摘要

将异构任务调度到异构资源上的问题，即任务分配问题，一般是 NP 问题。存在许多任务调度问题的具体实例的启发式算法，但多数情况下效率都不高。Holland 提出的遗传算法应用进化策略实现了对调度空间的更快搜索，能够更快更好的找到优质的解。该算法在许多领域都得到了应用。许多研究者探索了遗传算法在同构和异构多处理机上对任务调度的应用，并取得了许多成绩。

但是，他们都多情况作了各种假设，降低了解决方案的通用性，如调度必须事先离线做出并且不能改变，所有的通信时间事先知道，所有的处理机具有相同的处理能力，处理机专门用来处理来自调度器分配来的任务等。这些假设限制了这些调度策略在分布式系统中的通用性。网格具有动态性和异构性，所以这些策略不适合网格中的应用。

本文提出了一种遗传算法的改进算法—动态遗传算法 DGA（Dynamic Genetic Algorithm），根据网格系统各服务节点的计算能力、负载及网络状态进行动态调度，从而向用户提供最优性能，不仅使总的完成时间最短，还尽量考虑到使主机的空闲时间最短，同时要满足每个任务的 deadline 的要求。动态遗传算法主要是在基本遗传算法的基础上，针对网格任务调度的动态特性，提出了新的编码机制和适应度函数，能够很好的适应空闲主机数量和等待调度任务数量的动态变化。根据新的编码机制，重新设计了选择算子、交叉算子和变异算子。

在 OPNET 环境中构建了一个局部网格的仿真模型，对所提出的动态遗传算法进行了仿真实现，并与常见的其他网格任务调度算法（如 Min-min、Max-min、FCFS 等）进行了对比，试验结果表明，动态遗传算法具有很好的优化能力，提供了较好的服务质量。

关键词： 网格计算，遗传算法，OPNET，任务调度

ABSTRACT

The problem of scheduling heterogeneous tasks onto heterogeneous resources, otherwise known as the task allocation problem, is an NP-hard problem for the general case. Many heuristic algorithms exist for specific instances of the task scheduling problem, but are inefficient for a more general case. The use of Holland's genetic algorithms in scheduling, allows good solutions to be found quickly and for the scheduler to be applied to more general problem. Many researchers have investigated the use of GAs to schedule tasks in homogeneous and heterogeneous multi-processor systems with notable success.

Unfortunately, assumptions are often made which reduce the generality of these solutions, such that scheduling can be calculated off-line in advance and cannot change, all communications times are known in advance, all processors have equal capabilities and are dedicated to processing tasks from the scheduler. These assumptions limit the generality of these scheduling strategies in real-world distributed systems. The main characters of grid are dynamic and heterogeneous, so these strategies are not appropriate for grid computing.

Desirable goals for grid task scheduling algorithms would shorten average delays, maximize system utilization and fulfill user constraints. In this paper a scheduling strategy is presented which uses a Dynamic GA (DGA) to schedule heterogeneous tasks onto heterogeneous hosts, allowing for tasks to arrive for processing continuously, and considers variable system resources, which has not been considered by other schedulers. In DGA we develop a new coding system and fitness function for local grid computing, and design new operators of selection, crossover and mutation.

A local grid simulation model is constructed by OPNET. We realize the DGA algorithm in this simulation, and compare with other grid task scheduling algorithms (such as Min-min, Max-min, FCFS). The simulation results show that the DGA scheduling algorithm has good ability of optimization, and provide good quality of service.

Key Words: Grid Computing, Genetic Algorithm, OPNET, Task Scheduling

独创性声明

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和取得的研究成果，除了文中特别加以标注和致谢之处外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 天津大学 或其他教育机构的学位或证书而使用过的材料。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了谢意。

学位论文作者签名：薛桂香　　签字日期：　2006 年　02月01 日

学位论文版权使用授权书

本学位论文作者完全了解 天津大学 有关保留、使用学位论文的规定。特授权 天津大学 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，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。同意学校向国家有关部门或机构送交论文的复印件和磁盘。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：薛桂香　　导师签名：赵珩

签字日期：2006年02月01日　　签字日期：2006年02月01日

第一章 绪论

1.1 研究背景

网格就是能够对高端计算能力提供独立、一致、普遍且廉价访问的硬件和软件体系结构[1]。网格系统由大量异构资源组成，具有复杂、动态、自治、可扩展等特点。用户通过向网格系统提交计算任务来共享网格资源，网格调度程序再按照某种策略把这些任务分配给合适的资源。高效的调度策略或算法可以充分利用网格系统的处理能力，从而提高应用程序的性能。在网格环境里如何有效地调度计算任务是影响网格计算是否成功的最重要因素之一。

网格计算系统与分布式系统和并行系统相比有很多相同的特征，但与二者又有着非常重要的区别。与分布式系统类似，位于多个管理域下的超级计算机通过不可靠的网络进行连接，并且需要对广域分布的动态资源进行集成，但是网格计算系统对高性能的要求使其编程模型及接口与分布式系统有极大的差别。网格计算系统作为并行系统还需要进行超级计算机之间的通信调度以满足应用对性能的要求，然而由于网格计算系统的异构性以及动态性，现有的并行计算技术不能很好的适应这种需求。

一般而言网格计算系统具有以下几个特征：

（1） 扩展性：网格计算系统初期的规模较小，随着超级计算机系统的不断加入，系统的规模随之扩大。

（2） 系统多层次的异构性：构成网格计算系统的超级计算机有多种类型，不同类型的超级计算机在体系结构、操作系统及应用软件等多个层次上具有不同的结构。

（3） 结构的不可预测性：与一般的局域网系统和单机结构不同，网格计算系统由于其地域分布和系统的复杂使其整体结构经常发生变化。

（4） 动态和不可预测的系统行为：在传统的高性能计算系统中，计算资源是独占的，因此系统的行为是可以预测的，而在网格计算系统中，由于资源的共享造成系统行为和系统性能经常变化。

（5） 多级管理域：由于构成网格计算系统的超级计算机资源通常属于不同的机构或组织并且使用不同的安全机制，因此需要各个机构或组织共同参与解决多级管理域的问题。

对于网格计算系统来说，最根本的问题是实时获得系统的结构和状态信息，通过这些信息对网格应用进行配置，并能实时获得计算资源的状态信息。

网格计算系统的目标是使用户能够共享其中的计算资源并以合作的方式进

行计算，为此有两个层次的工作要做：其一是网格计算前端，主要解决最终用户通过统一的界面来使用广域网上各类计算资源的问题；其二是网格计算内核，主要解决计算任务在广域网中各种超级计算机上协作完成的问题，提供一个完整的程序开发和运行环境。当用户提出计算请求时，计算问题的执行代理 agent 在系统内部的计算资源上进行合理的调度和管理，最后得出运行结果并通过网格计算前端反馈给最终用户。

为实现上述目标，必须重点解决以下三个问题：

（1） 异构性：由于网格是由分布在广域网上不同管理域的各种计算资源组成，那么怎样实现异构机器间的合作和转换是首要问题。

（2） 扩展性：要在万个资源规模不断扩大、应用不断增长的情况下，不降低性能。

（3） 动态自适应性：在网格计算中，某一资源出现故障或失败的可能性较高，资源管理必须能动态监视和管理网格资源，从可利用的资源中选取最佳资源服务。

1.2 主要研究工作和创新之处

任务调度是网格计算中一个至关重要的问题，其算法将直接影响到网格环境中任务执行的效率以至成败。用户通过向网格系统提交计算任务来共享网格资源，网格调度程序再按照某种策略把这些任务分配给合适的资源。高效的调度算法可以充分利用网格系统的处理能力，从而提高应用程序的性能。本文在OPNET环境中构建了一个局部网格的仿真模型，提出了一种遗传算法的改进算法——动态遗传算法DGA（Dynamic Genetic Algorithm），根据网格系统各服务节点的计算能力、负载及网络状态进行动态调度，从而向用户提供最优性能。

本论文的主要贡献和创新点如下：

（1） 本论文首先总结了网格任务调度的常见算法。

（2） 本论文引入了基于agent的网格模型，很好地解决了扩展性的问题，并在OPNET仿真软件上实现了该模型。

（3） 设计并实现了一种遗传算法的改进算法——动态遗传算法DGA（Dynamic Genetic Algorithm），根据网格系统各服务节点的计算能力、负载及网络状态进行动态调度，从而向用户提供最优性能。

1.3 论文结构

本论文正文共分六章。

第一章是论文的绪论，叙述工作的研究背景、主要的研究工作内容及创新之处，介绍论文的总体结构安排。

第二章的内容给出网格计算环境中任务调度的一般模型，列举并分析一些现有的任务调度策略及网格仿真环境。

第三章介绍基本遗传算法的原理。主要包括基本遗传算法的产生与发展、概要、基本遗传操作等基本原理。

第四章的内容是提出基于动态遗传算法DGA的网格任务调度算法。本章的内容包括对基本遗传算法的理论介绍，动态遗传算法的设计思想和关键设计方案的详细介绍。

第五章的内容是对基于动态遗传算法的网格任务调度算法的实现。首先介绍了OPNET仿真环境，并在此仿真环境中设计仿真了基于agent的网格模型，实现了论文所提出的基于动态遗传算法的网格任务调度算法，给出了核心算法的结构，并对仿真环境及算法结果进行了分析与评价。

第六章是对论文工作的总结，并指出亟待解决的一些问题，以及对未来工作的展望。

论文的最后是参考文献、发表论文和科研情况的说明以及致谢。

第二章 网格计算中任务调度的基本问题

2.1 网格计算

2.1.1 网格计算的历史背景及现状

网格这一术语于20世纪90年代中期提出，用来表述一种适用于高端科学和工程的分布式计算体系结构。当前在建设这种基础设施以及它的扩展和将其应用于商业计算方面都取得了很大的进展。网格的概念和相关技术最初是为实现科研协作中的资源共享提出来的，首先是在早期的Gb/s试验台，然后规模日益扩大。这种环境下的应用包括满足数据分析等计算需求的分布式计算、各种分布式数据集的联合、大量科学数据的协同可视化、科学仪器与远程计算机和档案库的联合，这些不同使用模式下的共同问题需要在多个机构组成的动态虚拟组织间实现协作式资源共享和问题求解。我们所关注的共享主要不是简单的文件交换，而是对计算机、软件、数据和其他资源的直接访问，这种共享是工业界、科学界、工程界出现的协同问题解决和资源代理策略所需要的。这种共享有必要进行高度控制，资源提供者和用户需明确、仔细地定义共享什么，谁可以共享，以及共享时所要满足的条件。由这些共享规则定义的一组个体/或机构形成我们称之为虚拟组织（VO）的概念。虚拟组织概念是许多现代计算理论的基础。它使得不同的组织与个人所形成的群体可以通过一种受控的方式共享资源，以便组员来协作完成一个共同的任务。虚拟组织之间在他们的目的、范围、大小、持续时间、结构、所在的社区和社会关系等方面会有很大的不同。虚拟组织可以是小规模的或大规模的，短生存期的或长生存期的，单组织的或多组织的，同构的或异构的。然而，我们仍然可以确定很多共同关心的问题和技术需求，具体来说，我们看到的是一种高度灵活的共享关系的需求，这种共享关系可以是客户-服务器关系，也可以是对等关系；是对资源共享提供怎样的高级、精确的控制功能的需求，包括细粒度的和多所有者的访问控制、授权、本地和全局策略的应用；对共享各种资源的需求，包括从程序、文件、数据，到计算机、传感器和网络等；将各种资源虚拟化为服务的需求，以便于这些资源能以标准的方式来访问，而不需要考虑其他物理位置和实现方法；对满足各种应用模式的需求，从单个用户到多个用户，从性能敏感型到成本敏感型，还包括服务质量、调度、协同分配和记账等问题。

当在跨越异构和动态的资源集上根据服务质量提供服务时，虚拟组织应用

程序开发者面临着一些共同需求：服务质量是否根据共同的安全语义、分布式工作流和资源管理、协同故障恢复、问题确定服务来评估或者根据其他机制来度量。资源共享通常是有条件的，共享关系是随着时间的变化而动态变化的，通常不是简单的客户-服务器方式，而是 P2P 方式。同一资源可能在不同的上下文中以不同的方式被使用，需要考虑资源使用的优先级。除了安全和策略这类问题之外，网格用户通常更关心在虚拟系统中获得各种服务质量。从许多方面来看，这种需求可以简单的认为是需要一种基础设施，这种基础设施方便了跨组织边界资源的控制共享，它就是网格。信息技术及其基础设施的根本目的是让人们处理日常任务变得更加高效或有效。这些任务的完成在某种程度上依赖于和其他人的协作。网格不仅仅是一种应运而生的技术，更是我们的基础设施必须发展的方向。

网格技术提供了共享和协调使用各种不同资源的机制，因此使得我们能够从地理上、组织上分布的组件中创建出一个虚拟计算系统，这个虚拟计算系统充分集成了各种资源以获得理想的服务质量[28]。这些技术包括：当计算跨越多个机构时，支持管理证书和策略的安全解决方案；资源管理协议和服务，支持安全的远程访问计算和数据资源以及协同分配多种资源；信息查询协议和服务，提供关于资源、组织和服务的配置信息和状态信息；还有数据管理服务，在存储系统和应用之间定位和传输数据集。

学术界和企业界历经 10 年左右的研究和开发，网格技术开始成型，并延续至今继续发展。如图 2-1 所示，我们可以将这个发展过程分为四个不同的阶段。

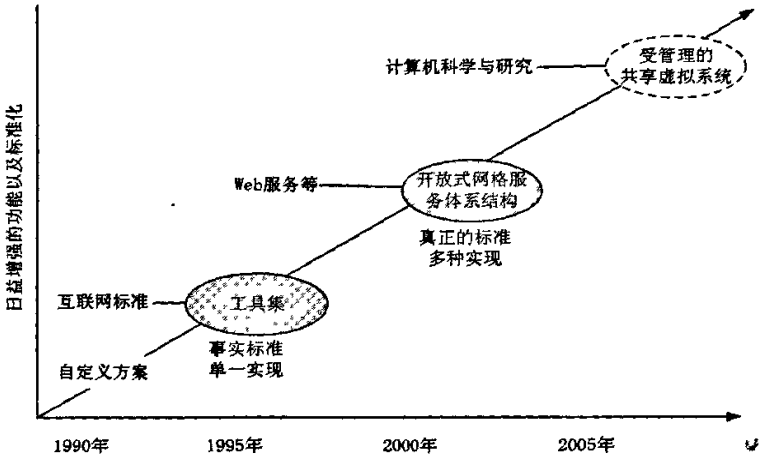

图 2-1 网格计算的发展

我们将网格定义成一种系统，该系统通过标准、开放的通用协议和接口来协调分布式的资源以提供最好的服务质量。网格继承和协调存在于不同控制域内的资源和用户，并且也处理安全、政策、付费和成员资格等在协调分布资源场景中出现的问题。网格建立在多目标协议和接口之上，这些协议和接口用来处理这些基本问题，如认证、授权、资源发现和资源访问，这些协议和接口标准化并公开化是很重要的。全球网格论坛 GGF 和其他组织正在加紧开发相关标准。网格允许协调使用它的组成资源，以便提供各种服务质量，如响应时间、吞吐量、可用性和安全性。网格还可能需要协同分配多种资源类型来满足复杂的用户需求，这样的组合系统效用将远大于它的各部分效用之和。

网格就是能够提供对高端计算能力的独立、一致、普遍且廉价的访问的硬件和软件体系结构[1]。通过使用网格技术，人们能够实现廉价、普适(Ubiquitous)的高性能计算，能够合作存取各种数据信息，能够提供广域多媒体应用等等。然而，由于网格计算是个新的研究领域，网格计算环境相对于网络计算环境有着更为复杂的特征，最主要的特征就是动态性和异构性。

任务调度是网格计算中一个至关重要的问题，其算法将直接影响到网格环境中任务执行的效率以至成败。用户通过向网格系统提交计算任务来共享网格资源，网格调度程序再按照某种策略把这些任务分配给合适的资源。高效的调度算法可以充分利用网格系统的处理能力，从而提高应用程序的性能。由于网格计算是个新的研究领域，在网格环境里如何有效地调度计算任务是影响网格计算是否成功的最重要因素之一。

目前国内外网格研究已经成了一个热点，具有代表性的研究项目和成果主要有 Globus、Legion、Nimrod_G、Netsolve、AppleS 和 Condor 等。

Globus[2]是美国多家研究机构的合作项目，Globus 的核心是一个工具集，包括一组执行基本服务的组件，如安全控制、资源获取、资源管理、资源预约、数据管理和通信等，向用户应用提供分布异构资源的一个虚拟机。Globus 最初只支持计算网格，现在也支持数据网格。Globus 提供了调度组件作为其工具集的一部分但没有提供具体的调度策略，依赖高层调度器执行任务调度，很多调度器中间件如Nimrod-G、AppleS 和Condor 都是基于Globus 服务开发的。

Legion[3]是美国维吉尼亚大学研究的基于面向对象的网格操作系统，提供资源预约和应用级的周期任务调度或批任务调度。Legion 资源管理结构是分层结构的，支持分散的调度策略，并支持通过资源代理进行调度策略的扩充，因此可以用Nimrod-G 和AppleS 等应用级调度器代替Legion 的默认调度策略。

澳大利亚莫那西大学开发的计算和服务网格Nimrod-G[4]，利用网格系统如 Globus 或Legion 进行资源发现，通过网格资源代理GRACE，利用微观经济模型

对网络资源进行管理和调度，支持资源预约和基于微观经济模型的QOS，通过周期性重调度实现系统的负载平衡。

Netsolve[5]是一种基于客户-代理-服务器结构的服务网格，由代理执行信息存储与维护、资源发现和资源调度，客户端支持C、FORTRAN、METLAB 和网页应用程序。Netsolve 的目标是提供简单易用的客户端，隐藏并行处理的复杂性，由服务器端支持各种科学计算服务。

AppLeS(Application Level Scheduling) [6]是加利福尼亚大学的研究项目，采用应用程序级的调度方法，用网络气象服务（Network Weather Service）监测资源性能的动态改变情况。

Condor[7]是美国威斯康星大学开发的一个高吞吐量的计算环境，可管理分属于不同机构的大量PC 机、工作站以及集群，以利用闲散CPU 周期著名，每个机器上运行的资源代理周期性广播其可提供服务，客户代理广播其需求，匹配器执行任务到资源之间的调度。

中科院计算所正在进行的“织女星计划”（Vega 计划）[8]的目标是具备以下几种能力：大规模的数据处理能力、高性能计算能力，以及具备资源共享和提高资源利用率的能力。与国内外其它网格研究项目相比，“织女星网格”的最大特点是“服务网格”（Service Grid）的概念。而服务网格有三个要点：第一，它是一种通用网格，不只是支持科学计算，还支持其它服务，包括通信服务、数据服务、信息服务、计算服务、交易服务等等；第二，服务是基本的应用模式，即客户端向网络发出服务请示，网格完成服务，并将结果通知客户端；第三，网格的主要评价标准不单纯是计算速度等传统指标，而是类似“服务等级协议”（Service Level Agreement）这样的一套用户满意度或服务质量评价标准。

当前国内外重要网格应用研究项目包括：亚太地区网格ApGrid（Asia Pacific Grid）[9]，美国NASA 网格 IPG（Information Power Grid）[10]，欧洲数据网格DATAGrid [11]，欧洲网格EuroGrid [12]，全球信息网格GIG[13]，美国NSF 网格TeraGrid [14]，德国网格Deutschland Grid/D-Grid [15]，国家网格CNGrid （科技部863 计划）[16]，中国教育科研网格计划ChinaGrid，E-Science 网格研究计划（国家基金委），中国空间信息网格，上海城市信息网格，等等。可见，网格计算的应用已经引起了世界各国的高度重视。

2.1.2 网格体系结构

网格计算目前比较重要的体系结构是 Ian Foster 等提出的 5 层沙漏体系结构和结合 Web Service 的开放网格体系结构 OGSA。WSRF 草案规范于 2004 年年初推出，可以看作是 OGSA 的进一步发展，但目前对其标准的讨论仍存在一些

争议。5层沙漏中强调以“协议”为中心，主要侧重于定性的描述，OGSA以“服务”为核心，定义了网格服务（Grid Service）的概念。 Web 服务资源（WS-Resource）结构是表示有状态资源和 Web 服务之间的关系的方法。后两种体系结构的基本思想都是将网格中的各种计算资源抽象为虚拟组织，各个虚拟组织通过网格服务连接起来构成虚拟的网格环境。近来，网格体系结构又出现了一个新的发展趋势：WSRF（Web Service Resource Framework，Web 服务资源框架）[17]。WSRF 草案规范于 2004 年年初推出，可以看作是 OGSA 的进一步发展，但目前对其标准的讨论仍存在一些争议。虽然此规范出现较晚，也还没有得到正式认同，但其中一些思想还是很值得我们借鉴。Web 服务资源（WS-Resource)结构是表示有状态资源和 Web 服务之间的关系的方法。Web 服务资源框架是一组被提议的 Web 服务规范，它根据特定的消息交换和相关的 XML 定义来定义 Web 服务资源方法的描述。这些规范使程序员可以声明和实现 Web 服务和一个或者多个有状态的资源之间的关联。它们描述了定义资源状态的视图以及将它与 Web 服务描述相关联来形成 Web 服务资源的总的类型定义的方法。它们也描述了如何通过 Web 服务接口来访问 Web 服务资源的状态，并且定义了与 Web 服务资源分组和寻址相关的机制。

综上所述，这两种体系结构的基本思想都是将网格中的各种计算资源抽象为虚拟组织，各个虚拟组织通过网格服务连接起来构成虚拟的网格环境。与传统的分布式计算环境相比，网格计算环境具备了更高的可管理性与自治性[18]。

2.1.3 网格特点

一般而言网格计算系统具有以下几个特征：

（1） 扩展性：网格计算系统初期的规模较小，随着超级计算机系统的不断加入，系统的规模随之扩大。

（2） 系统多层次的异构性：构成网格计算系统的超级计算机有多种类型，不同类型的超级计算机在体系结构、操作系统及应用软件等多个层次上具有不同的结构。

（3） 结构的不可预测性：与一般的局域网系统和单机的结构不同，网格计算系统由于其地域分布和系统的复杂使其整体结构经常发生变化。

（4） 动态和不可预测的系统行为：在传统的高性能计算系统中，计算资源是独占的，因此系统的行为是可以预测的，而在网格计算系统中，由于资源的共享造成系统行为和系统性能经常变化。

（5） 多级管理域：由于构成网格计算系统的超级计算机资源通常属于不同的机构或组织并且使用不同的安全机制，因此需要各个机构或组织共同参与解决

多级管理域的问题。

对于网格计算系统来说，最根本的问题是实时获得系统的结构和状态信息，通过这些信息对网格应用进行配置，并能实时获得计算资源的状态信息。

网格计算系统的目标是使用户能够共享其中的计算资源并以合作的方式进行计算，为此有两个层次的工作要做：其一是网格计算前端，主要解决最终用户通过统一的界面来使用广域网上各类计算资源的问题；其二是网格计算内核，主要解决计算任务在广域网中各种超级计算机上协作完成的问题，提供一个完整的程序开发和运行环境。当用户提出计算请求时，计算问题的执行代理 agent 在系统内部的计算资源上进行合理的调度和管理，最后得出运行结果并通过网格计算前端反馈给最终用户。

为实现上述目标，必须重点解决以下三个问题：

(1)异构性：由于网格由分布在广域网上不同管理域的各种计算资源组成，怎样实现异构机器间的合作和转换是首要问题。

(2)扩展性：要在万个资源规模不断扩大、应用不断增长的情况下，不降低性能。

(3)动态自适应性：在网格计算中，某一资源出现故障或失败的可能性较高，资源管理必须能动态监视和管理网格资源，从可利用的资源中选取最佳资源服务。

网格计算系统中某一资源出现故障或失败的可能性较高，系统的资源会不断扩大，应用会不断增长，系统的整体结构和整体性能会不断的发生变化，并且随时有不可预测的系统行为发生，这就要求资源管理程序能够动态监视和管理网格资源，从目前可利用的资源中选取最佳资源服务，尽量减小由于这种故障或失败、整体结构和整体性能发生变化或不可预测的系统行为等问题对网格整体性能的影响，因而提出了自适应调度算法，根据网格系统各个计算模块的计算能力、负载、及网络状态进行自适应调度，从而向用户提供最优的性能。即，在包含大量不同计算机中的网格中，选择最合适给定任务的计算机进行计算，使得在满足给定任务的 deadline(截止时间)要求的前提下，尽量使得系统负载平衡、吞吐率高、利用率高等。

2.2 任务调度的基本问题

2.2.1 任务调度的定义

在分布系统中，一个程序可以看作是一个任务集，这些任务可以串行或并行地执行。任务之间一般是有优先次序约束的。调度问题的目标[19]是要在满足一定

的性能指标和优先约束关系的前提下，将可并行执行的任务按适当分配策略确定一种分派和执行顺序，合理分配到各处理机上有序地执行，已达到减少总的执行时间的目的。

性能和效率是评价调度系统的两个基本特征。我们应以任务分派（调度）的质量和调度算法（调度程序本身）的效率为基础来评价调度系统。调度质量以产生的优化调度的性能为基础来衡量；调度算法的效率以时间复杂性为基础来衡量。例如，如果以优化一个程序的完成时间来衡量，显然完成时间越短越好。如果两个调度算法产生相同的调度质量，则调度算法越简单越好。

任务调度的相关定义如下[7]：

在有 m 个处理单元和任务图 $G=(T,E)$ 之上的调度是一个函数 f，f 将每个任务以某个特定的开始时间映射到某个处理单元。

从形式上，一个调度可描述为：

$$f:T\to\{1,2,\cdots,m\}\times[0,\infty]$$

如果存在 $v\in T, f(v)=(i,t)$，则表示任务 v 被调度到处理单元 P_i 上，且从时间 t 开始执行。

函数 f 可以表示成如下定义的 Gantt 图。

Gantt 图用于直观地表示所有任务的开始时间和完成时间。在一个 Gantt 图中，有一个分布系统的处理单元表，对于表中每个处理单元都有一个任务分配表，即将分配到这个处理单元的所有任务按执行时间排列成表，其中包括开始时间和结束时间，分别用 $ST(t_j,P_i)$ 和 $FT(t_j,P_i)$ 表示。

实际上，Gantt 图是一个四元表：

$$Gantt\left(P_i,t_j,ST(t_j,P_i),FT(t_j,P_i)\right)$$

其中，

$P_i(i=1,2,\cdots,m)$ 为处理单元；

$t_j(j=1,2,\cdots,n)$ 为分配到 P_i 上的任务；

$ST(t_j,P_i)$ 为 t_j 在 P_i 上的开始时间；

$FT(t_j,P_i)$ 为 t_j 在 P_i 上的结束时间；

一个任务调度系统的最大调度长度 SL（Schedule Length）定义为所有处理机 P_i 中的

$$\max_i\{\sum_j FT(t_j,P_i)\}$$

调度的目标就是要将任务适当分配到处理机并协调任务之间的执行顺序，使得并行执行时满足并行任务之间的优先约束关系，而且 SL 最小。

任务调度问题属于NP完全问题，没有最优解。任务调度问题的最常见的目标函数就是完成时间ω（makespan）。用CP来表示调度算法的综合性能指标，该指标主要通过三个参数（makespan, idle_time, over_deadline）来表示。计算公式如下：

$$CP=\frac{W^{t}\Gamma+W^{m}\omega+W^{o}\theta}{W^{t}+W^{m}+W^{o}}$$

其中，Γ，ω和θ分别表示makespan，idle_time，over_deadline，而W^{t}，W^{m}和W^{o}分别表示它们的权值。对于一族给定的权值，CP值越小，算法的综合性能越好。

基本上每个调度算法都需要估计系统中每个任务所需要的执行时间T。可以把执行时间T分成两部分T_n和T_c，这里，T_n是经过网络发送数据到计算机m的时间加上从计算机m上接受结果的时间，T_c是在计算机m上计算所用的时间。时间T_n可以通过下列计算下列参数得到：

（1）agent与计算机m之间的网络延迟和带宽；

（2）任务长度的大小；

（3）返回结果的大小。

时间T_c需要计算下列参数：

（1）任务的规模；

（2）使用算法的复杂度；

（3）执行任务计算机m的性能，主要取决于m的负载和计算速度。

2.2.2 任务调度策略的分类

一般从调度策略本身考虑，可将任务调度分为静态调度和动态调度两类。静态调度的主要特点是实现简单，但调度质量不佳。动态调度的主要特点是能充分发挥各个资源的处理能力，但实现起来比较复杂，可能由于任务的动态调整影响个别任务的执行效率。目前出现了将静态调度与动态调度相结合的混合调度方法，可以兼顾二者的优越性。

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。

网格任务调度与负载平衡是提高网格计算环境可用性和效率的关键问题，

Nimrod-G、AppLeS 和Condor-G 等网格任务调度器都基于特定的调度策略，适用于特定情况的任务调度。Globus任务调度方法基本是轮循方式的；Nimrod-G 采用应用程序级的基于微观经济模型的调度方法，适用于各个资源的定价机制都很完整的情况；AppleS 采用应用程序级的调度方法，用网络气象服务监测资源性能的动态改变情况作为任务调度的依据，信息维护代价较高，且要求各节点间的连接都具有监测机制；Condor-G 要求每个资源代理周期性广播其可提供的服务，客户代理广播其需求，匹配器执行集中式的任务到资源之间的匹配调度。

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任务调度策略。从系统状态估计的支持上，可将网格任务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。

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 可扩展调度策略。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。为了尽可能提高网格计算资源的利用率，应该尝试设计面向某个具体应用或者某类应用的任务调度模式。

2.2.3 任务调度算法的研究进展

除了一些特殊情况的最优调度，绝大多数最优调度问题是NP完全的，有的甚至是NP难解的。目前，主要的研究努力集中在下面的方法：

.对特殊情况的最优调度，比如，带特殊约束的模型。

.非最优调度，可以分成局部最优方案和次优方案。

局部最优方法使用有效的搜索技术确定问题的解空间中的(局部)最优方案。这些方案可进一步分成下面四种：

.解空间枚举和搜索。首先构建状态空间，其中每个状态对应一个可能的调度方案。它依靠各种搜索技巧和边界搜索在解空间中寻找解。

.数学编程。一个给定的调度问题转换成一个整数、线性或者非线性编程问题。有三个步骤 ：(a)一个目标函数，以求目标函数最小化，常常定义成程序执行时间。(b)一个约束集合和通信开销。(c)使用动态编程技巧解决基于a和b的约束最优问题。

.模拟退火。模拟退火利用物理退火过程的特性构建一种局部最优方案的搜索方法。通常这样一个过程由三部分组成:(a)对初始非最优调度方案做最小幅度的随机的变动。(b)评估新的调度方案。(c)继续这一过程直到没有更好的方案产生，也就是说得到了一个局部最优方案。模拟退火从理论上讲是全局最优算法，因为它能以一定的概率接受差的能量值，从而有可能跳出局部极小，但它在实际求解中需要的计算时间比较长。

.遗传算法。遗传算法是基于自然界遗传和进化规律的搜索算法。它结合己有结果和搜索空间的新领域来产生新的结果。具有如下优点：（1）搜索过程中不易陷入局部最优，能以极大的概率找到全局最优解；（2）具有并行性，非常适合于大规模并行分布处理；（3）易与别的技术（如，神经网络、启发式规则等）相结合，形成性能更优的算法。但是，遗传算法的搜索效率低，易过早收敛，理论还不成熟，收敛定理证明有比较困难，其编码方式、群体大小、遗传算子的概率等尚需进一步研究。本文就是对遗传算法进行了深入研究，根据网格计算中任务调度的特点（主要是动态性和异构性），重新对编码机制、适应度函数确定、选择算子、交叉算子、变异算子等进行了设计，提出了动态遗传算法。

和特殊情况的最优解一样，局部最优解也有极大的计算量并且极端耗费时间。次优解法依靠启发式方法取得次优解。

调度问题一直是计算机科学的一个热门的研究领域。有许多专著和期刊专题讨论调度问题，大量的论文在许多计算机科学方面的期刊上发表。不仅如此，许多新兴的学科都把各自领域的研究成果应用于解决调度问题，如：基因算法的研究、人工神经网络的研究、人工智能和分布式人工智能的研究都把解决调度问题作为其应用领域之一[20]。本文就是对基本的遗传算法加以改进，提出动态遗传算法，应用到网格计算的任务调度中。

调度问题作为分布计算和机群计算的核心问题之一，其研究工作在国内一直十分活跃。许多大学和科研机构在并行分布计算和调度问题方面进行了卓有成效的研究。其中，清华大学的李三立院士在分布式动态负载平衡及其仿真方面[21]、郑纬民教授等在利用人工智能中的搜索技术来解决调度问题方面、王怀民教授等在基于 Agent 的分布计算环境方面、谢立教授在智能分布式任务调度方面、刘大有教授在分布 Agent 协商调度方面都取得了很好的研究成果。

2.2.4 网格计算中任务调度的主要难点及常见算法

一、网格计算中任务调度的主要难点

任务调度是并行分布计算的重要组成部分，在该领域已经进行了大量的研究，取得了许多理论和实践上非常重要的成果[22,23,24]。然而，随着计算网格的

出现，需要新的调度算法来解决网格所带来的新要求。

传统的并行调度问题是在一个并行机上调度一个应用程序的各个子任务，同时忽略资源的具体共享特性，以减少轮转时间（turn around time）。在超级计算中心，通常是调度一组来自不同用户的应用程序到并行计算机上，以最大化系统利用率，这样调度问题被增强了。然而，在网格中问题更加复杂，因为调度要使用许多这样的计算中心。广义的网格调度算法的目标应包括：提高吞吐量，最大化系统利用率，满足经济性和用户的条件限制。首先我们需要发现对提出的一组任务可用的一组资源。接着，从这些可用资源中选择那些满足预先定义的调度约束（如最小化运行时间等）的资源。解决完匹配问题，最后执行作业和在选中的环境里进行数据传输。在调度模型中，至少有两个新的网格描述需要考虑。第一个是如何处理非专用网络，对于非专用网络，由于他们有本地作业需要处理，不可能为远程作业提供排他性服务。因此，我们需要解决好如何预测在非专用网络上预测作业执行的时间。第二个是服务质量。在网格环境中，期望竞争到最好的服务质量和远程资源已完成应用任务的约束。资源为应用程序提供多级的服务质量。网格环境的调度这需要考虑应用程序和服务质量的限制，以取得应用程序和资源的更好的匹配。

为解决非专用网络的限制，主要采用性能预测的方法。这里有几种预测任务/主机对计算时间的方法。一种方法[27,28]是使用相同或类似主机的早期性能来预测当前的运行时间。短期预测方法 NWS[29]在[28]中被采用来预测任务执行时间。由于预测基于短期信息，如 5 分钟，这种方法可能在应用程序长度短和调度期间主机没有剧烈变化的情况下有效。但是，对长的应用程序或剧烈变化的运行环境，基于 NWS 的调度可能因为差的预测性能而不能满足要求。在[30]中采用了 GHS[31]系统中应用的长期的、应用程序级的预测模型。它是从严格的数学分析和大量的方针基础上产生的，分别定义了极其利用率、计算力、完成任务后局部作业服务和任务分配等的效果，预测效果更好一些。对服务质量，可对不同性质的任务加上不同的优先级来实现。

二、网格计算中常见的任务调度算法

网格计算系统中某一资源出现故障或失败的可能性较高，系统的资源会不断扩大，应用会不断增长，系统的整体结构和整体性能会不断的发生变化，并且随时有不可预测的系统行为发生，这就要求资源管理程序能够动态监视和管理网格资源，从目前可利用的资源中选取最佳资源服务，尽量减小由于这种故障或失败、整体结构和整体性能发生变化或不可预测的系统行为等问题对网格整体性能的影响，因而提出了启发式自适应调度算法，根据网格系统各个计算模块的计算能力、负载、及网络状态进行自适应调度，从而向用户提供最优的

性能。常见的启发式调度算法有：MET，MCT，Min-Min，Max-Min 等。

现存的启发式调度算法可以分为两类：在线式和批处理模式。在线式启发式算法在任务一到达调度器时就将任务映射到一台机器上。批处理模式启发式算法并不是在任务到达时就将它们映射到机器上，而是将它们存到一个集合里，到预先定义好的调度时间即映射事件发生时检查这个集合来进行调度。我们将在映射事件发生时考虑的独立任务集称为一个元任务。在线式启发式算法中，进行匹配和调度时每个任务仅仅需要被考虑一次，如最小完成时间优先的启发式算法(MCT)和最小执行时间优先的启发式算法(MET)[32]。在批处理模式启发式算法中，调度器在每个调度事件进行匹配和调度时都要考虑元任务。这使得该启发式映射算法能够做出更好的决策，因为启发式算法需要元任务的资源需求信息，并且知道大量的任务实际执行时间（因为许多任务在等到映射事件时可能已经执行完了）。Min-min 和 Max-min 算法是常见的批处理模式启发式算法。Min-min 和 Max-min 算法在 SmartNet 项目中实现。Min-min 算法的思想是把任务分配给能最早完成它的处理器。Max-min 把最高优先级给能完成时间最早的最大的任务。Max-min 算法的思想是把运行时间长的任务与运行时间短的任务重迭。比如，如果只有一个长任务，Max-min 算法就会在执行长任务时并行的执行多个短任务。相反，Min-min 算法就会并行的执行多个短任务之后再执行长任务。所以，在这个例子中，Max-min 算法要优于 Min-min 算法。Max-min 算法和 Min-min 算法需要处理器速度和任务长度的预测信息。

第三章 基本遗传算法的原理

3.1 遗传算法的产生与发展

遗传算法通过模拟达尔文的进化论而创建，它模拟生物遗传进化的过程，引入了选择、复制、交叉重组和变异等方法，并将进化论中的“物竞天择，适者生存”的概念引入到算法中，因此被命名为“Genetic”。遗传算法（Genetic Algorithms, GA）最早是由美国 Michigan 大学的 Holland 提出[26]的。早在 20 世纪 50 年代就有将进化原理应用于计算机科学的努力，但缺乏一种普遍的编码方法，只能依赖于变异而非交配产生新的基因结构。50 年代末到 60 年代初，受一些生物学家用计算机对生物系统进行模拟的启发，Holland 开始应用模拟遗传算子研究适应性。在 1967 年 Bagley 关于自适应下棋程序的论文中，他应用遗传算法搜索下棋游戏评价函数的参数集，并首次提出了遗传算法这一术语。1975 年 Holland 出版了遗传算法历史上的经典著作《自然和人工系统中的适应性》，系统阐述了遗传算法的基本理论和方法，并提出了模式定理（schemata theorem），证明在遗传算子选择、交叉和变异的作用下，具有低阶、短定义距以及平均适应度高于群体平均适应度的模式在子代中将以指数级增长，这里的模式是某一类字符串，其某些位置有相似性。同年，DeJong 完成了他的博士论文《遗传自适应系统的行为分析》，将 Holland 的模式理论与他的计算试验结合起来，进一步完善了选择、交叉和变异操作，提出了一些新的遗传操作技术。进入 80 年代后，遗传算法得到了迅速发展，不仅理论研究十分活跃，而且在越来越多的应用领域中得到应用。1983 年，Holland 的学生 Goldberg 将遗传算法应用于管道煤气系统的优化，很好地解决了这一非常复杂的问题。1989 年，Goldberg 出版了《搜索、优化和机器学习中的遗传算法》一书，这本可能是遗传算法领域被引用次数最多的书为这一领域奠定了坚实的科学基础。80 年代中期，Axelrod 和 Forrest 合作，采用遗传算法研究了博奕论中的一个经典问题—囚徒困境。在机器学习方面，Holland 自提出遗传算法的基本理论后就致力于研究分类器系统（classifier system），Holland 希望系统能将外界刺激进行分类，然后送到需要的地方去，因此命名为分类器系统，这里的 classifier 是指一个二进制串，代表一类情况。分类器系统将某一条件是否为真与串的某一位相对应，从而将产生式系统中的规则编码为二进制串，这样就可以应用遗传算法来进行演化，同时引入了基于经济学原理的信用分配机制—桶队（bucket

brigade）算法来确定规则的相对强度。Holland 和 Santa Fe 的 Arthur 等人合作，用分类器系统模拟了一些经济现象，得到了满意的结果。

遗传算法发展的简要回顾如下：

（1）1950 年，将进化原理应用于计算机科学的初步努力。

（2）50 年代末到 60 年代初，Holland 应用模拟遗传算子研究适应性。

（3）1967 年，Bagley 的论文中首次提出了遗传算法这一术语。

（4）1975 年，Holland 的经典著作《自然和人工系统中的适应性》出版，系统阐述了遗传算法的基本理论和方法。

（5）1975 年，DeJong 的博士论文《遗传自适应系统的行为分析》，将 Holland 的模式理论与他的计算试验结合起来。

（6）1983 年，Holland 的学生 Goldberg 将遗传算法应用于管道煤气系统的优化，取得了很好的效果。

进入 20 世纪 90 年代，随着计算机技术的高速发展，遗传算法在各个领域得到了广泛的应用。不同领域的专家、学者根据各自的实际问题、构造出了多种能很好地解决行业问题新的遗传算法。正如 De Jong 所说："……简体遗传算法的应用与推广已远远超出了我们最初的理论和理解，并产生了重新研究和扩展它们的必要"。新的遗传算法有的对给定问题的搜索空间采用了更自然的表达方式，有的采用了更适合具体问题的遗传运算，Michalewicz 称它们为遗传算法的演化程序。

3.2 遗传算法概要

3.2.1 遗传算法的基本思想

遗传算法[33, 34]是从代表问题可能潜在解集的一个种群（population）开始的，而每一个种群则由经过基因（gene）编码（coding）的一定数目的个体（individual）组成。每个个体实际上是染色体（chromosome）带有特征的实体。染色体作为遗传物质的主要载体，即多个基因的集合，其内部表现（即基因型）是某种基因组合，它决定了个体的形状的外部表现，如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此，在一开始需要实现从表现型到基因型的映射即编码工作。由于仿照基因编码的工作很复杂，我们往往进行简化，如二进制编码。初代种群产生之后，按照适者生存和优胜劣汰的原则，逐代（generation）演化产生出越来越好的近似解。在每一代，根据问题域中个体的适应度（fitness）大小挑选（selection）个体，并借助于自然遗传学的遗传算子（genetic operators）进行组合交叉（crossover）和变异

（mutation），产生出代表新的解集的种群。这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境，末代种群中的最优个体经过解码（decoding），可以作为问题近似最优解。

遗传算法采纳了自然进化模型，如选择、交叉、变异、迁移、局域与邻域等。

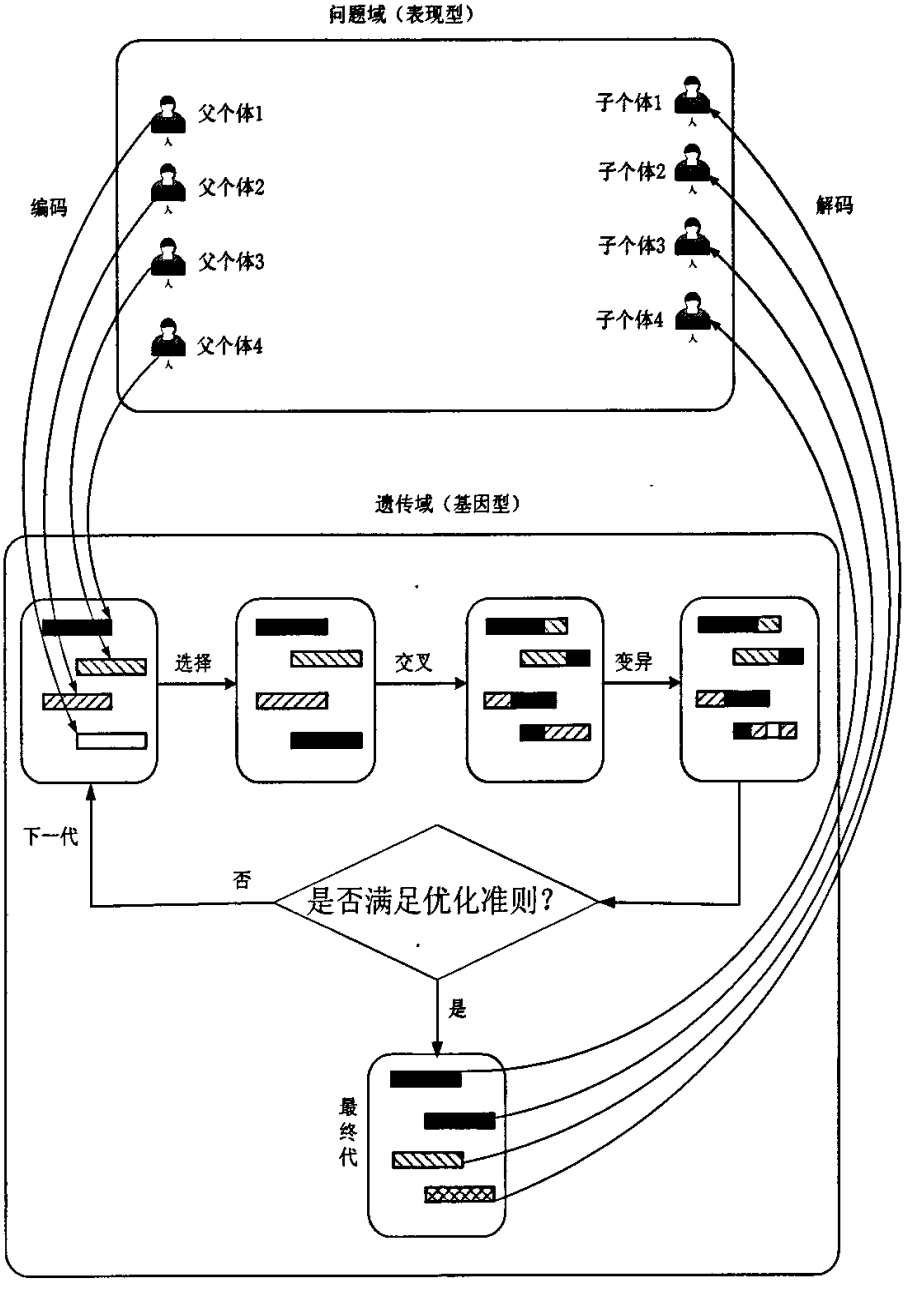

图 3-1 遗传算法的过程

图 3-1 表示了基本遗传算法的过程。计算开始时，一定数目 N 个个体（父个体 1、父个体 2、父个体 3、父个体 4……）及种群随机的初始化，并计算每个个体的适应度函数，第一代也即初始代就产生了。如果不满足优化准则，开始产生新一代的计算。为了产生下一代，按照适应度选择个体，父代要求基因重组（交叉）而产生子代。所有的子代按一定概率变异。然后子代的适应度又被重新计算，子代被插入到种群中将父代取而代之，构成新的一代（子个体 1、子个体 2、子个体 3、子个体 4……）。这一过程循环进行，直到满足优化准则为止。

3.2.2 遗传算法的特点

传统的优化方法主要有三种：枚举法、启发式算法和搜索算法：

（1）枚举法：枚举出可行解集合内的所有可行解，以求出精确最优解。对于连续函数，该方法要求先对其进行离散化处理，这样就可能因离散处理而永远达不到最优解。此外，当枚举空间比较大时，该方法的求解效率比较低，有时甚至在目前先进计算工具上无法求解。

（2）启发式算法：寻求一种能产生可行解的启发式规则，以找到一个最优解或近似最优解。该方法的求解效率比较高，但对每一个需求解的问题必须找出其特有的启发式规则，这个启发式规则一般无通用性，不适合于其他问题。

（3）搜索算法：寻求一种搜索算法，该算法在可行解集合的一个子集内进行搜索操作，以找到问题的最优解或者近似最优解。该方法虽然保证不了一定能够得到问题的最优解，但若适当地利用一些启发式知识，就可在近似解的质量和效率上达到一种较好的平衡。

随着问题种类的不同以及问题规模的扩大，要寻找一种能以有限的代价来解决搜索和优化的通用方法，遗传算法正是为我们提供的一个有效的途径，它不同于传统的搜索和优化方法。主要区别在于：

①自组织、自适应和自学习性（智能性）。应用遗传算法求解问题时，在编码方案、适应度函数及遗传算子确定后，算法将利用进化过程中获得的信息进行组织搜索。由于基于自然的选择策略为“适者生存，不适应者被淘汰”，因而适应度大的个体具有较高的生存概率。通常，适应度大的个体具有更适应环境的基因结构，再通过基因重组和基因突变等遗传操作，就可能产生更适应环境的后代。进化算法这种自组织、自适应特征，使它同时具有能根据环境变化来自动发现环境的特性和规律的能力。自然选择消除了算法设计过程中的一个最

大障碍，即需要事先描述问题的全部特点，并要说明针对问题的不同特点算法应采取的措施。因此，利用遗传算法的方法，我们可以解决那些复杂的非结构化问题。

②遗传算法的本质并行性。遗传算法按并行方式搜索一个种群数目的点，而不是单点。它的并行性表现在两个方面：一是遗传算法是内在并行的，即遗传算法本身非常适合大规模并行。最简单的并行方式是让几百甚至数千台计算机各自进行独立种群的演化计算，运行过程中甚至不进行任何通信（独立的种群之间若有少量的通信一般会带来更好的结果），等到运算结束时才通信比较，选取最佳个体。这种并行处理方式对并行系统结构没有什么限制和要求，可以说，遗传算法适合在目前所有的并行机或分布式系统上进行并行处理，而且对并行效率没有太大影响。二是遗传算法的内涵并行性。由于遗传算法采用种群的方式组织搜索，因而可同时搜索解空间内的多个区域，并相互交流信息。使用这种搜索方式，虽然每次只执行与种群规模 n 成比例的计算，但实质上已进行了大约$o(n^2)$次有效搜索，这就使遗传算法能以较少的计算获得较大的收益。

③遗传算法不需要求导或其他辅助知识，而只需要影响搜索方向的目标函数和相应的适应度函数。

④遗传算法强调概率转换规则，而不是确定的转换规则。

⑤遗传算法可以更加直接地应用。

⑥遗传算法对约定问题，可以产生许多的潜在解，最终选择可以由使用者确定(在某些特殊情况下，如多目标优化问题不止一个解存在，有一组最优解。这种遗传算法对于确认可替代解集而言是特别合适的)。

3.2.3 遗传算法的优点

遗传算法具有以下优点：

（1）广泛的适用性：遗传算法是模拟生物界而构造出的一种自然算法，以概率选择为主要手段，不涉及复杂的数学知识，也不关心问题本身的内在规律。因此，遗传算法可以处理任意复杂的目标函数和约束条件。

（2）全局优化：由于遗传算法不采用路径搜索，而采用概率搜索，所以是概率意义上的全局搜索。因此，解决的问题无论是否为凸性的，理论上都能获得最优解，避免落入局部极小点。

3.3 遗传算法的基本操作

遗传算法包括三个基本操作：选择、交叉和变异。这些基本操作又有许多

不同的方法，下面我们逐一进行介绍。

3.3.1 选择

选择是用来确定重组或交叉个体，以及被选个体将产生多少个子代个体。首先计算适应度：

① 按比例的适应度计算(proportional fitness assignment)；

② 基于排序的适应度计算(rank-based fitness assignment)。

适应度计算之后是实际的选择，按照适应度进行父代个体的选择。可以挑选以下的算法：

① 轮盘赌选择策略(roulette wheel selection)；

这种选择策略的基本思想是：根据各染色体适应度的大小与全体染色体适应值之和的比值，确定各染色体被选择进入下一代的概率。适应度越大，进入下一代的可能性就越大；反之，则越小。具体做法是：将所有染色体的适应度之和看做是一个轮盘，每个染色体根据其适应度的大小划分在轮盘中所占据的范围。然后，旋转轮盘，当轮盘停下时，指针所对应的染色体即被选中，完成一次选种。不断旋转轮盘和选种，直到选择到所需要的染色体个数为止。整个过程可以分为两部分，即概率计算和选择。

概率计算过程可分为四个步骤：

步骤 1：计算每个染色体的适应度；

步骤 2：计算种群中所有染色体的适应度之和；

步骤 3：计算每个染色体的选择概率；

步骤 4：计算每个染色体的累积概率 q_k。

选择过程可分为两个步骤：

步骤 1：在[0，1]区间内产生一个均匀分布的伪随机数 r；

步骤 2：若 $r \le q_1$，则选择第一个染色体 v_1；否则，选择第 k 染色体 v_k $(2 \le k \le P)$，使得 $q_{k-1} \le r \le q_k$ 成立。

② 随机遍历抽样(stochastic universal sampling)；

③ 局部选择(local selection)；

④ 截断选择(truncation selection)；

⑤ 锦标赛选择(tournament selection)。

3.3.2 交叉或基因重组

基因重组是结合来自父代交配种群中的信息产生新的个体。依据个体编码

表示方法的不同，可以有以下的算法：

① 实值重组(real valued recombination)：

- 离散重组(discrete recombination)；
- 中间重组(intermediate recombination)；
- 线性重组(linear recombination)；
- 扩展线性重组(extended linear recombination)。

② 二进制交叉(binary valued crossover)：

- 单点交叉(single-point crossover)：

对最简单的单点交叉法，首先，在种群中根据预先确定的交叉率随机选择一定数量的染色体对作为双亲；然后，随机选择一个断点，交换双亲断点右侧的基因链，产生新的子代；最后，用子代染色体替代父代染色体，产生新种群。

- 多点交叉(multiple-point crossover)；
- 均匀交叉(uniform crossover)；
- 洗牌交叉(shuffle crossover)；
- 缩小代理交叉(crossover with reduced surrogate)。

3.3.3 变异

交叉之后子代经历的变异，实际上是子代基因按小概率扰动产生的变化。变异是指染色体的基因以预定的概率发生改变。变异操作过程可以描述为：首先，根据变异率随机选样参与变异的父代染色体，这与选择参与交叉操作染色体的方法相同；然后，在父代染色体上随机确定发生变异的基因位置并进行变异转换，产生子代染色体；子代染色体替换父代染色体。依据个体编码表示方法的不同，可以有以下的算法：

①实值变异；

②二进制变异。

3.4 遗传算法的运行过程

遗传算法的运行过程可用如下步骤进行表述：

(1)随机产生初始种群。

(2)以适应度函数对染色体进行评价。

(3)选择高适应度的染色体进入下一代。

(4)通过遗传、变异操作产生新的染色体。

(5)不断重复第(2)—(4)步，直到预定的进化代数。

遗传算法可以归纳为两种运算过程：遗传运算(交叉与变异)与进化运算(选择)，遗传运算模拟了基因在每一代中产生新后代的繁殖过程，进化运算则是通过竞争不断更新种群的过程。

3.5 遗传算法解分布式任务调度问题的研究现状

Benten和Sait在[35]中提出了一种将一个DAG调度到一个由有限个处理机组成的全联网络(忽略网络阻塞)中的遗传算法。在他们的算法中，每个解或者说是调度方案被编码成含有v个基因的染色体，每个染色体是由任务的索引表和其被指派的处理机索引表组成的有序对。交叉算子采用标准交叉算子；变异算子是在两个被任意选中的等位基因之间互换被指派的处理机。为了产生初始化种群，他们先产生N_p个从1到v的数字的任意排列。每个排列中的数代表任务关系图的任务索引表，于是任务集合被一致地分配到各个PC上：例如，第一个v/p个任务被分配到$PE0$上，接下来的v/p个任务被分配到$PE1$上，等等。在他们的仿真研究中，证明其算法的性能比[36]中Hwang提出的算法要好。

Hou，Ansari和Ren在[37]也提出了一个使用遗传搜索的调度算法。在他们的算法中，每一个染色体是一个列表的集合，每个列表代表了在一个特定处理机上的调度方案。因此，每个染色体是二维结构的。其中一维是一个特定的处理机索引表，另一维是调度到处理机上的任务的序列。使用这样的编码方式给调度方案的表现造成了一定的限制：每一个处理机中的任务列表必须按照它们在任务图中的拓扑高度升序排列。所谓拓扑高度就是从入口结点到结点本身的最大边数。这一限制也简化了交叉算子的设计，在一次交叉中，从两个染色体中选择两个处理机，每个处理机上的任务列表被分成两部分，然后两个染色体相应地交换它们的低位部分。虽然这种交叉机制总是可以产生有效后代，然而在编码时的高度限制使得搜索不能得到最优解。于是Hou在他们方案中加入了启发式技术来解决这种病态的情况。变异算子的设计比较简单。在一次变异操作中，在调度方案中随意地选择有相同拓扑高度的任务结点进行互换。至于产生初始化种群，在算法的开始，随意地产生N_p个符合高度排序限制的排列。在仿真研究中，他们证明他们的算法产生的调度方案与最优解的差值在20%以内。

Ahmad和Dhodhi[38]使用模拟进化算法(一种遗传算法的变形)提出了一种调度算法。他们使用问题空间邻域规划方法，用染色体代表一个任务优先级列表。由于任务的优先级有赖于输入的DAG，不同的任务优先级集合代表不同的问题实例。首先，从输入DAG得到一个优先级列表，然后通过任意打乱初始列表来产生初始种群。算法中，将标准遗传算子应用到染色体以确定有最大适应值的染色体，以这个染色体为基础，使用启发式列表调度方法后，最后得到的指派与排序可以使

得原来的调度问题有最小的调度长度。整个遗传搜索过程是在问题空间中而不是在解空间中进行。这与一般的遗传算法解最优化问题类似。

Kowk和 Ahmad [39]提出了一个并行遗传算法(PGA)用于多处理机的DAG调度。他们使用遗传搜索方法的目的是因为遗传算法的重组性质可能确定一个能导致最优调度方案的最优调度列表。一个染色体是一个有效的调度列表(即是符合拓扑顺序的DAG的结点列表)，他们使用的交叉算子是顺序交叉方法；变异算子也是将两个结点互换；所有的遗传算子操作都必须保证结点间的优先级关系，以防止产生无效调度列表。然后，使用分解型并行遗传算法，对初始化种群进行遗传操作，最后产生一个最优调度列表。

第四章 基于动态遗传算法DGA的网格任务调度算法原理

4.1 动态遗传算法整体设计

当任务可以重新排序时,调度的目标就变化了，不再是寻求每个任务各自的最早完成时间，而是集中于总的完成时间（makespan）ω，它表示考虑所有任务时的最近完成时间，被定义为

$$\omega = \max_{1\le j\le m}\{te_j\}$$

目标就是最小化ω，同时尽量满足 $\forall j, te_j \le tr_j$ 。

为得到这一组合优化问题的最优解，采用了迭代启发式算法（这里采用了遗传算法GA）来发现满足上述标准的调度。这个过程包括构造一组调度，识别满足要求特征的方案，并应用到下一代中。

这一技术需要一个编码机制，能代表所有优化问题的合法方案，任何可能的方案都可以由一个具体的字符串唯一表示，字符串可以进行各种操作，直到该算法汇聚到一个近似的最优解上。为了使操作正确进行，需要一个描述每个解决方案字符串适应度值(fitness)的方法，提供该值的算法叫做适应度函数(fitness function f_v)。编码机制包含两部分：排序部分S_k（用来描述任务执行顺序），和映射部分M_{ijk}（用来描述分配给每个任务的主机)。设k 表示调度集中的调度数，映射M_{ijk}的顺序和任务的顺序是相称的。各种任务的执行时间由预测系统提供，并且与适应度函数（fitness function f_v）评价的任务目标相联系。

使用了组合的代价函数，它考虑到了总的完成时间makespan，空闲时间和期限。使用S_k和M_{ijk}直接计算调度k 的总完成时间ω_k，设T_{jk}为根据编码机制的排序部分S_k重排过后的任务集，则

$$ts_{jk} = \max_{\forall l, M_{ijk}=1}\left(\max_{\forall p<j, M_{ipk}=1}(te_{pk})\right)$$

$$te_{jk} = ts_{jk} + texe_{jk}$$

$$\omega_k = \max_{1\le j\le m}\{te_{jk}\}$$

该算法与先来先服务算法的区别在于：（1）任务顺序根据S_k决定而不是根据任务的到达时间决定，所以不必考虑到达时间t_j；（2）使用M_{ijk}来确定对主机的选择，预测评价的结果$texe_{jk}$使用相应的资源模型直接计算，而在先来先服务中，需要考虑和比较所有可能的不同主机选择集$\overline{H}$。

空闲时间的特征应该考虑进来，通过使用所有主机的平均空闲时间ψ_k表示：

$$\psi_k = \max_{1\le j\le m}(te_{jk}) - \min_{1\le j\le m}(ts_{jk}) - \frac{\sum_{j=1}^{m}\sum_{i=1}^{n} M_{ijk}(te_{jk} - ts_{jk})}{n}$$

调度前的空闲时间是不受欢迎的，因为它是首先被浪费的处理时间，最不可能被接下来的遗传算法的迭代发现，或者有更多的任务加入。有大量空闲时间的方案可以通过给ψ_k空闲时间的多余的包加以修正，主要修正早期的空闲时间，少量修正后期的空闲时间。

合约处罚θ_k由预期的期限t_r和任务完成时间t_e推出：

$$\theta_k = \frac{\sum_{j=1}^{m}(te_{jk} - tr_j)}{m}$$

调度 k 的代价值可以通过这些度量值推出，其影响由下列公式预先决定：

$$f^k = \frac{W^m\omega_k + W^t\varphi_k + W^c\theta_k}{W^m + W^t + W^c}$$

然后代价值使用动态扩展技术通过适应度值进行规范化：

$$f_v^k = \frac{f_c^{\max} - f_c^k}{f_c^{\max} - f_c^{\min}}$$

其中$f_c^{\max}$和$f_c^{\min}$分别代表调度集中最好和最坏的代价值。

遗传算法使用固定的入口大小和随机剩余选择，在两部分编码机制中采用了特殊的交叉和变化函数。交叉函数首先将两个排序的字符串以任意位置结合起来，然后重排这个字符串对，以产生合法的方案。映射部分通过下列步骤交叉：首先进行重排使其与它的任务顺序一致，然后执行单点交叉（二进制）。需要重排是因为要保持各代之间节点映射和具体任务的相关。变化阶段也分为两部分，交换操作随机的用于排序部分，位翻转随机的用于映射部分。

在 scheduler 使用上述算法执行中，必须考虑到系统的动态性。本文所提到的迭代算法的一个优点就在于它是一个进化过程，因此能够吸收系统的变化，如任务的加入和删除，网格资源中可用主机数量的变化等。

这两种方法都可以实现，并且在 scheduler 中可以交互使用。该算法提供了一个局部网格资源中跨多主机间的动态任务调度和负载平衡的好的解决方案。然而，同样的方法不能直接应用到大规模的网格环境，因为该算法不能扩展到成千上万的主机和任务情况。

4.2 遗传算法的设计原则

4.2.1 编码问题

遗传算法[33]主要通过遗传操作、交叉与变异对种群中的个体施加结构重组处理，通过选择操作不断优化群体中的个体结构，从而搜索到最优结构的个体，达到逼近问题的最优解的目的。由此可见，标准遗传算法不能直接对问题空间进行操作，必须将问题空间的解变量转换成遗传空间，由基因按一定结构组成染色体，这一转换操作就叫做编码。相反，将染色体编码映射为问题空间的操作，称为解码。遗传算法在编码空间对染色体进行遗传运算，而在解码空间对解进行评价和选择。

（一）编码问题

评价编码策略通常考虑以下三个方面：

（1）完备性。问题空间中的所有点（候选解）都能作为遗传空间中的点（染色体）来表现。

（2）健全性。遗传空间中的染色体能对应所有问题空间中的候选解。

（3）非冗余性。染色体和候选解一一对应。

应该指出，对于很多问题很难设计出同时满足上面三条要求的编码方案，但是完备性是必须满足的一项。

这三个评价规范缺乏具体的指导思想。De Jong 提出的编码规则的操作性较强，具体为：

（1）有意义基因块的编码规则：所定编码应当易于生成与所求问题相关的短距和低阶的基因块。

（2）最小字符集编码规则：所定编码应采用最小字符集，以使问题得到自然的表示或描述。

在实际应用这些规则时，仍然要针对具体问题，设计出具体有效的编码。

（二）编码方法

（1）一维染色体编码。一维染色体编码指问题空间的候选解转换到遗传空间后，其相应的染色体呈一维排列的基因链码。通常采用的一维染色体编码技术有：

①二进制编码。二进制编码是将问题空间的候选解转换为遗传空间的各位数值为“0”或“1”的字符串。

它的优点主要有：

• 符合最小字符集的编码原则；

·编码简单，便于进行交叉、变异等遗传操作，且物理概念清晰；

·便于用模式定理进行分析和预测。

但是，对于多维、高精度问题，二进制编码就会显示出一些不足：

* 二进制编码不能直接反映出所求问题的本身足够特征；

* 二进制编码的染色体长度与问题空间的区域大小和精度要求直接有关，对于大空间、高精度的问题，染色体的长度会很长，搜索空间也会很大。这样的遗传搜索相当困难；

* 相邻的二进制编码可能会有较大的 Hamming 距离，从而降低了遗传算子的搜索效率。利用灰码编码方法可以在一定程度上克服上述缺点。

②灰码编码。灰码编码是将二进制编码通过一个变换而得到的编码。其表现也为二进制的形式，所不同的是灰码编码技术保证了在遗传空间相互靠近的两个点也必须在问题空间里靠近，反之亦然。

设有二进制串 $b_1b_2\cdots b_n$，相应的灰码串为 $a_1a_2\cdots a_n$，则由二进制编码到灰码编码的变换公式为：

$$a_i=\begin{cases} b_1 & , \ if \quad i=1 \\ b_{i-1}\oplus b_i & , \ if \quad i\neq 1 \end{cases}$$

式中⊕——模 2 加法。

从一个灰码串到二进制串的变换为：

$$b_i=\sum_{j=1}^{i} a_j \pmod 2$$

灰码性质：任何两个在问题空间彼此相邻的点在遗传空间的编码只有一位不同。

灰码编码只是保证了问题空间相邻的点在遗传空间有数值为 1 的 Hamming 距离，但遗传空间还不是问题空间。采用浮点编码可以解决这个问题。

③浮点编码。如果采用浮点表达的编码方法，即每个染色体向量被编码成一个与解向量相同长度的浮点数向量，那么，在执行上，遗传空间就是问题空间，染色体直接反映了问题的规律与特性。

优点：

·精度依赖于所使用的机器，与编码本身无关，比二进制灵活方便；

·能够表达很大的区域，而二进制编码必须以牺牲精度来增加表达区域；

·容易设计出封闭的、动态的遗传算子，容易处理非常规约束。

所以，在我们的任务调度算法中，就采用了浮点编码机制。

一维编码的方法还有很多，如交叉编码、多参数编码、可变长编码等。

（2）二维染色体编码。很多实际问题中，解本身就是以二维或多维的形式

存在的，为了使问题的表达更直观，可直接采用多维染色体编码。

4.2.2 适应度函数

在遗传算法的进化过程中，对染色体的评价是由适应度函数来完成的，适应度函数的函数值作为选择运算的依据。由于进化的准则是“优胜劣汰’，即遗传算法向着适应值增加的方向进化，所以目标函数的寻优方向应该与适应度函数值增加的方向一致，这是适应度函数设计的先决条件。另外，为了理论分析的方便，最好保证适应度函数非负，有时对适应值为负的情况要进行适当的转换。

将目标函数转换成适应度函数一般要遵循下面两个基本原则：

(1)优化过程中目标函数的优化方向(如寻求目标函数的最大值或最小值)与种群进化过程中适应度函数值增加的方向相一致。

(2)适应度函数值大于等于 0。

(一)适应度函数的设计

①求最大值问题：

因为目标函数 $f(x)$ 本身就是求最大值问题，所以适应度函数可写为：

$$fit(x)=f(x)+C,\quad C+f_{min}\geq 0$$

②求最小值问题：

当目标函数为最小值问题时，可通过下式将目标函数映射成适应度函数：

$$fit(x)=C-f(x),\quad C\geq f_{max}$$

如果目标函数非负，也可采用如下的转换方式：

$$fit(x)=1/f(x),\quad f(x)>0$$

③也可利用指数函数来建立目标函数到适应度函数的转换：

$$fit(x)=C^{f(x)}$$

式中 C ——正数，具体问题根据问题选定。

此外，还可以写出其他形式的目标函数和适应度函数之间的映射关系公式。只要满足两个基本原则。在实际问题中，采用哪种转换形式要依据具体情况而定。

(二)适应度函数的标定

当种群规模不是很大的情况下，在遗传算法进化的初始阶段，如果存在最佳染色体的适应值远远高于其他染色体适应值的情况时．最佳染色体被选择的

概率非常大。这种竞争力很强的个体，很快就充斥了整个种群，从而出现遗传算法过早收敛而陷入局部极小点的现象。所以，在进化的初始阶段，应限制高适应值个体的过度繁殖，保持种群的多样性。与此相反，在进化的后期，种群中大多数染色体都有比较高的适应值，这时种群的平均适应值与最大适应值比较接近，因而平均适应值附近的个体与高适应值的个体被选择的概率几乎相等，使优良个体失去了其应有的竞争力，个体的选择近乎于随机选择。这对于搜索最佳染色体是极为不利的。所以，在进化的后期，应尽量体现个体的差异性，种群中个体的多样性和差异性是一对相互对立的矛盾属性，如何控制它们在不同进化阶段的表现程度，这就是选择压力的概念。

基于上述原因，进化过程中，各染色体适应值之间既要保持一定的差异，又不能让差异过大，可以采用对适应度函数进行标定的办法来控制这种差异。

4.2.3 选择问题

选择运算是推动进化的直接投动力。如果选择压力过大，遗传算法会过早收敛，使搜索落入局部极小点；选择压力过小，搜索过程会非常缓慢。一般来讲，在进化的初始阶段，宜采用较小的选择压力，尽量保持种群个体的多样性；在进化的后期，宜采用较大的选择压力，以提高优良个体的竞争力，使搜索向着最优解的方向发展。

选择可以通过以下三个方面进行描述：

(1)采样空间。

(2)采样机理。

(3)选择概率。

三者对于选择压力以至于遗传算法的性能都有很大的影响。

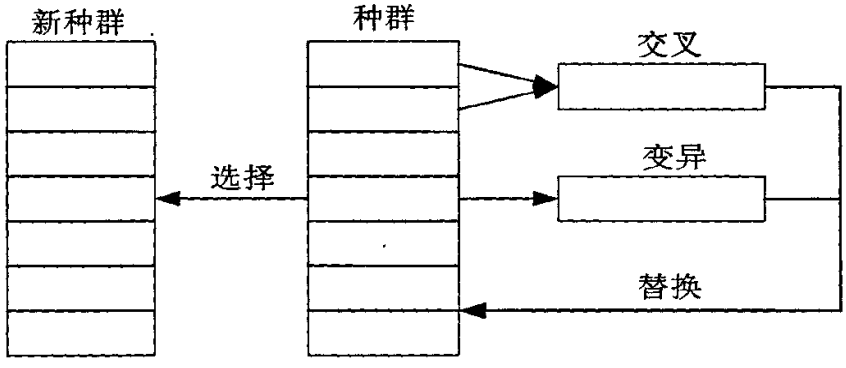

图 4-1 基于规则采样空间的选择示意图

(一)采样空间

采样空间由两个因素确定：采样空间的大小和采样空间的组成。采样空间可划分为规则的采样空间（如图4-1）和扩大的采样空间（如图4-2）两种。

令P为种群的大小，P_0为每代产生的后代数。规则的采样空间保持P不变。

扩大的采样空间为$P+P_0$，即采样空间包括了所有的双亲和后代。图3-13为基于扩大的采样空间的选择示意图。扩大的采样空间最显著的特点是有效限制了由于高交叉率和变异率造成的随机变动。

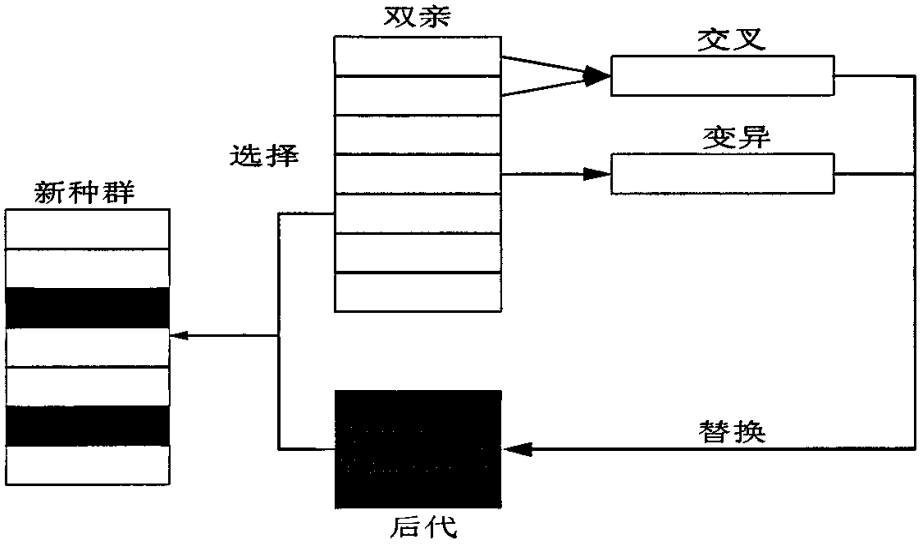

图4-2 扩大的采样空间的选择示意图

(二)采样机理

采样机理是如何从采样空间中选择染色体的理论，对种群个体的选择从原理上可分为三种：

(1)随机采样。

(2)确定采样。

(3)混合采样。

随机采样方法首先根据各染色体个体的适应值评价出其生存概率，再由生存概率确定每个染色体的实际繁殖数量。每个染色体的生存概率是确定性的，而染色体个体实际的繁殖数量是根据其生存概率随机产生的，是随机性的。整个选择过程可由两步完成：

i）确定种群中个体的生存概率。

ii）根据个体的生存概率进行随机选择。

轮盘选择是典型的随机采样方法。假定染色体v_k的适应值为f_k，其选择概

率 p_k 为：

$$p_k = \frac{f_k}{\sum_{i=1}^{P} f_i}$$

随机选择过程如下：

```
begin
  sum = 0;
  for k = 1 to P do
    p_k = sum + p_k;        计算各染色体的累积生存概率
  end

  for k = 1 to P do
    q_k = rand(0,1);        在（0，1）区间生成随机数序列
  end
  q = sort(q);              将随机数序列按升序排序

  for k = 1 to P do
    while p_k > q_k         选择染色体
      k = k + 1;
    end
  end
```

随机采样的基本思想是：适应值大的个体对应较高的生存概率，则被选择进入下一代的几率相对较高。但是，并不能保证最优的个体被选择。

确定采样的基本思想是：保证最好的染色体被选择进入下一代。常用的方法有截断选择法、精英选择法和期望选样法等。

混合采样同时具备随机采样和确定采样的特征。典型混合采样的例子是Goldberg等人提出的竞争选择法。该方法随机地选出一组染色体，从中选择最好的一个进行繁殖，每组的染色体个数称为竞争人数。通常的竞争人数为2人，称为2人竞争。

(三)选择概率

种群中染色体的选择概率与选择压力直接相关，理想的选择压力是随着进化代数的增加而变大。Holland 的原始遗传算法所采用的轮盘算法是一种正比选择方法，选择概率正比于染色体的适应值。正比选择的缺点是在进化的早期，一些超级染色体会充斥选择过程，容易失去种群染色体的多样性；而进化的后期，优良染色体的竞争能力减弱，使选择变得像随机搜索。标定法和排序法就

是为解决上述问题而提出的。

(1)标定法。标定法就是通过调整适应度函数来控制种群中染色体的生存概率。对适应度函数进行调整的意义在于：a、使种群中染色体的适应值之间保持合理的差距；b、限制某些超级染色体的繁殖速度，以满足早期限制竞争，后期鼓励竞争的要求。

(2)排序法。排序法以种群中染色体适应值的大小顺序来确定各个染色体的生存概率，而不是以适应值的数值大小来确定。该方法克服了直接基于适应值进行选择的缺点。常用的方法有：线性排序法和指数排序法。

4.2.4 交叉运算

交叉运算是最重要的遗传操作，种群通过交叉产生新的染色体，从而不断扩展搜索空间，最终达到全局搜索的目的。交叉算子的设计与编码方法密不可分，这里主要介绍二进制编码的交叉算子和浮点编码的交叉算子。

(一)二进制编码的交叉算子

二进制编码的交叉算子包括以下两项基本内容：①从父代种群中随机选择两个染色体作为双亲；②确定交叉点，从交叉点处对双亲的对应部分进行互换，产牛新的子代个体。

基本的交叉算子有：

(1)单点交叉。单点交叉也叫简单交叉。具体的操作为：在双亲的染色体串上随机确定一个交叉点，将双亲交叉点右边的部分进行互换，形成新的染色体串。

例如：

考虑如下两个 11 位变量的个体：

父个体 1：　1 1 0 0 1 0 1 0 0 1 1

父个体 2：　0 0 1 0 1 1 0 0 1 0 0

交叉后两个新个体为：

子个体 1：　1 1 0 0 1 1 0 0 1 0 0

子个体 2：　0 0 1 0 1 0 1 0 0 1 1

这里，交叉点是随机确定的，设为 5。当染色体长为 l 时，可能确定为交叉点的位置有 $l-1$ 个，所以单点交叉可能出现 $l-1$ 种不同的交叉结果。

(2)多点交叉。对于多点交叉，m 个交叉位置 K_i 可无重复随机地选择，在交叉点之间的变量间续地相互交换，产生两个新的后代，但在第一位变量与第一个交叉点之间的一段不作交换。交叉点的位置同样是随机产生的。多点交叉的搜索更加健壮。

如：

考虑如下两个 11 位变量的个体：

父个体 1： 1 1 0 0 1 0 1 0 0 1 1

父个体 2： 0 0 1 0 1 1 0 0 1 0 0

交叉后两个新个体为：

子个体 1： 1 1 0 0 1 1 1 0 0 0 0

子个体 2： 0 0 1 0 1 0 0 0 1 1 1

这里，交叉点是随机确定的，设为 3，6 和 9。对于染色体长度为l的两点交叉，可能被确定为交叉点的位置有$(l-1)(l-2)/2$个。

(3)均匀交叉。

均匀交叉方法是对双亲的每一位根据概率确定是否进行交换。具体操作方法：先随机产生一个与双亲等长度的二进制字符串掩码。其中“1”表示不交换，“0”表示交换。该二进制字符串相当于一个模板。双亲染色体串根据模板上标定的信息进行交换操作。

考虑如下两个 11 位变量的个体：

父个体 1： 0 1 1 1 0 0 1 1 0 1 0

父个体 2： 1 0 1 0 1 1 0 0 1 0 1

掩码样本为：

样本 1： 0 1 1 0 0 0 1 1 0 1 0

样本 2： 1 0 0 1 1 1 0 0 1 0 1

交叉后两个新个体为：

子个体 1： 1 1 1 0 1 1 1 1 1 1 1

子个体 2： 0 0 1 1 0 0 0 0 0 0 0

均匀交叉类似于多点交叉，可以减少二进制编码长度与给定参数特殊编码之间的偏差。均匀交叉在进行基因位的交换时，一方面，破坏模式的概率较大，另一方面，能产生以前不存在的新模式。

(二)浮点编码的交叉算子

浮点编码的交叉算子与经典的交叉算子有很大的不同。它直接在问题空间进行运算。在浮点编码的执行中，每个染色体向量被编码成一个浮点数向量，且各向量在规定的区域内，交叉算子的设计也要保证所产生的后代在规定的区域内，这是浮点编码交叉算子设计的首要条件。主要有两种：算术交叉算子和启发式交叉算子。在本论文中，对交叉算子进行了重新设计，并进行了实现，本文将在后面详细介绍。

4.2.5 变异运算

变异运算是指对染色体串的某些基因位做改变的操作。遗传算法引入变异算子可以提供初始种群不含的基因，或找回选择过程中丢失的基因，为种群提供新的内容。变异算子的作用效果在于：一方面，遗传算法具有局部搜索的能力，使遗传算法能够搜索到精确解；另一方面，可以保持种群的多样性，防止出现过早收敛的现象。在变异操作中，变异率不能太大，如果变异率大于 0.5，遗传算法就退化为随机搜索，而遗传算法的一些重要的数学特性和搜索能力也不复存在了。下面介绍几种常用的变异算子。

（一）基本变异

基本变异是针对二进制编码设计的，包括两个步骤：

（1）以概率 p_m 在种群中随机选择参与变异的染色体个体，作为父代；

（2）随机的确定父代发生变异的基因座，并对基因值进行变异。

如下所示，有 11 位变量的个体，第 4 位发生了翻转：

变异前： 0 1 1 1 0 0 1 1 0 1 0

变异后： 0 1 1 $\bar{0}$ 0 0 1 1 0 1 0

下面的算子是针对浮点编码提出的。

（二）均匀变异

均匀变异由单个的父体 x 变异产生单个的子代 x'。

设算子选择了父代 $x=(x_1,x_2,\cdots x_k,\cdots x_d)$ 中的一个随机元素 x_k，其中 $k\in(1,2,\cdots d)$。以 x_k' 替换 x_k。这里，x_k 为问题空间 $[l(k),r(k)]$ 里的一个均匀分布的随机数。

均匀变异的算子在进化计算的早期，推动可能解在搜索空间内自由移动；在进化计算的后期，允许在搜索过程中离开局部最优解，进行可能的移动，以便搜索到最好的可能解，该算子非常有用。在我的设计中就采用了该算子，并且加以修改。

（三）边界变异

（四）非均匀变异

4.2.6 主要参数的选择

遗传算法设计所涉及的参数主要有：种群规模、交叉率、变异率、进化代数等等，另外，选择具体算子时，有时还会涉及选择与算子相关的参数。遗传算法参数的选择合理与否直接关系到算法的收敛速度和精度。但是，由于影响

参数选择的因素很多，有的与问题本身内涵的客观规律有关，有的与所选择的算于有关。很难找到一个普遍的规则可寻。

(一)种群规模

种群规模是遗传算法首先需要确定的参数，是算法能否陷入局部解的主要影响因素。

一般来讲，决定种群规模的因素主要有：

(1)问题的内在规律。如果在问题空间内目标函数的极值点越多，所要求的种群规模越大；如果问题空间内目标函数的变化越平滑，所要求的种群规模越小；

(2)问题空间的范围。问题空间的取值范围越大，要求的种群规模也越大。

(3)交叉率、变异率的选择。交叉率和变异率较大时，要求的种群规模较小；反之，较大。一方面，种群规模选择得过小，容易使算法陷入局部最优解；另一方面，种群规模选得过大，增加了算法的计算量，从而减缓了算法的进化速度。

本实验中种群规模折中，取为20。

(二)交叉率

交叉率是最主要的遗传运算，遗传算法的性能在很大程度上取决于所采用的交叉算子的性能和交叉率的大小。交叉率是指各代中交叉产生的后代数与种群规模之比。交叉运算产生新个体，不断拓展搜索空间，较高的交叉率可以搜索更大的解空间，从而降低停在非最优解的机会；但是交叉率太高，会因搜索不必要的解空间而耗费大量的计算时间。常用的交叉率的取值范围为0.2~0.5，本实验中交叉率取为0.3。

(三)变异率

变异率是指种群中变异的基因数占总基因数的百分比。变异率控制了新基因引入的比例。若变异率太大，随机的变化太大，那么后代就可能失去从双亲继承下来的好的特性；若变异率太小，一些有用的基因就没有机会产生。常用变异率的数量级范围为0.1~0.001，本实验中变异率取为0.02。

(四)进化终止条件

进化计算的终止可以从两方面进行控制：预先设定进化代数或者以种群的进化程度进行控制。种群的进化程度是指种群的当前代最大适应值与种群的平均适应值的比例关系。进化的终止条件可以用下面的判断来完成：

```
begin
  for  gen = 1 to termgen
    while (max fitness − meanfitness) ≤ ε
      break;
    end
    进行进化计算;
end
```

这里，ε 是一个小的正数，根据具体问题而定。本实验中 ε 取为 0.001。预先设定最大进化代数为 100。

在实际应用中，遗传算法参数的选择和各算子的确定与具体问题是密不可分的，遗传算法只是充当了一种计算工具，它对所要解决的问题一无所知，要通过遗传计算得到满足要求的解，首先要对具体问题进行深入、细致的分析，找出问题的内在规律，将问题合理地模型化，即将实际问题概化为遗传算法可以解决的问题。这一部分相当重要，是遗传算法能够有效发挥作用的基础。

4.3 各部分实现详解

4.3.1 问题的描述

为简化起见，我们假定有 n 个等待任务，m 个空闲主机。我们建立如下约束：

①每个任务之间是独立的；

②任务之间没有优先约束关系；

③每个任务是不可分的；

④一个任务只能分配到一台主机上执行；

⑤一台主机只能一次分配到一个任务。

每个任务具有长度属性（*task_length*）和截止时间（*deadline*）属性。每个主机具有速度（*speed*）属性。为体现网格的动态性，任务产生的间隔时间是服从指数分布的随机数，每个任务的长度是按均匀分布随机产生的，截止时间也是相应随机产生的，同时每个主机的处理速度是按均匀分布动态变化的，并且主机可以变得有效或无效。

我们任务调度的目的就是使总的任务执行时间最短，同时要满足每个任务的截止时间要求。总的完成时间（makespan）ω 表示考虑所有任务时的最近完成时间，被定义为

$$\omega = \max_{1 \le j \le n} \{te_j\}$$

目标就是最小化ω，同时尽量满足 $\forall j, te_j \le tr_j$ 。

4.3.2 编码机制

这一技术需要一个编码机制，能代表所有优化问题的合法方案，任何可能的方案都可以由一个具体的字符串唯一表示，字符串可以进行各种操作，直到该算法汇聚到一个近似的最优解上。为了使操作正确进行，需要一个描述每个解决方案字符串合适值（fitness）的方法，提供该值的算法叫做适应度函数（fitness function f_v）。

对于同构处理机系统的静态任务调度来说，任务分给不同的处理机处理效率相同，因此我们分配任务时不必考虑处理机之间的差别，编码时相应的秩序对任务调度序列编码就可以了。如果任务调度集是独立任务集，任务调度序列的编码也不受优先关系的约束，自由度较大，编码也容易。对异构集群系统的相关任务调度来说，我们就不仅要考虑任务在不同处理机上处理时间的区别，任务之间的通讯时间，还要考虑任务之间的优先关系。

我们采用浮点表达的编码方法，即每个染色体向量被编码成一个与解向量相同长度的浮点数向量，那么，在执行上，遗传空间就是问题空间，染色体直接反映了问题的规律与特性。编码机制包含两部分：排序部分S_k（用来描述任务执行顺序），和映射部分M_{ijk}（用来描述分配给每个任务的主机）。设k表示调度集中的调度数，映射M_{ijk}的顺序和任务的顺序是相称的。各种任务的执行时间由预测系统提供，并且与适应度函数（fitness function f_v）评价的任务目标相联系。

4.3.3 适应度函数值

适应度函数使用了组合的代价函数，它考虑到了总的完成时间makespan，空闲时间和期限。使用S_k和M_{ijk}直接计算调度k的总完成时间ω_k，设T_{jk}为根据编码机制的排序部分S_k重排过后的任务集，则

$$ts_{jk} = \max_{\forall l, M_{ljk}=1} \left(\max_{\forall p<j, M_{pjk}=1} (te_{pk}) \right)$$

$$te_{jk} = ts_{jk} + texe_{jk}$$

$$\omega_k = \max_{1 \le j \le m} \{te_{jk}\}$$

该算法与先来先服务算法的区别在于：（1）任务顺序根据S_k决定而不是根据任务的到达时间决定，所以不必考虑到达时间t_j；（2）使用M_{ijk}来确定对主机的选

择，$texe_{jk}$ 使用相应的资源模型直接计算预测，而在先来先服务中，需要考虑和比较所有可能的不同主机选择集 $\overline{H}$。

空闲时间的特征应该考虑进来，通过使用所有主机的平均空闲时间 ψ_k 表示：

$$\psi_k = \max_{1\le j\le m}(te_{jk}) - \min_{1\le j\le m}(ts_{jk}) - \frac{\sum_{j=1}^{m}\sum_{i=1}^{n} M_{ijk}(te_{jk} - ts_{jk})}{n}$$

调度前的空闲时间是不受欢迎的，因为它是首先被浪费的处理时间，最不可能被接下来的遗传算法的迭代发现，或者有更多的任务加入。有大量空闲时间的方案可以通过给 ψ_k 空闲时间的多余的包加以修正，主要修正早期的空闲时间，少量修正后期的空闲时间。

合约处罚 θ_k 由预期的期限 t_r 和任务完成时间 t_e 推出：

$$\theta_k = \frac{\sum_{j=1}^{m}(te_{jk} - tr_j)}{m}$$

调度 k 的代价值可以通过这些度量值推出，其影响由下列公式预先决定：

$$f^k = \frac{W^m\omega_k + W^l\varphi_k + W^c\theta_k}{W^m + W^l + W^c}$$

然后代价值使用动态扩展技术通过适当值进行规范化：

$$f_v^k = \frac{f_c^{\max} - f_c^k}{f_c^{\max} - f_c^{\min}}$$

其中 $f_c^{\max}$ 和 $f_c^{\min}$ 分别代表调度集中最好和最坏的代价值。

4.3.4 选择算子

遗传算法使用固定的入口大小和随机剩余选择，在两部分编码机制中采用了特殊的交叉和变化函数。交叉函数首先将两个排序的字符串以任意位置结合起来，然后重排这个字符串对，以产生合法的方案。映射部分通过下列步骤交叉：首先进行重排使其与它的任务顺序一致，然后执行单点交叉。需要重排是因为要保持各代之间节点映射和具体任务的相关。变化阶段也分为两部分，交换操作随机的用于排序部分，位翻转随机的用于映射部分。

以标准化级和分布规律随机对种群中的染色体进行选择，该方法是一种排序选择方法，以最佳染色体的选择概率作为基本参数，按染色体的排列序号确定其选择概率。选择机理仍然是适应值越大的染色体被选择的概率越大。操作

步骤如下：

步骤 1：确定选择概率 p_s。

步骤 2：计算标准分布值：

$$t=\frac{p_s}{1-(1-p_s)^P}$$

步骤 3：计算染色体的选择概率：

$$p_k=t(1-p_s)^{N(k)-1},\quad k=1,2,\cdots P$$

式中 $N(k)$——k 染色体的适应值在种群中按由大到小排列的序号。

步骤 4：计算染色体的累积选择概率值：

$$q_k=\sum_{j=1}^{k}p_j,\quad k=1,2,\cdots P$$

步骤 5：在[0，1]区间产生按升序排列的随机数序列 r。

步骤 6：对染色体进行选择：

```
fitIn=1;
newIn=1;
while newIn<=P
  if (r(newIn)<fit(fitIn))
       Newv(newIn)=oldv(fitIn);
       newIn=newIn+1;
  else
       fitIn=fitIn+1;
  end
end
```

4.3.5 杂交算子

采用均匀分布随机选择的方法选择交叉父代。父代以单点交叉的方式产生子代，参考[40]，我们采用新的交叉算子---类遗传杂交算子◊，如下：

令□(X_i)为 X_i 的杂交点，且记□$^+$(X_i)为 X_i 的杂交点之前的部分，□$^-$(X_i)为 X_i 的杂交点之后的部分。类遗传杂交算子◊是如下所示的三元算子：

◊(X_1,X_2,X_3)=X'_1,X'_2,X'_3，其中：

X_1:□$^+$(X_1) □$^-$(X_1)　　X'_1:□$^+$(X_1) □$^-$(X_3)

X_2:□$^+$(X_2) □$^-$(X_2)　　X'_2:□$^+$(X_2) □$^-$(X_1)

X_3:□$^+(X_3)$ □$^-(X_3)$ X'_3:□$^+(X_3)$ □$^-(X_2)$

具体步骤如下：

步骤 1：以交叉率 p_c 确定交叉操作的次数 n_c：

$$n_c=[\frac{p_cP}{2}]$$

步骤 2：在种群中均匀随机选择三个染色体中的任务顺序部分 v_{1l}、v_{2l}、v_{3l} $(l=1,2,\cdots n_c)$ 作为交叉父代。

步骤 3：在[0，n]区间产生随机数 r_l 作为交叉位置，n 为染色体长度。

步骤 4：根据类遗传杂交算子交叉运算计算产生后代 v'_{1l}，v'_{2l}、v'_{3l}。

此操作过程如图 4-3 所示，图中所示的杂交顺序只是为了表达方便起见假设的顺序，实际上三个父本杂交点之后的部分可以任意交换。

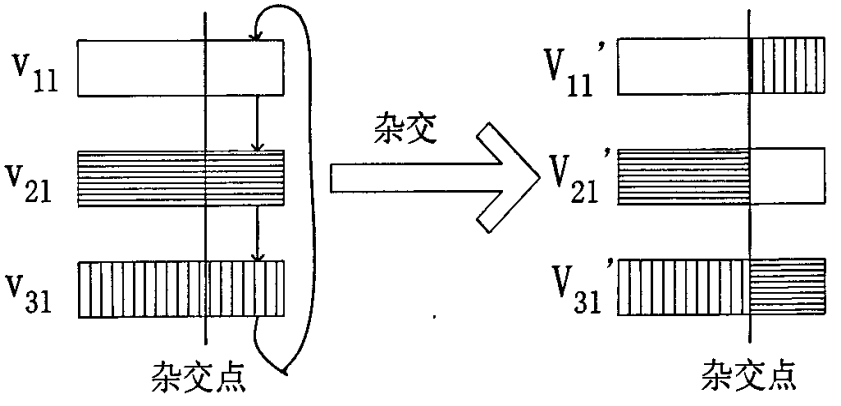

图 4-3 类遗传杂交算子

由于采用浮点数编码而不是二进制编码，交换之后，还要消除其中重复的基因，消除过程如下例所示：

设两个任务顺序为：

$l_A=[T_1,T_3,T_5,T_2,T_0,T_6,T_4,T_7]$

$l_B=[T_4,T_5,T_2,T_1,T_6,T_7,T_0,T_3]$

设选取的任意的交叉位置为 4，则交叉后生成新的任务顺序表为：

$$l'=[l_{A0},l_{A1},l_{A2},l_{A3},l_{B4},l_{B5},l_{B6},l_{B7}]$$
$$=[T_1,(T_3),T_5,T_2,T_6,T_7,T_0,(T_3)]$$

其中，l_{An} 表示 l_A 中的第 n 个元素，l_{Bn} 表示 l_B 中的第 n 个元素。

然后再查找新的任务顺序表中是否有重复的元素，可以发现有重复的元素 T_3，然后用没有在该顺序表中出现的任务序号替换重复出现的第一个位置元素。

以此类推，可替换掉所有的重复元素，生成一个合法的任务顺序表，如下：

$l' = [T_1, T_4, T_5, T_2, T_6, T_7, T_0, T_3]$

步骤5：重复步骤2~4，直到$l = n_c$为止。

参考文献[40]中已经证明，这种遗传算子具有全局收敛性，并且在搜索效率和搜索较优解方面与经典遗传算法相比都有明显的改善。

4.3.6 变异算子

遗传算法中的所谓变异运算，是指个体染色体编码串中的某些基因座上的基因值用该基因座的其他等位基因来替换，从而形成一个新的个体。变异运算是产生新个体的辅助方法，但它也是必不可少的一个运算步骤，因为它决定了遗传算法的局部搜索能力。在任务调度算法中可采用的变异算子有两种。(1)交换变异，是指相互交换个体编码串中两个选区的基因座之间的基因值，从而产生一个新的调度列表。(2)插入变异，是指先在个体编码串中选取两个基因座，然后再将其中的一个基因座插入到另一个基因座之后。具体采用何种变异算子以及具体如何实现还在探讨之中。

在现有的文献中，任务调度算法中可用的变异算子都为交换变异，即相互交换染色体中两个被选取的基因座之间的基因值，从而产生出一个新的调度列表。这种变异算子的最大缺点是容易产生无效染色体，这一缺陷降低了遗传算法在运行时特别是算法运行后期的局部搜索能力。在我们的算法中将采用一种被称为插入变异的变异算子，其方法是现在染色体中选取两个基因座，然后再将其中的一个基因座插入到另一个基因座之后或之前。如下所示：

$l' = [T_1, T_4, T_5, T_2, T_6, T_7, T_0, T_3] \Rightarrow$

$l' = [T_1, T_4, T_0, T_5, T_2, T_6, T_7, T_3]$

其中，基因座T_0插入到了基因座T_4之后。

变异操作的具体步骤如下：

步骤1：确定变异率p_m和形状系数b。

步骤2：计算变异操作的次数n_m：

$n_m = [p_m P]$

步骤3：在种群中按均匀分布随机选取染色体v_l $(l = 1, 2, \cdots n_m)$作为变异父代。

步骤4：计算进化标记t。

步骤5：由父代变异产生子代v_l'。

步骤6：重复步骤3~7，直到$l = n_m$为止。

第五章 基于动态遗传算法的网格任务调度算法的实现

5.1 仿真模型

5.1.1 网络仿真简介

1. 网络仿真技术简介：

网络仿真是一个很有用的网络研究工具，它以系统理论、形式化理论、随机过程和统计学理论、优化理论为基础。在设计阶段，仿真方法可提供一个虚拟模型来预测并比较各种方案的性能；运行阶段，通过对不同环境和工作负荷的分析和比较，来优化系统的性能。在某些情况下，仿真是唯一可行的方法和技术。仿真方法的抽象化程度比数学分析方法低，耗费的时间比测量技术少，其低成本和有效性是其他传统方法不可替代的。随着网络新技术的不断出现和数据网络的日趋复杂，对网络仿真技术的需求必将越来越迫切，网络仿真的应用也将越来越广泛，网络仿真技术已成为研究、规划、设计网络不可缺少的工具。网络仿真技术是一种通过建立网络设备、链路和协议模型，并模拟网络流量的传输，从而获取网络设计或优化所需要的网络性能数据的仿真技术。从应用的角度上看，网络仿真技术有以下特点：（1）全新的模拟试验机理，使其具有在高度复杂的网络环境下得到高可信度结果的特点。网络仿真的预测功能是其他任何方法都无法比拟的；（2）适用范围广，既可以用于现有网络的优化和扩容，也可以用于新网络的设计，而且特别适用于大中型网络的设计和优化；（3）初期应用成本不高，而且建好的网络模型可以延续使用，后期投资还会不断下降。

2. 网络仿真的一般步骤：

一个完整的仿真一般要经过问题定义、建模、模型确认、数据采集、程序编制和验证、模型运行和结果展示等一系列环节。通过问题定义，可以进一步的明确仿真的目的和要求。网络仿真模型是仿真系统的核心，它是对真实网络系统的简化，必须包含决定网络系统本质属性的主要因素及其逻辑关系，使模型具有很好的代表性。建模和模型的验证是两个技术性很强的工作。

模型确认一般有三种途径：专家分析；对模型假设的检验；初步方针结果与真实系统的一致性比较等。数据采集视网络仿真的一个重要方面。系统的性能不但取决于系统本身，还和系统上运行的工作负荷等多种因素有关；历史数据往往是仿真的数据来源之一，实时网络数据的采集可以为优化网络性能提供第一手的

资料，模拟产生的数据鼻息考虑到多种随机因素的综合作用，涉及到多类分布。早期的网络仿真系统多采用通用程序设计语言，由于计算机网络系统的复杂性和对数据的苛刻要求，使得设计难度非常之大，因此引发了对面向过程、面向事件和面向对象的仿真工具的开发。目前所实施的仿真，多采用此类工具。通过模型的运行和结果的展示，可以得到方针的结果。

实际上，上述过程是一个需要不断修改和完善的过程。需要指出的一点是：网络仿真模型并不是越详细越好，详细意味着需要更多的参数，无用的参数可能使仿真结果变得不准确。另外，详细的模型会使系统的处理效率下降。因此，在建模过程中，选择合理的仿真层次非常重要。

5.1.2 OPNET 网络仿真软件简介

OPNET 是MIL3 公司开发的一套集开发和应用为一体的通信系统模拟软件。目前全世界多家公司正在使用这套软件，包括美国军方和世界著名的电信公司。OPNET是一种功能十分强大的网络仿真平台，支持在网络各个层次的设备、链路和协议的精确建模，并提供丰富的外界开发接口，可以无与伦比的灵活性用于设计和研究通信网络、设备、协议和应用。OPNET采用面向对象的建模方法和图形化的编辑器，反映了实际网络和网络组件的结构，因此实际的系统可以直观地映射到模型中。

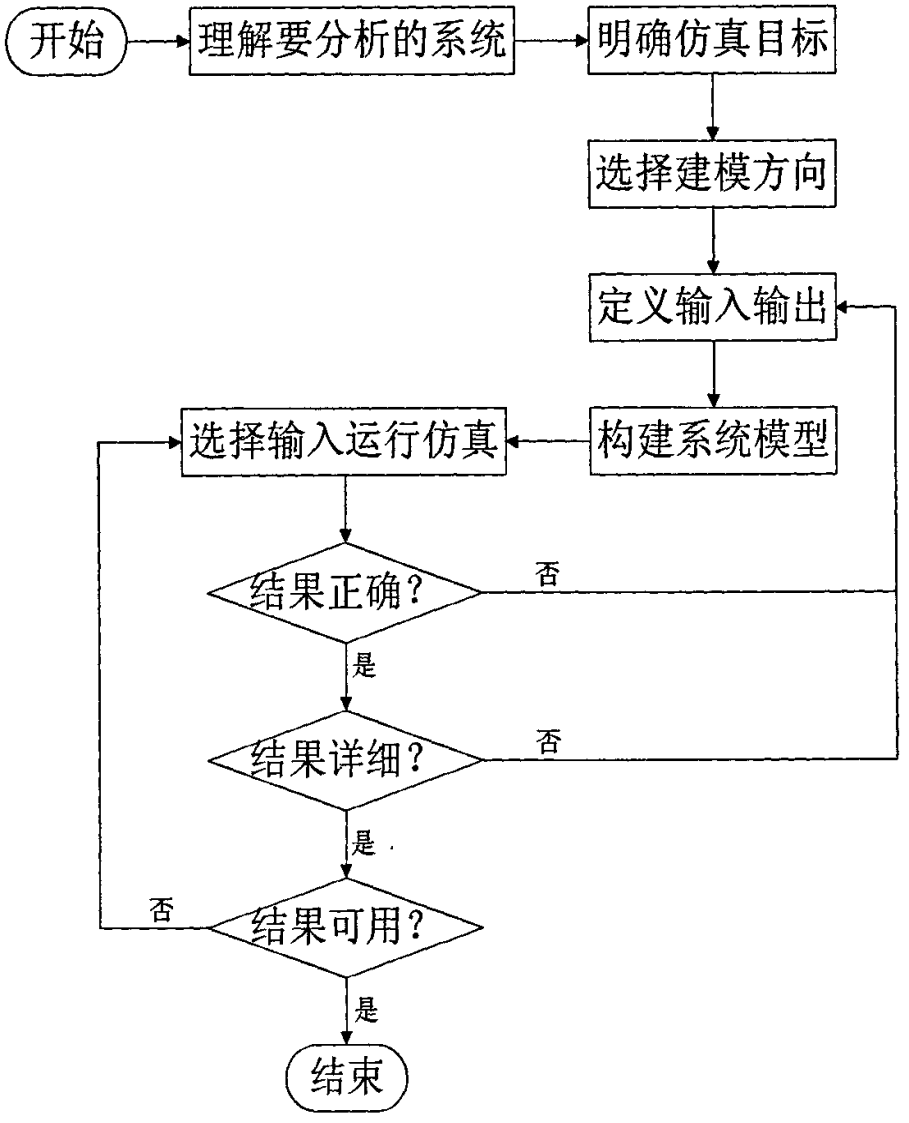

图5-1 OPNET仿真流程

OPNET Technology 公司的仿真软件 OPNET 具有下面的突出特点，使其能够满足大型复杂网络的仿真需要：

1. 提供三层建模机制，最底层为 Process 模型，以状态机来描述协议；其次为 Node 模型，由相应的协议模型构成，反映设备特性；最上层为网络模型。三层模型和实际的网络、设备、协议层次完全对应，全面反映了网络的相关特性；

2. 提供了一个比较齐全的的基本模型库，包括：路由器、交换机、服务器、客户机、ATM 设备、DSL 设备、ISDN 设备等等；

3.采用离散事件驱动的模拟机理（discrete event driven），与时间驱动相比，计算效率得到很大提高。

4.采用混合建模机制，把基于包的分析方法和基于统计的数学建模方法结合起来，既可得到非常细节的模拟结果，也大大提高了仿真效率。

5. OPNET 具有丰富的统计量收集和分析功能。它可以直接收集常用的各个网络层次的性能统计参数，能够方便地编制和输出仿真报告。

6.提供了和网管系统、流量监测系统的接口，能够方便的利用现有的拓扑和流量数据建立仿真模型，同时还可对仿真结果进行验证。

5.1.3 局部网格仿真模型

本文只做了局部网格结构的仿真模型。在设计和实现此仿真环境的过程中，充分考虑了具体应用情形，对实现结构进行了细致地模块化划分，其主要部分由两个模块构成：智能 Agent（包括仿真数据提供模块、算法控制模块）和节点计算模块 Node。其中，仿真数据提供模块模拟添加网格任务，算法控制模块模拟算法调度，节点计算模块接收仿真数据提供模块提交的任务，模拟任务执行。

该网格任务调度系统结构相当于排队论中研究的 M/M/m 队列，任务的服务时间是指数分布的，任务的达到时间间隔也是指数分布的，本文为了研究方便，只暂时设定 12 个服务器（m=12），即有 12 个计算节点，如图 5-2 局部网格结构。

为了符合网格计算环境的特点，每个计算节点的速度都在不停的随机变化，计算节点本身的状态也在变化，随时可能加入或者离开网格系统，如果节点离开网格系统，状态为不可用。离开的计算节点要加入网格，状态由不可用变为可用。这些计算节点互相不影响，之间并没有通信，计算节点只与 Agent 进行通信，独立接受并完成任务。接收到任务以后，节点状态由空闲变为忙碌，一个节点一次只能执行一个任务，在节点处并不存在任务队列。在任务完成以后，节点会将任务完成结果提交给 Agent，同时每当计算节点改变速度、状态或者任务完成的时候都会向 Agent 发送一个报告。

Agent 负责接收任务源的用户提交的任务，任务长度呈指数变化，任务到达时间间隔指数分布。程序设定任务属性如任务长度和任务截止时间，二者都是指数分布，可以证明指数分布的最大值是平均值的三倍。Agent 根据任务长度和截止时间按照调度算法分配给网格中的计算节点，每隔一段不同的时间，节点速度会变化，随时会加入和离开网格系统。计算节点定时向 Agent 分别报告状态和速度，Agent 会根据节点的报告来时时更新计算资源的数据库。

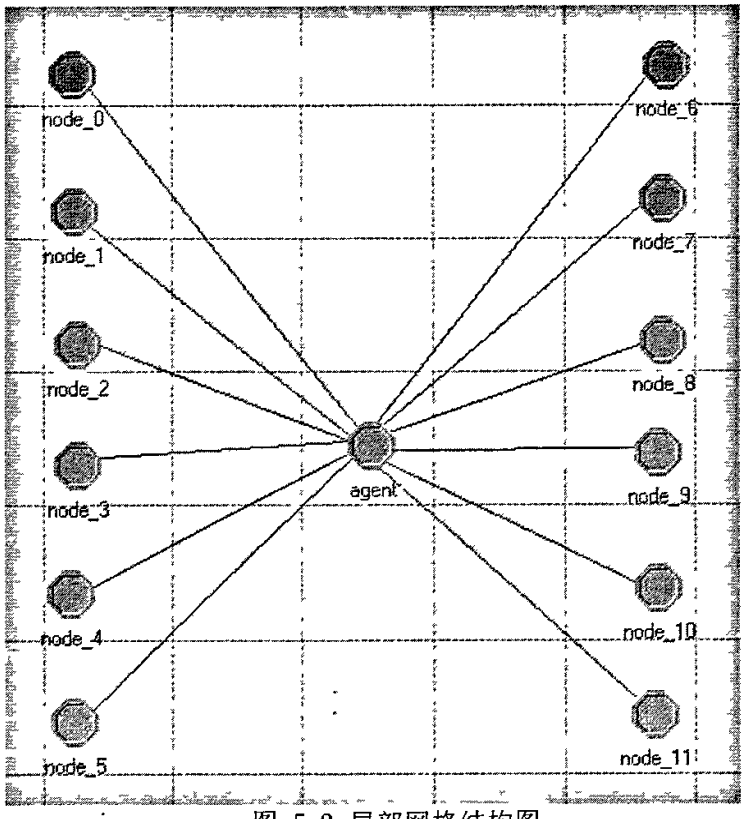

图 5-2 局部网格结构图

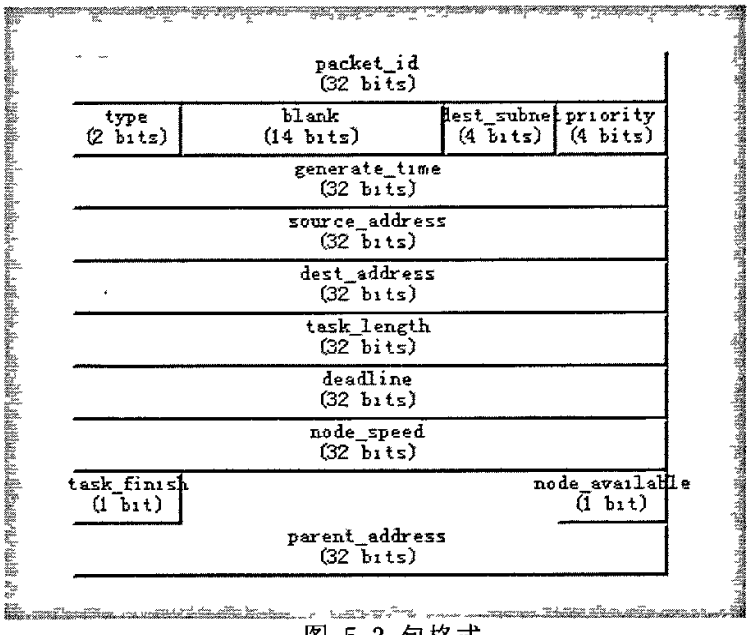

图 5-3 包格式

OPNET 采用基于包的建模机制，本文仿真模型中包格式如图 5-3。

包格式中 type 字段说明包的种类，一共有四种包括任务包（00）、结果包（01）、速度包（10）和状态包（11）。任务包中字段 task_length 表示任务的长度，deadline 表示任务的截止时间，priority 表示任务的优先级别，generate_time 表示任务提交时间；结果包中字段 task_finish 表示任务是否成功返回；速度包中字段 node_speed 表示计算节点速度改变值；状态包中字段 node_available 表示计算节点的状态是否可用。共用字段 parent_address 表示上层 Agent 地址，source_address 表示包发送的源地址，dest_address 表示包发送的目的地址。

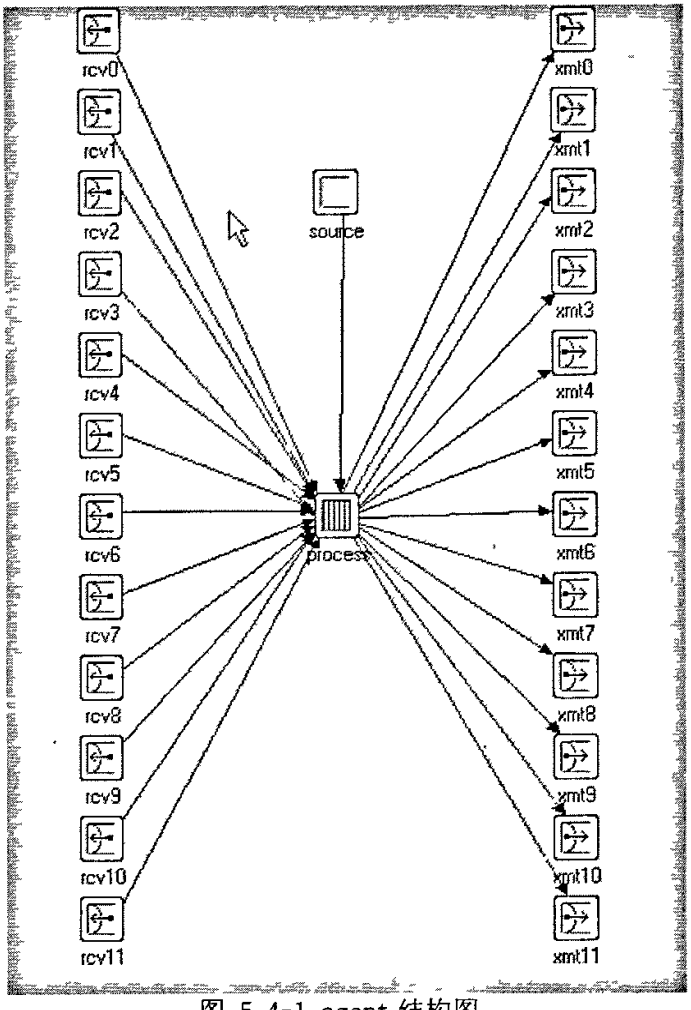

图 5-4-1 agent 结构图

如图 5-4-1、5-4-2 所示，在本文的仿真模型中，Agent 包含仿真数据提供模块 source 和算法控制模块 process 两个模块，以及与 12 个计算节点收发包的接口。

模块 source 生成任务包，并同时规定了任务包的大小和任务截止时间，这就好比从用户那里接收到提交上来的任务。任务包生成以后，向 process 模块发包，因为两个模块同属于 Agent 结构，所以包发送过程中没有延迟，任务产生源地址一律设定为特定地址 100。

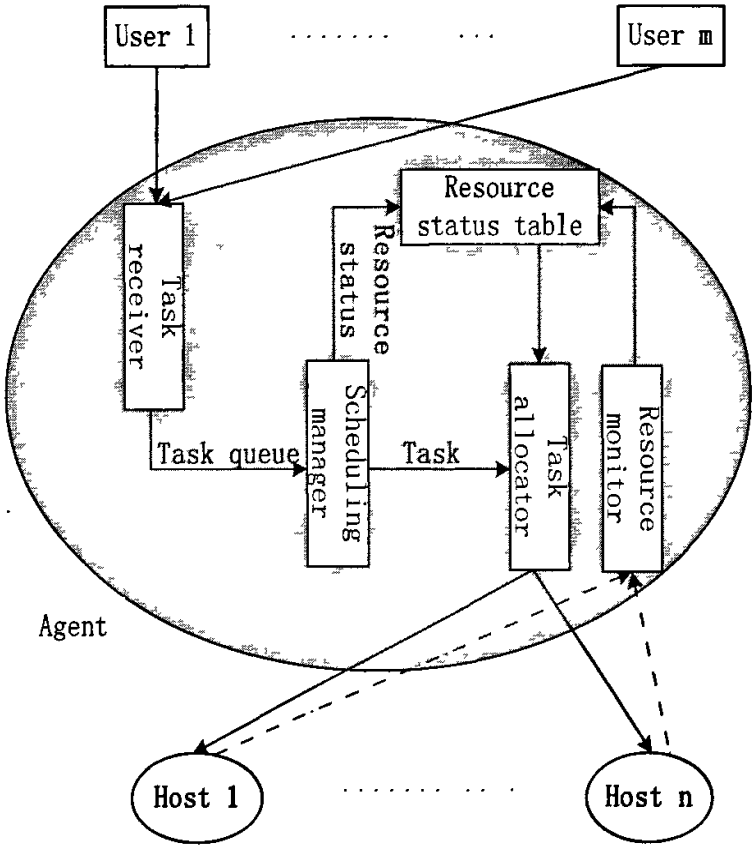

图 5-4-2 agent 功能图

图 5-5 是模块 source 状态转移图，模块在初始化（init）状态下，读取模块的属性，设置统计量，然后 OPNET 调用自中断来开始任务包的生成。在产生包（generate）状态下，生成任务包并同时规定了任务包的大小（pksize）和任务截止时间（deadline），暂时设定任务截止时间为平均任务执行时间的 10 倍。在 source 模块中，可供观测的统计量有任务包长度（packet length）和任务到达时间间隔（inter arrival time）。以下是包产生一段程序：

```
static void ss_packet_generate (void)
{
    Packet*         pkptr;                          //包指针
    int             pksize;                         //任务长度
```

```
    int             priority;                       //优先级别
    double          generate_time;                  //包产生时间
//任务包长指数分布，平均值为1024000 bit
    pksize = (int)op_dist_exponential (1024000);
    priority = (int)op_dist_uniform (5);
    pkptr = op_pk_create_fmt ("packet_format_xue");//创建包格式
    generate_time = op_pk_creation_time_get (pkptr);
    op_pk_nfd_set(pkptr,"task_length",pksize);
    op_pk_nfd_set(pkptr,"source_address",100);
//任务截止时间是任务平均执行时间的10倍
    op_pk_nfd_set(pkptr,"deadline",(double)op_dist_exponential
(10*pksize / 512000));
    op_pk_nfd_set(pkptr,"generate_time",generate_time);
    op_pk_nfd_set(pkptr,"priority",priority);
//设置仿真统计量，任务包长度和任务到达时间间隔
    op_stat_write (packets_sent_hndl, 1.0);
    op_stat_write (packets_sent_hndl, 0.0);
    op_stat_write (bits_sent_hndl, (double) pksize);
    op_stat_write (bits_sent_hndl, 0.0);
    op_stat_write (packet_size_hndl, (double) pksize);
    op_stat_write (interarrivals_hndl, next_intarr_time);
    op_pk_send (pkptr, 0);//发包
    }
```

算法控制模块Process较为复杂，也是最重要的一个模块，调度过程最终在这个模块里面完成。由于网格计算相关技术甚至模型本身仍处于迅速发展的过程之中，因而对网格任务调度算法本身运行效率的评价，当前尚无成熟的模型与基准，难以进行横向对比。所以，论文在算法效率的本身的两个重要方面进行分析与评价，任务执行的平均延迟（Average delay）和超时概率（Tardy rate）。Average delay是从任务产生到任务成功返回的时间延迟，其中包括任务的排队时间、传输时间和执行时间。Tardy rate是任务超时（未能在截止时间之间完成）的个数与任务成功完成总数的比值，即在完成的任务中，超时任务所占比例。

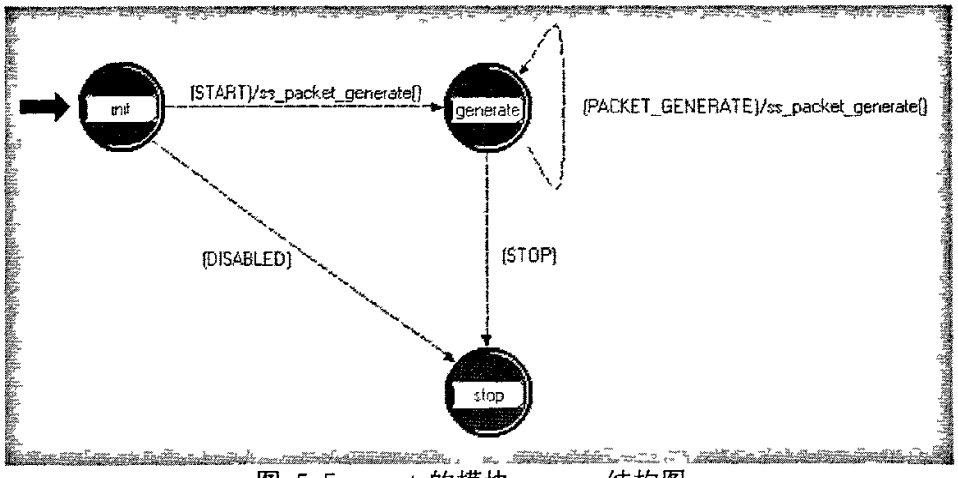

图 5-5 agent 的模块 source 结构图

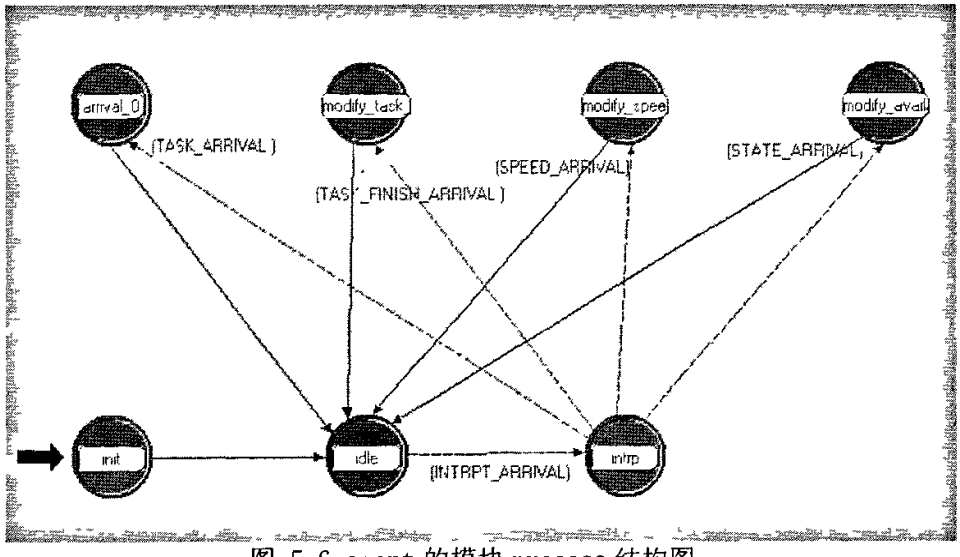

图 5-6 agent 的模块 process 结构图

图 5-6 是模块 process 的状态转移图，模块在初始化（init）阶段定义了一个任务队列和一个计算资源队列，包含 12 个计算节点的属性值，计算节点的地址（dest_address）、可用性（available & unavailable）、状态（busy & free）和计算速度（node_speed）。

```
//定义一个计算资源队列
int j;
for ( j = 0; j < HOST_NUMBER; j++)
   {
   host_queue[j].id=j;
   host_queue[j].available=0;//可用
```

```
    host_queue[j].state=0;//空闲
    host_queue[j].speed=512000;
    }
//设置统计量
average_delay = op_stat_reg ("AVERAGE DELAY",
                OPC_STAT_INDEX_NONE,OPC_STAT_GLOBAL);
tardy_rate = op_stat_reg ("TARDY RATE",
                OPC_STAT_INDEX_NONE,OPC_STAT_GLOBAL);
//刷新任务队列
op_subq_flush(0);
```

初始化之后进入调度（schedule）状态，当任务队列空的时候，系统什么都不做（这种情况很少出现）；在队列不为空的时候，尽量将任务调度到优先级较高的计算资源上去运行，而优先级主要由计算资源的能力和稳定性来决定，所以，这也进一步提高了任务能够成功执行的概率。

```
if (Q_NOT_EMPTY)//任务队列不空
  {
  int k=-1;
  int i;
  int temp_speed=-1;
  for (i=0; i< HOST_NUMBER; i++)
      { //检查有没有空闲可用的计算节点
      if (host_queue[i].available==0 && host_queue[i].state==0)
      if (host_queue[i].speed > temp_speed)
        {
        temp_speed = host_queue[i].speed;
        k=i;}
      }//挑选出计算速度最快的计算节点
      if (k!=-1)
        { //将任务调度到最快的计算节点上去运行
        host_queue[k].state=1;
        q_pkptr = op_subq_pk_remove (0, OPC_QPOS_TAIL);
        op_pk_nfd_set(q_pkptr,"dest_address",k);
        op_pk_send(q_pkptr, k);
        }
}
```

在调度阶段，每当有包到达触发中断，模块随时响应中断（intrp），包中断分为四种形式，响应的处理分别为：

1） 模块到达（TASK_ARRIVAL）。先进先出FCFS策略只是简单的把任务包加入任务队列的队尾，什么都不做直接返回调度状态；短任务优先SJF策略会读取任务

包属性，将任务按照长短排序；早截止时间任务优先EDF策略则是将任务按照截止时间的早晚排序。

```
//读取任务包属性
op_pk_nfd_get (pkptr,"task_length",&pk_length);
op_pk_nfd_get(pkptr,"priority",&priority);
op_pk_nfd_get(pkptr,"deadline",&deadline);
op_pk_nfd_get(pkptr,"generate_time", &generate_time);
/*先进先出策略下，将任务包加入任务队列的队尾 */
/* if (op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL) != OPC_QINS_OK)
    {
    insert_ok = 0;//插入队列失败
    }
else{
    insert_ok = 1;
    } */
```

2）结果包从计算节点到达（TASK_FINISH_ARRIVAL）。将对应的计算节点从忙碌状态设为空闲状态，检查任务是否成功执行，如果未能成功执行需要重新进行调度，如果执行成功，根据返回任务的执行结果、产生时间和截止时间计算任务的延迟时间（delay）和是否超时（tardy），写入仿真统计量并返回调度状态；

```
//读取结果包属性
op_pk_nfd_get(pkptr,"task_finish",&task_finish);
op_pk_nfd_get(pkptr,"deadline",&deadline);
op_pk_nfd_get(pkptr,"source_address",&source_address);
op_pk_nfd_get(pkptr,"generate_time", &generate_time);
op_pk_nfd_get(pkptr,"dest_address",&stamp_time);
//将对应的计算节点从忙碌状态设为空闲状态
host_queue[source_address].state = 0;
if (task_finish == 0)//任务成功返回
    { //计算仿真统计量
    task_finish_count ++;
    task_delay = op_sim_time() - generate_time;
    total_delay_time = total_delay_time + task_delay;
    delay_average = total_delay_time / task_finish_count;
    op_stat_write (average_delay,task_delay);
    if(task_delay > deadline)    timeout ++;
    tardy = (double) timeout / task_finish_count;
    op_stat_write (tardy_rate,tardy);
    }
```

3）状态包从节点到达（STATE_ARRIVAL）。根据包中信息，改变计算资源队列中的相应计算节点状态，返回调度状态；

```
//读取状态包属性
int j;
op_pk_nfd_get(pkptr,"source_address",&j);
op_pk_nfd_get(pkptr,"node_available",&node_available);
host_queue[j].available = node_available;
```

4）速度包从节点到达（SPEED_ARRIVAL）。根据包中信息，改变计算资源队列中的相应计算节点速度，返回调度状态；

```
//读取速度包属性
int k;
op_pk_nfd_get(pkptr, "source_address", &k);
op_pk_nfd_get(pkptr, "node_speed", &node_speed);
host_queue[k].speed = node_speed;
```

模块 process 中可观测的统计量有任务执行的平均延迟（Average delay）和超时概率（Tardy rate），还有系统自带的统计量任务队列长度（Queue size）以及 Agent 与每个计算节点接收（RCV）和发送（XMT）的吞吐量。另外在 Agent 结构中，链路设置为双工传输，传输速率（data rate）为 1Mbps，错误比特率为零，相对应计算节点也设置相同。

本文仿真模型中计算节点 Node 模块如图 5-7 所示，node 的模块 process 结构图如图 5-8 所示，初始化（init）阶段进入空闲（idle）状态，没有任务系统处在空闲状态，每当有包到达触发中断，模块随时响应中断。这一模块可供观测的统计量有点到点延迟(ETE delay)和计算节点接收(RCV)和发送(XMT)的吞吐量。ETE delay 是在 Agent 队列内排队时间以及从 Agent 到计算节点的传输时间之和，不包括任务执行时间。

包中断分为四种形式，响应的处理分别为：

1) 任务包从 Agent 到达（RCV_ARRVL_INTRPT）。将机器状态由空闲设置为忙碌，根据当前计算节点速度计算任务执行时间，以及点到点延迟（ETE delay），然后发出任务结果包，设置计算节点状态为空闲，返回空闲（idle）状态；

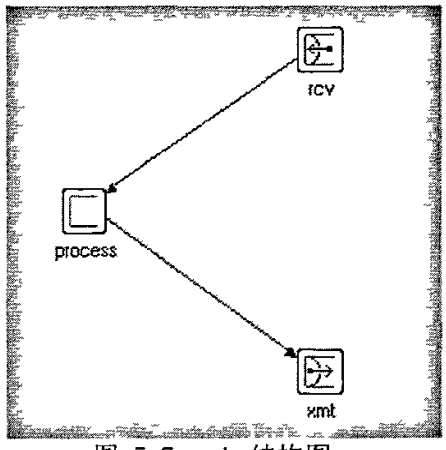

图 5-7 node 结构图

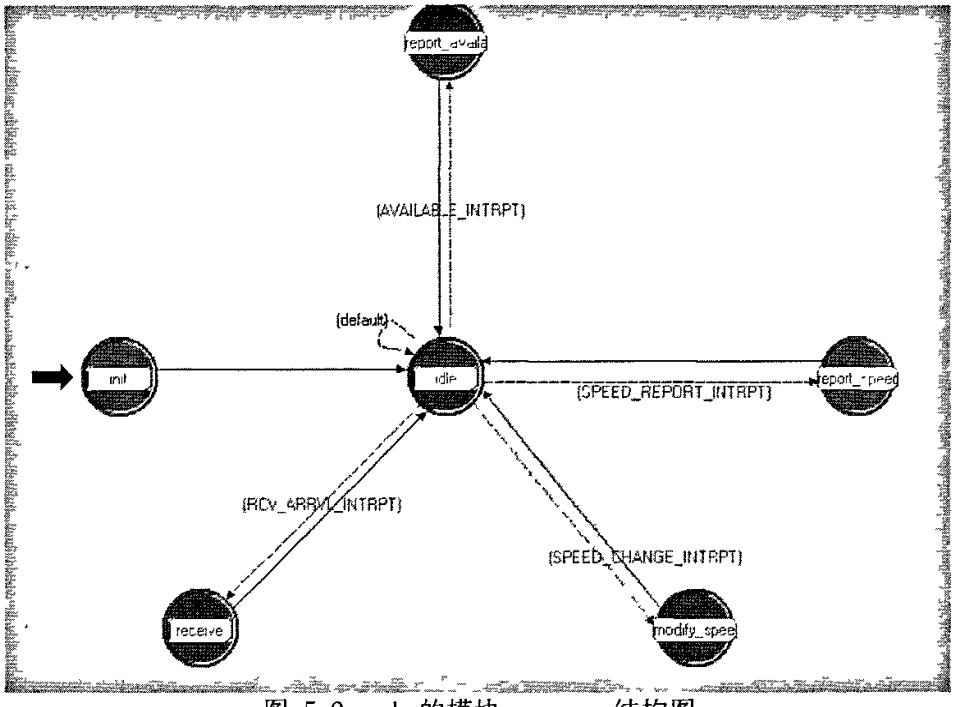

图 5-8 node 的模块 process 结构图

```
Packet * pkptr;
Packet * f_pkptr;
int pk_length;
double pk_svc_time;//任务执行时间
double delay_ete;
double generate_time;
server_state = SERVER_BUSY;
pkptr = op_pk_get (RCV_IN_STRM);
op_pk_nfd_get (pkptr, "task_length", &pk_length);
```

```
op_pk_nfd_get (pkptr, "deadline", &deadline);
op_pk_nfd_get (pkptr, "generate_time", &generate_time);
op_pk_nfd_get (pkptr, "dest_address", &dest_address);
pk_svc_time = (double) pk_length / server_speed;
delay_ete = op_sim_time () - generate_time;
op_stat_write (ete_gsh,delay_ete);
op_pk_destroy (pkptr);
//创建任务完成结果包
f_pkptr = op_pk_create_fmt ("packet_format_xue");
op_pk_nfd_set(f_pkptr,"type",TASK_FINISH);
op_pk_nfd_set(f_pkptr,"source_address",dest_address);
op_pk_nfd_set(f_pkptr,"task_finish",TASK_IS_FINISH);
op_pk_nfd_set(f_pkptr,"deadline",deadline);
op_pk_nfd_set(f_pkptr,"generate_time",generate_time);
op_pk_send_delayed (f_pkptr, 0, pk_svc_time);
server_state = SERVER_FREE;
```

2) 速度改变中断到达（SPEED_CHANGE_INTRPT）。速度改变频率设置为指数分布，平均值为300s，速度改变以后返回空闲状态；

```
//计算节点速度的改变
server_speed = speed[n];
n++;
//速度改变频率的设置
next_intarr_time = op_dist_exponential(300);
sim_current_time = sim_current_time + next_intarr_time;
op_intrpt_schedule_self (sim_current_time, NODE_SPEED);
```

3) 速度报告中断到达（SPEED_REPORT_INTRPT）。创建速度包，每50s向Agent报告一次计算节点的速度，返回空闲状态；

```
Packet*  pkptr;  //创建速度包
pkptr = op_pk_create_fmt ("packet_format_xue");
op_pk_nfd_set(pkptr,"type",NODE_SPEED);
op_pk_nfd_set(pkptr,"source_address",source_address);
op_pk_nfd_set(pkptr,"node_speed",server_speed);
op_pk_send (pkptr, 0);
```

4) 状态改变中断到达（AVAILABLE_INTRPT）。状态变化间隔时间设置为指数分布，平均值 2000s，每次状态的改变设定以 90%概率可用，10%概率不可用。然后向Agent发送状态包，返回空闲状态；

```
Packet*  pkptr;
Distribution*  next_dist;
//加入贝努利概率分布函数
next_dist = op_dist_load ("bernoulli", 0.1,0.0);
```

```
if (next_dist != OPC_NIL)
available = op_dist_outcome (next_dist);
//创建状态包
pkptr = op_pk_create_fmt ("packet_format_xue");
op_pk_nfd_set(pkptr,"type",NODE_AVAILABLE);
op_pk_nfd_set(pkptr,"source_address",source_address);
op_pk_nfd_set(pkptr,"node_available",available);
op_pk_send (pkptr, 0);
```

5.2 其他常见任务调度算法详细设计

5.2.1 FCFS 算法

假定一个网格资源具有n个主机，每个主机H_i具有它自己的类型ty_i，资源模型可以用来描述这个主机的性能信息：

$$H=\{H_i \mid i=1,2,.....,n\}$$
$$ty=\{ty_i \mid i=1,2,.....,n\}$$

设m为任务数，任务T_j的到达时间为t_j，PACE 应用模型tm_j可以用来描述每个任务的应用性能级别信息，每个任务完成的期限为tr_j，开始时间为ts_j，结束时间为te_j，任务可表示为：

$$T=\{T_j \mid j=1,2,\cdots m\}$$

$$(t,tm,tr,ts,te)=\{(t_j,tm_j,tr_j,ts_j,te_j) \mid j=1,2,\cdots m\}$$

MT_j是分配给任务T_j的主机的集合：

$$MT=\{MT_j \mid j=1,2,.....,m\}$$

$$MT_j=\{H_{i_l} \mid l=1,2,.....,l_j\}$$

其中l_j是分配给任务T_j的主机数，M是一个描述主机和任务之间映射关系的二维数组：

$$M=\{M_{ij} \mid i=1,2,.....,n;\ j=1,2,.....m\}$$

$$M_{ij}=\begin{cases}1 & if\ H_i \in MT_j \\ 0 & if\ H_i \notin MT_j\end{cases}$$

可根据应用模型tm_j和资源模型ty产生性能预测信息。选择的合适的主机子集$\overline{H}$（不能为空集）可评价和表示为：

$$\forall \overline{H} \subseteq H, \overline{H} \neq \Phi, \overline{ty} \neq \Phi, \overline{texe_j}=eval(\overline{ty},tm_j)$$

局部 scheduler 管理函数的功能是发现每个任务完成的最早可能时间，同时保持任务到达的序列顺序：

$$te_j = \min_{\forall \bar{H} \subseteq H, \bar{H} \neq \Phi} (\overline{te_j})$$

一个任务可能分配到任何主机集，scheduler 用来考察所有这些可能性，并且选择其中最早任务完成时间，其中完成时间等于最早可能开始时间加上完成时间，即：

$$\overline{te_j} = \overline{ts_j} + \overline{texe_j}$$

$$\overline{ts_j} = \max(t_j, \max_{\forall l, H_l \in \bar{H}} (td_{lj}))$$

其中 td_{lj} 是 t_j 时主机 H_l 的最晚空余时间，它等于任务 T_j 到达前分配给主机 H_l 的最大完成时间，即

$$\overline{ts_j} = \max_{\forall p \prec j, M_{lp}=1} (te_p)$$

总之，

$$te_j = \min_{\forall \bar{H} \subseteq H, \bar{H} \neq \Phi} (\max(t_j, \max_{\forall l, H_l \in \bar{H}} (\max_{\forall p \prec j, M_{lp}=1} (te_p))) + \overline{texe_j})$$

并非把所有的主机都调度到一个任务上会取得好的性能，有时如果选定的处理器数少，任务开始执行的时间可能早一些，有时一些任务当被分配的主机任务多时反而执行时间长。上述算法的复杂性取决于可能选择的主机数，即

$$C_n^1 + C_n^2 + \cdots + C_n^n = 2^n - 1$$

显然，如果一个网格资源的主机数增加调度的复杂性会按指数增加。该算法是先进先服务策略，任务到达的顺序决定了任务的执行情况。重新对任务集进行排序可能改善任务的执行情况，但会增加算法的复杂性。这一矛盾可以使用迭代的启发式算法解决。

5.2.2 Min-Min 算法

Min-min算法如下：

(1) for all tasks *ti* in meta-task *Mv* (in an arbitrary order)

(2)　for all hosts *mj* (in a fixed arbitrary order)

(3)　　$CTij = ETij + dj$

(4) do until all tasks in *Mv* are mapped

(5)　for each task in *Mv*, find the earliest completion time and the host that obtains it

(6) find the task *tk* with the minimum earliest completion time
(7) assign task *tk* to the host *ml* that gives it the earliest completion time
(8) delete task *tk* from *Mv*
(9) update *dl*
(10) update *CTil* for all *i*
(11)end do

算法中 *ETij* 表示任务 *ti* 在主机 *mj* 上的期待执行时间，*dj* 的下一个可用时间，*CTij* 表示任务 *ti* 在主机 *mj* 上的期待完成时间。

5.2.3 Max-min算法

Max-min算法如下：
(1) for all tasks *ti* in meta-task *Mv* (in an arbitrary order)
(2) for all hosts *mj* (in a fixed arbitrary order)
(3) $CTij = ETij + dj$
(4) do until all tasks in *Mv* are mapped
(5) for each task in *Mv*, find the maximum completion time and the host that obtains it
(6) find the task *tk* with the minimum earliest completion time
(7) assign task *tk* to the host *ml* that gives it the earliest completion time
(8) delete task *tk* from *Mv*
(9) update *dl*
(10) update *CTil* for all *i*
(11)end do

算法中 *ETij* 表示任务 *ti* 在主机 *mj* 上的期待执行时间，*dj* 的下一个可用时间，*CTij* 表示任务 *ti* 在主机 *mj* 上的期待完成时间。

5.3 实验参数及仿真结果

5.3.1 实验参数

任务的长度length指的是任务中的指令数，处理器的速度指的是每时间单位计算的指令数。网格是异构的，所以网格中的处理器本质上具有不同的速度，而且，由于最初用户所使用的公共资源的负载影响，每个处理器的速度也是随着时间变化的，即每个处理器的速度是最初用户没有使用的处理器的多余的能力，并且是专门用于网格的。设task_length表示任务的长度，假定task_length是满足均值为1024000的指数分布的随机变量。设 $S_{p,t}$ 表示在时间间隔[t, t+1] 处理

器p的速度，其中t为非负整数。我们假定没有产生损耗，在足够短的单位时间内每个处理器的速度不变；假定$s_{p,t}$是满足均值为512000的指数分布的随机变量，如果最初用户的负载很重或处理器失效，那么$s_{p,t}$为0。简单的说，本文中没有考虑处理器的增加、删除和错误。设deadline表示任务的截止时间要求，假定deadline=5*task_length。

这里有许多描述任务调度的性能指标，选取哪个性能指标主要取决于系统要求。我们主要考虑了两个性能指标来刻画网格任务调度算法的性能。主要是average delay 和 tardy rate。在一段时间 t 内，N 个独立的任务被调度到具有 M 个主机的网格上，利用每个任务 T_j的参数（截止时间δ_j和完成时间η_j）定义性能指标如下：

Tardy rate：

$$\gamma = \frac{\sum_{j=1}^{N} \Delta(d_j - \eta_j)}{N}$$

Average delay：

$$\overline{T} = \frac{\sum_{j=1}^{N} (\eta_j - g_j)}{N}$$

其中 $\Delta(x) = \begin{cases} 1, & if \quad x \geq 0 \\ 0, & if \quad x < 0 \end{cases}$。

5.3.2 实验仿真结果及分析

从图 5-9 和图 5-10 可以看出，动态遗传算法的平均延迟时间要小于 minin 算法，因为动态遗传算法是批处理算法，一次可以处理等待队列中的所有任务，而 minmin 算法要每次只调度一个任务，并且要消耗一定量的调度时间。同时由于先进先出算法不需要大量的调度时间，所以在负载轻的情况下，平均延迟时间最短。但是由于先进先出算法只是盲目的把等待队列中的第一个任务调度到任意一个空闲的主机上，没有考虑性能因素，所以超时概率最大。动态遗传算法考虑到了截止期限的约束条件，每次调度时都尽量把等待队列中的所有任务分配到合适的空闲主机上去执行，并且根据估计参数要满足截止期限的要求，但是由于在实际主机执行任务的过程中，主机的执行速度等性能是不断变化的，而且计算调度方案的花销较大，所以也有可能出现超时的现象。Minmin 和 Maxmin 算法的超时概率要小于前两种算法。总的来说，动态遗传算法是一种批处理算法，并且能够满足网格环境中动态的特性，综合性能指标比较好。而且，还可以对该算法进行进一步的扩展，引入优先级，使用更加好的预测机制，该算法具有很大的潜力。

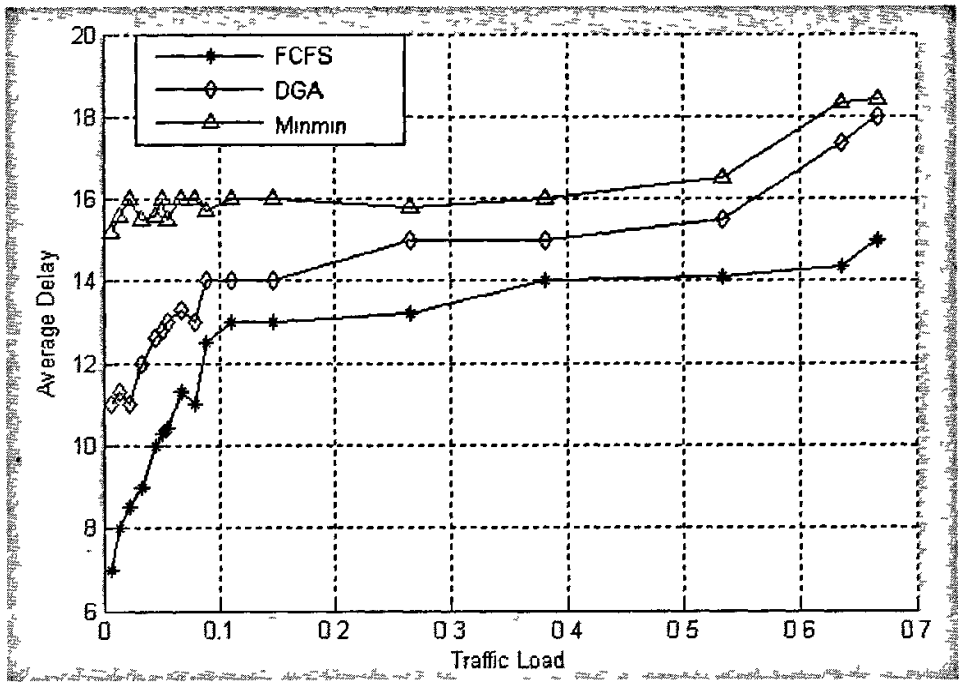

图 5-9 平均延迟时间

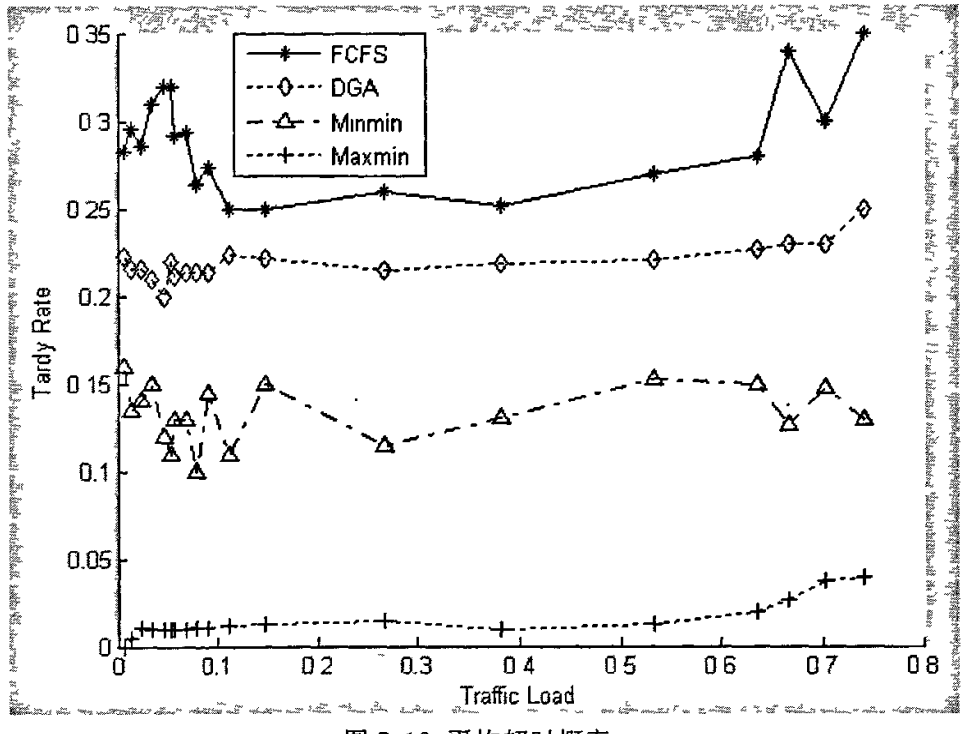

图 5-10 平均超时概率

第六章 总结与展望

任务调度是网格计算中一个至关重要的问题，其算法将直接影响到网格环境中任务执行的效率以至成败。用户通过向网格系统提交计算任务来共享网格资源，网格调度程序再按照某种策略把这些任务分配给合适的资源。高效的调度算法可以充分利用网格系统的处理能力，从而提高应用程序的性能。

传统的调度算法主要是静态调度算法，没有考虑到动态性。有的即便考虑到了动态性，也只是在给定的任务集中，讨论对该任务集执行调度的性能情况，没有从系统的角度去分析。

本文提出了一种遗传算法的改进算法——动态遗传算法 DGA（Dynamic Genetic algorithm），根据网格系统各服务节点的计算能力、负载及网络状态进行动态调度，从而向用户提供最优性能，不仅使总的完成时间最短，还尽量考虑到使主机的空闲时间最短，同时要满足每个任务的 deadline 的要求。在系统级主要提出了两个评价指标：tardy rate 和 average delay。动态遗传算法主要是在基本遗传算法的基础上，针对网格任务调度的动态特性，提出了新的编码机制和适应度函数，能够很好的适应空闲主机数量和等待调度任务数量的动态变化。根据新的编码机制，重新设计了选择算子、交叉算子和变异算子。

在 OPNET 环境中构建了一个局部网格的仿真模型，对所提出的动态遗传算法进行了仿真实现，并与常见的其他网格任务调度算法（如Min-min、Max-min、Deadlin_first，FCFS 等）进行了对比，试验结果表明，动态遗传算法具有很好的优化能力，提供了较好的服务质量。

在以后的工作中，还需要探索以下方面：

（1） 如何实现遗传算法和模拟退火算法相结合是一个新的方向，能够很快提高收敛的速度。

（2） 如何在遗传算法中实现扩展性的问题，还需要进行深入的研究。

（3） 本文中对主机性能的预测比较简单，目前有人提出了 PACE、GHS 等预测模型，效果很好，可以加以借鉴。

（4） 人工智能技术应用于任务调度也是最近兴起的比较热的方向，应考虑如何将 agent 技术引进到调度模型中，更好的实现智能调度。

参考文献

[1] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. of Supercomputing Applications, 2001

[2] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications. USA: Sage Publications, 1997 11(2). 115-128

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

[4] Rajkumar Buyya. Economic-based Distributed Resource Management and Scheduling for Grid Computing. Doctor of Philosophy.Monash University. 2002

[5] Dongarra, J. Netsolve and its application. Network Computing and Applications, 2001. NCA 2001. IEEE International Symposium on, 8-10 Oct. 2001. Pages:21 – 21

[6] Su A., Berman F., Wolski R. and Strout M.M. Using AppLes to schedule simple SARA on the computational grid. International Journal of High Performance Computing Applications, v 13, n 3. Fall 1999. 253-62

[7] Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S. Condor-G: a computation management agent for multi-institutional grids. High Performance Distributed Computing, 2001. Pages:55 - 63

[8] Guojie Li, Zhiwei Xu, Yuan Wang, Wei Li. The Vega Grid and autonomous decentralized systems. Autonomous Decentralized System, 2002. The 2nd International Workshop on, 6-7 Nov. 2002. Pages:2 – 7

[9] http://people.cs.uchicago.edu/~krangana/ChicSim.html

[10] W. E. Johnston, D. Gannon, B. Nitzberg. Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. Eighth IEEE International Symposium on High Performance Distributed Computing, Redondo Beach, CA, Aug. 1999

[11] R.Buyya, H.Stockinger et al. Economic models for resource management and scheduling in Grid computing. The Journal of Concurrency and Computation: Practice and Experience (CCPE) Special issue on Grid computing environments, 2002

[12] K. Nowinski, B. Lesyng, M. Niezgódka, P. Bala. Project EUROGRID (abstract). PIONIER 2001 Conference Proceedings, pp. 187--191, Poznan 2001, ISBN 83-913639-2-9

[13] John P. Stenbit. Moving Power to the Edge. CHIPS - The Department of the Navy Information Technology Magazine, Volume 21, Issue 3, Summer 2003

[14] Catlett C. The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility. Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002 , 21-24 May 2002. Pages:5 – 5

[15] http://www.d-grid.de/

[16] Depei Q. CNGrid: a test-bed for Grid technologies in China. Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of , 26-28 May 2004. Pages:135 – 139

[17] http://www.globus.org/wsrf/

[18] A.G. Ganek and T.A. Corbi. The dawning of the autonomic computing era. IBM Systems Journal, Vol 42, No 1, 2003, pp.5-18

[19] 朱福喜，何炎祥. 并行分布计算中的调度算法理论与设计，武汉：武汉大学出版社，2003

[20] S. S. Fatima, M. Wooldridge. Adaptive Task and Resource Allocation in Multi-Agent Systems. In Agents 2001: Proceedings of the Fifth International conference on Autonomous Agents Montreal, 2001

[21] 林成江，李三立. 一种可适应的分市式动态负载平衡策略及其仿真. 计算机学报，1995. 10，18(10)：721—727

[22] Braun T D, Siegel H J, Beck N, et al. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. IEEE Workshop on Advances in Parallel and Distributed Systems, West Lafayette, IN, Oct. 1998, pp. 330-335

[23] Maheswaran M, Ali S, Siegel H J, et al. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. In the 8th IEEE Heterogeneous Computing Workshop (HCW '99), San Juan, Puerto Rico, Apr. 1999, pp.30-44

[24] K. Al-Saqabi, S. Sarwar and K. Saleh. Distributed gang scheduling in networks of heterogeneous workstations. Computer Communications Journal, 1997, pp.338-348

[25] Freund R F, Gherrity M, Ambrosius S, et al. Scheduling resources in muti-user, heterogeneous, computing environments with SmartNet. In Proc. The 7th IEEE Heterogeneous Computing Workshop (HCW '98), Orlando, Florida, USA, Mar. 1998,pp.184-199

[26] http://ai.ia.ac.cn/chinese/publication/dong/dga.htm

[27] Eric B. Baum, Warren D. Smith. Propagating distributions up directed acyclic graphs. Neural Computation, 1999, 11(1): 215-227

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

[29] Rich Wolski, Neil Spring, and Jim Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. Journal of Future Generation Computing Systems, October, 1999, 15(5-6): 757-768

[30] X.H. Sun and M. Wu. "A Performance Prediction and Task Scheduling System for Grid Computing", Proc. of 2003 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003), Nice, France, April, 2003

[31] X.H. Sun and M. Wu. "Grid Harvest Service: A System for Long-Term, Application-Level Task Scheduling," in Proc. of 2003 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003), Nice, France, April, 2003

[32] Kavitha S. Golconda, Fusun Ozguner, Atakan Dogan. "A Comparison of Static QoS-Based Scheduling Heuristics for a Meta-Task with Multiple QoS Dimensions in Heterogeneous Computing," ipdps, p. 106a, 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 1,2004

[33] 苑希民，李鸿雁，刘树坤等. 神经网络和遗传算法在水科学领域的应用，北京:中国水利水电出版社，2002

[34] 王小平, 曹立明. 遗传算法——理论、应用与软件实现，西安:西安交通大学出版社, 2002

[35] S.Ali,S.M.Sait, and M.S.T.Benton, GSA: Scheduling and Allocation Using Genetic Algorithm, Proc. EURO-DAC '94, pp. 84-89, 1994.

[36] Sekhar Darbha , Dharma P. Agrawal, A task duplication based scalable scheduling algorithm for distributed memory systems, Journal of Parallel and Distributed Computing, v.46 n.1, p.15-27, Oct.10, 1997

[37] E.S.H.Hou,N.Ansari,H.Ren, A Genetic Algorithm for Multiprocessor Scheduling, IEEE Transactions on Parallel and Distributed Systems, v.5 n.2, p.113-120, February 1994

[38] Imtiaz Ahmad, Muhammad K. Dhodhi, Multiprocessor Scheduling in A Genetic Paradigm, Parallel Computing, v.22 n.3, p.395-406, March 1996

[39] Yu-Kwong Kwok , Ishfaq Ahmad, Efficient scheduling of arbitrary task graphs to

multiprocessors using a parallel genetic algorithm, Journal of Parallel and Distributed Computing, v.47 n.1, p.58-77, Nov. 25, 1997

[40] 王翠平，分布式系统中的任务调度问题及遗传算法应用研究：[硕士学位论文]，青岛：青岛大学，2002

发表论文情况

发表论文：

薛桂香，赵政，马懋德，网格任务调度策略研究，微处理机，已录用，拟于2008年第1期发表。

致谢

本论文的工作是在导师赵政教授的悉心指导下完成的，他渊博的知识、敏锐的洞察力、严谨的治学态度以及民主的作风给我留下了深刻的印象，为我开阔了视野，丰富了学识，并将使我受益终身。值此论文完成之际，对赵老师曾经给予的学术上的指导、生活上的关心和帮助致以最诚挚谢意！

同时，也非常感谢我的指导老师马懋德老师，从论文的选题、方案的制定、实验的调试以及论文的撰写，每一步都倾注着马老师的心血，两年多来对我悉心指导，严格要求，不仅在知识上，更在研究方法和为人处事上受益良多。

在论文的写作和实验过程中，我得到了师姐王红梅、师兄孙志伟、张强、张杰、赵智超和李志圣的悉心指导、关心和帮助。特别是师兄赵智超和李志圣，在整个实验的设计、调试过程中，都给予了耐心细致的指导，师兄张杰更是解决了我在实验过程中电脑所遇到的各种软硬件故障！能和这样的人在一起学习我感到非常的幸运！

非常感谢我的爱人宋建材，他的爱和支持给了我无穷的动力！多年来，在生活、学习上都给予了许多的关心和鼓励，在整个论文的工作过程中，他更是对我关心备至，对实验的实现等提供各种建议，遇到困难总是陪我一同完成！能找到这样的爱人我感到非常的幸福！

多年来的求学过程，更离不开家人的大力支持及他们细致入微的关怀。父母每日辛勤劳作，省吃俭用，再苦再累他们都自己扛着，一心一意支持孩子们上学。父亲，母亲，你们辛苦了！我的每一分成绩，都是你们的功劳！我为你们感到骄傲！