一种针对依赖性周期任务的实时多核调度算法*

李凤彬[a]，朱怡安[a,b]，黄姝娟[b]，唐毓毅[a]
（西北工业大学 a. 计算机学院；b. 软件与微电子学院，西安 710072）

摘　要：针对软实时系统中的一类同时具有依赖性与周期性的任务，提出一种基于单行树矩阵（MST）的动态因子均衡调度算法 SMD（schedule on matrix of the single tree and dynamic load factor）。该算法通过对 MST 矩阵的特性进行分析，将任务划分为若干并行集，再综合考虑已执行时间、任务间的依赖关系及任务最早截止时间几个要素，以动态因子的形式对任务进行实时调度。最后，还以证明的形式给出了可充分调动的任务集的充分条件，并以此为基础随机生成了测试任务集，进行了对比实验。实验表明，与文献中现有经典算法相比，新算法使处理器利用率提升近 15%，任务丢失率降低 2%。

关键词：多核调度；依赖关系；多任务；实时系统；调度算法
中图分类号：TP309　　**文献标志码**：A　　**文章编号**：1001-3695(2013)05-1340-05
doi:10.3969/j.issn.1001-3695.2013.05.015

New scheduling algorithm for real-time multiprocessor systems aimed to independent and periodic multi-tasks

LI Feng-bin[a], ZHU Yi-an[a,b], HUANG Shu-juan[b], TANG Yu-yi[a]
(*a. School of Computer Science & Engineering, b. School of Software & Microelectronics, Northwestern Polytechnical University, Xi'an* 710072, *China*)

Abstract: With the aim to work out a kind of task, which is meanwhile based on the dependence and period, this paper proposed a schedule algorithm based-on matrix of the single tree and dynamic load factor, named SMD. SMD dynamically aggregated jobs into several parallel sets by analyzing the feature of the MST(matrix of the single tree). Then, SMD scheduled the real-time tasks with judging the value of the dynamic factor by compromising executed-time, task dependence and deadline. Finally, it gave a proof of how to generate an appropriate task set and do an experiment based on this proof. The experimental results demonstrate that compared with the classic algorithm, SMD enhances CPU-utilization by nearly 15% and reduce the miss-rate by 2% respectively.

Key words: multi-core scheduling; dependence-based; multi-task; real-time system; scheduling algorithms

0　引言

随着实时应用范围的加大和复杂度的提高，多处理器系统由于其高性能及经济性，逐渐成为处理这种复杂应用的有效手段，而实时多处理器系统的调度算法则成为一个重要研究课题。其中，文献[1]研究并提出了经典静态调度算法中的速率单调算法（RM 算法），并给出了在单核下任务可调度的证明。随着实时系统开放、灵活多变的发展趋势，实时调度算法的研究逐渐转向了动态、分布式、软实时以及混合调度[2]。目前已被证明的能够成功应用的动态调度算法是最早时间优先（EDF）法[3]和最少空闲时间优先（LLF）法[4]。而针对周期性任务与少量非周期任务混杂情况下的多核调度，Baruah 等人又提出了几种能使处理器利用率较高的算法，典型算法如 GEDF（global earliest deadline first）[5] 以及 Pfair（proportionate fairness）[6]。上述文献针对有依赖性的非周期任务和独立的周期性任务进行了较为深入的研究，并针对不同的上下文情景提出了较为高效的算法，然而针对同时具有依赖关系与周期性的任务却鲜有文献提及。

事实上，本文所述的这种一般性的、具有依赖关系的周期性任务，在工业生产控制及航空航天等众多领域有着极为广泛的应用。例如，航空航天控制领域，需要大量的传感器节点来采集与分析各种环境信息（飞行设备高度信息、经纬度信息、气压与氧含量信息等），这些被采集并处理完成的信息又需要按照一定的顺序与路径最终汇总到主控设备上，从而根据这些汇总信息对整个系统进行优化控制。本文正是基于这种广泛的应用前景，在对比分析单核调度算法与经典多核调度算法的基础上，提出了一种基于单行树矩阵（MST）的动态因子均衡调度算法。该算法通过将 MST 矩阵划分为若干并行集，再综合考虑已执行时间，任务间的依赖关系及任务最早截止时间几个要素，以线性多项式的形式构造了任务动态因子这一参数，并根据这一参数，以优先级的形式对任务集进行调度，相比通用的经典算法，在保持软实时系统低丢失率的同时，提高了多核

收稿日期：2012-09-05；**修回日期**：2012-10-29　　**基金项目**：航天科技创新基金资助项目（2011XR60001）；航空科学基金资助项目（20100753022）；校基础研究基金资助项目（JC20110283）；西北工业大学研究生创业种子基金资助项目（Z2012139）

作者简介：李凤彬（1986-），男，硕士，主要研究方向为嵌入式计算（Kingdom-free@hotmail.com）；朱怡安（1961-），男，教授，博导，主要研究方向为高性能计算、云计算、普适计算；黄姝娟（1975-），女，讲师，博士研究生，主要研究方向为嵌入式计算、普适计算；唐毓毅（1987-），女，硕士，主要研究方向为嵌入式计算.

处理器的利用率。

1　问题背景

1.1　问题相关概念

定义 1　设 T 为具有 $n(n\geqslant 0)$ 个节点的有限集,当 $n>1$ 时,有且仅有一个节点为根节点,其余节点又构成为一个子集 $T_i(i\leqslant n)$,而每一个子集又是一棵单行树。将具有 T 这种性质的有限集称为单行树。

定义 2　将一棵单行树中所含任务节点的个数称为单行树的长度。

定义 3　将每一棵从给定情况中抽象出来的单行树叫做一个任务集合,用 T_i 来表示,其中 i 表示任务集编号;单行树中的每一个节点叫做一个任务(task),用 $T_i(\alpha,e,p,d)$ 来表示,其中 i 表示任务编号,α 表示任务最早到达的时间,e 表示任务执行完成所需的最坏时间,p 表示任务的周期,d 表示任务的死限(deadline)。任务是对待处理的周期数据的一种描述。

定义 4　将周期任务的每次实际调用称为工作(job),用 J_{mn} 来表示,其中 m 表示所述任务编号,n 表示所述任务的第几个工作。

定义 5　将每一个任务中执行时间与其周期的比值称为该任务的利用率,形式化地将其表示为 $U(T_i)=\dfrac{e_i}{p_i}$;将一段时间内,每个处理器核上的实际处理时间与这段时长的比值称为该处理器核在这段时间内的平均利用率,可以形式化地表示为 $\overline{U}(P_i)=\dfrac{E_i}{\Delta t}$,其中 P 表示处理器核,Δt 表示时长,i 表示处理器核的编号,E_i 表示该核在 Δt 时长内真正处理计算所消耗的时长。又由于 $E_i=\sum_{k=0}^{m-1}J_k(P_i)$,故上述处理器平均利用率表示又可写为

$$\overline{U}(P_i)=\frac{\sum_{k=0}^{\varepsilon-1}J_m(P_i)}{\Delta t}$$

其中:ε 表示系统所需处理的任务数;m 表示当前正在处理的任务;$J_m(P_i)$ 表示在 Δt 时长内,第 m 个任务在第 i 个处理器核上的执行时间。

定义 6　将软实时系统中处理器所未能在任务要求死限前完成的任务与总到达任务的比值,称为任务的丢失率。

定义 7　每一棵单行树所构成周期依赖系统的执行过程都可以使用一个矩阵来表示,从而简化了问题的分析难度。将这种矩阵称为单行树矩阵(matrix of the single tree,MST),形式化地将其表示为一个 $m\times n$ 的矩阵:

$$\mathrm{MST}=\begin{pmatrix} J_{00} & \cdots & J_{0n} \\ \vdots & & \vdots \\ J_{m0} & \cdots & J_{mn} \end{pmatrix}$$

其中:J_{mn} 为单行树的各个实际工作;m 为单行树的任务数;n 为任务的实际执行次数,即周期任务 T 实际执行了几个周期。

定义 8　将在任务集本身具有周期依赖关系的条件下仍然可以并行执行的特性的直连线工作集称为一个并行集,并行集中所含工作的数量称为并行集的长度。

定义 9　将任务的权值与任务最坏截止时间、已执行时间之和的比值叫做任务的动态因子,形式化地表示为 $\gamma_i=\dfrac{I}{d_i+c_i}$,其中 γ 表示任务动态因子,c 表示任务已完成的时间,i 表示任务编号,I 表示对应的权值(权值为任务号加 1,如任务号为 0,则任务权值为 1)。此时,任务的表示增加了一个动态因子属性,即一个任务又可以更详细地表示为 $T_i(\alpha,e,p,d,\gamma,c)$。

引理 1　在一定条件下,针对非周期有依赖关系的多任务的调度算法与针对周期依赖关系的多任务的调度算法是等价的。

证明　设非周期调度算法为 A^{α},周期性调度算法为 A^{β},则考虑极限情况,当任务集中只有一个任务时,周期性任务的执行可视为等执行时间等周期的非周期任务的多次执行。故 $A^{\alpha}\ni A^{\beta}$,即在此条件下,A^{β} 成为了 A^{α} 的真子集。

证毕。

定理 1　设 $\tau=\{T_0,T_1,\cdots,T_{m-1}\}$ 为一非周期有依赖关系的任务集,则 $\min_p(\tau)\geqslant\sum_{i=0}^{m-1}e_i$ 是引理 1 成立的充分条件。

证明　当 $\min_p(\tau)\geqslant\sum_{i=0}^{m-1}e_i$,即任务集 τ 中周期最短的任务的周期都不小于当前任务集中所有任务执行时间之和时,显然,不管采用何种调度方式,在当前任务集全部任务执行完之前都不会有其他任务集进入,故此时对周期任务与非周期任务的调度是等价的。

证毕。

定理 2　$\mathbb{Q}_{\mathrm{deadline}}=\{T_n\mid d_{\max},\cdots,T_1\mid d_{\min}\}$ 是给定任务集中各任务按照死限降序排列的一个序列,$n\sim 1$ 为任务按从大到小递减的整数编号,则 $d_{\max}\geqslant\sum_{i=1}^{n}e_i$,$d_k\geqslant\sum_{i=j}^{k}e_i(j,k\in n)$,$\cdots$,$d_{\min}\geqslant\sum_{i=1}^{n-(n-1)}e_i$,总能保证任务在死限前被调度完成。

证明　当任务集序列满足条件 $d_{\max}\geqslant\sum_{i=1}^{n}e_i$,$d_{\max-1}\geqslant\sum_{i=1}^{n-1}e_i$,$\cdots$,$d_{\min}\geqslant\sum_{i=1}^{n-(n-1)}e_i$ 时,总能保证 $d_k\geqslant\sum_{i=j}^{k}e_i(j,k\in n)$,即在每一个任务的死限到达前,总有足够的时间来执行该任务和其前驱任务,故无论采用何种调度方式,总能保证任务在死限前被调度完成。

证毕。

引理 2　当 $n\geqslant 2$ 时,最坏情形下可调度的总利用率阈值为

$$U^*(n,\beta_{RM})=(n\beta_{RM}+1)[2^{\frac{1}{\beta_{RM}+1}}-1]\tag{1}$$

其中:$\beta_{RM}(\alpha)$ 表示可由 RM 调度算法在单个处理器上可调度的利用率为 α 的任务数,n 为处理器核数。

证明　参见文献[7]。

证毕。

定理 3[8]　由引理 1 可知,实时周期任务集合按 RM 可调度条件配置在 n 个并行处理器上的充分条件为

$$U\leqslant U^*(n,\beta_{RM})\left\{1+\alpha_0\frac{(n-1)(n-1+m)}{[n-1+(1+\alpha_0)m]^2}\right\}\tag{2}$$

其中:$\alpha_0=\sqrt[m]{\dfrac{u_1}{u_2}}-1$,$m$ 为任务集中的任务数量。

证明　略,参见文献。

证毕。

1.2　问题模型

1.2.1　问题的简化

根据问题的特点,可将问题抽象成一棵具有若干个节点的树状模型。如图 1 所示,以六个节点、深度为 3 的一棵树为例,圆圈表示任务,箭头表示任务间的依赖关系(箭头指向的任务为依赖任务,另一端为被依赖任务)。根据集合的性质,任何

一棵树都能拆分成若干单行树的集合,如图 2 所示。为了解决类似图 1 这类一般性问题,只需先解决类似图 2 这类的子问题即可,本文所提出的算法即是用来解决图 2 这类子问题的。

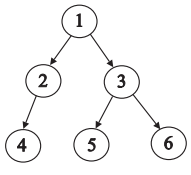

图1 一般性情况

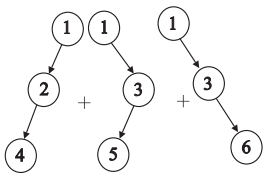

图2 特殊性情况

1.2.2 问题模型特性说明

假设一个具有多核处理器的软实时系统具有 k 个处理器核($k>1$),且这些处理器核是同构的。考虑具有 m 个周期任务的任务集 $\tau = \{T_0, T_1, \cdots, X_{m-1}\}$ 在同构多核处理器上的可部分抢占式的调度问题,本文中有依赖关系的周期任务具有以下特性:

a)只考虑周期任务按序到达且任务周期和死限都为相等正整数的情况,即每个任务 $T_i(\alpha, e, p, d)$ 中,$e \leqslant p$,且 $d = p \in \mathrm{N}$。

b)任务之间是可抢占的且中断后可立即恢复执行,但其中一些任务有前后依赖关联,不考虑同一处理器上任务切换带来的开销及其他一些资源(如变量及缓冲区)调度访问的开销。

c)每个任务 T 一般都包含有无限(或指定个数)的工作,以 J_{mn} 表示第 m 个任务的第 n 个工作。

d)$\overline{U}(P) \xrightarrow{\text{def}} \sum_{i=0}^{k-1} \overline{U}(P_i)$ 表示任务的总平均利用率,其中 k 表示系统中处理器核的个数。显然总平均利用率 $\overline{U}(P) \leqslant k$。

e)以序列 $P = \{P_1, P_2, \cdots, P_k\}$ 来表示含有 k 个具有相同处理能力的同构多核处理器集;同时假定该处理器集内部的核直接的通信开销可以忽略不计。

1.3 调度器模型设计

本文采用对等调度器模型,即不设立专门的中心调度器,而是把每一个处理器核都看做等价的。如图 3 所示,处理器序列中每一个处理器核都可作为选择\调度器。具体来说,当任务集到达时,首先进入选择\调度器的缓冲区中,并将选择\调度器初始化为选择器模式,此时:a)选择\调度器会从处理器序列中依次挑选第一个空闲的处理器并将其选入选择\调度器中;b)选择\调度器会被置为调度器模式,使用已经选择的处理器进行调度运算,并将运算得出的调度结果输入选择\调度器中,且将调度标志位置为“开始状态”;c)选择\调度器会根据 b)中输入的调度方案,将缓冲于选择\调度器的任务分配到相应的处理器上。

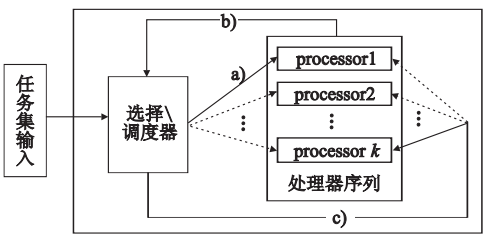

图3 调度器模型设计

2 算法阐述

多处理器环境的任务调度存在全局方案与划分方案两种。在全局调度方案中,实时任务可以在不同的处理器上运行,所有处理器上运行同一种调度算法。任务在执行完之前可以被抢占,并且可以在不同的处理器间迁移。相反,在划分调度方案中,一个任务的所有出现都在同一处理器上执行,全部任务由任务分配算法预先划分到处理器,每个处理器可以运行不同或者相同的单核处理器任务调度算法。这种划分方法主要对任务集进行脱机分配,它利用了分治算法的设计思想,将复杂的问题分解为若干个子问题,分而治之,逐一突破。

划分方案与全局方案相比,具有调度的复杂性低、子问题的针对性强及难度低等优点。然而划分方案的使用,必须依赖于任务集本身具有参数已知、任务优先级确定等前提条件。

因此,结合前文所阐述问题的任务集模型,本文提出了一种将两种方案结合,且利用 MST 与任务动态因子来进行调度的 SMD(schedule on matrix of the single tree and dynamic load factor)算法。下文就利用传统 RM 算法和本文提出的 SMD 算法来对本文问题任务模型进行调度的方案进行进一步的说明。

2.1 基于经典 RM 算法的旧方案

由于在现阶段国内外文献中均未提及本文所述问题模型,所以在当前遇到此类问题时只能使用针对通用问题的算法来进行处理。文献[9]所提出的算法是一种基于 RM 算法的针对有依赖关系的多任务集的多核调度算法。虽然该算法是针对有依赖关系的非周期任务调度问题最优的近似算法之一,但是由于该算法针对的问题不具备周期性特性,所以必须对该算法应用于本问题的可用性加以讨论:

a)当所有任务的周期都足够大时,根据定理 1 可知,在这种情况下,文献提出的算法也可纳入本文所述的问题解决域中。

b)当任务集中有任务的周期并不能满足定理 1 中的条件时,意味着需要考虑在本次的单行树任务还未完全执行完成时下一个周期中新任务的加入情况。虽然该情况是文献所给出的算法未考虑的情况,然而本文通过将周期任务作为一种新的依赖关系来加入到待执行队列中,使得文献提出的 RM 算法也可以处理该类问题。

2.2 基于 SMD 算法的新方案

本文提出的 SMD 算法的核心在于 MST 与任务动态因子。分析 MST 矩阵可知,将矩阵中行标与列标之和相等的元素放在一起可以构成一条与矩阵正对角线平行的直连线,如下方矩阵中所示的 J_{10} 和 J_{01}。

$$\mathrm{MST} = \begin{pmatrix} J_{00} & J_{01} \cdots & J_{0n} \\ J_{10} & & \\ \vdots & \ddots & \vdots \\ J_{m0} & J_{m1} & J_{mn} \end{pmatrix}$$

显然,这些直连线所划分出的工作集合具有在任务集本身具有周期依赖关系的条件下仍然可以并行执行的特性。不难看出,对于任何一个 $m \times n$ 的 MST 矩阵来说,其并行集的长度函数为

$$f(i) = \begin{cases} 1 + f(i-1) & i \leqslant m \\ 1 & i = 1 \\ f(i+1) - 1 & i > m \end{cases}$$

其中:i 为工作的所属任务号。而对于由循环周期性任务构成的 MST 矩阵来说,其并行集长度函数又可进一步简化为

$$f(i) = \begin{cases} 1 + f(i-1) & i \leqslant m \\ 1 & i = 1 \\ m & i > m \end{cases}$$

解以上方程,得其函数拐点对应的函数值为 m,即 m 为最大值。所以,为了尽可能地让可以被执行的工作得到调度,提高处理器核的利用率,SMD 算法将 m 设为首次缓冲区存储工作(job)的容量上限,之后会根据已完成的并行集的数量进行动态调整。

由于本文假定最早截止时间 d 与周期 p 一致,故分析动态因子 γ 可知,γ 的引入事实上是 EDF 算法针对本文问题模型的一种改进。具体来说,由于本文的单行树模型是一种具有自上到下依赖关系的模型,所以在一般情况下尽早处理上方节点任务,必然会增加处理器核的利用率,降低任务丢失率。然而又考虑到 RM 算法本身具有单位时间任务吞吐量大等优势,故根据定义 8 所述,将任务动态因子作为衡量调度优先级的指标。经过分析可知,当一个工作的任务号为 0,且首次被调度执行时,SMD 算法就演变为 RM 算法。然而当任务量激增时,SMD 算法却比 RM 算法更能优先执行单行树上侧任务,从而保证系统的高利用率与低丢失率。SMD 算法的具体步骤和相应的 C 语言伪代码如下所示:

```
/ * input: S is taskset,S[n] is an array of constructing linkabe table,
output: set S[n] is the linktable * /
typedef struct linktable{
Data data;
Task * nextjob;
int num;
} Linktable;
Linktable s[n];
void SMD_Schedule_Agorithm(S){
cmst:construct_MST(s);
/ * construct linkable table * /
for(int x = 0; x<m; x++) {
for (int i=0; i<m ; i++ ){
for (int j =0; j < p ; j++ ) {
if(i+j = = x) s[x] = MST[i][j]; }
}}
sort:SortByLamda(s[n]);//Lamda is dynamic load factor
   / * distribute cpu core * /
   for(int i=0; i<m; i++ ){
      for (int j = 0; j<k ; j++ ){
         if(seq_cpu[i].state = = idle){
            while(nextjob ! = null){
            seq_cpu[i].currentexe = s[i];
               nextjob = s[i].nextjob;}}
      }
   }
Timeforward();
Updata();
if(buffer = = null) goto cmst;
   else goto sort;
}
```

a)对于输入的一个有依赖关系的周期性任务集,先以动态链表的形式将其构建成一个网状的矩阵结构,即在缓冲区中构建 MST 矩阵。

b)根据任务的行列号之和将所构建的 MST 矩阵中的并行集提取出来,以邻接表的结构存储于缓冲区内。邻接表的数组长度为 MST 矩阵的列数。

c)将邻接表中的工作依 γ 值的降序读取,找到处理器核序列中第一个空闲的处理器核,并将该工作分配给该处理器,直到处理器队列无空闲处理器为止。

d)整个系统的时间片向后推移一个时间片周期。

e)将上一个时间片内,经处理器处理后的工作(job)的状态更新,包括 c 和 γ 等参数值,以及邻接表的更新。并将处理器中已执行两个时间片以上的工作重新放回邻接表中。

f)判断缓冲区内还有无工作,若已经没有工作,则尝试读取新的工作,执行步骤 a);否则,执行步骤 c)。

3　仿真实验

本文实验环境是:CPU Dual-Core Pentium® E5800/3.2 GHz, RAM 4 GB,操作系统:Fedoral 14。采用 Codeblock 集成开发环境及 GCC 编译器编程。采用文献[10]中的实验方法,利用随机函数生成不同数量的任务集,并用生成的任务集分别对 SMD 与经典 RM 算法进行测试。由于任务集本身的特性所限,可能出现任务集本身就不可调度,或最好利用率达不到 1、丢失率大于 0 的情况,从而无法准确地对调度算法的性能进行客观评估。为了避免这种情况的发生,对任务集的选取进行了一些限制。

3.1　处理器集和任务集描述

由于本文的算法是针对固定数量的处理器核的调度算法,因此为方便数据统计,仿真实验将处理器核数定为 6(处理器都是同构的且处理能力相同),并将任务集定为 6 组。每组任务集中任务数的取值范围分别设定为 50、100、150、200、250、300。每组任务集中的每一个任务的周期皆为 $T_i \in [1, 1000]$,所有任务都满足死限等于周期且执行时间小于周期的条件。并且在选取任务集的过程中,根据定理 2、3,排除利用率不符合条件的任务集。例如,当随机产生的一组任务集为 $\tau = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$,其利用率分别为 $u_1 = 0.8, u_2 = u_{1/2}, u_3 = u_1/10, u_4 = u_1/8, u_5 = u_1/8, u_6 = u1/8, u_7 = u_1/8$,则由于式(2)的左边 $U = \sum_{i=1}^{7} U_i = 1.62$,右边为 1.43,左边大于右边,所以根据定理 3 可知,此次产生的任务集不能被完全调度,应该排除。

3.2　性能指标

对于一个软实时系统来说,决定其性能优劣的核心指标是任务能够按时完成的数量,完成的数量越多,系统就越可靠,性能就越好。为了分析本文所述任务的丢失率,根据定义 6,使用总任务的丢失率作为评判调度算法实时性的指标。

针对调度算法本身而言,就是要保证在实时系统所规定的丢失率条件下,使得处理器中的每一个核都尽可能地被利用起来。所以根据定义 5,使用处理器的平均利用率这一指标来衡量一个调度算法的调度性能。

本文需要对比的两种算法的输出,正是以这两个指标为核心设计的,整个仿真实验的评价也是针对这两个指标来建立的。

3.3　实验结果及分析

图 4 是根据 3.1 节所构建的 6 组任务集合和 10 核处理器所得到的实验数据。从图中可以直观地看到,由 SMD 算法所造成的任务丢失率的最大值都在 0.15 以下,且整条曲线几乎都位于由文献经典 RM 算法所生成的曲线之下;同时,在任务集中的任务数增大时,SMD 算法所造成的任务丢失率有趋于缓和的态势,而经典 RM 算法的丢失率走势有进一步增加的趋势。

图 5 也是采用 3.1 节中的方法所构建的相同的 6 组任务集及处理器所得到的关于系统整体平均利用率的一组实验数据。分析数据可知,在任务集中任务数不多的情况下,SMD 与经典 RM 算法基本持平甚至略逊于经典 RM 算法。这是由于当任务数量还不大时,任务执行的先后顺序对整体的利用率影响不大,而且由于算法刚刚开始调度时易受任务动态因子的频

繁抖动影响,使得任务被频繁地剥离出处理器核,造成处理器在处理的过程中会产生间隙,同时总任务数较低,也拉低了系统的整体利用率;而随着任务数的逐渐增多,可以很容易地观察到 SMD 算法优先执行被依赖的节点任务的优势得以体现,其利用率稳步上升,当任务集中的任务数达到 300 个时,其任务利用率甚至比经典 RM 算法提高了近 15 个百分点。

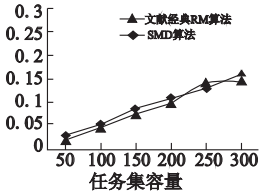

图4 两种算法执行6组任务集后的丢失率

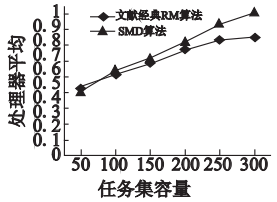

图5 两种算法执行6组任务集后的处理器利用率

4 结束语

本文通过对多核处理器环境下的特殊实时调度问题进行分析研究,从如何提高复杂实时系统中的处理器利用率及降低软实时系统的丢失率入手,提出了一种新的基于 MST 矩阵的及任务动态因子的调度算法。该算法通过对 MST 矩阵的特性进行分析,将任务划分为一个个的并行集,再通过综合考虑已执行时间、任务间的依赖关系及任务的最早截止期几个要素,以动态因子的形式对任务进行实时调度。实验结果表明,该算法能较好地对文中所提出的这种既具有依赖关系又具有周期性的实时任务集进行调度,比之经典 RM 算法,在处理器利用率及任务丢失率两项重要指标上皆有所提升。未来工作的重点在于将单行树的任务集扩展为一般树的形式,以及将一棵树的形式扩展为多棵树的形式,以适应范围更广的应用要求。

参考文献:

[1] LIU C L, LAYLAND J W. Scheduling algorithms for multiprogramming in a hard-real-time environment[J]. Journal of ACM, 1973, 20(1): 46-61.

[2] LEONTYEV H. Compositional analysis techniques for multiprocessor soft real-time scheduling[D]. Chapel Hill: the University of North Carolina, 2010.

[3] CHETTO H, CHETTO M. Some result of the earliest deadline scheduling algorithm[J]. IEEE Trans on Parallel and Distributed Systems, 1989, 15(10): 1261-1269.

[4] MOK A K. Fundamental design problems of distributed systems for the hard-real-time environment[D]. Massachusetts: Massachusetts Institute of Technology, 1993.

[5] BRANDENBURG B B, ANDERSON J H. On the implementation of global real-time schedulers[C]//Proc of the 30th IEEE Real-time Systems Symposium. Washington DC: IEEE Computer Society, 2009: 214-224.

[6] BARUAH S K, COHEN N K, PLAXTON C G, *et al*. Proportionate progress: a notion of fairness in resource allocation[J]. Algorithmica, 1996, 15(6): 600-625.

[7] 杨根科,吴智铭. 周期实时任务在多处理器下的可调度条件[J]. 上海交通大学学报, 2004, 38(9): 1597-1600.

[8] LOPEZ J M, DIAZ J L, GARCIA D F. Minimum and maximum utilization bounds for multiprocessor RM scheduling[C]//Proc of Euromicro Workshop on Real-time Systems. [S. L.]: IEEE Press, 2001: 67-75.

[9] 黄金贵,李荣珩. 独立多处理机任务静态调度问题的近似算法[J]. 软件学报, 2010, 21(12): 3211-3219.

[10] 王涛,刘大昕. 多处理器单调速率任务分配算法性能评价[J]. 计算机科学, 2007, 34(1): 272-277.

(上接第 1311 页)

参考文献:

[1] LAWRENCE R D, ALMAS G S, KOTLYAR V, *et al*. Personalization of supermarket product recommendations[J]. Data Mining and Knowledge Discovery, 2001, 5(1/2): 11-32.

[2] McLAUGHLIN M R, HERLOCKER J L. A collaborative filtering algorithm and evaluation metric that accurately model the user experience[C]//Proc of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM Press, 2004: 329-336.

[3] 李春,朱珍民,叶剑,等. 个性化服务研究综述[J]. 计算机应用研究, 2009, 26(11): 4001-4005.

[4] 王卫平,王金辉. 基于 Tag 和协同过滤的混合推荐方法[J]. 计算机工程, 2011, 37(14): 35-36.

[5] 李改,李磊. 一种解决协同过滤系统冷启动问题的新算法[J]. 山东大学学报:工学版, 2012, 42(2): 11-17.

[6] 余小高,余小鹏. 基于隐式评分的推荐系统研究[J]. 计算机应用, 2009, 29(6): 1585-1589.

[7] JAMALI M, ESTER M. Using a trust network to improve top-*N* recommendation[C]//Proc of the 3rd ACM Conference on Recommender Systems. New York: ACM Press, 2009: 181-188.

[8] LI Rui-feng, ZHANG Yin, YU Hai-han, *et al*. A social network-aware top-*N* recommender system using GPU[C]//Proc of ACM/IEEE Joint Conference on Digital Libraries. 2011: 287-296.

[9] ELEFTHERIOS T, YANNIS M. Product recommendation and rating prediction based on multi-modal social networks[C]//Proc of the 5th ACM Conference on Recommender Systems. New York: ACM Press, 2011: 61-68.

[10] MA Hao, ZHOU Deng-yong, LIU Chao. Recommender systems with social regularization[C]//Proc of the 4th ACM International Conference on Web Search and Data Mining. New York: ACM Press, 2011: 287-296.

[11] ADOMAVICIUS G, TUZHILIN A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions[J]. IEEE Trans on Knowledge and Data Engineering, 2005, 17(6): 734-749.

[12] SEN S W. Nurturing tagging communities[D]. Minnesota: University of Minnesota, 2009.

[13] JEH G, WIDOM J. SimRank: a measure of structural context similarity[C]//Proc of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2002: 538-543.

[14] HERLOCKER J L, KONSTAN J A, TERVEEN L G, *et al*. Evaluating collaborative filtering recommender systems[J]. ACM Trans on Information Systems, 2004, 22(1): 5-53.