PracExtractor: Extracting Configuration Good Practices from Manuals to Detect Server Misconfigurations

Chengcheng Xiang
University of California, San Diego

Haochen Huang
University of California, San Diego

Andrew Yoo
University of Illinois Urbana-Champaign

Yuanyuan Zhou
University of California, San Diego

Shankar Pasupathy
NetApp Inc.

Abstract

Configuration has become ever so complex and error-prone in today's server software. To mitigate this problem, software vendors provide user manuals to guide system admins on configuring their systems. Usually, manuals describe not only the meaning of configuration parameters but also *good practice recommendations* on how to configure certain parameters. Unfortunately, manuals usually also have a large number of pages, which are time-consuming for humans to read and understand. Therefore, system admins often do not refer to manuals but rely on their own guesswork or unreliable sources when setting up systems, which can lead to configuration errors and system failures.

To understand the characteristics of configuration recommendations in user manuals, this paper first collected and studied 261 recommendations from the manuals of six large open-source systems. Our study shows that 60% of the studied recommendations describe specific and checkable specifications instead of merely general guidance. Moreover, almost all (97%) of such specifications have not been checked in the systems' source code, and 61% of them are not equivalent to the default settings. This implies that additional checking is needed to ensure the recommendations are correctly applied.

Based on our characteristic study, we build a tool called PracExtractor, which employs Natural Language Processing (NLP) techniques to automatically extract configuration recommendations from software manuals, converts them into specifications, and then uses the generated specifications to detect violations in system admins' configuration settings. We evaluate PracExtractor with twelve widely-deployed software systems, including one large commercial system from a public company. In total, PracExtractor automatically extracts 338 recommendations and generates 173 specifications with reasonable accuracy. With these generated specifications, PracExtractor detects 1423 good practice violations from open-source docker images. To this day, we have reported 325 violations and have got 47 of them confirmed as real configuration issues by admins from different organizations.

1 Introduction

1.1 Motivation

Misconfiguration (error in configuration settings) has become one of the major causes of failures in large-scale cloud and Internet systems, as reported by many system vendors [29,45,64] and service providers [20,26,32,34,37,51]. While various fault tolerance and recovery mechanisms are effective in handling hardware and software failures, they are less effective in handling configuration errors [27,32,37]. In 2017, a configuration error at Level 3, an Internet backbone company, caused a nationwide network outage [22]. On March 13th, 2019, the recent outage in Facebook was also caused by a server configuration error, affecting millions of users [52]. In addition to reliability, configuration errors can also lead to security issues [59]. OWASP reports misconfiguration as one of the top 10 most critical web security risks [38]. In 2017, a configuration error of Amazon S3 storage exposed personal information of 200 million U.S. voters [53].

One of the primary reasons for configuration errors is the ever-increasing configuration complexity, especially with system software [60]. Configuration complexity is partially reflected by the large and almost always increasing number of configuration parameters, as well as their configuration constraints and inter-dependency [31, 35, 44, 47, 63], which inevitably increase system admins' error rates [40, 46]. For example, MySQL 8.0 has more than 460 configuration parameters. Similarly, Apache httpd 2.4 has more than 550 parameters. Such a high level of complexity makes system configuration an error-prone task.

While research efforts have been attempted on reducing configuration complexity [55, 60], it is still a long journey to fully tame the complexity issue. Today, to assist system admins, software vendors typically release user manuals together with their software. A manual describes in detail the name, usages and sometimes constraints of each configuration parameter. It can be in print as a PDF file or accessed electronically as HTML/XML files, providing good guidance

Software	Pages	Software	Pages
COMP-A[1]	8283	Httpd	1009
MySQL	5494	HBase	787
PostgreSQL	3724	Freebsd	726
CentOS	2297	Ubuntu server	413
Hadoop	2331	Zookeeper	181

Table 1: Number of pages in ten popular software's manuals.

and reference for system admins to configure and manage server software.

Unfortunately, system manuals are quite large, containing hundreds or even thousands of pages. Table 1 lists the numbers of pages in the manuals of ten software, including one commercial software, COMP-A [1]. from a large public company. As the table shows, manuals of MySQL, PostgreSQL, CentOS and Hadoop have 2331-5494 pages. COMP-A has 8283 pages in its technical documentation.

With such a daunting number of manual pages, system admins find manuals hard and time-consuming to refer to and understand. As such, system admins often do not refer to them when configuring systems. Instead, they either rely on their own judgment/guesswork or ask for help from other admins [36]. Previous studies have shown that system admins solved only a small proportion of usage problems (4% to 25%) by referring to manuals [21, 33, 36].

However, manuals still contain useful information and ignoring manuals can lead to configuration errors that cause server downtime and data center outages. Figure 1 gives six real-world configuration errors of commercial and open-source software, in which system admins clearly do not follow good practice recommendations in manuals. The misconfigured parameters in these examples were set to incorrect values, leading to problems of systems' availability, performance and security. Since these incorrect values are totally legal values (i.e. violating no constraints in source code), they cannot be detected by software's own checking logic, as well as tools that focus on checking for illegal values [63]. However, in all these cases, the corresponding manuals actually have clearly given recommendations on how to set these parameters. Had these recommendations been followed by system admins, these misconfigurations would have been avoided.

Unfortunately, good practices recommended in manuals or other documents are not fully utilized by system admins to avoid configuration errors mainly due to three reasons:

- Recommended practices are spread out in various parts of manuals and cannot be easily found by system admins due to manuals' bulkiness and poor navigation [36].
- Many good practice recommendations are not always the same as default settings (more details in §2). A recent study shows that admins tend to go with default settings for more than 80% of configuration parameters, and many configuration errors were caused exactly because admins do not change the default setting [60]. As later shown in our evaluation (cf. Table 12), we also found many (997) cases that system admins just went with bad defaults. Had system admins read the recommendations in the manuals, they could have avoided some of these mistakes.
- As shown in all the examples in Figure 1, good practices recommended in manuals are often *soft* constraints, which usually are not checked inside software. Thereby, the violations of them cannot be detected by previous tools that were built by either inferring configuration specification from the software's source code [63] or just directly reusing the source code to check configuration [61].

[1]We are required to keep the company and the product anonymous.

1.2 Our Contributions

This paper studies the research questions on whether it is useful to automatically extract good practice recommendations from manuals and use them to detect system admins' configuration issues, and if so, how to do it. We first collected and studied 261 recommendations from six large open-source software manuals. Our study shows that 60% of the studied recommendations described specific, checkable specifications instead of just general guidance. In addition, almost all (97%) of the checkable specifications are not checked in source codes, and 61% of them are different from the default settings (reasons and details are discussed in §2).

Based on our characteristic study, we build a tool called PracExtractor, which employs Natural Language Processing (NLP) techniques to automatically extract good practice recommendations from manuals, converts them into specifications, and then uses the generated specifications to detect violations in system admins' configurations.

We evaluated PracExtractor with manuals of *twelve* widely-deployed software systems, including one from a *commercial* company with tens of thousands of customers. Overall, PracExtractor automatically extracts 338 recommendations, with a precision of 86% and a recall of 83%. PracExtractor converts 173 recommendations into specifications with reasonable accuracy. For the six "new" manuals not included in our characteristic study, PracExtractor can achieve a precision of 83% for recommendations and 88% for specifications.

To evaluate the capability of detecting real-world misconfigurations, we run PracExtractor against real-world configurations from top-downloaded container images on DockerHub [24]. PracExtractor detects 1423 violations in 853 images. We reported 325 violations to the image maintainers and got 47 confirmed as real configuration issues, including six issues in images with over 1M downloads and 28 in images with over 1K downloads.

Interestingly, in addition to detecting system admins' configuration problems, PracExtractor also detects a few incorrect

Comp-A

Parameter: *export-policy*
Incorrect setting by admins: *none*
Outcome: NFS mount permission deny
Fix: Specify a rule for export-policy

Recommendation in manual:
"Always *create a rule* with a policy to control access to a volume."

(a)

Comp-A

Parameter: *security-style*
Incorrect setting by admins: *mix*
Outcome: NFS volumes are not mounted
Fix: change *security-style* to *unix*

Recommendation in manual:
"Choose either NTFS- or UNIX-style security unless there is a specific reason to use mix."

(b)

Comp-A

Parameter: *rsize*
Incorrect setting by admins: *32k*
Outcome: Performance slowdown
Fix: change *rsize* to *64k*

Recommendation in manual:
"Comp-A highly recommends using *64k* rsize/wsize for better performance with NFS v4."

(c)

Httpd

Parameter: *EnableSendfile*
Incorrect setting by admins: *on*
Outcome: web pages become unavailable
Fix: change the value to *off*

Recommendation in manual:
"Within some filesystems, it is better to *disable* this feature to avoid operational problems."

(d)

HBase

Parameter: *hbase.regionserver.thrift.framed*
Incorrect setting by admins: *false*
Outcome: vulnerable to DoS attack
Fix: change the value to *true*

Recommendation in manual:
"Setting this to false will select the default transport, vulnerable to DoS..."

(e)

Cassandra

Parameter: *enable_transient_replication*
Incorrect setting by admins: *true*
Outcome: potential service failure
Fix: change the value to *false*

Recommendation in manual:
"Transient replication is experimental and is not recommended for production use."

(f)

Figure 1: Six real-world configuration errors that were made by system admins without following recommendations from manuals. (a)(b)(c) are from COMP-A's customer ticket database, (d)(e)(f) are new misconfigurations our work discovered from public Docker images and have been confirmed by multiple image maintainers [3–10].

default settings, three of which have already been confirmed by MySQL and Cassandra developers as real bugs. Incorrect default settings can easily cause configuration errors since system admins are most likely to go with the default [60].

2 Characteristic Study

Before we build a tool to extract good practice recommendations from manuals, we first collected and studied 261 real-world recommendations from manuals of six widely-deployed systems listed in Table 2. Our study answers two questions: (1) *Is it useful to extract those recommendations from manuals?* If they are all general advice such as "recommend to set it to a large value", extracting them is not very helpful since they cannot be used as specifications for automatically checking system admins' settings. In contrast, if the recommendations are clear specifications such as "recommend to set this to greater than 2000", extracting them out from manuals can help build checkers to detect violations to them. Additionally, have developers already put in their code to check if system admins follow these recommended practices? If so, there is no need to extract them from manuals. Finally, how often are these recommendations the default settings for the corresponding configuration parameters? If they are not default, why? *(2) How difficult is it to extract good practice recommendations from manuals?* In particular, are manuals structured enough for information extraction?

Observation 1: *157 (60%) of the studied good practice recommendations are specific instead of just general advice.* We manually studied all recommendations and categorized them based on their contents. If a recommendation is about something that is hard to be checked automatically, it is classified as a "general advice" (e.g. Table 3 last row). Otherwise, it is classified as a "clear specification", which is further categorized into value, usage, correlation, and property by what is recommended, as explained in the caption of Table 2. An example is given for each category in Table 3.

Table 2 shows the number of recommendations of each category. In total, 157 (60%) of the 261 recommendations describe clear specifications that if extracted can be used for automatically checking system admins' configuration settings. The remaining 104 recommendations are general advice that is hard to check automatically.

Observation 2: *152 (97%) of the specific good practices recommended in manuals are not checked in source code.* For each recommendation, we manually examine the source code of each software to see if the recommended practices are checked in source code to warn/inform system admins upon violations. Table 4 shows that only five out of the 157 specific recommendations in manuals are checked in source code.

Listing 1 shows an example where a recommended practice is checked in HBase code. In this case, if the practice is violated by system admins, they will be warned to reexamine the setting of this parameter more carefully.

The goal of our work is exactly to generate more checkings like the HBase case shown in Listing 1, i.e. automatically extract good practice specifications from manuals and build a checker to warn system admins when their settings do not follow the recommended practices.

Violations to good practices may not always be configuration errors. However, as previous work [60, 62] has shown,

Software	#Rec	Specific					General
		value	usage	correl	property	total	
MySQL	78	27	6	2	5	40	38
Httpd	92	25	16	8	3	52	40
PostgreSQL	49	21	1	3	3	28	21
HDFS	18	13	0	3	0	16	2
HBase	12	10	1	0	0	11	1
Spark	12	9	0	0	1	10	2
Total	261	105	24	16	12	157	104

Table 2: Characteristics of the 261 studied good practice recommendations from six widely used software. "Specific": describe a clear specification; "value": recommend to (not) set to one or multiple values (e.g. Table 3 row 2); "usage": recommend to (not) use an option, typically for command-line options without a value (e.g. Table 3 row 3). "correlation": recommend to set to a value smaller, larger or equal to another parameter (e.g. Table 3 row 4). "property": recommend to set to a value with some property, such as in the absolute path format(e.g. Table 3 row 5).

Category	Example Practice Description in Manuals
Value	It is generally not desirable to set this to a value *greater than 2000*.
Usage	This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is *not recommended for production use*.
Correlation	Setting this *lower than the dfs.namenode.replication.min* is not recommend and/or dangerous for production setups.
Property	It is best to specify the datadir value as *an absolute path*.
General	We recommend that this setting be kept to *a high value* for maximum server performance.

Table 3: Real examples of recommendations of different types.

Software	# (%) of prac checked in code
MySQL	1 (2.5%)
Httpd	1 (1.9%)
PostgreSQL	1 (3.6%)
HDFS	1 (6.3%)
HBase	1 (9.1%)
Spark	0 (0.0%)

Table 4: Number of good practices checked in source code.

```
if(balancedPreferencePercent
< 0.5) {
  LOG.warn("The value of " +
  DFS_DATANODE_BALANCED_SPACE
  _PREFERENCE_FRACTION_KEY +
  " is less than 0.5 so
  volumes with less available
  disk space will receive
  more block allocations");

}
```

Listing 1: Example of a good practice check in HBase's source code.

Software	Same -val	Multi -val	Rela -val	Cond -rec	No default	Others
MySQL	14	10	1	10	4	1
Httpd	16	6	3	9	16	2
PostgreSQL	10	8	2	6	0	2
HDFS	9	1	3	3	0	0
HBase	6	3	0	1	0	1
Spark	6	2	0	1	1	0
Total	61 (39%)	30 (19%)	9 (6%)	30 (19%)	21(13%)	6 (4%)

Table 5: The number of recommendations that are the same as the default and different categories of recommendations that are not the same as the default (multiple-value, relative-value, condition-recommendation, no-default, and others).

many system admins simply rely on guesswork or unreliable sources (e.g. online forums) to configure complex server software. If our checker can give a warning like Listing 1 when the settings do not follow practices, system admins can at least have a chance to *reexamine* the settings more carefully.

Observation 3: *96 (61%) of the specific good practice recommendations are not equivalent to default settings.* It is conceivable that some recommended practices might be the default settings (after all, the vendor recommends them). If this is the case, there is no need to extract recommended practices from manuals. System admins simply just go with default if they do not know how to set it better.

However, as shown in Table 5, only 61 (39%) good practices are equivalent to the default settings. For the majority (61%) cases, recommendations are not the same as default due to several reasons, including (a) 30 recommend multiple different values, e.g. a range or a set of values. In real settings, they may need to be modified to accommodate different situations, so it is worthwhile for sysadmins to double-check if the settings follow recommendations.; (b) 30 recommend some settings based on some conditions, e.g. "Enable A along with B"; (c) 21 recommendations are on command line options that have no default values; (d) 9 recommend relative values, such as "25% system RAM size"; (e) 6 cases have no clear reason why the default is different. They may be potential bugs and we have one of them confirmed as a bug by developers.

Observation 4: *The six studied manuals are organized in a similar structure.* As shown in Table 6, the six manuals are either in HTML or XML format and parameters in them are described in a similar structure:

- Each parameter is described in one separate section.
- Parameter names are often used as the section headings.
- There is some meta-info of the parameter described in the format of `<key>:<value>`, such as "Type:string".
- Most information related to each parameter is described in one or several paragraphs of free texts.

The per-parameter section structure makes it possible to relate each parameter name and its description by parsing the section structure. In addition, data types and default values can be used to identify parameter values in plain text descriptions which is necessary for generating specifications.

3 Design and Implementation

We design and implement PracExtractor to automatically extract recommendations from manuals, convert them into specifications and then uses them to detect violations. PracExtrac-

Software	Manual format	Parameter section?	Data type?	Default value?
MySQL	html	Yes	Table	Table
Httpd	xml	Yes	No	Table
PostgreSQL	html	Yes	KV	Text
HDFS	xml	Yes	No	Table
HBase	html	Yes	No	KV
Spark	html	Yes	No	Table

Table 6: Format and structure of manuals regarding how they describe configuration parameters. "Parameter section"—a separate section describes each individual parameter, "Data type"/"Default value" — the format they are described in, including table and KV (`<key>:<value>`).

tor faces two main challenges: (1) As manuals are written in plain texts and have a large amount of texts unrelated to recommendations, how to effectively filter noises and extract only recommendations? (2) Even after we extract recommendations, how to convert them into formal specifications that can be used to automatically check for violations?

To address the first challenge, PracExtractor breaks manual texts into sentences and extracts recommendation sentences with two filtering steps: keyword-based filtering (coarse grained) and syntactic-pattern-based filtering (fine grained). PracExtractor mines the keywords and syntactic patterns from the studied 261 recommendations.

To address the second challenge, PracExtractor first identifies semantic entities (e.g. parameter name and values) in a recommendation sentence and then convert it into a formal specification by matching them with semantic patterns.

3.1 Preprocessing and Parsing

PracExtractor first preprocesses and parses software manuals into parameter name, meta-info and free-text descriptions. The meta-info, including type and default value, is necessary for recognizing setting entities later (cf. §3.3). One special parameter type is enum, for which manuals usually also indicate all valid values along with a parameter. PracExtractor extracts the valid values for each parameter and uses them to identify enum values from a sentence in §3.3.

PracExtractor parses manuals based on the observed manual structure. Table 6 shows that manuals are usually written in HTML/XML formats with separate sections for different parameters. PracExtractor parses HTML and XML files, identifies each separate parameter section, and extracts parameter name, meta-info and free-text description from each. PracExtractor then breaks free-text descriptions into sentences.

Different manuals may still have slightly different formats for parameter sections. To handle that, PracExtractor takes an input of a small code snippet (format spec). A format spec is easy to write: according to our evaluation of twelve large manuals, they are typically fewer than 30 lines of Python code, and each of them can be written in 0.5-2 hours. (cf. Table 14).

Word	Covered sentences
recommend	74
well	26
good	26
appropriate	21
want	17

Bigram	Covered sentences
be recommended	34
should only	20
may want	13
good idea	7
with caution	7

Table 7: 10 sample keywords (words and bigrams) collected by PracExtractor from 261 studied recommendations and how many recommendations each covers.

3.2 Recommendation Sentences Extraction

Most sentences in manuals do not contain recommendations. For the twelve evaluated software manuals, 696–25510 sentences are extracted from parameter sections, but only 0.4%–2.7% of them contain recommendations. To extract these small percentage of recommendations, PracExtractor performs two steps filtering:

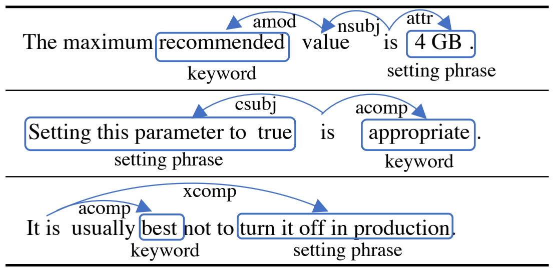
Keyword-based Filtering Following the intuition that recommendations are usually described with certain keywords (e.g. "recommend", "suggest"), PracExtractor extracts candidate recommendations with keyword filtering. To find out which words/phrases should be used as the keywords, PracExtractor first breaks the studied 261 recommendation sentences into individual words and bigrams (two consecutive words) and uses them as the candidate keywords T. PracExtractor then uses inverse document frequency (IDF) to rank the candidate keywords. IDF reflects how frequently a term t (word/bigram in our case) occurs in a set of sentences set S, as:

$$IDF(t,S) = \log \frac{|S|}{|\{s \in S : t \in s\}|}.$$

PracExtractor calculates $IDF(t,R)$ for the studied recommendations R and $IDF(t,S)$ for all the manual sentences S. PracExtractor ranks T based on $IDF(t,R)$ and $IDF(t,S)$ and get the smallest 100 and 300 terms separately as T_R and T_S. PracExtractor uses $T_R - T_S$ as the final keywords. The intuition behinds this is to find the words that are important in recommendations but not normal sentences. In Table 7, the sample keywords show that PracExtractor has effectively found keywords related to recommendations.

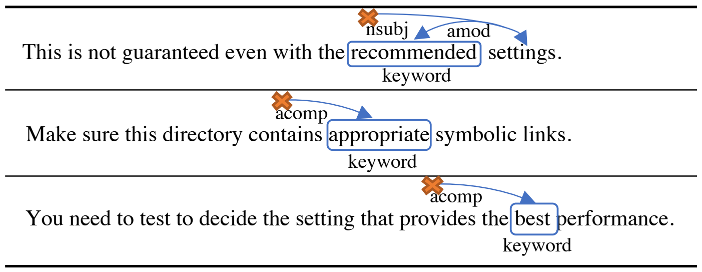
Syntactic-Pattern-based Filtering Using keyword filtering alone is not enough. After keyword-based filtering, only 7.3% of the remaining sentences are recommendations. Many sentences with the recommendation-related keywords are not true recommendations. Figure 2 (a) and (b) gives examples that the same keywords can be contained both in recommendation and non-recommendation sentences.

The key difference between these recommendations and non-recommendations in Figure 2 is their syntactic patterns.



amod
nsubj
attr
The maximum recommended value is 4 GB.
keyword
setting phrase
csubj
acomp
Setting this parameter to true is appropriate.
setting phrase
keyword
xcomp
acomp
It is usually best not to turn it off in production.
keyword
setting phrase

(a) Recommendation sentences



nsubj
amod
This is not guaranteed even with the recommended settings.
keyword
acomp
Make sure this directory contains appropriate symbolic links.
keyword
acomp
You need to test to decide the setting that provides the best performance.
keyword

(b) Non-recommendation sentences

Figure 2: Comparison of syntactic patterns of recommendation and non-recommendation sentences that contain likely-recommendation keywords. The patterns are labeled as undirected dependency paths from a keyword to a setting phrase, where a dependency path consists of a sequence of syntactic relations annotated with Universal Dependencies [23]: **amod** – link from a noun to an adjective modifier; **nsubj** – relation between a verb/noun and a prepositional phrase; **attr** – relation between a verb/adjective and a complement, etc.

Besides a keyword, the recommendations also contain a *setting phrase*, a noun/verb phrase describing what setting is recommended. Between such setting phrases and keywords, there are certain syntactic relations (patterns), which do not exist in non-recommendation sentences. PracExtractor leverage the syntactic-patterns to do fine-grained filtering.

PracExtractor first adopts the universal dependency (UD) tree [23] to represent a sentence's syntactic structure. A UD tree $T = (V,E)$ consists of vertices V and edges E, where $v \in V$ is labeled with a word's part of speech (POS) and $e \in E$ represents the syntactic dependency between two words (cf. Figure 2). Let $T' = (V,E')$ be an undirected correspondence of T, the syntactic pattern between a keyword and a setting phrase can be represented with an undirected path $p = (v_0, e'_{v_0,v_1}, v_1, \ldots, v_n)$, where v_0 is the keyword, v_n is the setting phrase, and $e'_{v_{i-1},v_i} \in E'$ for $i \in [1,n]$.

With the UD representation, PracExtractor mines the unique patterns for recommendations from the studied 261 recommendations S_{rec} and a set of non-recommendation samples $S_{\text{not\_rec}}$ that contains the keywords. For each sentence s, PracExtractor builds $T'_s = (V_s, E'_s)$ and extracts all paths

$$\begin{aligned}\rho_s = \{(v_0,\ e'_{v_0,v_1},\ v_1,\ \ldots,\ v_n) :\\ v_0 \in \textsc{Keywords} \ \wedge\ \forall i \in [1,n]\ e'_{v_{i-1},v_i} \in E'_s\\ \wedge\ \forall i \in [0,n]\ v_i \in V_s \ \wedge\ v_n \in \textsc{SettingPhrases}\},\end{aligned}$$

that starts from each keyword and ending at each setting phrase. The keywords are from the last step and the setting phrases are labeled by human inspectors. PracExtractor extracts all such paths from all recommendations and non-recommendation samples, denoted as P_{rec} and $P_{\text{not\_rec}}$. PracExtractor then extracts patterns P_{pattern} with Algorithm 1.

With the identified syntactic patterns P_{pattern}, PracExtractor classifies a new sentence s' into a recommendation or non-recommendation. PracExtractor traverse P_{pattern} and check if any pattern matched with s'. If at least one pattern matched then s' is classified as a recommendation otherwise non-recommendation. Such a matched pattern also labels the setting phrase in s' at the pattern's end (cf. Figure 2). The setting phrase will be used in specification generation (cf.§3.3).

Algorithm 1: Syntactic-pattern extraction algorithm

```
Input: $P_{\text{rec}} = \bigcup_{s \in S_{\text{rec}}} \rho_s$, $P_{\text{not\_rec}} = \bigcup_{s \in S_{\text{not\_rec}}} \rho_s$
Output: a pattern set $P_{\text{pattern}}$
$P'_{\text{rec}} \leftarrow []$, $P_{\text{pattern}} \leftarrow \emptyset$;
for $\rho_i \in P_{\text{rec}}$ do
    // Collect all the prefixes of $\rho_i$
    for $\rho_{i,j} \in \texttt{prefix}(\rho_i)$ do
        $P'_{\text{rec}}.\texttt{append}(\rho_{i,j})$;
// Traverse elements in $P'_{\text{rec}}$ in the order of frequency
// to extract the most general patterns
for $\rho_i \in \texttt{mostFrequentElement}(P'_{\text{rec}})$ do
    if $\texttt{prefix}(\rho_i) \cap P_{\text{pattern}} \neq \emptyset$ then
        continue;
    if $\rho_i \notin P_{\text{not\_rec}}$ then
        $P_{\text{pattern}} = P_{\text{pattern}} \cup \{\rho_i\}$;
return $P_{\text{pattern}}$
```

3.3 Specification Generation

In §3.2, PracExtractor identifies recommendation sentences and the setting phrases within it. PracExtractor then converts the setting phrases into checkable, formal specifications. Table 8 gives three example recommendations and the corresponding specifications. In general, PracExtractor can generate four types of specifications, including value, correlation, usage and property, as shown in Table 9.

A naïve way to generate specifications is to match setting phrases with predefined regular expressions and convert them accordingly. This can transform simple phrases with numbers (e.g. "less than 8"), but cannot convert more complex phrases (e.g. phrases with `enum` or parameter names). For instance, a phrase could be "set to chain", where "chain" is an `enum` value in Httpd. Such software-specific words can hardly be predefined in regular expressions and so cannot be matched.

Sentence	Specification
It is recommended to $\boxed{\text{enable this option}}$.	p == true
$\boxed{\text{A value between 8 to 16}}$ is suggested.	p ∈ [8, 16]
We suggest to set it $\boxed{\text{less than ThreadsPerChild}}$.	p < ThreadsPerChild

Table 8: Examples of specifications generated by PracExtractor. Setting phrases are marked with rectangles.

Category	Specification	Description Patterns Example
value/ correlation	p == v p < v | p > v p ∈ [v, v′] p ∈ {v, v′}	$\text{v}_{\text{<value>}}$ less_{syn} | more_{syn} than $\text{v}_{\text{<value>}}$ $\text{between}_{\text{syn}}$ $\text{v}_{\text{<value>}}$ to $\text{v}'_{\text{<value>}}$ $\text{v}_{\text{<value>}}$ or $\text{v}'_{\text{<value>}}$
correlation	with (p, p′) prefer (p, p′)	$\text{along}_{\text{syn}}$ with $\text{p}'_{\text{<para>}}$ $\text{prefer}_{\text{syn}}$ $\text{p}'_{\text{<para>}}$
usage	use (p)	used_{syn} | $\text{useful}_{\text{syn}}$
property	format (p, f)	$\text{f}_{\text{<format>}}$

Table 9: Category of specifications PracExtractor generates and example of patterns for each specification. "<value>" is defined as "<bool>|<num><unit>?|<enum>|<parameter>" from Table 10. "less_{syn}" means the synonyms of "less".

PracExtractor addresses this issue in three steps. First, given a recommendation sentence, PracExtractor identifies which parameter the sentence is associated with. Then, PracExtractor uses the parameter's meta-info (e.g. type and default value) extracted before (cf. §3.1) to identify setting entities, such as values and formats. Third, PracExtractor matches the identified setting entities with predefined semantic patterns to generate specifications.

Identify Parameter Names PracExtractor first identifies which parameter a sentence is associated with. For a recommendation sentence s in a paragraph p related to parameter X, there are four possible cases: 1) Only X is mentioned in s. PracExtractor determines this sentence is for X; 2) Another parameter Y is mentioned in s. PracExtractor checks if Y is a subject or object to a verb like "set" or "specify" and determines the sentence is for Y if it is the case; 3) No parameter is mentioned in s. PracExtractor further searches previous sentences in paragraph p; 4) No parameter is mentioned in p, PracExtractor determines the sentence is for X.

Identify Settings Entities Give the identified setting phrase and associated parameter of a sentence, PracExtractor then recognizes setting entities (e.g. values and formats) from the setting phrase based on the associated parameter's type. Table 10 shows the seven types of setting entities that PracExtractor can identify. For different types of setting entities, PracExtractor identifies them with different syntax:

Type	Setting Syntax
<bool>	"enable" | "on" | "true" | "disable" | "false" | "off"
<num>	[-+]?\d+(\.\d+)?
<unit>	"byte" | "MB" | "ms" | "%" | "% of RAM" | …
<enum>	∀w ∈ VALID_VALUES
<parameter>	∀w ∈ ALL_PARAMETERS
<format>	"email address" | "absolute path" | "domain name" | …
<string>	∀w ∉ (<bool> ∪ <num> ∪ <unit> ∪ <enum> ∪ <parameter> ∪ <format>)

Table 10: Types of setting entities PracExtractor identifies from recommendation sentences. `VALID_VALUES` and `ALL_PARAMETERS` are from the type-info and parameter list that PracExtractor identifies in the parsing step (cf. §3.1).

- For basic types, including `<bool>` and `<num>`, PracExtractor identifies them with regular expressions. For `<num>`, PracExtractor also identifies common `<unit>` (e.g. "GB", "byte") along with it.
- For `<enum>`, PracExtractor takes advantage of parameters' type-info to identify it. The type-info of an `<enum>` parameter does not only indicate it is `<enum>` but also indicates the valid values that can be set to the parameter. PracExtractor has parsed these in the parsing step (cf. 3.1) and now searches a setting phrase for the valid values.
- `<parameter>` is for the case that the setting phrase describes current parameter with respect to another parameter, such as "set A to be larger than B". PracExtractor identifies this by searching for valid parameter names in the setting phrase. Note PracExtractor has extracted all parameter names in the parsing step (cf. 3.1).
- For `<format>`, PracExtractor identifies common formats (e.g. "email address", "absolute path") of parameters based on word matching. PracExtractor allows users to provide new words to extend the identifiable formats.
- All other words/phrases are identified as `<string>`. PracExtractor further handles two kinds of strings that have special meaning. First, some of them use "this value" or "this" to refer to a value mentioned in previous sentences. PracExtractor recognizes such references and identifies the actual value from previous sentences. Second, some may use "default" to refer to parameters' default value. In this case, PracExtractor uses the corresponding default value extracted from the parsing step (cf. §3.1) as the recommended setting.

Generate Specification PracExtractor generates checkable formal specifications by matching the setting phrases with predefined semantic patterns. Table 9 lists the types of specifications that can be generated by PracExtractor, each with an example pattern. Before blindly matching a setting phrase with patterns, PracExtractor first considers the associated parameter's data type. For example, if the data type is `<bool>` or `<enum>`, there is no need to match semantic patterns like "less than `<value>`", "between `<value>` and `<value>`", or anything that is for `<num>`. Second, synonyms are also considered for these patterns. For instance "less$_{syn}$ than" can also match with "lower than" and "smaller than", etc.

Detect Negation PracExtractor also detects negation in a recommendation sentence and negates a specification when necessary. Two different types of negations are handled by PracExtractor. First, PracExtractor finds direct negation words, such as "not" (or abbreviation "n't") "none" and "never", in sentences. In addition, PracExtractor also detects indirect negation words and phrases, such as "avoid", "with caution", "rarely", and "seldom", etc. to identify the negation.

3.4 Violation Detection from Configurations

PracExtractor parses a configuration file and turns the settings into key-value pairs of (`parameter name`, `value`). Although the formats of configuration files for different software can vary depending on the software implementation, most of them have similar formats. The configuration files of the twelve popular systems in our evaluation, including MySQL, Httpd, HBase, HDFS, Spark, Squid and even the commercial system, all follow two common and simple formats: key-value pairs with separator like '=' or ':' or XML format.

Then PracExtractor checks the parsed parameter values against extracted specifications and generates warning messages if it detects violations. PracExtractor uses the *original sentences* from manuals as the warning messages. An example warning could be "Setting dfs.safemode.replication.min lower than dfs.replication.min is not recommended and is dangerous for production setups". This warning can remind sysadmins to double-check the configurations to avoid potential mistakes, just like the one shown in Figure 1 and 3.

Most of the violations detected by PracExtractor cannot be detected by previous works. Previous works [44, 61, 63] mainly use configuration checking/usage logic in source code to detect misconfigurations. However, most recommendations (97% as in Table 4) are not checked in source code, so violations to them cannot be detected by these works.

4 Experimental Evaluation

As shown in Table 11, we evaluate PracExtractor on twelve large software systems including eleven popular open-source server systems and one commercial systems (COMP-A) from a public company that serves many enterprise customers. These systems' manuals have 543 to 8283 pages. These manuals include not only the six manuals in our characteristic study, but also six new manuals that are not studied before. We evaluate both the precision and recall of PracExtractor by comparing its results with recommendations identified by human inspectors. In our evaluation, two human inspectors manually and independently examined each manual to identify recommendations.

For violation detection, we evaluate PracExtractor with real-world settings from top-ranked container images on DockerHub (200 images for each open-source systems). We run PracExtractor against configuration files in these docker images to detect violations to the specifications extracted from manuals. The evaluated images include both Linux-based and Windows-based one. However, as PracExtractor currently does not support platform-specific checking, our evaluation does not include checking platform-related specifications. In total, only 4 specifications are platform-related.

Recommendation and Specification Extracted As shown in Table 11, PracExtractor extracts a total of 338 good practice recommendations (including specific and general advice) from manuals and automatically converts 173 of them into formal specifications that can be used to check system admins' configuration settings. Among all software, Httpd has the most number (81) of recommendations extracted as well as the most number (31) of specifications generated.

Results with the six "new" manuals excluded from our study PracExtractor works reasonably well with the six "new" software manuals that are not included in our characteristic study. PracExtractor extracts 117 recommendations and 59 specifications from these manuals. For example, it extracts 35 recommendations and 22 specifications for the commercial software, COMP-A. The precision and recall are only slightly lower than the one for the six studied manuals, but are still reasonably good (with a 0.83 precision and 0.80 recall for recommendations, and 0.88 precision and 0.66 recall for specifications).

Violations Detected PracExtractor detects in total 1423 practice violations from 853 unique images. We manually validated all the violations. We reported 325 (that are maintained on GitHub) of them to their maintainers and have got 47 confirmed as real configuration issues, including six in images with >1M downloads and 28 in images with >1K downloads.

Table 12 shows a breakdown of all detected violations. 426 are "wrong change", namely a parameter is explicitly changed to a non-recommended value by system admins. 997 violations are "wrong default", namely a parameter has a non-recommended default value but is not changed. This also matches with the finding from a previous real-world misconfiguration study [60] that many configurations are left as default. In this case, system admins are afraid of changing default settings, even though the default is not recommended or is simply a placeholder value, which needs system admins to explicitly change to fit their own system environments.

Software	Category	Update Time of Manuals	# Recommendations total	# Recommendations extracted	# Recommendations precision	# Recommendations recall	# Specifications total	# Specifications generated	# Specifications precision	# Specifications recall
MySQL	database	Aug. 2019	78	61	0.90	0.78	40	30	0.88	0.75
Httpd	web server	Aug. 2019	92	81	0.83	0.88	52	31	0.79	0.60
PostgreSQL	database	Aug. 2019	49	38	0.95	0.78	28	20	0.87	0.71
HDFS	distributed storage	Aug. 2019	18	17	1.00	0.94	16	14	0.93	0.88
HBase	distributed storage	Aug. 2019	12	12	1.00	1.00	11	11	1.00	1.00
Spark	distributed computing	Aug. 2019	12	12	0.86	1.00	10	8	0.89	0.80
COMP-A	commercial storage	May 2019	49	35	0.70	0.71	37	22	1.00	0.59
Nginx	proxy	Jul. 2019	26	24	0.92	0.92	6	4	0.50	0.67
Flink	stream processing	Aug. 2019	10	6	0.67	0.60	6	4	1.00	0.67
Squid	proxy	Feb. 2019	22	18	0.86	0.82	13	9	0.82	0.69
Mapred	distributed computing	Aug. 2019	25	20	0.95	0.80	15	9	1.00	0.60
Cassandra	distributed storage	Aug. 2019	15	14	0.93	0.93	13	11	0.85	0.85
studied			261	221	0.89	0.85	157	114	0.87	0.73
new			147	117	0.83	0.80	90	59	0.88	0.66
overall			408	338	0.86	0.83	247	173	0.87	0.70

Table 11: Numbers of recommendations extracted and specifications generated by PracExtractor and corresponding accuracy (precision=$\frac{\text{TruePositive}}{\text{TruePositive+FalsePositive}}$ and recall=$\frac{\text{TruePositive}}{\text{TruePositive+FalseNegative}}$). Recommendations consist of both general advice and specific ones that can be converted into specifications. "studied" refers to the software included in our characteristic study (first 6 rows) , while "new" refers to other software not included in our study (last 6 rows) .

Software	Wrong change	Wrong default	Software	Wrong change	Wrong default
MySQL	20	200	Nginx	0	0
Httpd	338	200	Flink	0	0
PostgreSQL	8	0	Squid	20	0
HDFS	21	0	Mapred	0	0
HBase	0	199	Cassandra	9	398
Spark	10	0			
Total	Wrong change	426	Wrong default	997	

Table 12: Detected good practice violations in container images from Dockerhub. We reported 325 violations to the image owners, and 47 of them have been confirmed as real configuration errors. Three wrong defaults are also confirmed by MySQL and Cassandra.

Figure 3 gives three examples of real-world violations that are detected by PracExtractor from popular container images on DockerHub. They have been confirmed by the image owners as real configuration errors or by the software developers as real bugs. Here are the root causes:

(a) HDFS manual recommends to leave the parameter as `true` to avoid registration of excluded hostnames. However, it is ignored and violated in 21 images, which can cause security issues. We reported them and so far two of them have been confirmed [11, 12].

(b) Cassandra manual does not recommend to enable the experimental feature as it may cause potential failure. However, the default setting enables it, which is a bug, and 199 images just keep the default. The bug has been confirmed and fixed by Cassandra in its new version [1].

(c) The default setting for this parameter in MySQL is much larger than the recommended value in the manual, and the default value (set by MySQL developers) is actually incorrect. We report it to MySQL official Bugzilla and it has been confirmed as a bug [16].

Indeed, not all the 1423 violations are configuration errors or bugs. However, as discussed before in the real-world example from HBase (cf. §2 Listing 1) that explicitly performs such checks in its source code, when such violations are warned to system admins, they at least get a chance to *reexamine and reconsider* the settings more carefully.

Maintainers' Feedback on Violations We reported 325 to the image maintainers and so far have got 47 violations confirmed that they need to be changed. We list three example confirmations in Table 13 (row 1, 2, 3). We took a further look into the impact of these confirmed violations: 11 cause security vulnerabilities, 31 cause unreliable services, 2 cause performance issues and 3 cause database inconsistency.

We also got 46 other feedbacks that the maintainers hesitated to fix the violations. They either think it is the upstream vendors' responsibility to handle the issues (Table 13 row 4, 5) or are aware of the limitations but make the settings for particular environments (Table 13 row 6).

Lines of Customized Code for Each Manual Table 14 shows the lines of Python code (LOC) of the format spec for each manual. It needs only 6-73 LOC for the software manuals with hundreds and thousands of pages. Also, the LOC is not proportional to the number of pages in manuals, and it is only one-time effort to each software. On average, it needs little effort (0.5-2 hours) to customize PracExtrac-

HDFS

Parameter: *dfs.namenode.datanode.registration.ip-hostname-check*
Default: *true*
Recommendation: *true*
Misconfiguration: *false*
Violated docker images:
Babbleshack/hadoop, jamesmcclain/hadoop

Recommendation in manual:
"It is recommended that *this setting be left on* to prevent accidental registration of datanodes listed in the excludes file…"

(a)

Cassandra

Parameter: *enable_materialized_views*
Default: *true*
Recommendation: *false*
Misconfiguration: keep wrong default
Violated docker images:
199 images

Recommendation in manual:
"Materialized views are considered experimental and are not recommended for production use."

(b)

MySQL

Parameter: *max_binlog_cache_size*
Default: *2^64*
Recommendation: *4GB*
Misconfiguration: keep wrong default
Violated docker images:
200 images

Recommendation in manual:
"The maximum *recommended value is 4GB* …MySQL currently cannot work with binary log positions greater than 4GB."

(c)

Figure 3: Example of new violations detected by PracExtractor in popular images from DockerHub. The violations have been confirmed by image maintainers [11, 12] and software vendors (Cassandra and MySQL). Cassandra has fixed (b) in its new version [1].

Image	Feedback on Violation Reports from Image Maintainers
eviles/ httpd	"Ok, I've fixed two images: eviles/httpd, eviles/httpd-tomcat.' [3]
newnius/ hbase	"Thanks for pointing out this. I have added that to the default configuration files and rebuilt the images." [5]
oscerd/ cassandra	"Yes we can do that. I can update the configuration for 3.10 and 3.11." [9]
vitessio/ mysql	"As a general strategy, I plan to use MySQL's default values unless there is a strong use-case to override." [13]
madflojo/ cassandra	"Since we are using the upstream cassandra/latest, I'd prefer that this issue go to them." [14]
publicisw/ httpd	"We are aware of these limitations...This is only used on new linux kernels, which should support this feature..." [15]

Table 13: Example of positive and less-positive (in gray background) feedback. Due to page limit, we list three for each.

Software	Manual pages	LOC	Software	Manual pages	LOC
MySQL	5494	73	COMP-A	8283	18
Httpd	1009	10	Nginx	543	24
PostgreSQL	3724	24	Flink	6152	6
HDFS	1031	12	Squid	1391	10
HBase	787	14	Mapred	1318	12
Spark	599	6	Cassandra	913	11

Table 14: Lines of code (LOC) of format specs for the twelve evaluated manuals.

tor for a new software manual, including even for the large commercial software manual with 8283 pages.

Accuracy — False Negatives and False Positives Table 11 shows PracExtractor's accuracy in terms of recall and precision. For recommendation extraction, PracExtractor has a reasonable high recall, 0.83. In other words, overall, PracExtractor misses only 17% of the recommendations. For specification extraction, PracExtractor's recall is slightly lower (0.70) (i.e. miss 30% of the specification). This is mainly because some descriptive texts and string values are hard to be automatically recognized and converted into specification (cf. Table 15). This can be further improved by analyzing the semantics of parameter names so that string values can be better matched with parameter meanings.

PracExtractor has low false positives, too. Its overall precision is 0.86 for recommendation and 0.87 for specification. That is, only 14% of the recommendations and 13% of the specifications extracted by PracExtractor are false positives. The false positives are introduced mainly due to texts are incorrectly identified as parameter values (cf. Table 15).

Impacts of False Positives The false positives will not cause serious impacts as they can be recognized easily. Table 16 shows three cases of false recommendations that PracExtractor extracted from the evaluated manuals. They *are* descriptions related to the parameters but just are not recommendations. For example, "there may be circumstances where it is desirable for a configuration section's authorization to be combined with that of its predecessor" is a false positive, which describes a parameter usage but gives no recommendation. Since PracExtractor includes such an *original* sentence from manuals in a warning message, sysadmins who read it can easily realize that it is not a recommendation and will not be misguided.

Evaluation with Existing Misconfiguration Dataset We also evaluate PracExtractor with existing configuration issues. As there is no existing dataset for good practice violations, we use a dataset [2] for general configuration issues used in previous works [60, 61]. This dataset contains configuration issues from various online forums and mailing lists. We use the issues categorized as "error" to evaluate PracExtractor. Out of the 63 evaluated configuration errors, PracExtractor can detect 7 (10%) of them. We validated that these detected errors cannot be detected by previous works [35, 61] as they violate no constraint in source code while previous works use code constraints to detect errors. For the undetected errors, we found that manuals do not provide recommendations for

Software	False Negatives		
	R1	R2	R3
Httpd	10	11	0
COMP-A	8	3	4
Mapred	1	2	3

Software	False Positives	
	R4	R5
Httpd	4	4
Nginx	4	0
Squid	2	0

Table 15: Root causes for PracExtractor's False Negative and False Positive. R1 — descriptive recommendations that are not identified, such as "it is recommended to not configure a ticket key file". It is hard to automatically infer that this refers to not using `SSLSessionTicketKeyFile`. R2 — unknown string values that are not identified. For example, in "it's recommended the username 'anonymous' is in allowed userIDs", it is hard to recognize the common word "anonymous" as a value. R3 — sentences that are not covered by our syntactic patterns. R4 — texts that are incorrectly identified as parameter values. R5 — non-recommendation sentences that are mismatched with syntactic patterns of recommendations.

the error-related parameters. With a grain of salt, this shows that current manuals have not provided enough recommendations for avoiding many real-world misconfigurations. Further enriching manuals with good practices may improve PracExtractor's detecting capability and also benefit system admins who refer to manuals to resolve configuration issues.

5 Discussion

Generality. PracExtractor is reasonably general for manuals from different software. As our evaluation on six new manuals shows, PracExtractor can identify most (80%) recommendations from these manuals with a precision of 83%.

Human Effort. PracExtractor can easily be extended with new format specs to accommodate future manuals. In our evaluation, the specs for the six new software manuals are all less than 30 lines (cf. Table 14) and were written by a first-year graduate student, each in 0.5-2 hours.

Accuracy of Manuals. Another concern is whether manuals themselves deliver accurate information for PracExtractor to extract. After all, manuals can be outdated and can have mistakes made by the writers. In our work, we considered these factors. First, we validated the last update time of our evaluated manuals. As shown in Table 11, all twelve manuals are updated recently this year. Second, we compared the specifications extracted from manuals against source code and reported all differences to developers to check. If either is wrong/obsolete, it may introduce problems. Interestingly, in three cases, developers from MySQL and Cassandra confirmed that the source code is wrong (cf. Figure 3).

6 Related Work

Misconfiguration detection and troubleshooting. Many works have been done on detecting [25, 28, 30, 61, 66, 68] and troubleshooting [17–19, 42, 43, 54, 56, 57, 65, 69] configuration errors. Almost all of these works detect configuration errors by either checking against (a) patterns mined from tons of configuration files, or (b) constraints inferred from source code. Our work is complementary to these approaches. We extract recommendations from vendor-provided manuals as specifications, use them to detect violations, and warn system admins to reexamine the violations. Each of the approaches has its own strengths and weaknesses. Below, we compare our approach with each previous one respectively.

Xu [61, 63], Rabkin [44], and Nadi [35] propose approaches to extract configuration constraints from source code using static analysis. While it can infer simple constraints such as parameter types, range and some simple dependencies, it is less effective in checking against more complex constraints, especially *configuration settings that are legitimate but may not be good or optimal*. In our work, we focus on good practices recommended by vendors. If a system admin configures a parameter with a valid setting but does not achieve the intended goal (e.g performance, reliability or security goal), previous works that focus on detecting *invalid* configurations will not report any problem. In comparison, our PracExtractor can still warn him/her about the setting if it does not follow the good practices PracExtractor extracts from manuals.

Encore [68], PeerPressure [54] and Santolucito et al's work [47, 48] propose to extract configuration constraints and good practices from existing settings. These approaches assume that a large number of *independent* configuration samples are available for learning and the correct configurations are the common ones. This assumption can be true for some systems where configuration settings can be collected from users/customers back to vendors. However, for many enterprise software such as database or storage systems that are mainly deployed in enterprises (e.g. financial companies and government), each system's configuration settings are confidential information and cannot be shared back with vendors. As such, these approaches are less applicable. In comparison, PracExtractor is applicable to such scenarios because it automatically extracts good practice recommendations that *are already written in vendor-provided manuals*, and the check-

Parameter	False Positive Recommendations
adaptive_hash _index	It may be desirable to dynamically enable or disable adaptive hash indexing to improve query performance.
AuthMerging	There may be circumstances where it is desirable for a configuration section's authorization to be combined with that of its predecessor.
LimitRequest Line	When name-based virtual hosting is used, the value for this directive is taken from the default (first-listed) virtual host best matching the current IP address and port combination.

Table 16: Example of false positive recommendations PracExtractor extracted. They can be easily recognized by system admins and will not misguide them to make wrong changes.

ers generated by PracExtractor can be shipped to enterprise customers to check their configuration settings.

In addition to the above two approaches, another closely related work is ConfSeer [41], which takes a user's configuration of one or multiple parameters to search against the vendor's Knowledge Base (KB) articles and identifies those highly relevant ones so that users can read those KB articles to self-diagnose and self-correct misconfiguration (with no need to call customer support). While this work also uses NLP techniques, their goal is to narrow down the match so that users do not need to read hundreds of KB articles. For a given configuration parameter setting from a user, ConfSeer returns *a ranked list of KB articles for the user to explore, in a way similar to a search engine like Google that tries to return the most relevant web pages to a user's query*. In other words, ConfSeer is an improved Google search engine for configuration-related KB articles. In comparison, our goal is to extract configuration recommendations from manuals and convert them into *formal and checkable* specifications. Our checker can be shipped to the customer sites to *automatically* detect violations and warn system admins to reexamine their settings to proactively avoid problems instead of waiting for postmortem troubleshooting.

Inferring specification from text. Some past works also aim to infer specifications from program-related texts, including from program comments [49, 50], API documents [39, 67, 70, 71], and man pages [58]. Our work differs from previous works both on purposes and techniques. First, instead of helping developers find bugs in source code as in [49, 50, 70], our work aims to help system admins detect misconfigurations in their system settings. Secondly, comments and API documents have relatively uniform structures, which makes it easy to extract information like function names and variable types. In comparison, manuals are much less structured. The only structure is that each parameter has its own section/chapter. Inside a section, it is mostly free text. Thirdly, PracExtractor extracts much more complex constraints than previous work on man pages [58]. DASE [58] uses regular expression to extract valid options from man pages. In comparison, PracExtractor can extract option value, correlation and property from software manuals.

7 Conclusions and Future Work

This paper focused on the usefulness and feasibility of extracting good practice recommendations from software manuals to detect configuration problems. Specifically, we first conducted a characteristic study on 261 recommendations from six large open source software manuals. Based on the observations learned, we designed and implemented a tool, called PracExtractor, that can extract 338 recommendations and generate 173 specifications with reasonable accuracy from twelve large software manuals, including one for a large commercial software system. Additionally, with the generated specifications, PracExtractor have detected 1423 violations from 853 container images on DockerHub. We reported 325 of them and so far have got 47 confirmed as real configuration issues by the image maintainers from different organizations.

Interestingly, in addition to detecting system admins' configuration problems, PracExtractor can also help detect incorrect default settings for configuration parameters. When a default setting differs from a recommendation in the manual, it may indicate that the default setting is wrong. Incorrect default settings can easily cause configuration errors because system admins are most likely to go with default [60]. In our experiments, we did discover a few such software bugs, and three of them have already been confirmed respectively by MySQL and Cassandra.

PracExtractor is far from perfect. First, there is still much space to further improve its accuracy based on our analysis of false positives and false negatives (cf. Table 15). Further semantic analysis of parameter descriptions can improve the identification accuracy of parameter type, name and value. Second, PracExtractor currently cannot extract specifications with descriptive conditions, such as "set A with a large workload". This may possibly be handled by further incorporating domain knowledge of common descriptions and sub-clause analysis techniques.

Several other directions are also valuable to explore in the future. First, from our experience, we found that a uniform structure (e.g. per-parameter section) and consistent word usage (e.g. recommend) benefit extracting recommendations a lot. Therefore, it would be interesting to explore how manuals may be restructured so more information can be automatically extracted. Second, PracExtractor may be potentially used to detect documentation drift — manuals are not updated along with source code. By combining PracExtractor with source code analysis tools, it is possible to compare the configurations described in manuals and used in source code. Third, while PracExtractor focuses on analyzing user manuals, the approach may be applicable to extract good practices from other text-based documents such as knowledge-base (KB) articles, which are used by support engineers to troubleshoot customer issues.

Acknowledgments

We greatly appreciate the anonymous reviewers and our shepherd, Scott D. Stoller, for their insightful comments and feedback. We thank the Opera group, the Systems and Networking group at UCSD as well as Anita Jinda, Sam Lin and Jianmo Ni for useful discussions and paper proofreading. We thank Tianyin Xu for sharing a configuration issue dataset with us. This work is supported in part by NSF grants (CNS-1814388, CNS-1526966) and the Qualcomm Chair Endowment.

References

[1] Cassandra release note. https://gitbox.apache.org/repos/asf?p=cassandra.git;a=blob;f=NEWS.txt;h=ead28f0ac3d8d93c1ad87a3b944c0c72345257c1;hb=HEAD.

[2] Configuration datasets. https://github.com/tianyin/configuration_datasets.

[3] Github issue. https://github.com/eviles/docker/issues/1.

[4] Github issue. https://github.com/sspreitzer/docker-httpd-mirror/issues/1.

[5] Github issue. https://github.com/newnius/Dockerfiles/issues/4/.

[6] Github issue. https://github.com/F21/hbase/issues/2.

[7] Github issue. https://github.com/binhnv/docker-hbase/issues/1.

[8] Github issue. https://github.com/Boostport/hbase-phoenix-all-in-one/issues/1.

[9] Github issue. https://github.com/oscerd/cassandra-image/issues/1.

[10] Github issue. https://github.com/femiwiki/cassandra/issues/3.

[11] Github issue. https://github.com/Babbleshack/docker-hadoop-yarn/issues/1.

[12] Github issue. https://github.com/jamesmcclain/HadoopDocker/issues/8.

[13] Github issue. https://github.com/vitessio/vitess/issues/5056.

[14] Github issue. https://github.com/madflojo/cassandra-dockerfile/issues/1.

[15] Github issue. https://github.com/publicisworldwide/docker-stacks/issues/31.

[16] Mysql bugzilla. https://bugs.mysql.com/bug.php?id=94487.

[17] Bhavish Agarwal, Ranjita Bhagwan, Tathagata Das, Siddharth Eswaran, Venkata N Padmanabhan, and Geoffrey M Voelker. Netprints: Diagnosing home network misconfigurations using shared knowledge. In *NSDI*, volume 9, pages 349–364, 2009.

[18] Mona Attariyan, Michael Chow, and Jason Flinn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 307–320, 2012.

[19] Mona Attariyan and Jason Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *OSDI*, volume 10, pages 1–14, 2010.

[20] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013.

[21] Irina Ceaparu, Jonathan Lazar, Katie Bessiere, John Robinson, and Ben Shneiderman. Determining causes and severity of end-user frustration. *International journal of human-computer interaction*, 17(3):333–356, 2004.

[22] CNN. Here's why you may have had internet problems today. https://money.cnn.com/2017/11/06/technology/business/internet-outage-comcast-level-3/index.html, 2017.

[23] Marie-Catherine De Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D Manning. Universal stanford dependencies: A cross-linguistic typology. In *LREC*, volume 14, pages 4585–92, 2014.

[24] Inc. Docker. Docker hub. https://hub.docker.com/, 2019.

[25] Nick Feamster and Hari Balakrishnan. Detecting bgp configuration faults with static analysis. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 43–56. USENIX Association, 2005.

[26] Haryadi S Gunawi, Mingzhe Hao, Riza O Suminto, Agung Laksono, Anang D Satria, Jeffry Adityatama, and Kurnia J Eliazar. Why does the cloud stop computing?: Lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 1–16. ACM, 2016.

[27] Peng Huang. *Toward Understanding and Dealing with Failures in Cloud-Scale Systems*. PhD thesis, UC San Diego, 2016.

[28] Peng Huang, Chuanxiong Guo, Jacob R Lorch, Lidong Zhou, and Yingnong Dang. Capturing and enhancing in situ system observability for failure detection. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 1–16, 2018.

[29] Weihang Jiang, Chongfeng Hu, Shankar Pasupathy, Arkady Kanevsky, Zhenmin Li, and Yuanyuan Zhou. Understanding customer problem troubleshooting from storage system logs.

[30] Yu Jin, Nick Duffield, Alexandre Gerber, Patrick Haffner, Subhabrata Sen, and Zhi-Li Zhang. Nevermind, the problem is already fixed: proactively detecting and troubleshooting customer dsl problems. In *Proceedings of the 6th International Conference on emerging Networking EXperiments and Technologies*, page 7. ACM, 2010.

[31] Emre Kiciman and Yi-Min Wang. Discovering correctness constraints for self-management of system configuration. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 28–35. IEEE, 2004.

[32] Ben Maurer. Fail at scale: Reliability in the face of rapid change. *ACM Queue*, 13(8):30, 2015.

[33] Valerie Mendoza and David G Novick. Usability over time. In *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, pages 151–158. ACM, 2005.

[34] Justin Meza, Tianyin Xu, Kaushik Veeraraghavan, and Onur Mutlu. A large scale study of data center network reliability. In *Proceedings of the Internet Measurement Conference 2018*, pages 393–407. ACM, 2018.

[35] Sarah Nadi, Thorsten Berger, Christian Kästner, and Krzysztof Czarnecki. Mining configuration constraints: Static analyses and empirical results. In *Proceedings of the 36th International Conference on Software Engineering*, pages 140–151. ACM, 2014.

[36] David G Novick and Karen Ward. Why don't people read the manual? In *Proceedings of the 24th annual ACM international conference on Design of communication*, pages 11–18. ACM, 2006.

[37] David Oppenheimer, Archana Ganapathi, and David A Patterson. Why do internet services fail, and what can be done about it? In *USENIX symposium on internet technologies and systems*, volume 67. Seattle, WA, 2003.

[38] OWASP. Top 10-2017 a6-security misconfiguration. https://www.owasp.org/index.php/Top_10-2017_A6-Security_Misconfiguration, 2017.

[39] Rahul Pandita, Xusheng Xiao, Hao Zhong, Tao Xie, Stephen Oney, and Amit Paradkar. Inferring method specifications from natural language api descriptions. In *Proceedings of the 34th International Conference on Software Engineering*, pages 815–825. IEEE Press, 2012.

[40] C Perrow. Normal accidents: living with high-risk technologies (basic, new york). 1984.

[41] Rahul Potharaju, Joseph Chan, Luhui Hu, Cristina Nita-Rotaru, Mingshi Wang, Liyuan Zhang, and Navendu Jain. Confseer: leveraging customer support knowledge bases for automated misconfiguration detection. *Proceedings of the VLDB Endowment*, 8(12):1828–1839, 2015.

[42] Andrew Quinn, David Devecsery, Peter M Chen, and Jason Flinn. Jetstream: Cluster-scale parallelization of information flow queries. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 451–466, 2016.

[43] Ariel Rabkin and Randy Katz. Precomputing possible configuration error diagnoses. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 193–202. IEEE Computer Society, 2011.

[44] Ariel Rabkin and Randy Katz. Static extraction of program configuration options. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 131–140. IEEE, 2011.

[45] Ariel Rabkin and Randy Howard Katz. How hadoop clusters break. *IEEE software*, 30(4):88–94, 2013.

[46] James Reason. *Human error*. Cambridge university press, 1990.

[47] Mark Santolucito, Ennan Zhai, Rahul Dhodapkar, Aaron Shim, and Ruzica Piskac. Synthesizing configuration file specifications with association rule learning. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):64, 2017.

[48] Mark Santolucito, Ennan Zhai, and Ruzica Piskac. Probabilistic automated language learning for configuration files. In *International Conference on Computer Aided Verification*, pages 80–87. Springer, 2016.

[49] Lin Tan, Ding Yuan, Gopal Krishna, and Yuanyuan Zhou. /* icomment: Bugs or bad comments?*. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 145–158. ACM, 2007.

[50] Lin Tan, Yuanyuan Zhou, and Yoann Padioleau. acomment: mining annotations from comments and code to detect interrupt related concurrency bugs. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 11–20. IEEE, 2011.

[51] Chunqiang Tang, Thawan Kooburat, Pradeep Venkatachalam, Akshay Chander, Zhe Wen, Aravind Narayanan, Patrick Dowell, and Robert Karl. Holistic

configuration management at facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 328–343. ACM, 2015.

[52] TechCrunch. Facebook blames a server configuration change for yesterday's outage. https://shorturl.at/opuEG, 2019.

[53] virtualizationreview. Configuration error leads to another amazon web services data breach. https://virtualizationreview.com/articles/2017/06/21/configuration-error-leads-to-another-aws-data-breach.aspx, 2017.

[54] Helen J Wang, John C Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang. Automatic misconfiguration troubleshooting with peerpressure. In *OSDI*, volume 4, pages 245–257, 2004.

[55] Shu Wang, Chi Li, Henry Hoffmann, Shan Lu, William Sentosa, and Achmad Imam Kistijantoro. Understanding and auto-adjusting performance-sensitive configurations. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, volume 53, pages 154–168. ACM, 2018.

[56] Yi-Min Wang, Chad Verbowski, John Dunagan, Yu Chen, Helen J Wang, Chun Yuan, and Zheng Zhang. Strider: A black-box, state-based approach to change and configuration management and support. *Science of Computer Programming*, 53(2):143–164, 2004.

[57] Andrew Whitaker, Richard S Cox, and Steven D Gribble. Configuration debugging as search: Finding the needle in the haystack. In *OSDI*, volume 4, pages 6–6, 2004.

[58] Edmund Wong, Lei Zhang, Song Wang, Taiyue Liu, and Lin Tan. Dase: Document-assisted symbolic execution for improving automated software testing. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 620–631. IEEE Press, 2015.

[59] Chengcheng Xiang, Yudong Wu, Bingyu Shen, Mingyao Shen, Haochen Huang, Tianyin Xu, Yuanyuan Zhou, Cindy Moore, Xinxin Jin, and Tianwei Sheng. Towards continuous access control validation and forensics. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 113–129, 2019.

[60] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 307–319. ACM, 2015.

[61] Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. Early detection of configuration errors to reduce failure damage. In *OSDI*, pages 619–634, 2016.

[62] Tianyin Xu, Han Min Naing, Le Lu, and Yuanyuan Zhou. How do system administrators resolve access-denied issues in the real world? In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 348–361. ACM, 2017.

[63] Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. Do not blame users for misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 244–259. ACM, 2013.

[64] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N Bairavasundaram, and Shankar Pasupathy. An empirical study on configuration errors in commercial and open source systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 159–172. ACM, 2011.

[65] Chun Yuan, Ni Lao, Ji-Rong Wen, Jiwei Li, Zheng Zhang, Yi-Min Wang, and Wei-Ying Ma. Automated known problem diagnosis with event traces. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 375–388. ACM, 2006.

[66] Ding Yuan, Yinglian Xie, Rina Panigrahy, Junfeng Yang, Chad Verbowski, and Arunvijay Kumar. Context-based online configuration-error detection. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, pages 28–28. USENIX Association, 2011.

[67] Juan Zhai, Jianjun Huang, Shiqing Ma, Xiangyu Zhang, Lin Tan, Jianhua Zhao, and Feng Qin. Automatic model generation from documentation for java api functions. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 380–391. IEEE, 2016.

[68] Jiaqi Zhang, Lakshminarayanan Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou. Encore: Exploiting system environment and correlation information for misconfiguration detection. *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 42(1):687–700, 2014.

[69] Sai Zhang and Michael D Ernst. Automated diagnosis of software configuration errors. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 312–321. IEEE Press, 2013.

[70] Hao Zhong, Lu Zhang, Tao Xie, and Hong Mei. Inferring resource specifications from natural language api documentation. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 307–318. IEEE Computer Society, 2009.

[71] Yu Zhou, Ruihang Gu, Taolue Chen, Zhiqiu Huang, Sebastiano Panichella, and Harald Gall. Analyzing apis documentation and code to detect directive defects. In *Proceedings of the 39th International Conference on Software Engineering*, pages 27–37. IEEE Press, 2017.