A Novel Framework for Java Based Data Race Detection using ECC-EVKM-BERT and SS-LSGRU with KL-FBIS

Devesh Lowe[1], Mithilesh Kumar Dubey[2]

Jagannath University, Jaipur, Rajasthan, India[1]
Lovely Professional University, Punjab, India[2]

Corresponding author: Devesh Lowe, Email: devesh.lowe@jimsindia.org

When two or more threads visit a shared variable concurrently and at least one of those accesses is a write operation without the necessary synchronization mechanisms in place, it's known as a data race in multi-threaded algorithms. Unpredictable behavior, such as damaged data, software crashes, and inconsistent outputs, may result from this. Since data races can appear inconsistently based on thread scheduling and time, they are infamously hard to debug. Data races can have serious consequences, such as unstable software that responds erratically to various stimuli. They jeopardize data integrity, which can lead to subtle errors that are challenging to find and correct. Furthermore, they may give rise to security flaws since unauthorized parties may be able to take advantage of the access to data. Numerous methods, including hybrid methods, dynamic analysis, and static analysis, are used to identify data races. Deep Learning algorithms, recently, have proved to be an effective way to identify new data races more accurately. Through the identification of intricate patterns and anomalies that conventional methods might overlook, these techniques can be used to examine big codebases and execution patterns in order to understand how to spot data races. To improve the accuracy and recall of detection efforts, models that predict the likelihood of data races can be trained using labeled datasets including multi-threaded code. In this work, the authors have proposed a framework for identifying data races using Java as implementation language, Galois Chameleon Swarm Optimization Algorithms for identification of test cases, K-Means for efficient clustering process, Linear Scaling Gated Recurrent Unit for classification accuracy. The proposed model thrives to reduce false positives and false negatives in identifying data races.

Keywords: Dynamic Data Race Detection, Machine Learning, Machine Learning, Deep Learning.

2025. In Chetan Sharma, Vaishali Maheshwari & Harish Sharma (eds.), *Advancements in Intelligent Systems*, 17–23. Computing & Intelligent Systems, SCRS, India.
DOI: https://doi.org/10.56155/978-81-975670-3-2-2

1 Introduction

Recently, modern programming languages with a more object-oriented philosophy have seen the light. One of these is the Java programming language designed by Sun at the beginning of the 1990s. Java was designed with multi-threading in mind. When applying a multi-threaded design to a programming task, a task is split into several threads. Each of these subtasks can be executed on a separate processor, increasing parallelism and speed. A group of threads can be dedicated to respond to external events, so a very fast response time can be obtained using this design. Together with the advantages of multi-threaded design, a new type of bug has appeared that is, data race. The data race occurs in a multithreaded program when two threads access the same memory location with no ordering constraints enforced between the accesses, such that at least one of the accesses is a write [1].

In many cases, a data race is a programming error. Furthermore, programs with data races are notoriously difficult to debug because they can exhibit different behaviours when executed repeatedly with the same set of inputs. In general, techniques are proposed to detect data races in one particular execution and not in all possible executions of a program. To detect the data race in JAVA programs, already two tools exist that detect data races in particular executions of Java programs: AssureJ and JProbe [2]. But these tools had limitations due to overhead problems. A time overhead of at least a factor of 10 is common. This is smaller than general programs because each thread in Java has a local stack. The values on this stack can only be manipulated by the thread owning the stack. Therefore, all read and write operations performed on this stack need not be observed to find data races. Also, the data race detection tools used in the existing models has many false positives and false negatives. Hence, to overcome these problems, the proposed model is designed based on the LSGRU[8] model with ECC-KVM-BERT and FBIS techniques.

2 Literature Survey

Researchers have presented context-sensitive data race detection techniques for concurrent programs [3]. In this technique, call graphs were constructed using control-flow analysis, and the escaped objects among the threads were determined in the escape analysis phase. Further, alias-objects were determined via context-sensitive alias analysis. Thus, the usage of context-sensitive analysis reduced the false positives and false negatives rate.

Yu et al. developed a noise injection-based race detection technique [4]. Here, the hidden-race candidates were collected using an imprecise hybrid detector, and the thread execution was delayed using read-exclusive, write-exclusive, and read-shared non-racing access types. Further, the normal locking sequences were disturbed between the threads using a race inducer. Experimental results proved the efficacy of the presented race detection technique.

Ding et al. employed Wed Service-Business Process Execution Language (WS-BPEL) for effective harmful data race detection [5]. Initially, potential data races were obtained via static detection, and then the false positives were statically pruned followed by the generation of test cases. Thus, the benign data races were effectively identified with better accuracy and reduced false negatives.

Liao et al. suggested a pending period representation-based scalable data race detection method [6]. In this methodology, a leverage global clock was utilized for representing the execution orders between the events. After obtaining a succession of relaxed pending periods, the desired data races were detected using the race detection algorithm. Hence, the occurrence of the off-the-shelf synchronization problem in the race detection phenomena was effectively eradicated.

Jia et al. propounded an adaptive dynamic data race detection approach using a multi-virtual machine method [7]. In this method, Selective Dynamic Analyzer(SDA)-cloud technique was used for detecting

data race. Here, the holistic approach followed by dynamic binary instrumentation acts as a detector in this approach. Thus, the incurring unacceptable slowdown problem was avoided using this technique. Race Detection is crucial for identifying several thread instances in multithreaded programs. Machine learning approaches such as unsupervised learning, supervised learning, and reinforcement learning provide precise detection by effectively managing intricate interactions within concurrent programs, as compared to traditional static or dynamic analytic methods [13] [14]. Machine learning techniques improve the accuracy of detection by leveraging lessons from past data, forecasting possible races, and adjusting to emerging patterns of concurrency problems, hence enhancing the dependability and performance of multi-threaded systems.

Jacobson J. et al., 2024 [9] presented a technique for precise and efficient source-level race-checking of GPU applications and algorithms. First and foremost, the program type was determined using CUDA. Subsequently, the memory access types and thread interactions were monitored in order to identify any races by state machines. Furthermore, the shadow memory structure was created by replicating the original data array. Finally, the shadow memory entry was modified to accurately represent the access type and thread connectivity using HiRace. This analysis identified 346 instances of data races among 580 CUDA kernels that were examined. However, due to the requirement for a wide range of test inputs to encompass all program pathways, this model faced challenges in terms of completeness.

The dynamic detection of data mapping problems in diverse OpenMP applications was demonstrated by Yu L, et al., 2021 [10]. Initially, the data mapping issues were identified based on the inaccurate configurations of maps in target offloading constructions. The introduction of the unified memory model in OpenMP 5.0 was aimed at preventing the emergence of data mapping problems. The implementation of ARBALEST facilitated the mapping of each variable to the accelerator and the monitoring of the visibility of the last writing. This process required a constant time for each memory access. This model displayed inaccurate findings in both false positives and false negatives as a result of the absence of available runtime information.

In their study, Chatterjee N. et al. (2023) [11] presented a suite of productivity accelerators that focus on a dynamic instrumentation approach for DRD. This study presents the implementation of PARALLELC-ASSIST for the purpose of detecting concurrency-related bugs in C multi-threaded applications. Furthermore, this framework offered user-friendly interfaces for developing several other analytical tools specifically designed for C code. Ultimately, based on the interaction between threads and resources, the first set of tools was used to troubleshoot and extract design components relevant to concurrency, resulting in a recall rate of 98.56%. However, due to the absence of source code analysis, this model was unable to adequately manage the macro on run-time binary instrumentation.

In their study, Karchi J. E. et al. (2024) [12] demonstrated the method of prior knowledge-centric augmented self-supervised feature learning for the purpose of few-shot intelligent defect diagnosis of machines. In order to establish a preliminary collection of features, 24 signal feature indicators were constructed. The conventional auto-encoder was then employed to extract the general features. Finally, the diagnostic model was trained using the self-supervised learning approach, resulting in an accuracy of 91.5%. The implementation of this model was necessary in order to improve the performance and address the limited number of training data.

3 Problem Identification

Existing research methodologies have some drawbacks, which are described as follows. The existing techniques do not handle dynamic control flow, does not support target offloading constructs, and does not support OpenMP tasks and are often subject to false positives and false negetives.Existing research has mostly focused on data race detection at the user level and significant progress has been made in this regard. Most of the existing works did not focus on identifying harmful data races from benign data

races.Most dynamic detection techniques are based on limited input sets for program execution. Due to the lack of effective program input generation methods, the program paths that can be analysed are limited. And, some data races may be hidden in the path without being detected, resulting in false negatives.The main aim of this paper is to design an efficient framework for Data Race Detection in Java Application Programming language using ECC-EVKM, BERT, LSGRU WITH KL-FBIS. All the enhancements are described below.

The objectives of this research work is to propose a model for effective data race detection by adopting and enhancing the existing processes of test case selection, clustering and windowing. To improve the process of efficient test case selection, the Galois - Chameleon Swarm Optimization Algorithm (G-CSOA) will be used. To perform efficient clustering process, Eisen Cosine Correlation distance-based Entropy Variance KMeans (ECC-EVKMeans) clustering technique will be used.To perform an efficient windowing process, KullbackLeibler Divergence-based Fuzzy Bayesian Inference System (KL-FBIS) will be used. To improve the classification accuracy of Data Race Detection, the SoftSwish – Linear Scaling Gated Recurrent Unit (SS-LSGRU) Algorithm will be used.To compare the proposed technique with the existing technique using the result parameters like precision, recall, F-measure, accuracy, clustering time, fitness vs iteration and test case selection time, etc.

4 Proposed System

Data races have become one of the severe issues affecting the correctness of concurrent programs. Existing detection tools for data races have many false positives and false negatives. To improve the correctness, the proposed system was designed. The proposed system will start with Java Programs. Next, the no. of variables, methods, and variable types will be extracted. Next, this information will be given to the Thread Analysis phase. This phase is mainly used to analyze the context-sensitive information of java source code. It has three parts, Control flow Analysis: Control-flow analysis is the basis of our data race detection method, which is a static-code-analysis technique for determining the control flow of a program. This will be done by using a Control flow graph from java code.

Next, Escape Analysis: Escape Analysis is a method for determining the dynamic scope of pointers. Finally, Compositional Pointers: It is designed to analyze each method once to produce a single parameterized analysis result that can be specialized for use at all of the call sites that may invoke the method. This escape analysis and compositional pointers are find out from variables and methods.

Parallelly, the test cases will be generated from the java code. After that, the important test cases will be selected by using Galois - Chameleon Swarm Optimization Algorithm (G-CSOA). The existing Chameleon Swarm Optimization Algorithm has a fixed value of the scaling factor in eye rotation. This will lead to the creation of the wrong rotation matrix. In order to solve this problem, the Galois technique will be included in the existing Chameleon Swarm Optimization Algorithm. And also, this modified algorithm will be compared with existing Meta-heuristics algorithms using the result parameters such as fitness vs iteration and test case selection time.

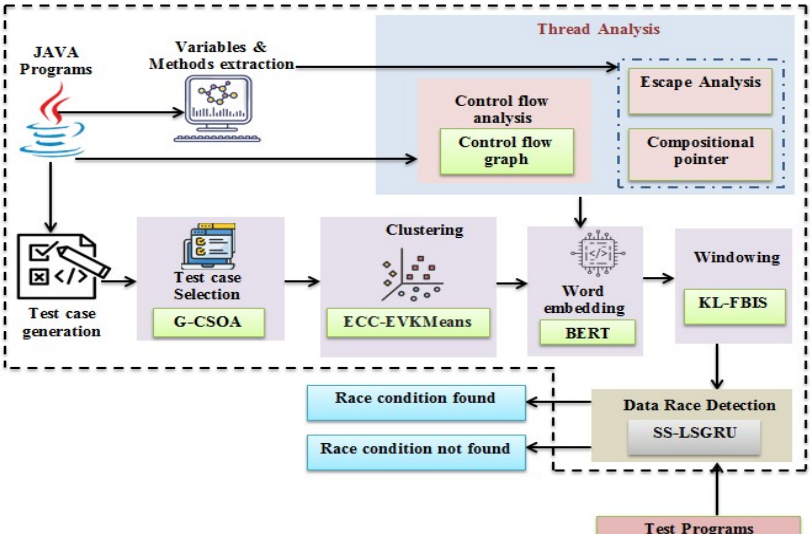

Figure 1. Framework for Proposed System

At the end of this, the optimized test cases and thread analysis results will be given to the word embedding phase. This phase will be mainly used to return the vector value for corresponding words and it will be done by using BERT Algorithm. Here, to solve the problem of unstructured data training for misclassification, the Eisen Cosine Correlation distance-based Entropy Variance KMeans (ECC-EVKMeans) clustering technique will be done before word embedding process. In this, to improve the clustering accuracy, Eisen Cosine Correlation distance will be used. And also to solve the problem of variance that is it gives added weight to outliers in existing kmeans algorithm; Entropy Variance technique will be used. Finally, this proposed modified algorithm will be compared with the existing clustering algorithm using the result parameter such as clustering time.

Next, the windowing process will be handled. This phase will be mainly used to reduce the use of nested loops and replace it with a single loop based on the rules, thereby reducing the time complexity and separating the codes. This phase will be done by using KullbackLeibler Divergence-based Fuzzy Bayesian Inference System (KL-FBIS). In the existing fuzzy system, a large number of non-feasible rules will be generated due to the fuzzy inference system. In order to solve this issue, KullbackLeibler Divergence Bayesian technique will be included in the existing Fuzzy algorithm. And, this proposed modified algorithm will be compared with the existing fuzzy algorithm using the result parameter such as rule generation time, fuzzification time and defuzzification time, etc.

Finally, these separated vector values will be trained by using SoftSwish – Linear Scaling Gated Recurrent Unit (SS-LSGRU) Algorithm. In this, to improve the classification accuracy, the SoftSwish activation function will be used in the existing Gated Recurrent Unit deep learning technique. And also, to reduce the loss value and solve the gradient vanishing problem, the linear scaling technique will be included in the existing Gated Recurrent Unit deep learning technique. In this, the proposed algorithm will be compared with the existing deep learning algorithms such as Long Short Term Memory Networks (LSTM), Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), and Gated Recurrent Unit (GRU) using the result parameter such as precision, recall, f-measure, and accuracy, etc.
The proposed framework is illustrated in Figure 1.

5 Experimental Setup

The dataset used in the proposed framework is “data_race_bugs”. These datasets are collected from publicly available sources using the below link. https://github.com/misunyu/data_race_bugs. The implementation is done using JAVA and the result proves that our proposed method is better than many existing techniques. Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "Write Once, Run Anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to byte code that can run on any Java virtual machine (JVM) regardless of computer architecture.
Hardware requirements

- Processor : Intel i5/core i7
- CPU Speed : 3.20 GHz
- OS : Windows 10
- System Type : 64 bit
- RAM : 8GB

6 Conclusion

Finally, a thorough framework for dynamic data race detection has been presented in this article, which outlines the basic approach and methodology without getting into the details of the implementation or actual code. Through the detection and mitigation of data races, the suggested framework claims to improve the dependability and efficiency of multi-threaded programs. The framework will be put into practice, a lot of testing will be done, and its efficacy will be compared to other alternatives in the future. Further research and development should focus on investigating the integration of deep learning techniques for enhanced accuracy and scalability.

References

[1] D. Lowe, B. Galhotra, and M. Kumar Dubey, “Data Race Detection Techniques In Java: A Comparative Study,” INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH, vol. 9, p. 2, 2020, [Online]. Available: www.ijstr.org

[2] J. J. Bai, Q. L. Chen, Z. M. Jiang, J. Lawall, and S. M. Hu, “Hybrid Static-Dynamic Analysis of Data Races Caused by Inconsistent Locking Discipline in Device Drivers,” IEEE Transactions on Software Engineering, vol. 48, no. 12, pp. 5120–5135, Dec. 2022, doi: 10.1109/TSE.2021.3138735.

[3] Y. Zhang, H. Liu, and L. Qiao, “Context-Sensitive Data Race Detection for Concurrent Programs,” IEEE Access, vol. 9, pp. 20861–20867, 2021, doi: 10.1109/ACCESS.2021.3055831.

[4] M. Yu, Y. S. Ma, and D. H. Bae, “Efficient noise injection for exposing hidden data races,” Journal of Supercomputing, vol. 76, no. 1, pp. 292–323, Jan. 2020, doi: 10.1007/s11227-019-03031-0.

[5] Z. Ding and Z. Zhou, “RaceTest: harmful data race detection based on testing technology in WS-BPEL,” Service Oriented Computing and Applications, vol. 13, no. 2, pp. 141–154, Jun. 2019, doi: 10.1007/s11761-019-00261-1.

[6] X. Liao, M. Lin, L. Zheng, H. Jin, and Z. Shao, “Scalable Data Race Detection for Lock-Intensive Programs with Pending Period Representation,” IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 11, pp. 2599–2612, Nov. 2018, doi: 10.1109/TPDS.2018.2836899.

[7] C. Jia, C. Yang, W. K. Chan, and Y. T. Yu, “SDA-CLOUD: A Multi-VM Architecture for Adaptive Dynamic Data Race Detection,” IEEE Trans ServComput, vol. 10, no. 1, pp. 80–93, Jan. 2017, doi: 10.1109/TSC.2016.2596288.

[8] M Sharma, R Bagoria, P Arora, "Hybrid CNN-GRU Model for Handwritten Text Recognition on IAM, Washington and Parzival Datasets", IEEE2023 2nd International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN), doi10.1109/ICSTSN57873.2023.10151552

[9] Jacobson J., Burtscher M., and Gopalakrishnan G., "HiRace: Accurate and Fast Source-Level Race Checking of GPU Programs", arXiv preprint arXiv:2401, 2024, pp. 1-11. https://doi.org/10.48550/arXiv.2401.04701

[10] Yu L., Protze J., Hernandez O., and Sarkar V., "ARBALEST: Dynamic Detection of Data Mapping Issues in Heterogeneous OpenMP Applications", IEEE International Parallel and Distributed Processing Symposium, 17-21 May, 2021, Portland-OR,USA. https://doi,org/10.1109/IPDPS49936.2021.00055

[11] Chatterjee N., Majumdar S., Das P. P., and Chakrabarti A., "PARALLELC-ASSIST: Productivity Accelerator Suite Based on Dynamic Instrumentation", IEEE Access, Vol. 11, 2023, pp. 73599– 73612. https://doi.org/10.1109/ACCESS.2023.3293525

[12] Karchi J. E., Chen H., TehraniJamsaz A., Jannesari A., Popov M., and Saillard E., "MPI Errors Detection using GNN Embedding and Vector Embedding over LLVM IR", arXiv preprint arXiv:2403, 2024, pp. 1-13. https://doi.org/10.48550/arXiv.2403.02518

[13] Zhang T., Chen J., He S., and Zhou Z., "Prior Knowledge-Augmented Self-Supervised Feature Learning for Few-shot Intelligent Fault Diagnosis of Machines", IEEE Transactions on Industrial Electronics, Vol. 69, No.10, 2022, pp. 10573-10584. https://doi.org/10.1109/TIE.2022.3140403

[14] Xu Z., Shen J., Luo P., and Liang F., "PVcon: Localizing Hidden Concurrency Errors With Prediction and Verification", IEEE Access, Vol. 8, 2020, pp. 165373-165386. https://doi.org/10.1109/ACCESS.2020.3022992