ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2000/00(00)-0000-00
doi: 10.3778/j.issn.1673-9418.

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-89056056

基于上下文的飞控软件老化缺陷定位方法研究*

杜晓婷[1+], 王楠[2]

1. 北京航空航天大学 自动化科学与电气工程学院, 北京市 100191 ,中国
2. 杜克大学 电子与计算机工程学院, 北卡罗莱那州 达勒姆市 27708 ,美国

Context Based Fault Localization Method for Flight Control Software Aging Defects*

DU Xiaoting[1+], WANG Nan[2]

1. School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China
2. School of Electrical and Computer Engineering, Duke University, North Carolina Durham 27708, USA
+ DU Xiaoting: E-mail: xiaoting_2015@buaa.edu.cn

DU Xiaoting, WANG Nan. Context Based Fault Localization Method for Flight Control Software Aging Defects. Journal of Frontiers of Computer Science and Technology, 2016, 0(0): 1-000.

Abstract: With the development of the aviation industry, the complexity of the flight control software as well as requirements of the reliability and the real-time performance become higher. In this paper, a special type of aging defects which will affect the reliability and the real-time performance of software systems is investigated, due to the reason that it will result in the slowly response. However, there is no effective method to address this type of defects for flight control software. Inspired by that, a context based fault localization method is proposed in this paper. First of all, the main loop of the flight control software is modelled as a task tree, then the repeated running processes of the main loop can be considered as a task tree series. By utilizing Mann-Kendall Test, the time attribute of the central node of the task tree is detected to find the suspicious task. Thereafter, according to the relationship among the task tree series, suspicious tasks are selected to locate defects. The function of suspicious task is the location of the defect. Finally, an experiment was implemented and results show that the proposed method can not only locate the defect, but also give the call context.

Key words: Flight control software; Aging defect; Fault localization; Context; Task tree

Received 2016-07, Accepted 2000-00.

摘 要: 随着飞机的功能越来越强大，飞控软件的复杂度越来越高，对飞控软件可靠性和实时性的要求也越来越高。研究了软件老化缺陷中一类特殊的缺陷，这类缺陷会使系统响应逐渐变慢，最终出现失效，严重影响飞控软件的可靠性和实时性，目前并没有有效的定位这类缺陷的方法。针对这类缺陷的定位，提出了一种基于上下文的定位方法。将飞控软件的主循环建模成一棵任务树，将飞控软件反复运行的主循环抽象成一个任务树序列。通过 Mann-Kendall Test 对这些任务树中心节点的时间属性进行趋势检测，找到可疑的任务，并根据这些任务树之间的关系对可疑任务进行筛选，定位含有缺陷的任务，该任务所对应的函数即为缺陷所在的位置。实验表明，该方法不仅能够给出缺陷的具体位置，还能给出发生失效时的调用上下文。

关键词: 飞控软件; 老化缺陷; 缺陷定位; 上下文; 任务树

文献标志码：A **中图分类号**：TP311

1 引言

由于特殊的工作环境，相对于一般软件而言，飞控软件对实时性和可靠性都有着更高的要求。然而存在一类缺陷，会使飞控软件响应速度逐渐变慢，最终出现失效，对飞行安全带来严重挑战，这类缺陷属于老化缺陷中的一种。

老化缺陷会随着时间表现出越来越显著的现象，比如内存升高，系统响应变慢等，即出现软件老化。在执行过程中遇到老化缺陷时，可能会导致不可预测的失效，这种失效会对飞控软件的正常运行造成严重影响，进而威胁飞行安全。为了增加飞控软件的可靠性，本文针对造成飞控系统响应变慢的老化缺陷，根据飞控软件运行过程特点，提出了基于上下文的老化缺陷定位方法。

飞控软件的主要运行过程是定时执行一个主循环，在主循环中，飞控系统读取外界传感器数据和驾驶指令，计算输出，并输出到执行机构，此外飞控系统会完成一些额外的工作，比如日志记录，通讯等。从广义的角度来看，每一次主循环都可以看作一个测试用例，输入是传感器信号和控制指令，输出是舵机角度，油门，写入的日志等。整个运行过程中反复运行的主循环，就是一连串测试用例输入的过程，我们将这个过程称为一个测试用例链。如果可以把飞控软件反复运行的主循环拆分成不同的子任务，然后在测试用例链运行过程中，检测这些子任务随着测试用例链输入而运行时间变长的情况，就可以利用这些信息来定位老化的源头，从而找到造成系统响应变慢的老化缺陷。

飞控软件中每一个任务的执行实际上对应于一个函数调用，所以某一个任务的执行时间逐渐变长，也就说明了该任务所对应的函数调用时间逐渐变长，也就是说，这个函数可能含有引起老化的缺陷。然而这个结论是不完全，因为任务的执行或者函数的调用存在嵌套关系。一个函数的调用时间出现老化，既有可能是该函数本身的实现有问题，也可能是这个函数所调用的函数的实现有问题。只有当我们观察到其子任务或子函数的执行时间都没老化现象的时候，我们才能确定是这个函数含有缺陷。另外，一个函数含有缺陷，并不代表每一次运行都会表现出来，缺陷的触发可能依赖程序运行的上下文，比如函数 A 的缺陷可能只有在 B→A→C 这样的调用顺序才会触发，而在D→A→C这样的情况下不会触发。捕捉这样的上下文显然有利于后续的调试工作。为了对此类造成飞控系统响应变慢的老化缺陷进行定位，并捕捉发生老化的上下文，本文建立了飞控软件运行过程的任务树模型。通过该模型，将飞控软件反复运行的主循环抽象成一个任务树序列。通过分析任务树内的任务之间的父子关系，以及这些关系给缺陷定位带来的影响，根据任务和函数运行之间的对应关系，提出了针对任务树序列来定位老化缺陷的方法。通过该缺陷定位方法，我们不仅可以找到发

生老化的任务，还可以找到老化发生时的函数调用上下文，对后续的调试过程有非常积极的意义。

文章其余部分结构如下：第二部分介绍相关研究，第三部分对飞控软件的运行过程主循环进行建模；第四部分介绍基于任务上下文的飞控软件老化缺陷定位方法的基本思想和方法流程；第五部分通过实验，验证本文方法的有效性；最后一部分是结束语。

2 相关研究

1994 年 David 在[1]中首次提出了软件老化的概念，Huang 等人在[2]中对老化缺陷进行了具体描述：软件老化是由内存膨胀和泄漏，未释放的文件锁，数据损坏，存储空间碎片化和舍入误差的积累等原因造成的。在软件老化中，错误随着时间积累，导致系统性能下降或出现失效。目前解决软件老化问题的方法可以分为两类，一类是 reactive 方法[3]，这种方法是在失效发生之后，采取一定的恢复机制尝试修复失效，恢复率较高。但这种方法无法保证在每次出现失效后，始终能够提供一种令人满意的恢复机制来保证软件的长期运行。

另一类 proactive 方法与 reactive 方法不同，这是一种软件恢复的方法[2-4]，相比 reactive 方法效果更好，应用也更广泛。软件恢复的主要步骤包括终止程序运行，清理程序的内部状态和环境，最后重新启动该程序。软件恢复本质上通过对软件进行恢复，预防软件的老化。由于软件恢复需要很高的代价，如何寻找最优软件恢复时间和最优软件恢复策略成为研究的重点，这些工作可以分为两类：基于模型的分析方法和基于测量的分析方法。基于模型的分析方法建立系统的数学模型[5-9]，通过这些模型确定软件恢复的有效性并得到最优恢复方案，模型的精确度取决于捕获老化信息时所做的假设。[5]中使用 MRSPN（马尔科夫再生随机 Petri 网）模型来描述系统的行为，并计算出了软件可靠性最大情况下的最优恢复间隔。基于模型的恢复策略，一般假设太过理想，在实际的工程领域中不容易验证。

基于测量的方法通过监控软件系统的操作状况[9-14]，收集关键性能数据，以此来分析软件的老化程度。[9]中给出了一种检测和估计 UNIX 操作系统老化的方法，使用基于 SNMP（简单网络管理协议）的分布式检测工具收集系统资源的使用数据和系统活动数据，应用趋势检测技术对这些数据进行处理，检测老化的存在，并使用坡度估计方法计算“资源耗尽时间”。[12]中提出了一种基于时间序列的分析方法，检测和估计一个网络服务器在软件老化的影响下，系统资源的耗尽时间，并用线性回归方法对数据进行趋势检测。

基于目前提出的各种软件恢复方法，[15]中对各种恢复策略的代价进行了估计，分析了这些方法对消除软件老化影响的效果，提出选择软件恢复策略时应合理平衡恢复效果和使用代价二者的关系，为选择合适的恢复策略提供指导。

上述这些方法都只是暂时的延缓了失效时间，并没有从根本上解决软件中存在的缺陷，老化缺陷对飞控系统的实时性和可靠性仍然存在很大威胁。如何定位老化缺陷，从而为修复缺陷提供依据，是本文要研究的问题。

由于传统软件缺陷定位技术依赖于对失败测试用例的分析[16-19]，而对于老化缺陷而言，失效难以复现，因此失败测试用例非常少，严重影响了现有的缺陷定位方法的效果。本文针对一类造成飞控软件响应变慢的老化缺陷，结合飞控软件运行过程的特点，设计了一种该类缺陷的定位方法，将在第三部分和第四部分进行详细阐述。

3 飞控软件运行主循环任务树模型

本节中首先对飞控软件的运行过程进行描述，然后提出缺陷定位的任务树模型，并对该模型进行分析。

因为飞控软件对运行过程的可预测性要求很高，所以一般不会采用抢占式多任务调度器。飞控软件的主循环调度过程如下：在主循环开始时，先运行与飞行器控制相关的关键任务；当关键任务运

行完后，如果还有时间，将按照预先设定的优先级调度需要 CPU 时间的任务；在运行某一任务之前，先判断该任务的运行时间上限是否已经超过了主循环还剩下的时间，如果没超过，那么正常运行，如果超过了，在本次主循环中就跳过该任务。可以看出，在该调度策略下，不同任务的运行时间并不会有交叠。另外，一个任务通常包含子任务，比如，更新飞行控制状态的任务包含读取 GPS，读取陀螺仪等子任务。于是，我们可以把一次主循环用一棵任务树来表示。

下面给出任务树及其相关定义：

定义 1(任务集 任务树 任务树序列) *将所要讨论的所有任务组成一个集合，叫任务集。如果选取任务集的一个子集，用这些任务构成一棵树，并给每个节点赋上一个时间属性，这样一棵树叫做任务树。如图 1 所示，任务树的根节点为飞控软件运行的主循环，其余每个节点代表一个任务，节点的第一行是任务的名字，第二行是任务的执行时间，即该任务本次的消耗。节点之间的连线表示任务的分解，除叶子节点外，每个节点还包括一个或多个子节点，表示每个任务的完成都是通过一系列子任务来实现的。*

在飞控软件中，飞控主循环的每一次运行都对应着这样一棵任务树。飞控软件的运行过程中反复运行的主循环就构成一个任务树序列。

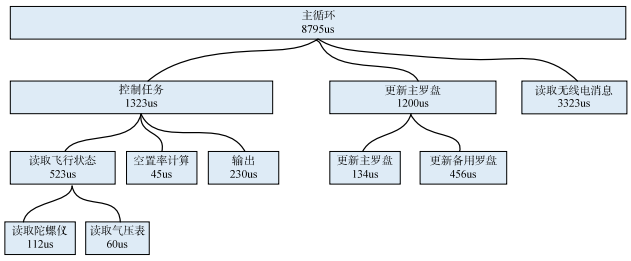


Fig.1 Example of the task tree

图 1 任务树的例子

任务树还满足一定的性质：

(1) 父任务的执行时间总是大于等于其所有子任务的总和。所以当某子任务的执行时间变长时，有可能使其所有直系先辈节点的执行时间变长。

(2) 不同时间段上的主循环所对应的任务树并不相同，这是因为并不是所有的飞行任务在每次主循环都会执行。有些不紧急的任务周期会很长，可能长达几秒，因而被主循环执行的频率会很低；而有些紧急任务的周期可能只有几毫秒，但却会被主循环频繁执行。

定义 2(上下文树 完全上下文树) *关于任务 t 的上下文树是一棵包含 t 的任务树 T，并且在该任务树中，t 只有直系的祖先节点，且 t 叫做任务树 T 的中心节点。关于任务树 T 和该任务树中的一个任务 t 的完全上下文树$Context_T(t)$是任务树的一棵子树，该子树是 t 的上下文树，并且包含 t 的所有后代节点和 T 的根节点。*

当我们说一个完全上下文树发生老化时，表示在一个任务树序列中，该完全上下文树的中心节点的时间属性呈现上升趋势。

基于上文的描述和定义，所考虑的缺陷定位问题可以这样描述：某一任务 t 含有执行时间老化的缺陷，我们把它叫做目标任务。在某一任务树序列中，t 的执行时间仅在某一个上下文树出现时才可能发生老化，我们把它叫做目标上下文树。缺陷定位问题就转化为寻找这样一棵目标上下文树的问题，也是第四部分所要解决的问题。

4 基于任务上下文的飞控软件老化缺陷定位方法

本节首先介绍缺陷定位方法的基本步骤，并通过一个例子对该方法进行分析，最后对本文中使用的 Mann-Kendall 趋势检测方法进行具体描述。

在飞控软件中，每一个任务的执行对应于一个函数调用，当某一个任务的执行时间逐渐变长，也就说明了该任务所对应的函数调用时间逐渐变长。因为任务的执行或者函数的调用存在嵌套关系，一个函数的调用时间出现老化，既有可能是该函数本身的实现有问题，也可能是这个函数所调用的函数的实现有问题。只有当到其子任务或子函数的执行时间都没老化现象的时候，才能确定是这个函数含有缺陷。另一方面，一个函数含有缺陷，并不代表

每一次运行都会表现出来，缺陷的触发可能依赖程序运行的上下文。为了给出缺陷的位置并捕捉缺陷触发的上下文，我们利用上一节给出的任务树模型对飞控软件的老化缺陷进行定位。如图 2 所示，缺陷定位方法共分为六个步骤，下面对这六个步骤进行具体描述。

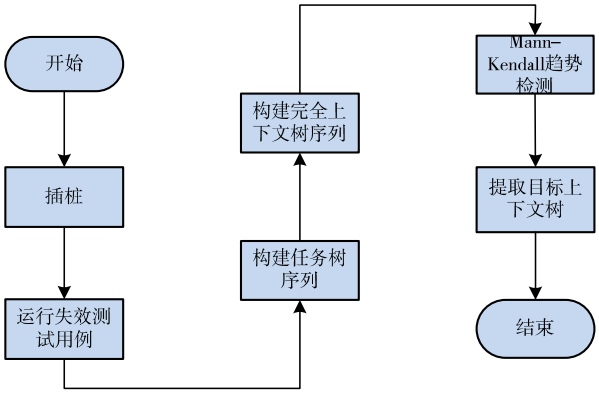


Fig.2 Procedures of the context based fault localization method

图 2 基于任务上下文的缺陷定位方法步骤

（1）对飞控软件进行插桩，并运行能够引起老化的失效测试用例。

在飞控软件的源码中插入记录语句，记录函数的进入时间和退出时间，以便计算函数的运行时间，以及生成函数之间的调用关系，这些信息被写入日志文件中。由于我们只关心在主循环中能够被调用到的函数，所以，只有这一部分函数需要被插桩。通过运行能够引起老化的失效测试用例，我们可以得到一个记录了所有被插桩函数进入时间和退出时间的文件。

（2）构建任务树序列和完全上下文树序列。

在记录文件中，我们可以找到每一次主循环的开始和结束时间，对于每一次主循环的运行，我们可以根据各个函数的进入和退出时间，确定它们之间的调用关系，由此生成一棵任务树，如图 3 所示。其根节点是 main 函数，其余每个节点都表示一个函数，从上到下的连线表示函数之间的调用关系。对于每一次主循环的运行，我们都生成这样一棵任务树，于是得到一个任务树序列。对于任务树序列中的每一棵任务树，选择其任意一个节点生成一棵完全上下文树，并把相同的完全上下文树归并到一个集合里面。然后根据它们根节点的进入时间进行排序，这样就得到了若干完全上下文树序列，其中每个序列中的完全上下文树是相同的，不同序列中的完全上下文树是不同的。

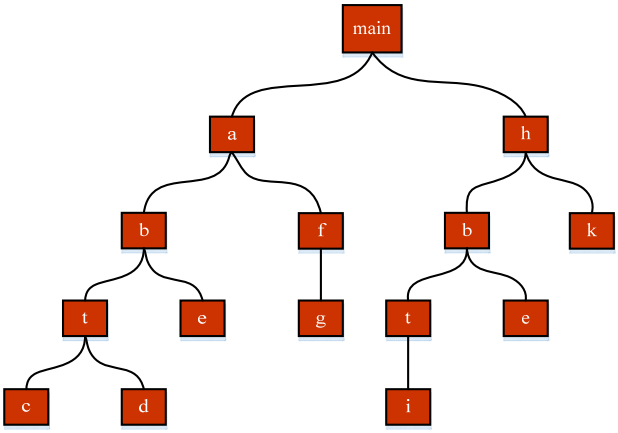


Fig.3 Task tree

图 3 任务树

（3）对完全上下文树的中心节点进行趋势检测

对于每一个完全上下文树序列，我们将其中心节点的运行时间提取出来，组成一个运行时间序列。接下来，对每一个运行时间序列进行趋势检测，图 4 所示为出现老化现象的完全上下文树的一个例子，图中的浅色节点是这些上下文树的中心节点，也是被检测出运行时间存在上升趋势的点。那么就可以找到存在上升趋势的时间序列所对应的完全上下文树，这些上下文树就是于老化相关的完全上下文树。本文使用 Mann-Kendall 趋势检测方法，会在下一小节进行详细论述。

（4）提取目标上下文树

上一步中，我们得到了若干与老化相关的完全上下文树。接下来，我们需要识别出这些完全上下文树的包含关系，对这些完全上下文树进行筛选，并提取完全上下文树的公共部分，得到目标上下文树。当一棵完全上下文树包含另一棵完全上下文树时，前者的中心节点所表现出来的老化现象很可能是受到后者的影响，因此，我们应当将前者去掉。

得到已经去掉含有子树的完全上下文树的集合后，最后一步就是去掉这些完全上下文树的多余节点。在前面的步骤，我们考虑了每一个中心节点运行的全部上下文，然而缺陷的触发有可能只依赖部分的上下文，甚至根本不依赖上下文。这一步中，我们只需要提取出上一步获得的集合中所有树的公共部分，我们就能找到真正与缺陷相关的上下文，也就是我们需要寻找的目标上下文树。

如图 4 中，(3)树的中心节点的时间属性之所以呈上升趋势，是因为(3)的中心节点 a 调用了(2)的中心节点 b，因此，a 的老化现象很可能是 b 带来的，所以我们需要排除掉(3)树。同理，要把(2) (3) (5) (6)树都去掉。去掉这些完全上下文树后的集合如图 5 所示。对图 5 中的完全上下文树提取公共部分后，得到的目标上下文树如图 6 所示。于是，我们定位到了引起老化的缺陷所在的位置：任务 t 所对应的函数。不仅如此，我们还给出了这种缺陷触发所依赖的上下文：从 b 到 t 的调用。

从上述步骤可以看到，基于任务树的缺陷定位方法能够通过完全上下文树之间的包含关系，排除因为受其他任务影响而老化的任务，有效减少 false positive 的产生。另外，该方法还通过完全上下文树之间的共性，给出了与老化密切相关的调用上下文。

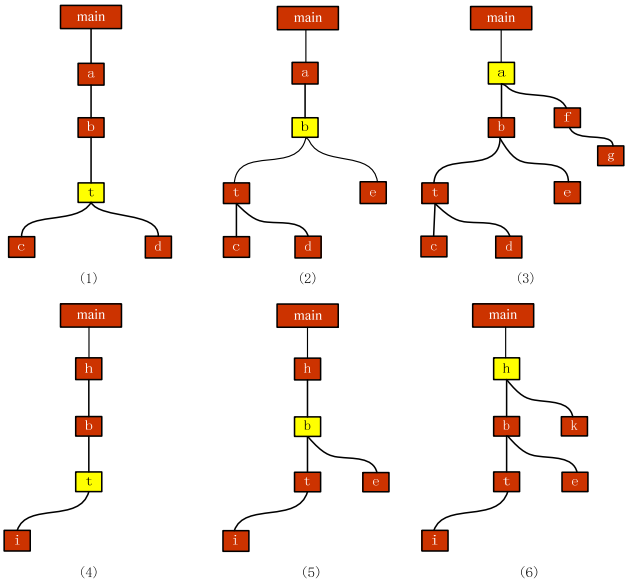


Fig.4 Sets of the aging full context trees

图 4 发生老化的完全上下文树集合

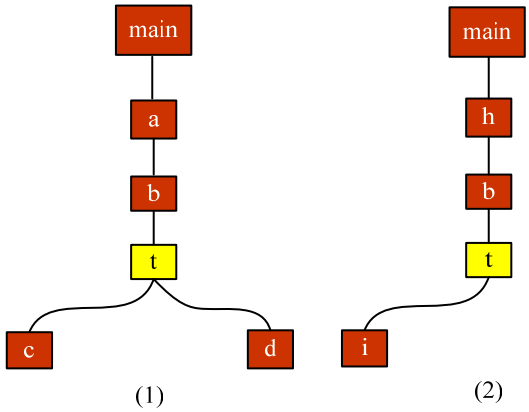


Fig.5 Sets of the aging full context trees after filtering the subtrees

图 5 去掉含有子树的完全上下文树的集合

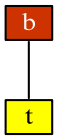


Fig.6 Goal context tree

图 6 目标上下文树

4.1 基于Mann-Kendal的老化趋势检测

Mann-Kendall 趋势检测方法[20]，是最广泛使用的非参数检验方法，不要求样本必须服从一定分布，且检验不受个别异常值的干扰，计算简便。本文中我们关注的是任务树中的某一节点的时间属性是否有递增的趋势。对于每一个完全上下文树序列，我们将其中心节点的运行时间（退出时间减去进入时间）提取出来，组成一个运行时间序列，通过 Mann-Kendall 对每一个运行时间序列进行趋势检测，筛选出被检测到具有上升趋势的运行时间序列。

设运行时间序列为$\{x_t: t = 1,2,3,\dots,n\}$，原假设$H_0$：时间序列$\{x_t\}$没有变化趋势；备择假设$H_1$：时间序列$\{x_t\}$有变化趋势。统计量$S$的计算如公式（1）所示:

$$S = \sum_{k=1}^{n-1}\sum_{l=k+1}^{n} sgn(x_l - x_k) \quad (1)$$

其中$l > k$，sgn 为符号函数。

如果时间序列$x_1, x_2, \dots, x_n$中不存在相同数值，S的方差为：

$$\mathrm{Var}(S)=\frac{1}{18}[\mathrm{n}(\mathrm{n}-1)(2\mathrm{n}+5)] \quad (2)$$

如果时间序列$x_1,x_2,\ldots,x_n$中存在m组相同数值，其中t_p为第p组对应的数值，S的方差为：

$$\mathrm{Var}(S)=\frac{1}{18}\left[\mathrm{n}(\mathrm{n}-1)(2\mathrm{n}+5)+\sum_{p=1}^{m}t_p\,(t_p-1)(2t_p+5)\right] \quad (3)$$

对S标准化，可得

$$Z=\frac{S-sgn(S)}{\sqrt{Var(S)}} \quad (4)$$

给定显著性水平α，若$|Z|>U_{1-\frac{\alpha}{2}}$，则拒绝$H_0$，表明时间序列$\{x_t\}$存在变化趋势，在此条件下，若$Z>0$，表明该时间序列有上升趋势，$Z<0$，表明时间序列$\{x_t\}$存在下降趋势；若$|Z|<U_{1-\frac{\alpha}{2}}$，则接受$H_0$，表明时间序列$\{x_t\}$不存在变化趋势。

5 实验分析

在这一节中，通过在开源飞控软件 Ardupilot 上人工植入老化缺陷，利用第四节提出的缺陷定位方法进行缺陷定位实验，验证算法的有效性。首先我们对实验对象及实验设计与实现过程进行描述，然后将收集到的实验数据通过 Mann-Kendall 进行趋势检测，最后对实验结果进行分析。

5.1 实验对象

Ardupilot 是一套无人机自动飞行控制系统，支持多旋翼飞行器，固定翼飞机和传统直升机。系统由固定翼/多旋翼载机，APM 飞行控制板，数传模块，电子罗盘，各类传感器和地面站组成。虽然不如载人飞行器的飞控软件复杂，但是它也有近 20 万行代码（不包括实时操作系统 Nuttx 的代码和底层驱动的代码），足够满足本次实验的要求。更重要的是，无论是 Ardupilot 还是客机的飞控软件，它们的主要运行过程都是一个反复执行的主循环，本文提出的缺陷定位方法正是基于飞控软件的这个特点。

5.2 实验设计与结果分析

首先，我们对 Ardupilot 进行了插桩，即：在函数入口处和出口处插入了语句记录函数的进入和退出，并且记录了响应的进入时间和退出时间。我们在调度器，硬件抽象层和主循环中这些经常运行的 66 个函数进行了插桩。接下来，在 Ardupilot 的 Copter::motors_output 函数中植入一个模拟老化的缺陷，即加入了一个随着运行次数增加的延迟。将修改好的程序写入了一个无人机中，然后飞行了该无人机 10 分钟。最后，将无人机的日志文件取出。依据该记录文件，可以计算函数的运行时间并确定它们的调用关系。根据这些信息，每一次 Ardupilot 主循环的运行都会可以建模成一棵任务树，图 7 和图 8 给出了这种任务树的两个例子。其中的节点对应着一个函数（或任务），其编号和具体函数的对应关系如表 1 所示。

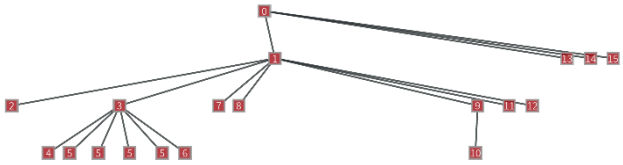


Fig.7 Example 1 of the main loop task tree

图 7 主循环任务树的样例 1

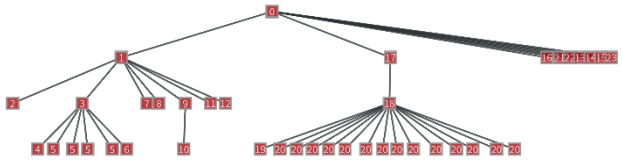


Fig.8 Example 2 of the main loop task tree

图 8 主循环任务树的样例 2

通过运行实验，收集任务树，提取完全上下树文序列，得到每一个完全上下文树序列中的中心节点的运行时间，组成一个运行时间序列。并使用 Mann-Kendall 趋势测试对所有的长度大于 100 的完全上下文树序列进行测试（长度太小难以进行趋势检测），其中 α 值取10^{-7}。

实验表明有两个完全上下文树中心节点的时间属性呈现上升趋势，结果如表 2 所示。从表中可以发现，一共有两个完全上下文树表现出了显著的老化，它们的中心节点分别是 Copter::fast_loop 和 Copter::motors_output，所对应的完全上下文树分别如图 9 和图 10。可以看到，中心节点为 Copter::fast_loop 的完全上下文树完全包含了中心

节点为 Copter::motors_output 的完全上下文树，因此，前者应该去掉，这是因为前者之所以发生老化是因为后者的影响。排除掉前者后，我们就找到了真正发生老化的任务或函数，即 Copter::motors_output，图 10 即为其调用的上下文。

最终的定位结果与我们植入的缺陷位置是一致的，因此说明了基于上下文树的缺陷定位方法的有效性。

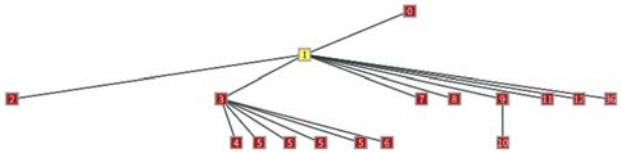


Fig.9 A full context tree with a central node Copter::fast_loop

图 9 中心节点为 Copter::fast_loop 的完全上下文树

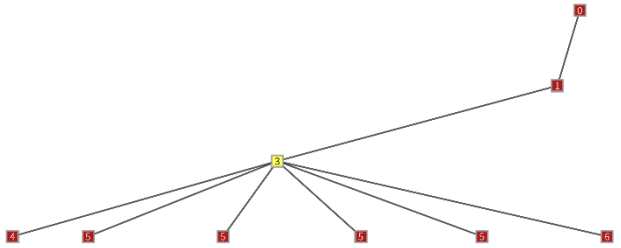


Fig.10 A full context tree with a central node Copter::motors_output

图 10 中心节点为 Copter::motors_output 的完全上下文树

6 结束语

本文提出了一种针对飞控软件中一类造成系统响应变慢的老化缺陷的定位方法。根据飞控软件的运行特点，建立了任务树模型。通过对飞控软件中部分函数进行插桩，并运行会使软件老化的测试用例，分析得到若干任务树序列和完全上下文树序列。使用 Mann-Kendall 趋势检测方法对完全上下文树序列的时间属性进行了趋势检测，证明老化的存在。最后通过识别包含关系筛选完全上下文树，去除完全上下文树中的多余节点，提取出了目标上下文树。此外，以 Ardupilot 为实验对象，本文设计并进行了实验。实验结果表明，本文中的方法不仅能定位出老化缺陷的位置，还能给出发生老化时的调用上下文，对后续的调试有很大帮助。

在对飞控软件缺陷定位方法的研究中，我们仅仅针对单缺陷问题进行了研究，没有分析多个老化缺陷存在时的缺陷定位方法。此外，本文中只分析了造成飞控软件响应变慢的一类缺陷。内存泄露缺陷与使飞控软件响应变慢的缺陷类似，这类缺陷的失效也有一个积累的过程，但它们并不一定会使系统响应变慢，因为这类缺陷的触发并不会像与响应时间有关的缺陷一样有一个向上传播的过程，因此无法使用本文所提出的缺陷定位方法。未来我们将针对上述两个问题做进一步研究。

Table1 Function names and their number

表 1 编号和函数名对应关系

编号	函数名	编号	函数名
0	main	1	Copter::fast_loop
2	Copter::read_AHRS	3	Copter::motors_output
4	AP_HAL::RCOutput::cork	5	AP_HAL::RCOutput::write
6	AP_HAL::RCOutput::push	7	Copter::read_inertia
8	Copter::check_ekf_yaw_reset	9	Copter::update_flight_mode
10	Copter::stabilize_run	11	Copter::update_home_from_EKF
12	Copter::update_land_and_crash_detectors	13	gcs_check_input
14	full_rate_logging_loop	15	dataflash_periodic
16	update_optical_flow	17	rc_loop
18	Copter::read_radio	19	AP_HAL::RCInput::new_input
20	AP_HAL::RCInput::read	21	update_thr_average
22	compass_accumulate	23	compass_cal_update
24	throttle_loop	25	update_GPS
26	AP_GPS::update	27	AP_HAL::UARTDriver::begin
28	AP_HAL::UARTDriver::begin	29	AP_HAL::UARTDriver::set_flow_control
30	run_nav_updates	31	barometer_accumulate
32	update_notify	33	AP_HAL::GPIO::usb_connected
34	gcs_send_deferred	35	gcs_data_stream_send
36	update_mount	37	fifty_hz_logging_loop
38	Copter::Log_Sensor_Health	39	AP_HAL::GPIO::write
40	rpm_update	41	auto_disarm_check

42	auto_trim	43	update_altitude
44	AP_Baro::update	45	ekf_check
46	landinggear_update	47	lost_vehicle_check
48	ten_hz_logging_loop	49	AP_HAL::RCOutput::read
50	read_receiver_rssi	51	epm_update
52	update_batt_compass	53	AP_BattMonitor::read
54	AP_HAL::AnalogIn::board_voltage	55	AP_HAL::AnalogIn::servorail_voltage
56	AP_HAL::AnalogIn::power_status_flags	57	read_aux_switches
58	arm_motors_check	59	frsky_telemetry_send
60	AP_HAL::RCInput::read	61	AP_HAL::RCInput::num_channels
62	three_hz_loop	63	one_hz_loop
64	AP_HAL::RCOutput::enable_ch	65	gcs_send_heartbeat

Table2 Results of the trend test

表 2 趋势测试结果

完全上下文树的中心节点函数名	趋势检测结果	P 值
Copter::read_AHRS	no trend	2.00561859831e-06
Copter::motors_output	increasing	0.0
AP_HAL::RCOutput::cork	no trend	0.57086923687
AP_HAL::RCOutput::write	no trend	0.382142690614
AP_HAL::RCOutput::push	no trend	0.447584404251
Copter::read_inertia	no trend	0.0246549490797
Copter::check_ekf_yaw_reset	no trend	0.976207573973
Copter::update_flight_mode	no trend	0.0429176018518
Copter::stabilize_run	no trend	0.530130204928
Copter::update_home_from_EKF	no trend	0.864387150582
Copter::update_land_and_crash_detectors	no trend	0.96668459171
gcs_check_input	no trend	0.611479982466
full_rate_logging_loop	no trend	0.399484796904
dataflash_periodic	no trend	0.16364779152
Copter::read_radio	no trend	0.0743642010545
AP_HAL::RCInput::new_input	no trend	0.333228305536
AP_HAL::RCInput::read	no trend	0.79361407775
update_thr_average	no trend	0.22620129573
compass_accumulate	no trend	0.000747078053683
throttle_loop	no trend	0.538252657989
AP_GPS::update	no trend	0.473831827291
run_nav_updates	no trend	0.413826800767
barometer_accumulate	no trend	0.388885411808
AP_HAL::GPIO::write	no trend	0.622742425685
gcs_send_deferred	no trend	0.364916395192
gcs_data_stream_send	no trend	0.547644567011
update_mount	no trend	0.102947339387
fifty_hz_logging_loop	no trend	0.837092018383
Copter::fast_loop	increasing	0.0
AP_HAL::RCOutput::read	no trend	0.945722576753
AP_HAL::RCInput::read	no trend	0.913776410495
Copter::Log_Sensor_Health	no trend	0.0472994278222

References：

[1] Parnas D L. Software aging[C]//Proceedings of the 16th international conference on Software engineering. Surriento, 1994: 279-287.

[2] Huang Y, Kintala C, Kolettis N, et al. Software rejuvenation: Analysis, module and applications[C]//Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on. IEEE, 1995: 381-390.

[3] Candea G, Cutler J, Fox A. Improving availability with recursive microreboots: a soft-state system case study[J]. Performance Evaluation, 2004, 56(1): 213-248.

[4] Trivedi K S, Vaidyanathan K. Software aging and rejuvenation[J]. Wiley Encyclopedia of Computer Science and Engineering, 2008.

[5] Garg S, Puliafito A, Telek M, et al. Analysis of software rejuvenation using Markov regenerative stochastic Petri net[C]//Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on. IEEE, 1995: 180-187.

[6] Garg S, Puliafito A, Telek M, et al. Analysis of preventive maintenance in transactions based software systems[J]. Computers, IEEE Transactions on, 1998, 47(1): 96-107.

[7] Trivedi K S, Vaidyanathan K, Goševa-Popstojanova K. Modeling and analysis of software aging and rejuvenation[C]//Simulation Symposium, 2000.(SS 2000) Proceedings. 33rd Annual. IEEE, 2000: 270-279.

[8] Bao Y, Sun X, Trivedi K S. A workload-based analysis of software aging, and rejuvenation[J]. Reliability, IEEE Transactions on, 2005, 54(3): 541-548.

[9] Castelli V, Harper R E, Heidelberger P, et al. Proactive management of software aging[J]. IBM Journal of Research and Development, 2001, 45(2): 311-332.

[10] Garg S, Van Moorsel A, Vaidyanathan K, et al. A methodology for detection and estimation of software aging[C]//Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on. IEEE, 1998: 283-292.

[11] Vaidyanathan K, Trivedi K S. A measurement-based model for estimation of resource exhaustion in operational software systems[C]//Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on. IEEE, 1999: 84-93.

[12] Li L, Vaidyanathan K, Trivedi K S. An approach for estimation of software aging in a web server[C]//Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n. IEEE, 2002: 91-100.

[13] Avritzer A, Czekster R M, Distefano S, et al. Software Aging and Rejuvenation for Increased Resilience: Modeling, Analysis and Applications[M]//Resilience Assessment and Evaluation of Computing Systems. Springer Berlin Heidelberg, 2012: 167-183.

[14] Grottke M, Li L, Vaidyanathan K, et al. Analysis of software aging in a web server[J]. Reliability, IEEE Transactions on, 2006, 55(3): 411-420.

[15] Alonso J, Matias R, Vicente E, et al. A comparative experimental study of software rejuvenation overhead[J]. Performance Evaluation, 2013, 70(3): 231-250.

[16] Li W, Zheng Z, Hao P, et al. Predicate Execution-Sequence Based Fault Localization Algorithm[J]. Chinese Journal of Computers, 2013, 36(12): 500-511.

[17] Hao P, Zheng Z, Zhang Z Y. et al. Self-Adaptive Fault Localization Algorithm Based on Predicate Execution Information Analysis[J]. Chinese Journal of Computers, 2014, 37(3): 500-511.

[18] Pytlik B, Renieris M, Krishnamurthi S, et al. Automated fault localization using potential invariants[J]. arXiv preprint cs/0310040, 2003.

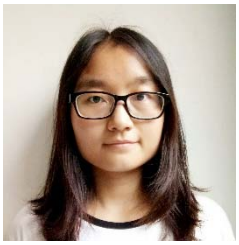
[19] Xuan J, Monperrus M. Learning to combine multiple ranking metrics for fault localization[C]//2014 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2014: 191-200.

[20] Richard O. Gilbert. Statistical methods for environmental pollution monitoring[M]. John Wiley & Sons, 1987.

附中文参考文献：

[16] 李伟, 郑征, 郝鹏, 等. 基于谓词执行序列的软件缺陷定位算法[J]. 计算机学报, 2013, 36(12): 2406-2419.

[17] 郝鹏, 郑征, 张震宇, 等. 基于谓词执行信息分析的自适应缺陷定位算法[J]. 计算机学报, 2014, 37(3): 500-511.



DU Xiaoting was born in 1992. She is a M.S. student at Beihang University. Her research interests include software testing and software fault localization.

杜晓婷(1992-), 女, 山东济宁人, 北京航空航天大学硕士研究生, 主要研究领域为软件测试和缺陷定位。

WANG Nan was born in 1990. He received his master degree in Aero-nautical Engineering at Beihang University, China. He is currently a Ph.D. candidate in Electrical and Computer Engineering at Duke University, USA. His research interests include software dependability modeling and software fault localization.

王楠(1990-), 男, 四川沐川人, 美国杜克大学博士研究生, 主要研究领域为软件测试建模和缺陷定位。