该文于 2001 年 3 月 27 日被<计算机学报>录用

基于异构分布式系统的实时容错调度算法*

秦 啸　　韩宗芬　　庞丽萍

（华中科技大学计算机科学与技术学院 武汉 430074）

摘要 目前文献中研究的实时容错调度算法都是基于同构分布式系统，该系统中的所有处理机完全相同。本文首先建立了一个基于异构分布式系统实时容错调度模型，异构分布式系统中的各个处理机均不相同。基于该异构分布式系统模型，本文引入了可靠性代价（Reliability cost）概念，并提出两种静态实时容错调度算法（RTFTNO 和 RTFTRC），用于调度周期性实时容错任务。算法 RTFTRC 在调度任务时，尽量使系统的可靠性代价最小；而算法 RTFTNO 在调度实时任务时，没有考虑系统的可靠性代价。本文详细讨论了两种调度算法的性能。性能模拟实验分别比较了两个算法的可靠性代价，超时比率和可调度性；并研究了任务的计算时间与可靠性代价的关系以及调度长度阈值与最小处理机个数的关系。实验结果表明，算法 RTFTRC 的性能优于算法 RTFTNO。

关键字 容错，实时调度，异构分布式系统，模拟实验，性能分析

中国图书分类号 TP 302.8

实时系统的一个重要需求是实时任务必须在规定的时限完成，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[1][2]。所以实时技术与容错技术必须结合在一起进行研究[3]。分布式投票技术[4]，反转恢复技术[5]和基/副版本技术[6]都是典型的容错技术， 然而这些容错技术由于没有考虑实时问题，所以不能很好适应实时系统要求。论文[7][8]分别研究了速率单调算法和动态实时调度算法，论文[9][10]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。硬件的冗余是解决实时系统中永久性错误的有效途径[11][12]。实时容错调度算法是一种通过软件解决实时容错问题的有效方法，该方法的优点是不需要额外的硬件代价实现实时系统的可靠性。文献[13]研究了一种非精确计算（imprecise computations）的实时调度算法，该算法通过非精确技术来满足实时容错需求。文献[14][15]分别研究了实时容错调度算法，这种算法则是通过基/副版本技术在实时调度算法中增加容错能力。基于基/副版本技术，我们提出一种实时容错调度算法，可以同时调度具有容错需求和无容错需求的实时任务[16]。在此基础上，我们深入研究了混合型实时容错调度算法[17][18]。然而，文献[14-18]中的多机系统或分布式系统都是同构模型，即系统中的处理机完全相同。随着高速网络和高性能 PC/工作站机的发展，异构分布式系统在高性能计算中的应用越来越广泛。例如，异构分布式系统已经被应用在数字信号处理，医学图像处理，天气预报系统以及其他很多计算密集/数据密集型的应用系统中。这要求我们针对异构分布式系统提出高效的实时容错调度算法。文献[14-18]中提出的实时容错调度算法是基于同构系统，所以不能适应异构分布式系统的特点。本文首先提出一个异构分布式系统模型，引入可靠性代价概念，并提出异构分布式系统的实时容错调度算法。性能模拟实验表明，由于算法 RTFTRC 在调度任务时，考虑了异构分布式系统的特点，使系统的可靠性代价尽量最小，所以该算法在异构分布式系统中具有很好的性能。

1 系统模型及前提假设

实时任务集合表示为，$T = \{t_1, t_2,...,t_n\}$。假设集合中的各个任务之间是相互独立的，且不互相强占执行。实时任务 t_i

* 本文受国防预研基金资助。**秦啸**，男，1974 年生，硕士，主要研究领域为实时系统，实时容错技术，调度理论，网络通信和数据库管理系统。**韩宗芬**，女，1950 年生，副教授，主要研究领域为分布式实时系统，操作系统，软件工程。**庞丽萍**，女，1944 年生，教授，主要研究领域为并行分布式程序设计，分布实时系统。
通信地址：Xiao Qin, 2000 J Street APT #1, Lincoln, Nebraska, 68510, USA.　Email: xqin@cse.unl.edu　　Homepage: http://www.cse.unl.edu/~xqin

定义为一个四元组，$t_i = (p_i, d_i, t_i^p, t_i^b)$，其中 p_i 和 d_i 分别是周期和截止期限，t_i^p 和 t_i^b 是任务的基版本和副版本。假设两个版本 t_i^p 和 t_i^b 的程序代码完全相同。$t_i^p = (C_i, s_i^p, \rho_i^p)$，$t_i^b = (C_i, s_i^b, \rho_i^b)$，其中 s_i^p 与 s_i^b 是开始执行的时间，ρ_i^p 和 ρ_i^p 指明两个版本所分配的处理机。在异构分布式系统中，每个任务 t_i 在不同的处理机上有不相同的执行时间，所以为每一个任务 t_i 定义一个计算时间向量 C_i，$C_i = [c(i, 1),\ldots,c(i, m)]$，$c_{ij}$ 表示版本 t_i^p 和 t_i^b 在处理机 p_j.上的执行时间。异构分布式系统中描述为一个处理机集合，$\Omega = \{P_1, P_2,..., P_m\}$。$P_i = (\Delta_i, \xi_i, \lambda_i)$，其中$\Delta_i$ 是一个任务集合，Δ_i 中的任务均为分配到处理机 P_i 上的任务，ξ_i 是处理机的调度长度，λ_i 是处理机的失效率。假设各个处理机之间的出错过程是相互独立，并且符合泊松过程。在系统无处理机失效时，可靠性代价（Reliability cost）$RC_0(\Omega)$的定义如下，文献[19]给出了相似定义。

$$RC_o(\Omega) = \sum_{j=1}^{m} \sum_{t_i^p \in \Delta_i} \lambda_j c(i, j)$$

当处理机 P_i 出错失效时，系统的可靠性代价 $RC_1(\Omega, i)$定义如下，

$$RC_1(\Omega, i) = \sum_{j=1, j\neq i}^{m} \sum_{t_k^p \in \Delta_i} \lambda_j c(k, j) + \sum_{j=1}^{m} \sum_{t_k^p \in \Delta_i \wedge t_k^b \in \Delta_j} \lambda_j c(k, j)$$

$$RC_1(\Omega) = \sum_{i=1}^{n} (\lambda_i RC_1(\Omega, i) / \sum_{j=1}^{m} \lambda_j)$$

$RC_1(\Omega)$是系统在一个处理机失效时的可靠性代价，$RC_1(\Omega)$通过 $RC_1(\Omega, i)$推导出,

要使实时任务满足其截止期限，必须保证该任务的基版本和副版本的完成时间在截止期限之前。如果一组实时任务的调度结果能够容忍一个处理机失效，则称该调度为可容忍一处理机失效的调度，简称为 1TFTS（1-Timely-Fault-Tolerant schedule）。给出一组实时任务集合 T 和处理机集合Ω，实时容错调度的目标是产生一个 1TFTS，并使系统的可靠性代价尽量的低。调度模型假设所有的任务具有相同的截止期限和周期，即：$\forall i, j\in[1, n], i \neq j \rightarrow (d_i = d_j = p_i = p_j = DL)$。在模型中，

如果 t_i^b 和 t_j^b 都调度到同一个处理机上，则 t_i^b 和 t_j^b 之间没有时间上的重叠， 即$\forall i, j \in [1, n], i \neq j: \rho_i^b = \rho_j^b \rightarrow ((s_i^b + c(i, \rho_i^b) \leq s_j^b) \vee (s_j^b + c(j, \rho_j^b) \leq s_i^b))$。我们的调度算法很容易满足这点要求。该模型的三个属性描述如下；

性质 1 系统容忍一个处理机失效，则所有实时任务的基版本和副版本的计算时间之和小于或等于截止期限，即，(One processor failure is tolerated) $\rightarrow \forall i \in [1, n]: c(i, \rho_i^p) + c(i, \rho_i^b) \leq DL)$。

性质 2 系统容忍一个处理机失效，则每一个实时任务的基版本和副版本调度到不相同的处理机上，且两个版本间无时间上的重叠；该性质可形式化描述为, (One processor failure is tolerated) $\rightarrow \forall i\in[1, n]:(\rho_i^p \neq \rho_i^b) \wedge (s_i^p + c(i, \rho_i^p) \leq s_i^b)$。

性质 3 对于异构分布式系统中的每一个处理机 P_j， 满足：$\forall j\in[1, m]: \sum_{t_i^p \in \Delta_j} c(i, j) + \sum_{t_i^b \in \Delta_j} c(i, j) \leq DL$。

2 实时容错调度算法

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，本节提出两个启发式算法。算法 RTFTNO 是一个经典的实时容错调度算法，该算法在调度任务时，仅仅考虑如何使所有处理机的调度长度尽量小，由于该算法没有考虑到异构分布式系统的特点，它只适用于同构分布式系统，而在异构分布式系统中的性能比较差。第三节将详细研究该算法的性能。算法 RTFTNO 的描述如下，

算法 RTFTNO(输入: T, Ω, DL; **输出:** sch_success, Ω, $RC_0(\Omega)$) {

1. 初始化处理机集合Ω, 任务集合 T 和系统的可靠性代价 $RC_0(\Omega)$;
2. 对于每一个实时任务基版本 t_i^p { /* 首先调度实时任务的基版本 */
3. 选择 j, 满足：$\forall k \in [1, m]: \xi_j \leq \xi_k$； /* t_i^p 被调度到处理机 P_j */
4. $\Delta_j = \{t_i^p\} + \Delta_j$； $\rho_i^p = j$； $s_i^p = \xi_j$； $\xi_j = \xi_j + c(i, j)$； $RC_0(\Omega) = RC_0(\Omega) + \lambda_j \times c(i, j)$;
   }
5. 对于每一个实时任务副版本 t_i^b { /* 调度实时任务副版本 */
6. 选择 $j \neq \rho_i^p$，满足：$\forall k \in [1, m]: k \neq \rho_i^p \wedge \xi_j \leq \xi_k$; /* t_i^b 被调度到处理机 P_j */

7. $\Delta_j = \{t_i^p\} + \Delta_j$; $\rho_i^p = j$; $s_i^p = \xi_j$; $\xi_j = \xi_j + c(i, j)$; $RC_0(\Omega) = RC_0(\Omega) + \lambda_j \times c(i, j)$;
8. if ($\xi_j > DL$) return(FAIL);
}
9. return(SUCCESS);
}

与算法 RTFTNO 相比，算法 RTFTRC 在不增加系统硬件代价的情况下，提高了系统的可靠性。算法 RTFTRC 的调度目标有两个，减少处理机的调度长度和减少系统的可靠性代价。对于每一个基版本 t_i^p，当其调度到处理机 P_j 上时，使系统中的可靠性代价增加尽量的少。然而，当一个处理机的调度长度 Lp 过大后，可能会造成一些任务的副版本无法在规定的截止期限内完成。对每一个处理机的调度长度进行限制将有利于解决这个问题。算法 RTFTRC 在调度任务的副版本时，要求满足三个情况：(1) 副版本所在的处理机与其基版本所在的处理机是不同的；(2) 副版本被调度到处理机 P_j 上时，使系统中的可靠性代价增加尽量的少； (3) 副版本可以在截止期限内完成。在给出算法 RTFTRC 之前，为每一个任务 t_i,定义一个可靠性代价向量，$RCV_i = [rcv_{i1},\ldots,rcv_{im}]$，其中 $rcv_{ij} = (rc_{ij}, \rho_{ij})$。$rcv_{ij}$ 可以通过向量 C_i 和处理机集合Ω推导出，$rc_{ij} = c(i, \rho_{ij}) \times \lambda_{\rho ij}$。$RCV_i$ 中的元素按其可靠性代价非减的顺序进行排序，形式化描述排序结果为，$\forall i \in [1,n]: \forall j,k \in [1,m] (j < k \rightarrow rcv_{ij} \le rcv_{ik})$。算法 RTFTRC 的描述如下，

算法 RTFTRC(**输入:** T, Ω, Lp, DL; **输出:** sch_success, Ω, $RC_0(\Omega)$) {
1. 初始化处理机集合Ω, 任务集合 T 和系统的可靠性代价 $RC_0(\Omega)$;
2. 通过向量 C_i 和处理机集合Ω推导出每一个任务的可靠性代价向量 RCV_i;
3. 对每个任务的可靠性代价向量进行排序，使 RCV_i 中的元素满足可靠性代价非减的顺序，即排序后满足：
$\forall i \in [1,n]: \forall j,k \in [1,m] (j < k \rightarrow rcv_{ij} \le rcv_{ik})$;
4. 对于每一个实时任务基版本 t_i^p { /* 首先调度实时任务的基版本 */
5. for (j = 1 to m) and (没找到合适的处理机) {/* 选择合适的处理机 P_j，使该基版本调度到 P_j 上后， */
6. $k = \rho_{ij}$; /* 系统可靠性代价的增加尽量小，且调度长度不超过 Lp */
7. if ($\xi_k + c(i, k) < Lp$) /* t_i^p 被调度到处理机 P_j */
8. {$\Delta_k = \{t_i^p\} + \Delta_k$; $\rho_i^p = k$; $s_i^p = \xi_k$; $\xi_k = \xi_k + c(i, k)$; $RC_0(\Omega) = RC_0(\Omega) + rc_{ij}$; 标志已找到合适的处理机；}
}
9. if (没找到合适的处理机) return(FAIL);
}
10. 对于每一个实时任务副版本 t_i^b { /* 调度实时任务副版本 */
11. for (j = 1 to m) and (没找到合适的处理机) {/* 选择合适的处理机 P_j，使该副版本调度到 P_j 上后， */
12. $k = \rho_{ij}$; /* 系统可靠性代价的增加尽量小，且调度长度不超过 DL */
13. if (($MAX(\xi_k, s_i^p + c(i, \rho_i^p)) + c(i, k) < DL \wedge (k \ne \rho_i^p)$){ /* t_i^b 被调度到处理机 P_j */
14. $\Delta_k = \{t_i^b\} + \Delta_k$; $\rho_i^b = k$; $s_i^b = MAX(\xi_k, s_i^p + c(i, \rho_i^p))$;
15. $\xi_k = s_i^b + c(i, k)$; $RC_0(\Omega) = RC_0(\Omega) + rc_{ij}$; 标志已找到合适的处理机；
}
}
16. if (没找到合适的处理机) return(FAIL);
}
17. return(SUCCESS);
}

定理 1. 算法 RTFTRC 的计算时间复杂度为 $O(nm(3+\log m))$，其中 n 和 m 分别是实时任务个数与处理机个数。

证明： 为每一个实时任务生成可靠性向量的需要时间为 $O(nm)$； 对每个任务的可靠性代价向量进行排序的时间为 $O(nm\log m)$；调度基版本和副版本的最大时间上限都为 $O(nm)$； 故算法 RTFTRC 的所需时间为 $O(3nm + nm\log m)$, 即有计算时间复杂度为 $O(nm(3+\log m))$。■

定理 2. 如果算法 RTFTRC 可以成功产生一个调度结果，则所产生的调度为 1TFTS 。

证明：根据性质 1-2，需要证明对于每一个任务 t_i:(a) 副版本所在的处理机与其基版本所在的处理机是不同的； (b) 两个版本 t_i^p 和 t_i^b 之间无时间上的重叠； (c) 基版本和副版本都能在截止时限之前完成。即必需证明， $\forall i \in [1, n]: (\rho_i^p \ne \rho_i^b) \wedge (s_i^p + c(i, \rho_i^p) \le s_i^b) \wedge (s_i^b + c(i, \rho_i^b \le DL)$。

(1) 因为 $\forall i \in [1,n]: \rho_i^b = k \wedge k \neq \rho_i^p$ (参见算法第 12 步和 13 步)，得到 $\forall i \in [1, n]: (\rho_i^p \neq \rho_i^b)$；

(2) $\forall i \in [1, n]: s_i^b = MAX(\xi_k, s_i^p + c(i, \rho_i^p))$, (参见算法第 14 步)，所以，$\forall i \in [1, n]: s_i^p + c(i, \rho_i^p) \leq s_i^b$；

(3) $\xi_k = MAX(\xi_k, s_i^p + c(i, \rho_i^p)) + c(i, k) = s_i^b + c(i, k)$ and $MAX(\xi_k, s_i^p + c(i, \rho_i^p)) + c(i, k) < DL$, , (参见算法第 13，14 和 15 步), 可以保证，$\forall i \in [1, n]: s_i^b + c(i, k) \leq DL$，因为 $k = \rho_i^b$，所以 $\forall i \in [1, n]: s_i^b + c(i, \rho_i^b) \leq DL$。

综合(1)-(3)，如果算法 RTFTRC 可以成功产生一个调度结果，则所产生的调度为 1TFTS 。定理 2 得证。■

3 算法性能评价

我们使用带有 300MHz Pentium II CPU 的 PC 机进行模拟试验。模拟程序的语言是 C++，编译程序是 Borland C++3.0。模拟实验的步骤如下： (1) 首先确定系统的截止期限 DL 和调度长度阈值 Lp；(2) 其次是选择处理机的个数以及各个处理机的失效率；(3) 然后，随机生成每一个任务的计算时间向量 C_i， C_i 中的每个元素 c_{ij} 满足[5, 100]上的均匀分布； (4) 最后分别调用算法 RTFTNO 和 RTFTRC。如果算法成功产生调度结果，则在调度算法执行完成后，可以根据本文对可靠性代价的定义计算出相应的 $RC_0(\Omega)$和 $RC_1(\Omega)$。对于每一组相同的实验输入数据，我们分别独立的产生 10000 组任务集合，并分别运行在两个算法上，模拟程序将计算出 $RC_0(\Omega)$和 $RC_1(\Omega)$的平均值。在性能评价实验中，算法的可调度性通过指标超时比率 PMD（ percentage of missed deadlines）体现出，PMD 的定义如下， PMD = Number of schedules which is not schedulable * 100 / Total schedules，其中本实验的 total schedules 设为 10000。

3.1 可靠性代价比较

本节比较两种调度算法的可靠性代价。模拟实验的基本参数为：DL= 1400, Lp = 600 且 m = 5。五个处理机的失效率分别设为 0.9, 0.95, 1, 1.05 和 1.10，其基本单位是 10^{-6} 次/小时。由于本实验的目的是研究算法所产生的系统可靠性代价，所以实验参数的设定将保证，对每一个任务集合，调度算法都能成功的产生一个 1TFTS 调度，即本实验中的 PMD 值被控制为零。对于算法 RTFTNO， 其 RC_0 和 RC_1 的值差别很小，所以算法 RTFTNO 的 RC_1 值在图 1.中省略。图 1 显示出随着任务个数的增加，系统的可靠性代价成线性增加。在负载完全相同的条件下，算法 RTFTRC 的可靠性代价大大低于算法 RTFTNO，（例如，当任务个数分别为 40 和 55 时，算法 RTFTRC 的可靠性代价分别比算法 RTFTNO 的要低 61.4% 和 61.1%）。因为算法 RTFTRC 在调度是考虑降低系统的可靠性代价，而算法 RTFTNO 在调度是忽视这一因素。对于算法 RTFTRC 而言，其 RC_0 和 RC_1 的值很接近；然而在相同的负载情况下， RC_1 的值比 RC_0 的值稍微大一点，（例如，当任务个数为 30 时，RC_1 与 RC_0 的值分别为 0.74×10^{-3} 和 0.64×10^{-3}；当任务个数为 50 时，RC_1 与 RC_0 的值分别为 1.23×10^{-3} 和 1.06×10^{-3}）。从可靠性代价来看，算法 RTFTRC 的性能优于算法 RTFTNO。

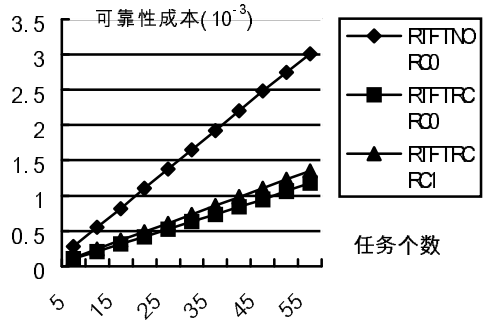

图1 算法RTFTNO与RTFTRC的比较，
m = 5, DL = 1400, Lp = 600

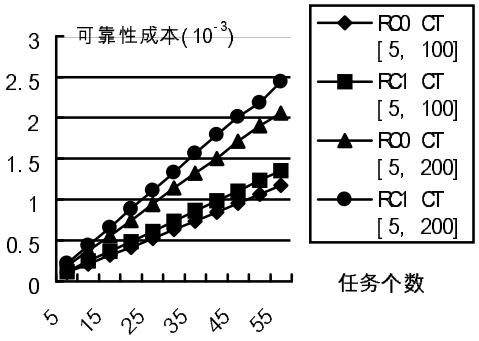

图2 计算时间与可靠性成本的关系
m = 5, DL = 1400, Lp = 600

3.2 计算时间与可靠性代价的关系

本实验是研究任务的计算时间与系统的可靠性代价之间的相互关系。这里只讨论算法 RTFTRC，因为算法 RTFTNO 与算法 RTFTRC 具有相似的性质。实验的基本参数与实验 3.1 的基本相同，不同的是每一个任务的计算时间向量 C_i 中的元素 c_{ij} 满足[5, 200]上的均匀分布。图 2 描绘了模拟实验结果，容易观察到，如果任务个数相同，当计算时间满足[5, 200]的均匀分布时系统的可靠性代价，比计算时间满足[5, 100]的均匀分布时要高（例如，如果任务个数为 45，当计算时间满足[5, 100]的均匀分布，RC_1 与 RC_0 的值分别为 1.10×10^{-3} 和 0.95×10^{-3}；当计算时间满足[5, 200]的均匀分布，RC_1 与 RC_0 的值分别为 2.01×10^{-3} 和 1.71×10^{-3}）。这说明在任务个数相同时，任务的计算时间越大，系统所需的可靠性代价越高。因为系统的可靠性代价定义为处理机的失效率与任务计算时间的乘积，故可靠性代价随任务的计算时间的增加而增大。本实验的模拟结果再次证明实验 3.1 的结论，即：在相同的负载情况下，RC_1 的值比 RC_0 的值稍微大一点。

3.3 超时比率的比较

模拟实验的基本参数与实验 3.1 相同， 本实验研究调度参数对可调度性指标 PMD 的影响。在表 1 中观察算法 RTFTNO，发现其 PMD 值为零，当 n 小于 56 时；而当 n 大于 70 时，PMD 将恒定为 100%。与算法 RTFTNO 相比，算法 RTFTRC 的 PMD 仅仅在区域[97, 103]发生变化；PMD 为零，当 n 小于 97 时；而如果 n 超过 103，PMD 等于 100%。表 1 的数据同时显示两种算法的 PMD 值在变化区域增加很快。该结果说明在相同的实验前提下，实时任务越多，算法的可调度性越小。实验 3.4 将进一步讨论可调度性问题。本实验数据证明，对于性能指标 PMD 而言，算法 RTFTRC 的性能优于算法 RTFTNO。因为如果两个算法具有相同的 PMD 值，则算法 RTFTRC 可以调度的任务个数比算法 RTFTNO 的任务个数要多。例如在 PMD 值为 1%的情况下，算法 RTFTRC 可以调度的任务个数是 97，然而算法 RTFTNO 的任务个数只有 56。从超时比率这项性能指标而言，算法 RTFTRC 的性能优于算法 RTFTNO。

表 1 算法 RTFTNO 和算法 RTFTRC 的 PMD 值比较 (DL = 1400, Lp = 700, m = 5)

n	56	58	60	62	64	66	68	70
RTFTNO	1%	8%	24%	50%	74%	89%	97%	99%
n	97	98	99	100	101	102	103	
RTFTRC	1%	2%	3%	5%	35%	97%	99%	

3.4 可调度性的比较

实时容错调度算法的可调度性通过最小处理机个数表现，即：给出一组任务集合，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。求解最小处理机个数算法（Find Minimal Number of Processors-FMNP）的设计请参见论文[18]，当给出任务集合，该算法可以为算法 RTFTNO 及 RTFTRC 分别计算出满足实时容错调度需求的最小处理机个数。为方便算法 FMNP 求解最小处理机个数，不妨在本模拟实验中设所有处理机的失效率均为 10^{-6} 次/小时。本模拟实验的基本参数与实验 3.2 的相同，实验结果如图 3 所示。该实验结果首先表明，对于两种算法而言，随着任务个数的增加，最小处理机个数随之增加。因为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任务。其次，实验证明算法 RTFTRC 的可调度性比算法 RTFTNO 的要好，而且任务的计算时间越长，RTFTRC 的性能比 RTFTNO 的可调度性越高。因为在相同的实验参数条件下，算法 RTFTRC 所需的最小处理机个数比算法 RTFTNO 的要小。例如，当有 100 个任务时，如果计算时间满足[5, 100]的均匀分布，RTFTNO 的最小处理机个数为 8.65，而 RTFTRC 的仅为 5.36；如果计算时间满足[5, 200]的均匀分布，RTFTNO 和 RTFTRC 的最小处理机个数分别为 16.64 和 7.09。本实验的第三个结论是，在任务个数固定的前提下，两个调度算法的最小处理机个数随着任务计算时间增加而增加。因为系统中的任务个数固定时，任务的计算时间越长，系统需完成的工作量越大，也就需要更多的处理机来保证在截止时限之前完成所有的大工作量。

3.5 调度长度阈值与最小处理机个数的关系

调度长度阈值不影响算法 RTFTNO 的性能，但该值决定了调度算法 RTFTRC 的性能好坏。本实验的目的是研究算法 RTFTRC 中，调度长度阈值与最小处理机个数的关系。实验参数与 3.1 基本相同，计算时间满足[5, 200]上的均匀分布。分别设任务个数为 40，60 和 100，进行三组实验。从图 4 的实验结果中可看出，随着 Lp 值的增加，最小处理机个数的值随之减小。当调度长度阈值的值大于 700 后，最小处理机个数进入相对平稳阶段。例如：当 Lp 值大于 700 后，如果任务个数为 40，则最小处理机个数在 4.48 到 4.50 之间；如果任务个数为 100，则最小处理机个数在 7.07 到 7.08 之间。这说明在调度算法 RTFTRC 中，将参数 Lp 值设为大于 DL/2 后，调度长度阈值 Lp 对算法 RTFTRC 的影响不大。所以，作为传统分布式系统调度算法负载平衡参数的调度长度阈值，在算法 RTFTRC 中的意义比不是很大。

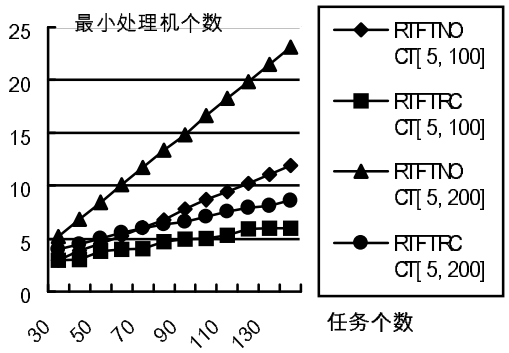

图3 算法RTFTNO与RTFTRC的最小处理机个数的比较 DL = 1400, Lp = 600

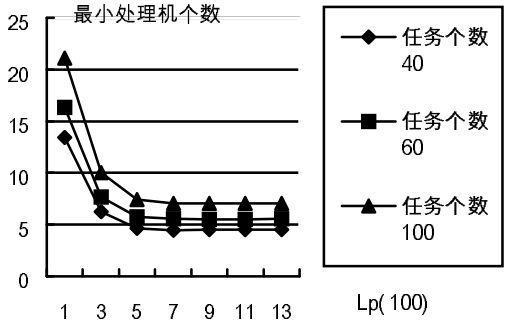

图4 算法RTFTRC的最小处理机个数与调度长度阈值Lp之间的关系 DL = 1400, CT [5, 200]

4 结束语

已有一些论文研究了实时容错调度算法，然而这些算法都是基于同构分布式系统的。同构分布式系统中的处理机完全相同，而在异构分布式系统中，各个处理机有不同的失效率，每个任务在不同的处理机上有不同的计算时间。本文的创新是提出了基于异构分布式系统的实时容错调度算法，并对调度算法的性能进行了模拟与评价。实验结果表明，算法 RTFTRC 的性能比算法 RTFTNO 的要好，算法 RTFTRC 更适合异构分布式系统。本文中研究的算法 RTFTNO 与 RTFTRC 是两个静态调度算法，这是我们基于异构分布式系统实时容错调度研究的第一步工作。正在进行的研究工作是，在此工作的基础上，结合论文[18]提出的混合调度技术，研究基于异构分布式系统的混合型实时容错调度算法。下一步的工作还包括，提出基于异构分布式系统的高效率实时容错调度算法，该算法的处理机利用率和算法的可调度性都将比本文提出的算法 RTFTRC 的要高。

致谢 Hong Jiang 教授, Steven Goddard 博士, 和 Olaf Barheine 与我们深入讨论了实时容错调度的问题，刘键教授和尹浩同学对本文第一版本提出大量有价值的建议。对他们给予的学术上的指导和帮助表示衷心地感谢。

参考文献

[1] Kieckhafer, R.M., C.J. Walter, A.M. Finn, and P.M. Thambidurai. The MAFT Architecture for distributed fault tolerance. IEEE Transactions on Computers, 1988, 37(4):398~405

[2] 1M. Factor. Real-Time Data Fusion in the Intensive Care Unit. IEEE Computer, 1991, 24(11): 45-54

[3] Shin, K.G., G.Koob, and F.Jahanian. Fault-tolerant in real-time systems. IEEE Real-time Systems Newsletter, 1991, 7(3) : 28-34

[4] 2Xu. Lihao and Bruck J. Deterministic Voting in distributed Systems Using Error-Correcting Codes. IEEE Transactions on

Parallel and Distributed Systems, 1998, 9(8):813~824
[5] 3Lin T.H. and Shin K.G. Damage Assessment for Optimal Rollback Recovery. IEEE Transactions on Computers, 1998, 47(5):603~613
[6] 4Davoli R. Giachini L. A. and Babaoglu O. Parallel Computing in Networks of Workstations with Paralex. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(4):371~384
[7] 9M.H.Klein, J.P.Lehoczky and R.Rajkumar. Rate-Monotonic Analysis for Real-time Industrial Computing. IEEE Computers, 1994, 27(1):24-33
[8] 11Manimaran G. and 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
[9] 8Harbour M.G, Klein M.H and Lehoczky J.P. Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems. IEEE Transactions on Software Engineering, 1994, 20(1):13~28
[10] 10Lee 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] H.Kopetz and G. Grunsteidl. TTP-A Protocol for Fault-Tolerant Real-Time Systems. Computer, 1994, 27(1):14-23
[12] K.P. Birman. The Process Group Approach to Reliable Distributed Computing. Comm. ACM, 1993, 39(12):37-53
[13] W.K.Shih, J.W.Liu, and J.Y.Chung. Fast algorithms for scheduling imprecise computations. Real-Time System Symposium, 1989, 12-19
[14] Y.Oh and S.H.Son. Multiprocessor support for real-time fault-tolerant scheduling. IEEE Workshop on Architecture Aspects of Real-Time System, San Antonio, Texas, 1991, 76-80
[15] Y.Oh and S.H.Son. An algorithm for real-time fault-tolerant scheduling in multiprocessor systems. 4th Euromicro Workshop on Real-Time Systems, Athens, Greece, 1992, 190-195
[16] Xiao Qin, Z.F Han, L.P Pang, and S.L Li. A Static Scheduling Algorithm for Real-time and Fault-tolerant tasks in Multiprocessor Systems. *Proc. of the Fifth International Conference for Young Computer Scientists*, Vol.2, pp. 765, International Academic Publishers, Nanjing, Aug., 1999
[17] Xiao Qin, Liping Pang, Shengli Li and Zongfen Han. Efficient Scheduling Algorithm with Fault-tolerance for Real-time Tasks in Distributed Systems. *Proc. of the Fifth International Conference for Young Computer Scientists*, Vol.2, pp.721-725, International Academic Publishers, Nanjing, Aug, 1999.
[18] 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, 10(5).
(Qin Xiao, Han Zong-fen, Pang Li-ping, and Li Sheng-li. Design and Performance Analysis of A Hybrid Real-timeScheduling Algorithm with Fault-Tolerance. Journal of Software. 2000, 10(5))
[19] S.M. Shatz, J.P. Wang, and M.Goto. Task Allocation for Maximizing Reliability of Distributed Computer Systems. IEEE Trans. Computers, 1992, 41(9):1156-1168.

Real-time Scheduling with Fault-tolerance in Heterogeneous Distributed Systems

Xiao Qin *Zongfen Han* *Liping Pang*
School of Computer Science and Technology, Huazhong Univ. of Sci. & Tech.Wuhan Hubei, 430074

Abstract Abstract Due to the wide availability of high-speed network connectivity and cheap but powerful PCs and workstations, heterogeneous distributed systems are becoming a viable and cost-effective high-performance computing platform. Heterogeneous systems are increasingly being used for the management and control of a variety of applications such as digital signal processing, high-definition television, medical imaging, seismic and whether prediction systems, and other computation and data intensive applications. Some real-time fault-tolerant scheduling algorithms have been studied in the literatures. These algorithms, however, are devised for homogeneous distributed systems, which have identical processors. The heterogeneous systems, on the other hand, consist of processors with various computation powers. Since the scheduling is a key factor in obtaining high performance in heterogeneous systems, scheduling algorithms play an important role in the efficiency of heterogeneous systems. This paper presents two real-time scheduling algorithms, RTFTNO and RTFTRC, that are able to map tasks onto processors and order their execution so that tasks' real-time requirements are satisfied and a minimum schedule length, when attainable, is given. We introduce the reliability cost into the scheduling algorithm in heterogeneous systems. The RTFTRC algorithm allocates each task to the processor, which gives rise to the minimum reliability cost. This

scheme is able to enhance the reliability of the systems without any extra hardware. The RTFTNO algorithm, however, does not take reliability cost into account. Therefore, our simulation results indicate that under the same workload, the reliability cost generated by the RTFTRC is significantly less than that generated by the RTFTNO algorithm. In addition, we observe from the simulation results that the schedule length generated by the RTFTRC algorithm is shorter than that generated by the RTFTNO algorithm. This fact is the main reason that the RTFTRC algorithm has lower PMD(percentage of missing deadlines) than that of the RTFTNO algorithm. Further more, since the RTFTRC has a shorter schedule length, its schedulability is relatively higher than that of the RTFTNO algorithm. As the schedule threshold is a factor in our algorithms, this paper also discusses the impact of schedule threshold on systems performance. Our simulation experiments show that the schedule threshold has no effect on the RTFTNO algorithm. Schedule threshold has an unnoticeable impact, which is not meaningful, on the RTFTRC algorithm.

Keyword Fault-tolerant, real-time scheduling, heterogeneous distributed systems, simulation experiments, performance evaluation



秦啸，男，1974 年 2 月生。他于 1996 年和 1999 年在华中科技大学分别获得学士和硕士学位。2000 年七月至 2001 年七月，秦啸为澳大利亚 Griffith University 的博士研究生。从 2001 年八月至今，他在美国 University of Nebraska-Lincoln 攻读博士学位。2000 年十月份至 2001 年八月，担任 IEEE 分布式系统在线杂志的编辑。秦啸是 IEEE 和 IEEE 计算机协会的学生会员，他也是澳大利亚工程师/科学家/管理者专业协会的学生会员。秦啸的主要研究领域为实时系统，并行与分布式计算，调度理论和容错计算。本文所讨论的算法是秦啸在华中科技大学攻读硕士学位期间的研究结果。该项研究受国防预研基金资助。

Xiao Qin was born in February, 1974. He received a B.S. and M.S. degree both in computer science from Huazhong University of Science and Technology (HUST) in 1992 and 1999, respectively. In July 2000, he became a Ph.D. student at the school of computing and information technology, Griffith University, Australia. Currently, he is a Ph.D. student at Department of Computer Science and Engineering, University of Nebraska-Lincoln, USA. From October 2000 to August 2001, he has served as an editor for IEEE Distributed System Online. He is a student member of IEEE and Computer Society. He is also a student member of Australian Association of Professional Engineers Scientists and Managers. His research interests are real-time systems，parallel and distributed computing, scheduling theory and fault-tolerance computing. The work reported in this paper had been done by the author when he was a M.S. student at Huazhong University of Science and Technology. This research was supported by National Defense Pre-Research Funding.

韩宗芬，女，1950 年生，教授，主要研究领域为分布式系统，实时计算，操作系统以及软件工程。
Zongfen Han, female, was born in 1950. She is a professor of computer science. Her research interests include distributed systems, real-time computing, operating systems and software engineering.

庞丽萍，女，1944 年生，教授，主要研究领域为并行分布式程序设计，实时系统和分布式系统。她是华中科技大学并行与分布式系统实验室主任，曾在国际会议和国内核心期刊上发表论文五十余篇。
Liping Pang, female, was born in 1944. She is a professor of computer science. Her research interests are parallel and distributed programming, real-time systems and distributed systems. She is the director of the lab of parallel and distributed systems at Huazhong University of Science and Technology. She had published more than 50 papers in the international conference and national journals.

通信地址：Xiao Qin, 2000 J Street APT #1, Lincoln, Nebraska, 68510, USA。邮编：430074
Tel: 1-422-477-5729 (home)
Email: xqin@cse.unl.edu
主页: http://www.cse.unl.edu/~xqin