Understanding Configuration Dependencies of File Systems

Tabassum Mahmud, Duo Zhang, Om Rameshwar Gatla, Mai Zheng
Department of Electrical and Computer Enginnering, Iowa State University
<tmahmud,duozhang,ogatla,mai>@iastate.edu

ABSTRACT

File systems have many configuration parameters. Such flexibility comes at the price of additional complexity which could lead to subtle configuration-related issues. To address the challenge, we study the potential configuration dependencies of a representative file system (i.e., Ext4), and identify a prevalent pattern called multi-level configuration dependencies. We build a static analyzer to extract the dependencies and leverage the information to address different configuration issues. Our preliminary prototype is able to extract 64 multi-level dependencies with a low false positive rate. Additionally, we can identify multiple configuration issues effectively.

CCS CONCEPTS

• **Software and its engineering** → **File systems management**; • **Computer systems organization** → **Reliability**.

KEYWORDS

File Systems, Configurations, Dependencies, Reliability

ACM Reference Format:
Tabassum Mahmud, Duo Zhang, Om Rameshwar Gatla, Mai Zheng. 2022. Understanding Configuration Dependencies of File Systems. In *14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '22), June 27–28, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3538643.3539756

1 INTRODUCTION

1.1 Motivation

File systems (FS) play an essential role in modern society for managing precious data. To meet diverse needs, file systems are often designed with a large set of configuration parameters controllable via many utilities (e.g., `mke2fs` [40], `resize2fs` [53]), which enables end users to tune the systems with different tradeoffs. For example, the Ext4 file system contains more than 85 configuration parameters with different types, the combination of which represents over 10^{37} configuration states [8].

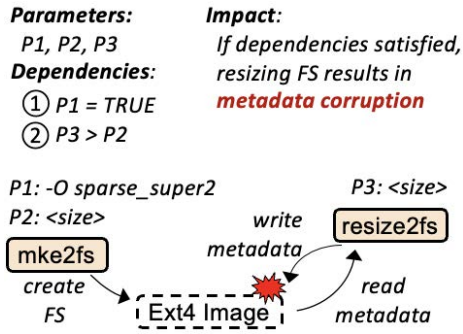

Figure 1: A Configuration-Related Issue of Ext4. When `sparse_super2` feature is enabled and the `size` parameter of `resize2fs` is larger than the Ext4 size, expanding the file system results in metadata corruption.

While configuration parameters have improved the system flexibility, they introduce additional complexity for reliability. Subtle correctness issues often rely on specific parameters to trigger [13, 69]; consequently, they may elude intensive testing and affect end users negatively. For example, in December 2020, Windows users observed that `ChkDsk`, the checker utility of the NTFS file system, destroyed NTFS on SSDs [33, 63]. It was confirmed later that the issue required two specific parameters to manifest: the '`/f`' parameter of `ChkDsk` and another (unnamed) parameter in the Windows operating system (OS) [62].

Similarly, Figure 1 shows another configuration-related issue involving Ext4 and the `resize2fs` utility [53]. Two conditions must hold to trigger the bug: (1) the `sparse_super2` feature is enabled in Ext4 (via `mke2fs`); (2) the value of the `size` parameter of `resize2fs` must be larger than the size of Ext4 (i.e., expanding the file system). Once triggered, the bug will corrupt the Ext4 metadata with incorrect free blocks. The root cause behind the issue was logical, i.e., with the specific configuration parameters, the free blocks count for the last group of file system was calculated before adding new blocks to the file system at the time of expansion.

HotStorage '22, June 27–28, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9399-7/22/06.
https://doi.org/10.1145/3538643.3539756

Due to the combinatorial explosion of configuration states and the substantial time needed to scrutinize file systems under each configuration state [11], it is practically impossible to exhaust all states for testing today. Moreover, with more and more heterogeneous devices (e.g., SmartSSD [59]) and advanced features (e.g., DAX [14]) being introduced, the potential configuration states of file systems are expected to grow rapidly. Therefore, effective methods to help improve configuration-related testing and identify critical configuration issues efficiently are much needed.

1.2 Limitations of the State of the Art

There are practical test suites (e.g., `xfstest` [68]) to ensure the correctness of file systems under different configurations. Unfortunately, their coverage in terms of configuration is limited based on our study: less than half of configuration parameters are used, which reflects the need of better tool support (see §2 for details).

Configuration-related issues have also emerged in other software systems and have received great attention [9, 12, 34, 69]. Unfortunately, existing efforts mostly only consider one single application, which is fundamentally limited for file system configurations involving multiple components (e.g., `ChkDsk` and NTFS, `resize2fs` and Ext4). More discussion is in §2 and §5.

1.3 Contributions

This paper presents one of the first steps to address the increasing configuration challenge of file systems. Inspired by a recent study on configuration issues in cloud systems [9], we focus on *configuration dependency*, which describes the dependent relation among configuration parameters. Such dependency has been identified as a key source of complexity causing configuration problems, and capturing the dependency is essential for improving existing configuration design and tooling [9, 61, 69].

While the basic concept of configuration dependency has been proposed (§2), the understanding of specific dependency patterns and usage in the context of file systems is still limited (to the best of our knowledge). Therefore, we first study the potential configuration dependency of Ext4, the default file system on Linux, by scrutinizing the source code and 67 configuration-related bug cases. In doing so, we answer one important question: what critical configuration dependencies exist in file systems?

Our study reveals a prevalent pattern called *multi-level configuration dependencies*. Many classic configuration constraints (e.g., value range [69]) are still observed in our dataset, which only involves parameters within one single component. More importantly, there are implicit dependencies between parameters from different components of the Ext4 ecosystem, which we call *cross-component dependency*. For example, in Figure 1, the `size` parameters of `mke2fs` and the `size` parameter of `resize2fs` have a cross-component dependency. The majority (97.0%) of issues in our dataset requires meeting such complicated dependencies to manifest, which implies the complexity of the problem as well as the need of new solution.

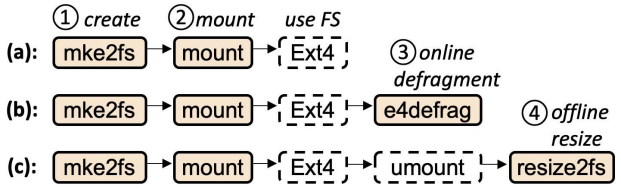

Figure 2: Methods of Configuring File Systems. This figure shows four typical scenarios to configure an FS: (a) at creation (e.g., `mke2fs`) or mount time (`mount`) before usage; (b) via online utilities (e.g., `e4defrag`); (c) via offline utilities (e.g., `resize2fs`).

Next, based on the study, we explore another question: how to extract and use the dependencies with minimal manual effort? One new challenge is to establish the mapping between parameters of different utilities, which tend to have different ways of configuration handling. We address the challenge based on one key observation: all components need to access the FS metadata structures. In other words, we can leverage the shared metadata structures as a bridge to connect relevant configuration parameters of different components.

We incorporate the idea with the classic taint analysis [38] and build a static analyzer based on LLVM [60] to extract the multi-level dependencies automatically. The preliminary prototype is able to extract 64 multi-level dependencies with a low false positive rate (7.8%). Moreover, based on the extracted dependencies, we were able to identify a number of configuration issues efficiently, including 12 inaccurate documentations and 1 bad configuration handling where `resize2fs` may corrupt the file system unexpectedly.

2 BACKGROUND & EXTENDED MOTIVATION

File System Configurations. The configuration methods of file systems are different from that of many applications, which makes it more challenging. As shown in Figure 2, a typical file system may be configured through a set of utilities at four different stages:

- **Create**. When creating file systems, the `mkfs` utility (e.g., `mke2fs` for Ext4) generates the initial set of configurations.

FS (OS)	Four Stages of Configuration			
	Create	Mount	Online	Offline
Ext4 (Linux)	[40]	[44]	[20], [53]	[18], [53]
XFS (Linux)	[43]	[44]	[65], [66]	[64], [67]
BtrFS (Linux)	[42]	[44]	[4], [6]	[5]
UFS (FreeBSD)	[49]	[45]	[29], [54]	[17], [25]
ZFS (FreeBSD)	[71]	[73]	[74], [75]	[72]
MINIX (Minix)	[41]	[46]	–	[23]
NTFS (Windows)	[21]	[48]	[10], [15]	[10], [55]
APFS (MacOS)	[16]	[16], [47]	[16]	[16], [24]

Table 1: Examples of configuration methods for different file systems. The last four columns list example utilities that can affect the configuration states of corresponding file systems.

- **Mount**. When mounting file systems, certain configurations can be specified via `mount` (e.g., '`-o dax`' to enable the DAX feature).
- **Online**. Many utilities can change file system configurations directly by modifying the metadata online (e.g., defragmentater `e4defrag` [20], Windows NTFS checker `ChkDsk` [10]).
- **Offline**. Offline utilities can also modify file system images and change the configurations (e.g., `resize2fs` [53], `e2fsck` [18])

Note that all the utilities have different configuration parameters to control their own behaviors, which will eventually affect the file system state. Moreover, the validation of parameters may occur at both user level and kernel level. For example, the '`-O inline_data`' parameter of `mke2fs` and the '`-o dax`' of `mount` are further validated in the `ext4_fill_super` function of Ext4. So 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.

Also, for simplicity, Figure 2 only shows the (partial) Ext4 ecosystem as an example. In fact, the configuration methodology is common across different file systems. As summarized in Table 1, many popular file systems follow similar modular designs and can be configured via different utilities at multiple stages. In other words, the configuration challenge is not limited to Ext4 or Linux.

FS Test Suites. Practical test suites have been created to ensure the correctness of file systems under various configurations. Unfortunately, due to the complexity of configurations, their coverage in terms of configuration is limited. As shown in Table 2, less than half of configuration parameters are used in the *de fatco* test suites of the popular Ext4 ecosystem (i.e., `xfstest` [68], `e2fsprogs-test` [19]). Note that Table 2 only counts whether a parameter has been used or not. Since each parameter may have a wide range of values representing different states, the total number of un-tested configuration states is much more than the number of unused parameters, which implies the need of tool support.

Test Suite	Target Software	# of Config. Parameter	
		Total	Used
xfstest	Ext4	>85	29 (< 34.1%)
e2fsprogs -test	e2fsck	>35	6 (< 17.1%)
	resize2fs	>15	7 (< 46.7%)

Table 2: Configuration Coverage of Test Suites.

Configuration Constraints & Dependencies. Configuration constraints specify the configuration requirements (e.g., data type, value range) of software [69]. 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 [9, 34, 69, 70]. 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 cloud systems [9]. For simplicity, we use constraints and dependencies interchangeably in the rest of the paper. Note that although the basic concepts have been proposed, there is limited understanding of them in the context of file systems. This paper attempts to fill the gap.

3 WHAT CONFIGURATION DEPENDENCIES EXIST IN FILE SYSTEMS

The key challenge in addressing configuration-related issues of file systems lies in the fact that file systems can be configured at different stages via different utilities (§2). The potential constraints may exist either within individual components or across components, which are often not specified well (largely due to the combinatorial explosion of states). As the first step to address the challenge, we perform a study on the representative Ext4 ecosystem. We present our methodology (§3.1) and key findings (§3.2) in this section.

3.1 Methodology

Our dateset includes two parts: (1) the source code of Ext4 and five important utilities (i.e., `mke2fs`, `mount`, `e2defrag`, `resize2fs`, `e2fsck`, which are described in Table 3); (2) a set of 67 configuration-related bug patches from the Ext4 ecosystem, which are collected via the following two steps:

First, in order to effectively identify configuration-related patches, we apply keyword search to the commit history of the git repositories of Ext4 and its utilities. We use a set of configuration-related keywords, such as 'configuration', 'parameter', 'feature', 'option', etc. The resulting set contains about 2,700 patches.

File System Usage Scenario (key configuration utilities are in bold)	Description	# of Bug	Multi-Level Config. Dependencies		
			SD	CPD	CCD
mke2fs - **mount** - Ext4	create & mount an FS to use	13	13 (100%)	1 (7.7%)	13 (100%)
mke2fs - **mount** - Ext4 - **e4defrag**	online defragmentation	1	1 (100%)	–	1 (100%)
mke2fs - mount - Ext4 - umount - **resize2fs**	resize an umounted FS	17	17 (100%)	–	17 (100%)
mke2fs - mount - Ext4 - umount - **e2fsck**	check FS consistency	36	36 (100%)	4 (11.1%)	34 (94.4%)
	Total	67	67 (100%)	5 (7.5%)	65 (97.0%)

Table 3: Distribution of Configuration Bugs in Four Scenarios. This table shows the distribution of 67 configuration bugs in four typical usage scenarios of file system. The last three columns shows the percentages of bug cases that involve Self-Dependency (SD), Cross-Parameter Dependency (CPD), and Cross-Component Dependency (CCD), respectively.

Multi-Level Config. Dependencies		Description	Exist?	Count
Self Dependency (SD)	Data Type	parameter P must be of a specific data type (e.g., integer)	Y	33
	Value Range	P must be within a specific value range (e.g., $P < 4096$)	Y	30
Cross-Parameter Dependency (CPD)	Control	$P1$ of $C1$ can be enabled iff $P2$ of $C1$ is enabled/disabled	Y	4
	Value	$P1$'s value depends on $P2$'s value (e.g., $P1 < P2$)	N	–
Cross-Component Dependency (CCD)	Control	$P1$ of $C1$ can be enabled iff $P2$ of $C2$ is enabled/disabled	Y	1
	Value	$P1$'s value depends on $P2$ from another component	N	–
	Behavioral	component $C1$'s behavior depends on $P2$ of $C2$	Y	64
		Total	5/7	132

Table 4: A Taxonomy of Critical Configuration Dependencies. This table summarizes the multi-level configuration dependencies observed in our dataset. *Pn* means parameter, *Cn* means component. The last column shows the count of each sub-category of dependency observed.

Second, we randomly sample 400 patches from the set for manual examination. Each of the sampled patch is analyzed by at least two researchers, and those irrelevant to reliability issues or do not rely on specific configurations are excluded based on our domain knowledge. The final set contains 67 configuration-related bug patches.

Note that the methodology is commonly used in empirical studies of practical systems [36, 37, 76]. While time-consuming, it has proved to be valuable for driving system improvements. On the other hand, similar to previous studies, the findings of our study should be interpreted with the method in mind. For example, the patch collection was based on configuration-related keywords and manual examination, which might be incomplete. Nevertheless, we believe such study is one important step to understand and address the configuration challenge.

3.2 Findings

Based on the dataset, we analyze each patch and the relevant source code in depth to understand the logic, which enables us to identify the configuration usage scenarios as well as configuration constraints that are critical. We summarize our findings in Table 3 and Table 4 and discuss them below.

Finding #1: *The majority cases (97.0%) involves critical parameters from more than one components.* The first column of Table 3 shows four typical usage scenarios of Ext4 which cover all bug cases in our dataset (67 in total). 97.0% of the bug cases require specific parameters from at least two key utilities (bold) to manifest. This reflects the complexity of the configuration issues, and suggests that we cannot only consider one single component.

Finding #2: *Multi-level configuration dependencies are prevalent.* We classify the configuration constraints derived from our dataset into three major categories as follows:

- **Self Dependency (SD)** means individual parameters must satisfy their own constraints (e.g., data type or value range). For example, the `blocksize` parameter of `mke2fs` has a value range of 1024 - 65536.
- **Cross-Parameter Dependency (CPD)** means multiple parameters of the same component must satisfy relative relation constraints. For example, two `mke2fs` parameters `meta_bg` and `resize_inode` cannot be used together.
- **Cross-Component Dependency (CCD)** means the parameters or behaviors of one component depend on the parameters of another component. The dependencies described in Figure 1 belong to this category.

As summarized in Table 4, each major category may contain a couple of sub-categories describing more specific constraints. Among them, SD and CPD only involve parameters within one single component, while CCD always involves multiple components. Together, these constraints form the pattern of *multi-level configuration dependencies*. Note that we only observe 5 out of 7 sub-categories in the dataset so far. We include the 2 unseen "Value" sub-categories based on the literature [69] for completeness.

For each observed sub-category, we further count the critical dependencies, i.e., the dependencies directly determine the manifestation of the bug cases (e.g., the dependencies described in Figure 1). We are able to derive 132 critical dependencies manually in total, which is larger than the number of bug cases. This is because a bug case may exhibit multiple critical dependencies (i.e., multi-level configuration dependencies).

As shown in the last three columns of Table 3, SD and CCD are almost always involved in all scenarios (100% and 97% respectively), while CPD is non-negligible (7.5%). This is because SD represents relatively simple constraints which always need to be satisfied the first (e.g., having the correct spelling). Such simple constraints are relatively easy to check and have been the focus of existing work [34]. However, this does not mean that 100% of the bugs could be avoided if SD is checked or satisfied. For example, a bug related to both the `bigalloc` and `extent` parameters (i.e., there is a CPD involved) may still occur even if the two parameters are spelled correctly. In other words, only considering simple constraints (e.g., SD) is not enough.

4 HOW TO EXTRACT & USE MULTI-LEVEL CONFIGURATION DEPENDENCIES

4.1 Deriving Configuration Dependencies

We build a static analyzer based on the LLVM framework [60] and apply the classic taint analysis [38] to track the propagation of each configuration parameter along the data-flow path in the source code. Specifically, we maintain a set to keep the initial configuration variables and any variables derived from the initial configuration variables. When a new variable is added to the set, we add the corresponding instruction to the taint trace too. We maintain a map to track if a variable is derived from multiple parameters. Based on the taint traces, we further analyze the dependencies between variables based on the multi-level dependency patterns derived in our study. The extracted dependencies are stored in JSON files which describe both the parameters and the associated constraints.

One unique challenge we encounter is how to establish the mapping between parameters of different components of the FS ecosystem. Unlike modern cloud systems (e.g., Hadoop [2], OpenStack [50]), the components in the FS ecosystem tend to load configurations in different ways and process equivalent FS information using different variables or functions. We address the challenge based on one key observation: all components need to access the FS metadata structures. So we can leverage shared metadata structures to bridge relevant parameters of different components.

At the time of this writing, the static analyzer can handle intra-procedure taint analysis but not inter-procedure analysis, so we can only extract dependencies via a few preselected functions. But as we will show in §4.3, we can already extract critical dependencies effectively.

4.2 Using Configuration Dependencies

There are various ways to leverage configuration dependencies including configuration fault injection [34], configuration rule management [58], detecting error-prone designs [69], code refactoring, etc. As a starting point, we explore three specific usages: (1) **ConDocCk** checks the potential inconsistency between user manuals and source code in terms of configuration requirements, which has been a longstanding issue in open source software [52]. (2) **ConHandleCk** intentionally violates dependencies to test if the FS ecosystem can handle violations gracefully. (3) **ConBugCk** is a plugin for enhancing existing FS test suites and bug detectors which often have limited configuration coverage (§2). It replaces the configuration loading logic and manipulates configurations without violating dependencies. This is to allow the enhanced tool drive deeply into the target code area (e.g., newly added features) under a variety of configuration states (without early crashing due to shallow errors).

4.3 Preliminary Results

Table 5 summarizes our preliminary results of extracting multi-level configuration dependencies using the static analyzer. Overall, we are able to extract 64 unique dependencies automatically, including 32 SD, 26 CPD, and 6 CCD. The overall false positive rate is 7.8% (5/64), which is comparable to cDEP [9]. Note that our study has shown the importance of identifying CCD (e.g., 97% in Table 3), while we only extract a relatively small number of CCD in the experiments. This is mainly because CCD represents complex relations requiring sophisticated inter-procedure analysis. We expect to extract more dependencies especially CCD once the static analyzer scales out with more complete inter-procedure analysis.

Based on the 59 extracted true dependencies, we have identified 12 inaccurate documentation issues. For example, there is a cross-parameter dependency in `mke2fs` specifying that `meta_bg` and `resize_inode` can not be used together, which is missing from the manual. Moreover, we have found one

File System Usage Scenario (key configuration utilities are in bold)	Self Dependency		Cross-Parameter Dep.		Cross-Component Dep.	
	Extracted	FP	Extracted	FP	Extracted	FP
mke2fs - **mount** - Ext4	31	0	24	1 (4.2%)	0	–
mke2fs - **mount** - Ext4 - **e4defrag**	31	0	24	0	0	–
mke2fs - mount - Ext4 - umount - **resize2fs**	32	3 (9.4%)	26	0	6	1 (16.7%)
mke2fs - mount - Ext4 - umount - **e2fsck**	32	0	26	0	0	–
Total Unique	32	3 (9.4%)	26	1 (3.9%)	6	1 (16.7%)

Table 5: Evaluation Results of Extracting Multi-Level Configuration Dependencies. This table shows the numbers of multi-level dependencies extracted under each scenario. 'FP' means False Positive.

unexpected configuration handling case where `resize2fs` may corrupt the file system.

5 RELATED WORK

Analysis of Software Configurations. Configuration issues have been well studied in many software applications [9, 12, 13, 34, 69]. For example, ConfErr [34] manipulates parameters to emulate human errors; ConFu [13] fuzzes annotated variables in configuration files and tests selected functions. In general, these works do not consider deep dependencies of the software. The closest work is cDEP [9], which looks into the configuration dependencies in cloud systems (e.g., Hadoop, OpenStack). cDEP observes *inter-component dependencies*, which are different from our cross-component dependencies because Hadoop components share XML configuration files and use generic configuration libraries [1], which makes them equivalent to one single program in terms of configuration. In contrast, the dependencies in our study may across different programs and the user-kernel boundary. Also, cDEP relies on a Java framework which cannot handle C-based file systems.

Reliability of File Systems. Great efforts have been made to improve the reliability of file systems [3, 22, 35, 39, 51] and their utilities [27, 28, 30, 56, 57]. For example, Prabhakaran *et al.* [51] analyze the failure policies of four file systems and propose improved designs based on the IRON taxonomy; Spiffy [56] creates an annotation language for developing correct utilities; SQCK [30] and RFSCK [27] improve file system checkers to avoid inaccurate fixes. While effective, these works do not consider multi-component configuration issues. The dependencies derived in this paper could potentially be integrated with existing tools to improve their coverage. Therefore, we view them as complementary.

6 DISCUSSIONS & FUTURE WORK

The work presented in this paper suggests many opportunities for further improvements and follow-up research, e.g.:

Automation, Integration, Evaluation, & Open Source. Our current static analysis requires certain manual annotations, which we hope to reduce. Also, we will fully implement inter-procedure analysis and integrate with complementary tools (e.g., fuzzers) to amplify the effectiveness. We plan to apply the methodology to analyze other popular open-source file systems (e.g., XFS, BtrFS) and evaluate with more metrics (e.g., false negatives, overhead). Ultimately, we hope to develop the prototype into a practical open source tool to help address storage configuration issues in general.

Dependencies between file systems and other software. Researchers and practitioners have observed functionality or correctness dependencies between local file systems and other software (e.g., databases [77], distributed storage systems [7, 26, 31, 32]), many of which are also related to specific configurations (at different layers). The configuration dependencies studied in this work may serve as a foundation for investigating such cross-layer issues, which we leave as future work.

Better configuration design. An alternate perspective of the configuration challenge studied in this work is that we may have too many parameters today. One might argue that it is perhaps better to reduce the parameters to avoid vulnerabilities or confusions, instead of adding new configurations for more features. Also, one might suggest that (in theory) we can implement every utility functionality in the file system itself to replace the modular design commonly used in practice (e.g., Table 1) and thus avoid the multi-level configuration dependencies. Essentially, these are trade-offs of the configuration design that deserve more investigation from the communities, which we leave as future work.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful feedback. We also thank Runzhou Han and Wei Xu for their help on reproducing and validating a few bug cases. In addition, Carson Love and Jahid Hasan helped investigate the configurations on Windows and MacOS. 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] *Apache Common Configuraitons*. https://commons.apache.org/proper/commons-configuration/userguide/upgradeto2_0.html.

[2] *Apache Hadoop*. https://hadoop.apache.org/.

[3] James Bornholt et al. "Specifying and checking file system crash-consistency models". In: *SIGPLAN Not.* 51.4 (2016). DOI: 10.1145/2954679.2872406.

[4] *btrfs-balance*. https://man7.org/linux/man-pages/man8/btrfs-balance.8.html.

[5] *btrfs-check*. https://man7.org/linux/man-pages/man8/btrfs-check.8.html.

[6] *btrfs-scrub*. https://man7.org/linux/man-pages/man8/btrfs-scrub.8.html.

[7] Jinrui Cao et al. "PFault: A General Framework for Analyzing the Reliability of High-Performance Parallel File Systems". In: *Proceedings of the 2018 International Conference on Supercomputing (ICS)*. 2018.

[8] Zhen Cao et al. "Carver: Finding Important Parameters for Storage System Tuning". In: *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST)*. 2020.

[9] Qingrong Chen et al. "Understanding and Discovering Software Configuration Dependencies in Cloud and Datacenter Systems". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 2020.

[10] *chkdsk*. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/chkdsk.

[11] Edmund Clarke et al. "Model Checking and the State Explosion Problem". In: Jan. 2012, pp. 1–30. ISBN: 978-3-642-35745-9. DOI: 10.1007/978-3-642-35746-6_1.

[12] *ctest*. https://cmake.org/cmake/help/latest/manual/ctest.1.html.

[13] Huning Dai et al. "CONFU: Configuration Fuzzing Testing Framework for Software Vulnerability Detection". In: *Int. J. Secur. Softw. Eng.)* 1.3 (2010). DOI: 10.4018/jsse.2010070103.

[14] *DAX: Page cache bypass for filesystems on memory storage*. https://lwn.net/Articles/618064/.

[15] *defrag*. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/defrag.

[16] *disk utility*. https://www.dssw.co.uk/reference/diskutil.html.

[17] *dump*. https://www.freebsd.org/cgi/man.cgi?query=dump&apropos=0&sektion=8&manpath=FreeBSD+13.1-RELEASE+and+Ports&arch=default&format=html.

[18] *e2fsck*. https://linux.die.net/man/8/e2fsck.

[19] *e2fsprogs-test*. https://sourceforge.net/projects/e2fsprogs/files/e2fsprogs-TEST/.

[20] *e4defrag*. https://man7.org/linux/man-pages/man8/e4defrag.8.html.

[21] *format*. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/format.

[22] Daniel Fryer et al. "Recon: Verifying File System Consistency at Runtime". In: *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST)*. 2012.

[23] *fsck*. https://man.minix3.org/cgi-bin/man.cgi?query=fsck&apropos=0&sektion=1&manpath=Minix+3.1.5&arch=default&format=html.

[24] *fsck_apfs*. https://www.manpagez.com/man/8/fsck_apfs/.

[25] *fsck_ufs*. https://www.freebsd.org/cgi/man.cgi?query=fsck_ufs&apropos=0&sektion=8&manpath=FreeBSD+13.1-RELEASE+and+Ports&arch=default&format=html.

[26] Aishwarya Ganesan et al. "Redundancy Does Not Imply Fault Tolerance: Analysis of Distributed Storage Reactions to Single Errors and Corruptions". In: *Proceedings of the 15th Usenix Conference on File and Storage Technologies (FAST)*. 2017.

[27] Om Rameshwar Gatla et al. "Towards Robust File System Checkers". In: *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST)*. 2018.

[28] Om Rameshwar Gatla et al. "Towards Robust File System Checkers". In: *ACM Transactions on Storage (TOS)* 14.4 (2018). DOI: 10.1145/3281031.

[29] *growfs*. https://www.freebsd.org/cgi/man.cgi?query=growfs&apropos=0&sektion=8&manpath=FreeBSD+13.1-RELEASE+and+Ports&arch=default&format=html.

[30] Haryadi S. Gunawi et al. "SQCK: A Declarative File System Checker". In: *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI)*. 2008.

[31] Runzhou Han et al. "A Study of Failure Recovery and Logging of High-Performance Parallel File Systems". In: *ACM Transactions on Storage (TOS)* 18.2 (2022). DOI: 10.1145/3483447.

[32] Runzhou Han et al. "Fingerprinting the Checker Policies of Parallel File Systems". In: *IEEE/ACM Fifth International Parallel Data Systems Workshop (PDSW)*. 2020.

[33] *HotHardware: Windows 10 20H2 Update Reportedly Damages SSD File Systems If You Run ChkDsk*. https://hothardware.com/news/windows-10-20h2-update-damages-ssd-file-systems-chkdsk.

[34] Lorenzo Keller et al. "ConfErr: A tool for assessing resilience to human configuration errors". In: *Proceedings of the 38th IEEE International Conference on Dependable Systems and Networks (DSN)*. 2008.

[35] Seulbae Kim et al. "Finding Semantic Bugs in File Systems with an Extensible Fuzzing Framework". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*. 2019.

[36] Lanyue Lu et al. "A Study of Linux File System Evolution". In: *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST)*. 2013.

[37] Shan Lu et al. "Learning from Mistakes: A Comprehensive Study on Real World Concurrency Bug Characteristics". In: *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2008.

[38] Aravind Machiry et al. "DR. Checker: A Soundy Analysis for Linux Kernel Drivers". In: *Proceedings of the 26th USENIX Conference on Security Symposium (SEC)*. 2017.

[39] Changwoo Min et al. "Cross-Checking Semantic Correctness: The Case of Finding File System Bugs". In: *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP)*. 2015.

[40] *mke2fs*. https://linux.die.net/man/8/mke2fs.

[41] *mkfs*. https://man.minix3.org/cgi-bin/man.cgi?query=mkfs&apropos=0&sektion=1&manpath=Minix+3.1.5&arch=default&format=html.

[42] *mkfs.btrfs*. https://man7.org/linux/man-pages/man8/mkfs.btrfs.8.html.

[43] *mkfs.xfs*. https://man7.org/linux/man-pages/man8/mkfs.xfs.8.html.

[44] *mount*. https://man7.org/linux/man-pages/man8/mount.8.html.

[45] *mount*. https://www.freebsd.org/cgi/man.cgi?query=mount&apropos=0&sektion=8&manpath=FreeBSD+13.1-RELEASE+and+Ports&arch=default&format=html.

[46] *mount*. https://man.minix3.org/cgi-bin/man.cgi?query=mount&apropos=0&sektion=1&manpath=Minix+3.1.5&arch=default&format=html.

[47] *mount_apfs*. https://www.manpagez.com/man/8/mount_apfs/.

[48] *mountvol*. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/mountvol.

[49] *newfs*. https://www.freebsd.org/cgi/man.cgi?newfs(8).

[50] *OpenStack*. https://www.openstack.org/.

[51] Vijayan Prabhakaran et al. "IRON File Systems". In: *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP)*. 2005.

[52] Ariel Rabkin et al. "Static extraction of program configuration options". In: *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*. 2011.

[53] *resize2fs*. https://linux.die.net/man/8/resize2fs.

[54] *restore*. https://www.freebsd.org/cgi/man.cgi?query=restore&apropos=0&sektion=8&manpath=FreeBSD+13.1-RELEASE+and+Ports&arch=default&format=html.

[55] *shrink*. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/shrink.

[56] Kuei Sun et al. "Spiffy: Enabling File-System Aware Storage Applications". In: *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST)*. 2018.

[57] *Swift*. https://docs.openstack.org/swift/latest/.

[58] Chunqiang Tang et al. "Holistic Configuration Management at Facebook". In: *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP)*. 2015.

[59] *The First and Only Adaptive Computational Storage Platform*. https://www.xilinx.com/applications/data-center/computational-storage/smartssd.html.

[60] *The LLVM Compiler Infrastructure*. https://llvm.org/.

[61] Scott Klemmer Tianyin Xu Vineet Pandey. "An HCI View of Configuration Problems". In: *arXiv*. 2016.

[62] *Windows 10 2004/20H2: Microsoft fixes chkdsk issue in update KB4592438*. https://borncity.com/win/2020/12/21/windows-10-2004-20h2-microsoft-fixes-chkdsk-issue-in-update-kb4592438/.

[63] *Windows 10 20H2: ChkDsk damages file system on SSDs with Update KB4592438 installed*. https://borncity.com/win/2020/12/18/windows-10-20h2-chkdsk-damages-file-system-on-ssds-with-update-kb4592438-installed/.

[64] *xfs_admin*. https://man7.org/linux/man-pages/man8/xfs_admin.8.html.

[65] *xfs_fsr*. https://man7.org/linux/man-pages/man8/xfs_fsr.8.html.

[66] *xfs_growfs*. https://man7.org/linux/man-pages/man8/xfs_growfs.8.html.

[67] *xfs_repair*. https://man7.org/linux/man-pages/man8/xfs_repair.8.html.

[68] *xfstest*. https://github.com/kdave/xfstests.

[69] Tianyin Xu et al. "Do Not Blame Users for Misconfigurations". In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP)*. 2013.

[70] Zuoning Yin et al. "An Empirical Study on Configuration Errors in Commercial and Open Source Systems". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP)*. 2011.

[71] *zfs-create*. https://www.freebsd.org/cgi/man.cgi?query=zfs-create&sektion=8&apropos=0&manpath=FreeBSD+13.1-RELEASE+and+Ports.

[72] *zfs-destroy*. https://www.freebsd.org/cgi/man.cgi?query=zfs-destroy&sektion=8&apropos=0&manpath=FreeBSD+13.1-RELEASE+and+Ports.

[73] *zfs-mount*. https://www.freebsd.org/cgi/man.cgi?query=zfs-mount&sektion=8&apropos=0&manpath=FreeBSD+13.1-RELEASE+and+Ports.

[74] *zfs-rollback*. https://www.freebsd.org/cgi/man.cgi?query=zfs-rollback&sektion=8&apropos=0&manpath=FreeBSD+13.1-RELEASE+and+Ports.

[75] *zfs-set*. https://www.freebsd.org/cgi/man.cgi?query=zfs-set&apropos=0&sektion=8&manpath=FreeBSD+13.1-RELEASE+and+Ports&arch=default&format=html.

[76] Duo Zhang et al. "A Study of Persistent Memory Bugs in the Linux Kernel". In: *Proceedings of the 14th ACM International Conference on Systems and Storage (SYSTOR)*. 2021.

[77] Mai Zheng et al. "Torturing Databases for Fun and Profit". In: *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI)*. 2014.