International Conference on Computing Convergence and Applications(ICCCA)2017, August 2017

Multiple Concurrency Anomalies Classification for Mobile Applications using Support Vector Machine

Zhiqiang Wu, Asad Abbas, and Scott Uk-Jin Lee[*]

Department of Computer Science and Engineering, Hanyang University, 15588, South Korea
[Email :hhhwwwuuu@hanyang.ac.kr, asadabbas@hanyang.ac.kr, scottlee@hanyang.ac.kr]

Abstract

Mobile applications are integral part of daily life due to their portability and convenience. In recent research, mobile applications are facing the uniqueness of approach for specific anomaly and large number of false positive. In this study, we propose Support Vector Machine (SVM) based concurrency anomaly classification approach to dynamically distinguish the status in runtime. By using anomaly classification, the approach is enabled to classify multiple concurrency anomaly with vector clocks. We proposed anomalies classification for mobile applications to detect the potential exception and reduce the false positive in runtime.

Index Terms: Classification, Concurrency Anomaly, Mobile Applications, Support Vector Machine (SVM)

I. INTRODUCTION

Presently, mobile devices have become an important part of our daily lives because they allow people to access enormous number of ubiquitous services. Android OS plays a major role in the OS environment for mobile devices, such as smart phones and tablets, given that Android OS is open source [1].

Current OS in mobile systems are very different from the OS of the traditional personal computers due to limited resources and event-based programs. Therefore, mobile applications have the possibility to generate concurrency anomalies because of the non-deterministic execution and difficult to recur by same operations. The recent researches focus on one of concurrency anomaly in each approach. Their approaches lead to huge number of false positive and increased system load for each anomaly. Thus, Hsiao, et al. [2] attempted to prune the false positive to improve the accuracy of anomaly detection.

In this paper, we propose a classification approach for two kinds of concurrency anomalies i.e., deadlock [3] and data race [4], which are the most common anomalies in concurrent systems. We deduce the execution sequences of deadlock and data race based on event-driven applications in mobile platforms. These execution sequences will convert to vectors as training examples for classification. Then, Support Vector Machine (SVM) model with Gaussian kernel will be applied to classify the distinct categories by features of anomalies in Octave. Using vector clocks and control flow graphs (CFG), the training examples are formalized to a one-dimension vector as pre-processing for simulation. The result shows that the prediction accuracy is 80%. The concurrency anomaly can be efficiently exposed by this method with lower false positive.

The rest of this paper is organized as follows: section II describes the related work of this study; section III describes the implementation of the classification of concurrent anomalies and related simulation results. Section IV gives conclusion and future work of this study.

II. RELATED WORK

Recently, many tools and algorithms are made available to detect and solve concurrency anomalies such DEvA [5] and Amandroid [6]. Li, et al. [7] proposed that a strategy for events scheduling can only schedule events with a certain probability. In other work, the data race is fully exposed by RacerDroid [8] from the given potential data races which can generate a large number of false positive.

The existing techniques can expose many concurrent anomalies in concurrent systems. However, the false positive always exists in the results of detection, and these techniques usually focused on one of concurrency anomaly using a unique tool. Our proposed approach covers multiple concurrency anomalies using machine learning algorithm, which is effective to prune the false positive of detection.

III. IMPLEMENTATION

We have implemented our proposed method by a sample application of multi-players game of dice. For instance, there are only two actions in this dice game. Two buttons which have the same action are allowed to be clicked for generating a random integer number between 1 and 6. Moreover, two buttons which represent two events in this application have respective function *OnClickListener()* to generate a random number by variable random. Then, the source code without locks can trigger data race due to the non-deterministic executions.

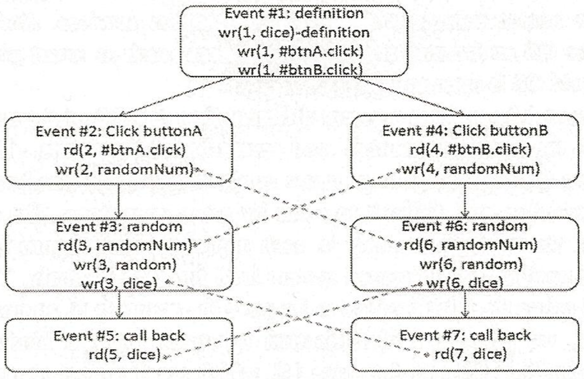

Fig. 1. An event semantic graph for the application

A. Execution Sequences Modeling

We applied the Android source code to generate the CFG, as shown in Fig. 1. If a user clicks two buttons with enough speed simultaneously, two events will be triggered with the miniature lag time. The triggered sequence of events is unknown and non-deterministic, so that all possible sequence will be enumerated in the CFG. Each path represents an execution path that is a set of instructions in the event by Depth First Search (DFS). Thus, the instructions can be interleaving processed in systems when multiple events triggered. The data race will be occurred, when variables are shared by two or more events and when any given multiple writing operations to a shared variable are synchronized. Therefore, there exist four potential data races in Fig. 1 which are denoted using blue dotted line.

B. Vectorization

We manually generated CFG for the sample application and explained how data race occurs in the execution. For classifying the concurrent anomalies using machine learning, the input features of concurrent execution should be presented as one-dimension vector. The vector clocks are used to convert execution sequence to a matrix. While the event executed the *i-th* node, the vector clock of this node that represented by bit code should be updated by equation (1) for each executable path as shown in Fig. 2.

$$VC_i(j) := VC_i(j) + 1 \qquad (1)$$

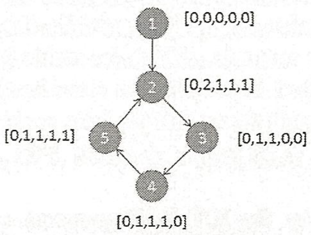

Fig. 2. The Vector Clock of Deadlock Example

The vector clock for each node can be easily generated by using eq (1). Each vector clock represents an executable sequence, which also denotes a specific happens-before relation. All vector clock from the same path will be merged into a matrix. Therefore, this step will generate high number of matrix for classification including anomalous sequences. According to this matrix, the happens-before relation can be determined easily. But this matrix cannot be an input for classification in SVM. The multiple dimensional matrix cannot be inputted as a feature vector in the SVM model. Therefore, *n-1* (n denotes the number of vector clocks) vector clocks will be generated by this method. The summation formula can be defined as:

$$V_{new} = \sum_{i=2}^{n} VT(i) - VT(i-1) \quad iff \quad VT(i) \geq 0 \qquad (2)$$

The equation (2) strengthens the CFG for applications, which will enable the simplification of the feature vector for the classification section. The vector clock VT_{new} is a one-dimension vector as feature input.

C. Classification

The classification methods used SVM with the Gaussian kernel Radial Basis Function (RBF) kernel. This work will classify three classes of anomalies in mobile applications, those belong to multi-class classification in SVM with multiple features. However, the one-vs-one method of SVM for multi-classes is used in this work. We generate *k(k-1)/2* (*k* denotes the number of classes) SVM models for three anomalies (i.e. normal, deadlock and data race). Nevertheless, we can get three results from different models. The result of prediction will be countered automatically. Then, the anomaly status that is the maximum of these number is the final predicted result. The simulation results shown in Fig. 3.

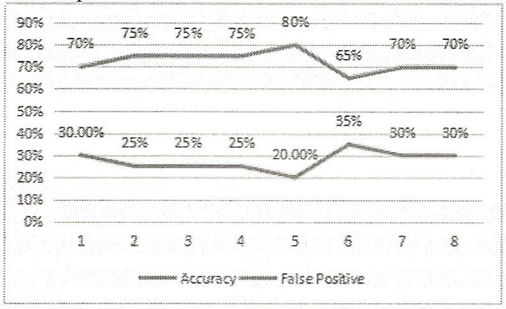

Fig. 3. Classification accuracy of SVM

IV. CONCLUSION AND FUTURE WORK

According to CFG and vector clocks, all executable path will be converted to a one-dimension vector for classification in SVM. Meanwhile, the SVM constantly collects data from testing set for training example, which is enabled to improve the accuracy of prediction. Our proposed method is effective on classification of concurrency anomalies for mobile applications. The result shows that the false positive is reduced to 20%.

ACKNOWLEDGMENTS

This work was supported by the ICT R&D program of MSIP/IITP [2014-0-00670, Software Platform for ICT Equipment].

REFERENCES

[1] É. Payet and F. Spoto, "Static analysis of Android programs," Information and Software Technology, vol. 54, no. 11, pp. 1192-1201, 2012.

[2] C. H. Hsiao et al., "Race detection for event-driven mobile applications," pp. 326-336, 2013.

[3] B. Glatz, R. Beneder, M. Horauer, and T. Rauscher, "Deadlock detection runtime service for Embedded Linux," in Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on, 2015, pp. 1-7: IEEE.

[4] J. Fiedor, B. Křena, Z. Letko, and T. Vojnar, "A uniform classification of common concurrency errors," Computer Aided Systems Theory–EUROCAST 2011, pp. 519-526, 2012.

[5] G. Safi, A. Shahbazian, W. G. Halfond, and N. Medvidovic, "Detecting event anomalies in event-based systems," in Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 25-37: ACM.

[6] F. Wei, S. Roy, and X. Ou, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 1329-1341: ACM.

[7] Q. Li et al., "Effectively Manifesting Concurrency Bugs in Android Apps," in Software Engineering Conference (APSEC), 2016 23rd Asia-Pacific, 2016, pp. 209-216: IEEE.

[8] H. Tang, G. Wu, J. Wei, and H. Zhong, "Generating test cases to expose concurrency bugs in android applications," in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016, pp. 648-653: ACM.