第33卷 第20期 计 算 机 工 程 2007年10月
Vol.33 No.20 Computer Engineering October 2007
·博士论文· 文章编号：1000—3428(2007)20—0018—03 文献标识码：A 中图分类号：TP311

基于快速分配机制的动态负载平衡算法

刘 滨[1,2]，石 峰[1]，高玉金[1]
(1. 北京理工大学计算机科学技术学院，北京 100081；2. 河北科技大学经济管理学院，石家庄 050018)

摘 要：针对同构型多处理机系统中的动态负载平衡问题，制定了若干规则，对搜索轻载节点的过程进行约束，提出一种能快速分配多余负载的、分布式控制、发送者驱动的动态负载平衡算法，实验证明该算法在处理计算密集型任务时，具有较好的有效性。
关键词：动态负载平衡；规则；消息；多处理机

Dynamic Load Balancing Algorithm Based on Fast Distributing Mechanism

LIU Bin[1,2], SHI Feng[1], GAO Yu-jin[1]
(1. School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081;
2. College of Economics and Management, Hebei University of Science and Technology, Shijiazhuang 050018)

【Abstract】 To realize dynamic load balance in homogeneous multiprocessor system, several rules are proposed and used to restrict the process of searching lightly loaded processor. A dynamic load balancing algorithm, which can fast distribute redundant loads, is distributed control and sender initiated, is proposed. Experiments prove the algorithm's validity in dealing great scale compute intensive task.
【Key words】 dynamic load balancing; rule; message; multiprocessor

目前，一些采用发送者(重载节点)驱动方式的动态负载平衡算法，每次运行只选择一个符合条件(最轻载[1]、距离发送者最近[2]或其他约束条件[3])的轻载节点作为负载迁移的目标节点。负载迁移后，若重载节点还未转为适载，则再次运行算法，直到实现负载平衡。本文提出的动态负载平衡算法致力于在尽量少的运行次数中，将发送者的多余负载快速分配到多个轻载节点，避免多次运行算法造成的时空开销。该算法面向同构型多处理机系统，采用发送者驱动、分布式控制方式，其有效性通过实验得以验证。

1 动态负载平衡算法

1.1 负载指标

同构型多处理机系统中节点 vi 的处理能力，即单位时间内 CPU 能处理的最多进程数用 C_i 描述，负载阈值用 A_i 描述(通常取 C_i 的 10%)，当前 CPU 运行队列长度用 Q_i 描述，则节点 vi 的负载状态 S 可以根据式(1)判定：

$$S(Q_i)=\begin{cases} heavy & Q_i > C_i + A_i \\ normal & C_i - A_i \geq Q_i \leq C_i + A_i \\ light & Q_i < C_i - A_i \end{cases} \quad (1)$$

其中，*heavy*表示vi处于重载状态，需要迁出的进程数 $EPN(vi)=Q_i-C_i-A_i$；*no*RM*al*表示 vi 处于适载状态，既不发送进程到其他节点，也不接受迁移进程；*light*表示vi处于轻载状态，可以接受的进程数 $IPN(vi)=C_i+A_i-Q_i$。

1.2 消息

负载平衡消息的表示形式：消息名(参数 1,…,参数 n)；消息和参数间的隶属用"."表示。一次成功的负载平衡过程见图 1。各类消息如下：

(1)RM(request message)请求消息。参数：消息的始发节点 *HV*(heavy vertex，重载节点)和 *EPN*(emigrated process number, 外迁进程数)。

(2)AM(answer message)应答消息。参数：目标节点 *HV*、始发节点 *LV*(light vertex，轻载节点)和 *APN*(accept process number，可迁入进程数)。

(3)CM(confirm message)确认消息。参数：始发节点 *HV*、目标节点 *LV*、布尔变量 *AOR*(accept or reject)和 *DEPN*(decided emigrated process number，决定外迁的进程数)。*AOR* 表示是否外迁进程到 *LV*。

(4)LMM(load migration message)负载迁移消息。参数：始发节点 *HV*、目标节点 *LV*、布尔变量 *FON*(final or not，是否是最后一个外迁进程)和 *PI*(process info，外迁进程的上下文信息)。

(5)DRM(dealing result message)处理结果消息。参数：目标节点 *DV*(destination vertex，迁出进程后保持重载或转为适载的节点)、始发节点 *BV*(beginning vertex，迁入进程后保持轻载或转为适载的节点)、布尔变量 *FON*(final or not，是否最后一个进程)和 *R*(result，外迁进程的处理结果)。

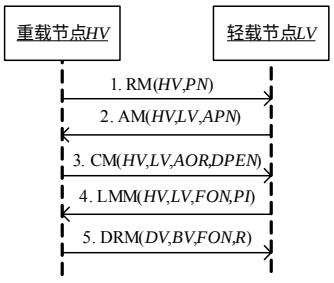

图 1 负载平衡中的消息传递过程

1.3 处理节点维护的信息

系统中任一节点 vi 需要维护如表 1 所示的信息。

作者简介：刘 滨(1975－)，男，讲师、博士研究生，主研方向：嵌入式计算，并行处理体系结构；石 峰，教授、博士生导师；高玉金，讲师、博士
收稿日期：2006-10-20 **E-mail：**bit_runner@163.com

表 1　节点 *vi* 维护的各类信息

名称	取值范围/模式	用途
变量 *rmtv* (RM Transfer Vertex)	null 或 *vi* 的邻接点	记录向 *vi* 发送 RM 并被其处理的邻节点
变量 *LEPN*(*Left EPN*)	-1(初始化值)≤LEPN≤EPN	HV 上的部分多余进程可能处于已经被分配给 LV，但还没被实际迁移的状态。*LEPN* 记录 HV 还未分配出的多余进程数，当 $LEPN=0$ 时，可以认为 HV 是“伪重载”
变量 *LIPN*(*Left IPN*)	-1(初始化值)≤LIPN≤IPN	LV 可能处于已经分派若干进程“席位”给 HV，但进程还未被迁移过来的状态。*LIPN* 记录 LV 上的剩余进程“席位”。当 $LIPN=0$ 时，可以认为 LV 是“伪轻载”
邻接点路由表 NRT (Neighbor Route Table)	(HV, 前节点 PV-Prior Vertex, 目标节点 DV-Destination Vertex, 后节点 NxV-Next Vertex)	在负载迁移和处理结果回迁两个过程中，辅助 *vi* 确定信息转发的下一个节点。所谓“前”、“后”节点，是以 *vi* 的邻接点中距离 HV 的远近划分的。*vi* 收到 AM 消息后，往表里添加相关记录；收到 CM 或 DRM 消息后删除相关记录

1.4　算法思想和规则

HV 向所有邻接点发送 RM，形成多条搜索 *LV* 的路径。搜索路径不一定止步于第 1 个 *LV*；对于回馈 AM 的 *LV*，立即分配多余负载，从而保证 *HV* 能快速分配负载，实现负载平衡。为避免某些节点被多条搜索路径覆盖而导致信息的误传或冗余，制定规则 1～规则 3 对搜索过程进行约束。

规则 1 *LV* 在响应了源自某 *HV* 的 RM 后，在收到源自该 *HV* 的 CM 前，不再响应 RM。

说明 (1)对于源自同一 *HV* 的多个 RM，多次响应无意义；(2)如图 2(a)所示，若 *R*1 先于 *R*3 到达 *v*2，则 *v*2 在收到源自 *v*1 的 CM 后，才可以响应 *R*3。因为收到 RM 的 *LV*，仅是 RM.*HV* 外迁负载的候选节点，而是否被 RM.*HV* 选定，以及若被选定，需要接受多少进程，只能从 CM 中得到确认。因此在收到 CM 之前，若响应源自其他 *HV* 的 RM，则反馈的 AM.*IPN* 信息不准确。

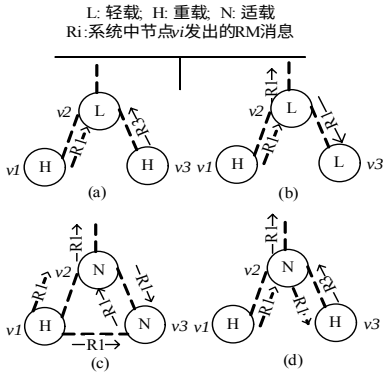

图 2　负载平衡中 RM 的传递情况

规则 2 *LV* 在自身无法满足 RM 时，转发该消息。

说明 如图 2(b)所示，若 $EPN(v1)\le LIPN(v2)$，则 RM 无须被转发；否则，修正后的 RM(即 $RM.APN=RM.APN-LIPN(v2)$)被转发给 *v*3 以及 *v*2 的其他邻接点。

规则 3 对于 RM，*HV* 不转发；NV(normal vertex, 适载节点)视情况转发。

说明 *HV* 自身还须外迁进程，因此不转发源自其他 *HV* 的 RM；而对于 *NV*：(1)对于源自同一 *HV* 的 RM，转发一次即可，如图 2(c)所示；(2)对于源自不同 *HV* 的 RM，如图 2(d)所示，一旦转发了某个 RM(即 $rmtv\ne\text{null}$)，在收到源自该 RM.*HV* 的 CM 前，不再响应其他 RM。因为 *NV* 转发了源自某 *HV* 的 RM 后，其延展出的所有搜索路径上的、没响应过其他 *HV* 的 *LV*，在收到源自该 RM.*HV* 的 CM 之前，都是该 RM.*HV* 外迁负载的候选节点(见规则 1)。

1.5　算法描述

HV 要外迁负载时，运行算法 1。

算法 1 搜索可接受外迁负载的 *LV*

输入 *HV*；$T_{\max}$：等待 *LV* 反馈 AM 的时间上限(设为 HV_i 发出的消息到系统中最远节点的时间的 2 倍)

输出 若干 *LV*

Step1 据 1.1 节，求出 $EPN(vi)=Q_i-C_i-A_i$，HV_i 向所有邻接点发送 RM；

Step2 若在 $T_{\max}$ 内收到 AM，且 $AM.HV=HV_i$，即有 *LV* 响应 RM，调用算法 2，多余负载分配完后转 Step3；否则说明没有合适的 *LV*，转 Step1；

Step3 结束。

对外来消息，任一节点 *vi* 执行算法 2。

算法 2 处理消息

输入 *vi*；MS 为来自邻节点 *vj* 的消息；*LIPN* (见表 1)；*LEPN* (见表 1)；

输出 消息处理策略

Step1 若 *vi* 为轻载，转 Step2；为适载，转 Step3；为重载，转 Step4；

Step2 MS 为 RM，转 Step2.1；为 AM，转 Step2.2；为 CM，转 Step2.3；为 LMM，转 2.4；为 DRM，转 2.5；

Step2.1 若 *LIPN* 为 0，转 Step5；为-1，令 $LIPN=IPN$。若 RM*tv* 为 null，即 *vi* 能响应 RM，令 $rmtv=vj$，反馈 AM(RM.*HV*, *vi* , *LIPN*)给 *vj*，在 NRT 中增加记录(RM.*HV*, RM*tv*, *vi*, *vi*)。若 $LIPN\le RM.EPN$，且 *vi* 还有除 *vj* 外的其他邻接点，调整 $RM.EPN=RM.EPN-LIPN$，转发 RM，转 Step5；

Step2.2 收到 AM，说明 *vi* 先前向 *vj* 发送过 RM，在 NRT 中检索 *HV* 值等于 AM.*HV* 的记录 *r*，若 *r* 存在(即已经有从 *vi* 到 AM.*HV* 的迁移路径)，在 NRT 中增加新记录(AM.BN, *R*.PN, AM.DN, *vj*)，并向 *R*.PN 转发该消息；若 *r* 不存在(即从 *vi* 到 AM.*HV* 的迁移路径已经被取消)，则在 NRT 中增加记录(AM.*HV*, *rmtv* , AM.*LV*, *vi*)，并向 *rmtv* 的节点转发该消息，转 Step5；

Step2.3 令 $rmtv=\text{null}$，即允许 *vi* 处理源自其他 *HV* 的 RM。若 CM.*AOR*=False, 在 NRT 中找到并删除 *HV* 分量值为 CM.*HV* 且 *DV* 分量值为 CM.*LV* 的记录 *r*。若 $vi\ne CM.LV$ (即 *vi* 不是 CM 的目标节点)，则在删除 *r* 前，向 *r*.*NV* 转发该消息。若 $CM.LV=vi$ (即 *vi* 是 CM 的目标节点)并且 CM.*AOR* 值为 True，则令 $LIPN=LIPN-CM.DEPN$，转 Step5；

Step2.4 从 LMM 中获取迁移进程的信息，重建该进程运行现场，进行处理。若 LMM.*FON*=True，令 $LIPN=-1$，转 Step5；

Step2.5 接收到 DRM，说明 *vi* 先前向 *vj* 转发过迁自 DRM.*DV* 的负载，在 NRT 中检索 *HV* 分量值等于 DRM.*DV* 的记录 *r*，将 DRM 转发给节点 *r*.*PV*。若 DRM.*FON* 为 True，则删除 *r*，转 Step5；

Step3 *MS* 为 RM，转 Step3.1；为 AM，转 Step3.2；为 CM，转 Step3.3；为 LMM，转 Step3.4；为 DRM，转 Step3.5；

Step3.1 若 *rmtv* 为 null 且 *vi* 还有除 *vj* 外的其他邻接点，则转发该消息，转 Step5；

Step3.2 同 Step2.2；

Step3.3 令 *rmtv* = null，若 CM.*AOR*=False，在 NRT 中找到 *HV* 分量值为 CM.*HV* 且 *DV* 分量值为 CM.*DV* 的记录 *r*，向 *r*.*NV* 转发该消息，删除 *r*，转 Step5；

Step3.4 在 NRT 中找到 *DV* 分量值等于 LMM.*LV* 的记录 *r*，转发该消息到 *r*.*NV*，转 Step5；

Step3.5 接收到 DRM，若 DRM.*HV*= *vi* (说明 *vi* 之前外迁过部分进程，并在迁出进程后转为适载状态)，取出 DRM.*R* 做本地处理。若 DRM.*HV* ≠ *vi* (说明 *vi* 先前向 *vj* 转发过迁自 DRM.*HV* 的负载)，在 *NRT* 中检索 *HV* 分量值等于 DRM.*HV* 的记录 *r*，将 DRM 转发给节点 *r*.*PV*。若 DRM.*FON* 为 True，则删除 *r*；

Step4 若 *MS*=AM，转 Step4.1；若 *MS*= DRM，转 Step4.2；

Step4.1 若 *LEPN* 为 0，向 AM.*LV* 发送 CM(*vi* , AM.*LV*, False, 0)，转 Step5；若 *LEPN* 为-1，令 *LEPN* = *EPN* 。若 *LEPN* > 0 ，则当 AM.*IPN* ≥ *LEPN* ，令 *LEPN* = 0 ；若 AM.*IPN* < *LEPN* ，则令 *LEPN* = *LEPN* – AM.*IPN* 。若 *vi* 没有正向其他节点外迁进程，则在 NRT 中增加记录 *r*(*vi* , *vi* , AM.*LV* , AM.*IPN*)，向 *vj* 发送 CM(*vi* , AM.*LV* , True, *LEPN*)，并封装外迁进程的相关信息到 LMM 中，开始负载迁移，转 Step5。

Step4.2 收到 DRM(说明 *vi* 之前外迁过部分进程，并在迁出进程后仍保持重载)，取出 DRM.*R* 做本地处理。在 NRT 中检索 *DV* 分量值等于 DRM.*LV* 的记录 *r*，若 DRM.*FON* 为 True，则删除 *r*，若 NRT 成为空，令 *LEPN* = –1，转 Step5。

Step5 结束。

2 实验结果与分析

2.1 实验环境

系统规模——6 台主机。并行编程环境——PVM。主机配置——操作系统：Windows 2000；CPU：Pentium4 1.6GHz；内存：256MB；网卡：100Mb/s。系统类型——非均匀存储访问模式(nonuniform memory access, NUMA)同构型集群系统。

2.2 对比算法与实验

实验中采用以下 4 种算法：(1)LOWEST[3]：分布式控制，*HV*从符合约束条件的*LV*集中选择最轻载节点进行负载迁移；(2)DASUD[4]：分布式控制，*HV*将搜索*LV*的范围首次限定为邻接点集，若无符合条件者，则将搜索范围向外扩张一层，依次迭代，直至找到符合条件的*LV*；(3)CCHSPMD[5]：集中式控制、适用于计算密集型的全局平衡算法；(4)RMTLB：本文提出的算法。

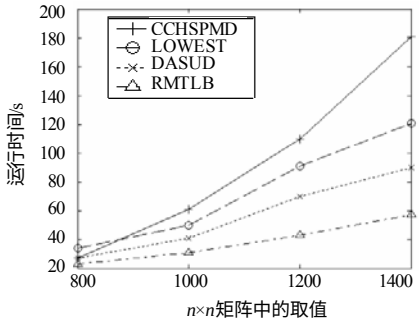

图 3 并行矩阵相乘中各算法的时间曲线

矩阵运算是大规模计算中具有代表性的一类，在科研和工程中都有重要意义。为此，进行了并行矩阵相乘实验，结果见图 3；求素数程序的循环迭代相互独立，并且每次迭代的执行时间也不相同，能够较好地模拟并行处理系统中的负载不平衡现象。为此，进行了大规模数据集中求素数的实验，结果见图 4。

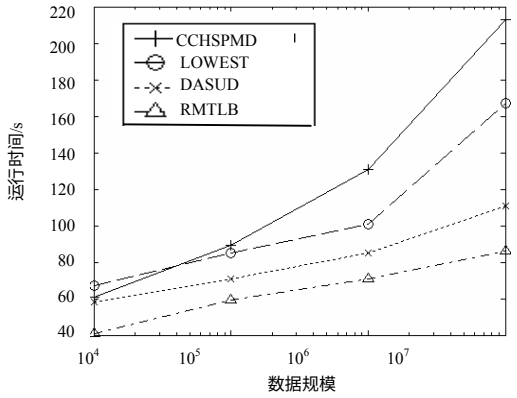

图 4 各算法求素数的时间曲线

2.3 实验结果分析

(1)计算规模较小时，算法 3 的时间性能均优于算法 1，次于算法 2、算法 4；计算规模增大时，算法 1、算法 2、算法 4 的时间性能均优于算法 3。这证明分布式控制方式优于集中式控制方式。

(2)算法 1 将负载迁移到最轻载节点，尽力减轻对目标节点负载状态的影响，却忽视了迁移距离。算法 2、算法 4 以 *HV* 为中心，逐步向远处“辐射”负载迁移请求消息，最先搜索到的 *LV* 在所有 *LV* 中与 *HV* 最近。因此，算法 2、算法 4 的时间性能均优于算法 1。

(3)算法 1、算法 2 每次运行只选择一个 *LV* 进行负载迁移，若该节点不能接受全部过载进程，则需要再次运行算法。算法 4 赋予搜索过程一定的延续性。若 *LV* 无法接受 RM 中的负载，则继续传递修正后的 RM，保证能在算法尽量少的执行次数中，尽快分配完 *HV* 上的多余进程。因此算法 4 的时间性能优于算法 1、算法 2。

3 结论

针对同构型多处理机系统中的负载失衡问题，本文提出的算法能够缩短负载迁移的通信时延，并能在尽量少的运行次数中快速分配完重载节点上的多余负载，提高了负载平衡的时效性。算法的有效性和实用性在实验中得以验证。

参考文献

1 Eager D, Lazowska E, Zahorjan J. DynAMic Load Sharing in Homogeneous Distributed Systems[J]. IEEE Trans. on Software Eng., 1986, 12(5): 662-675.

2 Cortes A, Ripoll A, Senar M A, et al. On the PerfoRMance of Nearest-neighbors Load Balancing Algorithms in Parallel Systems[C]//Proceedings of the 7th Euromicro Workshop on Parallel and Distributed Processing. 1999: 170-177.

3 Lee B. DynAMic Load Balancing in a Message Passing Virtual Parallel Machine[R]. Division of Computer Engineering, School of Applied Science, Nanyang Technological University, Singapore, 1995.

4 Kunz T. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme[J]. IEEE Trans. on Software Eng., 1991, 17(7): 725-730.

5 Le Mair M W, Reeves A P. A General DynAMic Load Balancing Model for Parallel Computers[R]. Cornell School of Electrical Engineering, Tech. Rep: EE-CEG-89-1, 1989.