计算机科学2002Vol.29№.1

非精确任务集的容错单调比率调度

Fault-Tolerant Rate-Monotonic Scheduling of Imprecise Computation Tasks

陈 宇[1] 熊光泽[1] 杨 春[2]

(电子科技大学计算机科学与工程学院 成都610054)[1](四川师范大学数学与软件科学学院 成都610066)[2]

Abstract Meeting deadline and geting high throughput are the main objective of real-time system. They are both important, but sometimes incompatible, such as reserving resource for fault tolerance. We describe a new scheme, through making use of the fault-tolerant rate-monotonic scheduling and imprecise computation, a real-time system can get high throughput when operating normally and can meet the deadline of critical tasks when there are one or more transient error.

Keywords Software fault-tolerant, Rate-monotonic scheduling, Imprecise computation

1 引言

实时系统的基本特性是任务响应时间的确定性和系统处理任务的高吞吐量[1]。随着实时系统越来越广泛地应用于人类生产生活的各个方面,如电站控制系统、病人监护系统、雷达跟踪和多媒体数据处理等,容错也逐渐成为实时系统的重要特性[2]。容错使系统在出现错误时,仍能满足关键任务的时限要求,否则可能造成重大财产损失,甚至造成人员伤亡。系统中的错误主要来源于软件运行产生的暂时错误,重复运行、恢复块技术和多版本编程技术是最主要的容错技术[3]。

在单处理器实时系统中,软件容错主要采用重复运行和恢复块技术,而资源预留是实现软件容错的主要保证。也就是说,系统减小正常情况下的任务吞吐量,预留足够的资源(如时间)以支持对出错任务进行必要的容错处理,以保证在规定时限获得正确的结果。很明显,在系统正常运行时,预留的资源被浪费了。本文将非精确计算技术与容错调度结合,提高容错调度算法的处理器利用率上限,扩大容错调度算法的适用范围,提高系统资源的利用率。

本文分析单调比率调度(RM)和容错单调比率调度(FT-RM)及其充分可调度条件。我们简要描述非精确计算技术的基本概念。我们证明利用非精确计算技术将在很大程度上提高容错单调比率调度的处理器利用率上限,扩大调度算法的适用范围,提高系统的资源利用率。最后我们给出结论。

2 容错单调比率调度

RM 调度是实时周期任务集静态最优调度,其中心思想是:存在任务集 $S(\tau_1,\tau_2,\cdots,\tau_n)$,若任务周期排列为 $T_1\leqslant T_2\leqslant\cdots\leqslant T_n$,则任务优先级排列为 $P_1\geqslant P_2\geqslant\cdots\geqslant P_n$,即周期最短的任务优先级最高。若被调度任务集 S 的最小处理器利用率 $U\leqslant n(2^{1/n}-1)$,则可由 RM 算法调度[4]。这一结论被证明是 RM 调度的充分可调度条件。

设由三个周期任务组成的任务集,其任务参数如下:

τ_1:$C_1=2,T_1=5$

τ_2:$C_2=2,T_2=15$

τ_3:$C_3=4,T_3=20$

当系统无错时,其处理器利用率 $U=0.733$,小于 RM 的最小处理器利用率上限 $U=0.779$,所以该任务集可调度。但当 τ_2 在运行时出现暂时错误时,若采用再次运行以实现容错,其处理器利用率 $U=0.866$,大于 RM 的最小处理器利用率上限,所以无法采用 RM 实现任务调度。

近年来,国外研究者已基于 RM 调度提出了多种容错方案,大都以降低正常运行时的处理器利用率,预留足够多的空闲时间以应对一个或多个暂时软件错误[5,6]。

S.R.Thuel 和 J.K.Strosnider 提出了 PRA(Private Reservation Algorithm)和 CRA(Communal Reservation Algorithm)[5]。PRA 中,每个任务均拥有私有的预留时间,任何任务在其每个周期均能获得规定次数的任务重试时间。对于同一实时任务集,PRA 的最小处理器利用率上限仅是 RM 的最小处理器利用率上限的 $1/(1+X)$,X 为系统容许出错的数量。显然,这是一种非常保守的算法。CRA 中,任务集拥有共有的预留时间,由出错任务竞争其使用权。对于同一实时任务集,CRA 的最小处理器利用率上限等于 RM 的最小处理器利用率上限减所有的预留时间与系统最小周期之比。因此,对于同样的容错要求,CRA 的资源利用率高于 PRA。相对 RM 而言,PRA 和 CRA 并没有改变基本的调度策略,只是预留大量空闲时间,实现容错功能。

调度算法的最小资源利用率上限决定了该算法的适用范围。PRA 和 CRA 由于最小资源利用率上限过低,因而不适用于工程实际。为解决这一问题,容错单调比率调度算法改变了资源预留方式和任务调度策略,以提高了资源利用率[6]。FT-RM 将系统预留时间看做备份任务。正常情况下,为备份任务分配的时间保持空闲;系统出错时,该时段被出错任务占用实现错误恢复。FT-RM 的备份任务出现在系统运行的每一个周期边界上,其资源利用率由系统的容错要求决定。当系统要求在任务出现一个暂时错误的情况下保证所有任务的正确运行时,$U_B=\max(C_i/T_i)$。因此,备份任务在每两个连续的周期边界内占用的时间为 $U_B(lT_i-kT_i)$。显然,根据 RM 算法,U_B 总是每个周期边界上第一个被调度的任务。但是,为实现系统的容错要求,系统正常运行时,U_B 总在调度器的管理下与其后的任务交换运行次序,为该任务提供容错保证。若当前任务

陈 宇 博士研究生;主要研究方向:实时操作系统、实时调度算法、软件容错技术。熊光泽 教授,博士生导师,主要研究方向:实时计算机系统、实时软件可靠性评测。杨 春 讲师,主要研究方向:分布式系统与网络安全。

运行出错，则 FT-RM 调用恢复算法实现容错。恢复算法如下所示：

```
Queue ReadyQ; /* 就绪任务队列 */
Queue BlockQ; /* 阻塞任务队列 */
/* 系统处于容错状态，τr 正在进行容错操作 */
Event:New instance of task τi arrives
if(Di>Dr)then
    Add τi to BlockQ;
else
    Add τi to ReadyQ;
    execute as in RM scheduling
Event:An instance of task τi terminates
if(τi=τr) /* 容错操作结束 */
    Move all tasks from BlockQ to ReadyQ;
endif
execute as in RM scheduling
```

S. Ghosh 等人已经证明，当满足以下条件时，实时任务集可采用 FT-RM 实现单错误容错：

$$U \leqslant U_{1\text{-}FT\text{-}RM} = (n-1)(1-U_B)([2(1-U_B)]^{\frac{1}{n-1}}-1)+U_B$$

当被调度任务的数量由1扩展到无穷时，$U_{1\text{-}FT\text{-}RM}$将由50%增长到69%。相比 PRA 和 CRA 而言，FT-RM 的资源利用率有了很大提高[6]。但是，对于任务数量较少，个别任务的资源利用率较高的情况，FT-RM 的应用范围和资源利用率仍不能令人满意。

3 非精确计算

在工程实际中，系统常常会因为处理突发事件而超载。为保证在超载情况下，系统性能不低于用户可以接受的阈值，系统设计者引入了非精确计算。采用非精确计算的实时系统被成为非精确实时系统。非精确计算模型将任务分割为两个部分，强制执行部分和可选执行部分。强制执行部分必须在任务的时限到来前结束，以保证该任务的输出满足用户的最基本性能要求。可选执行部分的执行可以有两种选择：其一是要么完全执行，要么放弃；其二是在时限到来前尽可能执行，以最大可能接近精确输出。当可选执行部分在到来前运行完成，则称该任务实现精确计算。

非精确计算对于实时系统而言，有着重要的意义。当系统出现超载时，调度器总可以选择放弃部分任务的执行，以尽可能小的性能代价保证系统继续运行，并在系统负载恢复正常后获得理想的系统输出。例如，在视频信号处理中，只要保证任务的强制执行部分能在时限到来前以最小可接受分辨率输出图像，则暂时性的超载不会导致整个系统的失败。

目前，研究人员已经提出了多种非精确计算模型，如 IC[7]，IRIS[8]，m/k-firm deadline[9]等，并在许多实验和实际系统中得到应用[10~12]。

4 非精确 FT-RM 调度

在文[3]中，Laprie 指出，实时系统中60%以上的错误是由软件设计阶段引入的暂时错误，这种错误往往通过任务重试等软件容错手段即可解决。从第2节中的例子可以看出，当采用 RM 调度时，任务的重试导致系统过载，使任务3超越时限。同时，由于任务集的处理器利用率大于 FT-RM 的最小处理器利用率上限，FT-RM 无法调度该任务集。我们认为：利用非精确计算技术，将任务分为强制执行部分和可选执行部分，则可选执行部分占用的时间可以看做为软件容错预留的时间。这样，在保证系统容错的基础上，提高了 FT-RM 的最小处理器利用率上限，扩大 FT-RM 的应用范围，提高系统资源的利用率。我们将这一改进算法称为非精确 FT-RM 调度(IC-FT-RM)。

4.1 系统模型

本模型适用于运行于单处理器的可抢占周期实时任务集，其中任务相互独立。实时任务集 $S(\tau_1(T_1,C_1),\tau_2(T_2,C_2),\cdots,\tau_n(T_n,C_n))$，若任务周期排列为 $T_1 \leqslant T_2 \leqslant \cdots \leqslant T_n$，则任务优先级排列为 $P_1 \geqslant P_2 \geqslant \cdots \geqslant P_n$，即周期最短的任务优先级最高。基于非精确计算技术，将任务 τ_i 分为强制执行部分 M_i 和可选执行部分 O_i，m_i 和 o_i 分别为 M_i 和 O_i 的执行时间，$C_i = m_i + o_i$，O_i 只能在 M_i 完成后开始。O_i 的执行按0/1限制决定，即只能选择完全执行和完全放弃两种方式。

当任务 τ_i 出现错误时，若错误出现在 M_i 执行阶段，则任务将重新执行。若错误出现在 O_i 执行阶段，因为 M_i 已经满足系统对 τ_i 的最基本需要，为避免重新执行 O_i 而影响其他任务，O_i 执行结果将被放弃。

4.2 调度算法

IC-FT-RM 为实现实时任务容错，为被调度任务集添加一个备份任务 T_B。T_B 在每个周期边界处就绪，其执行时间 $U_B(lT_i-kT_i)$，而 U_B 按以下表达式决定：

$$U_B = \max_{i \in S}\left(\frac{m_i - o_i}{T_i}\right)$$

即为所有被调度任务中，M_i 与 O_i 的处理器利用率的差的最大值。我们将包含 T_B 的 S 看做新任务集 S_1。很明显，若 S_1 可调度，则 S 可调度，因为 S_1 具有更高的处理器利用率。按 RM 调度，由于 T_B 周期最小，因此在任何两个连续周期边界间总是首先被调度。然而，IC-FT-RM 总将 T_B 与当前优先级仅次于 T_B 的实际任务 τ_i 交换执行次序。因此，只要 S_1 满足 RM 调度的可调度条件，则其一定可以调度，因为所有实际任务的响应时间均较 RM 调度时提前。若 M_i 出错，则恢复算法使 M_i 占用该任务的 O_i 与 T_B 的时段再次运行以在时限到来前获得正确的结果。在 M_i 进行容错操作期间，若一个更高优先级的任务 τ_j 就绪，则当 $D_i > D_j$ 时，M_i 被抢占，由 τ_j 运行；反之，τ_j 将被阻塞，直到 M_i 运行完毕。若 O_i 出错，则只有 M_i 的运算结果有效，且不对出错 O_i 做任何恢复性操作。IC-FT-RM 算法如下所示：

```
/* IC-FT-RM Scheduling */
Queue ReadyQ; /* 就绪任务队列 */
Queue BlockQ; /* 阻塞任务队列 */
/* 系统处于出错状态，τi 正在进行操作 */
Event:A error is appeared
if(error in Mi)then
    call retry( );
else
    give up the result of Oi;
    output the result of Mi;
/* Retry Scheme */
Event:New instance of task τj arrives
if(Pj>Pi)then
if(Dj>Di)then
        Add τj to BlockQ;
    else
        Add τj to ReadyQ;
        Dispatch( );
else
    Add τj to BlockQ
    Continue;
Event:An instance of task τj terminates
if (τj=τi) /* 恢复操作结束 */
    Move all tasks from BlockQ to ReadyQ;
endif
dispatch( );
```

4.3 算法的正确性

我们认为，上述算法只有具有以下特性：

1)对于任何任务 τ_i，在 kT_i 与 $(k+1)T_i$ 间必须保证 M_i 能运行两次；

2)当 M_i 出错时，恢复机制必须使 M_i 保证在 τ_i 的时限到来前完成再次运行；

3)M_i 的再次运行不能导致其他任务的运行超过时限。

才能在容许出现一个错误的基础上保证所有任务，至少是任务的强制执行部分能在时限到来前完成正确的操作。因此，我们将通过以下几个定理证明 IC-FT-RM 具有以上三个性质，所以是正确的。

定理1 如果 $U_B \geqslant \max[(m_i-o_i)/T_i], i \in S$，则条件1满足。

证明：若 $U_B \geqslant \max[(m_i-o_i)/T_i]$，则在任意任务周期 T_i 中，调度器为备份任务分配时间 $t_B \geqslant (m_i-o_i)$。由调度算法可知，当 M_i 因出错需要再次运行时，其可占用 O_i 和 T_B 的运行时间 $o_i+t_B \geqslant m_i$，所以条件1满足。 □

定理2 若条件1满足，且 IC-FT-RM 采用恢复算法实现容错，则条件2满足。

证明：定理1中我们已经证明任意任务周期 T_i 中，总有足够的时间保证 M_i 的再次运行。在这里，我们需要证明正在执行容错操作的任务 τ_i 不会因为被其他任务抢占而超越时限。因为 IC-FT-RM 采用的恢复算法拒绝时限晚于 τ_i 的高优先级任务抢占 M_i 的再次运行，所以只需要考虑时限早于 τ_i 的高优先级任务对 τ_i 的影响。

设 τ_i 为处于容错阶段的任务，τ_h 优先级高于 τ_i，且 $D_h < D_i$，则 τ_i 被抢占。当 M_i 的再次执行处于 o_i 时段，等同于正常情况下 o_i 被 τ_h 抢占。在正常情况下，抢占发生后，o_i 的剩余部分一定在前 D_i 结束，否则 τ_i 不可调度。同时，T_B 被交换到 τ_h 之后，o_i 的剩余部分之前。所以，M_i 能在时限 D_i 前结束再次运行。

当 M_i 的再次执行处于 t_B 时段，抢占发生。T_B 被交换到 τ_h 之后，为 τ_h 提供容错支持，则 t_B 一定不能超越 D_h。又因为 $D_h < D_i$，则 t_B 一定不能超越 D_i。所以，M_i 能在时限 D_i 前结束再次运行。

综合上述两种情况，采用恢复机制的 IC-FT-RM 满足条件2。 □

定理3 若条件1满足，且 IC-FT-RM 采用恢复算法实现容错，则条件3满足。

定理3的证明可以从两个方面入手：1)证明所有优先级低于 τ_i 的任务，其时限的满足不受 M_i 再次运行的影响；2)证明所有优先级高于 τ_i 的，且因 M_i 再次运行而阻塞的任务，其运行不会超越规定时限。在这里就不做详细描述了。

由以上三个定理可知，只要实时任务集满足 IC-FT-RM 调度的可调度条件，则任务集总可以在一个任务出现暂时错误的情况下提供可接受的运算结果。

4.4 IC-FT-RM 的可调度分析

IC-FT-RM 与 FT-RM 的最大区别在于，通过对任务的分解，将 FT-RM 完全依靠备份任务实现容错改变为依靠备份任务和任务本身的可选执行部分结合以实现容错。因此，IC-FT-RM 与 FT-RM 具有同样的可调度条件表达式，但参数 U_B 不同。

可调度条件直接决定调度算法的适用范围。因此，我们通过比较 IC-FT-RM 与 FT-RM 的最小处理器利用率上限，以确定其适用范围间的关系。所以，

$$\frac{U_{1\text{-}IC\text{-}FT\text{-}RM}}{U_{1\text{-}FT\text{-}RM}} = \frac{(1-U'_B)\left((n-1)\left([2(1-U'_B)]^{\frac{1}{n-1}}-1\right)-1\right)+1}{(1-U_B)\left((n-1)\left([2(1-U_B)]^{\frac{1}{n-1}}-1\right)-1\right)+1}$$

其中，$U_B=\max_{i\in S}(\frac{C_i}{T_i})$，$U'_B=\max_{i\in S}(\frac{m_i-o_i}{T_i})$

因为 $U_B \leqslant U_B$，所以

$$\frac{U_{1\text{-}IC\text{-}FT\text{-}RM}}{U_{1\text{-}FT\text{-}RM}} \geqslant 1$$

因此，对于非精确任务集，IC-FT-RM 的适用范围大于 FT-RM。对于任务集中某些任务必须实现精确执行的情况，IC-FT-RM 等同于 FT-RM。

4.5 举例

在本节中，我们将以一个实例直观地比较 IC-FT-RM 和 FT-RM。设任务集 S 参数如下：

$\tau_1: C_1=1.5, m_1=0.6, o_1=0.9, T_1=5$

$\tau_2: C_2=2, m_2=1.5, o_2=0.5, T_2=15$

$\tau_3: C_3=4, m_3=2.5, o_3=1.5, T_3=20$

则，$U_S=0.633$，$U_B=0.3$，$U'_B=0.125$，$U_{1\text{-}FT\text{-}RM}=0.567$，$U_{1\text{-}IC\text{-}FT\text{-}RM}=0.690$。因为，$U_S > U_{1\text{-}FT\text{-}RM}$，$U_S < U_{1\text{-}IC\text{-}FT\text{-}RM}$，所以1-FT-RM 无法调度该任务集，而1-IC-FT-RM 可以完成容错调度任务。

通过上述范例可以看出，对于适用于非精确计算的实时任务集，即完成部分操作就能够满足系统最基本要求，IC-FT-RM 具有比 FT-RM 更为广泛的适用范围和资源利用率。

结论 在本文中，我们将非精确计算引入实时容错单调比率调度，以提高其最小处理器利用率上限，扩大调度算法的应用范围。在实时系统中，任务在满足时限要求的前提下完全运行可以获得最令人满意的结果，但有时，在时限到来前完成规定部分的操作，其结果仍旧是可以接受的。非精确容错单调比率调度正是利用任务的这一特点，将任务可选执行部分的资源占用做为本任务强制执行部分实现容错的部分资源，提高了系统资源利用率，进而提高调度算法的最小处理器利用率上限，扩大调度算法的应用范围。

非精确容错单调比率调度保持单调比率调度原有的调度策略，仅在任务出错时添加了恢复机制，其实现较为简单。实际上，出错任务亦可以使用其他任务可选执行部分的资源占用作为实现容错的资源，但这将对调度策略做很大的改动，增加调度的复杂性，加大系统开销。我们今后将在这一方面进行更深入的探讨。

参 考 文 献

1 Furth B, Halang W A. A Survey of Real-Time Computing Systems. International Journal of Mini and Microcomputers, 1994, 16 (3)

2 Cristian H. Understanding fault-tolerant distributed systems. Communications of the ACM, 1991, 34(2): 56～78

3 Laprie J-C. Dependability of Computer System: from Concept to Limits. LAAS-CNRS, Toulouse, France

4 Liu C L, Layland J W. Scheduling Algorithm for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM, 1973, 20(1): 40～61

5 Thuel S R, Strosnider J K. Enhancing Fault Tolerant of Real-Time Systems through Time Redundancy. G. M. Koob and C. G. Lau, eds, Kluwer, 1994. 265～318

6 Ghosh S, et al. Fault-Tolernt Rate-Monotonic Scheduling

（下转第92页）

要的。这就需要在不考虑任何解空间的单个元素的前提下把一个特殊的解从所有可行解中隔离出来。一个好的随机策略应达到"隔离",通过绝对地在可行解上选择一个随机次序,然后要求所有处理器集中力量寻找阶最低的解。在分布式计算中,某些处理器经常需要打破"死锁"状态,从而达到一致。随机算法还是一个避免死锁的强大工具。

我们以选择协作(Choice Coordination)问题为例。在并行和分布计算中经常出现的问题之一是要打破一个可行解集合的对称性,这可以通过随机算法来实现。我们使用下述处理器之间通讯的简单模型:有 m 个登记表可被这 n 个处理器读写,几个处理器可能几乎同时地试图读写或修改一个登记表。为解决这种冲突,我们假设处理器使用一个加锁机制,当某几个处理器试图存取登记表而有一个处理器唯一地获得了存取权限时它加锁,其它的处理器要等到锁被释放后才能再次为存取登记表而竞争。所有这些处理器要运行一个策略,以便在 m 个表中作出一个选择,策略运行到最后 m 个登记表的每一个含有一个特殊符号。一个选择协作问题的复杂性可用 n 个处理器的总的读写步数来衡量。为了显示随机算法的重要贡献,我们给出一个随机策略,对任意 c>0,它将以成功的概率至少$1-2^{-n^{(c)}}$在 c 步内解决冲突[15]。

9 概率方法和存在性证明

通过论证一个随机选择的对象在正概率下有某些特性,可以建立对象具有该特性的存在性证明。这样的论证虽然直观易懂,却并不能给出如何寻找这样一个对象。有时,我们用这种方法来说明解决某个难题的算法确实存在,尽管知道算法存在,但不知算法是什么样的,也不知道如何去构造算法,这就引出算法中非一致性(non-uniformity)的观点。

综上所述,随机算法是从概率的观点而不是分析的观点去考察处理问题的过程,这种新工具的引入使得算法设计与分析重新呈现出勃勃生机。

早期的算法研究多半浅显易懂,富于技巧,以至于有些算法设计可作为中学的数学竞赛题。随机算法带来了一些意义深远的转变,这些转变的标志是一些抽象方法的引进和一些完全不平庸结果的出现。我们不必为这一转变担忧而应该为其喝彩,因为这不是经典算法分析丧失其地盘而是它因新工具的注入而正变得更加强壮。随机算法带来的好的结果揭示了好的算法分析与设计不仅存在而且比人们想象得要复杂得多,算法分析本身就是在设计算法。换个角度来看,把随机性引入算法,实质上是把经典数学应用于离散数学。这种各分支的相互交叉,可能代表着数学发展的一个最重要趋势,计算机科学界可以希望通过获取诸多数学分支中的有力工具而使算法领域变得更加壮大。

参 考 文 献

1 Hochbaum D S. Approximation algorithms for NP-hard problems. PWS, 1997

2 Irani S, Karlin A R. Online computation, in[1], 521～564

3 Snir M. Lower bounds on probabilistic linear decision trees. Theoretical Computer Science, 1985, 38: 69～82

4 Sleator D D, Tarjan R E. Amortized efficiency of list update and paging rules. Communications of the ACM, 1985, 28: 202～208

5 Feller W. An Introduction to Probability Theory and Its Applications, Volume Ⅰ, Ⅱ, John Wiley, New York, 1968

6 Aragon C R, Seidel R G. Randomized search trees. In: Proc. of the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989. 540～545

7 Seidel R G. On the all-pairs-shortest-path problem. In: Proc. of the 24th Annual ACM Symposium on Theory of Computing, 1992. 745～749

8 Harary F, Palmer E M. Graphical Enumeration, Academic Press, New York, 1973

9 Rabin M O. Probabilistic Algorithms, in Algorithms and Complexity. edited by J. Traub, Academic press, 1976

10 Blum M, Kannan S. Designing programs that check their work. In: Proc. of the 21st Annual ACM Symposium on Theory of Computing, 1989. 86～97

11 Valiant L G. The complexity of enumeration and reliability problems. SIAM Journal on Computing, 1979, 8: 410～421

12 Aleliunas R. Randomized parallel communication. In ACM-SIGOPS Symposium on Principles of Distributed Systems, 1982. 60～72

13 Upfal E. Efficient schemes for parallel communication. Journal of the ACM, 1984, 31: 507～517

14 Karp R M, Luby M. Monte Carlo algorithms for the planar multiterminal network reliability problem. Journal of Complexity, 1985, 1: 45～64

15 Cook S A. A taxonomy of problems with fast parallel algorithms. Information and CONTROL, 1985, 64(1-3): 2～22

(上接第98页)

7 Liu J W S, et al. Imprecise Computations. Proceedings of the IEEE, 1994, 82(1): 83～93

8 Dey J K, Kurose J, Towsley D. On-line Scheduling Policies for a Class of IRIS Real-Time Tasks. IEEE Transactions on Computers, 1996, 45(7): 802～813

9 Hamdaoui M, Ramanathan P. A Dynamic Priority Assignment Technique for Streams with (m, k)-Firm Deadlines. IEEE Transactions on Computers, 1995, 44(12): 1443～1451

10 Aydin H, et al. Optimal Reward-Based Scheduling of Periodic Real-Time Tasks. In: Proc. of 20th IEEE Real-Time Systems Symposium, Dec. 1999

11 Aydin H, Melhem R, Mossé D. Incorporating Error Recovery into the Imprecise Computation Model. In: The Sixth Intl. Conf. on Real-Time Computing Systems and Applications, Dec. 1999

12 Aydin H, Melhem R, Mossé D. Tolerating Faults while Maximizing Reward. In: Proc. of the Twelfth Euromicro Conf. on Real-Time Systems, June 2000