一种具有自适应能力的任务分配系统的设计*

黎 亮* 杨国纬 陈光礻禹
（电子科技大学计算中心 成都 610054）

【摘要】 当分布式系统中各处理结点相互完全独立，并且在对其他结点的状态不是很了解的情况下，利用组合数学和随机过程的方法提出了一种机制。该机制能够使各结点在没有全局管理者的参与下，仅根据系统运行的反馈来决定下一步的任务，从而使整个系统的任务分配达到最佳。该方法能很好地处理最普遍的任务分派问题。

关 键 词 独立结点； 任务分配； 自我优化； 马尔科夫链； 算法
中图分类号 TP338.8

在具有多个独立结点的分布系统(如一个计算机网络)中，能够让各独立结点协调各自任务，从而使整个系统达到最佳任务分配状态是十分重要的。传统的任务分配方法是，由一个中央结点统一安排完网络中所有其他结点的任务。这种方式存在着一旦中央结点失效，则整个系统将瘫痪的问题。既使不考虑这个问题，这样的系统也只能适用于小型网络。近年来的研究表明，对于大规模的网络，单一的集中控制是不可行的。大量的研究成果表明，在某种规则的制约下，网络各部分进行自我局部优化，最终得到整个网络的全局最优是大型网络中资源和任务分配的最佳途径[1~3]。假设一个有 N 个独立结点的系统中的每个结点可承担的任务可能有 m 个($0<m<\infty$)，称为一个(N，m)系统。若在(N，m)系统中，每个独立结点每次只能承担一个任务。由于(N，m)系统 N 个不清楚相互情况的人(或结点)，能通过自学合理地选择分配来做 m 类工作(或使用 m 类资源)，故它可以被看成是一种最一般的、具有普遍适应能力的任务分派系统。

本文从理论上对(N，m)系统进行了分析讨论，并提出一个能用于分布式系统进行任务分配的方法，它比文献[4]在适应范围上广泛得多且在效率上也高得多。在此基础上得出的具体实现算法，经多次验证，切实可行，能够解决最普遍的任务分派问题。

1 (N，m)系统任务分配算法

假若每个独立结点每次完全随机地在 m 个任务中任选一个任务，则每个任务被选中的概率为 $1/m$。定义 n_j^i 为系统处在 i 状态时，选择 j 任务的独立结点个数，定义 $f_j^i=n_j^i/N$，r_i 为系统处于状态 i 时的奖励概率($0\leqslant r_i\leqslant 1$)。这样系统状态 i 就可表示为$F^i=(f_1^i, f_2^i, \cdots, f_m^i)$。系统进入状态 i 的概率为

$$\frac{N!}{n_1^i!\times n_2^i!\times n_3^i!\times\cdots\times n_m^i!}\times\left(\frac{1}{m}\right)^N$$

系统总的状态数为 m^N，如果每个独立结点承担每个任务的效果完全一样，则此状态数可降为 C_{N+m-1}^N。从以上公式可知，系统进入状态（$1/m$，$1/m$，…，$1/m$）及其附近状态的概率最大；

* 1998 年 2 月 26 日收稿，1998 年 3 月 20 日修改定稿
* 男 30岁 硕士 讲师

而当状态中某个 f_i 趋于 1(则其他的 f_j 必然趋于 0)时，系统进入其中的概率最小。随着 N 的增大，系统进入这两种状态之间的概率差也将增大。这说明一个随机分配的(N，m)系统将把大量的时间耗费在那些所有 f_i 都趋近于 $1/m$ 的状态上，称为 $1/m$ 陷阱。当然并不是所有的系统在此状态下都具有最大的奖励概率。因此本文的目的之一就是寻找一种机制，能使系统可以摆脱 $1/m$ 陷阱，而保持在具有最大奖励概率的状态上。

解决上述问题的基本方法是使用 K 状态马尔科夫链，即每个独立结点使用一个大小为 K 的能够做线性转移的马尔科夫链来调整自己的行为(所承担的任务)。我们为每个独立结点定义一个二元组（*action*，*step*）代表其自身的状况；其中 *action* 代表结点当前所承担的任务，*step* 代表结点承担此任务的稳定程度；显然有 $1 \leqslant action \leqslant m$ 和 $1 \leqslant step \leqslant K$。初始化时，每个独立结点的 *step* 都设置为 1，其 *action*(任务)则完全随机地选择一个 1 ~ m 的数。另外所有的独立结点都必须遵循下列规则：

算法 1

1）若这次得到的是奖励，则每个独立结点增加各自的 *step*

$$step = step + 1$$

2）若这次得到的是惩罚，则每个独立结点减少各自的 *step*

$$step = step - 1$$

3）对所有的独立结点，若 $step > K$ 则

$$step = K$$

4）对每个独立结点，若 $step < 1$ 则

$$step = 1$$

$$action = 1 \sim m \text{ 之间的一个随机数}$$

即按此算法独立结点所承担的任务仅当 $step < 1$ 时才改变，并且完全随机地改变成任意一个任务。另外按此算法所有的独立结点都将同时改变或保持其任务。

2　任务分配算法的改进

一般来说(N，m)系统共有 C_{N+m-1}^{N} 个状态(若每个独立结点承担同一任务具有不同的权，则状态数最多可上升为 m^N)。由于状态空间很大，因此算法 1 在进行状态筛选时，会花去相当多的时间。特别是如果在非最优状态上停留的时间过长，则会严重影响到系统进入最优状态的时间，如图 1 所示(在(N, 5)系统中：If $f_1 \geqslant 0.51$ then $r = 0.85$, else $r = 0.6$)。当 $K = 1$ 时，系统进入最优状态的时间比 $K = 15$ 时的时间短得多。这是由于系统中所有状态的奖励概率都大于 0.5；这样，对系统来说，停留在任何状态的概率都比脱离该状态的概率大。因此，系统要从 $f_1 < 0.51$ 和 $r = 0.6$ 的那些状态中出来并进入新状态将花去大量的时间。这反映了局部最优值将影响全局最优点的搜寻。可以看出 K 设置越大，系统进入最优状态后将越稳定，而 K 设置越小越便于系统进行状态选择。由此，我们针对（N，m）任务分配系统提出一个新的算法。系统中的所有独立结点都必须服从以下规则：

算法 **2**

每个独立结点使用一个如下的七元组来代表其当前情况：

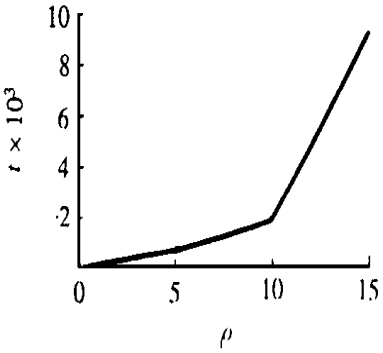

图 1 系统进入最优状态的时间随 K 值增大而增大($N = 5$ 时)

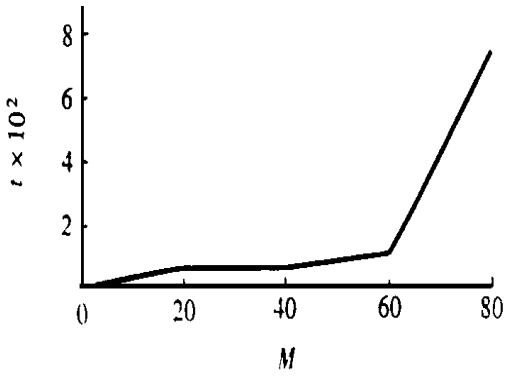

图 2 系统进入最优状态的时间随着 N 值增大而增大

(*action*, *step*, *satisfy*, *total-account*, *reward-account*, *memory*, *statistics-step*)

action ∈[1..*m*] :其初始值为 1 ~ *m* 之间的随机数

step ∈[1..*memory*] : 代表独立结点承担目前任务(*action*)的稳定程度, 其初始值为 1

statistics-step : *total-account* 的上限

total-account :奖惩的统计数, 其初始值为 0

reward-account :奖励次数的统计数, 其初始值为 0

satisfy ∈[0..1] :满足要求的奖励概率值

memory : *step* 的上限值, 其初始值为 1

对于每个独立结点来说:

1) 如果这次得到了奖励, 则

$$step = step + 1 \qquad total\text{-}account = total\text{-}account + 1$$
$$reward\text{-}account = reward\text{-}account + 1$$

2) 如果这次得到了惩罚, 则

$$step = step - 1 \qquad total\text{-}account = total\text{-}account + 1$$

3) 如果 $total\text{-}accout \geqslant statistics\text{-}step$, 则

$$temp = memory + reward\text{-}account - (total\text{-}account \times satisfy)$$
$$\text{if } (reward\text{-}account / total\text{-}account) \geqslant satisfy$$
$$\text{then } memory = \lceil temp \rceil \text{ else } memory = \lfloor temp \rfloor$$
$$total\text{-}account = 0 \qquad reward\text{-}account = 0$$

4) 如果 $step > memory$, 则

$$step = memory$$

5) 如果 $step < 1$, 则

$$step = 1 \qquad memory = 1$$
$$action = 1 \sim m \text{ 之间的一个随机数}$$
$$total\text{-}account = 0 \qquad reward\text{-}account = 0$$

在算法 2 中, *memory* 参数相当于算法 1 中的参数 K; *memory* 设置得越大越利于稳定在最优

状态;设置得越小越利于状态筛选;故算法 2 中对此参数的值进行了反复调整。从图 2 可以看出算法 2 的效率要比算法 1 明显高得多。此外,算法 2 可以并发执行,每个独立结点都可以在每一个处理器上执行自己的算法,而每个处理器上的 N 个结点的算法也具有相对的独立性,其总的并行度高达 $C_{N+m-1}^{N}\times N$,具有很好的并行性。

3　结 束 语

本文提出了一种能在分布系统中广泛使用的、具有很好健壮性的任务分配方法。无论系统的奖惩函数有多么复杂,这种方法都可以使系统中相互不了解情况的各独立结点 在没有全局管理者的情况下,通过自适应协调一致,最终使系统的任务分配达到最优状态。

感谢电子科技大学青年科技基金对本文课题的资助。

参　考　文　献

1　Lazar A A, Orda A, Pendarakis D E. Virtual path bandwidth allocation in multi—user networks. IEEE INFOCOM' 95, 1995:312 ~ 320

2　Shenker S J. Making greed work in networks: a game—theoretic analysis of switch service disciplines. IEEE/ACM Trans Networking, 1995, 3:819 ~ 831

3　Korilis Y A, Lazar A A, Orda A. Achieving network optima using stackelberg routing strategies. IEEE/ACM Trans Networking, 1997, 5:161 ~ 173

4　Tung Brian, Kleinrock L. Using finite state automata to produce self—optimization and self—control. IEEE Trans Parallel and Distributed System, 1996, 7:439 ~ 448

An Assignment Method for Distributed Systems

Li Liang　　Yang Guowei　　Cheng Guangju

(Computer Center, UESTC of China　Chengdu　610054)

Abstract　In a distributed system without a central administrator, suppose all independent processing units do not know the conditions of one another, a new system is designed to undertake the tasks in satisfying proportions to cooperatc for the best results, in which only self-learning is used. This paper also analyzes the performance of this system by Markov chains, and presents a robust method of self-learning for independent processing units in this kind of systems. The method can also be used to solve the general assignment problem.

Key Words　independent processing unit;　assignment problem;　optima;　Markov chain;　algorithm

编辑　叶　红