Zou DQ, Qin H, Jin H. UiLog: Improving log-based fault diagnosis by log analysis. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 31(5): 1038–1052 Sept. 2016. DOI 10.1007/s11390-016-1678-7

UiLog: Improving Log-Based Fault Diagnosis by Log Analysis

De-Qing Zou, *Member, CCF*, Hao Qin, and Hai Jin, *Fellow, CCF, Senior Member, IEEE, Member, ACM*

Services Computing Technology and System Laboratory, Huazhong University of Science and Technology Wuhan 430074, China

Big Data Technology and System Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China

Cluster and Grid Computing Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China

School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

E-mail: deqingzou@hust.edu.cn; qinhao8810@gmail.com; hjin@hust.edu.cn

Received May 18, 2015; revised February 4, 2016.

Abstract In modern computer systems, system event logs have always been the primary source for checking system status. As computer systems become more and more complex, the interaction between software and hardware increases frequently. The components will generate enormous log information, including running reports and fault information. The sheer quantity of data is a great challenge for analysis relying on the manual method. In this paper, we implement a management and analysis system of log information, which can assist system administrators to understand the real-time status of the entire system, classify logs into different fault types, and determine the root cause of the faults. In addition, we improve the existing fault correlation analysis method based on the results of system log classification. We apply the system in a cloud computing environment for evaluation. The results show that our system can classify fault logs automatically and effectively. With the proposed system, administrators can easily detect the root cause of faults.

Keywords fault diagnosis, system event log, log classification, fault correlation analysis

1 Introduction

With the widespread development of cloud computing, computer systems are becoming increasingly complex, and the components in the entire system are also becoming diverse. If some key part fails, the whole system may be seriously implicated due to frequent interactions and high coupling. An effective fault detection and analysis method, which plays an essential role in the management of large systems, can help system administrators to locate the fault and identify the cause.

System and software logs are important sources for diagnosing system and software faults. However, for large systems, various components will generate amounts of log information in real time. If a fault occurs, it is difficult to extract useful information from the system efficiently and locate the fault accurately. Traditionally, we have to manually extract the useful information from vast amounts of data, which would seriously delay the response time of fault recovery. Therefore, a unified management system for fault log analysis is required, which can automatically identify the fault type and analyze the cause of faults.

A lot of studies have been proposed on log-based fault analysis focusing on the following aspects. The first one is log collection and analysis, which investigates how to effectively and efficiently gather log information[1]. The information will be used to acquire profile information for analyzing the system situation[2] and extracting the feature[3]. The second one is fault location, which aims at determining the control flow of software through logs[4] and using the source code[5] to locate the position of the faults. The studies mentioned above use the log fault analysis in different scenarios. However, few studies aim to design an integrated fault analysis system from perspective of system administrators. The third one is fault correlation analysis. In

Regular Paper

This work was supported by the National Basic Research 973 Program of China under Grant No. 2014CB340600, the National Natural Science Foundation of China under Grant No. 61272072, and the Program for New Century Excellent Talents in University of China under Grant No. NCET-13-0241.

the multi-node environment, the fault propagation is a critical problem for fault diagnosis. This area focuses on using log information to determine the connection between different faults. Some researchers have used time and spatial correlations to find this connection, but few of them have considered the meaning of logs. Furthermore, the accuracy is not satisfactory[6].

An integrated fault log analysis system can assist administrators in performing fault analysis, improve the administrators' ability to respond to the system fault, and reduce the time required in fault processing. We implement a new fault classification method to assist administrators to understand the computer system and use fault correlation analysis to locate the root cause of faults. In this paper, we propose an integrated fault log analysis platform (UiLog) to collect and manage various components' logs. It can analyze logs for administrators to quickly locate faults and analyze the cause of faults.

The platform consists of three components: 1) the fault log collection module, which collects log data from various components, 2) the fault log analysis module, which classifies logs by the fault type in real time, and 3) the fault log correlation analysis module, which collects fault logs caused by the same root fault as a tuple and tries to find the root cause of those faults. UiLog has been deployed in a practical cloud environment, which helps administrators to troubleshoot and find the root cause of faults. Our main contributions are as follows.

We propose a novel classification method for fault logs, using the fault keyword matrix to improve accuracy. It reduces the time required to determine the fault type and the workload of manual processing. Moreover, this method makes it convenient to add a new fault type without recalculating.

We improve the existing log correlation analysis. In our study, we combine the result of the fault classification and correlation analysis. Our method uses the fault type of the logs as one factor in determining the size of the time window. It improves the accuracy of the log correlation and the location of the root cause of faults.

In this paper, we illustrate a comprehensive log management system, which can help administrators to quickly grasp the operation status of the system and save time with troubleshooting.

The rest of this paper is organized as follows. Section 2 discusses the background of log analysis and related work. Section 3 outlines the proposed structure of the UiLog system and describes the implementation of the system. Section 4 describes the evaluation of our system, and Section 5 presents our conclusions.

2 Background and Related Work

2.1 Background

The objective of log collecting is to extract useful information from fault logs. The major techniques of previous work mainly relied on regular expressions[7]. However, to develop rules of regular expressions requires experts in different fields[8]. In addition, the deployment of a new application and the upgrading of a system will change these templates of log frequency. It is, therefore, difficult to design these regular expressions[9].

Thus, there are other automatic methods used in extracting templates from logs. Visualization is a new technology that helps managers to understand systems. Takada and Koike[10] proposed an information visualization system for monitoring and auditing computer logs.

For log classification, many famous algorithms have been designed for clustering high-dimensional data like CURE, MAFIA, and CLIQUE[11]. However, considering the characteristics of logs, these algorithms are not suitable. The log contains data with different attribute types, not only high-dimensional data. A good algorithm should discover the structure of the log that exists in the subspaces of the high-dimensional data and ignore the order of input data[12]. Moreover, for log classification, the position of words in the log is more important than that in other normal texts, as the sequence of words is not overwhelming.

In the area of log analysis and fault diagnosis, data correlation is an important issue. It aims to reconstruct the system fault by grouping logs that have the same fault source. This is based on the fact that many software and hardware detectors are triggered by the same fault. Data correlation will generate reports of multiple faults[6]. In addition, these reports may repeat more than once[13].

Correlation techniques can be distinguished based on time, space, and content. Time correlation is based on the assumption that fault logs caused by the same fault are close in time. An important methodological method for time correlation is the tuple heuristic. It uses time windows to collect logs. All of the same source fault logs will fall into a specific time window called tuple. The critical point in the time correlation is the se-

lection of an appropriate size for the time window. Spatial correlation[13] adds multi-node support from time correlation. Considering the topology among the multi-node environments, spatial correlation can distinguish a log from different nodes in the same time. In addition, this method also uses time-correlation techniques to gather logs in a single node.

Content-based correlation techniques are based on grouping the system logs by checking the specific contents in the log messages. Pecchia *et al.*[14] applied the lift data by the mining operator to find frequent event patterns. It starts with log contents and attempts to isolate accidental patterns.

2.2 Related Work

Various studies are looking for how to understand the meaning of logs, which may be useful for detecting faults in cloud computing systems[15]. These studies have displayed many interesting features. For instance, Stearley[16] found that fault type could not be detected from logs only by words. The position of each word is a powerful indicator to distinguish different messages.

Researchers have also found other ways to diagnose systems, such as application console logs[17]. However, this technique is limited by application-specific anomalies and requires source codes[18]. Wen *et al.*[19] proposed a fault detection and diagnosis method based on the observer for the nonlinear modeling ability of the neural network. However, this method cannot provide the cause of faults very well.

In addition, several techniques and algorithms for automatic log classification have been developed. Park *et al.*[20] attempted to classify different raw logs into a set of categories. Moreover, Li *et al.*[21] tried to use the modified Naive Bayesian Model (NBM) and Hidden Markov Models (HMMs) to classify event logs based on the IBM Common Base Event (CBE) format. On the other hand, SLCT[11] and Loghound[22] are designed specifically to discover the format of logs and classify rows of logs automatically. They use two similar algorithms, which are useful to extract the template from logs. In detail, the two algorithms need the administrator to provide a support threshold value as input initially. The support threshold is used to control the size of the output for the two tools. Therefore, these tools can be customized by the manager for different systems. However, these algorithms require customizing filtering rules for different systems. In addition, with the system and software updates, the filtering rules may also need to be adjusted. Hence, these algorithms cannot adapt to changing systems because they lack an online method.

Similar to correlation analysis, time and spatial correlation techniques have been applied to a variety of large-scale computing systems[23]. Hansen and Siewiorek[6] described the Tandem system, which represents a heuristic for selecting a single correlation time window. They found that there was a fixed pattern between the size of the time windows and the number of tuples for given logs. This pattern could be drawn like an "L"-shaped curve. According to this paper, the inflection point of the curve represents the generation time of the system log. Therefore, the appropriate size of the time window should be chosen right after the inflection point. The current trend of this study is to use the tuple with a fixed value for a time window, such as 5 minutes[14] or 20 minutes[24]. In the actual environment, there is not a variable time window to be used in the time correlation.

3 UiLog System

3.1 Overview of UiLog

The Unify Log Analysis System (UiLog) is a fault analysis and diagnosis system, which collects the system log information of each component and tracks logs for statistics.

It can accurately and globally analyze the system combined with the fault correlation analysis, which assists administrators in managing the operation of the system. Through the fault classification system, UiLog learns the classification rules from a training set of artificial classification. After that, the system determines the fault type in real time, according to the rule library. In addition, the fault correlation analysis can be deployed when system administrators need to diagnose a fault. Considering the propagation of the fault, UiLog can mine the association among the faults. Faults generated from the same root cause can be collected into the same cluster. The result will be presented as a fault propagation tree sorted by a causal relationship of the occurrence.

To implement the procedure, we apply three modules to represent the UiLog: the fault log collection, the fault log analysis, and the fault log correlation analysis. As shown in Fig.1, the fault log collection module collects logs from the entire target node. It collects software and system logs from all of the components. The analysis node is responsible for fault analysis and

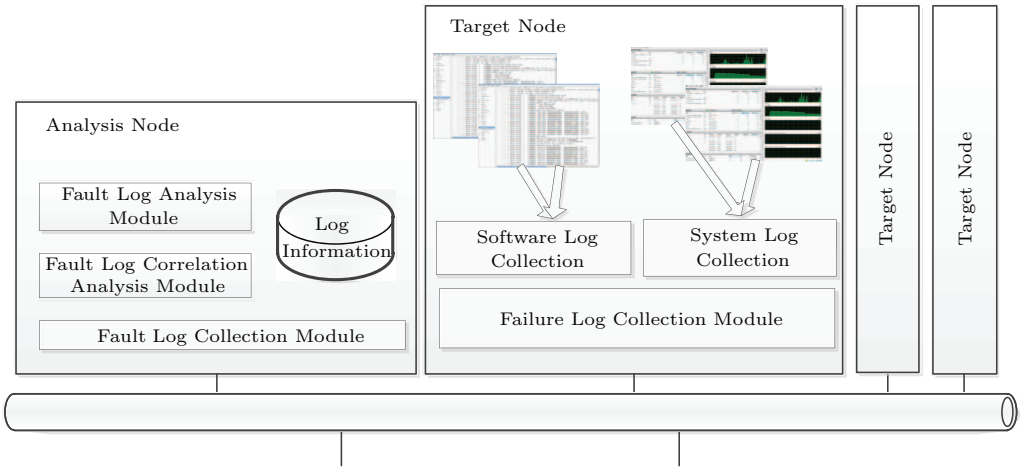


Fig.1. Architecture of UiLog.

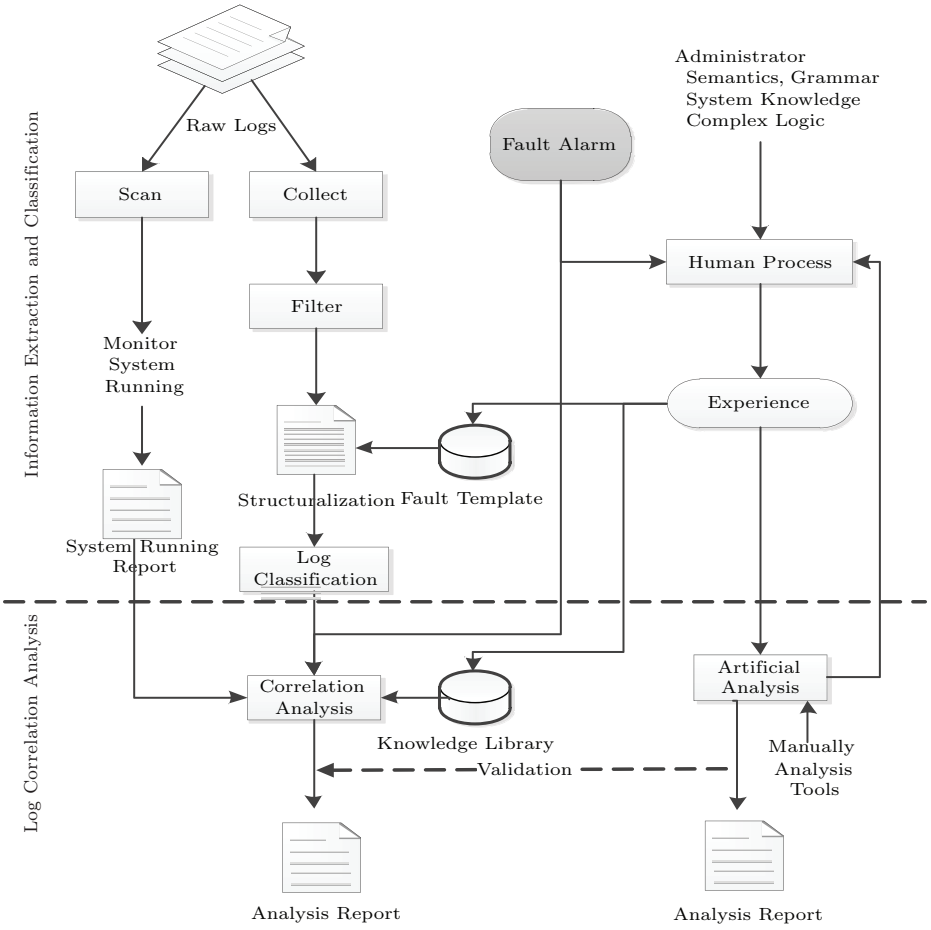


Fig.2. Workflow for log analysis and fault diagnosis.

diagnosis. In the analysis node, the fault log collection module stores all logs in a log information database for analyzing.

The procedure of the fault log analysis module is shown in Fig.2. The fault log collection module will be deployed seperately into the target node for log collection and into the analysis node for storage. This module will gather the software log and the system log from the target node and store log information into the log information database. In the analysis node, the fault

log analysis module will extract the log structure from the fault log collection module. At first, these structures will be classified into different fault types by the administrator. The fault log analysis module will learn the classification rules from the administrator and store these rules into a fault template database. After that, the fault log analysis module will automatically classify the logs to make a log diagnosis.

The correlation analysis module will use the result of the fault classification and human knowledge to provide an automatic correlation analysis report from an artificial analysis.

3.2 Fault Log Analysis

The fault log analysis module aims at classifying the logs into different log types. We adopt an example to illustrate the procedure: when an administrator analyzes the logs (Table 1), the first step is to determine whether the logs are part of the fault record. In this part, the most regular method is to find keywords. In Table 1, we can determine that both logs belong to the fault log because of the words "fatal", "failed", and "error". The second step is log analysis. The system administrator will weed out the details of the log information to determine the cause of the fault. The first row in Table 1 shows "read from socket failed," which indicates a socket problem. Similarly, the leaving information in the second row means the required file cannot be found. The third step is log classification. It is easy to determine that the first log belongs to the network fault, and the subcategory is the remote network connection fault.

Table 1. Example of Log Messages

Date	Host	Device	Message
2013-11-25 02:39:34	f1	sshd[18108]	fatal: read from socket failed: connection reset by peer
2013-11-23 16:57:13	f0	httpd[27807]	[error] [client 192.168.63.15 9] File does not exist: /var/www/html/favicon.ico

Through the above analysis, we can conclude that if we can deal with the semantic analysis of the keywords and determine the fault type in advance, a large load of work can be automatically processed by log analysis and classification. In the example above, the analysis module can understand the meanings of five words: "sshd", "socket", "connection", "file", and "exist". In fact, through a detailed analysis of a large number of logs (Fig.3), for this particular system, we can find that the number of these keywords is actually limited. We can obtain almost all of the keywords for classification by learning from a small training set.

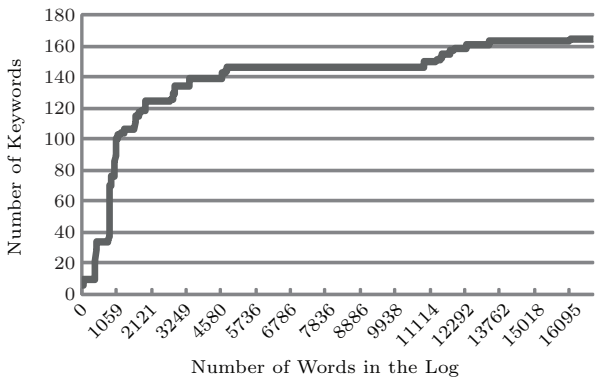


Fig.3. Number of keywords that appear in the fault logs.

Based on our observation, we propose a new fault log classification method to achieve faster and more accurate results.

The UiLog log classification method is divided into five steps (Fig.4). These steps will gradually extract log information for fault classification and remove irrelevant content. The steps are as follows:

step 1: preprocess logs;
step 2: extract invariants;
step 3: filter template information;
step 4: obtain fault keyword matrix;
step 5: classify log information.

Among them, the second and the fourth steps have different functions in two different periods. One is the initial learning period before the system operation. The other is an adjustment period in the system in which we can obtain user feedback. Moreover, the third step only runs in the learning stage. This step is the preparation step, which is required to obtain the fault keyword matrix. The details of this step will be described later.

The input of the whole algorithm is the log data flow obtained by the fault log collection module. The output has two parts. One is the log classification rules (output at step 4) based on the training set. The rules in our algorithm appear as the fault keyword matrix. The other is the fault category of every log (output at step 5). The following subsections describe each step of the algorithm in more detail.

3.2.1 Log Preprocessing

Log preprocessing contains two parts: the filtering of redundant logs and meaningless words.

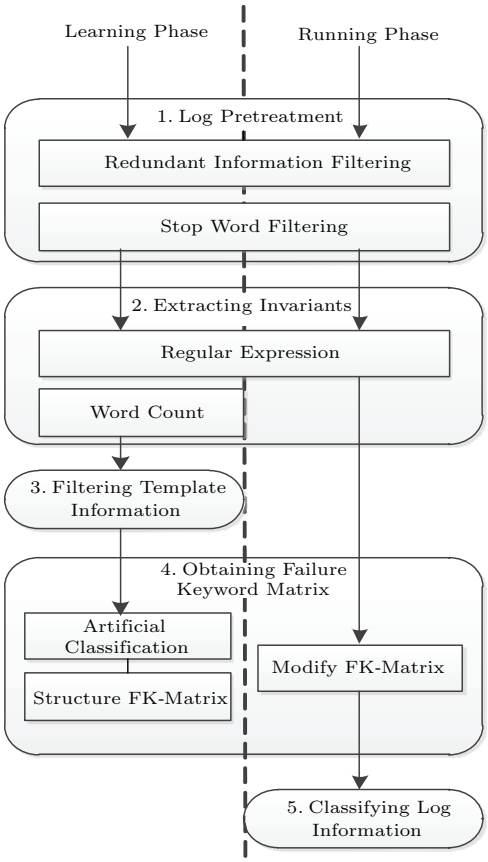


Fig.4. Main steps of log classification.

The first part is to filter repeated logs that are generated by the system. Many software faults are not sufficient to cause a collapse in the computer system, and the component will send persistent fault messages. These messages will waste the storage resources and disturb the fault log analysis. Through the statistics of fault logs, we can find that these redundant logs are generated in a small period of time. Therefore, we can choose an appropriate threshold to filter the duplicate logs. When detecting whether the first log is the same with the second log and that the generated time of these two logs is within the predetermined time threshold, UiLog will remove one of the logs.

The second part is the log filter of meaningless words. We use English Stop Word Table① to filter meaningless words. In addition, we also filter out most of the adjectives and adverbs.

3.2.2 Extracting Invariants

After preprocessing, the next step is to extract the template information. The template of the log is used for classification. This step can reduce the solution space and compress the size of the fault keyword matrix. In addition, the template will be used to match logs for fast classification.

The variable of logs refers to the parts that are always changing in the text of the log, including the number, IP address, directory, file name, program name, port, and so on. To extract the invariant, we can judge the purpose of why the programmer has written this log by returning to the original static information of the log output statement as in the following

sprintf(*message, "Connection from %s port %d," ipaddress, portnumber*).

Extracting invariants has two steps. The first step is to remove the parts, such as the directory, IP, and address of the memory. UiLog uses regular expressions to match the parts. Here, we use a token to replace the invariant that has a different meaning, instead of "*".

In the second step, we use statistical methods based on word frequency. This step uses the improved term frequency-inverse document frequency (TFIDF) to calculate the frequency of each word and arrange a proper weight to each word. Finally, the administrator can choose an appropriate upper limit to filter the words.

In the learning period, UiLog will implement the above two steps to obtain an accurate classification template of the log. While in the running period, it only executes step 1. Because we can use other approaches to classify logs which will be described in Subsection 4.1, this stage is just for removing the interference.

3.2.3 Filtering Template Information

Before classification, we use an automatic classification approach to classify the logs in order to further reduce the manually determined space.

The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm[25] can be adopted for automatic clustering. The greatest advantage of this algorithm is that it does not need to set the number of clusters in advance. It will automatically divide the sample during the cluster. DBSCAN will scan every log. If a log is close enough to an existent cluster, the log will be marked as the one belonging to this cluster. If not, DBSCAN will create a new cluster and a new label, and it will then add the log to the new cluster.

For a classification algorithm, there are two key points. The first one is the distance function. The distance function is used to measure the distance between

①http://www.ranks.nl/stopwords/, Aug. 2016.

a log and a cluster. Considering the text of the log, a suitable function should not only judge the number of different words but also be sensitive to the structure of the log text. The position of the word is more important than the content. We use the Levenshtein distance (LD)[26] as our distance function. The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e., insertions, deletions, or substitutions) required to change one word into the other.

However, the original LD is complex for two logs because it focuses on the alphabet of words. In our system, the word is the smallest unit to identify. For greater accuracy and efficiency, we can modify the original Levenshtein distance. Because the difference of the length of the logs will cause the great interference of LD, the length is a factor in LD to reduce the effect of the length when calculating the distance.

We define our log Levenshtein distance (LLD). The distance between log A and log B is described in (1).

$$LLD\,(A,B) = \frac{2 \times LD\,(A,B)}{length\,(A) + length\,(B)}. \qquad (1)$$

Because $LD(A, B)$ is the original Levenshtein distance, $length(\ )$ indicates the length of the log.

The second key point is an appropriate threshold. When the distance of the two logs is less than the threshold, we believe that the two logs are similar. Fig.5 illustrates the number of clusters of DBSCAN when different thresholds are adopted. The number of clusters is only one factor for choosing a suitable threshold. Another important factor is whether the logs in the same cluster belong to the same fault type. This factor relies on subjective judgment. After comparing the quality of logs in each fault type, we choose 0.5 as the threshold in StrongCloud.

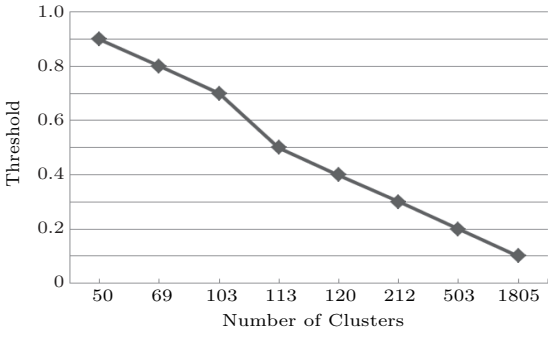


Fig.5. Number of clusters in different thresholds.

After the automatic classification by using DBSCAN, a manual classification is imported to judge the result. We check every cluster to ensure the result is appropriate. The workload is small here because before the manual classification, two steps "log pretreatment" and "filter template information" have been adopted. The number of fault logs has become small for the automatic classification in DBSCAN.

3.2.4 Obtaining Fault Keyword Matrix

When the system is in the learning stage, this step will learn the result of the artificial classification. However, in the running stage, this step will modify the classification's rules.

Through acquiring filter template information, the remaining number of raw logs is less than that of the original. Fig.6 shows the number of logs before and after filtering in different months. We can find that the logs which need the administrator to classify are only a small part.

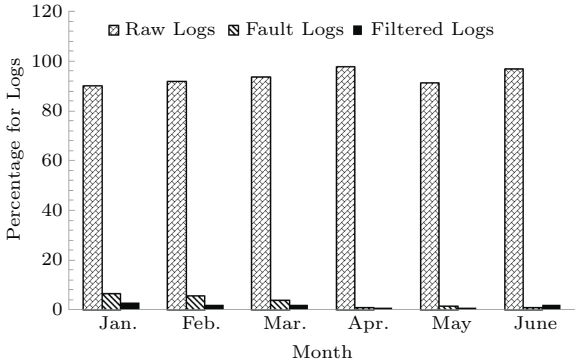


Fig.6. Percentage of log distribution in half a year.

In the learning stage, the administrator first needs to manually classify the logs. Here we post the actual fault catalog (Table 2) in the StrongCloud cluster environment. In the last step, the automatic classification has already labeled each cluster. Now the administrator will first modify the automatic label and then adjust the result of the automatic classification.

After indicating the category, the result will be stored in the way of "content: mark". The label is a number that represents the fault type, as shown in Table 3.

Next, UiLog needs to learn the results of the artificial classification. Here we propose the fault keyword matrix (FK-matrix) for saving the learning results of the artificial classification.

Table 2. Illustration of Fault Types

Category	Description
Disk	Errors about SCSI disk or damaged sectors
Device (DEV)	Errors related to peripherals and PCI cards
Input/Output (I/O)	Errors about drive or communication
Memory (MEN)	Memory-related errors
Network (NET)	Communication issues
Processor (PRO)	Processor exceptions, machine check issues
Authentication (Auth)	Authentication errors
Web	Web engine errors
Database (DB)	Errors about database
File	File system errors

Table 3. Example of Fault Classification for Logs

Content of Log	Label	Meaning
INFO: task * blocked * more * |NUM| seconds	14	Disk
udevd (|NUM|): |DIR| is deprecated, please use |DIR| instead	11	File
pam_succeed_if(*): error retrieving information about user *	7	Authentication
Kernel reported iSCSI connection |NUM| error * state	6	Drive
* received packet with * address * source	5	Network

Note: "*" represents a useless vocabulary for log fault classification. These words will be replaced in the extraction of the invariants.

The FK-matrix (matrix $\boldsymbol{A}$) is a two-dimensional matrix that is constructed by the probability that each word appears in each fault type in the template. It is illustrated in (2).

$$\boldsymbol{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}. \tag{2}$$

The FK-matrix is an $m \times n$ matrix. m represents the different number of words in all of the sample logs, while n represents the number of fault types, and $a_{i,j}$ denotes the probability of the i-th word belonging to the j-th catalog. (Note: $a_{i,j}$ is only a relative probability factor, not the true probability.)

Through the FK-matrix, we know the distribution of the keywords in different fault types. The FK-matrix can easily save the results for automatic learning. Further, when the system is running, it can easily add a new fault type and dynamically modify the FK-matrix in real time. In addition, using the FK-matrix can quickly determine the fault type of the logs through calculating the probability of different types in the matrix. Furthermore, if a log has a high probability with more than one different type, UiLog will ask the administrator to determine the fault type of this log and fix the FK-matrix.

The following describes how to calculate $a_{i,w}$. The value of $a_{i,w}$ is used to determine whether a word belongs to a certain fault type. As the value indicates the frequency of each word in a particular type of fault, we consider that the probability of the word i appearing in the fault type w is the ratio between the number of word i in type w and the total number of words in type w. The basic formula is described in (3). $P(i, w)$ represents the probability that the i-th word appears in the fault type w, and $count(i, w)$ represents the times that the i-th word appears in the fault type w.

$$P(i,w) = \frac{count(i,w)}{\sum_{j=1}^{m} count(j,w)}. \tag{3}$$

However, this formula only considers the distribution of different words in the same fault type, and it ignores the same word between different fault types in the log template. For example, if word i only appears in type w, then we believe that a log is very likely to belong to the fault type w as long as it contains word i, no matter how many times word i appears in type w. Therefore, we can amend the formula by adding a scale factor, as shown in (4):

$$K(i,w) = -\log\left(\frac{sum(i) - count(i,w)}{sum(i)}\right) + 1. \tag{4}$$

$sum(i)$ represents the times that word i appears in all of the fault types of the template library, namely in (5):

$$sum(i) = \sum_{i=1}^{n} count(i,t). \tag{5}$$

$K(i, w)$ indicates the importance of word i in type w, and it is in inverse proportion to the frequency in which word i occurs in other types (i.e., if the occurrence of word i appearing in type w is more than that in the other fault types, word i is more important to determine whether the log belongs to type w).

Therefore, we can conclude that the probability coefficient $a_{i,w}$ is calculated as the product between the frequency of the words in the fault type and the importance in the entire template, namely in (6).

$$a_{i,w} = P(i,w) \times K(i,w). \tag{6}$$

Importing (3) and (4) can obtain (7).

$$a_{i,w} = \frac{count(i,w)}{\sum_{j=1}^{m} count(j,w)} \left[-\log\left(1 - \frac{count(i,w)}{sum(i)}\right) + 1 \right]. \tag{7}$$

The following describes the learning process by using the FK-matrix. After obtaining the results of the manual classification, according to (7), we will calculate matrix $\boldsymbol{A}$ by column. Next, we will describe how to solve the w-th column as an example.

The program will count the total number of words in type w from the entire library template. The amount is named $T(w)$, which is shown in (8).

$$T(w) = \sum_{j=1}^{m} count(j,w). \tag{8}$$

To facilitate the revised and updated FK-matrix, a new row will be added in the FK-matrix to store $T(w)$ in an attempt to reduce the amount of double counting.

For each word i in the fault type w, the program will calculate $sum(i)$. Similar to $T(w)$, in order to reduce the number of calculations, the additional space will be used to store $sum(i)$. Thus, there is an expanded FK-matrix, which will include an additional row and column to store statistical information. The final matrix $\boldsymbol{A}$ is shown in (9).

$$\boldsymbol{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & sum(1) \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & sum(2) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} & sum(m) \\ T(1) & T(2) & \cdots & T(n) & \end{pmatrix}. \tag{9}$$

According to (7), we should calculate $count(i,\ w)$ before calculating $a_{i,w}$. Because the value of $count(i,\ w)$ will be changed with the update of the FK-matrix, $count(i,\ w)$ will be saved in the FK-matrix, instead of $a_{i,w}$. UiLog will calculate ai, w until the occurrence probability of the word is obtained. Because the related variables have been restored in the FK-matrix, any additional overhead does not be added.

After obtaining the FK-matrix through the learning period, UiLog can classify each log through the FK-matrix. In the running period, administrators might modify or add new categories into different templates to cope with the changes in the system or software environment. At that moment, UiLog will modify the FK-matrix. The following will discuss how to edit the FK-matrix under three different conditions.

1) When the result of the classification needs to be modified, the system will scan all the words in the template and change some corresponding lines in the FK-matrix.

Assuming that word i exists in the template, it appears n times in this template, and the template belongs to the original type w. Now we want i to belong to catalog u. Algorithm 1 gives the pseudo-code for the modification of the FK-matrix.

Algorithm 1. FK-Matrix Modification

```
1:  function Modification()
2:      for i ← fault type needed to be modified
3:          count(i, w) = count(i, w) − n
4:          count(i, u) = count(i, u) + n
5:          T(w) = T(w) − n
6:          T(u) = T(u) + n
7:          Calculate a_{i,w}
8:          Calculate a_{i,u}
9:      end for
10:   end function
```

2) When we need to add a new template, the system will directly modify the FK-matrix. The specific method is similar to the establishment of the FK-matrix in the learning period. After confirming certain catalog of the new template, UiLog will modify the corresponding column and add the value of $sum(i)$ for every word that appears in the new template.

3) When we need to add a new template to the new classification, the system will add a new column to store the new classification (classification u). Then for each word (word i) in the template, UiLog will update $sum(i)$ and calculate $T(u)$. Finally, it will calculate $count(i, w)$, and the results will be stored in the FK-matrix.

3.2.5 Classifying Log Information

Through the fault keyword matrix, we can easily classify the fault log in the system. In the running period of the system, the fault log collection module will send fault logs to the fault log analysis module from various components. In this step, UiLog will scan every log message to compute the probability of different fault types of each log, according to the fault keyword matrix. Suppose the fault log L needs to be classified. In this instance, UiLog will compare each word in L with the FK-matrix. It will use an array (array s) to store the probability that every word in L belongs to a

different fault type. Algorithm 2 gives the pseudo-code for calculating this probability.

Algorithm 2. Log Classification

```
1:  function Classification()
2:      for w in {0, ..., number of fault types}
3:          for i ← every word in w
4:              if (i∈L ) then s[w] ← s[w] + a_{i,w}
5:          end for
6:          if (maxpossible < s[w]) then
7:              maxpossible ← s[w]
8:              f ← w
9:          end if
10:     end for
11:     Inform administrator L belongs to fault type f
12:     for w in {0, ..., number of fault types}
13:         if ((maxpossible−s[w]) < threshold t) then
14:             Inform L may belong to fault type w
15:     end for
16: end function
```

3.3 Fault Log Correlation Analysis

The fault log correlation analysis module diagnoses faults generated by different components of systems and software through logs. It will use the result of the log analysis to find connections between different fault logs.

In the cluster environment, there is a high correlation between components. When a component fails, it generates a lot of logs continuously, and those components that rely on this component also will produce fault logs. It is a great problem for an administrator to determine the root cause of the fault because the number of these fault logs is large. Thus, in the UiLog system, we put together all of the fault logs that are caused by the same error through a fault correlation analysis. These logs that are caused by the same error will generate a fault propagation tree ordered by the occurrence of times to help the administrator to determine the root cause of the fault.

Currently, the most effective method for fault correlation analysis is the time correlation analysis. This method is based on the hypothesis that the homology of faults occurs in closed time segments. Therefore, the specific method entails setting up a time window so that when a fault occurs, the algorithm will sweep every fault log in a chronological sequence. Then we believe that those fault logs that fall within the same time window are generated from the same root cause. These logs will be classified as a class (called a "tuple"). However, this method has two main problems as follows.

The first is the truncation error. A truncation, as shown in Fig.7(a), occurs when the time between two or more events caused by a single fault is longer than the classification time, thus causing the events to be split into multiple tuples. The second is collision error shown in Fig.7(c). A collision occurs when two independent faults occur close enough in time such that their events overlap and they are erroneously combined into a single tuple. Consequently, the advantage of the coalescence process is strictly dependent on the selected time window.

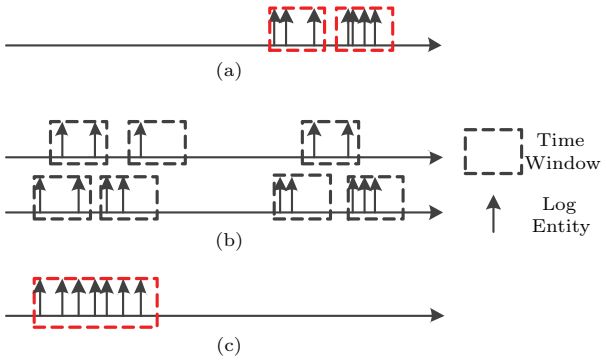


Fig.7. Errors of log correlation analysis. (a) Truncations. (b) Original. (c) Collisions.

In UiLog, we use the results of our previous fault log classification to improve the traditional fault correlation method based on time. We note that different fault types have different time ranges to affect the cluster systems. For example, a hardware fault has a relatively small range of time to affect the cluster system, while it has a huge impact on the system within a short time. On the other hand, the time range of the influence of the network fault is relatively wide. The associated component will produce a fault report after a long time. Thus, we can use different sizes of time windows to diagnose different fault types. The accuracy of our judgment of the homologous fault can be improved.

As a practical application of the fault log classification, we collect log information that is generated by all hosts and virtual machines in StrongCloud within 10 months and analyze the faults that occurred in this time. We first use the traditional fault correlation method based on time to diagnose the faults. This method uses a uniform window size. Then we manually analyze each log in the tuple to determine different window sizes for every fault type. UiLog provides an interface for displaying the fault type of each log in time sequence. Logs can be added or deleted from the

fault group by human. The program will automatically record each operation, and dynamically adjust the size of the time window. Table 4 gives part of the results for the time window.

Table 4. Example of Time Windows

Category	Description	Time Window (s)
Disk	Errors about SCSI disk or damaged sectors	91
Device (DEV)	Errors related to peripherals and PCI cards	93
Input/Output (I/O)	Errors about drive or communication	60
Memory (MEN)	Memory-related errors	68
Network (NET)	Communication issues	312
Processor (PRO)	Processor exceptions, machine check issues	71
Authentication (Auth)	Authentication errors	85
Web	Web engine errors	107

Similarly, UiLog still uses the fault keyword matrix. We add a new line in the matrix to store every time window for different fault types.

In the specific process, the administrator will first point out which log needs a diagnosis. UiLog will query the fault keyword matrix to find the appropriate time window after confirming the fault type of the log. Then UiLog will use this time window to diagnose the fault.

4 Performance Evaluations

In this section, we test the main functions of the UiLog system and evaluate the effects, including fault log analysis and fault correlation analysis in different ways. This section is separated into two parts. The first part will show the results of the log analysis module. The second part will evaluate the results of the fault correlation analysis. Our test is based on the StrongCloud disaster recovery cloud platform, which has been deployed for one year. The evaluations show the importance of log management. Because of the diversity of the host system, the contents and format of logs generated by different hosts have the great difference. As the universality of log processing, this paper has not concentrated on the process of the specific log. The log collected from StrongCloud is a sample to prove our system.

4.1 Log Analysis

For the log analysis module, the most important thing is to test the efficiency of log classification. In this test, UiLog classifies all of the fault logs collected by the log collection module. The sample data is derived from StrongCloud within one year. The details are shown in Table 5. The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e., insertions, deletions or substitutions) required to change one word into the other.

Table 5. Environment

System	Beginning	Ending	Days	Size	Messages
CentOS	2012-12	2013-12	386	1.7 GB	14 450 302

We first use the data in January and February to train UiLog to obtain the basic fault keyword matrix. The test was launched on the data from March 2013.

To classify logs, we use the trigger mechanism and the user defined function (UDF) in the database for analysis log timely. Whenever there is a log from the log collection module, the trigger is activated, and the UDF function will call upon the log analysis module. The log analysis module will classify the fault log according to the described steps of the running period by using the fault keyword matrix. If the type of log cannot be determined, this log will be saved in an unprocessed database, and UiLog will inform the administrator to manually classify this log. After dealing with this unsorted log artificially, UiLog will automatically learn from these new results and modify the fault keyword matrix to improve classification accuracy.

The bar chart (Fig.8) shows the classification results for the one-year fault log of StrongCloud.

We divide the fault log into 12 different types. Fig.8 gives the ratio of each type. From the figure, we can conclude that the main function of the StrongCloud platform is network-related. The NET errors and the WEB errors equal 43% of all faults. In addition, we can find that the platform has been subject to an external attack, as network authentication faults occupy 15% of all faults. The port SSL services or disable services are not commonly used.

From Fig.8, we can demonstrate that through the fault log classification analysis, UiLog can help the administrator to manage computer systems properly, finding problems and bottlenecks in the system to compensate for the deficiencies in the system.

In addition, we evaluate our log analysis method in different lengths of the logs and different types of faults.

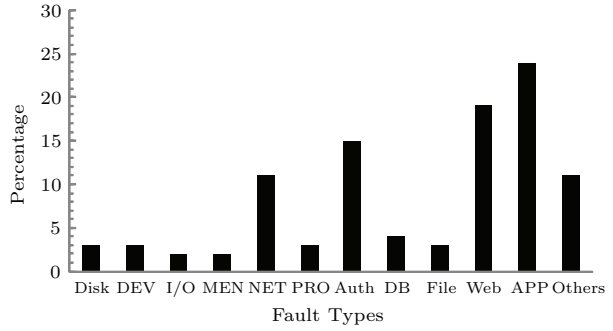


Fig.8. Types of logs.

When UiLog calculates the probability of each log belonging to a different catalog, the length of the log is an important factor, as our algorithm will accumulate the probability of every word. Therefore, we test for precision in different lengths of the logs so that our method can work for the logs of different lengths. Fig.9 shows the precision in the logs of different lengths. We can find that the length actually influences the precision. For a shorter log, our method reveals a good performance. We receive 98% precision when the length is between 100 and 125 words. When the length is more than 200 words, the precision will be lower than 88%. That means that our method also maintains a higher precision.

Fig.10 illustrates the precision of log classification for different fault types. We see that the Web errors are the easiest faults that we can identify. This seems to occur when the amount of Web fault logs is the largest. Therefore, we have more samples to distinguish this fault type from others.

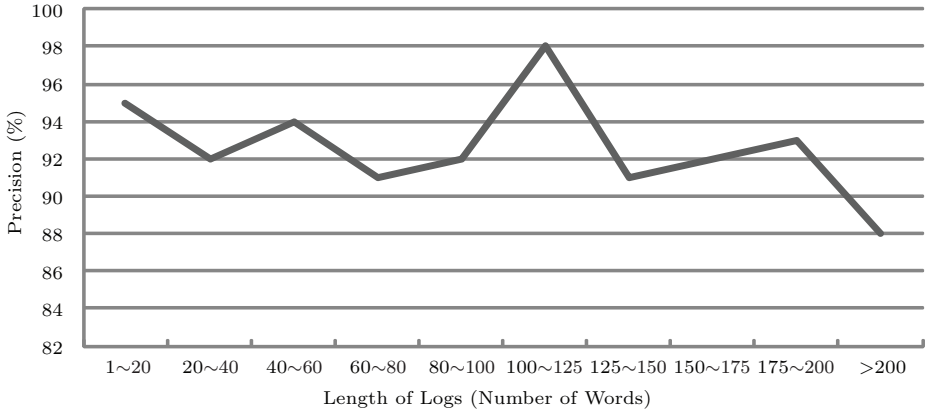


Fig.9. Precision in different lengths of logs.

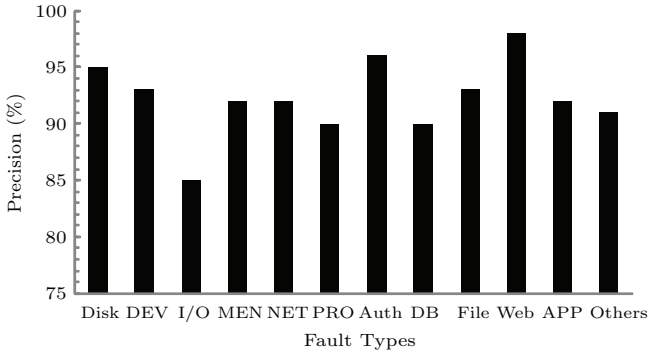


Fig.10. Precision in different fault types.

4.2 Learning Efficiency

The learning efficiency of artificial classification is a very important indicator for evaluating the new fault classification method of UiLog. We use two months of fault logs as a sample for this learning period. When the system is running, we will classify those non-classification logs manually at the end of every month. UiLog will re-learn these new classification results to improve the fault key matrix for better automatic classification results.

Fig.11 illustrates the number of classified fault clusters and unclassified fault clusters from March to October. It shows that there are only 568 fault log types that cannot be judged in March after the learning step compared with 2 350 defined types. Then UiLog studies the results of the new type rules at the end of March. The number of unclassified fault logs dropped significantly, reaching 208 types in April. After that, the apparent decreasing tendency can be seen from March to June, reaching the lowest point (48 types) in June, which is due to the re-learning that occurred at the end of every month.

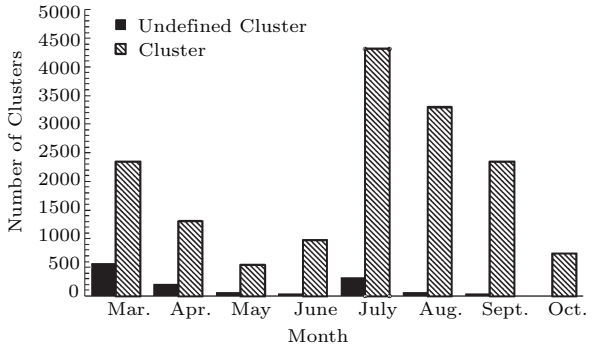


Fig.11. Number of clusters from Mar. to Oct.

However, Fig.11 also presents that the percentage of unclassified fault logs experienced growth in July. According to the previous analysis, this is due to the new application or software developed in July. This change generated a lot of new fault logs and led to 324 new fault types. After finishing the re-learning step at the end of July, the percentage of unclassified logs dropped dramatically, and the system gradually stabilized. There are only three undefined log types.

Through the above experiments and analysis, we can determine that our fault classification method is effective. It can be sufficiently carried out using the results of the automatic learning to automatically determine the type of a fault log.

4.3 Fault Log Correlation Analysis

The main function of the fault correlation analysis module is to make fault diagnosis, which gathers fault logs generated by the homologous failure to get a fault propagation tree.

In this subsection, we test our UiLog fault correlation analysis module and compare our method with three other renowned methods: the fixed method, the Tscao method[27], and the SC07 method[28]. To test the performance in different environments, we deploy our test system to 2, 4, 6, 8 and 10 nodes.

First, we evaluate the percentage of truncation errors that appeared in different nodes. Fig.12 illustrates that all algorithms have good performance, which is lower than 15% in the case of having fewer nodes, except for the fixed method. With the increased number of nodes, truncation errors also increase, reaching approximately 30%. However, our approach maintains a low error rate. Even when the number of nodes is 10, the truncation error rate of UiLog is lower than 20%, while this rate is 36% for Tscao and 42% for SC07. It is relatively stable with an increasing number of nodes.

In addition, it can be observed from the figure that the tendency of the fixed method is different from those of the others. This method is more suitable for a multi-node situation. As the number of nodes increases, the error rate decreases simultaneously.

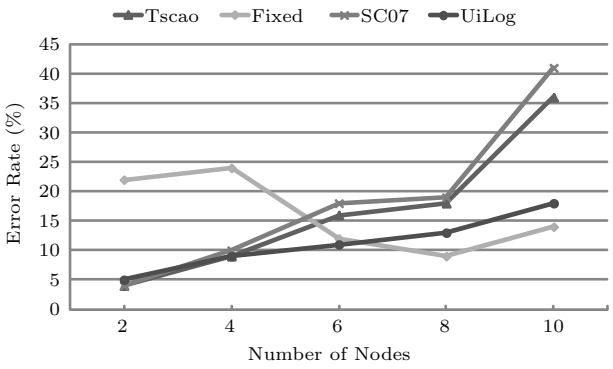


Fig.12. Comparasion of truncation errors.

Second, Fig.13 also shows the percentage of the collision errors that appeared in different nodes. It can be seen that the trends of the four methods are similar. At the beginning, the error rate is approximately 5% for every method. In addition, this percentage increases as the number of nodes increases. It is worth noting that the collision error rate of our method decreased to about 20% when 10 nodes were present. It helps the overall error rate be maintained at lower than 25%, achieving an optimum result.

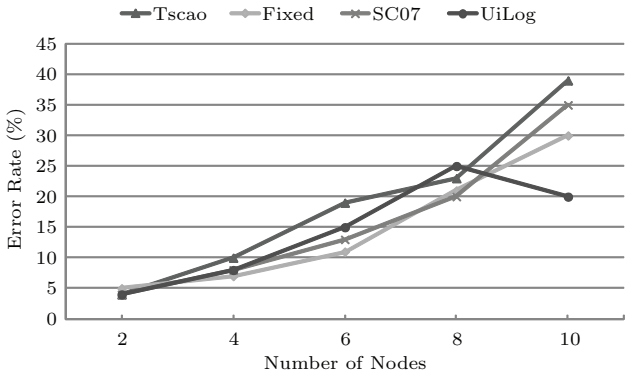


Fig.13. Comparasion of collision errors.

From the evaluations we can conclude that our improved method for fault correlation analysis is effective in reducing the probability of truncation errors and collision errors.

5 Conclusions

With the development of cloud computing, the architectures of systems and software are more complicated than what they were in the past. The communication between different layers of hardware and software is becoming more frequent, which enhances the difficulty of fault diagnosis based on system logs. This paper presented an integrated fault log analysis platform, the UiLog system, which helps administrators to manage logs generated by all of the components of a system, monitors the running of the system, and diagnoses faults.

To effectively analyze the logs, we proposed a new method to classify logs into different catalogs according to different fault types. We used the fault keyword matrix to accelerate the speed of classification and to efficiently learn the results of classification. In addition, we improved the fault correlation analysis. We used the results of fault classification to fix the time correlation window, allowing us to reduce the truncation and collision error rates.

In the future, we will optimize the procedure of log processing and apply some measures to automatically determine the time window.

References

[1] Zawoad S, Dutta A K, Hasan R. SecLaaS: Secure logging-as-a-service for cloud forensics. In *Proc. the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, May 2013, pp.219-230.

[2] Rao X, Wang H, Shi D *et al*. Identifying faults in large-scale distributed systems by filtering noisy error logs. In *Proc. the IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, June 2011, pp.140-145.

[3] Yuan D, Mai H, Xiong W *et al*. SherLog: Error diagnosis by connecting clues from run-time logs. *ACM SIGARCH Computer Architecture News*, 2010, 38(1): 143-154.

[4] Fu Q, Lou J G, Wang Y *et al*. Execution anomaly detection in distributed systems through unstructured log analysis. In *Proc. the 9th IEEE International Conference on Data Mining*, December 2009, pp.149-158.

[5] Xu W, Huang L, Fox A *et al*. Detecting large-scale system problems by mining console logs. In *Proc. the 22nd ACM Symposium on Operating Systems Principles*, October 2009, pp.117-132.

[6] Hansen J P, Siewiorek D P. Models for time coalescence in event logs. In *Proc. the 22nd IEEE International Symposium on Fault-Tolerant Computing*, July 1992, pp.221-227.

[7] Mi H B, Wang H M, Zhou Y F *et al*. Localizing root causes of performance anomalies in cloud computing systems by analyzing request trace logs. *Science China* (*Information Sciences*), 2012, 55(12): 2757-2773.

[8] Prewett J E, James E. Listening to your cluster with LoGS. In *Proc. the 5th LCI International Conference on Linux Clusters: The HPC Revolution*, May 2004.

[9] Jain S, Singh I, Chandra A *et al*. Extracting the textual and temporal structure of supercomputing logs. In *Proc. the 16th IEEE International Conference on High Performance Computing*, December 2009, pp.254-263.

[10] Takada T, Koike H. Tudumi: Information visualization system for monitoring and auditing computer logs. In *Proc. the International Conference on Information Visualization*, July 2002, pp.570-576.

[11] Vaarandi R. A data clustering algorithm for mining patterns from event logs. In *Proc. the 3rd IEEE Workshop on IP Operations and Management*, October 2003, pp.119-126.

[12] Bellec J H, Kechadi T M. Cufres: Clustering using fuzzy representative events selection for the fault recognition problem in telecommunication networks. In *Proc. the 1st ACM Ph.D. Workshop on Information and Knowledge Management*, November 2007, pp.55-62.

[13] Ganapathi A, Patterson D. Crash data collection: A windows case study. In *Proc. the IEEE/IFIP International Conference on Dependable Systems and Networks*, June 28-July 1, 2005, pp.280-285.

[14] Pecchia A, Cotroneo D, Kalbarczyk Z *et al*. Improving log-based field failure data analysis of multi-node computing systems. In *Proc. the IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2011, pp.97-108.

[15] Stearley J, Oliner A J. Bad words: Finding faults in Spirit's syslogs. In *Proc. the IEEE International Symposium on Cluster Computing and the Grid*, May 2008, pp.765-770.

[16] Stearley J. Towards informatic analysis of Syslogs. In *Proc. the IEEE International Conference on Cluster Computing*, September 2004, pp.309-318.

[17] Xu W, Huang L, Fox A *et al*. Mining console logs for large-scale system problem detection. In *Proc. the IEEE Conference on Tackling Computer Systems Problems with Machine Learning Techniques*, December 2008.

[18] Salfner F, Tschirpke S. Error log processing for accurate failure prediction. In *Proc. the 1st USENIX Workshop on Analysis of System Logs*, December 2008, p.4.

[19] Wen X, Zhang X, Zhu Y. Design of fault detection observer based on Hyper Basis Function. *Tsinghua Science and Technology*, 2015, 20(2): 200-204.

[20] Park J, Yoo G, Lee E. Proactive self-healing system based on multi-agent technologies. In *Proc. the ACIS International Conference on Software Engineering Research, Management and Applications*, August 2005, pp.256-263.

[21] Li T, Liang F, Ma S *et al*. An integrated framework on mining logs files for computing system management. In *Proc. the 11th ACM International Conference on Knowledge Discovery in Data Mining*, August 2005, pp.776-781.

[22] Vaarandi R. A breadth-first algorithm for mining frequent patterns from event logs. In *Proc. Intelligence in Communication Systems*, November 2004, pp.293-308.

[23] Oliner A, Stearley J. What supercomputers say: A study of five system logs. In *Proc. the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2007, pp.575-584.

[24] Simache C, Kaâniche M, Saïdane A. Event log based dependability analysis of Windows NT and 2K systems. In *Proc. the Pacific Rim International Symposium on Dependable Computing*, December 2002, pp.311-315.

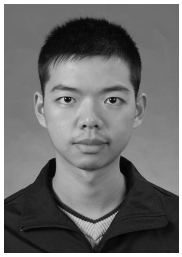
[25] Ester M, Kriegel H P, Sander J *et al*. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. the 2nd ACM International Conference on Knowledge Discovery and Data Mining*, August 1996, pp.226-231.

[26] Levenshtein V I. Binary codes capable of correcting deletions, insertions and reversals. *Journal of Soviet Physics Doklady*, 1966, 10: 707-711.

[27] Tsao M M, Siewiorek D P. Trend analysis on system error files. In *Proc. the IEEE International Symposium on Fault-Tolerant Computing*, June 1983, pp.116-119.

[28] Fu S, Xu C Z. Exploring event correlation for failure prediction in coalitions of clusters. In *Proc. the ACM/IEEE Conference on High Performance Networking and Computing*, November 2007.

De-Qing Zou received his Ph.D. degree in computer architecture from the Huazhong University of Science and Technology (HUST), Wuhan, in 2004. He is a professor of computer science at HUST. His main research interests include system security, trusted computing, virtualization and cloud security. He has been the leader of one National High Technology Research and Development 863 Program of China and three NSFC (National Natural Science Foundation of China) projects, and the core member of several important national projects, such as projects of National Basic Research 973 Program of China. He has applied almost 20 patents, published two books (one is entitled "Xen Virtualization Technologies" and the other is entitled "Trusted Computing Technologies and Principles") and more than 50 high-quality papers, including papers published by IEEE Transactions on Dependable and Secure Computing, IEEE Symposium on Reliable Distributed Systems and so on. He has always served as a reviewer for several prestigious journals, such as IEEE TPDS, IEEE TOC, IEEE TDSC, IEEE TCC and so on. He is on the editorial boards of four international journals, and has served as a PC chair/PC member of more than 40 international conferences.



Hao Qin received his B.S. degree in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, in 2011, where he is currently a Master student in information security. His research interests include fault diagnosis and system security.

Hai Jin received his Ph.D. degree in computer engineering from the Huazhong University of Science and Technology (HUST), Wuhan, in 1994. He is currently a Cheung Kung Scholars Chair Professor of computer science and engineering at HUST. He worked at the University of Hong Kong between 1998 and 2000 and as a visiting scholar at the University of Southern California between 1999 and 2000. He is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientist of the National Basic Research 973 Program Project of Virtualization Technology of Computing System. He is a member of Grid Forum Steering group. He has coauthored 15 books and published more than 400 research papers. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security. He is the steering committee chair of the International Conference on Grid and Pervasive Computing, the AsiaPacific Services Computing Conference, the International Conference on Frontier of Computer Science and Technology, and the Annual ChinaGrid Conference. He is a member of the steering committee of the IEEE/ACM International Symposium on Cluster Computing and the Grid, the IFIP International Conference on Network and Parallel Computing, the International Conference on Grid and Cooperative Computing, the International Conference on Autonomic and Trusted Computing, and the International Conference on Ubiquitous Intelligence and Computing. He was awarded the Excellent Youth Award from the National Natural Science Foundation of China in 2001. In 1996, he was awarded a German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz, Germany. He is a fellow of CCF, a senior member of IEEE, and a member of ACM.