软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(12):2894−2904 [doi: 10.3724/SP.J.1001.2011.03955]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

容错优先级可提升的抢占阈值容错调度算法*

丁万夫[1,2+], 郭锐锋[2], 秦承刚[1,2], 刘 娴[1,2], 郭凤钊[1,2]

[1](中国科学院 研究生院,北京 100049)

[2](中国科学院 沈阳计算技术研究所,辽宁 沈阳 110004)

Preemption Threshold Scheduling Algorithm with Higher Fault-Tolerant Priority

DING Wan-Fu[1,2+], GUO Rui-Feng[2], QIN Cheng-Gang[1,2], LIU Xian[1,2], GUO Feng-Zhao[1,2]

[1](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[2](Shenyang Institute of Computing Technology, The Chinese Academy of Sciences, Shenyang 110004, China)

+ Corresponding author: E-mail: dingwanfu@sict.ac.cn

Ding WF, Guo RF, Qin CG, Liu X, Guo FZ. Preemption threshold scheduling algorithm with higher fault-tolerant priority. *Journal of Software*, 2011,22(12):2894−2904. http://www.jos.org.cn/1000-9825/3955.htm

Abstract: Based on the worst-case response time (WCRT) schedulability analysis for hard real-time systems, a new scheduling algorithm called extended fault-tolerant fixed-priority with preemption threshold (FT-FPPT*) is proposed in the software fault-tolerant model. This algorithm can be used, together with the schedulability analysis, to effectively enhance the fault-tolerant capability when the traditional fault-tolerant fixed-priority preemptive (FT-FPP) scheduling and fault-tolerant fixed-priority scheduling with preemption threshold (FT-FPPT) are no longer appropriate. At length, an optimal priority assignment search algorithm (PASA) is presented. PASA is optimal in the sense that the fault resilience of task sets is maximized for the proposed analysis. The effectiveness of the proposed approach is also evaluated by simulation.

Key words: real-time system; fault-tolerant scheduling; preemption threshold scheduling; schedulability analysis

摘 要: 基于软件容错模型,提出了允许容错优先级提升的抢占阈值容错调度算法(extended fault-tolerant fixed-priority with preemption threshold,简称 FT-FPPT*).该算法能够在抢占式容错调度算法(fault-tolerant fixed-priority preemptive,简称 FT-FPP)和抢占阈值容错调度算法(fault-tolerant fixed-priority with preemption threshold,简称 FT-FPPT)无法提高系统容错能力的情况下,进一步提高系统的容错能力.为了获得系统中任务优先级分配的最佳策略,基于任务最坏响应时间的可调度性分析,提出了一种最优的优先级配置搜索算法(priority assignment search algorithm,简称 PASA).经过深入分析和实验证明,与 FT-FPPT 算法相比,FT-FPPT*算法能够有效地提高硬实时系统的容错能力.

关键词: 实时系统;容错调度;抢占阈值调度;可调度性分析

中图法分类号: TP316 **文献标识码**: A

硬实时系统在军事、经济、核工业等多个关键领域发挥着极其重要的作用,该类系统具备两个典型特征[1]:

* 基金项目: 国家科技重大专项(2009ZX04009-022)
收稿时间: 2010-01-22; 修改时间: 2010-07-28; 定稿时间: 2010-11-03

严格的实时性以及高度的可靠性.其中:实时性是指系统必须保证所有实时任务在其截止期内完成;可靠性是指当系统的硬件或软件出现故障时,实时任务仍可以正常运行.容错实时调度算法是硬实时系统实时性和可靠性保障的重要手段,为此,研究人员提出了多种容错模型.软件容错模型[2−7]是一种广泛采用的容错调度模型.在该模型中,每个任务由两个版本组成:主版本和替代版本.主版本是在无错误发生时系统调度的任务版本.每个主版本都对应着一个或多个替代版本,替代版本是对主版本相同功能的不同实现,且两者只要求完成一个即可.

对于固定优先级实时调度算法,文献[8,9]提出了速率单调 RM(rate monotonic)算法和时限单调 DM(deadline monotonic)算法.RM 和 DM 两种算法具有运行开销小和可预测性好的优点,被广泛应用于硬实时系统的任务调度中.以这两种算法为基础,研究人员进一步提出了相应的容错调度算法.文献[2]提出了 BCE 算法,为软件容错模型提供了一种有效的实时调度策略.在该算法中,主版本的预测精度是影响调度性能的关键.为了提高任务可执行性的预测精度,文献[3,4]对主版本的可执行性进行精确预测,避免了任务早期的失败对后续任务的影响.文献[5]不仅考虑了如何尽可能多地执行主版本,还考虑了如何减少抢占次数,从而减少了系统中因抢占次数过多带来的额外运行时调度开销等负面因素.文献[6]基于软件容错模型提出了一种高效的调度算法,该算法将单处理器 RM 算法扩展到容错多处理机上,能够在调度过程中最大限度地利用替代版本重叠和分离技术来提高算法调度性能.但是,RM,DM 及其扩展算法本质上都是基于单一优先级的抢占式调度,其任务运行前后的优先级始终不变,灵活性较差.而抢占阈值调度[10]是一种优先级可变的任务调度策略,任务在执行前后具有两个不同的优先级,可以通过对优先级的恰当选择获得理想的调度效果.与抢占式调度相比,这种调度策略更加通用灵活,并且能够有效减少任务间的抢占次数.文献[11−13]考虑了系统正常运行时抢占阈值调度.文献[14]对软件容错模型下的抢占阈值调度(fault-tolerant fixed-priority preemptive,简称 FT-FPP)进行了研究.当采取 FT-FPPT 算法进行调度时,若主版本发生错误,立即执行替代版本进行容错,并且系统为替代版本分配与主版本相同的常规优先级和抢占阈值.虽然这种方法实现简单,但是在硬实时系统中,当任务出错时,其响应时间肯定大于或等于在无容错系统中的响应时间,采取继承主版本常规优先级和抢占阈值的分配策略,可能会导致任务无法在截止期限内完成.针对这一问题,本文对允许容错优先级提升的抢占阈值容错调度算法(extended fault-tolerant fixed-priority with preemption threshold,简称 FT-FPPT*)进行了研究,给出了软件容错机制下的任务最坏响应时间(worst-case responsible time,简称 WCRT)的计算公式,并提出了一种计算最优优先级配置的算法(priority assignment search algorithm,简称 PASA).在理论上给出算法最优性证明的同时,大量仿真实验也表明了该算法的有效性.

1 软件容错模型

实时任务集合$\Gamma=\{\tau_1,\tau_2,\ldots,\tau_n\}$为包含 n 个周期任务的集合.集合Γ中的任务τ_i拥有多个替代版本,若假定$\overline{\tau}_i$是τ_i的替代版本中执行时间最长的,其执行时间用$\overline{C}_i$表示,则当τ_i出现错误时该任务的最坏响应时间只需考虑替代版本$\overline{\tau}_i$的执行.很显然,替代版本$\overline{\tau}_i$的周期、截止期均与τ_i相同.因此对于给定的任务τ_i,均可由四元组$\langle T_i,D_i,C_i,\overline{C}_i\rangle$表示,其中,$T_i$表示主版本$\tau_i$的周期,$D_i$表示$\tau_i$的截止期,$C_i$表示$\tau_i$的最坏执行时间,$\overline{C}_i$表示替代版本$\overline{\tau}_i$的最坏执行时间.

进行任务调度时,不但给任务τ_i分配一个常规优先级$\pi_i\in[1,2,\ldots,n]$,还分配一个抢占阈值$\gamma_i\in[\pi_i,\ldots,n]$.其中,$\pi_i$用于抢占其他的任务,而$\gamma_i$是任务运行时使用的优先级.对于任务$\tau_i$,每次开始执行时,其优先级将从$\pi_i$提升为$\gamma_i$;执行完毕后,优先级再从$\gamma_i$降为$\pi_i$.如果当前正在运行的任务是$\tau_i$,那么对于就绪任务$\tau_j$,必须有$\pi_j>\gamma_i$,$\tau_j$才能抢占$\tau_i$.而任务$\tau_i$相对应的替代版本$\overline{\tau}_i$也拥有两个优先级,分别是容错优先级$\overline{\pi}_i$和容错阈值$\overline{\gamma}_i$.对于$\Gamma$中的任务,系统根据某种固定优先级调度算法$FP(\Gamma)$来分配优先级,例如 RM 或者 DM.在本文中规定,优先级的数值越大,任务的优先级越高.

本文研究的前提条件:

(1) 仅考虑单处理器系统的暂时性软件错误.一旦任务τ_i出现错误,系统立即调度相应的替代版本$\overline{\tau}_i$进行容错处理,而当$\overline{\tau}_i$出现错误时,所采取的容错技术是重复执行$\overline{\tau}_i$;

(2) 任务之间是相互独立的,即没有由于共享资源产生的阻塞;

(3) 这里假设软件容错模型中错误出现的频率有一个上界,即两个相继出现的错误之间有最小间隔限制,用 T_E 表示.T_E 是系统容错能力的一个标准,它的值越小,系统的容错能力就越强.

2 问题的提出

文献[5]对软件容错模型下的抢占式调度(FT-FPP)进行了研究,推导出了任务最坏响应时间公式.在这种调度下,任务 τ_i 的最坏响应时间只与 T_E 有关,用 $R_i(T_E)$来表示,其计算公式为

$$R_i(T_E)=C_i+\left\lceil\frac{R_i(T_E)}{T_E}\right\rceil\cdot\max_{\pi_k\geqslant\pi_i}\bar{C}_k+\sum_{\forall j,\pi_j>\pi_i}\left\lceil\frac{R_i(T_E)}{T_j}\right\rceil\cdot C_j \tag{1}$$

当采取抢占式容错调度进行调度时,任务主版本 τ_i 只有一个优先级 π_i,其替代版本 $\bar{\tau}_i$ 的优先级与 τ_i 相等,即 $\bar{\pi}_i=\pi_i$.这种单一优先级的分配方式灵活性较差,可能造成任务无法满足其时限要求.现在对表 1 中的容错实时任务集合采取抢占式容错调度算法进行调度.当 T_E=11 时,根据公式(1)计算所得到的任务最坏响应时间 $R_i(9)$均小于各自对应的截止期,因此在 T_E=11 时,系统是可调度的;但是当 T_E=8 时,任务 τ_3 的最坏响应时间 $R_3(8)$大于其截止期 D_3,导致整个系统不可调度.该实例说明了在抢占式容错调度算法下,该系统最大的容错能力是 8.

Table 1 Limitations of FT-FPP

表 1 抢占式容错调度算法的限制

Task	Attributes				Priority	WCRT	
τ_i	T_i	C_i	$\bar{C}_i$	D_i	π_i	$R_i(9)$	$R_i(8)$
τ_1	12	1	1	12	3	2	2
τ_2	25	3	3	25	2	7	7
τ_3	34	5	5	34	1	34	**40**

文献[12]提出了抢占阈值容错调度算法(FT-FPPT),FT-FPPT 算法通过为任务分配两个优先级来减少任务之间的抢占次数.由于该算法的响应时间计算公式较为复杂,这里不再赘述,详见文献[12].当采取 FT-FPPT 算法进行调度时,系统为替代版本分配与主版本相同的常规优先级以及抢占阈值,即有 $\bar{\pi}_i=\pi_i$ 和 $\bar{\gamma}_i=\gamma_i$ 成立.对表 1 中的任务采取 FT-FPPT 进行调度,所得到的任务最坏响应时间见表 2.从任务的响应时间来看,当 T_E=8 时,无论抢占阈值如何配置,任务集始终不可调度.从系统的容错能力来看,容错能力仍为 9.从例子可以看出,与抢占式容错调度相比,系统容错能力并没有得到提高.这是因为抢占阈值容错调度采用的是优先级继承策略,以致出错任务无法利用高优先级任务的空闲时间来满足其截止期要求,最终导致整个系统不可调度.针对此问题,本文提出了容错优先级可提升的抢占阈值容错调度算法(FT-FPPT*).这种算法的目的是通过挪用高优先级任务的空闲时间来处理低优先级任务的容错,以保证出错的任务满足截止期的要求,进一步提高系统的容错能力.

Table 2 Limitations of FT-FPPT

表 2 抢占阈值容错调度算法的限制

FT-FPPT	T_E=8											
τ_i D_i	γ_i	R_i	γ_i	R_i	γ_i	R_i	γ_i	R_i	γ_i	R_i	γ_i	R_i
τ_1 12	3	2	3	2	3	**16**	3	7	3	7	3	**16**
τ_2 25	2	7	2	**31**	2	**31**	3	7	3	**30**	3	**30**
τ_3 34	1	**40**	2	31	3	**39**	1	**40**	2	31	3	**39**

3 可调度性分析

在容错实时系统中,任务 τ_i 的最坏响应时间通常包括 3 部分:(a) 任务最坏执行时间 C_i;(b) 容错部分 F_i,即由于错误引起的时间开销;(c) 抢占部分 I_i,即高优先级任务引起的抢占时间.因此,任务 τ_i 的最坏响应时间等于这 3 部分的总和,即 $C_i+F_i+I_i$.其中,任务的执行时间 C_i 是不变的,而 F_i 和 I_i 主要是基于当前执行任务的优先级.在容错优先级可提升的抢占阈值容错调度算法中,主版本任务具有两个优先级,即常规优先级 π_i 与抢占阈值 γ_i;

而替代版本具有两个优先级,即容错优先级 $\overline{\pi}_i$ 与容错阈值 $\overline{\gamma}_i$.而根据固定优先级调度算法 $FP(\Gamma)$进行任务调度时,π_i 是确定已知的.因此,如何分配任务 τ_i 的抢占阈值γ_i 以及替代版本 $\overline{\tau}_i$ 的常规优先级 $\overline{\pi}_i$ 和抢占阈值 $\overline{\gamma}_i$ 是问题的研究重点.本文提出了任务 τ_i 抢占阈值配置Ω_x、替代版本 $\overline{\tau}_i$ 的容错优先级配置 H_y 以及容错阈值配置 $\overline{H}_z$ 的概念,定义如下.

定义 1. 任务 τ_i 的抢占阈值配置 H_x 是一个 n 元组$\langle h_{x,1},h_{x,2},\ldots,h_{x,n}\rangle$,其中,$h_{x,i}=\gamma_i-\pi_i$,且有 $0\leqslant h_{x,i}\leqslant(i-1)$.

定义 2. 任务 $\overline{\tau}_i$ 的容错优先级配置Ω_y是一个 n 元组$\langle\sigma_{y,1},\sigma_{y,2},\ldots,\sigma_{y,n}\rangle$,其中,$\sigma_{y,i}=\overline{\pi}_i-\pi_i$,且有 $0\leqslant\sigma_{y,i}\leqslant(i-1)$.

定义 3. 对于给定的容错优先级配置Ω_y,任务 $\overline{\tau}_i$ 的容错阈值配置 $\overline{H}_z$ 是一个 n 元组 $\langle\overline{h}_{z,1},\overline{h}_{z,2},...,\overline{h}_{z,n}\rangle$,其中,$\overline{h}_{z,i}=\overline{\gamma}_i-\overline{\pi}_i$,且有 $0\leqslant\overline{h}_{z,i}\leqslant(i-\sigma_{y,i}-1)$.

根据 H_x,Ω_y, $\overline{H}_z$ 的定义可知,$h_{x,i}$表示γ_i相对于π_i的增量,$\sigma_{y,i}$表示 $\overline{\pi}_i$ 相对于π_i的增量,而 $\overline{h}_{z,i}$ 表示 $\overline{\gamma}_i$ 相对于 $\overline{\pi}_i$ 的增量.值得注意的是,$\overline{h}_{z,i}$ 的取值范围依赖于$\sigma_{y,i}$.当 $H_x=\Omega_y=\overline{H}_z=\langle0,0,...,0\rangle$ 时,所表示的就是 FT-FPP 调度算法;当 $H_x=\overline{H}_z$ 且Ω_y=$\langle0,0,\ldots,0\rangle$时,所表示的就是 FT-FPPT 调度算法.可以看出,FT-FPP 与 FT-FPPT 均是 FT-FPPT*的特例.

3.1 任务最坏响应时间的计算

对于任务 τ_i 最坏响应时间的计算,可以分两步进行:首先推导出任务 τ_i 最坏开始时间 S_i 的计算公式,然后推导出τ_i最坏响应时间 R_i 的计算公式.当 H_x,Ω_y, $\overline{H}_z$ 和 T_E 取不同的值时,任务的最坏开始时间 S_i 以及任务的最坏响应时间 R_i 都是不同的,可见它们均是 H_x,Ω_y, $\overline{H}_z$ 和 T_E 的函数,分别用 $S_i(x,y,z,T_E)$以及 $R_i(x,y,z,T_E)$来表示.

3.1.1 任务最坏开始时间 $S_i(x,y,z,T_E)$的计算

对于任意给定的任务τ_i,只有当抢占阈值大于等于π_i,且常规优先级低于π_i的任务才可以阻塞τ_i的运行.因此,τ_i被低优先级任务阻塞的时间可表示为

$$B_i=\max_{\gamma_j\geqslant\pi_i>\pi_j}C_j \tag{2}$$

对于给定的错误间隔 T_E,在时间间隔Δt 内发生错误的最大次数可以表示为 $\left\lceil\frac{\Delta t}{T_E}\right\rceil$.因此在$\Delta t$ 内,由错误所引起的最大时间开销可以通过 $\left\lceil\frac{\Delta t}{T_E}\right\rceil\cdot\max_{\forall\tau_k\in S_{\Delta t}}C_k$ 计算,其中,$S_{\Delta t}$ 表示Δt 内所有可能执行的任务集合.最坏情况下,任务 τ_i的开始时间 $S_i(x,y,z,T_E)$,可通过如下公式计算:

$$S_i(x,y,z,T_E)=B_i+\sum_{\forall j,\pi_j>\pi_i}\left(1+\left\lfloor\frac{S_i(x,y,z,T_E)}{T_j}\right\rfloor\right)\cdot C_j+E_i \tag{3}$$

$$E_i=\left\lceil\frac{S_i(x,y,z,T_E)}{T_E}\right\rceil\max_{\forall k,\overline{\gamma}_k\geqslant\pi_i,k\neq i}\overline{C}_k \tag{4}$$

在公式(3)中:第 1 项是 $S_i(x,y,z,T_E)$被低优先级任务阻塞的最大时间,可以通过公式(2)计算;第 2 项表示在 $S_i(x,y,z,T_E)$内,所有高优先级任务所带来的抢占时间;最后一项 E_i 表示 $S_i(x,y,z,T_E)$期间进行错误恢复的替代版本执行时间总和.E_i 可以通过公式(4)计算,$S_i(x,y,z,T_E)$可以通过迭代计算获得.迭代的初始值可以为 $B_i+\sum_{\forall j,\pi_j>\pi_i}C_j$,当两次迭代计算得到的 $S_i(x,y,z,T_E)$值相同时,迭代结束.

3.1.2 任务最坏响应时间 $R_i(x,y,z,T_E)$的计算

由于任务 τ_i 在执行过程中是否发生错误将会导致该任务有不同的响应时间,因此需要将 $R_i(x,y,z,T_E)$的计算分为两种情况:(1) 任务 τ_i 在执行过程中自身未发生错误,此时的最坏响应时间由 $R_i^{ext}(x,y,z,T_E)$ 表示,简称为外部错误最坏响应时间;(2) 任务 τ_i 在执行过程中自身发生错误,此时的最坏响应时间由 $R_i^{\mathrm{int}}(x,y,z,T_E)$ 表示,简称为内部错误最坏响应时间.显然,任务 τ_i 的最坏响应时间 $R_i(x,y,z,T_E)$为 $R_i^{ext}(x,y,z,T_E)$ 和 $R_i^{\mathrm{int}}(x,y,z,T_E)$ 的最大值,

亦即

$$R_i(x,y,z,T_E)=\max(R_i^{ext}(x,y,z,T_E),R_i^{\text{int}}(x,y,z,T_E)) \tag{5}$$

下面将分别对 $R_i^{ext}(x,y,z,T_E)$ 和 $R_i^{\text{int}}(x,y,z,T_E)$ 进行分析.

(a) $R_i^{ext}(x,y,z,T_E)$ 的计算

外部错误是指任务 τ_i 在执行过程中,其他任务发生错误的情况.此时,由于 τ_i 没有发生错误,因此,τ_i 始终在其抢占阈值 γ_i 上执行.容错部分 F_i 需要考虑容错阈值大于或等于 π_i 的任务.由于此时考虑的是外部错误情况,因此,在容错部分需要排除任务 τ_i 自身发生错误的情况.除此之外,F_i 部分应当排除在 $S_i(x,y,z,T_E)$中重复计算的 E_i.抢占部分 I_i 仅需考虑常规优先级高于 γ_i 的任务;同理,I_i 部分也需要排除在 $S_i(x,y,z,T_E)$中重复计算的情况.所以,在 FT-FPPT* 调度下,任务 τ_i 的外部错误最坏响应时间 $R_i^{ext}(x,y,z,T_E)$ 可通过如下公式计算:

$$\begin{aligned}R_i^{ext}(x,y,z,T_E)=S_i(x,y,z,T_E)+C_i+\left(\left\lceil\frac{R_i^{ext}(x,y,z,T_E)}{T_E}\right\rceil\right)\max_{\forall k,\bar{\gamma}_k\geqslant\pi_i,k\neq i}\bar{C}_k-E_i+\\ \sum_{\forall j,\pi_j>\gamma_i,j\neq i}\left(\left\lceil\frac{R_i^{ext}(x,y,z,T_E)}{T_j}\right\rceil-\left(1+\left\lfloor\frac{S_i(x,y,z,T_E)}{T_j}\right\rfloor\right)\right)\cdot C_j\end{aligned} \tag{6}$$

(b) $R_i^{\text{int}}(x,y,z,T_E)$ 的计算

如图 1 所示,假设任务 τ_i 在时刻 t 第 1 次发生错误,则 τ_i 的内部错误最坏响应时间 $R_i^{\text{int}}(x,y,z,T_E)$ 可以分为两个部分:τ_i 在时刻 t 前的响应时间 $R_i^{\text{int}^0}(x,y,z,T_E)$ 和在时刻 t 后的响应时间 $R_i^{\text{int}^1}(x,y,z,T_E)$.由于任务 τ_i 在错误出现后的执行情况与错误前的执行无关,因此可以先推导出 $R_i^{\text{int}^1}(x,y,z,T_E)$ 的计算公式,然后再根据 $R_i^{\text{int}^1}(x,y,z,T_E)$ 推导出 $R_i^{\text{int}^0}(x,y,z,T_E)$ 的计算公式,最后推导出 $R_i^{\text{int}}(x,y,z,T_E)$ 的计算公式.下面将按照这种方法进行分析.

首先考虑 $R_i^{\text{int}^1}(x,y,z,T_E)$ 的计算公式.对于抢占阈值大于或等于容错优先级 $\bar{\pi}_i$ 的任务,其替代版本也能抢占 $\bar{\tau}_i$ 的执行,在这其中,当然也包含了 $\bar{\tau}_i$ 再次发生错误的情况.由于在 $R_i^{\text{int}^1}(x,y,z,T_E)$ 期间最多可能发生 $\left\lceil\frac{R_i^{\text{int}^1}(x,y,z,T_E)}{T_E}\right\rceil$ 次错误,而其中第 1 个错误就是任务 τ_i 在时刻 t 发生的错误,而其余 $\left(\left\lceil\frac{R_i^{\text{int}^1}(x,y,z,T_E)}{T_E}\right\rceil-1\right)$ 个错误可能来自抢占阈值高于容错优先级 $\bar{\pi}_i$ 的任务.对于抢占部分,当任务 τ_i 发生错误后,其替代版本 $\bar{\tau}_i$ 开始在其容错阈值 $\bar{\gamma}_i$ 上执行,则常规优先级大于 $\bar{\gamma}_i$ 的任务才可以抢占 $\bar{\tau}_i$ 的执行.根据上面的分析,可以得到 $R_i^{\text{int}^1}(x,T_E)$ 的计算公式为

$$R_i^{\text{int}^1}(x,y,z,T_E)=\bar{C}_i+\left(\left\lceil\frac{R_i^{\text{int}^1}(x,y,z,T_E)}{T_E}\right\rceil-1\right)\cdot\max_{\bar{\gamma}_k\geqslant\pi_i}\bar{C}_k+\sum_{\forall j,\pi_j>\bar{\gamma}_i,j\neq i}\left\lceil\frac{R_i^{\text{int}^1}(x,y,z,T_E)}{T_j}\right\rceil\cdot C_j \tag{7}$$

现在来考虑 $R_i^{\text{int}^0}(x,y,z,T_E)$ 的计算公式.对于容错部分,只有容错阈值大于或等于 π_i 的替代版本才有可能执行.值得注意的是,若 $\sigma_{y,i}$=0,即 $\bar{\pi}_i=\pi_i$,最坏情况是所有的错误都发生在任务 τ_i 或者 $\bar{\tau}_i$ 上,此时 $\bar{C}_i=\max_{\bar{\gamma}_k\geqslant\pi_i}(\bar{C}_k)$.因此当 $\sigma_{y,i}$=0 时,需要考虑 τ_i 自身发生错误的情况;若 $\sigma_{y,i}$>0,即 $\bar{\pi}_i>\pi_i$,容错部分则需要排除 τ_i 自身发生错误的情况.这里,我们用集合 Ψ来表示这两种情况:

$$\Psi=\begin{cases}\bar{\gamma}_k\geqslant\pi_i, & \text{if } \sigma_{y,i}=0\\ \bar{\gamma}_k\geqslant\pi_i,k\neq i, & \text{if } \sigma_{y,i}>0\end{cases} \tag{8}$$

在 $R_i^{\text{int}^0}(x,y,z,T_E)$ 期间,可以抢占 τ_i 执行的任务,其常规优先级应该大于 γ_i.但是在 $R_i^{\text{int}^1}(x,y,z,T_E)$ 内已经考虑了大于 $\bar{\gamma}_i$ 的情况,这就意味着应该减去在 $R_i^{\text{int}^1}(x,y,z,T_E)$ 内已经考虑的抢占开销.与抢占部分一样,容错部分也需要排除重复计算的情况.因此,$R_i^{\text{int}^0}(x,y,z,T_E)$ 的最坏响应时间计算公式为

$$R_i^{\text{int}^0}(x,y,z,T_E)=S_i(x,y,z,T_E)+C_i+\left(\left\lceil\frac{R_i^{\text{int}}(x,y,z,T_E)}{T_E}\right\rceil-\left\lceil\frac{R_i^{\text{int}^1}(x,y,z,T_E)}{T_E}\right\rceil\right)\cdot\max_{\psi}\bar{C}_k-E_i+$$
$$\sum_{\forall j,\bar{\gamma}_i\geqslant\pi_j>\gamma_i,j\neq i}\left(\left\lceil\frac{R_i^{\text{int}^0}(x,y,z,T_E)}{T_j}\right\rceil-\left(1+\left\lfloor\frac{S_i(x,y,z,T_E)}{T_j}\right\rfloor\right)\right)\cdot C_j+ \tag{9}$$
$$\sum_{\forall j,\pi_j>\bar{\gamma}_i,j\neq i}\left(\left\lceil\frac{R_i^{\text{int}}(x,y,z,T_E)}{T_j}\right\rceil-\left(1+\left\lfloor\frac{S_i(x,y,z,T_E)}{T_j}\right\rfloor\right)-\left\lceil\frac{R_i^{\text{int}^1}(x,y,z,T_E)}{T_j}\right\rceil\right)\cdot C_j$$

基于以上分析,任务τ_i的内部错误最坏响应时间$R_i^{\text{int}^1}(x,y,z,T_E)$为$R_i^{\text{int}^0}(x,y,z,T_E)$与$R_i^{\text{int}^1}(x,y,z,T_E)$之和,即

$$R_i^{\text{int}}(x,y,z,T_E)=R_i^{\text{int}^1}(x,y,z,T_E)+R_i^{\text{int}^0}(x,y,z,T_E) \tag{10}$$

当系统中所有任务的最坏响应时间均不大于其截止期时,任务集Γ是可调度的;否则,任务集Γ不可调度.

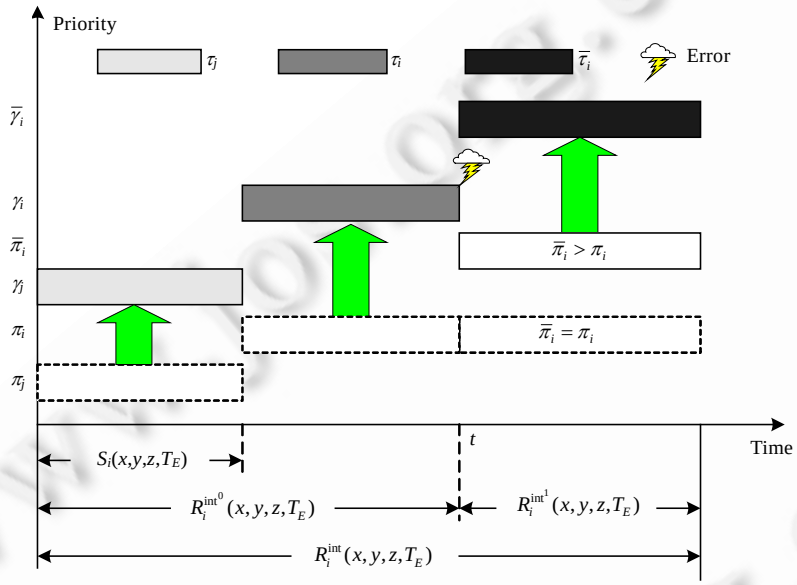

Fig.1 Illustration of the derivation of $R_i^{\text{int}}(x,y,z,T_E)$

图 1 $R_i^{\text{int}}(x,y,z,T_E)$的计算过程

从上面的分析可知,FT-FPP 调度是 FT-FPPT*调度的特例.当所有替代任务的容错优先级与容错阈值均未发生变化时,两种调度所计算的最坏响应时间应该相等,由下面的定理 1 可以证明.

定理 1. 一个容错实时任务集合$\Gamma=\{\tau_1,\tau_2,\ldots,\tau_n\}$,对于任意的错误间隔 $T_E>0$,当$H_x=\Omega_y=\bar{H}_z=\langle 0,0,...,0\rangle$时,FT-FPP 与 FT-FPPT*两种调度计算所得的最坏响应时间相等.

证明:当$H_x=\Omega_y=\bar{H}_z=\langle 0,0,...,0\rangle$时,有$\bar{\gamma}_i=\bar{\pi}_i=\gamma_i=\pi_i$成立.

由公式(6)和公式(10)可得:

$$R_i^{ext}(T_E)=C_i+\left\lceil\frac{R_i^{ext}(T_E)}{T_E}\right\rceil\cdot\max_{\pi_k>\pi_i}\bar{C}_k+\sum_{\forall j,\pi_j>\pi_i}\left\lceil\frac{R_i^{ext}(T_E)}{T_j}\right\rceil\cdot C_j,$$
$$R_i^{\text{int}}(T_E)=C_i+\left\lceil\frac{R_i^{\text{int}}(T_E)}{T_E}\right\rceil\cdot\max_{\pi_k\geqslant\pi_i}\bar{C}_k+\sum_{\forall j,\pi_j>\pi_i}\left\lceil\frac{R_i^{\text{int}}(T_E)}{T_j}\right\rceil\cdot C_j+(\bar{C}_i-\max_{\pi_k\geqslant\pi_i}\bar{C}_k).$$

比较两式,显然有:

① 当 $\bar{C}_i = \max_{\pi_k \geqslant \pi_i} \bar{C}_k$ 时,有 $R_i^{\text{int}}(T_E) \geqslant R_i^{ext}(T_E)$ 成立;

② 当 $\bar{C}_i \neq \max_{\pi_k \geqslant \pi_i} \bar{C}_k$ 时,有 $R_i^{\text{int}}(T_E) \leqslant R_i^{ext}(T_E)$ 成立.

根据公式(5),可得统一表达式:

$$R_i(T_E) = C_i + \left\lceil \frac{R_i(T_E)}{T_E} \right\rceil \cdot \max_{\pi_k \geqslant \pi_i} \bar{C}_k + \sum_{\forall j, \pi_j > \pi_i} \left\lceil \frac{R_i(T_E)}{T_j} \right\rceil \cdot C_j.$$

综合以上分析,当 $H_x = \Omega_y = \bar{H}_z = \langle 0,0,...,0 \rangle$ 时,FT-FPP(公式(1))与 FT-FPPT*计算所得的最坏响应时间相等. □

3.2 实例分析

现对表 1 中的任务集合采取 FT-FPPT*算法进行调度,任务的最坏响应时间见表 3.当 T_E=8 时,由于 Ω_x=⟨0,1,0⟩,τ_2 的抢占阈值变为 3.这样,τ_1 就不能在 τ_2 执行时抢占其运行,从而缩短了 τ_2 的响应时间,使得 τ_2 能够按时完成;由于 H_y=⟨0,0,1⟩且 $\bar{H}_z = \langle 0,0,0 \rangle$,$\tau_3$ 的容错优先级和容错阈值都变为 2.这样,τ_2 就不能抢占 τ_3 执行,从而缩短了 τ_3 的响应时间,使得 τ_3 也能够按时完成.同时,τ_1 也能够满足截止期的要求,整个系统是可调度的.

从 3 种容错调度算法对系统容错能力的影响来看,在 FT-FPP 和 FT-FPPT 两种调度策略下,系统的容错能力均为 T_E=9;而在 FT-FPPT*调度策略下,系统容错能力可以达到 T_E=6,提高了 33%.这说明,FT-FPPT*能够在 FT-FPP 和 FT-FPPT 均无法提高系统容错能力的情况下,进一步提高系统的容错能力.

Table 3 Improvement of FT-FPPT* corresponding to FT-FPPT

表 3 FT-FPPT*与 FT-FPPT 容错能力的比较

FT-FPPT*	T_E=8				T_E=7		T_E=6	
τ_i D_i	Ω_x=⟨0,0,0⟩ H_y=⟨0,0,0⟩ $\bar{H}_z = \langle 0,0,0 \rangle$		Ω_x=⟨0,1,0⟩ H_y=⟨0,0,1⟩ $\bar{H}_z = \langle 0,0,0 \rangle$		Ω_x=⟨0,1,0⟩ H_y=⟨0,0,0⟩ $\bar{H}_z = \langle 0,0,1 \rangle$		Ω_x=⟨0,1,0⟩ H_y=⟨0,0,2⟩ $\bar{H}_z = \langle 0,0,0 \rangle$	
	$R_i^{\text{int}} \mid R_i^{ext}$	R_i	$R_i^{\text{int}} \mid R_i^{ext}$	R_i	$R_i^{\text{int}} \mid R_i^{ext}$	R_i	$R_i^{\text{int}} \mid R_i^{ext}$	R_i
τ_1 12	2|1	2	2|7	7	2|7	7	2|12	12
τ_2 25	7|5	7	7|5	7	7|5	7	12|5	12
τ_3 34	40|16	**40**	21|16	21	21|19	21	24|22	24

4 最优的抢占阈值配置搜索算法

基于任务最坏响应时间的可调度性分析,提出一种最优的优先级配置搜索算法 PASA(priority assignment search algorithm).该算法能够保证任务在满足其截止期的情况下,使得系统所能承受的错误发生间隔时间达到最小.

首先引入函数 $T_e(x,y,z)$.给定配置 Ω_x,H_y 和 $\bar{H}_z$,函数 $T_e(x,y,z)$ 表示 T_E 所能达到的最小值.若 T_E 比 $T_e(x,y,z)$ 的值更小,则必有任务不可调度,这样的任务称为临界任务.

定义 4. 当 $T_E = T_e(x,y,z) - 1$ 时,至少存在一个任务,使得整个集合 Γ 不可调度,称这样的任务为临界任务.所有的临界任务构成了一个集合,称为临界任务集.若用 $Z(x,y,z)$ 表示此集合,则有

$$Z(x,y,z) = \{\tau_i \in \Gamma \mid R_i^{\text{int}}(x,y,z,T_e(x,y,z)-1) > D_i \vee R_i^{ext}(x,y,z,T_e(x,y,z)-1) > D_i\}.$$

根据临界任务的属性不同,可将临界任务分为主动临界任务与被动临界任务,定义如下:

定义 5. 由于自身发生错误而导致自己最终错过时限的任务称为主动临界任务(active-task),由此类任务组成的集合称为主动临界任务集.若用 $Z_a(x)$ 表示此集合,则有 $Z_a(x,y) = \{\tau_i \in \Gamma \mid R_i^{\text{int}}(x,y,z,T_e(x,y,z)-1) > D_i\}$.

定义 6. 由于自身发生错误而导致其他任务错过时限的任务称为被动临界任务(passive-task),由此类任务组成的集合称为被动临界任务集.若用 $Z_p(x)$ 表示此集合,则有

$$Z_p(x,y) = \{\tau_i \in \Gamma \mid \exists \tau_j \in \Gamma : \bar{\pi}_i > \pi_j \wedge R_j^{ext}(x,y,z,T_e(x,y,z)-1) > D_j\}.$$

为了获得最优的双重优先级配置,最直观的方法是尝试使用每一种可能的配置方案,并分别计算每种配置下的 T_e 值,而能够获得最小 T_e 值的配置即为最优配置.为了提高搜索算法的效率,避免在优化的过程中检验所有的双重优先级配置,由公式(6)可以推导出两个必要非充分条件,以减少算法的时间复杂度,即

$$Cond^0(x,y,i,j)\equiv\exists\pi_j>\overline{\gamma}_i:\left\lceil\frac{R_i^{\text{int}}(x,y,T_E)}{T_j}\right\rceil>\left\lceil\frac{R_i^{\text{int}^0}(x,y,T_E)}{T_j}\right\rceil \tag{11}$$

$$Cond^1(x,y,i,j)\equiv\exists\pi_j>\overline{\gamma}_i:\left\lceil\frac{R_i^{\text{int}}(x,y,T_E)}{T_j}\right\rceil>\left\lceil\frac{R_i^{\text{int}^1}(x,y,T_E)}{T_j}\right\rceil \tag{12}$$

算法过程:给定 Ω_x,H_y 和 $\overline{H}_z$,若 $T_E=T_e(x,y,z)-1$,寻找使得任务集可调度的 Ω'_x,H'_y 和 $\overline{H}'_z$.如果 Ω'_x,H'_y 和 $\overline{H}'_z$ 存在,算法找到它;否则,算法停止,迭代的初始条件是 $H_x=\Omega_y=\overline{H}_z=\langle 0,0,...,0\rangle$.图 2 给出了该算法的伪代码.优先级配置搜索算法 PASA 由初始化部分(第 1 行~第 6 行)和搜索部分(第 7 行~第 40 行)组成.初始化部分的工作是按照给定算法 $FP(\Gamma)$分配任务的常规优先级,并设定 H_x,Ω_y, $\overline{H}_z$ 和 T_E 的初始值.搜索部分则完成最优优先级配置的搜索.在搜索部分中,任务的响应时间 $R_i(x,y,z,T_E)$与时限 D_i 的关系可以分为两个分支.当 $R_i(x,y,z,T_E)\leqslant D_i$ 时(第 9 行~第 15 行),即任务集可调度时,则将当前的 H_x,Ω_y, $\overline{H}_z$ 和 T_E 值分别保存在 H_x^*,Ω_y^*,$\overline{H}_z^*$ 和 T_E^* 中,并将 T_E 的值减 1,开始下一轮的搜索;当 $R_i(x,y,z,T_E)>D_i$(第 16 行~第 34 行)时,则需要提高主版本的抢占阈值或者替代版本的容错优先级与容错阈值,以减少被抢占时间.若 $T_E<L$(第 15 行)或 $\Omega_{PromotionSet}=\varnothing\cap\overline{H}_{PromotionSet}=\varnothing$ (第 32 行),则结束整个搜索过程.此时,所得到的 H_x^* 为 τ_i 的最优抢占阈值配置,Ω_y^* 为 $\overline{\tau}_i$ 的最优容错优先级配置,$\overline{H}_z^*$ 为 $\overline{\tau}_i$ 的最优容错阈值配置,而 T_E^* 则为在 FT-FPPT* 调度算法下所能承受的最小错误间隔.

现在考虑 PASA 算法的时间复杂度.当 $T_E<L$(第 15 行)以及 $\Omega_{PromotionSet}=\varnothing\cap\overline{H}_{PromotionSet}=\varnothing$ (第 32 行)时,PASA 算法停止.在整个搜索迭代过程中,由于条件 $L\leqslant T_E$ 是保证算法为真的必要条件,因此,搜索部分的时间复杂度取决于条件 $\Omega_{PromotionSet}=\varnothing\cap\overline{H}_{PromotionSet}=\varnothing$ 为真时的迭代次数.在最坏情况下,每次迭代过程集合 $Z(x,y,z)$中仅有一个临界任务,算法通过提升该任务的容错优先级和容错阈值使得该集合 Γ 变为可调度的,而每次提升容错优先级和容错阈值都需要两次迭代,第 1 次用来提升临界任务的容错优先级和容错阈值,第 2 次用来保存容错优先级配置和容错阈值配置.又因为提升所有临界任务容错优先级和容错阈值的迭代次数为 $\sum_{i=1}^{n-1}i=\frac{n\cdot(n-1)}{2}$,那么在最坏情况下,总的迭代次数为 $n\cdot(n-1)$,所以搜索空间的复杂度为 $O(n^2)$.如果假设计算任务响应时间的时间复杂度为 $O(\gamma)$,则整个 FTPCSA 算法的时间复杂度为 $O(\gamma\times n^2)$.

最后,给出 PASA 算法的最优性证明.

定义 7. 对于任意一个可调度的任务集,给定的优先级配置为最优配置,当且仅当该配置使得系统所能承受的故障发生间隔达到最小,而能够获得最优配置的算法即为最优算法.

定理 2. PASA 算法是一种最优的容错优先级配置搜索算法.

证明:PASA 算法是否为最优算法,仅需考虑最后保存到 H_x^*,Ω_y^* 和 $\overline{H}_z^*$ 的配置是否为最优配置即可.首先考虑保存到 H_x^* 的抢占阈值配置是否为最优配置.根据 H_x^* 是否等于 $H_0=\langle 0,0,\ldots,0\rangle$,可分为以下两种情况:

(1) 若 $H_x^*=H_0$,则对于 $T_E=T_e(0)-1$,该任务集必是不可调度的.此时,H_0 即为最优配置;

(2) 若 $H_x^*\neq H_0$,则至少有两个被连续保存的配置,依次假设为 H_m 和 H_n.根据 PASA 算法的执行过程可知,搜索部分使得 T_E 非递增,即 $T_e(n)\leqslant T_e(m)$.因此,该算法最后保存到 H_x^* 中的配置能够使得 T_E 达到最小.

同理可证保存到 Ω_y^* 和 $\overline{H}_z^*$ 中的容错优先级配置和容错阈值配置均为最优配置. □

Procedure:

(1) $\pi_i \leftarrow FP(\Gamma)$, $i=1,2,\ldots,n$;

(2) $H_x \leftarrow \langle 0,0,...0\rangle$; $H_x^* \leftarrow H_x$

(3) $\Omega_y \leftarrow \langle 0,0,...0\rangle$; $\Omega_y^* \leftarrow \Omega_y$

(4) $\bar{H}_z \leftarrow \langle 0,0,...0\rangle$; $\bar{H}_z^* \leftarrow \bar{H}_z$

(5) $L \leftarrow 1+\max_{\forall \tau_i \in \Gamma} \bar{C}_i$

(6) $T_E \leftarrow T_e(x,y,z)$; $T_E^* \leftarrow T_E$

(7) while (TRUE)

(8) calculate $R_i^*(x,y,z,T_E)$, $i=1,2,...,n$;

(9) if $(\forall \tau_i \in \Gamma, R_i^*(x,y,z,T_E) \leqslant D_i)$

(10) $H_x^* \leftarrow H_x$

(11) $\Omega_y^* \leftarrow \Omega_y$

(12) $\bar{H}_z^* \leftarrow \bar{H}_z$

(13) $T_E^* \leftarrow T_E$

(14) $T_E \leftarrow T_E-1$

(15) if ($T_E<L$) exit while

(16) else

(17) if $Z(x,y,z)\neq\varnothing$ let τ_i be a task in $Z(x,y,z)$

(18) $H_{PromotionSet}=\{\tau_j \in \Gamma | Cond^0(i,j) \wedge Cond^1(i,j)\}$

(19) if $H_{PromotionSet}\neq\varnothing$

(20) $h_{x,i} \leftarrow \min_{\forall \tau_j \in H_{PromotionSet}}(\gamma_j)-\pi_i$

(21) if ($|H_{PromotionSet}|=1$)$T_E \leftarrow \min(T_e(x,y,z),T_E)$

(22) else if $Z_p(x,y,z)=\varnothing$

(23) let τ_i be a task in $Z_a(x,y,z)$

(24) $\Omega_{PromotionSet}=\{\tau_j \in \Gamma | Cond^0(i,j)\}$

(25) if ($\Omega_{PromotionSet}\neq\varnothing$)

(26) $\sigma_{x,i} \leftarrow \min_{\forall \tau_j \in \Omega_{PromotionSet}}(\pi_j)-\pi_i$

(27) if ($|\Omega_{PromotionSet}|=1$)$T_E \leftarrow \min(T_e(x,y,z),T_E)$

(28) $\bar{H}_{PromotionSet}=\{\tau_j \in \Gamma \mid Cond^1(i,j)\}$

(29) if $(\bar{H}_{PromotionSet} \neq \varnothing)$

(30) $\bar{h}_{x,i} \leftarrow \min_{\forall \tau_j \in \bar{H}_{PromotionSet}}(\gamma_j)-\bar{\pi}_i$

(31) if $(|\bar{H}_{PromotionSet}|=1)T_E \leftarrow \min(T_e(x,y,z),T_E)$

(32) esle

(33) exit while

(34) end if

(35) end if

(36) end while

(37) $H_x \leftarrow H_x^*$

(38) $\Omega_y \leftarrow \Omega_y^*$

(39) $\bar{H}_x \leftarrow \bar{H}_x^*$

(40) $T_E \leftarrow T_E^*$

Fig.2 Optimal priority assignment search algorithm (PASA)

图 2 最优的优先级配置搜索算法

5 实 验

为了对比 FT-FPPT 和 FT-FPPT*两种调度算法在提高系统容错能力方面的性能,本实验将增量ΔT_e作为衡量系统容错能力好坏的一项性能指标.

$$\Delta T_e = \frac{T_e(x,0,x)-T_e(x,y,z)}{T_e(x,0,x)} \times 100\%.$$

显然,在 FT-FPPT 调度算法下,有ΔT_e=0 成立.$FP(\Gamma)$使用 DMS 算法分配任务的常规优先级π_i.即任务的截止期越短,其常规优先级越高.

本实验模拟了 5 000 个任务集,每个任务集包含 10 个任务.任务集Γ的处理器利用率$U=\sum_{\forall \tau_i \in \Gamma} C_i/T_i$,且满足$U_{\min} \leqslant U \leqslant U_{\max}$,其中,$U_{\min}$=0.01,$U_{\max}$=0.9.对于任意给定的任务$\tau_i$,其周期$T_i$、截至期$D_i$、执行时间$C_i$和替代版本的执行时间$\overline{C}_i$都是随机产生的,但需要满足如下条件:

(1) T_i和D_i分别服从区间[10,1000]上的均匀分布;

(2) $C_i=\mu \cdot D_i$,其中参数μ服从$U_{\max}/10$的指数分布;

(3) $\overline{C}_i$服从区间$[1,C_i]$上的均匀分布,因此有$1 \leqslant \overline{C}_i \leqslant C_i$成立.

如图 3 所示,黑点表示给定任务集计算出的ΔT_e值,而图中的黑线则表示同一处理器利用率下所获得ΔT_e的平均值.从图中可以看出,与 FT-FPPT 调度相比,采用 FT-FPPT*调度在容错性能上得到了明显提高.当处理器利用率在〈0.4,0.9〉变化时,ΔT_e的平均值增长幅度较大,始终保持在 10%~20%之间,ΔT_e的最大值可达到 86%.这是因为在处理器利用率较低的情况下,系统中的空闲时间较多,当发生错误时,系统可以利用这些空闲时间来达到容错的目的,因此,此时采用双重优先级分配策略对系统容错能力的提升影响不大;而在处理器利用率较高的情况下,系统中的空闲时间较少,当发生错误时,重优先级分配策略可以通过挪用高优先级任务的空闲时间来处理低优先级任务的容错,提高了系统的容错能力.

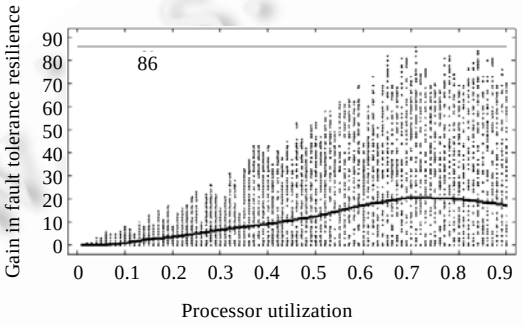

Fig.3 Gain in fault tolerance resilience under FT-FPPT*

图 3 FT-FPPT*调度下的系统容错能力的提升

6 结 论

考虑到硬实时系统对实时性以及可靠性方面的严格要求,本文对软件容错模型下的实时调度算法进行了研究.为提高硬实时系统的容错能力,减少任务间的抢占次数,本文基于软件容错模型提出了允许容错优先级可提升的抢占阈值容错调度算法(FT-FPPT*),并举例说明了在提高系统容错能力方面,FT-FPPT*要优于 FT-FPP 与 FT-FPPT 两种调度算法.根据基于任务最坏响应时间的可调度性分析,提出了一种最优的优先级配置搜索算法(PASA).通过模拟实验表明,与 FT-FPPT 调度算法相比,FT-FPPT*调度算法具有更高的系统容错能力.

References:

[1] Sha L, Abdelzaher TF, Årzén KE, Cervin A, Baker TP, Burns A, Buttazzo GC, Caccamo M, Lehoczky JP, Mok AK. Real time scheduling theory: A historical perspective. Real-Time Systems, 2004,28(2-3):101−155. [doi: 10.1023/B:TIME.0000045315.61234.1e]

[2] Han CC, Shin KG, Wu J. A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults. IEEE Trans. on Computers, 2003,52(3):362−372. [doi: 10.1109/TC.2003.1183950]

[3] Li QH, Han JJ, Essa AA, Zhang W. Dynamic scheduling algorithms with software fault-tolerance in hard real-time systems. Journal of Software, 2005,16(1):101−107 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/101.htm

[4] Liu D, Zhang CY, Li R, Huang Y, Li Y. Fault-Tolerant real-time scheduling algorithm in software fault-tolerant module. Journal of Computer Research and Development, 2007,44(9):1495−1500 (in Chinese with English abstract).

[5] Wang J, Sun JL, Wang XY, Yang XH, Wang SK. Partial preemptive real-time scheduling algorithm in software fault-tolerant model. Journal of Zhejiang University (Engineering Science), 2009,43(6):1047−1052 (in Chinese with English abstract). [doi: 10.3785/j.issn.1008-973X.2009]

[6] Wang J, Sun JL, Wang XY, Wang SK, Chen JB. Efficient scheduling algorithm for hard real-time tasks in primary- backup based multiprocessor systems. Journal of Software, 2009,20(10):2628−2636 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/577.htm [doi: 10.3724/SP.J.1001.2009.00577]

[7] Punnekkat S. Schedulability analysis for fault-tolerant real-time systems [Ph.D. Thesis]. York: University of York, 1997.

[8] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 1973, 20(1):46−61. [doi: 10.1145/321738.321743]

[9] Audsley NC, Burns A, Wellings AJ. Deadline monotonic scheduling theory and application. Control Engineering Practice, 1993, 1(1):71−78. [doi: 10.1016/0967-0661(93)92105-D]

[10] Wang Y, Saksena M. Scheduling fixed-priority tasks with preemption threshold. In: Proc. of the 6th Int'l Conf. on Real-Time Computing Systems and Applications. 1999. 328−335.

[11] Regehr J. Scheduling tasks with mixed preemption relations for robustness to timing faults. In: Proc. of the IEEE Real-Time Systems Symp. Texas: IEEE Computer Society Press, 2002. 315−326.

[12] Chen JX. Extensions to fixed priority with preemption threshold and reservation-based scheduling [Ph.D. Thesis]. Waterloo: University of Waterloo, 2005.

[13] Ghattas R, Dean AC. Preemption threshold scheduling: Stack optimality, enhancements and analysis. In: Proc. of the Real-Time and Embedded Technology and Applications Symp. IEEE Computer Society Press, 2007. 147−157. [doi: 10.1109/RTAS.2007.27]

[14] Wang L, Wu ZH, Zhao MD, Yang GQ. Fault tolerant scheduling for fixed-priority tasks with preemption threshold. In: Proc. of the Embedded and Real-Time Computing Systems and Applications. IEEE Computer Society Press, 2005. 220−225. [doi: 10.1109/RTCSA.2005.50]

附中文参考文献:

[3] 李庆华,韩建军,Essa AA,张薇.硬实时系统中基于软件容错的动态调度算法.软件学报,2005,16(1):101−107. http://www.jos.org.cn/1000-9825/16/101.htm

[4] 刘东,张春元,李瑞,黄影,李毅.软件容错模型中的容错实时调度算法.计算机研究与发展,2007,44(9):1495−1500.

[5] 王健,孙建伶,王新宇,杨小虎,王申康.软件容错模型中的部分抢占实时调度算法.浙江大学学报(工学版),2009,43(6):1047−1052. [doi: 10.3785/j.issn.1008-973X.2009]

[6] 王健,孙建伶,王新宇,杨小虎,王申康,陈俊波.容错多处理机中一种高效的实时调度算法.软件学报,2009,20(10):2628−2636. http://www.jos.org.cn/1000-9825/577.htm [doi: 10.3724/SP. J.1001.2009.00577]

丁万夫(1982－),男,辽宁营口人,博士生,主要研究领域为实时系统,容错调度.

刘娴(1989－),女,硕士生,主要研究领域为实时调度.

郭锐锋(1968－),男,博士,研究员,博士生导师,主要研究领域为实时系统,数控技术.

郭凤钊(1987－),男,硕士生,主要研究领域为实时调度.

秦承刚(1983－),男,博士生,主要研究领域为实时系统,反馈调度.