高技术通讯 2012 年 第22 卷 第3 期:231 ~239

doi:10.3772/j.issn.1002-0470.2012.03.002

基于改进 EDF 的多核处理器混合任务调度算法①

郭秀岩②*** 张 武③** 王劲林*** 吴 刚*

(*中国科学技术大学自动化系 网络传播系统与控制安徽省重点实验室 合肥 230027)

(**中国科学院声学研究所 国家网络与新媒体技术工程研究中心 北京 100190)

摘 要 为解决多核处理器系统中的实时任务调度问题,尤其是实时任务和非实时任务的混合调度问题,在对最早截止时间优先(EDF)算法进行改进的基础上,提出多核处理器混合任务调度算法——EDF-segment 算法。EDF-segment 算法可以整理调度混合任务时出现的碎片,并通过对碎片的迁移、合并提高处理器的利用率,从而提高系统处理混合任务的性能。通过 EDF-segment 算法不但可以解决混合任务的调度问题,还可以避免使用 EDF 算法时造成的多核处理器利用率下降,在保证实时任务处理延迟的前提下提升多核处理器的利用率。经过理论推导和实验分析证明,EDF-segment 算法可以有效地应用于多核处理器系统中。

关键词 混合任务调度,最早截止时间优先(EDF)算法,时间片整理,多核处理器

0 引 言

随着计算机技术的迅猛发展,各种新兴应用层出不穷。这些应用对任务处理的实时性、I/O 吞吐量等性能参数有极高的需求,传统的单核处理器的处理能力已经不足以满足这些需求。随着多核框架的逐渐流行,越来越多的应用以多核处理器作为实现的基础。多核处理器不但在实现难度和功耗上优于单核处理器,而且可以同时处理多个任务,能极大地提高系统的并行性,从而提高系统的总体性能。在使用多核处理器调度任务时,通常要通过有向无循环图(directed acyclic graph,DAG)将任务分解为子任务,根据子任务的特点对其进行调度。为了充分利用多核处理器在任务并行处理上的优势,以提高系统性能,必须合理地在多核间分配、调度任务。

将一个复杂的任务分解为多个子任务时,通常要按照固定的顺序执行子任务才能保证原始任务的正确完成,例如网络应用中的数据包处理,可以分解为数据包包头封装和数据包发送两个流程,但是要求数据包封装在数据包发送之前完成。又例如 IPsec 应用、数据流加密、路由器的数据包转发、数据包分类和视频点播系统中的数据流处理,都有类似的要求。这样就产生了一类特殊的混合任务系统,在该任务系统中所有的任务都由子任务组成,子任务有固定的执行顺序,相互之间有严格的依赖关系,并且最后一个子任务为实时任务,而之前的子任务为非实时任务。虽然传统的混合任务系统中也包含实时任务和非实时任务,但是通常实时任务和非实时任务之间没有严格的依赖关系。本文以数据包处理任务为例,总结出一类特殊的混合任务模型,结合最早截止时间优先(earliest deadline first,EDF)算法、Pfair 算法等多核任务调度算法研究了此类任务如何在多核处理器上进行调度,并对 EDF 算法进行了改进,提出了一种 EDF-segment 算法,通过该算法可以提高多核处理器在调度这类任务时的 CPU 利用率。

1 研究背景与相关工作

目前已经有很多的算法可以解决多核处理器上的任务调度问题,例如单调速率优先级(rate monotonic,RM)算法、EDF 算法和 Pfair 算法等。RM 算法以任务的周期为优先级调度任务,广泛用于单核处

① 863 计划(2008AA01A317)资助项目。

② 男,1982 年生,博士生;研究方向:网络传播系统与控制;E-mail: guoxy@dsp.ac.cn

③ 通讯作者,E-mail: zhangw@dsp.ac.cn

(收稿日期:2010-11-16)

理器上的周期性任务调度，文献[1]对 RM 算法在多核处理器上进行周期性任务调度的可行性进行了分析。文献[2]对 RM 算法进行了改进，提出了 RM-US $[m/(3m-2)]$ 算法，该算法保证在 m 个处理器的利用率不高于 $m^2/(3m-2)$ 时，可以成功地调度任何周期性任务。RM 算法以及基于 RM 的算法都属于静态调度算法，即要求多核系统中的任务周期不会发生频繁的变化，但是在多核处理器上调度实时任务时，任务周期通常会不断地变化，所以 RM 算法以及类 RM 算法不适用于调度这类任务。

与静态调度算法不同，EDF 算法[3]可用于动态优先级任务调度。EDF 算法以任务的截止期为优先级对任务进行调度，但是单纯的以此为依据进行任务调度并不能得到最优的调度结果，尤其是在多核环境中 EDF 算法并不是最优的调度算法。为了进一步改进 EDF 算法的性能，文献[4]提出了一种结合 EDF 算法和最小松驰度优先(last laxity first, LLF)算法的混合调度算法，该算法先对要调度的任务进行分组，然后用 EDF 为执行权重大于 1/2 的任务分配优先级，用 LLF 为执行权重低于 1/2 的任务分配优先级，该算法可以有效地减少使用 EDF 算法时造成的任务迁移。文献[5]提出了适用于大规模多核平台的混合 EDF(hybrid EDF, H-EDF)算法，该算法结合了分段 EDF 调度(partitioned EDF, P-EDF)算法和全局 EDF 调度(global EDF, G-EDF)算法，首先将核分为不同的组，然后使用 P-EDF 对任务进行划分到不同组中，最后使用 G-EDF 算法对组内的任务进行调度，并且保证任务只在组内进行迁移，这样可以有效地减少任务的迁移次数，从而减少任务错过截止期的概率。文献[6]提出的一种算法可以缩短任务组内子任务调度跨度(spread)，并由此提高任务在多核上调度的成功率，在文献中也给出了基于 EDF 算法的解决方案。文献[7]分析了 EDF 算法在统一多核处理器上调度硬实时任务的可行性和健壮性，分析了 EDF 算法用于统一多核处理器的可行性，并提出了适用于统一多核处理器的 EDF-feasible 算法。虽然已经证明 EDF 在用于多核任务调度时并不是最优的算法，但是由于 EDF 在实际应用中有算法复杂度低、对任务组扩展性好等优点，并且文献[8]和文献[9]指出，使用 G-EDF 算法造成的任务截止期延迟有固定上限，因此还是被广泛地用于解决多核系统中的任务调度问题。

为了更好地解决多核处理器上的实时任务调度问题，文献[10]提出了 Pfair 算法，在该算法中任务被分解为相同长度的子任务，并且按照不同任务组对处理器的使用权重进行调度。已经证明，当任务组对处理器的使用权重之和不超过处理器的处理能力时，使用 Pfair 算法可以得到最优的任务调度方案。文献[11]对 Pfair 算法进行了改进，提出 PD^2 算法，该算法改进了使用 Pfair 算法调度任务时对任务优先级的选择方式，提高了 Pfair 算法的执行速度。随后文献[12]对 Pfair 算法的使用条件进行了分析，并提出了及早释放(early-release, ER)任务模型，通过 ER 任务模型可以使任务组中的子任务提前调度，ER 任务调度模型已经被证明是符合 Pfair 算法可调度条件的，并且该算法可以提高 Pfair 算法的执行效率，所以 ER 任务模型也被广泛应用。文献[13]进一步改进了 Pfair 算法，提出了一种扩展 Pfair 算法，该算法通过重新计算任务的优先级提高 Pfair 在处理器超载情况下的性能和任务调度错失率。文献[14]对任务进行了分组，将可能造成处理器 L2 缓存和内存进行频繁数据交换的任务划分成组，并且分析了使用 Pfair 对这些任务组进行调度的可行性，通过实验结果证明了基于任务组的 Pfair 算法可以有效地提高 L2 缓存的命中率。在文献[15]中提出一种 EDF 和 Pfair 结合的算法，与传统的 EDF 算法相比该算法可以有效地减少任务调度时出现的任务迁移开销，提高任务调度的成功率。

虽然 EDF 算法和 Pfair 算法被广泛用于多核系统中的实时任务调度，但是它们在使用过程中都存在一定的问题。首先，以上提及的 EDF 算法都基于抢占式任务调度环境，而在实际应用中任务不一定是可抢占的，当系统中任务数量很多时频繁的任务抢占、切换会造成很大的系统开销；其次，Pfair 算法虽然可以应用于非抢占式任务调度中，但是文献[16]指出 Pfair 算法要求调度的子任务有相同的执行时间(如图 1 所示)，而子任务的长度通常不等或者很难分解，如图 2 所示。在使用多核平台发送数据包时，封装数据包的非实时任务和发送数据包的实时任务执行时间相差较大，并且很难将这些任务分解为长度相同的任务，所以当任务组中的子任务无法统一长度时，就需要将所有的子任务扩展为相同的长度才能使用 Pfair 算法，因此会有一定的计算资源被浪费。最后，在处理类似数据包这样的任务时，只有发送任务是实时任务，发送之前的封装任务为非实时任务，可以在发送任务之前的任何时刻执行，但是又一定要保证在发送任务到达时完成，传统的 EDF 算法和 Pfair 算法不能很好地解决这个问题。

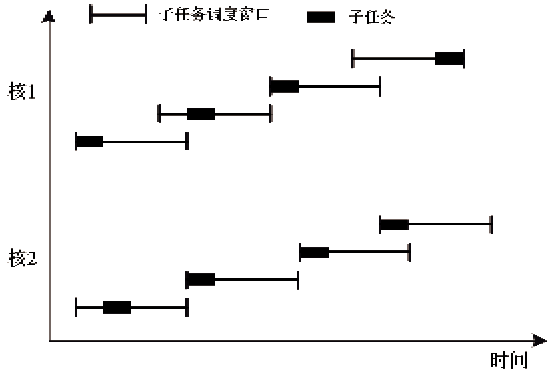

图 1　Pfair 多核调度算法

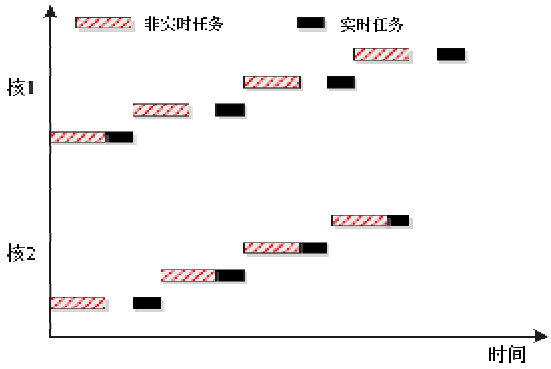

图 2　实时与非实时混合任务组

针对以上问题,本文基于 EDF 算法提出一种改进的 EDF-segment 算法,通过该算法可以调度这类特殊任务,并且还可以提高 EDF 算法对多核处理器计算资源的利用率。

2　系统模型和问题描述

2.1　系统模型

多核处理器按照计算能力和功能可分为同构多核处理器系统和异构多核处理器系统,本文的研究工作主要集中于同构多核处理器系统。为了便于描述混合任务调度问题,这里针对使用 EDF 算法的调度模型给出一些符号和函数定义。

设多核处理器平台包含 m 个核,将每个核的运行时间划分为等长的时间单位,定义函数

$$time(c,t) = \begin{cases} 0, & \text{核 } c \text{ 在时刻 } t \text{ 空闲} \\ 1, & \text{核 } c \text{ 在时刻 } t \text{ 工作} \end{cases} \tag{1}$$

用于表示核 c 在时刻 t 时是否空闲。

定义函数

$$idle(c,t_1,t_2) = \sum_{t=t_1}^{t_2} time(c,t) \tag{2}$$

用于表示核 c 在 $[t_1,t_2]$ 是否空闲。当 $idle(c,t_1,t_2) = 0$ 时,表示 $[t_1,t_2]$ 可以用于调度任务。

在 m 个核上调度 n 组作业,用 $T_i(T_{i1},T_{i2},\cdots)$ 表示每组作业,其中 T_{ij} 表示第 i 组作业中的第 j 个任务。每个任务 T_{ij} 包含 k 个子任务,用 $T_{ij,s}$ 表示每个子任务,其中 $1 \leqslant s \leqslant k$,设所有任务同一阶段的子任务有相同的处理时间,用 e_s 表示处理 $T_{ij,s}$ 需要的时间。用 $r_{ij,s}$ 表示 $T_{ij,s}$ 的释放时间(即任务开始在多核系统中执行的时间),$T_{ij,s}$ 子任务需要按固定顺序执行,因此应满足

$$r_{ij,s} + e_s < r_{ij,s+1},\ s = 1,2,\cdots,k-1 \tag{3}$$

在任务组 T_{ij} 中,只有 $T_{ij,k}$ 有实时性要求,$T_{ij,k}$ 的完成意味着 T_{ij} 任务的完成。用 p_{ij} 表示 T_{ij} 的执行周期,用 $w_{ij} = \sum_{s=1}^{k}(e_s)/p_{ij}$ 表示作业 i 要执行任务 j 时对处理器的利用率,用 $w_i = \frac{\sum_{j=1}^{s} w_{ij}}{s}$ 表示作业 i 对处理器的平均利用率,其中 s 为作业 i 已经调度的任务数量。

由于在网络应用中任务通常由非实时任务和实时任务混合组成,如数据包的封装和数据包发送,所以每个任务的处理可简化为 2 部分,即 $k = 2$。因为 $T_{ij,2}$ 子任务有实时性要求,所以按照 $T_{ij,2}$ 子任务的释放时间 $r_{ij,2}$ 依次将所有 T_{ij} 任务加入到优先级队列,使用非抢占式 EDF 算法对每个 T_{ij} 任务进行调度,具体流程如下:

(a)选择优先级队列中任务组 T_{ij},并满足

$$\forall x \neq i,\ y \neq j$$
$$r_{ij,2} \leqslant r_{xy,2} \tag{4}$$

(b)选择最早可用的核 c_1 来执行 $T_{ij,1}$ 以保证子任务 $T_{ij,2}$ 能够在截止期之前执行,一旦确定可用的核立即执行 $T_{ij,1}$,所以核 c_1 的最早可用时间即为 $T_{ij,1}$ 释放时间 $r_{ij,1}$:

$$R = \{(c,t) \mid idle(c,t,t+e_1) = 0,\ 0 < t, 0 < c \leqslant m\}$$
$$\text{取}(c_1,t_1) \in R,\ \text{满足}\ \forall (c,t) \in R,\ t_1 \leqslant t$$
$$r_{ij,1} = t_1 \tag{5}$$

(c)再次选择最早可用的核 c_2 来执行子任务 $T_{ij,2}$。$T_{ij,2}$ 为实时任务,且要保证条件式(3),因此 c_2 的定义如下:

$$t_2 = \max\{r_{ij,2}, r_{ij,1} + e_1\}$$
$$c_2 = \min_{0<c\leqslant m}\{c \mid idle(c,t_2,t_2+e_2) = 0\} \tag{6}$$

2.2 问题描述

为保证任务 T_{ij} 可以在多核处理器上调度，要求在任意时刻作业对处理器的利用率满足

$$\frac{\sum_{i=1}^{n} w_i}{\min(w_i)} \leqslant m \tag{7}$$

否则将不能保证所有任务中的 $T_{ij,2}$ 子任务都能按时调度。

上述可行性条件只适用于可抢占式 EDF 任务调度算法，当任务不可抢占时，即使计算资源足够时任务组也不一定能够在多核上执行。这一点可通过图 3 来说明，从图 3 中可以看出，使用 EDF 算法在 4 个核上进行任务调度，每个核的处理时间将会被 $r_{ij,2}$ 划分为许多的区间，在这些区间内调度 $T_{ij,1}$ 后将产生大量的时间碎片，如果这些时间碎片不能用于调度任务则会导致处理器计算资源被浪费。如图 3 所示，时间碎片 1、2、3 的长度小于 e_1，因此不能用于调度 $T_{ij,1}$ 子任务，并且 EDF 算法以 $r_{ij,2}$ 为优先级调度任务组，所以也不会有 $T_{ij,2}$ 子任务在这些时间碎片中调度，处理器的利用率因此而下降。

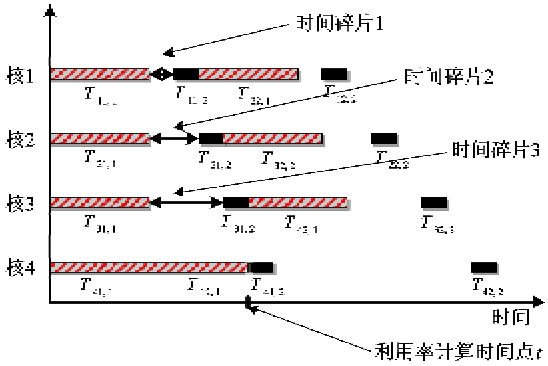

图 3 使用 EDF 算法时造成的时间碎片

定义多核处理器的利用率为

$$f(t,m) = \sum_{i=1}^{m} \frac{idle(i,0,t)}{t} \tag{8}$$

$f(t,m)$ 约束于条件式(5)和(6)。式(6)中 t_2 选择 $r_{ij,2}$ 时表示 $T_{ij,2}$ 子任务按时执行，选择 $r_{ij,1} + e_1$ 时表示 $T_{ij,2}$ 子任务将延迟执行，定义每个任务的执行错失率(deadline miss)为

$$d(t,n) = \sum_{i=1}^{n} \sum_{t_2=0}^{t} \frac{t_2 - r_{ij,2}}{p_{ij}} \tag{9}$$

其中 t 为利用率计算时间点，T_{ij} 为时刻 t 之前调度的任意一个任务，t_2 为 $T_{ij,2}$ 子任务实际执行时间点。

为了同时保证处理器利用率和实时任务执行的准确性，最终要解决的是约束于条件式(5)和(6)的多目标优化问题：

$$\begin{cases} \max(f(t,m)) \\ \min(d(t,n)) \end{cases} \tag{10}$$

为了解决这个问题，本文提出 EDF-segment 算法，通过 EDF-segment 算法可以有效地提高各个处理器的使用效率。与 EDF 算法相比，使用 EDF-segment 算法的多核处理器可以在相同的时间内调度更多的任务，并且能整体降低实时任务的错失率。

3 算法与分析

3.1 算法

由图 3 可以发现，使用 EDF 算法在多核系统中调度任务时将会出现时间碎片，如果在时间碎片内不能调度任务，该区间内的计算资源将会被浪费，因而降低处理器的利用率。同时由于时间碎片的出现，会推迟处理器最早可用时间，即导致式(5)中 t_1 增加，当系统中处理的作业数量增加，或者 T_{ij} 执行周期 p_{ij} 缩短时，将会导致 $r_{ij,2} < r_{ij,1} + e_1$，即 $T_{ij,1}$ 子任务不能在 $T_{ij,2}$ 子任务释放之前完成，所以式(6)中的 t_2 将选择 $r_{ij,1} + e_1$，即 $T_{ij,2}$ 子任务将被延迟执行。因此，时间碎片的产生不但会降低处理器的利用效率，还会间接地影响处理器的可服务能力。

如果调整图 3 中 $T_{ij,2}$ 子任务位置，将会减少或者消除部分时间碎片。图 4 所示为使用 EDF-segment 算法时造成的时间碎片，从图中看出，将 $T_{11,2}$ 从核 1 上迁移到核 3，则不会出现时间碎片 1，并且可以将 $T_{22,1}$ 子任务的调度时间提前，这样核 1 在执行完 $T_{22,1}$ 之前的利用率可以达到 100%，同理如果能消除时间碎片 2 和时间碎片 3，核 2 和核 3 的利用率也会得到提升，并且还可以将 $T_{32,1}$ 子任务的调度时间提前。

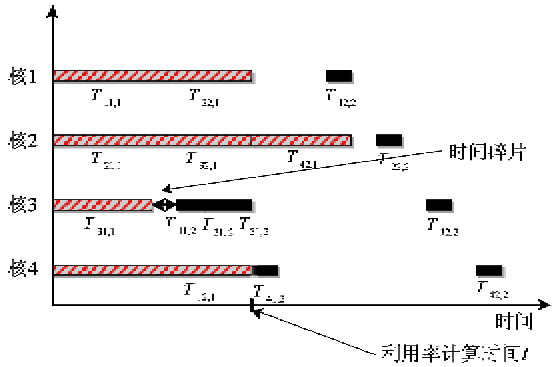

图 4 使用 EDF-segment 算法时造成的时间碎片

本文提出 EDF-segment 算法整理多核处理器上的时间碎片,以提高处理器的使用效率,降低实时子任务 $T_{ij,2}$ 错失率。EDF-segment 算法的主要思想是,以实时任务 $T_{ij,2}$ 的释放时间建立优先队列,使用 EDF 算法从优先队列中选择任务组 T_{ij},依次选择最先空闲的核来执行 $T_{ij,1}$ 和 $T_{ij,2}$ 子任务,每次调度 $T_{ij,2}$ 子任务之后要在一个时间窗口内检测其他核上 $T_{ij,2}$ 子任务的执行时间,落入该时间窗口的 $T_{ij,2}$ 子任务将被迁移到当前核上执行,通过不断地进行时间碎片整理,可以有效地提高处理器的使用效率。如图 4 所示,对图 3 中的任务调度使用 EDF-segment 算法,将 $T_{11,2}$ 从核 1 迁移到核 3,将 $T_{21,2}$ 从核 2 迁移到核 3,这样就消除了图 3 中的时间碎片 1 和时间碎片 2,不但提高了核 1、核 2 和核 3(考察利用率计算时间点 t 之前的情况)的利用率,还将 $T_{22,1}$、$T_{32,1}$ 子任务的调度时间提前,这样可以保证 $T_{22,2}$ 和 $T_{32,2}$ 按时释放,降低 $T_{22,2}$ 和 $T_{32,2}$ 的错失率。

EDF-segment 算法的伪代码描述如下:

```
初始化优先队列 priority;
初始化 n 个作业;
For i = 1 to n
{
        //将第 i 个作业中的第 1 个实时子任务加入到优先队列中
        Priority. in (T_{i1,2});
}

While not priority. empty( )
{
    //使用 EDF 算法选择要调度的任务组
T_ij = priority. out( );
    //找到最先空闲的核调度 T_{ij,1} 子任务
    (core,time) = 使用式(5)确定可用于调度 T_{ij,1} 的核 core 和时间 time
    Schedule(core, time, T_{ij,1}); //在 time 处调度非实时子任务 T_{ij,1}
    (core,time) = 使用式(6)确定可用于调度 T_{ij,2} 的核 core 和时间 time
    Schedule(core, time, T_{ij,2}); //在 time 处调度实时子任务 T_{ij,2}
    //以 T_{ij,2} 的实际调度时间为碎片整理窗口的右边界进行碎片整理
    Handle _ segment(core, time,e_1);
    //将第 i 个作业中的下一个实时子任务加入到优先队列中
    Priority. in( T_{i(j+1),2});
}
```

Handle _ segment(core, time, e_1)的具体描述如下:

```
//确定当前核从时间 time 向前的空闲时间长度
for t = time to 0
{
        //coretime 表示当前核的使用情况,coretime[core][t] = =0 表示核 core 在时间 t 没有被使用,反之则被使用
        if coretime[core][t] ! = 0
            break;
}
idle _ size = time - t;
//确定碎片整理窗口长度
window _ size = idle _ size % e_1   //% 为取模运算,由 4.2定理 1 证明该操作的必要性和正确性
for c = 1 to m
{
    //core 为刚刚调度了实时任务的核,将其他核上落入碎片整理时间窗的实时任务调度到这个核上
        if c = = core
            continue;
        判断核 c 在[time-window _ size, time]内是否有实时子任务 T_{ij,2} 执行,如果有则将其迁移到核 core 的相应位置执行
}
```

每次执行 Handle _ segment 之后,所有核上落入一个时间窗口内的实时子任务 $T_{ij,2}$ 都被整理到同一个核上,这样就可以有效地减少系统中的时间碎片。下面证明 EDF-segment 算法的正确性和时间窗口的选择。

3.2 EDF-segment 算法分析

定理 1 设 T_{ij} 和 T_{xy} 是 2 个依次被调度的任务组,由式(6)分别计算出 $T_{ij,2}$、$T_{xy,2}$ 的实际执行时间 t_{2j}、t_{2y}。为保证 $f(t,m)$ 不会降低,在 EDF-segment 算法中进行时间碎片整理的窗口不应超过 e_1,即 window _ size < e_1。

证明:首先证明当 $T_{xy,2}$ 的窗口 window _ size 足够大,且 t_{2y} - window _ size < t_{2j} 时,$T_{ij,2}$ 一定可以加入这个窗口。

在调度 $T_{ij,2}$ 子任务时,将出现两种情况:

(1) $t_{2j} = r_{ij,2}$,即 $T_{ij,2}$ 按时发送。由 EDF 算法的性质可知,$r_{ij,2} \leqslant r_{xy,2}$,所以有

$$t_{2j} = r_{ij,2} \leqslant r_{xy,2} \leqslant t_{2y}$$

t_{2y} 为窗口的右边界,当 t_{2y} - window _ size < t_{2j} 时,$T_{ij,2}$ 可以加入该窗口;

(2) $t_{2j} = r_{ij,1} + e_1$,即 $T_{ij,2}$ 延时执行。由 EDF 算法的性质和第 3 节中的系统模型设定可知,T_{ij} 任务组先于 T_{xy} 任务组被调度,则 $r_{ij,1} \leqslant r_{xy,1}$,所以有 $r_{ij,1} + e_1 \leqslant r_{xy,1} + e_1$,$r_{ij,1} + e_1 = t_{2j} \leqslant r_{xy,1} + e_1 \leqslant t_{2y}$,$t_{2y}$ 为

窗口的右边界，当 t_{2y} - window _ size < t_{2j} 时，$T_{ij,2}$ 可以加入该窗口。

所以当 window _ size 足够大，且 t_{2y} - window _ size < t_{2j} 时，$T_{ij,2}$ 一定可以加入这个窗口。

下面证明 window _ size 应该满足 window _ size < e_1，假设 window _ size ≥ e_1。

如果 $T_{ij,2}$ 不满足 t_{2y} - window _ size < t_{2j} < t_{2y}，则执行 handle _ segment 时不会进行碎片整理，t_{2y} 之前的多核处理器利用率 $f(t_{2y},m)$ 和错失率 $d(t_{2y},n)$ 保持不变；

如果 $T_{ij,2}$ 满足 t_{2y} - window _ size < t_{2j} < t_{2y}，将对 $T_{ij,2}$ 进行碎片整理。当出现图 5 所示情况时，$f(t,m)$ 将会下降：

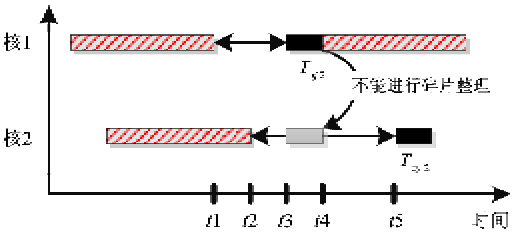

图 5 任务迁移对窗口的分割

在核 1 上调度 $T_{ij,2}$ 时出现了时间碎片 $[t_1,t_3]$，且 $t_3 - t_1 < e_1$，不足以调度 $T_{ij,1}$，由之前的推导可知，该区间也不会用于调度 $T_{ij,2}$，所以 $[t_1,t_3]$ 一定会被浪费；在核 2 上时间碎片 $[t_2,t_5]$ 长度足以调度 $T_{ij,1}$，但是进行了碎片整理后，$[t_2,t_5]$ 被分为 $[t_2,t_3]$ 和 $[t_4,t_5]$，长度都不足以调度 $T_{ij,1}$，并且也不会用于调度 $T_{ij,2}$，所以 $[t_2,t_3]$ 和 $[t_4,t_5]$ 一定会被浪费。$t_3 - t_2 + t_5 - t_4 > t_3 - t_1$，所以经过碎片整理后处理器利用率 $f(t,m)$ 将会下降，与定理中的假设矛盾。证毕。

通过定理 1 可知，当窗口过大时，引入其他核上的 $q-1$ 个 $T_{ij,2}$ 任务后，会将窗口分为 q 个子窗口，窗口分割之前的时间浪费为 window _ size % e_1，分割之后的时间浪费为

$$\sum_{i=1}^{q}(window\_size(i)\ \%e_1) \qquad (11)$$

在最坏情况下每个子窗口都被浪费，式(11)中的值为 window _ size，大于 window _ size % e_1。所以，在使用 EDF-segment 算法时，窗口的大小会影响碎片整理的效果。

定理 2 当窗口的长度满足 window _ size < e_1 时，与 EDF 算法相比，EDF-segment 算法不会降低处理器利用率 $f(t,m)$。

证明：设进行碎片整理的窗口在核 1 上。由定理 1 的证明过程可知，当 window _ size < e_1 时，窗口内无法调度任何任务，因此窗口一定被浪费，如果没有其他 $T_{ij,2}$ 迁移到当前窗口，则时间浪费为 window _ size。如果有 $T_{ij,2}$ 从核 2 迁移到当前窗口，则核 2 上由于调度 $T_{ij,2}$ 产生的时间碎片将被消除，核 1 上的时间碎片长度不变，系统中总时间浪费缩短，$f(t,m)$ 增大。证毕。

通过以上两个定理可以证明，当碎片整理窗口长度满足 window _ size < e_1 时，EDF-segment 可以提高多核处理器的利用率 $f(t,m)$。

4 实验结果及分析

为了验证 EDF-segment 算法的性能，本文设计了一个试验系统。该系统模拟 1 个包含 m 个核的多核处理器，并在该处理器上运行 n 个作业，每个作业中的任务 T_{ij} 执行周期 p_{ij} 满足 [1500,6500] 上的均匀分布。T_{ij} 包含两个子任务，$T_{ij,1}$ 子任务没有实时性要求，该任务的释放时间即是执行时间，$T_{ij,2}$ 为实时子任务，该子任务的释放时间约束于式(6)，即一旦 $T_{ij,1}$ 的完成时间出现延迟，$T_{ij,2}$ 的执行时间也要相应顺延。设 e_1 = 100，e_2 = 5。

首先考察 m = 4 时，EDF 算法和 EDF-segment 算法在不同负载下的性能。图 6 给出了下同负载下两种算法的性能对比。

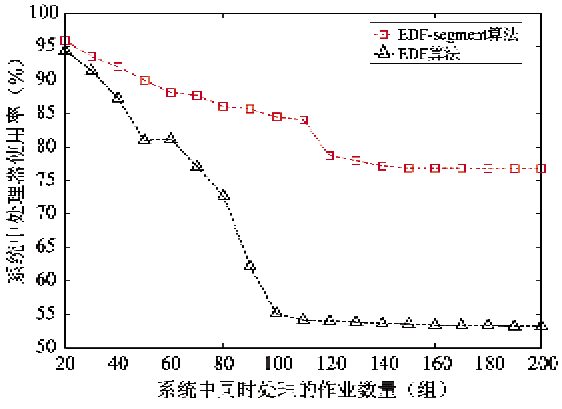

图 6 不同负载下算法性能对比

从图 6 中可见，随着系统中处理的作业数量的增加，固定时间内要处理的实时任务 $T_{ij,2}$ 随之增加。由于每个作业起始释放任务的时间以及释放任务的周期不同，这就在时间上将处理器的计算资源分为若干区间，并且每个区间都会出现不同程度的时间

浪费,随着要处理的实时性任务 $T_{ij,2}$ 的增加,区间的数量也随之增加,处理器的利用率随之下降。

由于 EDF-segment 算法能够对 $T_{ij,2}$ 任务进行整理,将多个浪费的时间碎片整理到同一个窗口内,因此 EDF-segment 算法可以提高处理器的利用率,从图 6 中可以得到同样的结论。

从图 6 中还可以发现,当作业的数量超过 90 后,使用 EDF 算法的系统处理器使用率迅速下降,而使用 EDF-segment 算法的系统没有明显的下降,主要原因是系统中作业数量的增加,导致在同一时间段内要执行的任务 T_{ij} 数量增加,因此要处理的 $T_{ij,1}$ 子任务增加,但此时 $m \leqslant \sum w_i$,与式(7)冲突,所以导致延迟执行的 $T_{ij,2}$ 任务逐渐增多,即大量的 $T_{ij,2}$ 任务在式(6)中选择 $r_{ij,1} + e_1$。假设系统中 m 个处理器按照最先空闲次序排列,t_i 对应核 i 的最早空闲时间,$i = 1,2,\cdots,m$。当 m 为偶数,$tm - t_1 \leqslant e_1$ 时,使用 EDF 算法的系统将出现如图 7 所示的情况。

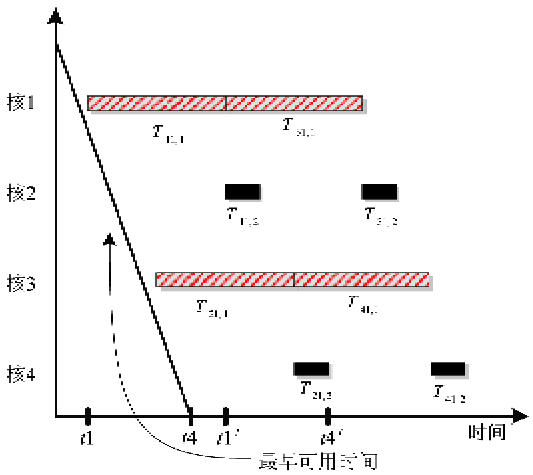

图 7　负载过高时 EDF 算法出现的问题

$T_{11,1}$ 在核 1 的 t_1 时刻调度,由于 $T_{11,2}$ 被延迟调度,虽然核 2 已经空闲,但还是要等到核 1 上的 $T_{11,1}$ 执行完毕才能执行,同理在核 3 上调度 $T_{21,1}$,在核 4 调度 $T_{21,2}$,而此时所有核的最早可用时间顺序与调度 $T_{11,1}$ 之前相同,并且 $t_4 - t_1' < e_1$,因此导致后续的所有 $T_{ij,1}$ 任务在核 1 和核 3 上执行,$T_{ij,2}$ 任务在核 2 和核 4 上执行。核 1 和核 2 的利用率为 100%,而核 2 和核 4 的利用率为 e_2/e_1,系统总体利用率为 $(1 + e_2/e_1)/2$。但是使用 EDF-segment 算法时,$T_{11,2}$ 和 $T_{31,2}$ 会被整理到核 4 上,将不会出现上述问题,如图 8 所示。

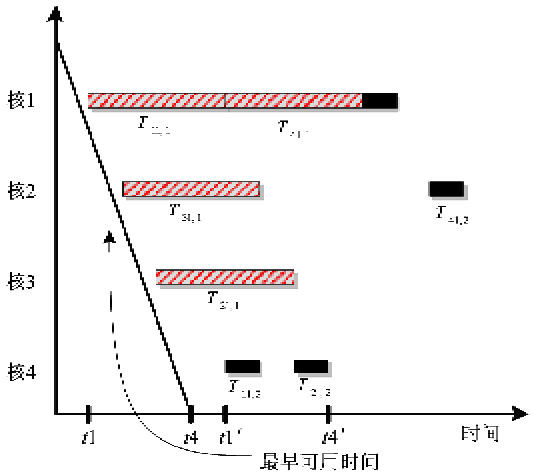

图 8　负载过高时使用 EDF-segment 算法

其次观察 EDF-segment 算法在不同负载下对实时任务执行延迟的优化。使用与图 6 相同的实验条件,可得到如图 9 所示的结果。

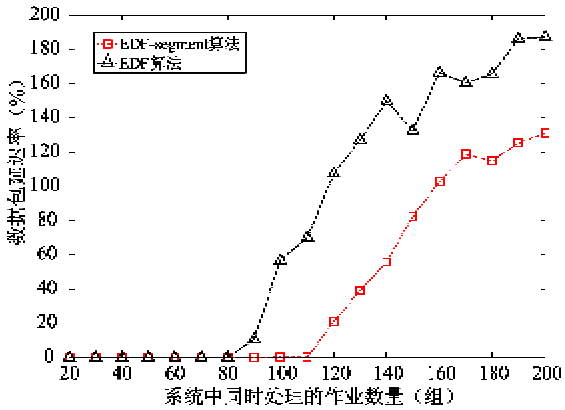

图 9　系统可支持的最大作业

由图 9 可见,EDF 算法只能保证 80 个作业不出现延迟,而 EDF-segment 算法可以保证 110 个作业正常执行。这就说明 EDF-segment 不但可以提高处理器的使用效率 $f(t,m)$,还可以通过对 $T_{ij,2}$ 任务使用碎片整理减少系统中的碎片。碎片的减少可以保证 $T_{ij,1}$ 得到更早的调度时间,从而保证 $T_{ij,2}$ 任务不会被延迟或者减少其延迟时间,这一点可以从图 9 中观察到,使用 EDF-segment 算法的系统中任务延迟率始终低于使用 EDF 算法的系统。

最后观察 EDF 算法和 EDF-segment 算法对系统中任务出现延迟概率的影响:

与图 9 给出的任务延迟时间率结果相似,EDF-segment 算法相对于 EDF 算法而言,在处理相同数量的作业时,可以降低实时任务出现延迟的概率。

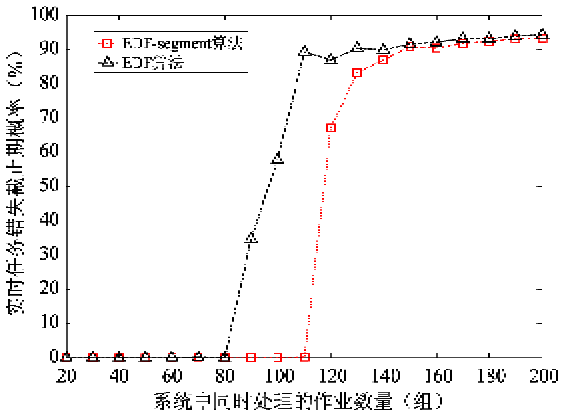

图 10 任务延迟概率

综上所述，使用 EDF-segment 算法的多核处理器相对使用 EDF 算法的多核处理器可以保持更高的 $f(t,m)$ 和更低的 $d(t,n)$。

5 结 论

多核处理器的使用逐渐成为一种趋势，但是在实际应用中应当根据不同的情况，制定合理的任务调度策略。例如在调度混合任务时，尤其是类似本文提出的具有相互依赖性的混合任务，虽然可以采用通用的任务调度算法，但是在实时任务的实时性、处理器资源的利用率等方面还有提升的空间。本文引进具有相互依赖性的混合任务模型，分析了多核处理器上实时任务调度算法，通过对 EDF 算法进行改进，提出了可以进一步提高系统利用率和降低实时任务处理延迟的 EDF-segment 算法，并且通过理论推导和实验对其正确性和性能进行了验证。

参考文献

[1] Baruah S K, Goossens J. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Transactions on Computers*, 2003, 52(7):966-970

[2] Andersson B, Baruah S, Jonsson J. Static-priority scheduling on multiprocessors. In: Proceedings of the 22nd IEEE International Conference on Real-Time Systems Symposium, London, England, 2001. 193-202

[3] Chetto H, Chetto M. Some results of the earliest deadline scheduling algorithm. *Software Engineering, IEEE Transactions on*, 1989,15(10):1261-1269

[4] Lee L T, Chang H Y, Chao S W. A hybrid task scheduling for multi-Core platform. In: Proceedings of the 2nd International Conference on Future Generation Communication and Networking Symposia, Sanya, China, 2008. 40-45

[5] Calandrino J M, Anderson J H, Baumberger D P. A hybric real-time scheduling approach for large-scale multicore platforms. In: Proceedings of the 19th Euromicro Conference on Real-Time Systems, Pisa, Italy, 2007. 247-258

[6] Anderson J H, Calandrino J M. Parallel task scheduling on multicore platforms. In: Proceedings of the 16th ACM Conference on Special Interest Group on Embedded Systems, New York, USA, 2006. 1-6

[7] Baruah S, Funk S, Goossens J. Robustness results concerning EDF scheduling upon uniform multiprocessors. *IEEE Transactions on Computers*, 2003, 52(9): 1185-1195

[8] Devi U M C, Anderson J H. Tardiness bounds under global EDF scheduling on a multiprocessor. In: Proceedings of the 26th IEEE International Conference on Real-Time Systems Symposium, Miami, USA, 2005. 330-341

[9] Valente P, Lipari G. An upper bound to the lateness of soft real-time tasks scheduled by EDF on multiprocessors. In: Proceedings of the 26th IEEE International Conference on Real-Time Systems Symposium, Miami, USA, 2005. 311-320

[10] Baruah S K, Cohen N K, Plaxton C G, et al. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 1996, 15(6): 600-625

[11] Baruah S K, Gehrke J E, Plaxton C G. Fast scheduling of periodic tasks on multiple resources. In: Proceedings of the 9th IEEE International Conference on Parallel Processing Symposium, New York, USA, 1995. 280-288

[12] Anderson J H, Srinivasan A. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. In: Proceedings of the 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands, 2001. 76-85

[13] Zhou B H, Qiao J Z, Lin S K. Research on synthesis parameter real-time scheduling algorithm on multi – core architecture. In : Proceedings of the 21th Control and Decision Conference, Guilin, China, 2009. 5116-5120

[14] Anderson J H, Calandrino J M, Devi U C. Real – time scheduling on multicore platforms. In: Proceedings of the 12th IEEE International Conference on Real-Time and Embedded Technology and Applications Symposium, California, USA, 2006. 179-190

[15] Kumar K P, Shanthi A P. Pfairness applied to EDF to reduce migration overheads and improve task schedulability in multicore platforms. In: Proceedings of the 38th IEEE International Conference on Parallel Processing Workshops, Vienna, Austria, 2009. 35-41

[16] Anderson J H, Calandrino J M. Parallel real-time task scheduling on multicore platforms. In: Proceedings of the 22nd IEEE International Conference on Real-Time Systems Symposium, Rio de Janeiro, Brazil, 2006. 89-100

A hybrid task scheduling scheme for multicore processors based on improving the EDF algorithm

Guo Xiuyan*, Zhang Wu**, Wang Jinlin**, Wu Gang*

(*Joint Laboratory of Network Communication System & Control, Department of Automation, University of Science and Technology of China, Hefei 230027)

(**National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190)

Abstract

In order to schedule the real-time tasks in multicore processor systems, especially to handle the hybrid tasks composed of real-time tasks and non-real-time tasks, this paper presents a hybrid task scheduling algorithm based on the improvement of the earliest deadline first (EDF) algorithm, called the EDF-segment algorithm. The EDF-segment algorithm can arrange the segments caused by hybrid task scheduling, and migrate and merge segments between cores to enhance the processor utilization. The EDF-segment algorithm can not only solve the hybrid tasks scheduling, but also avoid the reduction of the multicore processor utilization by using the EDF algorithm, and enhance the utilization with the guarantee that real-time tasks can be processed on time. Proved by theoretical and experimental analysis, the EDF-segment algorithm can be effectively applied to multicore processor systems.

Key words: hybrid task schedule, earliest deadline first (EDF)-based algorithm, time slice arrange, multicore processor