

一种基于 A * 算法的多处理机调度算法

高 亮, 赵 冬

(中原工学院 软件学院, 河南 郑州 450007)

摘 要: 提出一种基于 A * (A Star) 思想的启发式算法, 求解多处理机环境下的实时任务调度问题. 将系统中各个处理机与各实时任务的截止期进行综合考虑并进行形式化描述, 建立 A * 算法的估价函数, 实现了实时任务的高效调度. 仿真结果表明, 其实现相对简单, 收敛速度快, 算法空间复杂度和时间复杂度相对较低, 较好地解决多处理机系统中实时任务的调度问题.

关键词: 多处理机; A * 算法; 任务调度; 分布式系统; 启发式算法

中图分类号: TP311

文献标识码: A

文章编号: 1000-7180(2013)08-0128-04

An A-Star Algorithm-based Multiprocessor Task Schedule Method

GAO Liang¹, ZHAO Dong¹

(Software College, Zhongyuan University of Technology, Zhengzhou 450007, China)

Abstract: A heuristic algorithm based on A * principle is proposed for the multiprocessor task schedule. On the basis of comprehensive considerations and formalization definition of the multiprocessor and tasks' deadline, the evaluation function of A * algorithm is constructed thereby, achieving efficient schedulment of the tasks. The simulation results show that this algorithm is easy to realized and can get the final result quickly with the lower space complexity and time complexity.

Key words: multiprocessor; A-Star algorithm; task schedule; distributed system; heuristic algorithm

1 引言

在分布式系统中, 多个处理机能够并行处理多个任务, 以提高系统的整体效率. 但多处理机上的任务调度是一个 NP 问题, 找出任务调度的最优方案往往需要耗费大量的系统资源^[1-3]. 目前一般采用贪婪算法、遗传算法、最短路径算法等各类算法来得到一个解决方案, 以在系统的资源耗费与整体性能之间取得一个平衡. 但这些算法或者是无法保证所有的实时任务能够在截止时间前完成^[4], 或者是耗费太多的时间.

而 A * 算法作为一种启发式算法, 在选取较好的估价函数后, 能够有效地解决静态图中的最短路径问题^[5], 本文提出一种基于 A * 算法的多处理机动态调度算法, 首先将处理机、实时任务及相关的子

任务进行形式化描述, 并定义调度状态来描述系统的调度过程, 把多处理机环境下的实时任务调度问题转换成调度状态的转移问题, 从而可以利用现有的 A * 算法来得到一个保证各个实时任务截止期的多任务调度方案. 仿真实验证明了算法的有效性.

2 系统和任务模型

在一个分布式系统中可能包含多个处理机与多个实时任务, 每一个任务可以分解为若干个子任务, 这些子任务能够分配在不同的处理机上运行, 最后一个子任务完成的时间, 必须满足截止时间的要求. 处理机及子任务等相关概念的形式化定义如下.

定义 1 处理机. 一个处理机 P 可以用二元组 $\langle pID, pC \rangle$ 来描述. 其中 pID 表示处理机的 ID 号, pC 表示这个处理机的处理能力.

收稿日期: 2012-10-18; 修回日期: 2012-12-07

基金项目: 河南省基础与前沿技术研究计划项目(122300410048)

定义2 实时子任务. 实时子任务 ST 是指一个可以分配到处理机上运行且不可分割的任务, 可以用三元组 $\langle fID, stID, tQ \rangle$. 其中 fID 是指此子任务所属的父任务编号, $stID$ 是指此子任务的编号, 而 tQ 则指此子任务运行时所需要的资源.

定义3 实时子任务的运行时间. 一个实时子任务 ST 在分配处理机 P 后, 其运行时间可由 ST . tQ 与 P . pC 确定, 即运行时间为这两个参数的函数, 记作 $t(ST, P, pC)$.

定义4 实时子任务之间的先行关系. 对于属于同一个实时任务的两个实时子任务 STa 与 STb , 如果 STb 需要在 STa 执行完后才能开始执行, 则定义 STa 为 STb 的先行任务, 记为 $STa \prec STb$.

定义5 实时任务及其子任务划分. 一个实时任务 T 可以描述为 $\langle tID, tSP, tTime \rangle$. 其中 tID 是指此实时任务的编号, 与定义2中的 fID 相对应, $tTime$ 是指此实时任务的截止时间, tSP 则是对此实时任务所包含实时子任务的一个有序划分, 此有序划分可表示为一个集合 $\{tsp_0, tsp_1, \dots, tsp_n\}$, 其中的每一个划分 tsp_i 中包括一个或若干个实时子任务, 其划分方式由下列方法定义:

(1) 对于任意一个 $st_i \in T$, 如果不存在 $st_j \in T$, 使得 $st_j \prec st_i$, 则有 $st_i \in tsp_0$.

(2) 如果 $tsp_0, tsp_1, \dots, tsp_k$ 的元素均已确定, 则对于某个 st_i , 如果有 $\forall st_j (st_j \prec st_i \cap st_j \in \bigcup_{i=0}^k tsp_i)$ 成立, 则 $st_i \in tsp_{k+1}$.

定义6 实时任务的运行态. 一个实时任务的运行态 TR 是指一个实时任务 T 的运行状态, 可以描述为 $\langle tID, tSP, tTime, k \rangle$, 其中 tID 是指此实时任务的编号, $tTime$ 为截止时间与当前时间的差值, k 为一个整数 ($0 \leq k \leq n$), tSP 满足如下条件:

(1) TR . tSP . tsp_0, TR . tSP . $tsp_1, \dots, TR$. tSP . tsp_{k-1} 均为空;

(2) TR . tSP . $tsp_k \subseteq T$. TSP . tsp_k ;

(3) TR . tSP . tsp_{k+1}, TR . tSP . $tsp_{k+2}, \dots, TR$. tSP . tsp_n 分别与 T . tSP . tsp_{k+1}, T . tSP . $tsp_{k+2}, \dots, T$. tSP . tsp_n 相等.

定义7 处理机的运行态. 一个处理机的运行态 PR 可由三元组 $\langle pID, st, time \rangle$ 来描述, 其中 pID 是指此处理机的编号, st 是指当前正运行于此处理机的子任务, $time$ 是指此子任务执行完还需要的剩余时间, 如果当前并没有分配子任务, 则 st 与 $time$ 均赋值为 null.

定义8 调度状态. 一个调度状态 $State$ 是指在某个时间点的系统运行状态, 可由 $\langle TRS, PRS \rangle$ 来描述, 其中 TRS 为当前系统中所有实时任务运行态集合, PRS 则为当前系统中所有处理机运行态的集合.

定义9 实时任务的截止状态. 对于实时任务 T , 如果调度状态 $State$ 满足: 对于任意一个 $st_i \in TR$. TSP . tsp_n , 都存在一个处理机运行态 $PR_j \in State$. PRS , 使得 PR_j . $st = st_i$, 则称此调度状态 $State$ 为实时任务 T 的截止状态. 并且定义完成时间 $State_{T, time} = \max_{j=1}^n (State, PRS, PR_j, time)$.

实时任务的截止状态定义的是此实时任务在执行过程中的一个特殊状态, 在这个状态中, 实时任务的所有子任务或者已经执行完毕, 或者正在执行, 并在 $State_{T, time}$ 后任务执行结束.

定义10 调度状态之间的转移及转移代价. 设 S_1 为一个调度状态, 则 S_1 可以以如下方式进行状态转移:

(1) 如果 S_1 . PRS 中, 某个处理机运行状态 PR_i 的子任务为空, 则 S_1 可以转移到状态 S_2 , 设 st 为 S_1 . TRS . TR_j . tSP . tsp_k 中尚未开始运行的任务, 则 S_2 . PRS . PR_i . $st = st$, S_2 . PRS . PR_i . $time = t(st, P_i)$; S_2 的其他元素与 S_1 相同.

(2) 设 S_1 . PRS . PR 中 $time$ 最小的那个处理机状态为 PR_i , 则 S_1 可以转移到状态 S_2 , 其中 S_2 . PRS . PR_i . st 与 S_2 . PRS . PR_i . $time$ 均设置为空, 其他处理机状态中的 $time$ 修改为 $time - S_2$. PRS . PR_i . $time$ (如果为零, 就设置为空), 对于 $time$ 为空的处理机状态, 假设此处理机上运行的子任务为 st , 对应的父任务编号为 j , 从 S_2 . TRS . TR_j . tSP . tsp_k 中删去 st , 如果 S_2 . TRS . TR_j . tSP . $tsp_k = \emptyset$, 则 $k++$; S_2 的其他元素与 S_1 相同.

调度状态转移的第一种情况描述了在某个处理机空闲时, 将一个实时子任务调度于这个处理机上运行的情形, 在理想状态下, 这种调度并不耗费时间, 因此转移代价为零. 调度状态的第二种情况描述了耗时最少的实时子任务运行完成的情形, 此时这个处理机将转为空闲状态, 转移代价即为这个实时子任务的运行时间.

3 基于A*算法的多处理机调度算法

3.1 多处理机调度算法中的转移代价预估函数设置在对系统的调度状态及其转移进行了形式化描

述后,多处理机上实时任务的调度过程可以看成是一个从初始状态向最终状态转移的过程. 设实时任务为 $T_1, T_2, \dots, T_n$, 其对应的截止时间满足如 $T_1.time < T_2.time < \dots < T_n.time$, 则调度过程可描述为: 从初始状态 $State_0$ 出发(其中所有的实时任务均处于初始态, 所有的处理机运行态均未分配实时子任务), 向最终状态 $State_{end}$ (其中所有的实时任务运行态均为空, 所有的处理机运行态均为空)转移, 而且在转移过程中, 经过各个实时任务 T_i 的截止状态 $State_{Ti}$ 的转移代价 $Cost_i$ 必须满足 $Cost_i \leq T_i.time - State_{Ti}.time$. 这样就将多处理机上的实时任务调度问题, 转换成了一个寻路问题, 从而可以采用 A* 算法来进行解决.

在 A* 算法中, 需要定义当前状态与下一状态的转移代价, 以及当前状态与最终状态的转移代价估计函数, 当前状态与下一状态的转移代价 $g()$ 在定义 8 中已经进行了说明, 而由上述的分析可知, 对于当前状态与最终状态的转移代价估计函数, 不仅仅是考虑到当前状态与最终状态的转移代价, 还需要综合考虑转移过程中各个截止状态的转移代价, 因此对于当前状态 $State_{nw}$, 设其已经经过了 $State_{T1}$ 至 $State_{Ti}$ 的截止状态, 即已经运行完成了实时任务 $T_1, T_2, \dots, T_i$, 记 $Cost$ 为初始状态 $State_0$ 到当前状态 $State_{nw}$ 的总转移代价(即已运行时间), 当前状态至每个截止状态的转移代价预估函数 $k(n)$ 定义如下:

$$k(0) = k(1) = \dots = k(i) = 0;$$

$$k(i+1) = \frac{1}{n} \sum_{j=1}^n State_{nw}. PRS. PR_j. time + t(\sum_{j=1}^m MAX(TR_{i+1}. TSP. tsp_j. tQ), \frac{1}{n} \sum_{j=1}^n P_j. pC).$$

式中, n 为处理机的个数, $\frac{1}{n} \sum_{j=1}^n State_{nw}. PRS. PR_j. time$ 为当前系统中处理机上运行的实时子任务运行完的平均时间, $t(\eta \sum_{j=1}^m MAX(TR_{i+1}. TSP. tsp_j. tQ), \frac{1}{n} \sum_{j=1}^n P_j. pC)$ 表示实时任务 T_{i+1} 剩余子任务执行完毕的估计时间, 其中第一个参数 $\eta \sum_{j=1}^m MAX(TR_{i+1}. TSP. tsp_j. tQ)$ 表示剩余子任务的最大资源需求量, η 为调整系数, $\frac{1}{n} \sum_{j=1}^n P_j. pC$ 表示处理机的平均处理能力.

当前状态至每个截止状态的转移代价预估函数 $k(n)$ 定义以后, 当前状态至最终状态的转移代价预

估函数可以定义如下: $f(n) = Cost + \sum_{j=1}^n k(j) * \frac{T_n.time - Cost}{T_j.time - Cost}$,

式中, $k(j)$ 表示当前状态与第 j 个任务对应截止状态的转移代价, $\frac{T_n.time - Cost}{T_j.time - Cost}$ 则表示当前时间点第 j 个任务的紧急程度.

3.2 算法描述

(1) 初始化. 在全局维护两个列表: OPEN 列表和 CLOSED 列表. OPEN 列表存放本次搜索中待转移的状态, 初始化时, 其中只有一个调度状态 $State_0$, CLOSED 列表存放本次调度中已转移的状态, 初始化为空.

(2) 执行.

a. 将 OPEN 列表中的第 1 个调度状态 $State$ 移出, 放入 CLOSED 列表中;

b. 若 $State$ 为最后一个任务的截止状态, 则跳出步骤 2;

c. 从 $State$ 开始转移, 生成 $State$ 的可转移状态集 M . 对于 M 中的每一个转移状态, 计算其 $f(n)$ 值, 并进行判断:

若该调度状态在 OPEN 中, 且新 $f(n)$ 值 $>$ 旧 $f(n)$ 值, 则更新列表中该点的 $f(n)$ 值, 父状态指向 $State$;

若该调度状态在 CLOSED 中, 且新 $f(n)$ 值 $>$ 旧 $f(n)$ 值, 则更新列表中该调度状态的 $f(n)$ 值, 父状态指向 $State$, 并将该调度状态从 CLOSED 列表中移出, 重新放入 OPEN 列表中;

若该调度状态既不在 OPEN 也不在 CLOSED 列表中, 则将此调度状态放入 OPEN 列表中, 父状态指向 $State$.

d. 按 $f(n)$ 的递增顺序对 OPEN 列表重新排序.

e. 返回步骤 a.

(3) 执行结束. 通过状态 $State_{end}$ 的父状态指针, 在 CLOSED 表中回溯产生所求路径, 即可获得解决方案.

3.3 算法复杂度分析

3.3.1 空间复杂度分析

本算法中, 除了各调度状态自身维护的数据以外, 全局中用到两个列表: OPEN 和 CLOSED 表. 设处理机数为 n , 实时任务数为 m , 平均划分数为 k , 则 OPEN 和 CLOSED 表的最大长度不超过 nm/k . 因此, 算法的空间复杂度为 $O(nm/k)$, 本算法具有

良好的空间性能。

3.3.2 时间复杂度分析

由上面的算法描述可以看出,算法的主要时间耗费在计算各转移状态的估价函数时与步骤2的第d步,即对OPEN中的调度状态进行排序,但各转移状态的估价函数计算只耗费CPU运行时间,并且只需要有限次的计算,对整个算法的效率影响可以忽略。而对于OPEN表的最大长度为 nm/k ,因此排序的时间复杂度为 $O(nm * \log(nm/k)/k)$,这也是算法的整体时间复杂度。

4 实验设计与分析

为验证算法的性能,我们搭建了包括4台服务器的多处理机模拟系统,服务器配置3.0 GHz双核CPU,8 GB内存及2 TB硬盘,每台服务器可以模拟多个处理机。对于调度算法的执行效果,我们采用实时任务的平均结余时间与正常完成率这两个指标来进行衡量。

实验在每台服务器上模拟若干个处理机,测试了调度多个实时任务的算法执行时间,并与贪婪算法及通用的最短路径算法进行比较,结果如图1所示,相应平均结余时间与正常完成率的比较结果如表1所示。

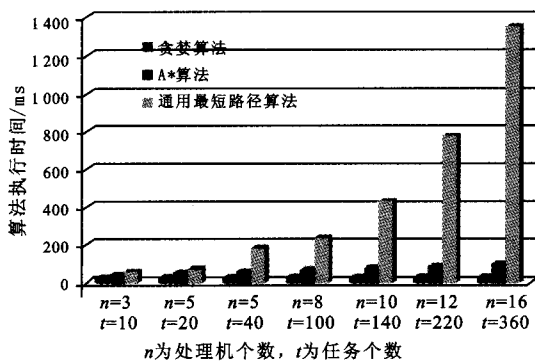


图1 算法执行时间比较

由实验结果可以看出,在执行时间方面,贪婪算法由于每次调度时都只是简单的比较时间的紧急程度,因此执行时间基本维持在一个常数时间,最短路径算法每次都需要对全局的状态进行比较,因此耗时较多,而本文所采用的A*算法,执行时间虽然会随着实时任务数量的增加与处理机的数量减少而线性增长,但线性系统在不断变小。而在结果方面,最短路径算法和本文所采用的A*算法都能保证所有的实时任务都在截止时间之前完成,同时平均结余时间方面,最短路径算法略优于A*算法,而贪婪算法在实时任务数量增加或处理机数量较小时,有可能会发生某些实时任务无法在截止时间之前完成,这会影响到系统的稳定性,因而实用性不强。

表1 平均结余时间及正常完成率比较

	平均结余时间/s			正常完成任务个数		
	贪婪算法	A*算法	最短路径算法	贪婪算法	A*算法	最短路径算法
$n=3, t=10$	6	8.6	9.2	10	10	10
$n=5, t=20$	14.8	21	22.6	19	20	20
$n=5, t=40$	38.1	66.3	72.3	37	40	40
$n=8, t=100$	65.7	83.4	81	94	100	100
$n=10, t=140$	86.6	95.2	97	132	140	140
$n=12, t=220$	133.5	166	168	210	220	220
$n=16, t=360$	205.1	398	413	345	360	360

5 结束语

A*算法将每个任务的截止时间综合起来进行考虑,能够保证实时任务在截止时间之前完成,优化处理机资源配置,较好地解决多处理机系统中实时任务的调度问题。其实现相对简单,收敛速度快,算法空间复杂度和时间复杂度相对较低。进一步研究将集中于如何改进算法,使之能够适应更多的环境,使算法更加实用化。

参考文献:

- [1] Palencia J, Harbour M. Offset-based response time analysis of distributed system's scheduled under EDF [C]//proceedings of the 15th Euro micro Conference on Real Time Systems. Porto, Portugal: IEEE, 2003:3-12.
- [2] Shiyuan J, Guy S, Damla T. A performance study of multiprocessor task scheduling algorithms [J]. The Journal of Supercomputing, 2008, 43(1):77-97.
- [3] Rahmani A M, Vahedi M A. A novel task scheduling in multiprocessor systems with genetic algorithm by using elitism stepping method [J]. Journal of Computer Science, 2008, 7(2):58-64.
- [4] 任钢. 基于进化算法的多处理机任务调度器研究 [D]. 北京:中国科学院, 2001.
- [5] Koenig S, Likhachev M, Furcy D. Lifelong planning A* [J]. Artificial intelligence Journal, 2004, 155(12):93-146.

作者简介:

高亮男, (1981-), 硕士, 讲师。研究方向为分布式计算、网络安全。

赵冬女, (1976-), 硕士, 讲师。研究方向为分布式计算、计算机网络。