International Journal of Database Theory and Application
Vol.7, No.4 (2014), pp. 273-284
http://dx.doi.org/10.14257/ijdta.2014.7.4.22

Online Filtering of Massive Log Data in the Cloud Computing System

Li Zhou[1], Baojin Zhu[2], Xiaopeng Zheng[3] and Liye Zhang[4]

[1,2]*School of Computer Science and Technology, Hangzhou Dianzi University Hangzhou, China*
[3,4]*China Academy of Launch Vehicle Technology, Beijing, China*
[1]zhouli@hdu.edu.cn, [2]zhubaojin222@gmail.com, [3] zhxp1014@163c.om,
[4]zhangliye@163.com

Abstract

Log data is a valuable resource for failure prediction and troubleshooting in large-scale systems. However, with the rapid growth of the system scale and the popularity of various applications in productional environments, the volume of logs emerged per day becomes huge, posing serious challenges for storage and analysis. To solve these problems, we propose an online log filtering mechanism to eliminate the redundant and noisy log records through event filtering and instance filtering, aiming to minimize the log size without losing important information required for the fault diagnosis. Our proposed log filtering is evaluated on a real log data derived from a productional cloud computing system, observing that over 76% of the storage space are saved without losing important information.

Keywords: *log filtering, anomaly detection, vector space model, message type*

1. Introduction

With the constant growth of system size and complexity, reliability has become a major concern in productional environments. To describe the characteristics of system fault behaviors [1-2], event log is usually regarded as an important information source. With the continuous growth of the number of applications, massive log data are generated at a high speed, and thus massive storage resource is consumed, and therefore, it is rather significant to find effective ways to reduce the storage resource consumption. Many researchers have proposed some processing algorithms which are helpful to the system behavior analysis [3-5]. However, the complex inner relation between the large data volume and event logs is a handicap to most algorithms and it takes hours to obtain the important information from the massive data. The log data is the valuable information source, but sometimes, these data are useless except taking up the disk space.

As the huge log consume large-scale storage space, many systems have to selectively collect and archive key events like ERROR or FATAL events. A common-used prediction method is to extract the failure mode through the causal correlation between fatal and nonfatal events [6-7]. These studies ignore the fact that key events generally do not contain full information which can identify the root cause of failures. Liang et al. [8] proposed to delete the redundant records of system logs with temporal and spatial filters. Despite the high compression ratio, these filtering methods may delete some important failure features, that is, warning information stream before the failure that contains valuable information for failure

ISSN: 2005-4270 IJDTA

analysis. Therefore, it is necessary for a log filtering method which could fully reduce the log data capacity without losing any important information involved in logs.

To solve these problems, this paper proposes an online filtering mechanism to remove the redundant and noisy log data by using the methods of event filtering and instance filtering. In log collecting stage, redundant event logs are filtered from the time dimension through event filtering, and original logs are decomposed into a log document named instance according to the source and time of logs, then an unsupervised anomaly detecting algorithm based on the information entropy is adopted to find out the instance set R_k which may contain abnormal information. By contrasting the similarity degree of R_k with the remained instances, the most similar instance set R_k' is obtained. Besides the R_k and R_k', other instances are deleted.

In order to prove the effectiveness of our filtering methods, real event log data are collected from the production distributed system to evaluate the filtering mechanism. The experimental results show that our this filtering mechanism can approximately reduce 76% of the disk space without losing any important information used for failure predication and root cause analysis.

The remain of this paper is organized as follows. Section 2 describes the system event log data involved in this paper, and then presents an anomaly detection algorithm. Section 3 offers a detailed description of the filtering methods. Section 4 is the description of experimental procedures and experimental result analysis. Section 5 gives a brief review of some related work. Section 6 concludes this paper.

2. Background

To facilitate the understanding of the filtering of logs in this paper, we first give a brief description of output log formats in the target system[1]. Then, we introduce the processing of the algorithm which used to anomaly detection in event logs.

2.1. Overview of the System Log

Real event logs from a cloud computing system are used as the test data in our experiments. In this section, we give a brief description of output log formats and important observations found in the preliminary analysis in the target system. Event logs in the target system is mainly used to record system software behavior, but not the description of the underlying hardware failure or hardware state.

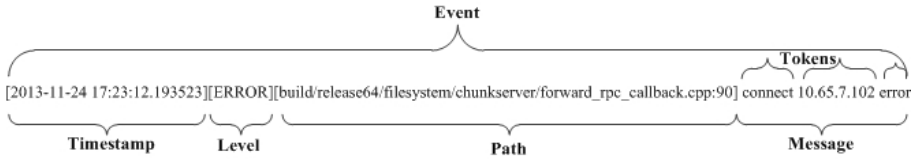

Figure 1. The Structure of System Log Event

As shown in Figure 1, a log entry mainly consists of four description fields: ***Timestamp***, which indicates the time when the event appears with the accuracy of microsecond; ***Level*** is the severity of the log, mainly including INFO, WARNING, ERROR and FATAL; ***Path*** is the exact line of the source code; ***Message*** refers to the specific content of the log, which is generally divided into static text, the status variables and the track trace. In addition, Host IP and Server Process can be obtained in the log generating circumstance, which respectively

[1] This target system is from Alibaba Cloud Computing Company

refer to the IP address of log generating host and the corresponding process name. In [1, 8, 9], ERROR and FATAL event logs are collected for failure detection or root cause diagnosis, but ignore the important contextual information contained in WARNING and INFO levels.

2.2. Anomaly Detection Algorithm

The task of anomaly detection is to find out events which can reflect any system exceptions or log areas that contain exceptional events through the analysis of event logs. Oliner *et al.*, [11] adopt more complex weighting model of *log.entropy* lexical items to make anomaly detection with event logs and propose and implement the anomaly detecting algorithm(*Nodeinfo*) of system logs. *Nodeinfo* algorithm decomposes logs into *nodehours*: all lines from a single node(c) in one hour intervals. Let H_j^c be the j^{th} *nodehour* for node(c). *Nodeinfo* bases its assessment of each *nodehour* on the information content of the individual tokens, t_1 to t_p in the free form message(m_i) of an event e_i.

A sparse $M \times N$ matrix is first created for log indexing, where non-zero values in the matrix indicate the number of times word *i* occurring during the *nodehour j*. The following step is to compute the amount of information each token conveys. Let *W* be the set of all terms(a concatenation of a word and its token position within a message). Let *C* be the count of nodes in the system. Let *X* be a $|W| \times C$ matrix, where $x_{w,c}$ is the number of times term *w* appears in messages generated by node c. A vector *G* with cardinality $|W|$, indicating how each term is distributed among nodes on the network, where g_w is calculated by Eq.1 that corresponds to 1 plus each terms Shannon information entropy[12]. Its value ranges between 0 and 1, with 0 signifying low information content for the term and 1 signifying the highest information content possible. Terms with high information content are more likely to be conditions which are useful for an administrator.

$$g_w = 1 + \frac{1}{\log_2^{(C)}} \sum_{c=1}^{C} p_{w,c} \log_2 p_{w,c} \tag{1}$$

$$p_{w,c} = \frac{x_{w,c}}{\sum_{c=1}^{C} x_{w,c}} \tag{2}$$

A weight of 1 is received by a term appearing on only a single node, and a term appearing the same number of times on all nodes receives a weight of 0.

A Nodeinfo score can then be assigned to each *nodehour* based on the entropy of the terms contained in the *nodehour* and how many times they appear. Let *H* be the set of all *nodehours*, and let *Y* be $|W| \times |H|$ matrix, where $y_{w,c,j}$ is the count of the number of times term *w* appears in *nodehour* H_j^c. The Nodeinfo value of each *nodehour* in *H* set can be calculated by Eq.3.

$$Nodeinfo(H_j^c) = \sqrt{\sum_{w=1}^{W} (g_w \log_2 (y_{w,c,j}))^2} \tag{3}$$

Finally, a list of *nodehours* ranked by decreasing their Nodeinfo value can be established. It is considered that *nodehours* with high Nodeinfo score are more likely to contain anomaly events than those coming up lower in the ranking. Although *Nodeinfo* algorithm has utilized the concepts of encoding token and position pairs, it does not fully capture message contexts because it does not consider message types.

3. Filtering Procedures

As shown in Figure 3, our proposed filtering mechanism consists of two major components: event filtering and instance filtering. Specifically, instance filtering is mainly composed of three steps: instance archiving, anomaly instance selection and instance similarity evaluation. In the first step, logs are decomposed into several instances, the region of event log, by source and time interval. In the second step, anomaly instances are selected based on the anomaly detecting algorithm of information entropy. In the last step, the similarity degree between the selected anomaly instances and the remaining instances is computed to find out and save the instance set with higher similarity degree, together with the anomaly one.

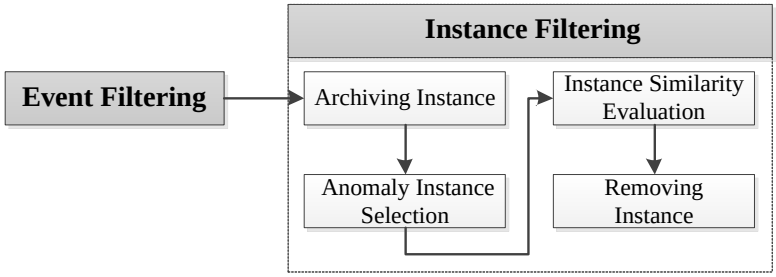

Figure 3. The Processing of Log Filtering

3.1. Event Filtering

With the continuous increase of data center scale, log collection has become a challenging task, especially the ways to ensure the real-time collection. In the log collector component, inotify mechanism [13], which is included within Linux Kernel to monitor filesystem events, is applied to ensure the realtime requirement. By using inotify mechanism, the system can periodically monitor the target directory or object files to achieve the real-time collection of log data in the target system.

The main goal of event filtering in the log collection phase is to remove duplicate event reported within one second in the same node. As has mentioned in Section 2, a event duration for second level will be recorded for thousands of times, so duplicate entries should be compressed from a temporal view. By using the event filtering method in log collectors deployed in the target machines, we can not only delete large amounts of redundant data, but also reduce network resources consumed by data transmission.

3.2. Instance Filtering

3.2.1 Archiving Instance: According to the log generated node and time interval, log data are decomposed into instance, which indicates log region. As shown in Figure 4, D_j^c refers to the j^{th} instance generated by the node c, and ΔT is the time interval, which is not equal to the one hour set in Nodeinfo [10-11]. We can observe that the average load and handling capacity of machines in the target system are relatively high, which indicates the high log output particle size of the system. If the time interval is valued one hour, the instance size will be big, which is against the after-treatment. In this paper, the value of ΔT is firstly enumerated as 30min, 40 min, 60 min, *etc*. By observing the effect of these values on the evaluation index in the experiments, we can obtain a proper guideline value.

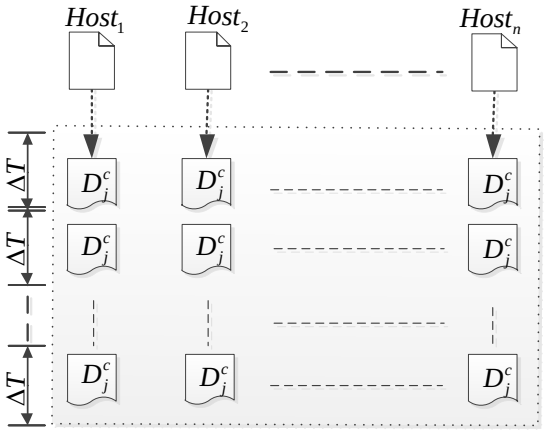

Figure 4. Diagram of Decomposing Logs

A sparse $|W|\times C$ matrix is first created for log indexing, where non-zero values in the matrix indicate the number of times word *i* occurring during the instance *j*. Let *W* be the vector of all terms, which are formed by a concatenation of a word, its token position within a message and index within message type template (Section 3.2.2) and whose encoding is different from one in [10-11]. Let *C* be the count of nodes in the cluster. The instance is defined with a combination of dates and host name. For example, a notation "20130513axdfbafa06" of instance, where "20130513" represents the date, "axdfbafa" indicates the only identity for a machine in the cluster and "06" is the j^{th} instance for node "axdfbafa" in "20130513".

3.2.2. Message Type Template: Although event logs are emitted in the form of free text, as a matter of fact, they are pretty structured because they are generated entirely from a relatively small set of log printing statements in source code.

For example, the followings are four lines of log message text content:
"connect 10.65.7.56 error"
"connect 10.65.7.123 error"
"connect 10.65.7.58 error"
"connect 10.65.7.83 error"

The contents of these four log records are the same except the IP address, and therefore can be classified as a same type of events. If using message type template, these four logs can be described with the event type of "connect(*) error".

Before the instance filtering, all log output statements in source codes are analyzed to abstract a unique template of event message type consisting of constant symbols and variable symbols, then a template index in the form of $Map < key , value >$ is structured based on the message type. Let key be the hexadecimal four figures from "0x0000", and value is the element in the message type template. The structure is shown in Figure 5.

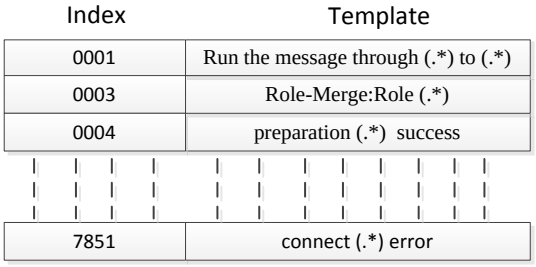

Figure 5. Message Type Index Table

The abstracted message type template has two effects on instance filtering. On the one hand, it can save the computational overhead of term vector construction and reduce memory overhead. On the other hand, considering the insufficiency of the similarity calculation method based on vector-space in the aspect of semantic similarity judgment, when comparing the similarity measure among instances, we introduce the degree of overlap message type(Section 3.2.4), which is a novel metric for measuring the similarity between instances.

3.2.3. Anomaly Instance Selection: The goal of anomaly instance selection is to choose instances containing critical information for system behavior analysis and remove useless ones. Generally, instances can be classified into two types: one contains information related to system anomaly and the other has no anomaly events. For administrators, instances containing system anomaly have more useful information than the normal ones. By improving encoding scheme of terms, we present *Messageinfo* algorithm to select instances which are most likely containing abnormal events.

It has been proved that, the encoding of word and position pair is a simple form of message context information, position which refers to a words position in the message is encoded as a four digit hexadecimal prefix. For instance, the term "0004Error" indicates that the word "Error" appears as the fourth word of a message. To create the whole vector of terms, we have to scan all event logs generated by all nodes in the system, which may increase the size of indexing matrix and bring huge calculation of entropy in *Nodeinfo* algorithm. As the distribution of terms across the nodes is more important than terms themselves, we can utilize the term set created by message type template extracted from source codes to transform the messages in logs and reduce the number of unique terms.

By using *Messageinfo* algorithm, the Messageinfo score of all instances can be computed and ranked in descending order. Let R_k be the set of instances formed by taking top k instances from above ranking list, where k is assigned to 1000. Let R_{n-k} be the set of instances excluding R_k.

3.2.4. Instance Similarity Evaluation: Vector space model (VSM) is the most commonly used similarity calculation model, which has been widely used in natural language processing. The document is represented as $D=D(W_1,W_2,...,W_n)$, where W_n refers to a weight to the term which calculated by the *tf.idf* weighting scheme. The document similarity is often defined by the cosine of the angle between document vectors, whose value is calculated by Eq.4.

$$sim\ (D_1, D_2) = \frac{\sum_{k=1}^{n} W_{1k} \times W_{2k}}{\sqrt{(\sum_{k=1}^{n} W_{1k}^2)(\sum_{k=1}^{n} W_{2k}^2)}} \quad (4)$$

where W_1, W_2 indicate the k^{th} feature weight of document D_1 , D_2 respectively. However, traditional text similarity only calculates the frequency of feature items, but ignores the semantic differences between the texts. Instances here refers to log documents. The degree of overlap message type between two instances reflects the similarity to some extent, so we introduce the new measure metric that indicates the semantic similarity between instances.

$$Semantic \quad Similarity \quad (SS) = \frac{n_{D_1, D_2}}{Min\ (D_1, D_2)} \tag{5}$$

where n_{D_1, D_2} indicates the number of message type which appears in both D_1 and D_2, and $Min(D_1,D_2)$is the lower value of the number of message type from D_1 and D_2, respectively. The value of *SS* ranges from 0 to 1, the higher value indicates that two instances have great similarity in semantics, with 0 signifying on the same message type in two instances, and 1 signifying that two instances cover all message types. Eq.6 is an improved similarity formula, where b is just used to zoom in similarity calculation result.

We first calculate similarity between all instances in R_{n-k} and ones in R_k, and then select the maximum value from results as the eventual similarity of each instance with R_k. Let $R_k^{'}$ be the set of instances which are most similar to R_k, where the similarity is over 0.5. Finally, R_k and $R_k^{'}$ will be saved in database, while others are removed.

4. Experiments

4.1. Evaluation Metric

The goal of our filtering mechanism is to significantly reduce the data volume of logs without losing critical information for failure prediction or root cause diagnosis. Hence, we evaluate our filtering mechanism by following metrics.

Data Reduction Rate: Eq.7 is adopted to evaluate the amount of data volume reduced by the filtering mechanism, with the following definition:

$$Data \quad Reduction \quad Rate(DRR) \quad = \frac{Raw \quad Size \quad - \quad Filterd \quad Size}{Raw \quad Size} \times 100 \ \% \tag{7}$$

Raw Size refers to the size of raw data volume before filtering, while *Filtered Size* is the size of log data volume filtered by our filtering mechanism.

4.2. Results and Analysis

The evaluation experiments are respectively conducted on three different clusters composed of 100 nodes, 200 nodes and 500 nodes. Therefore, we make four sets of experiments on event log collection.

4.2.1. Off-line Usage: In the first set of experiments, we create two sparse matrix for log indexing, one stands for term-instance matrix without combining with message type information, and the other represents term-instance matrix obtained by using message type. Based on the above data set in three different clusters, *Nodeinfo* algorithm and *Messageinfo* algorithm are used to calculate each instance score and obtain the corresponding ranking lists of $R_{nodeinfo}$ and $R_{messageinfo}$. Due to the limited article length and similar experimental results in

three clusters of different scales, here we only present the results gained from clusters containing 100 nodes.

To demonstrate the effectiveness of *Messageinfo* algorithm in detecting event log anomaly, the differences of *Messageinfo* algorithm and *Nodeinfo* algorithm in anomaly detection are compared. According to the $R_{nodeinfo}$ and $R_{messageinfo}$, the step size ΔK is divided to compare the number of instances in every step size. The experimental results are shown in Figure 6. In this set of experiments, the time interval is set in 3600 seconds, the testing data is the 90-day log data generated by the system consisting of 100 nodes, the total number of instance is 204,453, and the step size ΔK is 1000.

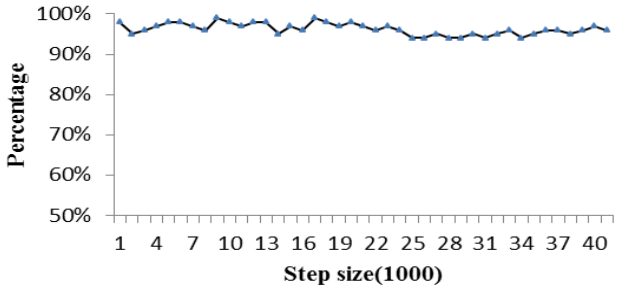

Figure 6. The Percentage on the Same Instances within Same Step k in Two Rankings Obtained by Nodeinfo and Messageinfo Algorithm.

As shown in Figure 6, the percentage obtained the same instances with two different algorithms in each step size ranges from 94% to 98%, which indicates that in the aspect of anomaly detection, *Messageinfo* algorithm can achieve the same detecting effect with *Nodeinfo* algorithm.

In the second set of experiments, different values of time interval are used to check the corresponding data reduction rata, and contrast experiments are made in different cluster scales. Figure 7 shows that, in each cluster when the time interval is 2400 seconds, the value of DRR is the biggest, respectively of 78%, 76%, and 79%. Moreover, it can also be observed that in each combination, the value of DRR ranges from 65% to 79%, which shows that our filtering method can increase the reduction rate in any case. Though the reduction rate is not as high as 99.9% in some research, our goal, as has mentioned before, is not only focusing on the high reduction rate, but trying to maximally reduce log data without losing any information related to failure cause diagnosis.

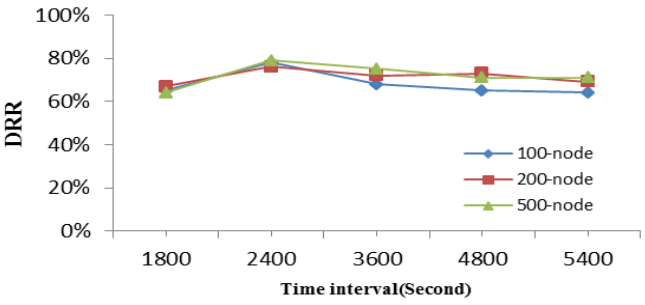

Figure 7. The Value of DRR on Testing Data with Different Combinations of the Size of Cluster and Time Interval

To compare the influence of *Nodeinfo* algorithm and *Messageinfo* algorithm on the data reduction ratio and observe the effect of traditional cosine similarity(Eq.4) and improved similarity measurement(Eq.6), a new treating process is added in the third set of experiments on the basis of the second set by using *Nodeinfo* algorithm to conduct abnormal instance selection and traditional cosine similarity measurement as the similarity judgment among instances. According to the result of the second set of experiments, the time interval is set as 2400 seconds. As shown in the Figure 8, the DRR value obtained in the newly added treating process is lower than that in the second set of experiments. The causes can be divided into two aspects: on the one hand, there are less abnormal instances and more normal instances detected in the set with *Nodeinfo* algorithm, thus more normal instance logs are saved, which will cause the reduction of DRR; on the other hand, traditional text similarity calculation formula might cause the inaccuracy of the calculation result due to the lack of feature words among texts. For instance, if Eq. 4 is used, the similarity between the two texts $D = D(10,1,1,1,1 \quad ,1,0,1,0,0 \quad)$ and $D = D(10,0,0,0,0 \quad ,0,0,0,1,1 \quad)$ would be 0.96, which is obviously incompatible with the fact. Moreover, if the dissimilar instances are judged as similar ones, more instance logs will be saved, thus causing the reduction of DRR value.

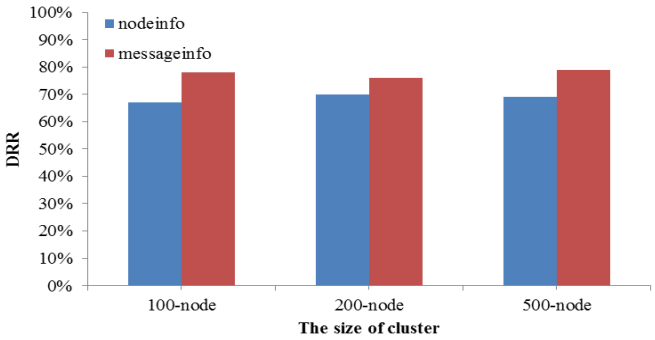

Figure 8. The Histogram of DRR Value on Testing Data. The Nodeinfo Indicates the Newly Added Treating Process and the Messageinfo is the Result of the Second Set of Experiments, which Time Interval is 2400 Seconds

4.2.2. Online Usage: The filtering mechanism as described is directly suitable for event log analysis in the offline environment, which is valuable for the deep data mining. However, in the production environment, online processing technology with low latency is needed. Therefore, to ensure the availability in the production environment, sliding time window model is used to implement the online version of our filtering mechanism.

In the fourth set of experiments, time window size is changed in different clusters to observe the DRR changes. According to the results in the second set of experiments, time interval is set in 2400 seconds, with the time window size of 7 days, 14 days, 21 days and 28 days. The motivation of the experiments is to evaluate the effectiveness of this online filtering mechanism in different cluster scales, that is, data reduction ratio. The experimental results are shown in Figure 9.

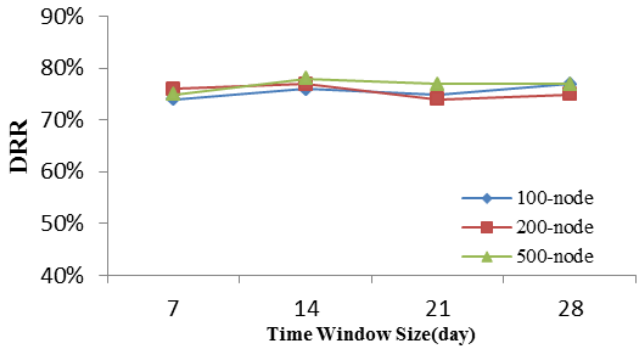

Figure 9. The Value of DRR on the Testing Data. There are Three Curves in the Plot, Indicating Different Combinations of the Size of Cluster and Time Window Size

From Figure 9 we can see that, in each combination, the value of DRR ranges from 74% to 78%, which is extremely similar with that in Figure 7 at the time interval of 2400 seconds. Besides, three smooth curves without obvious ascending or descending trends indicate the small influence of time window size on DRR. Considering the computing and memory overhead of the whole filtering process, it is suggested to set the time window in 7 days.

5. Related Work

System logs have provided a rich information source for failure detection, failure prediction and root cause diagnosis, but with the continuous increase of the system size, it is a challenging task to collect, analyze and manage logs. Relevant work has been made on the log compressing or semantic filtering from time and spatial dimensions by removing logs generated by different machines and fixed time windows to realize high log compression [1, 14]. However, this log data treatment, which mainly focuses on data compression, cannot ensure the loss of valuable information of failure cause diagnosis. Our filtering method could preserve all level event related to faults.

Log filtering technology has been widely used in system log analysis and handling process. The existing research can be approximately divided into the instance based method and the feature based method. The instance based approach is generally used to identify instances containing abnormal information and delete instances with redundant information. Zheng *et al.*, [14] propose a causal correlation filtering method to accurately find a collection of common occurring frequent fatal events and then filter them out. The major goal of the feature based method is to select or extract the subset of relevant features. Yang *et al.*, [15] present a log reduction method based on the statistical data to find a system index which can fully describe the application behaviors, thus to effectively reduce the management data volume. Zhang *et al.*, [16] put forward a bayesian network model to predict the dependency of system behavior indexes in the dynamic environment by using the feature selection method for system index subset.

An online filtering method is proposed in [17] in view of the environmental log data, which include the hardware temperature, clock frequency, fan speed and voltage information in runtime. The archiving instance part of our filtering mechanism is inspired by the work of Oliner *et al.*, [11], which explores *Nodeinfo* to alert detection in system logs. This paper extend the encoding of term with message type template and present the new measure metric used to indicate the semantic similarity between

instances. Once a fault is detected, our mechanism could preserve both fatal and non-fatal events.

6. Conclusions

Based on the real event logs in the target system, this paper proposes a log filtering mechanism, aiming to maximally remove redundant and noisy data in these logs without losing any important information. Event logs usually contain abundant information related to failure detection, performance analysis and root cause diagnosis of the system. However, with the continuous growth of application scale, these logs will completely occupy our disks with great ease and increase the time overhead of log treatment. By analyzing log output statements of source code, we can abstract a log message type template and use it to reduce the size of characteristic vectors in anomaly detecting algorithm, thus to reduce the CPU and memory overhead in the computing process. Moreover, an improved similarity measure is put forward to evaluate the similarity between instance and anomaly instance set. The experimental results show that this filtering mechanism can reduce about 76% of the log data volume.

Acknowledgements

These should be brief and placed at the end of the text before the references.

References

[1] Y. Liang, Y. Zhang, A. Sivasubramaniam and R. K. Sahoo, "Filtering failure logs for a bluegene/l prototype," in Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on. IEEE, **(2005)**, pp. 476–485.

[2] W. Xu, L. Huang, A. Fox and D. Patterson, "Detecting large-scale system problems by mining console logs," in Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, **(2009)**, pp. 117–132.

[3] Z. Xue, X. Dong, S. Ma and W. Dong, "A survey on failure prediction of large-scale server clusters," in Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on, vol. 2. IEEE, **(2007)**, pp. 733–738.

[4] F. Salfner, M. Lenk and M. Malek, "A survey of online failure prediction methods," ACM Computing Surveys (CSUR), vol. 42, no. 3, **(2010)**, pp. 10.

[5] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high performance computing systems," Dependable and Secure Computing, IEEE Transactions on, vol. 7, no. 4, **(2010)**, pp. 337-350.

[6] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta *et al.,* "Critical event prediction for proactive management in large-scale computer clusters," in Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, **(2003)**, pp. 426–435.

[7] Z. Lan, J. Gu, Z. Zheng, R. Thakur *et al*., "A study of dynamic meta-learning for failure prediction in large-scale systems," Journal of Parallel and Distributed Computing, vol. 70, no. 6, **(2010)**, pp. 630-643.

[8] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on. IEEE, **(2007)**, pp. 583–588.

[9] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam *et al*., "Bluegene/l failure analysis and prediction models," in Dependable Systems and Networks, 2006. DSN 2006. International Conference on. IEEE, **(2006)**, pp. 425–434.

[10] J. Stearley and A. J. Oliner, "Bad words: Finding faults in spirit's syslogs," in Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on. IEEE, **(2008)**, pp. 765–770.

[11] A. J. Oliner, A. Aiken, and J. Stearley, "Alert detection in system logs," in Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on. IEEE, **(2008)**, pp. 959–964.

[12] M. W. Berry, Z. Drmac and E. R. Jessup, "Matrices, vector spaces, and information retrieval," SIAM review, vol. 41, no. 2, **(1999)**, pp. 335-362.

[13] R. Love, "Kernel korner: Intro to inotify", Linux Journal, vol. 2005, no. 139, **(2005)**, pp.8.

[14] Z. Zheng, Z. Lan, B.-H. Park and A. Geist, "System log pre-processing to improve failure prediction," in Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on. IEEE, **(2009)**, pp. 572–577.
[15] L. Yang, J. M. Schopf, C. L. Dumitrescu and I. Foster, "Statistical data reduction for efficient application performance monitoring," in Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on, vol. 1. IEEE, **(2006)**, pp. 8–pp.
[16] S. Zhang, I. Cohen, M. Goldszmidt and J. Symons, "Ensembles of models for automated diagnosis of system performance problems," in Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on. IEEE, **(2005)**, pp. 644–653.
[17] L. Yu, Z. Zheng, Z. Lan and T. Jones, "Filtering log data: Finding the needles in the haystack," in Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on. IEEE, **(2012)**, pp. 1–12.

Authors

Li Zhou received her Master Degree from Hangzhou Dianzi University, Hangzhou, China, in 2003. She is currently an associate professor in School of Computer Science and Technology, Hangzhou Dianzi University. Her current research interests include virtual storage system, cloud storage, cloud computing and high performance computing, etc.

Baojin Zhu received the Bachelor Degree of Computer Science and Technology in Changchun University of Science and Technology, Jilin, China, in 2011. He is now studying the Master of computer technology in Hangzhou Dianzi University, China. His research interest is big data process and system log analysis.

