*Regular Article*

# Two-Phase Defect Detection Using Clustering and Classification Methods

**Ha Manh Tran[1], Tuan Anh Nguyen[1], Son Thanh Le[2], Giang Vu Truong Huynh[3], Tuan Bao Lam[3]**

[1] Department of Information Technology, University of Foreign Languages - Information Technology, Ho Chi Minh City, Vietnam
[2] School of Computer Science and Engineering, International University - Vietnam National University, Ho Chi Minh City, Vietnam
[3] Center of Information Management, Vietnam Posts and Telecommunications Group, Tay Ninh, Vietnam

Correspondence: Ha Manh Tran, hatm@huflit.edu.vn
Communication: received 25 August 2021, revised 19 September 2021, accepted 20 September 2021
Online publication: 23 October 2021, Digital Object Identifier: 10.21553/rev-jec.296
The associate editor coordinating the review of this article and recommending it for publication was Prof. Vo Nguyen Quoc Bao.

*Abstract*– **Autonomous fault management of network and distributed systems is a challenging research problem and attracts many research activities. Solving this problem heavily depends on expertise knowledge and supporting tools for monitoring and detecting defects automatically. Recent research activities have focused on machine learning techniques that scrutinize system output data for mining abnormal events and detecting defects. This paper proposes a two-phase defect detection for network and distributed systems using log messages clustering and classification. The approach takes advantage of K-means clustering method to obtain abnormal messages and random forest method to detect the relationship of the abnormal messages and the existing defects. Several experiments have evaluated the performance of this approach using the log message data of Hadoop Distributed File System (HDFS) and the bug report data of Bug Tracking System (BTS). Evaluation results have disclosed some remarks with lessons learned.**

*Keywords*– **Defect Detection, Clustering, Classification, HDFS Logs, BTS Bugs, Network Fault Management.**

## 1 Introduction

Defect detection is one of main functions of fault management on network and distributed systems. Automating this function becomes highly demanding due to the rapid development of these systems today with complexity, dynamicity and scalability. However, detecting defects automatically is challenging because this problem heavily depends on expertise knowledge and supporting tools usually used to organize the workflow of fault management process. To overcome these obstacles, a prevailing approach has applied artificial intelligence methods for exploiting system output data and detecting defects on a such system. LogPAI and Loghub projects [1, 2] provide an open-source artificial intelligence platform and a large collection of system log data for automated log analytics. Many research activities have successfully performed effective machine learning methods on the platform and log data for different purposes including anomaly detection or problem identification. Moreover, a more feasible and effective approach is to exploit multiple datasets instead of a single dataset including log data, bug data and other specific data for improving the performance of defect detection.

This study aims at combining clustering and classification methods for detecting defects on network and distributed systems. We propose a two-phase defect detection approach based on mining log messages and bug reports data. The first phase applies K-means clustering method to divide log messages into three clusters: *normal* cluster contains log messages that cannot link to defects, *abnormal* cluster contains log messages that can link to defects, and *unknown* cluster contains log messages that need further investigation. The second phase applies random forest method on bug reports to build a classifier of optimal decision trees. This classifier determines whether log messages in the abnormal and unknown clusters are defects or not. This two-phase approach is characterized by the workflow similarity of fault management process and the capability of exploiting multiple machine learning methods on log and bug datasets for improving defect detection. The contribution of this study is thus threefold:

- Propose a two-phase defect detecting approach based on machine learning methods for network and distributed systems
- Apply the K-means clustering and random forest methods for clustering and classifying log messages and bug reports to detect defects.
- Provide analysis of appropriate features for the real HDFS log and BTS bug data, and performance evaluation of the proposed approach.

The rest of the paper is structured as follows: The next section presents several recent research activities of mining system log data and applying machine learning

methods for monitoring and detecting defects on network and distributed systems. Section 3 describes the statistical information of the HDFS log data and the BTS bug data, and analyzes appropriate log and bug features for the K-means clustering and random forest methods. Section 4 provides the design of two-phase defect detection, the algorithms of clustering abnormal log messages and detecting defects from these abnormal messages, and some implementation remarks. Several experiments in Section 5 measure the performance of the proposed approach using the HDFS log and BTS bug datasets before the paper is concluded in Section 6.

## 2 Related Work

With the growth of data science, system log mining has recently attracted many research activities for improving system dependability. The authors of the study [3] have proposed the combined approach for mining console logs. The approach takes advantage of program analysis and information retrieval techniques to analyze sequences of events from the system and detect operational problems. The study has presented the resulting analysis of the critical log messages associated with the detected problems. The study of Alspaugh et al. [4] has reported some experience of log analysis, such as filtering, reformatting, and summarizing log data. The study also presents state machine based description of typical log analysis pipelines and cluster analysis of the most common transformation types to assist in making decisions. The authors of the study [5] have provided an experience report of using log analysis for anomaly detection. The study reviews and evaluates six machine learning methods on a large dataset of 16 millions log messages and 365 thousands anomaly instances. The authors of the study [6] have proposed a clustering-based approach to identify impactful system problems. The approach applies a cascading clustering algorithm on a sequence of log events for correlating the clusters of log sequences with system services. The LogPAI project [1] aims to build an open-source artificial intelligence platform for automated log analysis. The project provides a large collection of system log datasets for automated log analytics, namely Loghub [2]. Loghub includes several log datasets obtained from real distributed systems, supercomputers or servers and attracts many research activities from both industry and academia. Our study also uses the HDFS log data (part 2) in Loghub for experiments.

Several studies of defect monitoring, diagnosis and detection in network and distributed systems have exploited machine learning methods. The study of Uddin et al. [7] has proposed an approach of using a distributed network search algorithm as a primitive for building future network management systems. This algorithm operates on network search systems organized as the overlay of the physical network topology. The overlay provides distributed query processing facilities for retrieving operational state and configuration data from network elements. The study of Tran et al. [8] has proposed an approach for searching bug reports semantically. This approach includes crawlers that obtain bug reports from bug tracking systems, and then extracts semi-structured bug data, and describes a unified data model to store bug data. The approach also exploits package dependency, fault dependency, fault keyword and classification to seek the relationships between defect causes using machine learning methods. The study of Wang et al. [9] has addressed the problem of automatic defect diagnosis and presented an online incremental clustering method for cloud computing applications. This method learns access behavior patterns and analyzes the correlation between workload and resource utilization metric. The method detects anomalies by identifying the abrupt change of correlation coefficients, and locates suspicious metrics using the feature selection method. The study of Zhou et al. [10] has applied the traditional K-nearest neighbor algorithm to recommend appropriate solutions for defects. This algorithm uses the latent Dirichlet allocation method to improve the effectiveness of similarity measure between the events and previous resolutions of similar events. The study of Ferreira et al. [11] has proposed an approach of using machine learning methods for automating defect detection on solar-powered wireless mesh networks. This approach applies knowledge discovery methodology and a pre-defined dictionary of defects and solutions for classifying new defects.

## 3 Log and Bug Data

### 3.1 HDFS Log Data (part 2)

The Loghub study [2] maintains a collection of system logs for free access. The HDFS log data (part 2) contains log files obtained from the HDFS system of 33 nodes at a university. The log data is collected at the node level without modification or labeling and may involve both normal and abnormal cases due to the repair of three nodes. Table I reports some statistics of this log data:

Table I
Some Statistics of the Log Data

| | |
|---|---|
| number of log files | 33 |
| size of log files (GB) | 16.05 |
| number of log messages | 58095613 |
| number of INFO messages | 57570609 |
| number of WARN messages | 500971 |
| number of ERROR messages | 24030 |
| number of FATAL messages | 3 |

A log message contains several features: *date* and *time* in format of yy/MM/dd HH:mm:ss; *severity* level to affect the system; *component* and *class* names to issue the message; *content* to present the detail of the message. The severity level accepts the following values:

- FATAL: The error is shown on the status console and probably stops the application or system.

- ERROR: The runtime error or unexpected condition is shown on the status console and probably keeps the application or system running.
- WARN: The undesirable or unexpected condition is shown on the status console and potentially presents dangerous conditions
- INFO: The interesting message in application or system progress is shown on the status console.
- DEBUG: The detailed information of an event to debug the application or system is written to logs only.
- TRACE: The more detailed information than DEBUG to help debug the application or system is written to logs only.

An example of the WARN log messages is presented, as follows:

```
2017-01-26 20:01:44 WARN org.apache.
hadoop.hdfs.server.datanode.DataNode:
Slow BlockReceiver write data to disk
cost:892ms (threshold=300ms)
2017-01-26 20:01:47 WARN org.apache.
hadoop.hdfs.server.datanode.DataNode:
Slow manageWriterOsCache took 822ms
(threshold=300ms)
2017-01-26 20:01:50 WARN org.apache.
hadoop.hdfs.server.datanode.DataNode:
Slow BlockReceiver write data to disk
cost:1653ms (threshold=300ms)
```

There are many repeated log messages with various occurrence frequencies. The date and time features are merged and a repetition feature is added to reduce the repeated message. The repetition feature can accept values: none, intermittent and high. The severity level focuses on three values: FATAL, ERROR and WARN. The component and class names are separated by two features, e.g., `org.apache.hadoop.ipc.Server` includes `org.apache.hadoop.ipc` as component name and `Server` as class name. The keyword feature contains significant words or word phases from the content message. While processing and evaluating words by the term frequency–inverse document frequency method, the category feature is determined by the keyword feature, e.g., Memory, Disk, Cache, IO, Process, etc. Table II presents a list of extracted log features used to cluster log messages.

### 3.2 BTS Bug Data

The EMANICS project [12] provides a flexible and highly re-usable distributed computing and storage testbed to support joint research activities of European partners. The study [8] uses the testbed to obtain bug reports from multiple BTSs including Buzilla [13], Trac [14], Mantis [15], Launchpad [16], etc., and store them in a database with a unified bug data schema. Table III reports some statistics of this bug data.

A bug report contains several features divided into two groups: numeric and enumerate features, such as, identity, priority, severity, status, platform, component, etc., and textual features, such as, title, description, discussion, attachment, etc. Most of bug features can be extracted from bug reports. However, BTSs provide different values for some features. The schema thus specifies the values of the severity feature as critical, normal, minor and feature, and the values of the priority feature as urgent, high, normal and low. Table IV presents a list of extracted bug features used to classify bug reports. These features play an important role in determining the relation of log messages to the existing bug reports with certain severity and priority levels. The component, software and platform features provide the scope of bugs and possibly reveal related or similar bugs. The category feature specifies which areas bugs occur, e.g., hardware, software, service, network, security, etc. The relation feature includes previously related bugs that are unavailable for some BTSs. The keyword feature contains the resulting set of distinct keywords after processing words in the textual features.

Several features in bug reports are infeasible for training and evaluating classifiers. The data processing task is therefore important to transform raw data to feasible data for improving the performance of classifiers. This task includes dropping unnecessary data items, filling blank data items, re-formatting data features from various data types to enumerate data type. Firstly,

Table II
List of Extracted Log Features

| Feature | Description | Type |
|---|---|---|
| datetime | issuing date and time | DateTime, original |
| severity | influence level | Enumerate, original |
| component | occurring component | Enumerate, derived |
| class | occurring class | Enumerate, derived |
| keyword | distinct word phases | String, derived |
| category | specifying log category | Enumerate, derived |
| repetition | repeated log message | Enumerate, derived |

Table III
Some Statistics of the Bug Data

| | |
|---|---|
| number of bug files | 5 |
| size of bug files (GB) | 0.77 |
| number of bug reports | 483000 |
| number of gentoo bug reports | 150000 |
| number of redhat bug reports | 83000 |
| number of mozilla bug reports | 250000 |

Table IV
List of Extracted Bug Features

| Feature | Description | Data Type |
|---|---|---|
| severity | influence level | Enumerate, original |
| priority | fixing order | Enumerate, original |
| status | bug state | Enumerate, original |
| component | occurring component | Enumerate, original |
| software | occurring software | Enumerate, original |
| platform | occurring platform | Enumerate, original |
| keyword | distinct word phases | String, derived |
| relation | relating bugs | Identifier, derived |
| category | specifying bug category | Enumerate, derived |
| parameter | specifying system parameters | Enumerate, derived |

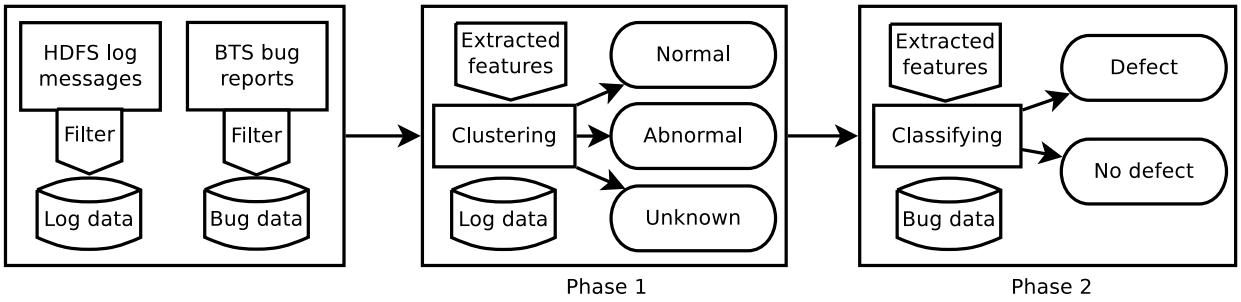


Figure 1. A design of two-phase defect detection.

**Algorithm 1:** Constructing clusters for log messages

**Input**: Dataset $X$ and number of clusters $K$
**Output**: List of centroids $M$ and their data points $Y$
1 Choose $K$ points randomly as initial centroids
2 Assign each data point to a cluster with its closest centroid
3 Stop the algorithm if the assignment no longer changes
4 Compute the average values of all data points in clusters
5 Update centroids for clusters $K$
6 Repeat step 2
**Return** $M$ and $Y$

the dataset is loaded into panda data frame which allows data items to be manipulated with ease. Then the dataset is extracted by necessary features listed in Table IV. All the features are required to be enumerate. The content feature usually contains long strings for describing bugs in details. We use the term frequency–inverse document frequency (tf×idf) method to obtain significant keywords for the keyword feature. This method evaluates the significance of keywords by the occurrence frequency of the keywords in both bug reports and the dataset. A distinct group of keywords contains related keywords with high significance. As a consequence, the keyword feature includes a set of keywords and groups that best describe bug reports. However, since bug reports are obtained from various BTSs, their descriptions and discussions contain redundant words, nonsense words or even meaningless words, such as: memory address, debug information, system path, article, etc. We filter out these words from the dataset.

## 4 Two-phase Defect Detection

Most previous research activities have applied machine learning methods for exploiting single datasets including either log data, bug data or other specific data. This approach focuses more on the two-phase defect detection that follows the workflow of fault management process:

- Phase 1: filtering and partitioning log messages into normal, abnormal or unknown clusters
- Phase 2: diagnosing and classifying the abnormal messages into defect or no defect classes.

Figure 1 depicts a design of two-phase defect detection. The input data includes various log messages, e.g., HDFS log messages in this study are console log messages with INFO, WARN, ERROR and FATAL severity levels. Since INFO messages provide less dangerous information than other messages, this approach filters INFO message out, eliminates the repeated messages and processes the remaining messages for applying K-means clustering and random forest methods. The input and output of phase 1 is the processed log messages and the abnormal log messages, respectively. The input and output of phase 2 is the abnormal log messages and the detected defects, respectively.

### 4.1 K-Means Clustering

The K-means clustering method [17] aims to partition data points into clusters such that data points in the same cluster share the same features. This unsupervised learning method possesses no knowledge of the label of data points. Assume the dataset $X = [x_1, ..., x_N]$ of $N$ log messages; each log message presents a vector $x_i = [x_{i1}, ..., x_{id}]$, where $d$ denotes number of the extracted features of a log message; $K < N$ denotes number of clusters. This method seeks centroids $M = [m_1, ..., m_K]$ and their log messages, e.g., normal, abnormal or unknown labels. Algorithm 1 presents steps to construct clusters for log messages.

The algorithm starts with $K$ centroids, where each centroid is a vector of $d$ elements with initially random values (1). Using the Euclidean distance, each log message of $d$ features as a vector is assigned to a cluster by the closest distance with the cluster's centroid (2). The algorithm stops if the assignment of log messages to clusters no longer changes (3). Otherwise, the algorithm continues to update new centroids for clusters by computing the average values of all log messages in

**Algorithm 2:** Constructing random forest for bug reports

**Input**: Dataset $Y$ and a threshold $\sigma$
**Output**: Random forest of optimal trees $R$
1 Divide $Y$ randomly into testing $E$ and training $T$ subsets
2 Use $T$ to build a tree
3 Find the best split for each feature
4 Find the node for the best split
5 Stop if the node generation no longer applies
6 Split the node using its best split
7 Repeat step 3
8 Evaluate the tree with $E$ and $\sigma$, then update $R$
9 Stop if the tree generation no longer applies
10 Repeat step 1
**Return** $R$

**Algorithm 3:** Selecting distinct keywords for log and bug data

**Input**: Raw keyword set (title, description, discussion, etc.)
**Output**: Distinct keyword set with weight
1 Load original keyword set
2 Remove frequent or redundant words with stop-word set
3 Reduce inflected words with stemming and lemmatization
4 Remove meaningless words with regular expression
5 Process tf×idf on filtered keyword set
6 Select distinct keywords with high weight
**Return** Distinct keyword set with weight

the clusters (4&5) and then repeats step 2. The result is a list of centroids $M$ and sets of log messages $Y$ for clusters.

### 4.2 Random Forest

The random forest method [18] aims to classify data points into classes using a number of decision trees. This supervised learning method uses the training dataset to construct several optimal decision trees at training time and exploits the decision of these trees to evaluate the realistic dataset. An algorithm of building decision trees top-down from the root node to leaf nodes thus plays an important role in this ensemble tree-based method. Assume the dataset $Y = [y_1, ..., y_M]$ of $M$ bug reports; each bug report presents a vector $y_i = [y_{i1}, ..., y_{ik}]$, where $k$ denotes number of the extracted features of a bug report; $\sigma$ is a threshold for choosing qualified trees. This method returns random forest $R$ of optimal trees. Algorithm 2 presents steps to construct random forest for bug reports.

The algorithm starts with dividing randomly the dataset $Y$ into the testing $E$ and training $T$ subsets by a ratio 25% and 75% (1). The training subset $T$ is used to build a tree (2). The process to grow a tree contains three steps. The first step is to find the best split for each feature (3). This step uses entropy splitting rules to choose the best split among all the possible splits that consist of possible splits of each feature, resulting in two subsets of features. Each split depends on the value of only one feature and the best split maximizes the defined splitting criterion. The second step is to find the best split of the node among the best splits found in the first step (4). The best split also maximizes the defined splitting criterion. The third step is to split the node using its best split found in the second step (6). This process repeats the first step if the stopping rules are not satisfied. After growing the tree, the algorithm uses the testing subset $E$ to evaluate the tree and compare the evaluation result with $\sigma$. The qualified tree is then added to random forest $R$ (8). The algorithm continues to divide $Y$ and grow several trees until having sufficient number of qualified decision trees (9).

### 4.3 Implementation

We have used *python* and several libraries *sklearn*, *pandas*, *numpy*, etc. to filter and process log and bug data and implement the K-means clustering and random forest methods. It is essential to extract features for log messages and bug reports because the used methods only accept ordinal categorical, nominal categorical, or continuous features. However, both log messages and bug reports contain textual features such as title, description, or discussion, etc. that hold important information for mining.

We have applied text processing methods for the textual features. Algorithm 3 selects distinct keywords with weight from log messages and bug reports. The

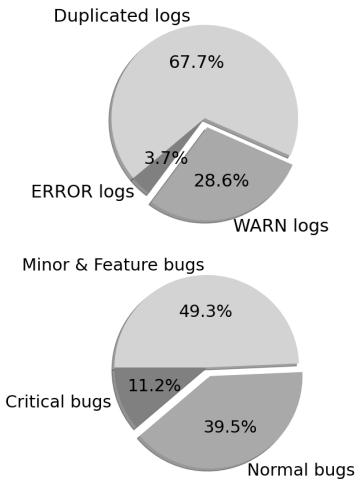


Figure 2. Statistics of filtered log messages and bug reports.

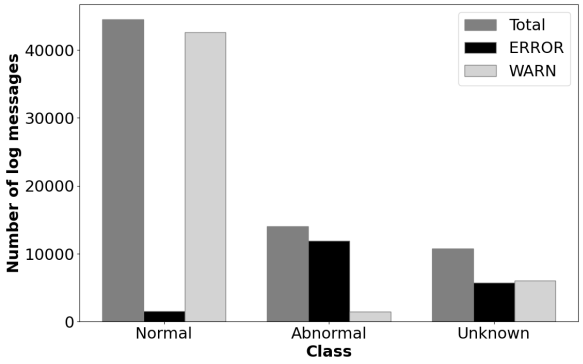


Figure 3. Numbers of log messages in the resulting clusters.

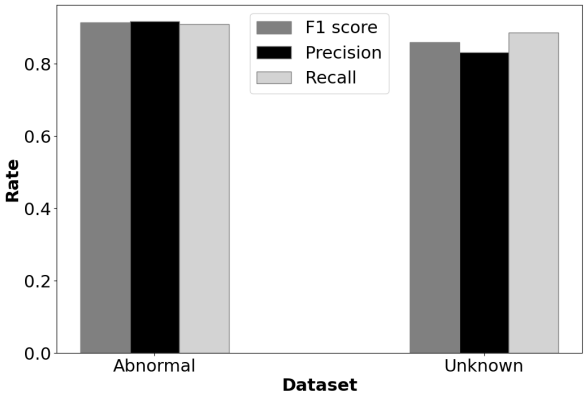


Figure 4. Performance of the random forest classifier on the abnormal and unknown datasets.

algorithm starts with loading original keyword set (1), applies several steps to remove trivial words and correct inflected words (2, 3, 4), and produces filtered keyword set. This algorithm then uses the term frequency–inverse document frequency (tf×idf) method to weigh filtered keyword set (5) and return distinct keywords with high weight (6).

## 5 Evaluation

We have used the HDFS log and BTS bug datasets to evaluate the proposed approach. The log dataset contains a large number of INFO log messages and duplicated messages. After filtering these trivial messages from total 520 thousands messages, the remaining dataset possesses 28.6% WARN messages and 3.7% ERROR messages including 3 FATAL messages. WARN and ERROR log messages are used to form clusters. Similarly, the bug dataset contains a large number of bug reports with minor and feature severity levels. After filtering these trivial reports from 480 thousands reports, the remaining dataset possesses 11.2% critical reports and 39.5% normal reports. Critical and normal bug reports are used to build random forest. Figure 2 displays filtered log messages and bug reports statistics. The log dataset reports various problems: 75% of IO exceptions, 20% of general exceptions, 5% of other problems: creating new image, failing to write to disk, network connectivity, managing operating system cache, output error, failing to transfer data, unknown operation, invalid directory, failing to delete file, etc.

The first phase applies the K-means clustering method to partition the filtered WARN and ERROR log messages into normal, abnormal and unknown clusters. Figure 3 reports numbers of log messages in the resulting clusters. There are 44, 14 and 10 thousands totally for normal, abnormal and unknown clusters, respectively. A large number of 85% WARN messages are referred to as normal, while a small number of 15% WARN messages are referred to as abnormal or unknown. Conversely, a large number of 62% ERROR messages are referred to as abnormal or unknown, while a small number of 38% ERROR messages are referred to as normal or unknown. In addition, the unknown cluster contains 12% of WARN messages and 30% of ERROR messages. These messages cannot be determined by the method.

The second phase applies the random forest method on the filtered bug reports and bug features to construct a random forest classifier with several optimal decision trees. The classifier is then used to classify the abnormal and unknown datasets into defect or no defect classes. Three metrics of Precision, Recall, F1-score are used for evaluating the performance of the random forest method, as defined below:

$$\text{Precision} = \frac{\text{Number of true defects detected}}{\text{Number of true and false defects detected}},$$

$$\text{Recall} = \frac{\text{Number of true defects detected}}{\text{Number of all defects detected}},$$

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision + Recall}}.$$

ERROR messages and a small number of WARN messages related to system failure are determined as true defects to computing the metrics. Figure 4 reports the performance of the random forest classifier on the abnormal and unknown datasets.

The precision and recall rates of the abnormal dataset both reach 0.91 as the classifier detects almost ERROR log messages correctly, and also confuses several

WARN log messages. It is interesting to observe that the classifier detects some WARN log messages relating to existing defects correctly. However, the precision and recall rates of the unknown dataset are lower than that of the abnormal dataset, i.e., 0.83 and 0.87 respectively. The first reason is the unknown dataset contains a large number of WARN log messages and many of them are not defects. The second reason is several ERROR log messages in this dataset contain too general information that confuses the classifier.

## 6 Conclusion

We have proposed the two-phase defect detection approach for network and distributed systems. This approach applies machine learning methods to clustering and classifying log messages obtained from the systems in order to seek defects. We have presented the design of two-phase defect detection, where the first phase uses the K-means clustering method to partition log messages into normal, abnormal and unknown clusters; the second phase uses the random forest method and bug reports to train a random forest classifier and determine the abnormal and unknown log messages whether they are defects or no defects. We have used realistic HDFS log message and BTS bug report datasets to evaluate the performance of the proposed approach. The experimental results have specified some remarks. The HDFS dataset contains a large number of INFO messages and duplicated messages, and the BTS dataset contains a large number of bug reports with feature and minor severity levels. These messages and reports must be filtered in advance and several log and bug features must be supplemented into the lists of extracted features. The K-means clustering method partitions 88% of WARN messages and 70% of ERROR messages into the normal and abnormal clusters, respectively, and 12% of WARN messages and 30% of ERROR messages into the unknown cluster. The random forest method reaches the F1 score rates of 91% and 86% for the abnormal and unknown datasets, respectively. Future work focuses on exploring further log and bug datasets and exploiting further log and bug features for improving the performance of the proposed approach.

## Acknowledgment

This research activity is funded by Ho Chi Minh City University of Foreign Languages–Information Technology under the grant number H2021-05.

## References

[1] M. R. Lyu, J. Zhu, P. He, S. He, and J. Liu, "The logpai project," http://www.logpai.com/, 2019.

[2] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," *ArXiv*, vol. abs/2008.06448, 2020.

[3] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 117–132.

[4] S. Alspaugh, B. Chen, J. Lin, A. Ganapathi, M. A. Hearst, and R. Katz, "Analyzing log analysis: An empirical study of user log mining," in *Proceedings of the 28th USENIX Conference on Large Installation System Administration*, ser. LISA'14. USA: USENIX Association, 2014, pp. 53–68.

[5] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *Proceedings of the 27th IEEE International Symposium on Software Reliability Engineering (ISSRE'16)*. IEEE, 2016, pp. 207–218.

[6] S. He, Q. Lin, J. G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE'18. New York, NY, USA: ACM, 2018, pp. 60–70.

[7] M. Uddin, R. Stadler, and A. Clemm, "A Query Language for Network Search," in *Proceedings of the 13th IFIP/IEEE International Symposium on Integrated Network Management (IM'13)*. IEEE, 2013, pp. 109–117.

[8] H. M. Tran and S. T. Le, "Software Bug Ontology Supporting Semantic Bug Search on Peer-to-Peer Networks," *New Generation Computing*, vol. 32, no. 2, pp. 145–162, 2014.

[9] T. Wang, W. Zhang, J. Wei, and H. Zhong, "Fault Detection for Cloud Computing Systems with Correlation Analysis," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM'15)*. IEEE, 2015, pp. 652–658.

[10] W. Zhou, L. Tang, C. Zeng, T. Li, L. Shwartz, and G. Y. Grabarnik, "Resolution recommendation for event tickets in service management," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 954–967, 2016.

[11] V. C. Ferreira, R. C. Carrano, J. O. Silva, C. V. N. Albuquerque, D. C. Muchaluat-Saade, and D. G. Passos, "Fault Detection and Diagnosis for Solar-Powered Wireless Mesh Networks Using Machine Learning," in *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM'17)*. IEEE, 2017, pp. 456–462.

[12] D. Hausheer and C. Morariu, *Distributed Test-Lab: EMANICSLab*, University of Zurich, Switzerland, June 2008, The 2nd International Summer School on Network and Service Management (ISSNSM'08).

[13] M. Foundation, "Bugzilla bug tracker," https://www. bugzilla. org/, 1998.

[14] E. Software, "Trac bug tracker," https:// trac. edgewall. org/, 2004.

[15] M. Team, "Mantis bug tracker," https:// www. mantisbt. org/, 2000.

[16] C. Company, "Launchpad bug tracker," https:// bugs. launchpad. net/, 2004.

[17] J. A. Hartigan, *Clustering Algorithms*, 99th ed. USA: John Wiley & Sons, Inc., 1975.

[18] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

**Ha Manh Tran** is associate professor of computer science at HCMC University of Foreign Languages–Information Technology. He obtained his master degree in computer science in 2004 from the University of Birmingham, United Kingdom and his doctoral degree in computer science in 2009 from Jacobs University Bremen, Germany. His research interests include communication networks, distributed systems, network management, information retrieval and machine learning.

**Tuan Anh Nguyen** is senior lecturer of computer science at HCMC University of Foreign Languages–Information Technology. He obtained his master degree in computer science in 2004 from University of Natural Sciences–HCMC Vietnam National University and his doctoral degree in computer science in 2012 from La Trobe University, Australia. His research interests include mobile computing, context-aware computing and information security.

**Son Thanh Le** received his master degree in computer science in 2006 from Korea Advanced Institute of Science and Technology (KAIST), Korea. He is working as senior lecturer at School of Computer Science and Engineering, International University–HCMC Vietnam National University. His current research interests focus on data science, web technology, information retrieval, mobile computing and computer networks.

**Giang Vu Truong Huynh** is master student in computer science at Posts and Telecommunications Institute of Technology. He received his bachelor degree in electrical and electronic engineering in 2018 from University of Technology–HCMC Vietnam National University. His research interests include network management, autonomous fault management, communication networks and distributed systems.

**Tuan Bao Lam** is master student in computer science at Posts and Telecommunications Institute of Technology. He received his bachelor degree in computer science in 2017 from Posts and Telecommunications Institute of Technology. His research interests include network management, autonomous fault management, communication networks and distributed systems.