软件动态执行网络建模及其级联故障分析

王小龙[1] 侯 刚[1] 任龙涛[1] 周宽久[1,2] 常军旺[1] 王 竹[1]
(大连理工大学软件学院 大连 116620)[1] (大连理工大学软件评测中心 大连 116620)[2]

摘 要 随着人们对软件功能需求的不断增加,软件系统的结构和规模越来越复杂。如何对复杂软件系统的拓扑结构及其质量进行有效分析和评估是软件工程中亟待解决的难题。采用复杂网络理论对软件系统进行建模和求解,将软件源代码中的函数作为节点,函数之间的调用关系看作有向边,函数调用次数作为边的权重,提出了一种软件动态执行加权网络模型的构建方法。通过对TAR、GEDIT、EMACS这3个开源软件系统的建模及网络特征分析,发现软件系统动态执行的加权拓扑网络满足小世界效应和无标度特性,即符合复杂网络特性。基于此结论,进一步利用CML(耦合映像格子)网络故障传播模型对软件系统的级联效应进行了模拟,通过实验发现了影响软件级联故障的主要因子,这些因子为软件质量保证等研究提供了重要支持。

关键词 复杂网络,软件执行路径,加权拓扑网络,CML模型,级联故障
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.08.024

Software Dynamic Execution Network Modeling and Cascading Failure Analysis

WANG Xiao-long[1] HOU Gang[1] REN Long-tao[1] ZHOU Kuan-jiu[1,2] CHANG Jun-wang[1] WANG Zhu[1]
(School of Software, Dalian University of Technology, Dalian 116620, China)[1]
(Software Evaluation & Test Center, Dalian University of Technology, Dalian 116620, China)[2]

Abstract As the functional requirements of software keep growing, the structure and scale of software systems become more and more complicated. In order to analyze the topology and quality of complex software systems, the theory of complex networks was introduced to model and solve software engineering problems. This paper regarded functions in the source code of the software as nodes, function-calls in the source code of the software as directed edges, and the number of function-calls as the weight of edges, then presented a method of constructing the weighted software dynamic execution routes topological network. The results on the statistical analysis of the networks obtained from three software programs, TAR, GEDIT and EMACS show that the weighted network of the software execution process fits in with the small-world effect and the scale-free property of complex networks. Based on that, we further took advantage of the CML (Coupled Map Lattice) model in complex networks to simulate and analyze the cascading effect for software systems and discovered the main factors that influence the cascading failures in software systems, which will give an important support for the research of software quality assurance.

Keywords Complex networks, Software execution route, Weighted topological network, CML model, Cascading failure

1 引言

随着人们对软件功能的需求越来越高,软件自身以及不同软件之间的结构也日益复杂,函数与函数、模块与模块、系统与系统等实体之间的相互作用使得软件系统的整体结构表现出非常复杂的特性。如果把实体抽象为节点,不同软件实体之间的关系抽象为节点之间的边,则可以对利用软件系统建立网络模型进行不同层次的分析。近年来,针对软件系统的拓扑结构建模已有了深入研究,现有的研究结果表明软件系统静态拓扑结构具有复杂网络特点[1-4],即小世界性和无标度性等统计特性。2002年,Valverda等人[5]首先将复杂网络方法引入到软件系统拓扑结构分析中,将类作为节点、类之间的继承和依赖等关系作为边,使用无向网络来表示软件系统的结构,发现软件系统的网络结构具有"小世界"和"无标度"特征[6]。但无向网络忽略了类之间关系的方向性,因此实验结果受到了质疑。2003年Myers[7]进一步使用有向网络从复杂网络角度对Linux、MySQL、XMMS等软件的结构特性进行了细致分析,发现了同样的现象,并且提出了系统鲁棒性的分析思路和策略。此后很多研究人员和研究机构使用复杂网络理论建模和分析软件内部结构,借助复杂网络的数学理论和方法分析软件网络的各种统计特性,对软件进行质量度量及优化。2004年,美国卡内基·梅隆大学软件工程研究所

到稿日期:2013-07-01 返修日期:2013-07-25 本文受国家自然科学基金(61272174)资助。
王小龙(1989-),男,硕士生,主要研究方向为软件可靠性、嵌入式系统建模,E-mail:wxl_dut@163.com(通信作者);侯 刚(1982-),男,博士生,讲师,主要研究方向为软件可靠性、复杂性理论;任龙涛(1988-),男,硕士生,主要研究方向为软件可靠性、形式化验证;周宽久(1966-),男,博士,教授,主要研究方向为软件可靠性、嵌入式系统建模;常军旺(1989-),男,硕士生,主要研究方向为软件可靠性、模型检测;王 竹(1992-),男,主要研究方向为软件测试。

发布了复杂巨系统软件专项调研，试图解决复杂巨系统的软件工程问题。2005 年，13 个跨国 IT 公司（包括 IBM、HP、Nokia 等）宣布合作研制网络化软件。2006 年，我国学者何克清和李德毅等人将复杂网络理论和方法引入到软件工程的设计之中[8]。蔡开元教授将软件系统的执行过程视为一个演化的有向复杂网络结构，引入了软件镜像图概念，并发现虽然在拓扑层面上软件执行过程表现出小世界现象，但在时间角度上不再体现小世界特征，其度分布可以表示为幂律分布也可以表示为指数函数[9]。

由于软件系统不同模块、函数之间彼此联系，因此一个模块出现故障或受到威胁会引起邻居模块或函数的故障，进而逐步扩散造成系统故障。其具体的过程如图 1 所示，黑色节点表示发生故障的节点，白色节点表示没有发生故障的节点。节点 6 在 $t+1$ 时刻发生故障，由于节点 3 调用了节点 6，在 $t+2$ 时刻，节点 3 也发生故障，在 $t+3$ 时刻，节点 1 和 2 也发生了故障，随着时间的推移，故障在节点之间传播。因此，如果一个或者少数几个函数发生故障，该故障可能会随着调用或依赖关系传播至其它函数而引发其它函数无法正常运行，最终导致部分或者整个系统崩溃，这种现象称为“级联故障”[10]。

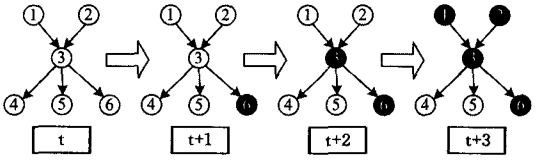

图 1　级联故障的扩散过程示意图

例如，1986 年 10 月，因特网第一次拥堵崩溃，劳伦斯伯克利实验室和加州大学伯克利分校之间的距离仅有 200 米，因为相继故障，使得它们之间的连接崩溃[11]；1996 年 6 月 4 日，阿丽亚娜 5 型火箭由于整型加速度值产生溢出，造成以加速度为参数的速度、位置等变量计算错误，程序只得进入异常处理模块，引爆自毁；2009 年 9 月 12 日，通讯软件 MSN 美国总部服务器瘫痪超过 1 小时，导致全球 900 万用户受到影响；2010 年初，由于一个软件兼容性故障导致美国约有 8000 至 10000 台 GPS 接收机失效[12]。

近几年，基于复杂网络理论对软件系统的研究工作大多集中在从不同层次和粒度（包、类和方法）建立软件系统的无权网络模型，揭示软件网络的一些普遍拓扑特性[13-15]。但是无权网络模型忽视了节点之间关系的差别以及节点自身属性和动力学行为对整个软件系统的影响，因此目前的研究必须考虑用加权有向网络来描述软件系统结构，并应用动力学模型分析软件系统级联效应，研究软件质量特征[16,17]。本文将软件源代码中的函数看作节点，函数间调用关系看作有向边，函数调用次数作为权重，给出了构建软件系统动态执行加权网络模型的方法。通过对 Linux 下 3 个开源软件 TAR、GEDIT、EMACS 加权网络特征进行分析，发现软件系统动态执行过程的加权网络具有小世界效应和无标度特性。最后引入 CML 耦合映像格子模型[18]，分析了影响软件系统级联效应的主要因子。

2　软件动态执行加权网络模型

2.1　网络模型建立

GNU 有许多开源软件项目，选取其中常见的 3 个 C 语言程序，即 TAR 程序、GEDIT 程序和 EMACS 程序作为实验对象。在 Linux 操作系统中，我们使用 GCC 编译器对 GNU 下的开源软件进行分析，通过增加一些调试信息，自动获取软件执行路径和相应拓扑信息。

为了构建软件动态执行网络，我们将软件动态执行网络表示为具有 n 个节点、e 条边的加权有向图 G，记为 $G=(V,E)$，其中 V 是节点集合，V 中的每个元素 v_i 代表软件源代码中的一个函数，E 是边的集合，E 中的每个元素 $\langle v_i,v_j\rangle$ 是一个有序对，当且仅当 v_i 调用 v_j 时，$\langle v_i,v_j\rangle\in E$，即 $v_i->v_j$。在现有大多数基于函数实体的软件网络模型中，节点间只有“连接”和“不连接”两种方式对应着函数间的“调用”和“不调用”两种关系，但这不能准确地反映函数之间实际的紧密程度。因此，本文在建立软件系统的网络模型时，为每条边赋予一个权值 W_{ij}，W_{ij} 为多次实验得到的 $v_i->v_j$ 的实际平均调用次数。

由于本文建立的是软件系统动态执行网络，整个网络的拓扑结构不可能通过软件的一次执行就全部得到，因此需要多次从不同使用角度运行软件，得到足够多的软件执行路径图，再将每次运行得到的执行路径图进行合并，从而获取软件的整体动态执行网络。在实验中，每个软件的动态执行网络均为 1000 次执行路径的合并，合并后的网络节点数与其静态分析所得节点数相同，从而保证了软件动态执行网络的完整性。

动态执行网络图的合并方式如下：初始网络为空，对于每次测试得到的执行路径图中的节点，新增节点直接加入网络中，已存在的节点被忽略。对于每次测试得到的执行路径图中的边，新增边直接加入网络中并记录其初始权重，同时与之相对应的计数器被设置为 1；已存在的边不再加入网络中，但其权重需要累加到已存在边的权重上，同时与之相对应的计数器增加 1；当所有节点和边都被合并到网络中之后，即形成了所需的软件执行过程的加权网络拓扑图，其边权重需按式(1)重新计算（平均权重），其中 w_k 是第 k 次向节点 i 到节点 j 之间边累加的权重，n 是节点 i 和节点 j 之间边对应的计数器值。

$$w_{ij}=\frac{1}{n}\sum_{k=1}^{n}w_k \tag{1}$$

下面给出上述过程的形式化描述：设软件 n 次执行得到的加权执行路径图分别为 $G_1=(V_1,E_1)$，$G_2=(V_2,E_2)$，…，$G_n=(V_n,E_n)$，n 次执行路径合并后的动态执行网络图为 $G=(V,E)$，合并过程必须满足下面 3 个条件：

条件 1（节点合并条件）：

$$V=V_1\cup V_2\cup\cdots\cup V_n=\bigcup_{k=1}^{n}V_k \tag{2}$$

条件 2（有向边合并条件）：

$$E=E_1\cup E_2\cup\cdots\cup E_n=\bigcup_{k=1}^{n}E_k \tag{3}$$

条件 3（权值合并条件）：合并后的动态执行网络图 G 的边 $\langle v_i,v_j\rangle$ 权重为

$$W_{ij}=\frac{\sum_{k=1}^{n}Y_{ij,k}}{\sum_{k=1}^{n}X_{ij,k}} \tag{4}$$

其中，$X_{ij,k}$ 和 $Y_{ij,k}$ 定义为

$$X_{ij,k}=\begin{cases}0, & \langle v_i,v_j\rangle\notin E_k\\ 1, & \langle v_i,v_j\rangle\in E_k\end{cases}$$
$$Y_{ij,k}=\begin{cases}0, & \langle v_i,v_j\rangle\notin E_k\\ w_{ij}, & \langle v_i,v_j\rangle\in E_k\end{cases} \tag{5}$$

图 2 给出了合并后的 GEDIT 程序动态执行加权网络图。

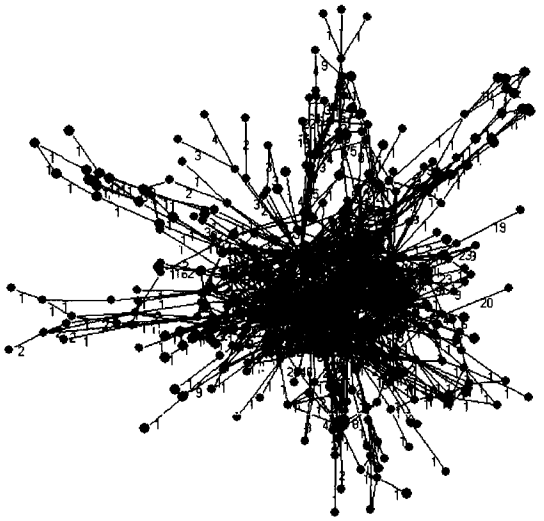

图 2 GEDIT 程序的动态执行网络图

2.2 网络特征统计分析

无权网络忽略了边的权重，因此仅能体现软件的拓扑结构。但从实际情况来看，函数间的调用次数也是影响软件系统特性的重要因素。因此，加权网络不仅能够体现软件的拓扑结构，同时还能够反映出函数节点间实际的耦合程度，这对于软件系统的级联故障分析有着重要的作用。所以，在实验中我们将分析软件动态执行加权网络的统计特性。

(1)点强度和点强度分布

在加权网中，与节点度 k_i 相对应的自然推广就是点强度或者点权 S_i[19,20]，其定义为：

$$S_i=\sum_{j\subseteq N_i} w_{ij} \tag{6}$$

其中，N_i 是与节点 i 直接相连的节点集合。同时，点强度又分为入点强度 IS_i 和出点强度 OS_i，分别为节点 i 的所有入边/出边权重之和。点强度分布 $P(s)$ 描述网络中一个节点的点强度大于等于 s 的概率。

对于本实验的 3 个对象，我们在双对数坐标系下得出了它们的累积入点强度分布曲线，如图 3 所示。从曲线的线性趋势上可以看出，函数节点的入点强度分布满足幂律分布。

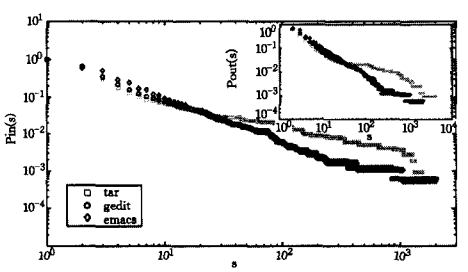

图 3 入点强度和出点强度累积分布图

在图 3 的子图中也给出了 3 个程序在双对数坐标系下的出点强度累积分布曲线，从曲线的线性趋势上看，出点强度也满足幂律分布的形态。

(2)权重和权重分布

权重在加权网络中是重要参数，在式(1)中已经给出了它的定义和计算方法，权重分布可用式(7)表示，其中 m 表示节点集合的总数。

$$P(w)=\sum_{i=1}^{m}\Pr\{W_{ij}=w\} \tag{7}$$

通过实验，我们得到的累积权重分布如图 4 所示。从权重的分布图不难看出，软件执行路径的加权网络的权重同样也具有无标度特性。

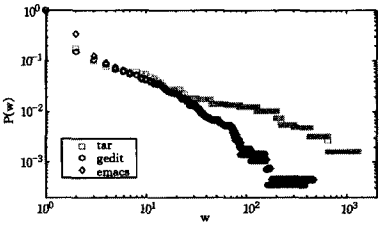

图 4 权重累积分布图

(3)平均路径长度

在加权复杂网络中，并没有明确的距离概念。每条边上的距离可以看作是权重的某种函数[21]。本文对每条边上的距离进行了重新定义：

$$L_{ij}=k_i*[(1-\frac{w_{ij}}{s_i})/(k_i-1)] \tag{8}$$

其中，L_{ij} 表示相邻节点 i 和 j 之间的距离，k_i 表示节点 i 的出度，w_{ij} 表示节点 i 和节点 j 之间边的权重，s_i 表示节点 i 的出点强度。依据式(8)重新计算加权复杂网络中任意两个节点间最短距离 d_{ij} 并计算均值，即可得到软件系统加权网络的平均路径长度 L。

(4) 聚类系数

在 Watts 给出无权复杂网络聚类系数定义的基础上，Petter 等人给出了加权复杂网络聚类系数的定义[22]：

$$c^w(i)=\frac{\sum_{jk} w_{ij}w_{jk}w_{ki}}{\max_{ij}w_{ij}\sum_{jk} w_{ij}w_{ki}} \tag{9}$$

表 1 给出了对加权网络进行实验得到的统计数据。S 表示所有节点强度的平均值(在网络中所有节点的入点强度等于出点强度)，L 表示节点间的平均路径长度，C 表示网络的聚类系数。作为对比，L_{rand} 表示相同规模的随机网络平均路径长度，C_{rand} 表示相同规模随机网络的聚类系数，文献[18]给出了 L_{rand} 和 C_{rand} 的计算方法。

表 1 加权网络的统计信息

实验对象	节点数	边数	S	L	L_{rand}	C	C_{rand}
TAR	1065	1807	22.854	1.357	13.181	0.039	0.0016
GEDIT	1645	2428	5.581	1.222	19.021	0.037	0.0009
EMACS	1827	2375	6.062	1.226	28.634	0.024	0.0007

从表 1 中可以看出，在加权网络中平均路径长度 L 远小于 L_{rand}，而聚类系数 C 远大于 C_{rand}，因此，软件动态执行过程的加权网络具有小世界效应。

通过上述实验数据的分析，我们可以得出软件系统动态执行的加权网络满足小世界性和无标度性，因此软件动态执行网络满足复杂网络特性。

3 软件系统故障级联效应分析

在前文已经证明了软件系统动态执行的加权网络满足复杂网络特性，下面使用耦合映像格子(Coupled Map Lattice，CML)模型[18]来分析软件系统的级联故障。

3.1 CML 模型介绍

CML 最初由 Kaneko[18]于 1984 年提出，是一个时间、空

间都离散而状态保持连续的非线性动力学模型，近年来已经被广泛应用于研究复杂系统的时空动力学行为，在神经网络[23]、城市交通[24]等方面都取得了广泛的推广和应用。

在 CML 模型中，节点状态值的计算不仅考虑了节点之间的耦合关系，还考虑了节点自身的波动情况，这一点在研究软件系统的动力学行为中尤为重要。函数节点是否发生故障，不仅与它相关联的函数故障情况有关，同时还与函数自身的状态相关。结合软件系统故障传播特点，即当函数 i 调用函数 j 时，j 发生故障，故障将由 j 向 i 传播，包含 N 个节点的有向 CML 模型如式(10)所示：

$$x_i(t+1)=|(1-\varepsilon)f(x_i(t))+\varepsilon\sum_{j=1,j\neq i}^{N}w_{i,j}f(x_j(t))/s(i)| \tag{10}$$

其中，$x_i(t)$表示第 i 个节点在 t 时刻的状态，w_{ij}表示节点 i 到节点 j 出边权重，$s(i)$表示节点 i 的出边强度。ε 表示耦合强度，即 ε 越大，节点间关联也就越紧密。函数 f 表示节点自身的动态行为，这里选择混沌 Logistic 映射：$f(x)=4x(1-x)$。绝对值符号保证各节点的状态非负。

若节点 i 的状态在 m 个时序内始终在(0,1)范围内，即 $0<x_i(t)<1, t\leqslant m$，那么称节点 i 处于正常状态；若在 m 时刻，节点 i 的状态 $x_i(m)\geqslant 1$，那么称节点 i 在此时刻发生故障，且该节点在以后的任意时刻状态都衡等于零。如果所有节点的初始状态都在(0,1)范围内，且没有外部扰动，那么所有节点都将永远保持正常状态；如果在 t 时刻给节点 b 施加一个外部扰动 $R\geqslant 1$，如式(11)所示：

$$x_b(t+1)=|(1-\varepsilon)f(x_b(t))+\varepsilon\sum_{j=1,j\neq b}^{N}w_{bj}f(x_j(t))/s(b)|+R \tag{11}$$

那么节点 b 在 t 时刻发生故障。在 $t+1$ 时刻，所有与节点 b 相连的节点都将受到 t 时刻 b 节点的状态 $x_b(t)$的影响，并且这些节点的状态按照式(10)计算得出。此时计算的节点状态值也可能大于 1，从而引发新一轮的节点故障。这个过程反复进行，节点故障就将扩散，直到稳定。

软件系统相继故障传播实验过程如下：初始时刻，将外部故障 $R(R\geqslant 1)$施加到一个或多个节点上，这些节点会立即发生故障，并逐步传播到调用它们的节点上，直到没有新的故障节点为止，此时记录发生故障的节点数在整个网络中的故障规模。

3.2 模型参数选择

软件系统级联故障分析的关键问题是通过模拟实验找出影响级联故障扩散的因素。在实验中，我们通过 CML 模型分别对 GEDIT、EMACS 和 TAR 3 个软件动态执行过程的加权网络进行分析，下面给出在本实验中 CML 模型参数的选择方法。

(1)ε-耦合系数

参数 ε 体现软件系统的耦合强度，ε 越大，系统内部各节点间的紧密程度也就越高。一般情况下，面向过程的软件系统耦合强度高于面向对象的软件系统。本文实验中使用的 3 个开源软件均为面向过程的软件，因此，在实验中我们选择参数 $\varepsilon=0.6$。

(2)$X_i(0)$-节点初始状态值

参数 $X_i(0)$为第 i 个节点在 $t=0$ 时刻的初始状态值。对于没有发生故障的软件系统来说，各节点的初始状态值 $0<X_i(0)<1$。结合软件系统的特点，我们可以按照式(12)对各函数节点的 $X_i(0)$赋值。其中，P_k 为在函数 i 中存在编号为 k 的程序缺陷类型概率，N 为程序缺陷类型总数。

$$X_i(0)=1-\prod_{k=1}^{N}(1-P_k) \tag{12}$$

在 CML 模型中采用一阶 logistic 混沌映射：$f(x)=4x(1-x)$ 计算函数节点的自身波动情况。函数节点状态值是初始状态值经过多次迭代之后计算得到的，这个值体现出一种随机性。也就是说，函数节点的状态值是与初始值相关的随机值，因此在实验中初始值 $X_i(0)$对实验结果的影响微弱。

(3)R-外部扰动强度

为了使软件系统产生级联故障，我们需要对函数节点引入外部扰动。在 CML 模型中，外部扰动 R 的取值为 $R\geqslant 1$。R 越大，我们对网络中节点施加的故障强度也就越大。在实验中，通过对 R 设置不同的值来观察 R 对于整个网络级联故障的影响。

(4)C-注入故障数目

参数 C 为初始注入的故障数目，在实验中，我们通过攻击不同数目的节点来观察初始故障数目 C 对于整个网络级联故障的影响。

3.3 实验结果分析

在本实验中，我们通过改变模型参数来考察各参数对级联故障规模和故障传播速度的影响。实验结果中的所有曲线数据均为 50 次实验的平均。

(1)实验 1：注入故障数目 C 的影响

图 5 显示了针对 TAR×GEDIT 和 EMACS 3 款软件进行随机的攻击中，初始注入故障数目对于级联故障的影响。在实验中设置 CML 模型参数如下：$\varepsilon=0.6$，$\overline{X_0}=0.4$，$R=4$，在 $t=0$ 时刻外部注入故障的节点数分别为 10,20,30,40。从图中可以看出总的故障规模随时间逐渐增长，并最终达到一个稳定值。同时，从最终故障规模来看，少量的初始故障节点并不会诱发级联故障的大规模爆发，这也说明软件内部是允许存在少量错误的，它们的触发并不会对软件正常工作带来巨大影响。当初始故障节点数目增加时，级联故障会逐步蔓延开来，但最终被感染的节点比例依然不会超过 30%。

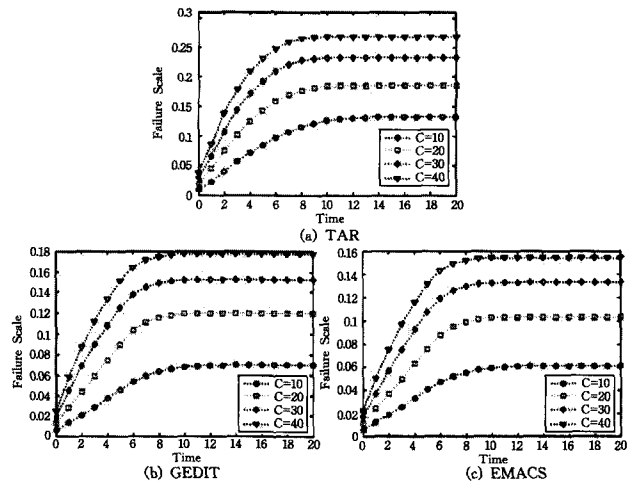

图 5 级联故障扩散过程——不同注入故障数目(C=10,20,30,40)

(2)实验 2：外部扰动强度 R 的影响

图 6 显示了针对 TAR、GEDIT 和 EMACS 3 款软件进行的随机攻击中，外部扰动强度对级联故障的影响。在实验中我们设置 CML 模型参数如下：$\varepsilon=0.6$，$\overline{X_0}=0.4$，$C=20$，在 $t=0$ 时刻，针对随机选择的 20 个节点施加强度 R 为 1.5、2.0、2.5、3.0 的扰动。与上一实验类似，故障规模随时间逐渐增大，并最终达到一个稳定值。3 种软件对故障强度表现出一致的反应，即外部扰动 R 越强，软件的故障规模就越大，同时，软件故障的扩散速度也就越快。当 $R\leqslant 1.5$ 时，软件故障几乎没有扩散；当 $R=3.0$ 时，软件故障规模达到最大。同时，我们还发现当 $R>3.0$ 时，对于同一批实验节点，即使再增强外部扰动的强度，故障规模也不再增大。

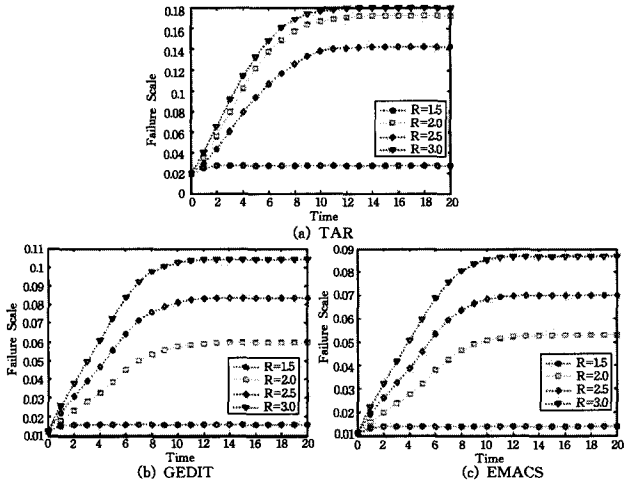

图 6 级联故障扩散过程——不同的扰动强度($R=1.5,2.0,2.5,3.0$)

(3)实验 3：随机攻击与蓄意攻击的影响

蓄意攻击一般针对入点强度高的节点进行，因此，实验中对每一软件选取入点强度排名前 10 位中的 3 个节点分别进行攻击。随机攻击中，在全部节点中随机地选择一个节点进行攻击，将结果与蓄意攻击进行比较。图 7 显示了针对 TAR、GEDIT 和 EMACS 3 款软件进行随机攻击和蓄意攻击的比较，通过实验我们发现对于入点强度较高的节点蓄意攻击会产生较大规模的软件故障，3 款软件均达到 50%左右，同时故障传播速度也较快。而随机攻击产生的软件故障规模很小，接近于 0。

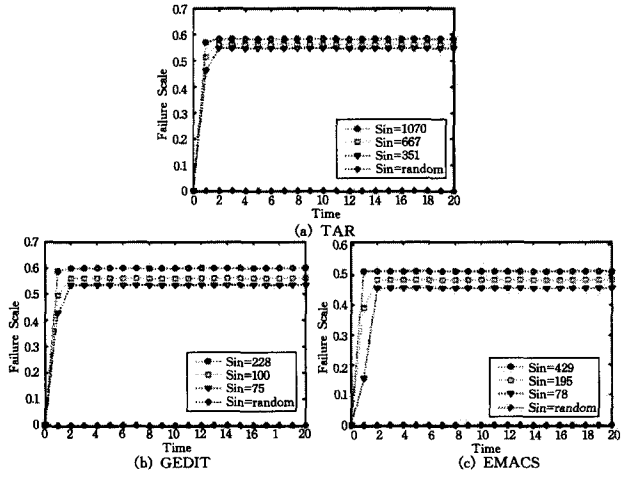

图 7 级联故障扩散过程——蓄意攻击和随机攻击

实验说明，3 款软件对于随机攻击有着较高的鲁棒性，但对于蓄意攻击则脆弱性较高。这是因为网络中大量的节点入点强度很低，只有少数节点入点强度很高。在加权网络中，入点强度高的节点不仅意味着该节点被调用的次数多，而且通过对网络形态的观察，这样的节点从结构上也往往被更多的节点调用，因此故障容易传播。所以，从软件结构上看存在着某些“关键函数”，这样的函数对整个系统稳定性的贡献更大。

更深入地分析实验曲线，比较实验 1 和实验 2 中不同被测程序最终故障规模之间的差别，我们还发现相同条件下的随机攻击，TAR 程序的故障规模要大于 GEDIT 和 EMACS。而考虑 3 个程序的结构特点，表 1 中 TAR 程序的平均入点强度和图 3 中 TAR 程序入点强度高的节点比例，均明显高于 GEDIT 和 EMACS。因此，结合实验 3 的结论，我们可以认为函数节点的入点强度大小是从结构上影响软件动态执行加权网络级联故障的关键因素。

结束语 本文提出了一种软件动态执行加权网络的建模方法，通过该方法对 TAR、GEDIT、EMACS 3 个软件系统的执行过程进行建模，并从动态角度对其网络特性进行分析。实验统计结果显示软件系统的动态执行网络具有“小世界性”和“无标度性”。在此基础上，引入 CML 耦合映像格子模型来模拟和分析软件系统的级联效应。通过实验发现，初始注入的故障数目、外部扰动强度和故障触发方式(随机攻击或蓄意攻击)是影响软件系统级联故障扩散速度和最终规模的主要因子。也就是说，更多的初始故障数目和更大的外部扰动强度将会加速软件系统级联故障的扩散，并引起更大的故障规模；入点强度大小是影响软件动态执行网络级联故障的关键因素，对软件中入点强度大的函数进行蓄意攻击相比随机攻击将会产生更大的故障规模。这些结论对从软件拓扑结构视角研究软件质量有着重要的意义。未来我们的研究将关注于把函数节点的容错能力引入级联故障模型，并建立基于动态执行网络的软件质量评价体系。

参 考 文 献

[1] Albert R, Barabási A L. Statistical mechanics of complex networks[J]. Rev. Mod. Phys., 2002, 74(1): 47-97

[2] Ta Hung, Yoon Chang, Holm Liisa, et al. Inferring the physical connectivity of complex networks from their functional dynamics [J]. BMC Systems Biology, 2010, 4(1): 70

[3] Emmert-Streib F. A Brief Introduction to Complex Networks and Their Analysis[M]. New York: Structural Analysis of Complex Networks, 2011: 1-26

[4] Kemper A. Complex Networks Theory[M]. Frankfurt: Valuation of Network Effects in Software Markets, 2010: 135-157

[5] Valverde S, Ferrer-Cancho R, Sole R V. Scale-free Networks from Optimal Design[J]. Europhysics Letters, 2002, 60(4): 512-517

[6] Wheeldon R, Counsell S. Power Law Distributions in Class Relationships[C]// Proc Third IEEE International Workshop Source Code Analysis and Manipulation. 2003

[7] Myers C R. Software systems as complex networks: structure, function, and evolvability of software collaboration graphs[J]. Phys. Rev. E, 2003, 68(4): 046116

[8] He Ke-qing, Peng Rong, Liu Jing. Design methodology of networked software evolution growth based on software pattern

[J]. Journal of Systems Science & Complex, 2006, 19(2): 157-181

[9] Cai Kai-yuan, Yin Bei-bei. Software execution processes as an evolving complex network [J]. Information Science, 2009, 179(12): 1903-1928

[10] Crucitti P, Latora V, Marchiori M. Model for cascading failures in complex networks[J]. Phys. Rev. E, 2004, 69(4): 045104

[11] Jacobson V. Congestion control and avoidance[J]. ACM Computer Communications Review (CCR), 1988, 18(4): 314-329

[12] 王健,刘衍桁,刘雪莲. 复杂软件的级联故障建模[J]. 计算机学报,2011,34(6):1137-1147

[13] Hyland-Wood D, Carrington D, Kaplan S. Scale-Free Nature of Java Software Package, Class and Method Collaboration Graphs [C]// The 5th International Symposium on Empirical Software Engineering. 2005

[14] Concas G, Marchesi M, Pinna S, et al. Power-laws in a large object-oriented software system[J]. IEEE Transactions on Software Engineering, 2007, 33(10): 687-707

[15] Zheng Xiao-long, Zeng D, Li Hui-qian. Analyzing open-source software systems as complex network[J]. Physics A, 2008, 387(24): 6190-6200

[16] 马于涛,何克清,李兵,等. 网络化软件的复杂网络特性实证[J]. 软件学报,2011,22(3):381-407

[17] Hou Gang, Wang Xiao-long, Zhou Kuan-jiu. Network Model Construction and Cascading Effect Analysis for Software Systems[C]// 2012 3rd World Congress on Software Engineering (WCSE 2012). Wuhan, China, 2012, 11: 9-12

[18] Kabeko K. Period-doubling of kink-antikink patterns, quasiperiodicity in antiferro-like structures and spatial intermittency in coupled map lattices[J]. Prog. Theor. Phys, 1984, 72(3): 480-486

[19] Dorogovtsev S N, Mendes J F F. Evolution of Networks [J]. Advances in Physics, 2002, 51(4): 1079-1187

[20] Banova T, Mishkovski I, Trajanov D, et al. Organizations Analysis with Complex Network Theory[J]. Communications in Computer and Information Science, 2010, 83(2): 255-265

[21] Yook S H, Jeong H, Barabési A L, et al. Weighted evolving networks[J]. Phys. Rev. Lett, 2001, 86(25): 5835-5838

[22] Holme P, Park S M, Kim B J, et al. Korean university life in a network perspectiveL: dynamics of a large affiliation network [J]. Physica A, 2007, 373(1): 821-830

[23] Lynch S. Neural Networks [M]. Boston: Dyn-amical Systems with Applications using Maple, 2010: 395-426

[24] Chen X G, Zhou J, Zhu Z T. Cascading failure study of urban traffic system based on CML[J]. Mathmatics in practice and theory, 2009, 39(7): 79-84

(上接第 93 页)

个评价指标上均取得较好的结果,进一步证明了可以将用户相似度作为信任用户判断的特征。

但算法对所选取的信任传播模型的依赖性较强,从实验的结果中可以看出,传统的信任传播模型中衰减系数的选取对结果有较大影响,除此之外我们没有对所有的传播路径长度进行对比。如何有效地确定信任在传播过程中的衰减系数及选择合适的传播路径长度可以作为下一步的工作。

参 考 文 献

[1] Kamvar S, Schlosser M, Garcia-Molina H. The Eigentrust algorithm for reputation management in P2P networks[C]// Proceedings of the 12th international conference on World Wide Web. Budapest, Hungary, 2003: 640-651

[2] Guha R, Kumar R, Raghavan P, et al. Propagation of trust and distrust[C]// Proceedings of the 13th international conference on World Wide Web. NY, USA, 2004: 403-412

[3] Golbeck J. Computing and applying trust in Web-based social networks [D]. University of Maryland, 2005

[4] Avesani P, Massa P, Tiella R. Moleskiing. it: a trust-aware recommender system for ski mountaineering [J]. International Journal for Infonomics, 2005

[5] Massa P, Avesani P. Trust Metrics in Recommender Systems [M]// Computing with Social Trust. Springer London, 2009: 259-285

[6] Chen Xiao-cheng, Liu Run-jia, Chang Hui-you. Research of collaborative filtering recommendation algorithm based on trust propagation model [C] // Computer Application and System Modeling (ICCASM). Taiyuan, China, 2010, 4: 177-183

[7] Ziegler C, Lausen G. Analyzing Correlation between Trust and User Similarity in Online Communities [C] // Trust Management, 2004, 2995: 251-265

[8] O'Doherty D, Jouili S, Roy P. Towards trust inference from bipartite social networks[C]// Proceedings of the 2nd ACM SIGMOD Workshop on Databases and Social Networks. Scottsdale, Arizona, 2012: 13-18

[9] Bachi G, Coscia M, Monreale A, et al. Classifying Trust/Distrust Relationships in Online Social Networks[C]// Proceeding of the 2012 ASE/IEEE International Conference on Social Computing and 2012 ASRE/IEEE International Conference on Privacy, Security, Risk and Trust. Chicago, USA, 2012: 552-557

[10] Borgs C, Chayes J, Kalai A, et al. Tennenholtz M. A Novel Approach to Propagating Distrust [J]. Internet and Network Economics, 2010, 6484: 87-105

[11] Victora P, Cornelisa C, Cocka M, et al. Gradual trust and distrust in recommender systems [J]. Fuzzy Sets and Systems, 2009, 160(10): 1367-1382

[12] Richters O, Peixoto T P. Trust transitivity in social networks [J]. PLOS ONE, 2011, 6: 1-14

[13] Shekarpour S, Katebi S. Modeling and evaluation of trust with an extension in semantic web [J]. Web Semantics: Science, Services and Agents on the World Wide Web, 2010, 8(1): 26-36