Analyzing Configuration Dependencies of DAX File Systems

Tabassum Mahmud, Om Rameshwar Gatla, Duo Zhang, Carson Love, Ryan Bumann, Mai Zheng
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA

1 Introduction

File systems (FS) play an essential role for managing precious data. To meet diverse needs, they often support many configuration parameters. Such flexibility comes at the price of additional complexity which can lead to subtle configuration-related issues. For example, in Dec. 2020, Windows users observed that the `ChkDsk` utility of NTFS may destroy NTFS on SSDs due to a configuration-related bug [2]. With more and more heterogeneous devices (e.g., PM/CXL devices, SmartSSD) and advanced features being introduced (e.g., DAX), the combinatorial explosion of configuration states is expected to exacerbate, which calls for novel solutions.

To address this challenge, we study 78 configuration-related issues of two major Linux file systems with DAX support (i.e., Ext4 and XFS), and identify a prevalent pattern called *multilevel configuration dependencies*. Based on the study, we build an extensible tool called ConfD to extract the dependencies automatically, and create six plugins to address different configuration issues. Our experiments on Ext4 and XFS show that ConfD can extract more than 150 dependencies with a low false positive rate (8.4%). Moreover, the dependency-guided plugins can identify various configuration issues, including inaccurate documentations, configuration handling issues, and regression test failures induced by valid configurations.

2 FS Configurations & Challenges

File System Configurations. A typical file system may be configured through a set of utilities at four different stages, which makes the configuration problem challenging:

- **Create**. When creating file systems, the `mkfs` utility (e.g., `mke2fs` for Ext4) generates the initial configurations.
- **Mount**. When mounting file systems, certain configurations can be specified via `mount` (e.g., '`-o dax`' to enable DAX).
- **Online**. Many utilities can configure a mounted FS directly by modifying the metadata online (e.g., `ChkDsk` for NTFS).
- **Offline**. Offline utilities can also modify file system images and change the configurations (e.g., `resize2fs`, `e2fsck`).

Note that the validation of parameters may occur at both user level and kernel level (e.g., '`-O inline_data`' of `mke2fs` and '`-o dax`' of `mount` are further validated in the `ext4_fill_super` function of Ext4). Therefore, we believe it is necessary to consider the file system itself as well as all the associated utilities as an *FS ecosystem* to address the configuration challenge. For simplicity, we call the file system and utilities as *components* within the FS ecosystem.

FS Test Suites. Practical test suites have been created to ensure the correctness of file systems under various configurations. Unfortunately, their coverage in terms of configuration is limited: less than half of configuration parameters are used in the regression test suites of Linux file systems (i.e., `xfstests`, `e2fsprogs/tests`) based on our calculation. Since each parameter may have a wide range of values representing different states, the total number of missed configuration states is much more than the number of unused parameters, which implies the need of better tool support.

Configuration Constraints & Dependencies. Configuration *constraints* specify the configuration requirements (e.g., data type, value range) of software. Intuitively, such information can help identify important configuration states, and it has proved to be effective for addressing configuration-related issues in a wide range of applications. Configuration *dependency* is one special type of constraint describing the dependent correlation among parameters, which has shown recently to be critical for addressing complex configuration issues in Hadoop and OpenStack. Unfortunately, although the basic concepts have been proposed, there is limited understanding of them in the context of file systems, let alone potential solutions.

3 What Configuration Dependencies Exist in FS

As the first step to address the challenge, we study the Ext4 and XFS ecosystems in depth to understand the real-world patterns of configuration issues. Our dateset includes two parts: (1) the source code of Ext4 and XFS and seven important utilities including `mke2fs`, `mount`, `e4defrag`, `resize2fs`, `e2fsck`, `mkfs.xfs`, and `xfs_repair`; (2) a set of 78 configuration-related bug patches for the two FS ecosystems. We summarize the major findings below.

First, the majority cases (96.2%) involve critical parameters from more than one component, which suggests that we cannot only consider one single component. Second, even within the same an FS ecosystem, the parameters are handled in heterogeneous ways in different components, which makes the problem even more challenging. Interestingly, despite the complexity, 71.8% cases do not require configuration-specific workloads to manifest, which suggests that re-using existing configuration-agnostic tests is possible.

Most importantly, we found a hierarchical pattern which we call *multilevel configuration dependencies*. As summarized in Table 1, the hierarchy includes Self Dependency (SD), Cross-Parameter Dependency (CPD), and Cross-Component Dependency (CCD), each of which may contain a couple of sub-categories which describe more specific constraints. For example, we observed both CPD and CCD between the DAX feature and other Ext4 configurations. In one case, a corruption was triggered when '`-O inline_data`' was used in `mke2fs` and the image was mounted with '`-o dax`' subsequently. In another case, the DAX feature conflicted with the '`has_journal`' configuration, which may lead to corruptions when changing the journaling mode online. Such unexpected dependencies implies the complexity of adding DAX support to the Linux kernel. As we will show in the next sections, this fine-grained characterization of real dependencies enables building dependency-guided tools effectively.

4 How to Extract & Use Dependencies

Based on the study, we build an extensible framework called ConfD to leverage the dependency information to address configuration-related issues. As shown in Figure 1, ConfD consists of two main parts: (1) *ConfD-core* (yellow) for extracting configuration dependencies and generating critical states; (2) *ConfD-plugins* (green) for detecting various configuration-related issues.

ConfD-core. This part contains three sub-modules. First, the

Multilevel Config. Dependencies		Description	Observed?
Self Dependency (SD, 100%)	Data Type	parameter P must be of a specific data type (e.g., integer)	Y
	Value Range	P must be within a specific value range (e.g., $P < 4096$)	Y
Cross-Parameter Dependency (CPD, 10.3%)	Control	$P1$ of $C1$ can be enabled iff $P2$ of $C1$ is enabled/disabled	Y
	Value	$P1$'s value depends on $P2$'s value (e.g., $P1 < P2$)	N
	Behavioral	component $C1$'s behavior depends on $P1$ and $P2$ of component $C1$	Y
Cross-Component Dependency (CCD, 96.2%)	Control	$P1$ of $C1$ can be enabled iff $P2$ of $C2$ is enabled/disabled	Y
	Value	$P1$'s value depends on $P2$ from another component	Y
	Behavioral	component $C1$'s behavior depends on $P2$ of $C2$	Y

Table 1: **Multilevel Configuration Dependencies Derived From Real Configuration Bugs.** *Pn* means parameter, *Cn* means component.

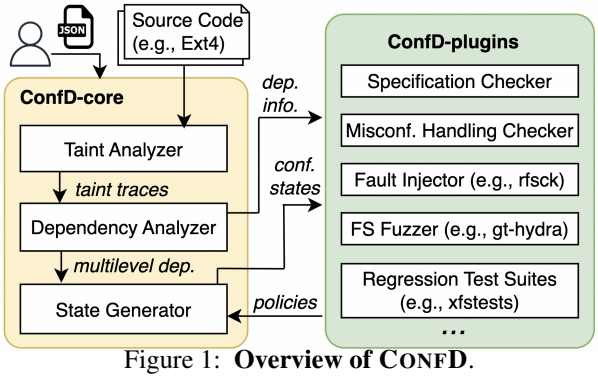

Figure 1: **Overview of ConfD.**

Framework	# of States	# of Duplicate	# of Invalid
FB-HYDRA	56,592	42,745 (75.5%)	15,146 (26.8%)
ConfD	30	0	0

Table 2: **Comparison of State Generation (10 Parameters).**

Taint Analyzer performs metadata-assisted taint analysis and generates taint traces to capture the propagation flow of configuration parameters in the target FS ecosystem. Second, given the taint trace of every parameter, the *Dependency Analyzer* analyzes the potential correlations between parameters based on the multilevel dependencies (§3). Third, the *State Generator* generates concrete configuration states for specific use cases; instead of randomly generating combinations of configurations which can easily lead to useless states, it leverages dependencies to generate critical states selectively. In addition, there is a JSON interface for customization.

ConfD-plugins. The current prototype includes six plugins for addressing different configuration issues. #1 `ConfD-specCk` parses Linux man-pages and looks for potential mismatches between the documented dependencies and those dependencies extracted from the source code. #2 `ConfD-handlingCk` applies invalid configuration states which violate certain dependencies intentionally to see if a misconfiguration can be handled elegantly. #3 `ConfD-rfsck` and #4 `ConfD-gt-hydra` integrate ConfD with two open-source research prototypes (i.e., a fault injector to interrupt the checker and an FS fuzzer), and amplify their effectiveness by providing dependency-guided input images. #5 `ConfD-xfstests` and #6 `ConfD-e2fsprogs` integrate ConfD with standard test suites of Linux file systems and enhance them with dependency awareness.

5 Experiments & Conclusion

We have applied ConfD to analyze Ext4 and XFS and extracted 154 unique dependencies automatically, including 35 SD, 58 CPD, and 61 CCD, with a low false positive rate 8.4% (13/154). Moreover, compared to traditional dependency-agnostic solutions, we found that ConfD can help address configuration-related issues more effectively. For example, Table 2 compares the states generated by FB-HYDRA (a state-of-the-art configuration management framework) and ConfD given the same set of configuration parameters. FB-HYDRA may easily generate many duplicated or invalid states due to lack of dependency guidance. Similarly, `ConfD-rfsk` and `ConfD-gt-hydra` also reported more issues compared to their configuration-agnostic counterparts with dependency-guidence.

ConfD Plugin (Type of Issue Reported)	# of Issue Reported		
	Ext4	XFS	Total
`ConfD-specCk` (undoc./wrong dep.)	13	4	17
`ConfD-handlingCk` (bad reaction)	13	5	18
`ConfD-xfstests` (test case failure)	5	4	9
`ConfD-e2fsprogs` (test case failure)	1	N/A	1
`ConfD-rfsck` (uncorrectable image)	280	–	280
`ConfD-gt-hydra` (hang)	17	–	17

Table 3: **Summary of Issues Observed via ConfD Plugins.**

Table 3 summarizes the configuration issues triggered by ConfD in our experiments. Overall, we observe more than 300 issues of various types, including 17 specification issues, 18 configuration handling issues, 10 regression test failures induced by valid configurations, etc. Note that all the issues require dependency-guided configuration states generated by ConfD to manifest. In other words, continuously running the original research prototypes i.e., rfsck, gt-hydra) or standard test suites (i.e., xfstests, e2fsprogs) cannot expose the issues. ConfD not only helps in reducing states, it also helps with detectability of configuration-related issues.

To conclude, we have presented a study on 78 real configuration-related issues of DAX file systems as well as the ConfD framework for addressing configuration issues. Although ConfD works on all the configurations, since ConfD is designed to be generic and extensible, we expect the core analysis to be applicable to other FS and applications beyond FS. We expect that adding DAX-specific rules and/or other NVM programs (e.g., NDCTL) to ConfD will likely help address NVM-specific configuration issues in NVM software stack more effectively. We hope that by open-sourcing ConfD, our work can facilitate follow-up research in the community.

Original Publication: [1]

Acknowledgments: The authors would like to thank the anonymous reviewers their insightful feedback. This work was supported in part by National Science Foundation (NSF) under grants CNS-1855565, CCF-1853714, CCF-1910747 and CNS-1943204. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsor.

References

[1] Tabassum Mahmud et al. "ConfD: Analyzing Configuration Dependencies of File Systems for Fun and Profit". In: *Proceedings of the 21st USENIX FAST*. 2023.

[2] https://hothardware.com/news/windows-10-20h2-update-damages-ssd-file-systems-chkdsk.