ISSN 1000-1239/CN 11-1777/TP
46(2): 302-309, 2009

FPTS:一种任务间存在共享资源时的抢占阈值调度算法

贺小川　　贾　焰
(国防科学技术大学计算机学院网络与信息安全研究所　长沙　410073)
(xiaochuanhe@gmail.com)

FPTS: A Fixed-Priority Preemption Threshold Scheduling Algorithm in the Presence of Resources Sharing

He Xiaochuan and Jia Yan
(*Institute of Network Technology and Information Security, College of Computer, National University of Defense Technology, Changsha* 410073)

Abstract Fixed-priority with preemption threshold (FPPT) is an important form of real-time scheduling algorithm, which fills the gap between fixed-priority preemptive (FPP) and fixed-priority non-preemptive (FPNP). FPPT can prevent tasks from unnecessary task preemption, reduce the additional memory usage, and improve the schedulability of task set. In real-world real-time applications, access to exclusively-shared resources is a very common operation, and therefore shared resources management in FPPT is necessary. The correlations between tasks, resulting from the shared resources, have a great influence on the priority assignment and preemption threshold assignment of task set. However, current research on FPPT real-time scheduling techniques focuses on the time guarantees of independent real-time tasks rather than the complexity coming from exclusively-shared resources. Stack resource protocol (SRP) is well-known resource access and control protocol in real-time systems, and has many nice features such as deadlock avoidance, earlier blocking, shared run-time stack and so on. Proposed in this paper is a new FPTS scheduling paradigm, which integrates FPPT with SRP, including the new critical instant, preemption threshold assignment and appropriate schedulability analysis, based on response time analysis. Furthermore, an algorithm to compute the feasible preemption threshold assignment is presented, and the proofs for the correctness of these algorithms are also presented.

Key words fixed-priority with preemption threshold (FPPT); scheduability analysis; stack resource protocol(SRP); preemption threshold assignment; shared resources

摘　要 受到广泛关注的抢占阈值调度算法能够有效减少现场切换次数,防止不必要的任务抢占,降低资源额外消耗,提高任务集合的可调度性.目前该调度算法的研究工作大多围绕独立任务集合展开,在实际实时系统中任务经常需要互斥访问共享资源,任务之间由于资源共享而导致的相关性对于任务集合的优先级分配和抢占阈值分配都有很大的影响.SRP 协议是在实时系统中得到广泛应用的资源访问控制协议,具有死锁避免、提前阻塞、共享任务栈等一系列优良特性.将 SRP 和抢占阈值调度算法结合起来,提出 FPTS 调度模型,给出相应的可调度性判定公式,考虑在任务之间使用 SRP 协议时求解任务抢占阈值分配,最后给出计算抢占阈值分配的伪多项式时间算法.

收稿日期:2007-01-23;**修回日期**:2008-06-27
基金项目:国家自然科学基金项目(90412011);国家"九七三"重点基础研究发展计划基金项目(2006AA10Z237)

关键词 抢占阈值调度(FPPT);可调度性分析;栈资源协议(SRP);抢占阈值分配;共享资源

中图法分类号 TP391

抢占阈值调度(preemption threshold scheduling, PTS)首先由文献[1]于1997年提出,并应用到商业实时操作系统ThreadX中.加拿大Concordia大学的Wang和Saksena于1999年形式化地论述了抢先阈值调度算法[2],并证明其特性,通过实例说明了在抢占式调度和非抢占式调度下均不调度的任务集合在抢先阈值下则是可调度.

提出抢占阈值调度模型的目的是减少抢占式调中由于频繁抢占所造成的执行现场切换次数,从而降低额外资源消耗,提高处理器的利用率.在抢占阈值调度模型下,任务集合为 $\Gamma=\tau_1,\tau_2,\cdots,\tau_n$,每个实时任务不仅分配任务优先级(常规优先级)$\pi_i\in[1,2,\cdots,n]$,还分配抢占阈值(实际优先级)$\gamma_i$,而且 $\pi_i\leqslant\gamma_i$,即 $\gamma_i\in[\pi_i,\cdots,n]$.如果任务 τ_i 使用常规优先级来竞争处理器,如果任务的常规优先级 π_i 大于当前正在运行任务 τ_j 的抢占阈值 γ_j,τ_i 就抢占 τ_j,获得处理器,此时 τ_i 就提升到自己的抢占阈值 γ_i.也就是说,τ_i 用于竞争处理器的使用权,而 γ_i 则是实际运行过程中使用的优先级.

当所有实时任务的抢占阈值等于各自的任务优先级时,抢占阈值调度就退化为抢占式调度;如果任务的抢占阈值等于任务集合内最高任务优先级时,抢占阈值调度就退化为非抢占式调度.因此,抢占式调度和非抢占式调度是抢占阈值调度的两个特例,而抢占阈值调度通过调节任务的抢占阈值,减少不必要的现场切换次数,但是同时保持部分的任务间可抢占性来满足任务的响应时间需求,从而提高整个任务集合的可调度性.

不过目前研究工作都假定任务集合内的各个实时任务之间是相互独立的,没有考虑任务之间共享非抢占资源的情况.然而在实际的实时系统中,任务之间共享资源是非常普遍的现象.为此,汉城国立大学的Kim和Hong等人于2002年提出在抢占阈值调度中集成两种资源访问控制协议[3],即集成优先级继承协议(PIP)和优先级冲顶协议(PCP)[4].其中PIP+PTS的方案依然会产生死锁和链式阻塞,任务最大阻塞时间也不收敛;而PCP+PTS的方案则容易产生阈值过限问题.为了解决这个问题,信息工程大学的王保进等人于2005年提出:通过使用伪资源[5]的概念将PTS纳入栈资源协议(stack resource protocol)的框架[6]下,即SRP-PT调度模型[7].在该模型下,同步任务集合中每个任务分配优先级之后,每个共享资源的冲顶值也就确定了,进而也确定了每个任务在各个阶段的SRP抢占阈值.文献[7]还分析了SRP-PT调度模型的特征,最后给出经过优化的优先级分配算法.不过SRP-PT调度模型没有考虑伪资源具体分配问题,所给出的算法是针对真实资源共享情况下的力求获得最优的优先级分配,不过,没有给出算法的复杂度分析和性能特征;同时,SRP-PT调度模型将使用伪资源的任务与使用实资源的任务分开判定各自的可调度性,但实际上,这两类任务之间存在相互影响.

本文针对目前研究的不足,针对任务间存在共享资源的静态优先级任务集合,提出一种使用SRP协议的抢占阈值调度算法FPTS(fixed-priority with preemption threshold under SRP).首先推导出任务之间存在资源相关性时的可调度性判定公式;然后在此基础上给出计算任务抢占阈值的分配算法;在理论上给出算法正确性证明的同时,大量仿真实验也表明该算法能够有效提高任务集合的可调度性.

1 假设与符号定义

周期性任务集合为 $\Gamma=\tau_1,\tau_2,\cdots,\tau_n$,每个任务 $\tau_i,i\in[1,2,\cdots,n]$,任务的一次执行称为任务实例,用 $J_{i,q}$ 表明任务 τ_i 的第 q 次执行时的任务实例.任务进入系统的时刻称为任务的到达时间(arrival time),任务实例的释放时间(release time)是其开始执行的时刻,任务实例从释放时间到结束执行所用的时间称为任务实例的响应时间(response time).在一个系统的整个运行过程中,任务的最长响应时间是所有任务实例中响应时间的最大者.

任务集合 Γ 中的任务用 $\tau_i=(T_i,C_i,D_i)(i=1,2,\cdots,n)$ 表示,其中 T_i 表示任务 τ_i 开始执行的周期(对于偶发性任务而言就是最小任务到达间隔),C_i 表示任务 τ_i 的最长执行时间(WCET),D_i 表示任务 τ_i 的相对截止期,D_i 与 T_i 不相关,即任务具有任意大小的截止期.对于任务 τ_i 而言,其任务实例 $J_{i,q}$ 在 $T_i\cdot(q-1)$ 时刻就绪并释放,任务执行最长需要 C_i 时间,最迟要在 D_i 之前完成.T_{mc} 表示任务集合中所有任务周期(最小任务到达间隔)的最小公倍数,即

任务集合的超周期. 如果一个任务实例执行后还没有结束,则称此任务实例是活动的. 这里假定系统中只包含数量有限的任务实例,而且当任务的一个实例正在运行时,不允许该任务的后继实例投入执行.

任务实例在执行期间需要用到处理器、内存栈空间和其他非抢占资源. 调度算法和资源访问控制协议管理如何在各个任务实例之间分配处理器时间、栈空间和非抢占资源. 处理器是可以抢占的,但是许多非抢占资源只能被互斥访问. 假设 $Re=\{\rho^1,\rho^2,\cdots,\rho^p\}$ 表示嵌入实时系统需要使用的非抢占资源列表,请求某个非抢占资源 $\rho^k(1\leqslant k\leqslant p)$ 的任务实例必须在获取该资源之后才能继续执行,等待资源的任务实例处于阻塞状态. 得到该资源的任务实例在释放资源之前都会持有该资源,此时资源就处于使用状态. 如无额外说明,在本文中所有的资源均指非抢占资源.

2 可调度性分析

2.1 FPTS 关键时刻

在使用 FPTS 调度算法的静态优先级实时系统中,其关键时刻的定义与 RM 关键时刻[8]不同.

1) 每个具有较高常规优先级的任务实例在同一时刻到达;

2) 一个任务实例可能被其他具有较低常规优先级的任务阻塞,阻塞的原因有两种:

①任务实例的常规优先级低于具有较低常规优先级任务的抢占阈值;

②任务实例需要访问具有较低常规优先级任务所占据的非抢占共享资源且造成最大阻塞时间的任务实例恰好在第一个关键时刻之前开始执行;

3) 任务实例以最大频率到达.

2.2 FPTS 抢占阈值

任务 τ_i 在执行期间可能需要访问多个资源,同时这些资源可能同时为其他任务所用,此时很难使用单一抢占阈值来概括这些复杂的资源共享关系,所以,需要在任务 τ_i 的不同执行阶段定义不同的抢占阈值. 假设任务 τ_i 的执行时间因为访问共享资源而被划分为 M 段,即 $C_i=\sum_{m=1}^{M}C_{i,m}$. 执行其中每一段时,如果没有高优先级抢占,则系统的资源冲顶值[6]不变化. 第 1 段的起始时间是 S_i,结束时间(也是第 2 段的起始时间)定义为 $F_{i,1}$,这一阶段的 FPTS 抢占阈值定义为 $\gamma_{i,1}^{\mathrm{FPTS}}$. 第 2 段的结束时间(也是第 3 段的起始时间)定义为 $F_{i,2}$,FPTS 抢占阈值定义为 $\gamma_{i,2}^{\mathrm{FPTS}}$. 以此类推,倒数第 2 段的结束时间为 $F_{i,M-1}$,FPTS 抢占阈值定义为 $\gamma_{i,M-1}^{\mathrm{FPTS}}$. 最后一段的结束时间就是 $F_i(q)$,抢占阈值是 $\gamma_{i,M}^{\mathrm{FPTS}}$.

对于使用 FPTS 调度算法的静态优先级系统而言,任务的抢占阈值往往大于预先分配的常规优先级. 给每个任务 τ_i 定义一个 FPTS 抢占阈值 γ_i^{FPTS},$\gamma_i^{\mathrm{FPTS}}\geqslant\pi_i$;如果多个任务(比如 τ_i 和 τ_j)访问非抢占资源 ρ,那么这些任务在使用 ρ 期间是相互不可抢占的,即要求 $\gamma_{i,i'}^{\mathrm{FPTS}}\geqslant\pi_j,\pi_i\leqslant\gamma_{j,j'}^{\mathrm{FPTS}}$,或者说 $\max(\pi_j,\pi_i)\leqslant\min(\gamma_{i,i'}^{\mathrm{FPTS}},\gamma_{j,j'}^{\mathrm{FPTS}})$. 根据 SRP 的规则,这些任务之中最高优先级作为资源 ρ 的冲顶值 $\varphi(\rho)$,可以认为这些任务在访问资源 ρ 时的抢占阈值就是该冲顶值 $\varphi(\rho)$,其他任务的优先级只有大于此冲顶值 $\varphi(\rho)$ 才能够抢占这些任务的执行,可以看出,SRP 是 FPTS 的一种特殊情况. 因此,对于任务集合 Γ 而言,任务的抢占阈值分配定义为 $\Upsilon=[\gamma_1^{\mathrm{FPTS}},\gamma_2^{\mathrm{FPTS}},\cdots,\gamma_n^{\mathrm{FPTS}}]$,$\gamma_i^{\mathrm{FPTS}}\in[\pi_i,n]$;如果任务 τ_i 和 τ_j 访问资源 ρ,γ_i^{FPTS},$\gamma_j^{\mathrm{FPTS}}\in[\max(\pi_i,\pi_j),n]$.

所以,对于访问多个资源的任务 τ_i 而言,它的抢占阈值 γ_i^{FPTS} 实际上是一个向量 $\boldsymbol{\Lambda}_i^{\mathrm{FPTS}}$,$\boldsymbol{\Lambda}_i^{\mathrm{FPTS}}=[\gamma_{i,1}^{\mathrm{FPTS}},\gamma_{i,2}^{\mathrm{FPTS}},\cdots,\gamma_{i,M}^{\mathrm{FPTS}}]$;出于表示形式上的统一,对于不访问资源的独立任务 τ_j 而言,它的抢占阈值 γ_j^{FPTS} 也可视为一个向量 $\boldsymbol{\Lambda}_j^{\mathrm{FPTS}}$,只是向量中只有一个元素,即 $\boldsymbol{\Lambda}_j^{\mathrm{FPTS}}=[\gamma_j^{\mathrm{FTPS}}]$. 因而,给定任务集合 Γ 的抢占阈值分配表示为 $\Upsilon=[\boldsymbol{\Lambda}_1^{\mathrm{FPTS}},\boldsymbol{\Lambda}_2^{\mathrm{FPTS}},\cdots,\boldsymbol{\Lambda}_n^{\mathrm{FPTS}}]$.

2.3 可调度性判定

由于使用 SRP 协议来管理资源访问,每个任务的执行时间被分割为若干段,每段可能出现不同的抢占阈值. 为了更加容易判定任务集合在这种情况下的可调度性,可以将任务的抢占阈值结构转化为一种正则形式:

定义 1. 如果任务 τ_i 的抢占阈值向量 $\boldsymbol{\Lambda}_i^{\mathrm{FPTS}}$ 中,各个执行阶段的抢占阈值呈现非递减次序,就称 $\boldsymbol{\Lambda}_i^{\mathrm{FPTS}}$ 是正则的.

如果任务 τ_i 的抢占阈值结构不是正则的,从 τ_i 的最后一个执行阶段开始检查,如果 τ_i 第 u 个阶段的阈值为 $\gamma_{i,u}^{\mathrm{FPTS}}$,第 $u+1$ 个阶段的阈值为 $\gamma_{i,u+1}^{\mathrm{FPTS}}$,$\gamma_{i,u+1}^{\mathrm{FPTS}}\leqslant\gamma_{i,u}^{\mathrm{FPTS}}$,那么令 $\gamma_{i,u}^{\mathrm{FPTS}}=\gamma_{i,u+1}^{\mathrm{FPTS}}$;逐步处理,直到 τ_i 的第 1 个执行阶段为止,然后将具有相等抢占阈值的执行阶段合并;这样就可以将 τ_i 的抢占阈值 $\boldsymbol{\Lambda}_i^{\mathrm{FPTS}}$ 转化为正则格式 $\overline{\boldsymbol{\Lambda}}_i^{\mathrm{FPTS}}$. 可以证明,任务 τ_i 在 $\overline{\boldsymbol{\Lambda}}_i^{\mathrm{FPTS}}$ 下其最长响应时间不会增加.

定理 1. 任务 τ_i 的抢占阈值被转化为正则形式后,最长响应时间没有增加.

证明. 假设 $J_{i,q}$ 是任务 τ_i 的某个实例,$\gamma_{i,u+1}^{\mathrm{FPTS}} \leqslant \gamma_{i,u}^{\mathrm{FPTS}}$,$S_{i,l}$ 和 $F_{i,l}$ 分别是 $J_{i,q}$ 的第 l 个执行阶段 $J_{i,q,l}$ 的开始执行时间和结束时间,$J'_{i,q}$ 是使用正则抢占阈值的任务实例,$S'_{i,l}$ 和 $F'_{i,l}$ 是对应时间值. 由于可能经历来自更高优先级的抢占,所以任务到达后可能无法立刻开始执行. 不妨假设 $J_{i,q}$ 和 $J'_{i,q}$ 同时到达,那么在 $F_{i,u-1}$ 时刻之前,两者的执行序列是相同的. 如果能够证明 $S_{i,u+2}=S'_{i,u+2}$,那么在此之后两者的执行也是相同的. 对于转换前的任务而言,$[F_{i,u-1}(S_{i,u}), S_{i,u+2}(F_{i,u+1})]$ 时间间隔内包含了 $J_{i,q,u}$,$J_{i,q,u+1}$ 的一次执行和任务优先级大于或者等于 $\gamma_{i,u}^{\mathrm{FTPS}}$ 的任务实例 $J_{j,q'}$(这些任务实例的优先级自然也大于或者等于 $\gamma_{i,u+1}^{\mathrm{FTPS}}$)的一次或者多次执行,这里将更高优先级的任务实例 $J_{j,q'}$ 分为两类:1)对于 $\pi_j > \gamma_{i,u}^{\mathrm{FTPS}}$ 的任务实例 $J_{j,q'}$ 而言,将 $\gamma_{i,u}^{\mathrm{FTPS}}$ 降低为 $\gamma_{i,u+1}^{\mathrm{FTPS}}$ 并不影响 $J_{j,q'}$ 的抢占行为;2)对于 $\gamma_{i,u+1}^{\mathrm{FTPS}} < \pi_j \leqslant \gamma_{i,u}^{\mathrm{FTPS}}$ 的任务实例 $J_{j,q'}$ 而言,$J_{j,q'}$ 在 $\gamma_{i,u}^{\mathrm{FTPS}}$ 未变化时无法抢占 $J_{i,q,u}$($J_{i,q}$ 的第 u 个执行阶段)的执行,$J_{j,q'}$ 被 $J_{i,q,u}$ 阻塞,只有 $J_{i,q,u}$ 执行完毕时 $J_{j,q'}$ 才能够继续运行,由于 $J_{j,q'}$ 的优先级 $\pi_j > \gamma_{i,u+1}^{\mathrm{FTPS}}$,$J_{j,q'}$ 实际上可以抢占 $J_{i,q,u+1}$,将 $J_{i,q,u+1}$ 的开始执行时间 $S_{i,u+1}$ 推迟. 如果 $\gamma_{i,u}^{\mathrm{FTPS}}$ 降低为 $\gamma_{i,u+1}^{\mathrm{FTPS}}$,$J_{j,q'}$ 就会抢占 $J_{i,q,u}$,将 $J_{i,q,u}$ 的结束时间 $F_{i,u}$ 推迟,不过却可以将 $J_{i,q,u+1}$ 的开始执行时间 $S_{i,u+1}$ 提前. 所以,即使将 $\gamma_{i,u}^{\mathrm{FTPS}}$ 降低为 $\gamma_{i,u+1}^{\mathrm{FTPS}}$,$[F_{i,u-1}, S_{i,u+2}]$ 时间间隔内发生的变化也不会影响 $F_{i,u+1}(S_{i,u+2})$. 而且,如果这样两个任务实例是在某个 level-i 忙周期内首个到达,那么该结果在忙周期结束时依然有效. 因此任务 τ_i 的最长响应时间没有增加. 证毕.

定理 1 表明,任务抢占阈值的正则形式不会影响任务本身的可调度性. 所以,在任务集合的可调度性分析过程中,对于每个任务 τ_i,可以使用经过正则转化的任务阈值结构来考察 τ_i 的可调度性.

定义任务 τ_i 的最低抢占阈值 PT_i 为 $\min\{\gamma_{i,j}^{\mathrm{FPTS}}, 1 \leqslant j \leqslant M_j\}$,对于任务 τ_j 而言,能够影响任务 τ_i 可调度性的任务 τ_j 主要分为两类:

1) $MP_i=\{\tau_j \mid \pi_j > PT_i\}$,$MP_i$ 内任务能够在 level-i 忙周期内多次抢占 τ_i;

2) $H_i=\{\tau_j \mid \pi_j < \pi_i \leqslant \gamma_{j,1}^{\mathrm{FPTS}}\}$,$H_i$ 内任务能够阻塞 τ_i 的执行.

在 FPTS 调度模型中,如果 $\pi_i > \gamma_{j,1}^{\mathrm{FPTS}}$,即使 $\pi_i \leqslant \gamma_{j,u}^{\mathrm{FPTS}}$,$\tau_i$ 也不再会被 τ_j 阻塞,所以任务 τ_i 的阻塞时间 B_i 为

$$B_i = \max\{C_{j,1} \mid \tau_j \in H_i\}, \tag{1}$$

此时,level-i 忙周期 L_i 的求解公式为

$$L_i = B_i + \sum_{\forall j, \pi_j \geqslant PT_i} \left\lceil \frac{L_i}{T_j} \right\rceil \cdot C_j. \tag{2}$$

那么,任务 τ_i 开始时间 $S_{i,1}(q)$ 和 $F_{i,1}(q)$ 的求解公式分别为

$$S_{i,1}(q) = B_i + q \cdot C_i + \sum_{\tau_j \in MP_i} \left(1 + \left\lfloor \frac{S_{i,1}(k)}{T_j} \right\rfloor\right) \cdot C_j, \tag{3}$$

$$F_{i,1}(q) = S_{i,1}(q) + C_i + \sum_{\tau_j \in MP_i} \left(\left\lceil \frac{F_{i,1}(q)}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{S_{i,1}(q)}{T_j} \right\rfloor\right)\right) \cdot C_j. \tag{4}$$

第 2 阶段的结束时间为

$$F_{i,2}(q) = F_{i,1}(q) + C_{i,2} + \sum_{\tau_j \in MP_i \setminus \{\tau_j \mid \pi_j \leqslant \gamma_{i,2}^{\mathrm{FPTS}}\}} \left(\left\lceil \frac{F_{i,2}(q)}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{F_{i1}(q)}{T_j} \right\rfloor\right)\right) \cdot C_j. \tag{5}$$

第 k 阶段的结束时间为

$$F_{i,k}(q) = F_{i,k-1}(q) + C_{i,k} + \sum_{\tau_j \in MP_i \setminus \{\tau_j \mid \pi_j \leqslant \gamma_{i,k}^{\mathrm{FPTS}}\}} \left(\left\lceil \frac{F_{i,k}(q)}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{F_{i,k-1}(q)}{T_j} \right\rfloor\right)\right) \cdot C_j. \tag{6}$$

以此类推,经过有限次运算,可以得到:

$$F_i(q) = F_{i,M_i-1}(q) + C_{i,M_i} + \sum_{\tau_j \in MP_i \setminus \{\tau_j \mid \pi_j \leqslant \gamma_{i,M_i}^{\mathrm{FPTS}}\}} \left(\left\lceil \frac{F_i(q)}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{F_{i,M-1}(q)}{T_j} \right\rfloor\right)\right) \cdot C_j. \tag{7}$$

那么,任务 τ_i 在各个阶段的最长响应时间为

$$R_{i,m} = \max_{q=0,1,2,\cdots,\left\lfloor L_i/T_j \right\rfloor} (F_{i,m}(q) - q \cdot T_i) \quad (1 \leqslant m \leqslant M_i - 1), \tag{8}$$

任务 τ_i 的最长响应时间为

$$R_i = \max_{q=0,1,2,\cdots,\left\lfloor L_i/T_j \right\rfloor} (F_i(q) - q \cdot T_i). \tag{9}$$

在计算任务 τ_i 的最长响应时间时,考虑了该任务在第 1 个 level-i 忙周期内所有的任务实例,因为 FPTS 关键时刻并非通用的,从关键时刻开始,在第 1 个 level-i 忙周期中,可能存在一个或多个任务实例,但是第 1 个实例并不一定经历最长的响应时间. 在判定公式的推导中,没有限制任务周期与最长截止期之间的关系,所以上述判定公式可以针对任务周期大于、等于或者小于最终截止期的所有类型的任务集合. 此外,推导过程也没有限制使用何种静态优先级调度算法,所以,判定公式能够与各种优先级分配算法配合.

3 抢占阈值分配

第 2 节中的可调度性判定公式都假定每个任务的优先级和抢占阈值已分配,但是实际上,只有任务优先级能够预先给定.那么,要决定给定任务集合是否可调度,关键问题就是能否使用第 2 节中的可调度性判定公式找到一个可行的抢占阈值分配.但是,所有可能的抢占阈值分配的解空间在最差情况下是 $O(n!)$,显然,需要更加有效的搜索算法.本节给出根据预先分配的任务优先级来计算可行抢占阈值的有效算法,并证明算法得到的抢占阈值分配能够使得任务集合可调度,随后给出算法的计算复杂性分析.

3.1 抢占阈值分配算法

算法 1. *AssignThresholdsOnSRP*.

输入:各任务的优先级$[\pi_1,\pi_2,\cdots,\pi_n]$;

输出:各任务的抢占阈值$[\Lambda_1,\Lambda_2,\cdots,\Lambda_n]$.

① for ($i\leftarrow 1$ to n)
② for ($q\leftarrow 1$ to i_m)
③ $\gamma_{i,q}\leftarrow\pi_i$
④ or $\gamma_{i,q}\leftarrow\varphi(\rho)$
⑤ $\Lambda_i\leftarrow[\gamma_{i,1},\gamma_{i,2},\cdots,\gamma_{i,I_m}]$
⑥ endfor
⑦ endfor
⑧ for ($i\leftarrow 1$ to n)
⑨ $F_i\leftarrow WCRT(\pi_i,\Lambda_i)$ //①～⑦
⑩ while ($F_i > D_i$)
⑪ for ($q\leftarrow 1$ to i_m)
⑫ $\gamma_{i,q}\leftarrow\gamma_{i,q}+1$
⑬ if $\gamma_{i,q} > n$ then
⑭ return FAIL
⑮ endif
⑯ $F_i\leftarrow WCRT(\pi_i,\Lambda_i)$ //①～⑦
⑰ endfor
⑱ endwhile
⑲ endfor
⑳ return SUCCESS

算法过程的行①～⑦给出所有任务抢占阈值的初始值,这个初始值考虑到 SRP 协议的要求,对于访问共享资源的任务而言,它们在访问期间的抢占阈值至少要大于访问者中最高优先级,这才符合 SRP 的规则.而后算法行⑧～⑰按照任务优先级非递减的次序来计算每个任务的抢占阈值.对于任务 τ_i 而言,如果它在当前阈值设置下最长响应时间超过最终截至期,就提升任务 τ_i 某一执行阶段的抢占阈值;如果提升过的抢占阈值大于系统内最高任务优先级,那么就不存在能够让任务 τ_i 可调度的抢占阈值,算法截止,见算法过程行⑩～⑰.否则,算法过程继续执行,直到每个任务的抢占阈值都设置完毕.

3.2 正确性证明

定理 2. 如果任务集合存在可行抢占阈值分配,算法过程 *AssignThresholdsOnSRP* 能够确保该分配.

证明. 算法过程的起点和搜索方向都是正确的.算法初始将任务的抢占阈值按照 SRP 的规则来设置,满足 SRP 协议的阈值设置是 FPTS 最基本的考虑,因为首先要保证任务之间访问共享资源的串行性.根据 INVALID 引理[2],当任务 τ_i 某一执行阶段的抢占阈值超过系统最高任务优先级时,整个任务集合不可调度.而后,算法的搜索方向是从最低优先级任务开始,逐渐计算到最高优先级任务.根据 AFFECT 推论[2],改变更高常规优先级某个阶段的抢占阈值不会影响低优先级任务的最长响应时间.而且当考虑任务 τ_i 时,所有优先级低于 π_i 的任务都已经将抢占阈值计算完毕,此时这些任务的最长响应时间不会超过最终截止期.假设任务集合不存在可行阈值分配而算法返回 Υ.算法过程中对每个任务都计算 $WCRT(\pi_i,\Lambda_i)$,如果算法最终计算完最高优先级任务的抢占阈值后成功截止,那么给定任务集合必然存在可行抢占阈值分配,否则算法过程会在中途返回 FAIL. 证毕.

3.3 复杂性分析

在 *AssignThresholdsOnSRP* 算法过程中,$WCRT(\pi_i,\Lambda_i)$用来计算任务 τ_i 在优先级 π_i 和抢占阈值 Λ_i 下的最长响应时间,它使用式(1)～(7)来完成计算过程.其中式(2)～(7)需要迭代计算,求解这些方程的计算复杂度是不确定的,因此 $WCRT(\pi_i,\Lambda_i)$的计算复杂度同样也是不确定的.为讨论方便,假定它的复杂度为 $O(r)$,其中 r 不是常数,它随着任务集合内任务的各种时间特征、优先级分配和抢占阈值分配的变化而变化,r 的最差情况是超周期 T_{mc} 内所有的任务实例都必须被检查.算法过程的行⑧～⑰中最外围 for 循环需要迭代 n 次,内部的 while 循环最差情况下需要迭代 $n\cdot M_i$ 次,其中 M_i 是任务 τ_i 的执行阶段个数.因此,$WCRT(\pi_i,\Lambda_i)$在最差情况下需要执行 $n^2\cdot\sum_{i=1}^{n}M_i$ 次,所以算法过程的计算复杂度为 $O\left(n^2\cdot\sum_{i=1}^{n}M_i\cdot r\right)$,是伪多项式时间的.

4 仿真实验

本节通过仿真实验来证明,与单独使用 SRP 协议相比,FPTS 调度模型能够进一步提高任务集合的可调度性.实验使用 SPAK 工具来分析静态优先级系统中任务集合的可调度性.为了分析访问共享资源对任务集合可调度性的影响,产生的随机任务集合根据访问共享资源的任务大体分为 3 类:1)1/4R 任务集合,集合中有 1/4 的任务访问共享资源;2)1/2R 任务集合,集合中有 1/2 的任务访问共享资源;3)3/4R 任务集合,集合中有 3/4 的任务访问共享资源.在每类任务集合中,单个任务可能访问最多多达 4 个共享资源,而最多可能不超过 4 个任务会访问同一个共享资源.对于一组随机任务集合,如果集合个数为 M,可调度的任务集合数量为 N,定义可调度率 $SR=M/N$.

生成随机任务时,有 4 个参数是变化的:1)任务个数 $totalTasks$,从 4 开始,以 4 为步长递增到 40.2)任务集合的最大周期 $maxPeriod$,对于上述 3 类任务均取 100,300,600,900 等 4 个值.3)单个任务最多可能访问资源个数 $maxResource$,存在 1,2,3,4 等 4 个可能值.4)单个资源可能共享的任务个数 $maxTasks$,也存在 1,2,3,4 等 4 个可能值.这 4 个参数的取值构成不同的测试点,在每个测试点,任务集合按照如下规则生成:

① 在[1, $maxPeriod$]之间均匀、随机地选择任务周期 T_i;

② 在[0.1/$totalTasks$,2.0/$totalTasks$]之间均匀、随机选择任务利用率 U_i;

③ 任务具有相互独立的截止期 D_i.

在每个测试点,独立生成 100 个任务集合,获得 SRP,FPTS 等调度模型针对这 100 个任务集合生成的可调度集合个数,并计算可调度率 SR 值.仿真实验在 Pentium4 2.4 GHz, 512 MB RAM, RedHat10 环境下运行.实验结果如下.

1) $maxResource=1$,$maxTasks=2$ 时仿真结果

如图 1、图 2 所示,$maxPeriod$ 的增加意味着各个任务的周期差异性增加,某些任务的周期可能很小,而某些任务的周期可能很大,这种差异性的增加使得任务集合更容易调度.同时如果任务集合中访问资源的任务越多,任务集合的可调度性就越不容易达到,直接体现是可调度率降低.

2) $maxResource=1$,$maxTasks=3$ 时的仿真结果

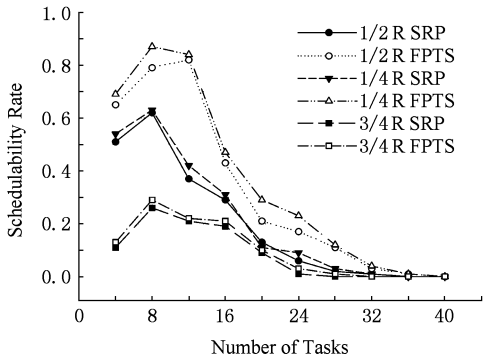

Fig. 1 Experiment results when $maxPeriod=100$.

图 1 $maxPeriod=100$ 时的实验结果

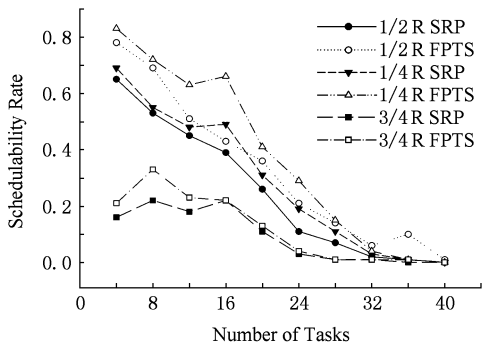

Fig. 2 Experiment results when $maxPeriod=600$.

图 2 $maxPeriod=600$ 时的实验结果

如图 3、图 4 所示,当更多任务使用同一种资源时,SRP 要求其中优先级最高者作为资源冲顶值,而各个任务的抢占阈值至少大于该值,那么能够用来调整任务可调度性的抢占阈值选择范围减少了,使得不可调度的任务集合增加.

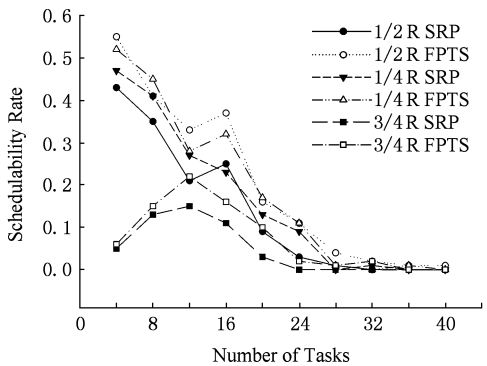

Fig 3. Experiment results when $maxPeriod=100$.

图 3 $maxPeriod=100$ 时的实验结果

3) $maxResource=4$,$maxTasks=4$ 时的仿真结果

如图 5、图 6 所示,当一个任务使用更多种资源时,任务在访问不同资源的执行阶段就需要不同的抢占阈值,这在某种程度上增加了通过抢占阈值的

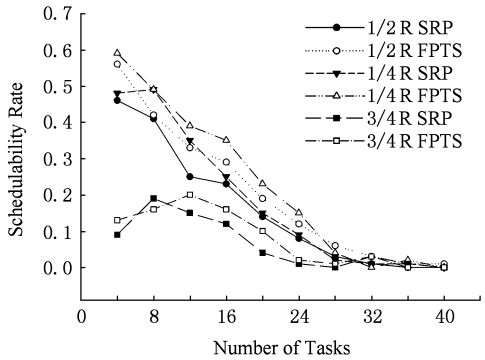

Fig. 4 Experiment results when $maxPeriod=600$.

图 4 $maxPeriod=600$ 时的实验结果

调整来提高任务集合可调度性的能力,能够调度的任务集合数量增加了,但是图 6 中由于 $maxTasks$ 的增加,可调度率的提高被抵消.

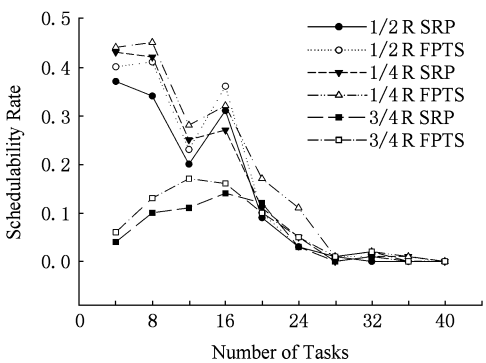

Fig. 5 Experiment results when $maxPeriod=100$.

图 5 $maxPeriod=100$ 时的实验结果

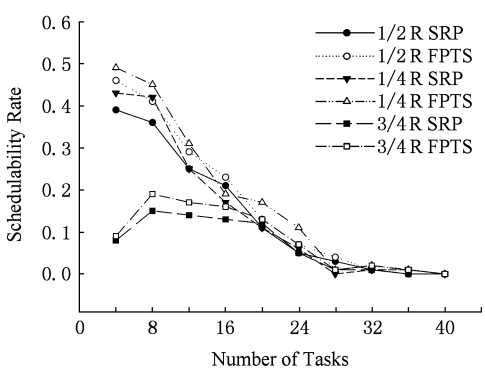

Fig. 6 Experiment results when $maxPeriod=600$.

图 6 $maxPeriod=600$ 时的实验结果

4) 可调度性判定所需时间的仿真结果

如图 7、图 8 所示,$maxPeriod$ 的增加对可调度性判定的时间复杂度也没有太多的影响. 当一个任务使用更多种资源时,可调度性判定需要考虑任务执行在不同抢占级下时的响应时间,这增加了处理时间. 如果任务集合中访问资源的任务越多,对可调度性判定的时间复杂度有一定程度的影响.

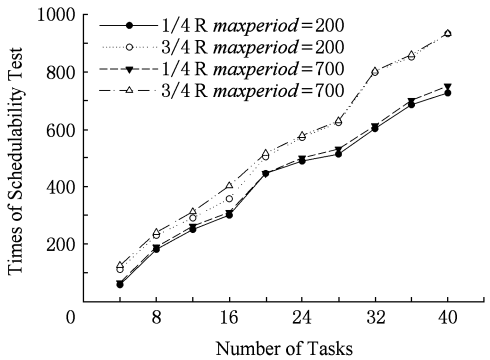

Fig. 7 Experiment results when $maxResource=1$, $maxTasks=3$.

图 7 $maxResource=1$, $maxTasks=3$ 时的实验结果

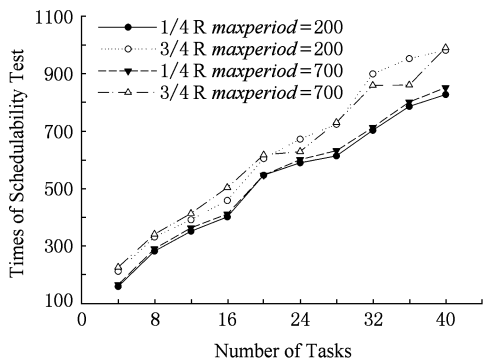

Fig. 8 Experiment results when $maxResource=4$, $maxTasks=4$.

图 8 $maxResource=4$, $maxTasks=4$ 时的实验结果

5 结束语

本文深入系统地讨论了将 SRP 协议与抢占阈值调度算法有效集成的问题,提出 FPTS 调度模型;由于任务之间存在资源相关性,任务在执行期间可能拥有多个抢占阈值;为简化分析,首先将任务的抢占阈值向量转换为正则形式,而后使用 level-i 忙周期分析方法推导出 FPTS 模型下任务第 1 个执行阶段的最长响应时间,而后分段计算其他阶段的响应时间,最终使用整个任务的最长响应时间与任务最终截止期比较来判定任务的可调度性. 而后在上述工作的基础上,给出计算抢占阈值分配的有效算法,该算法使用新推导的判定公式判断分配了优先级和抢占阈值之后每个任务的可调度性,该算法的复杂度是伪多形式时间的. 最后通过仿真实验表明 FPTS 调度模型能够提高任务集合的可调度性.

参 考 文 献

[1] Lamie W. Preemption-threshold [OL]. (1997) [2005-11-10]. http://www.threadx.com/wppreemption.html

[2] Wang Yun, Saksena Manas. Scheduling fixed-priority tasks with preemption threshold[C] //Proc of the 6th Int Conf on Real-Time Computing Systems and Applications (RTAS'99). Los Alamitos, CA: IEEE Computer Society, 1999: 328-335

[3] Kim Saehwa, Hong Seongsoo, Kim Tae-Hyung. Integrating real-time synchronization schemes into preemption threshold scheduling[C] //Proc of the 5th IEEE Int Symp on Object-Oriented Real-Time Distributed Computing. Los Alamitos, CA: IEEE Computer Society, 2002: 145-152

[4] Ragunathan R, Lui Sha, John J P. Priority inheritance protocols: An approach to real-time synchronization [J]. IEEE Trans on Computers, 1990, 39(9): 1175-1185

[5] Paolo G, Giuseppe L, Marco D N. Minimizing memory utilization of real-time task sets in single and multi-processor system-on-a-chip[C] //Proc of the 22nd Real-Time Systems Symp (RTSS 2001). Los Alamitos, CA: IEEE Computer Society, 2001: 73-83

[6] Baker T P. A stack-based resource allocation policy for real-time processes[C] //Proc of the 11th Real-Time Systems Symposium. Los Alamitos, CA: IEEE Computer Society, 1990: 191-200

[7] Wang Baojin. The research on the task scheduling, resources shared model and algorithms in real-time embedded systems [D]. Zhengzhou: PLA Information Engineering University, 2005 (in Chinese)
(王保进. 嵌入式实时系统的任务调度与资源共享模型及算法研究[D]. 郑州: 解放军信息工程大学, 2005)

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

He Xiaochuan, born in 1977. Received his B. A's and M. A's degrees in computer engineering from the PLA Information Engineering University, Zhengzhou, China, in 1999 and 2002 respectively. From 2004, he became the Ph. D. candidate in computer science and techniques from the National University of Defense Technology (NUDT), Changsha, China. His current research interests include embedded systems, real-time scheduling algorithms and ubiquitous computing.

贺小川,1977 年生,博士研究生,主要研究方向为嵌入式系统、实时调度算法、普适计算.

Jia Yan, born in 1964. Ph. D. supervisor. She has been professor of the National University of Defense Technology(NUDT) since 2000. Her main research interests include database system, software component techniques and distributed computing.

贾 焰,1964 年生,教授,博士生导师,主要研究方向为数据库、软构件技术与分布计算.

Research Background

Fixed-priority scheduling with preemption threshold (FPPT) allows a task to disable preemptions from tasks up to a specified preemption threshold priority. Tasks with a priority greater than the preemption threshold priority are still allowed to preempt. The preemption threshold scheduling model has been shown to reduce the run-time costs by eliminating unnecessary task preemptions. Furthermore, FPPT allows tasks to be partitioned into non-preemptive groups to minimize the number of threads and the stack memory requirement [4], thereby leading to scalable real-time systems. In this paper, concentrating on the correlations resulting from the exclusively-shared resources between tasks, we introduce a new scheduling scheme called FPTS, which combines the FPPT scheduling approach and SRP resource access, and can improve the schedulability of task set further. Our work is supported by the National Natural Science Foundation of China (90412011) and the 973 Plans (2006AA10Z237).