一种高效的实时多处理器系统的资源回收算法*)

宾雪莲[1] 杨玉海[1] 宾 亚[1] 金士尧[2]

(空军雷达学院 武汉 430019)[1] (国防科技大学计算机学院 长沙 410073)[2]

摘 要 实时多处理器系统中,为了更好地利用资源,常常采用资源回收算法。在分析了已有资源回收算法的优缺点的基础上,提出了一种高效的实时多处理器的资源回收算法——可倒置法。可倒置法允许只存在资源冲实的任务,在不会引起运行时间异常的情况下,出现执行顺序倒置。实验结果表明改进的限制向量法要由于已有的资源回收算法。

关键词 实时多处理器,资源回收算法,资源冲突,执行顺序倒置

An Efficient Resource Reclaim Algorithm for the Real-Time Multiprocessor Systems

BIN Xue-Lian[1] YANG YU-Hai[1] BIN Ya[1] JIN Shi-Yao[2]

(Airforce Army Radar Academy, Wuhan 430019)[1]

(School of Computer Science, National University of Defense Technology, Changsha 410073)[2]

Abstract Many real-time multiprocessor systems use resource reclaiming algorithms to utilize resource left unused by a task when it finishes early. In this paper, we firstly analyze the existing resource reclaiming algorithms. And then, a new effective reclaiming algorithm, named by Passing Algorith(PA), is proposed. As long as there are no run-time anomalies, PA allows passing among tasks with resource conflict. Simulation results show that PA outperform the other reclaiming algorithms.

Keywords Real-time multiprocessor, Resource reclaiming, Resource conflict, Passing

1 引言

随着实时应用范围的加大和复杂度的提高,多处理器系统由于其高性能及可靠性,逐渐成为处理这种复杂应用的有效手段。目前,实时多处理器系统已被广泛应用于航天控制、核反映堆控制等领域,而实时多处理器系统的调度算法则成为一个重要研究课题。在文[1~2]中指出,基于多处理器的调度问题主要是确定任务在哪个处理器上执行,以及何时执行的问题。

在实时多处理器系统中,一般采用任务的最大计算时间进行可调度分析以及进行资源和处理器的分配[3~5],即确定任务在哪个处理器上运行以及任务的开始运行时间。但在实际执行过程中,任务的实际计算时间要小于其最大计算时间。这时,如果任务仍然按照原先确定的开始执行时间执行,系统的资源就没有得到充分利用。为了提高系统的性能,使得新到达的任务有较多的机会得到执行,可以采取资源回收算法回收未被任务使用的资源。

在文[7]中针对相互独立的任务提出基本回收算法(Basic reclaiming)、尽早启动算法(Early Start)。在文[8]中针对具有执行顺序限定关系(precedence constraints)的任务提出约束向量法(Restriction Vectors)算法。为了进一步回收资源,文[9]中提出估计策略(Estimation Strategy)。本文将首先分析上述算法的缺点,然后提出一种高效的回收算法。

2 问题描述

假设一个实时多处理器系统有 m 个同构处理器($m>1$)和 s 个资源($s\geqslant 1$),该系统中的实时任务具有以下特性:

- 每一个任务 τ_i 都是非周期的,其到达时刻为 a_i,释放时刻为 r_i,最坏计算时间为 c_i,截止期限为 d_i;任务的释放时刻与到达时刻相等。任务的实际计算时间记为 c'_i,$c'_i\leqslant c_i$;
- 一个任务不能在同一时间在多个处理器上执行,即一个任务不能并行执行;
- 除了处理器以外,任务可能还需要其它一些资源,任务对资源的访问方式为互斥和共享方式。当 τ_i 和 τ_j 需要访问某个资源,并且至少有一个任务的资源访问方式为互斥方式,则称 τ_i 和 τ_j 发生了资源冲突。以 $\tau_i\odot\tau_j$ 表示 τ_i 和 τ_j 发生资源冲突,以 $\tau_i\overline{\odot}\tau_j$ 表示 τ_i 和 τ_j 未发生资源冲突。
- 任务之间可能具有执行顺序约束关系。若在 τ_i 与 τ_j 之间具有执行顺序约束关系,且 τ_i 必须在 τ_j 开始执行前完成,则以 $\tau_i<\tau_j$ 表示这种执行顺序约束关系,以 $\tau_i<>\tau_j$ 表示 τ_i 与 τ_j 不存在执行顺序约束关系。
- 任务之间是不可抢占的。

与文[8,9]相同,本文也采用集中式调度。在系统中,有一个处理器作为专门的调度器,所有任务都要先到达这个中心调度器,然后被分配到系统中其它处理器上执行。每个处理器都有自己的调度队列,这样在它执行完当前任务后,就从调度队列中取出一个任务来执行。调度器与各处理器之间的通信通过这些调度队列来实现。同时,调度器与各处理器并行地执行,它对新到达的任务进行调度并周期性地对各调度队列进行修改。任务队列中的任务按照截止期限从小到大的顺序排序,在截止期限相同时,按照最早开始执行时间排序。

在一个可行性调度 S 中,当任务以最大计算时间运行

*)本文受国家自然科学基金资助(项目编号:60073003)。宾雪莲 博士、讲师,主要研究方向为实时调度技术;杨玉海 博士生、讲师,主要从事计算机网络、并行与分布式系统等方面的研究;宾 亚 硕士、讲师,主要从事网络技术研究;金士尧 博士生导师、教授,主要从事仿真、实时、并行与分布系统等方面的研究。

时，其时间约束和资源需求都可以满足。在 S 中，每个任务都被分配到某个处理器上执行，以 $Proc(i)$ 表示任务 τ_i 被分配到第 $Proc(i)$ 个处理器上执行，且其在指定处理器上的开始执行时刻为 st_i 和结束时刻为 ft_i，$\forall i, ft_i \leqslant dt_i$。

定义 1 对于一个给定的可行性调度 S，S' 为该任务集合的实际调度。任务 τ_i 在 S' 中的开始执行时刻为 st_i'，结束时刻为 ft_i'。由于任务 τ_i 的实际计算时间 $c_i' \leqslant c_i$，因此 st_i'、ft_i' 可能与 st_i、ft_i 不同。

定义 2 如果所有任务在 S' 中的实际执行顺序与在 S 中的执行顺序相同，则称 S' 为 S 的 post-run。

定义 3 给定一个 post-run S'，若 $st_i' \leqslant st_i$，即任务 τ_i 的实际开始执行时刻不大于其在 S 中的开始执行时刻，则称及时地执行了该任务 τ_i。

定义 4 如果 $\forall i, ft_i' \leqslant dt_i$，则称 post-run S' 为正确的。

定义 5 若在实际调度 S' 中，$st_i' < st_j'$，$ft_j < st_i$，则称任务 τ_i、τ_j 出现了执行顺序倒置。若在可行性调度 S 中能够在截止期限前完成的任务，在 S' 中不能在截止期限前完成，则称出现了运行时间异常。为了保证不会发生运行时间异常，在 S' 中每个任务 τ_i 的开始执行时刻 st_i' 一定要小于等于其在 S 中的开始执行时刻 st_i。

定义 6 令 $\tau_{<i} = \{\tau_j : ft_j < st_i\}$；$\tau_{>i} = \{\tau_j : st_j > ft_i\}$；$\tau_{\approx i} = \{\tau_j : \tau_j \notin \tau_{<i}$ 且 $\tau_j \notin \tau_{>i}\}$。$\tau_{<i}$ 为可行性调度 S 中在 τ_i 开始执行之前完成的任务集合，$\tau_{>i}$ 为在 τ_i 完成以后开始执行的任务集合，$\tau_{\approx i}$ 为执行时间与 τ_i 有重叠的任务集合。

定义 7 令 $\tau_{<i}(j) = \{\tau_k : \tau_k \in \tau_{<i}, proc(k) = j\}$，$\tau_{<i} = \bigcup_{j=1}^{m} \tau_{<i}(j)$。$\tau_{<i}(j)$ 中的任务按照在可行性调度 S 中的开始时刻排序。

定义 8 每个任务 τ_i 都有一个资源约束向量 $RV_i[1\cdots m]$。$RV_i[j]$ 记录在处理器 P_j 上必须在 τ_i 之前执行完的 $\tau_{<i}(j)$ 中的最后一个任务。

$$RV_i(j) = \begin{cases} \tau_k & proc(i) = j, \tau_k \in \tau_{<i}(j), \text{且 } \tau_k \text{ 为在 } \tau_{<i}(j) \text{ 中的开始执行时刻最晚的任务} \\ \tau_m & proc(i) \neq j, \tau_m \text{ 为在 } \tau_{<i}(j) \text{ 中满足 } (\tau_m \odot \tau_i \text{ 或 } \tau_m < \tau_i) \text{ 条件的开始执行时刻最晚的任务} \\ - & \text{otherwise} \end{cases}$$

3 已有研究

基本算法(Basic)、尽早启动算法(Early-Start)和约束向量法(Restriction-Vector)中用一个全局变量 $reclaim\_\delta$ 来表示资源回收的时间。$Reclaim\_\delta$ 在算法起始时为 0，在回收算法的运行过程发生变化，用来表示任务的最早可提前运行的时间。

在基本算法中，不允许出现执行顺序倒置，任务的实际执行顺序与在可行性调度 S 中的顺序相同。为了满足该目的，其基本思想是在每个任务执行完成后，检查是否所有的处理器都处于空闲状态。如果所有的处理器都处于空闲状态，则未调度的任务的开始执行时刻可以提前 $reclaim\_\delta$ 时间。基本算法的时间复杂度为 $O(m)$。

在基本算法的基础上，研究人员提出了尽早启动算法。为了使得不会出现执行顺序倒置的情况，在尽早启动算法中，需先求出每个任务的 $\tau_<$ 集合。然后每当一个任务执行完成之后，检查所有空闲处理器。假设 τ_k 为空闲处理器 P_i 上即执行的任务，检查 $\tau_{<k}$ 中的任务是否都已完成。若是，则立即执行 τ_k。τ_k 实际开始执行 $st'_k = st_k - reclaim\_\delta$。尽早启动算法的时间复杂度为 $O(m^2)$。

在约束向量法中，在一个任务执行完后，检查所有空闲处理器。如果 τ_k 为空闲处理器 P_i 上即将执行的任务，检查 $RV_k[1\cdots m]$ 中的任务是否都已执行完毕。若是，则立即执行 τ_k。τ_k 实际开始执行 $st_k' = st_k - reclaim\_\delta$。在约束向量法中，只有当 $\tau_j <> \tau_j$ 且 $\tau_i \overline{\odot} \tau_j$ 时，才允许在 τ_i、τ_j 出现执行顺序倒置，时间复杂度为 $O(m^2)$。

由于基本算法、尽早启动算法和约束向量法只有在任务执行完成之后，才能改变 $reclaim\_\delta$ 的值，因此 $reclaim\_\delta$ 变化速度比较慢，不利于更好地利用资源，提高任务的接收率。为了克服 $reclaim\_\delta$ 变化慢的缺点，在尽早启动算法和约束向量法的基础上，文[9]提出了估计策略。估计策略采用了一个全局变量 $M(t)$ 表示资源回收的时间。$M(t) = \min_{i=1}^{m}\{(st_j - st'_j) \text{or} (ft_j - ft'_j)\}$，其中 τ_j 为在 t 时刻之前在 P_i 上开始执行的任务或者在 t 时刻结束的最后一个任务，st_j、ft_j 为 τ_j 在 S 中的开始执行时刻与结束时刻，st_j'、ft_j' 为在 S' 中的实际开始执行时刻与实际结束时刻。在估计策略中，在任务执行完后和在任务开始执行时都可改变 $M(t)$。

4 可倒置法(Passing Algorithm)

与基本回收算法和尽早启动算法相比，由于约束向量允许执行顺序倒置，因此约束向量能够更好地回收资源。仔细观察约束向量法的允许执行顺序倒置的条件可以发现，只有当 $\tau_i <> \tau_j$ 且 $\tau_i \overline{\odot} \tau_j$ 时，才允许 τ_i、τ_j 出现执行顺序倒置。这个条件仍然过于严格，不利于更好地回收资源。而实际上，只有当任务之间存在执行顺序约束关系时，才不允许出现执行顺序倒置。如果任务之间只存在资源冲突，不存在执行顺序约束关系，那么是可以允许出现执行顺序倒置的。在约束向量法中要求允许执行顺序倒置的任务之间不存在执行顺序约束关系和资源冲突的目的是为了防止在 S 中可以成功调度的任务在 S' 中不能调度的情况。如果能够允许只存在资源冲突的任务，在不会引起运行时间异常的情况下，出现执行顺序倒置，则能够更好地回收资源。如例 1 所示。

例 1 设系统中有 2 个处理器，分别为 P_1，P_2；有一个资源 R，资源 R 只有一个实例。任务特征参数见表 1。

表 1 任务特征参数

τ_i	$Proc(i)$	c_i	c_i'	d_i	R	st_i	ft_i
τ_1	2	225	125	225	-	0	225
τ_2	2	25	25	400	共享	225	400
τ_3	1	175	150	175	-	0	175
τ_4	1	25	25	200	互斥	175	200
τ_5	1	150	75	350	-	200	350
τ_6	2	200	100	500	-	400	500
τ_7	1	100	75	500	共享	350	500

图 1 为表 1 所示的任务以最大计算时间运行时的可行性调度 S。图 2 为采用约束向量法的实际调度 S'。图中的灰色部分表示处理器的空闲时间段。从图 2 可以看出由于 $\tau_2 \odot \tau_4$，并且 $RV_2[1] = \tau_4$，因此 τ_2 只有在 τ_4 执行完以后才能执行。由于在 $t = 125$ 时，$t + c_2 \leqslant 150$，因此将 τ_2 的开始执行时刻提前到 $st_2' = 125$，$ft_2' = 150$，不会使得 τ_4 的实际开始时刻

$st_4' > st_4$，因此不会出现运行时间的异常。

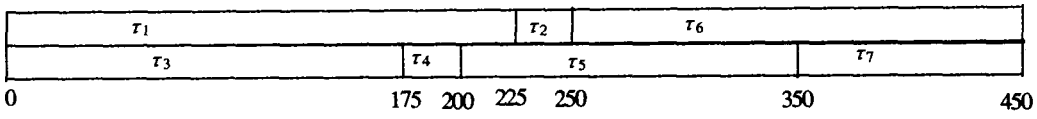

图 1　可行性调度 S

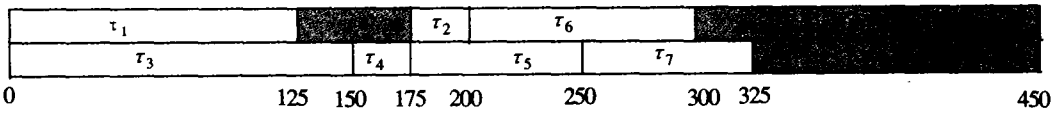

图 2　采用约束向量法的实际调度 S'

为了实现这一思想，采用可倒置法(*Passing Algorithm*)

令 *CBM*[i]表示 τ_i 的执行情况。

$$CBM[i]=\begin{cases}0 & \text{任务 } \tau_i \text{ 还未开始执行}\\ -1 & \text{任务 } \tau_i \text{ 已开始执行，但还未执行完毕}\\ 1 & \text{任务 } \tau_i \text{ 已执行完毕}\end{cases}$$

在可倒置法中，允许任务执行顺序倒置的条件为：

设在可行性调度 S 中，$st_i > st_j$、$ft_i > ft_j$，$proc(i) \neq proc(j)$，$proc(k)=proc(i)$，$RV_i(proc(i))=\tau_k$，$RV_i(proc(j))=\tau_j$，当 τ_i、τ_j 满足以下两个情况之一时，允许 τ_i、τ_j 出现执行顺序倒置：

情况(1)：$\tau_i <> \tau_j$；$\tau_i \overline{\odot} \tau_j$；

情况(2)：$\tau_i <> \tau_j$，$\tau_i \odot \tau_j$，$CBM[j] \neq -1$ 且 $ft_k' + c_i \leqslant st_j$。

以上两种情况分别表示：当 τ_i 和 τ_j 之间无执行顺序关系且无资源冲突时，允许 τ_i 和 τ_j 出现执行顺序倒置；当 τ_i 和 τ_j 之间无执行顺序关系，τ_i 和 τ_j 之间有资源冲突，若 τ_j 还没有开始运行，并且 τ_k 的实际结束时间与 τ_i 的最大执行时间之和不大于 τ_j 在可行性调度 S 中的开始执行时间 st_j 时，允许 τ_i、τ_j 出现执行顺序倒置。

为了实现这一思想，并且为了防止出现运行时间异常，用两个 $RV_{i1}(j)$和 $RV_{i2}(j)$向量表示在处理器 P_j 上与 τ_i 有资源冲突和有执行顺序约束关系的任务。其中 $RV_{i1}(j)$表示在处理器 P_j 上与 τ_i 有执行顺序约束关系的任务，且该任务在 τ_i 开始执行之前应已完成。$RV_{i2}(j)$表示在处理器 P_j 上与 τ_i 发生资源冲突的任务，且在 S 中该任务的开始执行时刻小于 τ_i 的开始执行时刻。

$RV_{i1}(j)=$

$$\begin{cases}\tau_k & proc(i)=j, \tau_k \in \tau_{<i}(j), \text{且 } \tau_k \text{ 为在 } \tau_{<i}(j) \text{ 中开始执行时刻最晚的任务}\\ \tau_m & proc(i)\neq j, \tau_m \text{ 为在 } \tau_{<i}(j) \text{ 中满足 } \tau_m < \tau_i \text{ 条件的开始执行时刻最晚的任务}\\ - & otherwise\end{cases}$$

$RV_{i2}(j)=$

$$\begin{cases}\tau_m & proc(i)\neq j, \tau_m \text{ 为在 } \tau_{<i}(j) \text{ 中满足 } \tau_m \odot \tau_i \text{ 条件的开始执行时刻最晚的任务}\\ - & otherwise\end{cases}$$

可倒置法的具体实现如下：

当在处理器 P_j 上的任务 τ_i 开始执行时或结束执行时

```
If τi开始执行 then CBM[i]=-1;
If τi 已结束执行 then
  { CBM[i]=1;
    for k=1 to m do
    { If Pk 空闲 then
            If (RVf1[k]中任务没有执行完毕或 RVf2[k]中存在已开始执行但未执行完毕的任务)
              //τf 为处理器 Pk 的调度队列 DQ[k]中的第一任务
              then
                  do nothing
              else{
                   if (RVf1[k]和 RVf2[k]中所有任务已执行完毕
                       then 开始执行 τf,将 τf 从 DQ[k]移走
                       if (RVf1[k]中所有任务已执行完毕)then{
                           flag=1;
                           for p=1 to m do
                           { τj=RVf2[p]
                             if(RVf2[p]<>nil) and (CBM[τj]=0)or((RVj1[k]和 RVj2[k]中存在已开始执行但还未执行完毕的任务)or(fti' + cf > stj))// fti' 为 τi 的结束时刻
                           then{flag=0;break;}
                           }
                       if flag=1 then 开始执行 τf,将 τf 从 DQ[k]移走}
    }
}
```

在可倒置法中，也可采用估计策略来进一步扩大资源回收时间。可倒置法使用估计策略的方法与约束向量的使用方法类似：令 M_arr(i)为处理器 P_i 上的资源回收时间。当在处理器 P_i 上的任务 τ_k 开始执行或者结束时，判断 $RV_{k1}[i]$、$RV_{k2}[i]$中任务是否已全部执行完毕。若已全部执行完毕，则更新 M_arr(i)。M_arr(i)的更新方法如下：

$M\_arr(i)=\{(st_k - st_k')or(ft_k - ft_k'), if\ proc(k)=i$ 并且在 $RV_{k1}[i]$和 $RV_{k2}[i]$中的所有任务都已执行完毕。全局变量 M(t)：$M(t)=min_{i=1\cdots m}\{M\_arr(i)\}$。

图 3 为对表 1 中的任务采用可倒置法的实际调度 S'。

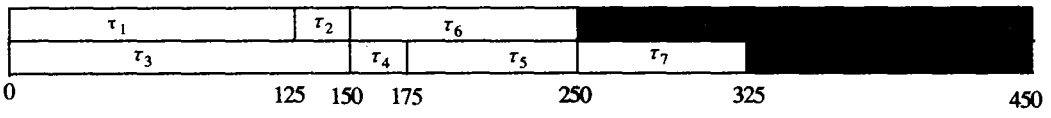

图 3　采用可倒置法的实际调度 S'

5 性能比较

为了对改进的限制向量法进行评估，采用与文[5]相同的任务生成方法，调度算法采用与文[8]相同的动态调度算法。可行性检查窗口为4。

在试验中，*Use_P* 表示使用资源的概率；*Share_P* 表示共享使用资源的概率。在进行任务调度时，调度器是按照任务的最大计算时间加上执行资源回收算法的时间进行调度的。这里任务的最大计算时间包括了执行资源回收算法的时间。令 *AC* 服从以[30,50]为参数的均匀分布。任务的最大计算时间为 *AC*+*RC*。*RC* 为执行资源回收算法的时间；其中当资源回收算法为基本回收算法时，*RC*=1；当资源回收算法为尽早启动算法、约束向量法、改进的约束向量法时，*RC*=*Procnum*(*Procnum* 表示系统中处理器个数)。任务的实际计算时间为 *aw-ratio* * 任务的最大计算时间。*aw-ratio* 表示实际计算时间与最大计算时间的比率，并且服从[0.6,0.65]的均匀分布。任务的截止期限为(*D* *(*Max_exec*+*Min_exec*)/2+任务的到达时刻)，其中 *D* 表示任务的延迟度，描述了任务的紧急程度；*D* 服从[1.3,1.5]的均匀分布。*density* 表示任务之间的相关度，当 *density*=0 时，表示该任务独立，*density* 等于有执行顺序约束关系的任务的个数除以任务集合中的个数。任务的到达时刻服从以 1/ *TaskFrq* 为参数的泊松分布，其中 *TaskFrq* 表示任务到达的平均时间间隔，是到达频率的倒数，*TaskFrq* 取值225。

按照以上参数生成200个任务集，每一个任务集包含1000个任务。在进行模拟时，所有的算法都未采用估计策略。试验结果如图4,5,6,7,8,9所示。

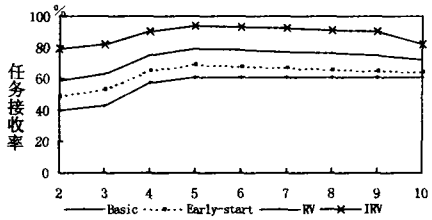

图4 处理器个数对任务接收率的影响

D=1.4,*Use_P*=0.5,*TaskFrq* =225,*Share_P*=0.5,*aw-ratio*=0.6,*density*=0.85

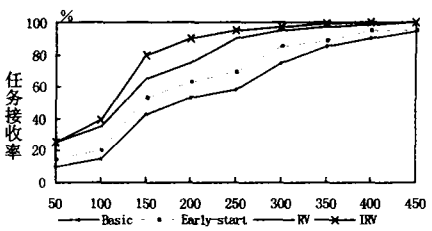

图5 TaskFrq 对任务接收率的影响

D=1.4,*Use_P*=0.5,*Share_P*=0.5,*Procnum* =6,*aw-ratio*=0.6,*density*=0.85

从图4中可以看出，随着处理器的个数增加，任务接收率也随着增加。当处理器个数增加到一定值时，任务接收率不会再随之增加，而是缓慢下降。任务接收率的增加是由于处理器增加了，但当处理器增加到一定值时，任务的到达时刻决定了接收率不会发生太大变化。而且由于处理器的增大，资源回收算法的开销也随之增大，从而造成了接收率的下降。从图4中可以看出 Passing Algorithm(PA)算法性能最好。

从图5中可以看出任务的接收率随着 *TaskFrq* 增加而增加。当 *TaskFrq* 增加到一定值时，由于系统负载较低，因此调度器几乎能够调度所有的任务。由于到达率是 *TaskFrq* 的倒数，因此 *TaskFrq* 越大，到达率越小。因此可以得出在任务到达率的变化过程中，PA算法的性能始终高于其它算法的结论。

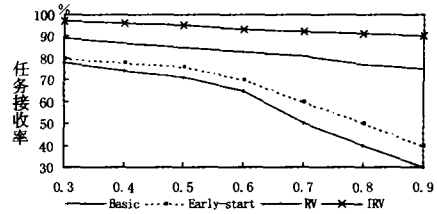

图6 aw_ratio 对任务接收率的影响

D=1.4,*Procnum* =6,*Use_P*=0.5,*Share_P*=0.5,*density*=0.85

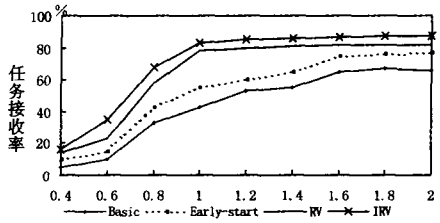

图7 任务的可延迟度对任务接收率的影响

Use_P=0.5,*Share_P*=0.5,*Procnum* =6,*arrival rate* =225,*density*=0.85

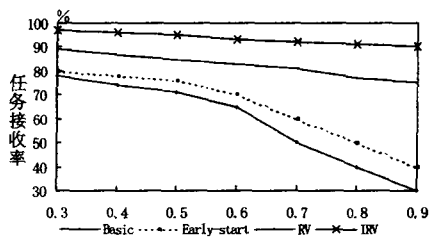

图8 资源利用率对任务接收率的影响

D=1.4,*Share_P*=0.5,*Procnum* =6,*aw_ratio*=0.6,*arrival rate*=225,*density*=0.85

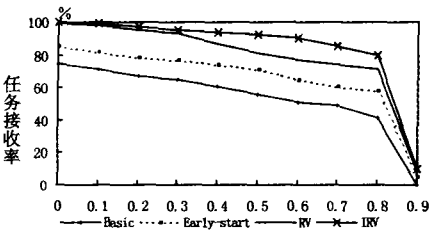

图9 任务相关度对任务接收率的影响

D=1.4,*Share_P*=0.5,*Procnum*=6,*aw_ratio*=0.6,*arrival rate*=225

从图6中可以看出任务的实际计算时间越小，接收率越高，并且 PA 算法始终优于其它算法。

从图7中可以看出 PA 算法的性能始终要优于其它3种算法。

从图8中可以看出任务的接收率随着资源利用率增加而减少，但 PA 算法始终要优于其它算法。

从图9中可以看出，任务的接收率随着任务相关度的增

(下转第279页)

Methods in System Design, 1996, 9(1/2): 105～130
8 Emerson E A, Trefler R J. From asymmetry to full symmetry: new techniques for symmetry reduction in model checking. In: Correct Hardware Design and Verification Methods. Vol 1703 of LNCS, 1999. 142～156
9 Weiser M. Program slicing. IEEE Transactions on Software Engineering, 1984, 10(4): 352～357
10 Clarke E M, Grumberg O, Long D E. Model checking and abstraction. ACM Transactions on Programming Languages and System (TOPLAS), 1994, 16(5): 1512～1542
11 Clarke E, Grumberg O, Jha S. Counterexample-Guided abstraction Refinement for symbolic Model checking, Journal of the ACM, 2003, 50(5): 752～794
12 Dams D, Gerth R, Grumberg O. Abstract interpretation of reactive systems. ACM Transactions on Programming Languages and System(TOPLAS), 1997, 19(2): 253～291
13 Rushby J. Integrated formal verification: using model checking with automated abstraction, invariant generation, and theorem proving. In: Theoretical and practical aspects of SPIN model checking. 5th and 6th international SPIN workshops, 1999. 1～11
14 Dams D. Abstraction in software model checking: Principles and practice(tutorial overview and bibliography). Model Checking Software. Vol 2318 of LNCS, Springer-Verlag, 2002
15 Dams D. Abstract Interpretation and Partition Refinement for Model Checking: [PhD thesis]. Eindhoven University of Technology, P. O. Box 513, 5600 MB Eindhoven, The Netherlands, July 1996
16 Lee W, Pardo A, Jang J, et al. Tearing based abstraction for CTL model checking. In: International Conference of Computer-Aided Design, 1996. 76～81
17 Park D. Concurrency and automata on infinite sequences. In: Proceedings of the 5th GI-Conference on Theoretical Computer Science, Springer-Verlag, 1981. 167～183
18 Milner R. An algebraic definition of simulation between programs. In: Proceedings of the 2nd International Joint Conference on Artificial Intelligence, 1971. 481～489
19 Cousot P, Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of programming languages, 1977. 238～252
20 Dams D, Namjoshi K S. The existence of finite abstractions for branching time model checking. In Nineteenth Annual IEEE Symposium on Logic in Computer Science(LICS'04), IEEE Computer Society Press, 2004. 335～344
21 Graf S, Saidi H. Construction of abstract state graphs with PVS. In: Proc. 9th International Conference on Computer Aided Verification(CAV'97). Vol 1254 of LNCS, Springer-Verlag, 1997. 72～83
22 Colon M, Uribe T. Generating finite-state abstractions of reactive systems using decision procedures. In: Computer Aided Verification, 1998. 293～304
23 Ball T, Podelski A, Rajamani S K. Boolean and Cartesian Abstraction for Model Checking C Programs. In: Proceedings of TACAS: Tools and Algorithms for the Construction and Analysis of Systems. Genova, Italy, April 2001
24 Ball T, Majumdar R, Millstein T, et al. Automatic Predicate Abstraction of C Programs. In: Proceedings of the Conference on Programming Language Design and Implementation(PLDI 2001), Snowbird, Utah, June, 2001
25 Das S, Dill D L, Park S. Experience with predicate abstraction. In Computer-Aided Verification, Vol 1633 of LNCS, Springer-Verlag, July 1999. 160～171
26 Clarke E, Grumberg O, Jha S, et al. Counterexample-guided abstraction refinement, In: Computer Aided Verification, 2000. 154～169
27 Clarke E, Kroening D, Sharygina N, et al. Predicate Abstraction of ANSI-C programs using SAT, Formal Method in System Design, 2004, 25: 105～127
28 Henzinger T A, Jhala R, Majumdar R, et al. Lazy Abstraction. In: Proceedings of the 29th Annual Symposium on Principles of Programming Languages(POPL). ACM Press, 2002. 58～70
29 Ball T, Rajamani S K. The SLAM project: debugging system software via static analysis. In: Proceedings of the 29th ACM SIGPLANSIGACT Symposium on Principles of Programming Languages. ACM Press, 2002. 1～3
30 SLAM. http://reseach. microsoft. com/slam
31 Java PathFinder. http://ase. arc. nasa. gov/visser/jpf
32 Visser W, Havelund K, Brat G, et al. Model Checking Programs. Proceedings of the 15th International Conference on Automated Software Engineering(ASE), Grenoble, France, September 2000
33 BLAST. http://www-cad. eecs. berkeley. edu/～rupak/blast
34 MAGIC. http://www. cs. cmu. edu/～chaki/magic
35 Chaki S J. A counterexample guided abstraction refinement framework for verifying concurrent c programs: [PhD thesis]. Carnegie Mellon University, 2005

(上接第 248 页)

加而减小，但 PA 算法的性能始终优于其它算法。

由此，可以得出可倒置法的性能优于基本回收算法，尽早启动算法以及约束向量法的结论。

结论 在实时多处理器系统中，为了提高系统的性能，使得新到达的任务有较多的机会得到执行，常常采取资源回收算法回收未被任务使用的资源。在研究已有资源收回算法的基础上，提出可倒置法。可倒置法通过允许只存在资源访问冲突的任务之间出现执行顺序倒置，提高资源回收效率，从而能够更好地回收资源。模拟结果表明，可倒置法优于基本回收算法、尽早启动算法以及约束向量法。

参 考 文 献

1 Ramamritham k, stankovic J A, Scheduling algorithms and operating systems support for real-time systems. Proceeding of IEEE, 1994, 82(1): 55～67
2 Shin K G, Ramanathan R. Real-time computing a new discipline of computer science and engineering. Proceedings of IEEE, 1994, 82(1): 6～24
3 Ramamritham K, Stankovic J A. Efficient scheduling algorithms for real-time multiprocessor systems. IEEE transactions on Parallel and distributed systems, 1989, 1(2): 184～194
5 Hou C J, Shin K G. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. IEEE transactions on Computers, 1997, 46(12): 1338～1356
6 Hong K S, Leung J Y T. On-line scheduling of real-time tasks. IEEE transactions on computer, 1992, 41(10): 1326～1331
7 Shen C, Ramamritham K, Stankovic J A. Resource reclaiming in multiprocessor real-time systems. IEEE Transactions on parallel and distributed systems, 1993, 4(4): 382～397
8 Manimaran G, Murthy C S R, Vigay M, et al. New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems. Journal of parallel and distributed computing, 1997, 44(2): 123～132
9 Gupta I, Manimaran G. A new strategy for improving the effectiveness of resource reclaiming algorithms in multiprocessor real-time systems. 1998