软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(4):1010−1021 [doi: 10.3724/SP.J.1001.2012.04004]

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

一种可行的分布式硬实时容错调度算法*

朱 萍[1], 阳富民[2+], 涂 刚[2], 张 杰[2], 周正勇[2]

[1](武汉纺织大学 数学与计算机学院,湖北 武汉 430073)

[2](华中科技大学 计算机科学与技术学院,湖北 武汉 430074)

Feasible Fault-Tolerant Scheduling Algorithm for Distributed Hard-Real-Time System

ZHU Ping[1], YANG Fu-Min[2+], TU Gang[2], ZHANG Jie[2], ZHOU Zheng-Yong[2]

[1](College of Mathematics and Computer Science, Wuhan Textile University, Wuhan 430073, China)

[2](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

+ Corresponding author: E-mail: yangfm@routon.com

Zhu P, Yang FM, Tu G, Zhang J, Zhou ZY. Feasible fault-tolerant scheduling algorithm for distributed hard-real-time system. *Journal of Software*, 2012,23(4):1010−1021. http://www.jos.org.cn/1000-9825/4004.htm

Abstract: In distributed hard-real-time systems, when a hardware failure occurs, the task instance in current period is usually more urgent than the subsequent ones. According to this, a novel strategy of delay in non-urgent period (referred to as DNUP) is proposed. DNUP strategy can postpone the execution of non-urgent instance as late as possible and reserve the slack time for the instance with low priority. Thus it has a better chance to complete its execution in an urgent period. Extensive simulations reveal that DNUP can improve the schedulability of periodic tasks and achieve a remarkable saving on the number of processors required with respect to several well-known fault-tolerant scheduling algorithms.

Key words: real-time; distributed system; scheduling; fault-tolerance; priority

摘 要: 针对分布式硬实时系统发生处理机故障后,当前周期内的任务实例和后续实例相对截止期限的不同紧迫程度,提出非紧迫周期内延迟策略——DNUP(delay in non-urgent period).该策略能够尽可能地推迟非紧迫实例的执行,使得低优先级实例有更多的机会完成其紧迫周期内的执行,从而实现处理器空闲(slack)资源的合理挪动.仿真实验结果表明,与其他几个著名的分布式容错调度算法相比,DNUP 策略能够提高任务的可调度性,从而有效减少了所需处理机的数目.

关键词: 实时;分布式系统;调度;容错;优先级

中图法分类号: TP316 **文献标识码**: A

分布式实时系统目前被广泛地应用于航天、军事等关键领域,这些领域不仅需要保证任务的强实时性,而且要求在局部处理机出错的情况下任务仍能在其截止期限之前正确完成,否则将会发生灾难性的后果.容错调度算法是一种在实时系统发生故障时通过软件实现容错的有效方法,不需要额外的硬件代价,是硬实时分布式系统的一个重要研究分支.

* 基金项目: 国家自然科学基金(60603032)
收稿时间: 2010-06-18; 修改时间: 2010-12-09; 定稿时间: 2011-02-17

分布式硬实时系统主要使用主/副版本(primary/backup,简称 PB)机制[1−4]来实现容错,即每个任务有两个副本分别分配到不同处理机上:主版本(primary copy)和副版本(backup copy),并且要求在发生处理机故障的情况下也能保证任务至少有一个副本在截止期限之前完成.

基于 PB 容错机制,文献[5]提出了 BKCL 算法,文献[6]提出了基于 EDF 的容错调度算法,但它们都要求副版本必须在主版本失效以后才能开始运行.

文献[7]提出的 FTRMFF 算法打破了副版本释放时间限制,定义了副版本的两种类型:主动副版本(active backup)和被动副版本(passive backup),主动副版本与主版本同时释放,而被动副版本只有当其主版本任务失效后才释放.显然,主动副版本具有比被动副版本更长时间的执行窗口,但在无错情况下会形成双倍冗余.分布式硬实时调度算法性能的衡量标准通常形式化为在保证实时性和可靠性的基础上所需处理机的个数,很多学者在 FTRMFF 算法的基础上,以减少处理机数目为目标提出了改进算法.

文献[8]提出的 Tecros 算法通过在任务主版本正确结束后立刻中止其主动副版本的执行来减少冗余,从而减少处理机数目.

文献[9]在 Tecros 算法的基础上提出 DABCBF 算法,尽量延迟主动副版本的执行,以最小化主动副版本的实际执行时间.

文献[10]提出的 ARR 策略与 DABCBF 算法类似,通过推迟主动副版本的相位达到主版本和主动副版本之间重叠区域最小的目的,并及时回收无错情况下主动副版本所占的处理器资源方式来减少所需处理机个数,提高调度性能.这些改进算法是按照 RM 优先级递减的顺序来为任务分配处理机.

文献[11]在 ARR 策略的基础上提出了 TPFTRM 算法,该算法根据负载将任务分组,并且采用了基于 S 优先级[12]的任务分配方法.基于 S 优先级的任务分配可将具有相同或倍数周期关系的任务分配到同一个处理机上,从而产生更紧凑的调度,但这种任务分配方式会大大增加时间复杂度.

上述算法重点考虑减少无故障发生情况下主动副版本与主版本之间的冗余,本文则在 ARR 的基础上进一步考虑在处理机发生故障情况下如何提高调度性能.

在故障发生后,其他非故障处理机上需要新增若干个被动副版本以屏蔽故障对分布式系统带来的影响.我们将任务从就绪到截止时限之间的时间段定义为相对截止时限,那么对副版本而言,发生故障时运行的实例比后续实例具有更紧迫的相对截至时限.

为便于描述,我们将故障时刻所在周期称为紧迫周期,紧迫周期内运行的实例称为紧迫实例,后续周期称为非紧迫周期,非紧迫周期内的实例称为非紧迫实例.

如果能够尽可能地推迟非紧迫实例的执行,就可以腾出处理器空闲(slack)供其他紧迫实例使用.这种延迟方法实质上实现了将处理机空闲从非紧迫周期向前挪到紧迫周期,我们称其为非紧迫周期延时策略(delay in non- urgent period,简称 DNUP).DNUP 策略通过合理挪动处理机空闲,使更多的紧迫实例有机会在截止期限之前完成,从而提高了任务的可调度性.

本文第 1 节给出符号、容错模型和调度问题的形式化定义.第 2 节通过图例方式提出待研究的问题.第 3 节详细介绍 DNUP 策略,重点分析非紧迫周期的最大响应时间和紧迫周期的最大延迟时间.第 4 节给出基于 DNUP 策略的 3 种算法:任务分配、任务恢复和故障恢复算法.第 5 节对提出的调度算法及相关算法进行实验与评价.第 6 节总结全文,并指出下一步研究目标.

1 实时容错模型

本文研究的周期任务 τ_i 定义为四元组:

$$\tau_i=(R_i,T_i,C_i,D_i),$$

其中,R_i 为释放时刻,T_i 为周期,C_i 为最大执行时间,D_i 为截至期限.

每个周期任务都是无穷个实例序列集合,第 k 个实例在时刻 $R_i+(k-1)T_i$ 释放,需要在时刻 $R_i+(k-1)T_i+D_i$ 之前执行 C_i 时间单元.任务第 1 个实例的释放时间也称为相位(phasing).假定任务的截止期限等于任务周期,即 $D_i=T_i$.

算法调度对象为包含 n 个硬实时周期任务的集合,表示为$\Gamma=\{\tau_1,\tau_2,\ldots,\tau_{2i-1},\tau_{2i},\ldots,\tau_{2n-1},\tau_{2n}\}$,其中,$\tau_{2i-1}$和$\tau_{2i}$分别表示第 i 个任务的主版本和副版本.根据 RM 算法为主版本任务分配优先级,周期越小,优先级越高,副版本具有与主版本相同的优先级.

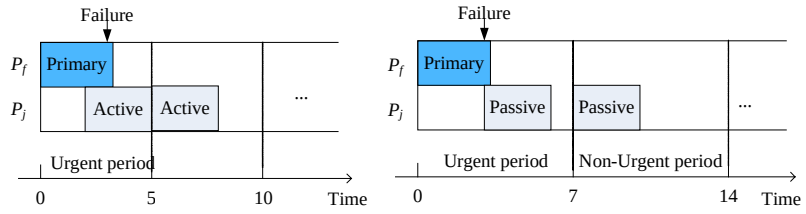
ARR 策略的重点是计算主动副版本任务相位的最大延迟时间.为了方便紧迫周期和非紧迫周期的形式化表述,调度模型将所有类型任务的相位统一为 0,通过下面描述的方式来实现与相位延迟等同的效果.在模型中,任务有 4 个状态:释放、就绪、运行和结束.每个实例释放后进入释放态,通过各自指定的时间段后进入就绪态,这个时间段我们定义为释放就绪延迟,用符号 DRR(delay from release to ready)表示.只有处于就绪态的实例才可以被处理器调度,在处理器调度过程中为运行态,当被高优先级抢夺处理器则回到就绪态,全部完成后则为完成态.如果将主版本的 DRR 设置为 0,主动副版本设置为 ARR 策略计算的最大延迟相位,被动副版本设置为其主版本无错情况下的最坏响应时间,则可以实现 ARR 策略.基于任务调度模型,紧迫周期和非紧迫周期的形式化定义如下.

定义 1(紧迫周期、非紧迫周期). 假设处理机 P_f在时刻θ发生错误,其他处理机 $P_j,j\neq f$ 上的任务τ_i(主或副版本),如果满足$(k-1)T_i<\theta\leqslant kT_i$,则时间区间$[(k-1)T_i,kT_i]$称为任务 τ_i 的紧迫周期(urgent period),其后的区间$[mT_i,(m+1)T_i]$,$m\geqslant k$ 称为非紧迫周期(non-urgent period).

与文献[7–10]中的单处理机故障模型相同,本文假定,当某个处理机发生故障后,其他处理机能够立刻对失效处理机上还未完成的任务进行容错调度,同时还假定分布式系统在任意时刻只有单个处理机故障.也就是说,从某个处理机发生故障到该故障修复期间,不再会有其他处理机故障.

2 问题的提出

在 ARR 策略中,主版本的相对截止时限始终与周期一致,而副版本从初始时刻到紧迫周期时间区间内,相对截至时限都小于其周期,在非紧迫周期内等于其周期.为便于理解,我们用图 1 表示 ARR 策略在发生故障后运行在非故障处理机上的副版本的调度情况.具体考虑处理机 P_f 上主版本在即将完成时刻θ失效,其分配在处理机 $P_j,j\neq f$ 上的副版本在无高优先级任务抢占情况下的执行情况.



(a) 主动副版本 (b) 被动副版本

Fig.1 Scheduling of backup copy when its primary copy fails in ARR

图 1 ARR 策略中,主版本失效后其对应副版本的调度情况

从上述调度情况来看,在紧迫周期内,任务可用的剩余时间只有$(T_i-\theta)$时间单元,如果是被动副版本,则立即启动;如果是主动副版本,则继续执行,以保证任务能在截止期限之前完成.在满足出错后的第 1 个周期请求后,任务进入非紧迫周期,可用的时间延长到 T_i 时间单元.因此,对同一个任务而言,处理器在其紧迫周期内,相对于非紧迫周期而言更加繁忙.如果能将所有类型任务在非紧迫周期内的执行适当推迟,就可以为其他较低优先级的任务在紧迫周期内的执行腾出处理器资源.基于该思想,我们提出 DNUP 策略,具体包括两部分:① 在紧迫周期内,任务相对截止期限较短,在故障时刻立即将 DRR 值设为 0,使任务能够尽可能地满足截止期限;② 在非紧迫周期,任务相对截止期限充足,在保证实时性的前提下,DRR 值设置为尽可能大,以便腾出更多的处理器资源供其他紧迫周期任务使用.

针对同一个例子,DNUP 策略在发生故障后的非故障处理机上各类型任务的调度情况如图 2 所示.在满足

出错后的第 1 个周期请求后,任务进入非紧迫周期,就绪时刻最多可推迟(T_i–C_i)时间单元而不破坏截止期限.

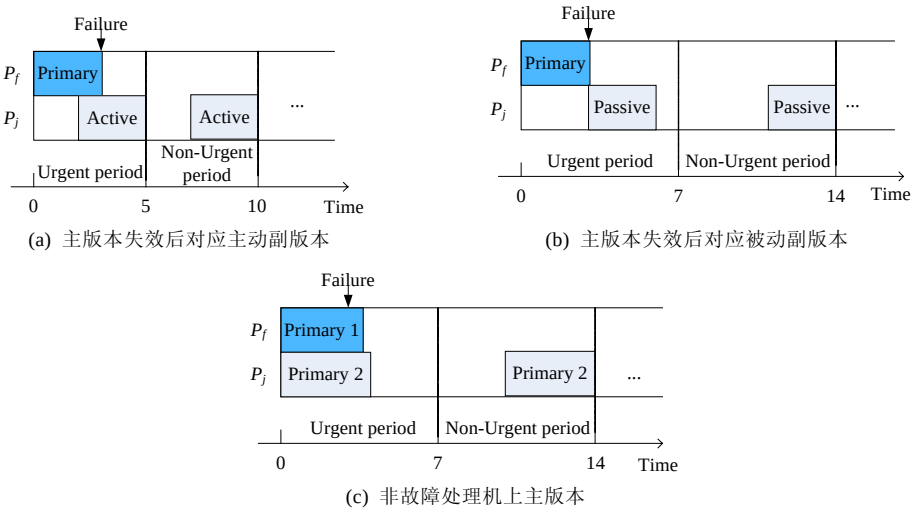


Fig.2 Scheduling in DNUP

图 2 DNUP 策略中各类型任务的调度情况

3 DNUP 策略

文献[13]提出的完备时间测试 CTT(completion time test)通过计算任务最大响应时间来进行单处理机上任务的可调度性判定,FTRMFF 算法将 CTT 扩展为容错完备时间测试 FTCTT(fault-tolerant CTT),实现具有容错能力的分布式系统上任务可调度性判定.FTCTT 包含 NoFaultCTT 和 OneFaultCTT,分别用于测试在无错误发生和单个处理机故障的情况下,任务在处理机上的可调度性.DNUP 策略中的 FTCTT 可形式化为:

- *NoFaultCTT*(τ_i,P_j):判定任务 τ_i(主版本或主动副版本)在无故障发生情况下能否分配到处理机 P_j 上,具体判定方法与 ARR 策略同,这里不再赘述;
- *OneFaultCTT*(τ_i,P_j,P_f):假设处理机 P_f 在时刻 θ 发生硬件故障,判定任务 τ_i(主版本或副版本)能否分配到处理机 P_j,$j\neq f$,需要考察下面两个条件:

  (C1) 在紧迫周期内,任务是否可以满足相对截止期限;

  (C2) 在非紧迫周期内,在保证实时性的前提下,任务的 DRR 值是否大于等于 0.

如果(C1)和(C2)同时满足,表明在 P_f 出错情况下,任务 τ_i 在处理机 P_j 上是可调度的.

如果每个任务在非紧迫周期内 DRR 都为 0,则 DNUP 调度与 ARR 调度完全一致.也就是说,ARR 可以看成是 DNUP 的一种特例.所以,DNUP 策略的核心问题是计算任务在紧迫周期内的最大响应时间和在非紧迫周期内的最大延迟时间.只有当满足最大响应时间不超过相对截止期限并且推迟时间不小于 0 的条件时,才能通过在出错情况下的调度测试.为方便描述,定义以下符号:

(1) $P(\tau_i)$:任务 τ_i 所在的处理机;

(2) *primary*(P_j):处理机 P_j 上的主版本任务集合;

(3) *active*(P_j):处理机 P_j 上主动副版本任务集合;

(4) *activeRecover*(P_j,P_f):主版本任务分配在处理机 P_f 上而自身分配在处理机 P_j 的主动副版本任务集合;

(5) *passiveRecover*(P_j,P_f):主版本任务分配在处理机 P_f 上而自身分配在处理机 P_j 的被动副版本任务集合;

(6) *recover*(P_j,P_f):*activeRecover*(P_j,P_f)$\cup$*passiveRecover*(P_j,P_f);

(7) *hp*(j,f,i):任务集 σ=*primary*(P_j)$\cup$*recover*(P_j,P_f)中优先级高于 τ_i 的所有任务;

(8) $W_{nf}(i,j)$:在无故障情况下,任务 τ_i(主/主动副版本)在处理机 P_j 上的最大响应时间;

(9) $W_{of}(i,j,f)$:在处理机 P_f 故障情况下,任务 τ_i 在处理机 P_j 上的最大响应时间;

(10) $d\max_i(Init)$:任务 τ_i 在初始时刻最大可延迟时间;

(11) $\mathrm{dmax}_i(NU)$:任务 τ_i 在非紧迫周期内最大可延迟时间;

(12) B_i:任务 τ_i 的恢复时间,可表示为 $B_i=T_i-d\max_i(Init)$.

DNUP 策略需要离线确定每个任务 τ_i 可调度的处理机 P_j 以及计算在该处理机上初始时刻和非紧迫周期内的最大可延迟时间,即 $d\max_i(Init)$和 $d\max_i(NU)$的值.主版本和主动副版本无故障情况下的最大响应时间和主动副版本在初始时刻的最大延迟时间可以通过 $NoFaultCTT(\tau_i,P_j)$计算得到.显然,在 DNUP 策略中,主版本初始时刻的最大延迟为 0,被动副版本为其对应主版本在无故障情况下的最大响应时间.下面,我们来详细分析故障情况下的任务最大响应时间,在此基础上,对紧迫/非紧迫周期内任务的可调度性进行分析,并给出其非紧迫周期内的最大可延迟时间.

3.1 最大响应时间分析

DNUP 算法属于固定优先级抢占算法,所以计算任务的最大响应时间需要考虑自身的执行以及高优先级任务的抢占情况.而且,高优先级任务会同时出现紧迫和非紧迫周期两种执行方式.在故障情况下,任务的最大响应时间可由定理 1 表示.

定理 1. 假定处理机 P_f 在时刻 θ 发生故障,在其他任意处理机 P_j 上任意任务 τ_i 的最大响应时间可以表示为

$$W_{of}(i,j,f)=C_i+\sum_{\tau_{2k-1}\in hp(j,f,i)}\left(C_{2k-1}+C_{2k-1}\times f\left(\frac{W_{of}(i,j,f)-T_{2k-1}}{T_{2k-1}}\right)+\phi 1(2k-1,W_{of}(i,j,f)-T_{2k-1})\right)+\sum_{\tau_{2k+2}\in hp(j,f,i)}\phi 2(2k,W_{of}(i,j,f)) \tag{1}$$

其中,

$$f(x)=\begin{cases}0, & x\leqslant 0\\ \lfloor x\rfloor, & x>0\end{cases},$$

$$\phi 1(k,t)=\begin{cases}0, & \text{if } d\max_k(NU)\geqslant t\%T_k\\ \min\{C_k,(t\%T_k-d\max_k(NU))\}, & \text{else}\end{cases},$$

$$\phi 2(k,t)=\begin{cases}C_k, & \text{if } t\leqslant B_k\\ C_k+C_k\times f\left(\frac{t-B_k}{T_k}\right)+\phi 1(k,t-B_k), & \text{else}\end{cases}.$$

证明:由文献[14]中定理 1 可知,固定优先级调度中,在关键时刻(critical instance)任务的响应时间最大,即与高优先级任务同时就绪的时刻.具体到 DNUP 策略,在单处理机故障情况下,任务 τ_i 的最坏情况发生在高优先级任务都处于紧迫周期且与任务 τ_i 同时进入就绪态.公式(1)中,等号右边的第 1 部分表示任务 τ_i 自身的工作时间(work time),第 2 和第 3 部分分别是高优先级的主版本任务和副版本任务的累积工作时间.单独看第 2 部分,对于高优先级的主版本任务 τ_{2k-1} 而言,第 1 个 C_{2k-1} 是其在紧迫周期内的执行时间,在后面大小为$(W_{of}(i,j,f)-T_{2k-1})$的时间段中处于非紧迫周期.在这些非紧迫周期中,任务的 DRR 设置为 $d\max_{2k-1}(NU)$,即延迟 $d\max_{2k-1}(NU)$时间单元进入就绪状态.$(W_{of}(i,j,f)-T_{2k-1})$时间段内包含 $f\left(\frac{W_{of}(i,j,f)-T_{2k-1}}{T_{2k-1}}\right)$ 个完整周期.最后一个非完整周期的长度为$(W_{of}(i,j,f)-T_{2k-1})\%T_{2k-1}$ 时间单元,有两种可能情况,如图 3 所示.

对于情况(a),τ_{2k-1} 不抢占处理器资源,工作时间为 0;对于情况(b),τ_{2k-1} 的工作时间为其最大执行时间 C_{2k-1} 和$((W_{of}(i,j,f)-T_{2k-1})\%T_{2k-1}-d\max_{2k-1}(NU))$中的较小者.单独看第 3 部分,高优先级副版本 τ_{2k} 紧迫周期内的相对截至期限小于 T_{2k} 的最坏情况为 B_{2k}.如果 $W_{of}(i,j,f)\leqslant B_{2k}$,则 τ_{2k} 始终处于紧迫周期,其工作时间为其最大执行时间 C_{2k};反之,τ_{2k} 在大小为$(W_{of}(i,j,f)-B_{2k})$的时间段内处于非紧迫周期.与主版本相同,在非紧迫周期中,副版本 DRR 设置为 $d\max_{2k}(NU)$且工作时间分析与主版本完全相同,这里不再赘述. □

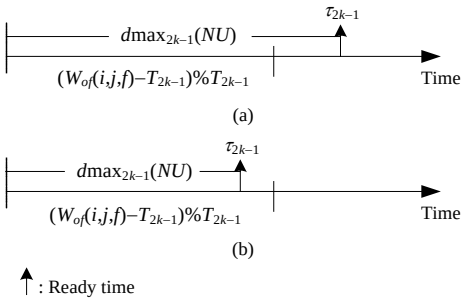


Fig.3 Two cases in the last non-urgent period

图 3 在最后一个非紧迫周期内的两种情况

3.2 任务可调度性分析

任务在紧迫周期内和非紧迫周期内的可调度性判定,以及非紧迫周期内的最大可延迟时间,分别由定理 2 和定理 3 给出.

定理 2. 假定处理机 P_f 在时刻θ发生故障,考虑其他任意处理机 $P_j,j\neq f$ 上任意任务τ_i,如果满足 $W_{of}(i,j,f)\leqslant T_i-d\max_i(Init)$,那么$\tau_i$的紧迫实例是可调度的.

证明:对于强实时系统而言,判定实时性需要确定在最坏情况下,任务的最大响应时间是否超过其相对截止期限.尽管 DNUP 策略会在故障时刻将任务的 DRR 值设置为 0,但还需要考虑在设置之前任务已经延迟的时间.最坏情况是,任务τ_i在紧迫周期内的最大延迟为初始时刻的 DRR 值,即 $d\max_i(Init)$.所以最坏情况下,τ_i的相对截止期限表示为 $T_i-d\max_i(Init)$. □

定理 3. 假定处理机 P_f在时刻θ发生故障,对于其他处理机 $P_j,j\neq f$ 上的任务τ_i,如果满足 $W_{of}(i,j,f)\leqslant T_i$,那么$\tau_i$非紧迫周期内是可调度的,且最大可延迟时间可表示为 $d\max_i(NU)=T_i-W_{of}(i,j,f)$.

证明:假定任务τ_i 在非紧迫周期内的延迟时间为 d,那么在最坏情况下,任务从释放到完成所需时间为 $(d+W_{of}(i,j,f))$,只要满足 $d+W_{of}(i,j,f)\leqslant T_i$,$\tau_i$的非紧迫实例就可以在截止期限之前完成.由于最大延迟时间必定满足 $d\geqslant 0$,所以满足 $W_{of}(i,j,f)\leqslant T_i$的任务必定满足实时性,且最大延迟表示为 $d\max_i(NU)=T_i-W_{of}(i,j,f)$. □

4 基于 DNUP 策略的 3 种算法

前面我们对 DNUP 策略的可调度性进行了分析,本节将进一步设计基于 DNUP 策略的具体算法:任务分配算法、任务恢复算法和故障恢复算法.任务分配算法通过离线方式将任务安排到满足可调度性的处理机上.任务分配算法会产生任务分配方案,计算每个主/副版本在初始时刻、紧迫周期和非紧迫周期的最大可延迟时间.根据释放就绪延迟 DRR 的定义,在实际调度过程中,每个任务释放后都需要经过当前 DRR 大小的时间延迟才能进入就绪态.在真正调度的初始时刻,任务的 DRR 设置为分配算法已计算好的初始时刻最大可延迟时间.在每个处理机上,系统根据 RM 算法调度就绪的主版本和主动副版本集合.主版本任务正确完成后给对应的副版本任务发送“正确执行完毕”的信息.如果副版本收到此信息后,则强制进入完成态,如果检测到某处理机发生了硬件故障,则立刻启动任务恢复算法.任务恢复算法安排出错处理机上未完成的任务能在其他正常处理机上恢复执行,保证所有任务至少有一个版本能在截止期限之前完成.当故障处理机修复之后,立即启动故障恢复算法,它能将修复好的处理机重新加入到分布式系统中,在不破坏任务集实时性的前提下,能够逐渐让各处理机调度故障前原本调度的任务集,保持系统健壮性.

4.1 任务分配算法

任务分配算法是按照 RM 优先级递减顺序,依次将每个任务的主/副版本分配到第 1 个符合调度条件的处理机上.在判定主版本是否能分配到某个处理机时,需要判定在无故障情况下和其他任意单处理机故障情况下,任务是否满足可调度性;在判定主动副版本是否能分配到某个处理机时,需要判定在无故障情况下和其主版本

所在的处理机故障情况下,任务是否满足可调度性;在判定被动副版本是否能分配到某个处理机时,需要判定在其主版本所在的处理机故障情况下,任务是否满足可调度性.DNUP 任务分配算法描述如下:

算法 1. DNUP 任务分配算法.

输入:按照 RM 优先级递减排序的硬实时周期任务的主副版本集和$\Gamma=\{\tau_1,\tau_2,\ldots,\tau_{2n-1},\tau_{2n}\}$;

输出:任务分配方案及所需处理机个数.

(0) 设置当前处理机个数 m=1.

(1) 依次为每个任务的主副版本分配处理机(for i=1,...,n):

(1.1) 依次对每个处理机进行判定(for j=1,…,m):

(1.1.1) $NoFaultCTT(\tau_{2i-1},P_j)$测试无故障情况下任务 τ_{2i-1} 在处理机 P_j 上的调度性,计算 $W_{nf}(2i-1,j)$;

(1.1.2) $OneFaultCTT(\tau_{2i-1},P_j,P_f),P_f(f=1,\ldots,m,f\neq j)$测试处理机 P_f 故障情况下,τ_{2i-1} 在处理机 P_j 上的可调度性,计算 $W_{of}(i,j,f)$,且 $d\max_{2i-1}(NU)=\min\{T_i-W_{of}(i,j,f),f=1,\ldots,m,f\neq j\}$;

(1.1.3) 如果 $W_{nf}(i,j)\leqslant T_{2i-1},W_{of}(i,j,f)\leqslant T_{2i-1}$ 且 $d\max_{2i-1}(NU)\geqslant 0$,将任务 τ_{2i-1} 分配到处理机 P_j 上,并设置 $d\max_{2i-1}(Init)=d\max_{2i-1}(U)=0$, break;

(1.2) 如果 m 个处理机中没有符合调度条件的处理机,则新增一个处理机 $m=m+1$,将 τ_{2i-1} 分配到处理机 P_m 上,并设置 $W_{nf}(2i-1,m)=C_{2i-1}$,$d\max_{2i-1}(Init)=d\max_{2i-1}(U)=0$ 和 $d\max_{2i-1}(NU)=T_{2i-1}-C_{2i-1}$;

(1.3) 如果满足 $T_{2i-1}-W_{nf}(2i-1,j)<C_{2i-1},P_j=P(\tau_{2i-1})$,副版本 τ_{2i} 的类型为主动,否则为被动.对于被动副版本,设置 $d\max_{2i}(Init)=W_{nf}(2i-1,j)$;

(1.4) 如果 τ_{2i} 为主动副版本,依次对每个处理机进行判定(for j=1,…,m):

(1.4.1) $NoFaultCTT(\tau_{2i},P_j)$测试无故障情况下,任务 τ_{2i} 在处理机 P_j 上的调度性,计算 $W_{nf}(2i,j)$和 $d\max_{2i}(Init)$;

(1.4.2) $OneFaultCTT(\tau_{2i},P_j,P(\tau_{2i-1}))$测试处理机 $P(\tau_{2i-1})$故障情况下,τ_{2i} 在处理机 P_j 上的可调度性,计算 $W_{of}(i,j,f),P_f=P(\tau_{2i-1})$和 $d\max_{2i}(NU)$;

(1.4.3) 如果 $W_{nf}(2i,j)\leqslant T_{2i},W_{of}(i,j,f)\leqslant T_{2i}-d\max_{2i}(Init)$且 $d\max_{2i}(NU)\geqslant 0$,将任务 τ_{2i} 分配到处理机 P_j 上,并设置 $d\max_{2i-1}(U)=0$, break;

(1.5) 如果 τ_{2i} 为被动副版本,依次对每个处理机进行判定(for j=1,…,m):

(1.5.1) $OneFaultCTT(\tau_{2i},P_j,P(\tau_{2i-1}))$测试处理机 $P(\tau_{2i-1})$故障情况下,τ_{2i} 在处理机 P_j 上的可调度性,计算 $W_{of}(i,j,f),P_f=P(\tau_{2i-1})$和 $d\max_{2i}(NU)$;

(1.5.2) 如果 $W_{of}(i,j,f)\leqslant T_{2i}-d\max_{2i}(Init)$且 $d\max_{2i-1}(NU)\geqslant 0$,则将任务 τ_{2i} 分配到处理机 P_j 上,并设置 $d\max_{2i}(U)=0$, break;

(1.6) 如果 m 个处理机中没有符合调度条件的处理机,则新增一个处理机 $m=m+1$,将 τ_{2i} 分配到处理机 P_m 上,并设置 $d\max_{2i}(U)=0$ 和 $d\max_{2i}(NU)=T_{2i-1}-C_{2i-1}$.如果 τ_{2i} 是主动副版本,设置 $W_{nf}(2i,m)=C_{2i}$ 和 $d\max_{2i}(Init)=T_{2i-1}-C_{2i-1}$;

(2) 返回分配方案和 m 值.

4.2 任务恢复算法

在调度过程中,假设有处理机发生硬件故障,需要立刻启动任务恢复算法.基于单处理机故障假设,对于故障处理机上主版本对应的被动副版本所在的非故障处理机,需要立即调度这些被动副版本而取消其他被动副版本任务.同时,根据 DNUP 策略,在故障发生时刻,所有任务即刻进入紧迫周期,将 DRR 值设置为 0.当紧迫周期结束后,将任务 DRR 设置为算法 1 中计算好的非紧迫周期最大延迟时间.对于所有没有分配故障处理机上主版本对应的被动副版本的非故障处理机,则像无故障发生一样继续运行.前面的可调度性判定和单处理机故障假设可以保证任务恢复算法的正确性.DNUP 任务恢复算法描述如下:

算法 2. DNUP 任务恢复算法.

输入:DNUP 任务分配方案、处理机个数 m、故障处理机 P_f,故障时刻 θ.

(1) 依次对每个满足 $1\leqslant j\leqslant m$ 和 $j\neq f$ 的处理机 P_j 执行如下操作:

(1.1) 如果 $passiveRecover(P_j,P_f)\neq\varnothing$,则:

(1.1.1) 调度对象由任务集 $primary(P_j)\cup active(P_j)$变为 $primary(P_j)\cup recover(P_j,P_f)$;

(1.1.2) 时刻θ,将 $primary(P_j)\cup recover(P_j,P_f)$中每个任务$\tau_i$的 DRR 值设为 $d\max_i(U)$;

(1.1.3) 时刻 $kT_i((k-1)T_i<\theta\leqslant kT_i)$,将 $primary(P_j)\cup recover(P_j,P_f)$中每个任务$\tau_i$ 的 DRR 值设为 $d\max_i(NU)$;

(1.2) 如果 $passiveRecover(P_j,P_f)=\varnothing$,则继续调度任务集 $primary(P_j)\cup active(P_j)$.

4.3 故障恢复算法

在调度过程中,假设故障处理机 P_f 在时刻θ修复,需要立刻启动执行故障恢复算法,将其加入到分布式系统中,使得系统能够恢复到故障发生前的运行状态,且该过程中不会破坏任何任务的实时性.我们将所有任务都恢复到就像无故障发生的运行状态的时刻定义为故障恢复时刻,同时,在调度模型中已假设某个处理机发生故障到该故障修复之间不再会有其他处理机故障.因此,故障恢复算法需要保证在这一过程中所有任务在每个请求周期内至少有一个副本能够正确执行完毕.

对于故障时刻进行任务切换的非故障处理机,将还未完成的副版本只执行一次将其移除.同时,在处理机 P_f 上,将原本分配在其上的主版本和主动副版本在时刻θ开始后的各自最近的周期开始时刻释放,并将 DRR 设置为初始时刻的最大可延迟时间.这样能保证处理机 P_f 上任务的实时性,并且能让 P_f 逐步恢复到故障前的调度状态.对于故障时刻进行任务切换的非故障处理机,在移除完最后一个副版本后,处理机 P_j 只剩下主版本,按照优先级递增顺序,依次将每个主版本在下一个周期开始时刻将 DRR 值设置为 0.如果在高优先级任务不变的情况下任务的 DRR 变小,那么相对截止期限将变大.所以,每个主版本在 DRR 值的变化过程中保持实时性.在完成了所有主版本 DRR 设置后,将原本分配在其上的主动副版本在下一周期开始时刻释放,并将 DRR 设置为初始时刻的最大可延迟时间.至此,该处理机上所有任务的执行方式与无故障发生时的完全相同.对于故障时刻未进行任务切换的非故障处理机,继续运行.第 3 节的可调度性分析和单处理机故障假设可以保证故障恢复算法的正确性.DNUP 故障恢复算法描述如下:

算法 3. DNUP 故障恢复算法.

输入:DNUP 任务分配方案、处理机个数 m、待加入的处理机 P_f、故障恢复时刻θ.

(1) 依次对每个满足 $1\leqslant j\leqslant m$ 和 $j\neq f$ 的处理机 P_j 执行如下操作:

(1.1) 如果 $passiveRecover(P_j,P_f)\neq\varnothing$:

(1.1.1) 用 $unfinishedRecover(P_j,P_f)$表示 $recover(P_j,P_f)$中在时刻θ尚未完成的副版本集合,将任务集 $unfinishedRecover(P_j,P_f)$中的任务只执行一次就从处理机 P_j 移除;

(1.1.2) 用时刻 t 表示步骤(1.1.1)中最后移除任务的截止时限,对 $primary(P_j)$按照优先级递增顺序,将每个任务τ_i,在时刻 $kT_i((k-1)T_i<t\leqslant kT_i)$设置其 DRR 值为 $d\max_i(Init)$;

(1.1.3) 对于 $active(P_j)$中的每个任务τ_i,在时刻 $kT_i((k-1)T_i<t\leqslant kT_i)$将其加入到处理机 P_j 中,并设置其 DRR 值为 $d\max_i(Init)$;

(1.2) 如果 $passiveRecover(P_j,P_f)=\varnothing$,则继续任务集 $primary(P_j)\cup active(P_j)$.

(2) 对处理机 P_f 执行如下操作:

(2.1) 对于 $primary(P_f)\cup active(P_f)$中每个任务$\tau_i$,在时刻 $kT_i((k-1)T_i<\theta\leqslant kT_i)$将其加入到处理机 P_f 中,并设置它们的 DRR 为 $d\max_i(Init)$;

4.4 性能分析及实例

任务分配算法属于离线算法,其时间消耗集中在用于计算最大响应时间的 CTT 测试上.从任务分配算法可以看出,在判定主版本在某一个处理机上是否可调度时,最多需要进行 m 次 CTT 测试,其中,m 为处理机个数.所以,分配一个主版本最多需要进行 m^2 次 CTT 测试.分配一个主动副版本或者被动副版本最多分别需要 $2m$ 次和

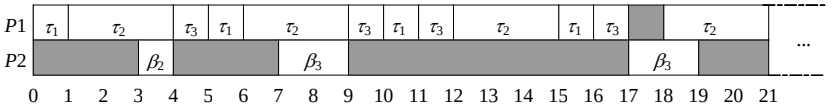
m 次 CTT 测试.所以,如果将一次 CTT 测试作为耗时单位,那么对包含 n 个任务的集合,分配算法的时间复杂度为 $O(nm^2)$,与 FTRMFF 和 ARR 相同.

任务恢复和故障恢复算法不需要进行 CTT 测试,所使用的都是任务分配算法已经离线计算好的数据.因此,它们的时间消耗几乎可以忽略不计,适合在线运行,但故障恢复算法从开始执行到执行完毕需要持续一段时间:在待加入处理机 P_f 运行时间不大于 $primary(P_f)\cup active(P_f)$ 中的最大周期,在无故障处理机 P_j 运行的时间不大于 $unfinishedRecover(P_j,P_f)$ 中的最大周期、$active(P_j)$ 中的最大周期和 $primary(P_j)$ 中所有周期之和.

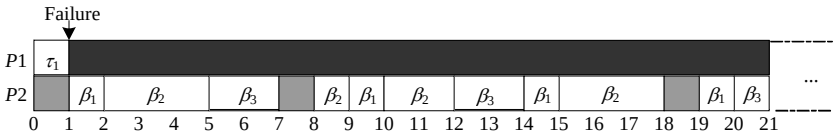
为直观地理解算法,我们给出一个具体例子,以详细说明 DNUP 策略的调度和恢复过程.考虑包含 3 个任务的集合 $\Gamma=\{\tau_1=(0,5,1,5),\tau_2=(0,6,3,6),\tau_3=(0,10,2,10)\}$.根据算法 1,任务集 Γ 需要两个处理机,分配方案为 $P1=\tau_1,\tau_2,\tau_3$;$P2=\beta_1(passive),\beta_2(active),\beta_3(active)$,主/副版本在初始时刻和非紧迫周期内的最大可延迟时间为

$$\tau_1: d\max(Init)=0, d\max(NU)=4,\quad \beta_1: d\max(Init)=1, d\max(NU)=4,$$
$$\tau_2: d\max(Init)=0, d\max(NU)=2,\quad \beta_2: d\max(Init)=3, d\max(NU)=2,$$
$$\tau_3: d\max(Init)=0, d\max(NU)=4,\quad \beta_3: d\max(Init)=7, d\max(NU)=0.$$

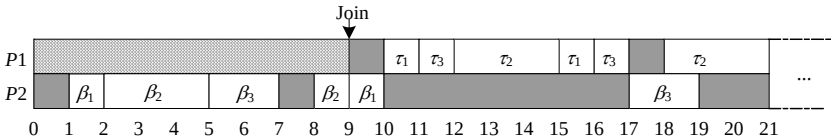
图 4 给出了任务集在 3 种情况下的调度过程,其中,灰色部分表示处理器空闲.在无故障情况下,任务集的调度过程如图 4(a)所示;假设在时刻 1 处理机 $P1$ 发生故障,其余处理机的调度情况如图 4(b)所示;假设时刻 9,处理机 $P1$ 修复完毕加入系统,任务集的调度情况如图 4(c)所示.



(a) 无故障情况



(b) 处理机 $P1$ 在时刻 1 发生故障的情况



(c) 已修复处理机 $P1$ 在时刻 9 加入系统的情况

Fig.4 Scheduling of task set Γ

图 4 任务集 Γ 调度实例图

5 模拟实验

为了验证算法的可行性和有效性,我们进行了模拟实验以比较 DNUP 策略和其他几个主要的分布式容错算法在处理机需求方面的情况.这几个待比较的任务分配算法是:FTRMFF,ARR,S-PRIORITY,TPFTRM 和 DNUP.其中, PRIORITY 算法是文献[10]提出的基于 ARR 策略的另一种算法,它将 ARR 中任务分配处理机的顺序由 RM 优先级递减替换为 S 优先级递减.在这些算法中,只有本文提出的 DNUP 算法考虑故障情况下非紧迫周期的任务延迟.为了使各种算法的性能比较更为清晰,本文采用的模拟实验方法和参数以及性能指标与文献[7–11]相同.具体为:

(1) 每次测试中的实时任务集数目为 n,n 取 100,200,…,600;

(2) 任务的周期 T_i 在[1,500]内均匀分布,定义所有任务的最大负载为 $\alpha=\max\{U_1,U_2,\ldots,U_n\}$,其中,$U_i=C_i/T_i$.每个任务的执行时间 C_i 在$[1,\alpha T_i]$之间随机分布;

(3) α分别设置为 0.2,0.5 和 0.8;

(4) 性能度量标准:处理机数目 M 与任务集负载 $U=U_1+U_2+\ldots+U_n$ 的比值,即 M/U.该值反映了单位负载所需处理机个数,该值越小,表示算法的调度性能越高.对每个 n 和α重复进行 30 次实验,取 30 次实验的平均值作为有效结果,实验结果如图 5~图 7 所示.

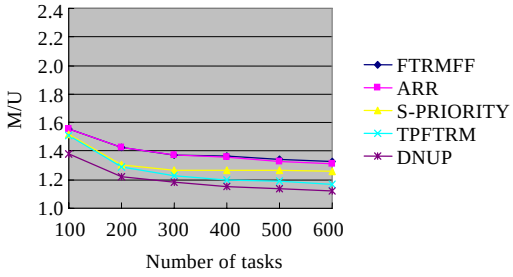


Fig.5 Ratios M/U of different algorithms when α=0.2

图 5 α=0.2 时算法的 M/U 比值

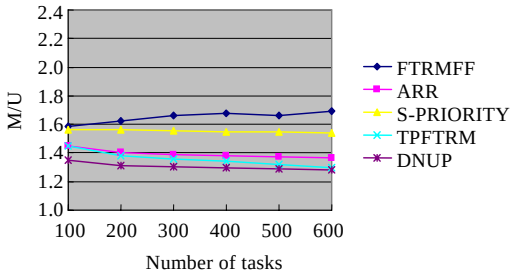


Fig.6 Ratios M/U of different algorithms when α=0.5

图 6 α=0.5 时算法的 M/U 比值

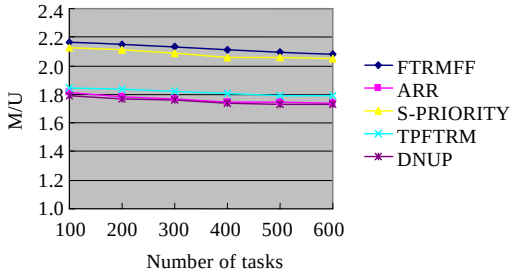


Fig.7 Ratios M/U of different algorithms when α=0.8

图 7 α=0.8 时算法的 M/U 比值

从实验结果可以看出,对于 3 种不同的任务负载,DNUP 较之其他算法都获得了最好的性能.值得注意的是,尽管 DNUP 是基于简单的 RM 优先级任务分配方式,但它所需的处理机个数仍然少于使用复杂度较高的 S 优先级顺序任务分配的 S-PRIORITY 和 TPFTRM 算法.从 3 个图中不难发现,与 ARR 算法相比,DNUP 的性能提高能力随着α的增加而有所降低.这是基于以下两个原因:

(1) ARR 算法在计算最大相位延时的过程中,已经考虑了主动副版本任务在出错情况下的最大响应时间,所以对 DNUP 算法而言,就很难大幅度地提高主动副版本任务的可调度性;

(2) ARR 算法没有考虑被动副版本任务的优化,而 DNUP 算法能够减少被动副版本紧迫周期内的最大响应时间,提高了可调度性.

所以随着α值的增大,副版本任务是主动方式的比例大为增加,性能提高的幅度也随之减小.因此,DNUP 算法更适用于低负载的任务集的容错调度.

6 结束语

本文的主要贡献是,考虑处理机故障后任务实例不同的紧迫程度,提出基于非紧迫周期内的延迟策略.该策略延迟非紧迫实例的执行,让紧迫实例有更多机会完成周期请求,从而本质上实现了处理机空闲资源的均衡分配.研究分析和实验结果均表明,非紧迫周期延迟策略能够在不破坏非紧迫周期内实例实时性的前提下,有效减少紧迫实例的响应时间,增加任务可调度性,减少分布式系统所需处理机个数.

注意到本文提出的分布式调度算法是一种理想情况,因此下一步的工作还包括:在任务调度中引入优先级和资源需求的约束,采用更加高效的任务分配算法,引入非周期任务调度等.

References:

[1] Al-Omari R, Somani AK, Manimaran G. A new fault-tolerant technique for improving schedulability in multiprocessor real-time systems. In: Proc the 15th IEEE Parallel and Distributed Processing Symp. San Francisco: IEEE Computer Society, 2001. 32−33. http://www.computer.org/portal/web/csdl/doi/10.1109/IPDPS.2001.924967 [doi: 10.1109/IPDPS.2001.924967]

[2] Zhu P, Yang FM, Tu G. Ral-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).

[3] Yang FM, Luo W, Pang LP. An efficient real-time fault-tolerant scheduling algorithm based on multiprocessor systems. Wuhan University Journal of Nature Science, 2007,12(1):113−116. [doi: 10.1007/s11859-006-0231-x]

[4] Luo W, Yang FM, Pang LP, Tu G. A real-time fault-tolerant scheduling algorithm of periodic tasks in heterogeneous distributed systems. Chinese Journal of Computers, 2007,30(10):1704−1749 (in Chinese with English abstract).

[5] Qin X, Han ZF, Pang LP, Li SL. Design and performance analysis of a hybrid real-time scheduling algorithm with fault-tolerance. Journal of Software, 2000,11(5):686−693 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/20000516.htm

[6] Liu H, Fei SM. A fault-tolerant scheduling algorithm based on EDF for distributed control systems. Journal of Software, 2003, 14(8):1371−1378 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/20030804.htm

[7] Bertossi AA, Mancini LV, Rossini F. Fault-Tolerant rate-monotonic first-fit scheduling in hard-real-time systems. IEEE Trans. on Parallel and Distributed Systems, 1999,10(9):934−945. [doi: 10.1109/71.798317]

[8] Luo W, Yang FM, Tu G, Pang LP, Qin X. TERCOS: A novel technique for exploiting redundancies in fault-tolerant and real-time distributed systems. In: Proc. of the 13th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007). Daegu: IEEE Computer Society Press, 2007. 275−282. http://www.computer.org/csdl/proceedings/rtcsa/2007/2975/00/29750275-abs.html [doi: 10.1109/RTCSA.2007.70]

[9] Luo W, Yang FM, Pang LP, Li J. A real-time fault-tolerant scheduling algorithm for distributed systems based on deferred active backup-copy. Journal of Computer Research and Development, 2007,44(3):521 −528 (in Chinese with English abstract).

[10] 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. Los Alamitos: IEEE Computer Society Press, 2006. 107−118. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4020794 [doi: 10.1109/DS-RT.2006.33]

[11] 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. 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]

[12] Burchard A, Liebherr J, Oh Y, Son SH. New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. on Computers, 1995,44(12):1429−1442. [doi: 10.1109/12.477248]

[13] Joseph M, Pandya P. Finding response times in a real-time system. The Computer Journal, 1986,29(5):390−395. [doi: 10.1093/comjnl/29.5.390]

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

附中文参考文献:

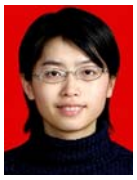
[2] 朱萍,阳富民,涂刚.基于被动副版本优先级提高策略的分布式实时容错调度.计算机研究与发展,2010,47(11):2003−2010.

[4] 罗威,阳富民,庞丽萍,涂刚.异构分布式系统中实时周期任务的容错调度算法.计算机学报,2007,30(10):1704−1749.

[5] 秦啸,韩宗芬,庞丽萍,李胜利.混合型实时容错调度算法的设计和性能分析.软件学报,2000,11(5):686−693. http://www.jos.org.cn/1000-9825/20000516.htm

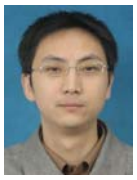
[6] 刘怀,费树岷.基于 EDF 的分布式控制系统容错调度算法.软件学报,2003,14(8):1371−1378. http://www.jos.org.cn/1000-9825/20030804.htm

[9] 罗威,阳富民,庞丽萍,李俊.基于延迟主动副版本的分布式实时容错调度算法.计算机研究与发展,2007,44(3):521−528.



朱萍(1981－),女,江西新干人,博士生,主要研究领域为实时容错调度,实时节能调度.

阳富民(1966－),男,教授,博士生导师,主要研究领域为嵌入式操作系统.



涂刚(1976－),男,博士,副教授,主要研究领域为嵌入式操作系统,弱硬实时调度.

张杰(1978－),男,博士,讲师,主要研究领域为嵌入式操作系统,实时容错调度.

周正勇(1979－),男,博士生,讲师,主要研究领域为嵌入式操作系统,实时容错调度.