iDSpeedy

An Incremental Extension of iDFlakies

Gabriel Darnell
Electrical and Computer Engineering
University of Texas at Austin
Austin, TX, USA
gdarnell@utexas.edu

Abel Philips
Electrical and Computer Engineering
University of Texas at Austin
Austin TX, USA
abel.philips@utexas.edu

ABSTRACT

Flaky tests are an issue in almost all software projects. Many flaky test tools have been developed to identify and sometimes solve these problems. Ideally, these tools would be integrated into an industrial pipeline to identify flaky tests after each commit.

This paper presents iDSpeedy, an extension of a current flaky test tool, iDFlakies. iDSpeedy takes iDFlakies current functionality and adds an incremental runner that allows the iDFlakies framework to work across versions of a project. This approach retains the accuracy of iDFlakies while also decreasing runtime by amortizing the cost of a full run of iDFlakies over multiple commits.

We evaluate the success of our tool by looking at its accuracy compared to iDFlakies, its runtime over versions compared to iDFlakies over one version, and its runtime over versions compared to iDFlakies over multiple versions.

We conducted three separate experiments to evaluate the accuracy and runtime differences between iDSpeedy and iDFlakies. We selected nine projects from the iDFlakies dataset to use in our evaluation [1]. Our evaluation showed that iDSpeedy retained similar accuracy to iDFlakies and a similar runtime to iDFlakies across multiple versions of a project. This shows that we successfully extended iDFlakies to be implemented incrementally.

CCS CONCEPTS

• Flaky Test • Order-dependent tests • Software Testing

KEYWORDS

Flaky Test, CICD Pipeline, Order Dependent Tests

1 INTRODUCTION

As the scale of software projects has become more complex, their test suites have grown increasingly larger. In these large test suites, one can often find tests that non-deterministically pass or fail on the same version of the code. These "flaky tests" are difficult to detect and debug. They are often due to varied causes such as faulty test code. One kind of flaky test is the order-dependent flaky test [1]. Previous work [1], classifies order-dependent flaky tests as tests that pass or fail depending on the order in which they are run.

iDFlakies is a tool designed to detect order-dependent flaky tests in Java projects [1]. It works by running a number of rounds during which it changes the order in which a class' tests are run and attempts a random order each round. The tool determines if a test is order-dependent and outputs its results after the specified number of rounds. However, as the tool must repeatedly run a project's tests in different orders, the time cost of the tool can become lengthy on larger projects. Larger test suites can take a large amount of time to run which can stretch into days or weeks, which significantly hurts developer productivity [2]. In such situations, it can be difficult to use a tool such as iDFlakies to find flaky tests in the project as it can add unwanted time to the development cycle.

We have added additional functionality for iDFlakies which allows for it to amortize the time spent detecting flaky tests by running for less rounds over multiple versions of a project. We named this modified tool iDSpeedy. It adds a new "incremental" configuration to iDFlakies which saves data on test orders between versions of the project.

We found that iDSpeedy was able to find a similar number of flaky tests in all of the projects it was run on in runtimes that were relatively close to that of iDFlakies. In one case, iDSpeedy was able to find 33 more flaky tests in one project than iDFlakies was. Overall, it appears that the tool is able to run as efficiently over multiple versions. The project can be found at github.com/gkd248/iDSpeedy.

2 MOTIVATION

While there are now many tools to identify and solve different types of flakiness in software projects, very few were designed with CICD pipelines in mind. We wanted to extend iDFlakies so it would be more feasible to integrate into an industrial software pipeline. The main factor to address for this integration is the overhead required to run the tool after every commit. While iDFlakies does remember previous orders ran between rounds, it does not remember previous orders ran between executions of the tool. This is a missed opportunity when attempting to integrate the tool into the pipeline since orders could be repeated over multiple executions because of the lack of tracking. This can result in unnecessary overhead since the same test orders could be running every commit. Additionally by splitting up the overall functionality of iDFlakies over multiple commits the same total number of rounds can be run, but at lower overhead per commit.

Our tool, iDSpeedy, tracks the orders run between commits, so that flaky tests can be identified after every commit while not having an overhead as expensive as a full run of iDFlakies. Over multiple commits enough rounds will have been run that would be equivalent to a full iDFlakies execution. Instead of choosing a full run of iDFlakies with every commit, which would have a significantly higher overhead, or running iDFlakies less frequently the developer would have the best of both worlds. They would be getting a flaky report after each commit and not having to deal with significant runtime overhead.

3 PRIOR WORK

Our tool, iDSpeedy, is an extension of iDFlakies, a tool developed by Lam et al. which detects and partially classifies flaky tests [1]. It works by running a number of rounds during which it changes the order in which a class' tests are run and attempts a random order each round. The tool determines if a test is order-dependent and outputs its results after the specified number of rounds. However, as the tool must repeatedly run a project's tests in different orders, the time cost of the tool can become lengthy on larger projects. We use iDFlakies' underlying logic within iDSpeedy, but manipulate how test orders are generated by treating new tests differently than old tests. We also add logic to enable iDFlakies to work over different commits by tracking orders previously used.

4 IMPLEMENTATION

Since our tool is an extension of iDFlakies we started our implementation with a fork of iDFlakies from GitHub. iDFlakies is made up of multiple different testrunners that allow the user to specify how they want their test orders to be generated. We added our own, "incremental," to the options and created our own shuffler called IncrementalShuffler.java. Each time a round is executed a shuffler is called that handles generating the test order that will be run. After the test order is generated it is passed into ExecutingDetector.java which handles identifying the flaky tests based on the order generated. We did not change any logic within ExecutingDetector.java, we only added the functionality to keep track of flaky tests identified between versions.

4.1 Incremental Shuffler

Our incrementalShuffler.java is unique because it handles new tests differently and ensures test orders aren't rerun. We keep track of what tests were included in the previous commit of the project and on the next commit we compare all tests and identify any new tests. We then run the new tests once at the front of all tests within the class and once at the end of all tests since that is a very common location to identify new tests as flaky. Once all new tests have been processed we generate a random order and check that it hasn't been run yet.

4.2 Tracking Data

To extend iDFlakies' functionality to build up incrementally across versions we needed to adjust how certain data is written. We had to adjust the flaky list file, and write to our own files to keep track of test orders run and the current pool of tests for the project.

Normally, the flaky-lists.json file overwrites whatever is in it previously. With the incremental runner we needed to append to this file so the list of flaky tests continues to grow the more iDSpeedy is run. To do this we added logic to append instead of overwrite the file when the runner is incremental.

We also added a folder called incremental to the .dtfixingtools directory to keep two files. We have a file to keep track of test orders run and the current pool of tests. To keep track of test orders we keep track of orders during the run in a data structure and then write to the file once we are done running all rounds. For the pool of tests we utilize a part of code that iDFlakies already has and write the maven provided list of tests to a file to later identify new tests. At the beginning of each run we load in all the tests from the previous run and compare that to the current pool of tests. By doing this we identify what tests are new to the current commit and run them at the beginning and end accordingly.

5 EXPERIMENTAL SETUP

To ensure that we had a collection of useful test suites with order-dependent flaky tests, we decided to utilize the same projects that are mentioned in the original iDFlakies paper [1]. As the paper mentioned that several developers had fixed detected flaky tests, we only ran iDSpeedy on commits that iDFlakies was run on in the original paper and commits that preceded them. We ran all of the projects using JDK 1.8 and iDSpeedy as a Maven plugin.

5.1 RQ 1: Does iDSpeedy retain the same accuracy as iDFlakies?

iDSpeedy's method of creating test orders differs from iDFlakies' "random-class-method" configuration as it puts new tests at the front and end of two subsequent test orders. To ensure that this change did not impact iDSpeedy's accuracy, we wanted to run both tools on the same version of the project and ensure that its ability to find flaky tests was comparable to iDFlakies. iDFlakies' Github repository recommends running the tool on the "random-class-method" configuration 20 times as the tool's default settings. We ran both tools for 20 rounds on the version of the project used in the original paper. We then compared the number of flaky tests found by iDFlakies and iDSpeedy.

5.2 RQ 2: How does the accuracy and runtime of iDSpeedy and iDFlakies compare when run the same number of total rounds?

To judge whether iDSpeedy was able to efficiently find flaky tests over multiple versions compared to iDFlakies, we ran iDSpeedy 5 rounds on five sequential versions. We used the git version in the iDFlakies paper and its four preceding commits. In this way, iDSpeedy also ran 25 times overall on the project. We then ran iDFlakies 25 times overall on the latest version of the five. We recorded the amount of time spent by the tool and the number of flaky tests it found. We also recorded the amount of unique flaky tests each tool identified over the 25 rounds.

5.3 RQ 3: How does the accuracy and runtime of iDSpeedy and iDFlakies compare when run the same number of times on each version?

To see how efficient iDSpeedy was on each version of the projects, we compared the amount of time that iDFlakies would run on all versions of the project with the same number of rounds. We ran iDFlakies on each of the five versions five times and recorded how much time it took on each version. We then compared it to the time taken by iDSpeedy based on results from the previous step of the experiment. We also recorded the amount of unique flaky tests each tool identified over the 25 rounds.

6 RESULTS

To compile results we cloned each project and modified the pom.xml of all 5 versions of all projects as needed to run both tools on them. We then recorded the number of flaky tests identified and the total runtime of the tool in a spreadsheet which can be found at https://bit.ly/2INcwn4. Our results for each research question are below.

6.1 RQ 1 Results

Research question one focused on checking that we did not manipulate iDFlakies' framework for identifying flaky tests. By running both tools for the same number of rounds on the same version of projects, we expected to have very similar results. Our expectations were correct, and iDSpeedy only missed 4 total flaky tests out of 279. Missing only 1.4% of the flaky tests that iDFlakies identified is not a significant difference and also could be attributed to the randomness of the tools when generating test orders. We believe that this shows iDSpeedy's test order generation is as accurate as iDFlakies.

Project	**iDSpeedy # Flaky Tests**	**iDFlakies # Flaky Tests**
TooTallNate/Java-WebSocket	268	268
kagkarlsson/db-scheduler	4	6
tbsalling/aismessages	2	2
spring-projects/spring-data-envers	0	2
jhipster/jhipster-registry	1	1

Figure 1: **RQ 1 Data Results Excluding Projects with No Flaky Tests**

6.2 RQ 2 Results

Research question 2 focused on if iDSpeedy's runtime and accuracy over multiple versions was comparable to that of iDFlakies. From the collected results, we can see that IDFlakies was able to find as many flaky tests in all of the projects the tools were run on. In most of the projects, both tools had relatively similar runtimes. In six out of the nine projects, iDSpeedy was slower than iDFlakies. However, the difference in time in most of these projects was not very large. It appears that none of the projects, except for Java-WebSocket, had more than a 300 second difference in runtime. Java-WebSocket had the most extreme difference in runtimes, where iDSpeedy took double the time of iDFlakies. However, iDSpeedy was able to find 33 more flaky tests within the project than iDFlakies. iDFlakies uses a verification method to check that tests are flaky which takes additional time to run. Due to the number of additional flaky tests detected, it is possible that the verification time of the new tests can account for the increase in runtime.

We can conclude that iDSpeedy was able to maintain accuracy over the multiple versions it ran on. Despite not being faster on all of the projects, iDSpeedy maintained a close runtime to iDFlakies. It is possible that this runtime difference could be improved if iDSpeedy has been running on several more commits over time.

Project	iDSpeedy Runtime (sec)	# of Flaky Tests	iDFlakies Runtime (sec)	# of Flaky Tests
TooTallNate/Java-WebSocket	12299.9	293	6937.0	260
kagkarlsson/db-scheduler	2045.2	4	1823.1	4
flaxsearch/luwak	529.7	0	802.5	0
jhipster/jhipster-registry	468.2	1	316.6	1
outbrain/aletheia	368.5	0	379.3	0
spring-projects/spring-data-envers	139.2	2	113.0	2
codingchili/excelastic	112.0	0	103.1	0
odrotbohm/sos/00-monolith	97.2	0	97.6	0
tbsalling/aismessages	46.7	2	28.2	2

Figure 2: **RQ 2 Data Results**

6.3 RQ 3 Results

Research question three focused on the comparison of the two tools when both are run five times on five versions of the same project. FIrst, looking at runtime the values are very similar except for db-scheduler where iDSpeedy takes twice as long to execute. For this project it is important to note that iDSpeedy finds four flaky tests while iDFlakies finds zero. Since, verifying a test is flaky does require additional overhead this could explain this drastically longer runtime compared to the other projects. Secondly, looking at the number of flaky tests identified shows that iDSpeedy performed just as well as iDFlakies, and in one case better.

Project	iDSpeedy Runtime (sec)	# of Flaky Tests	iDFlakies Runtime (sec)	# of Flaky Tests
kagkarlsson/db-scheduler	2045.2	4	1407.5	0
jhipster/jhipster-registry	468.2	1	467.7	1
spring-projects/spring-data-envers	139.2	2	121.7	2
odrotbohm/sos/00-monolith	97.2	0	84.1	0
tbsalling/aismessages	46.7	2	48.0	2

Figure 3: **RQ 3 Data Results**

7 FUTURE WORK

The future work for iDSpeedy includes completing a pull request to add our changes to the iDFlakies framework, additional logic for running new tests if we don't have enough rounds to run each at the front and back, and a smarter way to run tests after a new test is identified. We would like to work with the authors of iDFlakies to integrate our work into their framework. Since we worked off the iDFlakies project and added our own classes to work within the iDFlakies framework we should be able to communicate with the authors to complete a pull request.

Within the implementation we currently have two spots that could be improved if we had more time. First, when new tests are run at the front and back of test orders we do not have a way of saving which new tests haven’t been run yet. In the case that there aren’t enough runs to cover each new test at the front and back those new tests will not be run at the front and back. This could be avoided by writing the tests to a file and reading that in the next time iDSpeedy is executed. Secondly, our current implementation erases any test orders that have been run once new tests are identified since the pool of tests is different. It would make sense to use the previous test orders that have been run to decide what orders should be run first with the new tests. This option is complex and could be implemented a few ways. One option could be to run the new tests at the front or back of the previously run orders. Another option would be to run the new tests at the front and back of only the test orders that presented a flaky test to more efficiently attempt to identify new flaky tests. We would also like to expand the number of projects that we have run the tool on and see how it performs on more recent ones.

8 CONCLUSION

Overall we were successful in our attempt to extend iDFlakies to work over versions of a project. iDSpeedy was able to retain a similar accuracy to iDFlakies while also retaining a similar runtime when comparing an equal number of rounds. By writing to the file directory to save pertinent data and then reading from these files every execution we were able to pass data between versions. We also added logic to treat new tests differently so potential new flakiness added in a new commit is identified quicker.

However, there is still room for improvement. Our time frame limited the work we could complete, but we have ideas for the future work. Using past runs to choose what orders new tests be run at the front and back could further increase the efficiency by finding a new source of flakiness in less rounds. By saving previous runs there are a multitude of other possibilities, since the stored data can be used to inform smarter choices of test orders instead of the current way of just randomizing the test order.

ACKNOWLEDGMENTS

Thank you to Professor Shi for helping us throughout the semester. We had quite a few questions about the inner workings of iDFlakies and experimental setup. His answers and suggestions helped shape this paper and possible future work.

REFERENCES

[1] Wing Lam, Reed Oei, August Shi, Darko Marinov and Tao Xie. 2019. iDFlakies: A Framework for Detecting and Partially Classifying Flaky Tests. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST),* Xi'an, China, 312-322. DOI:https://doi.org/10.1109/ICST.2019.00038.

[2] Thomas Bach, Ralf Pannemans, and Sascha Schwedes. 2018. Effects of an Economic Approach for Test Case Selection and Reduction for a Large Industrial Project. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW),* Vasteras, Sweden, 374-379. DOI:https://doi.org/10.1109/ICSTW.2018.00076.