数据流上复杂事件处理系统 Eagle 的设计与实现*)

魏永超　陈立军

(北京大学计算机科学与技术系　北京 100871)

摘　要　近年来,随着传感器网络和 RFID 技术的发展,基于数据流的应用已经无所不在。数据流上的应用大多是监控型的,并且这些监控任务一般是简单事件的组合,但简单的数据流处理系统(DSMS)一般只能处理简单事件的过滤和聚合,缺乏复杂事件概念的支持,因此需要构建数据流上的复杂事件处理系统(CEP)来满足这类需求。本文将介绍基于数据流处理系统 Argus 的复杂事件处理系统 Eagle 的设计与实现。我们提出了事件代数来描述复杂事件,并基于此事件代数提出了复杂事件查询语言。我们使用基于查询计划的方式来进行事件探测,并采用了一些优化策略来提高复杂事件探测的效率。

关键词　复杂事件处理,事件代数,事件探测,数据流

Eagle: A Complex Event Processing System over Data Streams

WEI Yong-chao　CHEN Li-jun

(Department of Computer Science & Technology, Peking University, Beijing 100871, China)

Abstract　Recently, with the development of sensor network and RFID technology, data stream processing technology is being applied in many fields. The task of such application is monitoring the complex events which are combination of simple events. But simple data stream processing system (DSP) can only deal with simple filtering and aggregation. Being lack of support for detection of complex events make it necessary to build the complex event processing system (CEP). We will present the design and implementation of the complex event processing system-Eagle, which is based on the data stream processing system-Argus. First, we design complex event algebra to express complex event, and propose complex events query language based on it. Then we describe a query plan-based approach to detect complex event efficiently. We also design some optimization strategies to improve the efficiency of complex event detection.

Keywords　Complex event processing, Event algebra, Event detection, Data stream

1　简介

对实时数据的监控在金融、电信和工业界有着广泛的应用。这些数据通常具有实时、无限、有序、时变等流的特征,因此称为数据流。传统的数据库管理系统对这类应用束手无策,为了处理这类需求,数据流处理系统应运而生了。

近年来,随着传感器网络和 RFID 技术的发展,越来越多的领域使用它们进行称为事件监测的实时监视[1],对应的数据流称为事件流。这些应用通常包括供应链管理[8]、图书馆中的追踪[10]、环境监测[9]等。在这类应用中,人们关心的不再是简单事件[2],而是它们的组合-复杂事件[2],从这些组合中用户能够探测到他们需要的信息。由于复杂事件是简单事件组成的序列,会包含时序、不发生等各种情况,传统的数据流处理系统不能够处理这类应用。因此,需要开发数据流上的复杂事件处理系统来支持此类应用。

目前,不少研究机构对数据流上的复杂事件处理展开了研究,并取得了很多成果。SASE[4]提出了复杂事件处理语言,能够支持事件流上的滑动窗口,使用基于查询计划的方式来探测复杂事件,并提出了一些优化策略。Cayuga[1]提出了 Cayuga 事件代数,并提出了 Cayuga 自动机,使用 Cayuga 自动机来进行复杂事件探测,并通过合并状态来实现性能优化。RCEDA[5]提出了针对 RFID 事件流这个特殊应用领域的复杂事件处理,采用事件图的方式进行复杂事件探测。

Argus[3]是北京大学数据库实验室自主研发的数据流处理系统。目前它能够支持数据流上的类

*)本文受到国家高技术研究发展计划(863 计划)"基于蜜场技术的大规模网络主动安全防护系统"课题资助(课题编号 2006AA01Z445)。

SQL 查询和滑动窗口及跳动窗口模型，并能够根据负载自适应地调度算子，但是尚不能支持对复杂事件的监测。在这篇文章里，我们将介绍如何基于 Argus 实现复杂事件处理引擎 Eagle。本文第 2 节介绍了 Eagle 的复杂事件代数，第 3 节给出 Eagle 的系统框架，第 4 节将描述基于查询计划的事件探测方法并给出一些优化的策略和考虑，第 5 节给出了实验，比较了优化前后系统的效率。最后给出结论和将来的工作。

2 Eagle 事件代数

在这一部分，我们将介绍一下用来描述和表达复杂事件查询的事件代数。

2.1 基本概念

复杂事件处理系统的输入是一个或多个无穷事件序列，我们称之为事件流。事件是在一个时间点瞬间发生的有意义的事情。每个事件流都有一个固定的关系模式，而事件流上的每个事件都有时间戳，在这里，我们使用与 SnoopIB[7] 一样的策略，为每个事件分配两个时间戳，分别为起始时间和结束时间。我们如下定义事件流：

定义 1 一个事件流 E 是一系列（可能是无限的）满足特定关系模式 $R(A_1, A_2, \cdots, A_m)$ 的事件元组的集合，这些事件元组形如 $\langle H; begin, end\rangle$，其中 $H=\{e_1, e_2, \cdots, e_n\}$ 是构成此事件元组的事件的集合，其中每一个事件 *begin* 代表事件元组起始时间戳，*end* 代表事件元组的结束时间戳。

本文中我们假设简单事件的时间戳是事件进入 Argus 系统的时候系统为其分配的，而且起始时间戳和结束时间戳相同。除此之外，我们假设所有的事件元组是完全有序的，也即没有两个事件是同时发生的。

2.2 事件代数

Eagle 系统的事件代数包括功能操作符、事件操作符和聚集操作符，下面将详细介绍各个类别的操作符。

2.2.1 功能操作符

选择操作符的符号为 σ_θ，其中 θ 代表选择条件。选择条件可以是一个或者多个形如 a_1 cmp a_2 的布尔表达式的任意组合，a_1 和 a_2 为属性或者常量的运算结果，cmp 为 $<, <=, >, >=, =$ 中的一个。假设 E 为一个事件流，那么 $\sigma_\theta(E)$ 的结果是 E 中满足 θ 的事件组成的一个新的事件流。

投影操作符的符号为 Π_τ，其中 τ 是一系列属性的集合，这个集合必须是事件的关系的一个子集。假设 E 为一个事件流，其关系模式为 $R(A,B)$，那么 $\Pi_\tau(E)$ 返回的是一个新的事件流 E'，E' 满足关系模式 $R(A)$。

重命名操作符的符号为 ρ_ψ。其中 ψ 是从一个属性集合到另外一个属性集合的映射。假设 E 为一个事件流，其关系模式为 $R(A,B)$，那么 $\rho_{E1(A1,B1)}(E)$ 返回一个新的事件流，其名字为 $E1$，关系模式为 $R(A_1, B_1)$。

2.2.2 事件操作符

事件操作符包括时间相关的操作符和时间无关的操作符。为了描述的方便，我们假设有三个事件流 E, E_1 和 E_2，以下的事件操作符的示例都使用这三个事件流。由于复杂事件的起始时间戳是其子事件的最小起始时间戳，结束时间戳是其子事件的最大结束时间戳，在后续的描述中不对起始和结束时间戳进行说明。

时间相关的事件操作符有以下几个：

1）顺序事件（;）：E_2 在 E_1 之后发生。

$$E_1; E_2=\{\langle\{e_1, e_2\}; start, end\rangle \mid e_1.end<e_2.end \wedge e_1\in E_1 \wedge e_2\in E_2\}$$

2）事件序列（$;^+$）：E 顺序发生一次或者次。

$$;^+E=\{\langle\{e_1, e_2\cdots e_n\}; start, end\rangle \mid n\geqslant 1 \wedge \forall 1\leqslant i\leqslant n, (e_i\in E) \wedge \forall 1\leqslant i\leqslant n-1, (e_i.end<e_{i+1}.end)\}$$

在这个操作符里，我们分别用 *pre* 和 *next* 代表相邻两个事件的前一个和后一个。

3）事件窗口（ω）：E 发生，而且持续时间小于 t。

$$\omega_t(E)=\{\langle\{e\}; start, end\rangle \mid e\in E \wedge end\text{-}start<t\}$$

4）时限顺序事件：E_2 在 E_1 发生之后 t_1 到 t_2 之内发生

$$E_1 ;_{<t_1,t_2>} E_2 =\{\{\langle\{e_1, e_2\}; start, end\rangle \mid e_1\in E_1 \wedge e_2\in E_2 \wedge e_1.end<e_2.end \wedge e_2.end-e_1.end\leqslant t_2 \wedge e_2.end-e_1.end\geqslant t_1\}$$

5）时限事件序列：E 发生了一次或者多次，而且每两次的事件间隔为 3 秒钟。

$$;_t^+E=\{\langle\{e_1, e_2\cdots e_n\}; start, end\rangle \mid n\geqslant 1 \wedge \forall 1\leqslant i\leqslant n, (e_i\in E) \wedge \forall 1\leqslant i\leqslant n-1, (e_{i+1}.end-e_i.end=t)\}$$

时间无关的事件操作符有以下几种：

1）事件与（&）：E_2 和 E_1 都发生件。

$$E_1 \& E_2=\{\langle e_1, e_2\rangle; start, end\rangle \mid e_1\in E_1 \wedge e_2\in E_2\}$$

2）事件或（|）：E_2 或者 E_1 发生。

$$E_1 \mid E_2=\{\langle\{e\}; start, end\rangle \mid e\in E_1 \vee e\in E_2\}$$

3）事件非（～）：单独的事件非没有任何意义，需

要通过其他操作符与其他事件组合起来才有意义。下面的式子代表 E_2 在 E_1 之后发生，且中间没有 E 发生。

$$E_1;\sim E;E_2=\{\langle\{e_1,e_2\};start,end\rangle \mid e_1\in E_1\wedge e_2\in E_2\wedge e_1.end<e_2.end\wedge \neg\exists e\in E,(e_1.end<e.end\wedge e.end<e_2.end)\}$$

2.2.3 聚集操作符

在复杂事件中包含事件序列或者时限事件序列的时候，有时候我们需要存储该事件序列某些属性的聚集值（比如连续发生 10 次，需要计算发生 10 次）来支持我们的查询需求。由此，Argus 系统必须能够支持聚集操作符。在 Argus 系统中，我们将支持 MAX，MIN，AVG，COUNT，SUM 等聚集操作符。但是，需要注意的是这些聚集操作符只能作用于事件序列或者时限事件序列中事件元组或者其属性上。

2.3 一些例子

在超市中，有时会发生偷盗事件，而人工监控非常困难。若在货架、结账处和出口安装 RFID 阅读器，对应产生三个事件流 SHELF，COUNTER 和 EXIT。假设商品有一个属性为 id。若有人想偷窃商品，则会发生商品在货架读到，而后在出口读到，期间在结账处没读到，该复杂事件可以用如下查询表达：

$$\sigma_\theta(\omega_{2hours}(SHELF;\sim COUNTER;EXIT))$$

$$\theta=(SHELF.id=COUNTER.id)\wedge(EXIT.id=COUNTER.id)$$

在股市中，有人想在一支股票连续增长 5 次后抛出。这一应用里有事件流 S，每一个事件代表一次报价，假设每个事件有属性 id 和 price。可以使用如下查询表达这个事件：

$$\sigma_\theta(;^{+}S)$$

$$\theta=(S.pre.id=S.next.id)\wedge(S.next.price>S.pre.price)\wedge count(S)=6$$

3 系统框架

图 1 给出了支持复杂事件处理的 Eagle 系统的架构。其中蓝色部分为 Argus 数据流处理系统中本来就存在的功能，绿色部分是为了能够支持复杂事件处理对 Argus 系统原有的模块进行扩展后的模块，而紫色部分是为了支持复杂事件处理而新增加的模块。

存储管理模块负责接受事件，并根据系统负载进行抛弃元组或者生成大纲，为了能够支持复杂事件处理，存储管理模块为每个事件赋予一个时间戳。查询优化器负责接受查询，进行查询解析，生成语法树、查询计划，最后根据优化原则把查询计划与查询网络合并成新的查询网络。为了能够支持对复杂事件的处理，对查询优化器的查询解析和计划生成与合并部分进行了升级。查询网络是用户提交的所有查询构成的一个系统的查询计划，由于新增加了很多事件算子，因此查询网络也与以前的查询网络有了很大的变化。事件算子库是为了支持复杂事件处理而新增加的，其中增加了序列构成、非事件过滤、顺序事件过滤和窗口过滤算子等。算子库是 Argus 原来存在的算子，主要包括聚集算子、选择算子、投影算子、窗口算子等。调度器负责在存储管理器、（事件）算子库和查询网络之间进行调度，根据系统负载和算子优先级来调度算子的执行顺序。

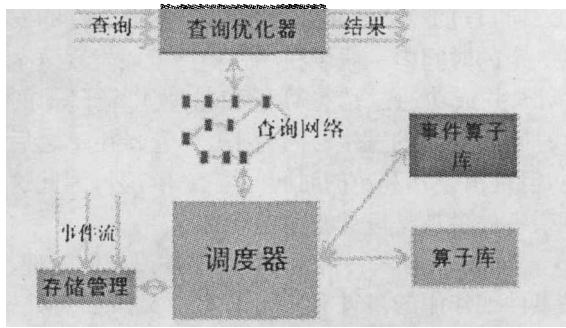

图 1 Eagle 系统架构

4 基于查询计划的事件探测

上面介绍了 Eagle 复杂事件代数和系统框架，这一部分将介绍基于查询计划方式的复杂事件探测，包括探测算法及其优化。

4.1 基本算子库

为了实现基于查询计划的复杂事件处理，我们需要把一个查询转化为一个计划，计划中的每一个节点对应一个算子，下面我们将介绍复杂事件探测过程中用到的算子。为了叙述的方便，我们给出一个复杂事件查询的例子，假设有事件流 A,B,C,D 和 E，都满足关系模式 $R(id,value)$，我们要进行如下查询 Q：

$$\omega_{6secs}(\sigma_\theta(E;(A|\sim B);\sim C;D))$$

其中 $\theta=\theta_1\wedge\theta_2$，$\theta_1$ 要求各个子事件在属性 id 上的值相等，θ_2 要求 D 的属性 $value$ 大于 3。

序列构造算子：序列构造算子是一个连接算子，通过连接操作把可能的复杂事件探测出来。在上面提到的事件操作符中，事件非操作符作用的事件一定不允许出现在复杂事件之中。除此之外，其他事件操作符作用的事件必须出现，在序列构造算子中，只考虑这些事件，但是在这里不会考虑事件发生的先后顺序。

选择算子：选择算子是一个过滤条件，它对构成复杂事件的事件作了某些限制条件，只有满足这些

限制条件的复杂事件才是合法的复杂事件。如在上例中,在序列构造算子中发现复杂事件 ad,但是 $a.id \; ! = d.id$,在选择算子就会把这个复杂事件过滤掉。

非事件过滤算子:在序列构造算子中,没有对事件非算子进行处理。在上面的例子中,假设探测到复杂事件 ad,而在非事件过滤中发现 a 和 d 之间有事件 c 发生,那么非事件过滤算子会把这个复杂事件过滤掉。

顺序事件过滤算子:由于在序列构造算子中没有考虑事件发生的先后顺序,如果在这一阶段发现了复杂事件 ad,a 的发生时间为 10,而 d 的发生时间为 8,这实际上不满足 d 在 a 之后发生,因此顺序过滤算子会把这个复杂事件过滤掉。

窗口过滤算子:窗口过滤算子要求事件的发生在一定的时间内。假设在序列构造阶段发现复杂事件 ad,并成功通过选择算子和非过滤算子,a 的发生时间为 3,d 的发生时间为 10,尽管 d 在 a 之后发生,但是由于 d 和 a 的时间间隔大于 5 秒,因此这个复杂事件会被窗口过滤算子过滤掉。

上面的算子有一个问题,对事件或如何处理,如果事件或作用的事件有一个非事件又该如何处理?在这里,我们对这个问题作专门的说明,该问题需要分为三种情况来处理。

1)事件或作用的没有非事件,直接在序列构造阶段处理。如果有一个或事件发生则该复杂事件通过或验证,输出事件序列,否则不通过。

2)事件或作用的全部是非事件,在非过滤阶段进行处理。如果有一个非事件使得条件满足则复杂事件通过或验证,否则不通过。

3)事件或作用的既有非事件,也有普通事件首先在序列构造阶段进行处理。如果有一个或事件满足条件,则满足条件,复杂事件通过或验证,输出包含该或事件的事件序列,该事件序列不必再由该或中包含的非事件过滤算子过滤;否则,需要在非事件过滤阶段验证,如果有一个非事件使得条件满足则通过或验证,否则不通过。同时,还要输出所有的不包含或事件的序列到非事件过滤算子进行过滤。

对查询 Q,我们生成如图 2 的查询计划

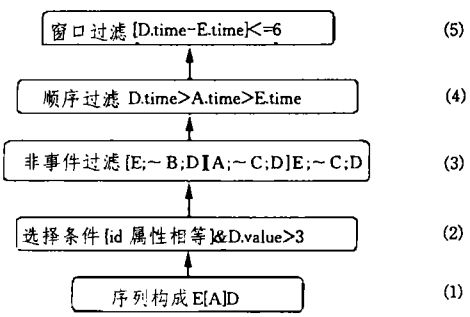

图 2 查询 Q 的查询计划

4.2 查询计划的优化

由于事件流通常具有持续、大量的特征,而系统同时要检测的复杂事件的数目又比较多,因此复杂事件探测是非常耗时的工作。为了提高效率,我们针对上面的特征以及基于查询计划的探测方法本身的特点提出了一些优化的策略。

4.2.1 尽早执行过滤算子

对上面的查询计划,如果有图 3 的事件序列,则经过(1)后会有序列 e1a1d1,e1a1d2…e2a2d4,e1d1…e2d4 等 24 个序列满足条件,而在经过(2)后还有 e1a1d1,e2a2d2,e2a2d3,e2a2d4,e2d2 等 5 个满足条件,经过(3)后还有 e2a2d2,e2a2d3,e2a2d4,e2d2 等 4 个满足条件,经过(4)后还有 e2a2d3,e2a2d4,e2d2 等 3 个满足条件,而在经过(5)之后只有 e2a2d3 和 e2d2 满足条件。

事件	e1	c1	a1	e2	d1	d2	a2	d3	c2	d4
id	1	1	1	2	1	2	2	2	3	2
value	0	2	3	4	5	4	4	4	6	4
时间	0	1	2	3	4	6	7	8	9	10

图 3 事件序列

如果在序列构造阶段不过滤,产生的候选序列会非常多,而大部分候选序列不满足查询条件。因此我们考虑通过减少产生的候选序列的个数来进行优化。经分析可以看到,第一,在选择算子中,如果某个谓词仅仅与序列构造算子中出现的事件相关,那么可以把基于这个谓词的过滤放在序列构造阶段进行;第二,顺序过滤算子的过滤条件仅仅与序列构造算子中出现的事件相关,因此可以把顺序过滤算子放在序列构造阶段进行;第三,窗口过滤算子可以放到序列构造阶段进行,这样能大量减少不满足窗口过滤条件的序列。

优化后,查询 Q 对应的查询计划如图 4 所示。

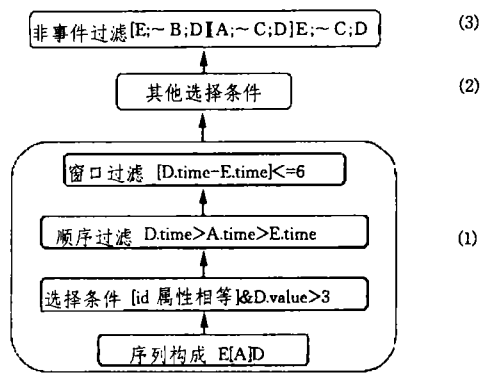

图 4 查询 Q 优化后的查询计划

使用优化后的查询计划,在(1)阶段之后,还有 e1a1d1,e2d2 和 e2a2d3 等三个序列满足条件,减少了候选序列的数目,从而提高了系统的效率。

4.2.2 共享序列构造算子

由于通常会有大量的查询在同时进行，因此会产生多个查询计划，每个查询计划都有自己的序列构造算子，而各个序列构造算子要求的子事件有可能完全相同，在这种情况下，可以通过共享序列构造算子来实现优化。假设有事件流 A,B,C,D，有两个查询 $Q1(A;\sim B;D)$ 和 $Q2(A;\sim C;D)$，在进行此项优化前，生成的查询计划如图 5(1)所示，而在优化后生成的查询计划如图 5(2)所示。

由图可以看出，查询计划(1)是孤立的，而查询计划(2)是一个查询网络。共享算子可以从两方面优化系统。第一，减少了系统的算子个数，因此可以减少对系统内存资源的使用。第二，不同查询共享该算子提供的计算和产生的中间结果，降低了系统对 CPU 和内存资源的压力。

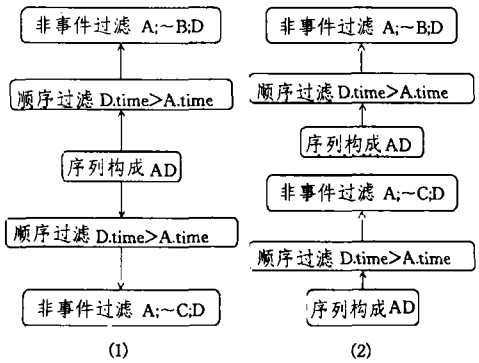

图 5　优化前 VS 优化后的查询计划

5　实验

5.1　实验环境

实验所用的机器是 Intel 酷睿 E4400 双核，主频 2.0GHz，内存 1G，操作系统为 Windows XP。所有与实验相关的程序都运行在这台机器上。

在实验中，我们使用 5 个事件流，每个事件流有 5 个属性，每个属性都是整型，取值范围在 1 到 10000 之间，为每个事件流我们产生了 100 个事件。每个查询有 1～2 个静态选择条件，1 个动态选择条件，每个查询有 0～1 个非操作符作用的事件，窗口长度默认为 100 秒，每个查询有 0～1 个迭代操作符，其余的操作符和条件由系统随机生成。

5.2　结果分析

5.2.1　实验参数说明

(1)查询的数量：系统中注册的查询的总数量。

(2)节点的数量：系统中存在的图节点的数量。

(3)中间序列数量：在事件探测过程中各个节点产生的中间事件序列的数量。

(4)平均响应时间：$Total(T)$ / $Total(e)$，其中 $Total(T)$ 代表系统进行复杂事件探测用的总时间，$Total(e)$ 代表参与事件探测的事件数目。

5.2.2　实验结果分析

我们分别比较了优化前后图节点个数、中间序列数量和平均响应时间。

图 6 是共享等价节点前后图节点的数目比较。由图可见，优化前的节点数量非常大，随着查询数量等比增长，而共享后，节点数量大幅减少，增长比较缓慢。

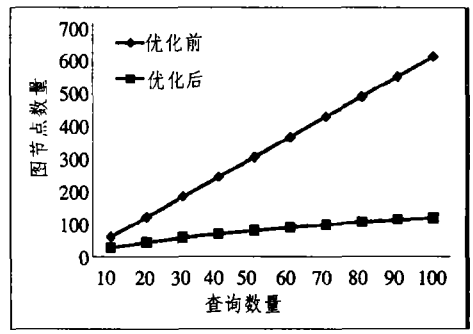

图 6　共享等价节点对图节点个数的影响

图 7 给出了优化前后的中间序列数量的对比，优化后中间序列的数量大概为优化前的 1/4。

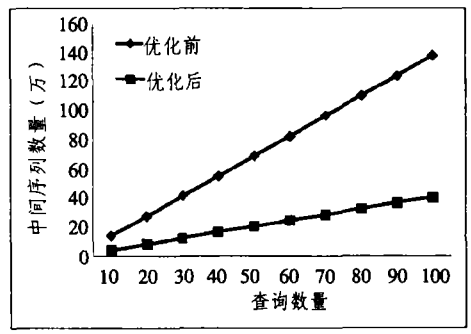

图 7　优化算法对中间序列数量的影响

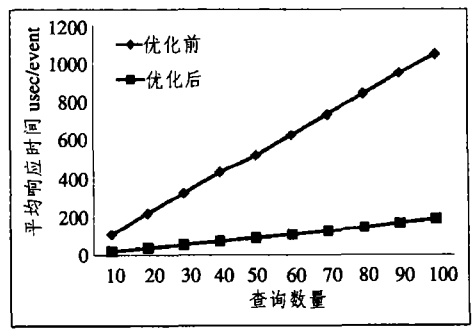

图 8　优化算法对平均响应时间的影响

图 8 给出了优化前后的平均响应时间的对比。可见，优化后，系统的处理速度有了很大提高。这得益于序列构造算子的共享和及时过滤不满足条件的中间序列。

结束语　本文在 Argus 数据流处理系统的基础上实现了复杂事件处理系统 Eagle。本文提出了表达能力较强的 Eagle 复杂事件代数，可以表达大部分复杂事件。提出了基于查询计划的事件探测方

(下转第 51 页)

其内部注入JAVA_CLASS的方法注入外部文件安全特性检测程序。这样数据库用户可以在Oracle内部编写读取文件安全特性的JAVA源程序，交给Oracle后台编译为JAVA_CLASS[8]。检测程序就可以建立存储过程或函数来调用这些JAVA_CLASS，以此获取文件的权限和内容。

结束语 数据库文件安全特性检测是数据库安全特性检测的重要内容，受到数据库安全专家和数据库管理员的广泛重视。一般在DBMS厂家的安装配置中都有安装配置指南，但缺少合适的数据库安全特性检测工具验证用户的数据库文件安装配置是否符合用户的安全目标。本文围绕数据库安全特性检测工具研发的目的，归并了不同操作系统平台上各种DBMS的差异，提出了一个通用的数据库文件检测内容，以及基于这个检测内容，针对不同平台的数据库文件检测方法和实现技术。文中所提的数据库文件检测内容和技术已经在Windows、Linux和Unix平台中，针对不同版本的Oracle数据库，得以实施。而且，本文提出的数据库文件检测内容、方法和技术具有通用性，只需要针对不同的DBMS，构建如图2所示的检测记录，就可以对其文件安全特性进行检测。目前我们正在DB2、SQL Server等数据库平台上进一步丰富数据库文件检测内容和技术，以便进一步验证和完善本文提出的检测框架。

参 考 文 献

[1] Litchfield D, Anley C, Heasman J, et al. The Database Hacker's Handbook: Defending Database Servers. Wiley, 2005

[2] Litchfield D. The Oracle Hacker's Handbook: Hacking and Defending Oracle. Wiley, 2007

[3] Shaul J, Ingram A. Practical Oracle Security. Syngress, 2007

[4] Knox D. Effective Oracle Database 10 g Security by Design. McGraw-Hill, 2004

[5] Natan R B. Implementing Database Security and Auditing. Elsevier digital press, 2005

[6] Powell G. Oracle High Performance Tuning for 9i and 10g. Elservier digital press, 2004

[7] Newman A. Database Activity Monitoring: Intrusion detection and security auditing. Application Security Inc white paper, 2006

[8] Oracle. Oracle Database Online Documentation. http://www.oracle.com/technology/documentation/database.html, 2008

（上接第31页）

式，在此基础上，通过把过滤算子放到序列构造阶段执行以减少候选序列的个数以及使用共享算子来减少系统算子的个数，降低了系统对CPU和内存的压力，提高了复杂事件探测的效率。

下一步，我们将从以下方面对系统进行完善：

1. 考虑进一步的优化策略，包括其他算子的共享和各个算子内部算法的优化。

2. 由于传感网络和RFID数据本身具有的不确定性，导致了其上复杂事件的发生也具有一定的不确定性，因此下一阶段要考虑概率复杂事件的处理策略。

参 考 文 献

[1] Demers A, Gehrke J, Panda B, et al. Cayuga: A general purpose event monitoring system // Proc. CIDR. www.cidrdb.org, 2007: 412-422

[2] Gehani N H, Jagadish H V, Shmueli O. Composite event specification in active databases: Model and implementation // Proc. VLDB. 1992: 327-338

[3] 董玮，陈立军，赵加奎，等. Argus：一个自适应的数据流管理系统// 2005中国计算机大会论文集

[4] Wu E, Diao Y, Rizvi S. High-performance complex event processing over streams. In SIGMOD, 2006: 407-418

[5] Wang F, Liu S, Liu P, et al. Bridging physical and virtual worlds: Complex event processing for rfid data streams. In EDBT, 2006: 588-607

[6] Chakravarthy S, Krishnaprasad V, Anwar E, et al. Composite Events for Active Databases: Semantics, Contexts and Detection // VLDB'94. 1994: 606-617

[7] Adaikkalavan R, Chakravarthy S. SnoopIB: Interval-based event specification and detection for active databases // Proc. ADBIS. 2003: 190-204

[8] Garfinkel S, Rosenberg B. RFID: Applications, Security, and Privacy. Addison-Wesley, 2006

[9] Chandy K M, Aydemir B E, Karpilovsky E M, et al. Event Webs for Crisis Management // Proc. of the 2nd IASTED Int'l Conf. on Communications, Internet and Information Technology. 2003

[10] Rizvi S, Jeffery S R, Krishnamurthy S, et al. Events on the Edge // SIGMOD'05. 2005: 885-887