启发式任务调度中的处理器选择策略*

陈华平 黄刘生

(国家高性能计算中心 合肥 230027)
(中国科学技术大学计算机科学技术系 合肥 230027)
E-mail: hpchen@ustc.edu.cn

摘要 任务调度是并行分布计算中最为基本、最为关键、也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.现有的基于任务静态优先级的启发式任务调度方法都是以"当前任务具有最早起始执行时刻"为目标来选择执行处理器.该文在详细分析讨论该种调度方法的基础上,指出了以该目标选择处理器存在的问题及缺点,并提出了以"当前任务的直接后继具有最早起始执行时刻"为目标选择处理器的方法,并给出了相应的约束条件.

关键词 并行分布计算,启发式任务调度,处理器选择.

中图法分类号 TP316

既然并行分布计算中一般的任务调度问题均具有NP难度,那么,利用任务图本身所包含的一些启发信息来获得最优调度的近似解是经常被采用的一种技术方法.其中基于任务静态优先级的启发式任务调度方法,即表调度(list scheduling)是并行分布计算中最为常用的一种任务调度技术[1~3].它的主要思想是,在编译时给任务图中的每个任务按一定规则赋一优先级,并且把当前的所有就绪任务按优先级的大小排列,建立一个就绪任务队列RTQ.然后,每次从队列中选择优先级最高的任务(称该任务为当前任务),按一定规则把该任务分配到合适的处理器上,并计算出此任务在该处理器上的起始执行时刻,最后获得以Gantt图[1,4]形式表示的整个任务图的调度结果.

目前,基于表调度的这类启发式任务调度算法很多,但它们的思想都很相似,主要区别在于给任务赋优先级的方式有所不同.有些是在基本表调度算法基础上通过任务插入和任务复制等技术来提高启发式任务调度算法的执行性能[2,5],这些算法均以"当前任务具有最早起始执行时刻"为目标来选择该当前任务的执行处理器号.本文的主要工作就是在分析基本的表调度算法和其改进算法的基础上,指出表调度算法中单以"当前任务具有最早起始执行时刻"为目标选择执行处理器存在的问题及缺点,并提出了以"当前任务的直接后继具有最早起始执行时刻"为目标选择处理器的方法,同时给出了相应的约束条件.

1 基本模型及定义

通常,一个并行程序的性质可用$(V,\prec,[d_{ij}],w(a_i))$来刻画,偏序关系"$\prec$"一般用任务图$G=(V,E)$来表示,其中$V=\{a_i|i=1,2,\ldots,n\}$,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;$E=\{(a_i,a_j)|a_i,a_j\in V\}$为表示任务间通信关系的带权有向边,$d_{ij}$为任务$a_i$到$a_j$的通信量;$IMP(a_i)=\{a_k|(a_k,a_i)\in E)\}$,表示任务图中任务$a_i$的直接前趋集,$IMS(a_i)=\{a_k|(a_i,a_k)\in E)\}$,表示任务图中任务$a_i$的直接后继集.

任务图的一个调度其实就是任务图G到目标机器的一个映射$f:V\mapsto\{1,2,\ldots,m\}\times[0,\infty)$,$f(a_i)=(p,t)$,表示任务$a_i$被调度到编号为$p$的处理器上,起始执行时间为$t$.一般地,我们可用Gantt图$GC=\{(p(a_i),$

* 本文研究得到国家863高科技项目基金和安徽省自然科学基金资助.作者陈华平,1965年生,博士,副教授,主要研究领域为并行分布计算、网络计算.黄刘生,1957年生,副教授,主要研究领域为分布算法,OODB.

本文通讯联系人:陈华平,合肥 230027,国家高性能计算中心

本文1998-09-18收到原稿,1999-03-01收到修改稿

$t(a_i))|a_i \in V\}$来表示调度结果，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.

定义 1. 设 a_i 为任务图的一个结点，如果 $IMP(a_i)=\varnothing$，则称结点 a_i 为入口结点；如果 $IMS(a_j)=\varnothing$，则称结点 a_j 为出口结点.

定义 2. 任务图中结点 a_i 到结点 a_j 的一条路径的计算长度为该路径上包括起始结点 a_i 和终止结点 a_j 在内的所有结点的权重之和.

定义 3. 任务图中一个结点的出口长度为从该结点到出口结点的最大计算长度，结点的入口长度为从入口结点到该结点的最大计算长度.

很明显，一般情况下，出口长度大的任务或入口长度小的任务优先执行. 当采用任务结点的入口长度作为该任务优先级的主要部分时，那么结点的出口长度就成为主要调度启发信息. 另外，入口长度、关键路径等也是常用的一种调度启发信息. 在以下的算法中，我们就用结点的出口长度作为主要启发信息. 当两个结点的出口长度相同时，可进一步通过它们的直接后继结点的数目来决定它们的优先级的大小.

如果已经把一个任务调度到某一处理器上，那么我们称该任务是已分配的. 如果结点任务 a_i 的直接前趋已全部分配完毕，可把该结点作为就绪任务插入到就绪任务队列 RTQ 中，插入后，RTQ 还是保持按照上面定义的优先级大小排列. 很明显，一开始只有入口结点任务为就绪任务. 在就绪任务队列中具有最高优先级的队首任务也叫当前分配任务，用 a_i 来表示，被选中执行 a_i 的处理器也叫当前处理器.

在分布存储的并行计算模型上，严格地讲，像出口长度这样的启发信息应该包括通信延迟. 但是，任务间的通信延迟与这两个任务所分配处理器的位置有关，把同一结点调度到不同处理器上所产生的通信延迟是不同的. 例如，任务 a_i 与 a_j 之间的通信量为 $d_{ij}\neq 0$，如果 a_i 与 a_j 被调度到同一处理器上，那么它们之间的通信延迟可以忽略，否则，就必须考虑这个通信延迟. 事实上，如果目标机器是异构的，那么各处理器的执行速度也不尽相同，这时，同一任务分配到不同处理器上的计算时间也是不一样的，所以，在这时的任务图中各结点的权重也是可变的，因而结点的出口长度和入口长度也是可变的. 在下面的分析讨论中，假定处理器之间是全连接的，任务图中的任务结点权重和通信边权重已换算成同一时间单位，并且通信边权重已包含消息传递启动时间.

2　表调度算法的基本思想

为了在分布存储的并行模型上采用静态的启发式任务调度算法，我们也可用结点的出口计算长度作为主要启发信息，但在求解任务 a_i 的起始执行时刻时，还必须考虑 a_i 的所有直接前趋任务与 a_i 之间的通信延迟带来的开销. 其实，在给一个任务 a_i 分配处理器时，优先考虑包含 $IMP(a_i)$ 结点的处理器，通过把消息传递量较大的源结点和目标结点放在同一处理器上以减少通信延迟.

设任务 a_i 在处理器 $p(a_i)$ 上的起始执行时刻为 $est(a_i,p(a_i))$，执行完成时刻为 $ect(a_i,p(a_i))$，那么，$ect(a_i,p(a_i))=est(a_i,p(a_i))+w(a_i)$.

定义 4. 设一个任务 a_u 在处理器 P_v 上的消息就绪时刻用 $mrdt(a_u,P_v)$表示，它为任务 a_u 在 P_v 上执行时，接收到所需全部消息的最早时刻.

一个任务的消息就绪时刻主要由它的直接前趋任务结点所决定，即

$$mrdt(a_u,P_v)=\max_{a_k\in IMP(a_u)}\{ect(a_k,p(a_k))+d_{ku}\}. \tag{1}$$

一个任务的消息就绪时刻与该任务所在的处理器有关，如果一个任务被分配到它的某一前趋所在处理器上的话，那么与该直接前趋的通信延迟可忽略. 使式(1)取得最大值的前趋任务也叫做任务 a_u 的关键前趋，由于 a_u 的每个直接前趋任务的执行完成时刻不同，所以，a_u 的关键前趋不一定是与 a_u 间通信量最大的直接前趋.

如果想把任务 a_u 分配在 P_v 上执行，那么至少要等到消息就绪时刻 $mrdt(a_u,P_v)$才有可能开始执行，至于到该时刻能否真正开始执行，还要看 P_v 上已分配的其他任务的执行情况. 我们把一个处理器已执行完其上面已分配任务的时刻称为该处理器的就绪时刻，用 $prdt(P_i)$来表示处理器 P_i 的就绪时刻，如果不考虑消息就绪时刻的话，它也就是 P_i 能执行其他再分配任务的最早时刻. 在整个系统中，当前具有最早就绪时刻的处理器称为最先

就绪处理器.

根据上面的分析可知,如果把一个任务 a_u 分配到处理器 P_v 上执行,那么任务 a_u 在 P_v 上的最早起始执行时刻为

$$est(a_u,P_v)=\max\{prdt(P_v),mrdt(a_u,P_v)\}. \tag{2}$$

目前,并行分布计算中基于表调度的启发式任务调度算法就是按照式(2)来选择使当前任务具有最早起始执行时刻的处理器.

图1是一个示例任务图,把每个任务的出口计算长度作为该任务的优先级.图2是以出口计算长度为启发信息的表调度算法LS1对该任务图进行调度所获得的Gantt图,其调度长度为12.

现在对图2稍加分析观察可知,对调度长度具有直接影响的是任务执行序列$(a_1,a_3,a_4,a_5,a_6,a_7)$,这个序列我们也称为关键任务序列.从图2简单地来看,任务 a_5 似乎对调度长度影响不大,但实际上,正是因为 a_5 到 a_6 也存在较大的通信延迟,使得 a_6 不能与 a_4 放在同一个处理器上,所以,a_5 也对调度长度有直接影响,也属于关键任务序列.而任务 a_2 虽然具有较高的优先级,但是从图2可以看出,即使晚一些执行 a_2,也不会对调度长度产生什么影响.既然直接影响调度长度的是关键任务序列,那么任务调度算法能否优先考虑该关键任务序列的执行呢?实际上,在获得表示调度结果的Gantt图之前,我们并不了解直接影响调度长度的关键任务序列,通过图1与图2对比可发现,(a_1,a_3,a_4,a_6,a_7)刚好是示例任务图的关键路径.一般情况下,关键路径与关键任务序列并不一定相同,但我们可以利用关键路径作为主要启发信息进行任务调度.

关键路径上的任务偏序关系是确定的.以关键路径作为主要启发信息的表调度算法LS2根据关键路径上任务的偏序关系选择任务 a_i,依消息就绪时刻的非增次序选择 $IMP(a_i)$ 中的任务结点 a_u,然后按照任务 a_u 能获得最早起始时刻为目标选择处理器号,同时也确定了它的执行时刻,最后,参照上面定义的任务优先级调度 G 中其余的结点,这些结点的执行对关键路径上的任务没有影响.图3显示了LS2算法对图1的调度结果,很明显,调度长度比图2少1.

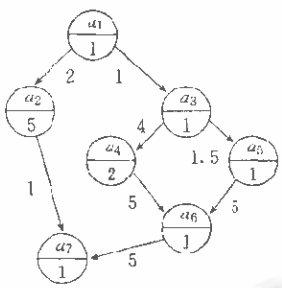

图1 示例任务图

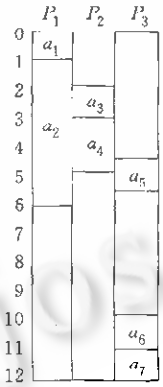

图2 LS1所求得的Gantt图

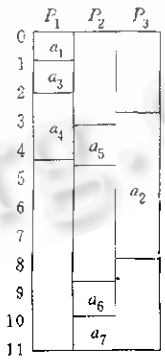

图3 LS2求得的Gantt图

3 对目标处理器选择规则的改进

通过对图2和图3的分析观察可以发现,不管把任务 a_6 放到哪个处理器上执行,由于任务 a_4 与 a_5 到任务 a_6 的通信量都比较大,而 a_4 与 a_5 又放在不同的处理器上,所以 a_6 不可能有较早的起始执行时刻.但是,如果把任务 a_4 与 a_5 分配到同一处理器上执行,如图4所示,这时反而能获得较短的调度长度,调度长度为8.分析其原因可以发现,在上面的启发式调度算法中,主要基于以每个当前任务 a_i 具有最早的起始执行时刻为目标来选择处理器,其目的就是为了使该任务的后继结点有可能尽早开始执行,以此类推,最终目标就是使出口结点能尽早执行,从而使调度长度较小.

但是,局部最优调度某任务 a_i 使其具有最早起始执行时刻这种思想忽略了这样一个事实,即 $IMS(a_i)$ 的其他已分配直接前趋与 a_i 一起,对 $IMS(a_i)$ 的最早起始执行时刻是相互作用、相互影响的.有时在某些条件下,如

果局部地以当前任务 a_r 具有最早起始执行时刻来选择处理器,反而使得 $IMS(a_r)$ 中的某些结点不能尽早执行.例如图 2,如果在给任务 a_5 分配处理器 P_3 时,再进一步测试如果把 a_5 分配到与 a_4 相同的处理器上时,能否使它们的共同直接后继结点 a_6 的起始执行时刻提前.很明显,如果把 a_5 放到与 a_4 相同的处理器上,与把 a_5 分配到处理器 P_3 相比,虽然 a_5 晚 0.5 个时间单位开始执行,违背了以当前任务具有最早起始执行时刻为目标选择处理器的规则,但它有可能使得 a_4,a_5 到 a_6 的通信延迟都为零(只要把 a_6 分配到与 a_4,a_5 相同的处理器上即可),因而把 a_6 的起始执行时刻提早到 6,并且只需要两个处理器就能获得这个调度长度,所以这时就不能单以 a_5 具有最早起始执行时刻来选择处理器.

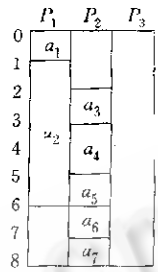

图4　选择策略改进后的调度结果

既然局部地以当前任务具有最早起始执行时刻为目标来分配处理器并不能获得较小的调度长度,那么到底在什么样情况下,我们才考虑把同为某一结点的一些直接前趋任务分配到同一处理器上呢?还是以图 2 与图 4 为例来说明,以 a_4,a_5 和 a_6 这 3 个任务的局部小范围调度而言,如果不断增加 a_5 的计算时间,那么图 4 的调度长度就会相应地增加,而图 2 的调度长度不变,因此,把 a_5 与 a_4 分配到同一处理器上所获得的收益也相对减小.当 a_5 的计算时间增加到 5 时,图 4 中把 a_5 与 a_4 分配到同一处理器上所获得的调度长度也为 12,与图 2 相同.如果 a_5 的计算时间大于 5,这时图 4 中把 a_5 与 a_4 分配到同一处理器上所获得的调度长度将大于 12.

如果 a_5 的计算时间保持不变,而不断减小 a_4 到 a_6 的通信时间,也会使图 2 的调度长度不断减小,而图 4 的调度长度不变.当 a_4 到 a_5 的通信时间减小到 1 时,图 2 的调度长度也为 8,与图 4 相同.同样地,如果考虑减少 a_5 到 a_6 的通信时间,也会使得图 2 的调度长度不断减小.与减小 a_4 到 a_6 的通信时间所不同的是,这时应把 a_6 分配到与 a_4 相同的处理器上.

一般地,在选定使 a_r 具有最早起始执行时刻的处理器 $p(a_r)$ 以后,还要进一步测试与 a_r 具有相同后继 a_i 的其他已分配的直接前趋,从中选择其执行处理器不为 $p(a_r)$(即 $p(a_j)\neq p(a_r)$),并且到 $p(a_r)$ 具有最大消息就绪时刻的任务 a_j,然后检查是否满足下面的条件:

$$est(a_r,p(a_j))+w(a_r)<ect(a_j,p(a_j))+d_{ji}. \tag{3}$$

如果满足上式条件,那么很明显,把 a_r 分配到 $p(a_j)$ 上时,就有可能使它们共同的直接后继结点 a_i 的起始执行时刻提前.

定理 1. 设一个任务结点 a_u 的直接前趋集为 $\{a_{u_1},a_{u_2},\ldots,a_{u_k}\}$,$a_{u_r}$ 是使 $\max\{(ect(a_{u_j},p(a_{u_j}))+d_{u_ju_0})\mid j=1,2,\ldots,k\}$ 取得最大值的结点,其中 $u_0=u$.如果任给 $y=1,2,\ldots,k$ 且 $y\neq r$,满足 $d_{u_ju_0}\leqslant w(a_{u_r})-(est(a_{u_y},p(a_{u_y}))-est(a_{u_r},p(a_{u_r})))$,那么把任务 a_u 分配到 $p(a_{u_r})$ 上,这样总能使任务 a_u 具有最早的起始执行时刻 T_u,$T_u=\max\{ect(a_{u_r},p(a_{u_r})),(ect(a_{u_j},p(a_{u_j}))+d_{u_ju_0})\mid j=1,2,\ldots,k,且\ j\neq r\}$.

证明:可分两种情形来说明.

(1) 如果上式中的 T_u 是由 $ect(a_{u_r},p(a_{u_r}))$ 取得,那么,$ect(a_{u_r},p(a_{u_r}))\geqslant(ect(a_{u_j},p(a_{u_j}))+d_{u_ju_0})\mid j=1,2,\ldots,k,且\ j\neq r)$,也就是说,在 a_{u_r} 执行完以前,a_u 的所有其他直接前趋任务的消息可以传送到处理器 $p(a_{u_r})$ 上,所以,任务 a_u 不可能有比 $ect(a_{u_r},p(a_{u_r}))$ 更早的起始执行时刻.

(2) 如果上式中的 T_u 是由 $\max\{(ect(a_{u_j},p(a_{u_j}))+d_{u_ju_0})\mid j=1,2,\ldots,k,且\ j\neq r\}$ 取得,不妨设 $j=q$ 时取得最大值,那么可以通过两种途径来减小 T_u.

(2.1) 把 a_u 分配到处理器 $p(a_{u_q})$ 上,这时,$T_u=\max\{ect(a_{u_q},p(a_{u_q})),(ect(a_{u_j},p(a_{u_j}))+d_{u_ju_0})\mid j=1,2,\ldots,k,且\ j\neq q\}$.根据条件,$ect(a_{u_r},p(a_{u_r}))+d_{u_ru_0}$ 具有最大值,所以

$$ect(a_{u_r},p(a_{u_r}))+d_{u_ru_0}\geqslant ect(a_{u_q},p(a_{u_q}))+d_{u_qu_0}>ect(a_{u_q},p(a_{u_q})).$$

因此,这时,$T_u=ect(a_{u_r},p(a_{u_r}))+d_{u_ru_0}$,即反而增大了 a_u 的起始执行时刻.

(2.2) 把任务 a_{u_q} 与 a_{u_r} 放到同一处理器上,根据条件

$$d_{u_qu_0}\leqslant w(a_{u_r})-(est(a_{u_y},p(a_{u_y}))-est(a_{u_r},p(a_{u_r}))),\quad 其中\ y=1,2,\ldots,k\ 且\ y\neq r,$$

可得 $$est(a_{u_r},p(a_{u_r}))+w(a_{u_r})\geqslant est(a_{u_q},p(a_{u_q}))+d_{u_qu_0},$$

在不等式两边同时加上 $w(a_{u_q})$，可得

$$est(a_{u_r},p(a_{u_r}))+w(a_{u_r})+w(a_{u_q})\geqslant est(a_{u_q},p(a_{u_q}))+w(a_{u_q})+d_{u_qu_0},$$

即 $$ect(a_{u_r},p(a_{u_r}))+w(a_{u_q})\geqslant ect(a_{u_q},p(a_{u_q}))+d_{u_qu_0}.$$

上式表明，把任务 a_{u_q} 与 a_{u_r} 放到同一处理器上并不会减少 T_u. □

定理1给出了在什么条件下使用以“当前任务具有最早起始执行时刻”为目标选择执行处理器才比较有效，而当拥有相同后继结点的任务间满足条件(3)时，应以“当前任务的后继结点具有最早起始执行时刻”为目标来选择执行处理器.

4 结 论

目前，基于任务静态优先级的启发式任务调度算法都是以“当前任务具有最早起始执行时刻”为目标选择执行处理器，但在某些条件下，这反而会引起后继结点的延迟执行. 本文在详细分析讨论了该种调度策略的基础上，指出了以该目标选择执行处理器所存在的不足之处，并提出了以“当前任务的直接后继具有最早起始执行时刻”为目标选择处理器的调度方法，同时给出了相应的约束条件.

参考文献

1 El-Rewini H, Lewis T G, Ali H H. Task Scheduling. New Jersey: PTR Prentice Hall, 1994. 56～78

2 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(增刊)：74～78
(Chen Hua-ping, Lin Hong, Chen Guo-liang. Heuristic task scheduling in parallel distributed computing. Computer Research and Development, 1997, 34(supplement): 74～78)

3 Sevcik K C. Application scheduling and processor allcation in multiprogrammed parallel processing systems. Performance Evaluation, 1994, 19(2/3): 107～140

4 陈华平，李京，陈国良. 并行分布计算中的任务调度问题(1). 计算机科学，1997，24(2)：23～27
(Chen Hua-ping, Li Jing, Chen Guo-liang. Task scheduling in parallel distributed computing (Part 1). Computer Science, 1997, 24(2): 23～27)

5 Ahmad I, Kwork Y K. A new approach to scheduling parallel programs using task duplication. In: Proceedings of the International Conference on Parallel Processing. Los Alamitos: IEEE Computer Society Press, 1994. 326～333

Processor Selection Policy in Heuristic Task Scheduling

CHEN Hua-ping HUANG Liu-sheng

(*National High Performance Computing Center Hefei* 230027)

(*Department of Computer Science and Technology University of Science and Technology of China Hefei* 230027)

Abstract As one of the most fundmental, critical and challengable problems in PDC (parallel distributed computing), task scheduling has great influence on the execution efficiency of PDC. The existing heuristic task schedulings based on static task priorities always use it as processor selection policy to make the current task have earliest start execution time. On the basis of analysis of the mechanism in priority-based heuristic task scheduling, the authors illustrate the drawbacks of the processor selection policy mentioned above, and propose a new processor selection policy, i.e, to make the successor of the current task have the earliest start time, and give the corresponding restraint.

Key words Parallel distributed computing, heuristic task scheduling, processor selection.