第36卷 第20期 计 算 机 工 程 2010年10月
Vol.36 No.20 Computer Engineering October 2010
·软件技术与数据库· 文章编号：1000—3428(2010)20—0090—03 文献标识码：A 中图分类号：TP311

贪心线性推移负载平衡算法

吴荣腾
(闽江学院计算机科学系，福州 350108)

摘 要：针对环与线性阵列的负载平衡速度较慢与迁移量较大的问题，提出一种贪心线性推移平衡算法。该算法适用于任何具有哈密尔顿通路的图结构网络。其平衡过程的负载迁移量一般不大，平衡负载速度较快。对二维网状网等网络结构的贪心线性推移平衡算法进行改进，得到分二阶段的贪心线性推移平衡算法。实验结果表明，此类改进在平衡条件减弱时能较大地提高算法的时间性能。

关键词：负载平衡；互连网络；任务调度；异构系统；负载迁移

Greedy Linear Shift Load Balance Algorithm

WU Rong-teng
(Department of Computer Science, Minjiang University, Fuzhou 350108, China)

【Abstract】 Greedy linear shift balancing algorithm is presented for ring or linear array to reduce balancing time and the amount of the transferred load. The algorithm can be used for balancing any system in which internetwork contains at least one Hamilton path. It need not a large amount of load transferred among nodes or a large amount of the consumed time. In addition, two-stage greedy linear shift balancing algorithm is presented to improve the greedy linear shift balancing algorithm for mesh etc. Experimental results show that the two-stage linear shift balancing algorithm can reduce execution time when the balancing condition is weaken.

【Key words】 load balance; interconnection network; task scheduling; heterogeneous system; load transfer

1 概述

随着系统结构的不断变化、网络速度的极大提高以及集群、分布式系统等异构系统的提出并得到实际的应用，异构系统的任务调度与负载平衡问题受到了更多的重视[1-2]。

系统的异构性主要体现在两方面：各处理器的处理速度(能力)不同和网络链路的通信能力不同。本文仅研究处理器处理能力不同而网络链路通信能力相同的异构系统。

异构互连网络系统中的负载平衡问题：对具有 n 个节点的网络，假定每个节点 i 的处理能力为 c_i，并具有任务负载 w_i，若负载平衡过程中没有新的任务产生，则经过负载迁移达到平衡状态时各节点 i 的任务负载为：

$$\tilde{w}_i = c_i \sum_{k=0}^{n-1} w_k \Big/ \sum_{k=0}^{n-1} c_k$$

此时每个处理器的执行时间都相同。显然，当所有的 c_i 都相等时，系统为同构系统。因此，同构系统可看成异构系统的特例。

文献[3-4]为解决上述问题分别提出了基于扩散通信模式的负载平衡算法，这2种算法的计算复杂性与关联矩阵的特征值相关。

文献[3]的负载平衡算法使系统达到平衡的最佳迭代步数不高于处理器数目，但额外开销较大；同时给出的另外2种方法使额外开销变小，但迭代步数不小于处理器数目。

文献[4]负载平衡算法的计算开销与一般化的扩散关联矩阵的特征值有关。

文献[5-6]提出了维交换通信模式的平衡算法。该方法仅对超立方体结构类型的网络具有良好的性能。

本文针对具有环或线性阵列结构(或子结构)的网络提出贪心线性推移平衡算法。分析与实验表明，对一般的互连网络结构，算法的平衡步数都不高于处理器节点的数目 n。

2 贪心线性推移平衡算法

贪心线性推移平衡算法是针对具有环或线性阵列(子)结构的网络提出的，该算法主要针对具有哈密尔顿通路(圈)的图结构。而线性阵列、二维网状网、超立方体等常用的静态网都是具有哈密尔顿通路的图结构。

贪心线性推移平衡算法的基本思想是：把负载节点过重的那部分负载按线性或环的路径“贪心”地推移到下一邻居节点，循环推移直到整个系统负载平衡。“贪心”迁移负载就是每一个节点在每一步发送负载时都尽可能使发送后本节点当前负载恰好等于其目标负载，即每个节点在发送负载时，若节点的当前负载大于目标负载，则该节点留下的负载要恰好等于它的目标负载，而将其余的负载全部发送给下一节点；若节点的当前负载量小于目标负载量，则该节点不发送任何负载给下一个节点。

令网络的节点个数为 n，分别编号为 $0,1,\cdots,n-1$ (也用 $v_0,v_1,\cdots,v_{n-1}$ 表示)，其处理能力分别为：$c_0,c_1,\cdots,c_{n-1}$；初始负载分别为：$w_0,w_1,\cdots,w_{n-1}$；目标负载分别为：$\tilde{w}_0,\tilde{w}_1,\cdots,\tilde{w}_{n-1}$。则 $\tilde{w}_i = c_i \sum_{k=0}^{n-1} w_k \Big/ \sum_{k=0}^{n-1} c_k \quad (i=0,1,\cdots,n-1)$。

基金项目：福建省教育厅A类基金资助项目(JA09187)；闽江学院启动基金资助项目(YKQ08002)

作者简介：吴荣腾(1971－)，男，博士，主研方向：并行分布式计算，计算机网络

收稿日期：2010-04-07 **E-mail：**rongtengwu@yahoo.com.cn

又设每一步开始时各节点的当前负载量为 $\hat{w}_0,\hat{w}_1,\cdots,\hat{w}_{n-1}$，显然，第 1 步开始时有 $\hat{w}_i = w_i \ (i=0,1,\cdots,n-1)$。

第 1 步：任意节点 i 首先计算 $w_i-\tilde{w}_i$。当 $w_i-\tilde{w}_i>0$ 时，节点 i 发送负载 $w_i-\tilde{w}_i$ 到下一节点；否则，节点 i 不发送负载。然后接收前一节点发来的负载。

第 k 步：任意节点 i 首先计算 $\hat{w}_i-\tilde{w}_i$。当 $\hat{w}_i-\tilde{w}_i>0$ 时，节点 i 发送负载 $\hat{w}_i-\tilde{w}_i$ 到下一节点；否则，节点 i 不发送负载。然后接收前一节点发来的负载。

如此继续直到整个网络系统负载平衡。

由于节点每一步都以“贪心”的形式发送负载，即若当前负载足够，那么留下的负载一定恰好等于目标负载，因此，当任意节点 i 的信息能够历经其余 n-1 个节点时，网络负载必将达到平衡。

2.1 贪心线性推移平衡算法设计与性能分析

2.1.1 环与线性阵列的贪心线性推移平衡算法设计

令 s_i 为第 i 个节点要发送的负载量，r_i 为第 i 个节点接收到的负载量。环(或线性阵列)的贪心线性推移平衡过程采用如下策略，所有节点可以并发地执行以下各步骤：

(1)所有节点 v_i ($i=0,1,\cdots,n-1$)计算 $\hat{w}_i-\tilde{w}_i$ 的值以判定它们的负载是否超出其目标负载。若超出，即 $\hat{w}_i-\tilde{w}_i>0$ 时，则把超出的那部分负载作为即将发送的负载 s_i，即 $s_i=\hat{w}_i-\tilde{w}_i$；若没超出，即 $\hat{w}_i-\tilde{w}_i\le 0$ 时，则即将发送的负载 s_i =0。

(2)所有节点 i 发送其超出的那部分负载 s_i 给它的下一个节点 $v_{(i+1)\bmod n}$，并接收前一个节点 $v_{(i-1)\bmod n}$ 传来的负载 $s_{(i-1)\bmod n}$，即 $r_i=s_{(i-1)\bmod n}$。

(3)各节点更新其负载量(即 $\hat{w}_i=\hat{w}_i+r_i$)。在实际更新负载量过程中，如果本身原有的负载已经达到平衡，即已有 $\hat{w}_i=\tilde{w}_i$，则可直接把当前接收到的负载 r_i 作为下一轮要发送的负载 s_i 转发给 $v_{(i+1)\bmod n}$。

(4)重复(1)~(3)直到负载平衡(或基本平衡)。

重复(1)~(3)n-1 次后，任一节点的信息必可传达至其余的 n-1 个节点，所以，经 n-1 步后必有 $\boldsymbol{W}=\tilde{\boldsymbol{W}}$，即达到平衡。

2.1.2 环与线性阵列的贪心线性推移平衡算法性能分析

对于环而言，由于所有节点是并发地执行各步骤，而且对每一轮的循环，所有的节点都仅与它的邻居节点进行通信，因此至多经过 n-1 步并行执行邻居节点间的负载迁移可达到完全平衡。每一步所有节点迁移的总负载量为：$\sum_{i=0}^{n-1}\max(0,(\hat{w}_i-\tilde{w}_i))$，该值的大小与初始负载在各节点的分布有关。图 1 为 8 节点环的贪心线性推移平衡算法执行过程中的一轮循环推移情况。

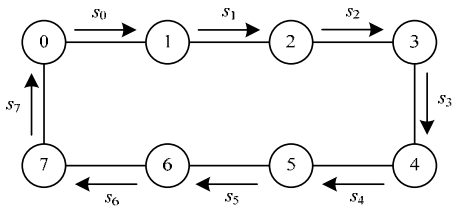


图 1　8 节点环的一轮循环推移

对于无环的线性阵列，由于链路是双向的，因此当负载从节点 n-1 传输到节点 0 时，可按与其他节点间通信方向相反的方向进行传输，经中间的 n-2 个节点到达节点 0。用直接穿透路由技术等按这种方式从节点 n-1 到节点 0 的传输时间开销与 2 个邻居节点的直接传输时间开销基本相同，并且不会形成线路竞争与死锁。因此，对于无环线性阵列而言，该算法的时间性能与环的情况基本一样，即经过 n-1 步并行执行邻居节点间的负载迁移可达到完全平衡。显然，上述算法适用于任何具有哈密尔顿通路的图。

2.2 二维网状网的贪心线性推移平衡算法

2.2.1 普通的贪心线性推移平衡算法

设二维网状网的节点个数为 $n=p\times p$，又令 s_i 为第 i 个节点要发送的负载量，r_i 为第 i 个节点接收到的负载量。二维网状显然存在哈密尔顿通路，因此，可以用一维线性阵列的贪心线性推移平衡算法平衡负载，最多只需 n-1 步并行执行邻居节点间的负载迁移，每一步所有节点总的负载迁移量为：$\sum_{i=0}^{n-1}\max(0,(\hat{w}_i-\tilde{w}_i))$，该值的大小与初始负载在各节点的分布有关。

2.2.2 分二阶段的贪心线性推移平衡算法设计

二维网状网每一行的 p 个节点都组成一个线性阵列，共有 p 个行线性阵列。据此可设计出每一步都分二阶段完成的贪心线性推移平衡算法。其中，任意第 k 步 2 个阶段的平衡策略如下：

第 1 阶段：

每个行线性阵列各自分别执行普通的贪心线性推移平衡算法，这个阶段每个行线性阵列都执行 p-1 次邻居节点间的直接负载迁移。

第 2 阶段：

(1)所有 $p\times p$ 个节点 v_i ($i=0,1,\cdots,n-1$)通过计算 $\hat{w}_i-\tilde{w}_i$ 的值判定它们的负载是否超出其目标负载以确定要发送的负载量：若超出即 $\hat{w}_i-\tilde{w}_i>0$ 时，则把超出的那部分负载作为即将发送的负载 s_i，即 $s_i=\hat{w}_i-\tilde{w}_i$；若没超出即 $\hat{w}_i-\tilde{w}_i\le 0$ 时，则即将发送的负载量为 s_i =0。

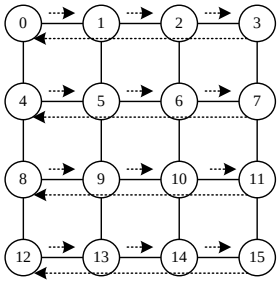
(2)所有节点 v_i ($i=0,1,\cdots,n-1$)发送其超出的那部分负载 s_i 给它下一行的对应节点 $v_{(i+p)\bmod n}$，并接收前一行对应节点 $v_{(i-p)\bmod n}$ 传来的负载 $s_{(i-p)\bmod n}$，即 $r_i=s_{(i-p)\bmod n}$。

(3)所有节点更新其负载量，即 $\hat{w}_i=\hat{w}_i+r_i$。这个阶段所有节点都只执行一次邻居节点间的负载迁移。

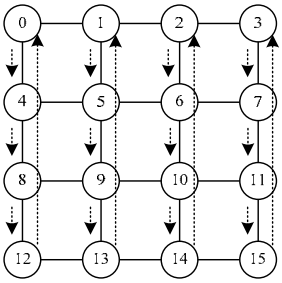
重复执行第 1 阶段与第 2 阶段直到负载平衡。

2.2.3 分二阶段的贪心线性推移平衡算法的性能分析

算法中每一步的 2 个阶段时间开销共为 p 次(第 1 阶段的 p-1 次和第 2 阶段的 1 次)所有节点并发执行邻居节点间的负载迁移。整个负载平衡过程总的时间开销由重复执行这 2 个阶段的次数决定。最多经 p 次重复可达到完全平衡，因为经 p 次重复后每个节点的信息都可以到达其余各节点。此时，总的通信时间开销为 $p\times p=n$ 次邻居节点间的负载迁移。但这比普通的贪心线性推移平衡算法的 n-1 次邻居节点间的负载迁移还多一次。这是因为第 p 次重复时第 1 阶段的迁移是多余的，事实上经 p-1 次重复后，再经第 p 次重复的第 1 阶段，任一节点的信息已经到达其余所有的节点，此时网络的负载已达到平衡。所以，可去掉第 p 次重复时第 2 阶段的那一步邻居节点间的负载迁移，即最大通信时间开销也是所有节点并发执行 n-1 次邻居节点间的负载迁移。图 2 为具有 16 个节点网状网的分二阶段的贪心线性推移平衡算法 2 个阶段的负载迁移情况。



(a)二维网状网贪心线性推移平衡算法第1阶段通信情况



(b)二维网状网贪心线性推移平衡算法第2阶段通信情况

图 2　二维网状网贪心线性推移平衡算法的通信情况

从以上分析可以看出，二维网状网分二阶段的贪心线性推移平衡算法与普通的贪心线性推移平衡算法在执行步数上相比没有任何的提高。但是，若负载平衡条件减弱，即当各节点的负载基本达到平衡时就认为系统负载已平衡而结束负载平衡过程，那么分二阶段的贪心线性推移平衡算法与普通的贪心线性推移平衡算法相比，时间性能有较大提高。这个结论将在实验部分作进一步讨论。

事实上，一般情况下负载平衡过程不一定要求达到完全平衡，而且实际应用程序中也几乎不可能达到完全平衡，因此，实际应用中只要 $\widehat{\boldsymbol{W}}$ 与 $\widetilde{\boldsymbol{W}}$ 基本相等(即系统负载基本平衡)，即可认为系统已达到负载平衡而停止平衡过程。

当前负载向量 $\widehat{\boldsymbol{W}}$ 与目标负载向量 $\widetilde{\boldsymbol{W}}$ 之间的基本相等(或接近程度)可以用它们之间的误差向量 $\boldsymbol{E}=\{e_0,e_1,\cdots,e_{n-1}\}$ 的范数 $\|\boldsymbol{E}\|$ 来表示，其中，$\boldsymbol{E}=\dfrac{\widehat{\boldsymbol{W}}-\widetilde{\boldsymbol{W}}}{\widetilde{\boldsymbol{W}}}$。

当 $\|\boldsymbol{E}\|$ 小于某个预先给定的阈值时可以认为负载已达到平衡而结束负载平衡过程。应用中可以按实际要求给定阈值，当 $\widehat{\boldsymbol{W}}$ 与 $\widetilde{\boldsymbol{W}}$ 的接近程度即 $\|\boldsymbol{E}\|$ 的值在该阈值之内时，可以认为系统已达负载平衡状态，此时平衡过程可以结束。实际应用时 $\|\boldsymbol{E}\|$ 常取 $\boldsymbol{E}$ 的 p -范数。取定范数后则可设定平衡条件(如 $\|\boldsymbol{E}\|\le 0.05$)对系统的平衡性进行检测和判定。

另一方面，与线性阵列一样，二维网状网贪心线性推移平衡算法中每一步所有节点迁移的总负载量为：$\sum_{i=0}^{n-1}\max(0,(\widehat{w}_i-\widetilde{w}_i))$，大小与初始负载在各节点的分布有关。

3　算法实验比较

为比较二维网状网的分二阶段的贪心线性推移平衡算法与普通的贪心线性推移平衡算法的性能，对 2 种算法进行模拟实验比较。首先在算法 1 与算法 2 的循环内添加语句：

```
if(系统负载已平衡或基本平衡)//  Ŵ 与 W̃ 相等或基本相等
    break;
```

用于控制负载基本平衡时跳出循环而结束程序。

取 $\|\boldsymbol{E}\|$ 的 ∞ 范数：$\|\boldsymbol{E}\|_\infty=\max_{i=0}^{n-1}|e_i|$ 作为平衡检测条件。

实验环境：各处理器的速度 c_i 为区间[1.00, 2.00]上的随机值。各节点的初始任务负载为一定范围的随机值。系统采用处理器数目为 $n=p\times p$ 的二维网状网结构。

实验 1　取 $\|\boldsymbol{E}\|=0.01$，应用分二阶段的平衡算法对 n 取 4×4、8×8、16×16、32×32、64×64 与 128×128 的二维网状网结构的系统进行实验。对各节点的初始负载 w_i 分别取区间[90,110]、[80,120]和[70,130]上的随机数时，分别运行这些系统 10 次并记录它们的平均执行步数(即平均执行轮数与处理器数目 n 的乘积)。图 3 给出了分二阶段的贪心线性推移平衡算法的平均执行步数与处理器数的关系。由图 3 可看出：(1)在平衡条件较弱时(即 $\|\boldsymbol{E}\|=0.01$)，分二阶段的二维网状网贪心推移算法能够比较明显地减少平衡负载的执行步数从而提高算法的性能。这可能是因为每一轮中各行形成的线性阵列内部已达平衡，而经过几轮循环后，若各行所形成的线性阵列间任务负载差别不大，则可提早形成基本平衡状态从而提高程序的时间性能。(2)初始负载分布状态对平衡时间性能有一定影响，若初始分布已较平衡，则算法更快达到平衡状态；但影响不是非常大，特别是 n 较小时，影响不是很明显。(3)分阶段线性平衡算法的平均执行步数与处理器的数目基本呈正比关系，因此，算法具有较好的可扩展性。

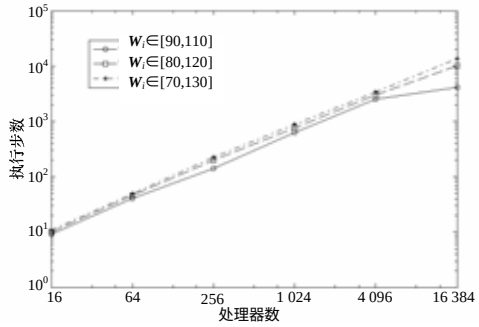


图 3　分二阶段的算法平均执行步数与处理器数的关系

实验 2　与实验 1 条件完全相同，对二维网状网使用普通的贪心线性推移平衡算法进行实验。即把二维网状网看成一维线性阵列而运行线性阵列的贪心推移算法以平衡负载。对 n 取 4×4、8×8、16×16、32×32、64×64 与 128×128 的网状网进行实验，其实验结果为：几乎每一次的运行步数都为 $n-1$。图 4 给出了应用普通的贪心线性推移平衡算法对二维网状网实验结果的平均执行步数与处理器数的关系。

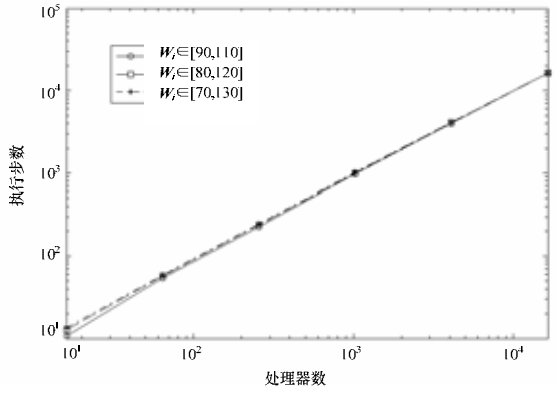


图 4　普通算法的平均执行步数与处理器数的关系

由图 4 可看出：(1)平均执行步数与处理器数目基本呈正比关系，因此，普通的贪心线性推移平衡算法也有较好的可

(下转第 105 页)