# 一种可行的容错实时系统可调度性分析*

李 俊+, 阳富民, 卢炎生

(华中科技大学 计算机科学与技术学院,湖北 武汉 430074)

## A Feasible Schedulability Analysis for Fault-Tolerant Real-Time Systems

LI Jun+, YANG Fu-Min, LU Yan-Sheng

[1](College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

+ Corresponding author: Phn: +86-27-87545247, E-mail: hustlee@126.com, http://www.hust.edu.cn

Received 2004-02-25; Accepted 2005-03-03

**Li J, Yang FM, Lu YS. A feasible schedulability analysis for fault-tolerant real-time systems. *Journal of Software*, 2005,16(8):1513−1522.** DOI: 10.1360/jos161513

**Abstract**: Based on the worst-case response time schedulability analysis for fault-tolerant real-time systems, a new fault-tolerant priority assignment algorithm is proposed. This algorithm can be used, together with the schedulability analysis, to effectively improve system fault resilience when the two traditional fault-tolerant priority assignment policies can't improve system fault resilience. A fault-tolerant priority configuration search algorithm is also presented for the proposed analysis. The effectiveness of the proposed approach is evaluated by simulation.

**Key words**: fault-tolerant real-time system; worst-case response time; schedulability analysis; fault-tolerant priority

摘 要: 针对容错实时系统中容错优先级两种分配策略存在的不足,通过对容错实时任务进行基于最坏响应时间的可调度性分析,提出了允许容错优先级降低的分配策略以提高系统的容错能力.经过深入的分析和实验证明,这种容错优先级的分配策略能够在以前两种分配策略无法提高系统容错能力的情况下,有效地提高系统的容错能力,设计并实现了改进的最佳容错优先级分配因子的搜索算法,并通过模拟实验进行了验证.

关键词: 容错实时系统;最坏响应时间;可调度性;容错优先级

中图法分类号: TP316 文献标识码: A

实时系统应具备两个特征,一是保证所有实时任务在截止期限内完成,二是系统的硬件或软件出现故障时,实时任务仍可以继续运行.因此,对于实时系统的可调度性分析更需要考虑有容错需求的实时任务.容错实时系统的可调度性分析主要是基于任务的最坏响应时间与其截止期限之间的关系.如果系统中所有任务的最坏响应时间均不大于其截止期限,则该系统是可调度的.反之,如果系统中至少存在一个任务的最坏响应时间大于其截止期限,则称该系统不可调度.如何处理这两者的关系,使最坏响应时间不大于截止期限,成为提高实时系统容错能力的主要研究方向之一.

文献[1]提出了一种独立于可调度性分析的容错调度算法,但是它所采取的容错技术只是重新执行出错任务;文献[2]基于利用任务集中的空闲时间,提出了离线可调度系统的容错恢复技术,但是它仅仅考虑了周期任

* 作者简介: 李俊(1979−),女,江西南昌人,博士生,主要研究领域为容错实时调度,嵌入式系统应用;阳富民(1966−),男,教授,主要研究领域为嵌入式操作系统;卢炎生(1949−),男,教授,博士生导师,主要研究领域为数据库系统,软件测试,信息系统.

务,并且有任务的周期等于其截止期限的限制条件;文献[3–5]主要是基于系统所能承受的故障出现的最大频率来分析系统的可调度性.这些调度算法对于容错优先级都只是采取继承任务在正常情况下的优先级的分配策略.但是在容错实时系统中,当任务出错时,其响应时间肯定大于或等于其在无容错系统中的响应时间,采取继承的分配策略可能会导致任务无法在截止期限内完成.针对这种情况,文献[6,7]提出了一种比较灵活的容错优先级分配策略——允许提高任务的容错优先级.这种分配策略在一定程度上保证了出错的任务在可能短的时间内满足截止期限,从而提高整个系统的容错能力.但是,在某种特殊情况下,允许容错优先级提高的分配策略并不能提高系统在容错优先级分配策略下的容错能力.本文通过对容错实时任务基于最坏响应时间的可调度性分析.研究和实验证明了允许任务容错优先级降低的分配策略能够有效地提高这种情况下的系统容错能力.

## 1 容错实时任务模型

定义一个实时任务集合$\Gamma=\{\tau_1,\tau_2,\ldots,\tau_n\}$,$\tau_i$ 表示任务,是在没有出现故障时系统调度的主要对象,它可以是周期任务,也可以是软非周期任务.任务 $\tau_i$ 定义为一个三元组,$\tau_i=(C_i,D_i,T_i)$,其中 $C_i$ 表示任务的执行时间,$D_i$ 表示任务的最后期限,$T_i$ 表示周期任务的周期,对于软非周期任务而言,它是任务到来的时间间隔最小值.每个任务 $\tau_i$ 拥有多个替代任务,而每个替代任务只能对应 $\tau_i$ 的一个特定容错处理,所以它们的计算时间各不相同.

由于本文是研究任务的最坏响应时间,为了方便分析,我们假设 $\bar{\tau}_i$ 是 $\tau_i$ 的替代任务中执行时间最长的,其执行时间用 $\overline{C}_i$ 表示;而且 $\tau_i$ 所有的替代任务均被分配在同一个优先级上执行,因此我们只需考虑在 $\tau_i$ 出现故障时,其对应的替代任务为 $\bar{\tau}_i$.很显然,$\bar{\tau}_i$ 与 $\tau_i$ 的截止期限相等.因此,$\tau_i$ 的整体响应时间等于 $\tau_i$ 本身的响应时间和 $\bar{\tau}_i$ 的响应时间之和.

对于 $\Gamma$ 中的任务,可以根据某种静态优先级调度算法 $FP(\Gamma)$ 来分配优先级,例如:RMS(rate monotonic scheduling)[8]或 DMS(deadline monotonic scheduling)[9].根据 $FP(\Gamma)$,每个任务被分配到各不相同的优先级.下面给出任务优先级的定义:

定义 **1**. $p_i$ 表示任务 $\tau_i$ 根据 $FP(\Gamma)$ 所分配的优先级别.$p_i\in\{1,2,\ldots,n\}$,且 $\forall i,j,i\neq j,p_i\neq p_j$.若 $p_i=n$,表示 $\tau_i$ 的优先级最高;而当 $p_i=1$ 时,表示 $\tau_i$ 的优先级最低.

我们用 $\bar{p}_i$ 表示 $\tau_i$ 的容错优先级,即 $\bar{\tau}_i$ 的优先级.本文研究的前提条件是:

(1) 若 $\tau_i$ 与 $\bar{\tau}_j$ 同时在同一优先级上处于就绪状态,则 $\bar{\tau}_j$ 先被执行.

(2) 只考虑单处理器系统的暂时性软件故障,所采取的容错技术可以是恢复块技术、异常处理或任务的重复执行.一旦任务 $\tau_i$ 出现故障,系统将立即调度相应的替代任务进行容错处理.而当替代任务 $\bar{\tau}_i$ 出现故障时,所采取的容错技术是重复执行 $\bar{\tau}_i$.另外,假设系统在任务与对应的替代任务之间的调度切换无损耗,而且故障出现时只会影响当前正在执行的任务,对其他任务不会产生任何影响.

(3) 这里假设容错模型为故障出现的频率有上界,即两个相继出现的故障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

在非容错实时系统中,无论是采取 RMS 还是 DMS 中的任何一种静态调度算法,任务的最坏响应时间 $R_i$ 均包括任务自身的执行时间 $C_i$ 和所有 $p_j>p_i$ 任务抢占时间的总和 $I_i$ 这两部分.而当系统考虑容错时,$R_i$ 就不仅包括 $C_i$ 和 $I_i$,而且还不得不将由于故障所引起的替代任务执行时间的总和 $F_i$ 考虑进来.因此,任务 $\tau_i$ 的最坏响应时间计算公式为

$$R_i=C_i+I_i+F_i \tag{1}$$

从式(1)可以看出,由于对于任何任务,其自身的执行时间 $C_i$ 是固定不变的,因而如何缩短 $R_i$ 就转变为如何缩短抢占时间 $I_i$ 和容错时间 $F_i$ 的问题.$I_i$ 和 $F_i$ 主要是基于当前执行任务的优先级.当前执行任务可能是正常情况下的任务,也可能是故障出现后进入执行状态的替代任务,这就必然涉及到系统对任务优先级 $p_i$ 和容错优先级 $\bar{p}_i$ 的分配策略.而根据某种静态优先级调度算法 $FP(\Gamma)$ 来分配任务优先级的情况,$p_i$ 是确定的.因而,我们所要讨论的是容错优先级 $\bar{p}_i$ 的分配策略.

文献[3]所采取容错优先级的分配策略是一种容错优先级继承的策略,即任务的容错优先级继承任务的优先级( $\overline{p}=p$ ).因而,在这种分配策略下,任务 $\tau_i$ 的最坏响应时间只与 $T_E$ 有关,我们用 $R_i(T_E)$来表示,其计算公式为

$$R_i(T_E)=C_i+\sum_{\tau_j\in hp(i)}\left\lceil\frac{R_i(T_E)}{T_j}\right\rceil C_j+\left\lceil\frac{R_i(T_E)}{T_E}\right\rceil\max_{\tau_k\in hpe(i)}\overline{C}_k \tag{2}$$

其中,$hp(i)$表示 $p_j>p_i$ 的任务集合,即 $hp(i)=\{\tau_j\in\Gamma|p_j>p_i\}$;而 $hp(i)$用于表示 $p_k\geq p_i$ 的任务集合,即 $hpe(i)=\{\tau_k\in\Gamma|p_k>p_i\}$.

当采取继承策略时,系统无须额外的容错调度处理机制,完全按照无容错需求时所采取的 $FP(\Gamma)$调度机制来处理容错,这样实现起来较为简单.但是,当系统考虑容错需求时,任务的响应时间肯定大于或等于无容错情况下的响应时间,若只是简单地按照无容错情况下的处理机制,可能会导致任务无法在截止期限内完成,结果整个系统不可调度,造成严重的损害.现在我们对表 1 中的容错实时任务集合采取容错优先级继承的分配策略.当 $T_E$=10 时,根据式(2)计算所得到的每个任务的最坏响应时间 $R_i(10)$均小于各自对应的截止期限,则在 $T_E$=10 时系统是可调度的,但是当 $T_E$ 缩小为 9 时,任务 $\tau_2$ 的最坏响应时间 $R_2(9)$大于其截止期限 $D_2$,因而导致整个系统变为不可调度的.这就说明了该系统在采取容错优先级继承的分配策略下,其最大的容错能力是 $T_E$=10.

**Table 1** The limitation of the fault-tolerant priority inheritance approach ( $\overline{p}=p$ )

表 **1** 容错优先级继承分配策略( $\overline{p}=p$ )的限制

| Task | $T_i$ | $C_i$ | $\overline{C}_i$ | $D_i$ | $p_i$ | $R_i(10)$ | $R_i(9)$ |
|---|---|---|---|---|---|---|---|
| $\tau_1$ | 12 | 4 | 4 | 12 | 3 | 8 | 8 |
| $\tau_2$ | 20 | 3 | 3 | 20 | 2 | 19 | 23 |
| $\tau_3$ | 35 | 1 | 1 | 35 | 1 | 20 | 35 |

从表 1 中任务响应的情况来看:当 $T_E$=9 时,任务 $\tau_1$ 还存在一些空闲时间,但由于采取的是容错优先级继承的分配策略,以致无法利用这些空闲时间来满足 $\tau_2$ 截止期限的要求,而导致整个系统不可调度.为了能够挪用高优先级任务的空闲时间,文献[7]则基于容错优先级分配因子的概念 $P_x$ 提出了允许容错优先级提高的分配策略( $\overline{p}\geq p$ ).其中 $P_x$ 的定义如下:

定义 **2**. 容错优先级分配因子 $P_x$ 是一个多元组$\langle h_{x,1},h_{x,2},\ldots,h_{x,n}\rangle$,其中 $h_{x,i}=\overline{p}_i-p_i$,且 $0\leq h_{x,i}<i$ .

根据 $P_x$ 的定义,$h_{x,i}$ 表示 $\overline{p}_i$ 相对于 $p_i$ 的增量,使得 $\overline{p}_i$ 的取值范围为 $p_i$ 与最高优先级之间.当 $P_x=\langle 0,0,\ldots,0\rangle$时,所表示的就是容错优先级继承的分配策略,换句话说,容错优先级继承的分配策略是允许容错优先级提高的分配策略的特例.当 $P_x$ 取不同的值时,任务的最坏响应时间是不同的,因而,任务的最坏响应时间又与 $P_x$ 的相关,用 $R_i(x,T_E)$表示.在允许容错优先级提高的分配策略下,$R_i(x,T_E)$的计算公式为

$$R_i(x,T_E)=\max(R_i^{ext}(x,T_E),R_i^{\text{int}}(x,T_E)) \tag{3}$$

其中, $R_i^{ext}(x,T_E)$ 表示的是任务 $\tau_i$ 在执行过程中自身未出现故障时的最坏响应时间,简称为外部故障最坏响应时间; $R_i^{\text{int}}(x,T_E)$ 则表示 $\tau_i$ 在执行过程中自身出现故障时的最坏响应时间,简称为内部故障最坏响应时间.任务 $\tau_i$ 在这种策略下的最坏响应时间是这两者中的最大值.由于 $R_i^{ext}(x,T_E)$ 和 $R_i^{\text{int}}(x,T_E)$ 的计算公式较为复杂,这里不再列出,详见文献[7].

**Table 2** The limitation of the fault-tolerant priority promotion approach ( $\overline{p}\geq p$ )

表 **2** 允许容错优先级提高的分配策略( $\overline{p}\geq p$ )的限制

| Task | $D_i$ | $T_E$=9 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $\langle 0,0,0\rangle$ | | $\langle 0,1,0\rangle$ | | $\langle 0,1,1\rangle$ | | $\langle 0,1,2\rangle$ | |
| | | $R_i^{\text{int}}$ | $R_i^{ext}$ | $R_i^{\text{int}}$ | $R_i^{ext}$ | $R_i^{\text{int}}$ | $R_i^{ext}$ | $R_i^{\text{int}}$ | $R_i^{ext}$ |
| $\tau_1$ | 12 | 8 | 4 | 8 | 7 | 8 | 7 | 8 | 7 |
| $\tau_2$ | 20 | 18 | 23 | 14 | 23 | 14 | 23 | 14 | 23 |
| $\tau_3$ | 35 | 9 | 35 | 9 | 35 | 9 | 35 | 9 | 35 |

允许容错优先级提高的分配策略在一定程度上提高了系统的容错能力,特别是当最低优先级任务在容错优先级继承分配策略下不能满足截止期限的要求时,允许其容错优先级提高可以解决它的可调度性.从这个意义上说,允许容错优先级提高的分配策略优于继承分配策略.但是,在特定情况下,允许容错优先级提高的分配策略却无法提高系统的容错能力.下面举例说明.

现对表 1 中的任务采取允许容错优先级提高的分配策略,所得到的任务最坏响应时间见表 2.从 $\tau_2$ 的响应时

间来看,当 $T_E$=9 时,无论 $P_x$ 如何配置, $\overline{p}_1$ 始终大于 $p_2$,而且 $\overline{C}_1$ 是在容错优先级不低于 $p_2$ 的任务中替代任务执行时间的最大值,这就导致了 $R_2^{ext}(x,9)$ 没有缩短.从系统的容错能力来看,容错能力仍为 $T_E$=10,与容错优先级继承分配策略相比并没有得到提高.

从上面的分析可以得出,在系统采取容错优先级继承的分配策略时,当碰到下面的情况时,采取允许容错优先级提高的分配策略是无法提高系统容错能力的:最低优先级任务可调度,而处于中间优先级任务不可调度,并且对于不可调度的任务 $\tau_i$ 来说,在优先级高于 $p_i$ 的任务中,至少存在一个这样的任务,其替代任务的执行时间比优先级低于 $p_i$ 的所有任务的替代任务执行时间要长.系统容错能力无法提高的主要原因是由于高优先级任务的容错优先级高于不可调度任务的优先级造成的,要提高这种情况下系统的容错能力,可以从降低高优先级任务的容错优先级的角度来考虑.后面我们将从允许容错优先级降低的分配策略( $\overline{p} \le p$ )来分析如何提高系统的容错能力,这种分配策略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限的要求,提高系统的容错能力.

## 3 可调度性分析

首先分析允许容错优先级降低对任务响应时间的影响,然后根据这种影响得出任务最坏响应时间的计算公式,最后根据计算公式举实例说明允许容错优先级降低的分配策略可以在容错优先级继承和提高两种分配策略无法提高系统容错能力的情况下能够有效地提高系统容错能力.

由于我们所讨论的是允许容错优先级降低的分配策略,则容错优先级分配因子 $P_x$ 的概念需要重新定义,定义如下:

定义 **3**. 容错优先级分配因子 $P_x$ 是一个多元组$\langle h_{x,1},h_{x,2},\ldots,h_{x,n}\rangle$,其中 $h_{x,i}=\overline{p}_i-p_i$,且$-(n+1-i)<h_{x,i}\le 0$.

根据 $P_x$ 的定义,$h_{x,j}$ 表示 $\overline{p}_i$ 相对于 $p_i$ 的减量,使得 $\overline{p}_i$ 的取值范围为最低优先级与 $p_i$ 之间.当 $P_x=\langle 0,0,\ldots,0\rangle$时,所表示的就是容错优先级继承的分配策略.

### 3.1 降低容错优先级

当系统允许降低容错优先级时,与容错优先级继承的分配策略相比,任务的响应时间肯定会受到影响.举例说明,任务集合$\Gamma=\{\tau_i,\tau_j\}$,其中任务之间的优先级和容错优先级的关系为 $\overline{p}_j<\overline{p}_i<p_j<p_i$.如图 1 所示,假设在$\tau_i$完成的瞬间出现故障,由于 $\overline{p}_i<p_j$,则 $\tau_j$ 在 $\overline{\tau}_i$ 之前执行.所以,相对于容错优先级继承分配策略在同样执行情况下,$\tau_i$的响应时间延长了 $C_j$,而 $\tau_j$的响应时间相应地减少了 $\overline{C}_i$.因此,这样的影响不得不加入到响应时间的分析中.而当 $\tau_j$ 在完成的瞬间出现故障,则如图 2 所示,$\tau_j$ 不仅遭受了自身故障的影响,还受到 $\tau_i$ 的第 2 次激活的影响.

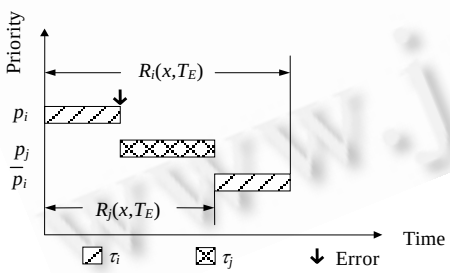


Fig.1 The worst-case response time of $\tau_i$

图 1 任务 $\tau_i$ 的最坏响应过程

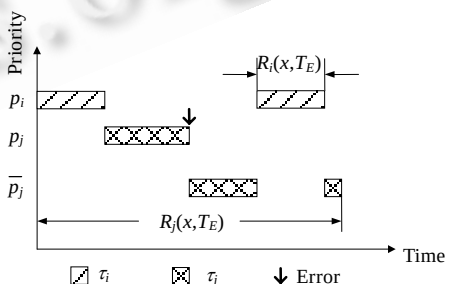


Fig.2 The worst-case response time of $\tau_j$

图 2 任务 $\tau_j$ 的最坏响应过程

为概括上述分析的影响,对于任务集合$\Gamma=\{\tau_1,\tau_2,\ldots,\tau_n\}$和给定的$\tau_i\in\Gamma$,定义以下与 $\tau_i$ 相关的$\Gamma$的任务子集合:

$ip(x,i)$用于表示容错优先级不低于 $p_i$ 的任务子集合.这个集合中的任务若在 $\tau_i$ 响应过程中出现故障,则其对应的替代任务会影响 $\tau_i$ 的响应时间,表示形式为 $ip(x,i)=\{\tau_j\in\Gamma\mid\overline{p}_j\ge p_i\}$.

$sp(x,i)$用于表示优先级大于 $\overline{p}_i$ 的任务子集合.这个集合中的任务会抢占 $\overline{\tau}_i$ 的执行,表示形式为 $sp(x,i)=\{\tau_j\in\Gamma\mid p_j>\overline{p}_i\}$.

$he\overline{pp}(x,i)$ 用于表示容错优先级不低于 $\overline{p}_i$ 的任务子集合.这个集合中的任务若在 $\overline{\tau}_i$ 执行过程中出现故障,则其对应的替代任务会影响 $\overline{\tau}_i$ 的响应时间,表示形式为 $he\overline{pp}(x,i)=\{\tau_j\in\Gamma\mid\overline{p}_j\geq\overline{p}_i\}$.

### 3.2 任务响应时间的计算

对于任务 $\tau_i$ 的响应时间,我们采取与允许容错优先级提高的分配策略的可调度性分析类似的方法,也分为两种情况来分析:内部故障响应时间 $R_i^{\text{int}}(x,T_E)$ 和外部故障响应时间 $R_i^{ext}(x,T_E)$.所以,$\tau_i$ 在允许容错优先级降低的分配策略下的最坏响应时间 $R_i(x,T_E)$ 就等于 $R_i^{\text{int}}(x,T_E)$ 和 $R_i^{ext}(x,T_E)$ 中的最大值.下面我们分别对 $R_i^{ext}(x,T_E)$ 和 $R_i^{\text{int}}(x,T_E)$ 进行分析.

(1) $R_i^{ext}(x,T_E)$

在这种情况下,$\tau_i$ 没有出现故障,则我们主要针对优先级 $p_i$ 进行讨论.就 $I_i$ 部分,能够抢占 $\tau_i$ 执行的任务,其优先级应该大于 $p_i$.由于是最长的抢占时间,因此考虑的是整个 $hp(i)$.而对于 $F_i$,$hp(i)$中的任务应该分为两个子集合:若 $\tau_j\in ip(x,i)-\{\tau_i\}$,则不但 $\tau_j$ 会抢占 $\tau_i$,而且 $\overline{\tau}_j$ 也会中断 $\tau_i$;而若 $\tau_j\in hp(i)-(ip(x,i)-\{\tau_i\})$,则在 $\tau_i$ 响应过程中,$\tau_j$ 虽然会中断 $\tau_i$,但由于 $\overline{p}_j<p_i$,则 $\overline{\tau}_j$ 不会影响 $\tau_i$ 的执行.所以,$R_i^{ext}(x,T_E)$ 中的 $I_i$ 包括了 $hp(i)$中的所有任务,而 $F_i$ 仅包括 $ip(x,i)-\{\tau_i\}$中的任务,故有 $R_i^{ext}(x,T_E)$ 的计算公式为

$$R_i^{ext}(x,T_E)=C_i+\sum_{\tau_j\in hp(i)}\left\lceil\frac{R_i^{ext}(x,T_E)}{T_j}\right\rceil C_j+\left\lceil\frac{R_i^{ext}(x,T_E)}{T_E}\right\rceil\max_{\tau_k\in ip(x,i)-\{\tau_i\}}\overline{C}_k \tag{4}$$

(2) $R_i^{\text{int}}(x,T_E)$

在这种情况下,相对于 $R_i^{ext}(x,T_E)$,情况更为复杂.可以将 $\tau_i$ 的响应过程分为两个阶段:故障出现前和故障出现后.主要基于以下几点考虑:首先,由于 $\overline{p}_i\neq p_i$,则故障出现前后,$\tau_i$ 与 $\overline{\tau}_i$ 的优先级是发生变化的;其次最坏响应时间的计算是一个迭代过程;最后,在执行过程中,无法预测 $\tau_i$ 出现故障的时间.为了使问题简单化,正如图 3 所示,我们用 $R_i^{\text{int}(0)}(x,T_E)$ 表示故障出现前的响应时间,而用 $R_i^{\text{int}(1)}(x,T_E)$ 表示故障出现后的响应时间,并且假设时刻 $t$ 是 $\tau_i$ 在响应过程中第 1 次出现故障的时刻,即 $R_i^{\text{int}(0)}(x,T_E)$ 和 $R_i^{\text{int}(1)}(x,T_E)$ 的分界点.由于任务在故障出现前的执行情况与故障出现后的执行无关,因而可以通过前面分析 $R_i^{ext}(x,T_E)$ 的方法先推导出 $R_i^{\text{int}(1)}(x,T_E)$,然后再根据 $R_i^{\text{int}(1)}(x,T_E)$ 推导出 $R_i^{\text{int}(0)}(x,T_E)$,从而推导出 $R_i^{\text{int}}(x,T_E)$ 的计算公式.下面将按照这种方法进行分析.

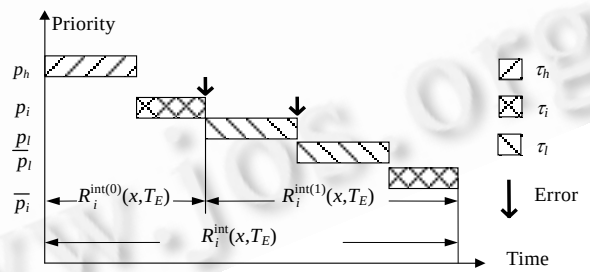


Fig.3 The worst-case response time of $\tau_i$ due to the internal errors

图 3 $\tau_i$ 在出现内部故障时的最坏响应过程

在 $R_i^{\text{int}(1)}(x,T_E)$ 期间,$\overline{\tau}_i$ 替代 $\tau_i$ 进行容错处理,则该阶段的响应时间主要是基于 $\overline{p}_i$ 的.因而,能够抢占 $\overline{\tau}_i$ 执行的任务优先级应大于 $\overline{p}_i$,则 $I_i$ 部分的时间是 $sp(x,i)-\{\tau_i\}$ 中任务的执行时间总和.而就 $F_i$ 部分,在这段时间内,故障出现的次数最多可达到 $\left\lceil\frac{R_i^{\text{int}(1)}(x,T_E)}{T_E}\right\rceil$:第 1 个就是 $t$ 时刻 $\tau_i$ 出现的第 1 次故障,而其他故障则来自于 $he\overline{pp}(x,i)$,这里包括 $\overline{\tau}_i$.因为在前面的任务模型中,我们提到,当 $\overline{\tau}_i$ 出现故障时,采取的容错技术是重复执行 $\overline{\tau}_i$,这就说明了在 $\overline{\tau}_i$ 的响应过程中,它的优先级是保持不变的,直到它执行完毕.故 $R_i^{\text{int}(1)}(x,T_E)$ 的计算公式为

$$R_i^{\text{int}(1)}(x,T_E)=\overline{C}_i+\sum_{\tau_j\in sp(x,i)-\{\tau_i\}}\left\lceil\frac{R_i^{\text{int}(1)}(x,T_E)}{T_j}\right\rceil C_j+\left(\left\lceil\frac{R_i^{\text{int}(1)}(x,T_E)}{T_E}\right\rceil-1\right)\max_{\tau_k\in he\overline{pp}(x,i)}\overline{C}_k \tag{5}$$

而在 $R_i^{\text{int}(0)}(x,T_E)$ 期间,情况就比 $R_i^{\text{int}(1)}(x,T_E)$ 要复杂.就 $I_i$ 而言,主要来自 $hp(i)$ 中的任务,然而由于 $\overline{p}_i \le p_i$,则 $hp(i)\subseteq sp(x,i)$,所以对于 $\tau_j \in hp(i)$,在 $R_i^{\text{int}}(x,T_E)$ 执行的总次数为 $\left\lceil \frac{R_i^{\text{int}}(x,T_E)}{T_j} \right\rceil$,而在 $R_i^{\text{int}(1)}(x,T_E)$ 已经计算了 $\left\lceil \frac{R_i^{\text{int}(1)}(x,T_E)}{T_j} \right\rceil$,因此在 $R_i^{\text{int}(0)}(x,T_E)$ 的执行次数为任务执行的总次数减去在 $R_i^{\text{int}(1)}(x,T_E)$ 中执行的次数.而对于 $F_i$ 部分,分两种情况讨论:

(1) $\overline{p}_i < p_i$:在这种情况下,容错操作可能发生在 $ip(x,i)$ 的任务上.因为若 $\tau_i$ 在该阶段出现故障,则其替代任务将执行在比 $p_i$ 低的优先级上,所考虑的抢占任务的集合就不是 $hp(i)$,而应该分为两部分:出现前为 $hp(i)$,出现后为 $sp(x,i)-\{\tau_i\}$.但是这与分析的前提是相矛盾的,我们假设的是 $t$ 时刻为优先级变换的时间点.因此,在这种情况下,$F_i$ 部分考虑的是 $ip(x,i)$,而不包括 $\overline{\tau}_i$.

(2) $\overline{p}_i = p_i$:可能执行容错的任务应该是包括 $\tau_i$ 的 $ip(x,i)$ 整体集合.由于 $\overline{p}_i = p_i$,则在 $R_i^{\text{int}}(x,T_E)$ 期间,$\tau_i$ 和 $\overline{\tau}_i$ 始终执行在同一优先级上,所以就有可能出现下面的情况:所有的故障都出现在 $\tau_i$ 和它的替代任务上,因此,在这种情况下,需要考虑的是 $ip(x,i)$.

基于以上两种情况的分析得出:对于 $F_i$ 部分,我们考虑的是 $ip(x,i)$.因而,$R_i^{\text{int}(0)}(x,T_E)$ 的计算公式为

$$R_i^{\text{int}(0)}(x,T_E)=C_i+\sum_{\tau_j\in hp(i)}\left(\left\lceil\frac{R_i^{\text{int}}(x,T_E)}{T_j}\right\rceil-\left\lceil\frac{R_i^{\text{int}(1)}(x,T_E)}{T_j}\right\rceil\right)C_j+\left(\left\lceil\frac{R_i^{\text{int}}(x,T_E)}{T_E}\right\rceil-\left\lceil\frac{R_i^{\text{int}(1)}(x,T_E)}{T_E}\right\rceil\right)\max_{\tau_k\in ip(x,i)}\overline{C}_k \tag{6}$$

通过以上对 $R_i^{\text{int}(0)}(x,T_E)$ 和 $R_i^{\text{int}(1)}(x,T_E)$ 的分析,我们将式(5)和式(6)相加得到 $R_i^{\text{int}}(x,T_E)$:

$$R_i^{\text{int}}(x,T_E)=R_i^{\text{int}(0)}(x,T_E)+R_i^{\text{int}(1)}(x,T_E) \tag{7}$$

$R_i^{\text{int}}(x,T_E)$ 的计算过程是:首先通过对等式(5)采取迭代方法计算出 $R_i^{\text{int}(1)}(x,T_E)$,然后将 $R_i^{\text{int}(1)}(x,T_E)$ 的值代入等式(6)算出 $R_i^{\text{int}(0)}(x,T_E)$,从而根据式(7)得到 $R_i^{\text{int}}(x,T_E)$.

从上面的分析可知,容错优先级继承的分配策略其实是允许容错优先级降低的分配策略的特例,在所有任务容错优先级未发生变化的情况下,两种策略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 $\Gamma=\{\tau_1,\tau_2,\tau_3,\ldots,\tau_{n-1},\tau_n\}$,对于任意的故障出现最小间隔 $T_E$ 和 $\tau_i\in\Gamma$,在 $P_x=\langle 0,0,\ldots,0\rangle$时,由式(2)计算所得到的最坏响应时间等于在允许容错优先级降低策略下计算所得的 $R_i^{\text{int}}(x,T_E)$ 和 $R_i^{ext}(x,T_E)$ 的最大值.

证明:当 $P_x=\langle 0,0,\ldots,0\rangle$时,$ip(x,i)-\{\tau_i\}=sp(x,i)-\{\tau_i\}=hp(i)$, $he\overline{pp}(x,i)=ip(x,i)=hpe(i)$,则由式(4)和式(7)可得:

$$R_i^{ext}(x,T_E)=C_i+\sum_{\tau_j\in hp(i)}\left\lceil\frac{R_i^{ext}(x,T_E)}{T_j}\right\rceil C_j+\left\lceil\frac{R_i^{ext}(x,T_E)}{T_E}\right\rceil\max_{\tau_k\in hp(i)}\overline{C}_k,$$

$$R_i^{\text{int}}(x,T_E)=C_i+\sum_{\tau_j\in hp(i)}\left\lceil\frac{R_i^{\text{int}}(x,T_E)}{T_j}\right\rceil C_j+\overline{C}_i+\left(\left\lceil\frac{R_i^{\text{int}}(x,T_E)}{T_E}\right\rceil-1\right)\max_{\tau_k\in hpe(i)}\overline{C}_k.$$

① 当 $\overline{C}_i=\max_{\tau_k\in hpe(i)}\overline{C}_k$ 时,有 $R_i^{ext}(x,T_E)<R_i^{\text{int}}(x,T_E)$;

② 当 $\overline{C}_i\ne\max_{\tau_k\in hpe(i)}\overline{C}_k$ 时,有 $R_i^{ext}(x,T_E)>R_i^{\text{int}}(x,T_E)$;

综合①②,$R_i^{\text{int}}(x,T_E)$ 和 $R_i^{ext}(x,T_E)$ 的最大值等于式(2)计算所得的最坏响应时间. □

### 3.3 实例分析

现对表 1 中的任务集合采取允许容错优先级降低的分配策略,允许任务 $\tau_1$ 的替代任务在更低的优先级上执行.在 3 种容错优先级分配策略下,任务的最坏响应时间见表 3.当 $T_E$=9 时,在允许容错优先级降低的分配策略($P_x=\langle -2,0,0\rangle$)下,由于 $\overline{p}_1=1$,使得 $\tau_1$ 的替代任务不能在 $\tau_2$ 的执行过程响应,从而缩短了 $\tau_2$ 的响应时间,并且 $\tau_2$ 能够在其截止期限内完成,同时,$\tau_1,\tau_2$ 也能满足截止期限的要求,则整个系统是可调度的.这主要是由于挪用了 $\tau_1$ 的空闲时间来执行 $\tau_2$.

**Table 3** The worst-case response time of 3 fault-tolerant priority assignment policies

表 3 3 种容错优先级分配策略的比较

| Task | $D_i$ | $\overline{p}=p$ $T_E$=9 ⟨0,0,0⟩ $R_i^{\text{int}} \mid R_i^{ext}$ | $\overline{p}>p$ $T_E$=9 ⟨0,1,0⟩ $R_i^{\text{int}} \mid R_i^{ext}$ | $\overline{p}\le p$ | |
|---|---|---|---|---|---|
| | | | | $T_E$=9 ⟨−2,0,0⟩ $R_i^{\text{int}} \mid R_i^{ext}$ | $T_E$=8 ⟨−2,0,0⟩ $R_i^{\text{int}} \mid R_i^{ext}$ |
| $\tau_1$ | 12 | 8\|4 | 8\|7 | 11\|4 | 11\|4 |
| $\tau_2$ | 20 | 18\|23 | 14\|23 | 13\|7 | 13\|7 |
| $\tau_3$ | 35 | 9\|35 | 9\|35 | 9\|35 | 13\|39 |

从 3 种容错优先级分配策略对系统容错能力的影响来看,在容错优先级继承和允许容错优先级提高的两种分配策略下,系统的容错能力均为 $T_E$=10;而当采取允许容错优先级降低的分配策略时,容错能力可以达到 $T_E$=9.这说明,在前两种策略下不具备可调度性的系统,当采取允许容错优先级降低的分配策略后,系统的容错能力在某些情况下可以得到提高.

## 4 最佳容错分配因子搜索算法

前面我们对允许容错优先级降低的分配策略下的任务可调度性进行了分析,下一步的主要任务就是设计最佳容错分配因子的搜索算法,使得 $T_E$ 尽量小,系统的容错能力更强.根据允许容错优先级降低的分配策略的性质,我们设计了一种改进的搜索算法.

首先引入函数 $T_e(x)$[7],表示在给定的 $P_x$,$T_E$ 所能达到的最小值,也就是说,若 $T_E$ 比 $T_e(x)$的值更小,则一些任务可能不可调度.特别是,当 $T_E=T_e(x)-1$ 时,至少存在这样一个任务 $\tau_i$,满足 $R_i(x,T_e(x)-1)>D_i$.我们称这样的任务为临界任务,用 $D(x)$来表示临界任务集合,即 $D(x)=\{\tau_i\in\Gamma \mid R_i(x,T_e(x)-1)>D_i\}$.

根据式(4)和式(7),分析允许容错优先级降低的分配策略对最高优先级任务、最低优先级任务和处于中间优先级任务 3 种不同优先级任务响应时间的影响,这里,我们假设 $\tau_1$ 的优先级最高,$\tau_n$ 的优先级最低,$\tau_i$ 表示任意一个中间优先级任务.

1. 最高优先级任务 $\tau_1$

由于 $hp(1)=\varnothing$,则 $R_1^{ext}(x,T_E)$ 与继承分配策略所计算出的外部故障时间相等($C_1$);而 $R_1^{\text{int}}(x,T_E)$ 肯定是大于或等于 $C_1+\overline{C}_1$,与继承分配策略相比是延长了 $\tau_1$ 的内部故障响应时间.因此,采取允许容错优先级降低的分配策略只会延长 $\tau_1$ 的响应时间.

2. 最低优先级任务 $\tau_n$

对于 $\tau_n$ 来说,$\overline{p}_n$ 只可能等于 $p_n$.根据式(4)和式(7)可知,在 $T_E$ 给定时,$R_n(x,T_E)$ 是一个定值,与 $P_x$ 无关.因此,采取允许容错优先级降低的分配策略不会影响 $\tau_n$ 的响应时间.

3. 中间优先级任务 $\tau_i$

相对于 $\tau_1$,$\tau_n$ 来说,$\tau_i$ 的情况要复杂得多.首先分析 $R_i^{ext}(x,T_E)$,由于允许容错优先级降低,则

$$ip(x,i)-\{\tau_i\}\subseteq hpe(i),\ \max_{\tau_k\in ip(x,i)-\{\tau_i\}}\overline{C}_k\le\max_{hpe(i)}\overline{C}_k,$$

因此,$R_i^{ext}(x,T_E)$ 可能缩短.而当 $\Phi(x,i)=\left\{\tau_k\in\Gamma \mid \overline{C}_k=\max_{\tau_j\in he\overline{pp}(x,i)-\{\tau_i\}}\overline{C}_i\right\}$ 不为空时,$R_i^{\text{int}}(x,T_E)$ 也有可能缩短.所以,采取允许容错优先级降低的分配策略会缩短 $\tau_i$ 的响应时间.

根据以上分析,我们可以得出允许容错优先级降低分配策略的性质:在 $P_x$ 给定的情况下,如果最高优先级任务或最低优先级任务不可调度,则降低容错优先级是不能缩短它们的响应时间的;而当中间优先级任务不可调度时,则可以通过降低高优先级任务的容错优先级可能会缩短其响应时间.所以在设计最佳容错分配因子的搜索算法时,我们主要针对的是处于中间优先级任务不可调度的情况.

假设 $\tau_i$ 不可调度,分两种情况讨论:若 $R_i^{ext}(x,T_E)>D_i$,则根据式(4),需要降低容错优先级的任务 $\tau_j$ 应该满足 $\overline{C}_j=\max_{\tau_k\in ip(x,i)-\{\tau_i\}}\overline{C}_k$;而若 $R_i^{\text{int}}(x,T_E)>D_i$,则根据等式(7),需要考虑降低容错优先级的任务应该属于 $\{\tau_j\in\Gamma \mid \overline{C}_j=\max_{\tau_k\in he\overline{pp}(x,i)-\{\tau_i\}}\overline{C}_k\}$.综合以上的分析,我们用 $reductionset(x,i)$ 来表示这两种情况下需要降低容错

优先级任务的集合.而且 $reductionset(x,i)$ 中任务的容错优先级只要降低到 $p_i$ 以下就可以缩短 $\tau_i$ 的响应时间,最简单的就是等于 $p_i-1$.

正如第 2 节的分析结果所描述的,允许容错优先级降低的分配策略是针对在容错优先级继承策略和提高策略均无法提高系统容错能力的情况下而提出的.换句话说,当某些任务集合在容错优先级继承和提高两种策略下均不可调度时,可以采取允许容错优先级降低的策略,使任务集合可调度,从而提高整个系统的容错能力.正如文献[7]所阐述的,允许容错优先级提高策略是为了解决任务集合在继承策略下不可调度的情况,而本文所提出的允许容错优先级降低的分配策略则是为了解决任务集合在容错优先级提高的分配策略不能提高系统在继承策略下的容错能力的情况,因此,我们综合这 3 种容错优先级的分配策略,在文献[7]所提出的 PCS 算法中加入允许容错优先级降低的策略,设计出一种改进的最佳容错分配因子搜索算法 IFPCS.

算法 **1**. Improved fault-tolerant priority configuration search algorithm.

(1) 根据 $FP(\Gamma)$分配 $p_i,i=1,2,\ldots,n$
(2) $\tau_{\max} \leftarrow$ 最高优先级任务
(3) $\tau_{\min} \leftarrow$ 最低优先级任务
(4) $P_x \leftarrow (0,0,...,0); P_x^* \leftarrow P_x$
(5) $L \leftarrow 1+\max_{\tau_i \in \Gamma} C_i$
(6) $T_E \leftarrow T_e(x); T_E^* \leftarrow T_E$
(7) PCS( $T_E$ , $P_x$ , $\Gamma$ )
(8) If ( $T_E == T_E^*$ )
(9) while(TRUE)
(10) 计算 $R_i(x,T_E)$ ,$i=1,2,\ldots,n$
(11) if ( $\forall \tau_i \in \Gamma$ , $R_i(x,T_E) \le D_i$ ) then
(12) $P_x^* \leftarrow P_x$
(13) $T_E^* \leftarrow T_E$
(14) $T_E \leftarrow T_E - 1$
(15) if ( $T_E < L$ ) exit while
(16) else
(17) if ( $\tau_{\min} \in D(x)$ ) exit while
(18) if ( $\tau_{\max} \in D(x)$ ) exit while
(19) $\tau_i$ 为 $D(x)$ 中的一个任务
(20) if ($reductionset(x,i)=\varnothing$) exit while
(21) $\tau_j$ 为 $reductionset(x,i)$ 中的一个任务, $h(x,j) \leftarrow \overline{p}_i - 1 - p_j$
(22) if ( $|reductionset(x,i)|=1$ ) $T_E \leftarrow MIN(T_e(x),T_E)$
(23) endif
(24) endwhile
(25) endif
(26) $P_x \leftarrow P_x^*$
(27) $T_E \leftarrow T_E^*$

改进的容错分配因子搜索算法可分为 3 步:首先计算出给定的任务集合 $\Gamma$ 在容错优先级继承策略下的 $T_E$ 和 $P_x$ 的值(见算法第 1 行~第 6 行),然后再计算出 $\Gamma$ 在容错优先级提高策略下的 $T_E^*$ 和 $P_x^*$ 的值(见算法第 7 行),接着对 $T_E$ 和 $T_E^*$ 进行比较,此时有两种可能: $T_E^*$ 小于 $T_E$ 和 $T_E^*$ 等于 $T_E$.根据文献[7]所给出的证明, $T_E^*$ 是不可能大于 $T_E$ 的.若 $T_E^*$ 小于 $T_E$,则说明允许容错优先级提高的分配策略能够提高系统的容错能力,这种情况下,所得到的 $P_x^*$ 即为该改进算法所得到的最佳容错分配因子;而当 $T_E^*$ 等于 $T_E$,则表示容错优先级提高策略无法提高 $\Gamma$ 在继承策略下的系统容错能力,此时我们就对 $\Gamma$ 采取本文所提出的允许容错优先级降低的策略(见算法第 9 行~第 24 行).综观该改进算法可知,该算法在提高系统容错能力方面优于文献[7]所提出的最佳容错分配因子搜索算法.

## 5 模拟实验

为了说明不同容错优先级分配策略在提高系统容错能力方面的性能,我们对 $T_e(0)$和 $T_e(x)$进行比较,其中 $P_x$ 是在不同策略下的最佳容错分配因子.因此,我们将$\Delta T_e=(T_e(0)-T_e(x))/T_e(0)$作为衡量系统容错能力好坏的一个性能指标.很显然,在容错优先级继承策略下,$\Delta T_e=0$.这里采用 DMS 调度算法来分配任务的优先级 $p_i$.

我们模拟了 10 000 组任务,每组有 5 个任务,每个任务的执行时间为 $C_i$,周期为 $T_i$,截至期限 $D_i$ 和替代任务的最大执行时间 $\overline{C_i}$ 是随机产生的.但是对于任务组$\Gamma_m(m=1,2,\ldots,10000)$需满足以下条件:

(1) $\forall \tau_i\in\Gamma_m, T_i\in[5*\lceil m/10\rceil, 50*\lceil m/10\rceil]$;

(2) $\forall \tau_i\in\Gamma_m, 0<\overline{C_i}\le C_i, 2C_i\le D_i\le T_i$,其中$I$表示任务组;

(3) $\forall \tau_i\in D(0), p_{\min}<p_i<p_{\max}$,其中 $p_{\min}$ 表示最低的优先级,$p_{\max}$ 表示最高的优先级;

(4) $\forall \tau_i\in D(0), \exists \tau_j\in hp(i), C_j>C_i$.

对每组任务分别采用 PCS 算法和 IFPCS 算法搜索最佳容错分配因子,图 4 和图 5 分别为这两种算法 $T_E$ 和 $\Delta T_e$ 的曲线图,这里我们仅给出了前 100 组任务的数据.基于 PCS 算法的 $T_E$ 和$\Delta T_e$ 分别记为$(T_E)_{PCS}$和$(\Delta T_e)_{PCS}$,而基于 IFPCS 的 $T_E$ 和$\Delta T_e$ 则分别记为$(T_E)_{IFPCS}$和$(\Delta T_e)_{IFPCS}$.

由图 4 可知,与 PCS 算法相比,通过 IFPCS 算法搜索,除了个别几组任务($(T_E)_{PCS}=(T_E)_{IFPCS}$)以外,大多数任务组的 $T_E$ 都得到降低.在这 100 组任务中,得到降低的比例达到了 98%,持平的比例仅为 2%,没有 $T_E$ 升高的现象出现.而又如图 5 所示,在这 100 组任务中,$(\Delta T_e)_{PCS}$ 均为 0,$(\Delta T_e)_{IFPCS}$ 平均在 0.106 9±.这说明当 PCS 算法无法提高系统容错能力时,即$(\Delta T_e)_{PCS}=0$,通过 IFPCS 算法,$T_E$ 的降低率平均为 10.7%.

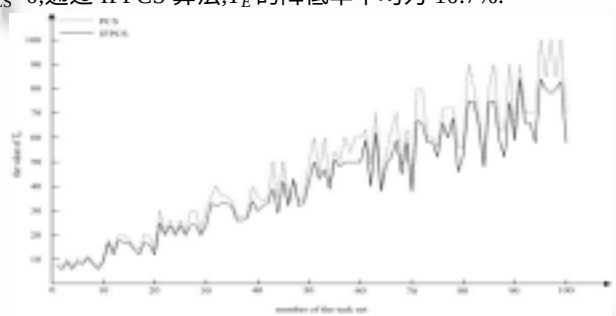

Fig.4 A curve of $T_E$

图 4 $T_E$ 的曲线

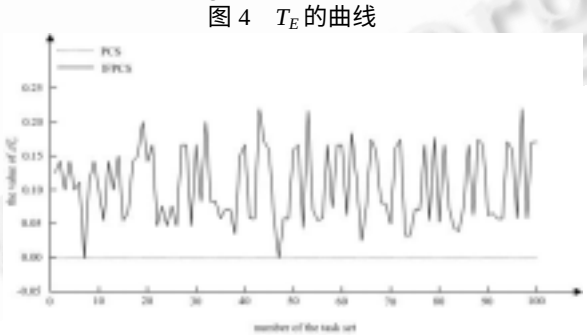

Fig.5 A curve of $\Delta T_e$

图 5 $\Delta T_e$ 的曲线

## 6 结束语

本文的主要贡献是针对容错优先级继承和允许容错优先级提高的两种分配策略在提高容错实时系统的容错能力所存在的缺陷,通过对容错实时系统进行基于最坏响应时间的可调度性分析,提出了一种允许容错优先级降低的分配策略;并且根据这种分配策略的性质,设计了改进的最佳容错优先级分配因子的搜索算法.经过研究分析和实验证明,在继承和提高两种容错优先级分配策略无法提高系统的容错能力的情况下,合理地降低任

务的容错优先级,能够有效地提高容错实时系统的容错能力.

**References**:

[1] Ghosh S, Melhem R, Mosse D. Enhancing real-time scheduled to tolerate transient faults. In: Proc. of the 16th IEEE Real-Time Systems Symp. Pisa: IEEE Computer Society Press, 1995. 120−129.

[2] Kandasmy N, Hayes JP, Murray BT. Tolerating transient faults in statically scheduled safety-critical embedded systems. In: Proc. of the 18th IEEE Symp. on Reliable Distributed Systems. Lausanne: IEEE Computer Society Press, 1999. 212−221.

[3] Burns A, Davis R, Punnekkat S. Feasibility analysis of fault-tolerant real-time task sets. In: Proc. of the 8th Euromicro Workshop on Real-Time Systems. L'Aquila: IEEE Computer Society Press, 1996. 29−33.

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

[5] Burns A, Punnekkat S, Strigini L, Wright DR. Probabilistic scheduling guarantees for fault-tolerant real-time systems. In: Proc. of the 7th Int'l Working Conf. on Dependable Computing for Critical Application. IEEE Computer Society Press, 1999. 339−356.

[6] De A Lima GM, Burns A. An effective schedulability analysis for fault-tolerant hard real-time systems. In: Proc. of the 13th Euromicro Conf. on Real-Time Systems. Delft: IEEE Computer Society Press, 2001. 209−216.

[7] De A Lima GM, Burns A. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. IEEE Trans. on Computers, 2003,52(10):1332−1346.

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

[9] Audsley NC, Burns A, Richardson M, Wellings AJ. Hard real-time scheduling: the deadline monotonic approach. In: Proc. of the 8th IEEE Workshop on Real-Time Operating Systems and Software. Atlanta: IEEE Computer Society Press, 1991. 133−137.