Input and Output Coverage Needed in File System Testing

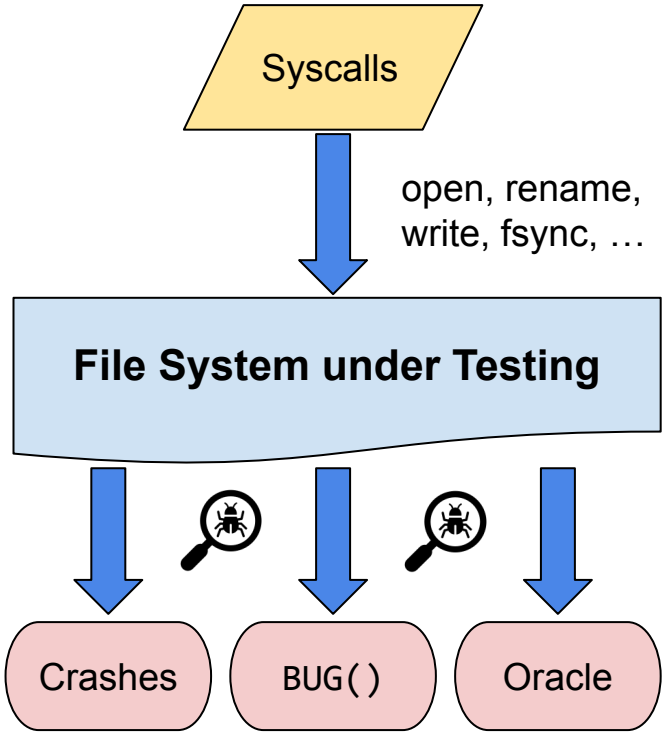
15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '23)

Yifei Liu[1], Gautam Ahuja[1], Geoff Kuenning[2], Scott Smolka[1], and Erez Zadok[1]

[1] Stony Brook University; [2] Harvey Mudd College

File System Testing

- File systems need testing
- Testing file systems is hard
- Various testing approaches
  - Regression testing
  - Model checking
  - Fuzzing
  - Automatic test generation
  - …



Limitations of File System Testing

- Bugs still emerging, even in heavily-tested file systems
- File system testing has to be comprehensive
- Important to evaluate and improve testing
- Code coverage metric
  - E.g., path coverage for fuzzing

Code Coverage

- Measures the extent to which source code is tested by the testing tool
  - **Levels:** lines, functions, branches, etc.
  - Evaluate test completeness and identify untested code
- Limitations of code coverage
  - Obscure connection between test inputs and file system code
  - Effort required to instrument kernel code
- **The correlation between code coverage and file system test effectiveness remains unclear**

Real-World Bug Study

- **Methodology:** check if a test suite can detect the bug when the buggy code is covered
- Studied recent bugs of Ext4 and BtrFS
  - Selected bugs from the latest 100 Git commits of each file system in 2022
  - 51 Ext4 Bugs; 19 BtrFS Bugs [Lu et al.]
- Ran xfstests on Ext4 and BtrFS and recorded its code coverage by Gcov
  - All the generic tests and FS-specific tests
  - Cross-validated if xfstests covered the lines, functions, and branches of the bugs
  - Whether xfstests detected the bugs

Bug Study Results

- **xfstests:** missed bugs in covered code
  - **Lines:** 37/70 bugs (53%) missed
  - Similar phenomenon for function and branch coverage
- Covering code does not mean detecting bugs
- **Code coverage is not strongly correlated with the test effectiveness of file system testing**

Discovering Bugs: What Really Matters?

- Analyzed inputs to trigger bugs from a clean file system
  - **Inputs:** system calls and arguments
- Input Bugs (71% of all bugs)
  - Require specific arguments
- Output Bugs (59%)
  - On the exit path to affect the behavior of syscall returns

```
fs/ext4/xattr.c, v6.0-rc1
sys_lsetxattr(...)
…
 vfs_setxattr(...)
…
  ext4_xattr_set(...)
…
   int ext4_xattr_ibody_set(...) {
-    if (EXT4_I(inode)->i_extra_isize == 0)
+    if (!EXT4_INODE_HAS_XATTR_SPACE(inode))
       return -ENOSPC;
```

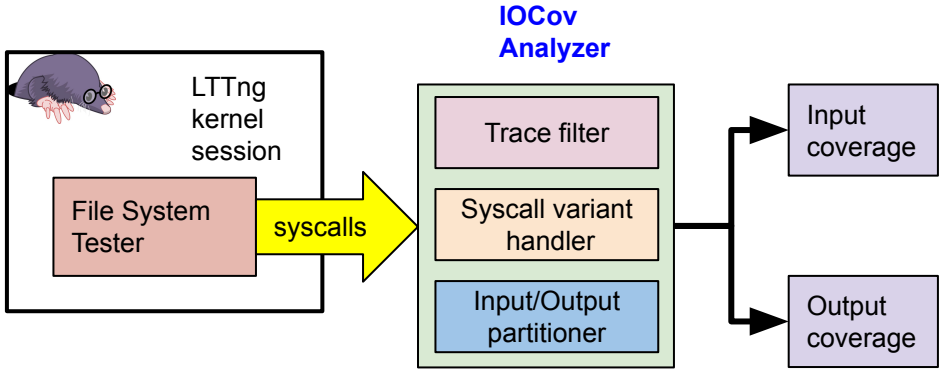
- **Covering both syscall inputs and outputs is essential for file system testing**

Input and Output Partitioning

- Syscall input and output space is massive
  - **Linux:** ~400 syscalls, dozens of them for file systems
  - **Input space:** various arguments, arbitrary values
  - **Output space:** different outputs, error codes
- Input space partitioning
  - **Identifiers:** file descriptors
  - **Bitmaps:** `open()` flags
  - **Numeric arguments:** `write()` size
  - **Categorical arguments:** `lseek()` whence
- Output space partitioning
  - Success or failure; Error codes; Powers of 2 for bytes
- **Input/output coverage:** coverage of input/output partitions

IOCov Framework

- **IOCov:** computing input and output coverage for file system testers
- **Syscall tracing:** LTTng tracer
- **IOCov Analyzer**
  - Trace filter
    - Filter out irrelevant traces
  - Syscall variant handler
    - Merge coverage of variants
  - Input/Output partitioner
    - Partition syscall input/output space
    - Obtain input and output coverage

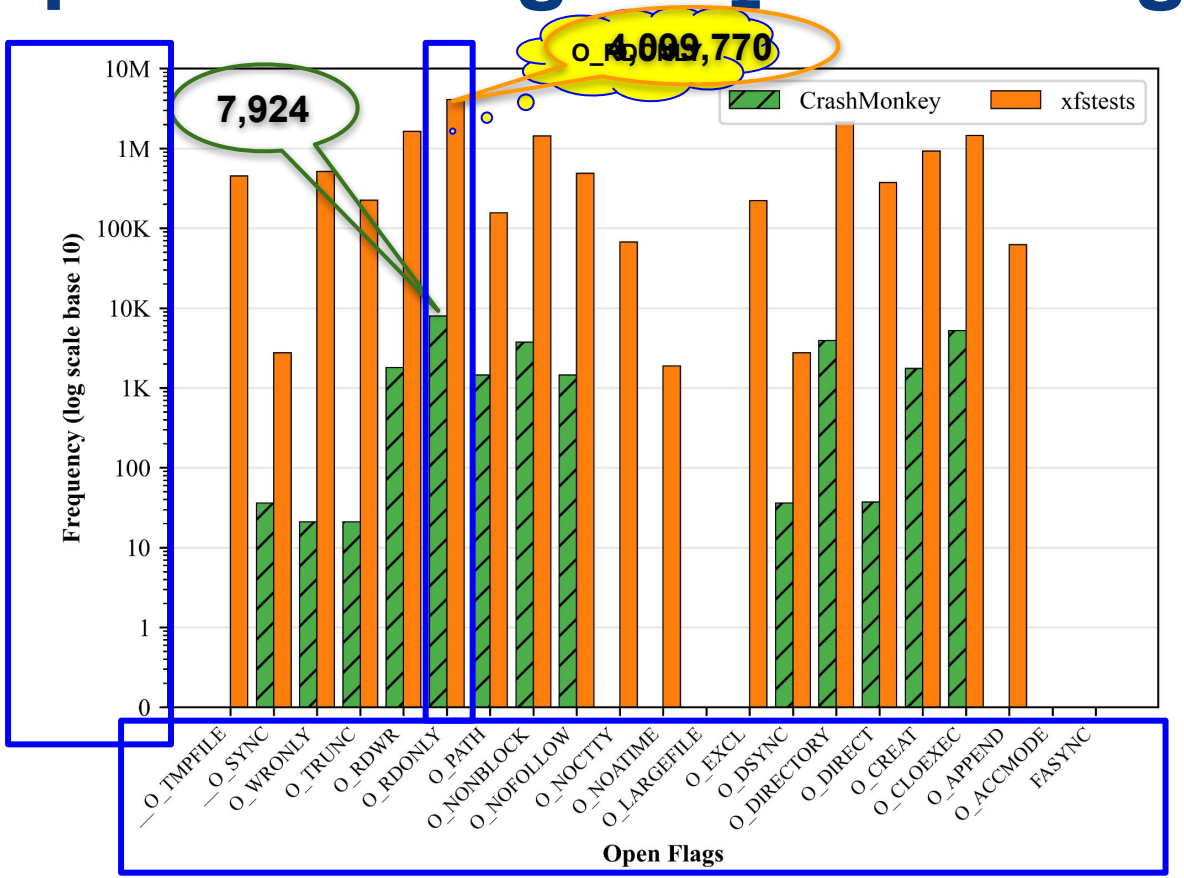


Evaluation Setup

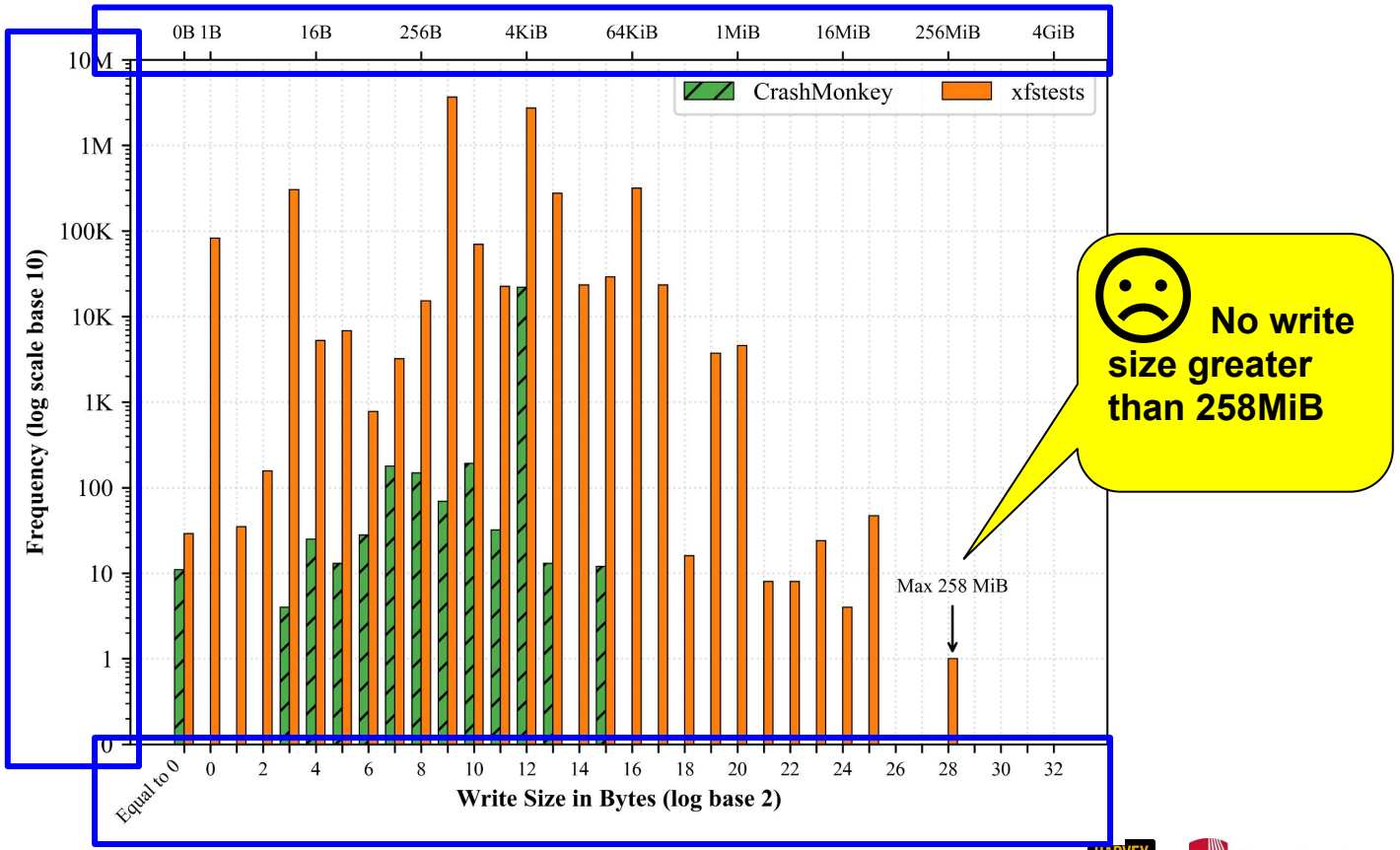
- Input and output coverage for 27 syscalls (including 11 base syscalls)
- Two file system testers to test Ext4
  - **Crashmonkey:** seq-1's 300 workloads and generic tests
  - **xfstests:** all generic and Ext4-specific tests

Base Syscall	Variants	Args Captured
open	openat creat openat2	flags mode

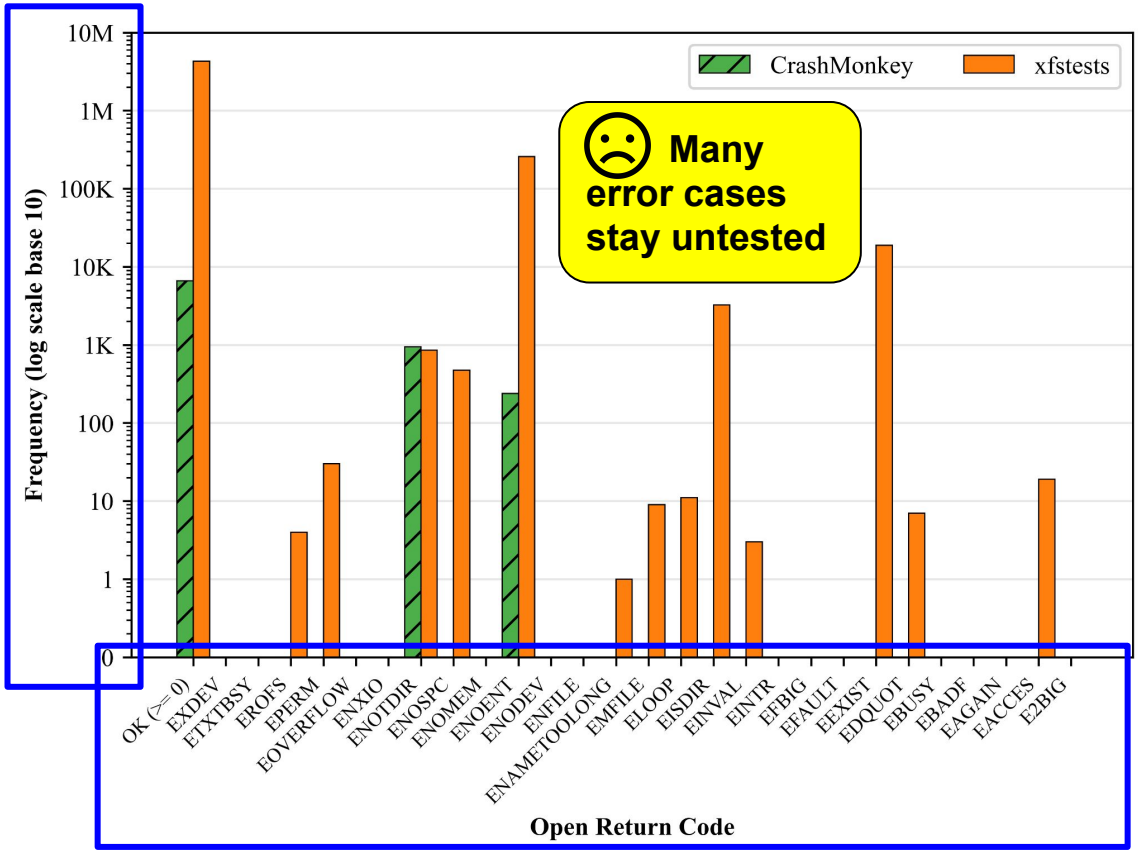
Input Coverage: `open()` flags



Input Coverage: `write()` sizes



Output Coverage: `open()` Returns



Test Coverage Deviation (TCD)

- **Under-testing:** miss bugs
- **Over-testing:** waste resources
- **Test Coverage Deviation (TCD):** Root Mean Square Deviation (RMSD) between coverage and target array, lower is better

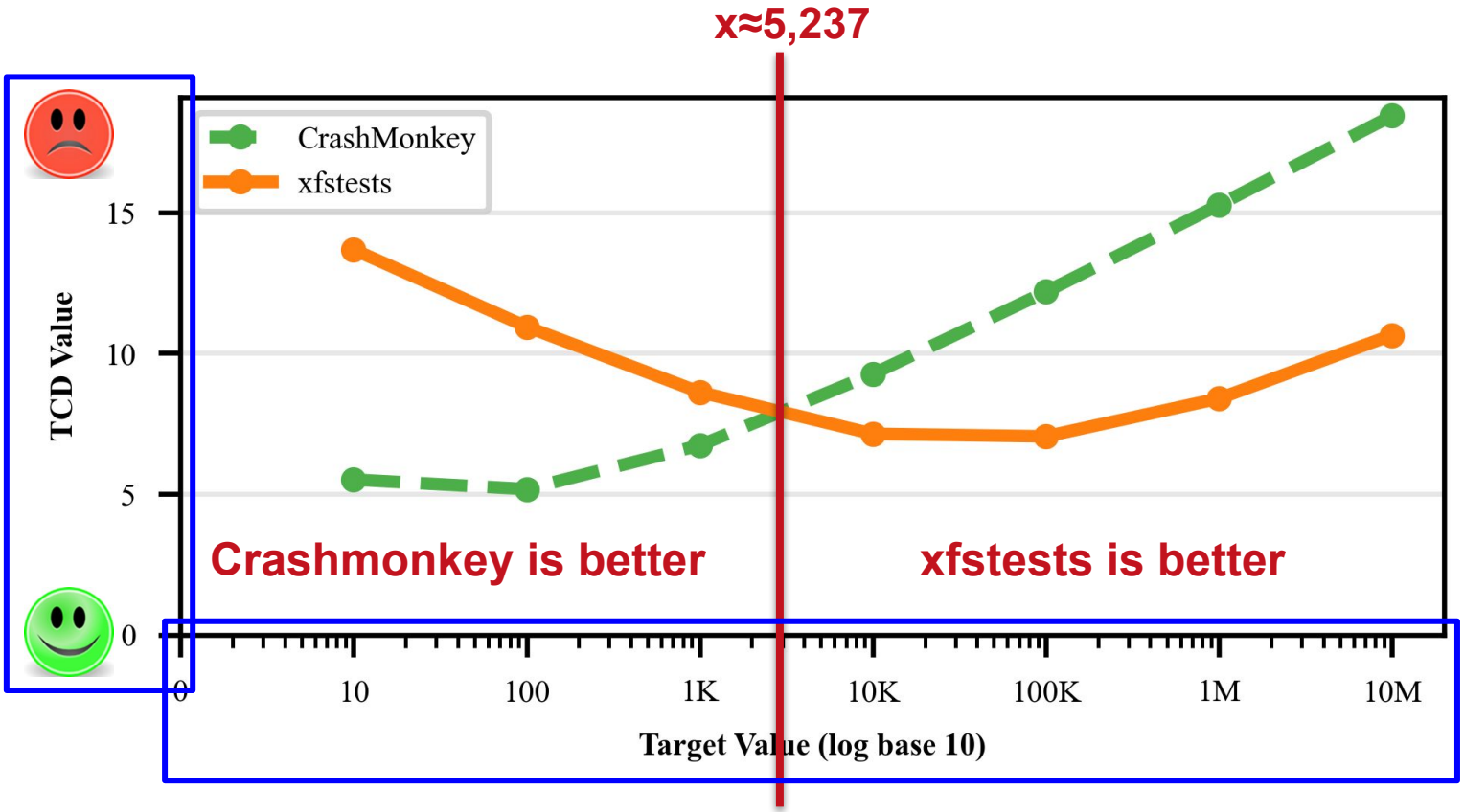
$$TCD_T = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log F_i - \log T_i)^2}$$

Number of Partitions

Frequency

Target

TCD Result: `open()` flags



Conclusions

- Code coverage has a weak correlation with file system test effectiveness
- File system testing requires input and output coverage alongside code coverage
- Existing file system testing needs to cover more inputs and outputs
- Proposed Test Coverage Deviation (TCD) to identify under- and over- testing problems with file system testing

Input and Output Coverage Needed in File System Testing

Thank You

Q&A

yifeliu@cs.stonybrook.edu