计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.20201082
59(6):1388-1408,2022

基于错误根因的 Linux 驱动移植接口补丁推荐

李 斌[1,2] 贺也平[1,2,3] 马恒太[1,2] 芮建武[1,2] 李晓卓[1,2]
[1](中国科学院大学 北京 100049)
[2](中国科学院软件研究所基础软件国家工程研究中心 北京 100190)
[3](计算机科学国家重点实验室(中国科学院软件研究所) 北京 100190)
(libin@iscas.ac.cn)

Recommending Interface Patches for Linux Drivers Porting Based on Root Cause of Error

Li Bin[1,2], He Yeping[1,2,3], Ma Hengtai[1,2], Rui Jianwu[1,2], and Li Xiaozhuo[1,2]
[1](*University of Chinese Academy of Sciences*, *Beijing* 100049)
[2](*National Engineering Center of Fundamental Software*, *Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100190)
[3](*State Key Laboratory of Computer Science* (*Institute of Software*, *Chinese Academy of Sciences*), *Beijing* 100190)

Abstract Linux kernel version changes bring inconsistency errors in driver calls to kernel interfaces very frequently. It is not only a heavy workload through manual repair, but also may introduce new errors. To overcome this problem, exiting researches on driver porting of middle library auxiliary adaptation and driver porting of auxiliary information provide auxiliary examples, but manual analysis and manual patch construction are still required, so the manual repair workload is still relatively large and the efficiency is low. To this end, we recommend high-quality patches to reduce the workload and improve the efficiency of manual error repair. Unlike traditional methods that identify the same type of errors through the similarity of error code forms, we propose to identify the same type of errors based on the same cause and origin of the error. A hierarchical search algorithm is proposed to obtain the root cause of the error to be fixed, through the root cause to identify the same type of error fix instances, extract and select targeted fix patterns to implement high-quality patch recommendations for the same type of unfixed errors. Experiments on the collected 19 real driver data sets show that the method in this paper has significant improvement in the correct rate of patch recommendation compared with the traditional methods.

Key words driver porting; existing fix instances; error-inducing changes; fix pattern; patch recommendation; root cause of error

摘 要 Linux 内核版本变更带来驱动程序调用内核接口的不一致性错误非常频繁,其通过手工修复不仅工作量繁重,还可能引入新的错误.针对这个问题,驱动移植中间库辅助适配和驱动移植辅助信息等方面的已有研究提供了辅助示例,但是还需要人工分析和手工构造补丁,人工修复的工作量依然较大并且效率较低.为此,通过推荐高质量补丁降低人工修复的工作量并提高修复效率.与传统方法通过错误代码形式的相似性识别同类错误不同,提出依据错误发生的相同原因和来源识别同类错误.提出了一种

收稿日期:2020-12-26;**修回日期**:2021-05-08
基金项目:中国科学院战略性先导科技专项(XDA-Y01-01,XDC02010600)
This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences (XDA-Y01-01, XDC02010600).
通信作者:贺也平(yeping@iscas.ac.cn)

分层搜索算法用于获取待修复错误对应的错误根因,通过错误根因识别同类错误的修复实例,从其中提取并选择针对性修复模板实现同类未修复错误的高质量补丁推荐.在收集的 19 个真实驱动程序数据集上的实验表明,所提方法相比传统方法的补丁推荐正确率有显著提高.

关键词　驱动移植;修复实例;引入错误变更;修复模板;补丁推荐;错误根因

中图法分类号　TP311

Linux 驱动程序是操作系统中非常重要的系统组件且代码体量庞大,占据 Linux 内核大约 70%的代码量[1].在实际操作系统研发中,如何使设备驱动程序跟上对应操作系统其他部分的版本演进是当前面临的最大问题之一[2].当 Linux 内核版本升级后,由于内核版本的升级可能引入一些内核接口服务的变更,使得驱动程序代码中一些内核接口调用不再适应更新后的内核版本,导致原有驱动程序在新的内核环境中由于不相适配可能产生不一致性错误[3].为了适配频繁变化的内核版本并维持已有设备在新内核环境中的可用性,开发者需要针对驱动程序代码进行持续的适配性修改并完成驱动程序移植.然而内核版本变更对驱动程序带来的关联影响程度和影响范围都很大,由于 Linux 内核版本的变更使得对应驱动程序在移植中所需的适配性修改代码行高达 35%以上[3],Linux 内核版本中引入的单一变更可能会影响驱动程序中分布在许多不同文件中的数百个代码调用点[4].因此,开发者手工进行适配性修改完成这种移植工作通常工作量繁重、效率较低,而且还可能引入新的错误.

针对上述问题,研究者们在驱动移植中间库辅助适配[5-6]和驱动移植辅助信息[7-8]等方面开展了研究.这些工作从提高辅助示例的检索精度和效率角度,通过提供辅助示例在一定程度上减轻了驱动移植开发者的负担.但是还需要开发者进一步分析辅助示例和出错代码并人工构造适配性补丁,人工修复的工作量依然较大并且效率较低.为此,本文针对驱动移植场景中接口调用不一致错误,通过推荐高质量补丁降低人工修复错误的工作量并提高修复效率.Tao 等人[9]通过实证研究也指出,相比辅助示例,如果有高质量的补丁辅助开发者,错误修复的正确率和效率都会大幅提升.

补丁推荐和程序自动修复[10-14]虽然都要针对具体错误生成补丁,但是二者在错误定位和补丁质量判定方面存在不同.补丁推荐[15]是针对程序中的错误生成补丁并建议合适补丁的过程,最终由开发者手工判定补丁质量.本文的工作主要是针对驱动移植场景中的接口不一致错误生成补丁并进行补丁推荐.

驱动移植场景中的接口调用不一致错误实际上是一类比较有特点的具体错误,由于不同驱动程序通过统一的内核接口服务大量复用内核接口,因此 Linux 内核版本升级引起的某个单点内核接口定义变化会导致多个驱动程序中该复用接口的不同调用点产生同类不一致错误.针对一般的错误,早期的研究[16-19]利用遗传算法对可复用的代码碎片进行交叉变异生成补丁,然而这种随机变异会产生大量无意义的补丁.为了提高补丁的质量,研究发现很多错误具有相同类型,一组同类错误的修复实例中实际上隐含了修复模板,因此针对未修复的同类错误可以在修复模板的约束和指导下生成补丁.其主要包含 2 个关键问题:1)识别同类错误用于提取和选择修复模板;2)基于修复模板生成补丁.修复模板抽象化了一组同类错误和对应补丁之间隐含的共性变更,基于修复模板生成补丁则是确定多样化的代码元素将选择的抽象修复模板具体化的过程.由此,已有研究从提取修复模板[20-24]、选择修复模板[25-29]和确定多样化的具体代码元素[30-36]这 3 个角度展开.

本文通过研究提取修复模板和选择修复模板这 2 个角度,实现高质量补丁推荐:

1) 传统方法提取修复模板主要通过相似度聚类来识别哪些是已修复的同类错误,并通过抽象化修复代码变更的共性提取修复模板.已有研究通过刻画错误和对应补丁之间的变更特点[20-22]、错误和对应补丁所在上下文特点[23],从句法或者语法层面进行相似度聚类,将代码变更通过迁移序列、抽象语法树(abstract syntax tree, AST)进行刻画,通过挖掘频繁序列、识别等价树提取修复模板.这些基于语法相似度的方法适合挖掘高频的原子变更模板,但是由于缺少语义依赖关系而对于由多个原子变更组成的复合模板挖掘存在限制.还有学者研究通过数据流和控制流图连接了从错误到对应补丁之间涉及的变更元素所对应的语义依赖关系[24],通过识别同构图挖掘重复的代码变更进行聚类,因此可以提取具有语义关联的复合模板.但是这 2 类基于语法和语义相似度的方法都是通过错误代码形式识别哪些

是同类错误的修复实例,即错误和对应补丁所在语句是错误的形式特征.针对一些本质上错误类型相同但表现形式不同的错误,则文献[20-24]的方法会存在识别同类错误范围狭窄的问题.本文研究的驱动移植接口不一致错误也是这样一类具体错误,该类错误虽然复用相同内核接口但是所在的调用点不同(所在驱动程序不同以及源代码文件位置不同),不同调用点的同类错误所在语句类型、上下文等表现形式可能不同[4].如果直接使用文献[20-24]的传统方法将出现把相同错误区分为不同错误,导致应该提取1个修复模板但实际提取了多个不同修复模板的问题.

2) 选择修复模板,要解决的主要问题是区分待修复错误的特征来选择合适的修复模板,其核心思想是同类错误共享同一个修复模板.一些研究工作[25-26]通过模板频度信息作为权重进行倾向性选择,相当于假设待修复错误都是常见错误,并没有考虑待修复错误的具体特点.另一些研究工作[27-29]通过错误的上下文特点选择合适的模板,认为上下文相似(如错误所在语句类型和修复模板附带的语句类型信息相似)则识别为同类错误,也是依据错误代码形式的特点识别同类错误.在错误相同但表现形式不同的驱动移植接口不一致错误中,如果直接使用文献[25-29]的传统方法将可能出现把相同错误区分为不同错误,导致应该选择1个修复模板却实际选择了多个不同修复模板的问题.

总之,将相同错误识别为不同错误,提取和选择不恰当的修复模板都会严重影响补丁的质量[37].

与已有研究通过错误代码形式的相似性识别同类错误的思想不同,本文提出通过分析错误自身的原因和来源识别同类错误.更具体地说,错误根因信息刻画了错误发生的原因和来源,引入错误变更(error-inducing changes, EIC)就是一种根因信息.本文通过分析和待修复错误具有相同引入错误变更信息识别同类错误修复实例,从同类错误修复实例提取并选择针对性修复模板实现同类待修复错误的高质量补丁推荐.Wen等人[38]通过实证研究分析了引入错误变更信息和修复补丁之间的关系,也指出利用引入错误变更素材提高补丁的质量具有一定实证基础.我们观察发现:相同内核接口在驱动程序的多个不同代码调用点存在接口复用现象,由于接口复用这些不同的接口调用点依赖相同的内核接口来源.Linux内核升级之后,在Linux内核主线版本的驱动程序历史开发信息中,可能存在针对内核版本升级引入的接口不一致问题的同类错误修复实例.因此,可以通过识别这些同类错误修复实例的共性信息提取并选择修复模板,用于实现同类未修复错误的高质量补丁推荐.虽然这些复用接口的修复实例分布在不同驱动程序的不同文件中,不同调用点所在的语句类型和上下文等具体表现形式可能不同,但是同类复用接口具有相同的错误原因和来源.本文通过根因信息(即引入错误变更)识别同类修复实例提取并选择针对性修复模板的技术路线更加合理.

实现基于根因识别同类错误的思路包含2个方面的技术困难:1)针对某个出错接口,在庞大的历史开发信息中找到对应的引入错误变更信息并不容易.这些引入错误变更信息可能是多样化的并且嵌入在历史开发的提交信息之中[39],而且历史信息中的混合提交又进一步给引入错误变更信息的获取带来了噪声和干扰[40].2)针对同类错误修复实例的识别,获取与待修复错误具有相同出错原因的同类错误修复实例是关键问题.即使准确找到了出错接口对应的引入错误变更信息,借助引入错误变更信息获取同类修复实例并不简单.由于错误表现形式的多样性,一些修复实例往往包含个性化代码元素,潜在的同类错误修复实例语句与待修复错误所在的语句、引入错误变更语句可能并不直接相似.因此,仅仅使用简单的匹配技术或者相似度计算并不能准确地分析出待修复错误、引入错误变更和潜在的同类错误修复实例三者之间的关系,因为它们的共性特征隐藏在语句内部的局部位置[41].

为了提取出错语句对应的引入错误变更信息,本文提出了一种基于编译的分层检索算法.依据引入错误变更信息的分层结构特点和提交时序特点,本文先识别引入错误提交(error-inducing commits, EICommits)再分析该提交信息所包含的潜在文件变更,对文件变更包含的代码块进行更细粒度切分,在切分的代码块变更集合中找到引入错误变更信息.在获取与待修复错误对应的同类错误修复实例时,面临历史开发信息庞大、待修复出错语句和对应引入错误变更与潜在的同类修复实例共性隐含在局部位置的问题.本文首先通过出错接口信息和启发式规则,在历史开发信息中获取一个候选修复实例集合.然后通过引用关系分析,从候选实例集合中检索哪些实例与待修复出错语句引用了相同的引入错误变更.因为共同定义变更(错误根因)的关联影响产生了多个不同调用点的同类接口错误及其修复实例,由此将具有相同错误原因的实例确定为一组同

类修复实例.最后,我们实现了基于错误根因进行驱动移植接口补丁推荐的方法 RCFix(root cause based fix),采用迭代方式逐个接口错误进行补丁推荐,通过人工判定补丁质量.在收集的 19 个真实驱动程序数据集上进行检验,本文的方法推荐补丁的 top-1,top-2,top-5 正确率分别为 58.7%,60.6%,66.1%,针对每款驱动程序补丁推荐的平均耗时大约在 14 min 以内.相比传统的方法,本文的方法有效辅助开发者降低了错误修复的工作量并提高了修复效率.

本文的主要贡献有 3 个方面:

1) 针对传统方法通过错误代码形式的相似性识别同类错误中存在的识别范围狭窄的问题,提出基于错误根因识别同类错误的补丁推荐方法.相比传统方法,本文通过错误根因能够更好地识别本质是同类错误但错误代码形式不相似的问题.根据现有文献调研分析,这是一种比较新颖的思路.

2) 提出了一种基于编译的分层检索算法,用于获取待修复错误对应的根因信息(即引入错误变更),其正确率达到 80.7%.利用相同错误根因识别同类错误修复实例,进而从同类错误修复实例提取并选择针对性修复模板实现同类待修复错误的高质量补丁推荐.

3) 检验了基于错误根因进行驱动移植接口补丁推荐的有效性,实验表明本方法推荐补丁的 top-5 正确率达到 66.1%,相比传统方法的补丁推荐正确率有显著提高.

1　动　　机

在驱动程序移植场景中,由于内核版本变更引入的接口变化导致驱动程序接口调用与内核提供的接口不相适配,需要针对驱动程序引入适配性补丁使其在更新后的内核环境中维持现有功能可用.我们对内核和驱动程序开发历史信息进行了分析,如图 1 是因特尔网卡 e1000 驱动程序从 Linux-3.5 移植到 Linux-4.0 过程中的一个出错点,该错误是宏 *NETIF_F_HW_VLAN_RX* 支持硬件操作 VLAN 的标签.代码文件 e1000_main 中行 818 的－语句出错,应该修改为对应的＋语句,即本文期望生成行 819 的＋语句补丁并推荐给开发者.

	drivers/net/ethernet/intel/e1000/e1000_main.c
818	－if(*features* & *NETIF_F_HW_VLAN_RX*) /* 错误语句 */
819	＋if(*features* & *NETIF_F_HW_VLAN_CTAG_RX*) /* 修复语句 */
⋮	⋮

Fig. 1　Interface errors of driver porting in driver e1000

图 1　驱动程序 e1000 中的驱动移植接口错误

1.1　修复实例

我们分析了移植区间(Linux-3.5 至 Linux-4.0)中其他驱动程序的开发历史信息.我们发现内核版本升级之后,由于内核接口复用,一些其他驱动程序的历史开发信息中包含了同类错误修复实例.如图 2 是宏 *NETIF_F_HW_VLAN_RX* 的修复实例,其中 2 个实例是瑞昱网卡 r8169 驱动和因特尔 igb 驱动程序针对 Linux 内核中的宏 *NETIF_F_HW_VLAN_RX* 的使用和变更情况.

进一步分析发现,出错语句和实例语句之间存在共性的同时也存在差异性.图 2 的实例中修复了和图 1 相同的宏 *NETIF_F_HW_VLAN_RX* 的错误,但是出错语句形式不同,图 1 的错误是条件语句而图 2 的错误是赋值语句.如果从工具的角度仅仅

修复实例 1	diff--git a/drivers/net/ethernet/realtek/r8169.c b/drivers/net/ethernet/realtek/r8169.c －*dev*→*hw_features* &=～*NETIF_F_HW_VLAN_RX*; ＋*dev*→*hw_features* &=～*NETIF_F_HW_VLAN_CTAG_RX*;
修复实例 2	diff--git a/drivers/net/ethernet/intel/igb/igb_main.c b/drivers/net/ethernet/intel/igb/igb_main.c －*dev*→*features*|=*NETIF_F_RXCSUM*|*NETIF_F_HW_VLAN_CTAG_TX*|*NETIF_F_HW_VLAN_RX*; ＋*dev*→*features*|=*NETIF_F_RXCSUM*|*NETIF_F_HW_VLAN_CTAG_TX*|*NETIF_F_HW_VLAN_CTAG_RX*;

Fig. 2　Existing fix instances of macro *NETIF_F_HW_VLAN_RX* in development history

图 2　宏 *NETIF_F_HW_VLAN_RX* 在历史开发中的已有修复实例

利用错误代码形式的相似性识别同类错误的修复实例,则只可能找到和出错语句相同或者非常接近的修复实例.然而,很多同类复用接口错误和修复实例所在的出错语句形式和语句内部使用的变量名称或者表达式并不相同,这些差异性阻碍了利用相似性识别同类错误修复实例的数量和质量.例如,同类复用接口的出错语句类型除了赋值语句和条件语句之外,实际历史信息中可能还包括各种循环语句、跳转语句、表达式语句或者语句块等更加多样化和复杂的不同语句上下文形式.另外,同类出错语句内部也可能使用各种个性化的元素,例如图 1 出错语句使用了 *features* 字段,而图 2 实例中使用了 *dev* 对象和 *hw_features*, *NETIF_F_RXCSUM*, *NETIF_F_HW_VLAN_CTAG_TX* 等宏字段.因此,通过相似性无法识别这些具有显著差异性的同类错误修复实例,这影响了潜在的同类错误修复实例识别的数量和质量.

根据本文的观察和分析,一方面,在历史开发信息中存在和出错接口类型相同的修复实例,这些同类错误修复实例可以通过有效的方法进行识别并提取修复模板,用于生成未修复的同类错误待推荐补丁;另一方面,由于复用接口所在的调用点语句类型、表达式或者变量名称等差异因素使得修复实例的表现形式存在多样性,利用错误代码形式的相似性计算仅可能获得语句形式非常接近的修复实例,但是准确地识别语句形式具有显著差异性的同类错误修复实例并不容易.

1.2 引入错误变更

为了识别表现形式具有差异性的同类错误修复实例,我们发现错误根因(即引入错误变更信息)是一种非常关键的信息.引入错误变更信息提供了理解错误是如何引入的重要信息并且解释了错误原因,如果出错语句与修复实例具有相同的错误原因则二者是同类错误问题.如图 3 是宏 *NETIF_F_HW_VLAN_RX* 错误在 Linux 内核历史开发信息中引入错误变更的代码片段,内核在 Linux-3.5 之后的版本升级过程中,内核开发组在头文件 netdev_features 中对宏定义 *NETIF_F_HW_VLAN_RX* 的名称进行了更新.图 1 中的错误语句和图 2 中的 2 个修复实例是同类错误,图 3 对宏定义 *NETIF_F_HW_VLAN_RX* 名称的更新解释了它们共同的出错原因.

	diff--git a/include/linux/netdev_features.h	b/include/linux/netdev_features.h
	index d6ee2d0..785913b 100644	
1	−#define *NETIF_F_HW_VLAN_FILTER*	*__NETIF_F(HW_VLAN_FILTER)*
2	−#define *NETIF_F_HW_VLAN_RX*	*__NETIF_F(HW_VLAN_RX)*
3	−#define *NETIF_F_HW_VLAN_TX*	*__NETIF_F(HW_VLAN_TX)*
4	+#define *NETIF_F_HW_VLAN_CTAG_FILTER*	*__NETIF_F(HW_VLAN_CTAG_FILTER)*
5	+#define *NETIF_F_HW_VLAN_CTAG_RX*	*__NETIF_F(HW_VLAN_CTAG_RX)*
6	+#define *NETIF_F_HW_VLAN_CTAG_TX*	*__NETIF_F(HW_VLAN_CTAG_TX)*

Fig. 3 Error-inducing changes of macro *NETIF_F_HW_VLAN_RX* in kernel development history

图 3 宏 *NETIF_F_HW_VLAN_RX* 在内核开发历史中的引入错误变更

尽管引入错误变更信息能够提高识别同类错误修复实例的数量和质量,但提取出错语句对应的引入错误变更信息并不容易.引入错误变更信息往往混合在其他代码变更语句中,例如图 3 的引入错误变更混合在另外 2 个宏定义 *NETIF_F_HW_VLAN_FILTER* 和 *NETIF_F_HW_VLAN_TX* 的变更代码块中.而且实际历史开发信息中引入错误变更代码块嵌入在引入错误提交信息中,引入错误提交一般又包含了其他文件代码块变更的干扰信息,例如其他代码块中可能包含和出错字段文本相似甚至同名的变量、同名函数等干扰因素.

实例分析启发我们利用错误原因和来源识别同类错误修复实例,即通过引入错误变更的中间信息提高获取同类错误修复实例的数量和质量,在识别与待修复错误具有相同原因的同类修复实例的基础上,通过提取并选择针对性修复模板提高补丁推荐的质量.

2 补丁推荐方法

2.1 补丁推荐方法概述

首先介绍本文方法的整体结构,如图 4 所示主要包括候选修复实例搜索、引入错误变更搜索、同类修复实例识别和修复模板提取、补丁推荐 4 个部分.

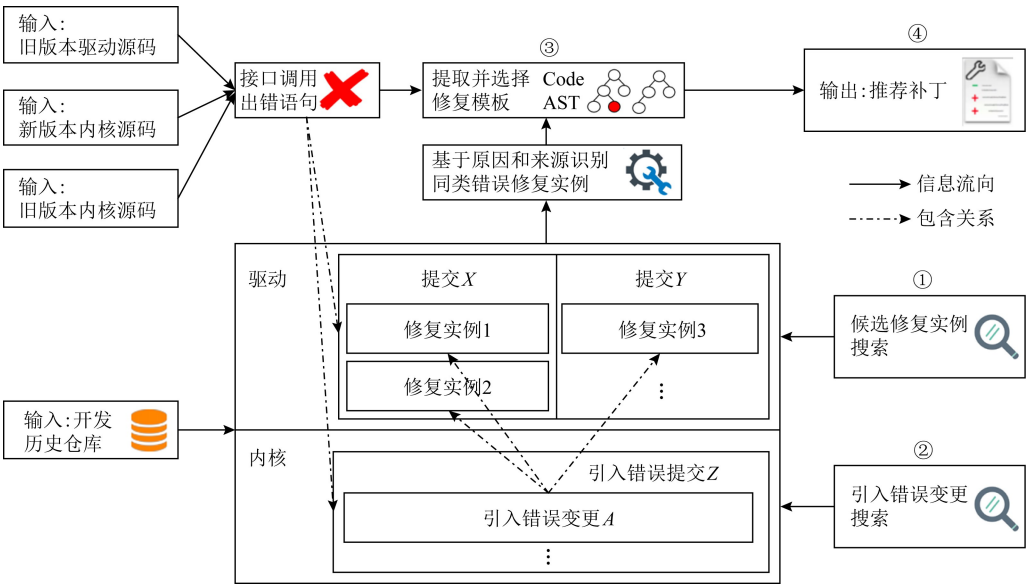

Fig. 4　Driver porting interface patches recommendation by inducing error changes

图 4　利用引入错误变更信息的驱动移植接口补丁推荐

本文的方法输入包括旧版本驱动程序源码(待移植驱动程序)、新旧版本内核源码(新版本内核为前向移植的目标内核版本)、新旧版本内核之间的开发历史仓库(git repository);输出内容是旧版本驱动程序移植到新版本内核过程中,针对驱动程序代码中内核接口调用错误推荐的补丁列表.本文将待移植驱动程序在新内核版本环境下进行编译,针对错误日志中的一系列接口调用出错语句,采用迭代方式逐个进行补丁推荐.由于历史信息庞大且存在混合提交,候选修复实例搜索通过启发式规则进行噪声处理,结合错误相关关键字检索从其他驱动程序的开发历史信息中检索接口调用出错语句对应的候选修复实例集合.引入错误变更搜索使用基于编译的分层检索算法,从 Linux 内核的历史开发信息中检索出错语句对应的引入错误变更信息.同类实例识别和模板提取通过引用关系分析出错语句、引入错误变更和候选修复实例三者隐含的因果依赖关系,从而确定一组和待修复错误具有相同错误原因的有效修复实例,使用细粒度差异比较技术分析多个有效修复实例的共性变更,进而抽象细节提取修复模板.补丁推荐使用编辑脚本技术将提取和选择的针对性修复模板进行实例化并生成待推荐补丁,通过补丁排序并推荐给开发者进行质量判定.

具体结合图 1~3 的实际示例进行说明.将旧版本 e1000 驱动程序(待移植驱动)在新内核版本 Linux-4.0 环境下编译,解析编译日志发现其中一个接口错误是图 1 中/drivers/net/ethernet/intel/e1000/e1000_main.c 文件中行 818 语句"if(*features* & *NETIF_F_HW_VLAN_RX*)"宏 *NETIF_F_HW_VLAN_RX* 错误,该宏实现的功能是对 VLAN 标签的开关控制.针对该错误语句,本文通过 4 个步骤最终实现接口补丁推荐:①候选修复实例搜索,针对出错语句"if(*features* & *NETIF_F_HW_VLAN_RX*)"在其他驱动程序开发历史中进行搜索,通过该出错语句的接口关键字和启发式规则从开发历史仓库中搜索其他驱动程序的修复实例获得候选修复实例集合,如图 2 中的修复实例 1 和修复实例 2 是其中 2 个候选修复实例.②引入错误变更搜索,针对图 1 的行 818 错误语句,通过本文基于编译的分层检索算法获得图 3 引入错误变更信息,即该错误发生的原因信息.③同类错误修复实例识别和模板提取,通过引用关系分析引起错误的相同依赖,通过引入错误变更信息识别出错语句和候选实例二者有相同的错误原因,从而确定同类修复实例,然后使用细粒度差异比较技术从一组同类修复实例中提取共性变更形成修复模板.④补丁推荐,选择已提取的针对性修复模板,使用修复模板附带的参数(例如变更后的宏名称)将修复模板具体化,使用编辑脚本技术生成待推荐补丁.

2.2　候选修复实例搜索

2.2.1　出错接口语句识别

本文聚焦在驱动移植的接口不一致错误进行补丁推荐,首先需要识别出错接口相关的语句信息.本文采用编译的方式,在新版本内核环境中编译待移植

驱动程序并从编译日志中提取出错接口相关的信息.通常接口相关的编译错误有3类报错字段:1)excepted error通常是宏定义相关的错误;2)implicit and undeclared error是接口名称相关错误,例如接口名称变化、接口类型变化、接口声明所在引用文件变化等;3)arguments error通常和接口参数错误相关,例如参数增加、参数减少、参数类型改变等.本文采用关键字正则匹配获得接口错误语句相关的信息,具体包括编译错误类型、出错接口所在源码文件路径信息、出错接口名称、出错接口所在源码文件的行号位置等.

2.2.2 历史信息噪声过滤

针对2.2.1节获取的每个出错接口语句,本文在其他驱动程序的开发历史信息中搜索与出错接口相关的候选修复实例.然而,驱动程序开发历史中的commit提交信息可能包含缺陷修复、新特性开发,或者二者的混合提交,一些不规范提交也会引入大量噪声为修复实例的检索造成干扰,同时检索语句中关键字的构造也可能使得提取的结果包含噪声.针对噪声问题,本文设计了3个启发式规则对commit检索和提取过程进行限制和过滤,在过滤噪声的基础上形成候选修复实例集合.

针对commit检索过程中的噪声过滤,本文设计3个启发式规则:

1)由于接口错误不同于其他类型错误可能引入大块代码变更,根据本文的分析一般接口调用的变更涉及2条语句,因此本文通过限制变更代码行数裁剪commit片段中的其他干扰代码,过滤后的commit代码片段最多包含2条语句代码块.例如,代码块包含减掉1条含有出错接口名称的语句,紧接着再增加1条变更语句的情况,或者只有增加或者删除语句.具体可能是占用2行修改量或者最多占用4行修改量,使用正则表达式识别有些语句因为编码过长导致存在换行的情况.

2)由于commit提交中可能包含错位提交或者截取已有修复实例和出错接口无关的情况,本文对commit中变更代码截取进一步限制,通过编译错误类型和截取的已有修复实例的样式进行匹配.通过编译日志识别3类错误,包括宏定义错误、接口名称错误和参数错误,而这3类错误各有特点可以用于匹配commit截取.例如,宏定义一般使用大写形式,接口名称相关代码往往采用驼峰命名规则(如大小写混合等),而参数一般对应的代码变更在括号区间中.因此,本文使用特征过滤与编译错误类型不符的实例,具体通过正则表达式匹配实现.

3)还需要去除冗余和重复的修复实例,因为驱动开发中可能存在代码块的复用、回滚、冗余功能等,具体通过计算每个修复实例的哈希值并进行比较后剔除冗余.

2.2.3 候选修复实例集合

为了获取候选修复实例集合,本文通过检索包含候选实例的commit信息并从中提取候选实例代码片段.Linux内核和驱动程序的历史开发信息由git工具[42]来维护,本文通过构造git检索关键字在对应移植区间进行检索.具体查询语句构造为:git log-no-merges-p-S 'function name' $V_{old}..V_{new}$--dir Linux/dirvers/>instances.txt.其中function name是出错接口名称,移植区间在新旧内核版本V_{old}和V_{new}之间,--dir标识待检索的驱动项目开发历史,出错接口的使用变更信息来自移植区间对应的驱动开发历史.通过2.2.2节的启发式规则对git检索结果进行限制和过滤,输出一组疑似的和出错接口相关的代码变更片段.

针对输出的一组和出错接口相关的代码变更片段,选择固定数量的实例组成候选修复实例集合.通过分析,驱动程序类型相近的修复实例更具有参考价值.本文通过驱动路径匹配来确定驱动程序类型是否相近,分析候选实例所在的驱动路径和待修复错误所在的驱动路径字段的文本相似性,依据相近驱动类型确定固定数量实例组成候选修复实例集合.

2.3 引入错误变更搜索

引入错误变更信息是包含在Linux内核历史提交信息中的代码变更片段.为了搜索待修复错误对应的引入错误变更信息,本文采用编译的方式进行检索.待移植驱动程序在该代码片段提交之前的内核版本中能够编译通过,而该引入错误变更提交之后会导致驱动程序中对应的接口调用出错而编译失败.针对待移植驱动程序中接口错误语句,本文通过分层遍历Linux内核历史开发信息的方式,识别哪些提交的代码片段导致了该错误.

2.3.1 引入错误变更信息分层结构分析

历史commit提交信息具有时序性和分层结构特点.如图5内核开发历史由很多分支组成,其他分支最终都合并到主分支形成对应的Linux内核版本.每个分支中的提交信息的时序关系可以简化地视为线性提交形式,每个提交信息可能包含若干文件的修改和变更,而引入错误变更信息嵌入在引入错误提交信息之中.引入错误变更在历史开发信息中具有分层提交的特点,更具体地说,引入错误变更

是 1 个代码修改片段并且包含在文件的修改中,而 1 个或者多个文件的修改组成 1 个 commit 提交,1 个或者多个 commit 提交组成了不同的提交分支,而不同的分支合并形成了具体的内核版本.

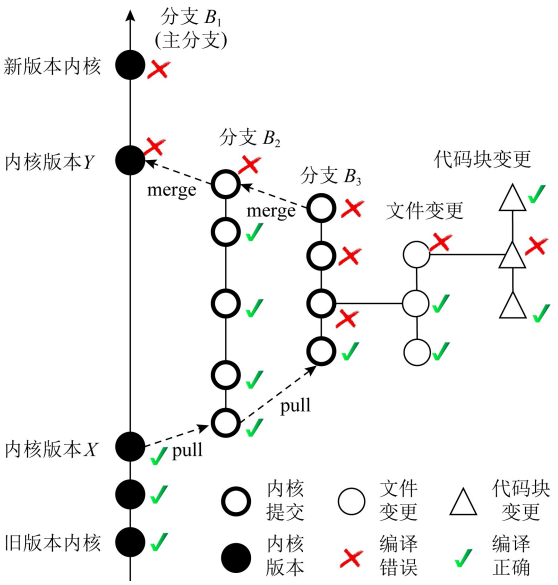

Fig. 5　Hierarchical search error-inducing changes

图 5　分层检索引入错误变更信息

本文结合 commit 提交信息的时序性和分层结构特点进行引入错误变更搜索,具体包括 4 个部分:1)将搜索空间缩小到 2 个内核小版本之间,在驱动移植区间(内核新旧版本之间)对应的搜索范围内通过二分查找遍历内核小版本,获得包含引入错误变更信息的 Linux 内核版本区间.2)获取引入错误提交,在已经确定的内核小版本区间中所包含的提交信息,通过分层的迭代检索(每一层是指每个分支中的提交信息是线性的,采用二分查找)获得包含引入错误变更信息的引入错误提交.3)获取文件变更,对引入错误提交包含的文件变更信息进行过滤,去除出错接口所在文件未引用的头文件代码变更,去除非源码变更(例如注释信息)等不相关信息.4)文件变更分割,连续的变更分割为代码块变更,包括连续增加、连续删除和增加删除混杂 3 种类型,引入错误变更则隐含在某个代码块变更信息中.

2.3.2　引入错误变更信息检索算法

引入错误变更是一个内核代码的变更片段,由于修改了驱动依赖的接口定义而引起驱动程序中相关调用发生不一致错误.该不一致错误首先会引起编译错误,即引入错误变更提交之前的内核源码和驱动程序中的接口调用一致,驱动程序在该提交发生之前的内核环境中编译成功,而包含引入错误变更的提交发生之后引入了定义变更导致定义和使用不一致,使得驱动程序在提交后的内核环境中编译产生对应编译错误.根据分层结构和提交的时序特点,该算法采用编译的方式分层查找出错接口对应的引入错误变更代码片段.

算法 1. 基于编译和分层的引入错误变更信息检索算法.

输入:待移植驱动程序源代码 *oldDriver*、接口错误语句代码片段 *errorP*、包含内核版本 commit 提交的开发仓库 *kernelGit*;

输出:接口错误语句对应的引入错误变更代码片段 EIC.

① (*kernelX*,*kernelY*)=*versionBinarySearch*(*oldDriver*,*errorP*,*kernelGit*);
② *branchCommitList*=*firstParent*(*kernelX*,*kernelY*);
③ IF (*commitBinarySearch*(*branchCommitList*))≠Null) THEN
④　RETURN *EICSearch*(*commitBinarySearch*(*branchCommitList*));
⑤ ELSE
⑥　RETURN not find;
⑦ END IF
⑧ Procedure *commitBinarySearch*(*branchCommitList*)
⑨　(*errInducingCommit*,*okParentCommit*)=*BinarySearch*(*kernelX*,*commitList*);
⑩　IF *errInducingCommit* 是一个合并提交 THEN
⑪　*errorParentCommit*=*getErrorParent*(*errInducingCommit*);
⑫　*commonAncentor*=*getCommonAncentor*(*errorParentCommit*,*okParentCommit*);
⑬　*nextBranchCommitList*=*firstParent*(*errorParentCommit*,*commonAncentor*);
⑭　RETURN *commitBinarySearch*(*nextBranchCommitList*);
⑮　ELSE
⑯　RETURN *errInducingCommit*;
⑰　END IF
⑱ END
⑲ Procedure *EICSearch*(*errInducingCommit*)
⑳　*fileChange*=*fileBinarySearch*(*errInducingCommit*);

㉑ *hunkChangelist*=*hunkSplit*(*fileChange*);

㉒ RETURN *findEIC*(*hunkChangelist*);

㉓ END

根据版本提交之间分层结构和提交的时序性特点,算法 1 的行①首先通过 *versionBinarySearch* 的二分查找获得最小版本区间,即驱动程序在内核 X 版本编译成功,而在下一个升级的内核 Y 版本编译失败.行②获取 2 个内核版本区间中的第 1 个分支的 commit 列表 *branchCommitList*,函数 *firstParent* 通过 git log--pretty=format:% h--first-parent[target version..original version]获取第 1 个分支的 *branchCommitList*.行③判断如果获取 *errInducingCommit*,则行④从 *errInducingCommit* 中查找获得引入错误变更 EIC.否则行⑤~⑦返回未找到 EIC 信息.行⑧~⑱引入错误变更提交 *errInducingCommit* 查找函数 *commitBinarySearch* 通过递归查询输出引入错误的提交 *errInducingCommit*,其中,引入错误提交查找函数 *commitBinarySearch* 进行分层检索,行⑨二分查找函数 *BinarySearch* 使用 git bisect 实现,行⑩判断获取的引入错误提交是否为一个合并(merge)提交节点,如果是合并节点则行⑪~⑭继续寻找下一层的引入错误父亲节点和祖先节点继续迭代,如果不是合并节点则行⑯输出引入错误提交.行⑫函数 *getCommonAncentor* 通过 git merge-base errParent okParent 获取.然后行⑲~㉓EIC 查找函数 *EICSearch* 调用文件查找 *fileBinarySearch* 和变更代码块切分 *hunkSplit*,*hunkSplit* 实现连续的代码块变更进行整体切分,最后由 *findEIC* 遍历变更代码块并找到 EIC 信息进行输出.

2.4 同类实例识别和模板提取

2.4.1 同类修复实例识别

本文通过分析与待修复错误具有相同原因的代码块来识别一组同类错误修复实例.即针对某个确定的待修复错误、对应的引入错误变更、对应的候选修复实例集合,识别候选集合中哪些修复实例和待修复错误是由于相同的引入错误变更产生的同类错误.由于引入错误变更实际上是对内核接口服务中接口定义的变更,而待修复的接口不一致错误和修复实例实际上是不同的接口调用点,因此引入错误变更与待修复错误和同类修复实例之间存在引用关系.

为了方便和准确地描述同类错误修复实例的识别,本文给出 6 个定义.

定义 1. 引入错误变更 E_{eic}.设 $E_{\text{eic}}=(M,N)$表示引入错误变更代码块,减语句块 $M=\{m_1,m_2,\cdots,m_m\}$,其中 m_e 表示出错元素字段($1\leqslant e\leqslant m$),M 由 m 个 token 元素组成,刻画了旧接口相关的定义,而 $N=\{n_1,n_2,\cdots,n_n\}$由 n 个 token 元素组成,刻画了新接口相关的定义,则 E_{eic}有 3 种情况:

$$E_{\text{eic}}=\begin{cases}M,\ N\text{ 表示 EIC 包含加减语句对,}\\ \quad\text{对旧接口定义进行了更新(}Update\text{)};\\ M,\ \text{Null 表示 EIC 仅包含减语句,}\\ \quad\text{仅删除(}Delete\text{)了旧接口定义;}\\ \text{Null},\ N\text{ 表示 EIC 仅包含加语句,}\\ \quad\text{仅新增(}Add\text{)了接口定义.}\end{cases}$$

定义 2. 修复实例 F_{efi}.设 $F_{\text{efi}}=(R,S)$表示修复实例的代码块,减语句块 $R=\{r_1,r_2,\cdots,r_r\}$由 r 个 token 元素组成,刻画了修复实例对旧接口的调用,而 $S=\{s_1,s_2,\cdots,s_s\}$,其中 s_x 表示更新元素字段($1\leqslant x\leqslant s$),S 由 s 个 token 元素组成,刻画了修复实例对新接口的调用,则和 E_{eic}类似,F_{efi}也有 3 种情况:

$$F_{\text{efi}}=\begin{cases}R,\ S\text{ 表示修复实例对旧接口相关的元素}\\ \quad\text{进行了更新(}Update\text{)};\\ R,\ \text{Null 表示修复实例对旧接口调用}\\ \quad\text{进行删除(}Delete\text{)};\\ \text{Null},\ S\text{ 表示修复实例直接调用了}\\ \quad\text{新增(}Add\text{)接口.}\end{cases}$$

定义 3. 旧引用链 C_{old}.如果 E_{eic}包含 1 对加减语句块,即 $M\neq\text{Null}\&\&N\neq\text{Null}$,则旧引用链 $C_{\text{old}}=m_e\rightarrow r_e$,表示 F_{efi}对 E_{eic}中更新前元素定义的使用,e 表示出错字段对应的元素下标;如果 E_{eic}仅包含减语句块,即 $M\neq\text{Null}\&\&N=\text{Null}$,则旧引用链 $C_{\text{old}}=m_e\rightarrow r_e$,表示 F_{efi}对 E_{eic}中已删除元素定义的使用,e 表示出错字段对应的元素下标;如果 E_{eic}仅包含加语句块,即 $M=\text{Null}\&\&N\neq\text{Null}$,则 $C_{\text{old}}=\text{Null}$.

定义 4. 新引用链 C_{new}.如果 E_{eic}包含 1 对加减语句块,即 $M\neq\text{Null}\&\&N\neq\text{Null}$,则新引用链 $C_{\text{new}}=n_x\rightarrow s_x$,表示 F_{efi}对 E_{eic}中更新后元素定义的使用,x 为更新字段对应的元素下标;如果 E_{eic}仅包含减语句块,即 $M\neq\text{Null}\&\&N=\text{Null}$,则 $C_{\text{new}}=\text{Null}$;如果 E_{eic}仅包含加语句块,即 $M=\text{Null}\&\&N\neq\text{Null}$,则 $C_{\text{new}}=n_x\rightarrow s_x$,表示 F_{efi}对 E_{eic}新增语句中元素定义的使用,x 为更新字段对应的元素下标.

定义 5. 同类错误修复实例 $F_{\text{efi}}^{\text{same}}$.2 个修复实例依赖相同的引入错误变更,则为同类错误修复实例

F_{efi}^{same}.如果给定 E_{eic} 的 $M\neq Null\&\&N\neq Null$,那么 2 个 F_{efi} 对应的 C_{old} 和 C_{new} 分别相同,则 2 个 F_{efi} 为 F_{efi}^{same};如果给定 E_{eic} 的 $M\neq Null\&\&N=Null$,那么 2 个 F_{efi} 对应的 C_{old} 相同,则 2 个 F_{efi} 为 F_{efi}^{same};如果给定 E_{eic} 的 $M=Null\&\&N\neq Null$,那么 2 个 F_{efi} 对应的 C_{new} 相同,则 2 个 F_{efi} 为 F_{efi}^{same}.

本文采用引用关系分析技术识别候选修复实例集合中的同类错误修复实例,以错误字段为中心进行局部依赖分析.本文将 Hajnal 等人[43]提出的定义-使用链分析推广到引用关系分析,利用引用关系分析判定哪些修复实例和待修复错误都是相同的引入错误变更引入的(即具有相同的错误原因),最终输出同类错误修复实例集合.如图 6 所示,是宏 *NETIF_F_HW_VLAN_RX* 对应的待修复出错语句、引入错误变更以及修复实例.旧引用链 $C_{old}=m_e\rightarrow r_e$,其中 m_e 为 EIC 中旧定义字段 *NETIF_F_HW_VLAN_RX*,r_e 为 EFI_1 和 EFI_2 中对旧定义 *NETIF_F_HW_VLAN_RX* 的使用.新引用链 $C_{new}=n_x\rightarrow s_x$,其中 n_x 为 EIC 中新定义字段 *NETIF_F_HW_VLAN_CTAG_RX*,s_x 为 EFI_1 和 EFI_2 中对新定义 *NETIF_F_HW_VLAN_CTAG_RX* 的使用.EIC 中 $M\neq Null\&\&N\neq Null$,EFI_1 和 EFI_2 均依赖相同的 EIC,即二者的 C_{old} 和 C_{new} 分别相同,则二者为同类错误修复实例 F_{efi}^{same}.

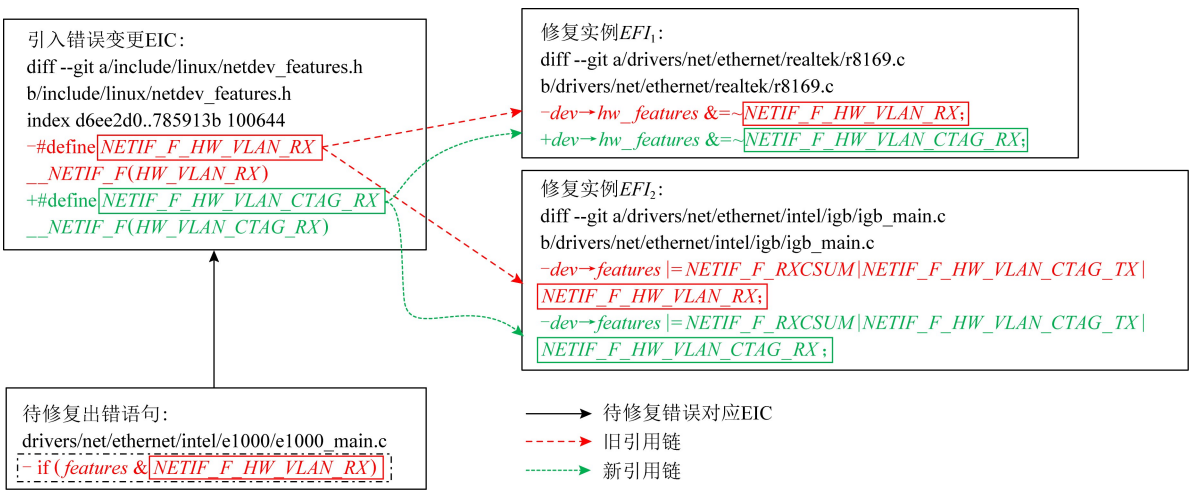

Fig. 6　Identify similar error fix instances based on EIC

图 6　基于 EIC 识别同类错误修复实例

2.4.2　修复模板提取

为了提取针对性修复模板,本文通过计算与待修复错误具有相同原因的 1 组修复实例之间的共性变更抽取修复模板.1 组修复实例提取 1 个修复模板,修复模板抽象了共性变更,而修复模板参数(可能是 1 组或者多组)同时记录了多样化的操作内容.

定义 6. 修复模板 OP.修复模板抽象了 1 组同类错误修复实例的共性变更,OP 表示为:[操作(*Update*,*Delete*,*Add*)][〈操作内容 1@频度,操作内容 2@频度〉].

如果操作是 *Update*,则每个修复实例包含 1 对操作内容.修复实例中相同的操作内容合并计算频度作为权重,相互不同的操作内容权重为 1,因此提取的修复模板可能包含 1 对或者多对操作内容.如果操作是 *Delete*,则每个修复实例包含 1 个单独的操作内容,根据操作内容的异同合并计算频度,因此最终提取的修复模板可能包含 1 个或者多个操作内容.如果操作是 *Add*,则每个修复实例包含 1 个附带相对索引位置的单独操作内容,根据操作内容和附带相对位置索引的异同合并计算频度,最终提取的修复模板可能包含 1 个或者多个附带相对索引位置的操作内容.

如图 7 所示,针对 1 组同类错误修复实例进行修复模板提取.具体包括 2 个步骤:①使用抽象语法树刻画每个修复实例内部的变更差异并将变更差异输出为编辑脚本[44];②比较编辑脚本的共性提取修复模板.本文使用 GumTreeDiff 框架[45]对每个修复实例在 AST 级别进行差异比较,首先将修复实例中包含的语句进行 AST 树形表示,然后进行节点映射和节点差异比较.本文通过映射匹配语句中未修改的相同节点,通过差异比较计算未匹配节点的变更并输出对应编辑脚本.如果未匹配节点只有 1 个,则直接输出该节点在语句迁移中的差异操作.如果存在多个未匹配节点则会对应 1 组语句迁移的操作序

列,需要进一步将相邻的操作和节点进行合并.具体将相邻的相同操作合并,本文仅处理未匹配节点可以合并为子树的情形,输出对应操作针对子树的差异变更.比较编辑脚本的共性提取修复模板,本文通过合并编辑脚本中的操作并计算操作内容的频度形成修复模板.

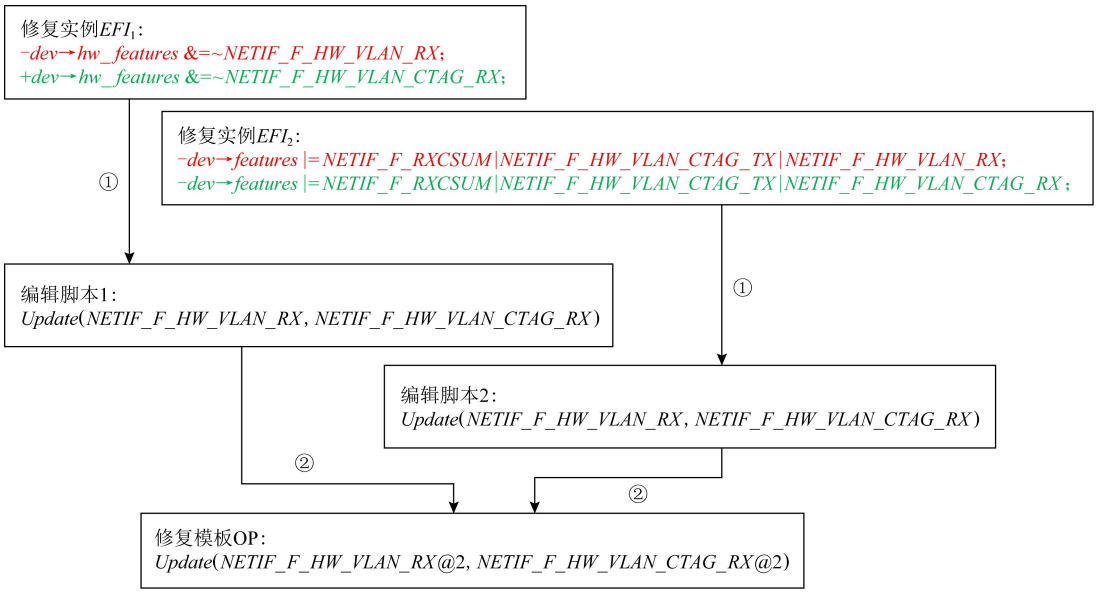

Fig. 7 Example of fix pattern extraction

图 7 修复模板提取示例

2.5 补丁推荐

2.5.1 补丁生成和排序

整个补丁推荐过程采用迭代方式逐个接口错误进行补丁推荐.首先利用已经提取的针对性修复模板生成待推荐补丁,然后对补丁进行排序产生补丁推荐列表.其中,待推荐补丁生成是将针对性修复模板实例化的过程.利用 2.2.1 节的方法确定出错接口语句行的位置,然后针对该出错语句使用修复模板进行代码变更.由于 2.4 节中识别的同类修复实例和待修复错误具有相同原因,同类错误共享 1 个针对性修复模板,因此同类错误修复实例中提取的修复模板具有指向性.

为了生成待推荐补丁,本文提取并选择 2.4.2 节生成的修复模板,通过修复模板匹配错误语句的语句内修改点,使用模板附带的操作内容参数具体化修复模板生成待推荐补丁.修复模板如果是 *Update* 或者 *Delete*,则使用操作内容字段进行匹配;修复模板如果是 *Add*,则使用相对索引位置进行匹配.生成待推荐补丁的过程中采用编辑脚本技术[44],在抽象语法树级别实现出错语句的转换并生成待推荐补丁.待推荐补丁生成过程中可能会合成位置错误或者语法错误的补丁,本文将编译出错的待推荐补丁直接抛弃.根据修复模板附带的操作内容的不同情况,本文的方法是输出 0 个、1 个或者多个待推荐补丁,如果没有输出待推荐补丁,则推荐失败.

针对待推荐补丁的排序,本文采用操作内容的频度和相似度计算相结合的方式进行排序.本文刻画的修复模板提取了同类修复实例的共性变更操作,同时以参数形式记录了修复实例中使用的操作内容以及频度.本文的方法很自然地使用修复模板的参数频度对最终待推荐补丁列表进行排序,对于排序位置相同的补丁,再依据变更字段和待修复驱动名称的相似度进行二次排序.经过排序计算,最后输出 top-N(实验中,$N=5$)的补丁列表进行推荐.

2.5.2 补丁质量判定

针对推荐补丁的质量判定,本文采用人工方式确认有效补丁.首先针对每款待移植驱动程序设定历史上已经完成移植的一个驱动程序版本为正确的参考程序版本;然后依据参考程序版本中的对应代码语句和推荐的补丁进行人工对比语义和相似性,逐个出错接口和对应的推荐补丁依次判定补丁的有效性.为了尽可能保证人工判断的准确性,本文采用 2 人分别进行手工判断的方式,如果 2 人判断一致

则记为正确补丁,如果 2 人判断不一致则直接记为错误补丁.

3 实　　验

本节介绍实验数据集和实验环境,以及相关实验结果和分析.为了检验方法的有效性,本文的实验期望回答 4 个问题:

1) 本文的方法针对真实驱动移植场景中的接口不一致错误进行补丁推荐的正确率?

2) 本文的方法是否相比传统方法的补丁推荐正确率有显著提高?

3) 本文的方法针对各个驱动程序在获取引入错误变更信息之后推荐有效补丁的比例?

4) 本文的方法进行补丁推荐的性能?

3.1 数据集建立

本文的实验在真实的前向移植场景中进行,目标是检验旧版本驱动程序的原有功能在新版本内核环境中复用时,针对出错接口的补丁推荐情况.为了方便检验方法的性能和有效性,本文采用合并在 Linux 内核主线分支中的驱动程序历史数据进行实验,选择实验对象中的驱动程序和内核环境依据 3 个标准.

1) 内核版本从 K_1 升级到 K_2,K_1 环境下的待移植驱动 D_1 在升级过程中已经存在 K_2 环境下对应的驱动程序的升级版本 D_2,从 D_1 到 D_2 的前向演化过程中引入的变化存在内核接口调用的变更,本文的研究只关注移植过程中发生的接口调用不一致问题.

2) 待移植驱动 D_1 在前向演化过程中所发生的接口调用变化,在 K_1 到 K_2 内核区间的 commit 提交信息中存在对应引入错误变更信息以及其他驱动对该接口调用的修复实例.

3) 待移植驱动 D_1 在原来的旧内核版本 K_1 上编译正常通过,而 D_1 在待移植的新内核版本 K_2 上编译产生的编译错误日志包含接口调用错误.

本文通过这 3 个标准并结合实际实验资源的综合考虑,筛选和收集待移植的驱动程序数据集.确定版本区间 K_1:Linux-3.5,K_2:Linux-4.0 版本,该区间总计跨越 43 个 Linux kernel 的小版本.最终收集了 Linux-3.5 对应的 19 款驱动程序,总计包含 142 个文件和 159 304 行代码.表 1 展示了本文收集的待移植驱动程序数据集的具体情况.

Table 1　Data set of Driver to Port

表 1　待移植驱动程序的数据集

类别	驱动名称	文件数量	代码量/行
网卡	e1000	8	17 268
	ixgbe	29	31 264
	virtio	8	2 958
	bonding	10	13 634
	vmxnet3	5	6 480
	dummy	1	209
	ifb	1	312
	tun	1	1 695
	veth	1	459
块设备	floppy	1	4 590
	loop	2	2 002
	mtip32xx	3	4 893
	nvme	1	1 740
	pktcdvd	1	3 116
声卡	ac97	8	8 343
	mpu401	3	932
显卡	i915	57	58 221
	vfb	1	610
	mdacon	1	578
合计		142	159 304

3.2 实验设计

我们实现了算法 1 以及本文的补丁推荐方法,形成了驱动移植场景中的接口补丁推荐工具 RCFix.采用本文的 EIC 检索算法获取待修复接口错误对应的引入错误变更信息,采用 git log-S 构造检索获取候选修复实例集合,依据相同原因识别同类错误修复实例.然后基于 GumTreeDiff 框架对修复实例进行 AST 映射和差异比较,通过比较编辑脚本的共性差异提取修复模板.其中代码仓库采用 git-2.7.4 版本,基于 AST 的差异比较采用 gumtree-2.1.2 版本,AST 解析前端采用 cgum-1.0.1 版本.由于本文提取的修复模板具有指向性,提取并选择针对性修复模板并使用操作内容参数进行具体化,采用编辑脚本技术合成待推荐补丁,利用操作内容的频度对补丁进行排序并推荐给开发者人工判定补丁质量.实验中,对收集的 19 款驱动分别进行前向移植实验,针对以上驱动从旧内核 K_1:Linux-3.5 版本移植到新内核 K_2:Linux-4.0 版本过程中产生的接口错误进行补丁推荐.历史开发信息 git 仓库采用 Linux 官方主分支信息,使用 RCFix 工具依次对每个驱动

移植中产生的接口调用错误生成补丁并推荐补丁，最后人工确认有效补丁并分析实验效果.本文的全部实验使用 ubuntu 16.06 64 位操作系统环境，编译环境使用 gcc-5.4.0 版本，硬件采用 1 台 PC 机器，具有 4 核 3.20 GHz Intel Core™ i5 处理器和 8 GB 内存.

3.3 实验结果

问题 1. 本文的方法针对真实驱动移植场景中的接口不一致错误进行补丁推荐的正确率？

表 2 展示了本文的实验结果，实验中针对 19 款驱动程序进行前向移植，针对移植过程中的接口不

Table 2 Detailed Distribution of Patches Recommendation

表 2 补丁推荐的细节情况分布

驱动程序	宏定义错误		接口名称错误		接口类型错误		修饰符错误		缺少参数错误		多余参数错误	
	有效补丁	推荐补丁	有效补丁	推荐补丁	有效补丁	推荐补丁	有效补丁	推荐补丁	有效补丁	推荐补丁	有效补丁	推荐补丁
e1000	3	4	3	3	0	0	2	2	1	1	0	0
ixgbe	2	2	6	9	0	0	1	1	1	1	1	1
virtio	3	3	0	1	2	2	2	2	1	1	0	0
bonding	3	3	0	7	0	0	0	0	2	5	0	0
vmxnet3	4	4	2	2	0	0	0	0	1	1	0	0
dummy	0	0	0	0	0	0	0	0	1	1	0	0
ifb	0	0	2	2	0	0	0	0	1	1	0	0
tun	4	5	0	5	0	0	0	0	2	3	0	0
veth	0	0	2	2	0	0	0	0	0	0	0	0
floppy	1	2	0	0	0	0	0	0	0	0	0	0
loop	0	0	2	4	0	0	0	0	0	0	0	1
mtip32xx	0	0	2	2	0	0	0	0	0	0	0	0
nvme	3	3	0	0	0	0	0	8	0	0	0	0
pktcdvd	0	0	1	2	0	0	0	0	0	1	0	0
ac97	0	0	0	0	1	1	0	0	0	0	0	0
mpu401	1	1	1	1	0	0	2	2	0	0	0	0
i915	0	0	0	0	0	0	1	1	1	1	0	0
vfb	1	1	0	0	0	0	2	2	0	0	0	0
mdacon	0	0	1	1	0	0	0	0	0	0	0	0
合计	25	28	22	41	3	3	10	18	11	16	1	2

驱动程序	参数类型错误		top-1		top-2		top-5		获取的有效 EIC 数量	接口错误的总数量
	有效补丁	推荐补丁	有效补丁	推荐补丁	有效补丁	推荐补丁	有效补丁	推荐补丁		
e1000	0	0	8	9	8	9	9	9	10	10
ixgbe	0	0	10	11	11	11	11	11	11	14
virtio	0	0	6	8	7	8	8	8	8	9
bonding	0	0	4	8	4	8	5	8	8	15
vmxnet3	0	0	6	7	6	7	7	7	7	7
dummy	0	0	1	1	1	1	1	1	1	1
ifb	0	0	3	3	3	3	3	3	3	3
tun	0	1	6	10	6	10	6	10	14	14
veth	0	0	2	2	2	2	2	2	2	2
floppy	0	0	1	1	1	1	1	1	2	2
loop	0	0	2	2	2	2	2	2	4	5
mtip32xx	0	0	2	2	2	2	2	2	2	2
nvme	0	0	2	3	2	3	3	3	3	11
pktcdvd	0	0	1	2	1	2	1	2	2	3
ac97	0	0	1	1	1	1	1	1	1	1
mpu401	0	0	3	4	3	4	4	4	4	4
i915	0	0	2	2	2	2	2	2	2	2
vfb	0	0	3	3	3	3	3	3	3	3
mdacon	0	0	1	1	1	1	1	1	1	1
合计	0	1	64	80	66	80	72	80	88	109

一致错误进行补丁推荐.实验中包括 7 类接口错误:宏定义错误、接口名称错误、接口类型错误、修饰符错误、缺少参数错误、多余参数和参数类型错误.实验总计包括 109 个接口调用不一致错误,针对其中 80 个错误生成了待推荐补丁,其中 72 个补丁人工确认正确.推荐的正确补丁通过和基准版本驱动程序中的对应代码进行人工比较确定语义相同并进行双人交叉确认.

本实验计算了 top-1,top-2,top-5 的补丁推荐正确率,分别为 58.7%,60.6%,66.1%.本文的主要特点在于依据错误原因和来源识别同类修复实例,然后提取并选择修复模板实现同类待修复错误的高质量补丁推荐.其中的错误原因和来源,即待修复错误对应的引入错误变更信息.实验中包括 109 个接口调用不一致错误,针对其中 88 个错误获取了有效的引入错误变更信息,引入错误变更信息提取的正确率达到 80.7%.本文的方法推荐高质量的适配性补丁,同时也会附上待修复错误对应的引入错误变更信息,更方便开发者理解错误原因和推荐的补丁.

问题 2. 本文的方法是否相比传统方法的补丁推荐正确率有显著提高?

针对驱动移植场景中的接口不一致错误,传统补丁推荐方法 DriverFix[46] 通过相似度识别同类错误修复实例提取和选择修复模板,并通过区分共性和个性代码元素生成待推荐补丁.本文将基于错误根因的补丁推荐方法与基于相似度的传统补丁推荐方法进行对比实验,表 3 展示了本文的对比实验结果.

Table 3　Comparison of Experimental Results with Traditional Patch Recommendation Methods

表 3　与传统补丁推荐方法的对比实验结果

驱动程序	基于相似度的传统补丁推荐方法 DriverFix		本文基于错误根因的补丁推荐方法 RCFix		接口错误数量
	top-1 有效补丁/推荐补丁	top-5 有效补丁/推荐补丁	top-1 有效补丁/推荐补丁	top-5 有效补丁/推荐补丁	
e1000	9/9	9/9	8/9	9/9	10
ixgbe	6/6	6/6	10/11	11/11	14
virtio	6/6	6/6	6/8	8/8	9
bonding	4/5	4/5	4/8	5/8	15
vmxnet3	3/5	4/5	6/7	7/7	7
dummy	0/1	1/1	1/1	1/1	1
ifb	1/2	1/2	3/3	3/3	3
tun	5/6	5/6	6/10	6/10	14
veth	0/1	1/1	2/2	2/2	2
floppy	1/2	1/2	1/1	1/1	2
loop	1/2	1/2	2/2	2/2	5
mtip32xx	0/1	1/1	2/2	2/2	2
nvme	1/1	1/1	2/3	3/3	11
pktcdvd	1/2	1/2	1/2	1/2	3
ac97	1/1	1/1	1/1	1/1	1
mpu401	4/4	4/4	3/4	4/4	4
i915	1/2	1/2	2/2	2/2	2
vfb	2/2	2/2	3/3	3/3	3
mdacon	0/1	1/1	1/1	1/1	1
合计	46/59	51/59	64/80	72/80	109

在相同的驱动移植场景下,通过相同数据集进行对比实验.相比驱动移植接口补丁推荐方法 DriverFix(top-5,51/109,正确率为 46.8%),本文基于错误根因的方法 RCFix(top-5,72/109,正确率为 66.1%)推荐补丁的正确率有显著提高,有效降低了人工修复错误的工作量并提高了修复效率.

另外,还有一些补丁推荐方法针对静态分析工具发现的错误、commit 提交的错误代码等不同场景

的研究对象,本文引用这些方法的数据也进行了比较.相比传统的补丁推荐方法 Getafix[47](top-5,526/1268,正确率为 41.5%)、CLEVER[48](34/72,正确率为 47.2%)和 iFixR[49](top-5,8/13,正确率为 61.5%),本文的方法推荐补丁的正确率(top-5,72/109,正确率为 66.1%)有显著提高.

问题 3. 本文的方法针对各个驱动程序在获取引入错误变更信息之后推荐有效补丁的比例?

图 8 展示了针对各个驱动程序在获取引入错误变更信息之后推荐有效补丁的比例,横轴是 19 个驱动程序,纵轴为 top-N 补丁修复错误的数量、获取有效 EIC 的错误数量和待修复接口错误的数量.

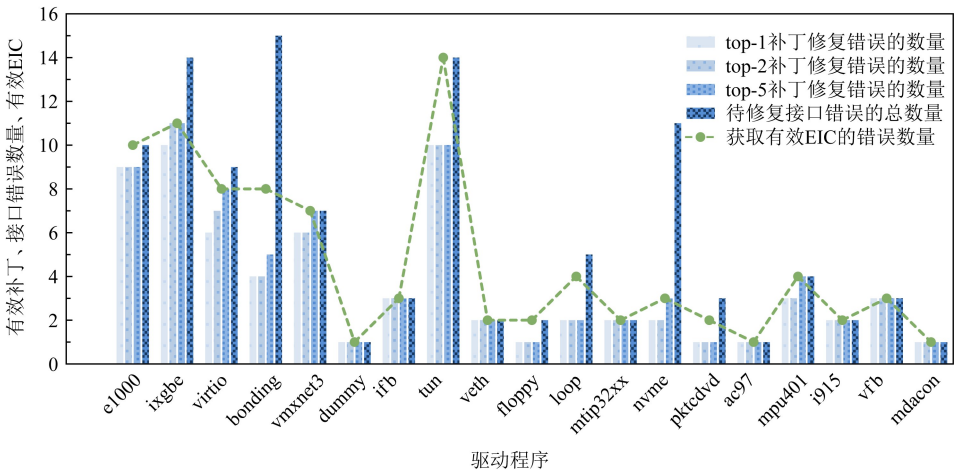

Fig. 8 Proportion of recommending effective patches for each driver

图 8 针对各个驱动程序推荐有效补丁的比例

我们分析了实验中获得的数据分布情况,各个驱动错误代码、推荐的补丁和对应 EIC 原因等详细情况.一方面,本文通过算法 1 自动获取 EIC 原因信息,减少人工的工作量;另一方面,针对大多数驱动程序获取根因和推荐有效补丁的比例较高,提升了补丁推荐的质量和自动化程度.例如,图 8 中针对大多数规模较小的驱动程序,获取根因信息和补丁推荐质量相对较高.

针对少数规模较大的驱动程序,相比规模较小的驱动程序更不易获取有效 EIC 原因信息.例如,驱动 ixgbe,bonding,nvme 等规模较大的程序获取有效 EIC 原因信息的比例相对较低.我们分析了其中的主要原因,这些规模较大的驱动程序本身比较复杂,例如存在内核导出符号(EXPORT_SYMBOL)、变化较大的接口不一致错误等不易获取对应的 EIC 原因信息.其次,获取有效 EIC 原因信息之后,还要识别同类修复实例,提取针对性修复模板和确定操作内容具体化模板进行有效补丁推荐,这些步骤最终都会影响推荐有效补丁的比例.例如,tun,loop 等驱动程序在获取有效 EIC 原因之后补丁推荐的正确率依然相对较低,具体在 3.4 节详细分析和说明.

问题 4. 本文的方法进行补丁推荐的性能?

图 9 统计了 19 款驱动程序进行适配性接口补丁推荐所消耗的时间,横坐标为实验中的 19 款驱动程序,纵坐标为生成待推荐补丁消耗的时间,总体上各个驱动生成待推荐补丁的数量和耗时成正比.

使用本文的接口补丁推荐工具 RCFix,在实验中主要时间消耗包括:编译以及编译错误日志解析、引入错误变更信息搜索、候选修复实例搜索、修复模板提取、待推荐补丁生成和排序等部分.已有研究[8]提供驱动移植的辅助示例,人工利用该方法输出的辅助信息构造补丁平均每个错误耗时在 30 min 左右.相比该方法,一方面本文的方法直接推荐补丁而并非辅助示例,能够有效降低开发者人工修复错误的工作量;另一方面本文的方法针对每个错误的补丁推荐平均耗时大约在 3 min,针对每款驱动程序补丁推荐的平均耗时大约在 14 min,显著提高了辅助人工修复的效率.

3.4 讨 论

本节对本文的方法进行讨论,主要讨论补丁推荐失败的详细情况.本文的方法中搜索待修复错误对应的 EIC 原因信息、识别同类修复实例,提取针对性修复模板和确定操作内容具体化模板等步骤都影响补丁推荐的结果.本文根据实验结果将推荐失败的情形分为 4 类:缺少有效引入错误变更、缺少修复实例、修复模板提取失败和推荐补丁错误.如图 10

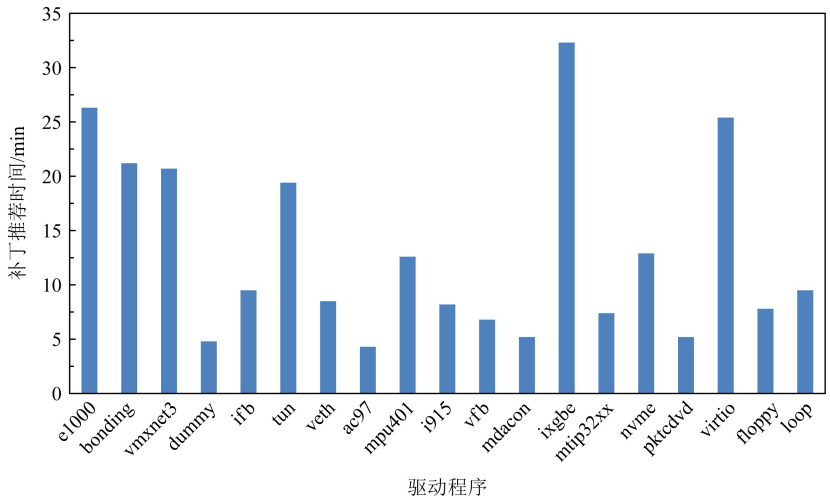

Fig. 9　Performance analysis of patch recommendation

图 9　补丁推荐性能分析

所示,总计 37 个待修复错误补丁推荐失败或者推荐的补丁错误,其中缺少有效引入错误变更(21/37)占比 56.7%,缺少修复实例(5/37)占比 13.5%,修复模板提取失败(3/37)占比 8.1%,推荐补丁错误(8/37)占比 21.6%.

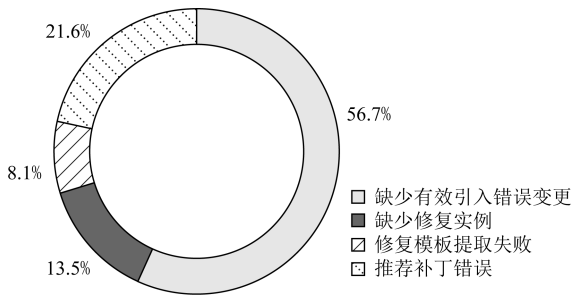

Fig. 10　Analysis of four types of patch recommendation failures

图 10　补丁推荐失败的 4 种类型分析

我们分析了补丁推荐失败的 4 类具体情况,这 4 类情形是递进关系.缺少有效引入错误变更的主要原因是一些规模较大的驱动程序获取对应的引入错误变更信息相对困难,因为规模较大的驱动程序本身比较复杂并且包含有些复杂的调用.而缺少修复实例的情形主要是一些内核接口被驱动程序使用的频度较小,针对这些相对不常用的内核接口及产生的不一致错误,驱动的历史开发信息中客观上几乎没有对应的修复实例,因此在候选修复实例搜索的方法层面也难以找到.修复模板提取失败和补丁推荐错误这 2 种推荐失败的情形,本文通过失败案例 1 和失败案例 2 进行详细说明.

失败案例 1. 修复模板提取失败.

错误为(drivers/block/floppy.c,970):

PREPARE_WORK(&*floppy_work*,(*work_func_t*)*handler*); /* 错误代码 */

floppy_workfn=*handler*; /* 期望的正确补丁 */

引入错误变更为(include/linux/workqueue.h):

−#define *PREPARE_WORK*(*_work*,*_func*)

−　do{

−　　(*_work*)→*func*=(*_func*);

−　}while(0)

修复实例 1 为(drivers/staging/fwserial/fwserial.c):

− *PREPARE_WORK*(&*peer*→*work*,*fwserial_handle_plug_req*);

+*peer*→*workfn*=*fwserial_handle_plug_req*;

修复实例 2 为(drivers/block/nvme-core.c):

−*PREPARE_WORK*(&*dev*→*reset_work*,*nvme_remove_disks*);

+*dev*→*reset_workfn*=*nvme_remove_disks*;

失败案例 2. 推荐补丁错误.

错误为(drivers/net/bonding/bond_main.c,1245):

err=*__netpoll_setup*(*np*); /* 错误代码 */

err=*__netpoll_setup*(*np*,*slave*→*dev*);

/* 期望的正确补丁 */

引入错误变更为(include/linux/netpoll.h):

−int*__netpoll_setup*(struct *netpoll* * *np*);

+int *__netpoll_setup*(struct *netpoll* * *np*,

struct *net_device* * *ndev*);

修复实例 1 为(net/8021q/vlan_dev.c):

$-err=\_\_netpoll\_setup(netpoll)$;

$+err=\_\_netpoll\_setup(netpoll,real\_dev)$;

修复实例 2 为(net/core/netpoll.c):

$-err=\_\_netpoll\_setup(np)$;

$+err=\_\_netpoll\_setup(np,ndev)$.

修复模板提取失败的情形,主要原因是模板提取过程中需要进行 AST 级别的细粒度映射和比较,一些情形会存在映射混乱导致提取失败.在失败案例 1 中,待修复错误是驱动 floppy 中 *PREPARE_WORK* 宏错误,错误原因是 Linux 内核更新后该宏定义被删除.该案例中虽然找到了有效的同类修复实例,但是提取修复模板失败.主要原因在于修复实例中的 2 条语句在 AST 映射阶段出现混乱,无法生成对应的编辑脚本导致提取修复模板失败.现有的映射方法依靠语句中元素的父子结构相对位置和字段名称进行映射,而语句 *PREPARE_WORK*(&*peer*→*work*, *fwserial_handle_plug_req*)和语句 *peer*→*workfn*=*fwserial_handle_plug_req* 是完全不同的 2 种语句结构,并且其中包含的字段也出现大量变化导致映射失败.

推荐补丁错误的情形,主要是具体化修复模板的过程中需要确定正确的操作内容,本文通过频度和驱动类型相似性对操作内容的排序方式,针对驱动程序中接口调用使用个性化参数(例如局部变量)的情形依然具有局限性.在失败案例 2 中,待修复错误是驱动 bonding 中函数 *__netpoll_setup* 错误,错误原因是 Linux 内核更新后该函数定义进行了变更,增加了第 2 个参数.该案例中虽然找到了有效的同类修复实例同时成功提取修复模板.但是由于各个实例中均使用个性化参数作为接口新增的实参,例如 *ndev*, *real_dev* 等私有变量作为实参的情形,针对该情形确定的实参 *slave*→*dev* 不正确,导致推荐补丁错误.综上所述,本文的方法依然有提升空间,例如在更复杂的模板提取和刻画、确定个性化代码元素等操作内容方面还需要更进一步的突破.

4 相关工作

4.1 程序自动修复

程序自动修复是一个自动修复错误的系统,输出的补丁经过程序规约的正确性判定,在程序规约完整的假设下保证了输出补丁的质量.针对一般错误问题,基于随机变异或者修复模板生成候选补丁,通过生成-检测(generate-and-validate)结构利用测试集自动判定候选补丁的质量,目标是输出通过全部测试用例的程序补丁.Le 等人[16]提出 GenProg 方法,假设修复错误的代码碎片存在于待修复程序所在的源代码中,利用遗传算对这些代码碎片进行交叉变异生成候选补丁并使用测试集检测输出补丁的正确性.Kim 等人[50]提出 PAR 方法,假设修复错误的素材存在于其他项目的源代码中,通过人工定义修复模板并随机选择约束变异操作生成候选补丁,克服了基于遗传算法的随机变异产生无意义候选补丁的不足.Le 等人[25]认为不同修复模板的能力并不相同,优先选择高频修复模板生成候选补丁,相比 PAR 方法提高了补丁生成的精度.Xin 等人[32]提出 ssFix 方法,针对规模较大的程序中包含的错误问题,提出基于语法搜索提取相关代码片段进行变换生成候选补丁.Sidiroglou-Douskos 等人[33]和 Ke 等人[34]分别提出 CodePhage 和 SearchRepair,利用语义搜索获取和错误相关的代码片段进行变换生成候选补丁.Wen 等人[27]提出 CapGen 方法,使用比语句级别更细粒度的素材以及利用错误的上下文相似性选择修复模板提高补丁的质量.程序自动修复和补丁推荐虽然在错误定位和补丁质量判定方面存在不同,但是二者都要生成补丁.程序自动修复的对象一般是程序单一版本中的常见错误,具有相应测试集和调试报告用于错误动态定位、补丁适应度检查和补丁质量自动判定等.但是在驱动移植场景中不易收集对应测试集用于补丁正确性判定,并且驱动移植涉及 2 个版本的演化场景,即使旧版本驱动程序有测试集也可能并不适用于新版本.相比使用修复模板的已有方法,通过错误代码形式识别同类错误修复实例提取和选择修复模板,先提取修复模板再利用一些策略选择合适的修复模板生成补丁.本文的方法将修复模板提取和选择合并为一步,从待修复错误原因和来源出发识别同类错误修复实例,进而提取并选择针对性修复模板实现同类待修复错误的补丁推荐.

4.2 补丁推荐

补丁推荐是针对具体错误生成补丁并建议合适补丁的技术.一些错误修复场景适合补丁推荐,这些错误问题没有或者不易获取足够的程序规约对输出的补丁进行质量自动检验.补丁推荐建议的补丁可能完全修复了错误,也可能还需要开发者进行手工修正.尽管补丁推荐技术并不能完全保证有效的修复,但推荐的补丁仍然有助于降低人工修复的工作

量并提高修复效率,因此形成了一系列补丁推荐的相关研究.Bader 等人[47]通过学习静态分析发现的错误对应的修复模板,用于推荐新发生的同类错误修复补丁.该方法将已有修复补丁分割成独立的编辑脚本,然后通过聚类算法对这些编辑脚本进行挖掘并抽象出同类错误的修复模板.最后使用获取的修复模板上下文去匹配待修复错误,并利用修复模板附带的上下文对候选补丁排序完成补丁推荐.相比该方法通过上下文识别同类错误提取修复模板,本文通过错误原因和来源识别同类错误修复实例提取模板.Nayrolles 等人[48]针对风险 commit 提交进行识别并在提交到中央仓库之前进行补丁推荐,使用克隆检测技术对风险提交和一些已知的错误提交进行比较识别潜在的错误.在潜在风险提交错误识别之后进行补丁推荐,该方法直接推荐匹配的已知错误提交对应的补丁.相比该方法推荐匹配的已有补丁,本文通过针对性修复模板生成待推荐补丁.Zhang 等人[15]针对电子商务等应用软件中的代码错误,在长期的开发和迁移过程中这些错误和对应的修复缺少关联关系的数据标记,因此无法直接从已知修复实例学习修复模板并用于推荐未修复的相似错误.针对这个问题,该方法利用代码挖掘技术从历史开发仓库中获取已有错误和对应修复的配对,然后使用基于 API 调用序列相似性识别同类错误并进行抽象形成修复模板数据库,用于类似的未修复错误的补丁推荐.相比该方法通过代码挖掘和聚类算法提取修复模板,本文基于原因和来源识别同类错误修复实例提取模板.Koyuncu 等人[49]针对开发环境中没有测试集的情况下,提出基于缺陷报告(bug reports)的补丁推荐方法.不同于很多传统的补丁生成方法使用测试集进行基于频谱(spectrum-based)的缺陷定位,该方法利用缺陷报告进行基于信息检索(information retrieval-based)的缺陷定位确定错误位置,然后在 PAR[50]的基础上通过人工定义的修复模板生成补丁并进行推荐.

4.3　驱动移植

Rodriguez 等人[6]提出 Coccinelle 自动分析驱动针对兼容库需要修改的变化内容,从而辅助驱动适配兼容库达到移植驱动的目的.Thung 等人[7]通过分析内核 commit 信息,以推荐系统的方式给出驱动移植辅助信息.具体通过遍历移植区间中内核 commit 并找到由于内核提交使得驱动出错的 commit 提交点,然后在该提交点 commit 中进一步分析移植相关的代码变化用于辅助后向移植,主要推荐的变化示例包括更改记录访问、删除函数参数、函数名的变化、常数变化和 IF 条件变化等类型.Lawall 等人[8]发现驱动移植中的一个错误问题所需的辅助信息可能来自多条开发历史提交数据,针对该类问题,其结合编译信息和出错源码信息提取检索关键字,再利用补丁查询语言(patch query language, PQL)查询开发历史提交数据获得移植所需的辅助信息.相比 git 查询和 Google 搜索,该方法通过加强检索条件并采用 PQL 查询进一步提高了辅助信息的精度和检索效率.已有方法给出了驱动移植中一般错误问题的参考信息和推荐示例,一定程度上减轻了驱动移植的负担.而本文的方法针对驱动移植中的接口调用错误问题,通过原因识别同类修复实例实现高质量的补丁推荐.相比仅提供辅助示例的方法,本文的方法推荐高质量补丁进一步降低人工修复的工作量并提高了修复效率.

5　总结和展望

本文提出了一种基于错误根因的补丁推荐方法,通过相同的错误根因和来源识别同类错误修复实例,通过同类错误修复实例提取并选择针对性修复模板实现同类待修复错误的高质量补丁推荐.一方面,相比传统方法通过错误代码形式的相似性识别同类错误修复实例,本文针对驱动移植场景中不同调用点所在同类错误语句类型和上下文等具体表现形式不相似的问题特点,通过相同的错误根因和来源识别同类错误修复实例的思路比较新颖.另一方面,本文通过待修复错误的原因识别同类修复实例,其隐含的修复模板是唯一的,从而将提取和选择修复模板合二为一.相比传统的补丁推荐方法,本文的方法推荐补丁的正确率有显著提高,有效降低了人工修复的工作量并提高了修复效率.本文除了推荐高质量补丁之外还附带相应的错误原因信息,这些信息能够辅助开发者更好地理解错误和推荐的补丁.

未来,我们计划在 2 个方面开展进一步的研究,我们希望使用机器学习的方法提高相同原因的同类错误修复实例识别正确率.我们还计划针对更复杂的错误进一步研究修复模板提取和修复模板刻画方法,例如修复实例中的语句包含多点修改、相互关联的接口调用以及变化较大的接口不一致错误等情形.

作者贡献声明:李斌提出研究问题,设计方法步骤以及完成实验,撰写论文初稿;贺也平负责把握研究思路,组织技术讨论和全文修订;马恒太和芮建武负责对技术路线进行讨论和分析;李晓卓负责对部分实验数据分析和实验结果整理.

参 考 文 献

[1] Asim K, Michael M S. Understanding modern device drivers [C] //Proc of the 7th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2012: 87-98

[2] Chou A, Yang Junfeng, Chelf B, et al. An Empirical Study of Operating Systems Errors [M]. New York: ACM, 2001

[3] Padioleau Y, Lawall J L, Muller G, Understanding collateral evolution in Linux device drivers [C] //Proc of the 1st ACM SIGOPS/EuroSys European Conf on Computer Systems. New York: ACM, 2006: 59-71

[4] Padioleau Y, Lawall J, Hansen R R, et al. Documenting and automating collateral evolutions in Linux device drivers [C] //Proc of the 3rd ACM SIGOPS/EuroSys European Conf on Computer Systems. New York: ACM, 2008: 247-260

[5] Brunel J, Doligez D, Hansen R R, et al. A foundation for flow-based program matching using temporal logic and model checking [C] //Proc of the 36th Annual ACM SIGPLAN-SIGACT Symp on Principles of Programming Languages. New York: ACM, 2009: 114-126

[6] Rodriguez L R, Lawall J. Increasing automation in the backporting of Linux drivers using coccinelle [C] //Proc of the 11th European Dependable Computing Conf. Piscataway, NJ: IEEE, 2015: 132-143

[7] Thung F, Le X B D, Lo D, et al. Recommending code changes for automatic backporting of Linux device drivers [C] //Proc of IEEE Int Conf on Software Maintenance and Evolution. Piscataway, NJ: IEEE, 2017: 222-232

[8] Lawall J, Palinski D, Gnirke L, et al. Fast and precise retrieval of forward and back porting information for Linux device drivers [C] //Proc of the 2017 USENIX Conf on Annual Technical Conf. Berkeley, CA: USENIX Association, 2017: 15-26

[9] Tao Yida, Jindae K, Sunghun K, et al. Automatically generated patches as debugging aids: A human study [C] //Proc of the 22nd ACM SIGSOFT Int Symp on Foundations of Software Engineering. New York: ACM, 2014: 64-74

[10] Xuan Jifeng, Ren Zhilei, Wang Ziyuan, et al. Progress on approaches to automatic program repair [J]. Journal of Software, 2016, 27(4): 771-784 (in Chinese)
(玄跻峰, 任志磊, 王子元, 等. 自动程序修复方法研究进展 [J]. 软件学报, 2016, 27(4): 771-784)

[11] Wang Zan, Gao Jian, Chen Xiang, et al. Automatic program repair techniques: A survey [J]. Chinese Journal of Computers, 2018, 41(3): 588-610 (in Chinese)
(王赞, 部健, 陈翔, 等. 自动程序修复方法研究述评[J]. 计算机学报, 2018, 41(3): 588-610)

[12] Goues C L, Forrest S, Weimer W. Current challenges in automatic software repair [J]. Software Quality Journal, 2013, 21(3): 421-443

[13] Martin M. Automatic software repair: A bibliography [J]. ACM Computing Surveys, 2018, 51(1): 1-24

[14] Li Bin, He Yeping, Ma Hengtai. Automatic program repair: Key problems and technologies [J]. Journal of Software, 2019, 30(2): 244-265 (in Chinese)
(李斌, 贺也平, 马恒太. 程序自动修复: 关键问题及技术 [J]. 软件学报, 2019, 30(2): 244-265)

[15] Zhang Xindong, Zhu Chenguang, Li Yi, et al. Precfix: Large-scale patch recommendation by mining defect-patch pairs [C] //Proc of the 42nd ACM/IEEE Int Conf on Software Engineering: Software Engineering in Practice. New York: ACM, 2020: 41-50

[16] Le G C, Nguyen T, Forrest S, et al. GenProg: A generic method for automatic software repair [J]. IEEE Transactions on Software Engineering, 2012, 37(1): 54-72

[17] Weimer W, Nguyen T V, Goues C L, et al. Automatically finding patches using genetic programming [C] //Proc of the 31st Int Conf on Software Engineering. New York: ACM, 2009: 364-374

[18] Goues C L, Dewey-Vogt M, Forrest S, et al. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each [C] //Proc of the 34th Int Conf on Software Engineering. New York: ACM, 2012: 3-13

[19] Qi Yuhua, Mao Xiaoguang, Lei Yan, et al. The strength of random search on automated program repair [C] //Proc of the 36th Int Conf on Software Engineering. New York: ACM, 2014: 254-265

[20] Long Fan, Peter A, Martin R. Automatic inference of code transforms for patch generation [C] //Proc of the 11th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2017: 727-739

[21] Liu Xuliang, Zhong Hao. Mining stackoverflow for program repair [C] //Proc of the 25th IEEE Int Conf on Software Analysis, Evolution and Reengineering. Piscataway, NJ: IEEE, 2018: 118-129

[22] Liu Kui, Kim D, Bissyandé T F, et al. Mining fix patterns for findbugs violations [J]. IEEE Transactions on Software Engineering, 2021, 47(1): 165-188

[23] NguyenH A, Nguyen T N, Dig D, et al. Graph-based mining of in-the-wild, fine-grained, semantic code change patterns [C] //Proc of the 41st Int Conf on Software Engineering. New York: ACM, 2019: 819-830

[24] Koyuncu A, Liu Kui, Bissyandé T F, et al. FixMiner: Mining relevant fix patterns for automated program repair [J]. Empirical Software Engineering, 2020, 25(3): 1980-2024

[25] Le X B D, Lo D, Goues C L. History driven program repair [C] //Proc of the 23rd IEEE Int Conf on Software Analysis, Evolution and Reengineering. Piscataway, NJ: IEEE, 2016: 213-224

[26] Jiang Jiajun, Xiong Yingfei, Zhang Hongyu, et al. Shaping program repair space with existing patches and similar code [C] //Proc of the 27th ACM SIGSOFT Int Symp on Software Testing and Analysis. New York: ACM, 2018: 298-309

[27] Wen Ming, Chen Junjie, Wu Rongxin, et al. Context-aware patch generation for better automated program repair[C/OL] //Proc of the 40th Int Conf on Software Engineering. New York: ACM, 2018[2019-01-10]. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8453055

[28] Lutellier T, Pham H V, Pang L, et al. CoCoNuT: Combining context-aware neural translation models using ensemble for program repair [C] //Proc of the 29th ACM SIGSOFT Int Symp on Software Testing and Analysis. New York: ACM, 2020: 101-114

[29] Li Yi, Wang Shaohua, Nguyen T N. DLFix: Context-based code transformation learning for automated program repair [C] //Proc of the 42nd Int Conf on Software Engineering. New York: ACM, 2020: 602-614

[30] Xiong Yingfei, Wang Jie, Yan Runfa, et al. Precise condition synthesis for program repair [C] //Proc of the 39th Int Conf on Software Engineering. New York: ACM, 2017: 416-426

[31] Saha R K, Lyu Y J, Yoshida H, et al. Elixir: Effective object-oriented program repair [C] //Proc of the 32nd IEEE/ACM Int Conf on Automated Software Engineering. Piscataway, NJ: IEEE, 2017: 648-659

[32] Xin Qi, Reiss S P. Leveraging syntax-related code for automated program repair [C] //Proc of the 32nd IEEE/ACM Int Conf on Automated Software Engineering. Piscataway, NJ: IEEE, 2017: 660-670

[33] Sidiroglou-Douskos S, Lahtinen E, Long Fan, et al. Automatic error elimination by horizontal code transfer across multiple applications [J]. ACM SIGPLAN Notices, 2015, 50(6): 43-54

[34] Ke Yalin, Stolee K T, Goues C L, et al. Repairing programs with semantic code search [C] //Proc of the 30th IEEE/ACM Int Conf on Automated Software Engineering. Piscataway, NJ: IEEE, 2015: 295-306

[35] Hassan F, Wang Xiaoyin. HireBuild: An automatic approach to history-driven repair of build scripts [C] //Proc of the 40th Int Conf on Software Engineering. New York: ACM, 2018: 1078-1089

[36] Lou Yiling, Chen Junjie, Zhang Lingming. History-driven build failure fixing: How far are we [C] //Proc of the 28th ACM SIGSOFT Int Symp on Software Testing and Analysis. New York: ACM, 2019: 43-54

[37] Liu Kui, Koyuncu A, Kim D, et al. TBar: Revisiting template-based automated program repair [C] //Proc of the 28th ACM SIGSOFT Int Symp on Software Testing and Analysis. New York: ACM, 2019: 31-42

[38] Wen Ming, Wu Rongxin, Liu Yepang. Exploring and exploiting the correlations between bug-inducing and bug-fixing commits [C] //Proc of the 27th ACM Joint Meeting on European Software Engineering Conf and Symp on the Foundations of Software Engineering. New York: ACM, 2019: 326-337

[39] Herzig K, Zeller A. The impact of tangled code changes [C] //Proc of the 10th Working Conf on Mining Software Repositories. New York: ACM, 2013: 121-130

[40] Kirinuki H, Higo Y, Hotta K, et al. Hey! Are you committing tangled changes? [C] //Proc of the 22nd Int Conf on Program Comprehension. New York: ACM, 2014: 262-265

[41] Martinez M, Weimer W, Monperrus M. Do the fix ingredients already exist? An empirical inquiry into the redundancy assumptions of program repair approaches [C] //Proc of the 36th Int Conf on Software Engineering. New York: ACM, 2014: 492-495

[42] Scott C, Ben S. Git [OL]. [2020-02-15]. http://git-scm.com/

[43] Hajnal A, Forgacs I. A precise demand-driven def-use chaining algorithm [C] //Proc of the 6th European Conf on Software Maintenance and Reengineering. Piscataway, NJ: IEEE, 2002: 77-87

[44] Chawathe S S, Rajaraman A, Garcia-Molina H, et al. Change detection in hierarchically structured information [C] //Proc of the 1996 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 1996: 493-504

[45] Falleri J R, Mihai C, Georg D, et al. GumTreeDiff [EB/OL]. [2020-02-15]. https://github.com/GumTreeDiff/gumtree

[46] Li Bin, He Yeping, Ma Hengtai, et al. Recommending interface patches for forward porting of Linux device drivers based on existing instances [J]. Journal of Computer Research and Development, 2021, 58(1): 189-207 (in Chinese)
(李斌, 贺也平, 马恒太, 等. 基于已有实例的 Linux 驱动程序前向移植接口补丁推荐[J]. 计算机研究与发展, 2021, 58(1): 189-207)

[47] Bader J, Scott A, Pradel M, et al. Getafix: Learning to fix bugs automatically [C/OL] //Proc of the ACM on Programming Languages. New York: ACM, 2019[2020-03-05]. https://arxiv.org/pdf/1902.06111.pdf

[48] Nayrolles M, Hamou-Lhadj A. CLEVER: Combining code metrics with clone detection for just-in-time fault prevention and resolution in large industrial projects [C] //Proc of the Int Conf on Mining Software Repositories. New York: ACM, 2018: 153-164

[49] Koyuncu A, Liu Kui, Bissyandé T F, et al. iFixR: Bug report driven program repair [C] //Proc of the 27th ACM

Joint Meeting on European Software Engineering Conf and Symp on the Foundations of Software Engineering. New York: ACM, 2019: 314-325

[50] Kim D, Nam J, Song J, et al. Automatic patch generation learned from human-written patches [C] //Proc of the 2013 Int Conf on Software Engineering. New York: ACM, 2013: 802-811

Li Bin, born in 1985. PhD, engineer. His main research intersets include program analysis and repair, safe operating system.

李 斌,1985 年生.博士,工程师.主要研究方向为程序分析和修复、安全操作系统.

He Yeping, born in 1962. PhD, professor. His main research intersets include system security, privacy protection.

贺也平,1962 年生.博士,研究员.主要研究方向为系统安全、隐私保护.

Ma Hengtai, born in 1970. PhD, associate professor. His main research intersets include software security analysis, operating system security.

马恒太,1970 年生.博士,副研究员.主要研究方向为软件安全分析、操作系统安全.

Rui Jianwu, born in 1972. PhD, senior engineer. His main research intersets include operating system, software testing.

芮建武,1972 年生.博士,正高级工程师.主要研究方向为操作系统、软件测试.

Li Xiaozhuo, born in 1992. PhD candidate. Her main research intersets include code analysis and fault location.

李晓卓,1992 年生.博士研究生.主要研究方向为代码分析和缺陷定位.