Converting Unstructured System Logs into Structured Event List for Anomaly Detection

Zongze Li, Matthew Davidson and Song Fu
Department of Computer Science and Engineering
University of North Texas
{Zongzeli2, MatthewDavidson}@my.unt.edu,
song.fu@unt.edu

Sean Blanchard and Michael Lang
HPC-DES Group, and Computer, Computational, and Statistical Sciences Division
Los Alamos National Laboratory
seanb@lanl.gov, mlang@lanl.gov

ABSTRACT

System logs provide invaluable resources for understanding system behavior and detecting anomalies on high performance computing (HPC) systems. As HPC systems continue to grow in both scale and complexity, the sheer volume of system logs and the complex interaction among system components make the traditional manual problem diagnosis and even automated line-by-line log analysis infeasible or ineffective. In this paper, we present a *System Log Event Block Detection* (SLEBD) framework that identifies groups of log messages that follow certain sequence but with variations, and explore these event blocks for event-based system behavior analysis and anomaly detection. Compared with the existing approaches that analyze system logs line by line, SLEBD is capable of characterizing system behavior and identifying intricate anomalies at a higher (i.e., event) level. We evaluate the performance of SLEBD by using syslogs collected from production supercomputers. Experimental results show that our framework and mechanisms can process streaming log messages, efficiently extract event blocks and effectively detect anomalies, which enables system administrators and monitoring tools to understand and process system events in real time. Additionally, we use the identified event blocks and explore deep learning algorithms to model and classify event sequences.

KEYWORDS

HPC systems, system reliability, behavior analysis, anomaly detection.

1 INTRODUCTION

High performance computing (HPC) systems continue growing in both scale and complexity. For example, the Trinity supercomputer at Los Alamos National Laboratory has more than 19,000 heterogeneous (i.e., Intel Haswell and Intel Knights Landing) compute nodes [1]. Titan at Oak Ridge National Laboratory has 18,688 compute nodes equipped with AMD Opteron processors and NVIDIA Tesla GPUs [2]. These large-scale, heterogeneous systems generate tens of millions of log messages every day. In addition to the sheer volume, both the format and the content of these log messages vary dramatically, depending on system architecture, hardware configuration, management software, and type of applications. Effective log analysis for understanding system behavior and identifying system and component anomalies and failures is highly challenging.

Log analysis for system behavior characterization has continuously been an important research topic. A good number of methods and tools have been proposed and developed. For example, in a study of 200,000 Splunk queries [4], queries were clustered into several categories and transformation sequences were detected to analyze connection among queries. In [5], console, netwatch, consumer and apsched logs were stored in a MySQL database and accessed through a web interface for application profiling. A three-layer filtering method was used to compress log data without losing important information [3]. System logs from Blue Gene/P and five supercomputers were analyzed in [6, 7] and several important observations about failure characteristics were reported. Time coalescence techniques were also used to analyze supercomputer logs [8]. Correlation among log messages was quantified by an apriori-like algorithm and correlation graphs were used to perform prediction [9].

Existing approaches use line-by-line log analysis. Although they can discover distribution and precedence relation among log messages, they are not effective for discovering subtle behavior patterns and their transitions, and thus may overlook some critical anomalies. However, log messages are not isolated from each other. An event of a component or the system may produce multiple messages. Analysis at the event level can provide a richer semantics of system behaviors and thus enable to detect more subtle anomalies that the traditional line-by-line analysis methods cannot find.

We use *event block* (EB) to refer to log messages that belong to a component or system event. Event block based analysis is not trivial. A fundamental challenge is how to identify log messages that belong to the same event and thus group them into an EB. The high concurrency in HPC systems causes messages from different events and even from different nodes to overlap with each other. It is common to see messages of an event are scattered into multiple pieces by messages from several other events. Moreover, the overlap does not follow a fixed pattern. Additionally, some messages may appear, disappear, or have variable contents in different instances of an event. Without detailed execution context, such as information of application workload, system processes, scheduling method, and device status, it is difficult to accurately identify event blocks. This context information itself, however, is not easy to obtain in large-scale production HPC systems.

Despite these challenges, the advantages of event block

ARES 2018, August 27–30, 2018, Hamburg, Germany
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6448-5/18/08…$15.00
https://doi.org/10.1145/3230833.3230855

based log analysis are intriguing. By converting the original, lengthy, and unstructured messages in system logs into a compact and structured list of EBs, the complexity of log analysis can be significantly reduced and the results are more interpretable and easier to understand. By working at the level of EB, we can find patterns of events, the evolution of system behavior, and the interaction among different system components, which is very difficult to achieve by using the traditional message-level analysis. Variation among instances of an event is also an indicator of possible anomalies. Thus, a larger set of anomalies can be detected.

In this paper, we present SLEBD, a System Log Event Block Detection framework, which can identify event blocks accurately and automatically. SLEBD explores the probability of messages occurring together in a flexible period of time and leverages the law of total probability to consolidate messages that occur together even with variations into Event Block (EB) patterns from system logs. The consolidated EB patterns are stored in an Event Block Database, called EBD. SLEBD is adaptive and noise-resilient, i.e., the more instances of messages are observed, the better consolidated event blocks will be generated. SLEBD is capable of processing streaming log messages and analyzing system events and behavior in real time.

We have implemented a prototype of SLEBD. We have evaluated this SLEBD prototype by using syslogs collected from the Mutrino supercomputer at Sandia National Laboratories. Experimental results show that SLEBD can convert hundreds of thousands of raw log messages to a concise and structured behavior report composed of a much smaller number of event blocks. SLEBD is also effective for anomaly detection and problem diagnosis. Several new types of system-level anomalies are found in our experiments. Based on the identified event blocks, we define an event sequence classification problem and leverage the deep learning method, more specifically, Recurrent Neural Network, to analyze event sequences and characterize system behavior.

SLEBD has the following attractive features.

1) It can convert millions of unstructured log messages into concise and structured event block lists, which facilitates system monitoring and behavior analysis.

2) It generates an event block database (EBD) from the original system logs. System administrators can use EBD to process message streams in real time.

3) It updates EBD by continuously analyzing multiple time-periods of log files. Thus, EBD evolves as the monitored system changes.

4) It analyzes system and component events to identify anomalies and facilitate fault diagnosis.

The remainder of this paper is organized as follows. Section 2 discusses the related research. Section 3 presents example messages from real-world system logs and describes the log preprocessing procedure. Section 4 describes SLEBD in detail. Section 5 presents the use of SLEBD for anomaly detection. Section 6 explains how SLEBD handles multiple log files. Experimental results are presented in Section 7 with discussion in Section 8. Section 9 concludes this paper with remarks on future research.

2 RELATED WORKS

Several tools are available for analyzing log messages. For example, Baler [10] uses a list of keywords to scan log messages and counts the number of times that each keyword appears. SLCT [11] clusters log lines based on the frequency of keyword-position occurrences. HELO [12] splits log messages in a training file into clusters. The splitting position in a message line is determined by those words that appear most frequently.

To pre-process system logs, Zheng et al. [13] used the occurrence probability to model the causal relation among single-line messages. Their method, however, cannot be used to determine causally related log messages that belong to an event with a high confidence.

There are also studies that explore system logs for anomaly detection. For example, Lou et al. [14] calculated the occurrence probability among log lines in a time window to analyze their dependency. Fu et al. [15] used a finite state automaton (FSA) to model execution paths of a system and applied these FSAs to detect anomalies from log messages. Xu et al. [16] grouped single-line patterns based on alphabetic words in line groups. They applied the principal component analysis method to the extracted feature vectors to detect anomalies. Baseman et al. presented a textual-numeric data ground graph [17] to analyze log messages and used Infomap [18] to extract clusters from subgraphs of the ground graph.

These approaches focus on individual log messages and analyzing distributions or precedence relation among log lines. By exploring event blocks, our proposed approach transforms unstructured log messages into structured event sequences, which enables us to identify event patterns and more subtle system and component anomalies.

There are also methods designed to model relationship among log lines, such as the frequent sequential pattern mining methods, including FreeSpan [19], SPADE [20], Aprioriall [21] and time coalescence [22]. They approximate message occurrence sequences. However, they are not effective for determining which log lines belong to the same event.

3 SYSTEM LOG PROCESSING

The format of system logs varies as system architecture, operation system, runtime, management tools, and applications can be different. For example, Mutrino, which is a Cray XC40 system at Sandia National Laboratories, has its syslogs in the following format:

```
2015-02-13T13:16:11.865060-06:00 c0-0c0s0n1 trying chooser simple
```

Syslog format on the Mutrino supercomputer (SNL)

where "2015-02-13T13:16:11.865060-06:00" is a timestamp, "c0-0c0s0n1" is a node ID, and "trying chooser simple" is the message body. Syslogs collected from Thunder, a Cray XT supercomputer at Lawrence Livermore National Laboratory, have a different format as follows:

```
Jun 6 11:20:18 nid00003 kernel: EXT3 FS on sde1, internal journal
```

Syslog format on the Thunder supercomputer (LLNL)

where "Jun 6 11:20:18" is the timestamp, "nid00003" is a node ID, and "kernel: EXT3 FS on sde1, internal journal" is the message body. Moreover, Mutrino is a newer system than Thunder with new types of hardware and software components.

Thus, its syslogs contain many messages which are not seen in Thunder's syslogs.

We aim to design our event block analysis framework to be generic, being capable of processing and analyzing system logs in different formats. Our tool requires limited user involvement. Users only provide message syntax information, indicating the structure of log messages. Then, SLEBD parses messages and extracts message elements by using the syntax information.

Log Preprocessing

System logs collected from production HPC systems are complex. Messages from all compute nodes and service nodes are mixed together, which makes it difficult to accurately identify event blocks.

The *log preprocessing process* filters and separates those mixed messages. We separate streaming messages or messages from large mixed logs into multiple files based on node IDs, that is one file for each node. These node-wise log files are first analyzed individually to extract *line patterns* (Section 4.1.1) and *event blocks* (Section 4.1.3) on each node, and then combined together to identify event blocks across multiple or all nodes (Section 4.1.4). Additionally, the preprocessing process formats messages and removes incomplete messages, which facilitates the learning of line patterns for event block extraction.

Event blocks contain multiple lines of messages, but may only appear once in a period of time on a single node. They cannot be captured if only one node's log file is analyzed. To solve this problem, we define an inspection time window and produce a directory to store log messages from all nodes in a window. Figure 1 shows the structure of the preprocessed log sets. By extracting and analyzing event blocks across time periods, we can confirm or remove event blocks that do not appear frequently in only one period. It also helps us understand the change of system events and thus behavior over time.

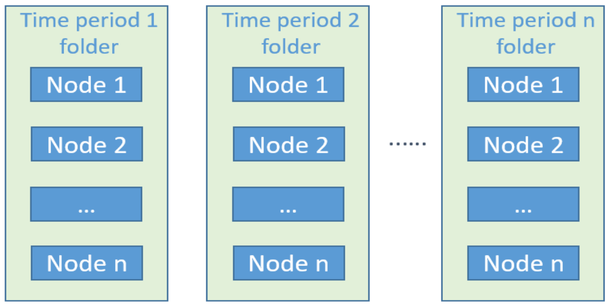

Figure 1: Structure of the preprocessed log sets

4 EVENT BLOCK DATABASE AND EVENT BLOCK EXTRACTION

Using the preprocessed log files as input, our System Log Event Block Detection (SLEBD) framework performs 1) event block database (EBD) generation, 2) event block extraction, and 3) anomaly detection. SLEBD uses a *Line Pattern Hash Table* (LPHT) and the EBD to conduct event block extraction. Figure 2 shows the major components and work flow of SLEBD.

4.1 Event Block Database (EBD) Generation

4.1.1 Single Line Patterns

Syslog listens for messages on /dev/log. Log messages of an event may be generated by multiple threads or devices. They have certain line patterns in syslog, but may contain variations. The following are two examples of messages taken from Mutrino's syslog.

```
2015-02-13T13:19:42.494097-06:00 c0-0c2s1n1 ACPI: PCI Root Bridge [PCI0] (domain 0000 [bus 00-fe])
```

Mutrino syslog message: example 1

```
2015-02-13T13:16:45.462890-06:00 c0-0c0s0n1 ACPI: PCI Root Bridge [UNC0] (domain 0000 [bus ff])
```

Mutrino syslog message: example 2

Even though the preceding messages are produced from two different nodes, they are generated by the same process and thus have similar message structure.

SLEBD processes messages in syslogs and generates a *line pattern* for each message. Each line pattern has a unique identifier in the form of "[LinePattern_$num]". A line pattern is created by alphabetic words in the message and their corresponding positions. Numbers are treated as variables, and not included in line patterns. The line pattern of the preceding messages is as follows. Although the two messages are different, their line patterns are the same.

[[0, ACPI:], [1, PCI], [2, Root], [3, Bridge], [5, (domain), [6, [bus]]

A single line pattern

Two messages are called *k%* similar, if at least *k*% of words and their positions in their line patterns are the same. Two messages have the same pattern if they are *k*% similar and *k* is greater than a defined threshold. Otherwise, we say their line patterns are different. Variables in a message have number(s) and may contain letter(s), such as username and node ID. It is not necessary to have an exact match for two variables as they are less critical than alphabetic words which are the skeleton of a message. For example, in the preceding log messages, the bus channel on Node "c0-0c0s0n1" is named "ff", while the bus channel on Node "c0-0c2s1n1" is named "00-fe". Both are variables and their mismatch does not affect the similarity of their corresponding line patterns. Variables are analyzed later for anomaly detection.

These generated line patterns are stored in a *Line Pattern Hash Table* (LPHT). For a new log message, SLEBD searches for the most similar (i.e., *k*-similar with high *k* value) pattern(s) from LPHT. If no similar pattern is found, a new line pattern is added to LPHT.

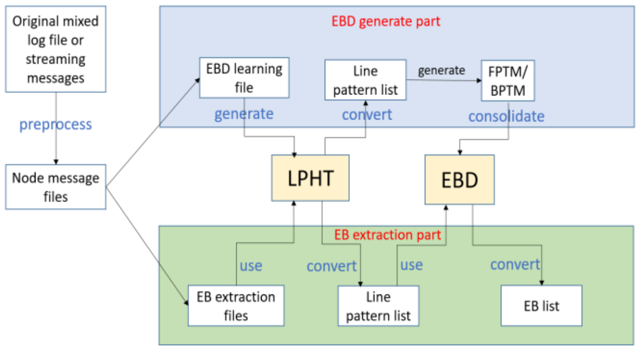

Figure 2: Major components and work flow of SLEBD

4.1.2 Line Pattern Forward/Backward Transition Matrix (FPTM/BPTM)

By using single line patterns stored in LPHT, we convert the original, unstructured log messages into a *line pattern list*. SLEBD produces a line pattern list for each node in a system. These line pattern lists for different nodes are then co-analyzed to determine how often every pattern occurs and what adjacent line patterns happen before (called *backward patterns*) and after (called *forward patterns*) the line pattern in question. A *Forward Probability Transition Matrix* (FPTM) and a *Backward Probability Transition Matrix* (BPTM) are generated. For example, assume the line pattern lists of a two-node system are as follows. A to F denote line patterns in the form presented in Section 4.1.1.

Table 1: An example of line patterns in a two-node systen

Node ID	Line pattern list
Node 1	A, D, E, F
Node 2	A, E, F

Table 2: FPTM and BPTM generated from line pattern lists

FPTM						BPTM					
	A	D	E	F	Finish		A	D	E	F	Start
A		0.5	0.5			A					1
D			1			D	1				
E				1		E	0.5	0.5			
F					1	F			1		

The Table 1's corresponding FPTM and BPTM are presented in Table 2.

In the example of Table 1, we can see LinePattern_A occurred twice and have two forward patterns, LinePattern D & E. So in the FPTM, PF (A → E) = 50% and PF (A → D) are equal to 50%. And LinePattern_D have only one backward pattern, LinePattern_A. So in the BPTM, PB (A → D) = 1. From this view we can find that BPTM is not a transpose of FPTM.

4.1.3 Event Block Generation and Consolidation

Based on the Bayes' theorem [23], the occurrence probability of an event depends on that of events that are related to this event. The Forward Probability Transition Matrix (FPTM) and the Backward Probability Transition Matrix (BPTM) described in the preceding section, contain information of forward and backward line patterns, respectively. We apply the Bayes' theorem by using the two matrices to find the most relevant line patterns in the forward and backward ranges of a line pattern in order to identify an event block.

To this end, we leverage an extended form of the Bayes' theorem, which is called the Law of Total Probability [24], expressed as:

$$\mathbf{P\ (E|A) = \Sigma\ P\ (E\ |\ A \cap\ B_i) * P\ (B_i\ |\ A)} \tag{1}$$

We use it to calculate the probability of a line pattern *E* happening in the forward range of a line pattern *A*. We use "PF(A→E)" to denote this forward probability and "PB(A→E)" to denote the backward probability. Thus, we have

$$\mathbf{PF\ (A \rightarrow E) = \Sigma\ PF\ (A \rightarrow B_i) * PF\ (B_i \rightarrow E)} \tag{2}$$

where B_i represents line patterns that immediately follow the line pattern *A*. We use the FPTM matrix to find those line patterns.

If both PF(A→E) and PB(A→E) are greater than a *Threshold of Occurring Together* (TOT), we say line patterns *A* and *E* occur together with a high confidence. The threshold TOT can be set to its initial value provided by system administrators or dynamically updates its value based on the occurrence frequency of the line pattern *A*. More details are presented in Section 4.1.4.

Those line pattern pairs that are identified to occur together are stored in an *Occurring Together pattern Pair List* (OTPL). In OTPL, each line pattern *p* has two lists, one of which contains the line patterns that happen before (i.e., backward) *p*. The other list stores those line patterns that appear after (i.e., forward) *p*. Table 3 shows an OTPL generated based on the Forward and Backward Probability Transition Matrices .

Table 3: Occurring Together pattern Pair List (OTPL)

Pattern ID	Backward Pattern List	Forward Pattern List
A	{}	E
D	{}	{}
E	A	F
F	E	{}

Event block consolidation: SLEBD starts with those line patterns whose backward pattern lists are empty but forward pattern lists are not empty in OTPL in the EB consolidation process. A first-in-first-out (FIFO) list and one *Temporary Event Block pattern List* (TEBL) are used. First, SLEBD puts a line pattern to the end of the FIFO list. Then, SLEBD

1. Selects the first line pattern p_{top} in the FIFO list and stores it in TEBL.
2. If the forward pattern list of p_{top} is not empty, puts those line patterns which are in the forward pattern list of p_{top} but not in the FIFO list yet, to the end of the list.
3. Adds p_{top} to TEBL if p_{top} does not exist in it.

This process continues until the FIFO list becomes empty.

In the preceding example, there is only one line pattern (i.e., *A*) whose backward pattern list is empty but forward pattern list is not empty. SLEBD performs event block consolidation as follows.

1. Put *A* into the FIFO list. Thus, FIFO = {*A*} and TEBL = {}.
2. Get *A* from the list and store *A* in TEBL. Put *A*'s forward list patterns, *E* into FIFO. Now FIFO = {*E*} and TEBL = {*A*}.
3. Get *E* from the list and store *E* in TEBL. Put *E*'s forward list patterns, *F* into the list. Then, FIFO = {*F*} and TEBL = {*A*, *E*}.
4. Get *F* from the list and store *F* in TEBL. As *F*'s forward pattern list is empty, the consolidation process stops. So, FIFO = {}, TEBL = {*A*, *E*, *F*}

When the EB consolidation process stops, SLEBD considers the line patterns in TEBL as one event block and assigns an EB ID in the form of "[Block_$num]" to the new EB pattern.

All EB patterns are stored in the Event Block Database (EBD). The line pattern whose backward list is empty but forward list is not empty, e.g., *A* in the example, is marked as the start line pattern. The line pattern whose forward list is empty but backward list is not empty, e.g., *F*, is marked as the end line pattern. SLEBD updates the information of those line patterns in the *Line Pattern Hash Table* (LPHT) to indicate to which EB patterns they belong.

4.1.4 Threshold of Occurring Together (TOT) and confidence interval

We use the *Threshold of Occurring Together* (TOT) in Section 4.1.3. SLEBD can dynamically update TOT at runtime.

The reason that TOT is used is that when consolidating EBs, SLEBD should tolerate noise when building EBD from syslogs. In SLEBD, two line patterns do not have to occur together all the time in order to decide they are one pair of occurring-together patterns. More often a line pattern appears, the higher accuracy in which the occurring-together patterns can be identified. This mitigates the influence from noise or incomplete information in the learning process. As a result, if a line pattern occurs less frequently, its TOT is lower. The TOT increases as a line pattern occurs more often and the conditions for identifying its occurring-together patterns are more accurately identified.

To realize this design, we use a confidence interval generated by the square root of the occurrence count to dynamically update TOT. The less frequently that SLEBD sees a line pattern, the wider the confidence interval for TOT. Table 4 provides some sample values of TOT for different occurrence counts.

Table 4: Threshold of Occurring Together (TOT)

Occurrence count	Square root	Occurrence count confidence interval	TOT
50	7	43 – 57	86%
100	10	90 – 110	90%
200	14	186 – 214	93%

This confidence interval enables SLEBD to identify line patterns that should not be considered as occurring together. For example, if LinePattern_2's occurrence count does not satisfy LinePattern_1's confidence interval, SLEBD does not calculate the occurring-together probability of LinePattern_1 and LinePattern_2. As the occurrence count and occurrence count confidence interval corresponding feature can avoid computing obviously unrelated Line Patterns' relationship, it also helps reduce computation workload.

The value of TOT should not be manually assigned very small, as this may cause unrelated line patterns to be considered as occurring-together.

Enhancing Event Block Database from using multiple time windows of system logs: The number of log messages affects the accuracy of event block identification. The longer period a log covers, the higher probability that more event blocks and their complete patterns can be included. In our study, the following problem are found in the logs.

1. Some unrelated line patterns always occur together in a short period of time.
2. A log contains incomplete, mingled, or noise messages.
3. New types of messages appear due to hardware upgrades, runtime updates, installation of new monitoring tools, change of configuration, and etc.

SLEBD selects log files of a period of time to generate the EBD. However even a carefully selected log set may still suffer from the preceding problems. To address this issue, SLEBD keeps updating EBD as more messages are processed and analyzed.

After the log files in Time Period one is analyzed, any line pattern in LinePattern_1's forward range and the pattern's occurrence count can fit LinePattern_1's confidence interval. SLEBD updates their relationship. As log files in Time Period two are analyzed, SLEBD computes LinePattern_1's occurrence count from these two periods and generates a new confidence interval for LinePattern_1. LinePattern_2 is another line pattern in LinePattern_1's forward range. If either of the following two conditions is satisfied, SLEBD adds LinePattern_1 into the Reconsidered Pattern List (RPL):

1. LinePattern_1 and LinePattern_2 used to be one pair of occurring-together line patterns. However, LinePattern_2's occurrence count cannot fit in LinePattern_1's recent confidence interval.
2. LinePattern_2's previous occurrence count cannot fit in LinePattern_1's previous confidence interval. However, LinePattern_2's latest occurrence count can fit in LinePattern_1's recent confidence interval. SLEBD reconsiders LinePattern_1's relationship with LinePattern_2.

Additionally, any new line pattern detected from a different inspection time window are also added to RPL.

SLEBD uses the latest FPTM/BPTM generated from multiple inspection windows to calculate the occurring-together probability of the line patterns in RPL. If one line pattern's relationship with other patterns changes, SLEBD updates its forward/backward pattern lists in *Occurring Together pattern Pair List* (OTPL).

After the line patterns in RPL are re-analyzed, SLEBD performs EBD consolidation using the updated the Occurring Together pattern Pair List . As a result,

1. Previously consolidated unrelated line patterns can be split in a correct way.
2. Previously separated but related line patterns can be consolidated.
3. Newly found line patterns can be added to *Line Pattern Hash Table* (LPHT) and updated EB patterns are added to EBD.

Cutoff for reconsideration: SLEBD terminates the reconsidering/update process according to certain configurations, which is called the *reconsideration cutoff*. Such cutoff conditions can be: 1) The relation of line patterns is stable for a specified period of time, for example, 30 days; and 2) A line pattern's occurrence count reaches a certain number, for example, 500 times.

4.2 Event Block Extraction

SLEBD uses *Event Block Database* (EBD) to extract event blocks from system logs. The EB extraction process uses the line pattern list as input. It explores LPHT and EBD. For each line pattern, it searches for a possible match pattern in LPHT. Then SLEBD identifies Event Blocks from the line pattern list by matching with the patterns stored in EBD.

SLEBD uses a stack of line patterns to record the number of log lines that are received and processed for each node. If a line pattern is the start line of an EB pattern, then the block ID and log line number are pushed into the stack. If the line pattern is the end of a block pattern, then the block ID stored on the top of the stack is popped out and compared with the line pattern's block ID. If they match, the block ID and the log line numbers from the start to the end are written to an extracted event block list.

The reason of using a stack to identify event blocks is that EBs can be nested, that is one EB happens inside another EB. If such nesting is detected, the outer block is kept and the inner block is removed from the output event block list.

When finding that the end line pattern does not match with the block at the top of the stack, or all line patterns have been processed but some start line patterns remain in the stack, SLEBD detects an anomaly. The details of anomaly detection are presented in the next Section.

5 ANOMALY DETECTION USING EVENT BLOCKS

SLEBD models and characterizes millions of messages from system logs by using only a limited number of event blocks. System operators can concentrate on analyzing the distribution and dynamics of event blocks, which facilitates their understanding of system behavior and identifying anomalous behaviors. As one of the implications, we present the benefit of using event blocks for anomaly detection.

5.1 Incomplete Event Block Anomaly

Every line pattern that is included in an EB pattern has a tag to identify which EB pattern it belongs to. When extracting the EB list, if SLEBD detects any line pattern happens individually instead of in an EB, this EB is not completely generated. Then SLEBD considers this case as an anomaly.

The cause of such type of anomaly can be that procedure is not executed completely, or there exists errors in the generation of log messages, or some other system or kernel level error happens.

We analyze the anomalies detected by SLEBD from the Mutrino logs. Here is an example.

One block (i.e., Block_10) is consisted of two line patterns: LinePattern_255 and LinePattern_256:

```
[LP_255]: hub 1-0:1.0: USB hub found
[LP_256]: hub 1-0:1.0: 2 ports detected
```

Block_10 Example

In the 300-day Mutrino logs, SLEBD finds Block_10 appears 3,665 times and detects 39 times of anomalies are related with Block_10. A typical anomaly is as follows.

```
[LP_735]: /scsi/) failed: hub 1-0:1.0: USB hub found
[LP_736]: Not a directory
[LP_256]: hub 1-0:1.0: 2 ports detected
```

Sequence [LP_735, LP_736, LP_256] Example

This sequence occurs 5 times. LinePattern_256 supposed to belong to Block_10 but appears individually. SLEBD reports an anomaly. By searching the event extraction record, we have detected that LinePattern_735 occurs 6 times and LinePattern_736 occurs 5 times. Thus, part of LinePattern_735 is not generated completely. LinePattern_735 contains part of LinePattern_255. We then check the related line patterns, and find LinePattern_502 is as follows.

```
[LP_502]: udevd-work[5561]: rename(/dev/scsi/.tmp-b8:128, /dev/scsi/) failed: Not a directory
```

LinePattern_502 Example

LinePattern_502 occurs 587 times and it always happens close to Block_10. Here is the symptom of the anomaly.

1. LinePattern_736 contains part of LinePattern_502 (**Not a directory**)
2. LinePattern_735 contains part of LinePattern_502 (**"/scsi/) failed:"**) and LinePattern_255 (**"hub 1-0:1.0: USB hub found"**)

Compared to LinePattern_502 and Block_10, LinePattern_735 and LinePattern_736 occur less often and their appearance is erroneous. SLEBD detects the anomaly.

This type of mixed-message anomaly happens on many nodes in the 300-day system log that we test. 18 event block patterns have the mixed-message anomaly and 197 such anomalies are detected by SLEBD. A possible explanation of these anomalies is that messages coming from different components on a node cause a race condition for syslogd to record them correctly.

Additionally, SLEBD detects system behavior that is different from the normal. For example:

```
[LP_558]: waiting for mysql to accept connections
[LP_559]: Checking SDB version
[LP_560]: SDB version up-to-date (4.10.14)
[LP_561]: Initializing SDB Table
```

Block_95 Example

```
[LP_558]: waiting for mysql to accept connections
[LP_1155]: Not initializing SDB on backup sdbnode (65)
```

Sequence [LP_558, LP_1155] Example

In this case, messages in Block_95 do not have the mixed-message anomaly. Block_95 occurs 61 times and the sequence [LP_558, LP_1155] appears five times. SLEBD detects that LinePattern_558 appears individually instead of being part of Block_95. Block_95 indicates SDB was successfully initialized. However, the sequence [LP_558, LP_1155] indicates the SDB initialization failed.

5.2 Anomalous Length of Event Block

SLEBD also uses the length of event blocks to identify anomaly detection. For example, Block_72 contains two messages. In the dataset that we test, Block_72 occurs 167 times and it contains two messages in 166 times. In one case, however, one time one Block_72 has 852 messages, i.e., from Line #1514 to Line #2365. This is an anomaly as the length of this event far exceeds the average length. Certain error happened during that event causing more messages to be included in the anomalous event block.

5.3 Deep Learning-based Event Sequence Classification

The execution of an HPC system can be modeled as sequences of various events. The pattern of the sequences and the criticalness of events vary from system to system. SLEBD extracts event blocks from raw log messages and generates event block lists. With the anomaly detection functionality provided by SLEBD, we know if an event is normal or not. Thus, we produce an event sequence classification problem in order to detect future anomalies. We explore deep learning methods to tackle this problem due to its strong classification capability.

We use Recurrent Neural Networks [26] on the event block lists generated from syslogs. For example, we collect sequences that contain 500 event blocks prior to a normal event (say Block_10) and anomalous events related to the block. We have

936 sequences in the normal cluster and 39 sequences in the anomalous cluster. The two clusters are biased, as the normal set is much larger than the anomalous set. We use the SMOTE method [27] to balance the two sets. We then divide the produced dataset into a training set (which contains 2/3 of the event block sequences) and a test set (which contains the rest 1/3 of event block sequences). We build a four-layer Recurrent Neural Network which has 150 nodes by using a deep learning library Keras [28] on the training set. The output is a Long Short-Term Memory (LSTM) [29] model. The LSTM model is evaluated by using the test set. We use three-fold cross validation and the classification performance is promising, i.e., the classification accuracy for the normal cluster and the anomalous cluster is 97.12% and 87.50%, respectively.

5.4 Semantics/Topic Analysis of Event Blocks

To understand the meaning of each detected event block, we explore the latent Dirichlet allocation (LDA) [30] method to extract topics from event blocks. LDA is a topic model that employs a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is modeled as an infinite mixture over an underlying set of topic probabilities. The topic probabilities provide an explicit representation of syslog. We apply LDA to the detected event blocks from the Mutrino syslogs. Table 5 presents the extracted 20 topics and their corresponding keyword lists. The keywords provide us indications about what parts or devices of the system each topic is about. For example, Topic 6 is about disk and Topic 13 is about network, while Topic 8 is about NFS. Table 6 shows the event blocks that are grouped based on these 20 topics. Blocks that belong to the same topic can be correlated as they are related to the similar subsystem. Some topics (such as Topic 15 with 22 blocks) is more popular than others (such as Topic 4 with only 2 blocks).

Table 5: Topics and Keywords Extracted by Using LDA on Mutrino Logs

Topic ID	**Keywords**	**Topic ID**	**Keywords**
Topic 1	to cray opt using	Topic 11	dev udevd scsi starting
Topic 2	igb ahci irq pci	Topic 12	init runlevel been control
Topic 3	found in no not	Topic 13	network down up daemon
Topic 4	table entries hash socket	Topic 14	acpi disabled type pci
Topic 5	bios mode data timer	Topic 15	the boot system volume
Topic 6	root disk by id	Topic 16	registered usb new ehci_hcd
Topic 7	intel intelr ver physical	Topic 17	using for device at
Topic 8	down nfs start cache	Topic 18	sd sdb sdg sdd
Topic 9	version kernel driver gemini	Topic 19	host csa job lbcd
Topic10	boot on started at	Topic 20	dev from file etc

Table 6: Event Belong to Different Topics

Topic ID	**Event Blocks**
Topic 1	261,34,245,240,33,344,237,236,29,227,343,130,145
Topic 2	129,154,270,244,170,175,162,253,106,159,200
Topic 3	143,153,137,263,127,221,126,125,103,114,139
Topic 4	177,116
Topic 5	132,84,345,309,198,9,158,110,217,6,176
Topic 6	7,181,264,52,161,259,308,152,160,258,131
….	….
Topic 15	63,73,68,265,70,83,76,75,64,256,25,56,65,74,124,61,257,71,67,275,291,62
Topic 20	209,216,271

Changes of topics and their block sets are possible indicators of updates of system configuration and changes of system workload, which is useful for anomaly detection.

6 CASE STUDY

In this section, we use an example to illustrate how SLEBD works.

6.1 Test logs

We use a shutdown event as an example. After log preprocessing, we generate two test log files part of which is shown in Table 7.

Table 7: Parts of two test log files

Line	Log File 1	Line	Log File 2
1	Got signal 15, exiting	1	Got signal 15, exiting
2	Exiting...	2	Exiting...
3	exiting	3	exiting
4	caught signal 15, shutting down normally.	4	caught signal 15, shutting down normally.
5	exiting (success)	5	exiting (success)
6	Disconnected from system bus	6	Disconnected from system bus
7	WARNING: no sender#012	7	WARNING: no sender#012
8	Demoting known real-time threads.	8	Demoting known real-time threads.
9	Demoted 0 threads.	9	Successfully demoted thread 2106 of process 2095.
10	rpcbindterminating on signal. Restart with "rpcbind-w"	10	Successfully demoted thread 2105 of process 2095.
11	The audit daemon is exiting.	11	Successfully demoted thread 2095 of process 2095.
		12	Demoted 3 threads.
		13	rpcbindterminating on signal. Restart with "rpcbind-w"
		14	The audit daemon is exiting.

6.2 Generating line patterns and producing line pattern list

SLEBD processes the log files and generates the Line Pattern Hash Table (LPHT). For a single message, such as "Got signal 15, exiting", the corresponding single line pattern is:

[[Got, 0], [signal, 1], [exiting, 3]]

SLEBD converts the input log files to a line pattern list:

Table 8: Line pattern Lists

Line pattern list 1	Line pattern list 2
LinePattern_1 [1, 1]	LinePattern_1 [1, 1]
LinePattern_2 [2, 2]	LinePattern_2 [2, 2]
…	…
LinePattern_8 [8, 8]	LinePattern_8 [8, 8]
LinePattern_9 [9, 9]	LinePattern_12 [9, 11]
LinePattern_10 [10, 10]	LinePattern_9 [12, 12]
LinePattern_11 [11, 11]	LinePattern_10 [13, 13]
	LinePattern_11 [14, 14]

6.3 Generating FPTM from line pattern list

For illustration, we use LinePattern_8 to LinePattern_12 in the line pattern lists. SLEBD generates the Forward Probability Transition Matrix as shown in Table 9.

Table 9: FPTM/BPTM for LP_8 to LP_12 in the line pattern lists

	FPTM					BPTM				
	LP_8	LP_9	LP_10	LP_11	LP_12	LP_8	LP_9	LP_10	LP_11	LP_12
LP_8		0.5			0.5					
LP_9			1			0.5				0.5
LP_10				1			1			
LP_11								1		
LP_12		1				1				

Then, SLEBD calculates the probability that LP_8 and LP_9 occur together. From the FPTM,

PF (LP_8 → LP_9) = PF (LP_8 → LP_9) + PF (LP_8 → LP_12) * PF (LP_12 → LP_9) = 50% + 50% * 100% = 100%.

From the BPTM,

PB (LP_8 → LP_9) = PB (LP_8 → LP_9) + PB (LP_9 → LP_12) * PB (LP_8 → LP_12) = 50% + 50% * 100% = 100%.

As both PF (LP_8 → LP_9) and PB (LP_8 → LP_9) are 100%, SLEBD outputs that LP_8 and LP_9 are a pair of occurring-together messages.

6.4 Occurring-Together pattern Pair List (OTPL)

By analyzing line patterns' occurring-together relation, SLEBD produces OTPL as shown in Table 10.

Table 10: Generated Occurring-Together pattern Pair List

Message name	Backward pattern list	Forward pattern list
LP_1	{}	LP_2, LP_3, LP_4
…	…	…
LP_8	LP_5, LP_6, LP_7	LP_9, LP_10, LP_11
LP_9	LP_6, LP_7, LP_8	LP_10, LP_11
LP_10	LP_7, LP_8, LP_9	LP_11
LP_11	LP_8, LP_9, LP_10	{}
LP_12	{}	{}

Using the OTPL, SLEBD consolidates an EB pattern:

Block_1: {LP_1, LP_2 …… LP_11}

where LinePattern_1 is the block's start and LinePattern_11 is the end. Line patterns from LP_1 to LP_11 are in one event block. The log files are converted to an event block list as shown in Table 11.

Table 11: Event Block Lists

Event Block List 1	Event Block List 2
Block_1 [1, 11]	Block_1 [1, 14]

7 PERFORMANCE EVALUATION

We have implemented a prototype software of SLEBD and conducted experiments using Mutrino's system logs. The Mutrino system [25], sited at Sandia National Laboratories, is a Cray XC40 system with 118 nodes. It is a test environment of the Trinity production supercomputer [1], the 7th most powerful supercomputer in the world. The dataset that we use contains all syslogs collected from 2/11/2015 to 6/13/2016 on Mutrino.

We use console logs in our experiments. In total, there are 553 console logs in the dataset. Because each console log is composed of mixed messages from all 118 nodes, we select the first 300 days as our test set and assign two days as the size of the inspection time window. After log preprocessing, 150 directories are generated and each one contains 118 node-wise log files.

We conduct our experiments in two steps: 1) performing Event Block Database (EBD) generation on part of the preprocessed logs; 2) performing EB extraction on the rest of logs using the generated EBD. After an EB list is extracted, SLEBD detects anomalies where system behavior is different from the normal.

7.1 Experiment Settings and SLEBD Configuration

Table 12 lists the configuration of the servers where we run SLEBD.

Table 12: Experiment Settings

CPU model	Intel Xeon X3460 2.8GHz
Core count	8
OS	Linux CentOS 7. 3.1611
Memory	32GB

In our experiment, a "valid line" refers to a log message which has at least one alphabetic word. Thus, a line pattern can be created for a valid line.

Two line patterns are considered to be similar if their similarity ratio is above the threshold. In our experiment we set the threshold *k* as 67%, which means that if two line patterns, for example each has three words, have at least two words and their positions in the patterns the same, then we can consider these two line patterns have the same line pattern.

We describe the forward block detection range in Section 4.1.3. In our experiment, we set this range to 4. The reason for this setting is explained in Section 7.4.

In our experiments, when SLEBD analyzes one time period directory, all line patterns whose transition probabilities in the Forward and Backward Probability Transition Matrices have a dramatic change compared with previous records are added to the Reconsidered Pattern List (RPL).

7.2 Generated EBD from Mutrino logs

The experimental results from the Mutrino logs are presented in Table 13.

Table 13: Mutrino experiment results

Number of log messages	2,817,016
Number of invalid lines (removed by preprocessing)	67,478
Number of line patterns	2,884
Number of line patterns included in event blocks	608
Number of event block patterns	189
Number of line patterns occurring only once	409
Number of line patterns occurring less than 5 times	949

From Table 13 we can see that SLEBD detects in total 2,884 types of line patterns from the Mutrino log set. 608 of them are included in the event block (EB) patterns. 409 of them occur only once in all test files and they are not consolidated into EB patterns. 949 of them appear less than 5 times.

SLEBD detects 189 EB patterns in total. 2,817,016 lines of the original logs are converted to 1,690,391 events, including both event blocks and single line messages.

7.3 Reconsidered line patterns and relationship changed line patterns

Figure 3 shows the number of patterns which are reconsidered and the relation is updated. We use all logs of the first 110 days in the experiment. The blue bars present the size of RPL, that is the number of line patterns that need to be reconsidered. The orange bars inside blue bars show the number of line patterns whose relationship with other patterns are updated.

From Figure 3 we can see:

1. For the first 10 days SLEBD produces 1103 line patterns and 526 patterns' relation. As the EBD is not established, SLEBD considers all newly detected line patterns.
2. Bars 2 to 6 show the size of RPL is around 500. The number of relationship updates becomes less. The reason is that the relation between line patterns becomes more stable.
3. For Bars 7 and 8, the number of reconsidered patterns and relationship updates suddenly increases. We check the original system logs and find that from Day 78 to Day 82 the system had a software update, which made the system produced many new line patterns which had not been seen before. SLEBD effectively reconsiders line patterns and updates relationship focusing on newly detected line patterns.

This result shows SLEBD is adaptive and it only reconsiders a subset of line patterns. For example, on Date 51 (i.e., the first day of Bar 6) SLEBD detects 1500 types of line patterns. As the RPL is generated, SLEBD only reconsiders 463 line patterns, which saves a considerable amount of computation. Meanwhile, we can see only 14 line patterns updated for Bar 6. If we carefully select a cutoff threshold, we can reduce the number of reconsidered patterns.

7.4 Distribution of EB pattern length

Figure 4 shows the distribution of the size of multi-line EBs extracted from Mutrino syslogs. From the figure, we can see 75% of the EBs have a length of 4 or less. However, some EBs have more than 60 line patterns. The largest event block contains 269 line patterns.

After consulting with the system administrator, we find the largest event block that has 269 line patterns corresponds to a system boot sequence which is important for detecting possible system failures and shutdown.

In our experiments, the forward block detecting range is set to 4. From Figure 4, we can see most EBs have a length less than 5. This means 4-line forward range is enough for most line patterns to find other occurring-together line patterns.

7.5 Execution time results

The time to generate EBD from the 300-day system logs is 97 minutes, while extracting event blocks from the same logs only takes 5 minutes.

As we present before, analysis of a log file directory follows these steps:

1. Analyzing all files and generating temporary line pattern list for each file. SLEBD generates a line pattern for every single line and compares its similarity with other patterns in the Line Pattern Hash Table (LPHT).
2. Generating the Forward/Backward Probability Transition Matrix (FPTM/BPTM) from the produced line pattern lists.
3. Performing occurring-together probability calculation on FPTM and BPTM and generating the Occurring Together pattern Pair List (OPTL). SLEBD uses OPTL to consolidate EB patterns and stores EB patterns in EBD.

We run SLEBD on one inspection-window directory and find that Step 3 consumes the most amount of time and Step 1 takes the second longest time to finish. Step 3 involves pair-wise probability computation and Step 1 performs a large number of comparison and similarity ratio computation.

As Steps 1 and 2 on a new directory do not affect Step 3 on the previous directory, we can optimize our program by pipelining the three steps.

Additionally, we can distribute FPTM and BPTM among multiple computers, parallelize the probability computation, and then merge the results into one OTPL to perform EB consolidation. This parallel computing method will reduce the time for EBD generation.

8 DISCUSSION

SLEBD can identify events from massive messages in system logs. Using the generated the Event Block Database (EBD), we can perform real-time log processing and system monitoring. However, there are still several issues with SLEBD.

We need to find a message that appears at least twice in order to include it in an event block pattern. More appearances can increase the confidence for the generated pattern. However, it also causes SLEBD to pay less attention to those messages that appear only once. Although those messages do not appear often, they could be useful for detecting anomalous behaviors. SLEBD can only identify these single-appearance messages. Existing methods that are good at line-by-line analysis can be leveraged to process those messages.

SLEBD calculates the probability of line patterns

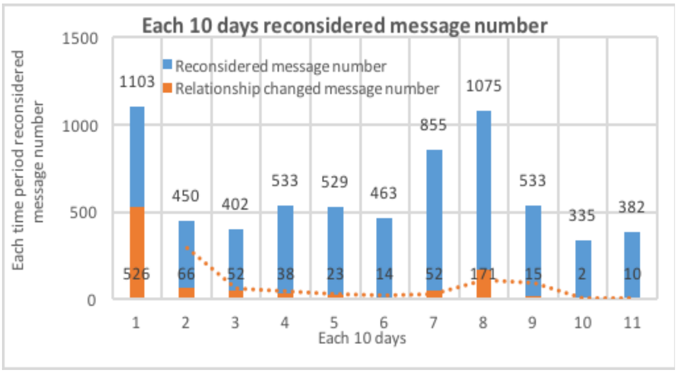

Figure 3: Reconsidered line patterns and updated pattern relation.

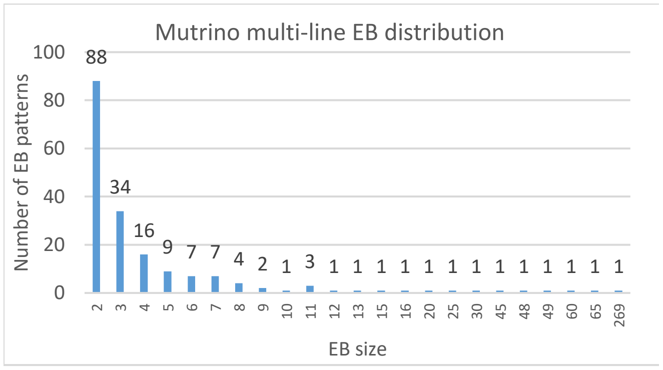

Figure 4: Mutrino System: Multi-line EB size Distribution

occurring together, which is resource-hungry and time-consuming. We limit the forward range and use the confidence interval of occurrence count to reduce the computation overhead. However, the program still needs to run for a long time to analyze a large log set and generate EBD. We will explore some hardware solutions to accelerate the computation.

Some existing software tools, such as Apache Kafka, Spark and Cassandra, can also store and process system monitoring data. An integration of SLEBD with these tools will enhance their capability of processing system data at a higher level with richer semantic information.

In this work, we study system logs to extract higher-level events and perform event-based anomaly detection. Our discussion focuses on system reliability and availability. We note that anomaly detection is also widely used in security applications, such as intrusion detection and fraud detection. Our SLEBD framework can also be used to detect abnormal events caused by security attacks. By grouping individual activities and messages into event blocks, we can have a better understanding of the connections among these events and the intension of malicious behavior, and thus detect more subtle attacks.

9 CONCLUSIONS

In this paper, we present the System Log Event Block Detection (SLEBD) framework. SLEBD generate event blocks which capture the major behavior of an HPC system and facilitate anomaly detection. By transforming the original mixed messages into clear event lists, system administrators can save their time and efforts to focus on analyzing high-level event distribution and relation with richer semantic information.

To further improve SLEBD, we are currently developing a machine learning based approach to study system behaviors using event block lists. We plan to make the Event Block Database generation process more efficient and automatic. We also consider using FPGAs and ASICs to accelerate log analysis and event block extraction.

ACKNOWLEDGEMENTS

The Mutrino Dataset is released by the Holistic Measurement Driven Resilience (HMDR) project to the research community in support of Extreme-Scale HPC Resilience Research. HMDR is funded by the U.S. Department of Energy Office of Advanced Scientific Computing Research. This work was funded in part by an NSF grant CCF-1563750 and an LANL research grant. This paper has been assigned an LANL identifier LA-UR-17-24869.

REFERENCES

[1] K Pedretti，S, Olivier, G Shipman, W Shu, K Ferreira. ” Exploring MPI Application Performance Under Power Capping on the Cray XC40 Platform, Proc. of EuroMPI , 2015.

[2] https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/

[3] Li Yu, Ziming Zheng, Zhiling Lan. "Filtering Log Data: Finding the needles in the Haystack", Proc. of IEEE/IFIP DSN, 2012.

[4] S. Alspaugh, Archana Ganapathi, Marti A. Hearst, Randy Katz. “Analyzing Log Analysis: An Empirical Study of User Log Mining”, Proc. of USENIX LISA, 2014.

[5] Raghul Gunasekaran, Sarp Oral, David Dillow, Byung Park, Galen Shipman, Al Geist. ”Real-Time System Log Monitoring/Analytics Framework”, Proc. of CUG, 2011.

[6] Ziming Zheng, Li Yu, Wei Tang, Zhiling Lan, R. Gupta, N. Desai, S. Coghlan, D. buettner. "Co-Analysis of RAS Log and Job Log on Blue Gene/P", Proc. of IEEE IPDPS, 2011.

[7] Adam Oliner, Jon Stearley. “What Supercomputers Say: A Study of Five System Logs”, Proc. of IEEE/IFIP DSN, 2007.

[8] Catello Di Martino, Marcello Cinque, Domenico Cotroneo. “Assessing time coalescence techniques for the analysis of supercomputer logs”, Proc. of IEEE/IFIP DSN, 2012.

[9] Xiaoyu Fu, Rui Ren, Jianfeng Zhan, Wei Zhou, Zhen Jia, Gang Lu. “LogMaster: Mining Event Correlations in Logs of Large-Scale Cluster Systems”, IEEE SRDS, 2012.

[10] Taerat, N., Brandt, J., Gentile, A. et al. “Baler: deterministic, lossless log message clustering tool.” Computer Science Research Development, 26: 285, 2011.

[11] R Vaarandi. “A Data Clustering Algorithm for Mining Patterns From Event Logs”, Proc. of IEEE Workshop on IP Operations and Management, 2003.

[12] Ana Gainaru, Franck Cappello, Stefan Trausan-Matu, Bill Kramer. “Event Log Mining Tool for Large Scale HPC Systems”, Proc. of Euro-Par, 2011.

[13] Ziming Zheng, Zhiling Lan, Byung H. Park, Al Geist. “System Log Pre-processing to Improve Failure Prediction”, Proc. of IEEE/IFIP DSN, 2009.

[14] JianGuang Lou, Qiang Fu, Yi Wang, Jiang Li. “Mining Dependency in Distributed Systems through Unstructured Logs Analysis”, ACM SIGOPS Operating Systems Review, Volume 44 Issue 1, 2010.

[15] Qiang Fu, JianGuang Lou, Yi Wang, Jiang Li. “Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis”, Proc. of ICDM, 2009.

[16] Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael Jordan. “Mining Console Logs for Large-Scale System Problem Detection” Proc. of ACM SOSP, 2009.

[17] Elisabeth Baseman, Sean Blanchard, Zongze Li, Song Fu. “Relational Synthesis of Text and Numeric Data for Anomaly Detection on Computing System Logs”, Proc. of IEEE ICMLA, 2016.

[18] M. Rosvall and C. T. Bergstrom, “Maps of random walks on complex networks reveal community structure,” National Academy of Sciences, Volume 105, Issue 4, pp. 1118–1123, 2008.

[19] Jiawei Han, Jian Pei, Behzad Mortazavi-asl, Qiming Chen, Umeshwar Dayal, Mei-Chun Hsu. “FreeSpan: frequent pattern-projected sequential pattern mining”. Proc. of ACM KDD, 2000.

[20] Mohanmmed J. Zaki. “SPADE: An Efficient Algorithm for Mining Frequent Sequences”, Machine Learning, Volume 42, pp. 31–60, 2001.

[21] Ming Hu, Guannan Zheng, Hongmei Wang. “Improvement and research on Aprioriall algorithm of sequential patterns mining”, Proc. of International Conference on Industrial and Intelligent Information, 2013.

[22] Catello Di Martino, Marcello Cinque, Domenico Cotroneo. “Assessing time coalescence techniques for the analysis of supercomputer logs”, Proc. of IEEE/IFIP DSN, 2012.

[23] Zdzislaw Pawlak. “Rough sets, decision algorithm and Bayes’ theorem”, European Journal of Operational Research, 136(1): 181-189, 2002.

[24] David Niju. “Law of Total Probability”, 2008 .

[25] Mutrino, Sandia National Labs, http://hpc.sandia.gov/aces/

[26] Mikolov T, Kombrink S, Burget L, et al. Extensions of recurrent neural network language model. Proc. of IEEE ICASSP, 2011.

[27] Jeatrakul P., Wong K.W., Fung C.C. (2010) Classification of Imbalanced Data by Combining the Complementary Neural Network and SMOTE Algorithm. *Neural Information Processing, Models and Applications*. Springer, 2010.

[28] https://keras.io/.

[29] Hochreiter, S. & Schmidhuber, J. Long short-term memory. Neural Comput. 9, 1735–1780 (1997).

[30] David M. Blei, Andrwe Y. Ng, Michael I. Jordan, “Latent Dirichlet Allocation”, Machine Learning Research, Volume 3, pp. 993-1022, 2003.