计算机科学 1999Vol.26№.6

并行分布计算中的任务调度模型*)

Task Scheduling Model in Parallel and Distributed Computing

陈华平　黄刘生　陈国良

(国家高性能计算中心(合肥)　中国科学技术大学计算机系　合肥 230027)

Abstract　In this paper, we first describe the concept of task scheduing in Parallel and Distributed Computing(PDC), then illustrate the task scheduling model in PDC and the way of calculating the execution cost and communication cost, and lastly discuss an approach to estimate the communication contention overhead.

Keywords　Parallel and distributed computing, Task scheduling, Model

1　调度问题的一般模型

调度问题在不同的领域有许多不同的描述方法,生产管理中的生产作业安排问题就是一个最为经典的调度问题。一般地,构成调度问题的基本元素有三个,即资源集、消费者集及这些资源为这些消费者服务所依据的一定规则,调度问题就是在满足资源集和消费者集约束条件的基础上,设计一个有效的调度系统来管理消费者如何高效地使用这些资源,并使得一些系统性能指标达到最优或近似最优,它的一般模型如图1所示。

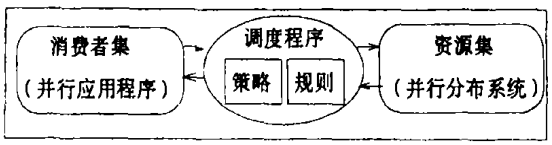

图1　调度问题的一般模型

组成一个并行程序的子任务、多用户系统下的作业、工厂中的生产作业或银行中的顾客相当于消费者,并行分布系统中的处理单元、工厂中的机器或银行中的出纳相当于资源。调度规则和策略多种多样,其中先来先服务是一个典型的调度策略,但它只适合于某些情形,如银行服务可以使用,但未必适用于工厂车间中的生产作业。

调度性能和调度效率是评价一个调度系统优劣程度的两个方面。每个调度问题都有自己特定的一些性能测试指标(有时也称为系统性能指标),如生产车间的产品产量、并行分布计算中一个并行应用程序的执行时间、多用户系统下作业的平均响应时间或系统吞吐率等,作为调度系统它的目标就是要尽量优化这些指标,调度性能也就是通过这些指标的取值来反映,它直接体现了调度结果的好坏。例如,如果把并行应用程序的执行时间长度作为性能指标,那么能产生较短执行时间的调度系统较优。调度效率主要指调度系统本身的复杂度,如果两个不同的调度系统产生相同的调度结果,那么复杂度小、简单明了的调度系统较优。在并行分布计算中,调度系统的复杂度主要指调度算法的时间复杂度,调度问题一般都具有NP完全难度[1,2],各种取得最优或近似最优的调度算法必须具有多项式时间复杂度时,才是有效的调度算法。

2　并行分布计算中的任务调度模型

2.1　应用程序任务

一个并行应用程序的性质可用$(A,\prec,D,W)$来刻画,其中$A=\{a_k|k=1,2,\cdots,n\}$为任务集,$D=\{d_{ij}|i,j=1,2,\cdots,n\}$是一个$n\times n$的通讯矩阵,$d_{ij}(\geq 0)$表示任务$a_i$传送给任务$a_j$的数据量;$W=\{w(a_i)\}$,

*)本文受中国科大青年基金和华为青年基金的资助。**陈华平**　博士,副教授,主要从事并行分布计算和网络计算的研究。**黄刘生**　副教授,现主要从事分布算法方面的研究。**陈国良**　博士生导师,国家高性能计算中心(合肥)主任,现主要从事并行分布计算和智能计算的研究。

其中 $w(a_i)$ 表示任务 a_i 的工作负载大小，在同构的并行分布系统中，由于每个结点的计算能力(包括运算速度、内存大小等)相同，所以也可以直接用 $w(a_i)$ 来表示任务 a_i 的执行成本，而在异构的并行分布计算环境下，相同负载在不同结点上的执行成本是不相同的；"$\prec$"定义了任务间的偏序关系，"$a_i \prec a_j$"表示任务 a_j 的执行依赖于任务 a_i 的执行。

在并行分布计算中，偏序关系"$\prec$"一般用任务图 $G=(V,E)$ 来表示，其中 $V=\{a_1,a_2,\cdots,a_n\}$ 为表示 n 个任务的有权结点，任务 a_i 的工作负载 $w(a_i)$ 为该结点的权重；$E=\{(a_i,a_j)\mid a_i,a_j\in V\}$ 为表示任务间通讯关系的带权有向边集，任务 a_i 与 a_j 之间的通讯量 d_{ij} 为这些有向边的权。$IMP(a_i)=\{a_k\mid(a_k,a_i)\in E\}$ 表示任务图中结点 a_i 的直接前趋集，$IMS(a_i)=\{a_k\mid(a_j,a_k)\in E\}$ 表示任务图中结点 a_j 的直接后继集。如果 $IMP(a_i)=\phi$，那么称结点 a_i 为入口结点；如 $IMS(a_j)=\phi$，那么称结点 a_j 为出口结点。用 $path(a_i,a_j)$ 表示 G 中结点 a_i 到结点 a_j 的一条路径，它本身就是包含结点 a_i 和 a_j 在内的该条路径上的所有结点集合。

2.2 目标机器

一般地，目标机器假定由 m 个处理单元组成，该 m 个处理单元可以是同型的，也可以是异型的，它们通过任意的互连网络进行连接。每个处理器某一时刻只能处理一个任务，每个任务可在任何一个处理器上运行。一般地，可用六元组 $TM=(P,[P_{ij}],[S_i],[I_i],[B_i],[R_{ij}])$ 来刻画目标机器的特性，其中：(1)$P=\{P_1,P_2,\cdots,P_m\}$ 为构成并行分布系统的一组处理器；(2)$[P_{ij}]$ 为 $m\times m$ 的互连网络拓扑矩阵；(3)$S_i(1\leqslant i\leqslant m)$ 为处理器 P_i 的执行速度；(4)$Q_i(1\leqslant i\leqslant m)$ 表示在处理器 P_i 上启动一个消息传递所需的时间；(5)$B_i(1\leqslant i\leqslant m)$ 表示在处理器 P_i 上启动一个进程执行所需的时间；(6)$R_{ij}(1\leqslant i\leqslant m,1\leqslant j\leqslant m)$ 为两个相邻处理器之间的消息传输率，即每单位时间传送的数据量。为了简明起见，许多任务调度算法对目标机器这六个参量进行了不同程度的假设，例如假定处理器是全连接的；每个处理单元的执行速度相同，这样任务 a_i 的执行时间可用任务的工作负载 $w(a_i)$ 来代表；每两个相邻处理器之间的消息传输率相同，为常数 R；每个处理器的消息传递启动时间相同，均为 α；进程启动时间与进程执行时间相比可忽略等。

2.3 执行成本与通讯成本

如用 t_{ij} 来表示任务 a_i 在处理器 P_j 上的执行时间，那么：

$$t_{ij}=w(a_i)/s_j+B_j$$

设 $C_{i_1j_1,i_2j_2}$ 表示处理器 P_{j_1} 上的任务 a_{i_1} 与处理器 P_{j_2} 上的任务 a_{i_2} 之间的通讯开销，如果这两个处理器相邻，那么：

$$C_{i_1j_1,i_2j_2}=d_{i_1i_2}/R_{j_1j_2}+I_{j_1}+CD_{j_1j_2}$$

其中 $R_{j_1j_2}$ 表示这两个处理器间的传输率，$CD_{j_1j_2}$ 表示 P_{j_1} 与 P_{j_2} 之间通讯线路的竞争开销。

如果 P_{j_1} 与 P_{j_2} 之间不相邻，为了表达简单起见，我们假定每两个相邻处理器之间的消息传输率均为 R，每个处理器的消息传递启动时间均为 I，用 $H_{j_1j_2}$ 表示处理器 P_{j_1} 到处理器 P_{j_2} 的线路条数(也称之为中继段数)，那么：

$$C_{i_1j_1,i_2j_2}=(d_{i_1i_2}/R+I)*H_{j_1j_2}+CD_{j_1j_2}$$

为了简化起见，许多任务调度算法[1,3](特别是基于 SPMD 执行模式的任务调度算法)把通讯成本函数定义为

$$y(x)=\alpha+\beta x$$

其中 α 为消息传递启动时间，$\beta=(1/R)$ 为每两个相邻处理器之间传送单位数据量所需要的时间，x 为通讯数据量大小，并且假定消息发送者每次只能发送一个消息，而消息接收者可缓冲接收所有传来的消息。另外，一般如无特别指出，假定给定的任务权重与通讯边权重已分别换算成同一时间单位的执行成本和通讯成本，这样就可以直接把这些数据应用于任务调度算法。

2.4 调度及调度目标

在并行分布计算中，并行应用程序的一个调度其实就是其对应的任务图 G 到目标机器的一个映射 $f:V\mapsto\{P_1,P_2,\cdots,P_m\}\times[0,\infty)$，$f(a_i)=(p,t)$ 表示任务 a_i 被调度到编号为 p 的处理器上，起始执行时刻为 t。一般地，我们可用 *Gantt* 图 $GC=\{p(a_i),t(a_i))\mid a_i\in V\}$ 来直观地表示调度结果，其中函数 $p(a_i)$ 表示分配给任务 a_i 的处理器号，$t(a_i)$ 表示任务 a_i 的起始执行时刻。

一般入口结点任务的起始执行时刻以零计算，那么所有任务执行完的那个时刻称为调度长度 SL，它也反映了整个并行程序任务的执行时间，显然 $SL=\underset{1\leqslant i\leqslant n}{MAX}\{t(a_i)+w(a_i)/S_j\}$，其中 $j=p(a_i)$。一般来说，任务调度的目标就是要获得最短的整个应用程序执行时间，亦即上述的调度长度。在许多启发式任务调度方法[4]中，有时直接以调度长度为目标函数来进行操作非常困难，所以经常以保持负载平衡、减

少任务间通讯量或减短关键路径长度[5]等为目标函数来执行调度操作，但最终还是以获得较短的调度长度为目标。

图2是一个示例任务图，每个圆圈表示一个任务，其中横线上面为任务名(任务号)，横线下面为该任务的计算负载的单位时间数量，圆圈间的有向边表示任务间的执行关系，有向边旁边的数字代表通讯延迟的单位时间数量。图3是目标机器模型，该目标机器由三个处理器组成，它们之间是全连接的，并且每个处理器的性能相同。图4就是图2所示任务图在图3所示目标机器上的一种调度方案，该调度结果用Gantt图来表示，其中顶部为处理器号，左边一串数字为执行时刻。由于任务 a_1 和 a_2 被分配到同一处理器上，因此它们之间的通讯延迟忽略为零，而由于任务 a_1 和 a_3、a_4 被分配到不同的处理器上，所以 a_1 到 a_3 和 a_1 到 a_4 的通讯延迟不能忽略，分别为1个时间单位和2个时间单位。

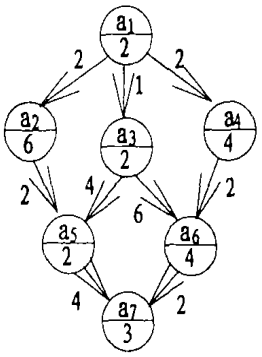

图2 一个示例任务图

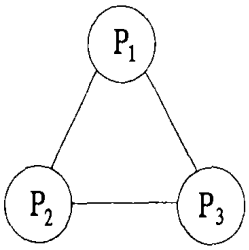

图3 目标机器

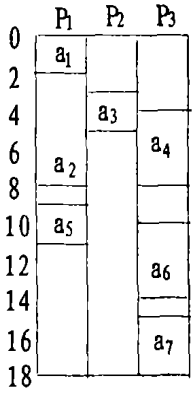

图4 调度结果的Gantt图表示

3 通讯竞争开销的估计

3.1 选路表

一般说来，通讯延迟由两部分构成，一部分是传送数据所引起的延迟，另一部分是由于多个消息竞争同一线路引起的排队延迟(也就是竞争开销)。在以前的许多任务调度算法中，只考虑前一部分的通讯开销，而忽略了后面一部分竞争开销，其实在消息传递比较频繁时，后一种开销占有相当的比例[6]。为了比较精确地估计由竞争引起的开销，任务调度算法为每个处理器 $P_i(1\leqslant i\leqslant m)$ 建立一个选路表 RTB_i，该选路表由 $m-1$ 条记录 $R_j(1\leqslant j\leqslant m$ 且 $j\neq i$，下同)组成，每条记录记载了 P_i 对其它处理器的当前通讯状况。每个记录由 H、L 和 C 三个域组成，为了简单起见，我们用 H_{ij} 表示处理器 P_i 到处理器 P_j 的中继段数；$L_{ij}(1\leqslant L_{ij}\leqslant m)$ 是处理器号，表示处理器 P_i 给处理器 P_j 传递消息时，P_i 准备选择使用的第一个中继处理器号，亦即表示了 P_i 准备使用的通讯线路；C_{ij} 表示当前 P_i 给 P_j 传递消息时，由于竞争而引起的通讯开销。初始时，任务调度算法把 CD_{ij} 均置为零，并根据拓扑邻接矩阵 $[P_{ij}]$，利用处理器两两之间的最短路径求得 H_{ij} 和 L_{ij}。如果两点间多于一条最短路径，那么任选一条作为 H_{ij} 和 L_{ij} 的初值计算依据。

选路表主要用于获得传递一个消息的最佳路径，以及通过竞争延迟估计，来更精确地选择执行某一任务的最佳处理器。在调度过程中需要不断地修改这些选路表，这样就能根据当前的通讯交通情况来选择适当的传送线路和执行处理器。一般说来，对这些选路表的修改操作频繁些，那么对整体通讯交通情况就更了解得清楚全面些，但这些表操作的时间复杂度也就要高一些，这两者之间本身有一个折衷。每当一个任务向其它处理器上的另一任务发送一个消息，或者一个消息到达目的地后，任务调度算法就根据消息发送和消息接收这两个事件来修改与相应通讯线路直接相关或间接相关的选路表。

3.2 选路表的修改操作

每当发生消息发送和消息接收事件时，就对某些处理器的选路表进行修改。修改操作可分为两大类，其中一类是修改与当前发生事件直接有关的选路表，也就是该通讯线路上的处理器的选路表。假定处理器 P_{j_1} 上的任务 a_{i_1} 给处理器 P_{j_2} 上的任务 a_{i_2} 发送一个消息，那么该消息在两个相邻处理器之间的单位传输延迟 $td=(d_{i_1i_2}/R+I)$，我们可通过递归过程 $DelayUpdating(j_1,j_2,var,delay)$ 来进行修改操作，其中 j_1 为源处理器号，j_2 为目的处理器号，变量

参数 delay 为修改后的竞争开销值 $CD_{j_1j_2}$，该递归过程的主要步骤为：

(1)如果 $L_{j_1j_2}=j_2$，那么 $delay=CD_{j_1j_2}+td$；否则转第(2)步；

(2)分别递归调用 $DelayUpdating(j_1,L_{j_1j_2},var\ delay_1)$ 和 $DelayUpdating(L_{j_1j_2},j_2,var,delay_2)$，然后把 $delay_1$ 和 $delay_2$ 值相加作为 $delay$ 值。

如果发生的是消息接收事件，那么只要把(1)中的"+"号改为"-"号即可，(1)也为递归终止条件。

另一类是修改与所发生事件间接有关的选路表，也就是与该通讯线路上的处理器相邻的那些处理器的选路表。为了实现该修改操作，可建立一个用于存放处理器号的队列 PQ，设消息发送和消息接收事件所对应的源处理器号和目的处理器号分别为 j_1 和 j_2，那么 PQ 的初始值为除了 $L_{j_1j_2}$ 以外与 j_1 相邻的处理器号。下面是间接修改算法的主要步骤。

(1)从 PQ 移出队首元素 P_u，如果 RTB_u 还没进行修改操作，就执行下一步，否则转(4)；

(2)设与 P_u 相邻的处理器号集为 $B=(b_1,b_2,\cdots,b_t)$，对 RTB_u 中的每个记录 $R_k(1\leqslant k\leqslant m$ 且 $k\neq u)$值进行下列修正：

(2.1)在 B 中找出使得 $CD_{ub_g}+CD_{b_gk}$ 最小的处理器号 b_g；(2.2) $H_{uk}=1+H_{b_gk}$，$L_{uk}=b_g$，$CD_{uk}=CD_{ub_g}+CD_{b_gk}$；

(3)把 B 中元素放入 PQ 中；

(4)重复上述过程，直到 PQ 为空为止。

结束语 任务调度是并行分布计算中最为基本、最为关键并最具有挑战性的问题之一，是影响并行分布计算执行效率的一个关键因素。随着基于并行分布处理的高性能计算的日益普及，如何针对不同的具体应用与并行分布计算环境，而采用有效的调度方法来提高系统执行性能，已愈来愈成为人们的研究热点。本文主要说明了并行分布计算中任务调度问题的一般模型，同时给出了通讯竞争开销的一种估计方法。随着网络工作站机群系统等并行分布环境的不断发展，作业调度、任务划分等这些具有一些特殊性的任务调度问题也将越来越显得十分重要[7]。

参考文献

1 El-Rewini H, et al. Task scheduling. PTR Prentice Hall, Englewood Cliffs, New Jersey, 1994. 07632

2 Lewis T, El-Rewini H. Intrduction to parallel computing. Prentice Hall, 1992

3 陈华平，周学海，陈国良．分布存储环境下 SPMD 执行模式的任务调度问题．计算机科学，1996，23(6)

4 陈华平，林洪，陈国良．并行分布计算中的启发式任务调度．计算机研究与发展，1997，34(suppl.)：74～78

5 Gerasoulis A, Yang T. A comparison of clustering heuristics for scheduling DAGs on multiprocessors. Journal of Parallel and Distributed Computing, 1992

6 El-Rewini H, Lewis T. Scheduling paralleling program tasks onto arbitrary target machines. Journal of Parallel and Distributed Computing, 1990：138～153

7 陈华平，计永昶，等．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

(上接第 77 页)

```
Service occur. delete
Service occur. query
.
End Specification
```

在作了上述详细的对象定义后，我们利用开发工具自身提供的对象设计和规范化定义，开发相应的数据输入界面、处理操作界面、数据查询界面、数据报表界面、数据维护界面等类库，利用类的继承关系，很容易生成其他模块的用户程序。先把相应的参数和基础数据类对象应用到企业 MIS 工具类库中生成物资管理分系统的不同模块，然后再在逻辑上作一定的调整，得到物资管理分系统—企业 MIS 的一个部件。

结束语 利用上述方法我们设计和开发了一个通过了 ISO9001 论证的大型制药企业的基于广域网的 MIS 系统，整个工程已完成并投入运行。在 UNIX/TCP IP/SYBASE 环境下，选用了 PowerBuild (PB)工具平台开发用户界面，很好地利用了 PB 的 OO 特性，大大地缩短了开发周期。我们正在致力开发和完善更方便更行之有效的企业 MIS 的工具，构筑好整个软件构架，并充分实现软件重用。

参考文献

1 车敦仁，周立柱，等．软件体系结构、应用平台及框架仓库技术．计算机研究与发展，1996，33(7)：501～506

2 Krueger C W. Software Reuse. ACM Computer Survey, 1992, 24(1)

3 Cheng J. Improving the Software Reusability in Object-Oriented Programming. ACM SIGSOFT, Software Notes, 1993, 18(4)

4 Wellsetala D L. Architecture of an Open Object-Oriented Database System. Computer, 1992, 25(10)

5 诸葛海．面向对象的 MIS 开发方法．软件学报，1995(2)