·体系结构与软件技术· 文章编号：1000-3428(2019)06-0108-07 文献标志码：A 中图分类号：TP301.6

适用于偶发实时系统的过载控制策略

崔丽群，郭相卓，郭 军，黄迪文

（辽宁工程技术大学 软件学院，辽宁 葫芦岛 125100）

摘 要： 针对偶发实时系统中过载处理资源消耗大的问题，提出一种基于最小可推迟时间和非精确计算的过载控制策略。结合最早截止期优先调度算法，利用最大挪用时间与最小可推迟时间动态判断系统负载状态，根据系统负载状态舍弃部分非重要任务，解决系统过载问题。实验结果表明，该策略可缩短过载处理时间，提高系统资源利用率。

关键词： 偶发实时系统；最小可推迟时间；过载处理；非精确计算；PID 控制

中文引用格式： 崔丽群，郭相卓，郭军，等. 适用于偶发实时系统的过载控制策略[J]. 计算机工程，2019，45(6)：108-114.

英文引用格式： CUI Liqun，GUO Xiangzhuo，GUO Jun，et al. Overload control strategy for sporadic real-time system[J]. Computer Engineering，2019，45(6)：108-114.

Overload Control Strategy for Sporadic Real-Time System

CUI Liqun，GUO Xiangzhuo，GUO Jun，HUANG Diwen

（School of Software，Liaoning Technical University，Huludao，Liaoning 125100，China）

【Abstract】 Aiming at the problem that overload processing consumes high system resources in sporadic real-time systems，an overload control strategy based on Minimum Delay Time and Imprecise Computation(MDTIC) is proposed. Combined with the Earliest Deadline First(EDF) scheduling algorithm，the system load state is dynamically determined by using the maximum stealing time and minimum delay time，then some non-critical tasks are discarded according to the system load state to solve the system overload problem. Experimental results show that the proposed strategy can reduce the overload processing time and improve the utilization rate of the system resource.

【Key words】 sporadic real-time system；minimum delay time；overload processing；imprecise computation；PID control

DOI： 10.19678/j.issn.1000-3428.0050894

0 概述

随着实时设备逐渐应用到各场景，实时系统[1-2]的种类和应用领域也得到不断丰富和扩展。偶发实时系统[3]是在周期任务中穿插执行偶发任务的实时系统，通常应用于工业控制中。偶发任务的释放时间是随机的，系统无法在任务释放之前获取任务相关信息。在偶发实时系统中，偶发任务的抢占容易导致系统过载，当系统过载时，很多周期任务无法顺利完成，导致任务成功率降低和系统资源浪费。

现有研究主要通过非精确计算和弹性计算[4-5]策略解决实时系统过载问题。文献[6]基于价值率优先原则，在系统出现过载时保证价值率高的任务优先执行，舍弃价值率低的任务，算法简单且系统价值积累高，但无法保证系统资源利用率。文献[7]提出贪心策略模型，将最优调度集的选择过程看作经典的 0-1 背包问题，根据贪心算法原则，在寻找最优调度集时，总是做出当前最优的选择，在很多情况下可以达到预期目的，但由于贪心算法的局部最优性，因此无法最大化利用系统资源。文献[8]提出改进的动态贪心策略，在贪心策略中引入价值因素，可以较好满足系统的过载处理要求，进一步提升系统性能，但系统资源消耗过高。文献[9]提出 HP-MOS 过载控制策略，通过拒绝不可能完成的任务达到消除任务调度中级联抢占问题，提高系统资源利用率。文献[10]提出基于 Lebesgue 的动态反馈调度策略，通过调整任务周期降低系统负载。

现有研究在一定程度上解决了实时系统过载问题，但应用于偶发实时系统时，无法适应周期任务与非周期任务混合调度或过载处理消耗系统资源过多的情况。为有效地解决偶发实时系统过载问题，本文提出一种基于最小可推迟时间和非精确计算

基金项目： 国家自然科学基金(61172144)；辽宁省教育厅科研项目(L2012113)。

作者简介： 崔丽群(1969—)，女，副教授，主研方向为嵌入式系统、智能视觉信息处理；郭相卓，硕士研究生；郭 军、黄迪文，本科生。

收稿日期： 2018-03-22 **修回日期：** 2018-04-23 **E-mail：** 843181688@qq.com

(Minimum Delay Time and Imprecise Computation, MDTIC)的过载控制策略。MDTIC 策略根据最大挪用时间和最小可推迟时间来精确判断系统过载情况,利用候选解表拒绝执行少量任务,保证尽可能多的任务顺利完成,从而实现过载时偶发实时系统的性能优化。

1　MDTIC 策略模型

1.1　符号说明

MDTIC 策略模型中所使用的符号及变量具体如下:T_{ij}表示作业编号,R_{ij}表示作业释放时间,P_{ij}表示作业周期,C_{ij}表示作业的最坏执行时间,E_{ij}表示作业的实际执行时间,D_{ij}表示作业的绝对截止时间,T_k表示偶发任务的编号,R_k 表示偶发任务的释放时间,C_k 表示偶发任务的最坏执行时间,D_k 表示偶发任务的绝对截止时间,t 表示当前系统的运行时间,S_{ij}表示作业的运行状态,V_{ij}表示作业完成产生的价值,H表示超周期的大小。

1.2　任务模型

假设 $TR=\{\tau_1,\tau_2,\cdots,\tau_n\}$为实时系统中的任务集合,$\tau_i=\{T_{i1},T_{i2},\cdots,T_{in}\}$为任务作业集合。周期任务释放时间间隔和相对截止时间等于其周期,即 $R_{i(j+1)}=R_{ij}+P_{ij}$、$D_{i(j+1)}=D_{ij}+P_{ij}$。偶发任务的释放时间是随机的,没有周期,其相对截止时间就是执行时间,即 $D_k=R_k+C_k$。简单的作业模型为$T_{ij}=\{R_{ij},C_{ij},D_{ij}\}$。针对该模型,本文假定以下条件:1)系统中的任务相互独立,无依赖关系;2)偶发任务被释放前,各项参数都未知;3)周期任务为软实时任务,可以被抢占,偶发任务为硬实时任务,不可被抢占;4)作业的优先级具有唯一性。

2　MDTIC 策略分析

在偶发实时系统中,由于偶发任务的不确定性、无周期性,增加了过载判定的难度,为了简单准确地判定系统过载情况,MDTIC 分析系统过载与偶发任务之间的关系,并给出如下定理、定义。

定理 1　实时系统过载的必要条件是任务系统在任意时间段产生的时间需求量超过该时间段的长度,形式化描述如下:

$$\forall 0\leqslant t_1<t_2, d(t_1,t_2)>t_2-t_1 \tag{1}$$

其中,$d(t_1,t_2)$为(t_1,t_2)时间段内产生的时间需求量。由于周期任务在每个超周期内保持其执行状态不变的特点,$d(t_1,t_2)$产生增量的主要原因为偶发任务的释放,因此只需在偶发任务释放时判定系统状态。

定义 1　最大挪用时间(Maximum appropriation time,Mat)是在一个超周期内,保证后续的所有周期任务可以正常调度的前提下,在任意时刻都可以被推迟挪用的最大时间段。

Mat 是文献[11]中提出的一种实时系统可调度性判断方法,根据 *Mat* 的定义和性质可以得出以下推论:

推论 1　当偶发任务被释放时,若 $C_k\leqslant Mat$,则偶发实时系统不会过载,$Mat=Mat-C_k$。

根据最早截止期优先(Earliest Deadline First, EDF)调度算法的特性,如果被抢占的时间段中包含空闲时间(等待新任务释放的时间段),则当前被抢占的时间段内可以被挪用的时间会增加,因此得出推论 2。

推论 2　当偶发任务被释放时,若 $C_k>Mat$,则偶发实时系统不一定过载。

由推论 2 可知,最大挪用时间并不能判定所有的系统过载,为提高过载判定的准确性,进一步研究影响系统过载的条件,给出定义 2。

定义 2　可推迟时间(Delay time,Dt)是在时间段(t,D_{ij})内,保证时间段内所有作业在截止期之前完成的条件下,可以被占用的最大时间段,其中 D_{ij}为 t 时刻之后已释放但未完成释放或未释放的作业的绝对截止时期。

为进行更好的说明,给出 EDF 调度执行示例,如图 1 所示。

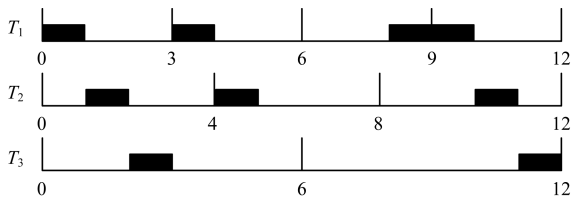

图 1　EDF 调度执行示例

图 1 为一个超周期中,在 $t=5$ 时推迟 3 个时间单位后,EDF 算法的调度执行,包含 3 个周期任务$\tau_1=\{T_{11},T_{12},T_{13},T_{14}\}$、$\tau_2=\{T_{21},T_{22},T_{23}\}$、$\tau_3=\{T_{31},T_{32}\}$,其中,$T_{11}=\{0,1,3\}$、$T_{21}=\{0,1,4\}$、$T_{31}=\{0,1,6\}$。根据定义,在 $t=5$ 时,(5,9)时间段内的 *Dt* 为 3,(5,12)时间段内的 *Dt* 为 4。可推迟时间可以反映不同时间段内对系统过载的容错能力,即 *Dt* 越高,任务越不容易错过截止期。根据可推迟时间的特点进一步提出最小可推迟时间的概念。

定义 3　最小可推迟时间(Minimum delay time, Mdt)是一个超周期内,从时刻 t 开始,任意时间段内 *Dt* 的最小值,计算公式为:

$$\forall(t,D_{ij}), Mdt=\min\{Dt_{ij}\} \tag{2}$$

根据实时系统调度的特点,在某一个超周期的某一段时间内,若某一次作业推迟一定时间,则后续任务一定会推迟相同的时间,若推迟的时间大于该时间段内的所有 *Dt*,那么一定会有任务无法在截止期之前完成,因此 *Mdt* 可以用来反映一个超周期内 t 时刻可占用时间段,得出推论 3。

推论 3　在某时刻 t,偶发任务被释放,$C_k>Mdt$

是偶发实时系统过载的充要条件。

根据定义，*Mat* 属于静态变量，在系统设计阶段，系统资源消耗为 0；*Mdt* 属于动态变量，在系统调度过程中，需要消耗一定的系统资源，MDTIC 策略将 *Mat* 与 *Mdt* 相结合作为判定偶发实时系统过载的条件，由此得到推论 4。

推论 4 若 $C_k \leqslant Mat$，则偶发实时系统不会过载；若 $Mat < C_k \leqslant Mdt$，则偶发实时系统不会过载；若 $C_k > Mdt$，则系统过载。

在分析系统过载的判定条件后，为了有效处理系统过载，MDTIC 策略基于非精确计算的思想，分析被拒绝执行作业与偶发任务之间的关系，给出以下定义：

定义 4 非精确计算是通过舍弃某些不重要的任务来降低系统负载的方法。由于其计算简单，因此是实时系统过载处理中常用的方法。

定义 5 被拒绝任务集（*TB*）是指经精确计算而被系统拒绝执行的任务集合。

为保证系统的资源利用率最高，被拒绝任务通常具有以下特点：

1）任务在待调度任务集中进行选择。

2）任务为对系统影响较小的软实时任务。

3）任务价值率尽可能低。

4）相同的周期任务不能连续成为被拒绝任务。

定义 6 候选解表（*TC*）是所有 *TB* 可能出现的组合的集合，形式化表示为：

$$TC = \{TB_1, TB_2, \cdots, TB_n\} \tag{3}$$

根据推论 4，可以得出降低偶发实时系统负载的 2 个必要条件。

1）$L(TB) + Mdt \geqslant C_k$，$L(TB)$ 为调度执行 *TB* 所需要的时间消耗。

2）待调度集中剩余的偶发任务和作业满足可调度性判断。

根据以上定义与推论，推出定理 2。

定理 2 $TB = \{TC \mid L(TB) \geqslant C_k - Mdt\}$ 是降低系统负载的必要条件。

根据定义 1，系统负载 $U = \dfrac{d(t_1, t_2)}{t_2 - t_1}$，在拒绝任务后，系统负载下降为 $U = \dfrac{d(t_1, t_2) - L(TB)}{t_2 - t_1}$，由此得出推论 5。

推论 5 在满足定理 2 的条件下，$L(TB)$ 尽可能小是保证系统资源利用率最高的必要条件。

根据推论 5，MDTIC 策略解决偶发实时系统过载的主要方法是在满足定理 2 的条件下，在候选解表 *TC* 中查找符合条件的 *TB*，从满足条件的 *TB* 中筛选出总价值最小的 *TB*。在作业调度中，如果该作业属于 *TB* 中的周期任务，则拒绝执行，达到降低系统负载的目的。

3 MIEDF 调度算法的设计与实现

由于 EDF 调度算法是最优的单处理器可抢占式调度算法[12-14]，可以调度周期任务与非周期任务，因此将 MDTIC 策略与 EDF 调度算法相结合，设计并实现基于 MDTIC 策略的最早截止时期优先（MIEDF）调度算法。MIEDF 调度算法可分为过载判定、过载处理、作业调度、PID 控制 4 个部分，如图 2所示。

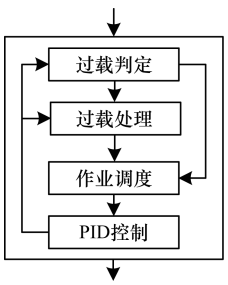

图 2 MIEDF 调度流程

3.1 过载判定

过载判定的目的是在偶发任务释放时，依据推论 4 判断系统的负载情况，下文对 *Mat*、*Dt*、*Mdt* 的计算方法进行说明。

在文献[15]中指出，抢占式 EDF 调度算法尽量提前周期作业的运行，调度时的空闲时间总是最靠后且最短。因此，一个任务集的最大挪用时间等于 EDF 调度在一个超周期内的最后一个空闲时间段。空闲时间（l_i）根据其总是在任务释放时间前这一特点求得。

Mat 的计算公式为：

$$l_1 = 0 \tag{4}$$

$$l_j = R_j - \sum_{i=0}^{n} \lceil R_i / P_i \rceil \times C_i - \sum_{i=0}^{j-1} l_i \tag{5}$$

$$Mat = l_n \tag{6}$$

根据定义 3，得到时间段 (t, D_{ij}) 内 Dt_{ij} 的计算公式为：

$$Dt_{ij} = D_{ij} - t - \max\left(0, \left(\sum_{m=1}^{n} (\lceil D_{ij} / P_m \rceil - n) \times C_i\right)\right) \tag{7}$$

其中，n 表示在时刻 t 前已经完成的任务个数。

根据式（2）可以求得 *Mdt*，但若求得所有的 *Dt*，则计算量过大，为提高 *Mdt* 的求解效率，需分析 EDF 调度算法的特点，得出定理 3。

定理 3 在采用 EDF 调度、周期任务集且系统负载 $U \leqslant 1$ 的前提下，$Dt_{i(j+1)} \geqslant Dt_{ij}$。

证明 根据反证法，若 $Dt_{i(j+1)} < Dt_{ij}$，在抢占一段时间后，则可能会出现 T_{ij} 在截止期前完成而

$T_{i(j+1)}$无法在截止期前完成的情况,这违背了 EDF 调度算法尽可能早执行任务的特性,因此证明成立。

根据定理 3,只需每个周期任务计算一次(t,D_i)内的 Dt 便可求出 Mdt,可大幅提高 Mdt 的求解效率。D_i 的计算公式为:

$$D_i = (\lfloor t/P_i \rfloor + S_i) \times P_i \tag{8}$$

其中,S_i 为周期任务 τ_i 在 t 时刻待调度作业的状态,若作业已被释放,则 $S_i = 0$;若作业未被释放,则 $S_i = 1$。

将 D_i 带入式(7)计算 Dt_i,得出 Mdt 的计算公式为:

$$Mdt = \min\{Dt_1, Dt_2, \cdots, Dt_n\} \tag{9}$$

Mat 可以在系统设计阶段根据公式计算得到,在过载判定过程中:若 $C_{ij} \leqslant Mat$,则跳转到作业调度;若$C_{ij} > Mat$,则根据公式计算 Mdt;若 $C_{ij} \leqslant Mdt$,则跳转到作业调度;若 $C_{ij} > Mdt$,则跳转到过载处理。

3.2　过载处理

过载处理的目的是,当系统过载时,根据定理 2 与推论 5,在 TC 中选择合适的 TB,在待调度集中根据 TB 拒绝执行部分任务,达到降低系统负载的目的。下文说明建立 TC 与选择 TB 的方法。

根据定义 6 和被拒绝任务的特点建立 TC,TC 中应包含所有 TB 可能出现的情况,TC 应便于查询,TB 中的作业为周期作业,TB 中的作业为不同周期任务的作业。根据以上特点得到定义 7。

定义 7　TC 是所有周期任务可能出现的组合及对应组合中任务时间需求 $L(TB)$、任务价值总量 $V(TB)$,按照 $L(TB)$ 与 $V(TB)$ 由小到大的顺序进行排列的表。图 1 中任务集的 TC 如表 1 所示。

表 1　任务集的候选解表

$L(TB)$	V_1	V_2	V_3
1	τ_1	τ_2	τ_3
2	τ_1、τ_2	τ_2、τ_3	τ_1、τ_3
3	τ_1、τ_2、τ_3		

为解决系统过载的同时保证系统资源利用率最高,根据定理 2 与推论 5,得出 EDF 调度中 TB 的计算公式为:

$$L(TB) \geqslant C_{ij} - Mdt; TB_a = f(L(TB)) \tag{10}$$

$$TB_b = \{TB_a \mid TB_a \supseteq \{\tau_i \mid D_i = g(\tau_i) \leqslant C_k\}\} \tag{11}$$

$$TB = h(\min\ V(TB_b)) \tag{12}$$

其中,TB_a、TB_b 为中间变量,函数 $f()$ 表示 $L(TB)$ 与 TB 之间的映射关系,即 $f(L(TB)) = TB$。函数 $g()$ 表示 τ_i 与 D_i之间的映射关系,即 $g(\tau_i) = D_i$。函数 $h()$ 表示 $V(TB)$ 与 TB 之间的映射关系,即 $h(V(TB)) = TB$。

在过载处理中,将 TB 中周期任务的下一次作业标记为拒绝执行作业,在作业调度过程中拒绝执行作业不会被调度和执行,从而保证系统稳定运行。

3.3　作业调度

MIEDF 调度算法中选择抢占式 EDF 调度算法[16]作为主要的作业调度算法。EDF 调度算法在每个时间点处理器都会从那些已经释放但未完成的若干作业中挑选出绝对截止期最小的作业优先执行。EDF 调度算法倾向于使那些最先可能错失截止期的作业抢占处理器资源,从而使尽可能多的作业能够在其相应的绝对截止期前完成。由于在作业调度过程中,并没有引入新的算法,因此本文不再讨论。

3.4　PID 控制

PID 控制策略[17]是最早发展起来的控制策略之一,由于其算法简单、鲁棒性强、可靠性高的特点,得到广泛应用。为解决由于作业的实际执行时间 E_{ij}与作业最坏执行时间 C_{ij}之间的误差导致 MDTIC 策略效果下降的问题,在 MIEDF 调度算法中加入 PID 控制策略,根据 E_{ij}的反馈结果动态改变 C_{ij},降低误差带来的影响。PID 控制流程如图 3 所示。

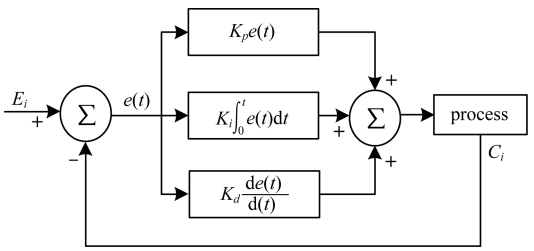

图 3　PID 控制流程

在 t 时刻 C_i 的值为:

$$C_i = K_p[e(t) + 1/K_i \int_0^t e(t)\mathrm{d}t + K_d \times \mathrm{d}e(t)/\mathrm{d}t] \tag{13}$$

其中,$e(t)$为 t 时刻 E_{ij}与 C_{ij}之间的差值,K_p 为比例系数,K_i 为积分时间常数,K_d 为微分时间常数。

PID 控制程序是周期执行的,因此可以将其作为软实时周期任务加入到 TR 中,在系统空闲时对任务的 C_{ij}进行动态调整并更新 TC,在充分利用系统空闲资源的同时保证 MDTIC 策略的效果。

3.5　算法实现

MIEDF 调度算法伪代码具体如下:

```
Procedure:Overload_determination(t,T_ij)
begin
if(Ck≤Mat) Mat = Mat - Ck return overload = 0;
if(Ck > Mat) {
compute Mdt;
if Ck≤Mdt Mat = 0 return overload = 0;
```

```
if Ck > Mdt return overload = 1;
}
end
Procedure:Overload_treatment (overload,TC)
begin
if(overload = = 1){
select TB from TC;
}
return TB;
end
Procedure:Schedule (Tij)
begin
if(τi ∈ TB) {Tij refuse executes;TB = TB - τi}
if(τi ∉ TB) {Tij executes}
end
```

在算法中,*overload* =0 表示系统未过载,*overload* =1 表示系统过载。

3.6 性能分析

实时系统最重要的是满足时间约束,本文主要在时间复杂度上对 MIEDF 调度算法的性能进行分析,由于作业调度中没有增加新元素,因此本节主要分析过载判定、过载处理、PID 控制的时间复杂度。

由于 *Mat* 可以在系统设计阶段计算得到,因此过载判定阶段的时间主要消耗在 *Mdt* 计算上,由式(9)可知,计算 *Mdt* 需要遍历一遍 Dt_i,计算时间复杂度为 $O(N\_\tau)$,其中 N_τ 为周期任务 τ_i 的个数。

过载处理阶段主要的时间消耗在 *TB* 计算上,由式(10)~式(12)可知,计算 *TB* 需要根据 $L(TB)$ 遍历一遍 *TC*,计算时间复杂度为 $O(N\_TC)$,N_TC 为 *TC* 的大小。

PID 控制主要的时间消耗在 C_i 计算上,由式(13)可知,每个周期任务需要计算一次 C_i,因此计算时间复杂度为 $O(N\_\tau)$。

综上所述,一次偶发任务释放在最坏情况下的时间消耗为 $O(2\times N\_\tau + N\_TC)$,由于实时作业通常是短作业,而且不是每次都需要计算 *Mdt*,并且 PID 控制阶段利用调度中的空闲时间,因此 MIEDF 调度算法的时间消耗通常较小,从而证明 MIEDF 调度算法的可行性与高效性。

4 实验结果与分析

本文实验硬件环境为 STM32 嵌入式微处理器,主频为 72 MHz。软件环境为 Keil MDK-ARM 上基于 C 语言的编程实现。在保证系统周期任务负载小于 1 的情况下,设置 8 个无依赖周期任务(τ_i)的最坏执行时间(C_i)、周期(P_i)、价值(V_i),时间单位为 0.1 s。周期任务的实验参数见表 2。

表 2 周期任务相关实验参数设置

τ_i	P_i	C_i	V_i
τ_1	9	1	3
τ_2	12	2	5
τ_3	15	3	7
τ_4	18	1	5
τ_5	30	1	2
τ_6	45	2	8
τ_7	60	1	10
τ_8	90	3	15

偶发任务的释放时间、执行时间和截至时期是随机的,为参照偶发任务的产生条件,按照一定的任务密度(λ)随机释放偶发任务。实验中设计的 λ 表示在 100 μs 内平均执行时间为 0.7 μs 的偶发任务个数,偶发任务的平均价值为 5。由于偶发任务的释放时间随机,不能连续释放,因此设计偶发任务的释放时间 $R_k = \frac{k}{n}\times H + randm$,其中,$n$ 为一个周期内要调度执行的偶发任务个数,*randm* 为 $(0,\frac{H}{n})$ 内的随机数。

4.1 评价指标

通过 4 种不同的评价指标进行实验分析:

1)作业成功率(*JSR*)。作业成功率是在截止期前完成的作业数(*success_num*)与需要调度的总作业数(*total_num*)的比值。

$$JSR = \frac{success\_num}{total\_num} \tag{14}$$

2)价值积累(*TV*)。价值积累是作业完成所积累的价值总和。

$$TV = \sum V_{ij} \tag{15}$$

3)系统资源利用率(*SRU*)。系统资源利用率是完成作业所用时间与系统总运行时间(*total_t*)的比值,是评价系统性能的重要指标。

$$SRU = \frac{\sum E_{ij}}{total\_t} \tag{16}$$

4)系统调度耗时(*SSTC*)。系统调度耗时是系统用于调度所花费的时间总和。

4.2 结果分析

为验证本文策略的有效性和高效性,在不同任务密度 λ 的条件下,将 MIEDF 调度算法与 EDF 调度算法、基于价值率最高优先策略的 HPEDF 调度算法[18]、基于贪心策略的 GSEDF 调度算法[19]进行比较。

为验证算法在无过载、轻度过载、重度过载等情况下的调度能力,在不同的 λ 下进行多次实验和比较。实验中 λ 取 30、40、50、60、70、80、90,其中,λ 为

30、40 表示轻度过载,λ 为 50、60、70 表示中度过载,λ 为 80、90 表示重度过载。

为保证实验的客观性与公平性,基于 4 种调度算法的偶发实时系统运行 100 s,反复进行 20 次实验,记录不同 λ 时的作业成功率、价值积累、系统资源利用率和系统调度耗时,实验结果如图 4、表 3 所示。

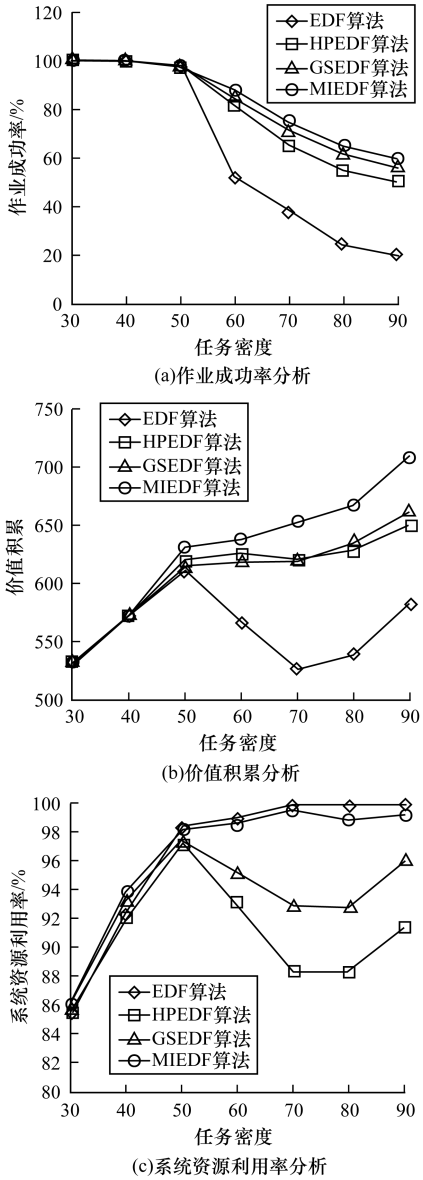

图 4　4 种算法性能比较结果

表 3　4 种算法调度耗时比较结果　　s

λ	EDF 算法	HPEDF 算法	GSEDF 算法	MIEDF 算法
40	13.2	14.1	26.4	16.8
50	13.1	13.9	30.5	16.7
60	12.5	13.4	27.8	17.3
70	11.1	12.0	25.0	16.9
80	9.9	10.7	24.7	16.5

由图 4(a)可知,在任务完成数量方面 MIEDF 算法要优于其他 3 种算法,系统处于重度过载时尤为明显。在无过载时,4 种算法的作业成功率相同,都能保证所有任务在其截至时期前完成;在轻度过载时,EDF 算法由于自身缺少过载处理能力的缺陷,因此作业成功率下降明显,虽然本文算法的作业成功率也会因非精确计算而下降,但要优于 HPEDF 算法和 GSEDF 算法;在重度过载时,MIEDF 算法的任务成功率仍能保持在 60% 以上,而 HPEDF 算法和 GSEDF 算法降低到 50% 左右,EDF 算法则低至 20% 。实验结果证明了本文算法在过载情况下尤其是重度过载下相比其他过载调度算法可以完成更多的任务。由图 4(b)可知,相比于其他 3 种算法,MIEDF 算法在不同的过载情况下都能积累较多价值。在无过载时,4 种算法所积累价值几乎相同;在轻度过载时,HPEDF 算法维持在稳定状态,MIEDF 算法和 GSEDF 算法依然能保持价值积累上升状态,这说明本文算法和 GSEDF 算法可以有效解决任务之间因相互抢占而导致的资源浪费问题,且本文算法价值增长较高;在重度过载时,所有算法都因执行高价值的偶发任务而出现价值积累上升的状况。实验结果证明了 MIEDF 算法可以有效解决任务因过载而产生的级联抢占问题,同时保证执行价值最高的任务。由图 4(c)可知,MIEDF 算法在过载时系统资源利用率保持在 98% 左右。在系统无过载时,4 种算法的系统资源利用率基本相同,在轻度过载时 MIEDF 算法与 EDF 调度算法的系统资源利用率保持在满载的状态,而其他 2 种改进算法的系统资源利用率随着过载程度的增加而下降,尤其是 HPEDF 算法,为了降低过载舍弃了过多的任务,导致系统空闲时间的增加。在重度过载时,系统资源都集中于调度偶发任务,因此系统资源利用率有所回升,但都不高于 MIEDF 算法,实验结果表明本文策略舍弃的任务及调度算法占用的系统资源均较少,进一步证明 MIEDF 算法的高效性。由表 3 可知,MIEDF 算法的运行时间较短,明显优于 GSEDF 算法。在不同的过载状况下,本文算法消耗的时间基本相同,稳定性较好,虽然时间略高于 EDF 算法,但通过牺牲少量系统资源和存储空间可以换取更多的任务在截止期前完成,同时证明本文策略是可行的。综上所述,MIEDF 调度算法的性能优于其他算法。

5　结束语

偶发实时系统容易因偶发任务的抢占发生系统过载,增大了任务截止期错失率,降低了系统性能。为解决系统过载并保障系统服务质量,本文提出 MDTIC 策略。在过载处理时利用候选解表快速选取拒绝执行的作业,保证系统资源利用率最大化,同时实现 MIEDF 调度算法,从而证明 MDTIC 策略的有效性和可行性。下一步考虑将 MIEDF 调度算法运用于其他实时系统中,扩展 MDTIC 策略的应用范围。

参考文献

[1] 张宏海,李成忠,陈祝亚.嵌入式实时系统[J].安徽工业大学学报(自然科学版),2003,20(1):58-61.

[2] 张晶,曾宪云.嵌入式系统概述[J].电测与仪表,2002,39(4):41-43.

[3] 邓昌义,郭锐锋,张忆文,等.面向硬实时系统零星任务低调度算法[J].小型微型计算机系统,2016,37(1):157-161.

[4] WU Jian. A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults[J]. IEEE Transactions on Computers, 2003, 52(3): 362-372.

[5] 秦啸,庞丽萍,韩宗芬,等.分布式实时系统的容错调度算法[J].计算机学报,2000,23(10):1056-1063.

[6] SABEGHI M, NAGHIBZADEH M, TAGHAVI T. Scheduling non-preemptive periodic tasks in soft real-time systems using fuzzy inference[C]//Proceedings of the 2nd International Conference on Information and Communication Technologies. Washington D. C., USA: IEEE Press, 2006: 27-32.

[7] 桑磊,陆阳,俞磊.基于贪心策略的 EDF 调度算法优[J].计算机工程,2015,41(12):96-100.

[8] MEJAALVAREZ P, MELHEM R, MOSS D, et al. An incremental server for scheduling overloaded real-time systems[J]. IEEE Transactions on Computers, 2003, 52(10): 1347-1361.

[9] 夏家莉,王文乐,曹重华.一种适用于实时系统的过载控制策略 HP-OMS[J].计算机应用研究,2013,30(6):1674-1678.

[10] 秦承刚,于东,吴文江,等.基于 Lebesgue 采样的动态反馈实时调度模型[J].计算机工程,2010,36(19):1-4.

[11] 张杰.最早截止期优先实时调度算法研究[D].武汉:华中科技大学,2009.

[12] DAVIS R I, BURNS A. A survey of hard real-time scheduling for multiprocessor systems[J]. ACM Computing Surveys, 2011, 43(4): 21-44.

[13] RAMAMRITHAM K, STANKOVIC J A. Scheduling algorithms and operating systems support for real-time systems[J]. Proceedings of the IEEE, 1994, 82(1): 55-67.

[14] 邹勇,李明树,王青.开放式实时系统的调度理论与方法分析[J].软件学报,2003,14(1):83-90.

[15] LIN Caixue, BRANDT S A. Improving soft real-time performance through better slack reclaiming[C]//Proceedings of IEEE International Real-Time Systems Symposium. Washington D. C., USA: IEEE Press, 2005: 410-421.

[16] AZIM A. Overloads in compositional embedded real-time control systems[C]//Proceedings of International Symposium on Rapid System Prototyping. Washington D. C., USA: IEEE Press, 2017: 51-57.

[17] ANG K H, LI Yun. PID control system analysis, design, and technology[J]. IEEE Transactions on Control Systems Technology, 2005, 13(4): 559-576.

[18] 王鹏超.过载条件下价值率优先的实时系统任务调度算法的研究[D].合肥:安徽大学,2016.

[19] CHENG Zhuo, ZHANG Haitao, TAN Yasuo, et al. Greedy scheduling with feedback control for overloaded real-time systems[C]//Proceedings of IFIP/IEEE International Symposium on Integrated Network Management. Washington D. C., USA: IEEE Press, 2015: 934-937.

编辑 陆燕菲

(上接第 107 页)

[3] GENG Haijun, SHI Xinggang, YIN Xia, et al. Algebra and algorithms for multipath QoS routing in link state networks[J]. Journal of Communications and Networks, 2017, 19(2): 189-200.

[4] GENG Haijun, SHI Xingang, WANG Zhiliang, et al. A hop-by-hop dynamic distributed multipath routing mechanism for link state network[J]. Computer Communications, 2018, 116: 225-239.

[5] YANG Yuan, XU Mingwei, LI Qi. Tunneling on demand: a lightweight approach for IP fast rerouting against multi-link failures[C]//Proceedings of International Symposium on Quality of Service. Washington D. C., USA: IEEE Press, 2016: 1-10.

[6] 杨芫.基于路径约束的互联网域内路由可用性与节能研究[D].北京:清华大学,2013.

[7] 张宝宝.互联网域内二维路由体系和算法研究[D].北京:清华大学,2015.

[8] ATLAS A K, ZININ A. Basic specification for IP fast reroute: loop-free alternates[EB/OL]. [2018-06-25]. https://www.heise.de/netze/rfc/rfcs/rfc5286.shtml.

[9] Cisco IOS XR routing configuration guide[EB/OL]. [2018-06-25]. https://www.cisco.com/c/en/us/td/docs/routers/crs/software/crs_r4-2/routing/configuration/guide/b_routing_cg42crs.pdf.

[10] MARKOPOULOU P, FRANCOIS P, BONAVENTURE O, et al. An efficient algorithm to enable path diversity in link state routing networks[J]. Computer Networks, 2011, 55(5): 1132-1149.

[11] GENG Haijun, SHI Xingang, YIN Xia, et al. Dynamic distributed algorithm for computing multiple next-hops on a tree[C]//Proceedings of IEEE International Conference on Network Protocols. Washington D. C., USA: IEEE Press, 2013: 1-10.

[12] RETVARI G, CSIKOR L, TAPOLCAI J, et al. Optimizing IGP link costs for improving IP-level resilience[J]. Computer Communications, 2013, 36(6): 645-655.

[13] GÁBOR R, JÁNOS T, GÁBOR E, et al. IP fast reroute: loop free alternates revisited[C]//Proceedings of 2011 IEEE INFOCOM. Washington D. C., USA: IEEE Press, 2011: 2948-2956.

[14] ATLAS A. U-turn alternates for IP/LDP fast-reroute[EB/OL]. [2018-06-20]. https://datatracker.ietf.org/doc/html/draft-atlas-ip-local-protect-uturn-01.txt.

[15] PAOLO N, SIU K Y, TZENG H Y. New dynamic SPT algorithm based on a ball-and-string model[J]. IEEE/ACM Transactions on Networking, 2001, 9(6): 706-718.

[16] Advanced networking for research and education[EB/OL]. [2018-06-28]. https://www.internet2.edu/products-services/advanced-networking.

[17] SPRING N, MAHAJAN R, WETHERALL D, et al. Measuring ISP topologies with rocketfuel[EB/OL]. [2018-06-25]. https://research.cs.washington.edu/networking/rocketfuel/papers/sigcomm2002.pdf.

编辑 吴云芳