An Empirical Analysis of Flaky Tests

Presented By
Mostafa Mohammed
As part of the Class CS6704 @ VT

Contents

- Background
- Problem statement: What is the problem?
- Approach: What technique is proposed?
- Results
- Related work: How is the research different from prior work?
- Conclusion: What is learnt from the paper?
- Discussion questions

Background

- Regression Test: Test used to check if the applied modifications to software does not affect its functionality.
- Unmodified test is expected to either always fails or always pass.
- Flaky Tests: tests that have non-deterministic outcome.
  - Some times the test fails and sometimes it passes for the same code version.

Background

- Flaky tests are common on large systems
- Flaky tests are troublesome to developers:
  - Due to their non-determinism, Flaky tests are hare to reproduce.
  - The apparel of flaky tests in the regression test may indicate to developers that there are some errors caused by the recent changes. (Time consuming)
  - In some cases, flaky tests are due to some real bugs, so it will be risky to ignore them.

Background

- The common approaches to deal with flaky tests are:
  - Ignore the test if it passes only one time (declare it passing).
  - Remove the test altogether.
  - Ignore the results of this test.

Problem statement

- The authors of this paper claim that researchers did not focus on flaky tests.
- The main goal of this paper is to introduce an extensive study of flaky tests to address the following:
  - What are the common causes of Flaky tests?
  - How to manifest the flaky test failure?
  - What are the common fixing strategies for flaky tests?
- This paper provides actionable information about avoiding, detecting, and fixing flaky tests.
- The results of this paper will help researchers and developers in developing automated tools to help in fixing flaky tests.

Approach

- Labeling Phase:
  - First, using the commits found in Apache Software Foundation to build a large dataset of commits that are likely about fixing flaky tests. The result is 1129 commits.
    - This is done by looking for two keywords “flaky” or “intermit” only.
- Filtering Phase:
  - Manually inspect each commit to determine the commits that are related to **fixed** flaky tests.
  - Two authors separately inspect each commit then merge their results to
    - increase confidence
    - The data set reduced to 855 commits.
    - other 274 commits are either about the CUT not about the test code or just incidental matches
  - determine the commits that are Likely Distinct Fixed Flaky Tests. (LDFFT in the table)

Approach

Table 2: Summary of commit info and flaky test categories

	"intermit"	"flak"	Total	Total w. Bug Reports	HBase	ActiveMQ	Hadoop	Derby	Other Projects
All commits	859	270	1,129	615	134	86	90	118	701
Commits about flaky tests	708	147	855	545	132	83	83	102	455
LDFFT commits	399	87	486	298	72	68	56	49	241
Inspected commits	167	34	201	124	23	20	29	12	117
Async Wait	62	12	74	43	10	11	7	3	42
Concurrency	26	6	32	19	2	3	3	1	23
Test Order Dependency	14	5	19	16	3	0	10	2	4
Resource Leak	9	2	11	8	2	2	0	1	6
Network	10	0	10	6	1	1	2	0	6
Time	5	0	5	2	0	1	1	0	3
IO	4	0	4	3	0	0	1	1	2
Randomness	2	2	4	4	1	0	3	0	0
Floating Point Operations	2	1	3	2	0	0	1	1	1
Unordered Collections	1	0	1	1	0	0	0	0	1
Hard to classify	34	6	40	21	4	2	2	3	29

Approach

- Analysis Phase
  - In depth study to a subset of the LDFFT commits
    - Sort the projects based on number of LDFFT commits
    - Split them into 2 groups
      - Small <6 commits. Why 6!!!
      - Large >=6 commits
    - The subset will be:
      - All the small groups
      - Third of the commits in every project in large group.
    - This subset will not be biased toward projects with large LDFFT commits
    - The subset contains 201 commits.
    - For each of the 201 commits, an author examines it to answer these questions:
      - Is the commit indeed fixing a flaky test?
      - What is the root cause of the flakiness for the test?
      - How can the flakiness be manifested?
      - How is the test fixed?

Results

Findings about flaky test causes	Implications
F.1 The top three categories of flaky tests are ASYNC WAIT, CONCURRENCY, and TEST ORDER DEPENDENCY.	**I.1** Techniques for detecting and fixing flaky tests should focus on these three categories.
F.2 Most flaky tests (78%) are flaky the first time they are written.	**I.2** Techniques that extensively check tests when they are first added can detect most flaky tests.
Findings about flaky test manifestation	**Implications**
F.3 Almost all flaky tests (96%) are independent of the platform (i.e., could fail on different operating systems or hardware) even if they depend on the environment (e.g., the content of the file system).	**I.3** Techniques for manifesting flaky tests can check platform dependence lower in priority than checking environment dependence (e.g. event ordering or time), especially when resources are limited.
F.4 About third of ASYNC WAIT flaky tests (34%) use a simple method call with time delays to enforce orderings.	**I.4** Many ASYNC WAIT flaky tests can be simply manifested by changing time delays of order-enforcing methods.
F.5 Most ASYNC WAIT flaky tests (85%) do not wait for external resources and involve only one ordering.	**I.5** Most ASYNC WAIT flaky tests can be detected by adding one time delay in a certain part of the code without the need of controlling the external environment.
F.6 Almost all CONCURRENCY flaky tests contain only two threads or their failures can be simplified to only two threads, and 97% of their failures are due to concurrent accesses only on memory objects.	**I.6** Existing techniques of increasing context switch probability, such as [10], could in principle manifest most CONCURRENCY flaky tests.
F.7 Many TEST ORDER DEPENDENCY flaky tests (47%) are caused by dependency on external resources.	**I.7** Not all TEST ORDER DEPENDENCY flaky tests can be detected by recording and comparing internal memory object states. Many tests require modeling external environment or explicit reruns with different orders [37].

Results

Findings about flaky test fixes	Implications
F.8 Many ASYNC WAIT flaky tests (54%) are fixed using `waitFor`. which often completely removes the flakiness rather than just reducing its chance.	**I.8 For developers:** Explicitly express the dependencies between chunks of code by inserting `waitFor` to synchronize the code. **For researchers:** Comparing the order of events between correct runs and failing runs, techniques could automatically insert order-enforcing methods such as `waitFor` to fix the code.
F.9 Various CONCURRENCY flaky tests are fixed in different ways: 31% are fixed by adding locks, 25% are fixed by making code deterministic, and 9% are fixed by changing conditions. Our results are consistent with a study on concurrency bugs [24].	**I.9** There is no one common strategy that can be used to fix all CONCURRENCY flaky tests. Developers need to carefully investigate the root causes of flakiness to fix such tests.
F.10 Most TEST ORDER DEPENDENCY flaky tests (74%) are fixed by cleaning the shared state between test runs.	**I.10 For developers:** Identify the shared state and maintain it clean before and after test runs. **For researchers:** Automated techniques can help by recording the program state before the test starts execution and comparing it with the state after the test finishes. Automatically generating code in `setUp/tearDown` methods to restore shared program state, such as static fields [7], could fix many TEST ORDER DEPENDENCY flaky tests.
F.11 Fixing flaky tests in other categories varies from case to case.	**I.11** There is no silver bullet for fixing arbitrary types of flaky tests. The general principle is to carefully use API methods with non-deterministic output or external dependency (e.g., time or network).
F.12 Some fixes to flaky tests (24%) modify the CUT, and most of these cases (94%) fix a bug in the CUT.	**I.12** Flaky tests should not simply be removed or disabled because they can help uncover bugs in the CUT.

Results

Table 4: Flaky test fixes per category

Category	Fix type	Total	Rem-ove	Decr-ease
Async Wait	Add/modify `waitFor`	42	23	19
	Add/modify `sleep`	20	0	20
	Reorder execution	2	0	2
	Other	10	9	1
Concurrency	Lock atomic operation	10	10	0
	Make deterministic	8	8	0
	Change condition	3	3	0
	Change assertion	3	3	0
	Other	8	8	0
Test order dependency	Setup/cleanup state	14	14	0
	Remove dependency	3	3	0
	Merge tests	2	2	0

Evaluation

- The work in this paper is done manually.
- This costs too much time to read, label, and filter the commits.
- There are high possibility of errors in every step.
- To reduce the errors, two authors worked separately on each commit and their results are combined.
- The paper connect its findings with other papers whether they are similar results or contradicting results
  - For example: they found that the main subcategories of Concurrency flaky tests, match the common bugs from concurrent programming (other paper).
  - They also stated that their finding about "the fixes for Concurrency flaky tests completely remove flakiness in the test" contradict with a previous study.

Related Work

This Paper	Related Work
An in depth study to flaky tests and how developers and researchers can manifest them and fix them.	Papers focus on specific category of non-determinism causes and how they can be fixed.
Focuses on characterizing flaky tests across all categories.	Focus on one category only.
Using commit logs to manually detect and classify flaky tests.	Using bug reports to identify flaky tests in a specific category.
Separating Concurrency and Async. Waiting from each other and give them more attention as they are the main cause of many flaky tests	Papers focus mainly on Test Order Dependency as a main cause of flaky tests
Reveal a number of different strategies for fixing flaky tests in different categories. The findings can help in developing automated techniques for fixing bugs.	Proposed automated techniques to fix concurrency bugs.
Study flaky tests that fail non-deterministically	Provided test repair that focuses on broken tests that fail deterministically,

Conclusion

- Flaky tests are common in large projects and some flaky tests appear in the first version of these projects.
- Flaky tests can hide major errors in CUT, so they should not be ignored.
- There is no single solution to solve all flaky tests.
- The paper provides some findings that will help researchers and developers to develop automated tools to manifest, debug and fix flaky tests
- It is so important to provide descriptive commits about failed tests to help researchers and other developers to understand the causes of tests failure and provide automated tools and methods to solve them.

Discussion questions

- How did this study impact the field?
- What questions remain open?
- What experiments are missing?
- How does it really relate the previous research?
- Future directions.
- Could a similar paper be published today?

Thank you,

شكراً

Questions?