西安电子科技大学

Snowcat: Efficient Kernel Concurrency Testing using a Learned Coverage Predictor

汇报人：王少豪

日期：2025.9.25

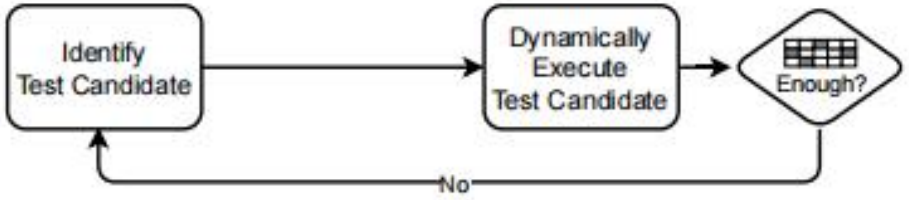
背景

- 1. 内核并发 Bug 危害大
  - 导致数据竞争、死锁、系统崩溃以及安全问题等

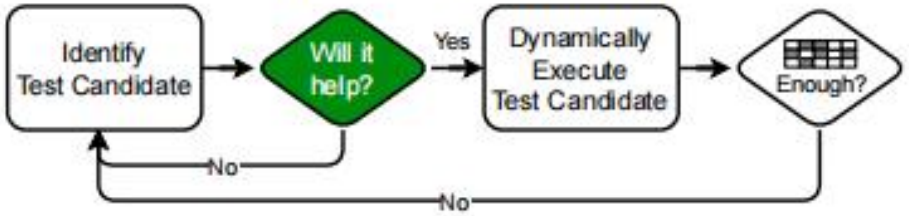
- 2. 现有方法效率低
  - 随机调度 (SKI)：盲测，浪费大量时间在无效测试上。
  - 静态分析 (Razzer)：覆盖不足，很多路径触发不了。
  - 启发式聚类 (Snowboard)：可能错过关键路径。

- 3. 核心问题：无效测试过多
  - 每次执行成本高，但多数测试无新覆盖或 bug

传统测试 vs Snowcat



(a) Generate/execute/cover workflow.



(b) Filtering out dynamic tests that are unlikely to be fruitful.

Figure 2. Comparison between the general workflow and a predictor-based workflow.

传统流程：没有筛选机制，很多 CTIs 没有价值（不会带来新覆盖），却浪费了时间

加入预测器（Snowcat）：提前过滤掉低价值输入，执行数量少，但效率更高。

Step1.输入生成(STIs)

$$STI \rightarrow CTI \rightarrow CT$$

顺序测试输入（STI）：由系统调用序列组成的输入，例如open("file") → write("file") → close("file")，即一条顺序执行的调用链

并发测试输入 (CTI)：由多个STI组成，再让它们在不同线程里并发执行

Thread1：open("file") → write("file")

Thread2：read("file") → close("file")

并发测试候选 (CT)：CTI+调度提示，Snowcat自动生成并插入调度点，比如Thread1执行到第2条指令后切到Thread2。CT是Snowcat的基本分析和执行单元

Step2.并发敏感代码块(SCB及URB)

先单线程运行得到 SCBs，然后通过CFG找出一跳可达的 URBs

SCB：在单线程顺序执行某个STI时实际被执行覆盖的基本块

URB：未覆盖可达代码块，即顺序执行无法覆盖但并发可能覆盖的区域

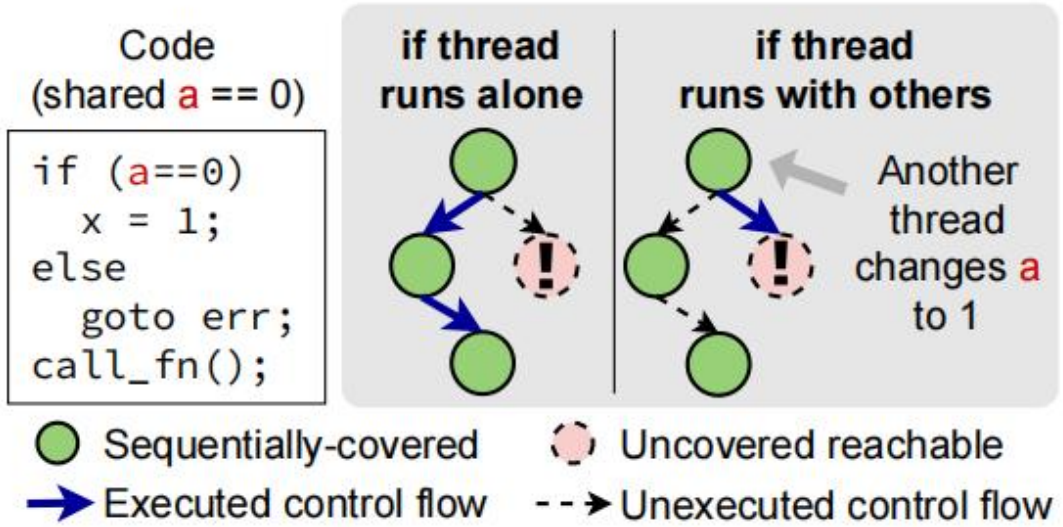


Figure 1. Example of how *uncovered reachable blocks* can be triggered under concurrent executions.

Step3.图表示(CT Graph)

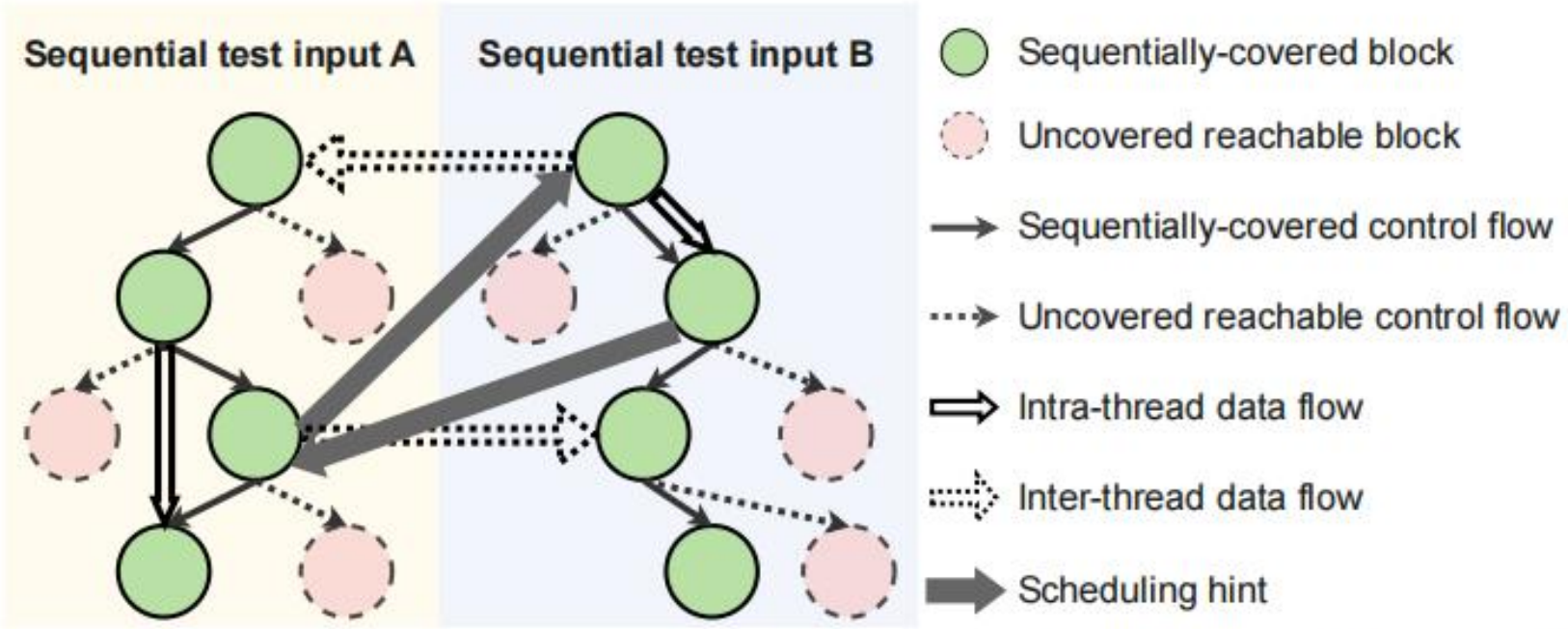


Figure 4. The SNOWCAT graph representation of a CT candidate example.

Step4.覆盖率预测器(PIC模型)预测

PIC=BERT+GNN

BERT：一种基于Transformer的预训练语言模型，负责根据代码块的汇编代码生成代码块的嵌入表示

GNN：以图作为输入，学习代码块嵌入展示之间的关系，捕捉线程间依赖，并对图中的每个节点进行二分类（0/1）

通过使用二元交叉熵损失训练覆盖率预测器，用来衡量预测概率和真实覆盖标签之间差距，保证模型能学会正确区分“代码块会不会被覆盖”，使预测概率尽量贴近真实覆盖

创新点：用深度学习预测覆盖率来筛选测试，从而更高效地发现并发 bug

Step5.策略筛选

S1：每个CT执行后都会产生一个覆盖位图，如果某个CT的覆盖位图和之前的完全一样，则不再执行，以此避免浪费在重复行为上

CT1 覆盖 {A, B, C}，CT2 预测也覆盖 {A, B, C} → 跳过，不再跑

S2：如果这个CT预测的覆盖块全部都已经在过去的执行中被覆盖过，那就不执行，只执行能带来新块覆盖的候选

CT3 预测覆盖 {C, D}，但 D是新块 → 执行

CT4 预测覆盖 {A, B}，A和B之前都覆盖过 → 跳过

S3：块的有限尝试次数。如果某个块已经被尝试了多次，仍然没被覆盖，就不再尝试新的CT去覆盖它，避免在“极难覆盖”的块上反复浪费精力

Step6.并发执行与反馈

- 在内核上执行真正选中的 CT

- 记录覆盖结果+bug信息

- 更新预测器的已知覆盖集，进入下一轮

总结Snowcat的核心就是：预测覆盖→过滤无效→高效发现bug

Snowcat 数据评估

Predictor	F1	Precision	Recall	Accuracy	BA
PIC-5	55.13%	48.34%	69.18%	99.01%	84.47%
All pos	2.17%	1.11%	99.55%	1.11%	49.77%
Fair coin	2.14%	1.10%	49.76%	49.99%	50.00%
Biased coin	1.02%	1.11%	1.17%	97.74%	50.22%

Table 1. *URBs* predictor performance. Average metrics across all graphs. BA stands for balanced accuracy.

结论：PIC-5 是其中唯一有效的预测器，在各项指标（尤其 F1 和 BA）上明显优于随机策略和极端策略，其他方法（All pos, Fair coin, Biased coin）基本无实际预测能力，仅靠策略或随机猜测。PIC-5 能较好兼顾捕获正样本（Recall）和降低误报（Precision），在不平衡数据下仍有较高的 BA，说明在实际应用中更可靠。

Snowcat 与 Razzer 结合

ID	Razzer			Razzer-Relax			Razzer-PIC		
	# CTIs	# TP CTIs	Hours to reproduce (avg / worst)	# CTIs	# TP CTIs	Hours to reproduce (avg / worst)	# CTIs	# TP CTIs	Hours to reproduce (avg / worst)
A	0	0	Na / Na	73	1	17.8 / 35.8	4	1	1.2 / 2.0
B	2	0	Na / Na	76	43	0.9 / 16.7	21	21	0.5 / 0.5
C	0	0	Na / Na	139	1	34.2 / 68.1	7	1	2.0 / 3.4
D	0	0	Na / Na	195	36	2.7 / 78.4	96	22	2.1 / 36.8
E	10	0	Na / Na	322	2	52.4 / 157.4	22	1	5.7 / 10.8
F	430	1	103.8 / 210.8	1119	3	136.1 / 547.5	435	2	72.1 / 212.7

Table 4. Data race reproducing results when using Razzer, Razzer-Relax and Razzer-PIC. "# CTIs " shows the number of CTIs selected by each approach. "# TP CTIs " shows the number of true positive inputs. Worst case to identify a true positive happens if it is at the end of the schedule queue. Average time to reproduce is computed by shuffling the CTI execution queue 1000 times and averaging the time taken to reach the true positive.

Razzer在加入了Snowcat预测期之后，用更少的 CTIs 找到更多 TP，工具筛选能力强，更高效；此外数据竞争平均复现时间显著降低，效率提升了大约15倍

Snowcat 与 Snowboard 结合

Bug ID	Cluster Size	Bug finding probability					# executed CTIs/ (sampling rate)				
		SB-PIC (S1)	SB-PIC (S2)	SB-RND (25%)	SB-RND (50%)	SB-RND (75%)	SB-PIC (S1)	SB-PIC (S2)	SB-RND (25%)	SB-RND (50%)	SB-RND (75%)
15	50	100%	95%	27%	53%	76%	45 (90%)	29 (58%)	12 (25%)	25 (50%)	37 (75%)
16	40	100%	66%	26%	50%	77%	40 (100%)	20 (50%)	10 (25%)	20 (50%)	30 (75%)
17	207	100%	55%	24%	48%	75%	207 (100%)	86 (41%)	51 (25%)	104 (50%)	155 (75%)
18	100	100%	100%	48%	77%	93%	100 (100%)	24 (24%)	25 (25%)	50 (50%)	75 (75%)
19	50	100%	100%	25%	50%	76%	50 (100%)	38 (76%)	12 (25%)	25 (50%)	37 (75%)
21	754	100%	50%	27%	50%	74%	743 (98%)	154 (20%)	188 (25%)	377 (50%)	565 (75%)

Table 5. Results of finding bugs using different sampling methods in Snowboard. Each Snowboard instance is repeated for 1000 times on every buggy INS-PAIR cluster. "Bug finding probability" is the number of runs in which the bug was found divided by 1000, "# executed CTIs" is the average amount of sampled/executed CTIs, and "sampling rate" is the percentage of CTIs sampled from the cluster. "Bug ID" refers to the bugs listed in Table 3b.

SB-RND：发现bug需要高采样率，所以成本过大
SB-PIC(S1)：保证性最好，但成本过高
SB-PIC(S2)：在发现率和效率之间取得最佳平衡

结论：Snowcat的预测(尤其使用S2策略)能大幅减少执行CTIS的数量，同时保持较高的bug发现概率，比随机采样更高效

西安电子科技大学

谢谢大家！

汇报人：王少豪　　日期：2025.9.25