基于语义模型的实时数据有效性保证策略研究

汤小春 田凯飞

(西北工业大学计算机学院 西安 710072)

摘 要 实时数据的有效性与CPU处理能力是信息物理融合系统(CPS)中的一对矛盾,提高采样频率可以保证实时数据的有效性,但是会增加CPU负荷,降低系统的计算能力。首先,利用实时数据的语义特点建立数据的有效性模型;然后,通过在CPU空闲期间设置预调度任务,合理地利用数据有效性模型,设置新的有效性间隔和实时数据的更新事务的开始时间,减少CPU执行时间;最后,在棉花采摘锭的自转、公转及油压等参数上,对基于语义模型的实时数据有效性保证策略进行了系统的评价,结果表明所提方法能够减少15%左右的CPU负荷。

关键词 实时数据,语义模型,有效性保证,信息物理融合系统

中图法分类号 TP301.6 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.12.002

Validity Protection Strategy for Real Time Data in CPS Based on Semantics

TANG Xiao-chun TIAN Kai-fei

(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract The validity of real time data and CPU processing capability is a contradiction in the cyber physical system (CPS). While increasing the sampling frequency can guarantee the validity of the real time data, it can also increase the CPU workloads and reduce the system's computing power. Firstly, this paper used the semantic features of real time data to establish the validity model of the data. Then, by setting the pre-scheduling task during the idle period of the CPU, making good using of the data validity model, setting the new validity interval and the start time of the update transaction of the real time data, the CPU execution time was reduced. Finally, the strategy based on semantic model was evaluated systematically on the parameters such as rotation, revolution and oil pressure of cotton picking ingot. The CPU load can be reduced by about 15%.

Keywords Real time data, Semantics model, Validity protection, Cyber physical system

1 引言

信息物理融合系统(CPS)[1-3]是近年来新兴的热点研究课题,它是一个由嵌入式传感器以及执行器组成的大规模分布式网络,传感器与执行器分别监视环境和控制环境[4];CPS被用在许多应用环境中,这些环境更加强调对数据处理的高实时性[5]。比如,工业控制过程中的传感器测量数据、空中交通管制中的飞机位置以及发动机控制中的压力和温度等都属于实时数据,它们被加工处理后存储在实时数据库中,用来描述物理实体的状态。与事务数据库中的数据不同,实时数据具有较强的时效性,即被采样的数据值只有在一个规定的期间内才有效[6-8]。时间有效性的概念在文献[6]中被首次提出,它被用来描述一个数据的正确性,如果一个实时数据对象的值能够正确地反映物理世界中实体的当前状态,那么就可以称这个实时数据是时间有效的。如果新的数据值是在前一个数据值的有效期结束之前被更新到数据库中,那么这个实时数据值是有效的,即与物理世界的实体状态一致。

通常,外部设备的状态更新事务具有定长的周期和截止期限[6,9-10],但是CPU的处理能力以及网络通信的延迟往往使得采集到的数据失去有效性。为了确保新数据在前一个状态失效前被更新,必须解决两个问题[5,10]:1)确定下一次更新事务的采样周期以及截止时间;2)更新事务的调度策略。目前,主要采用3种方法来降低更新事务的负载以满足数据的有效性。文献[6,9]提出了一种非常简单的保持数据有效性的方法,即Half-Half模式,其任务周期固定,实时数据的更新间隔为数据有效期限的一半。为了提高CPU的利用率,文献[10]对Half-Half模式进行优化,提出了More-Less模式,采用最迟响应时间来代替任务的相对截止期限。如果最迟响应时间小于数据有效期的一半,那么就利用数据有效期与最迟响应时间的差值来决定实时数据的更新周期。该方法由于扩大了数据的更新周期,减少了数据更新的频率,降低了CPU的利用率,因此提高了更新数据事务的可调度性。Half-Half以及More-Less模式都采用固定周期的任务更新,使用悲观的策略来设置采样任务的截止期以及采样周期。文献

到稿日期:2016-10-18 返修日期:2017-04-01 本文受中国科技部国家重点研发计划(2016YFB1000700)资助。

汤小春(1969-),男,副教授,主要研究方向为实时数据管理、高性能计算,E-mail:tangxc@nwpu.edu.cn;**田凯飞**(1993-),男,硕士生,主要研究方向为大数据计算,E-mail:tiankaifei0108@mail.nwpu.edu.cn。

[5]提出了一种非固定周期的 DS-FP 模式(即延迟启动的算法)来进一步降低 CPU 的负荷。DS-FP 模式使用一种随机发起更新事务的模式,在不破坏数据有效期的前提下,尽量推后更新事务的执行开始时间。DS-FP 的推后策略使得更新事务的周期比 Half-Half 以及 More-Less 更大,因此可以进一步降低 CPU 的负荷。虽然该策略降低了采样频率,但是由于它尽量推迟更新事务的执行开始时间,可能导致在数据有效期达到之前大量更新事务请求执行,即瞬时负荷过载,而且 DS-FP 采用更新事务优先级固定的方式,因此可能会导致部分更新事务无法在数据的有效期截止之前完成。为了解决这个问题,文献[11]提出了一种 DS-EXC 的改进方法,在大量更新事务同时到达后,通过交换更新事务次序的方法,使得部分更新事务提前开始,从而错开高峰计算时间段,解决了更新事务不能满足时间限制的问题。但是 DS-EXC 算法的交换过程比较复杂,有时可能会导致不可调度问题发生。文献[12]分析了传统的抢占式与非抢占式的优缺点,提出了预留的策略来解决实时调度问题,它对于不能使用预留资源的低优先级别任务的影响较大。文献[13]针对任务的不确定性,重新变更时间,从而保证各个任务都能够被调度。这种方法容易引起系统的波动。

上述研究方法的基本原则是在保持数据有效性的前提下,尽量减少 CPU 的负荷,以调度更多的更新事务。因此,其基本思想是在保证有效性的前提下,尽可能扩大采样的周期,以减少 CPU 计算负载。但是,在数据的有效性期间固定且也确定更新事务的计算时间的情况下,减少 CPU 负荷的能力是有限度的。综合以上的分析,本文提出了一种新的实时数据更新策略来保证实时数据的有效性。针对物理世界中实体连续性的特点,分析数据的语义特征,获得数据自身的变化规律,即数据的语义模型。在数据更新事务到达之前,利用 CPU 的空闲状态,依据数据的语义模型对数据进行预测。如果数据的预测值与采集的数据值在一个规定的误差范围内,那么就不执行对应的数据更新事务;如果预测值与实际采集的数据值之间的误差超过规定的范围,那么就执行数据更新事务,同时保存新的数据模型。预测任务通常在 CPU 空闲的时间启动,从而避免了 CPU 更新事务的瞬时负载过载问题,防止在某个时间点出现大量的更新事务;另外,通过预测减少了数据更新的次数,降低了 CPU 的总体负荷。当然,对于 CPU 负荷一直处于 100%状态而不可能存在空闲的极端情况,所提算法与原算法的实际效果基本相同;对于 CPU 的负荷达到 90%以上而存在忙闲不均的情况,采用本文中的调度可以有效减少不可调度的任务数量;对于复杂数据语义模型导致预测时间大于数据实际计算时间的情况,只要 CPU 有空闲时间,那么本文的策略就能够抑制由于截止时间达到导致的任务冲突现象。

本文第 2 节主要介绍语义模型以及有效性的扩展算法;第 3 节介绍基于语义模型的调度算法,并分析算法有效的必要条件;第 4 节给出一个 MRO 系统的验证结论。

2 基本概念及模型

本节主要介绍数据有效性的概念以及数据语义的概念。在不考虑数据从传感器采集再被传送到计算节点之间的传输时间的情况下,假设数据更新事务的开始时间就是传感器采集数据的开始时间,因此数据的有效期以采集数据的开始时间来计算。

2.1 实时数据管理模型

实时数据管理的核心概念在于维护数据的有效性,使得数据值能够反映物理世界实体的当前状态。实时数据管理的更新事务从外部设备读入传感器数据,经过计算后将数据保存到实时数据库中。周期性控制任务则向设备发送一些控制请求,控制设备的状态变化。

定义 1 能被系统调度和执行的单元叫事务。传感器数据更新事务 $\tau=(e,V,s,a,pr)$,其中 e 表示更新事务最坏情况下的执行时间,V 表示被更新数据的有效间隔,s 表示数据 d 的更新事务 J 的执行开始时间,a 表示数据 d 的更新事务 J 的绝对期限,pr 表示数据 d 的更新事务的优先级别。一个具体的数据 d_i 的采集和计算过程称为 d_i 的更新事务,用 τ_i 来表示。

定义 2 对于一个周期性采集的传感器数据 d_i,更新事务 τ_i 在某个采样周期 j 中的一个数据更新过程称为任务,用 $J_{i,j}$ 表示。更新事务 $\tau_i=\{J_{i,1},J_{i,2},\cdots,J_{i,n}\}$,其中 n 的取值可以非常大,表示数据监控的时间特别长。

由于一个系统中包含很多个传感器数据,因此更新事务是由不同的传感器数据更新事务组成的集合,即 $\tau=\{\tau_1,\tau_2,\cdots,\tau_m\}$,其中 m 表示传感器数据的个数。

实时数据随时间不停地变化,实时数据 d_i 的正确性依赖于对象的实时状态和当前的采样值。

定义 3 如果一个数据 d_i 的第 j 次更新完成是时刻 t 之前的最后一次更新,并且采样时刻 $s_{i,j}$ 与数据的有效期限 v_i 之和不小于时刻 t,即 $s_{i,j}+v_i\geqslant t$,那么就认为数据 d_i 是有效的。

在时刻 t 采样获得实时数据 d_i 的值,它的有效期限一直保持到时刻 $t+v_i$。超过这个时刻后就认为数据 d_i 不再有效或者为过期数据。

命题 1 当数据 d_i 的第 $j+1$ 次更新完成时间小于第 j 次采样数据的有效期时,数据 d_i 总是有效的。

证明:如图 1 所示,数据 d_i 包含两次更新事务 $J_{i,j}$ 和 $J_{i,j+1}$,数据的有效期为 V_i。矩形框表示更新事务从 $J_{i,j}.s$ 开始到 $J_{i,j}.a$ 结束。带纹路的矩形框表示更新事务从 $J_{i,j+1}.s$ 开始到 $J_{i,j+1}.a$ 结束。

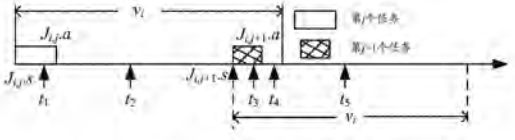

图 1 数据有效性模型

在时刻 t_2 访问数据时,因为 $J_{i,j}.a$ 已经结束,所以访问的数据 d_i 为 $J_{i,j}$ 的值;因为 $J_{i,j}.s+v_i\geqslant t_2$,所以数据是有效的。在时刻 t_3 访问数据时,因为 $J_{i,j+1}.a$ 还没有结束,所以访问的数据 d_i 为 $J_{i,j}$ 的值,因为 $J_{i,j}.s+v_i\geqslant t3$,所以数据是有效的。在时刻 t_4 访问数据时,因为 $J_{i,j+1}.a$ 已经结束,所以访

问的数据 d_i 为 $J_{i,j+1}$ 的值；因为 $J_{i,j+1}$，$s+v_i \geqslant t_4$，所以数据是有效的。在 t_5 时刻访问数据的情况与 t_4 类似，是有效的。

在 t_1 时刻访问数据时，使用的是更新事务 $J_{i,j}$ 前一次的数据，所以它也是有效的。

通常来说，实时数据 d_i 的有效期由用户根据物理实体的特性来指定。实时数据管理的一个重要设计指标是保证数据的有效性，即它们总是有效的。对过期数据的访问可能会影响系统的功能，产生一些错误的事件或者不能及时响应的错误。

2.2 实时数据的语义模型

实时数据的数据模型是其语义特征。每一个传感器数据可以采用如下的状态变化方程来描述。

$$x_{i+1}=x_i+\int_0^{\Delta t} h'(t)\mathrm{d}t \tag{1}$$

其中，$h'(t)$ 表示状态变化特征，是时间 t 的函数。对于物理世界，例如温度、压力等传感器数据，x 代表在时刻 k 的数值，而 $\mathrm{d}x/\mathrm{d}t$ 为在时刻 t 时偏离时刻 k 时的大小。当然也可以采用 $\mathrm{d}^2x/\mathrm{d}t^2$，这样就可以得到变化率，模型的精度会更高，但是需要更多的变量，计算也相对复杂，导致 CPU 代价更高，因此使用 $\mathrm{d}x/\mathrm{d}t$ 模型。我们认为物理世界的一个简单过程（例如温度、压力等）可以通过两个参数来描述。

2.3 实时数据有效性的扩展

实时数据的有效性语义是指数据的值能够反映物理实体当前的状态，而信息世界的数据值是离散的，如果信息世界中数据的值在一定范围内波动，那么可以认为物理世界的实体状态没有发生变化，此时实时数据的有效期间会被扩大，即实时数据的有效期限可以通过一个精度边界值 $\pm\delta$ 来推导。

算法 1　实时数据有效期限计算方法

输入：δ_i 为实时数据 d_i 的误差边界值；x_i 为实时数据 d_i 在 t 时刻的采样值；v_i 为当前时间的有效期限

输出：实时数据的有效期 v_i'

step1　在 t_1 时刻得到数据 d_i 的值 $\hat{x}_i$

step2　if $\mathrm{abs}(x_i-\hat{x}_i)<\delta_i*(t_1-t)$ then

$$v_i'=v_i+v_i*\frac{\delta_i*(t_1-t)}{\alpha_i+|x_i-\hat{x}_i|}$$

else $v_i'=v_i$

算法 1 中的 α_i 是一个权重系数，可以根据应用程序特征设置，通常设置 $\alpha_i=\delta_i$ 以确保有效期限不超过实时数据的两个指定的有效期限。

3 实时更新事务的调度算法

3.1 调度模型

设系统有 k 个数据 $d_1, d_2, \cdots, d_k$，每个实时数据对应一个更新事务，系统的更新事务为 $\tau=\{\tau_1,\tau_2,\cdots,\tau_k\}$，$\tau_i=\{J_{i,1}, J_{i,2},\cdots,J_{i,n}\}$，其中 $1\leqslant i\leqslant k$。实时调度的目的是给出一个执行序列，使得 k 个数据能够在其截止期内执行，并且尽量提高 CPU 的利用率。

区别于传统的单调速率调度、最早截止时间优先算法、最短空闲时间优先算法等调度方式，本文利用数据的语义特性，在 CPU 负荷较低的情况下插入数据的预测任务，从而增加实时数据的有效期间，降低高峰时间段实时任务的数量，保证实时数据能够在限制时间内完成。

如图 2 所示，两个数据 d_i 和 d_l 的更新事务分别是 $J_{i,j}$ 以及 $J_{i,j+1}$。$a_{i,j}$ 是任务 $J_{i,j}$ 的绝对截止时限，$S_{i,j}$ 是任务 $J_{i,j}$ 的开始时间，$a_{i,j+1}$ 是任务 $J_{i,j+1}$ 的绝对截止时限，$S_{i,j+1}$ 是任务 $J_{i,j+1}$ 的开始时间，$P_{i,j+1}$ 是 $J_{i,j+1}$ 的预测任务，$J_{l,j}$ 是比 $J_{i,j+1}$ 的优先级别更高的更新事务。

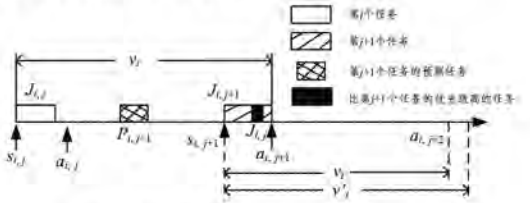

图 2　基于语义的实时更新事务调度模型

调度过程中，当 CPU 处于空闲状态时，在 $P_{i,j+1}$，t 执行第 $j+1$ 个任务的预测任务 $P_{i,j+1}$，取得当前的真实状态值 $x_{i,j+1}$，根据 $J_{i,j}$ 的采样时间 $J_{i,j}$，t 采样数据值 $x_{i,j}$ 以及数据 d_i 的语义模型。通过计算得到一个预测值 $\hat{x}_{i,j+1}$，如果 $|x_{i,j+1}-\hat{x}_{i,j+1}|\leqslant\delta$，那么更新事务 $J_{i,j+1}$ 就不用执行，即从队列中删除更新事务 $J_{i,j+1}$，同时根据算法 1 重新计算新的实时数据的有效期限 v_i'，该有效期限比 v_i 的值更大；相反，如果 $|x_{i,j+1}-\hat{x}_{i,j+1}|>\delta$，那么就必须执行更新事务 $J_{i,j+1}$，同时将新的数据模式保存，此时该实时数据的有效期限恢复为 v_i。第 $j+1$ 个任务的预测任务必须在 $J_{i,j}$ 执行后且 $J_{i,j+1}$ 执行之前的时间段内被启动执行。如果在此期间 CPU 负荷过重，没有空闲的 CPU 来调度 $P_{i,j+1}$ 任务，那么在执行 $J_{i,j+1}$ 时将 $P_{i,j+1}$ 从队列中删除。

3.2 调度算法

假设更新事务中包含 k 个任务，每个更新事务按照数据的有效期限设置一个优先级，即 $pr=\{\tau_1,\tau_2,\cdots,\tau_k \mid v_1\leqslant v_2\leqslant\cdots\leqslant v_k\}$。

假设在时间区间 $[start, stop)$ 中比更新事务 $J_{i,l}$ 优先级高的作业为 $J_{j,l}$，其中 $1\leqslant j\leqslant n$，那么更新事务 i 的最早执行时间为 $J_{i,l}$ 的最后截止时间减去 $J_{i,l}$ 的执行时间和其他高优先级任务在 $[start, stop)$ 区间占用的时间之和。

$$J_{i,l}=J_{i,l}.a-J_{i,l}.e-\sum_{j=1}^{n} et_j(start,stop) \tag{2}$$

其中，n 表示比任务 i 的优先级更高的其他事务的更新事务，$et_j(start,stop)$ 表示任务 $J_{j,l}$ 在 $[start, stop)$ 区间使用 CPU 的时间。

算法 2 给出了基于语义的调度算法。wq 是更新事务队列，pq 是预测任务队列。算法首先根据每个任务的有效期限设置其优先级别；然后建立任务的初始队列；最后调用 $TASK\_exec(wq, pq)$ 进行调度执行。

算法 2　主调度算法

输入：更新事务的个数 m，更新事务集合为 $\tau=\{\tau_i \mid 1\leqslant i\leqslant m\}$；每个更新事务中更新事务的执行时间为 $E=\{e_i \mid 1\leqslant i\leqslant m\}$；每个更新事务中更新事务的有效期限为 $v=\{v_i \mid 1\leqslant i\leqslant m\}$

输出：可行的调度序列或者拒绝

1. 初始化 t=0；

```
2. 创建一个优先队列 wq 以及 pq;
3. while(i++<=m)
4.   按照 J_{i,0}.v 的值设置 J_{i,0}的优先级;
5. for i=1 to m do //更新事务 i 的第 0 个更新事务开始时间为 0
6.   J_{i,0}.s=0; //更新事务 i 的第 0 个更新事务完成时间为 0
7.   J_{i,0}.a=e_i;
8.   根据式(2)计算 J_{i,0}.a,J_{i,0}.s;
9.   if(J_{i,0}.a>v_i-e_i) then
10.     拒绝调度;返回。
11.  else
12.    执行 wq.insert(J_{i,0});
13.    end for
14. endwhile
15. TASK_exec(wq,pq);
16. return;
```

算法 3 中,对于更新事务队列 wq 中的更新事务,一旦开始时刻点达到则立即启动执行,如果 $J_{i,j}$ 对应的 $PJ_{i,j}$ 还在队列中,需要将 $PJ_{i,j}$ 删除。若 CPU 有空闲时间,则取得一个预测任务 $PJ_{i,j}$,计算采样值与预测值之间的误差,如果误差小于 δ,那么就不执行 $J_{i,j}$ 的计算以及向数据库的更新,直接从第 15 行开始执行;如果误差大于 δ,那么执行 $J_{i,j}$。$J_{i,j}$ 被执行后,设置下一次调度任务的开始时间、结束时间以及截止时间,调用函数 *Sched* 将新的任务插入到更新事务队列中,同时生成下一次的预测任务,并插入到 pq 队列中。第 15—23 行实现了更新事务完成后需要做的工作。计算更新事务 τ_i 的有效时限,生成一个更新事务 τ_i 的新计算任务。新的更新事务可能与其他任务冲突,因此调用 *Sched* 函数重新设置更新事务的运行参数,从而安排合理的 CPU 时间。

算法 3 调度执行 *TASK_exec*(wq, pq)

```
输入:更新事务队列 wq,预测任务队列 pq
输出:结束或者异常退出
1. while(1) do
2.   if(wq 中有任务到达执行时间)
3.     执行第 13 行;
4.   else if(取出 pq 的第一个预测任务 p_{i,j})
5.     执行第 8 行;
6.   else
7.     continue;
8.   if J_{i,j-1}.value-PJ_{i,j}.pvalue<δ then
9.     根据算法 1 计算出新的 v_i;
10.    从 wq 队列中删除任务 J_{i,j};
11.    J_{i,j}.flag=complete goto 第 15 行
12.  endif
13.  启动 J_{i,j}的执行; J_{i,j}.flag=complete;
14.  若预测任务 PJ_{i,j}在队列 pq 中,则从 pq 中删除 PJ_{i,j};
15.  if (J_{i,j}.flag=complete) then
16.    生成一个新的作业 J_{i,j+1}及其预测任务 PJ_{i,j+1}
17.    J_{i,j+1}.a=J_{i,j}.s+v_i
18.    J_{i,j+1}.f=J_{i,j}.s+v_i
19.    J_{i,j+1}.s=J_{i,j}.a-e_i
20.    if(Sched(J_{i,j+1},wq)!=abort) then
21.      根据 J_{i,j+1}的开始时间设置预测任务 PJ_{i,j+1}的优先级,并插
         入队列 pq;
22.    endif
23.  endif
24. endwhile
25. return
```

算法 4 中的第 3—9 行的功能是,根据 $J_{i,j}$ 的优先级别和执行区间找到比 $J_{i,j}$ 优先级别高并且在其执行区间中占用 CPU 的任务。第 11 行调用函数 CountPreempt 来计算比 $J_{i,j}$ 的优先级更高的任务占用 $J_{i,j}$ 的 CPU 时间。第 13—20 行通过不断更改任务 $J_{i,j}$ 的开始时间,来避免与高优先级任务的冲突,若可以调度,就插入任务到 wq 队列,否则返回 abort。第 22—24 行,对于优先级比 $J_{i,j}$ 低的更新事务,由于 $J_{i,j}$ 的调度可能破坏了其调度执行时间,产生了冲突,因此采用递归的方式重新计算它们的调度参数。

算法 4 Sched($J_{i,j}$, wq) //将 $J_{i,j}$ 调度后插入到 wq 队列。

```
输入:J_{i,j},wq
输出:重新计算更新事务的开始时间和结束时间的等待队列 wq
1. 创建并初始化队列 temp=wq;
2. PTask=null;LTask=null;
3. while(J_{k,l}=pop(temp)==NULL){
4.   if((J_{k,l}.s∈[J_{i,j}.s,J_{i,j}.a))||J_{k,l}.a∈[J_{i,j}.s,J_{i,j}.a))
5.   if(J_{k,l}.pr>J_{i,j}.pr)
6.     PTask.Add(J_{k,l});
7.   else
8.     LTask.Add(J_{k,l});
9. }
10. hTime ← 0; oldTime ← 0;
11. hTime←CountPreempt(PTask,J_{i,j}.s,J_{i,j}.a);
12. while (hTime > oldTime) do
13. {// 与当前任务冲突的 CPU 时间}
14.   J_{i,j}.s ←J_{i,j}.a-hTime-J_{i,j}.e;
15.   if (J_{i,j}.s < J_{i,j-1}.s) then
16.     abort; // 更新事务不能调度,失败
17.   end if
18.   oldTime ← hTime;
19.   hTime ←CountPreempt(PTask,J_{i,j}.s,J_{i,j}.a);
20. end while
21. wq.insert(J_{i,j});
22. while((J_{k,l}=LTask[i])!=NULL) do
23.     Sched(J_{k,l},wq);
24. endwhile
```

算法 5 CountPreempt(*Task*[], t_1, t_2) //计算 t_1 到 t_2 之间已经使用的 CPU 时间

```
输入:任务集合,时间区间[t_1,t_2)
输出:Task 中任务在 t_1 到 t_2 之间占用 CPU 的时间
1. while((J_{k,l}=Task[i])!=NULL) do
2.   start=J_{k,l}.s>t_1?J_{k,l}.s:t_1;
3.   stop=J_{k,l}.a≤t_2?J_{k,l}.s:t_2;
4.   result+=stop-start;
5. endwhile
6. return result;
```

3.3 算法分析

数据的预测任务只能在没有数据更新事务执行时启动,即 CPU 有空闲时才能执行。因此,也存在这样的情况:CPU

负荷过高，没有时间执行预测任务，只能让更新事务在启动时刻达到后执行，同时抛弃对应的预测任务。在算法调度中，更新事务响应时间是不确定的，其最大的响应时间为 RT_i，而且它随着任务的开始时间以及任务完成时间而变化，但是任意一个更新事务 τ_i 的两次连续调度的任务之间存在以下关系：

$$v_i - e_i \geqslant s_{i,j} - s_{i,j-1} \geqslant v_i - RT_i \tag{3}$$

从式(3)可以看出，连续两个任务之间的最大间隔期间不超过 $v_i - e_i$。在只有一个更新事务的情况下，系统的 CPU 占用率为 $\frac{e_i}{v_i - e_i}$。第一种极端情况是如果 $v_i = 2e_i$，那么 CPU 的利用率将达到 100%，此时由于系统 CPU 满负荷工作，因此预测任务不能得到 CPU 的执行时间，而且由于系统在任务的开始时刻没有检查到预测任务的执行，更新事务自己执行，因此在这种情况下本文调度算法将没有意义。另一种极端情况是预测任务得到执行，执行时间为 PE_i，但是由于误差太大，更新事务也需要执行，因此 CPU 的占用率就变为 $\frac{PE_i + e_i}{v_i - e_i}$，此时 CPU 的负荷加大，但是调度效果与前面一样。比较好的结果是 PE_i 的执行时间相对于 e_i 小得多，并且预测结果与采集的数据误差在指定范围内，那么 CPU 的利用时间就变为 $\frac{PE_i}{v_i + \varepsilon - e_i}$，其中 ε 是因误差小而扩展的数据有效期限，在这种情况下，CPU 的占用率大大减小。

结论 1　在不考虑数据有效期扩展的情况下，如果更新事务的计算时间是预测任务计算时间的 x 倍，那么在 n 次连续计算中出现误差小于规定值的次数只需要 n/x 次，调度算法就有效果。

证明：假设存在 m 个更新事务，那么执行一次的 CPU 占用率为 $\sum_{i=1}^{m} \frac{e_i}{v_i - e_i}$，连续执行 n 个期间时 CPU 占用率为 $n * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i}$。

如果采用预测任务的方法，在 n 次连续执行过程中平均有 n_1 个更新事务的预测结果与误差超过指定的范围，平均有 n_2 个更新事务的预测结果与误差没有超过指定的范围，那么 CPU 的占用率为：

$$n_1 * \sum_{i=1}^{m} \frac{PE_i + e_i}{v_i - e_i} + n_2 * \sum_{i=1}^{m} \frac{PE_i}{v_i - e_i} \tag{4}$$

如果预测任务的执行时间是更新事务的执行时间的 $1/x$，那么式(4)就变为：

$$\begin{aligned} & n_1 * \sum_{i=1}^{m} \frac{PE_i + e_i}{v_i - e_i} + n_2 * \sum_{i=1}^{m} \frac{PE_i}{v_i - e_i} \\ =& n * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i} - n_2 * \sum_{i=1}^{m} \frac{xPE_i}{v_i - e_i} + n * \sum_{i=1}^{m} \frac{PE_i}{v_i - e_i} \\ =& n * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i} + (n - xn_2) * \sum_{i=1}^{m} \frac{PE_i}{v_i - e_i} \end{aligned} \tag{5}$$

当 $n - xn_2 < 0$ 时，式(5)比 $n * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i}$ 小，从而可以证明调度算法有效，即 $n_2 > n/x$，算法具有有效性。x 越大，说明预测任务的计算时间越小，此时使得算法有效的误差在范围内的次数也越少，当 $n_2 > n/x$ 时，算法一定能够减少 CPU 的负荷。

结论 2　在数据有效期 ε 扩展的情况下，如果更新事务的计算时间是预测任务的计算时间的 x 倍，那么在 n 次连续计算中出现误差小于规定值的次数仅为 n/x，调度算法对 CPU 负荷的减少量与 ε 成正比。

证明：针对式(4)考虑如下的计算式子：

$$\begin{aligned} & n_1 * \sum_{i=1}^{m} \frac{PE_i + e_i}{v_i - e_i} + n_2 * \sum_{i=1}^{m} \frac{PE_i}{v_i + \varepsilon - e_i} \\ =& n_1 * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i} + n_1 * \sum_{i=1}^{m} \frac{PE_i}{v_i - e_i} - n_2 * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i} + \\ & n_2 * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i} + n_2 * \sum_{i=1}^{m} \frac{PE_i}{v_i + \varepsilon - e_i} \\ =& n * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i} + n_1 * \sum_{i=1}^{m} \frac{PE_i}{v_i - e_i} - n_2 * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i} + \\ & n_2 * \sum_{i=1}^{m} \frac{PE_i}{v_i + \varepsilon - e_i} \\ =& n * \sum_{i=1}^{m} \frac{e_i}{v_i - e_i} + (v_i - E_i)((n - xn_2) + \varepsilon * (n_1 - xn_2)) * \\ & \sum_{i=1}^{m} \frac{PE_i}{(v_i - e_i) * (v_i + \varepsilon - e_i)} \end{aligned}$$

当 $(n - xn_2)$ 为负数时，$(n_1 - xn_2)$ 一定也为负数，因为 n_1 比 n 小。因此，在考虑数据有效期 ε 扩展的情况下，CPU 的负荷还会进一步减少。

结论 3　算法 sched 是收敛的。

证明：假设任意一个事务更新事务结束，因为是周期性任务，所以算法 3 需要产生一个新的更新事务，并且设置其开始执行时间、结束时间以及截止时间。在算法 3 第 3－9 行，首先找到优先级比自己高的更新事务，然后在第 12－20 行通过改变任务的开始时间来调整干涉，开始时间一致调整为上一个任务的结束时间。然后检查当前任务是否与低优先级的任务冲突，在冲突的情况下调整冲突的低优先级任务的参数。假设低优先级任务数是 m，最坏情况下循环 $m * m$ 次，要么为当前任务找到一个合适的执行时间，要么当前任务无法被调度，此时直接结束。因为任务的数量有限，所以调度算法 sched 一定可以收敛。

4　算法评价

4.1　实验环境以及参数

在对高端复杂收获机械 MRO 系统的设计应用中，针对棉花采摘机中采摘锭中的采摘头、风机、风管、棉箱、发动机、液压油管、变速箱以及梳棉球等部件，采用智能嵌入技术、信息融合与处理等核心技术手段获取 100 多个关键核心部件作业运行参数，进行相关工作参数的获取和诊断。采用本文方法对一台设备到多台设备同时进行处理的模式进行测试，计算调度率以及效用两个主要的评价指标，通过对比 DS-FP 算法得出两种算法在调度率与效用指标上的差距；另外，针对 δ 以及预测任务执行时间的不同，测试了不同情况下调度率与效用指标的变化。

调度率指一个调度集合的更新事务中可以被调度的更新事务。如果系统中事务 τ 的个数为 m，而可以被调度的更新事务个数为 n，那么调度率 SR 就等于 n/m。而效用的计算方式是 $\sum_{i=1}^{m} e_i / v_i$，其中 e_i / v_i 是单个更新事务的效用值。调度率与效用间存在关联关系。

我们使用单个 CPU 的实时数据库系统，实时数据对象的

个数从 50 变化到 300，每个实时数据对象的有效期从 120ms 增加到 300ms，其执行时间在 3～5ms 之间。

4.2 采样数值变化

我们对采摘头的自转转速进行了实验，图 3 给出了当选择 δ 的变化范围为实际数值的 5%时，某个时间区间内的数据变化关系。

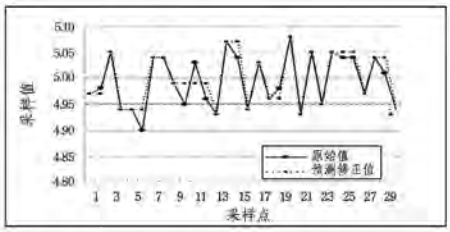

图 3 指定区间内的采样值变化关系

从图 3 可以看出，原始值的变化波动较大，而通过预测方法后，数值的瞬时精确度降低，在某些区间出现平滑现象，我们认为出现平滑的区间内的数据小于波动范围 δ。

4.3 CPU 负荷与调度率的比较

在实验中，当选择 δ 的变化范围为实际数值的 5%，并且有效期扩展的范围为有效期的 1.5 倍时，随着执行任务的增多，CPU 负荷越来越高，DS-FP 与本算法的比较结果如图 4 所示，1 号线是本文中的调度率曲线，2 号线为 DS-FP 算法的调度率曲线。

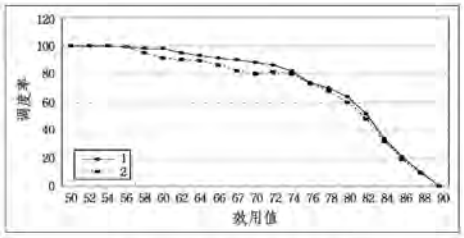

图 4 效用值与调度率关系

从图 4 可以看出，更新事务较少时，差别不大；当更新事务增多后，本文算法优于 DS-FP 算法。

4.4 δ 的变化和 v_i 的变化带来的调度率与负荷的变化

实验组合几种 δ 的变化以及 v_i 的变化情况，然后在更新事务数变化的情况下进行一系列测试，取得平均的调度率值和效用值。测试方法：选择一组参数，然后提交数量为 n 的更新事务，执行时间为 t。平均调度率 $AS=m/\sum_{i=1}^{n}(t/p_i)$，其中 m 为时间 t 内在截止期内完成的任务数，p_i 是实时事务的平均周期。平均 CPU 使用率 $AL=(\sum_{i=1}^{l}PE_i+\sum_{i=1}^{m}e_i)/t$，其中 l 表示预测事务执行的次数，m 表示更新事务的 CPU 实际使用时间。测试结果如表 1 所列，在误差不变的情况下，有效期的扩大会提高可调度性，但是随着业务数量的增大，提高速度较慢。在有效期一样、误差较大的情况下，可调度性较高。

在表 1 中，为了说明可调度率的变化，将第一种情况作为基准，即误差为 5%及有效期不变。当有 300 个任务且 CPU 使用率达到 85%时，其他 3 种情况的可调度率提高了 4%，1.5%及 5%。从实验结果可以看出，有效期对可调度率的影响较大。

表 1 平均 CPU 使用率以及平均调度率

	误差=5%；有效期不变		误差=5%；有效期 1.5 倍		误差=10%；有效期不变		误差=10%；有效期 1.5 倍	
任务数	AS	AL	AS	AL	AS	AL	AS	AL
100	1.00	0.48	1.00	0.46	1.00	0.48	1.00	0.45
150	1.00	0.65	1.00	0.65	1.00	0.65	1.00	0.64
200	0.98	0.80	0.98	0.78	0.98	0.79	0.99	0.76
250	0.82	0.84	0.86	0.83	0.84	0.83	0.88	0.84
300	0.73	0.85	0.76	0.85	0.74	0.85	0.77	0.85

采用表 1 的实验条件，要求不可调度任务数低于 0.1%，更新事务数为 300。在使用 DS-FP 方法时，CPU 使用率达到 98%以上。在误差为 5%及有效期不变的情况下，使用本文中的方法部分更新事务使用 CPU 时间，其他更新事务只使用语义判定而不执行更新事务，此时 CPU 的负荷达到 85%，CPU 负荷减少 13%；在误差为 5%及有效期为 1.5 倍时，CPU 负荷减少 16%；在误差为 10%及有效期不变时，CPU 负荷减少 14%；在误差为 10%及有效期为 1.5 倍时，CPU 负荷减少 18%。综合来说，CPU 负荷平均减少 15%左右。

4.5 CPU 使用率与任务的不可调度任务数

针对不可调度任务数量与 CPU 负荷的关系，图 5 给出了 DS-FP 与本算法的比较结果，x 轴代表 CPU 使用率，y 轴代表不可调度的任务数(6 代表有 6 个失败的更新事务)。曲线 1 代表 DS-FP 方法，曲线 2 代表本文的方法。CPU 使用率在 92%以上时，可以减少不可调度任务的数量。当 CPU 使用率在 99%以上时，该方法的优势消失。

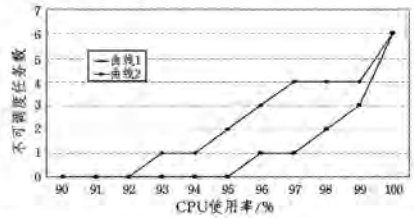

图 5 CPU 使用率与不可调度任务数的变化关系

结束语 在实时数据更新处理应用中，采用基于语义的实时数据有效性保证策略后大大缓解了因 CPU 负荷高而导致的实时数据处理能力下降的问题，系统的资源被充分合理地利用。本文算法应用于一个大型企业的高端复杂收获机械 MRO 时效果较好。

参 考 文 献

[1] FETH P, BAUER T, KUHN T. Virtual Validation of Cyber Physical Systems[J]. Software Engineering & Management, 2015, 239: 201-206.

[2] LU C Y, SAIFULLAH A, et al. Real-Time Wireless Sensor-Actuator Networks for Industrial Cyber-Physical Systems[J]. Proceedings of the IEEE, 2016, 104(5): 1013-1024.

[3] BAHETI, RADHAKISAN, GILL H. Cyber-physical-systems[C]// The Imapct of Control Techonoly. 2011: 161-166.

[4] LEE E A. Architectural support for cyber-physical systems[C]// ACM SIGARCH Computer Architecture News. 2015: 1.

[5] HAN S, CHENY D J, XIONG M, et al. Schedulability Analysis of Deferrable Scheduling Algorithms for Maintaining Real-Time

(下转第 22 页)

ta. Scottsdale, 2012: 481-492.

[3] DONG X, GABRILOVICH E, HEITZ G, et al. Knowledge vault: A web-scale approach to probabilistic knowledge fusion [C]// Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, 2014: 601-610.

[4] CARLSON A, BETTERIDGE J, KISIEL B, et al. Toward an Architecture for Never-Ending Language Learning[C]// AAAI. Atlanta, 2010.

[5] AUER S, BIZER C, KOBILAROV G, et al. Dbpedia: A nucleus for a web of open data[M]. The Semantic Web. Berlin Heidelberg: Springer, 2007: 722-735.

[6] BIEGA J, KUZEY E, SUCHANEK F M. Inside YAGO2s: A transparent information extraction architecture[C]// Proceedings of the 22nd International Conference on World Wide Web. Riode Janeiro, 2013: 325-328.

[7] WANG Y Z, JIA Y T, LIU D W, et al. Open Web Knowledge Aided Information Search and Data Mining[J]. Journal of Computer Research and Development, 2015, 52(2): 456-474. (in Chinese)
王元卓, 贾岩涛, 刘大伟, 等. 基于开放网络知识的信息检索与数据挖掘[J]. 计算机研究与发展, 2015, 52(2): 456-474.

[8] JIA P, LI X B, WANG J X. Comparison of several kinds of typical comprehensive evaluation methods[J]. Chinese Journal of Hospital Statistics, 2008, 15(4): 351-353. (in Chinese)
贾品, 李晓斌, 王金秀. 几种典型综合评价方法的比较[J]. 中国医院统计, 2008, 15(4): 351-353.

[9] Cold Start Knowledge Base Population at TAC 2015 Task Description[OL]. [2015-7-14]. http://www.nist.gov/tac/2015/KBP/ColdStart/guidelines.html.

[10] LIU X M, DAO K Q. Construction of Evaluation Model for Institutional Repository Basing on Fuzzy Comprehensive Evaluation[J]. Information Research, 2015(5): 22-24, 28. (in Chinese)
刘雪梅, 刀克群. 基于模糊综合评价法的机构知识库评价模型构建[J]. 情报探索, 2015(5): 22-24, 28.

[11] YAO K. The comprehensive evaluation system for agricultural knowledge base based on multiple dimension QoS[D]. Xianyang: Northwest A&F University, 2015. (in Chinese)
姚坤. 基于多维 QoS 的农业知识库综合评价系统[D]. 咸阳: 西北农林科技大学, 2015.

[12] ADELMAN L, RIEDEL S L. Handbook for evaluating knowledge-based systems: Conceptual framework and compendium of methods[M]. New York: Springer Science & Business Media, 2012: 317-318.

[13] SUN P S, FAN Z P, CHEN X, et al. Framework and Process for Evaluating the Construction Effectiveness of Knowledge Base [J]. Journal of Northeastern University (Natural Science), 2010, 31(9): 1361-1364. (in Chinese)
孙培山, 樊治平, 陈曦, 等. 评价知识库构建效果的框架与流程[J]. 东北大学学报(自然科学版), 2010, 31(9): 1361-1364.

[14] TJONG K, ERIK F, FIEN D. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition[C]// Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003. Edmonton, 2003: 142-147.

[15] PRADHAN S, LUO X, RECASENS M, et al. Scoring Coreference Partitions of Predicted Mentions: A Reference Implementation[J]. ACL, 2014(2): 30-35.

[16] TAC 2014 Cold Start[OL]. [2014-11-17]. http://www.nist.gov/tac/protected/2014/TAC2014-workshop-notebook/agenda.htm.

[17] CHEN X L, PANG L, JIA Y T, et al. Word Vector-based Recognition for Unstructured Text Domain Concepts[J]. Journal of Shanxi University (Natural Science Edition), 2015, 38(4): 553-559. (in Chinese)
陈新蕾, 庞琳, 贾岩涛, 等. 基于词向量的开放文本领域概念识别方法[J]. 山西大学学报(自然科学版), 2015, 38(4): 553-559.

[18] ZHAO Z Y, JIA Y T, WANG Y Z, et al. Link Inference in Large Scale Evolutionable Knowledge Network[J]. Journal of Computer Research and Development, 2016, 53(2): 492-502. (in Chinese)
赵泽亚, 贾岩涛, 王元卓, 等. 大规模演化知识网络中的关联推理[J]. 计算机研究与发展, 2016, 53(2): 492-502.

(上接第 16 页)

Data Freshness[J]. IEEE Transactions on Computers, 2014, 63(4): 979-994.

[6] LOCKE D, BESTAVORS A, LIN K J, et al. Real-timedatabase: Read-world requirements[C]// Real-Time Database Systems: Issues and Applications. Kluwer Academic, 1997: 83-91.

[7] RAMAMRITHAM K. Real-time database[J]. Distributed and Parallel Database, 1993, 1(2): 199-266.

[8] STANKOVIC J A, SON S H, HANSSON J. Misconceptions about real-time databases[M]// Real-Time Database Systems. 2002: 9-16.

[9] HO S J, KUO T W, MOK A K. Similarity-based load adjustment for real-time data-intensive applications[C]// Proceedings of IEEE Real-Time Systems Symposium. 1997: 144-153.

[10] XIONG M, RAMAMRITHAM K. Deriving deadlines and periods for real-time update transactions[J]. IEEE Transactions on Computers, 2004, 53(5): 567-583.

[11] BAI T, LI G H, LIU Y S. A sensor transaction scheduling algorithm for maintaining real-time data temporal validity[J]. Journal of Central South University of Technology, 2011, 18(6): 2068-2073.

[12] 周本海, 姚大鹏. CPS 中基于预留的实时调度算法研究[C]// 全国 Petri 网理论与应用学术年会. 2013.

[13] 白天. 数据的时间一致性维护策略[M]. 长沙: 中南大学出版社, 2015: 38-50.