Finding kernel concurrency bugs is important

- Bugs that depend on the instruction schedules.
- Such bugs have serious impact.

Root cause: a kernel concurrency bug
Find kernel concurrency bugs through testing

Input (program):
- \( s = \text{syscall}_1(\ldots) \)
- \( \text{syscall}_2(s, \ldots) \)

Test Generation:
- 
  - Input
  - Input

Schedule:
- 
  - Input
  - Schedule

Test Execution:
- 
  - Kernel

Ordering requirements:
- Kernel thread A
- Kernel thread B

1. ①
2. ②
3. ③
Prior work focuses on optimizing test generation

Snowboard [SOSP’21], Razzer [SP’19], Krace [SP’20]

Find effective pairs of inputs

Test Generation

Input
Input
Schedule

Test Execution
Kernel
Redundant tests reduce the testing efficiency

Each execution is expensive
- 2.8 seconds per test
- 1M redundant tests waste 1 month of machine time
Identify and only execute interesting tests

Support any generation techniques

Test Generation
- Input
- Input
- Schedule

Test Filtering
- interesting?
- Yes
- No

Test Execution
- Kernel

Only interesting tests
- Test result
- Test result
- Test result

Interesting tests
Redundant tests
What are interesting tests?

Thread a

```java
x = "A"
...
if (x == "B") {
    handle_issue()
}
```

Thread b

```java
x = "B"
```

Coverage of different schedules

Schedule 1

```java
x = "A"
...
if (x == "B") {
    handle_issue()
}
```

Schedule 2

```java
x = "A"
...
if (x == "B") {
    handle_issue()
}
```

New code executed. Interesting!
Snowcat predicts the kernel coverage for concurrency tests

Machine learning
Effective in predicting application coverage
Challenges in using ML to build the predictor

1. How to encode a concurrency test to the model?
   - Inputs are userspace programs
   - Schedule is kernel threads ordering

2. How to predict much faster than execute?
   - Predicting a full kernel CFG takes ~3s.
   - A concurrent execution takes ~2s.
1. Represent the schedule on two sequential control flows

Profile kernel sequential execution

Input A

Profile kernel sequential execution

Input A

Kernel control flow triggered by input A

Kernel control flow triggered by input B

Schedule (ordering edge)

Extract thread ordering requirements

Thread A

x = “A”

Thread B

x = “B”

if (x == “B”) {
    handle_issue()
}

Predict the coverage for this concurrent execution
2. Predict faster by only considering adjacent uncovered blocks

If it is covered, the control flow must have diverged.

A tiny subset of all kernel code blocks

zero model: please predict the coverage of each block in this graph
Snowcat encodes even more information in the graph

Kernel control flow triggered by input A

Kernel control flow triggered by input B

Schedule (ordering edge)

Adjacent uncovered blocks

Possible data flows
Implementation of Snowcat

1. Build the predictor using ML

<table>
<thead>
<tr>
<th>Training dataset</th>
<th>Model architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Data 1.3M tests and their coverage</td>
<td>• Code block encoder</td>
</tr>
<tr>
<td>• Kernel version Linux kernel 5.12</td>
<td>• Graph neural networks</td>
</tr>
</tbody>
</table>

2. Use the predictor for testing

<table>
<thead>
<tr>
<th>Predictor integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>• PCT [ASPLOS’10]</td>
</tr>
<tr>
<td>• Razzer [SP’19]</td>
</tr>
<tr>
<td>• Snowboard [SOSP’21]</td>
</tr>
</tbody>
</table>
Snowcat improves testing efficiency significantly

Testing Linux kernel 5.12

Number of data races

Testing time in days

~350 more data races

~6 days faster

~750 more data races

~6 days faster
Snowcat is effective in finding new bugs

New concurrency bugs found in Linux kernel

13 confirmed (6 fixed)
fs/, net/, drivers/, ...
Data loss, DDoS, ...
Existed for years (e.g, 10)
Concern about training cost

Linux kernel
5.12

Collect data, train model

Test kernel

10 days

Collect data, train model

Test kernel

Kernel version Y

Can be reduced or even skipped
Reuse the model for the new kernel

Linux kernel 5.12

Collect data, train model

Test kernel

Kernel version Y

Test kernel

10 days
The model is still effective on the new kernel

Testing **Linux kernel 5.13**

- ~1.5 days faster
- Similar results found when testing **Linux kernel 6.1**
Reuse the model for the new kernel

Linux kernel 5.12

Collect data, train model

Test kernel

10 days

Linux kernel 5.13

Test kernel

The foundational model works well even on new kernels
Fine tune the model for the new kernel

Linux kernel 5.12

Collect data, train model

10 days

Test kernel

Fine tuning

Linux kernel 5.13

Collect data, train model

1 day

Test kernel

3 days saved out of 7

The fine-tuned model brings higher efficiency
You’re welcome to read the paper

More details

• Test Linux 6.1
• Concurrency bug reproduction
• Existing framework integration
Key takeaway from Snowcat

Improve kernel concurrency testing using ML
Predict kernel coverage for concurrency tests

Efficient and effective

https://github.com/rssys/snowcat