计算网格作业调度算法的研究

摘　要

网格计算是借鉴电力网的概念提出来的。利用网络把分散在不同地理位置的计算机组织成一个“虚拟的超级计算机”，其中每一台参与计算的计算机就是一个“节点”，而整个计算环境是由成千上万个“节点”组成的一张“网格”。最终目的是希望用户在使用网格资源时就像现在使用电力一样方便，给用户提供可靠的、协调的、无处不在的和低廉的高端计算能力。计算网格为解决科学和工程领域一些大规模计算问题提供了理想的平台。

作业调度是计算网格中一个关键性的研究课题。在网格环境中，作业从提交给网格系统到作业结果处理完成，都一直处于网格作业管理系统的管理之下。由于网格具有大规模、异构、动态、分布和自治等特性，如何调度作业以满足用户的需求是一个极具挑战性的问题。

在研究分析计算网格作业调度算法现有成果的基础上，本文提出了一种基于贪心策略的调度算法和一种自适应调度算法。具体工作如下：

1、建立了计算网格作业调度的数学模型，考虑了网络延迟因素后，对模型进行了改进。

2、目前的作业调度算法有一个共同特点：一个作业只能分配给一个计算节点；而副本（Replica）利用了空间并行性，使多个性能不同的计算节点运行同一个作业，其优点是在付出一定的资源代价下减少作业的运行时间。在 Replica 的基础上，本文提出了一种贪心算法，在算法的不同阶段采用不同的贪心策略，模拟实验结果表明该算法可以减少资源浪费，提高系统的资源利用率。

3、针对网格环境的动态性和异构性，本文提出了一种自适应调度算法，该算法能动态地调用合适的调度算法。提出了一种系统负载平衡因子的定义，并给出了一种基于负载平衡的关键模块参考设计。

4、总结了计算网格作业调度算法的主要性能评价指标。针对提出的算法，给出了性能评价指标的计算公式。

关键词：网格计算；作业调度；贪心策略；副本；自适应；负载平衡

Research of Job Scheduling Algorithm in Computational Grid

Abstract

Grid computing is based on power grid. It organizes distributed computers as a "virtual super computer" by network. Every computer is called a node, and all the nodes form a "Grid". The target of grid is to make the users feel the use of grid is as convenient as using power grid. Grid computing provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. It provides an ideal platform to solve large-scale computing problems in scientific and engineering area.

Job scheduling is a key issue in computational grid. In grid environments, from job submission to result processing, all events about jobs are under the control of job management. Because grid environments are large-scale, heterogeneous, dynamic, distributed and autonomous, grid job management is complex and challenging.

Referring current research results on job scheduling algorithms in grid environments, this paper proposes a job scheduling algorithm based on greedy strategy and a self-adaptive algorithm. Our research works mainly include:

1. This paper constructs a mathematical model of job scheduling in grid environments. Considering the network delay, improvements are made to the model.

2. There is one common characteristic that one job can only be assigned to one node among the representative job scheduling algorithms. Replica exploits the space parallelism and enables various nodes execute the same job. The benefit is that it reduces the makespan at the expense of additional resource consumption. This paper proposes a greedy algorithm

based on replica. It adopts different greedy strategy at different phase of the algorithm. Simulation results show that it can reduce the resource consumption and improve the utilization.

3. With the grid environments are dynamic and heterogèneous, this paper proposes a self-adaptive algorithm which can dynamically call the suitable scheduling algorithm. The definition of load balance factor is also proposed. The key module based on load balance is designed.

4. This paper summarizes the key performance evaluation metrics. The formulas for the proposed algorithm' s performance evaluation metrics are also presented.

Keywords: Grid Computing; Job Scheduling; Greedy Strategy; Replica; Self-Adaptive; Load Balance

独 创 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含未获得_____（注：如没有其他需要特别声明的，本栏可空）或其他教育机构的学位或证书使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：王鹏　　签字日期：2006年5月28日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权学校可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。（保密的学位论文在解密后适用本授权书）

学位论文作者签名：王鹏　　导师签字：郭志文

签字日期：2006年5月28日　　签字日期：2006年5月28日

学位论文作者毕业后去向：

工作单位：　　电话：

通讯地址：　　邮编

0 前言

美国阿岗国家实验室的资深科学家、美国计算网格项目的领导人 Ian Foster 这样定义网格："网格是构筑在因特网上的一组新兴技术，它将高速互联网、高性能计算机、大型数据库、传感器、远程设备等融为一体，为科技人员和普通老百姓提供更多的资源、功能和交互性。因特网主要为人们提供电子邮件、网页浏览等通信功能，而网格功能更多更强，能让人们透明地使用计算、存储等其它资源"。网格计算将改变传统的 Client/Server 和 Client/Cluster 计算机系统结构，形成新的 Pervasive/Grid 体系结构，这种体系结构将整个网络视为一个巨大的虚拟计算机，可以提供一体化的、动态变化的、可灵活控制的、智能的、协作式信息服务。

网格把整个因特网整合成一台巨大的超级计算机，实现计算资源、存储资源、数据资源、信息资源、知识资源、专家资源的全面共享。当然，网格并不一定非要这么大。事实上，网格的根本特征是资源共享而不是它的规模，能透明而且高效率地为用户提供各种服务。

由于网格计算环境的资源具有分布共享、资源自治统一管理、资源异构和动态性等特点，使得网格计算环境的作业调度是一个比较棘手的问题，研究人员面临很大的挑战。本文在前人研究成果的基础上，对作业调度算法进行了深入地研究。主要提出了一种基于贪心策略的调度算法和一种自适应调度算法。

本文各部分安排如下：

第 1 章绪论介绍了研究背景、研究现状、研究内容及意义。

第 2 章对网格计算技术进行了概述。

第 3 章给出了计算网格作业调度的数学模型，在考虑了网络延迟因素后，对作业调度的数学模型进行了一定的改进。

第 4 章首先介绍了目前典型的网格作业调度算法；然后在 Replica 的基础上，提出了基于贪心策略的调度算法，称为 Greedy__Replica 算法，贪心策略的目标函数是作业运行时间和请求资源大小的乘积，在算法的不同阶段采用了两种截然相反的贪心策略，目的是提高资源的利用率；最后，通过一个实例来解释算法的运行过程。

第5章提出了一种自适应调度算法。由于网格计算环境存在很多的不确定因素和随机性，资源动态变化。针对这种情况，提出了一种自适应调度算法，该算法根据性能参数动态选择合适的调度算法，并给出了关键模块的参考设计，该模块只考虑系统的负载平衡。

第6章给出了调度算法性能评价的主要指标，并对提出的调度算法进行了仿真。

第 7 章对本文所做的工作进行了总结，并指出了需要进一步研究的问题。

1 绪论

1．1 研究背景

人类对计算能力的需求是无止境的。随着人类探索自然活动的深度、广度不断扩展，迫切需要功能更强、速度更快的计算机系统。目前，微处理器的时钟频率已经到了GHZ，其性能按照摩尔定律在持续增长。一方面制造技术与工艺、体系结构的发展在不断地提高着单个计算机设备的能力，各种计算资源的计算能力越来越强大；另一方面，Internet 飞速发展，网络速度成倍增长，主干网的带宽数量级从Kb/s发展到Gb/s。这一切使得聚集地理上分散的计算资源成为可能。目前，一种新的网络计算模式－网格计算[1]已成为产业界和学术界的研究重点。网格计算是指在动态的、异构的、广域的虚拟组织（Virtual Organization，VO）中进行的协同资源共享和问题求解。随着人们求解问题领域的不断拓展，所遇到的问题也越来越复杂，而且规模越来越大，解决这些问题所需要的计算能力也在大幅度提高。例如，天文望远镜每年所产生的数据不少于10Petabytes，处理如此大的数据量，即使是计算能力强大的超级计算机也不能胜任。此外，随着信息技术的快速发展，各个地理分散的组织部署了相当多的信息处理设备，这些资源在大部分时间内并没有被充分利用，它们处于分离状态，构成了一个个信息孤岛。如果我们从用户角度看计算机系统总体结构从1960年到2020年的演变，我们可以总结一条历史经验，姑且称之为三国定律：“天下大势：分久必合、合久必分”；每个分、合阶段大约主导15年。我们已经经历了三种模式。大型机/终端是早期的主导模式，其主要优点是使用方便和易于管理，其主要缺点是开放性差、不易扩展以及价格昂贵。为克服这些缺点，客户/服务器模式应运而生。集中在大型主机中的服务器功能被打散分布到多台独立的开放式服务器，通过网络与各类客户机（工作站、PC，网络终端，NC 等）相联。服务器聚集又被称为互联网数据中心（IDC）和服务器堆模式。它用一套物理上集中式服务器同时提供多台独立服务器的功能，并将尽量多的功能从客户端移回集中式服务器端，以提高系统的可管理性。我们目前正在进入一个新的“分”的阶段，即服务器聚集物理上分散到各地，但仍然保持虚拟的单一系统映像。这也可以看成是一种特殊的“合”，即多个IDC的资源被互连成为一个虚拟的网格计算机，各种客户端设备通过功用

方式使用网格资源。在这个网络计算时代，孤立的计算机系统、软件和应用将被网络化的产品和服务取代。世界将被互连成为一个开放的、一体化的、资源共享的全球电脑网络，也称为“全球大网格 （Great Global Grid)”，以满足人类对计算性能的不断需求。

网格计算是当今计算机科学领域最新兴起的一项有很高学术价值和应用价值的研究课题。目前，科学家们越来越多地使用超级计算机来研究复杂现象。例如，可以用来预测复杂的非线性现象，或者是在做实验之前，就可探索物理参数的变化规律，甚至还可以用来模拟现实世界中所发生的某些事件。然而超级计算机造价极高，通常只有一些特殊的部门，如航天、气象和军事等部门才有能力配置这样的设备；此外，某些应用对计算能力的要求极高，即使是超级计算机也无法胜任。这就需要将高性能计算依托 Internet 或其他高速网络将遍布世界各个角落的能力千差万别的计算资源联结在一起，形成大规模的几乎可以无限扩展的计算能力。例如由美国 NSF（国家自然科学基金）支持建设的网格计算系统 DTF（Distributed Tera scale Facility）就是一个成功的网格计算项目。该系统利用高速互联网络将 NSCA、SDSC、ANL 以及加州工学院等多家高性能计算中心连接起来，为数以万计的科研人员提供了万亿次的计算能力和超过 600T 的存储能力。与传统的高性能计算机不同，DTF 不是位于一个地点，而是由网络连接起来的一个计算联盟。通过分布在不同地点的高性能计算设备的接入，DTF 可以很容易的扩展计算能力和存储能力。DTF 所拥有的每秒 13.6 万亿次的计算能力大约是“深蓝”超级计算机的 1000 倍，强大的计算和存储能力使其能够满足如生命科学、大气分析、高能物理等众多领域对高性能计算的要求。因此，网格计算的研究具有很强的现实意义。

从网络发展的短暂历史来看，它对全球经济的发展起到了不可忽视的作用。网络的第一次浪潮是因特网，实现了全球计算机的连通；第二次浪潮是万维网，实现了网页的连通；第三次浪潮是网格技术，将实现全球资源的全面连通和共享。

1．2 研究现状

网格计算的重要战略意义及其广阔应用前景，使其成为当今吸引众多研究人员和巨大资金投入的研究热点。

美国政府从十年前就开始投资。1992 年，美国开始实施高性能计算方面的计划 HPCC，并投入巨资研究解决“巨大挑战问题”的环境、方法和技术，累计使用的基础研究经费已近五亿美元。美国军方更为积极，美国国防部已在规划实施一个宏大的计划，称为“全球信息网格（Global Information Grid)”，预计在 2020 年完成。英国政府已决定投资 1 亿英镑，用来建设“英国国家网格（UK National Grid)”。

随着网格研究在学术界的加速，信息产业界的大公司也相继公布了与网格目标一致的研究开发计划。惠普、IBM、微软、Sun 等公司最近取得共识，支持 XML、SOAP、UDDI 等万维网标准，从而更有利于开发新一代的网络应用，即万维网服务。其目的是将因特网上的资源和信息汇聚在一起，组合成企业和消费者所需要的服务。2002 年 8 月，IBM 宣布投入四十亿美元，启动一个全公司的“网格计算创新计划”，将网格计算从学术界的科学计算应用扩展到商业应用。其它公司也不甘示弱，Sun 公司宣布推出新的 Grid Engine 5.3 软件的 Beta 测试版，该软件使企业内的计算机的连接更为方便。Sun 目前也已启动了以 Grid Engine 分布式资源管理软件为基础的开放源代码战略。Microsoft 的研究部门也积极参与相关项目，包括容错远程文件系统 Farsite 和分布式系统 Millenium 的建设。Intel 大力倡导对等计算（P2P)，对等计算技术的主要用途之一是充分挖掘连接在网络上的成千上万个 PC 的处理能力和存储能力，以处理一些需要大型机，甚至超级计算机才能承担的任务。

国外具有代表性的研究项目和成果主要有 Globus[2]，Legion[3]，Nimrod-G，Netsolve，Apples 和 Condor 等。国内主要的网格研究项目是织女星（VEGA）网格[4]项目。

Globus 网格项目是由美国阿岗国家实验室等科研单位共同研发，是目前国际上最有影响的网格计算项目之一。它发起于 20 世纪 90 年代，其最初目的是希望把美国境内的各个高性能计算中心通过高性能网络连接起来，方便美国的大学和研究机构使用，提高高性能计算机的使用效率。Globus 项目组认为，大型应用项目应该由许多组织协同完成，这些组织通过网格计算环境形成一个统一的“虚拟组织”，网格计算环境中的用户、成员、资源可随时加入这种虚拟组织。各个组织拥有的各种资源都可被虚拟组织中的成员共享，并且各成员可以方便地协同完成各种分布式应用和工作。

Legion 是美国维吉尼亚大学开发的基于面向对象的网格操作系统，它和 Globus 类似，提供地理分布的无缝的异构系统集成。它是面向对象技术在网格计算领域应用的重要实例，支持透明的调度、数据管理、容错、站点自治和各种安全选项。它将网格计算环境视为一个世界范围的抽象计算机，其设计目标是让用户在 Legion 环境中只感觉到“一台”大的计算机，而网格计算用户在这台大计算机上进行程序设计。在 Legion 中，一切都是对象，Legion 规定了对象交互的消息格式与高级协议，通过对象的一组方法描述其接口，但是对编程语言和具体的通信协议没有规定。

织女星（VEGA）网格项目是由中国科学院计算技术研究所承担的研究课题。织女星网格项目的名称来源于其四个特性：

1、通用服务（Versatile Services）：通用服务与资源、共性技术；

2、辅助智能（Enabling Intelligence）：自动、自我、动态、交互、共性智能支持技术；

3、全局一体（Global Uniformity）：连通性、单一系统映像、互操作性；

4、自主控制（Autonomous Control）：用户自主、公开标准、以人为中心。

织女星网格项目组主要包括知识网格（Knowledge Grid）、信息网格（Information Grid）、服务网格（Service Grid）、基础研究和网格操作系统（Grid Operating System）五个部分。

可以将网格计算的发展总结为以下三个阶段：

1、萌芽阶段：90 年代早期，主要是建设千兆测试床，以及进行一些元计算实验；

2、早期实验阶段：在 90 年代中期到晚期，主要为学术研究实验；

3、迅速发展阶段：2002 年以来，出现了大量的应用社团和项目，如 IBM, Microsoft, SUN 等开展基础设施建设和使用。形成了具有相当规模的 GGF（Global Grid Forum）组织。

1．3 研究内容及意义

在网格系统中，有大量的应用程序要运行，这些应用程序共享网格的各种资源，如何才能够合理利用这些资源，获得最大的网格性能是网格环境研究需要解

决的主要问题；同时，资源管理和作业调度是连接网格底层与高层功能的纽带，是协调整个网格系统有效运转的中枢，是网格的核心技术问题，对这部分网格技术的研究具有重要的意义。

在分布式网格环境中，由于网格环境具有动态性、异构性和分布性等特点，使得作业调度比传统集中式操作系统中的作业调度要复杂得多。高效的作业调度算法对于满足网格用户的需求和提高系统的效率都是至关重要的。本文的作业调度模型是针对相互独立作业，即作业之间不存在相互依赖关系，它们之间没有数据通信。参数扫除（Parameter-Sweep）应用通常就属于这种类型。参数扫除应用存在于很多重要的应用领域，例如生物信息、数据挖掘、商业模型模拟和电子CAD 等，它们包含了很多彼此独立的同构作业。例如，SETI@home 利用互联网闲置计算能力寻找外星生命，每个 SETI@home 任务要花费 3.9 万亿个浮点操作，或者在 500MHz Pentium II 上运行大约 10 小时，而每个任务只需要下载 350KB 的数据和上传 1KB 的数据[5]。

2 网格技术概述

2．1 网格计算的概念

网格是借鉴电力网概念提出来的，网格最终目标是希望用户在使用网格时，就如同现在使用电力一样方便。网格计算的概念最早起源于元计算（Metacomputing）[6]，它是由 Smarr 和 Catlett 引入的。在 I-WAY[7]项目中提出了网格计算的概念。虽然对网格计算的研究已经有了很大的进步，但是到目前为止，学术界和产业界对什么是网格计算还没有一个普遍接受的定义，关于网格概念仍存在很大的分歧和争议。有人把网格看成为了下一代互联网，国外媒体常用“下一代因特网”、“国际互联网 2”等词语来描述网格相关的技术。1998 年，Ian Foster 在《网格：一种未来计算基础设施蓝图》[8]一书中将网格描述为：“网格是构筑在互联网上的一组新兴技术，它将高速互联网、高性能计算机、大型数据库、传感器、远程设备等融为一体，为科技人员和普通老百姓提供更多的资源、功能和交互性。互联网主要为人们提供电子邮件、网页浏览等通信功能，而网格功能则更多更强，人们可以透明地使用计算、存储等资源。一个计算网格是一个硬件和软件基础设施，此基础设施提供对高端计算能力可靠的、一致的、普遍的和不昂贵接入”。在 2000 年的一篇题为“网格剖析”[1]的文章中，指出网格计算关心的是：在动态的，多机构的虚拟组织中协调资源共享和协同解决问题。其核心概念是：在一组参与节点（资源提供者和消费者）中协商资源共享管理的能力，利用协商得到的资源池共同解决问题。此后，Ian Foster 等人进一步完善网格定义，认为网格实际上是满足如下三个条件的系统：

1、协调非集中控制资源

网格整合各种资源，协调各种使用者，这些资源和使用者在不同控制域中。例如，个人电脑和中心计算机，相同或不同公司的不同管理单元，网格还要解决在这种分布式环境中出现的安全、策略、使用费用、成员权限等问题。否则，只能算本地管理系统而非网格。

2、使用标准、开放、通用的协议和界面

网格建立在多功能的协议和界面之上，这些协议和界面解决认证、授权、资源发现和资源存取等基本问题。否则，只算一个具体应用系统而非网格。

3、得到非平凡的服务质量

网格允许它的资源被协调使用，以得到多种服务质量，满足不同使用者的需求，如系统响应时间、流通量、有效性、安全性及资源重定位，使得联合系统的功效比其各部分的功效总和要大得多。

总之，网格作为一种全新的、更加方便的计算模式，打破了传统共享与协作的限制，以“虚拟组织”的方法，实现了全社会范围内的资源共享与服务协作，可以有效地解决目前尚解决不了的复杂问题。它将彻底改变人们对“计算机应用”的看法，网格可以联合并放大全社会的计算能力，解决计算能力的限制，把“资源”送到你的桌面，把“全社会的计算能力”送到你的桌面，甚至可以把“应用”放到网格中完成，连“桌面”都可以节省。

2．2 网格的特点

网格作为一种新出现的基础性设施，和其他的系统相比，具有以下几个重要特点：

1、分布与共享

分布性是网格最主要的特点。网格的分布性首先是指网格资源是分布的。组成网格的各种物理设施分布在不同的地理位置，而不是集中在一起。网格资源虽然是分布的，但是它们却是可以充分共享的。共享是网格的目的，没有共享便没有网格，解决分布资源的共享问题是网格的核心任务。分布是网格硬件在物理上的特征，而共享是在网格软件下实现的逻辑上的特征。

2、自相似性

自相似性指局部和整体之间存在着一定的相似性，局部往往在许多地方具有全局的某些特征，而全局的特征在局部也有一定的体现。自相似性的典型例子是分形模型。自相似性在许多自然和社会现象中都大量存在，一些复杂系统都具有这种特征，网格也不例外。可以认为国家级的网格是在省一级的网格基础之上建造起来的，国家级主干网要有更大的带宽，只有这样才可以将不同省份的子网格连接起来提供满意的通信服务；国家级和省级网格都会有各自的计算中心，只不过在计算能力上有差异而已；它们也都需要管理结点，只不过国家级的管理结点管理功能需要更多、更强大。除了相似性外，整体和部分之间必然有不同的地方。

3、动态性与多样性

对于网格来说，决不能假设它是一成不变的。原来拥有的资源或者功能，在下一时刻可能就会出现故障或者不可用；而原来没有的资源，可能会随着时间的推移不断地加入进来。网格的动态性包括动态增加和动态减少两个方面的含义。当网格资源的动态减少或者资源出现故障时，要求网格能够及时采取措施，实现任务的自动迁移，做到对高层用户透明或者尽可能减少用户的损失。在网格的设计与实现时，必须考虑到新的资源能否很自然地加入到网格中来，并且可以和原来的资源融合在一起，共同发挥作用，解决网格的扩展性问题。网格扩展要求体现在规模、能力、兼容性等几个方面。开始网格的规模往往不是特别大，不需要也不可能一步到位，但是网格应该能够允许对它自身进行多种形式地扩展，网格规模扩展后，网格的相应管理软件也应该能够满足扩展性的要求，网格软件的升级要能够向下兼容。网格资源是异构和多样的，在网格环境中可以有不同体系结构的计算机系统和类别不同的资源，因此网格系统必须能够解决这些不同结构、不同类型资源之间的通信和互操作问题。

4、自治性与管理的多重性

网格上的资源，首先是属于某一个组织或者个人的，因此网格资源的拥有者对该资源具有最高级别的管理权限，网格应该允许资源拥有者对他的资源有自主的管理能力，这就是网格的自治性。但是网格资源也必须接受网格的统一管理，否则不同的资源就无法建立相互之间的联系，无法实现共享和互操作，无法作为一个整体为更多的用户提供方便的服务。因此网格的管理具有多重性，一方面它允许网格资源的拥有者对网格资源具有自主性的管理，另一方面又要求网格资源必须接受网格的统一管理。

2．3 资源的概念

网格资源是网格中所有可以被用户请求使用的实体的总称，是所有网格节点履行网格协议要求向网格提供的资源总和，网格就是资源和服务的集合。

服务表示用于提供访问某种资源的能力，例如读取文件和创建进程等。服务通常以接口的形式提供，在网格计算环境中通过协议进行请求。资源的共享通过服务实现。

网格资源种类繁多，功能各异，可以从不同的角度对资源进行划分[9]：

1、根据资源的可移动性，资源可以分为可移动资源（如数据、程序，它们可从一个位置移动到另外一个位置）和不可移动资源（如硬件、设备）。

2、根据资源是否可重复使用，资源可以分为可重复使用资源（如CPU、存储器）和不可重复使用资源（如计算周期、网络带宽）

3、根据资源可复制特性，资源可以分为可复制资源（如数据、应用程序、服务）和不可复制资源（硬件）。

从物理的角度看，网格中资源种类繁多而且多为异构的，包括挂接在网格上的各种计算机软件、计算机硬件、存储系统、分布式文件系统、集群系统以及信道传输能力等。从逻辑的角度看，网格资源是这些具体的物理设备所能提供的计算能力、存储能力、信息服务等。物理资源是逻辑资源的载体。

网格中的资源具有如下特点[9]：

1、异构性

网格中的资源种类繁多，功能各异，访问接口也各不相同。例如，有的机器运行Windows操作系统，有的机器运行Linux操作系统，即使运行同一个操作系统，其运行的操作系统版本也不一样。

2、动态性

网格中的资源可以自由地随时加入和离开网格系统，网格资源的可获得性是随时间的变化而动态变化的，网格资源的负载也是动态变化的。

3、自治性

网格资源有自己的本地管理机构或处在本地管理机构的管理之下，网格资源或强或弱的有本地自治能力。

4、二分特性

网格资源最终都是由具体的资源拥有者提供的，除了一部分专用的网格资源是专门提供给网格用户使用之外，大部分的资源都同时作为网格用户可以使用的网格资源和资源拥有者自己使用的本地资源。

2．4 网格体系结构

2．4．1 五层沙漏结构

网格体系结构就是关于如何建造网格的技术。它给出了网格的基本组成与功能，描述了网格各组成部分的关系以及它们集成的方式或方法，刻画了支持网格有效运转的机制。到目前为止，比较重要的体系结构有两个：一个是 Ian Foster 等人在早些时候提出来的五层沙漏结构；另一个是在考虑了 Web 技术的发展与影响后，Ian Foster 等人结合 Web Services 提出的开放网格服务结构 OGSA（Open Grid Service Architecture）。本小节介绍五层沙漏结构，下一小节介绍开放网格服务结构。

五层沙漏结构的主要思想是以“协议”为中心，每层的功能定义由内部协议和外部协议组成。五层沙漏结构侧重于定性的描述而不是具体的协议定义，因此很容易从整体上进行理解，是一种影响十分广泛的结构。网格的五层沙漏形状结构如图 2.1 所示。

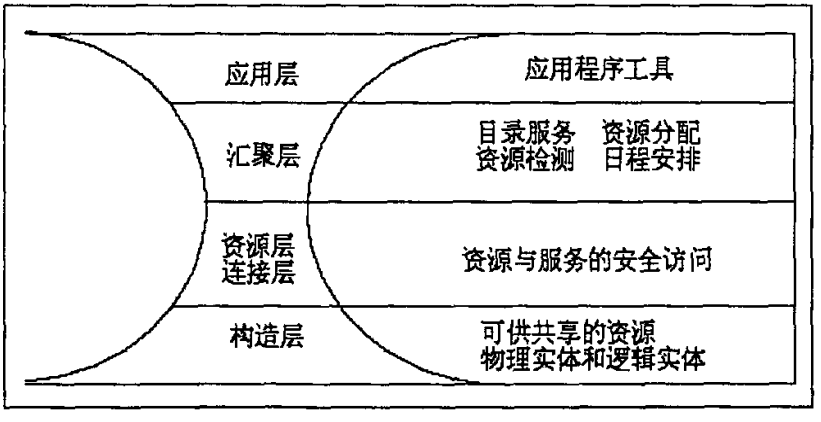

图 2.1 五层沙漏结构

1、构造层包括计算设备、存储设备、目录、分布式文件系统、分布式计算池、计算机集群、网络、传感器以及贵重仪器等。该层的功能是向上提供网格中可共享使用的资源接口，网格通过支持设备共享的协议来访问本地设备。

2、连接层主要为下层的物理资源提供安全的数据通信能力，使孤立的单个资源之间建立联系。构造层的各种资源间的数据交换都在这一层的控制下实现的。

3、资源层反映的是局部资源的抽象特征，资源层建立在连接层的通信与认证协议基础之上，是对个人资源安全共享操作的谈判、启动、监视、控制、记账和支付定义协议、API 和 SDK。

4、汇聚层的作用是将资源层提交的受控资源汇聚在一起，供应用程序共享使用。该层提供目录服务、资源分配、负载控制、账户管理等功能。

5、应用层包括用户代码和网格调用两部分。它关心的是有什么样的资源可以提供给网格用户，解决不同网格用户的具体问题。

五层结构的一个重要特点就是其沙漏形状，这种形状可以从两方面来理解。首先网格环境中对不同层的服务需求量是不同的，应用层是分布在各个客户端的，基本不会发生争用，汇聚层只有在多个资源协调时使用，争用的可能性较小。构造层分布在各个物理资源，也基本不会发生争用，而资源层和连接层是所有任务分配和使用资源时都要调用的部分，是最核心的部分，成为此结构中的瓶颈。同时因为资源层和连接层既要能够实现上层各种协议向自身的映射，同时又要实现自身向下层其它各种协议的映射，在所有支持网格计算的地点都应该得到支持，因此核心协议与其它层次协议相比数量应该比较少，以便于实现和移植。

2．4．2 开放网格服务结构

开放网格服务结构 OGSA 是 Global Grid Forum 4 的重要标准建议，是继五层沙漏结构之后最重要、也是最新的一种网格体系结构。目前的大多数的研究项目和开发都已经从五层沙漏结构转移到基 OGSA 架构。

OGSA 被称为是下一代的网格体系结构，它是在原来“五层沙漏结构”的基础上，结合最新的 Web Service 技术提出来的。OGSA 包括两大关键技术：网格技术和 Web Service 技术，把原来按照两条路线进行的研究活动归纳到一条主线上来。以服务为中心是 OGSA 的基本思想，在 OGSA 中一切都是服务[10]，主要突出从网格用户的角度看上去的网格系统结构是什么样子。

在 OGSA 框架中，将一切都抽象为服务，包括计算机、程序、数据、仪器设备等。这种观念，有利于通过统一的标准接口来管理和使用网格。在 OGSA 中，服务的概念更广泛，包括各种计算资源、存储资源、网络、程序和数据库等。简而言之，一切都是服务。将一切都抽象为服务有利于通过统一的标准接口来管理

和共享网格上功能各异的资源。服务可以实现资源共享与资源管理、任务调度、网格安全等，还可以访问网格上的各种分布式资源。在 OGSA 中，可以基于简单的基本服务，形成更复杂、更高级、更抽象的服务。

OGSA 架构由四个主要的层构成，如图 2.2 所示。从下到上依次为：资源(物理资源和逻辑资源)，Web 服务以及定义网格服务的 OGS 工扩展、基于 OGSA 架构的服务、网格应用程序。

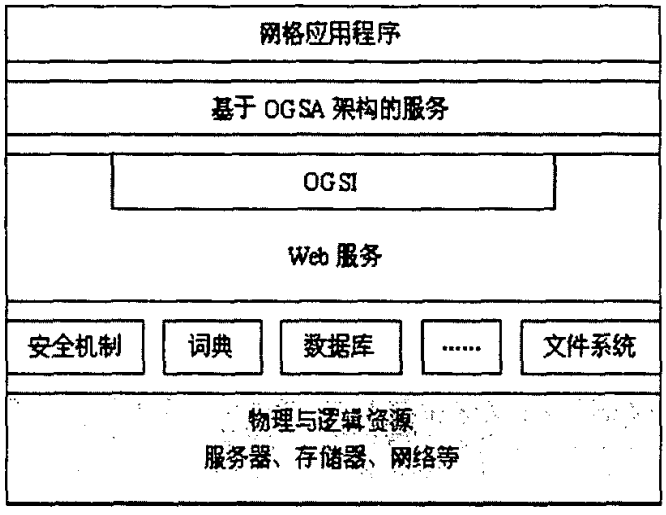

图 2.2 OGSA 架构

1、物理和逻辑资源层

资源的概念是 OGSA 以及通常意义上的网格计算的中心部分。物理资源通常包括处理器、存储器和网络。物理资源之上是逻辑资源。它们通过虚拟化和聚合物理层的资源来提供额外的功能。通用的中间件，例如文件系统、数据库管理员、目录和工作流管理人员，在物理网格之上提供这些抽象服务。

2、Web 服务层

这里有一条重要的 OGSA 原则:所有网格资源（逻辑的与物理的）都被建模为服务。OGSI 规范定义了网格服务并建立在标准 Web 服务技术之上。OGSI 利用诸如 XML 与 Web 服务描述语言（Web Services Description Language, WSDL）这样的 Web 服务机制，为所有网格资源指定标准的接口、行为与交互。OGSI 进一步扩展了 Web 服务的定义，提供了动态的、有状态的和可管理的 Web 服务的能力。

3、基于 OGSA 架构的网格服务层

Web 服务层及其 OGSI 扩展为上一层提供了基础设施一基于架构的网格服务。

GGF 目前正在致力于在诸如程序执行、数据服务和核心服务等领域中定义基于网格架构的服务。随着这些新架构的服务开始出现，OGSA 将变成更加有用的面向服务的架构。

4、网格应用程序层

随着时间的推移，一组丰富的基于网格架构的服务不断被开发出来，使用一个或多个基于网格架构的服务的新网格应用程序亦将出现。这些应用程序构成了 OGSA 架构的第四层。

3 计算网格作业调度的数学模型

3．1 网格作业调度概述

首先需要指出的是，在科技论文中存在“任务调度”和“作业调度”两种说法。为了消除岐义，本文统一用“作业”来表示。

网格上的计算资源是一种需要用户提供代码使用的资源，通过执行用户代码，处理用户数据给用户提供计算周期。大多数情况下，用户都是以提交作业的方式使用资源。作业管理是负责管理网格作业全周期的模块。作业是用户代码、数据及相应资源描述信息的集合[9]。网格中的计算资源经过抽象之后，用户看到的计算资源只是逻辑资源。用户不需要指定运行作业的物理位置，通常在提交作业时，只需描述作业对计算资源的需求情况，例如要求几个 CPU 并行计算，内存不小于 512MB,计算速度不小于 10^9FLOPS，应用软件等。

通常，我们一般将集群计算系统的作业管理系统称为“局部调度器”，而把网格作业调度系统称为“全局调度器”。全局调度器与局部调度器的区别是局部调度器仅仅管理单一的节点或集群，并通常拥有资源；而网格调度器负责资源发现，资源调度（资源分配和作业调度）和作业的执行管理，这通常要跨越多个自治域。本文以下如果没有特别说明，调度器指网格作业管理系统。

目前，网格系统中的调度器可以根据表 3.1 进行分类[11][12]：

表 3.1 调度器的分类

分类指标	具体分类
调度器的组织	集中、分层、分散
调度策略	固定的、可扩展的
状态估计	有预测（启发式、机器学习、经济模型）、无预测
重调度	阶段性、事件驱动

调度器的组织分为集中式、分层结构和分布式的三种。集中式调度器易于实施和管理，易于实现资源的协同分配，缺点是维护代价高，难于实现容错。分层调度器易于实现扩展和容错，易于资源的协同分配，但不支持资源地域自治和多种调度策略。分布式调度器易于扩展和容错，支持资源自治和多种调度策略结合使用，但不易于管理和资源协同分配。

调度策略可分为固定的和可扩展的，固定式调度又可以分为面向系统的和面

向应用程序的，前者的调度目标是最大化系统吞吐量，后者的调度目标是最优化应用程序完成时间。可扩展调度策略又分为 Ad-hoc 类型和结构化类型，前者执行固定调度策略，但允许系统外部实体修改调度结果，而后者允许外部代理修改系统作业调度策略。

从系统状态估计的支持上，可将网格作业调度分成有状态预测和无状态预测两类，预测方法包括启发式方法、基于经济模型的方法和机器学习方法。无预测的方法包括启发式方法和概率分布方法。

为了提高资源利用率、作业吞吐量，进行负载平衡，有些调度器支持作业的重新调度，重新调度策略分为批处理式和在线式，批处理重调度资源利用率高，在线式重调度响应速度快。

联机调度(动态调度)模型和批处理调度(静态调度)模型。联机模型启发式算法是当作业到达调度器就将它调度到机器上；批处理调度模型启发式算法并不是在作业到达时就映射，而是先将这些作业放在队列中，在预调度时间由映射事件调用映射。在联机模型中，每个作业在映射和调度中只考虑一次；在批处理调度模型中，调度器在每一次映射事件中都要考虑每一个作业的映射和调度。

Apples 采用分布式调度器，启发式状态估计，在线式重调度和固定的面向应用的调度策略。Condor 采用协作式集中调度器。Globus 和 Legion 采用分层式 Ad-hoc 扩展调度策略。Nimrod-G 采用分散式经济预测、事件驱动固定面向应用程序调度策略。

可见，各种调度策略各有优缺点，在各种网格系统中都有不同的组合使用。在网格环境中，资源的多样性和不稳定性较传统分布式计算环境更明显，用户的应用需求也更加多种多样，需要调度算法能适应多种网格资源管理结构，支持资源的动态变化，具有经验积累和预测性，支持作业的重新调度。所以，一般采用动态调度与静态调度相混合的调度策略，特别是能根据系统的情况随时调整分配方案的启发式调度方法是最适合的。

网格作业调度的目标与分布式计算系统、集群系统的作业调度目标是相似的，在这些系统中，衡量调度性能的指标包括资源利用率[13][14]、算法复杂度[15]和完成时间[16]。

3．2 数学模型的描述

为了方便定义网格作业调度的数学模型，首先给出定义模型中使用的参数，本文后面的章节中也使用了本节给出的参数。

假设，有 N 个作业，M 个计算节点（或者称为计算资源、机器、集群，本文统一用计算节点来表示）。

$SubmitTime_i$：作业 i 提交给调度器队列的时间，或作业到达调度器的时间。

$StartTime_i$：作业 i 在计算节点上开始执行的时间。

$EndTime_i$：作业 i 运行结束的时间。

$RunTime_i$：作业 i 在计算节点上的运行时间。

$ResponseTime_i$：作业 i 的响应时间。其中，$i=1,...,N$

$Available[j]$：计算节点 j 空闲的时间，表示计算节点 j 完成当前分配的作业后可用的时间。其中，$j=1,...,M$。

$RunTime[i,j]$：$N\times M$ 的二维矩阵，表示作业 i 在计算节点 j 上的运行时间。

$EndTime[i,j]$：$N\times M$ 的二维矩阵，表示作业 i 在计算节点 j 上的完成时间。

$Makespan$：作业调度的跨度，跨度的含义是从第一个作业开始执行到最后一个作业完成之间的时间。用数学公式表示如下：

$$Makespan=\max(EndTime_i) \qquad i=1,...,N$$

实际上，跨度是为了衡量系统的吞吐量而提出来的，并不衡量单个作业的服务质量。我们知道吞吐量指单位时间内完成的工作量。针对网格作业调度，吞吐量具体指单位时间内完成的作业数目，用公式表示如下：

$$\text{吞吐量}=\frac{\text{作业数目}}{\text{最后一个作业完成时间}}=\frac{N}{Makespan}$$

如果一个调度算法如果能获得较小的跨度，那么就意味着系统的吞吐量高。

图 3.1 给出了上述时间参数之间的关系图。

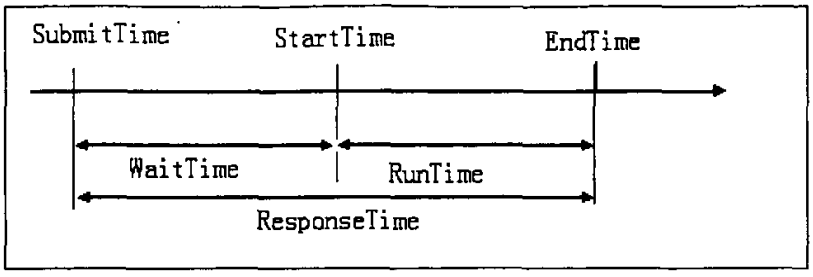

图 3.1 主要时间参数关系图

根据上述定义，可用得到如下关系式：

$$EndTime_i = StartTime_i + RunTime_i$$

$$ResponseTime_i = EndTime_i - SubmitTime_i$$

如果作业 i 在计算节点 j 上运行，那么在作业 i 运行结束后，有如下关系式：

$$Available[j] = EndTime_i = EndTime[i,j] = StartTime_i + RunTime[i,j]$$

网格作业调度的的基本问题：假设已经知道每个作业在各个计算节点上的运行时间，如何将 N 个相互独立的作业分配到 M 个异构可用的计算节点上，使得作业运行的跨度最小。

需要指出，上述数学模型是从理论角度来考虑作业调度，模型中使用的参数（当前网格的资源状态信息）可以通过网格信息查询服务获得。实际上网格环境中资源动态变化，很难预先准确的知道一个作业在计算节点上的运行时间，通常我们可以用如下公式来粗略的估计作业的运行时间，得到 $RunTime[i,j]$ 矩阵：

$$RunTime[i,j]=\frac{\text{作业}i\text{运行需要执行的指令条数}}{\text{计算节点}j\text{的计算速度}}=\frac{Instructions}{CPUspeed}$$

其中，$CPUspeed$ 通常可以用 MIPS（Million Instructions Per Second，每秒平均百万条指令数）速率[17]来衡量。

使用类似流水线分析中广泛采用的时空图[17]，可以直观的表达作业调度算法运行结果。此外，时空图还可以用来帮助我们计算作业调度算法的各项性能指标。6.1 节中给出了性能指标的具体计算公式。

3.3 数学模型的改进

上一节给出了网格作业调度的数学模型只考虑了作业运行时间，作业 i 在计算节点 j 上的完成时间 $EndTime_i$：

$$EndTime_i = EndTime[i,j] = StartTime_i + RunTime[i,j]$$

上述模型并没有考虑网格计算环境中的很多实际问题。本小节对上一节的数学模型进行改进。

实际的作业调度过程中，调度算法将作业映射到计算节点后，计算节点进行计算，得到作业的执行结果后，计算节点还要将计算结果返回给调度器，这意味着作业的执行结果要通过网络传输给调度器。由于网格环境的异构性，网络中存在不同的数据链路（如调制解调器、以太网、ATM 和光纤网等），网络带宽数量级从 Kbps 到 Gbps 不等，并且传输的数据量大小不同。因此，网络延迟在作业调度中不能忽略。衡量网络延迟的公式[17]如下：

网络总延迟＝发送方开销＋飞行时间＋传输时间＋接受方开销

$$=\text{发送方开销}+\frac{\text{传输的距离}}{\text{信号在链路中的传播速度}}+\frac{\text{传输的数据量}}{\text{网络带宽}}+\text{接收方开销}$$

其中，发送方开销指处理器把数据放到网络的时间，这包括软件和硬件所花费的时间。飞行时间指数据的第一位到达接收方所花费的时间，它包括由于网络中转发或其他硬件所引起的延迟。传输时间指数据通过网络的时间。接收方开销指处理器把数据从网络中取出来的时间，这包括软件和硬件所花费的时间。

例如，TeraGrid 平台已经在圣迭戈超级计算中心(SDSC)和美国国家超级计算应用中心（NCSA）之间建立了一个 40GBit/sec 的专用链路。期望的等待时间在 100 毫秒的范围内。假设数据传输可以使用全部带宽，粗略的计算表明要在 TeraGrid 中传输 1GByte 的数据，大约有三分之一的时间花在网络等待时间上。

考虑网络延迟因素，我们定义 TFF_i（File Transfer Time，文件传输时间），表示作业 i 执行完毕后，从计算节点将结果返回给调度器的时间。

考虑网络延迟因素后，对作业 i 的完成时间 $EndTime_i$ 进行修正：

$$EndTime_i = EndTime[i,j] = StartTime_i + RunTime[i,j] + FTT[i,j]$$

作业 i 的完成时间等于作业 i 的开始运行时间、运行时间与传输时间之和。

4 一种基于贪心策略的调度算法

4.1 典型的作业调度算法

作业调度算法的目标是使跨度最小，但最优调度算法属于NP－完全问题[18]，NP-完全问题是指尚未找到具有多项式时间复杂性算法的问题。寻找最优调度算法所需的执行时间均随着作业数及计算节点数目的增加而呈指数级增长，因此，只有在极少数情况下（对作业和计算节点模型加以严格限定）才存在有效的最优调度算法。为获得可行的调度方案，我们经常采用一些非最优算法进行调度，虽然不能保证获得最优解，但可以快速获得最优调度的近似解或可行解。通常采用的有代表性的近似方法有基于图论的分配算法、数学规划法、专家系统方法以及启发式算法等。前三种方法都基于一组已知的资源配置和作业要求，根据具体的算法或规则求出作业分配方案，求解过程较为复杂，且不适应网格环境下资源动态改变频繁的情况。

启发式算法指：系统事先不具备解决问题的知识，只是在环境中试着解决问题，并利用对一些间接影响系统的、易于计算或监控的特殊参数观察由此产生的环境变化，获得环境反馈信息，然后再根据这些反馈信息调整自己解决问题的模型，自到达到某种标准(时间或精度)的近似最优解。由于启发式方法可以快速、有效地得到近似最优调度方案，因此在分布环境作业调度中被广泛使用。但启发式算法不能保证得到问题的最优解，并且没有性能保证。

典型的网格作业调度算法有 DFPLTF(Dynamic FPLTF)[19],Min-Min,Min-Max[20][21][22], Sufferage[23], 以及 Min-Min 的改进算法 QoS guided Min-Min [24]。这些算法都是启发式算法。

4.1.1 Min-Min 和 Min-Max 算法

Min-Min 算法的主要思想是：将最快完成的作业分配到能够最快完成它的节点上。

首先将需要调度的作业组成一个作业集合，计算出作业集合中每个作业的最短完成时间。调度时，从作业集合中选择有最短执行时间的那个作业分配到相应的节点上，同时从作业集合中删除该作业，更新相应的参数。然后重复上述过程

一直到作业集合为空。

算法的伪代码描述如下：

For 作业集合 T 中所有的作业 job_i

For 所有计算节点 $node_j$

$c_{ij} = e_{ij} + r_j$

do until T 为空

计算 T 中每个作业最短完成时间，以及相应的节点

找到具有最早完成时间的作业 job_k，相应的节点为 $node_l$

将作业 job_k 分配到节点 $node_l$

从 T 中删除作业 job_k

更新 r_l

更新所有的 c_{il}

enddo

算法中的 e_{ij} 是期望执行时间，表示作业 job_i 在节点 $node_j$ 上没有负载的情况下执行时间。c_{ij} 是期望完成时间，表示节点 $node_j$ 执行完作业 job_i 的时间。r_j 表示节点 $node_j$ 执行作业的期望开始时间。该算法期望获得整体作业的最早完成时间，以最快的速度减少作业集合中的作业数量，缩短等待作业的队列长度。

Min-max 算法与此类似，第一步同样计算出每个作业的最小执行时间，但是在第二步中，取最小时间集合中具有最大执行时间的那个作业，投入到相应节点上运行，希望尽可能早地完成长作业。

4．1．2 Sufferage 算法

Sufferage 算法的主要思想是：一个节点被分配给这样一个作业，如果该作业不分配到该节点上，将会蒙受最大的损失。

算法为每个作业定义一个代表其优先权的 Sufferage 值，该值等于作业的次好完成时间减去最好完成时间的差值（最好、次好完成时间均针对每个节点计算得出，Sufferage 实际上代表了损失度，损失度是在作业相互比较中体现出来的）。调度时先选取对该作业具有最好完成时间的节点，计算 Sufferage 值，然后比较

该节点上已经分配了的作业，如果该作业 Sufferage 值高于已经分配作业的 Sufferage 值，那么取消已经分配的作业，而将该作业分配给节点。重复以上的过程，一直到作业集合为空。算法的目标是让所有作业的损失度总和最小。

算法的伪代码描述如下：

For 作业集合 T 中所有的作业 job_i

For 所有计算节点 $node_j$

$c_{ij} = e_{ij} + r_j$

do until T 中所有的作业都映射

For 作业集合 T 中所有的作业 job_k

寻找 job_k 具有最早完成时间的 $node_l$

Sufferage 值=次好完成时间－最好完成时间

If $node_l$ 没有分配

分配 job_k 给 $node_l$，从 T 中删除 job_k，标记 $node_l$ 为已经分配

Else

If 已经分配给 $node_l$ 的 job_i 的 sufferage 值小于 job_k 的 sufferage 值

取消 job_i 的分配，把 job_i 放回 T 中

将 job_k 分配给 $node_l$，从 T 中删除 job_k

endfor

根据分配给节点的作业，更新向量 r

更新 c 矩阵

enddo

4.1.3 QoS guided Min-Min 算法

典型的 Min-Min 算法没有考虑作业对网络资源的需求。有些作业在运行的过程中需要在各个计算节点之间快速地交换数据，因此，这些作业对网络带宽就有比较高的要求，需要计算节点能够提供较高的 QoS（Quality of Service，服务质量）保证；而另一些作业没有这样的要求，网络带宽较低的计算节点就能满足它们的需要。在 guided Min-Min 算法中，QoS 具体指网络带宽。

由于 Min-Min 算法没有考虑 QoS，在作业调度的过程中就可能出以下这种低效率的现象：低 QoS 需求的作业占用了高 QoS 保证的计算节点，而高 QoS 需求的作业只能排在低 QoS 作业的后面，等待其完成后才能被执行。与此同时，低 QoS 保证的计算节点却处于空闲状态，没有作业运行。这样一来就造成了资源的浪费，降低了执行效率。

为了克服上述缺点，QoS guided Min-Min 算法在原有的 Min-Min 算法基础上进行修改，考虑 QoS 后，能充分提高计算资源的利用率。

4．1．4 其他算法

GA：遗传算法(Genetic Algorithm)[25][26]是一类借鉴生物界的进化规律 （适者生存、优胜劣汰遗传机制）演化而来的随机化搜索方法。它是由美国的 J．Holland 教授在 1975 年首先提出，目前，遗传算法已成为进化计算研究的一个重要分支。与传统优化方法相比，遗传算法的优点是：群体搜索、不需要目标函数的导数、概率转移准则。算法具有内在的隐并行性和更好的全局寻优能力，采用概率化的寻优方法，能自动获取和指导优化的搜索空间，自适应地调整搜索方向，不需要确定的规则。遗传算法的这些性质，已被人们广泛地应用于组合优化、机器学习、信号处理、自适应控制和人工生命等领域。它是现代有关智能计算中的关键技术之一，是用来搜索大规模解空间的一种常用方法。遗传算法首先假定问题的潜在解决方案可以表示为一串参数的组合，称为染色体。一种方法是可以任意选择一定数量的染色体；另一种方法是将其他算法的输出结果作为一条染色体，再加上一定数量的任意解，然后所有染色体基于它们的值排队，通常使用轮转法选择合适的染色体，这能保证最合适的个体得到保留并作为副本。两个被选中的染色体可以进行交叉配对复制产生后代。染色体复制时会发生突变从而产生新个体。经过上述过程的多次反复，产生的新染色体会逐渐接近最优解。

A*：A*[27][28]搜索是一种开始于根节点的树搜索，该树节点通常是一个空解。正如树的生长，中间节点表示部分解，叶节点表示最终解。每个节点都有一个代价函数，并且具有最小代价函数的节点被他的子节点代替。任何时候当增加节点时，树要删去具有最大代价函数的节点从而重新调整数。持续进行该过程，直到达到全映射。

4．2 典型的算法比较

文献[29]对典型的作业调度启发式算法进行了性能比较。得到如下结论：Min-Min，GA 和 A*算法有较好的性能。这三个调度算法的调度完成时间的差距通常 10%以内，由 Min-Min 算法初始化染色体种群的遗传算法通常要比 Min-Min 算法要好几个百分点，A*算法则在不同的情况下性能有所不同，有时比 Min-Min 算法和遗传算法的性能好，有时则不如 Min-Min 算法和遗传算法的性能好。在这三个算法中，Min-Min 算法是最快的，遗传算法较慢，而 A*算法是最慢的。

4．3 基于 Replica 的调度算法

4．3．1 基本思想

4．1 节中介绍的典型作业调度算法有一个共同的特点：一个作业只在一个计算节点上运行。目前，研究人员已经提出了几种基于副本（Replica）的调度算法，其基本思想如下：

一个作业可能在多个计算节点上同时运行，即该作业有多个副本在性能不同的计算节点上执行。其中必有一个计算节点最先运行完作业，得到作业的运行结果后，其他计算节点没有必要再继续运行该作业了，因此立刻撤消该作业其他副本运行，以减少资源浪费。其优点是可以缩短作业的运行时间，但它也要付出一定的代价：由于一个作业在多个计算节点上同时运行，而其中，只有一个作业的运行是有效的，其他计算节点的计算资源被浪费了，作了一些“无用功”。

实际上，基于 Replica 的调度算法使多个计算节点同时运行一个作业，以此来缩短作业的运行时间，利用了空间并行性来缩短作业的运行时间

4．3．2 基于 Replica 的典型调度算法

Noriyuki Fujimoto 与 Kenichi Hagihara 提出了一种基于 Replica 的调度算法[30][31]，该算法在传统的调度算法 RR（Round-Robin，轮转）基础上引入了 Replica。该算法不需要资源的预测信息，例如，不需要事先知道计算节点的计算能力，作业在计算节点上的运行时间等。该算法是一种近似算法，即无论计算节点的计算能力如何变化，算法总能保证一定的近似率。

Daniel Paranhos da Silva 等人在传统的 WQ（Work Queue）[32] 算法的基础上采用了 Replica 思想，提出了 WQR(Work Queue Replica)算法[19]。

基于 RR（Round-Robin，轮转）的 Replica 调度算法将作业组成一个环形队列，每个空闲的计算节点选择环形队列（存放尚未完成的作业）中的头部作业运行。该调度算法没有考虑计算节点与作业之间的匹配，其缺点是一个资源需求小的作业可能运行在一个资源充足计算节点上，而一个资源需求大的作业却运行在一个资源小的计算节点上。更为不利的情况是：如果一个作业的副本运行在多个资源充足的计算节点上，得到作业的运行结果后，撤销其他副本的运行，许多资源充足的计算节点做了很多的“无用功”，造成大量的资源被白白地浪费掉。

下面通过一个例子来说明上述问题。

假设有 7 个作业｛J1，J2，J3，J4，J5，J6，J7｝，4 个计算节点｛S1，S2，S3，S4｝，算法的运行结果的时空图如图 4.1 所示。

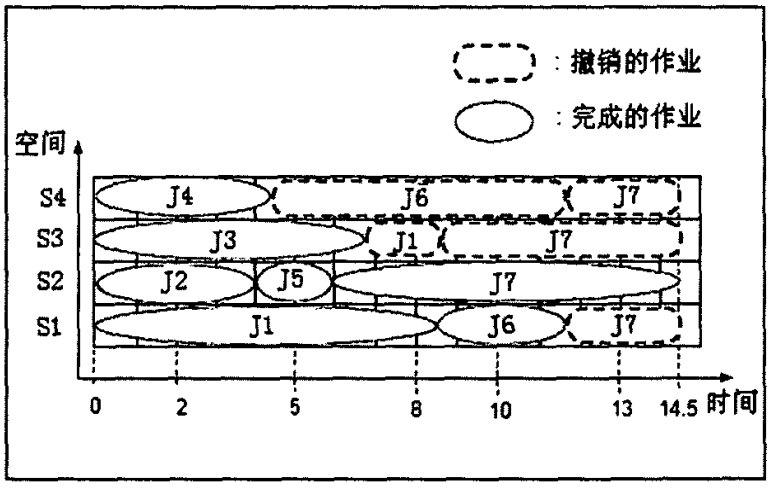

图 4.1　作业调度的时空图

从作业调度的时空图，我们可以看出:作业 J7 总共有 4 个副本在运行，其中计算节点 S2 上的作业 J7 副本从 6 时刻开始执行，并且在 14.5 时刻最先完成，撤销计算节点 S1、S3 和 S4 上作业 J7 副本的运行，造成计算节点 S1、S3 和 S4 上运行作业 J7 副本的计算资源被浪费了。

4.4 基于贪心策略的调度算法

4.4.1 贪心策略

贪心法[33][34]是一种重要的算法设计方法。通常，对于一个问题的最优解只能用穷举法得到时，用贪心法是寻找问题次优解的较好算法。贪心法是一种改进了的分级处理方法。用贪心法设计算法的特点是一步一步地进行，根据某个优化测度（可能是目标函数，也可能不是目标函数），每一步都要保证能获得局部最优解。每一步只考虑一个数据，它的选取应满足局部优化条件。若下一个数据与部分最优解连在一起不再是可行解时，就不把该数据添加到部分解中，直到把所有数据枚举完，或者不能再添加为止。这种能够得到某种度量意义下的最优解的分级处理方法称为贪心法。

文献[35]中提出了一种贪心调度算法，其贪心策略是对每个作业选择最小运行时间的计算节点运行。

本文在 Replica 思想基础上，设计了一种贪心策略，目的是减少资源浪费。为了避免计算节点与作业之间匹配的盲目性，提出的贪心策略采用“费用”作为目标函数，其中“费用” *cost* 的定义如下：

$$cost = RunTime \times RequestedResources$$

即费用＝运行时间×请求的资源大小，其物理意义表示作业占用的资源时空区大小。该目标函数不但考虑了作业的运行时间，而且考虑了作业占用的资源大小。请求的资源大小指计算节点的计算能力，可以用 MIPS(Million Instructions Per Second，每秒平均百万条指令数）速率来衡量或微处理器的主频（单位：GHZ）来衡量。

4.4.2 算法的描述

假设已知 N 个作业，M 个计算节点。$N \times M$ 二维矩阵 $RunTime[i,j]$ 和 $RequestedResources[i,j]$ 分别表示作业 i 在计算节点 j 上运行所需要的运行时间和请求使用的资源大小。本文把提出的算法称为Greedy_Replica算法。算法以3.2节提出的数学模型为基础，并不考虑网格环境中的不确定因素和随机性，但该算法也很容易扩展到网格环境中。

算法的类C语言描述如下：

```
Void Greedy_Replica(RunTime[N][M], RequestedResources[N][M])
{
  JobQueue  JQ1;  // JQ1 表示尚未分配出去的作业队列
  JobQueue  JQ2;  // JQ2 表示已经分配出去但尚未执行完成的作业队列
  ServerQueue SQ;  // SQ 表示空闲计算节点队列
  Init(JQ1);  // JQ1 初始化，N 个作业进入队列 JQ1
  Init(JQ2);  // JQ2 初始化为空
  Init(SQ);   // SQ 初始化
  for (i=1;i<=M;i++) //M 个计算节点，每个计算节点选择花费最大的作业运行
  {
   server=Extract_First_Available(SQ); //从 SQ 中取出最先空闲计算节点 server
   job=Max_Cost(server,JQ1);// 空闲计算节点 server 选择花费最大的作业 job
   Assign(job,server);// 将作业 job 分配给计算节点 server
   DelQueue(JQ1,job);//将作业 job 从 JQ1 中删除
   EnQueue(JQ2,job);//将作业 job 放入 JQ2 中
  }
 while (QueueEmpty(JQ1)!=NULL)  //一直循环，直到 JQ1 变为空为止
  {
   server= Extract_First_Available(SQ);//从 SQ 中取出最先空闲计算节点 server
   completedjob=server.job; //空闲计算节点 server 执行完作业 server.job
   DelQueue(JQ2,completedjob); //将作业 completedjob 从 JQ2 中删除
   job=Max_Cost(server,JQ1); // server 从 JQ1 中选择花费最大的作业 job
   Assign(job,server);  //将作业 job 分配给计算节点 server
   DelQueue(JQ1,job);  //将作业 job 从 JQ1 中删除
   EnQueue(JQ2,job);  // 将作业 job 放入 JQ2 中
```

```
  }
while (QueueEmpty(JQ2)!=NULL) //一直循环，直到 JQ2 变为空为止
 {
  server= Extract_First_Available(SQ);//从 SQ 中取出最先空闲计算节点
server
  completedjob=server.job;  //空闲计算节点 server 执行完作业 server.job
  DelQueue(JQ2,completedjob);  //将作业 completedjob 从 JQ2 中删除
  Invalidate_Replica(completedjob);//撤销 completedjob 其他副本的运行
  if (QueueEmpty(JQ2)!=NULL)
   {
    server=Extract_First_Available(SQ); //从 SQ 中取出最先空闲计算节点
server
    job=Min_Cost(server,JQ2); // 空闲计算节点 server 选择花费最大的作业
job
    Assign(job,server); //将作业 job 分配给计算节点 server
   }
  }
}
```

算法的执行过程如下：

第一步：初始化。完成尚未分配出去的作业队列 JQ1、已经分配但尚未执行完成的作业队列 JQ2 和空闲计算节点队列 SQ 的初始化工作。初始化后，JQ1 中存放了 N 个作业，JQ2 为空队列，SQ 中 M 个计算节点的 $Availbale[j]$ 值初始化为 0，表示 M 个计算节点当前均空闲可用。

第二步：第一个 for 循环语句。每个空闲计算节点 server 从 JQ1 中选择花费最大的作业 job，将作业 job 分配给 server，同时将作业 job 从 JQ1 中删除（作业 job 已经分配出去了），并将作业 job 放入 JQ2 中（作业 job 已经分配了，但还没有执行完成）。for 循环语句执行完后，M 个计算节点每个都选择了一个作业运行，但此时系统中还有 $N-M$ 个作业还没有分配出去。

第三步：第一个 while 循环语句，将 $N-M$ 个没有分配出去的作业分配给空闲的计算节点。server= Extract_First_Available(SQ)，表示计算节点 server

运行完其作业 server.job，成为空闲计算节点。将其完成的作业 completedjob 从 JQ2 中删除，server 继续从 JQ1 中选择其花费最大的作业 job，将作业 job 分配给 server，将作业 job 从 JQ1 中删除，并放入 JQ2 中。重复上述第三步的步骤，直到 $N-M$ 个作业都分配出去。

第四步：第二个 while 循环语句。首先需要指出，在开始执行 while 语句之前，JQ2 队列中存放了已经分配出去但还没有执行完的作业。一旦一个计算节点 server 变为空闲，就将其完成的作业从 JQ2 中删除，并撤销该完成的作业副本的运行。计算节点 server 继续从 JQ2 队列中选择花费最小的作业运行。重复第四步的步骤，直到所有的 N 个作业都执行完成。

4．4．3 算法的定性分析

本小节对提出的 Greedy_Replica 算法进行详细的解释，并着重指出两点：

1、Greedy_Replica 算法中 for 循环语句和第一个 while 循环语句中，采用了最大费用贪心策略，而第二个 while 循环语句中采用了最小费用贪心策略。这样安排的理由是：

根据提出的费用定义：$cost = RunTime \times RequestedResources$，如果请求的资源大小以 MIPS 速率来衡量，则费用的具体含义指作业运行过程中计算节点需要执行的指令条数。费用大意味着作业长；反之，费用小，意味着作业短。费用的大小反映了作业的长短。

JQ2 队列指已经分配出去但还没有执行完的作业队列，算法中采用 Replica 来加快 JQ2 队列中作业的执行速度。

for 循环语句和第一个 while 循环语句中分配出去的作业没有副本在执行，不存在撤销副本执行引起的资源浪费。每次按最大费用策略选择作业执行是为了使长作业得到优先的处理，等到第二个 while 循环语句执行时，JQ2 队列存放的是相对比较短的作业，如果空闲计算节点按照最大费用策略选择作业，一旦撤销该副本的执行会造成资源的不必要浪费，采用最小费用策略是一种“保守”策略。

总之，在不同的调度阶段，采用两种截然相反的贪心策略，前一阶段采用“最大”贪心策略，目的是使长作业得到优先的处理，剩下一些相对短的作业采用 Replica 来加快总的执行时间。在后一阶段采用“最小”贪心策略，目的是提高

资源的利用率，减少不必要的资源浪费。

2、从作业调度的数学模型角度讲，SQ 队列是基于计算节点的空闲时间 $Availbale[j]$ 的最小优先队列[36]。初始化时，$Availbale[j]=0$,其中$j=1,...,M$。最小优先队列可以有多种实现（如有序表、无序表和堆等），采用最小堆[36]这种数据结构实现效率高，提取最小元素和插入一个新元素的时间复杂度分别为 $O(1)$ 和 $O(\log M)$。在本文的实现中就采用了最小堆。Extract_First_Available(SQ)函数是提取最小优先队列中的最小元素，即 $Availbale[j]$ 值最小的计算节点。Assign(job, server)函数把作业 job 分配给计算节点 server，在具体实现过程中，要更新 server 执行完作业 job 后的空闲时间，即 $Availbale[j]=Availbale[j]+RunTime[job,j]$，并将更新后的 $Availbale[j]$ 插入到 SQ 队列中。

4．4．4 算法的一个运行实例

为了清楚地表达 Greedy_Replica 算法，本小节给出了一个实例，讲解算法的运行过程。

假设有 6 个作业 {J1，J2，J3，J4，J5，J6}，3 个计算节点 {S1，S2，S3}，其运行时间矩阵 $RunTime[i,j]$（单位：分钟）和请求的资源大小矩阵 $RequestedResources[i,j]$（单位：GHZ）分别如下：

$$RunTime[i,j]=\begin{array}{c} \\ J1\\J2\\J3\\J4\\J5\\J6\end{array}\begin{array}{c}\begin{array}{ccc}S1&S2&S3\end{array}\\\begin{bmatrix}6&8&14\\3&6&10\\2&5&6\\2&4&6\\2&4&5\\1&2&4\end{bmatrix}\end{array}$$

$$RequestedResources[i,j]=\begin{array}{c} \\ J1\\J2\\J3\\J4\\J5\\J6\end{array}\begin{array}{c}\begin{array}{ccc}S1&S2&S3\end{array}\\\begin{bmatrix}4&2&2\\4&2&1\\5&2&1\\5&2&1\\5&3&1\\3&2&1\end{bmatrix}\end{array}$$

第一步：JQ1＝ {J1，J2，J3，J4，J5，J6}

JQ2＝ {}

Available＝ {0，0，0}

第二步：for 循环运行结束后的作业运行的时空图如图 4.2 所示。

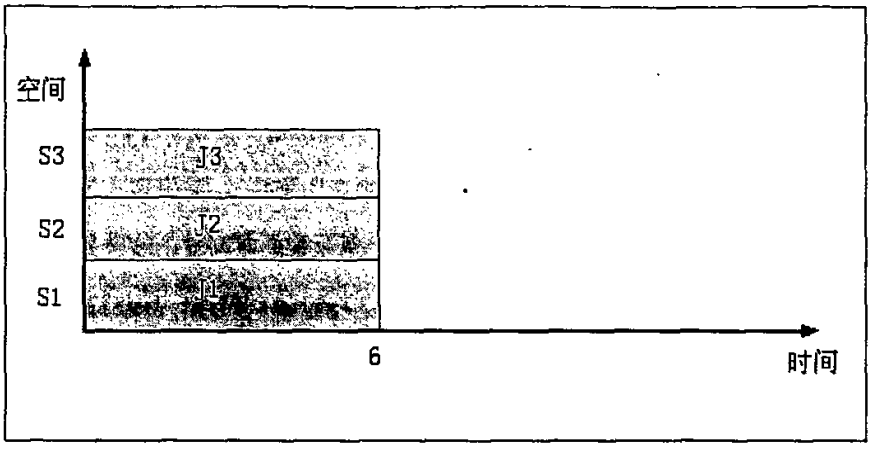

图 4.2　时刻 6 作业运行的时空图

JQ1＝｛J4，J5，J6｝

JQ2＝｛J1，J2，J3｝

Available＝｛6，6，6｝

第三步：第一个 while 循环结束后的作业运行的时空图如图 4.3 所示。

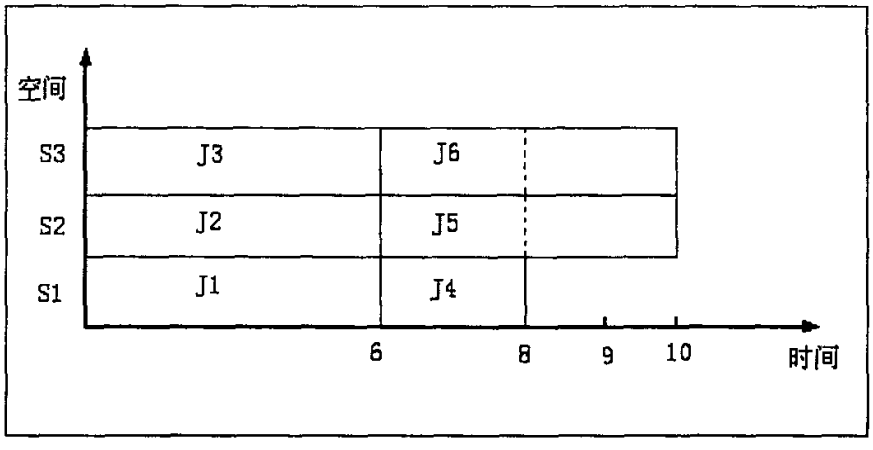

图 4.3　时刻 8 作业运行的时空图

JQ1＝｛｝

JQ2＝｛J4，J5，J6｝

Available＝｛8，10，10｝

第四步：第二个 while 循环结束后的作业运行的时空图如图 4.4 所示。

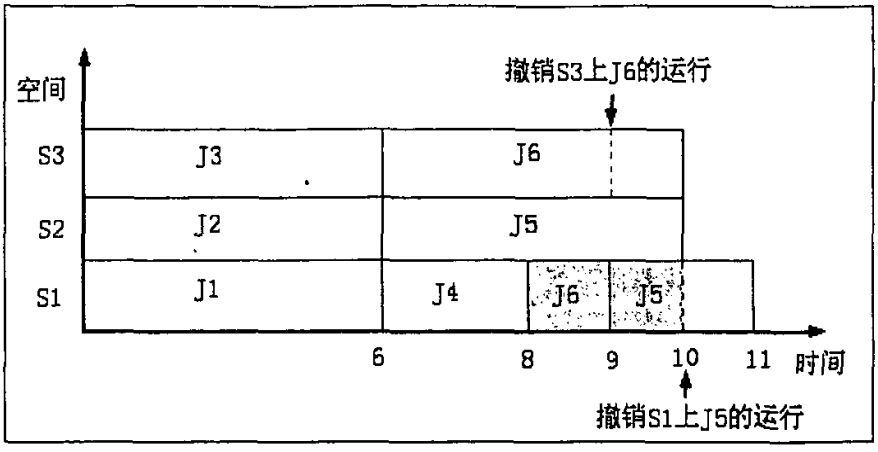

图 4.4　时刻 10 作业运行的时空图

JQ1= {}

JQ2= {}

Available= {10, 10, 9}

5 一种自适应作业调度算法

5．1 基本思想

典型的网格作业调度算法是建立在静态模型基础上，假设作业的运行时间已知，并且在作业的运行过程中，资源不会发生大的变化。但是，网格环境的异构性、动态性和分布性决定了上述模型的局限性，例如，在作业的运行过程中，资源很可能发生动态的变化，极端的情况下，作业运行了一半，由于某种原因计算节点中的计算机暂时关机，分配的作业无法正常完成。

针对上述情况，文献[37]提出了一种自适应调度算法，采用的策略是对作业重新进行调度，主要通过重新调度和作业迁移来实现，并需要建立性能预测的概率模型。

本章则针对以下情况，提出了一种自适应调度算法：

网格用户希望通过网格获得低价格、高效率、安全的服务，提交作业之后都希望快速得到系统响应，并在要求时间和价格范围内成功完成作业。网格系统管理者则希望提高系统的吞吐量，提高资源的利用率，而较少的考虑单个用户的服务质量。但是，一种调度算法很难同时满足高吞吐率、高利用率、较短的总运行时间、较短的平均等待时间和较好的公平性。通常，追求高吞吐率可能意味着平均等待时间的增大，而追求短的平均等待时间往往降低系统的吞吐率。

针对上述问题，本章提出了一种解决机制，其主要思想是：把各种性能良好、适用于不同场合的调度算法“组合”在一起，根据系统的性能状态或者用户提交的作业要求，调度器能自动的切换到不同的调度算法，这样的系统具有很好的适应性，能充分发挥各种调度算法的优势，做到“博采众长”，在系统的各项性能指标中找到一个平衡点。例如，SJF（Shortest Job First，短作业优先）的优点是作业的平均等待时间小，而 LJF（Longest Job First，长作业优先）的优点是系统的利用率高。如果能动态的调用两种算法，那么就可以在作业的平均等待时间和系统的利用率之间找到一个很好的折衷。

5．2 算法的描述

这里我们关心的是如何把各种已知的算法“组合”在一个算法中，假设已经

设计了两种网格作业调度算法，假设采用Min-Min和Sufferage这两种调度算法。这两种算法在系统开销、占用的资源、执行的效率和满足用户的需求等方面各不相同，系统能动态地选择本次调度应该采用哪种算法。

算法的类C语言描述如下：

```
typedef struct SystemParameter // 系统参数的数据结构定义
{
Policy    Algorithm ; //采用的调度算法
Time      Makespan; //调度的总的运行时间
float     Utilization; //资源利用率
...           //其他参数
}SystemParameter;
Adaptive_schedule(Min—Min, Sufferage ) //算法主体
{
   SystemParameter SP1,SP2;//定义两个变量
   SP1.Algorithm= “Min-Min”;
   SP1.Makespan=Compute_Makespan(“Min-Min”);//计算 Min-Min 算法的总的运行时间
   SP1.Utilization=Compute_Utilization(“Min-Min”); //计算 Min-Min 算法的资源利用率
   SP2.Algorithm= “Sufferage”;
   SP2.Makespan=Compute_Makespan(“Sufferage”);//计算 Sufferage 算法的总的运行时间
   SP2.Utilization=Compute_Utilization(“Sufferage”);//  计  算 Sufferage 算法的资源利用率
   Choose_Best_Algorithm(SP1, SP2);
   }
```

其中， Compute 模块根据假设采用的算法计算系统参数 SystemParameter，可以通过“虚拟”地调度作业，然后计算相应的性能参数。系统参数 SystemParameter 并不一定是一个标量，还可以是一个向量，例如{采用的算法，作业调度的跨度，平均响应时间，平均等待时间，系统的利用率，系统的负载平

衡性}。选择最优算法模块 Choose_Best_Algorithm 根据 SystemParameter 中各项性能指标，动态地选择一个最优的调度算法。

该算法给出了一种自适应调度的框架，其优点如下：

1、算法具有动态性，能根据系统参数的变化动态地调用不同的调度算法；

2、算法具有可扩展性，可以动态的添加到系统中，可以方便的验证不同算法组合的整体性能；

3、该框架的核心是选择最优算法模块，它留给算法的一个接口，但并没有给出具体的实现。这样做的好处是可以做到“抽象”－接口与实现的分离。Choose_Best_Algorithm 模块可以采用不同的设计方案。

5．3 选择最优算法模块的设计

5．3．1 系统负载平衡因子的定义

本小节我们给出了选择最优算法模块的一个参考设计。我们这里只考虑系统的负载平衡性，选择最优算法模块的目标是使系统的负载趋向于平衡。

首先给出系统负载平衡因子的定义如下：

$$\text{系统负载平衡因子} LBF=\frac{\min(Available[j])}{\max(Available[j])} \qquad j=1,...,M$$

其中，M 是计算节点的个数，$Available[j]$ 表示计算节点 j 空闲的时刻。

系统负载平衡因子的含义：N 个作业在 M 个计算节点上经过调度运行后，M 个计算节点中最先空闲的计算节点的空闲时间 $\min(Available[j])$ 与最后空闲的计算节点的空闲时间 $\max(Available[j])$ 的比值。LBF 的取值范围为[0，1]。当 $LBF=1$ 时，表示各个计算节点同时变为空闲，这样各个计算节点的负载都是一样的，当 $LBF=0$ 时，那么 $\min(Available[j])=0$，意味着存在某个计算节点没有分配作业运行，一直处于空闲状态，系统的负载极不平衡。显然，我们希望系统负载平衡因子 LBF 的值越大越好。系统参数 SystemParameter 的参数为负载平衡因子 LBF。

5．3．2 模块的设计

根据上一小节的负载平衡因子的定义，我们希望调用一个算法后，系统负载平衡因子值能变大一些，使系统的负载趋向于平衡。

算法的类 C 语言描述如下：

```
Choose_Best_Algorithm (SP1, SP2)
 {
  if (SP1.LBF< SP2.LBF)
   Call  SP2.Algorithm; //调用 Sufferage 算法
  else
   Call  SP1.Algorithm; //调用 Min－Min 算法
 }
```

Choose_Best_Algorithm 运行时，根据 Min-Min 和 Sufferage 算法带来的系统的平衡负载因子的不同，选择不同的调度算法。SP1 和 SP2 分别是 Min-Min 和 Sufferage 算法的系统参数。如果 SP1.LBF< SP2.LBF，意味着下一次作业调度采用 Sufferage 算法会使系统负载趋向于平衡（根据系统负载平衡因子的定义，值越大，系统越趋向于负载平衡），因此调用 SP2.Algorithm 算法，即 Sufferage 算法。否则，调用 SP1.Algorithm 算法，即 Min-Min 算法。

这里给出 Min-Min 和 Sufferage 算法的组合，只是作为例子来说明算法的思想，并不意味着这两种算法组合后系统的负载平衡性好。具体哪几种算法 “组合”后的实际性能最好，还需要作大量的仿真实验来验证。此外，实际的系统通常要考虑多项性能指标，本节给出的模块只考虑系统负载平衡性，如何根据多项性能指标进行调度有待进一步研究。

6 性能评价与模拟实验

6．1 性能评价的指标

6．1．1 常用的性能评价指标

在作业调度系统中，系统的所有者和系统的用户通常不是同一个人，而且他们所追求的性能目标也是不一样的。对于系统的所有者而言，他从整个系统的角度来考虑问题，希望该系统能够在一段时间内为更多的用户服务，从而有较高的吞吐量，另外，他希望系统中的资源尽量不要被浪费，保证系统要有高利用率，而同时又不会因服务质量的下降而导致失去用户。对于系统的用户而言，他更多考虑的是自己提交的作业，希望自己的作业能够尽快地被处理，其等待时间尽可能的短。在评价一个作业调度算法性能优劣的时候，可以从系统角度和用户角度两个方面来进行评价[38]。

面向系统的性能指标主要有：

1、跨度：从第一个作业提交到最后一个作业完成之间的时间。

2、吞吐量：单位时间中系统所完成的作业数。吞吐率是作业调度系统性能的主要参数和重要指标。

3、系统利用率：系统负载与其可承受负载的比值。调度算法在进行计算节点的选择时要充分考虑到结点间的负载平衡，要尽量使它满负荷地运转，不能使结点过载，也不能使结点空闲。

4、公平性：系统必须公平地对待每个用户，主要表现在每个用户对系统资源的平等享用上，即具有同一优先级的用户对于系统资源的占用机会是均等的，系统不能有任何偏见。另外，系统还必须公平地对待每个作业，对于具有相同优先级的作业，它们被调度的机会是均等的。

面向用户的性能指标主要有：

1、等待时间：一个作业从提交给系统，到作业开始运行所经历的时间。通常，用户希望自己的作业等待时间短，而系统管理员则希望平均等待时间短。

2、响应时间：一个作业从提交给系统，到作业运行完毕所经历的时间。

3、减速比（Slowdown）：作业在一个系统上的响应时间与其运行时间的比例，

它可以通过如下公式表示：

$$SlowDown = \frac{ResponseTime}{RunTime} = \frac{WaitTime + RunTime}{RunTime}$$

6．1．2 性能评价的实例分析

为了评价 Greedy_Replica 算法的性能，我们采用了以下几个性能评价指标：

1、跨度：$Makespan = \max(EndTime_i) \qquad i = 1,...,N$

2、系统的资源利用率：$Utilization = \frac{Effective}{Total} = \frac{Effective}{Effective + Invalid}$

其中，*Effective* 表示 N 个作业运行后占用的有效资源时空区大小，*Total* 表示 N 个作业运行后占用的实际资源时空区大小，*Invalid* 表示 N 个作业运行后占用的无效资源时空区大小。定义作业 i 在计算节点 j 上运行所占用的资源时空区大小为 *cost*，其中 $cost = RunTime \times RequestedResources$。

3、平均响应时间：$AverageResponseTime = \frac{1}{N}\sum_{i \in Jobs}(EndTime_i - SubmitTime_i)$

下面给出 4.4.3 节中实例的性能指标。作业的运行时间（单位：分钟）、请求的资源大小（单位：GHZ）和采用 Greedy_Replica 算法（以下简称为 GR 算法）运行结果的时空图分别如下：

$$RunTime[i,j] = \begin{array}{c} \\ J1 \\ J2 \\ J3 \\ J4 \\ J5 \\ J6 \end{array} \begin{array}{c} \begin{array}{ccc} S1 & S2 & S3 \end{array} \\ \begin{bmatrix} 6 & 8 & 14 \\ 3 & 6 & 10 \\ 2 & 5 & 6 \\ 2 & 4 & 6 \\ 2 & 4 & 5 \\ 1 & 2 & 4 \end{bmatrix} \end{array} \qquad RequestedResources[i,j] = \begin{array}{c} \\ J1 \\ J2 \\ J3 \\ J4 \\ J5 \\ J6 \end{array} \begin{array}{c} \begin{array}{ccc} S1 & S2 & S3 \end{array} \\ \begin{bmatrix} 4 & 2 & 2 \\ 4 & 2 & 1 \\ 5 & 2 & 1 \\ 5 & 2 & 1 \\ 5 & 3 & 1 \\ 3 & 2 & 1 \end{bmatrix} \end{array}$$

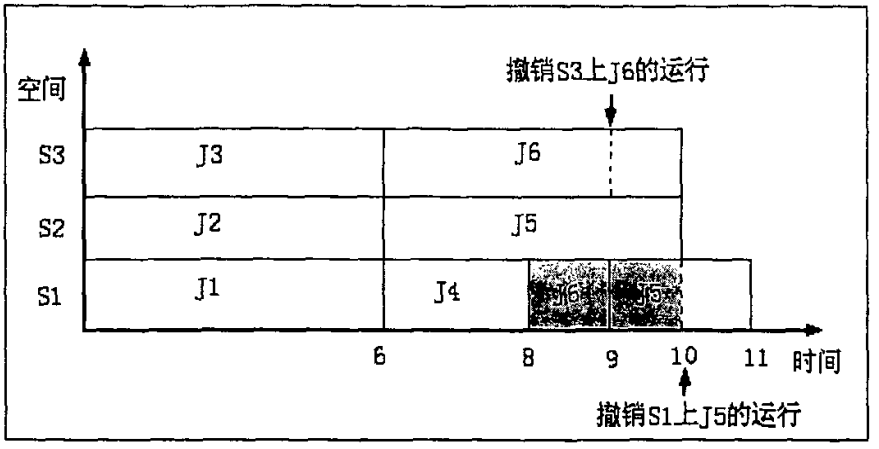

图 6.1 GR 算法运行结果的时空图

1、跨度：*Makespan* = 10 分钟

2、有效的作业与计算节点运行组合：<J1，S1>，<J2，S2>，<J3，S3>，<J4，S1>，<J5，S2>，<J6，S3>，因此，*Effective* = 6×4 + 6×2 + 6×1 + 2×5 + 4×3 + 1×3 = 67

被撤销的作业与计算节点运行组合为：<J6，S3>运行时间为 3 分钟，<J5，S1>运行时间为 1 分钟，因此，*Invalid* = 3×1 + 1×5 = 8

占用的实际资源时空区大小：$Total = Effective + Invalid = 67 + 8 = 75$

系统的资源利用率：$Utilization = \frac{Effective}{Total} = \frac{Effective}{Effective + Invalid} = \frac{67}{67 + 8} = 0.89$

3、假设 6 个作业同时提交给系统，即 $SubmitTime = 0$。$EndTime[i]$ = {6，6，6，8，10，9}。

作业平均响应时间：$AverageResponseTime = \frac{1}{6}(6 + 6 + 6 + 8 + 10 + 9) = 7.5$ 分钟

我们同时也给出基于 Replica 的轮转调度算法（以下简称为 RR 算法）的运行结果，如图 6.2 所示。作业的运行时间与请求的资源大小仍采用 4.4.3 节中实例的数据。由于 RR 算法的运行结果与作业的初始队列有关，这里假设初试化后的队列为 {J6，J5，J4，J3，J2，J1}。其中 J6 为队头， J1 为队尾。

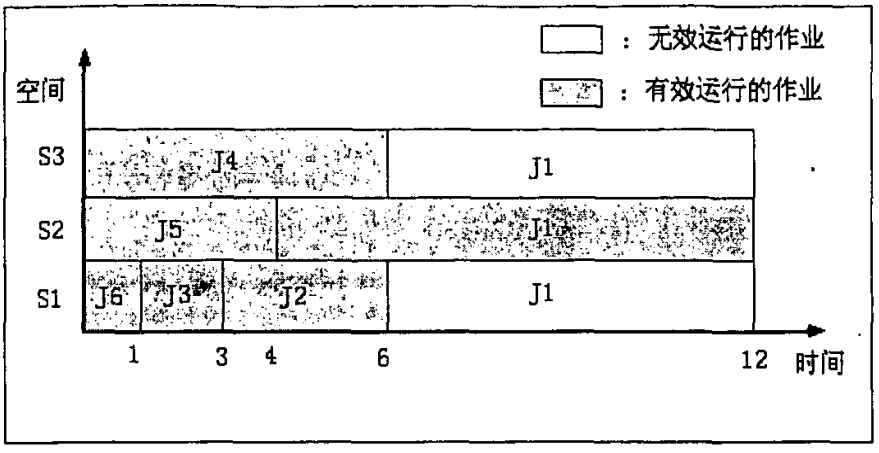

图 6.2 RR 算法运行结果的时空图

1、跨度：*Makespan* = 12 分钟

2、有效的作业与计算节点运行组合：<J6，S1>，<J5，S2>，<J4，S3>，<J3，S1>，<J2，S1>，<J1，S2>，因此，*Effective* = 1×3+4×3+6×1+2×5+3×4+8×2 = 59

被撤销的作业与计算节点运行组合为：<J1，S3>运行时间为 6 分钟，<J1，S1>运行时间为 6 分钟，因此，*Invalid* = 6×2+6×4 = 36

占用的实际资源时空区大小：$Total = Effective + Invalid = 59 + 36 = 95$

系统的资源利用率：$Utilization = \frac{Effective}{Total} = \frac{Effective}{Effective + Invalid} = \frac{59}{59 + 36} = 0.62$

3、假设 6 个作业同时提交给系统，即 $SubmitTime = 0$。$EndTime[i]$ = {12，6，3，6，4，1}。

作业平均响应时间：$AverageResponseTime = \frac{1}{6}(12 + 6 + 3 + 6 + 4 + 1) = 5.3$ 分钟

表 6.1 GR 算法与 RR 算法的性能比较

	跨度	消耗的总的资源时空区大小	资源利用率	平均响应时间
GR 算法	10	75	0.89	7.5
RR 算法	12	95	0.62	5.3

6.2 主要的仿真平台

为了验证相关调度算法的有效性，需要对其在不同场景下的性能进行评价。要构造具有代表性的网格环境需要各种各样的资源，在真实的网格环境中是难以实现的。即使有真实的网格环境，由于各个资源及其负载情况是在随时变化的，

而且每个资源所属的自治域各不相同，使得作业调度的性能测试非常困难。所以，网格性能研究一般采用网格仿真环境来完成。但是，目前网格仿真工具仍处于初级阶段，各种仿真工具各自有其特点，还没有一种普遍认同的仿真工具。

目前针对网格环境下的资源管理和作业调度的仿真环境主要有 Bricks、MicroGrid、SimGrid、GridSim、ChicSim 和 Optorsim。

Bricks 仿真环境[39]由日本东京技术学院主导开发，它是一个对高性能广域计算环境中的各种调度方案进行分析和比较的性能评价系统，用于仿真客户－服务器模式的全球计算系统，提供高性能计算机上的科学数据库的远程访问机制，使用集中或全局调度策略。

Bricks 由广域计算环境和调度单元两部分构成。广域计算环境主要由以下三部分实体组成：客户机代表用户提交请求，服务器代表可以获得的资源，网络代表客户机和服务器之间的网络行为。Bricks 采用队列系统来模拟真实环境中这三者的离散事件操作；而调度单元则用来对各种模拟行为进行协调。Bricks 采用“组件化”的设计方法，这使得它的组件可以被替换来测试别的调度算法，而且还可以通过 Bricks 的外部接口将现存的网格计算环境中的组件融合进来。

MicroGrid[40]由加利福尼亚大学研制，是专为 Globus 定制的仿真器。允许在可控虚拟网格环境中模拟执行用 Globus 构建的应用。仿真产生的结果比较真实可信。但生成多种应用、多种网格环境和不同调度策略的仿真结果非常费时，另外仿真代码的编制也比较困难。

SimGrid[41]是由美国加州大学圣地亚哥分校网格研究和创新实验室主导开发，它的目标是为在网格环境下进行分布并行应用调度研究提供一个合适的模型和抽象并生成准确的模拟结果。它是基于 C 语言的应用调度仿真环境，支持分时共享资源及负载情况的仿真，可以模拟由实际应用抽取来的参数评价以及用生成的数据约束实际数据。由于受限于单一调度实体和分时共享系统，难于仿真多个争用网格资源的应用及多种调度策略。SimGrid 使用基于 trace-driven 的模拟，它按照真实的网格资源中的访问 trace 记录来模拟网格资源，从而达到更真实的网格模拟。

GridSim[42]由澳大利亚墨尔本大学 Rajkumar Buyya 领导开发，它的首要目标是通过模拟来研究基于计算经济模型的有效资源分配方法。GridSim 通过资源的“买”和“卖”来引入“经济模型”，从而达到控制网格资源的使用目的。由于

调度策略基于微观经济模型，较难扩展为通用的网格仿真环境。

GridSim 是在 SimJava 的基础上开发的，它提供丰富的函数库以支持模拟网格环境中的异构资源、用户、应用程序、用户代理和调度器。网格资源、用户和用户代理被视为不同的实体，它们通过消息事件（输入和输出）进行通信。除了通过手工编程来实现模拟外，GridSim 还提供了一套图形界面工具 Visual Modeler（VM）帮助用户配置网格环境并产生相应的代码。模拟结束后，用户可以调用GridSim中的称为GridStatistics的库函数来收集各种模拟的统计数据。

ChicSim[43][44]是由美国芝加哥大学分布式系统实验室领导开发，它为我们提供了一个理解大量资源和用户的行为、研究资源分配策略是否可行的有效平台。ChicSim 建立在并行环境模拟器 Parsec 的基础上，主要由 3 个部分组成：站点、网络和驱动器。目前，对数据密集型的应用的研究主要采用 ChicSim 仿真器

OptorSim[45]基于欧洲数据网格的结构，主要用来模拟数据网格中的个体组件间的交互。它是为了研究典型的网格环境和评估各种副本优化算法而设计的一个仿真器。OptorSim 是一个用 Java 实现的基于时间的网格模拟器。用户只需要在模拟开始之前在配置文件中指定网格的拓扑结构和工作列表即可。不同的 ResourceBroker 的调度算法和 Replica Optimizer 的副本管理算法可以在一个参数文件中设定，同时还可以指定其他的参数，比如任务的文件传输模式，初始化的文件分布，处理时间和网络的拥塞程度等。模拟结束之后，会有一些统计数据输出。表 6.2 给出了主要仿真器的特点。

表 6.2　主要仿真器的特点

模拟器名称	是否基于库	可视化	开发语言	模拟实现
Bricks	否	不支持	Java	API
MicroGrid	基于 massf	不支持	C	配置文件
SimGrid	否	不支持	C	API
GridSim	基于 SimJava	支持	Java	API
ChicSim	基于 Parsec	不支持	C	配置文件
OptorSim	否	不支持	Java	配置文件

6．3 模拟实验

6．3．1 实验的设计

为了验证算法的性能，我们设计了一个模拟程序，对提出的 Greedy_Replica

算法进行测试。实验中，计算节点的个数固定为 20 个，作业数目分为 3 组，分别有 100，200 和 300 个作业。作业的运行时间矩阵和请求的资源大小矩阵均是已知的，并且是随机产生。本文提出的 Greedy_Replica 算法采用了 Replica 的思想，为了比较客观的测试算法的性能，选用基于 Replica 的轮转调度算法进行比较。比较的性能指标是：作业的运行时间、资源利用率、归一化消耗总的资源和归一化平均响应时间。考虑到 RR 算法和 GR 算法消耗的总资源不同，单纯考虑资源的利用率并不能充分的说明问题，所以我们又对两种算法消耗的总资源进行比较，消耗的资源指作业运行占的资源时空区大小。在消耗总的资源和平均响应时间两项指标中，均以 RR 算法作为基准，并对结果进行归一化。

6．3．2 模拟结果与分析

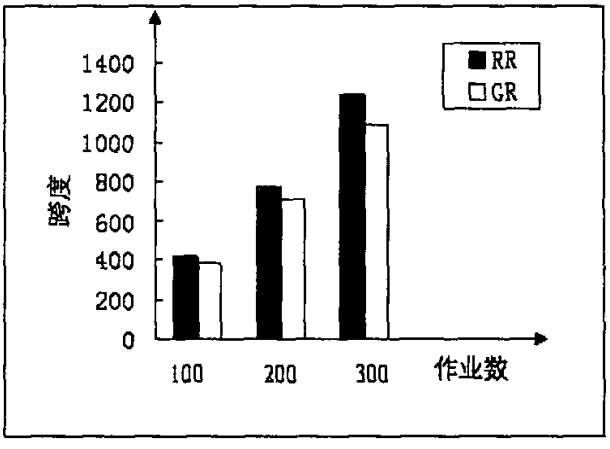

图 6.3　作业数目与总的运行时间

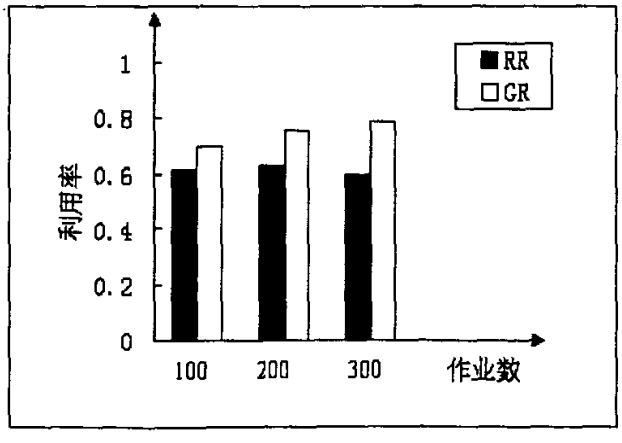

图 6.4 作业数目与资源的利用率

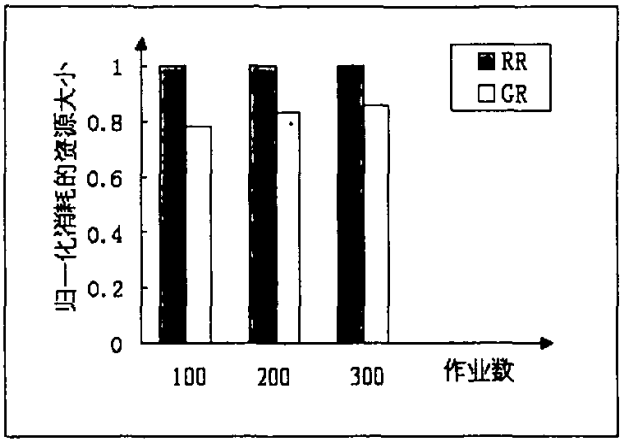

图 6.5 作业数目与消耗的资源

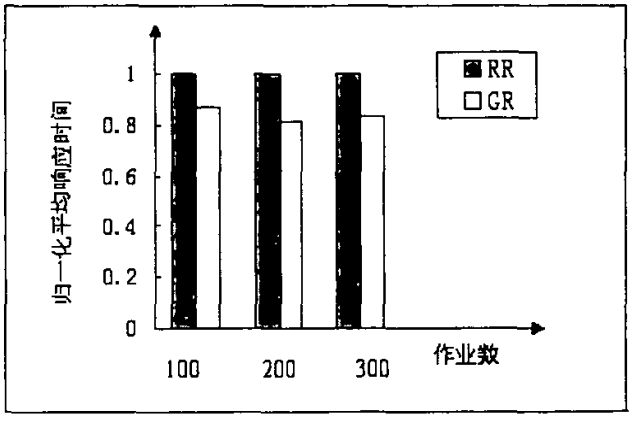

图 6.6 作业数目与平均响应时间

从模拟结果可用看出，随着作业数量的增加，RR 和 GR 算法的总的运行时间增加，在总的运行时间方面，GR 算法要优于 RR 算法，但两者之间的差距不大。在资源利用率方面，随着作业数目的增加，RR 算法的资源利用率有下降的趋势，这是由于 RR 算法没有考虑作业与计算节点之间的匹配、撤销副本的执行造成资源的浪费；而 GR 算法仍有较高的资源利用率，说明提出的贪心策略在保证资源利用率方面是有效的。GR 算法消耗的资源也要优于 RR 算法。

7 总结与展望

信息产业的第三次浪潮正在孕育中，其核心思想是通过支持虚拟组织的网格计算技术来克服目前的分布计算技术要求服务资源必须来自于单一组织的约束，从而实现因特网上所有资源的全面联通，最终实现网络虚拟环境上的资源共享和协同工作，以消除信息孤岛和资源孤岛。网格作为一种软件和硬件的基础设施，它协调地理上分布的各类资源，为用户提供可靠的、一致的、统一的访问，它通过中间件构成一个基于 Internet 的大规模分布计算环境，成为一种新的进行大规模科学和工程计算的平台。

网格计算系统极其庞大，涉及许多复杂的技术，例如：计算资源描述机制、计算资源发现管理模型、计算作业调度算法、计算网格安全体系结构等。对网格计算的研究充满了的困难和挑战。作者在整个研究生阶段的研究过程中，只对网格作业调度算法这一很小的领域进行了较为深入的探讨。本文的研究工作总结如下：

1、通过建立网格作业调度的数学模型，清楚的阐明了网格作业调度所要解决的基本问题。考虑了网络传输带来的延迟，对数学模型进行了一定的改进。

2、详细地分析了典型的网格作业调度算法，借鉴了 Replica 思想，开发空间并行性，提出了一种基于贪心策略的算法，称为 Greedy_Replica 算法。该算法采用了一种贪心策略，目标函数是作业的运行时间和请求的资源大小的乘积。在算法的不同阶段，采用两种截然相反的贪心策略，前一阶段采用“最大”贪心策略，目的是使长作业得到优先的处理，剩下一些相对短的作业采用 Replica 来加快总的执行时间。在后一阶段采用“最小”贪心策略，目的是提高资源的利用率，减少不必要的资源浪费。

3、针对网格的动态性和异构性，提出了一种自适应调度算法。该算法提供了一种框架，可方便地将各种调度算法“组合”在一起。另外，本文还提出了一种负载平衡因子的定义。

4、最后，本文总结了常用的作业调度算法的性能评价指标。通过实例，给出了性能指标的计算方法。设计了模拟程序，对提出的 Greedy_Replica 算法进行了仿真。

虽然对网格作业调度算法的研究取得了一点点成绩，但是，必须承认取得的

研究成果中还存在着许多不足之处和不尽如人意的地方。

由于网格计算是一个非常复杂的过程，涉及到许多随机和不确定的因素，加上目前很难对其进行非常精确的建模，还没有一种非常成熟可靠的数学工具来描述网格。本文建立的数学模型理想化的成分较多，不能真实地反映实际情况。

此外，对提出的 Greedy_Replica 算法，只给出了定性的说明，没有给出算法的定量分析，没有从理论上对提出的贪心策略的效率进行彻底研究。仿真实验的设计也不一定合理可靠，得出的仿真结果不可避免的存在误差，研究工作存在着不足。

针对前面工作的不足以及网格计算的发展状况，作者计划在将来进一步开展以下几个方面的工作：

1、给出的作业调度数学模型比较粗糙，存在着许多不足，很多细节我们是假设已知的，无法反映网格的动态性，因此有必要进行更加深入的研究。这一方向的工作需要有坚实的数学基础，主要是通过概率论和排队论建立预测模型。

2、本文给出的数学模型建立在作业之间相互独立的基础上，并没有考虑作业之间相互依赖的应用，而这种应用是广泛存在的，很有研究的价值。

3、进一步完善提出的 Greedy_Replica 算法。主要从两个方面入手，第一方面：通过算法分析，从理论上说明提出的贪心策略的效率，这需要扎实的数学功底和很强的算法分析能力。第二个方面：对提出的算法进行优化，Replica 的核心思想是利用了空间并行性减少作业的运行时间，我们知道微处理器中流水线主要通过开发时间并行性来提高性能，能否把作业的执行时间重叠，将流水线技术引入网格作业调度中，甚至借鉴超标量处理器中经典的 Tomasulo 算法[46]的设计思想，同时开发时间并行性和空间并行性，进一步提高算法的性能。这些问题有待进一步研究。

4、由于网格系统具有动态性、分布性和自治性等特点，资源随时会发生变化，因此自适应调度算法很有应用前景，需要进一步完善提出的自适应调度算法，考虑多种性能参数下的调度算法。考虑到多个性能指标之间通常会有约束，一种可能的方向是通过建立数学规划模型来解决。

参考文献

[1] I.Foster, C.Kesselman. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Journal of High Performance Computing Applications, 2001, 15 (3):200-222

[2] I.Foster, C.Kesselman. Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, 1997, 11(2):115-128

[3] Grimshaw A.S., Wulf W.A. The Legion Vision of a Worldwide Virtual Computer. Communications of the ACM, 1997, 40(1):39-45

[4] 徐志伟,李伟. 织女星网格的体系结构研究.计算机研究与发展, 2002, 39(8):923-929

[5] D.P.Anderson, J.Cobb, E. Korpela, M.Lebofsky. SETI@home: An experiment in public-resource computing. Communications of the ACM,2002, 45(11):56-61

[6] Larry Smarr, Charles E. Catlett. Metacomputing. Communications of the ACM, 1992, 35(6):44-52

[7] I.Foster, J.Geisier, S.Tuecke. MPI on the I-Way:A Wide-Area, Multimethod Implementation of the Message Passsing Interface. Proc.1996 MPI Developers Conference, 1996, 10-17

[8] I.Foster, C.Kesselman. The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufrmann Publishers, 1999

[9] 徐志伟,冯百明,李伟. 网格计算技术. 北京:电子工业出版社, 2004. 95-96.

[10]I.Foster, C.Kessemlan. Grid services for distributed system integration. Computer, 2002, 35(6):37-46

[11]Krauter, Klaus, Buyya, Rajkumar, Maheswaran, Muthucumaru. A taxonomy and survey of Grid resource management systems for distributed computing. Software-Practice and Experience, 2002, 32(2):135-164

[12]许智宏. 关于提高网格计算性能和服务质量的几点研究（天津大学博士论文）, 2004

[13]M.Pinedo. Scheduling:Theory,algorithms,and systems. Prentice Hall,Englewood Cliffs,NJ, 1995

[14]R.Mirchandaney, D.Towsley, J.A.Stankovic. Adaptive Load Sharing in Heterogeneous DistributedSystems. Journal of Parallel and Distributed Computing, 1990, 9(4): 331-346

[15]Y.Kwok, I.Ahmad. Benchmarking and comparison of the graph scheduling algorithm. Journal of Parallel and Distributed Computing, 1999, 59(3):381-422

[16]Maheswaran, H.J.Siegel. A dynamic matching and scheduling algorithms for

heterogeneous computing systems. In Seventh Heterogeneous Computing Workshop, IEEE Computer Socitey, 1998, 57-69

[17]郑纬民. 计算机系统结构. 第二版. 北京：清华大学出版社，1998. 400-401

[18]David Fernández-Baca. Allocating modules to processors in a distributed system. IEEE Transactions on Software Engineering, November, 1989, 15(11): 1427-1436

[19]D. Paranhos, W. Cirne, and F. Brasileiro. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In International Conference on Parallel and Distributed Computing, Lecture Notes in Computer Science, 2003, 2790(8):69-180

[20]O.H. Ibarra, C.E. kim. Heuristic algorithms for scheduling independent tasks on non-identical processors. Journal of the ACM, 1977, 24(2):280-289

[21]R.F. Rreund, M. Gherrity. Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In the 7th IEEE Heterogeneous Computing Workshop (HCW' 98), 1998, 184-199

[22]M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In the 8th IEEE Heterogeneous Computing Workshop(HCW' 99), 1999, 30-44

[23] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, Francine Berman. Heuristics for scheduling parameter sweep applications in Grid environments. In Proc. of the 9th Heterogeneous Computing Workshop (HCW'2000). Cancun, Mexico, 2000, 349-363

[24]Xiaoshan He, Xianhe Sun, Gregor von Laszewski. QoS guided min-min heuristic for grid task scheduling. Journal of Computer Science and Technology, 2003, 18(4): 442-451

[25]Goldberg D E. Genetic Algorithms in Search, Optimization and Machine Learning, MA: Addison-Wesley, 1989

[26]R.C. Correa, A. Ferreira. Scheduling Multiprocessor Tasks with Genetic Algorithms. IEEE Transactions on Parallel and Distributed Systems, 1999, 10(8):825-837

[27]K. Chow, B. Liu. On mapping signal processing algorithms to a heterogeneous multiprocessor system. 1991 International Conference on Acoustics, Speech, and Signal Processing (ICASSP 91), 1991, 3(5): 1585-1588

[28]M. Kafil, I. Ahmad. Optimal task assignment in heterogeneous distributed computing systems. IEEE Concurrency, 1998, 6(3): 42-51

[29] T. Braun, H. Siegel, R. Freund. A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing System. In 8'h IEEE Heterogeneous Computing Workshop(HCW'90), 1999, 15-29

[30]Noriyuki Fujimoto, Kenichi Hagihara. Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid. The 32nd Annual International Conference on Parallel Processing (ICPP-03). Taiwan: IEEE Press, 2003. 91-398

[31]Noriyuki Fujimoto, Kenichi Hagihara. A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks. SAINT 2004 Workshop on High Performance Grid Computing and Networking. Tokyo Japan: IEEE Press, 2004. 674-680

[32]R. L. Graham. Bounds for certain multiprocessing anomalies. Bell System Technical Journal, 1966, 45(11):1563-1581

[33]Thomas H. Cormen, Charles E. Leiserson. 算法导论. 第二版. 北京：高等教育出版社，2002. 370-379

[34]王晓东. 计算机算法分析与设计. 北京：电子工业出版社, 2001. 83-86

[35]Robert Armstrong, Debra A. Hensgen, Taylor Kidd. The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-Time Predictions. IEEE Heterogeneous Computing Workshop (HCW '98), 1998: 79-87

[36]Robert Sedgewick. C算法.第三版.北京：人民邮电出版社.2004.272-306

[37]M. Wu , X. Sun. A General Self-adaptive Task Scheduling System for Non-dedicated Heterogeneous Computing. IEEE Intl Conference on Cluster Computing, 2003

[38]Dror G. Feitelson. Metric and Workload Effects on Computer Systems Evaluation. Computer, 2003, 36(9):18-25

[39]Atsuko Takefusa. Bricks: A Performance Evaluation System for Scheduling Algorithms on the Grids. JSPS Workshop on Applied Information Technology for Science, 2001.

[40]Hyo J. Song, Xin Liu, Dennis Jakobsen, Ranjita Bhagwan, Xianan Zhang. The MicroGrid: a Scientific Tool for Modeling Computational Grids. Proceedings of Super Computing 2000, 8(3):127-141

[41]Henri Casanova. SimGrid: A Toolkit for the Simulation of Application Scheduling. Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid. Los Alamitos, CA :IEEE Computer Society Press, 2001. 430-437

[42]Rajkumar Buyya, Manzur Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. The Journal of Concurrency and Computation: Practice and Experience, 2002, 14(13-15):1175-1220

[43]Kavitha Ranganathan, Ian Foster. Identifying Dynamic Replication Strategies for a High Performance Data Grid. Proceedings of the International Grid Computing Workshop, Denver, Colorado, USA, 2001

[44]Kavitha Ranganathan, Ian Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. International Symposium of High Performance Distributed Computing, Edinburgh, Scotland: IEEE CS Press, 2002

[45]William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, Floriano Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. International Journal of High Performance Computing Applications, 2003, 17(4):403-416

[46]John L. Hennessy, David A. patterson 著，郑纬民，汤志忠，汪东升译. 计算机系统结构一量化研究方法. 第三版. 北京：电子工业出版社, 2004. 119-129

致 谢

首先，衷心感谢我的导师郭忠文教授。在三年研究生学习和科研过程中，我得到了郭老师的言传身教，无论是从理论研究还是工作方法乃至生活做人，郭老师都给予我极大的帮助和启迪。他扎实的理论基础和渊博的知识大大开阔了我的视野，他丰富的项目经验给了我很多有益的启发，他严谨求实的作风和对学生诲人不倦、耐心细致的教学态度给我留下了深刻印象，他是一位真正的良师益友。我在此向郭老师执着追求事业和默默无闻的奉献精神致以崇高的敬意，对郭老师无私的帮助和教诲表示感谢。

此外，我还要感谢在实验室一起学习的同学，尤其是魏诺、杨彬、赵扬帆、郭强、徐洪兴，他们总是在我最需要的时候给予我无私的帮助与关心，十分怀念我们共同度过的那些日子。特别感谢李之伟同学给我提供技术上支持。感谢我的同宿舍好友苗琦龙，很荣幸能结识这样的知心朋友。

最后，将最深切的谢意献给我的父母，他们无私的关爱和深切的期望是我求学路上的强大精神力量！是他们的爱使我能有今天，他们给我的支持和关怀将继续鼓舞我在人生的道路上不断前进！在此，我祝愿他们永远幸福安康！

感谢所有给予我关心和鼓励的人！

作者研究生期间论文发表情况

[1] 王鹏，郭忠文。基于 VerilogHDL 的高速可综合 FSM 设计与实现。计算机工程与设计，已录用。

[2] 徐丹田，栾新，王鹏。基于 UPnP 协议的 AV 框架的实现。 计算机应用研究，已录用。

[3] 郭忠文，赵扬帆，王鹏，李之伟。高速采样产品测试系统的设计与实现。计算机应用研究，已录用。