基于 Nash 均衡的网格多调度节点的任务调度算法

易 侃[1],王汝传[1,2]
(1. 南京邮电大学计算机学院,江苏南京 210003;2 南京大学计算机软件新技术国家重点实验室,江苏南京 210093)

摘 要: 目前网格任务调度算法主要是针对 1 ×*n* 型即单调度节点多资源的网格环境,而针对 *m* ×*n* 型的网格环境研究较少. 论文用 *M*/ *M*/ 1 排队系统对 *m* ×*n* 型网格环境建模,然后以每个调度节点调度任务的平均完成时间为优化目标,提出了 *m* ×*n* 型网格环境任务调度的 Nash 均衡问题,并利用粒子群算法求得该 Nash 均衡解. 通过仿真验证了该算法在单位时间内平均完成的任务数,网络平均负载,以及系统的平均负载上均优于基于均匀调度策略的调度算法.

关键词: 网格; 任务调度; Nash 均衡; 粒子群算法; Repast
中图分类号: TP393 **文献标识码**: A **文章编号**: 0372-2112 (2009) 02-0329-05

Nash Equilibrium Based Task Scheduling Algorithm of Multi-schedulers in Grid Computing

YI Kan[1], WANG Ru-chuan[1,2]
(1. *College of Computer, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu* 210003, *China*;
2. *State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing, Jiangsu* 210093, *China*)

Abstract: At present, grid task scheduling Algorithms focus on 1 ×n type grid, namely one scheduler and n resources but neglect m ×n type grid. We built a Grid model of m ×n type grid using M/ M/ 1 queue system, and promoted the concept of task scheduling Nash equilibrium among multi-schedulers. The optimal objective of each scheduler is mean complete time per task. The Nash equilibrium took advantage of PSO to be solved. By simulations, we conclude that the new algorithm is better than the algorithm based on the mean scheduling strategies in mean finished task numbers per time, mean load of network and mean load of Grid resources.

Key words: grid; task scheduling; Nash equilibrium; PSO; repast

1 引言

目前网格任务调度算法多是针对 1 ×*n* 型即单调度节点多资源的网格环境,如获得调度近似最优解的启发式算法 Min-Min, Sufferage 调度算法[1],基于演化算法的调度算法[2],另外还有从网格经济[3],从网格实体之间相互信认[4]等诸多角度考虑的调度算法,然而由于多个调度节点在网格环境中可同时存在,它们各自独立又相互制约. 因此 1 ×*n* 型环境下的调度算法不适合在 *m* ×*n* 型环境中使用.

博弈论[5]为解决非协作竞争实体的策略选择提供了很好的思路,将博弈论引入解决网格资源的竞争正逐渐成为网格任务调度的热点,文献[6,7]从资源的价格因素考虑资源提供者和资源消费者之间的 Nash 均衡,文献[8,9]是从网格系统的负载的角度考虑多个调度节点之间的 Nash 均衡,虽然 Nash 定理已经证明对于有限策略集的博弈必有混合策略的最优解,即 Nash 均衡,但并没有说明如何求解 Nash 均衡. 文献[10]证明了 Nash 均衡的求解属于 PPAD 完全类问题,文献[11]提出了寻找纯策略 Nash 均衡解的方法,而对于混合策略的 Nash 均衡求解通常采用演化算法来搜索一个近似最优解,文献[12]提出利用遗传算法求解 Nash 均衡,与遗传算法相比粒子群算(PSO)法也是一种基于叠代的优化工具,而同遗传算法比较,PSO 的优势在于简单容易实现并且没有许多参数需要调整,因此本文首先结合文献[8,9]中的网格调度模型提出了网格多调度节点任务调度的 Nash 均衡问题,然后通过 PSO 求解该 Nash 均衡并将其解作为每个调度节点的调度策略.

2 算法思想

假设网格环境是由 *m* 个调度节点和 *n* 个资源组成,每个调度节点 *i* 提交的任务数满足 Poisson 过程且

收稿日期:2008-03-06;修回日期:2008-07-06
基金项目:国家自然科学基金(No. 60573141, No. 60773041);国家高科技 863 项目(No. 2006AA01Z201, No. 2007AA01Z404, No. 2007AA01Z478);江苏省高技术研究计划(No. BG2006001);江苏省自然基金项目(No. BK2008451);江苏省高校自然科学研究计划(No. 07KJB520083)和江苏高校科技创新计划项目(No. CX08B-085Z)

平均任务数为 λ_i,每个资源 j 服务的任务数也满足 Poisson 过程且平均服务率为 u_j,对每个资源我们可以利用 $M/M/1$ 排队系统建模.现假设调度节点 i 提交任务到调度节点 j 的概率为 p_{ij}, $p_{ij}\geq 0$ 且 $\sum_j p_{ij}=1$,那么要使得每个资源稳定,则必须满足 $\sum_i p_{ij}\lambda_i \leq u_j$,而整个网格系统如果要稳定,必须满足 $\sum_i \lambda_i \leq \sum_j \mu_j$.

由于每个资源是一个 $M/M/1$ 排队系统,因此每个资源的任务平均逗留时间为:

$$W_j(p)=1/(u_j-\sum_i p_{ij}\lambda_i) \tag{1}$$

其中 $p=(p_1,p_2,\cdots,p_m)$, $p_i=(p_{i1},p_{i2},\cdots,p_{in})$.此外由于每个调度节点是通过网络连接到每个资源,因此需要考虑调度节点与资源之间的数据传输延迟,假设调度节点 i 中任务的平均传输量为 b_i,调度节点 i 到资源 j 的平均带宽为 c_{ij},那么调度节点调度任务到资源 j 的平均数据传输时延为 $l_{ij}=b_i/c_{ij}$.综上,调度节点 i 调度任务到资源 j 上执行的平均完成时间为:

$$R_{ij}(p)=W_j(p)+l_{ij} \tag{2}$$

考虑到每个调度节点以一定的分布将任务调度到 m 个资源上,因此调度节点 i 调度任务的平均完成时间为:

$$\begin{aligned}R_i(p)&=\sum_j p_{ij}*R_{ij}(p)\\&=\sum_j p_{ij}(1/(u_j-\sum_k p_{kj}\lambda_k))+\sum_j p_{ij}l_{ij}\end{aligned} \tag{3}$$

因此我们可以得到网格多调度节点调度任务的目标是:使得每个调度节点任务平均完成时间最小,即:

$$Min(R_i(p)),\text{且}\sum_{j=1}^{n}p_{ij}=1;\ p_{ij}\geq 0;\ \sum_{j=1}^{n}p_{ij}\lambda_i\leq\mu_j \tag{4}$$

实际上,由于每个调度节点之间并没有交互,网格系统和任务的属性是它们的共同知识,因此上述问题是一个多调度节点非合作静态完全信息博弈的问题,其中每个调度节点均是博弈的参与者,每个参与者的纯策略集合均为这 n 个资源的集合,而每个参与者的效益函数则为 $R_i(p)$,如果存在一个 p^* 对于所有的调度节点 i 都有 $Min(R_i(p))$,那么 p^* 就是网格多调度节点任务调度的最佳混合策略.

定义 1 网格多调度节点任务调度 Nash 均衡

(1) 博弈的参与者为 m 个非合作的调度节点,其提交任务的速率为 $\lambda=(\lambda_1,\cdots,\lambda_m)$;

(2) 参与者的纯策略集为 n 个资源组成,每个资源的平均服务率为 $u=\{u_1,\cdots,u_n\}$;参与者的混合策略集为每个参与者纯策略集对应的概率组成的向量 $p=\{p_1,p_2,\cdots,p_m\}$,其中 p_i 称为参与者 i 的混合策略;

(3) 参与者 i 的效益函数为 $R_i(p)=\sum_j p_{ij}(1/(u_j-\sum_k p_{kj}\lambda_k))+\sum_j p_{ij}l_{ij}$;

网格多调度节点任务调度的 Nash 均衡为 p^*,当且仅当对于所有的 i,满足 $R_i(p_i,p_{-i}^*)\leq R_i(p_i^*,p_{-i}^*)$,且 $\sum_{j=1}^{n}p_{ij}=1$; $p_{ij}\geq 0$; $\sum_{j=1}^{n}p_{ij}\lambda_i\leq\mu_j$.

根据 Nash 均衡的解释,如果我们求得网格多调度节点任务调度的 Nash 均衡解,那么每个调度节点选择的调度策略都是其他调度节点调度策略的最优反应,那么在相互竞争的非合作网格环境中,每个调度节点将保证自己会获得最佳的效益,即最小的任务平均完成时间.

3 网格多调度节点任务调度 Nash 均衡的 PSO 求解

根据定义 1 我们利用 PSO 算法求解该 Nash 均衡解,并将该算法简称 NEPSO,算法中每个粒子定义为一个博弈的混合策略,可展开为一 $m*n$ 的矩阵,那么第 t 次迭代具有 K 个粒子的种群 $p^1(t),\cdots,p^K(t)$ 展开如下:

$$p^1(t)=\begin{pmatrix}p_1^1(t)\\p_2^1(t)\\\cdots\\p_m^1(t)\end{pmatrix}_{m*n},\cdots,p^K(t)=\begin{pmatrix}p_1^K(t)\\p_2^K(t)\\\cdots\\p_m^K(t)\end{pmatrix}_{m*n} \tag{5}$$

NEPSO 的适应度函数为:

$$f(p)=\sum_i \max\{\max\{R_i(p_{i,-i})-R_i(p)\},0\} \tag{6}$$

该适应函数是根据如下的实事得出:从每个参与者的角度,如果他主动改变策略获得的最大收益仍然比采用当前的策略获得的收益小,那么它不希望改变当前策略即他对适应度的贡献为 0,否则他希望改变现状,而且当收益的差越大,他越希望改变现状且对当前策略的适应度的贡献应越大.显然当 p 为最优解 p^* 时, p 的适应度函数 $f(p)=0$.因此我们希望 $f(p)$ 越小越好.由于当 p 确定时, $R_i(p)$ 是常量,因此式(6)求解的关键是求解 $\max\{R_i(p_{i,-i})\}$,即在 $p_j=(p_{j1},\cdots,p_{jm})$, $j\neq i$,不变的情况下,变化 $p_i=(p_{i1},\cdots,p_{in})$,使得 $R_i(p_{i,-i})$ 的值最大.根据式(3)有

$$\begin{aligned}R_i(p_{i,-i})&=\sum_j p_{ij}(1/(u_j-\sum_k p_{kj}\lambda_k))+\sum_j p_{ij}l_{ij}\\&=\sum_j p_{ij}(1/(u_j-\sum_{k\neq i} p_{kj}\lambda_k-p_{ij}\lambda_i))+\sum_j p_{ij}l_{ij}\\&=\sum_j p_{ij}(1/(\Delta_j-p_{ij}\lambda_i))+\sum_j p_{ij}l_{ij}\end{aligned}$$

(令 $\Delta_j=u_j-\sum_{k\neq i}p_{kj}\lambda_k$)

因此,$\max\{R_i(p_{i,-i})\}$ 转化为求解多元函数:

$$\max(h(x_1,\cdots,x_n))=\max(\sum_{j=1}^{n}p_{ij}(1/\Delta_j-p_{ij}\lambda_i))+\sum_{j=1}^{n}p_{ij}l_{ij},$$

$$\text{且}\sum_{j=1}^{n}p_{ij}=1;\ p_{ij}\geq 0;\ \sum_{j=1}^{n}p_{ij}\lambda_i\leq\mu_j \tag{7}$$

其中 $(p_{i1},\cdots,p_{in})$ 是未知变量, Δ_j, λ_i, l_{ij} 是已知变量的条件极值问题,它将 $m*n$ 维 Nash 均衡求解问题降低到

式(7)的 n 维非线性约束最优化问题,该问题有很多进化算法可解得近似值,我们采用文献[13]中描述的 PSO 方法解决该问题.

一般的,粒子群的迭代过程采用式(8)所示的标准的迭代算法,其中第 i 个粒子在第 t 次迭代的最优值为 $p^i_{best}(t)$,全局最优粒子为 G_{best},那么第 i 个粒子的速度迭代函数为

$$V^i(t+1) = wV^i(t) + c_1 r_1(p^i_{best}(t) - p^i(t)) + c_2 r_2(G_{best} - p^i(t)) \quad (8)$$

其中 w 是惯性权重,c_1, c_2 是学习因子一般取 2,r_1, r_2 是(0,1)之间的随机数. 第 i 个粒子的更新函数为:

$$p^i(t+1) = p^i(t) + V^i(t+1) \quad (9)$$

根据定义 1,由于 p 的求解空间不是 $(0,1)^{m*n}$ 的全空间,带有两个约束条件,因此如何处理这两个约束条件是求解的关键.

定义 2　网格多调度节点任务调度 Nash 均衡的可行解

当解 p 满足 $\forall i', i, j, p^{i}_{ij}{}' \geq 0, \sum_j p^{i}_{ij}{}' = 1$ 时称解 p 为规范解.

当解 p 满足 $\forall i', i, j, u_j < \sum_{i=1}^{m} p^{i}_{ij}{}' * \lambda_i$ 时,称 p 为稳定解.

当解 p 既是规范解又是稳定解时称解 p 为可行解.

如果粒子 i 迭代的中间结果不是可行解即不满足约束条件,那么我们需要处理约束,文献[14]总结了四种处理约束的方法,文献[13]采用了其中的只保留可行解的方法. 对于非规范解,我们通过如下的式(10)和式(11)两步对其修正;而对于非稳定解,我们采用文献[15]提出的方法,适应度最好的不稳定解以一定的概率 ζ 丢弃,而其他不稳定解直接丢弃,使得算法加快收敛.

步骤 1:
$$p^{i}_{ij}{}' = \begin{cases} 0, & p^{i}_{ij}{}' < 0 \\ 1, & p^{i}_{ij}{}' > 1 \\ p^{i}_{ij}{}', & 0 \leq p^{i}_{ij}{}' \leq 1 \end{cases} \quad (10)$$

步骤 2:
$$p^{i}_{ij}{}' = \frac{p^{i}_{ij}{}'}{\sum_j p^{i}_{ij}}, \quad (11)$$

4　MSS_NE 算法描述

多调度节点网格任务调度算法简称 MSS_NE(Multi-Schedulers Scheduling-alogrithm Based on Nash Equilibrium)分为两步,首先通过 NEPSO 算法计算多调度节点的 Nash 均衡,其次每个调度节点根据 Nash 均衡解中相应的调度策略提交任务. 表 1 给出了 NEPSO 算法的描述,表 2 给出了 MSS_NE 算法的框架.

5　MSS_NE 仿真实验结果

我们采用 Repast[16] 仿真工具构造了如图 1 所示的具有 10 个调度节点(S1-S10)和 15 个网格资源(R1-R15)的网格多调度节点仿真系统. 通过对每个调度节点采用 MSS_NE 算法和采用均匀调度算法 MSS_MS(Mean Strategies)算法比较它们的性能,其中 MSS_MS 算法的思想是每个调度节点以资源总数的倒数为概率向每个资源提交任务.

表 1　NEPSO 算法的描述

NEPSO 算法输入:
(1) 种群大小 K,最大迭代次数 MAXITER,最小误差 ε;
(2) 惯性权重 w;学习因子 c_1, c_2,一般都取 2;
(3) 调度节点个数 m,资源个数 n;调度节点提交的平均任务数 $\lambda = (\lambda_1, \cdots, \lambda_m)$,资源的平均服务率 $u = (u_1, \cdots, u_n)$,任务的平均数据量 $b = (b_1, \cdots, b_m)$,调度节点与资源之间的平均带宽 $l = (l_1, \cdots, l_m)$,其中 $l_i = (l_{i1}, \cdots, l_{in})$.
NEPSO 算法输出:最优的混合策略 $p^* = G_{best}$;迭代次数 t;误差限 ε.
NEPSO 算法描述:
Step1　初始化种群. $\forall i$ 表示第 i 个粒子,$p^i(0)$ 中的每个分量取(0,1)的随机值,然后对 $p^i(0)$ 规范化处理以及稳定化处理;
Step2　根据式(8)计算第 i 个粒子的速度 $V^i(t+1)$,并根据式(9)更新种群中的每个粒子 $p^i(t+1)$;
Step3　根据式(10)和(11)对 $p^i(t+1)$ 规范化以及稳定化处理;
Step4　$\forall i, p^i(t)$, t 是迭代的次数,根据式(6)计算 $p^i(t)$ 的适应度,比较求得 $p^i_{best}(t)$ 和 G_{best};
　Step4.1　对 $p^i(t)$ 的 m 个分量分别利用文献[13]提出的 PSO 算法求得式(7);
　Step4.2　根据式(6)求得每个粒子的适应度;
　Step4.3　比较求得 $p^i_{best}(t)$ 和 G_{best};
Step5　迭代次数 t 增加 1;
Step6　如果 $f(G_{best}) \leq \varepsilon$,或者 $t = MAXITER$,则返回结果 G_{best},否则继续 Step2;
结束.

表 2　MSS_NE 算法的框架

Step1　每个调度节点根据 NEPSO 算法计算最优的混合策略 p;
Step2　每个调度节点根据各自的混合策略 p_i 提交任务,即通过轮盘赌算法在 n 个资源中选择一个资源提交任务,其中第 i 个调度节点的轮盘赌算法的 JAVA 实现原代码如下:

```
public GridNode rouletteWheelSelection() {
    double selectionThreshold = Math.random();
    double[][] summedStrategy = new double[m][n];
    for(int i=0;i<m;i++){
      for(int j=0;j<n;j++){
        if(j==0) summedStrategy[i][j] = p[i][j] else summed-
        Strategy[i][j] += p[i][j];
      }
    }//每个调度节点混合策略累加;
    for r(int i=0;i<m;i++){
      for (int j = 0;j < n;j++) {
        if (summedStrategy[i][j] >= selectionThreshold) {
          GridNode aNode = (GridNode) gridNodeList.get(j);
          return aNode;
        }
      }
    }//轮盘赌算法选择一个节点;
}
```

Step3　当检测到网格环境变化时,返回到 step1 重新计算混合策略.
*注:目前只考虑静态的网络环境.
结束.

现假设每个调度节点提交任务的平均任务量为 5000,平均传送 250Mbyte 的数据,调度节点与网格资源之间的通信带宽在[15,200]kbyte/tick 之间随机取值,

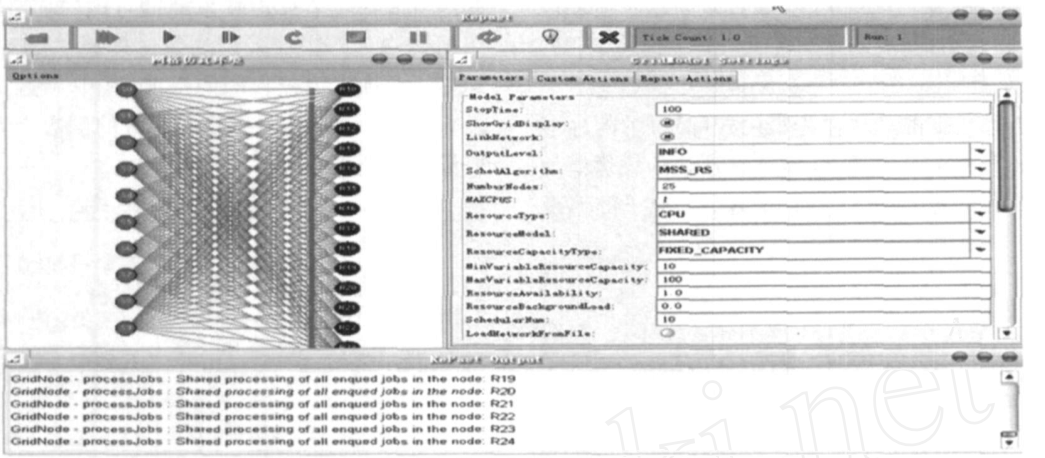

图1 基于Repast的网格仿真环境

每调度节点提交的任务数满足 *Poisson* 分布且提交任务的平均速率见表 3,15 个网格资源服务的任务数也满足 *Poisson* 分布且平均服务率见表 4.

表 3 10 个调度节点提交任务的平均速率

调度节点	1	2	3-5	6	7	8	9-10
提交任务的平均速率 λ_i	0.35	0.2	0.1	0.06	0.05	0.02	0.01

表 4 15 个网格资源的平均服务速率

资源(*CPU*)	1-3	4-7	8-10	11-15
服务平均速率 u_j	0.01	0.02	0.033	0.1

我们将 Nash 均衡的调度策略与均匀调度策略在规定的仿真时间(非实际时间)内,即调度算法运行的总时间(totalTime)为 2000ticks,从以下三个角度进行比较:

(1) 调度节点平均完成的任务数(单位:任务数/tick):

$$e_i = totalTaskNum_i / totalTime$$

其中 $totalTaskNum_i$ 是调度节点 i 在 $totalTime$ 时间内完成的任务总数;根据定义 1 网格多调度节点 Nash 均衡调度节点的目标是任务平均完成的时间最少,那么反过来调度节点平均完成的任务数应最多.

(2) 网络平均负载(单位:Mbyte):

$$avgNetworkLoad = (\sum_i \sum_j edge_{ij}) / m * n$$

其中 $edge_{ij}$ 是每 *tick* 调度节点 i 与资源 j 之间的数据传输量;

(3) 系统平均负载(单位:任务量/每 *tick*):

$$avgCPULoad = (\sum_j CPUload_j) / n$$

其中 $CPUload_j$ 是每 *tick* 资源 j 的当前排队的任务量.

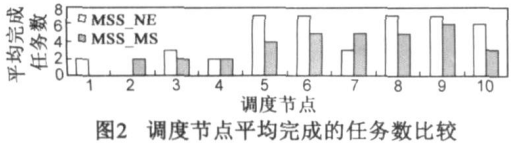

图2 调度节点平均完成的任务数比较

由图 2 可得,除了节点 2,4 和 7,其他使用 MSS_NE 算法的调度节点平均完成的任务数均大于使用 MSS_MS 算法的调度节点,总体表现为具有 10 个调度节点的网格环境,MSS_NE 算法平均完成的任务总数更多.这是因为 Nash 均衡的混合策略不同与传统意义上的最优策略,它的优势体现在整体上的优,局部可能比其他方法劣.

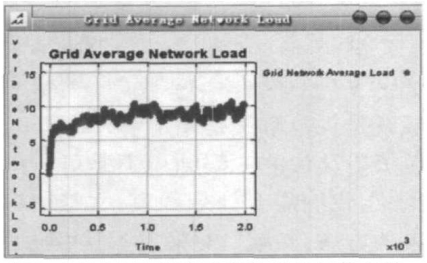

图3 MSS_NE算法对应的平均网络负载

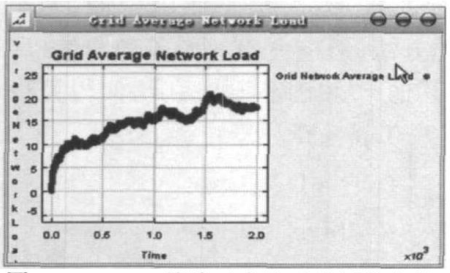

图4 MSS_MS算法对应的平均网络负载

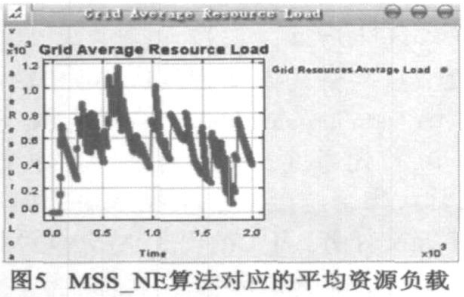

图5 MSS_NE算法对应的平均资源负载

由图 3 和图 4 可得,在同样的网格环境下由于 MSS_NE 算法考虑了网络环境的影响,其负载稳定在 10 左右,而使用 MSS_MS 算法网格系统的网络负载则上

升至 20 左右.

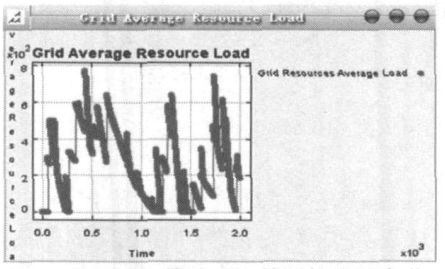

图6　MSS_MS算法对应的平均资源负载

由图 5 和图 6 可得,使用 MSS_NE 算法的网格系统比使用 MSS_MS 算法的网格系统的平均资源负载更高,这主要是因为前者完成的任务数更多,另外,前者的资源平均负载大部分时间维持在 600 附近,而后者则是从 0 到 800 之间剧烈的振荡,即从系统的稳定性角度前者比后者更优.

6　总结与展望

基于 Nash 均衡的网格多调度节点任务调度算法利用博弈理论将每个调度节点比作博弈的参与者,每个调度节点综合考虑资源和网络的性能并以最大化其平均任务完成时间为目标,它们的 Nash 均衡则是每个调度节点采用的混合策略. 通过仿真我们验证了该调度算法较均匀调度策略调度算法的优越性. 然而,Nash 均衡的求解目前仍然没有很好的解决方法,混合策略的 Nash 均衡解也不唯一,我们通过进化算法得到一个近似的值而且并没有证明该算法的收敛性. 论文的下一步工作是更快、更精确的求解 Nash 均衡.

参考文献:

[1] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, Richard F Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems[A]. 8th Heterogeneous Computing Workshop (HCW '99)[C]. San Juan, Puerto Rico: IEEE Computer Society Press, 1999. 25 - 55.

[2] Vincenzo D M, Marco M. Sub optimal scheduling in a grid using genetic algorithms[J]. Parallel Computing, 2004, 30(5-6): 553 - 565.

[3] Buyya Rajkumar. Economic-based Distributed Resource Management and Scheduling for Grid Computing[D]. Melbourne, Australia: Monash University, 2002.

[4] 王伟,曾国荪. 一种基于 Bayes 信任模型的可信动态级调度算法[J]. 中国科学(E 辑:信息科学), 2007, 37(02): 285 - 296.

[5] 施锡铨. 博弈论[M]. 上海:上海财经大学出版社, 2000. 1 - 50.

[6] Rzadca Krzysztof, Denis Trystram, Adam Wierzbicki. Fair game-theoretic resource management in dedicated grids[A]. Cluster Computing and the Grid, 2007. CCGRID 2007[C]. Rio de Janeiro, Brazil: IEEE Press. 2007. 343 - 350.

[7] 陶军,吴清亮,吴强. 基于非合作竞价博弈的网络资源分配算法的应用研究[J]. 电子学报, 2006, 34(02): 241 - 246. Tao Jun, Wu Qing-liang, Wu Qiang. Application research of network resource allocation algorithm based on non-cooperative bidding game[J]. Acta Electronica Sinica, 2006. 34(02): 241 - 246. (in Chinese)

[8] Altman Eitan. Nash equilibria in load balancing in distributed computer systems[J]. International Game Theory Review, 2002. 4(2): 1 - 10.

[9] Daniel Grosua, Anthony T Chronopoulosb. Noncooperative load balancing in distributed systems[J]. Parallel Distrib. Comput, 2005, 65(09): 1022 - 1034.

[10] Constantinos Daskalakis, Paul W Goldberg, Christos H Papadimitriou. The complexity of computing a nash equilibrium[A]. STOC 2006[C]. Seattle, WA, USA: ACM press, 2005. 71 - 78.

[11] Datta Ruchira S. Using computer algebra to find nash equilibria[A]. ISSAC '03[C]. Philadelphia, Pennsylvania USA: ACM press, 2003. 74 - 93.

[12] Riechmann Thomas. Genetic algorithm learning and evolutionary games[J]. Journal of Economic Dynamics & Control, 2001, 25(6): 1019 - 1037.

[13] Xiaohui Hu, Russell Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization[A]. Proceedings of the Sixth World Multiconference on Systemics[C]. Orlando, USA: Cybernetics and Informatics, 2002. 203 - 206.

[14] Slawomir Koziel, Zbigniew Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization[J]. Evolutionary Computation, 1999, 7(1): 19 - 44.

[15] Efrén Mezura-Montes, Carlos A, Coello Coello. A simple multi-membered evolution strategy to solve constrained optimization problems[J]. IEEE Transactions on Evolutionary Computation, 2005, 9(1): 1 - 17.

[16] Michael J North, Nicholson T Collier, Jerry R Vos. Experiences creating three implementations of the repast agent modeling toolkit[J]. ACM Transactions on Modeling and Computer Simulation, 2006, 16(1): 1 - 25.

作者简介:

易　侃　男,1981 年生,江苏南通人,博士研究生,研究方向为计算机网络、网格技术、p2p 技术和移动代理技术等.

王汝传　(见本期第 277 页)