摘 要

网格计算是构筑在互联网基础之上的新兴分布式计算技术。它通过整合分布在各地的资源，为动态变化的虚拟组织成员提供更为广泛的资源共享。国内外纷纷开展了网格技术的研究，而其中的网格作业管理是网格技术的一个重要方面。

本文以金融领域对网格计算的应用需求为背景，首先针对网格计算在金融领域的应用现状，在深入分析和研究网格作业管理技术的基础上，结合企业计算网格的特点和面向服务体系架构的优势，提出了一种企业计算网格作业管理系统体系架构。然后，应用该架构设计并实现了网格作业管理系统的核心组件，即作业调度管理系统。其设计思想是根据金融作业的特点，将一组相关的作业抽象成会话，并基于会话进行作业调度。在此基础上，为了满足需要及时响应的金融作业需求，设计了一种改进的基于优先级的作业调度策略，并将其应用到了系统中。

最后，根据设计目标对网格作业管理系统的各项指标进行了测试和分析。测试表明，在具有复杂分布式计算需求的金融领域，系统提供了一种面向服务的高性能网格解决方案。

关键词：网格计算　面向服务体系架构　作业管理系统　作业调度

Abstract

Grid computing is a new distributed computing technique which is based on the internet. It can provide the members of dynamic changing organizations with wider sharing resources. As researchers taking many works on the grid computing technique both in home an abroad, grid job management is becoming an important part of grid technology.

This paper takes the using of grid computing in financial field as background. Firstly, after analysising and researching the using status of grid computing in financial field, considering the characteristics of enterprise computing grid and the advantages of server-oriented architecture (SOA), this paper designs a architecture of job management system in enterprise computing grid. Secondly, by using this architecture, the paper completes the design and implementation of job scheduling management system which is the core components of grid job management system. According to characteristics of financial job, the design idea is abstracting a series of job to session and making job scheduling on session. After the former step, the paper has given a strategy of job scheduling based on PRI, which is fulfilled the requirement that responding in time with financial job, and has used it into the system.

Finally, the paper tests and analyzes the job scheduling management system according to the design goal. With the complex requirements of distributed computing in financial field, the system provides a high performance computing (HPC) service-oriented software solution.

Keyword: Grid Computing SOA Job Manager System Job Scheduling

目 录

第一章 绪论

本章首先阐述论文的写作背景和意义，然后分析了国内外研究现状，最后给出了本文研究目的、研究重点和章节安排。

1.1 论文背景

信息化的浪潮下，人类的应用需求正朝着高性能、多样化、多功能方向发展，需要计算能力更强大的计算机。许多大规模科学计算不仅需要一台超级计算机，更需要多种机器组成、多个系统合作、多个科学仪器设备相连的网络虚拟超级计算机。2008 年 9 月 10 日，由欧洲核子研究中心建设的、目前世界上最大、最强的粒子加速器——大型强子对撞机正式开始运作[1]。作为有史以来规模最大的科学实验，大型强子对撞机将产生海量的数据：每秒产生的数据流将达到大约 700 兆字节，每年达到 1.5 万亿字节，这种“高产”要持续 10 到 15 年，每年产生的数据足以装满 300 万张 DVD，如果将这些数据装进 CD 并搭成高塔，其高度可达到珠穆朗玛峰的两倍。依靠传统的计算方式很难满足现在的科学需要。正是这些需求鼓励人们在互联网基础上把现有的利用率不高的分散在不同地理位置的、异构的、动态的资源通过高速网络连接在一起，整合成一台虚拟的超级计算机，其中每一台参与计算的计算机就是一个“节点”。而整个计算是由成千上万个“节点”组成的“一张网格”，这种计算方式叫网格计算[2]。这样组织起来的“超级计算机”有两个优势，一个是数据处理能力超强；另一个是能充分利用网上的闲置处理能力。欧洲核子研究中心也正是采用网格计算来处理海量实验数据。

网格的价值并不局限于科学计算，在金融领域也有很好的前景。在金融领域，核心应用系统经常被成千上万的客户频繁访问，还不时出现业务“巅峰”，随着中间业务的深入开展，这种现象将有增无减；同时，许多金融业务诸如风险管理、成本分析、产品定价、利率计算、随机建模和蒙特卡洛模拟等都需要进行大量的数值计算。网格计算恰好能满足上述多方面的应用需求，发挥它的独特作用和优势，从而增强金融企业的核心竞争力，促进金融业的可持续发展。

随着网格技术的不断发展和成熟，网格技术迅速的扩展到金融行业的商业应用中[3]。企业网格已经被用于支持现实世界中的商业应用并带来了切实的商业利益和实际价值。通过企业网格，可以帮助那些在各地有许多分支机构的大型金融企业或机构共享资源、改进内部协作并获得更好的 IT 投资回报。许多金融机构特别是跨国机构，如美国摩根-大通银行、摩根斯坦利投资银行、法国兴业银行和 Charles Schwal 证券公司等都纷纷开始构建和使用企业网格。网格正在成为产品创新和商业智能的最佳工具。利用网格，金融企业不仅能够实现各类所需数据的集成和跨

整个交易链的协作，有利于个性化创新产品的设计和推出，而且还能够利用扩展的计算功能减少开发周期，降低开发成本和缩短进入市场所需的时间。利用网格，金融企业可以开展大型的数据控制、数据智能和数据研究等商业智能项目，从中获取业务发展的相关信息和知识，全面提升银行的核心竞争力。如果采用传统方式这些项目一般需要很长时间（数天、数周甚至更长）。网格计算则能充分利用未用的计算资源，大大加快分析速度，提高分析精度。

网格作业管理系统负责管理整个计算网格的资源，并协调上层的应用需求。随着网格节点数量的日益增多以及ERP(Enterprise Resourse Planning)、CRM(Customer Relationship Management)、企业智能和供应链管理等关键任务应用的实施，数据中心的复杂性与日俱增，无论是用户的业务应用还是后台的IT架构都越来越复杂，在这种情况下，网格作业管理系统变得越来越重要。目前，很多用户IT环境中普通存在的管理效率低，运营成本高昂和IT资源利用率低等都是因为没有一个好的网格作业管理系统。因此，网格作业管系统就成了网格应用的关键因素，国内外纷纷开展了对网格作业管理系统的研究。

1.2 国内外研究现状

作为二十一世纪科学的新基础，网格已成为 Internet 的第三次浪潮，在世界各国引起了前所未有的关注和重视。当前已有的几十种作业管理系统，在目标、结构、功能和实现上各有差异，从不同侧面反映了网格作业管理系统所应具备的特性。以下详细分析当今具有代表性和影响力的几种作业管理系统：

1. LSF (Load Sharing Facility)

LSF[4]是由加拿大 Platform 计算公司研制与开发的，由 Toronto 大学开发的 Utopia 系统发展而来。从强大的功能和广泛使用的角度看，LSF 可谓是一个成熟的网格作业管理系统。LSF 不仅用于科学计算，也用于企业的事务处理。LSF 的主要特点是：支持多种操作系统，包括 Windows NT 和 Windows 2000；支持批处理作业，串行作业和 MPI(Message Passing Interface)，PVM(Parallel Virtual Machine)并行作业；支持检查点操作和进程迁移；具有高可用性，消除单一故障点；提供了抢占式调度和关键资源保障，保证紧急作业的调度；可通过逻辑表达式创建作业依赖图，提供对依赖性作业的支持；提供了多种调度策略；动态的负载平衡与负载监测，负载指标包括节点状态、运行队列长度、CPU 利用率、分页速率、登录用户数、空闲时间、可用交换空间、可用存贮器、/tmp 目录下的可用空间；提供了完整的负载共享库；具有强大的资源管理功能。

2. CONDOR

CONDOR[5]是由威斯康星大学开发的网格作业管理系统。充分利用工作站的

空闲时间是 CONDOR 的最显著特征。当工作站空闲的时候，便把它纳入到资源池中。当工作站的主人开始使用该工作站时，CONDOR 便将运行在该工作站上的作业迁移到其它节点上继续运行，工作站主人对该工作站拥有最高优先级和完全的控制权。此外提供了队列机制、调度策略、优先级方案、资源监控、资源管理等功能。

CONDOR 的主要特征是：充分利用工作站的空闲时间；用户只需与库函数重新链接便可利用 CONDOR 提供的检查点和进程迁移功能；对于远程执行的进程，本地的执行环境被保留；工作站主人对该工作站拥有最高优先级和完全的控制权；作业保证彻底完成，不会因为系统的故障或工作站的退出而终止；本地磁盘空间不会被 CONDOR 作业所占用；对网络资源、数据传送和检查点操作的有效监控；对网络资源、CPU 的协同调度。

它的不足之处在于：它首先虽然它支持分布式环境，但是工作站的过于自治会带来很大的开销；其次它是一个高吞吐量的系统，注重的是高吞吐量而不是响应时间，所以对单个作业来说，响应时间偏长。

3. PBS (Portable Batch System)

PBS[6]最初由 NASA 的研究中心开发，为了提供一个能满足异构计算网络需要的软件包，特别是满足高性能计算的需要。它力求提供对批处理的初始化和调度执行的控制，允许作业在不同主机间路由。PBS 的独立的调度模块允许系统管理员定义资源和每个作业可使用的数量。调度模块存有各个可用的排队作业、运行作业和系统资源使用状况信息。使用它提供的 PCL, BACL, C 三种过程语言，它的调度策略可以很容易被修改，以适应不同的计算需要和目标，即系统管理员可以方便地实现自己的调度策略。

PBS 的主要特点有：代码开放，免费获取；支持批处理、交互式作业和串行、多种并行作业，如 MPI、PVM、HPF(High Performance Fortran)；提供 TCL, BACL, C 三种过程语言，容易实现新的调度策略；提供文件传送功能，File Stage-in 和 Stage-out；满足 POSIX1003.2d 标准；支持作业依赖；自动的负载平衡；完整的安全认证；提供了完整的 API，方便新的调度器的开发；提供用户影像功能，使 PBS 能用于用户不一致的系统中。

但是它也有不足：首先它主要适用于集群或工作站上的作业调度，是一种集中式的控制方式，不便网格中的扩展；其次用户对它的使用也很复杂，需要针对不同的作业写出脚本。

1.3 论文主要工作

在完成本论文的选题及写作过程中，论文的主要工作包括：

1. 研究了网格作业管理技术，在对网格作业管理系统架构和 SOA 架构进行深入分析的基础上，提出了一种基于 SOA 的网格作业管理系统体系架构，并对系统的各功能模块和服务流程进行了分析与设计。
2. 应用所提出的系统架构，设计并实现了网格作业管理系统的核心组件，即作业调度管理系统。其主要设计思想是根据金融作业的特点，将一组相关的作业抽象成会话，并基于会话进行作业调度。为了满足金融领域的小数据量、需要及时响应的作业需求，在研究作业调度策略的基础上，设计了一种改进的基于优先级的作业调度策略，并将其应用到了系统中。
3. 实现了一个专用于在分布式环境中诊断系统性能的测试工具，利用该工具对作业调度管理系统的各项指标（可用性、可靠性、可伸缩性和性能等）进行了测试分析。

1.4 论文组织结构

本文共分六章，章节安排如下：

第一章：绪论。本章主要介绍了论文的研究背景、研究现状、主要内容和组织结构。

第二章：网格作业管理系统相关技术概述。本章主要对网格、SOA 和作业管理等相关技术进行了研究。

第三章：企业计算网格作业管理系统体系架构分析与设计。本章首先分析了金融领域的应用需求，接着深入研究了作业管理系统模型，然后在此基础上结合 SOA 完成了对企业计算网格作业管理系统体系架构的设计，最后设计了系统服务流程。

第四章：作业调度管理系统的设计与实现。基于所提出的系统架构设计并实现了网格作业管理系统的核心组件，即作业调度管理系统。本章详细论述了系统各主要功能模块的设计及实现思路。其中着重研究了作业调度策略，设计了一种改进的基于优先级的作业调度策略，并将其应用到了系统中。

第五章：系统测试。本章首先实现了能在分布式环境中诊断系统的测试工具，然后利用该工具从可用性、可靠性、可伸缩性与性能等方面对系统进行了测试分析。

第六章：结束语。本章总结了本文的工作，并分析了设计、实现中需要进一步完善的工作。

第二章 网格作业管理相关技术概述

互联网技术不断发展，对人们的思维方式、工作模式以及生活理念都产生了巨大的影响与冲击。以 E-mail 为主要应用的第一代 Internet 把遍布于世界各地的计算机用 TCP/IP 协议连接在一起；第二代 Internet 则通过 Web 信息浏览及电子商务应用等信息服务，实现了全球网页的连通；第三代 Internet 将试图实现互联网上所有资源的全面连通，包括计算资源、存储资源、通信资源、软件资源、信息资源、知识资源等，这就是网格计算。网格计算的出现将为信息产业带来无限商机。

2.1 网格与 SOA 概述

2.1.1 网格概述

1. 网格概念

网格的概念是在网格技术的不断发展中逐步明确的。美国阿尔贡国家实验室(Argonne National Laboratory)的资深科学家、美国著名的网格计算项目 Globus 的主持人之一 Ian Foster，曾在 1998 年主编过题为《网格：一种新计算的基础设施的蓝图》一书。他在这本书中首次定义了网格概念：“网格是构筑在互联网上的一种新兴技术，它将高速互联网、高性能计算机、大型数据库、传感器、远程设备等融为一体，为科技人员和普通老百姓提供更多的资源、功能和交互性。互联网主要为人们提供电子邮件、网页浏览等通信功能，而网格功能则更多更强，能让人们透明地使用计算、存储等其它资源。”

2000 年，Ian Foster 在《网格的剖析》这篇论文中把网格进一步描述为“在动态变化的多个虚拟机构间共享资源和协同解决问题”。

2002 年，Ian Foster 在《什么是网格?判断是否是网格的三个标准》从三个方面更清晰地定义网格，他认为网格是一个满足如下三个条件的系统：

1) 在非集中控制的环境中协同使用资源。因为资源和使用者在不同控制域中，网格必须保证节点高度自治。资源可以动态的加入或者撤出网格计算。
2) 使用标准的、开放的和通用的协议和接口。网格是由多用途协议和接口来构建的，该协议将能解决诸如鉴别、授权、资源发现和资源访问等一些基本问题。
3) 提供非平凡的服务。资源是在网格系统的调度下利用的，以提供多种服务质量，满足不同的用户需求。网格允许按协作的方式来使用其成分资源，以提供各种各样的服务内容。

2. 网格分类

网格在不同的应用领域中都形成了自己的风格，例如科学研究网格、军事网格、地球系统网格、游戏网格和教育网格等等，但一些网格有相同的特点，可以对它们进行划分。网格是一种规模庞大的系统，本文以不同的角度为出发点，对网格进行分类[7]。按照网格系统所基于的组织结构的规模，网格可分为：

1) 部门网格

部门网格被部署用来解决企业内部的某个部门的分布式计算或存储问题。部门网格中的资源不能被其它部门共享。

2) 企业网格

企业网格将企业内部的各种资源统一管理并提供给企业内部的所有用户。根据 Platform Computing 公司的定义，企业网格被部署在一个大型企业或组织内，但地理位置可以广域分布，不同地理位置的资源可以共享。这种共享在企业防火墙的范围之内。

3) 跨域网格

跨域网格建立在企业、其合作伙伴或客户之间，网格资源在 VPN(Virtual Private Network)内部可用。

4) 全球网格

全球网格是在 Internet 上建立的网格，企业组织可以从 Internet 上的服务提供者购入部分或全部需要的服务项目。

而如果按网格系统的功能划分，网格系统又可分为以下几种[8]：

1) 计算网格

计算网格主要用来完成大规模计算问题，可以处理关键数据，或是为其他负载较重的计算机分担任务。

2) 数据网格

数据网格为同一机构中的所有数据存储库提供分布式存储方案和统一界面，通过这个界面可以实现对数据的查询、管理和保护。

3) 信息网格

信息网格是要利用现有的网络基础设施、协议规范、Web 和数据库技术，为用户提供一体化的智能信息平台，其目标是创建一种架构在 OS 和 Web 之上的基于 Internet 的新一代信息平台和软件基础设施。在这个平台上，信息的处理是分布式、协作和智能化的，用户可以通过单一入口访问所有信息。信息网格追求的最终目标是能够做到服务点播(Service On Demand)。

4) 服务网格

服务网格采用 Web service 和网格计算技术，遵循 OGSA(Open Grid Service Architecture)的标准，面向企业集成、支持服务连接、管理、集成、优化和运行。

服务网格将成为商业网格系统的一个重要发展方向，它为实现多企业或部门之间广域分布业务应用的集成和协同提供了按需服务、系统互操作和可监控等方面的有力支持。

5) 设备网格

设备网格指用网格管理分布在各地的贵重仪器系统，提供远程访问仪器设备的手段，提高仪器的利用率，方便用户的使用。

3. 企业计算网格特点

根据以上分类，本文研究的网格类型是企业计算网格，主要针对这一类型的网格的特点进行分析。

在 IBM.、Oracle、Sun 等公司的推动下，网格计算逐渐从科学领域进入商业领域。企业计算网格不是学术领域和科学领域所普遍使用的无限制、无结构网格。2004 年 4 月 20 日，以 Oracle 公司为首的企业网格联盟 EGA(The Enterprise Grid Alliance)正式成立，并提供参考模型、安全建议和规范，使企业能够运行网格计算。EGA 的主席 Deutsch 认为科学网格和企业网格主要有下面两方面的区别[9]：

1) 用于科研的网格建立在广泛的分布式系统上，且这些系统的所有者也是分布式的；而企业网格系统位于一个小的范围内，并且只被某个主体所拥有。
2) 运行在网格上的科学研究应用倾向于使用广泛分布的计算资源来解决一个大的问题；而运行在企业网格上的企业应用和特定商务软件大多是面向事务的。

4. 网格的发展

网格概念是借鉴电力网提出来的，其目标是实现网络虚拟环境上的高性能资源共享和协同工作，消除信息孤岛和资源孤岛，希望用户在使用网格时，就如同现在使用电力一样方便。网格的发展经历了三个阶段[10]：

第一阶段是萌芽期，开始于 20 世纪 90 年代早期，主要是千兆网的实验床以及一些元计算实验；

第二阶段是实验期，时间是 20 世纪 90 年代中晚期，出现了一些开创性和奠基性的研究项目，如 I-WAY 项目，学术性研究 Globus、Legion 以及一些应用；

第三阶段是迅速发展期，本世纪以来，出现了大量的计算服务网格研究和应用项目，出现了影响很大的组织——全球网格论坛 GGF(Global Grid Forum)，致力于制定全球网格计算的标准和规范。同时，网格计算也不再仅仅局限于科学研究，工业界与学术界联盟，正致力于网格计算在更广泛的科学、工程和商业领域得到推广和应用。

5. 网格相关技术

自 Internet 诞生以来就不断追求更有效的互联软件技术。网格相关技术主要包括 Web Service, OGSA, WSRF(Web Service Resource Framework)等，以下分别介绍：

1) Web Service 技术[11]。提供永久的无状态服务，实现了程序间远程访问透明性。上个世纪 90 年代基于 TCP/IP 的 Internet 的盛行，促使了 Web 网页的普及，同时网页的普及反过来促进 Internet 的发展。在技术上实现了分布于全球范围的网页定位与网页间（通过超连接技术实现）相互连接，各种网页查找技术与方法正不断涌现和高速发展中。IBM 公司把这种 Web 网页定位与连接技术扩展应用到了远程程序访问中，首先提出全球范围的程序定位与相互访问技术——称之为 Web Service 技术。Web Service 技术实现了 Web Service 的定位、访问与管理，是互联网工业的事实标准。
2) OGSA 体系结构[12]。网格计算研究领域最有影响的两大分支——以 Globus 为代表的计算网格和商业公司倡导的 Web Service，在经历了各自的独立发展之后，于 2002 年年初走到了一起。这种结合的直接结果就是产生了 Globus 研究小组和 IBM 联合提出的 OGSA。开放网格体系结构只有临时的有状态网格服务，实现了网格功能的多样化。虽然 Web Service 提供的功能有限，应用烦琐，但实现了分布式程序访问。OGSA 是明确提出的第一个基于 Web Service 的网格体系结构，具有理论及概念价值。但是 OGSA 具有先天与后天的双重缺陷，导致它诞生不足一年既被新的网格技术规范 WSRF 所取代。OGSA 的先天不足在于 OGSA 只提供网格服务，没有抽象出网格资源；后天不足在于 OGSI 把 Web Service 的 WSDL 扩展为 GWSDL，失去了应用开发技术上的物质基础支持，使得 OGSA 最终成为空中楼阁。
3) WSRF 规范[13]。为了更好与 Web Service 相融合，2004 年提出了 Web 服务资源框架 WSRF 以及一系列基于 Web Service 的规范(WS-Notification, WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup, WS-BaseFaults)。WSRF 在 Web Service 的永久无状态服务基础上加入有状态的（本质上是临时的）资源。虽然 WSRF 修正了 OGSA 的失误，即提供有状态的网格资源描述客观静态信息，又兼容了 Web Service 的工业应用开发技术，但仍只是笼统地表现为网格中间件，没有发展出网格基础设施与虚拟操作系统等关键概念和技术，有待于进一步发展和完善。

2.1.2 SOA 概述

1. SOA 概念

SOA[14][15]是面向服务的体系结构，它是一种关注服务的互操作、易于集成、可扩展和安全访问特性的分布式体系结构。这类系统是将异构平台上应用程序的不同功能部件（称为服务）通过这些服务之间定义良好的接口和规范以松耦合方式整合在一起，即将多个现有的应用软件通过网络将其整合成一个新系统。

SOA 是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。

对于面向同步和异步应用的、基于请求/响应模式的分布式计算来说，SOA 是一场革命。一个应用程序的业务逻辑或某些单独的功能被模块化并作为服务呈现给消费者或客户端。这些服务的关键是他们的松耦合特性。例如，服务的接口和实现相独立。应用开发人员或者系统集成者可以通过组合一个或多个服务来构建应用，而无须理解服务的底层实现。一个服务可以用.NET 或 J2EE 等不同技术来实现，而使用该服务的应用程序可以在不同的平台之上，也可以使用不同的语言。

2. SOA 应用框架

SOA 强调以服务为中心构建企业 IT 系统，在 SOA 的设计中自上而下有一条线，如图 2.1 所示。

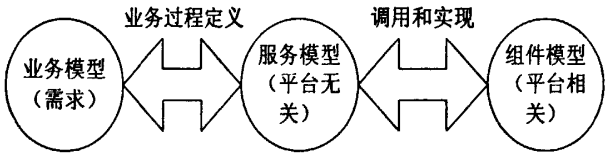

图 2.1 基于 SOA 的开发模型图

由图 2.1 可以知，基于 SOA 的开发首先由业务建模开始，通过定义业务过程，得到平台无关的服务模型。最后，通过设计组件群，得到平台相关的组件模型。

SOA 方法将功能方面涉及的对象、数据、组件、业务流程、界面等从服务提供者和消费者角度进行层次化。同时，将安全架构、数据架构、集成架构、服务质量管理等，应用公用的设施提取出来形成不同的层次，为所有的服务所共有。一个 SOA 应用系统的大体框架结构[16]大体上可以分为七个部分，如图 2.2 所示。

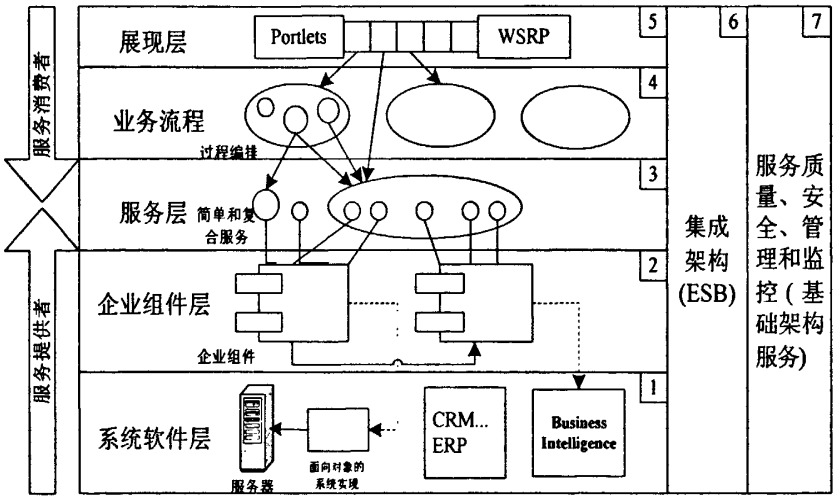

图 2.2 SOA 应用系统框架图

1) 系统软件层：图 2.2 中的 1 区，本层包含现有的自定义构建的应用程序，也叫做遗留系统，包含现有的 CRM 和 ERP，商业智能(Business Intelligence)，以及较旧的基于对象的系统实现。SOA 的复合层架构可以利用现有的系统并且用基于服务的集成技术来集成它们，是一个集成的平台。

2) 企业组件层：图 2.2 中的 2 区，本层由那些负责实现功能的组件和保证 QoS(Quality of Service)的服务组件组成。这些组件是平台相关的。因为到了这一层，许多底层软硬件平台的特性已经不再透明了。这些功能性的组件，是企业和业务单元范围内一种受管理和控制的企业资产。而那些用于保证 QoS 的服务组件，它们用于通过架构设计的最佳实践来确保 QoS。通常它们使用基于容器的技术，比如实现组件、负载均衡、高可用性和工作量管理的应用服务器。

3) 服务层：图 2.2 中的 3 区，整个 SOA 的核心层，它承上启下，对上响应业务模型，对下调用相关组件群完成业务需求，形成“业务驱动服务、服务驱动技术”的 SOA 事务处理格局。服务发现中被确定的公开服务处在这一层。它们可以被发现或者直接静态绑定，接下来被调用，或者被编排到合成服务中。

4) 业务流程层：图 2.2 中的 4 区，第三层中公开的服务合成和编排在这一层中被定义。通过配合、编排，服务被绑定成一个流程，从而作为单独的应用程序而被共同作用。这些应用程序支持特殊的用例和业务流程。

5) 展现层：图 2.2 中 5 区，通过 Portal 等技术建立展现平台，方便用户在这个界面上提出服务请求。

6) 集成架构（企业服务总线）：图 2.2 中 6 区，这一层使服务可以集成，通过引入一系列可靠的性能集合，比如智能路由、协议中介和其它转化机制，经常被描述为 ESB。

7) 基础架构：图 2.2 中 7 区，这一层提供了监视、管理和维持诸如安全、性能和可用性等 QoS 的能力。

3. SOA 优势

SOA 技术相对于传统的软件开发技术有以下的优势[17]：

1) 集成现有系统。

面向服务的体系结构可以基于现有的系统投资来发展，而不需要彻底重新创建系统。通过使用适当的 SOA 框架并使其用于整个企业，可以将业务服务构造成现有组件的集合。使用这种新的服务只需要知道它的接口和名称。服务的内部细节以及在组成服务的组件之间传送的数据的复杂性都对外界隐藏了。这种组件的匿名性使组织能够利用现有的投资，从而可以通过合并构建在不同的机器上、运

行在不同的操作系统中、用不同的编程语言开发的组件来创建服务。遗留系统可以通过 Web 服务接口来封装和访问。

2) 服务设计松散耦合。

服务是位置透明的，不必与特定的系统和特定的网络相连接。松散耦合消除了对系统两端进行紧密控制的需要。就系统的性能、可伸缩性以及高可用性而言，每个系统都可以实现独立管理。松散耦合给服务提供者和服务请求者提供了独立性，但要求基于标准的接口和中间件来积极地管理和代理终端系统之间的请求。

3) 统一业务架构，增强可扩展性。

在所有不同的企业应用程序之间，基础架构的开发和部署将变得更加一致。现有的组件、新开发的组件和从厂商购买的组件可以合并在一个定义良好的 SOA 框架内。这样的组件集合将被作为服务部署在现有的基础构架中，从而使得可以更多地将基础架构作为一种商品化元素来加以考虑，增强了可扩展性。又由于面向服务的敏捷设计，在应对业务变更时，表现出更强的“容变性”。

4) 加快开发速度，减少开发成本。

组织的 Web 服务库将成为采用 SOA 框架的组织的核心资产。使用这些 Web 服务库来构建和部署服务将显著地加快产品的上市速度，因为对现有服务和组件的新的创造性重用缩短了设计、开发、测试和部署产品的时间。SOA 减少了开发成本，提高了开发人员的工作效率。研究表明，一般系统的接口的开发费用占到整个开发费用的 33%，最高的竟达到了 70%。在 SOA 中，接口的重用会节省费用 60%。而且节省的费用不是一次性的，而是每年。随着业务需求的发展和新的需求的引入，通过采用 SOA 框架和服务库，为现有的和新的应用程序增强和创建新的服务的成本大大地减少了。

5) 持续改进业务过程，降低激变风险。

SOA 允许清晰地表示流程流，这些流程流通过在特定业务服务中使用的组件的顺序来标识。这给商业用户提供了监视业务操作的理想环境。业务建模反映在业务服务中。流程操纵是以一定的模式重组部件（构成业务服务的组件）来实现的。这将进一步允许更改流程流，而同时监视产生的结果，因此促进了持续改进。重用现有的组件降低了在增强或创建新的业务服务过程中带来的风险，也减少了维护和管理支持服务基础架构的风险。

2.1.3 网格与 SOA 关系概述

网格正在逐渐向 Web 服务架构发展，首先是 Globus 采用开放网格标准基础设施(OGSI)，然后是发布 Globus Toolkit 4.0(GT4)。SOA 和网格技术正基于诸如 WSRF 以及其他的一些解决方案，朝 Web Standards Interoperability 技术方向发展。网格

系统早期的设计实现并未采用 Web 服务技术，没有体现出 SOA 所具有的可扩展和松耦合性。随着 Web 服务技术的发展以及相关标准规范的日益成熟，SOA 和网格技术同其他 Web 服务的解决方案加速了两者之间的融合。与最初的网格系统设计相比，面向服务的网格系统更好地实现了网格计算、系统管理和 Web 服务的一致性与协调性。

SOA 和网格都可以为对方提供很多东西。从 SOA 的观点得知，网格为信息和资源的分布提供了解决方案，这是 SOA 模型的一个关键特性。从网格的观点得知，SOA 为调整网格解决方案的架构以及促进其透明性和更好地支持广泛的平台和环境，提供了一些可选的而又非常灵活的方法。

网格是一种资源共享、协同工作的分布式系统，为 SOA 架构的实施提供了资源整合和共享的平台。主要体现在以下几个方面[18]：

1. 网格被视为一个由各种计算资源组成的统一环境，由网格作业管理系统将网格整合成一个完整而协调的透明计算整体。
2. 网格是一个虚拟的应用服务器，也是一个应用实现和数据处理的理想平台。SOA 封装后的服务在网格中部署和调用执行，而服务调用作为网格程序在平台上运行。

2.2 网格作业管理技术概述

网格作业管理作为网格系统软件的重要组成部分，直接关系到网格资源性能的发挥和使用率的提高，也是当前网格研究的热点技术之一。

2.2.1 网格作业管理的特点

网格作业管理是根据作业的资源需求和网格资源的状态，对作业所要求的资源进行选择和分配，并进行任务的调度和作业执行的控制，其目标是实现对网格资源的优化使用，对网格用户提供更好的服务质量。

作业方式是使用网格资源的一种形式，它根据用户确定的流程，为用户提供使用资源的功能。网格作业一般都是在远程节点上运行，作业提交者对远程设备的控制能力是非常有限的，为了有效管理作业的运行，就需要网格作业管理机制来管理整个网格作业的运行过程。作业是用户代码、数据及其相应资源描述信息的集合。作业管理需要信息服务、资源管理、数据管理、安全通信的支持，它是网格中不可或缺的功能，尤其是在计算网格中，它是保证网格用户合理有序地使用网格环境下的计算资源的基础，应该做到每个用户提交的作业都能在合适的资源上执行。网格作业是用户要求网格进行计算的一个计算任务。网格上的计算资源是一种需要用户提供代码使用的资源，通过执行用户代码，处理用户数据给用

户提供计算周期。大多数情况下，用户都是以提交作业的方式使用资源[19]。

相比传统的操作系统作业管理或机群作业管理，由于网格资源具有大规模分布性、类型异构多样性、动态变化性等特点使得网格作业管理更加复杂和难于实现。一个典型的网格环境的系统结构如图 2.3 所示。

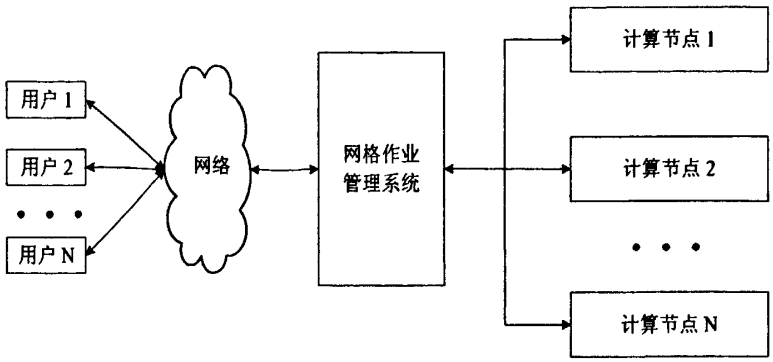

图2.3 典型的网格模型结构图

这一种通用网格系统结构，它采用典型的 C/S 模型[20]，大多数网格系统都采用这种原理进行工作。网格系统架构使用 C/S 模型构造的应用，由客户和计算节点协作完成，这种分工协作可以充分地发挥计算节点的高性能和客户端的灵活、易用性，使两者的效用都达到最高，使网络带宽资源的利用更加合理。网格作业管理系统的职责就是运行网格，它负责协调客户和计算节点：分发作业并与各个计算节点进行工作。作业管理是作业提交者和计算资源之间的中介，从资源的角度要求作业管理做到负载均衡，从用户的角度要求最大限度地保护用户的利益。以下针对这种模型分析作业管理系统的主要特征[21]：

1. 分布式

作业管理系统发展的初期，主旨是使大型机的集中式计算资源得到更充分地利用，整个系统是集中式结构。当高性能的工作站和高速网络被普遍使用后，大型机上的计算任务被分散在多台工作站上执行，人们使用自己桌面上的台式机完成了更多的任务。但经调查发现，工作站的资源利用率却很低（据美国 Los Alamos 国家实验室的一项调查表明，平均利用率不到总资源的 10%），通常一台工作站任务繁重，而另一台工作站却处于闲置状态，这对任何一个单位而言，硬件的投入都没有收到应有的效益回报。出于这种需要，分布式网格环境更需要作业管理系统担负起资源管理与作业调度的职能。实际上，正是网络计算成为主流的应用模式后，才产生了对作业管理系统的迫切需求。从此时起，作业管理系统以网络上的资源管理、作业调度为中心，采用 IPC 或 RPC 机制实现运行在多个机器上的作业管理系统之间相互通讯。

2. 异构性

以网络为核心的系统有一个主要特点，那就是系统的异构性，要充分使用这样的系统，应用软件必须具有各个系统间的互操作能力。服务器与客户机可以由

不同体系的计算机构成，根据处理能力的不同，服务器一般采用大型机、并行机或小型机、多处理器硬件平台，运行 Unix 等大型操作系统；客户机采用工作站、高性能 PC 或普通的台式机，运行 Unix 或 Windows NT 等操作系统。服务器与客户间采用成熟的 IPC 机制实现相互通讯。

3. 开放性

以一个开放的体系结构建立的系统，运行过程中可以灵活地增加、减少或变更各组成部分，包括机器硬件和软件模块。作业管理系统所运行的网络环境，会发生各种资源变更情况，如网络规模的扩大，需要作业管理系统管理更多的机器节点，操作系统的升级需要作业管理系统支持新的作业运行方式等，因此作业管理系统的体系结构必须是开放的。

4. 集中式管理

可以通过一点来了解整个网络的情况，这使大型网络的集中化管理得以实现。作业管理系统的运行环境是很复杂的，可能是一个小型的局域网，也可能是一个大型的广域网，要有效地维护整个系统，必须提供集中化管理机制，使管理员从网络中任意一台工作站上就可完成对整个系统的全面控制。

2.2.2 网格作业管理基本功能

网格作业管理不仅为网格用户作业在远程结点上运行提供透明支持，而且也是网格用户有效，合理的使用网格远程资源的工具。

网格作业管理的目的是为用户提供透明访问网格资源的接口，用户通过统一的接口使用远端的资源，把具体资源的技术细节隐藏起来。用户不需要考虑目标结点本地在调度策略，数据表示形式，文件存储格式等方面规定。具体分析，网格下作业管理主要有以下基本功能[22]：

1. 管理作业的整个生命周期，负责作业从用户提交开始直到给用户返回执行结果的全部过程。
2. 对作业进行有效调度，为作业查找合适的资源，匹配作业需求。根据用户作业的需求，从网格中当前可用的资源中选择合适的资源，并把所选择的资源分配给用户使用。
3. 管理作业的输入/输出。网格作业的输入/输出一般都在远程结点之间进行，但这个特点并不一定要在作业代码中体现出来，输入可以是读键盘，输出可能是写屏幕，网格作业管理系统要能够从正确的位置读到数据，能向正确的位置写数据。
4. 负责作业运行的监控。在作业运行过程中，管理员和最终用户可查询作业执行的状态信息，包括作业提交时间，运行状态，以及作业实例的执行历

史纪录等。对每一个已被调度的作业通过本地代理服务器的作业管理软件获取当前作业状态信息，并与信息服务中心保持通信从而不断更新作业状态信息。并允许管理员和作业的提交者通过管理接口控制作业的执行，例如可以暂停、继续、中止作业。而管理员还可以迁移作业，把作业从一个资源迁移到另一个资源接着运行，实现网格总体资源的负载平衡。由于不能准确预测作业运行的实际情况，在网格中也会出现负载不平衡并需要作业迁移的情况。而网格资源的动态进出也需要进行作业的迁移。

2.3 本章小结

本章主要介绍了网格作业管理的相关技术，首先分析了网格与SOA的基本概念，然后论述了网格和SOA的关系，最后分析了网格作业管理系统的特点和基本功能。

第三章 企业计算网格作业管理系统体系架构分析与设计

在第二章的分析中，分析了网格作业管理技术以及网格和SOA技术的可融合性。在此基础上，本章完成了基于SOA的企业计算网格作业管理系统体系架构的分析与设计。

3.1 系统需求分析

3.1.1 金融领域需求分析

在如今竞争如此激烈的市场环境下，金融企业要想生存和发展，必须能够应对客户和市场需求的变化，这就意味着它们必须构建足够强壮、灵活、可伸缩的基础架构以应对迅速增加的网络流量和使用量。金融企业需要快速部署新的业务处理系统，使客户能够快速访问他们所需的资源，同时维护系统的安全性。而且，这些资源可以共享并协同工作。企业计算网格就是能满足这些需求的基础设施，因而引起了全球的关注，并得以快速发展。

针对这一背景，金融领域对网格作业管理系统具体的需求分析如下[23]：

1. 提供资源共享的平台。支持异构平台，支持多种流行操作系统（Linux 和 Windows），能够提高整个企业的计算力和资源的利用率，有效地抑制对资源的无穷尽的投入，显著降低企业成本。
2. 高性能计算能力。金融领域存在大量的深度计算，如定价、风险估值、随机建模、蒙特卡洛模拟都需要实时执行。寻求最小风险下的最大盈利，始终是金融企业和商业银行追求的目标，而风险需要跨企业、产品组合和业务线进行计算，这就需要具备强大的计算能力，与多个应用共同工作，并实现更短的应用运行时间，这些都离不开高性能的计算环境。
3. 良好的可伸缩性。可伸缩性是指可以根据用户实际的需求进行相应的变化，如用户的变化、业务的增加，且具备持续良好的工作的能力。理论上，可伸缩性可以称为软件系统的工作负载能力和硬件资源的投入比例。在一个具有良好伸缩性的系统中，不仅能够支持管理和调度更大规模的主机数量，而且部署后的网格应用的性能应达到充分利用网格计算资源的目的。可伸缩性要求如表3.1所示。

表 3.1 可伸缩性要求

支持搭建最大主机数量	5000台主机
每个作业调度器支持的最大并发客户端数量	1000
每个作业调度器支持的最大计算节点数量	4000
Session最大数目	（每个作业调度器里最多支持1000个session）*（100个作业调度器）
每个作业调度能够支持的最大task数量	每个session最大支持1M task

4. 具有较高的可靠性，并具备具有一定的故障容错能力和灾难恢复能力。金融业务的特点使得高可靠性成为一个非常重要的因素，在压力下至少保证2个星期以上不会引起系统失效。
5. 能够降低IT操作成本和开发成本，支持企业IT的复杂管理，并能够更易于集成和管理复杂性。能够将基础设施和实现发生的改变所带来的影响降到最低限度。当更多的企业一起协作提供价值链时，这会变得更加重要。
6. 提供灵活的作业调度策略供用户配置。针对不同类型的作业，既要考虑系统整体较高利用率保证公平共享，又要考虑重要作业的实时响应，而且这些都可以由用户自主选择配置。
7. 提供更快的响应和上市速度。能够通过利用现有的组件，减少完成软件开发生命周期（包括收集需求、进行设计、开发和测试）所需的时间。使得可以快速地开发新的业务服务，并允许组织迅速地对改变做出响应和减少上市准备时间。
8. 能够延长应用程序运行时间，提高投资回报率，减少成本和增加重用。通过以松散耦合的方式公开业务，企业可以根据业务要求更轻松地使用和组合组件。这意味资源副本的减少、以及重用和降低成本的可能性的增加。

3.1.2 金融作业分析

网格技术在金融业应用的种类很多，可以从输入数据和计算量上两个方面分类[24]。金融应用程序的输入数据包括原始的和调整后的市场数据、场景数据、条件数据和交易数据等，输出数据则包括市场价值、市场风险值和信用风险值等。将金融行业计算问题按计算运行时间（代表计算量）和计算数据量大小两个维度进行分类，如图3.1所示。

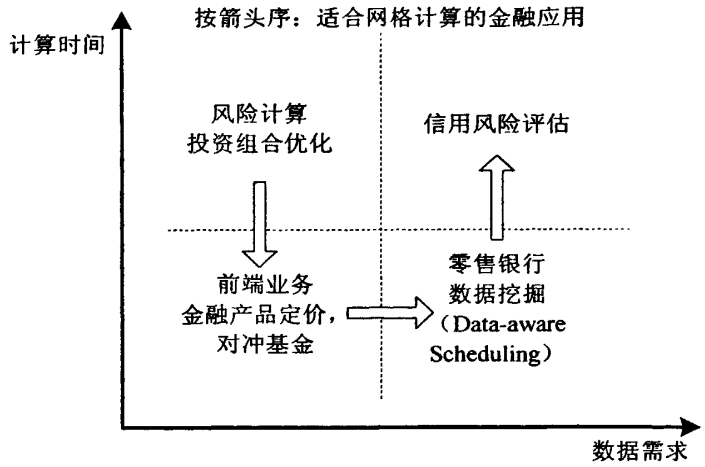

图 3.1 金融行业计算问题的分类图

每个金融应用的作业可分类为少量数据短时计算（如金融产品定价和对冲基金的净值计算）、少量数据长时计算（如市场风险计算和投资组合优化计算）、大量数据短时计算（如业务数据挖掘）和大量数据长时计算（如信用风险计算）四类。目前在金融分析领域最适合网格计算而且网格计算已经得到应用的是风险管理即少量数据长时计算这一类分布式计算，少量数据短时计算次之。对于数据量大的单个计算任务如零售银行数据挖掘和信用风险评估，网格系统需要提供基于数据的任务调度以尽量减少数据传输时间对分布式计算性能的影响。

根据对金融领域计算问题的分析，网格作业管理系统的设计目标主要针对的是少量数据计算这种类型。因此，对系统的网格作业进行了如下定义：将一组作业抽象成一个会话(Session)，其中每一个作业就是一个任务(Task)，以 Session 为单位进行作业调度。以下详细分析。

1. Task

Task 是独立的能够并行计算的基本数据单元，包括输入数据和输出数据。

2. Session

Session 是一组使用相同服务并且之间能共享数据的 Task 集合。它的提出主要基于以下两种目标：

1) 提高 Task 响应时间。

通过前面的分析可知，针对小数据量，需要及时响应的作业，如果将作业调度的粒度放在 Task 一级，那么系统需要为每一个提交的 Task 进行调度的时间为 t1，然后再进行 Task 的分发处理时间为 t2，这样一个 Task 的相应时间 t 就是：t = t1 + t2。而如果将作业调度的粒度放在 Session 一级，那么网格作业系统已经为该作业分配了计算资源，因此当提交一个 Task 时，系统就可以直接进行 Task 的分发处理，这样一个 Task 的相应时间 t 就是：t = t2。所以如果 Task 为单位进行作业调度就会导致响应时间减慢，尤其是对于执行时间很短的 Task 影响更大。

2) 避免数据的冗余传送。

对于相关的 Task 进行抽象，如果 Task 之间具有一些公共数据，那么可以作为 Session 的公共数据一次性传到服务端供各 Task 共享。

因此，这种通过 Session 以一组作业为单位分配资源的作业方式，适合于对实时性有要求的金融领域。

3.2 系统体系架构分析

根据上述需求分析，要满足金融领域的需求，就要改进网格作业管理系统体系架构，并结合 SOA 来完成网格作业系统架构的设计。本章首先分析了网格作业管理系统的功能结构，然后再结合 SOA 进行架构分析，具体设计在 3.3 进行。

3.2.1 网格作业管理模型

在网格这种分布式异构环境中，通常存在多种硬件系统平台（如PC，工作站，小型机等），在这些硬件平台上又存在各种各样的系统软件（如不同的操作系统、数据库、语言编译器等），以及多种风格各异的用户界面，这些硬件系统平台还可能采用不同的网络协议和网络体系结构连接。如何把这些系统集成起来并开发新的应用是一个非常现实而困难的问题。目前提出的多数网格作业管理模型对资源调度和负载均衡的功能界限划分不清，概念上的混用和不一致使得功能模型难以独立，更加剧了问题的复杂性。

从最终结果看，网格作业管理就是将作业映射到计算资源上的过程。但是这个过程实际上分为两步[25]：第一步，为网格应用程序分配资源；第二步，将网格应用程序申请到的资源在各个作业之间进行再分配。因此，需要把资源管理层单独划分出来。网格作业管理系统层次模型如图3.2所示。

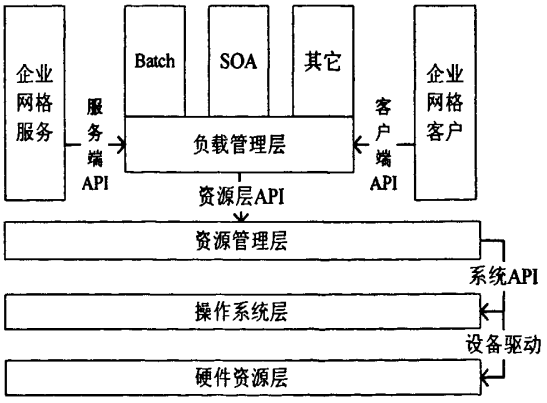

图 3.2 网格作业管理系统分层模型图

网格作业管理就被划分为资源管理和负载管理两块：资源管理主要专注于资源方面的管理，包括资源的描述、发现、分配、释放、执行单元的启动和停止等；

而负载管理则主要是在资源管理提供了资源的基础上进行资源的再分配以达到更细粒度的资源调度。

本地操作系统通过不同的设备驱动对各种硬件资源进行操作，而资源管理层通过不同的本地操作系统对异构计算资源进行管理，即该层控制管理网络节点资源，提供网格分散资源的单一视图表示，可以继续完善并发展为网格操作系统，而本地操作系统则相当于网格操作系统的设备驱动。

资源管理层提供了一个单一的管理环境，可在地理位置分散的站点之间，为所有的关键任务应用、服务及工作任务集中分配共享资源。这一层的资源分配粒度是消费者。消费者是一种抽象的资源使用者的表示，资源的消费实体存在于负载管理层。描述一个消费者需要提供可以使用的资源组、在各组内的资源策略以及和其他消费者之间的策略配置。根据这些消费策略，资源管理层就会根据不同的策略在负载管理层对资源的需求和可用的资源供给之间做出匹配。当一个特定的网格应用管理者或者消费者确定他们所需要的资源后，资源管理层就会负责提供相应的资源。资源管理层确定每个消费者有资格获得的资源的数目，然后对消费者的优先级进行评估，最后分配给消费者所需数目的资源(例如，计算资源个数，虚拟机或者是实际的物理资源)。

资源管理层扮演了资源提供者的角色，而负载管理层是资源消费者。资源管理层的实现只有一个，负载管理层可以有多个实现，并同时向资源管理层请求计算资源。

资源管理层之上可以是任何面向服务的应用系统、批处理系统或是任何需要使用分布式资源来提供功能的应用系统。

3.2.2 面向服务架构分析

系统的设计定位是面向金融行业的网格软件，实现以服务形式访问网格计算资源的接口，它的设计旨在满足这种针对小数据量、需要及时响应的特定应用领域的作业需求，实时解决关键任务定价和风险问题，从而实现卓越的性能和竞争优势。

根据前面的分析，网格作业管理被划分为资源管理和负载管理两个独立的模块。资源管理相当于网格操作系统，屏蔽了底层的异构系统，专注于资源的管理，是资源的提供者。而负载管理则相当于网格中间件，位于平台（资源管理层）和应用之间，主要是为处于它上层的应用软件提供运行与开发的环境，以降低应用开发的复杂程度，是资源的消费者。根据金融领域的需要，系统的关键在于对负载管理层的设计。因此，负载管理层应采用面向服务的应用系统来作为资源抽象层的消费者去阐述计算资源在负载管理层的再分配。以下结合 SOA 对负载管理层

进行分析。

SOA 界定了两个计算实体如何交互的方式（交互方式是指使一个计算实体代表另一个计算实体来执行一个单元的工作，该单元的工作是指作为一种服务）。而服务的交互是使用描述语言来定义，并且每个交互都是独立的。SOA 基于 4 个关键抽象：服务、应用程序前端、服务注册中心和服务交互[26]，以下从这四个方面论述系统 SOA 的设计。

1. 服务

设计 SOA 架构要先明晰服务的概念。服务是自包含的功能单位，负责转换，存储或检索数据。服务之间是松散耦合。服务可以通过明确界定的公共接入点访问，而所有的访问都是由服务契约来控制的（服务契约是由严格界定的一套消息，消息内容和适用的服务策略组成）。服务就是服务提供者为服务使用者提供的作业计算功能。

2. 应用程序前端

应用程序前端是业务流程的所有者，负责发起和控制系统的所有业务活动。应用程序前端有多种类型，不一定非与最终用户直接交互。应用程序前端可以将业务流程的大多数职责委托给一个或多个服务，但应用程序前端负责发起业务流程并接收结果。

3. 服务注册中心

服务注册中心担当可用服务清单的角色，提供调用服务的地址。通过服务注册中心，调用服务者可以发现服务（服务端点地址），获得使用服务的所有信息。用户开始使用服务时，将在此服务上创建一个依赖关系。服务注册中心能管理使用者和服务间的依赖关系。

4. 服务交互

服务交互，指的是作为服务使用者和服务提供者之间的互动通信并交换信息的过程。它负责将 SOA 的所有参与者（服务和应用程序前端）相互连接在一起。图 3.3 描述了一次服务交互的过程。

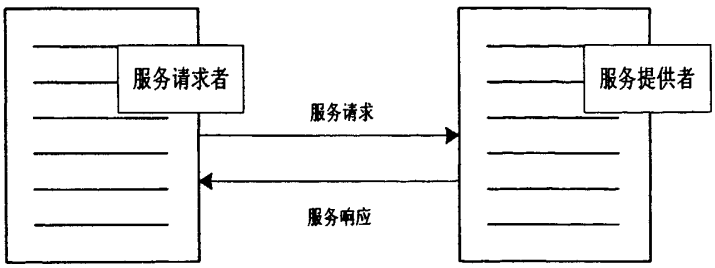

图 3.3 服务交互图

一个服务请求者（在图 3.3 左边）向一个服务提供者（在图 3.3 右边）发送服

务请求消息；服务提供者返回一个服务响应的消息给服务消费者。服务请求和其后的服务响应的交互是以一种被服务消费者和服务提供者都共同理解的方式进行。两者的角色并不固定，服务提供者也可以成为服务消费者。因此，负载管理层需要设计这样一种服务流程：

1) 客户端连接到系统。
2) 客户端提交一个或多个服务请求（独立自治的计算任务）。
3) 客户端接收每个服务请求的响应。
4) 客户端断开与系统的连接。

最为重要的是，这种服务流程没有次序和状态的概念。每个请求都独立于前一个请求，避免了服务请求者依赖于服务提供者的状态。

3.3 系统体系架构设计

根据上一节分析所得结论，可将网格作业管理系统划分为两个独立的功能层次：资源管理层和负载管理层，对负载管理层采用 SOA 架构设计。系统总体结构如图 3.4 所示。

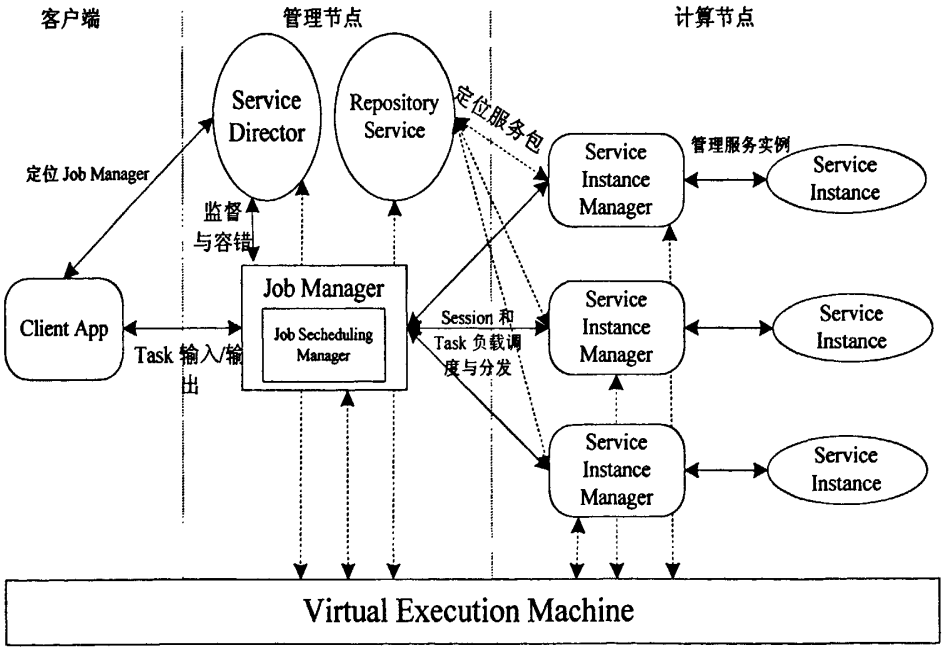

图 3.4 企业计算网格作业管理系统体系架构图

该架构主要分为两层：资源管理层和负载管理层。资源管理层通过 VEM(Virtual Execution Machine)实现，负载管理层通过前面的 SOA 分析主要包括以下几部分：服务管理(SD, Service Director)，服务包仓库(RS, Repository Service)，作业管理(JM, Job Manager)，服务实例管理(SIM, Service instance manager)和服务实例(SI, Service instance)，下面具体分析：

1. VEM (Virtual Execution Machine)

VEM 主要负责管理整个网格的计算资源，包括资源的供给和分配。其主要功能有：

1) 将虚拟化及关键任务应用集成至单一高效的系统之中。
2) 以单一网络环境覆盖地理位置分散的不同站点。
3) 根据业务策略对所有企业资源上的工作任务进行优先级划分。
4) 实时响应客户及供应链的业务需求变化。
5) 以模块化应用集成方式，稳步增长与扩充。
6) SD 和 RS 是作为 VEM 的服务被 VEM 启动，由 VEM 负责其错误恢复。当 SD 和 RS 发生错误时，通常不会对正在运行和未运行的作业产生影响。

VEM 在系统中相当于虚拟的操作系统，它提供以下基本接口供它之上的各种网格应用系统使用：

1) 注册、注销消费者。
2) 申请资源。
3) 释放资源。
4) 启动、停止执行单元。

2. SD(Service Director)

一个网格里只有一个 SD，它是作为 VEM 的一个服务被 VEM 启动的。它具有服务注册中心的职责，而且还提供了作业管理系统唯一的的访问点，主要完成以下功能：

1) 接受网格应用程序配置文件（见 3.4 节）的注册并管理这些配置文件。客户在使用任何一个网格应用程序的时候，都需要首先在 SD 注册网格应用程序的配置文档。
2) 接受和验证客户端的连接：当用户发送服务请求时，SD 查找指定的 JM 的 URL(Uniform Resource Locator)并通知客户。它提供给客户端访问系统的安全令牌，负责将客户程序连接到 JM 并开始计算 Task 的发送。
3) 负责向 VEM 请求资源启动 JM，并管理 JM 的生命周期。
4) 提供实现管理操作的接口，负责系统的多个服务当前的状态管理，提供系统的服务管理接口。

SD 的结构图如 3.5 所示。

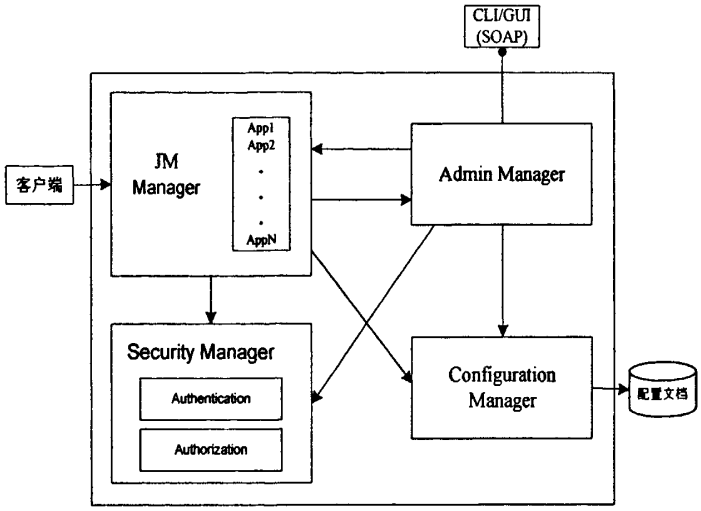

图 3.5 SD 结构图

JM 管理器(JM Manger)：负责接收网格应用程序配置文档的注册，并建立客户端与 JM 的连接，同时管理 JM 的生命周期。

控制管理(Admin Manager)：负责执行客户通过 CLI/GUI 发送的命令请求。

配置管理(Configuration Manager)：负责网格应用程序配置文件的管理。

安全管理(Security Manager)：负责系统安全管理。

3. RS(Repository Service)

RS 是作为 VEM 的一个服务被 VEM 启动的。它主要负责在注册服务时部署服务包，并管理已部署的服务包。开发者将一个服务打包上传到 RS 后，当 SIM 需要这个服务包的时候，RS 负责将其传送到管理节点的缓存中，这样服务的具体实现、位置和传输协议对调用者来说都是透明的。

4. JM(Job Manager)

在一个网格里有一个或多个 JM。它主要完成以下功能：

1) 在客户端和计算节点之间传递消息。
2) 向 VEM 申请资源启动 SIM，负责管理 SIM 的整个生命周期。当 SIM 发生错误时，JM 要负责将发送到发生错误或丢失的计算节点的作业重新分配到其它的计算节点。
3) 作业调度。JM 针对每个网格应用程序以 Session 为单位进行资源调度和负载管理。它的资源分配的粒度是 Session。
4) 提供管理和控制 Session 和 Task 操作。
5) 数据管理：记录 Session 和 Task 的状态和进展。提供作业容错功能，使得 JM 在非正常退出并重新启动后能够恢复作业。

JM 的结构图如 3.6 所示。

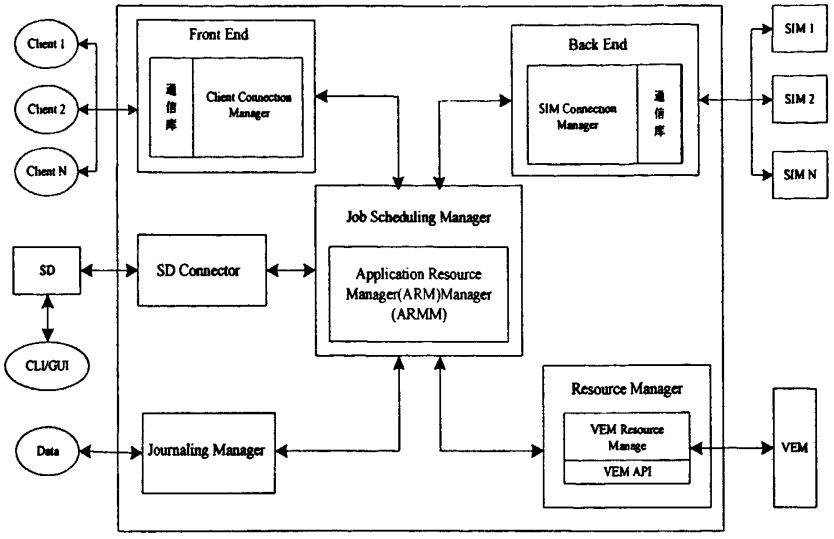

图 3.6 JM 结构图

前端(Front-End)：主要负责管理客户端的连接。

后端(Back-End)：主要负责与 SIM 的交互。

作业调度管理(Job Scheduling Manager)：作业调度管理是作业管理的核心，负责作业的调度管理，后面将详细讲述它的设计与实现。

资源管理(Resource Manager)：主要负责从 VEM 获得资源。它是位于资源调度管理和负载管理两个模块之间的一个轻量级资源适配器，按照消费者和网格应用程序之间的映射关系进行角色转换：在网格服务申请、释放资源或修改资源请求数时，将网格服务转换为 VEM 能理解的消费者；另一方面，在消费者作资源收回操作或资源层作其他回调通知操作时，将消费者转换为负载层的网格服务角色。

记录管理(Journaling Manager)：主要负责作业的状态记录以及数据的备份。记录管理负责将可恢复的作业数据保存至磁盘，这样，作业的数据不仅仅存在于 JM 的进程空间，在 JM 非正常退出后，系统重新载入 JM 模块，并在初始化过程中恢复作业的数据，使作业能够按照原来的状态继续运行。

5. SIM(Service Instance Manager)

计算节点负责处理服务请求。在一个计算核（虚拟化的一个计算单元）上有且只有一个 SIM。它主要完成以下功能：

1) 负责调用不同类型的服务容器去装载不同类型的服务，过程如图 3.7 所示。

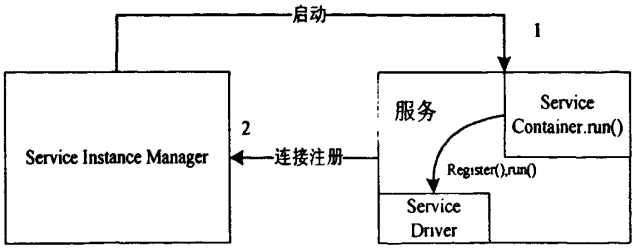

图 3.7 SIM 启动服务图

2) 负责服务实例的运行管理，包括服务进程的启动、停止、监视其状态等。SIM 支持多服务实例管理，但同一时间只有一个服务实例运行。

3) 在 JM 和服务之间传送信息。

6. SI(Service instance)

一个运行在服务容器中的服务执行实例就是一个 Service Instance。一个 Service Instance 可以为同一个网格应用程序的多个 Session 服务，也可以为一个 Session 指定一组 Service Instance 进行服务。服务容器装载服务，负责管理服务的整个生命周期。

3.4 系统服务设计

在 3.2.2 节的分析中，论述了作业管理系统需要向上层应用提供一个服务流程。在完成系统架构设计的基础上，以下对服务应用流程进行设计。服务应用流程就是一个面向服务的网格应用程序的执行过程，其流程如图 3.8 所示。

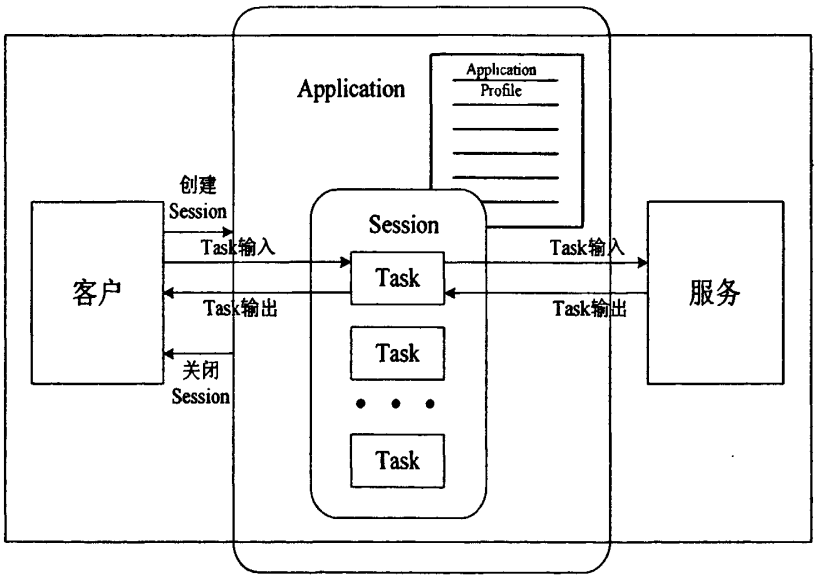

图 3.8 面向服务的网格应用程序流程图

服务应用流程包括客户应用，服务和网格应用程序配置文档。客户应用首先需要注册网格应用程序配置文档（配置文档描述了服务的各方面属性），接着客户应用才能发起一个或多个服务请求，即发送计算 Task，它负责业务逻辑中的串行处理。然后网格作业管理系统将多个逻辑相同但是数据不同的计算任务分派到多个计算节点上，按照服务的逻辑进行分布式计算。服务负责业务逻辑中的可并行处理部分，在完成对 Task 处理后，将结果再通过网格作业管理系统返回给客户应用。在整个交互过程中，对每个 Task 的处理都独立的，通过网格作业管理系统屏蔽了服务交互的底层处理细节。

1. 客户应用

客户应用是服务使用的发起者，为了充分利用现有的计算能力，同时能够方便客户使用系统，根据服务流程设计了一套客户应用程序 API 提供给客户。客户端程序结构如图 3.9 所示。

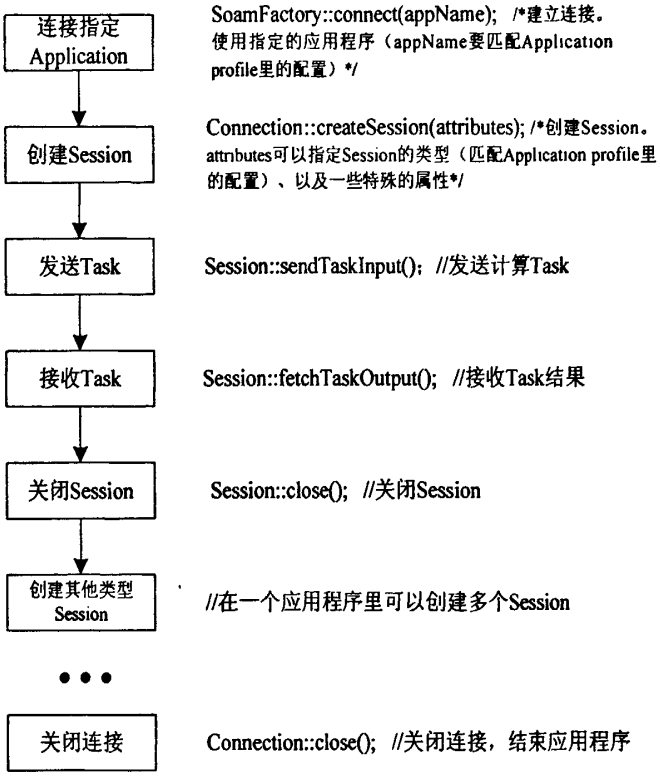

图 3.9 客户端程序结构图

客户应用程序可以通过这些数量不多且易于理解的 API 完成了对复杂客户应用的设计。

2. 服务

服务是一个抽象的概念，从以下两个角度分析服务的确切含义：

1) 客户应用通过使用某种服务来完成某项工作，客户并不关心具体分配多少服务实例。

2) 服务是开发和部署在网格上供客户使用的，是能够重用的。

结合上述分析，服务是企业计算业务逻辑的实现。原来需要重复执行的业务计算可以封装成网格服务部署、注册到负载管理层中，由服务名来标识，以此方式达到分布式复用。服务是运行在服务容器内的，它的生命周期如图 3.10 所示。

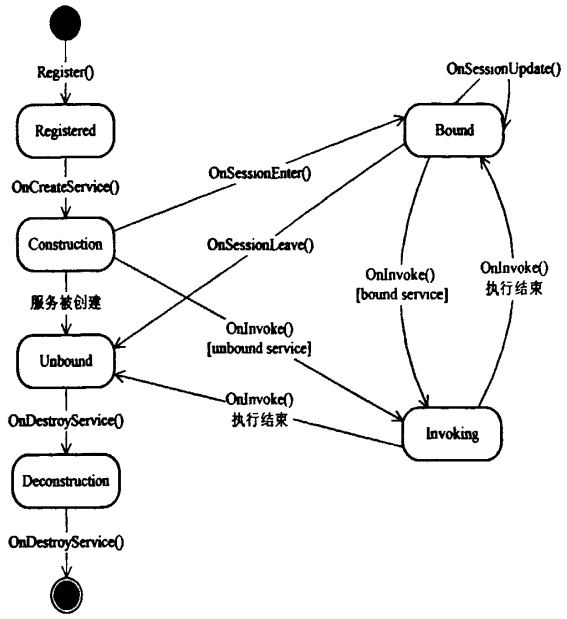

图 3.10 服务的生命周期图

在服务的生命周期中共有六个状态：注册(Registerd)、创建(Construction)、绑定(Bound)、解除绑定(Unbound)、计算(Invoking)和销毁(Deconstruction)。由服务容器代理服务消息并监控服务活动状态。服务容器驱动了服务生命周期的某一阶段所要求的操作。例如，onCreateService 方法是创建服务函数；onSessionEnter 方法是服务的 Session 初始化函数，在这个函数里面将 Session 和服务进行了绑定；onInvoke 方法是计算 Task 执行计算逻辑的函数。服务状态和任务的状态有对应关系。如果是预启动的服务，启动后服务处于等待计算任务的创建状态。如果服务执行的 Task 发生错误，服务根据配置的重试次数重新进入计算状态。

另外，服务和 Session 之间是有绑定关系的。在网格应用程序的配置文档里将一个 Session 绑定到某一服务，这样在系统创建该 Session 的时候，系统会启动指定的那个服务，并将 Session 与该服务建立连接，从而使 Session 的 Task 能够使用该服务。

在一个网格应用程序中可以创建多种类型的 Session，利用服务和 Session 之间的绑定关系，可以实现多个服务的组合：在一个网格应用程序中使用多个 Session，然后让每个 Session 又绑定到不同的服务上。这样在一个网格应用程序里就可以通过多服务的组合并发完成更复杂计算任务。

3. 网格应用程序配置文档(Application Profile)

网格应用程序配置文档描述了网格应用程序的服务契约，定义了网格应用程序的特征以及系统的行为。配置文档如图 3.11 所示。

```
<Consumer applicationName="SampleApp" consumerId="/Sample/SOASamples"
          taskHighWaterMark="1.0" taskLowWaterMark="1.0"
          preStartApplication="false" numOfPreloadedServices="1"/>
<Service description="The Sample Service" name="SampleService"
        packageName="SampleService">
        <osTypes>
          <osType name="all"
            startCmd="$SOAM_DEPLOY_DIR/SampleServiceCPP"
            workDir="$SOAM_HOME/work">
          </osType>
        </osTypes>
      ......
</Service>
```

图 3.11 网格应用程序配置文档图

它定义了服务的 QoS 和服务地址，如指定了消费者和网格应用程序、是否预启动服务以及预启动的服务实例个数、Session 的资源请求因子和资源释放因子和 Session 资源分配策略（见 4.2 节）。此外，系统可以提供更为丰富的控制参数：

1) 系统性能阀值。设置参数对 JM 进程对物理和虚拟内存的使用量进行控制，当内存使用量到一定百分比如 60%时，将当前所有 Session 的数据都写入硬盘并释放这些 Session 所占的内存；如果内存使用量继续增加比如 90%，JM 进入维护状态，拒绝接受所有的客户端请求，并停止执行当前作业，直至内存使用量降至正常值 JM 才恢复正常工作。

2) 各种 Session 类型的系统行为，如与客户端连接中断时是否放弃 Session、Task 重试次数、Session 重试次数、Session 优先级、Session 是否可恢复以及 Task 清理时间等。Task 清理时间指的是正在执行 Task 的计算单元有可能是从其它的资源消费者借来的，计算单元的拥有者有权收回并设置收回时间，在这个时间内，Task 还可以继续执行，如果执行完毕即发回计算结果，否则计算 Task 被强制中断。Session 是否可恢复（Recoverable 和 Unrecoverable）是系统对计算数据容错机制。对于可恢复的 Session，在 Task 发送至 JM 进程后，首先进行输入数据的序列化处理，将 Session 数据和 Task 输入数据按照二进制数据保存至磁盘，在任务完成后，将任务输出数据序列化。

3) 服务的异常处理，配置系统在服务容器 Register，CreateService，SessionEnter，Invoke，SessionLeave，DestroyService（服务容器程序 API）等各个执行服务阶段出错时的处理机制。可以指定在服务执行实例出现超时、非正常退出，返回指定控制代码，一般异常，严重异常等情况下对计算 Task 和服务实例的处理方法，处理方法有失败退出、重试和阻塞主机（放弃在此主机上启动的服务实例并将该主机放入阻塞主机队列里）等。

3.5 本章小结

本章首先分析了金融领域需求，对网格技术在金融领域的应用种类进行了划分，设计了系统的作业方式。然后研究了网格作业管理系统的功能层次模型，把网格作业管理系统划分为资源管理和负载管理两个独立模块，在此基础上结合SOA 完成了对企业计算网格作业管理系统的体系架构设计。最后，对客户应用、服务和服务契约等系统服务流程的几个方面进行了详细设计。

第四章 作业调度管理系统的设计与实现

作业调度管理系统是网格作业管理系统的核心，也是研究的重点之一。根据第三章设计的企业计算网格作业管理系统体系架构，本章应用此架构设计并实现了网格作业管理系统中的核心组件，即作业调度管理系统。

4.1 作业调度管理系统设计

4.1.1 系统设计目标

作业调度管理系统负责接受客户端提交的服务请求，然后根据调度策略进行作业的分发，此外还要负责管理作业的状态以及相关操作。它提供的功能如图 4.1 所示。

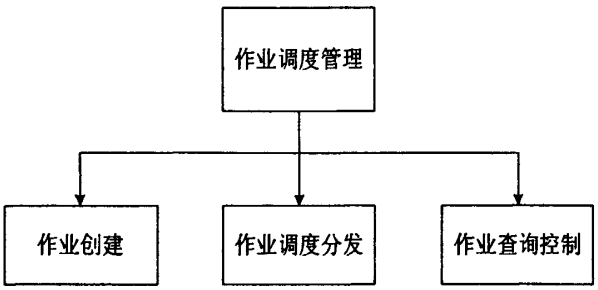

图 4.1 作业调度管理功能示意图

作业调度管理系统主要完成作业创建、作业调度分发和作业查询控制三个功能。

作业创建功能：主要是将客户端的服务请求在系统内部转化为作业，以便将其请求调度分发。根据 3.1.1 节的分析，作业被分为 Session 和 Task。因此，作业的创建就包括 Session 的创建和 Task 的创建。

作业调度分发功能：这是系统的核心功能，主要根据指定的调度策略产生作业分配方案（即将 Session 和服务进行绑定），然后根据作业分配方案将作业分派到各个计算节点上的服务中去。

作业查询控制功能：主要是对作业的相关控制操作和查询功能。用户可以查询作业执行的状态信息，包括作业提交时间，运行状态，以及作业实例的执行历史纪录等。并允许管理员通过管理接口控制作业的执行，例如可以暂停、继续、终止作业。

4.1.2 系统结构设计

根据功能分析对作业调度管理系统进行了设计。它的结构图如 4.2 所示。

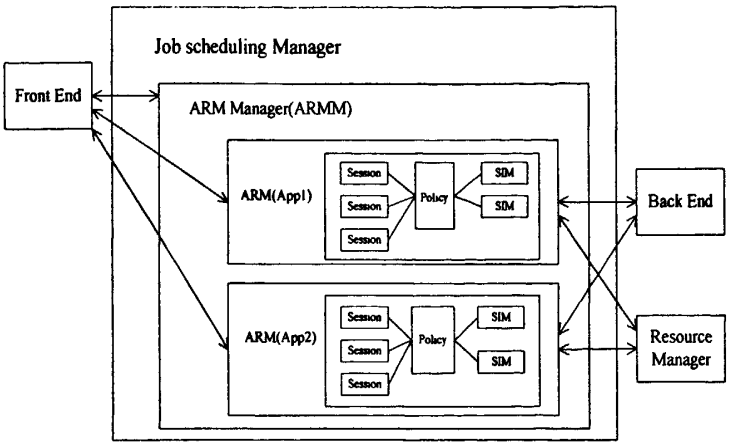

图 4.2 作业调度管理结构图

作业调度管理系统主要对象有：ARM Manager，ARM，Session，Task，和 Policy，主要类的描述如表 4.1 所示。

表 4.1 主要类的描述

ARM Manager	当客户端连接到系统后，被 Front-end 用来创建新的或获得已有 ARM。
ARM	联系 Front-end, Resource Manager 和 Back-end 几个组件。 拥有分配给网格应用程序的计算资源（SIM）的集合。 拥有网格应用程序里所有的 Session 集合。 实施作业调度，绑定/解除绑定计算资源与 Session 的关系。
Session	拥有 Task 集合并负责维护 Task 的状态并将下一个需要执行的 Task 送交给 Policy。
Task	拥有客户端发送的计算数据和服务返回的计算结果。
Policy	负责作业调度。

高效、健壮、可扩展和灵活是系统的设计目标，但是在网格环境中，会出现很多与构建并发和网络化系统有关的设计和编程难题。其中复杂性的一个来源是常见的多线程的事故，例如竞争条件和死锁，而另外一个来源是开发方法、工具和操作系统平台的局限，尤其是流行的硬件和软件平台的异构性，使开发在多种操作系统上运行的并发应用程序和工具变得极其复杂。在作业调度管理系统中，并发和同步是核心难题，因此这些问题在设计的时候就应该给予考虑。目前，已经有一些用于并发和网络化软件的关键设计模式[27]，这些模式能够并且已经用于解决在开发面向对象的中间件框架和应用时出现的许多常见的问题，并发模式有主动对象(Active Object)、监视器对象(Monitor Object)、半同步/半异步(Half-Sync/Half-Async)、领导者/追随者(Leader/Follower)和线程特定的存储器等，同步模式有定界加锁、策略化加锁和双检查加锁优化等。在设计类的时候，使用了这些常见的设计模式。以下详细论述几个关键的设计。

1. Policy 类的设计

Policy 类实现了作业调度策略。由于网格作业调度很复杂，每种调度策略都各有优缺点，适合于不同的场合和目的，所以需要考虑将调度策略的具体实现和抽象接口之间进行解耦。Strategy 模式体现了面向对象设计中的依赖倒置原则，低层模块实现高层模块声明的接口，低层模块依赖高层模块，这样控制权在父类（高层模块）。因此使用 Strategy 模式可以方便的在子类实现不同的调度策略，而对外部暴露统一的接口。如果有新的策略模式添加到系统中，只需要从父类继承并实现父类中的抽象方法即可。

另外，为了提高作业调度的速度，使用了主动对象模式。主动对象设计模式使方法执行与方法调用去耦合，以增强并发、并简化对驻留在它自己的线程控制中的对象的同步访问。Policy 类对象在初始化后就创建自己的工作线程，在工作线程里执行循环：执行调度算法；等待一个事件。这样既提高了工作效率，又简化了同步的复杂性。

2. Session 状态设计

Session 有四个状态：开始、挂起、结束和终止，其状态转移如表 4.2 所示。

表 4.2 Session 状态转换表

当前状态	活动	下一状态
无	客户端建立与系统连接	开始
开始	所有Task都完成计算	结束(如果收到最后一个计算结果)
		开始(如果没收到最后一个计算结果)
	被用户挂起	挂起
	失去与客户连接（Session被设置为可恢复）	开始
	失去与客户连接（Session被设置为不可恢复）	终止
	通过GUI或CLI被用户结束	终止
	至少有一个Task失败，同时AbortSessionIfTaskFails被设置为TRUE	终止
	Session结束	结束
挂起	被用户恢复	开始
	失去与客户连接（Session被设置为可恢复）	挂起
	失去与客户连接（Session被设置为不可恢复）	终止
	通过GUI或CLI被用户结束	终止
结束	无	无
终止	无	无

3. Task 状态设计

Task 的状态和服务实例的当前状态有对应关系，它有 5 个状态：就绪、执行、完成、出错和取消。其状态转移如表 4.3 所示：

表 4.3 Task 状态转换表

当前状态	活动	下一状态
无	客户端发送 Task	就绪
就绪	开始在计算节点上执行	执行
	Task 被用户挂起	就绪
执行	成功完成计算	完成
	Task 被用户挂起	就绪
	执行 Task 的 CPU 被占用	就绪
	计算节点被关闭	就绪
	计算节点死机	就绪
	计算节点和系统的连接中断	就绪
	SIM 崩溃	就绪
	SIM 宕	就绪
	运行 Task 的服务实例崩溃	就绪(Task 的重试次数没有超过指定限度)
		出错(Task 的重试次数超过了指定限度)
	服务抛出异常	就绪(Task 的重试次数没有超过指定限度)
		出错(Task 的重试次数超过了指定限度)
	Task 被终止.	取消
完成	无	无
出错	无	无
取消	无	无

根据上述分析，确定了主要类的职责及设计方法，系统主要类图如图 4.3 所示。

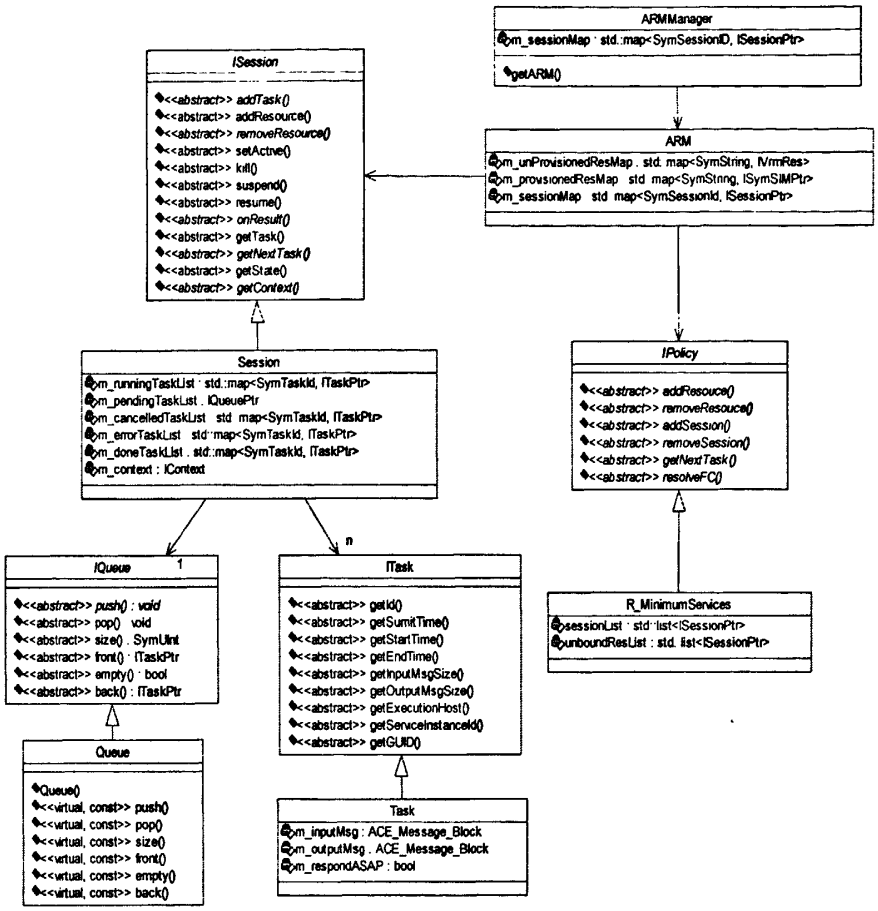

图 4.3 系统主要类图

图 4.3 列出了各个类的主要数据成员和成员函数以及各个类之间的关系。为了正确的释放资源，每个类都有自己的引用计数，表 4.4 描述了数据视图。

表 4.4 数据视图

数据类型	对象数量	存储类型	生命周期	
			创建	销毁
Application Profile	1	内存	客户连接系统	当 ARM 被析构
Session	1	内存	创建 Session	超时或结束
Context Data（Session 的公共数据）	1	内存/数据存储	创建 Session	Session 超时或结束
Task	1	内存	发送 Task	Session 超时或结束
Input（Task 输入数据）	1	内存/数据存储	发送 Task	SIM 返回计算结果或者被持久保存
Output（Task 输出数据）	1	内存/数据存储	SIM 返回计算结果	从客户端得到确认收到消息或者被持久保存

4.2 系统作业调度策略设计

作业调度策略为作业调度管理系统产生作业的资源分配表，然后作业调度管理系统就可以依据这张表进行作业的相关处理。这张表的优劣就决定了系统的作业调度的优劣，直接关系到系统的整体性能。

4.2.1 设计原则

作业调度的主要目的是匹配资源的供给与需求，即根据给定的策略申请资源，然后将申请到的资源分配给作业。它的资源分配粒度在是在 Session 一级。同时还要遵循以下原则：

1. 作业调度策略要避作业调度模块频繁的向资源管理申请、释放资源。
2. 作业调度策略应该能够有效地减少对于 Task 分派的性能影响，提高 CPU 的利用率。
3. 作业调度策略产生的资源分配方案应该是将资源分配给最应该得到的作业。
4. 作业调度策略的设计要保持与 Resource Manager（资源提供者）、作业（消费者）和 ARM(控制器)的紧密联系。
5. 调度策略的设计要保证重要的 Task 总是能够得到及时响应。

4.2.2 基于优先级的作业调度策略设计

作业调度策略的重要目标是保证作业调度管理系统行为的可预测性。可预测性指的是在系统运行的任何时刻，系统的作业调度策略都能为争夺计算资源的多个作业合理地分配资源，使每个作业的实时性要求都能得到满足。以下是几个常用的调度模式[28][29]：

1. FCFS(First Come First Service)模式

FCFS 模式是最基本的调度模式。队列中的作业根据其提交时间，顺序排列并依次被分派和执行。FCFS 最大的优点是实现容易，并且作业的执行顺序是可以预见的。它的缺点是系统的吞吐率和利用率低，而且无论对于作业还是对于用户都是一种不公平的调度策略，具有创建饥饿作业的缺点（所谓饥饿作业是指已经等待了很长时间的作业）。

2. 独占模式

独占模式使一个作业独占使用它运行所在的主机，直到该作业运行完。该模式保证了作业一旦运行就能顺利的运行完，不会被其他作业打断运行。它通过改变资源使用方式，确保了作业对资源的独立占有，进而提高作业本身的运行性能。

但是它的不足也很明显，这种模式没有考虑作业的优先性，可能导致次要作业占据了资源而无法保证重要作业对资源的需要。

3. 基于优先级的抢占调度模式

这种模式允许高优先级的挂起作业从优先级低的正在执行的作业中抢占资源。作业的优先级在提交作业的时候就设定了。这种模式适用于对实时性有要求的领域。它考虑了优先性，有效的克服了 FCFS 模式下，高优先级作业不能得到及时响应的缺点。但是，这种模式的不足之处在于很难设定作业优先级的最优准则。如果拥有较高优先级的作业需要的资源比目前可用的资源多，那么就会导致优先级较低的作业长时间等待，且不能充分利用资源。

这几个调度模式各有优缺点，适合于不同的场合和目的。根据设计目标，综合这三个模式的优点，设计了一个改进的基于优先级的作业调度策略。系统中作业调度的过程分为两步：第一步是为网格应用程序向资源管理申请资源；第二步是将申请到的资源在各个 Session 之间进行分配。因此所设计的调度策略包含这两步，下面分别论述：

1. 资源的申请与释放

资源的申请与释放是调度策略的第一步。作业调度模块的资源分配粒度是在一个网格应用程序运行时创建的 Session。一个网格应用程序在网格服务被部署之后，可能在同一时刻由多个网格客户发起 Session 请求，理想情况下所有作业的所有 Task 都有一个逻辑核为它计算，但这在大规模的分布式计算中是难以满足的。在一般情况下，每个逻辑核将有一个就绪（即将执行而未执行）Task 列表，在不断有新 Task 加入的情况下作业调度模块需要决定何时申请新的逻辑核以避免 Task 队列堵塞太多的 Task，并同时配备了两个参数：资源申请因子（H）和资源释放因子（L），通过这两个比例参数就形成了一个资源申请释放的缓冲，有效避免了资源的频繁申请释放，节约了系统开销，保证了系统的高利用率。

在一个网格应用程序中总共有 N 个开始状态 Session：S1, S2... Sn，（0=< N < 1000000），当前资源总数为 R，则问题的形式化分析如下：

1) 参与调度的 Session 优先级集合 SP={P1, P2, ..., Pn}，其中 Pn 代表的是第 n 个 Session 的优先级，则所有开始状态 Session 中优先级总数：P=P1+P2+...+Pn。
2) 参与调度的 Session 最小服务数配置集合 SM={M1, M2,...,Mn}，其中 Mn 代表的是第 n 个 Session 的最小配置服务数，则所有开始状态的 Session 中最小配置服务总数 M=M1+M2+...+Mn。
3) 参与调度的 Session 的未处理的 Task 数(包括就绪和执行状态)集合 ST={T1, T2, ..., Tn}，其中 Tn 代表的是第 n 个 Session 中的未处理的 Task 数，则所有开始状态 Session 中未处理的 Task 总数 T=T1+T2+...+Tn。

4) 参与调度的 Session 的实际最小服务数集合 MinSn={MinS1, MinS2, ..., MinSn}，其中 MinSn 代表第 n 个 Session 的实际最小服务数：MinSn = Max (Mn, Round (Tn/H) + 1)。

5) 参与调度的 Session 的实际最大服务数集合 MaxSn={MaxS1, MaxS2, ..., MaxSn}，其中 MaxSn 代表第 n 个 Session 的实际最大服务数：MaxSn = Max (Mn,Round(Tn/L) + 1)。

6) 根据(4)的定义：网格应用程序的最小服务数 MinS = MinS1 + MinS2 + ... + MinSn。

7) 根据(5)的定义：网格应用程序的最大服务数 MaxS = MaxS1 + MaxS2 + ... + MaxSn。

所以，一个网格应用程序的资源申请释放规则是：

当 R < MinS，作业调度模块将向 EGO 申请（MinS - R）个计算资源。

当 R > MaxS，作业调度模块将向 EGO 释放（R - MaxS）个计算资源。

当 MinS <= R<= MaxS，作业调度模块将保有现有计算资源，不申请也不释放。

2. 资源的分配

资源分配是调度策略的第二步。在获得资源后，作业调度再将这些资源分发给各个 Session。根据以上分析，设计了作业选择算法。算法步骤如算法 4.1 所示。

算法 4.1 作业选择算法

```
BEGIN
  FOR（所有的作业）
    使用先来先服务模式满足作业的最小服务数
  ENDFOR
  IF（在满足了所有 Session 的最小资源数后，还有剩余的资源）
      FOR（所有的作业）
         //作业的资源分配数为最小资源数与共享分配数之和
        按照基于优先级的比例分配模式计算共享分配数
      ENDFOR
  ENDIF
  FOR（所有的作业）
      IF（作业的未完成 Task 数小于最小资源数与共享分配数之和）
        该作业就完成了本次调度，将其移出调度作业队列。 //它的实际分配数即为最小资源数
      ENDIF
  ENDFOR
  FOR（仍有剩余资源）
      剩下的作业按照优先级从高到低排序，形成一个有序队列
      为队列中优先级最高的 Session 分配一个资源
      IF（该作业现在所分配的资源数大于未完成 Task 数）
        该 Session 就完成了本次调度，将其移出队列     //它的实际分配数即为共享分配数加 1
      ELSE
        将该作业移至队列底部
      ENDIF
  ENDFOR
END
```

这个算法相当灵活，它提供了最小服务数供客户根据需要配置。极端情况下，如果将最小服务数设置为零，则是抢占模式，而如果将最小服务数设为最大，则是独占模式。该策略首先使用 FCFS 模式满足每个 Session 的最小服务数要求，对于先分配给每个 Session 的资源，Session 采取独占策略，这就保证了重要 Task 总是能够得到响应，而且避免了启动时间较长的服务在服务的反复切换中消耗大量的系统时间；然后剩下的资源再基于优先级进行分配，考虑到为了避免优先级高的 Session 无论还有多少未完成的 Task 都占据资源，将根据作业的优先级和该作业中未完成的任务的多少共同决定作业的最终资源分配数。

4.3 作业调度管理系统实现

作业调度管理具有作业创建、作业调度和作业查询控制三个功能，下面详细论述这三个功能的实现。

4.3.1 作业创建功能实现

对于客户提交的服务请求，作业调度管理将其转换为作业。根据 3.1.1 节的分析，作业又分为 Session 和 Task。先创建 Session，再创建 Task。

1. Session 创建

作业调度的资源分配粒度在 Session 一级，因此，在 Session 创建以后就需要将其加入到 Policy 的调度队列里面去。Session 创建如图 4.4 所示。

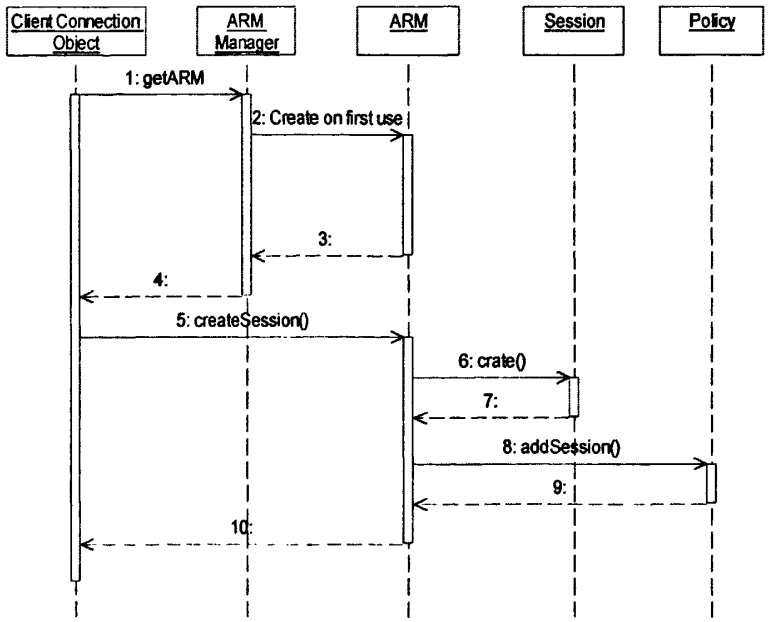

图 4.4 Session 创建时序图

客户端首先调用 getARM()从 ARM Manager 获得管理当前网格应用程序的

ARM，然后再调用 CreatSession()，此时 ARM 会创建一个 Session 对象，然后 ARM 调用 addSession()，将其添加到 Policy 的调度队列中，使 Session 能够被调度。

2. Task 创建

Session 在创建后，就需要创建实际的计算单元——Task，创建过程如图 4.5 所示。

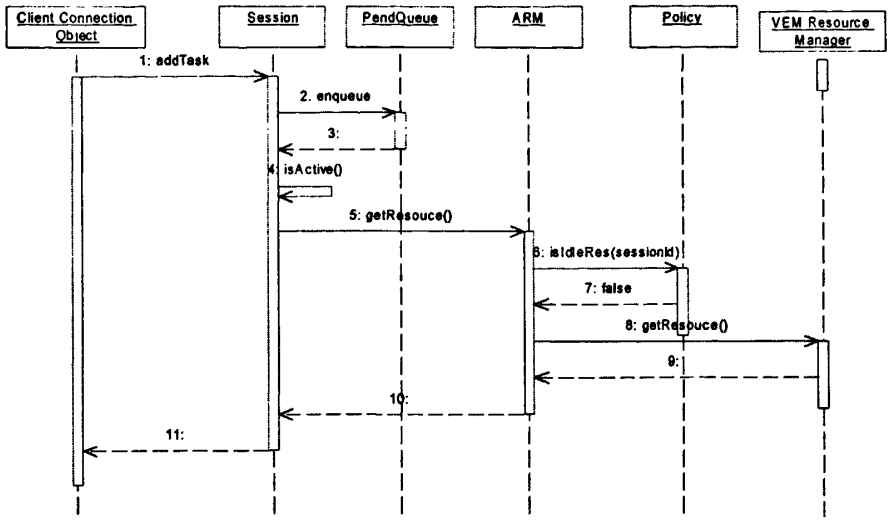

图 4.5 没有空闲资源时 Task 时序图

当客户端调用 addTask()时，Session 会创建一个 Task 对象，然后调用 enqueue()将其插入到等待队列中，此时 Task 的状态是就绪；然后 ARM 调用 IsIdleRes()向 Policy 查询 Session 当前是否有空闲资源，在当前没有空闲资源情况下，ARM 调用 getResource()向 VEM 申请资源。

4.3.2 作业调度分发功能实现

根据 4.2 节设计的调度策略实现 Policy 类。它根据调度策略产生当前的资源分配表，如果发现当前可用资源少于某种程度，则发出资源申请；如果发现当前可用资源多于某种程度，则发出资源释放申请；同时当资源的申请得到满足，它要负责将该资源添加进来；资源分配表就是 Session 和它可使用的资源的绑定表，当 Task 创建后，Task 就可以分发到它所属的 Session 所对应的空闲计算资源上进行计算。所以作业调度分发的功能包括资源的申请、释放、添加和作业的分发，下面分别论述。

1. 资源申请

在同一时间某个网格应用程序可能存在多个 Session，Session 的负载变化会导致资源的重新请求和释放，为了避免过于频繁地向资源层发送资源请求，定义了一个时间间隔参数（在网格应用程序配置文档里）。系统每隔一段时间，才将排在

队列里面的资源请求统一发送给资源层。资源层资源申请时序关系如图 4.6 所示。

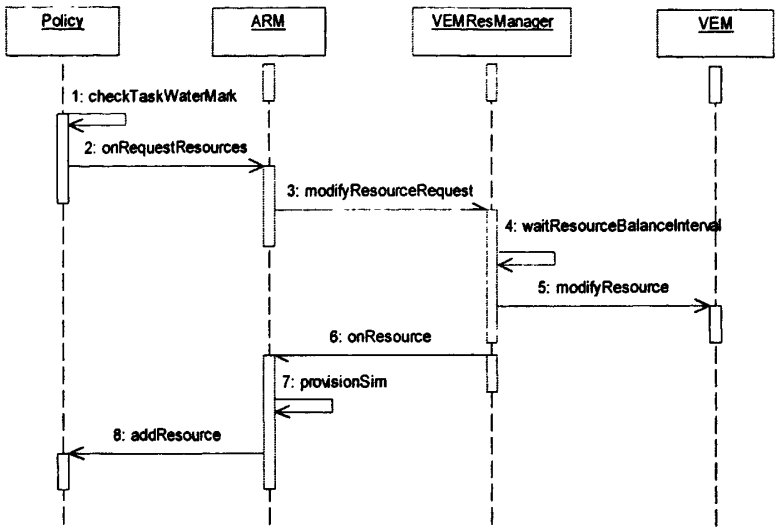

图 4.6 申请资源时序图

Policay 调用 checkTaskWaterMark()，根据作业的最小服务数和资源申请因子计算作业的最小资源请求数，当资源请求数目超过时才提出申请；VEM resource manager 会将 Policy 的请求入列，并在一定的间隔向 EGO 发送这些请求。

2. 资源释放

资源释放时序关系如图 4.7 所示。

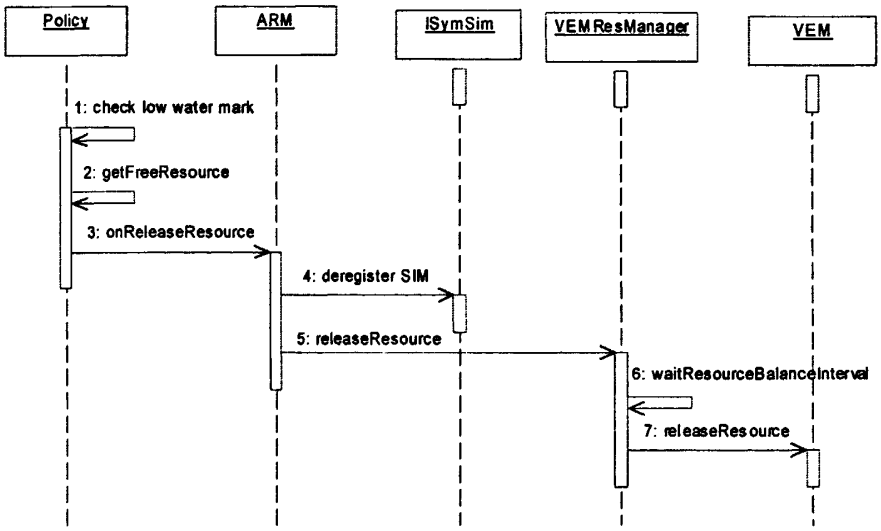

图 4.7 释放资源时序图

Policy 首先根据作业的最大服务数和资源释放因子计算作业的最大资源请求数，当资源请求数目小于这个数时才提出申请释放；VEM resource manager 会将 Policy 的请求入列，并在一定的间隔向 EGO 发送这些请求（在归还资源之前，ARM 需要解除该资源的注册）。

3. 资源添加

资源添加时序关系如图 4.8 所示。

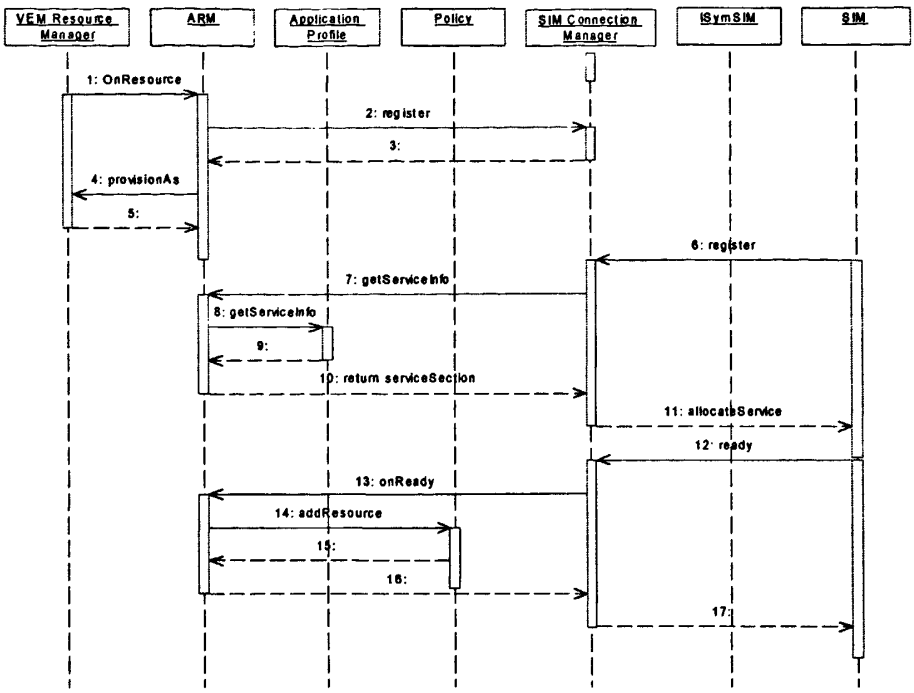

图 4.8 资源添加时序图

申请到资源后，EGO 会在该计算节点启动 SIM 进程，SIM 进行初始化后首先会调用 register()向 SSM 注册，接着 ARM 根据 Application Profile 配置的 Application 默认启动的服务调用 allocateService()通知 SIM 启动服务。在 SIM 成功启动服务后调用 ready()通知 ARM 服务准备就绪，这时 ARM 才调用 addresource(), 让 Policy 将这个申请到的资源加入到资源队列中。

4. 作业分发

作业的分发主要发生在以下几种情况：

1) 在 Task 创建的时候，如果 Task 所属 Session 拥有空闲资源的时候，Task 可以直接使用该空闲资源，时序关系如图 4.9 所示。

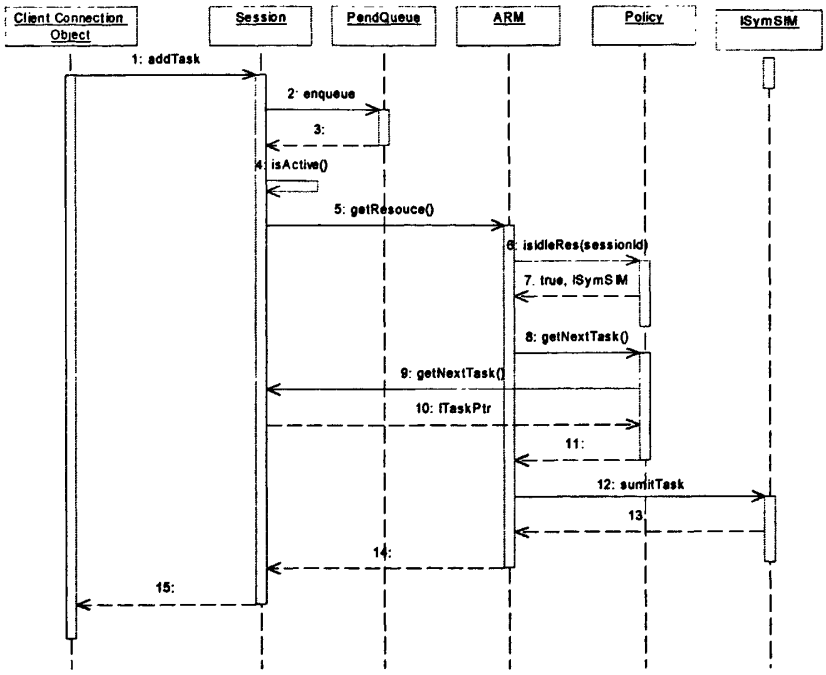

图 4.9 有空闲资源时 Task 时序图

Task 创建后，ARM 调用 IsIdleRes()向 Policy 查询 Session 当前是否有空闲资源，在当前有空闲资源情况下，ARM 调用 sendTask()向服务发送 Task。

2) 在计算节点返回计算结果的时候，如果 Session 的就绪 Task 队列里还有未计算的 Task，则将该 Task 发到计算节点中去。时序关系如图 4.10 所示。

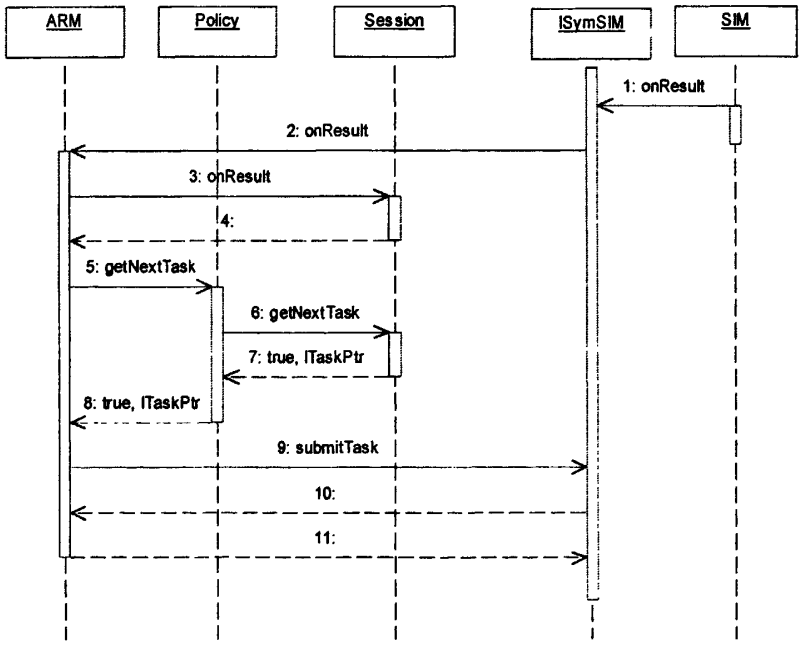

图 4.10 申请资源时序图

当 Task 计算完成后，SIM 会调用 onResult()通知 ARM 返回计算结果，在 ARM

将计算结果保存在 Task 后，ARM 会调用 getNextTask()以获得下一个 Task，如果作业还有 Task，ARM 调用 submitTask()继续向服务发送 Task，否则就结束。

4.3.3 作业查询控制功能实现

作业调度管理系统保存着作业，要根据作业的执行情况实时更新作业状态，并对外提供相关接口以方便对其状态的查询。作业的控制主要是对作业运行状态的控制，它主要是提供一些底层接口供 SD 进行操作。它分为对 Session 的控制和对 Task 的控制。控制行为的实现是根据 Session 和 Task 的状态表进行。系统中的每个 Session 和 Task 都有唯一的全球唯一标识符（GUID）。下面仅以停止 Session 为例讲述实现方法。时序关系如图 4.11 所示。

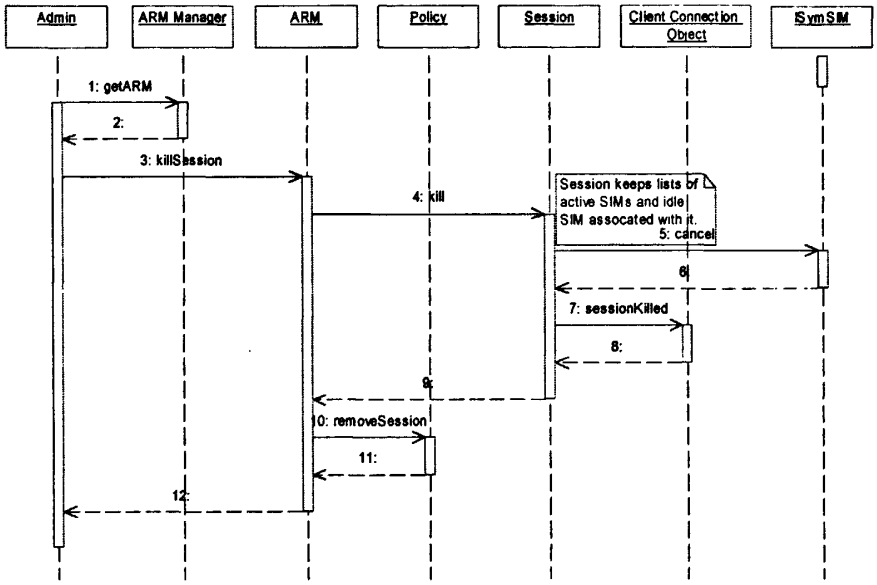

图 4.11 停止 Session 时序图

通过调用 getARM()从 ARM Manager 获得当前网格应用程序的 ARM，然后调用 killSession()向 ARM 发送停止 Session 消息。ARM 在调用 Kill()后，要做一些清理工作，要将与该 Session 绑定的资源都释放掉。最后，需要将 Session 移出 Policy 的调度队列。

4.4 本章小结

本章基于在第三章所提出系统架构设计并实现了网格作业管理系统中的核心组件，即作业调度管理系统。首先分析作业运行管理系统功能需求，然后完成了系统设计，并设计了一种改进的基于优先级的作业调度策略并将其应用到系统中。最后完成作业运行管理系统中的作业创建、作业调度和作业查询控制的实现。

第五章 系统测试

本章主要是从可用性、可靠性、性能和可伸缩性等四个方面，对作业调度管理系统进行具体的测试分析。为了便于在网格环境中测试，首先开发了一个系统诊断工具。系统的测试主要在企业的局域网内进行，测试环境如表5.1所示。

表 5.1 测试环境

	Windows	Linux
主机类型	IBM eServer BladeCenter LS20	IBM eServer BladeCenter LS20
CPU	Intel P4 3.06	AMD64 2.4
内存（GB）	2.5	2.5
操作系统	Windows 2003 Enterprise	RHEL4
网络	主机间 G 级网络	主机间 G 级网络
SCSI Driver	ST973401LC	ST973401LC

5.1 系统诊断工具

为了能够测试和验证在分布式环境中系统各组件的工作状况，开发了专门用于测试系统性能的测试服务。在客户端可以灵活指定计算 Task 和 Session 的各种特性，比如指定 Task 的输入输出数据大小，Task 执行时间，每个 Session 的 Task 数量等参数。测试服务命令如图 5.1 所示。

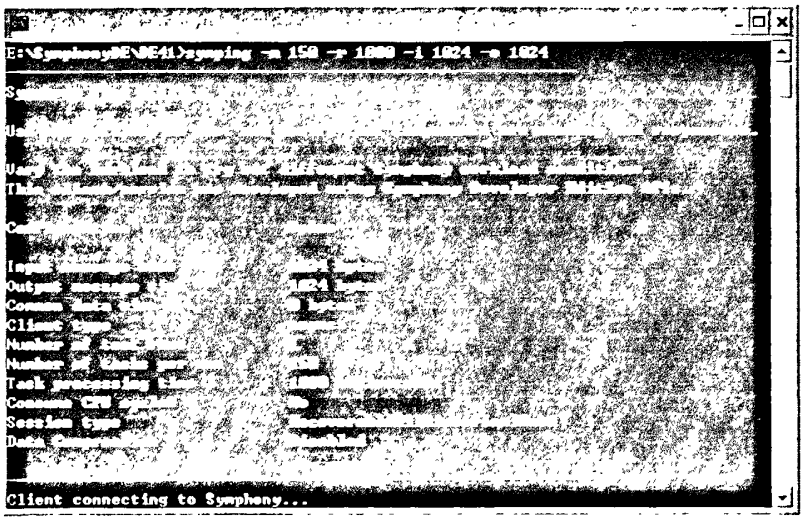

图 5.1 测试服务命令图

命令（Symping –m 150 –r 1000 –i 1024 –o 1024）同步发送 150 个计算 Task，每个 Task 执行 1 秒，输入数据 1KB，Task 执行完后返回 1KB 的输出数据。

此外，命令还可以指定 Session 数据的大小、每个客户端连接中的并发 Session 线程数、Task 的发送类型（同步/异步）以及 Session 是否可恢复等参数提供更为复杂的计算 Task 发送和处理方式。

通过此工具在多 CPU 环境下可以测试出中间件所能提供的性能指标，具体如图 5.2 所示。

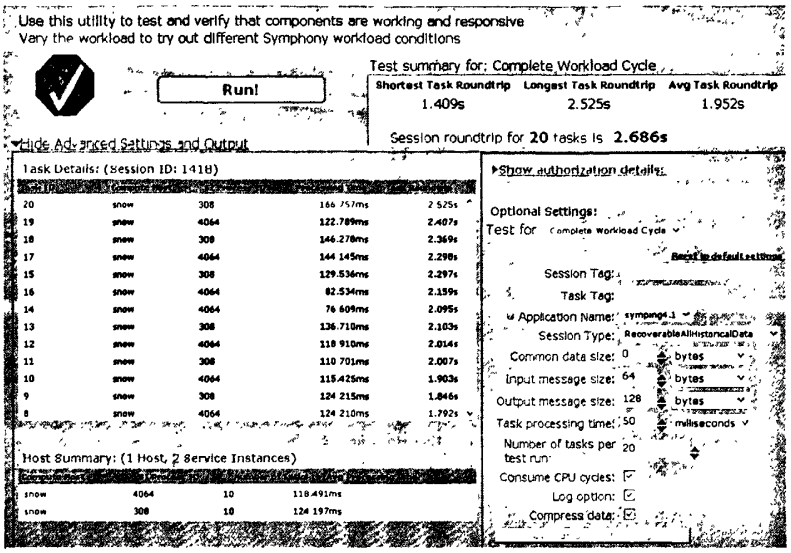

图 5.2 测试服务结果界面图

1. Task 开销时间

处理一个任务的系统开销时间。它等于 Task 的往返时间–Task 的实际执行时间。

2. Task 执行时间

任务的处理时间（无系统开销）：就是服务容器执行 onInvoke()调用的时间。时钟开始于 SIM 调用 onInvoke()，结束于 SIM 调用 onInvoke()返回。

3. Task 往返时间

一个任务的完整执行周期（Task 开销时间+Task 执行时间）。它等于接收到返回 Task 时间–发送 Task 时间。

4. Session 往返时间

Session 里所有 Task 的完整任务周期的总合。它等于接收到最后一个返回 Task 时间–发送第一个 Task 时间。

5. 测试时间

时钟启动于用户运行 symping 命令，结束于 symping 返回客户端应用程序的结果。测试时间包括初始化时间，连接时间，以及创建 Session 之前的所有时间。

5.2 系统功能测试

5.2.1 测试目标

系统功能测试主要是根据产品规格说明书来测试作业调度管理系统是否满足各方面功能的使用要求。以下根据设计目标设计了测试用例。

5.2.2 测试用例

测试用例主要分为三类：

1. 基本测试用例

使用系统诊断服务工具进行测试(Symping –m 20 –r 50 –i 64 –o 128)。测试结果如图 5.3 所示。

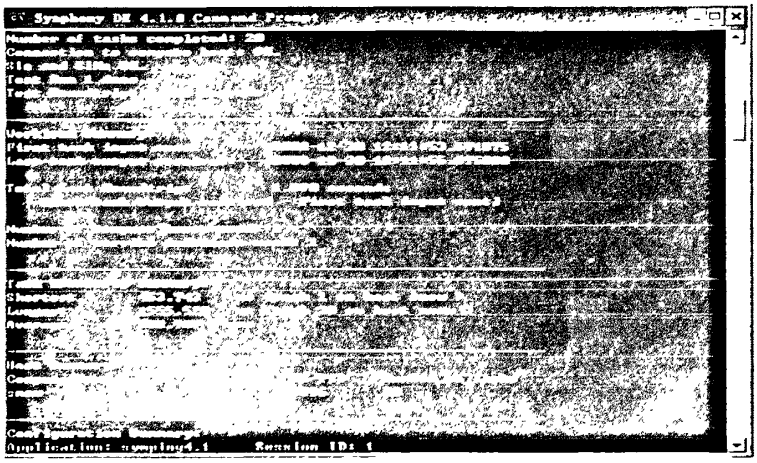

图 5.3 基本测试用例结果图

测试数据正确反映了测试命令，测试结果表明系统各模块都正常，能够完成基本的功能。

2. 作业的查询控制功能

使用系统诊断服务工具进行测试(Symping –m 20 –r 50 –i 64 –o 128)，在 Session（ID 为 2）运行时使用命令(soamcontrol session kill symping4.1:2) 停止该 Session，然后察看 Session 的状态(soamview session symping4.1:2 -l)，结果如图 5.4 所示。

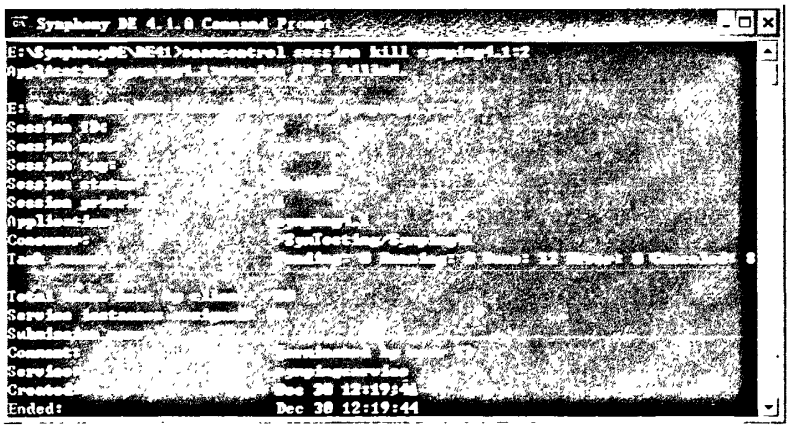

图 5.4 作业查询控制测试结果图

用户使用停止 Session 命令后，Sesion 就成为失败状态，再查询该 Session 的信息，状态、原因等信息均正确。测试结果表明作业查询控制功能正常。

3. 作业调度功能

资源的申请与释放测试用例如表 5.2 所示。

表 5.2 资源的申请与释放测试

用例编号	测试内容	测试条件					预期结果	测试结果
		资源申请/释放因子	Session 最小服务数(Si 比 Si+1 早创建)	Session 优先级	Task 数量 L- 长执行时间 task; S- 短执行时间 task	可用计算节点数		
1_1	申请资源	2.0/1.0	S1:5 S2:5 S3:10	相同	S1:20L S2:20L S3:20L	30	系统将会申请 30 个计算资源	正确
1_2	申请资源	2.0/1.0	S1:5 S2:5 S3:15	相同	S1:20L S2:20L S3:20L	30	系统将会申请 35 个计算资源	正确
1_3	申请资源	4.0/1.0	S1:5 S2:5 S3:10	相同	S1:20L S2:20L S3:20L	30	系统将会申请 20 个计算资源	正确
1_4	释放资源	1.0/1.0	S1:5 S2:10	相同	S1:10L,5S S2:10L	30	系统首先将会申请 25 个计算资源。在短 Task 结束后，会释放掉 5 个资源。	正确
1_5	释放资源	1.0/1.0	S1:5 S2:10	相同	S1:10L,5S S2:10S	30	系统首先将会申请 25 个计算资源。在短 Task 结束后，会释放掉 5 个资源。	正确
1_6	释放资源	2.0/1.0	S1:10 S2:15	相同	S1:10L,10S S2:10S	30	系统首先将会申请 25 个计算资源。在短 Task 结束后，系统不会释放掉任何资源。	正确

作业调度测试用例如表 5.3 所示。

表 5.3 作业调度测试

用例编号	测试内容	测试条件					预期结果	测试结果
		资源申请/释放因子	Session 最小服务数(Si 比 Si+1 早创建)	Session 优先级	Task 数量 L- 长执行时间 task; S- 短执行时间 task	可用计算节点数		
2_1	先来先服务(当可用的资源数不能满足最小服务要求)	1.0/1.0	S1:10 S2:11 S3:12	5/6/7	S1:100L S2:100L S3:100L	25	S1 得到 2 S2 得到 11 S3 得到 12	正确
2_2	资源保持(不在重新分配已经分了的最小资源)	1.0/1.0	S1:10 S2:10 S3:10	5/5/100	S1:10L S2:10L S3:100L	25	在 S3 创建之前: S1 得到 10 S2 得到 10 在 S3 创建之后: S1 得到 10 S2 得到 10 S3 得到 5	正确
2_3	满足了最小服务要求后,剩余的资源按照优先级分配	1.0/1.0	S1:5 S2:5 S3:10	5/5/10	S1:100L S2:100L S3:100L	25	S1 得到 6 S2 得到 6 S3 得到 13	正确
2_4	满足了最小服务要求后,剩余的资源按照优先级分配	1.0/1.0	S1:5 S2:5 S3:10	5/5/10	S1:1L S2:100L S3:100L	25	S1 得到 5 S2 得到 7 S3 得到 13	正确
2_5	满足了最小服务要求后,剩余的资源按照优先级分配	1.0/1.0	S1:5 S2:5 S3:10	5/5/10	S1:100S S2:100L S3:100L	25	S1 得到 6 S2 得到 6 S3 得到 13 在 S1 的所有计算任务完成后, S1 得到 5 S2 得到 7 S3 得到 13	正确
2_6	满足了最小服务要求后,剩余的资源按照优先级分配	1.0/1.0	S1:2 S2:2 S3:2	5/5/10	S1:100S S2:100L S3:100L	25	S1 得到 7 S2 得到 7 S3 得到 11 在 S1 的所有计算任务完成后, S1 得到 2 S2 得到 8 S3 得到 15	正确

5.3 系统可靠性测试

1. 测试目标

作业调度管理系统在压力测试下能够持续正常运行两个星期以上。

2. 测试条件

测试条件如表 5.4 所示。

表 5.4 测试条件

网格应用程序数量	100
每个网格应用程序的并发客户端数量	1000
Session 数量	每个客户端持续不断的创建 Session
Session 中 Task 数量	1M
Task 平均执行时间	1.5sec
公共数据大小	800MB
输入数据大小	100MB
输出数据大小	500MB
服务实例数量	4000

3. 测试结果

在测试期间，没有发生以下任何一个问题，通过可靠性测试。

1) 没有出现核心转储、崩溃或者宕情况；
2) 没有发生内存泄漏或其它内存问题；
3) 没有作业丢失或异常退出现象；
4) 所有的作业计数正确；
5) 作业的发送吞吐量没有下降；
6) 不影响对客户端应用程序的响应；
7) 没有明显性能恶化。

5.4 系统性能测试

作业调度管理系统性能测试的指标有四个：Task 响应时间(Task Latency)、Session 往返时间(Session Round Trip)、Task 发送吞吐量(Task Sending Throughput)和 CPU 利用率(CPU Efficiency)。以下对这四个指标进行测试。在测试 Task 发送吞吐量和 CPU 利用率中同时对系统的可伸缩性也进行了测试。

5.4.1 Task 响应时间测试

Task 响应时间是指一个执行时间为 0s 的 task 的往返时间：从发送 Task 时刻开始到返回计算结果结束（不包括建立连接和创建 Session 的时间），测试用例如表 5.5 所示。

表 5.5 Task 响应时间测试用例

Task Latency								
目标		Task 往返时间: 5ms						
测试数据		Task 实际执行时间: ms; task 往返时间: ms;						
主要参数		可恢复性: rec/unrec; 发送方式: sync/async; 接收方式: sync/async; 调度策略: R_MiniService						
固定参数		Task 实际执行时间: 0 seconds						
用例编号	CPU 数量	是否可恢复	发送方式	接收方式	公用数据 (bytes)	输入数据 (bytes)	输出数据 (bytes)	响应时间(ms) Windows/Linux
1	1	unrec	sync	sync	102,400	1,024	1,024	4.798/4.426
2	1	unrec	sync	async	102,400	1,024	1,024	4.972/4.072
3	1	unrec	async	sync	102,400	1,024	1,024	4.752/3.835
4	1	unrec	async	async	102,400	1,024	1,024	4.961/4.173
5	1	rec	sync	sync	102,400	1,024	1,024	4.725/4.182
6	1	rec	sync	async	102,400	1,024	1,024	4.989/4.236
7	1	rec	async	sync	102,400	1,024	1,024	4.742/4.243
8	1	rec	async	async	102,400	1,024	1,024	4.992/4.306

Recoverable 指的是 Session 是否可恢复。在拥有 1 个 CPU 的网格中，无论在 Windows 还是 Linux 网格中作业调度管理系统的 Task 的反应时间都小于 5ms，而且在 Windows 的响应时间要慢于 Linux。

5.4.2 Session 往返时间测试

Session 往返时间是指只包含一个执行时间为 0s 的 Task 的 Session 往返时间：从建立 Session 到关闭 Session 的时间（包括建立 Session 时间，Task 响应时间和关闭 Session 时间）。测试用例如表 5.6 所示。

表 5.6 Session 往返时间测试用例

Session Round Trip							
目标		Session 往返时间: 50ms					
测试数据		session 往返时间					
主要参数		可恢复性: rec/unrec; 发送方式: sync/async; 接收方式: sync/async;					
固定参数		#CPU: 1; 公用数据大小: 100Kbytes; 输入/输出数据大小: 1Kbytes; Task 实际执行时间: 0 seconds					
用例编号	是否可恢复	发送方式	接收方式	公用数据 (bytes)	输入数据 (bytes)	输出数据 (bytes)	session 往返时间 (ms) Windows/Linux
1	Unrec	sync	sync	102,400	1,024	1,024	14.222/8.961
2	Unrec	sync	async	102,400	1,024	1,024	12.547/8.473
3	Unrec	async	sync	102,400	1,024	1,024	12.403/8.067
4	Unrec	async	async	102,400	1,024	1,024	12.542/11.220
5	Rec	sync	sync	102,400	1,024	1,024	11.886/8.328
6	Rec	sync	async	102,400	1,024	1,024	12.016/8.426
7	Rec	async	sync	102,400	1,024	1,024	11.671/8.705
8	Rec	async	async	102,400	1,024	1,024	11.900/8.464

无论在 Windows 还是 Linux 网格中作业调度管理系统的 Session 的往返时间大大小于 50ms，而且在 Windows 的往返时间要大于 Linux。

5.4.3 Task 发送吞吐量测试

Task 发送吞吐量是指每秒从客户端发送给作业调度管理系统的 Task 数量。它等于发送的任务数/(第一个任务开始发送数据的时间–最后一个任务结束发送数据的时间)。

1. 测试目标

   Task 发送吞吐量测试目标是 1000 tasks / second。

2. 测试条件

   测试条件如表 5.7 所示。

表 5.7 测试条件

Session 数量	1
Session 中 Task 数量	100,000
Task 执行时间	1 sec, 2 sec(当且仅当 2000 个计算节点数量时)
公共数据大小	0Kbytes
输入数据大小	1Kbytes
输出数据大小	1Kbytes
计算节点数量	200(500), 1000, 2000

3. 测试结果

   测试结果如图 5.5 所示。

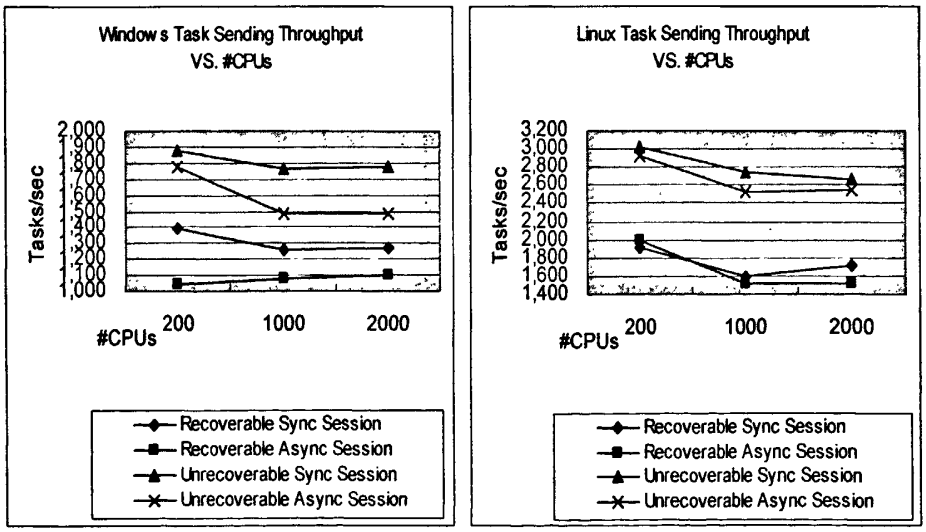

图 5.5 Task 发送吞吐量测试结果

无论在 Windows 还是 Linux 网格中作业调度管理系统的 Task 吞吐量都能够超过 1000 tasks / second，具体分析如下：

1. Task 发送吞吐量受计算节点数量的影响，随着计算节点数量的增加而减少。这主要是因为 Task 的结果返回速度。当有 Task 结果从计算节点返回

时，系统和客户端需要花费时间来处理，并将未处理的 Task 分发到那些闲置的计算节点。这就最终影响到 Task 发送吞吐量。

2. 不可恢复的 Session 比可恢复的 Session 单位时间内可以传送更多的 Task。这主要因为系统需要花时间将数据序列化到磁盘上以备恢复用。
3. 同步 Session 发送 Task 的速度比异步 Session 要快。这主要因为：异步 Session 时，客户端 API 需要为每个 Task 输出结果发送确认给系统。而对于同步 Session，客户端 API 只需要在所有 Task 都被接收后发送一个确认。
4. Linux 的发送吞吐量要好于 Windows 平台。
5. 作业调度管理系统的可伸缩性良好。在 200、1000 和 2000 规模的网格中 Task 发送吞吐量都能保持在 1000 以上。

5.4.4 CPU 利用率测试

CPU 利用率表示的是计算节点中的服务实例从客户端发送计算 Task 开始到全部接收完 Task 返回结果的这个时间段内有多少时间真正用于计算，它可以显示作业调度管理系统分发 Task 到计算节点的效率。它等于（Session 的实际执行时间总和 / Session 的逻辑核数）/（第一个 Task 开始发送数据的时间–最后一个 Task 结束接受数据的时间）。

1. 测试目标

CPU 利用率测试目标是 95%。

2. 测试条件：

测试条件如表 5.8 所示。

表 5.8 测试条件

Session 数量	1
Session 中 Task 数量	100,000
Task 执行时间	1 sec, 2 sec(当且仅当 2000 个计算节点数量时)
公共数据大小	1Kbytes
输入数据大小	1Kbytes
输出数据大小	1Kbytes
计算节点数量	200, 1000, 2000

3. 测试结果

测试结果如图 5.6 所示。

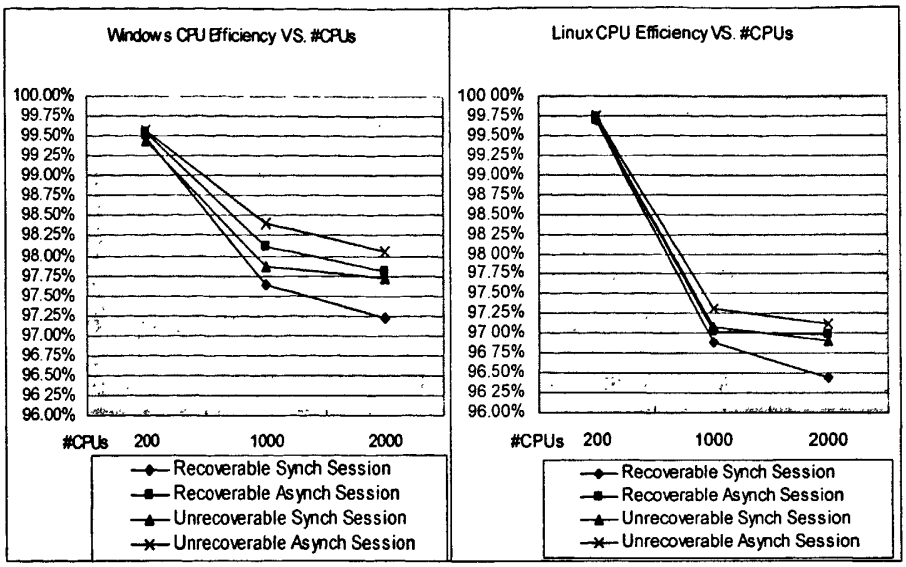

图 5.6 CPU 利用率测试结果图

无论在 Windows 还是 Linux 网格中系统的 CPU 效率都能够达到 96%以上。这就说明调度策略每秒钟能够分发 1000Task 到计算节点。作业调度管理系统的可伸缩性良好。在 200、1000 和 2000 规模的网格中 CPU 效率都能保持在 96%以上。

5.5 本章小结

本章首先给出了系统测试环境，并实现了一个方便测试的系统诊断服务工具，然后描述了作业调度管理系统在功能测试、可靠性测试、性能测试和可伸缩性测试中所使用的测试方法和测试工具，并对测试结果进行了分析。

第六章 结束语

6.1 论文工作的总结

本文以金融领域对网格计算的特定需求为背景，在深入分析和研究网格作业管理技术的基础上，通过对网格作业管理体系架构和SOA的理论研究，提出了一种基于SOA的企业计算网格作业管理体系架构，然后应用该架构设计和实现了网格作业管理中的核心组件，即网格调度管理系统。以下是对本文所做工作的总结和评价：

1. 研究了网格作业管理技术。首先，针对金融领域的应用需求，研究了网格作业管理系统模型，将网格作业管理划分为两个独立的功能模块：资源管理和负载管理，并结合 SOA 对负载管理层进行了结构分析。然后，本文提出了一种基于 SOA 的企业计算网格作业管理系统架构。该架构是一种面向服务的分布式应用的基础架构，它提供了诸如服务生命周期管理、QoS控制、服务组合和容错等方面的服务机制，为上层应用实现了这样一种服务流程：在计算节点创建服务实例，从客户端接收服务请求（即作业）并将这些请求分发到服务，然后再将服务结果返回给客户端，整个流程中网格作业管理系统屏蔽了服务交互的底层处理细节。
2. 基于所提出的网格作业管理体系架构设计并实现了其中的核心组件，即作业调度管理系统。作业调度管理系统包括作业创建、作业调度和作业查询控制等功能。其设计思想主要是根据金融作业的特点，以一组具有相同属性的作业为单位进行资源调度。为了满足对计算实时性的要求，在对作业调度策略研究的基础上，设计了一种改进的基于优先级的调度策略，这个调度策略相当灵活并保证了重要作业的实时响应。在该系统的设计和实现过程当中充分运用了用于并发和网络化对象的技术，并实践了 Strategy，Active Object 等设计模式。
3. 实现了能在分布式环境中测试系统的诊断工具，并利用该工具从可用性、可靠性、性能和可伸缩性等几个方面对系统进行了测试分析。测试表明，系统为金融领域提供了一个面向服务的高性能网格解决方案。

6.2 进一步的工作

本文提出的企业计算网格作业管理系统体系架构以及基于该架构实现的作业调度管理系统还存在一些问题，在很多方面还需要进一步完善，主要体现在以下方面：

1. 做出必要的改进使得此系统体系架构不但能够应用在金融领域，也能够在其它领域得到很好的应用。本文的网格作业管理系统体系架构是针对金融领域的特点设计出来的，行业特征很明显。如果希望扩大它的应用领域，必然要针对应用领域的特点对系统体系架构加以改进。所以需要研究对此企业网格系统架构的所需要的改进，以扩大它的应用领域。
2. 作业调度管理系统在内存中保存的作业数据过大时会耗光内存，这有可能成为系统系能瓶颈。虽然系统在设计时考虑了这个问题，对物理和虚拟内存的使用量进行了控制，设置了一些使用界限，但是这并不能从根本上解决问题，所以要研究能否将数据不经由作业调度管理系统而直接在客户与服务之间传输，使作业调度管理系统只需负责作业的调度。
3. 服务的跟踪和调试工具（分布式调试）研究。服务实例的分布式特性使得对服务的调试变得很困难，经常是服务运行在几千个计算节点的网格里，而只有其中很少一部分出现问题。一般的手工跟踪和调试服务都需要修改代码和配置。因此需要开发一种分布式调试工具，增强对服务的跟踪和调试功能。
4. 网格安全问题的研究。网格安全问题是网格技术中的一个关键问题，而且网格计算中的安全问题比一般的安全问题更复杂。在网格环境下出现了许多新的安全问题，传统的网络安全技术已经不能很好地满足网格安全需求，因此网格安全研究是一个重要、复杂而艰巨的工作。
5. IDE(Integrated Development Environment)工具的研究。IDE 为编程工作提供一站式服务，可以定制开发符合自己框架标准的代码生成框架，避免重复性代码的编写，把更多的精力放在应用逻辑上，有利于减少软件开发周期。

由于时间仓促及作者水平有限，对一些问题未能深入研究，虽经多次修改，错误与欠妥之处仍在所难免，敬请各位评阅老师不吝指正。

致谢

首先，将我最诚挚的谢意献给我的导师郑有才副教授。他渊博的知识，深厚的学术功底和独到的见解给了我莫大的帮助。他严谨的学风、孜孜不倦的钻研和勤奋进取的精神令我终生难忘。感谢郑老师为我创造了良好的学习和科研环境，感谢郑老师在我两年多的求学生涯中指导我如何为人、处事、做学问。

感谢我在公司实习时的同事，他们是：李超、郑颖、钟秋和许强等，他们在我论文的研究工作中给了我很大的帮助，我学到了很多宝贵的品质和精神。

感谢我的同学王文、刘月、史富波、王晓勇、王赞、田舒榕、郭荣侠等，感谢他们在生活中对我的帮助，感谢他们在科研实践中的合作。

感谢我的家人，他们无微不至的的关怀和鼓励，使我能够继续深造。

最后，衷心感谢在百忙之中评阅论文和参加答辩的各位专家、教授！

参考文献

[1] The Large Hadron Collider. http://public.web.cern.ch/public/en/LHC/LHC-en.html.

[2] Ian Foster, Carl Kesselman. The Grid 2 Blueprint for a New Computing Infrastructure[M]. Morgan-Kaufmann, 2004.

[3] Marc Jacobs. Grid in Financial Services: Past, Present and Future. http://www.gridtoday.com/grid/1685961.html, 2007.

[4] LSF. http://www.platform.com/Products/platform-lsf/.

[5] Condor. http://www.cs.wisc.edu/condor/.

[6] PBS. http://www.openpbs.org/.

[7] AHMAR Abbas. Grid Computing: A Practical Guide to Technology and Applications[M]. Charles River Media, 2004:27-112.

[8] 刘鹏. 网格概念的界定. 清华大学计算机系高性能所网格研究组.

[9] Oracle Lead a New Grid Consortium. http://www.eweek.com/article2/0,1759,1570876,00.asp.

[10] 都志辉, 陈渝, 刘鹏编著. 网格计算. 北京：清华大学出版社. 2002.

[11] Web services. http://www.w3.org/2002/ws/.

[12] OGSA-WG. http://www.Gridforum.org/ogsa-wg.

[13] Jack Dongarra, Alexey Lastovetsky. An Overview of Heterogeneous High Performance and Grid Computing. Engineering the Grid: Status and Perspective, 2006.

[14] 王满红, 陈荣华译. SOA概念、技术与设计. 北京:机械工业出版社. 2007.

[15] New To SOA and Web Services. http://www.ibm.com/developerworks/webservices/newto/.

[16] Service-oriented modeling and architecture. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1.

[17] 毛新生著. SOA原理、方法、实践. 北京:电子工业出版社. 2007.

[18] 网格与SOA(Service-Oriented Architecture). http://www.ibm.com/developerworks/cn/grid/zones/webservices/.

[19] 可扩展并行计算技术、结构与编程. 黄铠, 徐志伟著. 北京:机械工业出版社. 2000.

[20] Patrick N.Smith. Client/Server computing. SAMS Publishing. 1994.

[21] Kris Gaj, Tarek El-Ghazawi, Nikitas Alexandridis.Performance Evaluation of

Selected Job Management Systems. 2002.

[22] Maozhen Li, Mark Baker. The Grid Core Technologies. John Wiley & Sons.2005.

[23] Raymond Wu. Integration Strategy and Case Study in Financial Industry. Proceedings of the First Asia International Conference on Modelling & Simulation, 2007.

[24] White paper from Platform and Intel. Enterprise Grid - The Next Generation Architecture for Capital Markets. http://www.platform.com/whitepapers/PltIntel_EntGrid_WP.pdf.

[25] Ian Foster, Carl Kesselman, Jeffrey M. Nick, *et al.* STEVEN Tuecke. The Physiology of the Grid:An Open Grid Services Architecture for Distributed Systems Integration[S]. Global Grid Forum, 2002.

[26] Dirk Krafzig,Karl Banke,Dirk Slama. Enterprise SOA Service-Oriented Architecture Best Practices. Prentice Hall. 2005.

[27] Frank Buschmann, Regine meunier, Hans Rohnert,Peter Sommerlad. Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects. John Wiley & Sons. 2000.

[28] Rajkumar Buyya, David Abramson, and Jonathan Griddy. Grid Resource Management, Scheduling and Computational Economy.

[29] Henri Casanova. Modeling Large-Scale Platforms for the Analysis and the Simulation of Scheduling Strategies. San Diego Supercomputer Center.

作者在读期间的研究成果

1. 张雨，郑有才．网格计算关键技术研究．计算机应用研究．已录用
2. 张雨，郑有才．网格调度模型研究．西安电子科技大学计算机学院 2008 年学术年会．已录用