

# ProSPer: 一个支持 proactive 特性的 通用型事件监控系统

刘家红 吴泉源

(国防科学技术大学计算机学院网络与信息安全研究所 长沙 410073)

**摘 要** 大规模网络安全监控应用中需要对网络安全态势进行动态评估,在网络出现重大安全风险前进行 proactive 特性的有效防范.把网络安全监控系统建模为事件监控系统,对满足复合时序和属性值逻辑关系的多个事件进行关联,把多个原子事件复合为语义更丰富、更抽象的复合安全事件.已有研究提出了不同的复合事件检测模型,但缺乏 proactive 的事件监控能力.基于时序关系并不能提高事件监控的预测能力的假设,设计了基于 top- $k$  复合事件检测模型的事件监控系统 ProSPer,为网络安全监控等应用系统提供 proactive 特性的事件监控能力.与已有的复合事件检测系统相比,ProSPer 检测复合事件时无需读取全部成分事件,这种 proactive 特性是非常有意义的设计.

**关键词** 网络安全;事件监控;事件流处理;复合事件;proactive 特性

**中图法分类号** TP311 **DOI号**: 10.3724/SP.J.1016.2009.00773

## ProSPer: A Proactive Event Monitor for General Purposes

LIU Jia-Hong WU Quan-Yuan

(*Institute of Network Technology & Information Security, School of Computer,  
National University of Defense Technology, Changsha 410073*)

**Abstract** Network security monitor of large scale demands dynamic, continuous evaluation of security situation, proactive protection against approaching network risks. If the authors model security monitors as continuous Event Monitor Systems, that is to say, to correlate multiple events in complex temporal relationship and attribute logic relationship to richer semantic, more abstract complex event in security domain. Much works has studied to design complex event detection model but they lack the proactive capability to monitor event. Based on the assumption that temporal relationship does not improve the predictive ability of event monitor, the authors design a proactive event monitor system ProSPer for application domains like situation evaluation for network security and present a fast top- $k$  based algorithm to detect complex events. Current event detection techniques require the fully read of relevant records, while the new method only needs partial read, thus this proactive capability is desirous.

**Keywords** network security; event monitor; event stream processing; complex event; proactive capability

1 引 言

Proactive 特性是与 reactive 相对的,指能感知状态变化,并在这些变化产生对自身的影响前主动采取行动的一种特性.在基于监控的应用中,存在很多需要 proactive 特性的需求.例如,大规模网络安全监控应用中需要对网络安全态势进行动态评估,在网络出现重大安全风险前进行有效防范以减小损失.实时企业为了主动、准确且详细地获取关于库存、生产、市场等相关信息,以便企业能及时决策,实现企业最大效益,通常会使用 RFID、无线传感器网络等智能物件来对生产、物流情况进行全局把握和预测<sup>[1]</sup>;使用业务流程监控系统来主动地获取业务运营情况,以保证业务流程的合规性,留住客户,同时还可以发现潜在客户,拓展业务<sup>[2]</sup>.这些应用通过对连续到达的领域信息进行分析处理,为了最大化收益和最小化损失,通常需要 proactive 特性的处理能力.

2 动 机

事件监控系统处理的通常都是连续到达的信息,需要对这些信息进行实时、连续的监控和分析.事件流处理是对快速连续到达的事件进行计算的技

术,用于事件系统运行时对事件进行监控以及辅助构造事件驱动系统.根据应用领域的不同需求,事件流处理对连续到达的信息有不同的数据模型以及相应的处理模型.这些模型在结构模型和处理模型上都是事件驱动处理的特例,区别在于其对连续到达信息的速率的承受能力和对一致性的保证<sup>[3]</sup>.连续到达的信息的复合时序关系和时序关系上值属性的逻辑关系代表上层应用感兴趣的复合事件,体现了上层应用的业务逻辑,事件流处理系统依此对这些复合事件进行检测,连续输出经过复合后的结果事件,以触发相应的后继处理<sup>[3]</sup>.

目前,在线事件流处理系统通常采用图 1 所示的体系结构.对复合事件  $E=A \text{ op } B$  的检测过程大致如下,其复合事件的检测是观察到复合事件的终结事件(成分事件中时序上最后一个事件)时开始的:

- (1) 当事件类型  $B$  的实例  $b$  进入系统时,监控器注意到复合事件  $E$  的终结事件实例出现,告知事件处理器;
- (2) 事件处理检查复合事件表达式,获知复合事件由两个顺序的原子事件组成,即  $A, B$ ;
- (3) 根据事件表达式,考察事件历史,以检查是否事件类型  $A$  的实例在历史中;
- (4) 在历史中找到  $a$ ,执行事件实例消费,并更新事件历史;
- (5) 检测复合事件实例  $e$  成功,输出.

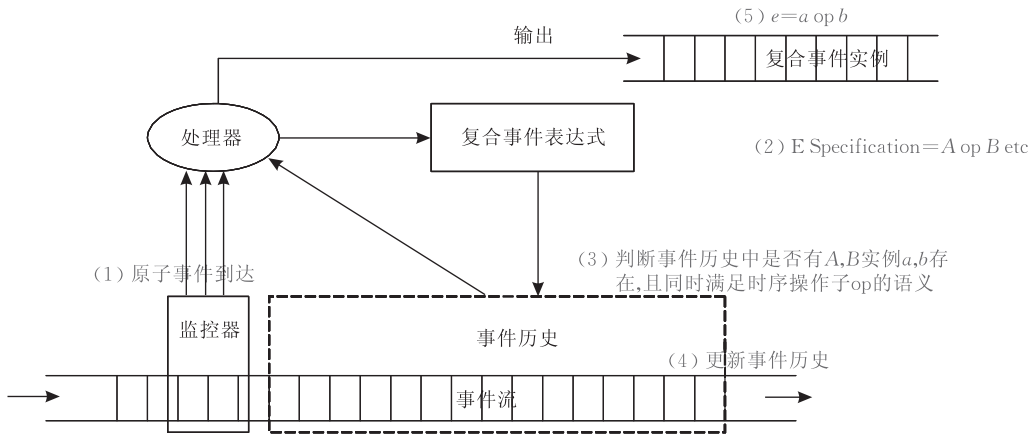


图 1 在线事件流处理体系结构

当外部输入速率高,注册的复合事件表达式数目多时,复合事件的检测存在以下问题:①对复合事件的检测依赖于终结事件,其监控缺乏主动式的 proactive 特性;②每个复合事件表达式在内存中都会载入一个监控器,内存开销大,只要有新的输入事件到达监控器时都会去搜索复合事件表达式,效率低下.

计算资源是有限的,把所有可能发生的复合事

件都检测出来不太实际,有时也没有必要.一个可能的解决方案是:只列出  $k$  个最可能发生的复合事件,即为了尽早评估复合事件发生与否,支持 proactive 特性的事件流处理,使用索引技术找出外部事件到达时可能受影响的复合事件,并根据连续到达的输入事件对复合事件发生的影响累积概率,在索引上执行 top- $k$  选择,找出目前为止发生概率较大即较可能发

生的复合事件,实现 proactive 的事件监控能力.

网络安全中,主动防御包含两方面的含义:一是能检测出未知的安全威胁,二是能在安全威胁发生前预警. ProSPer 这两方面对主动防御的支持如下.

2.1 未知模式的检测

基于统计分析的方法是检测未知安全威胁非常具有代表性的方法,主要通过已经收到的历史告警信息,基于概率来建立关于告警事件的预测模型. 然后,通过训练出的预测模型去计算新告警信息与哪个正处于关联过程中的攻击序列最接近,从而完成整个告警关联的工作. 该方法引入了基于时间序列分析的预测方法<sup>[4]</sup>. 本文也是基于类似的机制,区别在于 ProSPer 弱化处理时序信息,另外 ProSPer 的检测模型是基于数据流的实时、连续方式,并使用了索引技术来提高检测性能.

2.2 预警

以安全模型 P2DR<sup>[5]</sup>为例探讨 ProSPer 的预警能力. P2DR 是目前信息安全领域具有代表性的安全模型,基本原理为:信息安全相关的所有活动,不管是攻击行为、防护行为、检测行为和响应行为等,都与时间密切相关,因此可以用时间来衡量一个体系的安全性和安全能力. 如果防护时间大于检测时间和响应的时间的和,即  $P_t > D_t + R_t$ ,也就是在入侵者对攻击目标造成实际的危害之前就能被检测到,

若及时处理,则系统是安全的. 如前所述,若把安全威胁建模为复合事件,则其不需所有成分事件全部发生,即安全威胁出现苗头但又未发生前,ProSPer 通过动态评估发生概率最大的 top- $k$  个安全威胁的形式,尽早报告最可能发生的安全威胁告警.

基于此动机,本文设计并实现了支持 proactive 特性的事件监控系统 ProSPer (Proactive Stream Processor). ProSPer 构建在服务计算平台上,可对运行时服务和计算平台本身进行监控. 第 3 节给出本文设计的假设以及 ProSPer 的系统结构;第 4 节给出 ProSPer 用于复合事件检测的 top- $k$  算法 TCHybrid,给出了正确性证明和示例;随后给出了算法的实验性能;最后对相关工作进行总结.

3 假设与 ProSPer 系统结构

3.1 假设

麻省理工大学的 Tapia 在其硕士论文中使用复合事件检测来实现基于对象的活动识别,其研究表明,原子事件的顺序并不提高复合事件检测算法的预测能力<sup>[6]</sup>. 本文基于此结论,在事件监控系统的复合事件检测中,把时间上相邻发生的事件看作一个整体,而不考究其复合时序关系. 另外,上层应用感兴趣的复合事件对应用的重要程度显然是不同的,因此应

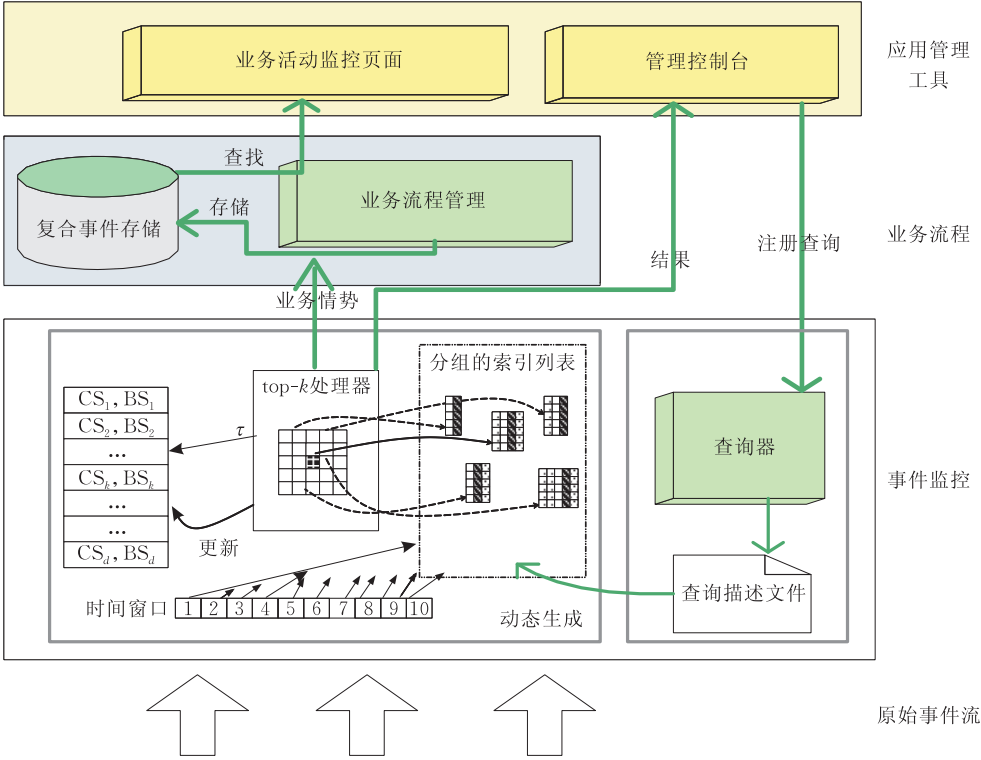


图 2 ProSPer 系统结构

当在复合事件检测中引入事件的重要性这一因素. 最后, 本文借鉴信息检索中的相关技术, 把检测时需要查找的元数据均存储在数据库中, 为其构建索引, 以提高检测性能.

3.2 系统结构

ProSPer 构建和运行在面向服务计算平台 InforSIB 上. InforSIB 基于 Java 开发, 是一个支持服务总线 and 事件通信的面向服务的应用集成开发和运行平台<sup>[7]</sup>. ProSPer 对外部触发的应用层事件、业务流程和流程内部服务触发的事件均能进行监控.

如图 2 所示, 安全监控等业务领域的应用通过管理控制台向 ProSPer 注册其感兴趣的安全态势即复合事件. ProSPer 为管理控制台构建底层的事件监控查询表达式, 结合查询描述文件中定义的用于检索的元数据, 动态生成相应的索引列表. 外部事件流输入后, 也可以触发相应的索引列表构建动作, 同时 ProSPer 会为其构建时间窗口. 时间窗口划分为多个等分时长的更小单元, 初始化后每次滑动的单位为一个单元, 如此时间窗口可以重用其中的多个单元. Top- $k$  处理器根据构建的时间窗口中到达的原子事件来查找索引列表, 根据查找的概率以及重要性信息, 动态评估此时最可能发生的  $k$  个安全态势的发生概率. 结果返回到管理控制台, 同时复合事件还可持久化, 便于之后的离线分析.

4 用于复合事件检测的 top- $k$  算法

传统的基于聚集排序 top- $k$  查询算法<sup>[8]</sup>中, 排

序访问是指对已排序序列至顶向下的依次访问, 随机访问是指从序列中随机获取某个对象的值. FA 必须在排序访问阶段得到至少  $k$  个匹配才会停止, 不适合 proactive 特性的事件监控; TA 算法至少要比 FA 停止得早, 但是其随机访问开销大; NRA 无随机访问, 不能用于最后精确输出 top- $k$  个复合事件; CA 算法权衡了随机访问和排序访问的代价, 可看作 TA 与 NRA 算法的结合, 然而其需要内存来记录很多的中间结果, 会加重算法的内存消耗, 因此也不适合直接应用于复合事件检测.

使用 top- $k$  算法进行复合事件检测存在以下两个难点:

- (1) top- $k$  数据集是多维的, 包含了引发事件发生的主体. 事件元组的关键字属性和引发主体的组合是区分元组唯一性的要素;
- (2) top- $k$  查询是根据用户指定的单调聚集函数从数据集中找出函数值最高的前  $k$  个结果. 简单的概率累加函数是不满足聚集函数性质的.

因此, 下文先给出复合事件定义的元数据、为复合事件设计的内部数据结构和聚集函数, 然后再讨论两阶段算法.

4.1 事件元数据

在事件监控系统的复合事件检测过程中需要查找事件元数据, 其中包括用于触发事件发生的事件主体、原子事件描述、复合事件描述、复合事件重要性、原子事件发生对复合事件发生的条件概率. 这些元数据用数据库模式表示, 如图 3 所示.

object		primitiveEvent		complexEvent		event_importance		primitive_event_probability	
PK	ObjectID	PK	PEID	PK	CEID			PK	CEID
	Name Group	I1	Owner		Event_expression Discription	I1	CEID	I1	Probability
		I1	Target Status			I2	Importance	I1	StageNumber
						I2	StageNumber		

图 3 事件定义用到的数据库表

4.2 内部数据结构

时间窗口中的 top- $k$  查询项是原子事件实例, 而被查询的数据结构是按事件类型保存的, 内部数据结构作为中间层, 以桥接事件实例和事件类型定义. 另外, 到达的安全事件作为原子, 还与导致其发生的主体有关, 相同主体上发生的原子事件间相互关联, 组合成安全态势即复合事件. 因此, 需要内部数据结构来区分不同的安全事件主体. 如图 2 所示, 原始事件流输入到 ProSPer 后, ProSPer 为之构建相应的、由多个等长的小单元组成的时间窗口. 每个

小单元中发生的原子事件均有指针指向索引列表. 索引列表分为两层, 事件主体为区分事件的第一层, 另外, 再根据态势即复合事件的步骤数来对数据进行第二层索引, 如图 4 所示. 内部数据结构维护 top- $k$  查询执行时的运行数据, 根据 top- $k$  查询来 cache 数据, 保存在 cache 的数据能提高访问的速度. 另外内部数据结构还负责访问数据库. 为了进一步提高访问速度, 同一事件类型的所有实例共享一个数据库连接(cache). 提供的内部数据结构具体如图 4 所示.

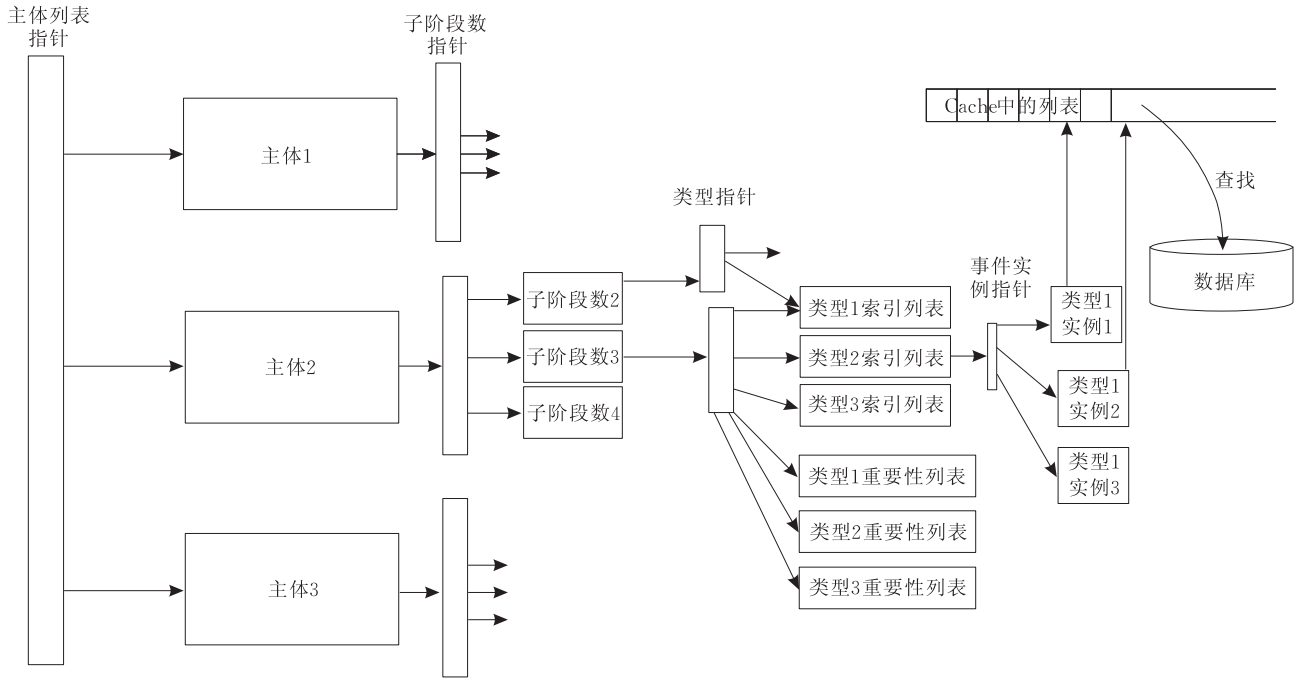


图 4 内部数据结构

### 4.3 Top-k 聚集函数

设每个原子事件都对应一张概率表,表示其发生时相应的复合事件发生的条件概率,表按降序排列。复合事件由多个原子事件组合而成。假设原子事件的发生不是互斥的,则原子事件都发生时复合事件发生的概率依据条件概率计算公式来计算。但条件概率的计算不是单调的,因此不能用来进行 top-k 事件查找。另一种方案则把复合事件看为原子事件所代表的不同子阶段的组合,每个子阶段的权重是平等的,把所有子阶段的概率累加,以得到复合事件发生的概率。此时的函数是单调的。设复合事件  $C_x$  由  $n$  个原子事件  $E_1, \dots, E_n (n \geq 1)$  组合而成,  $C_x$  的重要性为  $Imp$ 。  $C_x$  的发生概率可定义为  $\sum_{i=1}^n P_i + Imp$ 。因为每个复合事件其含有的原子事件个数  $n$  是不定的,可考虑复合事件发生概率计算为  $\sum_{i=1}^n P_i / n + Imp$ 。在事件检测过程中,假设  $E_1, \dots, E_m (1 \leq m \leq n)$  是已经发生的原子事件,  $C_x$  的发生概率可通过连续地计算  $\sum_{i=1}^m P_i / n + Imp$  得到。但  $m$  是不确定的,最后修正聚集函数为  $\sum_{i=1}^m S_i + Imp$ , 其中  $S_i$  是指对复合事件出现概率的实时打分值,将在后面详细介绍。

### 4.4 两阶段算法: TCHybrid

两阶段算法介于 TA 算法和 CA 算法<sup>[8]</sup>之间。

第一阶段使用排序访问来找出可能的复合事件候选列表,第二阶段使用随机访问来剪枝候选列表以得到最后的  $k$  个复合事件。一方面,排序访问阶段尽量剔除无关的复合事件,因此随机访问阶段所基于的候选列表比 TA 算法更具选择性,另一方面,与 CA 相比,并不使用周期性的排序和随机访问,因此记录的中间结果要少。

TCHybrid 算法可分为 5 步,其中语句 1, 2, 3 对应步骤 1, 用于创建并初始化候选列表。语句 3 调用初始化候选列表算法,在随后详细给出。语句 4 对应步骤 2, 根据得到的候选列表计算阈值  $\tau$ 。阈值  $\tau$  表示当前候选列表中第  $k$  个最大的打分值。语句 5 与 6 对应步骤 3, 维护一个矩阵,矩阵中每个单元根据不同事件主体和事件的子阶段数来区分。矩阵的每个单元包含综合打分值最高的分组,每个分组是一个索引列表。WHILE 循环中的语句 8 用于处理对应的最高打分值的分组  $g$ 。循环语句中的其余语句对应步骤 5, 用于更新候选列表、阈值、最高分数。当最高打分值低于阈值或索引列表均扫描完毕时循环中止。

算法 TCHybrid

输入: 时间窗口  $tw$

输出: 包含  $k$  个元素的复合事件候选列表 topList

1. OwnerListByStage olbs =  
getOwnerListByStage( $tw$ , MAX\_PECOUNT-1)
2. TopList topList = new TopList( $k$ Value,  
olbs.getOwnerListSize(), MAX\_PECOUNT-1)



```

3. initializeTopList(elbs, topList)
4. float threshold = topList.getThreshold()
5. Coordinates co = topList.getNextFetchItem()
6. float nextHigh = co.getHighValue()
7. WHILE ((nextHigh >= threshold) & & (!topList.isFinished()))
8.   processAccordingToCo(co, elbs, topList,
                           timeWindow, getTypeList())
9.   co = topList.getNextFetchItem()
10.  threshold = topList.getThreshold()
11.  nextHigh = co.getHighValue()
12. ENDWHILE
13. randomPhase(topList, elbs)

```

#### 4.4.1 排序访问

排序访问主要是对复合事件候选列表进行初始化. 循环包含两个主要的步骤: 第 1 步对应代码第 3~17 行, 获得当前需处理的原子事件对应的索引列表, 包括相应的概率索引列表和重要性索引列表, 对所有这些列表进行处理, 根据索引列表中的信息, 若综合打分值高于阈值, 则创建相应的复合事件实例, 若在候选列表 topList 中已有相应的复合事件实例, 则更新其打分值和综合打分值; 第 2 步对应 18~28 行, 返回最高的综合打分值, 对由主体和子阶段所分组的矩阵进行维护, 因此下一次循环可以知道从哪个扫描深度进行.

#### 4.4.2 随机访问

随机访问阶段同时使用了对候选列表进行剪枝的两种方法. 第 1 种剪枝方法是初步剪枝, 把综合打分值低于阈值的复合事件从候选列表中删除.

第 2 种剪枝方法包括两个步骤: 首先对候选列表进行预处理, 把候选列表分为两个按综合打分值降序排列的表, 一个表包含  $k$  个最高的值, 另一个表包含其余的项; 其后的操作对  $bl$  进行处理, 提取  $bl$  中综合打分值最高的复合事件实例, 然后检查内存中的数据并随机访问以得到此事件实例的分数, 如果综合打分值高于阈值  $\tau$ , 则此复合事件被插入到  $tl$  中, 同时更新  $\tau$ , 如果  $bl$  尾部的项的综合打分值低于新阈值  $\tau$ , 剪枝这些项. 第 2 步将循环执行, 直至  $bl$  为空.

算法 InitializeTopList

输入: 时间窗口  $tw$

输出: 复合事件候选列表 topList

```

1. FOR  $i=0$  to olbs.getOwnerListSize()
2.   FOR  $j=0$  to olbs.getNumberOfStages()
3.     peType = olbs.getOwnerStageList( $i$ ).getInverted-
        ListsOfStage( $j$ )
4.     HighTypeList htl = new HighTypeList()
        float highValueForOwnerStage = 0f
5.     //构造 High Value List, 得到分数高的列表

```

```

6.   FOR  $n=0$  to peTypes.size()
7.     htl.insert(((StageTypeList) peTypes.get( $n$ )).get-
        PEType(), ((StageTypeList) peTypes.get( $n$ )).
        getCachedRowAt(0).getProbability())
8.   ELSE typeIndicator++
9.   WHILE ((htl.getPointer() != null) & & ( $times <
        j+2$ ))
10.    highValueForOwnerStage += htl.getPointer().getH-
        ighValue()
11.     $times++$ ; htl.stepForward()
12.  ENDWHILE
13.  htl.resetPointer()
14.  //获得重要性列表索引, 更新打分值, 把 hvl 表中
        最高的  $j$  个值和重要性值累加
15.  IF (getCachedImpRowSets( $j$ ).size() > 0)
16.    topList.insertByParas(...)
17.    highValueForOwnerStage +=
        getCachedImpRow( $j, 0$ ).getImportance()
18.  //对每个原子事件对应的索引列表, 执行初始化,
        候选列表按打分值降序排列
19.  FOR  $n=0$  to peTypes.size()
20.    IF (((StageTypeList) peTypes.get( $n$ )).getNum-
        berOfCachedRows() > 0)
21.      ceid = ...; probability = ...; peid = ...
22.      topList.insertByParas(ceid,  $j+2$ ,
        olbs.getOwnerStageList( $i$ ).getEventObject(),
        probability, peid, htl, getCachedImpRow( $j, 0$ ).
        getImportance())
23.    ENDIF
24.  ENDFOR
25.  topList.setIndicator( $i, j$ , highValueForOwner-
        Stage)
26.  IF (typeIndicator < peTypes.size())
27.    topList.setFetchDepth( $i, j, 0$ )
28.  ELSE topList.addOneToFinishedCount()
29.  ENDFOR
30. ENDFOR

```

#### 4.4.3 检测结果的正确性证明

**定理 1.** 最终的 top- $k$  个复合事件输出肯定在排序访问阶段结束后得到的候选列表中.

证明. 通过逆否命题来证明.

(1) 在候选列表 topList 中, 可以得到阈值  $\tau$ , 其意义为第  $k$  大的当前打分值. 因此在候选列表 topList 中有  $k$  个可能的复合事件, 其最终打分值不小于  $\tau$ .

(2) 取不属于 topList 的复合事件  $ce_0$ , 其包含  $p$  个原子事件, 其所在的索引列表分组在  $g_p$  中. 因此

$$ce_0 \text{ 的最终打分值为 } S_{ce_0} = \sum_{i=1}^p S_{ce_i} + I_{ce_0}.$$

(3) 在索引分组  $g_p$  中会一直扫描, 直到组中其余项的最高打分值小于  $\tau$ . 因此, 设当前组  $g_p$  的扫描深度为  $d$ , 则有  $\sum_{i=1}^p S_{dmax_i} + I_d < \tau$ .

(4) 所有成分事件中  $g_p$  分组深度  $d$  之前的复合事件均已添加到候选列表. 设  $ce_0$  的任何成分事件被扫描到的深度为  $d_0$ , 则  $d_0$  不小于  $d$ . 设  $ce_0$  在组  $g_p$  中对应的索引列表为  $l_1, \dots, l_p$ , 由于列表是降序排列的, 则  $\forall i \in \{1, 2, \dots, p\}, S_{d_0 i} \leq S_{d_i}$  且  $I_{ce_0} \leq I_d$ . 累加函数是单调的, 因此  $\sum_{i=1}^p S_{ce_i} \leq \sum_{i=1}^p S_{d_0 i} \leq \sum_{i=1}^p S_{d_i}$ .

由(3),  $\sum_{i=1}^p S_{d_{max_i}} + I_d < \tau$ ,  $S_{d_{max_i}}$  是组  $g_p$  中深度  $d$  扫描的最大值, 因此  $\sum_{i=1}^p S_{d_i} \leq \sum_{i=1}^p S_{d_{max_i}}$ . 与(2)、(4)结合, 则  $S_{ce_0} = \sum_{i=1}^p S_{ce_i} + I_{ce_0} \leq \sum_{i=1}^p S_{d_i} + I_d \leq \sum_{i=1}^p S_{d_{max_i}} + I_d < \tau$ . 又由(1), 已有  $k$  个复合事件其打分值大于  $\tau$ , 因此  $ce_0$  肯定不在最后的 top- $k$  集合中.

定理 1 得证. 证毕.

**定理 2.** 随机访问阶段结束后, 最终能从候选列表中找到 top- $k$  个复合事件

证明. 随机访问阶段首先把候选列表 topList 分成两个列表, 当前打分值最大的  $k$  个复合事件放在列表 tl 中, 并按打分值降序排列. 候选列表中其余的项放在另一列表 bl 中. 通过对 bl 继续扫描更新打分值, 如果其实际打分值比 tl 表中最小的值要高, 把 tl 表中的最小值项替换为此项, 并从 bl 中剪枝此项. 更新 tl 的阈值. 任选候选列表中的元素 ce 且  $ce \notin tl$ , 有两种方式来剪枝 ce, 或者直接从候选列表中直接剪枝, 或者从 tl 中剪枝.

第 1 种剪枝方法中, ce 的综合打分值小于当前阈值  $\tau$ , 不需计算, 显然 ce 的实际打分值  $S_{ce}$  不会大于估计的综合打分值  $BS_{ce}$ , 而  $BS_{ce}$  小于  $\tau$ , 因此 ce 必定不在最终的  $k$  个结果中.

第 2 种剪枝方法中, ce 的实际打分值  $S_{ce}$  需要计算出来, 但其小于  $\tau$ , 不在最终的  $k$  个结果中.

定理 2 得证. 证毕.

算法 RandomPhase

输入: 候选列表 topList, 分组的主体类型列表 olbs

输出: 包含最终  $k$  个复合事件项的排序列表 tl

1. firstFilter(topList)
2. ReviewedCEMap rcm=new ReviewedCEMap()  
//访问过的复合事件 cache
3. ReviewedImpMap rim=new ReviewedImpMap()  
//重要性 cache
4. ReviewedProbMap rpm=new ReviewedProbMap()  
//概率 cache
5. //第 1 步

6. SortedList tkl=new SortedList()
7. SortedList bsL=new SortedList()
8. ComplexEventInstanceIncomplete ceii
9. FOR  $i=0$  to topList.getK()
10. ceii=topList.removeFirst()
11. IF(ceii!=null) //访问过的复合事件
12. //添加概率
13. //添加重要性
14. topList.insert(new SortedObject(ceii, ceii.getCurrentScore()))
15. ENDIF
16. ELSE //未访问过的复合事件
17. String exp=rs.getString("expression")
18. rcm.insertMapping(ceid, exp)
19. //查找数据库中的复合事件表达式, 判断原子事件是否在复合事件表达式中, 同时查找其它概率和重要性信息, 并添加进来
20. ENDFOR
21. //第 2 步
22. float kthScore=tkl.getKthScore(k)
23. WHILE(tl.getSize()>0)
24. ceii=tl.removeFirst()
25. bsL.insert(new SortedObject(ceii, ceii.getBestScore()))
26. ENDWHILE
27. WHILE((bsL.getSize()>0)&&  
(bsL.getLargestScore()>kthScore))
28. ceii=bsL.removeFirst()
29. //余下操作类同于第一步, 忽略
30. ...
31. ENDWHILE

#### 4.4.4 算法的 proactive 特性

要分析 TCHybrid 算法的 proactive 预警能力, 只需分析算法的检测窗口内, 对于检测出的  $k$  个, 各含有  $n_i (i=1, 2, \dots, k)$  个成分事件的复合事件, 运行算法需要检测的原子事件数目  $m$  (其中包含了与检测出的复合事件相关的成分事件数目为  $t_i$ ) 与  $n_i$  的关系. 因为 TCHybrid 算法有剪枝策略, 若  $m > n_i$ , 最佳

打分值  $BS$  为  $\sum_{j=1}^{t_i} S_j + \sum_{p=1}^{m_i-t_i} HS_p$ , 其中  $S_j$  为对相关的成分事件的概率,  $HS_p$  为从逆向索引列表中获取的可能最佳打分值, 此时访问了  $t_i$  次数据库, 对逆向索引列表的  $m_i - t_i$  次访问的时间开销是可忽略的;

若  $m < n_i$ , 最佳打分值  $BS$  为  $\sum_{j=1}^{t_i} S_j + \sum_{p=1}^{m_i-t_i} HS_p$ , 同理, 访问了  $t_i$  次数据库, 对逆向索引列表的  $m_i - t_i$  次访问的时间开销也是可忽略的. 综合来说, 算法检测时

对于一个有  $n_i$  个成分事件的复合事件, 实际输入时出现  $t_i$  个事件, 若  $t_i < n_i$ , 则以  $t_i$  次的数据库访问加索引访问的时间就能实现对其的检测, 达到预警的目标.  $t_i = n_i$  时只需要访问索引的延迟时间, 则可近实时地将其检测, 此时检测的 proactive 特性体现在基于统计分析安全事件报警关联上, 能实现对未知模式的检测.

实际上, 对于网络安全的主动防御来说, 多源告警信息关联得到的复合事件通常都是多步骤、时间跨度比较大的. ProSPer 设置的时间窗口大小(通常为 1min 内)一般小于其时间跨度, 实际运行时报告  $k$  个最可能发生的复合事件时两个阶段(排序访问和随机访问)只涉及到时间窗口内发生的成分事件, 因此实际检测时检测的原子事件数目通常是小于

PE1	PE2	PE3	综合打分值 (BS)
(CE1, 0.8)	(CE5, 0.9)	(CE3, 0.7)	2.4
(CE5, 0.5)	(CE2, 0.6)	(CE1, 0.6)	1.7
(CE2, 0.3)	(CE1, 0.5)	(CE4, 0.4)	1.2
(CE8, 0.1)	(CE4, 0.1)	(CE, 0.1)	

图 5 3 个原子事件时的算法运行过程示例

4.4.6 内存访问开销

当一个原子事件进入系统时, 若按如下方式处理:

- (1) 检查复合事件表达式定义;
- (2) 检查复合事件定义中出现在此原子事件之前的事件实例是否发生;
- (3) 使用随机访问来获取之前事件的概率;
- (4) 累积发生的事件的概率.

假设有  $k$  个复合事件包含原子事件  $E_x$ , 上述处理方式处理一个原子事件实例的 I/O 成本为  $\sum_1^k C_i$  ( $C_i$  为处理复合事件  $i$  的成本). 建立一个复合事件 id 上的索引表, 则  $C_i = 1 + \sum_1^t 1 + C_j$  ( $t$  为已发生的原子事件实例数,  $C_j$  表示查找索引的成本). 因此在出现  $t$  个原子事件后处理一个原子事件实例  $E_x$  的总成本为  $C = k + kt + \sum_{i=1}^k \sum_{j=1}^t C_{ij}$ . 但如果表中存储  $\sum_{i=1}^t S_i + Imp$ , 若原子事件到达, 则系统只需把表中的值与阈值比较, 然后只处理那些比阈值大的值. 此时的 I/O 成本大大减小. 假设有  $p$  个条目的值大于

$n_i$  的. 这说明了 TCHybrid 算法的预警能力.

4.4.5 示 例

下面给出一个示例, 设时间窗口内待考察的 3 个原子事件  $PE_1$ 、 $PE_2$ 、 $PE_3$ , 其内部数据结构如图 5 左侧所示. 事件的重要性其内部数据结构和算法处理与此处的事件概率类似, 为简单起见, 此处不再涉及. 为每个复合事件计算此时发生的概率. 综合打分为累加此扫描深度时所有的复合事件发生概率,  $k$  为 2 时阈值为第  $k$  个最大的概率, 综合打分值和阈值都会随着扫描深度增加而递减, 扫描直至综合打分值小于阈值结束. 扫描深度 1 时综合打分为  $0.8 + 0.9 + 0.7 = 2.4$ , 阈值为 0.8. 到扫描深度 3 时, 综合打分值 1.2 小于阈值 1.4, 扫描结束.

阈值	最终结果	候选列表
0.8	CE1 (0.8), CE5 (0.9)	CE1, CE5, CE3
1.4	CE1 (1.4), CE5 (1.4)	CE1, CE5, CE3, CE2
1.4	CE1, CE5	CE1, CE5, CE3

阈值, 则处理原子事件实例  $E_x$  的成本为  $1 + p$ ,  $p \ll k$ .

5 实 验

我们对 ProSPer 的性能进行了模拟实验(环境配置如表 1). 实验使用不同规模的数据集, 规模 1~4 分别包含了 40, 60, 80, 100 个原子安全事件以及相应的 1200, 3000, 5000, 10000 个复合事件. 复合事件和原子事件的表示都沿用了前文的实验所用的安全事件格式, 但复合事件中未使用表达式  $D$ , 因为其中的否定在 ProSPer 中无法检测<sup>[7]</sup>. ProSPer 解析这些表达式, 获知其含有的成分事件数目. 另外, ip 地址作为识别所有事件的 owner-id. 所有的实验均测试 10 次, 取其平均值.

表 1 实验环境配置

项	配置情况
操作系统	Windows XP SP2
硬件配置	Pentium 4 3.2GHz, 1GB RAM
JVM 版本	JRE 1.5.0Update6
数据库	MySQL Server 5.0



实验测试的不同参数包括:产生原子事件的源的数目  $n$ 、时间窗口长度  $twl$ ,以秒为单位、数据库缺省的 cache 长度  $d$ ,衡量标准为一次排序访问中从数据库中 cache 的记录数目. 参数的缺省配置为  $n=20$ ,  $twl=20$ ,  $d=200$ .

实验测试包括复合事件检测时 top- $k$  检测时间、随机访问对检测性能的影响、其它参数对 top- $k$  查询的影响.

5.1 检测时间

检测时间随原子事件和待检测的复合事件数目增大而增大,如图 6. 与时间序敏感的检测相比,检测结果只报告最可能发生的 5 个(默认数目)事件,检测时间大大减少,若把时间序敏感的系统检测时间(对单个的复合事件的检测延迟为毫秒级)作为复合事件的发生时机,TCHybrid 算法对复合事件的检测提前了很多,具有 proactive 的预警能力. 从图 6 可看出,TCHybrid 因为使用了小时间窗口来重用已计算的结果,并在随机访问阶段使用了逆向索引,比在相同数据集上使用 TA 和 CA 算法检测要速度快 50% 以上.

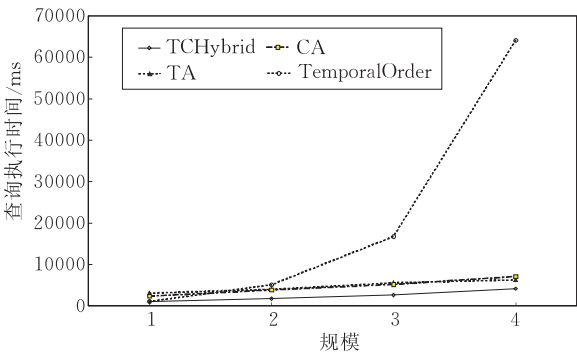


图 6 检测时间

5.2 不同参数对检测性能的影响

5.2.1 随机访问对检测性能的影响

随机访问阶段两种剪枝方法的剪枝效果如图 7,从图可看出,平均能剪枝 30% 以上. 之所以能

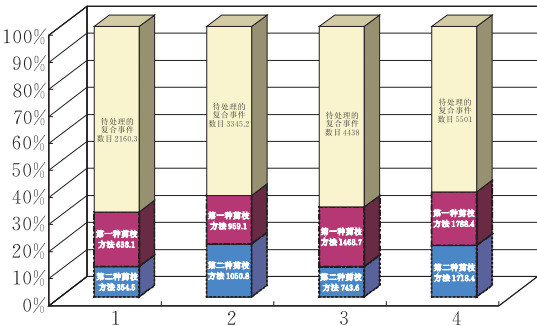


图 7 随机访问阶段的剪枝效果

够达到如此的剪枝效果,从而减少随机访问的次数,是因为在检测时时间窗口划分了更小的单元,每次检测滑动窗口只移动一个小单元窗口,其他部分的小窗口在上一次检测的随机访问记录已被 cache 到内存中,得到重用;另外,检测时在内存中还实时构建了原子事件与复合事件关系的逆向索引列表(索引为复合事件 id),也能减少随机访问的次数.

5.2.2 其他参数对检测性能的影响

在测试其它参数对检测性能的影响实验中,测试某个参数时,调整参数值,其它参数按缺省值配置. 实验说明,时间窗口初始比较小的情况下,查询执行时间增长很快,但在时间窗口比较长后(15s 开始),由于时间窗口对小单元的重用,检测时间保持比较稳定的区间. 另外,产生原子事件的源的数目  $n$  直接影响了需要构建的索引列表,检测时间与  $n$  存在一定的正比性. 而 cache 长度  $d$  几乎对检测性能不产生影响,因为从算法上分析,对检测性能起决定作用的是随机访问的次数,排序访问阶段所用时间占检测时间比例不高,因此对检测性能影响不大,如图 8.

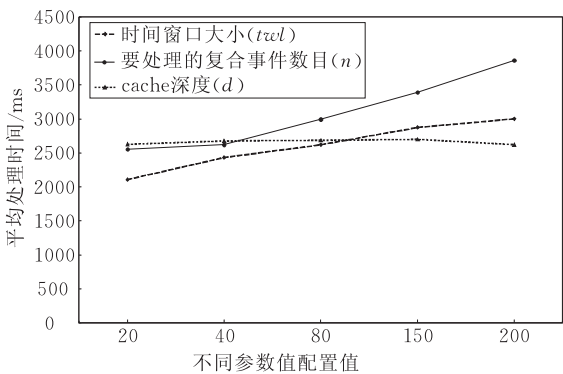


图 8 其它参数对检测性能的影响

6 相关工作

事件监控系统对连续到达的信息进行实时、连续的监控和分析,需要对满足复合时序和属性值逻辑关系的多源信息进行关联,把多个原子事件复合为语义更丰富、更抽象的复合事件. 已有研究通过对复合时序关系和属性值逻辑关系进行严格定义,采用一种反应式的事件流处理体系结构,提出了基于非确定性有限自动机、图、树、Petri 网、实时时态逻辑、事件代数等模型来进行复合事件检测<sup>[7,9-13]</sup>.

SASE 提出基于查询计划的方法,把时序关系、属性上的逻辑关系等操作分别使用查询计划中的操

作符进行处理,所有的查询都被翻译为查询计划中的 6 个基本操作阶段:序列扫描、序列构造、属性选择、窗口、否定操作和转换,操作阶段间存在先后关系. SASE 使用了很多的优化技术来快速地检测复合事件,包括尽早在事件序列中进行谓词选择、使用堆栈索引来减少中间结果等<sup>[9,14]</sup>. Cayuga 使用一种扩展的自动机,将状态间的转换分为自循环的转换和前向转换,将此两种转换分为用于过滤和绑定. 不同的操作有不同转换类型,并定义了一套规则来维护各个转换边上的选择谓词和换名谓词. Cayuga 还使用索引技术和多查询优化技术来提高快速到达的事件流时复合事件的检测性能<sup>[10]</sup>. 本文前期工作<sup>[7]</sup>是通过严格定义复合时序关系操作子,基于事件代数模型进行复合事件检测的,并使用 SEDA 并发处理以及满足复合事件的可组合性以及有效的原子事件剪枝策略来提高事件检测性能. CEDR 注重从理论角度考察复合事件模型,认为不同的事件流处理系统均有不同的一致性需求和负载承受能力,并因此定义了统一的一致性模型<sup>[3]</sup>. ReCEPtor 致力于在面向服务计算平台中集成复合事件处理引擎,使用与 SASE 类似的技术来进行复合事件检测<sup>[12]</sup>.

这些系统对复合事件的定义,虽然在模型语义以及事件语言表达能力上有差异,但均假设事件必须考虑复合时序关系,对顺序发生的事件必须严格按照时序关系来检测,提供的复合事件检测能力都是 reactive 特性的,基于索引和多查询优化技术等性能优化方法在事件流处理遇到大时间窗口时,多个原子事件到达、多个复合事件查询的应用场景不能提供足够有效的解决方案. 而本文基于顺序关系并不提高对事件检测预测能力的假设,使用 top- $k$  的复合事件检测模型,一定程度上解决了大时间窗口、多事件查询等应用场景下的复合事件检测问题.

## 7 结 论

本文设计了基于 top- $k$  的事件监控系统 ProS-Per,对复合事件的时序关系进行弱化处理,从而提供 proactive 的事件监控能力. 主要贡献为:(1) 针对面向事件监控的事件流处理中大时间窗口、多事件查询、事件输入负载高等应用场景均能有效并高效地进行监控;(2) 使用索引技术、大时间窗口分片,多事件查询时能提供有效的重用,提高多事件查询的检测性能;(3) 使用 top- $k$  复合事件检测,多事件

查询时能提供 proactive 的事件监控能力. 然而本文的工作也有很多局限性,首先本文对基于值属性逻辑关系关联的复合事件并不能提供有效的检测;第二,本文弱化事件的复合时序关系,复合事件检测模型不具有可组合性,提供的事件模型在多大程度上能反映通用应用场景下的应用语义还需进一步的验证.

## 参 考 文 献

- [1] Zang C. Complex event processing in enterprise information systems based on RFID. *Enterprise Information Systems*, 2007, 1(1): 3-23
- [2] Li Guo-Li, Jacobsen Hans-Arno. Composite subscriptions in content-based publish/subscribe systems//*Proceedings of the Middleware 2005*. Grenoble, France, 2005: 249-269
- [3] Barga Roger S, Goldstein Jonathan, Ali Mohamed, Hong Mingsheng. Consistent streaming through time: A vision for event stream processing//*Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*. Asilomar, California, USA, 2007: 363-374
- [4] Qin Xinzhou, Lee Wenke. Statistical causality analysis of INFOSEC alert data//*Proceedings of the International Symposium on Recent Advances in Intrusion Detection*. Pittsburgh, PA, 2003: 101-127
- [5] Feng Deng-Guo, Zhang Yang, Zhang Yu-Qing. Survey of information security risk assessment. *Journal on Communications*, 2004, 25(7): 10-18(in Chinese)  
(冯登国, 张阳, 张玉清. 信息安全风险评估综述. *通信学报*, 2004, 25(7): 10-18)
- [6] Tapia E M, Intille S S, Larson K. Activity recognition in the home using simple and ubiquitous sensors[M. S. dissertation]. Program in Media Arts and Sciences, School of Architecture and Planning, MIT, 2003
- [7] Liu Jia-Hong, Wu Quan-Yuan. Event-driven service-oriented computing platform. *Chinese Journal of Computers*, 2008, 31(4): 588-599(in Chinese)  
(刘家红, 吴泉源. 一个基于事件驱动的面向服务计算平台. *计算机学报*, 2008, 31(4): 588-599)
- [8] Fagin R, Lotem A, Naor M. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 2003, 66(4): 614-656
- [9] Diao Y, Immerman N, Gyllstrom D. SASE+: An agile language for kleene closure over event streams. *UMass Technical Report 07-03*, 2007
- [10] Demers A, Gehrke J, Panda B, Riedewald M, Sharma V, White W. Cayuga: A general purpose event monitoring system//*Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*. Asilomar, 2007: 412-422

[11]

Rizvi Shariq. Complex event processing beyond active databases: Streams and uncertainties. Technical Report UCB/EECS-2005-26, 2005

[12]

Wei Mingzhu, Ari Ismail, Li Ismail, Dekhil Mohamed. Re-CEPTor: Sensing complex events in data streams for service-oriented architectures. Digital Printing and Imaging Laboratory, HP Laboratories, Palo Alto: HPL-2007-176, 2007

[13]

Barringer Howard, Goldberg Allen, Havelund Klaus, Sen Koushik. Rule-based runtime verification. Department of Computer Science, University of Manchester, 2003

[14]

Wu Eugene, Diao Yanlei, Rizvi Shariq. High-performance complex event processing over streams//Proceedings of the SIGMOD. Chicago, Illinois, 2006: 407-418



**LIU Jia-Hong**, born in 1980, Ph. D. candidate. His current research interests include event stream processing and service-oriented computing.

**WU Quan-Yuan**, born in 1941, professor, Ph. D. supervisor. His research interests include distributed computing, artificial intelligence.

Background

Events play an important role in many computer systems. Network security events are obstacles to Internet development. The trend that organizations are linking system security monitoring efforts closely to real-time processes makes research and industrial community increasingly focus on the Event Stream Processing (ESP) and proactive network monitoring connection. Due to high speed arrival rate of events and vast volume of registered complex event queries, memory consumption and incremental event query evaluation demand a comprehensive dedicate ESP framework with security event detection in proactive, low-latency and high scalability. Much works have been studied to design complex event detection model but lack proactive capability to monitor events.

Inspired by web information retrieval, and the recent research conclusion that the order of primitive events does not add prediction power to the detection algorithms, the authors present a proactive ESP monitor ProSPer (Proactive Stream Processor). Effective complex event detection is conducted on events arrival in time window. By incorporating statistics probability of events, every primitive event instances in a stream can affect the probability of current or forthcoming occurrence of a complex event. Due to the high volume of events and event expressions available respectively in the stream, they only consider the most probable occurring

events according to some kind of indicator of uncertainty, that is the top-*k* monotone aggregation function they designed. Current event detection techniques require the fully read of relevant records, while the method only needs partial read, thus this proactive capability is desirous. Because event instances in time window are terms to query complex events, the top-*k* detection for ESP is multi-dimensional. The authors solve this by building inverted index lists in the database, access them on-demand during detection. To minimize costs for sorted and random accesses in top-*k* detection and improve processing efficiency, they re-use the smaller time window of detection. The authors justify performance by experiment results.

The method focuses on efficient event detection, and as for further performance improvement, the authors need to improve data structure in memory and the detection algorithm. Also, to detect complex event with more predictive power and effectiveness, they need to consider more factors into the top-*k* detection, like the probability effects caused by order of primitive events.

The authors gratefully acknowledge financial support from Project 863 under granted Nos.2006AA01Z451, 2007AA01Z474, the Ministry & Commission-Level Research Foundation of China under granted No.[2006]634.