# Hey, You Have Given Me Too Many Knobs!

## Understanding and Dealing with Over-Designed Configuration in System Software

Tianyin Xu*, Long Jin*, Xuepeng Fan*‡, Yuanyuan Zhou*,
Shankar Pasupathy†, and Rukma Talwadker†
*University of California San Diego, USA ‡Huazhong Univ. of Science & Technology, China †NetApp, USA
{tixu, longjin, xuf001, yyzhou}@cs.ucsd.edu
{Shankar.Pasupathy, Rukma.Talwadker}@netapp.com

## ABSTRACT

Configuration problems are not only prevalent, but also severely impair the reliability of today's system software. One fundamental reason is the ever-increasing complexity of configuration, reflected by the large number of configuration parameters ("knobs"). With hundreds of knobs, configuring system software to ensure high reliability and performance becomes a daunting, error-prone task.

This paper makes a first step in understanding a fundamental question of configuration design: "*do users really need so many knobs?*" To provide the quantitatively answer, we study the configuration settings of real-world users, including thousands of customers of a commercial storage system (Storage-A), and hundreds of users of two widely-used open-source system software projects. Our study reveals a series of interesting findings to motivate software architects and developers to be more cautious and disciplined in configuration design. Motivated by these findings, we provide a few concrete, practical guidelines which can significantly reduce the configuration space. Take Storage-A as an example, the guidelines can remove 51.9% of its parameters and simplify 19.7% of the remaining ones with little impact on existing users. Also, we study the existing configuration navigation methods in the context of "too many knobs" to understand their effectiveness in dealing with the over-designed configuration, and to provide practices for building navigation support in system software.

**Categories and Subject Descriptors:** D.2.10 [Software Engineering]: Methodologies

**General Terms:** Design, Human Factors, Reliability

**Keywords:** Configuration, Complexity, Simplification, Navigation, Parameter, Difficulty, Error

## 1. INTRODUCTION

### 1.1 Motivation

In recent years, configuration problems have drawn tremendous attention for their increasing prevalence and severity. For example, Yin et al. reported that configuration issues accounted for 27% of all the customer-support cases in a major storage company in the U.S., and were the most significant contributor (31%) among all the high-severity cases [75]. Rabkin and Katz reported that configuration issues were the dominant source of support cost in Hadoop clusters (based on data from Cloudera Inc.), in terms of both the number of support cases and the amount of supporting time [46].

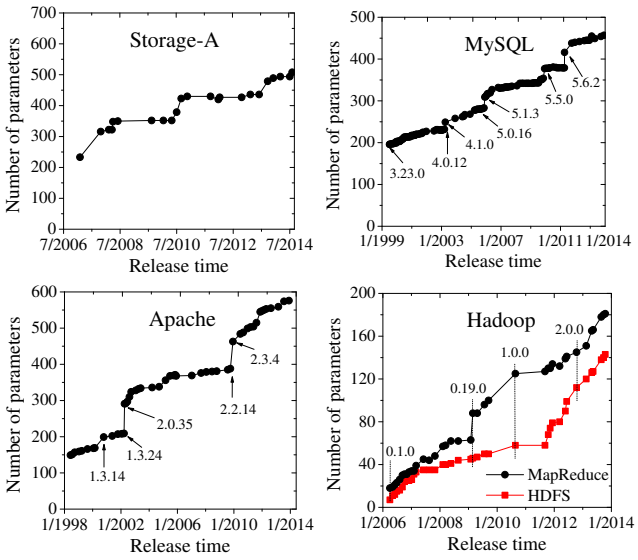


**Figure 1**: **The increasing number of configuration parameters with software evolution. Storage-A is a commercial storage system from a major storage company in the U.S.**

Moreover, configuration errors, the after-effects of configuration difficulties, have become one of the major causes of system failures. Barroso and Hölzle reported that configuration errors were the second major cause of service-level disruptions at one of Google's main services [16]. Recently, a number of outages of Internet and cloud services, including Google, LinkedIn, Microsoft Azure, and Amazon EC2, were caused by configuration errors [35, 59, 63, 68].

One fundamental reason for today's prevalent configuration issues is the ever-increasing complexity of configuration, especially in system software. This is reflected by the large and still increasing number of configuration parameters ("knobs"), as well as various configuration constraints and consistency requirements [32, 39, 45, 72] (known as complexity of interaction and tightness of coupling in human error studies [41,48]). For example, MySQL 5.6 database server has 461 configuration parameters; 216 of them are not with simple data types (e.g., Boolean or enumerative) but rather more complex ones. These parameters control different buffer sizes, timeouts, resource limits, etc. Setting them correctly requires domain-



*ESEC/FSE'15*, August 30 – September 4, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3675-8/15/08...$15.00
http://dx.doi.org/10.1145/2786805.2786852

**Configuration Parameter:** dfs.namenode.tolerate.heartbeat.multiplier

**Developers' Discussion:**

"*Since we are not sure what is a good choice, how about making it configurable?*"

"*We should add a configuration option for it. Even if it's unlikely to change, if someone does want to change it they'll thank us that they don't have to change the code/recompile to do so.*"

**Real-World Usage:**

- No usage found by searching the entire mailing lists and Google.
- No usage reported in a survey of 15 Hadoop users in UCSD.

**Figure 2**: **A real-world example of less useful configuration parameters from HDFS. This parameter is specific to internal system implementation and seldom set by any user. Such parameters should not be exposed to users (at least not to the common users).**

specific knowledge and experience. Similarly, Apache HTTP server 2.4 has more than 550 parameters across all the modules. Moreover, many of these parameters have dependencies and correlations [47, 79], which further worsens the situation. Such high complexity level makes system configuration a daunting, error-prone task (we discuss the cost of such complexity in § 3.3).

Fig. 1 depicts how the number of configuration parameters increases with software evolution of one commercial storage system and three popular open-source server software projects. Take Hadoop MapReduce as an example, its first release in Apr. 2006 had only 17 parameters, but the release in Oct. 2013 already had 173, an increase of more than nine times. In accordance with this trend, the configuration space will continue to increase. The situation is further worsened by the growing number of machines that replicate multiple software instances in data centers, and the increasing cost of human resources for managing them.

Typically, a configuration parameter is introduced when the developers want to provide flexibility for users. As one type of system interfaces, configuration needs to be balanced between flexibility and simplicity. This raises the question if it is worth satisfying a few advanced users or even imaginary users, while confusing the majority of ordinary users. Often, we (software developers) seem to be biased towards "advanced" users instead of focusing on the ease of use. Fig. 2 shows a real-world example where the Hadoop developers exposed a parameter in case some advanced users want it. However, since the parameter is specific to the internal system implementation, few user has set it.

It is worth noting that many configuration parameters are added with new software versions released, but are removed at a much slower rate. The slow rate is probably due to backward compatibility concerns, or developers' lack of sufficient knowledge or confidence in removing parameters introduced by someone else. Table 1 shows the number of parameter changes during software evolution. The parameter removal rate is almost 7x slower than the rate of addition (Storage-A has a faster rate than the open-source software because its release cycles are longer). Following such trends, we will be presented with more configuration parameters in the future releases, unless we take serious actions to simplify them.

Many desktop and mobile applications adopt GUI-based methods such as preference menus [28] to reduce user-perceived complexity caused by large preference space (e.g., colors, fonts, and layouts). Unfortunately, due to the scalability and accessibility issues [64], the primary and *de facto* configuration interfaces of system software are text files (e.g., in xml and .ini formats). GUIs are not widely used for system software configuration [15, 21, 29, 64], making GUI-based methods hard to apply.

In fact, some software developers have already sensed that today's configuration is overly complex and should be simplified to some degree. For example, Rob Pike, the developer of Unix and the Go programming language recently commented,

> "*There is too much configuration. There are too many options. There are too many dot files. Stuff should just work.*"

**Table 1**: **The average number of added, renamed, and removed configuration parameters per version release. The numbers are obtained from the official user manuals.**

| Software | Addition | Renaming | Removal |
|---|---|---|---|
| Storage-A | 13.65 | 0.26 | 1.87 |
| Apache | 5.19 | 0.23 | 0.61 |
| MySQL | 2.54 | 0.06 | 0.53 |
| Hadoop | 4.14 | 1.60 | 0.39 |

## 1.2 Our Contribution

Before jumping onto the simplification track of configuration, we need to answer a fundamental question: "*Do users really need so many knobs?* (after all, configuration knobs are introduced for the purpose of providing flexibility for users)?" However, providing an authoritative answer to this question would require lengthy, expensive, and prohibitively difficult user studies. This paper makes a first step in providing quantitative answers by studying the configuration settings of *many thousands of real users of a commercial system* from a major storage company in the U.S., and also the settings of hundreds of users of two widely-used open-source server software projects. In addition, we study 620 user-reported configuration issues (from both the commercial and open-source software), in order to understand the users' configuration problems (caused by the complexity of current configuration design) in the real world.

**Contribution 1: Understanding Users' Configuration Settings in the Field**

Specially, this paper studies and answers the following questions about configuration in system software.

**(1) Do users really need so many knobs?** Our study shows that only a small percentage (6.1%∼16.7%) of configuration parameters are set by the majority (50+%) of users in the studied systems, while a significant percentage (up to 54.1%) of parameters are seldom set. These results provide quantitative evidence that we (software developers) are providing more knobs than what the majority of users need or know how to set. Many knobs are neither necessary nor worthwhile: they make configuration more complex for common users, but produce little benefit as user-desired flexibility.

**(2) Should we offer more choices in setting a knob?** Software developers often use more "flexible" data types such as numeric types (instead of simple Boolean or enumerative ones) for parameters, in order to give users more flexibility. However, our study shows that users do not take full advantage of such flexibility. A significant percentage (up to 47.4%) of numeric parameters have no more than five distinct settings among all users' settings. Using simpler, enumerative data types would be sufficient. Similarly, for enumerative-typed parameters with many options, only two to three options are actually used. Simplifying these settings can make configuration easier without sacrificing much flexibility.

**(3) What is the "cost" of too many knobs?** One may argue that most parameters have default values; also, users can learn about parameters by referring to user manuals. Thus, there is no real harm in introducing a large number of configuration parameters or providing many options for parameters. However, too many knobs do come with a cost. Our study shows that 17.3%∼48.5% of users'

configuration issues (calls to the technical support center and questions posted on forums) are about their difficulties in finding or setting the correct parameters to achieve the desired system behavior.

"Too many knobs" also prevents users from understanding every parameter thoroughly and examining its settings carefully. It is indirectly reflected by many users' incorrectly keeping the default values for parameters that need to be set based on the runtime environments. Our study on 620 real-world configuration errors reveals that a significant percentage (17.5%∼53.3%) of them occurred because the settings were kept as default values incorrectly, which failed to meet the constraints of runtime environments. Note: these parameters are not the rarely-used ones reported in Finding (1).

**(4) What kinds of knobs are most utilized?** Parameters with explicit external impact are set by more users, compared with those specific to internal system implementation. The reason is intuitive. Software developers, especially open-source developers, often assume too much from users [72]. However, even though the code is open-sourced, it does not mean that users have time or the capability to read and understand the code.

**Contribution 2: Dealing with Over-Designed Configuration Complexity: Simplification and Navigation**

To deal with the over-designed configuration complexity, we further study the effectiveness of two approaches —simplifying configuration design and navigating configuration space.

**(5) How many configurations can we simplify?** Motivated by our study, we provide a set of concrete, practical guidelines for configuration simplification, including reducing unnecessary parameters and simplifying parameter settings. We demonstrate that the configurations of the studied systems can be significantly simplified with little impact on existing users, using these guidelines. Take Storage-A as an example, we can remove 51.9% of its configuration parameters, and further simplify 19.7% of the remaining ones.

**(6) How effective is configuration navigation?** Many software systems rely on the navigation features (e.g., built-in utilities tied in the manuals, searching boxes linked to external search engines) to help users find the configuration parameters and settings. We study the effectiveness of the existing navigation methods, including keyword search, Google search, and NLP-based navigation [4,28]. The study provides useful practices and design guidances for building navigation support for configurations of system software.

# 2. METHODOLOGY

## 2.1 Software

We study four system software projects, including the software of one commercial storage system and three open-source system software, as shown in Table 2. The commercial system, Storage-A, is from a major storage company in the U.S. It runs distributed storage software to manage network-attached storage devices. The open-source software includes Apache HTTP server, MySQL database server, and Hadoop (including MapReduce and HDFS). All these software projects are mature (with 9∼21 years of development history) and widely used, representing different types of system software (storage, Web, database, and data processing).

Note that all the studied software projects fall into the category of *system software*, which is used to provide services for client-side application software (e.g., Web browser, file manager). The users of system software are mostly system administrators and operators who usually have higher level of technical background and skills than ordinary end users. This is particularly true for the commercial storage systems which are mainly purchased and used by enterprise customers (instead of individuals).

**Table 2**: **The system software studied in this paper.**

| Software | Proprietary | Lang. | Dev. his. | # knobs |
|---|---|---|---|---|
| Storage-A | Commercial | — | 22 years | 412 |
| Apache | Open source | C | 21 years | 587 |
| MySQL | Open source | C++ | 21 years | 461 |
| Hadoop | Open source | Java | 9 years | 312 |

**Table 3**: **Configuration setting datasets used in this paper. Note: The numbers of parameters for Apache and MySQL are different from those in Table 2, as we only study the common parameters of the selected versions and exclude parameters of specific OS/modules (§ 2.2).**

| Software | Version | # parameters | System instances |
|---|---|---|---|
| Storage-A | a.b.c | 412 | many thousands |
| Apache | 2.2.x | 90 | 168 |
| MySQL | 5.x | 221 | 260 |

## 2.2 Configuration Settings of Real Users

We collect and study the configuration settings of real users of Storage-A, Apache, and MySQL. The Hadoop dataset we collected is not large enough to be statistically significant. Thus, we exclude Hadoop from the first part of our study on configuration settings. Table 3 summarizes the configuration setting datasets.

The dataset of Storage-A includes the configuration settings of *all the customers using the same version of Storage-A* (anonymized as "a.b.c"). It contains many thousands of customers' settings spanning one and a half years (from 6/1/2012 to 12/31/2013). The data was collected by a support system which recorded the customers' configuration settings on a weekly basis. In our study, we use each customer's most recent configuration settings. Note that this dataset provides the *exhaustive ground truth* of the users' configuration settings of Storage-A, version-a.b.c in the field, and is used as the primary data source in our study.

As the complimentary data references to verify the findings discovered from the Storage-A dataset, real users' configuration settings for Apache and MySQL are also collected from the Internet. We crawl configuration files attached by users from well-known online forums (including ServerFault [9], StackOverflow [10], Pro Webmasters [8], and Database Administrators [5]), and the entire archives of the official mailing lists of the two software projects. We only collect complete configuration files attached by users, and exclude any partial configuration snippets included in postings, because partial snippets does not reflect all the settings made by the user (we cannot know the settings of the parameters not appearing in the snippets); including them would cause biases. Moreover, we only collect *one configuration file per user* (identified by user IDs or emails) to ensure the representativeness of our datasets.

In this study, we use the configuration settings of the same major version for each software project so that we do not need to deal with the configuration difference across versions. To make sure the studied parameters were presented to all the users, we exclude parameters designed for specific plugins or OS (e.g., OS/2, BeOS) that are not the default OS and software modules of the studied software and are not needed for the majority of the users.

## 2.3 Real-world Configuration Issues

We collect 620 real-world cases of user-reported configuration issues related to the studied system software. Table 4 shows the numbers of collected cases of each software project. The cases of Storage-A are collected from the commercial company's customer-issue database which records the issues reported by the customers.

**Table 4: Real-world configuration-related issues studied.**

| Software | Studied cases |
|---|---|
| Storage-A | 329 |
| Apache | 97 |
| MySQL | 96 |
| Hadoop | 98 |

We randomly sampled 1,000 cases labeled as "configuration related" by technical support engineers, and only selected those that have been resolved and confirmed by the original users. For the open-source software, we collect user-reported configuration issues from two sources: the software's official mailing lists and the well-known online forums (the same as in § 2.2). This study focuses on issues related to parameter configuration, as they account for the majority of users' configuration problems [75].

## 2.4 Threats to Validity

As with all characteristic studies, there is an inherent risk that our findings may be specific to the studied software and may not apply to other software. While we cannot establish representativeness categorically, we have taken care to select diverse software with different proprietary licenses, functionalities, and languages of implementation (c.f., Table 2). As all the studied software projects are mature and widely used, we believe that their configuration accurately represents the configuration design practices of today's system software. However, our study only focuses on system software; we do not intend to draw any conclusion about other types of software, such as desktop software and mobile applications.

Another potential source of bias is in the collection of users' configuration settings of the open-source software. The configuration files of Apache and MySQL are collected from online forums and mailing lists. Many of them are associated with users' configuration problems; some of these files may have a few erroneous settings. Although these settings (including the erroneous ones) are configured and applied by real users, we acknowledge that the datasets may be biased to users who encountered configuration problems. To avoid the impact of the potential biases, we only use the datasets as complimentary references to the Storage-A dataset (which does not have such bias). We do not draw any conclusion directly from the open-source datasets. All the reported findings are discovered in Storage-A and then verified in the open-source datasets. As described in § 2.2, *the Storage-A dataset contains all the settings of Storage-A customers using the same version, which is exhaustive without bias to any special type of users.*

Another concern is the representativeness of configuration issues. We collect only user-reported issues. It is possible that users do not report easy-to-solve problems. Also, novice users are more likely to report problems, compared with experts. Unfortunately, it is hard to objectively judge whether a user is a novice or an expert. In fact, with new or major revisions of software being deployed in the field, there are always novice users. Therefore, our findings are still valid. Note: Our study mainly focuses on users' configuration difficulties and mistakes caused by existing configuration design, instead of general characteristics of configuration errors.

Finally, we still remind readers to interpret our findings and results in the context of our studied software and datasets.

# 3. UNDERSTANDING USERS' CONFIGURATION SETTINGS IN THE FIELD

In this section, we first study how the configuration parameters are set by real users. Then, we examine how users handle the increasing configuration complexity.

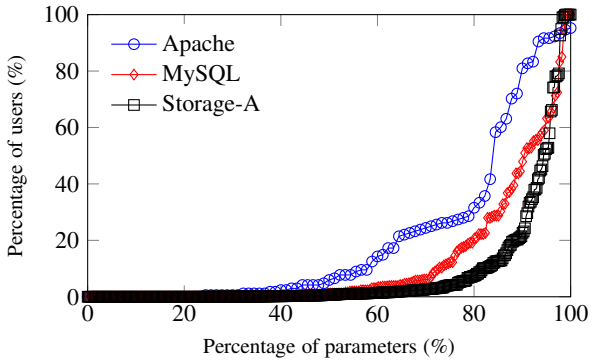


**Figure 3: How many parameters are used in the field by the users? Each data point (x, y) on a curve indicates that "x% of the parameters were set by fewer than y% of the users." Table 5 and 6 further zoom into the parameters set by 0%/1-% of users and 50+%/90+% of users.**

**Table 5: The percentage (number) of parameters that were set by 0% and by fewer than 1% of the users, respectively.**

| Software | % (#) of parameters | | Total (#) |
|---|---|---|---|
| | % of users = 0% | % of users < 1% | |
| Storage-A | 23.3% (96) | 54.1% (223) | 412 |
| Apache | 23.3% (21) | 31.1% (28) | 90 |
| MySQL | 33.0% (73) | 49.8% (110) | 221 |

**Table 6: The percentage (number) of parameters that were set by more than 50% and 90% of the users, respectively.**

| Software | % (#) of parameters | | Total (#) |
|---|---|---|---|
| | % of users > 50% | % of users > 90% | |
| Storage-A | 6.1% (25) | 2.4% (10) | 412 |
| Apache | 16.7% (15) | 7.8% (7) | 90 |
| MySQL | 10.0% (22) | 1.8% (4) | 221 |

Note that *we only study the configuration parameters exposed to users intentionally, instead of those to developers or technical-support engineers*. All the studied parameters are documented in the official user manuals, we exclude hidden parameters that are not visible to the common users. Therefore, all the reported findings in this section are only applicable to configuration design as the users' interface; they may not be applicable to developing, testing or debugging environments (which are discussed in § 6).

## 3.1 Do Users Really Need So Many Configuration Knobs?

**Finding 1(a):** *Only a small percentage (6.1%∼16.7%) of configuration parameters are set by the majority of users; a significant percentage (up to 54.1%) of parameters are rarely set by any user.* It seems that many parameters (at least those rarely-set ones) are not necessary to most of the users. They enlarge the configuration space (adding more complexity) without producing much benefit (in terms of user-desired flexibility) to the common users. We discuss the problems of "too many knobs" in § 3.3. The rarely-set parameters should be separated from the commonly used ones.

Figure 3 plots the real-world usages of configuration parameters, measured by the percentage of users whose settings are different from the defaults in the studied systems. Table 5 shows the percentage of the parameters that are seldom set (by fewer than 1% of users); Table 6 gives the percentage of the parameters set by the majority (more than 50%) of users.

These results give quantitative evidence that we (software developers) seem to have provided more configuration knobs than what

the majority of users need or know how to use. The configuration parameters, rarely set by users, should be either hidden (informing users by requests) or completely removed from common users, to avoid blowing up the user-perceived configuration space.

One may argue that users do not set these parameters only because the default settings are good. First, this may not always be true. As we will show in § 3.3, many times users do not change the default settings because they do not understand the meaning or impact of the parameters and thereby do not know how to set them appropriately. Second, even the above statement is true, if almost all the users never need to set the parameters to be different from the defaults, what is the need to expose these knobs to users? We can keep them hidden or completely remove them so that users can focus on the parameters that need to be changed.

**Finding 1(b):** *A small percentage (1.8%∼7.8%) of parameters are configured by more than 90% of the users.* Many of these parameters provide necessary information of the system runtime which is hard to have default values in advance.

These parameters should be exposed or recommended to the users as the "first-class" knobs. Software developers should provide simple tutorials, guidebooks, and templates, to explain in more details about these parameters. This can help users to focus and have an easier, quicker start, instead of skimming through a thick manual (e.g., MySQL's reference manual has a length of 3,989 pages [7]).

## 3.2 Should We Offer More Choices in Setting Configuration Knobs?

**Finding 2(a):** *Software developers often choose more "flexible" data types for configuration parameters to give users more flexibility of settings (e.g., using numeric types instead of the simple Boolean or enumerative ones). However, users seem not to take full advantage of such flexibility. A significant percentage (up to 47.4%) of numeric parameters have no more than five distinct settings among all the users' settings.*

Once again, this implies that the developers' goodwill in providing user flexibility is not fully appreciated by users. Reducing the *value space* of these parameters can simplify their settings, without sacrificing much flexibility. Developers can convert the complex types into Boolean or enumerative types which are more expressive and easier for users to configure.

Table 7 shows the similarity of users' settings for numeric parameters in the studied systems. For each numeric parameter, we select the five most popular values, and measure their coverage among all the users' settings. We exclude users who stayed with the default values (i.e., they did not set these parameters), in order to avoid the dominating coverage of the default values (including the default values would make the percentages even higher, because most users go with the defaults for many numeric parameters).

Note that the majority numeric parameters have a large *range*. Many parameters do not have specified min/max values, so their ranges depend on their data types. For example, the data range of parameters represented by `unsigned int` is [0, `UINT_MAX`]. We do not normalize the results in Table 7 by the numeric ranges (as denominators), because large ranges (e.g., [0, `INT_MAX`] and [0, `LONG_MAX`]) seldom have meanings to users and thus may not impact users' settings (making normalization nonsensical).

One reason of the small number of settings is the distribution of templates among the user communities. For example, the user communities of the configuration management tools (e.g., Puppet and Chef) have the tradition of sharing *recipes* for a variety of common software. The results indicate that the majority of users do not need large value space for configuration parameters. Providing users with a few representative options covering typical usage scenarios is simpler and more efficient.

**Table 7**: **The percentage of numeric parameters with no more than five distinct settings used by 90% or 100% of the users. We exclude users who did not set the parameters.**

| Software | % of parameters with five distinct values | |
|---|---|---|
| | Covering 90% users | Covering 100% users |
| Storage-A | 65.8% | 47.4% |
| Apache | 60.0% | 10.0% |
| MySQL | 26.7% | 12.2% |

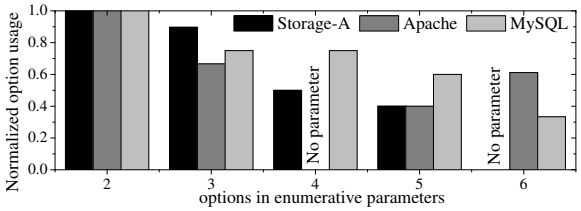


**Figure 4**: **Usages of enumerative parameters with different number of options (percentages of used options among all the provided options)**

**Finding 2(b):** *Similarly, for enumerative parameters with many options, typically only two to three of the options are actually used by the users, indicating once again the over-designed flexibility.* Figure 4 shows the number of used options among all the options.

Compared with numeric ones, enumerative parameters have less options with more representative values. However, if there are too many options with ambiguous semantics, users tend to stay on a few *safe* options. For example, Apache's `LogLevel` parameter has 16 options, corresponding to 16 different logging verbosity levels [2]. Though the user manual explains each level using log examples, only six options appear in our datasets. The log levels for specific debugging purposes (e.g., `trace1`–`8`) should not be exposed to the common users. In fact, even the developers themselves are often confused by the verbosity levels, not to mention the users [76].

## 3.3 What Is The "Cost" of Too Many Knobs?

Some software developers may argue that most of the parameters have default values; also, users can learn about these parameters by referring to user manuals. Thus, there is no real harm in introducing a large number of configuration parameters or providing many options for them. However, this argument is somewhat refuted by our study of real-world configuration issues reported by users.

**Finding 3:** *Too many knobs do come with a cost: users encounter tremendous difficulties in knowing which parameters should be set among the large configuration space. This is reflected by the following two facts: (1) a significant percentage (up to 48.5%) of configuration issues are about users' difficulties in finding or setting the parameters to obtain the intended system behavior; (2) a significant percentage (up to 53.3%) of configuration errors are introduced due to users' staying with default values incorrectly.*

To understand users' configuration problems in the real world, we categorize the user-reported issues into "difficulties," "errors," and "others." The *difficulties* refer to cases where users do not know *what* or *how* to configure to obtain their intended system functionalities or performance goals. The *errors* refer to erroneous settings that caused system misbehavior, such as crashes, hangs, or performance degradation. The users failed to reason out the misconfigurations as the root causes, and thus called support engineers or posted the problems on online forums and mailing lists. There are other configuration-related issues such as inquiries about general practices and internal usages, categorized as "others."

**Table 8**: **The distribution of the user-reported configuration issues across the categories.**

| Software | Difficulties | Errors | Others | Total |
|---|---|---|---|---|
| Storage-A | 17.3% (57) | 70.5% (232) | 12.2% (40) | 329 |
| Apache | 48.5% (47) | 44.3% (43) | 7.2% (7) | 97 |
| MySQL | 34.4% (33) | 47.9% (46) | 17.7% (17) | 96 |
| Hadoop | 35.7% (35) | 42.9% (42) | 21.4% (21) | 98 |

**Problem:** Two major data losses on a dozen machines. /*Hadoop*/
**Cause:** Stayed with the default values of the data-path parameters (e.g., dfs.name.dir, dfs.data.dir) which point to locations in /tmp. Thus, after the machines reboot, data losses occur.
"One of the common problems from users." (from Cloudera)

**Figure 5**: **A real-world example of configuration errors caused by users' incorrectly staying with default values.**

**Table 9**: **The number of error cases caused by the users' incorrectly staying with default parameter values, and their percentages among all the error cases. Most of these parameters were set by more than 5% of users. We exclude the cases which do not report users' settings.**

| Software | Total (#) | Incorrect defaults | Set by <5% users |
|---|---|---|---|
| Storage-A | 207 | 45.4% (94) | 3.2% |
| Apache | 40 | 17.5% (7) | 0.0% |
| MySQL | 45 | 53.3% (24) | 0.0% |
| Hadoop | 40 | 30.0% (12) | N/A |

Table 8 shows the distribution of the collected configuration issues across the categories. Remarkably, a significant percentage (17.3%∼48.5%) of issues fit into the "difficulties" category. This indicates users' tremendous difficulties to find the right knobs from the large configuration space. It is totally understandable, given the large quantity of parameters as well as the inefficiency of common navigation practices (discussed in § 5). Compared with open-source software, Storage-A has a lower percentage of "difficulties" issues (probably because Storage-A users are mostly professional administrators with better configuration experience). However, 17.3% still means a large financial cost, considering the human cost of configuration-related support calls [61, 75].

Similarly, "too many knobs" may prevent users from understanding the parameters thoroughly and tuning them carefully. Users tend to keep the settings (e.g., the default values) that work for the first run, instead of carefully, thoroughly examining the setting of every parameter. As a result, they may incorrectly miss parameters that need to be set according to the runtime environments, thus violating constraints of workloads, resources, cross-component correlations, etc. The consequence could be severe, such as failures and data losses. Fig. 5 gives an example where the user's incorrect staying with default values led to major data losses.

In fact, such cases (as the one in Fig. 5) are not rare. As shown in Table 9, a significant percentage (17.5%∼53.3%) of the configuration errors were caused by users' incorrectly staying with the default values, rather than setting wrong values.[1] We manually examined the users' settings reported in the issues (the cases without enough information about users' settings are excluded). Note that very few of these parameters are those rarely-set ones.

We cannot draw conclusions that smaller configuration space will definitely reduce the error-proneness of configuration activities, as the current datasets do not allow us to study the correlation between the size of the configuration space and the number (or rate) of configuration errors. However, as reducing the configuration space surely simplifies the configuration process, we believe it to have positive effect on the error-proneness of configuration.

[1] We acknowledge the possibility of users' intentionally setting the default values wrongly, but we believe that it is not the common case.

**Parameter:** optimizer_prune_level (Boolean) /*MySQL*/
**Desc.:** Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space.
**Values:** 0 or 1
**Usage:** No user set the parameter in our dataset.

(a) **Empirical, heuristic usages**

**Parameter:** key_cache_block_size (Numeric) /*MySQL*/
**Desc.:** The size in bytes of blocks in the key cache.
**Values:** [512, 16384]
**Usage:** All the users stay with the default value 1024 in our dataset.

(b) **Control internal data structures**

**Figure 6**: **Two examples of configuration parameters that are seldom set by any user in the MySQL dataset.**

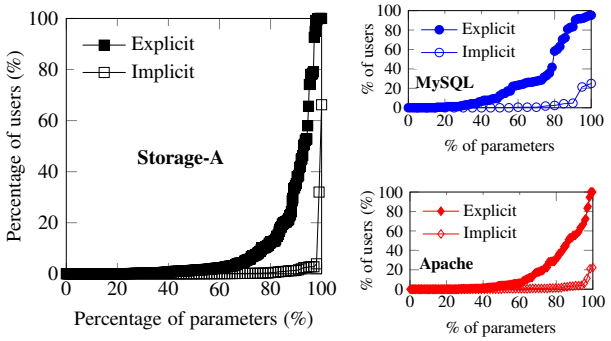


**Figure 7**: **Real-world usages of configuration parameters with explicit, visible external impact ("explicit") versus parameters specific to internal implementation ("internal").**

## 3.4 What Kinds of Knobs Are Most Utilized?

**Finding 4:** *Configuration parameters with explicit semantics, visible external impact are set by more users, in comparison to parameters that are specific to internal system implementation.* Thus, software developers should avoid exposing parameters specific to internal implementation. After all, users cannot, or may not have time to read the source code.

The distinct usages of different parameters drive us to think about the rationale behind how users set configuration parameters. We comparatively examine the parameters set by the majority of users and those seldom set. There is a remarkable difference between these two sets. Most of the frequently-set parameters have explicit semantics or visible external impact, e.g., enabling functionalities, switching between policies, enabling backup services. Thus, it is easy for users to understand and observe the effect of their settings.

On the contrary, many seldom-set parameters are *specific* to internal system implementation or protocol details (e.g., controlling data structures or library/system calls) and empirical/heuristic usages. Fig. 6 gives two examples of such knobs from MySQL.

Since most users have limited knowledge about system internals (even for the open-source ones), it is difficult for them to understand the semantics and potential impact of those internal parameters. As a result, most users do not have the confidence to touch these parameters, especially for system software running in production systems whose availability and performance are critical.

To validate our hypothesis, we manually annotate every studied parameter as "internal" or "explicit," based on whether or not it con-

**Table 10**: **Guidelines for simplifying configuration design.**

| Guideline | Support | Ref. |
|---|---|---|
| **1.** Hide or remove configuration parameters that are seldom set by any user. This requires to build user-feedback loops for configuration settings. | Finding 1(a) | § 3.1 |
| **2.** Promote parameters set by most users to be the "first-class" ones. Include them in tutorials and guidebooks to let users focus on these parameters first. | Finding 1(b) | § 3.1 |
| **3.** If possible, convert numeric parameters into enumerative or Boolean types with expressive, representative values to make the settings simple for users. | Finding 2(a) | § 3.2 |
| **4.** Avoid enumerative parameters with too many options. Five options should be sufficient in terms of user flexibility. | Finding 2(b) | § 3.2 |
| **5.** Only expose the configuration parameters with explicit semantics and/or visible external system impact. | Finding 4 | § 3.4 |

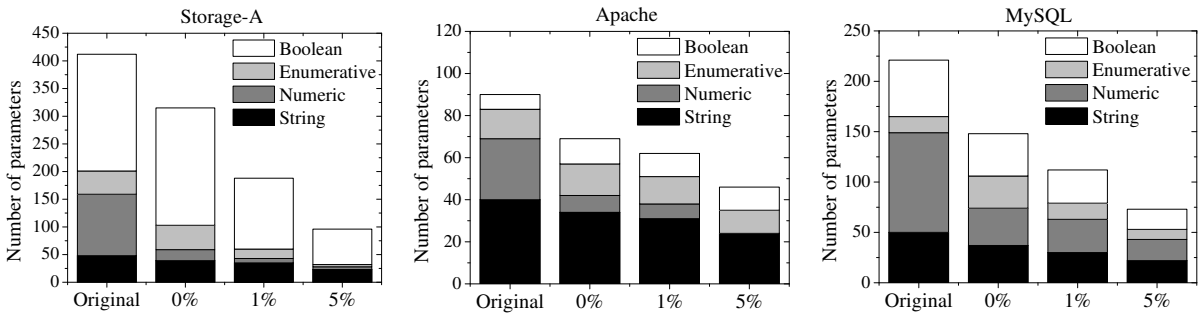


**Figure 8**: **How much can we simplify configuration? The number of configuration parameters and their data types before and after we apply the guidelines 1, 3, and 4 in Table 10, with fewer than 0%, 1%, and 5% of the existing users being impacted, respectively.**

trols internal implementation specific to the software. To minimize the subjectiveness during annotation, two inspectors separately labeled the parameters and compared the results with each other before consensus was reached. For some tough cases, we consulted with the developers to make decisions on the labeling. Fig. 7 shows the usages of "explicit" and "internal" parameters in the studied software. It confirms that the configuration parameters specific to internal implementation are seldom set by users.

Note that the usages of parameters are not strongly correlated with their data types. For example, Boolean parameters are usually more simple to set, compared with numeric parameters. However, if they are specific to internal implementation, they are still seldom set by users, as exemplified in Figure 6a.

# 4. CONFIGURATION SIMPLIFICATION

In this section, we study the opportunity and effectiveness of simplifying configuration by reducing the parameter and value space, as the fundamental approach to dealing with "too many knobs." We discuss other aspects of configuration simplification in § 6.

## 4.1 Simplification Guidelines

The findings in § 3 lead to a set of concrete, practical guidelines for simplifying configurations. Table 10 summarizes these guidelines, which mainly include the following two aspects:

- *Vertical:* Hiding or removing unnecessary configuration parameters and promoting the important ones (which are usually a small set), so that users can efficiently, correctly find the knobs.
- *Horizontal:* Reducing the value space of the parameters and providing meaningful, expressive options to help users set the parameters correctly and efficiently.

Note: These guidelines are only applicable to configuration design for the users (e.g., administrators) of the software, not for developers or support engineers. For example, hiding parameters should not prevent test engineers from finding or setting them. We discuss the implications of simplification to testing and debugging in § 6.

The proposed guidelines in Table 10 are general and do not consider system- or domain-specific information (which may provide opportunities to further simplify configuration). In this paper, we judge the necessity of a parameter based on its setting statistics. It is possible that even parameters set by many users can be eliminated, e.g., by automatically inferring or generating values from runtime environments (e.g., [18, 12, 17, 82]), formal models/specifications (e.g., [37, 51, 52, 60, 66]), historical settings (e.g., [36, 83]), etc.

Software vendors may raise the concern that simplifying configuration would hurt the advanced users who do need more flexibility compared with ordinary users; specially, some parameters are still under use even though by few users (e.g., 1%). In fact, this problem can be gracefully addressed by decoupling the advanced configuration from the basic ones. For example, the advanced parameters can be hidden from the common users and only be informed by requests; arbitrary parameter values are still allowed to be set if the user insists. Nevertheless, such advanced configuration should be introduced to users in separate manuals, templates, and files.

Please note that *we do not mean to prevent users from fine-tuning the configuration.* Instead, this paper advocates better design to facilitate users' configuration tuning by making it simple and less prone to errors. As we have demonstrated, the current design of configuration clearly does not consider users in the design process but assume that they need and can handle the level of complexity.

## 4.2 Effectiveness of Simplification

**Finding 5:** *The configuration of the studied software can be significantly simplified by reducing the configuration space both vertically and horizontally. For Storage-A, 51.9% of the original parameters can be hidden or removed, and 19.7% of the remaining ones can be further converted into simpler types, with the impact on fewer than 1% of the users. The similar reduction rates are also observed in the other two open-source software.*

We apply Guideline 1, 3, and 4 to the configuration of the studied software, allowing an impact on fewer than 0%, 1%, and 5% of the existing users, respectively. For Guideline 3, we convert a numeric parameter into an enumerative one if the parameter can be represented by no more than five options. Similarly, we convert an enumerative parameter into a Boolean if two options are sufficient to cover users' settings. We call the users being "impacted" by the simplification if their current settings would be changed to slightly different settings. Note: It does not necessarily mean that the new settings would result in failures or performance degradation.

Fig. 8 quantifies the effectiveness of the proposed configuration simplification methods. It shows the number of parameters and their data types after we apply the guidelines. As a first step in the direction of simplifying configuration, the results are promising, which also reflects the degree of the over-designed configuration.

# 5. CONFIGURATION NAVIGATION

To deal with too many knobs, many software projects rely on the navigation feature to help users find the right parameters and settings. In this section, we conduct measurement study to understand the effectiveness of the navigation methods using real-world cases.

As discussed in § 3.3, many users encounter difficulties in finding or setting configuration parameters. As shown Table 11, the majority of these "difficulties" cases are about finding the knobs rather than setting values. When a user knows which knob to set, it is relatively easy to find the information from the manual or using the Unix `man` command, and to learn how to set it.

## 5.1 Methodology

**Navigation Methods.** We study three navigation methods, keyword search, Google search, and NLP-based navigation.

- *Search by keywords:* Search by keywords on top of manuals is a pervasive navigation practice. Many software projects provide build-in search utilities tied into documentation (e.g., the search box in MySQL online docs [6]). Even without specific support, users can always rely on the search features offered by file readers/browsers to search keywords in PDF/HTML manuals.
- *Search on Internet:* Google search (or using other search engines such as Microsoft Bing) is another common practice to find the configuration knobs [15, 29]. Many software projects also provide search boxes that redirect users' queries to Google in their online manuals (e.g., Apache's online documentation [1]).
- *NLP-based navigation:* Recently, to help users find the right parameters, NLP-based navigation methods have been proposed [27, 4]. The idea is to build indexes for parameters based on their descriptions; users queries are matched to indexed contents and the best matched parameters are recommended to the users.

**Datasets and Queries.** We select out the "finding knobs" cases from the real-world cases studied in § 3.3. We exclude cases of Storage-A because the case reports were written by the company's support engineers, and thus do not contain users' original questions/queries. In most of the "finding knobs" cases, the users did find the target parameter(s) with the help of support engineers, or peer users from the online forums. We focus on these "closed" cases in our study and exclude the cases in which the target knobs do not exist. The number of the closed cases for Apache, MySQL, and Hadoop is 39, 25, and 26, respectively.

For each case, we use the original user-posted question as the original query. Every query is then filtered by common stop words which help remove meaningless words, such as interrogative words, personal pronouns, articles, etc. Then, we convert each word in a query string to the root forms of the word based on WordNets [38]. The final query strings are used for studying all the three methods. For example, the original question, "*How do I configure the proxy to forward all requests*" ends in the query, "*proxy forward request*."

**Table 11**: **User-reported "difficulties" cases in finding configuration knobs versus setting the values. The closed "finding knobs" cases (e.g., the target knobs exist) are used for studying navigation methods in § 5.**

| Software | Finding knobs | Setting knobs | Total |
|---|---|---|---|
| Storage-A | 82.5% (47) | 17.5% (10) | 57 |
| Apache | 89.4% (42) | 10.6% (5) | 47 |
| MySQL | 84.8% (28) | 15.2% (5) | 33 |
| Hadoop | 82.9% (29) | 17.1% (6) | 35 |

**Table 12**: **The percentage (number) of queries for which the keyword search returns pages containing the target parameter(s).**

| Software | % (#) of navigation cases | | Total Cases |
|---|---|---|---|
| | ∩{returned pages} | ∪{returned pages} | |
| Apache | 25.6% (10) | 79.5% (31) | 39 |
| MySQL | 24.0% (6) | 88.0% (22) | 25 |
| Hadoop | 15.4% (4) | 69.2% (18) | 26 |

**Table 13**: **The average number of returned pages per relevant page by keyword searching.**

| Software | Avg. number of returned pages | |
|---|---|---|
| | ∩{returned pages} | ∪{returned pages} |
| Apache | 2 | 32 |
| MySQL | 15 | 102 |
| Hadoop | 9 | 139 |

## 5.2 Effectiveness of Navigation

### 5.2.1 Search by Keywords

**Finding 6(a):** *Searching user manuals by keywords is not efficient to help users identify the target parameter(s).*

As a user's query often contains multiple keywords (such as "*proxy forward request*"), the user can first search for "*proxy*" and obtain all the pages containing it, and then search for "*forward*" and "*request*." Each keyword may be associated with multiple manual pages. We assume that a user can find the target configuration parameter(s) as long as she reads the page that contains the parameter (referred to as a *relevant page*).

We study two keyword-search strategies: (1) *union* (∪): returning all the pages contains at least one keyword, or (2) *intersection* (∩): only returning the pages that contain all the keywords.

As shown in Table 12, the intersection approach is not effective. It returns relevant pages for only 15.4%∼25.6% of the queries. The main reason is that the strategy is too *strict*. It *strictly* requires every keyword to appear on the manual pages of the parameters.

On the contrary, the union approach returns relevant pages for the majority of cases; however, it also returns many irrelevant pages. As shown in Table 13, the average number of returned pages per relevant page can be as large as one hundred (in these cases, the user's query contains certain "common" keywords). It is impractical to read through these many pages to find the target parameter(s).

Some commercial tools adopt the intersection method to provide navigation support on top of user manuals, e.g., Cloudera Manager [3] (a commercial tool for Hadoop administration). As discussed above, they are too strict and thus limited in finding knobs.

### 5.2.2 Google Search

We study Google search by sending the queries via Google search APIs. Then, we download the Web pages whose URLs are returned

**Table 14**: **Effectiveness of Google search. The percentage (number) of queries for which Google returns useful Web pages in top-five search results. "At the time" excludes pages posted after the original cases to emulate the situation when users encountered navigation issues.**

| Software | % (#) of queries w/ useful Web pages | | Total |
|---|---|---|---|
| | At the time | Postmortem | |
| Apache | 35.9% (14) | 74.4% (29) | 39 |
| MySQL | 40.0% (10) | 80.0% (20) | 25 |
| Hadoop | 26.9% (7) | 46.1% (12) | 26 |

**Table 15**: **Breakdowns of the sources that host the useful Web pages.**

| Source | Example | Percentage |
|---|---|---|
| Q&A forums | ServerFault.com | 37.4% |
| Blogs & articles | Articles on Blogger.com | 22.8% |
| Official docs | MySQL online docs | 22.8% |
| Third-party docs | Hortonwork's Hadoop docs | 8.1% |
| Docs of other SW | PHP's docs on MySQL conn. | 5.7% |
| Others | Wiki, Bugzilla | 3.2% |
| Mailing lists | Hadoop's mailing-list archive | 0.0% |

by Google. To make the searches explicit, we include the software name as a part of the query keywords. We analyze the top-5 Web pages returned by Google and examine if they are useful (existing studies show that the top-5 results attract most clicks [53, 54]). A *useful Web page* must meet the following two criteria: (1) containing the target parameter(s) (the *recall* metric); and (2) containing no more than five other parameters (the *precision* metric).

**Finding 6(b):** *Google search can provide useful information for 46.1%∼80.0% of the historical configuration navigation issues. However, it is less efficient in navigation parameters of less popular software or new issues. The majority of resources on the Web that host useful information for navigation are the contents contributed by users, such as Q&A forums and blog articles.*

Table 14 shows the effectiveness of Google search for configuration navigation. Google returns useful pages in the top-5 results for 46.1%∼80.0% of the queries. In other words, if these *historical* navigation issues are encountered by new users, 46.1%∼80.0% of them could be resolved by Google search. Hadoop has a remarkably low number (more than half of the historical cases cannot resolved by Google), mainly for two reasons. First, Hadoop has a much smaller user base with less online resources, compared with Apache and MySQL. Second, the primary Q&A sites of Hadoop is its official user mailing list. However, mailing list archives have very low page ranks, making them less "visible" by Google search.

Also, we emulate the situation when the user encountered a navigation issue and searched Google, by excluding Web pages posted after the original user question. As shown in Table 14, no more than 40% of these issues can be resolved by Google. Many of the returned Web pages "*at the time*" are online manuals and tutorials that include too many configuration information, which has similar efficiency as searching keywords on top of manuals (c.f., § 5.2.1).

To understand what types of Web pages are useful for configuration navigation, we classify the useful Web pages returned by Google based on their types, as shown in Table 15. Remarkably, user-generated pages contribute to more than 50% of these useful pages, such as Q&A posts, blogs articles. These Web pages usually record the users' experience and solutions to specific configuration problems, and only contains a small set of relevant parameters, which is more useful for navigation (compared with online manual pages that list parameters one by one). Therefore, methods to leverage user-generated contents, especially those with low page ranks (e.g., mailing-list archives) is desired for configuration navigation.

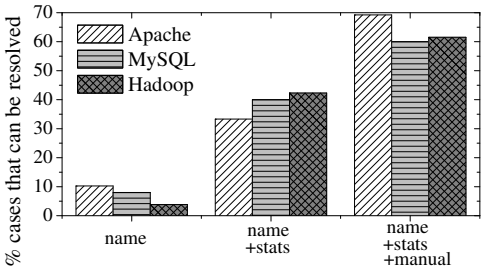


**Figure 9**: **The performance of NLP-based navigation using different information sources. We consider a case can be resolved if the target parameter can be returned in the top-5 navigation results.**

### 5.2.3 NLP-based Navigation

To help users get the right configuration knob (preference), NLP-based navigation methods have been proposed (e.g., PrefFinder [28] and Cox [4]). These NLP-based navigation methods take a user's query in natural languages as the input, and return the configuration parameters relevant to the query. This is achieved by parsing, analyzing, and indexing the information of every parameter (e.g., from its manual entries) using NLP techniques, such as stemming, stop-word filtering, text normalization, synonym expansion, etc. To return the relevant parameters, the navigation engine ranks the parameters by *scoring* how well they match the query.

We study the efficacy of NLP-based navigation based on Cox (PrefFinder is not open sourced). Cox is a configuration navigation library on top of Lucene [11]. It provides utilities to extract the texts for every configuration parameter by breaking manual pages, and allows us to change the parsing, analyzing, and scoring methods.

In addition to indexing and matching, we also take users' configuration statistics into account. The original match score is *boosted* by the *popularity* of the parameter, defined as the percentage of users who set the parameter among all the users (same as in § 3.1). The idea is to boost popular parameters with higher ranks if they match users' queries. Also, we assign a lower *scale* 0.4 to contents from manual entries while parameter names have scale 1.0.

**Finding 6(c):** *Well-engineered NLP-based navigation can return the target configuration parameter for more than 60% of the historical navigation issues. Boosting the results with the statistics of users' configuration settings in the field can significantly improve the performance of NLP-based navigation.*

Figure 9 shows the results of the NLP-based navigation using different combination of information sources (parameter names, statistics of users' settings, and manual pages). We observe that the navigation only based on parameter names has poor performance (worse than the dataset in [28, 26]). The reason is that in our dataset of system software, many of the queries do not contains any keyword appearing in the parameter name, which is different from queries to desktop software configuration [26]. In desktop software, users are usually able to specify useful keywords like "*color*," "*tab*," "*cache*" (which are parts of the target parameter's name), while the queries here are higher level intentions such as "*speedup insert performance*" (target parameter: `max_heap_table_size` and there are more than 80 size-related parameters in all). In this case, applying users' configuration statistics brings significant performance improvement, because *a parameter that is used by many users are likely to be needed by the current user.* In addition, the results show that leveraging contents from manual entries are useful: manual entries bring additional information about the parameter.

Overall, our NLP-based navigation implementation resolves more than 60% of the real-world cases. We manually examine the unresolved queries and find most of them indeed miss the keywords in

the contents of the target parameter. First, some queries are vague or misleading (even for human experts). For example, alias-related parameters are returned for the query, "*alias url without use host*," but the target one is related to virtual hosts. Second, some queries require domain-specific knowledge beyond the information base. For example, it fails to associate SSL with "encrypt," and fails to return SSL-related parameters for "*encrypt network channel*." This limitation can potentially be addressed by using word-cluster based techniques to capture "concepts" instead of "keywords."

# 6. DISCUSSION AND FUTURE WORK

## 6.1 Implications and Incentives

Reducing configuration space and simplifying configuration not only help users' configuration difficulties and problems, but also would bring tremendous benefits for software vendors by relieving their burden of testing, error detection and troubleshooting.

Testing software with large configuration space is extremely challenging. The number of possible configuration settings is an exponential function of the number of parameters and their value space, which makes it infeasible to test exhaustively. This is known as *configuration space explosion* [74]. To address this problem, a series of pioneer works have been proposed, including pruning the configuration space [33, 49, 55, 56], selecting typical configuration values [43, 19, 22, 24, 74], prioritizing certain important configurations [23, 58, 43], and reducing the number of test cases [42].

As shown in our study, many configuration parameters are not in real use, i.e., a large portion of the existing configuration space is not touched by users. Removing the unused configuration can significantly relieve the burden of software testing in the context of configuration space explosion, as complementary to the existing testing methods. Prioritization becomes natural, considering users' configuration statistics —the parameters/values set by more users should have higher popularity than the ones set by fewer users.

Smaller configuration space also benefits users and support engineers for misconfiguration detection and troubleshooting. Small configuration space comes with small error space, which not only makes it easy for users to examine and find the errors, but also make the automatic detection and troubleshooting procedures more efficient. Also, with smaller error space, the detection and troubleshooting tools can be more focused and targeted. Most importantly, with simplified configuration, users are likely to have less configuration difficulties and problems, which in turn results in lower support cost for software vendors.

With these incentives, we advocate software vendors to take actions in simplifying existing configuration and providing new configurations more cautiously with user-centric design philosophy.

## 6.2 Further Simplification

In this paper, we have mainly investigated the feasibility and opportunity of simplifying configuration in the aspect of reducing the configuration space (including both the parameter space and the value space). However, it is important to note that the configuration space is not equivalent to the entire configuration complexity. In other word, the efforts to simplifying configuration should not be limited to reducing the configuration space (which is only our first step towards addressing this problem).

Besides the large configuration space, other known root causes of configuration complexity include ambiguity and inconsistency of configuration semantics [20], dependencies among multiple parameters and multiple software components [47, 34, 70, 79], and poor system guidance/feedback [25, 72]. It remains as our future work to understand and address these aspects of configuration complexity perceived by users, with the goal of making configuration simple, efficient, and less prone to errors. Similar as the study in this paper, we believe the key towards addressing these problems is to follow the user-centric philosophy —to understand users' difficulties and problems in the field and to design configuration from the users' perspectives. After all, configuration is one type of user interface that are supposed to be operated by users.

# 7. RELATED WORK

Since the previous studies [16, 31, 40, 46, 75] revealed the prevalence and severity of configuration issues in different types of software systems, many recent research efforts have been made to attack configuration problems. As discussed in § 6.1, many efforts have been made to detect and troubleshoot configuration errors in users' configuration files [13, 14, 28, 44, 65, 67, 69, 70, 71, 77, 78, 79, 80, 81]. To harden systems against misconfiguration, previous studies have looked into configuration testing, including techniques to reduce the testing space [19, 23, 22, 24, 33, 42, 43, 50, 58, 74, 49, 55, 56], as well as tool support to generate constraint-guided test cases to expose system vulnerabilities to misconfiguration [30, 72]. Xu and Zhou [73] provide board overviews of these approaches.

While the aforementioned studies significantly help the situation of today's configuration problems in terms of testing, detection, and diagnosis, a probably more fundamental direction is rethinking and redesigning configuration to avoid users' configuration difficulties and errors in the first place. This paper is motivated exactly along this line by focusing on understanding the key questions of configuration design from the users' perspectives.

Previous work has studied the characteristics of configuration errors [75]. We also examine 620 real-world configuration issues for the purpose of understanding the consequence of too many knobs; however, the main focus of this paper is not to study the errors but to understand how users configure their systems: Do they need so many parameters? What kind of parameters do they use? What are their difficulties in configuring their systems?

Our work is fundamentally different from previous studies on end-user desktop or mobile software [57, 62]. The configuration of system software, as the focus of this paper, has high availability and reliability requirements, and is mainly on top of file/command interfaces, while end-user software configuration is inclined to preference and personalization based on GUIs. Also, the users of system software are usually technical inclined, which is different from ordinary end users [15, 64]. For example, configuration navigation for system software has different characteristics and may need additional practices, compared with end-user software (c.f., § 5.2.3).

# 8. CONCLUSION

The configuration of system software has become increasingly complex. To advocate cautious and disciplined thinking in configuration design, this paper has provided, perhaps for the first time, quantitative evidence for the over-delivered (or under-exploited) flexibility represented by configuration parameters. By studying the large-scale configuration settings of real users, we have revealed a number of findings, leading to a few guidelines for simplifying configuration. We also studied configuration navigation as an intermediate solution, if the simplification process takes time.

We hope that our work can inspire developers to design system configuration with the user-centric design philosophy, and carefully balance simplicity (usability) and flexibility (configurability). Similar to UI/UX design, it is important for developers to collect users' feedback and think from the users' perspective, before introducing yet another knob. Feedback loops should be initiated and followed to help developers improve the usability of their software systems.

## 9. REPLICATION PACKAGE

The datasets of the open-source software projects (including both the configuration files and the configuration issues), as well as the navigation implementation based on Cox have been successfully evaluated by the Replication Packages Evaluation Committee and found to meet expectations. They are publicly available at:

`https://github.com/tianyin/configuration_datasets`
`https://github.com/tianyin/cox`

## 10. REFERENCES


[1] Apache HTTP Server Version 2.4 Documentation. `http://httpd.apache.org/docs/2.4/`.

[2] Apache HTTP Server Version 2.4 Documentation (LogLevel Directive). `http://httpd.apache.org/docs/2.4/mod/core.html#loglevel`.

[3] Cloudera Manager. `http://www.cloudera.com/content/cloudera/en/products-and-services/cloudera-enterprise/cloudera-manager.html`.

[4] Cox: A configuration navigation tool and library for a thousand of knobs. `https://github.com/tianyin/cox`.

[5] Database Administrators. `http://dba.stackexchange.com/`.

[6] MySQL 5.6 Reference Manual (Online Version). `http://dev.mysql.com/doc/refman/5.6/en/index.html`.

[7] MySQL 5.6 Reference Manual (PDF Version). `http://downloads.mysql.com/docs/refman-5.6-en.pdf`.

[8] Pro Webmasters. `http://webmasters.stackexchange.com/`.

[9] ServerFault. `http://serverfault.com/`.

[10] StackOverflow. `http://stackoverflow.com/`.

[11] The Apache Lucene Project. `https://lucene.apache.org/`.

[12] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running Circles Around Storage Administration. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*, Berkeley, CA, USA, January 2002.

[13] M. Attariyan, M. Chow, and J. Flinn. X-ray: Automating Root-Cause Diagnosis of Performance Anomalies in Production Software. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12)*, Hollywood, CA, USA, October 2012.

[14] M. Attariyan and J. Flinn. Automating Configuration Troubleshooting with Dynamic Information Flow Analysis. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, Vancouver, BC, Canada, October 2010.

[15] R. Barrett, E. Kandogan, P. P. Maglio, E. Haber, L. A. Takayama, and M. Prabaker. Field Studies of Computer System Administrators: Analysis of System Management Tools and Practices. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW'04)*, Chicago, Illinois, USA, November 2004.

[16] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-scale Machines*. Morgan and Claypool Publishers, 2009.

[17] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[18] S. Duan, V. Thummala, and S. Babu. Tuning Database Configuration Parameters with iTuned. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB'09)*, Lyon, France, August 2009.

[19] E. Dumlu, C. Yilmaz, M. B. Cohen, and A. Porter. Feedback driven adaptive combinatorial testing. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'11)*, Toronto, ON, Canada, July 2011.

[20] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In *Proceedings of the 5th ACM Symposium on Cloud Computing (SoCC'14)*, Seattle, WA, USA, November 2014.

[21] E. M. Haber and J. Bailey. Design Guidelines for System Administration Tools Developed through Ethnographic Field Study. In *Proceedings of the 2007 ACM Conference on Human Interfaces to the Management of Information Technology (CHIMIT'07)*, Cambridge, MA, USA, March 2007.

[22] C. Henard, M. Papadakis, M. Harman, and Y. L. Traon. Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. In *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, Firenze, Italy, May 2015.

[23] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Traon. Bypassing the Combinatorial Explosion: Using Similarity to Generate and Prioritize T-Wise Test Configurations for Software Product Lines. *IEEE Transactions on Software Engineering (TSE)*, 40(7):650–670, July 2014.

[24] A. Hervieu, B. Baudry, and A. Gotlieb. PACOGEN : Automatic Generation of Pairwise Test Configurations from Feature Models. In *Proceedings of the 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE'11)*, Hiroshima, Japan, November 2011.

[25] A. Hubaux, Y. Xiong, and K. Czarnecki. A User Survey of Configuration Challenges in Linux and eCos. In *Proceedings of 6th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12)*, Leipzig, Germany, January 2012.

[26] D. Jin, M. B. Cohen, X. Qu, and B. Robinson. PrefFinder: Getting the Right Preference in Configurable Software Systems (Supplementary Data). `http://cse.unl.edu/~myra/artifacts/PrefFinder_2014/`.

[27] D. Jin, M. B. Cohen, X. Qu, and B. Robinson. PrefFinder: Getting the Right Preference in Configurable Software Systems. In *Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering (ASE'14)*, Västerås, Sweden, September 2014.

[28] D. Jin, X. Qu, M. B. Cohen, and B. Robinson. Configurations Everywhere: Implications for Testing and Debugging in Practice. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, Hyderabad, India, June 2014.

[29] E. Kandogan and E. M. Haber. Security Administration Tools and Practices. *Security and Usability, O'Reilly Media, Inc.*, August 2005.

[30] L. Keller, P. Upadhyaya, and G. Candea. ConfErr: A Tool for Assessing Resilience to Human Configuration Errors. In

*Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'08)*, Anchorage, Alaska, USA, June 2008.

[31] S. Kendrick. What Takes Us Down? *USENIX ;login:*, 37(5):37–45, October 2012.

[32] E. Kiciman and Y.-M. Wang. Discovering Correctness Constraints for Self-Management of System Configuration. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC'04)*, New York, NY, USA, May 2004.

[33] C. H. P. Kim, D. Marinov, S. Khurshid, and D. Batory. SPLat: Lightweight Dynamic Analysis for Reducing Combinatorics in Testing Configurable Systems. In *Proceedings of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13)*, Saint Petersburg, Russia, August 2013.

[34] M. Larsson and I. Crnkovic. Configuration Management for Component-based Systems. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*, Toronto, Ontario, Canada, May 2001.

[35] L. Y. Liang. Linkedin.com inaccessible on Thursday because of server misconfiguration. 2013. `http://www.straitstimes.com/breaking-news/singapore/story/linkedincom-inaccessible-thursday-because-server-misconfiguration-2013`.

[36] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang. Improving Trace Accuracy through Data-Driven Configuration and Composition of Tracing Features. In *Proceedings of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13)*, Saint Petersburg, Russia, August 2013.

[37] R. Michel, A. Hubaux, V. Ganesh, and P. Heymans. An SMT-based Approach to Automated Configuration. In *Proceedings of the 10th International Workshop on Satisfiability Modulo Theories (SMT'12)*, Manchester, UK, June 2012.

[38] G. A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, November 1995.

[39] S. Nadi, T. Berger, C. Kästner, and K. Czarnecki. Mining Configuration Constraints: Static Analyses and Empirical Results. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, Hyderabad, India, June 2014.

[40] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why Do Internet Services Fail, and What Can Be Done About It? In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, Seattle, WA, USA, March 2003.

[41] C. Perrow. *Normal Accidents: Living with High-Risk Technologies*. Basic Books, 1984.

[42] X. Qu, M. Acharya, and B. Robinson. Impact Analysis of Configuration Changes for Test Case Selection. In *Proceedings of the 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE'11)*, Hiroshima, Japan, November 2011.

[43] X. Qu, M. B. Cohen, and G. Rothermel. Configuration-Aware Regression Testing: An Empirical Study of Sampling and Prioritization. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'08)*, Seattle, WA, USA, July 2008.

[44] A. Rabkin and R. Katz. Precomputing Possible Configuration Error Diagnosis. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11)*, Lawrence, KS, USA, November 2011.

[45] A. Rabkin and R. Katz. Static Extraction of Program Configuration Options. In *Proceedings of the 33th International Conference on Software Engineering (ICSE'11)*, Honolulu, Hawaii, USA, May 2011.

[46] A. Rabkin and R. Katz. How Hadoop Clusters Break. *IEEE Software Magazine*, 30(4):88–94, July 2013.

[47] V. Ramachandran, M. Gupta, M. Sethi, and S. R. Chowdhury. Determining Configuration Parameter Dependencies via Analysis of Configuration Data from Multi-tiered Enterprise Applications. In *Proceedings of the 6th International Conference on Autonomic Computing and Communications (ICAC'09)*, Barcelona, Spain, June 2009.

[48] J. Reason. *Human Error*. Cambridge University Press, October 1990.

[49] E. Reisner, C. Song, K.-K. Ma, J. S. Foster, and A. Porter. Using Symbolic Evaluation to Understand Behavior in Configurable Software Systems. In *Proceedings of the 32th International Conference on Software Engineering (ICSE'10)*, Cape Town, South Africa, May 2010.

[50] B. Robinson and L. White. Testing of User-Configurable Software Systems Using Firewalls. In *Proceedings of the 19th IEEE International Symposium on Software Reliability Engineering (ISSRE'08)*, Seattle/Redmond, WA, USA, November 2008.

[51] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable Product Line Configuration: A Straw to Break the Camel's Back. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE'13)*, November 2013.

[52] A. S. Sayyad, T. Menzies, and H. Ammar. On the Value of User Preferences in Search-Based Software Engineering: A Case Study in Software Product Lines. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, San Francisco, CA, USA, May 2013.

[53] Search Engine Watch. How Much is a Google Top Spot Worth? 2010. `http://searchenginewatch.com/article/2050861/How-Much-is-a-Google-Top-Spot-Worth`.

[54] Search Engine Watch. 53% of Organic Search Clicks Go to First Link. 2012. `http://searchenginewatch.com/article/2050861/How-Much-is-a-Google-Top-Spot-Worth`.

[55] C. Song, A. Porter, and J. S. Foster. iTree: Efficiently Discovering High-Coverage Configuration Using Interaction Trees. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, Zurich, Switzerland, June 2012.

[56] C. Song, A. Porter, and J. S. Foster. iTree: Efficiently Discovering High-Coverage Configuration Using Interaction Trees. *IEEE Transactions on Software Engineering (TSE)*, 40(3):251–265, March 2014.

[57] J. Spool. Do users change their settings? 2011. `http://www.uie.com/brainsparks/2011/09/14/do-users-change-their-settings/`.

[58] H. Srikanth, M. B. Cohen, and X. Qu. Reducing Field

Failures in System Configurable Software: Cost-Based Prioritization. In *Proceedings of the 20th IEEE International Symposium on Software Reliability Engineering (ISSRE'09)*, Mysuru, Karnataka, India, November 2009.

[59] Y. Sverdlik. Microsoft: Misconfigured Network Device Led to Azure Outage. http://www.datacenterdynamics.com/focus/archive/2012/07/microsoft-misconfigured-network-device-led-azure-outage, 2012.

[60] G. Tamura, R. Casallas, A. Cleve, and L. Duchien. QoS Contract Preservation through Dynamic Reconfiguration: A Formal Semantics Approach. *Science of Computer Programming*, 94(3):301–332, November 2014.

[61] The Association of Support Professionals. Technical Support Cost Ratios. http://www.asponline.com/tscr.pdf, 2000.

[62] The Standish Group. Modernization: Clearing a Pathway to Success. 2010. https://www.standishgroup.com/sample_research_files/Modernization.pdf.

[63] K. Thomas. Thanks, Amazon: The Cloud Crash Reveals Your Importance. 2002. http://www.pcworld.com/article/226033/thanks_amazon_for_making_possible_much_of_the_internet.html.

[64] N. F. Velasquez, S. Weisband, and A. Durcikova. Designing Tools for System Administrators: An Empirical Test of the Integrated User Satisfaction Model. In *Proceedings of the 22nd Large Installation System Administration Conference (LISA'08)*, San Diego, CA, USA, November 2008.

[65] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic Misconfiguration Troubleshooting with PeerPressure. In *Proceedings of the 6th USENIX Conference on Operating Systems Design and Implementation (OSDI'04)*, San Francisco, California, USA, December 2004.

[66] T. Wang, M. Harman, Y. Jia, and J. Krinke. Searching for Better Configurations: A Rigorous Approach to Clone Evaluation. In *Proceedings of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13)*, Saint Petersburg, Russia, August 2013.

[67] Y.-M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang. STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support. In *Proceedings of the 17th Large Installation Systems Administration Conference (LISA'03)*, San Diego, CA, USA, October 2003.

[68] M. Welsh. What I Wish Systems Researchers Would Work On. 2013. http://matt-welsh.blogspot.com/2013/05/what-i-wish-systems-researchers-would.html.

[69] A. Whitaker, R. S. Cox, and S. D. Gribble. Configuration Debugging as Search: Finding the Needle in the Haystack. In *Proceedings of the 6th USENIX Conference on Operating Systems Design and Implementation (OSDI'04)*, San Francisco, California, USA, December 2004.

[70] Y. Xiong, A. Hubaux, S. She, and K. Czarnecki. Generating Range Fixes for Software Configuration. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, Zurich, Switzerland, June 2012.

[71] Y. Xiong, H. Zhang, A. Hubaux, S. She, J. Wang, and K. Czarnecki. Range Fixes: Interactive Error Resolution for Software Configuration. *IEEE Transactions on Software Engineering (TSE)*, December 2014.

[72] T. Xu, J. Zhang, P. Huang, J. Zheng, T. Sheng, D. Yuan, Y. Zhou, and S. Pasupathy. Do Not Blame Users for Misconfigurations. In *Proceedings of the 24th Symposium on Operating System Principles (SOSP'13)*, Farmington, PA, USA, November 2013.

[73] T. Xu and Y. Zhou. Systems Approaches to Tackling Configuration Errors: A Survey. *ACM Computing Surveys (CSUR)*, 47(4), July 2015.

[74] C. Yilmaz, M. B. Cohen, and A. A. Porter. Covering Arrays for Efficient Fault Characterization in Complex Configuration Spaces. *IEEE Transactions on Software Engineering (TSE)*, 32(1):1–15, January 2006.

[75] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy. An Empirical Study on Configuration Errors in Commercial and Open Source Systems. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*, Cascais, Portugal, October 2011.

[76] D. Yuan, S. Park, and Y. Zhou. Characterizing Logging Practices in Open-Source Software. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, Zurich, Switzerland, June 2012.

[77] D. Yuan, Y. Xie, R. Panigrahy, J. Yang, C. Verbowski, and A. Kumar. Context-based Online Configuration Error Detection. In *Proceedings of 2011 USENIX Annual Technical Conference*, Portland, OR, USA, June 2011.

[78] A. Zeller. *Why Programs Fail: A Guide to Systematic Debugging (2nd Edition)*. Morgan Kaufmann Publishers, June 2009.

[79] J. Zhang, L. Renganarayana, X. Zhang, N. Ge, V. Bala, T. Xu, and Y. Zhou. EnCore: Exploiting System Environment and Correlation Information for Misconfiguration Detection. In *Proceedings of the 19th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS-XIX)*, Salt Lake City, UT, USA, March 2014.

[80] S. Zhang and M. D. Ernst. Automated Diagnosis of Software Configuration Errors. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, San Francisco, CA, USA, May 2013.

[81] S. Zhang and M. D. Ernst. Which Configuration Option Should I Change? In *Proceedings of the 36th Internationl Conference on Software Engineering (ICSE'14)*, Hyderabad, India, May 2014.

[82] W. Zheng, R. Bianchini, and T. D. Nguyen. Automatic Configuration of Internet Services. In *Proceedings of the 2nd EuroSys Conference (EuroSys'07)*, Lisbon, Portugal, March 2007.

[83] W. Zheng, R. Bianchini, and T. D. Nguyen. MassConf: Automatic Configuration Tuning By Leveraging User Community Information. In *Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering (ICPE'11)*, Karlsruhe, Germany, March 2011.