ConfigFile++: Automatic Comment Enhancement for Misconfiguration Prevention

Yuanliang Zhang*, Shanshan Li*, Xiangyang Xu*, Xiangke Liao*, Shazhou Yang*, Yun Xiong†
*College of Computer Science, National University of Defense Technology, China
†Shanghai Key Lab of Data Science, School of Computer Science, Fudan University, China
{zhangyuanliang13,shanshanli,xuxiangyang11,xkliao,yangshazhou} @nudt.edu.cn, yunx@fudan.edu.cn

***Abstract*—Nowadays, misconfiguration has become one of the key factors leading to system problems. Most current research on the topic explores misconfiguration diagnosis, but is less concerned with educating users about how to configure correctly in order to prevent misconfiguration before it happens. In this paper, we manually study 22 open source software projects and summarize several observations on the comments of their configuration files, most of which lack sufficient information and are poorly formatted. Based on these observations and the general process of misconfiguration diagnosis, we design and implement a tool called ConfigFile++ that automatically enhances the comment in configuration files. By using name-based analysis and machine learning, ConfigFile++ extracts guiding information about the configuration option from the user manual and source code, and inserts it into the configuration files. The format of insert comment is also designed to make enhanced comments concise and clear. We use real-world examples of misconfigurations to evaluate our tool. The results show that ConfigFile++ can prevent 33 out of 50 misconfigurations.**

***Index Terms*—misconfiguration prevention; comment enhancement; constraint extraction**

I. Introduction

In recent years, customizable and flexible requirements lead to a high degree of software configurability. To meet these needs, software developers provide users with a large amount of configuration options. These options help users change the software behavior according to their own requirements; on the other hand, however, this can easily lead to misconfigurations, which has been a major cause of software failure. [1]

Studies on misconfiguration today usually focus on two approaches: misconfiguration diagnosis and misconfiguration prevention. A substantial amount of prior research into misconfiguration diagnosis has made contributions in the areas of troubleshooting and error diagnosis [2][3], which helps system administrators or developers by identifying the root cause of failure. However, misconfiguration diagnosis always takes place after the failure has already been manifested, meaning that the software system has already suffered great damage due to misconfigurations by the time the diagnosis takes place.

Misconfiguration prevention, by contrast, is far more effective at protecting the software system from failure damage and downtime, thus reducing the support costs. Several studies have been done in this field. Spex [4] studied configuration constraints and used program analysis to identify constraint violations. Xu et al. [5] analyzed the source code and checked for errors during initialization time. However, they pass these problems to the developers and ask them to handle it. In our opinion, it is unlikely that software bugs of this kind were introduced by developers; rather, misconfigurations are usually caused by users' inappropriate settings of configuration options. If we can directly and efficiently guiding users in setting these options, many future misconfigurations can be avoided, and developers will be freed from handling these complex configuration problems. In fact, developers have long since thought of this. Current mature software projects always provide authoritative guidance for users in the form of a user manual. However, user manuals are often long and complex and have no attraction to users. When experiencing confusion while setting configuration options, users rarely have the patience to find answers in the user manual but instead choose to rely on their existing experience and knowledge. This can easily result in misconfigurations. To help users with configuration, we endeavor to ensure that they can easily and directly get guiding messages. Configuration file is an important part of software configuration. After a thorough study of the configuration files of 22 widely used software systems, we find that information provided in the comments in these files is insufficient, and the format is also chaotic. Hence, our aim is to enhance the comments in configuration files in order to guide users through configuration.

Our contributions are as follows:

- By manually analyzing the configuration files of 22 popular open-source software systems, we summarize some of the characteristics of the comments currently in those files. Aiming at preventing defects, we here discuss the content and format of the high-quality comments.
- We design and present ConfigFile++, a tool for efficiently extracting and generating information about configuration options from existing resources (e.g. user manuals, original comments and source code). ConfigFile++ combines the relevant information into well-formatted comments and inserts them into the original configuration file.
- We use real-world error cases to evaluate the effectiveness of ConfigFile++. ConfigFile++ can prevent 33 out of 50 misconfigurations.

II. Study of Comments in Configuration Files

We first studied the configuration files of 22 widely used software projects such as Httpd, MySQL, Redis, and ProFTPd, etc. We counted the average **line number of comments**

TABLE I
CONFIGURATION FILE COMMENT QUALITY

Software	LNoC	Well-formatted	Effective Guidance	Comment quality
Cassandra	5.3	✓	✓	well-commented
Maven	5.7	✓	✓	well-commented
Redis	10.1	✓	✓	well-commented
Squid	15.3	✓	✓	well-commented
Tomcat	10.3	✓	✓	well-commented
PHP	3.9	✓	×	commented
Httpd	3.7	✓	×	commented
Jmeter	1.9	✓	×	commented
Sphinx	3.0	✓	×	commented
Hbase	1.9	✓	×	commented
Vsftpd	3.4	×	✓	commented
Samba	3.2	×	✓	commented
Derby	1.5	×	×	commented
PostgreSQL	1.9	×	×	commented
ProFTPd	1.5	×	×	commented
MySQL	1.9	×	×	commented
Log4j	1.6	×	×	commented
HAProxy	0.0	-	-	non-commented
Yum	0.0	-	-	non-commented
SSH	0.0	-	-	non-commented
Nginx	0.0	-	-	non-commented
OpenLDAP	0.0	-	-	non-commented

```
/*PostgreSQL.conf*/
listen_addresses = 'localhost'          # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
                                        # (change requires restart)
port = 5432                             # (change requires restart)
max_connections = 100                   # (change requires restart)
superuser_reserved_connections = 3      # (change requires restart)
unix_socket_directories = '/var/run/postgresql'  # comma-separated list of directories
                                        # (change requires restart)
unix_socket_group = ''                  # (change requires restart)
unix_socket_permissions = 0777          # begin with 0 to use octal notation
                                        # (change requires restart)
```

Fig. 1. Bad format of comment in configuration file

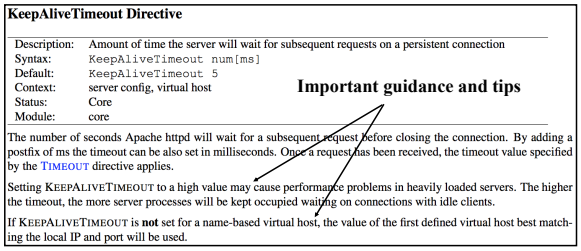
(**LNoC**) for a configuration option. We also evaluated the content and format of the comments in these configuration files. We classified these configuration files into three levels: non-commented, commented and well-commented. For a configuration file to be considered well-commented, the comments should also be well-formatted and contain effective guidance. Table I illustrates the details of these configuration files. We conclude two observations, as follows:

1) Many Configuration Files Do Not Have a Fixed and Canonical Comment Format: Fig. 1 provides an example of the poor formatting of comment in configuration files. Unlike source code, comments are written in natural language, meaning that they are more flexible and informal. However, a fixed and canonical format for comments is rather important in today's large-scale software systems. If the comments are not well-formatted, it will be a hard work for users to find the information they need in complex configuration files.

2) The Information Provided in Comments Is Ineffective at Guiding Users Compares to User Manual: We can see from Table I that softwares have divergent **LNoC**, and some even do not have comments. We also look into the content of these comments. After comparing these comments with guidance provided in user manual, we found that information provided by comments is insufficient. Fig. 2 shows us the substantial difference between comments in configuration files and user manual. In Fig. 2(a), we can see that there is only one sentence, presenting a brief introduction of "KeepAliveTimeout". However, the user manual in Fig. 2(b) gives users much more

```
/*Apache2.conf(Httpd)*/
#
# KeepAliveTimeout: Number of seconds to wait for the next
# request from the same client on the same connection.
#
KeepAliveTimeout = 5
```

(a) Comment in configuration file



(b) Information in user manual

Fig. 2. Information provided in configuration file and user manual

information about this configuration option. If a user does not see the tips in user manual and sets this option to a very high value, it may cause terrible performance problems and induce damage.

III. ARCHITECTURE OF CONFIGFILE++

Based on the observations in section II, we design a comment enhancement tool, ConfigFile++, that guides users through modifying the configuration files. Fig. 3 illustrates the architecture of our tool. The main usage of this tool is to extract sufficient guiding information about the configuration options and insert them into configuration file so that users can avoid inappropriate settings. Firstly, the value of a configuration option has a strong connection with its type. For example, you can only use integers that are between 1 to 65535 when setting a "port". Also, people sometimes set options with wrong types, such as setting a directory path with a file path. We infer option type using Type inference to help users avoid these misconfigurations. Besides, the option value also has some other constraints. For example, a memory-related configuration option has its own specific value range that users may not know. Such information is usually written in the user manual by the developers, and will be extracted by Guidance extractor. We want users to have a general understanding of an option so that they can quickly locate which option they need to change in future. To meet this need, we use a Key Usage extractor to briefly introduce the option usage in the form of keywords. To automate our tool, we use Augeas [6], a configuration editing tool, to help us with file preprocessing in the File parser. In the last stage, we insert combined information to the original configuration file.

IV. DESIGN AND IMPLEMENTATION

A. Type Inference

We manually study more than 2000 configuration options across several open-source software packages. Fig. 4 shows our classification tree for configuration options. The experimental results show that the coverage of this classification is

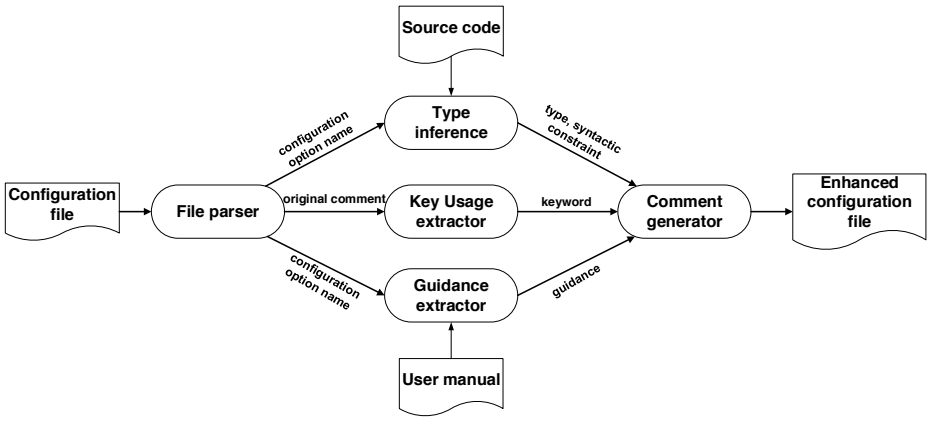


Fig. 3. The architecture of ConfigFile++.

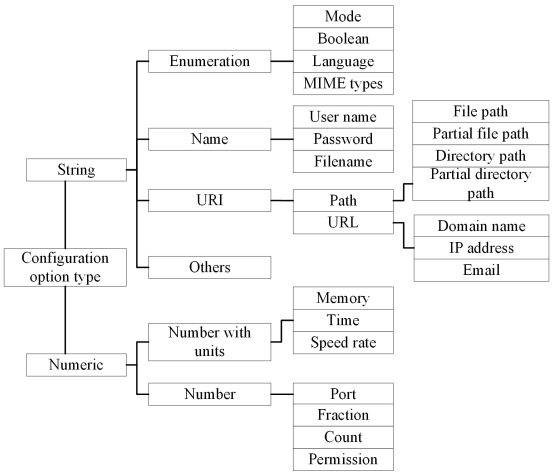


Fig. 4. Classification of configuration options

TABLE II
SYNTACTIC CONSTRAINTS OF CONFIGURATION OPTIONS

Type	Syntactic Constraints		
	Pattern	Element	Format of Element
File Path	(/%S)+	S	[\w.-]+
Directory Path	(/%S)+/?	S	[\w.-]+
Partial File Path	%S(/%S)*	S	[\w.-]+
Partial Directory Path	%S(/%S)*/?	S	[\w.-]+
Domain Name	[%S_1]?://%S_2	S_1	(telnet|https|http|ftp)
		S_2	[a-zA-Z0-9.]+
IP Address	%D_1.%D_2.%D_3.%D_4	D_{1-4}	[0-255]
Email	%S_1@%S_2	S_1	(\w)+(\.\w+)*
		S_2	(\w)+((\.\w+)+)
Mode	%S	S	(value1|value2|value3)
Boolean	%S	S	(on|off|yes|no|true|false)
Language	%S	S	[a-zA-Z]{2}
MIME-types	%S	S	[\w/-.]+
Memory	%D %S	S	(KM|G|T|KB|MB|GB|TB|B)
		D	[min-max]
Time	%D %S	S	(s|min|h|d|ms)
		D	[min-max]
Speed Rate	%D %S	S	(bps|Kbps|Mbps)
		D	[min-max]
Count	%D	D	[min-max]
Fraction	%F	F	[min-max]
Port	%D	D	[1-65535]
Permission	%O	O	[0-777]
Username	%S	S	[a-zA-Z][a-zA-Z0-9]*
Password	%S	S	N/A
Filename	%S	S	[\w -]+.[\w -]+

as high as 90% on average, with a minimum of 85.1%. During our study, we find that the names of configuration options chosen by software developers always have some sort of naming convention which is relevant to their type. We devised a name-based method to automatically infer configuration option type by extracting the semantic information from option names [7]. We use a score model to extract the semantic information in the configuration option name and thus infer the most likely type of an option. However, name-based analysis behaves poorly in inferring *enumeration* type, since there are too many words that can potentially express *enumeration* type and form the name.. Program analysis of the source code can be used to solve this problem, while the assignment of these options can be located by means of abstract syntax tree analysis.

B. Constraint Extraction

In our tool, we have two sources to help ensure that the extracted constraints are comprehensive and complete:

1) Syntactic Constraints Inferred by Option Type: By inferring the type of a configuration option, we can further obtain its syntactic constraints. We study the syntactic constraints of specific configuration option type [8]. Table II illustrates the details that we use to express syntactic constraints in normal forms. We use wildcard to represent the elements in normal forms. The general rule is concluded after extensive research and experimentation. For example, an IP address must consist of four integers between 0 and 255, divided by '.' . If a user sets one of the integers to be larger than 255, then the IP Address is invalid.

2) Guidance Extracted from User Manual: We also obtain semantic constraints from user documentation. After manually researching several user manuals, we summarize the following two situations, as shown in Fig. 5. We do not explicitly divide these two forms of statements when extracting because they both play important roles in guiding users configuring.

Fig. 6 shows the process of the Guidance extractor which is divided into three main stages: feature selection, model training and guidance extraction. First we need to consider

Configuration option: KeepAliveTimeOut **/*Httpd*/**
Statement:
In no event should you raise this above about 60 seconds, as most of the benefits are lost.
Constraint:
KeepAliveTimeOut < 60 seconds

(a) Direct form guidance

Configuration option: thread_stack **/*MySQL*/**
Statement:
If you increase the value of max_sp_recursion_depth, it may be necessary to increase thread stack size by increasing the value of thread_stack at server startup.
Constraint:
thread_stack has relevance with max_sp_recursion_depth

(b) Indirect form guidance

Fig. 5. Guidance in user manual

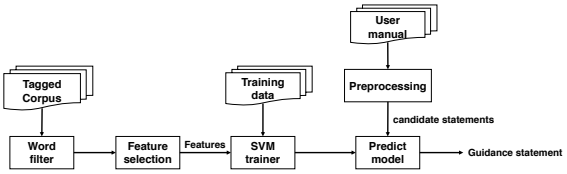


Fig. 6. Process of Guidance Extractor.

a statement as a vector of terms, which is called Vector space representation. Since there can be so many different words in user manual, the term-elimination techniques are required. Chi-square test [9] is than used to select the feature(keywords) of guidance. We manually labeled 1,000 statements from several software user manuals as our experimental data. Before using Chi-square test, we try to filter out noises, and POS tagging [10] help us to do semantic role labeling(SRL) to every word in a statement. Fig 7 shows an example of POS tagging techniques. We than discards some words with unconcerned tagging such as CD (Cardinal number), DT (Determiner) that may prevent meaningful features from being mined. Specifically, some useless words become meaningful when combined with another word, for example, be and must. We call them imperative phrases and we consider these situations individually. Word variants is also considered, words like recommend, recommends, recommended are all reduced to their bases: recommend. After preprocessing, we finally choose 20 words(phrases) to be the feature. Table III gives the count of every word tag. We use LIBSVM [11] to train the data set, then test the validity of the trained model by using 4-fold cross validation. The precision and recall are 0.944 and 0.896 respectively. We define a threshold of occurrence N (the default is set to 3). If an option name appears more than N times in a paragraph, we believe that this paragraph has a strong relevance to this configuration option, so we classify every statement in this paragraph using our model. If not, we deal only with the statements that contain the option name, rather than the whole paragraph.

Our approach has two main benefits. First, the user manual always mentions the most error-prone configuration situations to users. Second, statements in the user manual are always written in natural language, making them easy to read and understand. Our feature selection does not target any individual software, and if further training is required, this model can be expanded.

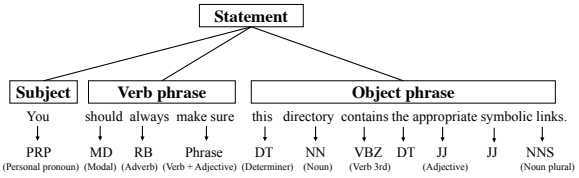


Fig. 7. Example of Postag

TABLE III
TAGS COUNT IN FEATURES.

Tag	Number	Example
PRP	1	”you”
RB	1	”only”
VB	6	”permit”
MD	2	”must”
JJ	2	”important”
IN	1	”unless”
Phrase	7	”should be”
Total	20	-

C. Key Usage Extraction

Most of the existing comments in configuration files describe the usage of options. Fig. 8 is an original comment snippet in redis.conf. We can use simple keywords to show the key usage of every option, and this will be of great value for users, who can then quickly locate the option they intend to change, rather than expending unnecessary manual effort reading every comment. We use TextRank [12], a graph-based ranking model for text processing to extract keywords from the comments.

D. Comment Design

We find that comments in many configuration files do not have a fixed or canonical format. Most of the current comments are written in natural language, we add regular expressions, wildcards and keyword phrases to our comment, which helps the comment to be more rigorous and concise without losing its readability. Fig. 9 illustrates an example of our design. Type refers to the type of configuration option. Key Usage helps users further understand the option. The syntactic constraints are written as regular expressions and wildcards to

```
# Set the replication backlog size. The backlog is a buffer that accumulates
# slave data when slaves are disconnected for some time, so that when a slave
# wants to reconnect again, often a full resync is not needed, but a partial
# resync is enough, just passing the portion of data the slave missed while
# disconnected.
# The biggest the replication backlog, the longer the time the slave can be
# disconnected and later be able to perform a partial resynchronization.
# The backlog is only allocated once there is at least a slave connected.
repl-backlog-size = 1mb
```

Keyword: ***replication backlog size, slave data, partial resynchronization***

Fig. 8. Example of Key usage

```
##########################################################################
#Type:              Memory
#Key Usage:         query cache, memory allocated
#Pattern:           %D%S
#Element format:    D = [min-max] , S = (K|M|G|T|B|KB|MB|GB|TB)
#Default:           16M
#Note:              If you set the value of query_cache_size too small, a warning will
#                   occur, as described in Section 9.10.3.3, "Query Cache Configuration".
#                   To disable the query cache at server startup, set the query_cache_size
#                   system variable to 0.
#                   When you set query_cache_size to a nonzero value, keep in mind that
#                   the query cache needs a minimum size of about 40KB to allocate its
#                   structures.
#                   If query_cache_size is 0, you should also set query_cache_type variable
#                   to 0.
##########################################################################
query_cache_size       = 16M
```

Fig. 9. New designed comment

TABLE IV
MISCONFIGURATION CASES OF 6 SOFTWARE

Software	Misconfiguration	Preventable
MySQL	14	10
PostgreSQL	7	4
Httpd	7	5
Php	10	7
HAProxy	8	4
OpenLDAP	4	3
Total	50	33

ensure that there is no ambiguity. The default value is also provided as a reference value if users wrongly modify the option. Other constraints extracted from the user manual are kept in natural language.

V. EVALUATION

To evaluate the effectiveness of our tool, we use real-world cases collected from StackOverflow [13] and ServerFault [14]. The experiments are carried out using six widely used software projects: MySQL, PostgreSQL, Httpd, PHP, HAProxy, and OpenLDAP. First we find issues of misconfigurations related to configuration files and record the answers (i.e. causes and solutions posted by other users) that have been confirmed by the questioner. We than use our tool to enhance the configuration file and we regard a preventable error as a situation that our enhanced comment can provide sufficient information to help user avoid the mistake.

A. Overall Result

We looked through more than 300 examples of misconfigurations on the two websites and selected 50 of them that related to configuration files. Table IV illustrates that our enhanced comments are sufficient to prevent 33 of these 50 misconfigurations.

B. Effective Prevention

Fig. 10 shows two real-world examples of errors that can be prevented by type information and syntactic constraint information. In Fig. 10(a), "extension_dir" is set to a file path instead of directory path, which results in database disconnection. The comment we add advises users the right type of option to use, enabling users to avoid this simple mistake. In Fig. 10(b), a user got error message because he missed the first slash in the directory name. Our enhanced comment tells users that this configuration option should be in the form "(/%S)+". This regular expression emphasizes the presence of the first slash.

Misconfiguration A: **/*PHP*/**
extension_dir = a file path
Symptom:
Database disconnection
Cause:
The path should be a directory path
Prevention Information:
…
#Type: String->Directory path
…

(a) Wrong Type Violation Prevention

Misconfiguration B: **/*OpenLDAP*/**
directory = "var/lib/ldap"
Symptom:
Invalid path: Permission denied
Cause:
Missing the first "/" in the directory name
Prevention Information:
…
#Format: (/%S)+/?
…

(b) Syntactic Violation Prevention

Fig. 10. Type and syntactic violation prevention

Misconfiguration C: **/*MySQL*/**
max_allowed_packet = 4M
Symptom:
Error: Throw an exception when sending a file
Cause:
The size of the packet exceeds the value of "max_allowed_packet"
Prevention Information:
…
#Note: When a MySQL client or the mysqld server receives a packet bigger than
max_allowed_packet bytes, it issues an ER_NET_PACKET_TOO_LARGE
error and closes the connection.
Both the client and the server have their own max_allowed_packet variable,
so if you want to handle big packets, you must increase this variable both in
the client and in the server.
…

Fig. 11. Constraint violation prevention

Fig. 11 gives an example of a misconfiguration that can be prevented by guidance extracted from the user manual. In this case, the MySQL client throws an exception when a user sends a file. This happens because that the size of the file exceeds the value of 'max_allowed_packet". There are two statements extracted by ConfigFile++ that can help users avoid this error: (1) "When a MySQL client or the mysqld server receives a packet bigger than max_allowed_packet bytes, it issues an ER_NET_PACKET_TOO_LARGE error and closes the connection." (2) "Both the client and the server have their own max_allowed_packet variable, so if you want to handle big packets, you must increase this variable both in the client and in the server." [15].

C. Defects and Difficulties

There are still some situations that are difficult to prevent using ConfigFile++, which can be our future work. V gives the result.

TABLE V
MISCONFIGURATIONS FAILED TO BE PREVENTED

Failure cause	Number
Cross-stack configuration	5
Software Evolution	3
User operation	1
Operating System related	2
Others	6
Total	17

1) Cross-Stack Configurations: We do not consider the cross-stack misconfigurations because all the information we inserted comes from a single software's resource. For example, a user sets "max_execution_time" in PHP to be longer than "net_read_timeout and "net_write_timeout in MySQL, with the result that MySQL disconnects. This requires analysis of the specific software combinations, which can be a direction for our future work.

2) Software Evolution: A user reports that an error was detected while parsing a switching rule in configuration file: /etc/haproxy/haproxy.cfg. The problem turns out to be that Ubuntu's repos contain a rather old (1.4) version of HAProxy, and the example uses some directives that were added in version 1.5.

3) User Operation: In some cases, configuration can fail if users do not follow the correct operation steps. For instance, some users change the configuration files, but the changes do not take effect until after the system is restarted.

4) System Related: Some errors are related to the operating system and are thus hard to prevent. For example, there is an error occurred in Httpd because SELinux only allows the web server to make outbound connections to a limited set of ports.

VI. RELATED WORK

In our study, we combine configuration option type inference and guidance extraction from user manuals to help users avoid misconfigurations. When inferring the type of configuration options, Rabkin and Katz [16] used static extraction to find well-defined APIs which read and pass the options, using them to infer the option type. In our work, we use name-based analysis to infer the type, which is more convenient and requires less human efforts. Our study also involves mining some information from user manuals. Some previous studies have made substantial efforts to use software documentation. API documentation was studied in [17]. However, it focused on the taxonomy of parameter constraints; rather than applying automated techniques. Other works focused on comment quality analysis include [18][19][20]. These works differ from ours because they target comment quality in source code while we try to enhance the comments in the configuration file to help users with configuring.

VII. CONCLUSIONS

Misconfigurations have become a notable problem in today's software systems. In this paper, we aim at misconfiguration prevention by enhancing the comments in configuration files. We manually study the configuration files of 22 software systems and find the defects in them. Based on these defects and our previous research, we design and implement ConfigFile++ to automatically extract information about configuration options and insert it into configuration files. We use real-world examples of misconfigurations to evaluate the effectiveness of our tool. The results show that our tool is effective at helping users avoid misconfigurations.

ACKNOWLEDGMENT

This paper is supported by National Key R&D Program of China 2017YFB1001802.

REFERENCES

[1] A. Rabkin and R. Katz, "How hadoop clusters break," *IEEE Software*, vol. 30, no. 4, pp. 88–94, 2013.
[2] M. Attariyan and J. Flinn, "Automating configuration troubleshooting with dynamic information flow analysis," in *Usenix Conference on Operating Systems Design and Implementation*, 2010, pp. 1–11.
[3] M. Attariyan, M. Chow, and J. Flinn, "X-ray: automating root-cause diagnosis of performance anomalies in production software," in *Usenix Conference on Operating Systems Design and Implementation*, 2012, pp. 307–320.
[4] T. Xu, J. Zhang, P. Huang, J. Zheng, T. Sheng, D. Yuan, Y. Zhou, and S. Pasupathy, "Do not blame users for misconfigurations," in *Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 244–259.
[5] T. Xu, X. Jin, P. Huang, Y. Zhou, S. Lu, L. Jin, and S. Pasupathy, "Early detection of configuration errors to reduce failure damage," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 619–634. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026925
[6] Augeas, http://augeas.net/.
[7] X. Xu, S. Li, Y. Guo, W. Dong, W. Li, and X. Liao, "Automatic type inference for proactive misconfiguration prevention," in *The International Conference on Software Engineering and Knowledge Engineering*, 2017, pp. 295–300.
[8] W. Li, S. Li, X. Liao, X. Xu, S. Zhou, and Z. Jia, "Conftest: Generating comprehensive misconfiguration for system reaction ability evaluation," in *The International Conference*, 2017, pp. 88–97.
[9] M. D. Franzen, *Chi-square*. Springer New York, 2011.
[10] L. Rquez, Llu, Rodr, and H. Guez, "A machine learning approach to pos tagging." *Machine Learning*, vol. 39, no. 1, pp. 59–91, 2000.
[11] C. C. Chang and C. J. Lin, "Libsvm: A library for support vector machines," *Acm Transactions on Intelligent Systems Technology*, vol. 2, no. 3, pp. 1–27, 2011.
[12] R. Mihalcea and P. Tarau, "Textrank: Bringing order into texts," in *Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A Meeting of Sigdat, A Special Interest Group of the Acl, Held in Conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*, 2004, pp. 404–411.
[13] StackOverflow, https://stackoverflow.com/.
[14] ServerFault, https://serverfault.com/.
[15] *MySQL 5.7 Reference Manual*, 2016.
[16] A. Rabkin and R. Katz, "Static extraction of program configuration options," in *International Conference on Software Engineering*, 2011, pp. 131–140.
[17] L. Shi, H. Zhong, T. Xie, and M. Li, "An empirical study on evolution of api documentation," in *International Conference on Fundamental Approaches To Software Engineering*, 2011, pp. 416–431.
[18] L. Tan, D. Yuan, G. Krishna, and Y. Zhou, "/*icomment:bugs or bad comments?*/," 2007, pp. 145–158.
[19] S. H. Tan, D. Marinov, L. Tan, and G. T. Leavens, "@tcomment: Testing javadoc comments to detect comment-code inconsistencies," in *IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 260–269.
[20] D. Steidl, B. Hummel, and E. Juergens, "Quality analysis of source code comments," vol. 33, no. 2, pp. 83–92, 2013.