Arm ETM/PMU

Haonan Li, Wenxuan Shi, Xueying Zhang

COMPASS

April 20, 2021

1 Outline

1 Plan of Last Week

- Syscall capturing: how to handle the content of `read`
- Paper writing: revise & rewrite

- `read` syscall:

  ```
  ssize_t read(int fd, void *buf, size_t count);
  ```

  - `fd`: the *file descriptor*
  - `buf` with `count` length: read from `fd` with `count` length to `buf`
  - return the actual length read to `buf`
- original version: only record the `return value` and the `fd`
- record `buf` [Option 1]: record the whole `buf` (may incur considerable overhead)
- record `buf` [Option 2]: truncate the `buf`, only first 256 bytes.

```c
int main(){
    freopen("record","r",stdin);
    int cnt = 0;
    while(scanf("%s",buf) != -1){
        cnt++;
    }
    printf("%d\n",cnt);
}
```

- file `record`: a huge file (2GB), *generated by 100 min of syscall capturing*
- 493,437 `reads`
- typically, the count of `read` is 4096/8192 bytes (for `scanf`, 4096 bytes)
  though `read` *support* `0x7ffff000` *bytes*

1 Performance Test: Read Huge file

perform well on an average program (*while there is no* `read`)

type	real time	record file	Estimated 24-hour file size
baseline	2 min 50.3 s	-	-
syscall capturer (all content)	3 min 11.552 s (+12.5%)	2.0 GB	902.1 GB
syscall capturer (truncate)	2 min 57.675 s (+4.33%)	120 MB	52 GB

type	real time	record file	Estimated 24-hour file size
baseline	130.758 s	-	-
syscall capturer (all content)	134.193 s (+2.6%)	329 MB	144 GB

Figure 1: Overhead for nginx

- Continue to revise the paper before delivery to Zhenyu

2 Outline

Write paper

Commits on Apr 18, 2021

background. concurrency bugs. without pic
hotaru555 committed 20 hours ago ✓

background.breakpoint
hotaru555 committed 22 hours ago ✓

Commits on Apr 17, 2021

related work: add 3 , modify 1 examples
hotaru555 committed 2 days ago ✓

related work 2nd attempt
hotaru555 committed 2 days ago ✓

Commits on Apr 14, 2021

Write paper (add a picture about concurrency bugs and more references).

3 Outline

3 Last week's plan

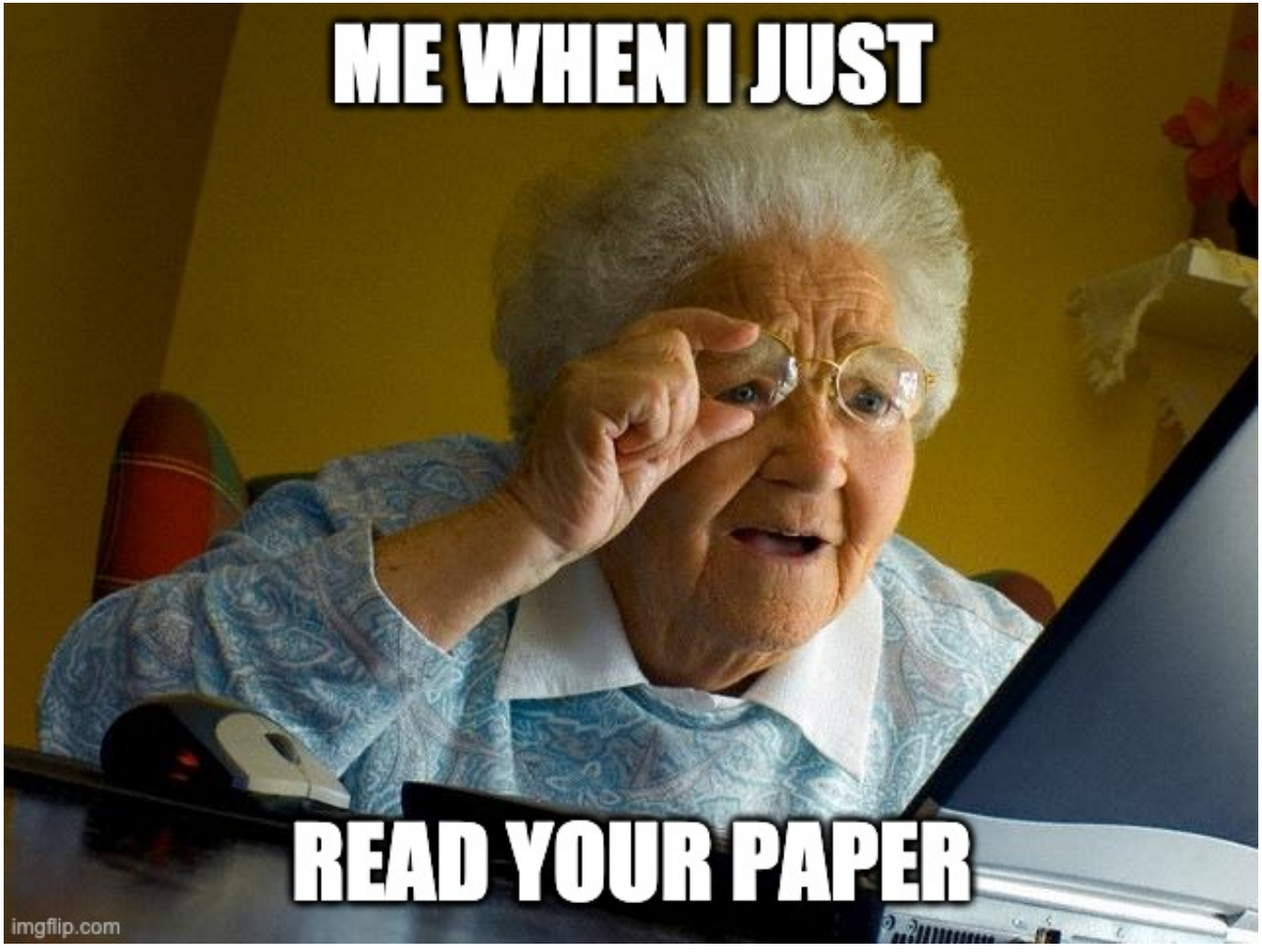
- ☐ Writing the non-concurrency evaluation part (still working on)
- ☒ Help writing other parts

```
- supported by \textit{Control Flow Builder} and \textit{Data Flow Builder} to
- finish the reconstruction of the execution and data of the original application
- in offline analysis phase. Subsequently, based on Builder results, \TheName
- uses the root cause detector to diagnose bugs.

  \subsubsection{Control Flow Builder}
- \textit{Control Flow Builder} uses the decoded ETM trace result and the binary
- code of the program to reconstruct the actual control flow. The control flow is
- arranged in chronological order by the instructions actually executed by
- different threads of the program. Control flow can help us find the specific
- instruction set that caused the program to crash, and then we can further
- analyze the root cause of the crash. Because the control flow is reconstructed
- based on the trace result, the reconstructed control flow only contains
- instructions in the user space. For multi-threaded programs, we use the
- timestamps in trace result to determine the order of instructions of different
- threads.


  \subsubsection{Data Flow Builder}
- \textit{Data Flow Builder} restores the data flow corresponding to the control
- flow, and then the actual operation value of each instruction in the control
- flow can be obtained by searching the data flow. Data flow can help us more
- accurately find out the root cause of program crashes. With the data flow, we
- can find out which instructions are operating on the same address, and then find
- the root cause of the crash caused by the concurrent operation.
-
- REPT uses the control flow and coredump generated when the program crashed to
- recover the data flow. This method combines reverse deduction and sequential
- deduction to complete the recovery. Because most instructions cannot deduct the
- value before the register or memory is changed, some data in the data flow
- cannot be restored or restored to the wrong value. Although not all data is the
- key information for analyzing bugs, a more accurate data flow can more
- accurately analyze the root cause.
-
- For reducing the possible errors in the data flow, we only use sequential
- derivation to restore the data flow. In order to complete the derivation, the
- initial state of each thread is required. We use breakpoints to output a
- coredump when each thread starts, and assume the content of the coredump to be
- the initial state of the thread.
-
- If we can know the state before the instruction is executed, combined with the
- meaning of the instruction, we can deduce the changes that the instruction will
- cause to the register and memory values. With the initial state of all threads
- and the changes made by each instruction in the control flow, we can restore the
- complete data flow.
-
- Because the control flow does not contain instructions executed by the system
- call, we directly use the changes obtained by the syscall capturer to help
- restore the data flow.
-
-
```

```
+ reconstruct the control flow and data flow of the original application in the
+ offline analysis with the help of \textit{Control Flow Builder} and \textit{Data
+ Flow Builder}. Subsequently, based on the results, \TheName uses the root
+ cause detector to diagnose bugs.

  \subsubsection{Control Flow Builder}
+ \textit{Control Flow Builder} uses the decoded ETM trace result and the
+ program's binary code to reconstruct the control flow. Control flow represents
+ the order of execution instructions, which can help us find the specific
+ instruction set that caused the program to crash. Since the control flow is
+ reconstructed based on the trace result, it only contains instructions in the
+ userspace. For parallel programs, we leverage the timestamps in the trace result
+ to determine the order of instructions in different processors.



  \subsubsection{Data Flow Builder}
+ \textit{Data Flow Builder} uses the control flow and coredump, along with a few
+ captured online data, to reconstruct the corresponding data flow. The effect of
+ each instruction in the control flow can be obtained by searching the data flow.
+ Data flow can help us find out the root cause of program crashes more
+ accurately. For example, it helps to reveal the root cause of crashes due to
+ concurrent operations by locating the instructions operating on the same
+ address.
+
+ % comments: Is it resonable to mention other's work in DESIGN part?
+
+ REPT utilizes the control flow and coredump generated by the crashed program to
+ recover the data flow \cite{cui2018rept}. This method combines both forward and
+ backward execution with error correction to complete the recovery.
+ Since most instructions are not reversible (i.e., one cannot trace back the
+ registers and memory status), the accuracy of this method cannot be guaranteed.
+ % Moreover, not all data contain key information for analyzing bugs. Therefore, a
+ % more accurate data flow should be adopted to analyze the root cause.
+
+ To reduce the possible errors in the data flow, we only use sequential
+ derivation to restore the data flow. By combining instructions in the binary
+ code and coredump produced at the initial state of each thread, we can evaluate
+ the changes to memory and registers for each instruction execution.
+
+ We use breakpoints to produce a coredump when each thread starts, and assume the
+ content of the coredump to be the initial state of the thread. Furthermore, we
+ can deduce the changes that the instruction will cause to memory and registers.
+ For system calls, due to the impractical of the kernel instruction trace, we
+ restore the system state based on the online capture information.
+
+ With the initial state of all threads and the changes made by each instruction
+ in the control flow, we can restore the complete data flow.

```

3 Revise Progress

- ☐ Abstract
- ☐ Introduction
- ☒ Background
  - ☒ ETM
  - ☐ Concurrency Bug
- ☐ Related Work
- ☐ Design
  - ☒ Online Record (ETM Manager, Syscall Capturer)
  - ☒ Offline Analysis (Control Flow builder, Data Flow builder)
  - ☐ Root Cause Detector
- ☒ Implementation
  - ☐ Online Record (ETM Manager, Syscall Capturer)
  - ☒ Online Record (Library Hook)
  - ☐ Offline Analysis (Control Flow builder, Data Flow builder)
  - ☐ Root Cause Detector
- ☐ Evaluation
- ☐ Conclusion



Don't consider taking the TOEFL/IETLS/GRE test.

3 Next week's plan

- [ ] Writing the non-concurrency evaluation part (today afternoon)
- [ ] Continue reviewing