Characterizing Flaky Tests in Node.js Applications

Xiaoning Chang[†‡*], Zheheng Liang[ξ§*], Guoquan Wu[†‡¶], Yu Gao[†‡],
Wei Chen[†‡], Jun Wei[†‡], Zhenyue Long[ξ§], Lei Cui[ξ§], Tao Huang[†‡]
[†]*State Key Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences, Beijing, China*
[‡]*University of Chinese Academy of Sciences, Beijing, China*
[§]*Joint Laboratory on Cyberspace Security, China Southern Power Grid, Guangzhou, China*
[ξ]*Guangdong Power Grid, Guangzhou, China*
[†]{changxiaoning17, gqwu, gaoyu15, wchen, wj, tao}@otcaix.iscas.ac.cn
[ξ]liangzheheng@qq.com, [ξ]cuilei@gdxx.csg.cn, [ξ]zhenyue@undecidable.org

2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE) | 979-8-3503-2996-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASE56229.2023.00025

***Abstract*—Regression testing is an important means of assessing the quality of Node.js applications. However, non-deterministic executions inside Node.js framework could make test cases intermittently pass or fail on the same version of code, which are called flaky tests. Flaky tests can cause unreliable test results, and make developers waste a significant amount of time debugging the bugs that do not belong to the target application.**

In this paper, we conduct an empirical study on 87 flaky tests from 7 popular Node.js applications, and analyze the non-determinism that causes these flaky tests. Through this study, there is a wide range of non-determinism to cause flaky tests, including non-deterministic event triggering order, non-deterministic function calls, non-deterministic process/thread scheduling order, non-deterministic execution of asynchronous tasks and non-deterministic event triggering data. The result reveals that, existing approaches on event race detection are not sufficient for flaky test detection. In future, researchers can design flaky test detection approaches targeted at different categories of non-determinism.

***Index Terms*—flaky tests, non-determinism**

I. Introduction

Regression testing is an important method for assurance the quality of code changes. During regression testing, test case failures typically indicate bugs in the code changes. However, existing researches [1], [2], [3], [4] reveal that test case failures may not necessarily be due to code modification, but rather due to *flaky tests*, which intermittently pass or fail on the same version of code.

Node.js is an event-driven server-side architecture, in which a user request triggers events, driving Node.js applications to process the user request. However, non-deterministic executions of Node.js applications can cause flaky tests. For instance, a test case creates a timer event $e_{timeout}$. Then, data arrival triggers event e_{data}. Event e_{data} receives the data and cancels the timer event $e_{timeout}$. This is the expected execution for the test case. However, due to non-deterministic event triggering, event $e_{timeout}$ can be triggered before event e_{data} unexpectedly. When processed, event $e_{timeout}$ throws an exception. Flaky tests cause developers not to trust testing results, ignoring actual bugs in the target application, which poses a serious threat to the quality of Node.js applications.

*Xiaoning Chang and Zheheng Liang contribute equally.
¶Guoquan Wu is the corresponding author.

Researchers have conducted empirical studies on flaky tests in multi-threaded programs and Android applications [1], [2], [5], [3]. They found that non-deterministic thread scheduling is the main cause of flaky tests. However, due to different programming models, Node.js applications have a single thread and multiple event queues with different priorities, existing researches cannot reflect characteristics of flaky tests in Node.js applications. The existing work [6] coarsely divides flaky tests in Node.js into Concurrency, Async Wait and other categories. Since non-determinism inside Node.js applications distributes from OS level to application level, it is still difficult to design flaky test detection approaches under the coarse category.

In this paper, we answer a question: *what kind of non-determinism causes flaky tests*, which is critical for designing flaky test detection approaches. We conduct a fine-grained empirical study on 87 flaky tests from 7 popular Node.js applications. We found that, Node.js applications have a wide range of non-determinism to cause flaky tests, including non-deterministic event triggering order, non-deterministic function calls, non-deterministic process/thread scheduling order, non-deterministic execution of asynchronous tasks and non-deterministic event triggering data. The result reveals that, existing approaches on event race detection [7], [8], [9], which mainly focus on non-deterministic event triggering, can fail to detect flaky tests. Researchers should take these different non-determinism into consideration to detect flaky tests.

II. Empirical Study on Flaky Tests

A. Dataset Collection

In this study, we leverage the existing benchmark [6], which collects flaky tests caused by a wide range of causes, including Async Wait, OS and hardware, from Node.js applications. Based on the benchmark, we collect 87 flaky tests caused by non-deterministic execution by removing flaky tests that are not caused by non-determinism at runtime, e.g., the flaky test caused by hardware. Therefore, flaky tests in our study do not correspond to specific categories (e.g., Concurrency) in [6]. The flaky tests we studied come from 7 popular Node.js applications, including electron, node, atom, jQuery, lodash, moment and meteor, covering multiple domains.

2643-1572/23/$31.00 ©2023 IEEE
DOI 10.1109/ASE56229.2023.00025

B. Root Cause of Flaky Tests

According to root causes that lead to flaky tests, we summarize sources of non-determinism into following categories according to the location it happens.

Non-deterministic event triggering order. Events in Node.js applications can be triggered at any time. For example, the timer event is triggered when the timer expires. Network requests can arrive at any time, triggering network-related events. Multiple asynchronous tasks are concurrently executed in the thread pool, and the order in which the completion events of asynchronous tasks are triggered is also uncertain. Non-deterministic event triggering orders cause 45 (52%) flaky tests in our study. Specially, we found that timer events and server shutdown events are particularly prone to causing flaky tests.

Triggering timer events prematurely. Developers register timer events and expect them to be triggered after a certain period of time has elapsed. In other words, other events are expected to be triggered before the timer event. However, in our study, 20 flaky tests were caused by premature timer event triggering.

Timer cancellation event triggered too late. Node.js provides the API $tid = setTimeout(e, d)$ to register a timer tid. After d milliseconds, the timer expires and triggers a timer event e. Node.js also provides the API $clearTimeout(tid)$ to cancel the timer tid that has not yet expired. We found that if a test case calls the API $clearTimeout(tid)$, it usually indicates that the developer expects the timer tid to be canceled and the timer event e not to be triggered. If the timer event is triggered before the timer cancellation event, it can cause the test case to fail. In our study, 6 flaky tests were caused by timer cancellation events being triggered too late.

Triggering server shutdown event prematurely. Since Node.js is a server-side framework, developers usually expect server shutdown events to be triggered at the end of test cases. Premature triggering of the server shutdown events can cause connection interruptions, throwing exceptions such as ECONNRESET, ECONNREFUSED and assertion failures [10], leading to flaky tests. In our study, 12 flaky tests were caused by premature triggering of the server shutdown events.

Non-deterministic function calls. Function $new\ Date()$ obtains the current time. Developers can utilize the difference of two $Date()$ objects to obtain the running time. However, $Date()$ objects return non-deterministic values, which could be out of developer's expectations (e.g., assertions), causing flaky tests. Similar is the case of Function $Math.random()$. In our study, 16 flaky tests are caused by non-deterministic running time.

Non-deterministic process/thread scheduling order. Although the Looper thread is a single thread, Node.js also provides the child_process module [11] and the cluster module [12] to create child processes. Furthermore, to support CPU-intensive tasks, Node.js also provides the worker_threads module [13], which enables parallel execution of multi-threaded JavaScript code. In our study, lack of synchronization among processes and threads can cause 15 (17%) flaky tests.

Non-deterministic execution of asynchronous tasks. A thread pool provides multiple threads to execute asynchronous tasks in parallel. The execution order between asynchronous tasks is non-deterministic. Additionally, the execution order between asynchronous tasks executed in the thread pool and events executed in the Looper thread is also non-deterministic. We found that, unordered asynchronous tasks and events that access to the same resource lead the system to unexpected states, causing 7 (8%) flaky tests in our study.

Non-deterministic event triggering data. Events are triggered with data. In test cases, developers usually intend events to be triggered with specific data. Events triggered with unexpected data can cause 4 (6%) flaky tests in our study. For example, in node#4602 [14], when the client receives the response from server, event $e_{upgrade}$ is triggered with data $upgrade$. Event $e_{upgrade}$ checks if the received data $upgrade$ matches the expected string *nurtzo*. However, if the response data is not in the same data block, the received data $upgrade$ is not the string *nurtzo*, causing a flaky test. In our study, 4 flaky tests are caused by unexpected event triggering data.

Others. We also discovered other non-determinism that caused flaky tests. For example, Node.js relies on the V8 engine to interpret and execute JavaScript code, and the V8 engine provides memory management and garbage collection functions. We found that the background threads of the V8 engine in Node.js release the memory of ArrayBuffer objects concurrently. This causes the size of the memory obtained by the ArrayBuffer objects during the execution of the test case to be non-deterministic and violates expectations of developers, leading to test case failures [15]. In our study, 4 test cases are caused in this way.

Implication. Node.js applications have a wide range of non-determinism to raise unexpected executions, causing flaky tests. Existing event race detection approaches are not sufficient to detect flaky tests. Researchers should take various categories of non-determinism into consideration to detect flaky tests.

III. Conclusion

Node.js applications severely suffer from flaky tests. In this paper, we conduct an empirical study on flaky tests from Node.js applications and find wide range of non-determinism that can cause flaky tests, including non-deterministic event triggering, non-deterministic function calls, non-deterministic process/thread scheduling order, non-deterministic execution of asynchronous tasks and non-deterministic event triggering data. These results can help developers design flaky test detection approaches targeted at different categories of non-determinism.

IV. Acknowledge

This work was partially supported by National Natural Science Foundation of China U20A6003, China Southern Power Grid Company Limited under Project 037800KK58190001.

REFERENCES

[1] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, 2014, pp. 643–653.

[2] A. Vahabzadeh, A. M. Fard, and A. Mesbah, "An empirical study of bugs in test code," in *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 101–110.

[3] W. Lam, K. Muşlu, H. Sajnani, and S. Thummalapenta, "A study on the lifecycle of flaky tests," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2020, pp. 1471–1482.

[4] M. Gruber, S. Lukasczyk, F. Kroiß, and G. Fraser, "An empirical study of flaky tests in python," in *Proceedings of the Conference on Software Testing, Verification and Validation (ICST)*, 2021, pp. 148–158.

[5] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli, "Understanding flaky tests: The developer's perspective," in *Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2019, pp. 830–840.

[6] N. Hashemi, A. Tahir, and S. Rasheed, "An empirical study of flaky tests in javascript," in *Proceedings of International Conference on Software Maintenance and Evolution (ICSME)*, 2022, pp. 24–34.

[7] X. Chang, W. Dou, J. Wei, T. Huang, J. Xie, Y. Deng, J. Yang, and J. Yang, "Race detection for event-driven node. js applications," in *Proceedings of International Conference on Automated Software Engineering (ASE)*, 2021, pp. 480–491.

[8] A. T. Endo and A. Møller, "NodeRacer: Event race detection for Node.js applications," in *Proceedings of International Conference on Software Testing, Validation and Verification (ICST)*, 2020, pp. 120–130.

[9] X. Chang, W. Dou, Y. Gao, J. Wang, J. Wei, and T. Huang, "Detecting atomicity violations for event-driven Node.js applications," in *Proceedings of International Conference on Software Engineering (ICSE)*, 2019, pp. 631–642.

[10] Errors in node.js. [Online]. Available: https://nodejs.org/api/errors.html

[11] Child process module. [Online]. Available: https://nodejs.org/docs/latest-v19.x/api/child_process.html

[12] Cluster module. [Online]. Available: https://nodejs.org/docs/latest-v19.x/api/cluster.html

[13] Worker threads module. [Online]. Available: https://nodejs.org/docs/latest-v19.x/api/worker_threads.html

[14] Test: Fix http-upgrade-client flakiness. [Online]. Available: https://github.com/nodejs/node/commit/d8ba2c0de4681fd22967a9d6d5981f3abe0658a8

[15] Test: Fix flaky test-memory-usage. [Online]. Available: https://github.com/nodejs/node/commit/06deb94714ddb5e18a803c517e9272a8fe657995