Validation Study of Snowboard

Gurpreet Dhillon

Abstract—The purpose of this paper is to validate the findings and techniques of Snowboard through systematical kernel testing using concurrent inputs and sequential kernel test inputs.

Index Terms—Snowboard, concurrency testing, sequential inputs, PMC

1 Introduction

Concurrent execution is difficult to reason about and perform correctly, which makes testing much more tedious. Infrequent process interleavings result in bugs which are hard to anticipate, difficult to find and debug (Heisenbugs), and it is hard to be certain whether these bugs are actually fixed. To tackle these situations, systematical concurrency testing can be used for concurrent systems to find concurrency errors or verify their absence. Also, systematical concurrency testing can be used to explore all possible ways that concurrent execution can influence a computer program's outcome. This paper covers a validation study on a recent systematical kernel concurrency testing framework, Snowboard [1], which generates and runs effective kernel concurrent tests. The study's intention is to improve Snowboard, whilst aiming to answer research questions brought up from performing experiments, that are specific to the workings of the testing framework.

2 Snowboard Overview

Snowboard systemically prioritizes concurrent tests and interleavings through heuristics to find kernel concurrency bugs effectively [1]. To create effective concurrent tests to locate kernel concurrency bugs, Snowboard uses concurrent input, which makes two kernel threads concurrently run the kernel code to serve the input. A concurrent input needs to make system calls, which are the interfaces provided by the kernel to both kernel threads. If the error inducing input is not provided, the kernel code that has the prone concurrency bugs are never reached or executed. This is a very challenging problem, since a sequence of system calls with parameters needs to be chosen, out of the hundreds of system calls available. Also, these system calls can be data-dependent on each other which makes the problem even more difficult to solve.

Moreover, once the concurrent inputs are in effect and two of the threads are running concurrently, the next step is to try to find the specific interleavings that expose the bug. This is a even bigger search space on top of choosing the concurrent inputs, since the number of instructions of the two threads leads to an exponential amount of possible interleavings.

To tackle this challenge, Snowboard predicts possible thread interactions that may happen under concurrent execution, finding the ones that may cause concurrency bugs. It focuses on finding potential memory communications (PMC), which are the interactions between two threads. For example, in a situation where two threads are operating on shared memory and running in parallel, if one thread reads a shared variable that another thread modified, then this results in a PMC, pairing of write-and-read accesses.

Overall, Snowboard's technique finds PMCs in the kernel, prioritizes specific PMCs to test, then tests those PMCs with different interleavings to find any potential concurrency bugs.

2.1 Finding PMCs

To find PMCs, Snowboard uses a technique called dynamic sequential input analysis. This utilizes sequential inputs, which are a sequence of system calls that triggers the execution of code in the kernel, and then is able to list out the identifiable PMCs. It is able to do this because the single thread execution of system calls are analyzed and profiled. Specifically, it profiles the accesses made by the thread to shared memory. After Snowboard repeats these steps on another system call sequence, it predicts the PMCs of the two threads by looking at the execution profiles and searches for the pair of write-and-read accesses that overlap the same memory region. Also to note, these profiles are collected from the same fixed kernel state, so the same resources are accessed.

2.2 Prioritizing PMCs

Testing all PMCs is a very expensive operation and lacks the efficiency, as some PMCs existing in the kernel can be harmless and won't produce bugs. A strategy Snowboard implements prioritizes PMCs for testing. To do this, Snowboard first clusters similar PMCs according to a strategy specified from the seven total clustering strategies [1]. Clustering helps reduce the search space because the similar PMCs can trigger the same behavior.

After the clusters are created, Snowboard prioritizes to the smallest clusters, since they are unlikely to be tested and could lead to rare, unexplored areas. When the priorities are given to each cluster, Snowboard tests a single PMC from each.

2.3 Testing PMCs

In the last step, Snowboard tests the prioritized PMCs. In order to test the PMC under concurrent execution, Snowboard executes the concurrent input on the fixed kernel state (two sequential inputs). This leads to the execution of two kernel threads and runs the code to serve the concurrent input. Once the PMC occurs, Snowboard explores the interleavings between the two kernel threads and monitors the execution to detect any possible concurrency bugs.

3 Goals and Motivation

The main goal of the paper is to answer the following research questions: will an increase in sequential tests and concurrent inputs result in an increase in the number of data races found and what is the most efficient method to ensure maximum data race coverage?

These research questions are derived from the experiments run to understand the usage of Snowboard. The experiments conducted led to more questions since the number of concurrent inputs exponentially increased as the number of sequential test inputs rises. Also, as we increased the sequential test inputs, the number of additional (unique) data races also rose. To reason with the results, another test was performed where unique 2 sequential test inputs were run several times until it covered approximately the same time as a single 16 sequential test input. The test led to interesting findings in terms of data race coverage, the magnitude of concurrent inputs, and sequential test inputs run.

4 Evaluation

In this section, we describe our evaluation of Snowboard [1] and validate the research questions discussed earlier.

4.1 Experimental Setup

In our evaluation, we set up Snowboard on a Linux machine supporting Ubuntu 20.04 LTS, which has an Intel core i9-10900T CPU @ 1.90GHz, 20 Cores, and 32 GB RAM. Testing will be conducted on the Linux kernel 5.12-rc3, provided as the default fixed kernel state by Snowboard. As for PMCs, the research questions don't revolve around the different clustering strategies and can be answered by a single one, in this instance, channel, which identifies PMCs that have the same instruction and memory ranges but have different read/write values, will be chosen for PMC selection.

A workload of small increments were performed, starting from 2 to 16 sequential test inputs. The small workloads were selected due to the limitations of the disk and memory of the equipment used. Afterwards, coverage of data races was tested against 16 sequential test inputs by producing a batch of fifty smaller sequential test inputs, which both lasted about the same time.

4.2 Findings

The exploratory investigation that was carried out contributes to inform the research questions with the results mentioned in Figure 1. The data indicates that profiling more sequential inputs and providing fewer concurrent inputs for the concurrent tests will lead to more data race coverage, in comparison to profiling the same chunk of sequential inputs and providing more concurrent inputs for

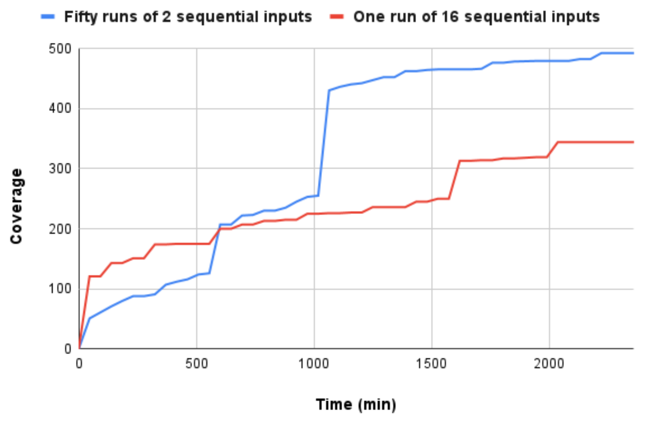

Fig. 1: Two experiments were performed to measure the data race coverage detected in the kernel over time

the concurrent tests. To be more specific, both experiments took about the same time, approximately 2300 minutes, the single run experiment used 12000 concurrent inputs on the same chunk of sequential inputs (16 sequential inputs), and the fifty run experiment used 500 concurrent inputs on 50 sequential input pairings. Though the growth of the coverage started diminishing (similar to a logarithmic curve), it is clear that the most efficient way to maximize data race exposure is to prioritize testing with more sequential inputs, rather than prioritizing more concurrent input coverage on fewer sequential inputs. This also validates the efficiency of Snowboard's method to predict PMCs to detect data races, and subsequently, kernel concurrency bugs.

5 Discussion

In this section, we discuss the limitations we encountered, as well as previous work that helped formulate our research questions and reasoning about the results.

5.1 Limitations

Only a small number of sequential tests were explored out of the vast search space provided by Snowboard [1], due to limitations and capabilities of the equipment used. The number of disk and memory utilized by Snowboard varies dependent on the size of the workload. Analyzing a single sequential test input can take an average of 0.35 GB space. To run larger tests, Snowboard recommended to be able to utilize up to 500 GB RAM and 10 TB disk space with file compression enabled. It is expensive to validate full workloads and run experiments which could take up to weeks, potentially months.

Due to these tight constraints, many variables had to be taken into consideration when analyzing and producing results from the experiments ran on the machine described in section four. For example, the time between invocation and termination couldn't be used for benchmarking these experiments, since these can be inconsistent and affected by other processes. Instead, the user CPU time—how long the experiment was running on the CPU—and system CPU time—how long the experiment was waiting for the operating system to start tasks—was added together and used for benchmarking. Also, the Unix operating system command nice, a utility to alter scheduling priority, was used to schedule experiments with a low priority, in order to reduce the intensity of the workload, allowing for more experiments to be run.

Another limitation was the generator-worker scheme that Snowboard uses to generate and run concurrent tests. Only a single worker is allowed to operate on a clustering strategy queue. Having multiple workers on a queue results in binding issues and workers being preempted from each other. This means the workers running the concurrent tests can't be parallelized, hence taking more time to finish during experimentation.

5.2 Related Work

Past work has focused on dynamic analysis testing to detect race conditions and concurrency errors. In addition to Snowboard [1], many alternative frameworks have been explored – Eraser [2], CHESS [3], PCT [4], FastTrack [5]. The researchers behind Snowboard referenced these papers, since it helped their reasoning in the schedule and input space exploration. Reading these papers gave a better understanding of Snowboard's objectives and helped formulate research questions to explore.

6 Conclusion

In this paper, we presented a validation study of Snowboard [1], a testing framework for detecting kernel concurrency bugs. Snowboard witnesses memory accesses between kernel sequential tests to identify PMCs. Using these PMCs, it runs concurrent tests on these memory communications to detect potential data races and concurrency bugs. The overall goal of this study was to perform a validation on the questions brought up of Snowboard as we were trying to investigate and improve its testing process. The experimental setup had its limitations and our findings led to interesting results, creating more questions of how Snowboard works, and more combinations of tests to run to verify those questions. Although, the research project didn't go in ways we'd expected, there were still a lot of opportunities to learn from and this definitely opened doors to exploring the methodologies testing frameworks like Snowboard are built upon.

Acknowledgements

I would like to thank Dr. Andrew Quinn for providing the equipment to setup Snowboard and being a mentor throughout this study.

References

[1] S. Gong, D. Altinbüken, P. Fonseca, and P. Maniatis, "Snowboard: Finding kernel concurrency bugs through systematic inter-thread communication analysis," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 66–83. [Online]. Available: https://doi.org/10.1145/3477132.3483549

[2] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, "Eraser: A dynamic data race detector for multithreaded programs," *ACM Trans. Comput. Syst.*, vol. 15, no. 4, p. 391–411, nov 1997. [Online]. Available: https://doi.org/10.1145/265924.265927

[3] M. Musuvathi, S. Qadeer, T. Ball, G. Basler, P. A. Nainar, and I. Neamtiu, "Finding and reproducing heisenbugs in concurrent programs." in *OSDI*, vol. 8, no. 2008, 2008.

[4] S. Burckhardt, P. Kothari, M. Musuvathi, and S. Nagarakatte, "A randomized scheduler with probabilistic guarantees of finding bugs," *SIGARCH Comput. Archit. News*, vol. 38, no. 1, p. 167–178, mar 2010. [Online]. Available: https://doi.org/10.1145/1735970.1736040

[5] C. Flanagan and S. N. Freund, "Fasttrack: Efficient and precise dynamic race detection," *SIGPLAN Not.*, vol. 44, no. 6, p. 121–133, jun 2009. [Online]. Available: https://doi.org/10.1145/1543135.1542490