Scientific Journal Impact Factor (SJIF): 1.711
e-ISSN: 2349-9745
p-ISSN: 2393-8161
International Journal of Modern Trends in Engineering and Research
www.ijmter.com

# Association Rule Mining Scheme for Software Failure Analysis

Ms. R. Gowri[1], Dr. M. Latha[2], Mr. R. Subramanian[3]
[1]*Research Scholar*
[2]*M.Sc., M.Phil., Ph.D, Associate Professor, Department of Computer Science*
[1,2]*Sri Sarada College for Women, Salem, Tamilnadu, India*
[3]*Erode Arts and Science College, Erode*

**Abstract-**The software execution process is tracked with event logs. The event logs are used to maintain the execution process flow in a textual log file. The log file also manages the error values and their source of classes. The error values are used to analyze the failure of the software. The data mining methods are used to evaluate the quality and software failure rate analysis process. The text logs are processed and data values are extracted from the data values. The data values are mined using the machine learning methods for failure analysis.

The service error, service complaints, interaction error and crash errors are maintained under the log files. The events and their reactions are also maintained under the log files. Software termination and execution failures are identified using the log details. The log file parsing process is applied to extract data from the logs. The associations rule mining methods are used to analyze the log files for failure detection process. The system uses the Weighted Association Rule Mining (WARM) scheme to fetch failure rate in the software execution flow. The system improves the failure rate detection accuracy in WARM model.

## I. INTRODUCTION

Software fault tolerance is the ability of computer software to continue its normal operation despite the presence of system or hardware faults. The only thing constant is change. This is certainly more true of software systems than almost any phenomenon, not all software change in the same way so software fault tolerance methods are designed to overcome execution errors by modifying variable values to create an acceptable program state. The need to control software fault is one of the most rising challenges facing software industries today. It is obvious that fault tolerance must be a key consideration in the early stage of software development.

There are different methodologies approaches in developing a well quality software tolerance from different angles by different researchers but the most widely used methods. N-versioned programming is based on hardware fault tolerance techniques which compare the output of duplicate hardware module run in parallel in an attempt to minimized coincidental failure caused by common logical flaws. Recovery blocks is based on software adaptation of the hardware fault tolerance techniques in which a set of redundant block is created, each of which create a comparable results. Robust data structure methods is quite suitable compare to the N-versioned and recovery blocks because it allow user- defined structure such as list and trees, to be detected and corrected. Periodically a particular detection algorithm examines a structure for errors.

## II. RELATED WORK

Logs are human-readable text files that report sequences of text entries ranging from regular to error events. A typical log entry contains a time stamp, the identifier of the source of the event, a severity and a free text message. Well-known logging frameworks are UNIX Syslog and Microsoft Event Logging.

### 2.1. Log-Based Failure Analysis

Over the years, several software packages have emerged to support log-based failure analysis, integrating state-of-theart techniques to collect, manipulate, and model the log data, e.g., MEADEP, Analyze NOW. Log-based analysis is not yet supported by fully-automated procedures, leaving most of the processing burden to log analysts, who often have a limited knowledge of the system. For instance, the authors define a complex algorithm to identify OS reboots from the log based on the sequential parsing of log messages. Furthermore, since a fault

activation can cause multiple notifications in the log a significant effort has to be spent to coalesce the entries related to the same fault manifestation [10]. Preprocessing tasks are critical to obtaining accurate failure analysis. We aim to obtain accurate logs which can be analyzed without needing to perform laborious preprocessing activities.

Despite efforts on log processing and analysis, logs are known to suffer from incompleteness and inaccuracy issues. For example, the analysis of a set of networked Windows NT workstations highlighted that 58 percent of reboots remained unclassified because a clear cause of the reboot was not reported in the logs. The authors show that the built-in detection mechanisms of the JVM are not capable of providing any evidence for around 45 percent of failures. Even more important, current logs are often ineffective in the case of software faults, recognized to be among the main causes of system failures. For instance, in the context of web applications [3] it has been observed that although logs can detect failures caused by resource exhaustion and environment conditions, they provide limited coverage of software failures caused by concurrency faults, e.g., deadlocks. In our earlier study [2], we demonstrated that around 60 percent of failures caused by the activation of software faults go undetected by current logging mechanisms.

### 2.2. Log Production Schemes

Several solutions have been proposed to address the inefficiencies of logs. For example, IBM Common Event Infrastructure [6] provides APIs and infrastructure for the creation, persistence, and distribution of log events. Apache log4x, e.g., [9], is a configurable environment to collect log events. These frameworks are valuable because they allow saving the time needed to collect, parse, and filter logs; they mainly address log format issues rather then addressing the problem of designing the logging mechanism.

A more systematic approach to log management is provided by software systems relying on the Aspect-Oriented Programming (AOP) paradigm, where logging can be treated as a system-wide feature orthogonal to other services or to the business logic. For instance, through aspect weaving, a log entry can be systematically produced for each runtime exception, supporting system-wide and application transparent exception reporting. It is argued that the correct aspect-oriented exception management and logging can lead to more reliable software. Aspect-oriented logging requires the adoption of an AOP framework, which may not be the case for several software industries. We aim to conceive a set of rules for log-based failure analysis that can be applied independently from a particular programming framework.

Other solutions, introduce a set of recommendations to improve the expressiveness of the logs. Similarly, Yuan et al. [7] propose to enhance the logging code by adding information, e.g., data values, to ease the diagnosis task in case of failures. These works improve the invocations of logging functions that already exist in the software platform. Nevertheless, incompleteness of the logs cannot be solved acting solely on the existing functions: Developers might forget to log significant events, and, in many cases, errors escape existing logging points. Finally, an approach to visualize console logs is proposed in [8]: The authors describe how to obtain a graph that can be used to improve the logging mechanisms, e.g., by adding missing statements. Differently from our objective, the improvement of the logging mechanism aims to enhance the semantic rather than the ability of the log at detecting errors.

### 2.3. Methods for Failure Detection

For the above reasons, other techniques are adopted to detect and analyze failures, such as runtime failure detection, executable assertions, or software tracing. Runtime failure detection consists of observing, either locally or remotely, the execution state of the system. It can be implemented either by means of query-based techniques, e.g., sending heartbeat messages, or by exploiting hardware support where available, such as watchdog timers. Other solutions rely on continuous monitoring of the status of system variables and on the comparison of these data with traces of normal and anomalous executions. Executable assertions, usually adopted in the embedded systems domain, are check statements performed on the program variables to detect application-specific content errors, e.g., invalid variable values for the given function. Software tracing solutions are widely adopted to monitor the execution of a software system, e.g., to perform Function Boundary Tracing (FBT), which registers function entry and exit events. They rely on software instrumentation packages, such as DTrace or Kerninst, usually working at the binary level, and hence with no or limited knowledge on the structure of the

 

system. Although FBT is typically adopted for performance evaluation or reverse engineering, it can be also adopted to detect system malfunction.

These diverse techniques are extremely valuable to detect given classes of failures; their aim is not formalizing the implementation of a comprehensive logging mechanism. The novel aspect of our proposal is to leverage design artifacts to infer a logging mechanism that supports the analysis of software failures. Design artifacts allow identifying errors that potentially lead to failures. A set of rules establishes how the logging mechanism must be implemented to detect such errors, i.e., in terms of instructions to be placed in the source code of software. The proposed rules partially leverage benefits of mentioned detection techniques, but, more importantly, formalize the use of such techniques based on an error model that allows achieving effective logs: To the best of our knowledge, this issue has not been addressed so far.

## III. SOFTWARE FAILURES ANALYSIS USING EVENT LOGS

Failure analysis techniques aim to characterize the dependability behavior of operational systems and are used in many industrial sectors, such as aerospace [1] and automotive. These techniques are valuable to engineers: For example, they allow us to understand system failure modes, establish the cause of failures, prevent their occurrence, and improve the dependability of future system releases. Failure analysis is often conducted by collecting event logs i.e., system-generated files reporting events of interest that occurred during operations. Logs are used for a variety of purposes, such as debugging, configuration handling, access control, monitoring, and profiling. Moreover, logs are often the only means to analyze the failure behavior of the system under real workload conditions.

The level of trust on log-based failure analysis depends on the accuracy of the event log. This is clarified by Fig. 3.1, which highlights the relationship between the fault-error failure chain and the event log. The errors ellipse in Fig. 3.1 contains all the activated faults. The subset of errors that reach the system interface is contained by the failures ellipse. Event logs report a subset of errors. Logs might report errors that do not lead to failures and only a fraction of actual failures with many failures remaining unreported. These issues represent a serious threat when logs are used to analyze system failures. Unreported failures may lead to erroneous insights into the behavior of the system. Similarly, reported errors, although useful when logs are used for other purposes, may be misinterpreted as false failure indications if not supported by a detailed knowledge of the system. The focus of this system is on the use of event logs to support accurate failure analysis: To this objective, the event log ellipse and the failures ellipse should perfectly overlap.

Despite a number of works proposing log filtering and coalescence algorithms [4], [10], the real problem with failure analysis is the scarce ability of current logs at reporting the right set of error events, i.e., the ones leading to failures. This is especially true in the case of software faults, which are among the main causes of system failures. Recent studies have recognized the difficulty in analyzing software failures by looking solely at logs and in our earlier study [2] we demonstrated that around 60 percent of failures due to software faults do not leave any trace in logs. The ambition of this work is to fill the gap in the knowledge about the ability of logs to report software failures and to propose a novel logging approach, named rule-based logging, achieving effective failure detection. Although several works have addressed the format and representativeness issues of logs, to the best of our knowledge this is the first contribution addressing the suitability of logs for the analysis of software failures.

 

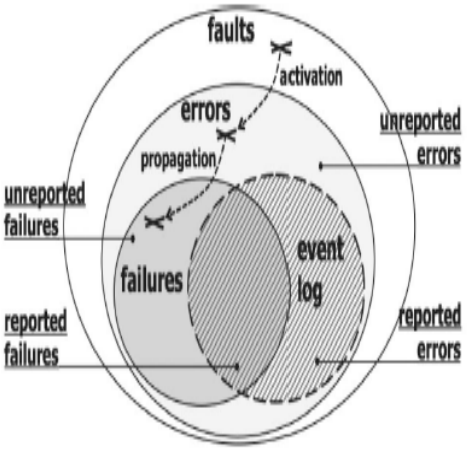


Fig. 3.1. Relationship Between The Fault-Error-Failure Chain And The Event Log

The system provides insights into the deficiencies of current logging mechanisms by analyzing eight successful open-source and industrial projects, accounting for a total of around 3.5 million lines of code (LOC). The analysis revealed that traditional logging mechanisms rely on a simplistic coding pattern and lack a systematic error model. The proposal leverages system design artifacts to define a model encompassing errors leading to failures. A set of rules establishes how the logging mechanism must be implemented to detect such errors. The rule-based logging approach is able to detect failures that escape current logging mechanisms and, if available, to complement the information provided by traditional logs.

The rule-based logging has been applied to two software systems: Apache web server and TAO Open Data Distribution Service (DDS). The improvement of the detection capability of the log is shown by means of around 12,500 software fault injection experiments. The most adopted logging pattern aims to detect errors via a checking code placed at the end of a block of instructions. Rule-based logging significantly improves the failure detection capability of the log: On average, around 92 percent of failures due to software faults are logged by the proposed mechanism. Furthermore, it produces few reported errors. Rule-based logging achieves a high compression rate: On average, the rule-based log is around 150 times smaller than the traditional one. Failures are notified with few lines, thus easing the interpretation of the log.

## IV. PROBLEM STATEMENT

Several software packages have emerged to support log-based failure analysis, integrating state-of-the art techniques to collect, manipulate, and model the log data, e.g., MEADEP, Analyze NOW and SEC. Log-based analysis is not yet supported by fully-automated procedures, leaving most of the processing burden to log analysts, who often have a limited knowledge of the system. For instance, the authors define a complex algorithm to identify OS reboots from the log based on the sequential parsing of log messages. Furthermore, since fault activation can cause multiple notifications in the log, a significant effort has to be spent to coalesce the entries related to the same fault manifestation. Preprocessing tasks are critical to obtaining accurate failure analysis. The system is aimed to obtain accurate logs which can be analyzed without needing to perform laborious preprocessing activities.

Despite efforts on log processing and analysis, logs are known to suffer from incompleteness and inaccuracy issues. For example, the analysis of a set of networked Windows NT workstations highlighted that 58 percent of reboots remained unclassified because a clear cause of the reboot was not reported in the logs. The authors show that the built-in detection mechanisms of the JVM are not capable of providing any evidence for around 45 percent of failures. Even more important, current logs are often ineffective in the case of software faults, recognized to be among the main causes of system failures. For instance, in the context of web applications it has been observed that although logs can detect failures caused by resource exhaustion and environment conditions, they provide limited coverage of software failures caused by concurrency faults. 60 percent of failures caused by the activation of software faults go undetected by current logging mechanisms. The Association Rule

 

Mining (ARM) produces failure information with low accuracy levels. The following problems are identified from the current software failure analysis schemes.

- High scalability is not supported.
- Failure detection accuracy is low
- Event summarization is not optimized
-

## V. FAILURE DISCOVERY WITH EVENT WEIGHTS

The software failure analysis system uses the event logs generated during the software run time. Event logs have been widely used over the last three decades to analyze the failure behavior of a variety of systems. Nevertheless, the implementation of the logging mechanism lacks a systematic approach and collected logs are often inaccurate at reporting software failures: This is a threat to the validity of log-based failure analysis. The system analyzes the limitations of current logging mechanisms and proposes a rule-based approach to make logs effective to analyze software failures. The approach leverages artifacts produced at system design time and puts forth a set of rules to formalize the placement of the logging instructions within the source code. The Association Rule Mining (ARM) method is used to estimate the log patterns with failure details. The Weighted Association Rule Mining (WARM) scheme is used to improve the log analysis for failure detection process.

### 5.1. Association Rule Mining Process

A number of data mining algorithms have been recently developed that greatly facilitate the processing and interpreting of large stores of data. One example is the association rule-mining algorithm, which discovers correlations between items in transactional databases. Priori algorithm is an example of association rule mining algorithm. Using this algorithm, candidate patterns that receive sufficient support from the database are considered for transformation into a rule. This type of algorithm works well for complete data with discrete values.

### 5.2. Rule-Based Logging

Current logging practices often postpone the insertion of the logging code at the last stages of the development cycle, according to inefficient patterns and with no knowledge about the system structure. Differently from current strategies, the proposal leverages system design artifacts to define a systematic error model supporting the log-based failure analysis of a given system. The implementation of the logging mechanism is formalized in terms of rules that allow detecting errors based on the model. The approach ata- glance, denoted as rule-based logging. The key idea is leveraging high- and/or low-level artifacts available at design time to infer a system representation model. The model identifies a set of entities in the system and interactions among them. A set of logging rules (LR), inspired by an error model, drives the unambiguous insertion of the logging instructions within the code of identified entities. The logging rules are conceived to ensure that the data needed to perform the failure analysis are provided by the event log.

### 5.3. Weighted Association Rule Mining (WARM) for Log Analysis

The weighted association rule mining (WARM) model is used to estimate the failure relationship with log event details. The association rule mining model uses the support and confidence for the rule selection process. The errors are assigned with associated importance based weight values. The weighted support and weighted confidence values are used to filter the rules from the weighted data values. The system improves the failure rate detection process with high accuracy levels.

## VI. SOFTWARE FAILURE ANALYSIS USING ARM

The software failure analysis based event log system is designed to fetch log failure details. The association rules mining and weighted association rule mining techniques are used in the system. The system is divided into four major modules. They are product, class, logs and failures modules. The product module is used to collect product information from the user. The class details are extracted from the product folder details. The logs module is used

 

fetch error details from the logs. The failure module is used to apply ARM and WARM techniques for failure detection process.

### 6.1. Product Module

The Product module is designed to fetch the product name and path. The classes and their size details are displayed in the product details window. The system also displays the product description in a separate window. The user can select and close the projects dynamically.

### 6.2. Class Module

This module is designed to collect the system information. Each system is composed with a set of classes. A class can be formed with a set of attributes and methods. The class module has three sub modules. They are the overall class information, attribute information and the method information. The application maintains a separate file for the system information.

### 6.3. Logs Module

The logs module is designed to maintain the software execution logs. The execution events are updated in the log files. Different types of errors are maintained under the logs. The service error, service complaints and interaction errors are highlighted in the logs. The log analysis process produces the summary of each error under each class in the product.

### 6.4. Failure Module

The software failure module is designed to identify the failure rate in the software logs. The log event details are analyzed with association rule mining and weighted association rule mining methods. The WARM produces accurate failure levels in each class execution process.

## VII. PERFORMANCE ANALYSIS

The software failure analysis is system is tested with different software products and log files. The Association Rule Mining (ARM) and Weighted Association Rule Mining (WARM) techniques are used in the analysis process. The false positive rate and false negative rate measures are used to evaluate the system accuracy levels. False positive rate analysis details are compared in figure 7.1. The analysis result shows that the WARM reduces the false positive error rate 30% than the ARM technique. The false negative error rate analysis is compared in figure 7.2. The analysis result shows that the WARM model reduces the error rate 25% than the ARM technique.

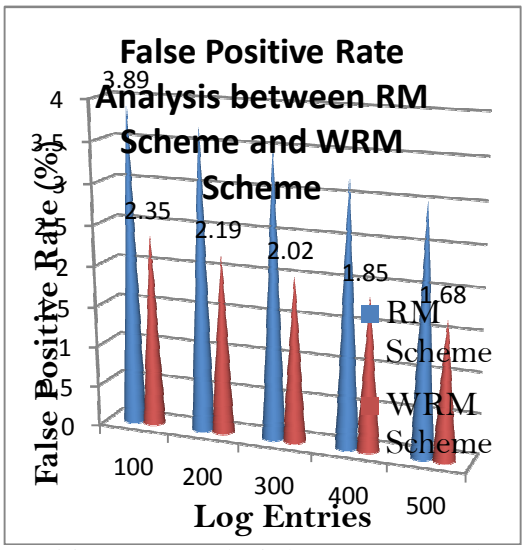


Figure No: 7.1. False Positive Rate Analysis between RM Scheme and WRM Scheme

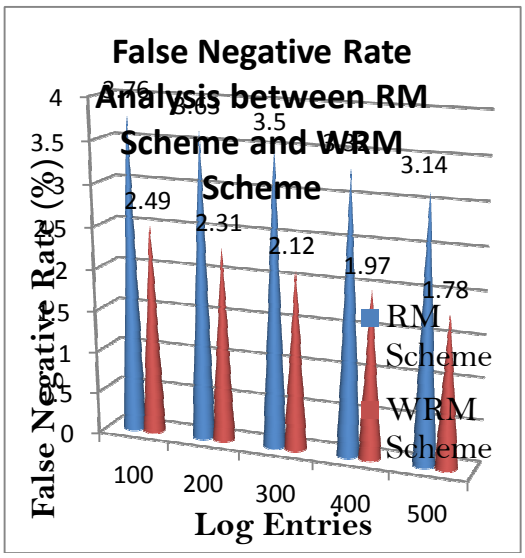


Figure No: 7.2. False Negative Rate Analysis between RM Scheme and WRM Scheme

## VIII. CONCLUSION AND FUTURE WORK

The Association Rule Mining (ARM) technique is used to evaluate the logs to detect software failures. The Apriori algorithm is used to identify the failure cases. The Weighted Association Rule Mining (WARM) models are used to improve the accuracy of the failure rate estimation process. The software error log analysis system is implemented using the Association Rule Mining (ARM) and Weighted Association Rule Mining (WARM) techniques. The WARM model produces better detection accuracy level than the ARM model. The system can be enhanced with the following features.

- The system can be enhanced to handle software failures automatically by controlling the software termination operations.
- The offline error log analysis mechanism can be adapted to online error analysis model to manage web site errors.
- The system can be enhanced with clustering techniques to fetch the error patterns.
- The system can be improved to identify the failure risk level based on the log and failure relationships.

## REFERENCES

[1] H. Barringer, A. Groce, K. Havelund and M. Smith, "Formal Analysis of Log Files," J. Aerospace Computing, Information and Comm., pp. 365-390, 2010.
[2] M. Cinque, R. Natella, and A. Pecchia, "Assessing and Improving the Effectiveness of Logs for the Analysis of Software Faults," Proc. Int'l Conf. Dependable Systems and Networks, pp. 457-466, 2010.
[3] Venera Arnaoudova, Laleh M. Eshkevari, Rocco Oliveto and Giuliano Antoniol, "REPENT: Analyzing the Nature of Identifier Renamings", IEEE Transactions On Software Engineering, Vol. 40, No. 5, May 2014
[4] Cemal Yilmaz, Myra B. Cohen and Adam Porter, "Reducing Masking Effects in Combinatorial Interaction Testing: A Feedback Driven Adaptive Approach", IEEE Transactions On Software Engineering, Vol. 40, No. 1, January 2014.
[5] Marcello Cinque, Domenico Cotroneo and Antonio Pecchia, "Event Logs for the Analysis of Software Failures: A Rule-Based Approach", IEEE Transactions On Software Engineering, June 2013.
[6] IBM, "Common Event Infrastructure," http://www-01.ibm.com/software/tivoli/features/cei, 2012.
[7] D. Yuan, J. Zheng, S. Park, Y. Zhou and S. Savage, "Improving Software Diagnosability via Log Enhancement," Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems, pp. 3-14, 2011.
[8] A. Rabkin, W. Xu and R. Katz, "A Graphical Representation for Identifier Structure in Logs," Proc. Workshop Managing Systems via Log Analysis and Machine Learning Techniques, 2010.
[9] Apache log4j, http://logging.apache.org/log4j/, 2012.
[10] A. Pecchia, Z. Kalbarczyk and R.K. Iyer, "Improving Log-Based Field Failure Data Analysis of Multi-Node Computing Systems," Proc. Int'l Conf. Dependable Systems and Networks, pp. 97-108, 2011.

 