多类型任务集的容错实时调度算法

王 亮，雷 航，熊光泽
(电子科技大学计算机科学与工程学院 成都 610054)

【摘要】针对以往容错实时调度算法只能调度单一的具有容错需求任务的情况，建立了一种单处理器上的容错实时调度模型，并提出了相应的容错实时调度算法。该算法不仅能同时调度具有容错需求和无容错需求的周期实时任务，还可调度随机性非周期任务，其适用范围广泛。

关 键 词 容错技术； 实时调度； 静态预分配； 可靠性
中图分类号 TP302.8 文献标识码 A

Fault-Tolerance Real-Time Scheduling Algorithm for Scheduling Various Tasks

Wang Liang, Lei Hang, Xiong Guangze
(School of Computer Science and Engineering, UEST of China Chengdu 610054)

Abstract Many real-time scheduling algorithms with fault-tolerance, reported in literature, can only schedule tasks with fault-tolerant requirements. The authors present a model of real-time fault-tolerant scheduling in single processor system, and propose a corresponding scheduling algorithm. The algorithm can not only schedule periodic real-time tasks with fault-tolerant requirements and those without fault-tolerant requirements, but also schedule stochastic aperiodic tasks. So its application range is extensive.

Key words fault-tolerant technique; real-time scheduling; static predistribution; reliability

实时安全关键系统要求系统在出现软硬件错误时仍能在规定时限内正确完成，而容错技术是保障实时安全关键系统可靠性的重要手段。文献[1～3]研究了投票技术、反转恢复技术和主/副版本技术，但未涉及实时性能；文献[4]研究了实时调度算法，但未考虑容错问题；文献[5]研究了多处理器系统的容错调度，但其以处理器硬件冗余为基础，不适用于单处理器环境；文献[6]探讨了单处理器容错实时调度，但只能调度单一类型的实时任务，既未考虑同时调度具有容错需求的实时任务和无容错需求的实时任务，也未能同时调度周期任务和非周期任务。本文建立了一个单处理器上的容错实时调度模型，并提出一种基于时间冗余和软件冗余的容错实时调度算法，对多种类型的任务进行调度。

1 容错实时调度模型

本模型中的任务相互独立，任务分为周期实时任务和非周期任务两大类。周期实时任务又分有容错需求和无容错需求两类，本文将有容错需求的周期实时任务称为关键任务。为检测任务执行是否正确，任务结束前将对结果进行正确性判断，这一开销计入运行时间。对于无容错需求的周期实时任务，调度算法只需保证在截止时间前分配其所需的处理器资源，无需考虑与容错相关的问题。

模型采用双版本策略对关键任务进行容错，每个关键任务都有两个相互独立的可执行版本(称为主/副版本)。主版本功能复杂、计算量大、运行结果精度高，执行它能提供高质量的服务，但由于主版本较高的

收稿日期：2003－09－10
基金项目："十五"国防预研基金资助项目(413150401)
作者简介：王 亮(1978－)，男，硕士生，主要从事实时操作系统，软件容错技术方面的研究.

复杂度和较大的资源需求，其执行较易出错。而副版本功能简单、计算量小、运行结果精度不高，执行它仅能提供标准的服务，但可满足用户的基本需求，由于复杂度较低，其执行不易出错。

假设副版本已被充分验证，其运行不会出错，所以本模型针对主版本运行中出现的错误(包括软件错误和偶发性硬件错误)进行容错。

调度开始前，算法为实时任务的副版本预分配处理器资源(执行时间)。调度开始后，先运行主版本。当主版本执行正确，无须运行副版本。当主版本执行出错，副版本投入运行。调度算法必须保证实时任务的主版本(或副版本)在截止时间前完成，为此，若主版本在副版本的最迟开始时间仍未执行完成，则放弃主版本转而执行副版本。为了提高调度性能，算法将利用处理器的空闲时间调度随机性非周期任务。

1.1 模型定义

定义 1　周期实时任务集合有两个子集，$\Phi_{pd}=\Phi_{ft}\cup\Phi_{nf}=\{\tau_1,\tau_2,\cdots,\tau_{n+m}\}$，其中，集合$\Phi_{ft}$包含具有容错需求的周期实时任务，集合$\Phi_{nf}$包含无容错需求的周期实时任务。设$T_i$和$D_i$分别表示任务$\tau_i$的周期和截止时间，假设两者相等即$T_i=D_i$。

设Φ_{pd}中所有任务周期的最小公倍数为超周期T，即$T=\mathrm{LCM}(T_1,T_2,\cdots,T_n,\cdots,T_{n+m})$。周期实时任务的每次执行称为一个实例，在每个$T$中，任务$\tau_i$的第$j$个实例记为$I_{ij}$ $(1\leqslant i\leqslant n+m,\ 1\leqslant j\leqslant(T/T_i))$，$I_{ij}$在$(j-1)T_i$时刻就绪，在本任务下一实例的就绪时间$(jT_i)$前必须完成。由于任务调度过程在每个超周期中重复，所以仅考虑一个T中的调度情况。为了简化问题且不失一般性，以超周期$[0,T]$作为研究对象。

定义 2　有容错需求任务集合(即关键任务集) $\Phi_{ft}=\{\tau_1,\tau_2,\cdots,\tau_n\}$，$\forall\tau_i\in\Phi_{ft}$是一个六元组，即$\tau_i=(T_i,R_i,P_i,B_i,p_i,b_i)$。其中$T_i$为周期，$R_i$为任务优先级，$P_i$和$B_i$分别为$\tau_i$的主版本和副版本，$p_i$和$b_i$分别为$P_i$和$B_i$的计算时间$(p_i\geqslant b_i,\ 1\leqslant i\leqslant n)$。

定义 3　无容错需求任务集合$\Phi_{nf}=\{\tau_{n+1},\tau_{n+2},\cdots,\tau_{n+m}\}$，$\forall\tau_i\in\Phi_{nf}$，由于$\tau_i$仅有一个运行版本，为了与$\Phi_{ft}$中的任务统一起来，将其看作没有主版本而只有副版本的任务，也预分配处理器资源。τ_i用三元组(T_i,B_i,b_i)表示，其中T_i为周期，B_i是τ_i的副版本，b_i是B_i的计算时间。若无特别说明，本文中的副版本均指Φ_{ft}和Φ_{nf}两个集合中的任务副版本。

定义 4　预分配时间段K是指为任务预留的处理器执行时间，任务利用其执行副版本。K由三元组(start, end, τ)表示，其中start和end分别为预分配时间段的开始和结束时间，τ指明该K被分配给哪个任务。一个K只能为某一任务服务，但任务的一次运行可能需要分配若干个K。

$P(i,j)$为任务实例I_{ij}预分配的时间段集合，则

$$P(i,j)=\{K\mid K.\tau=\tau_i\}=\{(x_1,y_1,\tau_i),(x_2,y_2,\tau_i),\cdots,(x_f,y_f,\tau_i)\}\tag{1}$$

式中　$x_1\leqslant y_1\leqslant x_2\leqslant y_2\leqslant\cdots\leqslant x_f\leqslant y_f$。

性质 1　分配给某任务实例的若干个K一定处于其就绪时间和截止时间之间的区域内，即在式(1)中，有$x_1\geqslant(j-1)T_i$，且$y_f\leqslant jT_i$。

性质 2　为某任务实例预分配的时间段总长度等于其副版本的计算时间，即在式(1)中

$$\sum_{r=1}^{f}(y_r-x_r)=\left(\sum_{r=1}^{f}y_r-\sum_{r=1}^{f}x_r\right)=b_i$$

在一个T内，将所有的K按照时间递增的顺序排列，可得预分配时间段表G，即

$$G=k_1\rightarrow k_2\rightarrow\cdots\rightarrow k_q$$

式中　$q=\sum_{i=1}^{m+n}\sum_{j=1}^{T/T_i}\left|p(i,j)\right|$，且有$\forall r\in[1,q]\rightarrow K_r\in\bigcup_{(1\leqslant i\leqslant m+n)\wedge(1\leqslant j\leqslant T/T_i)}P(i,j)$

性质 3　预分配时间段之间不存在交叉关系，即

$$\forall K_i,\ K_j\in G\ \ (i\neq j\rightarrow K_i.\mathrm{end}\leqslant K_j.\mathrm{start}\text{或}\ K_j.\mathrm{end}\leqslant K_i.\mathrm{start})$$

为了提高容错调度算法的性能，在处理器的空闲时间段中调度非周期任务。为了简化问题，假设非周期任务都是非实时任务，算法采用先来先服务原则为其分配处理器资源。

定义 5　设Φ_{ap}表示随机性非周期任务集合，$\Phi_{ap}=\{\tau_{n+1},\tau_{n+2},\cdots,\tau_{n+m}\}$。$\forall\tau_j\in\Phi_{ap}$是一个二元组，$\tau_j=(A_j,C_j)$，其中$A_j$是$\tau_j$的到达时间，$C_j$是$\tau_j$的计算时间。

定义 6 设S为处理机的状态集合，$S=\{S_f, S_p, S_b, S_a\}$，其中S_f表示处理机空闲，S_p和S_b分别表示处理机正在运行周期任务的主版本和副版本，S_a表示处理机正在处理非周期任务。处理器状态变迁过程如图1所示，可见它是一个有限自动机，初始状态为S_f。为了降低模型的复杂度，如图2所示，合并状态S_f与S_a，则$S=\{S_{fa}, S_p, S_b\}$，新的初始状态为S_{fa}，表示处理机空闲或正在处理非周期任务。图1，2中，1表示预分配时间段到达，2表示预分配时间段结束，3表示处理器空闲，4表示Φ_{ft}中的任务就绪，5表示非周期任务就绪，6表示非周期任务运行完成。

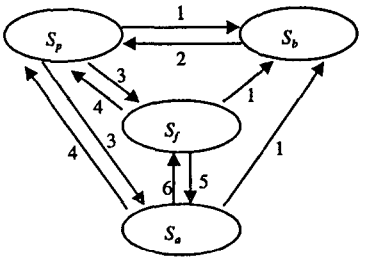

图1 处理机状态变迁图

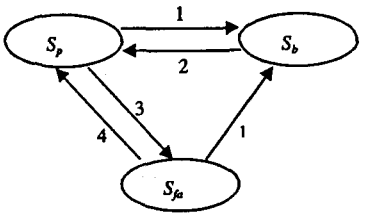

图2 简化的处理器状态变迁图

性质 4 如果一个处理机的状态是S_{fa}，且Φ_{ap}为空，则处理器空闲，即当前没有需要执行的周期任务和非周期任务。

1.2 算法目标及关键技术

任务的正确完成可以通过执行主版本或副版本来实现。相对于副版本，主版本能产生满足用户最大需求的运行结果。但若主版本运行出错，则必须保证副版本在截止时间前完成。可见，调度算法的目标是保证每个实时任务(主版本或副版本)在截止时间前正确执行的前提下，为关键任务的主版本提供尽可能大的运行空间，使尽量多的关键任务能完成主版本。为此，算法采用了预分配的思想，同时，在调度过程中采用了资源回收和资源检查的思想。

预分配思想是指算法为任务的副版本预留处理器时间，以保证在主版本出错时，副版本能在截止时间前完成。为了给主版本提供尽可能大的空间，算法采用最后机会原则进行资源的预分配[7]，即让任务的副版本在可能的最迟时机才开始执行，这一时机称为副版本的最迟开始执行时间，用L表示。

由于主版本执行出错属于小概率事件，预留给副版本的时间会有较大浪费。因此，当主版本运行正确后，算法撤销为其预分配的处理器资源，回收资源以提高处理器资源利用率。当周期任务完成时，节省的资源还可为非周期任务服务，缩短任务的响应时间。

若主版本在L时刻仍未运行完成，则放弃主版本的运行，执行副版本，主版本已经消耗的处理器时间将被浪费。为此，算法在调度主版本前，首先判断处理器的“资源供给”能否满足任务的“资源需求”，只有资源的供给大于需求时，主版本才能被调度，否则，主版本直接放弃运行。为了减少算法的运行开销，具体实现时，资源需求(RR)以主版本的剩余执行时间计算，资源供给(RS)以主版本当前可利用的最大处理器时间来计算(设系统当前时间为T_c)，则

$$RS=\left[(I_{ij}.L-T_c)-\sum_{T_c\le x_r<y_r\le I_{ij}.L}(y_r-x_r)\right] \qquad \forall(x_r,y_r,\tau)\in G$$

式中 RS表示L时刻之前除预分配外的时间段总长度。由于任务在运行主版本的过程中可能被新就绪的高优先级任务抢占，所以，计算的“资源供给”只是可能的最大值，而任务实际能得到的执行时间可能小于该值。由此，任务能够通过资源检查只是其主版本能执行完成的必要条件。

2 容错实时调度算法

按照上面建立的调度模型，本文提出最迟预分配容错实时调度算法LP_FT(Latest Predistribution Fault-Tolerant Real-Time scheduling algorithm)。调度前，算法为$\Phi_{pd}=\{\Phi_{ft}\cup\Phi_{nf}\}$中所有任务的副版本预分配处理器资源，生成$G$表并得到各任务实例的$L$。调度开始后，算法依据系统时间控制任务调度，当某个预分配的

时间段到达，相应任务立即得到处理器的执行权，并利用该段时间执行副版本。关键任务在其周期边界就绪，进入任务就绪队列RQ，并等待调度。在预分配之外的时间段中，算法总是选择RQ中通过了资源检查的最高优先级任务，调度其主版本运行。若有非周期任务到达，则其直接进入Φ_{ap}集合等待调度，当处理器空闲时，按照FCFS原则进行调度，LP_FT算法由以下五部分组成。

2.1 静态预分配算法

多任务调度开始前，系统首先启动执行静态预分配算法。如果输入的任务集可被LP_FT调度，则算法生成相应的G，并计算各任务实例的L。否则，算法输出失败信息。该算法首先将Φ_{pd}中的实时任务按周期长度非减进行排序，并在[T，0]这个反向区域内采用RM算法来进行逆向调度分析，即按照短周期任务优先分配的原则为超周期内的各任务实例分配时间段K。由于逆向调度中的最早完成时间恰好是正向调度中最迟的开始执行时间，所以各任务实例的L随着K的分配也随之产生。静态预分配算法(Predistribution)描述如下（为了简化问题，假设所有周期任务在初始时刻同时就绪）：

输入Φ_{pd}，输出Success，预分配表G。

STEP 1　按周期长度非减对集合Φ_{pd}中的实时任务进行排序，并重新取名为$\tau_1, \tau_2, \cdots, \tau_{n+m}$

STEP 2　G.head.start=0；G.head.end=0；/* 初始化一个空节点 */

STEP 3　FOR i=1 TO $n+m$ DO

STEP 3.1　FOR j= (T/T_i) DOWNTO 1 DO　/* 为I_{ij}分配$P(i,j)$*/

STEP 3.1.1　Finished=false；Deadline=jT_i；ReadyTime=$(j-1)T_i$；C_i=b_i

STEP 3.1.2　IF (Finished=true) THEN goto3.1.8

STEP 3.1.3　从G中选择Deadline时刻之前的最后一个预分配时间段，记为K_l，其满足约束条件 $K_l.\text{end} = \max_{K.\text{end}<\text{Deadlind}} (K.\text{end})$

STEP 3.1.4　Deadline = min(Deadline，K_l.next.start)；/*　K_l.next为K_l的直接后继　*/

STEP 3.1.5　IF (Deadline − K_l.end ≥ C_i)；THEN创建一个新节点K_e；K_e.start=Deadline − C_l，K_e.end=Deadline；将K_e插入到K_l之后；Finished＝true；goto3.1.8

STEP 3.1.6　IF ((Deadline − K_l.end > 0)AND(Deadline − K_l.end < C_i) THEN 创建一个新节K_e；K_e.start=K_l.end，K_e.end=Deadline；将K_e插入到K_l节点之后，Deadline=K_l.start；C_i= C_l − (K_e.end − K_e.start)；goto3.1.2

STEP 3.1.7　IF (Deadline=K_l.end) THEN Deadline=K_l.start；goto3.1.2

STEP 3.1.8　IF (K_e.start≥ReadyTime) THEN $I_{ij}.L$= K_e.start；ELSE Success= false；goto5

STEP 4　G.head从G中出队；/* 删除空结点　*/

STEP 5　Predistribution算法结束

2.2 时钟服务算法

为了保证处理器能够检测每个K的来临，这里引入时钟服务算法。系统每经过一个时钟单位，该算法将作为操作系统中时钟中断服务程序的一部分被启动执行一次。算法将系统当前时间转化为超周期T内的相对时间，若某预分配的时间段来临，则处理器进入S_b状态，进行任务重调度，运行相应任务的副版本。若到达T的初始时刻，则G恢复到(执行预分配算法后的)初始状态。

2.3 任务重调度算法

当需要进行任务切换时，任务重调度算法启动执行。如果处理器状态为S_b，表示某个预分配的时间段已经来临，调度相应任务运行副版本。否则，按照优先级从高到低的顺序遍历就绪队列RQ，判断各任务是否能够通过资源检查。若通过，处理器进入S_p状态，并调度该任务运行其主版本。若不通过，任务从RQ中出队，继续遍历。若遍历结束时仍未找到合适的就绪任务，处理器进入S_{fa}状态。此时，如果等待集Φ_{ap}中有非周期任务等待运行，则按先来先服务原则从Φ_{ap}中选择任务，调度其运行。可见，重调度算法是整个系统的调度核心，集中管理了系统中所有的任务重调度，实现了多任务的切换。任务重调度算法(Dispatch)描述如下：

输入处理器状态state，预分配表G，就绪队列RQ，Φ_{ap}；

输出新的当前运行任务τ_r，处理器状态state，就绪队列RQ，Φ_{ap}。

STEP 1 IF (state=S_b) THEN τ_r=G.head. τ; G.head从G中出队；若τ_r在RQ中，则τ_r出就绪队列(放弃主板本)；立即运行τ_r的副版本；goto7

STEP 2 IF (RQ is empty) THEN state=S_{fa}，goto6；ELSE τ_r=RQ.head；/* 选择RQ中优先级最高的任务 */

STEP 3 对任务τ_r进行资源检查

STEP 4 IF (τ_r通过资源检查) THEN state=S_p；立即运行τ_r的主版本；goto7；ELSE τ_r从RQ中出队

STEP 5 goto2；/* 重复STEP2至STEP4，遍历就绪队列RQ */

STEP 6 IF (Φ_{ap}非空) THEN选择$\tau_r \in \Phi_{ap}$满足$\forall \tau_e \in \Phi_{ap}$ ($A_r \leqslant A_e$)；立即运行非周期任务τ_r

STEP 7 Dispatch算法结束

2.4 动态插入算法

动态插入算法处理周期性关键任务和随机非周期任务，根据到达任务的类型和处理器状态进行相应的操作。必要时，执行任务重调度算法进行任务切换。设动态到达任务τ_i，当前运行任务τ_r，下述三种情况需进行任务重调度：1) 处理器空闲；2) τ_i和τ_r都是关键任务，τ_r的主版本正在运行且其优先级低于τ_i；3) τ_i是关键的周期任务，τ_r是非周期任务。除此之外，算法仅根据τ_i的类型将其插入到相应的等待队列RQ或等待集合Φ_{ap}中即可。

2.5 动态释放算法

当处理器完成某一任务，启动动态释放算法。设任务τ_r执行完成，算法将处理下述三种情况：1) 周期任务τ_r的主版本执行完成，判断运行结果的正确性，若正确，其副版本不需要执行，相应K表项从G中摘除，同时调整G表，使各K的分配始终遵循着最后机会原则；2) 周期任务τ_r的副版本执行完成，判断当前G中最早的K是否来临，若是，运行相应任务的副版本，否则，处理器进入S_p状态，进行任务重调度；3) 非周期任务τ_r运行完成，将τ_r从等待集Φ_{ap}中删除，进行任务重调度，选择合适的后继任务运行。

3 结 束 语

本文建立了一个多类型任务集的调度模型，提出一种容错实时调度算法LP_FT，可调度多种类型的任务，该模型与算法可应用于高可靠强实时多任务系统。同时，在以往对容错实时调度的研究中，任务优先级一般由算法统一安排(如短周期优先)，考虑到系统的实际情况，本算法放宽了这种限制，允许对有容错需求的关键任务自由安排优先级，进一步提高了算法的实用性。

参 考 文 献

[1] Xu Lihao, Bruck J. Deterministic voting in distributed systems using error-correcting codes[J]. IEEE Transactionson Paral-leland Distributed Systems, 1998, 9(8): 813-824

[2] Lin T H, Shin K G. Damage assessment for optimal rollback recovery[J]. IEEE Transactions on Computers, 1998, 47(5): 603-613

[3] Davoli R, Giachini L A, Babaoglu O. Parallel computing in networks of workstations with paralex[J]. IEEE Transactionson Parallel and Distributed Systems, 1996, 7(4): 371-384

[4] Liu C L, Layland J W. Scheduling algorithm for multiprogramming in a hard-real-time environment[J]. Journal of the ACM, 1973, 20(1): 40-61

[5] Ghosh S, Melhem R, Mossé D. Fault-tolerant scheduling on a hard real-time multiprocessor system[M]. In: International Parallel Processing Symposium, 1994

[6] Ghosh S, Melhem R, Mossé D. Fault-tolerant rate-monotonic scheduling[J]. Journal of Real-Time System, 1998, 15(2): 149-181

[7] Chetto H, Chetto M. Some results of the earliest deadline scheduling algorithm[J]. IEEE Trans, Software Eng, 1989, 15(10): 1 261-1 269

编 辑 徐培红