Sentiment Analysis based Error Detection for Large-Scale Systems

Khalid Ayedh Alharthi*¶, Arshad Jhumka*, Sheng Di†, Franck Cappello†‡, Edward Chuah§,
*University of Warwick, Coventry, UK
†Argonne National Laboratory, Lemont, IL, USA
‡ University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, USA
§ Lancaster University, Bailrigg, City of Lancaster, UK
¶ University of Bisha, Bisha, KSA
{Khalid.Alharthi, H.A.Jhumka}@warwick.ac.uk, sdi@anl.gov, cappello@mcs.anl.gov, e.t.chuah@lancaster.ac.uk

***Abstract*—Today's large-scale systems such as High Performance Computing (HPC) Systems are designed/utilized towards exascale computing, inevitably decreasing its reliability due to the increasing design complexity. HPC systems conduct extensive logging of their execution behaviour. In this paper, we leverage the inherent meaning behind the log messages and propose a novel sentiment analysis-based approach for the error detection in large-scale systems, by automatically mining the sentiments in the log messages. Our contributions are four-fold. (1) We develop a machine learning (ML) based approach to automatically build a sentiment lexicon, based on the system log message templates. (2) Using the sentiment lexicon, we develop an algorithm to detect system errors. (3) We develop an algorithm to identify the nodes and components with erroneous behaviors, based on sentiment polarity scores. (4) We evaluate our solution vs. other state-of-the-art machine/deep learning algorithms based on three representative supercomputers' system logs. Experiments show that our error detection algorithm can identify error messages with an average MCC score and f-score of 91% and 96% respectively, while state of the art ML/deep learning model (LSTM) obtains only 67% and 84%. To the best of our knowledge, this is the first work leveraging the sentiments embedded in log entries of large-scale systems for system health analysis.**

***Index Terms*—Sentiment analysis lexicon, large-scale systems, Stochastic Gradient Descent, logistic regression, error detection**

I. Introduction

Failure log analysis of HPC systems, such as supercomputers, is attracting more and more researchers from academia and industry in order to improve their reliability. These systems, consisting of sophisticated hardware and software, often fail due to their scale and design complexity. The components of these systems, such as OS and parallel file systems, generally generate masses of valuable log messages that are critical for system administrators to assess the state of the system [1]. These message or event logs are considered the first source about the system's health state for administrators, because it contains rich information about normal behavior (i.e., informational messages) or abnormal behavior (i.e., error messages) of various system components.

Failure-directed analysis, such as error detection, using these system logs has been extensively studied for years. In [2], the authors performed error detection in supercomputers by combining entropy, mutual information, and PCA approaches. Several recent works were generally based on detecting system anomalies [3]–[5]). Further techniques based on natural language processing (NLP) and artificial intelligence (AI) have also been applied towards failure log analysis [6]–[8].

Works on error detection in HPC systems have focused on various aspects such as identification (i) of erroneous log entries [9], (ii) of failure-inducing erroneous execution sequence [2], [10], [11] and (iii) of detecting quantitative relationship among logs [12], [13]. This paper addresses the first problem and focuses on the automated classification of failure log entries, thereby obviating the need for the time-consuming manual labelling of such entries. Often, automated labeling is achieved using rule-based techniques. However, such rule-based approaches are often inflexible as the rules need to be revised when new types of log entries are generated.

In this paper, we explore a novel perspective on the problem of failure log analysis. We conjecture that log messages often encapsulate the sentiment of system developers, which pertains to the perceived health of the system. For example, if a typical log entry states a timeout issue on a particular node, this message implies that there is something wrong at the network level. That is, such a message indicates that the system developer has a negative sentiment about the current network status. The question is: can we leverage these sentiments to develop a sentiment analysis-based technique for failure log analysis? Our answer is yes, as we propose two novel algorithms for detecting error messages and faulty nodes and components in these systems. This is particularly helpful for error detection in systems with non-labelled logs (such as Ranger and Lonestar4, to be detailed later).

Sentiment analysis, which is a text classification technique that combines NLP and AI, is based on assigning weighted sentiment scores to the text entities within a word, phrase, sentence, or document. One class of approaches for sentiment analysis makes use of a sentiment lexicon where the focus is on developing specific list of words that carry cues of affection or sentiment, instead of using every word as a feature [14]. However, the development process of the sentiment lexicon has some weaknesses: (i) it is often generated manually, which is tedious and inaccurate to users; (ii) it tends to be domain-specific for efficiency reasons. We address these two

respective problems as follows: (i) we develop a machine-learning approach that exploits the log sentiments to develop a sentiment lexicon to support the detection of errors in large systems[1] and identify the erroneous components or nodes and (ii) based on our observation that such HPC systems often share similar components such as OS, a lexicon for one system (i.e., source system) can be reused for another target system.

The fundamental principle of our design is that the sentiment intensity scores can accurately represent the system state, among which the system developers generally use very similar concepts or terms to record the events/messages across different systems. In fact, the system developers often use negative sentiments to log serious problems such as errors and failures, neutral sentiments to highlight informational messages indicating the system works as expected, and positive sentiments to mark the system faults/problems that have been fixed, which inherently captures the three main classes (sentiment polarities) of a log message.

However, to detect errors, sentiment polarity is not adequate. A sentiment intensity score keeps track of the strength of a sentiment, e.g., the features 'failed' and 'unexpected' may be associated with higher negative scores than the features 'slow' and 'monitor', while the features 'recovered' and 'successfully' are assigned higher positive scores than 'normal' and 'valid' states. This potentially allows us to exploit system logs of a source system that are labelled with severity levels (e.g., Blue Gene systems) and extract their sentiment features to automatically label log entries of other unlabelled (target) systems as an unsupervised approach.

We make four main research contributions:

1) We develop a ML-based method using stochastic gradient descent logistic regression, to automatically construct a reusable sentiment lexicon for such systems.
2) We develop an algorithm for error detection based on the sentiment intensity score of log messages.
3) We develop an algorithm to discover system components (e.g., nodes) which show erroneous behaviors based on sentiment polarity scores of messages logged by those components during a specific time period.
4) We perform the evaluation using the system logs of three large systems: (i) Blue Gene/Q Mira, (ii) Ranger and (iii) Lonestar4 which were built by three different vendors - IBM, Sun, and Dell respectively. We also compare our sentiment lexicon's performance with ML/deep learning classification algorithms, including Long Short Term Memory (LSTM), Random Forest (RT), Extreme Gradient Boosting (XGBoost), Multinomial Naive Bayes (Multinomial NB), and K-Nearest Neighbor (KNN). Experiments show that our sentiment-based solution can efficiently detect error messages based on their associated sentiment scores, with an average MCC-score of 91% and an average f-score of 96%, whereas state-of-art ML/deep learning model (LSTM) obtains only 67% and 84% respectively. Our technique identifies error messages with an f-score of about 99% for Blue Gene/Q Mira system. Using the sentiment lexicon items extracted based solely on the Blue Gene systems logs, a majority of errors in Ranger and Lonestar4 logs can be successfully detected, with f-scores of about 94%, and 95%, respectively. This effectively shows that our technique can generate reusable lexicon for such systems, enabling the automated labelling of any system's logs.

The remainder of this paper is organized as follows: Section II presents the background for the system/fault model & log data, and Section III formulates the research problem. Sections IV$\sim$VI present the main steps of our approach. Section VII shows the results of our evaluation performed on the logs of three large systems. Section VIII discusses related work, and we conclude the paper in Section IX.

II. System/Fault Model and Log Data

In this section, we describe the targeted system model, fault model, the supercomputers studied in our work as well as their system logs used in our experiments.

A. System Model

We present a generic system model of HPC cluster systems. A cluster system consists of a set of nodes $N_1, \ldots, N_m$ to execute a set of jobs $J_1, \ldots, J_n$ over a set of production time-slots $T_1, \ldots, T_p$. To support these activities, components such as a resource scheduler and a set of software components, such as a file system and an operating system, are needed. The nodes and production slots on the HPC system are allocated to a job by the resource scheduler. Data, as input or output, may be transferred to and from the file system by each node or job. As the software components execute, they output log messages and resource usage data which may be written to containers.

B. Fault Model

We assume that various discrete fault models may be considered, depending on the abstraction level. One may consider faults occurring at the node level, the file system level, or an aggregate cluster level. When a fault occurs, the resulting error leads to the output of an error message in the system log file. If the error is not adequately handled, a failure can occur which will also be logged. We assume that faults can occur in any component at any level within the HPC system.

We now describe the production clusters along with their logs used in our research, which is the fundamental information of our following sections (problem formulation, methodology, etc.).

C. Blue Gene/Q Mira Cluster and RAS Logs

The Mira supercomputer used to be one of the most powerful supercomputers in the world, and its comprehensive system logs (including RAS log, job scheduling log, I/O behavior log, etc.) have been released to the public to promote the understanding of extreme-scale systems. Mira consists of 48

[1]A system fault refers to a potential event that may adversely affect the system execution.

racks, each containing two midplanes. Every midplane has 32 compute nodes, each being facilitated with 16 active cores on a PowerPC A2 1600 MHz processor and a total of 16G DDR3 memory. As such, the entire Mira system has a total of 786,432 cores and 786,432 GB of memory.

In the Mira cluster with diverse system logs, the Reliability, Availability, and Serviceability (RAS) log is our focus. The event in the RAS log is identified by one of three severity levels (INFO, WARN, or FATAL). Although one message in the RAS log consists of 14 fields, we focus only on a few of them related to the system reliability, such as MESSAGE, MSG_ID, LOCATION, SEVERITY, and EVENT_TIME, as suggested by [15]–[17].

D. Ranger and Lonestar4 Clusters and System Logs

Here, we describe the Ranger and Lonestar4 cluster systems operated by the Texas Advanced Computing Center (TACC) [18]. Ranger was a Linux-based high-performance computing (HPC) system that consisted of 4,048 nodes. It was operated from 2007 to 2013. A high-speed Infiniband network provided communication among all the nodes. A job scheduler provided job scheduling and resource management services. The Lustre file system provided high-speed file access to the users of Ranger. Ranger is the first HPC system at a United States academic institution that deployed Rationalized logs [19].

Lonestar4 was a Linux-based HPC system that consisted of 1,888 nodes. It was operated from 2009 to 2015. As was described in the description of Ranger, a high-speed Infiniband network provided communication between all the nodes on Lonestar4, a job scheduler provided job scheduling and resource management services, and the Lustre file system provided high-speed data I/O.

Next, we give an example of a system log:

```
Apr 4 15:58:38 012324 mds5 kernel: LustreError
: 138-a: work-MDT0000: A client on nid .*.*
.5@o2ib was evicted due to a lock blocking
callback to .*.*.5@o2ib timed out: rc
```

The system logs generated on Ranger and Lonestar4 do not contain any severity level tags such as FATAL, FAILURE, ERROR and INFO. The fields in the system log are: (i) timestamp (`Apr 4 15:58:38`), (ii) job number (`012324`), (iii) node number (`mds5`), (iv) system software component (`kernel`) and (v) message (`A client on nid ...`). The message is a client eviction due to a lock blocking callback. We focus on the message field, as this will allow for the classification of the log as faulty or non-faulty. We collected 2 months worth of system logs on Ranger and 2 months worth of system logs on Lonestar4. We removed all the repeated messages to obtain a set of unique messages.

In the Ranger system logs, we identified: (i) 1,513 unique messages in June 2011 (reduced from 10,021,516) and (ii) 1,676 unique messages in July 2011 (reduced from 64,822,682). In the Lonestar4 system logs, we identified: (i) 2,804 unique messages in February 2013 (reduced from 8,993,154) and (ii) 2,576 unique messages in March 2013 (reduced from 12,267,629).

III. Problem Formulation

We formulate our research problem based on the three aforementioned supercomputers, including Blue Gene/Q Mira (involving 49K nodes with a total of 786K cores), TACC Ranger (involving 4K nodes) and Lonestar4 (about 2K nodes with up to 63K cores), which were built by three different vendors - IBM, Sun, and Dell respectively.

Without loss of generality, the general system model for a large-scale system (e.g. IBM Blue Gene) contains a set N of nodes, a queue of J jobs, a set T of production times, a job scheduler JS, and various software components such as a file system. The scheduler JS allocates the J jobs to the N nodes to execute during time period T. Further, the components write message logs in to a central writing container [18].

The problem that our approach addresses can be formulated as follows: Assume (i) a set of log messages is generated by a large-scale system, (ii) these log messages have different severity levels, and (iii) the message templates comprise system developers' sentiments (either negative, neutral, or positive), our objective is to develop an efficient approach that is able to:

1) Automatically construct a reusable sentiment lexicon intended for large-scale systems.
2) Using the lexicon, identify system individual faults.
3) Using the lexicon, develop an unsupervised approach to detect faults of other (target) systems that are missing the severity attribute information.
4) Using the generated lexicon, identify the erroneous nodes and components to assist precaution (e.g., avoid node crash), thereby preventing job failures.

IV. Lexicon Construction Using Stochastic Gradient Descent Logistic Regression

In this section, we describe a machine learning based model (namely **SGDLRSL**) that allows for the automatic construction of sentiment lexicons to detect errors in large-scale systems via the stochastic gradient descent logistic regression technique. A sentiment lexicon is a dictionary that consists of features (N-gram) associated with their sentiment polarity values, and these sentiment scores are estimated based on a model trained on a sample of log message templates. Stochastic gradient descent logistic regression was employed since it is a discriminative model which assigns high weights (sentiment scores) to the significant log message features that can distinguish error messages from non-error messages. To generate a sentiment lexicon for a large-scale system, our model requires four components:

- M input/label pairs of log message templates (x^i,y^i) where each input log message template x^i is represented by a vector of f^j features $[f^1, ..., f^j]$.
- the sigmoid (logistic) function to compute the estimated class $\bar{y} = \sigma(w.x + b)$ for each log message template.
- the cross-entropy loss function for features weights (i.e.,coefficients) learning through minimizing error on training log messages.

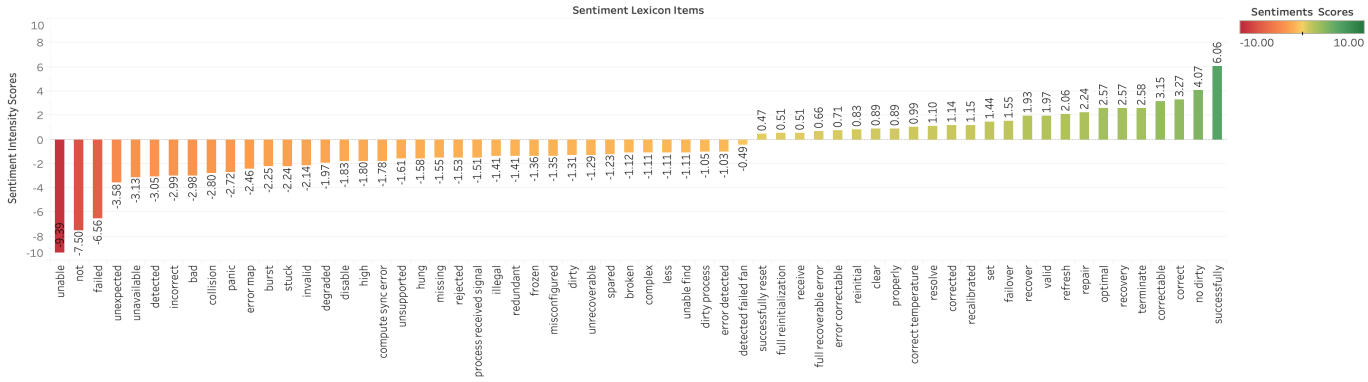

Fig. 1. $\sim$ 60 of IBM Gene systems lexicon' items associated with their sentiment intensity scores

- the stochastic gradient descent algorithm for optimizing the cross-entropy loss function and updating weights.

The SGDLRSL model performs two main steps to learn sentiment lexicon items (features and their weights).

A. Phase I. Log Message Template Preprocessing

In the first step, we use M representative training log message templates (x^i,y^i) that are labelled as either $[-1, 1]$ faulty or non-faulty messages where the numbers of erroneous and non-erroneous messages are balanced. We can use severity levels as log message template labels for those large-scale systems with this feature in their log data. The following preprocessing steps are conducted on the dataset with log message templates:

- Divide log message templates into tokens such as strings, variables, and punctuation.
- Remove all alphanumeric words, punctuation, stop words, variables that are not strings from log messages and use other NLP methods to clean text, such as lowercasing all texts, etc.

B. Phase II. Sentiment Lexicon Learning

We present the sentiment lexicon learning pseudo-code in Algorithm 1. Basically, the stochastic gradient descent logistic regression (SGDLR) technique obtains sentiment scores (a vector of weights) of lexicon elements by learning from the log message templates training set. Each weight w^j(used later as a sentiment score) is a real number ($\in \mathbb{R}$) and is linked with one of the log message features f^j. The weight w^j signifies how important the log message feature is to classifying a faulty log message from a non-faulty log message. Without loss of generality, we assume that a high positive weight indicates a message with a normal state or correctable error, and a very negative weight implies that the message is a failure or non-correctable error. This machine learning technique automatically computes the scores of lexicon items as follows:

Step 1. We employ the Term Frequency-Inverse Document Frequency (TF–IDF) representation technique [20] to extract n-gram features f^j from log message templates x^i and convert these features to numerical vectors $\in R^{|L|}$, where lexicon L is a set of n-gram features. The TF–IDF value of each feature is calculated by multiplying two metrics: Term Frequency

Algorithm 1: SGDLR to construct a sentiment lexicon for large-scale systems

Input: M log message templates (x, y),Logistic regression $h()$,Loss function $L()$, learning rate η , regularization parameter λ
Output: Log features f with their weights W
initial W, $\mathbf{b}$, $\eta \leftarrow 0.01$, λ;
for *each* $(x) \in M$ **do**
  1) Tokenization
  2) Removal of alphanumeric words, variables, etc.
  3) Convert into TF-IDF representation format via Formula (1).
end
repeat
  for $(x, y) \in M$ *randomly* **do**
    1) Compute $\bar{y} \leftarrow \frac{1}{1+e^{-\theta^T x}}$
    2) Compute the loss by Formula (5).
    3) Compute the gradient $g \leftarrow \eta \nabla L(h(x; \theta), y)$
    4) Update weights and bias $\theta_{t+1} \leftarrow \theta_t - \eta \nabla L(h(x; \theta), y)$
  end
until *SGD Converged*;
return (f, W)

$tf(f^j, x^i)$ and Inverse Document Frequency $idf(m_{f^j}, M)$ as follows:

$$tfidf(f^j, x^i, M) = tf(f^j, x^i) \times idf(m_{f^j}, M) \quad (1)$$

In TF-IDF, the TF part measures how frequently a feature f^j occurs in a log message template x^i and is defined as follows

$$tf_{(f^j, x^i)} = \frac{n_{f^j, x^i}}{\sum_k n_{f^k, x^i}} \quad (2)$$

where n_{f^j, x^i} is the total number of feature f^j occurrences in a log message template x^i divided by the total number of features $\sum_k n_{f^k, x^i}$ in that message template. The IDF part, $idf(f^j, M)$, measures how important a feature f^j is in all message templates M by taking the logarithm of the ratio of the total number of log message templates M to the total number of log message templates $m_{f^j} \leq M$ containing the feature f^j plus 1, to prevent dividing by zero, as follows:

$$idf(m_{f^j}, M) = \log\left(\frac{M}{m_{f^j} + 1}\right) \quad (3)$$

We refer the readers to [20] for more details.

Step 2. We use SGDLR to train our model on log message features to extract the dense weights vector W as follows:

1) Compute the estimated class $\bar{y} = \sigma(w.x+b)$ for each log message template via the following logistic regression function:

$$h(x) = \frac{1}{1 + e^{-\theta^T x}} \quad (4)$$

where θ includes two types of parameters: features' **weights** W and **bias** b (we neglect this parameter).

2) Then, we use the cross-entropy function and L1 regularization (Formula (5)) to compute the loss $L(\bar{y}, y)$ in order to measure how close $\bar{y}$ is to the actual label y.

$$L(\bar{y}, y) = -[ylog\bar{y} + (1-y)log(1-\bar{y})] + \lambda|w| \quad (5)$$

The weights W of log message features and bias b are learned from labelled log messages training set through a loss function that must be minimized to make $\bar{y}$ for each log message as close as possible to the actual output y. L1 regularization (i.e., Lasso Regression) $\lambda|w|$, where parameter $\lambda > 0$, is added to the cost function to prevent the overfitting problem and improving model generalization by penalizing weights.

3) Minimize the loss function in our model via the stochastic gradient descent technique (Formula (6)) to obtain the optimal weights W of log message features.

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y) \quad (6)$$

Stochastic gradient descent is a technique which is used to minimize the loss function by calculating its gradient after each mini-batch of log messages and updating the vector's parameters values θ (weights W and bias b). Since the loss function for logistic regression is convex, the SGD will reach the minimum of a lost function.

Step 3. The dense weight vector ***W*** is used as the sentiment scores with their associated log message features ***f*** as our lexicon items. For systematic observation, we normalize sentiment scores by dividing them by a uniform coefficient.

V. Sentiment Polarity-based Error Detection

In this section, we present a novel error message detection algorithm, i.e., log message labelling algorithm, exploiting the sentiment lexicon, which is constructed using stochastic gradient descent logistic regression. We present the approach in Algorithm 2.

The algorithm includes three phases, which are described as follows.

1) Phase I: Log Parsing: This phase is similar to the log message preprocessing phase presented in Section IV, where the opensource toolkit LogAider [15] and NLP techniques are employed to preprocess and clean the system logs. For example, all duplicated events with spatial or temporal correlations are filtered out by LogAider.

2) Phase II: Log Message Sentiment Scores Assignment: In this phase, each log message x_i is associated with a sentimental polarity score using an assignment method similar to NLP's lexicon techniques (e.g., Vader [21]), in which the lexicon learned from the SGDLRSL model is used to assign each log message a sentiment score made up of three different scores: positive, negative, and the overall log sentiment score, as follows:

$$x_{i\textbf{posScore}} = \frac{\sum_i w_i}{\sum_i f_i} \; ; \quad w_i > \mathbf{0} \quad (7)$$

$$x_{i\textbf{negScore}} = \frac{\sum_i w_i}{\sum_i f_i} \; ; \quad w_i < \mathbf{0} \quad (8)$$

$$x_{i\textbf{SentiScore}} = x_{i\textbf{posScore}} + x_{i\textbf{negScore}} \quad (9)$$

In general, the faulty messages contain the system developers' negative sentiments expressing concern about unexpected system operations, unusual situations, serious problems, failed services, and corruption. Consequently, the $x_{i\textbf{negScore}}$ is calculated by summing the negative valence scores $w_i < 0$ for each feature f_i of a log message that matches the sentiment lexicon features divided by their total number; the intensity of this score lies between 0 (neutral) and -1 (extremely negative). Moreover, log messages are generally embedded with more negative sentiments than positive or neutral ones, as the system developers tend to be more interested in abnormal systems' events. On the other hand, non-faulty messages include neutral sentiments indicating normal system behaviors and progress of system software (e.g., service started or stopped); positive sentiments indicate the system issues that are resolved (e.g., corrected errors, recovered failures). Therefore, the $x_{i\textbf{posScore}}$ is calculated by summing the positive valence scores $w_i > 0$ for each feature f_i of a log message that matches the sentiment lexicon features divided by their total number; the intensity of this score lies between 0 (neutral) and +1 (extremely positive). The $x_{i\textbf{SentiScore}}$ is the overall log polarity score that combines the $x_{i\textbf{negScore}}$ and $x_{i\textbf{posScore}}$. In other words, by summing the valence scores (i.e., whether it is positive or negative) for each feature f_i of a log message that matches the sentiment lexicon features divided by their total number. This score lies between -1 (extremely negative) and +1 (extremely positive).

3) Phase III: Detection Phase: Once system log messages are associated with sentiment polarity scores, the $x_{i\textbf{SentiScore}}$, detection threshold φ, and absolute lexicon threshold μ are used to detect whether these messages are faulty or non-faulty. A log message is classified as faulty when $x_{i\textbf{SentiScore}} < \varphi$

Algorithm 2: Error detection algorithm based on sentiment lexicon

Input: Unlabelled messages logs $(x_1, \cdots, x_n)$, Sentiment Lexicon Items(f_i, w_i),Absolute lexicon threshold μ = μ, detection threshold φ = φ
Output: $(x_{1\textbf{posScore}}, \cdots, x_{n\textbf{posScore}})$, $(x_{1\textbf{negScore}}, \cdots, x_{n\textbf{negScore}})$, $(x_{1\textbf{SentiScore}}, \cdots, x_{n\textbf{SentiScore}})$, **Labels**$(y_i, \cdots, y_n)$
Parsing messages logs $(x_1, \cdots, x_n)$
for *i = 1 to n* **do**
  $x_{i\textbf{posScore}} \leftarrow \frac{\sum_i w_i}{\sum_i f_i}$; $w_i > \mathbf{0}$
  $x_{i\textbf{negScore}} \leftarrow \frac{\sum_i w_i}{\sum_i f_i}$; $w_i < \mathbf{0}$
  $x_{i\textbf{SentiScore}} = x_{i\textbf{posScore}} + x_{i\textbf{negScore}}$
  if $x_{i\textit{SentiScore}} < \varphi$ **then**
    $y_i \leftarrow$ faulty;
  else
    $y_i \leftarrow$ non-faulty;
  end
end

and as non-faulty otherwise. We can refine the φ threshold until we achieve an optimal value which results in a satisfied classification, however, in our analysis, we note that $\varphi = 0$ is a near-optimal setting. Furthermore, the absolute lexicon threshold μ can be adjusted until satisfactory classification accuracy with fewer lexicon features is achieved.

VI. Erroneous Component Identification Based on Sentiment Polarity Scores

We can use our learned sentiment lexicon to calculate the sentiment polarity scores of the log messages for a certain time window t over some period (e.g., one hour time window over one day) for identifying the problematic components (e.g., nodes). The idea is that based on the sentiment scores, the system administrators can forecast which components may have erroneous behaviors, such that the jobs involved can be reassigned to other backup resources, and the problematic components would be temporally isolated until their problems are fixed. Specifically, the components' sentiment scores are anticipated to be neutral when they work as expected by logging informational messages or logging nothing. The negative scores are anticipated to associate with some components experiencing errors, especially when these abnormal states last for multiple consecutive time windows. However, when the components' issues have been resolved and they start to log the recovery and correction messages, their sentiment scores are expected to be increased or set to positive, indicating that they have been recovered well.

Our approach is composed of our sentiment lexicon that was learned in the previous section IV, system's components $(C_i, ..., C_k)$, log messages $(x_1, ..., x_n)$, time window t, where the system developers define the start time t_S and end time t_E , and a detection sentiment score φ. The phase of erroneous component identification is similar to that of detecting error message logs; however, the components' associated sentiment scores are calculated within a certain time window specified by the systems' administrators. We present the pseudo-code of erroneous component identification algorithm based on a specific window time $[t_s, t_e]$ in Algorithm 3.

1) Phase I: Component's Log Parsing: Unlike the log parsing phase in the error detection phase, the component's log messages should not be filtered out, in order to keep the spatial and temporal information. Moreover, in this phase, the system administrator are allowed to specify the time window in which the component logs are grouped together. Our model aims to assign sentiment polarity scores based on what each component logs about its health in a specific time.

2) Phase II: Component Sentiment Scores Assignment: In this phase, the system's components are associated with sentimental polarity scores using an assignment method similar to that used in the second phase of error detection presented previously. In particular, all negative features and positive features indicating error correction learned from the SGDLRSL model are used to assign each component a sentiment score for a specified time window. It is composed of three scores: positive, negative, and overall sentiment score (denoted as C_i**posScore**, C_i**negScore**, and C_i**sentiScore**, respectively), whose calculations are similar to those of the phase II in the error detection (see Formula (7), (8) and (9)).

Algorithm 3: Erroneous component identification in large-scale system based on sentiment scores

Input: Components $(C_i, ..., C_k)$, Unlabelled messages logs $(x_1, \cdots, x_n)$, Sentiment Lexicon Items(f_i, w_i), detection threshold φ , Start time $t_S \leftarrow t_s$, End time $t_E \leftarrow t_e$
Output: C_i**posScore**, C_i**negScore**, C_i**SentiScore**, C_i**State**
initial $\varphi \leftarrow \varphi$, C_i**Logs** $\leftarrow$ " "
Parsing messages logs $(x_1, \cdots, x_n)$
for *i = 1 to k* **do**
  for *j = 1 to n* **do**
    C_i**Logs** $\leftarrow$ **Concatenate** (x_j, x_{j+1}) ; $x_j \in C_i$ && $t_s \geq t \leq t_e$
  end
end
for *i = 1 to k* **do**
  C_i**posScore** $\leftarrow \frac{\sum_i w_i}{\sum_i f_i}$; $w_i > 0$
  C_i**negScore** $\leftarrow \frac{\sum_i w_i}{\sum_i f_i}$; $w_i < 0$
  C_i**SentiScore** $\leftarrow C_i$**posScore** $+ C_i$**negScore**
  if C_i***SentiScore*** $< \varphi$ **then**
    C_i**State** $\leftarrow$ Erroneous;
  else
    C_i**State** $\leftarrow$ non-Erroneous;
  end
end

3) Phase III: The Erroneous Component Identification: Once the system components are associated with sentiment polarity scores, the C_i**SentiScore** and detection threshold φ, are used to detect whether these components are erroneous or not. A component is classified as erroneous when it is attached with C_i**SentiScore** $< \varphi$ for consecutive time windows and as non-erroneous otherwise. This model can be plugged into each system's components to generate sentiment scores over the time windows customized by administrators time windows. It works as an assistant tool that continuously alerts the administrator of erroneous components within negative scores and positive scores as the components' issues are corrected.

VII. Experimental Evaluation

We perform the evaluation by automatically generating a sentiment lexicon from IBM Blue Gene systems' distinctive log message templates: BlueGene/L log templates(2007), BlueGene/P Intrepid log templates(2012), and BlueGene/Q MIRA log templates. Specifically, we carefully evaluate the viability of our error detection sentiment-based approach for three systems - Mira (year 2017&2020), Ranger, and Lonestar4, respectively. In the following, we first describe our evaluation indicators and then discuss the evaluation results.

A. Evaluation Metrics

Since our sentiment model not only detects erroneous logs but also accurately identifies non-erroneous logs, we utilize the weighted-average of (precision, recall, F1-score) and Matthew's correlation coefficient (MCC) to measure our sentiment model's performance. The precision, recall, F1-score are calculated for each category (faulty and non-faulty). So, for a given detection category i, we define: (i) True Positive$_i$

(TP_i) is the number of logs correctly detected as belonging to i category, (ii) False Positive$_i$ (FP_i) is the number of logs incorrectly detected as belonging to i category, (iii) False Negative$_i$ (FN_i) is the number of logs detected as not belonging to i category but, in fact, it does, and (iv) True Negative$_i$ (TN_i) is the number of logs correctly detected as not belonging to i category [9]. Based on TP_i, FP_i, FN_i, and TN_i, we then calculate precision$_i$ and recall$_i$ for each category i as follows:

$$Precision_i = TP_i/(TP_i + FP_i) \tag{10}$$

$$Recall_i = TP_i/(TP_i + FN_i) \tag{11}$$

and the F1-score$_i$, which is a balanced harmonic average of recall and precision, as shown below:

$$F1\text{-}score_i = 2 \times \frac{recall_i \times precision_i}{recall_i + precision_i} \tag{12}$$

Lastly, the overall three scores (precision, recall, F1) are weighted by each category's support and averaged[2]. The recall, precision and F1-scores are among the most popular statistical measures for a binary classification task. However, these measures can show overoptimistic results especially on imbalanced datasets. Matthew's correlation coefficient is a more reliable metric which produces a high score only if it obtains a good result in all the four confusion matrix categories (true positive, false positive, false negative and true negative), proportionally to both the size of positive and negative elements in the dataset. MCC is defined as [22]:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{13}$$

MCC returns a score between -1 to 1. A score of 1 represents a perfect detection. A score of 0 represents a random detection. A score of -1 represents total disagreement between detection and observation.

B. Evaluation of Error Detection

1) Learning Sentiment Lexicon for the IBM Blue Gene system: We use the SGDLR technique to automatically construct the sentiment lexicon based on the IBM Blue Gene system's RAS log message templates. We used $3k$ RAS log message templates as our dataset, whose labels are automatically inferred from the severity level field contained in a RAS log file. The RAS events are classified into two categories: faulty messages and non-faulty messages. The former include warnings, failures, and fatal levels, and the latter indicate informational messages to show the system software's progress or correction of errors.

We employ k-fold cross-validation with over-sampling methods to address the imbalance within RAS dataset. After the NLP text preprocessing phase, we adopt the Term Frequency-Inverse Document Frequency (TF–IDF) to transform terms (features) of RAS templates from text format to numerical vectors. The SGDLR algorithm with L1 regularization and default parameters are then employed to learn the sentiment lexicon items of the IBM Blue Gene systems by training RAS log template feature vector with their associated labels and obtaining the dense weights vector W (sentiment scores for lexicon items). Our model obtains around $\sim$932 discriminative features associated with their sentiment scores. Figure 1 shows $\sim$60 lexicon items associated with their sentiment intensity scores learned automatically. It is clearly observed that the learned sentiment lexicon of Blue Gene systems consists of the system developers' negative sentiments that show the systems' issues and the positive sentiments indicating that the system problems have been corrected or the system components work as expected. Moreover, the Blue Gene lexicon contains more negative sentiments than positive and neutral sentiments, demonstrating that the abnormal system issues are generally paid more attentions in the logs. The Blue Gene lexicon contains $\sim$664 negative sentiments, whereas there are $\sim$268 positive ones.

2) Mira Error Detection Performance: After the Blue Gene lexicon's items are generated from the previous phase, we evaluate its efficacy by detecting error messages using the entire year 2017 of the Mira RAS logs. The year-2017 RAS log contains 16.5 million messages. We first filter out the duplicated messages by using the open source toolkit - LogAider [15] based on the spatial or temporal correlations. The total number of messages is thus significantly reduced from $16,772,894$ to $2,380,211$. Then, we preprocess the log messages' content by the NLP techniques. After that, we classify all events based on their severity attributes into two categories - faulty and non-faulty - in order to obtain the ground truth for evaluating our error detection method. Our error detection sentiment-based approach takes the filtered $2,380,211$ messages each of which is assigned a sentimental polarity score, and detects the faulty messages based on comparing the associated scores with detection threshold φ. Figure 2 presents the evaluation results about error detection accuracy based on the RAS log of year 2017 with different lexicon absolute threshold μ values(i.e., the number of sentiment lexicon items) with detection threshold $\varphi = 0$. Experiments show that our machine learning-based sentiment lexicon achieved excellent error detection accuracy for the RAS log data of the year 2017, 99%, 99%, 99% of recall, precision, and f1-score, respectively, at lexicon absolute threshold μ = 0, 1, or 2, and detection threshold φ = 0.

In order to assess the effectiveness of our detection methodology under different conditions, we evaluate our sentiment lexicon approach with different values of μ and φ, which also aims at exploring the best threshold values with respect to high true positives and true negatives. We observe that a small change to the lexicon absolute threshold μ (i.e., the number of sentiment lexicon items) and detection threshold φ may have a significant impact on the detection result. In general, the detection accuracy increases as the former threshold value decreases, meaning that using fewer lexicon items affects the detection accuracy. Moreover, the same detection accuracy is

[2]The weighted-average method may result in an f-score that is not between recall and precision.

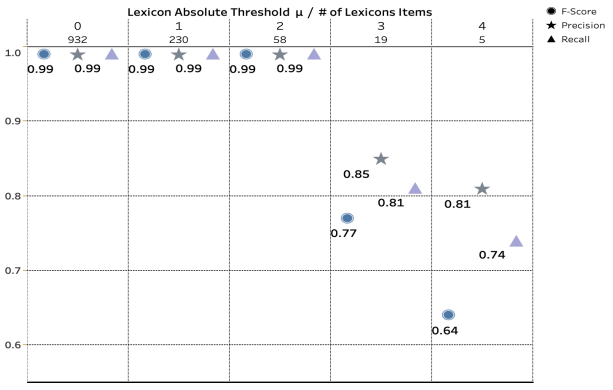

Fig. 2. Detection with different lexicon absolute threshold μ, with $\varphi = 0$

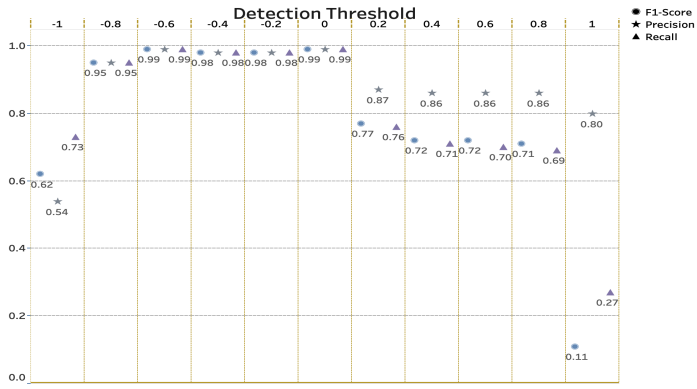

Fig. 3. Mira error detection performance with different detection threshold φ, with $\mu = 0$

reached when only $\sim$ 58 sentiment items are used (i.e., μ=2), and when using all the 932 features of our learned lexicon. This result verifies the fact that developers often use only a few sentiment words to log system issues and operations, as opposed to similar NLP sentiment analysis tasks that contain a high number of sentiment words.

We achieve a good error detection for the latter threshold value as the φ is chosen within the range between 0 and -0.8; this means the developers use high-intensity sentiments to log the system issues (see Figure 3). Moreover, we observe that our model's misclassifications occur due to two reasons: the first is the developer's classification of some logs with the incorrect severity. For example, the severity level is INFO for the '`nd receiver link error`' message and is ERROR for the '`correctable ecc error threshold`' message in the RAS log. However, our model correctly classifies the former as faulty since this is what is reflected through the feature **error**, and classifies the latter as non-faulty, since it shows the '`ecc error threshold`' is corrected. The second reason is that system developers tend to log some system events with unstructured text embedded with mixed negative and positive sentiments. For instance, the log message '`recoverable error message failed ecc parity error drill down error recoverable overable error cache parity error`' contains several negative and positive sentiments. Therefore, our model solves not only the problem of labeling the systems' logs with no severity levels but also fixes the misclassified severity levels within systems containing this feature.

3) Comparing Our Approach with ML/Deep Learning: New types of unlabelled logs are generated because the operators of large-scale systems (e.g., clusters, data centers) perpetually upgrade on the system components (software/hardware) and service to add new features, fix bugs, or enhance performance [23]. Furthermore, there are several large-scale systems (i.e., target systems) with non-labeled logs, such as Ranger and Lonestar4. Classifying the massive number of unlabeled logs (i.e., millions) of target systems manually into faulty and healthy is infeasible. Consequently, our approach can be used to detect potential faults/failures (classify logs to be faulty and non-faulty) automatically for the systems with non-labeled logs.

To demonstrate our approach's effectiveness and generality on cross-systems, we evaluate our sentiment lexicon's performance on logs of three large-scale systems to detect potential errors. We compare our solution with many state-of-the-art machine/deep learning classification techniques, including Random Forest (RT), Extreme Gradient Boosting (XGBoost), Multinomial Naive Bayes (Multinomial NB), K Nearest Neighbor (KNN), Long Short-Term Memory (LSTM), which have been commonly adopted by state-of-the-art approaches for error detection/prediction. There are multiple variants of LSTM showing better performance over standard LSTM and other ML techniques (e.g., CNN-LSTM [24] for disk failure prediction). For a comparison, we implemented a detection algorithm based on the standard LSTM based on our particular objective of detecting/classifying individual erroneous log messages. For fairness, we train the five machine/deep learning models by using the same data we employed to extract our sentiment lexicon, and then evaluate those models' performance versus our sentiment lexicon model's performance on logs of three large-scale systems: the first six months of 2020 Mira filtered RAS logs, and all distinctive log messages from Ranger and Lonestar4 (i.e., target systems).

As presented in Table I and Figure 4, the results reveal that using our lexicon achieves an average MCC score and f1-score of 91% and 96% respectively in error detection in the three large systems' log messages, whereas the best machine/deep learning model (LSTM) obtains only 67% and 84% respectively. The other models RF, KNN, XGBoost, and Multinomial NB achieve an average MCC score of 52%, 45%, 50%, and 44%, and an average f1-score of 68%, 66%, 64%, and 58%, respectively.

As mentioned above, new types of unlabelled message logs in such HPC systems were generated because of their upgraded or added components. It is non-trivial to manually classify a large number of new system log messages induced by the system upgrades or repairs. To cover this case, we evaluate our sentiment lexicon in classifying the latest RAS logs of IBM Blue Gene Mira 2020 (i.e., test dataset) and compare its performance with the other machine/deep learning

techniques. The IBM Blue Gene systems' distinctive log message templates: BlueGene/L (2007), BlueGene/P (2012), and BlueGene/Q log templates are combined as our training dataset to learn the lexicon and train other five machine/deep learning techniques. The results (see Figure 5) show that our sentiment lexicon achieves the best MCC and F1 score among the six methods, achieving high MCC and F1 scores of 98% and 99% respectively. All other techniques achieve satisfying accuracy because the training runs on old RAS message templates and tests on Mira 2020 RAS logs' messages. The deep learning technique, LSTM, also achieves a high MCC score of 96% and F1 scores of 98%, respectively, and all other ML models achieve MCC scores (91%) and f-scores (96%). In general, our sentiment lexicon successfully identified the developers' sentiments from old generations Blue Gene supercomputers to use them as an unsupervised approach classifying new types of logs Mira 2020 RAS logs.

As we stated before, our approach's primary goal is accurately identifying individual errors automatically for those HPC systems with a huge number of non-labeled logs. For example, the Ranger and Lonestar4 logged around 65 million and 12 million unlabelled messages in July 2011 and February 2013, respectively. To demonstrate the effectiveness of our solution in addressing this point, we utilize our sentiment lexicon features learned on labeled log messages from the source system (i.e., IBM Blue Gene) to detect errors in the target systems unlabelled log messages (i.e., Ranger and Lonestar4). As illustrated in Figure 6 and Figure 7, our lexicon achieves much higher scores in detecting error messages of Ranger and Lonestar4 logs than other solutions, even though they are different HPC systems developed by different companies with distinct logging methodologies. Our lexicon can detect the majority of errors accurately in Ranger and Lonestar4 unique logs. In absolute terms, the MCC scores reach up to 87% and 87%, the f-scores reach 94% and 95%, the recalls can get up to 94% and 95%, and the precisions achieve 94% and 95%, respectively. For other solutions including LSTM, RN, KNN, XGBoost, and Multinomial NB models, on Ranger Logs, the MCC scores are 55%, 31%, 19%, 26%, and 18%, respectively. The f1-scores are 76%, 51%, 46%, 44%, and 36%, respectively. On Lonestar4 Logs, those models get the MCC score only 50%, 35%, 25%, 32%, and 23%, respectively, and the f1-score only 79%, 57%, 55%, 51%, and 41%, respectively. The major limitations of detection accuracy on target systems for five machine/deep learning techniques are analyzed as follows: KNN does not work well because the log messages are imbalanced and contain many outliers, and LSTM, RF, and XGBoost are prone to overfitting, where Multinomial NB assumes the logs' features are independent, despite this not being the case.

These results demonstrate the reusable sentiment lexicon's benefits of deploying it as an unsupervised log analysis approach on different (target) systems. The key reason our solution outperforms other ML methods is that it successfully extracts developers' sentiment features hidden in labeled log messages from one source system (i.e., IBM Blue Gene) and transfers these features with their weights as lexicon items to detect errors in the target systems with unlabelled log messages (i.e., Ranger and Lonestar4. This verified the fact that developers of different systems really adopt similar sentiment features in their logging methodology. Thus, we can utilize a few system logs with severity levels to automatically extract their sentiment features and label the logs of other target systems with non-labeled logs. These promising results motivate us to collect more logs from different systems and generate a general sentiment lexicon using our technique to detect hardware and software issues in our future work.

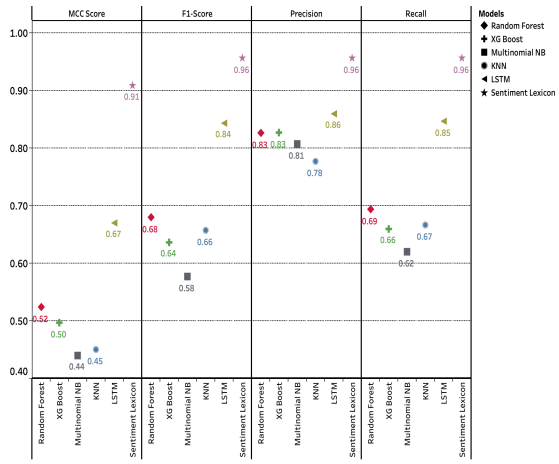

Fig. 4. The average of detection performance of our lexicon and ML models on all three systems

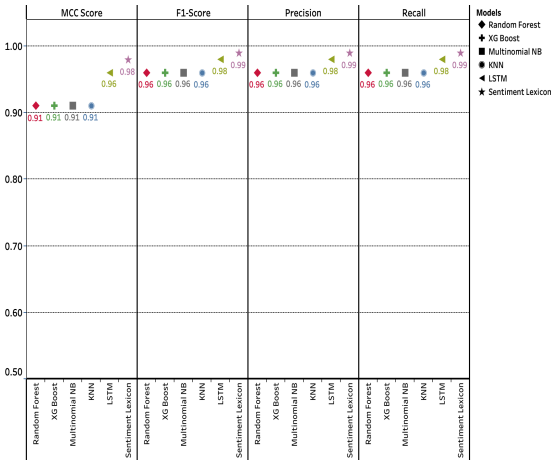

Fig. 5. Detection performance of our lexicon/ML models on 2020 Mira RAS logs

C. Evaluation of Erroneous Component Identification

We evaluate our approach of identifying erroneous components such as compute nodes (denoted Rxx-Mx-Nxx), I/O nodes (denoted Qxx-Ix-Jxx) and link modules (denoted Qxx-Ix-Uxx), based on Mira's RAS logs. We present the evaluation results based on one-day period (27-March-2017) due to space limit and massive components involved. We employ our sentiment lexicon learned in Section IV and detection

TABLE I
SCORES (RECALL, PRECISION, F1-SCORE, AND MCC) OF OUR LEXICON AND ML MODELS ON THREE SYSTEMS(MIRA, RANGER, AND LONESTAR 4)

	RAS Mira				Ranger				Lonestar4				Average of 3 Systems			
Technique	Recall	Precision	F1-score	MCC Score	Recall	Precision	F1-score	MCC Score	Recall	Precision	F1-score	MCC Score	Recall	Precision	F1-score	MCC Score
Random Forest	96%	96%	96%	91%	55%	74%	51%	31%	57%	78%	57%	35%	69%	83%	68%	52%
XGBoost	96%	96%	96%	91%	50%	74%	44%	26%	52%	78%	51%	32%	66%	83%	64%	50%
Multinomial NB	96%	96%	96%	91%	45%	71%	36%	18%	45%	75%	41%	23%	62%	81%	58%	44%
KNN	96%	96%	96%	91%	50%	66%	46%	19%	54%	71%	55%	25%	67%	78%	66%	45%
LSTM	98%	98%	98%	96%	76%	80%	76%	55%	80%	80%	79%	50%	85%	86%	84%	67%
Our Sol.	99%	99%	99%	98%	94%	94%	94%	87%	95%	95%	95%	87%	96%	96%	96%	91%

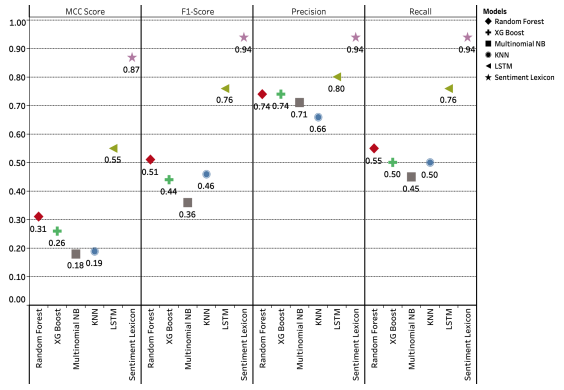

Fig. 6. Detection performance of our lexicon and ML models on Ranger logs

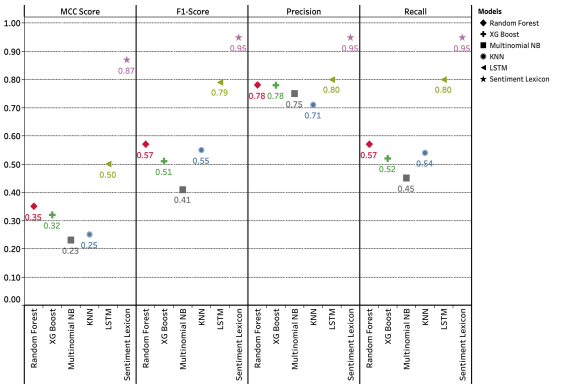

Fig. 7. Detection performance of our lexicon and ML models on Lonestar4.

sentiment score φ=0 to associate each component with 24 sentiment scores for the 24 hours, based on the hourly log messages regarding these components. Each score is composed of positive and negative scores, which are then summed up to produce the overall sentiment scores. Thus, each component is attached with a total of 72 sentiment scores (24 positives + 24 negatives + 24 overall scores).

Figure 8 demonstrates the states of some components in Mira. A gray color indicates 'working as expected by logging informational messages or logging nothing'; a red color indicates that the component is experiencing some abnormal events that affect its productivity; a green color indicates that the issues have been corrected, or silent errors have been fixed.

We observe that majority of Mira components experienced no issues (e.g., Q2H-I0-U02, Q2H-I0-U03, and Q2H-I0-U04). Negative sentiment scores were attached to a few components (e.g., Q2H-I0-J01, Q2H-I0-J02, Q2H-I5u-J00, Q2H-I5-J01, and R2B-M0-N00) for consecutive hours, which validates the high accuracy of our algorithm because each of them crashed due to one or two fatal events according to the

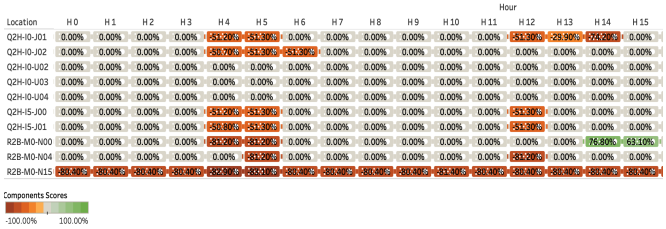

Fig. 8. Illustration of The Erroneous Component Identification on Mira

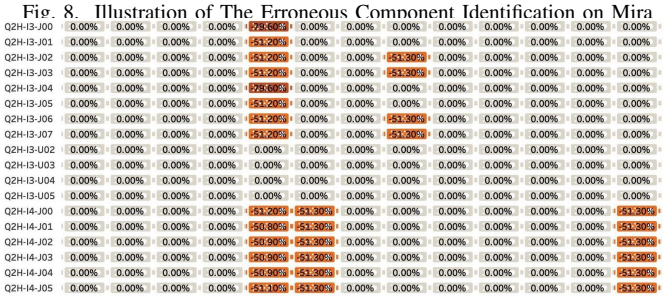

Fig. 9. Mira Erroneous Component Identification (Q2H-I3-J00∼Q2H-I4-J05)

RAS log. Moreover, some components (e.g., the node R2B-M0-N15) exhibit high negative scores for long consecutive hours since it was suffering from recurrent abnormal events. Some components such as R2B-M0-N00, exhibit positive sentiment scores in later hours (i.e., the 14th and 15th), as some correction events were triggered. Our technique can assist the Mira system's administrator to isolate these faulty components until the problems are fixed, since there are 152 fatal events occurring in that day, and our model highlights all the components that triggered them.

Furthermore, as shown in Figure 9 from sentiment scores attached to Q2H-I4-J00 ∼ Q2H-I4-J05, we first observe that components are associated with similar sentiment scores for a consecutive or recurrent time windows. This can be explained by the fact that components generate the same logs over those time windows to convey that they are still facing the same issues. Second, we also observe that neighboring components are assigned with similar sentiments scores within similar time windows, indicating that large-scale system components that exhibit similar behaviors generate similar logs. Thus, one important observation is that the issues encountered by similar components may result in the same sentiments, and thus similar sentiment scores, enabling our technique to be deployed by Mira's administrators to detect faulty components.

VIII. RELATED WORK

Log parsing is a crucial phase in log analysis which filters out and processes the large quantities of large-scale systems' message logs; different parsers have been developed (e.g. [15], [25]–[27]) and many have been evaluated on different

benchmark logs' datasets in [28]. Moreover, various tools and much research have been dedicated to diagnosing the root causes of failures such as [29]–[31].

Plenty of error detection/prediction methods are proposed. Research studies on error detection in HPC systems can be categorized into different aspects such as identification (i) of erroneous log entries, (ii) of failure-inducing erroneous execution sequence, and (ii) of identifying quantitative relationship among logs. For the first two categories, various machine/deep learning methods (e.g., Random Forest, (XGBoost), Multinomial Naive Bayes, K Nearest Neighbor, Long Short-Term Memory (LSTM)) were adopted by state-of-the-art studies for error detection in HPC systems. For instance, Meng et al. [9] identifies individual errors using partial labels based on PU learning and Support Vector Classifier techniques. Many other works (such as [32] , [33], and [2]) showed how the event logs can be transformed into a data set that is suitable for running some of above ML classification techniques. Also, the deep learning technique, LSTM, has been employed in DeepLog [7] and LogRobust [10] to identify erroneous log sequences. The works presented in [12], [13] aim to identify quantitative relationship among log messages.

Other works have been proposed in the domain of failure prediction. Lu et. al. [24] developed an effective hybrid model of the convolutional neural network and long short-term memory (CNN-LSTM) for disk failure prediction. Das et. al. [34] and Frank et. al. [35] developed novel techniques for predicting node failures in HPC systems. Das et. al. [36] developed a novel analysis approach to enable real-time prediction of node failures in HPC systems. Also, Nie et. al. [37] proposed various ML models for GPU error prediction in HPC systems. Very recent studies by Patel et. al. [38], Bin Nie et. al. [39] and Gupta et. al. [40] have provided valuable insights through long-term analysis and quantification of HPC jobs characteristics in large-scale systems [38], characterizing temperature, power and soft-error behaviour in data center systems [39] and long-term measurement and analysis of HPC system failures [40]. Differently to these works, our sentiment model focused on error detection in HPC systems.

Rao et. al. [41] developed a method to identify faults in large-scale distributed systems by filtering noisy error logs. Stearley et. al. [42] showed that a non-uniform distribution of log words across nodes is useful for error detection, as is the encoding of word position. Gainaru et. al. [43] modelled the normal and faulty behaviour of large-scale HPC systems. In [11], [44], the authors developed log entropy techniques to detect errors in HPC systems. All these tools are working on HPC systems and we complement them by constructing a sentiment lexicon automatically to classify non-labelled system logs for error detection in HPC systems.

Sentiment lexicon based analysis has been widely studied for years. Regarding the domain of sentiment lexicon being applied to human languages, Hutto et al. [21] built the VADER sentiment lexicon manually by multiple independent experts or crowdsourcing, which is particularly attuned to sentiments expressed in social media. By contrast, we construct a sentiment lexicon automatically to detect developers' sentiments hidden in system log messages. The Pointwise Mutual Information technique and SVM model learned on a distant supervised corpora were employed by [45] and [46], to build sentiment lexicons for English language. Also, logistic regression combined with other techniques are used to extract a lexicon for Portuguese language and stock market domains by [47] and [48] respectively. In contrast, we used SGDLR to construct a sentiment lexicon for error detection & erroneous component identification in large-scale systems.

There are also some existing works leveraging sentiment related techniques for log analysis. Allen et al. [49] is the first research group that utilized a sentiment lexicon through a pre-built library called IBM Watson API to analyze software logs and assign sentiment scores for log data. Yadwad et al. [50] applied machine learning and time series models (e.g., PCA, Naïve Bayes, logistic regression, and CNN) on combined data of the social tweets, mails and logs for service outage detection and predication. Based on the context and content attention model, Studiawan et al. [51] employed a deep learning technique to identify aspect terms and the corresponding sentiments to extract events of interest from log files in the forensic timeline. By comparison, our work is the first attempt in the domain of a large-scale system. Furthermore, our domain-specific sentiment lexicon items are extracted automatically with the use of a machine learning-based technique, since a feature's sentiment is affected by the domain in which it is used. Allen et al. [52] also proposed a method based on using at least keyword and synonym matching percentage analysis criteria to classify log messages' levels in applications code. In contrast, our model was designed to detect faulty components and errors of large-scale systems based on AI technique.

IX. Conclusion

In this paper, to the best of our knowledge, we are the first to propose a sentiment analysis based approach that can effectively build a reusable sentiment lexicon over the large-scale system logs, based on which errors could be detected and erroneous components could be identified. Using logs from three HPC systems from different vendors, our results show that our approach *significantly* outperforms other related state-of-the-art approaches.

Acknowledgments

This material was supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. We would like to thank the Argonne Leadership Computing Facility (ALCF) offering the MIRA system log, and also thank Eric Pershey - the principal software development specialist at the Argonne Leadership Computing Facility (ALCF) for answering our questions about Mira system and its RAS log. This work was also supported by the National Science Foundation under Grants CCF-1619253. We would also like to thank the Texas Advanced Computing Center at The University of Texas at Austin, USA for providing the Ranger and Lonestar4 system logs, and Security Lancaster at Lancaster University, UK for their support. We would also like to thank the five anonymous reviewers and our shepherd for their constructive feedback which helped improve the paper significantly.

REFERENCES

[1] E. Chuah, A. Jhumka, S. Alt, J. J. Villalobos, J. Fryman, W. Barth, and M. Parashar, "Using resource use data and system logs for HPC system error propagation and recovery diagnosis," in *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2019, pp. 458–467.

[2] N. Gurumdimma, A. Jhumka, M. Liakata, E. Chuah, and J. Browne, "Crude: Combining resource usage data and error logs for accurate error detection in large-scale distributed systems," in *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2016, pp. 51–60.

[3] A. Borghesi, A. Libri, L. Benini, and A. Bartolini, "Online anomaly detection in HPC systems," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2019, pp. 229–233.

[4] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

[5] S. Bursic, V. Cuculo, and A. D'Amelio, "Anomaly detection from log files using unsupervised deep learning," in *International Symposium on Formal Methods*. Springer, 2019, pp. 200–207.

[6] C. Xiao, J. Huang, and W. Wu, "Detecting anomalies in cluster system using hybrid deep learning model," in *International Symposium on Parallel Architectures, Algorithms and Programming*. Springer, 2019, pp. 393–404.

[7] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.

[8] A. Dwaraki, S. Kumary, and T. Wolf, "Automated event identification from system logs using natural language processing," in *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2020, pp. 209–215.

[9] W. Meng, Y. Liu, S. Zhang, F. Zaiter, Y. Zhang, Y. Huang, Z. Yu, Y. Zhang, L. Song, M. Zhang *et al.*, "Logclass: Anomalous log identification and classification with partial labels," *IEEE Transactions on Network and Service Management*, 2021.

[10] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[11] N. Gurumdimma, A. Jhumka, M. Liakata, E. Chuah, and J. Browne, "Towards detecting patterns in failure logs of large-scale distributed systems," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, 2015, pp. 1052–1061.

[12] S. Khatuya, N. Ganguly, J. Basak, M. Bharde, and B. Mitra, "Adele: Anomaly detection from event log empiricism," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2114–2122.

[13] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 102–111.

[14] D. Jurafsky and J. H. Martin, "Speech and language processing," *Chapter A: Hidden Markov Models (Draft of October 2, 2019). Retrieved October*, vol. 19, p. 2019, 2019.

[15] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello, "Logaider: A tool for mining potential correlations of HPC log events," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 442–451.

[16] S. Di, H. Guo, E. Pershey, M. Snir, and F. Cappello, "Characterizing and understanding HPC job failures over the 2k-day life of IBM bluegene/q system," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 473–484.

[17] S. Di, H. Guo, R. Gupta, E. R. Pershey, M. Snir, and F. Cappello, "Exploring properties and correlations of fatal events in a large-scale hpc system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 2, pp. 361–374, 2019.

[18] E. Chuah, A. Jhumka, J. C. Browne, B. Barth, and S. Narasimhamurthy, "Insights into the diagnosis of system failures from cluster message logs," in *2015 11th European Dependable Computing Conference (EDCC)*. IEEE, 2015, pp. 225–232.

[19] J. L. Hammond, T. Minyard, and J. Browne, "End-to-end framework for fault management for open source clusters: Ranger," in *Proceedings of the 2010 TeraGrid Conference*, 2010, pp. 1–6.

[20] S. Robertson, "Understanding inverse document frequency: on theoretical arguments for idf," *Journal of documentation*, 2004.

[21] C. J. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Eighth international AAAI conference on weblogs and social media*, 2014.

[22] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over f1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, 2020.

[23] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, Z. Zang, X. Jing, and M. Feng, "Funnel: Assessing software changes in web-based services," *IEEE Transactions on Services Computing*, vol. 11, no. 1, pp. 34–48, 2016.

[24] S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, and W. Shi, "Making disk failure predictions smarter!" in *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*, 2020, pp. 151–167.

[25] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 859–864.

[26] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 33–40.

[27] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 167–16 710.

[28] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 121–130.

[29] E. Chuah, A. Jhumka, S. Alt, D. Balouek-Thomert, J. C. Browne, and M. Parashar, "Towards comprehensive dependability-driven resource use and message log-analysis for HPC systems diagnosis," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 95–112, 2019.

[30] A. Das and F. Mueller, "Holistic root cause analysis of node failures in production HPC," *SC Poster Session*, 2018.

[31] E. Chuah, A. Jhumka, S. Alt, T. Damoulas, N. Gurumdimma, M.-C. Sawley, W. L. Barth, T. Minyard, and J. C. Browne, "Enabling dependability-driven resource use and message log-analysis for cluster system diagnosis," in *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. IEEE, 2017, pp. 317–327.

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

[33] B. Mohammed, I. Awan, H. Ugail, and M. Younas, "Failure prediction using machine learning in a virtualised HPC system and application," *Cluster Computing*, vol. 22, no. 2, pp. 471–485, 2019.

[34] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "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*, 2018, pp. 40–51.

[35] A. Frank, D. Yang, A. Brinkmann, M. Schulz, and T. Süss, "Reducing false node failure predictions in HPC," in *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 2019, pp. 323–332.

[36] A. Das, F. Mueller, and B. Rountree, "Aarohi: Making real-time node failure prediction feasible," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 1092–1101.

[37] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine learning models for GPU error prediction in a large scale HPC system," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 95–106.

[38] T. Patel, Z. Liu, R. Kettimuthu, P. Rich, W. Allcock, and D. Tiwari, "Job characteristics on large-scale systems: Long-term analysis, quantification, and implications," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, nov 2020, pp. 1–17.

[39] B. Nie, J. Xue, S. Gupta, C. Engelmann, E. Smirni, and D. Tiwari, "Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities," in *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2017, pp. 22–31.

[40] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "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*, 2017.

[41] X. Rao, H. Wang, D. Shi, Z. Chen, H. Cai, Q. Zhou, and T. Sun, "Identifying faults in large-scale distributed systems by filtering noisy error logs," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2011, pp. 140–145.

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

[43] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behaviour of large-scale HPC systems," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 1168–1179.

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

[45] S. M. Mohammad, S. Kiritchenko, and X. Zhu, "Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets," *arXiv preprint arXiv:1308.6242*, 2013.

[46] A. Severyn and A. Moschitti, "On the automatic learning of sentiment lexicons," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 1397–1402.

[47] K. Bhargava and R. Katarya, "An improved lexicon using logistic regression for sentiment analysis," in *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*. IEEE, 2017, pp. 332–337.

[48] R. A. Lage, "Sentiment lexicon generation with continuous polarities for portuguese using logistic regressions and semantic modifiers," Ph.D. dissertation, Universidade Federal do Rio de Janeiro, 2016.

[49] C. O. Allen, K. B. Haverlock, and M. D. Whitley, "Sentiment analysis of data logs," Jan. 3 2017, uS Patent 9,536,200.

[50] S. A. Yadwad and V. V. Kumari, "Predicting service outages using tweets," in *International Journal of Recent Technology and Engineering (IJRTE)*, 2020.

[51] H. Studiawan, F. Sohel, and C. Payne, "Sentiment analysis in a forensic timeline with deep learning," *IEEE Access*, vol. 8, pp. 60 664–60 675, 2020.

[52] C. O. Allen, A. R. Freed, S. N. Gerard, and D. B. Miller, "Applying consistent log levels to application log messages," Jun. 11 2019, uS Patent 10,318,405.