基于 Chameleon 聚类分析的多错误定位方法

曹鹤玲[1,3],姜淑娟[2]
(1. 河南工业大学信息科学与工程学院,河南郑州 450001;2. 中国矿业大学计算机科学与技术学院,江苏徐州 221116;
3. 粮食信息处理与控制教育部重点实验室,河南郑州 450001)

摘 要: 软件系统中往往存在多个错误,它们之间互相干扰,这抑制了错误定位的能力. 为解决该问题,提出一种基于 Chamelelon 聚类分析的多错误定位方法. 首先,将每一个失败程序执行轨迹和所有成功程序执行轨迹合并,计算其怀疑度,按怀疑度大小选取高可疑元素作为程序执行轨迹的特征元素,按照该特征元素对失败程序执行轨迹进行约简;其次,聚类分析将失败程序执行轨迹分簇,每簇包含一个错误;然后,将失败程序执行轨迹簇与所有成功程序执行轨迹合并,重新计算其怀疑度;最后,根据合并后的簇生成的怀疑度序列,采用并行调试模式同时定位程序中的多个错误. 实证研究表明该方法可以有效地定位程序中的多个错误.

关键词: 软件调试; 错误定位; 聚类分析; 多错误
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2017)02-0394-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.02.018

Multiple-Fault Localization Based on Chameleon Clustering

CAO He-ling[1,3], JIANG Shu-juan[2]
(1. *College of Information Science and Engineering, Henan University of Technology, Zhengzhou, Henan* 450001, *China*;
2. *School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangshu* 221116, *China*;
3. *Key Laboratory of Grain Information Processing and Control, Ministry of Education, Zhengzhou, Henan* 450001, *China*)

Abstract: There exist usually multiple faults in software systems. Mutual interference among them inhibits the ability of fault localization. A multiple fault localization based on Chameleon clustering was proposed. First, the suspiciousness of program elements is computed based on the combination of each failed program execution trace with all passed program execution traces. The most suspicious elements are selected as feature elements, which reduced the corresponding failed program execution traces. Second, the reduced failed program execution traces are performed by clustering analysis, after that, each failed cluster contains one fault. Third, each failed cluster merges passed cluster, and then the suspiciousness of program elements is computed. Finally, multiple faults are located simultaneously in terms of the descending suspiciousness of program elements in each failed cluster in parallel debugging mode. Experimental results show that the approach located multiple faults effectively.

Key words: program debugging; fault localization; cluster analysis; multiple faults

1 引言

目前,研究人员已在软件错误定位领域进行了大量研究,但是大多数工作只关注单个错误的定位问题,而对多错误的定位问题研究较少. 然而,一个软件系统中往往包含多个错误,它们之间会互相干扰,降低了错误定位的效率[1]. 这给软件错误定位研究带来巨大的挑战. 因此,深入研究多错误的定位问题,设计出更加高效的多错误定位模型或方法具有重要意义.

程序执行失效暴露了程序中包含错误. 王建峰等人根据组合测试数据执行结果生成可能的错误交互集来查找软件错误[2]. 然而调试人员通常不清楚是由单个错误还是由多个错误引起程序执行失效. 调试人员可以选择使用一个失败测试用例来寻找程序执行失败

收稿日期:2016-01-25;修回日期:2016-09-28;责任编辑:蓝红杰
基金项目:国家自然科学基金(No. 61602154, No. 61673384, No. 61502497, No. U1404617);粮食信息处理与控制教育部重点实验室资助项目(No. KFJJ-2016-105);河南省高等学校重点科研项目(No. 16A520005);河南工业大学高层次人才基金(No. 2015BS006);河南工业大学"省属高校基本科研业务费专项资金"(No. 2016QNJH28);河南省重点科技攻关项目(No. 162102310405, No. 152102110075)

的原因[3],也可以使用全部失败测试用例来检查程序中的错误[4]. 调试人员发现一个错误并对其修复,然后重新测试先前失败测试用例,执行成功则停止检查;执行失败则需要重新测试程序以查找错误. 若程序中有多个错误,则需要重复多次这样的测试过程,降低了错误定位效率. 如何识别软件中错误数量以及哪些测试用例执行失败是由同一错误导致的,是多错误定位中的一个难题.

针对多错误的定位问题,Jones 等人[5]提出了基于传统凝聚层次聚类的错误定位方法. 该方法首先收集每一个失败测试用例和所有成功测试用例的执行轨迹信息,计算程序实体怀疑度,并选取前 20% 可疑程序实体来约简失败执行轨迹信息;其次,使用 Jaccard 系数计算任意两个失败程序执行轨迹的相似度,根据给定阈值将失败执行轨迹合并;然后,重新计算合并后失败执行轨迹和所有成功执行轨迹的怀疑度,并重新选取前 20% 的可疑程序实体来约简失败执行轨迹信息;最后,当满足给定阈值时,聚类分析结束,将由同一错误引起的程序执行失败的轨迹信息聚为一簇,其结果应用于错误定位.

然而,上述方法存在以下 3 个问题:(1) 每次聚类分析合并后,需要将每簇中失败测试用例和成功测试用例的执行轨迹信息合并,重新计算怀疑度,选取前 20% 高可疑程序实体,计算量较大. (2) 传统凝聚层次聚类算法在某次迭代过程中若将某样本点错误地划分到某簇中,那么该样本点在后面的划分中永远属于这个簇,无法撤销,这将影响聚类分析结果的准确度. (3) 传统凝聚层次聚类算法时间代价较高,时间复杂度为 $O(tn^2)$,其中,t 为迭代次数,n 为样本点个数.

基于动态建模的 Chameleon 聚类分析算法[7]可以克服传统凝聚层次聚类的缺点,采用更加全局的观点对数据点进行聚类,若两个簇的相对互连度很大又离得很近就将其合并,自动适应被合并簇的内部特征. 本文在进行聚类分析时:(1) 只在聚类初始化时对执行轨迹信息进行预处理,即将每个失败执行轨迹和所有成功执行轨迹合并,计算程序实体的怀疑度,并挑选高可疑元素对该失败执行轨迹进行约简;(2) 聚类分析过程中无需迭代(1)的过程. 而文献[5]中聚类分析的处理过程中,聚类分析每合并一次就重复一次(1)的过程,时间开销较大. 为此,本文提出一种基于 Chameleon 聚类分析的多错误定位方法来解决多错误定位问题.

2　预备知识

2.1　Chameleon 聚类分析

Chameleon(变色龙)算法[7]是一种层次聚类算法,与传统凝聚层次聚类算法不同,该算法在合并两个簇时采用更高的标准来提高聚类质量. 具体而言:该算法中簇之间的相似度主要依据簇中对象的链接情况和簇的近邻性两个指标来评定,若两个簇相对互连度很大又离得很近就将其合并. 该算法的特征决定了它能使新簇自动地适应被合并簇的内部特征,尤其会考虑簇内在特征,这类聚类方法有利于构造任意形状的簇. 每个程序执行轨迹信息相当于聚类分析中的一个数据点,用一个圆圈表示,如图 1 所示.

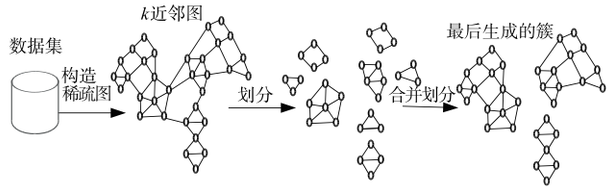


图1　基于k最近邻和动态建模的Chameleon层次聚类

Chameleon 聚类是一种两阶段聚类法. 第一阶段把这些数据对象分成很多小的簇;对软件错误定位来说是将程序失败执行轨迹信息分成很多小的簇. 根据数据点的相似度构造稀疏图,采用 k-邻近图的方式动态地捕捉某数据对象的邻域,划分生成较小的子簇. 第二阶段根据聚类相近程度合并这些小的簇,计算任意两个簇的相对互连度和相对接近,当两个指标都比较大时才合并这两个簇,生成最终聚类的簇,如图 1 合并划分生成的最终结果所示.

定义 1　相对互连度 $RI(C_i,C_j)$[6]. 定义为两个簇 C_i 和 C_j 之间的绝对互连度关于这两个簇 C_i 和 C_j 的内部互连度的规范化表示,也就是

$$RI(C_i,C_j)=\frac{|EC(Ci,Cj)|}{\frac{1}{2}(|EC(Ci)|+|EC(Cj)|)} \tag{1}$$

其中,$EC(C_i,C_j)$表示包含簇 C_i 和 C_j 的割边,$EC(C_i)$、$EC(C_j)$表示的是将 C_i、C_j 划分成大致相等两部分的割边的最小和.

定义 2　相对接近度 $RC(C_i,C_j)$[6]. 表示簇 C_i 和 C_j 之间的绝对接近度关于这两个簇 C_i 和 C_j 的内部接近度的规范化,即

$$RC(C_i,C_j)=\frac{|\bar{S}_{EC}(C_i,C_j)|}{\frac{|C_i|}{|C_i|+|C_j|}|\bar{S}_{EC}(C_i)|+\frac{|C_j|}{|C_i|+|C_j|}|\bar{S}_{EC}(C_j)|} \tag{2}$$

其中,$\bar{S}_{EC}(C_i,C_j)$是连接簇 C_i 中顶点和簇 C_j 中顶点的边的平均权重,$\bar{S}_{EC}(C_i)$和$\bar{S}_{EC}(C_j)$是最小二分簇的边的平均权重.

定义 3　最显著错误(prominent fault)[1]. 是在软件错误定位的检查过程中最先被检测出来的错误,而在软件中可能包含一个或多个错误.

2.2 调试模式

(1)串行调试模式(Sequential)[5]. 也称为序列调试模式,运行包含多个错误的程序版本以获取程序执行信息,调试人员定位到一个错误就修复该错误;然后,重新测试后定位出下一个错误并修复之,不断迭代该过程直到定位出程序中最后一个错误,这种一次定位一个错误(one bug at a time)的模式被称为串行调试模式.

(2)并行调试模式(Parallel)[5]. 运行包含多个错误的程序版本以获取程序执行信息,采用某种技术(例如聚类分析)或模型将失败执行轨迹划分成多个分区(簇),每个分区(簇)指向同一个错误,多个调试人员同时定位程序中的多个错误的模式被称为并行调试模式[5].

本文采用聚类分析对失败执行轨迹分簇,然后采用并行调试模式,称之为聚类_并行调试模式(Cluster_parallel). 假设两个调试人员同时调试程序中的多个错误,当一个调试人员定位出某个错误时,就以消息传递的方式通知另一个调试人员错误的语句行号,然后另一个调试人员就开始检查该错误是否是正在调试簇中执行轨迹执行失败的原因,若是此原因,则停止检查. 这是为了尽量减少两个簇中包含同一个错误而引起检查代价浪费. 若检查所有簇都没有定位到全部错误,则需要重新运行程序以定位更多的错误. 这样测试一次定位程序中的多个错误,可以减少调试交互次数和迭代测试的成本.

3 基于 Chameleon 聚类分析的多错误定位

针对软件中存在的多错误问题,本文提出一种基于 Chameleon 聚类分析的多错误定位方法. 该方法首先将每一个失败执行轨迹信息和所有成功执行轨迹信息合并,使用 Tarantula 方法[4]计算其怀疑度,选取前 20% 高可疑程序元素作为程序执行轨迹的特征元素,再按照特征元素对失败执行轨迹信息进行约简;其次,对约简后的失败执行轨迹信息进行 Chameleon 聚类分析;再次,将每个失败执行轨迹簇(C_i)和所有成功执行轨迹(S)合并为同一个簇($C_i \cup S$),使用 Tarantula 方法对该簇中程序元素进行怀疑度计算;最后,根据每个合并簇生成的怀疑度序列,采用多个调试人员同时调试的方式定位程序中的多个错误. 基于 Chameleon 聚类分析的多错误定位模型如图 2 所示,具体包括以下 3 个阶段.

3.1 数据预处理

首先,测试用例驱动源代码执行程序,将获得的执行轨迹信息根据测试结果分为成功执行轨迹信息和失败执行轨迹信息两类;然后,将每一个失败执行轨迹信息和所有成功执行轨迹信息构成一个簇,使用 Tarantula 方法计算程序语句怀疑度,对语句按怀疑度从大到小排序,取前 20% 高可疑语句[9]作为每个失败执行轨迹信息的特征元素;最后,根据该特征元素对失败执行轨迹信息进行约简.

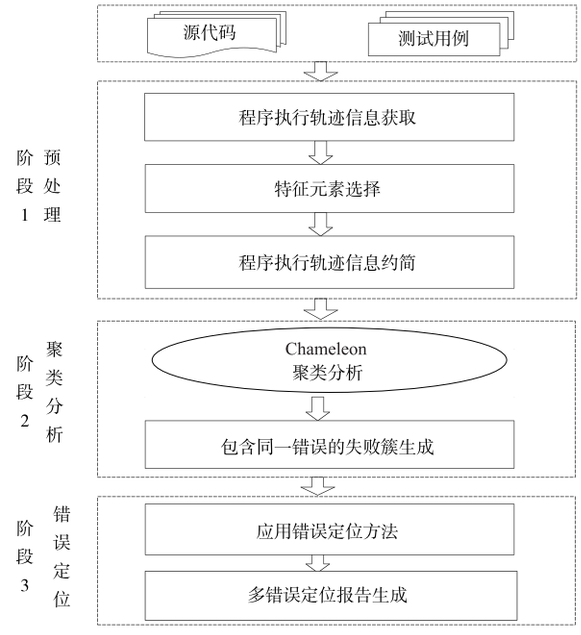


图2 基于Chameleon聚类分析的多错误定位模型

3.2 聚类分析

在包含多错误的程序执行轨迹信息中,若失败执行轨迹信息中的高可疑元素指向同一个区域,则它们可能是由同一错误引起的程序失败. 因此,采用聚类分析将失败执行轨迹信息分为不同簇,期望每个簇包含一个错误,簇数表示程序中错误个数.

Chameleon 聚类为两阶段聚类法. 第一阶段将失败程序执行轨迹信息分成很多小的簇. 首先,将任意两个失败执行轨迹中的元素存入集合 A 和 B;其次,根据 Jaccard 系数计算其相似度 $Similarity(A,B)$ 如公式(3)所示,同时将相似度作为两个失败执行轨迹连接边的权重值,从而生成权重矩阵;然后,根据该权重矩阵寻找一个数据对象最相似的 k 个近邻数据对象,若两个数据对象最相似,则它们之间存在一条边,这些边加权后反映数据对象间的相似度,生成 k - 近邻图;最后,根据图划分算法将 k - 近邻图划分成大量相对较小的子簇. 第二阶段根据数据对象之间的相似度将这些较小的簇合并. 计算任意两个簇的相对互连度和相对近邻度,当 $RI(C_i,C_j) * RC(C_i,C_j)$ 大于给定阈值时合并子簇,从而将失败执行轨迹信息分簇.

$$Similarity(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{3}$$

3.3 错误定位

在多错误定位过程中,首先将每个失败执行轨迹簇

C_i 和所有成功执行轨迹 S 构成一个簇,再使用 Tarantula 方法计算合并后的簇中的程序语句的怀疑度;然后,根据合并后簇生成的怀疑度序列;最后,定位出每簇中包含的错误,采用并行调试模式同时定位程序中的多个错误.

4　实验评估

为验证 Chameleon 聚类分析对多错误定位效率的提升程度,我们选取经典错误定位方法 Tarantula[4],分别比较聚类分析前后以及采用串行调试模式和并行调试模式定位多错误时的效率.

4.1　实验对象

本文主要测试了中等规模 gzip、grep、sed 和 flex 共 4 个 C 程序[9],从 SIR(Subject Infrastructure Repository)(网址:http://sir.unl.edu/portal/index.html)下载. gzip 包含 5000 多行代码、211 个测试用例,实现压缩和解压缩数据的功能;grep 包含 9000 多行代码、806 个测试用例,实现在某一文件内匹配指定模式的功能;sed 包含了 6000 多行代码、360 个测试用例,实现文件处理的功能;flex 包含近万行代码、567 个测试用例,实现词法分析功能.

对测试的每个程序版本,随机选择一个错误,迭代增加其它错误,每引入一个错误就保存一个程序版本,直到生成包含 10 个错误的程序版本,从而生成 1-,2-,3-,4-,5-,6-,7-,8-,9-,和 10-fault 的错误版本,如表 1 所示. 程序执行覆盖信息收集的实验平台为 Ubuntu 12.04.2,编译器为 gcc-4.6.3,使用 gcc 的 Gcov 组件获取该信息. 多错误定位的实验在 Eclipse 平台上实现.

表 1　程序及错误个数

错误数	gzip	grep	sed	flex
# of 1-Fault Versions	18	14	12	22
# of 2-Fault Versions	30	30	30	30
# of 3-Fault Versions	30	30	30	30
# of 4-Fault Versions	30	30	30	30
# of 5-Fault Versions	30	30	30	30
# of 6-Fault Versions	30	30	30	30
# of 7-Fault Versions	30	30	30	30
# of 8-Fault Versions	30	30	30	30
# of 9-Fault Versions	30	30	30	30
# of 10-Fault Versions	30	30	30	30
Total # Versions	288	284	282	292

4.2　多错误定位的评价指标

(1)错误定位代价(*Cost*)[8]. 为找到程序版本中错误时需检查的语句数(Rank of faults)占所有可执行代码行数(Number of executable statements)的百分比,如公式(4)所示.

$$Cost = \frac{Rank\ of\ faults}{Number\ of\ executable\ statements} \times 100\% \quad (4)$$

评价定位第 1 个错误的定位代价为 Cost,评价定位到全部(n 个)错误的定位代价如公式(5)所示.

$$Total = \sum_{i=1}^{n} \mathrm{Cost}_i \quad (5)$$

(3)检查得分(EXAM). 为了更加直观地比较不同方法的优劣,我们还使用 EXAM 指标进行评估. EXAM 指标为错误检出率(% of faults located)与代码检查率(% of code examined)的比值如公式(6)所示.

$$\mathrm{EXAM} = \frac{\%\ \text{of faults located}}{\%\ \text{of code examined}} \quad (6)$$

(4)调试迭代次数(Iterations). 程序中有多个错误,当采用一次定位一个错误的串行调试模式时,迭代次数 Iterations 为错误的个数 n;当采用聚类分析技术后,假设聚类分析后定位到 $m(m \leqslant n)$ 个错误,聚类_并行调试模式的迭代次数 Iterations 为 $(n-m+1)$,如公式(7)所示.

$$\mathrm{Iterations} = \begin{cases} n, & \mathrm{Sequential} \\ n-m+1, & \mathrm{Cluster\_parallel} \end{cases} \quad (7)$$

4.3　实验结果分析

(1) 定位到第 1 个错误的错误定位效率对比

为了直观地展示程序中包含 1 个或多个错误时,定位出第 1 个(最显著)错误的定位代价,我们以盒图形式展示错误定位代价. 图 3(a) ~ (d)展示了 gzip、grep、sed、flex 4 个程序定位多个错误中第 1 个错误的定位代价范围,其中,横坐标表示测试程序中包含的错误数量,纵坐标表示定位出程序中多个错误中第 1 个错误的定位代价范围;虚线左侧表示串行调试模式定位第 1 个错误的定位代价范围,虚线右侧表示聚类分析后并行调试模式定位第 1 个错误的定位代价范围. 每个盒图展示了错误定位代价的平均值、1/4 和 3/4 分位点、最大值和最小值.

图 3 展示了错误定位代价与程序中错误数量是负相关的关系,也就是说定位到第 1 个错误的定位代价随着错误数量的增加而减少. 例如,对于图 3(a)中 gzip 程序,采用串行调试模式,测试包含 1 个错误的程序,定位出第 1 个错误的代价为 3.75%;测试包含 2 个错误的程序,定位出第 1 个错误的代价为 3.00%;测试包含 8 个错误的程序,定位第 1 个错误的代价为 1.21%. 由每个错误数量的盒图的平均值可以看出,随着错误数量的增加,定位出第 1 个错误的平均错误定位代价有所降低. 图 3(b) ~ (d)中也呈现了相同的趋势,即定位出第 1 个错误的平均错误定位代价随着错误数量的增加而减少.

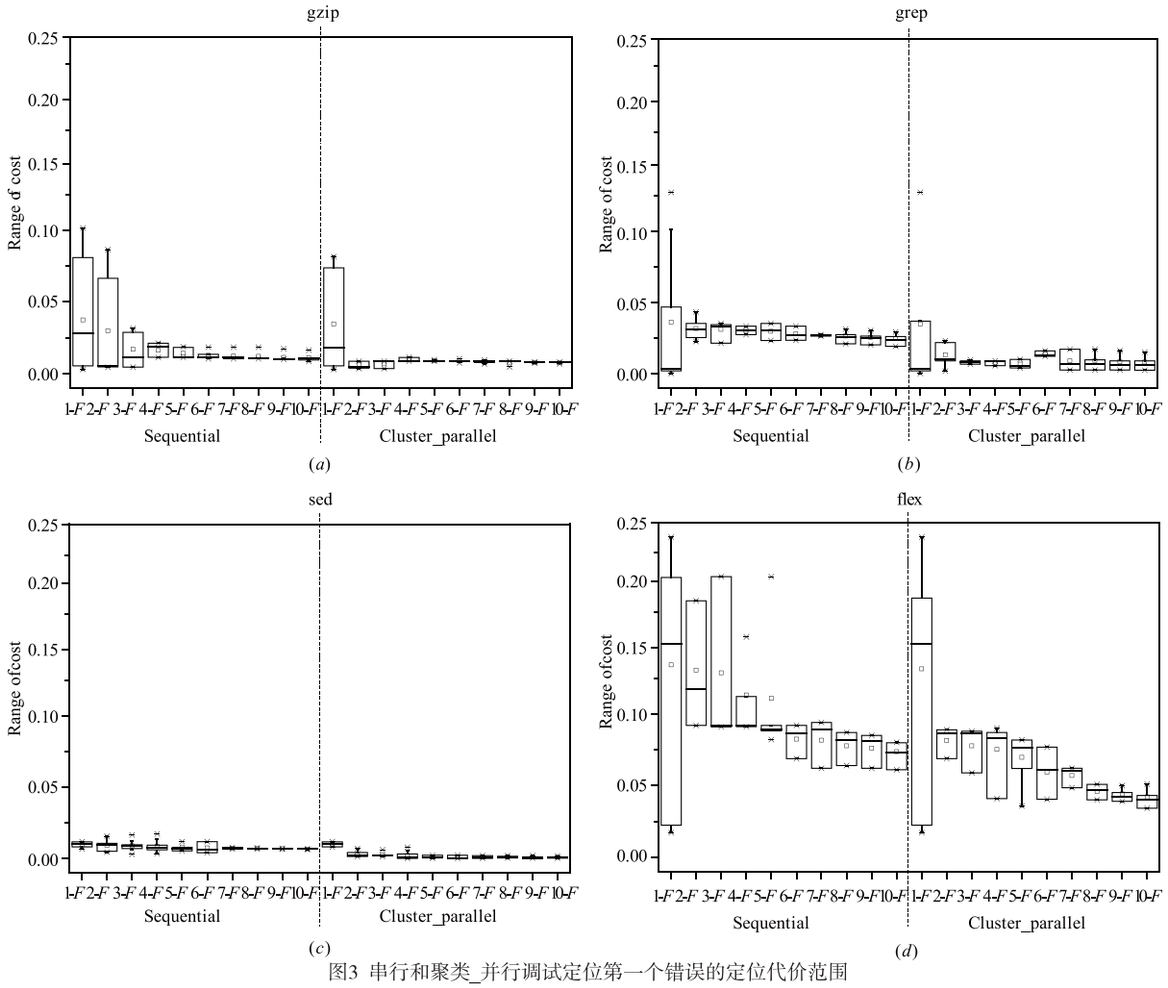


图3 串行和聚类_并行调试定位第一个错误的定位代价范围

从图3中虚线左侧和右侧每个错误数量对应的盒图数据的平均值可以看出:聚类分析能明显降低定位出多个错误中第1个错误的平均错误定位代价,但是错误定位代价的降低程度受程序中错误数量和不同程序的影响. 例如,对于gzip程序,包含2个错误时,聚类分析使定位第1个错误的平均错误定位代价从3.00%下降到0.64%,效率提升幅度为78.63%;而包含3个错误时,聚类分析使定位第1个错误的平均错误定位代价从1.70%下降到0.68%,效率提升幅度为60.08%.

(2)定位到全部错误的错误定位效率对比

图4以盒图形式分别展示了gzip、grep、sed、flex定位出全部错误的错误定位代价范围,其中,横坐标表示程序中的错误数量,纵坐标表示定位出程序全部错误的定位代价范围. 图4中虚线左侧表示串行调试模式定位出全部错误的定位代价,虚线右侧表示聚类分析后定位出全部错误的定位代价.

图4展示了定位全部错误的定位代价随着错误数量的增加而增加. 从虚线左右两侧对应的盒图可以明显看出:Chameleon聚类分析后的错误定位代价明显低于串行调试模式的错误定位代价. 例如,对于图4(*a*)中gzip程序,串行调试模式定位2个错误的平均错误定位代价为7.29%;而聚类分析后定位2个错误的平均错误代价为3.58%,效率提升幅度为50.89%. 图4(*b*)~(*d*)中也呈现相同趋势. 总体上,聚类分析明显降低了定位全部错误的错误定位代价.

(3)与文献[5]方法的对比

表2中数据为本文方法与文献[5]方法定位到全部错误时的实验结果,展示了程序中包含1~10个错误时,对于测试的4个程序,文献[5]中方法和本文方法定位出全部错误的平均错误定位代价. 从表2中数据可以分析出本文方法的错误定位代价低于文献[5]方法的错误定位代价,这是因为文献[5]方法主要采用传统凝聚层次聚类,在某次迭代过程中若将某样本点错误地划分到某簇中,那么该样本点在后面划分中永远属于这个簇,无法撤销,这将影响聚类分析结果的准确度,从而影响错误定位的效果.

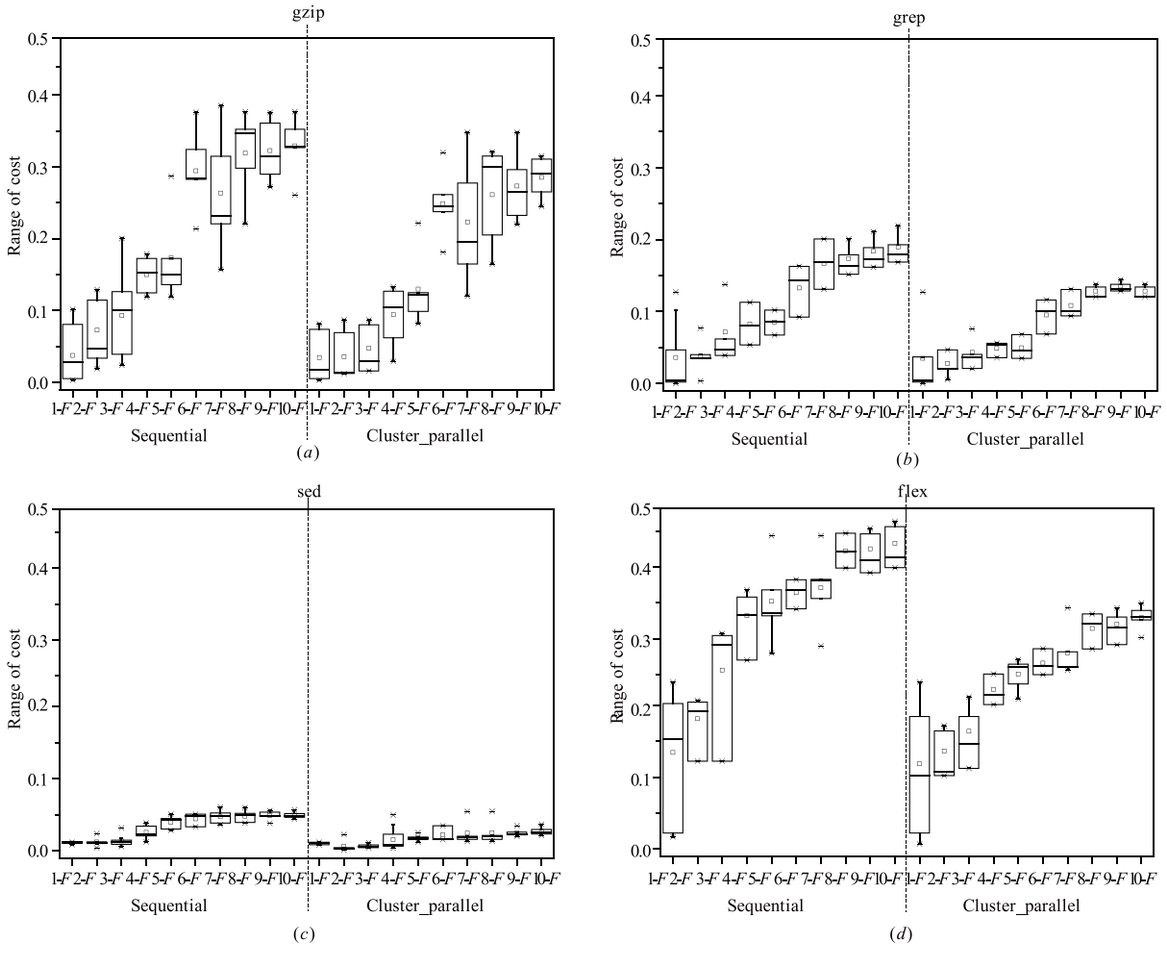


图4 串行和聚类_并行调试定位全部错误的定位代价范围

表 2　本文方法和文献[5]方法定位全部错误代价对比

程序	对比方法	1-F	2-F	3-F	4-F	5-F	6-F	7-F	8-F	9-F	10-F
gzip	文献[5]	3.65%	6.20%	7.35%	12.09%	15.21%	27.48%	25.36%	29.81%	30.20%	30.85%
	Cluster_parallel	3.47%	3.58%	4.75%	9.43%	12.99%	24.90%	22.33%	26.13%	27.32%	28.52%
grep	文献[5]	3.52%	3.62%	6.34%	7.29%	7.24%	11.21%	14.62%	15.31%	16.16%	17.81%
	Cluster_parallel	3.49%	2.79%	4.33%	4.87%	4.96%	9.52%	10.85%	12.84%	13.61%	14.08%
sed	文献[5]	1.05%	1.11%	1.22%	2.23%	3.10%	3.98%	3.98%	4.29%	4.65%	4.72%
	Cluster_parallel	1.02%	0.60%	0.68%	1.54%	1.79%	2.22%	2.45%	2.49%	2.51%	2.73%
flex	文献[5]	13.31%	17.24%	23.49%	31.38%	33.87%	35.32%	36.13%	41.24%	41.32%	41.45%
	Cluster_parallel	13.15%	13.64%	16.39%	22.93%	25.08%	26.60%	28.03%	31.41%	32.03%	32.95%

(4) 测试迭代次数对比

表 3 展示了采用串行调试模式和聚类_并行调试模式时,测试 gzip、grep、sed 和 flex 的测试迭代次数. 串行调试模式定位多个错误的迭代次数为程序中包含的错误数量,如表 3 中第 2 行所示("all"指代以上 4 个程序). 聚类_并行调试模式定位错误的测试迭代次数如表 3 中第 3 ~ 6 行所示. 从表中数据可见:当程序中包含 1 个或 2 个错误时,并行调试模式的测试迭代次数为 1,当程序中包含 3 ~ 10 个错误时,并行调试模式的测试迭代次数为 1.1 ~ 2.3 次,虽然这种情形下测试迭代次数大于 1,但是明显小于串行调试模式的测试迭代次数,从而降低了调试代价. 对于包含多个错误的程序来说,串行调试模式定位错误的迭代次数较多,代价较大.

表 3 串行和聚类_并行调试迭代次数

方法	程序	1-F	2-F	3-F	4-F	5-F	6-F	7-F	8-F	9-F	10-F
Sequential	all	1	2	3	4	5	6	7	8	9	10
Cluster_parallel	gzip	1	1	1.1	1.2	1.4	1.9	2.1	2.2	2.3	2.3
	grep	1	1	1.1	1.2	1.4	1.7	2.1	2.2	2.3	2.3
	sed	1	1	1.2	1.3	1.5	1.8	2	2	2.1	2.1
	flex	1	1	1	1.2	1.3	1.8	1.9	2.1	2.2	2.2

为验证实验结果可靠性,使用 Wilcoxon 符号秩和检验来验证本文方法与对比方法的效率提升是否显著,取显著性水平 $\alpha = 0.05$. Wilcoxon 符号秩和检验由 Wilcoxon 于 1945 年提出,用于比较两个非正态总体样本的显著性,是建立在秩和统计量基础上的非参数假设检验方法,错误定位代价的数据满足该特征. 验证如下假设:本文方法与对比方法的错误定位效率没有显著差别. Wilcoxon 符号秩和检验计算出 p 值都小于 0.05,拒绝原假设. 因此,认为本文方法相比对比方法的效率提升是显著的.

5 总结

多错误的定位问题是软件调试中的一个难题. 本文提出了基于 Chameleon 聚类分析的多错误定位方法,通过聚类分析技术把失败程序执行轨迹信息分簇,每簇包含一个错误,从而定位程序中的多个错误. 首先,将每一个失败执行轨迹和所有成功执行轨迹合并,计算其怀疑度,选取高可疑元素对相应失败执行轨迹进行约简;其次,计算任意两个失败簇的相对互连度和相对接近度,根据给定阈值合并子簇生成失败执行轨迹的最终簇;再次,将每个失败执行轨迹簇和所有成功执行轨迹合并为一个簇,计算合并后簇的语句怀疑度并生成语句检查序列;最后,按生成的语句检查序列定位出相应的错误,采用并行调试模式来定位程序中的多个错误. 实验表明,在包含多个错误的程序中定位出第 1 个错误或全部错误时,本文方法比串行调试模式错误定位效率有所提升,并且调试迭代次数有所下降.

参考文献

[1] DiGiuseppe N, Jones J A. Fault density, fault types, and spectra-based fault localization[J]. Empirical Software Engineering, 2014. 1 – 40.

[2] 王建峰,魏长安,盛云龙,姜守达,基于错误交互集的组合测试软件故障定位方法[J]. 电子学报,2014,42(6):1173 – 1178.
Wang Jianfeng, Wei Chang-an, et al. Locating errors in combinatorial testing using set of possible faulty interactions[J]. Acta Electronica Sinica, 2014, 42(6): 1173 – 1178. (in Chinese)

[3] Zhang X, Gupta N, et al. Locating faults through automated predicate switching[A]. Proceedings of the 28th International Conference on Software Engineering[C]. New York, USA: ACM, 2006. 272 – 281.

[4] Jones J A, Harrold M J, et al. Visualization of test information to assist fault localization[A]. Proceedings of the 24th International Conference on Software Engineering[C]. Los Alamitos, CA: IEEE Computer Society, 2002. 467 – 477.

[5] Jones J A, Bowring J F, et al. Debugging in Parallel[A]. Proceedings of the 2007 International Symposium on Software Testing and Analysis[C]. London, United Kingdom: ACM, 2007. 16 – 26.

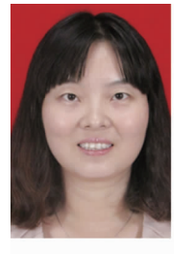
[6] 韩家炜,裴健. 数据挖掘:概念与技术(第 3 版)[M]. 北京:机械工业出版社,2014. 100 – 103.

[7] Karypis G, Han E-H, et al. Chameleon: Hierarchical clustering using dynamic modeling[J]. Computer, 1999, 32(8): 68 – 75.

[8] Renieris M, Reiss S P. Fault localization with nearest neighbor queries[A]. Proceedings of the 18th IEEE International Conference on Automated Software Engineering[C]. New York, USA: ACM, 2003. 30 – 39.

[9] Zhang Z, Chan W K, et al. Capturing propagation of infected program states[A]. Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering[C]. New York, USA, 2009. 43 – 52.

作者简介



曹鹤玲 女,1980 年 5 月出生于河南南阳. 河南工业大学信息科学与工程学院教师,CCF 会员. 主要研究领域为软件分析与测试、数据挖掘.
E-mail: caohl@haut.edu.cn

姜淑娟(通信作者) 女,1966 年 12 月出生于山东莱阳. 现为中国矿业大学计算机科学与技术学院教授、博士生导师,CCF 会员. 主要研究领域为编译技术、软件工程等.
E-mail: shjjiang@cumt.edu.cn