混合型实时容错调度算法的设计和性能分析*

秦啸 韩宗芬 庞丽萍 李胜利

(华中理工大学计算机科学与技术学院 武汉 430074)

E-mail: xqin@371.net

摘要 以往文献中研究的实时容错调度算法都只能调度单一的具有容错需求的任务.该文建立了一个混合型实时容错调度模型,提出一种静态实时容错调度算法.该算法能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.该文还提出了一个求解最小处理机个数的算法,用于对静态实时容错调度算法的性能进行模拟分析.为了提高静态调度算法的调度性能,提出了一种动态调度算法.最后,通过模拟实验分析了静态和动态调度算法的性能.实验表明,调度算法的性能与实时任务的个数、任务的计算时间、周期和处理机个数等系统参数相关.

关键词 混合型,容错技术,实时调度,启发式算法,分布式系统,模拟实验,性能分析.

中图法分类号 TP316

许多实时系统,特别是硬实时系统的实时任务必须在规定的时限内完成,否则会产生人员伤亡、财产损失等严重后果[1].实时系统不仅需要保证所有实时任务在截止时限内完成,而且在系统的硬软件出现错误时,实时任务仍可以继续运行.已有许多文献报道了各种容错技术[2~7].文献[2~4]分别研究了分布式投票技术[2]、反转恢复技术[3]和基/副版本技术[4~5].文献[6,7]则分别研究了3种检测暂时性错误的机制[6]和一种具有容错能力的负载平衡算法[7].然而,在文献[2~7]中所介绍的容错技术都没有考虑实时问题,所以不能很好地适应实时系统的要求.文献[8~11]研究了各种实时调度算法,但这些算法尚未考虑容错问题.在实时系统中,必须将容错和实时这两种技术相结合[12].文献[13]研究的容错技术是将失效处理机上的任务动态地迁移到其他处理机上,可是这种技术的开销又很大.文献[14,15]虽然在实时调度时采用了容错技术,但这些调度算法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.因为许多分布式实时应用中的实时任务类型是多种多样的,所以本文建立了一种混合型实时容错调度模型,其中包括了多种类型的实时任务,并在该模型的基础上,提出一种混合型的实时容错调度算法,对多类型的实时任务集合进行调度.最后,我们模拟分析了该调度算法的性能.

1 混合型实时容错调度模型

在混合型实时容错调度模型中,所有的任务之间是相互独立的.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.对于周期实时任务,采用静态调度算法,对于非周期任务,则采用动态调度算法,而对于具有容错需求的实时任务采用双版本策略进行容错调度.具有容错需求的实时任务都有基版本和副版本,系统首先运行基版本,当基版本正确完成时,其副版本不需要运行;而当处理机失效时,在另一个处理机上的副版本将投入运行.调度算法必须保证有容错需求的实时任务的基/副版本都在截止期限

---

* 本文研究得到国防预研基金(No.99j15.2.1jw0519)资助.作者秦啸,1974年生,硕士,主要研究领域为实时系统,容错技术,调度理论,网络通信,数据库管理系统.韩宗芬,女,1950年生,副教授,主要研究领域为分布式实时系统,软件工程.庞丽萍,女,1944年生,教授,主要研究领域为分布式实时系统,并行分布式处理软件.李胜利,1952年生,副教授,主要研究领域为实时系统,网络通信.

本文通讯联系人:秦啸,武汉 430074,华中理工大学计算机科学与技术学院

本文1999-01-12收到原稿,1999-06-15收到修改稿

之前完成.在多数情况下,处理机不会失效,所以预留给副版本的时间会被浪费,而使处理机的利用率很低.为了提高调度性能,混合型实时容错调度算法将在预留给副版本的时间段中插空调度动态到达的非周期任务.调度模型的形式化定义如下:

定义 1. 周期实时任务集合有两个子集,$\Psi_{pd}=\Psi_{ft}\cup\Psi_{nf}$,其中集合 Ψ_{ft} 中的周期实时任务都具有容错需求,而集合 Ψ_{nf} 中的周期实时任务都没有容错需求.

定义 2. 任务集合 $\Psi_{ft}=\{t_1,t_2,\dots,t_n\}$,其中 $t\in\Psi_{ft}$ 是一个四元组,$t=(p,d,tp,tb)$,p 和 d 分别是实时任务的周期和截止期限,tp 和 tb 分别为实时任务的基版本和副版本,tp 和 tb 由三元组描述:$tp=tb=(c,s,pcs)$,其中 c 和 s 是实时任务基版本和副版本的计算时间和开始时间,pcs 指明该版本被调度到哪个处理机上.

定义 3. 任务集合 $\Psi_{nf}=\{t_{n+1},t_{n+2},\dots,t_{n+m}\}$,其中 $t\in\Psi_{nf}$ 是一个五元组,$t=(p,d,c,s,pcs)$,p,d,c 和 s 分别是实时任务的周期、截止期限、计算时间和开始时间,pcs 指明该实时任务被调度到哪个处理机上.

定义 4. 非周期任务表示为一个三元组:$apt=(st,a,c)$,st 是非周期任务的状态,a 是任务的到达时间,c 是任务的计算时间.非周期任务有两个状态:等待状态和被服务状态,$st=\mathrm{WAIT}\mid\mathrm{SERVE}$.

定义 5. 非周期任务集合 Ψ_{ap} 有两个子集,$\Psi_{ap}=\Psi_{wt}\cup\Psi_{sv}$,其中 Ψ_{wt} 是等待集,Ψ_{sv} 是服务集,即:

$$\Psi_{wt}=\{apt\mid apt.st=\mathrm{WAIT},apt\in\Psi_{ap}\},\quad \Psi_{sv}=\{apt\mid apt.st=\mathrm{SERVE},apt\in\Psi_{ap}\}.$$

定义 6. 设分布式实时系统中的处理机个数为 k,处理机集合定义为 $\Omega=\{P_1,P_2,\dots,P_k\}$,$\Omega$ 中的元素 P 都是五元组,$P=(\Delta,\xi_s,\xi_p,\varpi,\tau)$,其中 Δ 是调度到该处理机上的一组周期实时任务集合;ξ_s 是该处理机的调度长度;ξ_p 是调度到该处理机上的周期实时任务的基版本和无容错需求的周期实时任务的总调度长度;ϖ 是处理机的状态;τ 是处理机所处理的非周期任务,当 $\tau=\varepsilon$ 时,处理机没有非周期任务处理.$\varpi=FR\mid PT\mid AP$,FR 表示处理机处于空闲状态,PT 表示处理机在处理周期实时任务,而 AP 表示处理机在处理非周期任务.

模型假设周期实时任务集 Ψ_{pd} 中的任务有相同的周期且截止时限与周期相同,该前提假设形式化的描述为:

$$\forall\, t_i,t_j\in(\Psi_{ft}\cup\Psi_{nf})(t_i\neq t_j)\rightarrow(t_i.d=t_j.d=t_i.p=t_j.p=DL=PD).$$

为了提高静态容错调度算法的性能,在预留给副版本的时间段中插空调度动态到达的非周期任务.为了简化问题且不影响研究提高调度性能的方法,模型假设非周期任务集 Ψ_{ap} 中的任务都是非实时任务,并采用先来先服务调度算法调度这些非周期任务.我们将另文详细研究两种动态调度非周期实时任务的算法.混合型实时容错调度模型还包括以下 5 个基本性质和两个定理.

性质 1. 集合 Ψ_{ft} 中的实时任务在一个处理机失效时,仍可以继续运行的必要条件是:基版本和副版本被调度到不相同的处理机上.即

$$\forall\, t\in\Psi_{ft}(\text{One processor failure is tolerated})\rightarrow\forall\, t\in\Psi_{ft}(t.tp.pcs\neq t.tb.pcs).$$

性质 2. 等待集 Ψ_{wt} 和服务集 Ψ_{sv} 是不相交的两个集合,即

$$\Psi_{wt}\cap\Psi_{sv}=\varnothing.$$

性质 3. 如果一个处理机 P 的状态是 AP,那么 P 所服务的非周期任务在集合 Ψ_{sv} 中,即

$$\forall\, P\in\Omega(P.st=AP\rightarrow P.\tau\in\Psi_{sv}).$$

性质 4. 如果一个处理机 P 的状态是 FR,那么 P 不处理任何非周期任务,即

$$\forall\, P\in\Omega(P.st=FR\rightarrow P.\tau=\varepsilon).$$

性质 5. 两个处理机不能同时处理一个非周期任务,即

$$\forall\, P_i,P_j\in\Omega(P_i.\tau\neq\varepsilon,P_j.\tau\neq\varepsilon,P_i\neq P_j\rightarrow P_i.\tau\neq P_j.\tau).$$

定理 1. 对于集合 Ψ_{ft} 中的实时任务而言,实时任务在一个处理机失效时仍能在截止时限之前执行完,其必要条件是:集合 Ψ_{ft} 中的所有实时任务的基版本和副版本的运行时间不交叉,而且副版本的开始时间不能早于基版本的完成时间,即

$$\forall\, t\in\Psi_{ft}(t.tp.s+t.tp.c\leqslant t.tb.s).$$

证明:用反证法证明.假设 $\exists\, t\in\Psi_{ft}(t.tp.s+t.tp.c>t.tb.s)$,不妨设 $t.tb.s=t.ps.s+t.tp.c\quad\varepsilon(\varepsilon>0)$,并设 $t.tb$ 的完成时间为 $t.d$,即 $t.tb.s+t.tb.c=t.d$.如果处理机在 $t.tp.s+t.tp.c-\delta(0<\delta<\varepsilon)$ 时失效,则立即开始运

行副版本,$t.tb$ 的完成时间是 $t.tp.s+t.tp.c-\delta+t.tb.c=t.tb.s+\varepsilon-\delta+t.tb.c=t.d+\varepsilon-\delta>t.d$,所以 $t.tb$ 不能在规定的截止期限 $t.d$ 内完成,矛盾,故定理 1 成立. □

定理 2. 集合 Ψ_{fi} 中的实时任务在一个处理机失效时,能在截止时限内完成的必要条件是:实时任务的基版本和副版本的计算时间总和必须小于或等于实时任务的截止时限,即 $\forall t\in\Psi_{fi}(t.tp.c+t.tb.c\leqslant t.d)$.

证明:用反证法证明.假设定理 2 的结论不正确,即 $\exists t\in\Psi_{fi}(t.tp.c+t.tb.c>t.d)$,不失一般性,令这个实时任务为 t',又由定理 1 知,$\forall t\in\Psi_{fi}(t.tp.s+t.tp.c\leqslant t.tb.s)$,所以 $t'.tb.s+t'.tb.c\geqslant t'.tp.s+t'.tp.c+t'.tb.c$.因为假设 $t'.tp.c+t'.tb.c>t'.d$,所以,$t'.tb.s+t'.tb.c\geqslant t'.tp.s-t'.tp.c+t'.tb.c>t'.tp.s-t'.d$.又因为 $t'.tp.s\geqslant 0$,故 $t'.tb.s+t'.tb.c>t'.d$,即无法保证实时任务 t' 的副版本在截止时限 $t'.d$ 之前完成.这与定理 2 的前提假设矛盾,所以假设非真,定理 2 成立. □

为了简化问题又不失对实时容错调度算法研究的一般性,本文假设集合 Ψ_{fi} 中实时任务的基版本和副版本的计算时间相同,即 $\forall t\in\Psi_{fi}(t.tp.c=t.tb.c)$.集合 Ψ_{fi} 中的实时任务的定义由四元组变成五元组,$t=(p,d,c,tp,tb)$,其中 p,d 和 c 分别表示为实时任务的周期、截止期限和计算时间.tp 和 tb 分别为实时任务的基版本和副版本,tp 和 tb 由原来的三元组变成两元组:$tp=tb=(s,pcs)$,其中 s 和 pcs 的定义与定义 2 相同.

2 混合型实时容错调度算法

调度模型中有 3 种不同类型的任务:有容错需求的周期实时任务、无容错需求的周期实时任务和动态到达的非周期任务.混合型实时容错调度算法由静态调度和动态调度两个部分组成.首先使用静态的副版本后调度算法调度所有周期实时任务,并将调度结果作为动态调度算法的输入,对非周期任务进行动态调度.

2.1 静态实时容错调度算法

本节将提出一个启发式的静态"副版本后调度算法"去解决实时容错调度问题.该算法简称为 BKCL(back-up copies scheduled last).该算法首先将集合 Ψ_{fi} 中的实时任务的基版本和集合 Ψ_{ni} 中的实时任务一起,按计算时间非增进行排序,然后采用最大计算时间优先调度算法(largest processing time first,简称 LPT)调度这些实时任务,最后再调度集合 Ψ_{fi} 中实时任务的副版本,其方法也是先按计算时间非增对所有的副版本进行排序,然后使用改进的最大计算时间优先调度算法(adaptive LPT,简称 ALPT)来调度这些实时任务的副版本.当分布式实时系统中的处理机个数充足时,BKCL 给出一个有效的实时容错调度结果,如果系统中的处理机个数不足以满足所有实时任务的要求,则 BKCL 算法输出调度失败信息.BKCL 算法描述如下:

副版本后调度算法——BKCL

输入:实时任务集合 Ψ,k;

输出:success,处理机集合 Ω.

(1) FOR $i=1$ TO k DO $P_i.\Delta=\varnothing$;$P_i.\xi_i=0$;$P_i.\xi_p=0$;

(2) IF $\forall t_i\in\Psi_{fi},\forall t_j\in\Psi_{ni}\left(\sum_{i=1}^{n}t_i.c-\sum_{j=1}^{m}t_j.c\geqslant k*DL\right)$ or $\exists\ \forall t\in\Psi_{fi}(t.c>DL/2)$

THEN success=失败,goto (9);

(3) 按计算时间非增对集合 Ψ_{fi} 中的基版本和集合 Ψ_{ni} 中的实时任务进行排序,并重新取名为 $t_1,t_2,\ldots,t_{n-m}$;

(4) FOR $i=1$ TO $n+m$ DO

(4.1) 选择处理机 P_j,使之满足 $\forall P\in\Omega(P_j.\xi_i\leqslant P.\xi_i)$;

(4.2) IF $t_i\in\Psi_{fi}$ THEN $t_i.tp.pcs=j$;$t_i.tp.s=P_j.\xi_i$;$P_j.\Delta=P_j.\Delta+\{t_i.tp\}$;

ELSE $t_i.pcs=j$;$t_i.s=P_j.\xi_i$;$P_j.\Delta=P_j.\Delta+\{t_i\}$;

(4.3) $P_j.\xi_i=P_j.\xi_i+t_i.c$;

(5) FOR $i=1$ TO k DO $P_i.\xi_p=P_i.\xi_i$;/* 生成各处理机的 ξ_p 值 */

(6) IF $\exists P\in\Omega(P.\xi_i>DL)$ THEN success=失败;goto (9);

(7) 按计算时间非增对集合 Ψ_{fi} 中的实时任务的副版本进行排序,并重新取名为 $t_1,t_2,\ldots,t_n$;

(8) FOR $i=1$ TO n DO/* 调度副版本 */

(8.1) 选择处理机 P_j,使之满足 $\forall P\in(\Omega-\{P_{t_i.tp.pcs}\})(j\neq t_i.tp.pcs \wedge P_j.\xi_s\leqslant P.\xi_s)$;

(8.2) $max\_len=MAX(t_i.tp.s+t_i.c,P_j.\xi_s)$;

(8.3) $t_i.tb.pcs=j;t_i.tb.s=max\_len;P_j.len=max\_len+t_i.c;P_j.\Delta=P_j.\Delta+\{t_i.tb\}$;

(8.4) IF $P_j.\xi_s>DL$ THEN success=失败;goto (9);

(9) success=成功;

(10) BKCL 算法结束.

2.2 求解最小处理机个数算法

BKCL 是启发式的静态实时容错调度算法,给出集合 Ψ_{ft},Ψ_{nf} 和处理机个数 k,实时容错算法 BKCL 可以判断 k 个处理机是否能保证所有实时任务在截止时限之前完成.本节设计一个求解最小处理机个数的算法(find minimal number of processors,简称 FMNP).当给出集合 Ψ_{ft} 和 Ψ_{nf} 时,该算法可以计算出满足静态实时容错调度需求的最小处理机个数 k.在算法 BKCL 的基础上,很容易实现算法 FMNP.算法 FMNP 的描述如下:

求解最小处理机个数算法 -- FMNP

输入:Task set Ψ;

输出:k,Ω.

(1) $L=\forall t_i\in\Psi_{ft},\forall t_j\in\Psi_{nf}\left(\left\lfloor\left(\sum_{i=1}^{n}t_i.c+\sum_{j=1}^{m}t_j.C\right)/DL\right\rfloor\right)$;$U=n+m$;

(2) $k=\lfloor(L+U)/2\rfloor$;IF ($Lower=k$) THEN $m=m+1$;goto (5);

(3) $BKCL(\Psi,k;success,\Omega)$; /* 调用静态实时容错调度算法 BKCL */

(4) IF (success=成功) THEN $U=k$;ELSE $L=k$;goto (2);

(5) FMNP 算法结束.

2.3 动态调度算法

当具有容错需求的周期实时任务的基版本正常完成时,其副版本不需要运行,由于处理机在多数情况下不会失效,所以对于处理机 P 在一个周期内的$[P.\xi_p,DL]$上的时间将被浪费.动态调度算法的主要目的是在处理机正常运行时,充分利用周期内$[P.\xi_p,DL]$上的时间.如果运行基版本的处理机失效,相应的副版本将运行在另一个处理机上,并可抢占正在运行的非周期任务.动态调度算法由动态插入、动态释放和动态处理机调度3个算法组成.动态插入算法处理新到达的非周期任务,如果系统中有空闲处理机,则选择一个处理机处理该非周期任务,并将这个任务放入到服务集 Ψ_{sv} 中;如果没有空闲的处理机,则将这个任务插入到等待集 Ψ_{wt} 中.动态插入算法描述如下:

动态插入算法 INSERT

输入:非周期任务 apt,等待集 Ψ_{wt},服务集 Ψ_{sv},处理机集 Ω;

输出:等待集 Ψ_{wt},服务集 Ψ_{sv},处理机集 Ω.

(1) IF ($\forall P\in\Omega(P.\tau\neq\varepsilon)$) THEN $apt.st$=WAIT;将任务 apt 插入到等待集 Ψ_{wt} 中;goto (5);

(2) FOR $i=1$ to k DO IF ($P_i.\tau=\varepsilon$) THEN goto (3); /* 查找可以处理非周期任务的处理机 */

(3) $P_i.\tau=apt$;$apt.st$=SERVE;将任务 apt 插入到等待集 Ψ_{sv} 中;

(4) IF ($P_i.st$=FR) THEN $P_i.st=AP$;立即开始执行非周期任务 apt;

(5) INSERT 算法结束.

如果系统中的处理机 P 完成一个非周期任务,动态释放算法启动执行,将这个任务从服务集 Ψ_{sv} 中删去.如果等待集 Ψ_{wt} 中没有非周期任务等待 P 的服务,则修改处理机的状态,否则按先来先服务原则从等待集 Ψ_{wt} 中选择一个任务,并将这个任务分配给 P.动态释放算法的描述如下:

动态释放算法 DISPOSE

输入:处理机 P.

(1) 将 $P.\tau$ 从服务集 Ψ_{sv} 中删去;

(2) IF ($\Psi_{wt}=\varnothing$) THEN DO $P.st=FR$;$P.\tau=\varepsilon$;goto (6); ELSE DO next Step;
(3) 选择 $apt\in\Psi_{wt}$,满足$\forall\ apt'\in\Psi_{wt}(apt.a\leqslant apt'.a)$; /* 按先来先服务原则从 Ψ_{wt}选择任务 */
(4) $apt.st$=SERVE;将 apt 从 Ψ_{wt}移入 Ψ_{sv};$P.\tau=apt$;
(5) IF $P.st=AP$ THEN 立即开始执行非周期任务 apt;
(6) DISPOSE 算法结束.

动态处理机调度算法对处理机的状态进行动态管理,并根据状态决定处理对象.该算法描述如下:

动态处理机调度算法 DYNSCH

输入:处理机集合 Ω,当前时间 $cur\ \ t$;

输出:处理机集合 Ω.

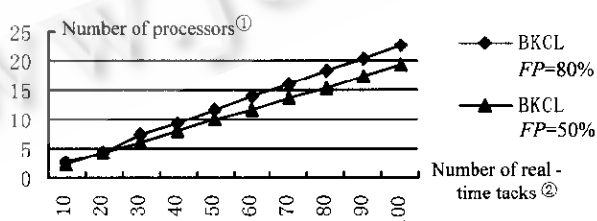
(1) $t=cur\_t$ MOD PR;/* 将当前时间转化为周期内的相对时间 */
(2) FOR $i=1$ TO k DO
  (2.1) IF $t=0$ THEN DO $P_i.st=PT$;开始处理 $P_i.\Delta$ 中的周期实时任务;
  (2.2) IF $t=P_i.\xi_p$ THEN goto (2.3);
  (2.3) IF $P_i.\tau\neq\varepsilon$ THEN $P_i.st=AP$;开始处理非周期任务 $P_i.\tau$;ELSE $P_i.st=FR$;
(3) DYNSCH 算法结束.

3 模拟实验和性能分析

对于混合型实时容错调度算法的性能分析有两个步骤,首先是模拟和分析静态实时容错调度算法,然后在此基础上,对动态调度算法的性能进行模拟和评价.模拟实验的具体方法是:首先确定截止时限 DL(deadline)和实时任务计算时间取值区域 CR(computation time range)两个实验基本参数.实验参数 R 定义为 $R=DL/CR$,当确定 R 时,CR 的值也就确定了.其次,确定集合 Ψ_{ft}和 Ψ_{nf}中实时任务的个数 $N(\Psi_{ft})$和 $N(\Psi_{nf})$,FTP(fault-tolerant percentage)是集合 Ψ_{ft}中实时任务的个数占总个数的比率,即 $FTP=N(\Psi_{ft})/N(\Psi)=N(\Psi_{ft})/(N(\Psi_{ft})+N(\Psi_{nf}))$.再次,随机生成集合 Ψ_{ft}和 Ψ_{nf}中的实时任务,生成的关键是随机产生实时任务的计算时间.实时任务的计算时间满足均匀分布,概率密度函数为 $P\{c=x\}=1/CR$,$x\in[1,CR]$.第4步,使用算法 FMNP 调用 BKCL 去计算出最小处理机个数(minimal number of processors,简称 MNP).最后,按以上论述的步骤(1)~(4),分别相互独立地做1 000组模拟实验,将这1 000次实验所得到的 MNP 值取平均.

3.1 模拟实验1

第1个实验是分析集合 Ψ 中实时任务的个数和最小处理机个数 MNP 之间的关系.由于版面的关系,图1中只给出了最典型的两组实验结果.实验的基本参数为 $R=4$,$DL=160$,FTP 分别设置为80%和50%.当系统中负载较小时,FTP 为50%和80%时的性能差别很小.结果表明:随着集合 Ψ 中任务个数的增加,MNP 的值呈线性趋势增加.因为分布式实时系统中的实时任务个数越多,就需要更多的处理机来保证在截止时限之前完成所有的实时任务.



①节点机个数,②实时任务总数.

Fig. 1 Relation between number of processors and number of real-time tacks
Parameters: $R=4$, Deadline $DL=160$

图1 节点机个数与实时任务总数之间的关系
实验参数:$R=4$,截止时限:$DL=160$

3.2 模拟实验2

本实验的目的是研究实验参数 R 与 MNP 之间的关系.DL 和 FTP 分别设置为210和80%,然后将参数 R 分

别设成2,5,7和10. 首先,如图2所示:随着集合 Ψ 中实时任务的个数的增加,MNP 的值呈线性增加的趋势. 其次,从图2中可以得出,当 R 值越大时,分布式实时系统所需的处理机个数越少. 其原因是,当实验确定了一个截止时限后,实时任务的计算时间的随机生成由 R 决定,R 取值越大,则实时任务计算时间随机产生的值越小,所以,系统就只需越少的处理机来保证所有实时任务在截止时限之前完成. 当集合 Ψ 中实时任务个数很小时,无论 R 设成2,5,7或10,性能差别都很小;但当 $N(\Psi)$ 增大时,性能差别也随之增大.

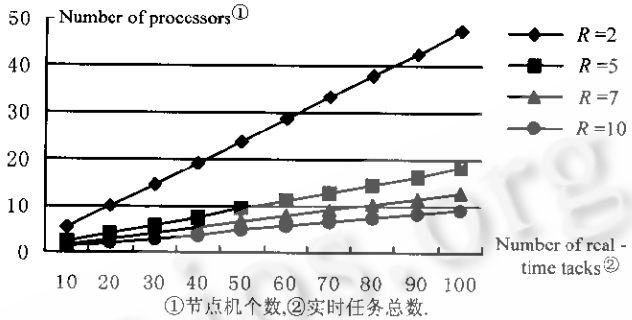


Fig. 2 Relation between parameter R and number of processors
Parameters: $DL=210$, $FTP=80\%$

图2 实验参数 R 与节点机个数之间的关系
截止时限:$DL=210$,$FTP=80\%$

在分析动态调度算法的性能时,非周期任务的平均响应时间是一个重要指标,模拟方法为:首先,确定周期实时任务的周期 PR(period)和需要模拟的非周期任务的个数(实验中非周期任务的个数都为300 000). 其次,确定处理机集合 Ω,其中包括处理机的个数 k 和每一个处理机的 ξ_p 值. 第3步,确定非周期任务的到达率 λ. 非周期任务的到达过程是一个泊松过程,即每两个非周期任务到达的间隔 A 是一个随机变量,它满足指数分布函数:$F_A(x)=P\{A\leqslant x\}=1-e^{-\lambda * x}$,其中 λ 是指任务的到达率,单位是个/s. 第4步,随机产生非周期任务的计算时间. 计算时间满足均匀分布,概率密度函数为 $P\{c=x\}=1/CR$,$x\in[1,CR]$,其中 CR 称为计算时间的取值区域. 最后,处理完成300 000个非周期任务以后,计算非周期任务的平均响应时间(average respond time,简称 ART).

3.3 模拟实验3

本实验用来分析周期实时任务的周期是如何影响平均响应时间的. 模拟实验的基本参数为 $k=4$,$CR=30$,$P_1.\xi_p=40$,$P_2.\xi_p=50$,$P_3.\xi_p=60$,$P_4.\xi_p=70$. 将周期分别设置为100,150,200和250,测出4组模拟数据. 首先,图3显示,随着非周期任务到达率的提高,平均响应时间的数值逐渐增加. 其次,从图3中可以看出:在相同的负载

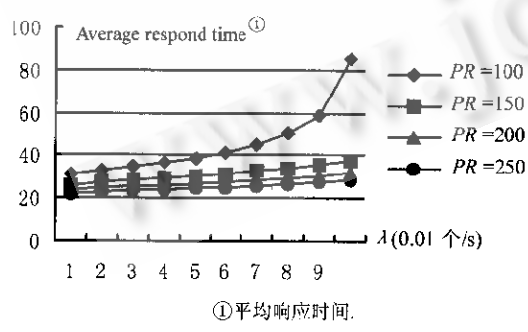


Fig. 3 Relation between preriod and average respond time
Parameters: $k=4$, $CR=30$

图3 周期与平均响应时间之间的关系
实验参数:$k=4$,$CR=30$

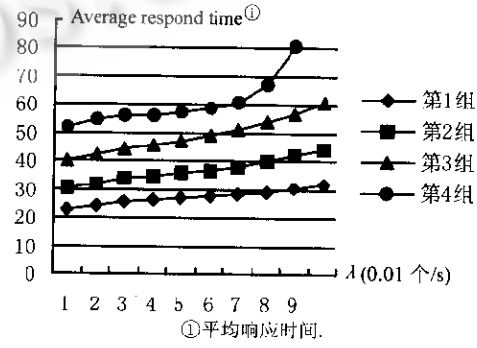


Fig. 4 Relation between scheduling time for primary copies, NFT tasks and ART
Parameters: $k=4$, $CR=30$, $PR=200$

图4 处理机中基版本和无容错需求任务所用的时间与非周期任务平均响应时间的关系
实验参数:$k=4$,$CR=30$,$PR=200$

和系统参数下，周期越大，非周期任务的平均响应时间越小.因为在所有处理机的 ξ_p 值一定的前提下，周期越大，留给非周期任务处理的时间越多，所以非周期任务的响应时间越小.

3.4　模拟实验4

本实验是研究处理机的 ξ_p 值如何影响平均响应时间的.模拟实验的基本参数为 $k=4$，$CR=30$，$PR=200$.第1组数据的处理机 ξ_p 值分别为40,50,60和70;第2组数据的处理机 ξ_p 值分别为60,70,80和90;第3组数据的处理机 ξ_p 值分别为80,90,100和110;第4组数据的处理机 ξ_p 值分别为100,110,120和130.图4的实验结果说明，随着集合 Ω 中处理机的 ξ_p 值的增加，非周期任务的平均响应时间也在增加.因为在周期相同的情况下，处理机的 ξ_p 值越大，则在一个周期内，允许非周期任务处理的时间越少，响应时间随之增大.

3.5　模拟实验5

实验5研究处理机的个数与非周期任务的平均响应时间之间的关系.PR 和 CR 分别设置为100和30.为了便于性能上的比较，这里将所有处理机的 ξ_p 值都设为50.分别将处理机的个数设成2,3,4和5,模拟出4组数据，结果如图5所示.图5表明：在负载完全相同的情况下，处理机的个数越大，动态调度算法的性能越好.原因很明显，处理机越多，在相同的时间内可以处理的动态到达的非周期任务越多.

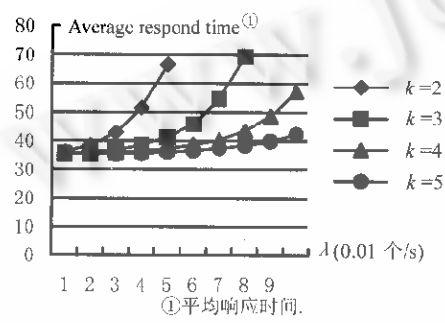


Fig. 5　Relation between number of processors and average respond time
Parameters: $PR=100$, $CR=30$

图5　处理机个数与平均响应时间之间的关系
实验参数：$PR=100$，$CR=30$

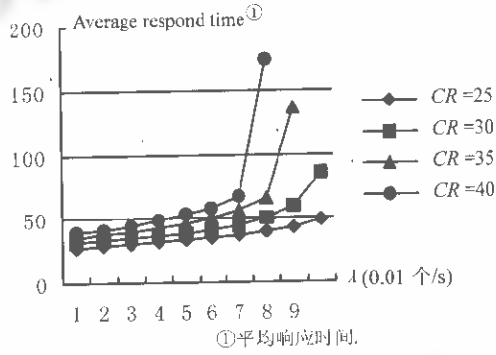


Fig. 6　Relation between computation time and average respond time
Parameters: $PR=100$, $k=4$

图6　任务的计算时间与平均响应时间之间的关系
实验参数：$PR=100$，$k=4$

3.6　模拟实验6

本实验用于分析非周期任务的计算时间与其平均响应时间之间的关系.周期设为100，处理机的个数设为4，4个处理机的 ξ_p 值分别为40,50,60和70.然后将参数 CR 分别设成25,30,35和40.分别计算出4组平均响应时间.实验结果如图6所示.模拟结果显示，非周期任务的计算时间越长，平均响应时间也越大.因为响应时间由计算时间和等待时间两部分组成，计算时间的增加，必然导致整个响应时间的增多.另外，计算时间增加，将造成其他非周期任务等待时间的增多，因此，平均响应时间随着计算时间的取值区域 CR 的增大而增加.

4　结束语

本文的主要创新工作如下：(1) 建立了一个混合型实时容错调度模型，给出了模型的定义和相关的性质，同时证明了该模型的两个定理.(2) 研究了一种静态实时容错调度算法，计算出该算法的时间复杂度.给出一个具体实例以说明该实时容错调度算法的调度过程，并证明了该算法的实时容错调度的正确性.(3) 提出一个求解最小处理机个数的算法，并计算了该算法的时间复杂度.(4) 在静态实时容错调度算法的基础上，提出了一种动态调度算法，该算法包括动态插入算法、动态释放算法和动态处理机调度算法.(5) 建立了性能模拟模型，并对混合型实时容错调度模型中的静态和动态两个调度算法进行性能分析.下一步的研究工作包括：(1) 在静态实时容错调度算法的基础上，研究一种高效的静态实时容错调度算法，在负载相同的情况下，新的高效的算法的性能将比本文提出的静态调度算法还要高.(2) 研究一种动态调度算法，并通过模拟实验分析动态算法的性能.

(3) 本文中的非周期任务都是非实时任务,下一步的研究将提出一种更复杂的混合型实时容错调度模型,模型中的非周期任务包括实时任务和非实时任务两种类型,并研究相关的实时调度算法.

致谢 对评审专家们所给予的学术上的指导和帮助表示衷心的感谢!

参考文献

1 Kieckhafer R M, Walter C J, Finn A M *et al*. The MAFT architecture for distributed fault-tolerance. IEEE Transactions on Computers, 1988,37(4):398~405
2 Xu Li-hao, Bruck J. Deterministic voting in distributed systems using error-correcting codes. IEEE Transactions on Parallel and Distributed Systems, 1998,9(8):813~824
3 Lin T H, Shin K G. Damage assessment for optimal rollback recovery. IEEE Transactions on Computers, 1998,47(5):603~613
4 Davoli R, Giachini L A, Babaoglu O. Parallel computing in networks of workstations with paralex. IEEE Transactions on Parallel and Distributed Systems, 1996,7(4):371~384
5 Siewiorek D P, Swartz R S. Reliable System Design: the Theory and Practice. New York: Digital Press, 1992
6 Miremadi G, Torin J. Evaluating processor behavior and three error detection mechanisms using physical fault-injection. IEEE Transactions on Reliability, 1995,44(3):441~453
7 Kim J, Lee H. Replicated process allocation for load distribution in fault-tolerant multicomputers. IEEE Transactions on Computers, 1998,47(4):499~505
8 Harbour M G, Klein M H, Lehoczky J P. Timing analysis for fixed-priority scheduling of hard real-time systems. IEEE Transactions on Software Engineering, 1994,20(1):13~28
9 Klein M H, Lehoczky J P, Rajkumar R. Rate-Monotonic analysis for real time industrial computing. IEEE Computers, 1994,27(1):24~33
10 Lee C G, Hahn J, Seo Y M *et al*. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. IEEE Transactions on Computers, 1998,47(6):700~713
11 Manimaran G, Murthy C S R. An efficient dynamic scheduling algorithm for multiprocessor real-time systems. IEEE Transactions on Parallel and Distributed Systems, 1998,9(3):312~319
12 Shin K G, Koob G, Jahanian F. Fault-Tolerant in real-time systems. IEEE Real-Time Systems Newsletter, 1991,7(3):28~34
13 Balaji S, Jenkins L, Patnaik L M *et al*. Workload redistribution for fault-tolerant in a hard real-time distributed computing system. Fault-Tolerant Computing Systems, 1989,19(3):366~373
14 Krishna C M, Shin K G. On scheduling tasks with a quick recovery from failure. IEEE Transactions on Computers, 1986,35(4):448~454
15 Bannister J A, Trivedi K S. Task allocation in fault-tolerant distributed systems. Acta Informatica, 1983,20(2):261~281

Design and Performance Analysis of a Hybrid Real-Time Scheduling Algorithm with Fault-Tolerance

QIN Xiao HAN Zong-fen PANG Li-ping LI Sheng-li

(*School of Computer Science and Technology Huazhong University of Science and Technology Wuhan* 430074)

Abstract Since 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 hybrid real-time fault-tolerant scheduling, and proposes a hybrid scheduling algorithm for real-time tasks in this paper. The static scheduling algorithm, a part of hybrid model can schedule tasks with fault-tolerant requirements together with those without fault-tolerant requirements. An algorithm, which is used to find out the minimal number of processors needed for the real-time tasks, is also presented in this paper, so the performance of the static scheduling algorithm can be simulated and analyzed. In order to enhance the performance of the static real-time scheduling algorithm with fault-tolerance, a dynamic scheduling algorithm is studied. The performance simulation and analysis of the scheduling algorithms are presented, and experiment results show that the performance is related with the number of tasks, computation time, period and the number of processors.

Key words Hybrid, fault-tolerant technique, real-time scheduling, heuristic algorithm, distributed system, simulation experiment, performance analysis.