协同调度算法Dasic的性能评价*

周笑波 陆桑璐 谢 立

(南京大学计算机软件新技术国家重点实验室 南京 210093)
(南京大学计算机科学与技术系 南京 210093)
E-mail: zbo@dislab.nju.edu.cn

摘要 对一个动态自适应的NOW(network of workstations)协同调度算法Dasic算法进行了性能评估.通过对Dasic算法和典型的NOW协同调度算法MAX算法、Grab算法的性能模拟,在响应时间和系统流量两个主要性能指标上进行分析和比较,从而验证了在较大规模的NOW系统中,动态自适应对协同调度的性能有较大的影响.

关键词 工作站网络,协同调度,动态自适应.
中图法分类号 TP316

协同调度[1]根据并行应用的各进程间需要通信交互的特性,尽可能让同一组进程在不同处理机上同时被调度运行,以保证进程在需要通信时总能处于调度状态,避免进程颠簸的发生,并减少环境转换(Context Swap)带来的开销.

由于工作站性能/价格比的优势,以及如ATM等高速通信网络的不断完善,当前工作站网络NOW因其优越的性能、良好的可扩展性和并行I/O等特点,已成为国际上并行及分布式计算领域近年来十分活跃的研究课题.协同调度同时也成为一个在NOW(network of workstations)环境中高效地利用整个网络的系统资源的方法.

协同调度算法的研究起源于1982年Ousterhout为Medusa操作系统[2]所做的工作.他当时在MPP环境里提出了3种算法,即Matrix算法、Continous算法和Undivided算法.目前NOW环境下已实现的协同调度算法很多,如Max算法、Grab算法、Static算法和Freelist算法等.尽管各自的任务分配策略不同,有的为尽量提高单个任务的并行度而以一段组等待时间为代价[3],有的则尽量避免组等待时间,但是这些算法大多不考虑系统中可利用处理机数的动态变化,并且限制进程的动态迁移.[4]

针对动态变化的NOW环境,充分考虑到工作站共享的特点——高度自治性,我们曾提出一个动态自适应的进程迁移和协同调度模型DASIC.[5]它一方面对并行或大型应用项目的交互进程进行协同调度,以避免进程颠簸的发生,另一方面根据NOW环境的动态变化,自适应地进行伸展、收缩和负载平衡调整.[6]为了进一步探讨在NOW环境下影响协同调度性能的因素,以开发高效的协同调度算法,本文提出一个基于NOW的协同调度算法的性能评估模型.

1 NOW协同调度算法

1.1 基本概念

(1) 任务的分类

按照任务计算目标的不同进行分类,NOW环境中工作站上运行的任务可分为两类:(1) 工作站节点本身的任务,如系统内务操作等,称为局部计算任务;(2) 分布在各工作站上的并行程序的计算任务,称为全局计算任务.

由于调度资源一般仅考虑到CPU的利用率,所以,NOW协同调度算法比较适合计算强度较大的计算密集型任务.它不干预局部计算任务,主要是根据全局调度策略,对全局计算任务进行排队、挂起、启动、撤销等控制操作,以确保并行系统达到较高的运行效率.

(2) 并行任务可申请的最大工作站数

在一个协同调度环境中,一个全局任务只能申请低于一个上限的任意数目的工作站数(即一个并行任务可划分的

---

* 本文研究得到国家攀登计划基金资助.作者周笑波,1973年生,博士生,主要研究领域为分布式处理,并行计算.陆桑璐,女,1970年生,博士,主要研究领域为分布式处理,并行计算.谢立,1942年生,教授,博导,主要研究领域为分布式计算,并行处理.

本文通讯联系人:周笑波,南京210093,南京大学计算机软件新技术国家重点实验室

本文1997-06-23收到原稿,1997-09-15收到修改稿

当系统负载较小、并行任务申请的处理器数较小以及系统中处理器总数较小时,3 种算法在响应时间和系统流量上差异不是很大,MAX 算法和 Grab 算法互有优缺点,Dasic 算法要略优于它们两者.

当系统负载较大、并行任务申请的处理器数较大以及系统中处理器总数较大时,Dasic 算法在响应时间和系统流量上要明显优于 MAX 算法和 Grab 算法,且对本地任务的影响较小.

这说明,在一个较大规模的 NOW 系统中,当并行处理要求的粒度较高、系统负载较大时,Dasic 算法的动态自适应能很大地提高协同调度的性能.

参考文献

1 Ousterout J K. Scheduling techniques for concurrent systems. In: Proceedings of the 3rd International Conference on Distributed Computing Systems (ICDCS). IEEE Press, October 1982. 22～30

2 Ousterout J K, Scelza D A, Sindhu P S. Medusa: an experiment in distributed operating systems structure. Communications of the ACM, 1980,23(2):92～105

3 Atallah M J, Lock C, Marinescu D C *et al*. Co-scheduling compute-intensive tasks on a network of workstations: models and algorithms. In: Proceedings of the 11th International Conference on Distributed Computing Systems (ICDCS). IEEE Press, May 1991. 344～352

4 Douglis F, Ousterhout J K. Transparent process migration: design alternatives and the sprite implementation. Software Practice and Experience, 1991,21(8):757～785

5 陆桑璐,谢立. 一个动态自适应的迁移和协同调度模型. 软件学报,1997,8(10):752～759
(Lu Sang-lu, Xie Li. A model for adaptive migration and coscheduling. Journal of Software, 1997,8(10):752～759)

6 Shoja G C. A distributed facility for load sharing and parallel processing among workstations. Journal of System and Software, 1991,14(3):163～172

7 周笑波,汲化,谢立. 一个基于 PVM 的异构型智能分布式任务调度系统. 计算机科学,1996,23(6):30～34
(Zhou Xiao-bo, Ji Hua, Xie Li. A heterogeneous intelligent distributed task scheduling system on the top of PVM. Computer Science, 1996,23(6):30～34)

Performance Evaluation of a Dynamically Adaptive Co-scheduling Algorithm

ZHOU Xiao-bo　LU Sang-lu　XIE Li

(*State Key Laboratory for Novel Software Technology　Nanjing University　Nanjing* 210093)
(*Department of Computer Science and Technology　Nanjing University　Nanjing* 210093)

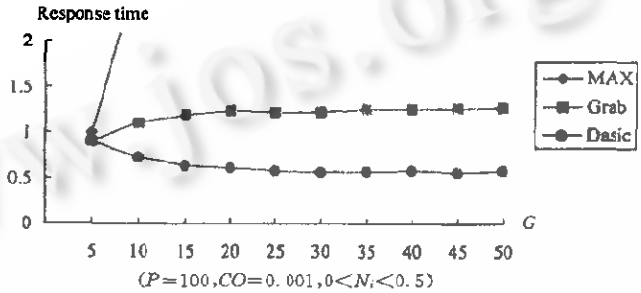
Abstract　In this paper, the performance evaluation of a dynamically adaptive co-scheduling algorithm —— Dasic based on network of workstations is presented. In terms of the aspects of mean response time and system flow, the Dasic algorithm is compared with traditional co-scheduling algorithm MAX and Grab algorithm by simulation. The conclusion is that dynamic adaptation can strikingly improve co-scheduling performance in a massive system of network of workstations.

Key words　Network of workstations, co-scheduling, dynamic adaptation.

当任务可申请处理器数 G 较小时，响应时间随着 G 的增大而减小得较快，这是因为随着 G 的增大，任务运行的并行度也随之提高，引起任务执行时间的降低. 但是，因受同步的影响，当 G 为 20 以后，MAX 和 Grab 算法的响应时间的降低程度变得非常缓慢，这表明，子任务之间的同步使得利用大量的处理器以提高并行变得困难，所以，只有在子任务数较少时，并行度的提高能够提高性能. Dasic 算法的响应时间在 G 大于 30 后降低程度变得不甚明显，但是仍要优于 MAX 算法和 Grab 算法.

(2) 负载较重时两者的关系

如图 4 所示，当系统负载较重时(相对利用率为 0.9)，MAX 算法由于其调度满足率大大降低，其增加的组等待时间变得难以接受，响应时间随着 G 的增大而呈指数上升. Grab 算法尽管对于某个任务的平行度可能会差于 MAX 算法，但是它尽量避免了组等待时间，其相应时间随着 G 的增大而呈良性上升. 此时，充分考虑提高系统利用率的 Dasic 算法的性能就比较明显地显示出来，其响应时间呈下降的趋势. 但是，当 $G>30$ 后，响应时间变化很小，这也是因为受到子任务需要同步的影响.

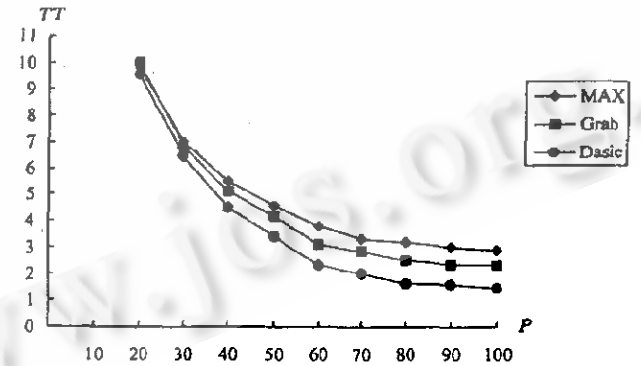


($P=100, CO=0.001, 0<N_t<0.5$)

图 4 响应时间与任务可申请工作站数的关系(2)

3.3 系统总完成时间和处理器数的关系

如图 5 所示，主要研究系统中处理器总数 P 从 20 变化到 100 时，系统总完成时间 TT 的变化，任务到达率为最大到达率.



($G=10, CO=0.001, 0<N_t<0.5$)

图 5 系统总完成时间与处理器数的关系

MAX 算法、Grab 算法和 Dasic 算法的系统总完成时间随着处理器数的增大均呈下降趋势. 因为在 ATM 上的通信代价极小，所以在处理器 $P_n<100$ 时，算法性能均没有出现极值点. 在起始阶段，MAX 算法的系统流量要优于 Grab 算法，但随着 P 的增大，Grab 算法的系统流量超过了 MAX 算法. 当 P 超过 40 时，Dasic 伸展调度带来的系统流量增大变得非常明显，其系统总完成时间比 MAX 和 Grab 算法明显要小.

4 讨论和总结

本文对 Dasic 算法、MAX 算法和 Grab 算法，在系统总完成时间与处理器数的关系、响应时间与相对利用率的关系以及响应时间与任务可申请处理器数的关系上作了分析和比较.

我们用 T_w^0 表示任务在一个所有工作站的 N_i 值为 0 的工作站组中的执行时间.但各工作站的 N_i 值往往不同,因为子任务之间的同步,执行得最慢的子任务决定了任务的完成时间.所以,对整个工作站组来说,有效的 N_i 值是组中最大的 N_i 值(表示为 N_g).

所以,对一个普遍性意义的任务而言,响应时间 T_r 可定义为:$T_r = T_q + T_w^g + T_x^g/(1 - N_g)$.

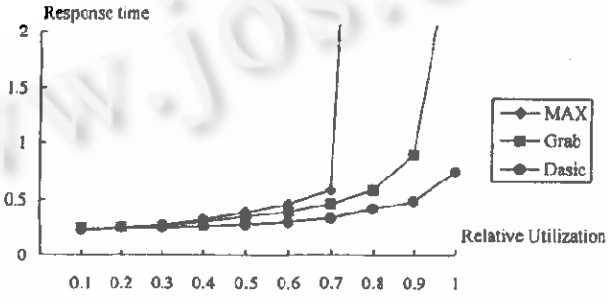
系统流量指的是,单位时间内系统同时处理的任务量.从系统的观点来看,希望系统流量越大越好,增加系统流量与减少系统总完成时间是一致的.所以,我们用系统总完成时间来反映系统流量这个性能指标.

3　性能分析

在进行性能分析时,为了叙述方便,我们约定下列符号所表示的概念:(1) G:任务可申请的最大处理器个数;(2) CO:每次通信所费代价;(3) P:系统中处理器的总数.

3.1　响应时间与相对利用率的关系

如图 2 所示,MAX 算法、Grab 算法和 Dasic 算法的响应时间随着负载的增大均呈上升的趋势.当系统负载较轻时,调度满足率较高,3 种算法得到的响应时间差异很小.



(P=100,CO=0.001,G=10,0<N_i<0.5)

图 2　响应时间与相对利用率的关系

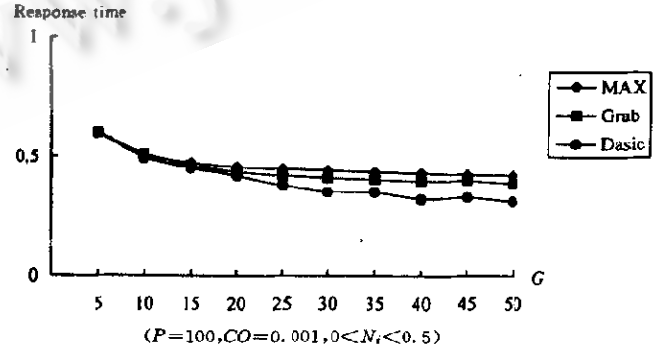
随着负载的增加,当相对利用率超过 0.7 后,MAX 算法因等待所申请的处理器数被满足,其增加的组等待时间掩盖了因并行度提高而得到的收益,其响应时间也随之急剧增大.相比之下,Grab 因为尽量避免组等待时间,此时得到的响应时间要小于 MAX 算法.Dasic 算法一方面尽量避免组等待时间,另一方面,当有空闲工作站时扩展,以得到较高的平行度,其响应时间随相对利用率的增加呈良性上升的趋势.

3.2　响应时间与任务可申请处理器数的关系

根据上述实验结果,我们分别在负载较轻和较重两种情况下分析响应时间随任务可申请处理器数 G 的变化情况.

(1) 负载较轻时两者的关系

如图 3 所示,当负载较轻时(相对利用率为 0.4),任务组调度的满足率较高,3 种算法的响应时间都随着 G 的增大而减小,但它们的响应时间差异显得很小.



(P=100,CO=0.001,0<N_i<0.5)

图 3　响应时间与任务可申请工作站数的关系(1)

新选择该工作站,这样,算法将会产生震荡现象.所以,当本地局部任务进入运行后,应先观察一段时间,再考虑迁移外来计算任务.

对于收缩算法来说,首要的目标应该是快速迁移,如果算法过于复杂,则可能大大降低系统的效率,因此,Dasic当前采用的是一个简单的收缩算法,即将散出的进程尽快地迁移到CPU利用率较低的处理机上.

2 系统评估模型

2.1 模拟系统

本文提出的模型中,假设系统中各工作站以最大优先级运行本地局部任务.一组由系统提交的计算密集型任务组成一个等待队列.当一个任务到达等待队列的列首时,由一个集中调度者根据其申请的工作站数(划分的子任务数),按不同的调度算法指定一组工作站,各工作站并行协同地运行这个任务.我们的模型中,任务是先来先服务的.当然,也可以用更复杂的策略,比如调度者计算等待任务队列,并排列好这些任务的分配次序,以求获得更好的响应时间.不过,因算法复杂而附加的计算时间对所获得的响应时间来说,有时是不可忽略的.

我们的模拟系统建立在3台通过155兆ATM相连的RISC 6000工作站上.如图1所示,模拟系统包括7个部分:任务发生器、协同调度器、定时器、模拟处理器池、随机状态发生器、数据处理器和虚拟网络.

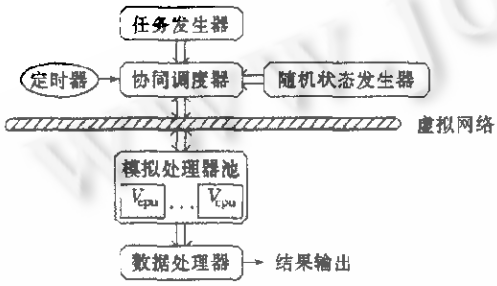


图1 模拟系统的结构

(1) 任务发生器 为了保证样本空间[7]的普遍性,任务发生器通过一个随机数发生器来构造任务组的调度特性.根据任务组长度限制等条件,以各任务的运行代价、通信代价和任务到达时间为其调度特征,任务发生器的频率受任务到达率的控制.在我们的模型里,任务组的到达满足泊松分布.

(2) 协同调度器 协同调度器的主要任务是模拟协同调度算法从获得当前系统状态到产生调度方案的过程,并且统计该过程的代价.它分为两个部分,状态获取器和调度产生器.在Dasic算法中还有一动态自适应协调管理器,能根据系统状态的动态变化进行动态自适应地协调管理.

(3) 定时器 定时器中有一个系统唯一的逻辑时钟,用于定时地唤醒协同调度器来控制虚拟处理器作虚拟协同任务调度,所设定时的时间段称为调度周期,也就是一个逻辑上的时间片.

(4) 模拟处理器池 模拟处理器池是一个由可变数量的虚拟处理器组成的模拟装置,每个虚拟处理器模拟实际处理器的运行.在实现中,一个处理器用一个进程来模拟.

(5) 随机状态发生器 随机状态发生器的功能就是针对NOW环境的动态变化,模拟其系统状态知识的不确定性,包括产生空闲的虚拟处理器组及其"空闲度",或产生局部任务的到达,以激活Dasic算法中动态自适应协调管理器的可伸缩调度.其发生频率由单位时间状态变化率参数控制.

(6) 数据处理器 数据处理器从模拟处理器池中收集状态数据,并对它们进行归纳、整理、加工.

(7) 虚拟网络 由于所模拟的协同调度器属于模拟处理器池中的某一个虚拟处理器,调度器与其他虚拟处理器之间存在一个虚拟网络.这个虚拟网络决定了各虚拟处理器之间的通信代价,其控制参数包括虚拟处理器的最大连接数以及它们之间的互连限制等.虚拟网络采用总线型结构.

2.2 性能指标

评价一个调度系统,一般从性能和效率两个方面来评价.性能刻画了用户从有效的调度中得益的程度和资源的有效利用率;而效率则与调度程序本身的额外开销有关.通常性能指标包括响应时间和系统流量两个方面.

响应时间是指任务从提交到完成所需的时间,包括任务的排队时间和运行时间,有些算法还包括组等待时间,如MAX算法.对提交任务的用户而言,响应时间越小越好.从系统的观点看,协同调度的目标之一就是极小化平均响应时间.

设T_q表示一个任务在等待队列中的等待时间,当一个任务到达队列的列首后,若此时没有一个工作站是空闲的,则T_q还包括任务等到第1个工作站变为空闲的时间.

设$T_{\&}$表示组等待时间,即系统当前存在有空闲的处理机,算法(如MAX)还要等申请的工作站组均变为空闲所花费的时间.

最大子任务个数).为什么要定义一个任务可申请的最大工作站数呢?我们认为在协同调度环境中,并行处理的粒度是有一定限制的.

(3) 空闲标准

NOW 环境中任何时刻都存在可利用的"空闲"工作站,通常情况下认为,一个工作站空闲与否主要依据其负载特性,实际也就是指处理器负载.所以,制定空闲标准的实质是:在本地工作站 CPU 利用率为多大时,还允许接收外来计算密集型任务.在我们的模型中,每一台工作站有一个负载参数 N_i 表示其 CPU 利用率,根据文献[2]使用的空闲标准,定义当 $N_i<0.5$ 时,工作站结点 i 是可被利用的,即剩下的处理能力可为外来的计算密集型任务所使用.

(4) 低利用率

在协同调度中,因为子任务之间的同步,一个任务只有当其执行得最慢的子任务执行完毕后才能完成,这样,在一个被指派运算一个任务的工作站组中,就有可能有几个实际上已"空闲"的工作站因等待组中运行最慢的工作站计算完毕而浪费其处理机资源.所以,协同调度算法中采用减小组内各工作站间的速度差异的选择策略,会有利于系统利用率的提高.

1.2 MAX 算法简介

当一个全局计算任务申请 q 个工作站来协同计算时,由集中调度员从系统中根据空闲标准的定义,选取 q 个"空闲"可利用的工作站.

(1) 若有不止 q 个工作站是空闲的,则调度员在其中选择 q 个 N_i 值最小的工作站.

(2) 若不到 q 个工作站是空闲的,则调度员就等待某个工作站上的任务计算完毕并释放此工作站...,直到有 q 个工作站成为空闲.

MAX 算法在申请工作站数时是较为"贪婪"的,因为它为了利用尽可能多的工作站来提高任务的并行度,而以一段组等待时间为代价.任务在其申请的工作站数被满足之前,将会一直等待,而此时一些工作站实际上可能是空闲可利用的,其资源将被浪费.

1.3 Grab 算法简介

当一个任务申请 q 个工作站来协同计算时,集中调度员从系统中根据空闲标准,选取 q 个可利用的工作站.

(1) 若有不止 q 个工作站是空闲的,则调度员在其中选择 q 个 N_i 值最小的工作站.

(2) 若不到 q 个工作站是空闲的,则调度员就将它们全部指派给这个任务,并开始执行(这时会产生一个工作站运行不止一个子任务的"折叠"情况).

(3) 若没有空闲工作站存在,调度员就一直等待到有一个任务运行完毕并释放其工作站.

Grab 算法利用当前可用的工作站数,而不是等待直到申请数目的工作站都变为空闲后才投入运行.所以,一旦系统当前存在有空闲可利用的工作站,它就尽量避免组等待时间,但其某个任务的并行度不一定高.

1.4 Dasic 算法简介

根据 DASIC 模型的思想,我们设计了 Dasic 算法,包括初期调度算法、伸展算法和收缩算法 3 个部分.

(1) 初始调度算法

类似于 Grab 算法,当系统当前可利用的空闲工作站数小于等待队列的列首任务申请的工作站数时,Dasic 初始算法将该任务"折叠"地分配到可利用的工作站组上运行.另外,Dasic 算法中增加了一个减小组内各工作站间的速度差异的选择策略.当系统开始运行后,由于空闲结点的加入和退出以及进程组的执行结束,造成进程空间的改变,这就需要进行动态自适应地可伸缩性调整.

(2) 伸展算法

当系统中因别的任务组运行结束而释放处理机资源,因而增加了 S 台空闲工作站后考虑"伸展"调度时,若初始调度存在"折叠"运行的任务,我们先将这种"折叠"的任务迁移到当前产生的空闲的工作站上运行.

若当时没有"折叠"的任务,则将等待队列的列首任务分配到空闲的工作站组上运行.

若等待队列也为空,则:如果当前被释放成空闲的工作站中有某运行任务初始调度选择的工作站组中的工作站(当时被别的任务占用)时,对已运行任务的工作站组中最大的"空闲度"与该空闲工作站的"空闲度"进行比较;若迁移的代价和浪费的已执行代价之和小于因空闲度增加而得到的加速,则迁移该任务至当前产生的"最优"空闲工作站上.

(3) 收缩算法

当某台工作站的本地用户返回使用时,其上的外来进程应该撤出以便于本地用户可以独占使用.这时,如果立即迁走其上运行的外来任务的子任务,可能会发生本地局部任务只运行的该子任务在重新选择空闲机时,还是有可能重