中国科学: 信息科学 2014 年 第 44 卷 第 10 期: 1323–1344
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS
测试技术专刊 · 论文

一种对程序故障行为和失效行为的聚类有效性验证方法

张丹青[①*], 江建慧[①], 陈林博[②]

① 同济大学软件学院, 上海 201804
② 中国证券登记结算有限责任公司上海分公司, 上海 200120
* 通信作者. E-mail: helen_dandan@hotmail.com

收稿日期: 2014–05–07; 接受日期: 2014–09–13
江苏省未来网络创新研究院未来网络前瞻性研究项目 (批准号: BY2013095-5-06) 和国家高技术研究发展计划 (863) (批准号: 2007AA01Z142) 资助项目

摘要 分析软件自身故障在软件运行过程中的行为规律是分析 "故障 — 错误 — 失效" 链式效应的基础. 但在研究软件故障行为特征时面临的关键问题是故障行为集合的庞大与复杂. 因此, 如何约简故障行为集合是研究故障行为规律的基本前提. "当两个程序具有相似的基本属性时, 其故障行为和失效行为也具有相似性" 这一推断为约简故障集提供了良好的依据, 但尚未得到验证. 本文核心工作即验证该推断的正确性. 为此, 首先提出一种针对程序基准行为、故障行为和失效行为的表征方法; 其次提出一种考虑最佳聚类数的程序行为聚类方法; 最后设计一组基于故障注入的程序行为聚类实验, 用以验证前述推断的正确性. 其中, 实验分别以计算密集型程序 (SPEC CPU2000 和 SPEC CPU2006 基准程序) 和 I/O 密集型程序 (IOZONE、DEBENCH 等) 作为目标程序集. 实验结果表明, (1) 程序行为的表征方法和聚类方法具有合理性和有效性; (2) 以程序基准行为聚类结果为依据的故障行为和失效行为的聚类质量良好, 以此验证了推断的正确性.

关键词 基准行为 故障行为 失效行为 行为特征表达 行为相似度

1 研究背景

为了保证软件系统的可信性 (如可靠性、可用性、安全性等), 仅通过合理设计、质量保证或避错技术等是远远不够的, 还需要采用合理的评测机制来获得. 目前, 软件可靠性的研究主要有两个方面: (1) 将所有软件故障或失效视为等同, 着重研究可靠性评估及预测模型; (2) 以可靠性评估过程中的定性分析或反馈机制为基础, 研究软件故障模型和失效类别. 根据 "故障 — 错误 — 失效" 链的基本原理, 上述两项研究分别归于该链条的两端, 而介于两者之间的研究, 由于缺乏对链上各个环节基本因果关系的分析, 故难以对软件可靠性的评测结果进行反馈. 因此, 必须越过以定量统计分析获取软件故障模型和以定性解析分析评测可靠性之间的 "鸿沟", 着眼于软件产品中故障发生的一般规律, 从而提高软件产品的可度量性和可控性 [1~6].

软件故障发生的一般规律可理解为软件故障空间模型的各属性, 其中包括故障时间分布、空间及位置分布、各类故障之间的数量关系等. 建立软件故障空间模型有助于分析 "故障 — 错误 — 失效"

引用格式: 张丹青, 江建慧, 陈林博. 一种对程序故障行为和失效行为的聚类有效性验证方法. 中国科学: 信息科学, 2014, 44: 1323–1344, doi: 10.1360/N112014-00109

的链式效应, 从而明确软件失效机理. 值得注意的是, 针对该链条中的原因 (故障类型分析) 和结果 (失效类型分析、可靠性评测) 均容易度量, 但仅通过设置跟踪手段对整个链式效应的发生过程进行监控是不够的, 还要使得度量结果或结果中的某子集对该过程是可解释的. 这便需要分析软件自身故障在软件运行过程中的行为规律.

不同软件或同一软件在不同输入条件下的运行时行为特征通常存在差异 [7], 这些差异来源于不同的用户群体、操作剖面和运行环境等. 并且, 对于输入域是无限集合的软件, 其行为特征也是无限的, 且不可预知. 加之软件故障集合本身十分庞大, 在研究软件故障行为特征时面临的关键问题便是故障行为集合的庞大与复杂. 因此, 如何约简故障行为集合是研究故障行为规律的基本前提.

文献 [8] 提出了一种假设: 当两个程序具有相似的基本属性 (如程序结构特性) 时, 其故障行为特征也具有相似性. 且根据文献 [9,10] 的描述, 若将具有相似行为特征的多个程序放在一组, 则该组可采用相近的故障/失效检测方法, 从而提高检测效率. 于是, 可以推测, 程序运行时的行为特征 (如基本块剖面、路径剖面、函数调用序列、数据流特征等) 对程序的故障行为和失效行为起着固有的决定作用. 换言之, 若将具有相似行为特征的程序放在一组, 则该组中的程序同样具有相似的故障行为特征和失效行为特征. 于是, 只需从该组中挑选若干程序进行深入分析, 获得相应的故障/失效行为特征, 便可表征该组中全部程序的故障/失效行为特征. 在这个推测成立的情况下, 在对程序按照其行为特征分类的同时, 故障行为和失效行为可以以 “类” 而非以 “个体” 作为研究单位, 从而实现了约简故障行为和失效行为的目的.

本文提出一种程序行为建模方法, 并设计一组基于故障注入的程序行为聚类实验, 用以验证上述推断的正确性. 核心工作分为三个阶段: (1) 建立程序行为模型, 主要表征程序正常行为、故障行为和失效行为; (2) 根据程序行为特征的相似性, 利用聚类方法将程序分类; (3) 以故障注入方法进行实验验证, 验证内容如下; 程序行为模型所表征的程序行为特征是否准确? “当两个程序具有相似的基本属性时, 其故障行为和失效行为也具有相似性” 这一推断是否正确? 即若以正常行为作为程序分类的依据, 其分类方式是否能够保证程序的故障行为和失效行为在各个类内具有高相似性、而在类间具有高离散性? 为确保实验结果的准确性、完整性和可信性, 实验所需的目标程序集包含了计算密集型基准程序 (SPEC2000 和 SPEC2006 基准程序集) 和 I/O 密集型基准程序 (IOZONE 和 DEBENCH 基准程序等) 两种类型.

2 相关工作

根据研究目标不同, 程序行为的研究方法可分为面向可信性评测的方法、面向软件测试的方法和面向系统/程序优化的方法. 本节针对每种方法适当展开说明.

在面向可信性评测的程序行为分析方法中, 文献 [11] 通过对 Linux 系统运行时的特征参数 (如进程数、内存分配大小、互斥信号等) 进行监控, 分析并识别出可用于表征系统失效的行为特征. 研究人员首先利用故障注入方法对系统各特征参数进行训练, 短时间获得了大量系统失效数据, 其次通过分析这些失效数据识别出能够表征系统失效的系统变量, 然后依据这些变量与失效之间的关系, 利用 F-Measure 方法对变量的重要度排序, 从而找出最能反映系统失效行为特征的系统参数. 文献 [12] 对程序运行剖面进行跟踪和记录, 旨在找出程序操作数和控制流的不变量, 并以这些不变量信息建库, 从而判断程序状态是否出错、是否受到攻击、是否发生失效等. 文献 [13] 提出了基于汇编级的软硬件结合的控制流检测算法, 用以判断由控制流错误引发的程序失效. 该方法比较编译时产生的签名值和运行时的签名值, 两者相同表示控制流正确, 不同则表示发生控制流错误. 上述 3 篇文献分别对应 3 种

典型的面向可信性评测的程序行为研究方法: 选取最优系统参数以表征和预测失效、源代码级的程序剖面分析和目标代码级的程序剖面分析等.

面向软件测试的程序行为分析方法主要分析在编码阶段及编码之后、系统测试和验收测试开始之前的程序行为, 这些阶段属于程序故障诊断初期, 以程序调试方法为主要手段. 程序行为与其语句、函数和路径等特征具有强相关性, 于是文献 [14] 选取 “调用栈” 来表征程序行为, 相同的调用栈意味着程序执行了相同的函数模块, 以此分析程序行为特征. 文献 [15] 以 Java 程序为对象, 以函数间带约束条件的调用关系作为程序状态转换的基本单位建立了扩展的有限状态机, 用以进行软件系统的测试和验证. 文献 [16] 定义了一组 “事件”, 用来表示程序运行时所有可监测的活动行为的抽象, 如语句、表达式、过程调用、消息传递等, 并根据 “事件” 之间的活动规律定义了两类基本关系, 即优先和包含, 随后对程序行为作形式化描述.

面向系统/程序优化的软件行为分析方法主要针对程序运行时与体系结构相关的行为特征建模, 当目标系统为自适应系统时, 还需建立程序行为的预测模型. 文献 [17] 通过分析 OLTP 在线事务处理程序集的行为特征来评测 Itanium 2 处理器的性能, 用以最小化其性能约束并最大化系统吞吐率. 选取表征程序行为特征的指标源于指令级的若干指标, 如 FE、TLB、OTH、NOT 等. 文献 [18] 选取了与文献 [17] 类似的指标刻画程序行为, 包括指令混合比、TLB、D-Cache/I-Cache 行为等, 并提出了一种基于程序流图的行为分析方法. 此外, 文献 [19∼21] 均结合了程序运行时的体系结构特征, 所选工作负载的运行参数也同前述一致.

在上述 3 种方法中, 面向可信性评测的方法和面向软件测试的方法的核心任务是诊断和预测软件的失效行为. 不同之处在于, 前者侧重于监测程序在一段运行周期内可能诱发失效的异常行为特征, 包括错误的系统参数、控制流和数据流等, 以便通过相应的容错或避错机制对即将发生的失效进行处理; 后者则将重点放在程序失效后的回溯过程中, 主要以调试方法来识别失效的具体位置, 以保证程序再次运行时的正确性. 这两种方法的研究对象也存在差别, 前者主要针对大型软件系统 (特别是支撑软件), 后者重在分析小规模软件系统或单个应用程序. 而面向系统/程序优化的方法的核心工作是通过对工作负载运行时与体系结构相关的参数进行收集和行为建模, 并建立工作负载行为预测模型.

若根据研究时采用的数学模型不同, 还可将程序行为分析方法分为形式化方法、统计方法和机器学习方法等. 文献 [22] 在分析程序行为时, 采用不同颜色的标签对程序运行时可能引发失效的代码片段进行分类标识, 其中, 设置标签和识别标签的过程采用形式化方法进行定义和描述. 文献 [23] 研究多环境因素下的程序行为特征, 以三元组 FBM={OBJS,B,E}(各元组分别表示系统的基本组成单元集、故障集和外因集) 表征程序运行时与系统平台的交互特性和约束条件, 并以形式化方法对程序行为进行描述. 文献 [24] 从系统的内在结构因素对软件故障行为模型的影响出发分析软件行为, 用扩展的 Petri 网模型来描述软件故障行为. 此外, 文献 [25,26] 也采用了形式化方法分析程序行为. 文献 [27] 用实验统计的方法验证了程序失效行为聚类方法的 3 个惯用假设条件的合理性, 选取的程序运行剖面包括语句覆盖率、分支覆盖率和数据依赖性等. 另有文献采用机器学习的方法进行程序行为分析, 如文献 [28] 以程序运行时的各种系统资源利用率作为源数据, 提出以决策树方法分析程序行为, 从而将嵌入式程序分类, 为相关编译器的开发和优化提供依据. 文献 [29] 通过分析 Web 服务器运行的工作负载行为的周期性, 采用 HMM 方法自动描述和聚类负载行为. 文献 [18] 采用支持向量机方法对基于程序流图描述的程序行为进行预测分析, 用以实现程序优化.

基于上述分析, 目前, 面向软件测试的方法重在分析程序的正确性问题, 研究切入点是程序失效之后的回溯过程, 即故障定位过程. 而面向可信性评测和面向系统/程序优化的方法所获得的程序行为特征最终都致力于优化程序的运行平台, 如体系结构、操作系统和编译器等, 而并非针对程序本身. 这

两种方法的研究对象一般为源程序或目标程序, 研究过程可分为三个阶段. 第一阶段为数据收集, 根据研究目标不同, 所收集的与程序行为相关的源数据也不同, 但一般都是在体系结构层、操作系统层和上层应用层等三个系统层次上进行数据收集. 第二阶段为数据分析, 即采用特定的原理或方法对所收集的源数据进行处理, 建立程序行为模型. 第三阶段为评估阶段, 主要针对第二阶段中所建立的程序行为模型的准确性、所采用方法或算法的复杂性等做出分析, 并结合特定的研究目标对程序行为模型的有效性做出判定. 本文的研究过程与这两种方法类似, 但从研究对象和核心思想上均有区别. 本文旨在分析程序固有的行为特征, 并结合程序的功能特征、结构特征等, 利用这些固有行为特征对程序的故障行为和失效行为进行分类.

后续内容安排如下: 第 3 章给出程序行为表征方法, 包括程序基准行为、故障行为和失效行为的表征方法; 第 4 章描述了考虑最佳聚类数的程序行为聚类方法; 第 5 章阐述了实验目标、实验设计方案和实验结果分析等. 结论与展望在第 6 章陈述.

3 程序行为表征方法

本文约定, 后续研究遵循以下假设条件:

假设 1: 为不失一般性, 以结构化程序作为研究对象, 即程序包含顺序结构、选择结构、循环结构等.

假设 2: 程序运行时的外部环境无故障.

文献 [30] 认为一个系统是由程序与其运行时的环境因素所构成, 程序的固有缺陷 (故障) 和环境故障通常会对程序行为产生不同的影响, 独立分析这些故障条件下的程序行为有利于更准确地进行故障定位、错误传播分析和失效预测等.

假设 3: 程序的可执行文件以静态链接方式得到.

程序在执行过程中不可避免地会调用库函数, 而库函数大都被视为一种环境因素. 若库函数在程序运行时出错, 则该错误可能会传播至库函数的返回值并最终影响程序自身代码的执行, 或是映射到库函数内部的 I/O 函数中, 从而导致终端设备输出错误信息并最终引发程序失效. 尽管库函数的错误状态可以透过程序本身的某些错误响应行为来间接表征, 但文献 [30] 指出, 程序行为是由其运行周期内在时间上连续但互不重叠的若干执行区间所构成, 因此在程序运行周期内所执行的库函数的相关操作应被视为程序行为的一部分.

此外, 本文的研究对象是 IA32 平台上运行的目标程序.

3.1 程序基准行为

这里, 将程序运行时的正常行为, 即 “Golden run”, 作为程序运行时的 “基准行为”, 用于表征与程序代码中固有特性 (如功能特性、结构特性等) 相关的行为特征.

3.1.1 基准行为的 “状态”

控制流分析是描述程序行为特征时最实用且最有效的方法之一. 较高程序抽象层上的控制流通常反映程序的功能特性, 如各模块之间的依赖关系等; 较低层的控制流则多用于针对软件安全性的行为建模, 如函数调用次序、系统调用序列等; 程序的机器代码是处理器可以直接识别并执行的指令序列, 该层次上的控制流行为可以揭示高级语言抽象层所屏蔽的程序运行时行为特征, 如程序计数器、用于

存放栈帧结构的整数寄存器堆、程序状态字等. 目前基于目标代码级的控制流分析通常用于程序异常行为的在线检测, 并表现出较高的检测能力. 这激发笔者提出一种基于跳转指令的运行时程序状态划分方法 (branch-instruction-based partition, BIP).

按照如下规则对运行时的程序进行状态划分:

Step1 以程序的 main 函数执行的第一条指令作为程序首个状态的首指令;

Step2 若程序执行过程中遇到一条跳转指令, 则将该跳转指令视为当前状态的尾指令;

Step3 将紧跟在 Step2 中跳转指令之后执行的第一条指令作为下一个状态的首指令, 若该指令不是程序执行的最后一条指令, 则转至 Step2, 否则, 程序的状态划分完毕.

BIP 方法与程序基本块划分方法类似, 区别在于, BIP 方法是基于运行时程序进行基本块划分, 每个基本块可反映当前运行时刻的程序特性, 所以可称之为程序状态. 根据跳转指令类型和程序运行时中间数据的变化规律将程序状态划分为如下 6 种类型:

• 无条件跳转状态 (unconditional-jump-state, UJS): 跳转指令类型为无条件跳转. 程序运行时与循环结构 (特别是 while 和 for 结构) 对应的指令序列通常是以 jmp 指令的执行作为开始的标志. 所以, UJS 可以表示程序当前正准备进入某个循环体内.

• 0 条件跳转状态 (conditional-jump-with-false-state, CJFS): 跳转指令类型为条件跳转, 且跳转时读取的跳转条件为 0. 条件跳转是根据条件码寄存器中多个标志位的组合条件确定具体的跳转地址, 而组合条件是由当前程序中特定变量或表达式值所决定的, 能够反映出程序中某些变量对应的算术操作或逻辑操作的属性. 当跳转条件表达式的值为 0 时即表示当前的程序状态类型为 CJFS.

• 1 条件跳转状态 (conditional-jump-with-true-state, CJTS): 跳转指令类型为条件跳转, 且跳转时读取的跳转条件表达式的值为 1.

• 函数调用状态 (CALL-state, CALLS) 和函数返回状态 (RET-state, RETS): 分别表示程序运行时函数的调用和返回操作.

• 中断状态 (INT-state, INTS): 程序运行过程中的中断操作, 如 int $0x80(系统调用) 指令对应的操作等. 从中断处理程序返回时的状态以 RETS 标识.

不难发现, 基于 BIP 方法得到的程序状态序列片段能够表征程序运行时所体现的程序结构上的特性, 如, 形如 UJS,CJFS,. . . ,CJFS,. . . ,CJTS,. . . 的状态序列片段反映出程序正在执行一段具有循环结构特征的程序代码, 形如 CALLS,. . . ,RETS 的状态序列片段表示程序正在完成一次函数调用过程等.

图 1 以一个实例说明根据 BIP 方法得到的程序状态序列能够反映程序结构上的特点. 图 1 描述的是 SPEC2000 基准程序集中 gzip 程序在某输入条件下运行得到的 3 个状态序列片段. 在图 1(a) 中, 存在两个由连续多个 CJFS 状态构成的序列片段, 经过分析得到这些连续执行的 CJFS 状态对应 gzip 代码中的一个循环结构. 图 1(b) 展示了 gzip 运行时的一个更为复杂的循环结构, 即从第 3 个状态到第 63 个状态对应了 gzip 运行时的一次循环过程. 从图 1(c) 可以看出, 存在多个连续出现 2 到 3 次的 CALLS 状态序列片段, 经过分析表明, 这些片段对应了 gzip 执行时的多个函数嵌套调用过程.

一般情况下, 程序功能决定程序结构, 而程序结构决定程序的运行特征. BIP 方法是根据跳转指令进行程序状态划分的, 所以程序状态序列能够反映程序运行时的控制流特征. 此外, CJFS 和 CJTS 这两个状态所对应的跳转条件表达式取决于程序运行时中间变量的值, 这使得状态序列还能反映程序运行时的数据流特征. 所以, 通过表征程序运行时的控制流和数据流特征, 基于 BIP 方法的程序状态序列可用于表征程序运行时的行为特征. 同时, 这些行为特征还能体现程序在结构上和功能上的特性. 因此, 通过程序状态序列可以分析不同程序或同一程序在不同输入条件下的行为特征的相似性或相异性, 从而实现对程序行为的分类.

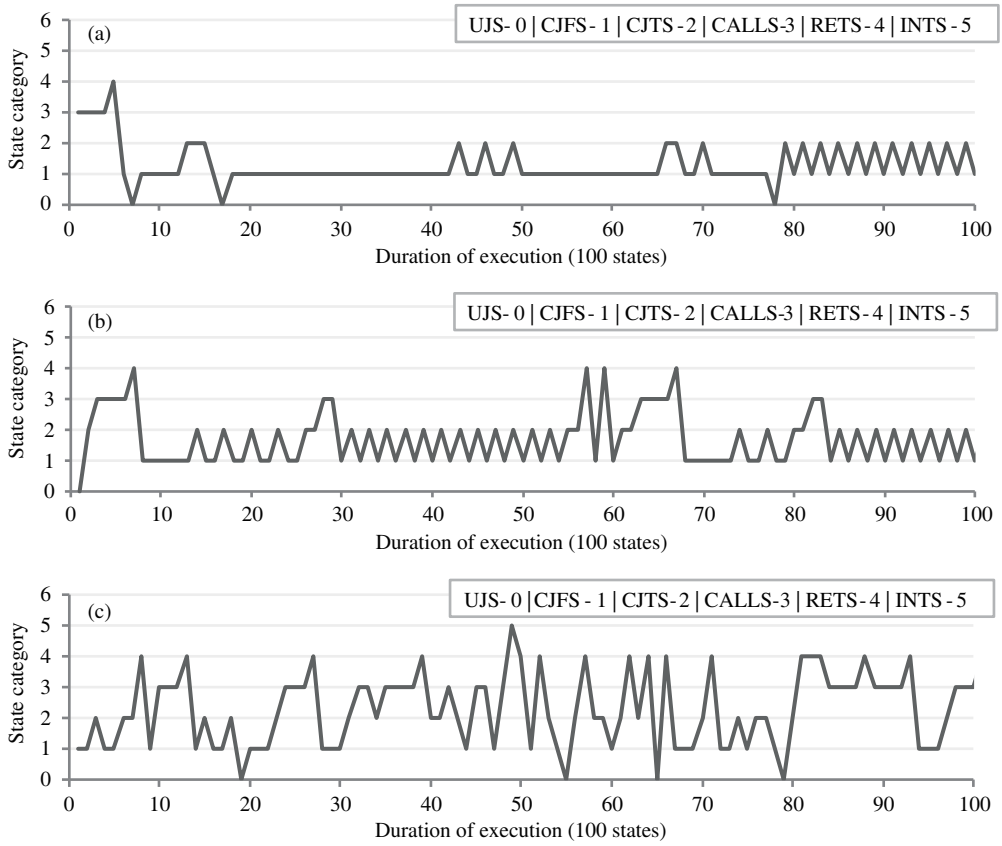

图 1　gzip 程序在某输入条件下运行得到的 3 个状态序列片段

Figure 1　Three fragments of the state sequence obtained during the execution of a gzip program

3.1.2　基准行为表征方法

根据 BIP 方法可将程序在特定外部输入条件下的一次运行行为对应于状态集合 $\sum$ ={UJS, CJFS, CJTS, CALLS, RETS, INTS} 上的一个特定状态序列. 与程序运行时的 “Golden run” 对应的状态序列是稳定的, 可用于表征程序运行时在特定输入条件下的基准行为. 下述定义用于建立程序行为模型:

定义 1　程序的 K– 长状态子序列 (the state sub-sequence of K mer, SSK) 程序状态序列可视为由集合 $\sum$ 中的 6 种状态类型构成的线性序列. 对于程序一次运行得到的状态序列, 其上任意连续的 K 个状态构成的序列称为程序的 K– 长状态子序列.

定义 2　基准行为的特征表达 (baseline behavior spectrum, BBS) 在一个完整的程序状态序列中, 所有 SSK 的频数分布称为程序行为的特征表达. 其中, 每个 SSK 的出现频度称为基准行为的一个特征表达值.

在完整的程序状态序列上取长度为 K 的滑动窗口, 从第一个状态开始做步长为 1 的顺次滑动, 则状态序列可分解为由多个 SSK 拼接而成的多重集合. 理论上讲, 不同 SSK 的个数为 6^K. 有些 SSK 可能因对应于程序的子结构特征而具有特殊含义, 因此, 选取的 K 值越合理, BBS 所对应的基准行为的表征就越准确.

为方便后续分析, 此处对一个基准行为特征表达中的 6^K 个 SSK 的排列顺序如表 1 所示.

若程序状态序列的长度为 L, 则该序列中 SSK 的总数为 $N = L - K+1$. 令 $N_i(1 \leqslant i \leqslant 6^K)$ 表示

表 1 K 个程序状态子序列的固定排序

Table 1 The ordered K-mer combinations

K-mer	The occurrence frequency
⟨ UJS,UJS,...,UJS⟩	b_1
⟨ UJS,UJS,...,CJFS⟩	b_2
⟨ UJS,UJS,...,CJTS⟩	b_3
⟨ UJS,UJS,...,CALLS⟩	b_4
...	...
⟨ INTS,INTS,...,RETS⟩	b_{6^K-1}
⟨ INTS,INTS,...,INTS⟩	b_{6^K}

第 i 个 K– 长状态组合对应的 SSK 在程序状态序列中的出现次数, 则表 1 中与第 i 个 K– 长状态组合对应的 SSK 的频度为 $b_i = N_i/N$. 于是, 该程序状态序列所对应的 BBS 可表示为

$$s^{\mathrm{T}} = \{b_1, b_2, \ldots, b_{6^{K-1}}, b_{6^K}\}. \tag{1}$$

同时, BBS 即用于表征程序在某一特定输入条件下的行为特征.

定义 3 运行时程序行为的特征表达谱 (runtime behavior profile, RBP) 令 $S = \{s_1^{\mathrm{T}}, s_2^{\mathrm{T}}, \ldots, s_n^{\mathrm{T}}\}$ 表示研究所需的目标程序在多个不同输入条件下运行时得到的行为特征表达集合, 集合中的每一个元素代表一次可重复的程序运行过程 (一次 "Golden run"), 其中 $|S| = n$ 表示目标程序在不同输入下的总运行次数. 那么, 由集合 S 中每个程序行为特征表达所组成的特征表达矩阵 $M_{n\times 6^K}$ 称为运行时程序行为的特征表达谱.

$$M = \begin{bmatrix} b_{1,1} & \cdots & b_{1,6^K} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \ldots & b_{n,6^K} \end{bmatrix}, \tag{2}$$

其中, $b_{i,j}$ 表示程序在某一外部输入条件下运行时的基准行为特征表达 (BBS)s_i^{T} 中第 j 个 K– 长状态子序列的特征表达值, 通常情况下有 $n \ll 6^K$.

后续, 在定义 3 的基础上给出程序的故障行为特征表达谱和失效行为表达谱的相关定义.

3.2 程序故障行为

考虑一个简单的程序行为模型, 即程序的下一个状态和目标输出是当前状态和源输入的函数, 如 (3) 式所示

$$(V_{i+1}, O) = H(V_i, I), \tag{3}$$

其中, V_i 和 V_{i+1} 分别表示程序的当前状态和下一个状态, I 和 O 分别为程序在当前状态和下一个状态中的输入向量和输出向量, H 表示程序运行时由当前状态转移至下一个状态并计算得到输出向量的映射函数. 那么, 在故障条件下, 程序行为可能表现出如下 4 种特征:

$$(V_{i+1}, O) = H^*(V_i, I), \tag{4a}$$

$$(V_{i+1}, O^*) = H^*(V_i, I), \tag{4b}$$

$$(V_{i+1}^{*}, O) = H^{*}(V_i, I), \tag{4c}$$

$$(V_{i+1}^{*}, O^{*}) = H^{*}(V_i, I), \tag{4d}$$

其中, $*$ 表示错误的状态转换关系、错误的变量值、错误的输出值等. (4a) 式表示程序中的故障未被激活, 原因主要有故障潜伏期过长、故障所在的程序代码片段不在当前的执行路径中、故障立即被屏蔽等. 根据 (3) 式可知, 若多个程序在正常条件下运行时的输入/输出向量均一致, 则这些程序在某一给定故障条件下运行时的故障行为特征也是一致的.

将程序的当前状态 V_i 和下一个状态 V_{i+1}、输入向量 I 和输出向量 O 映射到基于 BIP 方法划分的程序状态中, 即得到 $(V, I/O) \to \Sigma$. 于是, 程序在故障条件下的行为特征可做如下描述: $(\mathrm{stat}_{i+1}) = H^{*}(\mathrm{stat}_i)$ 用于描述 (4a) 式对应的行为特征; $\left(\mathrm{stat}_{i+1}^{*}\right) = H^{*}(\mathrm{stat}_i)$ 用于描述 (4b), (4c) 和 (4d) 式等对应的行为特征, 其中 $\mathrm{stat}_i \in \Sigma$. 为了说明程序故障行为的表征方法, 首先给出故障表现的定义.

定义 4 故障表现 (fault manifestation, FM) 考虑如下两个由程序在同一输入条件下运行时获得的状态序列:

$$\mathrm{Seq1} = \{\mathrm{stat}_1, \mathrm{stat}_2, \ldots, \mathrm{stat}_i, \ldots, \mathrm{stat}_L\}, \tag{5a}$$

$$\mathrm{Seq2} = \left\{\mathrm{stat}_1^{(f)}, \mathrm{stat}_2^{(f)}, \ldots, \mathrm{stat}_i^{(f)}, \ldots, \mathrm{stat}_{L'}^{(f)}\right\}. \tag{5b}$$

Seq1 和 Seq2 分别表示程序在无故障条件和有故障条件 (假设程序中存在故障 f) 下运行时获得的基于 BIP 方法的状态序列. 若对 $\forall j\,(1 \leqslant j < i)$ 有 $\mathrm{stat}_j = \mathrm{stat}_j^{(f)}$, 且 $\mathrm{stat}_i^{(f)} \neq \mathrm{stat}_i$ 或 $\mathrm{Addr}(\mathrm{stat}_i^{(f)}) \neq \mathrm{Addr}(\mathrm{stat}_i)$($\mathrm{Addr}\,(\mathrm{stat}_i)$ 表示程序状态 stat_i 的首地址), 则称 $\langle \mathrm{stat}_i, \mathrm{stat}_i^{(f)} \rangle$ 为程序在故障 f 下的故障表现.

由此可见, 故障表现实际上是标识程序在故障条件下运行时出现的第一个错误状态. 根据 (5a) 和 (5b) 式可知, 当且仅当对 $\forall i\,(1 \leqslant i \leqslant L')$ 有 $\mathrm{stat}_i^{(f)} = \mathrm{stat}_i$, 且 $L' = L$ 时, Seq2 和 Seq1 是完全相同的两个状态序列, 即程序在故障条件下的行为特征与正常行为特征完全一致. 由于 $\mathrm{stat}_i \in \Sigma$, 则理论上讲, 程序在故障条件 f 下的故障表现 $\langle \mathrm{stat}_i, \mathrm{stat}_i^{(f)} \rangle$ 共有 $6^2 = 36$ 种类型.

图 2 以一个实例说明了定义 4 所描述的故障表现. 图 2(a) 和图 2(b) 分别展示了在同一时刻下, 程序 gzip 在无故障条件下和实施故障注入之后运行时获得的状态序列片段. 在图 2(b) 中, 故障在程序执行时的第 31 个状态中被注入, 随后故障被激活并引发错误, 形成错误传播过程, 其中, 第 41 个状态为首个与图 2(a) 中的正常状态序列不一致的状态, 于是, 图 2(a) 和图 2(b) 中第 41 个状态的类型组合即为故障表现, 即 ⟨CJTS,CJFS⟩.

此外, 另有两种特殊的故障响应行为: 第一种是指程序在故障条件下运行时, 故障潜伏期 (fault latency) 和错误潜伏期 (error latency) 的时长总和为. 换言之, 程序运行时一旦发生故障即导致失效. 第二种是指程序运行时故障未被激活, 也未导致程序出错和失效, 即程序在故障条件下的行为特征与正常行为特征完全一致. 因此, 程序的故障响应行为特征共有 38 种, 包括 36 种故障表现类型和两种特殊的故障响应行为.

由于故障行为通常反映的是程序在某一类故障条件下的行为特征, 所以故障行为是一种具有统计意义的行为特征, 现给出故障行为的表征方法: 给定一个故障集 F, 程序在该故障集下运行时得到的所有响应行为的频数分布为

$$s_{\mathrm{fault}}^{\mathrm{T}} = \{P(fm_1), P(fm_2), \ldots, P(fm_{37}), P(fm_{38})\}, \tag{6}$$

其中 $fm_1, \ldots, fm_{36}$ 表示 36 种故障表现类型 fm_{37} 和 fm_{38} 表示两种特殊的故障响应行为. 将 (6) 式与定义 2 中的 (1) 式相比较可以发现, 两者实质上均为程序行为的概率分布列, 因此, $s_{\mathrm{fault}}^{\mathrm{T}}$ 即为程序

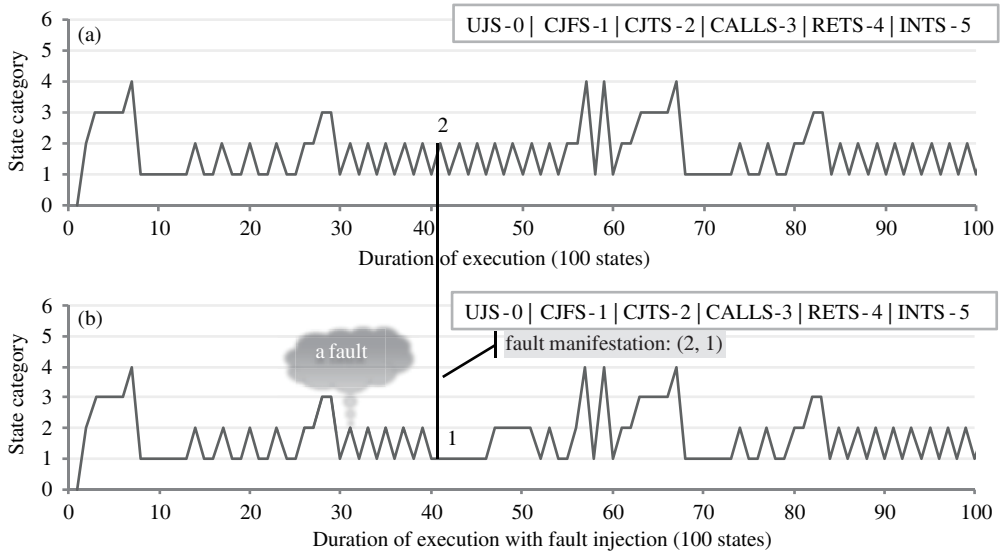

图 2 gzip 程序在无故障条件和故障条件下的状态序列对比, 用以描述故障表现

Figure 2 The comparative runtime behaviors of fragment execution of gzip program: (a) The runtime behavior of normal execution; (b) the runtime behavior with a fault injected during the execution of a specific state

的故障行为特征表达 (fault behavior spectrum, FBS). 而程序的故障行为特征表达谱 (fault behavior profile, FBP)$M_{n\times 38}^{\text{fault}}$ 表示为

$$M^{\text{fault}} = \begin{bmatrix} P_{1,1}(fm_1) & \cdots & P_{1,38}(fm_{38}) \\ \vdots & \ddots & \vdots \\ P_{n,1}(fm_1) & \cdots & P_{n,38}(fm_{38}) \end{bmatrix}. \tag{7}$$

3.3 程序失效行为

对程序失效行为的判定既依赖于目标系统的可观察性, 又与特定的实验集成环境和系统行为监测工具有关. 本文将程序的失效行为分为 3 种失效模式

异常终止 (Abort): 当异常条件 (如非法地址或操作码等) 发生并被检测时, 当前程序终止运行. 这种行为比较容易被监测工具捕获到, 属于较严重的程序失效行为.

响应超时 (Hang): 程序未能在规定的时间周期内提交数据, 以致影响程序执行时间或引发控制流错误, 该失效行为通常使得程序进入无限循环状态或忙等待状态, 较易被监测到, 但需进行人为处理, 如通过重启方式恢复程序运行.

非法输出 (Wrong): 程序运行完毕后的输出值与 Golden run 的运行结果不一致, 该行为无法通过控制流监测到, 可采用冗余技术或边界检查技术监测该失效行为.

和故障行为一样, 失效行为反映了程序在某一类故障条件下的响应行为特征, 所以, 失效行为也是一种具有统计意义的行为特征. 将程序失效行为的表征方法描述如下: 给定一个故障集 F, 程序在该故障集条件下运行得到的失效响应行为的频数分布为

$$s_{\text{fail}}^{\text{T}} = \{P(\text{Abort}), P(\text{Hang}), P(\text{Wrong}), P(\text{Correct})\}, \tag{8}$$

其中, P(Correct) 表示程序在故障条件下未发生失效的概率. (8) 式同样是一个概率分布列, 与定义 2 中的 (1) 式本质上一致, 因此, $s^{\mathrm{T}}_{\mathrm{fail}}$ 即为程序的失效行为特征表达 (failure behavior spectrum, FLBS). 而程序的失效行为特征表达谱 (failure behavior profile, FLBP)$M^{\mathrm{fail}}_{n\times 38}$ 表示为

$$M^{\mathrm{fail}}=\begin{bmatrix} P_{1,1}\,(\mathrm{Abort}) & \cdots & P_{1,4}\,(\mathrm{Correct}) \\ \vdots & \ddots & \vdots \\ P_{n,1}\,(\mathrm{Abort}) & \cdots & P_{n,4}\,(\mathrm{Correct}) \end{bmatrix}. \tag{9}$$

4 程序行为聚类方法

理论上讲, 若不对目标程序的类型和数据做出限制, 其行为特征集合将是一个无限集合. 为了检测、分析和提炼程序的行为特征, 必须先对这些行为特征进行分类, 归并在同一类中的程序行为具有某种相似性或相关性, 由此可以深入分析一类程序行为的其他特性, 如故障行为特征、失效行为特性和可靠性相关特性等.

为了得到合理的程序行为聚类结果, 将程序的聚类过程分为 3 个步骤: (1) 给出程序行为相似性的度量方法; (2) 给出程序行为聚类方法; (3) 通过优化聚类过程确定最佳聚类数.

4.1 行为相似性度量

根据上述对程序正常行为、故障行为、和失效行为特征表达的定义可知, (1) 式, (6) 式和 (8) 式对应的行为表征方法本质上均为全概率事件的概率分布. 由 Kullback 和 Leibler 提出的相对熵 [31] 能够反映两个概率分布之间的接近程度, 是一个精确且敏感的相似性测度. 但它不具有对称性, 故不能直接用于度量两个概率分布之间的相似度. 于是, 采用一种修正的相对熵方法 (FDOD 方法 [32]) 作为程序行为相似性度量方法.

定义 5 程序行为相似度 (similarity degree SD) 给定两个程序行为 $s^{\mathrm{T}}_i=\{a_{i,1},a_{i,2},\ldots,a_{i,w}\}$ 和 $s^{\mathrm{T}}_j=\{a_{j,1},a_{j,2},\ldots,a_{j,w}\}$, 其中, $1\leqslant i,j\leqslant n$($n$ 为目标行为总数), 当度量对象为基准行为时, $1\leqslant w\leqslant 38$; 当度量对象为故障行为时, $1\leqslant w\leqslant 38$; 当度量对象为失效行为时, $1\leqslant w\leqslant 4$. 则两个程序行为之间的相似度定义为

$$\mathrm{SD}\left(s^{\mathrm{T}}_i,s^{\mathrm{T}}_j\right)=\sum_{l=1}^{w}a_{i,l}\log\frac{a_{i,l}}{(a_{i,l}+a_{j,l})/2}+\sum_{l=1}^{w}a_{j,l}\log\frac{a_{j,l}}{(a_{i,l}+a_{j,l})/2}, \tag{10}$$

其中, 约定 $0\log\frac{0}{(a_{i,l}+a_{j,l})/2}=0$ 和 $0\log\frac{0}{0}=0$.

由定义 5 可知, $\mathrm{SD}(s^{\mathrm{T}}_i,s^{\mathrm{T}}_j)\geqslant 0$ 恒成立且 $\mathrm{SD}(s^{\mathrm{T}}_i,s^{\mathrm{T}}_j)$ 越小表示两个程序行为的特征表达越相似, 反之则差异越大. 当且仅当对 $\forall i,j\,(1\leqslant i,j\leqslant n)$ 有 $a_{il}=a_{jl}(l=1,2,\ldots,w)$ 时, $\mathrm{SD}\left(s^{\mathrm{T}}_i,s^{\mathrm{T}}_j\right)=0$ 这表示两个程序行为的特征表达完全相同.

定义 6 程序行为相似矩阵 (similarity degree matrix, SDM) 对给定的目标程序行为集合 $S(|S|=n)$, 集合中两两程序行为之间的相似性构成的一个 $n\times n$ 阶矩阵, 该矩阵称为程序行为相似矩阵

$$\mathrm{SDM}=\begin{bmatrix} \mathrm{SD}\left(s^{\mathrm{T}}_1,s^{\mathrm{T}}_1\right) & \cdots & \mathrm{SD}\left(s^{\mathrm{T}}_1,s^{\mathrm{T}}_n\right) \\ \vdots & \ddots & \vdots \\ \mathrm{SD}\left(s^{\mathrm{T}}_n,s^{\mathrm{T}}_1\right) & \cdots & \mathrm{SD}\left(s^{\mathrm{T}}_n,s^{\mathrm{T}}_n\right) \end{bmatrix}. \tag{11}$$

4.2 程序行为聚类

程序行为聚类是以程序行为特征表达谱 (包括基准程序行为特征表达谱 BBP、故障行为特征表达谱 FBP 和失效行为特征谱 FLBP 等) 为源数据, 对程序的基准行为、故障行为和失效行为进行特征分类的过程.

首先, 基于定义 5 和 6 得到与 BBP、FBP 和 FLBP 等程序行为特征表达谱对应的程序行为相似矩阵 $\mathrm{SDM}_{\mathrm{baseline}}$, $\mathrm{SDM}_{\mathrm{fault}}$ 和 $\mathrm{SDM}_{\mathrm{fail}}$, 并采用多维标度法 (multidimensional scaling, MDS) 为这些相似矩阵确定每个目标程序行为在二维空间中的标识.

然后, 采用聚类算法对目标程序行为进行分类. 选取 (fuzzy compactness and separation, FCS) 算法 [33] 进行程序行为聚类. FCS 算法属于已知聚类数 (cluster number, CN) 情况下选择最佳数据划分的方法, 该算法同时考虑了类与类之间的离散程度以及同一类内的紧凑程度, 这保证了聚类方法的有效性.

令 $Y=\{y_1,y_2,\ldots,y_n\}$ 为由 MDS 方法得到的程序行为特征表达在二维空间中的标识, n 为目标程序行为总数. 令聚类数为 c, 隶属函数为 $\mu=\{\mu_{ij}\}$, 满足 $\sum_{i=1}^{c}\mu_{ij}=1,\mu_{ij}\in[0,1]$. 令 $\boldsymbol{y}=\sum_{j=1}^{n}y_j/n$ 为所有目标程序行为的特征表达在二维空间中的均值. 又令 $V=\{v_1,v_2,\ldots,v_c\}$ 为每个类的聚类中心. 则 FCS 算法的目标函数表示如下:

$$J_{\mathrm{FCS}}=\sum_{i=1}^{c}\sum_{j=1}^{n}\mu_{ij}^{m}||y_j-v_i||^2-\sum_{i=1}^{c}\sum_{j=1}^{n}\eta_i\mu_{ij}^{m}||v_i-\boldsymbol{y}||^2, \tag{12}$$

其中, $\eta_i\geqslant 0$, 反映类间距离的权重. 根据最小化原理和 Lagrange 乘子法定义的 Lagrange 函数如下:

$$L_{\mathrm{FCS}}=\sum_{i=1}^{c}\sum_{j=1}^{n}\mu_{ij}^{m}||y_j-v_i||^2-\sum_{i=1}^{c}\sum_{j=1}^{n}\eta_i\mu_{ij}^{m}||v_i-\boldsymbol{y}||^2-\sum_{j=1}^{n}\lambda_j(\sum_{i=1}^{c}\mu_{ij}-1), \tag{13}$$

分别求 L_{FCS} 对 μ_{ij} 和 v_i 的偏导数, 并令其偏导数为零, 可得到最小化目标函数 L_{FCS} 的必要条件. 于是, 隶属度 μ_{ij} 和聚类中心 v_i 的迭代公式分别为

$$\mu_ij=\frac{(||y_j-v_i||^2-\eta_i||v_i-\boldsymbol{y}||^2)^{\frac{-1}{m-1}}}{\sum_{k=1}^{c}(||y_j-v_k||^2-\eta_k||v_k-\boldsymbol{y}||^2)^{\frac{-1}{m-1}}}, \tag{14}$$

$$v_i=\frac{\sum_{j=1}^{n}\mu_{ij}^{m}y_j-\eta_i\sum_{j=1}^{n}\mu_{ij}^{m}\boldsymbol{y}}{\sum_{j=1}^{n}\mu_{ij}^{m}-\eta_i\sum_{j=1}^{n}\mu_{ij}^{m}}. \tag{15}$$

另外, 类间距离权重 η_i 的更新公式 [33] 为

$$\eta_i=\frac{(\beta/4)\min_{i\neq i'||v_i-v_{i'}||^2}}{\max_k||v_k-\boldsymbol{y}||^2},\ 0\leqslant\beta\leqslant 1.0. \tag{16}$$

对某些 y_j 来说, 由 (14) 式确定的隶属度 μ_{ij} 可能出现负值, 根据文献 [33] 的规定, 若 $||y_j-v_i||^2\leqslant\eta_i||v_i-\boldsymbol{y}||^2$ 则 $\mu_{ij}=1$, 且 $\mu_{i'j}=0,\ i\neq i'$.

FCS 算法描述如下 [33]:

Step 1 初始化: 迭代次数为 $l=0$, 给定目标程序行为集合的聚类数 $c(2\leqslant c\leqslant n)$ 和初始聚类中心 $v_i^{(0)}(1\leqslant i\leqslant c)$, 给定 β,m 以及阈值 $\varepsilon(\varepsilon>0)$;

Step 2 以 (16) 式更新 $\eta_i^{(l+1)}$;

Step 3 以 (14) 式更新 $\mu_{ij}^{(l+1)}$;

Step 4 以 (15) 式更新 $v_i^{(l+1)}$;

Step 5 若 $\max_l||v_i^{(l+1)}-v_i^{(l)}||^2<\varepsilon$, 则停止; 否则转至 Step 2.

本文实验中, 参数 m 设为 2, 参数 β 设为 0.2, 阈值 ε 设为 10^{-6}.

4.3 确定最佳聚类数

文献 [34] 指出, 目标函数与聚类数之间存在如下关系: (1) 目标函数是聚类数的单调递减函数, 这说明聚类数越多, 数据集的划分就越精细, 其极限情况为每个数据作为一个类, 但这并不表示聚类结果就越有效、越合理; (2) 随着聚类数的增大, 相应的目标函数值会在某一时刻出现骤降, 且当达到一定的聚类数时, 目标函数值的变化将趋于平稳. 基于这种关系, 提出一种基于目标函数变化率与一阶差分相结合的方法, 用于确定程序行为特征的最佳聚类数 (optimal cluster number, OCN).

定义 7 目标函数 J_{FCS} 的变化率 (hazard rate of J_{FCS}) 给定预设的聚类数范围为 $[c_{\min},c_{\max}]$, 对预设范围内的每个聚类数根据 FCS 算法得到相应的最优目标函数, 将 $J(c_i)(c_{\min}\leqslant c_i\leqslant c_{\max})$ 记作目标函数 J_{FCS} 关于聚类数 c 的函数. 定义目标函数 J_{FCS} 的变化率为在特定聚类数的瞬时变化率 $h(c_i)$, 即

$$h\left(c_i\right)=\frac{p_{c_{i+1}}-p_{c_i}}{(c_{i+1}-c_i)(1-p_{c_i})},\tag{17}$$

其中, $p_{c_i}=\frac{J(c_i)}{\sum_{c_{\min}c_i c_{\max}}J(c_i)}\,(c_{\min}\leqslant c_i\leqslant c_{\max})$ 且 $h\left(c_{\max}\right)=h\left(c_{\max-1}\right)$. 另约定, 若 $\exists c_i(c_i\in[c_{\min},c_{\max}])$ 有 $J(c_i)<0$ 则对所有 c_i, 在计算 $h\left(c_i\right)$ 之前, 令 $J\left(c_i\right)=J\left(c_i\right)+\left|\min_{c_{\min}\leqslant c_i\leqslant c_{\max}}\{J\left(c_i\right)<0\}\right|$.

确定 OCN 的方法描述如下:

Step 1 预设聚类数范围为 $[c_{\min},c_{\max}]$, 令 $c_{\min}=2$(当 $c_{\min}=1$ 时, 表示数据集均匀分布, 数据间无明显差异), $c_{\max}=\sqrt{n}$[35], 其中 n 为数据集 (即程序行为特征集) 总量.

Step 2 根据 FCS 算法对 $[c_{\min},c_{\max}]$ 范围内的每个聚类数计算得到相应的最佳目标函数值 $J(c_i)$, 并形成一组由目标函数值构成的离散序列 $\{J(c_i)\}$.

Step 3 根据 (17) 式计算得到与每个目标函数 $J(c_i)$ 的变化率 $h\left(c_i\right)$, 并形成一组由目标函数 $J(c_i)$ 的变化率构成的离散序列 $\{h\left(c_i\right)\}$.

Step 4 计算离散序列 $\{h\left(c_i\right)\}$ 的向后一阶差分, 即 $\nabla h_{c_i}=h\left(c_i\right)-h(c_{i-1})(c_{\min}\leqslant c_i\leqslant c_{\max}$, $\nabla h_{c_1}=0)$.

Step 5 选取离散序列 $\{h\left(c_i\right)\}$ 一阶差分的极大值, 作为最佳聚类数 (OCN).

5 实验

5.1 实验目的及过程

以 IA32 体系结构和 Linux 2.6.32 操作系统作为实验平台, 以计算密集型 (CPU-bound) 程序和 I/O 密集型 (I/O-bound) 程序构成实验所需的目标程序集. 其中, 以 SPEC2000 和 SPEC2006 基准程序集作为计算密集型程序集, 以 IOZONE 基准程序、DBENCH 基准程序和 (linux test project, LTP) 测试集中与 I/O 相关的测试程序等作为 I/O 密集型程序集. 实验过程中, 均采用程序默认的输入集作为程序的外部输入条件, 而程序在某一输入条件下的正常行为被视为该程序在该输入条件下的一个基准行为. 由此生成 99 个计算密集型程序的基准行为和 54 个 I/O 密集型程序的基准行为.

表 2 实验所选故障类型列表

Table 2 Description of fault types selected in our experiments

Fault type	Description
MFC	Missing function call
MIA	Missing if construct around statements
MIFS	Missing if construct plus statements
MLAC	Missing “AND EXPR” in expression used as branch condition
MVAE	Missing variable assignment using an expression
MVAV	Missing variable assignment using a value
WPFV	Wrong variable used in parameter of function call
MVIV	Missing variable initialization using a value

在实验过程中, 为了得到基于 BIP 方法的程序状态, 利用 PIN[36] 对程序进行指令级动态插装, 通过判断当前跳转指令的类型确定程序运行时的状态类型.

此外, 实验中采用故障注入方法获得程序的故障行为和失效行为, 用以进行行为聚类分析. 选取基于 ODC 方法 [37] 分类的软件故障, 所选故障类型如表 2 所示. 为了得到具有统计意义的实验结果, 在故障注入实验中, 针对所有程序的执行过程实施了共超过 90000 次故障注入, 从而得到程序的故障行为特征表达谱 (FBP) 和失效行为特征表达谱 (FLBP).

实验针对上述提出的程序行为建模和聚类方法进行验证, 对计算密集型程序和 I/O 密集型程序分别从行为聚类的合理性、有效性和聚类质量等 3 个方面衡量实验结果, 其目的是为了验证 “具有相似行为特征的程序同样具有相似的故障和失效行为特征” 这一推断的正确性. 其中, 聚类的合理性是指针对程序基准行为的聚类能够反映程序在其固有特征, 如功能特性、结构特性、运行时的数据流和控制流特性等上的区分. 聚类的有效性是针对聚类方法而言的, 主要验证方法本身的合理性. 聚类质量则是用于评价上述推断正确性的关键指标.

5.1.1 聚类的合理性和有效性

聚类的合理性用于衡量程序基准行为分类的准确性, 而聚类的有效性用于评价聚类方法本身的准确性. 从两方面考虑这两个指标: 一是验证程序基准行为表征方法的准确性, 即基准行为的特征表达是否能够作为程序正常运行时对其固有特性的体现; 二是验证对这些基准行为的分类结果的准确性, 即同一类别中的基准行为是否具有相似性.

图 3 展示了验证聚类合理性和有效性的实验框架. 图 3(a) 展示了获取程序基准行为的特征表达谱 (BBP) 的过程. 在不考虑体系结构、操作系统等环境因素的影响下, 程序的输入剖面决定了程序运行时的执行路径和数据依赖关系, 为获得 BBP, 将每个程序在特定输入条件下的一次正常运行视为一个独立的基准行为. 尽管程序本身已具备明显的功能特性和结构特性, 且在一定程度上, 在输入剖面下的程序基准行为能够呈现这种功能和结构上的统计规律性, 但对于某些外部输入, 程序基准行为可能表现出对这种规律性的偏离. 因此, 将外部输入与程序基准行为一一对应有利于分析程序的真实行为特征, 同时, 在此基础上, 若能够通过回溯机制得到程序的实际外部输入分布, 可帮助程序可靠性评估设计合理的操作剖面.

在图 3(a) 中, 由 99 个计算密集型程序基准行为的特征表达和 54 个 I/O 密集型程序基准行为的

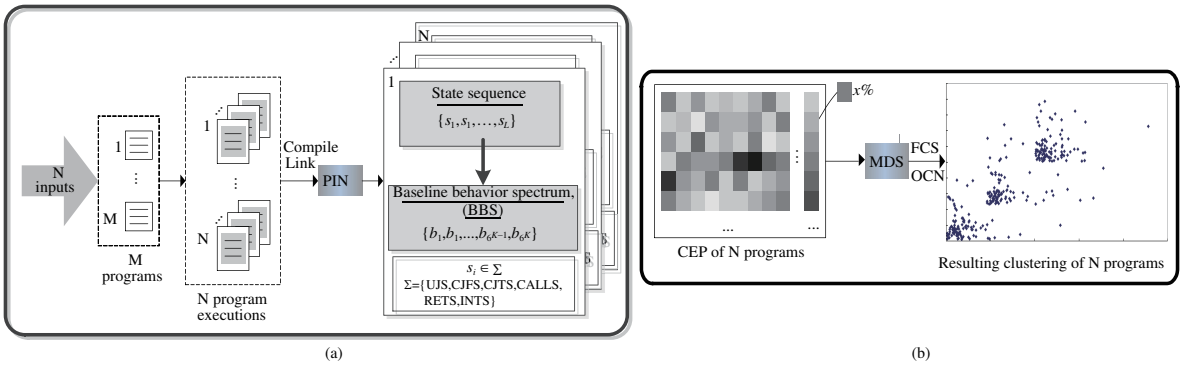

图 3　验证聚类合理性和有效性的实验框架

Figure 3　The experimental implementation framework for verifying the validity and effectiveness of clustering

特征表达分别构成了以 $M_{99\times 6^K}$ 和 $M_{54\times 6^K}$ 表示的基准行为特征表达谱 BBP.

图 3(b) 展示了根据 BBP 进行行为聚类的过程, 即首先通过 MDS 方法得到每个基准行为特征表达在二维空间中的标识, 然后根据 FCS 算法对基准行为聚类, 最后利用 4.3 小节的方法优化聚类结果, 确定最佳聚类数.

5.1.2　聚类质量

为了验证对程序基准行为的分类结果可否用于对程序故障行为和失效行为的分类, 以聚类质量来评价验证结果. 聚类质量用于表征以程序基准行为的聚类结果用于表征程序故障行为和失效行为的聚类结果的准确程度, 即如果基准行为的聚类结果与故障行为或失效行为本身的聚类结果越接近, 则说明聚类质量越高, 反之则越低. 由此可知, 聚类质量可用于判定 “具有相似行为特征的程序同样具有相似的故障和失效行为特征” 这一推断的正确性.

评价聚类质量的过程是: 首先根据 99(或 54) 个程序基准行为的特征表达对其进行聚类, 其次通过故障注入实验得到这 99(或 54) 个程序执行过程的故障行为和失效行为的特征表达, 然后将这些故障行为和失效行为依照程序基准行为的聚类方式进行分类, 最后, 采用两个指标来描述聚类质量: 类内平均相似度和聚合因子. 类内平均相似度是指, 以基准行为聚类方式区分故障行为和失效行为时, 计算每个类内两两故障行为 (或失效行为) 之间的相似度 (参见定义 5), 所有类内的行为相似度的平均值即为类内平均相似度. 聚合因子 (in-group proportion, IGP)[38] 通过衡量当程序故障行为和失效行为根据基准行为的聚类方法进行分类时, 类内的每个故障行为 (或失效行为) 样本距离最近的样本是否在同一类中, 来判定聚类质量. 聚合因子的计算方法如下:

$$\mathrm{IGP}=\sum_{i=1}^{c}\frac{\mathrm{igp}(i)}{c}, \tag{18}$$

其中 $\mathrm{igp}(i)=\frac{\#\{j|\mathrm{Cluster}(j)=\mathrm{Cluster}(j^N)=i\}}{\#\{j|\mathrm{Cluster}(j)=i\}}$, $c(c>1)$ 为聚类数 (聚合因子不适用于聚类数为 1 的情形), j 为某个程序行为样本, j^N 为距离样本 j 最近的样本, $\mathrm{Cluster}(j)$ 表示样本 j 的所属类, 与 i 一样均为类标, #表示满足中括号内条件的样本个数. 可以看出, 聚合因子 IGP 越大表示聚类质量越高, 当且仅当 IGP=1 时达到理论上的绝对最优聚类.

5.2 实验结果及分析

对一系列实验结果做出如下分析; (1) 聚类的合理性和有效性, 即程序的基准行为表征方法是否正确? 对其聚类结果是否准确? (2) 评价聚类质量, 即程序的故障行为和失效行为是否可以基准行为的聚类方式进行区分? (3) 探讨程序行为聚类在分析可靠性时的作用, 即对程序基准行为的聚类是否也可用于对程序可靠性指标的区分? (4) 针对程序基准行为表征方法及聚类方法中的两个关键参数进行分析, 即程序状态的子序列长度 K 和聚类数 (cluster number, CN), 并得到最优的参数值. 其中, 针对计算密集型程序集和 I/O 密集型程序集所考虑的 K 值/CN 值范围分别为: $2 \leqslant K \leqslant 9/2 \leqslant \mathrm{CN} \leqslant 10$ 和 $2 \leqslant K \leqslant 9/2 \leqslant \mathrm{CN} \leqslant 8$.

5.2.1 聚类的合理性和有效性

通过一个实例分析说明聚类的合理性和有效性. 选取的实例为当 $K = 5, \mathrm{CN} = 7$ 时的计算密集型程序基准行为聚类情况. 从 K 的取值范围 ($2 \leqslant K \leqslant 9$) 来看 $K = 5$ 是一个中间值, 以此作为程序状态子序列的长度可以得到一个相对均衡的基准行为特征表达. 在 $K = 5$ 的条件下将程序的基准行为进行聚类, 可得到如补充材料图 S4(a) 所示的当 $2 \leqslant \mathrm{CN} \leqslant 10$ 时聚类目标函数变化率的一阶差分. 从补充材料图 S4(a) 可见 CN=3,5,7 时均为一阶差分的极大值, 且当聚类数增大时, 类内样本的相似性和类间样本的相异性随之增加, 因此, 为突出说明聚类的合理性和有效性, 选取 CN=7 作为该实例分析中的最佳聚类数 (OCN). 此外, 为了简化分析, 将程序状态集 Σ 上的 6 种状态类型分别赋值: UJS=0, CJFS=1, CJTS=2, CALLS=3, RETS=4, INTS=5.

补充材料图 S4(b) 展示了当 $K = 5$ 和 $\mathrm{CN} = 7$ 时 99 个程序执行过程的基准行为特征表达经过聚类后的情形. 图中每个类以 Cluster #标记, 类中的数据为类内所有基准行为的特征表达, 即横轴表示 $6^5 = 7776$ 个 5- 长状态子序列 (参见定义 1), 纵轴表示每个 5- 长状态子序列的发生频度. 从补充材料图 S4(b) 中可以看出, 不同类中的基准行为特征表达之间存在差异. 通过分析, 每个类内的基准行为能够反映相应程序在其功能和结构等上的固有特性. 以下给出详细分析, 以阐述 3.1 小节描述的程序行为表征方法的合理性.

如补充材料图 S4(b) 所示, 在 Cluster 1 中执行最频繁的状态子序列是 11211,12112 和 21121, 这些子序列反映的是程序运行时 CJFS 状态连续出现, 或 CJFS 状态与 CJTS 状态交替出现. 由于状态子序列是由目标代码得到的, 通过分析与这些状态子序列对应的程序源码部分可知, 首地址不变的 CJFS 状态连续出现意味着程序正在执行一个循环体. 状态子序列中单一的 CJFS 状态 (或 CJTS 状态) 对应程序执行时的一个 if/if-else 结构. 因此, 通过 Cluster 1 展示的 3 个最频繁执行的状态子序列, 可以看出该类中的程序基准行为具有频繁地执行循环结构和 if/if-else 结构的特点. 不仅如此, 11211 这个状态子序列还体现了程序执行过程中循环结构与选择结构的交替执行. 这表明程序存在嵌套循环, 且 if 结构存在于某个嵌套层中. 通过阅读和分析 Cluster 1 中的程序代码 —— 包含 5 个 vortex 程序执行过程 (3 个行为来自输入集 ref, 2 个行为分别来自输入集 train 和 test)—— 确实发现代码中大量存在的诸如 for{...if{...}...}, for{...if{...for{...}...}...}等结构. 所以, 11211,12112 和 21121 这 3 个最频繁执行的状态子序列能够反映程序运行时所体现的程序结构上的特征. 此外, 13333, 23333, 33332 等若干子序列为 Cluster 1 中次频繁出现的状态子序列. 这表明程序运行时执行了许多函数嵌套调用操作, 如前述分析, 这些行为与程序固有的结构特性吻合.

在 Cluster 3 中, 11211, 12112, 21121 和 11111 是出现最频繁的状态子序列. 其中, 从前 3 个状态子序列可以看出, Cluster 3 中的程序基准行为有着与 Cluster 1 中程序相似的行为特征. 而子序列

11111 则反映出程序频繁执行某一个或某些循环体. 通过阅读和分析程序源码, 在 Cluster 3 中的 26 个程序执行过程 —— 如 crafty(3 个行为分别来自输入集 ref, train 和 test)、gzip(4 个行为来自输入集 ref, 2 个行为分别来自输入集 train 和 test)、mcf(3 个行为分别来自 SPEC2000 的输入集 ref, train 和 test)、sjeng(2 个行为分别来自 ref 和 test) 等 —— 确实明显具有上述 4 个状态子序列所反映的特征. 此外, 在 Cluster 6 中, 状态子序列 11111 的执行频度超过 0.8, 这表明类内程序执行了大量的循环操作, 如 libquantum(3 个行为分别来自输入集 ref, train 和 test)、h264ref(3 个行为分别来自输入集 ref, train 和 test) 等.

在补充材料图 S4(b) 中, Cluster 2 的程序基准行为特征表达中出现最频繁的状态子序列是 22222, 其次是 11111, 13311 和 33111. 如前述分析一致, gcc(3 个行为来自 SPEC2000 的输入集 ref, train 和 test)、parser(2 个行为分别来自输入集 ref 和 train)、xalancbmk(3 个行为分别来自输入集 ref, train 和 test) 等程序行为中呈现了大量与循环结构和函数嵌套调用结构相关的特性.

在 Cluster 5 和 Cluster 7 中, 状态子序列 11111 均为最频繁出现的子序列之一. 而 Cluster 5 中另一个频繁执行的子序列为 22222, 同时, Cluster7 中其余频繁出现的子序列为 21212 和 12121. 这说明 Cluster 5 和 Cluster 7 中的程序均执行了大量的循环操作. 此外, Cluster 7 中的程序还执行了大量的与选择结构相关的操作. 通过对程序源码的分析, Cluster 5 中的 13 个程序行为 (astar<2 个行为来自输入集 ref, 2 个行为来自输入集 train, 1 个行为来自输入集 test>, bzip2<5 个行为来自 SPEC2006 的输入集 ref, 2 个行为分别来自 SPEC2006 的输入集 train 和 test> 等) 和 Cluster 7 中的 27 个程序行为 (bzip2<3 个行为分别来自 SPEC2000 的输入集 ref, train 和 test>, gap<3 个行为分别来自输入集 ref, train 和 test>, gcc<2 个行为来自 SPEC2006 的输入集 ref, 2 个行为分别来自 SPEC2006 的输入集 train 和 test> 等) 均能体现这些程序在结构上的特性.

除了上述分析, 根据 $K = 5$ 和 CN= 7 这个实例, 还能发现如下几个问题:

(1) 在所有程序的基准行为特征表达中, INTS 状态很少出现, 这说明程序运行时发生软中断的次数很少 (这一点与计算密集型程序特性相符). 据实验统计表明, 在所有程序行为中, INTS 出现的频度仅为 0.016%.

(2) 补充材料图 S4(b) 还展示了一种例外情况, 即 Cluster 4 中的程序行为并不具有明显的特性 (虽然在 Cluster 4 仅包含 10 个程序行为), 50%以上的状态子序列的发生频度较为平均、没有明显差别, 而发生频度最高的状态子序列 (为 33120) 仅有 0.041 的发生频度. 通过分析 Cluster 1 至 Cluster 7 全部程序基准行为在各个类内的平均相似度, 得到平均相似度的直观值并展示在补充材料图 S5 中. 如补充材料图 S5 可知, Cluster 4 中程序行为之间的平均相似度最低, 即两两行为之间存在的差异较大. 经过分析和推测, 产生这种现象的原因可能在于聚类准则. 聚类算法根据样本之间的距离优先聚合距离较近的样本, 最后将余下的样本放入同一类中. 因此, 确定最佳聚类数是十分重要的.

(3) 通过 $K = 5$ 和 CN= 7 实例的聚类结果发现, 同一程序在不同输入下的基准行为并非都在同一类中, 有些程序在不同输入下的基准行为差异较大, 从而被划分在不同类中. 这表明, 以程序执行过程替代程序本身作为聚类样本更能说明行为表征的合理性和聚类的有效性.

通过上述对 $K = 5$ 和 CN= 7 的实例分析可知, 3.1 小节描述的程序基准行为表征方法不仅能够反映程序运行时的行为特征, 还能使这些基准行为反映出与程序固有特性 (功能特性、结构特性等) 相关的行为特征. 此外, 根据上述实例中的聚类结果可以看出, 在确定了最佳聚类数的情况下, 通过 FCS 算法得到的聚类结果可将具有相似行为特征的程序归类, 使得类内程序行为的紧致性较强.

表 3 与每个 K 值对应的最佳聚类数

Table 3 The optimal cluster number for all Ks

K		2	3	4	5	6	7	8	9
OCN	CPU-bound	3,6,9	3,7,9	3,8	3,5,7	3,5,9	3,7,9	3,6	3,6
	I/O-bound	3	3,7	3	3,6	3,7	3,7	3	3

5.2.2 聚类质量

根据实验结果, 首先针对程序基准行为的聚类结果确定最佳聚类数, 然后对聚类质量作出评价各个 K 值 ($2 \leqslant K \leqslant 9$) 下由 FCS 算法目标函数的变化率构成的离散序列所对应的一阶差分参见补充材料图 S1, 图中用小圆圈标注的点即为候选的最佳聚类数. 可以看出, 对于计算密集型程序和 I/O 密集型程序, 每个 K 值下程序基准行为的聚类结果均对应多个最佳聚类数, 以这些候选的最佳聚类数作为评价聚类质量的条件. 表 3 展示了根据补充材料图 S1 得到的每个 K 值对应的最佳聚类数列表. 根据补充材料图 S1 和表 3 可知, I/O 密集型程序的候选最佳聚类数略少于计算密集型程序, 这是由于实验过程中的 I/O 密集型程序总数少于计算密集型程序, 使得 I/O 密集型程序在聚类过程中根据聚类数不同而体现在聚类目标函数上的波动性较小.

补充材料表 S1∼ S3 展示的是根据每种 K 值得到的所有程序基准行为聚类结果用于程序故障行为和失效行为分类时的类内平均相似度 (计算密集型程序和 I/O 密集型程序分别作出分析). 补充材料表 S1∼S3 分别考虑了表 2 所示的每种故障类型在故障注入实验中所得的故障行为和失效行为. 与补充材料表 S1∼S3 中的 “Trend” 对应的补充材料图 S2 和 S3 表示根据实验数据展现的类内平均相似度的趋势或规律, 用以说明聚类质量与 K 值、故障类型和 OCN 之间可能存在的关系.

根据补充材料表 S1∼S3 可以看出, 无论是计算密集型程序还是 I/O 密集型程序, 当程序的故障行为和失效行为依照基准行为的分类方式进行特征划分时, 当 K 值一定时, 不同故障类型下的故障行为和失效行为的类内相似度并无明显差异, 即聚类质量比较平均, 不存在基准行为的表征和聚类方法仅仅适用于划分某种故障类型下的程序行为特征的情形. 这说明, 文中提出的程序基准行为的表征和聚类方法是秉着抽取程序中固有特性进行分析的基本思想的, 其聚类结果是根据程序所反映出的 “故障 — 错误 — 失效” 链的基本原理而得到的, 因此, 这种聚类结果适用于程序在通用故障类型下的行为特征的划分.

补充材料图 S2 和 S3 展示的是根据补充材料表 S1∼S3 得到的针对故障行为和失效行为聚类过程中类内平均相似度的趋势和规律. 通过分析这种趋势和规律可进一步了解聚类质量与 K 值和 OCN 值之间的关系. 从补充材料图 S2 和 S3 可以看出, 故障行为和失效行为的聚类结果具有一致性, 当 K 值下存在 3 个候选的最佳聚类数 (OCN) 时, 通常处于中间值的 OCN 对应的平均类内相似度最高, 即聚类质量最高; 而当 K 值下只有 2 个候选的最佳聚类数 (OCN) 时, 较大的 OCN 对应较低的平均类内相似度 (即较低的聚类质量). 根据补充材料图 S2 和 S3 可以推测, 聚类质量的总体变化趋势是, 在每个 K 值下, 聚类质量随着 OCN 的增大而降低, 但是, 当存在 3 个或 3 个以上 OCN 时, 随着 OCN 的增大, 聚类质量首先出现峰值, 然后逐渐降低.

此外, 通过聚类因子反映的聚类质量展示的补充材料图 S6 中, 其中, S6(a) 展示的是以计算密集型程序基准行为进行故障行为和失效行为聚类依据的聚类质量. 以 S6(a) 为例做出分析: 在 S6(a) 展示的故障行为中, Ideal 折线 (即 IGP 折线) 对应的是对 SPEC 基准程序故障行为根据其自身的行为特征直接进行聚类所得到的每个指定聚类数下的 IGP 值, 理论上讲, 这个聚类结果是对程序故障行为来讲最理想的聚类结果, 从图中可以看出, Ideal 折线对应的数值均接近于 1. 可将 Ideal 折线看做以

IGP 表征的聚类质量的上界, 若以基准行为作为依据得到的故障行为聚类结果与该上界越接近, 则聚类质量越高. 每个聚类数 (CN) 对应的柱状图反映了在 [2,9] 内每个 K 值下程序故障行为依循其基准行为的聚类方式进行特征划分的 IGP 值. 可以看出, 在每个 CN 对应的一组 IGP 数值中, $K = 7$ 对应的 IGP 值最大, 即 $K = 7$ 时, 程序故障行为聚类质量最高. 从 Average 折线对应的 IGP 平均值可知, 随着聚类数的增加, IGP 值首先升高, 在 CN= 3 达到最高, 随后, IGP 值随聚类数增加而降低. 于是, 可以推断, CN= 3 是针对程序故障行为进行聚类的最佳聚类数, 且根据 $K = 7$ 对应的基准行为特征表达进行的程序聚类最能反映故障行为间的相似性和差异性. 因此, 通过图 S6(a) 可知, 当 $K = 7$ 和 CN= 3 时, 在程序基准行为的聚类结果中, 类内程序故障行为具有最大相似性、类间程序故障行为具有最大差异性.

在图 S6(a) 展示的失效行为中, IGP 值的变化趋势与故障行为聚类结果类似, 即 IGP 首先随聚类数 CN 的增大而升高, 当 CN= 3 时, IGP 平均值达到最高, 而后, IGP 值随 CN 增大而降低. 与图 S6(a) 中故障行为的聚类结果相比, 失效行为的聚类质量普遍高于故障行为的聚类质量, 且 IGP 值随 CN 增大而降低的幅度小于故障行为. 因此, 当 $K = 7$ 和 CN= 3 时, 程序基准行为的聚类结果最能反映类内 (类间) 程序失效行为之间的相似性 (差异性). 可以推测, 由于失效行为种类仅 4 种, 而故障行为共有 38 种类型, 所以故障行为的复杂性远高于失效行为, 由故障注入实验得到的故障行为的多样性也高于失效行为. 并且, 失效行为是故障行为在程序输出时的体现, 即使程序的中间运行状态千差万别, 程序运行结果可能屏蔽部分的故障行为多样性, 于是, 不同程序在故障条件下的失效行为体现了一定的稳定性同样, 图 S6(b) 中展示的 I/O 密集型程序的聚类结果与补充材料图 S6(a) 中的结果具有一致性.

5.2.3 关于 K 值

通过上述实验分析可知, 聚类的合理性、有效性和聚类质量并不是可独立评价的指标, 它与 K 值、最佳聚类数 OCN 有紧密联系. 特别是 K 值, 它直接决定程序状态子序列的长度. 而由前述可知, 状态子序列是对程序结构上、功能上等固有特性的体现. 对于正常运行的程序, 其状态子序列具有稳定性, 可用于表征程序的基准行为. 因此, 由合理的 K 值形成的状态子序列对于程序的故障检测、诊断等都具有重要意义. 由此可见, 针对 K 值的研究价值与入侵检测中的经典问题 —— 系统调用短序列长度的选取 —— 相媲美.

补充材料图 S7 展示了在每个 K 值下、每个 OCN 下、每种故障类型下的程序故障行为和失效行为根据基准行为被聚类的类内平均相似度. 此处, 由于是对每个 K 值下所有 OCN 对应的聚类结果的类内平均相似度做出综合考量, 程序故障行为和失效行为所反映的数据变化趋势并不一致, 这是由于故障行为和失效行为本身存在差异造成的. 但是, 可以看出, 计算密集型程序和 I/O 密集型程序的故障行为和失效行为所反映的数据变化趋势基本一致, 这说明, 3.1 小节给出的程序基准行为表征方法是从程序的运行特征出发, 旨在揭示程序运行时与可靠性相关的一般规律, 而这些规律不会因为程序类型的不同而发生变化. 此外, 从图 S7 可以看出, 当 $K = 7$ 时, 程序的故障行为和失效行为聚类后的类内平均相似度最高 (两者的 (similarity degree, SD) 数值最小, 根据定义 5, SD 数值越小表示程序相似性越高), 这表明, 当 $K = 7$ 时, 以程序基准行为的聚类方式对故障行为和失效行为进行特征划分的聚类质量最高根据前述分析, 只有对程序基准行为做出合理描述, 并反映其固有特征, 才能在聚类过程中将程序的本质特征区分开来. 因此, 根据图 S7 可知, $K = 7$ 对应的程序基准行为特征表达最能反映程序的固有特性.

为了进一步解释 $K = 7$ 为最合理 K 值的原因, 从程序基准行为本身出发, 分析 K 值与行为表征合理性之间的关系. 首先在每个 K 值下根据程序基准行为特征表达谱 (BBP) 得到程序行为相似矩

阵 (SDM), 其次计算得到任意两个程序基准行为之间相似度的平均值. 这个平均值有两重含义, 一是反映程序本身存在的差异; 二是反映由 K 值的不同导致的程序行为表征上的差异. 补充材料图 S8 展示了不同 K 值下程序基准行为之间的相似度平均值. 从图 S8 可知, 计算密集型程序和 I/O 密集型程序在 K 值与行为表征合理性之间的关系上具有高度一致性. 即多个程序行为之间的相似度随 K 值的增加而减小, 但是, 当 $K \geqslant 7$ 时, 程序行为相似度几乎不随 K 值发生变化, 从而趋于平稳. 这表明, 当 $2 \leqslant K \leqslant 7$ 时, 程序基准行为的差异性同时包含了程序本身的差异和由 K 值不同引入的差异, 而当 $K \geqslant 7$ 时, 程序本身的差异性逐渐占据了主导地位, 而由 K 值引入的行为差异随着行为相似度趋于稳定可逐渐被忽略. 因此, 当 $K = 7$ 时, 程序基准行为的表征方法最能体现程序固有特性.

6 结论与展望

首先, 提出了一种对程序基准行为、故障行为和失效行为的表征方法, 分别通过定义基准行为的特征表达 (BBS)、故障行为的特征表达 (FBS) 和失效行为的特征表达 (FLBS) 用以表征程序基准行为、故障行为和失效行为. 其次, 提出了一种考虑最佳聚类数的程序行为聚类方法. 最后, 分别以计算密集型基准程序和 I/O 密集型基准程序作为目标程序集, 设计了一组基于故障注入的程序行为聚类实验, 用以验证 "当两个程序具有相似的基本属性时, 其故障行为和失效行为也具有相似性" 推断的准确性. 实验结果体现了 3 方面内容: (1) 本文提出的程序行为表征方法和聚类方法具有合理性; (2) 推断得到了验证; (3) 针对影响实验结果的两个重要参数 (K 值和最佳聚类数 OCN) 进行了探讨, 确定了合适的参数值.

"当两个程序具有相似的基本属性时, 其故障行为和失效行为也具有相似性" 这一推断更深层次的含义是, 在评估软件可靠性时, 通过分析 "故障 — 错误 — 失效" 的链式效应明确软件失效机理, 从而能够跨越以定量统计分析获取软件故障模型和以定性解析分析评测可靠性之间的 "鸿沟", 着眼于软件产品中故障发生的一般规律, 从而提高软件产品的可度量性和可控性. 未来的工作将着眼于探讨程序基准行为的聚类结果能否反映类内程序之间的可靠性特征 (如寿命分布类型等) 的相似性. 若能说明此点, 则故障模型的定量统计与软件可靠性度量的定性分析之间便是相通的. 换言之, 软件可靠性的评测过程可加入对 "故障 — 错误 — 失效" 链所表征的失效机理的判定, 从而使得评测结果是可反馈的、也是可解释的.

补充材料 图 S1–S8, 表 S1–S3. 本文的补充材料见网络版 info.scichina.com. 补充材料为作者提供的原始数据, 作者对其学术质量和内容负责.

参考文献

1 Sarbu C, Johansson A, Suri N, et al. Profiling the operational behavior of OS device drivers. In: Proceedings of International Symposium on Software Reliability Engineering, Seattle, 2008. 127–136

2 Thakkar D, Jiang Z M, Hassan A E, et al. Retrieving relevant reports from a customer engagement repository. In: Proceedings of IEEE International Conference on Software Maintenance, Beijing, 2008. 117–126

3 Deissenboeck F, Juergens E, Lochmann K, et al. Software quality models: purposes, usage scenarios and requirements. In: Proceedings of ICSE Workshop on Software Quality, Vancouver, 2009. 9–14

4 da Silva O J, Crespo A N, Chaim M L, et al. Sensitivity of two coverage-based software reliability models to variations in the operational profile. In: Proceedings of 4th International Conference on Secure Software Integration and Reliability Improvement, Singapore, 2010. 113–120

5 Fu J P, Lu M Y. Develop software operational profile with uniform design. In: Proceedings of IEEE International Conference on Industrial Engineering and Engineering Management, Hong Kong, 2009. 842–846
6 Chen T Y, Kuo F C, Liu H. Application of a failure driven test profile in random testing. IEEE Trans Reliab, 2009, 58: 179–192
7 Sherwood T, Perelman E, Hamerly G, et al. Automatically characterizing large scale program behavior. Operat Syst Rev, 2002, 30: 45–57
8 Czeck E W, Siewiorek D P. Observations on the effects of fault manifestations as a function of workload. IEEE Trans Comput, 1992, 44: 559–566
9 Dickinson W, Leon D, Podgurski A. Finding failures by cluster analysis of execution profiles. In: Proceedings of International Conference on Software Engineering, Washington, 2001. 339–348
10 Podgurski A, Leon D, Francis P, et al. Automated support for classifying software failure reports. In: Proceedings of International Conference on Software Engineering, 2003. 465–474
11 Irrera I, Duraes J, Vieira M, et al. Towards identifying the best variables for failure prediction using injection of realistic software faults. In: Proceedings of Pacific Rim International Symposium on Dependable Computing, Tokyo, 2010. 3–10
12 Stephenson M W, Rangan R, Yashchin E, et al. Statistically regulating program behavior via mainstream computing. In: Proceedings of IEEE/ACM International Symposium on Code Generation and Optimization, New York, 2010. 238–247
13 Zarandi H R, Maghsoudloo M, Khoshavi N. Two efficient software techniques to detect and correct control-flow errors. In: Proceedings of 16th Pacific Rim International Symposium on Dependable Computing, Tokyo, 2010. 141–148
14 McMaster S, Memon A M. Call stack coverage for test suite reduction. In: Proceedings of IEEE International Conference on Software Maintenance, 2005. 539–548
15 Lorenzoli D, Mariani L, Pezzè M. Automatic generation of software behavioral models. In: Proceedings of ACM/IEEE International Conference on Software Engineering, Piscataway, 2008. 501–510
16 Auguston M. Software architecture built from behavior models. ACM SIGSOFT Softw Eng Notes, 2009, 34: 1–15
17 Gerrit S, Badriddline K. Large scale Itanium 2 processor OLTP workload characterization and optimization. In: Proceedings of International Workshop on Data Management on New Hardware, New York, 2006. 1–7
18 Park E, Cavazos J, Alvarez M A. Using graph-based program characterization for predictive modeling. In: Proceedings of the 10th International Symposium on Code Generation and Optimization, New York, 2012. 196–206
19 Khan A, Yan X. Workload characterization and prediction in the cloud-amultiple time series approach. In: Proceedings of Network Operations and Management Symposium, Piscataway, 2012. 1287–1294
20 Pentakalos O I, Menascé D A, Yesha Y. Automated clustering-based workload characterization. In: Proceedings of NASA Conference Publication, 1996. 253–264
21 Eeckhout L, Vandierendonck H, De Bosschere K. Quantifying the impact of input data sets on program behavior and its applications. J Instr-Lev Parall, 2003, 5: 1–33
22 Stoerzer M, Ryder B G, Ren X, et al. Finding failure-inducing changes in Java programs using change classification. In: Proceedings of ACM SIGSOFT International Symposium on Foundations of Software Engineering, New York, 2006. 57–68
23 Zhao G Y, Zhao G. System fault behavior model. In: Proceedings of International Conference on Reliability, Maintainability and Safety, Chengdu, 2009. 925–929
24 Zhao G Y, Kang R, Zhao G, et al. System fault behavior model considering the effects of structural factors and method of its description. In: Proceedings of International Conference on Dependability, Venice, 2010. 118–124
25 Zhao G Y, Sun Y F, Hu W W. Structure-based system fault behavior modeling method. In: Proceedings of Reliability and Maintainability Symposium, Colorado, 2014. 1–7
26 Pazzi L, Interlandi M, Pradelli M. Automatic fault behavior detection and modeling by a state-based specification method. In: Proceedings of International Symposium on High-Assurance Systems Engineering (HASE), San Jose, 2010. 166–167
27 DiGiuseppe N, Jones J A. Software behavior and failure clustering: an empirical study of fault causality. In: Proceedings of International Conference on Software Testing, Verification and Validation, Montreal, 2012. 191–200
28 Fenacci D, Franke B, Thomson J. Workload characterization supporting the development of domain-specific compiler

optimizations using decision trees for data mining. In: Proceedings of International Workshop on Software and Compilers for Embedded Systems, New York, 2010. 1–10
29 Li N, Yu S Z. Periodic hidden Markov model-based workload clustering and characterization. In: Proceedings of IEEE International Conference on Computer and Information Technology, Sydney, 2008. 378–383
30 Du W L, Mathur A P. Testing for software vulnerability using environment perturbation. In: Proceedings of International Conference on Dependable Systems and Networks, Los Alamitos, 2000. 603–612
31 Kullback S, Leibler R A. On information and sufficiency. Ann Math Stat, 1951, 22: 79–86
32 Fang W W, Ma Z R, Roberts F S. A measure of discrepancy of multiple sequences. Inf Sci, 2001, 137: 75–102
33 Wu K L, Yu J, Yang M S. A novel fuzzy clustering algorithm based on a fuzzy scatter matrix with optimality tests. Pattern Recogn Lett, 2005, 26: 639–652
34 Porter R, Canagarajah N. A robust automatic clustering scheme for image segmentation using wavelets. IEEE Trans Image Process, 1996, 5: 662–665
35 Frey B J, Dueck D. Response to comment on "Clustering by passing messages between data points". Science, 2008, 319: 726d
36 Jin A, Jiang J H. Fault injection scheme for embedded programs at machine code level and verification. In: Proceedings of Pacific Rim International Symposium on Dependable Computing, Shanghai, 2009. 55–62
37 Duraes J A, Madeira H S. Emulation of software fault: a field data study and a practical approach. IEEE Trans Softw Eng, 2006, 32: 849–867
38 Kapp A V, Tibshirani R. Are clusters found in one dataset present in another dataset? Biostatistics, 2007, 8: 9–31

A method for validating the effectiveness of fault clustering and failure clustering of programs

ZHANG DanQing[1*], JIANG JianHui[1] & CHEN LinBo[2]

1 *School of Software Engineering, Tongji University, Shanghai* 201804*, China*;
2 *China Securities Depository and Clearing Corporation Limited Shanghai Branch, Shanghai* 200120*, China*
*E-mail: helen_dandan@hotmail.com

Abstract An important issue on software reliability is the investigation of fault behaviors of programs. However, due to that the set of fault manifestations is infinite, it is necessary to reduce the fault behavior space to a manageable level for promoting efficiency in reliability estimation. The hypothesis that programs having similar attributes being considered to have similar fault and failure behaviors provides a good basis for fault behavior space reduction. But this hypothesis has not been verified, which is our core work in this paper. Firstly, we characterize program executions in order to model runtime behaviors, including baseline behaviors, fault behaviors and failure behaviors. Then, a typical fuzzy technique with our improved method is proposed. After that, an experimental method of fault injection is used to verify the aforementioned hypothesis. For the CPU-bound benchmarks (SPEC CPU2000 and SPEC CPU2006 suites of benchmarks) and I/O-bound benchmarks (IOZONE, DEBENCH, etc.), our results show two things: (1) the method of program characterization and clustering shows its effectiveness; (2) it is able to examine and cluster fault and failure behaviors based on their baseline behaviors, which verifies the aforementioned hypothesis.

Keywords baseline behavior, fault behavior, failure behavior, behavior spectrum, behavior similarity degree

ZHANG DanQing was born in 1984. She received the Master's degree in computer science and technology in Wuhan Institute of Digital Engineering, China, in 2009. Currently, she is a Ph.D. candidate in the School of Software Engineering in Tongji University. Her main research interests include dependable computing and software reliability evaluation.

CHEN LinBo was born in 1984. He received the Ph.D. degree in computer science and technology in Tongji University, Shanghai, in 2013. Currently, he is a software engineer in IT Development Department in China Securities Depository and Clearing Corporation Limited Shanghai Branch. His main research interests include information security, intrusion detection and prevention and intrusion tolerance.

JIANG JianHui was born in 1964. He received the Ph.D. degree in computer science and technology in Shanghai Railway University, Shanghai, China, in 1999. Currently, he is a professor and Ph.D. supervisor in School of Software Engineering, Tongji University. His current research interests include dependable system and network, software reliability engineering, and VLSI/SoC fault-tolerance and test.