一个有效的动态负载平衡方法*

刘振英[1], 方滨兴[1], 胡铭曾[1], 张 毅[2]

[1](哈尔滨工业大学 计算机科学与工程系,黑龙江 哈尔滨 150001);

[2](哈尔滨理工大学 电气与电子工程系,黑龙江 哈尔滨 150040)

E-mail: zhenyingliu@hotmail.com

http://pact518.hit.edu.cn

摘要: 动态负载平衡问题是影响工作站网络并行计算性能的重要因素.首先分析出在负载平衡中产生额外开销的根本原因是负载的移动,进而定性地给出了每次移动负载的粒度公式.引入益处估计的方法,仅在有益的情况下进行负载平衡.另外还提出了一个动态负载平衡算法.最后,通过实验,将该算法的运行结果与其他人的负载平衡结果以及不作负载平衡的情况进行了对比.此负载平衡方法在工作站为空载以及不同的负载和应用问题的数据规模的情况下,都优于Siegell等人提出的方法.

关键词: 动态负载平衡;工作站网络;并行计算;数据并行

中图法分类号: TP393 **文献标识码**: A

工作站网络系统(network of workstations,简称NOWs)以其性能价格比方面的优势,成为当今的一个研究热点.但是,对于NOWs来说,1:50的负载不均衡现象也时有发生,这导致了系统常在低效的状态下运行.所以,解决负载平衡问题是提高NOWs性能的重要因素.负载平衡问题是一个经典的组合优化难题之一,其难度与Hamilton问题相当,是一个NP完全问题.负载平衡问题可分为静态负载平衡和动态负载平衡.人们通过对静态任务划分[1]的研究,间接地实现了静态负载平衡.而对于动态负载平衡,Hui等人[2]在理论方面提出了一个水动力学模型来抽象地描述负载平衡.文献[3]也给出了一个通用的模型,它综合了发送者驱动和接收者驱动这两种方法的优点.在专门针对数据并行程序方面,人们广泛地探讨了PVM任务的动态调度[4]和进程迁移[5],其中,由固定数目的多个循环迭代组成的任务是调度的基本单位.另外,Nicol等人[6]主要研究数据并行中不规则问题的动态调度.Zaki等人[7]研究了分布、集中、全局和局部的负载平衡方法,并且对这几种策略进行组合,通过试验总结了在不同的情况下哪种策略的效果最好.Siegell等人[8]研究了如何自动生成具有动态负载平衡特性的并行程序.本文与Zaki和Siegell的研究一样,都把循环迭代作为调度的基本单位,重点研究规则的数据并行程序的动态负载平衡问题.但是,与其不同的是,本文定性地探讨如何能让动态负载平衡带来效益,如何减小额外开销以及如何确定负载平衡的频率和粒度等关键问题.

1 动态负载平衡模型

当一个应用问题在一个NOWs上执行时,可以被模型化为一个无向图$G=(N,E)$.结点集N

* 收稿日期: 1999-07-29; 修改日期: 1999-12-08

基金项目: 国防科技预研基金资助项目(16.1.3)

作者简介: 刘振英(1972-),女,黑龙江哈尔滨人,博士生,主要研究领域为并行计算,并行编译;方滨兴(1960-),男,江西万年人,教授,博士生导师,主要研究领域为网络安全,Web技术;胡铭曾(1935-),男,江苏江阴人,教授,博士生导师,主要研究领域为高性能计算机系统结构,并行处理技术;张毅(1971-),男,贵州水城人,工程师,主要研究领域为网络计算,数据库技术.

代表任务的集合,这些任务合作完成该应用问题.边的集合 E 描述任务之间的通信.n_i 的计算速度为 s_i,它的剩余负载 l_i 反映了目前 n_i 的工作负载量.剩余执行时间 $t_i=\frac{l_i}{s_i}$ 是指 n_i 距离执行结束所剩的时间.根据每个任务的剩余执行时间,就可以知道该任务的负载状态.具有 $\max\{t_1,t_2,\ldots,t_n\}$ 和 $\min\{t_1,t_2,\ldots,t_n\}$ 的任务分别是最重载和最轻载的任务.为了描述方便,我们在下文中假定最轻载的任务为 n_i,最重载的任务为 n_j.每两个任务之间都存在一条通信链 $(n_a,n_b)\in E$,表示任务 n_a 和 n_b 能够交换负载信息,并且在它们之间移动负载.$c_{a,b}(l)$ 为两个任务间的通信时间,其中 l 代表通信量.由于在 message passing 编程模型中大多采用非阻塞式发送和阻塞式接收,所以本文中的通信时间特指数据的接收时间.因为非阻塞式的发送时间仅为微秒级,而阻塞式的接收时间为毫秒级.

对于该无向图 G,本文采用一个负载平衡算法来解决前文所提到的负载平衡的额外开销、效益和粒度问题,其解决方法如下:

(1) 减少额外开销

额外开销主要是由两个因素造成的:① 负载信息的采集.计算任务需要把自身的负载信息,包括 l_i 和 s_i,定期地发送给另外一个任务.最好是非计算任务,因为非计算任务不占用并行程序的执行时间,只是对总的执行时间产生很小的影响.而计算任务的数据发送时间只需几微秒就可以完成,所以采集负载信息所带来的额外开销很小.因此,我们可以采集尽量多的负载信息,使我们对负载的状态有一个较准确的估计,这也是我们下一步进行负载平衡的基础.但是,定时的时间间隔无论多小,总会有误差.例如,当采用 1 秒作为时间间隔时,可能产生的最大误差就是多估计或少估计 1 秒的计算量,即对 l_i 估计不准.② 移动负载.移动负载有时会导致计算任务接收大量的数据,因此,移动负载是产生额外开销的根本原因.为了减少负载的移动次数,我们规定:仅在某个任务执行完,即当 $l_i=0$ 时,才启动负载平衡.

(2) 益处估计

在移动负载时,有时移动得太少,会造成移动后的收益都不能抵消通信所产生的额外开销,得不偿失.所以,在 $l_i=0$ 时,需要权衡是否有必要移动负载.为此,我们定义了一个益处函数 f,当它大于 0 时,负载平衡就会给该并行程序带来性能上的提高;否则,只会降低性能.

设负载 l 要从 n_j 移动到 n_i 上执行,那么,益处函数

$$f=\frac{l}{s_j}-\frac{l}{s_i}-c_{i,j}(l).$$

使 $f>0$ 的条件是,l 必须满足

$$l>\frac{c_{i,j}(l)\times s_i\times s_j}{s_i-s_j}. \tag{1}$$

(3) 移动负载的粒度问题

因为并行程序的执行时间总是受限于执行速度最慢的任务.当最轻载的任务达到 $l_i=0$ 时,如果满足条件的话,显然要和最慢速的任务进行负载平衡.但是,移动多少负载为宜呢?如果负载移动的粒度太大,就会造成不必要的颠簸;如果移动得太少,还抵不上移动负载的额外开销,又不划算.我们采取的办法是,将最重载任务(n_j)和最轻载任务(n_i)重新分配负载,令 $t_j=t_i$,让它们在负载均衡后尽量同时结束.即

$$\frac{l_j-l}{s_j}\approx\frac{l}{s_i}. \tag{2}$$

其中 l 表示移动负载的大小.由式(2),我们可以计算出 l.

$$l-\left\lfloor\frac{s_i\times l_j}{s_i+s_j}\right\rfloor. \tag{3}$$

此外,要想移动负载量 l 并带来性能收益,还必须使 l 满足式(1).总的来说,只有当在式(3)中求得的 l 满足式(1)的条件时,我们才进行负载平衡,其中负载平衡的粒度是 $\left\lfloor\frac{s_i\times l_j}{s_i+s_j}\right\rfloor$.如前文所述,由于所采取的定时时间可能造成对 l_j 估计不准,也会进一步使求得的负载移动粒度有误差.如果该程序的执行时间与定时时间的比值足够大,就可以忽略掉负载移动的误差.

2 动态负载平衡算法

我们在一个数据并行程序中,构造了一个动态负载平衡算法.它包括一个调度服务器算法和一个计算任务算法.首先,调度服务器在每一个结点机上都派生出一个计算任务,并且把数据均匀地划分到每个计算任务.当一个计算任务先结束时,它就根据式(1)和式(3),判断是否进行负载平衡以及得出负载平衡的粒度.

• 调度服务器的算法

Step 1. 在每个结点机上都派生一个计算任务,并且把数据均匀地划分给各项任务;

Step 2. For (; ;)

接收来自任意计算任务 n_i 的消息;

(a) if (该消息是负载的状态信息),then 填入相应数据结构;

(b) if (l_i=0) then

1) 找出具有 $\max\{t_1,t_2,\ldots,t_n\}$ 的最重载任务 n_j;

2) 根据式(3)求 l;

3) if (l 满足式(1)) then 设置负载平衡标志;

把负载平衡标志、任务号 n_j 和 l 发送给计算任务 n_i;

软中断任务 n_j,并把任务号 n_i 和 l 发送给 n_j;

(c) 如果所有任务都计算完毕,则终止所有计算任务,退出;

End for

• 计算任务的算法(其他对负载平衡不重要的部分略)

Step 1. 开软中断和设置 1 秒的定时中断;

Step 2. For k=0 to up_bound−1 Step 1

(1) 计算 $A[k]$=...; /* 该应用程序的计算部分 */

(2) if (k==up_bound−1) then

发送负载信息给调度服务器,询问是否需要负载平衡;

接收调度服务器的负载平衡标志,任务号 n_j 和 l;

if (收到的负载平衡标志表明需要调度) then

(a) 从 n_j 接收数组 A 的 l 个数据;

(b) up_bound=l;

(c) 修改本机的数据分布;

(d) 重新开始循环;

End for;

Step 3. 发送"计算已完成"的信息给调度服务器;

• 计算任务的软中断服务程序

Step 1. 接收调度服务器发送来的消息 l;

Step 2. 给任务 n_i 发送 l 个数据;

Step 3. up_bound=up_bound−l;

Step 4. 修改本机的数据分布.

另外,计算任务的定时中断服务程序,定时向调度服务器发送自身的负载信息.

3 相关工作

前面我们提到,Siegell 等人[8].在实现数据并行程序的负载平衡时利用程序变换等方法产生适当的负载平衡点.例如,对于一个像矩阵乘这样的具有 3 层 for 循环嵌套的程序,他们认为最适合的负载平衡点应处于第 2 层的尾部,即 lbhook1,如下所示:

```
For( ) {
  For() {
    For() {
      ...
      lbhook2(); /* 负荷太大 */
    }
      lbhook1(); /* 适合 */
  }
  ...
  lbhook0(); /* 反应性太差 */
}
```

虽然文献[8]中指出了负载平衡点,但是对于一个大规模的计算而言,利用该点进行负载平衡还是太频繁了.如果不加以限制,在 lbhook1 点仍需平衡多次,这将引起很多不必要的通信开销,甚至颠簸.并且,文中不进行益处估计,有可能造成某些负载平衡,这是得不偿失的.本文与文献[8]所述方法不同的是,本文的算法采用益处估计,负载平衡点是 lbhook0,计算负载平衡粒度使最快速和最慢速的任务得到均衡.本文将在下面给出两者的实验结果比较.

4 实验结果

我们的实验是用 10M 以太网连接起来的 3 台 IBM PowerPC 工作站,主频都是 133Hz,并行编程环境是 PVM,并且采用典型的并行矩阵乘程序和求质数作为例子,在程序运行初始时都采用均匀划分.这两个程序的共同点是它们的循环迭代都是独立的,不同点是前者的每个迭代的执行时间都相同,而后者的每个迭代的执行时间不同.其中,任务 n_i 的 s_i 为每秒执行循环迭代的个数,l_i 为剩余循环迭代个数.我们还对通信时间 $C_{i,j}$ 进行了统计,网络空载时为 2～3ms.为了简单起见,无须在程序中动态地测试通信时间以增加额外开销,我们给 $C_{i,j}$ 赋常数值 10ms.采集负载的间隔一律采用 1s.我们在以下几个方面进行测试:

(1) 空载时的执行时间

首先,在 3 台机器都空载时,分别执行有负载平衡和没有负载平衡的并行矩阵乘程序.从表 1 中我们可以看出,无论矩阵的规模如何,两者的执行时间都非常接近.这是因为,并行程序的执行时间是由最慢速任务的执行时间所决定的,而我们给最慢速任务所增加的额外开销只是由它给调度服务器发送几次负载信息,并且在结束时给调度服务器发送一次消息.我们测试了消息发送语句执行一次的时间,平均为 2～3μs.这些微秒级的额外开销与总的秒级执行时间相比,显得微不足道.因此,在所有结点机都空载时,有时会发生负载平衡的矩阵乘程序甚至比不进行负载平衡执行得还

慢的情况.

Table 1 Execution time of parallel matrix multiplication without extra workload

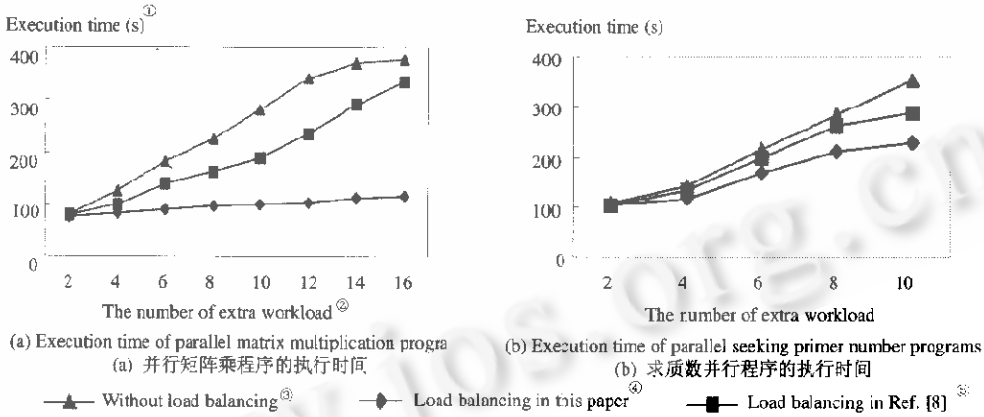
表 1 并行矩阵乘程序在空载时的执行时间

The scale of matrixes①	300×300	600×600	900×900	1200×1200
The execution time without load balancing (s)②	2.69	38.66	149.71	373.28
The execution time with load balancing (s)③	2.79	39.36	149.85	378.55

①矩阵规模,②不进行负载平衡时的并行执行时间(秒),③进行负载平衡时的并行执行时间(秒).

(2) 负载递增时的执行时间

我们还测试了采用负载平衡算法的 600×600 的矩阵乘以及求 4×10^6 以内的质数的并行程序在不同负载情况下的执行时间,如图 1 所示.首先,我们在 3 个结点机都为空载时测试该程序的执行时间;然后,我们仅在一台结点上依次增加 2,4,... 个额外负载再测试程序的执行时间,其中每一个"额外负载"都是同一个死循环的浮点运算程序.如图 1 所示,随着额外负载的增加,本文采用的负载平衡方法比不进行负载平衡的并行程序执行时间增长要缓慢得多.而文献[8]的方法,由于速度最慢的任务运行到负载平衡点,也需要与调度服务器进行通信,尤其是等待调度服务器的回信,增加了负载平衡的时间.对于求质数的并行程序,本文根据每秒平均执行循环迭代的个数来判断需要移动多少负载.由于每个迭代的执行时间不同,仅得对不同负载的移动所产生的负载效果不甚相同,从而加大了任务结束时间的差距,或者使负载平衡次数增加.因此,采用本文的负载平衡算法后的求质数并行程序,其性能的提高略逊于并行矩阵乘程序.



①执行时间(秒),②额外负载的个数,③无负载平衡,④本文的负载平衡,⑤文献[8]的负载平衡.

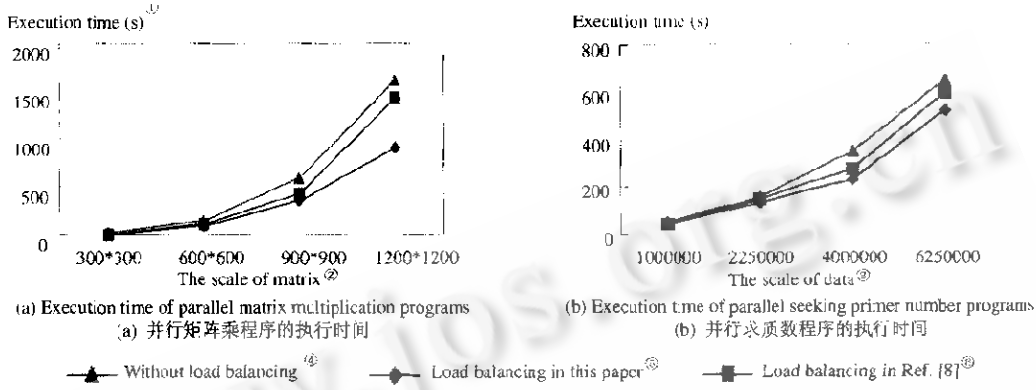
Fig. 1 The comparison of execution time for parallel programs in different workload status

图 1 并行程序在不同负载情况下的执行时间对比

(3) 数据规模增加时的执行时间

我们还在两个结点机空载,第 3 台结点机上增加了如上面(2)中所述的几个额外负载,并且在保持该环境不变的情况下,测试了在不同的数据规模下的并行矩阵乘和求质数程序的执行时间,如图 2 所示.从图 2 中我们可以看出,随着数据规模的增加,本文采用的负载平衡算法的性能比无负载平衡的算法的性能要好得多.需要说明的是,当执行 300×300 的矩阵乘时,虽然它的执行时间与定时采集负载的时间间隔的比值不大(在图 2 中比值约为 7),但是,负载平衡的效果仍然要好于不负载平衡.这是因为,即使负载平衡的粒度有误差,由于移动粒度必须满足益处函数,负载平衡仍然会带来益处.随着矩阵规模的增加,即使定时的时间间隔造成少移动 1 秒的计算量,也会使执行速

度最快与最慢的结点的计算完成时间相差1秒,但这与总的几百秒的执行时间相比,是可以忽略的.另外,由于我们采用的算法降低了额外开销,所以优于文献[8]的方法.



(a) Execution time of parallel matrix multiplication programs
(a) 并行矩阵乘程序的执行时间

(b) Execution time of parallel seeking primer number programs
(b) 并行求质数程序的执行时间

①执行时间(秒),②矩阵规模,③数据规模,④无负载平衡,⑤本文的负载平衡,⑥文献[8]的负载平衡.

Fig. 2 The comparison of execution time for parallel programs in the case of extra changeless workload

图2 在额外负载不变情况下的并行程序执行时间对比

5 结束语

本文得出了一个更适用于NOWs并行应用问题的动态负载平衡模型.该模型只在有益处的情况下,才进行负载平衡.而且,本文提出的算法具有负载平衡次数少、额外开销少的特点.从实验结果可以看出,采用本文所提出的算法,可以取得良好的效果.虽然本文仅给出了并行矩阵乘和求质数程序的例子,但是由于本文的算法调度在最外层循环体尾部,所以对Doall类型的数据并行应用问题都很适合.尤其是根据Wolf提出的单模变换[9]等循环变换方法,某些嵌套式的Do循环可以转化成Doall型的循环.另外,本文所采用的计算负载和网络负载的值,分别用的是过去一段时间的旧值和大概估计值,今后将考虑用预测的方法来代替.此外,该算法简洁,对编译器而言较易于实现.下一步的工作是将这个算法加入到我们的并行C++编译器中,使该并行C++编译器能够自动支持数据并行程序的动态负载平衡.

References:

[1] Wen, Yu-hong, Wang, Ding-xing, Zheng, Wei-min. An optimal processor allocation algorithm for heterogeneous clusters. Chinese Journal of Computers, 1996,19(3):161~167 (in Chinese).

[2] Hui, Chi-Chung, Chanson, S. T. Theoretical analysis of heterogeneous dynamic load-balancing problem using a hydrodynamic approach. Journal of Parallel and Distributed Computing, 1997,43(2):139~146.

[3] Chen, Hua-ping, Ji, Yong-chang, Chen, Guo-liang. A universal model of distributed dynamic load balancing. Journal of Software, 1998,9(1):25~29 (in Chinese).

[4] Fu, Qiang, Zheng, Wei-min. A dynamic task scheduling method in cluster of workstations. Journal of Software, 1999,10(1):19~23 (in Chinese).

[5] Ju, Jiu-bin, Wei, Xiao-hui, Xu, Gao-chao, *et al.* DPVM: an enhanced PVM supporting task migration and queuing. Chinese Journal of Computers, 1997,20(10):872~877 (in Chinese).

[6] Nicol, D. M., Reynolds, P. F. Jr. Optimal dynamic remapping of data parallel computations. IEEE Transactions on

Computers, 1990,39(2):206~219.

[7] Zaki, M.J., Li, W., Parthasarathy, S. Customized dynamic load balancing for a network of workstations. Journal of Parallel and Distributed Computing, 1997,43(2):156~162.

[8] Siegell, B.S. Automatic generation of parallel programs with dynamic load balancing [Ph.D. Thesis]. Pittsburgh: Carnegie Mellon University, 1995.

[9] Wolf, M.E., Lam, M.S. A loop transformation theory and an algorithm to maximize parallelism. IEEE Transactions on Parallel and Distributed Systems, 1999,2(4):452~471.

附中文参考文献:

[1] 温钰洪,王鼎兴,郑纬民.异构机群系统中的最优处理机分配算法.计算机学报,1996,19(3),161~167.

[3] 陈华平,许永昶,陈国良.分布式动态负载平衡调度的一个通用模型.软件学报,1998,9(1):25~29.

[4] 傅强,郑纬民.一种适用于机群系统的动态任务调度方法.软件学报,1999,10(1):19~23.

[5] 鞠九滨,魏晓辉,徐高潮,等.DPVM:支持任务迁移和排队的PVM.计算机学报,1997,20(10):872~877.

An Effective Dynamic Load Balancing Method*

LIU Zhen-ying[1], FANG Bin-xing[1], HU Ming-zeng[1], ZHANG Yi[2]

[1](*Department of Computer Science and Engineering, Harbin Institute of Technology, Harbin* 150001, *China*);

[2](*Department of Electrical and Electronic Engineering, Harbin University of Science and Technology, Harbin* 150040, *China*)

E-mail: zhenyingliu@hotmail.com

http://pact518.hit.edu.cn

Abstract: Dynamic load balancing is an important factor to determine the performance of a NOWs (network of workstations). First, it is analyzed that the load movement is the reason for overhead. Furthermore, a formula of data movement granularity is proposed. Besides, a method to evaluate the benefits of load balancing is presented in order to balance only when it is profitable. Meanwhile this paper puts forward a load balancing algorithm. Finally, the execution result of the proposed method is compared with that of others and that without dynamic load balancing by experiments. The load balancing method in this paper outperforms that of Siegell's in the cases of idle workloads, different workload and data scales of applications.

Key words: dynamic load balancing; network of workstations; parallel computing; data parallelism

* Received July 29, 1999; accepted December 8, 1999

Supported by the Preliminary Research Fund from the Department of Defence under Grant No. 16.1.3