第36卷 第19期 计 算 机 工 程 2010年10月
Vol.36 No.19 Computer Engineering October 2010
·博士论文· 文章编号：1000—3428(2010)19—0001—04 文献标识码：A 中图分类号：TP311

基于 Lebesgue 采样的动态反馈实时调度模型

秦承刚[1,2]，于 东[1]，吴文江[1]，丁万夫[1,2]，胡 毅[1,2]
(1. 中国科学院研究生院，北京 100039；2. 中国科学院沈阳计算技术研究所，沈阳 110004)

摘 要：提出一种基于 Lebesgue 采样方法和弹性调度算法的动态反馈实时调度模型。通过调整实时任务的执行速率，使软实时系统的系统负载始终保持在参考值以下。利用硬件看门狗技术在系统过载时产生中断，实现基于事件的 Lebesgue 采样。在实时操作系统 RTAI 中实现该调度模型，并对模型的暂态性能和稳态性能进行分析验证。实验结果表明，该模型不仅保持了系统的稳定性，还能显著降低调度算法的系统开销。

关键词：动态反馈调度；Lebesgue 采样；弹性调度算法

Dynamic Feedback Real-time Scheduling Model Based on Lebesgue Sampling

QIN Cheng-gang[1,2], YU Dong[1], WU Wen-jiang[1], DING Wan-fu[1,2], HU Yi[1,2]
(1. Graduate University of Chinese Academy of Sciences, Beijing 100039, China;
2. Shenyang Institute of the Computing Technology, Chinese Academy of Sciences, Shenyang 110004, China)

【Abstract】 This paper presents a dynamic feedback real-time scheduling model based on Lebesgue sampling and elastic scheduling algorithm. The workload of soft real-time system can be held below the reference value by adjusting the task rate. An interrupt can be triggered while system is overload, and the scheduling model can be regarded as an event-based system. The mechanism is realized by a watch dog. The scheduling model is realized in the RTAI real-time system, and the model's dynamic characteristics and steady state characteristics are tested. Experimental results of test show the model can reduce the workload of task scheduling, while the system is steady.

【Key words】 dynamic feedback scheduling; Lebesgue sampling; elastic scheduling algorithm

1 概述

在动态不可预测的实时系统中，经典实时调度算法很难持续稳定地提供性能保证[1]，这是因为经典实时调度算法在系统过载时不再是最优调度。而在动态不可预测环境下，系统负载会有较大的波动，经常出现过载的情况。反馈调度可以有效地弥补经典实时调度算法的不足。它能根据实时系统的当前状态，通过调整任务参数动态地调节系统负载，确保系统的实时性能。

文献[2-3]提出一种通过调节实时任务的周期来控制系统负载的反馈弹性调度模型。当系统过载时，可以通过延长任务的执行周期实现系统的优雅降级，保证任务集的可调度性。文献[4]在文献[2]的基础上提出一种基于迭代的周期延展算法。该方法主要针对实时任务的时限小于周期的情况。但文献[4]仅考虑了在系统过载时对周期的延展，没有给出系统轻载时压缩任务周期的策略。文献[5]将弹性调度模型应用在了控制系统的设计中，减轻了系统负载的变化对控制器性能的影响。目前提出的弹性调度算法都具有较大的执行开销，在资源紧张的系统中，难以充分发挥系统的性能。

在反馈控制系统中，采样方法可以分为时间驱动的 Riemann 采样和事件驱动的 Lebesgue 采样。目前的反馈调度算法都使用了 Riemann 方法，这种方法的分析和设计过程简单，但控制程序的频繁运行也带来了很大的系统开销[6]。Lebesgue 方法是基于事件的采样方法，当系统离开稳态时会触发一个事件，不需要进行周期性的采样。

本文在弹性调度模型的基础上，提出一个由弹性调度算法和完全基于事件的 Lebesgue 采样方法组成的动态反馈调度模型 FES-L。该模型在资源较为紧张的动态不可测系统中，能够显著提高系统的稳定性和资源利用率。

2 FES-L 调度模型

2.1 基本假设和定义

本文对任务模型的基本假设是，每个软实时任务的周期都有一个变化区间，调度程序动态地在此区间内调整任务的周期。为了方便说明，本文假设实时系统中仅含周期性实时任务，但本文的结论同样适用于含有非周期任务的混合实时系统。同时本文还假设，除了处理器之外，任务间不共享其他资源。下文给出一些重要概念的定义。

定义 1 软实时任务的调度参数。本文将软实时任务 $Task_i$ 用四元组表示为：

$$Task_i=(T_{i\mathrm{Min}},T_{i\mathrm{Max}},T_i,W_i)$$

其中，$T_{i\mathrm{Min}}$ 为 $Task_i$ 的最小周期；$T_{i\mathrm{Max}}$ 为 $Task_i$ 的最大周期；T_i 为 $Task_i$ 当前的周期，T_i 的变化范围是 $[T_{i\mathrm{Min}},T_{i\mathrm{Max}}]$，$T_{i\mathrm{Min}}$ 也可

基金项目：国家科技重大专项基金资助项目(2009ZX04009-013)；国家科技支撑计划基金资助项目(2007BAP20B01)
作者简介：秦承刚(1983－)，男，博士研究生，主研方向：实时操作系统；于 东，研究员、博士生导师；吴文江，研究员；丁万夫、胡 毅，博士研究生
收稿日期：2010-04-18 **E-mail：**qinchenggang@sict.ac.cn

以称为 $Task_i$ 的最佳周期，在此周期下， $Task_i$ 能够提供最好的 QoS；T_{iMax} 是在 QoS 可以接受的前提下， $Task_i$ 周期的最大值， $T_i \neq T_{iMin}$ 的任务称为被延展任务；W_i 为 $Task_i$ 的权重，表示该任务周期的调整对系统负载变化的影响程度。可以为不重要的任务分配较大的权重，此类任务的周期会优先被调整。

定义 2 任务的处理器利用率 u_i ，指 $Task_i$ 在一个周期 T_i 内的执行时间 C_i 与周期 T_i 的比值：

$$u_i = \frac{C_i}{T_i} \tag{1}$$

定义 3 系统的实际负载 U_{real} 是指为在时间段 T_{WG} 内，处理器忙的时间所占的比例。T_{WG} 称为参考时间段。U_{real} 等于系统中各任务的处理器利用率之和再加上系统开销 U_{sys}。

$$U_{real} = \sum_{i=1}^{n} u_i + U_{sys} \tag{2}$$

定义 4 系统负载的参考值 U_{ref}，调度算法的参数之一，反馈调度算法的目标是使系统的实际负载尽可能的接近 U_{ref}。当 $U_{real} > U_{ref}$ 时，系统过载。

2.2 调度模型的结构

动态反馈调度模型 FES-L 的结构如图 1 所示，其组件包括控制器、看门狗、基本调度器、到达队列和就绪队列。

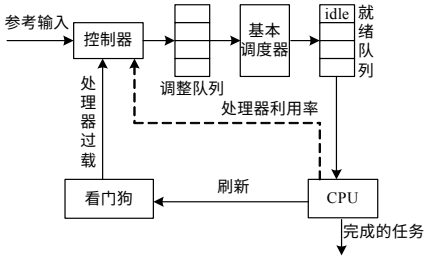

图 1 调度模型的结构

控制器的作用是根据周期调整算法在系统过载时延展调整队列中各任务的周期。为了简化周期调整算法，调整队列中的各任务按照权重从大到小排列。基本调度器使用 EDF 算法，将调整队列中已就绪的任务插入就绪队列，CPU 将依次执行就绪队列中的任务。看门狗用来判断系统是否过载。

系统的控制变量是系统的实际负载 U_{real}。参考输入是处理器利用率的参考值 U_{ref}。对 EDF 算法而言，只要处理器的利用率不超过 1 即可。但为了增强系统的稳定性，需要为此值留取一定的裕度，本文中选取参考输入 $U_{ref} = 0.9$。

2.3 周期调整算法

本文采用贪婪算法实现了实时任务的周期延展和压缩。在处理器过载时，需要对周期进行延展。首先计算系统的实际负载与其参考值的差 ΔU ，然后按照权重的顺序，依次增大各任务的周期。根据贪婪算法，先扩展权重最高的任务，若其最大的周期仍不能将 ΔU 降到 0，就再扩展权重次高的任务。因为 $u_i = C_i / T_i$ ， $Task_i$ 的新周期与计算时间及 ΔU 之间存在如下关系：

$$\frac{C_i}{T_{iNew}} = \frac{C_i}{T_i} - \Delta U \tag{3}$$

此处利用 $Task_i$ 的当前计算时间 C_i 来近似地估计该任务今后的计算时间。根据式(3)，可推导出 $Task_i$ 新周期计算公式：

$$T_{iNew} = \frac{C_i T_i}{C_i - \Delta U T_i} \tag{4}$$

该算法运行到 Δu 分配完毕，或所有任务的周期都达到了最大值时才结束。

如果系统负载恰在 U_{ref} 附近时，可能会导致负载频繁地超过 U_{ref} ，从而引起看门狗频繁地给出中断，使 Lebesgue 采样失去了优势。为了避免这种现象的发生，本文使用了一个调节参数 U_ε ，将系统的调节目标设定为 $U_{ref} - U_\varepsilon$ 。这样系统在临界状态运行时，不会频繁地触发中断。U_ε 反映了系统在稳定状态运行时的负载波动，其值可以通过实验来确定。

算法 1 周期延展算法

```
int expanded ; //标志变量，表示是否存在被延展的进程
Expand()
{
  i = Get a task from adjust queue with the largest weight;
  //从权重最大的任务开始调整周期
  ΔU=U-Uref+Uε
  if(ΔU <=0)   //实际利用率低于 Uref-Uε，不需要调整
      return;
  while(i == True &&ΔU > 0)
  {
      Δui=Ci/Ti-Ci/TiMax   //Taski 可以减少的负载
      if(Δui > 0)
      {//Taski 的周期还没有达到最大
          if(Δui >ΔU)
          {//Taski 的周期不需要延展到最大，即可完成调整
//的目标。
              Ti=CiTi / (Ci-ΔUTi)   //Taski 的新周期
              ΔU=0; //调整目标已达到
              expanded++;}
          else
          {   Ti=TiMax
              ΔU=ΔU-Δui;
              expanded++;
               i = Get next task from adjust queue with the
smaller weight than i;}
      }}}
```

当系统低于 U_{ref} ，需要进行周期压缩。首先将富余的处理器利用率 ΔU 分配给权重最低的任务，若有剩余再分配给优先级次高的软实时任务。直到 ΔU 为 0，或所有的软实时任务均恢复最小周期为止。可以根据式(5)，计算 $Task_i$ 的新周期：

$$T_{iNew} = \frac{C_i T_i}{C_i + \Delta U T_i} \tag{5}$$

算法 2 周期压缩算法

```
int expanded ;
Compress()
{
    i = Get a task from adjust queue with lowest weight   //从权重
//最低的任务开始调整周期
    ΔU= Uref -U -Uε;
    if(ΔU < 0 )
        return;
    while(i == True &&ΔU >0)
    {
        Δui= Ci/TiMax-Ci/Ti ;//Taski 可以增加的负载
        if(Δui > 0)
        {//Taski 的当前周期不是最小周期
            if(Δui >ΔU)
            {   Ti=CiTi / (Ci+ΔU*Ti);
                ΔU =0;//调整目标已达到
```

```
            expanded--;}
        else
        {   Ti=TiMin;
            ΔU=ΔU-Δui;
            expanded--;
             i = Get next task from adjust queue with the
larger weight than i;}
    }}}
```

周期延展算法和压缩算法的时间复杂度均为 $O(n)$。由于采用了贪婪算法，只需执行 1 次周期延展算法或压缩算法便可找到最优解，使系统恢复稳定状态。如果所有任务的周期延展到最大仍不能满足要求，则基于任何规则的弹性调度算法都无法使系统回到稳态。此时可以考虑在调度模型中加入其它的控制方法，比如准入控制等。

综合以上 2 种算法，可以将弹性调度模型描述为：

$$U_{k+1} = U_k + \Delta U_k = U_k - \sum_{i=1}^{n} \frac{C_i \Delta T_i}{T_i (T_i + \Delta T_i)} \tag{6}$$

其中，$Task_i$ 的执行时间 C_i 和任务的数量 n 为随机变量，该系统可以看成状态变量 U_k 的一阶随机系统。

2.4 Lebesgue 采样方法在实时调度中的应用

在使用 Riemann 采样方法时，系统在每个采样周期中至少要进行 1 次中断处理和 2 次上下文切换。基于事件的 Lebesgue 采样方法可以避免这样的开销。文献[6]在理论上比较了一阶随机系统中，Riemann 采样和 Lebesgue 采样的性能差异。为了获得同样的控制精度，Riemann 采样的采样频率必须是 Lebesgue 采样的 4.7 倍。根据式(6)，基于弹性周期的实时调度系统可以看成一阶随机系统。因此，在本文的调度模型中，若使用 Lebesgue 采样方法，采样频率会有明显降低。

由于难以判断系统何时离开稳定状态，Lebesgue 采样方法在实时调度算法中一直没有得到广泛应用。看门狗是一种已被广泛使用的系统状态监测技术，其本质上是一个定时器。该定时器可以在预定的时间到达后给出一个硬件中断，通知系统有错误发生。当系统正常运行时，会周期性地刷新看门狗，刷新的周期稍小于定时器的周期，使看门狗永远不会到期。但当系统出错，比如系统过载，会因为不能及时刷新看门狗而导致定时器到期。因此，可以在看门狗的中断处理函数中调用周期延长算法，降低系统的负载。

在任务执行的间隙，操作系统会执行一个空任务以等待新任务的到来。比如，在 Linux 操作系统中，进程 0 在完成系统初始化后便转变成空任务，即 idle 进程。由于 idle 进程仅在系统空闲的时候执行，可以根据 idle 进程在一个时间段内的执行时间来判断 CPU 的负载情况。若要 CPU 的在参考时间段 T_{WG} 内的负载不超过 $U_{ref} = 0.9$，那么 idle 进程在时间段 T_{WG} 内的执行时间便不能低于 $T_{idle} = (1 - U_{ref}) \times T_{WG}$，否则就可以认为系统过载。图 2 说明了 idle 进程的执行情况与系统负载的关系。其中，T_{WG} 是时间轴上的一个参考时间段，标号①②③分别表示 idle 进程从 T_{WG} 开始，到执行时间等于 T_{idle} 的 3 个可能的位置。若 T_{idle} 在位置①处结束，说明从 T_{WG} 开始到当前时刻，系统的负载小于 U_{ref}。若 T_{idle} 在位置②处结束，说明系统在此 T_{WG} 内的负载等于 U_{ref}。若 T_{idle} 在位置③，即 T_{WG} 之外结束，则说明在当前的参考时间段 T_{WG} 内，系统的负载超过了 U_{ref}。那么在 idle 进程执行 T_{idle} 时间后刷新看门狗，便可以使看门狗在系统过载时触发一个中断。在情况①和情况②中，由于看门狗及时得到了刷新，系统会开始对新一轮 T_{WG} 的判断。在情况③下，因为在 T_{WG} 内没有刷新看门狗，看门狗会在 T_{WG} 结束时给出中断，此时意味着系统过载。所以，只有发生情况③时，才需要对系统进行采样并执行控制算法。

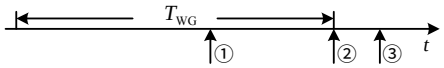

图 2　idle 进程的执行与系统负载的关系

当系统负载小于参考值 U_{ref}，即在情况①时，需要判断系统中是否存在延展进程，如果存在，则调用周期压缩算法。基于上文的讨论，本文在 idle 进程中添加了如下算法：

算法 3　idle 进程的执行算法

```
int begin = 1;
int expanded;
idle(void)
{
        preempt_disable();//禁止抢占
        if(begin == 1)
        {//开始一个新的参考时间段
            start = get_time(); //获取当前时间；
            begin = 0;}
        now = get_time();
        if( (now - start) >= Tidle)
        {//在当前的参考时间段内，idle 进程的累计执行时间已
//超过了 Tidle
            if(expanded) //存在被延展的进程
                compress();
            feed_dog(); //刷新看门狗
            begin = 1;}
        preempt_enable();//允许抢占
        default_idle(); //原有的 idle 进程
}
```

在上述算法中，T_{idle} 是 idle 进程在一个参考时间段 T_{WG} 中需执行的时间。该算法主要判断 idle 进程在参考时间段 T_{WG} 内的执行时间是否小于 T_{idle}。如 idle 进程的执行时间超过了 T_{idle}，则刷新看门狗的定时器，将定时器的计数值重置为 T_{WG}。

3　调度模型的实现及性能分析

3.1　调度模型在 RTAI 中的实现

本文在开源实时操作系统 RTAI 之上实现了 FES-L 调度模型。实验中使用的硬件平台是龙芯 2F 开发板，CPU 主频为 800 MHz，内存 256 MB。采用了龙芯中的 CP0-11 寄存器作为硬件看门狗。协处理器 CP0 的 11 号寄存器又称为 compare 寄存器，它用来在特定的时刻产生 1 个中断，该寄存器被写入 1 个初值后，便不断地将此值与计时器中的值进行比较，一但两者相等，便触发 63 号中断。笔者为 63 号中断编写了中断处理函数，并在此函数中调用了周期延展算法。

RTAI 在空闲时执行的是 Linux 的 idle 进程。本文按照算法 3 的思想改写了 idle 进程。为了尽量减小 Linux 用户进程的干扰，对 Linux 的文件系统进行了裁减，裁减后的 Linux 中仅存在 idle、shell 等进程，这些进程执行开销可忽略不计。

从控制理论的观点看，对反馈调度的评价包括实时系统的暂态性能和稳态性能。暂态性能主要是指系统从异常状态收敛到正常状态的时间。在本文中，异常状态为系统负载超过了参考值 U_{ref}，正常状态是指系统负载低于 U_{ref}。稳态性能

主要是指系统的稳态误差，即系统在正常运行的情况下，系统的真实负载与参考值之间的偏差。

3.2 暂态性能测试

实验创建了有 10 个进程的任务集，任务执行时间的变化范围通过统计得到。系统正常运行时，负载在 84%~86.76%之间变化。由于各任务仅执行一些简单的数值计算工作，因此负载的变化不大。本文将参数 U_ε 确定为 1.38。各任务的权重从 1~10，参考时间段设为 4 ms。

为了验证系统的暂态性能，本文在系统正常运行的时候，加入一个实时进程作为扰动，使系统利用率突然增加，促使看门狗到期并执行控制函数调整各任务的周期。扰动进程的周期固定为 1 ms，执行时间约为 217 ms。截取的部分实验记录如图 3 所示。

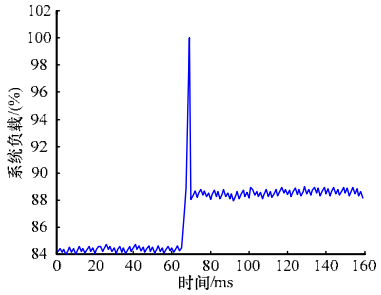

图 3　周期延展时的系统动态性能测试

在 69 ms 时，系统因为新增加一个实时进程，导致负载突然增加到 100%。此时系统中的任务集不再是可调度的。若不及时调整系统状态，将会有大量的任务超过其截止期。

当系统持续过载超过一个参考时间段时，看门狗因定时器到期而触发 63 号中断并使周期调整算法得以执行。周期调整算法根据当前的系统状态按照权重延展了各任务的周期，从而将系统负载降低到了 90%以下，并维持在 88%附近。周期调整以后，权重为 8, 9, 10 的 3 个任务均以最大周期运行，权重为 7 的任务，周期被调整为 3.415 ms。

在上述状态下，本文将扰动进程删除，系统负载陡然下降到 67.83%。随后，在 idle 进程执行的时候调用周期压缩算法，恢复了权重为 7, 8, 9, 10 的进程的周期。系统负载维持在 84%附近。上述实验说明，在系统过载时，看门狗能够及时地给出中断，并且验证了周期调整算法的有效性和快速响应的能力。周期压缩时的系统动态性能测试如图 4 所示。

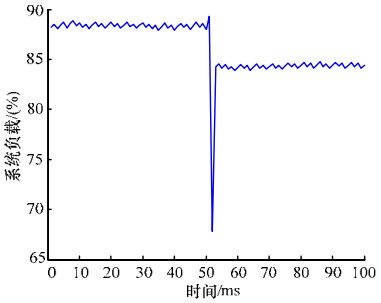

图 4　周期压缩时的系统动态性能测试

3.3 2 种采样方法的稳态误差与采样频率的比较

为了测试系统的稳态误差，本文构造了一个负载在 U_{ref} 附近随机变化的任务集。任务集中共 10 个任务，每个任务的功能都是执行一个简单的整数运算，运算的次数为一个均匀分布的随机序列。同时将每个任务的周期变化范围设为[1 ms, 2 ms]。通过调整随机序列的分布范围，将系统负载的变化范围限制在 90%~93%之间。在实验中记录了 Lebesgue 采样和 Riemann 采样方式下的系统负载的变化，以及采样的次数。从测试结果中截取的 1 000 组数据如图 5 所示，反映了 Lebesgue 采样方式下的系统负载变化。

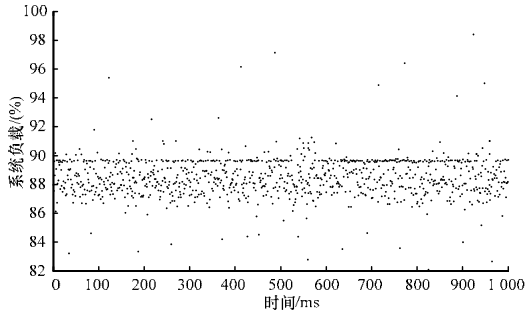

图 5　Lebesgue 采样方式下的系统负载变化

可以看到，系统负载大部分控制在 90%以下，平均系统负载为 88.4%，平均稳态误差为 1.6%。在持续 1 h 的测试中，系统共执行了 495 711 次采样，平均采样频率为 137.7 Hz。

为了在 Riemann 采样中获得同样的稳态误差，本文将采样周期设为 2 ms，采样频率为 500 Hz，是 Lebesgue 采样的 3.63 倍。2 种采样方式的稳态误差比较如表 1 所示。

表 1　Lebesgue 采样和 Riemann 采样的稳态误差比较

方法	稳态误差/(%)			采样频率/Hz	被调整任务数
	最大值	最小值	平均值		
Lebesgue	95	83.197 5	88.413 2	137.7	8.75×10^6
Riemann	95	84.257 3	87.536 9	500.0	5.81×10^7

4　结束语

本文提出基于事件的动态反馈调度模型 FES-L，以可变负载的软实时系统为控制对象，在保证系统实时性的同时，减小了系统开销。本文在 RTAI 实时操作系统的基础上实现了 FES-L 调度模型，并测试了系统的动态性能和稳态性能，比较了相同稳态误差下 Lebesgue 采样与 Riemann 采样的采样频率。由实验结果可知，本文的调度模型具有较好的暂态性能和稳态性能，并显著降低了系统开销，适合在资源较紧张的软实时系统中使用。

参考文献

[1] Stankovic J A, Lu Chenyang, Son S H, et al. The Case for Feedback Control Real-time Scheduling[C]//Proc. of the 11th Euromicro Conference. [S. l.]: IEEE Computer Society, 1999.

[2] Buttazzo G, Abeni L. Adaptive Workload Management Through Elasitc Scheduling[J]. Real-Time Systems, 2002, 23(1): 7-24.

[3] Buttazzo G, Abeni L. Elastic Scheduling for Flexible Workload Management[J]. IEEE Trans. on Computers, 2002, 51(3): 289-302.

[4] Chantem T, Hu Xiaobo, Lemmon M D. Generalized Elastic Scheduling[C]//Proc. of the 27th IEEE International Real-time Systems Symposium. [S. l.]: IEEE Computer Society, 2006.

[5] Buttazzo G, Velasco M, Marti P et al. Managing Quality-of-control Performance Under Overload Conditions[C]//Proc. of the 16th Euromicro Conference on Real-time Systems. Catania, Sicily, Italy: IEEE Computer Society, 2004.

[6] Årzén K J, Bernhardsson B M. Comparison of Riemann and Lebesgue Sampling for First Order Stochastic Systems[C]//Proc. of the 41st IEEE Conference on Decision and Control. Las Vegas, USA: IEEE Computer Society, 2002.

编辑　顾姣健