清 华 大 学

综 合 论 文 训 练

题目：基于虚拟化技术的面向数据流应用的网格资源调度研究

系　　别：自动化系

专　　业：自动化

姓　　名：陈威威

指导教师：曹军威　研究员

2009 年 6 月9 日

关于学位论文使用授权的说明

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

(涉密的学位论文在解密后应遵守此规定)

签 名：[illegible] 导师签名：[illegible] 日 期：2009.06.24

中文摘要

本文研究了在网格中基于虚拟化技术的面向数据流应用的细粒度资源调度问题。并提出一种融合最优控制、系统辨识和虚拟化技术的综合算法，能够达到提高资源利用效率和提高工作效率的平衡。通过开发出基于 Xen 虚拟化技术的 ViGrid 虚拟网格系统，我们测试了了该算法的可靠性。通过分析 Rmon 和 Montage 两个典型的网格应用，我们验证了该算法的实际运行效果并且讨论了存在的问题。

关键词：网格；资源调度；数据流；准虚拟化

ABSTRACT

Abstract: In this paper we present resource management in data grid. To deal with the on-demand and fine-grained resource allocation problem, we proposed an integrated algorithm that incorporate virtualization, system identification and optimal control techniques. We built a ViGrid system as a testbed to test the performance of this method. We also applied the Rmon and Montage application in this system and verify our method.

Keywords：Grid; Resource management; Data Stream; Virtualization

目 录

第1章 引 言

1.1 网格技术概述

网格(Grid) 是近年出现的新兴技术。从 20 世纪 90 年代中期出现的元计算系统开始, 网格的发展过程可以分为三个阶段[1]。网格在当时被称为元计算环境(Metacomputing Environment),通常用来连接超级计算机为高性能要求的应用提供计算资源, 典型的系统有 I-WAY; 在网格发展的第二阶段, 异构、分布的资源共享问题得到了相当的重视, 中间件和标准化是解决资源异构性的关键技术, 这个阶段主要的研究包括 Globus(3.0 版本以下) , Legion, Nimrod /G 等;在网格发展第三阶段中, 面向服务的模型和元数据是最关键的概念, 两者构成了 OGSA(Open Grid Service Architecture) 的核心思想[2] 。

网格能吸纳各种类型的计算机资源甚至相关的人力资源,并将它们转换成为标准的、经济的、随处可得的计算能力, 为用户提供高质量的服务。通常个人计算机在大多数时间中是空闲的，即使是实验室里的服务器在大多数时间内也只是在设备之间来回发送电子，显示最新的屏幕保护程序之外大部分的资源也是闲置的。据估计，一台计算机用来执行重要处理任务的时间少于其处理能力的 5% 。甚至大多数服务器只使用其 15% 以下的处理能力来满足用户的需要，剩下的服务器时(CPU、RAM、硬盘等)仍然空闲着未被使用[3]。网格计算(Grid Computing)技术的出现，可以将地理上分布、系统异构的多种计算资源通过高速网络连接起来，将它们各自的计算资源、存储资源、通信资源等结合起来，形成一种开放式的环境，共同解决各个学科方面的大规模高性能计算和协同科学研究。

Ian Foster 认为网格的含义本质上应该包括以下三点[2],[4]:

(1) 协调各种非集中控制的资源。网格中的资源在地理上和管理权上都是分散的, 协调分散的资源是网格最基础的功能。

(2) 使用开放、标准、通用的协议和接口。标准的协议和接口使得各方面能动态地在资源共享方面达成协议, 也是建立通用的服务和工具的重要方式。

(3) 给用户提供高质量的服务。网格中各资源应该能协作满足用户各种服务要求, 如高吞吐量的服务要求、迅速响应的服务要求等。

虽然网格经历了十多年的发展，并出现了一些比较成功的研究成果，但还有很多重要问题没能解决，资源管理中的调度问题就是其中之一。网格中的资源管理可以分为资源发现、资源调度、任务提交和监视这几个部分，其中资源发现、任务提交和监视可以由现存的中间件系统，如 Globus 工具箱[5]，提供的标准服务来完成，但资源调度，即为特定的任务选择合适的资源,在 Globus 的研究中很少涉及，其他的研究往往也只是对少数特定的应用提出了各自的调度策略，因此资源调度是资源管理的难点。

1.2 网格中的资源调度

20 世纪 70~80 年代因特网(Internet)技术的诞生，从此改变了人们的思维方式及工作模式，第一代 Internet 技术用 TCP/IP 协议实现了地理上分布的计算机之间的连接，主要用于传递电子邮件 E-mail。第二代 Internet，是万维网(World Wide Web)技术，主要通过超文本协议将数据和信息组织起来，以 Web 浏览的方式实现信息服务。第三代 Internet，是网格计算(Grid Computing)，其含义是把分布在不同地理位置的计算机、贵重仪器、数据库等资源用高速网络连接在一起。同时研究开发相应的中间件，使这些资源看起来就像一台单一映像的虚拟机器，用户通过网格共享资源及建立虚拟实验室，共同讨论以及合作开展科研项目。

网格计算实质上是使用网络中的多个计算资源来解决单一问题的计算模式。当一个计算工程需要的处理资源超过本地可提供的能力时，网格计算允许该计算工程通过网络使用远程机器的 CPU 和存储资源。因此在网格环境中，可能有大量的应用需要共享网格中的各种资源，比如处理器周期、网络带宽、数据库、共享的打印机和其它设备。因此网格资源管理是网格计算的核心问题之一。

在网格中，广义的资源可以是任何能为用户所分享的软件和硬件，比如处理器周期、共享的打印机和其他设备、网络带宽、数据库等。传统意义上的资源管理系统是指单台计算机或小规模局域网范围内，对资源有完全的控制，故可在与外界隔离的情况下实现高效的管理机制和调度策略。而网格资源管理是在开放的广域网内考虑，对资源无完全的控制，对资源的状态变化不可预料，且异构的资源大大复杂化了资源管理任务。网格资源管理是网格的核心组件，网格资源是指网络上能够被共享和利用的任何能力，包括传统意义上的物理资源，如计算资源、带宽资源、存储资源、传感器等，也包括虚拟的服务，如数据库、数据传输、仿真等软件应用。严格的说，网格资源管理不是有关资源和服务的核心功能，即资

源和服务能够为客户做什么，而是指功能执行的方式，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。网格中资源的主要特点如下：

(1)分布性，是指网格中的资源分布在地理上不同的地方，而不是集中在一起的。网格资源虽然是分布的，但却可以充分共享的。分布是网格硬件在物理上的特征，而共享是在网格软件支持下实现的逻辑上的特征。

(2)自主性，网格中不同资源的拥有者对其资源有自主的管理能力，网格资源管理必须尊重所有者和管理者的自主权。但是网格资源也必须接受网格的统一管理，不然资源之间就不能建立联系，实现共享和互操作。

(3)异构性，网格环境中存在不同体系结构的计算机系统和不同属性、不同类别的资源。

(4)动态性，网格中资源的状态不断变化的，某一时刻可用的资源可能在下一时刻就会出现故障或者不可用，而新的资源随时可能会加入进来。例如，CPU 在处理任务的时候，每个时刻的负载可能不相同；网络的链路带宽的流量每时每刻都在变化中；内存的占有量随着进程状态的变化而变化。

资源调度是资源管理的核心部分，无论是特定任务的执行性能，如时间、费用等，还是整个系统的吞吐率、资源利用率都受到资源调度质量的决定性影响。同时资源调度也是资源管理中极其繁琐复杂的问题。

资源调度的基本原则是保障资源为完成尽可能的任务服务，不能出现死锁是基本的要求，还要考虑资源在时间和空间上的搭配。目的是为了完成用户提交的任务和满足用户提出的要求，把网格中所有可用资源(计算资源、存储资源和网络资源等)进行匹配，找到最合理的资源分配方式和资源调度策略。目标是各个主机能够得到适合自己的任务，并且几乎能够同时完成任务。

1.3 面向数据流的网格资源管理

本文所针对的主要是网格中面向数据流的资源管理。网格中越来越强调数据流管理[6]，越来越多的应用不是在本地完成而是将数据中心和处理中心进行分离，即使是处理中心也可能不是集中式的而是分布式的节点。这一方面是基于安全的考虑，不能够在本地安装和运行处理服务，另一方面也是从效率考虑，数据中心本身不具备足够的资源来完成这些运行任务。与网格中的其他面向对象不同，数据流应用需要相结合的足够的带宽，足够的存储和计算能力，以保证稳定的和高效率的处理。一个典型的例子是 LIGO （激光干涉仪引力波天文台）[7]，正在从

开放科学网格 Open Science Grid(OSG)中获得更多的处理资源[8]。由于大多数的 OSG 站点都是 CPU 资源富余而储存空间有限，从而没有更多的 LIGO 数据能够进入处理系统。所以数据流支持是必要的，以利用 OSG 提供的 CPU 资源。当数据流量过大时，而流量应该减少，而且也不应该继续送入到计算系统，同时计算系统不断处理和执行这些任务使得数据总能够为本地存储获得。尤其提出了细粒度的资源管理要求，这也是本文着重要讨论的问题。

这种应用的特点在于：

1、数据流持续和长期运行;

2、能够根据网格系统的变化和用户请求 的改变实时调整计算资源。

3、由于有限的存储资源和大量的数据等待被处理，系统不可能存储所有的数据;

4、系统需要提高资源利用效率，同时也需要提供较高的服务水平，需要在这两者中获得一个平衡。

这里存在的一个巨大的挑战是如何提供足够的但不是多余的资源（主要是计算资源和带宽），来使他们能够满足他们的服务水平目标（Service Level Objectives SLOs），同时保持高资源利用率。在分配资源的时候既要满足一定应用的需要，又不能过多分配给单一节点，从而挤压其他应用乃至整个任务的资源。我们在接下来的分析中将会详细介绍我们是如何来满足这一细粒度的资源分配要求的。以往的算法往往集中于对某种计算资源单一的调整，但是面向数据流的资源分配需要更加综合和协调的多种资源管理，虚拟化技术是实现这一方案的核心。

网格资源是共享的。大量的并发访问将导致计算资源紧张。因此某一应用或者任务不应占用超出他们需求的多余的资源，更不用说占用是应为所申请的资源支付代价，即使他们根本不使用所有资源。因此，对资源的提供者和消费者两者来说，细粒度和随需求而变的资源分配是有必要且可取的，尤其是在数据流应用中。在提高数据流应用的效率的同时，吞吐量意味着在一个特定的时间内处理的数据量，是由计算能力和带宽共同确定的。正如证明我们以前的工作[9]，计算和带宽的分配必须以合作和综合的方式。在这种情况下，单方面分配多余的计算资源或带宽不一定会提高数据吞吐量，而仅仅会导致利用率远远低于整体能力;另一方面，或者计算不足或带宽将成为最终的吞吐量的一个瓶颈。所以为了能够获得高吞吐量，必须细粒度地分配带宽和计算资源，使其达到合理、协调的一个平衡，同时保持较高的资源利用效率。为了解决计算系统的随机性和时变的特性，我们

设计一种自动分配机制，可以对环境的变化做出快速的反应，使解决上述目标是可行的。一个可行的办法就是通过系统辨识和最优控制组成闭环反馈控制。

反馈控制已应用于计算系统[10]并且得出了一些有希望的结果。闭环控制器不断根据预先设置的设定点（控制理论中所说的参考）和目标系统的输出（称为产出）之间的误差，来启动其控制算法（如比例、积分和微分，即 PID）并且输出控制变量，以调节输出变量，即使存在意外干扰也能够使系统达到的理想状态。可以看出，在传统的反馈控制，这个数学模型是必不可少的。但是在具有挑战性的某些情况下这一模型并不适应，特别是在数据流应用中因为需要处理因变量耦合和大量的非线性系统。比如在网格应用中，系统参数（包括内存、CPU 和带宽以及网络拓扑结构）和用户请求都是时变且非线性的，所以简单的反馈控制在这种应用中有局限性。

为了解决这一问题，我们引入了系统辨识和最优控制的理念。我们通过系统辨识获得这一动态变化的模型参数，并通过最优控制器获取下一时刻的控制量输出。系统参数（包括内存、CPU 和带宽）都是时刻在变化的，它们也是最优控制器所要调整的主要参数，因此一个在线的系统辨识是有必要的。在线的系统辨识是根据系统的输入输出时间函数来确定描述系统行为的数学模型，是现代控制理论中的一个分支。对系统进行分析的主要问题是根据输入时间函数和系统的特性来确定输出信号。对系统进行控制的主要问题是根据系统的特性设计控制输入，使输出满足预先规定的要求。通常需要给出这一系统的特点，考虑到计算系统本身不是特别复杂，我们可以用 ARMAX 模型来描述，并且这一阶数不需要较高。

1.4 虚拟化技术

虚拟化技术已应用于网格计算领域中 [11],[12][13]。虚拟化技术在这几年的快速进步，如 Xen[14]，为细粒度分配计算资源提供了一种可能。虚拟机（VMs）能够在一台主机上独立配置多种客户环境的主机资源，同时保证各个性能之间的独立性。虚拟机也有能力进行动态配置，使得我们能够根据不断变化的应用需求来动态分配计算资源网格，从而获得精细的粒度和更高的利用率。

虚拟计算机的概念最早由 IBM 公司在上世纪六七十年代提出， 并将其运用于 VM/370 系统中以共享昂贵的大型机系统（Main Frame）。之后的发展起起伏伏，一度由于分时操作系统的出现而处于停滞状态。上世纪九十年代随着 JAVA 虚

拟机的推出，尤其是之后 Vmware 公司 Vmware ESX server 和 Vmware workstation 虚拟机的推出，使对虚拟化技术的研究再次成为处理器设计人员、软件设计人员、服务器设计人员和网络安全设计人员的热门研究课题。

如图 1.1 所示，虚拟化技术通过在现有平台（机器）上添加一层薄的虚拟机监控程序（Virtual Machine Monitor，简称 VMM）软件而实现对系统的虚拟化，如虚拟处理器，虚拟内存管理器（MMU）和虚拟 I/O 系统等。

虚拟机监控程序又被称之为监管程序（Hypervisor）。从应用程序的角度看，程序运行在虚拟机上同运行在其对应的实体计算机上一样。虚拟化技术使得一台物理计算机可以生成多个不同的虚拟机分别运行多个不同或相同的操作系统。虚拟化技术通过将不同的应用运行在不同的虚拟机上，可以避免不同应用程序之间的互相干扰，例如一个应用的崩溃不会影响到其它的应用等。这种由虚拟化技术实现的各个应用之间的完全隔离在服务器领域具有尤其重要的意义，同时虚拟化技术也可以使得企业，高校或研究所可以在不必购置大量物理计算机的情况下实现大规模的计算机网络以从事生产及研究，例如网络及网络应用研究， 操作系统内核（Kernel）软件的开发和用户操作系统的开发等。

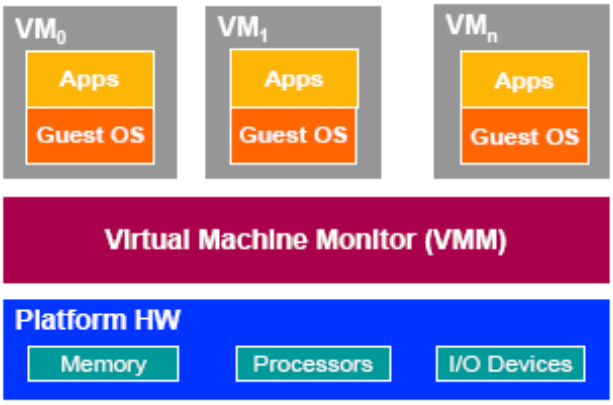


图1.1 虚拟化软件：一个物理平台同时运行多个操作系统

VMM 抽象的虚拟机的指令集（Instruction Set Architecture 简称 ISA）可以等同于它运行的物理机器，也可以作些微修改。当虚拟的指令集与物理的指令集相同时，该虚拟机可以运行没有任何修改的操作系统；而当两者不完全相同时，客

户机的操作系统就必须在源代码级或二进制代码级作相应修改。根据是否需要修改客户机操作系统的源代码，虚拟化技术又可以分为（1）准虚拟化或半虚拟化（Para-virtualization）和（2）完全虚拟化（Full-virtualization）。完全虚拟化由于不需要修改客户机操作系统，因此具有很好的兼容性和同时支持异种操作系统或不同版本操作系统的能力。相反准虚拟化技术则通常具有比完全虚拟化技术更好的性能。我们开发的 ViGrid 系统是基于准虚拟化技术的。

准虚拟化技术在最早的 IBM VM/370 上就已经使用，但它的使用仅仅是为了支持传统的操作系统，因此被限制在很小的范围。美国华盛顿大学计算机科学与工程系的 Steven D. Gribble 领导的 Denali 项目和英国剑桥大学计算机实验室的 Ian Pratt 和 Keir Fraster 领导的 Xen 项目组实现了 X86/PC 上的准虚拟化，使准虚拟化技术重新成为最热门的虚拟化技术之一。随着 Intel 公司在 2005 年初推出基于处理器硬件的虚拟化技术（Intel Virtualization Technology 简称 IntelVT 技术），支持未经修改的操作系统的完全虚拟化技术同准虚拟化技术一样成为当今虚拟化领域中的两个主要研究方向。

1.5 网格资源管理的相关研究现状

有很多文献讨论了在计算系统中通过控制理论来实现适应性。比如比例积分微分控制(PID) 就被应用在[15],[16]来支持表现优化效果和 QoS 支持 Apache 网络服务器。线性二次型调节器[17] linear quadratic regulator (LQR) 网络服务器的参数调整来改善 CPU 和内存的利用率。模糊控制也被应用在[18]IBM Lotus Notes Email Servers 来改善商业级别的矩阵比如利润。适应性控制在[19]中来改善应用层次的表现如响应时间，三层的电子商务网站的吞吐量。在[20]中，一个适应性的多变量控制器可以动态调整资源共享来独立的多个应用来满足特定层次的拂去区分。他们的工作也有相同的动力去保存 QoS 区分在某一个范围，跟我们的相比，尽快我们是通过一个不同的虚拟化技术来实现的。

目前主流的虚拟化技术有几种[21]-[24]。由于这些虚拟化技术被越来越多的广泛采用，研究学者也有越来越大的兴趣去研究如何自动化的部署和控制各种虚拟化的应用[25]-[30]。这些工作针对不同问题采取各不相同的控制机制，包括所开发的不同性能的建模方法和优化目标。但是，一个共同特点是，这些工作都有将应用程序视为黑盒子这一简单的模式特点，通常是系统也仅仅由少数几个参数决

定。还有些工作中的研究了自动化的应用部署和控制，包括资源分配和动态配置，但是并没有利用虚拟化技术[31]-[34]。虚拟化设计的问题，在[36] 也被提出来，但是没有得到解决。Ahmod[35]等人也在目标函数中引入 QoS 服务质量来改进模型，但是这些工作都针对各个具体的系统并且根据系统的不同性能做出了改进。较多的工作集中在数据库系统及适应性查询[28]，如调整数据库系统配置的具体工作量或执行环境[37]和问题的决策数据库系统更多适应在其使用的计算资源[38]-[42] 。

有一些资源管理与调度的研究[43]-[46]，但是他们侧重于分配问题某一个固定的资源，或通过调度运行于现有资源之上的查询和执行器。相比之下，我们不仅仅综合了虚拟化技术、最优控制和系统辨识的技术，更将它针对面向网格的一般应用，通过调整普适性的计算资源和带宽来获得更好的控制效果。

1.6 本文结构

本文的剩余章节是这样的安排的，第二章将介绍系统辨识、最优控制和 VM 系统，并且介绍仿真结果，第三章将着重讲述涉及到的 XEN 虚拟技术及我们搭建的 ViGrid 虚拟机网格平台，第四章将重点介绍 Rmon 和 Montage 两个网格应用的实验结果和分析。第五章给出结论。

第2章 理论分析及系统建模

本章将从理论上分析资源调度系统，构建系统的数学模型，提出基于最优控制器和系统辨识器的综合算法，并介绍这一套算法在 Matlab 平台上运行的仿真结果。仿真结果表明，我们的算法是有效的，并且具有良好的适应性。

2.1 适应性控制策略

在这项论文中，我们认为这一数据网格中的资源调度系统如图 2.1 数据流资源管理系统模型所示 。数据网格通常是由许多节点组成的，每一个都能够提供不同的数据服务。这篇论文不讨论数据复制策略应用，即不考虑不同的节点处理同一数据集。数据网格的应用，比如地震数据分析和处理，通常来说是一种管道式的任务，每一个都去处理不同数据集。用户不断发送请求进行网格数据处理，每一个都有不同程度优先级也即不同级别的 QoS 要求。QoS 的具体定义还需要根据应用和节点的不同而改变，但是都具有提高用户体验的共性。

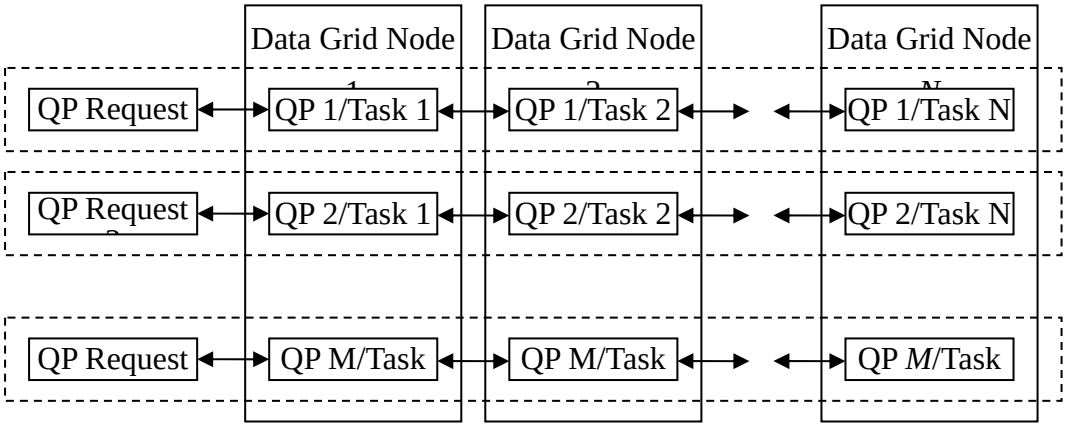


图2.1 数据流资源管理系统模型

一般而言，一个数据网格节点包括大型的存储器和相应的处理器，多个处理请求。其中一个关键特征是，所有的网格节点都是相互共享而不是针对网格的一个专门节点。因此，现有节点量化参数是随节点而不同且随着时间的推移而变化的。网格节点总是把最高优先给本地用户（资源拥有者），然后才是分享资源网络内的用户。当所有处理请求总数超过了一个节点的现有能力时，该节点的处理能力变得饱和，不能满足所有处理要求的 QoS。在这种情况下，将针对不同的网格的用户分配不同的优先级别和计算资源。

此外，除了多重请求同时访问一个节点的数据集，一个请求的不同任务组成访问不同的节点也是高度相关的。例如，一些科学的数据分析应用中，任务是通过类似管道的形式进行的，每一个循环通过一个数据集。在每一圈的任务，结果被移交给下一个节点并进行进一步数据查询和处理。分配越多的资源给一个任务，就能够实现越多的数据处理循环，并获得服务质量较高的水平。为了实现更高的端到端的服务质量，在一个管道内的每个任务也需要进行协调。降低某一任务的资源分配将导致减少前往下一个任务的负载，同时也为同处于本物理机上的其他虚拟机让渡出了资源，从而为该虚拟机获得更多的资源提供了可能。这种依赖性需要得到注意。

用 N 来表示数据网格应用中的数据集和任务的数量，每一个都位于一个数据网格节点，正如前文所述，我们不考虑同一个任务在多个节点被执行的情况，虽然出于实际情况会考虑这种冗余设计。节点 i（ i= 1,2 ， ... ... ， N ）的总处理能力模型，可以归一化并且最高位 100%。设 M 是从不同的用户中发出的有不同的 QoS 要求的并发请求。

在本文中，后述的服务质量主要是以任务完成的时间来计量的。推导一下速度与时间的关系。

让 t_{ij} 来表示所分配给节点 i 中任务 j 的资源，由于节点 i 的总处理能力是受到限制的：

$$\sum_{j=1}^{M} t_{ij} = p_i (1 \le i \le N)$$

因此总共有(M-1)*N 个独立的变量。

令 y_j (j=1,2,……,M)为针对请求 j 的归一化之后的 QoS 比率。所要求的针对请求 j 的 QoS 比率表示为 Q_j (j=1,2,……,M). 因此有:

$$\sum_{j=1}^{M} y_j = 1,$$

上式中总共有 M-1 个独立的输出。

我们主要强调的任务就是去寻找针对所有 i 和 j 都合适的 t_{ij}，因此有:

$y_j = Q_j (1 \le j \le M-1)$.

2.2 适应性控制器

在第三部分描述过的问题可以通过已有的控制方法来解决。如图 2.2 在线系统识别的适应性控制器所示，一个闭环的控制系统可以根据用户请求和数据网格两端来决定整个资源分配的架构 t_{ij}.

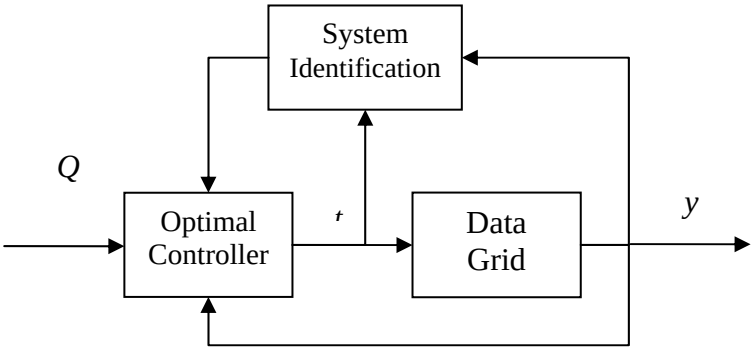


图2.2 在线系统识别的适应性控制器

为了对每一个请求都保持一定的 QoS 比率，这个系统必须计算出资源分配控制序列和 QoS 比率之间的关系。这个可以用一个线性的、自回归的多输入多输出模型来表示，模型的参数通过在线的系统辨识来获取。实际上使用的最有控制器产生最优化的资源分配的控制序列，基于上述估计模型和预定的目标函数。下面会详细描述这一过程。

2.3 在线系统辨识

这一系统是由 M 个用户请求和 N 个节点组成的，并且采用了线性自回归的多输入多输出模型。这一多输入多输出模型使得我们可以获取数据节点和不同的应用之间的关系。比如，减少资源分配给某一个 QP 任务会使增加其他 QP 任务在同一个节点上的资源分配，同时也可能同一个 QP 任务的下一个节点的工作负担。这种依赖关系是通过单输入单输出系统所不能获得的。多输入多输出系统辨识使得系统可以在不同的请求任务中获取一个平衡，当系统总体需求超过了系统的承

受能力时。为了简单化这一模型，这个系统被使用了 ARMAX 模型，拥有多个输入和多个输出变量，M=2。

$$A(q)y(k) = B(q)t(k-1) + C(q)e(k)$$

$$A(q) = 1 - A_1 q^{-1} - \cdots - A_n q^{-n}$$

$$B(q) = B_0 q^{-1} + \cdots + B_{n-1} q^{-n}$$

$$C(q) = 1 + C_1 q^{-1} + \cdots C_n q^{-n}$$

上述参数的值可能会随着系统条件和工作任务的改变而改变。在进行实际操作之前是很难确定这些变量的，所以自学习的在线系统辨识系统是很有必要的。当新的数据进入之后，系统即可根据这一新的信息进行重新估计系统模型和更新系统参数。在这个工作中，我们使用了 Matlab 自身提供的系统辨识工具。通常来说，计算机系统的阶数都比较低[34]，同时也可以事先离线决定相关参数。

为了方便后期计算，我们将上述模型重写为

$$y(k+1) = X\phi(k) + e(k+1)$$

其中

$$X = [B_0 \quad \cdots \quad B_{n-1} \quad A_1 \quad \cdots \quad A_n]$$

$$\phi(k) = [t^T(k) \quad \cdots \quad t^T(k-n+1) \quad y^T(k) \quad \cdots \quad y^T(k-n+1)]^T$$

为了方便计算，我们定义

$$\tilde{\phi}(k) = [0 \quad t^T(k-1) \quad \cdots \quad t^T(k-n+1) \quad y^T(k) \quad \cdots \quad y^T(k-n+1)]^T$$

我们使用了 ARMAX 模型和它所对应的估计器来估计相关参数 X，X 是由矩阵 A 和 B 组成的。

2.4 线性二次型最优控制器

适应性控制器的最终目标是让输出 y(k)跟踪输入所定义的 Q(k)尽可能的接近。注意到来自用户请求的 QoS 水平可能会随时间而变化，目标函数惩罚了大的变化量，使得控制序列变化量不会过于激烈。由此我们写出最优控制器的目标函数：

$$J = E[\|W(y(k+1) - Q(k)\|^2 + \|P(t(k) - t(k-1))\|^2]$$

为使 J 最小，下面的微分应该是等于 0.

$$\frac{\partial J}{\partial t(k)} = 0$$

我们可以得到控制序列应该是如下。注意到 X(k)和 B0 都是通过上述 ARMAX 模型估计器的来的系统辨识的结果。

$$t^*(k) = ((WB_0)^T WB_0 + P^T P)^{-1}\left((WB_0)^T W(Q(k) - X(k)\bar{\Phi}(k)\right) + P^T P\, t(k-1))$$

2.5 仿真结果

为了初步描述该综合算法的具体效果，我们搭建了 Matlab 的测试平台。

2.5.1 仿真环境

我们建立了一个基于 Matlab 平台的仿真环境，之所以采用 Matlab 平台是因为它本身提供了充足的数学函数及方便的数学运算。在本次仿真运行中，M=2，N=2，控制系统是一个双输入双输出系统。控制变量为$t(k) = [t_1(k) \quad t_2(k)]^T$，每一个都代表针对该应用的资源分配。在我们的模型中，这两个节点都具有相同的传递函数参数，分配给它们的资源上限也是相同的，这一点跟后来的实际运行环境也是相似的，因为虚拟机可以很方便地在不同物理机中进行迁移，迁移后的虚拟机跟原有的虚拟机各项性能是一致的。我们使用了一个 ARMAX 模型，通过多次的误差分析我们采用的具体参数为[n_a n_b n_c]=[2 4 1] 。

2.5.2 仿真结果

仿真效果说明了该算法具有较好的性能，能够跟踪预定值，并且信能够根据 W 和 P 参数的选择能够获得更好的效果。在每一个实验中，关于预定值，我们都令 Q(k) 在 t=0 时等于 10，在 t=100 事等于 15，而在 t=200 时等于 5，这是为了更好地看出跟踪的效果来。输出 y(k)基本可以在 10 之内跟踪上 Q(k)，这是在系统允许的范围内。另外，这个实验也说明了输入和目标输出是相关的，反映了一个事实随着资源分配得更多 QoS 会得到改善。因为我们给两个节点都使用了相同的传输函数，控制输入变量也是类似的。

图 2.3 说明了 W=1 和 P=1 时的输出跟踪预定值的效果。可以看到，虽然存在一定的毛刺，但是基本跟踪性能还是较好的。

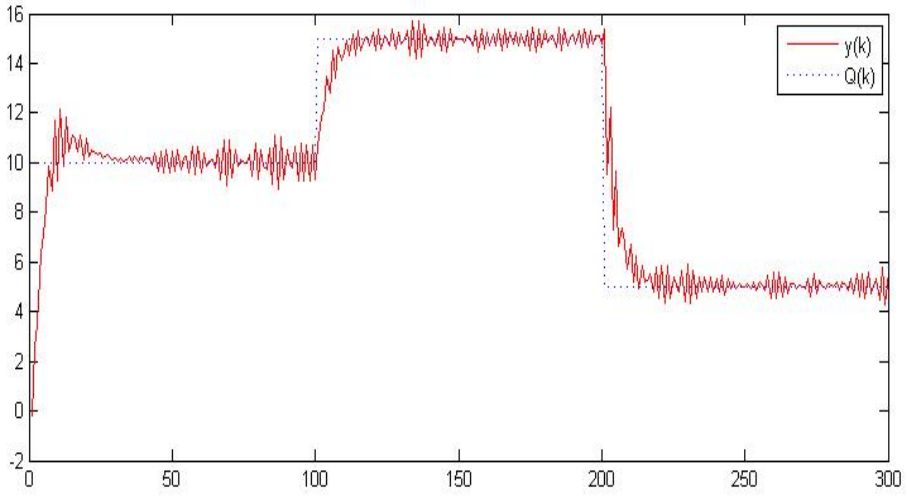


图2.3 W=1和P=1时输出和预定值

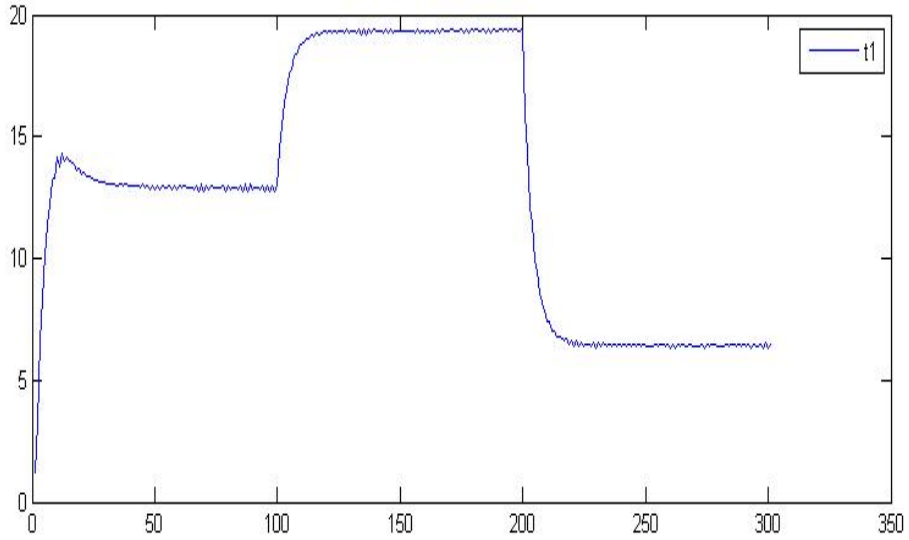


图2.4 W=1和P=1时的控制变量1

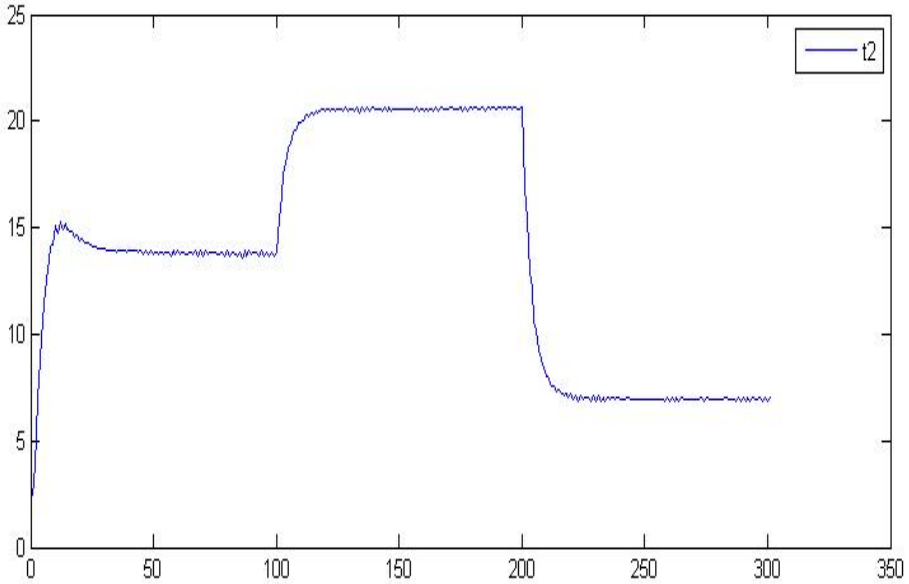


图2.5 W=1和P=1时的控制变量2

为了进一步展示控制策略的性能，我们选择了不同的 W 和 P 参数，来观察系统由此的不同表现。图 2.3~图 2.5 表现了在 W=1 和 P=1 时的输出跟踪效果，图

2.6~图 2.8 表现了在 W=0.4 和 P=1 时的跟踪效果，可以看到随着 W 的减小和 P 的增加，输出和输入曲线都趋于平缓，因为 P 实际上反映的是输入连续性的权值，而 W 表示的是输出误差的权值，P 越大，输入曲线越平缓输出曲线也越平缓，W 越大输出曲线的误差越小，但是曲线的变化较为剧烈表现出来就是毛刺更多。

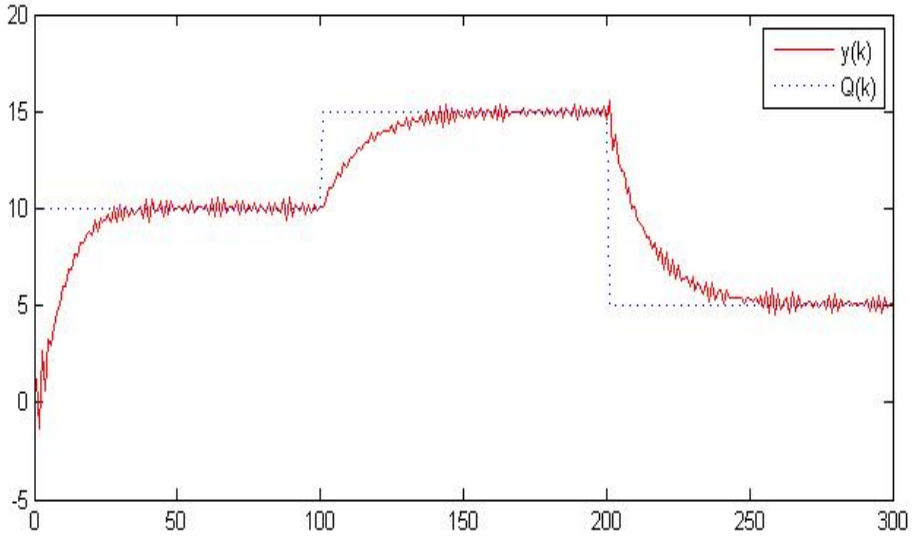


图2.6 W=0.4和 P=1时输出和预定值

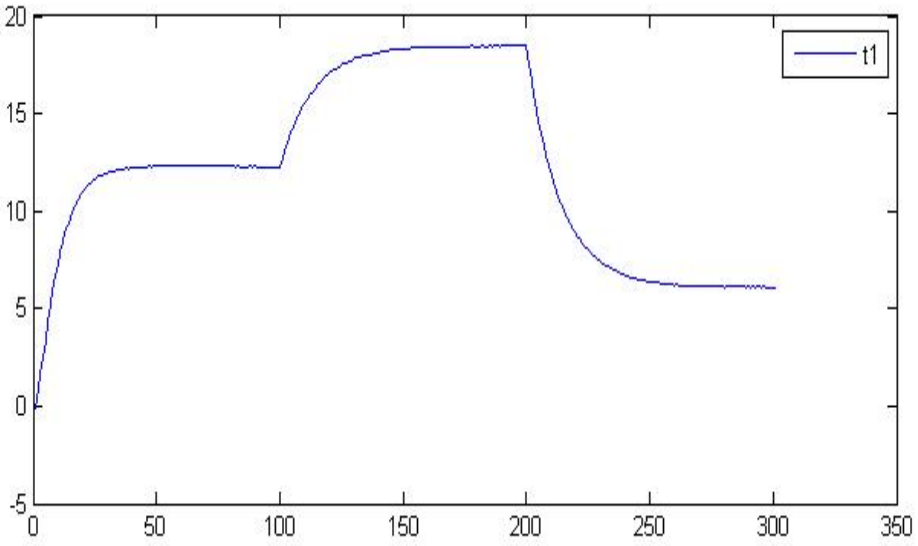


图2.7 W=0.4和P=1时控制变量1

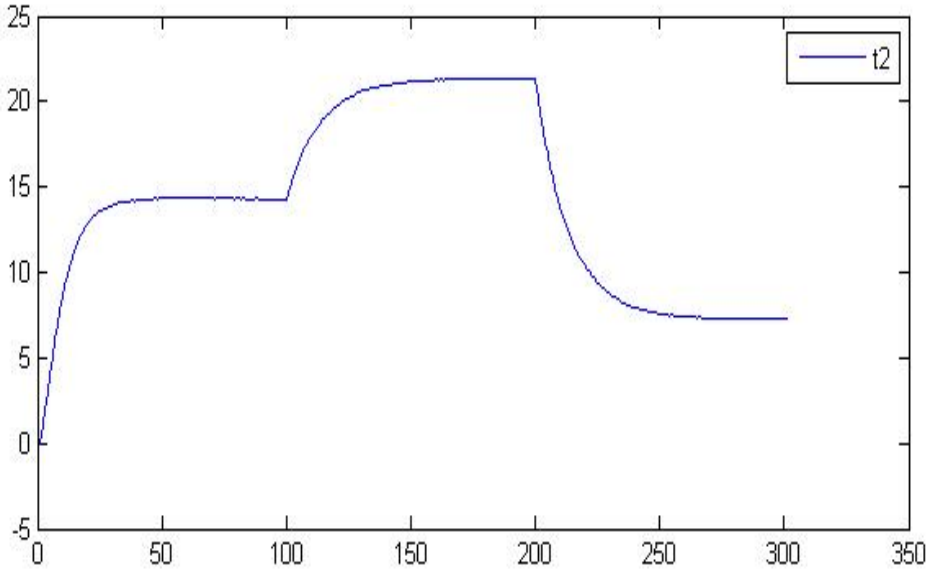


图2.8 W=0.4和P=1时控制变量2

上图描述的是 W=0.4 和 P=1 时控制变量 2 随时间变化的情况。因为使用的传递参数是类似的，所以控制变量 2 和控制变量 1 的变化情况相似。但是在实际情况中，这两者可能是有很大不同的，比如控制变量 1 可能是内存，取值范围从 128-512MB，控制变量可能是 CPU 周期，取值范围从 1%-100%。 不过具体关系如何都必须根据实际情况而定。

正如图 2.9~图 2.11 所描述的，因为 W 是参考输出量和当前输出量的误差的权值，所以该图曲线更为陡峭也具有更多的毛刺，表现出来就是变化更为剧烈。在实际系统中，为了得到快速响应特性，往往会导致不稳定性，这一点也在后面的实际实验中得到验证。为了同时获得稳定性和快速响应，W 和 P 矩阵的选择需要被设定在一定范围之内。通过自动控制理论的相关分析也可以得到类似的结论。

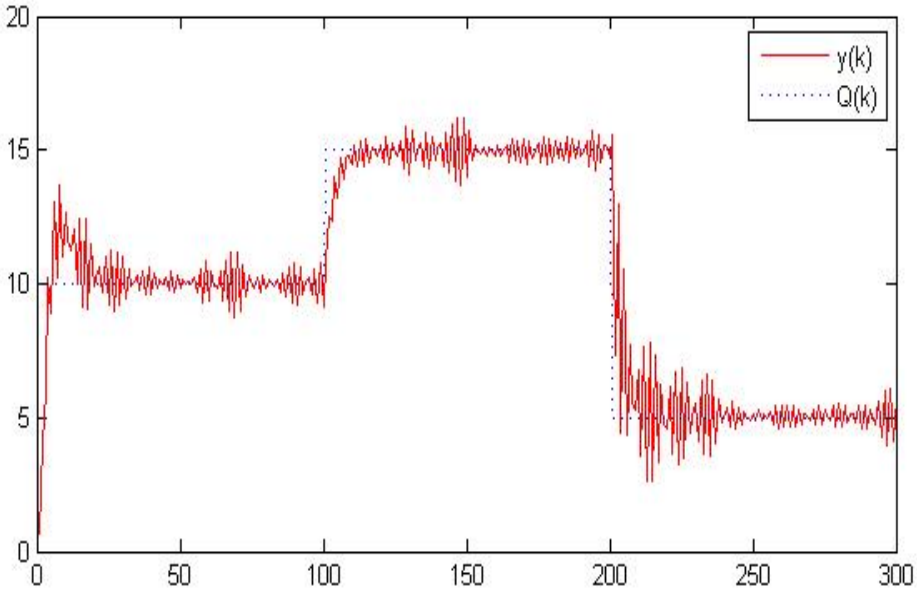


图2.9 W=1.4和P=1时输出和预定值

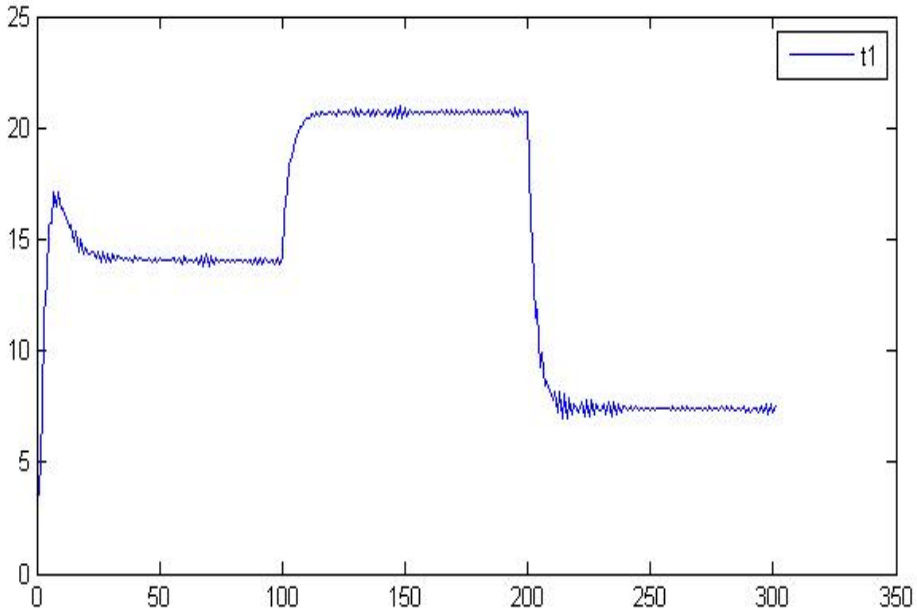


图2.10 W=1.4和P=1时控制变量1

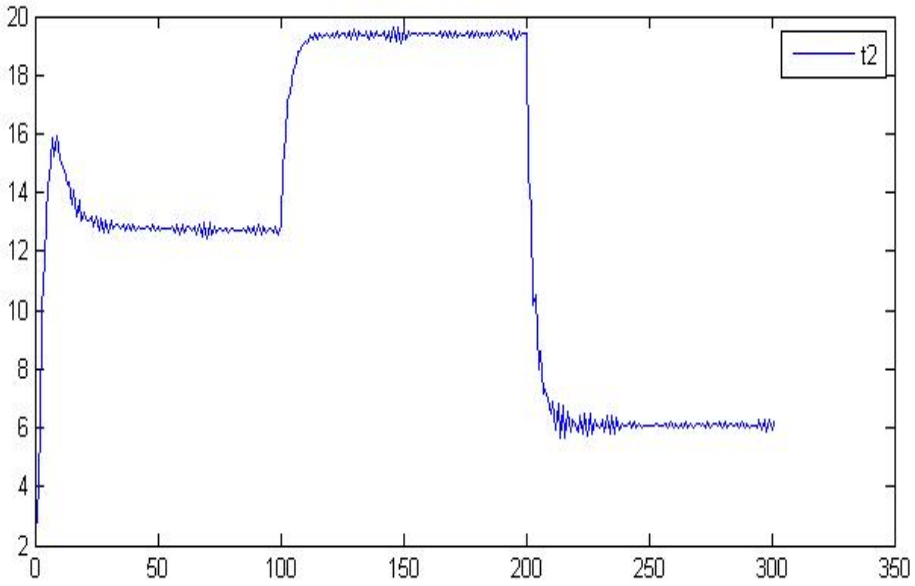


图2.11 W=1.4和P=1时的控制变量2

2.6 本章结论

在这篇文章中，我们针对数据网格中的并发处理，根据用户的需求提供更好的服务质量（ QoS ）。尽管资源和环境都是不断变化，算法仍然表现出了较好的随需求而变和细粒度调节的特性。

为了解决这个问题，我们提出了一个自适应控制器的动态调整资源以满足多个数据处理的要求，以便满足特定的不同的服务水平分布。控制器参数在运行时根据预先确定的成本函数和在线学习的方法来不断调整。Matlab 仿真测试评估了我们的控制器设计。仿真结果表明，我们的控制器能够满足 QoS 分布目标和适应动态系统资源的多种数据处理的要求，特别是当总需求的用户和应用所需超过系统的能力时。

第3章 开发 ViGrid 平台

在上一章中，理论分析和系统仿真验证了该算法的有效性。为了进一步的考察该算法在实际系统环境的表现，我开发了 ViGrid 虚拟网格系统(Virtualization-based Integrated Grid)，并且在其中进行了 Rmon 和 Montage 等网格应用的测试，本章将着重介绍这一 ViGrid 虚拟网络平台及其所依赖的虚拟技术。

3.1 ViGrid 平台所依赖的虚拟化技术

3.1.1 Xen 虚拟机

我开发的平台是基于现有的 Xen 虚拟机系统进行开发的，Xen 与 Linux 系统具有很好的兼容性，同时也支持对多种计算资源的动态调整。

Xen 是一个基于开源软件组织的虚拟机监控器（ 即 Virtual Machine Monitor 简称 VMM），可以允许在单一的物理机器上同时运行多个操作系统实例。最初基于 32 位 X86 体系结构而设计开发，支持同时运行多至约 100 个虚拟机。XEN 引入的管理接口（Hypercalls）和事件（Events）机制，以及预先定义的虚拟机和 VMM 之间的共享内存数据交换机制都使得新的客户机体系架构（XEN 虚拟机架构）具有更高的总体性能，但同时也就注定了它必须修改客户机操作系统源代码。

Xen 将客户机称之为虚拟域（Domain），其中 0 号虚拟域为服务域作为监控程序的扩展提供系统的管理服务。监控程序拥有部分硬件 IO 资源如定时器设备、中断设备 PIC/Local APIC/IO APIC 等，其他虚拟域也可以拥有部分的 IO 资源如硬盘网卡等。拥有物理设备的虚拟域称为隔离设备驱动域（Isolated Driver Domain）或简称设备驱动域（Driver Domain）。普通虚拟域只有虚拟设备而不拥有直接的硬件设备资源访问权。XEN 项目也将 Hypervisor 称为 XEN。

XEN 本身主要基于开源的 Linux 内核代码移植而来，同时运行其上的 XenLinux 也从 Linux 移植而来，意为支持 XEN 架构的 Linux。同样支持 XEN 架构的 free BSD 和 WindowsXP 也能够在 XEN 上运行。应用程序均（X86）不需任何修改就可以在 XEN（X86）上运行，如 Linux 应用程序可以在 XenLinux 上运行而 WindowsXP 应用程序可以在 XenXP 上运行。而我们的具体应用环境就是 CentOS。Xen 能够为 CentOS 提供很好的支持和兼容性。

3.1.2 Xen 准虚拟化体系结构

Xen 监管程序（hypervisor）运行在在最高优先级（Ring 0）上，准虚拟化的客户域运行在较低的优先级上（Ring 1-3）。Xen/X86 准虚拟化域的内核运行在优先级 1 上，而应用程序仍然运行在优先级 3 上。Xen 服务虚拟域拥有对整个（或部分）物理系统资源的管理功能如块设备，显示和输入输出等。

Xen 支持准虚拟化域和完全虚拟化域的同时运行，如图 3.1 Xen 虚拟化体系结构所示。虚拟域 0（即服务虚拟域）作为 hypervisor 的扩充，直接拥有系统硬件输入输出设备（IO，DMA 等），也因此必须具有对这些设备响应的原生（Native）驱动程序。虚拟域 0 提供对整个系统的管理平台，也是一个设备驱动域。虚拟域 1 是一个设备驱动域拥有部分物理设备，因此也需要 Native 驱动程序以及向其它客户机提供后端设备驱动程序。虚拟域 2 是一个用户准虚拟化域，它不具有物理硬件设备，相反通过虚拟设备（即前端设备驱动程序）向位于虚拟域 0 的后端设备驱动程序申请服务而实现对设备的访问。另外未经修改的操作系统上也可以运行准虚拟化的前端设备驱动程序以获取更高的性能。

这一 ViGrid 系统利用的就是 Xen 的准虚拟化技术，因为 Xen 的准虚拟化技术能够提供细粒度的资源调度，包括内存和 CPU，CPU 的管理可以更为精细。只有满足了能够对这两者进行细粒度的调度，我们才能进行进一步的系统开发并且得出合适的调度算法。Xen 的准虚拟化技术也支持多台虚拟机同时运行且动态调度，这一点虽然在试验阶段用处不大，但是在进一步的开发过程中，如果虚拟机的数量剧增，Xen 的这一点优势是具有重大作用的。在实验中发现，虚拟机如果配置过头，比如内存过小，就容易导致虚拟机操作系统崩溃；如果虚拟机内存配置过小，又可能影响物理机的内存使用，导致操作困难，比如文件读写等。所以所配置的资源必须在一定范围内，我们常用的内存配置范围是 128-256MB，更为宽松一点的配置范围是 128-512MB，但是存在一定的系统崩溃危险。CPU 的资源配置范围相对灵活，因为现代计算机的 CPU 往往都是非常高速的，能够满足一般的数据流管理的需求。这一点也对于我们进一步开发系统提出了建议，我们需要内存更大，但是 CPU 没有必要特别快的服务器，以实现更好的经济性能。

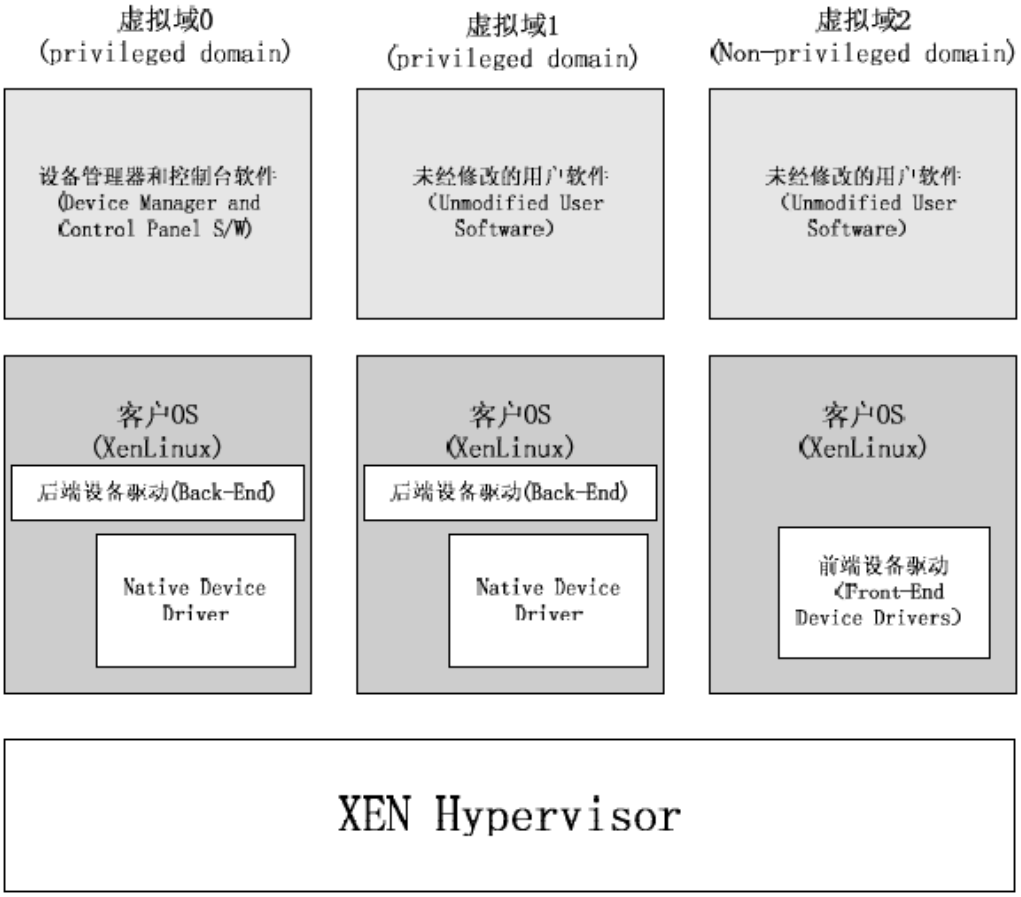


图3.1 Xen虚拟化体系结构

3.1.3 内存虚拟化

每一个 X86 客户机的内存地址总是从 0 开始的，而位于该地址的物理内存只有一块。因此监控程序必须把客户机从 0 开始的内存地址映像到其他物理内存页，也因此监控程序必须把客户机虚拟地址（ guest virtual address）到客户机物理地址（guest physical address）的映像（客户机页表）进行重新映像，即建立客户机虚拟地址到机器物理地址（ machine physical address 或 physical address）的映像。Xen 准虚拟化实现采用修改客户机页表的方式实现这一重新映像，硬件虚拟机采用影像页表（shadow page table）方式实现。

Xen 准虚拟化的 hypervisor 和客户机共享同一个地址空间。32 位 X86（非 PAE 模式）架构下的 hypervisor 占有 4G 空间的高 64M 地址空间即从

0XFC000000 到 0XFFFFFFFF，这通过重定向准虚拟化客户机的 GDT 表实现。Hypervisor 为每一个虚拟域准备一个内部 GDT 表以容纳客户机 GDT，但同时为 Xen 保留一部分描述符空间用于 1）hypervisor 将自身的代码和数据分别映像到 ring0，1 和 3；2）映像每一个 CPU 的 TSS；3）映像每一个 CPU 的 LDT。因此准虚拟化的 Xen 客户机只能使用 0 到 4G-64M 的地址空间，这并不会影响 Linux 应用程序的正常执行（应用程序只使用 0-3G 地址空间）。Xen 虚拟域的初始内存大小在该虚拟域启动时确定，各虚拟域以分区（partition）形式共享整个系统的物理内存。XenLinux 实现了气球（ballon）驱动程序，来调节各虚拟域之间的物理内存共享。当一个虚拟域 A 需要更多的内存时，它可以通过气球驱动程序向 Hypervisor 提交内存请求，Hypervisor 可以在未分配的内存中或向其它虚拟域 B 通过气球驱动程序回收内存而提供该 A。虚拟域 A 的气球驱动程序在得到 Hypervisor 新提供的内存后向操作系统释放。

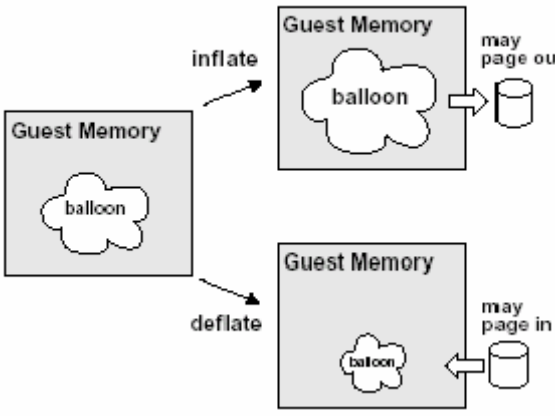


图3.2 Ballon 驱动程序

启动物理服务器时，所有的内存资源都会默认被分配给 Domain0。然后，其它虚拟机启动时，会从 Domain0 获取内存资源。如果虚拟机是在完全虚拟化模式下运行，hypervisor 将无法与虚拟内核对话，当前的内存分配也将无法改变。不过，如果是在准虚拟化模式（Para-virtualization）下，Xen 的 hypervisor 就可以动态更变内存分配。采用准虚拟化模式时，一定要确保 Domain0 至少可以分得一定的内存资源，以免它内存不足。对于 Domain0 的内存分配最小值，建议设置为 512MB。

要为 Domain0 预留内存，可以为内核添加一个启动项：dom0_mem= 。例如，dom0_mem=512M。打开 Grub 配置文件进行此设置。在 Grub 配置文件中，会看到启动 Xen 内核的启动项。它大体如下：

```
title XEN
root (hd0,0)
kernel /xen.gz
module /vmlinuz-2.6.16 .46-0.14-xen
root=/dev/system/root
vga=0x314
resume=/dev/system/swap splash=silent showopts
module /initrd-2.6.16 .46-0.14.xen
```

在此配置文件中的第一个“module”行后面添加 dom0_mem 启动项。添加之后应该是这样：

```
title XEN
root (hd0,0)
kernel /xen.gz
module /vmlinuz-2.6.16 .46-0.14-xen
root=/dev/system/root
vga=0x314
resume=/dev/system/swap
splash=silent showopts
dom0_mem=512M
module /initrd-2.6.16 .46-0.14.xen
```

设置好 Domain0 的内存分配后，就可以管理你的虚拟机内存分配了。启动一个虚拟机时，通常它会从 Domain0 获取内存资源。内存一旦分配给虚拟机，Domain0 将无法再收回，即使所有虚拟机都被停止也不能收回。正是因为这个原因，所以为 Domain0 设置内存最小值非常重要。

要想更改虚拟机的内存分配，可以利用两个 xm 命令：

xm mem-set：此命令可以更改一台虚拟机的当前内存分配；

xm mem-max：此命令可以限定一台虚拟机的内存使用最大值。不过，更改最大值之后需要重启才能生效。

更改内存分配之后，可以使用 xm list 命令检查设置是否生效和正确：

3.1.4 CPU 管理

与内存一样，我们也可以通过 Xen 提供的接口管理虚拟机的 CPU 分配。如果虚拟机使用的是准虚拟化，CPU 的分配也可以动态更改。为虚拟机分配 CPU 时，不一定要根据服务器中的物理 CPU 数目来分。如果你愿意，是可以这么做。不过，这样做是绝对优化不了性能的。如果将虚拟机与指定的物理 CPU 绑定，会帮助你大大地提高虚拟机性能。除此之外，还可以调整 CPU 的运行队列（run queue），使某台虚拟机在 CPU 中具有更高的优先级。

所有可运行的虚拟 CPU（VCPU）都是由物理 CPU 中的本地运行队列管理的。这个队列是按优先级进行排序的，队列中的每个 VCPU 平分 CPU 资源。VCPU 的优先级状态有两种值：over 和 under。Over 表示它占用的 CPU 资源超过了资源平分值，under 表示低于这个平分值。如果 VCPU 的当前状态为 under，调度程序下次则会优先服务该 VCPU。如果调度程序发现在其 CPU 上没有虚拟机为 under 状态，则会看其它 CPU 中是否有 VCPU 状态为 under，如果发现，则立即服务该 VCPU。通过这种方式，所有 CPU 都会平均分配 CPU 资源。

通过设置 weight 和 cap 参数值，管理员可以管理 CPU 的优先级。Weight 参数用于分配 CPU cycle，是一个相对值。一个 weight 为 128 的 VCPU 比一个 weight 为 64 的 VCPU 获得的 CPU cycle 多一倍。因此，利用这个参数可以决定哪个 VCPU 获得更多，哪个获得更少。第二个设置 CPU 的参数是 cap，它设置的是 domain 获得的 CPU cycle 百分数，是一个绝对值。如果设置为 100，就表示那个 VCPU 会 100%地占用物理 CPU 的可用 cycle。如果 cap 为 50，则表示该 VCPU 占用的 CPU cycle 绝不会超过总量的一半。

在如下命令示例中， id 为 3 的虚拟机 weight 为 128，允许使用两个物理 CPU 的所有 CPU cycle：

xm sched-credit -d 3 -w 128 -c 200

对于虚拟 CPU，还要做的一个重要工作就是 CPU 分配。默认情况下，虚拟 CPU 与物理 CPU 是没有固定联系的。要提高性能，就需要为它们建立一个这样的联系，这个工作很简单易行。为虚拟 CPU 和物理 CPU 建立“联系”的主要好处是可以防止虚拟 CPU 到处游荡。如果没有“联系”，调度程序会为虚拟 CPU 选择一个物理 CPU。当某个物理 CPU 处于繁忙状态时，虚拟 CPU 就会被转移，由另

一个物理 CPU 服务。这个工作对性能的影响是很大的。因此，将虚拟 CPU 与物理 CPU 绑定是个不错的办法。

最后，我们还可以更改虚拟机分配的 CPU 数量。要更改此设置，既可以利用虚拟机管理器（Virtual Machine Manager）进行，也可以使用 xm vcpu-set 命令。例如，将 domain 1 分配的 VCPU 数改为 4 个，则：

xm vcpu-set 1 4

使用该命令时，有时会发现它有时不起作用。这是因为，虚拟机的操作系统还必须支持动态更改 CPU 数量，不然就不能这样更改了。所以，在虚拟机的配置文件中更改其 VCPU 数更有效，而且不会因为重启虚拟机而失效。

对于虚拟机的性能优化，内存与 CPU 设置很重要，本章节已阐述了其原因。此外，本章节还介绍了如何调整虚拟机在物理 CPU 中的优先级。

3.2 ViGrid 系统

为了实现上一章中描述的对 CPU、内存和带宽等资源的动态调整，并且融合适应性控制技术和系统辨识技术，我们开发了 ViGrid 虚拟网格平台。

3.2.1 ViGrid 系统功能

这一平台实现的主要功能是：

1、创建、管理、迁移多台物理机内的虚拟机。每一个虚拟机就是一个节点，拥有可以动态调配的计算资源并且处于一定的网络拓扑结构中。网络拓扑结构不能动态改变。

2、处理不同的任务请求。因为每一台虚拟机都拥有一个操作系统，根据响应的软件服务能够处理各种请求，包括 Rmon 和 Montage 应用。

3、根据不同的 QoS 和控制策略为不同的任务和不同的节点得出控制变量。控制策略采用第二章所描述的最优控制和系统辨识的方法。

4、根据控制变量动态调整每一台虚拟机的计算资源。通过可靠的网络通讯来完成这一过程。

5、存储记录整个过程。防止系统崩溃。

基于此，我们设计了这一 ViGrid 系统。

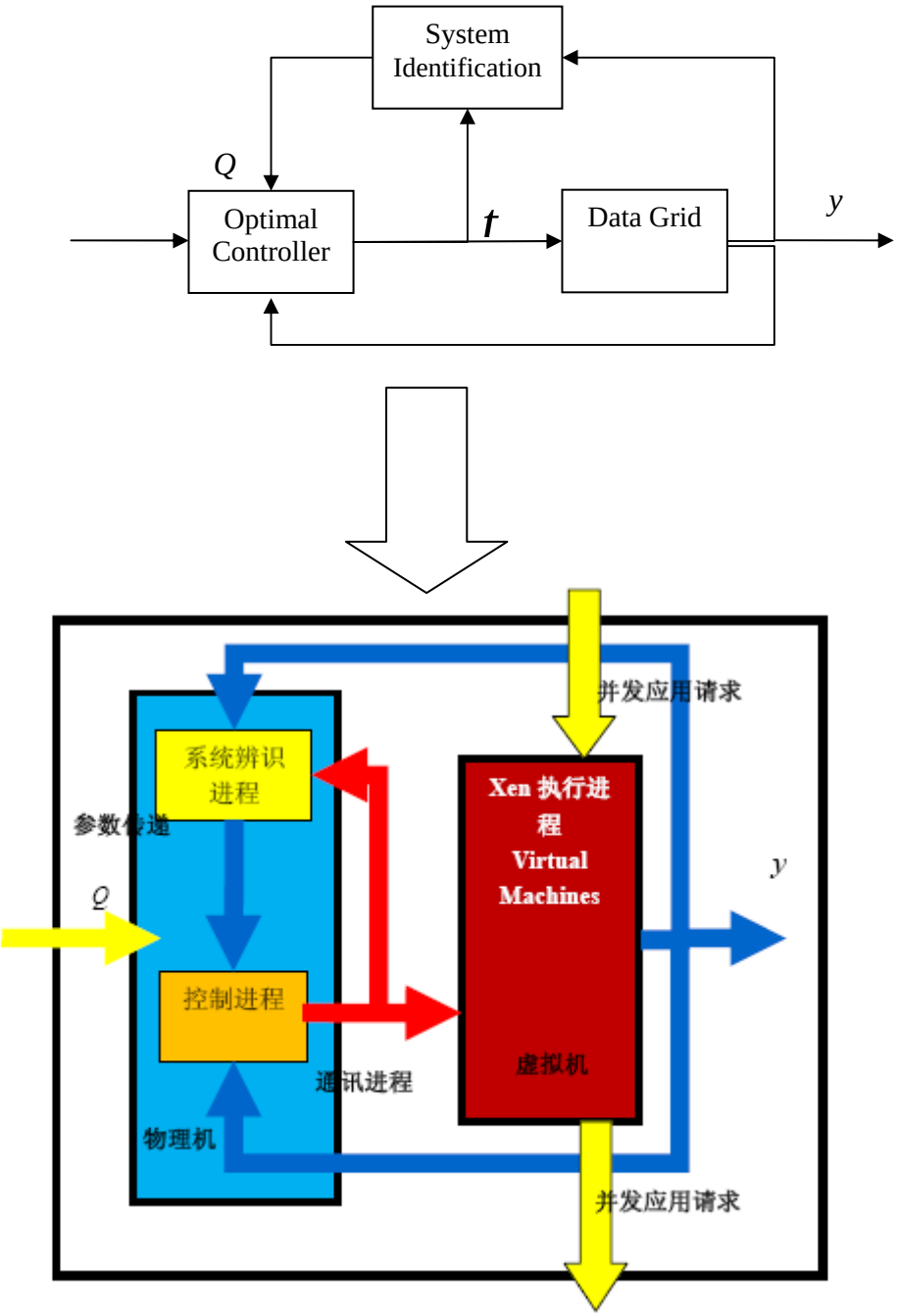


图3.3 ViGrid 虚拟网格架构

ViGrid 是有多台安装了 XEN 虚拟机的物理机组成的，每一台物理机和每一台虚拟机组成了局域网。虚拟机与物理机的网络连接方式采取桥接式，这样在网络层次上物理机和虚拟机是平等的。

在 ViGrid 虚拟网格中运行的各个实时的进程包括：

1、控制进程：根据系统辨识得到的系统参数和记录的 VM 系统的输出输入参数得出下一时刻的控制变量并且通过 Xen 提供的接口调整虚拟机资源分配。

对于 Xen 平台构筑的虚拟网络平台来说，因为物理机才能够调整虚拟机上的 CPU 和内存分配，所以控制进程是运行在物理机上，同时为了方便各个物理机之间的参数调用过程，我们也增加了辅助的命令传输进程。

3、指令通信进程：通信进程是在每一台虚拟机和物理机上都会运行的，其主要作用是可靠地保证指令被指定节点执行。

4、数据传输 进程：文件传输是为了保证各个节点之间的可靠数据传输。

5、系统辨识进程：辨识进程不断监控 VM 系统的输入输出状况，并且根据相关算法计算出实时的系统参数。

6、执行进程：运行在每一台虚拟机之中。虽然物理机和虚拟机在逻辑上和网络层次上是平等的，但是为了系统划分清楚及管理方便，物理机不处理数据而仅仅是控制和调节虚拟机的系统参数。执行进程就是运行在虚拟机中的每个管道中的任务。这些任务各不相同，通过预置指令集来配合不同的任务处理。预置指令集会在下一章详细介绍。

同时辅助运行的还包括记录进程，它的作用是类似“黑匣子”的功能。因为系统在运行大量的、可变的服务，而可能对虚拟机系统甚至物理机系统造成很大负担乃至引起系统崩溃等。记录进程就是为了防备数据丢失而不断地为运行数据进行备份的一个进程。

下面就是对应这几个进程的简单介绍：

在控制节点（一般为物理机系统，也可以是虚拟机本身）中：

```
pthread_t th_listen,th_process,th_control;
pthread_create(&th_listen,NULL,listen,NULL);
pthread_create(&th_process,NULL,process,NULL);
pthread_create(&th_control,NULL,control,NULL);
pthread_join(th_listen,NULL);
pthread_join(th_process,NULL);
pthread_join(th_control,NULL);
```

void* listen(void *data) 对应的进程是 th_listen，它监听某一端口，如果有数据流发起处理请求就将其传递给 process 进行处理，否则进入缓存，如果缓存溢出则数据被丢弃。

void* process(void*data) 对应的进程是 th_process，它实现的主要功能是从 command_list.txt 里边读取所要执行的任务，然后执行，并且根据系统缓存的多少进行调整。

void* control(void *data)对应的进程是 th_control，它实现的主要功能是根据获得信息进行计算，计算的核心函数是 get_u()，然后根据计算结果对计算资源进行实时控制。

double get_u() 不断地根据已有信息如当前输入输出进行计算以得出控制序列 u。

指令传输部分：

class CommandClient 指令客户端类，可以是接收指令，然后本地进行执行。其中的核心函数是 run();

void run(int listen_port)不断监听服务端口；

class CommandServer 指令服务端类，可以是发出指令，然后在指令客户端执行。其中的核心函数是 connect();

void connect(char* ServerIP,int ServerPort)发起连接；

系统也可以采取由客户端发出指令然后服务端本地执行的模式。这两种模式本质上并无不同；

文件传输部分：

class FileTransferClient 文件客户端类，发出请求传送，可以是要求文件服务端发送指令过来，也可以是本地传送过去。核心函数是 connect()；

void connect(char* ServerIP,int ServerPort)发出请求；

class FileTransferServer 文件服务端类，接收请求传送，可以是发出本地文件，也可以是接收远程文件。核心函数是 run()；

void run(int listen_port)监听端口并且处理请求；

3.2.2 虚拟机内部的网络

我们之前提过，采用 ViGrid 系统可以配置出不同网络拓扑结构的虚拟网络来。接下来的内容即详细介绍这一过程。ViGrid 系统是由多台物理机组成的，每一台物理机上运行多台虚拟机，目前是 2 台，但是可以随着物理机性能的提高而增加。物理机节点不参与数据处理，仅仅是允许控制进程及系统辨识进程以控制在其上运行的各个虚拟机系统。所以 ViGrid 的数据网络严格来说是虚拟机之间的网络拓扑结构。

在目前的系统设置中，我们使用了一种最为简单的方式。由已有的路由器为每一个节点分配 IP 地址，所有节点都处在同一个局域网内。因为受制于实验环境因素影响，我们不能给每一个节点都分配广域网 IP 地址。但是随着进一步的开发，更复杂的网络拓扑结构也是可行的，比如开放式的广域网连接与局部的局域网相结合等。其他网络拓扑结构也是可以考虑的。Xen 提供了多种网络连接方式诸如桥接、路由也包括 VPN 等。

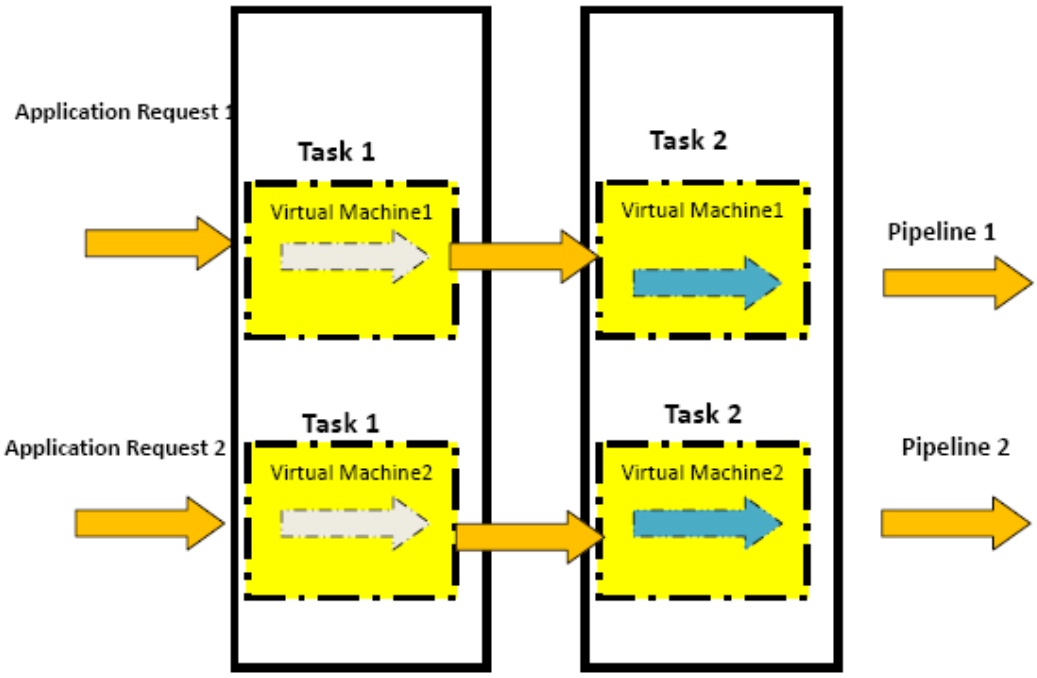


图3.4 一种网络拓扑结构下的虚拟机网络

图 3.5 是在某一台物理机内部的虚拟机结构及系统功能。

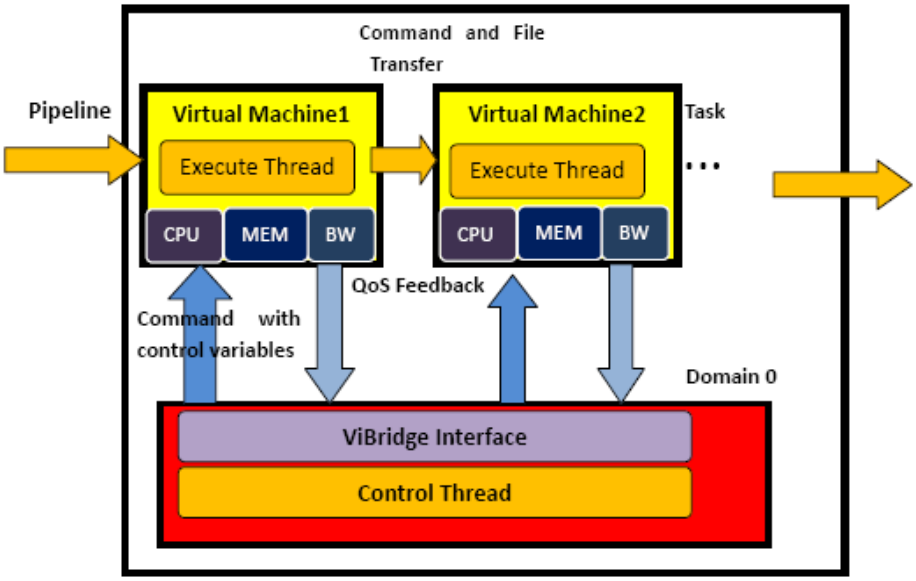


图3.5 虚拟机内部结构

在物理机内部运行的两台虚拟机的配置环境基本是一致的，因为他们都是来源自同样的一个虚拟机源文件并且通过迁移复制而得到的。但是在每一台虚拟机所运行的任务是不一样的，并且是根据实时情况不断得到修正的。在我们的实验平台中，如果需要修改虚拟机的执行任务的命令，之需要修改 command_list.txt 文件。

比如在 montage 这个实验中，command_list.txt 的内容是：

13

source /etc/prifile

mImgtbl rawdir images-rawdir.tbl

mProjExec -p rawdir images-rawdir.tbl template.hdr projdir stats.tbl

mImgtbl projdir images.tbl

mAdd -p projdir images.tbl template.hdr final/m101_uncorrected.fits

mJPEG -gray final/m101_uncorrected.fits 20% 99.98% loglog -out final/m101_uncorrected.jpg

mOverlaps images.tbl diffs.tbl

mDiffExec -p projdir diffs.tbl template.hdr diffdir

mFitExec diffs.tbl fits.tbl diffdir

mBgModel images.tbl fits.tbl corrections.tbl

mBgExec -p projdir images.tbl corrections.tbl corrdir

mAdd -p corrdir images.tbl template.hdr final/m101_mosaic.fits

mJPEG -gray final/m101_mosaic.fits 20% 99.98% loglog -out final/m101_mosaic.jpg

第一行表示其中有 13 个可执行命令需要得到执行，然后接下来的就是 13 个指令行。在执行之前需要保证 Montage 的可执行文件路径已经被添加到了系统路径中了。

其他几个配置文件包括 parameters.txt 是包含了系统辨识和控制器的一些参数文件，根据具体的辨识算法和控制器不同而变化。

比如在某次实验中，parameters.txt 内的内容是：

MAX MEMORY

256

MIN MEMORY

128

BUFFERSIZE

15

TIME SLEEP

60

这些都是控制算法或者其他进程所需要的参数。

同时存储进程不断地将系统运行所产生的输出、输入结果和其他临时参数写入 log 文件夹，形成包括 y.log，u.log，utmp.log 等几个文件来记录实验过程中的数据。

3.3 本章结论

本章主要介绍用以验证第二章所描述的算法，并且介绍了我的主要工作 ViGrid 系统和主要功能。ViGrid 是基于目前流行的 Xen 和 CentOS5 平台开发的，能够根据用于请求动态地调整计算资源，并且不断实时反馈得到更为可信的服务水平，同时实现了资源利用效率和工作效率的平衡。

第4章 网格实验及分析

构建好了这一平台之后，我们在这一 ViGrid 平台上面进行了两个主要的网格应用程序，Rmon 是在 LIGO 等项目中常使用的相关度分析程序，Montage 在天文学、地震、天气预报等方面得到很大的应用，是具有标尺验证功能的马赛克拼图程序。

实验结果较好地显示了前述算法的有效性及 ViGrid 的可行性。同时也反映了在实际环境下我们面对着更为复杂的计算环境。

4.1 Rmon 程序介绍

Rmon 是在 LIGO 数据分析中常用的一个数据流应用。来自两个观察点的数据流(H1 和 L1)不断产生数据输入请求，表现在这里就是两个文件列表。Rmon 计算连个数据集的 r 统计量来确定信号的相似性。如果某一信号在两个观测点（一个在华盛顿州，另外一个在路易斯安那州）同时出现，那么它就更有可能是引力波 burst 信号。一旦 Rmon 被运行，一个可选的文件将被用于定义数据幅度和每一步计算的频道。因为数据都是以时间序列的形式出现的，它可以成为数据流的形式且处理之后成为无用数据。这是 LIGO 数据分析的一个简化例子，因为实际运行中还需要很多前期准备和后期加工，一个完整的数据分析也是以一种管道的形式进行的。

LIGO 数据流是由小的数据文件组成的，每一个都包括了在 16 秒钟观察得出的数据。这里所用的数据文件是缩减后的 LIGO 第三层数据，只包括了来自引力波通道的数据。总共达到 4,354MB 的 1,188 组数据文件事先被存储在本地硬盘中。

Rmon 样例程序的功能如图 4.1 所示。

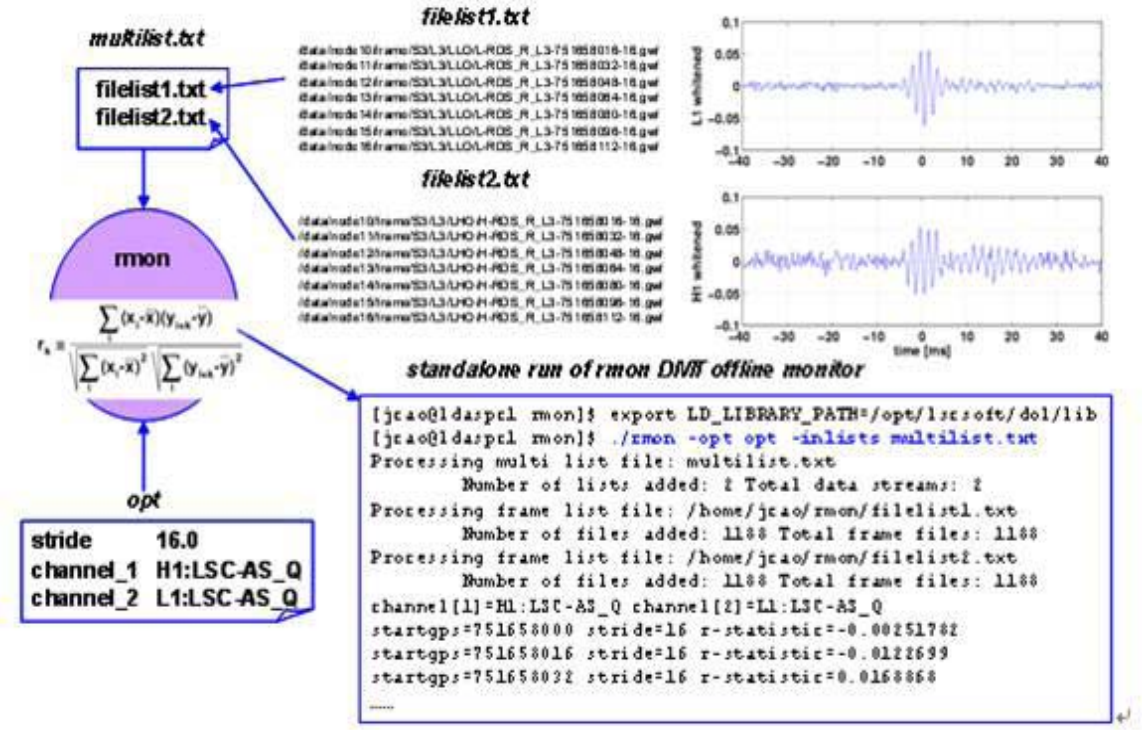


图4.1 Rmon样例程序

某一次的 Rmon 运行的效果如图 4.2 所示。其通过 Linux 命令行输出的既包括运行过程，也包括运行结果如 r-statistic 就是每一组文件的 R 统计值。Rmon 随着程序的运行不断地按照 filelist 内的文件顺序去处理、分析其内每组文件的 R 统计值并且显示。

出于减小不稳定因素考虑，我们的文件都是存储在本地文件夹内。但是通过输入控制机制实现模拟的输入流，这样子一方面可以更好地验证算法，隔离不稳定的扰动因素，另一方面也有利于系统维护，减少对外界系统的依赖性。

```
[nick1i@yushu rmon]$ source standalonerun
Processing multi list file: /home/nick1i/rmon/multilist.txt
        Number of lists added: 2 Total data streams: 2
Processing frame list file: /home/nick1i/rmon/filelist1.txt
        Number of files added: 1188 Total frame files: 1188
Processing frame list file: /home/nick1i/rmon/filelist2.txt
        Number of files added: 1188 Total frame files: 1188
channel[1]=H1:LSC-AS_Q channel[2]=L1:LSC-AS_Q
startgps=751658000 stride=16 r-statistic=-0.00251782
startgps=751658016 stride=16 r-statistic=-0.0122699
startgps=751658032 stride=16 r-statistic=0.0168868
startgps=751658048 stride=16 r-statistic=-0.028074
startgps=751658064 stride=16 r-statistic=0.0363762
startgps=751658080 stride=16 r-statistic=-0.0118638
startgps=751658096 stride=16 r-statistic=-0.00390466
startgps=751658112 stride=16 r-statistic=0.0369562
startgps=751658128 stride=16 r-statistic=-0.0270225
startgps=751658144 stride=16 r-statistic=-0.0431374
startgps=751658160 stride=16 r-statistic=-0.0423102
startgps=751658176 stride=16 r-statistic=0.0325139
startgps=751658192 stride=16 r-statistic=-0.0140542
startgps=751658208 stride=16 r-statistic=-0.0178756
startgps=751658224 stride=16 r-statistic=0.0387879
startgps=751658240 stride=16 r-statistic=-0.0406371
```

图4.2 Rmon运行效果

这是某一次运行 Rmon 时的简要程序：

```
#!/usr/bin/expect -f
foreach j {512 256 128} {
set i 5
while {$i<105} {
puts "Mem = $j, CPU=$i"
exec /usr/sbin/xm mem-set 3 $j
exec /usr/sbin/xm sched-credi -d 3 -w 128 -c $i
set password *******
spawn ssh root@166.111.137.250 /home/ligo/rmon/standalonerun
set timeout 200
expect "root@166.111.137.250's password:"
set timeout 2
send "$password\r"
set timeout 2400
```

```
expect "Time Consumed Is"
incr i 5
}
}
expect eof
```

它实际上是关于依托于现有的网格进行两层循环以获得 QoS 和计算资源的关系。

4.2 Rmon 应用表现

我们在 Rmon 应用上进行的实验主要是验证虚拟机资源分配跟 QoS 的关系，这里我们把针对 Rmon 的 QoS 指标定义为每一组数据流完成的时间，完成的时间越少，我们认为这一 QoS 指标越好。

三台虚拟机被建立了，各自的内存值分别为 512MB，256MB 和 128MB。之所分配给 CPU 的 caps 值从 5%一直到了 100%，且分度为 5%。试验中发现，内存对 QoS 指标的影响非常小，图 4.3 中也可以看到三个曲线都几乎重合在一起，但是 CPU caps 值有较大的影响。这说明不同的虚拟机系统资源针对不同的应用是有不同的影响的。后面的 Montage 应用对内存则更为敏感。

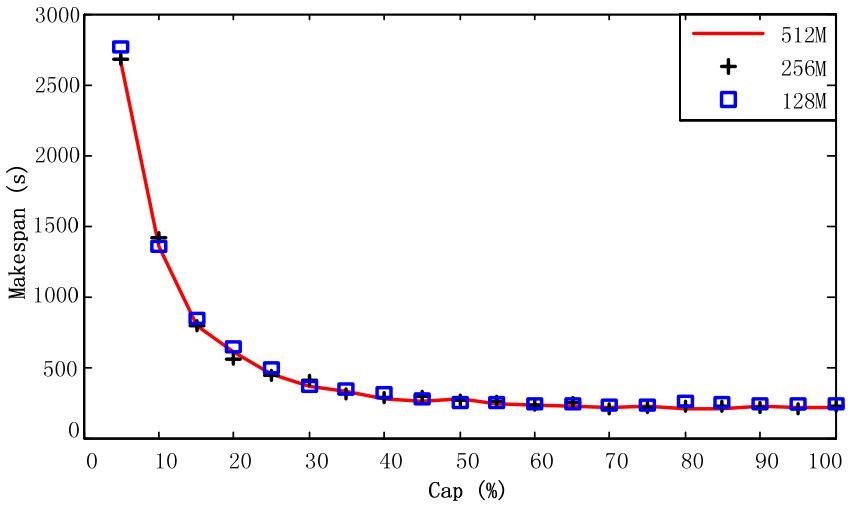


图4.3 Rmon实验结果

Rmon 之所以出现对 CPU 资源而不是内存资源敏感推测是因为 Rmon 主要是计算资源消耗多，而文件读取开销并不大，所以内存消耗不大。也有可能它所需要的内存远远小于 128MB，进一步的实验可以考虑将内存值继续降低，但是可能会危及系统的稳定性。因为试验中发现如果内存低于 100MB，操作系统可能会崩溃或者难以操作。

在图中处理速度用实线来表示。我们用多项式曲线拟合来表示 CPU cap 和处理速度的关系，并且用的是最小二乘法。我们得到一个二阶多项式方程：

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

C 是 cap 的缩写，它的曲线为图 4.4。

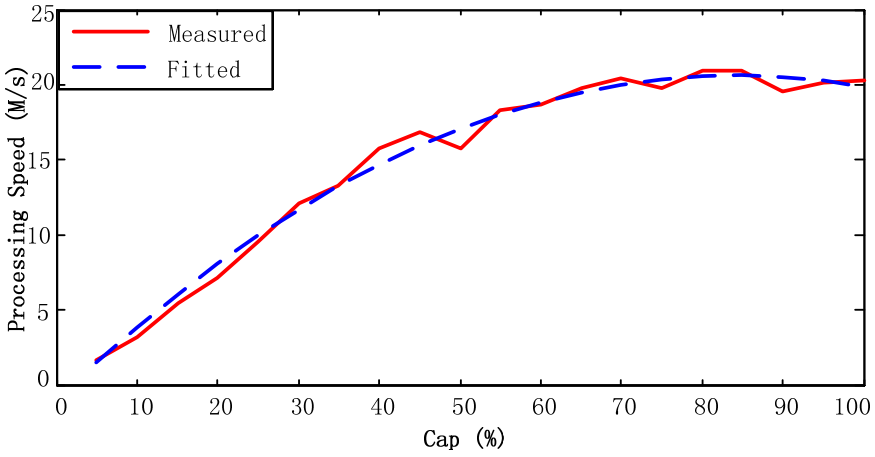


图4.4 运算速度与CPU关系

从上面两幅图中，我们可以看出，当 CPU caps 值超过 50%的时候，增长并不明显。在接下来的试验中我们可以看出，大部分情况下，CPU 配率是不需要超过 50%的，这对于我们在后期的试验中进一步调整分配的 CPU 资源是有重要作用的。

排除掉 50%以上的数据，我们重新获得了一个二阶多项式方程：

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

曲线在图 4.5 中得到表示：

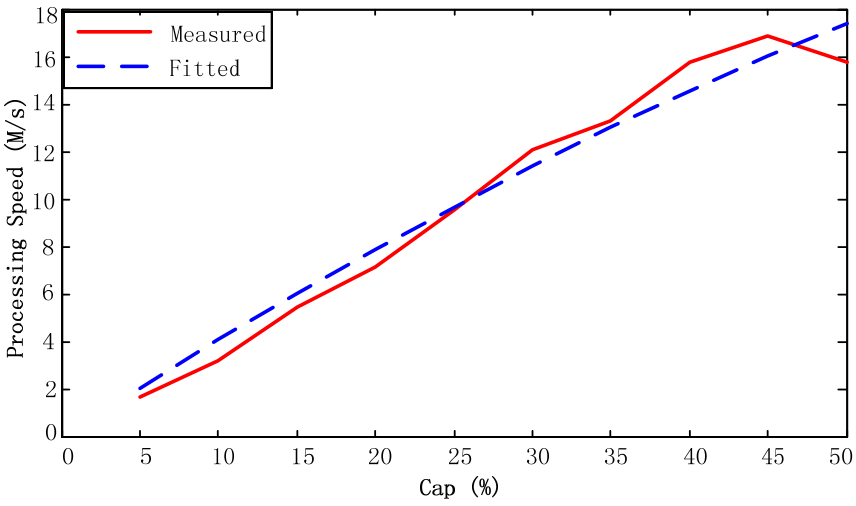


图4.5 运算速度与CPU关系(二阶多项式)

二阶参数比较小，我们可以得出一阶多项式方程：

$procpeed(c) \approx 0.3636*c, c \in [5,50]$

如图 4.6 所示。

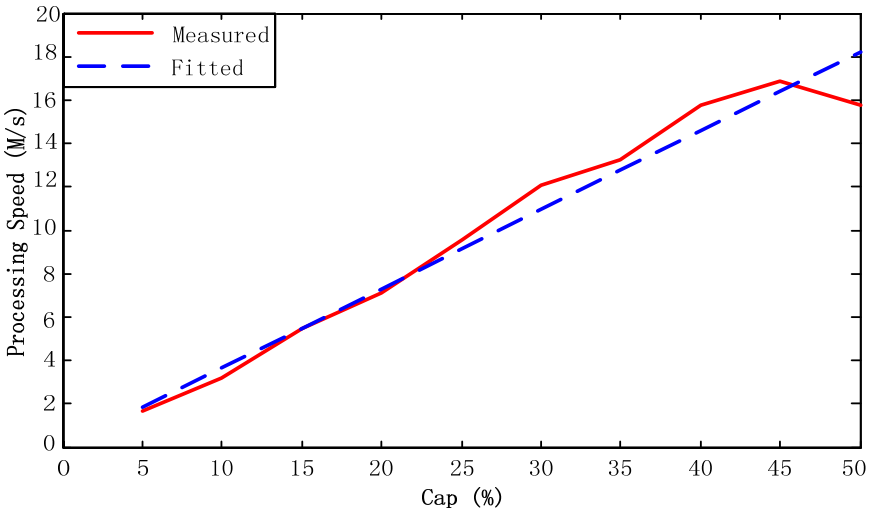


图4.6 运算速度与CPU的关系(一阶方程)

上述实验说明了我们的算法是可以对计算资源进行细粒度的调度的。

4.3 Montage 简介

为了进一步说明算法的有效性，我们在 Montage[47]应用上做了进一步的实验。Montage 在天文学、气象学、地震科学等多方面具有较多的应用。比如，天文学中在各个波段都有丰富的图像资源。在一个频率范围的图像搜集常常被割裂开来研究，而忽视了其他频率范围。这是由于不同性质的数据收集工作本身的困难：图片来自不同的坐标系，星图投影，空间取样和图像大小。像素本身很少是对应在天空上的某个点。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。

天文学及其他学科因此需要图像拼接软件，它应该能够从多个图像数据集中拼合出科学级别的马赛克，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 必须保持天体测量和光度的完整的原始数据，并从星空或者使用基于物理模型的设备来纠正背景辐射。 Montage 是一个工具包，它能够灵活地拼合 Flexible Image Transport System (FITS) [48]图像并转换为自定义的马赛克图像。它的主要特点有

高精度：能够保留空间和校准输入图像的保真度

可移植性：运行于所有常用的 Linux / Unix 平台

可扩展性：运行于台式机，集群和计算网格

可获得性：开放源代码和用户文件可供下载

通用性：支持所有的 World Coordinate System (WCS)[49]和其他常用的协作系统

高性能：能够在 128 节点的 Linux 集群中处理高达 4000 万像素、32 分钟的数据

灵活性：独立引擎来几何分析天空图像;重新映射影像;背景匹配;共同新增图片

方便性：提供管理工具，能够操纵大型图像文件

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。它将被部署的分布式 Teragrid 系统[50]，并且所有天文学家都能够获得这一应用。在其最初的应用中， Montage 将应用在 2 微米全巡天（ 2MASS ）[51]，数字帕洛马天文台巡天（ DPOSS ）[52]和斯隆数

字巡天（ SDSS ）[53]。因此，它将大大扩展天文研究的范围，并且是一个有价值的工具的任务规划和质量保证应用。它的功能包括：

多波长深空检测

预测来源点和波长

图像中每个像素光度法

基于波长的位置优化

扩展的波长结构

图像差分来检测微弱信号

发现新种类的物体

Montage 同时也是国家虚拟天文台[54]的一部分。接下来本文将描述 Montage 的结构和性能，并演示该工具包如何在天文学得到应用。

图 4.7 描述了一个简单的以拼合三张图片的 Montage 并行工作流。它可以在多处理器中方便的运行，也可以并行运行。有关图像重叠部分的最适合平面部分是可以并行计算的，只要所有图像的重映射已经完成。计算最适背景匹配参数需要所有的重合部分已经被处理完，不过针对独立图像的模型应用是可以并发执行的。

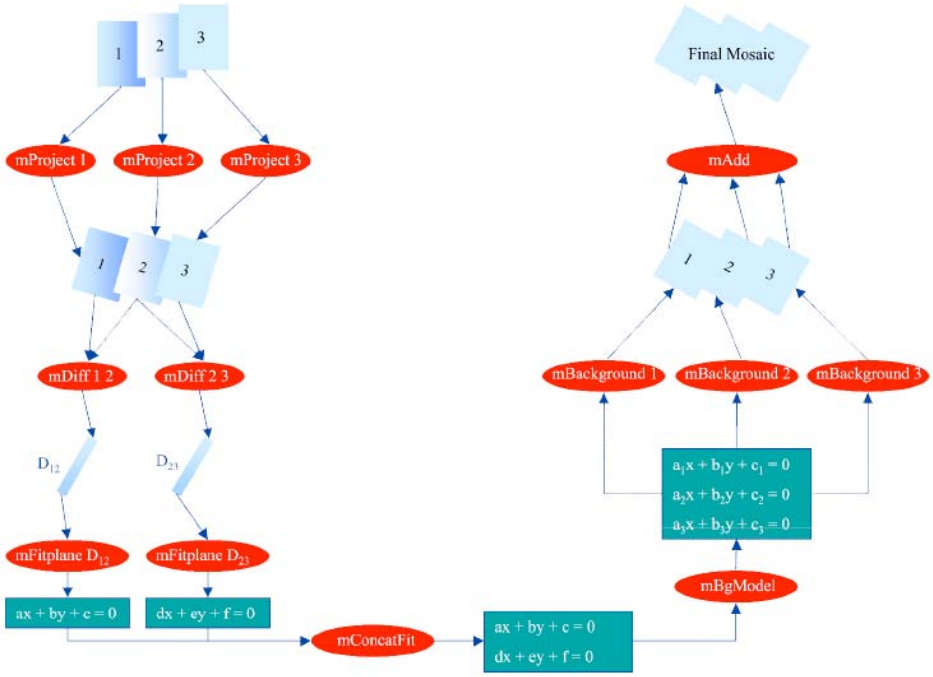


图4.7 Montage的一个实例工作流程

Montage TeraGrid Portal 是一个分布式的结构。它包括五个部分，每一个部分都包括一个客户端和一个服务端： 1. 用户结构，2. 抽象工作流服务， 3. 2MASS 图像列表服务， 4. 网格调度与执行服务，5. 用户通知服务。下图描述了这一组成。

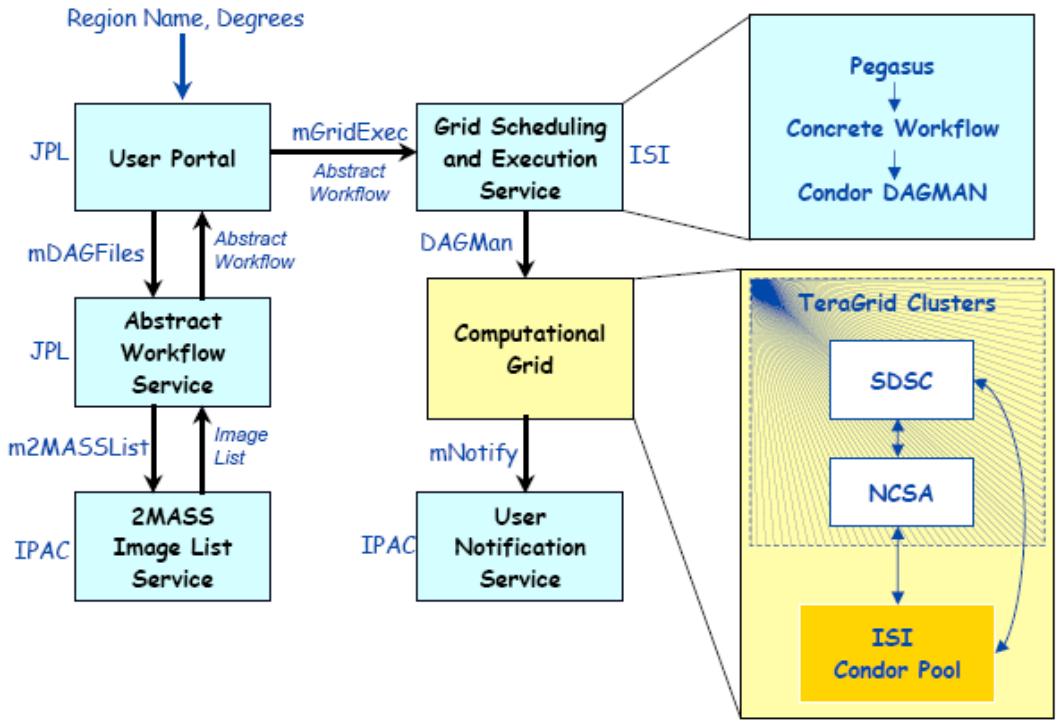


图4.8 Montage TeraGrid Portal

4.4 针对 Montage 的 M101 实验

M101 星系是位于大熊座中的风车星系。影像中的 M101 就是这样一个壮观的例子，其相对较近的距离——2,700 万光年也使得它已被研究得较为详细。图 4.9 是我们来自单一望远镜获取的的 M101 的完整图像。

图4.9 M101星系图(单一照片)

在这个例子中，我们将在 J 波段 m101 星系附近 0.2 度的范围内创建一个包含 10 幅 2MASS 图集的马赛克图像。我们将产生既背景匹配又未修改版的马赛克图像。最终的图像是将关于 m101 星系的两个马赛克图像及相对应的周围的图像。

本文不讨论 Montage 及针对 M101 更彻底的算法部分，详细内容可以参见 Montage 的官方网站[①]。当最终的图像建立时，这些被 Montage 用来增加图像以形成马赛克图的临时文件将不需要被继续保存。

① http://montage.ipac.caltech.edu/docs/m101tutorial.html

下载并安装完 Montage 安装包之后，我们也下载了 m101 拼合之前的图片。

它一共是 15.4M。 解压之后它会自动产生 m101 这个文件夹。它包含了五个子文件夹，其中 rawdir 包含了原始的 2MASS 文件，projdir 则则用于存储重映射图像，diffdir 用于存储微分图像，corrdir 用于存储背景图像，final 则是最终的处理好了的马赛克拼图。总体占用的存储空间会达到 20.7MB。

在 m101 文件夹中有模版文件(template.hdr)。在计算机的任何位置都可以运行 m101 文件，但是需要提前设置好 Montage 执行路径。

M101 任务的文件夹结构是

m101/

|--rawdir/

|　　|--10 2MASS atlas images, SIN projection

|

|--projdir/

|--diffdir/

|--corrdir/

|--final/

|--template.hdr

执行步骤如下：

1、解压压缩包

```
$ gunzip tutorial-initial.tar.gz
$ tar xvf tutorial-initial.tar
```

2、重映射图像生成一个图像的元数据表来描述已有的 rawdir 内的内容。

$ mImgtbl rawdir images-rawdir.tbl

执行成功之后显示

[struct stat="OK", count=10, badfits=0]

这一步将产生 images-rawdir.tbl，其中 images-rawdir.tbl 就是输出图像元数据表。

利用这一元数据表，运行 mProjExec 来重映射每一个图像：

$ mProjExec -p rawdir images-rawdir.tbl template.hdr projdir stats.tbl

输出结果是[struct stat="OK", count=10, failed=0, nooverlap=0]

其中涉及到的文件及文件夹为

Rawdir	包含了所有原始图像的文件夹
Images-rawdir.tbl	描述上述图像的元数据表
Template.hdr	马赛克拼图的头部模版
projdir	包含了重映射的所有图像
Stats.tbl	包含了每一图片的运行时间和状态

这一步骤将持续比较长时间，可能几分钟，取决于处理器的运算能力。这也是我们需要控制及调节的一点。因为输入的 2MASS 图像都是在一个正切平面映射中，mProjExec 可以利用快速映射算法(mProjectPP)来节省部分时间。

它同时也在 projdir 文件夹内产生一个重映射图像的集，和一个包含了处理时间和状态的文件:stats.tbl。当处理完了这些重映射图像之后，将生成一个新的元数据表，其包含了新的头部信息。

$ mImgtbl projdir images.tbl

系统输出为[struct stat="OK", count=10, badfits=0]

从而产生新的元数据表 images.tbl.

3、未修正马赛克拼图

现在我们从重映射的图像中生成一个未修正的马赛克拼图，也就是没有使用背景匹配技术。在 projdir 中运行 mAdd：

$ mAdd -p projdir images.tbl template.hdr final/m101_uncorrected.fits

系统输出[struct stat="OK", time=8]

其中涉及到的文件及文件夹有

Images.tbl	描述上述图像的元数据表
Template.hdr	马赛克拼图的头部模版
projdir	包含了重映射的所有图像
final/m101_uncorrected.fits	生成的输出文件

这一步骤大概要运行一分钟。

下一步就是产生一个灰度 JPEG 文件。

mJPEG -gray final/m101_uncorrected.fits 20% 99.98% loglog -out final/m101_uncorrected.jpg

输出显示[struct stat="OK", min=80.747, minpercent=20.00, max=180.914, maxpercent=99.98]

M101_uncorrected.jpg 图片显示如图 4.10：

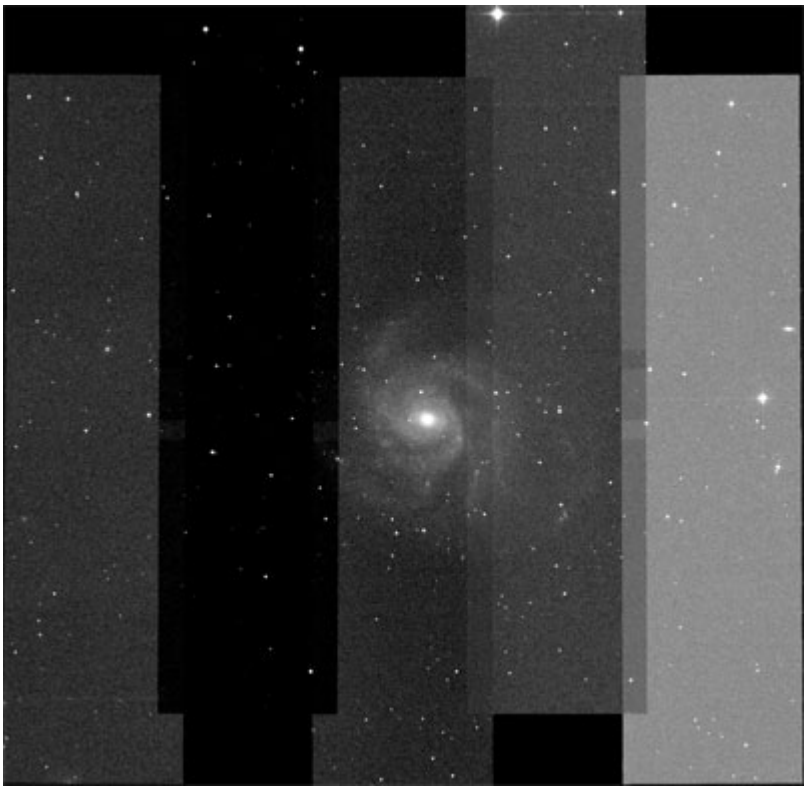


图4.10 M101 未修正图片

4、背景建模

为了使得重合图像之间的部分得到平滑处理，Montage 为每一个重合部分都生成了一个微分图像并且配置了一个平面。首先，Montage 需要知道那一部分是重合的，这个可以用 mOverlaps 来判断：

$ mOverlaps images.tbl diffs.tbl

[struct stat="OK", count=17]

其中 images.tbl 是描述文件的图像元数据表，diffs.tbl 输出文件表，同时也是产生的一个输出文件。这一过程需要几秒钟来完成，在整个 Montage 过程中是比

较短的。mDiffExec 使用这一文件来抽取出每一个重合的图像，并且在 diffdir 文件夹内产生一组微分图像。

$ mDiffExec -p projdir diffs.tbl template.hdr diffdir

[struct stat="OK", count=17, failed=0]

其中

Diffdir	微分图像的存储文件夹
Template.hdr	马赛克拼图的头部模版
projdir	包含了重映射的所有图像
Diffs.tbl	mOverlaps 生成的微分图像表

虽然 mDiffExec 的运行时间取决于要处理的微分图像数目，但是它的运行时间还是地狱 ProjExec.

下一步使用 mFitExec 来计算针对每一个微分图像的相平面适合参数。这一过程通常需要几秒钟，并且生成了相平面适合参数的元数据表 fits.tbl.

$ mFitExec diffs.tbl fits.tbl diffdir

[struct stat="OK", count=17, failed=0, warning=0, missing=0]

其中涉及到的文件和文件夹有：

Diffdir	微分图像的存储文件夹
Fits.tbl	包含了相平面适合参数的输出表
Diffs.tbl	mOverlaps 生成的微分图像表

5、背景匹配

现在 Montage 计算出平滑重合部分的最好的方式。这一过程将产生一个修正表，并且被应用在每一个图像上。输出文件是 correction.tbl。

$ mBgModel images.tbl fits.tbl corrections.tbl

[struct stat="OK"]

Images.tbl	重映射图像的元数据表
Fits.tbl	包含了相平面适合参数的输出表
Conrrection.tbl	全局修正表

接下来，使用 mBgExec 来真正的将背景匹配应用在重映射图像中：

$ mBgExec -p projdir images.tbl corrections.tbl corrdir

[struct stat="OK", count=10, nocorrection=0, failed=0]

这一步将在 corrdir 文件夹内产生一组背景匹配后的重映射图像。

这几步每一步都不需要超过一秒钟。

6、修正马赛克拼图

图4.11 M101修正后的图像

使用 mAdd 来修正马赛克拼图并且平滑化处理。

$ mAdd -p corrdir images.tbl template.hdr final/m101_mosaic.fits

[struct stat="OK", time=1]

一个灰度 JPEG 图像将被生成。

mJPEG -gray final/m101_mosaic.fits 20% 99.98% loglog -out final/m101_mosaic.jpg

[struct stat="OK", min=82.3272, minpercent=20.00, max=178.227, maxpercent=99.98]

4.5 Montage 实验结果

Montage 实验主要是考察内存对 QoS 指标的影响，这里我们的 QoS 指标定义为任务的完成时间，它同时也反映了处理速度。在实际应用中，这一速度如果太慢，后续节点就会空闲，而上一节点的大量输出不能得到及时处理；这一速度也不能太快，因为第一受制于系统资源本身的处理速度，第二受制于上一节点的输出速度也即本节点的输入速度。因为考虑到这一系统的输出输入数据流都是单一文件，所以我们令 QoS 的目标是本节点的处理速度跟踪输入速度。如此既避免了资源的浪费，提高了资源利用率，同时也提高了效率，满足优化目标。

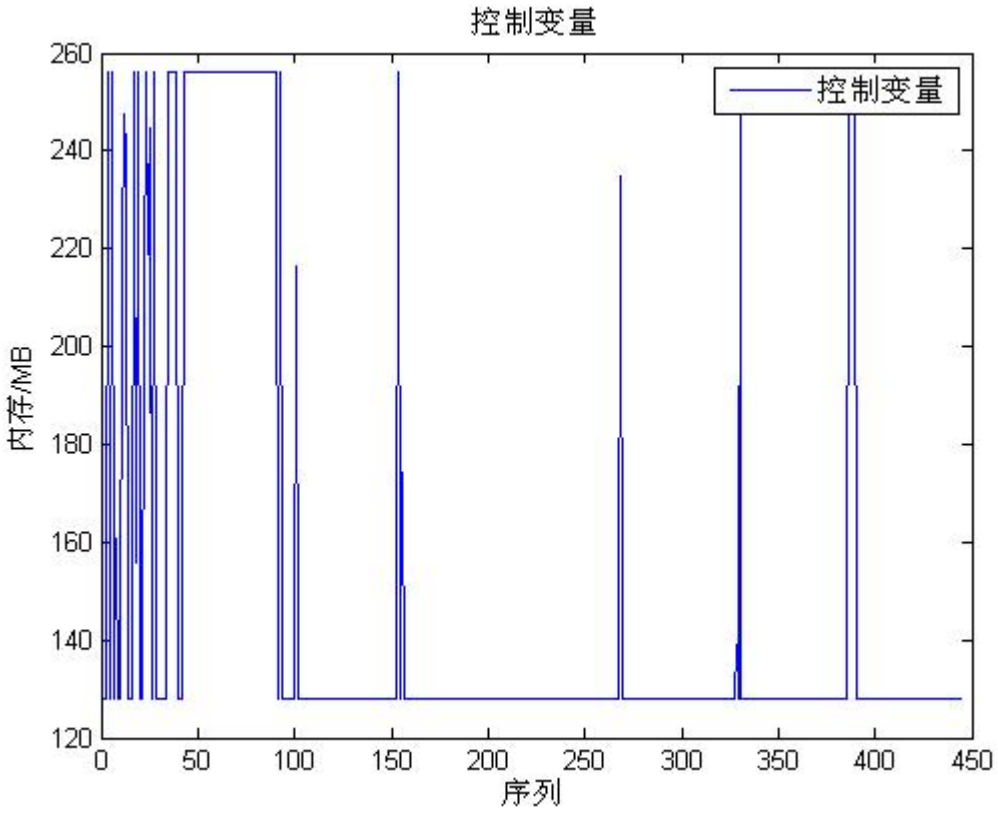


图4.12 设定内存变化曲线

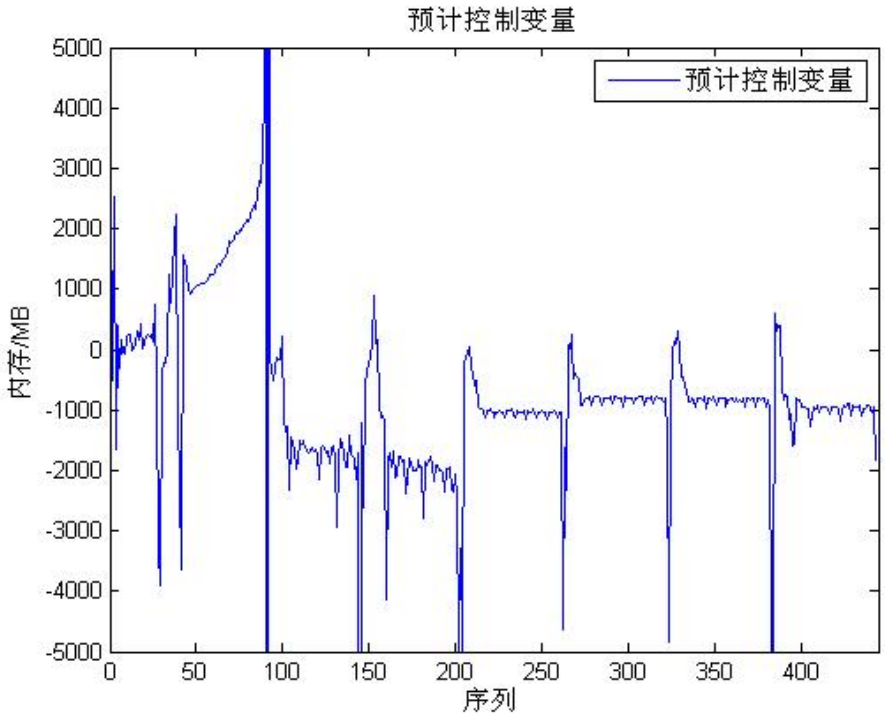


图4.13 预定内存变化曲线

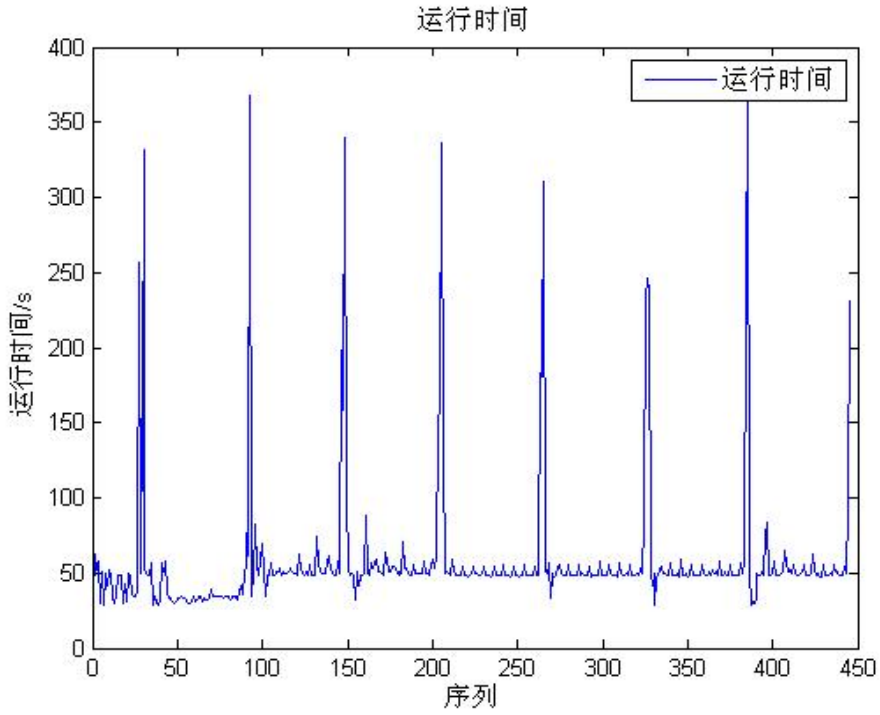


图4.14 QoS指标与内存资源(128~256MB)的关系

在这之前我们通过实验发现 CPU 对 Montage 的影响较小，为了方便分析系统控制的资源主要是内存。同因为 Montage 计算上的要求并不多，但是需要处理大文件，所以内存比较敏感。时 Xen 不支持动态调整带宽，所以我们主要分析内存对 QoS 的影响。我们定义 QoS 为该应用的处理时间。

图 4.12-图 4.14 表示的是在 QoS 指标（处理时间）设定为 50s 也就有上一节点的处理速度为 50s 时，该节点的性能。图 4.14 表示的是处理时间，图 4.12 表示的是控制的内存变化，图 4.13 表示的是最优控制器计算出来的希望的内存变化，但是由于内存的限制(128MB<RAM<256MB，此时工作于双机模式)，实际能够设定的内存值应该如图 4.12 所示。

这里介绍一下系统资源的有限性。由于我们做实验的物理机内存为 2G，排除掉自身占用的内存资源，虚拟系统占用的内存只有 700MB，如果两台虚拟机同时运行，每台虚拟机可以占用的最大内存不超过 256MB；如果单台虚拟机运行，那么该台虚拟机可以占用的最大内存也不超过 512MB。所以虽然通过最优控制器得出的预定的内存设定值可能较高也是无效的。

另外为了保证操作系统正常运行，我们设定虚拟机的内存不可低于 128MB，虽然通过最优控制器得出的预定的内存设定值可能较低也是无效的。这一最高和最低内存构成了内存设定的饱和环节，使得实际试验的效果不如 Matlab 仿真的好。

实验发现，虽然系统能够很快的追踪上设定的 QoS 值，但是每隔 50 个运算周期左右都会出现周期性的高峰，这个峰可能高达 300s，标明这个时候系统运行时间过长。下面分析可能的原因：

这里重点需要强调一下非线性的影响。使用的是较为简单的 ARMAX 模型。ARMAX 适合较为简单的计算系统，但是我们面对的系统是带延时且延时参数是随着应用的变化和系统性能而变化的。系统辨识本身也需要一定时间的数据积累才能够得到较为正确的辨识结果，通常来说是 100 个点的数据集。同时上述内存的饱和特性也可能影响了这一过程。

下面是另一次设定值，此时内存 RAM 在 128~512MB，此时也只能有一台虚拟机工作。

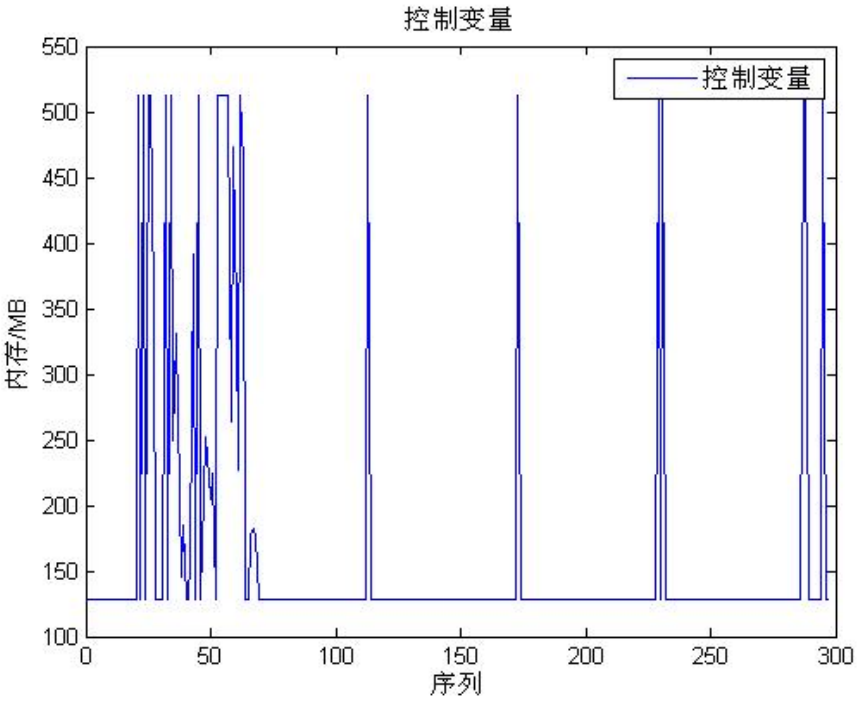


图4.15 内存变化曲线

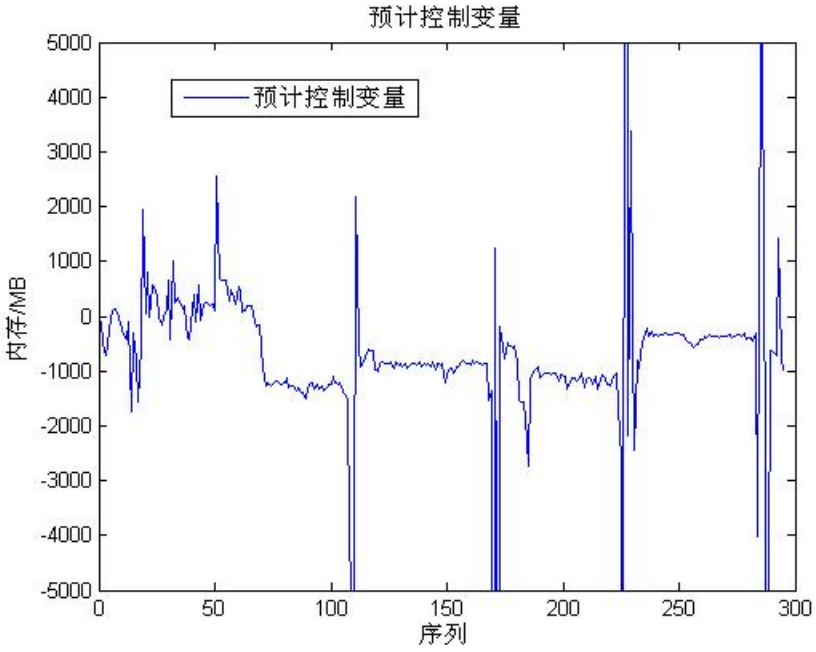


图4.16 预计内存变化曲线

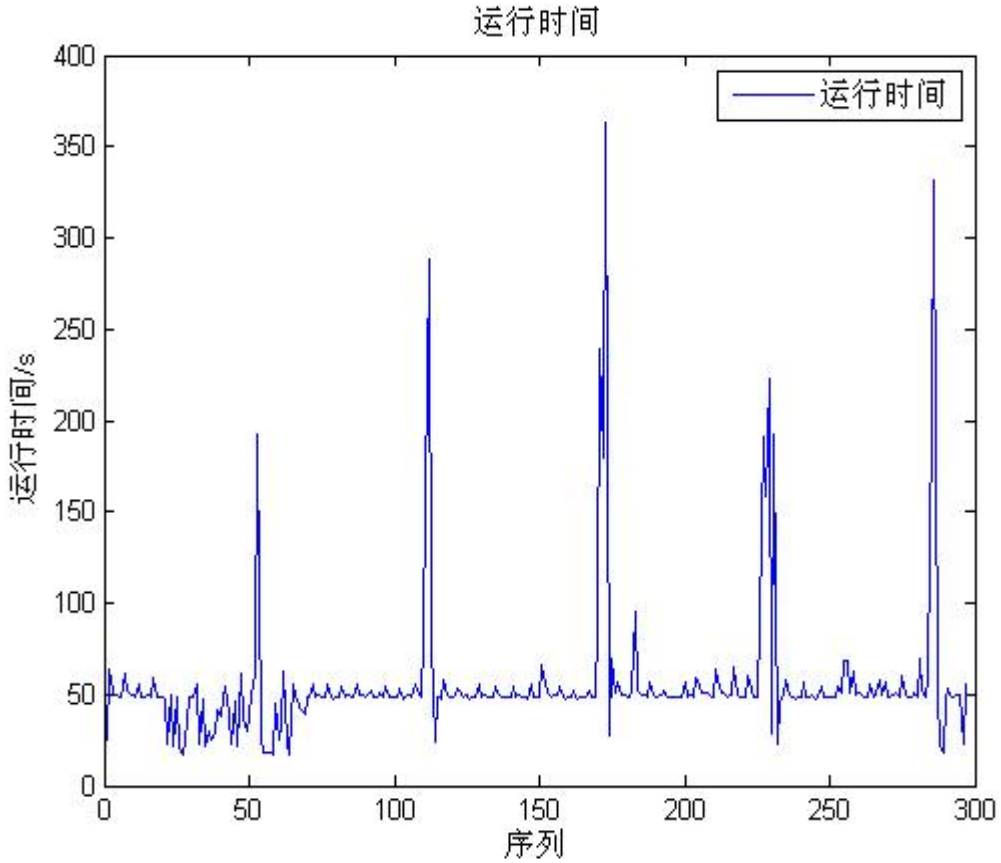


图4.17 QoS与内存设定(128~512MB)的关系

实验中也观察到了一定的震荡现象。系统设定值在某次阶跃之后，可能在较长时间内都处于不稳定乃至系统崩溃的状态，这些都是需要进一步解决的问题。图 4.18-图 4.20 所示的实验即是描述了这一过程。产生的原因经分析有以下几个可能：系统延迟时间随不同任务而改变，系统带宽的瓶颈问题，系统内存长时间分配过小导致操作系统不能正常工作。

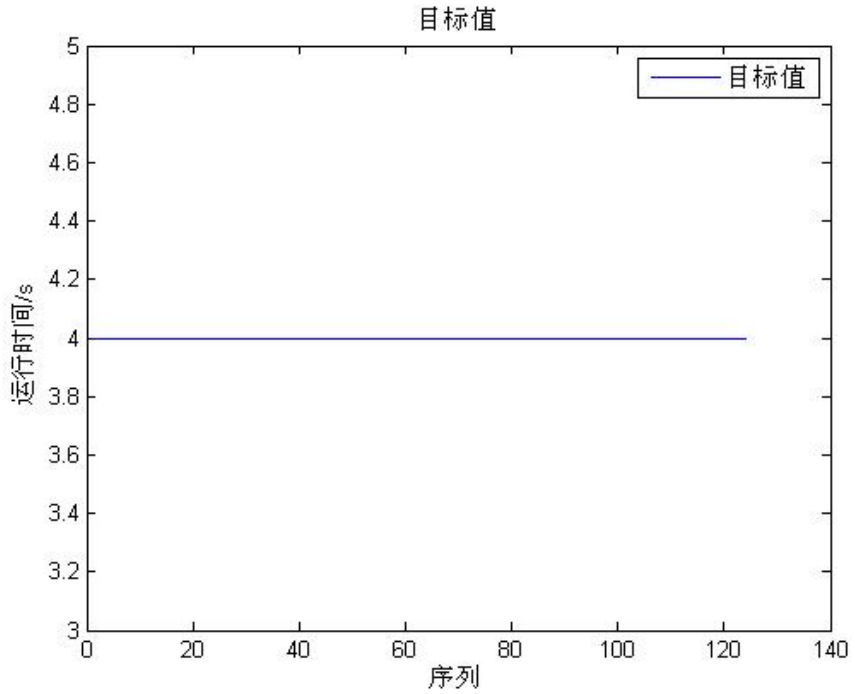


图4.18 跟踪目标值

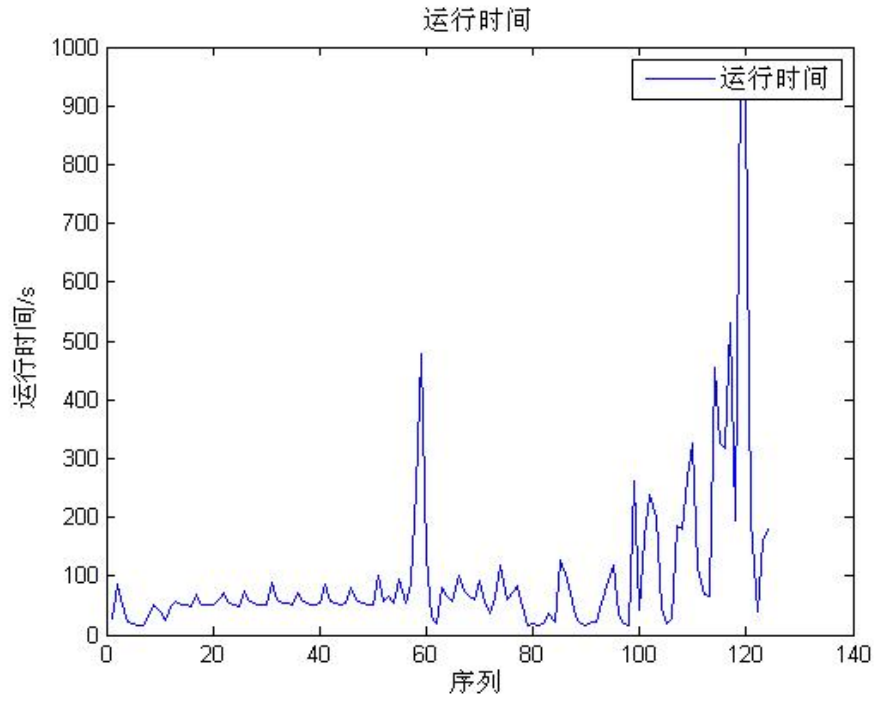


图4.19 实际运行时间

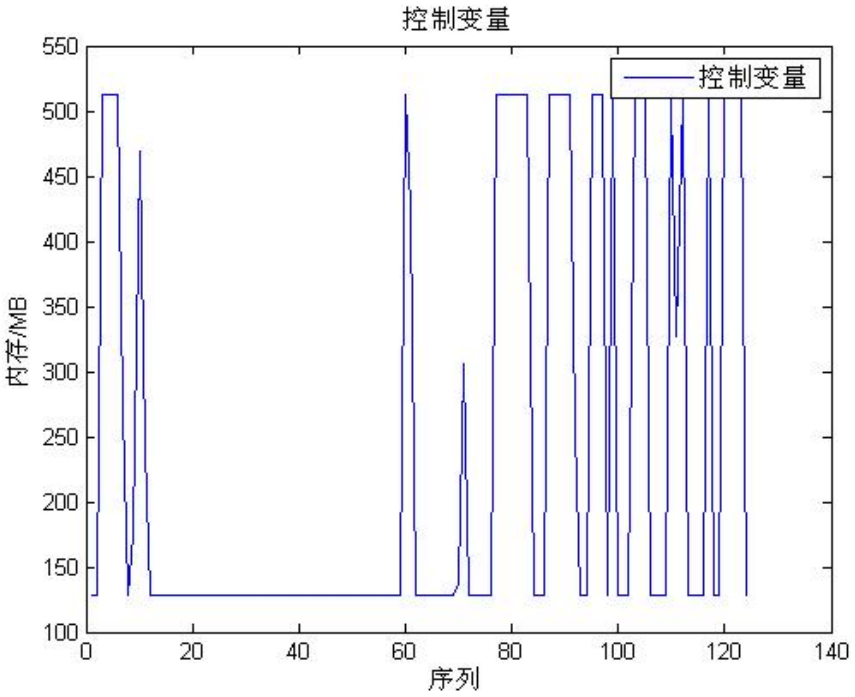


图4.20 内存变化曲线

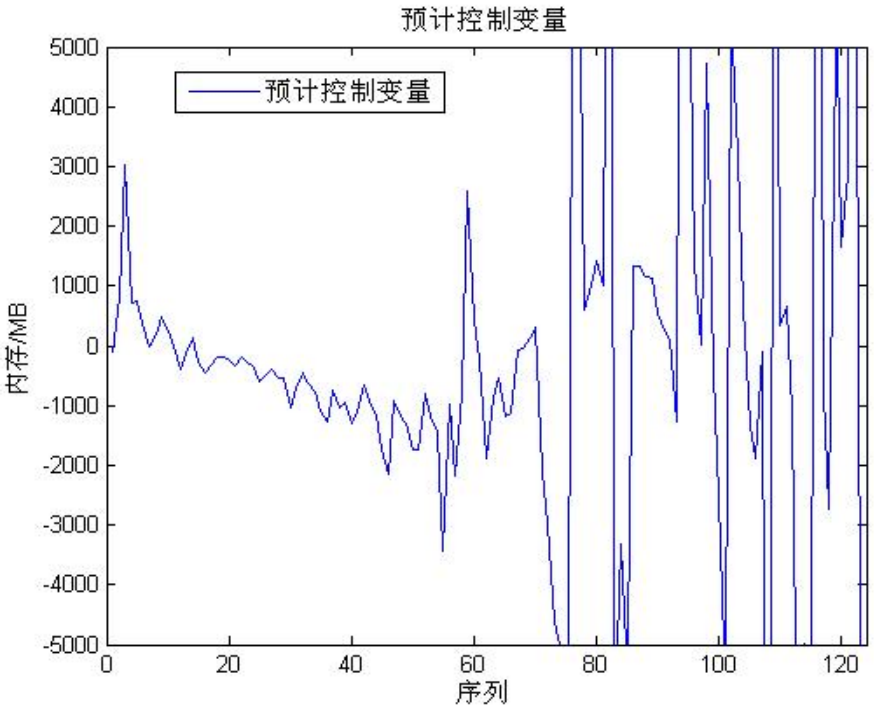


图4.21 预计内存变化曲线

4.6 本章总结

本章介绍了基于 Montage 和 Rmon 的网格应用及其在 ViGrid 系统中的综合表现，实验结果表明，ViGrid 系统能够有效地调整各个资源的分配，并且根据应用请求的不同表现出了对不同资源的敏感性。

第5章 总结

本文研究了在网格中面向数据流应用的基于虚拟技术的细粒度资源调度问题。并提出一种融合最优控制、系统辨识和虚拟化技术的综合算法，能够达到提高资源利用效率和提高工作效率的平衡。通过开发出基于 Xen 虚拟机的 ViGrid 虚拟网格系统，我测试了该算法的可行性，这也是我的主要工作。通过分析 Rmon 和 Montage 两个典型的网格应用，我验证了该算法的实际运行效果并且讨论了存在的问题。

通过实验结果，我发现 ViGrid 系统还存在不稳定的情况，且存在震荡现象，下一步的研究应该着重于对不稳定系统的建模分析及实验表现。同时，需进一步研究不同算法在 ViGrid 平台下的实际性能并且得出其比较结果。

图表索引

参考文献

[1] Ran Berman, Geoffrey Fox, Tony Hey. Grid Computing-making the Global Infrastructure a Reality [M] . USA: John Wiley and Sons Ltd. , 2003. 65 -100.

[2] I Foster, C Kesselman, et al. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration[EB/OL]. http://www.globus.org/research/papers.html, 2002.

[3] 网格服务的未来，http://www.csdn.net/subject/327/23503.shtm

[4] I Foster. What is the Grid? A Three Point Checklist[J/ OL] . http: / /www. gridtoday. com/02 /0722 /100136. html, 2002- 07- 20 .

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

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

[7] E Deelman, C Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda, GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists, Proc. 11th IEEE Int. Symp. on High Performance Distributed Computing, pp. 225-234, 2002.

[8] R Pordes for the Open Science Grid Consortium, “The Open Science Grid”, Proc. Computing in High Energy and Nuclear Physics Conf., Interlaken, Switzerland, 2004.

[9] Wen Zhang, Junwei Cao, Yisheng Zhong, Lianchen Liu and Cheng Wu, “An Integrated Resource Management and Scheduling System for Grid Data Streaming Applications”, Proc. 9th IEEE/ACM Int. Conf. on Grid (Grid 2008), pp.258-265, Tsukuba, Japan.

[10] J L Hellerstein, Y Diao, S Parekh, and D M Tilbury, Feedback Control of Computing Systems, Wiley-IEEE Press, August 2004.

[11] Figueiredo, R, P Dinda, and J Fortes, A Case for Grid Computing on Virtual Machines, Proc. 23rd Int. Conf. on Distributed Computing Systems, 2003.

[12] Keahey K, K Doering, and I Foster, From Sandbox to Playground: Dynamic Virtual Environments in the Grid, Proc. 5th Int. Workshop in Grid Computing, 2004.

[13] I Foster, T Freeman, K Keahy, D Scheftner, B. Sotomayer and X. Zhang, “Virtual Clusters for Grid Communities”, Proc. IEEE Int. Sym. on Cluster Computing and the Grid, May 2006.

[14] P Barham, B Dragovic, K Fraser, S Hand, T L Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, Xen and the Art of Virtualization, ACM Sym. on Operating Systems Principles, 2003.

[15] T F Abdelzaher, K G Shin, and N Bhatti, Performance Guarantees for Web Server End-systems: A Control-theoretical Approach, IEEE Trans. on Parallel and Distributed Systems, Vol. 13, 2002.

[16] P Bhoj, S Ramanathan, and S Singhal, Web2K: Bringing QoS to Web Servers, HP Labs Technical Report, HPL-2000-61, May 2000.

[17] Y Diao, N Gandhi, J L Hellerstein, S Parekh, and D M Tilbury, MIMO Control of an Apache Web server: Modeling and Controller Design, American Control Conference, 2002.

[18] Y Diao, J L Hellerstein, and S Parekh, Using Fuzzy Control to Maximize Profits in Service Level Management, IBM Systems J., Vol. 41, No. 3, 2002.

[19] A Kamra, V Misra, and E M Nahum, Yaksha: A Self-tuning Controller for Managing the Performance of 3-tiered Web Sites, in Proc. 12th IEEE Int. Workshop on Quality of Service, June, 2004.

[20] X Liu, X Zhu, P Padala, Z Wang, and S Singhal, Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform, in Proc. 46th IEEE Conference on Decision and Control, New Orleans, LA, 2007.

[21] P T Barham, B Dragovic, K Fraser, S Hand, T L Harris, A Ho, R Neugebauer, I Pratt, and A Wareld. Xen and the art of virtualization. In Proc. ACM Symposium on Operating Systems Principles (SOSP), 2003.

[22] M Rosenblum and T Garnkel. Virtual machine monitors: Current technology and future trends. IEEE Computer, 38(5), 2005.

[23] J E Smith and R Nair. The architecture of virtual machines. IEEE Computer, 38(5), 2005.

[24] VMware. http://www.vmware.com/.

[25] G Khanna, K Beaty, G Kar, and A Kochut. Application performance management in virtualized server environments. In Proc. IEEE/IFIP Network Operations and Management Symp. (NOMS), 2006.

[26] P Ruth, J Rhee, D Xu, R Kennell, and S Goasguen. Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. In Proc. IEEE Int. Conf. on Autonomic Computing (ICAC), 2006.

[27] P Shivam, A Demberel, P Gunda, D E Irwin, L E Grit, A R Yumerefendi, S Babu, and J S Chase. Automated and on-demand provisioning of virtual machines for database applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007. Demonstration.

[28] M Steinder, I Whalley, D Carrera, and I G D M. Chess. Server virtualization in autonomic management of heterogeneous workloads. In Proc. IFIP/IEEE Int. Symp. on Integrated Network Mgmt., 2007.

[29] X Wang, Z Du, Y Chen, and S Li. Virtualization-based autonomic resource management for multi-tier web applications in shared data center. Journal of Systems and Software, 2008.

[30] X Wang, D Lan, G Wang, X Fang, M Ye, Y Chen, and Q Wang. Appliance-based autonomic provisioning framework for virtualized outsourcing data center. In Proc. IEEE Int. Conf. on Autonomic Computing (ICAC), 2007.

[31] M Bennani and D A Menasce. Resource allocation for autonomic data centers using analytic performance models. In Proc. IEEE Int. Conf. on Autonomic Computing (ICAC), 2005.

[32] A Karve, T Kimbrel, G Pacici, M Spreitzer, M Steinder, M Sviridenko, and A Tantawi. Dynamic placement for clustered web applications. In Proc. Int. Conf. on WWW, 2006.

[33] C Tang, M Steinder, M Spreitzer, and G Pacici. A scalable application placement controller for enterprise data centers. In Proc. Int. Conf. on WWW, 2007.

[34] G Tesauro, R Das, W E Walsh, and J O Kephart. Utility-function-driven resource allocation in autonomic systems. In IEEE Int. Conf. on Autonomic Computing, 2005.

[35] Ahmed A Sorory, Umar Farooq Minhasy, Ashraf Aboulnagay, Kenneth Salemy, Peter Kokosielisz, Sunil KamathzAutomatic Virtual Machine Configuration for Database Workloads. Proc. of the 2008 ACM SIGMOD international conference on Management of data, 2008.

[36] A A Soror, A Aboulnaga, and K Salem. Database virtualization: A new frontier for database tuning and physical design. In Proc. Workshop on Self-Managing Database Systems (SMDB), 2007.

[37] G Weikum, A M onkeberg, C Hasse, and P Zabback. Self-tuning database technology and information services: from wishful thinking to viable engineering. In Proc. Int. Conf. on Very Large Data Bases (VLDB), 2002.

[38] R Agrawal, S Chaudhuri, A Das, and V R Narasayya. Automating layout of relational databases. In Proc. Int. Conf. on Data Engineering (ICDE), 2003.

[39] K Dias, M Ramacher, U Shaft, V Venkataramani, and G Wood. Automatic performance diagnosis and tuning in Oracle. In Proc. Conf. on Innovative Data Systems Research (CIDR), 2005.

[40] P Martin, H.-Y Li, M Zheng, K Romanufa, and W Powley. Dynamic reconguration algorithm: Dynamically tuning multiple buffer pools. In Proc. Int. Conf. Database and Expert Systems Applications (DEXA), 2000.

[41] D. Narayanan, E. Thereska, and A. Ailamaki. Continuous resource monitoring for self-predicting DBMS. In Proc. IEEE Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2005.

[42] A J Storm, C Garcia-Arellano, S Lightstone, Y Diao, and M Surendra. Adaptive self-tuning memory in DB2. In Proc. Int. Conf. on Very Large Data Bases (VLDB), 2006.

[43] M J Carey, R Jauhari, and M Livny. Priority in DBMS resource scheduling. In Proc. Int. Conf. on Very Large Data Bases (VLDB), 1989.

[44] D L Davison and G Graefe. Dynamic resource brokering for multi-user query execution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995.

[45] M N Garofalakis and Y E Ioannidis. Multi-dimensional resource scheduling for parallel queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996.

[46] M Mehta and D J DeWitt. Dynamic memory allocation for multiple-query workloads. In Proc. Int. Conf. on Very Large Data Bases (VLDB), 1993.

[47] The Montage project web page, http://montage.ipac.caltech.edu/

[48] The Flexible Image Transport System (FITS), http://fits.gsfc.nasa.gov, http://www.cv.nrao.edu/fits.

[49] http://www.atnf.csiro.au/people/mcalabre/WCS.htm.

[50] The Distributed Terascale facility, http://www.teragrid.org/

[51] The 2MASS Project, http://www.ipac.caltech.edu/2mass

[52] The Digitized Palomar Observatory Sky Survey (DPOSS), http://www.astro.caltech.edu/~george/dposs

[53] The Sloan Digital Sky Survey, http://www.sdss.org/

[54] Berriman, G B , Curkendall, D , Good, J , Joseph, J , Kataz, D S , Kong, Monkewitz, S , Moore, R , Prince, T , and Williams, R 2002. An Architecture for Access to a Compute Intensive Image Mosaic Service in the NVO. In Virtual Observatories, A S Szalay, ed. Proceedings of SPIE, Vol 4686, 91.

致 谢

本文的全部论文工作都是在导师曹军威教授的悉心指导下完成的，中心感谢曹军威教授对本人的谆谆教诲。他深厚扎实的学术功底、严谨求实的治学态度，使我受益匪浅。

感谢实验室全体同学对我学习和工作上的热情支持和鼓励。特别要感谢张文师兄在实验探究和论文写作方面提供的富有建设性的建议。

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：[illegible]　　日 期：2009.06.24

附录 A 外文资料的调研阅读报告

FINE-GRAIN RESOURCE ALLOCATION BASED ON VIRTUALIZATION IN DATA GRIDS

Virtualization technology was developed in the late 1960s to make more efficient use of hardware. Hardware was expensive, and there was not that much available. Processing was largely outsourced to the few places that did have computers. On a single IBM System/360, one could run in parallel several environments that maintained full isolation and gave each of its customers the illusion of owning the hardware.1 Virtualization was time sharing implemented at a coarse-grained level, and isolation was the key achievement of the technology. It also provided the ability to manage resources efficiently, as they would be assigned to virtual machines such that deadlines could be met and a certain quality of service could be achieved.

Traditional query processing in database management systems is usually carried out in two phases: optimization and execution. While the details of optimization have been improved over the years, the basic approach of optimization followed by execution has not been changed.

Adaptive Query Processing (AQP) is becoming more popular in recent years since most new projects that need query processing use some adaptive approach. The main reason is the emergence of new domains where it is nearly impossible to use traditional query processing, because of lack of reliable performance statistics or the dynamic nature of data and environments.

The data grid integrates wide-area autonomous data sources and provides users with a unified data query and processing infrastructure. AQP is required by data grids to provide better quality of services (QoS) to users and applications in spite of dynamically changing resources and environments.

Existing AQP techniques can only meet partially data grid requirements. Some existing work is either addressing domain-specific or single-node query processing problems. Data grids provide new mechanisms for monitoring and discovering data and resources in a cross-domain wide area. Data query in grids can benefit from this information and provide better adaptability to the dynamic nature of the grid environment.

In this work, an adaptive controller is proposed that dynamically adjusts resource shares to multiple data query requests in order to meet a specified level of service differentiation. The controller parameters are automatically tuned at runtime based on a predefined cost function and an online learning method. A testbed is built to evaluate our controller design. Simulation results show that our controller can meet given QoS differentiation targets and adapt to dynamic system resources among multiple data query processing requests while total demand from users and applications exceeds system capability.

1 Introduction

Resource allocation is an important problem in the area of computer science. Over the past years, solutions based on different assumptions and constraints have been proposed by different research groups [1, 2, 3, 4]. Generally speaking, resource allocation is a mechanism or policy for the efficient and effective management of the access to a limited resource or set of resources by its consumers. In the simplest case, resource consumers ask a central broker or dispatcher for available resources where the resource consumer will be allocated. The broker usually has full knowledge about all system resources. All incoming requests are directed to the broker who is the solely decision maker. In those approaches, the resource consumer cannot influence the allocation decision process. Load balancing [1] is a special case of the resource allocation problem using a broker that tries to be fair to all resources by balancing the system load equally among all resource providers. This mechanism works best in a homogeneous system.

To attain a fine-grain resource allocation, virtualization is the best method. Virtual machine monitor allows multiple commodity operating systems to share

conventional hardware in a safe and resource managed fashion (CPU, memory, bandwidth are all manageable), but without sacrificing either performance or functionality. Modern computers are sufficiently powerful to use virtualization to present the illusion of many smaller virtual machines (VMs), each running a separate operating system instance. This has led to a resurgence of interest in VM technology. In this thesis we present Xen, a high performance resource-managed virtual machine monitor (VMM) which enables applications such as server consolidation [5, 6], co-located hosting facilities [7], distributed web services [8], secure computing platforms [9, 10] and application mobility [11, 12]. Successful partitioning of a machine to support the concurrent execution of multiple operating systems poses several challenges. Firstly, virtual machines must be isolated from one another: it is not acceptable for the execution of one to adversely affect the performance of another. This is particularly true when virtual machines are owned by mutually untrusting users. Secondly, it is necessary to support a variety of different operating systems to accommodate the heterogeneity of popular applications. Thirdly, the performance overhead introduced by virtualization should be small.

To verify our theory, we would apply it in Adaptive Query Processing Problem among many grid applications. Query processing (QP) is an essential technology for traditional database management systems [1]. QP aims to transform a query in a high-level declarative language (e.g. SQL) into a correct and efficient execution strategy. Query optimization [14] is one of key techniques to achieve high performance data query using cost estimation in various types of database systems, e.g. multimedia, object-oriented, deductive, parallel, distributed databases, heterogeneous multidatabase systems, fuzzy relational databases, and so on [15]. Traditional query processing in database management systems is usually carried out in two phases: optimization and execution. While the details of optimization have been improved over the years, the basic approach of optimization followed by execution has not been changed.

Adaptive Query Processing (AQP) [16] is becoming more popular in recent years where optimization is required to be carried out during execution. The main reason is the emergence of new domains where it is nearly impossible to use traditional query processing, because of lack of reliable performance statistics or the

dynamic nature of data and environments. Two styles of adaptation in AQP is summarized in [17]: plan-change based adaptation provides a well-defined query execution plan but allow the plan to be changed during query processing; tuple-routing based adaptation views query processing as routing of tuples through operators and effects plan changes by changing the order in which tuples are routed.

The grid is such an environment where AQP is required for distributed data access. Grid computing aims for integration and sharing geographically distributed resources in multiple management domains [18]. While the grid is originally motivated by computational power sharing, data management turns out to be an essential service since large volumes of data processing are involved in most grid applications. Data grids [19] provide a transparent and seamless infrastructure for cross-domain distributed data access, leading to the following challenges for data query processing:

Performance of grid resources may change dramatically over time, since most these resources are shared and not dedicated to the grid.

QoS requirements of data query processing from grid applications may also change over time, since most grid applications last for a long time with large amount of data processing involved.

To adjust system parameters like CPU and Memory distribution, Xen provides a service to control the virtual machines dynamically.

Existing AQP techniques can only meet partially data grid requirements. Some existing work is either addressing domain-specific or single-node query processing problems [20]. Data grids provide new mechanisms for monitoring and discovering data and resources in a cross-domain wide area. Data query in grids can benefit from these information and provide better adaptability to the dynamic nature of the grid environment.

In this work, an adaptive controller is proposed that dynamically adjusts resource shares to multiple data query requests in order to meet a specified level of service differentiation. The controller parameters are automatically tuned at runtime based on a predefined cost function and an online learning method. A testbed is built to evaluate our controller design. Simulation results show that our controller can

meet given QoS differentiation targets and adapt to dynamic system resources among multiple data query processing requests while total demand from users and applications exceeds system capability.

The rest of this article is organized as follows: detailed research background of our work is introduced in Section 2; Section 3 provides a formal representation of the issue to be addressed in this work; corresponding adaptive controller is described in Section 4; Experimental evaluation results are included in Section 5; and the article concludes in Section 6.

2 Related Work

2.1 AQP

As mentioned above, AQP is required in scenarios where optimization is carried out during execution, e.g. continuous queries (CQs) and data streams [21]. In this section, a brief introduction to several existing projects is given below.

CQs are persistent queries that allow users to receive new results when they become available, and they need to be able to support millions of queries. NiagaraCQ [22], the continuous query sub-system of the Niagara project, a net data management system being developed at University of Wisconsin and Oregon Graduate Institute, is aimed to addresses this problem by grouping CQs based on the observation that many web queries share similar structures. NiagaraCQ supports scalable continuous query processing over multiple, distributed XML files by deploying the incremental group optimization ideas. A number of other techniques are used to make NiagaraCQ scalable and efficient:

NiagaraCQ supports the incremental evaluation of continuous queries by considering only the changed portion of each updated XML file and not the entire file.

NiagaraCQ can monitor and detect data source changes using both push and pull models on heterogeneous sources.

Due to the scale of the system, all the information of the continuous queries and temporary results cannot be held in memory. A caching mechanism is used to obtain good performance with limited amounts of memory.

The Telegraph implementation explores novel implementations for adaptive CQ processing mechanisms. The next generation Telegraph system, called TelegraphCQ [23], is focused on meeting the challenges that arise in handling large streams of continuous queries over high-volume, highly-variable data streams. Specifically, TelegraphCQ is designed with a focus on the following issues:

Scheduling and resource management for groups of queries

Support for out-of-core data

Variable adaptivity

Dynamic QoS support

Parallel cluster-based processing and distribution.

Researchers in Stanford University developed a general-purpose DSMS, called the STanford stREam dAta Manager (STREAM) [24], for processing continuous queries over multiple continuous data streams and stored relations. STREAM consists of several components:

The incoming Input Streams, which produce data indefinitely and drive query processing;

Processing of continuous queries typically requires intermediate state, i.e., Scratch Store;

An Archive, for preservation and possible offline processing of expensive analysis or mining queries;

CQs, which remain active in the system until they are explicitly reregistered.

Eddy [26] is a query processing mechanism continuously reorders operators in a query plan as it runs. By combining eddies with appropriate join algorithms, the optimization and execution phases of query processing is merged, allowing each tuple to have a flexible ordering of the query operators. This flexibility is controlled by a combination of fluid dynamics and a simple learning algorithm. Eddies are typical implementation of tuple-routing based adaptation.

Barham[25] have presented the Xen hypervisor which partitions the resources of a computer between domains running guest operating systems. their paravirtualizing design places a particular emphasis on performance and resource management. They have also described and evaluated XenoLinux, a fully-featured port of a Linux 2.4 kernel that runs over Xen.

Traditional query optimization can be successful is partially due to the ability to choose efficient ways to evaluate the plan that corresponds to the declarative query provided by the user. AQP merges optimization and execution because well-defined query plan cannot be achieved beforehand, especially for continuous queries and long-running data streaming.

2.2 AQP and the Grid

The grid brings more challenges for distributed data query processing. For example, information about data properties is likely to be unavailable, inaccurate or incomplete, since the environment is highly dynamic and unpredictable. In fact, in the grid, the execution environment and the set of participating resources is expected to be constructed on-the-fly. Existing solutions for AQP are either domain specific or focus on centralized, single-node query processing [27], so cannot meet adaptability demands of query processing on the grid. In this section, several efforts on AQP in the grid are given below.

Distributed query processing (DQP) is claimed in the work by University of Newcastle and University of Manchester to be important in the grid, as a means of providing high-level, declarative languages for integrating data access and analysis. A prototype implementation of a DQP system, Polar* [28], is developed running over Globus [29] that provides resource management facilities. The Globus components are accessed through the MPICH-G [30] interface rather than in a lower level way. To address the DQP challenge in a grid environment, the non-adaptive OGSA-DQP1 system described in [31] and [32] has been enhanced with adaptive capabilities.

A query optimization technique, Grid Query Optimizer (GQO) [33], aims to improve overall response time for grid-based query processing. GQO features a

resource selection strategy and a generic parallelism processing algorithm to balance optimization cost and query execution. GQO can provide better-than-average performance and is especially suitable for queries with large search spaces.

In the work described in [34], a data grid service prototype is developed that aims at providing transparent use of grid resources to data intensive scientific applications. The prototype targets three main issues

Dynamic scheduling and allocation of query execution engine modules into grid nodes;

Adaptability of query execution to variations on environment conditions;

Support to special scientific operations.

Based on the ParGRES database cluster, a middleware solution, GParGRES [35], exploits database replication and inter- and intra-query parallelism to efficiently support OLAP queries in a grid. GParGRES is designed as a wrapper that enables the use of ParGRES in PC clusters of a grid (Grid5000 [36]). There are two levels of query splitting in this approach: grid-level splitting, implemented by GParGRES, and node-level splitting, implemented by ParGRES. GParGRES has been partially implemented as database grid services compatible with existing grid solutions such as the open grid service architecture (OGSA) and the web services resource framework (WSRF). It shows linear or almost linear speedup in query execution, as more nodes are added in the tested configurations.

ObjectGlobe [37] is a distributed and open query processor for Internet data sources. The goal of the ObjectGlobe project is to establish an open marketplace in which data and query processing capabilities can be distributed and used by any kind of Internet application. Furthermore, ObjectGlobe integrates cycle providers (i.e., machines) which carry out query processing operators. The overall picture is to make it possible to execute a query with unrelated query operators, cycle providers, and data sources. Main challenges include privacy and security enduring. Another challenge is QoS management so that users can constrain the costs and running times of their queries.

Processing of multiple data streams in grid-based peer-to-peer (P2P) networks is described in [38]. Spatial matching, a current issue in astrophysics as a real-life e-Science scenario, is introduced to show how a data stream management system (DSMS) can help in efficiently performing associated tasks. Actually, spatial matching is a job of information fusion across multiple data sources, where transmitting all the necessary data from the data sources to the data sink for processing (data shipping) is problematic and in many cases will not be feasible any more in the near future due to the large and increasing data volumes. The promising solutions are dispersing executing operators that reduce data volumes at or near the data sources (query shipping) or distributing query processing operators in a network (in-network query processing). In-network query processing, as employed in the StreamGlobe [39] system, can also be combined with parallel processing and pipelined processing of data streams, which enables further improvements of performance and response time in e-Science workflows.

An adaptive cost-based query optimization is proposed in [40] to meet the requirements of the grid while taking network topology into consideration.

2.3 Control Theory for Adaptability

There have been many works on the implementation of adaptability of computing systems using control theory. For example, variations of proportional, integral, and derivative (PID) control is applied in [15] and [16] for performance optimization and QoS supports of Apache web servers. The linear quadratic regulator (LQR) is adopted in [17] for application parameter tuning in web servers to improve CPU and memory utilization. Fuzzy control is utilized in [18] for IBM Lotus Notes email servers to improve business level metrics such as profits. Adaptive control is used in [19] to improve application level metrics such as response time and throughput for three-tier e-commerce web sites. In the work described in [20], an adaptive multivariate controller is also developed that dynamically adjusts resource shares to individual tiers of multiple applications in order to meet a specified level of service differentiation. This work has the similar motivation to maintain QoS differentiation at a certain level with our work, though at a different context of virtualization based host sharing.

Traditional query processing research is focused on fine-grained adaptability within a single node or database. As mentioned in Eddies [26], eddies can be used to do tuple scheduling within pipelines, since they can make decisions with ongoing feedbacks from the operations they are to optimize. The work described in this article is focused on higher level coarse-grained data query processing optimization in a distributed data grid environment. Adaptability is achieved using feedbacks from real-time outputs of QoS levels of different applications.

3 Problem Statement

In this work, we consider a data grid query processing scenario described in Figure 1. A data grid is usually composed with many nodes, each serving a different dataset. If data replication strategies are used, different nodes can serve the same dataset, which is out of the scope of this work. A data grid application, e.g. scientific data analysis and processing, is in general a pipeline of tasks, each processing a different dataset. Users send requests to the grid for data query processing, each with different levels of priority corresponding to different levels of QoS requirements.

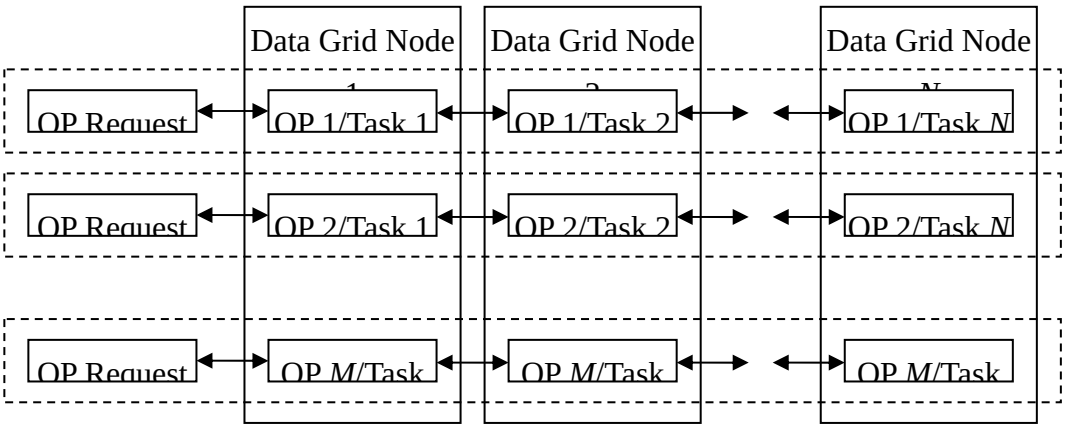


Figure 1 Query Processing in a Data Grid

In general, a data grid node is composed with large storage facilities and corresponding query processors, serving multiple QP requests. One of the key characteristics of the grid is that all nodes are shared instead of dedicated to the grid, so the available capacity of QP of a node varies over time. A grid node always gives highest priority to local users (resource owners) before sharing resources with grid

users. When demand from all QP requests from grid users exceeds the total available capacity of a node, the node becomes saturated and cannot meet QoS requirements of all QP requests. In this situation, since different grid users have different priorities and QoS requirements, it is desired to keep QoS differentiation among multiple QP requests.

Besides that multiple QPs are sharing one node to access a same dataset, different tasks of one QP on different nodes are also correlated with each other. For example, some scientific data analysis applications are pipelines of tasks, each looping through one dataset. After each loop of a task, the results are transferred to the next task for further data query and processing. The more resource located to a task, the more data query processing loops can be fulfilled, the higher QoS level can be achieved for a request. In order to achieve a higher end-to-end QoS, QoS levels of each tasks in an application pipeline have also to be coordinated. Reducing resource allocation to one task of an application leads to reduced load going to the next task in the pipeline. Such dependencies have also to be captured.

Let N be the number of datasets and tasks involved in a certain data grid application, each located at one data grid node. The total processing capacity of the node i, pi (i=1,2,……,N), can be normalized up to 100%. Let M be the number of concurrent requests sent from different users with different QoS requirements.

Let tij be the resource allocation for the task i of the request j. Since the total processing capacity of the node i is limit:

$$\sum_{j=1}^{M} t_{ij} = p_i (1 \le i \le N),$$

there are totally (M-1)*N such independent variables.

Let yj (j=1,2,……,M) be the normalized end-to-end QoS ratio for the request j. The desired QoS ratio for the request j is represented as Qj (j=1,2,……,M). Since there is:

$$\sum_{j=1}^{M} y_j = 1,$$

there are totally M-1 independent outputs.

The major issue we are trying to address in this work is to find appropriate tij, for all i's and j's, there is:

$$y_j = Q_j (1 \le j \le M - 1)$$

.

4 The Adaptive Controller

The problem described in Section 3 can be solved using existing methods in control theory. As shown in Figure 2, a closed-loop control system is designed between user requests and the data grid to determine the overall resource allocation scheme tij.

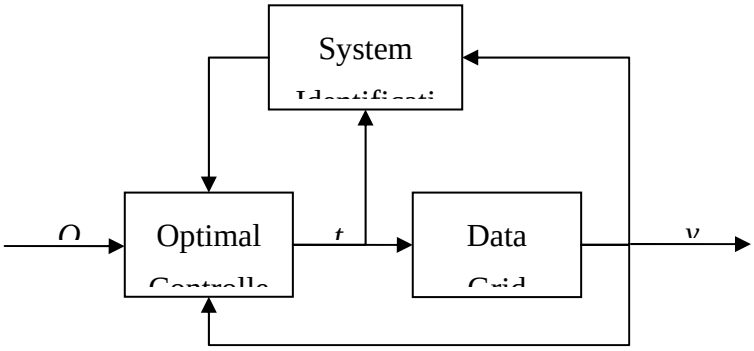


Figure 2 The Adaptive Controller for Query Processing in a Data Grid with Online System Identification Supports

In order to maintain QoS ratios for each request, the system has to figure out the relationship between the resource allocation scheme and QoS ratios. This can be represented using the linear, auto-regressive MIMO model and model parameters can be determined using online system identification. The actual optimal controller generates the optimal resource allocation scheme based on estimated model parameters and a predefined cost function. These are introduced in details below.

4.1 The Online System Identification

Composed with M users and N nodes, the system can be modeled using the linear, auto-regressive MIMO. The use of a MIMO model allows us to capture interactions and dependencies among data nodes for different application tasks. For example, reducing resource utilization for one QP task on a certain grid node will increase resource allocation for other QP tasks on the same node, and may reduce the load going into the next node of the same QP request. Such dependencies cannot be captured by individual SISO models. The MIMO model enables the controller to make tradeoffs between different QPs and their tasks when the system total demand from users' QP requests exceeds system capability. To simplify the problem, the

system model is written using the ARMAX model (with multiple inputs and single output, M=2):

$$A(q)y(k) = B(q)t(k-1) + C(q)e(k)$$
$$A(q) = 1 - A_1 q^{-1} - \cdots - A_n q^{-n}$$
$$B(q) = B_0 q^{-1} + \cdots + B_{n-1} q^{-n}$$
$$C(q) = 1 + C_1 q^{-1} + \cdots C_n q^{-n}$$

The values of above parameters may or may not change as system conditions and workload change. It is difficult to determine these values to represent all cases beforehand. Therefore, a self-learning approach is preferred where model parameters are estimated online and updated whenever new data has become available. In this work, the Matlab System Identification Toolbox is used to resolve system parameters online. In general, the order of the system is usually low in computer systems [47], which can be defined offline in advance.

For the convenience of computing, we rewrite this model to be:

$$y(k+1) = X\phi(k) + e(k+1)$$

Where

$$X = [B_0 \quad \cdots \quad B_{n-1} \quad A_1 \quad \cdots \quad A_n]$$
$$\phi(k) = [t^T(k) \quad \cdots \quad t^T(k \quad n+1) \quad y^T(k) \quad \cdots \quad y^T(k \quad n+1)]^T$$

For the sake of processing, we define:

$$\tilde{\phi}(k) = [0 \quad t^T(k-1) \quad \cdots \quad t^T(k-n+1) \quad y^T(k) \quad \cdots \quad y^T(k-n+1)]^T$$

We use an ARMAX model and its corresponding estimator to estimate the parameter matrix X, as provided by combing matrix A and B.

4.2 The Linear Quadratic Optimal Controller

The optimal goal of the adaptive controller is for the output y(k) to follow the reference input Q(k) as close as possible, as defined in Section 3. Note that the required QoS level from users may change over time. Meanwhile, we penalize large changes in resource allocation variables t(k). Here we adopt the following cost function:

$$J = E\{||W(y(k+1) \quad Q(k)||^2 + ||\Gamma(t(k) \quad t(k \quad 1)||^2\}$$

The following derivative is zero when the cost function J is at its minimum:

$$\frac{\partial J}{\partial t(k)} = 0$$

.

The derivation of the control law below is adapted from the controller synthesis in [20]. Note that X(k) and B0 are system identification results obtained using the ARMAX model estimator described in the last section.

$$t^*(k) = ((WB_0)^T WB_0 + P^T P)^{-1}\left((WB_0)^T W(Q(k) - X(k)\tilde{\phi}(k)\right) + P^T P\, t(k-1))$$

5 Performance Evaluation

As an example of the shared hosting platform presented in Section II, we present the experimental evaluation results of our controller and estimator design on a two-tier testbed.

5.1 Matlab Simulation Environment

We first build a simulation environment in the Matlab platform, which provides sufficient math functions. In this simulation environment, M=2, N=2, and the controlled system is a two-input-one-output system. The input variables are $t(k) = [t_1(k) \;\; t_2(k)]^T$, while each denotes the resource entitlement for the application. In our models, the two nodes have similar parameters of transfer functions, and therefore the resources would be like, which is investigated in next section. For the result shown in next section, we use a order of ARMAX model to be [na nb nc]=[2 4 1] by minimizing a robustified quadratic prediction error criterion.

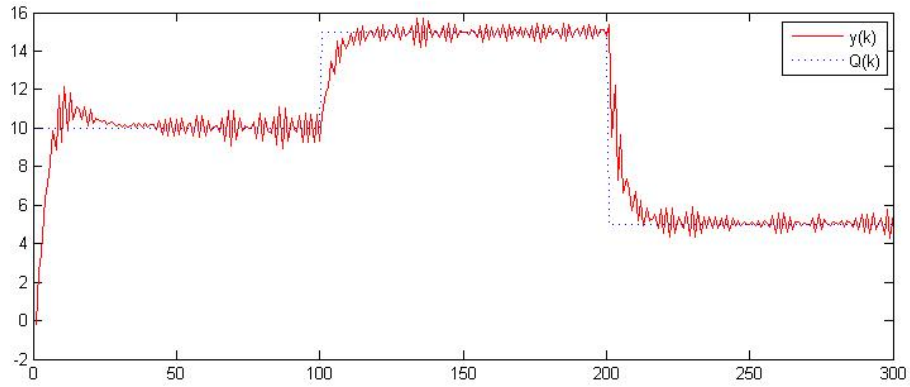
5.2 Matlab Experimental Results

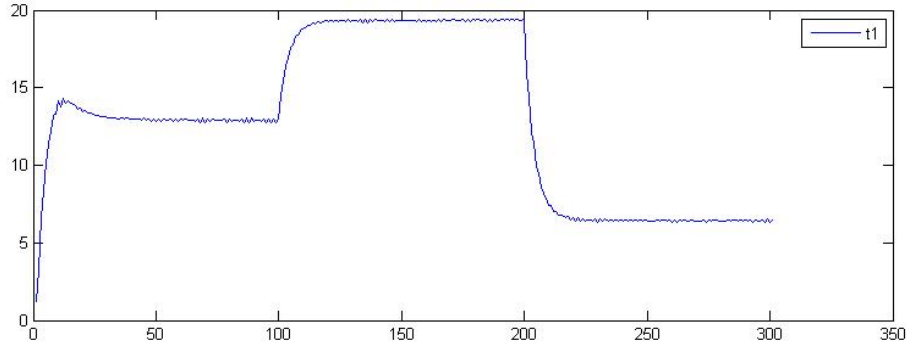
The experimental results depict clearly the effectiveness of this algorithm and the influence of Q and P in control performance.

In each experiment, we set the target output Q(k) to be 10 at t=0, and 15 at t=100, and 5 at t=200, in order to observe the tracking results. The output y(k) can get to Q(k) in less than 10 seconds, which is tolerable by the system requirements. What's more, in most cases, input of this system depicts the resource (usually CPU

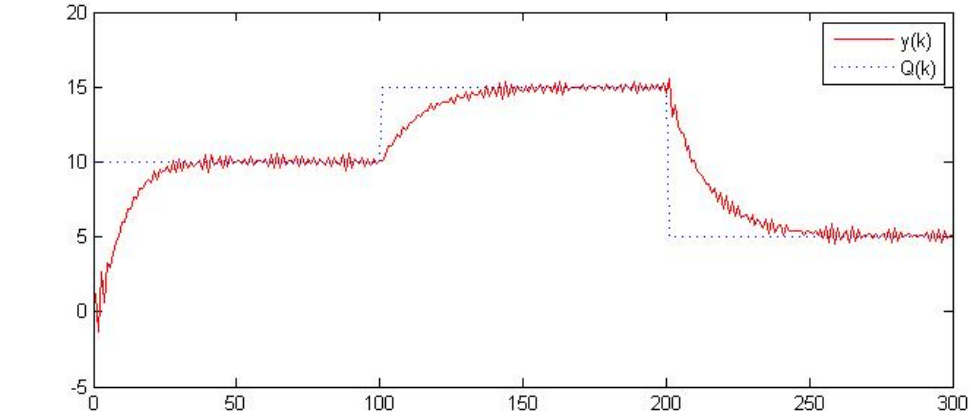
and memory consumption) allocated to an application. This figure shows inputs and target output are positively related, demonstrating a physical fact that QoS will improve with the increase of resource allocated. As we use similar transfer functions for the two nodes, the inputs are like.

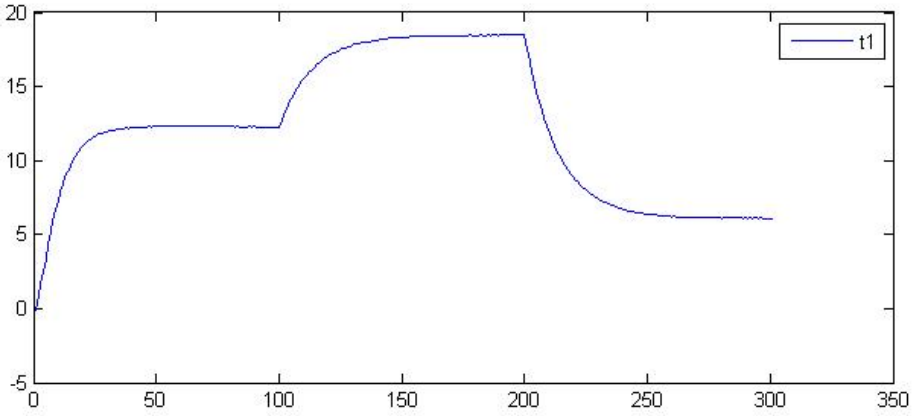
To further explore the characters of this algorithm, we select different Q and P to observe their influence in the control performance. Figure 3 shows the tracking result of output and the variation of two inputs with Q=I and P =1. Compared to Figure 4 with Q=1 and P=0.4, the decrease of P and relatively increase Q will smooth the curves of both output and inputs. This conclusion is credible as P serves as a weighted parameter of the continuity of inputs (and thus output). What's more, Q is a weighted matrix of the gap between target output and current output. Therefore, the curves of Figure 5 will be steeper at the point of change, compared to Figure 3 with a relatively increased Q. In practical systems, the fast tracking will increase the instability, while Figure 5 shows more burrs. In order to obtain a stable and fast performance, the Q matrix and P should be set in a specified zone.




t1


t2


y(k)
Q(k)


t1

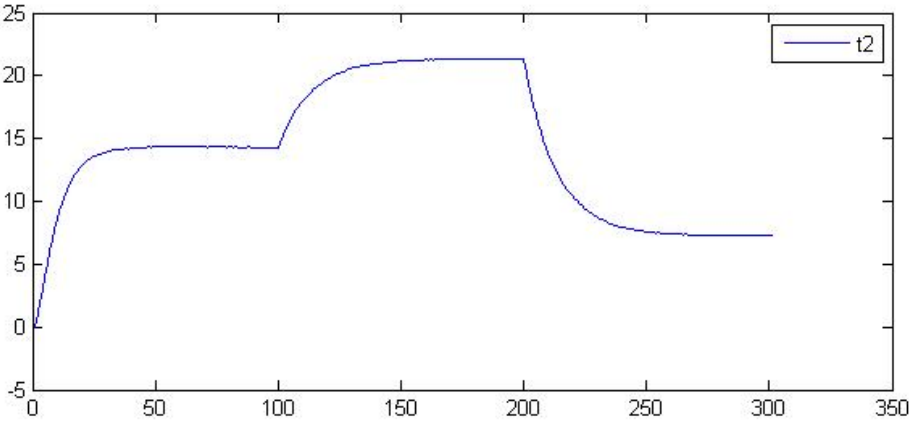

t2

Fig 4 Tracking result of output and variation of input with Q=I and P=0.4

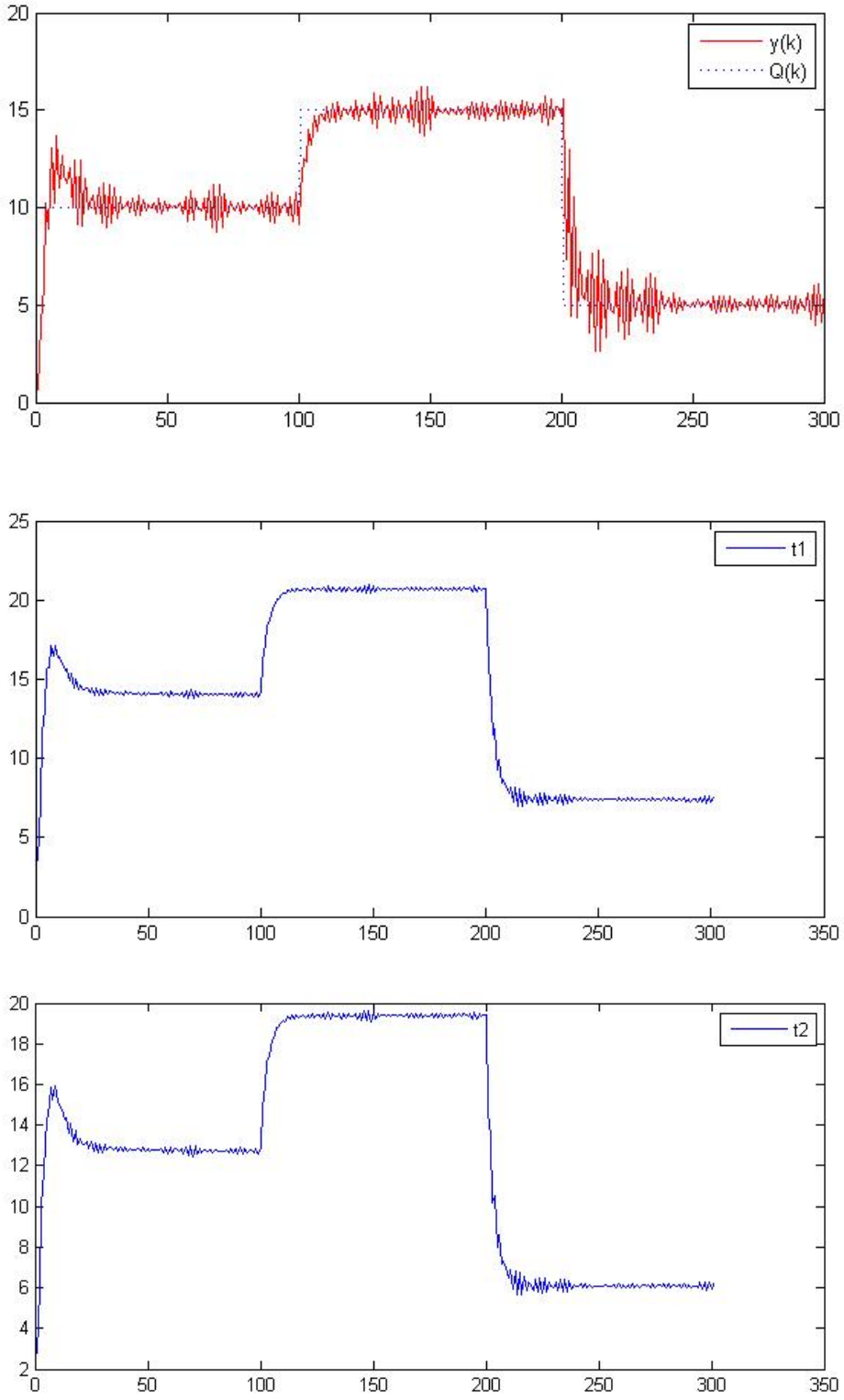


Fig 5 Tracking result of output and variation of input with Q=I and P=1.4

6 Conclusions

In this work, we address the connection of AQP and Data Grid, where AQP is required to provide better quality of services (QoS) to users and applications in spite of dynamically changing resources and environments. Existing AQP techniques are either addressing domain-specific or single-node query processing problems.

To solve this problem, we proposed an adaptive controller that dynamically adjusts resource shares to multiple data query requests in order to meet a specified level of service differentiation. The controller parameters are automatically tuned at runtime based on a predefined cost function and an online learning method. A testbed is built to evaluate our controller design. Simulation results show that our controller can meet given QoS differentiation targets and adapt to dynamic system resources among multiple data query processing requests while total demand from users and applications exceeds system capability.

Acknowledgement

This work is supported by National Science Foundation of China (grant No. 60803017), Ministry of Science and Technology of China under the national 863 high-tech R&D program (grants No. 2006AA10Z237, No. 2007AA01Z179 and No. 2008AA01Z118), Ministry of Education of China under the program for New Century Excellent Talents in University and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, and the FIT foundation of Tsinghua University.

参考文献

[1]. T. Bourke. Server Load Balancing. O’Reilly Media, 1 edition, August 2001.
[2]. T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on Software Engineering, 14(2):141–154, February 1988.
[3]. S. H. Clearwater. Market-based control. A Paradigm for Distributed Resource Allocation. World Scientific, Singapore, 1996.
[4]. C. Georgousopoulos and O. F. Rana. Combining state and model-based approaches for mobile agent load balancing. In SAC ’03: Proceedings of the 2003 ACM symposium on Applied computing, pages 878－885, New York, NY, USA, 2003. ACM Press.

[5]. C. A. Waldspurger. Memory resource management in VMware ESXserver. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), ACM Operating Systems Review, Winter 2002 Special Issue, pages 181.194, Boston, MA, USA, Dec. 2002.
[6]. Connectix. Product Overview: Connectix Virtual Server, 2003. http://www.connectix.com/products/vs.html.
[7]. Ensim. Ensim Virtual Private Servers, 2003. http://www.ensim.com/products/materials/datasheet_vps_051003.pdf.
[8]. A. Whitaker, M. Shaw, and S. D. Gribble. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. Technical Report 02-02-01, University of Washington, 2002.
[9]. G. W. Dunlap, S. T. King, S. Cinar, M. Basrai, and P. M. Chen. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), ACM Operating Systems Review, Winter 2002 Special Issue, pages 211.224, Boston, MA, USA, Dec. 2002.
[10]. T. Gar_nkel, M. Rosenblum, and D. Boneh. Flexible OS Support and Applications for Trusted Computing. In Proceedings of the 9th Workshop on Hot Topics in Operating Systems, Kauai, Hawaii, May 2003.
[11]. M. Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications, Calicoon, NY, Jun 2002.
[12]. C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), ACM Operating Systems Review, Winter 2002 Special Issue, pages 377.390, Boston, MA, USA, Dec. 2002.
[13]. W. Kim, D. S. Reiner, and D. S. Batory (Eds.), *Query Processing in Database Systems*, Springer Verlag, 1985.
[14]. M. Jarke and J. Koch, "Query Optimization in Database Systems", *ACM Comput. Surv.*, Vol. 16, No. 2, pp. 111–152, 1984.
[15]. C. T. Yu and W. Meng, *Principles of Database Query Processing for Advanced Applications*, The Morgan Kaufmann Series in Data Management Systems, 1997.
[16]. J. Hellerstein et al, "Adaptive Query Processing: Technology in Evolution", *IEEE Database Engineering Bulletin*, Vol. 23, No. 2, pp. 7-18, 2000.
[17]. A. Deshpande, J. M. Hellerstein, and V. Raman, "Adaptive Query Processing: Why, How, When, What Next", in *Proc. ACM SIGMOD 2006*, pp. 806-807, 2006.
[18]. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, CA USA, 1998.
[19]. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", *J. Network and Computer Applications*, Vol. 23, pp. 187-200, 2001.
[20]. A. Gounaris, N. W. Paton, R. Sakellariou, and A. A. A. Fernandes "Adaptive Query Processing and the Grid: Opportunities and Challenges", in *Proc. the 15th Int. Workshop on Database and Expert Systems Applications*, 2004.
[21]. B. Shivanath and W. Jennifer, "Continuous Queries over Data Streams", *SIGMOD Record*, Vol. 30, No. 3, pp. 109-120, 2001.

[22]. J. Chen, D. J. DeWitt, F. Tian and Y. Wang, “NiagaraCQ: A Scalable Continuous Query System for Internet Databases”, in Proc. ACM SIGMOD 2000, pp. 379-390, 2000.

[23]. S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, “TelegraphCQ: Continuous Dataflow Processing”, *in Proc. ACM SIGMOD 2003*, pp. 668, 2003.

[24]. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom, “STREAM: The Stanford Stream Data Manager”, *IEEE Data Engineering Bulletin*, March 2003.

[25]. Xen and Art of Virtualization, PPaul Barham, Boris Dragovic, Keir Fraser, PSteven Hand, PTim Harris, PAlex Ho, Rolf Neugebauer, PIan Pratt, PAndrew Warfield Dec. 2003 ACM SIGOPS Operating Systems Review Vol: 37 pp: 5

[26]. R. Avnur and J. Hellerstein, “Eddies: Continuously Adaptive Query Processing”, in *Proc. ACM SIGMOD 2000*, pp. 261–272, 2000.

[27]. Z. Ives, A. Halevy, and D. Weld, “Adapting to Source Properties in Processing Data Integration Queries”, in *Proc. ACM SIGMOD 2004*, pp. 395-406, 2004.

[28]. J. Smith, P. Watson, A. Gounaris, N. W. Paton, A. A. A. Fernandes, and R. Sakellariou, “Distributed Query Processing on the Grid”, *Int. J. High Performance Computing Applications*, Vol. 17, No. 4, pp. 353-367, 2003.

[29]. I. Foster, and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit”, *Int. J. Supercomputer Applications*, Vol. 11, No. 2, pp. 115-128, 1997.

[30]. N. Karonis, B. Toonen, and I. Foster, “MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface”, *J. Parallel and Distributed Computing*, Vol. 63, No. 5, pp. 551-563, 2003.

[31]. A. Gounaris, N. W. Paton, R. Sakellariou, A. A. A. Fernandes, J. Smith, and P. Watson, “Modular Adaptive Query Processing for Service-Based Grids”, in *Proc. IEEE Int. Conf. on Autonomic Computing*, pp. 295-296, 2006.

[32]. A. Gounaris, N. W. Paton, R. Sakellariou, A. A. A. Fernandes, J. Smith, and P. Watson, “Practical Adaptation to Changing Resources in Grid Query Processing”, in *Proc. 22nd Int. Conf. on Data Engineering*, pp. 165, 2006.

[33]. S. Liu and H. A. Karimi, “Grid Query Optimizer to Improve Query Processing in Grids”, *Future Generation Computer Systems*, Vol. 24, No. 5, pp. 342-353, 2008.

[34]. F. Porto, V. F. V. Da Silva, M. L. Dutra, and B. Schulze, “An Adaptive Distributed Query Processing Grid Service”, in *Proc. the Workshop on Data Management in Grids, VLDB 2005, LNCS 3836*, pp. 45-57, 2005.

[35]. N. Kotowski, A. A. B. Lima, E. Pacitti, P. Valduriez, M. Mattoso, “Parallel Query Processing for OLAP in Grids”, *Concurrency and Computation: Practice and Experience*, 2008.

[36]. R. Bolze, et al, “Grid'5000: a Large Scale and Highly Reconfigurable Experimental Grid Testbed”, *Int. J. High Performance Computing Applications*, Vol. 20, No. 4, pp. 481-494, 2006.

[37]. R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker, “ObjectGlobe: Ubiquitous Query Processing on the Internet”, *The VLDB J.*, Vol. 10, No. 1, pp. 48-71, 2001.

[38]. R. Kuntschke, T. Scholl, S. Huber, A. Kemper, A. Reiser, H. Adorf, G. Lemson, and W. Voges, “Grid-based Data Stream Processing in e-Science”, in *Proc. 2nd IEEE Int. Conf. on e-Science and Grid Computing*, Amsterdam, The Netherlands,

2006.
[39]. R. Kuntschke, B. Stegmaier, A. Kemper, and A. Reiser, “StreamGlobe: Processing and Sharing Data Streams in Grid-Based P2P Infrastructures”, in *Proc. Int. Conf. on Very Large Data Bases*, Rondheim, Norway, pp. 1259-1262, 2005.
[40]. S. Yahya, N. Faiza, and M. Najla, “An Adaptive Cost Model for Distributed Query Optimization on the Grid”, in *Proc. OTM 2004 Workshops, LNCS 3292*, pp. 79-87, 2004.
[41]. T. F. Abdelzaher, K. G. Shin, and N. Bhatti, “Performance Guarantees for Web Server End-systems: A Control-theoretical Approach,” *IEEE Trans. on Parallel and Distributed Systems*, Vol. 13, 2002.
[42]. P. Bhoj, S. Ramanathan, and S. Singhal, “Web2K: Bringing QoS to Web Servers,” *HP Labs Technical Report*, HPL-2000-61, May 2000.
[43]. Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D.M. Tilbury, “MIMO Control of an Apache Web server: Modeling and Controller Design,” American Control Conference, 2002.
[44]. Y. Diao, J. L. Hellerstein, and S. Parekh, “Using Fuzzy Control to Maximize Profits in Service Level Management,” *IBM Systems J.*, Vol. 41, No. 3, 2002.
[45]. A. Kamra, V. Misra, and E. M. Nahum, “Yaksha: A Self-tuning Controller for Managing the Performance of 3-tiered Web Sites,” in *Proc. 12th IEEE Int. Workshop on Quality of Service*, June, 2004.
[46]. X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal, “Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform”, in *Proc. 46th IEEE Conference on Decision and Control*, New Orleans, LA, 2007.
[47]. J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*, ser. ISBN: 0-471266-37-X, Wiley-IEEE Press, August 2004.

综合论文训练记录表

学生姓名	陈威威	学号	20050115c2	班级	自动.
论文题目	基于虚拟化技术的面向数据流应用的网格资源调度研究.				

主要内容以及进度安排

本研究的主要研究内容在于网格环境下的细粒度资源管理与调度，首先本研究面向时间需求最迫切的数据流应用，力求达到CPU，存储，带宽资源的协调。其次，本研究基于虚拟化技术，实现资源的定量控制。本研究利用控制理论中的系统辨识与反馈，实现QoS目标跟踪。调度安排包括于理论的学习与仿真，实验环境的安装测试，最终实现多功能的天文图像处理的应用使能与测试。

指导教师签字：[illegible]

考核组组长签字：[illegible]

2009 年 3 月 9 日

中期考核意见

本研究已经完成系统辨识与反馈最优控制调度理论研究与仿真测试，对资源管理与调度的效果有很好的体现。后面的主要工作在于搭建实验环境，取得实际的实验结果。

考核组组长签字：[illegible]

2009 年 4 月 27 日

指导教师评语

本论文是自动化学科与计算机学科的一个典型的交叉研究，主要是利用调度与优化算法实现复杂计算系统的资源管理与协调，内容覆盖面广，包括物格、虚拟化、控制理论等，研究方法包括 Method 仿真，还包括实际通过环境的搭建与测试，达到毕业设计的要求。

指导教师签字：

2009年 6月18日

评阅教师评语

该生的研究论文内容完整，表达清楚，在仿真和实际系统都取得结果，达到毕业设计的要求。

评阅教师签字：

2009年 6月18日

答辩小组评语

答辩讲述清楚，回答问题正确，同意通过答辩！

答辩小组组长签字：

2009年 6月18日

88×15%+86×25%+90×60%=

总成绩：89

教学负责人签字：

2009年 6月19日