Flaky Tests at Google

Samuel Myles
Zifan Wang
Runnan Zhou
Boyan Tian

Deterministic Unit Testing

- Tests for correct implementation in different parts of codebase.
- The same input and code should produce the same test result every time.

Nondeterministic Unit Testing (Flaky Tests)

What

- A test that can pass or fail given the same portion of code to test
- A failed test of this nature doesn't necessarily indicate an issue with new code
- Recreating failures can be a cumbersome process

```
1 @Test
2 public void testRsReportsWrongServerName() throws Exception {
3   MiniHBaseCluster cluster = TEST_UTIL.getHBaseCluster();
4   MiniHBaseClusterRegionServer firstServer =
5     (MiniHBaseClusterRegionServer)cluster.getRegionServer(0);
6   HServerInfo hsi = firstServer.getServerInfo();
7   firstServer.setHServerInfo(...);
8
9   // Sleep while the region server pings back
10  Thread.sleep(2000);
11  assertTrue(firstServer.isOnline());
12  assertEquals(2,cluster.getLiveRegionServerThreads().size());
13  ... // similarly for secondServer
14 }
```

Server responded on time ---> test pass

Server responded out of time ---> test fail

Reason for Flaky tests

- Async wait
- Concurrency
- Test Order dependency
- Network
- I/O

Advantages and Disadvantages of Flaky Tests

Advantages:

- Help to find some hidden bugs

Disadvantages:

- Can not achieve the test goal
- people gradually distrust test automation
- Reduce the efficiency of the R&D team

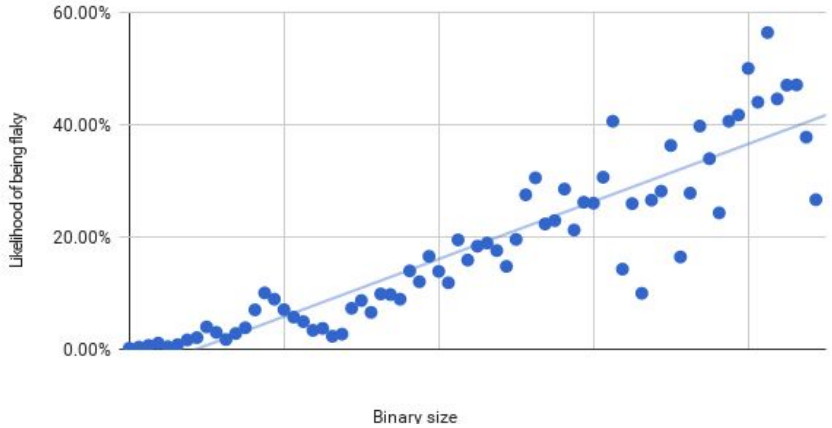
Google's Dilemma

- Around 2017 Google had around 4.2 million tests that ran on their continuous integration system
  - Around 62,000 tests (~1.5% of their total test bed) had flaky behavior a week
  - 84% of passing tests that transitioned to failing involved a flaky test
  - Spent between 2–16% of their compute resources re-running flaky tests
- Consequences:
  - Extra resources used to investigate whether test was flaky or legitimate
  - Some developers dismissed legitimate failures as being flaky
  - False positives kept needed code changes from immediately reaching deployment

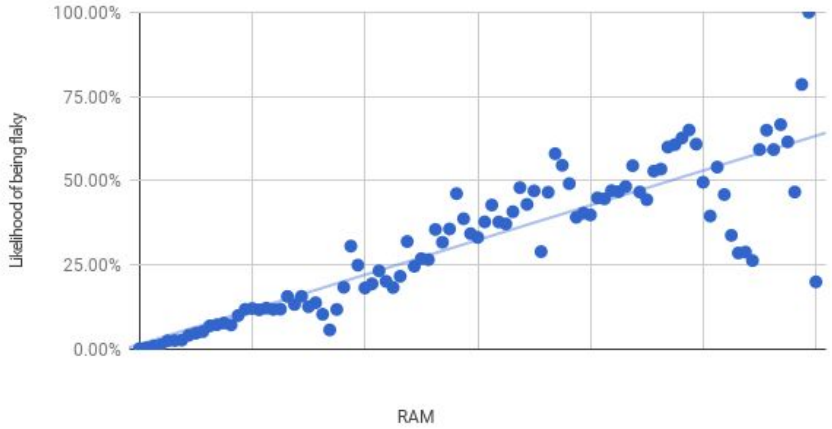
Trends in Flakiness

Flakiness of tests using some of our common testing tools		
Category	% of tests that are flaky	% of all flaky tests
All tests	1.65%	100%
Java WebDriver	10.45%	20.3%
Python WebDriver	18.72%	4.0%
An internal integration tool	14.94%	10.6%
Android emulator	25.46%	11.9%

Binary size vs. Flaky likelihood



Memory (RAM) vs. Flaky likelihood



Best(ish) Practices

- Quickly identify flaky tests
  - Logging of test conditions
    - Execution times, test types, run flags, consumed resources, etc.
  - Make use of tools to identify flakiness within tests
  - Rerun tests with a clean system state to check if a constant result occurs
- Ensure flaky tests are reported and fixed fast
  - Reported tests not yet fixed saves time for rest of development team
  - Quickly fixing flaky tests prevents dependent tests from failing

Discussion Questions

- Do you think smaller companies should invest in flaky test mitigation if they aren't working on large scalable systems?

- Should flaky tests, or at least their existence, be a topic covered when learning software development?

References

- https://hackernoon.com/flaky-tests-a-war-that-never-ends-9aa32fdef359
- https://testing.googleblog.com/2017/04/where-do-our-flaky-tests-come-from.html
- https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html
- https://engineering.salesforce.com/flaky-tests-and-how-to-avoid-them-25b84b756f60#:~:text=Resource%20leak%3A%20Tests%20can%20be,they%20are%20no%20longer%20needed.
- http://mir.cs.illinois.edu/~qluo2/fse14LuoHEM.pdf