第44卷 第11期 2017年11月
计 算 机 科 学 COMPUTER SCIENCE
Vol. 44 No. 11 Nov. 2017

基于测试例生成的 Android 应用数据竞争验证方法

佘 艺 唐弘胤 吴国全 陈 伟 魏 峻 黄 涛
(中国科学院软件研究所软件工程技术研发中心 北京 100190)

摘 要 Android 应用是一种事件驱动的并发程序。后台线程与异步事件执行顺序的不确定,导致数据竞争在 Android 应用中普遍存在。现有的针对 Android 应用的竞争检测工具会产生大量误报,且不能确定地重现竞争。针对以上问题,在现有的竞争检测结果的基础上,提出了一种基于测试用例生成的 Android 应用数据竞争验证方法。该方法首先构建应用的状态转化图,并基于状态转化图和现有竞争检测工具的检测结果自动生成包含潜在数据竞争的测试用例,然后在测试用例执行的过程中通过控制事件分发和线程的执行顺序来暴露竞争,观察竞争是否会引起程序异常。实验结果表明,该方法能有效地重现数据竞争引起的并发错误,并指出检测结果中的误报。

关键词 录制/重放,数据竞争,移动应用,Android,测试

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.005

Concurrency Bugs Verification in Android Applications Based on Test Case Generation

SHE Yi TANG Hong-yin WU Guo-quan CHEN Wei WEI Jun HUANG Tao
(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract Android application is an event-based concurrent program. Data race is common in Android apps due to unforeseen threads interleaving coupled with non-deterministic reordering of asynchronous events. Existing race detection tools for Android applications will report false positive, and cannot replay the concurrent bugs caused by race. To address this issue, this paper proposed a new technique to expose race based on the results obtained from existing race detection tools. Our technique firstly builds state transition graph (STG) for targeted application, and generates a test case that has potential data races based on the STG and the results reported by existing race detection tools. Then it reschedules test cases execution by controlling event dispatching and thread interleaving to determine whether such potential races really lead to thrown exceptions. Our experiments show that this technique is effective, and it can confirm and replay concurrency bug caused by real data races, while at the same time eliminates false warnings for Android apps.

Keywords Record/Replay, Data race, Mobile application, Android, Testing

1 引言

随着智能手机、平板电脑等移动触屏设备的普及,用户越来越依赖它们来完成各种复杂的计算任务。截止到 2015 年 5 月,谷歌官方应用商城上的 Android 应用数量已超过了 150 万,包括社交、工具、教育、摄影和音乐等多个种类。用户对移动应用进行广泛使用的同时,对应用的质量也提出了越来越高的要求。

Android 设备通常包含一组传感器并支持多种输入方式,Android 应用需要及时处理来自各种事件源(用户界面、网络、传感器、框架等)的异步事件,以保证应用的响应速度。事件驱动的并发编程模型在移动平台上被广泛使用,这种编程模型能容易地处理各种输入,方便开发灵活的、适应性强的应用,但异步处理方式和多线程的大量使用也带来了数据竞争问题。竞争可能会引起严重的并发错误,导致应用出现异常甚至直接崩溃,而这一类并发错误通常又很难调试和重现,因为它们仅在特殊的条件下才会出现。

程序分析是自动检测数据竞争的有效方法,目前已有很多针对 Android 应用的数据竞争检测方法和工具,它们大致可分为静态和动态两类。静态分析不执行程序,从程序的源代码中推导程序可能的执行路径。如,DEvA[1]是一种用来检测事件驱动的系统中事件异常(两个或者多个事件访问相同的内存位置,并且至少有一个是写访问)的静态分析技术。动态方法采用插桩技术记录程序的执行路径,包括事件的开始/结束、内存的读写等信息。如,EventRacer for android[2]是一种针对 Android 应用的动态竞争检测工具,它实现了一种可扩展的竞争检测算法,基于 Android 并发程序精确的事件发生序(Happen-Before)模型来检测竞争。

虽然已有的竞争检测工具和方法能定位竞争可能出现的位置,在一定程度上为开发人员解决竞争问题提供帮助,但仍存在一些不足:1)静态和动态的检测方法都会出现误报,人工进行确认会浪费开发人员大量的时间;2)无法确定数据竞争

到稿日期:2016-10-11 返修日期:2016-12-30 本文受科技支撑(2015BAF05B01),国家自然科学基金(61472407)资助。
佘 艺(1990-),男,硕士生,主要研究方向为软件测试,E-mail:sheyi14@otcaix.iscas.ac.cn;吴国全(1979-),男,博士,副研究员,主要研究方向为软件工程、程序分析、软件测试等,E-mail:gqwu@otcaix.iscas.ac.cn(通信作者)。

的影响，大部分的数据竞争并不会导致程序异常；3)不能重现竞争，为了修复竞争，开发人员需要花费大量时间来推测竞争出现的场景。

针对以上问题，本文提出了一种基于测试例生成的 Android 应用数据竞争验证方法。该方法首先根据现有的数据竞争检测工具(静态或者动态检测工具)的检测结果得到可能产生数据竞争的程序语句对；其次，动态构建应用的状态转化图，并利用状态转化图自动生成可能产生待验证数据竞争的测试用例；最后，多次执行测试用例，通过控制事件分发和后台线程的执行顺序，观察潜在的数据竞争是否会引起程序异常或者语义不一致，以确认并发错误。本文的主要工作和贡献包括：

1)提出了一种基于测试用例生成的 Android 应用数据竞争验证方法，通过构造包含潜在数据竞争的测试用例，并在测试用例执行过程中控制事件分发和线程的执行顺序来暴露并发错误，去除误报；

2)实验结果表明，对于现有检测工具报告的潜在数据竞争，RacerDroid 能够进行有效的确认，并重现数据竞争引起的并发错误。

本文第 2 节对 Android 应用背景进行介绍；第 3 节通过一个具体的案例来介绍 Android 应用中存在的数据竞争；第 4 节详细介绍本文所提出的基于测试用例生成的数据竞争检测方法；第 5 节介绍工具的具体实现；第 6 节通过实验对方法的有效性进行验证；第 7 节介绍相关工作；最后进行总结。

2 Android 背景

Android 是基于 Linux 内核的移动操作系统。Android 应用位于 Android 体系结构的最上层，通过调用 Android 框架层的服务进行实现。Android 应用逻辑由 Activity 串联，每个 Activity 对应一个用户界面，包括一组 UI 组件，如 button，TextView，EditText 和 ListView 等。

与大多数典型 GUI 框架(如 Swing，SWT)相同，Android 应用也是事件驱动的。Android 中的事件包括用户输入事件、生命周期事件、传感器事件等。对于每种类型的事件，Android 框架都提供了相应的回调函数供开发人员实现，以响应特定的事件。比如，一个 Activity 被创建后，onCreate 回调函数将被调用；当用户点击了某个 Button 时，它的 onClick 回调函数将被调用；当设备的位置发生改变时，onLocationChanged 回调函数将被调用。Android 框架通过调用用户实现的回调函数来响应各种事件，从而驱动应用的运行。

Android 支持多线程，在 Android 的并发模型中，每个应用都有一个主线程(又称“UI 线程”)。主线程主要负责事件的分发，如将用户的输入事件分发到合适的 UI 组件上，将生命周期事件分发给对应的 Android 组件(Activity，Service)等。为了避免无响应的线程阻塞用户界面，Android 规定只能在主线程中访问 UI 组件。非主线程若要访问 UI 组件，需要向主线程的事件队列(又称“消息队列”)发送消息，然后主线程会将对应的事件分发给合适的 UI 组件。Android 中有一类线程被称为 Looper 线程，每个 Looper 线程会关联一个事件队列。Looper 线程会不断检查事件队列中是否有待处理的事件，若有，则按照顺序依次处理；如果队列为空，则进入阻塞状态，直到下一个到来的事件将它唤醒。UI 线程就是一个 Looper 线程。除了 Looper 线程，Android 中还有后台线程(又称“工作线程”)和 Binder 线程。长时间运行的任务或是 CPU 密集型的操作通常运行在后台线程，而 Binder 线程主要用于 IPC(进程间通信)，如 Android 框架中的服务端与客户端通信。

为了方便使用多线程，Android 提供了 AsyncTask 类。AsyncTask 是一种轻量级的异步任务类，它封装了线程池和 Handler(用户事件分发处理的类)，它在线程池中执行异步任务，然后把执行的进度和最终的结果传递给主线程，并在主线程中更新 UI。通过 AsyncTask，可以更加方便地执行后台任务和访问 UI。

Android 应用在运行的过程中，各种事件可能随时出现，因此事件回调函数的执行时机不确定。在事件回调函数中，可能启动新的后台线程来执行耗时的工作。这样，多个事件的回调函数之间、回调函数与后台线程之间以及后台线程之间都可能出现数据竞争。

3 数据竞争示例

下面以 AnyMemo 应用中的一个数据竞争为例来介绍 Android 应用中的数据竞争是如何产生的。AnyMemo 是 Android 平台上一款开源的记忆辅助软件。

图 1 展示了与这个竞争相关的代码片段。

```
public class DownloaderAnyMemo extends DownloaderBase{
  public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    …
    initialRetrieve();
  }
  protected void initialRetrieve(){
    …
    mProgressDialog = ProgressDialog.show(this,
    getString(R.string.loading_please_wait),
    getString(R.string.loading_connect_net),true,true,
    new DialogInterface.OnCancelListener(){
        @Override
        public void onCancel(DialogInterface dialog){
            finish();
        }
    });
    new Thread(){
        public void run(){
            final ArrayList<DownloadItem> list=obtainCategories();
                mHandler.post(new Runnable(){
                  public void run(){
                    …
                    mProgressDialog.dismiss();
                  }
                });
        }
    }.start();
  }
}
```

图 1 AnyMemo 代码片段

当应用跳转到 DownloaderAnyMemo(用于从数据库下载

数据的 Activity)时,在 onCreate 回调函数中将调用 initialRetrieve()来创建一个下载数据的后台线程。在数据下载的过程中,会以进度条的方式向用户通知下载的进度。当下载任务完成后,后台线程会向主线程的事件队列中发送下载完成的消息,接着进度条消失,并在界面上展示下载的数据。检测工具报告后台线程中执行的下载任务与 onDestroy 回调函数之间存在数据竞争。分析代码可知,这是因为 onDestroy 执行回收了 mDialog 组件对象,当后台任务完成后,调用 mDialog. dismiss()方法尝试取消进度条时,Window Manager 找不到关联的 mDialog 组件,从而导致应用抛出异常:"java. lang. IllegalArgumentException: View not attached to window manager"。虽然在大部分情况下,后台任务执行完之后onDestroy()函数才会执行,但由于后台线程是异步执行的,在特殊的场景下,onDestroy()函数可能在后台线程还未执行完成时就先执行了。如果能在应用运行时调换这两个函数的执行顺序,就可确定性地重现这个竞争。

4 方法概述

方法的目标是验证现有静态/动态竞争检测工具(如 EventRacer for Android[2] 或者 DEvA[1])所报告的竞争是否存在,并确定数据竞争的影响。假设检测工具报告的竞争具有如下形式:(ShareVal,〈ATC1,Task1,statement1〉,〈ACT2,Task2,statement2〉),表示在 ACT1 中的 Task1 和 ACT2 中的 Task2 的执行没有确定的先后关系,Task1 中的 statement1 和 Task2 中的 statement2 都访问了共享变量 ShareVal,且其中至少有一个是写访问,它们具有竞争关系。其中,Task1/Task2 可能是异步事件的回调函数或者运行中的后台线程的方法。对于前者,ACT1/ACT2 是 Android 应用组件(如 Activity,Service),如果都是 Activity 类型,则通常指向同一个 Activity。对于后者,ACT1/ACT2 是在 Android 组件中创建的后台线程。statement1/statement2 为方法中执行的语句,也被称为"竞争语句对"。

图 2 描述了该方法的整体流程,主要包含两个步骤:测试用例生成和测试用例重放。第一步根据竞争检测工具的结果(潜在的数据竞争)插桩目标应用,通过自动探测技术构建应用的状态转化图,生成可能出现数据竞争的测试用例。第二步重放测试用例,通过调换事件分发和线程执行的顺序验证竞争是否会导致程序异常。

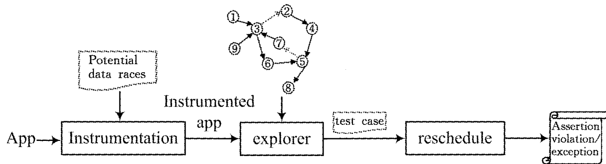

图 2 方法的整体流程图

4.1 测试用例生成

该步骤首先构建应用的状态转化图,并从状态转化图中寻找一条能覆盖竞争语句对的路径,根据该路径生成测试脚本。应用的状态转化图由一组节点和有向边构成。节点代表一个特定的应用状态,边代表引起状态转变的事件序列(用户操作)。状态可表示成一个三元组$\langle A,U,u\rangle$,其中 A 表示是该状态对应的 Activity,U 表示该状态下可见的 UI 组件集合,u 表示实现了自定义事件回调函数的 UI 组件集合。边对应三元组$\langle Ss,Se,Action\rangle$,$Action$ 表示事件,Ss 和 Se 分别表示 Action 发生前、后的源状态和目标状态。状态转化图可以通过多种方法来构建,主要可分为静态程序分析(如文献[3-4])和动态程序分析(如文献[5-6])两类,本文采用基于动态程序分析的自动探测方法。自动探测借助 Instrumentation 启动目标应用,动态获取当前状态下的 UI 组件信息,并根据预定的策略依次触发组件绑定的用户事件(模拟用户操作),从而驱动应用自动运行。每执行完一个用户操作,获取应用的状态信息,并通过状态比较来构建状态转化图。

为生成包含潜在数据竞争的测试用例,在探测 Android 应用的状态空间之前,首先根据现有数据竞争检测工具报告的数据竞争集合,对存在数据竞争的语句对 statement1/statement2 分别进行插桩。在状态空间的探测过程中,当某个事件导致竞争语句 statement1 或 statement2 被执行时,就在构建状态图时标出对应的边。若竞争语句 statement1/statement2 存在于 Android 框架层,方法便根据检测结果中的函数调用关系找到其在应用层的调用语句。方法不对 Android 框架层进行代码插桩,以保证方法能运行在任何 Android 设备上。

有了状态转化图的指导,根据图的搜索算法能找到一条包含竞争语句所对应的两条边的执行路径。将整条路径对应的事件序列解析成测试语句,即可得到最终的测试脚本。图 3 描述了测试用例的生成过程。

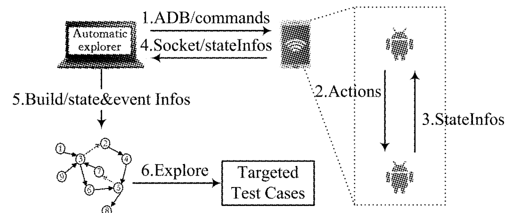

图 3 测试用例生成流程图

4.2 测试重放

该步骤首先执行上一步生成的测试脚本。通过在测试脚本执行的过程中控制事件分发和线程的执行顺序,来确认数据竞争是否为误报。

方法实现了一个测试执行调度器,通过修改现有的 Android 应用测试框架来访问应用主线程的事件队列,从而可以监听特定的事件消息,并控制事件的分发顺序。通过向被测应用的代码中插入同步原语来控制特定后台线程的执行时机。下面结合竞争出现的场景来描述方法的竞争重现方式。

1)如果竞争语句对 statement1/statement2 分别在 Task1/Task2的回调方法中被执行,且在测试用例执行生成的过程中 Task1 先于 Task2 执行,那么将调换事件分发的顺序,使 Task1 在 Task2 之后执行。如果出现 Task1 不执行且 Task2 也不能被执行(Task2 是在 Task1 中创建的)的情况,这说明 Task1 和 Task2 具有确定的执行顺序,竞争在这种场景下不存在,但这并不意味着竞争在其他的执行路径下也不存在。

为了进一步验证，回到第一步来寻找其他符合条件的测试用例，并重复以上过程。如果所有的测试例中都不能交换 Task1 和 Task2 的执行顺序，那么就推断这很可能是一个误报。我们无法确定性地判断一个数据竞争是否存在，因为数据竞争可能在一些我们没有考虑到的执行路径上发生。

2)如果 statement1 是在后台线程中被执行，statement2 在事件的回调函数中被执行(或者相反)，则为 statement1 插入线程同步原语，同时监控消息队列中 statement2 对应的消息。在重放过程中，控制后台线程执行和消息分发的顺序。当然，后台线程有可能是在回调函数中被创建的，如此我们的方法将会失败。

3)如果 statement1 和 statement2 都是在后台线程中被调用，则只需在被测应用中为这两个语句插入同步原语，再在重放过程中调换两个线程的执行顺序，就可以重现这个并发错误。

5 关键技术与实现

5.1 状态转化图的构建

状态转化图的构建过程由运行于 PC 上的自动探测器和运行于 Android 设备上的测试 App 配合完成。在状态转化图的构建阶段，自动探测器通过 Adb 以广播的形式向 Android 设备发送测试命令，测试 App 收到命令后，通过测试框架驱动被测程序执行相应的动作。等待一段时间(默认为 5s)，待应用界面稳定后，获取被测应用当前的状态信息，并通过 socket 将其发送到 PC 端。自动探测器通过状态比较来判断执行的动作前后应用的状态是否发生了改变。如果有状态发生改变，并且当前状态是之前未出现过的，则在状态转化图上添加代表新状态的状态节点和连接前一状态与当前状态的边，边对应之前执行的事件序列。自动探测基于深度优先策略，依次遍历各个状态下的 UI 组件子集 u，即绑定了用户自定义事件处理函数的 UI 组件集，依次触发相应的事件(如某个 Button 实现了 onClickListener 回调，就发送一个 click 该组件的命令)。在当前的实现中，仅根据组件的类型为其预定了一种事件，在后续的工作中将会实现为每个组件预定多种事件，以及实现为特定的组件(根据组件 Id)绑定不同的事件。如果事件引起了状态改变，就进入下一个状态继续处理之前未处理的 UI 组件，自动探测的过程可以用算法 1 所示的伪代码来表示。

算法 1 构建状态转化图算法

```
Intput:Instrumented app
Output:Partial State Transition Graph PSTG: Gs=(Vs,Es)
1. procedure BuildGraph(){
2.        initialTest() //start test app and instrumented app
3.        curState←getCurrentStateInfo () //get initial state infos
4.     guiElementSet←getGUIElements(curState)
5.     if guiElementSet is empty then
6.        goback() // switch to last Activity
7.     else
8.        for each guiElement in guiElementSet do
9.           guiElementSet. remove(guiElement)
10.          lastAction←fireBoundEvent(guiElement)
11.          ExtendGraph(Gs)
12.       end for
13.    end if
       }
14. procedure ExtendGraph(Gs){
15.    curState←getCurrentStateInfo()
16.    if curState is not equal to lastState then
17.       if curState is a new state then
18.           add curState to Vs
19.       end if
20.           newEdge←(lastState,curState,preAction)
21.           add newEdge to Es
22.           lastState←curState
23.           preAction. clear()
24.    else
25.       preAction. add(lastAction)
26.    end if
       }
```

状态比较是构建状态转化图的关键，若两个状态的 UI 组件子集 u 中对应的组件 id 和类型均相同，则判定两个状态相同。这种比较方法开销小，且不会影响自动探测。在测试 App 中，通过 Robotium[14] 框架提供 getViews API 来获取状态的 UI 组件集，接着遍历该集合，通过反射方法得到实现了用户自定义事件处理函数的 UI 组件子集 u。通过检查组件相应的事件回调接口成员变量是否为空，来判定 UI 组件是否实现了特定的事件回调函数。在自动探测的过程中，不断更新从初始状态到达各个状态的路径，并将其保存在各个状态的状态信息中，如果当前状态的路径包含的事件序列的长度大于前一状态的路径长度与触发状态转变的事件序列长度之和，则更新当前状态的路径。

5.2 测试用例的生成与执行

该步骤需要从构建好的状态转化图中寻找一条符合条件的路径来生成测试用例。这条路径需从初始状态开始，能覆盖触发竞争语句对应的事件所在的两条边 $E1$ 和 $E2$。符合条件的路径可能存在多条，要从中找到较短的一条，以使生成的测试用例尽可能短。考虑先找到一条从初始状态到 $E1$ 的初状态的较短路径 $s1$，再找到一条从 $E1$ 的末状态到 $E2$ 的初状态的较短路径 $s2$，则 $s1+E1+s2+E2$ 即为满足条件的路径。由于在构建状态转化图的过程中已记录了从初始状态到各节点的较短路径，因此 $s1$ 可以从状态信息中得到。求路径 $s2$，是一个求有向图两个节点间的最短路径问题，我们采用 Dijkstra 算法来求解。以 $E1$ 的末状态作为起点，$E2$ 的初状态作为目标节点，每条边对应的事件序列包含事件的个数作为边的权重。路径确定后，将边对应的事件序列解析成测试用例并执行(目前方法将事件序列转换成 Android Espresso 测试框架表达的测试用例)。

图 4 所示为第 3 节给出的数据竞争示例所对应的状态转化图(由于空间的限制，未显示出状态信息以及边对应事件序列)。图中 A 为初始状态，加粗和虚线边分别对应 $E1$ 和 $E2$。

因此,先从 $E1$ 的初状态 C 的前置路径中获得 $s1=\langle e4\rangle$;接着计算 $s2$,该示例中 $E1$ 的末状态和 $E2$ 的初状态是同一个状态 L,因此 $s2=\langle\rangle$。最终生成的路径 $s=\langle e4,e11,e16\rangle$。将 s 路径对应的事件序列解析成测试脚本,生成了如图 5 所示的测试用例。

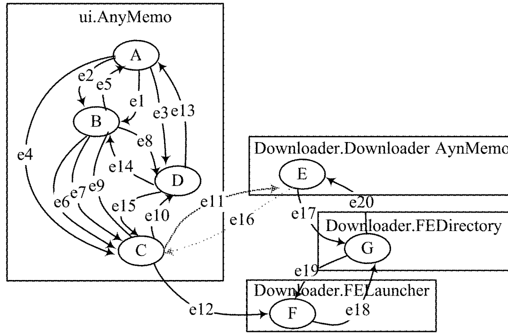

图 4　局部状态转化图

```
onView(withText("下载")).perform(click());
onView(withText("http://anymemo.org(默认)")).perform(click());
Espresso.pressBack();
```

图 5　生成的测试脚本

在该测试用例中,首先跳转到 Downloader AnyMemo Activity,立即启动后台下载线程,等下载完成后执行 goBack 操作,将会使得 onDestroy 回调函数被执行。

5.3　测试用例的重放

在重放阶段,为了控制事件分发的顺序,对 Android Espresso[9] 测试框架进行了修改。在测试执行时,Espresso 通过反射机制获得主线程的事件队列,然后直接分发事件给对应的处理对象,这样事件就不再由测试框架的 Looper 循环来处理。通过这种方式,可以延迟某个特定事件的分发,在另一个事件被执行之后再分发该事件。为了控制线程的执行顺序,使用同步原语来调度线程之间的执行顺序(包括 UI 线程和后台线程)。

5.4　数据竞争的确认

在重放过程中,如果测试用例执行失败或者应用发生执行异常(如 app crash),则可以确认报告的数据竞争为并发错误,但是有些数据竞争可能仅改变应用的语义(如在测试用例生成阶段和重放阶段观察到不一致的执行结果),并不会导致显式的执行异常。为了确认这类并发错误,方法在测试用例生成阶段和重放阶段收集每个 UI 事件(用户动作)执行后当前活动中的各种 UI 控件信息,并比较控件所显示的内容是否相同。如果检测结果不一致,则确认该数据竞争会导致并发错误。

为了保证检测结果的准确性,方法在执行测试用例前对应用进行代码插桩,通过拦截网络调用接口和传感器调用接口,以在测试用例生成阶段捕获并缓存网络消息和各种传感器数据,并在重放阶段直接返回缓存的网络消息和传感器数据,以隔离外界环境对执行结果的影响。

6　实验及结果分析

为了验证方法的有效性,选择了 8 个开源 Android 应用作为对象进行实验。使用静态数据竞争检测工具 DeVa[1] 和动态数据竞争检测工具 EventRacer for Android[2] 检测应用中潜在的数据竞争。针对实验对象,分别从静态/动态检测结果中随机选择 10 个或 5 个检测结果作为待验证的竞争。具体的实验设置和实验结果如表 1 所列,其中 FP 表示被推断为误报的竞争个数,TP 表示被验证为真实竞争的个数。最后通过人工分析应用源码来确定各个竞争是否是误报,并将其作为判断方法有效性的依据,表中最后两列数据对应人工检测的结果。从表 1 的实验结果可以看到,动态检测结果中误报相对较低,静态检测结果中误报比例较高,在 40 个由静态检测方法报告的潜在竞争中,仅有 9 个是真实的竞争(TP),31 个为假阳性(FP),而在 25 个由动态检测方法报告的潜在竞争中,有 16 个被确认为真实的(TP),9 个为假阳性(FP)。文中所提方法的结果与人工确认的结果基本一致。通过对实验数据的分析发现,静态检测结果的误报主要来自同一个 Android 组件(Activity,Service)的两个生命周期事件以及一个 Activity 的生命周期事件和一个用户事件两种情况。例如,共享变量在 onCreate 函数中被初始化,在 onDestroy 函数中被置空,交换两个函数的执行顺序就会出现空指针异常,然而 Activity 的生命周期事件有确定的执行顺序,onCreate 函数总是先于 onDestroy 执行,无法交换它们的执行顺序。在后面的工作中,可以尝试先通过对 Android 组件的生命周期进行建模来排除静态检测结果中的误报。文中方法的结果和人工检查的结果基本一致,这说明该方法是有效的,能帮助开发者确认并进一步修复竞争。

表 1　实验结果

Type	App	#potential data races	FP	TP	FP (manual)	TP (manual)
static	ConnectBot	10	7	2	7	3
	Music		8	2	8	2
	MyTracks		7	3	7	3
	MyExpense		9	1	9	1
dynamic	ConnectBot	5	2	3	2	3
	AnyMemo		2	3	2	3
	FBReaderJ		2	3	2	3
	Mileage		1	4	1	4
	Sanity		2	3	2	3

文中方法虽然能有效验证竞争,但也存在一定的局限性。由于一些竞争仅在非常特殊的情况下才会出现,我们无法生成相应的测试用例,因此不能验证相应的竞争。例如,对于 connectBot 应用,静态检测方法报告的一个数据竞争为:(bound,〈org.connectbot.ConsoleActivity,onServiceDisconnected〉,〈org.connectbot.ConsoleActivity,onResume〉),由于 onServiceDisconnected 仅在 Service 异常退出时才会执行,在测试用例生成阶段,文中方法无法生成相应的测试用例。对于这部分无法生成测试用例的数据竞争,后续拟采用众包测试的方法收集包含潜在数据竞争的应用执行,将其转换成测试用例后再使用文中提出的方法进行确认。

7　相关工作

目前已有许多针对移动应用的软件测试和数据竞争检测

的方法和工具,下面讨论一些最近的相关工作。

7.1 数据竞争检测与测试

已有的竞争检测方法大多针对多线程程序,直接应用到事件驱动的程序中效果将不太理想。最近,研究人员开始关注事件驱动应用(Web 应用、移动应用)中的竞争检测技术。Web 应用是运行于浏览器上的单线程程序,也是一种典型的事件驱动的程序。WebRacer[10] 和 EventRacer[11] 两个最近的研究工作表明,即使运行在单个线程中,数据竞争也可能发生;同时他们通过推导 Web 应用中的发生序关系,发现很多流行网站也存在数据竞争。

针对移动应用也有一些相关工作。CAFA[8] 通过推导事件序列的因果关系来检测一类由数据竞争引起的 use-after-free 错误。DroidRacer[7] 基于动态程序分析,通过构建 Android 应用的形式化语义模型来检测竞争。EventRacer for Android[2] 是针对 Android 应用的可扩展的竞争检测技术,实现了快速的算法来构建和查询发生序图,并通过特定的领域知识减少误报。DEvA[1] 是一种通过静态程序分析自动检测事件驱动的系统中事件异常的工具,它能处理隐式语义的调用、多语义的接口和隐式的并发。Asynchronizer[12] 是一种用来帮助开发者将长时间运行的任务转变为异步任务(AsyncTask)的重构工具,它能检测重构之后 AsyncTask 中的数据竞争。一些针对并发程序的随机测试技术很难被直接应用到 Android 应用中,因为测试用例中通常包含很多种可能的线程执行顺序,而且大部分不会产生数据竞争。如果盲目地尝试这些执行顺序,将会花费大量的测试资源。文中方法主要受到 RACEFUZZER[13] 的启发,它针对多线程应用,借助从现存的动态分析技术中获得的潜在数据竞争信息,通过控制线程的执行顺序,能以较大的概率重现竞争。

7.2 状态空间模型的构建

系统的探测应用的状态空间是各种应用分析与测试任务的基础,已有许多研究关注于如何系统地探测应用程序的状态空间。A^3E[6] 能系统地探测运行于真实设备上的 Android 应用且不需要应用的源码,它提出了两种互补的探测策略,能分别实现系统/快速地遍历 Android 应用的 Activity。Swifthand[5] 使用机器学习的方法在应用执行的过程中动态构建状态空间模型,基于学习到的模型来生成用户输入,用户输入又导致新的状态出现,通过状态比较和状态合并不断地更新状态模型。应用状态空间模型也可以通过静态的方法构建。比如,Yang 等在文献[3]中使用针对回调方法的上下文敏感的静态分析技术来构建 Android 应用的 GUI 模型图;文献[4]构建了应用的静态窗口转化图,它是一个能表示可能的 GUI 窗口序列及其关联事件和回调函数的完整模型。

结束语 文中介绍了一种通过自动生成测试用例来验证 Android 应用中的数据竞争的方法。该方法首先构建应用的状态空间模型,并基于该模型生成包含潜在数据竞争的测试用例;然后在重放测试的过程中,通过交换事件分发和线程执行的顺序来重现数据竞争引起的并发错误。在目前的基础实验中,该方法能准确地指出竞争检测工具的检测结果中真实的竞争和误报,并能通过重现竞争来确定真实竞争的影响。总体来说,前期的实验结果较好,表明该方法是可行的,能帮助开发人员自动验证并发错误。未来我们将进一步完善测试脚本生成工具,并进行更多的实验来评估该方法的有效性。

参 考 文 献

[1] SAFI G,SHAHBAZIAN A,HALFOND W G J,et al. Detecting event anomalies in event-based systems[C]∥Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM,2015:25-37.

[2] BIELIK P,RAYCHEV V,VECHEV M. Scalable race detection for android applications[J]. ACM SIGPLAN Notices,2015,50(10):332-348.

[3] YANG S,YAN D,WU H,et al. Static control-flow analysis of user-driven callbacks in Android applications[C]∥Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE,2015:89-99.

[4] YANG S,ZHANG H,WU H,et al. Static Window Transition Graphs for Android (T)[C]∥2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE,2015:658-668.

[5] CHOI W,NECULA G,SEN K. Guided gui testing of android apps with minimal restart and approximate learning[J]. ACM SIGPLAN Notices,2013,48(10):623-640.

[6] AZIM T,NEAMTIU I. Targeted and depth-first exploration for systematic testing of android apps[J]. ACM SIGPLAN Notices,2013,48(10):641-660.

[7] MAIYA P,KANADE A,MAJUMDAR R. Race detection for Android applications[C]∥ACM SIGPLAN Notices. 2014:316-325.

[8] HSIAO C H,YU J,NARAYANASAMY S,et al. Race detection for event-driven mobile applications[J]. ACM SIGPLAN Notices,2014,49(6):326-336.

[9] Espresso [OL]. https://google. github. io/android-testing-support-library/docs/espresso.

[10] PETROV B,VECHEV M,SRIDHARAN M,et al. Race detection for web applications[J]. ACM SIGPLAN Notices,2012,47(6):251-262.

[11] RAYCHEV V,VECHEV M,SRIDHARAN M. Effective race detection for event-driven programs[J]. ACM SIGPLAN Notices,2013,48(10):151-166.

[12] LIN Y,RADOI C,DIG D. Retrofitting concurrency for android applications through refactoring[C]∥Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM,2014:341-352.

[13] SEN K. Race directed random testing of concurrent programs [J]. ACM SIGPLAN Notices,2008,43(6):11-21.

[14] Robtotium[OL]. Github. https://github. com/RobotiumTech/robotium.