摘　要

数据流是一种新型的网格应用模式，它需要实时数据供应和处理的协调配合，因此这种应用同时需要多种资源，包括计算资源、网络带宽资源和存储资源。这三种资源是相互关联的，必须达到某种平衡才能提高数据吞吐量并避免资源浪费。

本文设计了一种面向数据流应用的混合资源管理和调度系统，在一个统一的框架内对计算资源、存储资源和带宽资源进行综合的、协调的、按需的调度和分配。面向网格数据流应用的混合资源调度在调度目标、调度算法和实现方式等方面都不同于网格领域现有的许多资源调度方法，为此本文提出了粗粒度和细粒度的资源调度和分配算法，它们分别适用于不同的场景，其主要区别在于对计算资源的控制力度和分配粒度上。

粗粒度的资源调度和分配算法是一种元调度，它不拥有对计算资源的控制权，只能将应用映射到合适的计算资源上。应用能够获得多少计算能力，则要服从于计算资源本地的调度策略。细粒度的资源调度和分配算法利用当前流行的虚拟化方法，为应用建立一个相互隔离的运行环境（即虚拟机），提供可控制、可预测的计算能力。基于模糊控制技术，细粒度算法能够动态的调整虚拟机的计算资源（主要是其获得的物理 CPU 周期）配置，为每个应用提供“恰恰足够”的计算能力。在带宽资源的分配上，这两种调度算法都采用了存储感知的迭代式算法，它可以根据应用的处理速度为各个应用分配带宽，既保证数据供应又避免数据溢出。由于采用了基于模块的数据传输和处理，应用对存储空间的要求大大降低，即数据流应用可以用较小的存储空间实现对较大规模的数据集的处理。

本文实现了一个面向天文应用的数据流使能环境，能够以简单友好的接口为用户提供服务。只需要用户提供一些与具体应用有关的信息，本使能环境就可以自动的完成对应用的封装并为其分配合适的资源，保证应用的顺利运行。

关键词：网格；资源调度；数据流；粗粒度；细粒度

Abstract

Data streaming applications are a novel kind of grid application scenario in that they require the cooperation of real-time data supply and processing. These applications need multiple kinds of resources simultaneously, including compute, bandwidth and storage, which are inherently related and must reach a balance to improve data throughput while avoiding resource waste.

This paper designs a hybrid resource management and scheduling system for the data streaming applications, which schedules and allocates compute, storage and bandwidth in an integrated, concerted and on-demand way. Such a scheduling system is different from the currently available grid resource scheduling implement in scheduling goals and algorithms. The so-called coarse-grained resource scheduling algorithm and a fine-grained one are proposed, and their difference is the application scenarios and allocation granularity of computing resources.

The coarse-grained resource scheduling algorithm is a high level scheduling for it lacks the full control of computing resources and it just maps the applications to the appropriate computing resources. It is up to the local scheduling policy of computing resources to decide the amount of computing capacity a given application can get. The fine-grained resource scheduling algorithm utilizes the popular virtualization technology to establish an isolated running environment (i.e., a virtual machine) for each application to provide the predictable and controllable computing capacity. Based on the fuzzy control technology, the fine-grained resource scheduling algorithm can dynamically adjust the computing resource configuration of a virtual machine, so as to provide just enough computing resources. Both scheduling algorithm adopt an iterative bandwidth allocation which is processing aware and storage aware to guarantee data supply for each application and avoid data overflow. Data is transferred and processed in units of blocks, which alleviate the storage requirement and makes it possible to process high volumes of data with just reasonable storage resources.

An enabling environment is implemented for the astronomic data streaming applications, which provides service with friendly interfaces. With just some application-specific parameters, this environment is able to schedule and allocate appropriate hybrid resources to guarantee the smooth running of applications.

Key words: Grid; Resource Scheduling; Data Streaming; Coarse-Grained; Fine-Grained

目　录

第 1 章　引言.....1
1.1　网格概述.....1
1.2　网格数据流应用概述.....2
1.3　本文的研究背景.....4
1.4　本文的主要贡献.....6
1.5　本文的组织结构.....7
第 2 章　相关研究工作.....9
2.1　网格数据流技术.....9
2.1.1　数据流传输.....9
2.1.2　数据流处理.....10
2.1.3　数据流调度.....11
2.1.4　数据流管理.....13
2.1.5　工作流管理.....13
2.2　网格资源管理和调度概述.....14
2.2.1　网格资源管理和调度的基本概念.....14
2.2.2　网格资源管理和调度的常用算法.....15
2.2.3　网格资源管理和调度系统的现状.....16
2.3　虚拟化技术.....17
2.4　模糊控制.....19
2.5　云计算.....20
2.6　群体智能优化与遗传算法.....21
2.7　小结.....23
第 3 章　网格数据流资源管理和调度系统.....24
3.1　网格数据流资源管理和调度系统的目的和功能.....24
3.2　网格数据流资源管理和调度系统的框架结构.....26
3.2.1　用户接口.....27
3.2.2　调度器.....27

3.2.3　分配器 27
3.2.4　安全服务 28
3.2.5　信息服务 28
3.3　网格数据流资源管理和调度的特点、目标和方法 28
3.3.1　网格数据流资源管理和调度的特点 29
3.3.2　网格数据流资源管理和调度的目标 30
3.3.3　粗粒度资源管理和调度算法 32
3.3.4　细粒度资源管理和调度算法 33
3.4　网格数据流资源管理和调度系统的工作流程 34
3.5　基于模块的并行数据传输和处理 36
3.5.1　基于块的并行数据传输 37
3.5.2　问题描述 37
3.5.3　实验结论 41
3.6　小结 44
第 4 章　粗粒度的数据流资源管理和调度 46
4.1　问题描述 46
4.2　调度参数的产生 47
4.2.1　接纳控制 47
4.2.2　估计/预测 49
4.2.3　优化参数的产生 51
4.3　资源分配 54
4.3.1　计算资源分配 54
4.3.2　存储资源分配 54
4.3.3　带宽资源分配 55
4.4　实验结论与算法性能评估 59
4.4.1　实验设置 61
4.4.2　实验结论 61
4.4.3　算法性能评估 62
4.5　小结 67
第 5 章　细粒度的数据流资源管理和调度 68
5.1　问题描述 68

5.1.1 问题的重要性 69
5.1.2 吞吐量和计算资源利用率的定义 70
5.1.3 技术解决方案 73
5.2 细粒度资源调度和分配 74
5.2.1 虚拟化与 Xen 74
5.2.2 模糊控制器概论 75
5.2.3 语言变量和模糊规则 79
5.2.4 细粒度混合资源调度和分配系统 80
5.3 实验结论 82
5.3.1 系统辨识 82
5.3.2 资源分配和利用 84
5.3.3 系统的反应能力 86
5.3.4 参数收敛性和稳定性 88
5.3.5 与其它分配方法的比较 89
5.4 小结 91
第 6 章 面向天文数据流应用的使能环境 92
6.1 应用对象简介 92
6.1.1 rmon 92
6.1.2 Montage 94
6.2 系统运行环境 95
6.3 系统架构 96
6.3.1 物理架构 96
6.3.2 安全架构 97
6.3.3 资源架构 98
6.3.4 功能架构 100
6.4 系统工作流程 103
6.4.1 需求分析 103
6.4.2 示例实现 105
6.4.3 调度结果 106
6.5 小结 107
第 7 章 结论 108

7.1　研究总结 108
7.2　需进一步开展的工作 110
参考文献 112
致谢与声明 119
个人简历、在学期间发表的学术论文与研究成果 120

主要符号对照表

QoS	服务质量
LIGO	激光干涉引力波天文台
OSG	开放科学网格
RFT	可靠文件传输
TCP	传输控制协议
GSI	网格安全基础结构
VM	虚拟机
CA	证书认证
PIO	群体智能优化
ATS	活跃任务集合
GA	遗传算法
RPS	实际处理速度
TPS	理论处理速度
RTP	实际吞吐量
TTP	理论吞吐量
UC	计算资源利用率
FLC	模糊逻辑控制器
PF	比例系数
NFS	网络文件系统

第 1 章 引言

网格计算是一种新型的分布式的计算模式，其跨管理域异构资源共享的特点为其资源管理和调度提出了全新的挑战，也引起了科学界和工业界的广泛关注和深入研究。作为一种新型的应用，网格数据流应用有其独特的要求，其资源管理和调度又不同于一般的网格资源管理和调度。本章将简要介绍网格和网格数据流应用，并说明本文的研究背景和主要贡献等。

1.1 网格概述

随着现代科学和工程技术的发展，人们对自然、对社会有了越来越多的了解，同时也面临着越来越复杂的问题和挑战，当今的科学和工程需要全行业、全国甚至全世界的大协作。同时，这些问题和挑战也需要极其巨大的计算能力，如生物、能源、交通、气象、水利、农林、教育、环保、国防等对高性能计算的需求量都非常巨大。实际上，对问题求解而言，今天的科学研究已经强烈的依赖于计算机的应用，计算已经成为继理论和实验之后的第三种方法。过去人们往往借助于超级计算机来解决极其复杂的问题，但是以超级计算机为中心的计算模式存在明显的不足，目前正经受着挑战。由于其高昂的购买、使用和维护费用，通常只有一些国家级的部门，如航天、气象、国防等部门才有能力配置这样的设备，大多数单位和个人只能望洋兴叹。人们需要一种造价低廉而能力超强的计算模式，最终科学家们找到了答案，这就是网格计算（Grid Computing）[1]。

网格计算是伴随着互联网迅速发展起来的、专门针对复杂科学计算的新型计算模式，被称为继传统互联网、万维网之后的第三代互联网应用。从本质上讲，网格是一种新型的分布式计算技术，它将充分利用并深度共享互联网的各种资源，包括计算资源、存储资源、通信资源、信息资源、知识资源、专家资源等，实现廉价、普遍的高性能计算，为人们面临的日益复杂的问题和挑战提供解决方法。网格的目标是利用互联网把分散在不同地理位置、属于不同管理域的异构的资源组织成一个“虚拟的超级计算机”，聚合网上的闲置资源，提供超强的计算能力。网格不仅将实现互联网上所有资源的互连互通，还将使其成为信息处理而不仅仅是信息传递的平台。当前的互联网和万维网虽然极大的改

变了人们获取信息的方式，但是还远远没有达到资源共享的目标，互联网上到处是信息孤岛和资源孤岛，大量的资源长期处于闲置状态。由于标准不统一，基于互联网的协同工作也很困难。网格计算为此提供了可行的解决方案。可以说，资源共享是网格的根本特征，消除资源孤岛是网格的奋斗目标。在网格上，人们不仅能共享信息资源（当前的互联网只能局部的、部分的共享信息资源），而且能共享计算资源、存储资源、设备仪器等各种可以通过网络使用的资源。网格的根本特征，不是它的规模，而是资源共享。

网格的本质是服务，网格上提供了各种各样的服务，包括计算服务、数据服务、通信服务、信息服务、教育服务、商务服务、娱乐服务、金融服务、政务服务、旅游服务、医疗服务等。简而言之，网格计算是一种新型的、以服务为导向（Service Oriented）的架构，或者说，网格就是资源一体化和服务一体化：资源一体化是为了提高资源的利用率，服务一体化的目标是提高服务质量。实际上，网格的主要目的是对资源进行虚拟化来解决问题。网格计算要访问的资源主要包括（但不局限于）计算/处理能力、数据存储/文件系统、通信带宽、应用程序软件等。集群、科学设施、网络等都是网格中的重要组件，但是它们本身却并不构成网格。

网格计算与时兴的云计算（Cloud Computing）[2]有着千丝万缕的联系。有人说，云计算是网格计算的自然延伸，它赋予了网格计算更多的内容和技术。华中科技大学的金海教授认为[3]，网格计算和云计算的本质区别在于：网格计算强调的是一个由多机构组成的虚拟组织，多个机构的不同服务器构成一个虚拟组织为用户提供强大的计算资源；而从目前云计算的成熟案例来看，云计算更强调一个机构内部的分布式计算资源的共享。从网格技术引进中国之时起，我国的学者便赋予了它云计算的特征，那就是更加注重资源共享，而不是单纯的高性能计算，我国的教育科研网格[4]就说明了这一点。

1.2 网格数据流应用概述

网格计算已经成为广域条件下分布式高性能计算的重要范式，具有广阔的应用前景。随着一些网格中间件投入使用，逐渐产生了一些新的应用，它们依赖于地理上分布的计算资源、数据和信息服务的无缝交互与合作，这就是数据流应用（Data Streaming Applications），如数据流挖掘[5][6]、大规模仿真[7]、传

感器网格[8]和音频视频数据流[9][10]等。这些应用的一个关键的服务质量（Quality of Service，QoS）要求是对高吞吐量和相关组件之间的低延迟的数据传输的支持。文献[11]描述了一个基于网格的融合（Fusion）仿真工作流，它包含一些相互耦合的仿真代码，同时运行在超级计算中心的不同的高性能计算资源上。在运行时，这个工作流必须与同时运行在远端节点上的其它服务相互合作，这些服务负责交互式数据监控、在线数据分析和可视化、数据存档以及协作。仿真代码产生大量的数据，这些数据必须被高效率的传输到其它分布式的组件上。而且，数据传输本身不能对仿真的运行产生太大的影响，应满足应用和用户对空间和时间的严格要求，并且保证没有数据丢失。

支持交互的组件之间的异步的高吞吐量低延迟的数据传输，是这类应用的最重要的需求。在上述仿真中，需要从高性能计算设备到辅助的数据分析和存储设备连续的传输数据。在数据流应用中，要处理的数据量很大，数据到达的速率是变化的，这带来了很多挑战，在大规模和高度动态的环境里更是如此。网格环境里有共享的计算和通信资源，它们在能力、费用以及应用的行为、需求和性能方面都存在很大的异构性，数据流应用面临着更多挑战。

数据流应用具有以下特点：①从本质上来说，它们是长时间连续运行的；②需要有足够的传输能力，将数据从数据源传送到数据处理终端；③要处理的数据通常不可能完整的存储在处理终端，因为相对于要处理的庞大的数据量，存储总是不足的，况且也没有必要这样做；④它们需要充分利用高性能计算（High Performance Computing，HPC）资源对数据进行高效处理。

数据流应用是将数据传送到处理程序所在的主机加以处理，而不是像其它应用那样，将处理程序运行在数据所在地。尽管由于巨大的通信和网络开销，人们通常避免在网络上传送或复制大量的数据，但是在一些情况下，对一些应用而言，数据流支持却是必不可少的。一个现实的例子就是激光干涉引力波天文台（Laser Interferometer Gravitational-Wave Observatory，LIGO）[12] 项目，它每天产生 TB 数量级的数据，但是天文台本身却缺乏足够的处理能力，只能进行一些简单的在线处理。另一方面，开放科学网格（Open Science Grid，OSG）[13]拥有大量的计算资源，可以对 LIGO 产生的大量数据进行深入的离线处理，以发掘其中可能隐藏的信息。这样，一个自然的想法就是将 LIGO 产生的数据移动到 OSG 上进行处理。但是，OSG 上没有足够的存储能力可以容纳 LIGO 的所有数据。实际上，这既不可能，更非必要，因为人们关心的是 OSG 的计算

资源对 LIGO 的原始数据进行分析后得到的结论，而不是原始数据本身。也就是说，OSG 既没有那样大的存储能力，也没有必要存储所有的数据。于是，可以把 LIGO 的数据分批的传送到 OSG 上进行处理，得到相应的结论后，立即删除被处理过的数据，以便为后来的数据腾出存储空间。另外的一些例子包括基于消息的实时数据流管理和高层服务[14]、大型科学仿真[15][16]等，它们的实时数据都将传送到远端的主机而不是在本地进行进一步的处理和分析。

由此可见，在数据流的应用场景中，数据应该以元组流（Stream of Tuples）的方式源源不断的传送到远端的处理能力所在地，处理程序不断的对这些元组进行处理，就好象元组一直存储在处理能力所在地。在网格数据流应用中，预先存储于数据库或数据文件中的数据以及实时产生的数据，通过网络（一般是 Internet）传输到计算资源所在地的存储或缓冲区中；只要在本地的存储或缓冲区中有足够的数据可以处理，数据处理程序就从中读取适当数量的数据进行处理；处理过的数据被及时删除，以节约存储空间。这样，数据就会被以“流”的方式连续的处理。如果数据供应不足，就会出现断流，导致计算资源闲置；如果处理能力不足，则会导致数据堆积或溢出；如果存储空间或缓冲区过小，则不能容纳足够数量的数据。断流、溢出或容量不足，都会影响数据的处理效率。

1.3 本文的研究背景

本文研究的网格数据流资源管理和调度，最初的应用背景是 LIGO 项目，在此作一简单介绍。需要指出的是，本资源管理和调度系统的应用范围并不局限于具体的 LIGO 应用，它是一个面向一般数据流应用的普适系统。

引力波（Gravitational Wave）是爱因斯坦在广义相对论中提出的，即物体加速运动时给宇宙时空带来的扰动。通俗地说，可以把它想象成水面上物体运动时产生的水波，如图 1.1 所示。但是，只有非常大的天体，如双星体系公转、中子星自转、超新星爆发或两个黑洞相撞时才会发出较容易探测的引力波，而这种情况非常罕见，一般的引力波信号十分微弱。因此，自 1916 年爱因斯坦发表广义相对论、在理论上预言引力波的存在以来，引力波至今未能在实验上被直接探测到，而“水星进动”和“光线偏转”等重要预言则被一一证实。虽然如此，人们探测引力波的努力一直没有中断，引力波天文学就是其中之一。为

此，美国建立了 LIGO。它由两套干涉仪组成，一套安放在路易斯安娜州的李文斯顿（Livingston），另一套在华盛顿州的汉福（Hanford）。在李文斯顿的干涉仪有一对封闭在 1.2 米直径的真空管中的 4 公里长的臂，而在汉福的干涉仪则稍小，只有一对 2 公里长的臂，如图 1.2 所示。LIGO 通过测量两条激光束相遇的时候所形成的干涉图样的变化来探测引力波。这些图样依赖于激光束的传播距离，当引力波穿过时激光束的传播距离会相应变化。这种名为激光干涉计的探测器的灵敏度，是与激光传播的距离成比例的。因为探测器需要寻找的是很微弱的信号，所以需要干涉计的尺寸相当大。

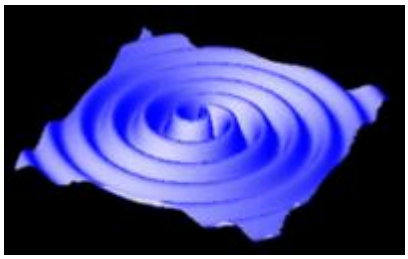


图1.1 引力波想象图

路易斯安娜州李文斯顿的 LIGO　　华盛顿州汉福的 LIGO

图1.2 LIGO鸟瞰图

这两套 LIGO 干涉仪在一起工作构成一个观测所，这是因为激光强度的微小变化、微弱地震和其它干扰都可能看起来像引力波信号。如果是此类干扰信号，其记录将只出现在一台干涉仪中，而真正的引力波信号则会被两台干涉仪同时记录。所以，科学家可以对两个地点所记录的数据进行比较得知哪个信号

是噪声。LIGO 从 2003 年开始收集数据，它是目前全世界最大的、灵敏度最高的引力波探测仪，一系列的升级计划将更进一步提高其灵敏度。

据 2009 年 8 月 20 日出版的《自然》杂志，一个国际科研小组终于锁定了引力波的探测范围，这是人类在寻找引力波过程中“第一次有意义的实验进展”。利用 LIGO，他们成功地锁定了引力波的“出没范围”，显示其能量值比原有推测值要小很多。他们预计，目前探测仪器的灵敏度到 2014 年可提高 1000 倍，到时极有可能直接观测到引力波。

2009 年 9 月 21 日，在匈牙利首都布达佩斯召开的 LIGO 和 Virgo 联合工作会议上，作者的副导师、清华大学信息技术研究院教师曹军威研究员做了 20 分钟的大会报告，详细介绍了清华大学信息技术研究院 LIGO 工作组的研究进展与计划，并被正式接受为 LIGO 科学合作组织（LIGO Scientific Collaboration，LSC）成员，从而使清华大学成为该组织的第一个来自中国的成员。这意味着清华大学信息技术研究院 LIGO 工作组填补了中国在引力波探测数据分析研究领域的空白，跻身于世界一流科学合作组织。

LIGO 的数据处理具有明显的数据流的特征：天文观测的大量数据不断产生，天文台没有足够的处理能力和存储空间，只能对数据进行简单的实时处理；数据通过网络传输到其它地方的计算机集群进行进一步的处理；远端计算机集群上不间断的进行数据处理，以期发现有用的信息；处理过的数据将及时的从计算机集群上删除，以节省存储空间。本文的研究就是着眼于 LIGO 数据异地处理时的资源管理与调度，即为应用分配合适的计算资源、存储资源和带宽资源，保证数据处理的高效执行。这里涉及的数据可能是 LIGO 实时产生的数据，也可能是 LIGO 预先存储在某地的数据。

1.4 本文的主要贡献

本文分析了网格数据流应用的特点及其对资源管理和调度提出的独特要求，提出了面向网格数据流应用的混合资源管理和调度问题，针对不同的应用场景，分别提出了粗粒度和细粒度的资源管理和调度算法，并设计实现了一个网格数据流应用使能环境。

首先，本文提出了面向网格数据流应用的资源管理和调度问题。网格数据流应用同时需要三种资源，即计算资源、带宽资源和存储资源，因而需要一种

混合资源（Hybrid Resources）管理和调度。对网格数据流应用而言，这三种资源必须综合的、协调的、按需的调度，才能保证其顺利运行且使资源得到合理利用。网格数据流应用涉及的三种资源的内在联系，使面向这种应用的资源管理和调度在调度目标、调度算法和实现方式等方面不同于网格领域现有的许多资源调度算法，后者往往主要关注计算资源，而缺乏对存储资源和带宽资源的通盘考虑。

其次，本文设计了一种粗粒度的资源调度算法，主要应用于对资源没有直接控制权的场景中。通常意义上的网格资源都是自治的，它们拥有自己的共享策略，决定着资源以何种方式向网格应用提供服务。在这种情况下，网格资源调度算法应服从于资源本地的策略，粗粒度的资源调度算法就是如此。具体说来，它借助于 Condor[17]将用户的应用映射到合适的计算资源，最终由计算资源本身的策略决定应用可以获得的计算资源（如 CPU 周期）的份额；根据应用运行的速度及存储使用情况，该算法迭代式的为各个应用分配带宽。该算法实现简单，但它受制于资源本地的策略。

再次，本文设计了一种细粒度的资源调度算法。这种算法的应用场景类似于当前流行的云计算，即在一个机构内部的分布式资源的共享。在一个机构内部，由于拥有对资源的完全控制权，可以制定一些更加精细的资源调度策略。具体说来，这种算法为每个应用建立一个虚拟机，并根据应用运行过程中虚拟机的计算资源的利用率，利用模糊控制的方法动态的调整虚拟机的计算资源配置，使其利用率保持在设定的水平，即为虚拟机分配恰恰足够的资源，从而保证计算资源的合理利用。在这种算法中，带宽资源的分配同样是采用迭代式的算法。这种算法实现了按需调度，既可以保证应用的运行效率，又可以避免资源的浪费。

最后，本文建立了面向网格数据流应用的使能环境。该使能环境屏蔽了资源的异构性，降低了资源使用的复杂性。用户只要提交一些简单的、与具体应用相关的信息，如数据源的位置等，该使能环境即可自动为应用分配资源。也就是说，对用户而言，本使能环境是透明的，它在应用和资源之间架起了桥梁。

1.5 本文的组织结构

面向网格数据流的资源管理和调度，是一个理论与实践相结合的课题。与

此相适应，本文中既有理论研究，也有系统实现。具体说来，本文组织结构如下：

第 2 章介绍了一些相关的工作，这些工作是本文理论研究和系统实现的基础，包括网格数据流技术、网格资源调度、虚拟化技术、模糊控制、云计算和群体智能优化算法等。

第 3 章介绍了网格数据流应用资源管理和调度系统的概况，包括其目的和功能、框架结构、特点和要求、工作流程等方面。

第 4 章介绍了一种粗粒度的资源管理和调度算法。它将任务映射到合适的计算资源并服从计算资源本地的调度策略，利用一个迭代式方法为各个应用分配带宽。

第 5 章介绍了一种细粒度的资源管理和调度算法。它以虚拟机为应用执行环境，利用模糊控制的方法动态调整每个虚拟机获得的计算资源。由于它可以实现对计算资源的精确控制和分配，提高了资源分配的灵活性，有利于实现按需分配。此外，本调度算法中对带宽和存储的分配方法与粗粒度调度方法一致。

第 6 章中给出了一个实际的系统实现，介绍了系统的运行环境和架构，并以实例说明了系统的工作流程。

第 7 章对全文进行了总结。

第 2 章　相关研究工作

本文的研究对象是面向网格数据流应用的资源管理和调度，涉及的具体内容包括网格数据流技术、网格资源管理和调度、虚拟化技术、自动控制理论、云计算以及群体智能优化技术等。本章介绍这些领域的相关工作。

2.1　网格数据流技术

网格数据流技术包括数据流传输、数据流处理、数据流调度、数据流管理和工作流管理等。

2.1.1　数据流传输

在网格数据流环境中，要从数据源（如仿真程序、望远镜和其它仪器等）传输大量的数据到远端的计算节点，以进行进一步的分析、处理、可视化等。有时候数据传输与正在执行并产生实时数据的程序是并行的，数据传输应该尽量减小对程序本身的影响。目前，一些网格工具可用来完成数据流传输。

Globus Toolkit[18]是一个广泛使用的网格中间件，提供了一些支持网格数据流应用的组件，其中 GridFTP 和可靠文件传输（Reliable File Transfer，RFT）[19]是用来传输数据的最常用的工具。为了支持在各种存储系统之间快速而有效地传输大量数据，Globus 提出了 GridFTP。它基于标准文件传输协议（File Transfer Protocol，FTP），并增加了一些新的特征，其中一些已经成为标准，包括自动调整传输控制协议（Transmission Control Protocol，TCP）缓冲 / 窗口大小、支持网格安全基础结构（Grid Security Infrastructure，GSI）、第三方控制的数据传输、并行数据传输、条状数据传输、部分文件传输等。第二版的 GridFTP [20]已经可以动态的分配资源，即动态的调整并行度和 TCP 缓冲区大小。 GridFTP 的一些参数，如传输并行度、TCP 缓冲区大小和窗口大小等，都会影响其性能。但是，并行度对数据传输速度影响最为直接。例如，用不同的并行度连续 20 次传输 2GB 的数据，得到的平均传输时间如图 2.1 所示。

GridFTP 有一个缺点，即当客户端失败时，它不知道从哪里重新启动传输，因为所有的传输信息都存储在内存中，所以需要人工重新启动数据传输。为了

解决这个问题，Globus Toolkit 开发了 RFT，它不需要用户的参与。这个服务建筑在 GridFTP 库之上，将所有的传输请求存放在数据库而不是内存中。用户只需要提交传输请求，RFT 将代表用户完成数据传输。当检测到错误时，RFT 从最近的检查点重新启动传输。可以在任何时候检查传输状态，当数据传输完成时，RFT 也可以通知用户。

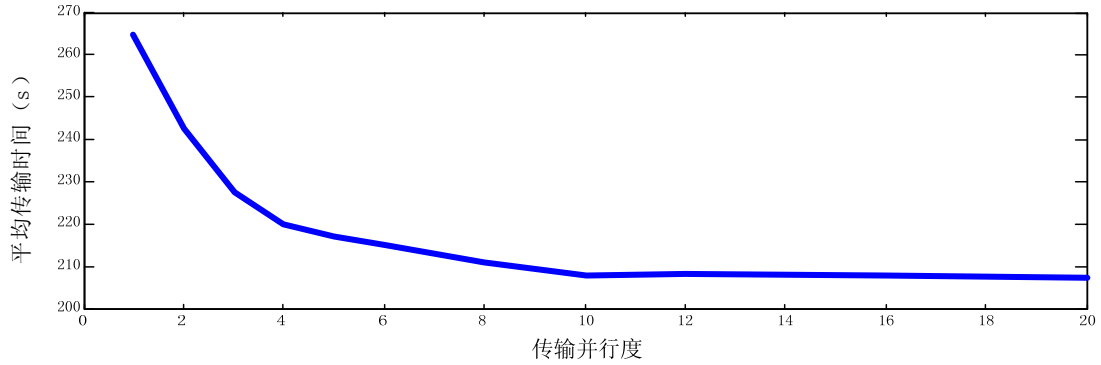


图 2.1 不同并行度对应的数据传输时间

DiskRouter[21]是由威斯康星大学麦迪逊分校（University of Wisconsin Madison，UWM）开发的，用于监控网络缓冲以实现广域的数据移动。它可以动态的建立一个应用级的覆盖网络（Overlay Network），并使用覆盖节点上分层的内存和硬盘缓冲来完成广域数据移动。

2.1.2 数据流处理

数据流是日渐流行的与时间相关的数据源，由于序列是无限的，请求也是连续执行的。在连续查询模型中，查询被部署到网格中的计算节点上，并对流数据进行连续处理。

佐治亚理工学院（Georgia Institute of Technology）的一个研究组开发了一个中间件，名为 dQUOB（Dynamic QUery OBject System 的缩写，即动态查询目标系统）[22]，以应对网格数据流应用的挑战，尤其是客户端从数据源请求数据和/或将大量数据发送到客户端时产生的数据流。这样的应用包括视频点播、支持用于协作研究的远程会议的访问网格（Access Grid）技术、支持科学家们通过数据和计算模型的多种显示方式开展同步研究的分布式科学实验室以及来自数字系统的大规模数据流。

文献[23]开发了一种新的数据管理器，称为网格数据管理器（Grid Data

Manager，GDM），主要面向空间物理应用，特别是 LOFAR/LOIS[24]项目，其目的是利用网格开发一个分布式的软件空间望远镜和雷达。LOFAR/LOIS 的传感器从空间接收信号，将以每秒数兆比特的速度产生大量的原始数据流，这对 GDM 的性能提出了很高的要求。

2.1.3　数据流调度

数据流应用经常分解为彼此交互的多个部分（也称为组件或环节）。这些组件以管道（Pipeline）的方式执行，前面的组件的输出就是后面组件的输入。组件在不同的计算节点上执行需要的时间不同，因为各个节点的 CPU、内存和软件不同。同样，由于网格上的资源是被多个应用共享的，网格节点的负载也是随时间变化的，应用间的竞争使得每个应用的执行效率也是变化的。因此，十分有必要为各个组件制定合适的调度，即将其映射到合适的节点上，以获取最优的整体性能。数据流应用的性能是以吞吐量（即单位时间或给定时间内处理的数据量）评价的。与其它类型的应用不同，在数据流应用中，处理的每个组件都是并行运行的，同时处理连续数据流的快照（Snapshot）。当前已经有一些针对数据流应用的网格资源管理和调度系统。

- E-Condor

文献[25]展示了一个基于 Condor 的简单的数据流调度器，称为 E-Condor，它使用 Condor 获取启动数据流应用的各个环节所必需的资源。在启动之前，E-Condor 会根据应用的数据流图在各个环节建立起必要的耦合。

- GATES

GATES [26][27][28]（Grid-based AdapTive Execution on Streams）是一个中间件系统，面向包含从分布式数据源产生的大量数据流、需要分布式数据处理的应用。它开发了一个基于最小生成树（Minimal Spanning Tree，MST）的资源分配算法，其目标是创建一个部署配置，包括：（1）数据源的数量及其位置；（2）目的地，即需要最终结论的节点；（3）应用中的环节数；（4）每个环节的实例数；（5）这些实例的连接方式；（6）每个环节分配的节点。给定一个由多个环节构成的数据流应用，就决定了一个配置部署的前三个组成部分，资源分配算法的重点是决定后面三个组成部分，以使应用能够获得最好的性能。一旦完成一个部署配置，就会自动调用启动程序以启动整个应用。

- Streamline

Streamline[25][29]是一个启发式调度算法，主要用来适应动态变化的网格环境和数据流应用不断变化的需求。它将最好的资源分配给计算和通信需求最强烈的环节，以此来最大化吞吐量，因此是一种列表调度（List Scheduling）算法。各个环节根据其估计的计算和通信开销排序，开销最小的资源将分配给最前面的环节。在这个算法中，必须提供关于计算和通信开销的准确估计。

- Pegasus

Pegasus [30][31]主要解决自动生成网格工作流的问题，帮助将用应用级术语描述的抽象工作流映射到可用的网格资源上。它以串行的方式先后处理数据传输、任务处理和数据清除。

以上这些项目主要关心计算资源和数据的管理和调度，没有考虑网络资源。有些项目提供了多种网格资源的共同分配，但是它们不是面向数据流应用的。

- EnLIGHTened

由于尽力而为（Best-Effort）的网络总是给高端应用带来性能瓶颈，EnLIGHTened 计算[32]项目被设计用来同时分配多种类型的网格资源：计算机、存储器、仪器以及确定性的高带宽的网络路径，包括光路径。这样一来，外部网络尤其是专用的光网络，可以成为网格环境中的“一等公民”（First Class Citizen），以保证在任务执行时从远端数据源传输大量数据。其基本思想与本论文中的混合资源调度相似，但本文实现了对资源调度的更多控制，以满足数据流应用的特殊需求，如存储感知（Storage Aware）的数据供应。

- G-lambda

G-lambda[33]项目在网格资源管理者和网络资源服务之间定义了一个标准的 Web 服务接口。一个网格资源调度系统使用网格资源调度器，通过 Web 服务接口，以预留的方式协同分配计算资源和网络资源。同样的，这是一个通用的资源分配框架，没有充分注意到数据流应用的特点，如可持续、可控制的数据供应。

- Phosphorus

Phosphorus [34]力图在多个域间使能按需的点到点的网络服务。Phosphorus 的网络概念和实验床使应用能够感知其所在的整体网格资源（计算和网络）环境和能力，可以动态的、自适应的、优化的使用连接高端资源的异构网络。

尽管看上去 EnLIGHTened、G-lambda 和 Phosphorus 试图解决与本文一样的问题，但它们将重点放在了网络资源上。另一方面，本文的资源调度和分配算

法是一个综合的算法，其中计算资源、带宽资源和存储资源是统一管理的。至于网络资源，它们拥有对光网络的完全控制，所以可以预先或按需获得一条确定性的光路径。然而本研究中的网络是公共的互联网，基于尽力而为的 TCP/IP 协议，只能通过调整相关协议，如 GridFTP 的特定参数来尽量逼近需要的带宽。

2.1.4　数据流管理

为了应对数据流分析和处理的挑战，目前已经开发了一些数据流管理系统（Data Stream Management System，DSMS），简单总结如下。

斯坦福大学（Stanford University）的研究者开发了一个通用的数据流管理系统，称为 STREAM （STanford stREam dAta Manager）[35], 对多个连续数据流进行连续查询处理。STREAM 包含几个组件：输入数据流产生无限的数据并驱动查询处理；Scratch Store 处理连续的查询，特别是中间状态；存档器保存数据并可能对分析或挖掘请求进行离线处理；连续查询[36]一直持续运行，直到被明确注销。连续查询的结果一般将作为输出数据流传输，但它们也可能是关系型结果，随时间更新。

Aurora[37]用来更好的支持监控应用，主要处理不精确数据组成的数据流，通常有实时性要求。它假定数据来自一系列数据源，如计算机程序、传感器和仪器设备等。Aurora 的基本任务是根据应用管理者定义的方法，处理输入的数据流。因此，数据元组流过处理操作的有向无环图，最终输出的数据流返还给应用。Aurora 也可以保存历史数据，大多是为了支持 ad-hoc 查询。

数据流管理的其它项目包括 Gigascope[38]和 TelegraphCQ[39]等。

2.1.5　工作流管理

高效鲁棒的数据流应用经常由一些组件构成，这些组件经常是由地理上分布在不同计算节点的不同研究者设计和调试的，以保证运行时无缝交互、耦合和数据依赖。这些应用的一个关键服务质量要求是对相关分布组件的高吞吐量低延迟数据流的支持。

通过定义组件之间的交互和数据依赖关系，科学工作流已经被用于将单独的应用联合为大规模的分析。数据流应用以及由此产生的工作流的规模，需要大量的计算、存储和数据资源以产生需要的结论。赛百[40]（Cyberinfrastructure）项目如 TeraGrid [41]和 OSG 能够提供工作流的运行平台，但是这需要终端用户具

有丰富的经验，才能有效的使用这些资源，所以它们对终端用户并不合适。

Pegasus[42]是一个工作流映射引擎，用于物理、天文、引力波科学、地震科学、纳米科学等方面的项目。Pegasus 依赖于 Condor DAGMan 工作流引擎，通过将高层工作流描述映射到分布式的资源并保持其依赖关系，在科学领域和执行环境之间架起桥梁。Pegasus 隐藏了网格资源的复杂性，允许科学家以抽象的方式创建工作流，这些工作流将被翻译并映射到底层的资源。Pegasus 可以将包含数以千计的任务、处理大量数据的复杂大规模的科学数据流映射到网格上，并自动的管理工作流执行期间产生的数据，如将它们传输到用户指定的位置、注册到数据目录中、发现蕴含的信息，这对网格数据流应用是非常可贵的。

由于数据流的特点，特别是其数据的大规模以及由此而来的对大规模的存储空间的要求，典型的工作流映射技术有可能产生不能执行的工作流[31]。当前，Pegasus 自动的产生清除动作，但它是在整个工作流完成且分析结论保存到用户指定的地方后，才删除所有输入的数据和计算节点产生的中间数据。也就是说，这是一个静态的清除过程，引入了可观的开销，因为有些数据处理的一次运行就可能需要很长时间，仍然会占用大量的存储空间。

2.2　网格资源管理和调度概述

作为网格研究和应用领域的一个热点和难点问题，资源管理和调度引起了科学界和工业界的高度重视，目前已经出现了一些比较成熟的管理和调度算法以及实际实现的系统，本节将对此进行综述。

2.2.1　网格资源管理和调度的基本概念

资源管理是网格系统最基本的核心功能之一[43]，它负责接受用户的请求，然后从网格资源池中选择合适的资源分配给请求者。在网格领域，节点（Node 或 Site）是由一个或多个资源组成的自治实体。资源是指用以满足作业请求的可复用实体，例如处理器、数据存储设备以及网络连接等。作业（Job）由一系列原子任务组成，具有一系列属性，如要完成的任务及其参量、资源 QoS 需求描述和有关资源选择的提示信息等；作业的执行有赖于某个资源集合。任务（Task）是最小的调度单元，其属性由反映任务需求的多个参数组成，例如 CPU/内存需求、完成任务的截止期限和优先级等。资源管理的流程一般包括四个步骤：作

业请求、资源发现、资源选择以及作业执行和监控等。任务调度（Task Scheduling），也被称为资源调度、资源选择或分配，是指通过优选，根据任务和资源的要求，按照一定的原则，建立从任务集合到资源集合的映射，即为任务选择合适的资源，或为资源选择合适的任务。网格调度包括 4 个主要阶段：资源发现、资源选择、调度产生和作业执行[44]。任务调度通常是一个优化问题，其算法通常包含目标函数和选择/搜索过程两部分。目标函数是任务调度的优化目标，从资源消费者（用户）角度看，目标函数一般包括总时间跨度（Makespan）、经济代价和其它 QoS 要求；从资源提供者的角度看，目标函数包括资源利用情况（例如吞吐量、资源利用率和负载均衡等）和经济效益等。选择/搜索过程是指在各种可选的任务－资源映射方案中进行选择和搜索，以便得到在目标函数度量下最优的解决方案。

资源管理和调度结构可以分为三种，分别为集中式、分布式式和层次式[45]。在集中式调度环境中，一个中央机器（节点）扮演着资源管理者的角色，负责接收用户提交的作业并由它将这些作业分配给适当的节点。由于中央调度者拥有关于可用资源的所有必要的、最新的信息，这种调度方式能够给出更好的调度策略。然而，这种结构容易导致单点失效而使整个环境不能正常工作。分布式调度中有多个本地调度者，它们可以通过直接通信或间接通信彼此交互以完成作业分配。分布式调度易于扩展，且提高了系统的容错能力和可靠性，但由于全局信息的匮乏，这种调度通常导致不够理想的调度决策。在层次式调度中，中央调度者是一个元调度者，它负责将已经提交的作业分派到本地调度者，由本地调度者完成任务的最终分配。这种调度方式的优点在于全局调度者和本地调度者对于调度作业可以有不同的策略，实际上，中央调度者要服从于本地调度者的策略。

2.2.2　网格资源管理和调度的常用算法

当前，网格资源调度关注的对象主要是计算资源，其优化目标大多是与时间相关的。根据任务到达时间和执行时间的准确程度，资源调度的策略可以分为静态调度和动态调度[46]。静态调度也叫确定性调度，它要求精确的知道任务的到达时间和执行时间，以便为应用指定计算资源，并合理安排应用执行开始时间和结束时间。一般来讲，这是一个 NP-难问题[47]，人们往往采用启发式的算法。这方面比较成熟的调度算法包括最短完成时间（Minimum Completion Time，

MCT）算法、最短执行时间（Minimum Execution Time，MET）算法、机会负载均衡（Opportunitistic Load Balancing，OLB）算法、最长处理时间优先（Longest Processing Time First，LPTF）算法、最短处理时间优先（Shortest Processing Time First，SPTF）算法、Min-Min 和 Max-Min 算法、Sufferage 算法等。动态调度也称为非确定性调度，要求任务的到达服从某种随机过程（如泊松过程），任务的执行时间满足某种随机分布（如指数分布）。动态调度的策略包括随机调度（Random Scheduling）、最大处理能力优先（Most Processing Power First，MPPF）、最小负载优先（Least Load First，LLF）和最小响应时间优先等（Minimum Response Time First，MRTF）。

人们还将经济学中的有关模型引入网格资源调度领域，提出了基于经济模型的资源调度机制。一般说来，这些模型包括招标/合同（Tendering/Contract）模型、议价（Bargaining）模型、商品市场（Commodity Market）模型、拍卖（Auction）模型，反映了不同的经济规则在网格资源调度中的作用。

2.2.3　网格资源管理和调度系统的现状

网格资源管理和调度系统最大的特点是虚拟化和协同使用广域分布和异构的资源，主要解决资源的定位、分配、认证、进程创建以及其它使用资源的准备活动。目前常用的网格资源管理系统包括 Globus Tookit 和 Condor 等。

Globus Tookit 是一个构筑网格计算环境的中间件，提供基本的安全架构、资源管理、数据管理、通信、错误检测等服务。该工具包是模块化的，允许用户按自己的需要定制环境。可以说，GlobusTookit 提供了网格基础设施的大部分内容。作为应用和资源之间的桥梁，Globus 资源管理结构包括中介者、协同分配器以及 Globus 资源分配和管理器（Globus Resource Allocation and Management，GRAM）几个部分。GRAM 主要由门卫（Gatekeeper）和作业管理器两部分构成。门卫响应用户的请求，负责用户认证，把远程用户映射到本地的一个安全环境中，为其建立本地的资源容器（Resource Container），使本地资源开始为其服务。作业管理器是门卫的一个服务，可直接与本地资源管理系统连接，处理与作业的远程交互。

然而，Globus Tookit 并没有包括调度组件。因此，需要选择一个调度组件来增强 Globus 的基础设施，并添加一些调度策略来最大化作业的吞吐量且提高资源的利用率。调度器可以确定在何时何地运行某个作业，然后与 Globus 进行

协调，从而执行相关的认证、文件划分、监视以及对远程执行环境的控制等任务。

Condor 是 UWM 研制的一个用来管理计算密集型任务的批处理队列系统，主要功能是利用位于工作站池中的空闲资源来实现高吞吐量计算（High Throughput Computing，HTC）。Condor 可以管理集群中的专用计算节点和有效地利用集群中未用的时钟周期，它可以自动定位工作站池中的空闲机来处理用户提交的任务。在为这些任务提供高吞吐量的同时，Condor 还可以有效的利用所有的可用资源。它不仅提供了传统的队列和调度功能，还可以对资源进行分类，以便为用户任务找到合适的资源，这是通过 Condor 的 ClassAds 和匹配（Matchmaking）方面的创新机制实现的。与其它批处理队列系统一样，提交给 Condor 的作业应该作为无人参与的后台任务运行。

也就是说，Globus Toolkit 提供了一个网格基础设施，在此基础上增加 Condor 的作业提交、管理和控制等特性，就可以扩展到 Condor 池之外的很多资源管理和调度系统所控制的资源，这些资源管理和调度系统包括 Sun 网格引擎（Sun Grid Engine，SGE）[48]、便携式批处理系统（Portable Batch System，PBS）[49]和负载共享软件（Load Sharing Facility，LSF）[50]等。此时，Globus 就是中央调度者，而 Condor、SGE、PBS 和 LSF 则是本地调度者，它们通常运行在同一个管理域中。结合使用 Condor 的特性与 Globus 控制的资源一起进行作业提交、监视和控制就称为 Condor-G[51]。Condor-G 使用 Condor 的作业管理特性，以及 Globus Toolkit 的安全性和资源访问特性。还有一些为面向批处理的任务提供支持的元调度器，如 Legion[52]和 Nimrod/G[53]。

在网格这样的复杂系统中，人们往往借助于仿真的方法对不同的作业调度算法的性能进行分析和比较。目前，已经涌现出一些仿真工具，主要包括 Bricks[54]、MicroGrid[55]、SimGrid[56]、GridSim[57]、Optor-Simt[58]和 ChicSim[59]。

2.3　虚拟化技术

本文提出并实现的细粒度的资源管理和调度方法中，利用虚拟化（Virtualization）技术为每一个应用建立了独享的运行环境，并通过模糊控制的方法精确的分配资源，以便提高应用的运行效率和资源的有效利用率。实际上，虚拟化技术已经在网格计算中得到应用[60][61][62]，有一些学者研究如何自动化的

部署和控制各种虚拟化的应用[63]−[68]。本节和下一小节分别对虚拟化技术和模糊控制加以介绍。

虚拟化提供了一个抽象层，将物理硬件与操作系统分开，从而提供更高的信息技术（Information Technology，IT）资源利用率和灵活性。虚拟化技术可以扩大硬件的容量，简化软件的重新配置过程。虚拟化允许具有不同操作系统的多个虚拟机（Virtual Machine，VM）在同一物理机上并行运行，并且应用程序都可以在相互独立的空间内运行而互不影响，从而显著提高计算机的工作效率。每个虚拟机都有自己的一套虚拟硬件（例如 RAM、CPU、网卡等），可以在这些硬件中加载操作系统和应用程序。无论采用了什么物理硬件组件，操作系统都将它们视为一组一致的、标准化的硬件。

当前，虚拟化技术得到了十分广泛的应用，主要关注于服务器的虚拟化，或在单个主机上寄存多个独立的操作系统。服务器或计算机的虚拟化使单个服务器或计算机看起来像多个计算机或完全不同的计算机。另一方面，虚拟化技术也可以使多台计算机看起来像一台计算机，这叫做服务器聚合（Server Aggregation）。对于业界来说，虚拟化不仅提升了服务器的利用率，因为它能够把一台服务器划分成多台虚拟机来运行应用，完成服务器上的应用整合；更重要的是，虚拟化使得云计算或者网格内的资源调度和分配真正实现了灵活性和按需分配。有了虚拟机之后，就可以解决网格资源的自由调度问题。本文正是利用虚拟化带来的灵活性，实现了细粒度的网格资源按需调度和分配。

虚拟机封装在文件中，因此可以快速对其进行保存、复制和部署，可以实现虚拟机的动态迁移（Live Migration），即可将整个系统（完全配置的应用程序、操作系统、BIOS 和虚拟硬件等）从一台物理服务器移至另一台物理服务器，以实现零停机维护和连续的工作负载整合。因此，有了虚拟化技术，不仅可以移动应用，还可以移动系统资源、系统环境。资源一旦能够移动，就立刻改变了整个数据中心的生态环境。此外，虚拟化可以带来分区、隔离等多方面的好处。

本文采用的是半虚拟化（Para-Virtualization）技术，它使用虚拟机管理程序（Hypervisor）分享存取底层的硬件，但是它的客户操作系统集成了虚拟化方面的代码，从而无需重新编译，也不会引起陷阱，因为操作系统自身能够与虚拟进程进行很好的协作。半虚拟化需要客户操作系统做一些修改以配合虚拟机管理程序，这是一个不足之处，但是它提供了与原始系统相近的性能，这又是它

的突出优点。就具体的开发技术而言，本文采用了 Linux 下的 Xen[69]。它由 XenSource 开发，是一种开源免费的操作系统级半虚拟化技术。在 Xen 环境下，可以动态调整虚拟机的内存与 CPU 分配，以优化虚拟机的性能。例如，通过设置 Weight 和 Cap 参数值，就可以调整 CPU 的优先级，从而决定一个虚拟机可能获得的实际的 CPU 时钟周期。为了动态的按需配置虚拟机的计算资源，本文采用了模糊控制的方法，详见第 5 章。

2.4　模糊控制

自从美国科学家维纳（N. Wiener）在 20 世纪 40 年代创立控制论以来，这种理论已经在各个领域获得了广泛的应用，其中也包括计算系统，文献[70]的第一章对此有详细的总结。控制理论中的许多典型方法在计算系统中得到了应用，如比例积分微分控制[71][72]、极点配置[73]、线性二次型调节器[73]以及自适应控制[74][75]等。

自动控制理论经历了经典控制和现代控制两个重大发展阶段，已经比较成熟。但是，许多复杂庞大的实际系统，很难建立有效的数学模型，传统的方法很难对这些系统实施有效的控制，需要借助于一些新兴的控制方法，如智能控制等。智能控制是由人工智能、控制论和运筹学等形成的交叉学科，它通过定性和定量相结合的方法，针对被控对象和控制任务的复杂性、不确定性和多变特性，有效自主的实现繁杂信息的处理、优化、判断乃至决策，实现对被控对象的控制。

模糊控制[76]是智能控制中应用比较广泛的一个分支，是基于模糊集合论、模糊逻辑推理并同经典控制理论相结合，用以模拟人类思维方式的一种控制方法。模糊控制的核心是模糊规则和模糊逻辑推理，其主要研究内容是使机器“识别、理解”模糊规则并进行模糊逻辑推理，最终得出新的结论并实现自动控制。模糊规则是由许多“若……则……”（IF-THEN）形式的模糊条件判断语句组成的，是人们的操作经验（或者说是人类智能）的反映。模糊逻辑推理是在二值逻辑基础上发展起来的一种不确定性推理方法，它以一些模糊判断为前提，经过一系列运算，最终推出新的模糊结论。典型的模糊逻辑系统由模糊器、模糊规则库、模糊推理机、去模糊器四部分组成，如图 2.2 所示。

模糊器和去模糊器（有时也称为清晰器）是模糊逻辑系统的输入和输出单

元，分别将论域 U 上的确定值 u 变换为论域 U 上的模糊集合以及将论域 V 上的模糊集合转换为论域 V 上的确定值 v；模糊推理机根据模糊规则库中的知识按一定的推理机制产生模糊结论，即获得论域 V 上的模糊集合。模糊规则库是模糊逻辑系统的核心，它由若干模糊规则构成，是设计控制系统的主要内容。

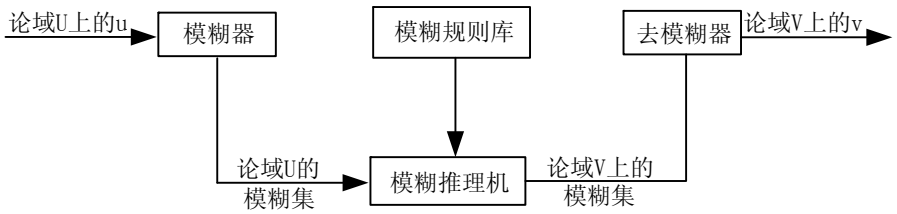


图2.2 模糊逻辑系统框图

1974 年曼达尼（E. Mamdani）等首次将模糊控制成功的应用于小型蒸汽机的控制[77]。此后，模糊控制的应用得到广泛的发展和推广。模糊控制也是计算系统的一个研究热点[78][79]，但主要用于接纳控制，以获得更好的服务质量。自适应模糊控制被用于周期性任务的利用率管理[80]，其中利用率定义为估计执行时间与任务周期的比率：基于模糊规则的推理用来决定一个阈值，一旦超过这个阈值，任务的服务质量就会降低，甚至会根据接纳控制拒绝一些任务。这里要求估计任务的执行时间，但这往往难以做到。而且在一些情况下，服务质量不能降低，因为有些任务不能分解为更小的任务，这限制了这种方法的广泛应用。

2.5　云计算

本文中提出的细粒度的资源调度和分配方法，已经具有云计算的特征，因此这里对云计算进行简要介绍。所谓的云计算，就是“随需应变”的延伸，即厂商按照用户不断变化的需求提供相应的硬件、软件和服务。鉴于互联网的迅速发展，与“随需应变”不同的是，用户需求的满足是通过互联网实现的。用户不再需要关心如何根据自己的业务需求来购买服务器、软件和解决方案，只要根据自己的需要，通过互联网来购买自己需要的计算资源和服务。由单个公司生产和运营的私人电脑系统，正在被中央数据处理工厂通过互联网提供的服务所取代。

云计算是并行计算、分布式计算和网格计算的发展，或者说是这些计算机

科学概念的商业实现。云计算是虚拟化、效用计算（Utility Computing）、基础设施即服务（Infrastructure as a Service，IaaS）、平台即服务（Platform as a Service，PaaS）、软件即服务（Software as a Service，SaaS）等概念混合演进并跃升的结果。在未来的应用场景中，云计算将呈现为具有中央运算形态的公共服务，就象今天的电力应用。云计算是网格计算的自然延伸，它赋予了网格计算更多的内容和技术，但本质上它和网格计算一样都是分布式计算的一种。与网格计算强调的由多机构组成的虚拟组织不同，云计算更强调在一个机构内部的分布式计算资源的共享。也正是由于这样的一个机构来提供资源，才能保证用户的服务质量。

一些机构声称他们可以按需提供计算资源，如亚马逊的弹性云计算（Elastic Cloud Computing，EC2）[82]。但实际上，他们只是提供粗粒度的计算资源。终端用户很难估计他们的计算需求，所以他们往往会多申请资源，这会造成资源的低效利用并浪费用户预算。在这种情况下，需要有一种机制，在不需要用户参与的情况下，能够为用户分配恰当数量的资源，既保证任务的运行，又避免资源的浪费。本文提出了模糊控制的机制，如第 5 章所述。

2.6　群体智能优化与遗传算法

优化技术以数学为基础，寻求各种工程问题的最优解或满意解。美国工程院院士、哈佛大学教授、清华大学自动化系智能与网络化系统研究中心讲习教授组教授何毓琦（Yu-Chi Ho）指出：“任何控制与决策问题本质上均可归结为优化问题”。

为了解决实际工程中面临的优化问题，人们提出了许多经典的优化方法，如牛顿法、共轭梯度法、爬山法等。这些方法依赖于目标函数的梯度或高阶导数以产生一个确定的计算序列，指明下一步的搜索方向。这些方法对目标函数的性质提出了较为严格的要求，且只有当问题具有凸性时才能找到全局最优解，否则这种点对点的搜索极有可能陷入局部最优解。对于解空间为有限集合的情形，可以使用枚举法对解空间中的所有点进行比较并找出最优解。这种方法最为简单，但计算量最大。动态规划就是典型的枚举法。

由于实际工程中的优化问题越来越复杂，如问题规模大、非线性度高、多极小、多目标、机理不明难以建模等，人们认识到常规的优化方法难以解决这

些问题[83][84]，需要寻求一些具有智能特性的、自组织、自适应的大规模并行算法，群体智能优化（Population-Based Intelligent Optimization，PIO）算法[85]就是其中之一。文献[86]给出了 PIO 算法的统一框架的数学化描述，其基本环节包括社会协作、自我适应、竞争等。PIO 算法的具体实例包括微粒群优化（Particle Swarm Optimization，PSO）[86]、蚁群优化（Ant Colony Optimization，ACO）[87][88]、进化计算（Evolutionary Computation，EC）[89]和遗传算法（Genetic Algorithm，GA）[90][91][92]等，它们分别是对自然界中不同生物群体行为或生命现象的模拟。

在本文第 4 章中，需要优化带宽分配的有关参数，这是一个非凸的非线性问题，其中采用的优化方法就是遗传算法，因此在此对遗传算法进行简要介绍。遗传算法是基于进化论的原理发展起来的一类自组织、自适应的高效的随机搜索与优化方法，它将生物进化的原理（如遗传、进化、竞争等）与最优化技术和计算机技术结合起来，在工程优化中得到了广泛应用。

遗传算法的常用形式是 Goldberg 提出的[93]。不同于传统的从单点开始的搜索，遗传算法从一组随机产生的初始解（称为种群，Population）开始搜索，种群中的每个个体都是问题的一个可行解，称为染色体（Chromosome）。在实际运算中，染色体是一串以二进制等方式编码的符号，其中的每一位代表一个基因（Gene），它们决定了染色体的性状，也称为适应值（Fitness）。染色体经过遗传，包括交叉（Crossover）和变异（Mutation）产生后代（Offspring）。这些后代可能被保留下来继续参与运算，也可能被淘汰。适应值高的染色体的后代被保留的概率较高，这就是选择（Selection）操作。经过若干代之后，算法收敛于最好的个体，它将被作为优化问题的解，它有可能是全局最优解，也可能是一个次优解（满意解）。

与传统的优化算法相比，遗传算法最大的优点在于它对目标函数基本没有限制，即它不要求目标函数连续，也不要求目标函数可微。此外，目标函数可以是显函数，也可以是隐函数，从而扩大了遗传算法的应用范围。从本质上讲，遗传算法是一种启发式的导向随机（不是盲目随机）搜索法，它从一个群体出发，以一定的概率改变当前的搜索方向，以在其它方向上进行搜索，因而遗传算法可以防止搜索过程收敛于局部最优解，有可能获得全局最优解。只要染色体编码合适，遗传和进化操作得当，遗传算法的搜索效率往往很高，仅仅经过有限次迭代便可接近最优的解。

2.7　小结

本章简要介绍了网格数据流的相关技术、网格资源管理和调度的相关算法以及面向网格数据流应用的资源管理和调度中涉及的相关技术，如虚拟化和模糊控制等。可见，网格领域中现有的大多数资源调度方法的优化目标都是与时间相关联的，如最短完成时间、最短平均执行时间等。这些方法要求预先知道任务的执行时间和达到时间或其概率分布，而数据流应用往往没有预期的执行结束时间，所以这些调度方法不适合于数据流应用。此外，面向网格数据流应用的资源管理和调度系统用到了虚拟化、模糊控制等特定技术，具有当前流行的云计算的一些基本特征。本章还介绍了群体智能优化技术和遗传算法，它将用于本文中带宽分配的参数寻优。

第 3 章　网格数据流资源管理和调度系统

网格资源具有异构性、动态性、自治性、二分性等特点，需要有效的管理机制，以透明的、可靠的、便捷的方式提供给用户。网格资源管理并不关心资源的具体功能（即资源能够为用户做什么），而是控制网格资源怎样向用户、应用或服务在内的其它实体提供可用能力的一系列操作，即资源的功能执行方式，如被请求的操作何时开始执行，或者它需要多长时间完成等。具体到网格数据流应用，资源管理需要解决的问题是如何协调、按需分配计算资源、带宽资源和存储资源，以及如何控制数据传输和处理的协调运行。用户提交的应用往往只涉及数据处理，很少考虑数据传输的问题。而对数据流应用而言，数据传输和处理必须并行进行，否则就会导致效率低下或资源浪费。因此，面向数据流应用的资源管理系统，必须同时考虑多种资源的分配以及数据传输和处理的同步。

本章介绍面向网格数据流应用的资源管理和调度系统的基本功能、框架结构、调度方法和工作流程等。另外，在本系统中，数据都是基于块（Block-Based）并行（Concurrent）传输和处理，本章一并进行了介绍。

3.1　网格数据流资源管理和调度系统的目的和功能

网格数据流资源管理和调度的任务就是管理和调度网格中的各种资源（包括计算资源、带宽资源和存储资源），以透明的方式向资源请求者提供服务。多个应用可能共享使用网格中的同一个资源，资源请求者也可以根据业务需要同时或先后使用网格中的多个资源。从用户的角度讲，网格数据流管理和调度系统可以接受用户提交的任务，并为其分配合理的资源，保证其顺利运行。从资源的角度讲，网格数据流管理和调度系统能够对资源进行有效的管理和调度，能够提高资源的利用率（Utilization）。

与其它应用不同，网格数据流应用中，数据传输和处理是并行进行的，因而一个数据流应用同时需要计算资源、带宽资源和存储资源，相应的管理和调度系统需要同时对这三种资源进行管理和调度，所以这是一个混合资源管理和调度系统。当前大多数网格资源管理和调度系统往往只侧重于单一资源的管理

和分配，关注的重点主要是计算资源，对网络带宽资源和存储资源等则很少涉及，这是不能满足网格数据流应用的要求的。本文设计开发的混合资源管理和调度系统，则是针对数据流应用的特点和要求，同时协调的管理和调度多种资源。这是本系统最大的特色，也是本文的主要创新点。

数据流资源管理和调度的目的主要是：

（1）方便用户访问资源。管理和调度系统隐藏资源实际使用的复杂技术细节，将物理资源抽象为逻辑资源，并提供简单的接口，主要是命令行（Command Line）工具，方便用户请求资源。在数据流资源管理和调度系统中，资源请求者只需要提供简单的参数，如数据源的位置、数据流的数量、数据处理的可执行程序、处理结果的返回路径等，就可以获得所需要的资源并保证数据传输和处理的自动运行，而不必关心其它复杂的技术细节。

（2）协调资源的共享使用。网格中的资源是由多个应用共享使用的，资源管理和调度系统根据资源本身的特性和拥有者制定的策略，制定相应的策略决定多个请求者如何使用同一个资源。在本文的粗粒度的资源管理和调度中采用的是基于优先级的带有回填（Backfilling）机制的排队策略，计算资源的具体分配是通过 Condor 实现的，而带宽分配则是利用迭代式算法实现的；在细粒度的资源管理和调度中，虚拟机的使用可以保证由一个应用“独占”一部分资源。同时，资源管理和调度系统还应支持一个请求者请求使用多个资源的需要，如在工作流处理中，一个应用的多个环节要同时使用多个资源，资源管理和调度系统要保证这些资源之间的协调和配合，既要避免性能瓶颈的产生，又要避免不必要的资源浪费。

（3）建立安全的网格资源使用机制。安全是资源提供者和使用者最关心的问题，也是资源管理和调度系统首先要解决的问题。本资源管理和调度系统采用了 Globus 的 SimpleCA 安全机制，建立了证书认证（Certification Authentification，CA）中心，负责为各个主机和用户签发证书，建立相互信任的关系，并将资源请求者映射为具有受控权限的本地用户，从而可以限定资源请求者可以执行的操作，由此保证资源的安全。

（4）为用户建立资源容器。在用户请求使用资源时，资源管理和调度系统为该用户在资源本地建立一个进行活动的场所——用户资源容器，用户在容器内使用资源。在数据流应用中，资源容器同时包含计算资源、带宽资源和存储资源，其重要功能就是确保三种资源的协调配合。

此外，网格数据流资源管理和调度系统还具有一般的资源管理和调度系统所具有的通用功能[94]，可以管理资源的整个生命周期，即资源的注册、共享到注销的整个过程，其基本功能还包括资源注册、资源发现、资源部署、资源代理和资源注销等。

3.2　网格数据流资源管理和调度系统的框架结构

网格数据流资源管理和调度系统中应该包含以下几个部分，如图 3.1 所示：

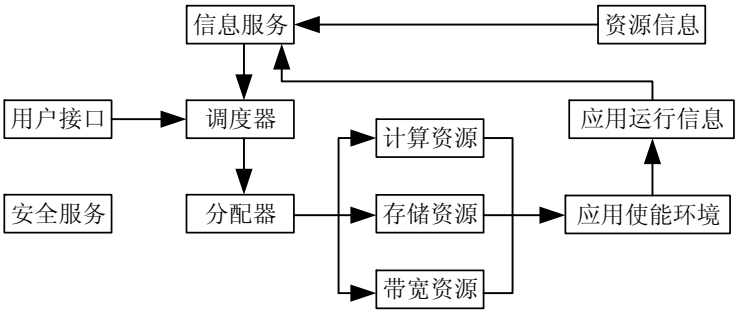


图3.1 网格数据流管理和调度系统的体系结构

①用户接口：用户提交任务、监控任务、取消任务、获取结果等的接口，还应该包括编程接口。在此，将用户接口称为客户端工具（Client Tool），其主要功能是以命令行的方式提供用户与系统的接口。

②调度器：这是资源管理和调度系统的核心，其主要功能是接受用户的请求，并根据用户的要求和当前的资源状况，对资源进行综合的、协调的、按需的调度，实际上调度器只是给出资源分配的参数。从实现结构上讲，调度器就是一个服务器，用户接口则是客户端，两者是分布式的，通过基于 TCP 的 Socket 通信。

③分配器：根据调度器产生的参数，协同分配各种资源。这里涉及与 Condor 等的交互；虚拟机的建立、配置和自动调整；GridFTP 参数调整；存储资源的分配等。

④安全服务：确保网格上资源和任务的安全，是任何一个网格系统最基本也是最重要的功能之一。

⑤信息服务：提供有关资源和任务的信息，为调度器的规划和调度提供依据。信息服务不仅要提供有关当前资源的使用情况，还要对各个应用的运行情

况进行监测，以便了解各种资源的协调运行情况，为下一步的资源调度和优化提供依据。

3.2.1　用户接口

用户通过用户接口以命令行的方式提交任务、监控任务运行情况、终止或删除任务等。用户接口要屏蔽资源使用的复杂性，使用户用可以简单、明了的方式使用资源。由于数据流应用的特点，数据处理和传输必须是同步的，以保证运行的效率，并避免资源浪费。这是数据流应用与其它应用最大的不同，也是数据流资源管理和调度的复杂所在。一般用户提交的只是数据处理的程序，这些程序不能直接提交给资源管理和调度系统，因为没有数据传输，就不能形成数据流。用户接口要完成对这些程序的包装，使数据处理和传输同步进行，然后将这些包装后的应用（已经是一个多线程应用）提交到资源管理和调度系统。资源管理和调度系统就可以为这些包装后的应用分配资源，建立资源容器，保证数据传输和处理的运行。

3.2.2　调度器

调度器根据任务的需求和当前资源的状况，产生调度方案，并传递给资源分配器。调度策略分为两种：粗粒度的和细粒度的，它们的区别主要体现在对计算资源的控制上。粗粒度的调度器是一个元调度器，它要维护一个任务列表，并将任务映射到合适的计算资源上。但是，它要服从资源本地调度器的策略。例如，它可以将一个任务映射到一个计算资源上，而这个任务到底可以获得多少计算资源，则取决于计算资源本地的调度策略。细粒度的资源调度器拥有对计算资源的完全控制，通过虚拟化技术，它可以让每个应用独享一个虚拟机；利用控制理论的方法，它可以为每个虚拟机分配“恰恰足够”的资源。两种调度器对存储资源和带宽资源的调度是一样的。由于存储资源对数据流的处理效率影响不大，这里主要考虑对带宽资源的分配。为了做到按需供应数据，本文采用了一种存储感知的带宽分配方法，它在实质上是一种迭代方法，由此产生的带宽分配方案通过调节 GridFTP 的并行度和 TCP 缓冲区大小来逼近。

3.2.3　分配器

分配器根据得到的调度方案，为应用综合分配计算资源、存储资源和带宽

资源，从而建立一个应用使能环境，保证任务的顺利运行。应当注意的是，对数据流应用而言，这三种资源必须彼此协调。一种资源的过分充足并不会提高处理的效率，但是任何一种资源（尤其是计算资源和带宽资源）的不足，一定会成为限制应用处理效率的瓶颈。

由于数据流应用的特点，资源调度和分配需要动态更新，不可能一成不变。所以，正在运行的应用会受到监视，以便了解其运行情况，发现影响其运行效率的资源瓶颈。这些信息会反馈给调度器，以便做出新的更符合要求的调度。这非常类似于预测控制[95]，即在每个调度时刻求解一个优化问题，得到一个调度周期内的资源调度方案；在下个调度时刻，重复相同的优化问题，周而复始。这种调度具有“边走边优化”的特点，以适应各个调度周期内的具体情况。

3.2.4　安全服务

网格的安全主要涉及远程访问安全管理、用户权利安全管理、作业和任务安全管理等三个方面，集中于解决如何将网格资源安全地分配给网格用户，以及如何保证网格用户安全地使用分配的网格资源。这里采用了 GSI，具体实现为 simpleCA。

3.2.5　信息服务

信息服务负责信息的提供、获取和管理，其基本功能包括信息注册、更新、查询、注销、分发等。这里说的信息，主要是对网格资源管理和调度有意义的数据，如 CPU 的利用率、硬盘的可用存储空间等，这些信息构成了资源管理和调度的依据。各个资源上都运行一个后台程序（Daemon），收集资源的信息，包括静态信息和动态信息。各个正在运行的应用的状态，如各种资源的匹配情况、存在的性能瓶颈等也会被监控，形成应用运行信息，这些信息也被提供给调度器。

3.3　网格数据流资源管理和调度的特点、目标和方法

网格数据流是一种新型的应用，与通常的批处理应用不同的是，在数据处理的同时，它需要实时的、不断的数据供应，且数据是通过网络从远端的数据源传输过来、暂存在计算资源本地的。面向数据流应用的资源管理和调度，具有独特的特点、目标和方法。

3.3.1　网格数据流资源管理和调度的特点

网格数据流应用同时需要足够的带宽、存储和处理能力，以保证数据处理的平稳、高效运行，需要同时分配计算资源、存储资源和带宽资源等，所以这是一种混合资源的分配和调度。总体上说，面向网格数据流应用的资源调度具有以下特点：

①综合调度：由于网格数据流应用的特点，它们同时需要足够的计算资源、存储资源和带宽资源，三者缺一不可。如果没有足够的计算资源，大量的数据即使传输过来，也得不到处理，这种应用方式也就失去了意义；如果没有足够的带宽资源，大量的数据无法及时传输到处理端，处理就无法进行；没有足够的存储资源，传输过来的数据就无法缓存，会导致数据丢失或缓冲区溢出。由此可见，面向网格数据流应用的资源管理和调度需要综合调度多种资源。对于一个具体的应用，调度系统将为其建立一个资源容器，其中的计算资源、存储资源和带宽资源是协同工作的。

②协调调度：三种资源的调度与分配不是独立的，而是相互联系、相互制约、相互依赖的。三种资源的调度与分配应该在一个统一的框架内协调的进行，各种资源应相互匹配。一种资源分配得多，比如分配大量的计算资源给某一个应用，并不一定可以保证期望的处理效率，因为如果没有足够的数据供应，这些计算资源只能处在空闲状态。另一方面，如果某一种资源分配得过少，则一定会成为整体的性能瓶颈。比如，为某一个应用分配的带宽资源太少，则数据从源端到处理端的传输速度太慢，很多时候计算资源没有数据可以处理，从整体上看，应用的处理效率很低。反之，当一个应用运行在一个繁忙的宿主环境中，多个任务之间的资源竞争导致每个应用能够得到的 CPU 周期很少，充足的带宽只会导致数据积压（Data Backlog）乃至溢出。

③按需调度：为了保证数据的处理效率，一种很自然的想法就是为应用分配足够多的甚至冗余的资源。但是，网格是一个共享的计算环境，各种资源都是由大量的应用竞争使用的。一个应用占用的资源超过它的实际需要，就会使其它应用的可用资源变少，加剧资源的紧张状况。同时，这也会导致资源的浪费，还会使资源的用户付出不必要的代价。所以，理想的资源调度应该为每个应用分配“恰恰够用”的资源，只有这样才能同时保证应用的处理效率和资源的利用率。

3.3.2　网格数据流资源管理和调度的目标

从根本上说，与大多数的调度问题一样，面向网格数据流应用的资源调度和分配也可以归结为一个优化问题。对数据流应用来说，优化目标包含两个方面：一个是数据的吞吐量，也就是在给定时间内，所有应用处理的数据量的总和；另一个是资源的有效利用率。由此可见，这是一个双目标优化问题，且这两个目标都是从资源的角度定义的。简单的说，优化的目的就是以最少的资源在给定时间内处理最多的数据。从本质上讲，这两个目标是相互冲突的，需要合理的规划和调度才能同时实现这两个目标。例如，为每个应用都分配充足的乃至超出实际需求的资源，当然可以提高系统的吞吐量，但这会导致资源的浪费，降低资源的有效利用率，且由于实际的资源约束，也不允许这样做。

这个优化问题的约束在于可用资源的有限性及应用之间对资源的相互竞争。相对于应用的需求，系统中可用的计算资源、带宽资源和存储资源都是有限的。要使有限的资源发挥出最大的效益，就必须在相互竞争的应用之间合理的调度和分配资源。

对数据流应用来讲，给定资源集合 R，其中包含计算资源（记作 C）、带宽资源（记作 B）和存储资源（记作 O），则有

$$R=\{C,B,O\}$$

应用 s 组成一个集合，记作 S，有 $s\in S$。设一个评价周期的长度为 L，在任意时刻 $t\in[0,L]$，分配给 s 的资源表示为

$$R_s(t)=\{C_s(t),B_s(t),O_s(t)\}$$

为了讨论的方便，这里不考虑存储空间的分配，因为采用了库存策略（见 3.5.1 节），存储空间对吞吐量和资源利用率的影响不大。记

$$R_s(t)=\{C_s(t),B_s(t)\}$$

此时，s 产生的理论处理能力（即有充足数据时的处理能力）记作 $P_s(C_s(t))$，是关于 $C_s(t)$的非降函数。$B_s(t)$要满足一定的条件，总是在一个有限区间内变化，见 4.4.3 节。s 的有效吞吐率（即给定资源后在单位时间内实际处理的数据量）记作 $E_s(t)$，有

$$E_s(t)=\min\{P_s(C_s(t)),B_s(t)\}$$

即 s 实际处理数据的速度是由理论处理能力和实际的数据供应能力的最小值决

定的，也就是说，计算资源和带宽资源的“短板”决定了一个应用的处理效率。一个周期内 s 的吞吐量记作 T_s，有

$$T_s = \int_0^L E_s(t)dt$$

所有应用的吞吐量之和记作 T，有

$$T = \sum_{s \in S} T_s$$

s 的计算资源利用率记作 H_s，有

$$H_s = \frac{T_s}{\int_0^L P_s(C_s(t))dt}$$

易知

$$H_s \le 1$$

可见，如果一个周期内带宽分配不足，会使 $H_s<1$，导致计算资源浪费；反之，如果计算资源分配不足，则导致计算资源“超载”，$H_s=1$。这两种情况都会限制吞吐量的提高。

至此，优化问题的目标可以定义如下

$$\max T$$

$$\left|H_s - H_0\right| < \varepsilon \quad \forall s \in S$$

其中，H_0 是设定的计算资源利用率，$H_0 \leqslant 1$，ε 是一个小的正数，一般设 H_0 为一个接近于 1 但小于 1 的数值。这两个目标函数的含义是：给定资源集合 R 和应用集合 S，要为每个 s 分配合适的资源，以使一个周期内所有应用的吞吐量之和最大，同时使各个应用的带宽资源和计算资源在一定范围内达到均衡，即两种资源要协调的按需分配。

网格领域现有的一些调度方法（如 2.2.2 节所述），主要是面向计算资源的，且一般要求预先知道任务到达的时间和执行时间，或要求任务的到达服从某种随机过程（如泊松过程），任务的执行时间满足某种随机分布（如指数分布）。调度的目标一般是与时间相关的，如最小化任务总体执行时间、最小化任务平均响应时间等。在数据流应用中，由于要处理的数据量非常庞大，且很多情况下数据是不断的实时的产生的，因此数据流应用从本质上来讲是长时间运行的，或者说，这种应用没有预期的完成时间。这样，以计算资源为主要关注对象、要求预先知道任务的完成时间或其概率分布、以相关时间测度为目标的传统网

格资源调度方法就不适合于数据流应用。

传统的网格资源调度方法中，一般以任务尽快完成为目标，因此会倾向于尽量提高应用的运行速度，会给应用分配一定条件下“最好”的资源。但是在数据流应用中，一味的提高任务的运行速度没有意义，因为这种应用的处理需要与数据供应相匹配。如果没有足够的数据供应，计算资源就只能处于等待状态，不会提高吞吐量，还会导致资源的浪费。对数据流应用来讲，应该给它们分配“恰恰足够”的资源。

3.3.3　粗粒度资源管理和调度算法

具体说来，本文提出并实现了两种资源调度算法，分别是粗粒度资源调度算法和细粒度资源调度算法，两者的区别主要在于对计算资源的控制和分配粒度。这里首先介绍粗粒度的资源调度算法。

在本算法中，对计算资源的分配是粗放式的。各个应用都会提出自己的资源需求，各个计算资源也会声明自己的服务偏好。高层调度器（元调度器）根据两者的匹配情况，在负载均衡的原则下，将各个应用分配到各个计算资源上。各个计算资源拥有自己的本地调度器，具体决定为各个应用分配 CPU 周期、内存空间等计算资源。可见，这里计算资源的分配是分为两个步骤的：高层调度器将应用分配到计算资源上；计算资源的本地调度器负责为应用分配一定数量的资源；高层调度器并不干涉计算资源本地调度器的策略。带宽的分配采用的是迭代式的算法，其中的参数通过遗传算法来决定。资源的调度和分配是周期性的进行的，以根据最新的资源和应用运行情况及时做出调整。

在粗粒度的资源调度和分配算法中，计算资源的分配是通过 Condor 实现的：Condor 根据应用的需求和可用资源的特征，将两者加以匹配，并将应用映射到合适的计算资源上。在这种方法中，一个应用可以获得多少计算资源，是由计算资源本地的调度策略决定的。一个应用获得一定的计算资源后，就会产生一定的处理能力。此时，带宽分配的作用就是依据总体吞吐量最大的原则，为其供应数据。也就是说，本算法无法控制一个应用可以得到的计算能力，可能会出现计算能力与数据供应能力不匹配的情况，所以它被称为粗粒度的调度和分配算法。带宽分配是通过一个迭代算法进行的，其中有一些参数决定着每个应用在一个调度周期中可以获得的带宽，从而决定了一个调度周期内各个应用的吞吐量的总和，详见 4.4.3 节。这些参数不同，带宽分配的方案也不同，所有应

用的总体的吞吐量也不同。如前所述，数据流资源调度和分配的一个重要的优化目标就是最大化数据的吞吐量。因此，有必要确定这些参数值以获得最大的吞吐量。由于本优化问题的性质（非凸非线性），本文利用遗传算法来对这些参数进行寻优。每个染色体的适应值定义为一个调度周期内所有应用的吞吐量的总和，适应值最大的染色体将被保留下来作为迭代式带宽分配的参数。已经知道，数据流应用的吞吐量是由计算能力和数据传输能力共同决定的。所以，为了计算每个染色体的适应值，就需要知道一个调度周期内各个应用的处理能力。在粗粒度方法中，采用的方法是预测。本文采用了分形预测的方法，从应用获得的计算能力的历史信息中对当前调度周期中应用可能获得的计算能力做出预测。带宽分配参数在一个调度内保持不变，以计算在每个最小时间粒度内每个应用获得的具体的带宽。一个调度周期中包含多个最小时间粒度，每个最小时间粒度内各个应用获得的带宽不变。例如，一个最小时间粒度可以定义为 1 秒，一个调度周期可以为 100 秒，即每个调度周期包含 100 个最小时间粒度；在第 10 个最小时间粒度内，某个应用获得的带宽为 0.8Mbps。有了各个最小时间粒度内各个应用获得带宽，就可以计算一个调度周期内各个应用的吞吐量的总和，如 4.2.3 所述。实际上，预测的处理能力主要用于决定数据传输的状态，即决定某一时刻是否为特定应用传输数据，这是因为本文采用了存储感知的带宽分配算法，详见第 4 章。

3.3.4　细粒度资源管理和调度算法

这是一种基于自动控制方法的资源调度算法。由于虚拟化技术的出现，可以使一个应用独立的运行在一个虚拟机上，利用自动控制的技术动态调整虚拟机的资源分配（包括 CPU、内存等），从而使资源分配粒度更细，更好的保证应用的运行效率和资源的利用率。传统的自动控制方法一般需要受控对象（Plant）的模型，这样才可以设计相应的控制算法。但是，在数据流应用中，由于涉及多种资源的协同分配，带宽分配和计算资源的分配是紧密耦合的（Coupling），其精确的数学模型不容易获得。为了便于设计控制算法，一般要进行解耦（Decoupling），这也比较困难。模糊控制提供了一种方便可行的方法，这是一种智能控制方法，它不需要受控对象的精确的数学模型，只要有一些直观的规则即可。模糊控制方法很贴近人类的思维，因而简单易行。本文设计实现的细粒度的资源调度算法，仅仅基于观测和简单的模糊规则，就可以

精确的控制计算资源的分配，动态调节虚拟机的 CPU 时钟频率，从而可以确保应用得到可预测、可控制的性能。在细粒度的资源调度中，带宽的分配依然采用了粗粒度的资源分配中的迭代算法。

细粒度的调度和分配算法拥有对计算资源的完全控制能力，可以更精确、更灵活的决定每个应用获得的计算能力。具体说来，每个应用将被运行在一个虚拟机上，这个虚拟机的资源配置决定了其计算能力。在每一个调度周期中，一个虚拟机的资源配置不变，也就是其计算能力不变。带宽分配采用了与粗粒度调度和分配算法中相同的方法，即基于虚拟机的处理能力，按照总体吞吐量最大的原则，用遗传算法决定带宽分配的参数。这些参数在一个调度周期内也保持不变，同样由迭代式算法计算一个调度周期的每个最小时间粒度内每个应用获得的带宽。对一个虚拟机而言，一个调度周期内获得的带宽的总和与其计算能力的比率，就是这个虚拟机在这个调度周期内的资源利用率（这里没有考虑预先存储的数据）。例如，一个调度周期内，某虚拟机获得的带宽总和为 10MB，即一个周期内可以为应用传输 10MB 数据，而在数据供应充分的情况下，一个周期内该虚拟机可以处理 50MB 的数据，那么其资源利用率就是 20%。与粗粒度的资源调度和分配算法不同的是，基于虚拟机的资源利用率，细粒度的资源调度和分配算法可以根据某些控制规则调整虚拟机的资源配置，以改变其计算能力。这样，在控制器的作用下，下一个调度周期内，虚拟机的资源利用率就会改变，带宽分配的具体参数也会改变，直到虚拟机的资源利用率达到设定的目标。此时，计算资源和带宽资源将达到一个平衡状态，既可以保证资源的利用率，又可以最大化总体的吞吐量，详见第 5 章。

可见，对计算资源来说，粗粒度的调度和分配算法类似于开环控制，因为它不对计算资源的分配做出相应的调整；而细粒度的调度和分配算法则属于闭环控制，因为它采用了反馈的形式，动态的改变计算资源的分配。相应的，粗粒度的调度和分配算法只注重一个优化目标，即吞吐量的最大化，而细粒度的调度和分配算法则可以同时实现两个优化目标。

3.4　网格数据流资源管理和调度系统的工作流程

根据资源调度算法发生作用的层面以及对资源的控制能力，本文提出并实现了粗粒度和细粒度的网格数据流资源管理和调度系统，分别如图 3.2 和图 3.3

所示。从图中可见，这两种系统的结构大致相同：存储资源、网络资源和计算资源上都部署了一些 Daemon，以监测资源的实际利用情况，如可用存储、可用带宽、CPU 利用率；正在运行的应用上也部署了相关的 Daemon，以实时监控各个应用的运行情况，以发现制约性能提高的瓶颈，如计算资源不足等；这些 Daemon 获得的信息，将以一定的频率输入到预测器或控制器；预测器或控制器经过一定的运算，得到性能预测或控制规律，并输入到调度器；调度器进行规划调度，并将调度方案传递给分配器，由后者进行数据传输、任务管理和数据清除。在两种粒度的管理和调度系统中，数据传输都是都是由 GridFTP 完成的，它根据迭代式带宽分配算法自动调整相关参数，详见 4.3.3。同样，在两种管理和调度系统中，处理过的数据会被及时清除，如图 3.2 和图 3.3 中的 *rm* 所示。

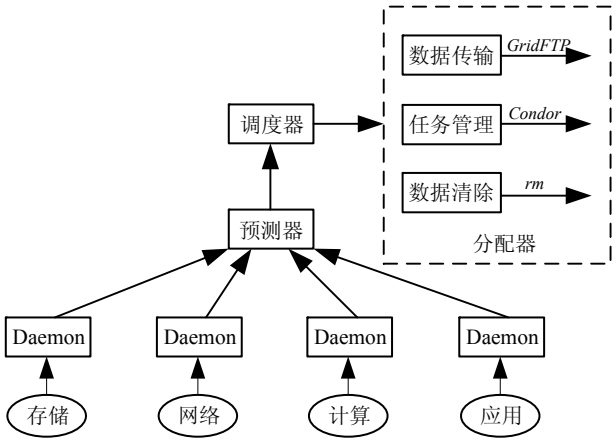


图3.2 粗粒度的网格数据流管理和调度系统工作流程

两种粒度的管理和调度系统最大的不同就是对计算资源的控制能力。在粗粒度的管理和调度系统中，调度器是一个元调度器，它只能将任务提交到计算资源的本地调度器并服从后者的调度策略。由于不能对计算资源实施低层的控制和分配，而计算资源又需要与带宽、存储等协同分配，或者说带宽的分配是处理感知（Processing Aware）的，这里只能对下一个调度周期内各个任务的处理速度进行预测。由于资源的变化以及各个其它因素的影响，预测与实际值总是有一些误差。

细粒度的管理和调度系统中，引入了虚拟化的机制，从而为各个任务建立了一个独立的、可控制、可预测的的运行环境。具体说来，每个任务都会得到一个相互隔离的虚拟机，即图 3.3 中的 VM，其中包含了一组虚拟的硬件。根据资源的利用情况和各个任务的运行情况，控制器按照一定的控制规则动态的调

整各个虚拟机的资源配置，主要是 CPU 的时钟频率，以保证各个任务的顺利运行，并提高资源的有效利用率。这里采用了模糊控制器，它不需要被控对象的数学模型，只需要一些基本经验，从而极大的降低了设计难度，扩大了系统的适用范围。第 5 章给出了详细的说明。

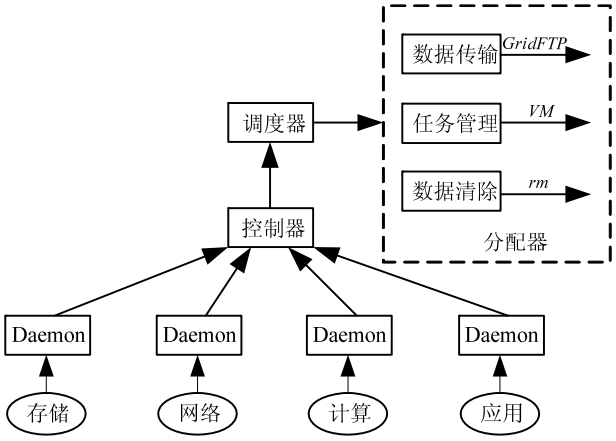


图3.3 细粒度的网格数据流管理和调度系统工作流程

3.5　基于模块的并行数据传输和处理

在一些应用场景中，要处理的数据集是由大量的小文件组成。例如，在 LIGO 中，有些数据是以时间序列的形式存储的二进制文件，每个文件包含在 16-256 秒内多个频道（Channel）所产生的数据。由此带来了数据传输性能方面的一个问题，因为 GridFTP 在传输大文件方面表现出色，但是传输大量的小文件时，其性能会急剧下降。也就是说，GridFTP 并不擅长于传输这样的小文件，这就是著名的“大量小文件”（Lots of Small Files，LOSF）问题。

为了解决这个问题，本文提出了基于模块的数据传输和处理策略[96][97]：数据将按块传输和处理，以提高数据传输和处理的性能。一个块是由一定数量的小文件组成的，而且数据块的大小（即每个块中的文件数量）是需要调度和规划的参数，因为它会影响后面提到的处理完成时间和需要的存储空间。这里一共考虑了三种典型的场景：数据处理比传输快、数据传输比处理快和工作流模式。前面两种场景中，数据传输和处理是并行的，第三种场景中数据传输和处理是串行的。实验结论表明，基于块的数据传输在处理完成时间和需要的存储两方面都有良好表现；并行的数据传输和处理比串行的数据传输和处理效率高；存储感知的传输方案可以大大降低对可用存储的要求，而不会降低处理效率。

3.5.1　基于块的并行数据传输

为了充分利用计算能力，数据传输必须与处理同步进行，即在数据处理的同时，数据必须不断的传输到计算节点以减少空闲时间，提高处理效率。数据是以块为单位传输和处理的，块的大小是一个优化参数，其原则是既要提高处理效率，也要考虑可用的存储空间。

在数据流应用中，并不是数据传输越快越好、存储中的数据越多越好。对数据流应用而言，数据传输必须是存储感知的，因为相对于要处理的大量数据，可用的存储空间总是非常小的。如不对数据传输加以控制，很可能导致数据溢出。这就是说，必须根据数据处理的速度和可用的存储空间，对数据传输加以控制。如果数据传输得太快，而不能及时处理，积累的数据可能需要大量的存储空间；另一方面，如果数据传输的速度低于处理速度，相关的应用就只能等待，从而造成计算资源的浪费。

在一些情况下，对每一个数据元组，数据处理比数据传输的时间长，数据传输就应该是断续而不是连续的，以防止数据溢出。这里采用了所谓的库存策略（Repertory Policy），设定上限和下限以控制数据传输：当存储中的模块数量小于下限时，开始传输数据，直到存储中的模块数量达到上限，此时数据传输暂停。然后，由于数据处理后将被及时删除，存储中的数据模块数量开始减少；当模块的数量达到下限时，数据传输又开始启动。由于数据传输和处理的共同作用，这个过程将会不断重复。上限和下限将根据数据模块和可用存储空间的大小来设定。

3.5.2　问题描述

基于块的数据传输和处理中，主要的优化参数是数据块的大小，用 z 表示，其目标是尽量缩短数据处理的完成时间，用 M 表示。这里假设每个小文件的大小都相等，用 F 表示。

GridFTP 用来按块传输数据，每次传输用来建立连接、完成安全认证、关闭连接的时间记为 t_a。假设每次激活处理程序都需要一小段时间，记作 t_c。进一步假设 t_a 和 t_c 都是常数值。设整个数据集中包含的数据块的数目为 $n(z)$。分别记传输和处理一个数据块的周期为 t_t 和 t_p。显然，不同的 z 对应不同的 t_t 和 t_p，记作 $t_t(z)$和 $t_p(z)$。为分析方便起见，假设 $n(z)t_t(z)$和 $n(z)t_p(z)$都是固定值，即不论整个数据集分成多少个数据块，纯粹用于传输和处理数据的时间是不变的。令 O_a 表

示可用的存储空间，这里将讨论三种应用场景。

⑴ 场景 1：处理快于传输的并行模式

这里，数据处理的速度比传输的速度快，如图 3.4 所示，其中上下两列分别表示数据供应（传输）和处理。

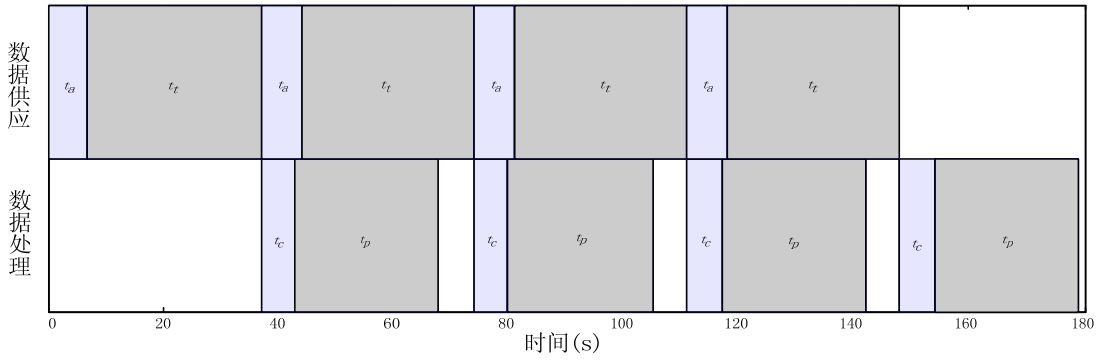


图 3.4 场景 1 的数据传输和处理

总的处理完成时间 M_1 可以计算如下：

$$\begin{aligned} M_1 &= n(z)\left(t_a + t_t(z)\right) + t_c + t_p(z) \\ &= n(z)t_a + t_p(z) + n(z)t_t(z) + t_c \\ &= n(z)t_a + t_p(z) + C_1 \end{aligned}$$

其中 C_1 是一个与 z 无关的常数

$$C_1 = n(z)t_t(z) + t_c$$

可见，z 越小，$n(z)t_a$ 越大，但是 $t_p(z)$越小，反之亦然。所以，必定存在一个最优的 z 使 M_1 最小，如 3.6.3 节的实验结论所示。

至于所需要的存储，可以断言，只要两倍于数据块的大小即可。所以，z 越大，需要的存储越大。但是，z 太小，会使 M_1 太大。所以，必须同时考虑处理完成时间和可用存储，在数据块大小上做出折中。

这个优化问题可以描述如下：

P.

$$\min M_1(z)$$

s.t.

$$2zF \le O_a$$

⑵场景 2：传输快于处理的并行模式

在这种情况下，处理一个数据块的时间比传输一个数据块的时间长，如图 3.5 所示。如果不对数据传输加以控制，将会有大量的数据积压，从而导致数据溢出。

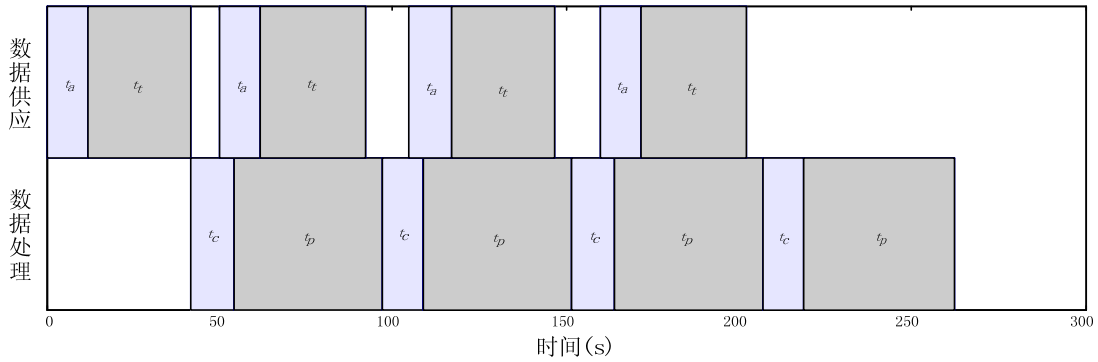


图 3.5 场景 2 的数据传输和处理

这时，总的处理时间记作 M_2，可以计算如下：

$$\begin{aligned} M_2 &= t_a + t_t(z) + n(z)\left(t_c + t_p(z)\right) \\ &= n(z)t_c + t_a + t_t(z) + n(z)t_p(z) \\ &= n(z)t_c + t_t(z) + C_2 \end{aligned}$$

其中

$$C_2 = n(z)t_p(z) + t_a$$

并且 C_2 是与 z 无关的另外一个常数。z 越小，$n(z)t_c$ 越大，但 $t_t(z)$越小。所以，一定存在一个 z 使得 M_2 最小。

此时，数据传输必须是存储感知的。一般的，库存策略的下限 n_L 可以设置为一个数据块的大小，这对并行的数据传输和处理已经足够；上限 n_P 要根据可用存储设定。上下限本身并不影响数据的处理完成时间，但是，上限和下限设置得越高，越有利于提高处理的鲁棒性。当网络发生故障，不能继续传输数据时，应用程序可以处理预先存储在本地的数据，从而避免计算能力的浪费。但是，这是以更多的本地存储为代价的。

每个数据块的大小 z 也需要仔细规划，并需要同时考虑处理完成时间和可用存储，表示如下：

P.

$$\min M_2(z)$$

s.t.

$$n_P zF \le O_a$$

$$n_L \le n_P$$

⑶场景 3：工作流模式

在这种模式下，数据传输和处理是串行的，即一个数据块传输到本地，然后加以处理并删除，再传输下一个数据块，如图 3.6 所示。

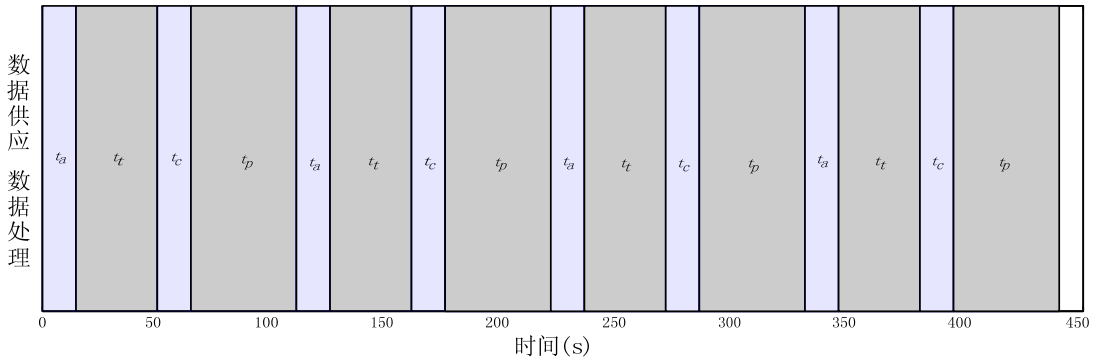


图 3.6 场景 3 的数据传输和处理

总的处理完成时间记作 M_3，可以计算如下：

$$\begin{aligned} M_3 &= n(z)\left(t_a + t_t(z) + t_c + t_p(z)\right) \\ &= n(z)\left(t_a + t_c\right) + n(z)\left(t_t(z) + t_p(z)\right) \\ &= n(z)\left(t_a + t_c\right) + C_3 \end{aligned}$$

其中

$$C_3 = n(z)\left(t_t(z) + t_p(z)\right)$$

而且 C_3 是另外一个常数。

可见，z 越小，处理完成时间越长，因为处理过程中会包含更多的认证和

函数调用。另一方面，z 越大，需要的存储空间越大。所以，在处理完成时间和可用存储之间也有一个折中。

这个优化问题可以表示为

P.

$$\min M_3(z)$$

s.t.

$$zF \le O_a$$

⑷ 形式化描述

从以上分析可知，该优化问题可以形式化描述如下：在占用的存储空间 O_o 不超过可用存储空间 O_a 的条件下，寻找一个最优的参数，即每个数据块的大小 z，使总的处理完成时间 M 最小，即

P.

$$\min M(z)$$

s.t.

$$o_o \le o_a$$

这里没有考虑可能影响处理完成时间的其它参数，如 t_a 和 t_c 等，因为它们不受终端用户或本优化算法的控制。目前只是定性的解决了这个优化问题，但是基于经验的确可以提供一些指导原则。还应该说明的是，图 3.4、图 3.5 和图 3.6 中 t_a、t_c 等并非精确的数值，只是为了说明数据传输和处理的过程。

3.5.3　实验结论

本节介绍了基于模块的数据传输和处理的一些实验结果。

⑴ 实验设计

实验中一共用到了 3 台计算机，其中 2 台是处理机，另外 1 台是存储器，计算机之间的理论带宽为 10 Mbps 且由应用共享。两台处理机的负载不同，所以其处理速度不同：一台处理机的处理速度比传输快，另外一台处理机的处理速度则比传输慢。 实验中用到的数据处理程序时 6.1.1 节中介绍的 rmon，共对 1188 对小文件进行处理。

⑵ 处理完成时间

在场景 1 和场景 2 中，随着数据块大小的变化，处理完成时间显示出相似

的特性，如图 3.7 和图 3.8 所示，这证明了场景 1 和场景 2 中的断言，即一定存在一个 z，记作 z_{opt}，使完成时间最短。当 $z<z_{opt}$ 时，完成时间随着 z 的增大而减小，其下降速率较快；当 $z>z_{opt}$ 时，完成时间随着 z 的增大而增大，其上升速率较慢，这反映出不同参数对完成时间的影响。在场景 3 中，数据处理完成时间与模块大小的关系与此不同，随着 z 的增加，处理完成时间是逐渐减小的，如图 3.9 所示。

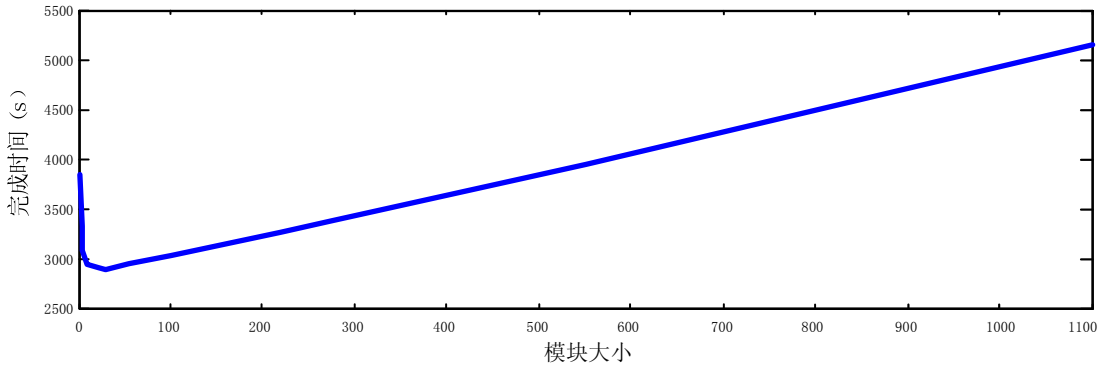


图 3.7　场景 1 的数据处理完成时间

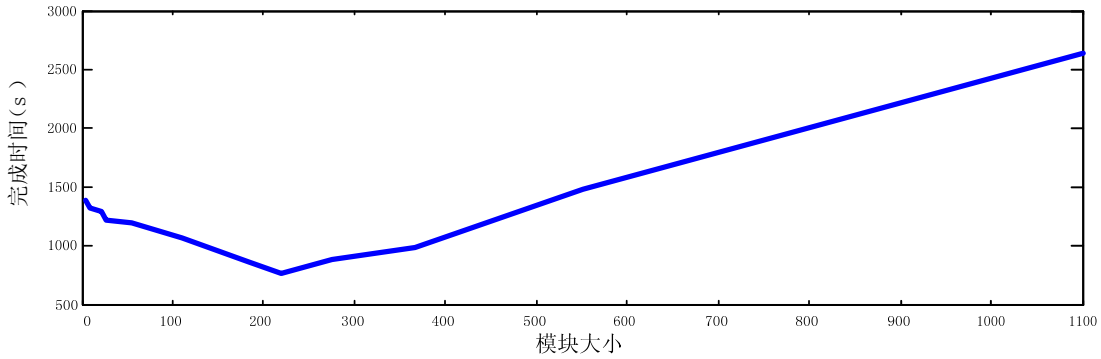


图 3.8　场景 2 的数据处理完成时间

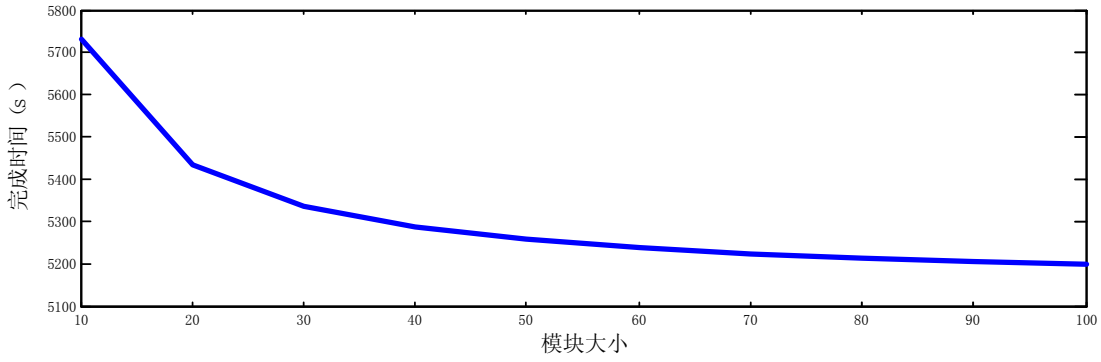


图 3.9　场景 3 的数据处理完成时间

⑶ 存储使用

在场景 1 和 3 中，所需的存储分别是两个和一个数据块的大小，所以单个数据块越小，所需要的存储空间越小。

这里，每个数据块包含 55 个数据文件。如图 3.10 所示，场景 1 中本地存储中的总的文件数量在一个有限的范围内周期性的变化，这是并行数据传输和处理共同作用的结果。在整个数据传输和处理过程中，占用的存储空间都不会无限增加。

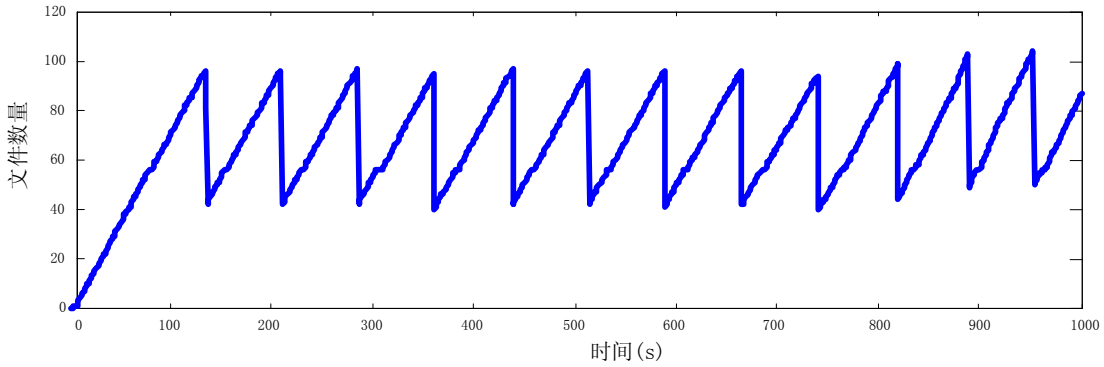


图 3.10 场景 1 的存储使用情况（z=55）

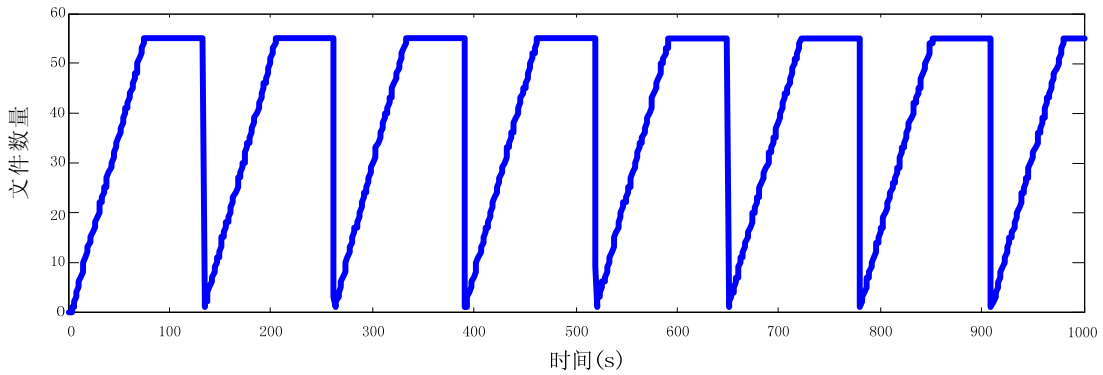


图 3.11 场景 3 的存储使用情况（z=55）

如图 3.11 所示，在场景 3 中，占用的存储空间不会超过一个数据块。 此时，数据传输、处理和删除是串行执行的。虽然需要的存储空间变小了，但其处理完成时间却变长了，几乎是场景 1 中的处理完成时间的两倍。很明显，本场景的整体吞吐量下降了，且资源没有得到充分利用。另外可以看到，存储中的文件数量不是连续的，曲线中有一些突变，这是因为数据处理和删除是以数据块而不是以单个数据文件为单位进行的。

场景 2 的实验结论如图 3.12 所示。可见，如果不采取库存策略，数据将不停传输到本地存储。由于数据处理比数据传输慢，数据文件可能积累到将近 1000 个，然后按块处理和删除。采用库存策略后，存储占用情况大为改善。这里，库存策略的上限和下限分别设置为 5 个和 1 个数据块。此时，数据传输是断续的而不是连续的，以便将占用的存储控制在合理的范围内。同时，这也可以保证存储中始终有数据，从而使处理可以不断进行。这样，仅仅利用少量的存储空间，就可以高效的处理大量的数据。

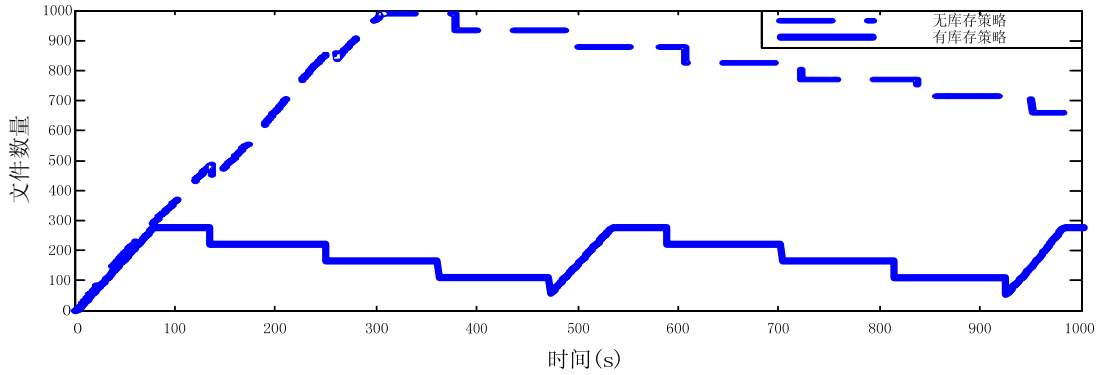


图 3.12 场景 2 的存储使用情况（z=55）

(4) 存储感知与处理完成时间

库存策略使数据传输做到了存储感知。存储感知特性（Storage Awareness）并没有延长处理完成时间，如图 3.12 所示，两种情况下的处理完成时间是一样的。而且，处理完成时间与库存策略的上下限无关。因此，存储感知特性在此是适用的，它使得可以用较小的存储空间来处理较大的数据集，又不会降低其处理效率。

3.6　小结

本章介绍了面向网格数据流的资源管理和调度系统的基本概况。本系统的目的是方便用户对资源的使用，同时对资源进行有效的管理和调度，以提高资源的利用率。本系统的组成部分包括用户接口、调度器、分配器等，并包含安全服务和信息服务等。本章重点介绍了面向网格数据流应用的资源调度的方法，这是一种混合资源调度方法，需要对计算资源、存储资源和带宽资源进行综合的、协调的、按需的调度和分配。本章简要介绍了粗粒度的调度方法和细粒度

的调度算法，两者的主要区别在于对计算资源的分配和控制粒度。面向数据流应用的混合资源管理和调度可以归结为一个优化问题，其目标是最大化系统的吞吐量，并提高资源的有效利用率，而其约束是资源的有限性和应用对资源的竞争。根据要处理的数据的特点，本系统对数据的传输和处理都是按模块进行的。实验结论显示，每个数据模块的大小对于应用的执行效率具有明显影响。

第 4 章　粗粒度的数据流资源管理和调度

针对网格数据流应用，本章提出了一种粗粒度的资源管理和调度算法。在每一个调度周期开始时，调度器调用 Condor 将通过接纳控制（Admission Control）的新应用映射到合适的计算资源上；运行遗传算法寻找带宽分配的最佳参数，并为各个应用迭代式的分配带宽。由于网格是一个开放、共享的资源环境，资源和应用的状态是经常变化的，所以调度是不断的、周期性的进行的。也就是说，这是一个动态调度的过程。一般而言，根据调度时机的不同，动态调度又分为在线（On-Line）调度和批处理（Batch）调度两种模式[43]。在线动态调度是指当一个任务送达后，调度组件立即把它映射到某个资源上，它每次只调度一个任务。在批处理动态调度中，只有映射事件（Mapping Event）发生后，才把新任务与其它尚未开始执行的任务一起进行调度。所谓映射事件，可以是到达预定义的调度周期，也可能是某些任务执行完毕。这里动态调度采用的是批处理模式：达到预定义的调度周期时，新接受的任务与正在运行的任务一起调度。

所谓资源调度的粗粒度，是指资源调度器不拥有对它所调度的资源（这里主要是指计算资源）的完全控制权。实际上，这里的调度器是一个高层调度器，也就是一个元调度器。元调度器只能服从资源本地的管理和调度策略，它做不到对资源的精确控制。

4.1　问题描述

粗粒度调度算法依赖于 Condor 完成计算资源的分配，后者是一个低层调度器，负责将任务映射到合适的计算资源；带宽则是根据规划好的参数进行迭代式的分配，并且这个分配方案是由 GridFTP 来逼近的，其控制参数（包括传输并行度和 TCP 缓冲区大小）是实时可调的；存储将按照规划的份额分配给每个应用。带宽分配的参数是由优化调度算法产生的，这个优化调度算法包括一个接纳控制器、一个状态估计器和一个参数优化器。调度和分配的整个流程如图 4.1 所示，其中①、②和③分别表示新接纳的任务、预测的处理速度和优化分配参数。

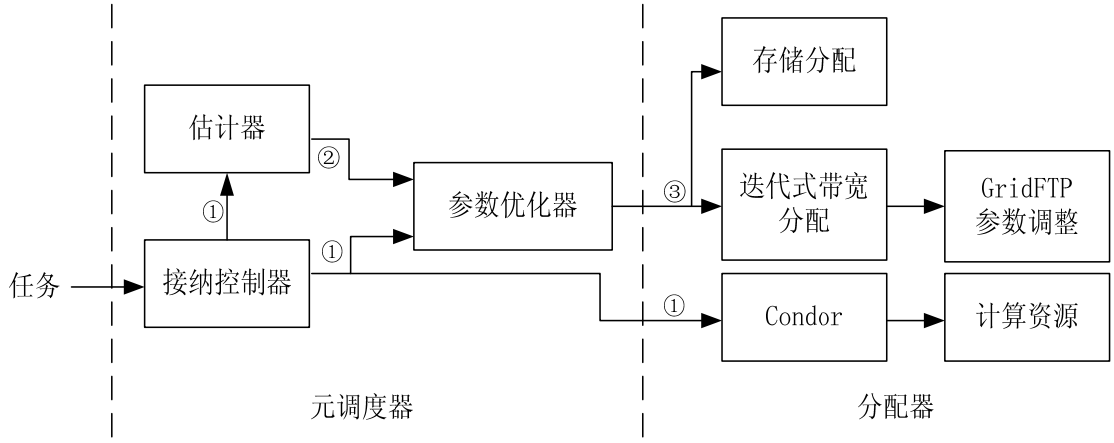


图 4.1 粗粒度调度和分配的整体流程

调度和分配的机制描述如下。当一个新任务到达时，根据其优先级，它将被放置到一个相应的等待队列中。在每个调度周期的开始，接纳控制器将根据任务对资源的要求和当前资源的状况，从各个等待队列中选择合适的任务。新接纳的任务将与当前正在运行的任务一起组成一个集合，称为活跃任务集合（Active Task Set），简称 ATS。同时，新接纳的任务将被提交到 Condor 以便进行计算资源分配。估计器将根据历史信息，预测 ATS 中各个任务在下一个调度周期中的处理速度，预测结果将输入参数优化器，由其产生带宽分配的最佳参数。接纳控制器、估计器和参数优化器组成了一个元调度器，它负责产生最佳的带宽分配参数。分配器将为 ATS 中的任务分配资源。

4.2　调度参数的产生

本节将介绍最优参数的产生过程，这些最优参数将被应用到下一节介绍的带宽分配算法。其工作过程如下：当一个应用被提交时，接纳控制根据当前资源的使用情况，决定是否将其放入 ATS 中；在下一个调度时刻，ATS 中的应用将被分配合适的资源；带宽资源分配方案的相关参数是由遗传算法给出的，其目标是最大化下一个调度周期内 ATS 中所有应用的总体吞吐量；遗传算法要用到下一个调度周期内各个应用的运行效率，而这是通过估计和预测实现的。

4.2.1　接纳控制

资源池不能同时接纳无限多的应用，因为太多的应用会导致对资源的严重

竞争，从而降低整体的处理效率。当一个新任务到达时，根据其声明的优先级（Priority），它将被放到一个等待队列中，如图 4.2 所示。系统同时维持多个队列，在同一个队列中的任务的优先级相同。此外，任务还将声明其对资源的最小需求。例如，它需要一个具有特定结构的 CPU、512MB 的内存和 1024MB 的硬盘空间。同时，系统实时监控可用资源，以获得最新的信息。

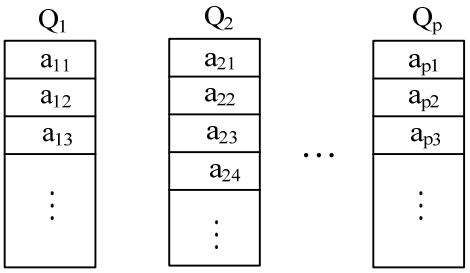


图 4.2 等待队列

在每个调度周期的开始，接纳控制器将决定将等待队列中的哪些任务放置到 ATS 中。首先，它将检查优先级最高的队列，如图 4.2 中的 Q_1。在每个等待队列中，选择的策略是先来先服务（First-Come-First-Serve，FCFS），同时将采用回填机制。例如，Q_1 中的任务按照其到达的时间顺序从上到下排列，即 a_{11} 最先到达。因此，在 Q_1 中，被选中的任务可能是 a_{11} 和 a_{13}，而 a_{12} 将被保持在等待队列中，因为当前它对资源的要求得不到满足。同样的选择过程将依次发生在优先级较低的等待队列，被选中的所有任务将被放置到 ATS 中。

而且，为了避免一些任务长时间的在队列中等待，还采取了一些预留策略（Reservation Policy）。一些任务可以预约一些资源，当这些资源被释放后，在下一个调度周期内这些任务将得到这些资源。每个任务的优先级会随着时间而增加，以避免低优先级的任务始终得不到资源。因此，优先级将是时间的函数，如

$$w_i(t) = f_i(w_{io}, t)$$

$$f_i(w_{i0}, 0) = w_{i0}$$

其中，w_i（t）是任务 i 在 t 时刻的优先级，w_{i0} 是任务 i 的初始优先级，并且 $f_i(t)$ 是关于时间 t 的非降函数。一个常见的函数是

$$w_i(t) = f_i(w_{io}, t) = w_{io} + d_i f(t, T_i)$$

$$f(t,T_i)=\lfloor (t/T_i) \rfloor$$

其中 d_i 是增长系数，且 $d_i>0$；T_i 是增长周期，$\lfloor\ \rfloor$表示向下取整。这样，每隔周期 T_i，w_i 将增加 d_i，相应的任务就会被迁移到优先级更高的队列中，直到达到最高优先级。通过为 d_i 和 T_i 设置合适的数值，经过一定的时间后，原本优先级较低的任务将获得足够高的优先级以便被从等待队列中选中。

如果没有接纳控制，所有的任务将在其到达后的第一个调度周期开始时放置到 ATS 中并分配资源。实验结论表明，无论从资源还是应用的角度来说，接纳控制都可以获得更好的整体性能。

4.2.2　估计/预测

图 4.1 中的优化参数是与 ATS 中任务的处理速度相关的，如 4.4.3 节所示。但是元调度器依赖于 Condor 并服从于其调度策略，也就是说，计算资源的分配是不受元调度器的控制的。这样，元调度器就不可能为某个特定的任务分配特定数量的 CPU 周期以获得期望的处理速度。所以，有必要根据历史数据，对 ATS 中的任务在下一个调度周期中的处理速度进行预测，这就是图 4.1 中的估计器的作用。在最初的调度周期中，由于性能信息不足，预测也就不够精确。随着性能信息的越来越多，就可以做出更好的预测。各个任务的实际运行速度将被实时采集作为以后的历史信息，以便预测未来的处理速度。

在粗粒度的管理和调度算法中，采用了分形预测的方法，下面以 CPU 利用率为例说明这种方法的基本原理。分形分布可以描述为

$$J=\frac{Y}{q^Z}$$

其中 q 是采样时间，是一个独立的变量；J 是以百分比表示的 CPU 利用率，它与 q 相关联；Y 是一个需要计算的常数，Z 代表分形维数。

定义初始数据 J 的一个序列 J_k，并计算其第 i 阶累加和 $S_{i,j}$ 如下：

$$S_{i,j}=\begin{cases}\sum_{k=1}^{j} J_k & i=1\\ \sum_{k=1}^{j} S_{i-1,k} & i>1\end{cases}$$

分形维数 $Z_{i,j}$ 和常数 $Y_{i,j}$ 分别计算如下：

$$Z_{i,j}=\frac{\ln\frac{S_{i,j+1}}{S_{i,j}}}{\ln\frac{q_j}{q_{j+1}}}$$

$$Y_{i,j}=S_{i,j}q_j^{Z_{i,j}}$$

其中 $q_j=j$ ($j=1,...,e-1$)，e 为采用点的个数。

这里采用了二阶累加和，因为在大多数情况下预测结果足够精确，并且其计算开销比较合理。这时候，其预测值可以表示为：

$$J_{e+1}=\frac{S_{2,e}e^{Z_{2,e-1}}}{(e+1)^{Z_{2,e-1}}}-S_{2,e}-S_{1,e}$$

表 4.1 给出了一些预测结果。J_j 是 10 秒内 CPU 利用率的平均值（以百分比表示）。从表中可以看出，尽管 $Z_{1,j}$ 在一定范围内变化，$Z_{2,j}$、$Z_{3,j}$ 和 $Z_{4,j}$ 分别接近于一个稳定值，这表示分形维数趋向稳定。J_{16} 的预测值和测量值分别为 36.9% 和 38.4%，相对误差为 3.91%。

表 4.1 CPU 利用率的分形预测（e=15）

J_j	$S_{1,j}$	$S_{2,j}$	$S_{3,j}$	$S_{4,j}$	$Z_{1,j}$	$Z_{2,j}$	$Z_{3,j}$	$Z_{4,j}$
39.6	39.6	39.6	39.6	39.6	-1.0641	-1.6280	-2.0324	-2.3479
43.2	82.8	122.4	162	200	-1.2830	-1.8742	-2.3712	-2.7917
56.5	139.3	261.7	424	630	-1.1941	-1.9462	-2.5477	-3.0579
57.1	196.4	458.1	882	1510	-1.1331	-1.9700	-2.6498	-3.2319
56.5	256.9	711.0	1593	3100	-0.9513	-1.9351	-2.6974	-3.3451
47.9	300.8	1011.8	2605	5700	-0.9716	-1.9243	-2.7274	-3.4239
48.6	349.4	1361.2	3966	9670	-0.9035	-1.9046	-2.7445	-3.4805
44.8	394.2	1755.4	5721	15390	-0.7084	-1.8544	-2.7451	-3.5191
34.3	428.5	2183.9	7905	23300	-0.8288	-1.8414	-2.7453	-3.5471
39.1	467.6	2651.5	10557	33850	-0.8653	-1.8385	-2.7469	-3.5689
40.2	507.8	3159.3	13761	47570	-0.8567	-1.8355	-2.7490	-3.5866
39.3	547.1	3706.4	17422	64990	-0.7597	-1.8204	-2.7489	-3.6008
34.3	581.4	4287.8	21710	86700	-0.8260	-1.8171	-2.7492	-3.6126
36.7	618.1	4905.9	26616	113320	-0.8692	-1.8198	-2.7507	-3.6229
38.2	656.3	5562.2	32178	145500	-----	-----	-----	-----

图 4.3 给出了 100 个采样点和预测值的比较，从中可以看到，尽管测量值和预测值有一些误差，分形预测的确给出了 CPU 利用率的趋势。而且还可以看

到，预测值非常接近于测量值的平均值，而 CPU 利用率的平均值显然比其瞬时值更有意义。

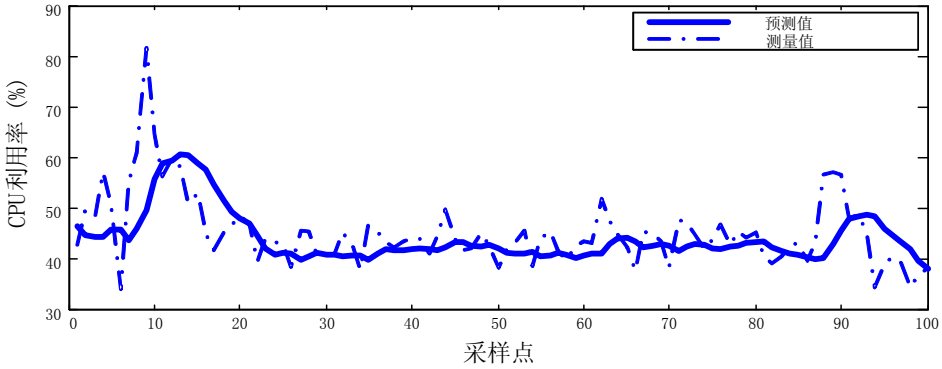


图4.3 CPU利用率的预测值与测量值

4.2.3　优化参数的产生

资源调度和分配是周期性进行的，所谓的优化参数也只对一个调度周期有效，因为资源和任务的状态都是经常变化的。调度算法最重要的评价标准是一个周期内的吞吐量，这意味着这个算法试图最大化一个周期内 ATS 中所有任务处理的数据的总和。

如 4.3.3 节所示，带宽分配（即数据传输）方案由几个参数决定，并且这几个参数对最大化一个周期内整体的吞吐量至关重要。假设一个调度周期包含 l 个时间单位，每个时间单位是预先定义的最小时间粒度。用 $x_{s,k}$ 表示应用 s 在第 $k(k=1,2,\ldots,l)$个时间单位的得到的带宽（$x_{s,k}$ 的计算见 4.4.3 节），用 $O_{s,k}$ 表示应用 s 在第 k 个时间单位的占用的存储空间。可以计算一个调度周期内应用 s 的吞吐量 T_s 和 ATS 中所有应用的吞吐量 T 如下

$$T_s = \sum_{k=1}^{l} x_{s,k} + O_{s,1} - O_{s,l} \tag{4-1}$$

$$T = \sum_{s} T_s \tag{4-2}$$

也就是说，一个应用在一个调度周期内的数据吞吐量是这个调度周期开始和结束时的存储中的数据量的差值再加上在这个周期内传输过来的数据量。为了简化问题，我们假设各个应用的数据吞吐量是可以比较的，所有 T_s 的总和可以用来表示整体的数据吞吐量。如果在一个真实的环境中，有些数据吞吐量是不可比较的，即应用之间的吞吐量可能相差很大，从而使各个应用的吞吐量直接相加没有意义或不够公平，则可以在各个应用之间进行归一化处理，即赋予各个

应用的吞吐量以合适的权重。同时，由于采用了库存策略，一个应用占用的存储空间是有限的，也就是说，（4-1）中后两项是比较小的，与数据流应用处理的数据量相比更是如此。因此，（4-1）往往可以简化如下

$$T_s = \sum_{k=1}^{l} x_{s,k} \tag{4-3}$$

粗粒度的资源调度和分配问题可以转化为一个优化问题，其目标是最大化每个调度周期内的总的吞吐量，即（4-2）中定义的 T，可用的计算资源、存储和带宽可以视为约束条件。给定 ATS 及其计算资源分配方案，总的吞吐量将取决于数据供应，即带宽分配方案。在一个分配周期内，带宽分配是由参数优化器输出的优化参数决定的。在一个调度周期内，这些参数是固定的，带宽是按照迭代式算法分配的。所以，这个优化问题就是要为带宽分配找到合适的参数。由 4.3.3 可知，带宽分配的三个参数分别是 α、β 和 ρ，其取值范围均在 0 到 1 之间。由（4-4）到（4-6）可知，给定一组 α、β 和 ρ，就可以确定由（4-2）和（4-3）定义的吞吐量。也就是说，一个调度周期内 ATS 中所有应用的吞吐量之和是 α、β 和 ρ 的函数，记为 $T(\alpha, \beta, \rho)$。因此，本问题可以描述如下

$$\max T(\alpha,\beta,\rho)$$

问题的可行域 V 是三维实数空间中的一个凸集，因为对于 V 中的任意两点 $v^{(1)}$，$v^{(2)}$及每个实数 λ∈[0，1]，都有

$$\lambda v^{(1)} + (1-\lambda)v^{(2)} \in V$$

但是，$T(\alpha, \beta, \rho)$并不是 V 上的一个凸函数，因为，对于 V 中的任意两点 $v^{(1)}$，$v^{(2)}$及每个实数 λ∈[0，1]

$$T\left(\lambda v^{(1)} + (1-\lambda)v^{(2)}\right) \le \lambda T\left(v^{(1)}\right) + (1-\lambda)T\left(v^{(2)}\right)$$

并不总是成立。如令 $v^{(1)}$=(0.8, 0.8, 0.8)，$v^{(2)}$=(0.1, 0.1, 0.1)，λ=0.5，有 $v^{(3)}$=$\lambda v^{(1)}$+(1-λ)$v^{(2)}$=(0.45, 0.45, 0.45)，且 T($v^{(1)}$)=21700，T($v^{(2)}$)=16698，T($v^{(3)}$)=21703，故 T($v^{(3)}$)>λT($v^{(1)}$)+(1-λ)T($v^{(2)}$)。同样可以举例证明，$T(\alpha, \beta, \rho)$也不是线性的。实际上，固定一个参数 ρ，可以看到另外两个参数和与一个周期内 ATS 中所有任务的吞吐量的总和的关系如图 4.4 所示。这样，一些常用的优化方法[98]在此不适用，因此本文采用了遗传算法进行参数寻优。

在遗传算法中，一个染色体包括带宽分配的有关参数，如下所示：

$$CHROM_{i,g} = \{\alpha_{i,g}, \beta_{i,g}, \rho_{i,g}\}$$

其中 g=1, 2, …, G，其中 G 是一个种群中染色体的数量，i 表示进化的代数。$\alpha_{i,g}$、

$\beta_{i,g}$ 和 $\rho_{i,g}$ 以二进制形式编码，这三个参数都在 0 到 1 之间变化，其二进制字长为 7，即各个参数的范围为[U_{min}, U_{max}]，其中 U_{min} =0，U_{max} =1，编码长度 y=7。则其编码精度为

$$\Pi = \frac{U_{\max} - U_{\min}}{2^{y} - 1}$$

任何参数 U∈[U_{min}, U_{max}]，对应的二进制编码的数值 u 为

$$u = \frac{U - U_{\min}}{U_{\max} - U_{\min}}\left(2^{y} - 1\right)$$

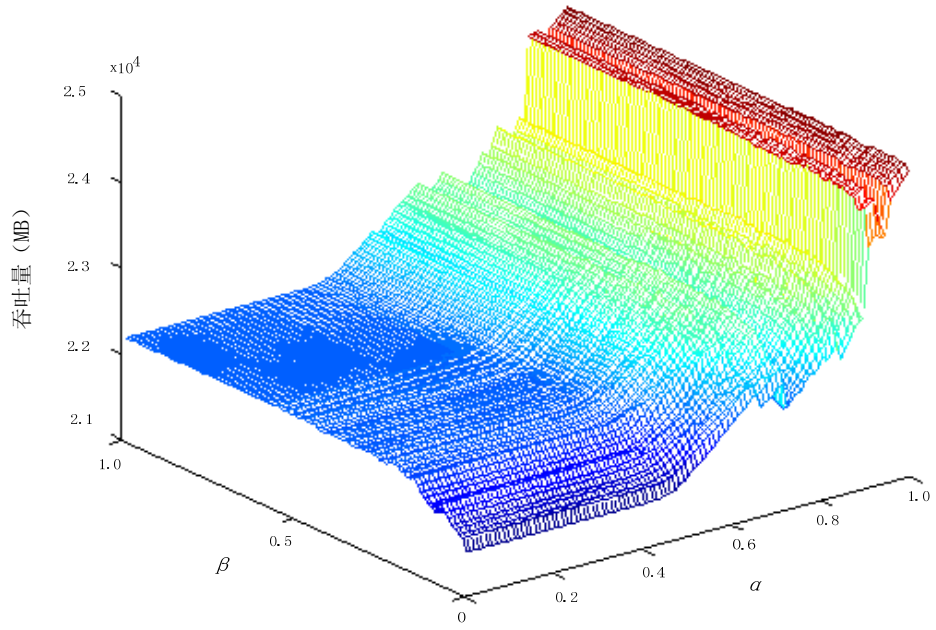


图 4.4　带宽分配参数与吞吐量的关系

给定一个染色体，其适应值记作 $f_{i,g}$，定义为一个调度周期内的总的吞吐量，即 ATS 中的所有任务处理的数据的总和，也就是本节中定义的 T。染色体种群数量的初始值设置为 40，即 G=40。在每一代中，染色体被随机分成 20 组，每组内的两个染色体相互交配产生两个新的后代，这相当于染色体的交叉。在此操作中，$\alpha_{i,g}$、$\beta_{i,g}$ 和 $\rho_{i,g}$ 的二进制编码将从一个随机选择的点交换部分基因。这样，在每一代中，就有 80 个染色体（是初始染色体数量的两倍），但是只有适应值排在前面的 40 个染色体可以保留到下一代。变异概率设置为 0.1，以保证不断有新的染色体产生，防止染色体种群整体性能的下降，即防止陷入局部最优解。如果相邻两代中最大的适应值的相对误差小于一定的评判标准，即

$$E\left(f_{i+1,g},f_{i,g}\right)=\frac{\max\left(f_{i+1,g}\right)-\max\left(f_{i,g}\right)}{\max\left(f_{i,g}\right)}<\chi$$

即可认为已经找到了最优解，此时便可停止迭代，输出最优解，否则继续进行迭代。其中，$E(f_{i+1,g},f_{i,g})$是两次迭代的相对误差，$\max(f_{i,g})$与 $\max(f_{i+1,g})$分别是第 i 次迭代和第 i+1 次迭代时各染色体的最大适应值，χ 是预先定义的评判标准（一般取为百分之一或千分之一）。

4.3　资源分配

对数据流应用而言，计算资源、带宽和存储是在一个统一的框架内分配的，以保证其顺利运行。

4.3.1　计算资源分配

新接纳到 ATS 中的任务将被包装为一个多线程的应用，保证其数据传输和处理的同步进行，详见第 6 章。这个被包装过的应用，将被提交到 Condor，后者负责计算资源的分配。这里没有任务迁移的情况，即一个应用将固定在一个计算节点上运行直到结束。在每个调度周期，Condor 只需要将新进入 ATS 的应用映射到合适的计算节点。

4.3.2　存储资源分配

存储分配的基本原则是充分利用可用的存储，以提高处理的鲁棒性，同时为新来的任务准备好存储空间。如果资源池中只有少量的任务在运行，分配给每个任务的存储就可以多一些。当同时运行的任务数增加时，分配给每个任务的存储就可能少一些。这样，每个任务得到的存储是可以相应扩展的。

一个存储分配方案包括上限和下限。上限和下限主要用作控制数据传输的开始或结束的门限（Threshold）：如果存储中的数据量达到下限，就应该启动数据传输，直到数据量达到上限，这就是 3.5.1 中介绍的库存策略。每个任务的存储的上限用来保证总的数据量不会超过可用的存储空间。下限保证在网络故障、不能传输数据时，系统可以处理存储中的数据，以避免计算能力的浪费，这提高了系统的鲁棒性和 CPU 的利用率。如图 4.5 所示，大多数时间内，存储中的文件数量在设定的上下限之间变化。在某个时刻，由于网络故障，不能继续传

输数据，则任务会处理预先存储下来的数据，这时存储中的数据量会减少到下限之下，但是处理并没有中断。

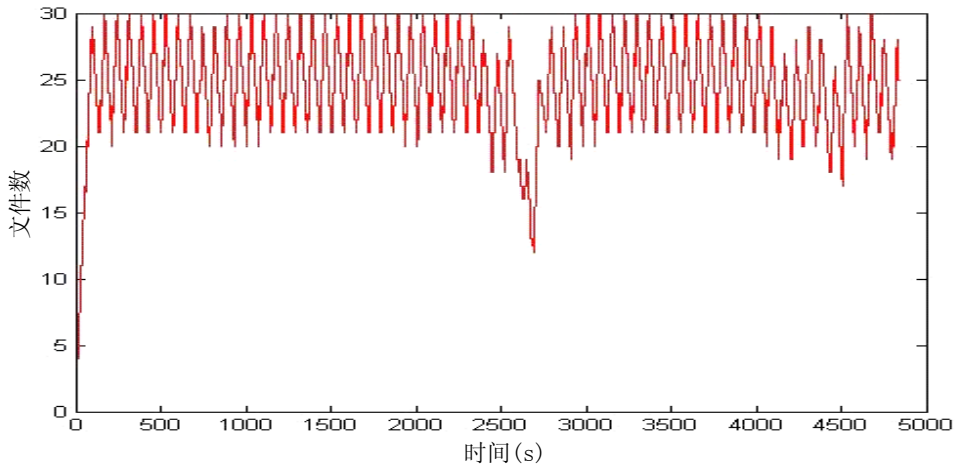


图 4.5 存储使用情况

4.3.3　带宽资源分配

在整个资源分配方案中，带宽分配起着非常重要的作用，因为只有恰当数量的带宽才能保证应用顺利运行。与传统的带宽分配方案不同，本文的方案是存储感知的，即数据传输可能是断续的而不是连续的，以防止数据溢出。当存储空间中的数据量达到规定的上限时，数据传输就会暂停，直到一些数据被处理并被清除而使存储中的数据量达到规定的下限。在任何时刻，存储中的数据量都是由数据传输和处理共同决定的。

数据流应用运行在一个集群中的处理器上，而相应的数据源则在远程的节点上，所以需要通过网络（比如互联网）把数据传输到本地的存储中，如图 4.6 所示。从集群到互联网的带宽是有限的，用 I 表示，这个带宽是由许多应用共享的。ATS 中的应用形成了一个集合，记作 S。每个 s 将被分配一定的带宽，记作 x_s，其中 $x_s \in X_s$，$X_s=[b_s, B_s]$，$b_s>0$，$B_s<\infty$。b_s 表示应用 s 需要的最小带宽，B_s 表示从相应的数据源到 s 的最大可用带宽，即从数据源到 s 的瓶颈连接处的带宽。应当指出，b_s 和 B_s 也是时变的。

当分配给应用 s 的带宽为 x_s 时，引入一个效用函数，记作 $U_s(x_s)$。在区间上 $[b_s, B_s]$上，$U_s(x_s)$应该是凸的、连续的、递增的。这里并不要求所有的应用都采用同一形式的效用函数。于是，带宽分配就转化为一个优化问题：

P.

$$\max \sum_{s \in S} U_s(x_s)$$

s.t.

$$\sum_{s \in S} x_s \le I$$

$$x_s \in X_s$$

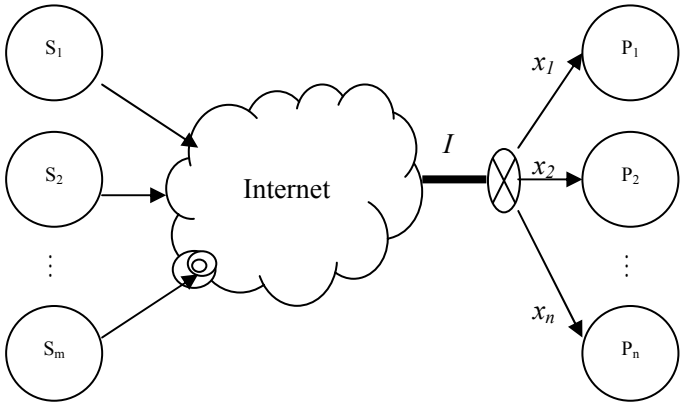


图4.6 数据流应用场景

根据存储的使用状态和库存策略，在任何时刻，应用 s 的数据传输都有两个可能的状态，即活跃的或休眠的，这些状态指示数据传输是在进行还是停止的。分别用 H_s 和 L_s 表示应用 s 存储使用的上限和下限。令 $A_{s,k}$ 表示在第 k 时刻，应用 s 的数据传输的状态，$O_{s,k}$ 表示应用 s 在第 k 时刻占用的存储空间。$A_{s,k}$ 的初始值设置为 1，并按如下方式演化：

$$O_{s,k} = O_{s,k-1} + x_{s,k} - p_{s,k} \tag{4-4}$$

$$A_{s,k+1} = \begin{cases} 1 & A_{s,k} = 1, O_{s,k} < H_s \\ 0 & O_{s,k} \ge H_s \\ 0 & A_{s,k} = 0, O_{s,k} \ge L_s \\ 1 & O_{s,k} < L_s \end{cases} \tag{4-5}$$

其中 $x_{s,k}$ 和 $p_{s,k}$ 分别是在第 k 个时刻应用 s 获得的带宽和处理速度，$p_{s,k}$ 需要用 4.2.2 节中提到的分形方法加以预测。$A_{s,k}$ 的数值（0 或者 1）分别表示应用 s 的数据传输是正在进行还是暂时停止。

关于带宽分配，文献[99]中提出了一个迭代优化方法，并分析了其收敛性，但是它要求知道每个链路上的拥塞情况，在互联网上这是很难做到的。本文对

这个迭代算法进行了改进，这里只需要根据存储占用情况和入口带宽的使用情况，就可以迭代式的为各个应用分配合适的带宽。相对而言，这种算法更容易实现。

$$x_{s,k+1}=\begin{cases}\left[x_{s,k}+\alpha U'\left(x_{s,k}\right)\right]_s & A_{s,k}=1,\sum_{s\in S}x_{s,k}\le \rho I\\ \left[\beta x_{s,k}\right]_s & A_{s,k}=1,\sum_{s\in S}x_{s,k}>\rho I\\ 0 & A_{s,k}=0\end{cases} \tag{4-6}$$

α、β 和 ρ 是 4.2.2 节中利用遗传算法得到的带宽分配参数，它们在一个调度周期内保持不变。$[\bullet]_s$ 表示一个投影运算，计算如下

$$[x]_s=\min\left(B_s,\max\left(b_s,x\right)\right) \tag{4-7}$$

且有

$$U(\cdot)=\sum_{s\in S}U_s\left(x_s\right)$$

$$U'\left(x_s\right)=\frac{\partial U(\cdot)}{\partial x_s}$$

一个经常用到的效用函数是

$$U_s\left(x_s\right)=\mu_s\ln\left(1+x_s\right),\ \forall s\in S$$

其中 μ_s 是应用 s 的权重。

可见，给定带宽分配的参数 α、β 和 ρ，就可以根据（4-4）到（4-7）为各个应用分配带宽，并且可以按照（4-2）和（4-3）计算数据吞吐量。

本质上来说，这里提出的带宽分配方案，是一种“加性增乘性减”（Additive Increase Multiplicative Decrease，AIMD）的方法，类似于 TCP 中的拥塞控制。这种方法的主要特征是它的存储感知性，即当存储中的数据量达到其上限时，数据传输就将停止，以避免数据溢出。这种方法也是处理感知的，因为处理能力可以通过存储占用情况反映出来。这种方法使数据可以按需供应，同样也可以避免拥塞。图 4.7 给出了三种数据供应（即带宽分配）方案及其占用的存储空间，从上到下分别是连续数据供应且没有及时清除处理过的数据、连续数据供应且及时清除处理过的数据、按需数据供应且及时清除处理过的数据。这里的按需数据供应就是本节介绍的存储感知的迭代式的带宽分配算法。从图 4.7 中可见，存储感知的带宽分配算法只需要有限的存储空间，且存储中的数据量

不会随时间无限增长，总是在一个有限的范围内变化；而其它两种数据供应方案都要占用大量的存储空间，随着时间持续延长，这是不可取的。

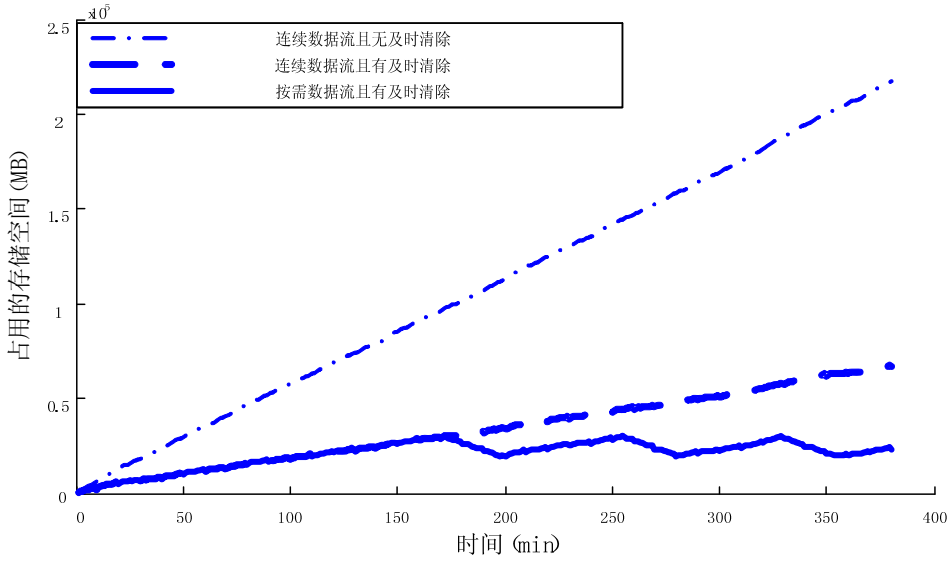


图4.7 存储感知的带宽分配

在工程实践上，GridFTP 被用来尽量逼近迭代式算法所给出的带宽分配方案。在很大程度上，GridFTP 的性能取决于其控制参数，如并行度等，但是 GridFTP 本身并没有指出如何根据当前网络状况合理的配置这些控制参数。有些文献[100][101][102]给出了一些自动调节的机制，但是它们仅限于调整 TCP 的并行连接数。然而，GridFTP 的性能同样受到 TCP 缓冲区大小的影响，所以，TCP 缓冲区大小也应该根据网络状况动态配置。文献[103]通过整合多个 TCP 连接的连续时间模型，给出在稳定状态下 GridFTP 的连续时间模型，其有效吞吐量（所有的 TCP 连接的有效吞吐量之和）可以表示如下：

$$\overline{G}^{*}=\min\left(\frac{NW}{D},\frac{N(1-p^{*})}{2D}\left(-3+\frac{\sqrt{6+21p^{*}}}{\sqrt{p^{*}}}\right)\right) \tag{4-8}$$

其中 N、W、D 和 p*分别是并行 TCP 连接数、TCP 缓冲区大小、往返时间和丢包概率。括号里面的两部分分别表示 W 小于或大于每个 TCP 连接的带宽延迟积（Bandwidth-Delay Product，BDP）[104]时的有效吞吐量。带宽延迟积是指链路的容量（比特/秒）和它的端到端的延迟（秒）的乘积，它等于任意给定时刻正在传输（即已经发送出去但是还没有被接收）的最大数据量，它也描述了发送端应该保存在内存中以便重发的最大数据量。更进一步，p*是被 B、D 和 N

决定的：

$$p^* = \left(-2 + \frac{2BD}{N} + \frac{2}{3}\left(\frac{BD}{N}\right)^2 \right)^{-1}$$

其中 B 是瓶颈连接处的可用带宽。实际上，D 和 B 可以通过一些工具测量[105]，但是不能在 GridFTP 的参数中设置，所以，有效吞吐量是 N 和 W 的 函数，即

$$\overline{G}^* = g(N,W)$$

理想情况下，W 的最优设置为

$$W = BD$$

这时候，瓶颈连接处的可用带宽得到了充分利用，并且没有数据包丢失，所以

$$\overline{G}^* = NB$$

然而，由于路径上还有其它数据传输，瓶颈连接处的带宽也是随时变化的，这种理想的情况几乎是不可能实现的。在实践上，TCP 的缓冲区大小不应该太大，以避免占用太多的内存。所以，W 应该尽量大，但是不能大于网络的带宽延迟积。TCP 的并行度可以相应的调整。

总之，本文通过调整 GridFTP 的并行 TCP 连接数和 TCP 缓冲区大小来获得迭代式算法给出的带宽。在工程实现上，为了减少带宽分配的震荡，一个调度周期被分成几个次周期。在一个次周期中，迭代式算法得到的带宽分配方案将被它们的平均值取代，也就是说，在一个次周期中，分配给应用的带宽是固定的。有了测量到的 D 和 B，给定表 4.2 中的 W，就可以计算并行连接数 N，以产生有效吞吐量，尽量逼近次周期中的固定带宽，如图 4.8 所示。值得注意的是，这里并不总是试图获得最大的有效吞吐量，而是尽量逼近每个次周期中的带宽的平均值。同样需要注意的是，由公式（4-8）获得的是在稳定状态下的有效吞吐量，实时的有效吞吐量总是由于一些随机因素而有一些波动，如图 4.9 所示。可见，实际分配的带宽总是在迭代式算法得到的带宽上下变化，两者之间有一些误差。

4.4　实验结论与算法性能评估

为了验证本章中提出的粗粒度的资源管理和调度算法，本节设计实现了一个实验：一些数据流应用在不同的时刻被提交到计算池中，本资源管理和调度

算法周期性的给出调度方案。实验结论说明了调度算法的有效性。

表 4.2 GridFTP 的参数

次周期	*D* (s)	*W* (kb)	*N*	*B* (kbps)
1	2.5	700	11	20000
2	1.5	512	4	
3	1.8	400	16	
4	0.8	256	9	
5	1.2	256	14	
6	0.9	256	9	
7	1.4	400	8	
8	1	128	20	
9	1.2	200	13	
10	0.9	1024	3	

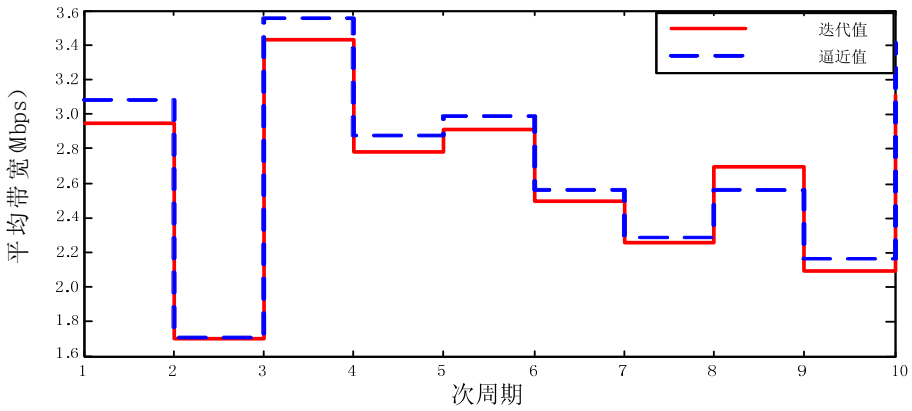


图4.8 迭代式算法和逼近算法的平均带宽

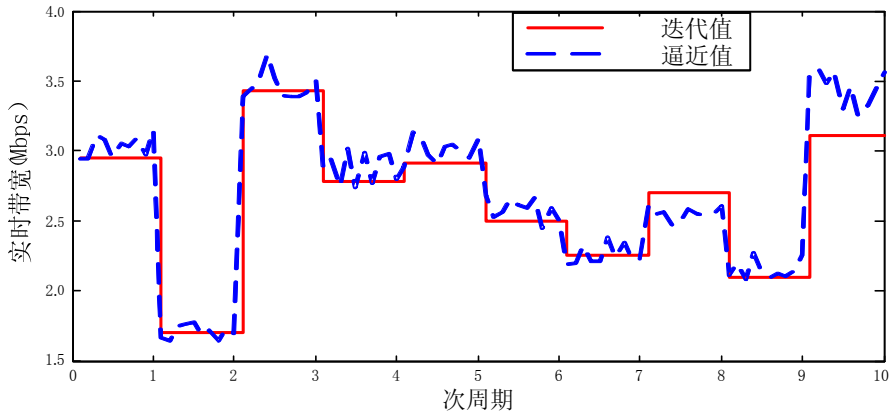


图4.9 迭代式算法的平均带宽和逼近算法的实时带宽

4.4.1　实验设置

实验网格共有 16 个处理器，30M 入口带宽，一共有 30 个应用在不同的时刻被提交到这个网格。根据各自的特点，处理器被分成 3 组。这里，每个应用需要一个处理器。实验持续了 10000 秒，每秒作为一个最小时间粒度。资源调度周期为 200 个时间单位，即每 200 秒调度器就会根据当前的资源和应用情况，给出新的调度参数。

4.4.2　实验结论

图 4.10 和图 4.11 给出了资源利用的详细信息，其中两个相邻的包络线之间的差值分别表示分配给相应应用的存储和带宽。从图 4.10 中可以看出，由于采用了所谓的库存策略，各个应用占用的存储只是在一定的范围内变化，即使要处理的数据量十分庞大且同时运行的应用很多，占用的存储也不会急剧增加，这是本算法的一个非常优良的特点。当一个应用结束，其占用的存储就会被释放，在下一个调度周期，这些存储就可以重新分配。

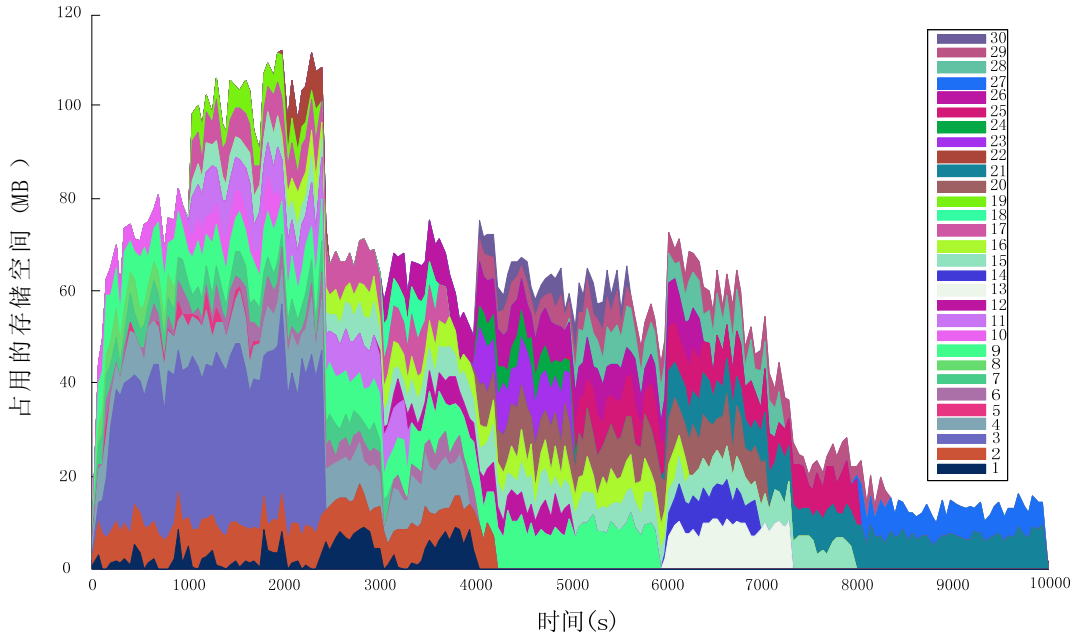


图 4.10　实验结论——存储使用情况

如图 4.11 所示，带宽的使用相当经济，或者说，带宽是按需分配的。更准确的说，带宽是根据处理速度和网络限制来分配的。这样，数据供应就可以得到保证，且不会浪费带宽，有利于缓解网络带宽的紧张状况。而且数据供应是

存储感知的，即数据供应是受存储使用和库存策略的控制的，以防止数据溢出。数据供应可能是间断的而不是连续的，从图 4.11 中可以看出，在某些时刻分配的带宽是零，表示数据供应是暂停的。

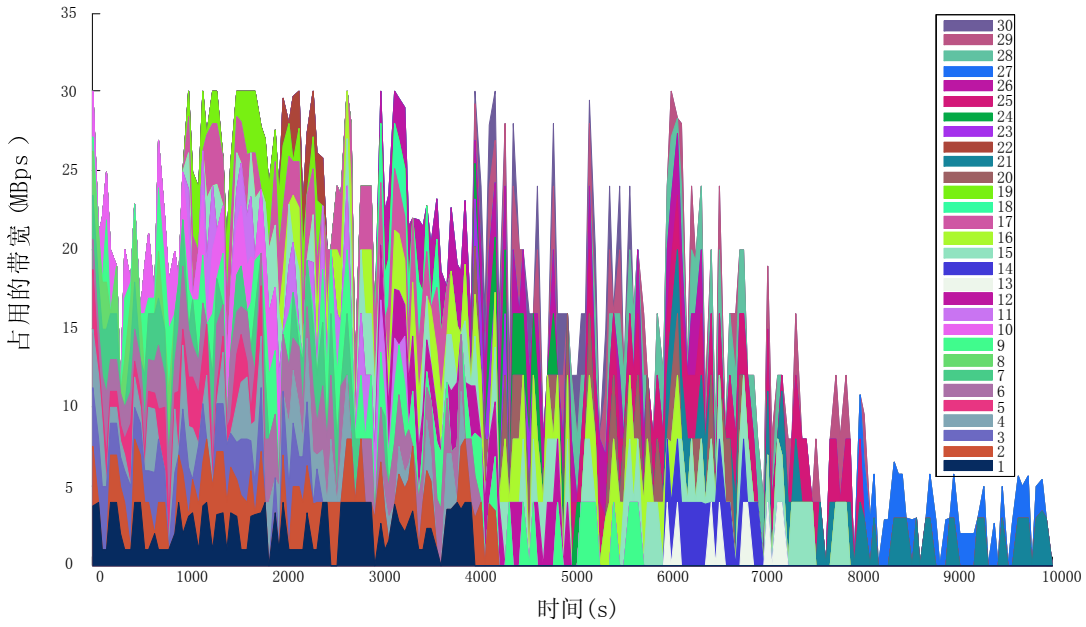


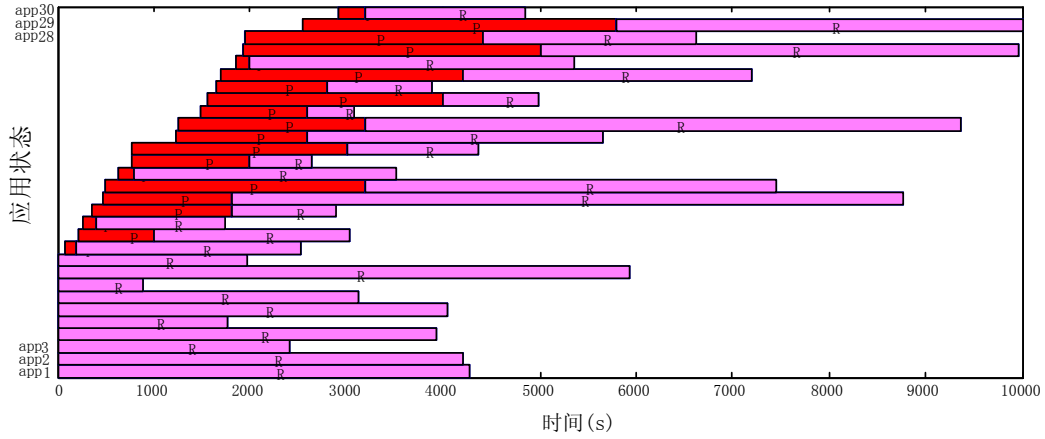
图 4.11 实验结论——带宽使用情况

4.4.3 算法性能评估

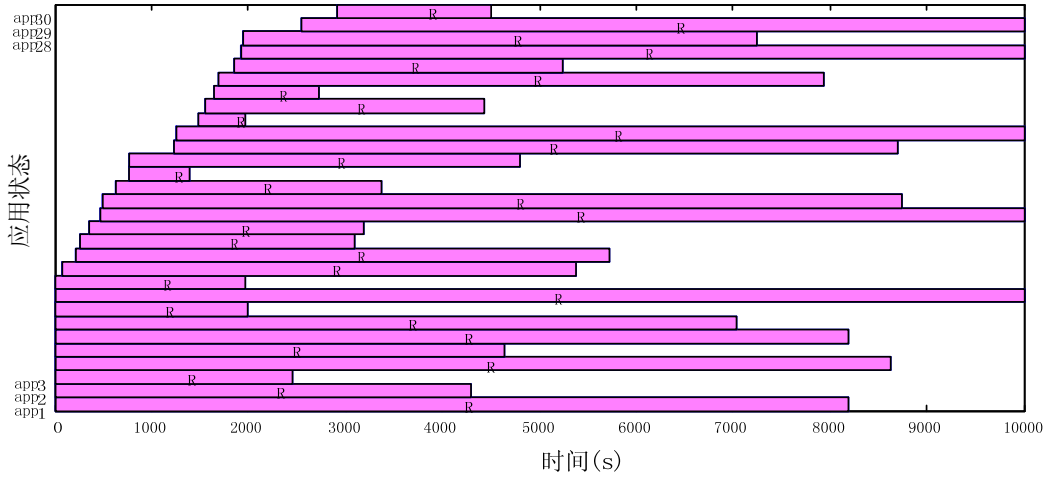
(1) 接纳控制

网格中的资源是有限的，并且每个数据流应用都有自己的资源要求，如果网格中同时运行太多的应用，将会导致应用间的激烈的资源竞争，使各个应用都不能得到充足的资源，尤其是很难保证多种资源的同时满足，从而导致整体处理效率低下。因此，接纳控制是必要的，实验结论也说明了这一点。

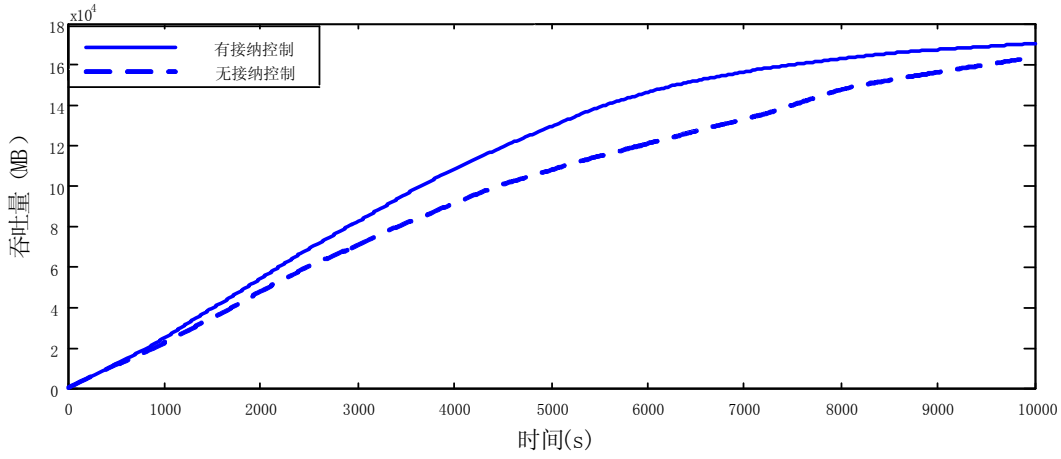
接纳控制会缩短应用的完成时间，如图 4.12（a）和 4.12（b）所示，其中横条上的字母 P 和 R 分别表示应用正在等待（Pending）和应用正在运行（Running）。有无接纳控制两种情况下，运行完成的应用分别有 29 和 25 个。更重要的是，没有接纳控制时，大多数应用的完成时间要比有接纳控制时长，这显然是不利于保证服务质量的。另外，如图 4.12（c）所示，接纳控制可以带来更高的数据吞吐量。由此可见，接纳控制在保证服务质量和提高吞吐量方面具有重要作用。



（a） 有接纳控制时各个任务的状态



（b） 没有接纳控制时各个任务的状态



（c） 吞吐量的比较

图4.12 性能评估——接纳控制

⑵处理器利用率

一个应用开始在一个特定的处理器上运行之后，并不意味着处理器会一直繁忙。由于这些应用依赖于数据供应，如果本地存储中没有足够的数据可用，处理器就将处于空闲状态。因此，更高的处理器利用率意味着更好的整体性能。如图 4.13 所示，三种资源各自独立调度时，处理器的利用率较低，这是因为三种资源不能相互协调配合，而本文提出的综合调度方案的处理器利用率较高。

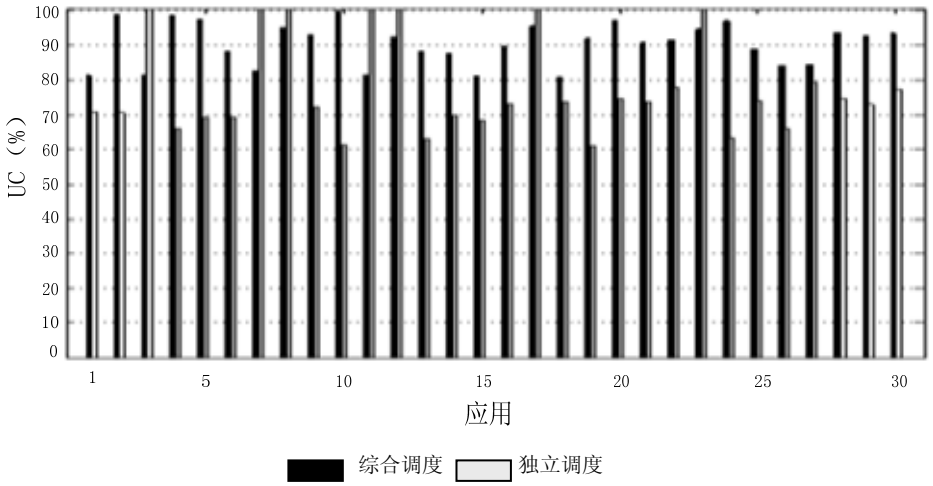
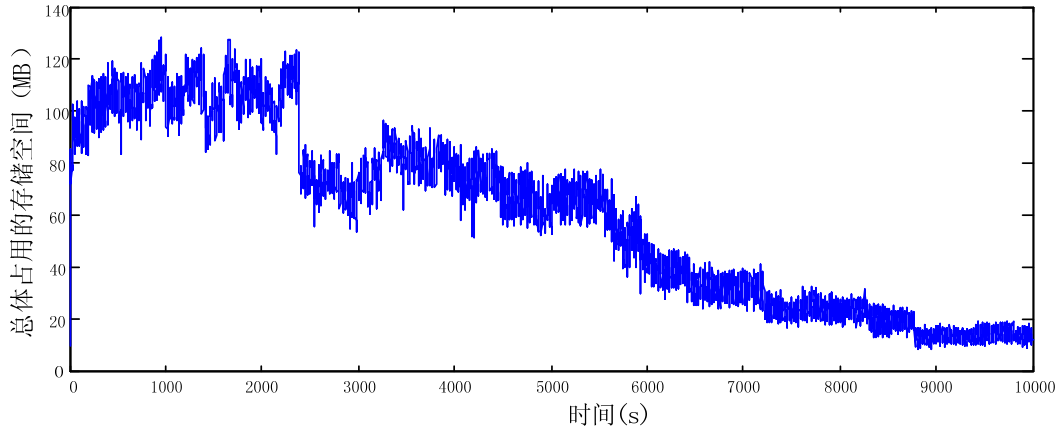


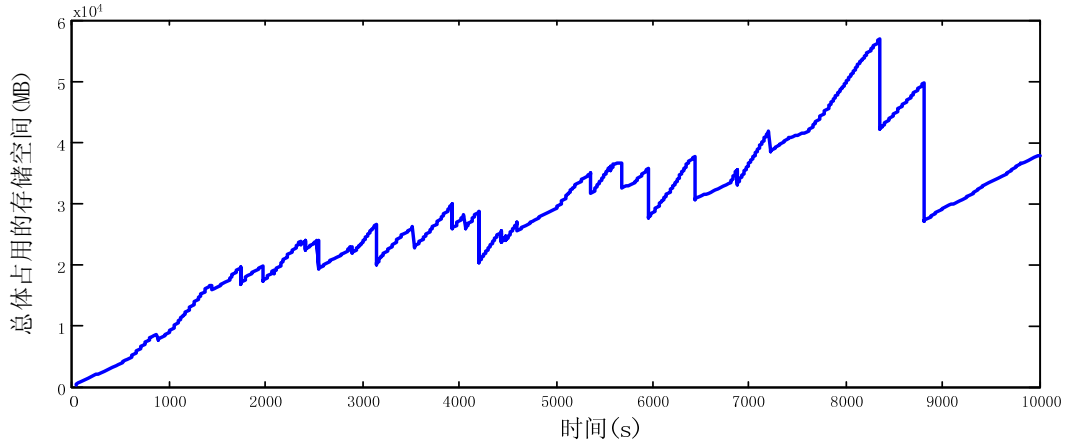
图4. 13　性能评估——处理器利用率

⑶存储感知性

本章中算法的另一个特点就是数据供应是存储感知的，也就是说，数据传输是由存储的使用情况控制的，而不是自发的。这里的基本原则就是供应“恰恰足够”的数据。数据供应可能是不连续的，大量的数据处理也仅仅需要较小数量的存储，如图 4.14（a）所示。



（a）　存储感知的数据传输



（b） 非存储感知的数据传输

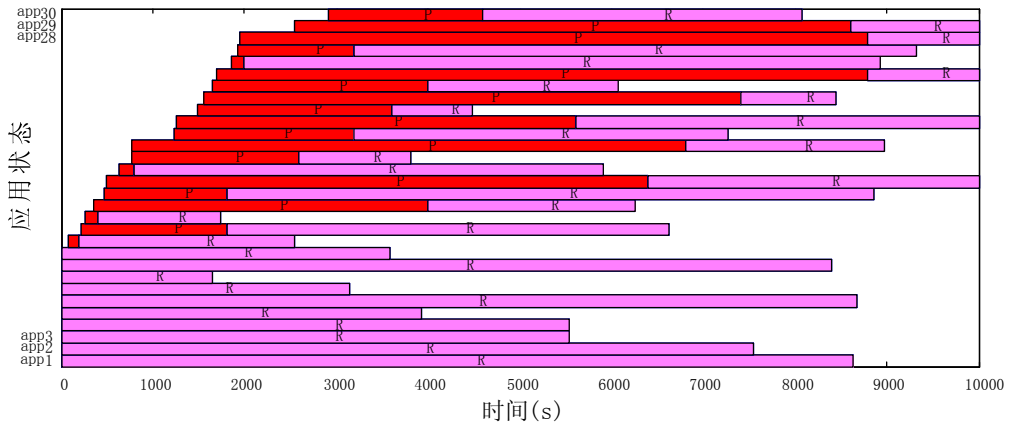
图4. 14 性能评估——存储利用率

如果可用存储足够大，且数据供应是连续的，占用的存储量就可能很大，如图 4.14（b）所示。对数据流应用而言，有了存储感知的数据传输，小的存储也可以实现很大的吞吐量。

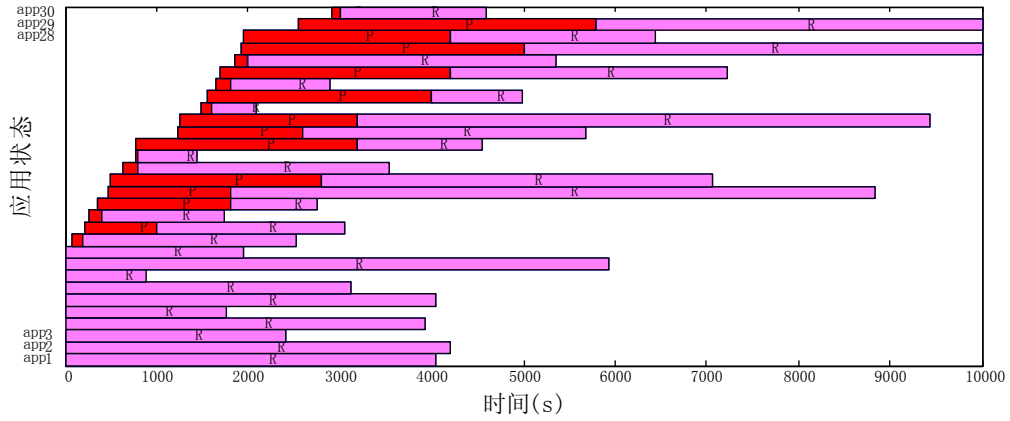
⑷迭代式带宽分配

带宽被分配给各个应用以保证其数据供应。带宽分配是一个迭代式的过程，自适应于可用带宽和应用需求，其参数由遗传算法得到。为了验证这个迭代式的算法，做了另外一个实验，其中的带宽是平均分配的，而不考虑各个应用的需求。如图 4.15（a）所示，当总的可用带宽比较小，即 I=30Mbps 时，采用平均分配带宽的方法，只有 25 个应用得以完成。在图 4.15（b）中，可用带宽增加到 40Mbps，此时平均分配带宽的方案可以为每个应用分配足够的带宽，其结论与图 4.12（a）相近，这说明迭代式带宽分配方法可以利用较小的带宽得到更好的性能。

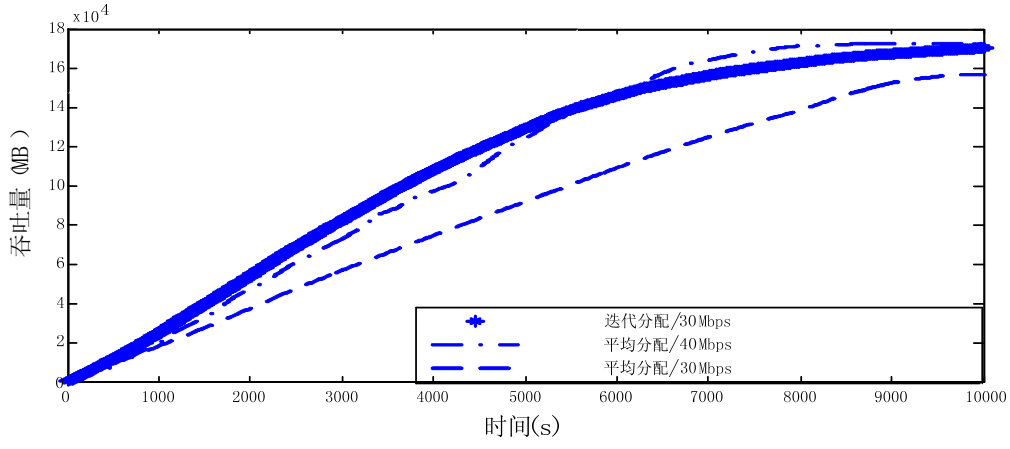
迭代式方法和平均分配的方法得到的数据吞吐量如图 4.15（c）所示。可见，迭代式方法可以得到更高的吞吐量，即使可用带宽比较小。这是因为迭代式方法是存储和处理感知的，所以在一定程度上，这种方法是智能的。在平均分配带宽的方法中，一些应用得不到足够的数据供应，而另外一些应用得到了过多的带宽，这影响了整体的处理效率，导致数据吞吐量较小。这个结论也从另外一个侧面说明，在数据流应用中，计算资源和带宽资源需要协调分配，它们共同决定了应用的处理速度和吞吐量。



（a） 平均分配带宽时各个任务的状态 （*I*=30）



（b） 平均分配带宽时各个任务的状态 （*I*=40）



（c） 吞吐量的比较

图4.15 性能评估——带宽分配

4.5　小结

本章介绍了粗粒度的资源管理和调度算法。从根本上来说，本算法包含两个部分，分别负责计算资源和带宽资源的分配。但是这两部分不是彼此独立的，而是相互联系的。计算资源的分配由 Condor 完成，各个应用最终获得计算资源的多少取决于计算资源本地的调度策略，不在本管理和调度算法的控制范围内。带宽资源则是根据一个迭代式方法分配的，其中的参数是由遗传算法基于总体吞吐量最大的原则确定的。遗传算法在计算总体的吞吐量时，需要各个应用的处理速度的信息，而这是通过分形预测得到的。

第 5 章　细粒度的数据流资源管理和调度

在网格环境中，由于计算资源本地自治的缘故，网格调度程序不可能完全拥有对资源的控制权。这样，作业一旦提交给资源本地管理机构，还需要本地作业调度器的二次调度。在作业执行结束之前，作业调度需要服从本地调度。除非网格系统要求迁移作业或取消作业，否则作业结束时间取决于当时资源本地的具体情况。因为在本地需要调度的作业既有网格作业，也有本地作业，网格作业和本地作业的数目都可能不止一个，这就导致了网格作业调度不易保证应用的 QoS。

应用需要对资源的精确控制，以便获得可预测、可控制的性能，在工作流应用中尤其如此[106]，为此本文提出了基于虚拟化的细粒度的网格资源管理和调度。虚拟化技术为每个应用建立相互隔离的、独享的运行环境，使为应用提供可预测、可保证的性能成为可能。这里需要一种自动的、灵活的资源分配机制，使各个应用可以获得“恰恰足够”的资源，既可以保证其运行效率，又不会导致资源的浪费。自动控制技术是一种很自然的选择，但是传统的自动控制方法需要被控对象的清晰的（至少是近似的）数学模型。通过后面的分析可以看到，在数据流应用中，由于计算资源和带宽资源（即数据处理能力和供应能力）的耦合，不太容易获得这样的数学模型，这给传统的自动控制法带来了一些困难。但是一些智能控制方法，却不一定需要这样的数学模型，从而为问题的解决提供了另一种可行的方案。本文采用的模糊控制就是这样的一种智能控制方法。

现在的情况是，由于缺乏灵活的、可自由扩展的资源管理和调度，一方面网格资源非常紧张，一些应用往往不能及时得到需要的资源，制约了应用的运行效率；另一方面，一些网格用户大量占用超出需要的资源，进一步导致资源紧张，也白白付出经济代价。因此，本文提出的细粒度的资源调度方法具有理论和实践上的双重意义。

5.1　问题描述

网格上的资源是共享的，并发的大量应用会导致资源紧张，所以一个用户最好不要占用超过需要的资源，何况有时用户需要为他们申请的资源付费，即使他们根本没有实际利用这些资源。所以，对资源的提供者和使用者来说，都

需要细粒度和按需的资源分配。对数据流应用来说，计算资源和带宽资源应该在细粒度上达到平衡，既可以获得理想的吞吐量，又可以保证资源的利用率。由于计算系统的动态随机性和负载的时变性，人工分配资源的方式不能胜任，这里需要一种自动分配机制。

反馈控制（Feedback Control）已经被应用到计算系统中并取得了一些有价值的成果。反馈控制器观测设定值（又称为参考值）和受控对象的输出之间的误差，以此激活其控制算法，产生控制规律，并改变受控对象的输出，直到系统进入预定的状态。在传统的反馈控制中，需要用一个或多个微分方程建立被控对象的数学模型，以描述被控对象对输入的响应。但是在数据流应用中，由于输入（数据处理能力和供应能力）之间的耦合，很难建立这样的数学模型。幸运的是，模糊控制不需要这样的模型，它可以利用一些模糊规则来描述人类的经验知识，直接为控制器建立从输入到输出的非线性映射，大大降低了控制系统的设计难度。

由于虚拟化技术的发展，现在已经可以为应用分配细粒度的计算资源：在一个主机上同时配置多个独立运行的虚拟机，每个虚拟机上运行一个应用；虚拟机可以动态配置，以便为应用按需分配计算资源。虚拟机的计算资源（主要是它获得的物理 CPU 周期的份额）可以精确的调节，这就是细粒度的含义。本文利用虚拟化技术为每一个数据流应用建立一个虚拟机，使每一个应用可以“独享”一部分资源；利用模糊控制技术动态的调整虚拟机的资源配置，使虚拟机的资源利用率达到设定的水平，从而保证应用的顺利运行，又避免资源的浪费。

5.1.1　问题的重要性

与大量的应用相比，今天的网格资源仍然是稀缺的。另一方面，网格上的资源并没有得到很好的管理和调度，没有实现按需的细粒度的资源调度和分配，导致了资源的浪费和低效利用。在数据流应用中，数据从远端的数据源通过网络传输到本地的处理节点，传输带宽是由多个应用分享的，所以在大多数情况下，每个应用可以获得的带宽是有限的。因此，由于没有足够的数据供应，为一个数据流应用分配过多的计算资源会导致其低效利用。安迪·格鲁夫（Andrew S·Grove）在 20 世纪 90 年代指出，电信部门的带宽每一个世纪才增加一倍，这就是所谓的格鲁夫定律。格鲁夫定律主要是对电信部门提出批评，但它也表达了一个基本的事实：在电脑应用历史上，通讯网络传输能力的提升远远落后于

电脑处理能力。正如格鲁夫的断言所揭示的，人们长期以来一直公认：通讯带宽的缺乏是一个阻碍因素，使人们无法有效进行高效率的电脑运算。在数据流应用中也是如此，可用的带宽往往不能满足计算资源对数据的需求。但是，Google 公司首席执行官埃里克·施密特（Eric Schimidt）在 1993 年（当时他是 Sun Microsystems 公司的首席技术官）曾预言：“当网络的速度与微处理器一样快时，电脑就会虚拟化并通过网络传播”。因此，可以预见，随着网络速度的进一步提升，数据流应用有着广阔的发展前景。

另一方面，如果仅仅将一个应用分配到一个处理器上，并由本地调度器负责进程调度，那么很难保证应用可以得到适当数量的 CPU 周期，这可能导致应用的处理效率很低甚至导致可用带宽的浪费。从资源利用率或吞吐量的角度考虑，这两种情况都应该避免。所以，应该为每个应用分配恰当数量的、与可用带宽匹配的计算资源，以提高数据处理效率并保持一定水平的资源利用率。

本文第 4 章提出了一个粗粒度的资源调度和分配算法。在一些应用场景中，这个粗粒度的调度和分配算法的确可行，但是它很难保证应用的 QoS。而且在网格经济学中，资源使用者需要为他们占用的资源付费，即使这些资源没有得到充分利用，所以资源使用者希望能够按需的分配资源。另一方面，如果资源以更细的粒度进行管理和调度，就可以避免不必要的资源浪费，这有助于缓解网格上的资源紧张状况。所以，对数据流应用来说，细粒度的按需资源分配是需要的，甚至是必不可少的。

5.1.2　吞吐量和计算资源利用率的定义

数据流应用的两个关键 QoS 要求是吞吐量和计算资源利用率，本节给出它们的定义。在时刻 t, 对数据流应用 i, 存储中的数据量记作 $Q_i(t)$。如果 $Q_i(t)$大于某个值，数据处理就将进行，否则计算资源将会空闲。$Q_i(t)$是由数据供应和处理共同决定的，其变化可由下列微分方程表示：

$$Q_i'(t) = r_i(t) - d_i(t) \tag{5-1}$$

$$Q_i(0) = 0$$

其中，$Q'_i(t)$、$r_i(t)$和 $d_i(t)$分别表示 $Q_i(t)$的导数、应用 i 的传输带宽和处理速度。

如文献[96]所示，要处理的数据集是由大量的小文件组成的。数据是按块传输和处理的，一个块中包含一定数量的小文件。为应用 i 定义一个状态指示

器，记作 R_i，如果存储中有可以处理的块，R_i 的值就被设置为 1，否则为 0。所以 $d_i(t)$可以描述如下

$$d_i(t)=\begin{cases}0 & R_i=0\\ >0 & R_i=1\end{cases}$$

下面给出一些定义：

实际处理速度（Realistic Processing Speed，RPS）：给定数据供应和计算资源分配方案，应用 i 实际获得的处理速度，就是这里的 $d_i(t)$。

理论处理速度（Theoretic Processing Speed，TPS）：假定存储中一直有足够的数据，分配给应用 i 的计算资源可以产生的处理速度，记作 $p_i(t,C_i(t))$，其中 $C_i(t)$是在 t 时刻分配给应用 i 的计算资源，主要是指 CPU 周期的比例，即后面所说的 Cap。$p_i(t,C_i(t))$和 $C_i(t)$之间的关系将由系统辨识得到。

实际吞吐量（Realistic Throughput，RTP）：给定数据供应和计算资源分配方案，应用 i 在一个调度周期内实际处理的数据量。

理论吞吐量（Theoretic Throughput，TTP）：假定存储中一直有足够的数据，给定计算资源分配方案，应用 i 在一个调度周期内可以处理的数据量。

对带宽和计算资源的调度是周期性进行的，以应对资源和应用的最新状态。假设一个调度周期的长度是 L，对第 h 个调度周期

$$d_i(t,C_i(t))=\begin{cases}0 & R_i=0\\ p_i(t,C_i(t)) & R_i=1\end{cases}$$

$$TTP_{i,h}=\int_{(h-1)L}^{hL}p_i(t,C_i(t))dt$$

$$RTP_{i,h}=\int_{(h-1)L}^{hL}d_i(t,C_i(t))dt$$

其中，$d_i(t,C_i(t))$表示在时刻 t，给定计算资源 $C_i(t)$，应用 i 的实际处理速度；$TTP_{i,h}$ 和 $RTP_{i,h}$ 分别表示第 h 个调度周期内应用 i 的理论吞吐量和实际吞吐量。由 (5-1) 有

$$\begin{aligned}RTP_{i,h}&=\int_{(h-1)L}^{hL}\left(r_i(t)-Q'(t)\right)dt\\&=\int_{t=(h-1)L}^{hL}r_i(t)dt+Q_i((h-1)L)-Q_i(hL)\end{aligned}$$

定义第 h 个调度周期内应用 i 的计算资源利用率 $UC_{i,h}$ 如下

$$UC_{i,h}=\frac{RTP_{i,h}}{TTP_{i,h}} \tag{5-2}$$

即

$$UC_{i,h}=\frac{\int_{t=(h-1)L}^{hL} r_i(t)dt+Q_i((h-1)L)-Q_i(hL)}{\int_{t=(h-1)L}^{hL} p_i(t,C_i(t))dt} \tag{5-3}$$

表示分配的计算资源的利用程度。由于采用了库存策略，一个调度周期内存储占用的空间是有限的，与应用处理的大量数据相比更是如此。因此，（5-3）可以改写如下

$$UC_{i,h}=\frac{\int_{t=(h-1)L}^{hL} r_i(t)dt}{\int_{t=(h-1)L}^{hL} p_i(t,C_i(t))dt} \tag{5-4}$$

$\mathrm{RTP}_{i,h}$ 可以由另外一种方式定义

$$RTP_{i,h}=\int_{\Omega_{i,h}} p_i(t,C_i(t))dt \tag{5-5}$$

其中 $\Omega_{i,h}$ 表示 R_i=1 的时间段的长度，计算资源利用率可以用另外一种方式定义

$$UC_{i,h}=\frac{\Omega_{i,h}}{L} \tag{5-6}$$

注意（5-6）暗含着这样一种假设：给定计算资源，$\mathrm{TPS}_{i,h}$ 将在一个调度周期中保持恒定。

另一方面，资源利用率也可以定义为 RPS 和 TPS 的比值。给定带宽分配方案，分配合适数量的计算资源将有助于使 RPS 尽量接近 TPS。很明显，过多的计算资源将使 TPS 远远大于 RPS，这意味着计算资源的有效利用率很低。如果可用的带宽是有限的，过多的计算资源将导致 RPS 在很多时刻为 0。因为缺少足够的数据来处理，所以过多的计算资源将导致无谓的浪费。RPS 为 0 的时候越多，计算资源的利用率越低，也就应该分配越少的计算资源。这与（5-6）是一致的。数据供应和处理的内在联系使得必须按需分配计算资源，以使 RPS 接近于 TPS。

由（5-2）到（5-6）可以推断，资源利用率和吞吐量是由带宽和计算资源

共同决定的。第 4 章介绍的迭代式带宽分配算法是处理感知的，也就是说，数据是根据处理能力传输到本地的，所以计算资源不足将浪费带宽并导致吞吐量较低。另一方面，超过需要的计算资源可以充分利用带宽并获得较高的吞吐量，但是这会导致资源利用率较低。因此，对数据流应用来说，带宽和计算资源有必要达成一个均衡。但由于数据供应和处理之间的耦合，系统建模比较困难，即计算资源和带宽资源与吞吐量或计算资源利用率之间的定量关系比较复杂。在动态环境中，数据流应用资源分配需要一个简单有效的方法，详见 5.2.4。

5.1.3　技术解决方案

由于虚拟化技术的发展，可以使每个应用运行在一个独立的虚拟机上，以提供稳定而可预测的性能。为了实现 5.1.2 中定义的吞吐量和资源利用率两方面的目标，虚拟机资源需要进行动态配置，为此需要设计一些自动机制，在不需要人工参与的情况下，实现细粒度的按需资源分配。

如果将一个数据流应用看作一个被控对象，则其输入为分配给它的计算资源和带宽资源，而其输出则是吞吐量和计算资源利用率。由于采用了虚拟化技术为每个应用建立了一个独立运行的虚拟机，分配给数据流应用的计算资源就表现为虚拟机的计算资源配置，主要是虚拟机的时钟频率，即虚拟机获得的物理 CPU 的份额。这里的带宽分配依然采用了第 4 章中介绍的迭代式分配方法。由第 4 章的分析知道，对数据流应用而言，其吞吐量和计算资源利用率是由其获得的计算资源和带宽资源共同决定的，而带宽资源的分配是处理速度感知的，也就是说这种应用获得的计算资源和带宽资源是相互关联的。实际上，给定计算资源，不同的带宽分配方案将产生不同的计算资源利用率。作为一个被控对象，数据流应用的两个输入不是独立的，而是相互耦合的。一般而言，此时要进行解耦，才能获得从一个输入到一个输出的映射关系（如传递函数）。有了这样的映射关系，才可以方便的为各个输入设计控制规律，以使被控对象的输出达到预定值。但是，一般说来，解耦操作并不十分容易。实际上，在细粒度的资源调度中，主要的控制变量是数据流应用获得的计算资源，即各个虚拟机的计算资源配置，而输出是虚拟机的计算资源利用率。在一个调度周期内，给定各个虚拟机的计算资源配置，然后利用遗传算法，根据各应用的吞吐量之和最大的原则寻找合适的带宽分配参数，并迭代式的分配带宽资源。由此产生的计算资源利用率则反过来影响计算资源的分配，以使计算资源利用率达到预定

的参考值。由此可见，需要建立从虚拟机的计算资源配置到其计算资源利用率的映射关系，以此作为被控对象的数学模型，才能实施传统的自动控制。但是，由于计算资源和带宽资源的耦合以及它们对计算资源利用率的共同影响，很难建立这样的数学模型，因而很难应用传统的控制方法。幸运的是，作为一种智能控制方法，模糊控制不需要这样的数学模型，它只需要一些直观的认识，就可以有规律的改变被控对象的输入，以使其输出达到预定的参考值。

细粒度资源调度的目标是在系统特性不明确的情况下，实现吞吐量和资源利用率方面的目标。计算资源利用率低说明在大多数时刻，由于没有数据可以处理，计算资源处于空闲状态，此时的吞吐量也很低。另一方面，太高的计算资源利用率（如 100%）说明计算资源超负荷运行，此时需要更多的计算资源。在这两种情况下，吞吐量都会受到限制。计算资源利用率极高的应用应该分配更多的资源以提高吞吐量，而对计算资源利用率很低的应用，则应该减少其资源，以提高资源利用率。实际上，对模糊控制器的设计来说，这样的直观认识就几乎足够了。而对其它控制器（如比例－积分－微分控制器）来说，则至少需要一个近似的数学模型。此外，还可以看出，计算资源利用率与吞吐量也是有着内在联系的。在稳定状态，可以同时实现这两个目标。

5.2　细粒度资源调度和分配

如前所述，细粒度资源调度和分配是由虚拟化技术和模糊控制共同完成的。虚拟机由 Xen 实现，它提供了一种半虚拟化的方法；模糊控制器负责为虚拟机配置合适的资源。最后，设计了一种细粒度混合资源调度系统，协调分配计算资源和带宽资源。

5.2.1　虚拟化与Xen

为了实现吞吐量和资源利用率方面的目标，需要为每个数据流应用提供资源隔离和可预测、可控制的性能。但是，很难找到现成的配置合适的资源。幸运的是，虚拟化技术的进步发展，使按需配置资源成为可能，从而可以实现细粒度的资源按需分配。虚拟化在分布式计算环境中提供了强有力的资源抽象，将物理硬件和操作系统分开，以便提高 IT 资源的利用率和灵活性。

Xen 是一种半虚拟化技术，对设备进行了简单明了的抽象，大多数的指令

可以以原始速度运行，所以虚拟化带来的性能损失很小，虚拟机性能接近于物理机。利用 Xen，虚拟机的配置可以动态调整以优化性能。虚拟机的 CPU 称为虚拟 CPU（Virtual CPU，VCPU）。一个 VCPU 可以获得的物理 CPU 周期的份额，是由两个参数决定的，一个是权重（Weight），另一个是 Cap。Weight 是一个相对值，Cap 是一个绝对值。Weight 为 128 的 VCPU 获得的物理 CPU 周期将是 Weight 为 64 的 VCPU 的两倍。对 Cap 而言，50 表示该 VCPU 将获得的物理 CPU 周期将不超过物理 CPU 的一半。而且，将 VCPU 绑定到物理 CPU 上将显著提高性能。

5.2.2　模糊控制器概论

一个模糊控制系统以模糊逻辑为基础，其中的一些模糊概念不能表示为“真”或“假”，而是“部分真”。模糊控制系统的输入是一些逻辑变量，它们取从 0 到 1 连续变化的数值，以表示其真实度，而不是象传统的数字逻辑那样只取 0 或 1。模糊逻辑可以用人类能够理解的方式表达问题的解，因此在设计控制器时，可以利用人类的经验。有些问题人类已经成功解决，但是其数学模型很难获得，对此模糊控制提供了更简单的方法。所以，当传统的控制方法很难应用时，模糊控制便可能显示出其优越性。

一个模糊控制器（Fuzzy Logic Controller，FLC）包含输入阶段、处理阶段和输出阶段，如图 5.1 所示。输入阶段将包括 UC 和△UC（即 UC 的差分）在内的输入映射到合适的隶属函数（如图 5.2、图 5.3 和图 5.4 所示）并获得真值，这被称为模糊化。在处理阶段，这些映射被输入到规则库中，基于推理机制，激活相关规则并产生相关结果，然后这些结果会联合起来。这里规则之间的关系是“与”（AND），所以将取各个规则的结果中的最小值。最后，在输出阶段，选择合适的输出并赋予相应的隶属度，然后利用面积中心法等方法进行清晰化。这里，输出是一个比例系数（Proportional Factor，PF），将用来计算为虚拟机分配的 CPU 周期的份额，也就是 Cap, 如（5-8）所示。这里将解释有关的基本概念，帮助更好的理解模糊控制器及其原理。

论域（Scope of Discourse）是模糊控制器的输入和输出的范围。输入和输出必须分别通过量化因子（K_e 和 K_{ec}）和比例因子（K_u）映射到相关的论域，这使得模糊逻辑可以不加修改的应用到其它问题。

量化因子是模糊控制器的输入接口，负责对输入变量进行变换；比例因子

是模糊控制器的输出接口，负责对输出变量进行变换。量化因子和比例因子分别对清晰值进行比例变换，其作用主要是使变量按一定比例进行放大或缩小，以便跟相邻模块很好的匹配。这种变换会对整个系统的工作性能产生一定的影响。

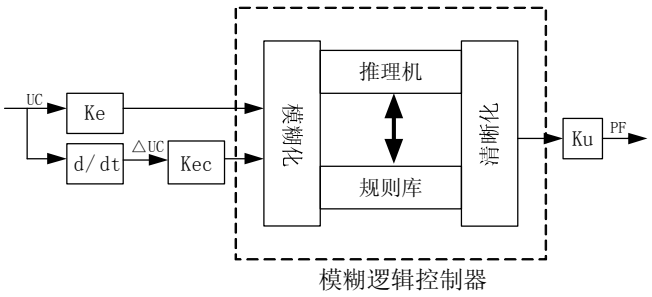


图5.1 模糊控制器原理图

为了使输入的清晰值能够与语言表达的模糊控制规则相匹配以进行模糊推理，必须把它们变换成模糊量，即模糊子集。把这些输入的清晰值映射成模糊子集及其隶属函数的变换过程，称为模糊化（Fuzzification）。

经过模糊逻辑推理之后输出的结论是模糊量，也就是模糊子集，用它们不能直接推动执行机构进行控制，需要变成清晰量。把模糊量变成清晰量的过程，称为清晰化，或去模糊化（Defuzzification）。面积中心法经常作为一种清晰化的方法用来计算控制信号，如（5-7）所示。

语言变量描述了模糊控制器的输入和输出。这些语言变量是对人类处理存在于大多数计算机系统中的不确定性的方法的一种自然的模仿。本章中的语言变量包括 UC、△UC 和 PF。

语言值用来描述语言变量的特性。非常低（Very Low）、低（Low）、中等（Medium）、高（High）和非常高（Very High）是 UC 的语言值，而△UC 和 PF 的语言值是 NB、NM、NS、ZE、 PS、PM 和 PB，其中，N、P、B、M、S 和 ZE 分别是负（Negative）、正（Positive）、大（Big）、中等（Medium）、小（Small）和零（Zero）的缩写, 它们的结合说明了真实的程度。

语言规则形成了一个“IF-THEN”形式的规则集合，将模糊控制器的输入映射到输出，即指导模糊控制器的行为。这些规则是由语言变量定义的，不同于传统的数字控制器。例如，一条语言规则为：如果 UC 为高（High），并且△UC 是 NB，那么 PF 就是 NB。规则库是控制器的一部分，存储了一系列的 IF-THEN

规则，说明如何根据模糊化了的 UC 和△UC 获得 PF。

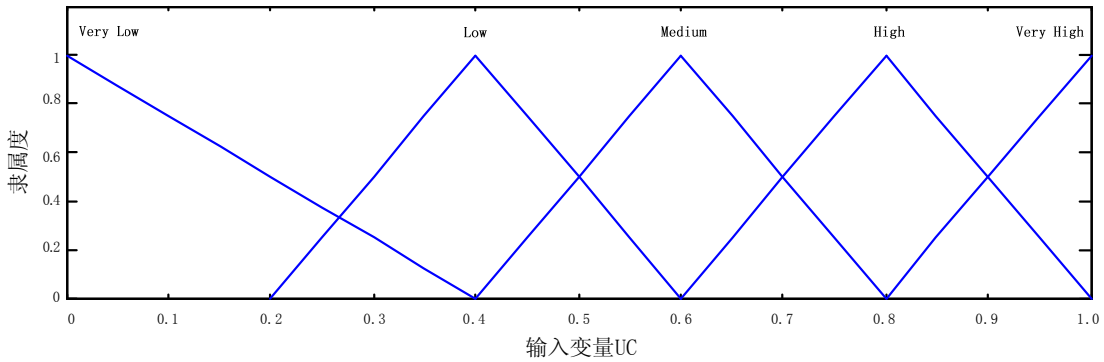


图 5.2 输入 UC 的三角形隶属函数

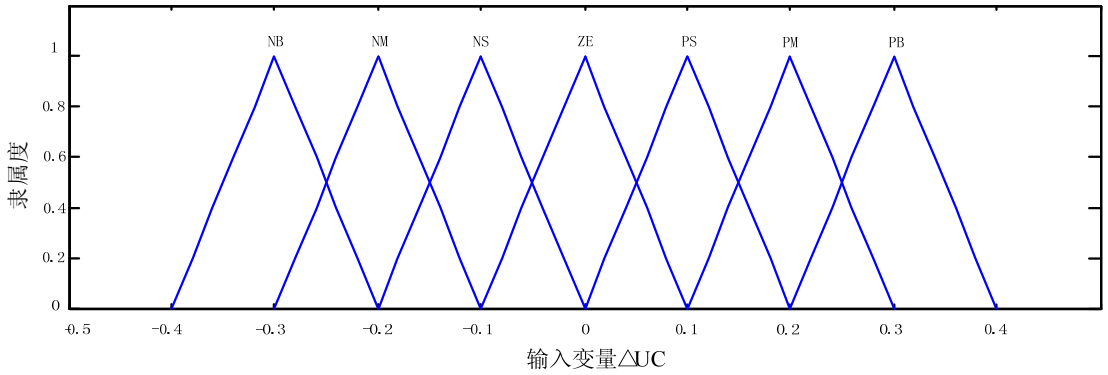


图 5.3　输入△UC 的三角形隶属函数

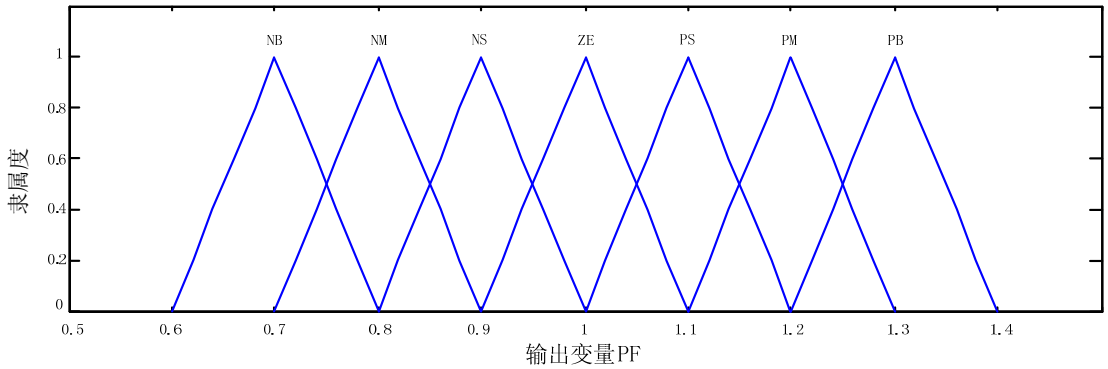


图 5.4 输出 PF 的三角形隶属函数

隶属函数说明了 UC、△UC 和 PF 的确定性对应的特定的语言值。除了 UC 的非常低（Very Low）这样一个语言值，其余语言变量的语言值均采用了对称三角形隶属函数，且相邻的隶属函数有 50%的重叠，如图 5.2、图 5.3 和图 5.4 所示。与传统的集合理论不同，在模糊集合中，隶属函数不是二值函数，而是

连续的，以处理不确定性。一个模糊输入或输出可能属于多个模糊集合，其隶属度各不相同。这里，一个模糊输入或输出最多属于两个模糊集合。

基于模糊化的 UC 和△UC，图 5.1 中的推理机制决定在第 k 个采样时刻，有哪些规则将被激活。为了计算相应的 IF-THEN 规则中的前提（IF）的确定性，这里采用了 UC 和△UC 的确定度的最小值, 因为结果的确定度不可能大于前提。注意，不是所有的规则都会被激活，这里最多 4 条规则将被同时激活，因为 UC 和△UC 最多同时属于两个模糊集合。

模糊推理是一种近似推理，有两种基本推理方法。其一为广义取式（肯定前提）推理，推理过程为

前提　If x is A, then y is B

前提　　x is A′

———　————————

结论　　y is B′

另外一种为广义拒式（肯定结论）推理，推理过程如下

前提　If x is A, then y is B

前提　　y is B′

———　————————

结论　　x is A′

式中 A′ 与 B′ 分别为论域 U 与 V 上的模糊集合。

如果以 X 表示模糊蕴含 x is A→y is B，即 A→B，记其隶属度函数为 $\mu_{X(x,y)}$。那么广义取式推理可以表达为

$$B' = A \bullet X = A \bullet (A \to B)$$

对于广义拒式推理，则可以表达为

$$A' = X \bullet B' = (A \to B) \bullet B'$$

如果采用 Max-Min 复合运算，广义取式推理可表示为

$$\mu_{B'}(y) = \left\{ \bigvee_x \mu_{A'}(x) \wedge \mu_X(x,y) \right\}$$

而拒式推理可以表示为

$$\mu_{A'}(x) = \left\{ \bigvee_y \mu_X(x,y) \wedge \mu_{B'}(y) \right\}$$

其中“∧”与“∨”称为扎德（A. Zadeh）运算符，通常定义为取大和取小运算；

$\mu_{A(x)}$表示变量 x 对于模糊子集 A 的隶属度。

这里采用了广义取式模糊推理机制，并采用 Max-Min 复合运算。控制过程中，首先定义 UC 和△UC 到 PF 的模糊蕴含关系；根据控制实施时刻观测到的实际的 UC 和△UC，经过广义取式推理，得到 PF 对模糊子集的隶属度。

5.2.3　语言变量和模糊规则

模糊控制器的输入是观测到的资源利用率 UC 及其差分△UC。尽管有两个输入，实际上这是一个单输入系统，因为△UC 可以由 UC 差分获得：

$$\Delta UC(k) = UC(k) - UC(k-1)$$

在第 h 个调度周期，对应用 i 而言，模糊控制器的输出是一个比例系数，记作 $PF_{i,h}$。给定输入 UC 和△UC，假设与输出相关的模糊集合记作 $F_{i,h}$，其隶属函数为 $F_{i,h}(u)$，其中 $u \in U_{i,h}$ 且 $U_{i,h}$ 是相关论域，则由面积中心法，输出可按如下公式计算：

$$PF_{i,h} = \frac{\int_{U_{i,h}} F_{i,h}(u) u du}{\int_{U_{i,h}} F_{i,h}(u) du} \tag{5-7}$$

假设应用 i 的初始 Cap 为 $C_{i,0}$，在第 h 个调度周期，应用 i 的 Cap 将是

$$C_{i,h} = C_{i,h-1} + (PF_{i,h-1} - 1) CapScale, h \geq 1 \tag{5-8}$$

且初始条件为

$$PF_{i,0} = 1$$

其中 *CapScale* 是 Cap 的变化步长。$C_{i,h}$ 与第 h 个调度周期内应用 i 的理论处理速度 $p_i(t)$之间的关系可以通过系统辨识获得，这里采用了一个线性模型

$$p_i(h) = \sum_{l=1}^{d} a_l p_i(h-l) + \sum_{m=1}^{f} b_m C_{i,h}(h-m)$$

其中 d 和 f 称为模型的阶次。

模糊控制的目标是将计算资源利用率保持在高水平，这里的设置值是 80%，所以很低或极端高的利用率都是不可取的。Xen 采用了半虚拟化技术以使大多数指令以原始速度运行，但是，由于数据流应用是输入/输出密集型的（I/O Intensive），所以有一些额外的开销，即性能损失。由此，为了保证处理效率，

这里将利用率的目标设置为 80%而不是 100%，这意味着分配的 CPU 份额要比实际需要的多，以此补偿系统开销，如（5-2）或（5-6）所示。

如前所述，计算资源利用率低意味着应该减少分配的 CPU 份额以释放多余的计算资源，这不会影响最终的吞吐量；另一方面，极端高的计算资源利用率说明需要更多的计算资源以提高处理效率。所以，当计算资源利用率为非常低（Very Low）、低（Low）或中等（Medium）时，产生的 PF 应该小于 1；当计算资源利用率非常高时（Very High），产生的 PF 应该大于 1。为防止振荡，也应该注意△UC 。当计算资源利用率远离设定值时，调整可以是大幅度的；当计算资源利用率在设定值附近变化时，需要更精细的调节，如表 5.1 所示。可见，这些模糊规则非常简单，但是非常稳健。

表 5.1 模糊规则

PF		UC				
		Very Low	Low	Medium	High	Very High
△UC	NB	NB	NB	NB	NB	PB
	NM				NM	
	NS				NS	
	ZE				ZE	
	PS				PS	
	PM				PM	
	PB				PB	

5.2.4　细粒度混合资源调度和分配系统

面向数据流应用的细粒度的混合资源调度和分配系统如图 5.5 所示。混合资源分配器接受模糊控制器和带宽分配算法给出的参数，为每个数据流应用实际分配下一个周期内的资源，包括带宽和计算资源。在每一分配周期内，根据（5-2）或（5-6），可以计算得到每个数据流应用的计算资源的利用率，即图 5.5 中的 UC。UC 以及 UC 差分后得到的△UC，传递给模糊控制器，由模糊控制器据此计算一个周期内各个应用的计算资源变化的比例系数，即图 5.5 中的 PF，由（5-8）即可以获得一个周期内每个应用的虚拟机获得的 CPU 的份额，即前面所说的 Cap。带宽分配采用第 4 章中介绍的迭代式方法。不同的是，数据流应用池中的各个应用的性能，可以通过系统辨识得到的模型准确的观测，而不必使用第 4 章中介绍的分形方法加以预测。

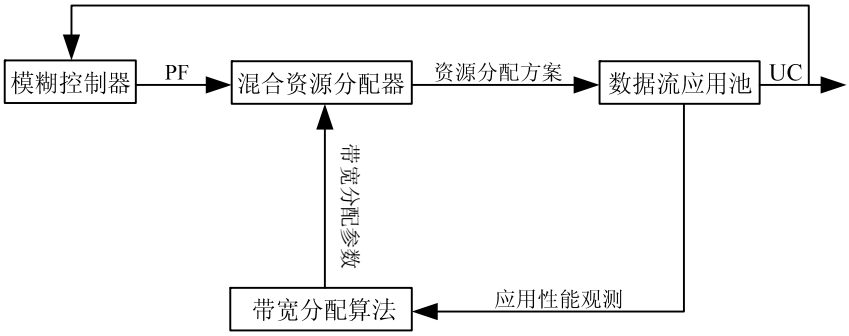


图5.5 细粒度混合资源调度和分配框图

由此可见，一个虚拟机的计算资源利用率是由其计算资源配置和其获得的带宽共同决定的。在第 h 个调度周期，对应用 i 而言，模糊控制器的输出 $PF_{i,h}$ 可以根据 $UC_{i,h}$ 通过模糊推理得到，由此就可以根据（5-8）得到一个虚拟机在该周期内的计算资源。但是，$UC_{i,h}$ 还受到该周期内带宽分配的影响，而带宽分配是处理感知的，即带宽分配受到应用的处理能力（亦即计算资源的配置）的影响。也就是说，对图 5.5 中的混合资源分配器而言，这是一个双输入单输出系统，且这两个输入是相互耦合的。这就使得不容易建立从计算资源分配（即图 5.5 中的 PF）到计算资源利用率（即图 5.5 中的 UC）的数学模型。而传统的控制方法，如比例积分微分控制，则需要从 PF 到 UC 的清晰的至少是近似的数学模型。正是在这种情况下，本文采用了模糊控制器来设计 PF 的控制规律，它不需要精确的数学模型，只要一些基本的操作经验，或 5.1.3 中所说的直观认识。实际上，如果图 5.5 中的带宽分配算法与计算资源是各自独立的，即带宽分配不是处理感知的，图 5.5 中的模糊控制器也可以根据 UC 得到合适的 PF，从而使 UC 保持在预定的水平，且应用的处理能力与数据供应能力达到均衡[108]。

与粗粒度的资源调度和分配方法一样，细粒度的资源调度只给出资源分配的相关参数，实际的资源分配是由分配器完成的。与粗粒度的资源分配不同的是，细粒度的资源分配中，计算资源的分配是通过虚拟机实现的，模糊控制器用于精确的计算每个虚拟机获得的物理 CPU 的份额，这决定了虚拟机的时钟频率。也就是说，在细粒度的资源分配中，计算资源的分配是“精确的”、可控制的，这就是“细粒度”的含义。这里，带宽资源的分配同样是通过调整 GridFTP 的相关参数，逼近带宽分配方案实现的。实际上，粗粒度的资源分配和细粒度的资源分配的差别，主要是两者的计算资源分配方法的不同，带宽分配和存储

资源的分配是完全一样的。

另外还可以看到，细粒度的资源分配是一个闭环系统。从数据流应用的角度看，其输入是模糊控制器给出的 PF 和带宽分配算法给出的带宽分配参数，而其输出包括 UC 和各个应用的性能预测。UC 反馈给了模糊控制器，正是由于这个反馈作用，才使得模糊控制器可以适当的调整各个虚拟机的计算资源分配。而各个应用的性能观测，则是与带宽分配算法息息相关的，它影响到以整体的吞吐量为指标的遗传算法的结果，也就是下一个周期内带宽分配的参数。与此相对应，图 4.1 所示的粗粒度的资源分配方法，就是一个开环系统，因为资源分配的结果及其产生的影响，如应用的吞吐量等，没有形成反馈。

5.3　实验结论

实验用的服务器为 HP DL580G5，共有 4 个物理 CPU、16 个 Xeon E7310 核、8GB 内存，虚拟机就建立在这个服务器上。来自 LIGO 的应用运行在这些虚拟机上，数据从数据源传输到相关的虚拟机。

根据处理的复杂性，应用被分为两类：轻量级应用和重量级应用。在轻量级应用中，数据处理比较简单，而重量级应用则需要更复杂的处理，需要更多的计算资源。实验中分别采用了轻量级和重量级应用各 3 个，以检验算法性能。总的可用带宽 *I* 设置为 5Mbps，*CapScale* 设置为 3。

在 100 个调度周期内对本章提出的细粒度的混合资源调度和分配算法进行了评估。性能指标包括模糊控制器的输出（PF）、计算资源利用率（UC）、每个应用分配的计算资源（Cap）

5.3.1　系统辨识

为了揭示各个应用的 TPS 和分配的计算资源之间的关系，这里采用了系统辨识的方法，以下辨识的对象为 rmon，详见第 6 章的介绍。

建立了三个虚拟机，其内存分别为 512MB、256MB 和 128MB。分配给每个虚拟机的 Cap 从 5%以步长 5%均匀变化到 100%。总的数据处理完成时间如图 5.6 所示，从中可以看出内存对处理完成时间的影响很小。所以，目前为止，本文主要研究 CPU 份额的细粒度分配。

处理速度如图 5.7 中的实线所示。多项式拟合的方法用来得出 Cap 到处理

速度的函数，这里采用的是最小二乘法。二阶多项式拟合的结果为

$$p(c) = -0.9830 + 0.5135c - 0.0031c^2, c \in [5,100]$$

其中 c 是 Cap 的缩写，其拟合曲线如图 5.7 中的虚线所示。

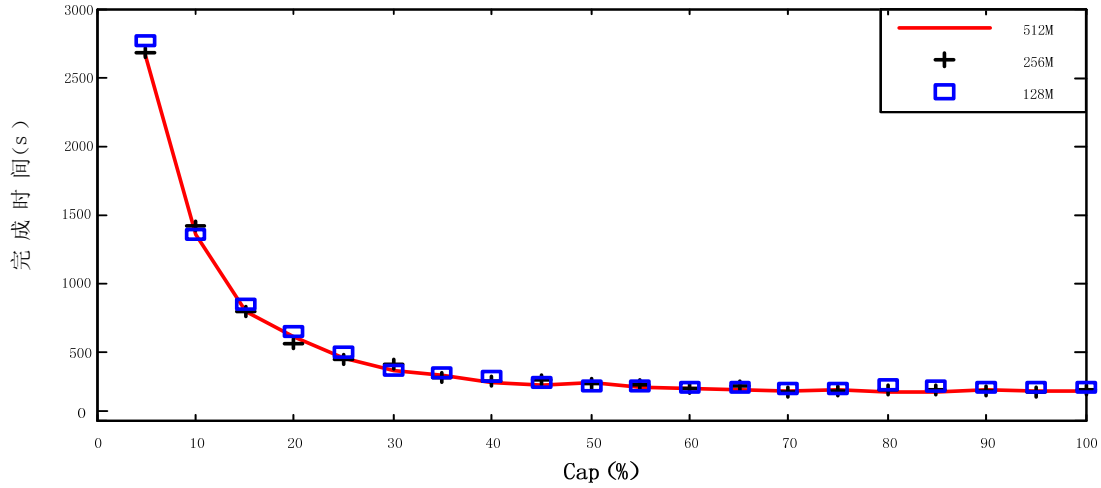


图 5.6 不同 Cap 和内存的处理完成时间

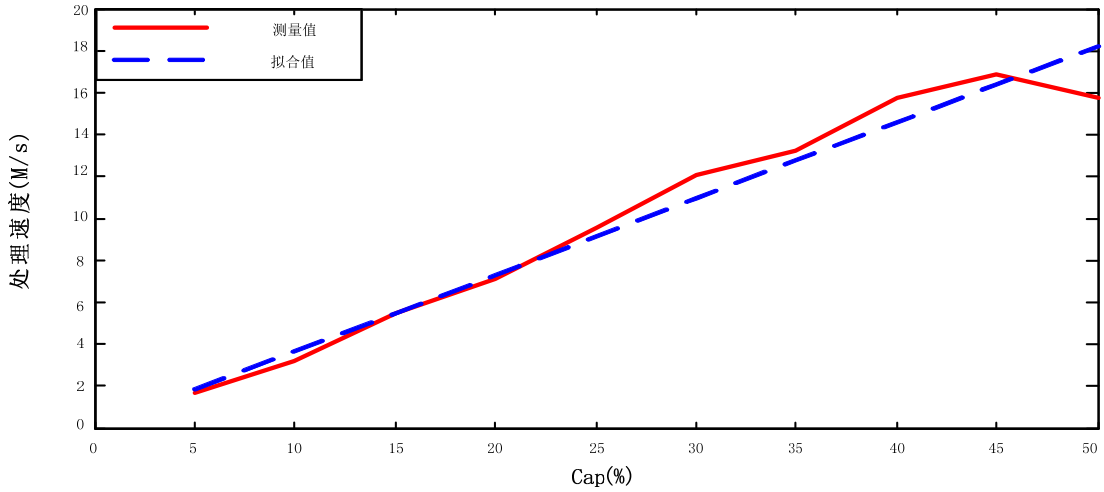


图 5.7 大范围内的二次多项式曲线拟合

从图 5.6 和图 5.7 可以看出，一旦分配的 Cap 超过 50%，再增加同样的 Cap，增加的处理速度就不会象在 5%到 50%段那样明显。所以，在 Cap 从 5%到 50%这个范围内变化时进行曲线拟合更有意义。此时的二次多项式为

$$p(c) = -0.1111 + 0.4330c - 0.0017c^2, c \in [5,50]$$

其曲线如图 5.8 中的虚线所示。实际上，一次多项式拟合就足够了：

$$p(c)=0.0375+0.3636c, c\in[5,50]$$

其曲线如图 5.9 所示。函数中的常数项非常小，可以忽略不计，这时就会得到一个线性比例公式

$$p(c)=0.3636c, c\in[5,50]$$

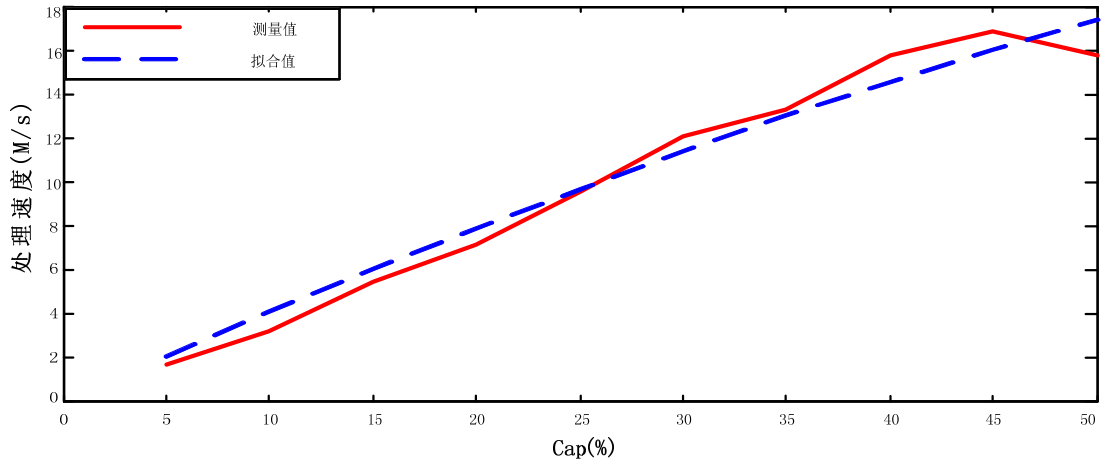


图 5.8 小范围内的二次多项式曲线拟合

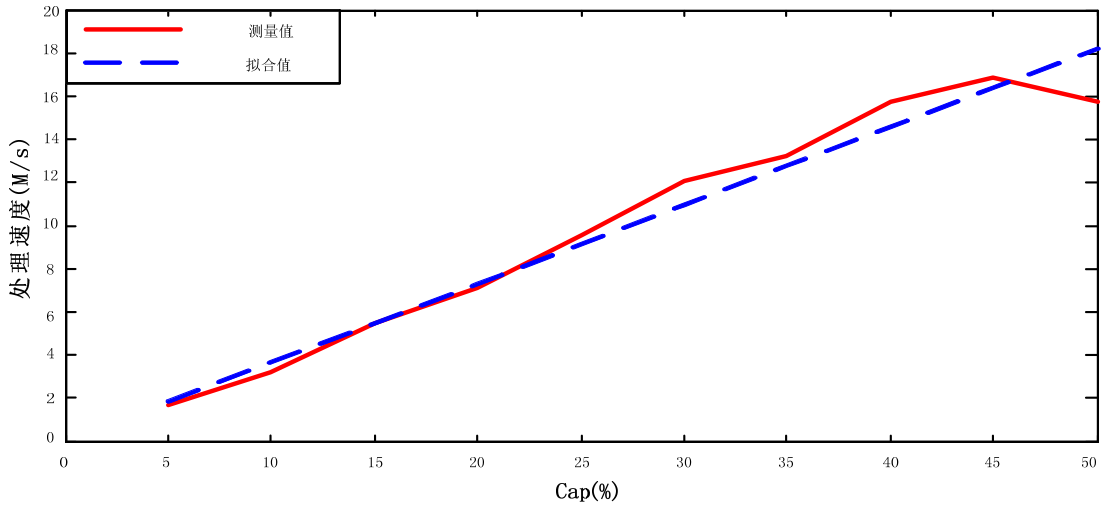
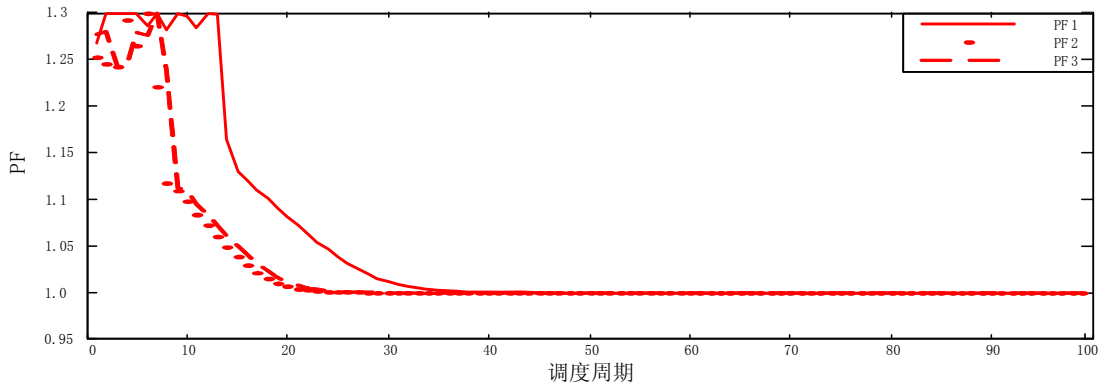


图 5.9 小范围内的一次多项式曲线拟合

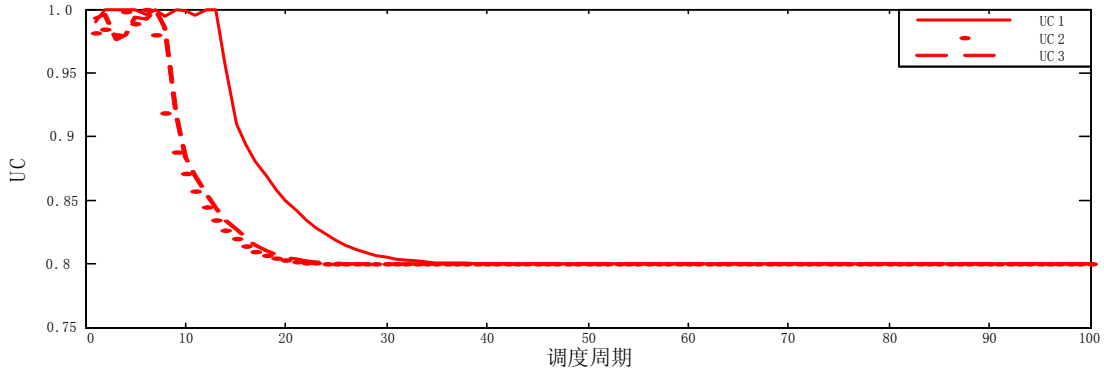
5.3.2　资源分配和利用

根据细粒度资源分配方案，轻量级和重量级的应用都获得了合适数量的资源，如图 5.10 和图 5.11 所示。可见，每个应用需要的计算资源远远少于一个完

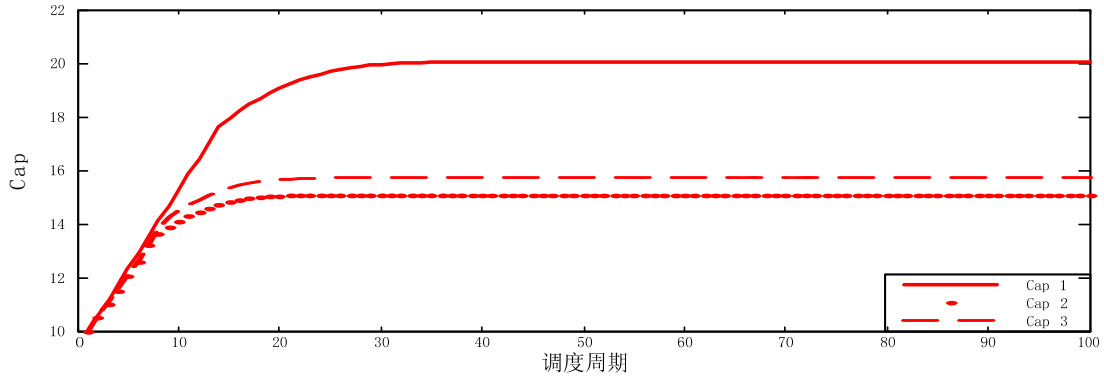
整的物理 CPU，因为每个应用获得的 CPU 份额都在 20%甚至 10%以下每个应用的初始 Cap 都设置为 10%，经过一段时间的调整后，所有的资源分配方案都收敛到一个稳定状态，且 PF 和 UC 没有稳态误差。



（a） 比例系数

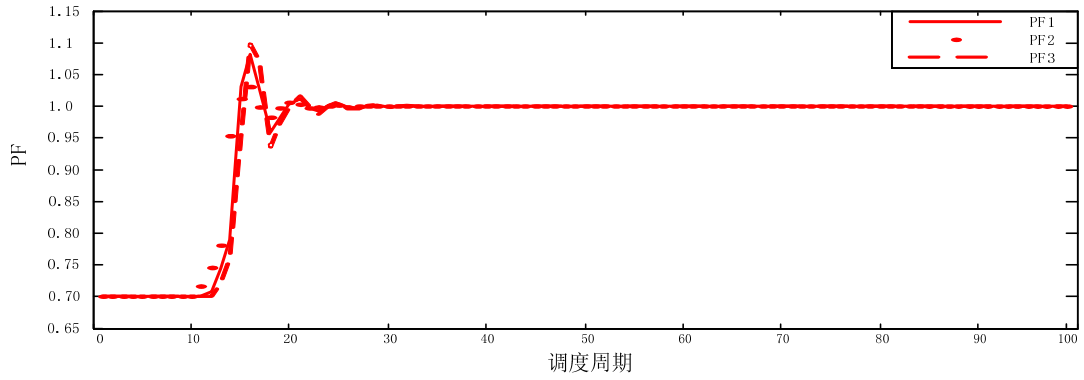


（b） 利用率

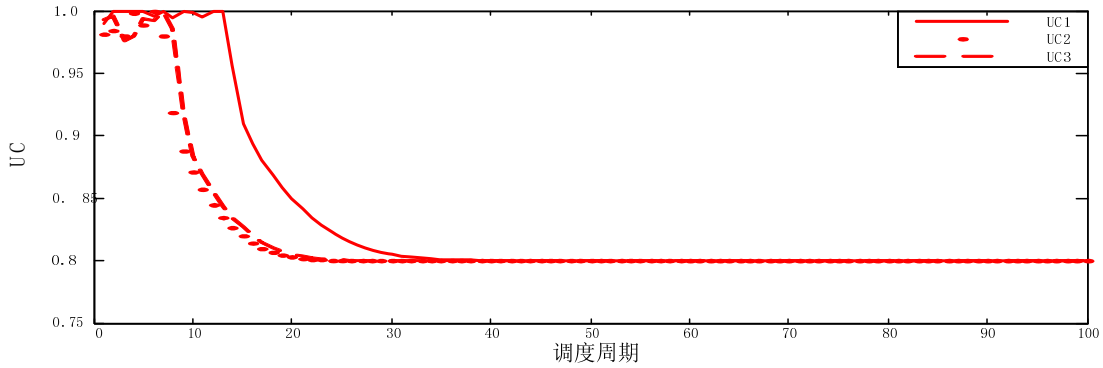


（c） Cap

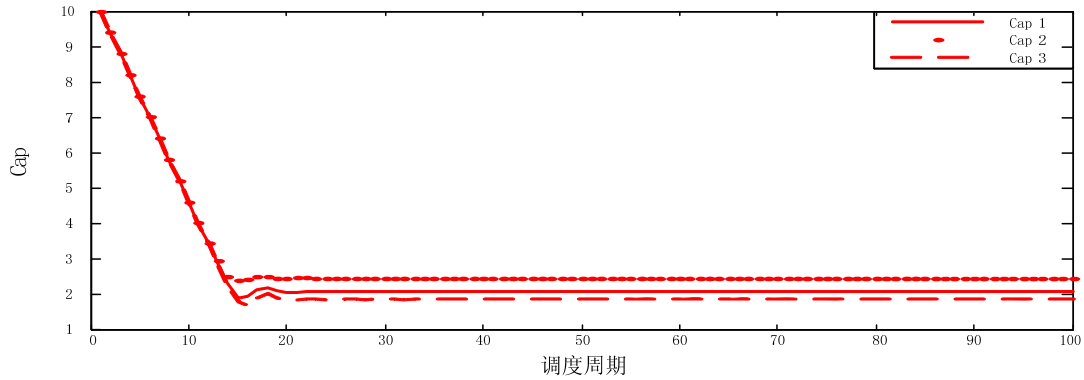
图5.10 重量级任务的性能



（a）比例系数



（b） 利用率



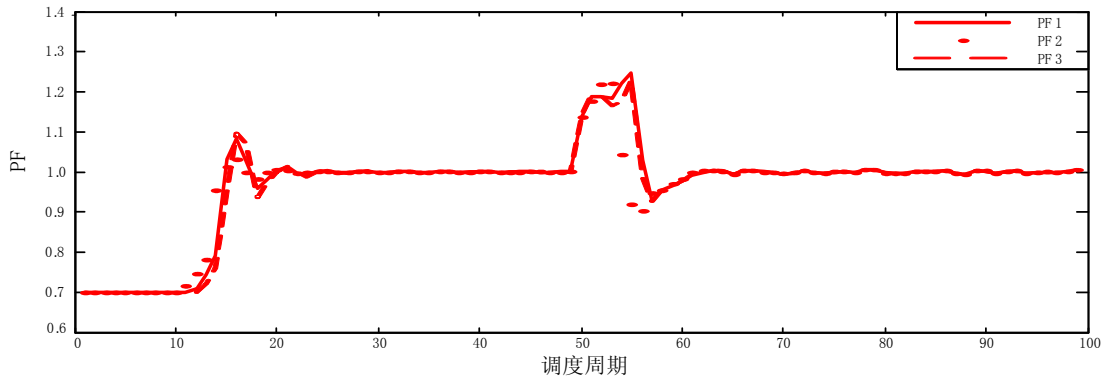
（c） Cap

图5.11 轻量级任务的性能

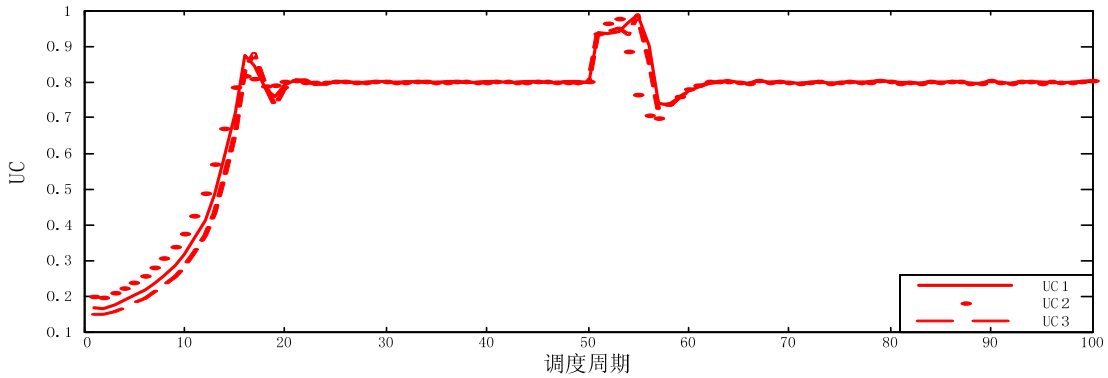
5.3.3 系统的反应能力

当可用资源发生变化时，本系统可以快速的做出反应。例如，在第 50 个调度周期，可用带宽突然增加到 10Mbps，PF、UC 和 Cap 都会发生变化。经过一

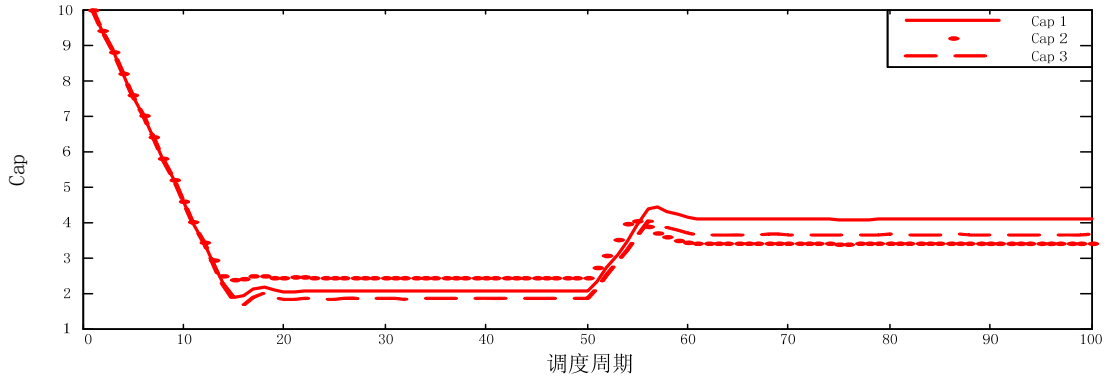
段时间的调整，系统重新进入稳定状态，如图 5.12 所示。此时，各个虚拟机获得了更多的计算资源以应对更多的数据供应。



（a） 比例系数



（b） 利用率

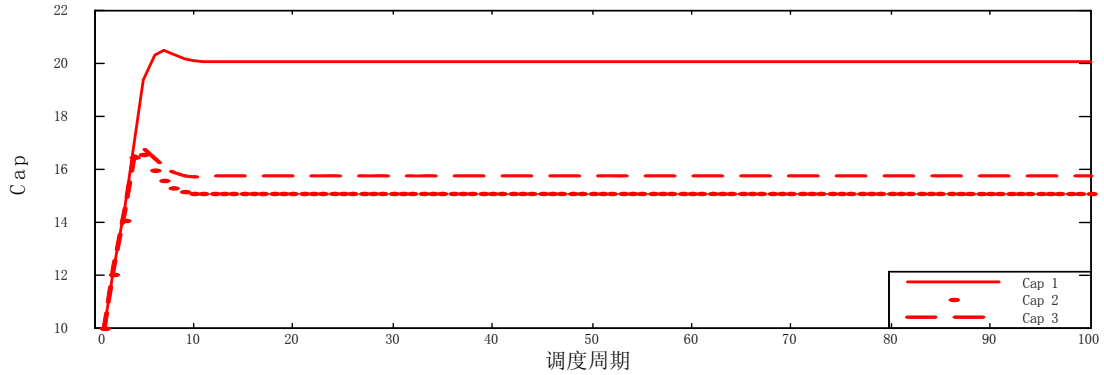


（c） Cap

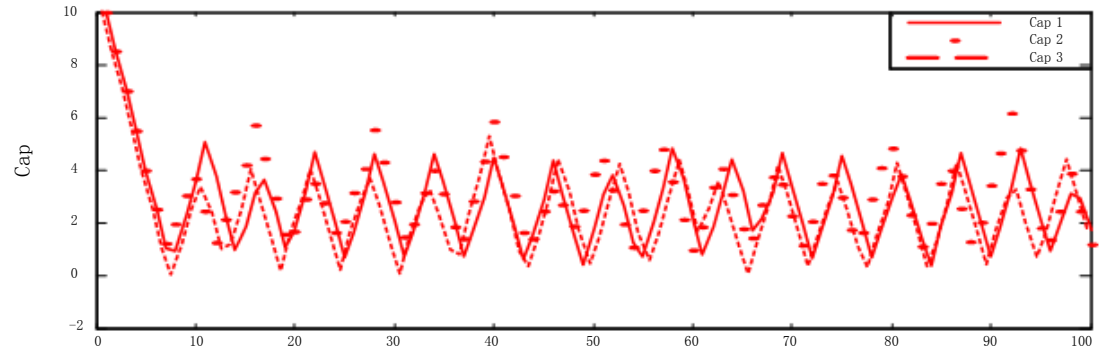
图5.12 对带宽的反应能力

5.3.4　参数收敛性和稳定性

实验中发现，参数 CapScale 对系统的收敛速度和稳定性有影响。对有些应用来说，CapScale 越大，收敛速度越快。如图 5.13（a）所示，当 CapScale 设置为 8 时，一些应用（主要是重量级应用）达到稳态的时间更短。但是， CapScale 大也可能带来性能上的振荡甚至是不稳定。例如，如图 5.13（b）所示，CapScale 同样设置为 8，轻量级应用的一些性能将不会收敛到一个稳定状态，所以这个控制系统不能正常工作。



（a）重量级应用



（b） 轻量级应用

图5.13　大CapScale情况下任务的性能（Cap）

从实验结果中，可以得出以下结论：

①在稳定状态，数据供应的速度和处理速度在高水平和细粒度上达到了均衡，从而保证了应用的运行并获得了较高的吞吐量，而仅仅占用了合理数量的资源，尤其是计算资源的利用是相当经济的；

②在稳定状态，计算资源的利用率保持在 80%，即收敛于图 5.2 中语言值

High 的隶属函数的中心，且没有稳态误差，因此计算资源得到了有效利用；

③在稳定状态，重量级应用将比轻量级应用占用更多的计算资源；

④可用带宽越高，需要的计算资源越多；同样，一个应用得到的计算资源越多，其使用的带宽就越多，这反映了带宽和计算资源之间的交互和耦合，从表 5.3 中也可以看出；

⑤吞吐量是由总的可用带宽和计算资源共同决定的，带宽或计算资源的单方面的充足不一定可以产生高的吞吐量，只会导致资源的低效利用，所以细粒度的综合资源调度是必需的；

⑥通过调度和规划，多个应用可以同时运行在一个物理处理器上，一个应用独占一个虚拟机，这样可以使资源得到高效利用；相反的，将一个物理处理器完全分配给一个数据流应用，将会导致严重的浪费和低效利用。

5.3.5　与其它分配方法的比较

为了验证本章中的算法，这里比较了另外几种分配方法，如表 5.2 所示：“迭代”表示迭代式分配带宽，“平均”表示均匀分配带宽；“动态”表示细粒度的按需计算资源分配，而“固定”表示分配给每个应用的计算资源是固定的，不会按需调整。很明显，本章的分配方法属于算法 1。

表 5.2 算法设置

	带宽	计算资源
算法 1	迭代	动态
算法 2	迭代	固定
算法 3	平均	动态
算法 4	平均	固定

表 5.3 给出了一些结论，其中从上而下的性能指标表示吞吐量、以百分比表示的 CPU 的使用情况（Usage）和带宽使用情况。字母 H 和 L 表示应用的类型，H 表示重量级应用，L 表示轻量级应用。这里所说的 CPU 的 Usage 是指运行在一个物理 CPU 上的各个虚拟机获得的 Cap 的总和，与前面定义的计算资源利用率 UC 的含义不同。带宽使用情况是各个应用获得的带宽的和占总的可用带宽的百分比。由于各个应用可以获得的带宽总是在一定范围内变化的，其上限就是从数据源到数据处理节点的瓶颈连接处的带宽，因此，有时候各个应用获得的带宽之和小于总的可用带宽。

可见，本章的算法在各个方面都有优势，这进一步证明了细粒度综合资源分配的必要性。例如，该方法利用更少的资源获得了更高的吞吐量。相反的，在一些情况下，例如在情况 4 中，不论应用的实际需求如何，带宽都是平均分配的，且计算资源的分配方案一成不变，这时应用各方面的性能都有所下降，尽管它们占用了更多的资源。情况 2 和情况 3 稍微好一些，但是它们的性能也不理想。这表明带宽或计算资源的单方面调整不足以同时实现吞吐量和资源利用率方面的目标，从另一个方面说明对数据流应用来说，带宽和计算资源应该综合的协调分配。

表 5.3 各种算法的性能

指标	算法		总的可用带宽		
			5	10	15
最终吞吐量	H	1	49951	97080	117140
		2	46542	77107	77269
		3	46745	87495	104660
		4	48566	80004	82296
	L	1	49999	99994	134980
		2	46569	94019	123458
		3	47576	95018	124990
		4	48967	94948	104458
CPU Usage (%)	H	1	40.17	77.76	98.81
		2	50	50	50
		3	39.16	66.09	91.88
		4	50	50	50
	L	1	9.79	20.74	31.93
		2	50	50	50
		3	9.76	19.90	28.93
		4	50	50	50
带宽 Usage (%)	H	1	93.49	91.50	81.77
		2	93.08	77.11	51.51
		3	92.90	87.08	80.10
		4	90.13	80.00	54.86
	L	1	92.14	92.02	89.99
		2	93.14	93.14	90
		3	91.41	90.50	83.33
		4	93.24	91.99	80.33

5.4　小结

本章介绍了网格数据流资源管理和调度的细粒度方法。利用虚拟化技术，可以为每个应用提供一个独享的运行环境，且其资源配置可以通过模糊控制动态改变。细粒度的资源管理和调度方法，可以有效的改善对资源的使用方式，即资源可以按需的分配给各个应用，既保证了应用的运行效率，又避免了资源的无谓浪费。

第 6 章　面向天文数据流应用的使能环境

一些天文应用具有数据流的典型特征：数据实时产生；总的数据量十分庞大，且不断增加；原始数据需要深入分析和处理才能发现其中隐藏的宝贵信息；天文台本身没有足够的计算资源，数据需要传输到其它组织的计算资源上进行处理。作为 LSC 的成员，我们初步设计实现了一个面向天文数据流应用的使能环境，为类似于（但不仅限于）LIGO 的数据流应用提供支持。

6.1　应用对象简介

如前所述，一些天文应用充分体现了数据流的特点，现以两个具体的应用进行说明。同时这些应用还表明了数据流应用对资源管理和调度系统的要求，是本应用使能环境设计和开发的基础。

6.1.1　rmon

这是 LIGO 数据分析中的一个典型的数据流应用。来自两个天文台的数据流分别用两个文件列表表示，它们是处理程序 rmon 的输入。这个程序用来计算一个统计参数，记作 *r*，以此度量两个数据集的相似性，如图 6.1 所示。如果两个天文台同时出现了具有相似波形的信号，那么就很有可能是探测到了一个引力波的脉冲（Burst）信号。数据是以时间序列的方式组织的，所以可以用数据流的方式处理。这只是一个简化了的 LIGO 数据处理例子，实际上在一个完整的处理流程中，还有许多前期和后期的处理。

r 定义如下

$$r=\frac{\sum_{j=0}^{n-1}\left(x_{1j}-\bar{x}_1\right)\left(x_{2j}-\bar{x}_2\right)}{\sqrt{\sum_{j=0}^{n-1}\left(x_{1j}-\bar{x}_1\right)^2}\sqrt{\sum_{j=0}^{n-1}\left(x_{2j}-\bar{x}_2\right)^2}}$$

其中

$$\bar{x}_i=\frac{1}{n}\sum_{j=0}^{n-1}x_{ij},\ i=1,2$$

rmon 的数据流由许多小文件组成，每个小文件包含 16 秒的数据。这里用到的文件是 LIGO 的第三级数据，仅包含引力波频道的数据。

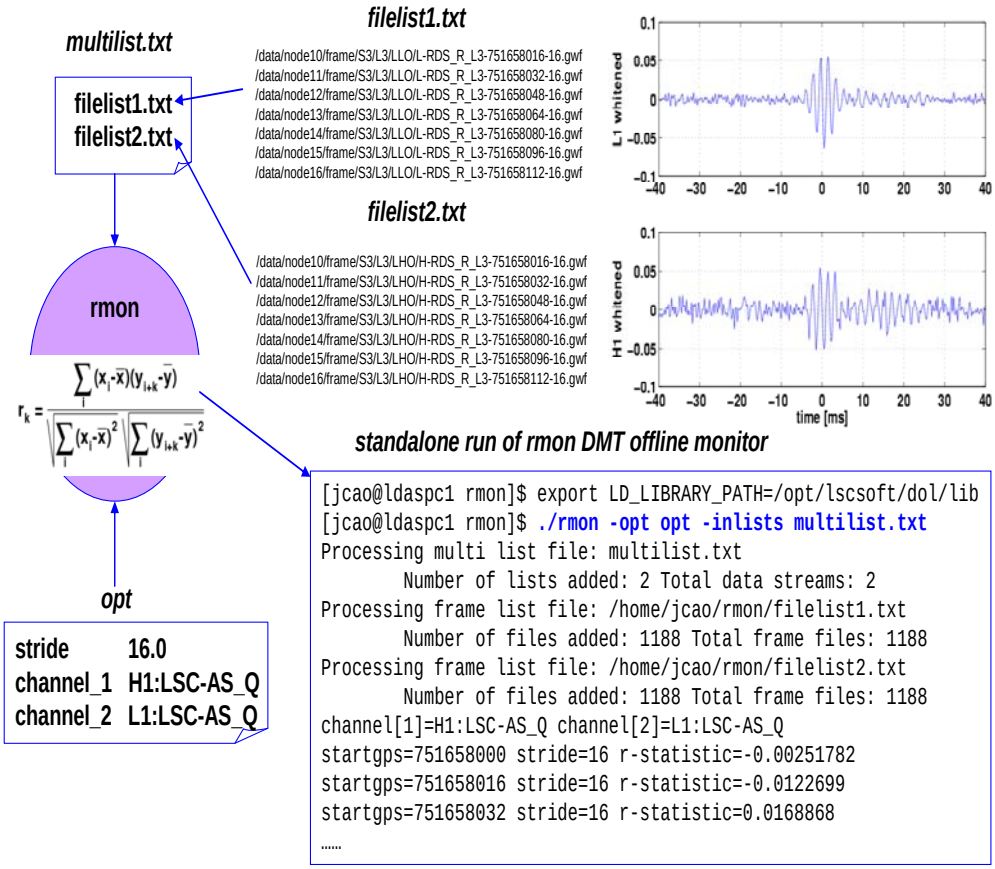


图6.1 rmon示意图

rmon 程序的运行需要 LSC 开发的一些库函数的支持，需要事前将这些库函数通过网络（一般是 yum 的形式）安装到运行这个程序的主机[109]。rmon 程序有一些参数需要设置，这些参数写在一个文件中，内容如下：

stride　　16.0

channel_1　　H1:LSC-AS_Q

channel_2　　L1:LSC-AS_Q

其中第一个参数说明了每个数据文件的时间跨度，即每个数据文件包含 16 秒的数据；后面的两个参数则说明了数据的来源，其中的 H 和 L 分别表示这些数据来自 Handford 和 Liveston 的天文台。

rmon 程序要处理的对象写在另一个配置文件中，名为 multilist.txt，其内容

如下：

filelist1.txt

filelist2.txt

其中 filelist1.txt 和 filelist2.txt 中包含了要处理的文件名（如 H-RDS_L3-75165800-16.gwf 和 L-RDS_L3-75165800-16.gwf）。设置好这些配置文件，并经编译后，rmon 程序就会顺利运行。

6.1.2 Montage

天文学中各个波段都有丰富的图像资源。由于坐标系、星图投影、空间取样和图像大小等方面的不同，图像搜集常常被限制在一定的频率范围内进行。此外，许多天文研究的对象，如星系团和恒星形成区域已经大大超过独立的图像。天文学及其他学科需要图像拼接软件，以便从多个图像数据集中拼合出科学级别的图像，就好像他们是来自具有一个共同的坐标系和星图投影的单一图像。Montage[110]就是这样的一个工具，它在天文学、气象学、地震科学等多方面具有较广泛的应用。具体说来， Montage 是一个工具包，可以将复杂图像传输系统（Flexible Image Transport System）[111][112]的图像组装为用户定义的格式。对终端用户而言，其关键特性包括：

- 准确性：保留了输入图像的空间位置及校准信息。
- 便携性：可以运行在普通的 Linux/Unix 平台上。
- 可扩展性：可以运行在桌面、集群和计算网格上。
- 可用性：代码是开源的，用户文档可以自由下载。
- 通用性：支持所有的世界坐标系投影和普通的坐标系统。
- 高效性：在 128 个节点的 Linux 集群上，在 32 分钟内可以处理 4,000 万个像素点。
- 灵活性：一些独立的引擎分别负责天空中图像的几何分析、图像的重新投影、将背景发散调整到普通水平、图像叠加等。

Montage 已被设计成一个可扩展的、便携的工具包，天文学家可使用桌面电脑来完成他们的科学分析，并纳入项目和任务的管理，或运行在计算网格中以支持大规模计算任务、任务规划和质量保证。因此，它将大大扩展天文研究的范围。它的功能包括多波长深空探测、预测来源点和波长、基于波长的位置优化、扩展波长结构、图像差分检测微弱信号和发现新种类的物体等。图 6.2

描述了一个简单的拼合三张图片的 Montage 并行工作流。

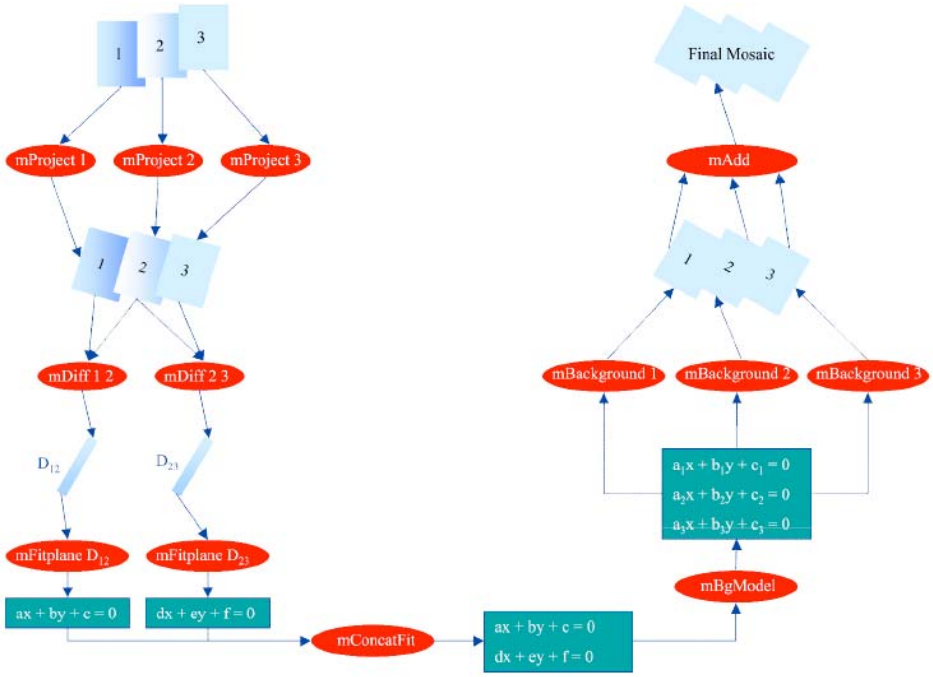


图6.2 Montage的一个实例工作流程

6.2　系统运行环境

系统中所有的服务器和计算机，均安装了 Linux 操作系统，其发行版本为 CentOS 5.3。系统的开发语言为 C++；编译工具包括 gcc、g++、make 和 autoconf/automake 等；开发库包括 zlib、python-dev 等。

为了保证资源管理和调度的顺利进行，本系统采用了一些网格中间件，如 Globus 和 Condor。此外，还采用了虚拟化工具 Xen。

⑴ Globus

系统中所有的服务器和计算机上，都安装了 Globus Toolkit 4.2.1 作为网格资源管理的中间件。在系统实现中，Globus 要建立网格基础环境，对加入网格的所有资源和服务进行统一的管理，使用户提交的应用可以透明的、安全的访问所需要的资源。其中，Globus 提供的最重要的两个功能是安全服务和数据传输，分别由 simpleCA 机制和 GridFTP 实现。simpleCA 提供了主机、用户之间的相互验证，通过 grid-mapfile 实现了外部用户到本地用户的映射以及权限控制。GridFTP 用于在数据源和数据处理程序之间传输数据，可以实现资源的动态分

配，即根据需要动态的调整 GridFTP 的相关参数，包括 TCP 缓冲区大小、并行度等，以实现性能的最优化。

⑵ Condor

系统中所有的服务器和计算机上，都安装了 Condor 6.8.6。在这里，Condor 的任务是与 Globus 配合，为应用寻找并分配计算资源，主要用于粗粒度的资源分配中。当前，Condor 的最新版本是 7.2.4，它引入了一些不错的新特性。但是由于它采用了基于 kerberos 的安全机制，导致用户的使用不如以前方便，所以本实现中没有采用最新版本。

安装了 Condor 并设定了同一个管理者（Condor 中的术语为 CONDORHOST）的机器，组成一个 Condor 池（Pool）。Condor 池中的计算资源接受管理者的统一调度和分配，对外呈现为一个整体提供计算服务。

⑶ Xen

作为一个开放源代码的半虚拟化管理程序，Xen2.6 包含在 CentOS 的发行版本中。Xen 支持在一套物理硬件上安全的执行多个虚拟机，从而提供资源隔离和性能保证。在虚拟平台上可以安装其它操作系统，即客户操作系统。

6.3　系统架构

本节介绍系统的物理结构、安全结构、资源结构和功能结构等。应当指出的是，这些结构是对本系统的不同角度、不同层次的描述，它们在逻辑上是统一的。

6.3.1　物理架构

如图 6.3 所示，数据源（图中 S_i 所示）与处理资源（图中 P_j 所示）是通过网络（图中显示为 Internet）联系在一起的，且带宽（图中 I 所示）是共享的。在进行数据处理时，数据通过网络传输到处理资源本地的存储器中。为了方便处理资源对传输到本地的数据的访问，系统中部署了网络文件系统（Network File System，NFS）。

NFS 是由 SUN 公司于 1984 年推出的，用来在网络中的多台计算机之间实现文件或其它资源的共享。它采用了与主机和操作系统无关的的通信协议，以实现不同的系统之间的交互使用。NFS 可以将远程计算机上的文件系统挂载到

本地系统中，本机可以方便的访问这些文件，就象这些文件是存储在本机一样。

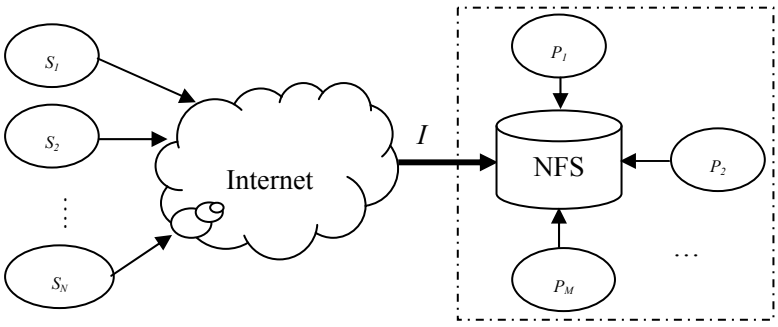


图6.3 系统物理结构

此外，NFS 还有利于保持应用目录的一致性，便于应用的移植。如 6.1.1 节所述，数据流应用的一些配置文件中，往往采用的是绝对路径。当应用被提交到系统中，且被调度到其它主机（包括物理机和虚拟机）上运行时，就要求宿主机上也必须有相应的目录，否则程序就不能顺利运行。在一个开放式的运行环境中，这显然是不合理的。但是有了 NFS，这个问题就迎刃而解了。因为在 NFS 中，目录对所有的主机都是一样的。例如，将主机 166.111.137.17 中的一个目录/storage 作为共享目录，只要在其文件/etc/exports 中加入如下语句

/storage 166.111.137.19(rw,sync,no_root_squash)

/storage 166.111.137.87(rw,sync,no_root_squash)

并在 166.111.137.19 和 166.111.137.87 中建立/storage 目录，然后在其/etc/fstab 中加入如下语句

166.111.137.17:/storage /storage nfs rsize=8192,wsize=8192,timeo=14,intr 0 0

并启动 portmap 和 nfs 等服务，即可完成 NFS 的挂载。此时，无论在哪台主机上操作/storage，都会在另外两台主机上反映出来。这样就将极大的方便 6.3.4 中工作目录和数据目录的建立，因为此时不必考虑宿主机上有没有相关目录，只要在本机上操作 NFS 目录即可。

6.3.2　安全架构

安全是所有网格系统的最基本功能之一，也是网格系统要解决的核心问题之一。本系统采用了Globus Toolkit中包含的GSI组件提供的健壮的安全性机制，具体实现为 simpleCA。

本系统中，在一台计算机上建立了证书认证中心，负责为其它主机和用户签发证书。在每个主机上，都将建立一个名为 grid-mapfile 的文件，其中包含了证书签发者、外部用户以及映射到本地的用户等的信息。如一个 grid-mapfile 中可能含有如下语句

/O=Grid/OU=GlobusTest/OU=simpleCA-hanqiu.riit.tsinghua.edu.cn/OU=tsinghua.edu.cn/CN=zhangwen globus

其含义是：本 CA 认证中心为 hanqiu.riit.tsinghua.edu.cn（这是网格中的 CA 的域名）；外部用户 zhangwen 将映射为本地账户 globus。

一般来说，为了简化 grid-mapfile 里的内容，会在本机中建立一个公共账户，并赋予其相应的权限。外部用户来访问时，一律映射为此用户。这简化了操作，但是也带来了一定的安全隐患，需要系统管理员就此做出选择。

6.3.3　资源架构

在本系统中，存储资源通过 NFS 由所有的计算机和服务器共享。专门的存储器用来为此提供可以共享的存储空间，在进行数据传输时，GridFTP 不必关心将数据传输到哪台计算机或服务器，而只要知道其数据宿主在共享的存储器中的目录即可，而这个目录是在用户提交任务时由调度器自动给出的，详见 6.3.4。

带宽资源，即图 6.3 中的 I，是由多个应用共享的，其分配采用了第 4 章中介绍的迭代式算法。在实际的工程实现中，迭代式算法给出的结果，是通过动态调整 GridFTP 的相关参数（包括传输并行度、TCP 缓冲区大小等）逐步逼近的。

在粗粒度和细粒度的资源管理中，计算资源的管理是不同的。在粗粒度资源管理和调度中，计算资源的管理和调度是通过 Condor 实现的。在 Condor 池中，一个计算节点可以同时拥有三种角色：提交者（Submitter）、执行者（Executer）和管理者（Manager），三者分别运行不同的后台程序。提交者负责将用户的任务提交到 Condor 池中，执行者将实际执行这些任务，而任务到执行者的映射是由管理者完成的。Condor 池中有一个中心节点，它一般只充当管理者的角色，而不承担具体的计算任务。其它的计算资源，则可以同时扮演提交者和执行者的角色，即这些资源既可以提交任务到其它主机，也可以执行其他主机提交过来的任务，如图 6.4 所示。

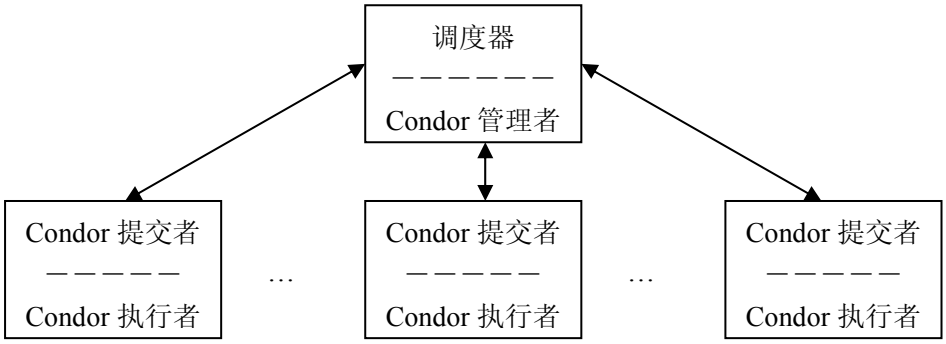


图 6.4 粗粒度计算资源结构

在细粒度的资源管理中，计算资源表现为一个个的虚拟机。这些虚拟机是在应用到达时由调度器实时生成或唤醒的，如图 6.5 所示。虚拟机的初始配置相同，在应用运行过程中，由调度器给出相关参数，动态的调整虚拟机的 CPU 份额，即第 5 章中所说的 Cap。这些虚拟机可以运行在同一台主机上，也可以运行在不同的物理机上。

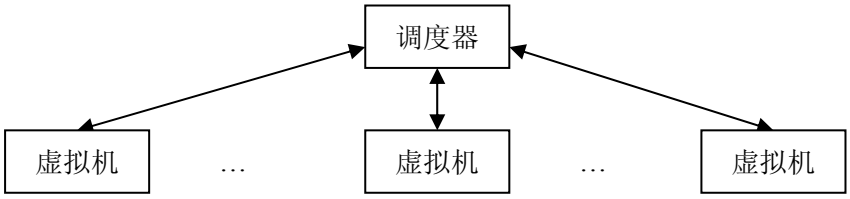


图 6.5 细粒度计算资源结构

值得注意的是，细粒度资源管理中的调度器是层次式的：在每个物理机上，都有一个调度器，称为本地调度器，它负责本地虚拟机的资源调整；此外，还有一个中央调度器，它负责与客户端交互。本地调度器需要向中央调度器报告本地资源的信息，如当前可用虚拟机的数量和配置等；中央调度器根据这些信息，决定将客户端传来的任务提交到哪个物理机上的本地调度器。中央调度器与本地调度器的关系如图 6.6 所示。将调度器分为中央调度器和本地调度器，一是可以减轻中央调度器的负载，减少单点失效；二是本地调度器更容易实现对本地虚拟机的控制和调整。实际上，任何一个调度器都有中央调度器和本地调度器的功能，在意外或灾难情况下，任何一个本地调度器都可以替代失效的中央调度器，从而提高了系统的可靠性。此外，几个中央调度器还可以联合起来，以应对大量的客户访问。

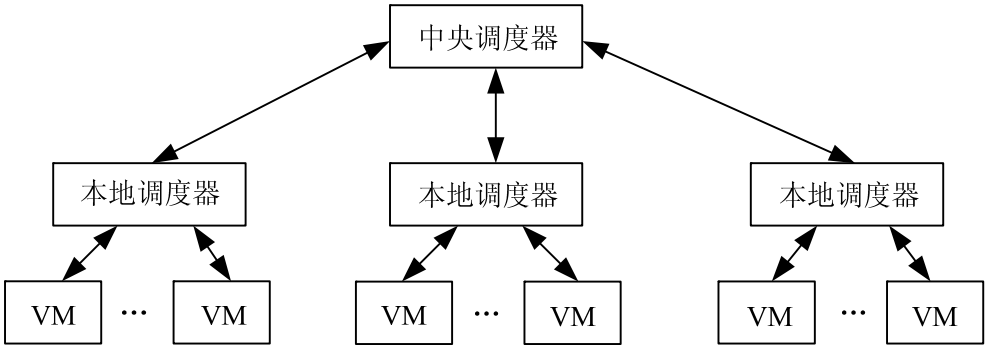


图6.6　细粒度计算资源的层次式调度

这些虚拟机利用桥接方式（Bridge）组成虚拟机网络。同一个物理机上的虚拟机共享一个公共 IP 地址，如 166.111.137.18，而每个虚拟机只有一个内部 IP 地址，如 192.168.0.101。在本实现中，Globus 等支撑软件需要域名服务器（Domain Name Server，DNS）的支持。这里采用的 DNS 是清华大学的服务器，IP 地址为 166.111.8.28、166.111.8.29 和 166.111.131.4.5。这些服务器不能对虚拟机的内部 IP 地址进行解析，因此 GridFTP 等工具不能运行在虚拟机上。需要将数据通过 GridFTP 传输到具有外部 IP 地址、可由公共 DNS 进行域名解析的存储器上。由于采用了 NFS，虚拟机可以直接访问这些数据，如同这些数据就是存储在本地。这也是本系统采用 NFS 的重要原因。

6.3.4　功能架构

从使用者的角度看，系统的功能结构包含提交者、调度器、分配器等实体。其中，提交者与调度器需要进行交互，以确定特定的参数，如数据源地址和路径、应用的序列号等；调度器和分配器的相互作用，是隐藏在系统内部的，对用户是不可见的。因此，本部分重点介绍任务提交者和调度器之间的交互作用，而调度器和分配器之间的相互作用已经在第 4 章和第 5 章进行了介绍，在此不再详述。

为了更好的描述任务提交者和调度器之间的交互，这里需要说明数据流应用的一般属性。作为一类应用，数据流应用具有一些共同的特性。首先，数据流应用是分布式的。数据流应用是将数据从远端的数据源通过网络传输到计算资源所在地进行处理，需要确定数据源地址和数据目的地址。由于采用了 NFS，数据的目的地址也可以转化为相应的数据目录。其次，数据流应用是多线程的。

为了提高数据处理的效率和资源的利用率，在数据流应用中，应当至少同时运行两个线程，分别用于数据传输和处理，这两个线程需要协同配合。对于同时涉及多个数据源的应用，数据流应用就应该是多线程的。再次，数据流应用往往需要一些应用程序库等的支持，这些支持库应当放置在合适的位置。例如，LIGO 的一些应用，往往需要 LSC 的相关程序库等的支持。最后，数据流应用往往需要一些配置文件，说明数据源的位置、程序的编译方式等。一般说来，这些配置文件需要与放置数据处理程序放置在同一个目录中，在后面的叙述中，本文将这个目录称为工作目录。

用户提交的数据流应用，一般只是数据处理程序和一些相关的配置文件，很少考虑数据传输的问题。为了协调数据传输和处理，必须对用户提交的应用进行包装，使其成为一个同时包含数据传输和处理的多线程的程序。实际上，真正提交给计算系统的，就是这个包装过的程序，而不是用户提交的原始应用。包装过的程序，可以是一般的可执行文件，也可以是脚本文件等。对用户应用的包装，是在客户端自动完成的：客户端提供一个封装的模板，并交互式的提示用户提供相应的参数；这些参数，连同调度器给出的调度参数，一并写入一个脚本文件；在粗粒度的资源调度中，这个脚本文件将作为最终的提交文件（Submission File）提交给 Condor，由 Condor 为封装后的应用分配计算资源；在细粒度的资源调度中，这个脚本文件将直接提交到虚拟机上。封装后的程序在运行时，将通过命令行解析的方法获得相关参数，并确保数据传输和处理的协调运行。这里，调度器只给出调度参数，大部分的操作发生在客户端，从而可以减轻服务器端的负载，也使用户的操作更加方便。

在此，借鉴面向对象的一些方法，对数据流应用进行抽象和描述。数据流应用的一般属性包括：

①数据源地址。只需要提供数据源的 IP 地址或其域名（Hostname），不必提供其具体的路径，因为这些路径一般会在相关的配置文件中说明。

②数据流条数。即需要同时传递几条数据流，这随不同的应用而不同。例如，LIGO 中的 rmon 程序同时处理两个天文台的数据，因此需要两条数据流，有些应用则可能只需要一条数据流（如 Montage）。

③相关的配置文件。配置文件应提供明数据处理的程序、编译方式等信息。实际上，应当将这些配置文件传递到预定义的工作目录中。

④工作目录。包装后的程序和配置文件等，都将传输到这个目录中。这个

目录可以由用户指定，也可以由调度器自动生成。需要注意的是，这个目录必须是绝对路径，而不能是相对路径，因为相对路径会引起寻址方面的困难和错误。

⑤数据目录。这个目录是数据传输的目的地，是由调度器自动产生的。调度器保持一个全局变量作为计数器，每接受一个应用，就将此计数器的值加 1，并将这个值添加到一个字符串后面，以此作为应用的数据目录。

引用面向对象的术语，数据流应用的方法有两个，一个是数据处理，另一个是数据传输。数据处理的方法就是用户提供的数据处理程序，一般是一个可执行文件。数据传输则是由客户端生成的，并受调度策略控制。这两个方法分别构成包装后的应用的一个线程，并在调度器的控制下协调工作。

总之，客户端负责完成对应用的封装、建立运行环境等工作，使用户可以用近乎透明的方式提交自己的应用；调度器则要根据前面两章提出的算法，动态的产生调度参数。粗粒度和细粒度资源管理和调度系统中客户端与调度器的交互分别如图 6.7 和图 6.8 所示。

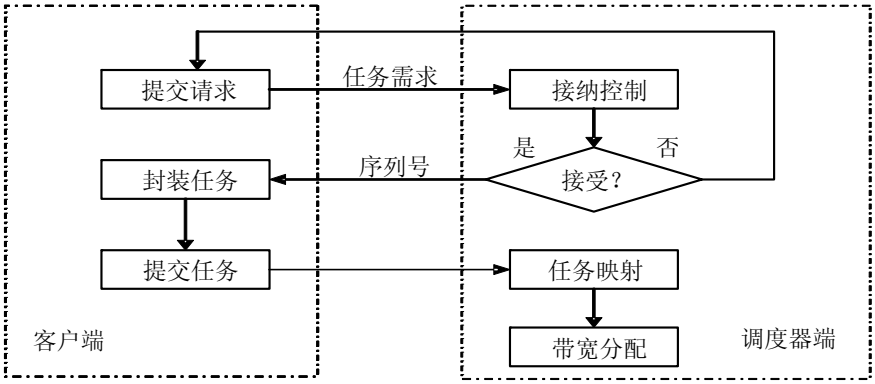


图6.7 粗粒度客户端与调度器的交互

由此可见，粗粒度和细粒度的资源管理和调度的主要区别在于两者的调度器端。由图 6.7 可见，粗粒度的资源管理和调度中，调度器只是将通过接纳控制并经客户端封装的应用映射到合适的资源，然后利用迭代式算法为这些应用分配带宽资源。这个过程中，调度器对计算资源没有控制。但是，由图 6.8 可见，在细粒度的资源管理和调度中，调度器不仅要将应用映射到虚拟机上，并为应用分配带宽，还要根据虚拟机的资源利用率，由本地调度器动态的调整其资源配置。对计算资源的分配来说，这就是一个闭环控制。

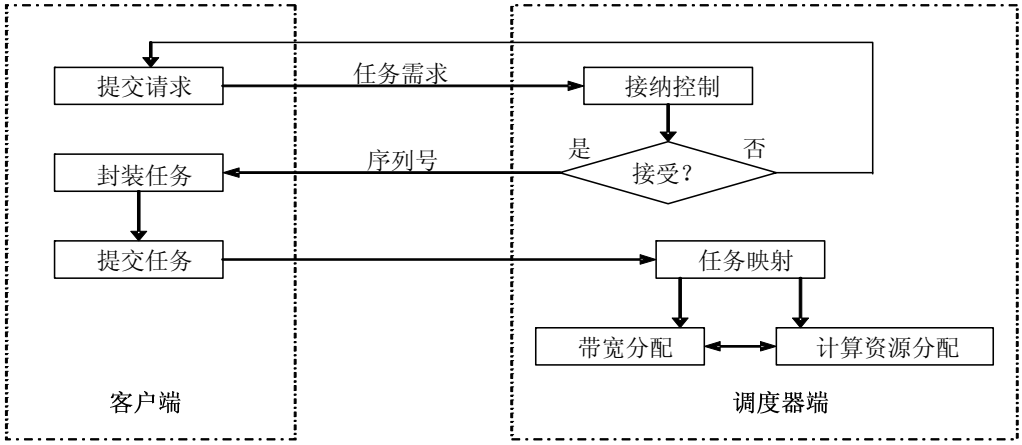


图6.8 细粒度客户端与调度器的交互

6.4　系统工作流程

本节以一个具体的例子来说明系统的工作流程。本节中用到的例子就是6.1.1 中介绍的 rmon，这个例子很有代表性，可以充分说明数据流应用的特点以及系统中客户端和调度器的交互，包括从用户提交任务、客户端封装任务直到调度器产生参数、分配器分配资源的整个过程。

6.4.1　需求分析

由 6.1.1 的分析可知，rmon 要处理的数据是由一些小的文件组成的。要处理的数据文件列表写在两个文本文件中，分别为 filelist1.txt 和 filelist2.txt，这些数据文件是存储在两个目录里面的。rmon 原本是处理已经存储在本地的数据，因此它只有一个数据处理的程序。在数据流应用中，rmon 需要在其它主机或虚拟机上处理数据，而这些数据是从其它数据源不断传输过来的。况且，在未来的宿主机器上，事先并没有 rmon 在本地运行时需要的目录结构，包括工作目录和数据目录。

为了使 rmon 可以按照数据流的模式在未知的宿主机器上处理不断的传输过来的数据，需要进行以下工作：

⑴对 rmon 程序进行包装，使其成为一个多线程的程序，且保证两个线程的同步。这里采用了多线程编程中的生产者/消费者模型：生产者线程负责供应数据，它受到 3.5.1 中介绍的库存策略的影响，其供应速度取决于带宽分配方案；消费者线程的作用是在有充足的数据可以处理时，调用 rmon 程序进行数据处

理。由于生产者和消费者线程的共同作用，存储中的数据量是变化的。因此需要实时的监测存储中（即数据目录中）数据文件的数量，以便为传输控制中的库存策略提供支持。

本文开发了一个程序封装模板，可以将原来的数据处理程序封装为多线程的程序。本模板可以适用于不同的应用，因为数据流应用中需要的参数，如数据源的位置、数据处理的程序等，都是通过命令行参数提供给封装模板的。模板对命令行参数进行解析，就可以得到实际的参数。这大大降低了应用封装的复杂度，也拓宽了本封装模板的适应范围。

⑵对应用进行编号。每一个提交到调度器且通过接纳控制的应用都会得到一个唯一的编号，称为应用的序列号。这个编号由调度器自动生成，其初值为1，每接纳一个应用，该编号自动增加。

⑶为应用建立工作目录和数据目录。本实现将 166.111.137.17 上的目录/storage 作为共享的 NFS，所有的工作目录和数据目录都建立在这个目录之下。由于采用了 NFS，无论在哪台机器上，都可以直接在本机的/storage 目录下建立工作目录和数据目录，并将直接反映到 166.111.137.17 和其它主机上。具体到每一个应用，其工作目录和数据目录分别命名为 workDir*和 dataDir*，这里的*就是其序列号。

⑷为封装后的应用建立脚本文件。在实际实现中，一般是将封装过的多线程应用连同其命令行参数，一起写到一个 shell 脚本文件中。这个脚本文件命名为 pack*.sh，其中的*也是该应用的序列号。

⑸将相关的程序和配置文件拷贝到工作目录。由于采用的是 NFS，无论封装后的程序被分配到哪个主机上，都可以直接使用本地拷贝命令（如 Linux 中的 cp）进行数据拷贝。

特别需要说明的是，由于本实现采用了基于模块的数据传输和处理，且数据模块的大小是一个调度参数，可以优化数据流应用的整体效率，因此需要对一些配置文件进行动态设置。在本节涉及的 rmon 应用中，rmon 要处理的数据文件是在 filelist1.txt 和 filelist2.txt 中设定的。在基于模块的数据传输和处理中，就要对这两个文件进行动态更新，使其仅仅存储正在处理的那部分数据文件名。例如，假如设置每个数据模块的大小为 40，也就是说一个数据模块中包含 40 个数据文件，当前正在处理第二个模块的数据，则 filelist1.txt 和 filelist2.txt 中就应只保存第 41-80 个文件名。具体的实现中，为方便起见，将数据文件统一

以其原来的顺序命名，如原来的第 54 个文件就命名为 file54，以此类推。

⑹生成 Condor 的提交文件。一般的，这个提交文件命名为 condor*.submit，其中的*也是应用的序列号。在提交文件中，execute 项的值设置为 pack*.sh，以便将脚本文件提交给 Condor。这一步仅对粗粒度资源管理和调度是必需的，因为细粒度的资源管理和调度中，pack*.sh 直接被提交到一个虚拟机上。

以上所有步骤中，除⑵是在调度器端完成的外，其余都是在客户端完成的。这样可以减轻调度器的负担，使其专注于参数的调度和规划。这样的分工平衡了客户端和调度器端的计算负载，不仅可以防止调度器过载导致单点失效，还可以提高调度的效率。

实际上，网格数据流应用的难点有两个，一个是确保数据传输和处理的同步，另一个是为数据流在未知主机上建立相关的工作目录和数据目录。因为在应用实际执行之前，用户无法预知其将被分配到哪个主机上，这样就无法建立工作目录和数据目录。本文开发了一个专门用于封装数据处理程序的模板，并采用了 NFS，解决了这两个方面的问题，为数据流应用的顺利运行奠定了基础。

6.4.2　示例实现

客户端与调度器是一种客户端/服务器的关系，两者通过 Socket 通信。现以粗粒度资源管理和调度说系统的工作流程，细粒度资源管理和调度的工作流程与此类似。设定调度器名为 Scheduler，运行在 166.111.137.17 上；客户端名为 Client，运行在 166.111.137.19 上；rmon 要处理的原始数据存储在 166.111.137.87 上。

在 166.111.137.19 上运行 Client，且指定 Scheduler 的地址，就将启动 Client 与 Scheduler 的交互。Scheduler 首先要求 Client 声明应用的资源需求，如对内存大小的要求等，然后据此进行接纳控制。如果系统内还有足够的资源可以满足应用的要求，则 Scheduler 给 Client 返回一个数值，假设为 5，这就是该应用的序列号；否则，Scheduler 将给出相关的提示信息。Client 将提示用户提供其它信息，如数据源的位置等。Client 将根据这个序列号建立应用的工作目录/storage/workDir5 和数据目录/storage/dataDir5；然后 Client 将生成 pack5.sh，其内容为：

/storage/workDir5/package –s 5 –a 4 –f /storage/workDir5/filelist1.txt –h 166.111.137.87 –w /storage/workDir5/ -d /storage/dataDir5/data –e

/storage/dataDir5/data1 –p /standalone

其含义为：封装模板为 package；数据模块的大小为 5，总量为 4；要处理的文件信息包含在 filelist1.txt 中；工作目录为 workDir5；数据应该传输到 /storage/dataDir5/data 和 /storage/dataDir5/data1；要运行的数据处理文件为 standalone。

package 会通过命令行解析获得这些参数。其中，standalone 是 rmon 的源文件经过编译以后得到的可执行文件。

Client 还会生成一个对 Condor 的提交文件，即 condor5.submit，其内容如下：

Universe=vanilla

Execute=pack5.sh

Out=pack5.out

Log=pack5.log

Error=pack5.err

Should_transfer_file=YES

When_to_transfer_file=ON_EXIT

Queue

Client 将这个文件提交给 Condor。至此，一个粗粒度的任务提交过程结束，Scheduler 将进行后台的规划和调度。然后，Condor 将为 pack5.sh 寻找合适的计算资源。pack5.sh 中的可执行文件（即封装模板 package）将进行命令行解析，以获得相关的参数，如数据模块的大小和数量、工作目录、数据目录等，并同时启动数据传输和处理线程，以数据流的方式处理数据。

6.4.3　调度结果

本例运行的结果保留在 pack5.log 和 pack5.out 中。前者是一个日志文件，记录了应用的提交时间、完成时间以及传输的数据量等；后者是一个输出文件，记录了程序执行的结果和过程中的一些信息。如果程序运行过程中出现错误，则会记录在 pack5.err 文件中。本例中，数据传输线程将通过 GridFTP 成模块的传输到数据目录中。同时，数据处理线程将从数据目录中读取数据，并计算数据文件中数据的相关系数。pack5.out 的部分信息如下：

```
  L-RDS_R_L3-751658224-16.gwf  ->  file15
Source: gsiftp://166.111.137.87/opt/rmon/data/
Dest:   file:///home/zhangwen/data1/
  H-RDS_R_L3-751658160-16.gwf  ->  file11
Processing multi list file: /home/zhangwen/rmon/multilist.txt
        Number of lists added: 2 Total data streams: 2
Processing frame list file: /home/zhangwen/rmon/list1.txt
        Number of files added: 5 Total frame files: 5
Processing frame list file: /home/zhangwen/rmon/list2.txt
        Number of files added: 5 Total frame files: 5
channel[1]=H1:LSC-AS_Q channel[2]=L1:LSC-AS_Q
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
No more requested files
rmon data monitoring is finished
Data environment has terminated with Term/Attn/finish/EOF =0/0/0/1
Source: gsiftp://166.111.137.87/opt/rmon/data1/
Dest:   file:///home/zhangwen/data2/
  L-RDS_R_L3-751658240-16.gwf  ->  file16
Source: gsiftp://166.111.137.87/opt/rmon/data/
Dest:   file:///home/zhangwen/data1/
```

6.5　小结

本章介绍了面向天文应用的数据流使能环境，是本文研究的工程实现。本研究是由应用驱动的，因此工程实现处于重要的位置。本章首先介绍了几个具体的天文数据流应用，以此说明天文数据流应用的特点和需求，并说明一些天文数据很适合以数据流的形式进行处理。作为一个实际实现的系统，本章介绍了该使能环境的具体架构，包括其物理架构、安全架构、功能架构等。本章利用一个实例，具体说明了数据流管理和调度的基本流程，重点说明了客户端如何对用户提交的应用进行封装、为用户应用建立运行环境以及如何将封装过的应用提交到系统中以获取资源等。

第 7 章　结论

本章对全文进行总结，概括全文的主要工作，说明研究结论和主要创新点，并展望需进一步开展的工作。

7.1　研究总结

本文首先分析了网格数据流应用的特点及其对资源管理和调度的独特要求，指出这种应用需要数据传输和处理的协调配合，因而这种应用同时需要计算资源、存储资源和带宽资源，且这三种资源是相互关联的，它们共同决定着数据流应用的吞吐量和资源利用率。为此，本文提出了面向这种应用的混合资源管理和调度问题，并指出这三种资源的调度和分配必须是综合的、协调的和按需的，它们需要在一个统一的框架内调度和分配。本文设计了两种混合资源调度和分配的算法，分别称为粗粒度资源调度算法和细粒度资源调度算法。它们分别适用于不同的场景，主要区别体现在对计算资源的控制和分配粒度上。

粗粒度资源调度算法适用于传统的网格应用环境中，由于不同管理域中的资源本地自治的缘故，粗粒度资源调度算法不能实现对资源的精确控制。具体说来，粗粒度资源调度算法将应用提交给 Condor，由后者将应用映射到合适的计算资源，而应用最终获得的计算资源（如 CPU 周期）的多少则取决于计算节点本地的进程调度策略。带宽资源是由一个迭代式算法分配的，它体现了数据传输对数据处理的感知特性，力图为各个应用按需分配带宽，其中的分配参数是由遗传算法根据各个应用的吞吐量之和最大的原则确定的。另一方面，数据传输受到库存策略的控制，以防止数据溢出，因而存储中的数据量总是有限的，也就是说数据流应用可以用较小的存储资源实现大量的数据处理。

细粒度混合资源调度算法的应用场景类似于云计算，即在一个机构内部管理和调度分布式资源。由于拥有对资源（主要是计算资源）的完全控制权，该算法可以在更小的粒度上实现计算资源的调度和分配。具体而言，该算法利用虚拟化技术为每个应用建立一个虚拟机，并根据虚拟机的计算资源利用率，利用模糊控制的方法动态的调整虚拟机的计算资源（主要是其获得的物理 CPU 周期）配置，以便为各个应用分配“恰恰足够”的计算资源。细粒度资源调度算

法中的带宽分配与粗粒度资源调度算法相同。细粒度的资源调度算法可以同时实现资源利用率和吞吐量两方面的目标，即其为各个应用分配的数据处理能力和传输能力可以在细粒度上达到平衡，从而可以提供更好的 QoS 保证。

本文还实现了一个面向天文数据流应用的网格数据流应用使能环境。它提供了友好的客户端工具，只需要用户提供简单的参数即可完成应用的封装和提交。使能环境中的调度器将利用粗粒度或细粒度的调度算法为应用分配资源，保证应用的顺利运行。

本文形成如下研究结论：

第一，面向数据流应用的资源管理和调度必须综合的、协调的、按需的调度和分配计算资源、带宽资源和存储资源，才能保证这种应用的顺利运行。

第二，虚拟化和自动控制技术相结合，可以实现更加精细、准确的计算资源调度和分配，有助于实现数据流应用的吞吐量和资源利用率方面的目标。

第三，对由大量小文件组成的数据集按模块进行传输和处理，可以提高处理的效率，并降低对存储空间的要求，其中每个模块的大小对处理效率有较大影响。

本文的主要创新点归纳如下：

首先，提出了面向网格数据流应用的混合资源调度的问题和解决方法。目前，网格领域的大多数资源调度方法往往关注某一种资源（主要是计算资源），其调度目标是与任务完成时间等相关联的，且这些方法要求预先知道任务的执行时间和到达时间，或要求任务的执行时间服从某种分布、任务的到达服从某种随机过程。数据流应用的特点和调度目标均与此不同。例如，数据流应用是长期运行的，其调度目标是应用的数据吞吐量和资源的利用率，而不是最短完成时间（实际上很多数据流应用并不存在预期的完成时间）等。为此本文提出了面向这种应用的混合资源管理和调度问题，并设计实现了粗粒度调度算法和细粒度调度算法，能够实现对计算资源、存储资源和带宽资源的综合的、协调的和按需的调度和分配。

其次，利用虚拟化和自动控制技术实现了对计算资源的更加精细、更加灵活的调度，真正实现了按需调度和自由调度。网格领域传统的计算资源调度一般是将某个任务映射到某个计算资源，但是这些计算资源是受到本地调度策略管理的，一个任务最终可以获得多少计算资源，取决于计算资源本地的调度策略。因此，这些调度方法对计算资源的调度粒度较粗，很难实现按需调度。在

本文提出的细粒度调度算法中，计算资源的调度粒度变得更加精细：根据计算资源与带宽资源的相互匹配情况（即计算资源的利用率），在模糊控制器的作用下，在细粒度层面上动态调整计算资源的分配，真正做到按需调度和分配。这样既可以保证应用的运行效率，又可以避免资源的浪费。

最后，本文利用简单易行的方法，解决了制约数据流应用顺利运行的封装和寻址问题，建立了数据流应用运行使能环境。一般说来，对数据流应用进行调度的难点有两个：一是对用户提交的任务进行封装，确保数据传输和处理的同步进行；二是应用和数据的寻址。用户提交到资源管理和调度系统中的应用一般只是数据处理的程序，没有考虑数据传输。要形成数据流，必须确保数据传输和处理的协调配合。为此，本文设计开发了一个通用的封装模板，只需要用户提供少量信息，即可将应用封装为包含数据传输和处理的多线程应用并提交到调度系统中。至于应用和数据的寻址，本文提出了工作目录和数据目录的概念，分别指明应用程序和数据所在地。工作目录和数据目录分别根据应用的序列号加以命名，具有全局唯一性。本文部署了网络文件系统，解决了任务提交主机和任务执行主机之间工作目录和数据目录的一致性问题，方便了应用和数据的寻址。

7.2　需进一步开展的工作

目前，云计算正风起云涌，引起了科学界和工业界的广泛重视。可以预见，在不远的未来，云计算将深刻改变人们的计算方式。但是也应当看到，云计算中还有一些问题亟待解决，其中之一就是按需提供资源，包括计算资源、带宽资源和存储资源等。目前，提供云计算服务的厂商对资源的管理还比较粗放，没有做到精细化和集约化。从本文的第 5 章可以看出，细粒度资源管理和调度方法，完全可以应用到云计算的资源管理和调度中。实际上可以看出，本文研究的网格数据流应用，已经具备云计算的特征。

本文重点研究了对计算资源的虚拟化和细粒度的管理和调度，在后续的工作中应将重点转移到对网络资源的精确控制和分配。网络的虚拟化，应当是今后的研究重点。如果实现了对网络资源的按需分配，则可以进一步提高网络资源的有效利用率，同时提高数据流应用的吞吐量。目前，关于这方面的资料和成功案例还比较少，需要进一步的探索和实践。

此外，关于虚拟机的动态迁移，也是一个值得关注的研究方向，目前已经有一些相关的研究成果。一旦实现了虚拟机的动态迁移和应用的不间断执行，整个计算系统的生态环境就会得到根本的改变，其可靠性和可用性都将得到极大的提高。

参考文献

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

[2] （美）卡尔（Carr N）. IT 不再重要. 北京: 中信出版社, 2008.

[3] 金海. 漫谈云计算. 中国计算机学会通讯, 第五卷第六期总第 40 期.

[4] www.chinagrid.edu.cn

[5] Giannella C, Han J, Pei J, et al. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. Cambridge, Mass: MIT Press, 2003.

[6] Cai Y D, Clutter D, Pape G, et al. MAIDS: mining alarming incidents from data streams. Proc. 23rd ACM SIGMOD, 2004, Paris, France.

[7] Klasky S, Beck M, Bhat V, et al. Data management on the fusion computational pipeline. Journal of physics: conf. series 16, 2005: 510-520.

[8] Aydin G, Qi Z, Pierce M E, et al. Building a sensor grid for real time global positioning system data. Proc. workshop on principles of pervasive information systems design Sunday, 2007, in conjunction with pervasive 2007 toronto, Ontario, Canada.

[9] Fox G, Wu W, Uyar A, et al. Global multimedia collaboration system. Proc. 1st int'l. workshop on middleware for grid computing, 2003, Rio de Janeiro, Brazil.

[10] Fox G, Wu W, Uyar A, et al. A web services framework for collaboration and audio/videoconferencing. Proc. int'l. multiconf. in computer science and computer engineering, internet computing, 2002, Las Vegas, USA.

[11] Bhat V, Parashar M, Khandekar M, et al. A self-managing wide-area data streaming service using model-based online control. Proc. 7th IEEE int'l. conf. on grid computing, 2006: 176-183, Barcelona, Spain.

[12] Deelman E, Kesselman C, Mehta G, et al. GriPhyN and LIGO: building a virtual data grid for gravitational wave scientists. Proc. 11th IEEE int'l. symp. on high performance distributed computing, 2002: 225-234.

[13] Pordes R. The open science grid. Proc. computing in high energy and nuclear physics conf., 2004, Interlaken, Switzerland.

[14] Fox G, Aydin G, Gadgil H, et al. Management of real-time streaming data grid services: research articles. Concurrency and computation: practice & experience, 2007, 19(7):983-998.

[15] Bhat V, Klasky S, Atchley S, et al. High performance threaded data streaming for large scale simulations. Proc. 5th IEEE/ACM int'l. workshop on grid computing (grid'04), 2004:

243-250.

[16] Klasky S, Ethier S, Lin Z, et al. Grid-based parallel data streaming implemented for the gyrokinetic toroidal code. Proc. ACM/IEEE conf. on supercomputing (SC'03), 2003.

[17] Litzkow M, Livny M, Mutka M. Condor- a hunter of idle workstations. Proc. 8th int'l. conf. on distributed computing systems, 1988: 104-111.

[18] http://www.globus.org

[19] Madduri R, Hood C, Allcock W. Reliable file transfers in grid environments. Proc. 27th IEEE conf. on local computer networks, 2002: 737-738, Tampa, Florida, USA.

[20] Ohaski H, Imase M. Performance evaluation of data transfer protocol GridFTP for grid computing. Proc. world academy of science, engineering and technology, 2006, 16: 297-302.

[21] Kola G, Livny M. DiskRouter: a mechanism for high performance large scale data transfers. http://www.cs.wisc.edu/condor/diskrouter/.

[22] Plale B, Schwan K. dQUOB: managing large data flows using dynamic embedded queries. Proc. IEEE int'l. high performance distributed computing, 2000: 263-270.

[23] Koparanova M G, Risch T. High-performance grid stream database manager for scientific data. Proc. 1st European across grids conf., 2003: 86-92, Universidad de Santiago de Compostela, Spain.

[24] Thide B. (ed.) First LOFAR/LOIS workshop. 2001, Sweden.

[25] Agarwalla B, Ahmed N, Hilley D, et al. Streamline: a scheduling heuristic for streaming applications on the grid. Proc. SPIE multimedia computing and networking, 2006.

[26] Chen L, Reddy K, Agrawal G. GATES: a grid based middleware for processing distributed data streams. Proc. 13th IEEE int'l. symp. on high performance distributed computing (HPDC-13), 2004, Honolulu, Hawaii USA.

[27] Chen L, Agrawal G. Resource allocation in a middleware for streaming data. Proc. 2nd workshop on middleware for grid computing (MGC'04), 2004, Toronto, Canada.

[28] Chen L, Agrawal G. A static resource allocation framework for grid-based streaming applications. Concurrency and computation: practice & experience, 2006, 18: 653–666.

[29] Agarwalla B, Ahmed N, Hilley D, et al. Streamline: scheduling streaming applications in a wide area environment. Multimedia systems, 2007, 13(1): 69-85.

[30] Deelman E, Blythe J, Gil Y, et al. Mapping abstract complex workflows onto grid environments. Journal of grid computing, 2003, 1(1): 25-39.

[31] Ramakrishnan A, Singh G, Zhao H, et al. Scheduling data-intensive workflow onto storage-constrained distributed resources. Proc. 7th IEEE int'l. symp. on cluster computing and the grid, 2007: 401-409, Rio de Janeiro, Brazil.

[32] Battestilli L, Hutanu A, Karmous-edwards G, et al. EnLIGHTened computing: an

architecture for co-allocating network, compute, and other grid resources for high-end applications. Proc. int'l. symp. on high capacity optical networks and enabling technologies, 2007: 1-8.

[33] Takefusa A., Hayashi M, Nagatsu N, et al. G-lambda: coordination of a grid scheduler and lambda path service over GMPLS. Future generation computer systems, 2006, 22(8): 868-875.

[34] Figuerola S, Ciullib N, Leenheerc M, et al. PHOSPHORUS: single-step on-demand services across multi-domain networks for e-science, Proc. SPIE 6784, 67842X, 2007.

[35] Arasu A, Babcock B, Babu S, et al. STREAM：the Stanford stream data manager. IEEE data engineering bulletin, 2003, 26(1): 19-26.

[36] Shivanath B, Jennifer W. Continuous queries over data streams. SIGMOD record, 2001, 30(3):109-120.

[37] Carney D, Cetinterne U, Cherniack M, et al. Monitoring streams: a new class of data management applications. Proc. conf. on very large data bases, 2002: 215-225.

[38] Cranoe C, Johnson T, Shkapenyuk V, et al. Gigascope: a stream database for network applications. Proc. int'l. conf. on management of data, 2003: 47-651.

[39] Chandrasekaran S, Cooper O, Deshpande A, et al. TelegraphCQ: continuous dataflow processing for an uncertain world. Proc. conf. on innovative data systems research, 2003, Asilomar, CA, USA.

[40] Cao J (Ed.), Cyberinfrastructure Technologies and Applications, 2009, Nova Science Publishers.

[41] Woodward P R, Anderson S E, Porter D H, et al. Distributed computing in the shmod framework on the NSF Teragrid. Technical Report, CS Dept, University of Minnesota, 2004.

[42] Deelman E, Blythe J, Gil Y, et al. Pegasus: mapping scientific workflows onto the grid. Proc. 2nd European across grids conf., 2004, Nicosia, Cyprus.

[43] 许骏, 柳泉波, 李玉顺, 等. 面向服务的网格计算——新型分布式计算体系与中间件. 北京: 科学出版社, 2009.

[44] （美）李（Li M）,（美）贝克（Baker M）著, 王相林, 张善卿, 王景丽译. 网格计算核心技术. 北京: 清华大学出版社, 2006.

[45] Krauter K, Buyya R, Maheswaran M. A taxonomy and survey of grid resource management systems for distributed computing. Int'l journal of software: practice and experience, 2002, 32(2): 135-164.

[46] Casvant T L, Kuhl J G. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE transactions on software engineering, 1988, 14(2): 141-154.

[47] Shiraz B A, Huron A R, Kavi K M. Scheduling and load balancing in parallel and distributed systems. IEEE computer sciety press, 1995.

[48] http://gridengine.sunsource.net

[49] http://www.pbsgridworks.com

[50] http://www.cisl.ucar.edu/docs/bluefire/lsf.html

[51] http://www.cs.wisc.edu/condor/condorg

[52] Chapin S J, Katramatos D, Karpovich J, et al. The legion resource management system. Job Scheduling Strategies for Parallel Processing, 1999: 162-178, Springer Verlag.

[53] Buyya R, Abramson D, Giddy J. Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid. Proc. int'l. conf. on high performance computing in asia-pacific region, 2000.

[54] Takefusa A, Matsuoka S, Nakada H. Overview of a performance evaluation system for global computing scheduling algorithms. Proc. 8th IEEE int'l. symp. on high performance distributing computing (HPDC8）, IEEE computer society press, 1997.

[55] Xia H, Dail H, Casanova H, et a1. The Microgrid：using online simulation to predict application performance in diverse grid network environments. Proc. 2nd int'l.workshop on challenges of large applications in distributed environments, IEEE computer society press, 2004.

[56] Legrand A, Marchal L, Casanova H. Scheduling distributed applications：the SimGrid simulation framework. Proc. 3rd IEEE/ACM int'l. symp. on cluster computing and the grid, IEEE computer society press, 2003.

[57] Buyya R, Murshed M. GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for grid computing. Concurrency and computation: practice & experience, 2002, 14(13-15): 1175.

[58] Bell W H, Cameron D G, Capozza L, et a1. Simulation of dynamic grid replication strategies in OptorSim. Proc. 3rd int'l. workshop on grid computing, 2002, Spring Verlag.

[59] Ranganathan K, Foster I. Decoupling computation and data scheduling in distributed data-intensive applications. Proc. 1st IEEE int'l. symp. on high performance distributed computing, 2002, Edinburgh, Scotland.

[60] Figueiredo R, Dinda P, Fortes J. A case for grid computing on virtual machines. Proc. 23rd int'l. conf. on distributed computing systems, 2003.

[61] Keahey K, Doering K, Foster I. From sandbox to playground: dynamic virtual environments in the grid. Proc. 5th int'l. workshop in grid computing, 2004.

[62] Foster I, Freeman T, Keahy K, et a1. Virtual clusters for grid communities. Proc. IEEE int'l. symp. on cluster computing and the grid, 2006.

[63] Khanna G, Beaty K, Kar G, et a1. Application performance management in virtualized server environments. Proc. IEEE/IFIP network operations and management symp., 2006.

[64] Ruth P, Rhee J, Xu D, et a1. Autonomic live adaptation of virtual computational

environments in a multi-domain infrastructure, Proc. IEEE int'l. conf. on autonomic computing (ICAC), 2006.

[65] Shivam P, Demberel A, Gunda P, et a1. Automated and on-demand provisioning of virtual machines for database applications. Proc. ACM SIGMOD int'l. conf. on management of data, 2007.

[66] Steinder M, Whalley I, Carrera D, et a1. Server virtualization in autonomic management of heterogeneous workloads. Proc. IFIP/IEEE int'l. symp. on integrated network management, 2007.

[67] Wang X, Du Z, Chen Y, et a1. Virtualization-based autonomic resource management for multi-tier web applications in shared data center. Journal of systems and software, 2008.

[68] Wang X, Lan D, Wang G, et a1. Appliance-based autonomic provisioning framework for virtualized outsourcing data center. Proc. IEEE int'l. conf. on autonomic computing (ICAC), 2007.

[69] Barham P, Dragovic B, Fraser K, et a1. Xen and the art of vrtualization. Proc. 9th ACM symp. on operating systems principles, 2003: 164-177, N.Y., USA.

[70] Hellerstein J L, Diao Y, Parekh S, et a1. Feedback Control of Computing Systems, Wiley-IEEE Press, 2004.

[71] Abdelzaher T F, Shin K G, Bhatti N. Performance guarantees for web server end-systems: a control-theoretical approach. IEEE trans. on parallel and distributed systems, 2002, 13.

[72] Parekh S, Gandhi N, Hellerstein J L, et a1. Using control theory to achieve service level objectives in performance management. Real time systems journal, 2002, 23(1-2).

[73] Diao Y, Gandhi N, Hellerstein J L, et a1. MIMO control of an apache web server: modeling and controller design. Proc. American control conf., 2002.

[74] Kamra A, Misra V, Nahum E M. Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites. Proc. 12th IEEE int'l. workshop on quality of service, 2004.

[75] Lu Y, Lu C, Abdelzaher T, et a1. An adaptive control framework for QoS guarantees and its application to differentiated caching services. Proc. 10th IEEE int'l. workshop on quality of service, 2002.

[76] 石辛民, 郝整清. 模糊控制及其 MATLAB 仿真. 北京：清华大学出版社, 北京交通大学出版社, 2008.

[77] King P J, Mamdani E H. Application of fuzzy algorithms for control simple dynamic plant. IEEE proc. control theory app., 1974, 121: 1585-1588.

[78] Diao Y, Hellerstein J L, Parekh S. Using fuzzy control to maximize profits in service level management. IBM systems journal, 2002, 41(3).

[79] Li B, Nahrstedt K. A control-based middleware framework for quality of service adaptations.

Communications, 1999, 17: 1632-1650.

[80] Suzer M H, Kang K D. Adaptive fuzzy control for utilization management. Proc. IEEE int'l. symp. on object/component/service-oriented real-time distributed computing, 2008.

[81] Park S M, Humphrey M. Feedback-controlled resource sharing for predictable escience. Proc. ACM/IEEE conf. on supercomputing, 2008.

[82] http://aws.amazon.com/ec2/

[83] Singh N. Systems Approach to Computer-Integrated Design and Manufacturing. Johh Wiley & Sons, New York, 1996.

[84] Hitomi K. Manufacturing Systems Engineering, 2nd ed. Taylor & Francis, London, 1996.

[85] Hart E, Ross P, Corne D. Evalutionary scheduling: a review. Genetic programming and evolvable machines, 2005, 6: 191-220.

[86] 王凌, 刘波. 微粒群优化与调度算法. 北京: 清华大学出版社, 2008.

[87] 李士勇, 陈永强, 李研. 蚁群算法及其应用. 哈尔滨: 哈尔滨工业大学出版社, 2004.

[88] （意）Marco D, （德）Thomas S 著, 张军, 胡晓敏, 罗旭耀, 等译. 蚁群优化. 北京: 清华大学出版社, 2007.

[89] 王正志, 薄涛. 进化计算. 长沙: 国防科技大学出版社, 2000.

[90] 张文修, 梁怡. 遗传算法的数学基础. 西安: 西安交通大学出版社, 1999.

[91] 朱剑英. 智能系统非经典数学方法. 武汉: 华中科技大学出版社, 1999.

[92] （日）玄光男, 程润伟 著, 汪定伟, 唐加福, 黄敏 译. 遗传算法与工程设计. 北京: 科学出版社, 2000.

[93] Goldberg D. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA, 1989.

[94] 徐志伟, 冯百明, 李伟. 网格计算技术. 北京: 电子工业出版社, 2004.

[95] 丁宝苍. 预测控制的理论和方法. 北京: 机械工业出版社, 2008.

[96] Zhang W, Cao J, Zhong Y, et al. Block-based concurrent and storage-aware data streaming for grid applications with lots of small files. Proc. 9th IEEE/ACM int'l. symp. on cluster computing and the grid, 2009: 538-543, Shanghai, China.

[97] Zhang W, Cao J, Zhong Y, et al. Scheduling data blocks for concurrent and storage-aware grid data streaming. to appear in int'l. journal of utility and grid computing.

[98] 陈宝林. 最优化理论与算法（第 2 版）. 北京: 清华大学出版社, 2005.

[99] Kar K, Sarkar S, Tassiulas L. A simple rate control algorithm for maximizing total user utility. Proc. IEEE INFOCOM, 2001.

[100] Ito T, Ohaski H, Imase M. Automatic parameter configuration mechanism for data transfer protocol gridftp. Proc. int'l. symp. on applications and the internet (SAINT 2006), 2006: 32-38.

[101] Ito T, Ohaski H, Imase M. GridFTP-APT: automatic parallelism tuning mechanism for data transfer protocol GridFTP. Proc. 6th IEEE int'l. symp. on cluster computing and the grid (CCGrid2006), 2006: 454–461.

[102] Inoue F, Ito T, Ohsaki H, et al. Implementation and evaluation of gridftp automatic parallelism tuning mechanism for long-fat networks. Proc. 7th Asia-Pacific symp. on information and telecommunication technologies (APSITT), 2008: 189-194.

[103] Ito T, Ohsaki H, Imase M. On parameter tuning of data transfer protocol gridftp in wide-area grid computing. Proc. 2nd int'l. workshop on networks for grid applications, 2005: 415–421.

[104] Jain M, Prasad R S, Dovrolis C. The TCP bandwidth-delay product revisited: network buffering, cross traffic, and socket buffer auto-sizing. Technical report, GIT-CERCS-03-02, Georgia Institute of Technology, 2003.

[105] Prasad R, Dovrolis C, Murray M, et al. Bandwidth estimation: metrics, measurement techniques and tools. IEEE network, 2003, 17: 27-35.

[106] Zhang W, Cao J, Zhong Y, et al. Concurrent and storage-aware data streaming for data processing workflows in grid environments. to appear in *T*singhua Science and Technology.

[107] Liu X, Zhu X, Singhal S, et al. Adaptive entitlement control to resource containers on shared servers. Proc. 9th IFIP/IEEE int'l. symp. on integrated network management, 2005, Nice, France.

[108] Zhang W, Cao J, Zhong Y, et al. Fuzzy allocation of fine-grained compute resources for grid data streaming applications. to appear in Int'l. journal of grid and high performance computing.

[109] https://www.lsc-group.phys.uwm.edu/daswg/download/repositories.html

[110] http://montage.ipac.caltech.edu

[111] http://fits.gsfc.nasa.gov

[112] http://www.cv.nrao.edu/fits

致　谢

本文是在导师钟宜生教授、曹军威研究员的悉心指导下完成的。在笔者进行博士论文研究其间，两位老师从学术、人生乃至个人生活的方方面面给予了无微不至的关怀和指导。钟宜生教授温柔敦厚的风范、高深的学术造诣、淡然的人生态度，深刻的影响着我。在论文研究期间，曹军威研究员给予了具体的指导和帮助。曹军威研究员对学术前沿的把握、对科学研究的热忱，给笔者树立了学习的典范。感谢两位老师为我付出的心血！我相信，在今后的人生道路上，我仍将从中受益。

感谢邢笑笑、赵斌强两位师兄。他们多次给予极富建设性的意见和建议，使笔者的研究取得进展。他们扎实的理论功底、严谨的科学态度，给我留下了深刻的印象。感谢实验室的余瑶、郑博、张帆、王震、李俊伟、侯勇等同学，我与他们一起度过了难忘的求学时光。

衷心感谢中国人民解放军总装备部重庆军事代表局的首长和领导们，特别感谢驻七八九厂军事代表室的魏国总代表和全体同事！在我读书期间，单位的领导和同志们从组织上、生活上、思想上关心着我，让我始终感受到浓浓的暖意。在单位人手紧、任务重的情况下，他们始终如一的支持着我，给了我无穷的动力和坚定的信心。

最后，感谢我的家人！这么多年，他们一直默默的支持着我。我相信，顺利完成学业，是对他们最好的报答！

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：_____

个人简历、在学期间发表的学术论文与研究成果

个人简历

1978 年 6 月 1 日出生于山东潍坊。

1996年9月考入中国人民解放军军械工程学院光学与电子工程系火力与指挥控制专业，2000 年 7 月本科毕业获得工学学士学位并被保送为硕士研究生；2003 年 4 月获工学硕士学位。

2003 年 4 月至 2005 年 8 月在中国人民解放军总装备部重庆军事代表局工作。

2005 年 9 月进入清华大学自动化系攻读控制科学与工程博士至今。

已发表的学术论文

[1] Zhang W, Cao J, Zhong Y, et al. An integrated resource management and scheduling system for grid data streaming applications. Proc. 9th IEEE/ACM int'l. conf. on grid computing (Grid 2008), 2008: 258-265, Tsukuba, Japan. (EI 收录，20090111822653）

[2] Zhang W, Cao J, Zhong Y, et al. Agile data streaming for grid applications. Proc. 7th int'l. conf. on grid and cooperative computing, 2008: 739-746, Shenzhen, China. (EI 收录，20090111828659）

[3] Zhang W, Cao J, Zhong Y, et al. Storage aware resource allocation for grid data streaming pipelines. Proc. IEEE int'l. conf. on networking, architecture, and storage, 2008: 179-180, Chongqing, China. (EI 收录，20083911584752）

[4] Zhang W, Cao J, Zhong Y, et al. Block-based concurrent and storage-aware data streaming for grid applications with lots of small files. Proc. 9th IEEE/ACM int'l. symp. on cluster computing and the grid, 2009: 538-543, Shanghai, China. (EI 收录，20094212379065）

已录用的学术论文

[1] Zhang W, Cao J, Zhong Y, et al. Scheduling data blocks for concurrent and storage-aware grid data streaming. International journal of utility and grid computing. （EI 源刊）

[2] Zhang W, Cao J, Zhong Y, et al. Fuzzy allocation of fine-grained compute resources for grid data streaming applications. International journal of grid and high performance computing. （EI 源刊）

[3] Zhang W, Cao J, Zhong Y, et al. Concurrent and storage-aware data streaming for data processing workflows in grid environments. Tsinghua science and technology. （EI 源刊）

在学期间参与出版的专著

[1] Zhang W, Cao J, Zhong Y, Liu L, Wu C. Grid Data Streaming. In J. Wong (Ed.), Grid Computing Research Progress, 2008: 267-290, Nova Science Publishers.

[2] Zhao C, Cao J, Wu H, Chen W, Sun X, Zhang W, Hou Y. Adaptive Query Processing in Data Grids. In Antonopoulos N, Exarchakos G, Li M, Liotta A (Eds.), Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications, IGI Publishing, 2009.

[3] Zhao C, Wan Y, Wu H, Zhang W. Cyberinfrastructure for Agricultural Data and Knowledge Sharing in China. In Cao J (Ed.), Cyberinfrastructure Technologies and Applications, 2009: 317-330, Nova Science Publishers.

在学期间参与的研究项目

[1] 国家高技术研究发展计划（863 计划）项目“数字农业知识网格关键技术研究”（No.2006AA10Z237），2006 年 12 月-2010 年 6 月。

[2] 国家高技术研究发展计划（863 计划）项目“海量农业知识资源组织、高效存储技术研究”（No.2007AA01Z197），2007 年 7 月-2009 年 12 月。

[3] 国家高技术研究发展计划（863 计划）项目“大规模网络数据集成与查询技术及其应用研究”（No. 2008AA01Z118），2008 年 1 月-2009 年 12 月。

[4] 教育部高等学校本科教学质量与教学改革工程项目“国家精品课程集成项目门户分系统研制”，2007 年 6 月-2010 年 6 月。

[5] 国家自然科学基金项目“动态时变约束下的赛百平台资源优化理论与算法研究”（No.60803017），2009 年 1 月-2010 年 6 月。