网络化软件异常行为传播研究

彭 成[1,2],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: 软件中的漏洞不可避免,研究由漏洞引发的网络化软件的异常行为传播机制,为人们把握异常行为传播规律并采取相应的牵制措施提供了依据.根据异常行为在不同粒度的软件实体中传播的情形,提出了影响异常行为传播的三类因素:传播概率、交互频率和连接率,并给出了相关定义及其计算方法;结合仓室模型和个体模型及上述三因素,构建了描述软件异常行为传播过程的模型,提高了模型的表达能力,增强了模型的完备性和准确率.将异常行为传播分析方法应用于典型的网络化软件系统,通过实验计算出各参数及其变化规律,验证了该传播机制的正确性和可行性.

关键词: 网络化软件; 异常行为; 传播模型; 系统漏洞
中图分类号: TP302.7 **文献标识码:** A **文章编号:** 0372-2112 (2013) 10-2074-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.10.031

Research on Networked Software Abnormal Behavior Propagation

PENG Cheng[1,2], 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: Bugs in software are inevitable. The study on the networked software abnormal behavior propagation mechanism trigged by bugs provided the way for people to grasp the execution rule and to adopt corresponding pinning measurements. Based on the situation of abnormal behavior propagation at different granularity software entities, three factors----propagation probability, interactive frequency, and connection rate----affecting the abnormal behavior propagation were proposed, and the corresponding definition and calculation method were also investigated. The software abnormal behavior propagation process model was constructed in reference to the compartment model and individual models and three factors mentioned above, which improved model expression ability and enhanced the model competence and accuracy. Then, the abnormal behavior propagation analytical method was applied to the typical networked software system, and the results verified the correctness and feasibility of the propagation mechanism.

Key words: networked software; abnormal behavior; propagation model; system bug

1 引言

异常行为是指软件系统自身存在缺陷或漏洞,而使系统行为与正常行为之间存在偏差,使系统处于某种错误的运行状态,最终将导致系统失效.漏洞不可避免,检测和修复的代价很大,而且一旦被利用,将产生不可估量的损失和破坏.以互联网为基础,整合了各种异构资源、服务等的网络化软件[1],其行为复杂难控[2]且持续演化[3],元素内部的小错误可能引发整个系统的崩溃,对用户数据安全和服务质量带来极大挑战.据此,研究网络化软件异常行为传播机制刻不容缓,以期在灾难爆发之前对其进行控制并将其消除,维持系统的正常稳定运行,为用户提供持续可靠服务.

目前,研究软件漏洞的分类[4]、分布情况[5],漏洞的检测[6]及其建模、修复机制[7]取得了很多有益成果,但还存在一些限制:(1)分析对象主要针对单机系统软件,对网络环境下的分布式大型软件系统的分析不足;(2)采用错误注入[8]的方式仅能检测验证已知错误,对潜在的未知错误及其传播规律缺乏表达能力;(3)参考计算机病毒传播模型和生物病毒传播模型建立的分析模型,并不能完全适应于网络化软件,不能真实刻画和揭示其异常行为传播特征;(4)模型粒度较大,建模主要基于系统级(System-Level)及组件级(Component-Level),未结合程序级(Program-Level)考虑其内部的错误传播过程;(5)

收稿日期:2013-02-22;修回日期:2013-07-18;责任编辑:蓝红杰
基金项目:国家自然科学基金(No.60773110);国家技术创新基金(No.11C2614302856);湖南省自然科学基金(No.12JJ2036,No.11JJA4050)湖南省教育厅优秀青年项目(No.11B039)

错误行为与系统执行环境脱离,将错误行为视为一定范围的固定模式,模型的可扩展和普适性较弱;(6)分析模型主要针对低级语言系统,需有硬件支持,代价较大.

鉴于此,本文以前期工作[9]为依托,将仓室模型[10]和基于个体的模型[11]结合起来,建立网络化软件的异常行为传播模型(ABPM)(Abnormal Behavior Propagation Model),从粗粒度的组件、类到细粒度的函数、属性进行分析,提出了影响异常行为在网络化软件中传播的三类因素,包括错误传播概率、连接率及交互频率.通过求解模型的微分方程,找出各因素对异常行为传播的影响,并将其应用于典型的网络化软件系统,用实证的方法验证了提出的理论的正确性和可用性,为后续的异常行为牵制控制等工作提供了有力依据.

2　相关工作

近年来,国内外研究人员从设计阶段[12],编码阶段[12],测试阶段[13,14]以及系统可靠性评估阶段[15]做出了有益的尝试.他们从不同的角度,基于系统级或组件级建立了分析模型,对异常行为进行检测和定位,为提高系统的稳定性和可靠性起到了一定的作用.另外,Avizienis 等[16]指出异常行为建模时应该考虑多层故障,单一的逻辑值(有/无故障)不能充分表达一些关键的反常行为,作者还强调了系统组件之间的错误传播过程的重要性.然而,大多数建模方法并没有关注基于组件系统中的错误传播[17],并且,据我们所知,从多粒度的角度构建异常行为传播模型更是鲜有提及.基于其它的因素方面,Elmqvist 和 Nadjm-Tehrani[18]提出具有安全接口的形式化建模方法,并提供关于系统安全性能的组合推理.Grunske 和 Neumann[19]使用故障诊断及转换符号 FDTN(Failure Diagnosis and Transformation Notation)建立系统的错误行为模型,从该模型中推导出系统组件故障树 CFT(Component Fault Trees),进行安全性分析.Rugina[20]等利用体系结构分析与设计语言 AADL(Architecture Analysis and Design Language)的误差模型附件创建复合系统错误行为模型,然后将这些模型转化为随机 Petri 网或马尔科夫链进行分析,Joshi[21]等还将这些错误行为模型转换为故障树进行进一步的研究.然而,这些工作的重点主要集中在已知错误行为的模型构建上,而对影响错误行为传播的因素、提高模型的完备性等属性方面还存在很大的研究空间.

3　异常行为传播机制

3.1　系统异常传播模型

图 1 为多粒度的网络化软件系统错误传播模型.每一个组件由若干类构成,类的实现中包含了各种功能的函数.函数间的调用、类与类之间的相互关联、组件之间的消息传递均用有向边表示,组件、类及函数为不同粒度的节点.同时,模型中给出了影响异常行为传播的 3 类参数.

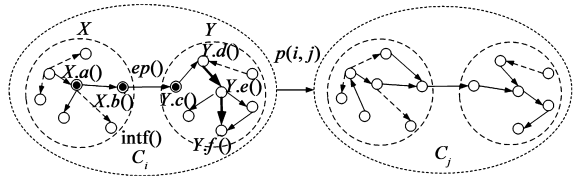

图1　系统异常传播模型及参数

组件 i 内部的错误率 intf(i),是指实现组件 i 中隐含错误的概率.然而,组件内部的错误不一定会传播至其他组件而使系统产生不正确的结果.因此,在模型中引入参数 $ep(i)$,它表示组件 i 内部的错误传播概率.假定系统由 C 个交互组件构成,用 $p(i,j)(1\leqslant i,j\leqslant C)$ 表示组件之间的错误传播概率,显然,$(\forall i)\ \sum_j p(i,j)=1$,且模型中允许自转换,即 $p(i,i)\neq 0$.

函数之间的调用关系是连接率的直观反映.软件系统的执行过程构成了节点相对固定,边数随时间变化的时变网络 $G(V,E(t))$.其中,N 为节点集,$E(t)$ 为动态的有向边集.图 1 中有向实线表示了某一时刻函数之间的调用,有向虚线表示的是可能存在的调用关系.图 1 中带黑点的圆圈表示隐含错误的节点.

交互频率 $\mu(t)$ 指两个节点之间在某个时间段内调用与被调用的次数.某些隐含错误的节点在少数几次执行时不会将错误传播至与其交互的节点,但是当交互次数达到上限值时,可能引起其他调用该节点的节点产生错误.交互频率的大小也体现了网络化软件中某些操作的执行频繁度和重要性.图 1 中用加粗的有向实线表示节点之间的多次交互.

3.2　模型建立

3.2.1　相关定义

定义 1　(组件内部错误概率)组件内部错误概率 intf(i).它是一个静态估计值.

$$\mathrm{intf}(i)=\frac{FD\cdot l_i}{1000} \tag{1}$$

其中,l_i 为组件 i 包含的代码行数,FD 为错误密度.

定义 2　(组件内部错误传播概率)组件内部错误传播概率 $ep(i)$.首先依据 CK 六类标准[22]提出类的错误倾向指数 FPIC(Failure Proneness Index of Class),再通过 FPIC 推导组件内部错误传播概率 $ep(i)$.

$$\mathrm{FPIC}_X=\alpha\frac{\mathrm{CBO}_X}{\mathrm{CBO}_i}+\beta\frac{\mathrm{WMC}_X}{\mathrm{WMC}_i}+\gamma\frac{\mathrm{RFC}_X}{\mathrm{RFC}_i} \tag{2}$$

其中,CBO_X,WMC_X,RFC_X 分别为类 X 的 CBO、WMC、

RFC 属性值,CBO_i,WMC_i,RFC_i 分别为组件 i 中所有类的CBO、WMC、RFC属性值之和. α,β,γ 是三个属性对应的权重,根据对错误倾向的影响大小确定.已知组件内所有类的错误倾向指数,可将组件内部错误传播概率 $ep(i)$ 描述为:

$$ep(i)=\frac{1}{M_C}\sum_{X=1}^{M_C}\mathrm{FPIC}_X \tag{3}$$

其中,M_C 为组件 i 内包含的类总数.

定义 3 (组件间错误传播概率) 组件间错误传播概率 $p(i,j)$ 可定义为:

$$p(i,j)=P(\lfloor C\rfloor_j(p)\neq\lfloor C\rfloor_j(q)\mid p\neq q) \tag{4}$$

其中,$\lfloor C\rfloor_j$ 表示组件 j 实现的功能,它包含了组件 j 中函数调用产生的节点状态[9]转换,以及由组件 j 执行的输出结果. p、q 为组件 i 与组件 j 之间通信时传递的消息.公式(4)给出的传播概率可用组件 j 中的状态的产生、状态之间的转换概率及组件 i 与组件 j 之间的消息传播的熵来描述:

$$p(i,j)=\frac{1-\sum_{p\in S_{C_j}}P_{C_j}(p)\sum_{q\in S_{C_j}}P_{C_i,C_j}[F_p^{-1}(q)]^2}{1-\sum_{m\in M_{C_i,C_j}}P_{C_i,C_j}[m]^2} \tag{5}$$

其中,S_{C_j} 为组件 j 中的状态,P_{C_j} 为组件 j 中产生的状态概率分布,P_{C_i,C_j} 为组件 i 与组件 j 之间传递的消息 M_{C_i,C_j} 的概率分布,转换函数 $F_p^{-1}(q)=\{m\in M_{C_i,C_j}\mid F_p(m)=q\}$,为状态 p 转换为状态 q 时需输入的消息 m. $\sum_{q\in S_{C_j}}P_{C_i,C_j}[F_p^{-1}(q)]^2$ 与 $\sum_{m\in M_{C_i,C_j}}P_{C_i,C_j}[m]^2$ 为二阶 Renyi 熵指数.我们假定组件 j 中的状态 S_{C_j} 与组件 i 传播至组件 j 的消息 M_{C_i,C_j} 是等概率分布的,那么公式(5)可简化为:

$$p(i,j)=\frac{1-\frac{1}{|S_{C_j}|\,|M_{C_i,C_j}|^2}\sum_{p\in S_{C_j}}\sum_{q\in S_{C_j}}|F_p^{-1}(q)|^2}{1-\frac{1}{|M_{C_i,C_j}|}} \tag{6}$$

在未知状态转换函数 F 的情况下,假定 S_{C_j} 中的每一个初始状态被触发转换到另一个新状态所需的消息数是大致相同的,那么,可求得 $P(i,j)$ 的上限值,即:

$$p(i,j)\leqslant\frac{1-\frac{1}{|S_{C_j}|}}{1-\frac{1}{|M_{C_i,C_j}|}} \tag{7}$$

定义 4 (多步传播概率) $err^{(k)}(i,j)$:给定执行从组件 $i(0\leqslant i,j\leqslant C)$ 开始,经过 $k(k\geqslant 0)$ 步控制转移之后到达组件 j 并产生错误输出的概率.我们可以用以下递归公式,将概率 $err^{(k)}$ 与 $err^{(k-1)}$ 关联起来:

$$err^{(k)}(i,j)=p^{(k)}(i,j)\cdot\mathrm{intf}(j)+ep(j)\cdot(1-\mathrm{intf}(j))\cdot\sum_{h=0}^{C}err^{(k-1)}(i,h)p(h,j) \tag{8}$$

当 $k<0$ 时,$err^{(k)}(i,j)=0$,且有 $err^{(0)}(i,j)=\mathrm{intf}(j)(\forall i=j)$,$err^{(0)}(i,j)=0(\forall i\neq j)$.

定义 5 (连接率) 考虑有向图 $G(V,E(t))$,连接率 $c(t)$ 可表示为:

$$c(t)=\frac{|E(t)|}{|V|\cdot(|V|-1)} \tag{9}$$

其中,$|E(t)|$ 为时刻 t 软件系统交互过程形成的有向图 G 中的边数,$|V|$ 为节点个数.连接率描述了节点之间连接程度强弱,连接率越大,节点之间连接程度越高,异常行为在这种情况下传播的可能性越大.

定义 6 (交互频率) $\mu(t)$:软件系统交互过程形成的有向图 G,由于 G 中节点之间的连边是随时间变化的,节点之间的这种可变交互用邻接矩阵 $\boldsymbol{A}(t)$ 表示.某时刻 t,节点 i 与其邻居节点之间的连接情况可用 i 的邻域 $\boldsymbol{V}_i(t)$ 描述,它表示的是矩阵 $\boldsymbol{A}(t)$ 中的第 i 行向量,即 $\boldsymbol{V}_i(t)=\{a_{ij}(t)\mid a_{ij}(t)\in\boldsymbol{A}(t),j=1,2,\cdots,N\}$,节点 i 与节点 j 之间在时刻 t 有连接,$a_{ij}(t)=1$,反之,$a_{ij}(t)=0$.多个时间片段,可形成多个邻接矩阵 $\boldsymbol{A}(t)$,将这些邻接矩阵叠加起来,构成某个时间段 T 内节点之间的交互频率矩阵 $\boldsymbol{B}$,即 $\boldsymbol{B}=\sum_{t=1}^{T}A(t)$,此时,交互频率矩阵 $\boldsymbol{B}$ 中节点 i 的邻域 $\boldsymbol{V}_i(\boldsymbol{T})$ 表示它与其邻居节点在时间 T 内交互的次数,因此,我们可以进一步将节点 i 与 j 之间的交互频率表示为:$\mu_{ij}(t)=b_{ij}/\sum_{i=1}^{N}\sum_{j=1}^{N}b_{ij}$,而整个网络 G 的交互频率为 $\mu(t)=\sum_{i=1}^{N}\sum_{j=1}^{N}d_i\mu_{ij}(t)$,$d_i$ 为节点 i 的度.

3.2.2 模型建立

本文采用两状态仓室模型中的 SI 模型及个体模型对软件异常行为传播进行分析. $S(t)$ 表示健康节点数,$I(t)$ 表示隐含错误节点数.图 2 表示两状态仓室模型,α 为感染强度.

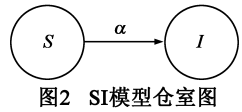

图2 SI模型仓室图

由于我们考虑的是网络化软件系统,其交互行为复杂,节点数相对较大,因此采用确定性仓室传播模型,每一个仓室的平衡方程为:

$$\begin{cases}\dfrac{\mathrm{d}S(t)}{\mathrm{d}t} = -\alpha S(t)\\ \dfrac{\mathrm{d}I(t)}{\mathrm{d}t} = \alpha S(t)\end{cases}$$

某时刻 t,隐含错误的节点 i 在已知连接率 $c(t)$的情况下,至多与图 G 中的 $c(t)(N-1)$个节点相连,而这些节点中未被错误感染的比例为$(1-I(t)/N)$,那么,$c(t)(N-1)(1-I(t)/N)$为时刻 t 与节点 i 相连的健康节点数 $S(t)$,节点之间的交互频率为 $\mu(t)$,节点之间的传播概率为 β,由上文定义可知 β 针对不同粒度的节点,采用不同的概率计算方法.从而,健康节点被感染的比率为:$\beta\mu(t)S(t)I(t)/N$,感染强度:

$$\alpha = \frac{\beta\mu(t)}{N}I(t)$$

感染速率:

$$a = \frac{\beta\mu(t)}{N} \tag{10}$$

故

$$\frac{\mathrm{d}I(t)}{\mathrm{d}t} = aS(t)I(t) \tag{11}$$

其中,$S(t) = c(t)(N-1)(1-I(t)/N)$,可得异常行为传播分析模型:

$$\frac{\mathrm{d}I(t)}{\mathrm{d}t} = ac(t)(N-1)(1-I(t)/N)I(t) \tag{12}$$

当 N 较大时,$N-1\approx N$,上式可简化为:

$$\frac{\mathrm{d}I(t)}{\mathrm{d}t} = Nac(t)I(t) - ac(t)I^2(t)$$

求得其通解为:

$$I(t) = \frac{N}{1+CNe^{-ac(t)Nt}}$$

其中 C 为常量,假定初始时刻 $t=0$ 时,图中仅有一个隐含错误的节点,即 $I(0)=1$,有

$$I(0)=1\Rightarrow\frac{N}{1+CN}=1\Rightarrow C=\frac{N-1}{N}$$

从而最终解为:

$$I(t) = \frac{N}{1+(N-1)e^{-ac(t)Nt}}$$

可得错误节点的扩散比率:

$$i(t) = \frac{I(t)}{N} = \frac{1}{1+(N-1)e^{-ac(t)Nt}} \tag{13}$$

4　实验及分析

我们以目前典型的网络化软件系统——开源在线电子商务购物系统(Web Shop)为例来验证异常行为传播模型,图 3 为该电子商务购物平台架构.该系统主要构成组件为 C_1-C_8.多用户可同时通过分散在不同地理位置的 GUI 登陆或注册进入购物平台,它们与云数据中心的确认组件(Identifier)及数据库管理系统组件(DBMS)进行交互,触发确认任务,然后,并发进入商品市场组件(Market),进行货物选购、下订单等事务,与云数据中心账户管理组件(Account Manager)构成系统的核心.后者通过与其他所有组件(如消息组件(Messenger)、验证机构(Verifier)、支付平台(Payment Platform))交互,使用数据库系统提供的数据,管理来自用户的操作请求.

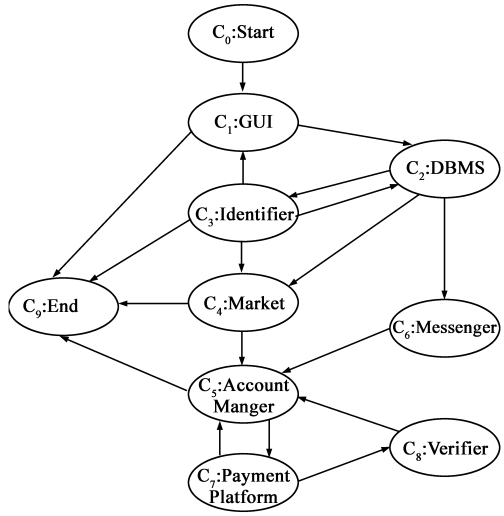

图3　电子商务购物平台架构

4.1　错误传播概率

以 GUI 组件为例,它实现了用户登录,注册并发送确认邮件的功能,表 1 给出了 GUI 组件中包含所有类的 CK 标准属性值,由开源软件 CKJM 以及 Eclipse 标准插件获取.根据公式(3)可计算出相应组件内错误传播概率 $ep()$.见表 2.

表 1　组件 GUI 内部各类 CK 标准属性值

类名	RFC	CBO	WMC	FPIC
javax.mail.Properties	16	15	2	0.13896
javax.mail.Session	7	4	1	0.05193
javax.mail.Transport	21	18	6	0.21087
javax.mail.MimeMessage	5	3	2	0.05103
javax.mail.Internet Address	3	7	4	0.08042
javax.mail.Internet.MimeMultpart	2	5	4	0.06883
javax.mail.Internet.MimeBodyPart	6	4	2	0.05868
javax.activation.DataHandler	17	16	6	0.18826
javax.activation.FileDataSource	14	12	5	0.15099

表 2　组件内错误传播概率 $ep()$

C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
0	0.11	0.56	0.25	0.74	0.62	0.08	0.43	0.23	0

图 4 为部分组件交互形成的动态模型及各组件内部状态转换图.组件之间传递的消息集合可用 Rose-RT

及相似工具获取,输入消息为:Msg1(C_3_C_4),输出消息为:Msg1(C_4_C_5),Msg2(C_4_C_5),Msg3(C_4_C_5),Msg4(C_4_C_5),Msg5(C_4_C_5),由此,根据公式(7)可计算出组件 C_4 与 C_5 间的错误传播概率上限值,有 $C_i=C_4$,$C_j=C_5$,$S_{C_5}=2$,$M_{C_4,C_5}=5$,$P(4,5)\leqslant(1-0.5)/(1-0.2)=0.625$.依次类推,可得系统中任意两个组件之间可能的错误传播概率,另外,根据公式(1),可得各组件内部的错误率 intf(),见表 3.

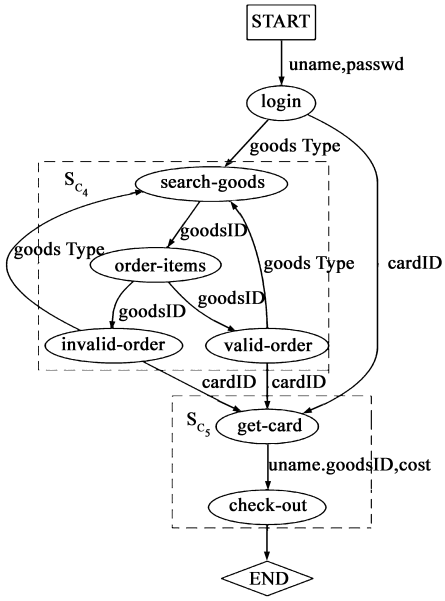

图4 动态模型[9]及各组件内状态转换

表 3 组件间错误传播概率

	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	intf()
C_0	0	1	0	0	0	0	0	0	0	0	0
C_1	0	0	0.999	0	0	0	0	0	0	0.001	0.019
C_2	0	0	0	0.235	0.667	0	0.098	0	0	0	0.032
C_3	0	0.056	0.026	0	0.917	0	0	0	0	0.001	0
C_4	0	0	0	0	0	0.625	0	0	0	0.375	0.007
C_5	0	0	0	0	0	0	0	0.9377	0	0.0623	0.005
C_6	0	0	0	0	0	1	0	0	0	0	0
C_7	0	0	0	0	0	0.01	0	0	0.99	0	0
C_8	0	0	0	0	0	1	0	0	0	0	0.1021
C_9	0	0	0	0	0	0	0	0	0	1	0

4.2 交互频率

我们对某个时间段 T 软件的执行过程进行划分,将其视为多个时间片上的交互行为图 $G(V,E(t))$ 的叠加.以组件 C_4 为例,其中包含 4 个状态,分别为:search-goods、order-items、invalid-order 和 valid-order.在时间 $T=200$s,它们之间的调用关系构成的交互行为图 $G_4(V,E(t))$,其中,$V=\{V_1,V_2,V_3,V_4\}$,总的调用关系为 $|E(t)|=9$,该时间段内对应的执行踪迹如图 5 所示.那么,根据定义(6),可得时间 $T=200$s 组件 C_4 内部节点之间的交互矩阵 $\boldsymbol{B}$,假定节点 V_1 与其它外部节点之间存在交互,设 $b_{11}=1$.从而,可计算出组件 C_4 节点间的交互频率 $\mu(t)$.

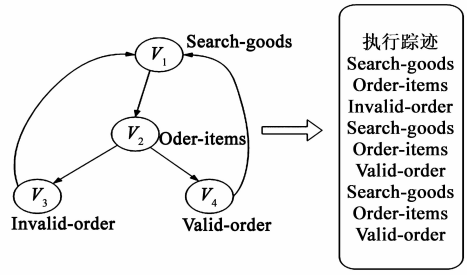

图5 组件4交互行为图及对应执行踪迹

$$\boldsymbol{B}=\begin{bmatrix}1&3&0&0\\0&0&1&2\\1&0&0&0\\1&0&0&0\end{bmatrix}\qquad[\mu_{ij}]=\begin{bmatrix}1/9&13&0&0\\0&0&1/9&2/9\\1/9&0&0&0\\1/9&0&0&0\end{bmatrix}$$

$[\mu_i]=[4/3\quad 1/29\quad 2/9\quad]$,从而,$T=200$s 时,组件 C_4 的交互频率为:$\mu(t)\approx2.99$.实验共收集组件 C_4 在时间 7×24 小时内执行踪迹日志 18000 条,统计分析发现,交互频率服从泊松分布,即 $\mu(T=t)=\dfrac{e^{-\lambda}\lambda^t}{t!}$ $(t=0,1,2,\cdots)$.

4.3 连接率

实验发现,在不同时间片上形成的交互行为图集合$\{G_1,G_2,G_3,\cdots,G_t\}$,其连接率满足幂律分布,即 $c(t)=Ht^{-\theta}$,其中,$\theta=1.53846$,$H=345.167$,且相关系数为 98.9605%.图 6 为连接率曲线,由于异常行为在软件系统中传播,随着时间推移,连接率逐渐下降,此后的剩余时间内连接率稳定在某个值.这说明程序正常执行受到错误干扰,函数之间的调用明显减少,组件的功能无法完成,软件系统处于瘫痪状态.实验还发现,通过拟合连接率,所获得的直线方程与 $c(t)$间非常近似,当 t 较大时$(t>600)$,相对误差小于 10%,如图 7.

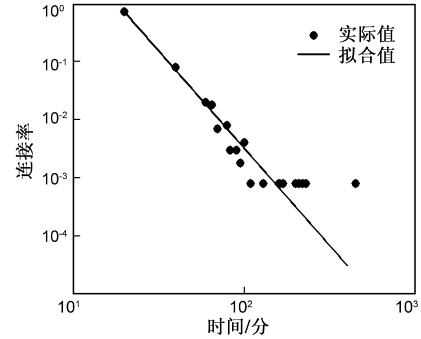

图6 连接率曲线

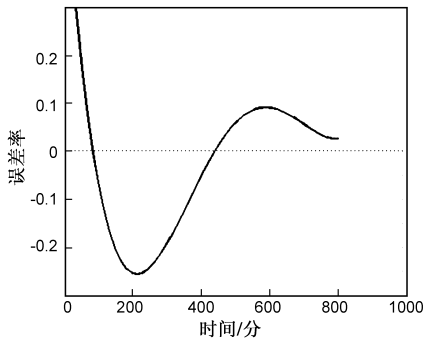

图7　相对误差曲线

4.4　扩散比率与感染速率

根据公式(13),随机获取 90 分钟内 63 个时间点形成的交互行为图上的传播概率、连接率及交互频率,计算出扩散比率的分布及其拟合曲线,如图 8.在时刻 $T=90$min 时,扩散比率达到最大值,接近 94%,表明此时系统中大部分节点受异常行为传播的影响.同样地,根据公式(10),统计系统在 400 分钟内,受异常行为影响时,其感染速率变化曲线,如图 9.在较短的时间内,一旦软件内部错误被触发,隐藏错误的节点对健康节点的感染过程加速,感染速率迅速上升,在时刻 $T=96$min 时,感染速率也达到峰值,约为 0.40,与扩散比率在这一时刻的值是相对应的.另外,感染速率与交互频率正相关,而交互频率服从泊松分布,感染速率的变化趋势也验证了这一实验结果.

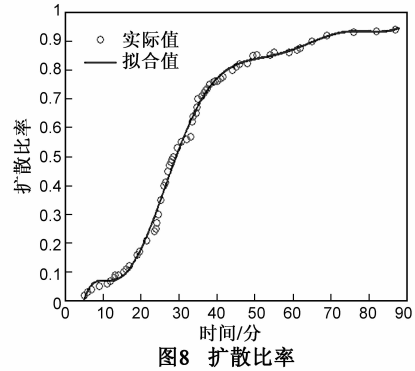

图8　扩散比率

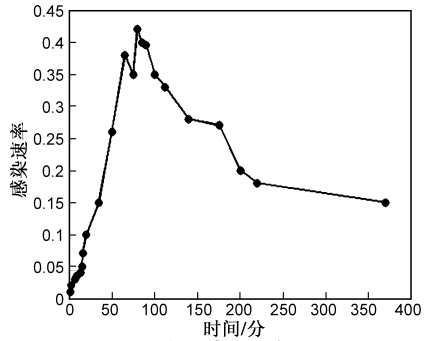

图9　感染速率

4.5　模型准确性比较

我们将模型的准确率定义为模型检测到的异常行为传播路径中包含的错误节点数与注入错误时引起异常行为传播产生错误节点平均数的比值.首先,采用工具 PROPANE 注入错误并追踪异常行为的传播过程,统计异常行为传播直至系统失效时产生的错误节点总数.图 10 为注入错误在组件 C_4 出度最大(Highest out-degree)的节点上时,经过 2 个中间节点,异常行为开始蔓延,经过 10 个中间节点后,系统的错误率达到最大值 100%;注入错误在组件 C_4 出度最小(Lowest out-degree)的节点上时,经过 4 个中间节点,异常行为开始蔓延,经过 13 个中间节点,系统的错误率达到最大值 100%,陷入瘫痪.重复上述实验过程,所有的数据都是 100 次实验的平均值.可知,系统的平均错误节点数为 12.

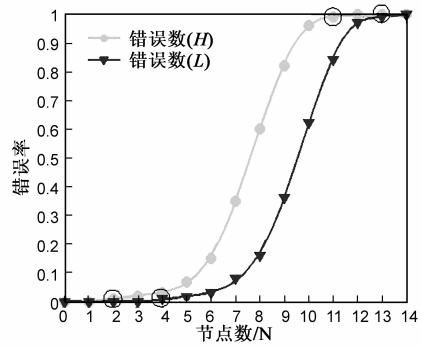

图10　错误节点分析

为了和本文提出方法对比,我们采用目前比较流行的基于离散时间马尔科夫过程建模(DTMP)方法、随机 Petri 网(SPN)及贝叶斯网(BN)建模方法进行异常行为传播分析,实验结果为 100 次运行的平均值,准确率比较如图 11.实验表明,ABPM 结合不同粒度的软件实体,综合考虑了影响异常行为传播的三类因素,模型的分析能力随着参与交互实体的增加,受系统规模干扰波动较小,准确率在 99% 左右.DTMP 将系统的执行过程假定服从马尔科夫属性,即在给定的任意时间点上,仅有一个组件可以执行,不能处理并发、同步等情况,

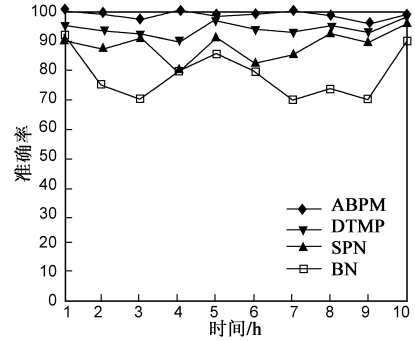

图11　准确率比较

组件的失效也是独立的.这种假定与网络化软件的实际运行场景是不相符的.因此模型的准确性也受到影响.SPN 对并发、同步及不确定性具有较强的分析能力,但它也存在着状态空间随模型规模呈指数级增长的问题,直接影响模型的准确性.BN 虽然能够处理隐含错误的节点与健康节点之间异常传播的因果关联,但它同样基于节点故障独立性假设,对系统规模也十分敏感,模型的准确性也还有待改进.

5 总结与下一步工作

研究网络化软件异常行为传播机制,对提高这种新型软件系统的稳定性和可靠性起重要作用.本文针对已有模型的不足,提出了影响网络化软件异常行为传播的因素,并给出详细的定义和计算方法,在此基础上,建立了异常行为传播模型.与其他模型相比,网络化软件异常行为传播模型从不同粒度综合考虑了软件系统执行时的动态变化过程,具有更强的表达能力,尤其是对由系统中未知错误引起的异常行为分析时,该模型更准确,更合理.但也存在亟待解决的问题:是否可挖掘其它影响异常行为传播的因素,进一步提高模型完备性;如何对异常行为进行有效控制,提高系统的稳定性等.

参考文献

[1] 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)

[2] Fang J Q, Wang X F, Zheng Z G. Research of dynamical complexity of nonlinear networks[J]. Complex Systems and Complexity Science, 2010, 7(2 – 3): 5 – 9.

[3] 何成万,张立军,张慧.基于元数据和反射的面向方面软件演化方法[J].电子学报,2011,39(8):1771 – 1777.
He Cheng-wan, Zhang Li-jun, Zhang Hui. An approach to aspect-oriented software evolution based on metadata and reflection[J]. Acta Electronica Sinica, 2011, 39(8): 1771 – 1777. (in Chinese)

[4] Sureka A. Learning to classify bug reports into components[J]. Objects, Models, Components, Patterns, 2012, 73(4): 288 – 303.

[5] Hongyu Zhang. On the distribution of software faults[J]. IEEE Transactions on Software Engineering, 2008, 34(2): 301 – 302.

[6] 陈平,韩浩,沈晓斌.基于动静态程序分析的整形漏洞检测工具[J].电子学报,2010,38(8):1741 – 1747.
Chen Ping, Han Hao, Shen Xiao-bin. Detecting integer bugs based on static and dynamic program analysis[J]. Acta Electronica Sinica, 2010, 38(8): 1741 – 1747. (in Chinese)

[7] 林闯,王元卓,杨扬,曲扬.基于随机 Petri 网的网络可信赖性分析方法研究[J].电子学报,2006,34(2):322 – 332.
Lin Chuang, Wang Yuan zhuo, Yang Yang, Qu Yang. Research on network dependability analysis methods based on stochastic Petri Net[J]. Acta Electronica Sinica, 2006, 34(2): 322 – 332. (in Chinese)

[8] Monson J S, Wirthlin M, Hutchings B. A fault injection analysis of Linux operating on an FPGA-embedded platform[J]. International Journal of Reconfigurable Computing, 2012, 2012(1): 7 – 18.

[9] 彭成,杨路明,满君丰.网络化软件交互行为动态建模[J].电子学报,2013,41(2):314 – 320.
Peng Cheng, Yang Lu-ming, Man Jun-feng. Dynamic modeling of networked software interactive behavior[J]. Acta Electronica Sinica, 2013, 41(2): 314 – 320. (in Chinese)

[10] Yukihiko Nakata, Philipp Getto, Anna Marciniak-Czochra , Tomás Alarcón. Stability analysis of multi-compartment models for cell production systems[J]. Journal of Biological Dynamics, 2012, 6(1): 2 – 18.

[11] Alsaade F, Fouda Y, Khan A R. Efficient cellular automata algorithm for template matching[J]. Journal of Artificial Intelligence, 2012, 5(3): 122 – 129.

[12] Vinay Singh, Vandana Bhattacherjee, Sandeep Bhattacharjee. An analysis of dependency of coupling on software defects [J]. ACM SIGSOFT Software Engineering Notes, 2012, 37(1): 1 – 6.

[13] 苏璞睿,冯登国.基于进程行为的异常检测模型[J].电子学报,2006,34(10):1809 – 1811.
Su Pu-rui, Feng Deng-guo. An anomaly intrusion detection model based on nonhierarchical clustering[J]. Acta Electronica Sinica, 2006, 34(10): 1809 – 1811. (in Chinese)

[14] 徐建军,谭庆平,熊磊,叶俊.一种针对软错误的程序可靠性定量分析方法[J].电子学报,2011,39(3):675 – 679.
Xu Jian-jun, Tan Qing-ping, Xiong Lei, Ye Jun. A quantitative approach for program reliability analysis of soft errors[J]. Acta Electronica Sinica, 2011, 39(3): 675 – 679. (in Chinese)

[15] Wei-Feng Pan, Bing Li, Yu-Tao Ma. Measuring structural quality of object-oriented software via bug propagation analysis on weighted software networks[J]. Journal of Computer Science and Technology, 2010, 25(6): 1202 – 1213.

[16] Avizienis A, Laprie J, Randell B, et al. Basic concepts and taxonomy of dependable and secure computing[J]. Nato Security through Science Series E Human and Societal Dynamics, 2007, 23(10): 11 – 33.

[17] Hiller, M, Jhumka, A, Suri, N. Epic. Profiling the propagation and effect of data errors in software[J]. IEEE Transactions Computers, 2004, 53(5): 512 – 530.

[18] Elmqvist J, Nadjm-Tehrani S. Safety-oriented design of com-

ponent assemblies using safety interfaces[J]. Electronic Notes in Theoretical Computer Science, 2007, 182: 57 – 72.

[19] Grunske L, Neumann R. Quality improvement by integrating non-functional properties in software architecture specification [J]. EASY, 2002, 2(1): 23 – 32.

[20] Rugina A E, Kanoun K, Kaaniche M. An architecture-based dependability modeling framework using AADL [J]. arXiv preprint arXiv: 2007, 3(10): 0704 – 0865.

[21] P. Popic, D. Desovski, W. Abdelmoez. Error propagation in the reliability analysis of component-based systems[A]. Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering[C]. Chicago: Academic Press, 2005. 10 – 15.

[22] Chidamber S R, Kemerer C F. A metrics suite for object oriented design[J]. IEEE Transactions on Software Engineering, 1994, 20(6): 476 – 493.

作者简介

彭　成　男,1982 年生于湖南,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

杨路明　男,1947 年生于江西,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.