ANALYZING TRACE SEQUENCES FOR ANDROID CRASHES

Authors

Abhyuday Pandey   Ayush Kumar   Umang Malik
Supervisor - Prof Subhajit Roy

Abstract

Android has now overtaken Windows to become the world's most popular operating system, according to data from Statcounter (2017). Since, Android is still a very young idea, many of its aspects have not yet received wide attention. Trace obtained from Android crashes is also one of them. Generally, these crash reports are taken over by the Application development team and worked upon. We conduct a brief study on these traces and their behaviour.

Generating Crashes

The Monkey is a command-line tool that you can run on any emulator instance or on a device. It sends a pseudo-random stream of user events into the system, which acts as a stress test on the application software you are developing. [1]

Below is a snippet of monkey output :-

```
:Sending Touch (ACTION_DOWN): 0:(757.0,805.0)
:Sending Touch (ACTION_UP): 0:(716.1527,789.4009)
:Sending Touch (ACTION_DOWN): 0:(334.0,1585.0)
:Sending Touch (ACTION_UP): 0:(391.0497,1505.206)
:Sending Touch (ACTION_DOWN): 0:(948.0,584.0)
:Sending Touch (ACTION_UP): 0:(950.26117,581.4244)
:Sending Trackball (ACTION_MOVE): 0:(-4.0,3.0)
:Sending Touch (ACTION_DOWN): 0:(737.0,1277.0)
:Sending Touch (ACTION_UP): 0:(776.3995,1379.9172)
```

We limited the study to touch actions generated by Monkey to perform our investigations.

Replaying Crashes

Replay of crashes is a bit difficult task for Android as the results are also affected by flaky tests [2] and *stress* [3]. Also, the simulation of sequence generated by monkey is not replayable without taking some assumptions. We used *MonkeyRunner* [4] to replay our traces.
The MonkeyRunner tool provides an API for writing programs that control an Android device or emulator from outside of Android code.
The assumptions taken were :

- If the distance between an ACTION_DOWN and ACTION_UP command is more than 10 units we classify it as a “drag event”.
- If the distance between an ACTION_DOWN and ACTION_UP command is less than 10 units we classify it as a “touch event”.

This assumption is kept in mind considering the dimensions of stylus/fingers which are used to perform the actions.
We used monkeyrunner now to replay the crashes based on our formatting (source - format_actions.py and trace_runner.py).

Tracking state of program

We simulate the entire trace as a graph *G(V,E)* where *V* is the state of the program and *E* denote the transition from one state of program to another. So, we captured the status of program after each action using *uiautomator.*
The uiautomatorviewer tool provides a convenient GUI to scan and analyze the UI components currently displayed on an Android device. You can use this tool to inspect the layout hierarchy and view the properties of UI components that are visible in the foreground of the device. (source - getView.py and trace_runner.py).

Flaky Tests

A **flaky test** is one that sometimes passes and sometimes fails. Most **flaky tests** are **flaky** because of how the **test** was written, and not due to an actual bug. [5]

Monkey provides an option to set a delay between the events. These can be done using the ` --throttle <milliseconds>` argument.

We summarize our observations in the following table :

Throttle Values	Observation
Very low throttle value (<100 ms)	Results in the crash almost instantly, especially on Emulator/slow devices due to the high frequency of events. Can be used for stress testing.
Throttle values in the range 100-500 ms	Result in *flaky tests*. When the same event sequence is replayed on the device, the device may or may not crash. It is not recognized if the crash was caused due to app or the high frequency of events.

Throttle values above 500ms	Works fine for all devices/emulators and also crashes generated are due to bugs in the app only. These event sequence can be replayed with stable outcome.

For the purpose of our project, we decided to use a threshold of 500ms. This tackles the problem of flaky tests to some extent.

Study on reduction of trace sequences

This is an aspect which attracts industry relevance. The crash sequences in the crash reports should be as compact as possible for efficient transfer and handling of them. The two approaches we studied thoroughly are :

- Using Delta Debugging [6]
    - The paper proposes SimplyDroid, with three hierarchical delta debugging algorithms, namely Hierarchical Delta Debugging (HDD), Balanced Hierarchical Delta Debugging (BHDD) and Local Hierarchical Delta Debugging (LHDD).
    - Balanced HDD is proposes a method deal with real-world scenarios to balance the heights of the tree.
    - Local HDD algorithm adopts the heuristic to preserve the transition from last node of one level to first node of another level.
    - The states are tracked using both the event and the Activity ID (to represent the GUI state of the device).
- Using Bayesian methods [7]
    - The algorithm is based upon Zeller’s delta debugging algorithm reformed to tackle non-determinism in the application behaviour.
    - Remove “subtraces” and maintain a notional probability based upon Bayes Theorem. That is selection of next sub trace will be according to the results when different chunks of these subtraces were removed.
    - Since, the previous point is highly parallelizable, it is worth capitalising.
    - Transform the problem into a Markov Decision Problem to handle non-determinism in the program.

Future Work

An obvious work package will be to inherit the ideas of previous sections and test them on droix-bench, 24 most commonly used apps. The only bottleneck is getting crash sequences. Since, our method relies on generating crash using a pseudo random sequence it becomes virtually impossible to find a crash with this approach when throttle value is as high as 500 ms. An approach that finds a crash avoiding flaky tests in limited time will definitely get a breakthrough in this domain.

REFERENCES

1. https://developer.android.com/studio/test/monkey
2. https://developer.android.com/reference/android/test/FlakyTest
3. https://medium.com/default-to-open/stress-testing-android-apps-601311ebf590
4. https://developer.android.com/studio/test/monkeyrunner
5. https://openedx.atlassian.net/wiki/spaces/TE/pages/161427235/Flaky+Test+Process
6. *SimplyDroid: Efficient event sequence simplification for android application by Bo Jiang, Yuxuan Wu, Teng Li and W. K. Chan at ASE 2017.*
7. *Minimizing GUI Event Traces by Lazaro Clapp, Osbert Bastani, Saswat Anand and Alex Aiken at FSE 2016.*