Hindawi
Scientific Programming
Volume 2022, Article ID 4415366, 13 pages
https://doi.org/10.1155/2022/4415366

Research Article

A Real-Time Detection Method of Software Configuration Errors Based on Fine-Grained Configuration Item Types

Li Zhang,[1] **Shengang Hao**,[2] **and Meng Ming**[2]

[1]*School of Media Engineering, Communication University of Zhejiang, Hangzhou 310018, China*
[2]*School of Computer Science, Beijing Institute of Technology, Beijing 100081, China*

Correspondence should be addressed to Li Zhang; nythhsg@163.com

Received 24 November 2021; Accepted 6 January 2022; Published 22 February 2022

Academic Editor: Rahman Ali

With the continuous expansion of software scale and the continuous complexity of software functions, abnormal parameter configuration often brings adverse effects to the software system and even leads to system failure. Therefore, a method is needed to detect whether the parameter configuration is correct. Most of the current configuration errors detection methods focus on the passive diagnosis after the system failure, which cannot solve the potential delay problem of configuration error. This paper proposes an automatic real-time detection method of software configuration errors, in which the configuration items are classified based on the fine-grained configuration item types and related syntax patterns, and the configuration constraint rule base is generated. Then, the real-time exception detection of configuration update operation is realized by using the file operation monitoring function. The experimental results show that this method can accurately classify the types of configuration items and verify the effectiveness of detecting software configuration update exceptions in real time through the constraint rule base. The classification accuracy reaches up to 90.4% on MySQL and 87.4% on Apache.

1. Introduction

In recent years, with the development of computer technology and the continuous improvement of computer performance, the complexity of user software is becoming higher and higher. The abundant number of configuration items is one of the manifestations of software complexity. At present, the number of configuration items of large-scale open source software such as Apache, Hadoop, and MySQL has reached hundreds, and the number is increasing year by year [1, 2]. By 2018, the number of MySQL configuration items has reached 700. It is extremely difficult for users to fully master complex configuration rules and correctly configure a large number of parameters.

However, the impact of incorrect configuration on the system is often fatal. According to relevant research, more than 50% of the failures related to system operation and maintenance management come from configuration errors [3]. More than 80% of network failures are caused by configuration errors [4]. Configuration errors are difficult to avoid, and large application systems of well-known enterprises such as Amazon EC2 [5], Facebook [6], and Microsoft azure [7] have experienced service interruption caused by configuration errors, resulting in significant losses. Moreover, once these system failures occur in the fields such as unmanned driving, it may cause disastrous consequences [8]. Therefore, detecting configuration errors is one of the important directions of software reliability research [9].

Most of the traditional methods to detect configuration errors are manually eliminated by the maintenance personnel. However, the increasing number of configuration items and complex setting rules makes the manual troubleshooting gradually infeasible, and the reliability of manual operation is difficult to be guaranteed [10]. Most of the existing research work on configuration errors detection focuses on the passive diagnosis after fault [11], but it cannot solve the potential delay problem of configuration error. A small part of the work studies the active prevention before the occurrence of configuration errors. Such methods can do error detecting before the system runs, but their real-time

performance is poor, and they cannot solve the unavailability of configuration files caused by the accidental deletion or the format damage.

Therefore, this paper proposes an automatic method that can monitor and detect whether there are errors in the updates of software configuration parameters in real time. When the detection passes, the software system continues to run under the protection of the predesigned configuration data rules. When the detection fails, the configuration recovery module immediately started to change the modified abnormal configuration into the initial normal configuration, so as to ensure the stability of the software system and to avoid the service interruption caused by abnormal configuration.

The contributions of this paper are as follows:

(1) We propose a real-time configuration error detection method, which can realize the fine-grained classification of the configuration items by analyzing their syntax structure, the semantic information in their names, and the system environment information.

(2) We build the configuration constraint rule base to monitor and protect the software configuration file in real time, and to actively detect whether there are exceptions when the configuration update operation occurs.

(3) Our method does not need to analyze the source code of the software system, so it has good scalability and compatibility. The whole detection process of configuration errors is fully automated, without user intervention, so it has good stability and higher efficiency.

The rest of this paper is organized as follows: Section 2 briefly introduces the background and related work. Section 3 discusses the design principle and the overall architecture and describes the implementation details of our method. Section 4 gives the experimental results and analysis, and section 5 is the conclusions.

2. Background and Related Work

2.1. Configuration Error. Software configuration management (SCM) is an auxiliary software development tool. It includes a series of technology of identification, organization, and control modification with more organized logical structure. The entity unit in software configuration management is the software configuration item (CI), including program, data structure, and document [12]. Software configuration error refers to the situation that the deterministic failure is caused by the wrong parameter setting of software configuration items, which affects the operation of the system (also called the misconfiguration) [13]. According to relevant research [14], common configuration errors are mainly reflected in three aspects: parameter value, compatibility, and component placement position. Among them, parameter value errors account for the largest proportion in actual situations. Such errors include the single configuration item value error and the multiple configuration item value errors.

2.2. Configuration Error Detection. At present, software configuration error detection methods can be mainly divided into four types. They are program analysis, statistical learning, comparison, and replay technology [11]. According to their dependence on the target system in the detection process, these methods can be divided into the white box detection and the black box detection, as shown in Figure 1.

The method based on program analysis belongs to the white box method. By analyzing the source code, bytecode, or binary code of the software system, the program execution logic statements can find that they are affected by the configuration parameters, so as to locate the configuration fault. The research productions in this field have a tool called ConfAid [15] to eliminate the configuration fault by analyzing the dynamic information flow, a detection tool called Conf_Analyzer that can analyze the static data flow of software [16], a detection tool called ConfDebugger [17] using the reverse static data flow analysis technology, and a configuration constraint reasoning tool SPEX [18].

The method based on statistical learning regards the software system as a black box; it mines rules from the historical data related to the system behavior, the system state and the system events with the technology of statistics and machine learning, and realizes the error diagnosis based on the rules violated by the system. For example, Zhang [19] first proposed that there may be a connection between the value of software configuration items and the operating environment, other configuration items in 2014. On this basis, the author designs Encore. It can mine the correlation constraints among the multiple configuration items and can learn the configuration rules iteratively from a given set of configuration data sets. Finally, it can actively detect whether the configuration of the software system is correct or not based on the rich constraint rules. In addition, there are some detection tools such as ConfTest [20] and ConfVD [21] that classify the configuration items based on the tree structure and that extract the configuration constraint rules according to the configuration item type.

The method based on comparison belongs to the black box method, which needs to treat a huge amount of the existing error information as the reference benchmark to establish the configuration error knowledge base in order to diagnose the configuration error. For example, Talwadker R implemented the detection tool called Dexer [22], in which he established a case knowledge base based on the system log in the correct configuration state. When an error occurs, the possible solutions can be found out from the knowledge base.

The method based on replay technology also belongs to the black box method, which sets the various configuration parameter values and observes the system behavior state over and over again to find out the potential configuration constraints so as to detect the exceptions and repair them. For example, the replay technology is applied to the tool called Triage [23] to capture the scene when the fault occurs and reproduce it, and then to track the program running path and the data flow by using the dynamic program analysis technology. Finally, the correct parameters are compared with the wrong parameters observed from the

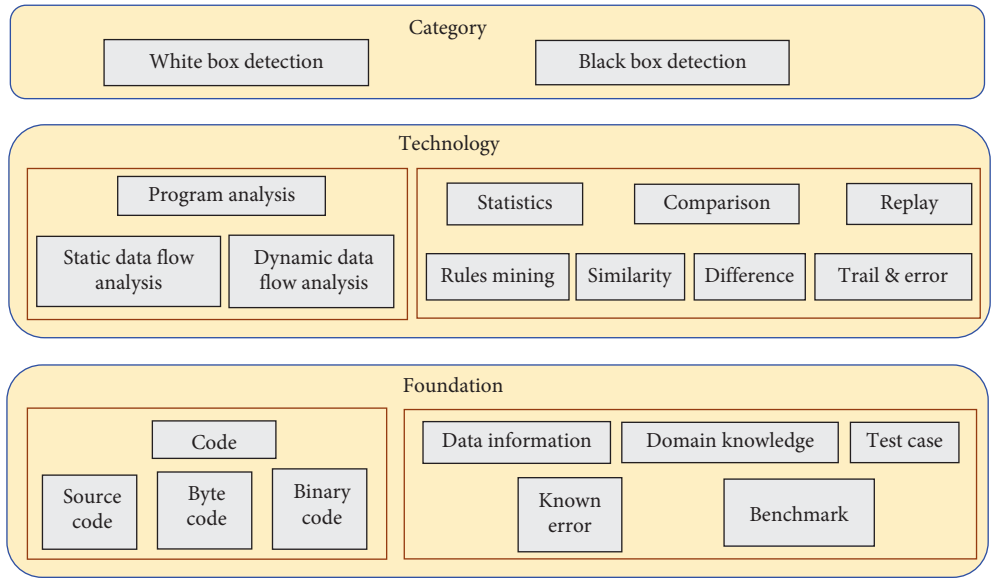

Figure 1: Configure error detection method [11].

repeated scene to locate the configuration errors and to find the possible repair methods. In addition, the method called Autobash [24] realizes the tracking and playback of the process state changes by rewriting the function of Linux kernel, and it can isolate other application software.

2.3. Configuration Type Classification. Xu et al. [25] investigated and analyzed more than 3000 configuration items of eight large open source software such as Apache, mysql, nginx, and PostgreSQL and found that the configuration item types contain rich semantic constraints, such as DocumentRoot in httpd.conf and in squid.conf_ who_ Take the two configuration items of server as an example, as shown in Table 1. The configuration item DocumentRoot is of filepath type, while as_ who_ Server is a URL. It can be judged that the configuration value set by the user is wrong according to the constraint rules of the corresponding type. Therefore, it is meaningful to set the constraint rules according to the configuration type to detect whether there are errors in the configuration items.

Rabkin [26] et al. have conducted the relevant research on the configuration type classification, but the classification method has the coarse granularity. Based on this work, Xu et al. proposed a fine-grained configuration type classification after the investigation and analysis of more than 3000 different software configuration items, which classified the software configuration items into 21 types. The author described the 21 types of configuration items by using a hierarchical tree structure. The closer it is to the upper layer, the coarser the classification granularity and the configuration item type it belongs to. For example, all configuration item types can be roughly classified into the string types and the digital types, and each upper type can be further subdivided into other lower types. This classification tree has good scalability.

The author reapplies the configuration type classification results to the eight investigated software for testing. The fine-grained configuration type classification structure can cover at least 85.1%, and the average coverage is more than 90%. The author also found from the survey that more than 91% of the configuration item names contain two (or more) words. The pattern of naming configuration items using hump naming method can reflect the type semantic information. For example, the configuration items related to the number of user connections in MySQL often contain keywords such as "connection," "Max," and "current." The configuration items representing the path usually contain keywords such as "path," "directory," "location," "dir." Therefore, the author proposes a type inference method based on configuration name analysis and designs and implements the tool called ConfTypeInferer.

Table 1: Examples of incorrect configuration [25].

Configuration file	httpd.conf	squid.conf
Configuration item	DocumentRoot	as_who_server
Type of Configuration item	Filepath	URL
Configuration constraint	/.+(/.+)*	[A-Z]+://*
User's parameter	Root	Server1
Correct parameter	/Document/root	http://aswho.server.com

It can be seen from the above that most of the work in the research based on program analysis, comparison, and replay technology belongs to the passive detection after configuration errors occur. Such methods cannot eliminate the negative impact of the configuration errors in time. However, the method based on statistical learning can actively detect the configuration errors of the software system based on the learned configuration rules and can be used for real-time detection of configuration errors, so it can quickly reduce the negative effect on the software system.

This paper proposes an online real-time detection method of software configuration errors based on the fine-

grained configuration type classification. It is not necessary to analyze the source code of the software system and to understand the specific semantics of the codes in our method, so it has good scalability and compatibility.

3. Design and Implementation

3.1. Design Principle. As shown in Table 1, configuration types contain a lot of semantic information about constraint rules. Such semantic constraints not only are effective for a single configuration item itself, but also reflect the constraint relationship between the configuration item and the running environment to some extent. For example, when the configuration parameter belongs to the URL type, you can judge whether the configuration value is correct according to the composition structure of the URL. When the two configuration parameters belong to the same path type, there may be path inclusion and ownership relationship. The memory type configuration parameters related to memory allocation need to meet the constraints of the available memory of the current system. The potential constraint relationship can be reflected through configuration types. Therefore, in this scheme, the constraint rule template is mainly formulated by exploring the constraint conditions of configuration types.

This scheme is designed based on the fine-grained configuration item types and the related syntax patterns proposed in literature [25] to classify configuration items and formulate syntax constraints. The difference is that literature [25] classifies configuration items through semantic information in configuration item names and program analysis, while this scheme realizes the classification of configuration items through the hybrid method of syntax structure, system environment information, and name analysis. At the same time, this scheme generates the configuration constraint rule base through type constraints and uses the constraint rule base to realize the real-time exception detection of configuration update operation in combination with the file operation monitoring function.

3.2. Architecture Design. The real-time detection of configuration errors always occurs when the configuration items are modified. This detection process is generally divided into two stages, as shown in Figure 2. The first stage is to generate the configuration constraint rules base. Therefore, it is necessary to extract the constraint rules from all kinds of the configuration files. The second stage is to detect the configuration errors when the configuration update operation is done. It is the prerequisite to monitor the software configuration dynamics in real time and to capture the access operation of the configuration file and to identify the modified configuration item.

In detail, the first stage of detection process can be divided into three steps. The first step is to collect and analyze the configuration files of the software, to extract the configuration items with the tool called Augeas [27], and then to convert them into key-value pairs. After that, in the second step, the type information of configuration items is mined by analyzing their syntax structure, the semantic information in their names, and the system environment information, and they are classified according to the fine-grained configuration item type lists in Figure 3. Finally, the resolved configuration items are written into the predefined template of type constraint to form the configuration constraint rule base.

The key problems to be solved in the whole detection process are as follows:

(1) Collect the configuration files of software and parse the configuration items into multiple key-value pairs

(2) Classify these key-value pairs of configuration items into the corresponding types based on the fine-grained type lists

(3) Define the constraint template of the configuration type and generate the configuration constraint rule base

(4) Monitor the configuration modification dynamics in real time and detect whether the configuration update operation break the constraint rules

3.3. Implementation

3.3.1. Key-Value Pairs of Configuration Item Extraction. In order to generate the constraint rule base of software configuration items, first, it is needed to obtain all configuration items of the software system and parse them into the key-value pairs. Due to the "everything is a file" feature of Linux system, the configuration items of application software are stored in the configuration file in a specific format, and the configuration is set and managed through the configuration file. The Linux system has the directory/etc. for uniformly storing and managing various configuration files. When the software is successfully installed, a directory named the software name will be generated under/etc., such as/etc./apache, which means that all configuration files related to the apache will be stored in this directory. Therefore, it is easy to gather the configuration files of the target software.

Generally, the configuration files of application software in Linux are encoded in ASCI, so the configuration items can be easily extracted only by parsing the file content. The storage structure of these configuration files in Linux generally has two forms. One is similar to that of the INI file, which is composed of multiple sections. Each section starts with the section name with square brackets, followed by the parameter name and its value. Take the configuration file of MySQL as an example; its file structure is shown in Figure 4. Each section starts with a section label, such as [mysqld], and the configuration item is represented by a key-value pair of parameter name = parameter value. "#" is a comment character. Except that, a semicolon may be a comment character too. The comment line that begins with "#" or " ;" will be ignored in the actual operation. Since the configuration items in this kind of configuration files are stored in the form of key-value pairs, they are easy to deal with.

Another format is similar to that of the XML file, such as the configuration files of Apache. For the configuration files

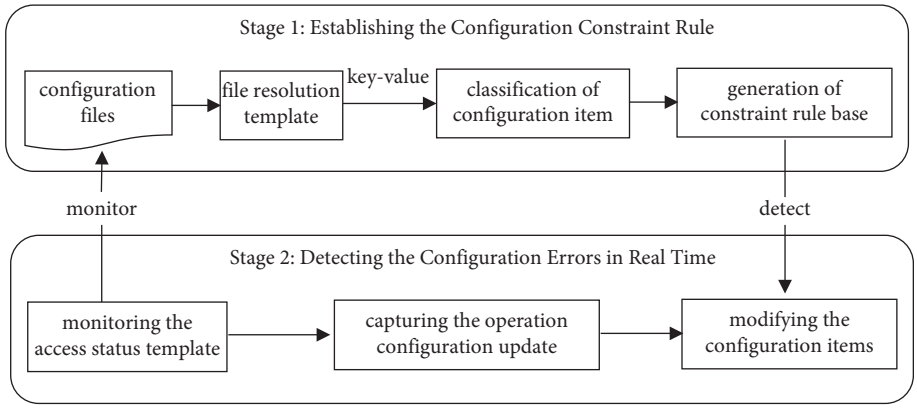

Figure 2: The architecture of configuration errors detection in real time.

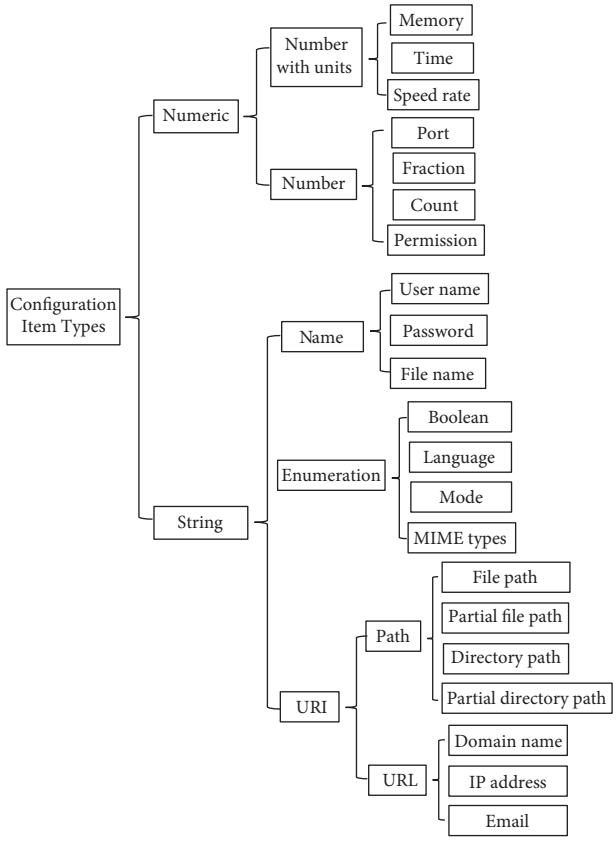

Figure 3: The fine-grained classification lists of configuration items [10].

in the XML format, the special parsing methods are needed to extract the names and values of configuration items. Augeas is one of the most widely used automatic analysis and management tools for the configuration files in Linux system. It can analyze the profiles of most software including Httpd, MySQL, Nginx, and PostgreSQL. It provides many interfaces of different programming languages, including python, ruby, and Java, and it is applied in the configuration management tools such as puppet and bcfg. Therefore, we can realize the analysis of configuration files in complex format based on Augeas.

Take the configuration file Apache as an example; there are two main components in it: section and directive. Section can be nested and contain multiple directives, and each directive is composed of a name and the corresponding arguments. Figure 5 shows a fragment of the configuration file named ports.conf. They include one section and two directives: IfModule is a section, and the two directives are two listen commands corresponding to two ports.

The parse tree transformed by Augeas is shown in Figure 6. It entirely reflects the hierarchical structure of the configuration file and distinguishes the configuration item name and its value. The configuration item in the form of key-value pairs can be extracted from the parse tree, as shown in Table 2.

```
[Mysqld]
character-set-server=utf8
#
# * Basic Settings
#
user        = mysql
pid-file    = /var/run/mysqld/mysqld.pid
socket      = /var/run/mysqld/mysqld.sock
port        = 3306
datadir     = /var/lib/mysql
```

Figure 4: The structure of MySQL configuration file.

```
Listen 80
<IfModule ssl_module>
        Listen 443
</IfModule>
```

Figure 5: The fragment of Apache configuration file.

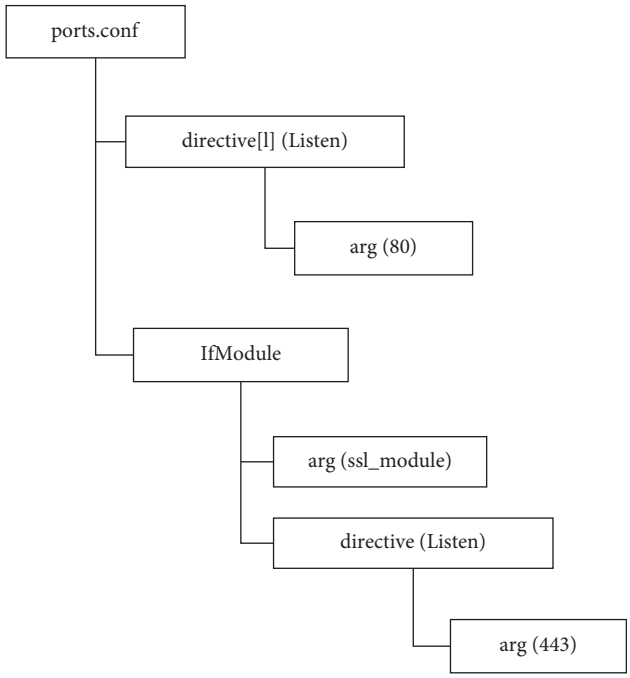

Figure 6: The parse tree of ports.conf generated by Augeas.

Table 2: The key-value pairs included in the parse tree.

Key	Value
/Directive [1]	Listen
/Directive [1]/arg	80
/IfModule/arg	ssl_module
/IfModule/directive	Listen
/IfModule/directive/arg	443

Augeas has built-in API based on C library. We can call the interface aug_load(augeas ∗aug) , and int aug_print(const augeas ∗aug, FILE ∗out, const char ∗path) to load a configuration file and to generate the parse tree.

The nodes in the configuration tree can be output to the specified location, and the configuration item information can be accurately extracted by transforming the configuration file into a configuration tree.

3.3.2. Configuration Item Classification. According to section 2.3, configuration item types contain semantic constraint information. Accurate type division of configuration items helps explore the constraints of the configuration. This section designs a configuration item classification method based on the mixed mode, in which both value and name of a configuration item are considered as the classification bases, combined with the system environment information. Finally, a configuration item can be classified into one of 21 types shown in Figure 3.

After collecting and parsing all configuration items of the software in Section 3.3.1, each configuration item is represented in the form of key-value pairs. In this section, the converted key-value pairs are used to classify the items. Firstly, the parameter value of the configuration item is considered. In the software system, the values of many configuration items have a fixed expression format. For this kind of parameter values with unified format, you can greatly reduce the type range of the configuration item and even directly determine the configuration type by simply making a rough type judgment through regular expression matching. For example, a configuration item containing "/" in value may be of type path.

In addition, some parameter values are expressed in global format, including IP address, domain name, and e-mail address Among them, IP address is generally expressed in dotted decimal notation. Domain names are generally in the form of "agreement +://+ specific address." The expression of e-mail address is usually "user name @ e-mail server name." Such configuration parameters can directly infer the configuration type through regular expression matching. In addition, the configuration parameter value with fixed expression has another format; that is, it has a specific flag, which is generally reflected by the unit suffix representing a certain attribute. For example, for the memory type of memory allocation, the allocated memory size will be specified during the actual configuration of this parameter, so the parameter value will be marked with "KB/MB/GB"; the configuration item of time type usually has the time unit "MS/S/min/h/D"; the unit suffix of speed rate type is usually "BPS/Kbps/Mbps."

Table 3 is the syntax pattern of each type that belongs to one of the 21 types of configuration items. Each configuration item can be preliminarily classified based on this table, and the syntax pattern of this table can be used as the syntax rule detection standard in the subsequent configuration exception detection process.

Through the preliminary regular matching of the values of the configuration key-value pair, the configuration item type can be inferred, or the type range can be reduced. When the type cannot be inferred directly, the system environment information can be introduced into the reduced type range for further classification. For example, if the syntax pattern

Table 3: Syntax patterns of different configuration item types.

Configuration item type	Syntax pattern
Memory	[\d]+[KB|MB|GB]
Time	[\d]+[ms|s|min|h|d]
Speed rate	[\d]+[bps|Kbps|Mbps]
Port	[\d]+
Fraction	[\d]+
Count	[\d]+
Permission	[\d]+
User name	[a-zA-Z][a-zA-Z0-9]∗
File name	[\w−]+[\w−]+
Boolean	[true|false|on|off|yes|no]
Language	[a-zA-Z]2
MIME types	[\w/-.]+
Path	/.+(/.+)∗
Domain name	[a-z]+://.∗
IP address	[\d]1,3(.[\d]1,3)3
E-mail	(\w)+(\.\w+)∗@(\w)+((\.\w+)+

matches that the parameter value of the configuration item belongs to the string type and contains "/", then the type of the configuration item can be located to path. Next, you can judge whether the configuration parameter is set to a file or directory and whether it is an absolute path according to the metadata information of the file system.

Some types of configuration item types cannot be distinguished by syntax mode and system environment information, such as port and count in configuration item type classification. The values of these two types of configuration parameters are expressed in digital form without any special flags that can be used to distinguish types. For such configuration items, the scheme further classifies them according to the configuration item name keywords on the basis of syntax pattern matching and system environment information verification. By observing the software configuration file and user configuration manual, developers usually use words or abbreviations containing relevant semantic information to name configuration items and use "" Separator or hump nomenclature connects multiple words or abbreviations. This scheme establishes a key thesaurus for each configuration type and assigns different frequency scores according to the occurrence frequency of keywords. For configuration items that meet multiple types, first segment the configuration item name according to the separator and capital letters, then calculate the similarity between the current configuration item name and each type of keywords, and classify them according to the type with the highest similarity.

It should be noted that, in the configuration item name composed of multiple keywords, the keywords that appear in the front are mainly used to modify the subsequent keywords. The later keywords are often more able to show the type semantics. Therefore, in the process of similarity calculation, different weights will be allocated according to the order in which the keywords appear in the configuration item name. The later the keywords, the greater the weight. The calculation formula is as follows:

$$S_t = \sum_{w_i}^{n} F_i * W_i, \quad (1)$$

where S_t is the similarity calculation score of configuration items in each type; n is the number of words obtained by word segmentation of configuration item name, that is, generally 2 or 3; w_i is the i*th* keyword after word segmentation; F_i is the frequency score of the i*th* keyword; w_i is the order weight of the i*th* keyword, and the value increases in order of word occurrence.

3.3.3. Generating the Configuration Constraint Rule Base. After the software configuration items are extracted and classified, the constraint rules will be generated in this section based on the predefined constraint templates. Firstly, the constraint conditions that these configuration item types need to meet are analyzed to formulate the constraint template. Software configuration error always is presented as the parameter value error, which includes the abnormal spelling, the format error, the datatype error, and the value range exception.

For the errors of datatype and format, you can easily judge whether the current value is correct, and it is needed to specify the syntax pattern that the corresponding type must meet in the constraint conditions. The syntax pattern is shown in Table 3. However, in practice, most of the configuration items do not have the fixed values or ranges, and their values are related with the current system execution environment and other configuration items. Therefore, when defining the configuration type constraint template, attention should also be paid to the relationship between the item type and the environmental information, other types in order to generate more comprehensive constraint rules. This section analyzes the constraint rules of software configuration item types from these three aspects, which are the syntax patterns, the system environmental constraints, and the associated constraints with other configuration item types. This section will focus on the last two aspects.

Firstly, the relationship between all configuration item types in Figure 3 and the system execution environment will be analyzed. The values of some parameters will be affected by the current system environment during the software running process. For example, we need to judge whether the parameter value meets the standard of allocable memory in the current system when the configuration item of memory is set. In addition, we need to check whether the parameter value is within the range of [0–65535] and whether the allocated port has been occupied by other services when the configuration item of port is set. For another example, when we set the configuration item of path, we need to check whether the current path exists, etc. By analyzing the above 21 type of configuration item, some constraints of each type can be extracted from the system execution environment, as shown in Table 4. The commands of obtaining system environment information related to each configuration item are also indicated in Table 4.

Except the above description, this scheme also considers the possible correlation constraints among the configuration items. As mentioned in section 2.3, the configuration item types contain rich semantic information. The semantic information can not only help users understand the values of

Table 4: System environment constraints met by different configuration item types.

configuration item type	System environment constraints	Command of obtaining system environment information
Memory	Less than the available memory	Free - m
Speed rate	Less than the upper bandwidth limit	ip a and ethtool [ethernet port]
Port	Not occupied by other services	Netstat - anp | grep [port]
User name	User exists	cat/etc/passwd | grep [user name]
File name	File name exists	Stat [file name]
Language	Available language types	ISO 639–1 [54]
MIME types	Available MIME types	IANA [55]
Path	Path exists	Stat (path)
Domain name	Domain name is available	Curl - I [domain name]
IP address	IP address is accessible	Ping [IP address]

configuration items, but also help infer the configuration dependency constraints. For example, for the configuration item of file path, it is assumed that the function of the configuration item is to output the data to the file specified by parameter value. If the parameter value contains the system paths, such as/boot,/lib,/bin, the insufficient permissions may occur. Therefore, the current user right should be considered when setting the path parameter.

In addition, for the configuration item belonging to "Number," especially for the item name with the keywords such as "Max," "Min," the value range of these parameters should be considered. Based on the above analysis, this section extracts the correlation constraint rules of some configuration items types that will be applied in the configuration error detection. These constraint rules are shown in Table 5.

The configuration type constraint template is constructed based on the configuration item value constraint, the constraint between the value and the system environment attribute, and the configuration type correlation constraint. The configuration items classified under each type are substituted into the corresponding type of constraint rules to generate the constraint rule base of software configuration items. The constraint rule base structure is as follows.

{"K1": "configuration item name," "K2": "configuration item name," "rule": "detailed rules," "value": "specific value"}

In the representation of the above constraint rules, for the syntax constraints of the type to which the configuration item belongs and the constraints of value and environment attributes, the configuration item name of "K2″" item uses "_ "Self" means that the "rule" item is the syntax constraint expression of this type or the command to obtain system environment information in Table 4. For configuration type dependency constraints, "K1" and "K2" are, respectively, the names of the two configuration items with dependency, and "rule" is the dependency constraint, such as "> =", "< =", "=". During detection, relevant constraint rules are obtained from the constraint rule library according to the name of the currently updated configuration item and then detected in turn.

3.3.4. Anomaly Real-Time Configuration Error Detection. In this scheme, the constraint rules base of the software configuration items generated in the above section is used to judge whether the current configuration update is abnormal. When the modification operation of the configuration file is monitored, the modified configuration item will be obtained, and the modification of the current configuration item will be judged whether it is against the constraint rule according to the corresponding constraint rule record in the rule base.

In this paper, the listener in the user layer will not only send the information of the modification operation to the backup and control module, but also send the modified file path and the modified items to the configuration error detection module, as shown in Figure 7. The prerequisite for detecting the configuration error in real time is to timely capture the update operation of the configuration file and the modified configuration item. This question has been solved in the article [28]. The author analyzed the mechanism of inotify in the Linux kernel and designed their own hook function to capture the file update operation. When the write operation is executed in the user layer, the file system of Linux will obtain the file descriptor fd according to the open operation and call the unified interface of write operation called vfs_wirte() by VFS layer, and then the vfs_wirte() function will call the write () function provided by the corresponding inode node based on the file operation pointer f_op. Therefore, the author designed the custom write function mywirte( ) and modified f_op to point to mywirte_iter(). In the mywrite_iter() function, the file path of the updated file and the updated operations are sent into the listener of the user layer through the custom file in the/proc file system. Thus, the purpose of monitoring the update operation of a configuration file is achieved.

After receiving the related information of the modified configuration item, the configuration error detection module will check whether this modified configuration item meets all the corresponding constraint rules in the constraint rule base. If so, it will judge whether the modification times of the current configuration file reach the threshold. When they do, it is needed to trigger the snapshot operation of this configuration file; otherwise, the modification record is stored in the configuration item update lists; if not, it indicates that the current modification is abnormal, and then the fast error recovery module will be called.

The real-time error detection algorithm of the configuration update operation is as follows.

4. Experiments and Results Analysis

In this section, we will conduct some experiments to verify the effect of the real-time configuration error detection

Table 5: The constraint rules of some configuration item types.

Configuration item type	Related configuration item type	Constraint rule
File path	User name	Operation permission
Directory	User name	Operation permission
Partial file path	User name	Operation permission
Partial directory path	File name	[A < partial file path>]+[B<file name>] = [C<file name>]
Partial directory path	User name	Operation permission
Partial directory path	File name	[A < partial directory path>]+[B<file name>] = [C < directory name>]
Number	Number	max,min
Memory	Memory	max,min

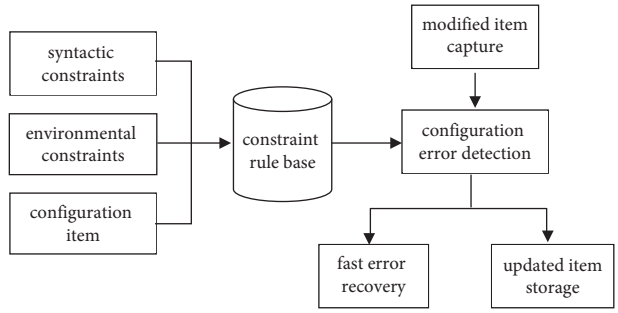

Figure 7: The design architecture of real-time configuration error detection.

scheme and analyze the experimental results. Firstly, the accuracy of configuration item classification will be verified based on the hybrid semantic classification method. Then, it will be tested whether the new scheme can monitor the modification operation of the configuration file in real time. Finally, the experiment will test whether the software configuration update error can be detected based on the constraint rule base algorithm 1.

4.1. Accuracy of Configuration Item Classification. Since the configuration constraint rule template of this scheme is defined based on the configuration type, during the detection process, the classification of configuration item type will directly affect the detection results when the configuration item is updated. Therefore, it is necessary to test the accuracy of configuration item classification. Due to the fact that Apache and MYSQL have a large number of configuration items and have the complex item types, this section takes the two software as the examples. By analyzing the official user guide and their configuration files, the configuration items in the two software tools are collected and classified according to the configuration item type list in Figure 3. The classification results are compared with the manual classification results to judge whether the scheme can classify the configuration items correctly. The experimental results are shown in Table 6.

It can be seen from the classification results that the hybrid classification method of regular matching based on configuration parameter values and combined with environmental information and configuration item names can achieve an accuracy of more than 87%. Based on the analysis of classification results, check the official description of configuration items that cannot be classified correctly. It is found that the parameter values of these configuration items basically belong to enumeration type; that is, the system provides several specific parameter values for users to choose. For such configuration items, it is necessary to analyze the source code to obtain more relevant information. At present, the classification method of this scheme is not combined with the analysis process of software source code, so it is impossible to correctly classify such configuration items, which is also the work that needs further research in the follow-up.

4.2. Real-Time Monitoring of the Configuration File Modification. In this experiment, we will take the configuration file of MySQL as an example to test whether the scheme can monitor the update operation of the configuration file in real time. The profile named mysqld.cnf was modified every 2 minutes, and the modification is executed three times during the experiment. Then, the dmesg command is used to view the system kernel log. The results are shown in Figure 8.

As can be seen from Figure 8, the goal of the experimental design is achieved, and the continuous update operations at a certain time interval are captured in the monitored file. When the mysqld.cnf is updated, the VFS layer calls the custom mywrite_iter() function, in which the time of write operation and the file path are captured. Therefore, this scheme can realize the real-time monitoring of profile update operation.

4.3. Effectiveness of Configuration Error Detection. In order to verify the effect of online configuration anomaly detection proposed in this scheme, it is necessary to modify the

Input: the modified configuration file path
Output: the backup data
(1) execute such the command as "diff config_file snapshot_file > patch_file" and get all the modified configuration items since the last snapshot
(2) Filter the modified operation based on the record in the configuration item update table
(3) get the currently modifying configuration item
(4) judge whether the current item parameter values meet all the related constraint rules in the rule base
(5) **If** OK **then**
(6) compare the current modification times and the threshold of modification times
(7) **If** the current times are less than the threshold **then**
(8) write the current modifying configuration item into the configuration item update table and change it into the modified item
(9) **else**
(10) create a new snapshot of the configuration file
(11) **end**
(12) **else**
(13) call the fast error recovery module to perform the recovery operation
(14) **end**

Algorithm 1: anomaly error detection of the configuration update operation

Table 6: Accuracy of configuration item classification results.

Software name	The number of configuration items	The number of correctly classified items	Classification accuracy (%)
MySQL 5.7.33	673	608	90.4
Apache 2.4.18	564	492	87.4

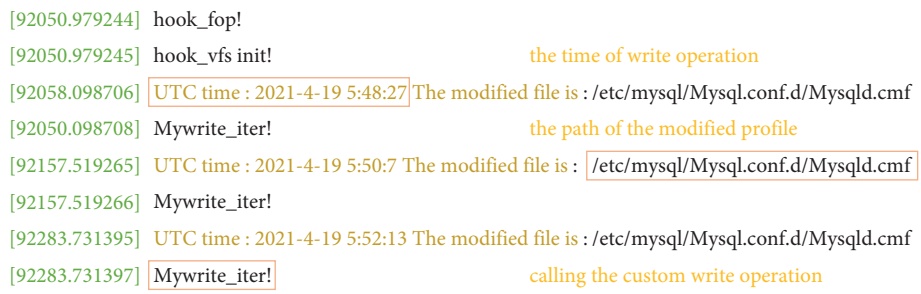

Figure 8: Real-time monitoring results of the profile update operation.

configuration parameter value in the form of violating the constraint rules in the test process to verify whether the system can detect the abnormal configuration in time. During the experiment, based on the idea of conferr [29] and confvd [21] configuring the fault generation tool, the representative and possible abnormal configuration update behavior is simulated, and the configuration is modified manually. At the same time, this experiment also refers to the problems caused by configuration exceptions in 8 real environments collected from major websites in reference [30] and restores the exception setting value based on its problem description to verify the effectiveness of this scheme for the exception detection of profile update.

The design of test cases in this section is based on the idea of the above two configuration fault injection tools. For MySQL, Apache, PostgreSQL, and PHP software, nine examples are designed that violate the syntax constraints of configuration values, the constraints of values and environment attributes, and the constraints of configuration correlation defined in this scheme. The syntax constraint test of configuration values includes syntax exceptions of values, abnormal value type and range. For example, the designed test is shown in Table 7, where the marked "∗" is the collected real configuration problems.

Based on the parameter values of test cases in Table 7, the corresponding parameter values of the related configuration file are modified to simulate the configuration errors. Then, our scheme is executed to verify whether it can monitor the modification of the configuration file in real time and whether it can detect these configuration errors. The test results are shown in Table 8.

Table 7: Test cases.

No	Software	Failure description	Test case
1	MySQL	Failed to get data	Key_ buffer_ size = 16N
2*	MySQL	Local server connection failed	Bind-address = 192.168.0.0
3	MySQL	Data query failed	Query_ cache_ limit = 16M
4	MySQL	Database connection failed.	Port = 8080
5	MySQL	Cannot output the log	General_ log_ file = /var/log/mysql
6	PHP	Cannot load the web page	Max_ execution_ time = 50
7	PHP	Refresh web page update failed	session.entropy_ file = /dev/random
8	PHP	Web pages cannot display	Default_ mimetype = "test/html"
9*	PHP	Failed to load file	- With_ config_ file_ path
10*	PHP	Database connection failure	Extension_ dir = ""/usr/local/php5/lib/extension"
11*	PostgreSQL	Software responds slowly	Work_ mem = 64 KB
12	PostgreSQL	Failed to load data	Shared_ buffers = 128 Mb
13*	PostgreSQL	Cannot output the log	Log_ directory = 'pg_ log'
14*	PostgreSQL	PID file was not written successfully	externam_pid_file = '/var/run/postgresql'
15*	Apache	User authentication failed	Authuserfile/etc/phpMyAdmin/passwd.setup
16	Apache	Web page cannot respond	Listen 1
17*	Apache	Cannot resolve the web page	AddType application/x-httpd-php .php.html*

Table 8: Test results of profile update operation.

No	Is the configuration update captured	Is the exception detected	Error analysis
1	Yes	Yes	The syntax constraint rule is broken
2*	Yes	Yes	The IP address is inaccessible
3	Yes	No	The parameter value is too large
4	Yes	Yes	The port is occupied
5	Yes	Yes	The parameter type should be the file type
6	Yes	Yes	The parameter value is abnormal
7	Yes	Yes	The file path does not exist
8	Yes	Yes	An unsupported MIME type
9*	No	No	The parameter type should be the directory type
10*	Yes	Yes	The parameter type should be the directory type
11*	Yes	No	The parameter value is too small
12	Yes	Yes	The syntax constraint rule is broken
13*	Yes	Yes	No write permission
14*	Yes	Yes	The parameter type should be the file type
15*	Yes	Yes	The path does not exist
16	Yes	Yes	The port is occupied
17*	Yes	Yes	An unsupported file type

It can be seen that, for the 17 test cases, our scheme can capture almost all the modification operations of the profiles, and it can detect most of the configuration errors. The number of errors is 17, while the number of detected errors is 14. By analyzing the detected configuration errors, it is found that the error scenes include the spelling and value of the parameter, whether the parameter value is accessible, and whether the port is occupied, etc. Therefore, it can be seen that this scheme performs well in detecting the syntax constraints of the configuration parameter and the environment constraints.

For the undetected configuration errors, we analyzed them and found the following questions:

Firstly, our scheme cannot well detect the dependency exceptions among configuration items. For example, we found the value of the parameter query_cache_limit in the test Case 3 should be less than that of query_ cache_ size after analyzing its configuration file. In practice, there are the dependencies not only between one configuration item and the other one in the same software, but also among the configuration items in different software. Therefore, one of our future work is to study how to more comprehensively analyze and extract the correlation constraints between two configuration items in different software.

Secondly, this scheme cannot detect the configuration errors caused by the operations outside the configuration file, such as the operation in the test Case 9. The operation fails due to the wrong type of uploaded file, while this operation does not belong to the configuration update operation, so our scheme cannot deal with it.

Thirdly, our scheme cannot detect the software performance exception caused by the improper configuration parameter values, such as the test Case 11, in which the value of the parameter work mem is too small, which leads to the

software response speed becoming slow, and the scheme still conclude that the configuration item value is normal.

5. Conclusion

In order to solve the problem that the major configuration error detection methods have the poor real-time performance, which may lead to potential threat to the software system, we design a real-time configuration error detection method of configuration based on the fine-grained configuration item type and a type constraint rules base.

Firstly, it is needed to collect the software configuration files in different formats and use the configuration management tool Augeas to resolve the configuration items into key-value pairs. Then, a hybrid classification method is proposed to classify all configuration items based on the fine-grained configuration item type lists. After that, a constraint template is predefined according to the syntax constraints of configuration parameter values, the connection constraints between the parameter values and the system execution environment, and the dependency constraints between two configuration item types. Finally, the configuration items are written into the constraint template to generate the configuration constraint rule base. On the foundation of the constraint rule base, our method can detect whether there are exceptions in the configuration update operations in real time, combining with the custom real-time monitoring function for the configuration file. The experimental results verify that our method can correctly classify the configuration items and can effectively detect the exceptions about the parameter spelling, the parameter value type, and the dependencies between two parameters. The classification accuracy and detection effectiveness reached 87% and 82%, respectively. In future, we plan to extend our work and test it on more complex software programs and frameworks. We intend to make use of big data analytics tools to be able to test large number of programs and frameworks in bulk.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under grant no. 61802210.

References

[1] T. Xu, L. Jin, X. Fan, Y. Zhou, and S. Pasupathy, "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*, pp. 307–319, Bergamo, Italy, September 2015.

[2] L. Welling and L. Thomson, *Php and Mysql Web development*, Sams Publishing, Carmel, Indiana, 2003.

[3] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, US, March 2003.

[4] B. Hale, *Why Every it Practitioner Should Care about Network Change and Configuration Management*, 2012.

[5] H. S. Gunawi, M. Hao, R. O. Suminto, and A. Laksono, "Why does the cloud stop computing? lessons from hundreds of service outages," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pp. 1–16, Santa Clara, USA, October 2016.

[6] T. Ryan, A. Chester, and J. Reece, "The uses and abuses of facebook: a review of facebook addiction," *Journal of behavioral addictions*, vol. 3, no. 3, pp. 133–148, 2014.

[7] Y. Sverdlik, *Microsoft: misconfigured network device led to azure outage*, Data Centre Dynamics Ltd (DCD), London, 2012.

[8] J. Garcia, Y. Feng, J. Shen, Y. Xia, and Q. Chen, "A comprehensive study of autonomous vehicle bugs," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 385–396, Seoul South Korea, July 2020.

[9] L. A. Barroso and U. Hölzle, "The datacenter as a computer: an introduction to the design of warehousescale machines," *Synthesis lectures on computer architecture*, vol. 4, no. 1, pp. 1–108, 2009.

[10] F. Oliveira, K. Nagaraja, R. Bachwani, and R. P. Martin, "Understanding and Validating Database System administration." in *Proceedings of the USENIX Annual Technical Conference*, pp. 213–228, Boston, Massachusetts, June 2006.

[11] W. Chen, X. Huang, and X. Qiao, "Research on software configuration error diagnosis and repair technology," *Journal of Software*, vol. 26, no. 06, pp. 1285–1305, 2015.

[12] F. Li, J. Yang, and J. Wu, "Research on Internet automatic configuration," *Journal of Software*, vol. 25, no. 1, pp. 118–134, 2014.

[13] T. Xu, X. Jin, P. Huang, and Y. Zhou, "Early detection of configuration errors to reduce failure damage," in *Proceedings of the 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 619–634, Savannah, GA, November 2016.

[14] Z. Yin, X. Ma, J. Zheng, and Y. Zhou, "An empirical study on configuration errors in commercial and opensource systems," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 159–172, Cascais, Portugal, October 2011.

[15] M. Attariyan and J. Flinn, "Automating configuration troubleshooting with dynamic information flow analysis," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation(OSDI)*, pp. 237–250, Berkeley, CA, USA, October 2010.

[16] A. S. Rabkin, *Using Program Analysis to Reduce Misconfiguration in Open Source Systems software*, Spring, UC Berkeley, 2012.

[17] Z. Dong, M. Ghanavati, and A. Andrzejak, "Automated Diagnosis of Software Misconfigurations Based on Static analysis," in *Proceedings of the 2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 162–168, IEEE, Pasadena, CA, USA, November 2013.

[18] T. Xu, J. Zhang, P. Huang, and D. Yuan, "Do not blame users for misconfigurations," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 244–259, New York; NY, November 2013.

[19] J. Zhang, L. Renganarayana, X. Zhang, and Y. Zhou, "Encore: exploiting system environment and correlation information for misconfiguration detection," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 687–700, Salt Lake City, USA, March 2014.

[20] W. Li, S. Li, X. Liao, and Y. Zhou, "Conftest: generating comprehensive misconfiguration for system reactionability evaluation," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pp. 88–97, Karlskrona, Sweden, June 2017.

[21] S. Li, W. Li, X. Liao, and S. Peng, "Confvd: system reactions analysis and evaluation through misconfiguration injection," *IEEE Transactions on Reliability*, vol. 67, no. 4, pp. 1393–1405, 2018.

[22] R. Talwadker, "Dexter: faster troubleshooting of misconfiguration cases using system logs," in *Proceedings of the 10th ACM International Systems and Storage Conference*, pp. 1–12, 2017.

[23] J. Tucek, S. Lu, C. Huang, and Y. Zhou, "Triage: diagnosing production run failures at the user's site," *ACM SIGOPS - Operating Systems Review*, vol. 41, no. 6, pp. 131–144, 2007.

[24] Y. Y. Su, M. Attariyan, and J. Flinn, "Autobash: improving configuration management with operating system causality analysis," *ACM SIGOPS - Operating Systems Review*, vol. 41, no. 6, pp. 237–250, 2007.

[25] X. Xu, S. Li, Y. Guo, and D. Wei, "Automatic type inference for proactive misconfiguration prevention," in *Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering(SEKE)*, pp. 295–300, San Francisco Bay, CA, July 2017.

[26] A. Rabkin and R. Katz, "Static extraction of program configuration options," in *Proceedings of the 33rd International Conference on Software Engineering*, pp. 131–140, New York. NY, May 2011.

[27] D. Lutterkort, "Augeas–a configuration api," in *Proceedings of the Linux Symposium*, pp. 47–56, Citeseer, Ottawa, ON, 2008.

[28] M. Ming, L. Liu, and G. Zhao, "An adaptive data protection scheme for optimizing storage space," in *Proceedings of the International Conference on Machine Learning for Cyber Security*, pp. 250–260, Guangzhou, China, October 2020.

[29] L. Keller, P. Upadhyaya, and G. Candea, "Conferr: a tool for assessing resilience to human configuration errors," in *Proceedings of the 2008 IEEE International Conference on Dependable Systems and Networks with FTCS and DCC (DSN)*, pp. 157–166, IEEE, Anchorage, AK, June 2008.

[30] S. Zhou, S. Li, X. Liu, and X. Xu, "Easier said than done: diagnosing misconfiguration via configuration constraints analysis: a study of the variance of configuration constraints in source code," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pp. 196–201, Karlskrona, Sweden, June 2017.