软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(12):3158−3171 [doi: 10.13328/j.cnki.jos.004917]

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

副版本不可抢占的全局容错调度算法*

彭 浩, 陆 阳, 孙 峰, 韩江洪

(合肥工业大学 计算机与信息学院,安徽 合肥 230009)

通讯作者: 韩江洪, E-mail: hanjh@hfut.edu.cn, http://www.hfut.edu.cn

摘 要: 容错是硬实时系统的关键能力,容错调度算法可以在有错误发生的情况下满足任务的实时性需求.在主副版本机制的容错调度算法中,主版本出错后留给副版本运行的时间窗口小,副版本容易错失截止期.针对副版本需要快速响应的问题,提出副版本不可抢占的全局容错调度算法 FTGS-NPB(fault-tolerant global scheduling with non-preemptive backups),赋予副版本全局最高优先级,使副版本在主版本出错后可以立刻获得处理器资源,并且在运行过程中不会被其他任务抢占.这样,副版本可以在最短时间内响应.分别基于截止期分析和响应时间分析建立了 FTGS-NPB 的可调度性测试,并分析了两种可调度性测试分别适用于不同的优先级分配算法.仿真实验结果表明,FTGS-NPB 可以有效地减少实现容错的代价.

关键词: 多处理器;硬实时系统;主副版本;容错调度;全局调度

中图法分类号: TP316

中文引用格式: 彭浩,陆阳,孙峰,韩江洪.副版本不可抢占的全局容错调度算法.软件学报,2016,27(12):3158−3171. http://www.jos.org.cn/1000-9825/4917.htm

英文引用格式: Peng H, Lu Y, Sun F, Han JH. Fault tolerant global scheduling with non-preemptive backups. Ruan Jian Xue Bao/Journal of Software, 2016,27(12):3158−3171 (in Chinese). http://www.jos.org.cn/1000-9825/4917.htm

Fault Tolerant Global Scheduling with Non-Preemptive Backups

PENG Hao, LU Yang, SUN Feng, HAN Jiang-Hong

(School of Computer and Information, Hefei University of Technology, Hefei 230009, China)

Abstract: Fault tolerance is a critical capability of hard real-time systems. Even with faults, fault tolerant scheduling algorithms are able to guarantee the real time property of tasks. In primary-backup based fault tolerant scheduling algorithms, only a small time window is left for the backup when the primary faults occur, therefore the backup will likely miss its deadline. This paper proposes a fault tolerant global scheduling with non-preemptive backups (FTGS-NPB). By assigning the highest priority to all backups, the backup can attain processor immediately in case of primary faults, and keep executing until finishing its job. In this way the backup can achieve the shortest response time. The schedulability tests are set up based on deadline analysis and response time analysis. The compatibility of priority assignment algorithms and schedulability tests is discussed. The simulation results show that FTGS-NPB can reduce the amount of additional processors for achieving fault tolerant capability.

Key words: multiprocessor; hard real-time system; primary-backup; fault-tolerant scheduling; global scheduling

面对不断增长的性能需求,多处理器芯片被越来越多的硬实时嵌入式系统采用,例如汽车电子[1−3]和航空电子[4]系统,在提高了硬件平台计算能力的同时给系统设计带来了新的问题.硬实时系统中的任务都有严格的实时性要求,即截止期约束,任务的每一次运行都需要在截止期之前正确响应,否则就视为系统失效.然而,由于

* 基金项目: 国家自然科学基金(61370088); 国家国际科技合作专项(2014DFB10060)

Foundation item: National Natural Science Foundation of China (61370088); International S&T Cooperation Program of China (2014DFB10060)

收稿时间: 2014-11-11; 修改时间: 2015-05-12, 2015-08-24, 2015-09-07; 采用时间: 2015-09-10

外部干扰或是设计缺陷,硬实时系统在运行过程中不可避免地会出现无法预知的错误.硬实时系统大多是安全关键系统,系统失效可能会带来严重的后果,因此,如何在有错误发生的情况下依然能够保证任务的实时性要求,是硬实时系统研究领域的一个关键问题.容错调度算法是实现这个目标的重要手段.

在容错调度算法中,主副版本机制是最为广泛采用的容错方法.在该方法中,每个任务有一个主版本以及一个或若干个副版本,在无错误时,调度的是任务的主版本;主版本发生错误时将被终止并调度一个副版本运行,如果副版本出错,则启动下一个副版本,直至有一个副版本正确响应.主副版本中有任意一个正确响应都视为任务正确运行[5].

任务的主版本和副版本释放作业的时间和截止期相同,主版本作业在释放后立刻进入就绪状态被调度运行,而副版本作业在主版本作业出错后才会被调度,可供副版本作业运行的时间窗口比主版本作业小很多,调度难度大.针对副版本需要快速响应的问题,本文提出副版本不可抢占的全局容错调度算法(fault-tolerant global scheduling with non-preemptive backups,简称 FTGS-NPB).FTGS-NPB 以全局固定优先级调度算法为基础,每个任务有唯一的优先级,主版本释放的作业继承该优先级,并按照固定优先级抢占调度的方式调度,而所有任务的副版本都有全局最高优先级,因此在任务出错后,副版本作业可以立刻获得处理器资源并一直占用处理器直到完成运行.这样,副版本的响应时间最短,不容易违反实时性约束.

1 相关研究

硬实时容错调度算法分为分组容错调度和全局容错调度.分组容错调度采用与分组调度相似的方法,将任务的主副版本分组,每个组分别运行在一个处理器上,采用单处理器实时调度算法分别调度每个处理器上的任务,同时,需要保证一个任务的主副版本分别被分配在不同的处理器上.在过去 10 多年中,分组容错调度算法得到大量研究[6−10].分组容错调度算法研究的问题在于:一旦某个任务的主版本发生错误,在同一个处理器上运行的其他任务的主版本也都被视为无法正常运行,并激活运行在其他处理器上的副版本;而故障处理器恢复或是备用处理器加入之后,需要通过一个很复杂的恢复过程,使系统恢复到故障之前的状态[11].在全局容错调度中,所有的处理器都被视为一致的,错误发生后,只需要激活出错主版本对应的副版本并将其加入就绪队列,该主版本错误周期之后释放的作业按照正常工作方式运行,其他未出错任务的运行不受影响,容错和恢复过程都非常简单.

与分组容错调度相比,全局容错调度的研究比较少.文献[12]采用错误概率模型建立了全局容错调度的可调度性测试,但是该测试是基于任务使用率的,算法复杂度低,但是准确性较差.文献[11]提出的全局容错调度算法 FTGS 采用副版本继承主版本优先级的方法,该方法实现简单,但是任务出错后副版本运行窗口短,又受到和主版本同样类型的干涉,很容易错失截止期.文献[13]采用基于混合遗传算法的在线动态调度算法调度开放系统中的实时任务集,该算法拒绝运行不能通过在线可调度性测试的作业,以保证被接受作业的实时性.这种方法不符合硬实时系统的要求,即,每个作业都要正确响应.

本文对错误的假设遵循一次错误模型,即,在一次作业的生命周期内只会出现一次错误.这一假设在容错调度领域被广泛采用[6−10,13].同时,本文假设发生瞬态故障的处理器能够立刻恢复功能,符合瞬态错误的特点[14,15].如果错误持续时间不能忽略,则需要通过增加备用处理器的方式来维持系统的容错能力.

2 系统模型

2.1 任务模型

硬实时系统由硬件平台和运行在其上的实时任务集组成,硬件平台包含 m 个同构处理器,实时任务集Γ包含 n 个实时任务.

任务集中的单个任务 $\tau_i(i=1,2,\ldots,n)$有一个主版本 $P_i=\langle T_i,D_i,C_i,PRIO_i\rangle$和一个副版本 $B_i=\langle T_i,D_i,E_i\rangle$.

- $PRIO_i$表示任务主版本的优先级,同时也表示该任务的优先级,假设 $PRIO_i$数值越小任务的优先级越高,

即,$PRIO_i$=1 的任务优先级最高,$PRIO_i=n$ 的任务优先级最低,每个任务的主版本有全局唯一的优先级,副版本都有最高优先级,因此在模型中省略;

- T_i表示相邻两次作业释放的最小时间间隔;
- D_i表示相对截止期;
- C_i和 E_i分别表示主、副版本的最坏情况运行时间.

在不需要特别说明作业的序数时,用 r_i表示主、副版本同时释放一次作业的时刻,用 d_i表示作业的绝对截止期,$d_i=r+D_i$.

2.2 调度模型

在 FTGS-NPB 中,调度器维护一个按优先级从高到低排序的全局就绪作业队列,在任意时刻,调度器总是选取就绪队列中优先级高的作业在所有功能正常的处理器上运行.任务的主、副版本在某一时刻(时间触发或事件触发)同时释放一次作业,主版本释放的作业立刻进入就绪状态并被放入就绪作业队列,副版本释放的作业被挂起,只有主版本作业出错副版本作业才会进入就绪状态,并被放入就绪作业队列.在系统运行过程中,抢占是始终允许的,如果就绪作业队列中有作业的优先级高于正在运行的作业,调度器会立即让高优先级作业抢占低优先级作业运行.在没有错误发生的情况下,主版本作业正确响应时立刻取消对应的副版本作业.主版本作业发生错误时将被立刻终止,并将对应的副版本作业由挂起状态转为就绪状态.由于副版本作业有最高优先级,该作业将立即抢占正在运行的最低优先级作业,并持续占用处理器运行直至正确响应.

2.3 调度算法

FTGS-NPB 算法包括在线和离线两部分:算法的在线部分是运行第 2.2 节所述调度过程的调度器;离线部分是算法的可调度性测试,将在后面的章节详述.可调度性测试是硬实时调度算法的重要组成部分,在系统设计阶段,必须通过可调度性测试证明系统中运行的实时任务集是可调度的,即,没有任何一次作业会错失截止期.本文分别基于 RTA-LC[16]和 DA-LC[17]可调度性测试建立 FTGS-NPB 的可调度性测试,分别称为 NPB-RTA 和 NPB-DA 可调度性测试.文献[17]通过实验说明:尽管 RTA-LC 测试准确性优于 DA-LC 测试,但结合优先级分配算法考虑,DA-LC 和 OPA 优先级分配算法的组合比 RTA-LC 和任意启发式优先级分配算法的组合能够调度的任务集比率更高(RTA-LC 测试和 OPA 算法不兼容),本文也将通过仿真实验进行类似对比.

2.4 相关定义

在后面各节的讨论中,需要用到下面的定义:

定义 1(有效负载). 在某个时间区间内,任务 τ_i(非被测试任务)得到运行的所有作业中,优先级高于被测试任务的所有作业的累积运行时间.

定义 2(干涉). 在某个时间区间内,任务 τ_i(非被测试任务)得到运行的作业占用处理器,导致被测试任务无法使用该处理器运行的时间.

定义 3(最大干涉总量). 在某个时间区间内,任务集中其他所有任务(除了被测试任务之外)产生的干涉之和的最大值.

定义 4(带入作业). 相对于某个时间区间,释放时刻在区间开始之前,截止期在区间开始时刻之后的作业.

3 可调度性测试

假设被测试任务是 τ_k,任务 τ_k的主版本 P_k和副版本 B_k在时刻 r_k同时分别释放一次作业 JP_k和 JB_k,JP_k和 JB_k在运行过程中受到高优先级作业的最大干涉,响应时间最迟.如果在 JP_k不出错时 JP_k可调度,并且在 JP_k出错时 JB_k可调度,则可以判定被测试任务 τ_k是可调度的.为方便讨论,假设 r_k为时间 0 点.

任务 τ_k每一次运行可能遇到的出错模式有 4 种:(1) 系统中没有错误发生(no-fault);(2) 任务 τ_k的主版本作业出错(self-fault);(3) 高优先级任务(相对于 τ_k)的主版本作业出错(high-fault);(4) 低优先级任务(相对于 τ_k)的主版本作业出错(low-fault).任务 τ_k在 4 种出错模式下都是可调度的,才可以被判定为可调度.

3.1 NPB-DA可调度性测试

首先建立任务$\tau_i(i\neq k)$在r_k时刻之后长度为L的时间窗口内(后面用简称L代表这个时间窗口)对被测试任务主版本作业JP_k产生干涉的最大有效负载和最大干涉的计算公式,副版本作业JB_k有全局最高优先级,因此其运行不受到干涉.根据τ_i和JP_k之间的优先级关系以及τ_i的主版本作业是否出错,τ_i的干涉模式有3种不同的类型,分别称为A,B和C类型.

(A) $PRIO_i<PRIO_k$且任务τ_i的主版本不出错.此时,只有τ_i主版本释放的作业产生干涉(副版本作业不会运行).τ_i在L内产生最大有效负载的运行模式如图1所示,这里区分有带入作业(CI)和没有带入作业(NC)两种情况是为了在计算最大干涉总量时采用限制带入作业(limited carry-in)技术[18],使用这一技术可以有效减小对JP_k就绪但无法获得处理器运行时长(由于所有处理器同时被高优先级作业占用)的过高估计,增加判定的准确性.

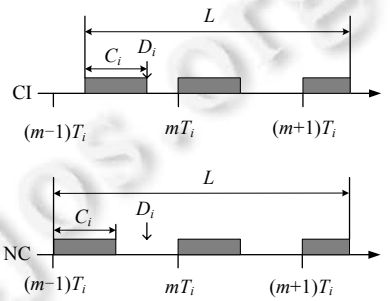

Fig.1 Upper bound of the effective load of a high priority task when no fault exists

图1 系统中无错误时高优先级任务的最大有效负载

τ_i无错误时在L内有带入作业的最大有效负载$WCI_i^{\mathrm{A}}(L)$和没有带入作业的最大有效负载$WNC_i^{\mathrm{A}}(L)$使用公式(1)和公式(2)计算:

$$WCI_i^{\mathrm{A}}(L)=N_iC_i+\min(C_i,L+D_i-C_i-N_iT_i) \tag{1}$$

其中,$N_i=\left\lfloor\dfrac{L+D_i-C_i}{T_i}\right\rfloor$,表示在可以在$L$内完整运行作业的最大数量.

$$WNC_i^{\mathrm{A}}(L)=\left\lfloor\frac{L}{T_i}\right\rfloor C_i+\min\left(C_i,L-\left\lfloor\frac{L}{T_i}\right\rfloor T_i\right) \tag{2}$$

τ_i产生的最大干涉为

$$ICI_i^{\mathrm{A}}(L)=\min(WCI_i^{\mathrm{A}}(L),L-C_k+1) \tag{3}$$

$$INC_i^{\mathrm{A}}(L)=\min(WNC_i^{\mathrm{A}}(L),L-C_k+1) \tag{4}$$

将τ_i的最大干涉限制为不大于$L-C_k+1$的原因在于:当τ_i在L内的最大有效负载达到$L-C_k+1$时,可以视为该任务占用了一个处理器,使得JP_k不能在这个处理器上被调度(调度该作业要求处理器至少有$L-C_k$的空闲时间).而继续增加最大干涉值,会导致大于$L-C_k+1$的部分在判定可调度性时(公式(14)、公式(16)和公式(18))被错误地分配到所有处理器上(上述3个公式左边第2项),使对所有处理器都被其他任务占用而JP_k不能运行的最大时长的估计增大,导致判定结果不准确.

有带入作业和没有带入作业时最大干涉之差为

$$DIF_i^{\mathrm{A}}(L)=ICI_i^{\mathrm{A}}(L)-INC_i^{\mathrm{A}}(L) \tag{5}$$

(B) $PRIO_i<PRIO_k$且任务τ_i的主版本出错.此时,τ_i主版本释放的作业和其副版本在出错周期内释放的作业产生干涉.在L内任务τ_i可能运行1次或多次,其中,任意一次主版本作业都可能发生错误.这里假设τ_i主版本在L内的第1个作业发生错误,如图2所示.这样,不论L的长度如何(是否足够任务τ_i运行1次以上),τ_i唯一运行的副版本作业(一次错误假设)可以在L中获得最大运行时间,这是任务τ_i产生最大干涉的最坏情况.

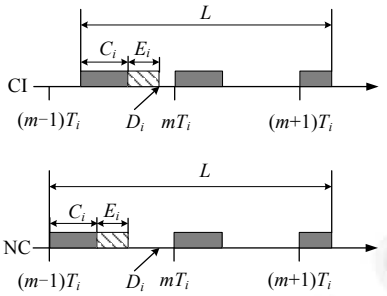

Fig.2 Upper bound of the effective load of a high priority task with fault

图 2 高优先级任务出错时的最大有效负载

高优先级任务 τ_i 出错时,在 L 内的最大有效负载使用公式(6)和公式(7)计算:

$$WCI_i^{\mathrm{B}}(L)=\begin{cases}C_i+E_i+\left\lfloor\dfrac{L'}{T_i}\right\rfloor C_i+\min\left(L'-\left\lfloor\dfrac{L'}{T_i}\right\rfloor T_i,C_i\right), & L'>0\\ \min(C_i+E_i,L), & L'\leqslant 0\end{cases} \tag{6}$$

其中,$L'=L+D_i-C_i-E_i-T_i$,表示除去第 1 个出错作业所在周期,L 剩下的时间长度;

$$WNC_i^{\mathrm{B}}(L)=\begin{cases}C_i+E_i+\left\lfloor\dfrac{L-T_i}{T_i}\right\rfloor C_i+\min\left(C_i,L-T_i-\left\lfloor\dfrac{L-T_i}{T_i}\right\rfloor T_i\right), & L>T_i\\ \min(C_i+E_i,L), & L\leqslant T_i\end{cases} \tag{7}$$

τ_i 产生的最大干涉为

$$ICI_i^{\mathrm{B}}(L)=\min(WCI_i^{\mathrm{B}}(L),L-C_k+1) \tag{8}$$

$$INC_i^{\mathrm{B}}(L)=\min(WNC_i^{\mathrm{B}}(L),L-C_k+1) \tag{9}$$

有带入作业和没有带入作业时,最大干涉之差为

$$DIF_i^{\mathrm{B}}(L)=ICI_i^{\mathrm{B}}(L)-INC_i^{\mathrm{B}}(L) \tag{10}$$

(C) $PRIO_i>PRIO_k$ 且任务 τ_i 出错,此时只有 τ_i 副版本在出错周期内释放的作业产生干涉.很明显:当该作业在 r_k 时刻开始运行并持续占用处理器时,会产生最大干涉.同时,产生 C 类型干涉的任务不符合限制带入作业技术的假定条件[19],因此不需要区分有带入作业和没有带入作业两种情况,在计算最大干涉总量时,将 C 类型干涉作为独立项.τ_i 在 L 内的最大有效负载为

$$W_i^{\mathrm{C}}(L)=\min(E_i,L) \tag{11}$$

τ_i 产生的最大干涉为

$$I_i^{\mathrm{C}}(L)==\min(W_i^{\mathrm{C}}(L),L-C_k+1) \tag{12}$$

至此,可以求得任务集中任意一个任务产生的最大干涉.在 self-fault 模式中,τ_k 的副版本作业 JB_k 只要不晚于 D_k-E_k 时刻开始运行,就能在截止期前正确响应(有最高优先级,不会被抢占).JB_k 的最迟开始运行时间,即 τ_k 主版本作业 JP_k 发生错误的最迟可能时间,不会晚于其无错误情况下的最大响应时间,所以只要 JP_k 能够在 $[r_k,D_k-E_k]$内响应,τ_k 在 self-fault 模式下是可调度的.而在 no-fault 模式中,τ_k 可调度的条件是主版本作业 JP_k 可以在$[r_k,D_k]$内响应,因此,如果 τ_k 在 self-fault 模式下可调度,那么在 no-fault 模式下也一定可调度,所以,NPB-DA 测试需要判定的是 self-fault,high-fault 和 low-fault 出错模式下 τ_k 的可调度性.下面针对这 3 种不同的出错模式,分别建立 JP_k 在问题窗口内受到最大干涉总量的计算公式以及可调度性判定条件.

- self-fault(sf)

这种模式下,要求被测试任务 τ_k 的主版本作业 JP_k 能够在$[r_k,D_k-E_k]$内完成运行.在$[r_k,D_k-E_k]$内系统中没有错误发生的情况下,只有高优先级任务产生 A 类型的干涉,因此,JP_k 在问题窗口$[r_k,D_k-E_k]$内受到的最大涉总量

$I_k^{sf}(D_k-E_k)$ 为

$$I_k^{sf}(D_k-E_k)=\sum_{PRIO_i<PRIO_k} INC_i^{\mathrm{A}}(D_k-E_k)+\sum_{\substack{PRIO_j<PRIO_k\\ \max(m-1)}} DIF_j^{\mathrm{A}}(D_k-E_k) \tag{13}$$

其中, $\sum_{\substack{PRIO_j<PRIO_k\\ \max(m-1)}} DIF_j^{\mathrm{A}}(D_k-E_k)$ 表示干涉差项中前 $m-1$ 大的所有项之和.

公式(13)使用了限制带入作业(limited carry-in)技术[18],即,只允许最多有 $m-1$ 个(m 是处理器个数)任务有带入作业(有带入作业任务产生的干涉大于或等于无带入作业任务),这样可以有效减少对最大干涉总量的过高估计,提高判定准确性.

如果公式(14)成立,则在自身主版本作业出错的情况下(self-fault),τ_k 是可调度的:

$$C_k+\left\lfloor \frac{I_k^{sf}(D_k-E_k)}{m} \right\rfloor \leqslant D_k-E_k \tag{14}$$

公式(14)左边第 2 项的取下界函数表示的是:主版本作业 JP_k 在问题窗口$[r_k,D_k-E_k]$内受到的最大干涉总量 $I_k^{sf}(D_k-E_k)$ 能够同时占用所有处理器的最大时长.在多处理器环境下,只有所有处理器同时被高优先级作业占用,JP_k 才不能被调度.因此,公式(14)左边表示的是 JP_k 完成运行所需的最长时间.

- high-fault(hf)

这种模式下,要求被测试任务 τ_k 的主版本作业 JP_k 能够在$[r_k,D_k]$内完成运行.假设出错任务为 $\tau_f(PRIO_f<PRIO_k)$,τ_f 产生 B 类型的干涉,其他高优先级任务(相对于 τ_k)产生 A 类型干涉,因此 JP_k 在问题窗口$[r_k,D_k]$中受到的最大干涉总量 $I_k^{hf}(D_k)$ 为

$$I_k^{hf}(D_k)=INC_f^{\mathrm{B}}(D_k)+\sum_{\substack{PRIO_i<PRIO_k\\ i\neq f}} INC_i^{\mathrm{A}}(D_k)+\sum_{\substack{PRIO_j<PRIO_k\\ \max(m-1)}} (DIF_j^{\mathrm{A}}(D_k)\cup DIF_f^{\mathrm{B}}(D_k)) \tag{15}$$

其中, $\sum_{\substack{PRIO_j<PRIO_k\\ \max(m-1)}} (DIF_j^{\mathrm{A}}(D_k)\cup DIF_f^{\mathrm{B}}(D_k))$ 表示干涉差项中前 $m-1$ 大的所有项之和.

如果公式(16)成立,则在高优先级任务 τ_f 出错的情况下,τ_k 是可调度的:

$$C_k+\left\lfloor \frac{I_k^{hf}(D_k)}{m} \right\rfloor \leqslant D_k \tag{16}$$

- low-fault(lf)

在这种模式下,要求被测试任务 τ_k 的主版本作业 JP_k 能够在$[r_k,D_k]$内完成运行.假设出错任务为 $\tau_f(PRIO_f>PRIO_k)$,τ_f 产生 C 类型的干涉,高优先级任务(相对于 τ_k)产生 A 类型干涉,因此,JP_k 在问题窗口$[r_k,D_k]$中受到的最大干涉总量 $I_k^{lf}(D_k)$ 为

$$I_k^{lf}(D_k)=I_f^{\mathrm{C}}(D_k)+\sum_{PRIO_i<PRIO_k} INC_i^{\mathrm{A}}(D_k)+\sum_{\substack{PRIO_j<PRIO_k\\ \max(m-1)}} DIF_j^{\mathrm{A}}(D_k) \tag{17}$$

其中, $\sum_{\substack{PRIO_j<PRIO_k\\ \max(m-1)}} DIF_j^{A}(D_k)$ 表示干涉差项中前 $m-1$ 大的所有项之和.

如果公式(18)成立,则在低优先级任务 τ_f 出错的情况下,τ_k 是可调度的:

$$C_k+\left\lfloor \frac{I_k^{lf}(D_k)}{m} \right\rfloor \leqslant D_k \tag{18}$$

测试一个任务 τ_k 可调度性的过程是:依次假设任务集中的一个任务出错,包括被测试任务 τ_k,使用对应的可调度性测试(self-fault,high-fault 或 low-fault)判定在该任务出错时任务 τ_k 的可调度性,如果在所有假设情况下 τ_k 都被判定为可调度的,则任务 τ_k 是可调度的;否则,τ_k 是不可调度的.

测试任务集的可调度性时,对所有任务运行一次测试单任务可调度性过程:如果所有任务都被判定为可调度,则任务集是可调度的;否则,任务集是不可调度的.

3.2 NPB-RTA可调度性测试

NPB-RTA 测试和 NPB-DA 测试的不同之处在于对产生干涉任务的带入作业响应时间的假设:在 NPB-RTA 测试中,假设干涉任务的带入作业在其最大响应时间处响应,而不是截止期,这样减少了有带入作业情况下任务产生干涉的时间窗口长度,也就减少了对最大干涉的过高估计.在 NPB-RTA 可调度性测试中,任务在没有带入作业情况下的最大有效负载和最大干涉的计算公式和 NPB-DA 可调度性测试相同,本节中不再赘述.

在本节中,分别用 R_i^{nf},R_i^{sf},R_i^{hf} 和 R_i^{lf} 表示任务 τ_i 在 no-fault(nf),self-fault(sf),high-fault(hf)和 low-fault(lf)出错模式下的最大响应时间.

NPB-RTA 测试的分析过程和上一节相同,这里首先建立不同类型干涉模式的任务在 r_k 时刻之后的时间区间 L 内有带入作业情况下最大有效负载的计算公式,没有带入作业情况下最大有效负载、两种情况下的最大干涉及差值的计算公式和 NPB-DA 可调度性测试相同,即,公式(2)~公式(5)、公式(7)~公式(10)和公式(12).

(I) $PRIO_i<PRIO_k$ 且系统中没有错误发生.此时,τ_i 在有带入作业情况下产生最大有效负载的运行模式如图 3 所示.

任务 τ_i 在 L 内有带入作业情况下的最大有效负载使用公式(19)计算:

$$WCI_i^{\mathrm{I}}(L)=N_iC_i+\min(C_i,L+R_i^{nf}-C_i-N_iT_i) \tag{19}$$

其中,$N_i=\left\lfloor\dfrac{L+R_i^{nf}-C_i}{T_i}\right\rfloor$,表示在可以在 L 内完整运行的作业的最大数量.

τ_i 在 L 内的没有带入作业情况下最大有效负载、两种情况下的最大干涉及差值分别使用公式(2)~公式(5)计算.

(II) $PRIO_i<PRIO_k$ 且任务 τ_i 的主版本出错.依然假设 τ_i 在 L 内的第 1 个主版本作业出错(和第 3.1 节类型 B 相同的最坏情况假设),此时,τ_i 在有带入作业情况下产生最大有效负载的运行模式如图 4 所示.

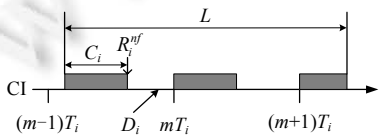

Fig.3 Upper bound of the effective load of a task with carry-in job when no fault exists

图 3 系统中无错误时任务在有带入作业情况下的最大有效负载

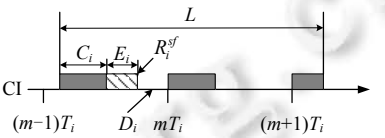

Fig.4 Upper bound of the effective load of a fault task with carry-in job

图 4 出错任务在有带入作业情况下的最大有效负载

任务 τ_i 在 L 内有带入作业情况下的最大有效负载使用公式(20)计算:

$$WCI_i^{\mathrm{II}}(L)=\begin{cases}C_i+E_i+\left\lfloor\dfrac{L'}{T_i}\right\rfloor C_i+\min\left(L'-\left\lfloor\dfrac{L'}{T_i}\right\rfloor T_i,C_i\right), & L'>0\\ \min(C_i+E_i,L), & L'\leqslant 0\end{cases} \tag{20}$$

其中,$L'=L+R_i^{sf}-C_i-E_i-T_i$,表示除去第 1 个出错作业所在周期,$L$ 剩下的时间长度.在 L'内,τ_i 只有主版本作业运行.

τ_i 在 L 内的没有带入作业情况下最大有效负载、两种情况下的最大干涉及差值分别使用公式(7)~公式(10)计算.

(III) $PRIO_i<PRIO_k$ 且出错任务优先级低于任务 τ_i.此时,τ_i 产生最大有效负载的运行模式和类型I中相同,只需将对 τ_i 第 1 个作业的响应时间假设由 R_i^{nf} 改为 R_i^{lf},即

$$WCI_i^{\mathrm{III}}(L)=N_iC_i+\min(C_i,L+R_i^{lf}-C_i-N_iT_i) \tag{21}$$

其中, $N_i=\left\lfloor\frac{L+R_i^{lf}-C_i}{T_i}\right\rfloor$.

τ_i 在 L 内的没有带入作业情况下最大有效负载、两种情况下的最大干涉及差值分别使用公式(2)~公式(5)计算.

(IV) $PRIO_i<PRIO_k$ 且出错任务优先级高于任务 τ_i.此时,τ_i 产生最大有效负载的运行模式也和模式 I 相同,只需将对 τ_i 第 1 个作业的响应时间假设由 R_i^{nf} 改为 R_i^{hf} ,即

$$WCI_i^{\mathrm{IV}}(L)=N_iC_i+\min(C_i,L+R_i^{hf}-C_i-N_iT_i) \tag{22}$$

其中, $N_i=\left\lfloor\frac{L+R_i^{hf}-C_i}{T_i}\right\rfloor$.

τ_i 在 L 内的没有带入作业情况下最大有效负载、两种情况下的最大干涉及差值分别使用公式(2)~公式(5)计算.

(V) $PRIO_i>PRIO_k$ 且任务 τ_i 的主版本出错.此时,τ_i 产生的最大有效负载和最大干涉和第 3.1 节中的 C 类型一致,分别使用公式(11)和公式(12)计算.

至此,可以求得任务集中任意一个任务在 L 内产生的最大干涉,下面针对 4 种不同的出错模式(no-fault, self-fault,high-fault 和 low-fault)分别建立 JP_k 在问题窗口内受到的最大干涉总量和 JP_k 或 JB_k 最大响应时间的计算公式.

- no-fault(nf)

在问题窗口 $[r_k,R_k^{nf}]$ 中,只有高优先级任务产生 I 类干涉,因此,JP_k 受到的最大干涉总量为

$$I_k^{nf}(R_k^{nf})=\sum_{PRIO_i<PRIO_k}INC_i^1(R_k^{nf})+\sum_{\substack{PRIO_i<PRIO_k\\ \max(m-1)}}DIF_i^1(R_k^{nf}) \tag{23}$$

其中, $\sum_{\substack{PRIO_i<PRIO_k\\ \max(m-1)}}DIF_i^1(R_k^{nf})$ 表示干涉差项中前 $m-1$ 大的所有项之和.

从 $R_k^{nf}(1)=C_k$ 开始,迭代计算公式(24)直至 $R_k^{nf}(n+1)=R_k^{nf}(n)$,$R_k^{nf}(n)$ 就是 JP_k 在系统中无错误情况下的最大响应时间,即,τ_k 在系统中无错误情况下的最大响应时间 R_k^{nf} :

$$R_k^{nf}(n+1)=C_k+\left\lfloor\frac{I_k^{nf}(R_k^{nf}(n))}{m}\right\rfloor \tag{24}$$

在公式(24)中, $\left\lfloor\frac{I_k^{nf}(R_k^{nf}(n))}{m}\right\rfloor$ 项表示的是:在第 n 次迭代计算得到的 $R_k^{nf}(n)$ 时间长度内,高优先级作业占用所有处理器的最大时间长度,如果在 $[r_k,R_k^{nf}(n)]$ 内 JP_k 可以完成运行,计算公式(24)会得到 $R_k^{nf}(n+1)=R_k^{nf}(n)$. 如果在 $[r_k,R_k^{nf}(n)]$ 内 JP_k 不能完成运行,说明所有处理器被高优先级作业占用的最大时长 $\left\lfloor\frac{I_k^{nf}(R_k^{nf}(n))}{m}\right\rfloor$ 和 JP_k 需要的运行时间 C_k 之和超过了这个时间区间的长度,因此,计算公式(24)会得到 $R_k^{nf}(n+1)>R_k^{nf}(n)$,需要继续进行迭代计算.迭代计算过程中不会出现 $R_k^{nf}(n+1)<R_k^{nf}(n)$ 的情况,这里使用归纳法来证明:

在第 1 次迭代计算时, $R_k^{nf}(1)=C_k,\left\lfloor\frac{I_k^{nf}(R_k^{nf}(1))}{m}\right\rfloor\geqslant 0$,因此 $R_k^{nf}(2)\geqslant R_k^{nf}(1)$.

假设有 $R_k^{nf}(n)=C_k+\left\lfloor\frac{I_k^{nf}(R_k^{nf}(n-1))}{m}\right\rfloor$ 且 $R_k^{nf}(n)\geqslant R_k^{nf}(n-1)$,由于 $I_k^{nf}(L)$ 函数是非减函数,因此,

$$I_k^{nf}(R_k^{nf}(n))\geqslant I_k^{nf}(R_k^{nf}(n-1)),\left\lfloor\frac{I_k^{nf}(R_k^{nf}(n))}{m}\right\rfloor\geqslant\left\lfloor\frac{I_k^{nf}(R_k^{nf}(n-1))}{m}\right\rfloor.$$

于是可以得到 $R_k^{nf}(n+1)\geqslant R_k^{nf}(n)$.

- self-fault(sf)

在问题窗口 $[r_k,R_k^{sf}]$ 中,在出错时刻前高优先级任务产生 I 类干涉,τ_k 的主版本作业 JP_k 出错后副版本作业 JB_k 不受到干涉,JP_k 出错时刻越迟,JB_k 的响应时间也就越迟,而 JP_k 出错的最迟时间不会晚于其无错误情况下的最大响应时间,因此在在自身主版本作业 JP_k 出错的情况下,τ_k 的最大响应时间为

$$R_k^{sf}=R_k^{nf}+E_k \tag{25}$$

- high-fault(hf)

假设出错任务为 $\tau_f(PRIO_f<PRIO_k)$,在问题窗口 $[r_k,R_k^{hf}]$ 中,τ_f 产生 II 类型干涉,优先级高于 τ_f 的任务产生 III 类型干涉,优先级低于 τ_f 高于 τ_k 的任务产生 IV 类型干涉,因此,JP_k 受到的最大干涉总量为

$$I_k^{hf}(R_k^{hf})=INC_f^{\mathrm{II}}(R_k^{hf})+\sum_{PRIO_i<PRIO_f}INC_i^{\mathrm{III}}(R_k^{hf})+\sum_{PRIO_f<PRIO_j<PRIO_k}INC_j^{\mathrm{IV}}(R_k^{hf})+\sum_{\substack{PRIO_i<PRIO_k\\PRIO_f<PRIO_j<PRIO_k\\ \max(m-1)}}(DIF_f^{\mathrm{II}}(R_k^{hf})\cup DIF_i^{\mathrm{III}}(R_k^{hf})\cup DIF_j^{\mathrm{IV}}(R_k^{hf})) \tag{26}$$

其中, $\sum_{\substack{PRIO_i<PRIO_k\\PRIO_f<PRIO_j<PRIO_k\\ \max(m-1)}}(DIF_f^{\mathrm{II}}(R_k^{hf})\cup DIF_i^{\mathrm{III}}(R_k^{hf})\cup DIF_j^{\mathrm{IV}}(R_k^{hf}))$ 表示干涉差项中前 $m-1$ 大的所有项之和.

从 $R_k^{hf}(1)=C_k$ 开始,迭代计算公式(27)直至 $R_k^{hf}(n+1)=R_k^{hf}(n),R_k^{hf}(n)$ 就是 JP_k 在高优先级任务 τ_f 出错情况下的最大响应时间,即,τ_k 在高优先级任务 τ_f 出错情况下的最大响应时间:

$$R_k^{hf}(n+1)=C_k+\left\lfloor\frac{I_k^{hf}(R_k^{hf}(n))}{m}\right\rfloor \tag{27}$$

依次假设每个高优先级任务(优先级高于 τ_k)出错,每次假设求出的最大响应时间的最大值就是 τ_k 在任意高优先级任务出错情况下的最大响应时间 R_k^{hf}.

- low-fault(lf)

假设出错任务为 $\tau_f(PRIO_f>PRIO_k)$,在问题窗口 $[r_k,R_k^{lf}]$ 中,τ_f 产生 V 类型干涉,优先级高于 JP_k 的任务产生 III 类型干涉,因此,JP_k 受到的最大干涉总量为

$$I_k^{lf}(R_k^{lf})=I_f^{\mathrm{V}}(R_k^{lf})+\sum_{PRIO_i<PRIO_k}INC_i^{\mathrm{III}}(R_k^{lf})+\sum_{\substack{PRIO_i<PRIO_k\\ \max(m-1)}}DIF_i^{\mathrm{III}}(R_k^{lf}) \tag{28}$$

其中, $\sum_{\substack{PRIO_i<PRIO_k\\ \max(m-1)}}DIF_i^{\mathrm{III}}(R_k^{lf})$ 表示干涉差项中前 $m-1$ 大的所有项之和.

从 $R_k^{lf}(1)=C_k$ 开始,迭代计算公式(29)直至 $R_k^{lf}(n+1)=R_k^{lf}(n),R_k^{lf}(n)$ 就是 JP_k 在低优先级任务 τ_f 出错情况下的最大响应时间,即,τ_k 在低优先级任务 τ_f 出错情况下的最大响应时间:

$$R_k^{lf}(n+1)=C_k+\left\lfloor\frac{I_k^{lf}(R_k^{lf}(n))}{m}\right\rfloor \tag{29}$$

依次假设每个低优先级任务(优先级低于 τ_k)出错,每次假设求出的最大响应时间的最大值就是 τ_k 在任意低优先级任务出错情况下的最大响应时间 R_k^{lf}.

测试一个任务 τ_k 可调度性的过程是:假设系统中没有错误,计算被测试任务 τ_k 在 no-fault 情况下的最大响应时间 R_k^{nf},再依次假设任务集中的一个任务出错,包括被测试任务 τ_k,计算 τ_k 在 self-fault,high-fault 和 low-fault 情况下的最大响应时间 R_i^{sf},R_i^{hf} 和 R_k^{lf},如果 4 种出错模式下 τ_k 的最大响应时间都小于 D_k,则任务 τ_k 是可调度的.

使用 NPB-RTA 可调度性测试判定任务集的可调度性时,需要按照优先级顺序由高到低依次测试每个任务的可调度性,这是因为 NPB-RTA 测试判定单个任务可调度性时需要使用高优先级任务在各种出错模式下的最大响应时间.如果所有任务都被判定为可调度,则任务集是可调度的;否则,任务集是不可调度的.

4 时间复杂度分析

在NPB-DA测试中,计算一个任务在被测试任务 τ_k 的问题窗口中最大干涉的时间复杂度是 $O(1)$,计算任务集中所有任务对 τ_k 的最大干涉的时间复杂度就是 $O(n)$,即:在假设某一个任务出错时,判定 τ_k 可调度性的时间复杂度.判定 τ_k 可调度性的过程需要依次假设所有任务出错,一共有 n 种不同假设,因此,判定 τ_k 可调度性的时间复杂度为 $O(n^2)$.判定任务集可调度性的时间复杂度为 $O(n^3)$.

在NPB-RTA测试中,计算一个任务在被测试任务 τ_k 的问题窗口中的最大干涉的时间复杂度是 $O(1)$,计算任务集中所有任务对 τ_k 的最大干涉的时间复杂度就是 $O(n)$,在最坏情况下,每次迭代计算问题窗口长度增加1个单位时间,因此,假设某一个任务出错时,判定 τ_k 可调度性的时间复杂度是 $O(D_k n)$.判定 τ_k 可调度性的过程需要依次假设 n 个任务中的一个出错,所以判定 τ_k 可调度性的时间复杂度是 $O(D_k n^2)$,判定任意一个任务可调度性的时间复杂度是 $O(D_{\max} n^2)$,$D_{\max}$ 是所有任务相对截止期的最大值.判定任务集可调度性的时间复杂度为 $O(D_{\max} n^3)$.

5 优先级分配

在实时系统调度中,优先级分配是影响性能的一个重要方面.为了提高全局固定优先级调度算法的调度能力,研究人员提出多种启发式优先级分配算法,例如DM-DS[20],SM-US[21],TkC[22].文献[23]提出的OPA算法在单处理器调度中是最优的优先级分配算法.文献[17]讨论了OPA算法对可调度性测试的要求,证明了在多处理器调度中,对于满足OPA算法要求(称为OPA兼容)的可调度性测试,OPA算法是最优的优先级分配算法.通过实验说明,使用DA-LC可调度性测试和OPA算法调度随机生成的任务集可以获得最高的可调度比率.

可调度性测试必须满足3个条件才是OPA兼容的[17].

(1) 被测试任务 τ_k 的可调度性可以取决于高优先级任务的自身属性(周期、截止期、最坏情况运行时间),但与它们相互的优先级顺序无关;

(2) 被测试任务 τ_k 的可调度性可以取决于低优先级任务的自身属性,但与它们相互的优先级顺序无关;

(3) 将任意两个相邻优先级任务的优先级互换,如果原先低优先级任务是可调度的,在获得高优先级之后仍然是可调度的.

定理1. NPB-DA可调度性测试是OPA兼容的.

证明:假设被测试任务为 τ_k,任务集中的其他任务可能对 τ_k 产生A,B,C类型干涉中的一种(见第3.1节).从公式(1)~公式(12)可知:任意一个任务 τ_i 在长度为 L 的时间窗口内对 τ_k 的最大干涉只和 τ_i 的自身属性相关,即,T_i,D_i,C_i 和 E_i,因此,τ_k 受到的最大干涉总量也只和其他任务的自身属性相关,所以NPB-DA满足条件(1)和条件(2).假设有两个被判定为可调度的任务 τ_i 和 τ_j,且 $PRIO_i=PRIO_j-1$,即,二者有相邻优先级且 τ_i 的优先级高,如果互换二者的优先级,从公式(13)、公式(15)和公式(17)可知:对于 τ_j 来说,其在问题窗口$[r_j,D_j]$内受到的最大干涉总量减少了 τ_i 产生的项,而其他项不变,即,最大干涉总量下降,所以 τ_j 的优先级提升之后依然是可调度的,因此,NPB-DA满足条件(3).综上所述,NPB-DA可调度性测试是OPA兼容的. □

定理2. NPB-RTA可调度性测试不是OPA兼容的.

证明:假设被测试任务为 τ_k,在NPB-RTA中,计算高优先级任务 τ_i 在长度为 L 的时间窗口内对 τ_k 的最大干涉,需要使用 τ_i 在各种错误模式下的最大响应时间参数(公式(19)~公式(22)),τ_i 的最大响应时间取决于其受到的最大干涉总量(公式(23)、公式(26)和公式(28)),而 τ_i 的优先级决定了其他任务对其的干涉情况,所以高优先级任务的优先级排列会影响各个任务的最大响应时间,也就对被测试任务 τ_k 的可调度性判定产生影响,因此,NPB- RTA可调度性测试不满足条件(1),不是OPA兼容的. □

NPB-DA测试和OPA算法兼容,因此使用NPB-DA测试时OPA算法就是最优的优先级分配算法[17].而NPB-RTA测试和OPA算法不兼容,NPB-RTA测试只能用于使用启发式优先级分配算法分配优先级的任务集,即:先使用某种启发式优先级分配算法给任务集中的所有任务分配优先级,再使用NPB-RTA测试判定任务集的

可调度性.

6 仿真实验

本文采用随机生成任务集的方法,以处理器需求(m)和任务集使用率(U)的比值(m/U)为评价指标,比较本文提出的 FTGS-NPB、不考虑容错的全局调度算法(global scheduling,简称 GS)以及副版本继承主版本优先级的全局容错调度算法(fault tolerant global scheduling with priorityinheritance,简称 FTGS-PI)在调度相同任务集时需要的处理器资源,m/U 比值越小,说明该算法调度性能越好.在实验中加入 GS 算法,是为了直观地展示不考虑容错时调度相同任务集需要的处理器资源,使读者可以清晰地看出实现容错需要的资源代价.FTGS-PI 是文献[11]中提出的容错调度算法取 f=1 的形式,该算法采用优先级继承策略,即,发生错误后副版本作业就绪并继承主版本作业的优先级.使用优先级继承策略会造成副版本作业在有限的运行窗口内受到很大的干涉,于是,为保证其实时性,需要增加大量的额外处理器资源.FTGS-NPB 主要解决的就是这个问题.

生成随机任务集采用和文献[6–10]中类似的方法,任务集中单个任务的使用率(C/T)上限设定为 a,任务周期 T 在[1,500]内均匀分布,最坏情况运行时间 C 取[1,aT]中的随机值,截止期 D=T,任务的副版本简单的设定为主版本的复制.

实验共 4 组,a 分别取 0.2,0.3,0.4 和 0.5.在每组实验中,任务集中的任务数量 n 以 50 个为间隔取[50,300]中的整数,对每一个(a,n)组合重复 30 次实验,每次实验分别使用在不考虑容错时性能较好的 DkC 优先级分配算法和 OPA 优先级分配算法[17]分配优先级,使用 DkC 算法时采用基于响应时间分析(RTA)的可调度性测试,使用 OPA 算法时采用基于截止期分析(DA)的可调度性测试(GS 中采用文献[17]中的可调度性测试,FTGS-PI 中使用文献[11]中的可调度性测试,FTGS-NPB 中使用 NPB-DA 可调度性测试).在单次实验中通过在[$\lceil U \rceil$,n]区间从低到高搜索的方法获得使任务集可调度的最少处理器数量 m,取 30 次实验平均值作为有效结果.实验结果如图 5~图 8 所示.

从图 5~图 8 中可以看出:为实现容错,FTGS-NPB 和 FTGS-PI 都需要额外的处理器资源.与 GS 相比:

- 当使用 DkC 算法和基于 RTA 的测试时,FTGS-NPB 的 m/U 比值最小增加了 10.04%(a=0.2,n=50),最大增加了 36.24%(a=0.5,n=200),平均增加 22.98%;FTGS-PI 的 m/U 比值最小增加了 22.23%(a=0.2,n=50),最大增加了 65.37%(a=0.4,n=300),平均增加 43.52%;
- 当使用 OPA 算法和基于 DA 的测试时,FTGS-NPB 的 m/U 比值最小增加了 3.84%(a=0.2,n=50),最大增加了 16.44%(a=0.4,n=100),平均增加 11.67%;FTGS-PI 的 m/U 比值最小增加了 12.83%(a=0.2,n=200),最大增加了 34.06%(a=0.4,n=100),平均增加 25.54%.

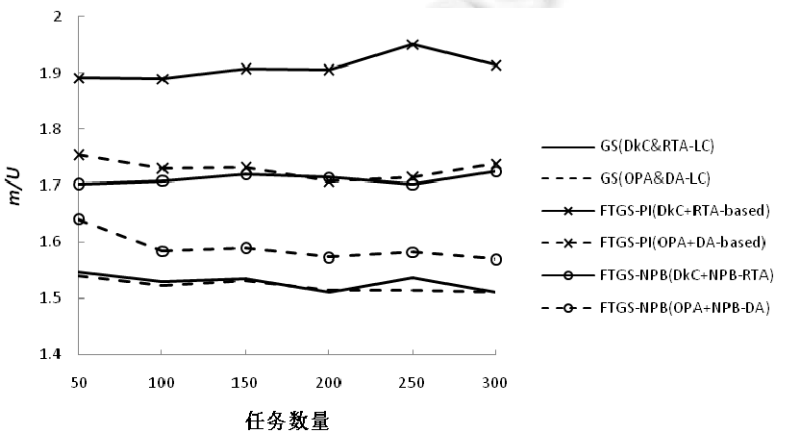

Fig.5 m/U ratio when a=0.2

图 5 a=0.2 时,不同算法的 m/U 比值

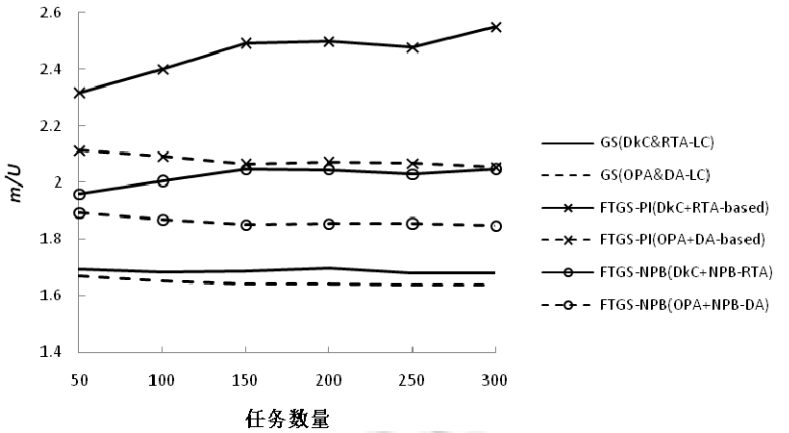

Fig.6 m/U ratio when a=0.3

图 6 a=0.3 时,不同算法的 m/U 比值

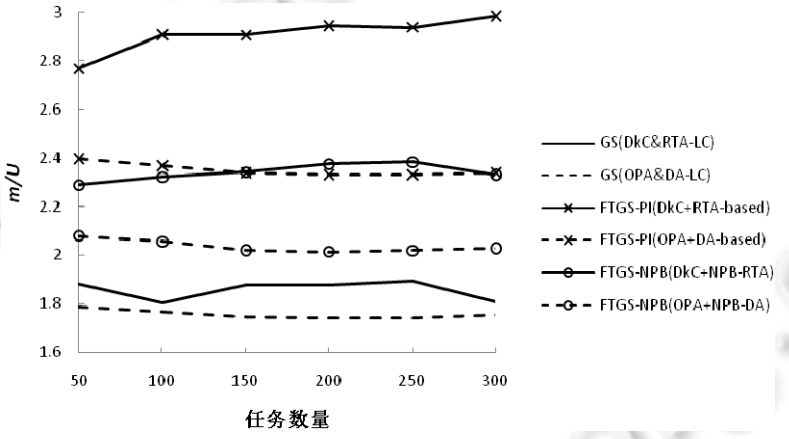

Fig.7 m/U ratio when a=0.4

图 7 a=0.4 时,不同算法的 m/U 比值

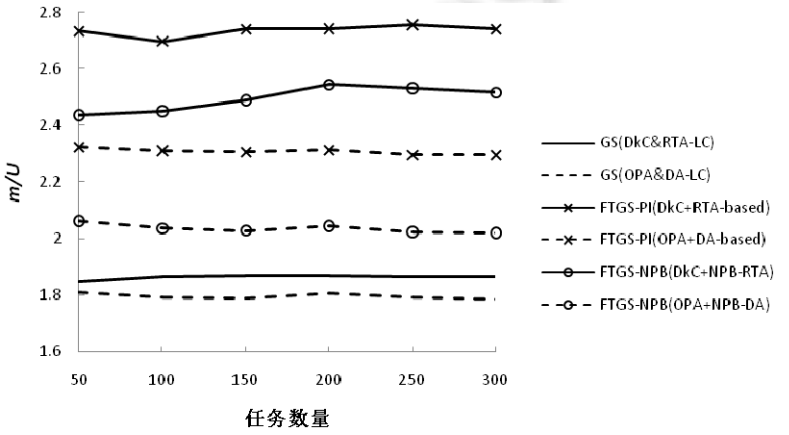

Fig.8 m/U ratio when a=0.5

图 8 a=0.5 时,不同算法的 m/U 比值

当采用相同的优先级分配算法和可调度性测试时,在各种 a,n 取值下,FTGS-NPB 的 m/U 比值都比 FTGS-PI 小,即,FTGS-NPB 需要较少的处理器资源就可以容错调度相同的实时任务集.产生这一结果的原因在于:在 FTGS-PI 中,主版本作业出错后留给副版本作业运行的时间窗口相对较短,而副版本作业继承主版本作业的优先级,如果其优先级较低,在这个时间窗口内处理器会被大量高优先级作业长时间占用,该副版本作业很容易错失截止期,为了保证低优先级副版本作业的实时性,就需要增加大量处理器资源来运行低优先级副版本作业运行窗口内的大量高优先级作业,导致 FTGS-PI 的 m/U 比值大幅增加.而在 FTGS-NPB 中,副版本作业有最高优先级,不受到干涉,只需要主版本作业的最迟出错时间,即,最大响应时间在 D_k–E_k 之前,副版本作业就一定可以在截止期之前响应.当 a 取值较小时,所有任务的副版本最坏情况运行时间都远小于其截止期,因此,相比于 GS, FTGS-NPB 只是需要在一个略小的时间区间内能够成功调度主版本作业,需要的额外处理器资源也就很少.当 a 取值较大时,优先级分配算法会将高优先级分配给使用率大的任务,高优先级主版本的最坏情况响应时间短,不需要额外处理器资源或是需要少量额外处理器资源就能够满足副版本正确响应的需求.

同时,在同一种调度算法中(GS,FTGS-PI 或 FTGS-NPB),使用 OPA 算法和基于 DA 的可调度测试时的 m/U 比值比使用 DkC 算法和基于 RTA 的可调度测试时小.这一实验结果和文献[17]中的实验结果类似,说明前一种组合的调度能力在考虑容错和不考虑容错的情况下都更强.优先级分配算法对调度算法的影响可以从 FTGS-NPB(DkC+NPB-RTA)和 FTGS-PI(OPA+DA-based)两条曲线中看出:当 a 取 0.2,0.3 和 0.4 时,采用 DkC 优先级分配算法和 NPB-RTA 测试的 FTGS-NPB 算法的 m/U 比值略低于采用 OPA 优先级分配算法和 DA-based 测试的 FTGS-PI 算法;但当 a 取 0.5 时,后者的 m/U 比值低于前者,说明尽管 FTGS-NPB 算法的调度性能比 FTGS-PI 算法好,优先级分配算法和可调度测试的差异可能会抵消调度算法的性能提升.

7 结束语

本文针对全局容错调度中副版本运行时间窗口小、调度难度大的问题,提出了副版本不可抢占的全局容错调度算法 FTGS-NPB.不可抢占的副版本可以在主版本出错后的最短时间内响应,最大程度提高了副版本的实时性,从而减少了实现容错所需的额外资源.仿真实验结果说明:和基于优先级继承策略的全局容错调度算法相比, FTGS-NPB 可以节省大量的处理器资源.

References:

[1] Monot A, Navet N, Bavoux B, Simonot-Lion F. Multisource software on multicore automotive ECUs—Combining runnable sequencing with task scheduling. IEEE Trans. on Industrial Electronics, 2012,59(10):3934–3942. [doi: 10.1109/TIE.2012.2185913]

[2] Navet N, Monot A, Bavoux B, Simonot-Lion F. Multi-Source and multicore automotive ECUs—OS protection mechanisms and scheduling. In: Proc. of the IEEE Int'l Symp. on Industrial Electronics. IEEE, 2010. 3734–3741. [doi: 10.1109/ISIE.2010.5637677]

[3] Mossinger J. Software in automotive systems. IEEE Software, 2010,27(2):92–94. [doi: 10.1109/MS.2010.55]

[4] Gaska T, Werner B, Flagg D. Applying virtualization to avionics systems—The integration challenges. In: Proc. of the 29th Digital Avionics Systems Conf. Salt Lake City: IEEE, 2010. 2155–2195. [doi: 10.1109/DASC.2010.5655297]

[5] Krishna CM. Fault-Tolerant scheduling in homogeneous real-time systems. ACM Computing Surveys, 2014,46(4):48:1–48:34. [doi: 10.1145/2534028]

[6] Bertossi AA, Mancini LV, Menapace A. Scheduling hard-real-time tasks with backup phasing delay. In: Proc. of the 10th IEEE Int'l Symp. on Distributed Simulation and Real-Time Applications. IEEE, 2006. 107–118. [doi: 10.1109/DS-RT. 2006.33]

[7] Wang J, Sun JL, Wang XY, Yang XH, Wang SK, Chen JB. Efficient scheduling algorithm for hard real-time tasks in primary-backup based multiprocessor systems. Ruan Jian Xue Bao/Journal of Software, 2009,20(10):2628–2636. http://www.jos.org.cn/1000-9825/577.htm [doi: 10.3724/SP.J.1001.2009.00577]

[8] Zhu P, Yang FM, Tu G. Real-Time fault-tolerant scheduling for distributed systems based on improving priority of passive backup. Journal of Computer Research and Development, 2010,47(11):2003–2010 (in Chinese with English abstract). [doi: 10.3724/SP.J.1001.2009.00577]

[9] Chen HM, Luo W, Wang W, Xiang J. A novel real-time fault-tolerant scheduling algorithm based on distributed control systems. In: Proc. of the Int'l Conf. on Computer Science and Service System. Nanjing: IEEE, 2011. 80–83. [doi: 10.1109/CSSS.2011.5972233]

[10] Zhu P, Yang FM, Tu G, Zhang J, Zhou ZY. Feasible fault-tolerant scheduling algorithm for distributed hard-real-time system. Ruan Jian Xue Bao/Journal of Software, 2012,23(4):1010−1021 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4004.htm [doi: 10.3724/SP.J.1001.2012.04004]

[11] Pathan RM, Jonsson J. FTGS: Fault-tolerant fixed-priority scheduling on multiprocessors. In: Proc. of the IEEE 10th Int'l Conf. on Trust, Security and Privacy in Computing and Communications. Changsha: IEEE, 2011. 1164−1175. [doi: 10.1109/TrustCom.2011.158]

[12] Berten V, Goossens J, Jeannot E. A probabilistic approach for fault tolerant multiprocessor real-time scheduling. In: Proc. of the 20th Int'l Parallel and Distributed Processing Symp. Rhodes Island: IEEE, 2006. 1−10. [doi: 10.1109/IPDPS.2006.1639409]

[13] Samala AK, Mallb R, Tripathy C. Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm. Swarm and Evolutionary Computation, 2014,14(1):92−105. [doi: 10.1016/j.swevo.2013.10.002]

[14] Baumann R. Soft errors in advanced computer systems. Design & Test of Computers, 2005,22(3):258−266. [doi: 10.1109/MDT.2005.69]

[15] Krishna I, Krishna CM. Fault-Tolerant Systems. New York: Morgan Kaufmann Publishers, 2007.

[16] Guan N, Stigge M, Wang Y, Ge Y. New response time bounds for fixed priority multiprocessor scheduling. In: Proc. of the Real-Time Systems Symp. Washington: IEEE, 2009. 387−397. [doi: 10.1109/RTSS.2009.11]

[17] Davis RI, Burns A. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. Real-Time Systems, 2011,47(1):1−40. [doi: 10.1007/s11241-010-9106-5]

[18] Lee J, Shin I. Limited carry-in technique for real-time multi-core scheduling. Journal of Systems Architecture, 2013,59(7):372−375. [doi: 10.1016/j.sysarc.2013.05.012]

[19] Davis RI, Burns A, Marinho J, Nelis V, Petters SM, Bertogna M. Global fixed priority scheduling with deferred pre-emption. In: Proc. of the IEEE 19th Int'l Conf. on Embedded and Real-Time Computing Systems and Applications. Taipei: IEEE, 2013. 1−11. [doi: 10.1109/RTCSA.2013.6732198]

[20] Bertogna M, Cirinei M, Lipari G. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In: Proc. of the 9th Int'l Conf. on Principles of Distributed Systems. Pisa: Springer-Verlag, 2005. 306−321. [doi: 10.1007/11795490_24]

[21] Andersson B. Global static-priority preemptive multiprocessor scheduling with utilization bound 38%. In: Proc. of the 12th Int'l Conf. on Principles of Distributed Systems. Luxor: Springer-Verlag, 2008. 73−88. [doi: 10.1007/978-3-540-92221-6_7]

[22] Andersson B, Jonsson J. Fixed-Priority preemptive multiprocessor scheduling: To partition or not to partition. In: Proc. of the 7th Int'l Conf. on Real-Time Computing Systems and Applications. Cheju: IEEE, 2000. 337−346. [doi: 10.1109/RTCSA.2000.896409]

[23] Audsley NC. On priority assignment in fixed priority scheduling. Information Processing Letters, 2001,79(1):39−44. [doi: 10.1016/S0020-0190(00)00165-4]

附中文参考文献:

[8] 朱萍,阳富民,涂刚.基于被动副版本优先级提高策略的分布式实时容错调度.计算机研究与发展,2010,47(11):2003−2010.

[10] 朱萍,阳富民,涂刚,张杰,周正勇.一种可行的分布式硬实时容错调度算法.软件学报,2012,23(4):1010−1021. http://www.jos.org.cn/1000-9825/4004.htm [doi: 10.3724/SP.J.1001.2012.04004]

彭浩(1984－),男,安徽合肥人,博士,主要研究领域为硬实时系统.

孙峰(1989－),男,博士,主要研究领域为嵌入式系统.

陆阳(1967－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为安全关键系统.

韩江洪(1954－),男,教授,博士生导师,主要研究领域为安全关键系统.