Intelligent Software Service Configuration Technology Based on Association Mining

Fei Wang*, Zhengjian Zhao, Zhichao Wang, Minchao Ma and Junjie Lu
Cetc Taiji (group) Ltd. Xian R&D Center, Xi'an, China.
*Email: 920721263@qq.com

Abstract. Association relationship and types between software service configuration parameters determine that the configuration items must meet the corresponding rules and constraints in parameter value and setting order. However, the distribution of software services and the architecture based on the loose coupling of heterogeneous components make the association of configuration parameters more concealed. The correlation and constraints of configuration parameters are crucial to ensure system configuration. When deploying, migrating, and updating systems, violating constraints will lead to configuration errors, resulting in system failures. To minimize the loss caused by configuration errors, association rules between configuration items can be mined to avoid potential incorrect settings. This paper proposes an intelligent software service configuration technology based on association mining by analyzing configuration items. The sample data of the configuration file is extracted from the open source code base, and the configuration file is parsed to obtain the configuration item name and value. Then mine the association rule of each configuration item pair based on the configuration item name, value, and type. According to the mined configuration constraints, configuration item error detection, cascade update and value recommendation are performed. Finally, the proposed method is tested and evaluated on six typical open source software, and the experimental results prove the accuracy and effectiveness of the method in this paper.

1. Introduction
With the continuous growth of software systems in terms of scale and function, correctly configuring software systems has become a very complex work, and configuration errors are prone to occur during system configuration [1]. At present, configuration errors have become one of the main causes of system failures, and such failures usually bring serious economic losses, security risks and functional failures [2, 3]. For example, the Microsoft search engine Bing encountered downtime on the second day of its release, and this downtime was caused by changes in the service configuration due to Bing’s internal testing. Facebook has also experienced serious downtime incidents due to server configuration changes. Among configuration errors, errors caused by the association of configuration items account for a large part, so it is very necessary to improve the quality of software service configuration by mining the association relationship between configuration items [4].

At present, program analysis methods are usually used for configuration error detection, which mainly includes static analysis and dynamic analysis [5, 6]. Huang et al. [7] used the static data flow method to analyze the software source code, the data flow and the control flow to obtain the configuration item type, value range and dependency constraints, and calculate the possible errors in advance to improve the reliability of the software. In this method, when analyzing configuration variables, the program mapping between configuration items and source code is realized by manual
writing, which not only increases the development burden and depends on the developer's ability, but also needs to provide software source code, which limits the application scenarios. Zhang et al. [8] combined static analysis with dynamic analysis and proposed a method to determine the root cause of configuration errors -- a single configuration item that can be changed to produce the desired behavior, associating the unwanted behavior with a specific configuration item. This method does not require the user to provide test data to check whether the software function is correct, it is completely automated. However, this approach only focuses on key-value pair options and can only diagnose software involving a configuration item error, so it has great limitations.

The configuration error detection method based on program analysis relies on analyzing the source code to analyze the configuration items. It is not only limits the programming language, but also has great limitations in application scenarios [9, 10]. To solve this problem, some researchers proposed to use the configuration file to determine the correlation between configuration items for configuration error detection. Lakshminarayanan et al. [11] proposed a framework to automatically detect software configuration errors, which used environment information to set derived semantic type for each configuration item, and used annotators to expand the environment-related attributes of all configuration items, then presenting the correlation of configuration items in the form of templates based on machine learning. This method used a set of predefined regular expressions to infer the possible types of configuration items, rather than representing a single type. This not only speeds up the inference process of association rules, but also reduces the number of incorrect configurations. Xu et al. [12] calculated the correlation probability by using the similarity of configuration item values, and used outlier filter, infrequent value filter and normalized Google distance filter for filtering. However, these filters cannot be used due to limitations in practical applications. In addition, outlier and infrequent value filters require multiple service components and cannot be applied when there is only one.

To make better use of configuration files to calculate the association relationship between configuration items and reduce system failures caused by configuration errors, this paper proposes an intelligent software service configuration technology based on association mining. Firstly, the configuration file is parsed to obtain the configuration item name and configuration item value, and the regular expression [13] is used to infer the configuration item type; Then, according to the configuration item name, value and type, the association relationship between each configuration item is analyzed by FP-growth algorithm [14]; Finally, configuration item error detection, cascade update and value recommendation are carried out according to the mined association relationship. The method proposed in this paper uses configuration files to analyze configuration items, which does not limit programming languages and can be widely used. The regular expression is used to infer the type of configuration items instead of using a single type, which can quickly carry out association rule inference; And we use the FP-growth algorithm to calculate the association rules between configuration items. On the premise of ensuring the accuracy of association rules, this algorithm also greatly improves the calculation efficiency.

The rest of this paper is arranged as follows: the second part introduces the overall technical route of software service configuration method, configuration item type inference and FP-growth algorithm; The third part is the experiment and results; The last part summarizes this paper.

2. Intelligent Software Service Configuration Method
The technical route of the intelligent software service configuration method is shown in Figure 1, which mainly includes five parts: configuration library construction, configuration item analysis, configuration item type inference, association rule calculation, and intelligent software service configuration [15].
A complete software system often relies on mainstream middleware, database, application server, security components and other basic services, so there are numerous configuration files to ensure the normal use of software [16]. This paper grabs the configuration files of open source software from the code websites GitHub, CSDN, StackOverflow, etc. And all the original configuration information is parsed using the configuration resolution template to obtain the configuration item name and configuration item value, which are saved as configuration records in the key-value format.

2.1 Configuration Item Type Inference

Configuration item types usually include value types (short, int, long, double), Boolean and string types. String types can be subdivided according to keywords (mysql, Kafka, rabbitmq, elasticsearch, etc.) [17]. In this paper, each configuration item value is matched with the regular expression for configuration item type identification. The first step is syntax matching, which uses a predefined syntax pattern to roughly guess the configuration item type. For example, any string containing slashes is a potential FilePath type. The second step is semantic verification, which verifies the type by checking the corresponding external resources (such as the file system). For example, if the configuration item type is classified as the FilePath type in the first step, the complete file system data will be searched to verify whether the path exists. The first step of matching eliminates unlikely types and makes reasoning effective. The second step ensures the accuracy of the reasoning. The combination of syntax matching and semantic verification can ensure the validity and accuracy of configuration item type inference. Table 1 shows the detailed information of some typical configuration item types.

| Type         | Regular expressions     | Semantic       |
|--------------|-------------------------|----------------|
| FilePath     | ./+(/.+)*               | File System    |
| UserName     | [a-zA-Z][a-zA-Z0-9 ]*   | /etc/passwd    |
| GroupName    | [a-zA-Z][a-zA-Z0-9 ]*   | /etc/group     |
| IPAddress    | [\d]{1,3}([.\d]{1,3}){3}|              |
| PortNumber   | [\d]+                   | ip, ping       |
| FileName     | [\w-]+[.\w-]+          | File System    |
| Number       | [0-9]+[0-9]*            | N/A            |
| URL          | [a-zA-Z]+://./*         | Curl           |
| PartialFilePath | /?.+/(.+)*         | File System    |
| MIME Types   | [\w/-. ]+               | IANA [3]       |
| CharSet      | [\w]+                   | IANA           |
| Language     | [a-zA-Z]{2}             | ISO 639-1      |
| Size         | [\d]+[KMG]T            | N/A            |
| Boolean      | Values Set              | N/A            |
2.2. Association Rule Calculation

2.2.1. Formal Description of Association Rules. Association rules reflect the dependence and correlation between one transaction and other transactions [18]. Set \( I = \{ I_1, I_2, \ldots, I_m \} \) as an item set, \( m \) is the number of items, and \( I_i \) represents the \( i \)th item. Transaction \( t \) represents a subset of \( I \), corresponding to each configuration item pair.

Association rules are implied expressions like \( X \rightarrow Y \), where \( X \) and \( Y \) are disjoint item sets. Association analysis is measured by support and confidence. Support is the probability of an item set, usually a threshold \( \text{minsupport} \) is set. When the support exceeds \( \text{minsupport} \), it is considered a frequent item. It is defined as equation (1):

\[
\text{support}(X \rightarrow Y) = \frac{P(X) \cap P(Y)}{P(T)} = \frac{\text{num}(X \cap Y)}{\text{num}(T)}
\]

Where \( T \) represents the total number of transactions and \( \text{num}(\cdot) \) represents the number of occurrences of a specific item set in the transaction set. Confidence represents the probability that the item set \( Y \) appears simultaneously in the transaction of item set \( X \). The definition is shown in equation (2):

\[
\text{confidence}(X \rightarrow Y) = P(Y | X) = \frac{P(X \cap Y)}{P(X)} = \frac{P(X \cap Y)}{P(X)}
\]

Association rule mining is to find rules that meet the minimum support and minimum confidence at the same time. This task can be divided into two stages [19]:

(1) Generation of frequent item sets: find out all item sets that meet the minimum support, and these item sets are called frequent item sets.

(2) Rule generation: find rules that meet the minimum confidence from frequent item sets.

2.2.2. FP-growth Algorithm. The basic idea of the FP-growth algorithm [20] is to compress transactions by constructing FP-tree, so that the generation of frequent item sets only needs to scan data records twice. Moreover, this algorithm does not generate candidate item sets, so the efficiency is relatively high. The basic process of the algorithm is divided into two steps, constructing FP-tree and mining frequent item sets from FP-tree. FP-tree is a prefix tree that connects similar items through links, frequent items with higher support are closer to the root node. After constructing the FP-tree, frequent item sets can be mined from the FP-tree. Firstly, mining the conditional pattern base from FP-tree, the conditional pattern base represents the set of all paths ending with the searched element. Then, a conditional FP-tree is established according to the conditional pattern base. Finally, mining frequent association rules. The specific process of the FP-growth algorithm is shown in Table 2:

| Table 2. FP-growth algorithm flow. |
|-----------------------------------|
| **Input:** transaction data set, minimum support. |
| **Output:** complete set of frequent patterns. |
| 1) Create the original FP-tree. |
| a) Traverse the transaction data set, find all 1-frequent item sets \( L \), and sort \( L \) in descending order of support. |
| b) Create FP-tree with "null" as the root node. |
| c) For each transaction in the data set, first sort the item sets in the transaction according to the order in \( L \), and then insert each sorted item set into the tree recursively. |
| 2) Mining the maximum frequent item set in FP-tree: implemented by calling \( \text{FP\_Growth(tree, null)} \). |

\( \text{FP\_Growth(tree,} \alpha \text{)} \): \( \alpha \) is the frequent pattern tree to be mined, \( \alpha \) is the conditional pattern.
base, and the initial value is empty.

a) If the tree contains a single path \( p \), the combination of each node in \( p \) (denoted as \( \beta \)) produces mode \( \beta \cup \alpha \), and its support is equal to the minimum support of the node in \( \beta \).

b) If the tree does not contain a single path \( p \), pattern \( \beta = \alpha \cup \alpha \) is generated for each \( \alpha \) in the tree header table, and its support is equal to that of \( \alpha \). Construct the conditional pattern base of \( \beta \), and then construct the condition \( \text{FP}_\beta \) of \( \beta \). If \( \text{FP}_\beta \) is not empty, call \( \text{FP}_\beta \text{Growth}(\text{FP}_\beta, \beta) \).

2.3. Intelligent Software Service Configuration

Analyzing the correlation between configuration item sets by parsing configuration files is the basis for realizing the automation and intelligent configuration deployment of large-scale distributed software. Once the correlation between configuration items is ignored, configuration constraints may be violated, leading to system failure. Configuration constraints describe the requirements, association rules, semantic conditions or restrictions of application configuration. They are the abstraction of general rules and invariance characteristics to be followed in the software service configuration. And they are closely related to the association relationship of configuration parameters and their use in the system, such as consistency constraints, parameter format constraints, value range constraints, etc. We will study the constraint inference method based on configuration parameter association, analyze and mine the corresponding relationship between configuration parameter association and configuration constraint, establish parameterized template representation method of configuration constraint, and realize the template based configuration constraint generation mechanism.

Software service configuration constraint design is realized by configuration item template. The user can use the default rules of the software configuration item set or customize the configuration template rules. When adding a configuration item, we can verify the entered configuration item based on the configuration item template rule. If it meets the template rules, it will be added successfully. If it does not, an error message will be prompted. Template-based constraint reasoning: 1) Considering that different software may have different value requirements for the same configuration item, the template sets a scope (global shared and software private), allowing two scopes of configuration items to have the same name, priority: private>global. 2) The restriction of the value range is constrained by regular expression. 3) Restrictions on value types. For configuration files such as yml, properties, ini, xml, etc., check the value of the value type to determine whether it is a legal value. If it is not a legal value, an error message will be displayed. 4) Providing strategies for shielding global templates. Constraint inference based on configuration item association: combined with the calculated configuration item association rules, when modifying the current configuration item, it is necessary to check whether there are strongly associated configuration items. If it exists, compare the values of the currently modified configuration item and the strongly associated configuration item. If the association relationship between the current configuration item and the strongly associated configuration item becomes weak after the modification, it will prompt that the modified configuration item may be illegal.

The correlation between components and their parameter configurations in software services makes the change and update of one or some parameter configurations produce a certain range of cascade propagation and impact through configuration correlation. Ignoring or omitting other affected components and configurations due to configuration complexity will lead to inconsistent configurations among associated components, resulting in system failures. Therefore, before configuration update, the impact domain should be analyzed through the association rules of configuration items to obtain the strongly associated configuration items, so as to form the target collection and update sequence of parameter configuration cascade update. According to the parameter configuration update and value setting, we will combine the constraint configuration, predict the value range of the parameter, and generate the corresponding value recommendation.
3. Experiments and Results

3.1. Experimental Environment Setting
To evaluate the effectiveness of the proposed intelligent software service configuration method, we studied the configuration items in six representative open source software configuration files. The information of six software is shown in Table 3. For each software, we can get different amounts of training data according to the availability of software configuration files. All experiments were conducted on a server with an 8-core processor and 16GB memory.

Table 3. List of software for experiments.

| Application | Description     | Application | Description     |
|-------------|-----------------|-------------|-----------------|
| MySQL       | Database        | SSH         | Remote connection tool |
| Apache      | Web server      | Nginx       | Reverse proxy   |
| PHP         | Scripting language | Redis | Database        |

3.2. Experimental Procedures and Results
The specific implementation steps of intelligent software service configuration technology include: 1) Grab the configuration file of the target software, parse the configuration file into key-value format configuration item and save it in the configuration library. Figure 2 shows the number of configuration items parsed from the six target software; 2) Define regular expressions to determine the type of configuration items; 3) Use the FP-growth algorithm to mine the association relationships between configuration items; 4) The mined association rules are used to form configuration constraints for error detection, cascading update and value recommendation of configuration items.

![Figure 2. The number of target software configuration items.](image)

To evaluate the correlations calculated in this paper, we randomly sampled 20 problems that are prone to configuration errors from six target software, and analyzed the fault errors caused by these 20 problems. Among these 20 questions, 12 need to detect the correlation between configuration items, and the remaining 8 problems are caused by incorrect input of the configuration item name or configuration item value, and these problems can be solved by association rules and configuration constraints. Next, we use precision (P) and recall (R) as indicators to evaluate the association relationship calculated in this paper, as shown in equation (3):

\[
P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}
\]

(3)

Where true positive (TP) represents the number of associations that are correctly found, false positive
(FP) represents the number of associations that are incorrectly judged, and false negative (FN) represents the number of associations that are judged to be unrelated. We use precision and recall to test and verify the correlation calculated in this paper on six target software, and the results are shown in Figure 3. According to the results, we can find that the method in this paper can effectively calculate the association relationship between configuration items.

![Figure 3. Comparison of accuracy and recall.](image)

To prove the effectiveness of the intelligent software service configuration technology based on association mining proposed in this paper, the error configuration detection rate of six target software was used as an indicator for experimental analysis, and the results are shown in Figure 4. It can be seen from Figure 4 that the error configuration detection rate of the six target software is above 90%. The advantage of this configuration error detection method is that it sets the configuration type for each configuration item, uses the FP-growth algorithm to calculate the association relationship between the configuration items, analyses and mines the corresponding relationship between configuration parameter association and configuration constraint, establishes parameterized template representation method of configuration constraint, and uses the constraints of the template to complete the error detection, cascade update, and value recommendation of configuration items. The contents of configuration items are closely related to the actual operating environment. Therefore, misconfiguration detection can effectively avoid system faults caused by configuration errors, which is of great significance to the automatic deployment, migration, and update of application services.

![Figure 4. Comparison and analysis of error configuration detection rate.](image)

4. Conclusions
The correlation between components and parameter configurations in software services makes the change and update of one or some parameter configurations produce a certain range of cascade propagation and impact through configuration association. Ignoring or omitting other affected
components and configurations due to configuration complexity will lead to inconsistent configurations among associated components. Configuration settings are the bridge between the application and the execution environment. Configuration errors caused by service deployment, migration, and updates have become the main cause of software failures. However, configuration error diagnosis based on program analysis faces the challenge of mapping configuration items to related programs, depends on the source program, and the usage scenarios are also limited. To make up for these defects, this paper proposes an intelligent software service configuration technology based on association mining, which mines the association relationship between configuration items by parsing configuration files, and uses the mined association rules to form configuration constraints for configuration error detection, cascade update and value recommendation. This method can automatically map configuration items to relevant program variables without understanding complex source code. At the same time, we also carried out experimental evaluation on six typical open source software systems. The experimental results proved the accuracy and effectiveness of the proposed method. In addition, the research can effectively avoid system downtime caused by configuration errors, greatly reduce the human and material resources required for fault handling and system change review, and has a wide application prospect.

Acknowledgments
This work was supported by the National Natural Science Foundation of China (No. 61762025).

References
[1] Xiong Y, Zhang H and Hubaux 2015 A Range Fixes: Interactive Error Resolution for Software Configuration IEEE Transactions on Software Engineering 41(6) 603-19
[2] Liao X, Zhou S and Li S 2018 Do You Really Know How to Configure Your Software? Configuration Constraints in Source Code May Help IEEE Transactions on Reliability 67 832-46
[3] Huang P, Bolosky W J and Singh 2015 A ConfValley: a systematic configuration validation framework for cloud services Proceedings of the Tenth European Conference on Computer Systems 19 1-16
[4] Polynomials L S 2016 Context-based Online Configuration-Error Detection 1-14
[5] Rabkin A S 2012 Using Program Analysis to Reduce Misconfiguration in Open Source Systems Software University of california berkeley
[6] Santolucito M, Zhai E and Piskac R 2016 Probabilistic Automated Language Learning for Configuration Files International Conference on Computer Aided Verification 9780 80-87
[7] Huang P 2014 Do Not Blame Users for Misconfigurations 244-59
[8] Zhang S, Ernst M 2013 Automated diagnosis of software configuration errors 35th International Conference on Software Engineering (ICSE) 13799252 312-21
[9] Zhang S 2013 ConfDiagnoser: An automated configuration error diagnosis tool for Java software 35th International Conference on Software Engineering (ICSE) 13799200 1438-40
[10] Rabkin A and Katz R 2011 Static extraction of program configuration options Proceedings of the 33rd International Conference on Software Engineering 131-40
[11] Jiaqi Zhang and Lakshminarayanan 2014 EnCore: Exploiting System Environment and Correlation Information for Misconfiguration Detection Acm Sigplan Notices A Monthly Publication of the Special Interest Group on Programming Language 687-700
[12] XuTianyin and ZhouYuanyua 2021 Systems approaches to tackling configuration errors: A survey ACM Computing Surveys 47(4) 7082
[13] Guizzetti L 2021 FINDREGEX: Stata module to find variable names matching a regular expression pattern Statistical Software Components 145-57
[14] Wang S and Shi W 2011 Data Mining and Knowledge Discovery Kluwer Academic Publishers 32 197-229
[15] Zhou S Liu X and Li S 2016 ConfMapper: Automated Variable Finding for Configuration Items
in Source Code *IEEE International Conference on Software Quality, Reliability and Security Companion* 228-35

[16] Huang G Luo C and Wu K 2019 Software-Defined Infrastructure for Decentralized Data Life cycle Governance: Principled Design and Open Challenges *IEEE 39th International Conference on Distributed Computing Systems* 1674-83

[17] Tu C 2018 DevOps-based software configuration item management *Wireless Internet Technology* 35-46

[18] Babaei A and Khedmati M 2021 A new DEA model for ranking association rules considering the risk, resilience and decongestion factors *European J of Industrial Engineering* 15(4) 463-86

[19] Meesala S R and Subramanian S 2021 Feature based opinion analysis on social media tweets with association rule mining and multi-objective evolutionary algorithms *Concurrency and Computation: Practice and Experience* 6586

[20] Ls A Fv C and Rya B 2020 A guided FP-Growth algorithm for mining multitude-targeted item-sets and class association rules in imbalanced data *Information Sciences* 553 353-75