物联网中具有时间持续性特征的乱序事件查询处理技术研究

周春姐[1] 戴鹏飞[2] 李洪波[1] 张振兴[1]
(鲁东大学信息与电气工程学院 烟台 264000)[1] (中国科学院计算技术研究所烟台分所 烟台 264000)[2]

摘 要 现实世界中的很多事件都是基于时间段的,具有明显的时间持续性特征。具有这种特征的事件之间的时态关系复杂多变,在发生乱序事件时,查询处理极具挑战性。物联网环境中对于时序事件的有序到达具有非常严苛的要求,但网络延迟和机器故障却导致事件乱序问题频发。基于建立的时态事件语义表示模型,提出了一种用于处理具有时间持续性特征的乱序事件的查询处理模式,并构建了一种混合解决方案,使物联网环境中乱序事件在到达后的一定时间阈值内达到正确有序。最后,通过实验验证了所提方法的有效性。

关键词 时间持续性,物联网,乱序事件,时态语义

中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.5.033

Research of Interval-based Out-of-order Event Processing in Internet of Things

ZHOU Chun-jie[1] DAI Peng-fei[2] LI Hong-bo[1] ZHANG Zhen-xing[1]
(School of Information and Electrical Engineering, Ludong University, Yantai 264000, China)[1]
(Institute of Network Technology, Institute of Computing Technology, Chinese Academy of Sciences, Yantai 264000, China)[2]

Abstract Many events in real world applications are long-lasting events which have certain durations. The temporal relationships among those durable events are often changeable and complex. Moreover, when there are out-of-order events, the query processing becomes more challenging. In applications of internet of things, ordered arriving of order events is strictly required. However, network latencies and machine failures often cause events to be out-of-order. In this work, we analyzed the preliminaries of event temporal semantics. A tree-plan model of out-of-order durable events was proposed. A hybrid solution was correspondingly introduced. Extensive experimental studies demonstrate the efficiency of our approach.

Keywords Interval-based, Internet of things, Out-of-order events, Temporal semantic

1 引言

随着物联网用户数量的日益增多,越来越多的应用需要处理事件流上的查询[1-4]。这些事件流是一些具有时间持续性特征的事件序列(以下简称时间段事件),它们或者是有序的,或者是无序的。这些物联网中原子事件之间的时态关系对于事件模式的识别非常关键。然而,现有的物联网环境中时间管理的研究工作往往只考虑了其中的一个方面,要么关注于有序的时间段事件,要么关注于乱序的时间点事件。但在物联网环境下急需整合这些因素,支持乱序的时间段事件模式检测是非常必要的。据我们所知,之前处理乱序事件的研究工作都没有考虑时间段的因素。在时间段事件的模式查询中,状态转换既取决于事件的类型,又取决于事件之间的关系。而有限状态自动机(Non-deterministic Finite Automata, NFA)模型的状态转换是严格有序的,所以传统的基于NFA的方法都不能直接、有效地模拟非事件和并行事件,而这类事件在时间段事件中又是很常见的。本文提出一种综合考虑了时间段事件和乱序事件的检测方法。

现有的研究工作大多只考虑了时间点事件,也就是假设事件是没有持续时间的[5-8]。这种假设通常将事件简化成一个有序序列,例如:"头痛->胃痛->呕吐"。然而现实世界中的很多事件都是有持续时间的,并且这些事件之间的时序关系也是很复杂的[9-11],所以上述时序模式不足以表达复杂的时序关系。在医疗、多媒体、气象学和财政学等领域,事件的持续时间都起了很重要的作用。例如许多糖尿病患者的症状都是血糖的增高和尿糖的缺失同时并存,只有地表示这两者的重叠性才能很好地诊断;又如,骨热病的一般症状是在发热后的第三天血小板就开始减少,只有很好地表示这种间隔性和时序性才能准确地把握治疗时间。显然,需要一种有效的算法来处理带有时间段的事件。为了解决这一问题,提出

到稿日期:2015-03-31 返修日期:2015-07-03 本文受国家自然科学基金项目(61202111,61472141,61273152,61303017),山东省高等学校科技计划项目(J12LN05),山东省自然科学基金联合专项项目(ZR2013FL009),烟台市科技发展计划项目(2013ZH092),鲁东大学博士基金项目(LY2012023)资助。

周春姐(1981-),女,博士,副教授,主要研究方向为物联网、数据挖掘,E-mail:lucyzcj@gmail.com;戴鹏飞(1982-),男,硕士,主要研究方向为大数据、云计算;李洪波(1969-),男,硕士,副教授,主要研究方向为数据挖掘、云计算;张振兴(1981-),男,博士,讲师,主要研究方向为物联网、模式挖掘。

了两种基于时间段的查询模式：串行查询模式和并行查询模式。

许多跟踪和监控系统应用中，对分布式设备上的事件流进行实时处理是一个关键的挑战性问题。而目前的很多系统，无论是基于事件的，还是基于流的，都假设事件是完全有序的。但是在物联网环境下，机器故障或者网络延迟经常导致处理引擎中的事件流发生乱序。因此，在这种情况下现有的方法就会抛错，出现漏解或者产生错误的事件模式。以书店中追踪书籍为例，每本书都贴有一个 RFID 标签。RFID 读卡器被放在书店的几个关键位置，如书架、结账口和出口等。如果在书架上和出口处的读卡器检测到同一本书，但是在两者之间的结账口处没有检测到该书，那么这本书可能被偷窃了。但是如果在结账口处的读卡器事件发生了乱序，那么就会漏掉一些期望的结果或者发出错误的预警信号，从而事件处理的准确性就无法得到保证。显然，高效、实时地处理有序和无序的事件流已需亟待解决。

现有的很多复合事件处理系统都是利用 NFA 来检测事件模式的[4,10,12]。但是基于 NFA 的方法存在 3 个限制条件：(1)现有的基于 NFA 的方法中事件间的状态转换是有严格次序的；(2)在 NFA 中无法直接有效地模拟非事件；(3)NFA 状态转换的严格有序使其很难支持并行事件。因此，本文提出了基于树的查询模式来表示事件的逻辑和物理结构。

本文的主要贡献包括：

- 给出了时态语义的严格定义，并且基于时间段提出了两种查询模式——串行模式和并行模式；
- 提出了时间段乱序事件的逻辑和物理表示模型，以及这些事件的检测方法；
- 提出了基于树的复合事件查询结构，可以进行多项优化操作；
- 提出了一种混合方法来解决基于时间段的乱序事件，可以根据需求在准确性和实时性之间进行切换。

本文第 2 节介绍相关的研究工作；第 3 节是预备知识；第 4 节提出具有时间持续性特征的乱序事件模型和检测方法；第 5 节提出一种混合的解决方法和一些优化策略；第 6 节是实验结果与分析；最后是结束语。

2 相关工作

目前，在乱序事件和时间段事件方面都有一些相关的研究工作。然而据我们所知，还没有研究工作将两者综合考虑。同时考虑乱序事件和时间段事件在物联网中是非常重要和亟待解决的问题。

乱序事件方面的研究工作可被分为两大类：一类专注于实时性，其输出结果是乱序的；另一类专注于准确性，其输出结果是有序的。因为输入到查询引擎中的事件流是乱序的，所以输出乱序的事件也是合理的。文献[10]中允许输出乱序的事件序列，并提出了一种侵略性策略(aggressive strategy)的方法。在假设乱序事件很少的情况下，侵略性策略产生最大量的输出结果。相对应地，为了解决乱序事件的突发情况，侵略性策略设计了一些适当的错误补偿方法。

如果要求输出结果是有序的，就需要一些额外的语义信息，例如 K-Slack 因子或者标志符，用来限制准备输出的候选序列。下面分别简单介绍这两种方法。处理乱序事件的最原始的方法就是利用 K-Slack 作为乱序输入事件流的先验界定。它对输入队列中的输入事件缓冲 K 个时间点，直到能够保证输出事件的有序性。K-Slack 方法的最大缺点就是 K 的刚性限制，它无法适应异构 RFID 读卡器网络中网络延迟的变化。例如，K 的一个合理值可以设定为网络中平均延迟的最大值。然而，随着平均延迟的不断变化，K 可能变得过大(从而缓冲一些冗余数据，并产生不必要的低效和延迟)或者过小(从而不足以缓冲候选的乱序事件，并且产生错误的输出结果)。在评价事件之前，K-Slack 方法也需要额外的空间，并产生更多的延迟。

另外一种处理乱序事件的方法是利用标志符法(punctuation)。这种技术假设在事件流中直接插入断言(assertions)，用来确保某个特定的值或者时间标签在将来的输入流中不再出现[10,14]。文献[10]中利用这种技术提出了一种保守的解决方法。在假设乱序事件频繁发生的情况下，它仅输出准确性可以保证的结果。偏序确保模型(Partial Order Guarantee，POG)的提出使得这种准确性得到保证。这些技术很有趣，但是它们需要事先产生某些服务，并且适当地插入一些断言。

关于时间段事件，目前已经有很多的研究工作[7,8,13,15]。Kam 等[15]提出一种算法，利用异构的表达方式来发现频繁时态模式。然而，异构的表达方式是存在歧义的，从而产生很多假模式。Papapetrou 等[13]提出了一种算法 H-DFS，用来挖掘时间段的频繁模式。这些研究工作都用 ID 序列号将事件序列转化成列式表达方式。一个事件的 ID 序列与其他事件的 ID 序列合并，从而产生时态模式。但是随着时态模式长度的增加，这种技术不能很好地扩展。Wu 等[7]提出了一种算法 TPrefix，用来从时间段事件中挖掘非歧义的时态模式。TPrefix 首先从映射数据库中发掘单个频繁事件；然后从时态前缀和已经发现的频繁事件中产生所有可能的候选模式；最后再次扫描映射数据库来统计模式的数目。TPrefix 有其本质的局限性，即需要对数据库进行多次扫描，算法没有采用任何过滤策略来减小查询空间。为了克服这些缺陷，文献[8]中给出了一种松散的表达方式来维持事件的底层时态结构，并且提出了算法 IEMiner，用来从时间段事件中发现频繁的时态模式。然而，他们仅仅利用这种表达方式进行分类，而没有考虑乱序事件的问题。

3 预备知识

3.1 时态语义

每个事件都有一个 ID 和两个时间标签。应用时间标签表示事件提供者产生事件的时间；到达时间标签表示事件到达用户(负责处理该事件)的时间。应用时间标签可以被进一步分为有效时间和发生时间[16]。下面介绍物联网中事件时间标签的一些特定属性。

定义 1(时间粒度) 时间粒度是用来描述事件时态约束的粒度，如秒、分、小时、天、星期、月、年等。

在比较它们的大小之前，必须首先选择一个特定的粒度。然而，两粒度之间的比率可能是不固定的。例如，星期和月份，以及工作日和小时之间的比率都是不固定的。

定义 2(时间段) 假设 $H=\{T_1,\cdots,T_n\}$ 是一个线性异构的时间单元集合，其中，对于每个 $1\leqslant i\leqslant n$，都有 $T_i\subseteq T_{i+1}$。例如 $H=\{minute, hour, day, month, year\}$，其中 $minute\subseteq$

$hour$。H 中的某个时间段是一个 n 元组$(t_1,\cdots,t_n)$，其中对于所有的 $1\leqslant i\leqslant n$，t_i 是时间单元 T_i 中的一个时间段。

时间段根据字典序 $<_H$ 进行排序。因此时间段 $T=(t_1,\cdots,t_n)<_H T'=(t_1',\cdots,t_n')$ 当且仅当存在一个 $i(1\leqslant i\leqslant n)$ 使 $t_i<_{T_i} t_i'$，并且对于所有的 $j=i+1,\cdots,n$，都有 $t_j=t_j'$。注意：如果 $i=n$，那么 $t_j=t_j'$。$T<_H T'$ 表示 T 在 T' 之前发生；$T=T'$ 表示 T 和 T' 同时发生。

定义 3(乱序事件)　令 $e.ats$ 和 $e.ts$ 分别表示事件 e 的到达时间标签和发生时间标签。给定一个事件流 $S:e_1,e_2,\cdots,e_n$，其中 $e_1.ats<e_2.ats<\cdots<e_n.ats$。对于 S 中的任意两个事件 e_i 和 $e_j(1\leqslant i,j\leqslant n)$，如果 $e_i.ts<e_j.ts$，并且 $e_i.ats<e_j.ats$，则该事件流是有序的；如果 $e_j.ts<e_i.ts$，并且 $e_j.ats>e_i.ats$，则 e_j 是一个乱序事件。

如图 1 所示的例子中，事件 e_1-e_4 的发生时间是有序的，但是 e_2 的到达时间却比 e_3 晚，这时就产生了乱序。

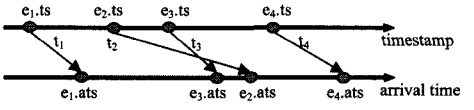


图 1　乱序事件

3.2　查询模式

查询模式指定了个体事件的过滤方法，以及多个事件如何通过时间和值的约束条件相互关联。基于时间段的事件，其查询模式可被分为两大类：串行查询模式和并行查询模式。

3.2.1　串行查询模式

串行查询模式是大多数事件处理系统都支持的基本功能。基于之前事件的顺序，预测事件之间的关联性，这对复合事件检测是很有用的。为了保证效率，包含多个输入事件的、能够产生输出结果的所有操作符有一个时态约束，记作 w。例如，如果满足条件：

(1)对于 $\forall_i\in 1,\cdots,n-1$，都有 $e_i.endtime<e_{i+1}.starttime$；

(2)$e_n.endtime-e_1.endtime\leqslant w$。

则 $seq(e_1,e_2,\cdots,e_n;w)$ 输出一个复合事件，其中 $t_1=e_1.starttime$，$t_2=e_n.endtime$。因此，seq 包含一系列有序发生的、无重叠的事件。

以图 2 所示的老年护理为例。因为老年人往往是比较健忘的，所以应该时刻监控老人的行为，并提醒他们去做某些重要的事情(如吃药)。假设某位老人应该在吃过晚饭后的 30 分钟内服药。如果我们接收到吃晚饭的开始和结束时间、洗漱的开始和结束时间，但是在睡觉的开始时间之前没有接收到服药的开始或者结束时间，这时就会发出预警信号进行提醒。

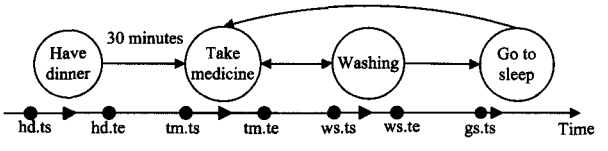


图 2　串行查询模式

3.3.2　并行查询模式

并行查询模式是事件处理系统的另一特征，尤其对于时间段事件更为常见。为了提高性能，人们往往并行地进行复合事件的检测。并行查询模式也有一个时间约束，记作 w。例如，$pal(e_1,e_2,\cdots,e_n;w)$ 输出一个复合事件，其中 $t_1=\min\{e_1.starttime,\cdots,e_n.starttime\}$，$t_2=\max\{e_1.endtime,\cdots,e_n.endtime\}$，并且 $t_2-t_1\leqslant w$。因此，pal 允许事件之间的重叠和无序。并行模式包括合取和析取两个过程。合取表示事件 A 和 B 都在时间约束 w 内发生，与其发生顺序无关；析取表示或者 A，或者 B，或者二者都在时间约束 w 内发生。

如图 3 中的例子所示，假设银行中有一项优惠活动，即满足条件的前 N 个客户可以得到一份大礼包。因为银行中的客户数目是很多的，所以需要几个处理器并行地处理客户的请求信息。而不同处理器的服务速率不同，先申请的客户需求可能反而到达得晚。

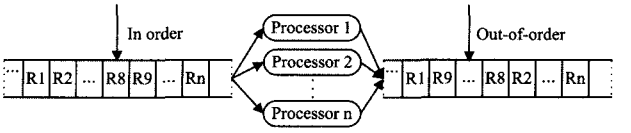


图 3　并行查询模式

4　具有时间持续性特征的乱序事件检测

在介绍了乱序事件的定义及其可能导致的问题之后，本节提出具有时间持续性特征的乱序事件检测方法。

4.1　具有时间持续性特征的事件模型

如第 3 节所介绍的，每个事件可以表示为(ID,Vs,Ve,Os,Oe,Ss,Se,K)。其中 Vs 和 Ve 分别表示事件的有效开始和结束时间；Os 和 Oe 分别表示事件发生的开始和结束时间；Ss 表示事件到达时系统的时钟时间；Se 表示事件结束时的系统时间；K 表示起始的插入和所有相关的撤销操作，每个操作都会根据之前的匹配实体减小 Se 的值。除了时间标签外，事件还可能拥有其他的属性，如价值、价格、名字等。

4.1.1　逻辑表示方式

复合事件的查询模式如下(其中 Event Pattern 通过不同的事件操作将事件关联起来；WHERE 通过事件之间的谓词定义事件模式的上下文；WITHIN 定义了相匹配的事件模式发生的时间范围；Real-time Factor 用来标注不同用户对实时性的需求)：

```
⟨Query⟩::=EVENT ⟨event pattern⟩
    [WHERE ⟨value constraints⟩]
    [WITHIN ⟨time constraints⟩]
    [Real-time Factor {0,1}]
    ⟨event pattern⟩::=SEQ/PAL((Ei(relationship) Ej)
                        (!Ek)(relationship)El))(1≤i,j,k,l≤n)
    relationship::={contain, overlap, before, after, meet, finish,
    equal}
    ⟨time constraints⟩::=Time Window length W
```

图 2 中的老年护理流程可以表示为 *Query* 1。A 表示吃晚饭；B、C 分别表示吃药和洗漱；D 表示上床睡觉。事件 B 和 C 可以并行或交叉执行。

Query 1　Elder Care Processing

EVENT PATTERN	SEQ(A(Before) PAL(B,C)(Before)D)
WHERE	A. Oe<B. Os
AND	A. Oe<C. Os
AND	A. Oe<D. Os
AND	B. Oe>D. Os
AND	C. Oe>D. Os
WITHIN	4 hours
Real-time Factor	1

查询 1 可以被简化成 Q，图 4 表示了查询模式 Q 的一个实例。

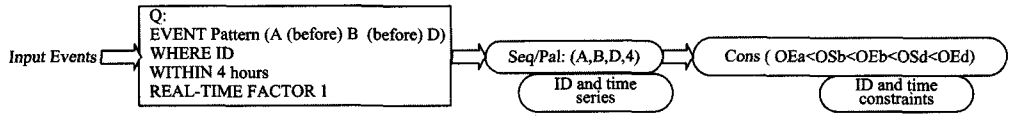


图 4　查询 1 的逻辑表示方式

4.1.2　物理表示方式

目前，有限状态自动机(NFA)是进行复合事件检测最常用的方法[1]。如图 5(a)所示，状态 1 是起始节点，当事件 A 发生时，转换到状态 2；当 B 发生时，转换到状态 3；当 C 发生时，转换到最终的输出状态。当 NFA 转换到最终状态时，就认为该模式是匹配的。然而，由于 NFA 严格的顺序状态转换，传统的基于 NFA 的方法不能直接有效地模拟否定事件和并行事件。在此，扩展了传统的 NFA 表示方法，并将查询 $Q1$ 表示成图 5(b)。当事件 B 或 C 发生时，就会产生中间状态 2 和 3。状态 4 是表示模式匹配的最终状态。然而，由于事件 B 与 C 之间存在谓词 $B.price<C.price$，当事件 B 到达时，因为其相对应的事件 C 可能还没有到达，所以很难判断其状态转换。因此这种扩展的 NFA 方法也无法处理带有谓词的并行查询模式，而这种模式在时间段事件中是很常见的。

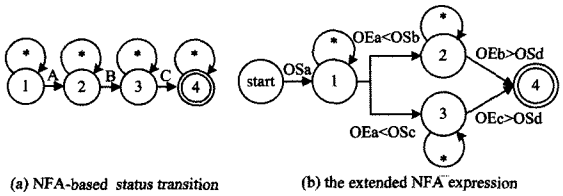


图 5　基于 NFA 的表示方法

对于时间段事件的模式查询，其状态转换不仅依赖于事件类型，而且取决于事件之间的关系和谓词。本文提出了一种基于树的查询模型用于查询模式的逻辑和物理表示。为了处理查询，首先将查询转化成一种树形结构。其中叶子节点存储到达的原子事件，中间节点存储子树得到的中间结果。每个中间节点都对应查询中的一个操作符以及一系列谓词。图 6(a)表示查询 $Q1$ 的树模型结构。这是一棵左深树，因为 A 和 Pal 首先结合，其输出结果再与 D 匹配。同理，Pal 和 D 首先结合，然后再与 A 匹配的右深树也是可行的。

树中的每个节点都有一个栈，分别用来存储到达的原子事件(对于叶子节点)和中间结果(对于中间节点)。每个栈都包含一系列记录值，每条记录都有一个事件指针，分别指向该事件的开始和结束时间。对于栈中每个实例的开始时间，都有一个额外的值 $PreEve$ 来记录前一个状态栈中的、按照时间序列排列的最邻近的一个实例(如算法 1)。对于每个实例的结束时间，$PreEve$ 首先指向其对应的开始时间。当其开始时间成为其他实例的 $PreEve$ 时，其 $PreEve$ 就改为指向该实例。例如，图 6(b)中，$OEa(7)$ 的 $PreEve$ 首先设定为 $OSa(3)$，而类型 A 的 Sa 栈中距离 $OSb(6)$ 最近的实例是 $OSa(3)$，所以 $OSb(6)$ 的 $PreEve$ 就指向 $OSa(3)$，而 $OEa(7)$ 的 $PreEve$ 改为指向 $OSb(6)$。中间节点的开始和结束时间是组成该复合事件的所有原子事件的最早开始时间和最晚结束时间。

算法 1　栈的存储模式

Input: The number of events in the stack, N;
　　The occurrence start time of the event, OS;
　　The occurrence end time of the event, OE;
　　One of the events that is already stored in the stack, k;
Output: The correct TList of the stack;

```
string[] A;
1. for i=0; i<N; i++ do
2.   for j=1; j<=N; j++ do
3.     if A[i].OE has arrived and there is no k satisfying the prior
       pointer of A[k] is A[j].OS then
4.       the prior pointer of A[j].OS is set to A[i].OE;
5.       the prior pointer of A[j].OE is set to A[j].OS;
6.     else if A[i].OE has arrived and there is a k satisfying the prior
       pointer of A[k] is A[j].OS then
7.       the prior pointer of A[j].OS is set to A[i].OE;
8.       the prior pointer of A[j].OE is set to A[k];
9.     else if A[i].OE has not arrived and there is no k satisfying the
       prior pointer of A[k] is A[j].OS then
10.      the prior pointer of A[j].OS is set to A[i].OS;
11.      the prior pointer of A[j].OE is set to A[j].OS;
12.    else
13.      the prior pointer of A[j].OS is set to A[i].OS;
14.      the prior pointer of A[j].OE is set to A[k];
15.    end if
16.  end for
17. end for
```

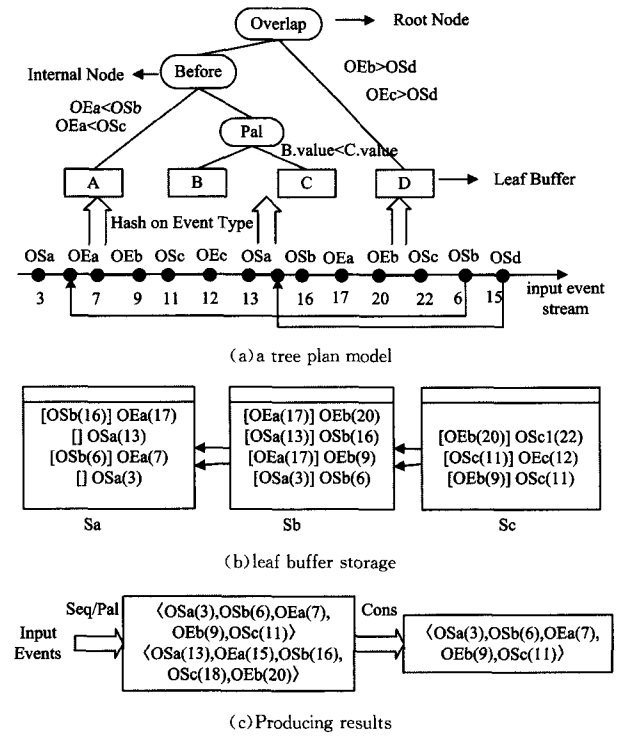


图 6　基于树的物理表示方法

图 6(b)表示树模型中每个节点的栈存储。在每个栈中，各个实例按照其到达的时间顺序自上而下存储(如算法 2)。对于有序事件，每个到达的事件只是简单地存放于相应栈的底部，其 $PreEve$ 指向前面栈的最后一个事件。然而，这种简单的添加措施并不适用于乱序事件的插入。一个乱序事件 $e_i \in E_i$ 被存放于类型为 E_i 的相应栈中，并且按照到达时间进行排序。如果类型为 E_k 的栈中的所有事件的结束时间都已

经到达，并且事件 e_k 的开始时间 $e_k.starttime>_H e_i.starttime$ ($e_i.endtime$)，那么 $e_k.starttime$ 的 *PreEve* 将指向 $e_i.starttime$($e_i.endtime$)；否则，将等待缺失的结束时间。如果其开始时间已经到达，那么事件 e_k 的结束时间 $e_k.endtime$ 的 *PreEve* 将指向其对应的开始时间 $e_k.starttime$。例如，图 6(c)中，假设存在乱序事件 *OSb*(8)和 *OEb*(10)。在 *OEb*(9)到达之前，*OSb*(8)的 *PreEve* 不能被设置为 *OSa*(3)，因为在时刻 8 之前，只收到事件 *B* 的开始时间，而没有结束时间。同理，*OEb*(10)的 *PreEve* 设置为 *OSb*(8)，因为其对应的开始时间已经到达。

算法 2 栈的插入操作

```
Input: The current number of events in the stack, N;
       The occurrence start time of the event, OS;
       The occurrence end time of the event, OE;
       The new received event ID, k;
       The maximal size of the stack, MS;
Output: The correct TList of the stack;
1. While k<MS do
2.   if all events in the stack are in order then
3.     the prior pointer of A[k]. OS is set to A[N]. OE;
4.     the prior pointer of A[k]. OE is set to A[k]. OS;
5.   else if there are out-of-order events in the stack, and the end time
     of all events have arrived then
6.       for i = N-1; i>=0; i--do
7.         if A[k]. OS>A[i]. OS then
8.           the prior pointer of A[k]. OS is set to A[i]. OS;
9.         else
10.          the prior pointer of A[k]. OS is set to A[i]. OE;
11.          the prior pointer of A[k]. OE is set to A[k]. OS;
12.        end if
13.      end for
14.  else
15.      wait until there is no absent end time;
16.  end if
17. end while
```

4.2 具有时间持续性特征的乱序事件检测方法

为了检测基于时间段的乱序事件，需要定义一个变量 Sync(a synchronization point，同步点)。表 1 引入了包含该变量的列用来描述引入注释的历史表格。变量 *Sync* 的计算方式如下：对于插入操作，$Sync=O_s$；对于删除操作，$Sync=O_e$。变量 *Sync* 的本质是引入了乱序事件的全局定义。即如果一个流中不存在乱序事件，当且仅当按照 S_s 进行排序的事件的全局次序与按照混合关键字$\langle Sync, S_s\rangle$进行排序的事件的全局次序完全一致。

表 1 历史注释表

K	Sync	O_s	O_e	S_s	S_e
E_0	1	1	10	0	7
E_0	5	1	5	7	10

5 具有时间持续性特征的乱序事件解决方案

5.1 基本框架表示

所提方法的基本框架结构如图 7 所示，主要包含 3 个组件。终端层包括移动设备、智能电话、PDA、PC 等，是原始数据的来源。这些来自不同数据源的事件都是有序的。事件处理引擎存储来自终端层的事件，并做一些查询处理。在终端层向事件处理引擎的传输过程中，网络延迟或机器故障经常导致事件发生乱序。终端层到事件处理引擎的数据传输有两种方式：推送式和自取式，其详细介绍见 5.3 节。数据管理存储事件的历史记录、它们之间的关系，以及一些知识和规则。数据管理的本质是历史事件记录和实时事件记录的结合体。此外，数据管理应保存一个知识库，包括一些外围信息，比如空间位置信息和某些地方的可能行为等。数据管理中所存储的关系反映了来自终端层的原子事件之间的关系。在传统的关系模型中，关系仅仅是一系列元组，不包含任何时间和语义的信息。数据管理中的关系既包含了传统的元组信息，又包含了元组的时间因素。

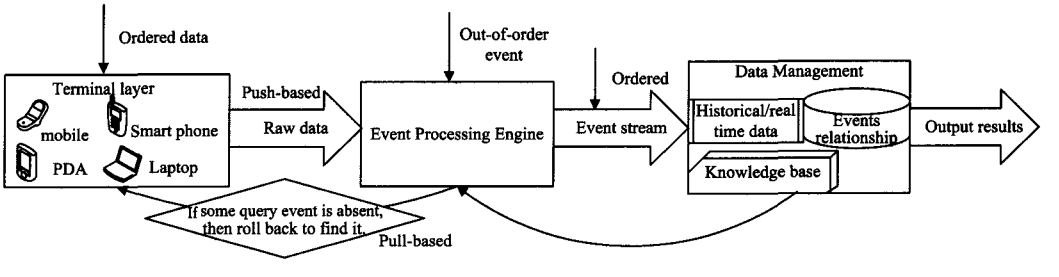


图 7 具有时间持续性特征的乱序事件解决框架

除了框架结构之外，特定的查询表达式也是一个挑战性的问题。与点事件查询不同的是，时间段事件查询的表达式存在歧义性。这种歧义性会造成事件之间关系的不正确性。

例如给定表达式(*A* Overlap *B*) Overlap *C*，据此无法推断 *C* 仅仅与 *B* 重叠，或者 *C* 同时与 *A*、*B* 重叠。图 8 给出了这种时态模式的不同解释，这些不同的解释就会导致对具有确切关系的事件的错误推断。为了解决这个问题，文献[8]提出了一种无损耗的表示方法，它是在分层表示法的基础上引入了一些附加信息，即用 5 个变量(包含数目 c，结束数目 f，相交数目 m，重叠数目 o，开始数目 s)来区分所有可能的情况。因此，复合事件 *E* 可表示为：

$$E=(E_1\ R_1[c,f,m,o,s]E_2)R_2[c,f,m,o,s]E_3)\cdots R_{n-1}[c,f,m,o,s]E_n)$$

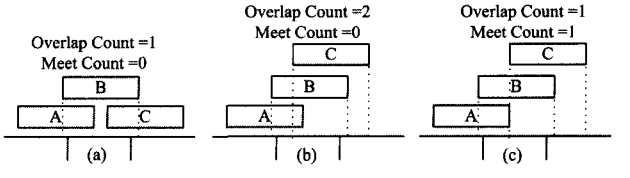


图 8 模式(*A* Overlap *B*) Overlap *C* 的不同解释

图 8 中的时态模式可分别表示如下：

(*A* Overlap[0,0,0,1,0] *B*) Overlap[0,0,0,1,0] *C*

(*A* Overlap[0,0,0,1,0] *B*) Overlap[0,0,0,2,0] *C*

$(A\ \text{Overlap}[0,0,0,1,0]\ B)\ \text{Overlap}[0,0,1,1,0]\ C$

现实世界中，不同应用对一致性有不同的需求。有些应用对准确性有严格的要求，而有些应用却更关注于实时性。如果用户能够根据不同的应用场景为每个查询指定不同的需求，那么系统就可以在运行中实时地调整一致性需求。因此，为每个查询添加了一个属性“Real-time Factor”。如果用户更关注实时性，“Real-time Factor”就设置为 1；否则就设为 0。鉴于用户对一致性的不同需求，将介绍两种不同的解决方案。

5.2 实时的解决方案

如果一个查询的“Real-time Factor”设置为 1，其目标就是尽可能地减少延迟，尽快地输出结果。这时假设大部分数据都能够按时有序地到达。一旦出现乱序的情况，将提供一种补救措施来更正之前已经输出的结果。这种方法类似于文献[10]，但是效率更高。因为树模型的表示方法可以减少补偿的时间和频率。

当用户提交查询到“事件缓冲区”后，“事件缓冲区”处理该查询，并直接输出相应的结果。这种方法保证了实时性的需求，从而能够及时地采取某些应急措施。但是，一旦出现乱序事件，就会出现这样的情况：或者之前的输出结果是错误的，或者遗漏了正确的结果。为了补偿，引入了两种类型的流信息。当出现乱序的肯定事件时，需要引入 Insertion〈+，E〉，其中 E 是一个新的输出结果。当出现乱序的否定事件时，需要引入 Deletion〈−，E〉，其中 E 表示一个之前输出的结果。Deletion 操作撤销由于乱序否定事件的出现而已经输出的、无效的结果。

例如，如果查询是(A(overlap) B(!D)(before) C)，其时间约束是 10 分钟。那么根据之前介绍的时间段表示方法，可以得到该查询的一个唯一的时间序列{OSa,OSb,OEa,OEb,OSc,OEc}。对于图 9 中的事件流，当一个乱序的 *seq/pal* 事件 $OSb(6)$到达时，一个新的正确结果{$OSa(3)$,$OSb(6)$,$OEa(7)$,$OEb(9)$,$OSc(11)$,$OEc(12)$}将被输出〈+，{$OSa(3)$,$OSb(6)$,$OEa(7)$,$OEb(9)$,$OSc(11)$,$OEc(12)$}〉。当乱序的否定事件 $OSd(15)$到达时，发现一个错误的输出结果{$OSa(13)$,$OSb(16)$,$OEa(17)$,$OEb(20)$,$OSc(22)$}，并发出一条补救信息〈−，{$OSa(13)$,$OSb(16)$,$OEa(17)$,$OEb(20)$,$OSc(22)$}〉。

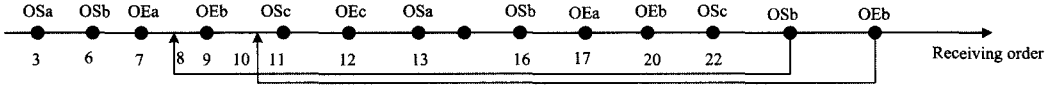


图 9 乱序事件流

当乱序事件发生时，这些补救操作也应该存储到“数据管理中心”。这样经过一段时间后，“事件缓冲区”产生的查询结果应该首先发送到“数据管理中心”，根据“数据管理中心”中的历史记录和知识规则对结果进行重新检查。如果该结果和“数据管理中心”的内容是正相关的，就输出最后的结果；否则，该结果就需要缓冲一段时间。缓冲的时间长短取决于不同的应用场景。这种优化方法虽然消耗了一定的时间，但是减少了大量的错误结果和补偿操作。从全局来看，其性能优于现有的方法。

5.3 基于准确性的解决方案

如果某查询的 “Real-time Factor”设置为 0，其目标就是保证输出的结果都是正确的，而不是太计较延迟。将时间段事件考虑在内，该解决方案描述如下。

5.3.1 查询处理

利用之前介绍的框架结构，当用户提交查询到“事件缓冲区”时，首先将查询抽取成相应的串行/并行事件模式。然后，基于第 4 节介绍的事件模型，通过向前或向后深度优先搜索的方法，可以得到一个事件序列。向前搜索是从当前实例 e_i 的开始时间开始，遍历从 e_i 出发的所有边。向后搜索是从接收状态事件实例的结束时间开始，遍历到达事件 e_i 的所有路径，也包括事件 e_i。一个最终的从根到叶子的路径包含新事件 e_i 和一个匹配的事件序列。如果 $e_i.endtime$(或 $e_i.starttime$) 属于接收状态(或开始状态)，则计算过程只需要用向后(或向前)搜索的方法。

5.1 节介绍的五元组可以保证事件之间关系的表达方式的唯一性。基于该五元组，可以将查询转换成一个特定的时间序列。通过与查询的时间序列进行比较，可以进一步地过滤事件序列集合，例如，不同事件的开始时间和结束时间之间的关系约束、时间窗口的限制，以及事件序列中否定事件的约束。经过这些步骤之后，把得到的事件结果存储于“数据管理中心”的缓冲区中。

“数据管理中心”的缓冲区利用 K-ISlack 策略进行事件的缓冲和清洗。与传统的 K-Slack 方法不同的是，本文考虑了时间段。该方法假设所有乱序事件的开始时间和结束时间都在 K 个时间单元内到达，也就是说，该事件最多延迟 K 个时间单元。缓冲区比较被检测事件与系统接收到的最后一个事件之间的距离。设置一个时钟变量($CLOCK$)，其值等于目前接收到的所有事件的最大到达时间。$CLOCK$ 的值是不断更新的。对于缓冲区中的任何一个事件实例 e_i，根据滑动窗口的时间约束，如果$(e_i.starttime+W)<CLOCK$，那么就将该事件从栈中滤掉。当考虑到乱序事件时，上面的约束条件就变为$(e_i.starttime+W+K)<CLOCK$，这是因为当等待 K 个时间单元后，所有开始时间小于$(e_i.starttime+W)$的乱序事件将不会再到达。因此，e_i 将不会再组成新的候选序列。

5.3.2 优化策略

为了更好地优化该解决方法，根据时间窗口的约束将缓冲区分为两部分：过期的事件实例和更新的事件实例。在缓冲区中设置一个隔离带，隔离带之上的实例是过期的，之下的是更新的。过期的事件实例存储的是已经超过时间窗口约束的事件序列；而更新的事件实例存储的是时间窗口范围内的事件序列。对于没有过期事件的栈，该隔离带设置为 NULL。有序事件能够触发序列的重组，这时只考虑每个栈中隔离带之下的事件。

另外，当乱序事件发生时，可能会引起某些属性的延迟，例如，一个事件的结束时间已经到达，但是还没有接收到其开始时间。如图 7 所示，对缺失的属性采取一些积极的行动，而不是一味地等待。如果“事件处理引擎”中没有发现该属性，就直接返回数据源查看该缺失值是否已经发生。如果数据源

中没有该属性，则相应的查询结果被直接输出或丢弃；否则，“事件处理引擎”继续等待，直到该属性到达，再将结果通过“数据管理中心”输出。如 5.1 节介绍的，“数据管理中心”存储了历史记录和知识规则，从而在输出之前可以过滤掉一些错误的结果。

6 实验结果与分析

为了测试和验证方法的有效性，设计了一个实验来仿真事件的产生和查询。该模拟器是用 C# 语言实现的。

6.1 实验环境

实验环境包含两部分：一个是事件发生器，一个是事件处理引擎。事件发生器用来连续地产生不同类型的事件。采用多线程的机制来模拟不同的传感器，从而随机地产生不同类型的事件。事件处理引擎包含两部分：接收单元和处理单元。接收单元负责接收来自传感器的各类事件；处理单元负责事件的查询，并输出正确的结果。同时，处理单元还负责报告如表 2 所列的各种性能参数。

表 2 参数和性能指标

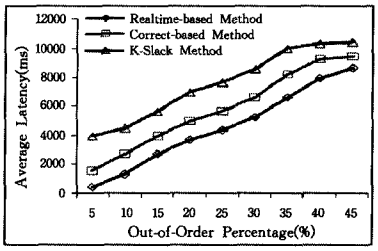
参数	定义和描述
P_{io}^3	乱序事件所占的比率
Buf	树模式的缓冲区大小
QL	事件的查询长度
NoR	输出结果的数目
NoC	需要补偿的结果数目
NoCR	正确结果的数目
K	乱序事件的最大延迟
AET	平均执行时间
AL	平均延迟
RoC	补偿率，NoC/NoR
ACC	结果的正确率，NoCR/NoR

实验运行在两台机器上，其 CPU 均为 2.0GHz，内存分别为 4.0GB 和 8.0GB。机器 PC1 用作事件发生器，PC2 用作事件处理引擎。在 PC1 中，模拟了大约 1000 个发生器（传感器），每个事件发生器可以随机地产生 1000 多种不同类型的事件。因此至少 1000000 个事件将到达 PC2 中的接收单元，并等待查询处理。基于如此大规模的事件数据，我们的实验可以更好地测试和验证算法的性能。为了使实验结果更可信，运行实验 300 次，并取所有结果的平均值。在下面的实验中，将主要关注表 2 所列的各项性能指标。

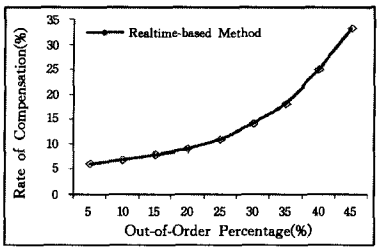
6.2 性能评测

图 10—图 12 主要测试了乱序率 P_{io}^3 对各项性能指标的影响。P_{io}^3 的调整范围是 0 到 45%。图 10(a)表示的是没有时间段事件到达的情况，由该图可知，随着乱序率的增加，3 种方法（实时性解决方法、准确性解决方法、K-Slack 方法）的平均延迟增加，并且实时性解决方法的增加速度比其他两种的都快，这是因为将补偿操作的代价也加入到了平均延迟中。当有时间段事件存在时，原始的 K-Slack 方法就不适用了，而实时性解决方法和准确性解决方法的变化趋势与图 10(a)所示的类似。

图 10(b)只与实时性解决方法有关，因为只有实时性解决方法才有补偿操作。补偿率定义为 *NoC/NoR*。由图 10(b)可知，随着乱序率的增加，产生越来越多的补偿操作，并且补偿率的增加速度越来越快。



(a)平均延迟的变化趋势

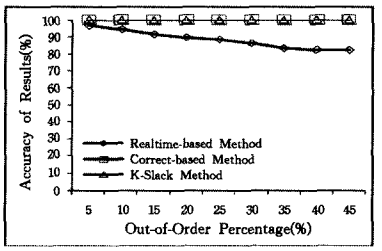


(b)补偿率的变化趋势

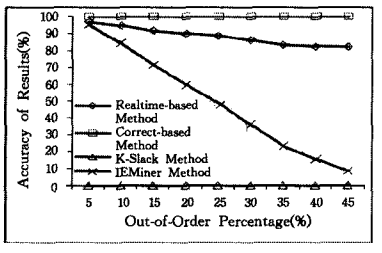
图 10 乱序率对平均延迟和补偿率的影响

输出结果的准确性定义为 *NoCR/NoR*。图 11(a)显示的是在没有时间段事件的情况下 3 种方法的准确性。这时准确性方法和 K-Slack 方法的准确性都与乱序率无关，而实时性方法的准确性随着乱序率的增加而降低。这是因为乱序率越高，就有越多的输出结果需要补偿和更改。

图 11(b)显示的是有时间段事件的情况下 4 种方法（实时性方法、准确性方法、K-Slack 方法和 IEMiner 方法）的准确性。这时 K-Slack 方法的准确性几乎为零，因为它不能处理乱序的时间段事件。随着乱序率的增加，IEMiner 方法的准确性急速下降，因为它只能处理时间段事件，而不能处理乱序事件。实时性方法和准确性方法的准确性变化趋势与图 11(a)类似。



(a)时间点事件



(b)时间段事件

图 11 乱序率对准确性的影响

图 12 测试的是平均执行时间，反映各操作符执行时间的总和。当没有时间段事件时，可以观察到两点：(1)随着乱序率的增加，平均执行时间增加，因为有更多的输出结果需要重新计算；(2)准确性方法的平均执行时间起初大于实时性方法的，但随着乱序率的增加，它们逐渐趋于一致。K-Slack 方法的执行时间总是大于其他两种方法的。如果存在时间段事

件，K-Slack 方法的执行时间将趋向于无穷大，而实时性方法和准确性方法的趋势几乎不变。

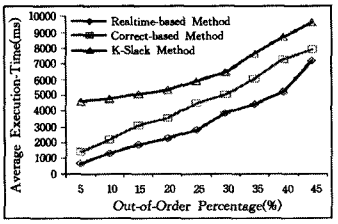


图 12　平均执行时间的变化趋势

图 13—图 15 测试的是缓冲区大小对各项性能指标的影响。图 13 显示随着缓冲区的增大，两种方法的平均延迟逐渐减少。当树模型中的缓冲区大小小于 500eventnumber 时，准确性方法的平均延迟小于实时性方法；当缓冲区大于 500eventnumber 时，反之。也就是说，实时性方法的减少速率比准确性方法的快，即缓冲区大小对实时性方法影响更大。因为当缓冲区太小时，必然输出了很多错误的结果，从而导致很多补偿操作，增加延迟。当缓冲区足够大时，需要补偿的输出结果迅速下降，因此准确性方法的平均延迟又大于实时性方法。

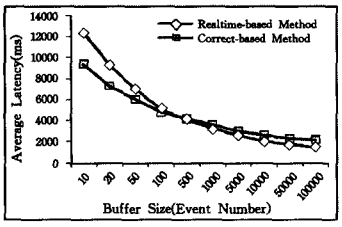
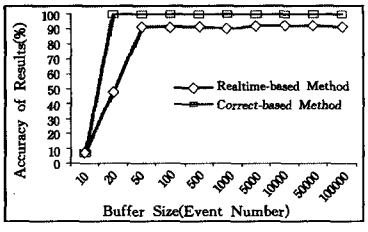


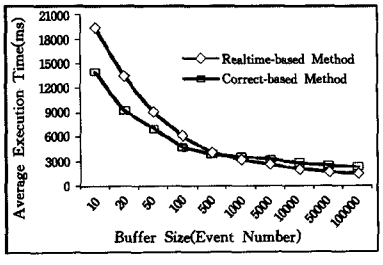
图 13　平均延迟的变化趋势

图 14(a)显示的是缓冲区大小不同时两种方法的准确性变化趋势。当缓冲区大小趋近于 0 时，几乎没有输出结果，因此两种方法的准确性都几乎为 0。当缓冲区稍微大些时，两种方法的准确性立即增加。因为总是能够从“数据管理中心”获取之前的事件，所以随着缓冲区的增大，两种方法的准确性都基本一致。也就是说，缓冲区大小对准确性影响很小。

图 14(b)测试的是平均执行时间的变化情况，其趋势和平均延迟类似。它们之间仅存在一个常数差异，即从第一个事件的到达时间到相应的最后一个事件的到达时间之差。



(a)准确性

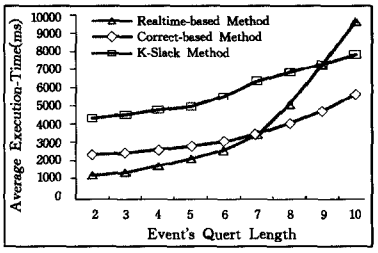


(b)平均执行时间

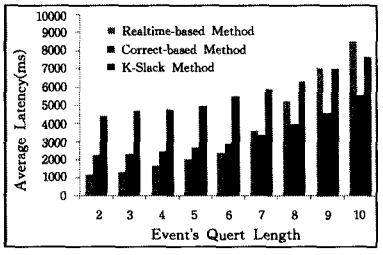
图 14　缓冲区大小对准确性和执行时间的影响

图 15(a)显示的是没有时间段事件的情况下，事件查询长度对平均执行时间的影响。由图可知，其变化趋势可划分为两部分。当事件查询长度很小时，准确性方法和 K-Slack 方法的平均执行时间大于实时性方法的；随着查询长度的增加，它们逐渐趋于相等；然后实时性方法的平均执行时间变大。这是因为当事件查询长度太长时，实时性方法中必然有很多补偿操作。K-Slack 方法的平均执行时间总是大于准确性方法的。与实时性方法相比，事件查询长度对准确性方法和 K-Slack 方法的影响较小。当存在时间段事件时，K-Slack 方法的执行时间趋向于无穷大。

图 15(b)显示的是没有时间段事件的情况下，随着事件查询长度的增加，3 种方法的延迟增加。由图可知，实时性方法大于其他两种方法的增加速度，并且 K-Slack 方法总是大于准确性方法的延迟。当存在时间段事件时，K-Slack 方法的延迟趋向于无穷大，因为它不能处理时间段事件。



(a)执行时间



(b)平均延迟

图 15　事件长度对执行时间和平均延迟的影响

结束语　本文的目的是解决物联网中具有时间持续性特征的乱序事件查询处理问题。文中提出了一种基于树结构的、具有时间段特征的乱序事件逻辑表示与物理描述模型。在综合考虑了事件时间持续性与乱序问题的基础上，建立了一种解决时间段乱序事件的混合处理方法，可以根据不同的应用场景，选择不同的事件输出策略：1)准确性优先；2)实时性优先。通过与 K-Slack 方法和 IEMiner 方法的实验比较，得出本文所提方法可以准确、有效地处理具有时间持续性特征的乱序事件查询处理问题。

本文主要考虑了用户指定输入事件的模式。未来的工作中，将考虑从时间段乱序事件中自动地发现和挖掘复合事件模式。另外，将考虑“数据管理中心”的查询优化策略和算法，将查询频率看作性能指标的又一个影响因子。

参 考 文 献

[1] Mei Y, Madden S. ZStream: a cost - based query processor for adaptively detecting composite events[C]// Proceedings of the 35th SIGMOD International Conference on Management of Da-

ta. SIGMOD,2009:193-206

[2] Antunes C, Oliveira A L. Generalization of pattern growth methods for sequential pattern mining with gap constraints[M]. Machine Learning and Data Mining in Pattern Recognition. 2003:239-251

[3] Pei J, Han J, Mortazavi B, et al. Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth[C]// Proceedings of the 17th International Conference on Data Engineering(ICDE). 2001:215-226

[4] Wu E, Diao Y, Rizvi S. High performance complex event processing over streams[C]//Proceedings of the 32th SIGMOD International Conference on Management of Data. SIGMOD, 2006:407-418

[5] Alex D, Robert R, Subrahmanian V S. Probabilistic temporal databases[J]. ACM Transaction on Database Systems, 2001, 26(1):41-95

[6] Barga R S, Goldstein J, Ali M, et al. Consistent streaming through time: a vision for event streamprocessing[C]// The 3rd Biennial Conference on Innovative Data Systems Research. IDAR, 2007

[7] Wu S, Chen Y, Mining nonambiguous temporal patterns for interval-based events[J]. IEEE Transactions on Knowledge and Data Engineering, 2007, 19(6):742-758

[8] Patel D, Hsu W, Lee M L. Mining relationships among interval-based events for classification[C]//Proceedings of the 34th SIGMOD International Conference on Management of Data. SIGMOD, 2008:393-404

[9] Hammad M A, et al. Scheduling for shared window joins over data streams[C]// The 29th International Conference on Very Large Data Bases. 2003, 29:297-308

[10] Liu M, Li M, Golovnya D, et al. Sequence pattern query processing over out-of-order event streams[C]// Proceedings of the 25th International Conference on Data Engineering (ICDE). 2009:274-295

[11] Eyerman S, Eeckhout L, Karkhanis T, et al. A mechanistic performance model for superscalar out-of-order processors[J]. ACM Transactions on Computer Systems(TOCS), 2009, 27(2): 824-833

[12] Wang F, Liu S, Liu P. Complex RFID event processing[J]. The International Journal on Very Large Data Bases(VLDBJ), 2009, 18(4):913-931

[13] Papapetrou P, Kollios G, Sclaroff S, et al. Mining frequent arrangements of temporal intervals[J]. Knowledge & Information Systems, 2009, 21(2):133-171

[14] Ding L, Mehta N, Rundensteiner E A, et al. Joining punctuated streams[M] // Advances in Database Technology (EDBT). 2004:587-604

[15] Kam P S, Fu A W. Discovering temporal patterns for interval-based events[C]// Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery. 2000: 317-326

[16] Barga R S, Goldstein J, Ali M, et al. Consistent streaming through time: a vision for event stream processing[C]// The 3rd Biennial Conference on Innovative Data Systems Research. 2006:363-374

[17] Zhou C J, Meng X F. A framework of complex event detection and operation in pervasive computing[C]//The Third SIGMOD PhD Workshop on Innovative Database Research. 2009

(上接第 178 页)

[9] Yuan P S, Sha C F, Wang X L, et al. c-Approximate nearest neighbor query algorithm based on learning for high-dimensional data[J]. Journal of Software, 2012, 23(8): 2018-2031(in Chinese)
袁培森,沙朝锋,王晓玲,等. 一种基于学习的高维数据 c-近似最近邻查询算法[J]. 软件学报,2012,23(8):2018-2031

[10] Lu Yan-sheng, He Ya-jun, Pan Peng. Improvement of the EINN Algorithm for NN Query Traversing[J]. Computer Engineering and Science, 2005, 27(7):62-64(in Chinese)
卢炎生,何亚军,潘鹏. EINN 最近邻居查询索引遍历算法改进[J]. 计算机工程与科学,2005,27(7):62-64

[11] Li Song, Zhang Li-ping. Simple Continues Near Neighbor Chain Query in Dynamic Constrained Regions[J]. Computer Science, 2014, 41(6):136-141(in Chinese)
李松,张丽平. 动态受限区域内的单纯型连续近邻链查询方法[J]. 计算机科学,2014,41(6):136-141

[12] Gao Y, Zheng B. Continuous obstructed nearest neighbor queries in spatial databases[C]// Proceedings of the 28th ACM SIGMOD International Conference of Management of Data(Sigmod'09). New York, USA, 2009:577-590

[13] Nutanong S, Tanin E, Zhang Rui. Incremental evaluation of visible nearest neighbor queries[J]. IEEE Transactions on Knowledge Engineering, 2010, 22(5):665-681

[14] Gao Y, Yang J C, Chen G, et al. On efficient obstructed reverse nearest neighbor query processing[C]// Proc. of the 19th Int'l Conf. on Advances in Geographic Information Systems. Chicago: ACM Press, 2011:191-120

[15] Li Chuan-wen, Gu Yu, Qi Jian-zhong, et al. A safe region based approach to moving KNN queries in obstructed space[J]. Knowledge and Information Systems, 2014, 24(3):179-195

[16] Wang Yan-qiu, Zhang Rui, Xu Chuan-fei, et al. Continuous visible k nearest neighbor query on moving objects[J]. Information Systems, 2014, 44(8):1-21

[17] Yu Xiao-nan, Gu Yu, Zhang Tian-cheng, et al. A Method for Reverse k-Nearest-Neighbor Queries in Obstructed Spaces[J]. Chinese Journal of Computers, 2011, 34(10): 1917-1925(in Chinese)
于晓楠,谷峪,张天成,等. 一种障碍空间中的反 k 最近邻查询方法[J]. 计算机学报,2011,34(10):1917-1925