Log mining and knowledge-based models in data storage systems diagnostics

Mikhail B. Uspenskij^{1,*}

¹Peter the Great St. Petersburg Polytechnic University, St. Petersburg, Russian Federation

Abstract. Modern data storage systems have a sophisticated hardware and software architecture, including multiple storage processors, storage fabrics, network equipment and storage media and contain information, which can be damaged or lost because of hardware or software fault. Approach to storage software diagnostics, presented in current paper, combines a log mining algorithms for fault detection based on natural language processing text classification methods, and usage of the diagnostic model for a task of fault source detection. Currently existing approaches to computational systems diagnostics are either ignoring system or event log data, using only numeric monitoring parameters, or target only certain log types or use logs to create chains of the structured events. The main advantage of using natural language processing method for log text classification is that no information of log message structure or log message source, or log purpose is required if there is enough data for classifier model training. Developed diagnostic procedure has accuracy score comparable with existing methods and can target all presented in training set faults without prior log structure research.

1 Introduction

Diagnostics of the data storage systems is a sophisticated and important task since such equipment is now widely used for storage of the sensitive data, including data related to environment monitoring, maps and satellite photos of the earth surface, ecological monitoring etc. Damage or loss of such data can be both expensive and/or harmful for education and research activities. Modern data storage systems consist of the different subsystems and components, such as storage processors, storage fabrics, network equipment and storage media. Fault detection in such systems means not only detection of the problems in the separate components, but also detection of problems in the component interaction. Although diagnostic approaches, data-driven or model-based, using various sources of monitoring data are fairly widespread [1-3], there is a reliable alternative source of diagnostic information, which is not so actively used – software logs. Logs, generated by storage system components, contain a lot of specific information, considering system behaviour and component interaction[4].

Currently there is a number of recent researches in a field of computing system diagnostics, on a possible means of diagnostic information extraction from software logs [5-7], but they usually are based on transformation of log plain text to chains of structured events. This approach has several major drawbacks:

- For each log, used in diagnostic procedure, log message structure should be painstakingly analysed in order to extract useful message templates;

- For each message template, a separate parsing procedure should be developed;
- To find a relation between chain of messages and a fault event, the diagnostic procedure developer should have an extensive prior knowledge about fault route cause and log message purpose and source.

Log template parsing algorithms [8-11] partially solve first problem, but the other two are hard to tackle. Alternative approach suggests using text classification algorithms for log data error labelling [12-13], but presented algorithms have some serious drawbacks:

- only logs in certain format can be processed;
- log analysis is performed not in a run-time, but as a process of fault investigation, so the whole log text is used for classification purposes.
- log analyzing procedure takes only single log message for parsing and fault detection.[13]

2. Methods

This work presents a new approach to storage system technical diagnostics, based on the fusion of model-based and data driven approaches. Implementation of this approach implies usage of the storage system software event and system logs as a source of the diagnostics data together with the expert knowledge of the storage system hardware and software architecture to diagnose faults and prevent data loss and performance degradation.

2.1 Ontological diagnostic model

The approach, presented in current paper, uses ontological modelling as method for formal definition of the expert knowledge about storage system.

* Corresponding author: mikhail.uspenskiy@spbpu.com

Diagnostic model contains:

- Storage system health model, including dependencies of storage system subsystems and components health states.
- Storage system components hierarchical structure.
- Relation between storage system parameters and states.
- Other classes and properties, used e.g. for log pre-processing.

Ontological model consists of ontological classes for the storage components, states, parameters etc., object properties for the definition of the relations between classes and data properties for the parameter values description.

Because of the fuzzy nature of the relations between monitoring data and health states of the sophisticated hardware and software systems, such as data storage systems, this paper presents a method for a definition of implicit links between parameter and failure classes of the ontological model (figure 1). An implicit link between parameter values vector $\{P_i\}$ and possible component failure types $\{Y_k\}$ consists of ontological object properties «described_by» and «solves_with» and an ontological implicit relation class, which contains the external programming means for state diagnosis.

Fig.1. Implicit relation

In case of the storage processor logs as a source of diagnostic parameters, such implicit relation class contains model of the text classification algorithm [14-15], pre-trained on the training set.

2.2 Implementing diagnostic procedure based on ontological model

Figure 2 describes the fault diagnosis procedure based on the current implementation of ontological model.

Fig.2. Diagnostic procedure

Presented diagnostic procedure can be easily adopted to any storage system with the similar

structure by redefining the rules for topology and diagnostic data extraction. The model structure redevelopment is not required. Current implementation of diagnostic procedure uses system and event logs located on storage processors as a data source for the diagnostic parameters.

On the first step, the ontology model is created using expert data in an external ontology editor, such as Stanford Protégé[16] software.

On the second step, diagnostic service generates a simplified graph form of the model. Diagnostic service creates vertices and edges of the oriented graph from the classes, data and object properties of the ontological model and their instances from current storage system topology providing service, according to ontological model class and property definitions.

After that, diagnostic service stores generated model into a graph database[17-18] and queries it in the diagnostic parameters processing loop.

For the simplified graph representation of the ontological model, nquads format is used, where each two vertices are connected with the edge that can have a weight object, containing some kind of additional information, such as, e.g. default value of the diagnostic parameter.

2.3 Storage system event logs as a source of diagnostic data

Enterprise level unified storage system usually contains a set of storage processors, used to control the access from external network to actual disk drives. Storage processor software consists of a number of services, all of which generate one or more logs, used to store information about service events, including description of different faults and errors occurred in service run-time. Usually, the huge size of generated data makes log analysis impossible without some sort of pre-processing.

Approach presented in current paper implies application of the natural language processing methods to log mining in order to label log fragments as related to different system faults.

2.3.1 Storage system log types

Since storage processors in analysed storage system are linux-based, software logs are mostly located in “/var/log/” system folder. All software logs could be classified according to software purpose and type of the logging data:

- OS logs.
- application logs.
- service logs.

OS logs usually contain messages about system activity, such as authentication, system daemon management, system error events, etc., e.g. authorization log, daemon log, debug log for Ubuntu; warn, messages, lastlog for OpenSUSE and Suse Linux Enterprise Server. Table 1 contains some most common OS log types.

Table 1. Common log types

Log type, log name	Description
Logs for system messages (syslog, messages etc)	Contain messages from different sources, such as applications, kernel from the time of the system boot. They may contain information about errors, hardware problems etc.
Logs for system boot messages (localmessages, boot.msg, boot.log)	Contain information about process of boot and initialization of OS or software services
Logs for device messages (dmesg)	Contain data, generated by device drivers, including fault and error messages
Logs for user activity (utmp, wtmp, lastlog)	Contain information about recent user activity, such as login/logout
Logs for error messages (warn, faillog)	Contains messages about errors, occurred in OS

Some applications like web-severs or clustering software also create their own logs in common log folder. Each application can have one or more dedicated logs in a corresponding folder inside `“/var/log/”`.

In addition, there is a separate group of logs, called “Non-Human-Readable Logs”, processed only with the help of special software utilities.

Ontological diagnostic model may contains all possible types of logs that are human-readable and contain messages with timestamp, because these are the only restrictions of the presented approach based on NLP text classification.

2.3.2 Storage system log message format

Currently, there are many different log formats, most of which are more or less related with the Syslog format. Syslog is defined by several major specifications, such as Syslog – Syslog Protocol (RFC 5424), BSD syslog (RFC 3164), Common Logfile Format (RFC 6872). In addition, RFC 5427 describes major text rules for Syslog message classification, such as category types and severity levels of log message.

According to a syslog specification, message consists of three major parts – header, structured data area and message area. Even if the log is not exactly syslog-compatible, it usually contains all or some of the following common fields:

- **TimeStamp.** This field contains information about log message event occurrence time. There is no common format for the timestamp, so it can be like `«Feb 26 17:21:22», «2019-08-05T09:53:24.081021+03:00»` or `«2019/08/05 09:32:46»`.
- **Hostname.** This field contains message identified, which can be, in example, ip-address or string identifier.
- **Application/PROCID.** String field, containing application name and process identifier;

- **Severity Level.** Field contains either numeric code or string representation of log message severity level.

Some software applications like Nginx web-server have their own log format message specifications, but allow exporting messages in one of the common formats, like `[yyyy/mm/dd hh:mm:ss] [[level]] [pid#tid:] [*cid] [message]`, where first two fields are timestamp and criticality level, followed by process identifier as `[Process identifier#Message identifier]` and connection identifier as `[*Connection identifier]`.

Other commonly used log format – common logfile format (clf) is usually used in web-server applications and has following structure: `[remotehost] [rfc931] [authuser] [date] ["request"] [status] [bytes]`, where:

- `[remotehost]` is a name or ip-address of the remote node.
- `[rfc931]` – user identifier according to RFC 1413[23/08/2019-2].
- `[authuser]` – request author identifier.
- `[date]` – date (default format: `%d/%b/%Y:%H:%M:%S %z`).
- `["request"]` – request text.
- `[status]` – HTTP Status Code of the reply;

3 Results and Discussion

Presented approach text classification process consists of following steps:

- **Text preprocessing.** Text preprocessor removes not-alphabetical characters, splits tokens using whitespace delimiter and applies Porter stemming algorithm.
- **Word embedding[19].** Word embedding algorithm transforms each token to a set of numerical values, based on term frequency and inverse document frequency ranks.
- **Additional parameters calculation.** Additional parameters are calculated automatically based on text properties. All additional parameters are presented in table 2.
- **Text classification.** Text classification is performed with Random Forest[20] machine learning classification algorithm.

Table 2. Additional log parameters

Parameter	Value type	Parameter description
Log number of words	Int	All words in i-th log in current time interval
Log number of messages	Int	All messages in i-th log in current time interval
Log message length	Float	Average message length in i-th log in current time interval
Log average word count per second	Float	Average words per second in i-th log in current time interval
Log average message count per second	Float	Average messages per second in i-th log in current time interval

Data set, used for the approach evaluation, contains ~6000 log packages, containing 1600 fault events, occurred in storage system previously with 41 type of distinct fault events. Log package contain a set of 33

distinct system logs with specified fault event timestamp, if any.

In the process of evaluation, the described dataset was splitted 10 times randomly in training and testing subsets. Average fault detection rate on testing subset is 0.74, which is comparable with currently existing approaches. Separate evaluation of the algorithm with and without additional parameters shows, that without additional parameters it is around 20% less accurate.

Although the presented approach has an apparent drawback that it requires a decent amount of training data in order to add any new fault definition to the diagnostic model, it's advantage is that it does not requires any log structure analysis. Developed diagnostic procedure implementation is embedded in storage system software and located on one of the storage processors (fig.3).

Fig.3. Diagnostic procedure

4 Conclusions

The developed approach based both on the diagnostic modelling and log data mining has some important advantages over traditional model-based and data-driven approaches:

- External procedures (pre-trained models in case of log processing) allow defining implicit relations between parameters and states of the storage system that cannot be defined using the ontological object and data properties.
- Model creating process is much more simple and straightforward.
- Model adoption process requires only topology provider and data sources configuration.
- Diagnostic procedure can rely on machine learning algorithms and expert knowledge about storage system structure and faults.

Proposed log mining algorithm can be implemented for any plain-text log type, without prior knowledge of the log purpose or structure, if its messages have timestamps and are correlated to some faults, occurred in storage system.

A possible future improvement of the text classification quality may include introduction of the more complex word embedding models, such as BERT embedding[].

Acknowledgements: the research is carried out with the financial support of the Ministry of Science and Higher

Education of Russian Federation within the framework of the Federal Program "Research and Development in Priority Areas for the Development of the Russian Science and Technology Complex for 2014 - 2020". The unique identifier is RFMEFI58117X0023.

References

1. F. Lopes, J. Agnelo, C. A. Teixeira, N. Laranjeiro, and J. Bernardino, *Futur. Gener. Comput. Syst.* 102, 932 (2020)
2. D. W. Kang and T. S. Kim, *Appl. Energy* 212, 1345 (2018)
3. A. Slimani, P. Ribot, E. Chanthery, and N. Rachedi, *IFAC-PapersOnLine* 51, 1205 (2018)
4. M. Cinque, D. Cotroneo, and A. Pecchia, *IEEE Trans. Softw. Eng.* 39, 806 (2013)
5. W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, in *Proc. ACM SIGOPS 22nd Symp. Oper. Syst. Princ. - SOS'09* (ACM Press, New York, New York, USA, 2009), p. 117
6. I. Yagoub, M. A. Khan, and L. Jiyun, in 2018 *Int. Conf. Adv. Big Data, Comput. Data Commun. Syst. IcABCD 2018* (Institute of Electrical and Electronics Engineers Inc., 2018)
7. K. Yamanishi and Y. Maruyama, in *Proceeding Elev. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. - KDD '05* (ACM Press, New York, New York, USA, 2005), p. 499
8. R. Vaarandi, in *Proc. 3rd IEEE Work. IP Oper. Manag. (IPOM 2003)* (IEEE Cat. No.03EX764) (IEEE, n.d.), pp. 119–126
9. C. Liu, Y. Zhong, and Y. Wang, in 2018 *5th Int. Conf. Syst. Informatics, ICSAI 2018* (Institute of Electrical and Electronics Engineers Inc., 2019), pp. 1193–1198
10. A. Pande and V. Ahuja, in *Proc. - 2017 IEEE Int. Conf. Big Data, Big Data 2017* (Institute of Electrical and Electronics Engineers Inc., 2018), pp. 1095–1100
11. P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, in *Proc. - 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, DSN 2016* (Institute of Electrical and Electronics Engineers Inc., 2016), pp. 654–661
12. C. Bertero, M. Roy, C. Sauvanaud, and G. Tredan, in *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE (IEEE Computer Society, 2017)*, pp. 351–360
13. N. Aussel, Y. Petetin, and S. Chabridon, in *Proc. - 26th IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst. MASCOTS 2018* (Institute of Electrical and Electronics Engineers Inc., 2018), pp. 237–243
14. B. Altinel and M. C. Ganiz, *Inf. Process. Manag.* 54, 1129 (2018)
15. A. M. Aubaid and A. Mishra, *TEM J.* 7, 902 (2018)
16. M. Vigo, N. Matentzoglou, C. Jay, and R. Stevens, *J. Web Semant.* 57, (2019)

17. F. Holzschuher and R. Peinl, *J. Comput. Syst. Sci.* 82, 45 (2016)
18. S. La-Ongsri and J. F. Roddick, *Inf. Syst.* 52, 1 (2015)
19. F. K. Khattak, S. Jeblee, C. Pou-Prom, M. Abdalla, C. Meaney, and F. Rudzicz, *J. Biomed. Informatics X* 4, 100057 (2019)
20. L. Breiman Random Forests, *Machine Learning* 45 (2001)
21. Lai, S., Liu, K., He, S., Zhao, J., *IEEE Intelligent Systems*, 31,6 (2016)