Misconfiguration Detection for Cloud Datacenters using Decision Tree Analysis

Tetsuya Uchiumi, Shinji Kikuchi and Yasuhide Matsumoto
Cloud Computing Research Center, FUJITSU LABORATORIES LTD.
4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211-8588, Japan
{uchiumi.tetsuya, skikuchi, ymatsumo}@jp.fujitsu.com

Abstract— **Since many components comprising large scale cloud datacenters have a great number of configuration parameters (e.g. hostnames, languages, and time zones), it is difficult to keep consistencies in the configuration parameters. In such cases, misconfigured parameters can cause service failures. For this reason, we propose a misconfiguration detection method for large-scale cloud datacenters, which can automatically determine possible misconfigurations by identifying the relations existing among majority of the parameters using statistical decision tree analysis. We have also developed a pattern modification method to improve the accuracy of the decision tree approach. We evaluated the misconfiguration detection performance of the proposed method by using both artificial data and actual data. The results show that we can achieve higher accuracy (78.6% in the actual data) in misconfiguration detection by using the pattern modification.**

Keywords-component; large scale datacenter, cloud computing, decision tree, pattern identification, misconfiguration detection

I. INTRODUCTION

Recently, many people has been interested in cloud computing technology [1], which can provide many users IT resources (e.g. hardware, application and network) on-demand basis. The cloud providers can reduce costs and improve efficiency in resource utilization by consolidating many users in their datacenter and taking advantage of economies of scale [2, 3].

However, large-scale cloud datacenters have management issue in keeping proper values for a great number of configuration parameters (e.g. IP addresses, Gateway addresses) in a cloud datacenter consisting of many components such as servers, storages, and middleware [1, 9]. Especially, identifying "hidden" misconfigurations in the system which seems working properly is quite difficult. For example, if some parameters for disaster recovery functions are not configured properly, it cannot be revealed until an actual disaster occurs and the disaster recovery fails.

From the above background, we propose a misconfiguration detection method to detect candidates of misconfigurations (including "hidden" misconfiguration) automatically by identifying the exceptional values of configuration parameters using statistical analysis. In our approach, first, we identify hidden patterns that consist with the majority of configuration parameters by using decision tree analysis based on the assumptions of some "uniformities" in a certain part of large-scale cloud systems. In order to improve the accuracy of the misconfiguration detection, we add a pattern modification method based on the cloud computing system's structure which is constructed by iteratively connecting some units (e.g. racks and regions) which have similar configuration. Then, we determine candidates of misconfigured parameters, which are inconsistent with the derived patterns.

The rest of paper is organized as follows. We first explain the background of our research in Section II. In Section III, we propose a misconfiguration detection framework. Then we evaluate the proposed method in Section IV. After showing the related work in Section V, finally Section VI concludes this paper.

II. BACKGROUND

A. Configuration of cloud datacenter

In this paper, we target a cloud computing system consisting of several datacenters deployed in different locations called "regions" (Figure 1). A large number of servers in each region can be divided in two groups; (1) servers accommodating virtual machines for cloud users' applications, and (2) management servers to control cloud infrastructures and services (e.g. administration, DNS, monitoring, and accounting). Here we concentrate on the latter group, because the misconfiguration in management functions can give serious impact on many services provided by the cloud datacenter. Figure 2 shows the example of server configuration (In actual data center, servers have more configuration parameters than this example). In order to make the cloud infrastructure work properly, proper values must be assigned to these configuration parameters. However, due to the too many configuration parameters and relationships among them, it is quite difficult to manage the configuration parameters and various misconfigurations can occur.

B. Typical types of misconfiguration

We investigated misconfigurations in an actual datacenter and found the two types of typical misconfigurations. For evaluation, we inject these misconfigurations on purpose in Section IV.

(1) Typing error

The "typing error" type misconfiguration includes simple typo (e.g. "jp"→"jpp" or "jj"). To distinguish this error from the following "copy and paste error," we suppose that there is no typo which can be a correct value for parameter of other component in the system (e.g. "jp"(Japanese)→"en"(English)).

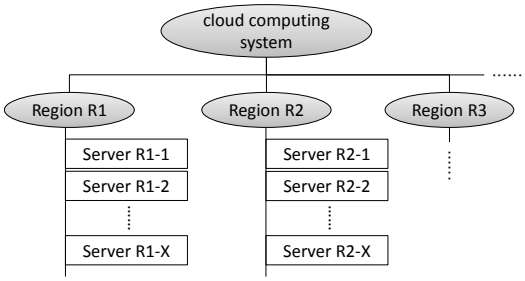

Figure 1. A structure of cloud computing system

ID	Region	Role	NETMASK	UTC	LANG
Server T-1	Tokyo	admin	255.255.222.0	FALSE	jp
Server T-2	Tokyo	DNS	255.255.225.0	TRUE	jp
Server T-3	Tokyo	monitoring	255.255.255.0	FALSE	jp
Server K-1	Kyoto	admin	255.255.222.0	FALSE	jp
Server K-2	Kyoto	DNS	255.255.225.0	TRUE	jp
Server K-3	Kyoto	monitoring	255.255.255.0	FALSE	jp
Server B-1	Berlin	admin	255.255.222.0	TRUE	en
Server B-2	Berlin	DNS	255.255.225.0	TRUE	de
Server B-3	Berlin	monitoring	255.255.252.0	TRUE	de
Server N-1	New York	admin	255.255.222.0	TRUE	en
Server N-2	New York	DNS	255.255.225.0	TRUE	en
Server N-3	New York	monitoring	255.255.255.0	TRUE	en
Server M-1	Munchen	admin	255.255.222.0	TRUE	de
Server M-2	Munchen	DNS	255.255.225.0	TRUE	de
Server M-3	Munchen	monitoring	255.255.252.0	TRUE	de
Server L-1	London	admin	255.255.222.0	TRUE	en
Server L-2	London	DNS	255.255.225.0	TRUE	en
Server L-3	London	monitoring	255.255.255.0	TRUE	en

(2) Copy and paste error

The "copy and paste error" type misconfiguration includes improper copy. For example, it occurs if IP address (which should be different in all servers) of "Server T-1" in Tokyo region is copied to "Server B-1" in Berlin region.

III. MISCONFIGURATION DETECTION FRAMEWORK

In order to detect misconfigurations in configuration parameters in cloud datacenters, we developed the framework outlined in Figure 3. Our approach consists of the following three steps.

1) Majority pattern identification: Collect the data of the value assigned to each configuration parameter by sampling from each server, and determine the patterns that consist with majority of configuration parameters using the decision tree analysis.
2) Pattern modification: Modify the determined majority pattern if there is no majority and servers can be divided into some groups with the same size.
3) Misconfiguration detection: Determine the configuration parameter contradictory to the identified patterns as a candidate of misconfiguration.

The details of each step are as follows.

A. *Majority pattern identifiction*

In our approach, the hidden patterns among assigned values of configuration parameters are identified by the following way.

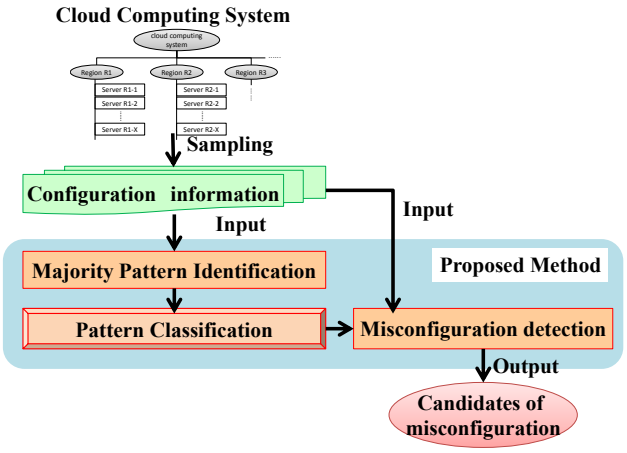

Figure 3. Outline of misconfiguration detection using decision tree analysis

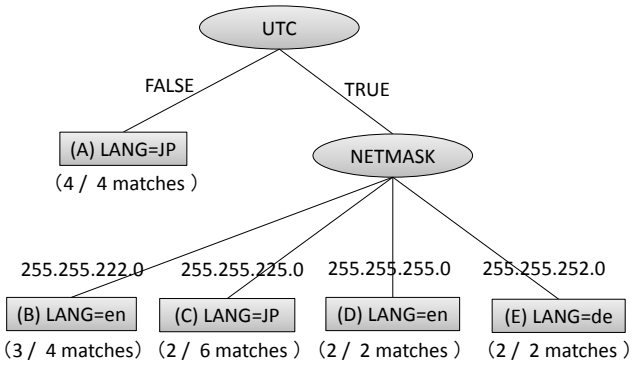

Figure 4. Example of decision tree of parameter "LANG"

Based on the assumption that proper values are assigned to the majority of configuration parameters in a large system and there is some "uniformities" (e.g. the same values are assigned to "nameserver" parameters in almost all servers), we can identify the hidden structures that consist with the majority of assigned values of configuration parameters. We call this structure "the majority pattern." The majority patterns can be represented by IF-THEN rules, such as "IF UTC = TRUE and NETMASK = 255.255.222.0 THEN LANG = en."

To identify the majority pattern, we use decision tree analysis [10]. The decision tree analysis is a clustering method which makes tree-like models. In our approach, we classify the servers in the cloud system into several groups having similar values for configuration parameters using the decision tree. We regard a path from root node of the decision tree to each leaf as a majority pattern which represent a set of parameters and their values that many servers have in common.

As the algorithm of decision tree analysis, we use C4.5 algorithm [4]. Figure 4 shows an example of a decision tree for the "LANG" parameter constructed from the data in Figure 2. In this figure, "(y/x matches)" means that x is the number of servers consistent with the "IF" part of the majority pattern and y is the number of servers in x servers, which also consist of the "THEN" part.

B. *Pattern modification*

According to the assumption that a cloud infrastructure is divided by units such as regions and racks, there is a possibility that the servers can be divided into several subgroups with

almost the same size. For example, in Figure 4, we detected 6 servers (Server T-2, K-2, B-2, N-2, M-2, and L-2 serving as a DNS server in each region) categorized in the leaf node (C). The set of the values assigned to the "LANG" parameter in these servers is $\boldsymbol{W}$ = {jp, jp, en, en, de, de}. Figure 2 indicates that each server has the value for the LANG parameter corresponding to the located country of its region (e.g. since Tokyo and Kyoto regions are in Japan, the value of "LANG" parameter should be "jp" (Japanese)). Since there is the same number of servers for each assigned value (jp, en, and de) in this case, we cannot determine one majority pattern. For example, if we perform misconfiguration detection using the majority pattern "IF UTC = TRUE and NETMASK = 255.255.225.0 THEN LANG = jp," 4 servers (Server B-2, N-2, M-2, and L-2) inconsistent with the THEN part can be determined to be misconfigured, while they are configured properly.

In order not to conclude that there are misconfigurations in such situations, we improved the decision tree algorithm by adding a pattern modification based on the typical configurations in the large scale systems, which have uniform structure. Here, if we can divide x servers into x/y subgroups with y servers having the same assigned values for a target pattern, we concatenate all of the values assigned to x server into the "THEN" part of the majority pattern by the "or" operator (Figure 5).

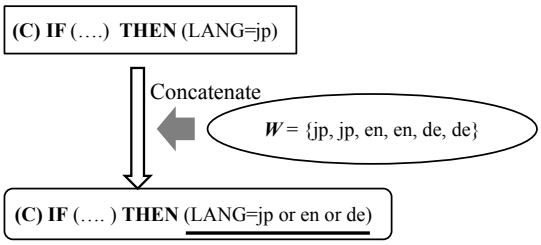

Figure 5. IF-THEN rule using pattern modification

C. Misconfiguration detection

In this step, we determine the configuration parameters inconsistent with the majority patterns identified in the majority pattern identification step. If a server has a set of parameters consistent with the "IF" part of the majority pattern while the value of a target parameter contradicting the "THEN" part, we conclude that the target parameter's value is inconsistent with the pattern and regard it as a candidate of the misconfigured parameter. Otherwise, we conclude that the parameter's value is assigned properly.

IV. PERFORMANCE EVALUATION

We conducted the evaluation of our approach using two types of configuration parameter data; artificial data and actual data. The artificial data is constructed by adding some intentional errors at random into the correct configuration parameters. We also evaluate the proposed method by using actual data obtained from actual datacenters containing hidden misconfigurations which is difficult to identify.

The condition of evaluation in the artificial and actual data is shown in Table. I. To preserve data privacy, all numbers in the actual data are normalized to 10 regions and 100 servers in each region.

TABLE I. THE CONDITION OF EVALUATION

	Artificial data	Actual data (Normalized)
Number of regions	6	10
Number of servers in each region	60	100
Number of all parameters	20424	67700

For evaluation functions, we use Precision and Recall.

- ***Precision:*** The ratio of the correct misconfiguration set including candidates of misconfiguration set.
- ***Recall:*** The cover ratio of the correct misconfiguration set including actual misconfiguration set.

Since Precision and Recall are in a trade-off relation, it is difficult to obtain high accuracy in both of them. In order to evaluate both the functions at the same time, we use F-measure[8], which is the combination of them.

$$F\text{-}measure = 2 \times precision \times recall / (precision + recall) \quad (1)$$

A. Evaluation of "Pattern modification" by artificial data

We evaluated the proposed method by using two-pattern artificial data to which we injected the errors described in Section 2-B. In the pattern 1, we add "Typing Errors" at random in about 1% (205 pieces) of all values (20424). In the pattern 2, we add "Copy and Paste Errors" at random in about 1% (208 pieces) of all values (20424). Misconfigurations added at random makes the identification more difficult than the identification in an actual situation. So, this error injection is appropriate for the evaluation for identification accuracy.

The results are summarized in Figures 6 and 7. Figure 6 and 7 plot the number of candidates of misconfigured parameters and the F-measure performance as a function of the number of minimum instances (corresponding to servers) for each leaf node. Generally, the number of minimum instances in a leaf node (the lowest size of cluster) is large, clustering tends to allow a lot of outliers, while it tends to allow few outliers [4] when the number is small. From Figure 6, we can see that we can reduce the number of candidates of misconfiguration by using pattern modification method. As for the accuracy, it can be seen from Figure 7 that we achieved higher F-measure in most values of minimum instances for each leaf by using pattern modification than without pattern modification. Therefore, we can conclude that the pattern modification contributes to both narrowing down the misconfiguration candidates and improving the identification accuracy.

B. Actual Data

We analyzed the actual data with setting the number of minimum instances for each leaf node to 2 because we achieved highest Precision by this setting in the experiment with artificial data. By executing our misconfiguration identification, we obtained "Precision"=286/364=0.786. This result shows that most (78.6%) of the detected candidates by the proposed method are misconfiguration actually. The result is almost the same in the evaluation with artificial data. Here we did not evaluate "Recall" since it is difficult for administrator to find out all the misconfigurations from too many configuration parameters.

In the evaluation, we detected some "hidden misconfigurations". For example, we identified a misconfiguration in "DNS server" parameter. Here we

identified the majority pattern "IF Region = Tokyo THEN DNS server = jp.dns.com" for the parameter "DNS server". However, we found that one server having an IP address of DNS server (e.g. 192.168.10.1) for the value of this parameter, instead of server name (e.g. jp.dns.com). This inconsistency can cause an inaccessible error if the configuration of DNS server itself is changed.

V. RELATED WORK

Prior to this paper, some works related in misconfiguration detection for configuration management were researched. In [5], the method to describe complex pattern for detection of misconfiguration was proposed. Since in this method the rules for misconfiguration detection must be defined manually, it cannot discover the latent constraints unnoticed by administrators. In [6], a method which can automatically identify the relations among configuration parameters was proposed, while the administrators have to decide the threshold that means even which relations are proper. In [7], the authors proposed a scheme which identifies not only the relation of configuration parameters but also the constraints among parameters and their values. However, this scheme can detect only 33% of misconfigurations, so it cannot achieve high accurate detection of misconfiguration. In [7], for estimating constraints from structures identified by clustering, top-down pattern identification approach is done by preparing four kinds of constraint templates. Therefore, this scheme can estimate only the range assumed beforehand and cannot identify enough patterns to detect misconfigurations. So, this scheme is not suitable for the cloud data center model in which there are complexity relations among parameters.

VI. CONCLUSION

We proposed the misconfiguration detection by finding majority patterns using decision tree analysis. We also added the pattern modification method to handle the case where we cannot determine the majority because servers can be divided into groups with the same size. By using the pattern modification, we obtained higher accuracy of misconfiguration detection than the case without it. In the evaluation, we performed the misconfiguration detection that the most (78.6% in actual data) of the detected candidates are misconfiguration actually.

As our future works, first we are going to compare our approach to existing approaches of misconfiguration detection and examine the advantage of our approach. Then, we are also planning to improve our technique so that it can suggest correct values for misconfigured parameters. This function will allow administrators to manage the configuration of cloud computing systems easier.

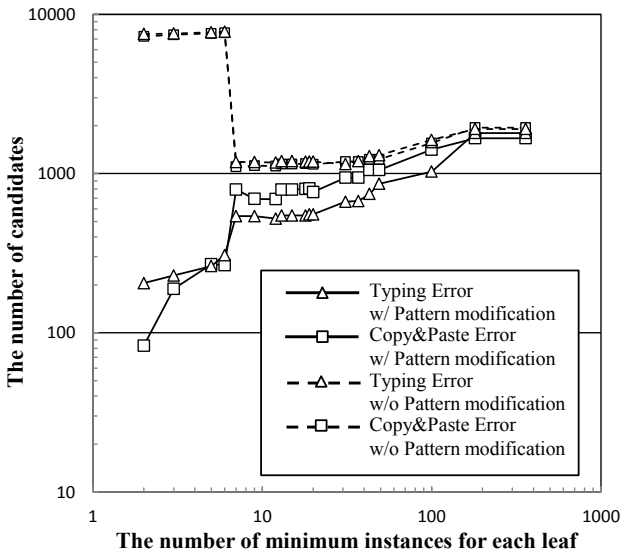

Figure 6. Number of candidates

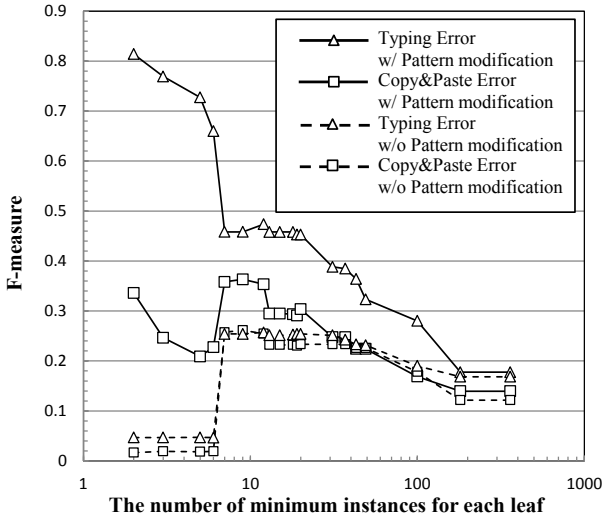

Figure 7. F-measure vs. the number of minimum instances for each leaf

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica and M. Zaharia. "Above the Clouds: A Berkeley View of Cloud Computing." Technical Report No. UCB/EECS-2009-28, University of California at Berkley, USA, Feb. 2009.

[2] J. Baliga , R. W. A. Ayre , K. Hinton and R. S. Tucker "Green cloud computing: Balancing energy in processing, storage, and transport." Proc. IEEE, vol. 99, no. 1, pp.149 -167, Jan. 2011

[3] X. Li, Y. Li, T. Liu, J. Qiu, and F. Wang, "The Method and Tool of Cost Analysis for Cloud Computing." Cloud Computing, 2009. IEEE International Conference on, pp. 93 –100, Sep. 2009.

[4] J. Ross Quinlan. "*C4.5: Programs for Machine Learning*." Morgan Kaufmann, 1993

[5] T. Delaet and W. Joosen, "Podim: A language for high-level configuration management." in Proceedings of the 21st Large Installation Systems Administration Conference. Dallas, pp. 261-273, Nov. 2007

[6] V. Ramachandran, M. Gupta, M. Sethi, and S. R. Chowdhury, "Determining configuration parameter dependencies via analysis of configuration data from multi-tiered enterprise applications." in Proceedings of the 6th international conference on Autonomic computing, pp.169-178, Jun. 2009.

[7] E. Kiciman and Y.-M. Wang, "Discovering correctness constraints for self-management of system configuration," Autonomic Computing, 2004. Proceedings. International Conference on, pp. 28–35, May 2004.

[8] C. van Rijsbergen and P. D, "Information retrieval," 1979.

[9] The NIST Definiction of Cloud Computing, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[10] Quinlan, J. R. "Induction of Decision Trees." Mach. Learn, 1986.