DOI:10.13196/j.cims.2018.10.003

基于滑动窗口的低能耗高可靠调度算法

邓昌义[1,2],郭锐锋[2,3],吴昊天[2,3],彭阿珍[2,3],杜少华[4],盖荣丽[5]

(1.国家工业信息安全发展研究中心 系统所,北京 100040;2.中国科学院大学,北京 100049;
3.中国科学院 沈阳计算技术研究所,辽宁 沈阳 110168;4.沈阳高精数控智能技术股份有限公司,辽宁 沈阳 110168;
5.大连大学 信息工程学院,辽宁 大连 116021)

摘 要:针对开放式数控系统的高功耗和可靠性差的问题,通过优化空闲时间分配策略,实现在保证开放式数控系统可靠性前提下最小化系统能耗。提出基于滑动窗口的低能耗调度算法(LPRSW),以在保证系统可靠性的前提下降低系统功耗,同时根据任务出错场景的不同,将算法分为 LPRSW-H 算法和 LPRSW-A 算法,前者以最高速度恢复出错任务,后者采用动态电压调节后的速度恢复出错任务。在此基础上,提出一种基于滑动窗口的低能耗与可靠性协同优化调度算法,该算法对任务容错采用共享全局空闲时间代替预先给每任务都分配一个备份任务的方法,获得更多空闲时间用于降低系统功耗,同时空闲时间—能耗因子用于将空闲时间更合理地分配给后续任务,实现可靠性和低能耗之间协同优化。通过实验对比,COSALPRSW 算法平均比 LPRSW-A 和 LPRSW-H 算法节约 0～31.62%和 0～44.4%的能耗,验证了所提方法的有效性。

关键词:开放式数控系统;动态电压调节;实时调度;能耗;可靠性

中图分类号:TP311 **文献标识码**:A

Low-power and high-reliability scheduling algorithm based on sliding window

DENG Changyi[1,2], *GUO Ruifeng*[2,3], *WU Haotian*[2,3], *PENG Azhen*[2,3], *DU Shaohua*[4], *GAI Rongli*[5]

(1. Institute of Systems Research, China Industrial Control Systems Cyber Emergency Response Team, Beijing 100040, China;
2. University of Chinese Academy of Sciences, Beijing 100049, China;
3. Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China;
4. Shenyang High-Precision CNC Intelligent Technology Co., Ltd., Shenyang 110168, China;
5. Department of Information Engineering, Dalian University, Dalian 116021, China)

Abstract: To solve the problem of high energy consumption and poor reliability of open CNC systems, through optimizing the slack time allocation issues, a set of real-time tasks in the open CNC system was researched to minimize energy consumption while maintaining the reliability of open CNC system. A Low Power and Reliability Based on Sliding Window (LPRSW) algorithm was proposed, which could reduce the system power consumption while ensuring the system reliability. With the different mission scenarios, LPRSW algorithm was divided into LPRSW-H and LPRSW-A algorithm, the former used the highest speed to tolerate faults, while the latter tolerated faults task by dynamic voltage scaling technology. On this basis, a Cooperative Optimal Scheduling Algorithm for Low Power and Reliability Based on Sliding Window (COSALPRSW) algorithm was built, which allocated global shared slack time to backup tasks instead of assigning a backup task to each task, and more slack time was available to reduce system power consumption. In addition, the slack-energy factor was used to rationally allocate slack time to subsequent tasks, which realized the system's reliability and low power consumption co-optimization. Experimental results

收稿日期:2017-05-22;**修订日期**:2017-10-30。Received 22 May 2017;accepted 30 Oct. 2017.

基金项目:国家科技重大专项资助项目(2014ZX04014-021);国家自然科学基金资助项目(61602074)。**Foundation items**: Project supported by the National Science and Technology Major Project of the Ministry of Science and Technology, China(No. 2014ZX04014-021), and the National Natural Science Foundation, China(No. 61602074).

showed that COSALPRSW algorithm could average 0～31.62% and 0～44.4% energy savings over LPRSW-A and LPRSW-H algorithms respectively.

Keywords: open CNC system; dynamic voltage scaling; real-time scheduling; energy consumption; reliability

0 引言

随着计算机系统,特别是网络技术的发展,计算机集成制造系统的实现形式从集中型向以系统应用平台化、系统结构开放化、制造资源服务化、管理决策智能化和绿色化制造方向发展[1],其变化不仅是技术上的升级,更符合实际生产的需要。在实际工业生产中,数控机床的种类很多,许多机床是面向特定用户的需要开发的。控制器厂家提供的数控系统大多是全功能的数控系统,但实际使用中许多参数需要根据机床的实际情况进行配置和二次开发,厂家提供的数控系统功能不一定能够满足机床制造企业的需要。在这种情况下,具有"开放性"的数控系统成为企业需求,即不仅有高度模块化的结构,还可以重新配置、修改、扩充和改装的开放式数控系统。

作为一种典型的实时系统,开放式数控系统是数控机床的核心部件。数控机床在数控系统的控制下,根据编程指令设定的速度运动实现高速高精加工[2]。数控系统不但要求在截止期限内完成任务,而且要保证任务的正确执行。当前,快速发展的技术和复杂制造工艺已经对处理器可靠性产生潜在影响[2-3],宇宙射线也增加了处理器出错的可能性[4],同时高能耗导致系统运行在较高的温度环境中,进一步降低了系统的可靠性。因此,设计满足高可靠低能耗的开放式数控系统成为一个非常重要的问题。为方便快捷地开发出专用控制器,允许系统进行二次开发,文献[3]提出开放式专用数控系统的核心模型,包括功能模型、数据模型、结构模型和行为模型,该模型采用嵌入式多 CPU 架构实现了专用数控系统的分布型操作;文献[4]提出一种面向开放式数控系统、基于预分配的空闲挪用算法。算法开始前,采用预分配算法为实时周期任务预留处理器时间,使得在调度周期任务时空闲时间尽可能提前。在调度非周期任务上,采用预留处理器最大可用空闲时间机制实现以较小的计算和较低存储开销获得最短非周期任务响应时间。

动态电压调整(Dynamic Voltage Scaling, DVS)是嵌入式实时系统中普遍采用的一种节能技术[5],它通过降低电路的供电电压和运行频率来降低动态能耗,许多主流的商业处理器,如 Intel XScale[6]、AMD G-series[7]及龙芯[8]系列的多核处理器均支持 DVS。通过 DVS 改变电压和频率会带来时间、能耗的开销,但这些开销非常小。动态功耗管理(Dynamic Power Management, DPM)主要用于降低处理器之外,如内存、I/O 等模块的能耗[9],当这些设备模块响应处理请求时,为正常工作和耗电状态;当设备空闲时,进入低能耗的睡眠状态。但采用 DPM,处理器状态切换造成的时间和能消耗不可忽视。因此,采用 DVS 或 DPM 降低能耗时,必须同时考虑实时性和可靠性限制。为了保证实时任务在出现系统软/硬件故障后仍能在截止期限前完成,需要为实时嵌入式系统提供具有实时性的容错保障,从而确保系统的可靠性不受影响。主/副版本备份技术(Primary/Backup copy, P/B)的每个运行任务包含主、副 2 个相互独立的任务[10],当主版本任务执行发生错误时可以启动执行副版本任务,从而保证系统的可靠性不受影响。

文献[4]提出一种基于预分配的空闲挪用算法,用于调度开放式数控系统的混合任务。算法首先采用预分配算法为实时周期任务预留处理器时间,将调度周期任务的空闲时间尽可能提前。在调度非周期任务上,采用最大的可用空闲时间机制,实现以较小的计算和较低存储开销下获得最短的非周期任务响应时间。文献[8,11]提出一种基于能耗的调度算法,该算法以最小化能耗为目标同时满足处理器吞吐量和响应时间的要求。文献[9,12]提出一种能耗管理算法,使用流水线和并行处理技术调度单处理器上的任务。在容错方面,要实现对瞬态故障的容错,需要故障检测机制[13-14]。故障检测机制需要满足数控系统高可靠性需要,文献[10,13]提出基于 N 模块冗余技术,该技术不需要额外硬件检测故障,而是使用多数投票的方法进行故障检测和容错。因为所有模块不可能同时出现相同的错误,所以多数投票可以实现对系统故障的检测和容错[13],但 N 模块冗余需要消耗更多的能耗开销来实现容错[14]。Melhem 等[15]利用系统的空闲时间,在 DVS 降低能耗的同时基于检查点技术实现容错;Zhu 等[16]提出可靠性感知能耗管理(Reliability-Aware Power

Management, RAPM),系统的可靠性定义为每个任务可靠性的乘积,RAPM 的目标就是在保证系统原有可靠性的同时利用 DVS 节能。RAPM 的基本思路是,对于每一个电压调整的任务,在满足实时性约束下另外调度一个以最高电压及 CPU 速率的恢复任务,从而保证该任务的原有可靠性。Zhao 等[17]提出利用预留部分全局空闲时间代替给每一个任务备份空闲时间的节能机制,通过减少任务备份数量实现低能耗和可靠性的平衡,但是所提的 GSHR(generalized shared recovery)算法只能在一个范围内保证系统的可靠性,不具有一般性。

基于以上研究,本文首先提出通用任务模型的可靠性调度算法,该模型充分考虑系统可靠性,同时考虑降低系统功耗和保证系统可靠性协同调度,实现系统低能耗和高可靠的目的。在此基础上,提出基于滑动窗口的低能耗调度算法(Low Power Scheduling Algorithm Based On Sliding Window, LPRSW),并根据任务出错场景将算法分为 LPRSW-A 和 LPRSW-H 算法。其次,依据出错任务的特点,优化备份任务数量,提出基于滑动窗口的低能耗与可靠性协同优化调度算法(Cooperative Optimal Scheduling Algorithm for Low Power and Reliability Based on Sliding Window, COSALPRSW)。该算法以提高系统的可调度性为目标进行任务的分配,通过备份任务共享空闲时间和优化后续任务的执行速度,在不增加硬件代价的前提下实现系统低能耗高可靠的目标。最后,通过实验结果表明,所提出的算法都能满足系统可靠性的要求。

1　系统模型与功耗模型

1.1　实时任务模型

数控系统的加工任务包括处理轨迹插补、位置控制、运动控制等周期性任务。考虑有 n 个实时任务组成的任务调度模型 $J=(J_1,J_2,\cdots,J_n)$,J_i 表示在一个任务集合的第 i 次调用。任务的参数用一个三元数组表示为 $J=(a_i,c_i,d_i)$,各参数定义如下:

a_i 表示任务 J_i 将要执行的时间;

c_i 表示 J_i 在最大速度 $s_{\max}$ 下的最坏执行时间,$s_{\max}$ 表示处理器能够提供的最大速度;

d_i 表示任务 J_i 的绝对截止期限。

该任务模型具有通用性,可以很好地拓展到其他实时任务模型,例如周期任务模型和非周期任务模型。所有的任务相互独立,采用 EDF(earliest deadline first)调用策略。

1.2　功耗模型

假设处理器速度频率或速度可以连续调节;拓展算法支持处理器在离散速度中调节;当任务 J_i 在速度 S_i 下执行时,需要的执行时间为 c_i/s_i;整个任务集合对应的速度可以表示为 $S=\{s_1,s_2,s_3,\cdots,s_n\}$,这里 S_i 是任务 J_i 运行时对应的速度。本文采用文献[17]的功耗模型,系统的功耗为

$$\begin{aligned} P &= P_s+h(P_{ind}+P_{dep}) \\ &= P_s+h(P_{ind}+C_{ef}s^m)。 \end{aligned} \tag{1}$$

式中:P_s 为系统静态功耗,当整个系统处于关闭状态时 $P_s=0$,静态功耗主要包括系统的基本电路功耗,维持系统的时钟、主存和 I/O 设备在睡眠状态下的功耗;P_{ind} 为与处理器频率无关的功耗,主要包括不收 CPU 频率变化或者电压变化影响的功耗,如主存储器和外部设备的漏电功耗;P_{dep} 为依赖于处理器频率的动态功耗,包括 CPU 功耗,以及其他依赖处理器速度而产生的功耗;C_{ef} 为系统有效的负载电容;m 为动态指数($2\leqslant m\leqslant 3$);h 为系统状态,当系统处于活动状态时 $h=1$,当系统处于睡眠状态时 $h=0$。实际 CPU 能提供的频率$\{f_1,f_2,\cdots,f_{\max}\}$和电压$\{v_1,v_2,\cdots,v_{\max}\}$是离散的,因此提供的速度也是离散的。对频率进行归一化处理,$s_{\min}=\dfrac{f_1}{f_{\max}}$,$s_{\max}=\dfrac{f_{\max}}{f_{\max}}=1$,因此速度的取值范围可以在区间$[s_{\min},\cdots,1]$内的离散点选取。当系统处于活跃状态时,一个任务在频率 f_i 下的能耗可以表示为[17]

$$\begin{aligned} E_i(f_i) &= (P_{s,i}+h(P_{ind,i}+P_{dep,i}))\cdot\frac{c_i}{f_i} \\ &= P_{s,i}\cdot\frac{c_i}{f_i}+h\cdot P_{ind}\cdot\frac{c_i}{f_i}+h\cdot C_{ef}\cdot c_i\cdot f^{m-1}。 \end{aligned} \tag{2}$$

1.3　可靠性模型

研究表明,发生瞬时错误的概率比永久错误的概率大,瞬时任务出错概率符合泊松分布[17]。对于支持动态电压调节的实时系统,考虑到调节电压对瞬时错误产生的不利影响,由瞬时错误引起的任务平均出错率和处理器调整频率 f(对应电压 V)之间的关系可以表示为[17]

$$\lambda(f)=\lambda_0\cdot g(f)。 \tag{3}$$

式中:λ_0 为在频率 $f_{\max}$ 最大情况下任务平均出错率。这里 $g(f_{\max})=1$。低频率或低电压将导致由瞬时错误引起的任务错误率增加。因为 $f<f_{\max}$,所以 g

$(f)>1$。

本文考虑由瞬时错误引起的任务错误指数评价模型，根据文献[17]有

$$g(f)=10^{\frac{d\cdot(1-f)}{1-f_{\min}}}。\tag{4}$$

式中 $d(>0)$是一个常量，表示任务对处理器速度调节出错的敏感度，即当处理器处于最低的频率工作时，具有最高的失效率 $\lambda_{\max}=\lambda_0\cdot 10^d$。瞬时错误引起的任务出错可以使用接受测试(Acceptance Test，AT)的方法[19]进行故障检测。每一个任务执行完毕时向对应的备份任务发送消息，通知主版本执行完毕，如果备份任务在指定的时间内仍未收到任务发来的消息，则认为任务所在的处理器出现故障。一旦发现故障，则将执行备份任务。

系统的可靠性为所有任务成功正确执行的概率。一个任务按最坏时间 c_i 执行，在处理器频率 f 下的可靠性为

$$R_i(f_i)=e^{-\lambda(f_i)\cdot\frac{c_i}{f_i}}。\tag{5}$$

规定 $R^j(T_1,\cdots,T_n)$表示系统的可靠性，即每一个任务成功执行的可能性。如果任务出错，则使用空闲时间恢复第 j 个出错的任务。规定 R_g 表示系统要保证的稳定性。

当整个系统不存在出错任务时，可靠性为 $R^0(J_1,\cdots,J_n)=\prod_{i=1}^{n}R_i(f_i)$。当系统出现一个任务需要容错时，可靠性为

$$R^1(J_1,\cdots,J_n)=R_1(f_1)R^1(J_2,\cdots,J_n)+(1-R_1(f_1))R_1(f_{\max})R^0(J_2,\cdots,J_n)。\tag{6}$$

推广到一般模型，当有 j 个任务需要容错时，系统的可靠性为

$$R^j(J_1,\cdots,J_n)=R_1(f_1)R^j(J_2,\cdots,J_n)+(1-R_1(f_1))R_1(f_{\max})R^{j-1}(J_2,\cdots,J_n)。\tag{7}$$

式中：加号前半部分表示任务 J_1 成功执行，余下的 J 个备份任务可以给后续任务使用；加号的后半部分表示 J_1 执行失败，后续任务需要一个备份任务容错。系统的可靠性 $R^j(J_1,\cdots,J_n)$可以使用动态规划快速计算出，线性渐进复杂度为 $O(J\cdot n)$。

1.4 问题定义

低能耗和可靠性协同系统优化算法的问题可以表示为：在保证系统可靠性前提下找到最优的任务备份数，同时分配调整后的处理器频率使得总能耗 E 最小。

$$\text{最小化 } E=\sum_{i-1}^{n}E_i(f_i)=P_{s,i}\cdot\frac{c_i}{f_i}+h\cdot P_{ind}\cdot\frac{c_i}{f_i}+h\cdot C_{ef}\cdot c_i\cdot f^{m-1}$$

；同时满足约束条件：

$$R^j(J_1,\cdots,J_n)\geqslant R_g;\tag{8}$$

$$\sum_{i=1}^{n}\frac{c_i}{f_i}\leqslant D^*。\tag{9}$$

式中 D^* 为执行任务集合 J 所需的总时间。显然，用于容错的空闲时间越多，系统的可靠性 R_g 越容易保证。当考虑实时任务的截止期限时，系统的可靠性和低功耗在空闲时间上存在竞争关系，因为每一个备份任务需要占用空闲时间，处理器通过调节频率获得低能耗同样有赖于空闲时间的分配。

2 低能耗与可靠性优化调度算法

2.1 动态滑动窗口调度任务机制

根据文献[20]，在 LPEDF 算法基础上进行拓展，使其成为一个能够容错的低能耗调度 LPRSW 算法。

定义 1 给出一个实时任务集 J，$J(I)$表示集合中所有任务完成所需要的最长时间区间，$I=[t_s,t_f]$，$J(I)=\{J_i\mid t_s\leqslant a_i<d_i\leqslant t_f\}$；负载 $W(I)$是时间间隔 $I=[t_s,t_f]$中所有任务的执行时间之和，$W(I)=\sum_{J_i\in J(I)}c_i$；时间区间 I 的强度定义为 $S(I)=W(I)/L(I)$，$S(I)$表示处理器速度的下界。这里 $L(I)$是时间区间 I 的长度，$L(I)=t_f-t_s$；将 $I=[t_s,t_f]$称作滑动窗口，如果该滑动窗口有最高的强度即任务的执行最大速度，则 t_s 和 t_f 是相应任务的初始到达时间和最后截止期限。

在一个滑动窗口内容错相关的开销可以定义为 $W_{ft}(I)=W_R(I)+W_{TO}(I)$，其中：$W_R$ 表示在最坏情况下用来恢复出错任务的预留负载；$W_R=\sum_{R_i\in R(I)}R_i$，$W_{TO}(I)$表示在正常任务中发现出错任务时的转换开销，$W_{TO}(I)=\sum_{J\in J(I)}TO_i$。

基于以上定义，给出一个实时任务集合 J，LPRSW 算法能够在保证系统可靠性的前提下降低系统能耗，算法步骤如下：

(1)根据定义 1 确定滑动窗口。

(2)调整窗口大小，将所有的任务纳入窗口中，设置所有任务的速度为 $S(I)$，调整释放时间和截止期限。特别地，调整滑动窗口任务集合 $J-J(I)\rightarrow J$ 按照下面规则进行：如果 $d_i\in[t_s,t_f]$，则滑动窗口更

新 $d_i \rightarrow t_s$;如果 $a_i \geqslant t_f$,则 $a_i-(t_f-t_s)$;如果 $a_i \in [t_s,t_f]$,则 $a_i \rightarrow t_s$;如果 $d_i \geqslant t_f$,则 $d_i-(t_f-t_s)$。

(3)当任务执行出现错误时,启动容错机制同时增加滑动窗口负载。重新计算出错任务的强度,用 $S_m(I)$代替 $s(I)$,

$$S_m(I)=\frac{W(I)+R_1+R_2+\cdots+R_k}{L(I)-K\times TO_x}。\quad(10)$$

(4)重复以上步骤直到集合 J 为空。

根据公式(10),在任务集合中有 K 个错误需要容错的条件限制下,任意一个采用最早截止期限优先(Earliest Deadline First,EDF)调度策略的实时任务集合可调度的条件为

定理 1　文献[21]给出一个实时任务集合 J,在 $S_{\max}=1$ 下实现 K 个容错。如果对于每一个滑动窗口 I,

$$\frac{W(I)+W_{ft}(I)}{L(I)}\leqslant 1,\quad(11)$$

则任务集合是可调度的。注意到,当一个任务出错后,LPRSW 执行一个备份任务可以有两种选择,第一种是使用调节后的处理器速度策略 LPRSW-A,第二种是使用处理器最高速度策略 LPRSW-H。下面通过实例说明两种算法在不同任务下的节能效果。使用算法 LPRSW-H 时,滑动窗口的强度表示为

$$S_e=\frac{W(I)}{L(I)-W_{ft}(I)}。\quad(12)$$

可以明显看出在滑动窗口 I 中,如果 $W(I)+W_{ft}(I)\leqslant L(I)$,则 $S_e\leqslant S_m$。该方法的优势在于它需要很少的备份资源去保证任务容错,因此可以在出错率较低情况下使用该策略。为了进一步说明,考虑任务集合调度实例(如表 1),令 $m=3$,$P_{md}=0.02$,$C_{ef}=1$,简单起见,时间和能耗转换开销忽略不计。LPRSW-A 算法(如图 1a)调度任务集合的能耗为 11.06,LPRSW-H 算法(如图 1b)能耗消耗为 11.5,LPRSW-A 消耗相对较少的能耗。然而,当系统不存在出错任务时(如图 1c),LPRSW-A 算法消耗的能耗为 5.53,LPRSW-H(图 1d)消耗的能耗为 4.36,反而 LPRSW-A 消耗更多的能耗。由于出错率在实际系统中通常较低,LPRSW-H 采用处理器最大速度执行出错任务,相对占用的空闲时间较少。因此,在降低能耗消耗上,LPRSW-H 相比 LPRSW-A 更有优势。

表 1　任务集合调度实例

J	a_i	c_i	d_i
J_1	0	2	10
J_2	8	3	14
J_3	10	2	20

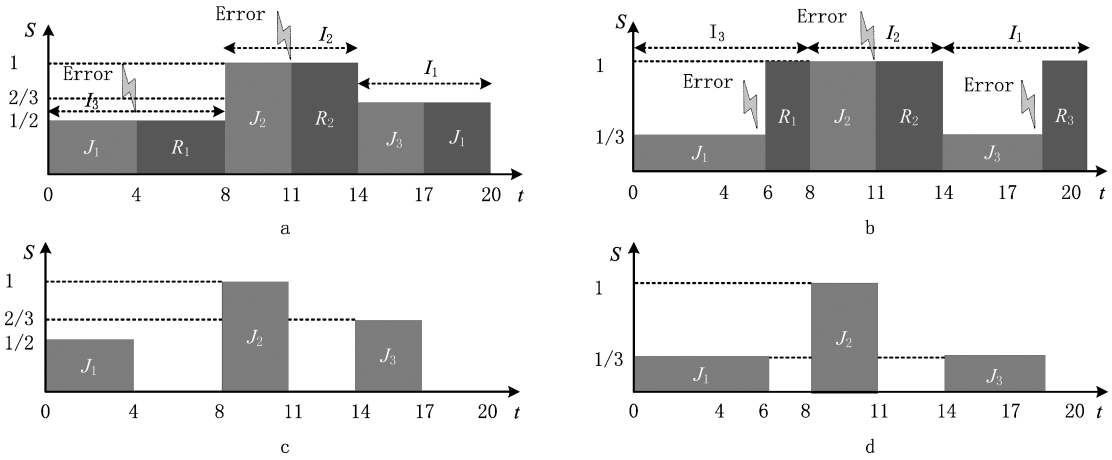

图1　任务集合在LPRSW-A和LPRSW-H算法下的调度过程

为了满足实时任务的截止期限,上述方法每次迭代调整滑动窗口大小时,假设在滑动窗口中按最坏执行时间,所有 k 个出错任务将影响执行时间最长的任务。这个假设相对比较悲观,因为每一个滑动窗口需要计算满足 k 个容错所需的预留资源,有可能导致一个可调度任务集合不可调度。使用下面的例子进行说明。

根据表 2,考虑一个系统中有两个确定的任务需要调度(如图 2),最多有一个任务需要容错。为了简单阐述,假定发现任务出错的开销为 0。根据 LPRSW 算法,第一个滑动窗口为[3,7],根据式(12)其强度为 1。使用任务 J_2 调整窗口[2,6]后,

d_1 调整为 3，则第二个滑动窗口为[0，2]，其强度为 2。调度过程如图 2 所示，这里 I_1 和 I_2 表示第一个和第二个滑动窗口。可见 $S_e(I_2)$比 $S_e(I_1)$大，这种情形为单调性优先级翻转，更重要的是，$S_e(I_2)$超过了系统中的最高速度($S_{\max}=1$)，因此所需要的速度是不可能获得的。事实上，任务集合在速度低于 1 下是可以调度的。从以上讨论可以看到，需要考虑容错的能耗最小化问题不能简单地修改 LPEDF(low power earliest deadline first)算法来实现，需要在调度过程中保证调度结果是有效的。

表 2 任务集合调度实例

J	a_i	c_i	d_i
J_1	0	2	5
J_2	2	2	6

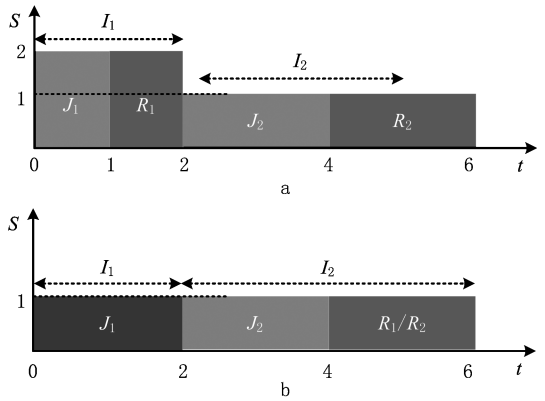

图2 任务集合出现优先级翻转实例

为解决单调优先级反转问题，观察发现反转的滑动窗口一定和之前迭代的窗口相邻。具体而言，有以下引理：

引理 1 LPRSW 算法从 $i_{th}\sim(i-1)_{th}$ 迭代，相应的两个窗口为 I_i 和 I_{i-1}。如果 $S_e(I_i)>S_e(I_{i-1})$，则 I_i 和 I_{i-1}是相邻的窗口。

证明 当调整窗口 I_{i-1}时，在没有重叠的窗口中工作负载将不发生变化，仅当 I_{i-1}中重叠部分将发生小部分改变，改变量为 Δ，$0<\Delta\leqslant L(I_{i-1})$；因此，在下一次迭代中窗口强度将增加。

正如引理 1 的证明，当调整滑动窗口重叠部分中的空闲时间，并用于备份任务时，滑动窗口将发生反转，因此窗口调整将减少这些任务的执行时间。为了消除这种反转，将这些任务融合到之前的窗口中，方法如下：

引理 2 LPRSW 算法从 $i_{th}\sim(i-1)_{th}$ 迭代，相应的两个窗口为 I_i 和 I_{i-1}。如果 $S_e(I_i)>S_e(I_{i-1})$，则在 I_i 和 I_{i-1}之间维持任务集合可调度的最低恒速是 $S_e(I_{i-1})$。

证明 在调整滑动窗口 I_{i-1}前，所有在任务集合 J 剩下的任务在恒速 $S_e(I_{i-1})$将是可调度的。因此，融合的两个窗口 I_i 和 I_{i-1}后任务集合将是可调度的。

定理 2 LPRSW 算法滑动窗口第 1，2，3，…迭代的强度表示为 S_{e1}，S_{e2}，S_{e3}，…。S_{e1}是整个任务集合在错误不超过 K 个情况下，保证任务不发生错过截止期限最低恒定速度。

证明 定理 2 可以被定理 1 证明。在第一次 LPRSW 迭代中，考虑滑动窗口的定义，由于 $S_{e1}\geqslant S_e(I)$，有$\frac{W(I)}{S_{e1}}\leqslant\frac{W(I)}{S_e}$。将式(12)带入上式不等式的右边，两边同时加上 $W_{ft}(I)$，有$\frac{W(I)}{S_{e1}}+W_{ft}(I)\leqslant L(I)$，因此任务集合在速度 S_{e1}下是可调度的。

此外，假设 S_{e1}是滑动窗口 I_1 的速度，则 $S_{e1}=\frac{W(I_1)}{L(I_1)-W_{ft}(I_1)}$，$S^*$ 是任务集合保证可调度的最小速度，有 $S^*<S_{e1}$。可以调节滑动窗口 I_1 的工作负载，如$\frac{W(I_1)}{S^*}+W_{ft}(I_1)>\frac{W(I_1)}{S_{e1}}+W_{ft}(I_1)=L(I_1)$，而这违反了定理 1 的可调度条件。

定理 3 考虑 S_{e1}，S_{e2}，S_{e3}，…作为 LPRSW 算法滑动窗口第 1，2，3，…迭代的速度，有 $S_{e1}\geqslant S_{e2}\cdots\geqslant S_{em}$。

证明 因为优先级反转在算法 LPRSW 中被消除，后续滑动窗口的关系可以很容易地确定。

如果 LPRSW 算法能够满足以下两个条件，则可以保证任务的最后截止期限：

(1)不超过 K 个任务出错。

(2) $\forall_i\in[1,m]$，m 表示总的迭代次数，有 $S_{ei}\leqslant 1$。

在 LPRSW 算法中，一个滑动窗口 I_i 有专属的预留空间给任务执行和任务备份。高优先级任务 J_h 有可能和 I_i 发生重叠，并将强制在 I_i 之前完成。类似地，低优先级任务 J_l 可能和 I_i 发生重叠执行，窗口 I_i 通过调整任务的到达时间和截止时间消除低优先级任务的执行。因此，为了证明定理 3，仅需证明如果设置处理器速度为 S_{ei}，即整个窗口 I_i 的速度，在最坏情况下(发生 K 个任务出错)所有任务

是可调度的条件为 $S_{ei}\leqslant 1$。

通过反证法证明:

当处理器速度设定为 S_{ei},令 $J_c=(a_c,c_c,d_c)\in J(I_i)$错过其截止期限,则能够找到一个时刻 $t\leqslant a_c$,这样对于窗口 $I'=[t,d_c]$,有 $\frac{W(I')}{S_{ei}}+W_{ft}(I')>L(I')$。因为 $S'=\frac{W(I')}{L(I')-W_{ft}(I')}>S_{ei}$,且 $I'\in I_i$。与假设矛盾,所以 I'是一个滑动窗口。证毕。

算法 1　LPRSW。

```
1.  任务集合初始化输入:
2.  任务集合:J={J1,J2,…Jn};
3.  处理器支持的最小运行速度:Smin
4.  输出:{S1,S2,…}
5.  Si=Smax,for i=1,2,…,n;
6.  p=1;{初始化滑动窗口下标}
7.  While J≠Φ do
8.  遍历任务集合 J,确定下一个滑动窗口 I 和对应的处理器速度 S,p++;
9.  If S<Smin then
10. Si=Smin,∀Ji∈J;
11. break;
12. End if
13. If S>Sp-1 AND p>1 then
14. Ip&Ip-1Ip//从上次迭代中恢复任务的信息,将滑动窗口 Ip 和 Ip-1融合成一个滑动窗口;消除优先级反转。
15.  p--;
16. End if
17. End While
18. If 任务出错
19. 根据 LPRSW 算法第 4 步启动容错机制,更新备份任务的新速度
20. End if
21. Return{S1,S2,…}
```

其中:第 8 行表示当前初始滑动窗口和它的速度;第 9～12 行检查是否当前速度小于最小速度,如果小于则终止迭代;第 13～16 行表示当出现反转窗口时即消除它;第 18～20 行表示当任务出错时,重新计算备份任务的速度。LPRSW 算法的复杂性主要来自第 8 行滑动窗口的计算,因此算法时间复杂度为 $O(n^2)$。

2.2　多任务下的共享空闲时间分配

由于所有任务在 LPRSW 算法中都和一个滑动窗口相关,当相应的速度被选定时,所有任务在滑动窗口内均可调度。虽然 LPRSW 算法在最大出错任务数 K 个下能够保证实时任务集合的可调度性,但是每一个窗口内需要预留计算资源给每一个备份任务,而实际系统中,任务全部出错的概率较少,造成空闲时间的浪费。例如两个滑动窗口 I_1 和 I_2,如果出错任务为一个且发生在 I_1 时,I_2 将不会出错,则 I_2 可以将预留的备份任务资源全部分给主任务;如果出错任务发生在 I_2,则 I_1 中全部正确执行,留下的资源可以供 I_2 提前执行任务。更复杂的情况,如图 3a～图 3b 所示,揭示了多任务共享空闲时间情况下的任务可靠性保证。

重新调度表 1 的任务,所有任务都在一个滑动窗口内。3 个任务对应需要的容错空闲时间为 R_1,R_2,R_3,其中 R_3 任务恢复需要的时间最多。如图 3c 所示,当以最高速度执行任务时,不会留下多余的空闲时间,能耗为 10。如果系统中有一个任务出错,例如最长任务 J_2 出错(如图 3a),则可以将任务 J_1 和 J_3 的备份资源用于其他任务调节速度,当所有空闲时间都调节 J_1 任务时,能耗为 8.22,相比 3c 的调度策略节约 17.8%。另一种情况,如图 3c 所示,如果将空闲时间用于 J_1 和 J_3,则能耗为 6,相比 3c 的调度策略节约 40%的能耗。此外,假设最长任务出现错误进行全局预留资源 SR,则实际中空闲时间可以进一步回收。回收的空闲时间可以分给不同的任务共享,从而进一步优化能耗。因此,新算法在保证可靠性前提下还可以进一步降低能耗。

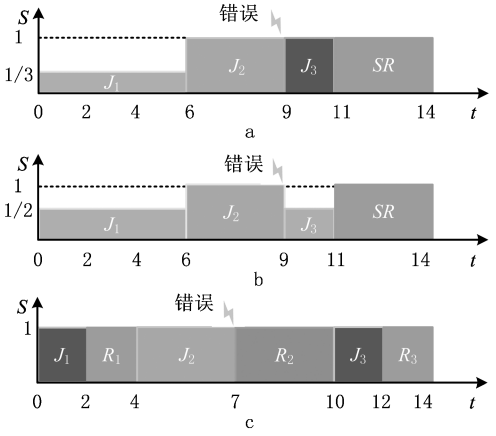

图3　在全局共享空闲时间下将不同的空闲时间分配给多个任务

因此,除了共享预留空闲时间可以在保证系统可靠性前提下降低能耗外,合理分配空闲时间也可以降低能耗。新的算法需要解决以下问题:

(1)如何将回收的空闲时间分配给其他任务,在保证截止期限内实现更低的能耗。

(2)如何在降低能耗和保证系统可靠性之间找到平衡点。

新算法 COSALPRSW 从功耗模型出发,代替

之前每个任务分配一个备份任务，其滑动窗口的变化和备份任务个数通过动态共享空闲时间确定，从而减少任务备份占用的资源，在保证系统可靠性前提下使能耗更低。

为了评估每个任务的能耗和空闲时间的使用效率，定义了以速度 f 运行的任务 J_k 的空闲时间能耗，空闲时间—能耗因子：

$$Slack\_Energy(f)=\frac{E_k^*-E_k(f)}{S_k(f)}。\quad(13)$$

其中 E_k^* 和 $E_k(f)$分别是 $f_{\max}$和 f_k 时任务 J_k 的能量消耗；$S_k(f)$为任务 J_k 在 f_k 运行时所需的空闲时间总量(包括保留用于备份任务预留的的空闲时间)。任务 J_k 以某一速度运行时所节省的能量与所需空闲时间总量的比率为空闲时间—能耗因子。空闲时间—能耗因子的值越高，每单位空闲时间可以节省的能量越多。

在最大频率 $f_{\max}$处没有空闲时间和能耗节约，并且任务的 $Slack_Energy(f_{\max})$被定义为 0。随着 f 的减小，节省能量更多，$Slack\_Energy$ 增加。由于预留用于恢复的空闲时间的影响，预期在某一阈值之后，$Slack_Energy(f)$将增加然后减小。因此，对于每个任务 J_k，应该存在最佳速度 $f_o=(>f_k)$，使得 $Slack_Energy(f)$最大。

$$S_k(f)=C_k/f;\quad(14)$$

$$E_k=P_{s,k}\cdot S_k(f)+h\cdot P_{ind}\cdot S_k(f)+h\cdot C_{ef}\cdot c_k\cdot f^{m-1}。\quad(15)$$

由式(13)～式(15)和关于 f 的微分 $Slack\_Energy(f)$，当 f 满足时，可以得到

$$Slack\_Energy(f)'=P_s+h\cdot P_{ind}+h\cdot C_{ef}-m\cdot h\cdot C_{ef}\cdot f^{m-1}。\quad(16)$$

当 $m=2$ 或 $m=3$ 时，分别产生立方或四次方程，得到最大化 $Slack_Energy(f)$下的最佳速度

$$f_o=\sqrt[\frac{1}{m-1}]{\frac{P_s+h\cdot(P_{ind}+C_{ef})}{m\cdot h\cdot C_{ef}}}。\quad(17)$$

算法 2 COSALPRSW。

1. 按照任务的执行时间降序排列任务集合，$J_1,\cdots,J_n$；
2. 计算可以利用的空闲时间 SL=D−W(I)；
3. $k=\max\{j|\sum_i^j J_i\leqslant SL\}$//计算最大备份任务数量。
4. Set $f_i=f_e(i=1,\cdots,n)$；
5. Set $E=E_i(f_i)$；
6. For n=0 to k do
7. If n>0 then
8. SL=SL−J_n
9. end If
10. End for
11. Assign-frequencies({f},SL)；
12. 根据公式(2)和(7)重新计算新速度下的系统可靠性 R_f 和总能耗 E_{total}；
13. If $E>E_{total}$ and $R_f<R_g$ then
14. Set $E=E_{total}$；
15. Set k=n;//确定全局最优备份任务数量
16. Set $f=f_o(i=1,\cdots,n)$；
17. Else
18. Set $f=f_{max}$
19. End if
20. Return k and $f_1,\cdots,f_n$

算法 3 Assign-frequencies。

1. 计算每个滑动窗口的 Slack_Energy(f)，f_o 和 $SL_k(f_o)$；
2. $SL_{remain}=SL$；
3. 队列中任务按照 Slack_Energy(f)的值降序排列
4. While(Queue=! Φand $SL_{remain}>\min\{J_i|(J_1,J_2\cdots,J_n)\}$ do
5. If($SL_k(f_o)<=S_{remain}$)then
6. 分配 $SL_k(f_o)$给任务 J_k；
7. $SL_{remain}--SL_k(f_o)$；
8. Else
9. 将任务 J_k 加入到没有被选择的队列，等待下一轮调度
10. End If
11. End While
12. Return f_o

对于给定的任务集，Slack_Energy 值越高，任务将越节能。基于这一观察，提出 COSALPRSW 算法，该算法根据空闲时间—能耗因子决定选择任务的顺序和速度。首先优化备份任务数量，共享空闲时间。其次，根据剩余的空闲时间分配给将要调度的任务，找到在系统可靠性前提下能耗最小的速度分配策略。任务集合需要的时间资源总量为 D，可用空闲时间量为 $SL=D-W(I)$，则可以安排备份的数量为 $J_i\leqslant SL$。在保留 SL 个空闲时间量用于恢复出错任务(以重新执行的形式)后，剩余空闲时间($SL-J_i$)可用于缩小所选择任务的处理频率。在算法 Assign-frequencies 中，SL_{remain}表示剩余空闲时间量。队列用于按照其最大空闲时间使用效率因子 $Slack\_Energy(f)$(行 3)的递减顺序对任务进行排序，空闲时间以该顺序分配给任务，使其能够以最佳速度(行 5～7)运行。算法的复杂度只有 $O(n\log n)$，主要是对任务的 $Slack_Energy(f)$值进行排序。

3 实验结果与分析

实验采用沈阳蓝天数控 GJ400，采用支持动态电压调节的 64 位龙芯 3B-1500 处理器，该处理器允

许调整的电压范围为 1.0 V～1.3 V,对应可变的处理器频率为 1.0 GHz～1.5 GHz[20]。基于低能耗龙芯处理器的开放式数控系统 GJ400 如图 4 所示,系统分为二次开发层、CNC 数控系统层和运动控制层。其中低功耗嵌入式平台的运动控制器负责进给驱动控制、主轴驱动控制和开关量控制,实现任务严格的实时性要求。低能耗可靠调度算法部署在 CNC 数控系统层。在实验中,修改操作系统内核,以便在创建线程之前,使用系统调用将获得的任务相关信息发送到内核和调度程序。当任务调用 Pthread_create()创建线程时,它会依次调用一系列系统调用函数,包括 Sys_clone(),Do_fork()和 Copy_process()函数。修改后,Copy_process()和 Sched_fork()函数将收集到的任务执行信息反馈到本文的能耗模型和可靠性模型,用于分析所提算法的能耗和可靠性。

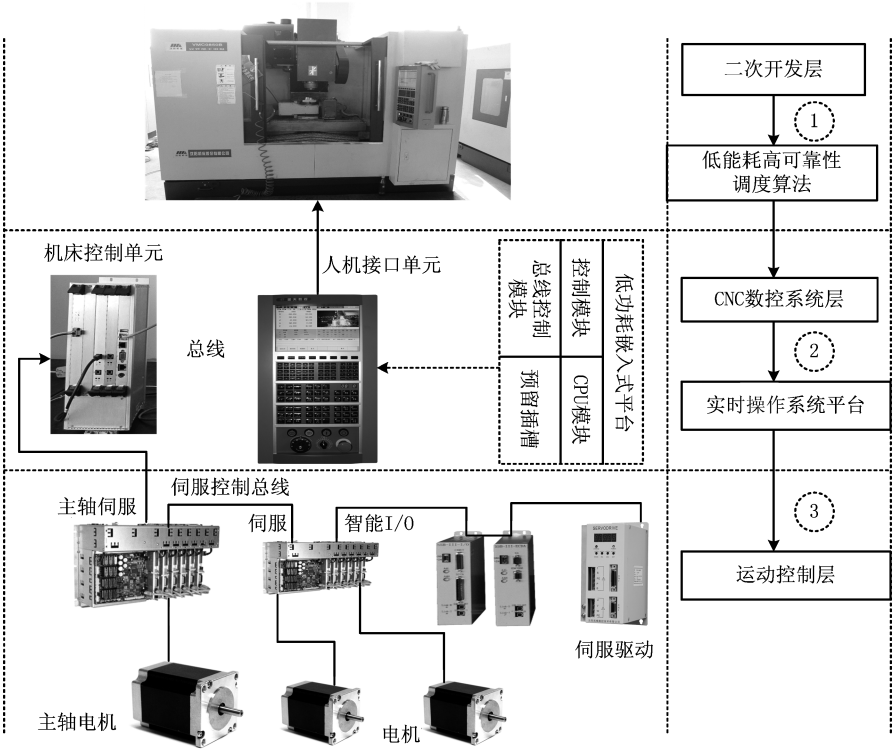

图4　基于低能耗龙芯处理器的开放式数控系统

本章实验比较所提出的 4 个算法——NODVS,LPRSW-A,LPRSW-H 和 COSALPRSW 算法。其中 NODVS 算法没有采用能耗管理,执行任务按照最高速度执行,当系统空闲时将系统置于省电的睡眠状态,NODVS 算法为其他算法能耗归一化的对比算法。数控系统中执行一次插补任务的周期为 6 ms,每次插补计算的结果通过命令寄存器分若干次传送到伺服控制单元,以控制机床各个坐标轴的位移,位置控制任务的周期为 3 ms;执行一次运动控制处理任务为 15 ms[5]。系统任务出错数 K 根据任务集合和任务到达错误率确定。根据文献[17]确定关键实时系统中的任务达到错误率在 10^{-10}/h～10^{-5}/h 之间。如果实时系统在复杂的环境中,任务到达出错率则更高,在 10^{-2}/h～10^{2}/h 之间。NODVS 算法的原始故障概率(即 $1-R_0=1-\prod R_i(f_{\max})$)由 PoF(possibility of fault)表示。用能耗和可靠性指标来测试所提其他算法的性能,分别为在线性速度和离散速度集合测试算法。每次实验时间为 10 000 个时间片,每个任务集运行 10 次,将这 10 次运行结果的平均值作为最终的实验结果。

3.1　空闲时间对系统可靠性的影响

系统可靠性定义为 1－PoF,通过使用故障率模型和执行时间信息计算系统的可靠性。图 5 所示为故障率模型中的指数 $d=2$ 时对系统可靠性的影响,结果和 NODVS 的结果进行归一,观察到即使故障率只有适度增加,NODVS 仍导致系统可靠性大幅度下降。此结果符合预期,因为 NODVS 使用最高速度执行任务,而不考虑对可靠性的影响。然而,LPRSW-A,LPRSW-H 和 COSALPRSW 均为可靠性感知方案,是通过在减少频率之前保留空闲时间

来恢复出错任务，从而确保系统在故障率急剧增加时，也能保证系统的可靠性。这与理论结果一致。随着空闲时间的增加，用于容错的时间增加，可以使系统管理更多的任务并获得更低的故障概率。随着空闲时间的增加，LPRSW-A 的出错率高于 LPRSW-H，这是因为 LPRSW-A 算法采用调整后的速度执行出错任务，执行时间相对 LPRSW-H 算法采用最大速度恢复出错任务要长，从而增加了出错机率。在系统空闲时间 0.6 ms 以后，随着空闲时间的增加，LPRSW-A 和 LPRSW-H 的出错率表现更明显，而 COSALPRSW 算法出错率更低，这是因为 COSALPRSW 采用全局共享空闲时间。

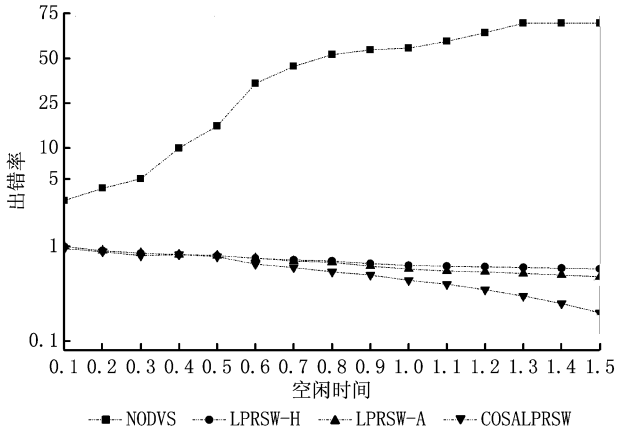

图5 空闲时间对系统可靠性的影响(d=2)

3.2 空闲时间对能耗的影响

可用空闲时间对能量消耗的影响，如图 6 所示频率无关功率设置为 $P_{ind}=0.05$，可用空闲时间由 $SL=D-W(I)$ 表示。在图 6 中，目标可靠性被设置为原始系统可靠性 $R_g=1-PoF$。开始 LPRSW-A 和 LPRSW-H 的能耗非常接近，高于 COSALPRSW 算法。在空闲时间大于 0.7 以后，LPRSW-A 算法相比 LPRSW-H 算法节约的能耗更多，这是因为空闲时间增多，当任务出现错误时，LPRSW-A 算法使用 DVS 调节后的速度执行。COSALPRSW 算法的能耗更低，是因为 COSALPRSW 算法进行恢复出错任务分配时优于 LPRSW-A 和 LPRSW-H。空闲时间越多，任务进行动态电压调节 DVFS 和恢复出错任务的机会越多；并且 COSALPRSW 算法通过全局回收空闲时间再分配能够实现降低能耗。然而，LPRSW-A 和 LPRSW-H 需要为不同的任务分配单独的空闲时间，因而算法节能效果低于 COSALPRSW 算法。平均 COSALPRSW 算法比 LPRSW-A 和 LPRSW-H 算法节约 31.62% 和 44.4%的能耗。

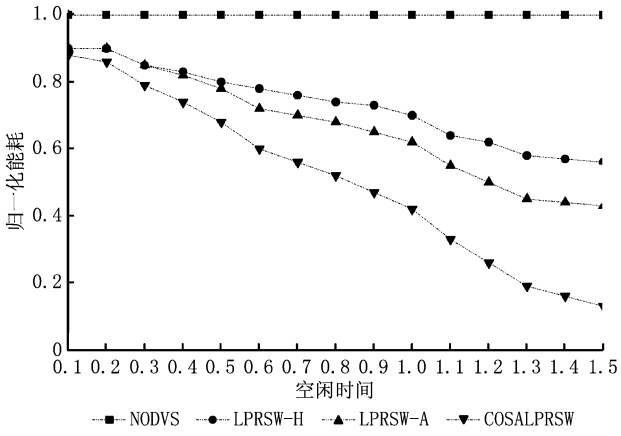

图6 空闲时间对系统能耗的影响

3.3 系统可靠性对能耗的影响

图 7 所示为可靠性目标 R_g 对系统能量消耗的影响，空闲时间 $SL=1.1\cdot C$，$P_{ind}=0.05$。通过调节任务的出错率，测试算法能耗。随着出错率 PoF 从 10^{-4} 改变为 10^{-10}，所有算法的能耗随着可靠性目标的增加(即降低 PoF)变得更高。这是由于需要预留更多资源来恢复出错任务，以满足系统的可靠性，从而减少了用于动态调节的可用空闲时间。随着可靠性的增加，需要更多的资源用于恢复任务，并且保证不超过截止期限，因此能耗急剧上升，当 $PoF=10^{-11}$ 时，系统可靠性最大，所有算法都以最大频率 f_{max} 运行。

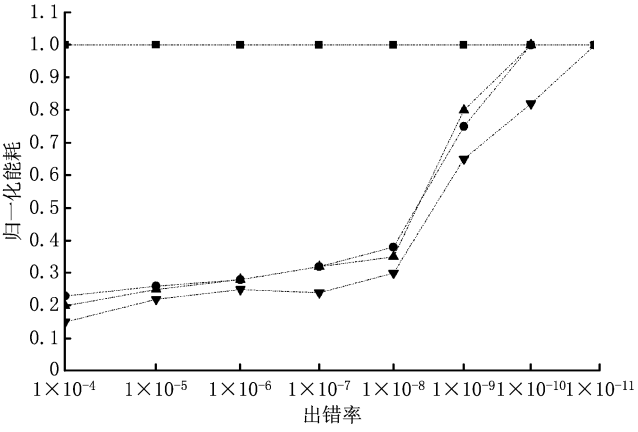

图7 *PoF*对系统可靠性的影响

4 结束语

本文研究开放式数控系统的低能耗高可靠性调度算法。对 LPEDF 算法进行拓展，使其成为能够容错的低能耗调度算法，考虑可能出现的任务优先级反转问题，通过融合相邻滑动窗口消除优先级反

转。提出 LPRSW 算法,能够在保证系统可靠性前提下降低系统功耗,同时将 LPRSW 算法拓展为以处理器最高速度执行出错任务的 LPRSW-H 算法和采用 DVS 技术调节后的执行出错任务的 LPRSW-A 算法,并分析二者在不同情况下的优势和不足。在此基础上,提出一种性能更好的低能耗和可靠性优化调度算法 COSALPRSW,该算法对任务容错采用共享全局空闲时间代替预先给每任务都分配一个备份任务的方法,因此可以回收更多的空闲时间,大幅度降低系统能耗,同时提出空闲时间—能耗因子用于决定任务动态调整后的最优速度,实现在保证系统可靠性前提下进一步降低系统能耗。通过实验对比,COSALPRSW 算法比 LPRSW-A 和 LPRSW-H 的效果更好,能平均节约 31.62% 和 44.4%的能耗。下一步将基于开放式数控系统对能耗和可靠性的需求,从实时任务模型构建、多核处理器能耗分析、实时系统稳定性以及基于机器学习的调度算法等方面研究开放式数控系统,并将所提算法进行实际系统验证,从而实现开放式数控系统高可靠低能耗的目的。

参考文献:

[1] WANG Ping, GE Shilun, WANG Nianxin, et al. MIS 4.0 research for Industrie 4.0[J]. Computer Integrated Manufacturing Systems,2016,22(7):1812-1820(in Chinese).[王平,葛世伦,王念新,等.面向工业 4.0 的 MIS 4.0 研究.计算机集成制造系统,2016,22(7),1812-1820.]

[2] ZHOU Gang, WU Yijie, PAN Xiaohong. Software modules real time scheduling framework in CNC system[J]. Journal of Mechanical Engineering,2009,45(1):162-166(in Chinese).[周　刚,邬义杰,潘晓弘.数控系统软件模块实时调度方法[J].机械工程学报,2009,45(1):162-166.]

[3] LIU Ning, WANG Gao. Development of open architecture special NC system based on embedded multi-CPUs structure [J]. Computer Integrated Manufacturing Systems, 2012, 18(8):1854-1860(in Chinese).[柳　宁,王　高.嵌入式多 CPU 架构的开放式专用数控系统的研发[J].计算机集成制造系统,2012,18(8):1854-1860.]

[4] LIU Xian, GUO Ruifeng. Preallocation-based hybrid task scheduling algorithm in numerical control system[J]. Computer Integrated Manufacturing Systems,2015,21(6):1529-1535(in Chinese).[刘　娴,郭锐锋.数控系统中基于预分配的混合任务调度算法[J].计算机集成制造系统,2015,21(6):1529-1535.]

[5] WEISER M, WELCH B, DEMERS A, et al. Scheduling for reduced CPU energy[C]//Proceedings of the 1st Usenix Conference on Operating Systems Design and Implementation. Berkeley, Cal.,USA:USENIX Association,1994:13-23.

[6] PENNYCOOK S J, HUGHES C J, SMELYANSKIY M, et al. Exploring SIMD for molecular dynamics, using Intel® Xeon® processors and Intel® Xeon Phi Coprocessors[C]//Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing. Washington,D.C.,USA:IEEE Computer Society,2013:1085-1097.

[7] JARUS M, VARRETTE S, OLEKSIAK A, et al. Performance evaluation and energy efficiency of high-density HPC platforms based on Intel, AMD and ARM processors[C]//Proceedings of the European Conference on Energy Efficiency in Large Scale Distributed Systems. Berlin, Germany:Springer-Verlag,2013:182-200.

[8] WANG Gang, LI Wenming, HEI Xiali. Energy-aware real-time scheduling on heterogeneous multi-processor[C]//Proceedings of the Information Sciences and Systems. Washington, D.C.,USA:IEEE,2015:1-7.

[9] ZHAO Baoxian, AYDIN Hakan. Minimizing expected energy consumption through optimal integration of DVS and DPM [C]//Proceedings of the IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers. Washington, D.C.,USA:IEEE,2009:449-456.

[10] LIU Xian, GUO Ruifeng, DENG Changyi. Fault-tolerant real-time scheduling algorithm with pre-allocation in primary/alternate model[J]. Journal of Computer Research and Development,2015,52(3):760-768(in Chinese).[刘　娴,郭锐锋,邓昌义.主/副版本模型中预分配容错实时调度算法[J].计算机研究与发展,2015,52(3):760-768.]

[11] KIM N S, KGIL T, BOWMAN K, et al. Total power-optimal pipelining and parallel processing under process variations in nanometer technology[C]//Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. Washington, D.C.,USA:IEEE,2005:535-540.

[12] SHAO Zili, WANG Meng, CHEN Ying, et al. Real-time dynamic voltage loop scheduling for multi-core embedded systems[J]. Circuits & Systems II Express Briefs IEEE Transactions on,2007,54(5):445-449.

[13] AVIZIENS A. Fault-tolerant systems[J]. IEEE Transactions on Computers,2006,C-25(12):1304-1312.

[14] MILLER T, SURAPANENI N, TEODORESCU R. Flexible error protection for energy efficient reliable architectures [C]//Proceedings of the International Symposium on Computer Architecture and High Performance Computing. Washington, D.C.,USA:IEEE,2010:1-8.

[15] MELHEM R, MOSS D, ELNOZAHY E. The Interplay of power management and fault recovery in real-time systems [J]. IEEE Transactions on Computers, 2004, 53(2): 217-231.

[16] ZHU Dakai, AYDIN Hakan, CHEN Jianjia. Optimistic reliability aware energy management for real-time tasks with probabilistic execution times[C]//Proceedings of the IEEE in

Real-Time Systems Symposium. Washington, D. C., USA: IEEE, 2008: 313-322.

[17] ZHAO Baoxian, AYDIN Hakan, ZHU Dakai. Generalized reliability-oriented energy management for real-time embedded applications[C]//Proceedings of the 48th Design Automation Conference. New York, N. Y., USA: ACM, 2011: 381-386.

[18] HAN Qiushi, NIU Linwei, QUAN Gang, et al. Energy efficient fault-tolerant earliest deadline first scheduling for hard real-time systems[J]. Real-Time Systems, 2014, 50(5): 592-619.

[19] JOHNSON B W. Design & analysis of fault tolerant digital systems[M]. Bosten, Mass., USA: Addison-Wesley Publishing Company, 1988.

[20] YAO F, DEMERS A, SHENKER S. A scheduling model for reduced CPU energy[C]//Proceedings of the IEEE 36th Annual Foundations of Computer Science. Washington, D. C., USA: IEEE, 1995: 374-382.

[21] AYDIN H. Exact Fault-sensitive feasibility analysis of real-time tasks[J]. IEEE Transactions on Computers, 2007, 56(10): 1372-1386.

作者简介：

邓昌义(1989－)，男，河南周口人，博士研究生，研究方向：信息物理系统，E-mail：dengchangyi@sict.ac.cn；

郭锐锋(1968－)，男，辽宁沈阳人，研究员，博士，博士生导师，研究方向：实时系统、数控系统等；

吴昊天(1992－)，男，辽宁沈阳人，博士研究生，研究方向：实时系统、低功耗调度算法等；

彭阿珍(1992－)，女，河南周口人，博士研究生，研究方向：实时系统、数字控制系统等；

杜少华(1982－)，男，辽宁沈阳人，副研究员，博士，研究方向：实时系统、数控系统；

盖荣丽(1980－)，女，辽宁大连人，副教授，博士，研究方向：嵌入式系统、数控系统、信息安全等。

REFERENCES

[1] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, 2017.

[2] Z. G. W. Y. P. Xiaohong, "Software modules real time scheduling framework in cnc system [j]," *Chinese Journal of Mechanical Engineering*, vol. 1, 2009.

[3] L. Ning and W. Gao, "Development of open architecture special nc system based on embedded multi-cpus structure," *Computer Integrated Manufacturing Systems*, no. 8, p. 26, 2012.

[4] F. Zhao, Y. Hong, D. Yu, Y. Yang, Q. Zhang, and H. Yi, "A hybrid algorithm based on particle swarm optimization and simulated annealing to holon task allocation for holonic manufacturing system," *The International Journal of Advanced Manufacturing Technology*, vol. 32, no. 9-10, pp. 1021–1032, 2007.

[5] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Mobile Computing*. Springer, 1994, pp. 449–471.

[6] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy, and S. A. Jarvis, "Exploring simd for molecular dynamics, using intel® xeon® processors and intel® xeon phi coprocessors," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 1085–1097.

[7] M. Jarus, S. Varrette, A. Oleksiak, and P. Bouvry, "Performance evaluation and energy efficiency of high-density hpc platforms based on intel, amd and arm processors," in *European Conference on Energy Efficiency in Large Scale Distributed Systems*. Springer, 2013, pp. 182–200.

[8] G. Wang, W. Li, and X. Hei, "Energy-aware real-time scheduling on heterogeneous multi-processor," in *2015 49th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2015, pp. 1–7.

[9] B. Zhao and H. Aydin, "Minimizing expected energy consumption through optimal integration of dvs and dpm," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, 2009, pp. 449–456.

[10] L. Xian, G. Ruifeng, and D. Changyi, "Fault-tolerant real-time scheduling algorithm with pre-allocation in primary/alternate model," *Journal of Computer Research and Development*, vol. 52, no. 3, p. 760, 2015.

[11] N. S. Kim, T. Kgil, K. Bowman, V. De, and T. Mudge, "Total power-optimal pipelining and parallel processing under process variations in nanometer technology," in *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.* IEEE, 2005, pp. 535–540.

[12] Z. Shao, M. Wang, Y. Chen, C. Xue, M. Qiu, L. T. Yang, and E. H.-M. Sha, "Real-time dynamic voltage loop scheduling for multi-core embedded systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 5, pp. 445–449, 2007.

[13] A. Aviziens, "Fault-tolerant systems," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1304–1312, 1976.

[14] T. Miller, N. Surapaneni, and R. Teodorescu, "Flexible error protection for energy efficient reliable architectures," in *2010 22nd International Symposium on Computer Architecture and High Performance Computing*, Oct 2010, pp. 1–8.

[15] R. Melhem, D. Mossé, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Transactions on Computers*, vol. 53, no. 2, pp. 217–231, 2004.

[16] D. Zhu, H. Aydin, and J.-J. Chen, "Optimistic reliability aware energy management for real-time tasks with probabilistic execution times," in *2008 Real-Time Systems Symposium*. IEEE, 2008, pp. 313–322.

[17] B. Zhao, H. Aydin, and D. Zhu, "Generalized reliability-oriented energy management for real-time embedded applications," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2011, pp. 381–386.

[18] Q. Han, L. Niu, G. Quan, S. Ren, and S. Ren, "Energy efficient fault-tolerant earliest deadline first scheduling for hard real-time systems," *Real-Time Systems*, vol. 50, no. 5-6, pp. 592–619, 2014.

[19] B. W. Johnson, *Design & analysis of fault tolerant digital systems*. Addison-Wesley Longman Publishing Co., Inc., 1988.

[20] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proceedings of IEEE 36th annual foundations of computer science*. IEEE, 1995, pp. 374–382.

[21] H. Aydin, "Exact fault-sensitive feasibility analysis of real-time tasks," *IEEE Transactions on Computers*, vol. 56, no. 10, pp. 1372–1386, 2007.