融合 PSO 与 ACS 的网格资源分配研究

梁正友,支成秀
LIANG Zheng-you,ZHI Cheng-xiu

广西大学 计算机与电子信息学院,南宁 530004
School of Computer and Electronic Information,Guangxi University,Nanning 530004,China
E-mail:lzyfile@sohu.com

LIANG Zheng-you,ZHI Cheng-xiu.Combination algorithm of POS and ACS for grid resource allocation.Computer Engineering and Applications,2009,45(9):102-104.

Abstract: One key problem of grid computing is resource allocation.A combination algorithm of discrete particle swarm optimization algorithm(PSO) and ant colony algorithm(ACS) for grid resource allocation is presented.Combining PSO and ACS,the proposed algorithm can effectively avoid the local optima of PSO algorithm and the blindness search of ant colony algorithm.Theories analytical and simulation experiment show the algorithm conducts good performance.

Key words: grid computing;task scheduling;resource allocation;particle swarm algorithm;ant colony algorithm

摘 要: 资源分配和任务调度是网格计算的一个关键问题之一。提出一种融合离散粒子群优化算法和蚁群算法的新型算法来解决网格资源分配问题。该算法通过在粒子群算法中引入蚂蚁算法,可有效克服粒子群算法后期的局部搜索能力差和蚁群算法前期盲目搜索的缺陷。理论分析及模拟实验表明该算法具有良好的性能。

关键词: 网格计算;任务调度;资源分配;粒子群算法;蚁群算法
DOI: 10.3778/j.issn.1002-8331.2009.09.029 **文章编号:** 1002-8331(2009)09-0102-03 **文献标识码:** A **中图分类号:** TP393

1 引言

在网格环境下的资源分配和任务调度是关键问题,也是一个 NP 难的问题[1]。近年来,人们分别将蚁群算法(ACS)[2-3]和粒子群算法(PSO)[4-5]应用到网格资源分配和任务调度中,取得了较好的效果。

蚁群算法和粒子群算法都是拟生态优化算法,它们各有优缺点。蚁群算法的优点[2,6-7]:(1)具有信息正反馈机制;(2)分布式计算特征;(3)通用型随机优化方法;(4)具有启发式搜索特征。存在的缺点是:初期信息素匮乏,求解速度慢。PSO 算法的优点[4,8-9]:(1)具有大范围全局搜索能力;(2)搜索从群体出发,具有隐并行性;(3)搜索使用评价函数值启发;(4)收敛速度快,参数调整简单;(5)具有扩展性,容易与其他算法结合。存在的缺点是:在算法后期的局部搜索能力差,反馈信息利用不充分。

将粒子群算法与蚁群算法融合可克服各自算法的缺点,能获取更好的优化性能;提出的融合粒子群算法与蚁群算法[10]在连续空间中具有良好的优化性能。文中设计一种融合离散粒子群算法与蚁群算法,使得该算法能解决象任务调度那样的离散空间的优化问题,并适合用于网格资源分配和任务调度。模拟实验表明该融合算法在网格资源分配和任务调度方面取得较好的效果。

2 网格资源分配的数学模型

考虑由独立子任务组成的作业在网格环境下的调度问题,这类调度问题可描述为:假设作业 $T=\{t_1,t_2,\cdots,t_n\}$要在异构的网格 $G=\{g_1,g_2,\cdots,g_m\}$上完成,其中,$t_j(j\in\{1,n\})$为独立子任务,$g_i(i\in\{1,m\})$为异构的网格节点。网格的每个资源节点的计算速度用每单位时间执行的指令数表示,任务长度用指令数表示。每个子任务只能分配给一台机器处理,不同的分配花费不同的代价。资源分配和任务调度问题是要求找出一种时间花费的代价最小的分配方案。

定义 $C_{i,j}(i\in\{1,2,\cdots,m\}),j\in\{1,2,\cdots,n\}$表示子任务 t_j 在资源 g_i 上的完成时间,C_i 表示资源 g_i 处理所有分配到该资源上执行的子任务的总完成时间。$C_{\max}=\max\{C_i\}$表示一次分配的适应值。优化的目标是使 $C_{\max}$ 最小。

3 融合离散粒子群算法与蚁群算法

3.1 离散粒子群算法

传统的粒子群算法主要解决连续优化问题。而调度问题是个离散化的问题。根据网格资源分配和任务调度问题的特殊性提出一种离散粒子群算法(DPSO),通过重新定义粒子的位置、速度以及它们的更新规则以解决网格调度问题[5]。

3.1.1 位置与速度的定义

(1)位置的定义

假设作业 T 要在网格 G 上调度执行,则粒子 i 的位置 $X_i=\{x_1,x_2,\cdots,x_j,\cdots,x_n\}$是个 N 维向量。其中,数据 x_j 表示子任务 t_j

基金项目: 广西教育厅科研项目(桂教科研[2006]26 号);广西大学博士启动基金项目。
作者简介: 梁正友(1968-),男,壮族,副教授,博士,主要研究领域为网格计算、并行分布式计算;支成秀(1973-),女,硕士,主要研究领域为网格计算。
收稿日期: 2008-01-31 **修回日期:** 2008-05-08

被分配给资源号 x_j 执行,$l\leqslant x_j\leqslant m$。$X_i$ 表示一个可行的资源分配方案。在对粒子的位置进行更新时,粒子的位置 X_i 转化为一个用元素为 0 或 1 的位置矩阵 $\boldsymbol{XX}_i$:

$$\boldsymbol{XX}_i=\begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & & \vdots \\ s_{m1} & s_{m2} & \cdots & s_{mn} \end{bmatrix}$$

如果子任务 t_j 在资源 g_i 上执行,则 $s_{ij}=1, i\in\{1,2,\cdots,m\}, j\in\{1,2,\cdots,n\}$,否则 $s_{ij}=0$。由于一个子任务只能在一个资源节点上执行,所以 $\sum_{i=1}^{m}s_{ij}=1, i\in\{1,2,\cdots,m\}, j\in\{1,2,\cdots,n\}$。

(2)速度的定义

离散化空间的速度不再是连续空间所说的速度,而是用来计算粒子的位置变化的概率的值,粒子 i 的速度用一个矩阵来表示:

$$\boldsymbol{V}_i=\begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{bmatrix}$$

$v_{ij}\in[-v_{\max},v_{\max}], i\in\{1,2,\cdots,m\}, j\in\{1,2,\cdots,n\}$,$v_{\max}$ 取 4,能使后面公式(1)表示的模糊函数的取值范围在 0.018~0.982,这种取法能使算法以较大的概率发生变化[8]。

3.1.2 位置与速度的更新规则

由于速度是一个表示粒子的位置变化的概率值,它应该被限制在[0,1]之间。因此,就必须对粒子速度的值进行调整,在对速度进行限制在[0,1]之间前,先按公式(1)把速度限定在$[-v_{\max},v_{\max}]$内,定义子函数:

$$g(V_{ij}^k)=\begin{cases} V_{\max}, & V_{ij}^k>V_{\max} \\ V_{ij}^k, & V_{\min}<V_{ij}^k<V_{\max} \\ V_{\min}, & V_{ij}^k<V_{\min} \end{cases} \tag{1}$$

引入 Sigmoid 函数(公式(2)),它具备把数限制在[0,1]的特点:

$$sig(V_{ij}^k)=\frac{1}{1+\exp(-V_{ij}^k)} \tag{2}$$

然后就可以利用下列式子来更新粒子速度和位置:

$$\Delta V_{lj}^k=c_1\cdot r_1\cdot(PB_{lj}^k-X_{lj}^k)+c_2\cdot r_2\cdot(GB_{lj}^k-X_{lj}^k) \tag{3}$$

$$V_{lj}^k=g(V_{lj}^k+\Delta V_{lj}^k) \tag{4}$$

$$XX_{lj}^k=\begin{cases} 1, R(0,1)\geqslant sig(V_{ij}^k) \\ 0, \text{其它} \end{cases} \tag{5}$$

其中,c_1,c_2 取常数,r_1,r_2,$R(0,1)$均是处于 0 和 1 之间的随机数。

粒子的位置经过公式(5)更新后,有可能违反每列和为 1($\sum_{i=1}^{m}s_{ij}=1$)的约束,因此,检查更新后的粒子的位置,看每列的和是否为 1,如果违反约束,则对当前列清 0,该列要么按随机概率取粒子最好历史记录位置,要么在该列速度最大的位上取 1。

粒子的位置完成更新后,把二进制的位置矩阵转换成用自然数表示的 N 维向量。

3.2 基于蚁群算法的网格资源分配

文[2-3]对基于蚂蚁和蚁群算法的网格资源分配进行了详细的讨论。在[2-3]的基础上进行了改进。适用于网格资源分配和任务调度的蚁群算法如下:

3.2.1 信息素初始化

蚁群算法部分的信息素初始化分成如下两部分:

(1)一部分是网格资源本身的实时可用的 MIPS,即新资源 j 加入网络时,要提供如下信息:资源 j 有 m 个处理器;pro_{ij} 表示资源 j 的第 i 个处理器的 MIPS;资源 j 的第 i 个处理器在 t 时刻的 CPU 空闲率 $\rho_{iCPUj}(t)$;参数包的大小 c;数据传输时间 s_j;CPU 类型,内存大小,磁盘空间等信息。

于是,下面的公式(6)反映了资源 j 的实时可用的固有计算能力和通信能力:

$$\eta_j(t)=\sum_{i=1}^{m}\rho_{iCPUj}(t)pro_{ij}+\frac{c}{s_j} \tag{6}$$

(2)另一部分就是粒子群算法寻优后的信息素部分。PSO 算法寻优结果的信息素部分为:

$$\tau'_j(t)=\theta\cdot\frac{length_i}{sumLength}\cdot Staticinfosum \tag{7}$$

其中 θ 为挥发系数,$length_i$ 为分配到该资源上执行的任务的长度之和,$sumLength$ 为所有作业长度之和,$Staticinfosum$ 是所有资源的 MIPS 之和。对于新加入网络的网格计算资源,这部分的信息素为零,即:

$$\tau'_j(t)=0 \tag{8}$$

那么,信息素初始值由两部分的和组成:

$$\tau_j(t)=\tau'_j(t)+\eta_j(t) \tag{9}$$

3.2.2 信息素更新

信息素的散布方式也就是信息素通过何种途径对最优解产生影响。在本文蚁群算法部分,采用如下两种信息素更新方式:

(1)局部信息素更新公式:

$$\tau_j(t)=(1-\frac{length_i}{sumLength})\tau_j(t) \tag{10}$$

其中 $length_i$ 为分配到该资源上执行的任务的长度,$sumLength$ 为所有作业长度之和。在构造解的过程中(同一次分配),被选择的资源信息素应该挥发,模拟该结点在处理作业。根据要处理的作业长度来定挥发的信息素大小。

(2)在构造完一个解后(一次分配完后),如果这个解是目前最优的,那么根据这个解的状态进行全局信息素更新。在把任务分配到相应的资源且处理完成后,相当于一次寻优后的信息素更新,而不是恢复构造前的信息素值。全局信息素更新的是根据前一次寻优结果来更新信息素的,更合理地利用信息素的正反馈机制。全局信息素更新公式:

$$\tau_j(t)=\rho\cdot\tau_j(t)+\theta\cdot\frac{length_i}{sumLength}\cdot Staticinfosum \tag{11}$$

ρ 是全局信息素挥发系数,θ 是寻优结果信息素的挥发系数。

3.3 融合算法

融合 PSO 算法与蚁群算法目的是利用这两个算法的优点,克服它们各自的缺点。整个算法过程分成两部分,首先用离散粒子群算法进行调度,选出一个较优的调度列表。其次,根据这个调度列表初始化蚁群算法部分的初始信息素。在作业 T 的每一个子任务 $t_j(j\in\{1,2,\cdots,n\})$处分别设置 1 个蚂蚁,作业分配资源 i,蚂蚁就在资源 i 上留下信息素,$\tau_i(t)$表示资源 i 的总信息素,每个蚂蚁选择资源 i 的概率为:

$$p_i=\frac{\tau_j}{\sum_{i=0}^{m}\tau_i} \tag{12}$$

如果选择资源 i,则按公式(4)进行局部分信息素更新。完整的融合算法设计如算法 1:首先,用离散粒子群算法找出一个初步的优化任务调度方案(算法中的(0、1、2)步);其次,用第一步获取的结果作为蚁群算法的初始信息素(3 步),再用蚁群算法进一步对任务调度进行优化(第 4 步以后),最后得到一个更优化的结果。

算法 1　基于融合 PSO 和 ACS 的网格资源分配

(0)确定离散粒子群算法的参数值

(1)初始化所有粒子的位置和速度

(2)while(结束条件)

(2.1)计算各粒子的适应值

(2.2)对每个粒子

(a1)计算适应度值,把粒子位置换成二进制表示

(a2)用它的适应值和个体极值 $pBest$ 比较,如果较好,则替换 $pBest$;

(a3)适应值与全局极值 $gBest$ 比较,如果较好,则替换 $gBest$;

(a4)按公式(4)和(5)更新粒子的速度和位置

(a5)检查粒子是否合法

(a6)把粒子位置换成自然表示法

(3)离散粒子群算法结束,根据产生的调度序列初始化蚁群算法的信息素(按公式(9))

(4)while(结束条件不满足)

(4.1)for(每只蚂蚁也即每个任务)

(b1)计算 $\tau^{\alpha}\cdot\eta^{\beta}$,即概率的分子项

(b2)计算分子项的和,即概率的分母

(b3)任务按概率公式(12)选择资源

(b4)把选择好的资源添加到分配列表

(b5)计算所选资源号的执行时间总和

(b6)局部更新信息素(按公式(10))

endfor

(4.2)第(4.1)步计算得到的分配列表在各资源上预计执行时间,选择最优值作为这次预分配的适应值

(4.3)如果适应值小于目前为止的适应值,则替换当前的分配列表,且更新信息素列表(按公式(11))

(5)整个算法结束

4　实验结果及性能评价

根据提出的融合算法,利用网格模拟工具 GridSim Toolkit 开发一个网格模拟系统进行模拟实验。网格模型取自[11],由异构的 10 个网格资源组成;3 个网格用户。

这里共进行了 10 个具有不同计算量的任务调度实验。在每一次实验中,每个用户随机地提交一个作业,每个作业包含 100,200,500,1 000,2 000,或 4 000 个子任务,用户 1 的每个子任务长度为 8 000,用户 2 每个子任务长度为 16 000,用户 3 每个子任务长度为 24 000。

将本文的融合算法、模糊粒子群算法[4]、离散粒子群算法 DPSO[5]、蚁群算法[2-3]一同进行实验比较。除各个算法特有参数不同外,其他均取值一样,参数设置如下:粒子规模 10;粒子群代数 20,融合算法中设置参数除粒子群代数为 10,其他与[4]一样;蚁群算法部分的参数 α=0.5,β=0.5,ρ=0.9,θ=0.8,迭代 10次。

实验时,提出的融合算法与离散粒子群算法[5]、模糊粒子群算法[4]、蚁群算法[2-3]经过 10 次实验后对实验数据进行统计,对各种工作量情况下的作业平均完成时间、每个资源按计算能力平均完成的计算量进行分析比较,实验结果如图 1 和图 2。

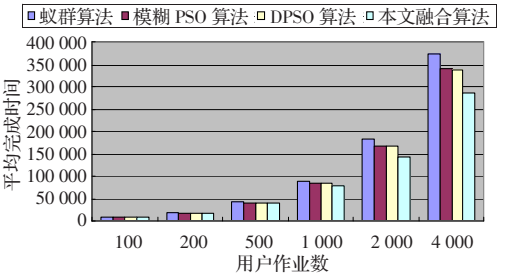

图 1　作业完成时间

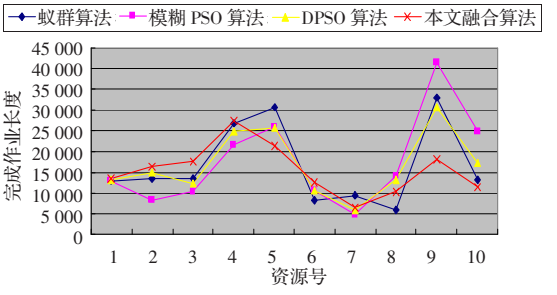

图 2　网格资源的工作量分布

图 1 是四种算法在不同作业量的平均作业完成时间,图 1 显示提出的融合算法在平均作业完成时间上优于其他三种算法,并随着作业量的增加,优势越明显。统计分析表明,融合算法的平均作业完成时间小于蚁群算法 12.7%,小于模糊粒子群算法 8.1%,小于离散粒子群算法 7.4%。

图 2 是四种算法在实验中所获得的工作量(作业长度)情况。由于网格资源的计算力不一样,为反映作业分配的均衡性,图 1 中的一个资源完成作业长度定义为:完成作业长度=计算资源所完成的总计算量/该计算资源总计算力,即每 MPIS 所完成的工作量。在图 3 中,融合算法负载曲线的波动情况要小于蚁群算法、模糊粒子群算法和离散粒子群算法(各算法的 AVEDEV 值(离散度):融合算法,3.3e+08;蚁群算法,8.6e+08;DPSO 算法,5.5e+08;模糊 PSO 算法,1.1e+09),表明融合算法在网格任务调度的均衡性方面要优于其他三种算法,从而能充分地利用网格系统的计算能力,因而比其他算法具有更少的完成时间。

以上实验结果表明,本文的融合算法在作业完成时间方面和负载均衡方面具有较大的优势,其原因在于,模糊粒子群算法与粒子群算法由于后期局部搜索能力差,对信息的利用不充分,优化效果不理想;而本文的融合算法既利用粒子群算法全局寻优能力,又利用蚁群的正反馈机制并能充分利用信息,克服了蚁群算法初始信息素匮乏的缺陷和粒子群算法后期局部搜索能力差的缺点,因此融合算法在解决网格资源调度问题具有良好的效率和优化效果。

5　结束语

本文提出一种融合离散粒子群算法与蚁群算法的网格资源分配的优化算法,可有效克服粒子群算法后期的局部搜索能力差和蚁群算法前期盲目搜索的缺陷,又充分利用粒子群算法的快速收敛性和蚁群算法的正反馈机制的优点。仿真实验表明该融合算法具有较好的优化调度和负载均衡的能力。

(下转 113 页)