传感器网络中协作任务的实时调度*)

胡 侃 刘云生
(华中科技大学计算机学院 武汉 430074)

摘 要 在传感器网络实时监测应用中,大量传感器散布在监测区域中感知监测域的各种环境或监测对象的信息,一组功能有限的传感器往往相互协作地完成一个大的实时感知任务,协作性是传感器网络的重要特性,它要求实时任务之间的资源共享。单纯的实时系统为保证任务的实时性通常采用资源隔离机制而不能很好地解决传感器网络环境中的采集流数据处理的协作性问题。本文基于服务器的调度框架,使用了实时环境中的时间属性,将数据时间与程序时间相结合,从而将应用语义与系统中运行的程序相联系,提出了一种基于时间依赖关系的实时调度模式,并给出了基于此模式的事件驱动并发数据流程图模型及其实现机制。分析表明,该模型能有效地解决传感器网络监测区域中采集流数据处理过程的协作性问题,减少了数据丢失,提高了系统响应的实时性。

关键词 实时系统,协作执行,流数据,数据流模型,传感器网络

Real-time Scheduling for Cooperative Tasks in Sensor Networks

HU Kan LIU Yun-Sheng
(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract Wireless sensor networks are being developed for a variety of applications. With the continuing advances in network and application design, appropriate middleware is needed to provide the capability to support efficient real-time QoS for coordinative concurrent applications on sensor networks. In traditional simple real-time system, resource isolation mechanism is utilized for ensuring the real-time response of the tasks, it is futile to the problems where streaming data are processed collaboratively by concurrent fusion tasks and timely responses are important. The paper studies a schedule approach for providing efficient real-time services to concurrent coordinative tasks in sensor networks. We first propose a coordinative execution scheme for data stream processing. And then, by evolving server scheduling techniques and combining timestamp of data stream with the task execution time to tailor the design of schedule software, we present a three-layer scheduling architecture to provide real-time services while guaranteeing coordination consistency. Finally, we design a three-phrase scheduling algorithms based on the model. Our performance evaluations show that by applying the three-phrase scheduling algorithms, the outer transformation ratio of collection data can be significantly enhanced and the amount of lost data can be reduced correspondingly.

Keywords Real-time system, Collaboration execution, Data stream, Data-flow model, Sensor networks

随着计算机技术、通信技术、传感器技术的发展,计算机的应用越来越广泛,如在国防军事、智能交通、网络监控等领域。在传感器网络实时监测应用中,大量传感器散布在监测区域中感知监测域的各种环境或监测对象的信息,一组功能有限的传感器往往相互协作地完成一个大的实时感知任务,系统能协作地实时监测、感知和采集网络分布区域内的各种环境和监测对象的信息,并对这些信息进行分析、处理,将结果外化给系统观察者。协作性是传感器网络的重要特性,它要求实时任务之间的可见性,另外还有实时性、移动性、断接性、能源有限性等特点[1]。对于大量实时采集流数据的管理导致了流数据(streaming data)[2,3]模型的研究,出现了许多流数据模型下的管理系统(DSMS),包括斯坦福大学的STREAM、加州大学伯克利校的 Telegraph、布朗大学和麻省理工合作的 Aurora 等。针对具体的行业,它们能有效地管理流数据。

目前关于实时任务的研究已有不少,但大部分是在单纯实时系统中,仅仅考虑任务的定时特性[4~7],系统根据任务的实时要求分配资源,要求应用之间的资源隔离机制支持。这种实时任务的资源隔离无法解决传感器网络实时应用中由传感器采集数据驱动相互独立执行的实时任务的协作问题。

本文利用实时操作系统的调度机制、DSMS 的数据管理机制,使用控制区域的概念提出一种适宜传感器网络实时流数据协作处理的基于时间依赖的数据驱动实时任务调度模型及其实现机制。通过分析表明,该模型提高了系统对采集监测流数据的分析处理的描述能力和实时响应性能。

1 流数据模型及协作实时任务

1.1 流数据模型

令 t 表示任一时间戳,a_t 表示在该时间戳到达的数据,流数据可以表示成$\{\cdots,a_{t-1},a_t,a_{t+1},\cdots\}$。区别于传统模型,流

*)本课题得到国家自然科学基金项目(60073045)、博士点基金及国防预研基金(00J15.3.3.JW0529)资助。**胡 侃** 博士生,讲师,主要研究多元实时数据集成、移动数据管理及实时数据库;**刘云生** 教授,博士生导师,主要研究方向为现代(实时、主动、内存、移动等非传统)数据库理论与技术及其集成实现、数据库与信息系统开发、实时数据工程、软件方法学与工程技术。

模型具有以下4点共性：

(1) 数据实时到达；

(2) 数据到达次序独立，不受应用系统控制；

(3) 数据规模宏大且不能预知其最大值；

(4) 数据一经处理，除非特意保存，否则不能被再次取出处理，或者再次提取数据代价昂贵。

为了满足这种规模宏大且到达速度很快的数据处理的实时要求，现在通常使用一种不同于传统实时数据库的称之为概要数据结构(synopsis data structure)数据模型[2]及基于流数据模型的DSMS。用户通过DSMS获取流数据。如图1所示。

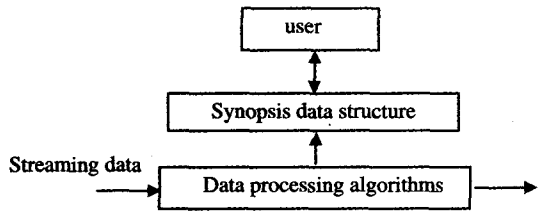

图1 数据流处理模型

DSMS给用户(用户程序)提供了带有时标的数据 X_t，该时标 t 及数据值 VALUE(X)反映了数据所代表的监控对象(或经处理的导出对象)在时刻 t 的状态信息。

1.2 协作实时任务

在传感器网络应用中，一组功能有限的传感器协作完成一个大的感知任务。一个或几个传感器采集数据事件往往驱动一个实时任务的独立执行，因此传感器的协作反映在系统中是一组独立执行的实时任务相互协作的完成一个实时应用，协作性要求协作任务相互间持有资源的可见性。图2表明了基本的实时任务协作执行过程。

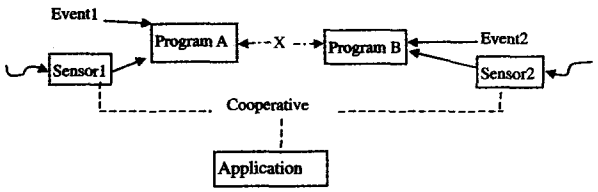

图2 协作任务的执行模式

A、B是两个独立执行的实时任务，它们由事件1和事件2独立驱动，并分别对传感器1和传感器2采集的数据进行分析、处理。通常A、B各自都是相对完整的逻辑单位，他们可以给外界用户提供关于被监测对象的有用的局部信息。由于传感器1和传感器2是一组协作传感器，因此，从更高的应用抽象层次来看，A、B是一个完整的语义逻辑单位，它们协作地完成一个大的感知任务，给外界用户提供被监测对象的准确的全局信息，协作性使得异步执行的A、B在执行时可以相互间同步地使用共享资源X。

这种异步执行的实时任务由更高层次的应用语义决定的相互间的同步合作关系是传感器网络的一种基本的协作方式。这意味着协作任务间在一些同步点上应可获得对方持有的资源，从应用语义逻辑完整的角度来看，这导致了协作任务的相互依赖而不能各自独立完成，最终影响了各个独立执行任务的实时性，使得它们因超截止期而失败。

通常的实时调度中，为保证应用逻辑完整性，A或B不能单独作为系统的调度对象。编程者只能编制一个更大的程序C，它将A、B作为子程序，并由事件1和事件2都发生来驱动，对传感器1和传感器2同步采集的数据进行处理。这样尽管保证了外化信息的准确性，但是不仅影响了A、B的实时性，而且造成了传感器采集数据的丢失。

1.3 实时任务调度

传感器网络实时系统往往是一个集于多类型任务共存的开放式实时系统，由传感器驱动的异步执行的实时任务应可单独开发和验证，当系统中传感器动态扩展时无须作全局的实时性分析。目前的相关调度方法都采用基于服务器的策略，这里的服务器指系统调度机制创建的特殊任务，它为调度对象提供服务，可看作是一个抽象的独立CPU。图3是典型的双层调度框架。

底层由采用EDF算法的系统调度器对服务器的集合 $\{S_1, S_2, \cdots, S_n\}$ 进行调度，每个服务器按一定比例占用处理机时间，同时它又与一个TSS(task-set and scheduler)相关联，为其提供资源服务。上层的每个TSS包含一个局部调度器，它决定TSS中的任务组 $\{T_1, T_2, \cdots, T_m\}$ 的调度顺序。局部调度器会产生一个作业的执行队列，由服务器执行。一个服务器相当于一个具有一定速率的处理机，服务器的能量反映了服务器从当前时刻起可占用系统处理机的时间。

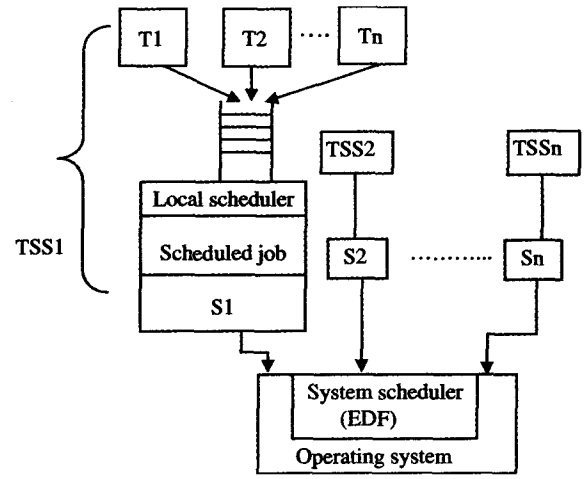

图3 双层调度框架

按照这种双层调度框架，若将图2中的A、B作为单独的调度对象则存在一些关键性的问题需要解决：

(1) 协作语义一致性问题。一个任务可以多次执行(多实例)，系统如何识别一个服务器上独立执行的任务的哪次执行实例与另一个服务器上调度任务的哪次执行实例是一个大的感知应用的协作执行过程，以保证应用语义逻辑正确性。如一个监测飞机的应用包括计算飞机高度、计算飞机速度并依此计算飞机方位的三个独立的协作任务。若高度是两分钟以前的监测值，速度是两分钟以后的监测值，这样计算的方位将存在很大的偏差。若将此交给应用开发者，则只有当多个传感器都采集到监测对象当前值时才能执行任务，这些任务作为一个作业在一个服务器上运行，这样影响了系统的实时响应。

(2) 服务器能量补充问题。在一个服务器上运行的任务由于协作性与其它服务器上运行的任务相互共享资源而在更高的抽象层次形成一个语义上完整的应用。一个独立执行的任务完成，其对应的应用作业并未完成。为保证应用语义逻辑完整，各个服务器能量只有在应用作业完成后才能补充，这

样降低了平均任务响应时间，不适于传感器网络实时监测。系统如何保证各个服务器能量在任务执行完成后就得到补充，但是协作任务实例间使用的全局共享资源在应用作业完成后才能释放。

(3) 资源共享问题。现有的实时调度都要求资源隔离机制支持，而协作性要求一个服务器上运行的任务使用的全局共享资源在它结束后仍然能被其它协作任务使用。如何控制传感器网络中协作的共享资源是一个关键的问题。

对上述协作问题，现有的实时调度无法有效的支持。下面给出一种基于任务应用语义依赖的任务调度机制。它是事件驱动数据流程图模型[8]一种扩展。

2 基于应用语义的任务调度

2.1 三层的调度模型

在传感器实时应用环境中，不仅实时采集数据带有时间标记，任务、作业、线程、进程也带有时间标记。记 $t(O)$ 表示对象 O 的时标。

定义 1 设 S 是一个由带有时间标记的对象组成的集合，$S=\{a_i \mid i=1,\cdots,n\}$。则定义 S 的时标为 $t(S)=1/n\sum t(a_i)$。

在复杂应用中，任务间存在着多种依赖关系。为定义我们的模型，引入了时间依赖关系。

定义 2 任意两个带有时间标记的对象 X、Y。对一个给定的阈值 T_C(小于对象的有效期)，若 $|t(X)-t(Y)|<T_C$，则称对象 X、Y 是时间依赖的。同样可定义时间依赖对象集为该集合中任意两个元素间存在时间依赖关系。

实时任务(以下简称任务)是指完成某一特定功能的软件实体，它是实时系统的一个功能独立的事件(如传感器采集数据)触发的独立执行的单位，是系统的调度控制单位，任务的一次执行称为进程，它是对外界监测对象某时刻状态信息(采集数据集 $D=\{d_i \mid i=1,\cdots,n\}$)进行分析处理的过程，通常在服务器上按截止器最小调度运行，D 是一个时间依赖集合。同一个任务的两次执行是两个不同的进程。任一个进程 P 具有一个时标，它等于 $t(D)$，即处理的采集数据集时标的均值。一个进程是一个相对完整的语义逻辑单位，它可以产生外化数据给外界，也可以通过使用全局共享变量与其它协作进程通信，对写入全局变量的数据系统应赋予与该进程时标值相同的时标。

定义 3 一个任务是一个六元组 $RTT=(E,T_I,E_I,T,T_O,C)$。E 是一个可激活函数，返回一个布尔值，当该任务的执行事件发生时，返回真值。T_I 是任务处理的所有实时流数据集合组成的输入端子，$T_I=\{a_i \mid i=1,\cdots,n\}$。$E_I$ 是一个可激活函数，返回一个布尔值。当 E 的返回值为真时，E_I 被激活，当 T_I 的元素都到达(或到达一定量)且 T_I 是一个时间依赖集，则返回真值。T 是 RTT 的执行程序体，当 E_I 返回值为真时创建进程放入服务器队列调度执行，系统赋予该进程时标值等于 $t(T_I)$。T_O 是输出端子，其值(系统赋予时标 $t(T_I)$)作为进程的全局变量值由其父超进程(见定义 4)控制被其它协作进程使用。C 为 RTT 的定时限制。RTT 的一次执行记作 RTT-t。

一个实时应用(以下简称应用)由多个任务协作完成，它是一个任务组。在传感器网络中，一个任务组由一组按照应用语义相互协作的事件独立驱动的任务组成。应用的一次执行称为超进程，它是系统自动生成的由一组相互时间依赖的协作进程组成的集合，每个进程分别是任务组中一个任务的一个执行实例。即一个超进程是一组协作进程对外界监测对象同一个时刻状态信息进行分析处理的一次协作执行过程。

定义 4 一个超进程是一个四元组 $RTA\text{-}t=(RTT\text{-}t^*,AR,TR,T)$。$RTT\text{-}t^*$ 是一个由一组协作的 $RTT\text{-}t$ 组成的集合。AR 是定义在 $RTT\text{-}t^*$ 上的由应用语义决定的协作依赖关系约束，保证 $RTT\text{-}t^*$ 是一组协作任务组的一组实例集合。TR 是 $RTT\text{-}t^*$ 上的时间依赖约束，保证这组协作进程集合是对监测区域中监测对象某一时刻 t 的状态信息的分析处理的一次执行协作执行过程。T 是超进程的定时限制。

按照上述定义，传感器系统的调度对象由两个层次组成，如图 4 所示。

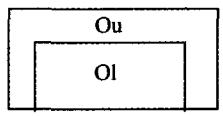

图 4 两层调度对象

从动态的角度看，下层对象是进程集合，上层对象是由存在时间依赖关系及协作关系的一组下层元素组成的对象(超进程)集合。

一个超进程其实质是系统自动根据进程间的时间依赖、应用协作依赖关系建立的一个控制一组由独立事件驱动的进程间的一次协作执行过程的控制区域，系统通过它控制一个进程组的协作执行语义一致性、时间一致性、协作进程间使用的共享资源等问题，它作为一个特殊的进程在系统中存在，它的地址空间自动生成，由全局的共享资源地址空间及其当前子进程地址空间组成并随着子进程的变化而自动变化，一个超进程包含多个地址空间，即一个超进程的地址空间包含多个进程。当一个进程创建时，系统自动地查找其父超进程，若存在父进程，则将该进程加入父超进程，否则，创建一个父超进程。超进程的结束分为两种情况：一种是它的所有子进程都结束而正常结束，此时，系统不仅给用户外化了局部信息，也外化了全局信息；另一种是由于传感器的断接性、能源有限性、移动性等原因造成的采集数据未传入基站造成部分子进程未执行而超时终止，这种情况下系统只外化了局部信息(但有用的)。每个进程是系统中独立的调度单位，由系统的实时资源隔离机制分配地址空间，其开始、结束不受其它进程的影响，由系统事件触发调度器控制，它有一个时标，说明该进程是对监测对象哪个时刻的状态进行处理。系统通过实时调度机制确保实时性。协作进程使用的共享数据资源由超进程管理，进程可以使用进程到超进程的切换(SEND,RECEIVE)请求使用共享资源。本文定义 3 中，进程通过输出端子将其全局数据释放给父超进程。超进程通过时间依赖控制一组协作语义一致的进程产生一个时间一致的导出数据模版，驱动另一个协作进程的执行，从而实现协作进程之间的满足一致性的通信。进程只有私有地址空间，进程结束释放私有资源，全局共享资源由超进程控制直到超进程结束。

基于上述定义，我们将由一个(或几个)传感器采集数据(或一个导出数据模版)事件驱动的任务在进程服务器上调度，该服务器具有一个队列，它们一起称为进程服务器类。一个服务器类在一个网络站点上运行。超进程在超进程服务器类上运行。进程独立执行、结束，不受超进程控制影响，进程产生和使用的共享信息资源由超进程控制，超进程控制协作

进程的共享资源直到多个协作进程执行完。图5是扩展的三层调度框架。ESS代表事件驱动调度器。

A、B分别由事件1和事件2驱动，并对传感器1和传感器2采集数据处理，它们可以输出局部完整的信息给外界并产生全局导出数据，超进程根据时间依赖将一组满足时间一致性的导出数据组成一个数据模版，事件3驱动C并对该模版数据处理。A、B、C的一次满足时间依赖的执行系统将建立一个超进程来控制。

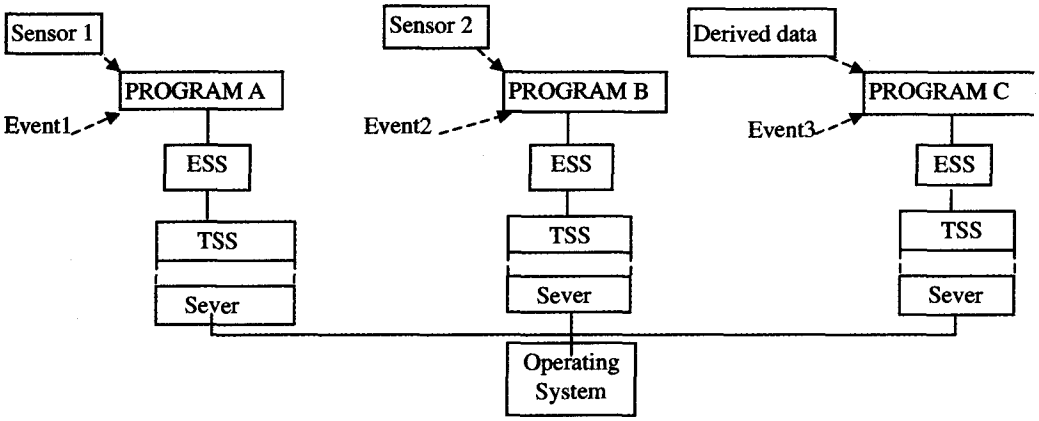

图5 三层调度结构

对ESS调度的任务，系统是在一个进程服务器类上按照实时调度FIFO响应的，如进程服务器采用平均响应比高的TBS(total bandwidth server)调度方案。每个应用的一次执行对应于一个超进程，在超进程服务器类上调度。这样系统保证了对每个实时任务请求的实时性，同时也保证了协作进程之间的一致性及应用逻辑正确性。

2.2 调度算法

ESS调度器包括事件驱动调度和超进程调度。当一个事件发生时，系统创建一个进程，并计算出其时标值。然后调用超进程管理器，将该进程加入超进程。每个进程在相应的站点按照TSS调度执行。系统通过一个定时器来控制超进程的结束，当超进程的所有子进程结束或它的定时器到则该超进程结束。进程结束时将输出端子的数据放入其父超进程的全局数据模版中，一个超进程的全局数据模版中存放着一组协作任务的一组时间依赖的进程产生的导出数据，它可以驱动一个进程执行。下面是ESS调度器的主要算法。

```
Event-trigger () {
  IF event occurs and collection data arrives
    THEN calculate timestamp;
         Create a process into TSS queue;
         Invoke super-process ();
         Invoke TSS schedule;
    ELSE wait until event occurs;
}
Super-process () {
 Search father super-process;
   IF there not exists a father
     THEN create a new father super-process and set a time-out
       Allocate the global data template to its father;
  Join the process into its father;
  }
End-process () {
  Announce father;
  Release resource to system;
  Release output-port to father's data template;
  }
End super-process () {
  IF all sons end or time-out
    THEN release all resource;
  }
```

3 实现框架

3.1 系统数据模型

按照简单的数据模型，即将不同组件及其之间的关系看作实体类型表示，描述上述定义的关键术语如图6所示。

此模型描述了九个实体类型。它们分别将相应的事件请求映射为实际的超进程、进程和线程。系统一个设备发生一个事件，该事件表示想要系统提供一个服务。有一个任务程序或节点函数实现了该服务。多个服务可能由一个程序(函数)实现。对于每个任务程序系统提供一个进程服务器类，这个服务器类由一组进程组成，每个进程是任务程序对于不同时间的实时采集流数据(导出数据)的一次执行，称为进程服务器，进程的执行需要输入模版。一个应用实现了用户希望的一个完整的实时应用要求，它由多个协作任务程序完成，系统建立一个超进程服务器类和超进程服务器(限于篇幅图中未画出)。一个超进程由多个时间依赖相互协作的进程组成，一个进程由多个线程组成。当发生一个事件请求，它表明了所需服务，该服务需要对实时采集数据处理，实时数据(包括中间导出数据)表示为端子类，一个端子类由多个描述现实世界很小的时间段内不同监测对象(或同一对象的不同属性)的存在时间依赖的实时数据集合(端子模版)组成。对于每个事件请求，系统必须指派正确的进程服务器类中的一个进程服务器，系统必须根据时间依赖和应用依赖建立正确的超进程服务器类中的超进程服务器，进程是事件驱动的调度和分配资源的单位，超进程是系统自动建立的与应用语义相关的一组时间相关的进程集及回收资源的单位。

该数据模型基本上描述了系统的数据结构。

3.2 执行模型

按照正常的处理流程，图7描述了基于时间、逻辑依赖关系的数据流模型中应用的执行模型。

管理器监听事件并转为事件请求给实时监控器，实时监控器(类似于实时代理)处理到达的请求、分配相应的应用程序及其资源，创建进程并根据时间依赖关系形成超进程。进程所需的输入模版及产生的输出模版由端子管理器管理，为保证实时性，一个协作进程结束其处理的部分结果可以外化给系统观察者，当一个超进程结束释放资源给系统并通过接口服务外化协作处理结果，为保证处理结果的有序性系统将自动终止其它未结束的时标值小于该超进程时标的其它超进程。数据采集器周期性的将SDMS中的流数据采集到端子管理器以便任务进程使用。

执行模型反映了系统各组件功能及相互间调用关系。

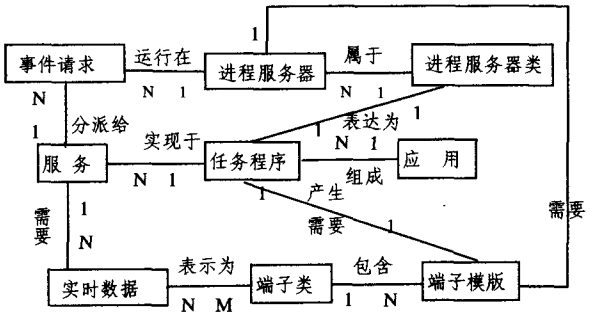

图 6 系统数据模型

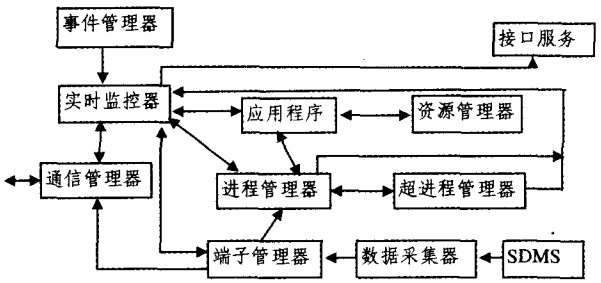

图 7 统执行模型

4 性能分析

我们在由国家自然科学基金资助、自行研制的分布式主动实时数据库原型系统 ARTs-Ⅱ及嵌入式实时操作系统上模拟传感器网络计算环境完成了时间依赖事件触发数据流模型的性能测试。实验采用采集数据的外化率(单位时间观察者看到数据)与采集数据的采集率(单位时间输入系统的采集数据)之百分比 MR(相对外化率)为测试指标。

实验例程为系统对 A、B 两个不同工作站点上采集的代表监测区域不同地点的温度值进行处理,使用图形方式分别显示每个地点及整个区域的温度变化情况。每个节点有个缓冲区,传感器不断地将采集数据输入该缓冲区,采集器将该缓冲区中数据附时标交给端子管理器。若采用一般事件驱动数据流模型,为保证采集数据的时间一致性,当两个缓冲区都满时才驱动进程。一个缓冲区先满,新到的采集数据自动覆盖先前的数据直到另一个缓冲区满。按照该模型,应用由一个实时任务描述,它包括四个节点函数,分析温度函数、两个分别显示 A、B 点温度的函数及一个显示全局温度的函数。若采用我们的时间依赖事件驱动模型,应用由三个独立的协作任务组成,它包括两个分别在 A、B 站点由该点缓冲区满驱动的处理局部温度、显示局部温度及将导出结果交给端子管理器的任务及一个由数据(存在一个时间依赖的包括 A、B 点温度导出值的输入模版)驱动的显示全局温度的任务组成。表 1 是主要模拟参数。图 8 是传感器采集数据周期对 MR 值的影响。

从图中可以看出,当传感器采样周期越小,时间依赖事件驱动模型明显优于纯事件驱动模型。另外在测试中发现当采集周期小于等于 10ms 时,纯事件驱动模型发生 FIFO 队列溢出,若改变 A、B 点采集速度,对局部温度变化输出时间依赖事件驱动模型明显要比纯事件驱动模型的输出快,这说明前者在处理突发事件上比后者好。

表 1 模拟参数

parameter	Value	parameter	Value
test time	10s	average message send time	450us
number of nodes	5	data send period	10—100ms

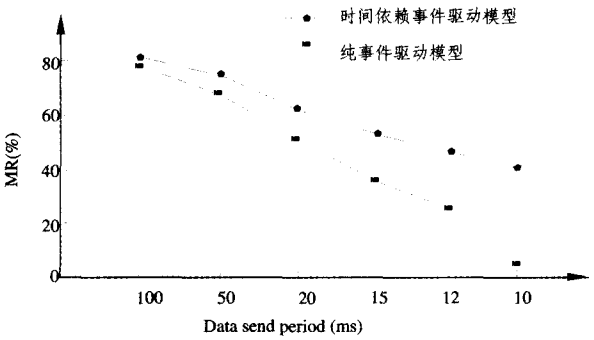

图 8 采样周期对 MR 的影响

结束语 本模型主要是为解决在复杂的实时传感器网络环境中,系统能及时地、准确地将监测区域中监测对象的状态进行分析处理传递给决策者。经分析表明,基于时间依赖的事件驱动模型不仅能保证异步执行的协作进程之间的协作语义一致性、时间一致性,还能减少采集数据的丢失,提高系统的实时响应。

参 考 文 献

1 Tilak S, Abu-Ghazaleh N B, Heinzelman W. A taxonomy of wireless micro-sessor network models. Mobile Computing and Communications Review, 2002, 1(2): 1～8

2 Babcock B, Babu S, Datar M, et al. Models and issues in data strerms. In: Popa L, ed. Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symp on Principles of Database Systems. Madison: ACM Press, 2002. 1～16

3 Carney D, Cetinternel U, Cherniack M, et al. Monitoring streams—A new class of DBMS applications: [Technical Report]. CS-02-01. Province: Department of Computer Science, Brown University, 2002

4 Parekh A K. A generalized processor sharing approach to flow control in integrated services networks: [Ph D Thesis]. MIT, 1992

5 Muthiayen D, Alagar V S, Khendek F. An approach to a synthesis of formal and description techniques for the development of real-time reactive systems. In: IEEE, ed. Processings of the IEEE Conference on Real-time Computing Systems and Applications. Cheju Island: IEEE, 2000. 491～497

6 Lipari G, Carpenter J, Baruah S. A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments. In: Jacobs A, ed. Proceedings of the 21th IEEE Real-time Systems Symposium. Los Alamitos, CA: IEEE Computer Society Press, 2000. 217～226

7 Deng Z, Liu J W S. Scheduling real-time applications in open environment. In: Proceedings of the 18th IEEE Real-time Systems Symposium. Los Alamitos, CA: IEEE Computer Society Press, 1997. 308～319

8 Whiting P G, Pascoer S V. A history of data-flow languages. IEEE Annuals of the History of Computing, 1994, 16(4): 38～59