Understanding Configuration Issues in Storage Systems

Tabassum Mahmud Mai Zheng
Department of Electrical & Computer Engineering, Iowa State University

1 Motivation

Storage systems are complicated. They often consist of multiple software modules and utility programs, each of which may have a large set of configuration parameters. For example, `mke2fs` has more than 80 parameters to control the configuration states of Ext-family file systems [1]. Similarly, `mdadm` has more than 90 parameters for configuring software RAID [2]. These diverse parameters and their combinations represent a huge space of possible configuration states, which is almost impossible to exhaust for effective testing.

Moreover, there are inherent dependencies among storage software's behaviors and configurations (e.g., `e2fsck`'s behavior may depend on an Ext4 feature which is configured by `mke2fs`), which makes the correctness of storage systems difficult to check or reason.

Great efforts have been made to address configuration-related issues in storage systems or general programs [8, 5, 7, 4, 3]. For example, Carver [3] combines sampling and a greedy algorithm to identify important parameters for storage system tuning. While the methodology is effective for improving storage performance, it relies on a large set of performance traces. Consequently, it cannot not be directly applied to address configuration-related correctness issues in storage systems.

As another example, Xu *et.al.* [8] investigates the configuration requirements of 7 software systems/programs and builds a tool to automatically infer the configuration constraints. However, the tool only considers each program individually, which is fundamentally limited for identifying the complex dependencies exhibit in typical storage systems.

Besides research prototypes, there are practical test suites targeting different storage systems. For example, `xfstests` includes more than 1400 test cases for Ext4, XFS, and other file systems. Similar, `e2fsprogs` includes more than 370 test cases for the utility programs of Ext-family file systems. Nevertheless, we find that the configuration coverage of these state-of-the-art test suites is surprisingly low. For example, based on our investigation, `xfstests` only covers 34.1% configuration parameters of Ext4 file system. Similarly, more than half of the parameters of the file system utilities are not tested by `e2fsprogs`. Such low coverage implies that a new methodology for effectively testing configuration-related issues in storage systems is much needed.

2 Our Approach

To address the challenge, we first conduct an empirical study on 56 configuration-related correctness issues in storage systems to understand the bug patterns and the relation of the bugs to the configuration parameters. We find that among the 56 real bugs studied, 52 cases are related to the file system configurations specified via `mke2fs` parameters, 18 cases are related to the parameters of file system utilities (e.g., `e2fsck`, `resize2fs`, `e2freefrag`), and 14 cases are related to both. Moreover, we identify three general types of parameter dependencies that are critical for manifesting the configuration-related bug cases: (1) *self dependency*, where a parameter is dependent on a predefined set of values of itself; (2) *cross-parameter dependency*, where one parameter is dependent on another parameter of the same program, and (3) *cross-program dependency*, where one parameter or the behavior of one program is dependent on a parameter of another program.

Based on the key findings, we are building a prototype based on the LLVM compiler infrastructure [6] to analyze the parameter dependencies in storage systems automatically. We extend the classic taint analysis to identify the code paths that are related to configuration parameters and derive the three types of dependencies accordingly. Our preliminary results show that the extracted dependencies can help reduce the configuration states for testing configuration-related issues efficiently (e.g., eliminating irrelevant parameter combinations by recognizing the dependencies).

References

[1] E2fsprogs: Ext2/3/4 Filesystem Utilities. http://e2fsprogs.sourceforge.net/.

[2] mdadm - manage MD devices aka Linux Software RAID. https://man7.org/linux/man-pages/man8/mdadm.8.html.

[3] CAO, Z., KUENNING, G., AND ZADOK, E. Carver: Finding important parameters for storage system tuning. In *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)* (2020), pp. 43–57.

[4] DAI, H., MURPHY, C., AND KAISER, G. E. Confu: Configuration fuzzing testing framework for software vulnerability detection. In *Security-Aware Systems Applications and Software Development Methods*. IGI Global, 2012, pp. 152–167.

[5] KELLER, L., UPADHYAYA, P., AND CANDEA, G. Conferr: A tool for assessing resilience to human configuration errors. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)* (2008), IEEE, pp. 157–166.

[6] LATTNER, C., AND ADVE, V. Llvm: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* (2004), IEEE, pp. 75–86.

[7] SUN, X., CHENG, R., CHEN, J., ANG, E., LEGUNSEN, O., AND XU, T. Testing configuration changes in context to prevent production failures. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)* (2020), pp. 735–751.

[8] XU, T., ZHANG, J., HUANG, P., ZHENG, J., SHENG, T., YUAN, D., ZHOU, Y., AND PASUPATHY, S. Do not blame users for misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), pp. 244–259.

Understanding Configuration Issues in Storage Systems

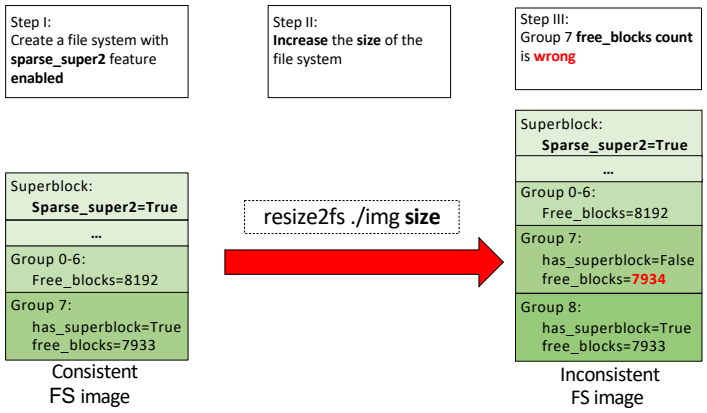
Tabassum Mahmud, Mai Zheng
Department of Electrical and Computer Engineering
Iowa State University

FAST'22
20th USENIX Conference on File and Storage Technologies
FEBRUARY 22–24, 2022
SANTA CLARA, CA, USA
https://www.usenix.org/conference/fast22

Motivation

- **Storage systems have many configuration parameters**
  - Complex software layers with multiple programs
    - e.g., applications, file systems, software RAID, ...
  - Each program having huge number of parameters
    - e.g., *mke2fs* has **85** parameters, *mdadm* has **95** parameters
  - Configuration dependencies among programs
- **Impractical to exhaust all configuration states for testing**
  - e.g., Ext4 has **85** parameters representing over 10^{37} configuration states [2]
  - Leads to the possibility of configuration issues in the programs

Example



Need to minimize the configuration states for effective testing!

Are existing work enough?

- Research Prototypes [1], [2], [3]
  - They may miss many configuration states
  - Do not check cross-program relationships
  - Only cover performance related configurations
  - Rely on profiling traces, which might not be available
- Practical Test suites (xfstests, e2fsprogs, etc.)
  - They may miss many configuration states

Test Suite	Target Program	# of Param.	Coverage
xfstests	Ext4 FS	85	29 (34.1%)
e2fsprogs	e2fsck	35	6 (17.1%)
e2fsprogs	resize2fs	15	7 (46.7%)

Table 1: Coverage of Configuration parameters in Existing Test Suites

Acknowledgements

- This work was supported in part by NSF under grant CNS-1943204. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of NSF.
- We thank Om Rameshwar Gatla, Duo Zhang, Nidhi Dalvi and Haolun Ping from Data Storage Lab for their invaluable discussions and efforts.

Our Approach

Understanding the configuration dependencies

Study Bug Patches of Storage Systems
- To identify configuration bug patterns and classify them
  - Self, Cross-parameter and Cross-program dependency

Automated Configuration Dependency Analyzer

- Automatically identify the configuration dependencies
  - Using inter-procedural Taint Analysis
  - The analysis is context-sensitive and field-sensitive
- Generate critical configuration states based on the dependency
  - Minimizing the number of configuration states

Violating extracted dependencies
- Whether the system handles the violation gracefully

Following extracted dependencies
- Drive deeply into the code and check for unexpected behavior

Preliminary Results

Program Name	# of Bugs	Related to FS config. Parameters	Related to FS Utility Parameters	Related to Both
e2fsck	36	34 (94.4%)	5 (13.9%)	3 (8.3%)
resize2fs	17	15 (88.2%)	13 (76.5%)	11 (64.7%)
e2freefrag	3	3 (100%)	-	-

Table 2: Characterization of Existing Configuration Bugs

Program Name	Self Dependency	Cross-param. Dependency	Cross-program Dependency
mke2fs	26	29	-
e2fsck	4	10	-
resize2fs	-	3	8

Table 3: Manually Derived Configuration Dependency

- We studied 56 existing configuration bugs and derived 3 major types of critical dependency
- Manually extracted these 3 types of dependency from source code
- Developing an automated tool to extract the dependencies
  - Taint Analysis to automatically identify the dependencies
    - Based on LLVM compiler infrastructure [4]
    - Each configuration is tainted and the taint trace is recorded
    - The traces are further used to identify the dependencies

Future Work

- Fully implement automated configuration dependency analyzer
- Evaluate the prototype on more storage systems

References

[1] Xu, Tianyin, "Do not blame users for misconfigurations." *(SOSP)*, 2013.

[2] Cao, Zhen, "Carver: Finding important parameters for storage system tuning." *(FAST)*. 2020.

[3] Keller, Lorenzo, "ConfErr: A tool for assessing resilience to human configuration errors." *(DSN)*, 2008.

[4] Lattner, Chris, "LLVM: A compilation framework for lifelong program analysis & transformation." *(CGO)*, 2004.