Improving Log-Based Fault Diagnosis by Log Classification

Deqing Zou, Hao Qin, Hai Jin, Weizhong Qiang, Zongfen Han, Xueguang Chen

Improving Log-based Fault Diagnosis by Log Classification

Deqing Zou, Hao Qin, Hai Jin, Weizhong Qiang, Zongfen Han, Xueguang Chen

Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
deqingzou@hust.edu.cn

Abstract. In modern computer systems, system event logs have always been the primary source for checking the system status. As computer systems become more complex, such as cloud computing systems, the interaction among software and hardware is increasingly frequently. These components will generate enormous log information, including running reports and fault information. The massive data is a great challenge for analysis with manual method. In this paper, we implement a log management and analysis system, 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 log management and analysis system to cloud computing environment for evaluation. The results show that our system can classify fault logs effectively and automatically. By using the proposed system, administrators can easily detect the root cause of faults.

1 Introduction

With the widespread usage of cloud computing, computer systems are becoming increasingly complex and the components within the entire system also become diverse. Once some key parts failed, the whole system would be seriously implicated due to the frequent interactions and high coupling. Therefore, an effective fault detection and analysis method can help system administrators to locate the fault and identify the cause, which plays an essential role in large systems management.

System and software logs are important sources for diagnosing the 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. Typically, we have to manually extract the useful information from vast amounts of data, which would seriously delay the response time of fault recovery. Therefore, an unified management system for fault log analysis is required, which can automatically identify the fault type and analyze the cause of the faults. It will provide a great help for system management.

A lot of studies have been proposed on log-based fault analysis, mainly falling in the following directions. The first is log collection and analysis, which investigate how to effectively and efficiently gather log information [1]. This information will be used to get profile information for analyzing the system situation [2] and extract the feature [3]. The second part is fault location, which aims at determining the control flow of software through logs [4] and uses the source code [5] to locate the position of the faults occurred. The researches mentioned above use the log fault analysis in different scenarios. But few of them focus on the system administrator's perspective to design an integrated fault analysis system. The third part is fault correlation analysis. In the multi-node environment, the fault propagation is a critical problem for fault diagnosis. The area focuses on using log information to determine the connection between different faults. Researches use time and spatial correlation to find some connection, few of them consider the meaning of logs.

An integrated fault log analysis system will assist administrators to perform fault analysis, improving the administrator’s ability to respond to the system fault and reduce the time consuming of fault processing. We implement a new fault classification method to assist manager to understand computer system and use fault correlation analysis to locate the root-cause of fault. In this paper, we propose an integrated fault log analysis platform (*UiLog*) to collect and manage various components logs, storing, filtering, and analyzing logs for administrators to quickly locate fault and analyze the cause of faults.

This platform consists of three components: 1) Fault log collection module is mainly used to collect log data from various components. 2) Fault log analysis module classifies log into the identifies fault type in real time. 3) Fault log correlation analysis module collects fault log caused by same root-fault as a tuple and tries to find the root cause of such faults. *UiLog* have deployed in a practical cloud environment, helping administrator to troubleshoot and find the root cause of fault. Our main contributions are:

We propose a novel classification method for fault logs, using fault keyword matrix to improve the accuracy. It reduces the time of determining the fault type and the workload of manual processing. Moreover, this method can be more convenient to add a new fault type without recalculated.

We improve the existing log correlation analysis. It combines the result of fault classification and time windows correlation analysis. Our method uses the fault type of 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 fault’s root cause.

We illustrate a comprehensive log management system, which can help administrators to quickly grasp the operation status of the system and save troubleshooting time.

The remaining of this paper is organized as follows. Section 2 discusses background and related work. Section 3 outlines the structure of *UiLog* system and describes the implementation of the system. Section 4 describes the evaluation of our systemwhereas section 5 presents conclusions and future work.

2 Related Work

The aim of log information is to extract useful information from fault logs. The important techniques of previous work are mainly relied on regular expressions [5]. However, the rules of regular expressions require different knowledge from laborious and expert [6]. In addition, the deployment of new application and upgrading of system will change these templates of log frequency. It is difficult for designing these regular expressions [7].

Various studies are looking for how to understand the mean of logs, but it may be useful for detecting faults in cloud computing system [8]. Many interesting features are displayed by these studies. For instance, Stearley [9] has a new discovery that only through words cannot detect the fault type from logs. The position of each word is a powerful indicator to distinguish different messages.

Researchers have also looked at other way except system logs to diagnose system, such as application console logs [10]. However, this technique is limited to application specific anomalies and requires source code [11].

In addition, several techniques and algorithms for automatic log classification have been developed. [12] attempts to classify different raw logs into a set of categories. Moreover, in [13], the authors try to use the modified naive Bayesian model and *Hidden Markov Models* (HMM) to classify event logs based on the IBM CBE (*Common Base Event*) format. On the other hand, SLCT [14] and Loghound [15] are designed specifically to discover the format of logs and classify row log automatically. They use two similar algorithms, which are useful to extract the template from logs.

Similar to correlation analysis, time and spatial correlation techniques have been applied to a variety of large scale computing systems [16]. The current trend of this study is to use tuple with a fixed value for time window, such as 5 minutes [17].

Content-based correlation is also a hot topic. For example, [17] applies the lift data by mining operator to find frequent event patterns starting from log contents and try to isolate accidental patterns.

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 track logs for statistics.

Through the fault classification, *UiLog* learns the classification rules from training set of artificial classification. After that, the system determines the fault type in real time according the rule library. In addition, the fault correlation analysis can be deployed when system administrators need to diagnosis fault. Considering the propagation of the fault, *UiLog* can mining the association between faults generated by the same root-cause and collect these fault logs into the same cluster.

To implement the process, we apply three modules to represent the *UiLog*: fault log collection, fault log analysis, and 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 in all components. The analysis node is responsible for fault analysis and diagnose. In the analysis node, the fault log collection module stores all logs in log information database for analysing.

The process of fault log analysis module is shown in Fig. 1. The fault log collection module will be deployed respectively into Target Node for log collection and Analysis Node for storage. This module will gather software log and system log from Target Node and store log information into Log Information database. In the Analysis Node, the Fault log analysis module will extract log structure from fault log collection module. At first, these structures will be classified into different fault type by administrator. The Fault log analysis module will learn classification rules from administrator and store it into fault template database. After that, the fault log analysis module will automatic classify log for log diagnose.

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

3.2 Fault Log Analysis

The fault log analysis module aims at classifying log into different log types. We adopt an example to illustrate the process: when an administrator analyzes the logs (Table 1) the first step is to determine whether the logs are fault record or not. The most commonly used method is finding keyword. The second step is the log analysis. The system administrator will weed out the details of log information to determine the cause of the fault. The first row in Table 1 shows that "Read socket failed", which indicates a socket problem when reading. The third step is the log classification. It is easily to determine that the first log belongs to the network fault and the subcategory is the remote network connection fault.

Through the above analysis, we can conclude that if we can deal with the semantic analysis of keywords and determine the fault type in advance, a large load of work can be automatically processed by the log analysis and classification.

As shown in Fig. 2, *UiLog* log classification method is divided into five steps. 1) Log Pretreatment. 2) Extracting Invariants. 3) Filtering Template Information. 4) Obtaining Fault Keyword Matrix. 5) Classifying Log Information. These steps will gradually extract log information for fault classification and remove irrelevant content.

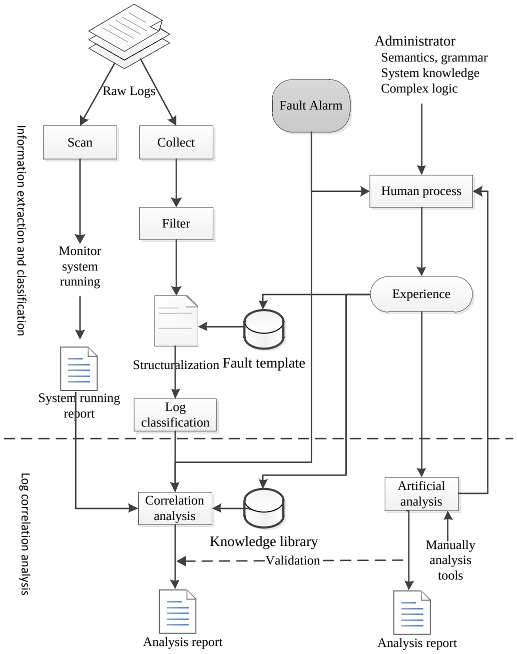

Fig. 1. Workflow for *log analysis* and fault diagnosis

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

The input of the whole algorithm is the log data flow obtained by 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 fault category of every log (output at Step 5). The following sub-sections describe each step of the algorithm in more detail.

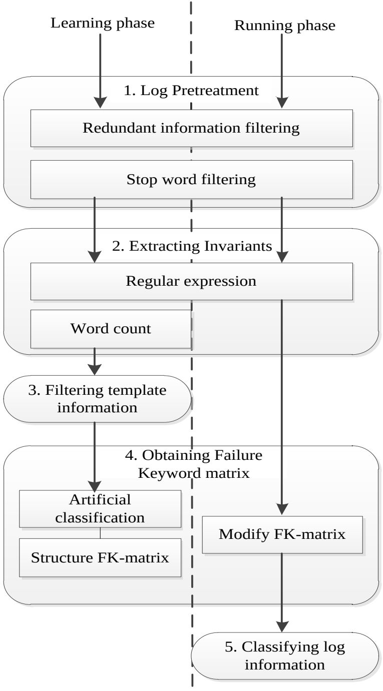
Learning phase
Running phase
1. Log Pretreatment
Redundant information filtering
Stop word filtering
2. Extracting Invariants
Regular expression
Word count
3. Filtering template information
4. Obtaining Failure Keyword matrix
Artificial classification
Structure FK-matrix
Modify FK-matrix
5. Classifying log information

Fig. 2. Number of key words appeared in the fault logs

1) Log Preprocessing

Log preprocessing contains two parts. The first part is to filter repeated logs generated by the system. Many software faults are insufficient to cause the collapse in computer system and the component will persistent send fault message. We can choose an appropriate threshold to filter the duplicate logs.

The second part is the log filter of meaningless words. We use *English Stop Word Table* to filter meaningless words. In accessory, considering the special of logic, we also filter out the most of adjectives and adverbs.

2) Extracting Invariants

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

3) Filtering Template Information

Before classification, we use automatic classification approach DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) [18] to classify the log for further reducing the manually determining space.

For a classification algorithm, based on *Levenshten Distance* (LD), we define our *Log Levenshten Distance* (LLD) to measure the distance between a log and a cluster. The distance between log A and log B is described in (1).

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

wherein *LD(A, B)* is the original Levenshten distance, *length()* indicates the length of the log.

4) Obtaining Fault Keyword Matrix

When the system is in the learning stage, this step will learn the result of artificial classification. While in the running stage, this step will modify the classification's rules. Through the step of filter template information, the remaining numbers of raw logs have less than original.

For learning stage, the first requires administrator to classify logs manually. We pre-define a fault catalogue in the cloud environment. Through the last step, the classification has already marked a label to each fault cluster. Now the administrator will first modify the automatically label, then adjust the result of the classification. As shown in Table 2, after indicated the category, the result will be stored in the way of "Content: Mark". Label is a number that represents the fault type.

Table 2. Fault types

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	Authenticate
Kernel reported iSCSI connection |NUM| error * state	6	Drive
* received packet with * address * source	5	Network

Next, *UiLog* needs to learn the results of the artificial classification. Here we propose the *Fault Keyword Matrix* (FK-Matrix) for saving the learning result of artificial classification.

The FK-Matrix (matrix A) is a two-dimensional matrix constructed by the probability of each word appeared in each fault type in the template. It is an $m\times n$ matrix. M represents the different number of words in all the sample log, while n represents the number of fault types, a_{ij} denotes the probability of the i-th word belong to the j-th catalogue. (Note: $a_{i,j}$ is only a relative probability factor, not the true probability.)

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

The 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 type. As the value indicates the frequency of each word in a particular type of fault, we consider the probability of the word i in the fault type w as 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, *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, but it ignores the same word between different fault types in the log template. For example, if the word i only appears in the type w, then we believe that a log is very likely to belong to the fault type w as long as it contains the word i, even if how many time the word i appears in the type w. Therefore, we can amend the formula by adding a scale factor shown as (4).

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

The *sum(i)* represents the number of times that the word i appears in all the fault type 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 the word i in the type w and it is in inverse proportion to the frequency of word i occurs in other types, i.e. if the occurrence that word i appears in the type w is more than in the other fault types, the word i is more important to determine whether the log is belong to type w.

Thereby, we can conclude that the probability coefficient $a_{i,w}$ is calculated as the product between the frequency of 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 the equations (3) and (4) can obtain equation (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 manual classification, according to the formula, we can calculate the matrix 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 as *T(w)*, which is shown in (8).

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

To facilitate revised and updated the FK-matrix, a new row will be added in the FK-matrix to store *T(w)* for reducing the number of double counting.

For each word *i* in the fault type *w*, the program will calculate *sum(i)*. Similar to the *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 adding the additional row and column for storing statistical information. The final matrix *A* is shown in (9).

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

According to equation (7), we should calculate *count(i, w)* before calculating $a_{i,w}$. Considering the value of *count(i, w)* will be changed with updating of FK-matrix, *count(i, w)* will be saved in the FX-matrix instead of $a_{i,w}$ in practice. *UiLog* will calculate $a_{i,w}$ until the occurrence probability of word is needed. Due to related variables has restored in the FK-matrix, it does not add any additional overhead.

After obtaining the FK-matrix through learning period, *UiLog* can classify log through FK-matrix. In the running period, administrators might modify or add new categories to different template for coping changes of system or software environment. At that moment, *UiLog* will modify FK-matrix.

5) Classifying Log Information

Through the fault keyword matrix, we can easily classify the fault log on 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 log, according the Fault Keyword matrix. Suppose the fault log *L* need to be classified, *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 different fault types. The Algorithm 1 gives the pseudo-code for calculating the probability.

Algorithm 1. Log classification

```
function Classification()
    for $w$ in counts of fault types
        for $i \leftarrow$ every word in $w$
            if( $i \in L$ ) then $s[w] \leftarrow s[w] + A[w][i]$
        end for
        if($maxpossible < s[w]$) then
            $maxpossible \leftarrow s[w]$
            $f \leftarrow w$
        end if
    end for
    Inform administrator $L$ belongs to fault type $f$
    for $w$ in counts of fault types
        if( ( $maxpossible - s[w]$ ) < threshold $t$ ) then
            Inform $L$ may belongs to fault type $w$
    end for
end function
```

3.3 Fault Log Correlation Analysis

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

In the *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 type has different time range to affect system. For example, a hardware fault has a relatively small range of time to affect the system, but it has a huge impact on the system within a short time. On the other hand, the time range of the influence of network fault is relatively wide. The associated component will produce a fault report after a long time. Thus, we can use different size of time windows to diagnose different fault types. It can improve the accuracy of judgment homologous fault.

As a practical application of the fault log classification, we collect log information generated by all hosts and virtual machines in *StrongCloud* within 10 months and analyze the fault happened in this time. Here we first use traditional fault correlation method based on time to diagnosis the fault. It uses the uniform window size. Then we manually analyze each log in the tuple to determine the different window size for every fault type.

In the specific process, the administrator will point out which log they want to diagnose. *UiLog* will query the *fault keyword matrix* for finding the appropriate time window after confirming the fault type of log. Then *UiLog* will use this time window to diagnose the fault.

4 Performance Evaluations

In this section, we test the main functions of *UiLog* system and evaluate the effect including fault log analysis. We construct a fault-tolerance testbed, *StrongCloud* [19], which is made up of five Inspur's NF5240M3 servers, each with 24 Intel Xeon E5-2420 1.90 GHz processors, 32 GB of RAM and four Gigabit Ethernet ports. Each host has a Domain0 with CentOS release 6.3 of kernel 3.7.1-xen.x86_64, and the hypervisor is Xen 4.2.1. *UiLog* is a sub-system of *StrongCloud*. The evaluations show the importance of log management.

4.1 Log Analysis

For log analysis module, the most important is to test the efficiency of log classification. In this test, *UiLog* classifies all the fault logs collected by log collection module. The sample data is derived from *StrongCloud* within one year. The details are shown in Table 3.

Table 3. Experimental environment

System	Beginning	Ending	Days	Size	Messages
CentOS	2012-12	2013-12	386	1.7GB	14450302

We first use the data of January and February to train *UiLog* to obtain the basic Fault Keyword matrix. The test is launched on the data from March 2013.

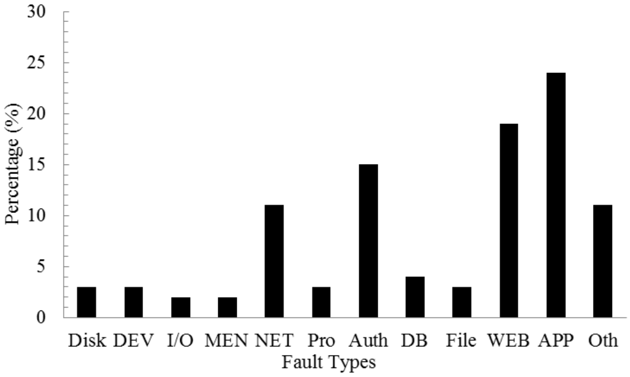

Fig. 3. Types of logs

To classify logs, we use trigger mechanism and *User Defined Function* (UDF) in database for analysis log timely. Whenever there is a log from the log collection module, the trigger is activated and UDF function will call log analysis module. The log analysis module will classify the fault log according to the described steps of running period by using the fault keyword matrix. If the type of log cannot be determined, this log will be saved in 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. 3) shows the classification results for the one year fault log of *StrongCloud*. We divide fault log into 12 different types. Fig. 3 gives the ratio of each type. From the figure we can conclude that the main function of *StrongCloud* platform is network-related. The net error and web error are sum up to 43% of all faults. In addition, we can find that the platform has been subject to external attack, network authentication faults occupy 15%. The port SSL services or disable services are not commonly used.

From Fig. 3, we can demonstrate that through the fault log classification analysis, *UiLog* can help administrator to manage computer systems, finding problems and bottlenecks in the systemto compensate the deficiencies in the system.

4.2 Learning Efficiency

The learning efficiency of artificial classification is a vary import indicator for evaluation the new fault classification method of *UiLog*. We use two months fault logs as a sample for 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 result.

Fig. 4 illustrates the number of classified fault logs and unclassified fault logs from March to October. It shows that there are only 568 fault log types cannot be judged in March after learning compared 2350 defined types. Then *UiLog* studies the results of new type rules by the end of March. The number of unclassified fault logs had dropped significantly, reaching 208 types on April. After that, the apparent decreasing tendency can be seen during March to June, reaching the lowest point (48 types) in June. That is because of re-learning by every end of month.

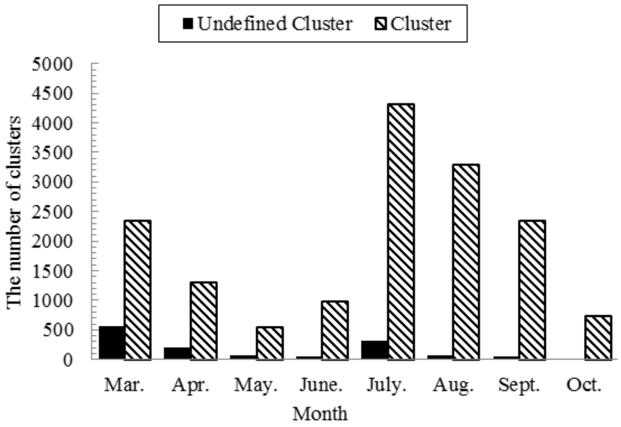

Fig. 4. Number of clusters

However, Fig. 4 also presents that the percentage of unclassified fault logs has a growth in July. According the previous analysis, it is due to the new application or software developed in July. This change generates a lot of new fault logs and leads to 324 new fault types. After finishing the re-learning step by the end of July, the percentage of unclassified logs has dropped dramatically and the system becomes gradually stabilizing. There are only 3 undefined log types.

Through the above experiments and analysis, 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 fault log.

5 Conclusions and Future Work

With the development of cloud computing, the architecture of system and software are more complicated than before. This paper presents an integrated fault log analysis platform *UiLog* system, helping administrators to manage the log generated by the all the components of system, monitoring the running of the system and diagnosing the fault.

To effectively analyze the logs, we propose a new method to classify log into different catalogs according to the different fault types. We use *Fault Keyword matrix* to accelerate the speed of classification. In addition, we improve the fault correlation analysis. We use the result of fault classification to fix the time correlation window to reduce the truncation error and collision error.

Acknowledgments. This work is supported by National 973 Fundamental Basic Research Program under grant No. 2014CB340600 and National Science Foundation of China under grant No. 61272072.

References

1. Zawoad, S., Dutta, A. K., Hasan, R.: SecLaaS: secure logging-as-a-service for cloud forensics. In Proceedings of the ACM Symposium on Information, Computer and Communications Security (2013) 219-230

2. Rao, X., Wang, H., Shi, D., Chen Z.: Identifying faults in large-scale distributed systems by filtering noisy error logs. In Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (2011) 140-145
3. Yuan, D., Mai, H., Xiong, W., Tan, L., Zhou, Y., Pasupathy, S.: SherLog: error diagnosis by connecting clues from run-time logs. Computer Architecture News. 38, doi: 10.1145/1735971.1736038 (2010)143-154
4. Fu, Q., Lou, J., Wang, Y., Li, J.: Execution anomaly detection in distributed systems through unstructured log analysis. In Proceedings of the IEEE International Conference on Data Mining (2009) 149-158
5. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Detecting large-scale system problems by mining console logs. In Proceedings of the ACM Symposium on Operating Systems Principles (2009) 117-132
6. James, E. P.: Listening to your cluster with LoGS. In Proceedings of the LCI International Conference on Linux Clusters: TheHPC Revolution (2004) 1-10
7. Jain, S., Singh, I., Chandra, A., Zhang, Z., Bronevetsky, G: Extracting the textual and temporal structure of supercomputing logs. In Proceedings of the IEEE International Conference on High Performance Computing (2009) 254-263
8. Stearley, J., Oliner, A. J.: Bad words: Finding faults in Spirit's syslogs. In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (2008) 765-770
9. Sandia, J. S., Stearley, J.: Towards informatic analysis of syslogs. In Proceedings of the IEEE International Conference on Cluster Computing (2004) 309-318
10. Xu, W., Huang L., Fox A., Patterson, D., Jordan M.: Mining Console Logs for Large-Scale System Problem Detection. In proceedings of the IEEE Conference on Tackling Computer Systems Problems with Machine Learning Techniques (2008) 4-14
11. Salfner, F., Tschirpke S.: Error Log Processing for Accurate Failure Prediction. In Proceedings of the USENIX Workshop on Analysis of System Logs (2008) 23-31
12. Park, J., Yoo, G., Lee, E.: Proactive self-healing system based on multi-agent technologies. In Proceedings of the ACIS International Conference on Software Engineering Research, Management and Applications (2005) 256-263
13. Li, T., Liang, F., Ma, S., Peng, W.: An integrated framework on mining logs files for computing system management. In Proceedings of the ACM International Conference on Knowledge Discovery in Data Mining (2005) 776-781
14. Vaarandi, R.: A data clustering algorithm for mining patterns from event logs. In Proceedings of the IEEE Workshop on IP Operations and Management (2003) 119-126
15. Vaarandi, R.: A breadth-first algorithm for mining frequent patterns from event logs. In Proceedings of Intelligence in Communication Systems (2004) 293-308
16. Oliner, A., Stearley, J.: What supercomputers say: A study of five system logs. In Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (2007) 575-584
17. Pecchia, A., Cotroneo, D., Kalbarczyky, Z., Iyer, R. K.: Improving log-based field failure data analysis of multi-node computing systems. In Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (2011) 97-108
18. Ester, M., Kriegel H., Sander J., Xu X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the ACM International Conference on Knowledge Discovery in Data Mining (1996) 226-231
19. StrongCloud. http://211.69.198.202:91