Appears as a non-archival poster presentation in the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI '23)

The Case for Model Checking Emerging File Systems

Yifei Liu,[*1] Gerard Holzmann,[2] Geoff Kuenning,[3] Scott A. Smolka,[1] and Erez Zadok[1]
[1]*Stony Brook University,* [2]*Nimble Research,* [3]*Harvey Mudd College*

Abstract. We present MCFS, a new differential-testing-based model-checking framework for emerging file systems. MCFS performs joint state-space exploration on the reference file system and the file system under investigation, and reports any behavioral discrepancies as potential bugs. We created a new file system, VeriFS, as the reference file system. VeriFS uses `ioctl`s to support MCFS and improve model-checking performance, especially compared to kernel file systems running on RAM disks. Applying MCFS to five emerging file systems, we discovered 11 potential bugs, 5 of which have been confirmed by developers. Most of these bugs were previously unknown.

Introduction and Motivation. File systems are the fundamental way an operating system preserves data and interacts with storage devices. A number of file systems have been developed to exploit the potential of new storage technology (*e.g.*, persistent-memory file systems) and to embrace advanced features (*e.g.*, continuous snapshots in NILFS2). We call these relatively new file systems *emerging file systems*. Owing to their critical role, file systems require testing to eliminate bugs and ensure reliability. Although testing has long been applied to established file systems such as Ext4 and XFS, directly using existing test suites on emerging file systems is challenging because adapting tests to the unique attributes of these systems requires substantial effort. Due to a lack of thorough testing, however, emerging file systems tend to hide bugs and thus pose serious vulnerabilities.

We comprehensively studied existing file system testing tools and classified them into four categories. *Regression test suites* (*e.g.*, xfstests and LTP) consist of hand-written test cases to check different file system functionalities. However, modifying regression tests to check a new file system is laborious due to their complexity. *Fuzzing* mutates test cases to stress file systems for maximum path coverage, but requires instrumentation to trace coverage information from the source code. Emerging file systems, however, are not guaranteed to be compatible with code coverage tools (*e.g.*, Gcov) in kernel space. Certain fuzzing tools rely on the utilities of a file system to identify metadata blocks, but many emerging file systems do not yet have such utilities. *Model checking* extracts an abstract model from the file system implementation and checks its adherence to its specification. Prior research has designed models for mature file systems such as Ext4, yet constructing a model for another file system entails an entirely new effort. *Automatic test generation* produces system call sequences automatically as test cases and relies on kernel crashes or `BUG_ON()` calls to identify file system bugs. However, emerging file systems usually involve bugs that do not trigger kernel crashes or `BUG_ON()`. Hence, it is imperative to develop user-friendly testing tools for emerging file systems.

Design and Evaluation. We designed and implemented the MCFS (Model Checking File Systems) framework to test emerging file systems. MCFS runs the file system under test alongside a more trustworthy *reference file system*. MCFS builds on the top of the *Spin* model checker to perform state-space exploration and uses *swarm verification* to scale the exploration. The concrete representation of a file system state is expected to differ from system to system. We therefore designed an appropriate "abstract state" representation for file systems to drive state exploration and to minimize "false positives" that occur when two states are found to be different due to some non-critical implementation detail. The abstract state is a hash value of cross-system consistent data, such as file contents, directory structure, and essential metadata. MCFS uses abstract states to recognize previously-visited states and thus help mitigate the "state explosion" problem.

One major challenge of MCFS is to find a "gold standard" file system as the reference, so that we can identify erroneous behavior in the file system under test. Given the absence of a verified bug-free file system, our initial selection for the reference file system was the widely adopted Ext4 file system, since Ext4 is the de facto file system in the Linux kernel. Subsequently, we developed a new in-memory file system, VeriFS, that supports saving and restoring its full file system state in response to `ioctl` calls. During the development of VeriFS, we used MCFS to discover and fix eight bugs in VeriFS. We have since tested VeriFS against Ext4 for more than 18 days without a discrepancy, exploring more than 1.5 billion system calls and 339 million unique abstract states. This demonstrates the reliability of VeriFS.

MCFS can be scaled with nearly linear performance improvement. In a 13-hour swarm-verification experiment, MCFS ran 6 exploration processes on each of three machines, 18 processes in total. This resulted in exploring 11.2× more unique states than in a single process. We applied MCFS to five in-kernel emerging file systems: BetrFS, F2FS, JFFS2, JFS, and NILFS2, finding discrepancies in all of them. By analyzing MCFS log files, we certified that all of the 11 discrepancies came from problems in the file systems under test. Developers have confirmed 5 of the discrepancies as real bugs. We are investigating the unconfirmed discrepancies and testing more emerging file systems like NOVA and PMFS.

[*]Student at Stony Brook University