DOI:10.7544/issn1000-1239.20210202
59(6):1202-1212,2022

固定优先级混合关键偶发任务能耗感知算法

张忆文[1] 高振国[1] 林铭炜[2]
[1](华侨大学计算机科学与技术学院 福建厦门 361021)
[2](福建师范大学数学与信息学院 福州 350117)
(zyw@hqu.edu.cn)

Fixed Priority Mixed-Criticality Sporadic Tasks Energy-Aware Algorithm

Zhang Yiwen[1], Gao Zhenguo[1], and Lin Mingwei[2]
[1](*College of Computer Science and Technology, Huaqiao University, Xiamen, Fujian* 361021)
[2](*College of Mathematics and Informatics, Fujian Normal University, Fuzhou* 350117)

Abstract Mixed-criticality systems integrate different criticality levels applications and components into a common shared platform. Energy consumption is very important for mixed-criticality systems due to size, weight and volume constraints. Energy-aware scheduling algorithm is the effective method to solve the energy consumption problem of mixed-criticality systems. Existing energy-aware algorithms based on dynamic priority schemes have lower slack time utilization. Fixed priority mixed-criticality scheduling (FPMCS) algorithm is proposed to solve the energy consumption problem of mixed-criticality systems. Firstly, a criticality rate monotonic scheme (CRMS) is proposed to schedule mixed-criticality sporadic tasks. In addition, the scheduling feasibility of CRMS is analyzed and the energy-aware speed is computed. Secondly, the slack time reserved for higher criticality level task is used to re-compute the utilization of higher criticality level task. The dynamically update the utilization of mixed-criticality sporadic tasks method through the method of event triggering is proposed to reclaim slack time generated from the random arrival of sporadic tasks. Thirdly, the speed of tasks is determined by the mixed-criticality sporadic tasks utilization to save energy. Finally, FPMCS algorithm is verified to be feasible by theoretical analysis and experiments. The experimental results show that the proposed FPMCS algorithm can save about 33.21% of energy consumption than existing algorithms.

Key words fixed priority; real-time scheduling; energy management; mixed criticality; sporadic tasks

摘 要 混合关键系统是将不同关键层次的应用或组件集成到同一个共享平台.由于受尺寸、重量与体积的限制,能耗对于混合关键系统而言尤其重要.能耗感知调度算法是解决混合关键系统能耗问题的关键,现有的能耗感知算法主要基于动态优先级策略且空闲时间利用率低.针对固定优先级混合关键系统偶发任务能耗感知问题,提出节能效果更好的固定优先级混合关键调度(fixed priority mixed criticality schedule, FPMCS)算法.首先,提出关键层次单调速率策略(criticality rate monotonic scheme, CRMS)调度混合关键偶发任务,分析该策略的调度可行性,且计算出能耗感知速度.其次,利用高关键层次任务

收稿日期:2021-03-12;**修回日期**:2021-06-25
基金项目:厦门市青年创新基金项目(3502Z20206012);国家自然科学基金项目(61872086,61972166)
This work was supported by the Youth Innovation Fund Projects of Xiamen City (3502Z20206012) and the National Natural Science Foundation of China (61872086, 61972166).

预留的空闲时间,通过事件触发的方法动态更新混合关键偶发任务集的利用率来回收偶发任务到达时间不确定产生的空闲时间.再次,利用混合关键偶发任务集的利用率决定任务的执行速度以达到降低能耗的目的.最后,通过理论分析和实验验证 FPMCS 算法是可行的;仿真实验表明:所提出的 FPMCS 算法比现有的方法可以节约大约 33.21%的能耗.

关键词 固定优先级;实时调度;能耗管理;混合关键;偶发任务

中图法分类号 TP316.2

随着处理器技术和计算机技术的快速发展,嵌入式系统的发展趋势是将多个不同关键层次的应用集中到同一共享平台形成混合关键系统,以满足时限、功耗与体积的需求.目前,混合关键系统的应用比较广泛,例如航空领域、汽车和工业标准[1](IEC 61508,DO-178B DO-178C,ISO 26262).无人机就是混合关键系统的典型应用.无人机的任务按照属性可以分为安全关键功能和任务关键功能,而安全关键功能必须通过认证.由于无人机等混合关键系统采用电池供电,因此能耗成为设计混合关键系统的重要目标.

目前,混合关键系统的研究已经吸引了广大研究者的兴趣;而这些研究者大多数都关注混合关键系统调度可行性分析方面.Vestal[2]最早研究混合关键系统的调度问题并且分析偶发任务模型的调度可行性.随后,Baruah 等人[3]提出了 2 种有效的混合关键偶发任务调度算法.这 2 种算法分别为静态混合关键(static mixed criticality, SMC)调度和可适应的混合关键(adaptive mixed criticality, AMC)调度.SMC 算法和 AMC 算法主要区别在于系统处于高模式时,对低关键层次任务的处理.SMC 算法在系统处于高模式时,允许部分低关键层次任务执行,而 AMC 算法舍弃所有的低关键层次任务.Davis 等人[4]扩展了 SMC 和 AMC 算法,考虑系统模式切换和上下文切换的开销.

文献[2-4]的研究工作主要针对固定优先级系统.Baruah 等人[5]最早研究动态优先级混合关键偶发任务的调度问题,并且提出基于虚拟截止期限的最早截止期限优先(earliest deadline first-virtual deadline, EDF-VD)算法.EDF-VD 算法在系统处于高模式时,放弃所有低关键层次任务.Liu 等人[6]提出一种新的混合关键偶发任务模型即非精确混合关键(imprecise mixed criticality, IMC)任务模型.此外,基于利用率和时间需求分析的方法被用来分析 IMC 模型.Chen 等人[7]提出可行混合关键(flexible mixed criticality, FMC)调度模型,该模型中混合关键偶发任务的最小释放时间可变.此外,Chen 等人[7]提出一种基于 EDF-VD 的利用率分析方法调度混合关键偶发任务.这些研究中,系统处于高模式时,所有的低关键层次任务都将不执行.Su 等人[8]研究弹性的混合关键(elastic mixed criticality, E-MC)偶发任务模型且提出最早释放最早截止期限优先(earliest releases-earliest deadline first, ER-EDF)算法.该算法在系统处于高关键模式时,允许执行部分低关键层次偶发任务.

动态电压频率调节(dynamic voltage frequency scaling, DVFS)是降低系统能耗的主要技术,很多研究者将实时调度理论与 DVFS 技术结合起来解决系统的能耗问题,但这些研究工作主要针对传统的嵌入式系统[9-15].近来,有些研究者开始关注混合关键系统能耗感知调度问题.Huang 等人[16]最早研究混合关键偶发任务的能耗感知问题,提出基于 EDF-VD 能耗感知算法,该算法没有利用高关键层次偶发任务预留的空闲时间.Ali 等人[17]扩展了 Huang 等人[16]的研究成果,提出不仅能够利用静态空闲时间,而且还能利用高关键层次偶发任务预留的空闲时间降低能耗的算法.文献[4,18]针对动态优先的混合关键系统提出动态利用率更新算法,该算法能够动态利用偶发任务产生的空闲时间,进一步降低系统能耗.然而,这些研究工作在系统处于高模式时,将所有低关键层次任务直接舍弃.Sruti 等人[19]针对精确混合关键任务模型,结合 DVFS 技术,提出基于利用率分析的能耗感知算法.该算法在系统处于高模式时,通过减少低关键层次任务的服务速率来降低系统能耗,而不是简单地将所有低关键层次任务舍弃.文献[16-19]研究工作的重点在于系统的能耗与实时性,Jiang 等人[20]研究混合关键系统软实时任务的可靠性与能耗感知问题,通过软实时任务执行时间服从随机分布的特点,提出动态编程最优算法.此外,Safari 等人[21]研究混合关键双处理器系统的可靠性感知调度问题,提出一种新的

可靠性感知能耗管理算法,该算法通过减少低关键层次任务的服务速率来提高系统可靠性.

通过文献[16-21]的调研发现,现有的混合关键能耗感知算法主要关注动态优先级混合关键系统的偶发任务调度问题,很少有研究者关注固定优先级混合关键系统偶发任务的能耗感知调度问题.此外,现有的研究工作没有利用偶发任务到达时间不确定产生的空闲时间,空闲时间的利用率低,导致节能效果不佳.

1 系统模型

1.1 混合关键偶发任务模型

在单处理器系统上考虑相互独立的混合关键偶发任务集 Γ,该任务集由 n 个偶发任务组成即 $\Gamma=\{\tau_1,\tau_2,\cdots,\tau_n\}$,每个偶发任务 $\tau_i(1\leqslant i\leqslant n,i$ 为整数)由五元组 $\{T_i,D_i,L_i,C_i(\mathrm{LO}),C_i(\mathrm{HI})\}$ 表示.其中,T_i 是 τ_i 的最小释放时间;D_i 是 τ_i 的相对截止期限;L_i 是 τ_i 的关键层次,其可以表示为 $L_i=\{\mathrm{LO},\mathrm{HI}\}$;$L_i$ 的取值为 LO 代表 τ_i 是低关键层次任务,其不需要进行安全认证;L_i 的取值为 HI 代表 τ_i 是高关键层次任务,其需要进行安全认证;$C_i(\mathrm{LO})$和 $C_i(\mathrm{HI})$分别表示 τ_i 低模式和高模式的最坏情况执行时间.本模型中,混合关键系统的低模式是指系统既执行低关键层次偶发任务又执行高关键层次偶发任务且不需要进行安全认证;混合关键系统的高模式是指系统只执行高关键层次偶发任务且需要进行安全认证.考虑 τ_i 具有隐性的截止期限,即其相对截止期限等于最小释放时间,即 $D_i=T_i$.低关键层次偶发任务 τ_i 的高模式和低模式最坏情况执行时间相等,即 $C_i(\mathrm{HI})=C_i(\mathrm{LO})$;高关键偶发任务 τ_i 高模式最坏情况执行时间大于或者等于低模式最坏情况执行时间,$C_i(\mathrm{HI})\geqslant C_i(\mathrm{LO})$;这个假设是合理的,因为系统处于高模式时,高关键层次任务需要进行安全认证.此外,假设任务的执行时间与处理器速度为线性关系,即以速度 S 执行,其执行时间变为 $C_i(\mathrm{LO})/S$.令 τ_{ij} 表示偶发任务 τ_i 的第 j 个任务实例.

系统的执行过程为:τ_i 以速度 S 完成执行且其执行时间大于 $C_i(\mathrm{LO})/S$ 但不超过 $C_i(\mathrm{HI})/S$ 时,系统处于高模式;τ_i 以速度 S 完成执行且其执行时间超过 $C_i(\mathrm{HI})/S$ 时,系统存在错误行为;τ_i 以速度 S 完成执行且其执行时间不超过 $C_i(\mathrm{LO})/S$ 时,系统处于低模式;开始时,系统处于低模式;一旦发现 τ_i 以速度 S 执行,且其执行时间超过 $C_i(\mathrm{LO})/S$ 时,系统立即切换到高模式;系统处于高模式时,所有高关键层次任务完成执行或者系统处于空闲状态时,返回到低模式.

调度正确标准为:考虑相互独立的混合关键偶发任务集 $\Gamma=\{\tau_1,\tau_2,\cdots,\tau_n\}$,算法调度该任务集可行必须满足 2 个条件:

1) 系统处于低模式时,所有任务以速度 S 执行,必须在其相应的截止期限内完成执行,且其执行时间不超过 $C_i(\mathrm{LO})/S$.

2) 系统处于高模式时,低关键层次任务全部被丢弃,只执行高关键层次任务;此外;所有高关键层次任务以速度 S 执行,必须在其相应的截止期限内完成执行,且其执行时间不超过 $C_i(\mathrm{HI})/S$.

1.2 能耗模型

处理器的功耗包括静态功耗 P_{s} 和动态功耗 P_{d}[1,9-10].静态功耗 P_{s} 主要由亚阈值电流 I_{sub} 产生,$P_{\mathrm{s}}=I_{\mathrm{sub}}SV_{\max}$,其中 S 是处理器的归一化速度,其值为 $S=\dfrac{V_i}{V_{\max}}=\dfrac{f_i}{f_{\max}}$,$V_i$ 和 f_i 是处理器的电压和频率,$V_{\max}$和 $f_{\max}$是处理器最大的电压和频率[1];动态功耗 P_{d} 由频率相关功耗 P_{dep}和频率无关功耗 P_{ind} 组成,即 $P_{\mathrm{d}}=P_{\mathrm{dep}}+P_{\mathrm{ind}}$,其中 P_{ind}是一个常数.频率相关功耗 P_{dep}主要由电路交换活动产生,$P_{\mathrm{dep}}=C_{\mathrm{ef}}\times(S^2V_{\max}^2)\times S\times f_{\max}$,其中,$C_{\mathrm{ef}}$是处理器的有效负载电容.因此,处理器的功耗 $P(S)$的计算公式[1]为

$$P(S)=P_{\mathrm{ind}}+SP_{\mathrm{s}}^{\max}+S^3P_{\mathrm{dep}}^{\max},\tag{1}$$

其中,$P_{\mathrm{s}}^{\max}=I_{\mathrm{sub}}V_{\max}$是最大的静态功耗,$P_{\mathrm{dep}}^{\max}=C_{\mathrm{ef}}\times V_{\max}^2f_{\max}$是最大的频率相关功耗.从式(1)中可以看出,处理器功耗由 3 部分构成:1)P_{ind},频率无关功耗主要由 I/O 设备和内存操作产生,独立于处理器的频率,不能通过低功耗技术降低;2)$SP_{\mathrm{s}}^{\max}$,由漏电流产生的静态功耗;3)$S^3P_{\mathrm{dep}}^{\max}$,由电路交换活动产生的频率相关的功耗.最大静态功耗与最大的频率相关功耗之间存在着相应的关系,即 $P_{\mathrm{s}}^{\max}=\theta P_{\mathrm{dep}}^{\max}$,其中 θ 为常数($0<\theta\leqslant 1$).因此,式(1)可以改写为

$$P=P_{\mathrm{ind}}+P_{\mathrm{dep}}^{\max}(\theta S+S^3),\tag{2}$$

处理器在区间$[t_1,t_2]$的能耗 $E=\int_{t_1}^{t_2}P(S)\,\mathrm{d}t$.

2 关键层次单调速率调度

传统的嵌入式调度策略例如单调速率(rate monotonic, RM)策略和最早截止期限优先(earliest

deadline first, EDF)策略调度混合关键偶发任务效率低下,甚至产生不可行的调度[5].为了提高混合关键任务的调度效率,充分利用系统资源,借鉴文献[3]的调度策略,提出关键层次单调速率调度策略(criticality rate monotonic scheme, CRMS).CRMS是固定优先级调度策略,其分配优先级的规则为:1)根据任务的关键层次分配优先级.关键层次越高,其优先级越高;关键层次越低,其优先级越低.2)相同关键层次的任务,按照RM策略分配优先级.最小释放时间越小,其优先级越高;最小释放时间越大,其优先级越大.CRMS分配任务的优先级规则简单、容易操作,且开销小;更重要的是使用该调度策略调度混合关键任务,低关键层次任务出错不会影响到高关键层次任务的执行.

尽管存在一些特例使得CRMS出现优先级反转问题,导致其不是单纯的RM调度.例如存在一个高关键层次任务 τ_1 和低关键层次任务 τ_2,但 τ_1 的最小释放时间间隔大于 τ_2,导致最小释放时间间隔大的 τ_1 的优先级大于 τ_2.通过将高关键层次 τ_1 划分为若干虚拟任务,使得每个虚拟任务的最小释放时间都小于 τ_2 的最小释放时间,且这些虚拟任务的利用率之和等于 τ_1 的利用率.通过虚拟任务的构建,可以避免CRMS中出现优先级反转问题,使得CRMS是完全的单调速率调度.

介绍CRMS调度可行的充分条件之前,先介绍一些概念.

任务集的利用率 $U_x^y(\Gamma)$ 定义为

$$U_x^y(\Gamma)=\sum_{\tau_i\in\Gamma\wedge L_i=x}\frac{C_i(y)}{T_i},\tag{3}$$

其中,x 代表任务的关键层次即 $x\in\{\mathrm{LO},\mathrm{HI}\}$,$y$ 代表系统的模式即 $y\in\{\mathrm{LO},\mathrm{HI}\}$,$C_i(y)$ 是系统在 y 模式下的最坏情况时间.例如:$U_{\mathrm{HI}}^{\mathrm{LO}}(\Gamma)$ 代表高关键层次任务集低模式利用率;$U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)$ 代表高关键层次任务集高模式利用率.

定理1. 系统处于低模式时,CRMS调度混合关键偶发任务集 Γ 调度可行的充分条件为

$$U_{\mathrm{LO}}^{\mathrm{LO}}(\Gamma)+U_{\mathrm{HI}}^{\mathrm{LO}}(\Gamma)\leqslant F(n),\tag{4}$$

其中,$F(n)$ 是RM策略调度可行的充分条件,$F(n)=n(2^{\frac{1}{n}}-1)$.

证明. CRMS是固定优先级调度策略,其优先级根据任务的关键层次和RM策略进行分配.如果CRMS调度混合关键偶发任务集 Γ 是可行的,其必须满足单处理器抢占式RM策略的可信特征[22].CRMS策略在系统处于低模式时,任意 τ_i 的执行时间都不超过 $C_i(\mathrm{LO})$.从RM策略调度可行的充分条件可知[23],系统处于低模式时,式(4)是CRMS调度混合关键偶发任务集 Γ 调度可行的充分条件.

证毕.

定理2. 系统处于高模式时,CRMS调度混合关键偶发任务集 Γ 调度可行的充分条件为

$$U_{\mathrm{LO}}^{\mathrm{HI}}(\Gamma)+U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)\leqslant F(n).\tag{5}$$

证明. 由混合关键偶发任务模型可知,任意的低关键层次偶发任务 τ_i 都有 $C_i(\mathrm{HI})=C_i(\mathrm{LO})$,因此可以得出 $U_{\mathrm{LO}}^{\mathrm{HI}}(\Gamma)=U_{\mathrm{LO}}^{\mathrm{LO}}(\Gamma)$.由系统行为可知,系统最初是处于低模式,可以同时执行低关键层次任务和高关键层次任务,只有当高关键层次任务 τ_i 的执行时间超过 $C_i(\mathrm{LO})$ 时,系统才进入高模式.根据RM策略调度可行的充分条件[23],式(6)必须成立:

$$U_{\mathrm{LO}}^{\mathrm{HI}}(\Gamma)+U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)\leqslant F(n)\tag{6}$$

$$\Rightarrow U_{\mathrm{LO}}^{\mathrm{LO}}(\Gamma)+U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)\leqslant F(n),\tag{7}$$

所以,系统处于高模式时,式(5)是CRMS调度混合关键偶发任务集 Γ 调度可行的充分条件. 证毕.

推论1. 系统处于低模式时,CRMS以速度 S_1 调度混合关键偶发任务集 Γ 调度可行,则速度 S_1 需要满足:

$$U_{\mathrm{LO}}^{\mathrm{LO}}(\Gamma)/F(n)+U_{\mathrm{HI}}^{\mathrm{LO}}(\Gamma)/F(n)\leqslant S_1.\tag{8}$$

证明. 系统处于低模式时,所有的任务都以统一的速度 S_1 执行,任意的 τ_i 其执行时间将变为 $C_i(\mathrm{LO})/S_1$,其利用率将变成 $C_i(\mathrm{LO})/(S_1\times T_i)$.因此,式(5)可以改写为 $U_{\mathrm{LO}}^{\mathrm{LO}}(\Gamma)/S_1+U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)/S_1\leqslant F(n)$,进而得出式(8). 证毕.

推论2. 系统处于高模式时,CRMS在低模式和高模式分别以速度 S_2 和最大处理器速度 $S_{\max}$(归一化处理,$S_{\max}=1$)调度混合关键偶发任务集 Γ 调度可行,则速度 S_2 需要满足:

$$U_{\mathrm{LO}}^{\mathrm{LO}}(\Gamma)/S_2+\sum_{\tau_i\in\Gamma\wedge L_i=\mathrm{HI}}\left(\frac{C_i(\mathrm{LO})}{T_i\times S_2}+\frac{C_i(\mathrm{HI})-C_i(\mathrm{LO})}{T_i}\right)\leqslant F(n).\tag{9}$$

证明. 系统开始处于低模式,低关键层次任务以速度 S_2;只有高关键层次任务在低模式以速度 S_2 执行且其执行时间超过 $C_i(\mathrm{LO})/S_2$,系统才进入高模式;进入高模式之后所有低关键层次任务都被丢弃.而高关键层次任务 τ_i 在进入高模式之后以最大处理器速度 $S_{\max}=1$ 执行,所以其执行时间可以分为2个部分 $C_i(\mathrm{LO})/S_2+(C_i(\mathrm{HI})-C_i(\mathrm{LO}))/S_{\max}$,其利用率变为 $C_i(\mathrm{LO})/(S_2\times T_i)+(C_i(\mathrm{HI})-C_i(\mathrm{LO}))/T_i$,因此将其带入式(5),进而得出式(9). 证毕.

从推论 1 和推论 2 可以直接得到推论 3.

推论 3. CRMS 在低模式和高模式分别以速度 S 和 $S_{\max}$ 调度混合关键偶发任务集 Γ,调度可行,则速度 S 需要满足:

$$S=\max\{S_1,S_2\}. \tag{10}$$

3 混合关键偶发任务能耗感知算法

3.1 研究动机和问题阐述

混合关键系统的能耗感知调度问题受到一些研究者的关注[16-18,24-25].然而,这些研究者主要关注动态优先级混合关键偶发任务能耗感知调度,这些研究成果不能直接应用于固定优先级混合关键系统且这些研究成果空闲时间利用率不足,导致节能效果差.此外,目前的研究工作假设偶发任务以其最小释放时间释放任务实例;然而,偶发任务的到达时间是不确定的,这必然导致系统产生大量空闲时间.通过表 1 的任务集来解释现有研究的空闲时间利用率低、节能效果差.

Table 1 Mixed Criticality Sporadic Tasks Set Γ

表 1 混合关键偶发任务集 Γ

任务	T_i	D_i	L_i	C_i(LO)	C_i(HI)
τ_1	8	8	HI	1	2
τ_2	12	12	LO	3	3
τ_3	16	16	LO	4	4

假设 τ_1 在时刻 0,11,20,32,44 释放任务实例;τ_2 在时刻 0,14,28,40 释放任务实例;τ_3 在时刻 0,18,34 释放任务实例.假设处理器提供的速度范围为[0.3,1],能耗感知算法在区间[0,48]内调度混合关键偶发任务集 Γ.通过实例来解释现有能耗感知调度算法空闲时间利用率不高,节能效果不佳.

CRMS 算法为:系统处于低模式时,所有偶发任务以速度 S(式(9))执行;系统处于高模式时,高关键层次偶发任务以最大处理器速度 $S_{\max}$ 执行.通过计算可知 $F(3)=0.78$,$U_{\mathrm{LO}}^{\mathrm{LO}}(\Gamma)=0.5$,$U_{\mathrm{HI}}^{\mathrm{LO}}(\Gamma)=0.125$,$U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)=0.25$,$S=0.97$.系统处于低模式时,CRMS 算法调度混合关键偶发任务集 Γ 的结果如图 1 所示:

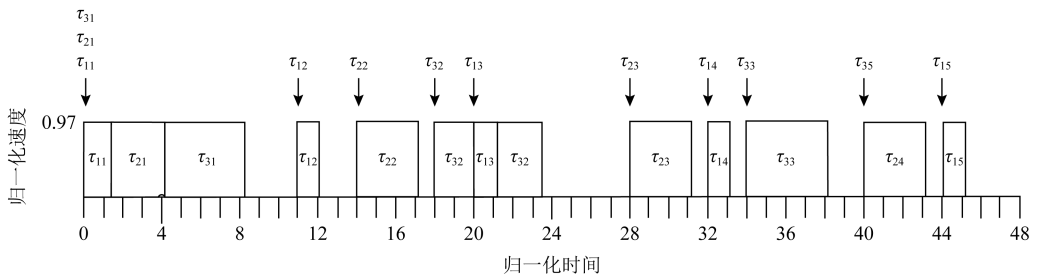

Fig. 1 CRMS algorithm schedules mixed criticality sporadic tasks Γ in low mode

图 1 CRMS 算法在低模式调度混合关键偶发任务集 Γ

图 1 中黑色箭头代表任务实例的到达时间,τ_1 的优先级最高,τ_2 的优先级次之,τ_3 的优先级最低.τ_1 在时刻 0 以速度 0.97 开始执行且在时刻 1.03 完成执行.在时刻 1.03,τ_2 以速度 0.97 开始执行且在时刻 4.12 完成执行.τ_3 在时刻 4.12 以速度 0.97 开始执行且在时刻 8.24 完成执行.在时刻 11,τ_1 以速度 0.97 开始执行且在时刻 12.03 完成执行.τ_2 在时刻 14 以速度 0.97 开始执行且在时刻 17.09 完成执行.τ_3 在时刻 18 开始以速度 0.97 执行,在时刻 20 被 τ_1 抢占;τ_1 在时刻 21.03 完成执行.τ_3 恢复执行且在时刻 23.15 完成执行.剩余的偶发任务以相似的方法调度,不再赘述.

高关键层次空闲时间回收(reclaim high-criticality slack-time, RHS)算法[17]为:文献[17]提出基于动态优先级能耗感知调度算法,该算法不仅能够利用系统的静态空闲时间,而且还能够利用高关键层次任务预留的空闲时间.现将文献[17]提出的算法拓展到固定优先级系统.RHS 算法基于 CRMS 调度混合关键偶发任务集 Γ 且能够利用高关键层次偶发任务预留的空闲时间.系统处于低模式时,偶发任务开始以速度 S(式(10))执行;当高关键层次偶发任务完成执行时,改变处理器速度进一步降低能耗.系统处于高模式时,高关键层次偶发任务以最大处理器速度 $S_{\max}$ 执行;系统处于低模式时,RHS 算法调度混合关键偶发任务集 Γ 的结果如图 2 所示.

图 2 中黑色箭头代表任务实例的到达时间,τ_1 在时刻 0 以速度 0.97 开始执行且在时刻 1.03 完成执行.RHS 算法利用高关键层次偶发任务 τ_1 预留

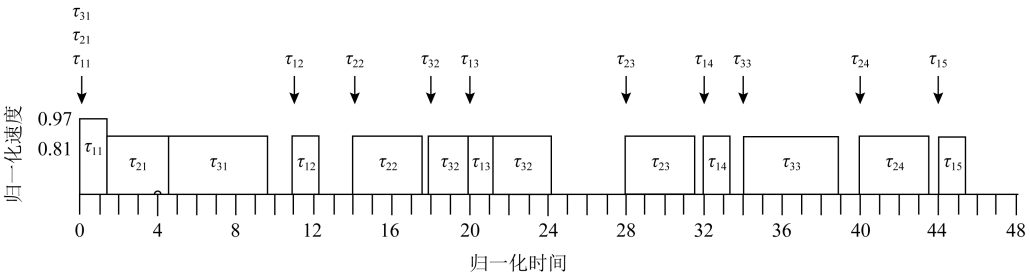

Fig. 2 RHS algorithm schedules mixed criticality sporadic tasks Γ in low mode

图 2 RHS算法在低模式调度混合关键偶发任务集 Γ

的空闲时间调节处理器速度.在时刻 1.03,τ_2 以速度 0.81 开始执行且在时刻 4.73 完成执行.τ_3 在时刻 4.73 以速度 0.81 开始执行且在时刻 9.67 完成执行.在时刻 11,τ_1 以速度 0.81 开始执行且在时刻 12.23 完成执行.τ_2 在时刻 14 以速度 0.81 开始执行且在时刻 17.70 完成执行.τ_3 在时刻 18 开始以速度 0.81 执行,在时刻 20 被 τ_1 抢占;τ_1 在时刻 21.23 完成执行.τ_3 恢复执行且在时刻 24.17 完成执行.剩余的偶发任务以相似的方法调度,不再赘述.

从图 1 中可以看出存在大量的空闲时间区间如[8.24,11],[12.03,14],[17.09,18],[23.15,28],[31.09,32],[33.03,34],[38.12,40],[43.09,44],[45.03,48].在图 2 中依然存在大量的空闲时间区间如[9.67,11],[12.23,14],[17.70,18],[24.17,28],[31.70,32],[33.23,34],[38.94,40],[43.70,44],[45.23,48].产生这些空闲时间的主要原因是偶发任务释放任务实例的时间存在不确定且都大于其最小释放时间,而所计算的能耗感知速度假设偶发任务以其最小释放时间释放任务实例,导致任务的真实利用率低于计算能耗感知速度所使用的利用率.因此,可以利用这些空闲时间进一步降低系统能耗.

问题阐述为:利用 CRMS 调度混合关键偶发任务集 $\Gamma=\{\tau_1,\tau_2,\cdots,\tau_n\}$,计算静态能耗感知速度 S,动态地回收高关键层次偶发任务预留的空闲时间以及偶发任务到达时间不确定所产生的空闲时间,确定偶发任务最终的执行速度 S_c 来降低系统低模式的能耗且满足调度正确标准.

3.2 计算偶发任务空闲时间

混合关键偶发任务集的实时利用率根据算法 1 计算,当 τ_i 释放任务实例且 $\tau_i\in M$ 时,提高偶发任务集的利用率,且将其从集合 M 中移除(算法 1 的步①~④).M 代表可延迟偶发任务集合,该集合的偶发任务其释放 2 个相邻任务实例的时间大于最小释放间隔,初始时 $M=\Gamma$.τ_i 在时刻 r_i+T_i 没有释放任务实例且 $\tau_i\notin M$,降低偶发任务集的利用率,且将 τ_i 加入到集合 M(算法 1 的步⑤~⑧).需要注意的是高关键层次偶发任务 τ_i 有任务实例完成执行,回收其预留的空闲时间,所以其相应的利用率变为 $C_i(\mathrm{LO})/(S\times T_i)$.

算法 1. 利用率更新.

① 事件 1:τ_i 释放任务实例且 $\tau_i\in M$;

② 如果 τ_i 是低关键层次任务,则

$$U=U+C_i(\mathrm{LO})/(S\times T_i);$$

③ 如果 τ_i 是高关键层次任务且其在当前时间之前有任务实例完成执行,则 $U=U+C_i(\mathrm{LO})/(S\times T_i)$,如果其在当前时间之前没有任务实例完成执行,则 $U=U+C_i(\mathrm{LO})/(S\times T_i)+(C_i(\mathrm{HI})-C_i(\mathrm{LO}))/T_i$;

④ 将 τ_i 从集合 M 中移除;

⑤ 事件 2:τ_i 在时刻 r_i+T_i 没有释放任务实例且 $\tau_i\notin M$;

⑥ 如果 τ_i 是低关键层次任务,则

$$U=U-C_i(\mathrm{LO})/(S\times T_i);$$

⑦ 如果 τ_i 是高关键层次任务且其在当前时间之前有任务实例完成执行,则 $U=U-C_i(\mathrm{LO})/(S\times T_i)$,如果其在当前时间之前没有任务实例完成执行,则 $U=U-C_i(\mathrm{LO})/(S\times T_i)-(C_i(\mathrm{HI})-C_i(\mathrm{LO}))/T_i$;

⑧ 将 τ_i 加入到集合 M;

⑨ 事件 3:处理器处于空闲状态,设置 $M=\Gamma$,$U=0$;

⑩ 如果 $U<0$,设置 $U=0$.

3.3 固定优先级混合关键调度算法

本节将提出固定优先级混合关键调度(fixed priority mixed criticality schedule, FPMCS)算法,该算法不仅能够利用静态空闲时间和高关键层次任务预留时间,而且还可以利用偶发任务到达时间

不确定产生的空闲时间来进一步降低能耗.FPMCS算法基于CRMS算法分配任务优先级以及调度任务.FPMCS主要由利用率更新(算法1)、任务调度(算法2)以及速度选择(算法3)构成.

算法2. 任务调度.

① 计算速度S(式(10)),设置$M=\Gamma$,$U=0$,根据CRMS算法分配任务的优先级;

② 将就绪队列设置为空集,将所有没有释放的任务实例放在等待队列中,偶发任务释放实例之后将其放到就绪队列中;

③ 在就绪队列中,选择优先级最高的τ_j执行;

④ 利用算法1和算法3动态决定τ_j的执行速度S_c;

⑤ 如果高关键层次偶发任务τ_j的执行时间超过$C_j(\mathrm{LO})/S_c$,系统切换到高模式;

⑥ 如果τ_j完成执行,将其从就绪队列移动到延迟队列.

算法3. 速度选择.

① 如果高关键层次偶发任务τ_i的第1个任务实例完成执行,则$U=U-(C_i(\mathrm{HI})-C_i(\mathrm{LO}))/T_i$;

② 如果系统处于高模式,则$S_c=S_{\max}$;

③ 如果系统处于低模式,则$S_c=U\times S/F(n)$;

④ 如果$S_c<S_{\min}$,则$S_c=S_{\min}$.

需要注意的是,之所以$S_c=U\times S/F(n)$是因为要确保系统的可调度性,计算出的速度必须满足CRMS调度可行的条件(算法3的步③).

算法1的时间复杂度为$O(1)$,算法2的时间复杂度为$O(n^2)$,算法3的时间复杂度为$O(1)$,因此FPMCS算法的时间复杂度为$O(n^2)$.

3.4 算法实例

利用表1的混合关键偶发任务集Γ来解释FPMCS算法能够取得更好的节能效果.FPMCS算法在区间[0,48]调度混合关键偶发任务集Γ如图3所示:

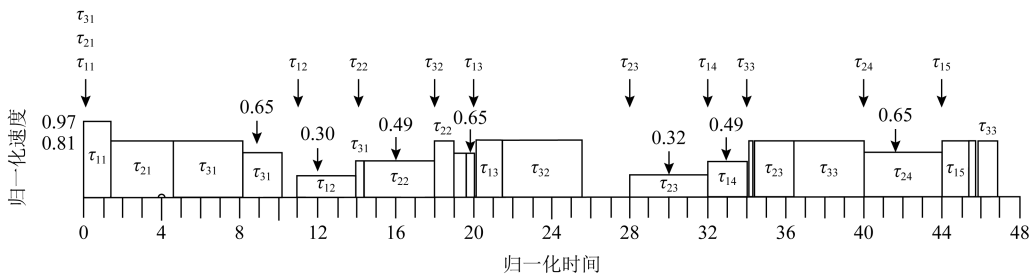

Fig. 3 FPMCS algorithm schedules mixed criticality sporadic tasks Γ in low mode

图3 FPMCS算法在低模式调度混合关键偶发任务集Γ

图3中黑色箭头代表任务实例的到达时间,箭头上方的数字代表任务的执行速度.在时刻0,τ_1以速度0.97开始执行且在时刻1.03完成执行.在时刻1.03,τ_2以速度0.81开始执行且在时刻4.73完成执行.τ_3在时刻4.73以速度0.81开始执行.在时刻8,τ_1没有释放任务实例且其不属于集合M,降低偶发任务集利用率U;因此,τ_3以速度0.65继续执行且在时刻10.08完成执行.在时刻11,τ_1以速度0.30开始执行;在时刻14,τ_2到达且其属于集合M,提高偶发任务集利用率U;因此,在时刻14,τ_1以速度0.49继续执行且在时刻14.20完成执行.τ_2在时刻14.20以速度0.49开始执行.在时刻18,τ_3到达且其属于集合M,提高偶发任务集利用率U;因此,τ_2以速度0.81继续执行.在时刻19,τ_1没有释放任务实例且其不属于集合M,降低偶发任务集利用率U.τ_2以速度0.65执行且在时刻19.51完成执行.在时刻19.51,τ_3以速度0.65开始执行.在时刻20,τ_1到达且其属于集合M,提高偶发任务集利用率U,此时,τ_1抢占τ_3,以速度0.81执行且在时刻21.23完成执行.在时刻21.23,τ_3以速度0.81继续执行且在时刻25.77完成执行.剩余的偶发任务以相似的方法调度,不再赘述.

采用文献[1,18]中功耗模型的参数,即$P_{\mathrm{ind}}=0.1$,$P_{\mathrm{dynamic}}^{\max}=1$,$\theta=0.2$;处理器处于空闲状态的功耗等于0.1.图1的CRMS算法和图2的RHS算法在低模式调度混合关键偶发任务集Γ的能耗分别为36.04和29.89.图3的FPMCS算法在低模式调度混合关键偶发任务集Γ的能耗为26.43.因此,FPMCS算法与CRMS算法和RHS算法相比可以分别节约26.66%和11.58%的能耗.

3.5 可行性分析

FPMCS算法利用CRMS算法分配任务优先级以及调度任务.如果混合关键偶发任务集Γ的利用率不超过$F(n)$,则CRMS算法调度混合关键偶发

任务集 Γ 是可行的.因此,只需证明 FPMCS 算法以速度 S_C 执行,混合关键偶发任务集 Γ 的利用率不超过 $F(n)$.定理 3 将证明 FPMCS 算法是可行的.

定理 3. 混合关键偶发任务集 Γ 的利用率不超过 $F(n)$,FPMCS 算法以速度 S_c 调度混合关键偶发任务集 Γ 是可行的.

证明. 只需证明 FPMCS 算法在超周期内调度混合关键偶发任务集 Γ 是可行的.令 $\beta_{\text{speed}}=\{\beta_1,\beta_2,\cdots,\beta_m\}$ 为所有速度变化区间的集合,其中区间 β_i 开始时刻是在上一个空闲区间的结束和其结束时刻是在下一个空闲区间的开始.因此,β_{speed} 的长度正好等于混合关键偶发任务集 Γ 的超周期.令 $S_{\text{speed}}=\{S_1,S_2,\cdots,S_m\}$ 为与区间 β_{speed} 的相对应处理器速度集合.令 $a_{\text{uti}}=\{a_1,a_2,\cdots,a_m\}$ 为 β_{speed} 的相对应 M 的利用率集合.令 U_i 为 β_i 以速度 S 执行的利用率,U_i 计算为

$$U_i=\sum_{i=1,\tau_i\in M}\frac{C_i(\text{LO})}{T_i\times a_i\times S/F(n)}. \tag{11}$$

在区间 β_i 开始之前,延迟到达任务集合 M 包括所有释放任务实例时间超过其最小时间释放的任务,因此 β_i 的相应利用率 a_i 计算为

$$a_i=\begin{cases}\displaystyle\sum_{i=1,\tau_i\in M}^{n}\frac{C_i(\text{LO})}{T_i\times S}, \text{第 1 个实例完成},\\ \displaystyle\sum_{i=1,\tau_i\in M}^{n}\frac{C_i(\text{LO})}{T_i\times S}+\\ \displaystyle\sum_{i=1,\tau_i\in M\wedge L_i=\text{HI}}^{n}\frac{C_i(\text{HI})-C_i(\text{LO})}{T_i}, \text{没完成},\end{cases} \tag{12}$$

其中,高关键层次任务第 1 个任务实例完成执行之后 a_i 的值为 $\displaystyle\sum_{i=1,\tau_i\in M}^{n}\frac{C_i(\text{LO})}{T_i\times S}$;高关键层次任务第 1 个任务实例的没有完成执行 a_i 的值 $\displaystyle\sum_{i=1,\tau_i\in M}^{n}\frac{C_i(\text{LO})}{T_i\times S}+\sum_{i=1,\tau_i\in M\wedge L_i=\text{HI}}^{n}\frac{C_i(\text{HI})-C_i(\text{LO})}{T_i}$. a_i 的值在高关键层次任务第 1 个任务实例的没有完成执行时大于完成执行的情况.FPMCS 算法中任务速度由利用率决定,利用率越大其速度越大.如果 FPMCS 算法用较小的利用率计算出的速度调度是可行的,那么用较大的利用率计算出的速度调度一定是可行的;将 $a_i=\displaystyle\sum_{i=1,\tau_i\in M}^{n}\frac{C_i(\text{LO})}{T_i\times S}$ 带入式(11),得到:

$$U_i=F(n). \tag{13}$$

区间集合 β_{speed} 的利用率 $U_{\beta_{\text{speed}}}$ 等于子区间与全部区间的比值乘以该子区间的利用率之和,即:

$$U_{\beta_{\text{speed}}}=\sum_{i=1}^{m}\frac{\beta_i\times U_i}{\displaystyle\sum_{k=1}^{m}\beta_k}. \tag{14}$$

由于 $\displaystyle\sum_{k=1}^{m}\beta_k$ 是常数,其值等于任务集的超周期,因此

$$U_{\beta_{\text{speed}}}=F(n). \tag{15}$$

证毕.

4 仿真实验

利用 C 语言实现混合关键偶发任务的调度仿真器来评价所提出算法的性能.该调度仿真器基于固定优先级调度策略.仿真实验中比较 3 种算法:1)CRMS 算法,在低模式所有任务以速度 S 执行;2)RHS 算法,该算法基于 CRMS,在低模式能够利用高关键层次任务预留的空闲时间[17]降低能耗;3)FPMCS 算法,该算法不仅能够利用静态空闲时间和高关键层次任务预留时间,而且还可以利用偶发任务到达时间不确定产生的空闲时间来进一步降低能耗.

通过文献[16-18]的方法产生随机混合关键偶发任务集.和文献[16,18]一样,混合关键偶发任务集 Γ 包含 4 个偶发任务,其中高关键层次偶发任务和低关键层次偶发任务各 2 个.偶发任务 τ_i 的最小释放时间 T_i 服从区间[10,100]的均匀分布,$C_i(\text{LO})$ 服从区间[1,T_i]的均匀分布,低关键层次偶发任务的 $C_i(\text{HI})=C_i(\text{LO})$,高关键层次偶发任务的 $C_i(\text{HI})$ 服从区间[$C_i(\text{LO})$,T_i]的均匀分布.通过修改 $C_i(\text{HI})$ 和 $C_i(\text{LO})$ 使得 $U_{\text{LO}}^{\text{LO}}(\Gamma)$,$U_{\text{HI}}^{\text{LO}}(\Gamma)$,$U_{\text{HI}}^{\text{HI}}(\Gamma)$ 不超过给定的值.由于不同的混合关键偶发任务集有不同的超周期,仿真实验的时间设置为 1 000 000 个时间片.采用文献[1,18]中功耗模型的参数,即 $P_{\text{ind}}=0.1$,$P_{\text{dynamic}}^{\max}=1$,$\theta=0.2$;处理器处于空闲状态的功耗等于 0.1.仿真实验考虑算法在低模式下的能耗,且以 CRMS 算法的能耗作为基准进行归一化.

4.1 $U_{\text{LO}}^{\text{LO}}(\Gamma)$对能耗的影响

将高关键层次任务高模式利用率 $U_{\text{HI}}^{\text{HI}}(\Gamma)$ 设置为 0.4,高关键层次任务集高模式和低模式的比值设置为 1.2;考察低关键层次任务低模式利用率 $U_{\text{LO}}^{\text{LO}}(\Gamma)$ 对算法能耗的影响.为了确保所选取的任务集都是

可调度的,将 $U_{LO}^{LO}(\Gamma)$取值限定为 0.05～0.35 之间,步长设置为 0.05.实验的结果如图 4 所示:

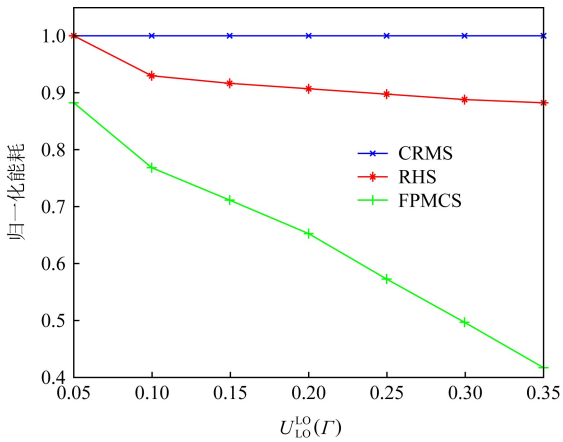

Fig. 4 Effect of $U_{LO}^{LO}(\Gamma)$ on energy consumption of algorithms

图 4 $U_{LO}^{LO}(\Gamma)$对算法能耗的影响

从图 4 可以看出,RHS 和 FPMCS 算法的归一化能耗都受 $U_{LO}^{LO}(\Gamma)$影响.随着 $U_{LO}^{LO}(\Gamma)$的增加,所有算法的能耗都增加.这是因为 $U_{LO}^{LO}(\Gamma)$增加,不仅导致任务的执行时间增加,而且导致任务的执行速度增加.然而,RHS 和 FPMCS 算法能耗增加速度远低于 CRMS 算法.因此,随着 $U_{LO}^{LO}(\Gamma)$的增加,RHS 和 FPMCS 算法归一化能耗降低.不管 $U_{LO}^{LO}(\Gamma)$如何变化,FPMCS 算法的能耗始终都低于其他算法的能耗.这主要得益于 FPMCS 算法能够利用偶发任务到达时间不确定产生的空闲时间降低能耗.通过计算可知,FPMCS 算法与 RHS 算法和 CRMS 算法相比分别节约 30.40%和 35.77%的能耗.

4.2 $U_{HI}^{HI}(\Gamma)$对能耗的影响

将低关键层次任务低模式利用率 $U_{LO}^{LO}(\Gamma)$设置为 0.3,高关键层次任务集高模式和低模式的比值设置为 1.2;考察高关键层次任务高模式利用率 $U_{HI}^{HI}(\Gamma)$对算法能耗的影响.为了确保所选取的任务集都是可调度的,将 $U_{HI}^{HI}(\Gamma)$取值限定为 0.05～0.45 之间,步长设置为 0.05.实验的结果如图 5 所示.

从图 5 可以看出,RHS 和 FPMCS 算法的归一化能耗都受 $U_{HI}^{HI}(\Gamma)$影响.随着 $U_{HI}^{HI}(\Gamma)$的增加,所有算法的能耗都增加.这是因为 $U_{HI}^{HI}(\Gamma)$增加,$U_{HI}^{LO}(\Gamma)$也相应的增加. $U_{HI}^{LO}(\Gamma)$增加不仅导致任务的执行时间增加,而且导致任务的执行速度增加.然而,RHS 和 FPMCS 算法能耗增加速度低于 CRMS 算法.因此,随着 $U_{HI}^{HI}(\Gamma)$的增加,RHS 和 FPMCS 算法归一化能耗降低.不管 $U_{HI}^{HI}(\Gamma)$如何变化,FPMCS 算法的

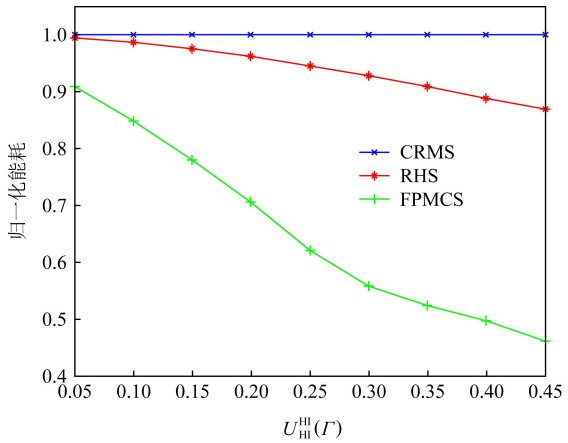

Fig. 5 Effect of $U_{HI}^{HI}(\Gamma)$ on energy consumption of algorithms

图 5 $U_{HI}^{HI}(\Gamma)$对算法能耗的影响

能耗始终都低于其他算法的能耗.这主要得益于 FPMCS 算法能够利用偶发任务到达时间不确定产生的空闲时间降低能耗.通过计算可知,FPMCS 算法与 RHS 算法和 CRMS 算法相比分别节约 30.82%和 34.49%的能耗.

4.3 $U_{HI}^{HI}(\Gamma)/U_{HI}^{LO}(\Gamma)$比值对能耗的影响

将低关键层次任务低模式利用率 $U_{LO}^{LO}(\Gamma)$设置为 0.3,高关键层次高模式利用率 $U_{HI}^{HI}(\Gamma)$设置为 0.4,考察高关键层次任务高模式和低模式利用率比值 $U_{HI}^{HI}(\Gamma)/U_{HI}^{LO}(\Gamma)$对算法能耗的影响.将 $U_{HI}^{HI}(\Gamma)/U_{HI}^{LO}(\Gamma)$的取值范围限定在 1.1～1.9,步长设置为 0.1.实验的结果如图 6 所示:

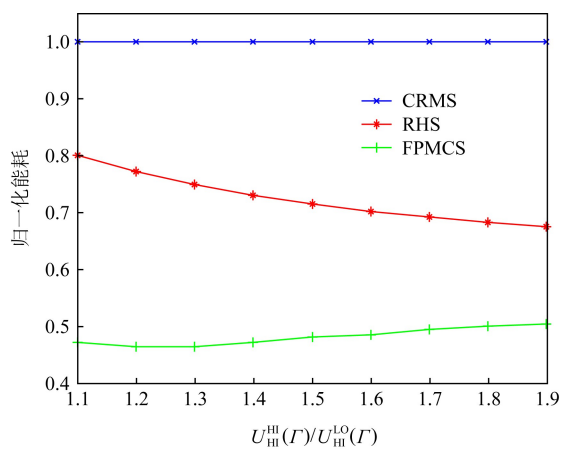

Fig. 6 Effect of $U_{HI}^{HI}(\Gamma)/U_{HI}^{LO}(\Gamma)$ on energy consumption of algorithms

图 6 $U_{HI}^{HI}(\Gamma)/U_{HI}^{LO}(\Gamma)$比值对算法能耗的影响

从图 6 可以看出,RHS 和 FPMCS 算法的归一化能耗都受 $U_{LO}^{LO}(\Gamma)$影响.随着 $U_{HI}^{HI}(\Gamma)/U_{HI}^{LO}(\Gamma)$比值

的增加,所有算法的能耗都降低.这是因为 $U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)$ 固定,$U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)/U_{\mathrm{HI}}^{\mathrm{LO}}(\Gamma)$ 比值增加,任务的执行时间减少,而且相应的速度也同时减少.RHS 算法能耗减少速度慢于 CRMS 算法,所以其归一化能耗随着 $U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)/U_{\mathrm{HI}}^{\mathrm{LO}}(\Gamma)$ 的增加而降低.FPMCS 算法能耗减少速度快于 CRMS 算法,所以其归一化能耗随着 $U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)/U_{\mathrm{HI}}^{\mathrm{LO}}(\Gamma)$ 的增加而升高.不管 $U_{\mathrm{HI}}^{\mathrm{HI}}(\Gamma)/U_{\mathrm{HI}}^{\mathrm{LO}}(\Gamma)$ 比值如何变化,FPMCS 算法的能耗始终都低于其他算法的能耗.这主要得益于 FPMCS 算法能够利用偶发任务到达时间不确定产生的空闲时间降低能耗.通过计算可知,FPMCS 算法与 RHS 算法和 CRMS 算法相比分别节约 33.21%和 51.86%的能耗.

5 结 论

固定优先级混合关键系统偶发任务能耗感知调度是解决系统实时性与低能耗的关键,而偶发任务到达时间不确定是调度过程中面临的一大难题,所提出的 FPMCS 算法通过事件触发的方法实时更新混合关键偶发任务集的利用率,通过该利用率实时计算任务的执行速度,达到充分利用偶发任务到达时间不确定产生的空闲时间降低能耗的目的.此外,FPMCS 算法还可以利用高关键层次任务预留的时间来降低能耗.最后,通过理论分析和仿真实验验证了 FPMCS 算法的可行性且具有良好的节能效果.仿真实验表明:FPMCS 算法比现有的 RHS 算法节约大约 33.21%的能耗.

作者贡献声明:张忆文负责文章的设计与实现;高振国协助完成实验数字分析;林铭炜协助数据的整理.

参 考 文 献

[1] Taherin A, Salehi M, Ejlali A. Reliability-aware energy management in mixed-criticality systems [J]. IEEE Transactions on Sustainable Computing, 2018, 3(3): 195-208

[2] Vestal S. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance [C] //Proc of the 28th Real-Time Systems Symp (RTSS). Piscataway, NJ: IEEE, 2007: 239-243

[3] Baruah S, Chattopadhyay B. Response-time analysis of mixed criticality systems with pessimistic frequency specification [C] //Proc of the 19th Int Conf on Embedded and Real-Time Computing Systems and Applications. Piscataway, NJ: IEEE, 2013: 237-246

[4] Davis I, Altmeyer S, Burns A. Mixed criticality systems with varying context switch costs [C] //Proc of the 24th Real-Time and Embedded Technology and Applications Symp. Piscataway, NJ: IEEE, 2018: 140-151

[5] Baruah S, Bonifaci V, D'Angelo G, et al. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems [C] //Proc of the 24th Euromicro Conf on Real Time System. Piscataway, NJ: IEEE, 2012: 145-154

[6] Liu Di, Guan Nan, Spasic J, et al. Scheduling analysis of imprecise mixed-criticality real-time tasks [J]. IEEE Transactions on Computer, 2018, 67(7): 975-990

[7] Chen Gang, Guan Nan, Liu Di, et al. Utilization-based scheduling of flexible mixed-criticality real-time tasks [J]. IEEE Transactions on Computer, 2018, 67(4): 543-558

[8] Su H, Zhu Dakai, Mosse D. Scheduling algorithms for elastic mixed-criticality tasks in multicore systems [C] //Proc of the 19th Int Conf on Embedded & Real-Time Computing Systems & Applications. Piscataway, NJ: IEEE, 2013: 352-357

[9] Zhang Yiwen, Li Haibo. Energy aware mixed tasks scheduling in real-time systems [J]. Sustainable Computing-Informatics & Systems, 2019, 23: 38-48

[10] Zhang Yiwen. Energy-aware mixed partitioning scheduling in standby-sparing systems [J]. Computer Standards & Interfaces, 2019, 61: 129-136

[11] Zhang Yiwen, Wang Cheng, Lin Changlong. Energy-aware sporadic tasks scheduling with shared resources in hard real-time systems [J]. Sustainable Computing-Informatics & Systems, 2017, 15: 52-62

[12] Zhang Yiwen, Xu Chugui. Low power fixed priority scheduling sporadic task with shared resources in hard real time systems [J]. Microprocessors and Microsystems, 2016, 45: 164-175

[13] Zhang Yiwen, Wang Cheng, Zhang Huizhen. Sporadic tasks scheduling algorithm with resources constraints based on RM scheme [J]. Journal of Huazhong University of Science and Technology: Natural Science Edition 2017, 45(7): 115-121 (in Chinese)
(张忆文, 王成, 张惠臻. 基于 RM 策略的资源受限偶发任务调度算法[J]. 华中科技大学学报: 自然科学版, 2017, 45(7): 115-121)

[14] Zhang Yiwen, Guo Ruifeng, Deng Changyi. Low power scheduling algorithm for mix tasks based on constant bandwidth server [J]. Journal of Computer Research and Development, 2015, 52(9): 2094-2104 (in Chinese)
(张忆文, 郭锐锋, 邓昌义. 常带宽服务器混合任务低功耗调度算法[J]. 计算机研究与发展, 2015, 52(9): 2094-2104)

[15] Zhang Yiwen. Energy-aware fixed-priority scheduling for periodic tasks with shared resources and IO devices [J]. International Journal of Embedded System, 2020, 12(2): 166-176

[16] Huang P, Kumar P, Giannopoulou G, et al. Energy efficient DVFS scheduling for mixed-criticality systems [C] //Proc of the 14th Int Conf on Embedded Software. New York: ACM, 2014: 11-20

[17] Ali I, Seo J, Kim K H. A dynamic power-aware scheduling of mixed-criticality real-time systems [C] //Proc of the 13th Int Conf on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing. Piscataway, NJ: IEEE, 2015: 438-445

[18] Zhang Yiwen. Energy-aware mixed-criticality sporadic task scheduling algorithm [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021, 40 (1): 78-86

[19] Sruti S, Bhuiyan A, Guo Zhishan. Work-in-progress: Precise scheduling of mixed-criticality tasks by varying processor speed [C] //Proc of the 39th Real-Time Systems Symp (RTSS). Piscataway, NJ: IEEE, 2018: 173-176

[20] Jiang Wei, Pan Xiong, Jiang Ke, et al. Energy-aware design of stochastic applications with statistical deadline and reliability guarantees [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 38 (8): 1413-1426

[21] Safari S, Hessabi S, Ershadi G. Less-mics: A low energy standby-sparing scheme for mixed-criticality systems [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(12): 4601-4610

[22] Baruah S, Burns A. Sustainable scheduling analysis [C] // Proc of the 27th Real-Time Systems Symp(RTSS). Piscataway, NJ: IEEE, 2006: 159-168

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

[24] Wang Yaqin, Ruan Youlin. Flexible mixed-criticality task scheduling and energy optimization [C] //Proc of the 4th Information Technology, Networking, Electronic and Automation Control Conf. Piscataway, NJ: IEEE, 2020: 602-606

[25] Ma Huirong, Chen Xu, Zhou Zhi, et al. Dynamic task offloading for mobile edge computing with green energy [J]. Journal of Computer Research and Development, 2020, 57 (9): 1823-1838 (in Chinese)
(马惠荣, 陈旭, 周知, 等. 绿色能源驱动的移动边缘计算动态任务卸载[J]. 计算机研究与发展, 2020, 57(9): 1823-1838)

Zhang Yiwen, born in 1988. PhD, associate professor. Member of CCF. His main research interests include real-time system and low-power scheduling.
张忆文,1988 年生.博士,副教授,CCF 会员.主要研究方向为实时系统、低功耗调度.

Gao Zhenguo, born in 1976.PhD, professor. Member of CCF. His main research interests include wireless networks, computer vision, artificial intelligence.
高振国,1976 年生.博士,教授,CCF 会员.主要研究方向为无线网络、计算机视觉、人工智能.

Lin Mingwei, born in 1985. PhD, professor. His main research interests include fuzzy decision-making analysis and storage performance optimization.
林铭炜,1985 年生.博士,教授.主要研究方向为模糊决策分析、存储性能优化.