基于层次自动机模型的复杂事件层次实现研究

金大卫[1] 施 斯[1] 易 彩[1] 杨 兵[2]

(中南财经政法大学信息与安全工程学院 武汉 430073)[1] (湖北大学教育学院 武汉 430062)[2]

摘 要 复杂事件处理技术从数据流中提取满足特定模式的事件序列,具有实时、海量、智能的特点,近年来引起了学术界和商业界的广泛关注。但是,之前的工作侧重于对单层复杂事件检测的研究。事实上,由于业务系统对信息有不同层次的需求,需要对事件进行分层处理,单层复杂事件检测并不能充分支持事件分层的需求。针对这种情况,在事件层次概念以及传统NFA模型的基础上,定义了分层复杂事件检测模型层次自动机NHA,基于NHA模型设计了更为直观高效的EH-Tree结构,并给出了分层复杂事件检测HCED算法和代价模型。最后以吞吐量和内存占用为指标,进行了大量的实验,对比并分析了HCED算法与传统基于NFA模型的SASE算法的时间性能和空间性能。实验结果表明,HCED算法能有效且高效地实现分层复杂事件检测,填补了CEP不支持分层复杂事件检测的空白,为下一步研究提供了基础。

关键词 复杂事件处理,模式匹配,事件层次,层次自动机,树

中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.07.028

Study of Complex Event Hierarchies Realization Based on Hierarchical Automata Model

JIN Da-wei[1] SHI Si[1] YI Cai[1] YANG Bing[2]

(School of Information and Safety Engineering, Zhongnan University of Economics and Law, Wuhan 430073, China)[1]

(School of Education, Hubei University, Wuhan 430062, China)[2]

Abstract Complex event processing technology extracts event sequences which satisfy the specific patterns (called complex events) from continuous event streams. It processes a large sum of data in real time intelligently and has attracted the attention from both academia and industry in recent years. However, the state-of-the-art researches focus on single-layer complex event detection. In fact, because business systems have different hierarchical needs of information, events in system should be processed according to their hierarchies. Single-layer complex event detection cannot afford to meet the needs of hierarchical information in different managerial hierarchies. To deal with such problem, in this work, a hierarchical complex event detection model, hierarchical automata, was defined based on the concept of event hierarchies and traditional nondeterministic finite automaton (NFA) model. Then we designed a more intuitive and effective EH-Tree structure and proposed hierarchical complex event detection (HCED) algorithm and its cost model. Finally, taking throughput and memory consumption as indexes, massive experiments were performed to compare the spatial and temporal performance of HCED algorithm based on EH-Tree and SASE algorithm based on NFA. The results testify that our HCED algorithm can perform hierarchical complex event detection effectively and efficiently, which fills in the blanks of hierarchical complex event detection and indicates that our work can be a basis for further research of complex event processing.

Keywords Complex event processing, Pattern matching, Event hierarchies, Hierarchical automata, Tree

1 引言

随着企业信息化的发展和信息更新速度的日趋加快,数据的时效性变短、时间成本越来越高。传统的基于历史数据和批处理方式的数据处理方法无法对数据进行实时处理,因此并不适合处理时效短或时间成本高的数据。复杂事件处理(Complex Event Processing,CEP)作为流处理技术的一种,能够实时处理多个来自不同事件源的通常具有不同类型的事件,具有实时、海量、智能的特点,可有效降低数据的处理时间,节约处理的时间成本。

Luckham于2002年在《事件的力量》[1]一书中提出了"复杂事件处理"的概念,并提出了CEP应用的蓝图。目前,复杂

到稿日期:2016-04-11 返修日期:2016-09-26 本文受国家社会科学基金(13CTJ003),中国博士后基金(2014M562025),湖北省自然科学基金(2013CFB003)资助。

金大卫(1977-),男,博士,教授,主要研究方向为金融信息工程、高频金融数据分析、算法交易,E-mail:jdw@znufe.edu.cn;施 斯(1992-),女,硕士,主要研究方向为复杂事件处理、高频交易;易 彩(1994-),女,硕士,主要研究方向为金融信息工程、复杂事件处理;杨 兵(1975-),男,博士,副教授,主要研究方向为自适应学习系统、无线传感器网络、现代数据库技术。

事件处理已经形成了较为完整的理论体系，研究主要包括功能组件、事件处理、部署架构、组件交互、数据组织、时间特性、规则表示、语言设计[2]等方面。在非功能特性方面，CEP 引擎的可靠性[3-5]、可扩展性[6]、用户友好性[7]以及 QoS[8]等方面都有深入的研究。目前也有许多已实现的 CEP 原型系统，如 Cayuga[9-11]，SASE[12-14]等，以及相对成熟的商业 CEP 产品，如 EsperTech 公司的 Esper、Sybase 公司的 CEP 平台、TIBCO 公司的 BusinessEvents 和 StreamBase、Oracle 公司的 CEP 产品等。在应用方面，CEP 已经被成功运用于商业行为监控[15]、金融服务[16]、虚拟环境中的资源分配[17]、车辆环境[18]、无线传感网络[19-20]、M2M 通信[21]等各个领域。

然而，纵观复杂事件处理的研究，当前的复杂事件处理系统普遍只能基于 IT 层数据进行简单的事务处理，无法实现更为抽象的信息管理和决策支持。这是因为目前 CEP 的研究侧重于对单层复杂事件检测的支持，较少关注分层复杂事件检测(Hierarchical Complex Event Detection)。实际上，分层复杂事件检测的需求在系统中非常常见。对于系统，尤其是较为复杂的系统，对信息的需求存在不同层次，如操作层、业务层、管理层、决策层等。不同层次的信息需求对应不同层次的复杂事件，被称为“事件层次(Event Hierarchies)”[1]。事件层次对系统各个层次的复杂事件进行建模，从而能够实现较为抽象的业务逻辑。而对事件层次的支持，需要 CEP 引擎来实现分层复杂事件检测。

目前多数 CEP 并不支持分层复杂事件检测，少数能够支持的系统[22]也缺乏统一的检测模型，在理论完备性上有所欠缺，同时由于缺乏整体的控制，会造成计算资源的浪费。另外，没有检测模型也无法建立整体的时间开销模型。

针对这一现状，本文在原有单层复杂事件处理机制的基础上，形式化定义了事件层次问题，引入了层次化的自动机检测模型——层次自动机。在层次自动机模型的基础上设计了用于模式匹配的 EH-Tree 结构和匹配算法，并进行复杂度分析，从理论上分析了 EH-Tree 结构相比于传统的 NFA 结构在复杂度上的优势。最后，对 EH-Tree 结构进行了大量的测试实验，结果表明，基于 EH-Tree 结构的匹配算法在有更丰富功能的情况下，较传统的 NFA 模型在时间和空间上性能都有所提升。

2 相关工作

传统的数据库管理系统存储数据，再对存储的数据进行查询。主动数据库[23]在此基础上加入了触发器(trigger)，使得数据库不再被动等待查询，而能够在满足条件时进行自主查询。数据流处理系统[24]则完全摒弃了数据存储和一次性查询，采用了内存缓存和持续查询的方式，使得数据流处理成为可能。此时数据流的处理更多侧重于事件流的计算，而非从低层事件中提取高层事件。

Luckham 在《事件的力量》[1]一书中介绍了复杂事件处理，指出 CEP 可以被看作是接收和匹配低层事件来产生高层事件的服务，提出了事件层次、事件处理代理(event processing agent，EPA)和事件处理网络(Event Processing Network，EPN)等概念(见图 1)。之后 Sharon 等人[25]建立了 EPN 的概念模型，认为 EPN 包含了事件生产者、事件消费者、事件处理代理以及事件通道(Event Channel，EC)。这些工作在概念上设计了事件层次处理的蓝图，但没有提出事件层次实现的具体方案。

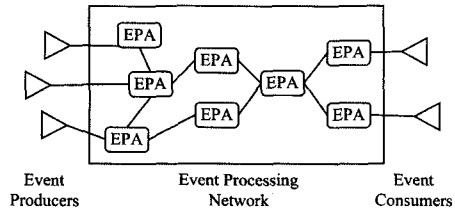


图 1 事件处理网络系统架构

复杂事件处理的早期研究集中于事件处理语言和检测模型的研究。复杂事件处理系统所采用的检测模型主要有基于非确定有限自动机(Nondeterministic Finite Automaton，NFA)的系统以及基于树模型的系统。

在基于 NFA 的复杂事件模式匹配中，查询语句按一定的算法被编译成 NFA(见图 2)，当进行复杂事件模式匹配时，自动机会根据当前状态以及到达的简单事件判断是否进行状态转换，若自动机到达接受状态则表明复杂事件被检测出来。典型的以 NFA 为检测模型的系统有美国康奈尔大学开发的 Cayuga 系统[9-11]和加州大学伯克利分校的 SASE 系统[12-14]。NFA 模型的每个状态都只能处理简单事件，其结构决定了其只能对简单事件进行模式匹配，因此以 NFA 为检测模型的复杂事件处理的研究始终针对简单事件流模式匹配，对于需要从复杂事件流中提取更高层次的复杂事件，NFA 模型难以实现。

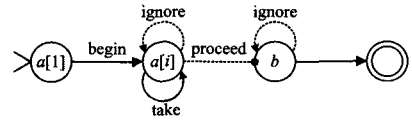


图 2 基于 NFA 模型的复杂事件处理

在基于树的复杂事件模式匹配中，查询语句被编译成匹配树(见图 3)，树的叶子节点为简单事件，中间节点为查询的中间结果。当进行复杂事件模式匹配时，对应叶子节点接收简单事件并向父节点传递，父节点判断是否匹配查询条件并向上传递，或者父节点主动向子节点询问，若到达根节点则说明复杂事件被检测出来。基于树模型的系统有 GEM[26-27]，ZStream[28]和 PMTree[29]等。以树为检测模型的复杂事件处理系统是对以 NFA 为检测模型的复杂事件处理系统的一种补充，由于 NFA 模型的局限性，对于某些如合取、析取、否定等的操作实现模式匹配时只能采用间接的方法，而使用树作为检测模型则可以克服这些问题。但是以树为检测模型的复杂事件处理系统所构建的树，其叶子节点是简单事件，根节点是复杂事件，模式匹配从叶子节点经过中间节点最后到达根节点，其本质仍然是从简单事件中提取复杂事件。

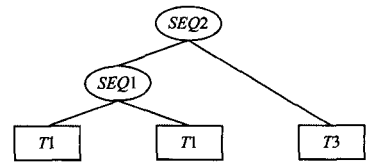


图 3 基于树模型的复杂事件处理

最近,复杂事件的研究集中于 CEP 非功能特性和领域相关的研究,其中包括 CEP 的可靠性[3-5]、可扩展性[6]、用户友好性[7]以及 QoS[8],对 CEP 在 SOA[30]、分布式环境[31-32]、高度动态的移动环境[33]、概率流[22,34-35]下进行应用时提出优化方案,以及商业行为监控[15]、金融服务[16]、虚拟环境中的资源分配[17]、车辆环境[18]、无线传感网络[19-20]、M2M 通信[21]等领域的应用。但是,这些工作都是在检测模型研究的基础上做扩展与优化,所采用的检测模型并没有根本改变,因此也无法支持事件层次的实现。这导致非功能特性和领域相关应用局限于较为底层的数据处理而无法实现更为抽象的业务逻辑。

总体而言,无论以 NFA 还是以树作为检测模型,都侧重于从简单事件中提取复杂事件,所采用的检测模型也只适用于简单事件的模式匹配,不提供从复杂事件中提取更高层次复杂事件的支持;其他工作由于都是建立在检测模型的工作之上,没有从根本上改变匹配的方式,因此也无法实现从低层复杂事件流提取高层复杂事件。本文的工作建立在 NFA 模型的基础上,同时借鉴了树模型层次化的特点,以层次化的 NFA 即非确定层次自动机(Nondeterministic Hierarchical Automata,NHA)为模型,以 EH-Tree 为匹配结构,来对事件层次进行层次化的模式匹配,从而实现事件层次。

3　事件模型

3.1　事件层次

定义 1(事件,event)　事件是有意义的状态变化,通常是记录动作或者活动发生的元组,记作 $E=\langle \bar{a};t_0,t_1\rangle$。其中,$\bar{a}=(a_1,\cdots,a_n)$是事件各属性对应的数据值,是事件的开始时间戳($t_0$)以及结束时间戳($t_1$)的时间值。

定义 2(简单事件,simple event)　简单事件是不能再分的事件,具有原子性和瞬时性,$e.t=e.t_0=e.t_1$。

定义 3(复杂事件,complex event)　复杂事件是由其他成员事件(包括简单事件和复杂事件)抽象得到的事件。复杂事件实际上表示了它的一组成员事件。

定义 4(单层复杂事件,single-layer complex event)　如果复杂事件的所有成员事件都是简单事件,则称该复杂事件为单层复杂事件。

定义 5(多层复杂事件,multi-layer complex event)　如果复杂事件的成员事件不仅有简单事件,还有复杂事件,则称该复杂事件为多层复杂事件。

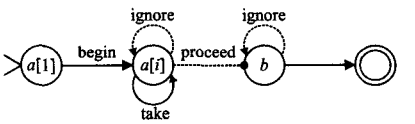
定义 6(事件层次,event hierarchies)　事件层次定义了一组事件的层次模块以及如何从低层次事件模块抽象成为高层次事件模块的事件模式抽象规则,其中包含了两个方面的内容:层次模块及各层次上的事件类型,以及层次之间的事件抽象规则。

3.2　单层复杂事件处理

目前,复杂事件处理系统所采用的检测模型以 NFA 为主。在基于 NFA 模型的复杂事件检测机制中,编译器将每个查询语句编译成带有对应约束条件的 NFA。形式上,将其用于处理复杂事件的自动机 $A=(Q,E,\theta,q_1,F)$,其中,A 是自动机,Q 是状态的集合,E 是有向边的集合,θ 是有向边上状态转移函数的集合,q_1 是初始状态,F 是接受状态。自动机的每一个状态只处理简单事件,因此 NFA 模型只支持从简单事件流中提取复杂事件。

以下 SASE 查询语句表示股票成交量大于 1000,经过一段时间价格呈上升或保持相对平稳时,成交量骤减的情况。查询语句对应的 NFA 如图 4 所示。

```
PATTERN SEQ(Stock+ a[ ],Stock b)
WHERE skip_till_next_match(a[ ],b) {
    [symbol]
    and a[1]. volume > 1000
    and a[i]. price > avg(a[.. i−1]. price)
    and b. volume < 80% * a[a. LEN]. volume }
WITHIN 1 hour
```



State	Edge	Transition Function
$a[1]$	Begin	$a[1].\text{volume}>1000$
$a[i]$	Take	$a[i].\text{symbol}=a[1].\text{symbol}\wedge a[i].\text{volume}>\text{avg}(a[..i-1].\text{price})\wedge a[i].\text{time}<a[1].\text{time}+1\text{ hour}$
	Ignore	$\neg(a[i].\text{symbol}=a[1].\text{symbol}\wedge a[i].\text{volume}>\text{avg}(a[..i-1].\text{price}))\wedge a[i].\text{time}<a[1].\text{time}+1\text{ hour}$
	Proceed	$\theta_{b\_\text{begin}}\vee(\neg\theta_{a[i]\_\text{take}}\wedge\neg\theta_{a[i]\_\text{ignore}})$
b	Begin	$b.\text{symbol}=a[1].\text{symbol}\wedge b.\text{volume}<80\%*a[a.\text{LEN}].\text{volume}\wedge b.\text{time}<a[1].\text{time}+1\text{ hour}$
	Ignore	$\neg(b.\text{symbol}=a[1].\text{symbol}\wedge b.\text{volume}<80\%*a[a.\text{LEN}].\text{volume})\wedge b.\text{time}<a[1].\text{time}+1\text{ hour}$

图 4　查询语句及对应的 NFA 模型

3.3　多层复杂事件处理

传统的 NFA 模型的每个状态都只处理简单事件,因此 NFA 模型只支持从简单事件流中提取复杂事件。事件层次的实现,除了需要支持从简单事件流中提取复杂事件,还需要支持从低层复杂事件中提取高层复杂事件。为了解决这个问题,本文引入(非确定)层次自动机作为检测模型(见图 5)。

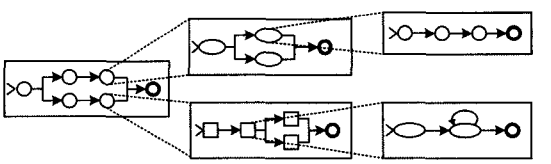


图 5　层次自动机的层次结构

形式地,非确定层次自动机的定义如下[36]。

定义 7((非确定)层次自动机)　非确定层次自动机 K 是模块(module)的元组$\langle K_1,\cdots,K_n\rangle$,每个模块 K_i 由以下几部分组成:

(1)节点(node)的有限集合 N_i。

(2)超节点(box)的有限集合 B_i(集合 N_i 和 B_i 互不相交)。

(3)一个入口节点(entry node)$in_i\in N_i$。

(4)一个出口节点(exit node)$out_i\in N_i$。

(5)索引函数(indexing function)$Y_i:B_i \mapsto \{i+1,\cdots,n\}$,把模块 K_i 的每一个超节点映射到大于 i 的模块上。如果 $Y_i(b)=j$,b 是模块 K_i 的一个超节点,那么 b 可以看作是模块 K_j 的引用。

(6)边关系(edge relation)E_i,E_i 的每一条边都是一对(u,v),u 称为源,v 称为汇。源 u 要么是 K_i 的一个节点,要么是一对(w_1,w_2),其中 w_1 是模块 K_i 的一个超节点,$Y_i(w_1)=j$,w_2 是 K_j 的一个出口节点;而汇 v 是 K_i 的一个节点或者超节点。

NHA 的各模块 K_i 对应事件模型中的事件层次模块,由一个查询语句编译而成。不同模块上的节点与超节点对应的是复杂事件在不同事件层次上映射的成员事件的集合,其中节点 N_i 表明该集合中的事件为简单事件,超节点 B_i 表明该集合中的事件为复杂事件。传统的 NFA 模型中只有节点没有超节点,而节点只处理简单事件,因此 NFA 模型只能处理简单事件而无法实现分层复杂事件检测。模块的入口节点 in_i 表示该模块对应的复杂事件模式匹配的开始,出口节点 out_i 表示该模块对应的复杂事件被检测出来。索引函数 $Y_i:B_i \mapsto \{i+1,\cdots,n\}$把模块的每一个超节点映射到索引值更大的另一个模块,也即把高层复杂事件映射到其低层匹配上,从而构造出层次状的结构。因为不同超节点可以引用相同的模块,所以同一个模块可能出现在不同的上下文中。边关系 E_i 则表明了状态的转移关系,反映了复杂事件被匹配的过程。

4 EH-Tree 的结构

NHA 模型是层次化的 NFA 模型,继承了 NFA 模型理论完备的优点,同时克服了 NFA 模型只支持从简单事件流中提取复杂事件的缺点,能够实现分层复杂事件检测。但传统的 NHA 结构应用于事件层次模式匹配时存在如下几个问题:1)上下文环境的不同会导致模块中约束条件的差异,传统 NHA 结构中不同超节点可以引用相同的模块,难以表达具体约束条件的差异;2)循环引用(cycle detection)问题需要额外进行检测;3)结构复杂,影响模式匹配的效率。为了解决这些问题,本文设计了基于 NHA 模型的多叉树结构 EH-Tree,示例 EH-Tree 结构如图 6 所示。

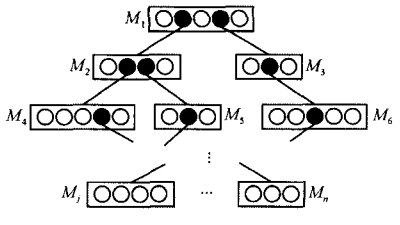


图 6 基于 NHA 模型的 EH-Tree 结构

4.1 模块

EH-Tree 结构是模块的元组$\langle M_1,\cdots,M_n\rangle$,NFA 模型可以看作是 $n=1$ 的 EH-Tree(示例 EH-Tree 结构的具体属性见表 1)。与传统 NHA 结构不同,EH-Tree 除根模块的每一个模块 $M_i(1<i\leqslant n)$,都有且仅有一个超节点指向。若在构造时发现该模块已有超节点指向,则复制该模块作为新的模块。

表 1 EH-Tree 的模块、(超)节点及索引函数

模块	节点	超节点	索引边
M_1	a_{11},a_{13},a_{15}	a_{12},a_{14}	$IE_1(a_{12})=2,IE_1(a_{14})=3$
M_2	a_{21},a_{24}	a_{22},a_{23}	$IE_2(a_{22})=4,IE_2(a_{23})=5$
M_3	a_{31},a_{33}	a_{32}	$IE_3(a_{32})=6$
M_4	$a_{41},a_{42},a_{43},a_{45}$	a_{44}	
M_5	a_{51},a_{53}	a_{52}	
M_6	$a_{61},a_{62},a_{64},a_{65}$	a_{63}	

EH-Tree 结构的各模块是一个七元组 $M_i=(N_i,B_i,TE_i,\theta_i,IE_i,in_i,out_i)$,其中,$M_i$ 是 EH-Tree 模块,N_i 是普通节点的集合,B_i 是超节点的集合,TE_i 是转移边(transition edge)的集合,θ_i 是转移边上状态转移函数的集合,IE_i 是索引边(indexing edge)的集合,in_i 是入口节点,out_i 是出口节点。

除了七元组,每一个模块都有模块头,用以标识模块的入口节点,并记录模块的基本信息,如模块标识、前趋模块中的相应超节点、节点的数目等。

4.2 节点

EH-Tree 的节点分为叶子节点($a_{ij}\in N_i$)和超节点($a_{ij}\in B_i$),并且两集合互不相交。叶子节点和超节点分别处理不同类型的事件,其结构如图 7 所示。

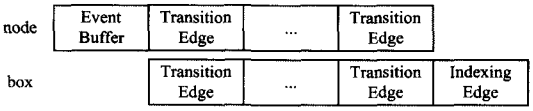


图 7 EH-Tree 的节点与超节点的结构

EH-Tree 的叶子节点用于处理简单事件。叶子节点有对应分区激活实例栈(Partitioned Active Instance Stack,PAIS),会在内存中缓存符合条件的事件以用于后续处理。

EH-Tree 的超节点对应自动机的超节点,用于处理复杂事件。除了顶层模块 M_1,在其他模块 $M_i(i>1)$中,超节点 a_{ij} ($a_{ij}\in B_i$)与索引值大于超节点所在模块的模块$\{M_k \mid IE_i(a_{ij})=k\}$一一对应。超节点并不直接处理简单事件,因此没有对应实例栈,不对事件进行缓存,只根据其他模块传入的中间结果来改变运行实例的状态。当需要回溯事件时,从相应的普通节点中获取事件,从而节省中间结果存储所需的内存空间。

由于自动机不同节点之间存在时序关系,即 $a_{ij}\prec a_{ik}$,$j<k$,因此 EH-Tree 的同一模块节点之间存在时序关系。每个模块的第一个节点是入口节点,$a_{i1}=in_i$,表明该模块模式匹配的开始;最后一个节点是出口节点,当到达出口节点则表明该模块成功匹配,通知订阅者,如果有前驱模块则报告前驱模块并把结果传递给前驱模块。

4.3 边

模块 M_i 内部的节点和超节点是高内聚的,因此通常在同一 EPA,可以保证频繁的数据交换能够集中处理。而模块之间是低耦合的,数据交换只在达到超节点或者出口节点时才进行。

EH-Tree 用边来表示层次自动机中的边关系与索引函数,分别为转移边 TE_i 和索引边 IE_i。转移边与索引边的差异如表 2 所列。索引边作为事件通道连接两 EPA。普通节点只有转移边,超节点有转移边和索引边,且索引边具有更高的优先级。

表 2　EH-Tree 结构中转移边与索引边的差异

	转移边	索引边
出发状态	N_i/B_i	B_i
指向状态	N_i/B_i	$in_j, i<j$
类型	begin, take, ignore, proceed	indexing
数量	≥1	1
是否缓存事件	是	否

转移边 TE_i 对应层次自动机中的边关系，表明节点的顺序以及节点间的转移关系，每一条转移边都有其对应的转移函数 θ_i 用于对流向下一节点的事件做约束。在模式匹配时，引擎将根据转移边及对应的转移函数对事件进行过滤、缓存等操作，并更新当前运行实例的状态。除了出口节点，每一个节点和超节点有指向下一个节点的转移边。出口节点是模块的最后一个节点，因此没有指向下一个状态的转移边。

层次自动机模型中索引函数把模块 M_i 的每一个超节点映射到模块 M_j $(i<j)$ 上，EH-Tree 的索引边 IE_i 对应索引函数，把超节点映射到与其对应的模块。每一个超节点 a_{ij} $(a_{ij} \in B_i)$ 都有且仅有一条索引边。索引边从超节点 a_{ij} 出发，指向该超节点所引用的下层模块 M_j，$Y_i(a_{ij})=j$，当模块 M_j 匹配成功后即通知模块 M_i 改变运行实例的状态。由于超节点和其引用的模块是映射关系，因此索引边一般没有对应的转移函数，但是在模型构建阶段可以通过把约束条件前置的方式给索引边附加约束条件，提前对事件进行过滤，从而减少产生的运行数，提高模式匹配的效率。

除了引导模块的转移，索引边 IE_i 还能记录模块 M_j 的前驱模块（precursor module），即指向模块 M_j 的超节点 a_{ij} $(a_{ij} \in B_i)$ 所在模块 M_i，$Y_i(a_{ij})=j$。当模块 M_j 的某运行实例成功匹配或者终止（超出时间窗口或匹配失败）时，其所有前驱模块的对应的运行实例都必须删除。

5　基于 EH-Tree 的模式匹配算法

在基于 NFA 的模式匹配中，模式匹配的过程实际上是管理和维护自动机的运行实例（run）集的过程。NFA 的一个运行实例是指一个 NFA 基于事件流之上的执行实例，用于维护 NFA 由于其非确定性而产生的每一个分支。NFA 的一个运行实例由以下 3 项唯一确定：被选择放入缓冲区中的事件序列、事件在缓冲区中对应的栈（即事件被选择时 NFA 所处的状态）以及当前 NFA 所处的状态。在基于 EH-Tree 的模式匹配中，模块间的跳转以及跳转时中间结果的处理都通过运行实例的创建与删除以及状态的更新来实现。

5.1　时间窗口约束

基于 NFA 模型的算法只需要考虑当前简单事件的时间戳 $e.t$（$e.t=e.t_0=e.t_1$）、运行实例的开始时间 $r.t_{begin}$ 以及时间窗口 T，只要保证 $e.t \leqslant r.t_{begin}+T$ 就能保证运行实例没有超出时间窗口。

但在多层复杂事件模式匹配中，只能保证运行实例没有超出当前模块的时间窗口，即局部时间窗口 $e.t \leqslant r.t_i+T_i$，并不能保证运行实例没有超出全局时间窗口。如图 8 所示，运行实例的生存时间不应超出 t_2+T_2。

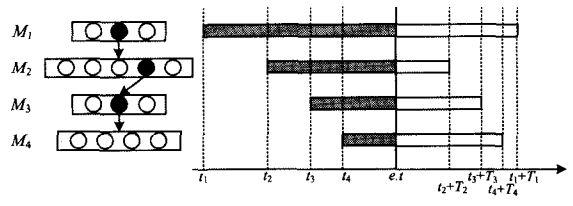


图 8　各模块实例的开始时间与时间窗口

假设模块 M_p 的跳转关系为 $\langle M_1, M_2, \cdots, M_n \rangle$，各模块的时间窗口分别为 $T_1, T_2, \cdots, T_n$，且 $T_1 \leqslant T_2 \leqslant \cdots \leqslant T_n$。此时事件 e 发生，其时间戳为 $e.t$。对于已经跳转到模块 M_n 的运行实例 r，其在模块 $M_1, M_2, \cdots, M_n$ 的开始时间分别为 $t_1, t_2, \cdots, t_n$，判断 e 是否超出时间窗口的条件应为：

$$\begin{cases} e.t \leqslant t_n + T_n \\ e.t \leqslant t_{n-1} + T_{n-1} \\ \quad \vdots \\ e.t \leqslant t_1 + T_1 \end{cases}$$

即 $e.t \leqslant \min\{t_n+T_n, t_{n-1}+T_{n-1}, \cdots, t_1+T_1\}$。当事件 e 在运行实例 r 中超出全局时间窗口，除了要在模块 M_n 的运行实例集中删除 r 外，还需要通知前驱模块删除其前驱运行实例。

5.2　模式匹配算法

与基于 NFA 的模式匹配算法不同，基于层次自动机的模式匹配在查询构建阶段构建的是具有层次结构的自动机（即层次自动机），因此在进行模式匹配时需要考虑模块间的跳转以及跳转时中间结果的处理。分层复杂事件检测算法的描述如算法 1 所示。

算法 1　分层复杂事件检测算法（Hierarchical Complex Event Detection Algorithm，HCED）

```
Input: e: event, R: runs set
Output: R': updated runs set
1. for r∈R do
2.   if checkTimeWindow (e,r) then
3.     if evaluatePredicate (e,r) then
4.       r. buffer ← r. buffer+e
5.       r. state ← r. state. TE
6.       if r. state is box then
7.         create a new run r'
8.         r'. state ← r. state. IE
9.         r'. preRun ← r
10.        R←R+r'−r
11.      else if r. state is accept-state then
12.        if r has precursor run then
13.          r''←r. preRun
14.          R← R+r''−r
15.        else then
16.          create a new match m according to r
17.          R← R−r
18.        end if
19.      end if
20.    end if
21.  end if
```

22. end for
23. R′←R

当一个简单事件 e 发生时，模式匹配引擎将对运行实例集 R 中的每一个运行实例 r 进行更新：首先对 r 进行时间窗口约束判断，并对 r 当前状态的边进行谓词的评估，如果 r 没有超出时间窗口而且能够进行跳转，则把 e 放入 r 的缓冲区中并把当前状态设置为下一个状态(第 4－5 行)。此时将根据 r 的当前状态是叶子节点、超节点还是接受状态进行不同的处理。

跳转后，如果 r 的当前节点是叶子节点，则不需更多处理，等待下一个事件到达。

如果当前节点是超节点，则必须跳转到低层模块进行匹配：先新建一个基于低层模块的运行实例 r'，其当前状态是超节点所引用的低层模块的入口节点，并把 r' 加入运行实例集 R 中；另外由于 r 到达超节点，在低层模块匹配完成前都处于停滞的状态，因此在 r' 的 preRun 中保存 r 的状态，并把 r 从运行实例集 R 中删除(第 7－10 行)。

如果当前节点是接受状态，则说明该模块已经匹配完成，如果此时记录中有高层模块的运行实例 preRun，则说明目前匹配的只是整个事件层次的中间结果(根据需要通知订阅者)，还需要继续进行高层模块的后续匹配，即把高层模块 r'' 加入运行实例集 R 中，并从运行实例集中删除已经匹配完成的运行实例 r(第 13－14 行)；如果记录中没有高层模块的运行实例，则说明已经到达根模块，最高层次的复杂事件已经被检测，整个匹配完成，把 r 的必要信息进行输出并把 r 从运行实例集 R 中删除(第 16－17 行)。

最后，保存更新后的运行实例集 R'，并等待下一个事件到达。

5.3 算法复杂度分析

批处理算法中输入规模通常为输入的元素个数，但是流处理算法的事件流规模是无限的，用整个事件流大小作为算法的输入规模并不适合。为了分析复杂事件处理的时间与空间开销，首先引入事件窗口(partition window)的概念，其表示在指定时间窗口下一个运行实例需要考虑的所有事件，以此作为算法的输入规模。若用 T 表示时间窗口，C 表示有相同时间戳的最大事件数，p 表示某事件被选中的概率，那么事件窗口的大小 $W=TCp(W\in\mathbb{N})$。算法复杂度分析研究当事件窗口增大时时间开销和空间开销的变化情况。从 HCED 算法的第 1 行可以看出，对事件窗口中各个事件进行处理的时间取决于当前同时存在的运行实例的数量。

对于状态序列为 $q_1,q_2,\cdots,q_{m+1}(m>1)$ 的 NFA 模型，事件 e 开启的运行实例的最大数量 $\tilde{r}_e=r_{q_1}(W_1)r_{q_2}(W_2)\cdots r_{q_m}(W_m)$，且 $\sum_{i=1}^{m}r_{q_i}(W_i)=W$，其中 W_i 是在状态 q_i 读取的事件数，$r_{q_i}(W_i)$ 是当读取 W_i 个事件时在状态 q_i 分裂出来的运行实例的数量，其值与 W_i 可能为线性关系或指数关系。显然 $\tilde{r}_e\leqslant|\max_{i=1}^{m}r_{q_i}(W_i)|^m$。因此，当事件序列 $e_1,\cdots,e_W$ 发生时，一个 NFA 的运行实例总数最大为 $W|\max_{i=1}^{m}r_{q_i}(W_i)|^m$[14]。

该运行实例计算模型可以扩展到模块为 $\langle M_1,\cdots,M_n\rangle$ 的 EH-Tree 中，模块 M_j 有普通节点 m_j 个，每个模块的最大运行实例数为 $W_j|\max_{i=1}^{m}r_{q_{ji}}(W_{ji})|^{m_j}$，记 $a=\max_{j,i}r_{q_{ji}}(W_{ji})$，则最大运行实例总数为 $I_{EH-Tree}=\sum_{j=1}^{n}W_ja^{m_j}$。与之对应的 NFA 模型状态序列为 $q_1,q_2,\cdots,q_{m+1}$，其中 $\sum_{j=1}^{n}m_j=m$，$\sum_{j=1}^{n}W_j=W$，其最大实例数 $I_{NFA}=Wa^m$。

为了探讨层数和实例数之间的关系，对问题进行简化。假设 EH-Tree 中每个模块只有 1 个超节点，则模块数即为层数，普通节点数都相同，$m_j=b$，EH-Tree 模型的最大实例数是 $a^b\sum_{j=1}^{n}W_j$，进一步假设各层的时间窗口 T、有相同时间戳的最大事件数 C 以及事件被选中的概率 p 相同，则各模块的事件窗口相同，$W_j=w$。EH-Tree 的最大实例数是 na^bw，这说明实例数随层数增加呈线性增长。

5.3.1 时间开销分析

文献[29]提出单个事件的时间开销主要由谓词约束判断时间和封装复合事件时间构成。在 EH-Tree 结构中，谓词约束判断的时间即为处理运行实例的时间，封装复合事件时间主要为从高层模块跳转到低层模块时中间结果处理的时间，以及从低层模块跳转到高层模块时中间结果处理的时间，即模块跳转时间，因为模块间跳转需要对运行实例进行相应处理，即 $C=C_{process}+C_{call}+C_{return}$。

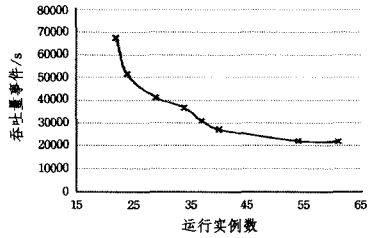


图 9 运行实例数与事件处理吞吐量的关系

处理运行实例的时间开销分析。从图 9 可以看出，随着运行实例数 I 的增加，处理引擎的吞吐量(处理事件总数/总处理时间)减少，大致现反比的关系，即事件平均处理时间 T_e(总处理时间/处理事件总数)大致呈正比例关系，$C_{process}=kI$，$k\in\mathbb{N}$，$k>0$。模块为 $\langle M_1,\cdots,M_n\rangle$ 的 EH-Tree 的最大运行实例总数为 $I_{EH\text{-}Tree}=\sum_{j=1}^{n}W_ja^{m_j}$，时间开销为 $C_{process\_EH\text{-}Tree}=k\sum_{j=1}^{n}W_ja^{m_j}$，其对应的 NFA 模型的最大实例数为 $I_{NFA}=Wa^m$，时间开销为 $C_{process\_NFA}=kWa^m$，由数学归纳法可证 $\sum_{j=1}^{n}W_ja^{m_j}\leqslant Wa^m$，$n=1$ 时取等号。因此 $C_{process\_EH\text{-}Tree}\leqslant C_{process\_NFA}$，说明处理运行实例时，EH-Tree 模型比 NFA 模型更具有时间优势。

模块间跳转的时间开销分析。NFA 模型是扁平结构，没有模块间跳转的时间开销。对于 EH-Tree 结构，当模块间通信的开销小于两个结构间处理运行实例的开销的差异，即 $\sum_{j=1}^{n-1}C_{call}+\sum_{j=2}^{n}C_{return}\leqslant C_{process\_EH\text{-}Tree}-C_{process\_NFA}$ 时，使用 EH-Tree 结构比使用非确定有限自动机模型有更高的性能。因此，提高 EH-Tree 结构性能的关键在于在保持层次特性的基础上，尽

量减少模块间通信带来的开销。

假设各模块节点数相同和事件窗口相同，EH-Tree 的最大实例数是 na^bw，处理运行实例的时间开销为 $n \cdot ka^bw$，这说明处理运行实例的时间耗费随着层数的增加呈线性增长。由于模块间通信与所采用的存储结构、算法以及通信效率有密切的关系，在此并不给定具体的计算方法；如果模块间的跳转时间与层数也呈线性关系，则总时间开销随层数增加呈线性增长；如果呈指数甚至幂关系，则总时间开销随层数增加呈指数关系甚至幂关系。

5.3.2　空间开销分析

在空间开销上，EH-Tree 由于共享了部分状态序列，因此于相比传统的 NFA 结构具有更小的空间开销。为了存储所有的运行实例，EH-Tree 模型的最大空间开销为 $W \cdot I_{EH\text{-}Tree} = \sum_{j=1}^{n} W_j^2 a^{m_j}$，NFA 模型的最大空间开销为 $W \cdot I_{NFA} = W^2 a^m$。由数学归纳法可证 $\sum_{j=1}^{n} W_j^2 a^{m_j} \leqslant W^2 a^m$，$n=1$ 时取等号。这说明 EH-Tree 模型比 NFA 模型更具有空间优势。

同样假设 EH-Tree 每个模块只有 1 个超节点，普通节点数都相同，$m_j = b$，各模块的事件窗口相同，$W_j = w$，EH-Tree 的空间耗费为 $n \cdot a^b w^2$，这说明在空间无限的情况下，随着层数的增加，空间耗费呈线性增长。

6　实验结果

6.1　实验准备

为了测试基于 EH-Tree 结构的模式匹配算法和多层结构优化算法的有效性，在原型系统 SASE 的基础上，实现基于 NHA 模型的 EH-Tree 结构，以及 HCED 算法和多层结构优化算法，从而有效保证不同系统之间实验的可比性。实验参数如表 3 所列。

表 3　实验参数

Parameter	Description	Value used
S	Event stream size	5000～25000
P	Probability of meeting constraints	70%
H	Layers of complex event hierarchies	1～5
L	Length of the sequences in each module	3～5
T	Time window size	100～250

本文使用一个事件生成器动态产生时间序列数据。与事件生成有关的参数主要包括数据流的大小 S 以及事件满足查询条件的概率 P。通过改变数据流的大小 S 测试算法的稳定性，即当数据流增大时算法是否能够保持性能不下降。

另外，与实验相关的参数还有复杂事件的层数 H、每个模块序列的大小 L 以及时间窗口 T。复杂事件的层数 H 是多层复杂事件处理的一个重要参数，层数增加使得运行实例数有所下降，同时又导致了层间跳转的开销。模块序列的大小 L 越长，同时存在的运行实例数就越多，缓冲区中的事件也越多，性能也会有所下降。当时间窗口 T 增大时，满足条件的事件数量也增多，存入缓冲区的事件也随之增多。

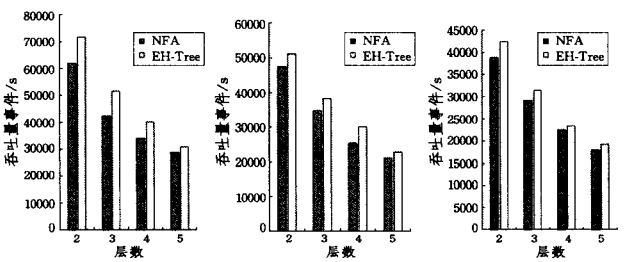
本文实验使用 Java 语言在 Eclipse 中实现。测试机器使用 Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz 的 CPU，5GB 内存，运行的操作系统为 CentOS 6.5 (64-bit)，内核版本为 2.6.32，JDK 版本为 1.7.0_79，Eclipse 版本为 4.5.1。

由于复杂事件处理系统的一个基本特性是实时性，因此实验的主要实验指标是吞吐量(throughput)，即事件流的大小除以处理全部事件所用时间。复杂事件处理引擎处理单一事件的时间非常短暂，事件平均处理时间(处理事件总数/总处理时间)之间微小的差别难以直观地体现系统的性能，因此通常使用吞吐量作为性能衡量的指标，可以看出吞吐量和事件平均处理时间呈反比例的关系，吞吐量越大说明事件处理的时间开销越小。另外，复杂事件处理对事件进行模式匹配使用的是内存空间，如果使用的内存过大则会降低系统性能，甚至涉及内外存调度的问题，因此另一个实验指标是内存占用(memory consumption)。

6.2　实验结果

实验 1　不同层数下 NFA 和 EH-Tree 结构性能和内存对比

在不同层数下，NFA 结构和 EH-Tree 结构的性能对比实验结果如图 10 所示。图 10(a)、图 10(b)、图 10(c)分别为当每个模块的序列长度为 3,4,5 时的实验结果。在每层匹配的序列长度相同的前提下，层数越多两个结构模式匹配的吞吐量越少。层数越多意味着等价的 NFA 结构的序列长度越长，运行实例的数量也越多，模式匹配的吞吐量就随之减少。对于 EH-Tree 来说，每层序列长度一致意味着每层运行实例数的差异不大，层数越多总的运行实例数就越多，吞吐量越少。实验结果也证明了时间开销分析的结论，由于层数增加，运行实例数随层数增加呈线性增长，导致处理运行实例的时间也呈线性增长，吞吐量下降。同时也可以看出，此时模块间跳转时间与层数为线性关系。



(a)序列长度:3　(b)序列长度:4　(c)序列长度:5

图 10　不同层数下 NFA 结构和 EH-Tree 结构的性能对比

对比图 10 可以看出，随着每层需要匹配的序列长度增加，两个结构模式匹配的吞吐量下降。这是因为序列长度越长，模式匹配产生的运行实例数越多，而当事件到达时模式匹配引擎将针对每一个运行实例进行更新，因此更多的运行实例意味着更多的处理时间，从而降低了模式匹配的吞吐量。

无论是每层的序列长度不同还是复杂事件层次的层数不同，EH-Tree 结构都比 NFA 结构的吞吐量大。实验证明了使用 EH-Tree 作为检测结构能够有效降低层次间跳转时的时间开销，使得层次自动机相比 NFA 结构运行实例数更少的特点更为突出，吞吐量也更大。但也可以看出，当层次越来

越多时，相比于 NFA，EH-Tree 结构吞吐量的差异变小，说明模块间通信的开销在增加，从而影响了 EH-Tree 的效率。

在不同层数大小下，NFA 结构和 EH-Tree 结构的内存对比结果如图 11 所示。

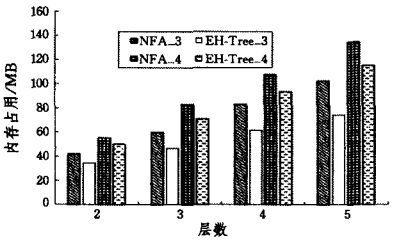
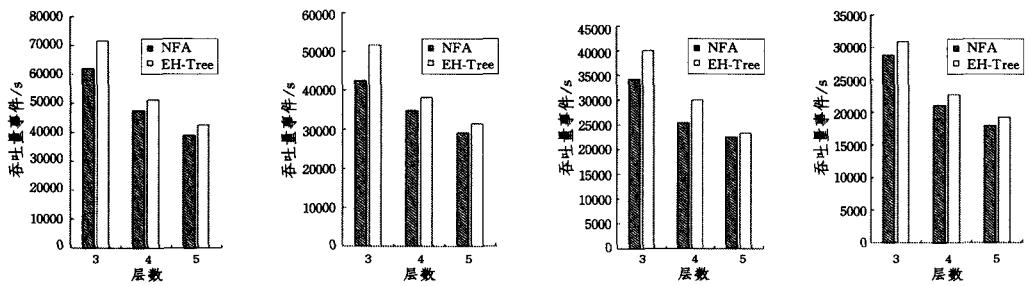


图 11 不同层数下 NFA 结构和 EH-Tree 结构的内存对比

图 11 示出了当序列长度为 3，4 时 NFA 和 EH-Tree 实验的结果。横向来看，每层需要匹配的序列长度越长，层数越多，模式匹配所占用的内存越多。根据空间开销分析，层数增加，运行实例数随层数增加呈线性增长，导致内存呈线性增长，实验结果也证实了这一观点。另外，序列长度越长，模式匹配产生的运行实例数越多，缓存的事件也越多，占用的内存也增加。层数变多对两种结构来说都意味着运行实例数增加，占用的内存增多。

实验 2 不同序列长度下 NFA 和 EH-Tree 结构性能和内存对比

在不同序列长度下，NFA 结构和 EH-Tree 结构的性能对比结果如图 12 所示。图 12(a)—图 12(d)分别为当层数为 2,3,4,5 时实验的结果。在层数相同的情况下，每个模块的序列长度越长，产生的运行实例数越多，进行处理的时间就越长，吞吐量就越小。对比图 12，层数越多两个结构模式匹配的吞吐量越少。在相同序列长度的条件下，层数越多运行实例数就越多，EH-Tree 结构匹配的效率就越低；同时，层数越多，与 EH-Tree 等价的 NFA 结构的序列长度越大，产生的运行实例数越多，吞吐量越小。在相同层数不同序列长度，或者相同序列长度不同层数时，EH-Tree 都比 NFA 结构有更大的吞吐量，这也说明了 EH-Tree 的层次结构比扁平的 NFA 结构具有更高的效率。



(a)层数:2 (b)层数:3 (c)层数:4 (d)层数:5

图 12 不同序列长度下 NFA 结构和 EH-Tree 结构的性能对比

在不同序列长度下，NFA 结构和 EH-Tree 结构(层数为 2,3)的内存对比结果如图 13 所示。序列长度的增大会导致所产生的运行实例数增加，为了保存运行实例的信息而产生的处理内存也增加，同时缓存的事件也增加，使用的内存空间随之增加。在不同序列长度下，EH-Tree 能保持比 NFA 更少的内存，这是因为层次结构能够降低运行实例数，减少了处理运行实例所占用的内存，另一方面也减少了缓存的事件，降低了内存占用。

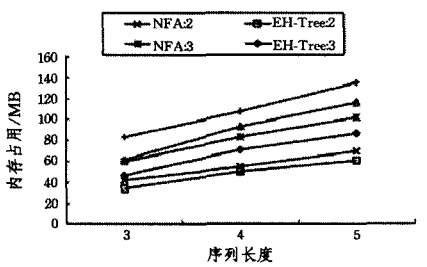


图 13 不同序列长度下 NHA 模型与 NFA 模型的内存对比

实验 3 不同时间窗口下 NFA 和 EH-Tree 结构性能和内存对比

在不同时间窗口下，NFA 结构和 EH-Tree 结构(层数分别为 2,3,4,5)的性能对比结果如图 14 所示。在同一时间窗口下，层数越大，模式匹配的吞吐量就越小，因为层数增大会增加运行实例数，从而增加模式匹配的时间开销。在不同时间窗口下，EH-Tree 结构比 NFA 结构具有更高的吞吐量。在相同的时间窗口下，EH-Tree 由于结构的原因，会产生更少的运行实例，因此有更少的模式匹配开销，同时 EH-Tree 结构的模块跳转开销较低，总体的时间开销比 NFA 结构要低，因此能够比 NFA 结构有更高的吞吐量。

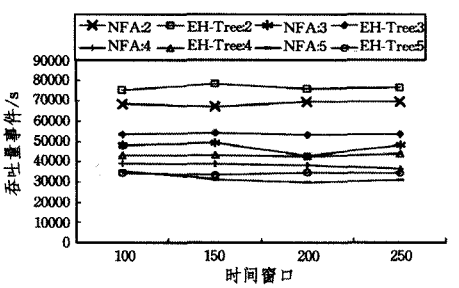


图 14 不同时间窗口下 NFA 结构和 EH-Tree 结构的性能对比

在不同时间窗口下，NFA 结构和 EH-Tree 结构(层数分别为 2,3,4,5)的内存对比结果如图 15 所示。在不同时间窗口下，EH-Tree 结构都比 NFA 结构占用更小的内存。说明相对于 NFA 结构，EH-Tree 结构由于产生更少运行实例，缓存更少的事件，从而占用更少内存的优势并不受时间窗口大小的影响。

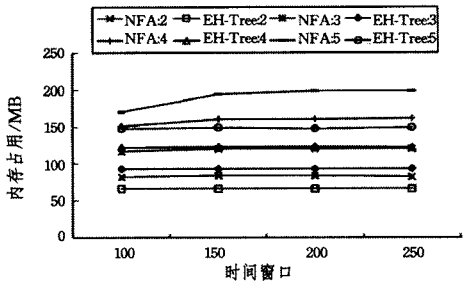


图 15　不同时间窗口下 NFA 结构和 EH-Tree 结构的内存对比

实验 4　不同事件流下 NFA 和 EH-Tree 结构性能和内存对比

在不同事件流大小下，NFA 结构和 EH-Tree 结构(层数分别为 3,4,5)的性能对比结果如图 16 所示。在同一事件流大小下，序列长度越长，模式匹配的吞吐量就越小，因为序列长度增大会增加运行实例数，从而增加模式匹配的时间开销。不同事件流下，EH-Tree 结构能够保持比 NFA 结构具有更高的吞吐量。EH-Tree 的层次结构使得其产生的运行实例数比扁平的 NFA 结构少，在进行模式匹配时减少了时间开销。随着事件流的增大，EH-Tree 结构和 NFA 结构的吞吐量都能保持平稳甚至略有增大，说明 EH-Tree 和 NFA 都能在事件流增大的情况下保持稳定的性能。

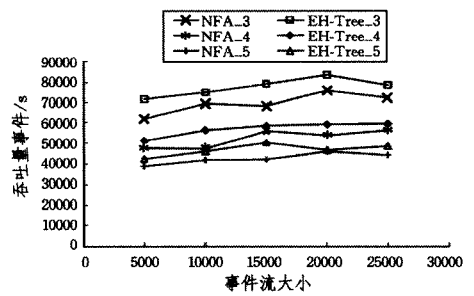


图 16　不同事件流下 NFA 结构和 EH-Tree 结构的性能对比

在不同事件流大小下，NFA 结构和 EH-Tree 结构(层数分别为 3,4,5)的内存对比结果如图 17 所示。随着事件流的增大，两种结构的内存都在增长，因为事件流增大使得满足条件的事件增多，从而存入缓冲区的事件也增多，使得内存增长。但在相同事件流的情况下，EH-Tree 结构比 NFA 结构内存更低，说明 EH-Tree 的层次结构比 NFA 的扁平结构有更高的效率。

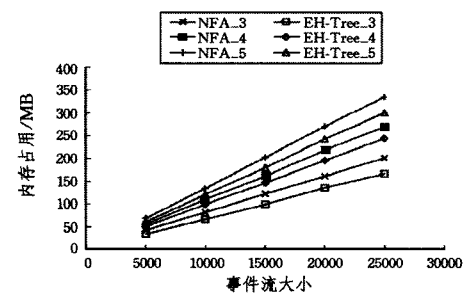


图 17　不同事件流下 NFA 结构和 EH-Tree 结构的内存对比

结束语　本文针对目前的复杂事件处理系统无法处理事件层次的缺陷，在传统 NFA 模型的基础上，提出了能够支持事件层次实现的模型——层次自动机，通过引入超节点和索引函数的概念提供多层复杂事件实现的支持。但是由于传统的层次自动机模型结构复杂，在对事件进行模式匹配时效率不高，因此本文提出了基于层次自动机模型的 EH-Tree 结构，以及基于 EH-Tree 结构的多层复杂事件模式匹配算法，并分析了算法复杂度。

本文首次将复杂事件处理领域“事件层次”从概念变为现实，并且证明了事件层次是可行甚至高效的。本文工作引出了未来基于“事件层次”的复杂事件处理技术的发展方向，但本文工作仍有不足之处：1)不同领域的业务系统有不同的信息需求，如何建立事件层次来支持业务的处理是一个值得研究的问题；2)本文只在参数方面测试了算法的稳定性，对于整个系统而言，事件层次的建立是否影响了系统的鲁棒性、纠错性、并行性、扩展性等非功能特性还需要进一步的工作才能确定；3)事件层次的算法本身仍然有很多优化的空间，如多查询优化、概率流模式匹配等都有待研究。

参 考 文 献

[1] LUCKHAM D. The power of events[M]. Reading: Addison-Wesley, 2002.

[2] CUGOLA G, MARGARA A. Processing flows of information: from data stream to complex event processing[J]. Acm Computing Surveys, 2011, 44(3): 359-360.

[3] BALAZINSKA M, BALAKRISHNAN H, Madden S R, et al. Fault-tolerance in the Borealis distributed stream processing system[J]. ACM Transactions on Database Systems (TODS), 2008, 33(1): 3-44.

[4] O'KEEFFE D, BACON J. Reliable complex event detection for pervasive computing[C]// Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems. ACM, 2010: 73-84.

[5] ZHANG H, DIAO Y, IMMERMAN N. Recognizing patterns in streams with imprecise timestamps[J]. Information Systems, 2013, 38(8): 1187-1211.

[6] ZAPPIA I, PAGANELLI F, PARLANTI D. A lightweight and extensible Complex Event Processing system for sense and respond applications[J]. Expert Systems with Applications, 2012, 39(12): 10408-10419.

[7] BOUBETA-PUGI J, ORTIZ G, MEDINA-BULO I. A model-driven approach for facilitating user-friendly design of complex event patterns[J]. Expert Systems with Applications, 2014, 41(2): 445-456.

[8] GU Y, YU G, LI C. Deadline-aware complex event processing models over distributed monitoring streams[J]. Mathematical and Computer Modelling, 2012, 55(3): 901-917.

[9] DEMERS A, GEHRKE J, HONG M, et al. Towards expressive publish/subscribe systems[M]// Advances in Database Technology-EDBT 2006. Springer Berlin Heidelberg, 2006: 627-644.

[10] DEMERS A, GEHRKE J, PANDA B, et al. Cayuga: A General Purpose Event Monitoring System[C]// CIDR. 2007: 412-422.

[11] HONG M, DEMERS A J, GEHRKE J E, et al. Massively multi-query join processing in publish/subscribe systems[C]// Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. ACM, 2007: 761-772.

[12] WU E, DIAO Y, RIZVI S. High-performance complex event processing over streams[C]// Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. ACM, 2006: 407-418.

[13] DIAO Y, IMMERMAN N, GYLLSTROM D. Sase+: An agile language for kleene closure over event streams[J/OL]. [2012-12-23]. http://archive.systems.ethz.ch/www.dbis.ethz.ch/education/ws0708/adv_top_infsyst/papers/sase_tr07.pdf.

[14] AGRAWAL J, DIAO Y, GYLLSTROM D, et al. Efficient pattern matching over event streams[C]// Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. ACM, 2008: 147-160.

[15] GREINER T, DÜSTER W, POUATCHA F, et al. Business activity monitoring of norisbank taking the example of the application easycredit and the future adoption of complex event processing (CEP)[C]// Proceedings of the 4th International Symposium on Principles and Practice of Programming in Java. ACM, 2006: 237-242.

[16] ADI A, BOTZER D, NECHUSHTAI G, et al. Complex event processing for financial services[C]// Services Computing Workshops, 2006(SCW'06). IEEE, 2006: 7-12.

[17] VASIĆ N, NOVAKOVIĆ D, MIUČIN S, et al. Dejavu: accelerating resource allocation in virtualized environments[C]// ACM SIGARCH Computer Architecture News. ACM, 2012: 423-436.

[18] TERROSO-SÁENZ F, VALDÉS-VELA M, CAMPUZANO F, et al. A complex event processing approach to perceive the vehicular context[J]. Information Fusion, 2015, 21(1): 187-209.

[19] ZOUMBOULAKIS M, ROUSSOS G. Complex event detection in extremely resource-constrained wireless sensor networks[J]. Mobile Networks and Applications, 2011, 16(2): 194-213.

[20] YAO W, CHU C H, LI Z. Leveraging complex event processing for smart hospitals using RFID[J]. Journal of Network and Computer Applications, 2011, 34(3): 799-810.

[21] BRUNS R, DUNKEL J, MASBRUCH H, et al. Intelligent M2M: Complex event processing for machine-to-machine communication[J]. Expert Systems with Applications, 2015, 42(3): 1235-1246.

[22] WANG Y H, CAO K, ZHANG X M. Complex event processing over distributed probabilistic event streams[J]. Computers & Mathematics with Applications, 2013, 66(10): 1808-1821.

[23] MCCARTHY D R, DAYAL U. The architecture of an active data-base system[J]. Proc. ACM Sigmod Conf. on Management of Data, 1989, 18(1): 215-224.

[24] BABCOCK B, BABU S, DATAR M, et al. Models and Issues in Data Stream Systems[C]// ACM Sigmod-sigact-sigart Symposium on Priniciples of Database Systems. 2002: 1-16.

[25] SHARON G, ETZION O. Event-processing network model and implementation[J]. Ibm Systems Journal, 2008, 47(2): 321-334.

[26] MANSOURI-SAMANI M, SLOMAN M. Monitoring distributed systems[J]. IEEE Network, 1993, 7(6): 20-30.

[27] MANSOURI-SAMANI M, SLOMAN M. GEM: A generalized event monitoring language for distributed systems[J]. Distributed Systems Engineering, 1997, 4(2): 96-108.

[28] MEI Y, MADDEN S. Zstream: a cost-based query processor for adaptively detecting composite events[C]// Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. ACM, 2009: 193-206.

[29] CHENG S J, WANG Y J, MENG Y, et al. PMTree: An efficient pattern matching method for event stream processing[J]. Journal of Computer Research and Development, 2012, 49(11): 2481-2493. (in Chinese)
程苏珺, 王永剑, 孟由, 等. PMTree: 一种高效的事件流模式匹配方法[J]. 计算机研究与发展, 2012, 49(11): 2481-2493.

[30] POTOCNIK M, JURIC M B. Towards complex event aware services as part of soa[J]. IEEE Transactions on Services Computing, 2014, 7(3): 486-500.

[31] CUGOLA G, MARGARA A. Deployment strategies for distributed complex event processing[J]. Computing, 2013, 95(2): 129-156.

[32] JAYASEKARA S, KANNANGARA S, DAHANAYAKAGE T, et al. Wihidum: Distributed complex event processing[J]. Journal of Parallel and Distributed Computing, 2015, s79-80: 42-51.

[33] OTTENWÄLDER B, KOLDEHOFE B, ROTHERMEL K, et al. MCEP: a mobility-aware complex event processing system [J]. ACM Transactions on Internet Technology (TOIT), 2014, 14(1): 1-24.

[34] KAWASHIMA H, KITAGAWA H, Li X. Complex event processing over uncertain data streams[C]// 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC). IEEE, 2010: 521-526.

[35] CUGOLA G, MARGARA A, MATTEUCCI M, et al. Introducing uncertainty in complex event processing: Model, implementation, and validation[J]. Computing, 2012, 97(2): 103-144.

[36] ALUR R. Formal analysis of hierarchical state machines[M]// Verification: Theory and Practice. Springer Berlin Heidelberg, 2003: 42-66.