Log-Based OpenStack Fault Diagnosis by Machine Learning

Leyi Zhang¹, Lei Fan¹ and Naiwang Guo²

¹School of Cyber Security, Shanghai Jiao Tong University
²Center of Grid Technology, SGSEPRI, Shanghai, China
Email: {allan2900936, fanlei}@sjtu.edu.cn, guonw@sh.sgcc.com.cn

Abstract. With the rapid development of cloud computing technology, OpenStack is widely used. Since failures often occur on cloud platforms, how to detect OpenStack system failures becomes an important issue. This paper proposes an algorithm for fault diagnosis, which requires only the raw logs. The raw logs are first unified and stored in the database. Then a well designed time window is selected to extract features. The extracted features are then used to locate the fault time period in the logs. Through the analyzing of the log fragment, the components of OpenStack system which account for the failure are determined. And detailed reasons are further confirmed. Experimental results showed that our proposed algorithm performed well in detecting OpenStack failures.

1. Introduction

With the development of cloud computing, more and more companies, research institutions and individuals put related business on cloud platform. Most public and private clouds are based on OpenStack, which is an open source project developed and initiated by NASA and Rackspace [1]. OpenStack is a cloud operating system that controls large pools of compute, storage and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface [2].

The log is an important record carrier for the running status of the system. Through the analysis of the log, the operating status of the system can be determined, providing a basis for the continuous and stable operation of the system. The log can also be used to detect if the system has failed and is used for fault location and system recovery after a failure.

In this paper, we propose an unsupervised algorithm for fault diagnosis. This algorithm is based on the machine learning to make automatic analysis and mining of logs generated in OpenStack system without any labeled data. Firstly, our algorithm determines whether there is a fault from the massive logs. Then, this method finds the time when the fault occurred. After these steps, our method analyzes the logs in the fault period to locate fault and find the causes of the fault. The feature of this algorithm is that it requires only raw system logs and does not require labeled data. Therefore, this algorithm has a wide range of application scenarios, requires no additional data, does not rely on expert knowledge and can even diagnose new faults.

2. Related Work

For fault diagnosis, regular expressions are important techniques [3]. But it is difficult to make the rules of regular expressions because it needs expert knowledge [4].

In addition, several techniques and algorithms have been developed using classification. [5] Classifies different logs into different categories. [6] Classifies event logs using the modified naive Bayesian model and Hidden Markov Models.
[7] Designed specifically algorithm to extract the template from logs. It is used to classify row log automatically. [8] Matched a database of failure signatures with undiagnosed failure data to monitor data.

[9] Uses application console logs to detect fault. However, this technique requires source code [10].

3. Algorithm

Our algorithm does not require labeled data, but only the original logs, which can be used in many scenarios.

3.1. Log Preprocessing

Preprocess contains two parts. The first part is to filter duplicate logs. The logs of OpenStack sometimes repeat, that is, an identical log is printed twice. A small amount of duplicate logs does not interfere with the results, but a large number of duplicate logs will affect the results so we need to filter duplicate logs.

The second part is the processing of the format. Each line of log is divided into four parts, which are time stamp, log level, code module and log content. Spaces between each section are printed as separators. Due to the complexity of the OpenStack system, there are some differences in the log format for many modules of the same system. For example, the time stamp can be printed with two different formats according to whether there is Time zone or not. Meanwhile, the order of the four parts is also different. When we perform analysis, we must first deal with the problem of the log format and unify the different formats. For time stamps, we replace them with the same format uniformly. In addition, we use one order to uniform the logs.

3.2. Extracting Features

As mentioned earlier, the distribution of the number of logs is relatively stable for a stable OpenStack system. Based on this idea, we extract features using the number of logs.

Locating the system fault from the logs, the first step in our algorithm is to find the time when the fault occurred. We use time as the primary key and count the number of logs in the time. For example, if you set the interval to minutes, then each minute is used as a primary key.

For every primary keys, extract features for them. The primary key is used as the center of the time window to calculate the number of logs in the current time period as a feature. Select N units of time before and after the center of the time to form a time window of 2N+1 time length. After that, take the number of logs in different time as features, so that we get 2N+1 features in total.

The length of time here can be fixed or not fixed. For example, when the fixed length of time is one minute, we can take two minutes before and after the center point to form a time window of five time lengths.

In addition, we can also take 1 minute, 2 minutes and 3 minutes before the center point, and then take 1 minute, 2 minutes and 3 minutes after the center point to form a time window of 7 lengths.

According to the actual situation, we choose the parameter of time window.

- Time interval. In general, the duration of the system failure can be at least several minutes to several tens of minutes. Therefore, it is completely possible to detect failures when time interval is one minute. In a special case where the system failure only occurs for a few seconds, the time interval may have to be adjusted to one second.
- The length of the time window. The length of the time window is related to the edge of the fault. In general, the number of logs before the failure is normal, and there will be a marginal time before the failure occurs, that is, the number of logs will be in a rising phase. If the time window covers both the fault time and the normal time, the edge can be detected. Each time a fault occurs, some time windows can cover the edge, which is helpful for fault diagnosis. Therefore, the length of the time window should be set according to the actual situation. If you set the time window too long, it will increase the system's computational load, so it does not need to be set too long. From the previous operating experience, the edge will be 1-2 minutes, so set the time window to 5 minutes to detect the edge.
3.3. Detection of Failure Time Using Clustering
After extracting features, we get some data. The primary key is the time and the feature is a series of numbers that represent the number of logs in the time window. According to these features, we use the clustering algorithm to obtain the cluster of the corresponding time and find the fault time period according to the cluster.

3.3.1. K-means Clustering. Classification requires that each category be defined and maps each element to a category [11]. As for clustering, you can not know the category or even the number of categories, which is unsupervised learning. The k-means algorithm is the representative of clustering [12]. The algorithm is simple and the effect is very good. The algorithm is very intuitive:
   a) Take k elements randomly from data as the respective centers of k clusters.
   b) Calculate the degree of dissimilarity between the remaining elements to the center of k clusters, and assign these elements to the cluster with the lowest degree of dissimilarity.
   c) According to the clustering results, the centers of the k clusters are recalculated. The calculation method is to take the arithmetic mean of the respective dimensions of all the elements in the cluster.
   d) Cluster all elements again according to the new center.
   e) Repeat b), c), d) until the clustering result no longer changes.
   f) Output the result.

3.3.2. How to Set K in K-means. In the k-means algorithm, k represents the number of clusters and k needs to be set in advance. Silhouette Coefficient is one method to set k.
   The core indicator of the method is silhouette coefficient, as in equation (1).
   \[ S = \frac{b - a}{\max(a, b)} \] (1)

In equation (1), a means the average distance between one sample and other samples in the same cluster and b means the average distance between the sample and all samples in the nearest cluster.

Calculate the silhouette coefficient of all samples and then average them to obtain the average silhouette coefficient [13]. The k with the largest average silhouette coefficient is the optimal number of clusters.

3.3.3. How to set K in the situation. In the process of fault detection, K is generally set to 4 according to the effect. The first cluster is in normal operation and the second cluster is at the edge of fault. The third cluster is completely in the period of failure and the fourth cluster is caused by the initial start of the system or log loss.

3.4. Fault Location
After finding the fault time, we can take the corresponding logs for analysis according to the timestamp, mainly using the TF-IDF to extract keywords.

TFIDF (Term Frequency-Inverse Document Frequency) is a commonly used weighting technique for information retrieval and information mining [15]. TFIDF is a statistical method for assessing the importance of a word for one document set or one of the documents in a corpus. The importance of a word increases in proportion to the number of times it appears in the file, but at the same time it decreases inversely with the frequency of its appearance in the corpus.

Term frequency refers to the number of occurrences of a given word in the file. This number is usually normalized to prevent it from biasing towards long files.

However, it should be noted that some common words do not have much effect on the subject. On the contrary, some words that appear less frequently can express the subject of the article. Therefore, it is not appropriate to use TF alone. In all statistical articles, some words appear in only a few of them, then such words have a great effect on the subject of the article. IDF is designed to solve this problem.
The main idea of IDF is: if the number of documents containing the word is less, the IDF is larger, indicating that the word has a good ability to distinguish categories. The way to calculate the value of IDF is as in equation (2).

\[ IDF = \log \frac{|D|}{1 + |J|} \]  

(2)

In equation (2), D means total number of files in the corpus and J means the number of documents containing this word.

TFIDF is obtained by multiplying TF by IDF.

For the logs in the fault time, we calculate the values of TFIDF of all words. Then we sort words according to TFIDF to get the most valuable words. Through the keywords, we can locate the fault, including the components and causes of the fault. In addition, we can infer causes from the keywords.

4. Experiments

4.1. Fault Log Collection

Some of the logs are from actual faults and the other logs is obtained through fault injection. Some cases of faults are showed in table 1.

| Faults                        |
|-------------------------------|
| Case 1 Network failure causes VM creation to fail |
| Case 2 Insufficient memory causes VM creation to fail |
| Case 3 Dongle error           |
| Case 4 Network failure of Nodes |

4.2. Detection of Failure Time

We follow the previous algorithm to process the log format. After unified formatting, we create a time window and extract features. The interval is one minute and we take two minutes before and after the center point to form a time window with a length of five.

When using K-means, the number of clusters is set to 4. The detected fault time and actual time are shown in the table 2.

| Actual Time          | Detected Time          |
|----------------------|------------------------|
| Case 1 13rd April 07:47-07:49 | 13rd April 07:47-07:49 |
| Case 2 19th April 2:10-2:15    | 19th April 2:14-2:15   |
| Case 3 28th April 19:55-20:27  | 28th April 19:55-20:27  |
| Case 4 12nd April12:27-12:30 | 12nd April12:27-12:30 |

We detected the failure time of the four cases sucessfully. In three cases, the detected time is almost as same as the actual time. In the other case, part of actual time is detected.

4.3. Fault Location

According to the detected fault time period, we will analyze the logs in this time. We can locate specific components or reasons using algorithms. Extract the first five keywords and the results are as in table 3.

In case 1, the five keywords are all related to this failure. Through these five keywords, it can be clearly understood that it is related to the creation of images and network.
In case 2, the keyword NOValidHost indicates that the fault was related to the creation of the virtual machine.

In case 3, the keyword dongle directly indicates that the failure was caused by a dongle.

In case 4, both keyword unreachable and keyword econnrefused indicate that the network was the cause of the failure.

| Case   | Cause                        |
|--------|------------------------------|
| Case 1 | creating image driver libvirt network_info |
| Case 2 | usr dist select_destinations NoValidHost file |
| Case 3 | Instinfoge lush_component monitor dongle get_cpu_rate |
| Case 4 | errno unreachable again econnrefused Trying |

4.4. Result and Analysis

In order to test the effectiveness of the algorithm, we calculated recall rate, fp rate and miss rate, as show in figure 1.

![Figure 1. Recall rate, fp rate and miss rate](image)

The recall rate is 0.8 and miss rate is 0.2. It is to say, we can detect 80% fault using the algorithm. The fp rate is 0.2 and the algorithm recognizes 20% normal cases as fault. Through the above experiments and analysis, our method is effective. Experimental results show that this algorithm has a very good effect and has a high practical value.

5. Conclusion

With the rapid development of cloud computing technology, how to detect the failure of the OpenStack system becomes a problem. This paper proposes a new algorithm that analyzes logs to detect failures by machine learning.

This algorithm does not need labeled data, only the original log. After the log format is unified, we create a time window, extract features extracted and discover a fault time period using clustering. Key words are extracted from the fault logs and the causes of the fault are found.

Through experiments, the algorithm has a very good effect on fault detection. It can detect 80% fault and only recognizes 20% normal cases as fault. It is of great help in locating OpenStack system faults.
6. References

[1] Lamourine M. OpenStack[J]. ; login:: the magazine of USENIX & SAGE, 2014, 39:pàgs. 17-20.
[2] Rao X, Wang H, Shi D, et al. Identifying faults in large-scale distributed systems by filtering noisy error logs[C]// Ieee/ifip, International Conference on Dependable Systems and Networks Workshops. IEEE Computer Society, 2011:140-145.
[3] Xu W, Huang L, Fox A, et al. Detecting large-scale system problems by mining console logs[C]// ACM Sigops, Symposium on Operating Systems Principles. ACM, 2009:117-132.
[4] Prewett J E. Listening to your Cluster with LoGS [C]// 2004.
[5] Park J, Yoo G, Lee E. Proactive Self-Healing System based on Multi-Agent Technologies[C]// Acis Int'l Conference on Software Engineering Research, Management and Applications. IEEE Computer Society, 2005:256-263.
[6] Li T, Liang F, Ma S, et al. An integrated framework on mining logs files for computing system management[C]// Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, Usa, August. DBLP, 2005:776-781.
[7] Vaarandi R. A data clustering algorithm for mining patterns from event logs[C]// Ip Operations & Management. IEEE, 2003:119-126.
[8] Duan S, Babu S, Munagala K. Fa: A System for Automating Failure Diagnosis[C]// IEEE, International Conference on Data Engineering. IEEE, 2009:1012-1023.
[9] Wei X, Ling H, Fox A, et al. Mining console logs for large-scale system problem detection[C]// Conference on Tackling Computer Systems Problems with Machine Learning Techniques. USENIX Association, 2008:4-4.
[10] Salfner F, Tschirpke S. Error log processing for accurate failure prediction[C]// Usenix Conference on Analysis of System Logs. USENIX Association, 2008:4-4.
[11] Kaufman L, Rousseeuw P J. Finding Groups in Data: An Introduction to Cluster Analysis [M]. DBLP, 1990.
[12] J A, Wong M A. Algorithm AS 136: A K-Means Clustering Algorithm [J]. Journal of the Royal Statistical Society, 1979, 28(1):100-108.
[13] Zhu L J, Bing-Xian M A, Zhao X Q. Clustering validity analysis based on silhouette coefficient[J]. Journal of Computer Applications, 2010.
[14] Kesorn K, Poslad S. An Enhanced Bag-of-Visual Word Vector Space Model to Represent Visual Content in Athletics Images [J]. IEEE Transactions on Multimedia, 2012, 14 (1): 211-222.
[15] Chum O, Philbin J, Zisserman A. Near Duplicate Image Detection: min-Hash and TFIDF Weighting [J]. 2008.