Problem

Mobile applications ("apps") built for smartphones and tablets become increasingly popular and indispensable in our daily life. However, the apps today are complex, insecure, and often plagued with errors [1]. My primary research goal is creating effective tools to help developers detect and fix software bugs, thus improving the reliability and security of mobile apps.

Detecting and reproducing software bugs is difficult in modern computer system as the hardware today is equipped with multi-core processor and the software is designed multi-threaded to fully utilize the hardware. The concurrency bugs also known as "heisenbugs", which surface occasionally but may result in disastrous errors, are notoriously hard to be captured and debugged [2]. To further complicate the problem, mobile platform is designed in an event-driven manner with high-bandwidth streams of input (e.g., touchs-screen, GPS, camera, network) and asynchronous events between different threads. The non-deterministic nature of event makes bug detection even harder.

Research Overview

My work, to date, has been focused on applying dynamic and static program analysis techniques to find and reproduce concurrent software bugs. Both are crucial: dynamic analysis records the concrete execution trace of the tested app and infer potential software errors by analyzing the trace; static analysis, on the other hand, does not require running the apps, and conservatively predicate the places where bugs are likely to manifest.

Record and Replay. Record-and-replay is a kind of dynamic analysis that proves to be one of the most effective dynamic analysis technique against concurrency bugs. By recording sufficient critical non-deterministic information during program execution, a record-and-replay tool would faithfully replay the execution with equivalent schedule as recorded and help developer to debug the errors if they occur. Prior approaches, targeting traditional desktop and server programs, have focused on replaying low-level machine instructions or system call, which incur very high runtime overhead and not suitable for mobile devices. To keep overhead low, prior record-and-replay approaches for mobile apps only capture GUI input [3] which hurts accuracy as they cannot replay input from the network or sensors that are used frequently by popular apps.

To address this challenge, I designed and implemented a tool called VALERA (VersAtile yet Lightweight rEcord and Replay for Android) [4], a novel *sensor- and event-stream driven* approach to record-and-replay that focuses on sensors and event streams, rather than system calls or the instruction stream, and achieves high accuracy with minimal overhead. VALERA works for Android, the dominant mobile platform. The key insight of VALERA's design is to hit a "sweet spot" on the accuracy vs. overhead curve: replay sensor inputs and events with precise timing allows VALERA to be lightweight yet achieve high accuracy. Experiments of 50 widely-popular Android apps(most of these apps have in excess of 10 million installs) that use a variety of sensors show that VALERA can successfully replay them in an efficient manner: it imposes just 1.01% time overhead for record, 1.02% time overhead for replay, 208KB/s space overhead on average, and can sustain event rates exceeding 1,000 events/second. Besides, VALERA's support for deterministic replay of asynchronous events allow developers to reproduce event-driven races in several apps, that would be very difficult to reproduce manually.

Static Race Detection. Ideally, we want to find and report all the harmful event-driven races that may crash the system or show incorrect results. Although dynamic analysis is useful to handle event-driven races, it is not sound in nature: dynamic analysis can only reproduce races in the paricular executed schedule but cannot detect bugs under specific circumstances where the executions do not reach. Alternatively, static analysis is not limited to the specific schedule because it conservatively reasons about every possible path that can be invoked at runtime.

Based on this idea, I design and implement a static event-driven race detector. First, to understand all the possible user interactive behavior of the app, the race detector extract all the available event handlers including UI handlers and system event handlers. Next, with the extracted event handlers as program entry points, the static analysis algorithm builds a happens-before [5] graph (HBG) and find all the data access operations. The HBG describes the strict execution order of the program. For example, in a thread creation case, the parent thread's fork operation always happens before child thread's start operation. By inferring the HBG, the race detector would report a race if there are two events, which do not have happens-before relation, access the same memory and at least one of those accesses is write. Finally, the static race detector filters out those false positives that could not occurs in real execution due to some constraints and prioritize the results to make those harmful races higher priority.

Applications

My work brings advanced programming languages techniques to mobile platforms. Based on this foundational step, my work can be applied into other related fields.

Software Testing. Leveraging VALERA's powerful ability to record and replay sensor inputs in Android apps, we could construct new approaches that help automate smartphone app testing. For example, we could alter the logged data in a semantically-meaningful way: by applying principled transformations (e.g., changing GPS coordinates or navigation speed), a new input data is constructed, which represents a new test case. This is a kind of test amplification. The basic idea of VALERA could also be applied to other platforms such as desktop applications, Apple's iOS app and Microsoft's windows mobile phone. Since all of these platforms are event and sensor driven, the porting work of VALERA is easy and straightforward.

Distributed System. Data race is a insidious programming error that manifests not only in mobile apps but also in distributed systems. Races are hard to detect and fix as they usually occur non-deterministically, under specific schedules. My work on static race detector could also be applied into detecting race in distributed system such as MapReduce program which is popular in cloud computing. The basic idea of using happens-before (HB) relation to infer program execution order could be shared among distributed program as the HB relation describes the system dependencies of various functions or modules. The race detection algorithm could be complemented by adding specific models of the distributed applications.

References

[1] Bo Zhou, Iulian Neamtiu, and Rajiv Gupta. A cross-platform analysis of bugs and bug-fixing in open source projects: Desktop vs. android vs. ios. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE '15, pages 7:1–7:10, New York, NY, USA, 2015. ACM.

[2] Madanlal Musuvathi, Shaz Qadeer, Thomas Ball, Gerard Basler, Piramanayagam Arumuga Nainar, and Iulian Neamtiu. Finding and reproducing heisenbugs in concurrent programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 267–280, Berkeley, CA, USA, 2008. USENIX Association.

[3] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. Reran: Timing- and touch-sensitive record and replay for android. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 72–81, Piscataway, NJ, USA, 2013. IEEE Press.

[4] Yongjian Hu, Tanzirul Azim, and Iulian Neamtiu. Versatile yet lightweight record-and-replay for android. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2015, pages 349–366, New York, NY, USA, 2015. ACM.

[5] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.