基于内外存调度的长过程复杂事件检测

王 亚[1] 徐传飞[2] 陈艳格[1]

(许昌学院计算机科学与技术学院 许昌 461000)[1] (东北大学信息科学与工程学院 沈阳 110819)[2]

摘 要 复杂事件检测能够帮助人们从海量数据中获取有用信息,因而近年来受到了广泛关注与研究。在复杂事件检测中,存在着一些面向长过程的复杂事件。由于内存的有限性,完全采用传统的检测技术来检测长过程复杂事件并不可行。为了支持这种长过程复杂事件的检测,依次提出了对象树内存存储结构、PR置换策略、基于事件分类的存储方式、事件实例映射表进行长过程复杂事件的检测。最后通过真实数据集实验,充分验证了提出的策略对于长过程复杂事件检测的可行性及高效性。

关键词 长过程复杂事件检测,对象树,PR置换,事件实例映射表,事件分类存储

中图法分类号 TP311.13 **文献标识码** A

Complex Event Detection for Long Process Based on Out-of-Core

WANG Ya[1] XU Chuan-fei[2] CHEN Yan-ge[1]

(Department of Computer Science and Technology, Xuchang University, Xuchang 461000, China)[1]

(Department of Computer Science and Engineering, Northeastern University, Shenyang 110819, China)[2]

Abstract Complex event detection can get useful information from vast of data, so it has drawn extensive research and attention in recent years. However, there are many complex events occurring in a very long period. Due to the constraint of memory, traditional complex event detection technologies are not suitable for this case. To detect complex events for long process, this essay proposed the policies of object tree as memory storage structure, PR replacement, storage of events by classify, the instance_map table. Finally, the effectiveness and efficiency of our proposed methods for complex event detection were verified through experiments on real data set.

Keywords Complex event detection for long process, Object tree, PR replacement, Instance_map table, Storage of events by classify

1 引言

复杂事件检测技术能够查找、过滤、关联相关事件,从看似无关的事件中发现复杂事件和隐含知识,以支持对隐藏的机遇和威胁做出迅速响应。随着当前的RFID(Radio Frequency Identification)、传感器等数据采集设备(Electronic Data Gathering Equipment, EDGE)的发展,该技术作为一种新兴的热点领域受到了广泛关注与研究。当前已研制出了很多的复杂事件检测产品用于商业活动监控[1]、系统自动交易[2]、欺诈检测[3]等领域。这给人们的生活和生产带来了极大的安全便利性。

传统的复杂事件检测技术大多通过建立某种内存数据结构并采用某种能够减少中间结果的优化策略,来达到对一段较短时间内发生的复杂事件进行检测的目的。然而,在现实生活中,除了这种在短期内即可发生的复杂事件之外,还存在许多复杂事件,从它们开始发生到其完全结束需要花费很长时间。也就是说,人们周围还存在着许多符合长过程特点的事件流。

对长过程复杂事件进行检测,涉及到从复杂事件开始到结束的海量事件。由于检测是在内存中进行的,而内存又是有限的,因此对长过程复杂事件检测存在着存储海量事件的挑战。内存的有限性,决定了必须借助于外存的大容量空间存储事件,并要设计高效的内外存调度的I/O算法。针对这些挑战,本文提出了一系列的策略。主要贡献点如下:

(1)设计内存对象树存储结构。该树是一种基于事件属性共享的结构,将具有相同对象的事件存储到同一棵树,相同对象和相同事件类型的信息只存储一次,从而使有限的内存空间能存储更多的事件实例。

(2)设计PR置换策略。该策略基于事件流的规律性,在内存受限时,将内存中最不会发生检测的对象事件调出到外存中,这样在检测时可以极大地减少I/O的次数。

(3)为了便于对事件的检索,对于调度到外存的事件,本文设计了分类存储的策略,其将事件按属性分类存储到不同的文件中,并以文件名作为索引,文件中只存储时间戳。

到稿日期:2011-10-17 返修日期:2012-02-17 本文受国家自然科学基金(61003058),东北大学创新项目基金(N100604014)和河南省教育厅科技攻关项目(2009B520026)资助。

王 亚(1986—),女,硕士,助教,主要研究方向为复杂事件处理、数据流管理,E-mail:wangya_xcc@126.com;徐传飞(1984—),男,博士生,主要研究方向为不确定数据管理、复杂事件处理;陈艳格(1982—),女,硕士,讲师,主要研究方向为计算机网络。

(4)设计外存事件实例映射表结构以方便查询外存事件，该结构采用比特位对事件在外存文件中的存在与否进行标识，占用空间少，所包含信息量大。

(5)采用真实数据集，设计详实的实验过程，验证了以上提出的一系列策略能对长过程复杂事件进行完整检测并使检测具有较高的效率。

本文第2节介绍复杂事件检测的相关工作；第3节给出抽取的事件模型及一些问题定义；第4节阐述了本文为进行长过程复杂事件检测而提出的一系列技术；第5节描述实验结果；最后总结全文。

2 相关工作

近年来，随着复杂事件检测技术的发展，对于不同的应用需要提出了不同的复杂事件检测模型，其主要有4种：基于有限自动机的、基于匹配树的、基于有向图的和基于Petri网的；并由此产生了基于自动机模型的系统SASE[4]和Cayuga[5]，采用匹配树模型的系统ZStream[6]和Ready[7]，基于有向图模型的系统Sentinel[8]和EVE[9]，现实生活中基于Petri网模型的监控系统HiFi[10]和主动数据库系统SAMOS[11]。

在自动机模型中，一个状态代表一个事件，自动机进入某个可接受的状态时，说明发生了复合事件。匹配树模型是当到达根节点时检测到一个复合事件。有向图模型采用有向无环图来表示复合事件。Petri模型采用跃迁守护函数来说明复合事件的发生。

在这4种模型中，基于自动机和Petri网的模型在复合事件检测时，只能匹配按照顺序到达的基本事件，而基于图或树的模型的过滤又没有考虑基本事件的顺序或者时序距离，因此可能会出现第一个事件没有发生，第二个事件也可能会被过滤的情况，从而导致不必要的开销。这4个模型在实际应用中虽然各有优点，但是它们也都具有一定的局限性。更重要的是其检测过程只在内存中执行，并没有考虑内、外存的调度问题，因此不适合长过程复杂事件的检测。

3 事件模型及问题定义

3.1 事件模型

事件是进行复杂事件检测的对象，是复杂事件检测结果的基本组成部分。为了方便地检测复杂事件，需要建立一种普适性的事件模型，使其可以应用到现实生活的方方面面。

本文从事件的各个应用中抽象出的事件模型为一个三元组：(OID, ET, T)，其中OID是对象ID，即“Object ID”，它可以唯一地标识所观测对象的ID属性；ET表示事件类型“Event Type”，是用户所关心的一种状态的描述；T表示时间戳，即timestamp。那么这个模型表示的语义即是对象ID在T时刻被检测到发生了ET类型的事件。

该事件模型可以应用到不同设备采集的数据上，无论是传感器数据还是RFID数据或者其它数据都可以套用。

定义1(事件实例) 事件的发生就叫事件实例。

如事件A，它的一次发生就称为事件A的事件实例，事件实例通常用小写字母来表示。

定义2(末端事件) 用户定义模式中的最后一个事件就称为末端事件。

如在用户定义模式中，要检测的复杂事件是$A;B;C;D$，那么事件D就是末端事件。与此对应的事件D的一次发生d就是末端事件实例。

定义3(不完全模式匹配实例) 满足部分用户定义的模式的匹配实例，就称为不完全模式匹配实例。

如要查询的模式是$A;B;C;D$，若事件实例c,d满足$C;D$的查询条件，则称$c;d$为不完全模式匹配实例；若有$a;b;c$满足$A;B;C$的查询条件，则$a;b;c$也是不完全匹配模式实例。

定义4(完全模式匹配实例) 满足用户定义的完整模式实例的发生，即称之为完全模式匹配实例。对于定义3中的例子，如果有事件实例$a;b;c;d$满足了模式$A;B;C;D$，则$a;b;c;d$就是完全模式匹配实例。

3.2 基于末端触发的匹配树检测模型

由于事件流的长过程性和内存的有限性，满足用户定义模式的事件实例不可能全部存放在内存中。因此，本文的匹配触发是由末端事件实例触发的。因为只有当末端事件实例到来时，才可能会有满足用户定义的事件模式的序列，此时再回溯查找能够与用户定义的模式匹配的实例，就可以避免将很多无用的实例存储在内存中造成的时间和空间的开销。

自动机和Petri网模型都是基于顺序匹配的处理，并且不支持对非事件的检测。相对来说，匹配树和有向图更为灵活，但是有向图的表现形式不如匹配树简洁，因此本文选用匹配树作为基本的内存检测模型，并基于末端事件触发进行对长过程复杂事件的检测。

如要检测事件模式$A;B;C;D$，当有事件D的事件实例d发生，就触发事件的检测，向前搜索与d有相同对象的事件C的实例发生，如果找到事件C的实例和d之间满足查询条件中时间和值的限制，则构造出一系列$c;d$的不完全模式匹配实例，再对各个不完全模式匹配实例$c;d$依次地查询满足条件的事件B的实例，直至检测到一系列满足条件的完全模式匹配实例$a;b;c;d$。

4 面向长过程复杂事件检测技术

在第3节已经给出了本文的事件模型和检测模型，本节将介绍为进行长过程复杂事件检测所提出的对象树、PR置换策略、外存分类存储策略和事件实例映射表结构，它们都是进行长过程复杂事件检测的技术。在详细介绍这些技术之前，我们先熟悉一下长过程复杂事件处理的整个过程。

4.1 长过程复杂事件处理流程

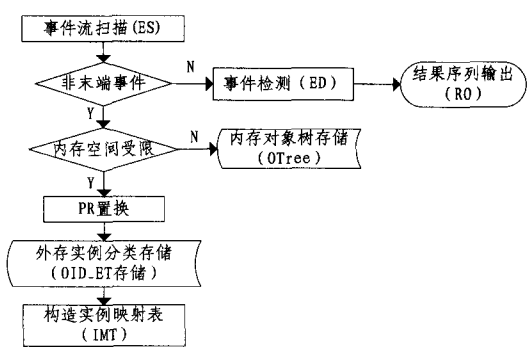

图1 长过程复杂事件处理流程

本文进行长过程复杂事件检测的处理流程由如下几部分组成：事件流扫描(Event Stream Scan，ES)、对象树存储(Object Tree，OTree)、PR置换、外存事件分类存储(Oid and Event Type，OID_ET存储)、事件实例映射表(Instant_Map Table，IMT)构造、事件的检测(Event Detection，ED)和结果

序列(Result Output,RO)的输出。其中,OTree、PR 置换、OID_ET 存储、IMT 是本文提出的用于长过程复杂事件检测的技术。下面通过图 1 所示的流程图来表示事件处理过程的几个步骤之间的关系。

检测的主要步骤解释如下:

事件流扫描(ES):读取事件流中的事件实例。

对象树的存储(OTree):若读取的事件是非末端事件,并且内存未受限,则将事件实例存储到相应的对象树中。

实例的内外存置换(PR 置换):当内存存储满指定数量的事件实例时,将引发事件实例向外存的调度。调度的执行是用本文提出的 PR 置换方法进行的。

外存实例的存储(OID_ET 存储):将调出到外存的事件在外存按照 OID_ET 策略进行存储的过程。

实例映射表的构造(IMT):将事件调出到外存后,在内存中的此事件对象的映射表做相应改变的过程。

事件检测(ED):如果读取的事件实例是末端事件,那么将引发事件检测。本文中所采用的事件检测方法是以末端事件触发,向前匹配的检测,它先通过找出不完全模式匹配的实例,最后得出完全模式匹配实例。

结果序列输出(RO):将满足用户定义模式的序列输出。

这里需要指明的是,文中内存受限指的是根据内存大小规定内存中一共可以存储的事件的数量,当存储的事件超出这个事件数量时,就认为是内存受限。

4.2 对象树

在现实生活的很多应用中,同一个对象 ID 在不同的时间可以有相同的事件发生。如传感器检测室内温度时,定义"温度≤10℃"为 A 事件,地理位置作为事件 ID,则不同的时刻,同一地理位置 ID 可能都会发生 A 事件。将事件完全按照 (OID,ET,T) 的方式存储显然是很浪费空间的。基于此,本文设计了对象树存储事件实例。

对象树是以树的形式存在的用来存储同一个对象发生的所有事件实例的内存结构。树的根结点存储着对象名称,第二层结点存储着对象所发生的事件类型,最后一层的叶子结点存放着一个动态数组(指数组大小可变)用于存储事件发生的时间。

如事件流上有 3 个事件实例,分别为$(1,a,5)$、$(1,a,8)$、$(1,b,10)$,由于这 3 个事件的对象都是 1,发生的事件两个为 a、一个为 b,即它们有着共享的对象和事件结点,因此这 3 个事件实例可以存储在图 2 的对象树中。

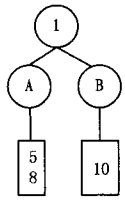

图 2 对象树结构图

内存对象树结构将具有相同对象的事件全部存储到同一棵树中,对于相同的对象和事件类型信息只存储一次,这一方面可以对事件实例的存储起到一定的压缩作用,另一方面也便利了检测的进行,因为一般的检测都是基于相同对象发生的事件的检测。

定义 5(时间块) 时间坐标轴上的一段时间跨度,通常包含固定长度的时间点。

如定义时间块所包含的时间跨度为 20s,则 1s~40s 就是两个时间块。本文用时间块是为了方便进行后续的存储和调度,它若划分得过大,内存中将不能完整存储一个时间块;若划分得过小,一个事件填充了一个时间块,又没有太大意义,会使分块功能失效。可以结合经验和实际情况,适当地人为规定其长度。本文中用 TB_NUM 表示一个时间块的跨度。

4.3 PR 置换算法

事件流到来,事件被存储到不同的内存对象树中。当内存为满时,就面临着要将什么样的事件调出到外存的问题。因为检测在内存中进行,如果检测的事件存储于外存,则会使用 I/O 将外存事件调入内存检测,而 I/O 的开销是很大的,所以当内存受限时,为了保证检测的效率,最好调出那些未来发生检测较少的事件或短时间内不会发生检测的事件,这样就能极大地减少 I/O 的开销,也即是最大地减少检测时从外存将事件调入到内存的次数,这就是本文调度算法的思想。

任何事件的发生都有一定的规律性,如某区域的环境温度变化总是慢慢升高或慢慢下降或保持恒温,这样就可以依据过去的温度变化来预测未来温度的变化情况。但事件的规律性并不具有均等性,即这并不是说很久以前的事件的发生情况和最近发生的事件在预测未来事件的发生上具有均等的概率,一般情况下,最近发生的事件更能预测未来事件的发生。基于过去的事件和最近的事件的发生情况来预测未来事件发生而采取的调度策略就是 PR(Past and Recent)置换策略。

基于事件规律性,当内存受限时,最好是将过去和最近发生检测较少的事件调出到外存,而内存中保留那些频繁发生或短时间内可能会触发匹配的事件,这样就能暂时解决内存空间紧张的问题,有效减少 I/O 的开销。进行 PR 置换需要用到一系列的统计项,本文将这些统计项存储在一个对象统计表中,其结构如图 3 所示。

OID	C_1	C_2	C_3	t

图 3 对象统计表的结构

对象统计表包括对象 OID 项、末端事件的发生次数 C_1、内存中非末端事件的发生次数 C_2、最近 R 个时间块的末端事件的发生次数 C_3 及末端事件的最近发生时间 t。

PR 置换策略可描述如下:

(a)对于 $C_1=0$ 的 OID,将 C_2 较多的事件的对象树调出到外存。

即首先选择调出那些末端事件为 0 的对象树,因为这么长时间它都没有匹配检测,说明此对象可能很长时间内才会引发匹配检测;并优先调出非末端事件 C_2 较多的对象,因为它们更占空间。

(b)对于 $C_1!=0$ 的 OID,计算 $Value=(C_3/R)*w1+[(C_1-C_3)/P]*w2$(其中 P 是过去的 P 个时间块,R 是当前的 R 个时间块);

①若 $Value$ 相差较大,将 $Value$ 值较小的事件对象树调出到外存;

②若 $Value$ 值相等,将最近刚刚发生末端事件的对象树

调出。

Value 值是用过去 P 个和最近 R 个时间块进行统计得到的每个时间块的平均末端事件的发生次数，末端事件发生次数也是匹配检测次数。由于最近 R 个时间块更能预测未来的事件，因此对最近 R 个时间块和过去 P 个时间块分别赋予不同的权重 $w1$，$w2$（$w1>w2$ && $0<w1<1$ && $0<w2<1$）。

在置换过程中，上面的两点是有优先级的。(a)的优先级比(b)高，即如果满足(a)的对象树结构的事件大于等于要调出的 K 个时间块的事件实例，就不再考虑调出按(b)规则决定的要调出的对象树。只有满足(a)规则决定的对象树的事件实例个数不足 K 个时间块的事件实例时，才从(b)规则中调出这些规则决定要调出的对象树。

PR 置换策略是基于事件的规律性而进行的调度。对于对象的末端事件发生较少的事件说明其发生的检测不频繁，优先将其调出到外存，这样就能在一定程序上减少 I/O 的开销，以提高检测的效率。假设当内存受限时要调出 T 个时间块的事件，T 个时间块包含了 *OUT_NUM* 个事件，*PR* 置换算法实现如算法 1 描述。

算法 1 事件实例的 PR 置换算法

输入：事件(oid,e,t)，应调出事件数目 OUT_NUM

输出：置换到外存的树

1. 对于对象统计表结构中的 n 项，N_1，N_2，…，N_n，找出 $C_1=0$ 的项，共 q 项
2. 按 C_2 由大到小排序此 q 项，排序后为结果 N_j，N_{j+1}，…，N_{j+q-1}
3. for(int m=0;m<q;m++)
4. if OUT_NUM−N_{j+m}.C_2>0;
5. 将 N_{j+m}对象调出到外存;
6. OUT_NUM=OUT_NUM−N_{j+m}.C_2;
7. 复位 N_{j+m}.C_2 为 0;
8. else if OUT_NUM−N_{j+m}.C_2<=0 && OUT_NUM > 0
9. 从 N_{j+m}对象树中调出 OUT_NUM 个事件实例
10. N_{j+m}.C_2=N_{j+m}.C_2−OUT_NUM;
11. OUT_NUM=0;
12. if OUT_NUM > 0
13. 计算(n−q)个对象的 value 值
14. 按 value 值从小到大将对象排序成 N_1，…，N_{n-q-1}，N_{n-q}
15. 依次调出 N_1，…，N_{n-q-1}…，直至调出 OUT_NUM 个事件实例

4.4 事件在外存的存储形式——OID_ET 形式

将事件调出到外存是以对象树为单位进行的，而一般的检测也是基于同一对象的。为了检测的便利性，本文将调出到外存的事件按对象和事件类型的不同存储到不同的外存文件中，其文件命名方式为“OID_ET”，即相同的 *OID* 和相同的 *ET* 放到同一文件中，文件中只存储时间戳。这样文件名其实就是对事件的一种索引。如对象树 *OID* 为 2，调出到外存，可存储到如图 4 所示的 OID_ET 文件中。

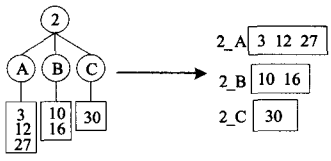

图 4 事件实例存储于外存文件 OID_ET 的实例

用这种分类存储的方式来存储事件实例有两方面的好处：(1)以文件名作为索引，文件中只存储时间戳其实是对事件实例的一种压缩。(2)相同对象相同事件类型的事件存放到同一文件中，便利了检测的进行。

4.5 事件实例映射表

事件流的长过程性，决定了存储于外存的事件亦是海量的。检测时一一地查找外存文件寻找所需事件是很耗时的，因此为了检测外存事件的便利性，本文设计一种事件实例映射表对外存事件进行索引。

事件实例映射表由一个事件类型列、对象 ID 列以及表示时间块的位串向量组成。每种事件类型下包含着各个对象的 ID 对应的位串向量。位串向量是由一系列的二进制位组成的，每个时间块对应着一个二进制位。如果这个时间块对应的二进制值为“1”，则表示此时间块内发生的该 *OID* 的相应事件类型 *ET* 的实例存在于外存文件中；若为“0”，则表示外存文件中不存在此时间块内发生的该 *OID* 的相应事件类型的实例。如当一个 *OID* 为 *id*、一个事件类型 *ET* 为 *e*、时间戳为 *t* 的实例被调出到外存后，事件实例映射表中 *ET* 行 *id* 对象对应的二进制向量的第（$\lceil t/TB\_NUM \rceil$+1）位将被设置成“1”（$\lceil t/TB\_NUM \rceil$+1 是该事件位于的时间块）。

图 5 的事件实例映射表描述了如下信息：一共有 5 个对象，分别为 1、2、3、4、5；3 个事件类型，分别为 A、B、C。在第一时间块，对象 2、3、4 在外存文件上有 A 事件的发生，对象 2、5 在外存文件上有 B 事件的发生，对象 4 在外存文件上有 C 事件的发生。在第二时间块，对象 2、3、4、5 在外存文件上有 A 事件的发生，对象 2、4、5 在外存文件上有 C 事件的发生。

		位串					…	
ET	OID	1	2	3	4	5	…	16
A	1							
	2	1	1					
	3	1	1					
	4	1	1					
	5		1					
B	1							
	2	1						
	3							
	4							
	5	1						
C	1							
	2		1					
	3							
	4	1	1					
	5		1					

图 5 事件实例映射表

事件实例映射表在事件被置换到外存进行存储以及检测时会用到。

当一个事件被调度到外存空间时，首先查找外存事件实例映射表，看是否有存储此事件的外存文件存在，如有，则将事件直接存储到对应的外存文件中；若无，则新建一外存文件将事件存入，并修改实例映射表。其实现见算法 2。

算法 2 基于外存事件实例映射表的存储算法

输入：要调出到外存的事件(oid,e,t)

输出：存储事件的外存文件是否存在标识 flag

1. 查找实例映射表中 oid,e 对应的位串中是否有“1”标识

2. 若有

3. 则将事件(oid,e,t)直接存入"oid_e"文件中

4. 否则

5. 新建一文件"oid_e",将事件(oid,e,t)写入

6. 并计算它所位于的时间块 TB=[t/TB_NUM]+1,将事件实例映射表中 OID、ET 对应的第 TB 位置"1"

检测存储于外存的事件时,通过实例映射表就可以看出外存文件中是否有此事件存在。若 OID、ET 对应的时间块下的位串不全为"0",说明外存文件"OID_ET"中存储有要寻找的 *ET* 类型的事件。否则,表示外存文件没有所要检测的事件。

综上,长过程复杂事件检测在内存空间不足时,通过 PR 置换算法将调出 *OUT_NUM* 个事件,并且采用基于外存事件实例映射表的存储算法将事件调度到外存文件,该流程的性能将在下一节进行评估。

5 实验分析

本文以真实数据集进行实验,验证为进行长过程复杂事件检测而提出的一系列方法的有效性。实验数据集来自 Intel 伯克利实验室布置的 54 个无线传感器[12]。每个事件包含传感器编号、时间戳、温度等属性。本文选择将温度作为检测的事件,基于温度值将其划分为不同的事件类型。如 A 事件类型代表"检测温度<19℃"。数据集中有一年的数据,一小时为 3600s,平均每 36s 产生一个事件。所有实验都在 PC 机上执行,CPU 配置为 Pentium 双核 3.16GHz,内存 3.48GB,硬盘 400GB。实验程序在 Windows XP 系统 Visual C++环境下运行。

本文实验在默认情况下均采用 3 天为一个时间块,内存中最多可存储 10 个时间块的事件实例,在置换时每次调出 3 个时间块的事件实例;对于权重选取,$w1=0.6$,$w2=0.4$,即最近 R 个时间块和过去的时间块发生的事件在预测中所占的权重分别为 0.6、0.4;对于最近的时间块选取为 $R=3$,即用当前的 3 个时间块作为最近的时间块进行预测。

5.1 对象树对存储空间代价的影响

在本实验中,将分别用对象树(OTree)、SASE 系统的实例栈(SASE_Stack)及不采用任何压缩结构(Without compress_stru)的方式(即将每个事件实例的所有属性进行完全存储)对内存中的 10 个时间块的事件实例进行存储,比较用 3 种方法存储同样多的事件实例所用空间的大小。其实验结果如图 6 所示。

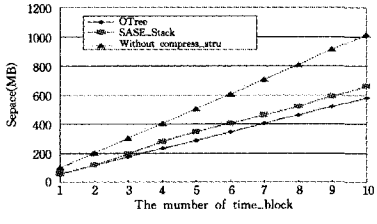

图 6 空间比较

从图 6 得知,本文提出的对象树对于存储相同时间块内的事件个数可以极大地节省内存空间。

5.2 PR 置换策略对复杂事件检测性能的评估

一般关于内外存的置换最朴素的方法是 FIFO 和 LIFO 置换算法。在本小节中,将对本文中提出的 PR 置换算法和 FIFO、LIFO 置换算法进行比较。

从图 7 中可以看出,PR 置换算法比较优越,检测相同天数的事件所用的总时间最少。之所以有这样的结果,是因为 PR 置换算法的思想正是根据事件流当前的发展规律来预测未来的发展趋势,这与事件流的发展趋势更加吻合。FIFO 置换算法和 LIFO 置换算法都没有考虑事件的发生规律。在这种情况下,很容易发生内存的抖动,即刚被调出到外存的对象事件,因为检测的需要又被调入内存,这样就会有大量的 I/O 开销,从而消耗大量的时间。

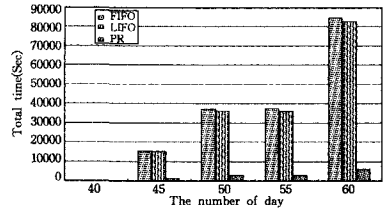

图 7 FIFO、LIFO 和 PR 置换算法对总时间的影响

5.3 权重 $w1$,$w2$ 的选择对检测的影响

$w1$、$w2$ 分别是过去和当前时间块的事件对未来事件发展预测的影响因子,值越大,说明预测过程中所占的作用也越大。本小节中,通过调整 $w1$、$w2$ 的值来评估它们对检测的影响。实验中选取 5 种不同的权重取值,对于相同天数的事件,检测其所用总时间。实验结果如图 8 所示。

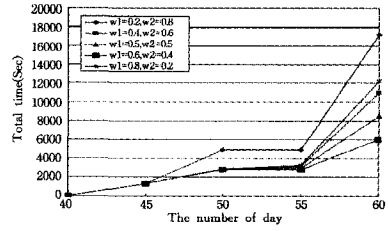

图 8 不同的权重的选择对总时间的影响

从图 8 中可以看到,检测处理相同天数的事件($w1=0.6$,$w2=0.4$)所花的总时间最少,效率最高。本节的实验证实,过去和当前时间块内发生的事件对未来事件的发展趋势都有影响,当前时间块的事件的发生更能预测未来事件的发展趋势,所以应赋予较大的权重。这并不是说事件流中过去的时间块内发生的事件不起作用,它的作用不可替代。$w1$、$w2$ 的合理选择对事件的检测起着重要的作用。

5.4 外存事件实例映射表对检测性能的影响

图 9 显示了随着所处理的事件天数的变化,不用实例映射(Without Instance_Map)和用实例映射(Instance_Map)的检测时间的变化。从图 9 可以看出,实例映射处理相同天数的事件,检测时间比不用实例映射的少得多。

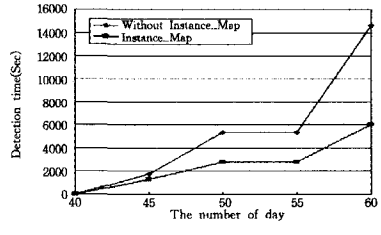

图 9 事件实例映射对检测时间的影响

这是因为进行检测时，在无实例映射的情况下，并不知道要查询的事件在外存中是否存在，这时需要打开可能存储此事件的外存文件，在文件中查找此事件，若查找不到，就浪费了大量查询时间。有实例映射表时，在实例映射表对应位就可以直接查到在外存文件中是否有此事件的存在，避免了无意义的查找。所以 Instance_Map 可以提高检测的效率，节省检测的时间。

结束语 面向长过程的复杂事件检测是目前研究比较少的一个领域，当前该领域的技术还处于空白，主要是由于长过程复杂事件的事件海量性。本文针对面向长过程的复杂事件检测，提出了一系列的存储和置换策略，内存对象树结构能够有效地对存储于内存中的事件进行压缩，使得有限的内存空间存储更多的事件实例。置换策略将检测发生不频繁的事件调出到外存，极大地减少了 I/O 的开销，提高了检测的效率。事件实例映射表便利了对存储于外存的事件的检测，大大提高了检测的效率。本文基于真实数据集验证了这一系列存储和置换策略能够使长过程复杂事件的检测具有完整性，并具有较高的效率。

参 考 文 献

[1] Vu C, Beyah R, Li Y S. Composite Event Detection in Wireless Sensor Networks[C]// Proc. of IPCCC. 2007: 264-271

[2] Al M, Simson G, Beth R. RFID: applications, security, and privacy [M]. Addison-Wesley, 2006: 33-47

[3] Angell I, Kietmann J. RFID and the end of cash [C]// Communications of the ACM(CACM). December 2006: 91-96

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

[5] Demers A, Gehrke J, Panda B, et al. Cayuga: a general purpose event monitoring system [C]// 3rd Biennial Conference on Innovative Data Systems Research(CIDR'07). 2007: 412-422

[6] Mei Y, Madden S. ZStream: a cost-based query processor for adaptively detecting composite events [C]// ACM SIGMOD Conference on Management of Data. 2009: 1-14

[7] Gruber R E, Krishnamurthy B B, Panagos E. The architecture of the READY event notification service [C]// Proc. of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop. 1999: 108-113

[8] Chakravarthy S, Mishra D. Snoop: an expressive event specification language for active databases [J]. Data Knowledge Engineering, 1994, 14(1): 1-26

[9] Geppert A, Tombros D. Event-based distributed workflow execution with EVE [R]. ifi-96. 05. University Zurich, Computer Science Department, 1996: 1-16

[10] Cooper O, Edakkunni A, Franklin M J, et al. HiFi: a unified architecture for high fan-in systems [C]// International Conference on Very Large Data Bases(VLDB'04). 2004: 1357-1360

[11] Gatziu S S, Dittrich K R. SAMOS: an active object-oriented database system [J]. IEEE Bulletin on Data Engineering, 1992, 15 (1-4): 23-26

[12] Intel Berkeley Research lab. Intel Lab DataSite [DB/OL]. http://db.csail.mit.edu/labdata/labdata.html, 2004-6-2/2011-3-1

(上接第 140 页)

证与分析过程中系统的形式化建模提供了新思路，并从安全质量方面改善了安全苛求软件的设计与开发，丰厚了基于模型的软件形式化开发方法。

参 考 文 献

[1] Neil S. Safety Critical Computer Systems[M]. Addison Wesley, 1996, 8

[2] CENELEC. EN 50128, Railway Applications: Communications, signalling and processing systems— Software for railway control and protection systems[S]. CENELEC. 2001

[3] CENELEC. EN 50126, Railway Applications: The specification and demonstration of Reliability, Availability, Maintainability and Safety(RAMS)[S]. CENELEC. 1999

[4] IEC. IEC61508, Functional Safety of electrical/electronic/programmable electronic safety-related systems-part3: software requirements [S]. IEC. 2000

[5] IEC. IEC61508, Functional Safety of electrical/electronic/programmable electronic safety-related systems-part7: Overview of techniques and measures[S]. IEC. 2006

[6] 吴芳美. 计算机联锁软件基于测试的安全性评价基准研究[J]. 铁道学报, 2005, 27(3): 97-101

[7] Blom S, Ioustinova N, Pol J, et al. Simulated Time for Testing Railway Interlockings with TTCN-3[C]// Proceedings of the 5th International Workshop on Formal Approaches to. Testing of Software. LNCS 3997, 2006: 1-15

[8] Hon Y M, Kollmann M. Simulation and Verification of UML-based Railway Interlocking Designs[C]// AvoCS. 2006: 168-172

[9] Nakarnatsu K, Kiuchi Y, Chen W Y, et al. Intelligent Railway Interlocking Safety on Annotated Logic Program and Verification Based its Simulator[C]// Proceedings of the 2004 IEEE International Conference on Networkinp, Sensing & Control. Taipei, Taiwan, 2004

[10] 王曦, 徐中伟, 梅萌. 基于模型检测的软件安全性验证方法[J]. 武汉大学学报, 2010, 56(2): 156-160

[11] Haxthausen A E, Peleska J. Formal Development and Verification of a Distributed Railway Control System[J]. IEEE Transactions on Software Engineering, 2000, 26(8): 1546-1563

[12] Garmhausen V H, Campos S, Cimatti A. Verification of a safety-critical railway interlocking system with real-time constraints [J]. Elsevier Science of Computer Programming, 2000, (36): 53-64

[13] Yang Shuang-hua, Yang Li-li. Automatic safety analysis of control systems[J]. Journal of Loss Prevention in the Process Industries, 2005, 18(3): 178-185

[14] Bozzano M, Villa_orita A. The FSAP/NuSMV-SA Safety Analysis Platform [J]. Software Tools for Technology Transfer, 2007, 9(1): 5-24

[15] Koh K Y, Seong P H. SMV model-based safety analysis of software requirements[J]. Reliability Engineering & System Safety, 2009, 94(2): 320-331

[16] 郭宁. UML 及建模[M]. 北京: 清华大学出版社, 2007

[17] Magee J, Kramer J. Associated Concurrency: State Models and Java Programs[M]. Wiley, 1999

[18] 赵志熙. 计算机联锁系统技术[M]. 北京: 中国铁道出版社, 1999