异构分布式系统混合型实时容错调度算法

邓建波 张立臣 邓惠敏
(广东工业大学计算机学院 广州 510006)

摘 要 基/副版本技术是实现实时分布式系统容错的一个重要手段。提出了一种异构分布式混合型容错模型,该模型与传统的异构分布式实时调度模型相比同时考虑了周期和非周期调度任务。在此基础上给出3种容错调度算法:以可调度性为目的SSA算法、以可靠性为目的RSA算法、以负载均衡性为目的BSA算法。算法能够在异构系统中同时调度具有周期和非周期容错需求的实时任务,且能够保证在异构系统中某节点机失效情况下,实时任务仍然能在截止时间内完成。最后从可调度性、可靠性代价、负载均衡性、周期与非周期任务数及任务周期与粒度5个方面对算法进行了分析。模拟实验结果显示算法各有优缺点,所以在选择调度算法时应该根据异构系统的特点来选择。

关键词 主/副版本,异构分布式,周期与非周期任务,负载均衡性,可靠性,可调度性
中图法分类号 TP316.4 **文献标识码** A

Real-time Scheduling Algorithm of Hybrid with Fault-tolerant in Heterogeneous Distributed Systems

DENG Jian-bo ZHANG Li-chen DENG Hui-min
(Dept of Computer Guangdong University of Technology, Guangzhou 510006, China)

Abstract The primary/backup process is commonly used in heterogeneous distributed systems with fault-tolerance. This paper proposed a heterogeneous distributed hybrid model with fault-tolerance. Compared with the traditional heterogeneous distributed scheduling models, this model can simultaneously schedule both periodical and aperiodical tasks. Three fault-tolerant scheduling algorithms based on this model were presented: SSA(Schedulability Scheduling Algorithm) algorithm aimed at schedulability, RSA(Reliability Scheduling Algorithm) algorithm aimed at reliability and BSA(Balanced Scheduling Algorithm) algorithm aimed at load equalization. These algorithms can simultaneously process real-time tasks in demand of periodical or aperiodical fault-tolerance in heterogeneous systems. And they can guarantee that real-time tasks could complete befor the cut-off time even if some node of the system fails. Finally, this paper analyseds the algorithms in five ways: schedulability, reliability cost, load equalization, number of periodical and aperiodical tasks, cycle and granularity. Experiment results show that the algorithms have advantages and disadvantages respectively, so they should be chosen according to the characteristics of a specail heterogeneous system.

Keywords Primary/backup copy, Heterogeneous distributed, Periodic and aperiodic task, Load balancing, Reliability, Schedulability

实时系统的一个重要需求就是实时任务必须在规定时间内完成,而且在系统出现软硬件错误时,实时任务仍可以得出可以接受的结果[1]。为了保证系统出现故障后仍能在期限前完成,必须为系统提供一定的容错能力[2]。实时容错调度是实现分布式实时系统容错的主要方法之一。针对现代分布式实时系统的特点,从多种性能指标出发,学者们分别研究和设计基于同构和异构分布式系统中的实时容错调度算法,研究的目的是通过高质量的调度算法来提高分布式实时系统的可调度性、系统可靠性,以及任务接收率等不同的性能指标。目前已有许多学者对容错技术进行研究[3-8]。文献[3]研究一种分布式投票技术,文献[4]给出一种反转恢复技术,文献[5-8]是目前比较流行的基于基/副版本备份技术(Primary/Backup copy,P/B),它广泛应用于分布式系统的容错调度算法,它是通过在备份处理机上执行备份任务来实现系统容错。但上述算法都没有同时考虑周期和非同期调度任务,所以不能很好地适应现实环境的要求。文献[9]分别研究了速率单调算法和动态实时调度算法。文献[10,11]讨论了固定优先级实时调度算法,不足的是这些算法并没有考虑容错问题。文献[12-14]给出了在同构环境下的周期和非同期任务周期模型,但是该模型下只适应于同构环境,并且在算法调度过程中没有考虑负载均衡问题。文献[15,16]基于可靠性代价对异构分布式模型及算法做了讨论,但都是基于非周期任务。文献[17]提出了一种基于基/副版本的负载均衡容错调度算法,但是没有考虑周期性任务的调度。文献[18]提出一种基于异构

到稿日期:2010-04-20 返修日期:2010-10-02 本文受国家自然科学基金重大研究计划(90818008),国家自然科学基金项目(60774095,60474072Z)资助。

邓建波(1984-),男,硕士生,主要研究方向为容错计算、形式化方法及分布式信息处理,E-mail:dengjianbo-52088@163.com;张立臣(1962-),男,教授,博士生导师,主要研究方向为分布式处理和实时系统;邓惠敏(1983-),男,硕士生,主要研究方向为实时系统。

分布式系统的周期性实时调度模型及算法，但是没有考虑非周期任务。目前在各个应用领域中要求实时系统在运行复杂任务时具有良好的容错性、可靠性、可调度性及负载均衡性。

本文在异构分布式实时系统模型的基础上结合上述文献部分思想，给出了一个异构分布式实时系统混合模型，并在此基础上综合上述文献部分思想，从可靠性、可调度性及负载均衡性3个不同角度给出了不同调度算法。最后通过模拟实验分析比较了各调度算法的复杂度和性能，并总结实验结果，得出了有意义的结论。由于容错任务调度问题是一个NP难题[19]，因此我们提出的调度算法都是启发式算法。

1 前提假设与计算模型

1.1 前提假设

本文考虑的是一个典型的多处理机和实时任务的分布式系统，基实时容错调度模型的假设前提是：(1)假设有 $m+n$ 个任务运行在具有 k 个处理机的异构分布式实时系统中，且一个处理机的失效可以立即被检测出，该处理机上的实时任务的副本将在其它的节点机上运行，其中 m 表示为有实时周期任务数，n 表示非周期任务数。由于研究实时任务只有一个副版本，因此我们一次只允许单点失效。若需要多点失效，则可以引入多个副版本。(2)系统中所有任务都相互独立，并且各任务在调度算法中是不可抢占的，即在一个任务没有完成前，其它任务不能抢占CPU及相关资源。(3)系统中所有的非周期实时任务具有相同的截止时间，即处理机的最大调度长度，且在同一处理机上的不同任务在时间上没有重叠。(4)由于本文使用的是被动副版本模式，因此同一任务的基副版本在不同的处理上时间不重叠且截止时限与周期相同。

1.2 计算模型

定义1 实时任务集合是一个二元组，$T=\{T_p, T_{np}\}$，其中集合 T_p 表示周期实时任务集，T_{np} 表示非周期实时任务集。在容错调度过程中保证实时任务集合在异构系统中某一处理机失效时，仍能够在规定的截止期内完成。由假设知实时任务集合只有一个副版本，且只有在基版本所在的节点机出现故障时副版本才投入运行。实时任务集合 T_p 与 T_{np} 定义如下。

定义2 集合 $T_p=\{t_{p1}, t_{p2}, \cdots, t_{pm}\}$，其中 $t_p \in T_p$ 是一个四元组，$t_p=\{p, d, tp, tb\}$，其中 p, d 分别表示任务周期和截止时间；tp 与 tb 表示调度任务的基版本和副版本，其中 $tp=tb=\{s, C, pcs\}$，s 表示实时任务的基或副版本的开始时间；C 是一个向量集合，表示调度到各处理机上的执行时间；pcs 表示基或副版本所调度到的处理机。

定义3 集合 $T_{np}=\{t_{np1}, t_{np2}, \cdots, t_{npn}\}$，其中 $t_{np} \in T_{np}$ 是一个三元组 $t_{np}=\{d, tp, tb\}$，d 表示实时任务截止时间；tp, tb 是实时任务 t_{np} 的基版本和副版本。假设两个版本的程序完全相同。Tp 和 tb 与定义2相同。

定义4 在异构分布式系统中，每个任务 t 在不同的处理机上有不同的执行时间，所以为每个任务 t 定义一个时间向量 C。$C_i=[c(i,1), c(i,2), \cdots, c(i,m)]$，$C_{ij}$ 表示任务 t_i 在处理机 j 上执行完所需的时间。

定义5 异构分布式系统描述为一个处理机集合，$\Phi=\{P_1, P_2, \cdots, P_k\}$。$P=\{PT, \theta, Lp, \lambda, \beta, DL\}$，其中 PT 表示调度到该处理机上的任务集合且 $PT=\{P, Np\}$，其中 P 和 Np 分别表示周期与非周期任务集合；θ 表示当前调度长度；Lp 表示调度到该处理机上周期任务的调度长度；λ 和 β 表示处理机的失效率和性能参数，$\beta \in (0,2)$ 以1为中心，β 的值是处理机在整个异构系统中性能衡量标准且值越小性能越好，反之亦然；任务 t_i 调度到处理机 P_j 的执行时间与 β_j 积，表示任务 t_i 在处理机 P_j 中的实际负载；DL 表示处理机最大调度长度。各个处理机之间出错过程假设为相互独立，且符合泊松过程。

为了计算可靠性代价及负载均衡性，我们进行如下定义：

定义6 在系统无处理机失效时，系统的可靠性成本(Reliability Cost) $RC_0(\Phi)$ 及当处理机 P_i 失效时，系统的可靠性代价 $RC_1(\Phi, i)$ 定义如下[引用文献20定义]：

$$Rc_0(\Phi)=\sum_{j=1}^{k}\sum_{t_i \in T_i}\lambda_j c(i,j)$$

$$Rc_1(\Phi,i)=\sum_{j=1 \wedge j\neq i}^{k}\sum_{t_h \in T_j}\lambda_j c(h,j)+\sum_{j=1}^{k}\sum_{t_i \in T_i \wedge t_h \in T_j}\lambda_j c(h,j)$$

定义7 定义 FT_β 表示处理机当前状态下在系统中的实际负载，即 $FT_\beta=P_i.PT.\beta_i$；$AVFT_\beta$ 表示当前系统的实际负载均值，即 $AVFT_\beta=(\sum_{j=1}^{k}P_j.\theta \times P_j.\beta) \div k$。

令 $\rho(P_j)$ 表示处理机实际负载与系统平均负载差值，即 $\rho(P_j)=FT_{\beta}-AVFT_\beta$。$\xi$ 表示处理机的均衡因子，其定义为 $\xi_i=\rho(P_j)/AVFT_\beta$。由此可知 ξ_i 的值以0为中心上下波动，大于0时表示负载高于平均负载，反之低于，且 ξ_i 等于0时为理想均衡因子，表示为绝对平均负载。通过求异构系统调度成功后 ξ_i 的绝对值的均值 ρ 来反应系统的均衡性。即 $\rho=\sum_{j=1}^{k}(|\xi_j|) \div k$，$\rho \in [0,1]$，且 ρ 值越小说明系统越均衡，ρ 称为系统均衡系数。

本文讨论的是在异构分布式系统中静态的被动式非抢占式混合调度算法。由上面的定义及假设可以得到以下3个性质和两个定理(引用文献[13,16]的部分思想)：

性质1 对于异构分布式系统中每一个处理机 P_i 都满足调度到该处理机上所有任务集的调度长度和不大于处理机的最大调度长度，形式化描述为：

$$\{\forall P_j \in \Phi, \forall j \in [1,k]\} \rightarrow \{\sum_{t_i \in TP_j.P}c(i,j)+\sum_{t_h \in TP_j.Np}c(h,j) \leqslant P_j.DL\}$$，其中 $\forall i \in [1,m], \forall h \in [1,n]$

性质2 任何任务的基/副版本不能同时调度到同一处理机上，形式化描述为：

$$\{(t_i \in T, (i \in [1,m+n]\} \rightarrow t_i.tp.pcs \neq t_i.tb.pcs$$

性质3 同一个任务的同一版本不能同时被两个或两个以上的处理机调度，形式化描述为：

$$\{\forall t_i \in T, \forall i \in [1,m+n]\} \rightarrow \{t_i.tp \,||\, t_i.tb \in P_j.PT \wedge t_i.tp \,||\, t_i.tb \notin P_h.PT \wedge j \neq h\}$$

式中，$\forall j, h \in [1,k]$。

定理1 集合 Φ 中的任务在一个处理机失效时仍然可以继续运行[13]，当且仅当任务的基/副版本间无时间上的重叠(因为本文讨论的是被动式副版本)，形式化描述为：

$$\{(t_i \in T, \forall i \in [1,m+n]\} \rightarrow \{t_i.tp.s+c(i,t_i.tp.pcs) \leqslant t_i.tb.s\}$$

证明：用反证法证明。假设 $\{\exists t_i \in T, \forall i \in [1,m+n]\} \rightarrow \{t_i.tp.s+c(i,t_i.tp.pcs) > t_i.tb.s\}$，不失一般性我们设 $t_i.tb.s=t_i.tp.s+c(i,t_i.tp.pcs)-a(a>0)$，并设副版本的结束时恰好为任务的截止时间(周期任务则为周期，非周期任务为处

理机最大调度长度),即 $t_i.tb.s+c(i,t_i.tb.pcs)=t_i.d$。如果处理机在 $t_i.tp.s+c(i,t_i.tp.pcs)-\delta(0<\delta<a)$时失效,则立即运行副版本任务(忽略处理机之间的通信时间),那么副版本的完成时间为 $t_i.tp.s+c(i,t_i.tp.pcs)-\delta+c(i,t_i.tb.pcs)$,由于 $t_i.tp.s+c(i,t_i.tp.pcs)-a<t_i.tp.s+c(i,t_i.tp.pcs)-\delta$,那么 $t_i.tp.s+c(i,t_i.tp.pcs)-\delta+c(i,t_i.tb.pcs)>t_i.tp.s+c(i,t_i.tp.pcs)-a\rightarrow t_i.tp.s+c(i,t_i.tp.pcs)-\delta+c(i,t_i.tb.pcs)>t_i.tb.s+c(i,t_i.tb.pcs)=t_i.d$。由上可知故障出现后副版本的结束时间超过截止时间。这与在截止时间 $t_i.d$ 完成相矛盾,即定理 1 得证。

定理 2 系统容忍一个处理机失效,则所有任务的基版本和副版本的计算时间之和不大于截止期限。由于系统调度两种不同的任务,因此我们将其分成周期任务和非周期任务。周期任务的截止期限为调度任务的周期,形式化描述为:

$\{(t_i\in T.T_p,\forall i\in[1,m]\}\rightarrow c(i,t_i.tp.pcs)+c(i,t_i.tb.pcs)\leqslant t_i.d=t_i.p$

非周期任务的截止期限为处理机的最大调度长度,形式化描述为:

$\{\forall t_i\in T.T_{np},\forall i\in[1,n]\}\rightarrow c(i,t_i.tp.pcs)+c(i,t_i.tb.pcs)\leqslant t_i.d=P_i.DL$

证明:使用反证法。假设$\{\exists t_i\in T,\forall i\in[1,m+n]\}\rightarrow\{c(i,t_i.tp.pcs)+c(i,t_i.tb.pcs)>t_i.d\}$,由定理 1$\{\forall t_i\in T,\forall i\in[1,m+n]\}\rightarrow\{t_i.tp.s+c(i,t_i.tp.pcs)\leqslant t_i.tb.s\}$知 $t_i.tp.s+c(i,t_i.tp.pcs)+c(i,t_i.tb.pcs)\leqslant t_i.tb.s+c(i,t_i.tb.pcs)$,又由假设知 $t_i.d+t_i.tp.s<t_i.tp.s+c(i,t_i.tp.pcs)+c(i,t_i.tb.pcs)\leqslant t_i.tb.s+c(i,t_i.tb.pcs)$,又因为任务调度的开始时间不小于 0,故 $t_i.tp.s\geqslant 0$;所以知 $t_i.d<t_i.tb.s+c(i,t_i.tb.pcs)$与定理 1 矛盾,定理 2 得证。

2 启发式调度算法

与实时调度算法一样,实时容错调度算法也是一个 NP 难度问题[19]。本节基于异构模型提出 3 个启发式算法:SSA,RSA,BSA。分别从不同角度对系统性能做出分析,算法的启发式规则如下:

(1)算法在调度过程中首先调度周期任务的基版本,根据上述 3 种不同算法选择执行时间、可靠性代价或负载均衡性最小的处理机。再调度周期任务的副版本。

(2)算法在调度非周期任务的基版本,计算处理机上所有空闲的时间片,根据最佳选择算法进行调度。调度副版本时同理。用 CTS(Calculate-Time-Slice)表示计算时间片方法,形式化描述如下:

```
Fun_CTS(Pj){/* P 表示输入的处理机 */
    对处理机 P 上的任务集合 PT 按任务的开始时间排序,并重新编
        号为 t1,t2,…tk;
        For i=0 to k do{Array_CTS[h]=Object(ti,ti+1.s-(ti.s+
        c(i,j)));/*计算时间片按非增排序*/}
        Return Array_ CTS;
        }
```

(3)判断是否调度完所有任务,是 Return SUCCESS,否则 Return FAIL。

2.1 SSA 算法

SSA 算法主要目的是考虑任务的可调度性。给出调度任务集合 T 及处理机个数,输出处理机集合。该算法首先将任务集合 T 中的基版本在各处理机上的执行时间按非增排序,并采用启发式贪婪算法选择执行时间最小的处理机,并同时满足性质 1、性质 3。其次在调度任务的副版本也采用改进贪婪算法选择执行时间最小的处理机,但要同时满足性质 1,2,3 和定理 1,2。最后调度非周期任务基版本选择执行时间最小的处理机使用 CTS 计算时间片,并在性质 1,3 满足时用首次适应算法时间片插入。第四步调度非周期任务副版本同理,但是要满足性质 1,2,3 和定理 1,2。算法的形式化描述如下:

SSA 算法

输入:调度任务集合 T 及处理机个数 k;

输出:处理机集合 Φ;

1. 初始化处理机集合 Φ,任务集合 T;
2. for(i=1 to m)and($t_i\in T_p.tp$) { /* 调度周期任务基版本 */
   2.1 选择执行时间最小处理机 P_j
   2.2 If($P_j.\theta+c(i,j)\leqslant P_j.Lp\wedge P_j.\theta+c(i,j)\leqslant t_i.d$){更新相关信息} Else 选择次优的处理机;}
3. If(j>=k) Return FAIL;/*没有合适的处理机则结束*/
4. for(i=1 to m)and($t_i\in T_p.tb$){ /*调度周期任务副版本*/
   4.1 选择执行时间最小处理机 P_j,$P_j.\theta$=Max($P_j.\theta$,$t_i.tp.s+c(i,t_i.tp.pcs)$)/*计算副版本开始时间*/
   4.2 If($P_j.\theta+c(i,j)\leqslant P_j.DL\wedge P_j.\theta+c(i,j)\leqslant t_i.d\wedge t_i.tp.pcs\neq j$){更新相关信息} Else 选择次优的处理机;}
5. If(j>=k) Return FAIL;/*没有合适的处理机则结束*/
6. for(i=1 to n)and($t_i\in T_{np}.tp$) { /*调度非周期任务基版本*/
   6.1 选择执行时间最小处理机 P_j,调用计算时间片段方法 Fun_CTS(P_j)
   6.2 If($\exists$CTS$\in$Fun_CTS(P_j)$\rightarrow c(i,j)\leqslant$CTS){用首次适应算法插入任务 t_i 并更新相关信息} Else 选择次优的处理机;}
7. for(i=1 to n)and($t_i\in T_{np}.tb$) { /*调度非周期任务副版本*/
   7.1 选择执行时间最小处理机 P_j,调用计算时间片段方法 Fun_CTS(P_j)
   7.2 If($\exists$CTS$\in$Fun_CTS(P_j)$\rightarrow c(i,j)\leqslant$CTS$\wedge$CTS.($t_k.s+c(k,j)$)$+c(i,j)\leqslant P_j.DL\wedge t_i.tp.pcs\neq j\wedge t_i.tp.s+c(i,t_i.tp.pcs)\leqslant$CTS.($t_k.s+c(k,j)$)) {用首次适应算法插入任务 t_i 并更新相关信息}Else 选择次优处理机;
8. If($T=\emptyset$)Return SUCCESS;Else Return FAIL;

SSA 算法的时间复杂度分析如下。选择任务执行时间最小的处理机在最坏情况下所需时间为 $O(k)$,那么对 m 个周期任务的基/副版本分配到 k 个处理机上所需时间为 $O(m\times k)$。在分配非周期任务时计算时间片段并排序在最坏情况下所需的时间为 $O((m+n)\log(m+n))$,那么 n 个非周期任务的基/副版本分配到 k 个处理机时所需的时间为 $O(n\times k\times(m+n)\log(m+n))$。最后 SSA 算法的时间复杂度总计为 $O(2\times m\times k+2\times n\times k\times(m+n)\log(m+n))=O(n\times k\times(m+n)\log(m+n))$。

2.2 RSA 算法

RSA 算法以最小可靠性代价为目的。算法与 SSA 算法类似,仅在选择处理机时选择可靠性代价最小的处理机。其算法的主要步骤如下:

Step 1 通过向量 C_i 和处理机集合 Φ 计算出每一个任务的可靠性代价向量 $RcV_i=[rcv_{i1},rcv_{i2},\cdots,rcv_{im}]$,其中 $rcv_{ij}=(rc_{ij},t_i.pcs)$。$rcv_{ij}$ 表示任务 t_i 被调度到处理机 j 上的可靠性代价,即 $rcv_{ij}=c(i,t_i.pcs)\times\lambda_{n.pcs}$。并对 RcV_i 中可

靠性代价非减排序，以满足{$\forall i\in[1,n+m],\forall j,h\in[1,k]|j<h\rightarrow rcv_{ij}<rcv_{ih}$}。

Step 2　调度周期任务基版本 $t_i.pt$，选择可靠性代价小且调度长度不大于 Lp，即{$rcv_{ij}=\mathrm{Min}\{RcV_i\}\wedge P_j+c(i,j)\leqslant P_j.Lp$}。

Step 3　调度实时周期任务的副版本类似于调度其基版本，但要同时满足性质 1,2,3 和定理 1,2。

Step 4　调度非周期任务基版本 $t_i.tp$，选择可靠性代价最小的处理机，以 Fun_CTS 方法计算时间片段采用首次适应算法且满足性质 1,3。否则选择次优处理机。

Step 5　调度非周期任务副版本 $t_i.tb$，选择一个可靠性代价小的处理机并以 Fun_CTS 方法计算时间片段采用首次适应算法且同时满足性质 1,2,3 和定理 1,2。即{$CTS.(t_k.s+c(k,j))+c(i,j)\leqslant P_j.DL\wedge t_i.tp.pcs\neq j\wedge t_i.tp.s+c(i,t_i.tp.pcs)\leqslant CTS.(t_k.s+c(k,j))$}，否则选择次优处理机。

Step 6　如果所有任务分配完毕，则 Return SUCCESS，否则 Return FAIL。

RSA 算法的调度过程与 SSA 算法一致，只是要选择可靠性代价最小的处理机，故 RSA 算法的时间复杂度与 SSA 算法相同。

2.3 BSA 算法

BSA 算法以异构系统负载均衡性为目的。算法与 SSA 算法类似，仅在选择处理机时选择负载均衡性最小的处理机。为了使算法能够综合考虑算法的均衡性和系统整个负载，我们引入下面定义：

定义 8　令 γ 表示处理机的负载综合系数，其值由均衡系数和处理机的性能决定，即 $\gamma=(\varepsilon_1\times\xi+\varepsilon_2\times\beta)$，其中 ε_1 与 ε_2 表示权值。均衡系数 ξ 由于是一个动态过程，因此可以在调度过程中很好反映系统的负载均衡性。其算法的主要步骤如下：

Step 1　首先在调度周期任务集合 T 基版本时，通过选择实际执行时间最小的处理机且满足调度到该处理机时处理机的调度时间不超过 Lp，采用启发式贪婪算法进行调度，以满足{$\forall t_i\in T.T_p,P_j\in\Phi,\forall i\in[1,n],\forall j\in[1,k]|t_i.pt\in P_j.PT.P\rightarrow\gamma\times c(i,j)=\mathrm{Min}\{\forall h\in[1,m]|\gamma\times c(i,h)\}\cap P_j.\theta\leqslant P_j.Lp$}。

Step 2　再按照启发式贪婪算法对副版本进行分配，但同时满足性质 1,2,3 和定理 1,2，且选择 γ 与执行时间积最小的处理机，即如果 t_i 的副版本调度到处理机 P_j 上，则满足{$\forall t_i\in T.T_p,P_j\in\Phi,\forall i\in[1,n],\forall j\in[1,k]|t_i.tb\in P_j.PT.P\rightarrow\gamma\times c(i,j)=\mathrm{Min}\{\forall h\in[1,k]|\gamma\times c(i,h)\}\cap P_j.\theta\leqslant P_j.DL\cap t_i.tp\notin P_j.PT.P$}。

Step 3　调度非周期任务基版本 $t_i.tp$，选择 γ 与执行时间积最小的处理机并使用 Fun_CTS 方法计算时间片段并采用首次适应算法且满足性质 1,3。否则选择次优处理机。

Step 4　调度非周期任务副版本 $t_i.tb$，选择 γ 与执行时间积最小的处理机，使用 Fun_CTS 方法计算时间片段并采用首次适应算法且同时满足性质 1,2,3 和定理 1,2。即{$CTS.(t_k.s+c(k,j))+c(i,j)\leqslant P_j.DL\wedge t_i.tp.pcs\neq j\wedge t_i.tp.s+c(i,t_i.tp.pcs)\leqslant CTS.(t_k.s+c(k,j))$}，否则选择次优处理机。

Step 5　如果所有任务分配完毕，则 Return SUCCESS，否则 Return FAIL。

BSA 算法在分配待调度任务时均需计算处理机的均衡系数 ξ 并求得负载综合系数 γ。求 ξ 需计算当前系统中所有已调度任务的负载，故在最坏情况下所需时间为 $O((m+n))$。BSA 算法在调度任务时与上述两种算法相同，只是在选择处理机时基于负载综合系数 γ，所以其时间复杂度总计为：$O(n\times k\times(m+n)^2\log(m+n))$。因此上述两种算法的时间复杂度优于 *BSA* 算法。

3　模拟实验与性能分析

我们使用带有 1.6GHz Core(TM)2 Duo CPU，内存 2G 的 PC 机进行模拟实验，模拟程序语言使用 Java，编译工具使用 Eclipse3.5。模拟实验的步骤是：首先确定系统的截止期限 DL，实时任务计算时间取值区域 C 及处理机 Lp 值且 $Lp=DL/2$。其中定义 $R=DL/C$ 在 R 确定时 C 也就确定；其次随机生成处理机的个数以及各个处理机的失效率 λ 及性能参数 β；第三步确定任务集合中周期任务 T_p 与非周期任务 T_{np} 的数量，定义 PP(Periodic Percentage) $=T_p/T=T_p/(T_p+T_{np})$，确定 R 的值并根据 C 随机生成每一个任务的计算时间向量 C_i，且满足均匀分布；最后分别调用算法 SSA，RSA 和 BSA，如果算法成功产生调度结果，则可以在调度算法执行完成后，计算出相应的 $Rc_0(\Phi)$ 和 ρ 的值及使用文献[15]的 FM-NP 算法求出最小的处理机。对每一组的实验输入数据，分别同时产生 100 组数据并求出其平均值。为了计算方便，上述参数均从表 1 取值。

表 1　模拟实验参数表

参数	说明	取值范围
DL	处理机最大调度长度	{1200,1500,1800}
λ	失效率	$(0.7,0.8,\cdots,1.4)\times10^{-6}$次/小时
β	性能参数	(0.1,0.2,…,2.0)
R	截止期限与计算时间域比	(30,40,…,100)
d	任务的周期	(100,200,…,600)
PP	周期与非周期任务数比	(10,20,…,90)%

3.1　算法可靠性代价分析

本节比较 3 种算法的可靠性代价，模拟实验(1)的基本参数为，处理机的最大调度长度 $DL=1500$，$Lp=800$，$R=50$，$PP=50\%$，为了方便计算，我们令任务的周期 $d=DL$，处理机个数为 $k=20$，其可靠性代价 λ 及性能 β 分别从表 1 中随机取出，同时在算法 BSA 中，我们取 $\varepsilon_1=1.8$，$\varepsilon_2=0.2$。该实验主要目的是在保证算法成功调度情况下对算法的可靠性代价进行分析。图 1 给出了模拟实验结果。

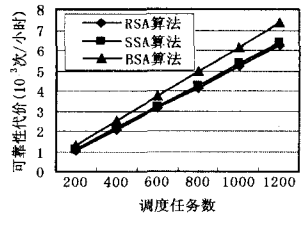

图 1　算法的可靠性代价 $R=30$，$PP=50\%$

从图 1 可知，随着任务数的增加，可靠性代价成线性增加，这是因为任务数增加需要更多的执行时间。同时还可以看出 BSA 的可靠性代价最高，RSA 的可靠性代价最低。这

主要是因为 RSA 每次调度任务时都考虑了最小可靠性代价。PP 的取值对可靠性代价的影响与图 1 基本相同，限于篇幅本文不再给出。其主要原因是因为周期任务与非周期任务都具有容错需求，PP 的取值在总任务数相同的情况下对所有任务总的执行时间没有影响。

3.2 算法负载均衡性分析

模拟实验(2)主要是为了从算法的均衡性对算法进行分析，实验参数与模拟实验(1)类似。模拟实验结果如图 2 所示。从图 2 可知，随着任务数的增加，算法的负载均衡性有所提高，这是因为调度任务数的增加使得调度到部分处理机的可能性变小。还可以看出在固定处理机个数及其它参数时，BSA 算法的负载均衡性明显优于另两种算法。例如在任务数为 800 时，3 种算法的均衡系数 ρ 分别为 0.454，0.398，0.153。这主要是因为 BSA 算法在调度任务时考虑了系统的均衡性。PP 取值对可靠性代价的影响与图 2 基本相同，不再给出。原因同 3.1 节分析。

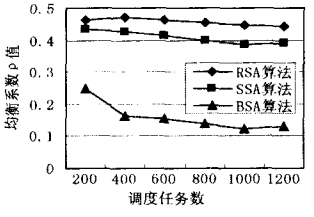

图 2 算法的负载均衡性 $R=30$，$PP=50\%$

3.3 算法可调度性分析

实时容错调度算法的可调度性通过最小处理机的个数表现，即给出一组任务集合 T，调用算法所需的处理机个数 k 越小，则说明该算法的可调度性越好，模拟实验(3)的实验参数类似于实验(1)。计算最小处理机算法请参考文献[12]。模拟实验(3)的结果如图 3 所示：从图中可知随着任务数的增加所需的处理个数要增加，因为任务越多就需要更多的处理来保证算法调度成功。同时可知在相同的任务数时 BSA 算法所需的处理机个数最多，而 SSA 算法最少。这主要是因为 BSA 算法更多考虑负载均衡性，在任务调度时总是选择当前负载最小的处理机进行处理，这就使得该处理的任务的执行时间不是最短。而 SSA 算法在任务调度时总是选择当前任务执行时间最短的处理机进行处理，所以在相同任务情况下系统总的执行时间最少。由于 PP 值对算法的可调度性影响不大，其实验结果与图 3 基本相同，本文不再给出。

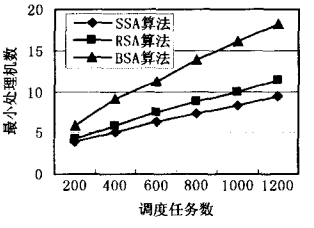

图 3 算法的最小处理机需求 $R=50$，$PP=50\%$

3.4 R 取值与算法可调度性关系

模拟实验(4)主要是讨论 R 的取值对算法的可调度性的影响。其基本实验参数与模拟实验(1)类似，我们取不同的 R 值来分析算法。图 4 表示 SSA 算法不取不同 R 时的实验结果，我们只给出了 $R=15,30,50$ 三个最有代表性的结果。从图 4 可知，随着 R 取值的增大，最小处理机需求变小，这主要是因为 R 值变大，任务的执行时间取值区间 C 在调度长度 DL 一定时变少，所以任务在系统中的总执行时间变少。RSA 与 BSA 算法也有相同的特征，所以本文不再给出。

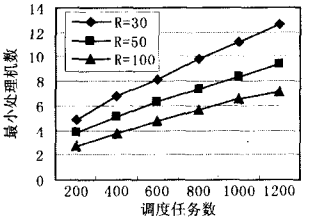

图 4 R 值与 SSA 算法最小处理机关系 $PP=50\%$，$DL=1500$

分析静态调度算法非周期任务的一个主要指标用非周期任务接收率来表示，即用 δ 来表示。δ 定义为相同条件下一个周期内在周期任务数确定情况下最多能够接收调度的非周期任务数与总非周期数的百分比，如果接收非周期任务数越多，要说明算法响应速度快，调度能力强。模拟实验方法为：首先确定处理机个数 k、性能参数 β、可靠性代价 λ、实时周期任务的 Lp 及确定实时周期任务数及周期。其次我们准备好待接收的非周期任务(模拟实验非周期数为 10 000)，同时计算出实时周期任务一个周期内在保证系统成功调度情况下最多能调度多少个非周期任务。最后通过 100 次实验最终取一个周期能调度的非周期任务数的平均值。

3.5 任务周期与算法 δ 的关系

模拟实验(5)主要是用来分析任务周期与算法对非周期任务的接收。模拟实验参数 $k=20$，$R=50$，周期任务数为 300，Lp 分别取 600，800，1100 和 1300，为了便于计算周期任务的任务周期 d 分别取 1000，1500，2000 和 2500 四组数据进行实验。分别调用 3 种算法在一个周期调度长度内能接收的非周期任务如图 5 所示。从图中可以看出任务的周期越大，在周期任务确定情况下 3 种算法对非周期任务的接收率越高，即在一个周期内对非周期任务的调度能力越强。这是因为周期越大，在周期任务数确定时，留给非周期任务的调度空间越大。同时我们还可以知道 SSA 算法的接收率最高，这主要是因为 SSA 在调度所有任务时都选择执行时间最小的处理机，在相同条件下就能调度更多的任务。

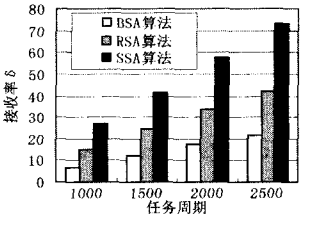

图 5 任务周期与 δ 的关系

3.6 Lp 值与算法 δ 的关系

模拟实验(6)与实验(5)类似，我们取任务周期 d 为 1500，在周期任务数确定为 300，处理机数 $k=20$，$R=50$ 时 Lp 分别取 500，750，1000 和 1250 四组数据来分析 Lp 值对算法非周期任务的接收率 δ。从图 6 可知，Lp 值在保证周期任务调度成功情况下对非周期任务接收率没有影响，如 SSA 算法在不同的 Lp 值时 δ 分别为 41.7，41.9，42.1 和 42.0，因为 Lp 值改变并不能改变任务的调度长度和周期长度。

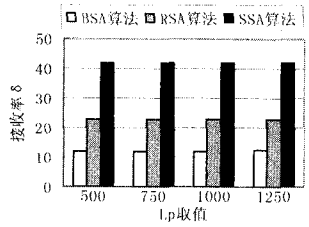

图 6 Lp 值与 δ 的关系

3.7 R 值与算法 δ 的关系

模拟实验(7)与实验(5)类似，我们取任务周期 d 为 1500，在周期任务数确定为 300，处理机数 $k=20$，Lp 取 800 时，R 分别取 30,50,75 和 100 四组数据来分析 R 值对算法非周期任务的接收率 δ。从图 7 可知，随着 R 值的增加，算法的接收率 δ 也随之增加。这主要是因为 R 变大，任务的执行时间变小，在相同的周期内就能够调度更多的非周期任务。

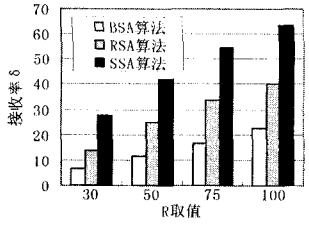

图 7 R 值与 δ 的关系

4 算法分析比较

针对异构分布式系统，本文基于不同的性能衡量标准阐述并分析了 3 种不同的算法。但它们各自都有优缺点，适合于不同情形的异构分布式实时系统。

在实际应用中选择调度算法时可以分为两种情况：(1)异构分布式系统不繁忙，即 $m+n$ 不大，且系统性能差异较大时选择 BSA 算法要优于其它两种算法。原因如下：a) $m+n$ 较小时 BSA 算法的时间复杂度与其它两种算法差别不太大；b) 在系统性能差异较大时使用其它两种算法可能出现性能越好的处理机越繁忙，反之亦然，这主要是因为性能越好的处理机在处理相同任务时所需的执行时间和可靠性代价越少，这就使得 SSA 算法和 RSA 算法在调度任务时把大量任务都调度到性能好的处理机，最终使得性能越好的处理机一直繁忙而性能较差的处理机却一直空闲。(2) 异构分布式系统繁忙即 $m+n$ 较大时，选择 SSA 算法和 RSA 算法，这是因为系统繁忙时所有处理机都可能达到最大负载，此时系统的均衡性不在重要。此时应该根据异构系统的要求选择 SSA 算法和 RSA 算法，如果系统可靠性代价差异较大且对整个系统的可靠性代价要求较高时，应该选择 RSA 算法，尽可能把任务调度到可靠性代价最小的处理机运行，以减少整个系统的可靠性代价来提高系统的可靠性。选择 SSA 算法与选择 RSA 算法同理，尽可能把任务调度到执行时间最短的处理机运行，以减少整个系统的任务总执行时间来提高系统的可调度性。综合分析 SSA 算法与 RSA 算法，在第二种情况下 SSA 算法要优于 RSA 算法，这可以从图 1 知从可靠性代价来说 SSA 算法略高于 RSA 算法，但是从可调度性实验 3 即图 3 可知 SSA 算法明显优于 BSA 算法。从总体性能来看 SSA 算法和 RSA 算法优于 BSA 算法，但 BSA 算法在特定环境中也表现出色。

结束语 基于基/副版本的容错调度算法是计算机容错系统研究的重要分支之一，其任务调度的优化问题属于 NP 难题[17,19]。本文所研究的进程调度属于静态进程调度。本文的主要贡献有：(1)提出一种异构分布式混合型调度模型，以同时调度周期与非周期实时任务，并引入一种分析负载均衡的方法。(2)基于该模型，从 3 种不同的异构系统性能衡量标准分别给出了实时容错调度算法 SSA，RSA，BSA；(3)对 3 种不同算法在不同的异构环境中进行模拟实验。实验结果分别反映出算法在可靠性、资源利用率、负载均衡性、任务特性及综合性能方面各自的优缺点。下一步的工作是：(1)研究一种在异构环境下的动态实时容错调度算法，并通过模拟实验分析动态算法的性能。(2)本文的模型假设是一种不可抢占式的混合实时任务，下一步的研究将提出一种在异构环境下可抢占的混合实时任务模型，并研究相关的算法。(3)本文没有考虑非容错需求的实时任务，下一步的研究将把非容错需求的实时任务考虑进来，使其更加接近现实。

参 考 文 献

[1] Kieckhafer R M, Walter C J, Finn A M, et al. The MAFT architecture for distributed fault tolerance[J]. IEEE Transations on Computers, 1988, 37(4): 398-405

[2] Andersson B, Jonsson J. Preemptive multiprocessor scheduling anomalies[R]. 01-9. Dep. of Computer Engineering, Chalmers University, 2001

[3] Xu L H, Bruck J. Deterministic voting in distributed systems using error-correcting codes [J]. IEEE Trans on Parallel and Distributed Systems, 1998, 9(8): 813-824

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

[5] Qinand X, Jiang H. A novel fault-tolerants cheduling algorithm for precedence constrained tasks in real-time heterogeneous systems[J]. Parallel Comput., 2006, 32(6): 331-356

[6] Al-Omari R, Somani A K, Manimaram G. An adaptive scheme for fault-toleran scheduling of soft real-time tasks in mulitprocessor systems[J]. Journal of Parallel and Distributed Computing, 2005, 65(5): 595-608

[7] Yang C H, Deconick G, Gui W H. Fault-tolerant scheduling for real-time embedded control systems[J]. Journal of Computer Science and Technology, 2004, 19(1): 191-202

[8] Zhao Qi, Qu Haitao. Research on static Fault-tolerance scheduling algorithm[J]. Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference, 2010, 5(3): 179-181

[9] Manimaran G, Murthy C S R. An efficient dynamic scheduling algorithm for multiprocessor real-time systems[J]. IEEE Trans Parallel and Distributed Systems, 1988, 9(3): 312-319

[10] Harbour M G, Klein M H, Lehoczky J P. Timing Aoalys fixes-priority scheduling of hard real-time systems[J]. Trans Software Engineering, 1994, 20(1): 13-28

[11] Lee C G, Hahn J, Seo Y M, et al. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling[J]. IEEE Trans Computers, 1998, 47(6): 700-713

(下转第 102 页)

性，该系统存在非法信息流。因为低级别进程 Low 在状态 s_0 同时发出的两个异步请求 l_{12} 响应的顺序有两种情况(情况一先位置 1 再到位置 2，情况二先位置 2 再到位置 1)，但高级别进程 $High$ 先对状态 s_0 进行高级别操作 h_1 后，会使得低级别进程 Low 同时发出的两个异步请求 l_{12} 响应的顺序确定下来(先位置 1 再到位置 2)，也就是说低级别进程可以根据请求响应的顺序来推断出高级别进程所进行的操作，即高级别进程向低级别进程泄漏信息。

结束语 本文首先回顾了已有前向可修正属性的研究成果，然后在基于状态转换系统的前向可修正属性定义的基础上，分析前向可修正属性的具体性质，提出了一种可靠且完备的前向可修正属性验证方法。该方法将不确定型系统前向可修正属性的验证问题归约为确定型系统上相应性质的验证，然后进一步归约为确定型系统的双构造上的可达性问题，进而借助于可达性检测技术完成验证。和已有的前向可修正属性“展开定理”相比，我们的方法有两方面的优点：(1)避免了“展开方法”中局部条件的构造和验证，即“展开定理”的构造，是一种完备的方法。(2)系统在不满足前向可修正属性时，能够给出属性失效的反例，反例的给出对非法信息流的消除和控制具有直接的帮助。最后，本文通过一个磁臂隐通道的例子说明了如何将该方法应用在实际的隐通道分析中。

将来的工作主要包括：(1)将本文提出的算术验证方法拓展到其他信息流安全属性(如不可演绎属性、限制属性、隔离属性等)，提出各属性具体验证算法。(2)研究模型检测中的谓词抽象技术，并利用其提炼出系统的有限状态机模型。

参 考 文 献

[1] 刘文清，韩乃平，陶喆. 一个安全操作系统 Slinux 隐蔽通道标识与处理[J]. 电子学报，2007，35(1)：153-156

[2] 王昌达，鞠时光，周从华，等. 一种隐通道威胁审计的度量方法[J]. 计算机学报，2009，32(4)：751-762

[3] 卿斯汉，沈昌祥. 高等级安全操作系统的设计[J]. 中国科学 E，2007，37(2)：238-253

[4] Denning D E. A Lattice Model of Secure Information Flow[J]. Communication of the ACM，1976，19(5)：236-242

[5] Millen J K. Security Kernel Validation in Practice[J]. Communication of the ACM，1976，19(5)：243-250

[6] Goguen J A，Meseguer J. Security policies and security models [C]// Proceedings of the 1982 IEEE Computer Society Symposium on Research in Security and Privacy. Los Alamitos：IEEE Computer Society Press，1982：11-20

[7] D'Souza D，Raveendra H，Janardhan K. On the decidability of modelchecking information flow properties[C]// 4th International Conference on Information Systems Security. ICISS 2008，December 2008：26-40

[8] 张原，史浩山. 信息安全模型研究[J]. 小型微型计算机系统，2003，24(10)：1878-1881

[9] 赵保华，陈波，陆超. 概率信息流安全属性分析[J]. 计算机学报，2006，129(8)：1447-1452

[10] Zhou Cong-hua，Chen li，Ju Shi-guang. Petri nets based noninterference analysis[J]. Journal of Computational Information Systems，2009，15(4)：1231-1239

[11] Millen J K. Hookup Security for Synchronous Machines[C]// Proceedings of The Computer Security Foundations Workshop III. IEEE Computer Society，1990：84-90

[12] McCoullough D. Specifications for multi-level security and a hookup property[C]// Proceedings of the Symposium on Research in Security and Privacy. New York：IEEE Computer Society，1987：161-166

[13] McCoullough D. Noniterference and the composability of security properties[C]// Proceedings of the IEEE Symposium on Research in Security and Privacy. New York：IEEE&Technical Committee on Security and Privacy，1988：177-186

[14] Johnson D，Thayer F. Security and the compositon of machines [C]// Proceedings of the Computer Security Foundations Workshop. IEEE Press，1988：14-23

[15] Jonathan K M. Unwinding forward correctability [C]// Proceedings of Computer Security Foundations Workshop VII. Franconia，1994：2-10

[16] 周伟，尹青，郭金庚. 计算机安全中的无干扰模型[J]. 计算机科学，2005，32(2)：159-165

[17] van der Heydena R，Zhanga C. Algorithmic Verification of Noninterference Properties[J]. Electronic notes in Theoretical Computer Science，2007，168(SPEC)：61-75

[18] 刘志峰，鞠时光，李沛. 基于操作语义的磁臂隐通道分析[J]. 计算机应用研究，2007，24(11)：157-160

(上接第 92 页)

[12] Yin Jin-yong，Guo Guo-chang，Wu Yan-xia. A Hybrid Fault-tolerant Scheduling Algorithm of Periodic and Aperiodic Real-time Tasks to Partially Reconfigurable FPGAs[J]. Intelligent Systems and Applications. ISA 2009：International Workshops，2009，6：1-5

[13] Qin Xiao，Han Zong-feng，Pang Li-ping，et al. Design and Performance Analysis of a Hybrid Real-Time Scheduling Algorithm with Fault-Tolerance[J]. Journal of Software，2000，11(5)：686-693

[14] Liu Huai，Lin Qiu-shi，Huang Jian-xin，et al. A Novel Fault-Tolerant Scheduling Algorithm for Periodic Tasks of Distributed Control Systems[J]. 2009 Chinese Controland Decision Conference，2009，8：1584-1588

[15] Qin Xiao，Han Zong-feng，Pang Li-ping. Real-Time Scheduling with Fault-Tolerance in Heterogeneous Distributed Systems [J]. Chinese Journal of Computers，2002，25(1)：49-56

[16] Wei Luo，Xiao Qin，Bellam K. Reliability-Driven Scheduling of Periodic Tasks in Heterogeneous Real-Time Systems[C]// the 7th ACIS International Conference on Software Engineering，Artificial Intelligence，Networking and Parallel/Distributed Computing(SNPD '07). China，2007：640-645

[17] Guo Hui，Wang Zhi-guang，Zhou Jian-li. Load Balancing Based Process Scheduling with Fault-Tolerance in Heterogeneous Distributed Systems[J]. Chinese Journal of Computers，2005，28(11)：1807-1816

[18] Luo Wei，Yang Fu-min，Pang Li-ping，et al. A Real-Time Fault-Tolerant Scheduling Algorithm of Periodic Tasks in Heterogeneous Distributed Systems[J]. Chinese Journal of Computers，2007，30(10)：1740-1749

[19] Garey R，Johnson D S. Computers and Intractability：a Guide to the Theory of NP-Completeness[M]. New York：W. H. Freeman Company，1979

[20] Zhang Kunlong，Qin Xiao，Han Zong-fen，et al. Study of the Model for Fault-Tolerant Scheduling in Heterongeneous Distributed Real-Time Systems [J]. J. Huazhong Univ. of Sci. & Tech.，2000，28(8)：17-19