An Approach to Pinpointing Bug-induced Failure in Logs for Open Cloud Platforms

Jia Tong, LI Ying*, Tang Hongyan, Wu Zhonghai
School of Software and Microelectronics, Peking University, Beijing, China
*National Engineering Center of Software Engineering, Peking University, Beijing, China

***Abstract*— Software bugs have been one of the dominant causes of system failures, especially in cloud systems based on open source platforms. One big challenge for troubleshooting these cloud systems is to pinpoint the software bug-induced failure in large and complex log files which is a nightmare for administrators. So far, there has been little study on how to identity bug-induced failures based on log analysis. In this paper, we analyze and describe features of bug-induced failure logs from bug repository and Q&A websites, and then propose a general automatic approach to pinpoint logs of bug-induced failure from log files of open cloud platform. In the approach, two algorithms called MPIN and SPIN are presented for log classification. We evaluate our approach by applying logs collected from bug repositories of OpenStack and Hadoop, and five Q&A websites. The experimental result shows that the proposed approach can identify logs of bug-induced failure in OpenStack logs with 83.9% precision, and for Hadoop logs with 82.52% precision.**

Keywords—open cloud platforms, log analysis, bug-induced failures, machine learning

I. INTRODUCTION

Recently, software bugs have been one of the dominant causes of cloud system failures, resulting in many severe cloud service outages and downtime and affecting millions of their customers[4][5]. For instance, an undetected bug brought down several Google services for about one hour on Jan. 24, 2014. Bug-induced failures tend to recur in many cloud systems because they are based on some popular open platforms like Hadoop and OpenStack. The recent analysis effort on open cloud platforms indicates that software bugs significantly undermine system dependability, and bugs can lead to almost all kinds of implications such as failed operations, component downtimes, data loss, corruption and so on[1]. To make things worse, bugs are the major root causes (accounting for 81.1-86.7%) of failures in modern open source software and it takes a long time to find and fix these bugs [7]. In practice, system logs are the main information source to perform troubleshooting of cloud system, but unfortunately it is notoriously difficult to identify bug-induced failure from system logs due to its ever increasing scale and complexity as well as the nature of software bugs that a software bug is inherent and hidden in code.

Manually examining millions of logs to find the cause of system failures is cumbersome. The traditional practical method is to search key words such as “error”, “warning”, “info” and so on, which obviously cannot identify bug-induced failure from log files automatically. Previous work on log analysis ([8],[9],[10],[11],[12]) have presented multiple methods for system anomaly detection which focus on diagnosing failure patterns and classify running logs into *normal* and *abnormal*. However, even with these automatic failure detection methods, it is still challenging to understand a large number of logs to find failure causes, not to mention identifying bugs-induced failures.

In this paper, we aim to identify logs of bug-induced failures from log files automatically via mining logs from the open bug repository and Q&A websites of open source projects. We observed that bug reports and Q&A discussions usually contain a certain number of logs that users hope to provide more contexts with for bugs or problems. While logs from bug repository represent the system status when bug-induced failures occur and logs from Q&A websites indicate other system failures such as misconfiguration, compatibility problems etc. These logs are closely correlated with system anomalies and depict the real system failures accurately, which indicates that bug repository and Q&A websites are reliable sources for understanding bugs and logs. Furthermore, after collecting logs from bug repository of OpenStack[2] and Hadoop [24], and their Q&A web sites [21][22][23][26][27][28][29] and considering logs as *bag-of-words*, we make an analysis on both the content and similarity among these logs. Result reveals that a vast majority of logs from bug repository is dissimilar to those from Q&A websites. It satisfies our assumption that the log of bug-induced failure can be distinguished from others via classifying logs from bug repository and Q&A websites. These two types of logs from different sources are called *bug logs* and *Q&A logs* separately in the rest of this paper.

Then a pinpointing approach is proposed with three steps: log retrieval, log parsing, and log classification, wherein two log classification algorithms called MPIN and SPIN based on machine learning and similarity matching are implemented. MPIN first groups logs based on their text features, then replace each log with its group id to generate feature vectors for classification. SPIN holds a log repository, and finds the most similar log from the repository to represent the feature of user logs. Our approach can help system administrators to narrow down

2159-6190/16 $31.00 © 2016 IEEE
DOI 10.1109/CLOUD.2016.45

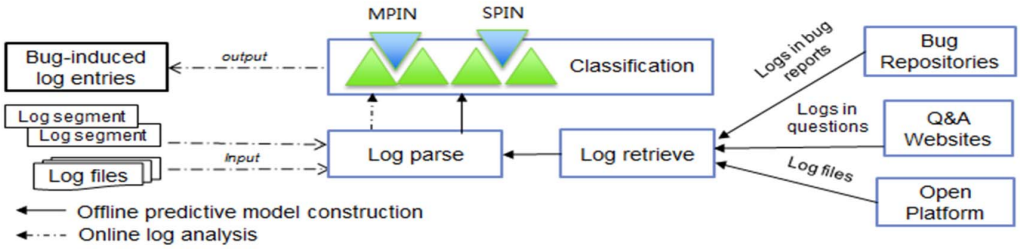


Figure 1: Framework of the approach for pinpointing bug-induced failure in logs

the examination scope by automatically extracting key log entries (bug related) from thousands of log entries.

To evaluate our approach, we conduct an extensive study with experiments. The collected logs from bug repositories and Q&A websites are mixed with real logs generated by OpenStack and Hadoop instances in our lab, as input to the pinpointing approach for examination. Experimental results show that the proposed approach can automatically identify log entry of bug-induced failure in OpenStack logs with 85.78% accuracy and 83.9% precision. Meanwhile, it can also identify bug-induced failures in Hadoop logs with 82.52% precision.

The rest of the paper is organized as follows. Section 2 discusses some related work. Section 3 describes the preliminary analysis of logs collected from bug repository and Q&A sites. Section 4 gives an overview of our approach. Section 5 presents two classification algorithms MPIN and SPIN. Section 6 elaborates the evaluation of our approach on OpenStack and Hadoop. At last, section 7 concludes this paper.

II. RELATED WORK

A. Anomaly detection based on log mining approach

Log mining is a popular technique for anomaly detection and intrusion detection. For instance, Wei Xu et al. [8][9] propose a log mining approach for detecting large-scale system problems. Their basic idea is to scan source code from software components to extract the template of voluminous intermixing logs. With these log "schema", they are able to eliminate most heuristics and guesses for log parsing used by existing solutions. Based on this, they label log entries as *normal* and *abnormal* for anomaly detection.

Antt Juvonen et al. [10] propose a log anomaly detection framework which facilitates quick anomaly detection and also provides visualizations of network traffic structure. Their method processes logs into a numerical data matrix and calculate an anomaly score for each data point. The data matrix is built on the frequencies of individual characters in log entries. Adam J. Oliner et al. [11] present Nodeinfo, an unsupervised algorithm for anomaly detection in system logs. Their work is to identify the regions of the log containing anomalous events, which is a little similar with our work. However, our goal is to identify the regions of logs containing messages of bug-induced failure.

Table 1: Collected bugs and logs for Hadoop

Component	Bug number	Log number
Hadoop Common	6800	5361
HDFS	4047	4581
MapReduce	3752	3021
Yarn	1847	1930
total	16446	14893

#Hadoop Common: The common utilities that support the other Hadoop modules.
#HDFS: A distributed file system that provides high-throughput access to application data.
#MapReduce: A YARN-based system for parallel processing of large data sets.
#YARN: A framework for job scheduling and cluster resource management.

Table 2: All collected logs for bugs and questions

Platform	Bug number	BLog number	Question number	QLog number
OpenStack	12268	3976	5412	2850
Hadoop	16446	14893	25059	54887

#Bug number: amount of bug reports extracted from bug repository.
#BLog number: amount of bug logs extracted from bug reports.
#Question number: amount of Q&A pages extracted from Q&A websites .
#QLog number: amount of Q&A logs extracted from Q&A websites.

Chinghway Lim et al.[11] apply log mining techniques to characterize the behavior of large enterprise telephony systems. They aim to detect and predict system anomalies using individual message frequencies to characterize system behavior and the ability to incorporate domain-specific knowledge through user feedback.

B. Bug analysis

Bugs are the main cause of software failures. Bug analysis focus on optimizing bug shooting and fixing process to help developers and operators manage software project. Akinori Ihara et al. [13] propose an analysis method which represents a bug modification process to help identify the bottleneck of a bug fixing process. Zhenmin Li et al. [7] present an empirical bug study in open source software. They use natural language text classification and analyze around 29,000 bugs from the Bugzilla databases in order to investigate the impacts of new factors (such as security, concurrency etc.) and open source related characteristics. Kypros Constantinides et al.[14] propose a flexible, low-overhead mechanism to

detect the occurrence of design bugs during online operations. Bug detection approaches usually focus on code and software design to see if the system contains bugs. Our approach also takes bug report as a valuable data source, however, logs buried in bug reports are our specific concerns.

To the best of our knowledge, our approach is the first to utilize bug reports for identifying bug-induced failures from system log files for open cloud platforms. Different from the related work, we combine log mining with bug analysis, and focus on log entries buried in bug reports. By utilizing these log entries from real buggy systems, we are able to pinpointing bug-induced failures beyond anomaly detection.

III. LOG COLLECTION AND ANALYSIS

Our approach is based on two major sources of data from the Internet: bug repository and Q&A websites. We collect these log data through website APIs, crawling techniques and log collection tools. Table 1 and table 2 show the whole dataset from log collection. After that, we make a statistical analysis on these data and observations are presented in this section.

A. log collection

OpenStack bug report consists of many items: *ID, bug title, description, affected component, status, importance, etc*. Some people choose to attach their logs to elaborate a certain bug in the *description* part of the bug report. Similar with reporting bugs, users often attach some runtime logs to describe their problems in detail when asking questions on social Q&A websites. We collect logs from two Q&A websites related to OpenStack: StackOverflow.com [22] and Ask.OpenStack[21].

Hadoop[24] tracks both bugs and enhancement requests called issues by using JIRA[25] from which we collect issues labeled bug in four years. Different from OpenStack, Hadoop consists of 4 major components (Table 1), and each bug report is assigned to a specific component.

Meanwhile, we extract question posts labeled Hadoop from five most popular Q&A websites in StackExchange community[23]: StackOverFlow[22], AskUbuntu[26], Programmers[27], ServerFault[28] and SuperUser[29]. Besides, we crawl 20,556 user mailings from the mailing list maintained in Hadoop community to enrich this dataset. Since user mailing list is the preferred communication for end-user questions and discussions, and it contains lots of system logs in mailing sessions that indicates usage problems or configuration errors.

B. Log dissimilarity

Intuitively, bug logs represent the system status when bug-induced failures occur while Q&A logs indicate other system failures such as misconfiguration error, application level error, etc. We design some analysis on the content of these logs to see if logs in bug reports are similar with those in Q&A websites.

Table 3: Log examples from OpenStack bug repository and Q&A website

Bug id	Log entry	Source
1423015	FATA[0032] could not find image: no such id: registry.access. redhat.com/kollaglue/fedora-rdo-docker-compose	Docker
1333814	2014-06-24 15:32:25 ERR (/Stage[main]/Ceph::Mon/Exec[ceph-deploy mon create]/returns) change from notrun to 0 failed: ceph-deploy mon create node-1:192.168.0.3 returned 1 instead of one of [0] (useless ruby stack trace removed)	Puppet
1506465	2015-10-15 11:25:48.118 ERROR nova.api.openstack.extensions [req-ca854e8c-68fe-47f6-9ce0-c2a5083b417d demo demo] Unexpected exception in API method	Nova-api

Q&A id	Log entry	Source
265799	Error running RPC method deploy: Failed to deploy plugin cinder_netapp-1.0.0	NetApp Plugin
265096	ERROR: The template version is invalid: Unknown version (heat_template_version: 2014-10-16).	Heat
144676	2011-02-08 17:51:11,657 ERROR nova.compute.manager [QJHBM06S7ATHYVS1NQQI anne IRT] instance 1: Failed to spawn	Nova-compute

First, we remove the variable parts of bug logs and Q&A logs, and make an analysis on the frequency of the words in these logs so as to check what information these logs indicate. After eliminating several meaningless but frequent words such as *java, org, at, in, to* and so on, words are sorted in descending order based on their frequency. Due to space limitations, detailed results are not listed in this paper. Taking Hadoop logs as an example, we observe that Q&A logs are usually related with application level errors such as error output of users' buggy code, errors while using management tools, etc., because words like *job, pig, grunt, hive, oozie, etc.* rank ahead. On the contrary, bug logs are usually system level errors with words like *dfs, thread, dfsclient, ipc, etc.* ranking ahead.

Second, we compute the similarity between the constant parts of each Q&A log and each bug log using *string edit distance* [19]. Result shows that only 2.6% of logs in Q&A questions of OpenStack can match those in OpenStack bug repository, meanwhile, 6,737 logs in a total of 54,887 Q&A logs can match to those in Hadoop bug repository resulting in a percentage of 12.27%. Considering that the same Hadoop log usually repeats many times in a question post or bug report, we filter the repeated log entries and then find that only 436 logs are left which is a very small fraction of Hadoop Q&A logs. Therefore, it is reasonable to identify bug-induced failures in log files by mining bug logs and Q&A logs.

C. Challenge of log processing

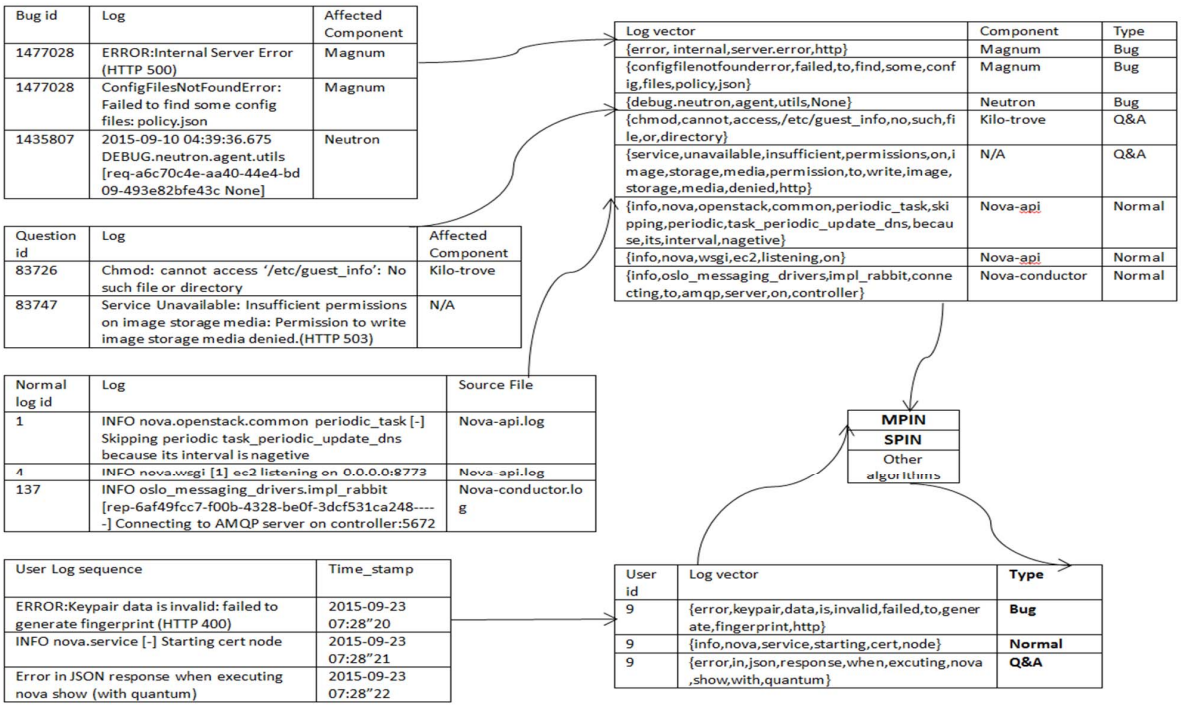


Figure 2: Example of log processing workflow

Logs in websites are sometimes incomplete and unclear because people usually paste a few logs onto the websites and erase a few parts for simplicity and security. For example, bug reporters or questioners are used to pasting incomplete log entries without a few parts such as timestamp, identifier, source file etc., which leads to a heterogeneous and "noisy" condition. Meanwhile, Cloud systems often consist of several software components and projects in each layer. OpenStack is composed of 252 sub-projects maintained in Launchpad most of which can be customized and print logs. Moreover, cloud systems usually integrate other projects, tools or plugins. For instance, log entries from three bug reports and three question descriptions are presented in Table 3. These logs are from different components and layers of OpenStack. Different components produce different structure of logs. Complex software architecture leads to a great complexity when analyzing and utilizing these logs.

IV. OVERVIEW OF APPROACH

Our approach mainly consists of three steps: log retrieval, log parsing, and log classification. Figure 1 shows the whole picture of our approach. In this framework, input includes cloud system log files or log segments. Log segments denote a series of log entries, and these log entries can be incomplete. The approach labels out bug-induced log entries as the output. Offline process aims at training predictive models/rules using multi-source logs from bug repository, Q&A websites and open cloud platform. Open cloud platform only produces system logs without failures called *normal logs* in this paper so as to verify the ability of recognizing bug-induced logs from both abnormal logs and normal logs. In offline process, all log entries are transformed into word vectors and labeled as *Bug*, *Q&A* and *normal* based on their sources. Then these word vectors are utilized for predictive model/rule construction. In online process, user logs experience several processing steps, and then enter trained predictive model/rule for classification.

Figure 2 displays an example of log process workflow. Our approach cleans the collected logs by removing numerical variables and then split log entries into word vectors. Classification algorithms use these word vectors. To elaborate our approach, we explain two key log processing steps in detail in the following subsections.

A. Log parsing

This step aims to preprocess log entries into vectors as the input of log classification step. For each log entry, straightforward method directly considers it as a whole, and each character in the log entry takes over one attribute of the output vector. This type of methods may lead to the curse of dimension. Therefore, our approach treats the log entry as *bag-of-words*. First, we remove numerical parameters such as ids, URIs, and IP addresses. These parameters are specific in each log entry and contain few information with system problems. Then by utilizing some special symbols such as colon or equal-sign as separators, each log entry is transformed into a high dimensional vector with words as attributes. This may omit some information or features, but largely reduce data dimensions and complexity of log processing, and meanwhile, is easy to implement. For instance, to process the first log entry in

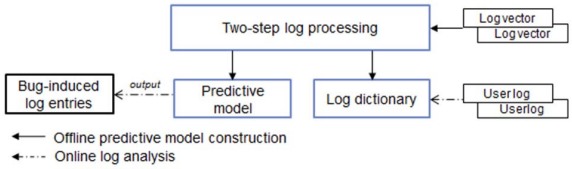


Figure 3: Workflow of the MPIN algorithm

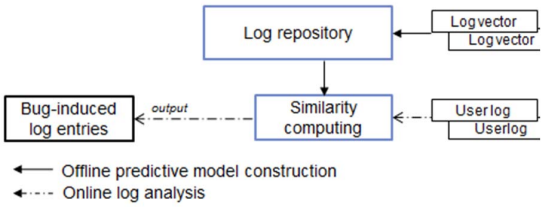


Figure 4: Workflow of the SPIN algorithm

Table 3, we erase special symbols and transform this log entry into a word vector *[FATA, could, not, find, image, no, such, id, registry.access, redhat.com, kollaglue, fedora-rdo-docker-compose]*.

In text writings, the vocabulary usually has a variety of morphed forms, thus reducing inflected (or sometimes derived) words to their stem or root form. This is called stemming process which is widely used for text mining. Similar with text writings, different components, projects and programs may print heterogeneous log expressions in morphed forms. For instance, log entries in table 3 contain "ERROR" and "Error". The two words are exactly the same but with different capitalization, which will increase meaningless data dimensions and interfere the quality of machine learning. Therefore, we further perform a normalization procedure, for instance, changing all upper case letters into lower case letters. Then for the example log entry, the final word vector is *[fata, could, not, find, image, no, such, id, registry.access, redhat.com, kollaglue, fedora-rdo-docker-compose],* which is called *log vector* in this paper.

B. Log classification

The input of this step is *log sequences* composed of several *log vectors* in order of time series generated by log parsing step. There are two types of log sequence: *training log sequence* and *user log sequence.* Typically, a training log sequence consists of *log vectors* parsed from log entries in a certain bug report or Q&A page, while a user log sequence consists of *log vectors* parsed from user system logs to be diagnosed. The order of log vectors in a training log sequence is the order of their corresponding logs in the description of bug report or question post. Offline process utilizes training log sequence, and online process identifies bug-induced user log sequence through predictive model/rule generated from offline process.

In this step, a machine learning algorithm called MPIN is performed, which is a two-step algorithm with an unsupervised clustering algorithm at first and a supervised classification algorithm next. In fact, the proposed approach is extensible to support more and different types of algorithms. To verify this, we provide another algorithm called SPIN based on similarity matching. The details of log classification algorithms are described in the following section.

Algorithm 1 two-step log processing

Definition: *wordvec* indicates the word vector generated in log parsing process:
$wordvec=\{word_1, word_2, \ldots word_n\}$
Scene indicates the set of log entries in a bug report or a question post:
$Scene = \{wordvec_1, wordvec_2, \ldots, wordvec_n\}$
$source\_tag \in \{BUG, Q\&A, NORMAL\}$
Input: key-value pair of word vectors and their labels *wordvec-{source_tag}*
Output: Predictive model

1. Compute the TF-IDF value of each word and generate a equilong *score vector* for each log entry.
2. **Cluster-step**: Cluster all *score vector* with K-MEANS algorithm and then generate a triple table for each log:
   wordvec-{source_tag}-{group_id}
3. Replace word vectors in a scene with the label of their clusters.
   $Scene = \{group\_id_1, group\_id_2, \ldots, group\_id_n\}$
4. **Classification-step**: Train the predictive model with supervised classification algorithm using *Scene* set as train set and *source_tag* as target value vector.

V. LOG CLASSIFICATION ALGORITHMS

MPIN algorithm consists of two procedures: two-step log processing and log dictionary setting up. Two-step log processing aims at extracting appropriate features from training log sequences and generating predictive model. Log dictionary is set up for processing user log sequence while reducing searching and distance computing attempts to decrease the overhead of our approach. Figure 3 illustrates the workflow of MPIN algorithm. Since our approach is flexible and extensible, a relatively simple algorithm called SPIN is also provided for a comparison with MPIN. Next, we elaborate the two procedures of MPIN separately, and present SPIN algorithm at last.

A. Two-step log processing

The challenge of this procedure is to generate appropriate feature vectors through utilizing input log sequences. On one hand, logs are a typical dataset with time-series characters. The order of log entries represents the execution traces of certain programs. Therefore, besides the content of log entries, context relation is also an important feature of log sequences. On the other hand, considering that the total amount of logs is often very large in a system and similar logs may repetitively occur many times, typical strategy clusters logs into several groups called *grouping* so as to decrease the dimensions of each log instance as well as the complexity of computing. There are two types of grouping methods. The first method is source code matching [8][9] which

matches each log with patterns in source code such as "printf" statement. The second method is content-based grouping [10] which utilizes some clustering algorithms based on the text content of logs.

Therefore, we propose the two-step algorithm of MPIN. The first clustering step is a *grouping* step to assign a group id to each log. Since bug logs and Q&A logs are not so complete and from many different projects, source code matching is not available to our situation, thus we design the clustering step as a content-based grouping method. Algorithm 1 shows the whole process of this two-step algorithm in which K-MEANS[17] algorithm is integrated as an example. By considering log data as a bag-of-words, values of each attribute in *log vectors* are transformed into Term Frequency-Inverse Document Frequency (TF-IDF) weight [16], which gives a higher weight on words that are frequent in certain data records but not too common across all records. This value vector is called *score vector* in this paper. The *score vectors* are fed into clustering algorithms, and a large amount of *log vectors* is divided into several groups each of which owns a *group id*. Then each log entry is replaced by its *group id*, and arranged in the same order as prior in the log sequence. Therefore, log sequence is transformed into a much simpler vector defined as *scene* whose attributes are various *group ids*. At last, *scene* vectors are fed into a classification engine to train a predictive model.

In this way, *log vectors* are divided into several groups based on their content similarity. *Log vectors* in the same group have similar text words or expressions, thus the cluster step fully considers the content character of logs. By replacing each *log vector* with its group id, log order is reserved as well.

B. Log dictionary setting up

While dealing with user log sequence, a log dictionary is built to assign each *log vector* in the user log sequence to a certain group. To set up the log dictionary, we choose *log key* as a representative of each group. Log key is defined as the center data instance of each group which is appropriate as the element in log dictionary, because log keys have the longest distance with other groups and the shortest distance with other instances inside a group. Then, each log key and group id perform as a pair stored in the log dictionary. Each *log vector* in user log sequences is assigned to a group according to its similarity with log keys. Then, user log sequence is transformed into *scene* vector, and fed into trained model for final classification.

This will further optimize the efficiency of MPIN algorithm. Once the groups of training *log vectors* are determined after cluster step, user *log vectors* are compared with these *log keys* instead of huge amount of *log vectors*. Then each user *log vector* choose the group of the most similar *log key* as the group of itself.

C. SPIN algorithm

Besides MPIN, we provide another algorithm SPIN based on similarity matching (Figure 4). SPIN maintains a log repository storing all preprocessed *log vectors* in training log sequences. Similarity computing step is flexible and different distance computing methods such as String edit distance, Cosine similarity can be perfectly "plugged in".

Given a user log sequence, SPIN takes it as a set of *log vectors* without considering the context relations of log entries. For each *log vector* in user log sequence, similarity computing step of SPIN computes the similarity value (i.e., *sv*) between this new *log vector* and all the prior *log vectors* maintained in log repository, and then rank the *sv* values in a descending order. The historic *log vector* with the highest *sv* value is the most similar prior record. The output of this algorithm includes identification result for each log vector in the user log sequence and matching result of the most similar *log vector* in log repository. With a little effort, SPIN can furthermore provide the page link to the bug report containing the matching result, and help system managers diagnose root causes and solutions.

VI. EXPERIMENT AND EVALUATION

In this section, we evaluate the proposed approach on two typical open source cloud platforms: Hadoop and OpenStack. Since our work is a brand new idea, we don't have a baseline to compare with, thus we propose five evaluation metrics which are commonly used in machine learning algorithm evaluation, and then give out experiment results. Finally, we present a discussion on these two algorithms.

A. Evaluation Metrics

Our experiment is using 10-fold cross validation which equally splits the dataset into ten pieces. Each time of ten iterations, we use one piece for testing and the other nine pieces for training. To evaluate the performance of classification model, we introduce five metrics: *accuracy, precision, recall, F-measure and variance*.

1) accuracy: the percentage of correct classified logs (including bug logs, Q&A logs and normal logs) in all logs.

$$accuracy = \frac{\#correct\ classified\ \log number}{\#all\ \log number}$$

2) precision: the percentage of correct recognized bug logs in all recognized bug logs.

$$precision = \frac{\#correct\ recognized\ bug\ number}{\#all\ recognized\ bug\ \log number}$$

3) recall: the percentage of correct recognized bug logs in all bug logs.

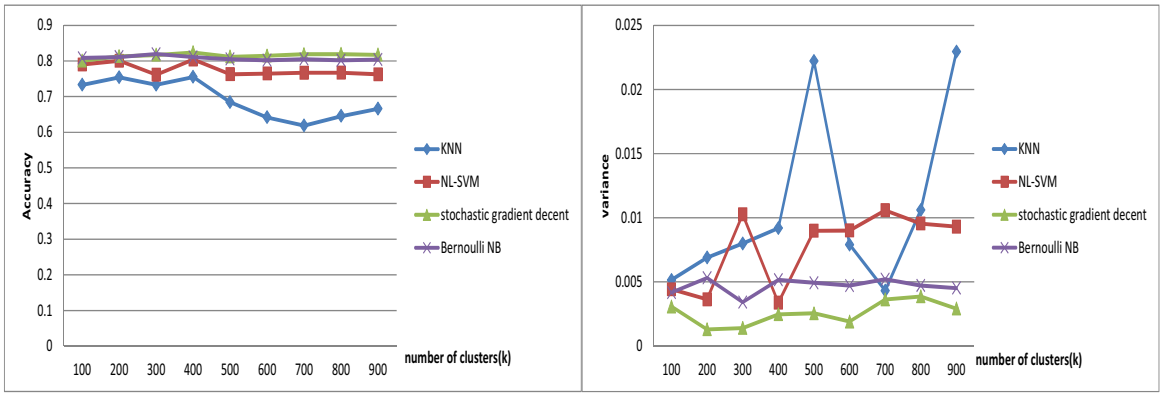
$$recall = \frac{\#correct\ recognized\ bug\ \log number}{\#all\ real\ bug\ \log number}$$

4) F-measure: a comprehensive description to the result that combines recall and precision.

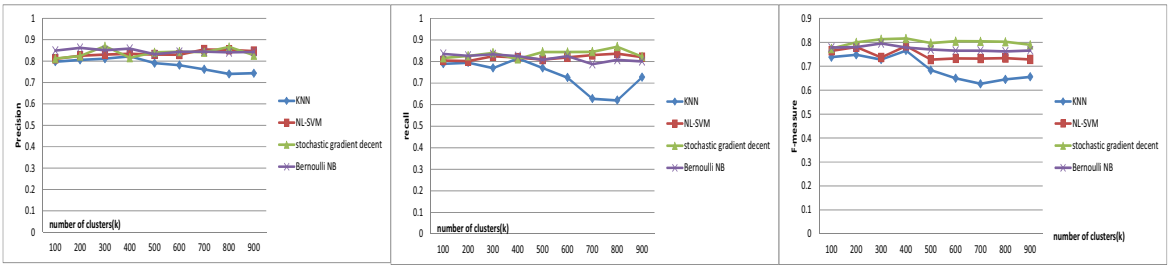
$$Fmeasure = \frac{2 * precision * recall}{precision + recall}$$

5) variance: the variance of accuracy results in ten times computation in cross validation.

$$variance = \frac{\sum_{i=1}^{10}(accuracy_i - accuracy_{avg})^2}{10}$$



a) Accuracy of cross validation b) Variance of accuracy



c) Precision of cross validation d) recall of cross validation e) F-measure of cross validation

Figure 5: Experiment result of MPIN algorithm for OpenStack

B. Experiment on OpenStack

To generate normal logs, we set up an OpenStack platform composed of two instances: one for controller and network node, the other for compute node, as a typical OpenStack deployment in our lab. Requests (e.g. spawning instances, destroying instances, starting instances, etc.) are generated as the workload of the running system and 2741 logs are collected. Since the experiment platform is for generating testing normal logs, a small size of platform is enough, meanwhile, easy to control and track.

MPIN is flexible both in algorithms and parameters which can be adjusted to balance efficiency and accuracy in different systems. To evaluate the performance of MPIN, specific algorithms are embedded in the two-step log processing with K-means algorithm in cluster step and four different classification algorithms including *KNN, NL-SVM, stochastic gradient decent* and *Bernouli NB* in classification step. Sensitivity test is also performed by initiating *k* as 100, 200, 300, 400, 500, 600, 700, 800 and 900 in K-means algorithm (Results are in figure 5). By utilizing different algorithms and parameters, we can test our approach comprehensively and thoroughly.

Result shows that when *k* is assigned 400, our approach produces the best precision and accuracy. Meanwhile, *stochastic gradient decent* method produces the best results among all algorithms. In the best condition, our method reaches 82.38% accuracy and 82.33% precision. As for the evaluation of SPIN, we measure the distance between each log vectors based on *string edit distance*.

Table 4: Results of MPIN and SPIN for OpenStack

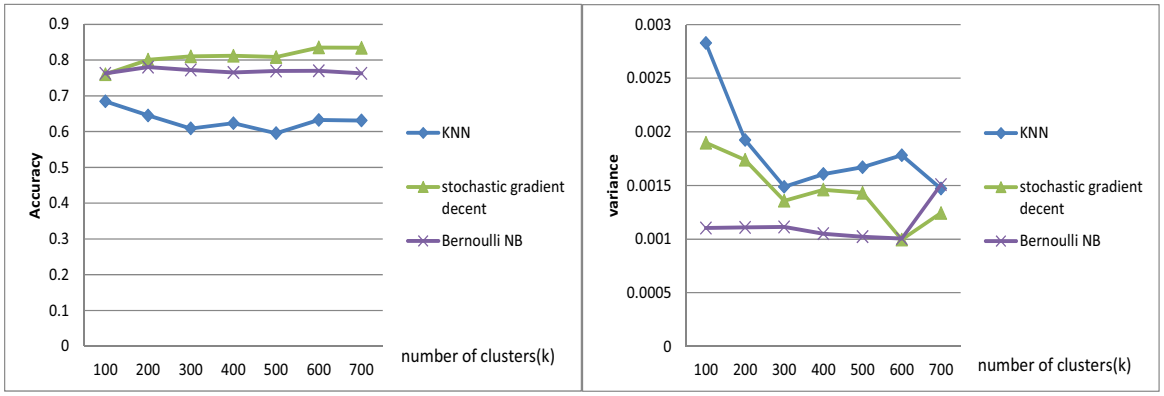
Metrics	MPIN	SPIN
Accuracy	82.38%	85.78%
Precision	82.33%	83.90%
Recall	81.64%	87.03%
F-measure	81.56%	84.96%
variance	0.0025	0.0044

The performance of this method is shown in Table 4 with an accuracy of 85.78% and precision of 83.90%.

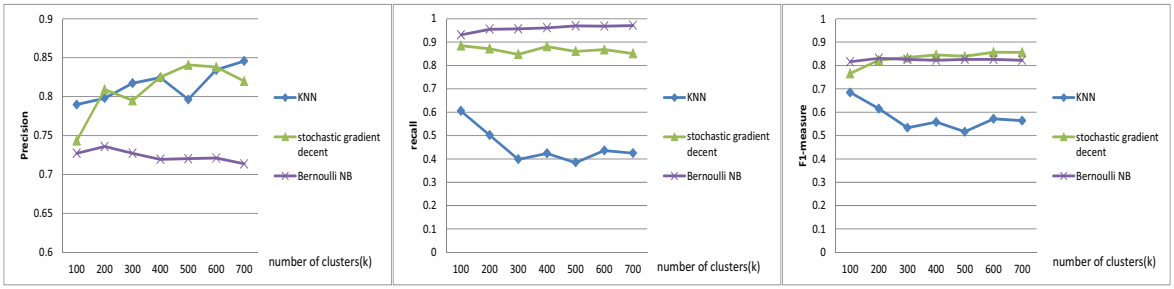
C. Experiment on Hadoop

Similar with experiment of MPIN on OpenStack, we choose K-means algorithm in cluster step and measure the sensitivity of parameter *k* in K-means. In classification step, three different classification algorithms including *KNN*, *stochastic gradient decent* and *Bernouli NB* are implemented. Results are shown in Figure 6 and Table 5. When k is assigned to 400 and *stochastic gradient decent* is applied, MPIN obtains the best results with 81.18% accuracy and 82.52% precision. *String edit distance* is also plugged in the experiment of SPIN. Results are shown in Table 5 with 85.07% accuracy and 60.34% precision leading to a 70.46% F-measure.

D. Discussion



a) Accuracy of cross validation b) Variance of accuracy



c) Precision of cross validation d) recall of cross validation e) F-measure of cross validation

Figure 6: Experiment result of MPIN algorithm for Hadoop

Both MPIN and SPIN performs good evaluation results in almost all metrics in the two open cloud platforms. In OpenStack experiment, the performance of SPIN is a little better than MPIN, however, the *variance* is much larger. *Variance* represents result fluctuation range and the dependency to different training set. *Variance* value of 0.0044 denotes that the accuracy result fluctuates from about 78% to 92%. This is because SPIN is too simple that directly depend on the source data. If data points of different groups have an overlapped area, SPIN cannot extract higher level features or map vectors to higher dimension leading to a large *variance*. For instance, if testing data points are in overlapped area, SPIN can produce completely random results, however, if testing data points are in other area, SPIN can make a particularly accurate prediction.

In Hadoop experiment, MPIN is as stable as OpenStack while SPIN produces a very low precision. This means that many Q&A logs and normal logs are mistakenly classified into bug logs. We analyze the Hadoop data set of our experiment and propose two causes of this low performance. Firstly, on one hand, Hadoop components are programmed in Java in which many logs are produced through log4j[30] leading to a similar structure. On the other hand, many log entries from bug reports and Q&A sites are java exception traces generated by native jdk packages. Therefore, logs are relatively similar with each other. MPIN uses TF-IDF value for each word to deal with this problem. Words that

Table 5: Results of MPIN and SPIN for Hadoop

Metrics	MPIN	SPIN
Accuracy	81.18%	85.07%
Precision	82.52%	**60.34%**
Recall	88.10%	85.52%
F-measure	84.56%	**70.46%**
variance	0.0014	0.0011

appear many times in the whole text composed of all *log vectors* will be given a low value. However, SPIN can hardly deal with this problem. Secondly, *String edit distance* computing is a simple algorithm. Considering of java exception traces, nearly all of them contain exception sentences such as “java.lang.NullPointerException”. Most parts of these sentences are the same, so their *String edit distance* is determined largely by their length rather than textual diversity which is not reasonable. We will deal with these problems in our future work, and apply other distance computing algorithms.

VII. Conclusion and Future Work

In this paper, we propose an approach to pinpointing bug-induced failure in logs for open cloud platforms. We collect data from bug repository, Q&A websites and testing platform in the lab. After a series of log processing, we transform log entries into *log vectors*. To generate a predictive model/rule based on these vectors, we present two algorithms called MPIN and SPIN. MPIN integrates

machine learning algorithms including a clustering step and a classification step. SPIN chooses to build a log repository and compute the similarity between user logs and training logs. Results show that MPIN has a better performance.

In the future, we intend to get more training data from the website and apply our approach in cloud systems beyond the lab. As for algorithm, so far we only consider text features of log entries. We plan to incorporate other features and try other algorithms to help improve classification accuracy and reduce the overhead. Moreover, we look forward to extract solutions to software bugs or misconfigurations for automatic bug fixing and failure recovery.

ACKNOWLEDGMENT

This work is supported by Key Program of National Natural Science Foundation of China (Grant No. 61232005) and Shenzhen Municipal Science and Technology Program (Grant No. JSGG2014051616 2852628).

REFERENCES

[1] Gunawi H S, Hao M, Leesatapornwongsa T, et al. What Bugs Live in the Cloud?: A Study of 3000+ Issues in Cloud Systems[C].Proceedings of the ACM Symposium on Cloud Computing. ACM, 2014: 1-14.

[2] OpenStack, https://www.openstack.org/

[3] Kamal Kc, Xiaohui Gu, ELT: Efficient Log-based Troubleshooting System for Cloud Computing Infrastructures, 2011 30th IEEE International Symposium on Reliable Distributed Systems.

[4] Today's outage for several Google services. http://googleblog.blogspot.com/2014/11/1/todays-outage-for-several-google.html

[5] Update on Azure Storage Service Interruption, http://azure.microsoft.com/blog/2014/11/19/update-on-azure-storage-service-interruption

[6] Google Traces Sunday's Cloud Outage to Faulty Patch, http://www.datacenterknowledge.com/archives/2015/03/09/google-traces-sundays-cloud-outage-to-faulty-patch/

[7] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, and C. Zhai, Have Things Changed Now?: An Empirical Study of Bug Characteristics in Modern Open Source Software, in Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability, New York, NY, USA, 2006, pp. 25–33.

[8] Xu W, Huang L, Fox A, et al. Detecting large-scale system problems by mining console logs[C]. Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel. 2010:37-46.

[9] Xu W, Huang L, Fox A, et al. Online System Problem Detection by Mining Patterns of Console Logs[C]. Data Mining, IEEE International Conference on. IEEE, 2009:588-597.

[10] Juvonen A, Hamalainen T. An Efficient Network Log Anomaly Detection System Using Random Projection Dimensionality Reduction[C]. 2014 6th International Conference on New Technologies, Mobility and Security (NTMS). 2014:1-5.

[11] Adam J. Oliner, Alex Aiken et al. Alert Detection in System Logs. Data Mining, 2008. ICDM 08:959-964.

[12] Lim C, Singh N, Yajnik S. A log mining approach to failure analysis of enterprise telephony systems[C]. Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on. IEEE, 2008:398-403.

[13] Akinori Ihara, Masao Ohira, Ken-ichi Matsumoto. An Analysis Method for Improving a Bug Modification Process in Open Source Software Development. Proceedings of the 28th Annual ACM Symposium on Applied Computing.2009:135-144

[14] Kypros Constantinides, Onur Mutlu, Todd Austin. Online Design Bug Detection: RTL Analysis, Flexible Mechanisms, and Evaluation. Microarchitecture, 2008. MICR.

[15] Rajaraman, A. Ullman, J. D. (2011). "Data Mining". Mining of Massive Datasets. *pp. 1–17.* doi:10.1017/CBO9781139058452.002. ISBN 9781139058452.

[16] Sparck Jones.K. "Scoring, term weighting, and the vector space model". Introduction to information retrieval, p 100.

[17] Hartigan, J. A.; Wong, M. A. (1979). "Algorithm AS 136: A K-Means Clustering Algorithm". Journal of the Royal Statistical Society, Series C 28 (1): 100–108. JSTOR 2346830.

[18] Scikit-learn. http://scikit-learn.org/stable/

[19] Schulz, Klaus U.; Mihov, Stoyan (2002). "Fast string correction with Levenshtein automata". International Journal of Document Analysis and Recognition 5 (1): 67–85. doi:10.1007/s10032-002-0082-8. CiteSeerX: 10.1.1.16.652

[20] Launchpad. https://launchpad.net/

[21] Ask.openstack. https://ask.openstack.org

[22] Jeff Atwood (2008-04-16). "Introducing Stackoverflow.com". Coding Horror. *Retrieved 2009-03-11*.

[23] Mager, Andrew (September 27, 2009). "Find the answer to anything with StackExchange". The Web Life. ZDNet. *Retrieved December 16, 2012*.

[24] Ashlee Vance (17 de marzo de 2009). "Hadoop, a Free Software Program, Finds Uses Beyond Search". New York Times. Consultado el 20 de enero de 2010.

[25] Hadoop in JIRA. https://issues.apache.org/jira/browse/HADOOP

[26] Humphrey, Benjamin (October 2010). ""Ask Ubuntu" Stack Exchange site out of beta, new awesome domain". OMG! Ubuntu!. *Archived* from the original on 17 October 2010. Retrieved 22 October 2010.

[27] Programmers. http://programmers.stackexchange.com/

[28] ServerFault. http://serverfault.com/

[29] Clarke, Jason (August 20, 2009). "Super User - question and answer site for power users". DownloadSquad. *AOL*. Retrieved December 16, 2012.

[30] "Log4j 2 Asynchronous Loggers for Low-Latency Logging - Apache Log4j 2". Logging.apache.org. 2014-07-12. Retrieved 2014-07-24.