网络化软件交互行为动态建模

彭 成[1],杨路明[1],满君丰[2]
(1.中南大学信息科学与工程学院,湖南长沙 410083;2.湖南工业大学计算机与通信学院,湖南株洲 412007)

摘 要: 目前的软件行为建模方法有其局限性,而网络化软件交互行为比传统软件更为复杂难控,对模型的定义和优化提出了更高的要求.本文提出一种基于不变量约束规则的挖掘方法,从监控收集的软件交互行为日志中挖掘出六类不变模式,简化了模型空间;并用事件描述状态,提高了数据集的表达能力;构建的动态模型考虑了参数传递和组件之间的交互,将数值关联关系映射到控制流中,从而更真实地刻画了软件交互行为;模型中的事件满足不变量约束规则,为软件行为分析提供了依据;同时,本文提出合并划分子图间的等价状态方法,对模型进行精化和抽象,确保了模型的确定性和完备性.仿真实验和实例分析证实了该方法的正确性和有效性.

关键词: 网络化软件; 交互行为; 不变量约束; 动态模型
中图分类号: TP302.7 **文献标识码:** A **文章编号:** 0372-2112 (2013) 02-0314-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.02.017

Dynamic Modeling of Networked Software Interactive Behavior

PENG Cheng[1], YANG Lu-ming[1], MAN Jun-feng[2]
(1. *School of Information Science and Engineering, Central South University, Changsha, Hunan* 410083, *China*;
2. *College of Computer and Communication, Hunan University of Technology, Zhuzhou, Hunan* 412007, *China*)

Abstract: Current behavior modeling methods have its limitation, to deal with more complicated and uncontrollable networked software behavior, model definition and optimization need to be set up. The invariant constraints mining algorithm was investigated, six types of invariants are mined from the software interactive behavior log, which simplifies the model space. To improve the data set expression capacity, events was adopted to describe the states, and data value relationship was mapped into control flow, which more realistically describes the interactive behavior, meanwhile, events in the model satisfied with invariant constraint rules, which provide the basis for software behavior analysis. To ensure the certainty and compleueness of the model, the method of merging equivalent states in the divided sub-diagram was proposed. The experimental and analysis results show that the effectiveness and feasibility of the methods are validated.

Key words: networked software; interactive behavior; invariant constraint; dynamic model

1 引言

随着计算机网络技术和软件技术的发展和进步,计算模式不断更新,先后出现了分布式计算,网格计算,云计算和透明计算技术[1],部署在其上的软件系统也发生了相应的变化,从单机版软件到网构软件再到网络化软件[2],软件系统的运行环境也从封闭,可控的环境转变到开放,跨平台,动态,资源共享,多变,协同,难控的环境.在这种多重复杂的情况下,如何实现用户任务的实时有效处理,提供稳定持续的在线服务,以及如何保证数据的安全性可靠性等是亟待解决的问题.解决问题的关键不仅在于各种网络协议和计算及存储机制的完善和创新,更在于对运行在各种架构下的软件系统本身的构造性和演化性[3]的分析和研究,不仅在软件的开发阶段,更在软件的运行阶段,对软件这个本体元素的生命体征和交互行为特性的把握,将有助于我们从另一个视角提高网络化软件的可信性[4],进而提高网络环境下用户获取的服务质量.

目前,软件行为建模方法主要分为对程序源码的静态分析和对运行时监测收集的日志文档进行动态分析两种,前者称为静态建模,后者称为动态建模[5~7].静态建模方法试图分析源码中程序调用关系形成的所有可能路径,实际上受限于版权、反编译软件的增多,由此产生的行为模型很难具备完整性和适用性.动态建模方法

收稿日期:2012-07-20;修回日期:2012-10-12
基金项目:国家技术创新基金(No.11C26214302856);湖南省自然科学基金(No.12JJ2036,No.11JJ4050);湖南省教育厅优秀青年项目(No.11B039)

通过监测收集软件交互产生的日志文档,从中提取程序调用关系,并生成行为模型.动态建模虽然简单,实用,被广泛采用,但依然存在着一些限制:(1)监测对象仅限于单机系统软件或应用软件[8~10],针对分布式环境下,网络化的大规模软件系统的运行时交互行为监测较少;(2)监测收集的日志文档[11]格式无标准,内容混乱,多冗余,给分析带来困难;(3)事件描述过于简单,缺乏考虑软件执行时参数的传递,导致在模型中控制流与数据流不能兼顾,模型对于数据语义的攻击检测将失效;(4)模型状态的设置欠合理,程序指针(PC)作为状态[12],虽然能刻画程序运行时调用关系,但是它仅包含地址信息,不能判断模型中的相似结构,对行为踪迹中状态的合并无法处理.从系统调用资源解析出对象(Object)作为状态[13],仅适用于操作系统的日志文档分析;(5)模型接受的事件序列必须是按时间顺序排列的有序序列[14,15],对于分布式环境下的并行[16~18]执行产生的无序或部分有序的事件序列失去接受能力.这个问题在前期工作中[19]已有探讨;(6)模型对于日志文档的完整性依赖很强,日志文档的不完整将导致从中挖掘的不变量(Invariant)[20~22]失去准确性,而不变量对于模型的精化和抽象起关键作用.

鉴于此,本文提出一种基于网络化软件交互行为的动态建模方法 IBDM(Interactive Behavioral Dynamic Modeling),拟解决上述问题.IBDM 通过预先设置所监测的信息格式,对软件内部的动态交互行为进行长期在线监测,并存为日志文档.首先,使用自动生成的正则表达式,将日志文档分割为若干踪迹图,然后,从踪迹图中挖掘 6 种不变量,根据踪迹子图是否满足不变量约束进行划分,最后,IBDM 合并满足不变量约束的踪迹子图间的等价结构,对初始模型进行精化和抽象,形成最终模型.理论分析和实验证明 IBDM 建模方法具有准确性和有效性,为后续行为分析奠定了基础.

2　相关工作

与日志文档挖掘相关的工作主要集中在检测日志文档中的相关性、异常以及性能调试等方面.这些工作的目的并不是从任意系统产生的日志文档中找到一个精确的模型.比如,工具 SALSA[23]和 Mochi[24]提取并可视化 Hadoop 日志中节点的行为来进行性能调试,它的主要任务是做 MapReduce 描述.J Yang[25]挖掘并可视化事件踪迹的临时属性,用它们研究程序的进化,也没用使用这些临时属性推断系统模型.模型推断方面,K-tail 算法[26]被广泛使用,它通过迭代地合并具有相同最大深度为 K 的状态子图生成一个精简的有限状态机模型.一般地,利用 K-tail 算法可以在无开发人员监督的情况下为小型或简易系统推断其模型,一旦系统复杂度增加,模型的准确度就会大幅下降,不过它的合并机制可以进一步地改进和完善,为我们提供很好的借鉴作用.另外,还有许多利用开发人员编写规约条件来推断系统模型的方法.Whittle 和 Schumann[27]从系统运行的场景和属性中推断出组件状态图,Damas[28]等利用开发人员提供的交互式场景归纳出标记变迁系统(Labeled Transition System,LTS).后来,又通过减少开发人员的参与对该方法进行了扩展,然而,这些方法可能产生过度冗余的模型,并且需要大量的人工输入.

3　IBDM 模型构建

3.1　模型定义

在定义 IBDM 模型之前,首先给出相关的基本定义.

定义 1　(事件)表示函数序列集合.对在线电子购物系统进行监控时,预设监控事件格式为 $e=(m,p,v,t,c)$,其中,$m\in M$,$p\in D_R$,$v\in D_V$,$t\in T$,$c\in C$,给定 M 是函数的有限集,R 是参数的有限集,V 是变量的有限集,T 是时间戳的集合,C 是系统所有类的集合,D_R 为参数的值域,D_V 为变量的值域.

定义 2　(事务)事务 $t_i=(e_{i1},e_{i2},\cdots,e_{in})$,是指完成某类操作所包含的事件实例序列,它是系统踪迹的子集.

定义 3　(踪迹)表示按时间戳排列的事件序列有限集合.$bt=(m_1,p_1,v_1)\cdots(m_n,p_n,v_n)$,其中 n 表示事件实例的个数,m 为属于不同事务的事件实例.

定义 4　(不变量)日志中记录并行执行的多个事务产生的事件实例时间戳上存在重叠,导致踪迹内部不完全有序,即事件之间部分有序(partly order),用符号 $<$ 表示.若 a_i,b_j 为事件类型,$\hat{a}_i,\hat{b}_j$ 是与之相应的事件实例,那么可以从踪迹中挖掘 6 种与事件类型 a_i 和 b_j 相关联的不变量.分别是:

$a_i\rightarrow b_j$:事务 i 产生的类型为 a 的事件实例总是被事务 j 产生的类型为 b 的事件实例跟随.可表示为:$\forall\hat{a}_i,\exists\hat{b}_j,\hat{a}_i<\hat{b}_j$.

$a_i\not\rightarrow b_j$:事务 i 处产生的类型为 a 的事件实例总是从来不被事务 j 产生的类型为 b 的事件实例跟随.可表示为:$\forall\hat{a}_i,\nexists\hat{b}_j,\hat{a}_i<\hat{b}_j$.

$a_i\leftarrow b_j$:事务 i 产生的类型为 a 的事件实例总是先于事务 j 产生的类型为 b 的事件实例.可表示为:$\forall\hat{b}_j,\exists\hat{a}_i,\hat{a}_i<\hat{b}_j$.

$a_i\parallel b_j$:事务 i 产生的类型为 a 的事件实例与事务 j 产生的类型为 b 的事件实例总是同时发生.可表示为:

$\forall a_i, b_j, (\hat{a}_i \prec \hat{b}_j \wedge \hat{b}_j \prec \hat{a}_i)$.

$a_i \nparallel b_j$:事务 i 产生的类型为 a 的事件实例与事务 j 产生的类型为 b 的事件实例从不同时发生.可表示为:$\forall a_i, b_j, (\hat{a}_i \not\prec \hat{b}_j \wedge \hat{b}_j \not\prec \hat{a}_i)$.

$a_i \leftrightarrow b_j$:事务 i 产生的类型为 a 的事件实例与事务 j 产生的类型为 b 的事件实例循环发生.可表示为:$\forall a_i, b_j, (\hat{a}_i \prec \hat{b}_j \vee \hat{b}_j \prec \hat{a}_i)$.

定义 5 (状态)表示程序执行过程中调用的函数序列,即事件集合,$S \supseteq \{START, END\}$.

定义 6 (等价状态)给定两个交互踪迹,$bt_1 = (m_1, p_{m_1}, v_1)\cdots(m_x, p_{m_x}, v_x)$, $bt_2 = (f_1, p_{f_1}, w_1)\cdots(f_t, p_{f_t}, w_t)$, $bt_1 \Leftrightarrow bt_2$,当且仅当 $x = t$,且 $\forall i = 1, \cdots, x$, $m_i = f_i$, $p_{m_i} = p_{f_i}$, $v_i = w_i$,那么踪迹中事件对应的状态也是等价的.

定义 7 (转移)转移 $t = (s, m, P, s')$,其中,$s, s' \in S$,分别表示转移中的源和目的状态,$m \in M$ 为转移中调用的函数,P 为转移判定.

定义 8 (判定)表示转移是否成立.$P: D_R \times D_V \rightarrow \{\text{True}, \text{False}\}$,它说明的函数是否接受输入的参数和变量的值.例如,函数 login 和判定条件 $P = \text{length}(\text{user}) > 0 \wedge \text{length}(\text{pwd}) > 0$ 联合表示状态转移时函数 login 的参数是否满足判定条件,从而决定是否产生转移.

定义 9 (路径)给定长度为 n 的踪迹 $bt = (\text{START}, e_1, \ldots, e_n, \text{END})$ $n \in \mathbf{N}$,要使模型接受踪迹 bt,成立的充分必要条件是 $\exists \prod = (\text{START}, m_1, P_1, s_1)(s_1, m_2, P_2, s_2)\cdots(s_{n-1}, m_n, P_n, \text{END})$ 对于 $\forall i = 1, \cdots, n$, $P_i(p_{m_i}, v_i) = \text{true}$,那么,$\prod$ 是模型 M 中的完全路径,一条完全路径表示模型接受的踪迹序列,是我们进行后续行为分析(异常行为诊断等)的依据.

定义 10 (IBDM 模型)为四元组 (S, T_P, s_c, s_z),其中,

(1) $S = \{m \mid m \in M, m$ 为软件执行时调用的函数$\}$,描述了软件执行时状态集合;

(2) $T_P: D_M \xrightarrow{P} S$, $D_M \subseteq M \times R \times V$,表示条件判断转移;

(3) $s_c = \{\text{START}\}$,为初始状态集合;

(4) $s_z \supseteq \{\text{END}\}$,为终止状态集合.

3.2 构建算法

算法 1 IBDM 构建算法

①输入日志文档 L,正则表达式 RegExps;

②从日志中提取踪迹图 traceGraph;

③根据定义 4,从 traceGraph 中挖掘不变量;

④划分 traceGraph 得到子图集合 $\{G\}$,设置初始化计算划分步数 Step 为需要进行划分次数,设置划分步数初值 $n = 0$;

⑤若划分的子图不满足不变量,即将这个划分子图再分割成两个部分:π_1, π_2, π_1 包含满足不变量的事件实例,π_2 不含满足不变量的事件实例;

⑥$n = n + 1$;

⑦若 $n = \text{Step}$,或者所有子图中的事件实例都满足不变量时结束.否则,重复循环;

⑧根据定义 6 合并子图间的等价状态,生成与事件关联的判定 P,形成精化和抽象的最终模型.

算法 2 挖掘不变量

①初始化事件类型为 a_i, b_j,初始循环次数为 N,设置初值 $i = 0$;

②根据定义 4,计算不变量:

若跟随在事件实例 $\hat{a}_i$ 之后出现的事件实例 $\hat{b}_j$ 的计数值 Follow$[a_i][b_j]$ 与事件实例 $\hat{a}_i$ 出现次数的计数值 Occ$[a_i]$ 相等,那么,生成不变量 $a_i \rightarrow b_j$;

若跟随在事件实例 $\hat{a}_i$ 之后出现的事件实例 $\hat{b}_j$ 的计数值 Follow$[a_i][b_j] = 0$,那么,生成不变量 $a_i \not\rightarrow b_j$;

若先于事件实例 $\hat{b}_j$ 出现的事件实例 $\hat{a}_i$ 的计数值 Prec$[a_i][b_j]$ 与事件实例 $\hat{b}_j$ 的出现次数计数值 Occ$[b_j]$ 相等,那么,生成不变量 $a_i \leftarrow b_j$;

若先于事件实例 $\hat{b}_j$ 出现的事件实例 $\hat{a}_i$ 的计数值 Prec$[a_i][b_j]$ 与跟随在事件实例 $\hat{a}_i$ 之后出现的事件实例 $\hat{b}_j$ 的计数值 Follow$[a_i][b_j]$ 相等,那么,生成不变量 $a_i \leftrightarrow b_j$;

若事件实例 $\hat{a}_i$ 与 $\hat{b}_j$ 在一个踪迹中的共现次数 CoOcc$[a_i][b_j] = 0$,且 Follow$[a_i][b_j] = 0$, Prec$[a_i][b_j] = 0$,那么,生成不变量 $a_i \nparallel b_j$;

若相继出现的事件实例对 $< \hat{a}_i, \hat{b}_j >$,它们的计数值 Followpair$[a_i][b_j]$ 与 Precpair$[a_i][b_j]$ 之和,与事件实例 $\hat{a}_i$ 与 $\hat{b}_j$ 在踪迹中的共现次数累加和 $\sum$CoOcc$[a_i][b_j]$ 相等,那么生成不变量 $a_i \parallel b_j$.

③$i = i + 1$;

④若 $i = N$,计算完毕,否则重复循环.

⑤输出不变量.

4 实例及分析

4.1 状态的设置

根据信息论的熵定义:$H(X) = -\sum p(x)\log p(x)$,它可以定量地描述事件(函数调用)、程序指针(系统调用)和系统对象作为状态所表征的信息质量.熵值越小,数据集表达的内容越规则,从这些数据集中建立的模型越准确.根据实验数据,计算出这三种数据源的熵值如图 1 所示.从图 1 可以看出事件数据集的熵值明显

小于程序指针和系统对象数据集的熵值,用事件来描述状态是合理的.

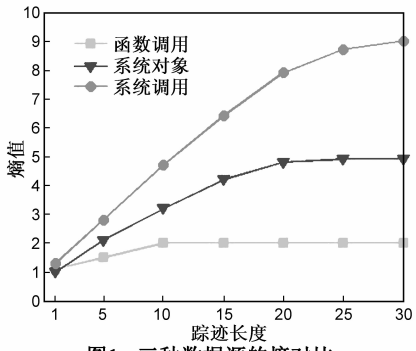

图1　三种数据源的熵对比

4.2　踪迹子图

电子购物系统运行时产生的交互日志被预先设定了监控格式,收集的事件按时间戳部分有序排列.针对这种特殊格式的日志文档,我们使用正则表达式对其进行解析,将相同的事件类型归为一类,图中每一个顶点代表一个事件实例,边表示事件实例发生先后关系,边上的属性表示传递的参数,这样数据流与控制流结合起来,我们把这种图的集合称为踪迹图.本文的日志文档,使用正则表达式:(? < timestamp >). +″(? < TYPE > . +)″,它将日志文档分割为三个踪迹,每个踪迹对应一个交互事务.如图 2 所示.其中,与事务 1 对应的踪迹为: < 0, login > , < 1, search-goods > , < 2, order-items >,< 3, valid-order >,< 4, get--card >,< 5, check-out > ,其中的整数是从日志行中导出的按序排列的时间戳.

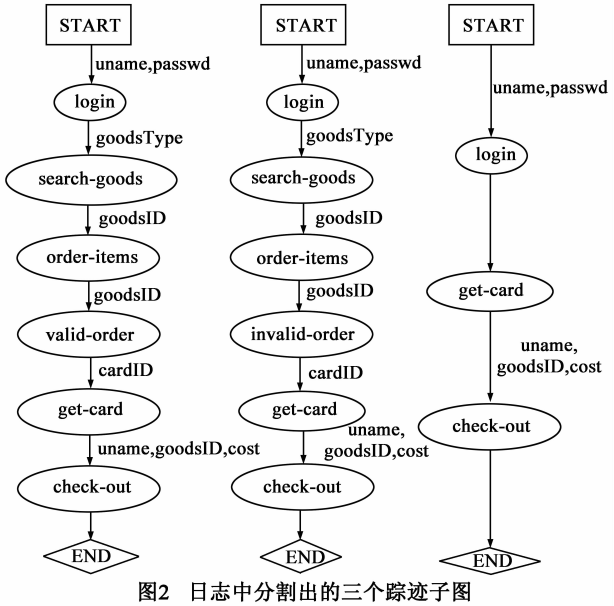

图2　日志中分割出的三个踪迹子图

4.3　不变量

实验从含 280 条记录的日志中挖掘出 6 种不变量,表 1 给出了挖掘的部分不变量.不难发现,在我们实际的购物操作中,不允许系统出现如“invalid-order→get-card”这种反例不变量,即在缺货的情况下,仍然对该货物进行订购并支付货款,在模型中体现为反例路径的存在.从我们的实验对象—电子购物系统中捕获了这种反例不变量,它与正常情形不相符,但实实在在发生在交易过程中,根据这些不变量构建模型,那么,遍历这个模型并判断是否存在反例路径,是检测系统漏洞的主要依据.

表 1　六种不变量部分结果

$a \rightarrow b$	$a \not\rightarrow b$	$a \leftarrow b$
get-card→check-out	check-out ↛get-card	login←check-out
get-card→login	check-out ↛valid-order	login←get-card
invalid-order→get-card	get-card ↛valid-order	login←invalid-order
valid-order→get-card	get-card ↛order-items	login←valid-order
$a \leftrightarrow b$	$a \parallel b$	$a \not\parallel b$
search-goods↔order-items login↔check-out search-goods↔invalid-order valid-order↔get-card	search-goods ‖ search-goods	get-card ∦ get-card

实验还发现,针对不同大小的日志文档,6 种不变量的数量相对稳定,表 1 中列出的 6 种不变量所占比例依次为,31.4% ,34.3% ,11.4% ,14.3% ,5.7% ,2.9% .如图 3 所示,不同大小的日志中挖掘的 6 种不变量的统计结果,6 种不同颜色表示了 6 类不变量.所占比例较大的不变量反映了系统中发生频率较高的事件实例之间关联关系,所占比例较小的不变量反映的是系统中不常发生或系统事件实例之间易隐含漏洞的关联关系,如不变量“get-card ∦ get-card”,表明对信用卡的操作不能同时进行,存在读写冲突.对不变量特别是小比例的不变量的关注和分析有助于我们更深入地理解系统行为,发现系统中存在的漏洞,提前预防系统错误行为的产生.

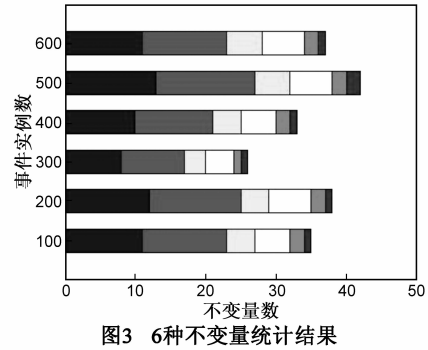

图3　6种不变量统计结果

4.4　精化和抽象

抽象和精化是 IBDM 建模的双重操作.从初始模型开始,首先执行模型精化,它是一个迭代过程.IBDM 精化每一个踪迹子图直到模型满足挖掘的所有不变量.

接着,通过抽象化合并由于不完整分割导致不必精化的子图,并保证不违反在精化过程中对挖掘的不变量的可满足性.当不能再进行抽象化,输出模型.

只要模型不满足不变量的约束,IBDM 继续执行分割.它使用基于 FSM 的模型检验器来验证模型是否满足不变量约束.IBDM 并行追踪模型中输入踪迹的反例来识别划分.在踪迹中,仅需要表示反例路径的前缀部分,IBDM 找出最长的这种前缀,模型中对这种前缀的最后一次划分就是精化的候选划分.在模型中允许反例路径的存在.通过搜索最终模型中的反例路径,可以检测出系统漏洞,在上文的不变量部分已经作了相应说明.

精化可能产生多个结果,当这种情况发生时,模型中包含了可以合并的划分,并且不违反可满足的不变量.IBDM 采用 K-tail 等价类方法进行抽象,它从最细粒度的模型开始,合并每一个 K-tail 等价类划分,直到不存在任意一对 K-tail 等价类划分.在 Linux 操作系统 ubuntu 环境下,运行 IBDM 算法,输出的最终模型如图 4 所示.

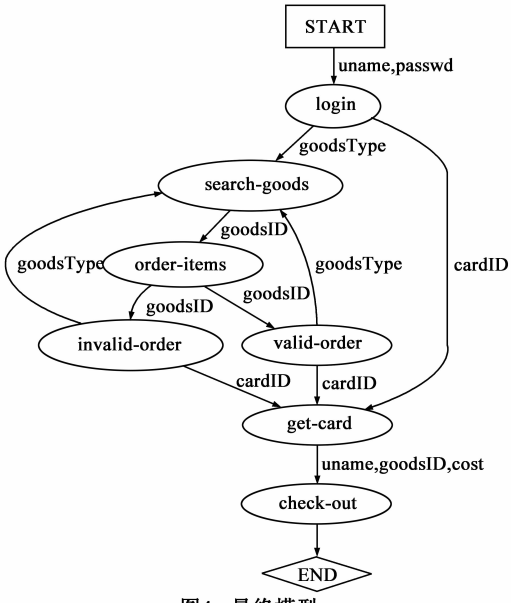

图4 最终模型

4.5 准确性和效率评估

定义 11 (模型准确率)是对模型接受能力的评估,描述了被接受的事件序列占日志中踪迹总数的比例 f.

通过假设检验和分布拟合,确定日志中事件的出现服从泊松分布 $P(\lambda)$,λ 为单位时间事件发生的平均次数.遍历日志中存在且同时出现在模型中的事件,将它们按时间顺序分割成若干操作序列,与模型中存在的完全路径 Π 条数之间的比率就是我们要求的准确率 f,将其表示为:

$$f=\frac{\varphi\left(\sum_{m\in M}P(\lambda)\times|L|-I_n\right)}{\psi(N)} \tag{1}$$

其中:$|L|$为某时刻日志文档长度,I_n 为某时刻未在模型中出现的事件个数,$m\in M$ 为日志中的事件,$\psi(N)$为模型中存在的完全路径条数,N 为模型的规模.

我们在图 5 中给出了针对在线电子购物系统的日志,采用目前比较流行的建模算法 K-tail 和本文提出的 IBDM 建模算法准确率的比较曲线.所有的数据都是 100 次运行的平均结果,取 $\lambda=4$.实验发现,由于 IBDM 建模算法采用不变量对模型进行约束,随着日志文档长度的增加,模型中完全路径与事件序列之间匹配成功概率也在增大,说明模型的接受能力也在增强.相反,随着模型规模的扩大,由于没有不变量的约束,K-tail 算法生成的模型中包含越来越多的冗余状态,对应为路径的重复,模型的接受能力受到重复匹配的干扰,准确率也受到影响.IBDM 建模算法准确率在 70% 左右,这与日志文档的完整性有关,算法对日志文档的依赖依然比较强,在相同的情况下,一个包含了所有未知事件的完整的日志文档将达到比较好的准确率.

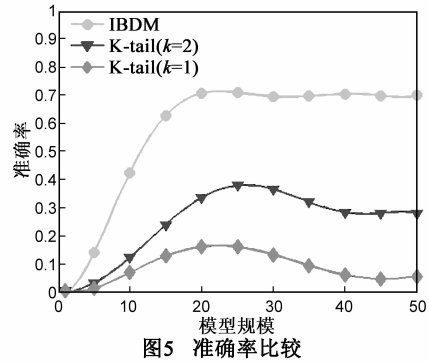

图5 准确率比较

图 6 针对在线电子购物系统运行一周内获取的日志记录进行分析,给出了 IBDM 算法包含的子过程的时间复杂度曲线.IBDM 可以分为不变量挖掘、子图提取、精化和抽象四个子过程.其中不变量挖掘所花费的时间明显比其他过程所花时间长,因此设计良好的挖掘算法将直接影响整个建模的效率.其次,模型建立时,

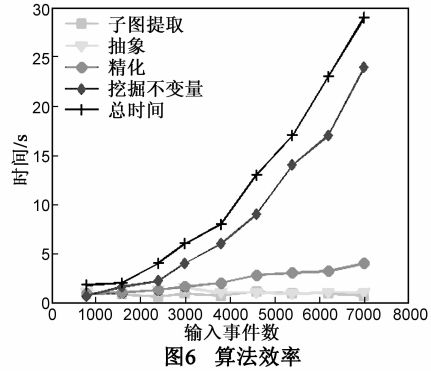

图6 算法效率

没有任何其他假设条件,该方法可以适应于任意网络化软件的行为建模.

5 总结与下一步工作

基于不变量约束的软件交互行为动态建模方法为软件行为分析提供了新的思路和方法.本文针对在线电子购物系统产生的交互日志文档,提出将这种规模庞大、内容纷繁复杂的记录文本抽象为直观、精简的模型,它不仅满足从日志中提取的不变量约束关系,而且形象地描述了程序行为规律.通过它可以加深我们对系统的理解和认识.分析模型的反例路径,可以揭示系统中存在的潜在威胁,验证已知漏洞,该模型还可以用于恶意软件的传播动力学分析,可以利用该模型提取恶意软件运行轨迹,对其进行控制和预防,从而提高系统的安全性和可靠性.算法仿真平台的搭建为我们的后续研究提供了基础.

通过该模型进行程序行为异常检测和发现漏洞时如何进行牵制控制使得整个系统行为维持正常稳定,以及对程序中的某些特殊行为的同步分析将是我们下一步的工作.

参考文献

[1] 张尧学.透明计算:概念,结构和示例[J].电子学报,2004,32(12A):169 – 173.
Zhang Yao-xue. Transparence computing: Concept, architecture and example[J]. Acta Electronica Sinica, 2004, 32(12A): 169 – 173. (in Chinese)

[2] 马于涛,何克清,李兵,刘婧.网络化软件的复杂网络特性实证[J].软件学报,2011,22(3):381 – 407.
Ma Yu-Tao, He Ke-Qing, Li Bing, Liu Jing. Empirical study on the characteristics of complex networks in networked software [J]. Journal of Software, 2011, 22(3): 381 – 407. (in Chinese)

[3] 杨芙清,梅宏,吕建,金芝.浅论软件技术发展[J].电子学报,2002,30(12A):1901 – 1906.
Yang Fu-qing, Mei Hong, Lu Jian, Jin Zhi. Some discussion on the development of software technology[J]. Acta Electronica Sinca, 2002, 30(12A): 1901 – 1906. (in Chinese)

[4] 陈火旺,王戟,董威.高可信软件工程技术[J].电子学报,2003,31(12A):1933 – 1938.
Chen Huo-wang, Wang Ji, Dong Wei. High confidence software engineering technologies [J]. Acta Electronica Sinica 2003, 31 (12A): 1933 – 1938. (in Chinese)

[5] Leonardo Mariani, Mauro Pezzè, Oliviero Riganelli, Mauro Santoro. SEIM: static extraction of interaction models[A]. International Workshop on Software Engineering [C]. Cape Town, South Africa: IEEE Computer Society, 2010. 22 – 28.

[6] Christopher Ackermann, Mikael Lindvall, Rance Cleaveland. Towards behavioral reflexion models [A]. Reliability Society [C]. Mysuru, India: IEEE Computer Society, 2009. 175 – 184.

[7] Jonathan E. Cook, Alexander L. Wolf. Discovering models of software processes from event-based data [J]. ACM Transactions on Software Engineering and Methodology, 1998, 7(3): 215 – 249.

[8] Kai-Yuan Cai, Bei-Bei Yin. Software execution processes as an evolving complex network[J]. Information Sciences, 2009, 179 (12): 1903 – 1928.

[9] Tao Li, Wei Peng, Charles Perng, Sheng Ma, Haixun Wang. An integrated data-driven framework for computing system management[J]. IEEE Transactions on Systems, 2010, 40(1): 90 – 99.

[10] Chun ying Zhao, Jun Kong, Kang Zhang. Program behavior discovery and verification: A graph grammar approach[J]. IEEE Transactions on Software Engineering, 2010, 36(3): 431 – 447.

[11] Anton ChuvAkin, GunnAr Peterson. How to doapplication logging right[J]. IEEE Computer and Reliability Societies, 2010, 8(4): 82 – 85.

[12] Zhen Li, Jun Feng Tian, Liu Yang. An improved software behavior model in system call level and trustworthiness evaluation[J]. Information Technology Journal, 2011, 10(11): 2208 – 2213.

[13] 傅建明,陶芬,王丹,张焕国.基于对象的软件行为模型[J].软件学报,2011,22(11):2716 – 2728.
Fu Jian-Ming, Tao Fen, Wang Dan, Zhang Huan-Guo. Software behavior model based on system objects[J]. Journal of Software, 2011, 22(11): 2716 – 2728. (in Chinese)

[14] Curtis E. Hrischuk, Murray Woodside. Logical clock requirements for reverse engineering scenarios from a distributed system[J]. IEEE Transaction on Software Engineering, 2002, 28 (4): 321 – 338.

[15] Selvaraj Srinivasan, R Rajaram. A decentralized deadlock detection and resolution algorithm for generalized model in distributed systems [J]. Distributed and Parallel Databases, 2011, 29(4): 261 – 276.

[16] Jonathan E. Cook, Zhidian Du, Chongbing Liu, Alexander L. Wolf. Discovering models of behavior for concurrent workflows[J]. Computers in Industry, 2004, 53(3): 97 – 319.

[17] Jonathan E Cook, Zhidian Du. Discovering thread interactions in a concurrent system[J]. Journal of Systems and Software, 2005, 77(3): 285 – 297.

[18] Jonathan E Cook, Cha He, Changjun Ma. Measuring behavioral correspondence to a timed concurrent model[A]. IEEE Computer Society's Technical Council on Software Engineering[C]. Florence, Italy: IEEE Computer Society, 2001. 332 – 341.

[19] 彭成,杨路明,满君丰.不完全事务行为踪迹标记研究

[J]. 小型微型计算机系统,2011,32(8):1593 – 1598.
Peng Cheng, Yang Lu-ming, Man Jun-feng. Research on tokenizing behavior footprints of incomplete transaction[J]. Journal of Chinese Computer Systems, 2011, 32(8): 1593 – 1598. (in Chinese)

[20] Jian-guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, Jiang Li. Mining invariants from console logs for system problem detection[A]. USENIX Workshop[C]. Boston, US: USENIX, 2010. 231 – 244.

[21] Andrzej Wasylkowski , Andreas Zeller. Mining temporal specifications from object usage[A]. ACM SIGART/SIGSOFT-Workshop[C]. Washington, DC: IEEE Computer Society, 2009. 295 – 306.

[22] Florian Skopik, Daniel Schall, Schahram Dustdar. Modeling and mining of dynamic trust in complex service-oriented systems[J]. Information Systems, 2010, 35(7): 735 – 757.

[23] J Tan, X Pan, S Kavulya, R G, P Narasimhan. SALSA: analyzing logs as state machines[A]. USENIX Workshop[C]. San Diego, CA: USENIX, 2008. 6 – 6.

[24] J Tan, X Pan, S Kavulya, R G, P Narasimhan. Mochi: visual log-analysis based tools for debugging hadoop[A]. IEEE International Conference on Distributed Computing Systems Workshops[C]. Geneva, Italy: IEEE Computer Society, 2010. 795 – 806.

[25] J Yang, D Evans. Dynamically inferring temporal properties [A]. Acm Sigartsigsoft Workshop[C]. Washington, DC: Association for Computing Machinary, 2004. 23 – 28.

[26] A W Biermann, J A Feldman. On the synthesis of finite-state machines from samples of their behavior[J]. IEEE Trans. Comput, 1972, 21(6): 592 – 597.

[27] J Whittle, J Schumann. Generating state chart designs from scenarios[A]. ACM Limerick[C]. New York, Ireland: IEEE-CS Computer Society, 2000. 314 – 323.

[28] C Damas, et al. Generating annotated behavior models from end-user scenarios[J]. IEEE Transaction on Software Engineering, 2005, 31(12): 1056 – 1073.

作者简介

彭 成(通讯作者) 男,1982 年生于湖南长沙,汉族,中南大学信息科学与工程学院计算机系博士研究生,主要研究方向为软件工程,可信软件,软件形式化方法.
E-mail: doc_pen@126.com; mjfok@qq.com

杨路明 男,1947 年生于江西,汉族,中南大学信息科学与工程学院计算机系教授,博士生导师,主要研究方向为软件工程,计算机网络.
E-mail: yang.601@163.com