第 69 卷 第 2 期
2023 年 4 月

武汉大学学报(理学版)
J. Wuhan Univ. (Nat. Sci. Ed.)

Vol. 69 No. 2
Apr. 2023,243~248

DOI: 10.14188/j.1671-8836.2022-0207

一种基于差分分析的负载相关的软件老化检测方法

谢文智[1,2],徐 斌[1],贾 凯[1,2],赵冬冬[1,2],田 璟[1],向剑文[1,2†]

1. 武汉理工大学 计算机与人工智能学院 交通物联网技术湖北省重点实验室,湖北 武汉 430070;
2. 武汉理工大学 重庆研究院,重庆 401135

收稿日期:2022-09-30 † 通信联系人 E-mail:jwxiang@whut.edu.cn
基金项目:国家自然科学基金(61672398);重庆市自然科学基金(cstc2021jcyj-msxmX1146);湖北省重点研发项目(2022BAA050)
作者简介:谢文智,男,硕士生,现从事软件老化研究。E-mail:xiewenzhi@whut.edu.cn

摘 要:传统的基于趋势分析的老化检测方法可能存在较高的误报。虽然多版本检测方法将先前的健壮版本作为基准版本与待测软件版本进行差分分析,但基准版本是否存在软件老化问题被忽略。为了解决以上问题,提出了一种基于差分分析的负载相关方法来检测软件老化。该方法对待测软件施加不同强度的负载,通过监控不同负载下内存资源消耗(驻留集大小(resident set size,RSS))趋势的差异并分析其与负载差异之间的关系,帮助开发人员在软件开发过程中检测单个版本(无需先验知识)的软件老化。实验结果表明,本文的方法能够检测出由内存泄漏引发软件老化,并且能够应用到真实的商业软件Squid中。

关 键 词:软件老化;老化检测;内存泄漏

中图分类号:TP311.53 **文献标志码**:A **文章编号**:1671-8836(2023)02-0243-06

A Workload-Related Software Aging Detection Method Based on Differential Analysis

XIE Wenzhi[1,2], XU Bin[1], JIA Kai[1,2], ZHAO Dongdong[1,2], TIAN Jing[1], XIANG Jianwen[1,2†]

1. Hubei Key Laboratory of Transportation of Internet of Things, School of Computer Science and Artifical Intelligence, Wuhan University of Technology, Wuhan 430070, Hubei, China;
2. Chongqing Research Institute of Wuhan University of Technology, Chongqing 401135, China

Abstract: The traditional aging detection methods based on trend analysis may have high false alarms. In addition, the multi-version detection methods have been proposed to differentially analyze the previous robust version as the baseline version from the software version to be tested, the presence of software aging in the baseline version is ignored. To address the above issues, a workload-related software aging detection method based on differential analysis is proposed to detect software aging. The method applies different intensities of load to the software under test. It helps developers to detect software aging of a single version (without a priori knowledge) during the software development process by monitoring the difference of memory resource consumption (resident set size, RSS) trend and analyzing its relationship with the load difference. The results show that the proposed method is capable of detecting software aging caused by memory leaks, and can apply our method to existing commercial software Squid.

Key words: software aging; aging detection; memory leak

0 引 言

软件老化是指在长时间运行的系统中,残留的老化缺陷(aging related bugs,ARBs)被激活导致的性能下降或崩溃故障[1]。软件老化现象已经在许多系统中观测到,例如:Linux系统[2]、安卓操作系统[3]、Web服务[4]、边缘计算[5]和TensorFlow[6]等。ARBs主要包括内存泄漏、未释放的文件锁、存储问题、套

引用格式:谢文智,徐斌,贾凯,等. 一种基于差分分析的负载相关的软件老化检测方法[J]. 武汉大学学报(理学版),2023,69(2):243-248. DOI:10.14188/j.1671-8836.2022-0207.

XIE Wenzhi,XU Bin,JIA Kai,*et al*. A Workload-Related Software Aging Detection Method Based on Differential Analysis [J]. *J Wuhan Univ* (*Nat Sci Ed*),2023,69(2):243-248. DOI:10.14188/j.1671-8836.2022-0207(Ch).

接字异常、未释放的文件句柄和错误的积累[7]。其中内存泄漏是导致软件老化的重要原因之一。内存泄漏主要由应用程序内存管理使用中的软件故障引起的,它会导致长时间不断运行的系统中内存资源耗尽。Macêdo等[8]介绍了内存相关软件老化的机制。Andrzejak等[9]在C/C++中收集了分配的内存片段、大小和它们的生命周期,并使用机器学习分类器来检测C/C++代码中的内存泄漏。Ghanavati等[10]研究了Java中内存泄漏的机制,当应用程序不再使用对象但垃圾回收器(garbage collector, GC)无法将它们从工作内存中删除时,就会发生内存泄漏,从而导致应用程序消耗越来越多的资源。内存泄漏是研究最多的软件老化问题,因为它会影响系统可用性,并且在实际系统中经常出现,因此本文将侧重研究检测由内存泄漏引发的软件老化。

现阶段软件老化最常用的两种检测方法是:基于趋势分析的软件老化检测方法和基于多版本比较的软件老化检测方法。基于趋势分析的老化检测方法是给待测软件施加一定的压力,并且收集对应时间序列的资源消耗,通过趋势分析可以得出该待测软件的资源消耗是否存在单调的趋势,如果存在单调递增的趋势,则认为该待测软件存在软件老化[11]。Zheng等[12]提出了一种基于Hodrick-Prescott的非线性趋势分析来检测软件老化。Machida等[13]研究了用Mann-Kendall趋势分析来检测软件老化。但是,基于趋势分析的软件老化检测方法并不能确保内存泄漏的存在,因为进程堆的大小可能会增加,这不仅是因为存在内存泄漏,还因为应用程序的内存使用模式[14]。因此,基于趋势分析的软件老化检测方法存在高误报的问题。

为了解决基于趋势分析检测不准确的问题,Langner等[15]首次提出基于比较的方法(多版本检测方法),他们希望通过比较相关开发版本之间的运行时的资源消耗来发现软件老化问题。Langner等[16]和Ghanavati等[17]在开发过程中提出了一种自动检测内存泄漏的方法,比较了同一软件相关开发版本的内存分配行为。Matias等[14]研究了内存泄漏的机制,并将软件差分分析与统计技术相结合来检测内存泄漏。基于差分分析的多版本比较方法一个前提就是需要该软件的健壮版本作为基准版本,然后跟待测版本的资源消耗进行差分分析,做出判断。但是基于多版本比较的软件老化检测方法的局限性在于:1)无法在没有先验知识(该软件之前的健壮版本)的情况下检测到软件老化;2)即使有该软件的先前版本,但不能确保该软件的先前版本不存在软件老化问题。比如在文献[17]中,Thrift-0.4被视无泄漏软件。但在Apache错误报告中[18]显示它存在内存泄漏问题,该bug的描述为:所有测试都在同一个Java虚拟机中运行,似乎TestAsymcClientManager正在泄漏套接字。如果不能保证软件的基准版本不存在软件老化的前提,最后得出的检测结果是不鲁棒的。因此本文提出:在单版本上基于差分分析的负载相关方法对软件老化进行检测。

这一想法是基于医学领域上,对于某些病人的检测是通过压力测试,正常人与病人在不同的压力下表现是不相同的。在不同压力下,病人的某些指标的差异会特别大,但正常人的某些指标差异则无明显差异[19]。因此,通过监控单个版本软件在不同负载下的资源消耗,存在老化问题的软件在高负载下资源消耗的增长速率比低负载下资源消耗的增长速率快,并且呈比例地增长,该比例接近高低负载的比值。本文的贡献在于:

1)降低了基于趋势分析的软件老化检测方法高误报的问题;

2)可对单版本软件老化进行检测,无需该软件先前的健壮版本作为基准版本。

1 本文方法

本文的方法流程如图1所示。使用负载生成器对单个版本软件施加不同的负载,以不同的时间间隔运行待测软件,时间间隔短的为高负载,时间间隔长的低负载。每隔一定的时间间隔,分别收集不同负载下软件的RSS(resident set size)使用情况。在不同的负载下,存在内存泄漏的软件与不存在内存泄漏的软件的行为不同:存在内存泄漏的软件在高负载下内存的积累比低负载下积累得多,所以在不同负载下资源的消耗差异很大;不存在内存泄漏的软件因为不存在内存的积累,所以在不同负载下资源消耗的差距会很小。用差异函数$R=(\mathrm{RSS}_{\mathrm{high}},\mathrm{RSS}_{\mathrm{low}})$量化差异变化,high和low分别代表高负载和低负载,$\mathrm{RSS}_{\mathrm{orignal}}$表示在最开始时收集的数据。$\mathrm{RSS}_{\mathrm{high}}$和$\mathrm{RSS}_{\mathrm{low}}$表示的是同一时间点不同负载下收集的数据。定义的差异函数R如下:

$$R=\frac{\mathrm{RSS}_{\mathrm{high}}-\mathrm{RSS}_{\mathrm{orignal}}}{\mathrm{RSS}_{\mathrm{low}}-\mathrm{RSS}_{\mathrm{orignal}}} \tag{1}$$

在同一时间点收集不同负载下软件的资源消耗,用当前时间点的资源消耗减去初始时间点的资源消耗,得到从开始时间点到当前时间点的内存积

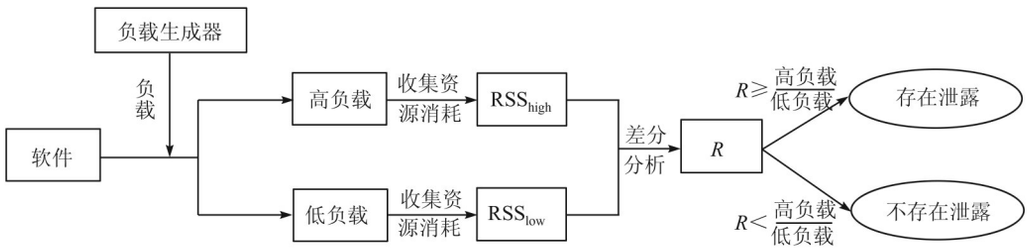


图1　方法概述

Fig. 1　Overview of our approach

累。R表示的是在不同负载下软件内存积累的比值,是对同一软件的不同负载下的资源消耗做差分分析。如果不同负载下每个时间点计算的R值(即软件内存积累的比值)接近或高于高低负载的比值则认为该软件存在由内存泄漏引发的软件老化。

文献[14]中,作者对软件的不同版本做差分分析,该方法的前提是需要将不存在软件老化的版本作为基准版本,使用基准版本跟待测版本的资源消耗进行比较。但是在现实中,大部分的待测软件都是新项目,并没有历史版本,或者不能保证历史版本不存在软件老化。本文的方法只需要软件的单个版本,对不同负载下软件的资源消耗进行差分分析,更适应于真实的场景。

2　实验部分

2.1　实验环境

实验平台是由一台计算机(主机)托管两台虚拟机(VM1和VM2)组成的。其中,主机配置为:Intel(R) Core(TM) i5-7300HQ CPU @ 2.50 GHz 2.50 GHz, 8.00 GB RAM,1 TB磁盘,Windows10;VM1的配置为:1个CUP内核,2.0 GB RAM,50 GB磁盘,CentOS-7;VM2的配置同VM1。

每个实验进行10次,以减少实验的偶然性误差。

2.2　老化指标的选取

特定于应用程序的指标常用来检测正在运行程序中的内存泄漏,这些度量指标有RSS、虚拟内存大小、堆大小和堆使用情况等。文献[14]将不同的指标进行比较认为RSS更适合检测软件老化。因此,本文将RSS作为老化指标,通过编写Shell脚本定时读取文件/proc/PID中status文件中的VmRSS字段内容来收集RSS使用情况。

2.3　调查应用程序(Squid)

Squid是一个Web的缓存代理,可以支持FTP、HTTP和DNS等协议的解析。它位于客户端和服务器之间,在不同的协议中发挥重要作用[20]。Squid的主要任务是当客户端访问页面时,会通过Squid获取这些数据,并且将这些数据缓存在Squid的cache中,当客户端下次访问该页面时,可以快速读取数据。它是一个比较成熟的软件项目,已经被很多互联网服务提供商和网站使用,比如Wikipedia(维基解密),它还被嵌入到许多网络设备产品中。因此,Squid的可靠性很重要[21]。本文的实验使用了两种不同版本的Squid,分别是存在泄漏的版本Squid-3.2.1和没有泄漏的健壮版本Squid-3.1.23。

2.4　实验分析

2.4.1　Java demo分析

为了找到负载和资源消耗之间的关系,将内存泄漏注入到Java程序中。Java中的内存泄漏机制如图2所示。当一个对象被不正确地引用时会发生内存泄漏,例如:当引用的对象未被使用。因此,我们编写了两个Java程序,一个是遭受了内存泄漏,泄漏内存约为250 Kb,另一个是不存在内存泄漏。采用作者编写的Java Demo可以手动注入泄漏并且知道每次内存泄漏的大小,每运行一次该代码就会有250 Kb的内存没有得到回收,可以更清晰地分析出不同负载下资源消耗的增长比值跟高低负载比值之间的关系。

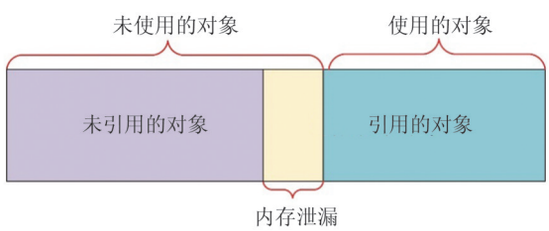


图2　Java中的内存泄漏机制

Fig. 2　The mechanism of memory leak in Java

采用两种不同的负载。高负载:每6 s运行一次代码;低负载:每30 s运行一次代码。

负载差异为5倍,Java程序一共运行2 h10 min,先运行程序10 min达到稳定状态,然后每个程序在不同的负载下执行2 h,每5 s收集一次程序的RSS

使用情况,共收集 1 440 个数据。Java 程序的 RSS 使用情况趋势如图 3 和图 4 所示。

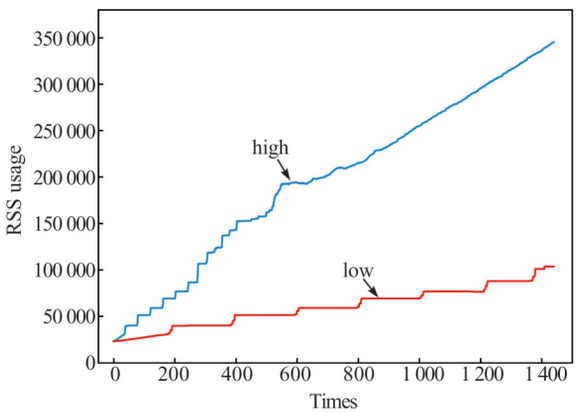


图 3 不同负载下存在泄漏程序的 RSS 使用情况

Fig. 3 The RSS usage of leaky procedure under different workload

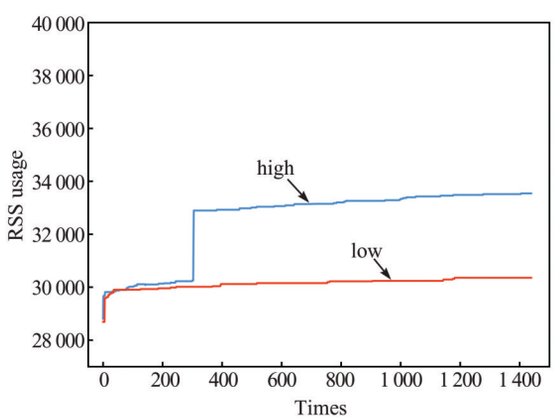


图 4 不同负载下无泄漏程序的 RSS 使用情况

Fig. 4 The RSS usage of no leaky procedure under different workload

通过观察图 3 可知,存在内存泄漏的程序在不同的负载下 RSS 使用情况差异非常大,但在图 4 中,无内存泄漏程序在不同负载下 RSS 使用情况差异很小。为了量化不同负载下的差异,对单个软件的版本在不同负载下的 RSS 使用情况进行差分分析,计算每个时间点的 R,并绘制出如图 5 所示的箱形图。

在图 5 中,箱形图中绿色的三角形代表的是平均值,箱子里的横线代表的是中位数,箱形外面的黑色圆圈代表的是异常值,存在内存泄漏的 Java demo 的 R 平均值接近 5,而无内存泄漏的 Java demo 的 R 远离 5。R 越大表示高负载和低负载下程序内存积累的差距也越大。因此可以得出结论:如果软件存在内存泄漏,则差值 R 接近于或者高于高低负载的比值;软件不存在内存泄漏,则差值 R 远低于高低负载的比值。

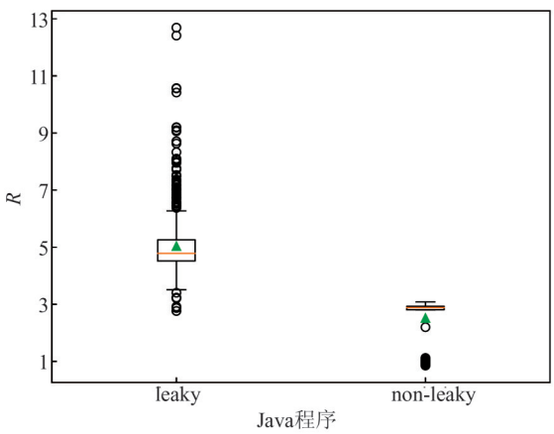


图 5 两个 Java 程序的差异 R 值

Fig. 5 The difference R of the two Java procedure

2.4.2 Squid 分析

监测 Squid 两个不同版本 Squid-3.1.23(健壮)和 Squid-3.2.1(泄漏)。在文献[21]中,作者发现当 Squid-3.2.1 在解析 FTP 协议时发生内存泄漏。本文在主机上构建了两台虚拟机 VM1 和 VM2。在 VM1 中的 Firxfox-52.2.0 浏览器上设置 Squid 缓存代理,并编写 Shell 脚本作为负载生成器控制 VM1 中的负载。在 VM2 中,安装软件 Squid 并监控 Squid 进程的 RSS 使用情况。实验平台设置如图 6 所示。实验采取高负载和低负载两种不同负载。其中,高负载:每 10 s 对 FTP url 进行一次访问;低负载:每 50 s 对 FTP url 进行一次访问。

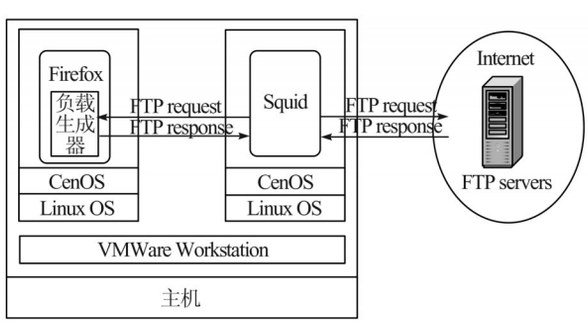


图 6 实验平台设置

Fig. 6 Testbed setup

使用 FTP url 是 *ftp://ftp.adobe.com*,负载相差 5 倍,每次实验持续 30 min,前 5 min 使运行的软件达到稳定的状态,每 10 s 收集一次数据,共收集了 300 个数据。实验结果如图 7 和图 8 所示。

从图 7 和图 8 可以得出与 Java demo 实验相同的结果。如图 7 所示,Squid-3.2.1 在不同负载下 RSS 的使用情况差距很大。在图 8 中,Squid-3.1.23 在不同负载下的差异很小。对不同负载下单个软件版本的 RSS 使用情况进行差分分析,计算每个点的 R 并制作出如图 9 所示的箱形图。

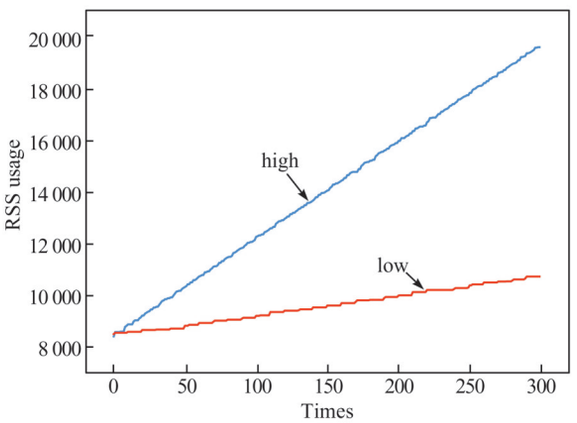


图7　Squid-3.2.1在不同负载下的RSS使用情况

Fig. 7　The RSS usage of Squid-3. 2. 1 under different workload

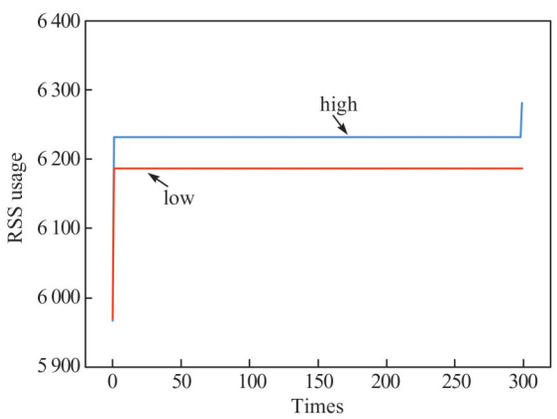


图8　Squid-3.1.23在不同负载下的RSS使用情况

Fig. 8　The RSS usage of Squid-3. 1. 23 under different workload

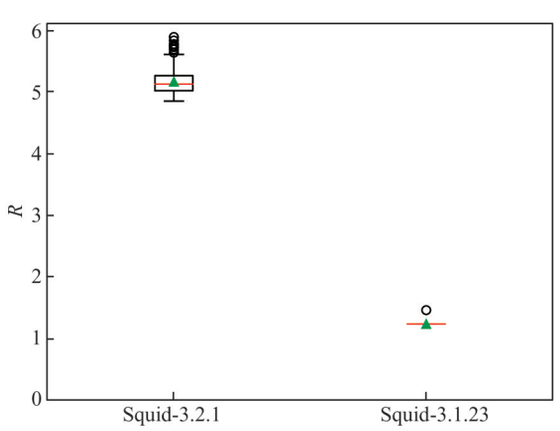


图9　Squid-3.2.1和Squid-3.1.23的差异值R

Fig. 9　The difference R of Squid-3. 2. 1 and Squid-3. 1. 23

从图9中可以发现,在Squid-3. 2. 1中,R的平均值接近5,$R=5$表示高负载下软件的资源消耗是低负载下的5倍,内存在不断积累,存在软件老化。而在Squid-3. 1. 23中R的值大多分布在1左右,$R=1$表示软件在不同负载下,资源消耗是一样的,不存在软件老化。在商业软件Squid中,如果软件存在泄漏,则差值R接近于高低负载的比值。

3　结　语

本文提出了一种基于差分分析负载相关方法,用于软件开发过程的早期检测软件老化(内存泄漏相关缺陷),并将它应用于成熟的商业软件Squid。提出的方法侧重于负载和资源消耗之间的关系。实验表明,该方法可以有效地检测开发阶段的软件老化问题。与基于趋势检测的方法相比[13],该方法可以有效的检测到软件老化;与多版本检测方法相比[14],该方法可以在单个版本上检测到软件老化。提出的方法不需要先验知识,避免了对先前健壮版本的错误选择导致检测误报。

本文所做的工作存在一定的局限性:1) 没有充分考虑负载和资源消耗之间的关系是否与其他条件有关;2) 只使用RSS作为老化指标,其他系统指标没有考虑;3) 仅讨论了该方法在Squid中的应用情形。在一些复杂的系统中,存在许多干扰因素,尚不清楚本文方法是否适用于其他软件。

在未来的工作中,我们计划将提出的方法应用到更多的系统,考虑更多的老化指标而不局限于单个指标RSS,并考虑这些指标间的相关性。此外,还将考虑采用更全面的数学模型进行建模。

参考文献:

[1] HUANG Y, KINTALA C, KOLETTIS N, *et al*. Software rejuvenation: Analysis, module and applications [C]//*25th International Symposium on Fault-Tolerant Computing*. New York: IEEE, 1995: 381-390. DOI: 10.1109/FTCS.1995.466961.

[2] COTRONEO D, NATELLA R, PIETRANTUONO R, *et al*. Software aging analysis of the linux operating system [C]//2010 *IEEE 21st International Symposium on Software Reliability Engineering*. New York: IEEE Press, 2010: 71-80. DOI: 10.1109/ISSRE.2010.24.

[3] COTRONEO D, DE SIMONE L, NATELLA R, *et al*. A configurable software aging detection and rejuvenation agent for android [C]//2019 *IEEE International Symposium on Software Reliability Engineering Workshops* (*ISSREW*). New York: IEEE Press, 2020: 239-245. DOI: 10.1109/ISSREW.2019.00078.

[4] PARRI J, SAMPIETRO S, SCOMMEGNA L, *et al*. Evaluation of software aging in component-based Web Applications subject to soft errors over time [C]//2021 *IEEE International Symposium on Software Reliability Engineering Workshops* (*ISSREW*). New York: IEEE

Press, 2022: 25-32. DOI: 10.1109/ISSREW53611.2021.00040.

[5] BAI J, CHANG X L, MACHIDA F, *et al*. Impact of service function aging on the dependability for MEC service function chain [J]. *IEEE* (99) *Transactions on Dependable and Secure Computing*, 2022: 1. DOI: 10.1109/TDSC.2022.3150782.

[6] DU X T, XIAO G P, SUI Y L. Fault triggers in the TensorFlow framework: An experience report[C]//2020 *IEEE* 31*st International Symposium on Software Reliability Engineering* (*ISSRE*). New York: IEEE Press, 2020: 1-12. DOI: 10.1109/ISSRE5003.2020.00010.

[7] 徐斌. 跨项目软件老化缺陷预测[D]. 武汉: 武汉理工大学, 2021.
XU B. Cross-*Project Aging-Related Bug Prediction*[D]. Wuhan: Wuhan University of Technology, 2021(Ch).

[8] MACÊDO A, FERREIRA T B, MATIAS R. The mechanics of memory-related software aging [C]//2010 *IEEE Second International Workshop on Software Aging and Rejuvenation*. New York: IEEE Press, 2011: 1-5. DOI: 10.1109/WOSAR.2010.5722097.

[9] ANDRZEJAK A, EICHLER F, GHANAVATI M. Detection of memory leaks in C/C++ code via machine learning [C]//2017 *IEEE International Symposium on Software Reliability Engineering Workshops* (*ISSREW*). New York: IEEE Press, 2017: 252-258. DOI: 10.1109/ISSREW.2017.72.

[10] GHANAVATI M, COSTA D, SEBOEK J, *et al*. Memory and resource leak defects and their repairs in Java Projects[J]. *Empirical Software Engineering*, 2020, **25**(1): 678-718. DOI: 10.1007/s10664-019-09731-8.

[11] PIETRANTUONO R, ALONSO J, VAIDYANATHAN K. Measurements for software aging[M]//*Handbook of Software Aging and Rejuvenation*. Singapore: World Scientific Publishing Co Pte Ltd, 2020: 73-90. DOI: 10.1142/9789811214578_0004.

[12] ZHENG P F, XU Q G, QI Y. An advanced methodology for measuring and characterizing software aging [C]//2012 *IEEE* 23*rd International Symposium on Software Reliability Engineering Workshops*. New York: IEEE Press, 2013: 253-258. DOI: 10.1109/ISSREW.2012.81.

[13] MACHIDA F, ANDRZEJAK A, MATIAS R, *et al*. On the effectiveness of Mann-Kendall test for detection of software aging[C]//2013 *IEEE International Symposium on Software Reliability Engineering Workshops* (*ISSREW*). New York: IEEE Press, 2013: 269-274. DOI: 10.1109/ISSREW.2013.6688905.

[14] MATIAS R, ANDRZEJAK A, MACHIDA F, *et al*. A systematic differential analysis for fast and robust detection of software aging[C]//2014 *IEEE* 33*rd International Symposium on Reliable Distributed Systems*. New York: IEEE Press, 2014: 311-320. DOI: 10.1109/SRDS.2014.38.

[15] LANGNER F, ANDRZEJAK A. Detecting software aging in a cloud computing framework by comparing development versions[C]//2013 *IFIP/IEEE International Symposium on Integrated Network Management* (*IM* 2013). New York: IEEE Press, 2013: 896-899.

[16] LANGNER F, ANDRZEJAK A. Detection and root cause analysis of memory-related software aging defects by automated tests[C]//*Proceedings of the* 2013 *IEEE* 21*st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*. New York: ACM, 2013: 365-369. DOI: 10.1109/MASCOTS.2013.53.

[17] GHANAVATI M, ANDRZEJAK A. Automated memory leak diagnosis by regression testing[C]//2015 *IEEE* 15*th International Working Conference on Source Code Analysis and Manipulation* (*SCAM*). New York: IEEE Press, 2015: 191-200. DOI: 10.1109/SCAM.2015.7335415.

[18] TODD L. THRIFT-873 [EB/OL]. [2010-08-27]. *https://issues.apache.org/jira/browse/THRIFT-873*.

[19] FULLER C M, RAIZNER A E, VERANI M S, *et al*. Early post-myocardial infarction treadmill stress testing: An accurate predictor of multivessel coronary disease and subsequent cardiac events[J]. *Annals of Internal Medicine*, 1981, **94**(6): 734-739. DOI: 10.7326/0003-4819-94-6-734.

[20] 王思家. 基于SPC的软件老化检测方法研究[D]. 哈尔滨: 哈尔滨工程大学, 2020.
WANG S J. *Research on Software Aging Detection Method Based on SPC*[D]. Harbin: Harbin Engineering University, 2020(Ch).

[21] MATIAS R, DE SENA G O, ANDRZEJAK A, *et al*. Software aging detection based on differential analysis: An experimental study[C]//2016 *IEEE International Symposium on Software Reliability Engineering Workshops* (*ISSREW*). New York: IEEE Press, 2016: 71-77. DOI: 10.1109/ISSREW.2016.44.

□