SentiLog: Anomaly Detecting on Parallel File Systems via Log-based Sentiment Analysis

Di Zhang, Dong Dai
University of North Carolina at Charlotte
{dzhang16,ddai}@uncc.edu

Runzhou Han, Mai Zheng
Iowa State University
{hanrz,mai}@iastate.edu

ABSTRACT

As one core component of high-performance computing (HPC) platforms, parallel file systems (PFSes) grow quickly in scale and complexity, which makes them vulnerable to various failures or anomalies. Identifying PFS anomalies in runtime is thus critically helpful for HPC users and administrators. Analyzing runtime logs to detect the anomalies of large-scale systems has been proven effective in many recent studies. However, applying existing log analysis to PFSes faces significant challenges due to the large volume and irregularity of PFS logs. This study proposes SentiLog, a new approach to analyzing PFS logs for detecting anomalies. Unlike existing solutions, SentiLog works by training a general sentimental, natural language model based on the logging-relevant source code collected from a set of PFSes. In this way, SentiLog learns the implicit semantic information embedded in PFS by developers. Our preliminary results show that SentiLog can accurately predict anomalies and perform better than state-of-the-art log analysis solutions on two representative PFSes (i.e., Lustre and BeeGFS). To the best of our knowledge, this is the first work demonstrating that sentiment analysis could be a promising method to analyze complex and irregular system logs.

CCS CONCEPTS

• **Software and its engineering** → *Software maintenance tools*; • **Computing methodologies** → *Natural language processing*; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks.*

HotStorage '21, July 27–28, 2021, Virtual, USA

ACM ISBN 978-1-4503-8550-3/21/07…$15.00
https://doi.org/10.1145/3465332.3470873

KEYWORDS

Anomaly Detection, Parallel File System, Sentiment Analysis

ACM Reference Format:
Di Zhang, Dong Dai and Runzhou Han, Mai Zheng. 2021. SentiLog: Anomaly Detecting on Parallel File Systems via Log-based Sentiment Analysis. In *13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '21), July 27–28, 2021, Virtual, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3465332.3470873

1 INTRODUCTION

Analyzing runtime logs to automatically detect the anomalies in large-scale systems has been proven effective. An extensive number of methods have been proposed [15, 18, 19, 22, 23, 27, 35–38, 40, 44, 49–51], which can be roughly classified into three categories: (a) *rule-based* methods that rely on rules summarized by experts to detect the anomalies, such as LogLens [22] and PerfAugur [44]; (b) *supervised learning* methods that train models based on experts labeled abnormal or normal logs, such as DeepLog [23] and LogAnomaly [38]; and (c) *unsupervised learning* methods that rely on checking the system invariant or mining patterns of events sequences to detect the anomalies, such as PCA [49, 50] and IM [37].

Parallel file systems (PFSes), as a critical part of high-performance computing (HPC) platforms, are also subject to various bugs, failures, and anomalies [13, 19, 26]. Applying the previously mentioned methods to analyze their logs to identify runtime anomalies would be greatly helpful to HPC operators and administrators. However, PFS logs have their own unique features which challenge the existing solutions.

First, the volume of PFS logs is typically large. For example, Lustre file system generates logs on all MDS and OSS servers in milliseconds [14]. Such a huge volume makes it expensive (if not impossible) to label every log entry as normal or abnormal. Consequently, lacking of enough accurately labeled data may severely limit the effectiveness of *rule-based* methods or *supervised learning* methods.

Second, PFS logs are often generated via in-house logging mechanisms instead of standard logging libraries (e.g., Log4J [3] or SLF4J [5]), which makes them relatively irregular and arbitrary. In representative PFSes such as Lustre and BeeGFS, most log entries do not share any common identifier (ID) to denote their relevance. This is in sharp contrast

to other systems where global IDs often exist to denote a sequence of relevant operations. For instance, Hadoop HDFS has *block_id* [1], Apache HTTP server has *cache_key* [2], and Hadoop MapReduce has *task_id* [21]. Missing such global IDs makes it difficult to identify the matching events or to build sequences of operations. This limits the use of *unsupervised learning* methods which heavily rely on such information.

In this study, we propose SentiLog, a new log analysis method for detecting anomalies in PFSes. SentiLog consists of three key ideas. First, due to the lack of labeled runtime logs in PFSes, SentiLog directly analyzes the source code statements that print these logs (named as *logging statements* below). The logging statements are naturally labeled by their logging levels based on the corresponding logging macros used (e.g., `CERROR` or `CDEBUG` in Lustre). The logging statements are also widely seen in modern systems, providing a sufficient volume of training data [52]. These labeled logging statements enables us to conduct supervised learning.

Second, SentiLog focuses on the natural language aspect of the logging statements. The logging statements are written by the developers and read by other developers or system operators. Hence, they are close to human languages and likely to be modeled well using natural language processing (NLP) methods [17, 24]. In fact, SentiLog focuses on the sentimental context of log statements since developers describe abnormal and normal system behaviors using different tones. They typically use negative tones such as 'error' or 'exception' for anomalies, and use neutral or positive tones such as 'connection successes' for normal behaviors. Such sentimental difference is general across different systems and can be captured via sentiment analysis.

Third, SentiLog trains the sentiment model using multiple PFSes' source code to avoid being impacted by a single PFS. Developers might not always be rigorous about logging statements. For instance, as we have observed, they may accidentally use `Log_ERR` in source code to log an event which in fact is not related to any anomaly. Training the model based on multiple PFSes (i.e., software of the similar type) allows SentiLog to capture the developers' consensus in the community to avoid bias. Doing so also allows SentiLog to train a generic model, which may be applied to various PFSes including new or closed-source ones, as confirmed in our evaluation results.

To summarize, we propose SentiLog, a new log-based anomaly detection approach targeting HPC parallel file systems. SentiLog includes three main contributions. First, to the best of our knowledge, this is the first work to apply sentiment analysis on PFSes logs to detect anomalies effectively. Second, SentiLog uses the source code logging statements instead of runtime logs to train the model, which avoids human efforts in labeling logs. Third, SentiLog is designed as a complete workflow integrating with an automatic fault injection tool to conduct effective validation. We have implemented a prototype of SentiLog, examined its performance on two widely used parallel file systems (i.e., Lustre [4] and BeeGFS [6]), and compared it with state-of-the-art log analysis solutions. The preliminary results show that on both systems, SentiLog is able to accurately predict the anomalies: it achieves over 99% accuracy on Lustre and 92% accuracy on BeeGFS, performing better than existing solutions. These results show that SentiLog is promising for analyzing complex and irregular system logs.

2 BACKGROUND AND MOTIVATIONS

2.1 Parallel File System Logs

PFSes typically deploy their own logging mechanisms to generate logs. For instance, Lustre uses a set of logging macros (e.g., `CDEBUG`, `CERROR`) to log events in different severity levels [4], while BeeGFS implements its own logging classes (e.g., `log`) for the similar purpose [6]. Lack of using standard logging libraries (e.g., Log4J [3] or SLF4J [5]) makes PFSes logs highly irregular and diverse. To show this, we ran Drain [28], a tool to parse possible patterns of system logs, onto PFS (i.e., Lustre) and cloud file system (i.e., HDFS) logs. Drain identified 174 patterns from Lustre and 30 patterns from HDSF logs. Such diverse logs from PFSes challenge the existing log analysis solutions.

Lustre

```
00000100:00080000:0.0:1607448618.327577:0:2290:0:(recover.c:58:ptlrpc_initiate_recovery
()) lustre-OST0000_UUID: starting recovery
00000100:00080000:0.0:1607448618.327580:0:2290:0:(import.c:681:ptlrpc_connect_import())
ffffa139cab87800 lustre-OST0000_UUID: changing import state from DISCONN to CONNECTING
00000100:00080000:0.0:1607448618.327589:0:2290:0:(import.c:524:import_select_connection
()) lustre-OST0000-osc-MDT0000: connect to NID 10.0.0.8@tcp last attempt 4296114409
00000100:00080000:0.0:1607448618.327593:0:2290:0:(import.c:568:import_select_connection
()) lustre-OST0000-osc-MDT0000: tried all connections, increasing latency to 11s
```

HDFS

```
081109 203518 143 INFO dfs.DataNode$DataXceiver: Receiving block blk_-
1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010
081109 203518 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock:
/mnt/hadoop/mapred/system/job_200811092030_0001/job.jar. blk_-1608999687919862906
081109 203519 143 INFO dfs.DataNode$DataXceiver: Receiving block blk_-
1608999687919862906 src: /10.250.10.6:40524 dest: /10.250.10.6:50010
081109 203519 145 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_-
1608999687919862906 terminating
```

Figure 1: Two log snippets of Lustre and HDFS.

We further show two snippets of logs produced by two storage systems (Lustre and HDFS) in Figure 1. In HDFS logs, we can easily observe a unique block Id across multiple HDFS log entries (marked as red). Such a unique ID plays a critical role in log analysis, as it allows to connect multiple log events into a sequence of events for pattern detection. However, in Lustre and many other PFSes, such unique IDs do not exist. There are some Ids as marked in gray in the figure. But they represent the storage server ID, whose granularity is too coarse to be useful. Missing such global Ids challenges the existing solutions and motivate SentiLog.

2.2 NLP and Sentimental Analysis

Sentiment analysis is a classic NLP (natural language processing) method and has been widely applied in real-world problems [17, 24], such as product reviews or twitter analysis [31, 48]. Sentiment analysis can be formulated as a classification problem on datasets with inputs as the texts and outputs as one of three labels: positive, neutral, or negative. Supervised learning methods, such as Naive Bayes [33], Maximum Entropy [12] and Support Vector Machines (SVM) [41], are often used to build the classification model. Recently, deep learning-based methods were proven to be more accurate, including deep recurrent neural network (RNN)[32] and deep convolutional neural network[43]. Although sentiment analysis itself has been widely used in many real world problems, to the best of our knowledge, SentiLog is the first to apply it to PFSes log analysis. To motivate sentimental analysis on PFS logs, we show a log snippet of BeeGFS below. Intuitively, there are strong sentiment words (highlighted in red) which could be helpful for identifying the anomaly.

```
Dec14 22:54:24 Main [App] » Unable to create subdir: buddymir/inodes/C/60
Dec16 15:39:34 Main [MgmtdTargetStateStore.cpp:446] » Could not read states. node-
Type: beegfs-meta; Error: Path does not exist
```

SentiLog uses deep learning method, particularly, the Long short-term memory network (LSTM) [30] to build the model. To improve the accuracy, we actually used the bidirectional LSTM (BiLSTM)[25, 45] which incorporates a hidden layer to passe sequence information from backward to generate a better hidden representation of the sequence. More details about BiLSTM-based sentimental analysis can be seen at [11].

3 SENTILOG DESIGN

Figure 2 shows the workflow of SentiLog in three stages: logging statements collecting, log pre-processing, and sentimental model training. The trained model can later be applied to PFSes runtime logs to identify anomaly.

3.1 Logging Statements Collecting

SentiLog uses logging statements from multiple open-source PFSes to train the sentimental model. Hence our first step is to collect the needed statements from source code. SentiLog needs two inputs from users to do this work. The first one is a list of open-source PFSes and their git URLs. With these URLs, SentiLog pulls their latest source code periodically. The second input is a list of keywords for each PFS. We follow the official instructions on the logging mechanism of each PFS to identify the keywords first. If there is no such instruction, we manually find them from the source code (i.e., the logging macros). SentiLog uses these keywords to parse necessary logging statements out of the source code. Since different PFSes use different logging mechanisms, each PFS will have its own set of keywords. All the log statements parsed based on the same keyword should be assigned to one sentimental state (negative or neutral) accordingly. The sentimental state is determined by the logging level that the keyword indicates. For instance, the `CERROR` keyword in Lustre will be assigned as negative; while `CDEBUG` keyword will be assigned as neutral. Table 1 shows the keywords for each PFS. The outputs of this stage are the raw training datasets, each of which contains a logging statement and its label. These raw datasets will go through data pre-processing stage before being feed to the model trainer.

3.2 Log Pre-Processing

The raw logging statements may contain texts that are useless in sentiment analysis. For example, a logging statement of Lustre may look like this: "`Error %d invoking LNET debug log upcall %s %s;`". The format strings ('`%s`') are clearly not useful for the sentimental analysis as they will be later substituted by the actual strings. In NLP, text pre-processing is a necessary and important step to obtain consistent training results. In SentiLog, we conduct similar pre-processing on the log statements[16], which includes the following steps: 1) lowercasing all the texts; 2) stemming words to their root form (e.g., invoking → invoke); 3) removing the stopwords (e.g., 'this','that','and','a','we'); 4) normalizing a text into a standard form; 5) removing noises such as the format strings (e.g., %) and punctuation. Figure 2 shows the logging statements before and after the pre-processing. The goal is to make the logging statements closer to natural language, so that the sentimental analysis can be Girmore accurate and stable.

3.3 Sentimental Model Trainer

The pre-processed logging statements will be fed to the sentimental model trainer to train the model. As mentioned earlier, SentiLog used a bidirectional LSTM (BiLSTM) as the deep neural network to conduct sentimental analysis. The BiLSTM network contains two layers, 100 neurons in each layer, and in total has 789K parameters. The model is trained in batch size 64 and using Adam as optimizer with learning rate 0.01. Note that, since BiLSTM network takes word vectors as the inputs, we will need to tokenize each single word of the logging statements. We did this based on the pre-trained Glove Embedding [42]. The output of BiLSTM network is simply a 2 dimension vector indicating negative or neutral.

3.4 Anomaly Detection

The trained sentimental model can later be applied to the runtime logs generated by other PFSes for anomaly detection. The procedure is straightforward. The runtime logs

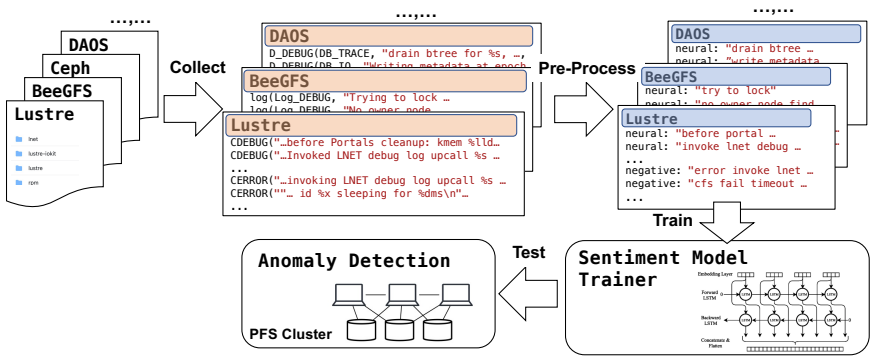

Figure 2: The overall workflow of SentiLog.

generated by a particular PFS will be collected in runtime; go through the data pre-processing stage described earlier; and then be tokenized as inputs of the BiLSTM neural network. The network will make a decision on whether the input log entry indicates indicates an anomaly or not depends on its sentimental prediction.

4 EVALUATIONS

4.1 Training Data Set

We selected a set of widely used open source PFSes to train SentiLog. These PFSes, as listed in Table 1, cover the majority HPC clusters in Top500 list [39]. We used the top four to train SentiLog and bottom two (marked in gray) to test the trained models (details about the testing data are in the next section). We briefly list the logging mechanisms used by each PFS and the total number of training samples collected from each PFS. Note that, `Debug` logs are labeled as neural and `Error` logs as negative to train SentiLog. Based on this data set, we train the SentiLog model using a computer with 4-Core 2.2 GHz Intel i7 CPU and 16 GB DRAM in 6 hours.

Table 1: Training Data Set

Log Source	Log Level		Log Mechanism
	Debug	Error	
OrangeFS [10]	1058	1202	`gossip_debug, gossip_err,...`
Ceph [7]	15459	2726	`dout, derr,...`
DAOS [9]	1549	3444	`D_DEBUG, D_ERROR,...`
GlusterFS [8]	2460	5260	`gf_msg, gf_log,...`
Lustre [4]	-	-	`CDEBUG, CERROR,...`
BeeGFS [6]	-	-	`log.log,...`

4.2 Testing Data Set

To test a trained SentiLog model, we need labeled runtime logs. However, to the best of our knowledge, there is no publicly-available labeled runtime logs for PFSes yet. In this study, we generated a small set of them for testing purpose. Specifically, we used an open-source fault injection tool called PFault [13] to inject faults to PFSes and record the generated logs. To label these logs, we simply consider logs generated before the faults as normal. For logs generated after fault injections, the domain experts will label each log entry as normal or abnormal depending on whether it is relevant with the injected faults. Specifically, our labelling criteria leverage standard Linux error numbers (or equivalent customized error numbers), since both Lustre and BeeGFS utilize them extensively in logging. Logs with a standard or equivalent error number are considered to be abnormal. In addition, we prune the potential noise by examining the log descriptions further. For example, logs related to transient network issues are exempted from abnormal logs since such transient issues are common in pre-fault logs as well.

PFault allows us to collect PFS logs under different faults models as shown in Table 2. They represent a wide range of failure scenarios that may occur to a PFS cluster in practice. Currently, PFault supports only Lustre and BeeGFS. So, our testing data is limited to these two PFSes. Our testing data was collected while running similar workloads on both PFSes. Based on our investigation on the underlying logging mechanism, the logging frequency of the two PFSes are different, which leads to different numbers of log entries. For Lustre, the numbers of normal and abnormal log entries are 150,473 and 7,401 respectively. For BeeGFS, the numbers are 119 and 39.

Table 2: Fault Models for Generating Failure Logs

Fault Model	Description
Whole Device Failure (a-DevFail)	a storage device becomes inaccessible entirely; caused by RAID controller failures, firmware bugs, etc.
Global Inconsistency (b-Inconsist)	local file systems on individual nodes are consistent; but the global PFS state across nodes is inconsistent
Network Partitioning (c-Network)	the PFS cluster splits into more than one "partitions" which cannot communicate with each other

To measure SentiLog, we use four metrics: *Accuracy*, *Precision*, *Recall*, and *F-Measure*. Here, *accuracy* is the percentage of correct predictions (both normal and abnormal) over all

predictions; *precision* measures how many percentages of the reported anomalies are actually anomalies; *Recall* measures how many percentage of actual anomalies are reported; *F-measure* is the harmonic mean of the precision and recall, which often indicates the quality of the model.

4.3 Preliminary Results

We conducted three sets of experiments to evaluate SentiLog. Although the results are still preliminary, we do observe the promising performance of SentiLog.

4.3.1 Comparing with Existing Solutions. We first compared SentiLog with DeepLog [23], a state-of-the-art log anomaly detection solution. DeepLog uses the sequences of normal log entries as input to train a LSTM model. It learns the sequence patterns and uses them to detect anomalies. We can not naively apply DeepLog to PFSes as PFSes logs do not present global Ids (such as *block_id* in HDFS) to determine log sequence. Similar to [29] and [20], we used a fixed window to generate the sequence of log entries in order to make it work on PFSes logs. Specifically, we sort logs based on their timestamp, choose a fixed window of 12 continuous logs to form a sequence, and move forward by one window each time to form the next sequence. All hyper-parameters are selected to be the same as DeepLog. We used the normal logs collected from Lustre and BeeGFS to train the model and applied it to the testing data collected in Section 4.2 to compare with SentiLog.

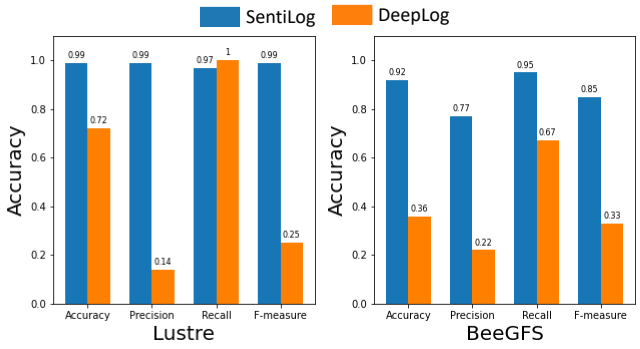

Figure 3: SentiLog vs. DeepLog.

Figure 3 reports the results. Here, we can observe that for all these metrics (accuracy, precision, recall and F-measure), SentiLog obtains better performance than DeepLog. As discussed in Section 2.1, lack of sequence info in PFSes logs makes DeepLog not suitable. Specifically, DeepLog detected 51,420 anomalies while only 7,390 of them are real anomalies, out of the total 7,401 anomalies. These numbers explain DeepLog's high recall but low precision performance, which simply indicates there are too many false positives. The results do show that SentiLog adapts to the irregular and diverse PFSes logs better, with an accuracy close to 99% on Lustre and 92% on BeeGFS.

It is worth noting that, these SentiLog models were not trained using Lustre nor BeeGFS source code, but still performed extremely well on them. This confirms the basis of SentiLog that sentimental context is general across similar systems, and SentiLog model is generic for PFSes.

4.3.2 Generality Evaluation. Previous comparison shows the generality of SentiLog. This experiment further evaluates where does the generality come from. SentiLog uses a set of different PFSes to train its model instead of using only the target PFS. This strategy leverages our assumptions that sentimental context is general and developers in the same community share the similar context during writing logs. To show this plays an important role in SentiLog, for each target PFS (Lustre or BeeGFS), we further trained SentiLog using only the target PFS (named as SentiLog-Self). We then compared the performance of SentiLog and SentiLog-Self models on the testing data. The results are shown in Figure 4.

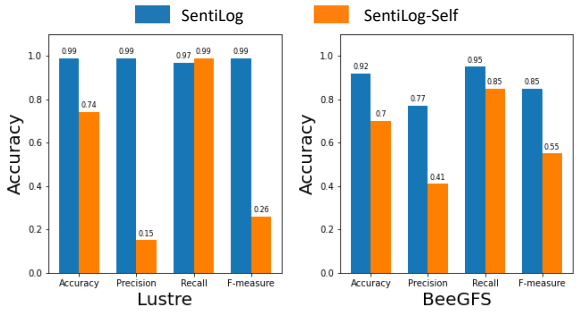

Figure 4: SentiLog vs. SentiLog-Self.

The results show SentiLog performs significantly better than SentiLog-Self on both Lustre and BeeGFS (F-measures of 0.99 vs 0.26 on Lustre and 0.85 vs 0.55 on BeeGFS). Although SentiLog-Self has slightly higher recalls on Lustre, it is at cost of low precision, which means more false alarms.

4.3.3 Comparing with Direct-Lookup. In this work, we conduct sentimental analysis on the logging-relevant source code. In fact, with the source code in hand, one might argue that there is a more straightforward way to determine the anomalies: for each runtime log, we may simply look up its corresponding logging statement in the source code and use its logging level to decide whether it is anomaly or not. We call this method *Direct-Lookup*. For comparison, we also implemented the Direct-Lookup method and compared it with SentiLog on Lustre and BeeGFS. As shown in Figure 5, SentiLog performs similar to Direct-Lookup on Lustre, but much better on BeeGFS (92% accuracy vs. 34%). We show a

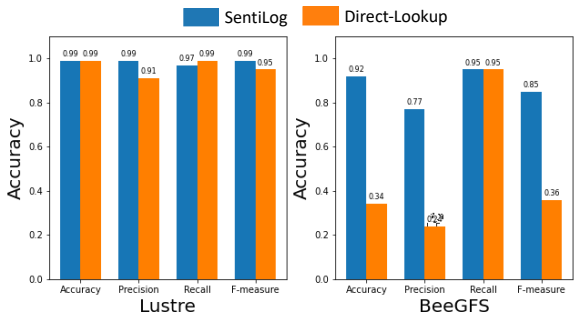

Figure 5: Performance of SentiLog vs. Direct-Lookup.

log snippet of BeeGFS below to explain why Direct-Lookup works poorly on BeeGFS:

```
Log_CRITICAL:Dec14 23:06:03 Main [App] » BeeGFS Helper Daemon Version: 7.2
Log_WARNING:Dec15 16:12:27 Main [App] » LocalNode: beegfs-mgmtd osboxes [ID:1]
Log_WARNING:Dec15 15:58:37 Worker1 [Node registration] » New node: beegfs-client 435-5FD9237D-osboxes [ID: 2]; Source: 10.0.0.121:59206
```

We can see that the three log entries above are simply reporting variable values, but they are labeled as 'Warning' or 'Critical' instead of normal by the developers. Previous study [34] actually suggested that such variable printing logs were reported as normal level in 95% of the time in multiple open-source software. BeeGFS seems to use inappropriate logging levels for these log entries, which makes Direct-Lookup ineffective.

In other words, the major drawback of Direct-Lookup is that it depends on a single PFS and is thus more sensitive to how developers write the logging statements. For example, developers may use error-level logs to record a network disconnection event, which actually does not indicate an anomaly because a following re-connection will resolve it. SentiLog, on the other hand, can tolerate such inexact logs from certain developers by training on a mix of multiple PFSes.

5 RELATED WORK

Using log analysis to detect system failures and anomalies has been extensively studied recently. Generally, they can be classified into following three categories.

Rule-based methods [18, 22, 27, 40, 44, 51] leverage expert-defined rules to assign unmatched log entries as anomalies. For example, LogLens [22] predefined a series of patterns of normal logs, and considered logs that do not match such patterns as anomalies. Rule-based methods rely on expert knowledge and regularity in the logs, both of which are not available in PFSes. SentiLog trains its model using PFSes source code, is applicable to PFSes logs.

Supervised learning methods rely on labeled logs to train the classification model. They can be further classified based on their learning algorithms. Traditionally, decision tree and SVM were used to train the classifier [15, 35]. Lately, deep learning-based solutions emerged. They mostly use LSTM or RNN networks such as DeepLog [23] and LogAnomaly [38]. The lack of enough labeled logs in PFSes basically rules out these methods. This also motivates SentiLog to use source code to train the model.

Unsupervised learning methods [36, 37, 49, 50] do not need labeled data. But they heavily rely on certain invariant or sequence across multiple log entries. For example, Invariant Mining [37] tracks invariant across multiple log events to detect system behaviors (e.g., the counts of *open-file* should match the counts of *close-file*). PFSes logs are highly irregular, typically do not present such sequences or invariant. Hence, existing solutions can not be applied. SentiLog focuses on single log entry and builds sentimental model to analyze it.

Some recent studies [46, 47] also apply sentiment analysis in log-based anomaly detection. However, they perform both training and testing directly on the runtime logs which needs extensive efforts to label the logs. SentiLog avoids the limitation by training the model directly using the source code.

6 CONCLUSION AND FUTURE WORK

This study presents SentiLog, a new log-based anomaly detection approach for parallel file systems. SentiLog works by training a generic sentimental model using the logging-relevant source code of multiple PFSes. Our preliminary results show that it works on two representative PFSes where existing solutions cannot be applied. This work suggests many promising directions for further improvements. For example, we plan to a) compare SentiLog with more complicated baseline, such as combining rule-based and supervised learning methods; b) investigate the potential impact of PFS versions on the results; c) explore the possibility to consider more features besides the log statement description; d) conduct more experiments to validate and quantify the generic sentiment across different software; e) extend SentiLog to other systems with complex and irregular logs. We hope to develop SentiLog into a fully-fledged framework to help improve large-scale storage systems in general.

ACKNOWLEDGMENT

We thank the anonymous reviewers and Youyou Lu (our shepherd) for their insightful feedback. This work was supported in part by NSF under grants CCF-1910727, CCF-1910747, CNS-1852815, and CCF-1853714/1717630. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the view of NSF.

REFERENCES

[1] Accessed: 03/2021. Apache HDFS. https://hadoop.apache.org.
[2] Accessed: 03/2021. Apache Http Server. https://httpd.apache.org.
[3] Accessed: 03/2021. Log4J. https://logging.apache.org/log4j/2.x/.
[4] Accessed: 03/2021. Lustre. https://www.lustre.org.
[5] Accessed: 03/2021. SLF4J. http://www.slf4j.org.
[6] Accessed: 04/2021. BeeGFS. http://beegfs.io.
[7] Accessed: 04/2021. CephFS. https://ceph.io/ceph-storage/file-system/.
[8] Accessed: 04/2021. GlusterFS. https://www.gluster.org.
[9] Accessed: 04/2021. Intel DAOS. https://daos-stack.github.io.
[10] Accessed: 04/2021. OrangeFS. http://www.orangefs.org.
[11] Zhang Aston, C. Lipton Zack, Li Mu, and J. Smola Alex. 2021. Dive into Deep Learning. https://d2l.ai.
[12] Adam Berger, Stephen A Della Pietra, and Vincent J Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational linguistics* 22, 1 (1996), 39–71.
[13] Jinrui Cao, Om Rameshwar Gatla, Mai Zheng, Dong Dai, Vidya Eswarappa, Yan Mu, and Yong Chen. 2018. PFault: A General Framework for Analyzing the Reliability of High-Performance Parallel File Systems *(ICS '18)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3205289.3205302
[14] Jinrui Cao, Simeng Wang, Dong Dai, Mai Zheng, and Yong Chen. 2016. A generic framework for testing parallel file systems. In *2016 1st Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS)*. IEEE, 49–54.
[15] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE, 36–43.
[16] Tse-Hsun Chen, Stephen W Thomas, and Ahmed E Hassan. 2016. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering* 21, 5 (2016), 1843–1919.
[17] Gobinda G Chowdhury. 2003. Natural language processing. *Annual review of information science and technology* 37, 1 (2003), 51–89.
[18] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. 2012. Event logs for the analysis of software failures: A rule-based approach. *IEEE Transactions on Software Engineering* 39, 6 (2012), 806–821.
[19] Anwesha Das, Frank Mueller, Paul Hargrove, Eric Roman, and Scott Baden. 2018. Doomsday: Predicting which node will fail when on supercomputers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 108–121.
[20] Anwesha Das, Frank Mueller, Charles Siegel, and Abhinav Vishnu. 2018. Desh: deep learning for system health prediction of lead times to failure in hpc. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. 40–51.
[21] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
[22] Biplob Debnath, Mohiuddin Solaimani, Muhammad Ali Gulzar Gulzar, Nipun Arora, Cristian Lumezanu, Jianwu Xu, Bo Zong, Hui Zhang, Guofei Jiang, and Latifur Khan. 2018. Loglens: A real-time log analysis system. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1052–1062.
[23] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1285–1298.
[24] Ronen Feldman. 2013. Techniques and applications for sentiment analysis. *Commun. ACM* 56, 4 (2013), 82–89.
[25] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks* 18, 5-6 (2005), 602–610.
[26] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. 2017. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.
[27] Stephen E Hansen and E Todd Atkins. 1993. Automated System Monitoring and Notification with Swatch.. In *LISA*, Vol. 93. 145–152.
[28] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 33–40.
[29] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 207–218.
[30] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[31] Clayton Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 8.
[32] Ozan Irsoy and Claire Cardie. 2014. Opinion mining with deep recurrent neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 720–728.
[33] David D Lewis. 1998. Naive (Bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*. Springer, 4–15.
[34] Zhenhao Li, Heng Li, Tse-Hsun Peter Chen, and Weiyi Shang. 2021. DeepLV: Suggesting Log Levels Using Ordinal Based Neural Networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1461–1472.
[35] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure prediction in ibm bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 583–588.
[36] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 102–111.
[37] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection.. In *USENIX Annual Technical Conference*. 1–14.
[38] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs.. In *IJCAI*, Vol. 7. 4739–4745.
[39] Hans Meuer, Erich Strohmaier, Jack Dongarra, and Horst Simon. 2001. Top500 supercomputer sites.
[40] Alina Oprea, Zhou Li, Ting-Fang Yen, Sang H Chin, and Sumayah Alrwais. 2015. Detection of early-stage enterprise infection by mining large-scale log data. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 45–56.
[41] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment classification using machine learning techniques. *arXiv preprint cs/0205070* (2002).
[42] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.
[43] Soujanya Poria, Erik Cambria, and Alexander Gelbukh. 2016. Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based Systems* 108 (2016), 42–49.
[44] Sudip Roy, Arnd Christian König, Igor Dvorkin, and Manish Kumar. 2015. Perfaugur: Robust diagnostics for performance anomalies in cloud services. In *2015 IEEE 31st International Conference on Data*

Engineering. IEEE, 1167–1178.

[45] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[46] Hudan Studiawan, Ferdous Sohel, and Christian Payne. 2020. Anomaly detection in operating system logs with deep Learning-based sentiment analysis. *IEEE Transactions on Dependable and Secure Computing* (2020).

[47] Hudan Studiawan, Ferdous Sohel, and Christian Payne. 2020. Sentiment analysis in a forensic timeline with deep learning. *IEEE Access* 8 (2020), 60664–60675.

[48] Ye Wu and Fuji Ren. 2011. Learning sentimental influence in twitter. In *2011 International Conference on Future Computer Sciences and Application*. IEEE, 119–122.

[49] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Online system problem detection by mining patterns of console logs. In *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 588–597.

[50] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 117–132.

[51] Kenji Yamanishi and Yuko Maruyama. 2005. Dynamic syslog mining for network failure monitoring. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 499–508.

[52] Jieming Zhu, Pinjia He, Qiang Fu, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2015. Learning to log: Helping developers make informed logging decisions. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 415–425.