清 华 大 学

综 合 论 文 训 练

题目：LIGO中的数据监测工具箱在网格中间件上的实现

系　　别：自动化系

专　　业：自动化专业

姓　　名：李俊伟

指导教师：曹军威　研究员

2008 年 6 月 10 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：_____导师签名：_____日　期：_____

中文摘要

自从 20 世纪 90 年代网格的概念出现以来，网格就作为一种能带来海量处理，存储等能力的新型网络而被人们誉为继 Internet 和 Web 之后的第三次信息技术浪潮。几乎与此同时，为了探测到爱因斯坦预言的引力波，美国自然科学基金启动了激光干涉引力波天文台项目（LIGO）。该项目对数据传输、处理、存储等有着很高的要求，而这恰恰就是网格所擅长的。

本文主要实现了 LIGO 中数据监测工具箱在由不同网格中间件搭建的网格上的运行，并在网格使能层上对其运行进行了一些改进，最后还从运行数据监测工具箱的视角对由不同网格中间件搭建的网格的性能进行了比较。

关键词：网格 网格使能层 网格中间件 依赖 并行

ABSTRACT

In the 1990s, the concept of grid comes out. From that time, grid has been hailed as the third tide of information technology after the Internet and Web because grid is a new type of net that can bring us with capability of processing mass data, storing mass file and so on. At the same time almost, to detect the gravitational wave that Albert Einstein predicted, National Science Foundation started a project named Laser Interferometer Gravitational-wave Observatory (LIGO). This project has high demands in data's transferring, processing and storing, and grid can meet these demands well.

This paper mainly realizes the running of LIGO's Data Monitoring Tool(DMT) in grid built by two different grid middleware, and makes some improvement to the running in grid enabling layer, at last, compares the performance between the grids built by two different grid middleware in the view of executing the Data Monitoring Tool.

Keywords：Grid enabling layer Grid middleware Dependence Parallel

目　　录

第1章 序言

1.1 LIGO简介

1916 年，艾伯特·爱因斯坦预言了引力波的存在，并将引力波作为广义相对论的重要组成部分：电荷被加速时会发出电磁辐射，同样有质量的物体被加速时就会发出引力辐射。

1974 年，约瑟夫·泰勒和拉塞尔·赫尔斯间接证明了引力波的存在。为了找到引力波存在的直接证据，1990 年，由美国国家自然科学基金（简称NSF）出资，美国麻省理工MIT和加州理工Caltech共同启动了激光干涉引力波天文台项目（Laser Interferometer Gravitational-Wave Observatory，简称LIGO）[1]，并成立了LIGO科学合作联盟（LIGO Scientific Collaboration），来自全球 40 多个研究机构的500 多名研究人员参与并共享着分布在美国和欧洲的3000 个高性能CPU及超过1000TB的存储资源[2]。

1.1.1 LIGO工作情况

LIGO用一个名为激光干涉仪的设备来探测时空中的重力波。LIGO共建造了三个这样的激光干涉仪，其中两个在美国，另外一个在德国。LIGO需要至少两个相距遥远的探测器，对它们进行同步操作才能去除错误的信号并确认重力波是否经过了地球[3]。

LIGO的这三个天文台，每秒总共能从 2000 多个感应器频道中产生 7 至 9M 的数据，而LIGO需要低延迟地对数据进行分析，因为只有这样，才能给天文学家提供足够的预警时间去观测重力波产生的方位所发生的天文事件。而为了能够实时地同时处理这些分散在三个相距遥远的天文台的数据从而找到引力波存在的证据，LIGO运用网格技术搭建了自己的一套网格系统，名为LIGO Data Grid[4]。

1.1.2 LIGO中的数据监测工具箱

顾名思义，数据监测工具箱（Data Monitoring Tool，简称 DMT）是一个为更好地监测 LIGO 实时产生的数据而特别开发的软件包。DMT 包括若干个库，定义了相关的工具和环境，能够支持对 LIGO 的天文台进行连续的数据监测。

1.2 网格技术介绍

对于不同的人来说，网格这个概念有着它不同的意义。用一个形象化的视角来看待网格，可以认为电力网格与它相似。一方面，由纵横交错的电缆连接起来的电网和由铺设在全球各地的光纤、网线等连接起来的互联网颇有几分相似之处：电器对应着计算机，插座对应着网卡，电线对应着网线；另一方面，就犹如人们可以通过墙上的插座获取电力，而不用考虑电力在何处通过何种方式生产出来一样，网格的目标也是如此。它的最终目的是让用户能方便快捷地存取和使用分布在因特网上海量的资源（计算能力，存储能力，数据，应用程序以及其他资源）而不用了解这些资源存放在何处，资源的存放形式和环境如何。对于终端用户而言，这些计算能力，应用程序等等就像是一个巨大的虚拟计算机所提供的[5]。

1.2.1 网格技术的起源及发展

网格技术由 80 年代开始的远程计算和分布式计算机的应用研究发展而来。

1992 年，美国国家超级计算应用中心的Charlie Catlett和Larry Smarr提出了元计算（Metacomputing）的概念。他们为了在网络上构筑虚拟计算机环境、执行大规模平行计算处理而开展研究。1995 年，美国伊利诺伊大学的Tom　De Fanti和阿贡国家实验室的Rick Stevens等人根据I-Way计划，进行了最早的大规模元计算实验。这一计划通过高速广域网连接了全美地区 17 家计算中心，实施了虚拟现实实验等很多应用验证。以I-Way计划为开端，1996 年美国阿贡国家实验室的Ian Foster和南加利福尼亚大学的Carl Kesselman的研究组启动了Globus计划，开发用于高性能分布式计算的中间件。1998 年，表示“网格技术”概念的蓝图出现了[6]。

1.2.2 网格技术的应用及意义

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。并且计算将理论和实验连接起来，成为二者间的桥梁。通过计算，能够完成许多单纯靠理论或实验无法完成的科学研究。比如工程结构仿真，有限元分析，天气预测，股票预测等等，计算已经在各个研究领域取得了重要的地位。

既然计算具有重要的地位，而网格中很重要的也是最本源的一种应用就是整合提供强大的计算能力。因此，网格计算大有用武之地。

在过去几年里，网格处理能力（每秒可以处理的位数）和微处理器的速度（依赖于集成电路中晶体管的数量）之间出现了一个巨大的差距[7]，如图 1.1 所示。

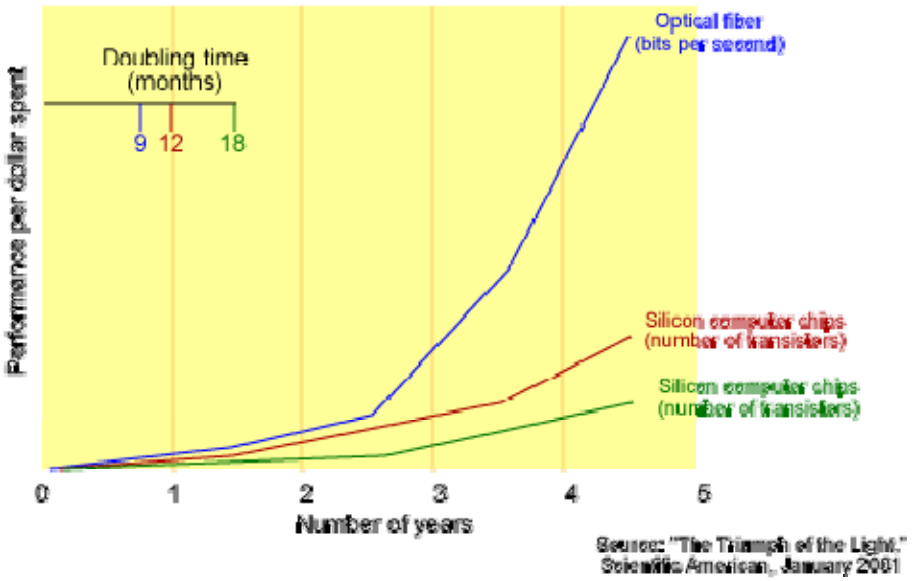

图1.1　网格处理能力与微处理器能力的比较

如图中所示，网络处理能力现在每 9 个月就要翻一番，而摩尔定律指出每个集成电路中晶体管的数量每 18 个月就会翻一倍[7]。这样就出现了这样一个问题：与网络能力的发展相比，处理器的发展速度要慢很多。比如从 1986 年到 2000 年，计算机的处理速度提高了 500 倍，网络的速度提高了约 340,000 倍。估计从 2001 年到 2010 年，计算机的处理速度将提高 60 倍，而网络的速度将提高 4000 倍[8]。目前的PC已经比 10 年前的超级计算机还要快，而大量的计算机在许多的情况下其计算能力是闲置的，因此可以共享利用其计算能力。

网格除了在前边提到的计算领域能施展本领外，在其他诸多领域也有很好的应用，比如网格能实现更为广泛深入的资源共享。最近几年 P2P（Peer To Peer）技术很流行，它能实现用户直接连接到其他用户的计算机，从而进行资源的共享和交换，资源可以是计算能力，文件，网络连接能力，打印机等等。而网格则比 P2P 更进一步，除了能进行各种资源的共享外，还提供了 P2P 所不具备的安全信任机制。

1.2.3 网格的特点[8]

网格主要有以下 4 个特点：

1. 分布与共享

网格目的就在于集成共享分散在各处的计算机资源，因此分布性是网格的最主要的特点。网格的分布性首先指的是网格的资源是分布的，且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就注定是分布式计算而不是集中式计算。而分布式计算里边则需要解决诸如资源与任务分配和调度，数据的传输等等问题。

分布在各处的网格资源是可以充分共享的，这也是网格的目的所在。共享的内涵非常广泛，不仅仅指可以提交任务到远程主机上运行，还可以指中间结果，数据库，专业模型库以及人才资源等方面的共享。

分布式网格硬件在物理上的特征，而共享则是在网格中间件支持下实现的逻辑上的特征，两者对于网络来说都是十分重要的。

2. 自相似性

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。

3. 动态性与多样性

网格是动态搭建的，任何主机或资源可以随时加入或退出网格。

此外网格上的资源是异构的，具有多样性。同一个网格中，可以存在不同硬件体系结构的计算机系统和不同类别的资源。

动态性和多样性就要求了网格系统能够解决这些不同结构、不同类别的资源之间的通信和互操作问题，从而能够使得新的资源能够加入网格并与原来的资源融合在一块。

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拥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。

但网格创建的本意是用来共享资源，因此在获得资源拥有者允许的情况下，网格的资源需要受到网格的统一管理。这就是网格管理的多重性的表现。

1.2.4 网格的结构[9]

目前对网格的体系结构有多种看法，其中五层沙漏结构最具代表性。参照该结构使用网格中间件来提供一组计算服务，构成沙漏的瓶颈部分，即最中间的那层。从该层向下可屏蔽资源的异构性、动态性和分布性；向上可以提供基于 Web 服务的、透明的、可视化的网格计算环境。从功能上看，该体系结构可以划分为以下四层：应用接口层、中间件层、抽象接口层和资源层，如图 1.2 所示。

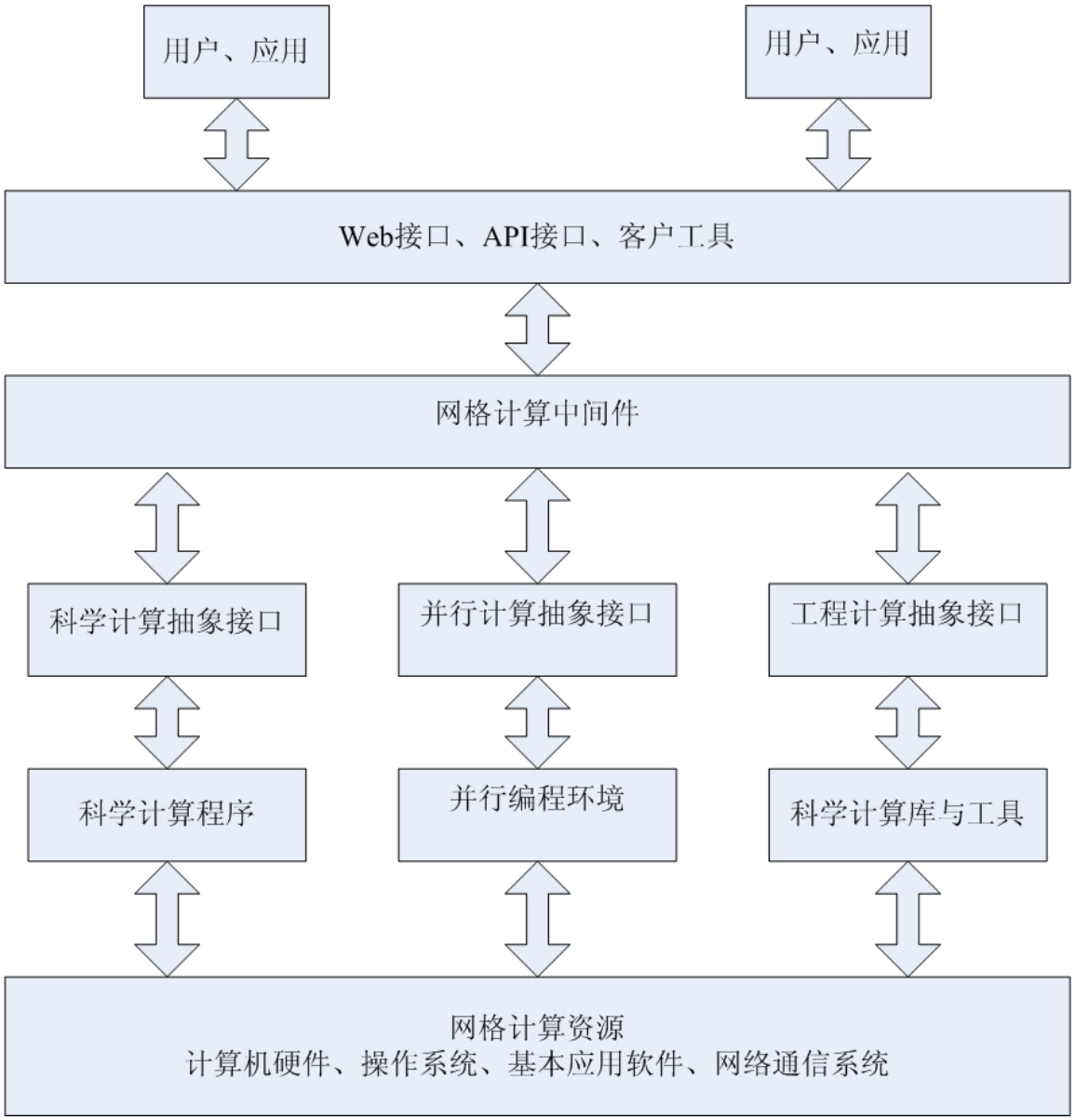

图1.2　网格的五层沙漏模型

其中网格计算资源是网格提供计算服务的基础，最终所有应用都要调用到网格的部分或是全部计算资源上来运行。抽象接口层则综合了网格计算模式下的计算服务功能，包括科学计算服务、并行计算服务、工程计算服务、应用接口层为用户提供了多样化的使用接口。网格中间件是网格计算的核心所在，为用户提供具有统一编程接口的虚拟平台，支持复杂应用问题的求解和资源的共享，它一般由抽象接口层和中间件层两部分组成，将分布于网格中的各个异构平台上的各种资源整合在一起，协同完成一个任务。

1.2.5 网格中间件

网格中间件（Grid Middleware）是一种在网格中应用的中间件（中间件，顾名思义，网格中间件位于操作系统与用户的应用软件中间。总的作用是为处于自己上层的应用软件提供运行与开发的环境，帮助用户灵活、高效地开发和集成复杂的应用软件）。

网格中间件是整个网格体系的核心部分，它能完成如下几个功能[9]：

1. 资源动态监测
2. 屏蔽节点异构
3. 优化资源选择
4. 协同计算

网格中间件的种类很多，其中 Globus 和 Condor 是两种具有代表性且使用较为广泛的网格中间件。

1.3 问题提出及解决方案

前边提到了 LIGO 的数据监测工具箱 DMT，由于 LIGO 实时产生的数据是放在网格上进行处理，因此为了监测 LIGO 的数据，DMT 也需要放在网格上运行。但是 DMT 在网格上的运行存在问题和不足，下边将逐一介绍，并简要说明解决方法。

1.3.1 数据监测工具箱在网格运行的限制

DMT 目前只有 Unix/Linux 下的版本，它有以下一些特征：

1. DMT 采用 C/C++编译而成。
2. DMT 的编译需要包含一些特有的头文件并链接到一些特定的专业动态链接库。
3. DMT 的运行同样需要链接一些特有的动态链接库。
4. 为了监测数据，DMT 必然要读取数据。
5. DMT 也需要输出监测结果。
6. 由于 LIGO 数据是动态连续产生，因此 DMT 也要实时读取，输出结果。

由上述的这些特征，可知 DMT 在网格上运行大致有如下几条限制：

1. 前边提到 DMT 的编译和运行均需要链接到一些特有的动态链接库，而 Linux 下的动态链接库与 Windows 下的动态链接库（Dynamic Link Library，简称 dll）

有所不同，Windows 下的动态链接库能够被编译进二进制程序中，而 Linux 下的动态链接库则不能被编译进程序中，除非动态链接库的源程序公开，但这个往往很不现实。因此 DMT 若想在由各个具有不同 Unix/Linux 操作系统的计算节点组成的网格上运行的话，就面临着动态链接库的版本随操作系统的不同而有所差异的问题，这也就是通常所说的程序的可移植性问题。但很

2. DMT 若想在某台机器上运行，首先数据得在该机器上，因此必须考虑到数据实时传送以供 DMT 读取的问题。
3. DMT 的结果也得考虑如何实时显示。

1.3.2 网格中间件的不足

目前的网格中间件仍然不够完善，如前边图 1.2 所示，从网格中间件所抽象出来的层次不够高：有太多的技术细节需要被关注。比如，若提交到网格上的程序依赖于一些特殊的动态链接库，只能预先人工在目标机器上安装好，程序才能运行，而目前的网格中间件还不能做到解决库的依赖性问题。再比如为了提高程序在网格上的计算速度，就需要程序同时在多台计算机上运行，而这涉及到数据分发，并行计算的问题，而这些功能网格中间件目前也不能直接支持。

网格中间件的这些不完善的地方，带来了这样的问题：

1. 由于关注过多的技术细节，降低了工作效率。
2. LIGO 中提交 DMT 到网格运行的物理学家们对于网格中间件很不了解，因此不仅工作效率降低，还有可能出现任务执行失败的情况。

1.3.3 解决方案

屏蔽网格中间件的技术细节，在网格中间件和最终用户之间增加一层，称为网格使能层（Grid Enabling Layer）。它的作用相当于中间人的角色，接受终端用户的命令，并解释这些命令，交给网格中间件执行。这样就给 LIGO 用户使用网格提供了一个友好的接口。

1.4 本文工作

为了解决前边提出的问题，本文的主要工作为：

第一步：熟悉一些常用的网格中间件的基本原理及其使用。

第二步：实现 DMT 样例程序在网格中间件上的运行。

第三步：在网格使能层上进行一些初步的开发，提供一个易用的接口给用户。

第四步：对不同网格中间件的性能进行分析，比较其表现，为 DMT 挑选网格中间件。

1.5 文章结构

本文的剩余章节安排如下：

第 2 章将对两种典型的网格中间件进行介绍，同时还介绍 DMT 样例程序 Rmon，并说明测试开发环境。

第 3 章将详细说明 Rmon 在网格中间件上的运行，并介绍在网格使能层上所做的一些开发，并对 Globus 和 Condor 的性能进行了比较。

第 4 章给出结论，分析所做开发的缺陷。简要总结本文工作，并提出将来可供改进的方向。

第2章 网格中间件及DMT样例程序

2.1 Globus简介

1996 年，Globus作为一个开源的软件项目而启动了。目前Globus已经发展到Globus Toolkit 4.0 版本，包括IBM和Microsoft在内的一些公司都公开宣布支持Globus，且Globus已经被应用到数十个网格计算项目，比如欧洲数据网格（European Data Grid），美国的网格物理网络（Grid Physics Network）和地球系统网格（Earth System Grid）[10]。

Globus 是一种能让人们在不牺牲本地主机控制权的情况下安全地分享计算能力，数据库和其他工具的一种网格中间件。Globus 提供安全机制，资源监测，发现和管理以及文件管理的功能。这些功能被打包成了一系列的模块，用户可以选择所需的模块搭建网格，进行开发。

2.1.1 Globus的组织结构[11]

Globus 在底层安全机制之上有三个主要组成部分，如图 2.1 所示。

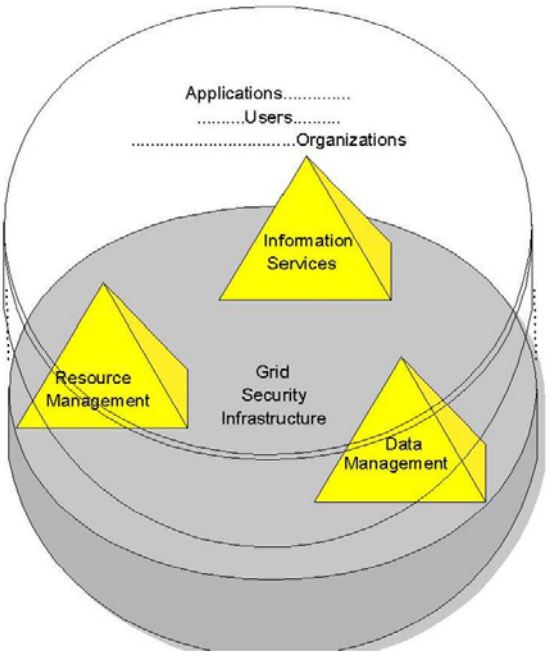

图2.1　Globus的组织结构

下面分别介绍这三个主要组成部分：

1. 资源管理

提供资源分派，任务提交（远程运行可执行文件且返回结果），管理任务状态和进展。

Globus 并没有提供任务调度的功能，因此不能自动找到合适的资源并发送任务到合适的机器上。因此 Globus 需要第三方的调度能力。

2. 数据管理

提供在机器间传输文件并管理这些传输的功能。

3. 通信

该组成部分提供搜集和查询网格上各种信息的功能，Globus 的通信部分基于 LDAP 协议（Lightweight Directory Access Protocol）。

所有的这些都建立在网格安全基础架构（Grid Security Infrastructure，简称 GSI）之上。GSI 提供包括单一/交互认证，加密通信，授权等安全措施。

2.1.2 Globus的组件[11]

对于前边提到的每个组成部分，Globus 都提供了一个相应的组件来实现，如图 2.2 所示。

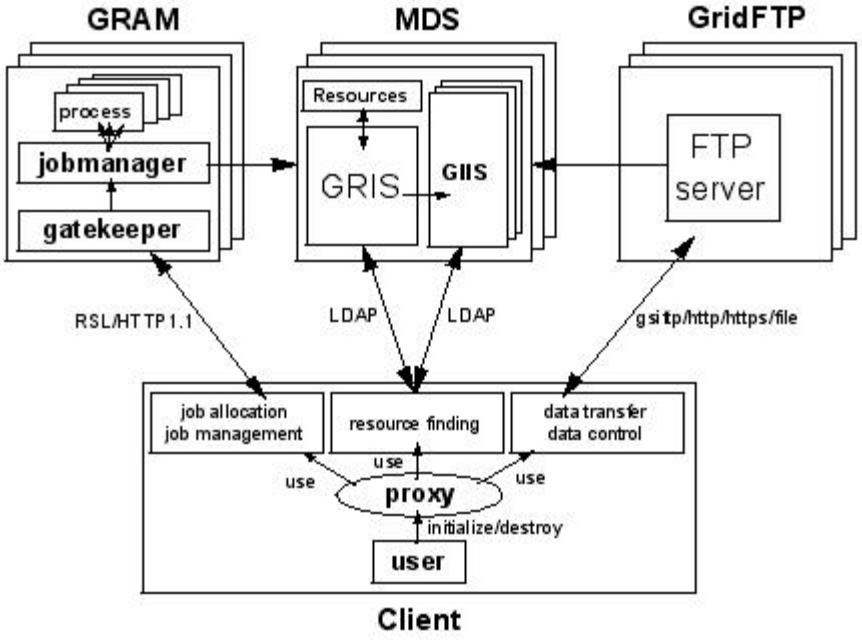

图2.2 Globus的组件

其中：

1. GRAM/GASS

资源管理部分最主要的组件，Grid Resource Allocation Manager（GRAM）和 Global Access to Secondary Storage（GASS）。

2. MDS(GRIS/GIIS)

基于 LDAP 协议（Lightweight Directory Access Protocol），Grid Resource Information Service（GRIS）和 Grid Index Information Service（GIIS）组件能搜集和分发网格上的信息。这两个组件统称为 Monitoring and Discovery Service（MDS）。被搜集的信息可以是关于机器的静态信息，也可以是显示目前 CPU 或者硬盘活动情况的动态信息。

3. GridFTP

GridFTP（Grid File Transfer Protocol）是安全、高性能地传输数据中的关键组件。

4. GSI

以上提到的三个组件都建立在底层的 Grid Security Infrastructure（GSI）之上。GSI 基于 SSL（Secure Socket Layer）协议，公钥加密（Public Key Encryption）和 X.509 认证标准。

2.1.3 安全机制GSI

若想用 Globus 搭建网格并在上边提交任务，仅仅安装完 Globus 是不够的，还需要 Globus 认证颁发证书。前边已经提到 Globus 的所有组件都建立在安全机制 GSI 之上，而 GSI 则采用了公钥加密技术。众所周知，公钥加密是不对称加密，使用一个对其他用户保密的私钥和一个可以对其他人公开的公钥。用公钥加密的数据只能用对应的私钥解密，而用私钥签名的数据只能用对应的公钥验证。

下边简要介绍一下GSI的具体功能[12]：

1. 认证

GSI 包含能够提供单一登录环境的底层结构和工具。通过执行 grid-proxy-init 命令，在用户私钥的基础上，一个临时的代理被建立起来，该代理有一定的时间限制。此代理能够提供认证并且能够用来生成可信任的远程连接信息并能允许服务端在用户的认证上做决定。

值得注意的是，在用户通过 Globus 组件提交任务或者传输数据之前必须生成一个代理。

2. 授权

认证只是安全问题中的一部分。这一部分是授权。授权指的是一旦用户被成功认证，他们应该被授予哪些权限。

在 GSI 中授权由一个映射控制，它将被认证的用户与请求被接收的本地机器上的一个用户对应起来。

由操作请求（例如请求运行一个任务）发出的代理信息包含一个独特的名字，这个名字能够代表被认证的用户。在接收请求的机器上有一个文件能够映射该名字到本地的某个用户。

通过以上的这种机制，网格上的用户要么能在网格中的每个系统上拥有一个用户 ID，要么能被指派到一个虚拟组当中。比如，一个特定区域内所有被认证的用户也许都会被映射到一个通用的用户 ID。

3. 安全通信

默认情况下，底层的通信是基于数字证书的相互认证和 SSL/TLS。

为了支持网格内的安全通信，OpenSSL 作为 Globus 的一部分而被安装。它被用来创建一个加密的信道通过在网格客户端和服务端之间使用 SSL/TSL。

数字证书则在网格客户端和服务端之间提供了相互认证。OpenSSL 提供的 SSL/TLS 功能将加密网格系统中所有数据的传输。这两个功能共同提供了基本的安全认证服务和保密性。

2.1.4 Globus安全认证机制

基于 GSI，Globus 开发了自己的安全认证机制。首先它需要在网格中的特定计算机上设置一个 CA（Certification Authority）中心，如果有新机器想加入该网格，则需在该机器上以 root 执行 grid-cert-request 命令，生成名为 hostcert_request.pem 的文件，然后将 hostcert_request.pem 发给 CA 中心的管理员，由 CA 中心对 hostcert_request.pem 进行签发，生成名为 hostsigned.pem，将其发送给新机器，由新机器的 root 账户独自保管。然后由新机器的一个普通用户再次执行 grid-cert-request，生成 usercert_request.pem，将其发给 CA 中心对其进行签发，生成 usercert.pem，将其发回新机器，由该普通用户单独保管，至此还没有结束，还需要配置一个名为 mapfile 的映射表。

在新机器上以 root 执行 grid-cert-request 命令时，会生成一个代表该机器的唯一身份标识字符串。如果该机器（令其为机器 A）想提交任务到远程机器 B，以机器 B 上的普通用户 C（Globus 不允许用 root 执行远程机器上的任务）来执行该

任务，则在 B 的 mapfile 映射表中，就必须添加 A 的身份标识字符串与普通用户 C 的用户名，并将其一一对应。由于只有 root 才有权限更改 mapfile 映射表，因此机器 B 可以随时控制机器 A 在机器 B 上的活动，而不用担心机器 A 控制 B，因为 A 永远获取不了 B 的 root 权限。当机器 B 不想让机器 A 提交任务到 B 时，B 只需要在 mapfile 映射表中删除 A 的身份标识字符串与普通用户 C 的用户名之间的对应就可以了。因此 mapfile 映射表就保证了每台机器的自治性，体现了前边第一章序言中提到的网格的特点之一：自治性与管理的多重性。

安全认证除了提高 Globus 的安全性外，还能带来一些意想不到的好处，比如由于 Globus 采用了证书来进行安全认证，因此在对远程主机进行相关操作时就不用输入远程主机用户的密码，减少了密码被窃取的风险，并且去除了输入密码的步骤，如图 2.3 所示。

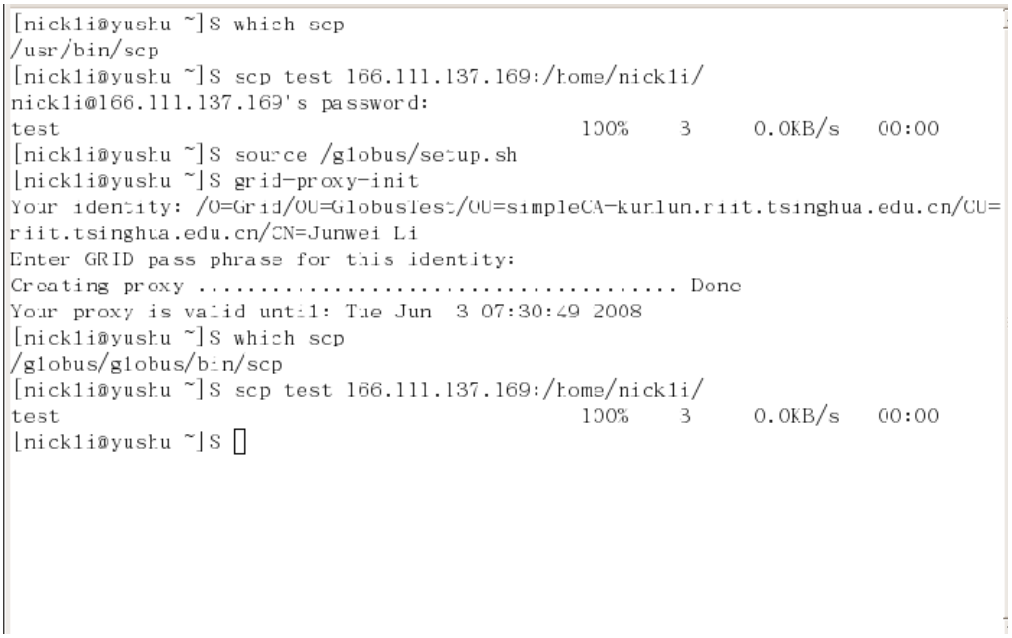

图2.3　Globus安全认证机制举例

如图 2.3 所示，最开始未启用 globus 时，用系统自带的/usr/bin scp 命令拷贝名为 test 的文件到 IP 为 166.111.137.169 的远程主机的/home/nickli 文件夹，需要输入 166.111.137.169 上用户 nickli 的密码。但是在设置了 globus 的环境变量后，并运行 grid-proxy-init 生成临时代理后，此时 scp 命令已经不是系统自带，而是

globus 的 scp 命令，存放在/globus/globus/bin 目录下，用它拷贝 test 文件到远程主机 166.111.137.169 的/home/nickli 文件夹就不用输入密码。

2.1.5 GridFTP[13]

GridFTP 在网格主机之间提供了安全和可靠的数据传输。它扩展了 FTP 协议以提供额外的特性，比如：

1. 第三方的数据传输，允许第三方对另外两台远程机器进行控制，从而实现两台远程机器之间的数据传输。
2. 使用多 TCP 流进行并行传输。
3. 数据的可靠传输，例如纠错。

Globus 提供了 GridFTP 服务端和 GridFTP 客户端，由 GridFTP 服务端的守护进程 in.ftpd 和 GridFTP 客户端的 globus-url-copy 命令等实现，支持 GridFTP 协议中的大部分特性。

GridFTP 支持两种文件传输方式：标准传输和第三方传输。

如图 2.4 所示，标准文件传输就是客户端发送文件到运行 FTP 服务器的远程机器的过程。

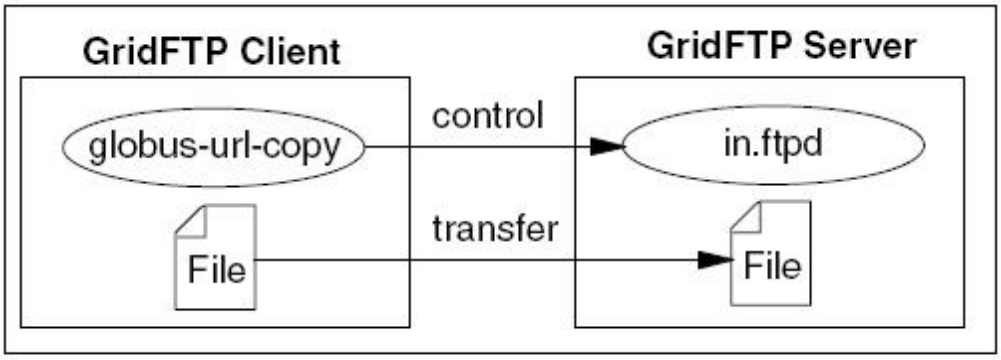

图2.4　GridFTP标准文件传输

第三方数据传输如图 2.5 所示，指的是第三方在两台服务器间传输数据。

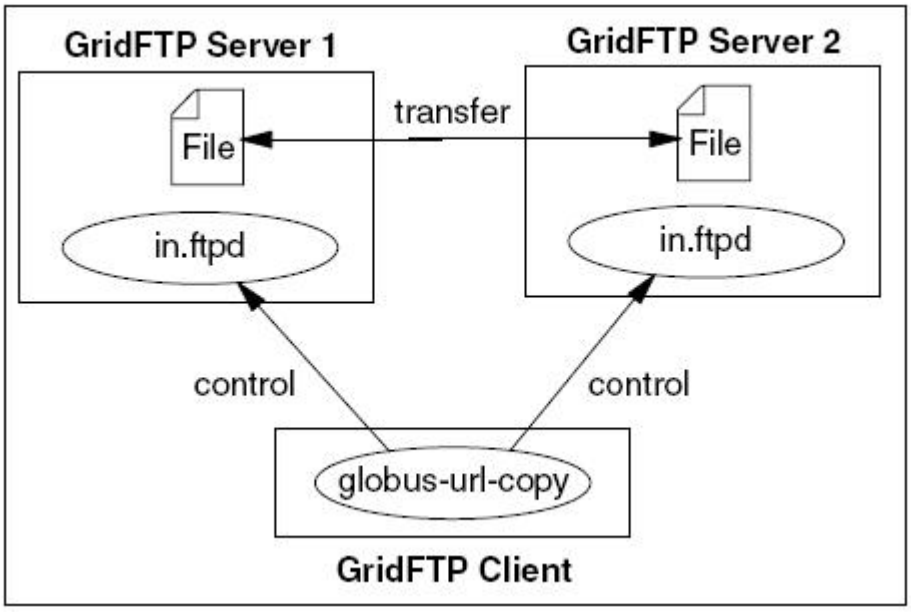

图2.5　GridFTP第三方文件传输

2.2 Condor简介

对于很多研究和工程项目而言，最终结果的好坏与可用计算资源的数量有很大的关系，往往有需要耗时几周甚至几个月的计算来解决问题的项目出现，因此这就需要一个能在较长时间内能够提供大量计算能力的计算环境。这种计算称为高吞吐量计算（High Throughput Computing，简称 HTC）。而 Condor 的目标是开发，实现，配置和评价那些支持基于分布式的计算资源上的高吞吐量计算的机制和策略。因此它的目标就决定了它的最大特点是它是一个专业的作业和资源管理调度系统。它能够建立一个大吞吐量的计算环境，并通过作业管理和资源调度，从而能在不影响网格中各计算节点自身正常使用的前提下有效利用所有可用的资源。

比如若某台机器 5 分钟内键盘无动作，则 Condor 认为该机器处于空闲状态，可将任务放到该机器上运行，如果机器主人回来，敲击键盘，则 Condor 可以选择将任务转移到其他机器上运行，从这个例子里，我们就可以很明显地看出 Condor 在保证用户不受到其他用户提交的任务影响的情况下，尽可能地利用了空闲的资源。

2.2.1 Condor组织结构

Condor 的组织结构如下：

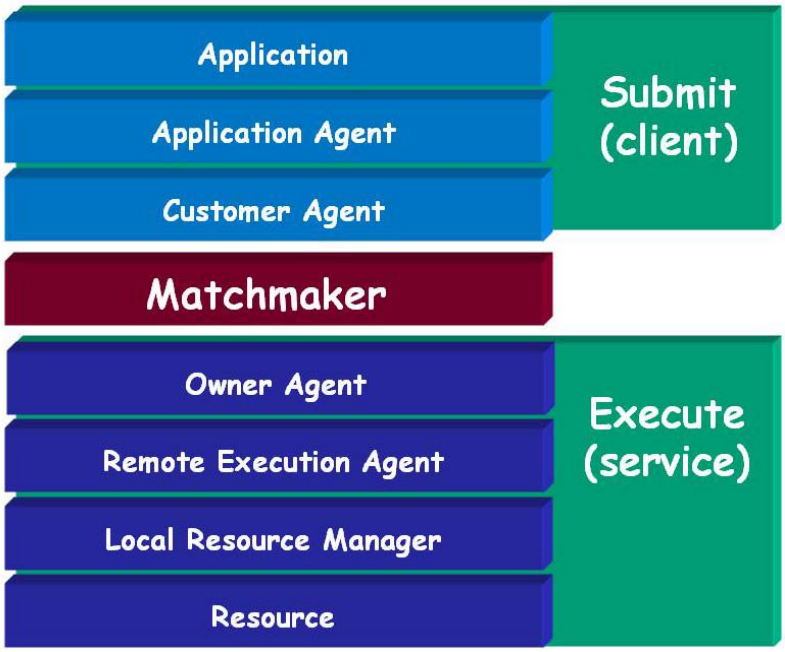

图2.6　Condor组织结构

2.2.2 Condor的特点

Condor 与前边提到的 globus 相比，有很多独特的地方：

1. ClassAds：Condor 中的 ClassAds 机制给资源请求和资源提供的配对提供了一个十分灵活易懂的机制。简单的举例来说，提交任务时要求该任务必须在内存不少于 1GB 的机器上运行，那么 Condor 就会自动去寻找符合该条件的机器，运行任务在符合条件的机器上。
2. 任务检查点和任务迁移：对于特定的任务，Condor 能创建一个检查点保存当前任务的完整状态并能随即从检查点文件中恢复该任务。周期性的检查点提供了某种形式上的容错，并能保护运行了一段时间的任务不至于前功尽弃。检查点的设置能够让任务从一台机器迁移到另一台机器。
3. 远程系统呼叫：当在远程机器上运行任务时，Condor 能通过远程系统呼叫维持任务运行环境不变。

4. 无需远程主机账户：通过链接到 Condor 的库，系统调用被捕获并由 Condor 完成，而不是由远程主机的操作系统完成。Condor 从远程主机上发送系统调用到提交任务的机器上。系统调用的函数执行，然后 Condor 将执行的结果返回到远程主机。

除了以上这些特点之外，Condor相比Globus并没有提供完善的安全认证机制，因此Condor更适合在较为安全的局域网网段内运行[14]。

2.2.3 Condor Pool

Globus 有一个 CA 中心，Condor 也有个中心，名字叫 Condor Pool。但它和 Globus 的 CA 中心不一样，并不负责安全认证，而是发挥一个用合适的资源去匹配等待中的请求的职能。Pool 是一个资源（机器）和资源请求（任务）的集合。Condor 组成的网格中的各个部分会周期性向中心传送更新信息，因此中心掌握了 Pool 中的所有机器和任务的状态信息。如图 2.7 所示。

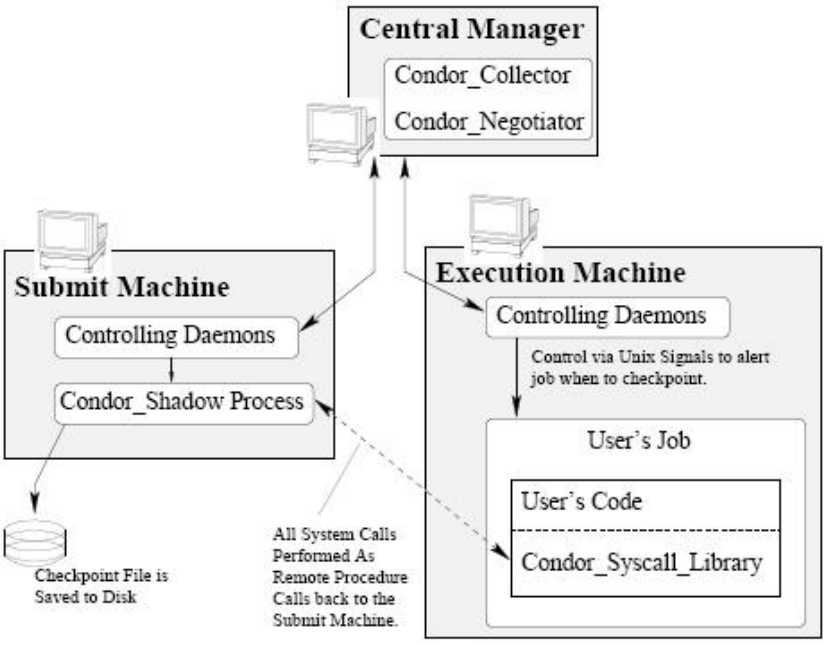

图2.7　Condor Pool的结构

具体地，为了后续开发的方便，我们设置了一个 Condor Pool，由 4 台计算机组成，共 8 个 CPU，用 condor_status 命令查询，如下图所示：

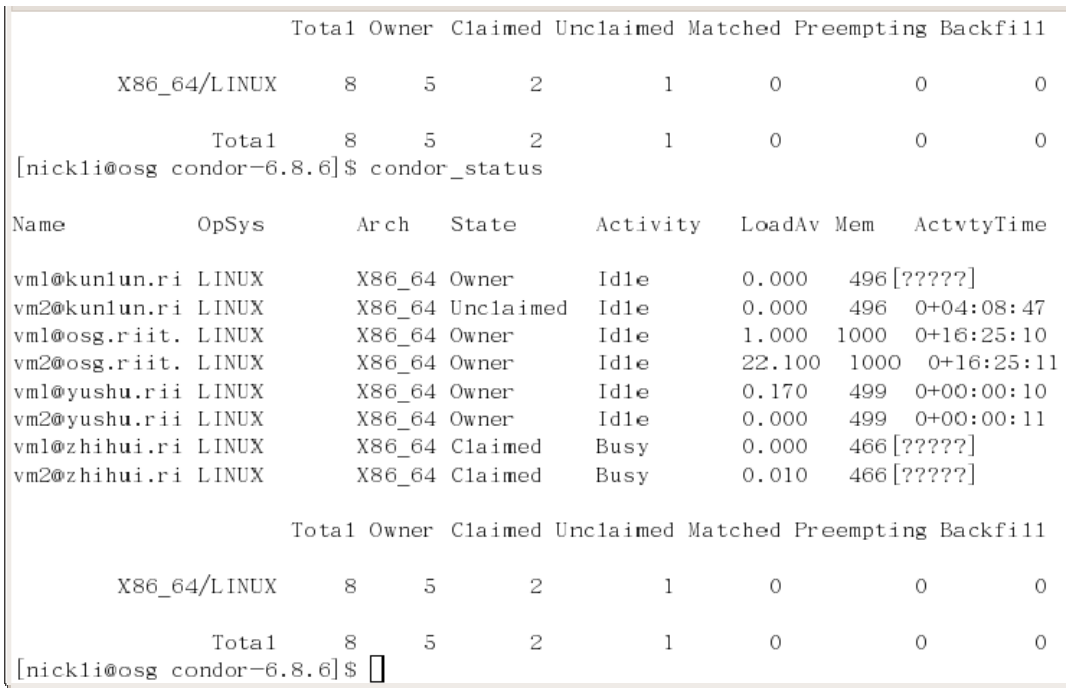

图2.8　搭建的Condor Pool的结构

2.2.4 Condor的任务提交及运行

Condor 的任务提交不同于 Globus，Globus 的任务提交语句为 globus-job-run，该语句的参数就是你要执行的程序，而 Condor 的任务提交语句为 condor_submit，参数是一个任务提交描述文件。该文件对任务提交的细节进行规定，包含如下内容，比如要运行的任务的名称，任务的运行参数，任务的优先级，要求在何种类型操作系统上运行等等。

其中有一个名为 Condor Universe 的变量设置需要我们注意。Condor 有数种运行环境，每种运行环境就称为一个 universe。在这些运行环境中，有两个最基本的运行环境：standard universe 和 vanilla universe。Standard universe 提供任务检查点和任务迁移的功能，但是它有较多的限制条件，比如它需要用 condor_compile 命令将程序重新链接到 Condor 自带的库进行编译，而这对于 DMT 来说并没有多大的实际用处，因为并不是所有时候都能访问 DMT 的源代码而重新编译。而 vanilla universe 则没有什么限制条件，它可以支持类似 DMT 这种不能重新链接到 Condor 自带的库的任务，但是与此同时，由于无法链接到 Condor 自带的库，就丧失了设置检查点和迁移任务的功能。但 DMT 所要处理的数据是实时的，因此

对于 DMT 来说设置检查点和迁移任务就没有多大的意义。所以综上所述，DMT 的运行应当选择 vanilla universe。

下边举个例子来说明任务提交描述文件。先以一个名为 test.sh 的简单 shell 脚本作为任务进行提交，该 shell 脚本放置在机器 IP 为 166.111.137.169 的 /opt/condor/condor-6.8.6/test 文件夹中，它里边的内容如下：

```
pwd
echo “hi”
```

而 test.sh 的任务提交描述文件名为 test.submit，和 test.sh 放在同一文件夹中，内容如下：

```
executable=test.sh
universe=vanilla
output=test.out
log=test.log
should_transfer_files=YES
when_to_transfer_output=ON_EXIT
queue
```

第一行的 executable 规定了所要运行的程序的名称，第二行的 universe 则规定了使用 vanilla 运行环境，第三行的 output 则规定了程序输出重定向到哪，第四行的 log 指明了该任务的日志文件。

执行 condor_submit test.submit 命令，日志文件中相关记录如图 2.9 所示。

```
...
001 (010.000.000) 06/05 16:48:42 Job executing on host: <166.111.137.17:35390>
...
005 (010.000.000) 06/05 16:48:42 Job terminated.
        (1) Normal termination (return value 0)
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00     Run Local Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
        58  -  Run Bytes Sent By Job
        282  -  Run Bytes Received By Job
        58  -  Total Bytes Sent By Job
        282  -  Total Bytes Received By Job
...
[nick1i@yrshu test]$ pwd
```

图2.9　在Condor上运行test.sh的日志文件

由图 2.9 可知，执行命令后，任务被提交到 166.111.137.17 的 35390 端口，任务执行完后，输出重定向到文件 test.out，test.out 文件内容如图 2.10 所示。

```
[nick1i@osg test]$ more test.out
/opt/condor/condor-6.8.6/loca1.yushu/execute/dir_21454
hi
```

图2.10 在Condor上运行test.sh输出的结果

/opt/condor/condor-6.8.6/local.yushu/execute/dir_21454 即是执行 test.sh 中 pwd 这条显示当前目录的命令的结果，而 echo “hi”则是打印 hi 这个英文单词，在 test.out 中也出现了。

由结果可以看出，condor_submit 将 test.out 中指明的程序 test.sh 传送到了 166.111.137.17 的/opt/condor/condor-6.8.6/local.yushu/execute/dir_21454 文件夹中执行，因此执行 test.sh 中的 pwd 命令就会显示如此结果。这也是 Condor 运行任务的一大特色。Condor 会在执行任务的机器上建立一个临时文件夹，将任务执行所需要的程序，输入输出文件等全部放进去，等到任务结束，再将任务执行的结果，也就是输出文件全部拷贝到提交任务的机器上，然后删除执行任务机器上的临时文件夹，释放空间。

2.3 DMT样例程序Rmon

现有一名为 Rmon 的 DMT 样例程序，它用 C++编写而成，编译时需要包含一名为 lscsoft 库中的头文件及链接其他一些专业软件的动态链接库，比如支持快速傅立叶变换的 fftw3.so 动态链接库等等。在它在本机上运行之前，我们需要设置一个名为 LD_LIBRARY_PATH 的环境变量，将该环境变量指向 Rmon 所需链接到的动态链接库的存储路径，这样 Rmon 在运行之前才能链接到它所需要的动态链接库，从而正确执行。

为了方便执行 Rmon，用一名为 standalonerun 的 shell 脚本存储设置 LD_LIBRARY_PATH 的命令以及运行 Rmon 程序的命令。具体的脚本内容如下。

```
export LD_LIBRARY_PATH=
/opt/lscsoft/dol/lib64:/opt/lscsoft/framecpp/lib64:/opt/lscsoft/libmetaio/lib64
./rmon -opt opt -inlists multilist.txt -status current.txt
```

上述第一条命令即为设置 LD_LIBRARY_PATH 变量，第二条命令（即字体较粗的那条命令）则是执行 Rmon 程序。

Rmon 样例程序的功能如图 2.11 所示。

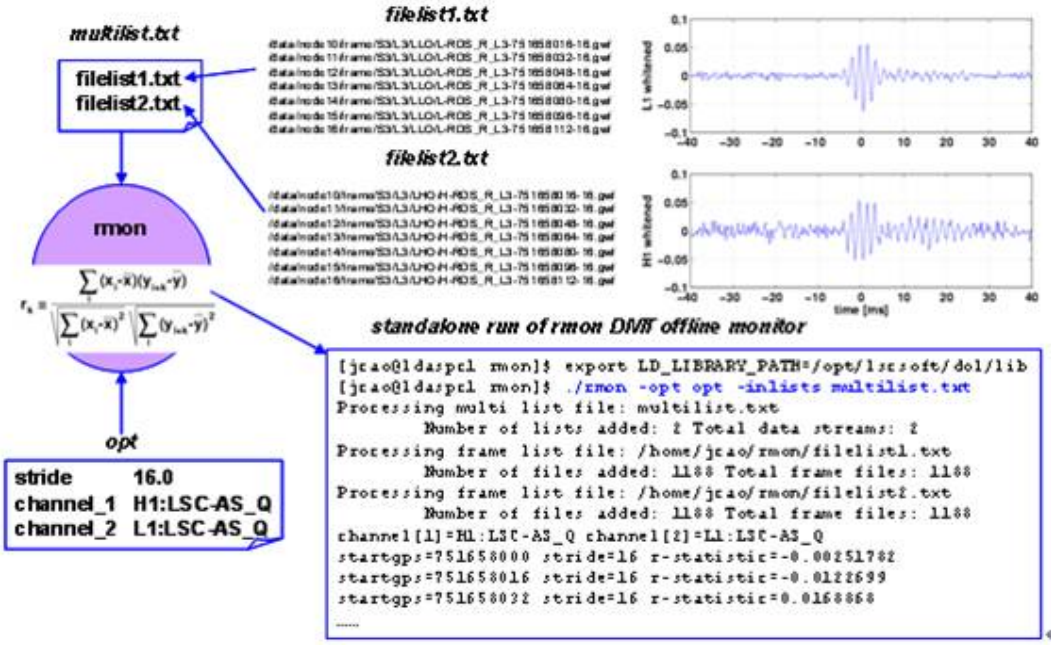

图2.11 Rmon的功能结构图

如图 2.11 所示，multilist.txt 中存储了 filelist1.txt 和 filelist2.txt 的绝对路径，而 filelist1.txt 和 filelist2.txt 中分别存放了两个天文台各自产生的 1188 个数据在硬盘中的绝对存储路径，这些数据都含有时间戳，filelist1.txt 和 filelist2.txt 中的绝对路径所分别指向的数据的时间戳所包含的时间一一对应，因此这构成了 1188 个数据对。

Rmon 依次读取 filelist1.txt 和 filelist2.txt 中的绝对路径所指向的数据，并对这 1188 个数据对依次进行处理，计算得到每对数据对之间的相似度，并实时输出结果。

运行 shell 脚本文件 standalonerun，结果如图 2.12 所示。

```
[nicklli@yushu rmon]$ source standalonerun
Processing multi list file: /home/nicklli/rmon/multilist.txt
        Number of lists added: 2 Total data streams: 2
Processing frame list file: /home/nicklli/rmon/filelistl.txt
        Number of files added: 1188 Total frame files: 1188
Processing frame list file: /home/nicklli/rmon/filelist2.txt
        Number of files added: 1188 Total frame files: 1188
channel[1]=H1:LSC-AS_Q channel[2]=L1:LSC-AS_Q
startgps=751658000 stride=16 r-statistic=-0.00251782
startgps=751658016 stride=16 r-statistic=-0.0122699
startgps=751658032 stride=16 r-statistic=0.0168868
startgps=751658048 stride=16 r-statistic=-0.028074
startgps=751658064 stride=16 r-statistic=0.0363762
startgps=751658080 stride=16 r-statistic=-0.0118638
startgps=751658096 stride=16 r-statistic=-0.00390466
startgps=751658112 stride=16 r-statistic=0.0369562
startgps=751658128 stride=16 r-statistic=-0.0270225
startgps=751658144 stride=16 r-statistic=-0.0431374
startgps=751658160 stride=16 r-statistic=-0.0423102
startgps=751658176 stride=16 r-statistic=0.0325139
startgps=751658192 stride=16 r-statistic=-0.0140542
startgps=751658208 stride=16 r-statistic=-0.0178756
startgps=751658224 stride=16 r-statistic=0.0387879
startgps=751658240 stride=16 r-statistic=-0.0406371
```

图2.12 Rmon本机运行结果

Rmon 调用了 DMT 中的一些工具，并拥有 DMT 的特征，即输入输出均要求实时，且编译及运行均存在依赖性问题。因此我们可以把 Rmon 作为 DMT 的代表，对其进行 Grid Enabling Layer 上的开发，下边的 2.3.2 和 2.3.3 将从最简单的情况入手，即假设 Rmon 在网格上运行的条件已经事先满足（预先将程序和它所需要的数据传到远程机器上，并在远程机器上事先安装好 Rmon 所需要的动态链接库等依赖性文件。），先实现 Rmon 在网格上的简单运行，然后再逐步增加限定条件。

2.4 相关技术

目前实现应用程序在网格中间件上运行的方法纷繁复杂，并没有一个统一的方法，这是因为每个应用程序几乎都有自己截然不同的需求，因此无法用一个通用的网格中间件实现所有应用程序在上边的运行。因此在网格中间件层之上的网格使能层上的开发也是如此，为了满足各个应用程序不同的需求，方法上肯定存在差异，但总的来说，这些方法还是有一个共同点，即它们都对最原始的在单机上运行的应用程序进行了改造，进行了重新编译，以适应特定网格环境下的运行。下边举两个例子。

MW 是 Condor 的一个子项目，它是一个能让主从结构程序轻松实现并行计算的软件框架。MW 是一系列 C++的抽象类，同时向应用程序开发者和网格底层开发者提供了接口。如果要用 MW 让应用程序在网格上跑起来，应用程序的开发者必须重新实现少数几个虚拟函数。同样地，把 MW 框架嫁接到一个新的网格中间件，网格底层开发人员只需要重新实现少数几个虚拟函数就可以了。

MPICH-G2 是消息传递接口的一个网格使能实现。通常地，我们一直在探索特制的，高层的针对网格异构环境的分布式计算模型。但是 MPICH-G2 开拓了一条不同的道路，它主张使用一个众所周知的底层并行计算模型，即消息传递接口（Message Passing Interface，简称 MPI），作为网格编程的基础。MPICH-G2 通过使用 Globus 的服务隐藏了异构性，比如认证，授权，进程创建，进程监控，重定向标准输入和标准输出等等。

与前边提到的相关技术有所不同，本文在实现 Rmon 在网格中间件上的运行时全部采用了 shell 脚本进行开发，并且没有对 Rmon 程序进行改造。Shell 作为 linux 的命令行解释程序，在所有的 linux 操作系统中都存在，这就给网格使能层上的开发带来了便利。

2.5 运行开发平台

网络环境：

清华大学 FIT 楼同一网段内 100M 以太网。

运行平台：

硬件方面为若干台硬件配置相同的计算机，CPU 为 Intel（R）Pentium（R） Dual 双核 1.80GHz，内存 1GB。

软件方面：

操作系统均为为 Linux 系统，Fedora Core 4 Linux。

网格中间件软件为 Globus 4.0.5 和 Condor 6.8.6。它们的下载地址分别为 www.globus.org 和 www.cs.wisc.edu/condor。

第3章 基于网格中间件的Rmon运行

3.1 基于网格中间件的Rmon简单运行

下面介绍基于 Globus 和 Condor，提交 Rmon 到单台远程主机运行的情况，在此设定一个前提，即 Rmon 读取的 1188 组数据和 Rmon 所需要的动态链接库已在该单台远程主机上预备好。我们先实现 Rmon 的简单运行，待后边再逐渐去除前提条件。

3.1.1 基于Globus的Rmon简单运行

Rmon 的运行除了需要那 1188 组数据和动态链接库外，还有一些其他的文件，比如 standalonerun 以及 multilist.txt 等。因此我们可以利用 2.1.3 中提到的 Globus 自带的 scp 远程拷贝命令不用输入密码的优势，把这些文件传送到远程主机上。

在传送 standalonerun 前，我们需要改写一下 standalonerun 文件，前边已经提到在本机运行时 standalonerun 文件内容为：

export LD_LIBRARY_PATH=

/opt/lscsoft/dol/lib64:/opt/lscsoft/framecpp/lib64:/opt/lscsoft/libmetaio/lib64

./rmon -opt opt -inlists multilist.txt -status current.txt

粗体字表示的命令中，采用的均为相对路径，表示程序在当前目录运行。但将 Rmon 提交到远程主机运行时，globus 默认的当前目录为远程主机用户的家目录，比如用户 nickli 的家目录为/home/nickli。而我们的 standalonerun 及 rmon 程序等文件并未存放在家目录，而是放在家目录的一个子文件夹 rmon 中，因此需要将 standalonerun 中的相对路径改为绝对路径，修改后的 standalonerun 为：

export LD_LIBRARY_PATH=

/opt/lscsoft/dol/lib64:/opt/lscsoft/framecpp/lib64:/opt/lscsoft/libmetaio/lib64

/home/nickli/rmon/rmon -opt /home/nickli/rmon/opt -inlists /home/nickli/rmon/multilist.txt -status /home/nickli/rmon/current.txt

接着用 globus-job-run 命令执行任务。

3.1.2 基于Condor的Rmon简单运行

与基于 Globus 的 Rmon 简单运行类似，基于 Condor 的 Rmon 简单运行下，standalonerun 文件也需要从相对路径改为绝对路径，因此两者的 standalonerun 文件相同。

Condor 下的任务提交描述文件如下：

```
executable=standalonerun
universe=vanilla
output=standalonerun.out
log=standalonerun.log
should_transfer_files=YES
transfer_input_files=rmon,filelist1.txt,filelist2.txt,opt,multilist.txt
when_to_transfer_output=ON_EXIT
queue
```

从任务描述文件可以看出，transfer_input_files 规定了在 standalonerun 运行于远程主机之前，应当把哪些文件拷贝到远程主机，供 standalonerun 使用，这和前边提到的用 globus 自带的 scp 命令拷贝 Rmon 所需文件有很多相似之处。

从 166.111.137.44 提交任务到 166.111.137.86 上，基于 Condor 的 Rmon 简单运行结果如下：

```
[nick1i@kun1un test]$ more test.1og
000 (001.000.000) 06/05 20:58:50 Job submitted from host: <166.111.137.44:57703>
...
001 (001.000.000) 06/05 21:09:21 Job executing on host: <166.111.137.86:50692>
...
006 (001.000.000) 06/05 21:09:29 Image size of job updated: 74812
...
005 (001.000.000) 06/05 21:11:58 Job terminated.
        (1) Normal termination (return value 0)
                Usr 0 00:00:48, Sys 0 00:00:08  -  Run Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00     Run Local Usage
                Usr 0 00:00:48, Sys 0 00:00:08  -  Total Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
        62524  -  Run Bytes Sent By Job
        269  -  Run Bytes Received By Job
        62524  -  Total Bytes Sent By Job
        269  -  Total Bytes Received By Job
...
```

图3.1　基于Condor的Rmon简单运行结果

从图中可看出，从任务被提交到任务开始运行，共花费了 631 秒，而从任务开始运行到任务结束共消耗时间 157 秒。重复几次，关于时间消耗，可得下表（单位为秒）：

表3.1　基于Condor的Rmon简单运行的时间消耗

次数	*第一次*	*第二次*	*第三次*	*平均*
从提交到运行	631	6	5	214
从运行到结束	157	159	160	159

从表中可看出，从提交到运行的这段时间波动特别剧烈，第一次花费的时间有 631 秒，而后边两次均 10 秒不到，这是由 Condor 的用合适资源匹配等待请求功能所决定的，可能当时整个 Condor Pool 中所有的机器都无法提供任务所需要的资源，比如当时所有的机器都正在忙碌，因此任务只能等待，而等待的时间有长有短。

观察表中的从运行到结束的时间，我们可以发现三次的时间消耗相差很小，这是因为我们使用的 Condor 的运行环境设置成了 vanilla universe，而 vanilla universe 不支持任务检查点和任务迁移功能，因此任务将只能在同一台电脑上持续进行，不会受到外界因素大的干扰，所以从任务运行到结束的时间的变化也就不会很大了。

3.2 网格使能层上Rmon的运行

在这里，我们将在网格使能层上进行一些开发，以解决第 1 章序言中提到的问题。

3.2.1 改善Rmon的依赖性

前边第 2 章介绍 DMT 样例程序 Rmon 的时候已经提到，Rmon 样例程序的编译和运行需要链接到专业的动态链接库。因此，为了能让 Rmon 能在远程主机上运行，就必须在远程主机上预先装好动态链接库，而这个在网格中实现存在困难，因为远程主机很多时候并不归提交 Rmon 任务的人所有，而安装动态链接库往往需要 root 权限，而 Globus 为了安全考虑禁止了 root 权限。因此可以采取如下的这种办法：在提交 Rmon 任务的主机上预先定义好 Rmon 和动态链接库的依赖关

系，提交 Rmon 任务时，将 Rmon 所需的动态链接库也传送到执行 Rmon 任务的远程主机上，这样同时也免去了在远程主机上进行重新编译的麻烦。

该方法由 shell 脚本实现，过程如下：

图3.2　改善Rmon依赖性方法的流程

第一步中不妨假设文件夹 A 为/home/nickli/hi/rmon-lib，编译配置文件放置在/home/nickli/hi 中。

下边详细解释下第二步中的动态库路径从绝对路径改为相对路径：

最开始 Rmon 的编译配置文件如下：

```
export DOL=/opt/lscsoft/dol
export FRAMECPP=/opt/lscsoft/framecpp
export LIBMETAIO=/opt/lscsoft/libmetaio
g++ -Wall -O3 -I. -I$DOL/include -I$FRAMECPP/include -I$LIBMETAIO/include -g
-c -o rmon.o rmon.cc
g++ -O3 -o rmon rmon.o    -L$DOL/lib64 -L$FRAMECPP/lib64
-L$LIBMETAIO/lib64 -ldmtenv -lmonitor -losc -lframeio -ldmtsigp -lgdstrig
```

-lgdscntr -lhtml -lxsil -lparsl -lgdsbase -lframecpp -lgeneral -lz -lmetaio -ldl -lframeutil -llxr –lxml

其中粗体标识命令中的$DOL/lib64，$FRAMECPP/lib64 ，$LIBMETAIO/lib64 即表示 g++使用了绝对路径进行动态库的搜索，将其修改成相对路径指向文件夹 A，/home/nickli/hi/rmon-lib 后，此条命令为：

g++ -O3 -o rmon rmon.o -L**./rmon-lib** -ldmtenv -lmonitor -losc -lframeio -ldmtsigp -lgdstrig -lgdscntr -lhtml -lxsil -lparsl -lgdsbase -lframecpp -lgeneral -lz -lmetaio -ldl -lframeutil -llxr –lxml

其中加粗的代码即为相对路径，表示的是 g++将会在编译配置文件所在的 /home/nickli/hi 文件夹的子文件夹 rmon-lib 中搜索动态库。

第三步，将 Rmon 及文件夹 A 等相关文件提交到远程主机上，standalonerun 文件中的 LD_LIBRARY_PATH 即为动态库的搜索路径，g++编译器会根据这个变量，去搜索动态库。因此，设置 LD_LIBRARY_PATH 为/home/nickli/hello/rmon-lib。

但该方法存在两个缺点：

1. 并不是所有的动态链接库都可以通过这种方法解决其依赖性问题，尽管大部分可以，但仍有少部分由于程序编写时固有的原因而不能用相对路径进行路径搜索。
2. 该方法只能适用于提交任务机器和执行任务机器的操作系统相同时，才能起作用，因为随着操作系统的变化，动态库的版本会变化，因此即使将库传过去，仍不能保证其能正常运行。

设想的改进办法：

设置一台服务器，上边存储有各种操作系统下的不同版本的动态链接库，当 Rmon 被提交到某台计算机上时，能自动识别该计算机操作系统的版本，然后从服务器上下载对应版本的动态链接库，从而解决其依赖性。这有点类似 Fedora Core Linux 的 yum 自动安装软件功能，也类似于 Ubuntu Linux 的 apt-get。

3.2.2 实现Rmon在Globus上的并行计算

前边已经提到 Globus 并不提供并行计算的功能，但往往为了追求更快的处理速度，仅仅把任务提交到一台远程计算机上进行处理是不够的，因此将任务分发到多台计算机上同时并行计算就是一种很自然的想法，这也是网格使能层应该解决的问题。

并行计算的结构图如图 3.3 所示。

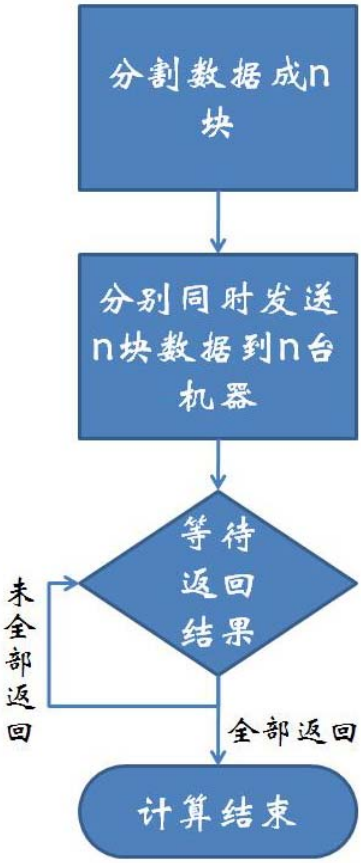

图3.3　并行计算的原理图

下边举例说明，如图 3.4 所示。

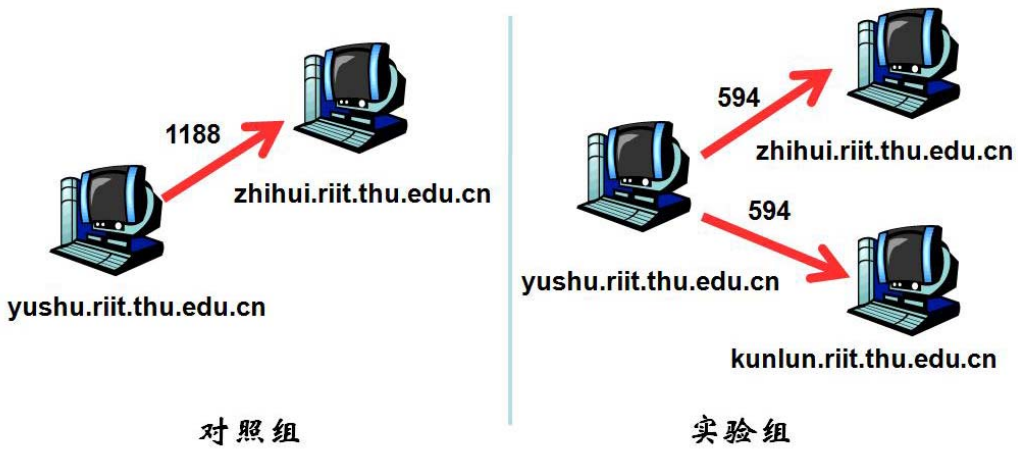

图3.4　并行计算举例

如图 3.4 所示，左边为对照组，右边为实验组。对照组中 yushu 这台机器上有 1188 组数据，将其全部传送到 zhihui 这台机器上处理；在实验组中，则将 1188 组数据平均分成两部分，分别在 zhihui 和 kunlun 上处理，处理后分别将各自结果返回 yushu。对照组和实验组在同一时间段内完成，且这几台机器硬件配置和软件配置几乎相同，又没有运行其他的任务，因此可认为对对照组和实验组的实验条件基本一致。

结果如表 3.2 所示，时间单位为秒。

表3.2　采用并行计算和未采用并行计算的时间消耗对比

次数 / *组名*	*第一次*	*第二次*	*第三次*	*平均*
对照组	155	154	154	154
实验组	91	90	91	91

可见采用并行计算后，计算速度确实得到大幅提高。下边再尝试在不止两台计算机上并发处理，研究不同处理单元下处理时间的变化，得到结果如表 3.3 所示，时间单位为秒。

表3.3　并行计算中并行单元数量对时间消耗的影响

次数 / *并发数量*	*第一次*	*第二次*	*第三次*	*平均*
2	91	90	91	91
3	64	63	63	63
4	49	50	50	50

由表 3.3 可以很清楚的看见，随着并发数量的增多，时间减少的比率逐渐递减，这说明除数据量因素之外的其他因素所发挥的影响逐渐上升，这合乎情理。这同时也告诉我们，并不是并发数量越多越好，数量过多，不仅计算成本较高而且计算效率也不一定会提高。

3.2.3 实现Rmon在Condor上的并行计算

Rmon 在 Condor 上的并行计算的实现与在 Globus 上的原理和方法上都很相似，但也有所不同。

因为 Condor 是通过提交任务描述文件来执行任务，所以在将数据分成 n 块的时候，也得同时生成 n 个对应的任务描述文件。

下边仅举一个例子，此时 n=2，数据块分别为 A 和 B，A 和 B 大小相等，创建了两个分任务，分别去处理 A 和 B，进行了三次同样的并行计算，数据如表 3.4 所示。

表3.4　Rmon在Condor上并行计算的时间消耗分析

次数	*数据块*	*时刻*			时间（秒）	
		任务被递交	*开始执行*	*任务结束*	*从递交到执行*	*从执行到结束*
第一次	A	18:29:08	18:29:13	18:30:33	5	80
	B	18:29:08	18:30:36	18:31:56	88	80
第二次	A	18:56:49	18:57:05	18:58:47	16	102
	B	18:56:49	18:57:25	18:58:59	36	94
第三次	A	19:14:35	19:14:44	19:16:03	9	79
	B	19:14:36	19:16:07	19:17:26	91	79

从表中可发现，处理 B 的分任务是紧跟着处理 A 的分任务提交的，但处理 B 的分任务开始执行的时刻总比处理 A 的分任务晚不少。这是因为处理 A 的分任务提交的时候整个 Condor Pool 中就只有一个 CPU 空闲，因此处理 A 的分任务将这唯一一个空闲 CPU 占据后，对于处理 B 的分任务来讲，整个 Condor Pool 中暂时就没有可匹配的资源。经过一段不确定的时间，出现空闲资源后，处理 B 的分任务就能开始执行了。

每次，处理 A 的分任务和处理 B 的分任务的从执行到结束的时间基本相等，这符合预想，因为 A 和 B 的数据量相等，Condor Pool 中所有的 CPU 的处理速度也一样。

综上，从表中可知，在 Condor 上实现并行计算存在缺陷，即虽然能提高计算速度，但 Condor 的自动匹配资源和请求的功能却很可能使得分发出去的数据块

不能被并行处理，只要有一个数据块找不到匹配资源而不得不等候，则发起并行计算的主机就得一直等待该数据块结果的返回，这大大降低了计算效率。

3.3 Globus和Condor的性能比较

Globus 和 Condor 这两种网格中间件各有所长，哪个更适合 DMT 在其上运行是我们值得考虑的一个问题。下面就提出本人的一些看法：

Globus 比 Condor 更适合 DMT 的运行，原因有如下几点：

1. Globus 提供了完善的安全机制，而 Condor 没有，安全对于 LIGO 这种大型的科研项目来说是不可缺少的。
2. Condor 中虽然提供了任务检查点和任务迁移等 Globus 不具备的功能，但是对于 DMT 这种实时处理数据流的工具来说，检查点和任务迁移的功能就显得可有可无，比如检查点的设置是为了方便恢复已经运行了一段时间的任务，而数据流实时处理中，已经被处理的数据流就丢弃了，不需要重新恢复。
3. Condor 的目标是为高吞吐量计算（High Throughput Computing，简称 HTC）提供支持，这就意味着 Condor 并不在意短期内数据处理量的多少，而是注重长时间内吞吐量的增长，同时这也就决定了 Condor 下的并行计算存在实时处理速度无法稳定，忽高忽低的情况，而这对于实时处理稳定数据流的 DMT 工具来说是不利的。

第4章 结论及展望

4.1 基于网格中间件的Rmon运行结论

在第 3 章中，首先我们设立了一个简单的前提，即 Rmon 所需的数据和动态链接库都已经事先在远程主机上放置好了。在该前提下，我们实现了 Rmon 最为简单的运行，将 Rmon 提交到了单台远程主机上去处理数据。接着我们实现了传送 Rmon 所需动态链接库到远程主机的功能，部分解决了 Rmon 程序依赖的问题。然后我们还在 Globus 和 Condor 两种网格中间件上分别实现了 Rmon 在多台远程主机上的同时运行，提高了 Rmon 的处理速度。最后还对 Globus 和 Condor 进行了性能上的一些比较，得出的结论是 Globus 比 Condor 更适合 DMT 工具箱的运行，而实际情况中 LIGO 也恰恰采用了 Globus。

而前边实现的这些功能都是建立在调用 Globus 和 Condor 命令的基础上，我们只需要执行相应开发好的集成化程序，再加上更改几个参数，就能实现这些功能，这也达到了当初的目的，即屏蔽 Globus 和 Condor 的技术细节并提供一个易用的接口给用户，用户只需要输入一条命令，然后耐心等待结果就可以了。

4.2 展望

本文对两种典型的网格中间件进行了网格使能层上的一些开发，实现了一些基本的功能，但仍存在不少问题急需解决：

首先，网格使能层所做的这些开发在正常使用的情况下，对于用户来说是易用的，但是一旦在使用过程中网格使能层下边的网格中间件出错，那么用户还是得了解其技术细节，才能纠正错误。如何才能提供更好的错误提示及调试功能，是以后进一步研究的内容。

其次，第 3 章中我们曾尝试解决 Rmon 的程序依赖性问题，但仍存在很大问题，在 Linux 中动态链接库和 Window 下的 dll 文件有所不同，是不能编译进二进制程序的，而为了追求性能，又不能采用兼容性好但效率较低的 java 等语言进行编译，而如果程序的依赖性问题不能结果，网格计算等等也就无从谈起了，因此网格上的依赖性问题是一个网格中亟待解决的问题。

最后，网格使能层上开发的这些功能并没有集成到一块，这也是因为各个子功能目前还存在缺陷，集成到一块反而没有多大意义。但今后随着各个功能的完善，还是有必要将它们集成到一块的。

插图索引

表格索引

参考文献

[1] LIGO - Laser Interferometer Gravitational-Wave Observatory http://www.ligo.caltech.edu

[2] LSC – LIGO Scientific Collaboration. http://www.ligo.org

[3] Overview of LIGO. http://www.ligo-la.caltech.edu/contents/overviewsci.htm

[4] LIGO. Data from the LIGO I Science Run [R]. California Institute of Technology: LIGO, 2001:1

[5] Bart Jacob. Introduction to Grid Computing [M]. 1st ed. USA: International Business Machines Corporation,2005:19-20

[6] 中国网格信息中转站. 网格技术可实现计算资源共享 [EB/OL] . [2005-6-23]. http://www.chinagrid.net/dvnews/show.aspx?id=1197&cid=23

[7] Matt Haynos. 网格观点：网格计算——下一代分布式计算 [EB/OL] . [2006-7-20]. http://www.ibm.com/developerworks/cn/grid/gr-heritage/

[8] 都志辉. 网格计算 [M]. 第一版. 北京:清华大学出版社,2002:7-12

[9] 尘昌华，刘方爱.网格计算中间件的研究 [J]. 信息技术与信息化：网络与通信，2008，1：21-23

[10] Globus Alliance. About the Globus Toolkit [EB/OL]. http://www.globus.org/toolkit/about.html

[11] Luis Ferreira. Introduction to Grid Computing with Globus [M]. 2nd ed. USA: International Business Machines Corporation,2003:156-159

[12] Bart Jacob. Enabling Applications for Grid Computing with Globus [M]. 1st ed. USA: International Business Machines Corporation,2003:33-34

[13] Luis Ferreira. Introduction to Grid Computing with Globus [M]. 2nd ed. USA: International Business Machines Corporation,2003:164-166

[14] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the Grid [M]// F. Berman, A. Hey and G. Fox. Grid Computing-Making the Global Infrastructure a Reality. USA: WILEY, 2003: 304-305

[15] Jean-Pierre Goux, Sanjeev Kulkarni, Jeff Linderoth, Michael Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid [J]. High-Performance Distributed Computing: The Ninth International Symposium, 2000: 43-50

[16] Cornell University Library. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface [EB/OL]. [2002-06-25]. http://arxiv.org/abs/cs/0206040v1

致　　谢

首先，要感谢我的导师，尊敬的曹军威研究员。他平时不仅对我的毕业设计多有关心和指导，在其他方面也对我很关心。另外他严谨的科研作风对我触动也很大，在将来研究生阶段的学习当中，这种严谨的科研作风是我需要不断学习和改进的。

此外，也感谢实验室里的张文师兄和王震师兄。张文师兄拥有扎实的科研功底，他给予我的帮助让我在毕设的前期少走了很多的弯路。而王震师兄则对我的毕业设计的内容及论文的撰写提出了不少的建议。

最后，再一次感谢诸位关心和帮助过我的人。

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：_____　　日　期：_____

附录A 外文资料的调研阅读报告

LIGO and Grid Enabling Layer

Albert Einstein predicted the existence of gravitational waves in 1916 as part of the theory of general relativity. The Laser Interferometer Gravitational-Wave Observatory funded by US National Science Foundation (NSF) is a facility dedicated to the detection of cosmic gravitational waves and the harnessing of these waves for scientific research[1]. LIGO interferometers produce audio-band time-series data digitized at a number of frequencies. In total, every second, the LIGO interferometers will generate between 7-9 MB of data as time series[2]. Because the data are all time series, it requires data analysis to proceed at same rate as data acquisition. And Low latency analysis is needed if opportunity is needed to provide alerts to astronomical community in the future. And LIGO's scientific pay-off is bounded by the ability to perform computations on the data. These characteristics can become true by using Grid Computing. We can call it LIGO Data Grid.

LIGO Data Grid is a combination of LIGO Scientific Collaboration computational and data storage resources with Grid Computing middleware to create a distributed gravitational-wave data analysis facility.

Grid Computing, which was called Metacomputing early on, is distributed computing and parallel computing taken to the next evolutionary level. Although it has much in common with both distributed and parallel systems, yet also differs from these two architectures in important ways. Its goal is to create the networked virtual supercomputers out of a large collection of connected heterogeneous systems sharing various combinations of resources.

To achieve the goal, it requires programming models and interfaces radically different from those used in distributed or parallel systems. So we need Grid Computing software which has some significant advances

in mechanisms, techniques and tools. For example, the Globus Toolkit, and Condor are the representatives of Grid Computing software. Their recommends are shown as below.

1. Globus

Globus Toolkit's greatest feature is its security mechanism, like the certificates for authentication/authorization. It has three pyramids of support built on top of a security infrastructure[3].

1) Resource management

The Resource management pyramid provides support for:

a) Resource allocation

b) Submitting jobs: Remotely running executable files and receiving results.

c) Managing job status and progress

2) Data management: It provides support to transfer files among machines in the grid and for the management of these transfers.

3) Information services: It provides support for collecting information in the grid and for querying this information, based on the Lightweight Directory Access Protocol.

All of these pyramids are built on top of the underlying Grid Security Infrastructure (GSI). This provides security functions, including single/mutual authentication, confidential communication, authorization and delegation[3].

We can use Globus Toolkit to build a Gridftp. To move large amounts of data and files across wide area networks is a hard thing, for example in the grid. But the Globus striped Gridftp framework, a set of client and server libraries designed to support the construction of data-intensive tools and applications can solve this problem.

2. Condor

Different from Globus Toolkit, Condor is a specialized job and resource management system (RMS) for compute-intensive jobs. It provides a job management mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their jobs to Condor,

and Condor subsequently chooses when and where to run them based upon a policy, monitors their progress, and ultimately informs the user upon completion[4].

To build a high-throughput computing environment, Condor should provide large amounts of fault-tolerant computational power over prolonged periods of time by effectively utilizing all resources available to the network. Some related distinctive mechanisms are shown as below.

ClassAds: The ClassAd mechanism in Condor provides an extremely flexible and expressive framework for matching resource requests with resource offers. ClassAds allow Condor to adapt to nearly any desired resource utilization policy and to adopt a planning approach when incorporating Grid resources.

Job checkpoint and migration: With certain types of jobs, Condor can transparently record a checkpoint and subsequently resume the application from the checkpoint file. A periodic checkpoint provides a form of fault tolerance and safeguards the accumulated computation time of a job. A checkpoint also permits a job to migrate from one machine to another machine, enabling Condor to perform low-penalty preemptive-resume scheduling.

Remote system calls: When running jobs on remote machines, Condor can often preserve the local execution environment via remote system calls. Remote system calls is one of Condor's mobile sandbox mechanisms for redirecting all of a jobs I/O-related system calls back to the machine that submitted the job. Therefore, users do not need to make data files available on remote workstations before Condor executes their programs there, even in the absence of a shared file system.

With these mechanisms, Condor can manage jobs and resource more effectively. Condor can also scavenge and manage wasted CPU power from otherwise idle desktop workstations across an entire organization with minimal effort. For example, Condor can be configured to run jobs on computer only when the keyboard and CPU are idle. If a job is running on a computer when the user returns and hits a key, the Condor can transmit

the job to a different computer and resume the job from where it is hold on.

In the previous summary, we have introduced what is LIGO and the Grid Middleware that LIGO uses. But if we want put applications on the Grid to run, we will meet some problems. Until now the Grid Middlewares are not so convenient, for example they can't parallelize the application directly, and they can't provide applications with portability. The two problems mentioned are the basic problems in Grid, bacause if the two problems are not properly solved, applications can't run on the Grid.

To solve these problems, people has tried a lot. We can add a new layer on Grid Middleware, it can provide a friendly interface to end users. The new layer is called Grid Enabling Layer. Below are two examples.

1. MW[5]

MW is a sub-project of Condor. It is a software framework that allows a user to easily parallelize a master-worker application on Grid resources. MW is a set of C++ abstract classes providing interfaces to both application programmer and Grid-infrastructure programmer. To Grid-enable an application with MW, the application programmer must re-implement a small number of virtual function. Likewise, to port the MW framework to a new Grid software toolkit, the Grid infrastructure programmer need only re-implement a small number of virtual functions.

2. MPICH-G2[6]

MPICH-G2 is a Grid enabled implementation of the Message Passing Interface. Typically, we explore specialized, often high-level, distributed programming models for heterogeneous Grid environments. But MPICH-G2 explores a different approach, it advocates the use of a well known low-level parallel programming model, the Message Passing Interface(MPI), as a basis for Grid programming. MPICH-G2 hides heterogeneity by using Globus Toolkit services for such purposes as authentication, authorization, executable staging, process creation, process monitoring, process control, communication, redirection of standard input and output, and remote file access.

References

[1] Overview of LIGO. http://www.ligo-la.caltech.edu/contents/overviewsci.htm

[2] LIGO. Data from the LIGO I Science Run [R]. California Institute of Technology: LIGO, 2001:1

[3] Luis Ferreira. Introduction to Grid Computing with Globus [M]. 2nd ed. USA: International Business Machines Corporation,2003:156-159

[4] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the Grid [M]// F. Berman, A. Hey and G. Fox. Grid Computing-Making the Global Infrastructure a Reality. USA: WILEY, 2003: 304-305

[5] Jean-Pierre Goux, Sanjeev Kulkarni, Jeff Linderoth, Michael Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid [J]. High-Performance Distributed Computing: The Ninth International Symposium, 2000: 43-50

[6] Cornell University Library. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface [EB/OL]. [2002-06-25]. http://arxiv.org/abs/cs/0206040v1