计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20148380
53(2):354-361,2016

副版本优先级可提升的全局容错调度算法

彭 浩 韩江洪 魏振春 卫 星
(合肥工业大学计算机与信息学院 合肥 230009)
(hanjh@hfut.edu.cn)

Fault Tolerant Global Scheduling with Backup Priority Promotion

Peng Hao, Han Jianghong, Wei Zhenchun, and Wei Xing
(*School of Computer and Information, Hefei University of Technology, Hefei* 230009)

Abstract Fault tolerance is of great importance in hard real-time systems due to the impossibility of eliminating faults. In such a system the fault tolerant scheduling algorithm plays a critical role for achieving fault tolerance capability. In primary-backup scheme based fault tolerant global scheduling algorithms, the execution window of backup is relatively small. When priority inheritance strategy is adopted, the response time of the backup is likely too long to guarantee deadline requirement. For improving the real time property of the backup, we propose a fault tolerant global scheduling algorithm based on backup priority promotion strategy—FTGS-BPP. In FTGS-BPP, the backup has a higher priority than its corresponding primary so that during the execution the backup suffers less interference. Consequently the response time of the backup is reduced which means better real time performance. FTGS-BPP can achieve fault tolerance with less processors than the algorithms which follow priority inheritance strategy. A backup priority searching algorithm is also proposed. The simulation result shows that, compared with the fault tolerant global scheduling algorithm based on priority inheritance strategy, FTGS-BPP is able to reduce processor requirement significantly when scheduling the same task set.

Key words multiprocessor; fault-tolerant scheduling; global scheduling; hard real-time systems; priority promotion

摘 要 在主副版本机制的全局容错调度中,副版本运行窗口短,采用优先级继承策略的副版本响应时间长,容易错失截止期.针对副版本实时性差的问题,提出基于优先级提升策略的全局容错调度算法(fault tolerant global scheduling with backup priority promotion, FTGS-BPP),通过赋予副版本比主版本高的优先级,减少副版本在运行过程中受到的干扰,缩短了副版本的响应时间,改善了副版本的实时性,从而减少了实现容错所需的额外处理器资源.仿真结果表明,和采用优先级继承策略的全局容错调度算法相比,FTGS-BPP 在调度相同的任务集时明显降低了处理器资源需求.

收稿日期:2014-12-17;**修回日期**:2015-06-09
基金项目:国家自然科学基金项目(61370088);国家国际科技合作专项项目(2014DFB10060);国家科技支撑计划项目(2013BAH51F01,2013BAH51F02)
This work was supported by he National Natural Science Foundation of China (61370088), the International S&T Cooperation Program of China (2014DFB10060), and the National Key Technology R&D Program of the Ministry of Science and Technology (2013BAH51F01,2013BAH51F02).
通信作者:魏振春(weizc@hfut.edu.cn)

关键词　多处理器;容错调度;全局调度;硬实时系统;优先级提升

中图法分类号　TP316.2

为了满足对计算能力不断增长的需求,多处理器芯片被越来越多的嵌入式硬实时系统采用,例如航空电子系统[1]、汽车电子系统[2]、工业控制系统[3]等.硬实时系统同时具有实时性和可靠性要求.实时性是指系统中的每个任务都必须能够在某个固定时限内完成设计的功能,这个时限被称为截止期;可靠性是指系统可以无故障执行设计功能的能力.有效的硬实时容错调度算法是满足系统实时性和可靠性要求的关键[4-6].

主副版本机制是实现容错调度最主要的方法[7].在该方法中,系统中的每个任务有一个主版本以及一个或多个副版本,任务开始运行时只有主版本作业就绪,副版本作业都处于挂起状态,如果主版本出错,则会有一个副版本作业进入就绪状态并被调度运行,该副版本作业出错则继续调度下一个副版本作业,直至该任务任意一个版本的作业正确响应,副版本作业的启动顺序由设计人员预先确定.

和多处理器实时调度算法相同,容错调度算法也分为分组容错调度算法和全局容错调度算法.前者将任务分组,并保证将同一个任务的主副版本分到不同的组中,每组单独占用一个固定的处理器,并使用单处理器实时调度算法调度每个组中的任务[5,8-9];后者不固定任务的运行位置,任务可以在处理器间迁移,调度器总是调度优先级高的主版本或者副版本在功能完好的处理器上运行,这样不会出现有处理器空闲而任务在其他处理器上等待却不能使用空闲处理器运行的情况.和分组容错调度相比,全局容错调度的研究还很少,文献[10]提出了基于概率模型的动态优先级全局容错调度算法 EDF^k,并建立了基于使用率的可调度测试.文献[11]研究了基于优先级继承策略的固定优先级全局容错调度算法 FTGS.优先级继承策略的问题是会造成副版本实时性差,在主副版本机制的容错调度中主版本和副版本的截止期是相同的,而副版本作业只有在主版本作业出现故障之后才会进入就绪状态被调度运行,因此可供副版本作业运行的时间窗口较短,容易错失截止期,为了保证副版本能够在短时间内响应,需要增加大量额外的处理器资源.

优先级提升策略是解决副版本实时性差的有效方法,该策略通过赋予副版本比较高的优先级,使副版本作业可以在较短时间内响应,从而满足实时性要求.本文基于优先级提升策略,针对固定优先级偶发实时任务集,提出副版本优先级可提升的全局容错调度算法(fault tolerant global scheduling with backup priority promotion, FTGS-BPP),解决主副版本机制的全局容错调度中副版本实时性差的问题.

本文的研究建立在以下假设的基础上:

1) 任务之间相互独立,任务中没有并行运行的代码,即一个任务同一时刻只能占用一个处理器;

2) 在一次作业的生命周期内只会出现一次故障,这一假设在容错调度领域被广泛采用[5,9-10,12];

3) 故障持续时间很短,即是瞬态故障[13-14];

4) 系统的硬件平台为同构多处理器平台,处理器速度相同,并共享存储器.

1　任务模型和相关定义

多处理器硬实时系统包含 1 个由 n 个实时任务组成的偶发任务集和 1 个由 m 个同构处理器组成的硬件平台.任务 $\tau_i(i=1,2,\cdots,n)$包含 1 个主版本和 1 个副版本,用 6 元组 $\tau_i=\langle C_i^{\mathrm{P}},C_i^{\mathrm{B}},D_i,T_i,P_i^{\mathrm{P}},P_i^{\mathrm{B}}\rangle$表示.其中,$C_i^{\mathrm{P}}$ 和 C_i^{B} 分别表示主、副版本的最坏情况运行时间;D_i 表示相对截止期,即任务一次运行的释放时刻和完成时限之间的时间长度;T_i 表示任务相邻 2 次释放作业的最小时间间隔;P_i^{P} 和 P_i^{B} 分别表示主、副版本的优先级,在这里规定数值越小代表越高的优先级,即最高优先级为 1,最低优先级为 n.在后面的章节中,不需要特别指明作业的序数时,用 J_i^{P} 和 J_i^{B} 分别表示任务 τ_i 主、副版本的一次作业,r_i 表示 J_i^{P} 和 J_i^{B} 的释放时刻,d_i 表示作业的绝对截止期,即 $d_i=r_i+D_i$,分别用 R_i^{P} 和 R_i^{B} 表示 τ_i 主、副版本的最大响应时间,用 R_i^{NF} 表示系统中无故障时 τ_i 主版本的最大响应时间.

任务的主、副版本在某一时刻(时间触发或事件触发)同时分别释放 1 次作业,主版本释放的作业立刻进入就绪状态,副版本释放的作业进入挂起状态,只有在对应主版本作业发生故障时才会进入就绪状态,主版本作业正确响应时,取消对应的副版本作业.就绪作业进入 1 个全局就绪作业队列,调度器按照优先级顺序调度前 n 个(n 是处理器数量)就绪作业

在功能完好的处理器上运行,作业和处理器之间没有绑定的关系,即作业可以在任意1个处理器上运行.在FTGS-BPP中,抢占是始终允许的,任意时刻就绪的高优先级作业会抢占低优先级作业,如果有多个低优先级作业则抢占其中优先级最低的,同时作业迁移也是始终允许的,即被抢占作业可以在另一个处理器上恢复运行.

在后面章节的讨论中,需要用到下面这些定义:

定义1. 任务 τ_i 在时间区间 A 内的负载是指在该时间区间内 τ_i 作业的累积运行时间长度.

定义2. 高优先级任务 τ_i 在时间区间 A 内对作业 J_k 产生的干扰负载是指 τ_i 释放的作业处于运行状态而 J_k 处于就绪状态但不能运行的累积时间长度.

定义3. 在时间区间 A 内作业 J_k 受到的累积干扰负载是指在该区间内所有高优先级任务产生的干扰负载的总和.

定义4. 在时间区间 A 内作业 J_k 受到的干扰是指在区间内 J_k 处于就绪状态但得不到运行的累积时间长度.

定义5. 如果作业的释放时间在时间区间之前,截止期在该区间开始时刻之后,则称该作业为带入作业,如图1中的 J_1.分别用CI(carry-in)和NC(no carry-in)表示一个任务是否有带入作业的情况.

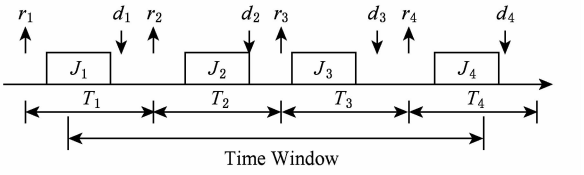

Fig. 1 Definitions of carry-in and carry-out job.

图1 带入作业和带出作业的定义

定义6. 如果作业的释放时间在区间结束时间之前、截止期在区间结束时间之后,则称该作业为带出作业,如图1中的 J_4.

2 可调度性测试

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,即在运行过程中没有作业会错失截止期.不同的硬实时调度算法安排任务运行次序、位置的策略不同,这二者决定了可调度性测试的形式,因此需要给每种调度算法建立专属的可调度性测试.

由于多处理器实时系统的临界时刻(critical instant)还是未知的,目前的可调度性判定条件都是任务集可调度的充分而非必要条件[15].文献[16-17]分别基于响应时间分析法(response time analysis, RTA)和截止期分析法(deadline analysis, DA)建立了多处理器实时系统的可调度性判定条件RTA-LC和DA-LC.基于响应时间分析的可调度性判定准确性更高,因此FTGS-BPP算法的可调度性测试采用这种方法,按照优先级从高到低的顺序依次判定任务集中每个任务的可调度性,如果所有任务都是可调度的,则任务集是可调度的.

2.1 可调度性分析

假设被测试任务为 τ_k. τ_k 可调度要求:1) τ_k 无故障时可以在截止期前正确响应;2) τ_k 发生故障时可以在截止期前正确响应.其中前者又包含其他任务发生故障和没有任务发生故障2种情况.

假设 τ_k 的主、副版本在时刻 r_k 各释放一次作业 J_k^{P} 和 J_k^{B}, τ_k 可能遇到的故障模式有3种:1)系统中没有故障发生, τ_k 使用主版本作业 J_k^{P} 的计算结果, J_k^{P} 的最大响应时间 R_k^{NF} 就是无故障时 τ_k 的最大响应时间;2)其他任务发生故障, τ_k 依然使用主版本作业 J_k^{P} 的计算结果, J_k^{P} 的最大响应时间 R_k^{P} 就是 τ_k 的最大响应时间;3) τ_k 发生故障,即 τ_k 的主版本作业发生故障, τ_k 使用副版本作业 J_k^{B} 的计算结果, J_k^{B} 的最大响应时间 R_k^{B} 就是 τ_k 发生故障时的最大响应时间.

只有 τ_k 在3种故障模式下的最大响应时间都不大于其截止期,即式(1)成立, τ_k 才是可调度的.2.2~2.4节将详细讨论如何计算3种故障模式下的最大响应时间.

$$\begin{cases} R_k^{\mathrm{NF}} \leqslant D_k; \\ R_k^{\mathrm{P}} \leqslant D_k; \\ R_k^{\mathrm{B}} \leqslant D_k. \end{cases} \tag{1}$$

2.2 系统中无故障

当系统中没有故障发生时,所有副版本作业都不会进入就绪状态,只有主版本作业运行,因此调度情况和不考虑容错能力的调度算法类似,使用RTA-LC[16]中的方法可以计算 J_k^{P} 的最大响应时间 R_k^{NF}.为了便于理解,这里根据本文使用的任务模型改写该计算方法中使用的公式,并简要叙述计算过程,这些在后面讨论其他故障模式时也需要使用.

首先要计算高优先级任务 τ_i ($P_i^{\mathrm{P}} < P_k^{\mathrm{P}}$)在从 r_k 开始长度为 L 的时间窗口(后面简称为 L)内产生的最大负载.图2是任务 τ_i 在 L 内有带入作业(CI)和没有带入作业(NC)时产生最大负载的释放模式,区

分这2种不同情况是为了使用限制带入作业(limited carry-in)技术[16,18],以提高测试准确性.

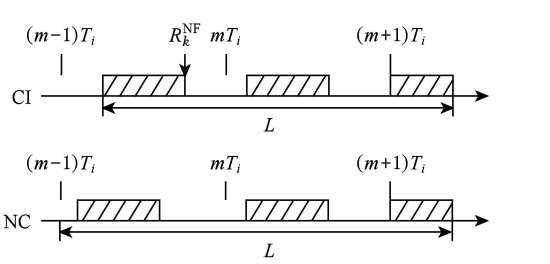

Fig. 2　Release pattern for calculating the upper bound of workload.

图2　产生最大负载的作业释放模式

分别用 $W_i^{\mathrm{CI}}(L)$ 和 $W_i^{\mathrm{NC}}(L)$ 表示 τ_i 在有带入作业(CI)和没有带入作业(NC)情况下的最大负载,使用式(2)和式(3)计算:

$$W_i^{\mathrm{CI}}(L) = N_i \times C_i^{\mathrm{P}} + \min(C_i^{\mathrm{P}}, L + R_i^{\mathrm{NF}} - C_i^{\mathrm{P}} - N_i \times T_i), \quad (2)$$

其中 $N_i = \left\lfloor \dfrac{L + R_i^{\mathrm{NF}} - C_i^{\mathrm{P}}}{T_i} \right\rfloor$,表示可以在 L 内完整运行作业的数量,$\min(C_i^{\mathrm{P}}, L + R_i^{\mathrm{NF}} - C_i^{\mathrm{P}} - N_i \times T_i)$ 表示带出作业产生的最大负载;

$$W_i^{\mathrm{NC}}(L) = \left\lfloor \frac{L}{T_i} \right\rfloor \times C_i^{\mathrm{P}} + \min\left(C_i^{\mathrm{P}}, L - \left\lfloor \frac{L}{T_i} \right\rfloor \times T_i\right). \quad (3)$$

任务 τ_i 在 L 内有带入作业(CI)和没有带入作业(NC)时产生的最大干扰负载 $I_i^{\mathrm{CI}}(L)$ 和 $I_i^{\mathrm{NC}}(L)$ 分别使用式(4)和式(5)计算:

$$I_i^{\mathrm{CI}}(L) = \min(W_i^{\mathrm{CI}}(L), L - C_k^{\mathrm{P}} + 1); \quad (4)$$

$$I_i^{\mathrm{NC}}(L) = \min(W_i^{\mathrm{NC}}(L), L - C_k^{\mathrm{P}} + 1). \quad (5)$$

限制 τ_i 的最大干扰负载不超过 $L - C_k^{\mathrm{P}} + 1$ 的原因在于:如果 τ_i 的最大干扰负载到达 $L - C_k^{\mathrm{P}} + 1$,τ_i 释放的作业在 L 内占用一个处理器的时间已经导致 J_k^{P} 无法在该处理器上被调度(运行高优先级负载不超过 $L - C_k^{\mathrm{P}}$ 的处理器才有足够的资源运行 J_k^{P});而继续增加 τ_i 的最大干扰负载会导致在计算最大干扰时,超过 $L - C_k^{\mathrm{P}} + 1$ 的部分被错误地分配到每个处理器上,影响最终计算结果.

用 $F_i(L)$ 表示 τ_i 在有带入作业(CI)和没有带入作业(NC)情况下最大干扰负载的差值:

$$F_i(L) = I_i^{\mathrm{CI}}(L) - I_i^{\mathrm{NC}}(L). \quad (6)$$

至此,通过式(7)可以计算得到所有高优先级主版本在 L 内产生的最大累积干扰负载 $S_k(L)$. 式(7)使用限制带入作业(limited carry-in)技术,只允许 $F_i(L)$ 项前 $m-1$(m 是处理器个数)大的任务有带入作业,该技术的具体讨论请见文献[16,18].

$$S_k(L) = \sum_{P_i^{\mathrm{P}} < P_k^{\mathrm{P}}} I_i^{\mathrm{NC}}(L) + \sum_{\substack{\max(m-1) \\ P_i^{\mathrm{P}} < P_k^{\mathrm{P}}}} F_i(L), \quad (7)$$

其中 $\sum\limits_{\substack{\max(m-1) \\ P_i^{\mathrm{P}} < P_k^{\mathrm{P}}}} F_i(L)$ 表示优先级高于 P_k^{P} 的所有任务的 $F_i(L)$ 项中前 $m-1$ 大的项的数量和.

在 L 内,J_k^{P} 受到的最大干扰 $I_k(L)$ 通过式(8)计算. 式(8)等号的右边表示所有高优先级主版本产生的最大累积干扰负载在 L 内能够同时占用所有处理器的最密集排列,这种排列导致 J_k^{P} 就绪但无法运行的时间最长.

$$I_k(L) = \left\lfloor \frac{S_k(L)}{m} \right\rfloor. \quad (8)$$

从 $R_k^{\mathrm{NF}}(0) = C_k^{\mathrm{P}}$ 开始,迭代计算式(9),直至 $R_k^{\mathrm{NF}}(n+1) = R_k^{\mathrm{NF}}(n)$,$R_k^{\mathrm{NF}}(n)$ 即为 J_k^{P} 在系统中无故障情况下的最大响应时间 R_k^{NF}.

$$R_k^{\mathrm{NF}}(n+1) = C_k^{\mathrm{P}} + I_k(R_k^{\mathrm{NF}}(n)). \quad (9)$$

2.3　其他任务发生故障

假设故障任务为 τ_f,J_k^{P} 除了受到高优先级任务主版本的干扰,还受到 τ_f 一次副版本作业 J_f^{B} 的干扰. 高优先级主版本的最大负载和最大干扰负载使用式(2)~(6)计算,但由于考虑了有故障发生的情况,在式(2)中应使用考虑故障的主版本最大响应时间 R_i^{P} 代替 R_i^{NF}.

τ_f 可能在任意时刻出错,即 J_f^{B} 可能在任意时刻就绪,这里对其做出最坏情况假设,即 J_f^{B} 在 r_k 时刻就绪. 考虑到只有优先级高于 J_k^{P} 时 J_f^{B} 才对其产生干扰,J_f^{B} 产生的最大干扰负载 $I_f(L)$ 使用式(10)计算:

$$I_f(L) = \begin{cases} \min(C_f^{\mathrm{B}}, L - C_k^{\mathrm{P}} + 1), & P_f^{\mathrm{B}} \leqslant P_k^{\mathrm{P}}; \\ 0, & P_f^{\mathrm{B}} > P_k^{\mathrm{P}}. \end{cases} \quad (10)$$

所有高优先级主、副版本在 L 内产生的最大累积干扰负载 $S_k(L)$ 为:

$$S_k(L) = I_f(L) + \sum_{P_i^{\mathrm{P}} < P_k^{\mathrm{P}}} I_i^{\mathrm{NC}}(L) + \sum_{\substack{\max(m-1) \\ P_i^{\mathrm{P}} < P_k^{\mathrm{P}}}} F_i(L). \quad (11)$$

J_k^{P} 受到的最大干扰 $I_k(L)$ 通过式(8)计算.

从 $R_{k,f}^{\mathrm{P}}(0) = C_k^{\mathrm{P}}$ 开始,迭代计算式(12)直至 $R_{k,f}^{\mathrm{P}}(n+1) = R_{k,f}^{\mathrm{P}}(n)$,$R_{k,f}^{\mathrm{P}}(n)$ 即为 τ_f 故障时 J_k^{P} 的最大响应时间 $R_{k,f}^{\mathrm{P}}$.

$$R_{k,f}^{\mathrm{P}}(n+1) = C_k^{\mathrm{P}} + I_k(R_{k,f}^{\mathrm{P}}(n)). \quad (12)$$

J_k^P 在任意一个任务(除 τ_k 之外)故障时的最大响应时间 R_k^P 是所有 $R_{k,f}^P$ 中的最大值,即:

$$R_k^P=\max\{R_{k,f}^P \mid f\in[1,n] \ \&\& \ f\neq k\}. \quad (13)$$

2.4 自身发生故障

假设 J_k^B 在时刻 t_0 就绪,即对应主版本作业 J_k^P 在时刻 t_0 发生故障.时刻 t_0 越迟,J_k^B 的响应时间也就越迟,而 J_k^P 发生故障的最迟时间不可能晚于系统无故障情况下其最大响应时间 R_k^{NF},因此假设 $t_0=R_k^{NF}$,这对于任务 τ_k 来说是最坏情况.

将 τ_k 的主副版本作业 J_k^P 和 J_k^B 结合为一个虚拟作业 J'_k,J'_k 分成 2 个部分:前一部分优先级为 P_k^P,长度为 C_k^P;后一部分优先级为 P_k^B,长度为 C_k^B.J'_k 的运行过程和 τ_k 的主版本作业 J_k^P 在 R_k^{NF} 时刻出现故障之后 J_k^B 就绪并完成运行的过程相同,因此 J'_k 的最大响应时间 R'_k 就是 J_k^B 的最大响应时间.此时,求 J_k^B 最大响应时间 R_k^B 的问题就转化为求 J'_k 的最大响应时间 R'_k 的问题.

由于 J'_k 在运行过程中有 1 次优先级变化,使得变化前后对其产生干扰的任务集合不同.优先级高于 P_k^P、低于 P_k^B 的主版本(假设序号为 i)在 $[r_k,R_k^{NF}]$ 内对 J'_k 产生干扰,优先级高于 P_k^B 的主版本(假设序号为 j)在 $[r_k,R'_k]$ 内对 J'_k 产生干扰,它们的最大负载和最大干扰负载分别通过式(2)~(6)可以得到.

J'_k 在时间区间 L 内受到的最大累积干扰负载 $S'_k(L)$ 通过式(14)计算.由于 $R'_k\geqslant R_k^{NF}+C_k^B$,在计算过程中 L 的最小取值为 $R_k^{NF}+C_k^B$,因此在式(14)中不考虑 $L<R_k^{NF}$ 的可能,以简化公式方便理解.

$$S'_k(L)=\sum_{P_k^B<P_i^P<P_k^P} I_i^{NC}(R_k^{NF})+\sum_{P_j^P\leqslant P_k^B} I_i^{NC}(L)+\sum_{\substack{\max(m-1)\\ P_k^B<P_i^P<P_k^P\\ P_j^P\leqslant P_k^B}}(F_i(R_k^{NF})\cup F_j(L)), \quad (14)$$

其中,$\sum_{\substack{\max(m-1)\\ P_k^B<P_i^P<P_k^P\\ P_j^P\leqslant P_k^B}}(F_i(R_k^{NF})\cup F_j(L))$ 表示优先级高于 P_k^P 的所有任务的 $F_i(L)$ 项中,前 $m-1$ 大的项的数量和.

J'_k 受到的最大干扰 $I'_k(L)$ 通过式(8)计算.

从 $R'_k(0)=R_k^{NF}+C_k^B$ 开始,迭代计算式(15)直至 $R'_k(n+1)=R'_k(n)$,$R'_k(n)$ 即为虚拟作业 J'_k 的最大响应时间,也就是主版本作业发生故障的情况下 J_k^B 的最大响应时间 R_k^B.

$$R'_k(n+1)=C_k^P+C_k^B+I'_k(R'_k(n)). \quad (15)$$

3 副版本优先级分配

在 FTGS-BPP 中,副版本的优先级会影响整个实时任务集可调度性.高优先级副版本可以在短时间内响应,实时性好,但是副版本优先级过高会给其他高优先级主版本带来额外的干扰负载,而高优先级主版本通常能承受的干扰相对较小,增加额外干扰负载有可能造成主版本不可调度.

本节建立一种启发式的副版本优先级分配算法.假设一个实时任务集的主版本已采用某种优先级分配算法分配优先级,例如 DM(deadline monotonic),DkC 等,并已按照优先级顺序排序,即 $P_1^P=1,P_2^P=2,\cdots,P_n^P=n$.在初始状态下,将副版本的优先级都设置为对应主版本的优先级,并判定整个任务集的可调度性,从最高优先级任务开始,依次检测每个任务副版本的可调度性,对于不可调度的副版本,将其副版本提升一个优先级,并更新再次判定该副版本和所有主版本的可调度性,不需要测试其他副版本的可调度性是因为从第 2 节的讨论中可以看出,副版本的可调度性只取决于高优先级主版本,改变副版本的优先级对其他任务的副版本没有影响.重复这个提升—测试过程直至副版本可调度,如果赋予副版本最高优先级仍不可调度,或者在这个过程中有主版本被判定为不可调度,则优先级分配失败,不能找到能够容错调度该任务集的副版本优先级分配方案.

副版本优先级分配算法的伪代码描述如算法 1 所示,其中 $primary_i$ 和 $backup_i$ 分别表示任务 τ_i 的主、副版本,假设任务集已按照主版本优先级顺序排序,因此任务及其主版本的下标就等于优先级.

算法 1. 副版本优先级分配算法.

```
① for(i=1;i≤n;i++)
②     P_i^B=P_i^P;
③ end for
④ 更新所有任务主副版本的可调度性状态;
⑤ if(有主版本不可调度)
⑥     return 任务集不可调度;
⑦ end if
⑧ for(i=1;i≤n;i++)
⑨     while(backup_i 不可调度)
⑩         P_i^B--;
⑪         if(P_i^B<1)
⑫             return 任务集不可调度;
```

⑬　end if
⑭　更新所有主版本和 $backup_i$ 的可调度性状态;
⑮　if(有主版本不可调度)
⑯　return 任务集不可调度;
⑰　end if
⑱　end while
⑲ end for
⑳ return 任务集可调度.

4　仿　真

仿真实验采用调度大量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 FTGS-BPP 和采用优先级继承策略的全局容错调度算法(FTGS-1)实现容错调度所需的处理器资源.这一方法在硬实时调度研究领域被广泛采用[5,8-9,16-17].m/U 比值反映了调度算法对处理器资源的需求,数值越小表示该调度算法的调度性能越好.FTGS-1 是文献[12]中提出的容错调度算法只考虑一次故障的形式.同时,实验中也计算了不考虑容错的全局调度算法(GS)的 m/U 比值,从而可以比较 2 种容错调度算法为实现容错需要的额外处理器资源.

生成随机任务集的方法如下:首先设置任务集中单个任务的使用率(C/T)上限 a 和任务数量 n,最小释放间隔 T 取[1,500]内的随机值,并假定任务以 T 为周期释放作业,最坏情况运行时间 C 取[1,aT]中的随机值,截止期 $D=T$,任务的副版本简单地规定为主版本的复制.

a 分别取 0.2,0.3,0.4,0.5,分别使用 DkC 和 DM 算法给随机任务集的主版本分配优先级,在[$\lceil U \rceil$,n]之间从低到高搜索使任务集可调度的 m 值(处理器个数),对每个 m 取值运行一次副版本优先级分配算法,如果成功分配优先级则说明在该 m 取值下任务集是可调度的,对每组参数(a 和 n)重复 30 次实验,取平均值作为有效结果.实验结果如图 3～6 所示.

从图 3～6 中可以看出,不论使用 DkC 或 DM 算法分配优先级,FTGS-BPP 算法容错调度随机任务集所需的处理器资源都明显少于 FTGS-1.和 FTGS-1 相比,使用 DkC 算法分配优先级时,FTGS-BPP 最多减少处理器需求 17.7%($a=0.4$,$n=40$),平均减少 11.4%;使用 DM 算法分配优先

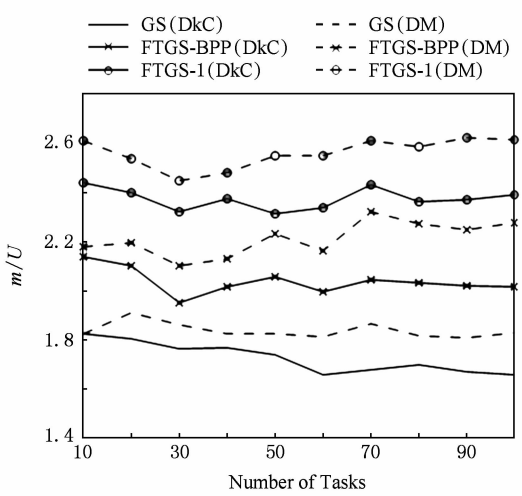

Fig. 3　m/U ratio when $a=0.2$.
图 3　$a=0.2$ 时的 m/U 比值

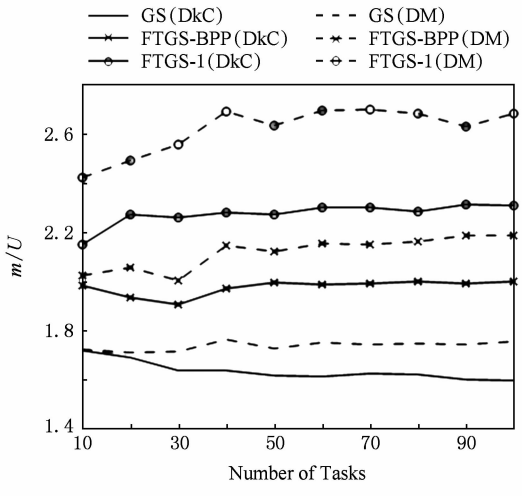

Fig. 4　m/U ratio when $a=0.3$.
图 4　$a=0.3$ 时的 m/U 比值

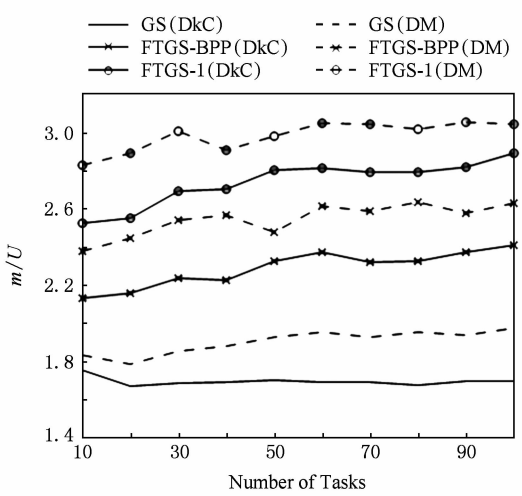

Fig. 5　m/U ratio when $a=0.4$.
图 5　$a=0.4$ 时的 m/U 比值

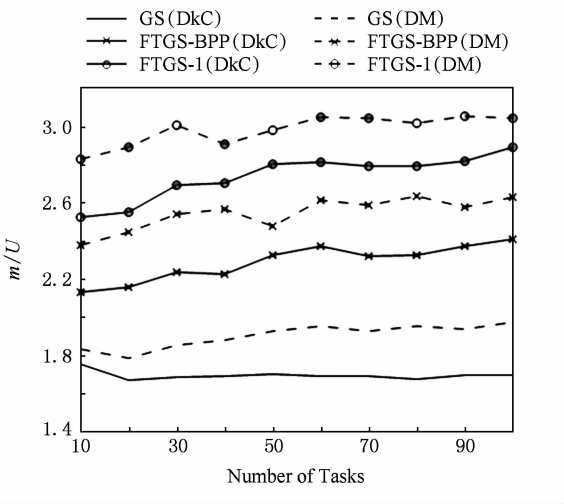

Fig. 6 m/U ratio when $a=0.5$.

图 6 $a=0.5$ 时的 m/U 比值

级时,FTGS-BPP 最多减少处理器需求 21.7%($a=0.3$, $n=30$),平均减少 12.1%. 实验结果说明 FTGS-BPP 通过提升副版本优先级,改善了副版本的实时性,从而减少了为保证副版本不错失截止期而需要的额外处理器资源.

5 总 结

本文基于优先级提升策略,提出副版本优先级可提升的全局容错调度算法 FTGS-BPP,改善了副版本的实时性,使副版本可以在较短的运行时间窗口内依然不会违反截止期约束.通过生成大量随机任务集仿真的方法,对比了 FTGS-BPP 和基于优先级继承策略的全局容错调度算法的调度性能,仿真结果表明,采用不同的优先级分配算法时,FTGS-BPP 都能够有效减少容错带来的额外处理器资源需求.

参 考 文 献

[1] Gaska T, Werner B, Flagg D. Applying virtualization to avionics systems—The integration challenges [C] //Proc of the 29th Digital Avionics Systems Conf. Piscataway, NJ: IEEE, 2010: 5. E. 1-1-5. E. 1-19

[2] Di Natale M, Sangiovanni-Vincentelli A L. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools [J]. Proceedings of the IEEE, 2010, 98(4): 603-620

[3] Adyanthaya S, Geilen M, Basten T, et al. Fast multiprocessor scheduling with fixed task binding of large scale industrial cyber physical systems [C] //Proc of 2013 Euromicro Conf on Digital System Design. Piscataway, NJ: IEEE, 2013: 979-988

[4] Ding Wanfu, Guo Ruifeng, Qin Chengang, et al. A fault-tolerant scheduling algorthm with software fault tolerance in hard real-time systems [J]. Journal of Computer Research and Development, 2011, 48(4): 691-698 (in Chinese)
(丁万夫, 郭锐锋, 秦承刚, 等. 硬实时系统中基于软件容错模型的容错调度算法[J]. 计算机研究与发展, 2011, 48(4): 691-698)

[5] Zhu Ping, Yang Fuming, Tu Gang, et al. Feasible fault-tolerant scheduling algorithm for distributed hard-real-time system [J]. Journal of Software, 2012, 23(4): 1010-1021 (in Chinese)
(朱萍, 阳富民, 涂刚, 等. 一种可行的分布式硬实时容错调度算法[J]. 软件学报, 2012, 23(4): 1010-1021)

[6] Xie Guoqi, Li Renfa, Liu Lin, et al. DAG reliability model and fault-tolerant algorithm for heterogeneous distributed systems [J]. Chinese Journal of Computers, 2013, 36(10): 2019-2032 (in Chinese)
(谢国琪, 李仁发, 刘琳, 等. 异构分布式系统 DAG 可靠性模型与容错算法[J]. 计算机学报, 2013, 36(10): 2019-2032)

[7] Krishna C M. Fault-tolerant scheduling in homogeneous real-time systems [J]. ACM Computing Surveys, 2014, 46(4): 48:1-48:34

[8] Bertossi A A, Mancini L V, Menapace A. Scheduling hard-real-time tasks with backup phasing delay [C] //Proc of the 10th IEEE Int Symp on Distributed Simulation and Real-Time Applications. Piscataway, NJ: IEEE, 2006: 107-118

[9] Chen H M, Luo W, Wang W, et al. A novel real-time fault-tolerant scheduling algorithm based on distributed control systems [C] //Proc of 2011 Int Conf on Computer Science and Service System. Piscataway, NJ: IEEE, 2011: 80-83

[10] Berten V, Goossens J, Jeannot E. A probabilistic approach for fault tolerant multiprocessor real-time scheduling [C] //Proc of the 20th Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2006: 1-10

[11] Pathan R M, Jonsson J. FTGS: Fault-tolerant fixed-priority scheduling on multiprocessors [C] //Proc of the 10th IEEE Int Conf on Trust, Security and Privacy in Computing and Communications. Piscataway, NJ: IEEE, 2011: 1164-1175

[12] Samala A K, Mallb R. Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm [J]. Swarm and Evolutionary Computation, 2014, 14(1): 92-105

[13] Baumann R. Soft errors in advanced computer systems [J]. IEEE Design & Test of Computers, 2005, 22(3): 258-266

[14] Koren I, Krishna C M. Fault-Tolerant Systems [M]. San Francisco, CA: Morgan Kaufmann, 2007

[15] Guan N, Wang Y. Fixed-priority multiprocessor scheduling: Critical instant, response time and utilization bound [C] //Proc of the 26th Parallel and Distributed Processing Symp Workshops & PhD Forum. Piscataway, NJ: IEEE, 2012: 2470-2473

[16] Guan N, Stigge M, Yi W, et al. New response time bounds for fixed priority multiprocessor scheduling [C] //Proc of the 30th IEEE Real-Time Systems Symp. Piscataway, NJ: IEEE, 2009: 387-397

[17] Davis R I, Burns A. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems [J]. Real-Time Systems, 2011, 47(1): 1-40

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

Peng Hao, born in 1984. PhD candidate in Hefei University of Technology. His main research interests include real-time system scheduling and embedded systems (superbinnitu@aliyun.com).

Han Jianghong, born in 1954. Professor and PhD supervisor in Hefei University of Technology. His main research interests include safety-critical industrial systems and embedded systems.

Wei Zhenchun, born in 1978. Associate professor in Hefei University of Technology. His main research interests include internet of things, wireless sensor networks, embedded system and distributed system.

Wei Xing, born in 1980. Associate professor in Hefei University of Technology. His main research interests include internet of things engineering and discrete event dynamic system.