime Search,寻找最大梅森素数)。验证一个大数是不是素数需要非常可观的计算量。何况这个大数长达千万位。目前寻找到的最大素数为 $2^{43,112,609}$-1，长达 12,978,189 位，于 2008 年 8 月 23 日发现。GIMPS 项目有着专门的网格平台，用户下载客户端后，程序会自动连接网络并开始工作。

网格计算技术正在以其对计算能力的充分挖掘帮助着大量的科学研究项目。但随着网格计算的应用日渐广泛，网格规模的逐渐扩大，网格计算粗粒度管理方式的弊端开始显现。

首先，网格计算项目在粗粒度的管理模式中，针对每一个项目需要构建一个专门的网格。构建专门的网格确实在技术上比较容易实现，但这样的网格计算比较僵化，当应用变化时往往需要另外搭建新的网格平台。在 LIGO 应用中，计算项目多种多样，针对每一个项目构建一个网格本身就是一个不小的开销。

其次，如果将一组网格计算项目强行放进同一个网格进行处理，项目相互之间便需要权限管理来区分。比如项目 A 中的成员只能访问自己的资源，而不能访问项目 B 中的资源。一个很直观的想法就是加入一个身份认证和访问控制机制。当项目 A 的资源收到来自项目 B 的成员的访问请求时，则拒绝。当收到项目 A 的成员的访问请求时，则允许访问。这看似很好地解决了项目之间的管理问题。然而，当项目 A 有新成员加入时，项目 A 中的所有资源都需要添加成员 A 的访问权限。这对于资源较多以及人员变更频繁的网络来说也是一个不小的负担。另外，当网格中的项目增多时，用户想查找并加入某一个具体的计算项目，或是某一个计算项目需要寻找一个合适的资源时也有着诸多不便。因此这种网格管理的机制也没有从根本上解决网格计算体系僵化的问题。

为了将网格计算大量灵活地应用到科学计算当中，需要一种灵活细粒度的管理机制来管理网格计算。

第2章 相关工作

2.1 虚拟组织的概念与应用

虚拟组织，不同于实际的组织，是一个临时的联盟，并不成立新的法律实体。在虚拟组织中，成员们共享各自的技能和资源，向着共同的目标前进。虚拟组织的成员在地理上可以十分分散，往往通过通信与协同办公的技术进行合作。他们功能互补，相互之间资源深度共享，可以更好地完成一个任务。在任务完成之后，虚拟组织自动解散，成员们寻找并加入新的虚拟组织进行下一项任务。

在科研工作中，虚拟组织的意义十分明显。往往一项科研需要许多来自不同领域的专家和工作室合作进行。一个虚拟组织较其它独立的研究室更加灵活，有更强的针对性，往往能够很好地完成任务。

通过动态管理虚拟组织的方式可以很方便地实现权限管理。同一个虚拟组织内的成员之间可以相互访问资源，而虚拟组织外的用户不能访问虚拟组织内部的资源。在收到访问请求后，资源提供者只需要在虚拟组织登记表中查询用户是不是与自己处于同一个 VO 中。在维护虚拟组织登记表的过程中，也只需要根据用户加入或退出虚拟组织的请求来对应修改表格，而不是维护所有成员的访问权限映射表。这对于一个节点众多的网格，省去了可观的重复操作。动态虚拟组织的这个特点使其很适合于网格管理工作。

动态创建与管理的虚拟组织能够实现细粒度的网格管理。对每一个具体的计算项目构建一个单独的虚拟组织，组织中的科学家可以访问组织中的资源。当科学家需要更多资源时，可以批准更多的资源加入虚拟组织。每个科学家并不需要直接与组织外的其他人建立信任关系。这正契合了计算项目自然的组织方式。Cyber Infrastructure 与效用计算是用虚拟组织方式管理网格计算的一种应用。

效用计算是对计算资源的打包。如计算与存储，作为一种计量收费服务，就像传统的公共设施（水电煤气电话）一样。使用这套系统只有很低至几乎没有的起始花费来收集硬件，计算资源完全是租用。对需要很大计算量的客户也不会由于需要聚集大量的计算机而有明显的延迟。

Cyber Infrastructure 以效用计算为基础，有效地将数据、电脑和人整合在一起，为科学家的工作提供了一个理想的计算平台，支持先进的数据查询、存储、管理、

整合、挖掘、可视化以及其它网络服务的新的研究环境。Cyber Infrastructure 对科研领域有着尤其重要的意义。

在 Cyber Infrastructure 的场景中，存在着有各种不同研究目标的大大小小的虚拟组织。效用计算的提供商加入需要计算的虚拟组织中，并提供计算服务，从而有针对性地对具体的项目提供廉价的计算能力。

2.2 VOMS

VOMS[4]（Virtual Organization Membership Service，虚拟组织成员服务）是一套按照虚拟组织理念进行网格计算管理的软件，由 European DataGrid Project 开发。

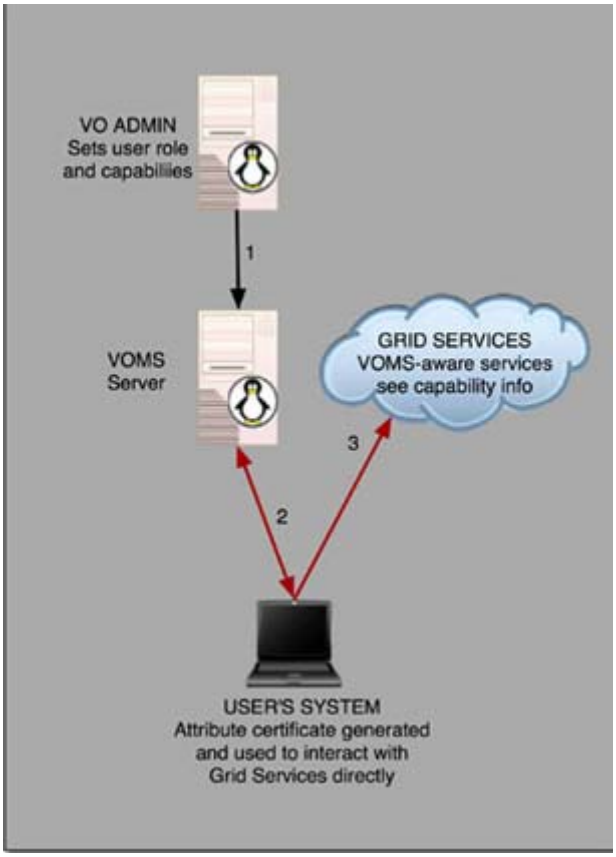

图2.1　VOMS的体系结构（转自Globus.org）[①]

[①] http://www.globus.org/grid_software/security/voms.php

VOMS 的核心是一个数据库。数据库中按照一定的格式存储着各种虚拟组织的信息。通过一套管理工具，管理者可以对虚拟组织中用户的角色和权限进行管理，每个成员可以在 VOMS 获取一个包含用户角色和权限信息的证书。凭借这个证书，用户可以按照规定的权限访问支持 VOMS 管理的网格。

VOMS 能够与现在的网格完美对接，因为 VOMS 只是在用户证书中多提供了包含用户权限的信息。在一般的网格上，VOMS 的信息不会被用到，但证书的剩余部分仍可以被正常使用。

VOMS 提供了良好的客户端支持。

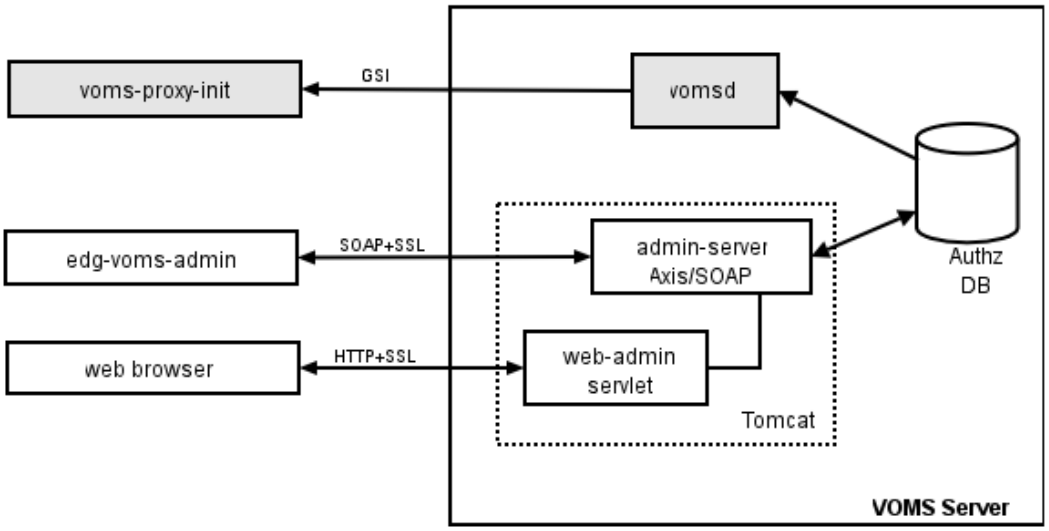

图2.2　VOMS的程序结构（转自VOMS的项目网站）[②]

然而 VOMS 也有一个显著的问题。由于网格程序并不会与 VOMS 服务器进行直接沟通，当虚拟组织成员或权限发生变更时，网格应用并不能第一时间获知这一变更。而若想让这一变更被获知，就需要用户重新签发证书。这对于管理一个成员频繁变动，十分活跃的虚拟组织来说并不是很有利。更细一步的访问权限管理，比如根据成员的诚信度调整成员的角色便很困难。VOMS 中成员角色的变更是由管理员来决定，操作量可能相对较大。

[②] http://edg-wp2.web.cern.ch/edg-wp2/security/voms/voms.html

2.3 GUMS

GUMS[5]（Grid User Management System）提供了网格用户身份映射的服务。当资源的访问控制与网格的认证方式相异时候，通过在网格身份与资源的访问权限之间建立映射关系，可以建立一套访问控制系统。

比如用户需要的资源都放在某一个 UNIX 帐户中，可以通过用户在网格上的身份与 UNIX 帐户的一个映射实现用户对指定帐户的访问。

GUMS 尤其适合网格中访问控制方式很多的情况。在每次访问控制方式转换时建立一个映射表，便可以很好地解决访问控制问题。通过访问映射表，GUMS 可以将一组用户映射到一个位置，而将另外一组用户映射到另外的位置来实现网格中有多类用户时的访问控制。

然而 GUMS 这种映射的方式带来了一个问题。如何维护映射表？在网格计算的动态环境中，映射表可能需要经常变化。因为一个用户大范围地修改映射表并不是一个很方便的方法。另外，在 GUMS 并没有反映出每个项目的信息。用户只看到了资源，而无法获知项目的组织信息。而这些信息只能在外部单独维护。

2.4 VO Services Project

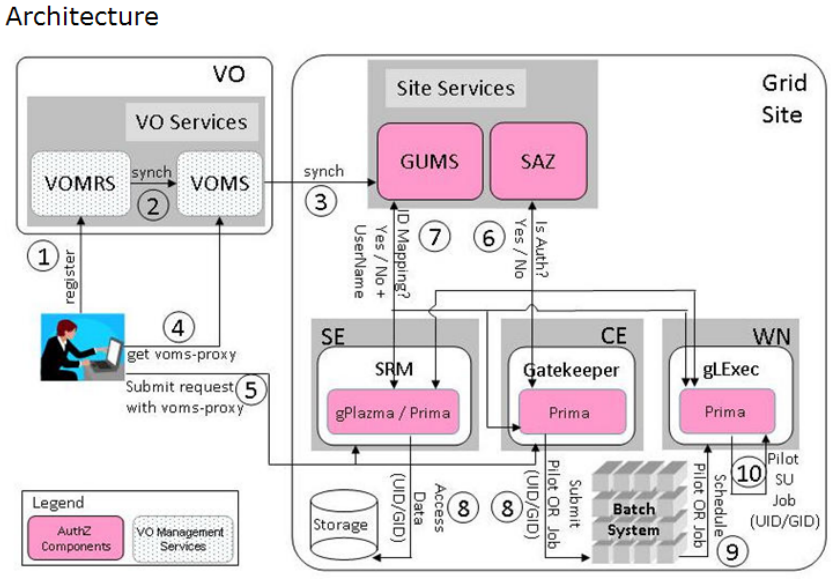

图2.3　VO Services Project的体系结构（转自项目的官方网站）[③]

[③] http://computing.fnal.gov/docs/products/voprivilege/

VO Services Project[6]是对 VOMS 和 GUMS 的一个综合。用户在 VOMS 处登记，GUMS 通过与 VOMS 同步获取信息对用户进行映射。

VO Services Project 很好地对二者的优缺点进行了互补，形成了一套相对完善的细粒度虚拟组织管理系统。

但 VO Services Project 依旧带有 VOMS 和 GUMS 的一部分缺点，如需要多次颁发证书等。

第3章 解决方案

3.1 带有评价系统的动态虚拟组织管理方式

动态虚拟组织通过动态地在网格内部创建、取消以及虚拟组织成员变更来实现网格计算的权限管理。

动态虚拟组织管理的核心是维护包含虚拟组织信息的数据库。这个数据库中有两类重要的表格：虚拟组织登记表和成员登记表。虚拟组织登记表中记录了虚拟组织的信息，包括虚拟组织的创建者，虚拟组织服务器的位置、虚拟组织的目标等等。成员登记表中记录了虚拟组织的成员信息。包括成员的身份标识（ID）、属于哪个虚拟组织、IP 地址、成员在虚拟组织中的角色等等。

当一个用户或资源提供者加入一个虚拟组织时，他将可以访问虚拟组织中的所有其他成员提供的资源，而同时他也需要贡献出自己的资源（如果有）供虚拟组织中其他成员来使用。虚拟组织外部的用户或资源提供者不能够访问虚拟组织内部的资源。而虚拟组织的关系可以通过查询虚拟组织成员的方式从数据库中得到，因此权限管理变得十分方便，只要通过读取数据库中的信息，即可确定成员是否可以访问，以及应该如何被映射。

当一个新用户或新资源提供者进入网格后，他可以在数据库中查找虚拟组织信息，寻找并加入与自己的目标或与自己的服务特性相吻合的虚拟组织，从而快速投入使用。

如果要开始一个新的项目，则可以创建一个新的虚拟组织。在虚拟组织登记表中添加新的一项，并在成员登记表中添加本虚拟组织对应的新的成员，或者新开一张成员登记表。当项目结束需要关闭时，则只需要撤销当前的虚拟组织，在虚拟组织登记表中删除对应项，并删除成员登记表中对应的内容。

虚拟组织在批准用户或资源提供者的加入请求之前会进行审核。审核并批准加入的过程可以是手动的，但对于有很多用户和资源提供者的大环境来说，自动批准或拒绝申请可以节省很多工作量。审核的过程可以通过一个函数的形式完成，比如。如果审核不通过，则拒绝该用户或资源提供者的加入申请；如果审核通过，则自动将该用户或资源提供者加入成为虚拟组织的成员。另外，审核过程中也可以同时对用户打分，并根据打分情况决定用户能访问的资源或决定用户的角色，从而实现更细粒度的权限管理。

审批的过程可以是有目的的。比如提高网格的 QoS 质量。王震等在[7]中给出了专门针对网格计算资源评价的算法。这个算法首先选择一些有威望的节点形成一个 Committee，由 Committee 对资源进行打分，最后通过一个模糊决策方式得到资源的最终得分。该系统被证明能够有效地提升网格资源的质量。通过有目的性的审核机制，可以针对虚拟组织的目标优化组织内部的人员和资源配置，从而更好地完成预定的任务。

相比传统的网格计算管理方式，动态虚拟组织的管理方式有如下优点：

1、动态虚拟组织的管理方式提供了细粒度的计算项目管理。所有网格计算项目可以共生在一个大的网格中，项目之间相互兼容。项目 A 的用户和资源提供者也可以加入项目 B 而成为项目 B 的资源提供者。每个用户或资源提供者只需要加入网格一次，即可参与网格中左右的计算项目。

2、不需要每个成员维护一个访问控制表。访问控制的信息由虚拟组织成员信息翻译得到，而虚拟组织成员信息表只需要在服务器端维护一份，不需要每个成员都保存。这样一来，在成员加入或退出一个项目的时候，并不需要在每个用户的位置上都作改动。访问控制的维护更加便捷。

3、动态虚拟组织的建立、加入、退出和解散完整地描述了一个计算项目的开始、资源收集、计算和结束的过程。一方面便于项目管理者对项目进行管理和评估，另一方面也契合了计算项目本身的自然特点。通过虚拟组织的管理方式可以很清楚地追踪一个计算项目的发展，而且可以分清项目与其他计算项目之间的界限。

4、增加了对用户和资源管理者的评价机制，使得虚拟组织的成员加入审批有了一个有价值的的参考标准。依照评价系统给出的结果，甚至可以把成员的加入审批变成一个完全自动的过程，从而节省大量的工作量。这种细粒度的权限管理方式可以使得依据审批规则生成的 VO 对计算项目的目标更有针对性，从而更好地完成 VO 所预定的任务。

3.2 动态虚拟组织管理的架构

动态虚拟组织管理的方式提供了便捷的细粒度访问控制功能。通过这种方式管理的网格计算系统由三部分组成：客户端、CI 服务器与 VO 服务器三大部分组成。

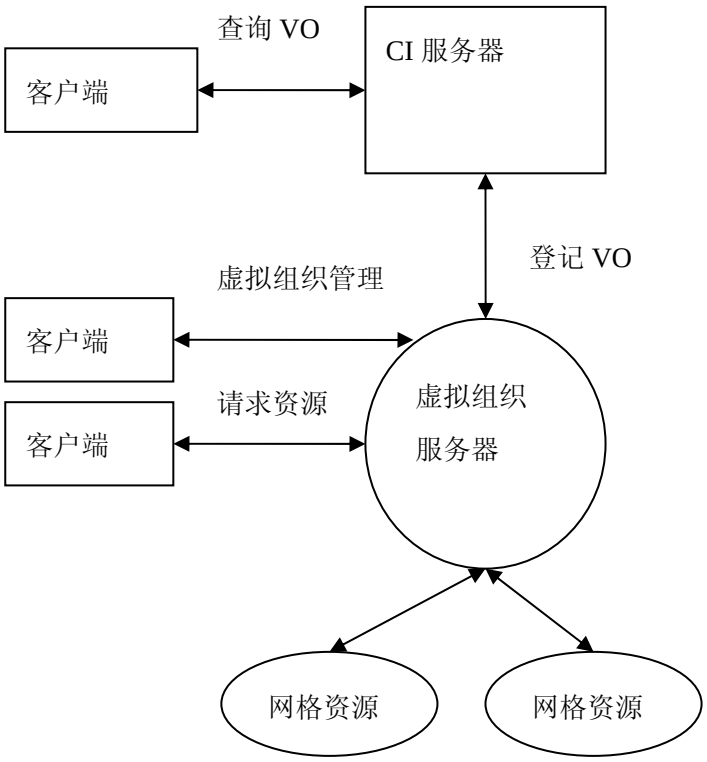

图3.1　动态虚拟组织的管理架构

在动态虚拟组织管理的架构中，用户对资源的访问需要虚拟组织的认证来授权。因此当用户访问资源时，需要向虚拟组织的服务器提交访问资源的请求，虚拟组织服务器验证用户身份后授权用户访问虚拟组织中的资源。

虚拟组织服务器还需要维护虚拟组织成员的表格。因此虚拟组织服务器需要支持用户的加入、退出、成员查询等操作。为了更好地维护虚拟组织成员登记表，虚拟组织服务器中还需要配备管理员的机制。虚拟组织的管理员应当能够在必要的时候批准加入申请，或踢出恶意的成员。虚拟组织服务器中还需要有合理的管理员任免机制。

用户和资源提供者还需要查找合适的 VO 来加入。一个登记所有 VO 信息的服务器也必不可少，那就是 CI 服务器。CI 服务器管理网格中所有用户和资源的身份信息，并登记所有 VO 的信息。CI 给予每一个用户以及每一个资源一个唯一的标识，用于访问。CI 登记所有 VO 的信息，如 VO 的创建者，VO 的目标，VO 服务器位置等等，方便用户和资源提供者查找并加入对应的 VO。

CI 服务器需要维护用户的 ID 以及虚拟组织登记表。

CI 需要支持虚拟组织创建者对虚拟组织的登记。CI 需要记录虚拟组织的 ID（名称）、创建者、IP 地址以及虚拟组织的目标等信息。CI 需要支持虚拟组织的登记、解散等操作。

另外，CI 还应当支持用户或资源提供者对虚拟组织的查找操作。CI 应当配备一个证书中心，供新用户注册使用。

动态虚拟组织管理系统应该有一个便于操作的客户端。这个客户端需要与 VO 服务器以及 CI 服务器进行通信，完成用户操作的功能。这需要客户端、CI 服务器和 VO 服务器之间有统一的通信标准和消息格式。

这个架构支持了用户从查找 VO 到使用资源的一整套流程，是动态虚拟组织管理的基本框架。

3.3 动态虚拟组织管理与 Globus 的连接

Globus Toolkit 是由 Globus 联盟以及一些其他的开发者开发及维护的一款构建网格以及相关应用的开源软件。目前 Globus Toolkit 已经得到了广泛的应用，各大公司、学校都在用 Globus Toolkit 或对其进行改写实现网格的搭建。

Globus 通过证书机制和帐户映射的方式完成用户认证以及访问控制。在 Globus 中有两种证书：Hostcert 和 Usercert。前者是设备的证书，在 GateKeeper 中用于用户身份的验证，代表资源提供者。后者是用户的证书，代表一个用户。在 Globus 中，用户用 Usercert 可以远程向资源提供者的计算机提交任务。用户首先连接到资源提供者 Globus 的 GateKeeper。GateKeeper 认证用户的证书，确保用户是可信的。之后 Globus 在映射文件 grid-mapfile 中查找用户 Usercert 的 Subject（主题）部分，并将用户映射到其 Subject 对应的 localname，如本地帐户上去。映射完成后，用户的任务便可以以 localname 对应的权限执行。

由于 Globus 并不支持主动查询成员登记表的方式进行访问控制。针对 Globus 通过配置 grid-mapfile 与制定 localname 权限实现访问控制的方式，动态虚拟组织管理框架可以通过修改 grid-mapfile 来完成对 Globus 构建的网格计算系统的访问控制。Globus 修改 grid-mapfile 的命令主要有两条：

添加：grid-mapfile-add –dn distantname –ln localname

移除：grid-mapfile-delete –dn distantname –ln localname

每个资源提供者均使用 grid-mapfile-add 命令，将自己所在虚拟组织中的所有用户在 grid-mapfile 中映射到一个供本虚拟组织成员访问的 localname。当有用户退出时，资源提供者使用 grid-mapfile-delete 命令，取消退出的用户对该虚拟组织对应的 localname 的访问权限。当资源提供者自己退出时，使用 grid-mapfile-delete 命令删除 grid-mapfile 中该虚拟组织中所有用户的访问权限。这样就实现了动态虚拟组织管理中的访问权限控制。

动态虚拟组织管理系统与 Globus 连接方式的示意图如图 3.2 所示。

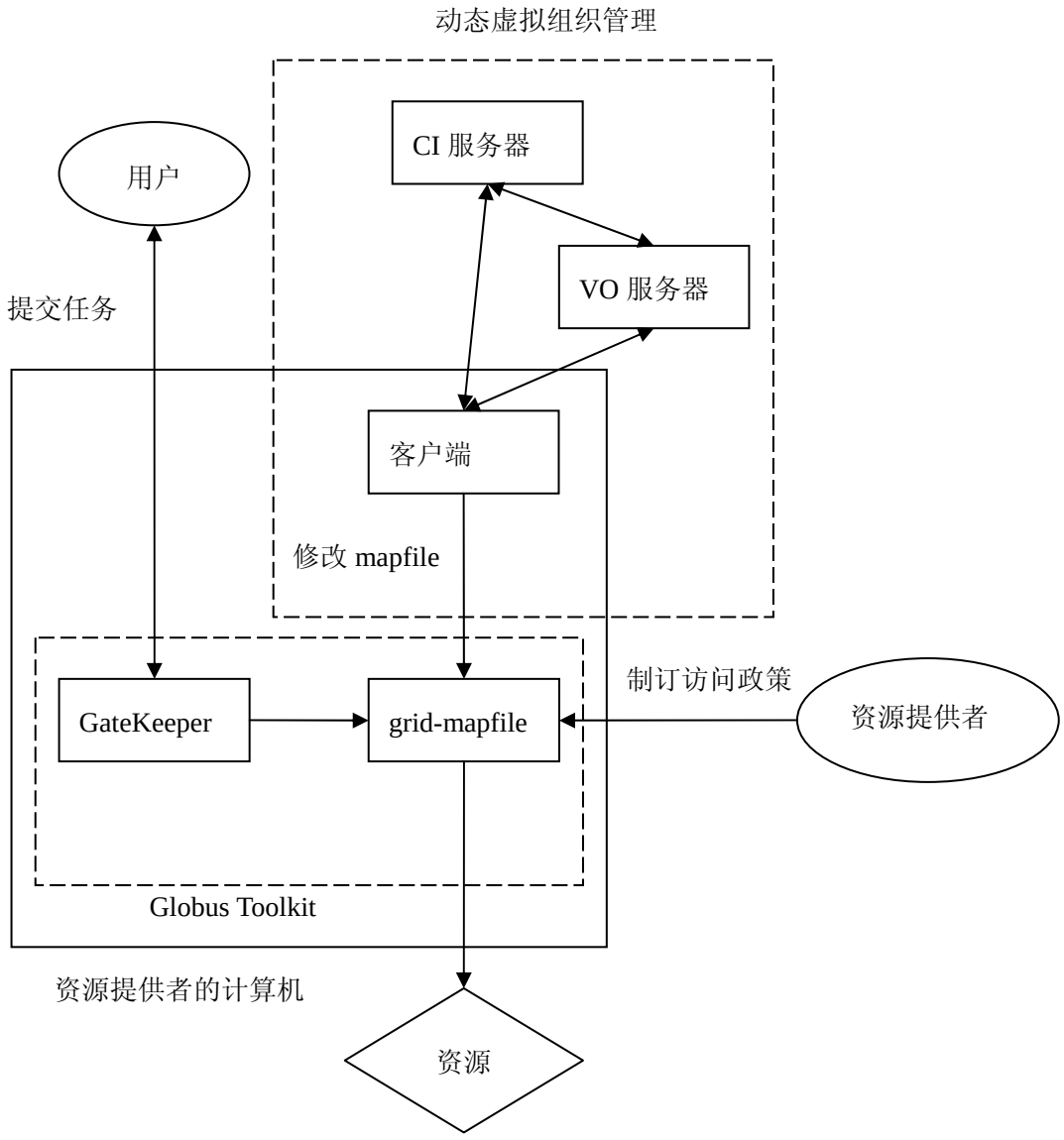

图3.2　动态虚拟组织管理与Globus的连接

当虚拟组织成员情况发生变化时，执行 grid-mapfile-add 和 grid-mapfile-delete 命令的过程应该自动完成，而不是让资源提供者自己手动输入。为了完成这个目标，需要服务器端与客户端的配合。当服务器端 VO 成员信息发生变化时，服务器需要将这种变化告知客户端。之后客户端通过变化成员的 Subject、加入还是退出和资源提供者的 localname 生成 grid-mapfile-add 或 grid-mapfile-delete 命令并执行，从而自动地动态修改 grid-mapfile。如果客户端能够长期在线，这种机制可以很好地完成根据虚拟组织成员变化动态修改 grid-mapfile 的工作。

但有些时候，客户端并不能够一直在线。比如资源提供者的计算机经常需要关机维护，或者资源提供者只是家庭的网格用户，只是共享了家用计算机的一些资源时，资源提供者并不能保持一直在线，甚至经常在线也可能很难保证。在这种情况下，服务器无法实时报告成员变化，而需要将成员变化信息保存起来，待资源提供者上线之后进行对应 grid-mapfile 的修改。为了解决这个问题，在 CI 服务器中建立一个消息表 Message，保存所有 VO 的成员变化信息，并记录每条变化发生的时间。资源提供者上线后向 CI 服务器请求 Message 表，按照时间顺序执行 VO 成员变化的修改，就可以实现自动修改 grid-mapfile 的功能。

动态虚拟组织管理架构与 Globus 连接可以完整地实现用动态虚拟组织管理网格计算的理念。Globus 应用范围广，因此动态虚拟组织管理与 Globus 连接一方面提供了优秀的网格计算管理模式，另一方面也更容易将这种管理模式推广应用。

第4章 动态虚拟组织管理系统的设计

4.1 系统的设计原则

为了实现用动态虚拟组织管理方式管理网格计算，需要按照第三章中的架构设计一个动态虚拟组织管理系统。

该系统应能够完成基本的虚拟组织管理操作，如建立、加入、退出、解散以及查询组织成员等功能。另外，该系统应该能够按照虚拟组织的信息向网格计算软件提供控制外来用户访问的信息。对于组织内部的成员则允许访问，对于组织外的成员则拒绝访问。

该虚拟组织管理系统应该具有以下特点：

（1）安装便捷

对于网格计算的搭建者，软件安装过程简捷，用户需要修改的信息少，则布置网格的成本会大大降低。让用户了解软件每一个细节应当怎么配置，对于一个非计算机专业的用户（如物理学家）来说不仅很困难，而且没有必要。最理想的情况就是程序可以无人值守而自动安装，这样一来可以大大节约人工成本。

给予每项需要配置的变量一个默认值，可以在对用户体验影响很小的情况下大大降低用户的操作量。如给程序一个默认的安装目录，以及一个默认的数据库名。这样在用户不需要软件细节信息时，直接按照默认配置安装，可以使软件具体实现对用户透明化。而对于一些要求较高的专业用户，可以对配置文件进行相应的改写，执行可订制的安装过程。

（2）对现有系统组件兼容性好

如果用户已经安装了一些能够实现同样功能的组件，则不应当要求用户安装额外的组件，而应当尽量使用现有的系统组件。当某一个组件由于不可抗力需要更换时，可以不需要更换其它组件。

如数据库，如果用户已经安装了一种数据库，则不应当要求用户安装另外的数据库。当对某个数据库的技术支持终止后，用户只需要更换数据库和仅仅修改系统的数据库模块即可完成改写。

这样的特性要求该系统的各组件尽量做到自治，耦合越小越好。

（3）安全性

虚拟组织管理信息的收发应该是绝对安全和保密的。如果不能保证信息传输的安全保密，则整个系统将形同虚设。想象在一个系统中，一个用户可以冒充任

何一个其它用户，提交计算任务，查看组织成员。那么整个服务器就只剩下一个虚拟组织了，那样整个系统将失去意义。

因此，安全是系统中十分重要的一环。

安全性体现在以下两个方面。

信息传输的安全性：信息传输过程中应该防窃听、防伪造。需要保证服务器与客户端双方身份的真实，需要保证数据传输的保密性。

组织管理的安全性：虚拟组织管理中需要严格控制权限。普通成员不能进行虚拟组织的创建者的操作，非组织成员不能进行成员操作，也不应该获知某个虚拟组织中有哪些成员。

（4）稳定性

虚拟组织服务程序要在服务器上长期运行，因此需要服务程序保持稳定，并能够在客户端发送不合法的消息时不出现问题。

（5）灵活性

虚拟组织管理的概念可以应用于很多领域。比如 P2P 网络以及协同办公等领域。如果需要，该管理系统应该能够用最少的修改即可作为组件参与相关领域的开发。另外，当虚拟组织管理有更多内容的时候，该系统也应能够胜任。

4.2 系统的总体架构

VO 服务器与 CI 服务器的核心都是数据库操作，可以合成为一个程序在一台服务器上运行。服务器部分由配置文件、通信、数据库、评价以及 VO 与 CI 管理模块构成。配置文件模块负责配置文件的读取与分析；通信模块负责通信规则与保证通信的安全；数据库模块负责管理数据库的访问操作；评价模块负责对用户或资源提供者给出评价；VO 与 CI 管理模块负责按照命令对 VO 数据库与 CI 数据库进行合理的操作，维护成员登记表和虚拟组织登记表。

客户端由配置文件模块、通信模块和用户界面模块组成。配置文件模块负责读取与分析配置文件；通信模块负责客户端与服务器的通信规则与安全；用户界面模块负责用户与客户端的接口。

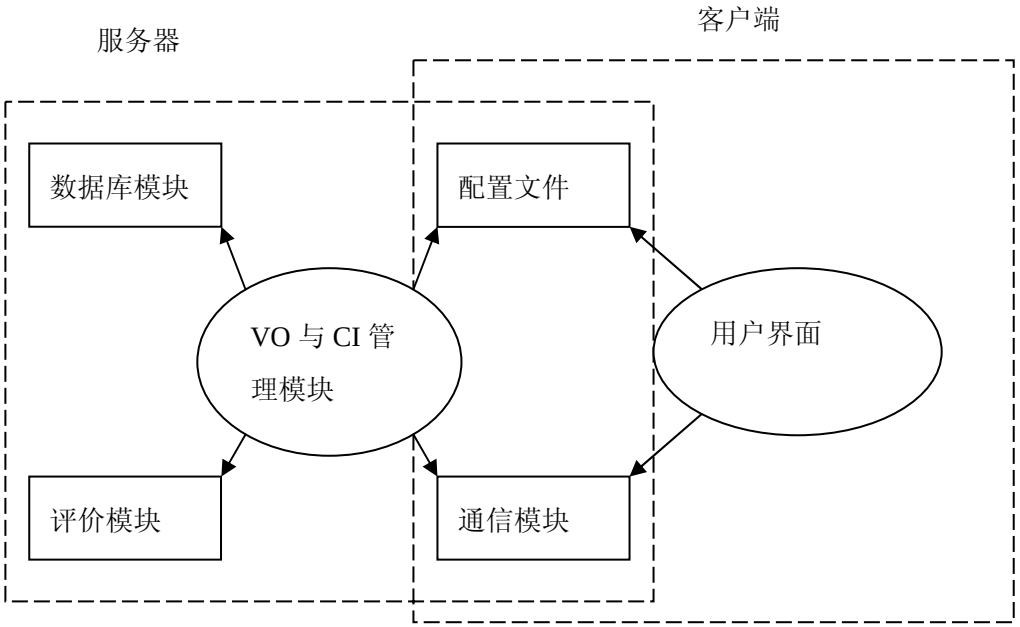

图4.1　动态虚拟组织管理系统的模块结构

4.3 配置文件模块

本系统使用 ConfigFile 类型读取并维护配置文件信息。

配置文件往往具有这样的格式：

```
[SERVER]
keyfile=18HOSTcert.pem
admin=host/yushu.riit.tsinghua.edu.cn
password=123456
port=8889

[ODBC]
username=root
password=524288
dns=SERVERDB
```

每个方括号中为域名，之后的内容都属于这个域中。内容的每一行都是“X=Y”的结构，等号左边是项目的名称，等号右边是该项的值。

配置文件的格式可以被抽象成 std::map 的结构而读入。

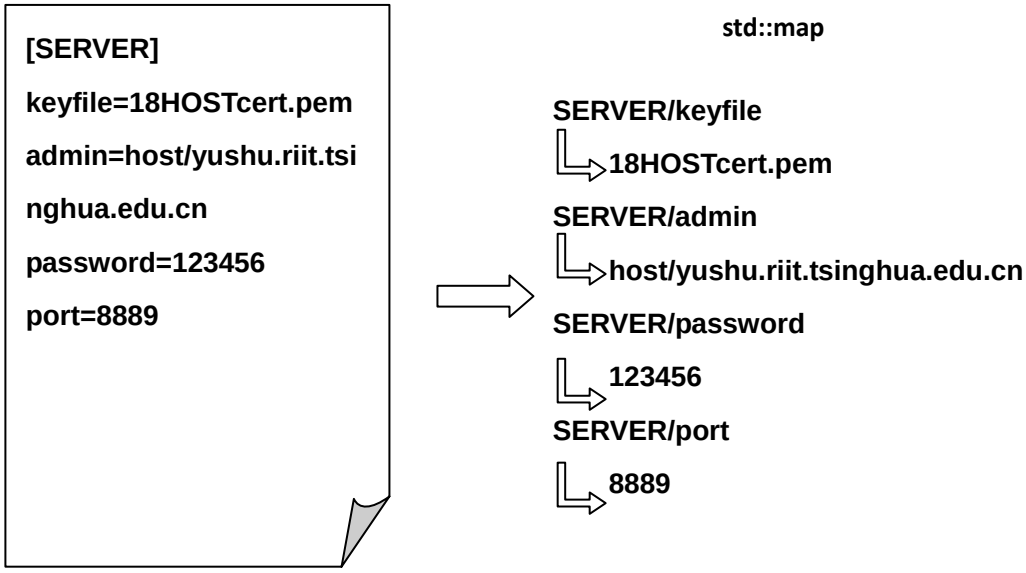

图4.2　配置文件与std::map结构的转换

std::map 的结构也比较便于修改、查找、遍历等操作。通过遍历 std::map，可以得到配置文件用 vector<string>的结构的一个表示，更加便于传输与显示。

由于 std::map 的动态特性，该结构在本系统中被广泛使用。

（1）函数通常需要很多输入，而命名、维护这些输入是一个很大的难题。每当一个处理函数需要更多输入的时候，如果不使用 ConfigFile 可能需要在声明、实现、调用等处做相应的修改。这会花费大量的工作量。另外，使用一个需要十多个输入的函数本身也很麻烦，是一个不小的问题。

（2）ConfigFile 被设计成可以很方便地动态创建、修改、保存的一个数据结构，很有利于文字型数据的处理。

（3）ConfigFile 结构可以使得程序更加直观易懂。

4.4 通信模块

为了保证用户与服务器交互的传输安全，并将 VO 与 CI 管理包装成 WEB 服务的形式，本系统的通信模块基于带 SSL 的 SOAP 协议建立。

4.4.1 SOAP 简介

SOAP（Simple Object Access Protocol，简单对象获取协议），用于 WEB 服务中发送结构化的信息。SOAP 中交互的信息是基于 XML 结构的。通过 SOAP 可以方便地完成 Web 服务的交互接口设计。

SOAP 通过 WSDL（Web Services Description Language，Web 服务描述语言）生成满足要求的通信接口。

WSDL 用于描述与发布 Web 服务，描述如何与 Web 服务进行通信，如输入与输出、地址等等。WSDL 采用 XML 结构编写。SOAP 中 wsdl2h 编译器可以将 wsdl 文件翻译成 SOAP 接口对应的头文件，从而大大简化 SOAP 通信接口的编写工作。

4.4.2 SSL 简介

SSL 提供了一套信息安全传输的机制。SSL 保证了用户和服务器都是真实的用户和服务器，信息不可伪造，不可被窃听，从而保证了信息传输的安全。

SSL 的基础是 PKI。PKI 从算法上支持了 SSL 的这些特性。在 PKI 中，每个用户都有一个公开的公钥证书和一个秘密的私钥证书，同样，签发证书的证书中心也有自己的公钥和私钥。

证书的签署：用户生成私钥和一个请求签发的 request 证书，将 request 证书发送至证书中心。证书中心用自己的私钥签署用户的 request 证书，生成用户的公钥，并发送回用户。

证书的使用：用户发送消息时用自己的私钥和对方的公钥对消息进行加密。解密时只能用公钥和对方的私钥解密。对于一个恶意用户，由于不知道二者中任何一人的私钥，既不能伪造成发送方，也不能伪造成接收方。

每个证书都有唯一的主题（Subject），在证书中占有单独的一行。如：

Subject: C=CN, ST=Beijing, L=Beijing, O=Tsinghua, OU=riit, CN=TinyCA/emailAddress=grrrgrrr@163.com

而 Subject 也成为验证证书持有人的方法。Subject 可以通过函数 X509_get_subject_name 读取出来。

在 SOAP 中提供了对 SSL 的支持。

4.4.3 服务器端的通信模块

在本系统中，服务器端的服务模型为接收一个 vector<string>，返回一个 vector<string>。

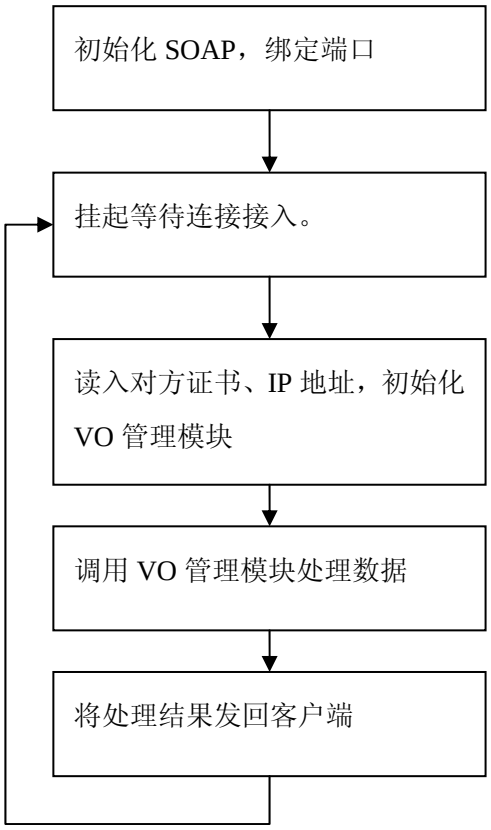

图4.3　服务器端通信模块的流程

SOAP 通信的服务端绑定并侦听一个端口。每当这个端口接收到一个数据包时，SOAP 首先对对方进行身份认证。身份认证通过后调用事先编写的一个回调函数进行处理。

在本系统中，客户端向服务器端发送请求，服务器处理请求并返回。处理客户端请求的函数是回调函数 ns__hdl。

ns__hdl 从 soap 中读取客户端的证书和 ip 地址，并依此初始化 VO 管理模块。之后 ns__hdl 将 soap 中接收到的信息发送给 VO 管理模块进行处理，将结果填入 soap 结构并返回。

4.4.4 客户端的通信模块

客户端的通信模块 Sender 完成客户端与服务器进行通信的工作。该模块发送指定的一个 vector<string>对象到服务器，并读取服务器返回的 vector<string>。

Sender 函数的流程图如图 4.4 所示。

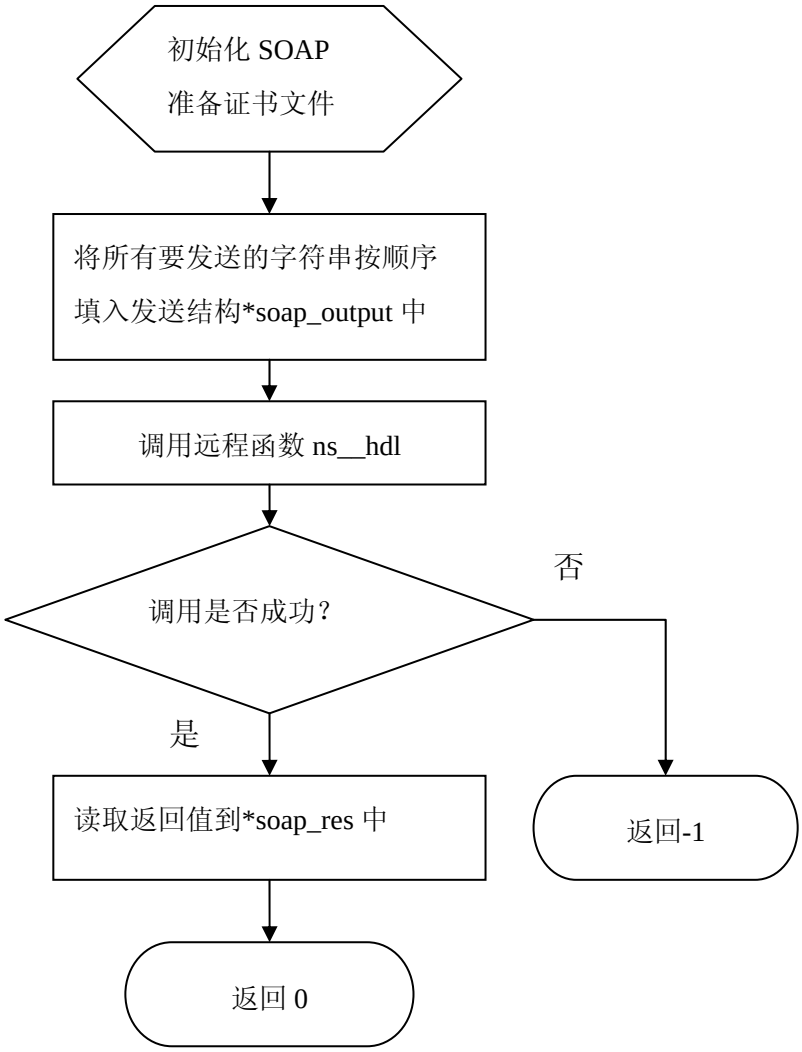

图4.4　客户端通信模块的流程

4.5 数据库模块

为了提高系统的灵活性和可扩展性，数据库模块以 ODBC 为核心，并通过配置文件的帮助，使数据库模块能够根据数据库信息和 VO 与 CI 管理模块给出的命令结合，动态生成 SQL 语句。

4.5.1 ODBC 简介

ODBC 是一套访问数据库的标准 API 接口。ODBC 试图将数据库操作者与具体的数据库、编程语言乃至操作系统独立开，让操作数据库变成简单而通用的操作。

UnixODBC 是 ODBC 在 Linux 系统上的实现。本系统使用 UnixODBC 完成对数据库的操作，使得用户可以自由地选择数据库，并最大程度地使本系统与平台无关。

UnixODBC 通过配置文件对用户数据库进行访问。用户指定 Data source name、数据库的用户名和密码。UnixODBC 读取 odbc.ini 寻找对应 DSN 数据源的配置信息。

一个典型的 odbc.ini 如下。

```
[SERVERDB]
Description = SERVERDB
Trace       = On
TraceFile   = stderr
Driver      = MySQL
SERVER      = localhost
USER        =
PASSWORD    =
PORT        =
DATABASE    = SERVERDB
```

DSN 为 SERVERDB，在读取 DSN 配置时，UnixODBC 通过 Driver=MySQL 一项获知需要使用 MySQL 的驱动。然后 UnixODBC 读取 odbcinst.ini 查找 MySQL 的驱动信息。

odbcinst.ini

```
[MySQL]
Description     = MySQL driver for Linux
Driver          = /usr/lib/odbc/libmyodbc.so
```

```
Setup           = /usr/lib/odbc/libodbcmyS.so
FileUsage       = 1
```

从 odbcinst.ini 中可以得到驱动库位置，从而打开与指定数据库的连接。

在使用 UnixODBC 之前，需要预先配置上述配置文件。

4.5.2 动态生成 SQL 命令的设计

服务器的数据库模块（DB.cpp）是对 CppODBC 库的再一次封装。通过配置文件的配置，虚拟组织管理模块不需要知道数据表的具体结构，就可以完成表的插入、删除、查询以及更新等一系列功能。

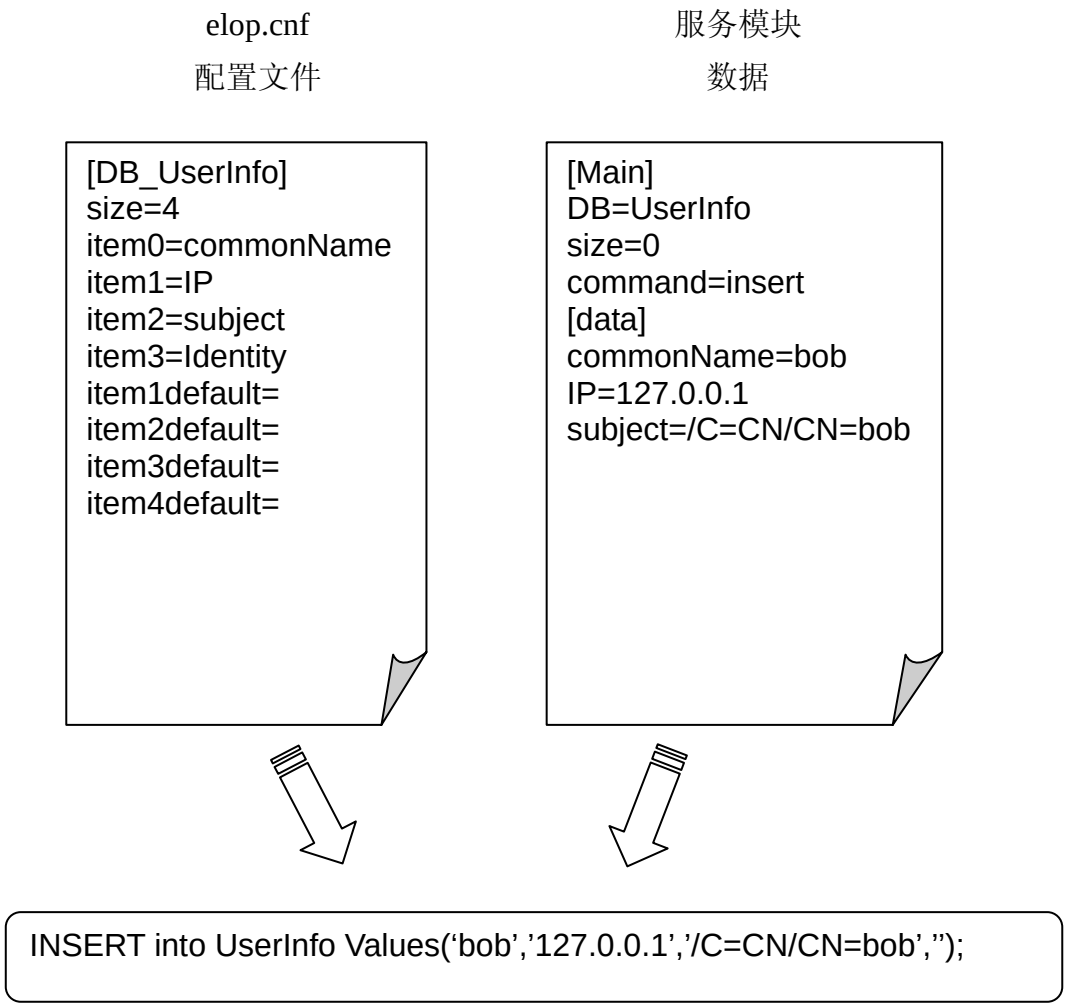

图4.5　数据库模块根据数据表结构动态生成命令

为了简化服务器端的操作，数据库模块为服务器量身定做了一套最简单的数据库访问方法。数据库模块中只有三个操作：初始化、执行数据库命令、关闭ODBC 连接。这样大大减轻了虚拟组织管理模块的负担。

在 SQL 命令中，服务模块与数据库信息是存在耦合的。比如 SQL 的插入命令“INSERT INTO TABLE VALUES(‘VALUE1’,’VALUE2’,’VALUE3’)”。为了生成这条命令，就需要知道数据库表的结构。而服务模块需要知道数据库表的结构，这对服务模块的组建以及今后对该系统的扩展都并不有利。为了实现服务模块与数据库信息的剥离，本系统中数据库模块可以通过读取配置文件获知数据库表格的结构，并可以通过表格结构动态生成 SQL 命令，而服务模块只需要向数据库模块发送用于生成 SQL 命令的数据。

以插入命令为例，如图 4.5，服务模块需要发送每个字段的值和数据库名。数据库模块读取配置文件，将每个字段的值填入合适的位置。对于服务模块没有指定的字段，数据库模块读取配置文件中指定的这个字段的默认值，并填入相应的位置。这样一来，不论数据库表的结构如何变化，服务模块仍然可以在兼容环境下正常工作。这就实现了服务模块与数据库信息的分离。

4.5.3 服务器端数据库表项设计

服务器的数据库模块（DB.cpp）是对 CppODBC 库的再一次封装。通过配置文件的配置，虚拟组织管理模块不需要知道数据表的具体结构，就可以完成表的插入、删除、查询以及更新等一系列功能。

为了简化服务器端的操作，数据库模块为服务器量身定做了一套最简单的数据库访问方法。数据库模块中只有三个操作：初始化、执行数据库命令、关闭ODBC 连接。这样大大减轻了虚拟组织管理模块的负担。

服务器端共有四个数据库表，UserInfo，保存 CI 中每个用户的信息；VOInfo，保存每个 VO 的信息，即 VO 登记表；Member，保存每个用户属于哪个 VO 的信息，即 VO 成员登记表；Message，保存消息信息，如成员变更时需要将 dn 和 ln 信息发送至客户端处的 Globus，而客户端并非一直在线，这就需要将发送的信息临时储存于 Message 表中。各个表格结构分别如下：

UserInfo(commonName varchar(50) NOT NULL, IP varchar(50) NOT NULL, subject varchar(200) NOT NULL, Identity varchar(50) NOT NULL);

VOInfo(VOName varchar(50) NOT NULL, Founder varchar(50) NOT NULL, Statement varchar(100) NOT NULL);

Member(user varchar(50) NOT NULL, VO varchar(50) NOT NULL, status varchar(50) NOT NULL, Membership varchar(50) NOT NULL, Reputation varchar(50) NOT NULL, SuccIntr varchar(50) NOT NULL, FailIntr varchar(50) NOT NULL, subject varchar(200) NOT NULL, additional varchar(50) NOT NULL);

Message(user varchar(50) NOT NULL, MessageType varchar(50) NOT NULL, Message varchar(300) NOT NULL, time int(50) NOT NULL);

4.6 VO 与 CI 服务模块

VO 与 CI 服务模块负责按照客户端发来的命令进行对应的 VO 和 CI 管理操作。

4.6.1 命令的接收与解析

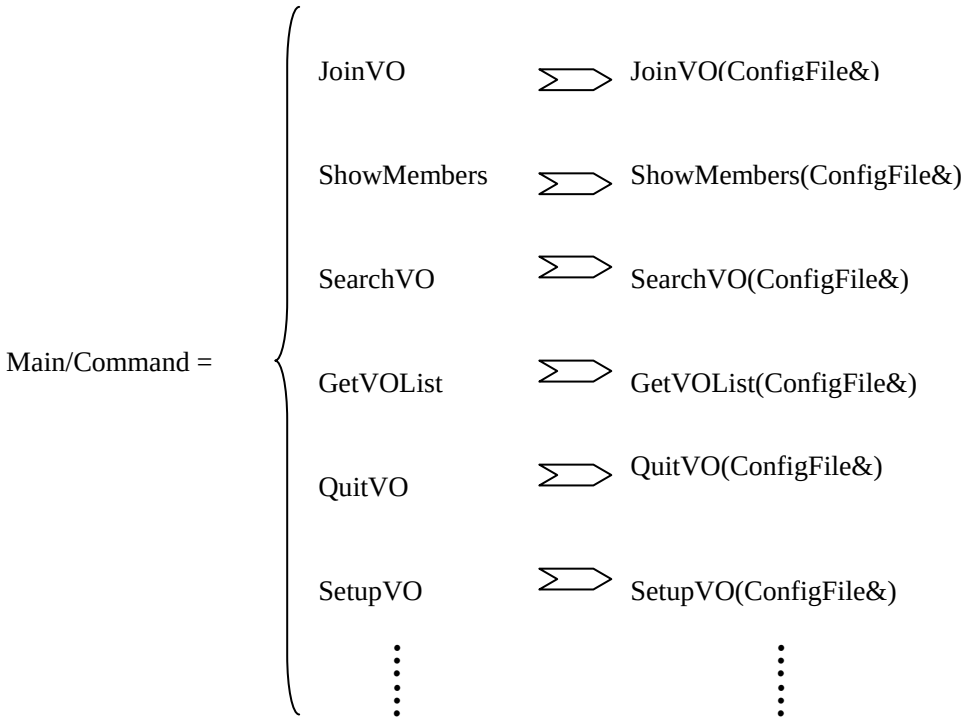

图4.6 命令的解析

VO 与 CI 服务模块从通信模块接收到用户发来的消息。消息是由 ConfigFile 转化成的 vector<string>结构，可以直接转换回 ConfigFile 结构。服务器分析 Main/Command 一项中的内容，并依据内容调用对应的函数进行处理。如图 4.6 所示。

对于暂时不存在的命令，程序会调用一个特定文件夹（默认是 bin 文件夹）下以 Main/Command 处字符串为名称的程序。这样如果需要对程序进行拓展，则只需把新编写的命令处理程序放置在指定目录（bin）下，那么就可以通过 Main/Command={文件名}来使用命令。这赋予了本系统一定的扩展能力。

4.6.2 服务器与客户端之间的表单机制

由于一个命令在处理过程中往往需要用户提供很多其他的信息，而如果将需要哪些信息全记录在客户端中，会大大增加客户端的负载，也使得服务器端在更新的过程中需要对应更改客户端，造成很大的麻烦。

因此绝大部分信息应当被记录在服务器中，并在需要时发给客户端，让用户填写对应的内容。可以像 HTML 中表单一样，由服务器设计好发给客户端，客户端将填好的表单发回给服务器。

在本系统中引入了这样的机制。服务器端保存很多客户端可能会用到的表单。客户端通过 GetTable 命令+表单名称可以获取指定的表单。随后用户根据表单上给出的规则，生成一条命令，并将信息填入命令的对应位置发给服务器。

每个表单都有一个固定的结构。表单同样使用 ConfigFile 来保存，Main/Name 项指明表单名称；Main/Title 为表单标题；Main/Affair 项指明客户端返回表单时 Main/Command 项的内容。Main/Size 为表单里有多少项需要用户填写。Itemx/ID 为返回时该项在命令的位置，Itemx/Name 为该项对用户显示的名称，Itemx/Maxlength 为该项的最大长度，Itemx/Suggestion 为该项对用户的提示。举例如下：

JoinVO.cnf

```
[Main]
Permission=Member
Name=JoinVO
Title=JoinVO
Affair=JoinVO
Size=2
```

```
[Item0]
ID=Main/VOName
Name=VOName
Maxlength=30
Suggestion=Name of VO
[Item1]
ID=Main/Additional
Name=Account
Maxlength=100
Suggestion=Your account for VO
```

该表单对应的命令包含两项，一项为 VO 名称，放在 Main/VOName 处，另一项为本地帐户名，放在 Main/Additional 处，供处理 JoinVO 命令的函数读取。

4.6.3 服务器端的权限机制

该系统需要严格区分 VO 创建者、管理员和普通成员等各个角色的权限。管理员不能够拥有创建者的权限，普通成员也不能获得管理员的权限。因此一个细致的权限系统十分必要，在服务器端每每做一项数据库操作时，都需要检查用户是否具有相应的权限等级。

VO 创建者：解散 VO，批准成员，指定管理员、解职管理员、踢出成员（除自己）

VO 管理员：批准成员、踢出成员（除创建者和管理员）、退出 VO

VO 成员：退出 VO

通过函数 QueryRing()和 QueryRingInVO()，可以查询任何一个人在 VO 中的身份和在 CI 中的身份。通过身份可以明确地辨别一个人是否有使用某一项功能的权力。

4.6.4 服务函数的设计

VO 与 CI 服务模块主要提供了以下服务函数：

（1）初始化(Init)

读取位于$ELOPHOME/conf/elop.conf 的配置文件

初始化 ODBC 接口

从输入读取用户证书信息
查询是否存在该用户？
存在：更新用户的 IP 地址
不存在：向数据库 UserInfo 表插入该用户的记录

（2）解析用户送来的指令(userCommand)
若用户命令长度为 0 则直接返回
经过文字过滤器之后将用户命令转化为 ConfigFile 格式
根据用户命令中 Main/Affair 一项决定调用哪个函数进行处理

（3）查询用户加入的 VO(GetVOList)
QueryRing 查看是不是 CIAdmin。如果是直接返回所有 VO
select VOName from VOInfo
对于一般用户，查找所有加入的 VO，并查找 Membership 看是不是 VO 管理员
select VOName,Membership from Member where user=’用户名’ and status=’Admitted’
如果 Membership 是 VOAdmin 或 VOFounder，就在 VOAdmin/VOx 上写 VOAdmin 或 VOFounder
在 VOLIST/VOx 上填写所有 VO 名称
在 Main/Size 上填写一共有多少个 VO
在 Main/Content 上标注 VOLIST
查收用户收到的所有组织成员变更的消息。

（4）提交加入 VO 的申请(JoinVO)
首先执行 QueryRinginVO 查询用户在 VO 中的权限等级。如果已经是 VO 成员就没必要提交了，返回。
如果是新用户就向 Member 表插入一条记录，说明该用户的状态为 Submitted。
如果是老用户就通过 Update 命令，将用户状态更新为 Submitted。
返回提交成功消息。

（5）退出 VO(QuitVO)

首先执行 QueryRinginVO 查找用户在 VO 中的身份。只有 Admin/Member 才能使用该命令

修改指定用户的状态

update Member set status='Retired',additional='%s' where user='%s' and VO='%s' and status='Admitted'

查询 VO 中所有成员

select user,subject from Member where VO='%s' and status='Admitted'

查询指定用户的信息

select subject from Member where VO='%s' and user='%s'

对所有成员，在 Message 表中加入移除指定用户的消息

对该成员，在 Message 中加入移除其他所有用户的消息

对该成员，在 Message 中加入移除自己的消息

查找用户收到的所有消息

（6）按照用户的指令查询 VO（SearchVO）

select VOName,Founder,Statement from VOInfo where VOName like '%s'

其中%s 为用户输入的条件。即用户输入的条件就是 SQL 的查找条件。

并填写到返回值 VOx 的对应位置.

在 Main/Size 填写查到了多少记录.

在 Main/Content 处标注结果为 SearchVOResult。

（7）查找指定 VO 中成员（SearchMember）

QueryRinginVO 查询用户身分。如果用户不是 VO 成员则不允许查询

按照用户的指令查询某个 VO 里的用户。查询的是 subject 字段。格式与证书格式相同.

select user,Membership,Reputation,subject,SuccIntr,FailIntr from Member where status='Admitted' and VO='%s' and subject like '%s'

并填写到返回值 Memberx 的对应位置。

在 Main/Size 填写查到了多少记录。

在 Main/Content 处标注结果为 SearchMemResult。

（8）列出指定 VO 中所有成员（QueryMember）

把送来的 VONAME 返回。填写在 Main/VOName 中。

QueryRinginVO 查询用户在本 VO 中的权限，并填写到 Main/Membership 里

查询该 VO 所有 Admitted 的用户的信息

select user,Membership,Reputation,SuccIntr,FailIntr,subject from Member where status='Admitted' and VO='%s'

并填写到返回值 Memberx 的对应位置

在 Main/Size 填写查到了多少记录

在 Main/Content 处标注结果为 MemberLIST

（9）获取表单（GetTable）

根据送来的 Main/Table，在服务期配置文件中的 Forms 域查找对应表格的相对路径。

读取该路径指定的表单并送回。

（10）查询该用户提交申请的所有 VO 的信息（QueryRequests）

select VO,status,additional from Member where user='用户名'

并填写到返回值 Itemx 的对应位置

在 Main/Size 填写查到了多少记录

在 Main/Content 处标注结果为 RequestLIST

（11）查询对某个 VO 的所有申请的信息（Requests）

QueryRinginVO 查询用户身分。如果用户不是 VO 管理员或创建者则不允许查询

select user,status,additional from Member where VO='$VONAME' and status='Submitted'

并填写到返回值 Itemx 的对应位置

执行评分函数，收集信息并对申请加入的每个用户进行评分

在 Main/Size 填写查到了多少记录

在 Main/Content 处标注结果为 Requests

将送来的 VOName 返回，Main/VOName。

（12）踢出成员（KickMember）

QueryRinginVO 查找用户在 VO 中的身份。只有管理员或创建者才能使用该命令修改指定用户的状态，踢出该成员。

update Member set status='Removed',additional='%s' where user='$USERNAME' and VO='$VONAME' and status='Admitted'

查询 VO 中所有成员

select user,subject from Member where VO='%s' and status='Admitted'

查询指定用户的信息

select subject from Member where VO='%s' and user='%s'

对所有成员，在 Message 中加入移除指定用户的消息

对该成员，在 Message 中加入移出其他所有用户的消息

对该成员，在 Message 中加入移出自己的消息

查找用户收到的所有消息

（13）批准某个成员的申请（AddMember）

QueryRinginVO 查找用户在 VO 中的身份。只有管理员或创建者才能使用该命令

查询当前 VO 中所有成员

select user,subject from Member where VO='%s' and status='Admitted'

修改指定用户的状态

update Member set status='Admitted',Membership='Member',additional='' where user='%s' and VO='%s' and status='Submitted'

查询指定用户的信息

select subject from Member where VO='%s' and user='%s' and status='Admitted'

对所有其他成员，在 Message 中加入添加指定用户的消息

对该成员，在 Message 中加入添加其他所有用户的消息

对该成员，在 Message 中加入添加自己的消息

查找用户收到的所有消息

（14）创建虚拟组织（SetupVO）

向 VOInfo 表添加一条新虚拟组织的记录

向 Member 表添加一条记录，说明该用户是该 VO 的创建者

向 Message 表添加一条 Founder 加自己的消息

查找用户收到的所有消息

（15）取消虚拟组织（StopVO）

QueryRinginVO 查询用户信息。只有 VO 创建者或 CI 创建者才可以使用这条命令

查询 VO 中所有用户信息

select user,subject from Member where VO='%s' and status='Admitted'

删除 Member 中所有关于这个 VO 的成员

delete from Member where VO='%s'

删除 VOInfo 中这个 VO 的信息

delete from VOInfo where VOName='%s'

向 Message 中添加每个 VO 成员删除所有 VO 成员的消息

查找用户收到的所有消息

（16）提升管理员（AssignAdmin）

QueryRinginVO 查询用户信息。只有 VO 创建者才可以使用这条命令

update Member set Membership='Admin' where status='Admitted' and user='%s' and VO='%s' and Membership='Member'

将用户身份更新为管理员。

（17）解职管理员（DisposeAdmin）

QueryRinginVO 查询用户信息。只有 VO 创建者才可以使用这条命令

update Member set Membership='Member' where status='Admitted' and user='%s' and VO='%s' and Membership='Admin'

将用户身份更新为普通用户。

（18）查找用户收到的所有消息

select MessageType,Message from Message where user='用户名' order by time Asc

在 TODOx/MessageType 和 TODOx/Message 上填写对应信息

在 TODO/Size 上填写有多少条消息

从数据库中删除读取过的消息

delete from Message where user='用户名'

4.6.5 评价接口的设计

该系统为评价函数预留了一个评价接口。评价接口作为 VO 管理模块中的一个私有函数存在。客户端不能够直接访问该评价函数。评价函数当用户或资源提供者提交加入请求的审批过程中调用，返回对指定用户或资源提供者的评价信息。

该评价接口具有访问数据库的权力，可以实现大多数的评价方法。

4.7 客户端模块

客户端模块承担了与客户的交互工作，包括整理用户命令向通信模块发送命令，以及将服务器的信息显示出来。

另外，为了与 Globus 进行连接，客户端还需要负责根据服务器端报告的 VO 成员变动修改对应的 grid-mapfile。

本系统中提供了命令行版的简单客户端形式，以及网页版相对复杂的客户端形式。

4.7.1 命令行版客户端的设计

命令行版客户端循环接收并发送用户给出的命令。服务器端返回后客户端对服务器返回的信息进行解析，并以用户友好的方式显示出来。

```
grrr@grrr-desktop:~/www/Terminal$ ./terminal2 Bob wangz

>ShowRequest LIGO
Requests

commonName      Alice
Status          Submitted
AdditionalInfo  LIGO scientist

>Approve LIGO Alice
Your CommonName:Bob
Send?[Y/N]:y
Message
Confirmed

>
```

图4.7　命令行客户端的用户界面

命令行版客户端命令如下：

表4.1　客户端可能用到的指令

命令	参数	说明
GetVOList		获得 VO 列表
ShowMember	$VONAME	显示$VONAME 成员列表
JoinVO		加入一个 VO
SetupVO		创建一个 VO
SearchVO	$Condition	查找 VO
MyRequest		列出自己加入/申请加入了哪些 VO
SearchMember	$VONAME $Condition	查找$VONAME 中的成员
QuitVO	$VONAME	退出$VONAME 这个 VO
Kick	$VONAME $MEMBER	把$VONAME 中的$MEMBER 成员踢出 VO
Approve	$VONAME $MEMBER	批准$VONAME 中$MEMBER 成员的加入申请
ShowRequest	$VONAME	查看所有对$VONAME 的加入申请
Set	+(-)Admin $VONAME $MEMBER	将$VONAME 中的$MEMBER 成员提升+Admin 或取消-Admin 管理员权限
StopVO	$VONAME	终止$VONAME 这个 VO
ForceStopVO	$VONAME	强制终止$VONAME 这个 VO（CI 管理员命令）
Exit		退出 terminal

对于大多数命令，客户端首先向服务器请求对应命令的表单，然后将表单交与用户进行填写。用户填写完毕后，表单被发回服务器，最后服务器将处理结果返回。

4.7.2 网页版客户端的设计

网页版客户端通过 PHP 实现。将命令转化为超链接的形式，在用户点击对应位置时向服务端发送。将服务器返回的信息解析成 HTML 格式显示出来。

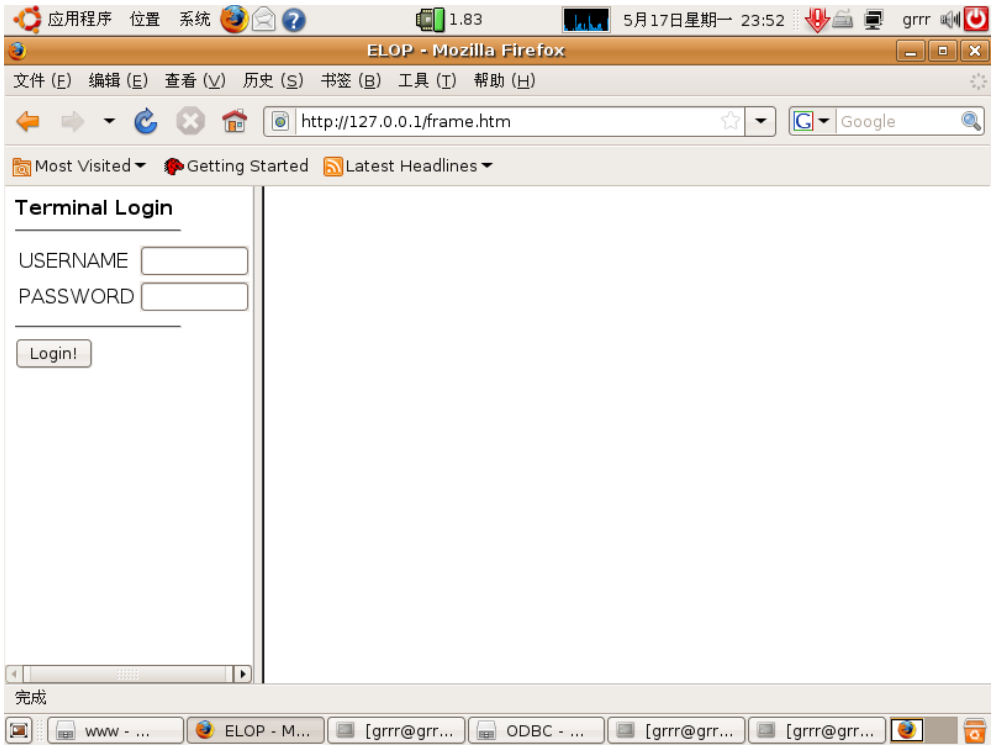

图4.8　网页版客户端（登录页面）

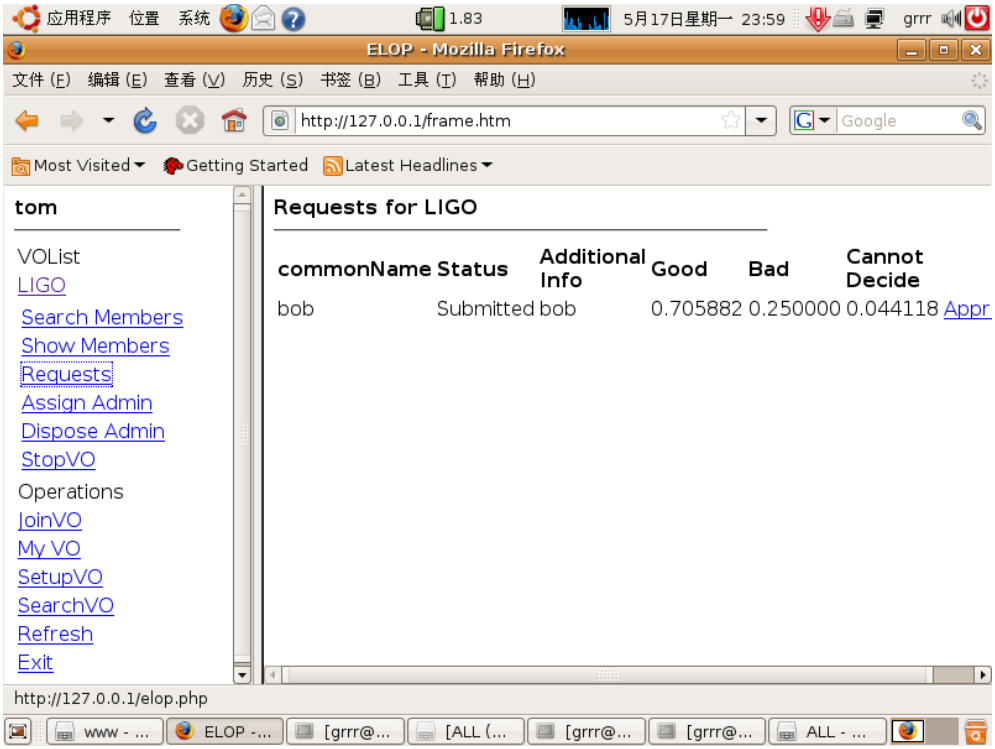

图4.9　网页版客户端（使用页面）

PHP 是一种免费、开源的 HTML 脚本语言。PHP 页面首先通过一个 PHP 解析器翻译为 HTML 页面，然后通过浏览器显示出来。

使用 PHP 可以很轻松地实现动态页面。对同一个页面，由于访问者以及数据的不同，解析出来的 HTML 页面可以大不一样。

网页版客户端命令的处理过程与命令行版客户端比较类似，不同的是网页版使用起来更加直观，但需要装额外的软件。

页面分为两栏。左栏为 VO 管理的操作，右栏为服务器返回的结果。

4.8 系统的打包与安装

VO 与 CI 服务模块为了使系统的结构更加清晰易读，并作为 ELOP 软件包的一部分便于后续开发，该系统按照 ELOP 软件包的目录结构规范进行了打包处理。

4.8.1 目录结构

该系统在包中的目录结构如图 4.10 所示：

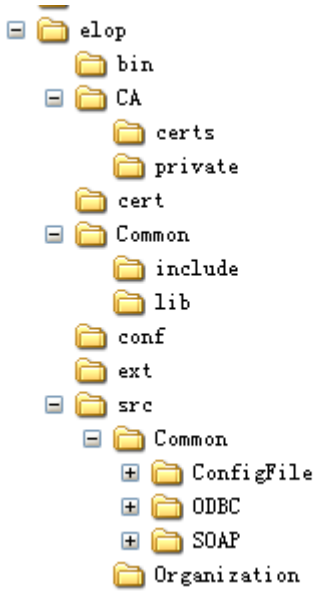

图4.10 ELOP打包结构

程序的源文件放在/elop/src/目录下。/elop/src/Common/目录下放的是公用，即其它程序开发时可能会用到的的模块，有 ConfigFile 类型模块、SOAP 通信模块

和 ODBC 数据库访问模块。组织管理模块属于 elop 中的 O，即 Organization 层，因此放在 Organization 模块下。

/elop/bin/目录下放编译好的二进制文件。如服务器 hdlserver，客户端 terminal。

/elop/CA/目录下是证书中心，可以签发用户证书。

/elop/cert/目录是默认的证书目录。

/elop/Common/下是这个系统的库和头文件。

/elop/conf/目录下是全部的配置文件和文本文件。

/elop/ext/目录中是用到的外部包。如 gSOAP。

4.8.2 系统的安装

本系统的依赖关系如图 4. 11 所示

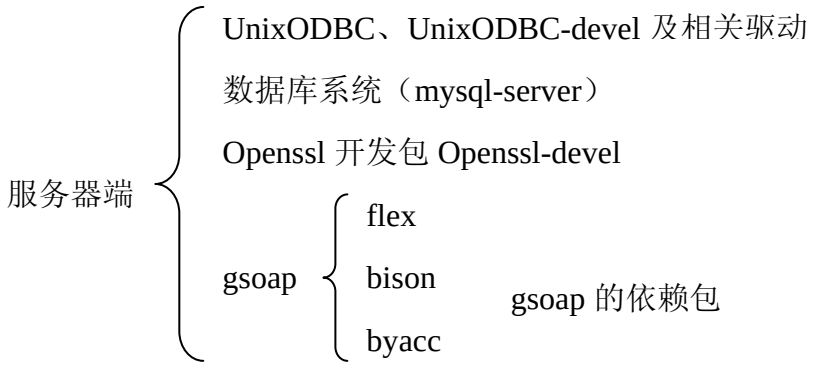

图4. 11 系统的软件依赖关系

从一个刚安装好的 CentOS 系统安装服务器的流程如下

（1）安装依赖包

yum install gcc-c++ unixODBC mysql-connector-odbc mysql-server flex bison byacc openssl-devel

（2）初始化 MySQL

mysql_install_db

初始化 mysql-server

mysqld_safe

启动 mysql-server

mysqladmin –u root password 524288

设置 root 密码

mysql –u root –p <db.sql

运行 db.sql 脚本，创建数据库

（3）配置 ODBC

配置 odbcinst.ini odbc.ini

默认的 odbc.ini 在/elop/目录下，覆盖/etc/目录中的 odbc.ini 即可

（4）编译安装

编译/ext/gsoap/处的 gsoap

export ELOPHOME

设置 elop 位置

export SOAP_DIR

设置 gsoap 位置

Make 运行/elop/src/下的 Makefile 编译 hdlserver 和 terminal

4.8.3 系统的运行

在任何程序运行之前，请确保 ELOPHOME 被设置在了 ELOP 软件包的根目录下。

服务器端的运行：/elop/bin/目录下 hdlserver。无参数。

命令行版客户端的运行：/elop/bin/目录下 terminal。参数 1 为证书名，参数 2 为证书密码。举例，若证书名为 bob，证书密码为 123456，则用命令

./terminal bob 123456

就可以运行命令行版客户端。

网页版客户端不需要运行，直接在浏览器栏输入对应的网址即可。

第5章 动态虚拟组织管理系统与 Globus 的协作

在安装并配置了动态虚拟组织管理系统与 Globus 的计算机上，通过启用 grid-mapfile-add 和 grid-mapfile-delete 命令对 Globus 的 grid-mapfile 进行修改，即可实现该动态虚拟组织管理系统与 Globus 的连接。

在测试中，我们使用四台计算机进行动态虚拟组织管理系统与 Globus 的协作测试。这四台计算机的 ip 地址分别是 166.111.137.17（grrr）、166.111.137.18（jinchun）、166.111.137.19（hanqiu）和 166.111.137.170（ligo）。其中 166.111.137.17 是 VO 与 CI 服务器，其余三台计算机上运行客户端。三台运行客户端的计算机上的用户分别有各自的 Globus 证书用于 Globus 身份识别，以及与服务器的安全通信与身份验证。三台客户端上的证书分别为 bob、tom 和 alice。使用 SSH 工具连接到这四台计算机上并进行操作。如图 5.1 所示。

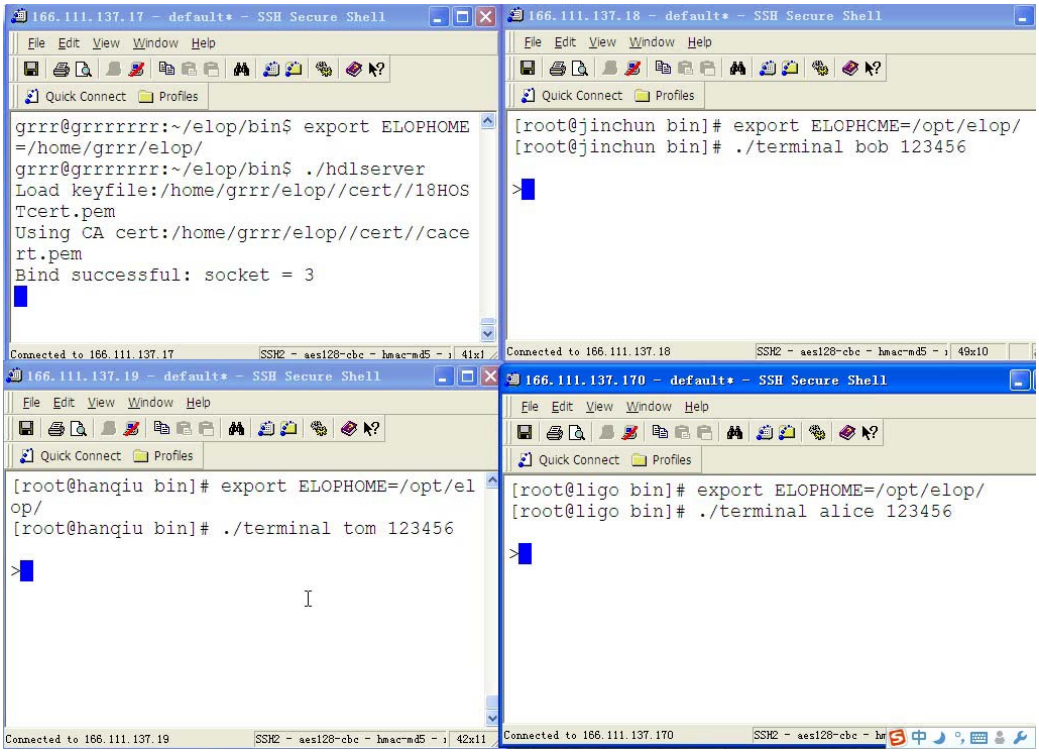

图5.1 SSH登录启动服务器与三个客户端

首先 tom 创建一个 VO，名为 LIGO。tom 在本地打开帐户 tom 供 VO 中的其它成员通过 Globus 进行访问。如图 5.2 所示。

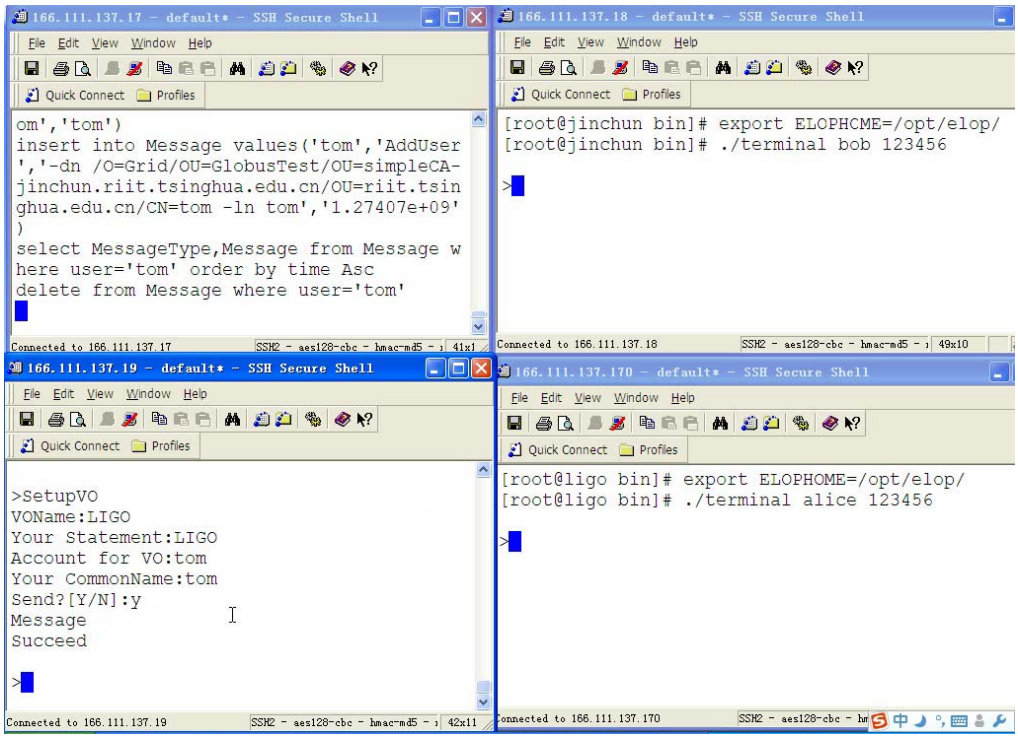

图5.2　tom创建VO：LIGO

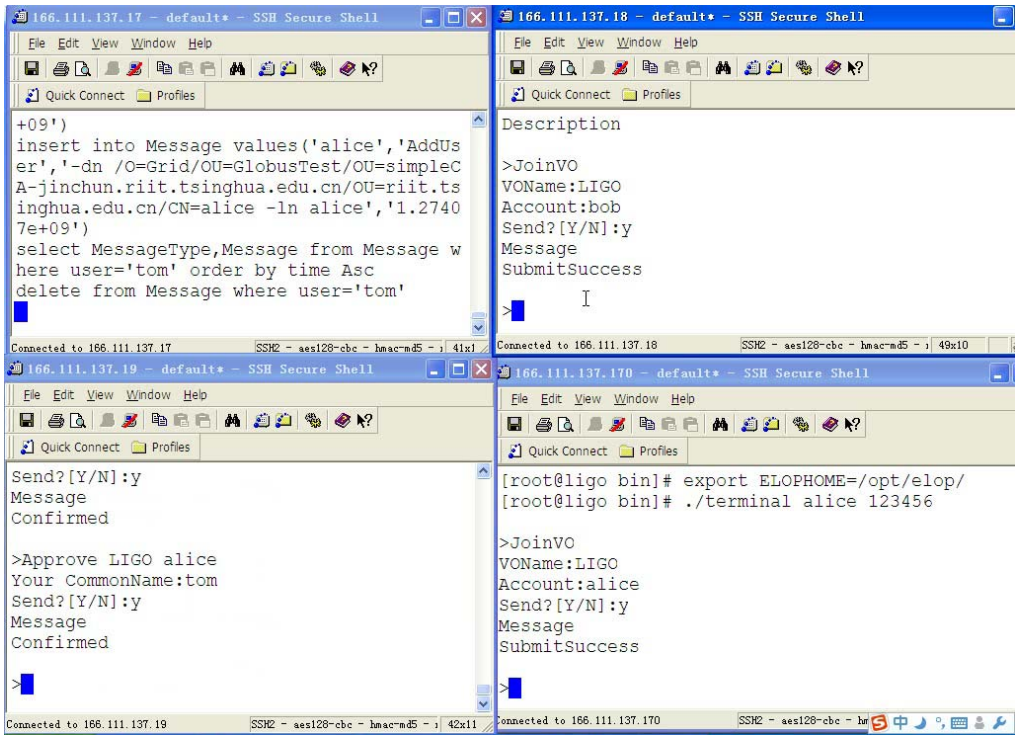

图5.3　bob和alice加入LIGO

随后，bob 和 alice 加入 LIGO，并各自打开帐户 bob 和 alice 供虚拟组织成员进行访问。tom 根据评价系统给出的评价值批准二者的加入请求。如图 5.3 所示。在批准加入申请之后，客户端再执行一条命令，就可以收到服务器发送的 VO 成员变化信息。根据变化信息，客户端执行 grid-mapfile-add 或 grid-mapfile-delete 命令。在本例中，bob 加入 alice 到本地帐户 bob 的映射，alice 加入 bob 到本地帐户 alice 的映射。从而使二者之间可以相互访问。

alice 使用 Globus 执行 bob 计算机（jinchun）上的/bin/date 命令显示当前时间。命令成功返回正确的结果。如图 5.4 所示。说明该动态虚拟组织管理系统成功地修改了 grid-mapfile，实现了与 Globus 的对接。

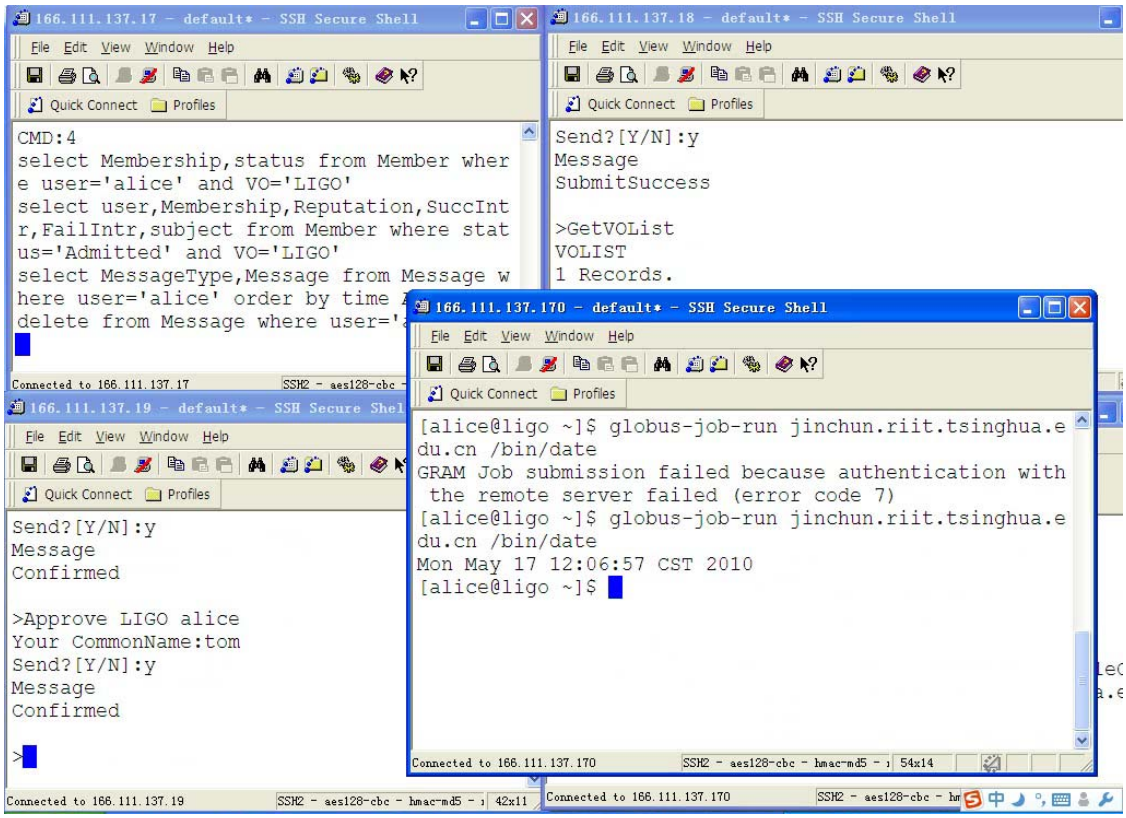

图5.4　alice访问bob计算机上的程序

VO 的创建者 tom 认为 alice 的评价值过低，而将 alice 踢出 VO。如图 5.5 所示。

随后，alice 再通过 Globus 的 globus-job-run 命令执行 bob 计算机上的/bin/date 命令时，出现了错误 7，身份认证失败。这个错误当 grid-mapfile 中没有 alice 的

映射条目时才会出现。这说明动态虚拟组织管理系统成功地使用 grid-mafile-delete 将 alice 从 bob 的访问映射表中移除。如图 5.6 所示

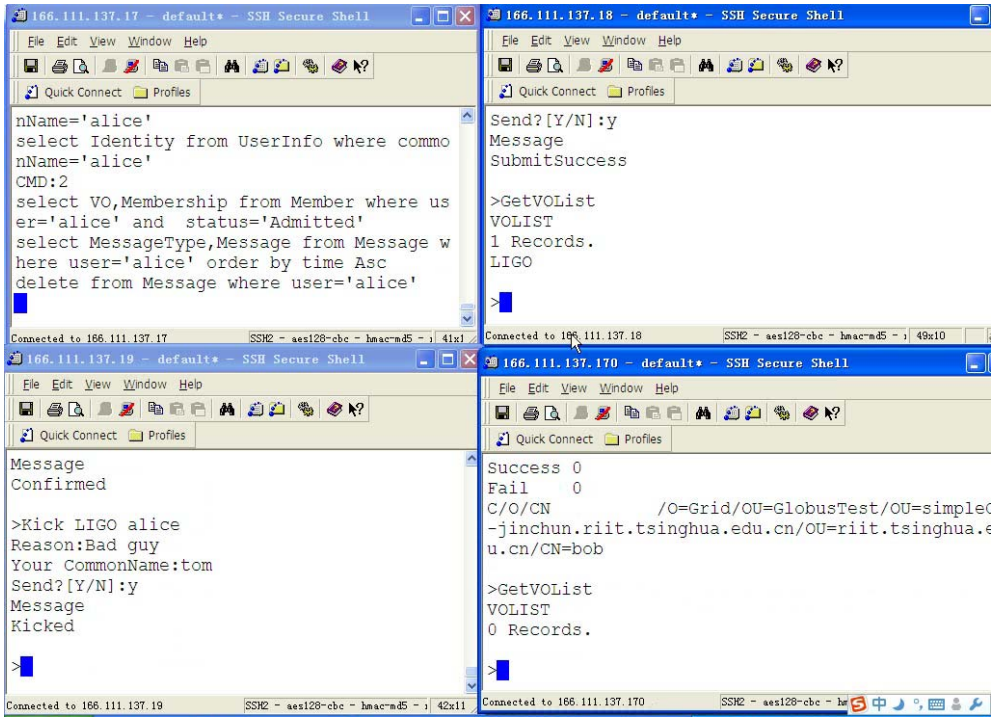

图5.5 tom踢出alice

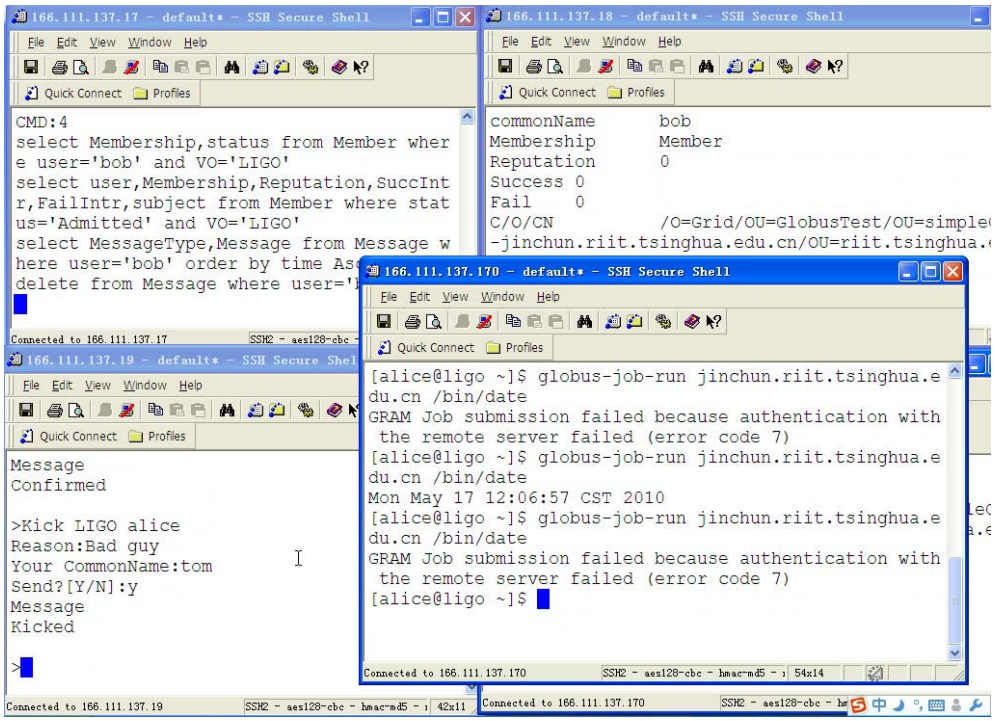

图5.6 alice失去bob计算机的访问权限

之后 alice 再次加入虚拟组织 LIGO，随后项目结束，tom 通过 StopVO 命令解散了该虚拟组织。

虚拟组织关闭后，alice 和 bob 之间都不能够通过 Globus 相互访问（错误 7）。如图 5.7 所示。

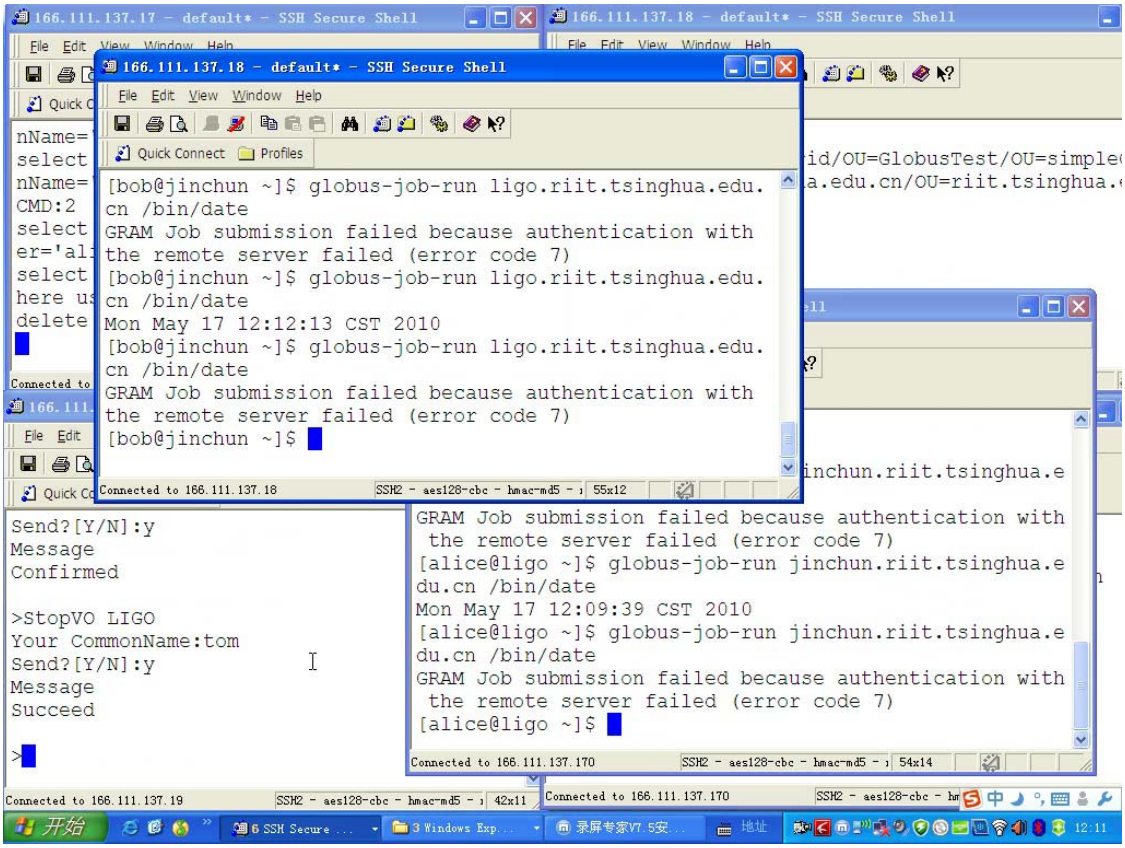

图5.7　tom关闭虚拟组织，bob和alice不能相互访问

整个过程直观地说明了动态虚拟组织管理系统与 Globus 的协作流程。

虚拟组织内部的成员能够访问虚拟组织内部的资源，而虚拟组织外的用户不能访问虚拟组织内的资源。通过对加入与退出虚拟组织的控制，该系统可以根据网格项目目标以及评价值调整用户和资源提供者的访问权限，达到了细粒度网格权限管理的目标。

该动态虚拟组织管理系统成功地完成了 Globus 网格中访问控制与权限管理的相关功能，说明动态虚拟组织管理的方案是切实有效的。

第6章 总结

本文针对网格粗粒度管理僵化的问题，提出了解决该问题的动态虚拟组织管理方案。该方案按照虚拟组织的方式管理网格计算项目，按照虚拟组织成员关系控制用户访问的权限。该方案中还包括了用于评价用户和资源提供者的评价函数，作为虚拟组织添加新成员的参考，使得虚拟组织更有针对性。对于该动态虚拟组织的管理方案，本文提出了一个能够实现该方案的客户端-VO 服务器-CI 服务器框架。

本文给出了这个框架的一个具体实现——动态虚拟组织管理系统。该系统是针对 Globus 网格平台进行设计的网格中间件，通过根据虚拟组织成员变化动态修改 grid-mapfile，控制用户的访问。该系统与 Globus 的结合实现了 Globus 平台上细粒度的访问控制，达到了用动态虚拟组织的管理方式管理网格计算的目的。

实际应用表明该系统实现了细粒度的网格项目管理，证明了动态虚拟组织管理方案的有效性。

插图索引

表格索引

参考文献

[1] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, San Francisco, 1998.

[2] LIGO，http://ligo.org.cn/

[3] GIMPS，http://www.mersenne.org/

[4] VOMS，http://edg-wp2.web.cern.ch/edg-wp2/security/voms/voms.html

[5] GUMS，https://www.racf.bnl.gov/Facility/GUMS/1.3/

[6] VO Services Project，http://computing.fnal.gov/docs/products/voprivilege/

[7] Z. Wang and J. Cao, Committee-based Evaluation and Selection of Grid Resources for QoS Improvement. Proc. 10th IEEE/ACM Int. Conf. on Grid Computing, Banff, Alberta, Canada, 138-144, 2009

[8] 曹军威，“赛百平台及其技术挑战”，国际学术动态[J]，2010 年第 2 期，P38-42。

[9] I. Foster, C. Kesselman, Steven Tuecke, The Anatomy of the Grid, Intl J. Supercomputer Applications, ISSU 2150, pages 1-4, 2001.

致 谢

感谢曹老师一直以来的悉心指导。曹老师对网格计算前沿领域的准确把握，极大地帮助了我确定选题方向和开展相关工作。

实验室的王震师兄从系统的架构到开发都给了我非常十分具体的指导。最后的管理系统部分是在王震师兄之前的工作基础上建立起来的。在此感谢王震师兄在理论和系统方面给予我的大量帮助。

万宇鑫师兄在 ELOP 系统框架与程序打包方面给了我很多有价值的建议，这些建议也很大地帮助了我完成软件的打包工作。在此一并感谢。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：林筱　　日　期：2010.7.1

附录 A 外文资料的调研阅读报告（或书面翻译）

调研阅读报告题目（或书面翻译题目）

写出至少 5000 外文印刷字符的调研阅读报告或者书面翻译 1-2 篇（不少于 2 万外文印刷符）。

参考文献（或书面翻译对应的原文索引）

[1] Sepandar D. Kamvar, Mario T. Schlosser, Hector GarciaMolina. “The EigenTrust Algorithm for Reputation Management in P2P Networks”, International World Wide Web Conference, 2003, ISBN:1-58113-680-3, Pages: 640 - 651

翻译第一章至第五章。

针对 P2P 网络声誉管理的 EigenTrust 算法

摘要：

P2P 文件共享网络，作为一种信息共享和发布网络，正受到越来越多的关注。然而最近的经验表明，这种网络匿名，开放的本性为有自我复制特性的假资源提供了一个近乎完美的传播环境。

我们描述了一个能够在 P2P 文件共享网络中减少假文件下载量的算法。这种算法根据节点历史的上传数据给每个节点一个全局信任值。基于幂迭代法，我们提出了一个分布且安全的计算全局信任值的方法。让这些节点按照全局信任值选择下载源，网络可以有效地识别并分离出恶意节点。

在仿真实验中，这个声誉系统"EigenTrust"明显地降低了网络中假资源的数量。即便在恶意节点联合起来试图颠覆系统的情况下，该算法也起到了很好的效果。

1 介绍

P2P 文件共享网络，相对于传统的客户端-服务器网络，在数据分发方面有很多优点。前者更加稳定，可扩展，资源更加多元化。然而，P2P 文件共享网络开放、匿名的天性导致了节点对其发布的内容不负有责任，从而为恶意节点攻击网络开启了大门。

截至目前，已经观察到了匿名恶意节点对现在流行的 P2P 网络的攻击。作为例子，恶意用户已经在用这些网络传播一种名为"VBS.Gnutella"的蠕虫病毒。这种病毒将自己拷贝到 Gnutella 的程序目录下，然后修改 Gnutella.ini，允许.vbs 文件的共享，从而进行传播。假文件的攻击是更常见的情况。恶意节点对几乎所有的查询都返回一个被篡改过的乃至不能工作的“诱饵文件”。

有人提出未来 P2P 系统的发展将很大程度地依赖于新的方法，保证节点能够获得可靠的关于它们正在接收的资源的信息。在这样的环境中，试图分辨提供假文件的恶意节点比试图分辨假文件本身更有意义。因为如果不封禁恶意节点的话，它们能够轻易地产生大量的假文件。我们给出了一种方法，在网络中每个节点 i 都被给予一个独一无二的全局信任值，反映出网络中其他所有节点与节点 i 的交互经验。在我们的方案中，网络中其他所有节点都通过一种分布而对称的方法参与计算全局信任值，并使网络开销最小。并且，我们描述了如何保证计算的安全性，最小化恶意节点通过撒谎而达到自己目的的概率。最后，我们展示了如何用这些信任值辨别出在提供被网络中其他节点认为是不合适的资源的节点，并有效地把它们分离出这个网络。

2　设计上的考虑

对任何一个 P2P 声誉系统，有五个问题必须注意。

1　这个系统应该自治

自治是说，共享文件的道德规范应该由网络中的用户决定，而不是某个权威机关。

2　这个系统应该维持匿名

每个节点的声誉应该与一个不透明的标识符联系起来（例如 Gnutella 中的用户名），而不是一个与网络外部相联系的一个身份（如节点的 IP 地址）。

3　这个系统不应给新成员额外的利益。

这是说，声誉值应当是通过稳定的几次优良交互建立起来的，不应当对声明狼藉的恶意节点更改标识符有利。

4　最小化系统在计算、设施、存储和消息复杂度上的消耗。

5　这个系统应该对恶意节点的合作攻击有鲁棒性。

3　声誉系统

成功声誉管理的一个重要案例是 eBay 的在线拍卖系统。在 eBay 的声誉系统中，购买者和销售者可以在交易后相互打分，每个交易参与者的总声誉是最近 6 个月来这些打分的和。这个系统依靠一个存储和管理这些评分的中心系统。

在一个分布是的环境中，节点也可以像 eBay 的系统那样在每项交易完成后相互打分。比如，每次节点 i 从节点 j 处下载一个文件，它可以把交易评为成功（tr(i,j)=1）或不成功(tr(i,j)=-1)。如当下载到了假的或篡改的文件，或者下载被中断了，节点 i 可以将这次下载评为不成功。就像 eBay 的模型一样，我们可以定义一个局部信任值 sij，为所有节点 i 对节点 j 每个下载评分的和。

sij=sigma(tr(i,j))

同样地，每个节点 i 可以存储自己与节点 j 满意交易的次数 sat(i,j)与不满意交易的次数 unsat(i,j)。从而，sij 可以定义为

sij=sat(i,j)-unsat(i,j)

之前对 P2P 声誉系统的工作都针对的是局部信任值这个概念。在分布式环境的声誉系统中，如何不通过中心存储和管理设施收集局部信任值。尽管引用的以前的每个系统都对这个问题提出了解决方案，但它们都有一两个缺陷。要么它们只能收集少量几个节点而不能反映全局的声誉值，要么它试图收集所有节点的评分，通过发送系统消息询问每个节点而造成网络拥塞。

我们提出了一个很自然而且能够收集所有用户的局部信任值的声誉系统。这个系统在消息复杂程度上有着最小的开销。我们的工作是基于信任传递思想的：节点 i 不仅信任传给它真文件的节点，而且按照那样的信任程度信任那些节点给出的信任值。因为提供真文件的那些节点往往会如实报告它们的局部信任值。

我们展示了信任传递将使得系统得全局信任值可以通过一个矩阵最左边的特征根求得。我们展示了如何通过很短的几行代码，实现分布式的特征根计算。消息的复杂程度可以被证明有上界，而且经验上很低。更重要的是，我们展示出这个系统在减少不满意的下载方面十分有效，即便 70%的节点都结成一个恶意的攻击该声誉系统的团体。

4 eigentrust

这一部分，我们描述了 eigentrust 算法。在 eigentrust 中，每个节点 i 的全局声誉是通过其他节点对节点 i 的局部信任值，对这些节点的全局声誉加权求和得到的。在 4.1 节中，我们展示了如何对局部信任值进行归一化。通过这种方法可以得到一个美观的概率解释，以及一个有效的收集局部信任值的方法。在 4.2 节中，我们讨论了如何通过一种合理的方法收集正规化后的信任值。在 4.3 节中，我们讨论了局部和全局信任值的概率解释。从 4.4 到 4.6 节，我们给出了一个用于计算全局信任值的算法。

4.1 对局部信任值进行归一化

为了收集局部信任值，有必要对它们先进行归一化。否则的话，恶意节点可能会给其他恶意节点一个非常大的局部信任值，而给好的节点一个非常低的信任值，很容易颠覆这个系统。我们这样定义归一化的局部信任值 cij

cij=max(sij,0)/sigma(max(sij,0),j)

这样就保证所有的值都在 0 到 1 之间。（注意如果 sigma(max(sij,0),j)是 0，那么 cij 无定义。我们会在 4.4 节讨论这个问题）这样归一化也有一些缺陷。一，归一化的信任值无法区分未被交互的节点和坏节点。二，cij 只是一个相对的数值，没有一个绝对的解释。这是说，如果 cij=cik，我们知道 j 和 k 在 i 看来声誉是一样的，但我们不知道它们的声誉是都很好还是都很差。但是，我们仍可以通过这种方法获得优良而稳定的结果，即便这种归一化有如上缺点。这样归一化之后，我们在迭代计算全局信任值的时候就不需要再归一化了(这时再归一化的消耗会非常大，以致不可实现)，并且可以得到一个美观的概率模型，这就是我们这样归一化局部信任值的原因。

4.2 收集局部信任值

要完成分布式收集归一化后的局部信任值的工作，一个很自然的方法就是节点 i 向它的“熟人”询问它们对其他节点的看法。可以通过用节点 i 对它们的信任值对它们的看法进行加权。

tik=sigma(cijcjk,j)

这里 tik 代表节点 i 对节点 k 通过向朋友寻味的方式得到的信任度。

我们可以把这个写成矩阵形式。定义 C 为矩阵[cij]，ti(vector)为包含 tik 的向量，那么 ti(vector)=C^Tci(vector)。注意其中 sigma(tij,j)=1。

这是每个节点获取整个网络信息很有效的方法。通过这种方法，每个节点可以获得超出自己经验的信息。然而，这时节点 i 储存的信任值仅能反应节点 i 和它熟人的经验。为了获得更宽广的视野，节点 i 也许会问它朋友的朋友(t=(C^T)^2ci(vector))。如果它继续这种方式(t=(C^T)^nci(vector))，当 n 足够大时，它就可以获得网络的完整信息（假设 C 是不可约正常返的。实践证明这个假设成立，我们在 4.5 节会提到）。

幸运地，如果 n 足够大，信任向量 ti(vecotr)对所有节点 i 将收敛到同一个向量，即 C 的主特征向量。换句话讲,t(vector)是这个模型的全局信任向量。它的每个元素 tj，表示了整个系统对节点 j 的信任程度。

4.3 概率意义上的解释

类似于[12]中的随机冲浪者模型，这个方法有一个直观的概率意义上的解释。如果一个人要搜索可信的节点，它可以在网上用这个算法攀爬搜索:从节点 i，以 cij 的概率爬行到节点 j。经过一段时间这样的爬行之后，这个人将更有可能处于可信的节点上。这个马尔可夫链对归一化的局部信任矩阵 C 的平稳分布就是我们提到的全局信任向量 t(vector)。

4.4 基本的 eigentrust

这一节，我们描述了基本的 eigentrust 算法。暂时先不考虑 P2P 本来的分布式特性，我们先假定有一个中心服务器知道所有的 cij，而且负责计算。在 4.6 节中，我们会讨论如何在分布式的环境中计算全局信任值。

我们只需要计算~t = (CT)n~。对于 n 非常大的情况。其中 e 为所有 m 个节点的均匀分布 ei=1/m（在 4.2 节中，我们说我们要计算~t = (CT)n~ci，其中 ci 为节点 i 归一化的局部信任向量。然而它们都收敛到 C 的主特征值，因此可以用 e 代替）

算法 1 展示了最基本的算法

4.5 实际的话题

这个简单的算法有三个实际的问题没有解决。先验信任，不活跃节点和恶意联盟。

先验信任度：

往往网络中有些节点已知是可信赖的。举个例子，比如 P2P 网络最早的几个节点往往不会试图毁坏它们搭建起来的网络。加入这些特性会很有用，会使得这个系统更加无缝和自然。我们可以通过对已经信任的节点定义一种分布来做到这一点。举个例子，如果有一组节点 P 已知是可信赖的，我们可以定义如果 i P，pi=1/|P|，否则 pi=0。我们可以通过三种方法使用 pi。首先，在恶意节点存在时，会比更快收敛。所以我们可以使用 p 作为初始向量。我们在下面会讨论其余两种使用 p 的方法。

不活跃的节点：

如果节点 i 并没有从其他人那里下载过，或者他对所有人的局部信任度都是 0，在等式 1 中的 cij 就无法定义。在这种情况下，我们不妨设置 cij=pi，所以我们可以重新定义 cij

这是说如果 i 不知道或不信任任何人时，它会选择信任它预先信任的节点。

恶意联盟：

在 P2P 网络中，恶意联盟是有可能产生的。恶意联盟是一组相互认识的恶意节点。它们相互之间会有很高的局部信任值，而对其他成员的局部信任值都很低，从而试图颠覆信任系统，获得很高的全局信任值。我们可以设置

来解决这个问题。其中 a 是一个小于 1 的常数。这等价于将所有节点的意见向量设置成，让每个节点对信任集 P 中的节点都至少有一个最低程度的信任。而 P 一定不是恶意联盟的成员。从概率角度讲，这等价于一个人在网络中攀爬时，不容易陷入恶意联盟。因为每一步时，他都有一个概率跳到 P 中。注意，这也使得 C 变得不可约且正常返，保证了运算的收敛。

修改后的算法由算法 2 给出。

有一点需要强调，那就是预信任的节点对这个算法十分重要。因为它们保证了收敛性，并且可以瓦解恶意联盟。因此，选择与信任的节点十分重要，尤其应注意不能选择恶意联盟中的节点，否则算法的效果将大打折扣。为了避免这一点，应该由系统选择很少的几个预信任的节点，比如网络的设计者。如何选择预信任的节点是一个很有意思的研究领域。但这超出了这篇文章的讨论范围。

4.6 分布式 eigentrust 算法

这里我们给出了一个网络中所有节点合作计算并存储全局信任向量的算法。而且这种算法的信息消耗是最小的。

在分布式的环境中，首先得一个挑战是如何存储矩阵 C 和向量 t。在之前的几节中，我们提到每个节点都可以存储它的局部信任向量 ci。这里，我们也提出每个节点可以存储它自己的全局信任向量 ti（为了演示的考虑，我们暂时忽略了安全上的考虑，而允许节点维护自己的全局信任向量。第 5 节会提到安全方面的问题）。实际上，每个节点都可以通过下式计算自己的全局信任向量。

观察可以发现这是抽出一行的版本。这里要注意，由于节点 i 只与其他有限的几个节点进行过交互，等式 6 中很多项都是 0。这导致等式 6 变为一个简单的分布式算法，如算法 3 所示。有两件事情很有意思。一、 只有预先信任的节点需要知道它们自己的 pi。这意味着预信任的节点仍可以是匿名的。没人需要知道它们是预信任节点（有人会想，通过具有高全局信任值就可以判断出来哪些是预信任节点。然而仿真指出，预信任节点的信任值只是略超平均水平，极少能获得最高的全局信任值）。

二、在大多数 P2P 网络中，每个节点余其它节点都只有有限次交互。这样一来带来两点好处。1，计算得计算量不大，因为大部分 cji 是 0。2，需要传输的信息量很小。因为 Ai 和 Bi 都很小。在一个网络中有大量极繁忙节点的时候，我们可以强制限制每个节点报告的 cij 来限制计算量。

4.7 算法复杂度

这个算法的复杂度可以通过两种方式定界。第一，这个算法的收敛速度很快。对于一个 1000 节点的网络，只需 100 轮查询即可收敛（在第 7.1 节我们提到了仿真的方法）。图 1 描绘了差。很明显，这个算法只通过 10 次迭代就收敛了。因此，全局信任值经过很短的几轮迭代之后就变化很小了。在算法的分布式版本中，这意味着只需要更新 10 次局部信任值。eigentrust 收敛速度快的原因在中有提到。

第二，我们可以限制一个节点报告的局部信任值的个数。在 eigentrust 的修改版本中，每个节点只报告它局部信任值的子集。初步的仿真表示这样的方法表现比这里提到的报告局部信任值的方法效果都要好。

5 安全的 eigentrust

在前一节我们提出的算法中，每个节点计算并报告自己的信任值 ti。恶意节点很容易报告假的信任值，从而颠覆这个系统。

我们通过实行两个简单的想法来应对这个问题。一，一个节点当前的信任值不应该由它自己来计算和储存，否则它可以很容易操纵信任值。因此，我们需要网络中一个其它的节点来计算这个节点的信任值。二，恶意节点在计算其他人的信任值时可能会给出错误的结果。因此，一个节点的全局信任值应该由更多的节点来计算得到。

在分布式信任算法的安全版本中，有 M 个节点计算节点 i 的信任值。如果一个节点需要节点 i 的信任值，它可以查询所有管理信用的 M 个节点。通过一个多数投票可以对恶意节点操纵信用进行打击。

为了指定信用管理者，我们使用了一个分布式哈希表（DHT）如 CAN 和 Chord。DHT 使用一个对特定关键字的哈希函数将其转换成逻辑空间中的一个点。在任何时候，对应的空间都被动态分割，而每一个节点覆盖空间中的一个区域。每个节点负责存储对应区域中的（关键字，值）的配对。

在我们的方案中，一个节点的得分管理者通过对这个节点一个唯一的 ID，如 IP 地址和 TCP 端口，进行分布式哈希操作来定位。负责这个区域的节点将被指定为信任值的管理者。系统中所有知道这个唯一 ID 的节点都可以定为这个信任值管理者。我们可以修改最初的算法，使之可以被信任值管理者执行。

作为一个例子，考虑图 2 所示的 CAN。节点 1 的唯一 ID，ID1，被哈希函数 h1,h2,h3 映射到节点 2、3、6 负责的区域中。因此，这些节点会变成节点 1 的信任值管理者。

为了适应 P2P 系统内在的动态性，我们需要设计一个稳定的 DHT 算法。举个例子，当一个信任值管理者离开这个系统时，它会把它的状态传送给在 DHT

空间中与它相邻的节点。DHT 算法也可以引入冗余来防止信任值管理者失效时带来的数据丢失。

5.1 算法描述

这里我们描述了计算全局信任向量一个安全的算法。我们将用到这些定义：每个节点都有 M 个信任值管理者，通过对一个节点唯一的 ID,posi,进行一系列单向哈希函数 h0,h1,...,hm-1 来确定。由于每个节点也是信任值管理者，它会有一组子节点 Di，即由它管理的信任值的节点。作为一个信任值管理者，节点 i 需要为它的子节点 d 维护局部信任向量 cid。而且，节点 i 需要知道 Aid，即从它子节点下载文件的节点集。它需要从那些节点收集对 d 的信任评价。最后，节点 i 还需要知道点集 Bid，它的子节点从 Bid 里下载过文件。为了全局信任值的计算，子节点 d 需要向信任值管理者提交它对其他节点的局部信任值和 Bid。

安全算法提高安全性和可靠性的方面如下：

匿名。由于不知道 ID，信任值管理者不知道它在为哪个节点计算信任值。因此恶意节点不能够提高其他恶意节点的信任值。

随机。节点在加入系统时不能够选择它的信任值管理者（这是一个优秀设计的 DHT 算法具备的特点）。因此，一个节点无法把自己精确地放在自己管理的区域中，即无法负责计算自己的信任值。

冗余。同一个信任值由数个信任值管理者合作计算，为了对同一个节点指定多个信任值管理者，我们使用了多维哈希函数。系统中的节点仍然负责对应空间中的一个区域，但现在有了多个空间，每个空间由哈希函数的一维产生。一个节点的唯一 ID 因此被映射成了多个点，每个维度上一个。

5.2 讨论

这里有两个要点需要着重提起。

一、P2P 网络中安全的分数管理是十分重要的问题，对于不论是信任系统、刺激系统还是 P2P 微型支付架构都十分重要。参考文献 20 中给出了更多的关于 P2P 网络中安全分数管理的讨论。本工作的主要目的并不是给出一个安全的 P2P 安全分数管理架构，而是在于 EigenTrust 评价算法。我们讨论安全的分数管理架构是因为有一些这样的架构可以支持 EigenTrust 算法。EigenTrust 其实可以与更多的安全分时管理架构协同工作。

二、这里和参考文献[20]钟给出的安全协议描述了如何通过大量的实体改变由一个人或一个组织操纵分数的局面。这些协议从传统的意义上来说并不安全，但我们证明了一个节点成功地谎报分数的概率是极低的。这个问题在参考文献[20]中有更深入的说明。

综合论文训练记录表

学生姓名		学号		班级	
论文题目					
主要内容以及进度安排	指导教师签字： 考核组组长签字： 年　　月　　日				
中期考核意见	考核组组长签字： 年　　月　　日				

指导教师评语	指导教师签字：_____ 年　　月　　日
评阅教师评语	评阅教师签字：_____ 年　　月　　日
答辩小组评语	答辩小组组长签字：_____ 年　　月　　日

总成绩：_____

教学负责人签字：_____

年　　月　　日