ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.11, November 2006, pp.2277−2288
DOI: 10.1360/jos172277
© 2006 by *Journal of Software*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于模糊聚类的网格 DAG 任务图调度算法*

杜晓丽[1,2+]，蒋昌俊[1,2]，徐国荣[1,2]，丁志军[1,2]

[1](同济大学 电子与信息工程学院,上海 201804)

[2](国家高性能计算机工程技术研究中心 同济分中心,上海 201804)

A Grid DAG Scheduling Algorithm Based on Fuzzy Clustering

DU Xiao-Li[1,2+], JIANG Chang-Jun[1,2], XU Guo-Rong[1,2], DING Zhi-Jun[1,2]

[1](Electronics and Information Engineering School, Tongji University, Shanghai 201804, China)

[2](Tonji Branch, National Engineering and Technology Center of High Performance Computer, Shanghai 201804, China)

+ Corresponding author: Phn: +86-21-69589864, E-mail: du_xiaoli@163.com, http://www.tongji.edu.cn

Du XL, Jiang CJ, Xu GR, Ding ZJ. A grid DAG scheduling algorithm based on fuzzy clustering. *Journal of Software*, 2006,17(11):2277−2288. http://www.jos.org.cn/1000-9825/17/2277.htm

Abstract: Focusing on the problem of task scheduling under large-scale, heterogeneous and dynamic environments in grid computing, a heuristic algorithm based on fuzzy clustering is presented. Many previous scheduling algorithms need to search and compare every processing cell in the target system in order to choose a suitable one for a task. Though those methods can get an approving Make-span, undoubtedly, it would increase the entire runtime. A group of features, which describe the synthetic performance of processing cells in the target system, are defined in this paper. With these defined features, the target system, also called processing cell network, is pretreated by fuzzy clustering method in order to realize the reasonable clustering of processor network. In the scheduling stage, the cluster with better synthetic performance will be chosen first. There is no need to search every processing cell in the target system at every scheduling step. Therefore, it largely reduces the cost on choosing which processing cell to execute the current task. The design of the ready task's priority considers not only the influence that comes from the executing of nodes on critical path, but also the influence induced by heterogeneous resource, on which the task will be scheduled. In the last part, the algorithm's performance is analyzed and compared with other algorithms, and the test results show that the bigger the target system is, the better performance the algorithm shows.

Key words: grid; DAG (direct acyclic graph); task scheduling; fuzzy clustering; heterogeneous computing environment

* Supported by the National Natural Science Foundation of China under Grant No.60534060 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.2003CB316902 (国家重点基础研究发展计划(973)); the Key Technologies R&D Program of China under Grant No.2004BA908B07-10 (国家科技攻关计划); the Program of Shanghai Subject Chief Scientist of China under Grant No.04XD14016 (上海市优秀学科带头人计划); the 2006 Mountaineering Program of Shanghai, China under Grant No.06JC14065 (上海市科委 2006 年度"登山行动计划")

Received 2006-05-22; Accepted 2006-08-07

摘 要: 针对网格环境中,任务调度的目标系统具有规模庞大、分布异构和动态性等特点,提出一种基于模糊聚类的网格异构任务调度算法.以往的很多调度算法需要在调度的每一步遍历整个目标系统,虽然能够获得较小的 makespan,但是无疑增加了整个调度的 Runtime.定义了一组刻画处理单元综合性能的特征,利用模糊聚类方法对目标系统(处理单元网络)进行预处理,实现了对处理单元网络的合理划分,使得在任务调度时能够较准确地优先选择综合性能较好的处理单元聚类,从而缩小搜索空间,大量减少任务调度时选择处理单元的时间耗费.此外,就绪任务优先级的构造既隐含考虑了关键路径上节点的执行情况对整个程序执行的影响,又考虑了异构资源对任务执行的影响.实验及性能分析比较的结果表明,定义的处理器特征能够实现对处理器网络的合理划分,而且随着目标系统规模的增大,所提出的算法优越性越来越明显.

关键词: 网格;DAG(direct acyclic graph);任务调度;模糊聚类;异构计算环境
中图法分类号: TP393 文献标识码: A

由于网络技术的高速发展,地理上分布的各种资源通过高速网络互联形成一种新型的计算平台——异构计算系统(heterogeneous computing,简称 HC).异构计算的出现,使得人们可以利用分布在各地的闲散计算资源处理较为复杂的计算密集型的并行分布式应用程序.然而,如何将应用程序的任务调度到可用的资源上,是实现高性能的关键因素之一.网格作为一种异构计算环境,其任务调度决策直接影响网格应用的性能,网格自身的自治性、异构性、动态性、分布性等特性则对传统的调度算法提出了新的挑战.当前,网格任务调度的研究往往针对元任务或者批任务开展,忽视了任务间的数据关联与优先约束关系,不能反映网格任务的实际特征[1].在并行计算与处理中,针对有向无环图 DAG(direct acyclic graph)来表示的并行任务在多处理机上的调度研究由来已久,而早期的研究并没有包含任务之间的通信关系.网格的出现标志着高性能计算领域进入了一个全新的阶段,任务调度不仅需要考虑任务之间的通信代价,还要考虑调度环境的异构性带来的影响、链路竞争、网络拓扑结构的松散易变性等问题.如何把复杂应用程序的所有任务调度到多处理器系统,并追求最小的整个执行时间的问题,一直是众所周知的难题.一般来说,对该问题寻求最优解,在绝大多数的情况下是 NP 完全问题.目前,典型的任务调度模型都是建立在图的基础上的,习惯上称它们为任务图.最常用的是 DAG 图,也就是任务优先图模型.根据任务图的基本信息、处理单元本身及其拓扑结构的基本信息是否在应用程序执行前可以得到、已经调度好的任务是否能够实时迁移等因素,任务调度算法主要分为静态调度和动态调度两大类.静态调度假设任务图和处理单元相关的信息在程序执行前可以精确获取,调度好的任务节点不能迁移,也称为编译时间调度算法;反之,则称为动态调度算法,也称为实时调度算法.本文的调度算法属于静态调度算法.

由于调度问题本身是 NP 完全问题,国内外的研究者提出了很多启发式算法,如本文所引用的文献中的算法,这些算法分为表调度算法、基于复制的调度算法、基于任务聚类的调度算法、基于随机搜索技术的调度算法等等.但是,它们中大多数仅考虑同构环境下的任务调度,而且很多算法假设处理器之间是完全连接的,考虑异构环境的任务调度算法如文献[2−5];有些算法通常仅考虑独立任务而不考虑任务之间的偏序关系,或者将 DAG 图分层考虑[4,5],这样每层中的就绪任务是独立的,然而,由于其没有考虑所有就绪任务,调度结果并不是很理想;文献[3,6]中的算法采用遗传算法和模拟退火技术,需要输入一些控制参数,而且算法的调度成本很高.还有少数调度算法,虽然考虑了任务之间的时序关系,但仅假设处理器计算能力不同而忽略链路的通信差异,或者认为处理器是完全连接的.此外,如果网格目标系统(处理单元网络)规模庞大,如何从众多的处理单元中选择有利于整个应用程序执行的处理单元也是影响调度结果的重要因素之一.需要对处理单元本身的性质以及处理单元互联结构进行分析.本文针对网格环境下目标系统规模较大的异构任务调度问题进行研究,文中的异构环境(又称为异构资源)主要是指处理器的计算能力和处理器之间链路的通信能力不同.本文的贡献在于:

1) 定义刻画网格环境中异构处理单元综合性能的特征向量,并通过实验验证其合理性;
2) 利用模糊聚类理论,将目标系统中综合性能相似的处理单元尽可能地划为一类,实现对网格异构资源的模糊划分,大量减少任务调度时选择处理单元的搜索空间,从而大量减少整个调度的运行时间;
3) 基于对目标系统的模糊聚类结果,提出一种任务调度启发式算法(fuzzy clustering based scheduling

heuristic,简称 FCBSH),该算法同时考虑了资源的异构性和任务之间的偏序关系,算法的每一步中尽量减小每个任务的完成时间,从而达到减少整个程序完成时间的目的;

4) 设计实现 DAG 图生成器和目标系统生成器.DAG 图生成器与目标系统生成器根据输入参数的不同,生成不同的 DAG 图和目标系统图,使我们能够较客观地将本文提出的算法与其他算法进行分析比较.

本文第 1 节阐述任务调度的一些相关工作.第 2 节介绍任务调度的一些基本概念和符号定义.第 3 节介绍目标系统的模糊聚类过程.第 4 节介绍 FCBSH 算法的构造过程.第 5 节给出实验结果并进行性能分析比较.第 6 节总结本文的工作.

1 相关工作

根据算法基本思想的不同,传统的并行静态任务调度的算法大致可分为 4 类:表调度算法、基于任务复制的调度算法、基于任务聚类的调度算法和基于随机搜索的调度算法.根据对目标系统的假设不同,静态任务调度又可分以为同构环境下的任务调度和异构环境下的任务调度两种.表调度的基本思想是:通过对节点分配优先级别来构造一个调度列表,然后重复从调度列表中顺序取出第一个节点,将节点分配到使它的启动时间最早的处理器上,直到任务图中所有节点被调度完毕.典型的算法有 HLEFT(highest level first with estimated times)[7],DLS(dynamic level scheduling)[2],MCP(modified critical-path)[8]等.大多数表调度算法假设目标系统是处理单元数目有限且完全连接的同构环境.通常,表调度算法比较实用,与其他调度算法相比,其时间复杂度相对较低,调度结果较好.基于任务复制调度的基本思想主要是:在一些处理器上冗余地映射任务图中的一些任务,以达到减少处理器之间通信开销的目的,即它利用处理器的空闲时间复制前驱任务,可以避免某些前驱任务的通信数据的传输,从而减少处理器等待的时间间隙.典型算法有 DSH(duplication scheduling heuristic)[8],BTDH (bottom-up top-down duplication heuristic)[9],CPFD(critical path fast duplication)[10]等.其目标系统一般是处理单元数目不受限制的同构环境.基于任务聚类调度的基本思想是把给定任务图的所有任务映射到数量不受限的集群上.如果两个任务分配到同一个聚类,则表示它们在同一个处理器上执行.除了执行聚类的步骤外,算法还必须对完成映射的聚类进行最后的调度,即对聚类中的任务在每个处理器上进行时间先后排序.典型算法有 DSC(dominant sequence clustering)[11]和 DCP(dynamic critical-path)[6]算法.基于随机搜索技术的调度算法主要是通过有导向的随机选择来搜索问题的解空间,而并不是单纯的随机搜索.这类技术组合前面搜索结果的知识和特定的随机搜索特点来产生新的结果.其主要代表是遗传算法和模拟退火方法,典型算法如文献[3].遗传算法比一般的启发式算法的调度结果要好,然而,它们往往有较高的时间复杂度,而且需要适当地确定一些控制参数.此外,适用于一个任务图的最优控制参数往往不适合另一任务图,很难找到对于大多数任务图都能产生较好的调度结果的控制参数集.当前,异构环境下(网格任务调度)任务调度的研究往往是针对元任务或者批任务开展,忽视了任务之间的数据关联和优先约束关系,即大多数针对独立任务的调度不能反映异构环境下任务(网格任务)的实际特征,其中典型算法有 Max-Min 和 Min-Min 算法等.此外,也有一些考虑任务之间数据约束关系的异构调度算法,如 DLS[2],LMT(levelized-min time)[4]等.

早期的大多数并行任务调度算法往往基于较为简单的假设,例如假设任务执行时间相同、任务之间无通信、处理器完全连接以及处理器数目不受限制等,而且,它们通常缺乏对异构资源特征的分析.随着网格计算的发展,大量闲散的计算资源可能被利用,如何从大量的计算资源中选取合适的计算资源,也是决定任务调度结果的重要因素之一.因此,本文利用模糊聚类的理论分析异构资源特征,并根据异构资源特征分析的结果提出一种启发式调度算法.

2 问题的定义

2.1 任务图

应用程序通常由有向无环图(DAG):$G=(V_t,E_t)$来表示,如图 1 所示[12].其中,V_t 是任务节点的集合,$N_t=|V_t|$是图

中节点的个数,E_t 是任务节点间边的集合,$C(t_i,t_j)$表示边$(t_i,t_j)\in E_t$ 的通信量.若 t_i 和 t_j 被分配给同一处理单元,则 $C(t_i,t_j)$为 0.$W(t_i)$为 $t_i\in V_t$ 所需的计算量.没有父节点的节点称为入口节点,没有孩子节点的节点称为出口节点.如果任务图中有多个入口(出口)节点,则可以将它们用权重为 0 的边连到一个伪入口(出口)节点上.在没有收到所有父节点发来的消息之前,节点不能开始执行.假设并行程序的每个节点(任务)可由目标系统中的任何处理单元执行.

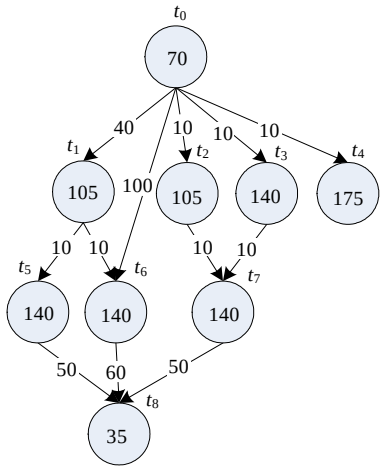

Fig.1 Task graph

图 1 任务图

$PRED(t_i)$为 t_i 直接前驱节点集合,$SUCC(t_i)$为 t_i 直接后继节点集合.SN(scheduled nodes)表示已调度任务集合.USN(unscheduled nodes)表示未调度任务的集合.RN(ready nodes)表示就绪节点集合.If $t_i\in RN$,then $t_i\in USN$ and $PRED(t_i)\in SN$.BL 即 Bottom-Level,为从节点到一个出口节点的最长路径.在同构环境下,其值是该最长路径上的节点的计算成本和各个边的通信成本之和.一般情况下,以 BL 降序进行调度的算法蕴涵着优先调度关键路径上的节点,这是因为关键路径上的节点在一定程度上决定了整个程序的执行速度.在异构环境下,由于各个处理器计算能力和处理器之间链路的通信能力不同,所以,我们取处理单元计算能力中值 M_p 和链路传输能力的中值 M_c 来衡量各个节点优先程度,用 BL^*表示.

$$BL^*(t_i)=W(t_i)/M_p+\underset{t_j\in succ(t_i)}{\mathrm{Max}}(C(t_i,t_j)/M_c+BL^*(t_j)) \tag{1}$$

$ESP_{p_j}(t_i)$ 表示任务 t_i 在处理器 p_j 上的最早开始时间(earliest start time),入口节点的最早开始时间为 0,即 $ESP_{p_j}(t_i)=0$. $ST_{p_j}(t_i)$ 为任务 t_i 在处理器 p_j 上的实际开始时间(start time),则任务 t_i 的完成时间(finished time)为

$$FT_{p_j}(t_i)=ST_{p_j}(t_i)+W(t_i)/W(p_j) \tag{2}$$

t_i 为 t_j 的直接前驱节点,即 $t_i\in PRED(t_j)$;p_x 为执行 t_i 的处理单元;p_y 为执行 t_j 的处理单元.若 $i=j$,则 $C(t_i,t_j)=0$.t_i 的数据到达 t_j 的时间为

$$AT_{p_y}(t_i,t_j)=FT_{p_x}(t_i)+C(t_i,t_j)/C(p_x,p_y) \tag{3}$$

$$EST_{p_y}(t_j)=\mathrm{Max}\left\{\underset{t_i\in PRED(t_j)\cap SN}{\mathrm{Max}}\{AT_{p_y}(t_i,t_j)\},FT_{p_y}(t_q)\right\} \tag{4}$$

这里,t_q 为处理器 p_y 上最后执行的任务.

2.2 目标系统

任务调度的目标系统是具有一定拓扑结构的处理器单元(PEs)网络,其中,每个 PE 由一个处理器(processor)和本地存储单元(local memory unit)构成.因此,PEs 不共享 memory,通信仅靠消息传递.目标系统用无向图 $P=(V_P,E_P)$来表示,其中:结点表示处理器;V_P 是处理器的集合,$N_p=|V_p|$是处理器的个数;E_P 是边(链路)的集合;$C(p_i,p_j)$表示链路$(p_i,p_j)\in E_P$ 单位时间内传输的消息量;$W(p_i)$为节点 $p_i\in V_p$ 单位时间内处理的计算量,如图 2 所示.对于目标系统我们有以下一些约定:(1) 每个处理器单元 PE 可以同时进行计算和通信.也就是说,每个 PE 的网络接口的运行不受其本身负载的影响;(2) 每个链路在每个时刻只能传输一条消息.只有当前传输的消息传输完毕时,此链路才能用于传输下一个消息;(3) 如果一个消息经过几条链路传输,则其通信时间为每个链路上的通信时间之和;(4) 网格目标系统的拓扑结构在一定时期内较为稳定.

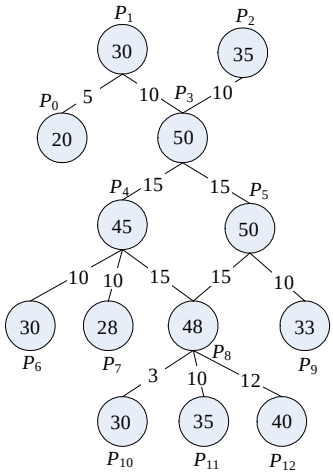

Fig.2 Target system

图 2 目标系统

3 问题的定义

本文的异构环境主要是指处理单元处理能力和不同处理单元之间链路的传输速度不同两个方面.我们不难看出,在异构环境下,任务所分配的处理单元的处理能力、通信能力、在目标系统中的位置等因素,无疑影响着其后继任务以及整个应用程序的完成.因此,在选择处理单元时,需要对处理单元的综合性能加以区分.然而,由于大多数处理单元之间没有严格的属性区别,因此更加适合于软划分,即模糊聚类.

我们定义了以下 5 个特征来刻画目标系统中的处理单元:(a) 处理能力:处理单位计算量所需的时间,即目标系统中处理单元节点的权重.其值越小,表明该处理单元的处理速度越快.(b) 平均通信能力:与该处理单元相连的链路通信能力的均值.(c) 最大传输能力:与该处理单元相连的传输速度最快的链路.(d) 网络位置:从该处理单元出发到处理单元网络中最远节点的跳数与此类节点数目的乘积.其值越小,说明该处理单元越处于网络的中心;反之,则处于网络的边缘.(e) 链路数:与该处理单元相连的链路个数.

因此,对于处理器集合 $P=\{p_1,p_2,\ldots,p_{Np}\}$中的每个处理器 p_k 都有一个模式矢量 $P(p_k)=(p_{k0},p_{k1},\ldots,p_{k4})$,其中:$p_{kj}$ (j=0,2,…,s−1)是第 k 个处理器的第 j 个特征上的值,这里,s=5,即 5 个特征.表 1 左半部分即图 2 中处理器的原始数据表 P'.由于所获得的数据往往不是[0,1]区间的数,因此,我们利用极差标准化对各个原始数据进行标准化处理得到 P'',如表 1 右半部分所示.

Table 1 The original data of the processors in target system, donated by P' (left) and results after the range standardization of P', represented by P'' (right)

表 1 目标系统中处理器的原始数据 P'(左)与极差标准化处理后的数据 P''(右)

	P'					P''				
	t_0	t_1	t_2	t_3	t_4	t_0	t_1	t_2	t_3	t_4
p_0	20.00	0.05	0.05	15.00	1.00	0.00	0.00	0.00	1.00	0.00
p_1	30.00	0.08	0.05	12.00	2.00	0.33	0.30	0.00	0.73	0.25
p_2	35.00	0.10	0.10	12.00	1.00	0.50	0.60	0.71	0.73	0.00
p_3	50.00	0.13	0.10	9.00	4.00	1.00	0.90	0.71	0.45	0.75
p_4	45.00	0.13	0.10	6.00	4.00	0.83	0.90	0.71	0.18	0.75
p_5	50.00	0.13	0.10	9.00	3.00	1.00	1.00	0.71	0.45	0.50
p_6	30.00	0.10	0.10	8.00	1.00	0.33	0.60	0.71	0.36	0.00
p_7	28.00	0.10	0.10	8.00	1.00	0.27	0.60	0.71	0.36	0.00
p_8	48.00	0.12	0.08	4.00	5.00	0.93	0.84	0.43	0.00	1.00
p_9	33.00	0.10	0.10	4.00	1.00	0.43	0.60	0.71	0.00	0.00
p_{10}	30.00	0.08	0.08	5.00	1.00	0.33	0.36	0.43	0.09	0.00
p_{11}	35.00	0.10	0.10	5.00	1.00	0.50	0.60	0.71	0.09	0.00
p_{12}	40.00	0.12	0.12	5.00	1.00	0.67	0.84	1.00	0.09	0.00

首先,利用目标系统中处理单元的每一维特征的均值和标准差对原始数据 P 进行数据标准化处理,则各数据标准化值 p'_{ik} 为

$$p'_{ik}=(p_{ik}-\bar{t}_k)/S_{t_k} \tag{5}$$

其中, $\bar{t}_k$ 是第 k 个特征向量 t_k 的原始数据的均值, S_{t_k} 为 t_k 的原始数据标准差.

这样得到的标准化数据 p'_{ik} 还不一定在[0,1]闭区间内,我们采用极值标准化方法:

$$p''_{ik}=(p'_{ik}-p'_{k\min})/(p'_{k\max}-p'_{k\min}) \tag{6}$$

其中, $p'_{k\min}$ 为 $p'_{1k},p'_{2k},...,p'_{Npk}$ 的最小值; $p'_{k\max}$ 为 $p'_{1k},p'_{2k},...,p'_{Npk}$ 的最大值.

然后,利用指数相似系数法得到处理器集 $P=\{p_1,p_2,\ldots,p_{Np}\}$的模糊相似关系 $\tilde{R}_s$:

$$\tilde{R}_s=\begin{bmatrix}
1.00 & 0.43 & 0.36 & 0.02 & 0.01 & 0.06 & 0.32 & 0.35 & 0.00 & 0.27 & 0.42 & 0.25 & 0.21\\
0.43 & 1.00 & 0.77 & 0.19 & 0.12 & 0.28 & 0.68 & 0.68 & 0.07 & 0.60 & 0.68 & 0.58 & 0.38\\
0.36 & 0.77 & 1.00 & 0.24 & 0.21 & 0.24 & 0.83 & 0.80 & 0.24 & 0.80 & 0.62 & 0.81 & 0.62\\
0.02 & 0.19 & 0.24 & 1.00 & 0.87 & 0.92 & 0.30 & 0.30 & 0.71 & 0.16 & 0.09 & 0.20 & 0.44\\
0.01 & 0.12 & 0.21 & 0.87 & 1.00 & 0.79 & 0.29 & 0.28 & 0.82 & 0.31 & 0.23 & 0.38 & 0.64\\
0.06 & 0.28 & 0.24 & 0.92 & 0.79 & 1.00 & 0.30 & 0.29 & 0.56 & 0.16 & 0.12 & 0.20 & 0.44\\
0.32 & 0.68 & 0.83 & 0.30 & 0.29 & 0.30 & 1.00 & 0.99 & 0.28 & 0.85 & 0.76 & 0.87 & 0.64\\
0.35 & 0.68 & 0.80 & 0.30 & 0.28 & 0.29 & 0.99 & 1.00 & 0.27 & 0.83 & 0.75 & 0.84 & 0.61\\
0.00 & 0.07 & 0.24 & 0.71 & 0.82 & 0.56 & 0.28 & 0.27 & 1.00 & 0.42 & 0.25 & 0.43 & 0.56\\
0.27 & 0.60 & 0.80 & 0.16 & 0.31 & 0.16 & 0.85 & 0.83 & 0.42 & 1.00 & 0.82 & 0.98 & 0.77\\
0.42 & 0.68 & 0.62 & 0.09 & 0.23 & 0.12 & 0.76 & 0.75 & 0.25 & 0.82 & 1.00 & 0.81 & 0.55\\
0.25 & 0.58 & 0.81 & 0.20 & 0.38 & 0.20 & 0.87 & 0.84 & 0.43 & 0.98 & 0.81 & 1.00 & 0.81\\
0.21 & 0.38 & 0.62 & 0.44 & 0.64 & 0.44 & 0.64 & 0.61 & 0.56 & 0.77 & 0.55 & 0.81 & 1.00
\end{bmatrix}.$$

$$r_{ij}=\frac{1}{S}\sum \mathrm{e}^{-\frac{3}{4}*\frac{(p''_{ik}-p''_{jk})}{S''^2_{t_k}}} \tag{7}$$

其中, S''^2_{tk} 是第 k 个特征的方差, r_{ij} 为处理器 p_i 和 p_j 的相似度.

最后,对相似系数矩阵作合成运算后得到具有传递闭包性的模糊等价关系 $\tilde{R}_e$:

$$\tilde{R}_e=\begin{bmatrix}
1.00 & 0.43 & 0.43 & 0.43 & 0.43 & 0.43 & 0.43 & 0.43 & 0.43 & 0.43 & 0.43 & 0.43 & 0.43\\
0.43 & 1.00 & 0.77 & 0.64 & 0.64 & 0.64 & 0.77 & 0.77 & 0.64 & 0.77 & 0.77 & 0.77 & 0.77\\
0.43 & 0.77 & 1.00 & 0.64 & 0.64 & 0.64 & 0.83 & 0.83 & 0.64 & 0.83 & 0.82 & 0.83 & 0.81\\
0.43 & 0.64 & 0.64 & 1.00 & 0.87 & 0.92 & 0.64 & 0.64 & 0.82 & 0.64 & 0.64 & 0.64 & 0.64\\
0.43 & 0.64 & 0.64 & 0.87 & 1.00 & 0.87 & 0.64 & 0.64 & 0.82 & 0.64 & 0.64 & 0.64 & 0.64\\
0.43 & 0.64 & 0.64 & 0.92 & 0.87 & 1.00 & 0.64 & 0.64 & 0.82 & 0.64 & 0.64 & 0.64 & 0.64\\
0.43 & 0.77 & 0.83 & 0.64 & 0.64 & 0.64 & 1.00 & 0.99 & 0.64 & 0.87 & 0.82 & 0.87 & 0.81\\
0.43 & 0.77 & 0.83 & 0.64 & 0.64 & 0.64 & 0.99 & 1.00 & 0.64 & 0.87 & 0.82 & 0.87 & 0.81\\
0.43 & 0.64 & 0.64 & 0.82 & 0.82 & 0.82 & 0.64 & 0.64 & 1.00 & 0.64 & 0.64 & 0.64 & 0.64\\
0.43 & 0.77 & 0.83 & 0.64 & 0.64 & 0.64 & 0.87 & 0.87 & 0.64 & 1.00 & 0.82 & 0.98 & 0.81\\
0.43 & 0.77 & 0.82 & 0.64 & 0.64 & 0.64 & 0.82 & 0.82 & 0.64 & 0.82 & 1.00 & 0.82 & 0.81\\
0.43 & 0.77 & 0.83 & 0.64 & 0.64 & 0.64 & 0.87 & 0.87 & 0.64 & 0.98 & 0.82 & 1.00 & 0.81\\
0.43 & 0.77 & 0.81 & 0.64 & 0.64 & 0.64 & 0.81 & 0.81 & 0.64 & 0.81 & 0.81 & 0.81 & 1.00
\end{bmatrix}.$$

通过设定不同的截集水平 α 值,得到不同的聚类结果. α 值越接近 1,表明聚类之间的相似程度越高; α 值越接近 0,则表明聚类之间的相似性越差.我们取 α=0.8,得到截集 $(\tilde{R}_e)_{0.8}$:

$$(\tilde{R}_e)_{0.8}=\begin{bmatrix}1&0&0&0&0&0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&0&0&0&0&0\\0&0&1&0&0&0&1&1&0&1&1&1&1\\0&0&0&1&1&1&0&0&1&0&0&0&0\\0&0&0&1&1&1&0&0&1&0&0&0&0\\0&0&0&1&1&1&0&0&1&0&0&0&0\\0&0&1&0&0&0&1&1&0&1&1&1&11\\0&0&1&0&0&0&1&1&0&1&1&1&1\\0&0&0&1&1&1&0&0&1&0&0&0&0\\0&0&1&0&0&0&1&1&0&1&1&1&1\\0&0&1&0&0&0&1&1&0&1&1&1&1\\0&0&1&0&0&0&1&1&0&1&1&1&1\\0&0&1&0&0&0&1&1&0&1&1&1&1\end{bmatrix}.$$

得到的聚类结果为{{p_0},{p_1},{p_2,p_6,p_7,p_9,p_{10},p_{11},p_{12}},{p_3,p_4,p_5,p_8}}.对聚类结果中的每个类称为一个 Cluster,记作 CL.可以看出,上例中共有 4 个 Cluster.这样,每个类(cluster)的综合性能可由式(8)估算:

$$PERF(CL_j)=\frac{1}{n}\sum_{p_k\in CL_j}\sum_{i=0}^{s-1}\alpha_i\cdot P''[k][i] \tag{8}$$

其中:n 为类 CL_j 中包含的处理器单元的个数;α_i 为处理器第 i 个特征的权重,其值一般根据任务对通信能力和计算能力的要求,依靠经验或实验获得.上述聚类结果按其综合性能从高到低排序为{{p_3,p_4,p_5,p_8},{p_2,p_6,p_7,p_9,p_{10},p_{11},p_{12}},{p_1},{p_0}}.

4 FCBSH 算法

显然,如果我们能够尽量缩短每个任务的完成时间,就能缩短整个程序的并行完成时间(make span).所以,本算法的核心思想是尽可能地缩短每个任务的完成时间.而任务的完成时间取决于两个因素,其一是任务的开始时间,其二是任务的执行时间.而任务的开始时间取决于其前驱任务的执行情况以及前驱任务所在处理单元的网络位置、当前任务所在处理单元本身的任务执行情况以及当前任务所在处理单元的网络位置.所以,将一个任务分配到目标系统中的哪个处理单元来执行,将在一定程度上影响其后继任务的执行情况.而第 3 节的聚类结果可以作为具体任务调度时如何选择处理单元的参考和指导.任务的优先级定义如下:

$$Priority(t_i)=BL^*+\Delta(Considered,t_i) \tag{9}$$

$$\Delta(Considered,t_i)=\max FTCon(t_i)-\min FTCon(t_i) \tag{10}$$

如前所述,在网格异构环境下,由于各个处理器计算能力和处理器之间链路通信能力的不同,导致无法像同构环境那样计算每个任务的 BL,因此,本文利用处理器计算能力的中值和链路传输能力的中值来衡量,记作 BL^*.图 1 中各个任务的 BL^*见表 2.

Table 2 The task BL^* of Fig.1

表 **2** 图 1 中任务的 BL^*值

Node	$^*n_0$	n_1	n_2	n_3	n_4	n_5	$^*n_6$	n_7	$^*n_8$
BL^*	23.86	15	14	15	2.14	10	11	10	1

我们把目标系统中的所有处理单元分为 *Considered* 集合和 *UnConsidered* 集合:*Considered* 集合是当前为任务分配处理单元对象的可选集合;*UnConsidered* 集合是当前暂时未考虑分配任务的处理单元集合.$\max FTinCon(t_i)$表示在 *Considered* 集合所有的处理单元中,处理任务 t_i 的最大完成时间;$\min FTinCon(t_i)$表示最小完成时间.Δ越大,表示将任务 t_i 分配到不同处理单元时的完成时间差越大,也就是说,不同处理单元的分配对其完成情况的影响越大.因此,应该给予Δ值较大的任务以较高的优先权,使其能够优先调度到完成时间最小的处理单元上,以减少由于未能及时将其调度到完成时间较早的处理单元上而导致其完成时间大幅增加;而对于Δ

值较小的任务来说,处理单元的完成时间相差不多,所以将其调度到哪个处理单元对于完成时间的影响相对小一些.

FCBSH 算法如下:

1. 对处理器网络进行模糊聚类,选取合适的截集水平,得到处理器聚类;
2. 将综合性能最好的处理器聚类放入 *Considered* 集合,其他聚类属于 *UnConsidered* 集合;
   //没有任务在其上执行的处理单元,记作 *NoLoadProcessor*
3. Let *makeSpan*=0;
4. Repeat
5. if (∃*NoLoad Processor*∈*Considered*)
6. Let max*Pri*=0, *task*=0;
7. For each $t_i \in RN$
8. 计算 *Priority*(t_i);
9. if (*Priority*(t_i)>max*Pri*)
10. max*Pri*=*Priority*(t_i);
11. *task*=t_i;
12. end if.
13. end for.
14. 将 *task* 调度到使其完成时间最小的处理单元 *p* 上;
    // 若有两个处理单元的完成时间相同,则选择网络位置较中心的处理单元
15. *makeSpan*=FT_p(*task*);
16. else
17. do Step 7~Step 13,找到具有最大优先权的任务 *task*;
18. if (FT_p(*task*)<=*makeSpan*)
19. 将 *task* 调度到使其完成时间最小的处理单元 *p* 上;
20. else
21. 从 *UnConsidered* 集合中综合性能最好的一个聚类中,选择一个距离 *Considered* 集合中负载最大的处理单元最近的处理单元 *p*′;
22. if (FT_p(*task*)>$FT_{p'}$(*task*))
23. 将 *task* 调度到 *p*′上;
24. add *p*′ into *Considered*;
25. *makeSpan*=$FT_{p'}$(*task*);
26. else
27. 将 *task* 调度到 *p*′上;
28. *makeSpan*=$FT_{p'}$(*task*);
29. end if.
30. end if.
31. end if.
32. remove *task* from *RN*;
33. update the *RN*;
34. Until each $t_i \in V_t$ has been assigned.

算法的第 1 步利用第 3 节提出的模糊聚类方法对目标系统中的处理单元进行模糊聚类,这项工作实际上是进行任务调度前的准备工作,可以看作预处理工作,当有大量同类应用程序在目标系统上计算时仅需执行一次.

经过模糊聚类后的目标系统聚类,极大地缩小了任务调度在选择处理单元时的所搜空间和时间复杂度,任务调度不必每次都检测所有的处理单元,算法在选择处理单元时优先选择综合性能较好的处理单元分配给当前任务.在算法的第 5~第 15 步,如果 *Considered* 集合中尚有空载的处理单元,则计算每个就绪节点的权值,将权值最大的节点调度到使其完成时间最小的处理单元上;在算法的第 16~第 31 步,对于 *Considered* 集合中无空载的处理单元且当前权值最大的节点调度导致并行完成时间增加的情况,则从 *UnConsidered* 中综合性能最好的一个聚类中选择一个距离 *Considered* 集合中负载最大的处理单元最近的处理单元.计算当前节点在此处理单元上的完成时间,如果其完成时间小于 *Considered* 集合中的所获得的最小完成时间,则将当前节点调度到该处理单元上,并将此结点加入 *Considered* 集合.否则,放弃选择该处理单元,将当前节点分配到 *Considered* 集合中具有最小完成时间的处理单元上.若有两个处理单元的完成时间相同,则选择网络位置较接近中心的处理单元;算法的第 32 步将当前调度的节点从就绪节点集合中删除,更新就绪节点集合.FCBSH 算法的时间复杂度在最坏情况下为 $O(N_t^2 \times N_p)$.

5 实验与性能分析

由于不同的任务调度算法,其问题假设也不尽相同.在某些情况下调度结果较好的算法,在其他情况下可能差强人意.由于 DLS 算法在异构环境问题方面与本文的问题假设基本相同,因此,我们将 FCBSH 算法与 DLS 算法进行性能分析比较.下面给出实验环境和用于性能分析的一些比较指标.

5.1 实验环境

5.1.1 随机生成任务图

我们设计了一个简单的随机 DAG 图生成器,根据用户定制的参数,产生相应的任务图.需要输入的参数如下:任务图中的节点个数 V_t.节点的最大权重 max*Node_weight_task* 和最小权重 min*Node_weight_task*,任务图中节点的计算量为 [min*Node_weight_task*,max*Node_weight_task*] 之间的一个随机数.边的最大权重 max*Edge_weight_task* 和最小权重 min*Edge_weight_task*,任务图中节点与节点之间的通信量为 [min*Edge_weight_task*,max*Edge_weight_task*]之间的一个随机数.节点的最大出度 max_*out_degree*,对于每个节点而言,其出度为 [0,max_*out_degree*] 之间的一个随机数.图的深度 *deepth*,对于一个 $V_t=n$, max_*out_degree*=$n-1$,*depth*=1 的任务图就是一个 fork 图.

5.1.2 随机生成目标系统

我们同样设计了一个简单的随机目标系统图生成器,根据用户定制的参数,产生相应的网格目标系统.需要输入的参数如下:目标系统类型 *type_tarsystem* 有完全连接图和任意连接图两种,分别用 1 和 0 表示.目标系统节点个数 V_p.节点的最大出度 max_*out_degree*,对于每个节点而言,其出度为[0,max_*out_degree*]之间的一个随机数.节点的异构率$\lambda \in [0,1]$,即处理单元的异构率,当$\lambda=1$ 时,目标系统的处理单元为完全异构,不存在相同的两个处理单元;当$\lambda=0$ 时,目标系统为处理单元同构的系统.边的异构率$\delta \in [0,1]$,即连接处理单元的链路的异构率,当$\delta=1$ 时,链路为完全异构,每条链路的通信能力都不相同;当$\delta=0$ 时,目标系统为链路同构的系统.节点的最大权重 max*Node_weight_tarsystem* 和最小权重 min*Node_weight_tarsystem*,目标系统图中节点的计算量为 [min*Node_weight_tarsystem*,max*Node_weight_tarsystem*] 之间的一个随机数.边的最大权重 max*Edge_weight_tarsystem* 和最小权重 min*Edge_weight_tarsystem*,目标系统图中节点与节点之间的通信量为 [min*Edge_weight_tarsystem*,max*Edge_weight_tarsystem*]之间的一个随机数.

5.2 性能分析指标

5.2.1 算法运行时间(runtime)

运行时间是算法获得给定任务图的调度结果所花费的执行时间,通常运行时间越小,算法越实用.

5.2.2 调度结果优于其他算法的次数

进行若干次实验以后,统计每个算法产生较好、较差、相同调度结果的次数.

5.3 性能分析

我们以表 3 中的数据为基础进行实验.首先,通过一组实验来比较 DLS 算法和本算法的运行时间.由于在实际应用中,一个目标系统接收大量同类应用程序时,本算法中对目标系统的模糊聚类预处理只需进行一次,因此所花费的时间可以忽略.在每次实验中,我们随机生成只有 5 个节点的 DAG 图({V_t=5,max*Node_weight_task*=50, min*Node_weight_task*=10,max*Edge_weight_task*=10,min*Edge_weight_task*=5,max_*out_degree*=3,*depth*=3}),将其调度到随机生成的目标系统上(V_p={5,10,20,40,50,60,70,80,90,100};max*Node_weight_tarsystem*=300; min*Node_weight_tarsystem*=100;max*Edge_weight_tarsystem*=50;min*Edge_weight_tarsystem*=10;*type_tarsystem*=1;λ,δ为表 3 中的随机值,max_*out_degree*=3 或 5),其结果如图 3 所示.可以看出:随着目标系统规模的扩大,DLS 的运行时间急剧上升;然而,FCBSH 算法的运行时间在 V_p<70 时基本为 0,而且在 70≤V_p≤100 时,其运行时间略有增加,但仍未超过 15;而 DLS 算法在目标系统规模达到 100 时,运行时间已经超过了 2 000.

Table 3 Simulation tests datasheet

表 3 模拟实验数据表

DAG		Target system	
Metrics	Value set	Metrics	Value set
V_t	{5,10,20}	V_p	{5,10,20,40,50,60,70,80,90,100,110}
max*Node_weight_task*	{100,200,300}	max*Node_weight_tarsystem*	{100,300,500}
min*Node_weight_task*	{10,50,100}	min*Node_weight_tarsystem*	{5,50,100}
max*Edge_weight_task*	{10,100,200}	max*Edge_weight_tarsystem*	{50,100,200}
min*Edge_weight_task*	{5,50,100}	min*Edge_weight_tarsystem*	{10,50,100}
max_*out_degree*	{3,5}	λ	{0.3,0.5,0.7,0.9,1}
deepth	{2,3,5}	δ	{0.3,0.5,0.7,0.9,1}
		type_tarsystem	1
		max_*out_degree*	{3,5}

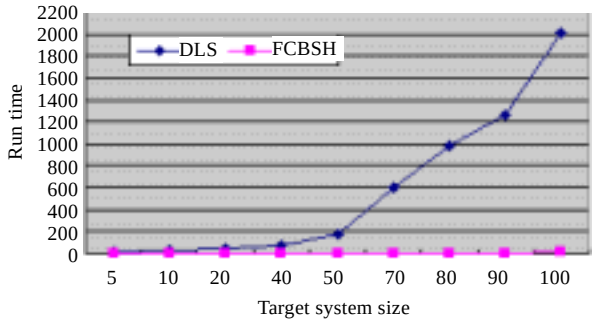

Fig.3 The runtime comparison between DLS and FCBSH

图 3 DLS 算法与 FCBSH 算法运行时间比较

其次,根据上述实验中 DAG 图和目标系统的参数进行 9 组实验,每组分别进行 50 次实验,每次实验中随机生成 DAG 图和相应节点数目的任务图,FCBSH 算法的调度结果比 DLS 算法调度结果较好(better)、略差(little worse 调度结果相差不超过 4 个时间单位)、较差(much worse 调度结果相差超过 4 个时间单位)和相同(equal)的实验结果在每组实验中所占的百分比如图 4 所示.由图 4 可以看出,FCBSH 在每组实验中产生比 DLS 较好的调度结果的情况为(10%~30%),产生略差调度结果的情况为(25%~55%),产生很差调度结果的情况为(5%~20%),产生相同结果的情况大部分为(20%~60%).图 5 为当 V_p=10 时,DLS 算法和 FCBSH 算法的 makespan 情况比较.

因此,我们可以得出如下结论:虽然 FCBSH 算法产生较好的调度结果仅为 10%~30%,但是,其产生很差调度结果的情况也仅为 5%~20%.而且 FCBSH 算法的运行时间远远小于 DLS,尤其是随着目标系统规模的扩大,FCBSH 算法的优越性越来越明显.

最后,我们将 FCBSH 算法、DLS 算法与忽略任务之间通信的调度算法(no communication,简称 NC)进行实验比较.NC 算法在调度任务时不考虑任务之间的通信,但满足任务之间的前驱与后继约束关系,为任务选择处

理单元时,尽量选择处理能力较高的处理单元.分别进行 6 组实验,每组实验中随机生成的目标系统的参数值 V_p={10,30,50,70,90,110};max*Node_weight_tarsystem*,min*Node_weight_tarsystem*,max*Edge_weight_tarsystem*,min*Edge_weight_tarsystem* 分别取表 3 中的随机值;*type_tarsystem*=1;λ,δ为表 3 中的随机值;max_*out_degree*=3 或 5;任务图大小固定为 V_t=5,其他参数的取值为表 3 中的随机值.每组实验进行 10 次,每次实验随机生成 DAG 图和目标系统.由图 6 可以看出,DLS 和 FCBSH 算法的 *makespan* 都比 NC 算法小得多,而且 FCBSH 算法和 DLS 算法的 *makespan* 相差不多.FCBSH 算法的调度结果比 DLS 算法的调度结果较好(better)、略差(little worse)、较差(much worse)和相同(equal)的实验结果在每组实验中所占的百分比如图 7 所示.由图 7 可以看出,FCBSH 在每组实验中产生比 DLS 较好的调度结果的情况大多为 3~4 次,仅有一组实验为 6 次;产生略差的调度结果的情况为 2~4 次,产生很差的调度结果的情况为 2~4 次,仅有一组实验为 5 次;产生相同结果的有两组实验分别为 1 次和 2 次,其余 4 组实验为 0 次.

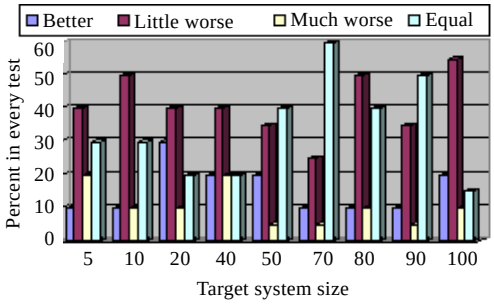

Fig.4 The percent that FCBSH produces better, little worse, much worse and equal results with DLS

图 4 FCBSH 算法产生比 DLS 好、略差、很差、相同结果的百分比

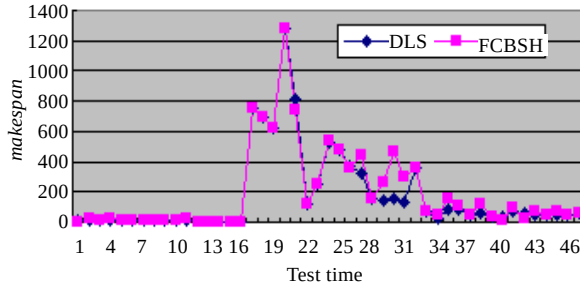

Fig.5 When V_p=10, the makespan comparison between DLS and FCBSH

图 5 当 V_p=10 时,DLS 与 FCBSH 的 makespan 比较

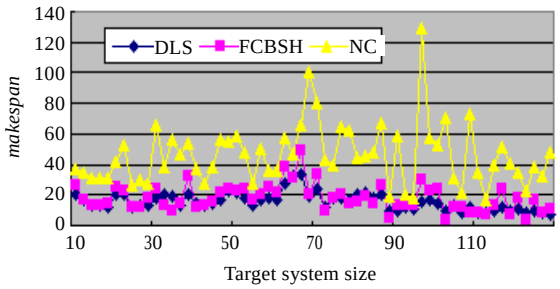

Fig.6 Comparison of makespan between FCBSH, DLS and NC algorithm

图 6 FCBSH,DLS 和 NC 算法的 makespan 比较

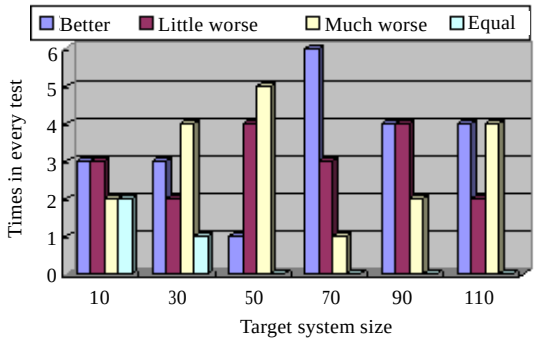

Fig.7 Comparison of scheduling results between FCBSH and DLS

图 7 FCBSH 算法与 DLS 算法的调度结果比较

6 总 结

本文针对网格异构环境下的任务调度问题进行研究,主要工作有以下两个方面:首先,定义用于刻画处理器综合性能的一组特征,基于这些特征对目标系统进行模糊聚类,从而有利于减少调度过程中选择处理器所花费的时间;其次,提出一种基于模糊聚类的启发式算法(FCBSH),并与问题假设相似的 DLS 算法进行性能分析和比较.然而,本算法也存在如下不足之处:首先,由于处理器特征的权重(α_i)取值与经验有关,需要反复测试;其次,截集水平α值与处理器网络本身的物理性质有关,其取值也需要经过测试,以便获得较理想的聚类结果;再次,本算

法适合于处理器网络拓扑结构在一定时期内相对稳定的情况,对于节点频繁加入或离开的网络,本算法的优越性也将无从体现;最后,本文仅考虑了处理单元处理能力和通信能力异构的较为简单的情况,链路可靠性、处理单元性能、通信能力动态变化等复杂情况,将是我们今后的工作重点.

致谢　在此,我们向对本文的工作给予支持和建议的同行,尤其是同济大学国家高性能计算机中心的同学和老师表示感谢.此外,本文的灵感来源于与同济大学交通运输学院杨志清博士的讨论,特此表示谢意.

References:

[1] Qiao WG, Zeng GS, Hua A, Zhang F. Scheduling and executing heterogeneous task graph in grid computing environment. In: Zhuge Hai, Fox G, eds. Proc. of the 4th Int'l Workshop on Grid and Cooperative Computing (GCC 2005). LNCS 3795, Beijing: Springer-Verlag, 2005. 474−479.

[2] Sih GC, Lee EA. A compile-time scheduling heuristic for interconnection constrained heterogeneous processor architectures. IEEE Trans. on Parallel and Distributed Systems, 1993,4(2):75−87.

[3] Hou ESH, Ansari N, Ren H. A genetic algorithm for multiprocessor scheduling. IEEE Trans. on Parallel and Distributed Systems, 1994,5(2):113−120.

[4] Iverson M, Ozguner F, Follen G. Parallelizing existing applications in a distributed heterogeneous environment. In: Proc. of the Heterogeneous Computing Workshop. Santa Barbara: IEEE Computer Society Press, 1995. 93−100.

[5] Maheswaran M, Siegel HJ. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In: Antonio JK, ed. Proc. of the Heterogeneous Computing Workshop. Orlando: IEEE Computer Society Press, 1998. 57−69.

[6] Kwok YK, Ahmad I. Dynamic critical-path scheduling: An effective technique for allocating task graphs onto multiprocessors. IEEE Trans. on Parallel and Distributed Systems, 1996,7(5):506−521.

[7] Adam TL, Chandy KM, Dickson J. A comparison of list scheduling for parallel processing systems. Communications of the ACM, 1974,17(12):685−690.

[8] Wu M, Gajski D. Hypertool: A programming aid for message passing systems. IEEE Trans. on Parallel and Distributed Systems, 1990,1(3):330−343.

[9] Chung YC, Ranka S. Application and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. In: Werner R, ed. Proc. of the Supercomputing'92. Minneapolis: IEEE Computer Society Press, 1992. 512−521.

[10] Ahmad I, Kwok YK. On exploiting task duplication in parallel programs scheduling. IEEE Trans. on Parallel and Distributed Systems, 1998,9(9):872−892.

[11] Yang T, Gerasoulis A. DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Trans. on Parallel and Distributed Systems, 1994,5(9):951−967.

[12] Kwok YK, Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys, 1999,31(4):406−471.

杜晓丽(1978－),女,河北鹿泉人,博士生,主要研究领域为网络计算,网格计算,并行算法.

徐国荣(1980－),男,硕士生,主要研究领域为 Web 数据挖掘,搜索引擎.

蒋昌俊(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网络计算,语义网格,Petri 网.

丁志军(1974－),男,博士生,讲师,主要研究领域为语义网格,Petri 网.